University of Tübingen
Computer Vision Lecture Notes

Computer Vision Lecture Students

Summer Term 2021

1 First lecture

1.1 Introduction

1.1.1 Artificial Intelligence

We introduce what computer vision is and also have a look at some of the challenges involved in this problem. To put computer vision into context, let us consider it as part of what it is typically considered: An aspect of the general field of artificial intelligence.

> *An attempt will be made to find how to make machines use language, from abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.* (John McCarthy, pioneer for artificial intelligence)

The subareas of artificial intelligence include:

- Machine Learning
- Computer Vision
- Computer graphics
- Natural language processing
- Robotics & Control
- Art, Industry 4.0, Education

And all of these areas are interlinked as we will see in this class. Here is an example. We want to build a robotic system. That robotic system must perceive the world, the environment and it must then accordingly update its internal representation of the world in order to make appropriate decisions to minimize some objective.

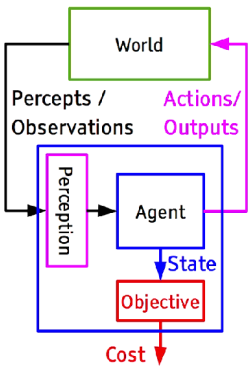


Figure 1: **Robot System**

1.1.2 Computer Vision

Computer vision can be very generally described as here in this picture as converting light into meaning. And meaning could be geometry, reconstructing the scene in terms of geometry or in terms of semantics, identifying certain aspects that we see. What computer vision receives as input is sampling of the light pattern.

On the top left is a light emitter. He emits light rays in all directions. And those light directions hit a surface point and then they bounce away to multiple directions from that surface. Then they hit another surface point. And at some point, they hit the observer, the eye or the digital camera. So all of these light rays that pass through the scene, are then sampled in a particular point, in a particular time, in a particular direction at a particular wave length and this forms the image that we then have to interpret.

This is one of the most famous books about vision and computer vision by David Marr and he writes about what it means to see the plain man's answer and Aristotle's two would be to know what is where by looking. This is how computer vision is often defined. We want a computer to know what is where by looking. Or in other words to discover from images what is present in the world, where things are, what actions are taking place to predict and anticipate events in the world. So this is more the robotics aspect where we want perform actions maybe in the world and in order to do actions we need to anticipate what's happening around us we need to make also predictions about the future. There are numerous computer vision applications and many of these have already commercialized today. And there are applications in robotics, in medicine, in structuring photo collections or building 3d models, in self-driving in mobile devices and accessibility just to name a few.

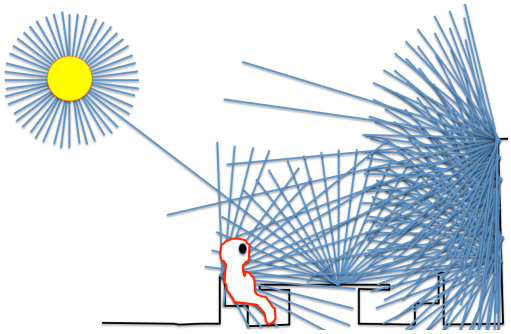


Figure 2: **Light field**

1.1.3 Computer Vision vs. biological vision

And as already mentioned the field of computer vision has connections to many neighboring fields. So for instance it has connections to neuro science and biological vision. In particular many computer vision algorithms take inspiration from biological vision and biological vision serves as a motivator for developing computer vision algorithms. In fact over 50 percent of the processing in the human brain is dedicated to visual information signifying the importance and the difficulty of that task that we have to solve if we want to replicate biological vision with a computer.

1.1.4 Computer Vision vs. graphics

Computer vision is also related to computer graphics. In computer graphics, given the 3D scene composed of objects, materials, certain semantic pose and motion we are concerned with rendering a photorealistic 2D image of that scene. In computer vision we are trying to solve the inverse problem. That is why sometimes we are also talking about inverse graphics. We are just seeing this 2d image where a lot of this information from the original 3d scene has been lost through this projection in process and the only thing that is accessible to us now is a raw pixel matrix. And we want to recover these latent vectors, these underlying factors that are present in the scene. Therefore vision is also an ill-posed process, an imposed inverse problem where a lot of this information that was originally present in the scene has been lost and has to be recovered. For example many 3D scenes yield exactly the same 2D image. Think about occlusion. If an object is occluded no matter where it's located the projection into the image looks the same. Or if an image is overexposed. No matter what the scene geometry is, it all looks the same. So these are just two extreme examples. But because this is pervasive in computer vision for almost all computer computer vision problems we need to integrate additional constraints additional knowledge about the world in order to solve this ill post inverse problem. And this is where a lot of research in computer vision goes into. These constraints can come from engineer constraints, it can come from first principles about the world and it can come from large data sets where this knowledge has been acquired from.

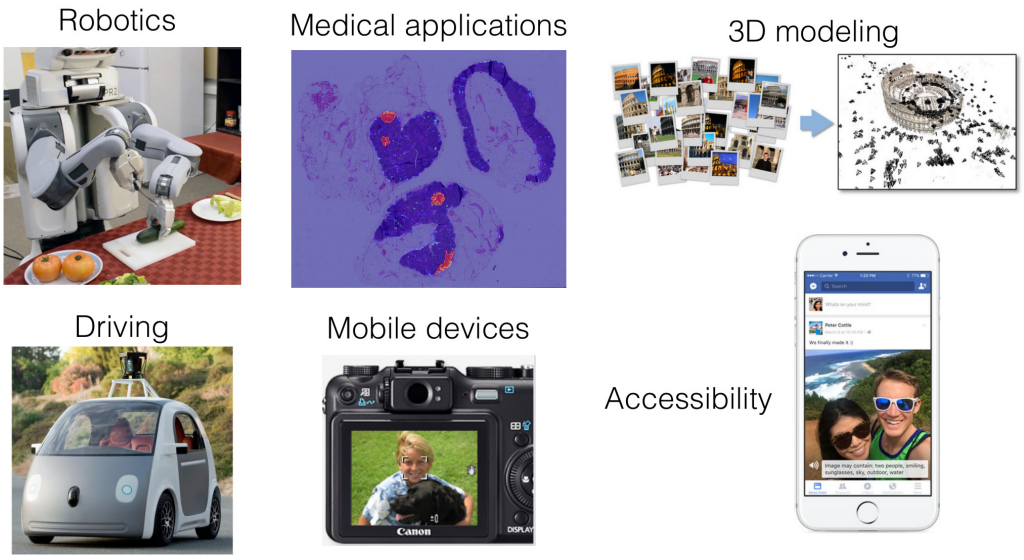


Figure 3: **Applications**

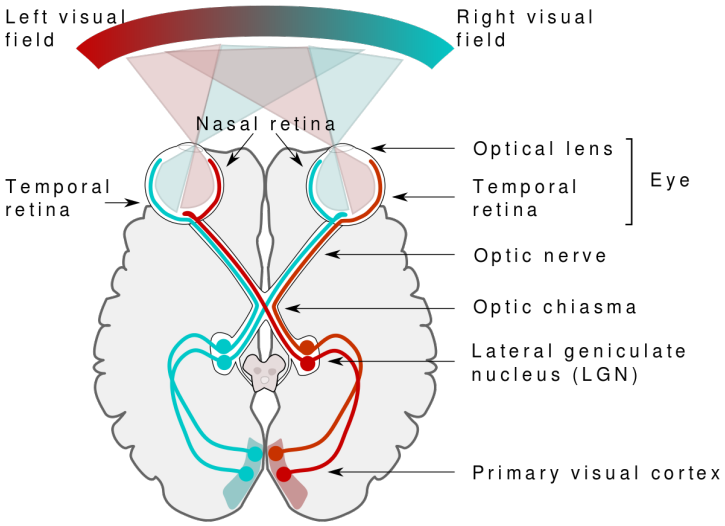


Figure 4: **Biological vision**

1.1.5 Computer Vision vs. image processing

Computer vision is also related to image processing. Image processing is concerned with low level manipulation of images like color adjustment, edge detection, denoising, image warping and image morphing. This all happens in 2D. As we will see in the history part computer vision has deviated from image processing primarily in the early stages, because it was concerned with getting a more holistic scene representation and scene understanding. In particular recovering the underlying 3D structure of the world, that is not present in this 2D representations that image processing is dealing with.

Image processing is not a focus of this class. We are focusing on the computer vision and on the more holistic aspects of this problem. There is also a relationship to machine learning. A lot of machine learning tools that have been developed are used everywhere in computer vision today. They have made computer vision successful, also commercially successful. They have provided the basis for computer vision to work in real-world applications. In particular in the last 10 or 15 years. On the other side in the other direction, computer vision often provides a very good motivation, good examples, good use cases for machine learning. It has helped tremendously several breakthroughs in machine learning and in other fields of machine learning such as language processing by accelerating the pace of the research.

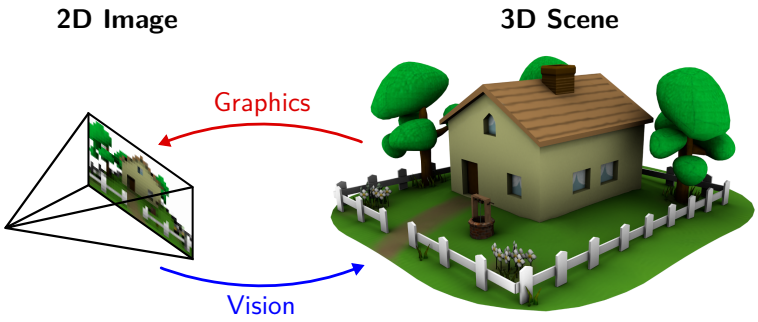


Figure 5: **Computer vision vs. graphics**

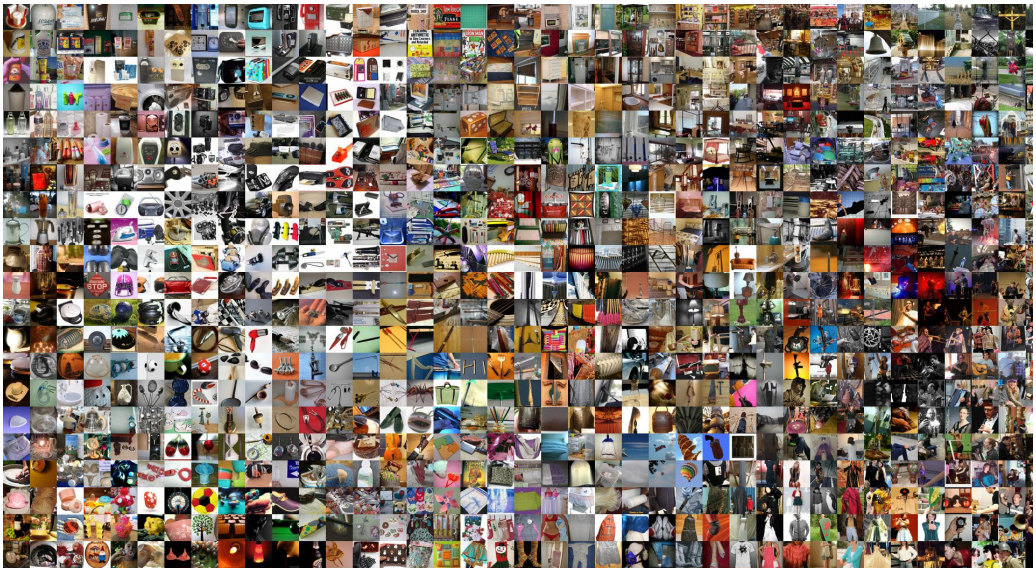


Figure 6: **ImageNet**

1.1.6 Computer Vision vs. Machine Learning

As you know in the last 10 years deep learning has revolutionized computer vision. But also computer vision has formed the basis for revolutionizing deep learning. Here you see as an example the progress on imagenet classification problem where there are one million images with 1000 different categories. The goal is to classify each of these images into one category. The progress in this problem has somewhat stalled until 2010 until it has been shown for the first time that deep learning can significantly improve on the performance on this problem. By 2015 we have surpassed human performance on this particular problem. While humans are not designed for it. It is not a very easy task for humans to solve. There was a steep decrease in top five error rate: How often you get an error if the correct example correct class label is not within the top five with your predictions.

1.1.7 CVPR

So this number has decreased tremendously since 2011 and 2012. Now we are surpassing human-level performance and are going down further every year. This fact that computer vision started to work really on relevant problems has also increased attention and accelerated the field. Here you can see the number of submitted (dark blue) and accepted papers (light blue) to CVPR (one of three top tier conferences in our field). You can also see that it's a relatively competitive conference, where about only one fourth of the submitted papers is accepted. But what you can see is that both the submitted and accepted papers have grown in particular over the last 10 years to numbers much much larger than you the size of the field in the 80s and 90s was. So the innovation cycles are very short in this field and a lot of people working on these problems today. This is also reflected in the attendance of the CVPR conference which is hitting the ten thousands, while it was around one thousand when I started doing research in computer vision around 2008/2009.

Because some of the tasks can now be solved with computer vision, there has been a huge industry developing around computer vision. Computer vision algorithms are used for all kinds of problems. We will see some of them in the third unit of this lecture. Here is just an example of some of the companies that are sponsoring. They're giving money for this you know for organizing the CVPR conference and that also presents their latest innovations that also collaborate with researchers in the fields. This is just growing very quickly due to the

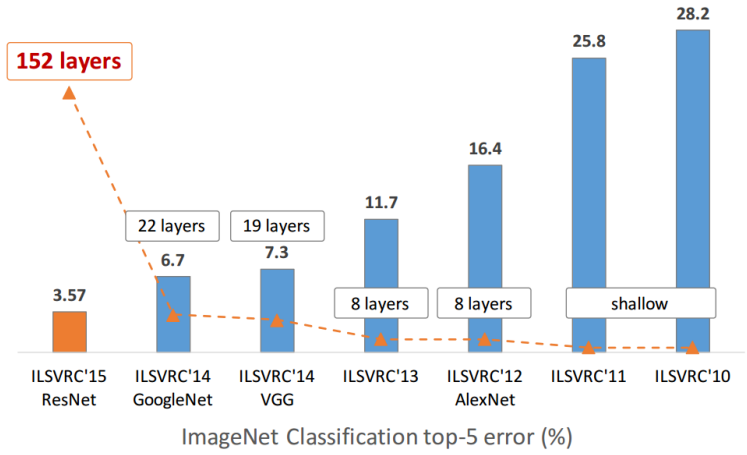


Figure 7: **ImageNet Classification ranking**

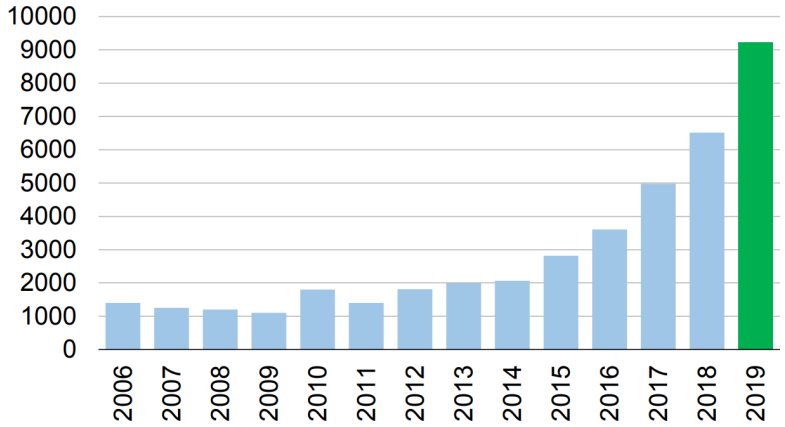


Figure 8: **CVPR attendance**

success that computer vision had recently.

1.1.8 Why is visual perception hard?

Why is visual perception hard? As in many areas, in the beginning, it was underestimated. There's this famous AI memo from Seymour Papert, who proposed in a summer vision research project with interns to basically solve computer vision over the period of the summer. Now it turns out that we're still working on this problem today, because it is not so easy. It is not like the problem can be completely separated into this independent subproblems and solving these independent sub-problems and composing them together will solve the situation. And also each individual sub-problem turned out to be harder than initially expected.

Now why is visual perception hard? It feels very easy to humans if we look at how babies grow and learn about interacting with the environment. They learn very fast even at an age of of one year they realize how to manipulate devices that are very hard to manipulate for a robot. One key aspect why visual perception is hard, is that we as humans look at pictures and immediately within milliseconds obtain an understanding of what is happening. In that picture what the computer sees is just a pixel matrix and we need to interpret this pixel matrix in order to come up with a hypothesis about what is happening in that image. And this interpretation is hard.

[Train]

If we take a 3D scene and project it into an image and then we want to reconstruct that 3D scene from that image then a lot of information has been lost through this projection process. So we are dealing with an ill post inverse process, where many 3D solutions are possible for a single 2D image. So here is this famous workshop metaphor from Adelson and Pentland. They they make the point that in order to explain an image in terms of reflectance, lighting and shape a painter a designer and a sculptor will design three different but plausible solutions. A painter might just paint that image as it is in 2D. A light designer might create a set of light sources that project light onto a wall such that the image appears with exactly that contours and shading.

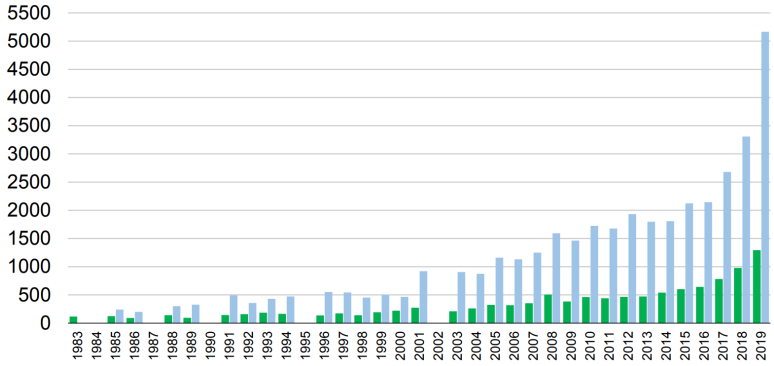
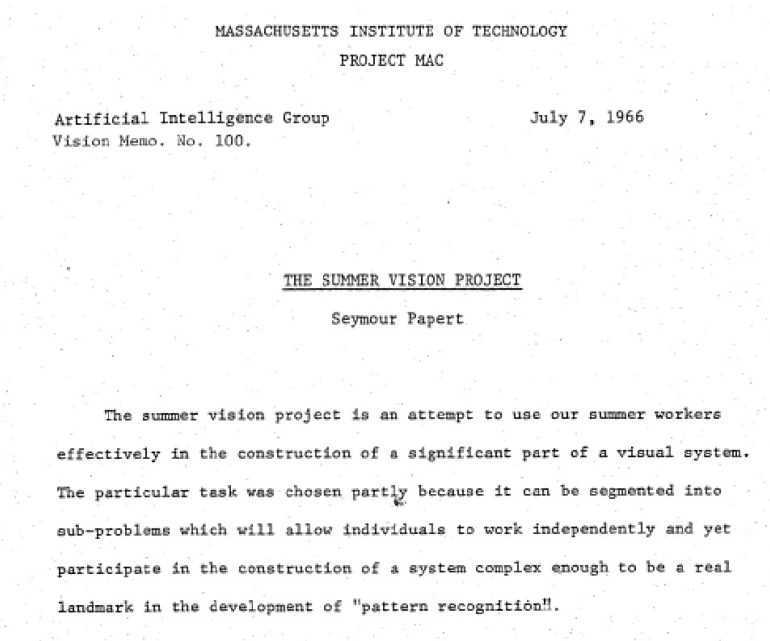


Figure 9: **CVPR papers**



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

Artificial Intelligence Group July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

Figure 10: **Summer project**

But it is actually a white wall. And a sculptor might create a sculpture that is oriented such that the shading of this picture appears through different inclination relative to the light direction that is coming in. So there are multiple solutions just to explain the single image. In fact there's infinitely many solutions and this is what makes this computer vision problem so hard and ill-posed.

[Post picture]

Below is the famous Ames room illusion. This is a room that from a particular point, from particular perspective looks like a real room but it is actually not. It is actually deformed and through this particular point of view it looks like it would be a real room but it is not, because it is not objects that are in that room. It looks as they if they were of different sizes.

Let's talk about some more challenges. One challenge in recognizing objects is that depending on the viewpoint the pixel matrix might dramatically change despite this being the same object. Depending on the viewpoint the pixels change dramatically and you need to recover this invariance and recover that this is the same object. The same holds true for deformation. There are many objects that are non-rigidly deforming giving rays to new pixel constellations. The goal is to be able to interpret that this is still the same object.

Occlusions are another challenge. There are parts occluded, which makes it hard for some of the objects to be recognized. Here is an example of illumination. This is a scene that has been illuminated by a particular light source. Can you recognize what that scene shows? Well this has been illuminated by a laser light and depending on where we project that laser light, the pixel matrix changes completely. And also our understanding of that scene changes completely. But it's always exactly the same scene pictured under always exactly the same viewpoint. So there is a intricate interplay between materials and light that give rise to a lot of different images despite showing exactly the same scene.

[Bathroom1, Bathroom2]

Here's another example from Jan Koenderik showing the same phenomena that it depends on where you



Figure 11: **Ames room**

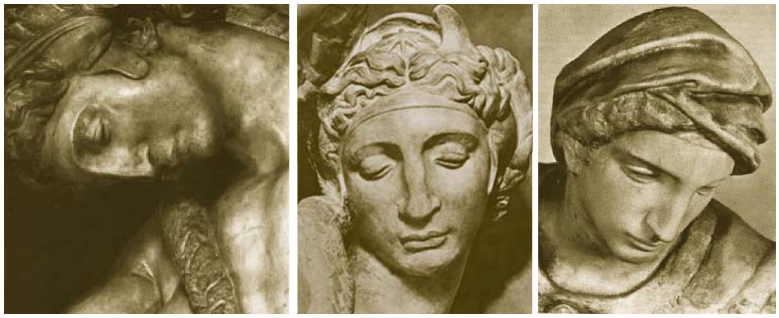


Figure 12: **Viewpoint challenges**

put the light source how the scene appears.

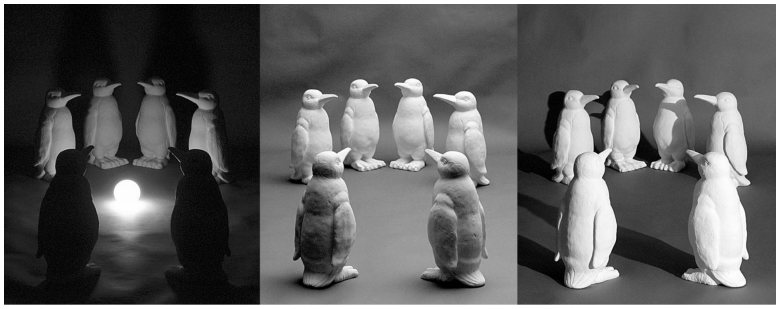


Figure 13: **Challenge of illumination**

Also motion is challenging. Motion can create blur. It is a useful cure. We're interested in recovering the motion itself.

Perception and measurement often differ and here are some examples. This is a famous example from Edward H. Adelson. I ask you to focus on these checkerboard patches that are indicated by A and B. If you look at this image, most people would say well it's clear that the A field is darker than the B field. But this is a this is what you perceive.

[Adelson Illusion 1 + 2 removed because too much space needed]

If you put stripes of the same color next to the A and B fields, you will realize that they actually have exactly the same color. So perception and measurement can be quite different. The measurement is saying you if I take a picture of this with a digital camera saying this has exactly the same grey value despite you perceive them very different. The same holds true for occluding contours. There's this famous Gaetano Kanizisa. If we look at this picture, despite there is no triangle, most humans perceive a triangle occluding three circles. The same holds true for these other objects that are only present because they appear through the contours through looking at the contours.

[Completition removed because too much space needed]



Figure 14: **Challenge of motion**

Here's another example. What do you see in the following picture? Can you recognize something? Typically it is hard in the beginning. But if you've seen it is very easy. There is a dog. So integrating this image based on local features alone will not lead you anywhere. It is very hard for computer vision algorithm to interpret based on the larger holistic structure. So perception and measurement can be quite different.

[dog removed because too much space needed]

Here's another example for a slight variation of this in terms of local ambiguities. So what you see in the circles is always exactly the same pixel arrangement. Depending on where you have it in which image it can mean something very different. So here from the context locally it is ambiguous but from the context we identify this as a telephone, shoes, car, etc. So sometimes just from local context is impossible to distinguish what an object is. But it becomes very clear if you look at the scene as a whole. So it's important to integrate scenes as a whole.

[Challenges local ambiguity removed because too much space needed]

Here is another example. It is actually a counterexample and serves as an illustration how strong your prior knowledge about scenes is. What you see here is a person. But you also see some objects. Do you recognize what these objects are? Despite it being very blurry you can probably recognize there is a phone, a mouse and a monitor. Maybe it's a printer in the back. But if i show you that same image unblurred with the original resolution you can see that none of these objects is actually present. So despite you were very sure here that these are the objects that are present they are actually not present in that image. And this is a kind of a counterexample. It is very unlikely for this scene. Probably you have seen such a scene before and that scene appears naturally in the distribution of images that you observe during your lifetime, but because you have such a strong prior knowledge you have developed such a good model of the world you have such a strong hypothesis about what these objects are that you're kind of surprised that the objects are not what you think they are.

[2 example pictures]

Another challenge is the variation in the object itself. One example is chairs. Because despite having all the same functionality, they look very different. If you want to distinguish chairs from some other object category you need to solve the problem that despite all the chairs look different, they are part of the same category. You want to categorize all of them as being some suitable objects. So the intraclass variation is a problem. If you wanna separate classes from each other but you want to classify each element of a class correctly, that is very challenging.

[ChairsVariety removed because too much space needed]

And then also the sheer number of object categories in the world that exist and can be named is very large and that makes this problem very challenging. It is estimated that there are between ten thousand to thirty thousand high level object categories and there are of course much much more if you go more fine-grained.

1.2 History

1.2.1 Pre-History

This unit is about a brief overview over the history of computer vision. Of course it is a very shortened summary of what has happened over the last 50 years and it's a personalized viewpoint. With this being said I hope it still can provide you some intuition about on where the field has come from and where the field is going to. Before we start with the last 50 years of computer vision I just want to give you a very rough pre-history of course.

The perception of humans has been investigated before computers have been invented. For instance the prospective projection process has already been described in the 15th century by Leonardo da Vinci. Johann Heinrich Lambert, after which the Lambertian reflection is named, has made important contributions to the

area of photometry. And Karl Friedrich Gauss has contributed to a lot of different aspects of science. But in particular relevant to name here is the method of least squares, which is a very useful tool across all of sciences but in particular also in computer vision as used everywhere. And then in the 19th century Charles Wheatstone has looked at the problem of stereopsis and producing stereo images. So this has happened all like several hundred of years before the history that we are looking at.

What you can see here is the so-called perspective graph. This is a tool that Leonardo da Vinci has developed for drawing better pictures of the real world that are that adhere better to the actual perspective projection that a human observes. That tool is basically a focal point where the eye of the observer or the artist is fixated and then the picture is drawn on a glass surface such that the points that are drawn correspond to actual 3D points in the scene. The perspective effects are taken into account by changing the distance between this glass surface and the focal point. He had had a pretty good understanding of how perspective projection worked.

[Perspectograph]

In 1839 a technique called "daguerreotype" has been introduced, which was the first publicly available photographic process invented by Louis Daguerre. It has been widely used in the 1840s and 1850s before other photometric techniques have been invented. On the right you can see such a camera and the resulting photograph. There is a silver plated copper sheet that was polished and treated with some films fumes to make it light sensitive. The resulting latent image that was then projected onto that sheet was made visible by fuming it with mercury vapor and removing its sensitivity to light by some chemical treatment. It was then rinsed, dried and sealed behind the glass. It was very fragile but it led to pretty astounding photographs at that time already.

[Daguerrotype removed because too much space needed]

Another important prehistoric landmark is the great arc of india survey, where this was a multi-decade project to measure the entire Indian subcontinent with scientific precision and was using trigonometric tools. In some parts of this process it was under the leadership of George Everest, after which the mountain was named and who was responsible for the survey of India. They did some form of manual bundle adjustment, the techniques that you're gonna get to know during this class. They have done this manually in order to prove amongst many other things that the mount Everest is the highest mountain on earth.

[GreatArcIndia removed because too much space needed]

So now let's look a bit into the more recent history of computer vision. If we want to categorize, the different eras of the computer vision research, this is what i would propose. So there's multiple waves of development. In the 1960s they were detecting lines and trying to fit simple geometric blocks to these lines in order to understand the world. For instance to give a robot a understanding of the world such that the robot can act in that world. But the limitations became clear very early on. One limitation was that low level vision was more difficult than was fought initially. So this led in the 70s to a surge of interest in low level vision research cumulating in the development of stereo and optical flow and shape from shading and other techniques.

1.2.2 Waves of development

In the 80s neural networks became popular again in particular through the invention of the backpropagation algorithm and the first models for self-driving have been proposed. Then in the 1990s Markov random fields and graphical models which are also part of this lecture have been quite popular in the community allowing for removing some of the ambiguities in dense stereo multiverse stereo by using a global optimization perspective on the problem. The 2000s was the decade of the feature descriptors and feature detectors like sift, which enabled for the first time really large scale structure from motion, 3D reconstruction and recognition approaches. Then from 2010 onwards neural networks have made a real impact and have surpassed the classical techniques that have been there in the computer vision community on many important problems and they are the dominating techniques today whenever large data sets are available leading to a very quick growth of the field and also commercialization.

1.2.3 Single events

[Steps of computer vision history]

So let's start with at 1957 with stereo. We can see on the right an analog implementation. But it is the first implementation of stereo image correlation. Given two images - in this case these were photogrammetry images - taken from an aerial perspective in order to recover an elevation map from the earth's surface. Given these two images correlate features in these images such that such a digital elevation map appears. This process was done manually until then, but this was the first machine that was automatizing this process through such an analog implementation of a stereo correlator. They implemented many of the techniques that are still useful and part of the state-of-the-art models today like image pyramids and course to find estimation and warping etc.

[Stereomat]

Then in 1958 Rosenblatt invented the famous perceptron and demonstrated with this very simple linear classifier that was trained using the perceptron algorithm. He showed that it is possible to classify simple patterns for instance into male and female. There were some nice properties about this algorithm for instance Novikoff proved the convergence of this algorithm but it was overhyped at the time. Rosenblatt claimed that percept the perceptron will lead to computers that will walk talk see write reproduce and are conscious of their existence and this was expected to happen over the course of the next decade. But we still don't have machines that can do all of this today.

[Rosenblatt]

Some people in the community consider the 1963 as the real cornerstone of computer vision. Larry Roberts, one of the inventors of the internet, actually but in his PhD, he was concerned with image understanding. Particular machine perception of three-dimensional solids. He wanted to understand images such that they are useful for robotic manipulation. So in contrast to simple pattern recognition, he wanted to interpret images as projections of 3D scenes and recover the underlying 3D structure of an object from the topological structure of the lines. The approach preceded by taking images of simple objects on simple backgrounds extracting edges and then coming up with a 3D model that can be rotated and manipulated and that was supposed to be the input for a robotic system in order to interact with the environment. So one of the motivating factors of computer vision was actually robotics. At some point the community deviated a little bit from robotics and robotics developed its own field and computer vision developed its own field. But nowadays these two fields are converging again to a unified research direction, because people have realized that these independent developments are insufficient to actually develop robotic systems that work in the real world. This was a very simple system. It had many problems and the challenges that were present were heavily underestimated. It was completely not robust to the to noise in the input image. It was also only working for such simple scenes with uniformly colored objects and perfect lighting and simple backgrounds.

[Blockworlds removed because too much space needed]

One example that's also often given for this underestimation of the challenge of computer vision is this paper Seymour Paport where in 1966 they thought that they can take a couple of interns over the summer and solve computer vision. And as we know computer vision in many aspects has made tremendous progress but it is far from being solved even in terms of the goals that they had at the time.

[Summer project]

In 1969 Minsky and Papert published a famous book "Perceptrons" that showed several discouraging results about neural networks and that led to a decline in research on neural networks. Only later in the 80s and again in 2010 reappeared and resurfaced. It contributed to this decline in the symbolic AI research in the 70s, which nowadays plays a relatively small role and hasn't led to a lot of success stories compared to what it has machine learning led to in 1970.

[Perceptron book]

Some researchers at MIT developed the Copy Demo, which was the culmination of this research on block world's perception, where we were using such block worlds perception algorithm to recover the structure of a simple scene. And then have a robot plan and build a copy of a set of blocks from another set of blocks. You can see this illustrated here. There is the robot arm. There is a set of blocks that should be then formed into an arrangement to replicate another set of blocks. This task already involved, vision, planning and manipulation. But it was realized very quickly that this approach that was taken the low level edge finding is not robust enough for this task. It's very brittle and so there was a lot of attention created on improving low level vision, because it was thought that once low level vision has been solved, this task can also be solved. But today we know that there are more challenges to it than just solving low-level vision. Solving low-level vision in isolation is also not easy.

[CopyDemo removed because too much space needed]

In 1970 Horn published the famous technical report in his PhD thesis basically on shape from shading on how to recover 3D from a single 2D image. Not based on geometric cues like lines and edge and point features, but looking at the shading of the surface. So you can for instance see that at the side of the nose we have much darker colors than on the front of the nose and from just the intensity of the colors there can be formulated constraints that if if they are well integrated leads to a elevation map of the surface. However it's a very ill-posed problem. There are many solutions to it. And so the method required a smoothness and regularization. Regulation to constrain the problem will see such type of approaches also during the lecture.

[Noses]

In 1978 intrinsic images have been proposed as a intermediate representation. The idea was to decompose an image into its different intrinsic components and into different 2D layers. This is not a 3D representation. It is a 2D pixel representation. For instance separate the reflectance from the shading or separating the image into motion and geometry. Now this was believed to be useful for many downstream tasks such as object detection. It helps the object become independent of shadows and lighting which makes it more invariant. This should help object detection. This intrinsic image composition has developed into its own field with many follow-up papers on this approach.

[Intrinsic images]

In 1980 photometric stereo was developed by Woodham. It is a method for determining surface orientation from multiple images. In contrast to shape from shading multiple images were used with known or unknown light source position but a point light source position. And so instead of using heavy smoothness for regularization in this case multiple observations per pixel are available. It can be shown that if you have at least three different observations per pixel you can actually reconstruct the surface normal from which you can integrate depth or geometry. This has led to many approaches that are also in industrial applications today with unprecedented detail and accuracy. The assumption of the shape from shading approach was that the object surface was homogeneous and following a lambertian reflectance in this case also the lambertian and homogeneity assumption were used but they have been later on subsequently relaxed. So this approach became more powerful. It could handle objects that were non-lambertian and that also had texture.

[Photometric stereo]

In 1981 despite the key ideas having been around for more than 100 years formerly the essential matrix has been reintroduced by Longuet-Higgen. The essential matrix defines two view geometry. We have two cameras with a left and right image plane. The essential matrix defines this geometry as a matrix that maps points for to a so-called epipolar line on the right. Such that if I have taken these two images at the same point in time or the scene has been kept static during acquisition. It is sufficient to search along this 1D line for the correspondence of this point in order to being able to triangulate it and to reconstruct the 3D geometry of that point or many points to form a surface. And so this was really a foundation of the development of stereo techniques. It has also been shown how this matrix can be estimated from a set of 2D correspondences, which is the extrinsic camera calibration problem.

[Epipolar removed because too much space needed]

Stereo is a method for extracting depth images from depth maps from two images just as we humans observe the world with two eyes. If you have two images taken from slightly different vantage points, you can also reconstruct 3D using the epipolar geometry as shown. But early approaches that correlated these features didn't work very well, because of all the ambiguities because points in these two images looked similar despite they were not the same 3D points. So in 1981 additional constraints have been added in particular they used dynamic programming (the viterbi algorithm) to introduce constraints along individual scan lines, smoothness constraints, occlusion constraints, etc. This allows for overcoming ambiguities in the stereo matching process but because each scan line was processed independently it led to streaking artifacts between the scan lines.

[BakerStereo removed because too much space needed]

In 1981 also the most popular paper on optical flow has been published called determining "Optical flow from Horn and Schunk". Optical flow is an even more challenging problem than stereo, because it is a 2D surge problem. Optical flow doesn't assume that the scene remains static between the two images have been taken but things can move arbitrarily and optical flow refers to the pattern of apparent motion of objects surfaces and edges in a visual scene. On the right is an example where you can imagine these little errors indicating the motion of a sphere that is rotating. This is an actual measurement that has been made with this algorithm. The goal for optical flow is for every pixel to determine how far the pixel has moved between two frames that have been captured. For instance of a scene where a sphere is rotating or a car is moving.

It has been investigated already by Gibson in the 1950s who was a psychologist to describe the visual stimulus of animals. For instance when animals are approaching a surface for landing this low-level cues are very important. So the Horn and Schunck algorithm is a variational algorithm that integrated this prior constraints in a mathematically concise fashion in order to recover dense optical flow fields. This algorithm had a lot of assumptions and it was subsequently revised relaxing these assumptions and yielding better optical flow algorithms.

[HornSchunk removed because too much space needed]

In a similar way in 1984 discrete Markov random fields have been considered as a way of encoding prior knowledge about a problem such as stereo or flow or image denoising. The idea is similar to this discretized variational flow formulation, where constraints between neighboring pixel sites have been introduced. But in this case it is a discrete formulation and there is a whole variety of inference algorithms that are used in our days for optimizing such problems such as variational inference sampling, belief propagation, graph cuts, ...

[Markov Random fields]

The 1980s also have been a time where part-based models have become popular. For instance in 1973 the famous pictorial structures model in 1976, in generalized cylinders or in 1986 superquadrics. This is a set of primitives that are parameterized with just a few parameters these are very basic 3D shape components that despite being very simple can be composed into quite complex scenes with very few bits. With Very few parameters we can create 3d scenes that are semantically meaningful and of course being able to infer such part-based representations is also useful for downstream tasks such as in robotics where we want to have a good understanding about the geometric relationships of different objects or parts. In the 1980s the first ideas haven't worked so well but they have been revised over the last couple of years using deep learning techniques and now start to work much better than they have been working in the past.

[Pentland]

In 1986 also the backpropagation algorithm for training neural networks has been popularized by Rumelhart, Hinton and Williams. It's really the main workhorse of deep learning still today that drives a lot of computer vision applications. The key of this algorithm as we all know is that it is using dynamic programming or dynamic computation in order to very efficiently compute millions of gradients of a loss function with respect to the parameters of such a deep neural network. This algorithm has been known already since the 1960s, but this was the time where it was demonstrated to be also empirically successful.

[backpropagation removed because too much space needed]

In 1986 also self-driving cars have been demonstrated for the first time. For instance this is a example. That has been a car that has been developed by one of the pioneers of self-driving in Europe and Dickmanns in the context of a European project called EUREKA-Prometheus. It was demonstrated in collaboration with Daimler-Benz. This first vehicle was driving up to 40 kilometers per hour on a highway. Follow-up vehicles were driving also up to 150 kilometers per hour there (Germany-Denmark and back). It seemed at the time that self-driving would not be very far away. But still today we do not have self-driving vehicles everywhere on the street. So it is also also an indicator for underestimating the difficulties in self-driving where accuracy is important, where safety is important and where you have to have models that are very robust and reliable.

[Vamors - self driving]

In 88 the self-driving car ALVINN has been developed which was the first self-driving car that was using neural networks for self-driving. So instead of these models that were using more like today self-driving vehicles different modules for perception and planning and control, this was a vehicle that was just taking the input image passing it through a neural network and outputting the steering control. It was demonstrated that on somewhat simple scenarios was able to drive up to 70 miles per hour. Already at the computer of course was much smaller the available compute.

[sfm factorization]

In 1992 structure from motion has made progress. Tomasi and Kanda published their famous factorization paper that allowed to estimating the 3D structure from 2D image sequences of static scenes with only a single camera and enclosed form solution using a singular value decomposition (at least for the autographic case). Later it has been extended to the projective case. Today these type of optimizations are then done using non-linear least squares but this is a very simple method that under certain assumptions can be solved with simple linear algebra.

There was also progress on processing point clouds from stereo or lighter or laser scans for instance the iterative closest point algorithm that was able to register to plot point clouds by iteratively optimizing a rigid or non-rigid transformation. In this example here that we have this green template and we have this red model. If we run that algorithm on this um this green template iteratively it converges to this red model such that the point clouds are registered as closely as possible. In blue is the fitted template after convergence of this algorithm. Of course this is a useful algorithm if you have 3D scans to build bigger 3D models to estimate relative camera poses for odometry or to localize a robot with respect to a map.

[ICP removed because too much space needed]

In order to integrate 3D information from multiple viewpoints or sensors another important technique that is still used today that has been introduced in 1996 in computer graphics is called volumetric fusion by Curless and Levoy. It is a technique that considers surfaces implicitly as a serial level set of a field that is defined discretely in space and that aggregates multiple of these observations. So here these blue points are in front of the surface and the red points are behind of the surface. If you have multiple of these observations this algorithm basically averages these observations and has certain properties in order to fuse multiple partial views of an object into a coherent 3D object.

[Volumentric fusion]

In 1998 there has also been first real progress on multiview stereo. This is a seminal work by Faugeras and Keriven called complete dense stereo vision using level set methods. They formulated stereo not as a matching problem but as a reconstruction of the images modeling the 3D surface as directly as the level set of an optimization formulation and projecting that into all of the input views trying to minimize the reconstruction error. This allows for flexible topology. Many of the ideas that we are still exploiting today were already present in this paper. They had a proper model of visibility and some convergence guarantees. There were also some other approaches in this era but that were rather dead ends like voxel coloring or space carving that made assumptions that in reality just don't hold. So we're restricted to very artificial scenes but this general level set and variational optimization or multi-view stereo is something that has survived until today.

[Faugeras removed because too much space needed]

In another example of global optimization in this case for stereo in the context of discrete Markov random fields was the graph cuts algorithm that has become popular in 1998. In the paper from Boykov et al. Markov random fields with efficient approximations, was an algorithm that comparably fast comparably efficiently was able to provide a global solution to a optimization problem that included unary and pairwise terms. Unary terms are per pixel observations or correlation measures. Pairwise terms encode smoothness constraints between

pixels, for instance it is more likely that two neighboring pixels have the same disparity or depth. Depth disparity is the inverse depth as we will learn about in the next lecture. Later versions also included specific forms of high order potentials and in comparison to these scanline stereo methods that I've shown previously this was a real global reasoning method that was reasoning about all the pixels in the image simultaneously not just about individual rows of the image and therefore led to much more consistent results.

[GraphCuts removed because too much space needed]

In 1998 convolutional neural networks have been proposed by LeCun and they are still one of the primary workhorses of computer vision and deep computer vision models today. After more than 20 years convolutional neural networks implement spatial invariancy via convolutions and max pooling and through this weight sharing of the kernels. They reduce the perimeter space and at the time they only got good results on MNIST and therefore it didn't get too much attention in the computer vision community. It was more in the machine learning community. In 2010 when Alexnet demonstrated that these type of models also produce competitive or state-of-the-art results on more challenging data sets, these models were adopted.

[MNIST]

In 1999 Blanz and Vetter demonstrated their morphable model for single view 3D face reconstruction. They took a repository of 200 laser scans of faces and built a subspace model from that. That was then subsequently fitted in terms of both geometry and appearance to real images with impressive performance at the time. So these results are really stunning today and also form the basis for some of the most advanced pose human body, pose and shape estimation techniques today.

[BlanzVetter removed because too much space needed]

A major landmark in 1999 was also the development of the so-called scale invariant feature transform algorithm, which is an algorithm for detecting and describing salient local features in an image. You can see an example here on the left. These little squares are little features that have been detected and described in these images. There is another picture of the same object, where the perspective has been changed, but roughly the same features have been detected at the same locations in these two views. Since you have now solved the correspondence problem you can use this for tracking and for 3D reconstruction etc. This really enables many applications, for instance image stitching, reconstruction motion estimation and many more.

[Sift]

One of these demonstrations was in 2006. This photo tourism is from a paper by Snavely Seitz and Szeliski presented at SIGGRAPH. The idea in phototourism was to produce a large-scale 3D reconstruction from internet photos. The key ingredients here really were the availability of the SIFT feature matcher, large compute resources and large-scale bundle adjustment with a very clever pipeline. It even led to a product in the end. There was the phototourism product by Microsoft, which has been just discontinued now but it was attracting a lot of interest. At a time you could take your photo collections, sort them in 3D and then browse them in 3D looking at the scene from novel viewpoints.

[Photosynth]

In 2007 PMVS(patch-based multi-view stereo) has been proposed by Furukawa and Ponce. This was a seminal paper on 3D reconstruction, because for first time it demonstrated reconstructions of various different object - quality that hasn't been seen before and also at a very high degree of robustness. So the performance of these 3D reconstruction techniques continued to increase.

[FurkawaPonce removed because too much space needed]

Here is one more example of large-scale 3D reconstruction. It was a paper building Rome in a day at ICCV. There was about building a 3d model of landmarks of cities but not from images of a single camera where you would know exactly the parameters of this camera but these were images from unstructured internet photo collections. So images were automatically extracted from the internet by searching for e.g. "colosseum" and then the algorithm was run for a few hours or sometimes a few days in order to reconstruct these landmarks or entire cities, where people have taken photographs of. There was a follow-up that's focused on making this more efficient running it. Not on a large CPU cloud but running it on a single computer.

[Rome in a day]

In 2011 the kinect camera has been put on the market. It was an active light 3D sensing device from Microsoft. And there are actually three generations of this camera. This is the newest version the Kinect camera that has just been released two years ago. The early versions version actually failed to commercialize. The idea was to use this 3D camera and an algorithm to extract information using machine learning to segment the image into different body parts to estimate the human body pose that could be used as a control allowing the user interact much more naturally with the computer by being the controller. It was discontinued, but it was heavily used for robotics and vision research because it was the first sensor that was cheap and it allowed other active depth sensing technology that was available at time allowed to navigate robots much more easily compared to other sensors that were available at the same price.

[Kinect]

In 2009, 2012 the development of ImageNet and the capability and demonstration of neural networks to solve ImageNet this classification task much better than classical techniques has really ignited both the area of

computer vision and the area of deep learning. This lead to this this increased interest in both areas today. So Imagenet was as already mentioned this large data set with 10 million annotated images. Each image has a one single class label out of 1000 categories and the goal is to classify the correct category. And Alexnet was the first neural network that significantly advanced the state of the art. They used GPU in a specific deep neural network architecture and they used a lot of data. This demonstrated that deep learning is actually effective also on challenging real world data sets.

[Alexnet removed because too much space needed]

It was realized in many sub-areas of computer vision that data sets are actually one of the important pieces, to drive progress in computer vision (allows compare approaches on a fair and equal basis, evaluate and them rank them on leaderboards), but also having these training sets as a possibility for training algorithms machine learning algorithms such as deep neural networks that had a lot of parameters. So here is a list of a couple of data sets that have been used in various image related tasks in this example here self-driving:

- Middlebury Stereo and Flow
- KITTI, Cityscapes: Self-driving
- PASCAL, MS COCO: Recognition
- ShapeNet, ScanNet: 3D DL item Visual Genome: Vision/Language
- MITOS: Breast cancer

Creating these data sets in, particular annotating these data sets, is expensive and therefore there has been also a lot of interest into how can we exploit synthetic data sets where maybe asset creation is also expensive but once you have created the assets you can actually render a lot of scenes and a lot of constellations of objects. This is still a very active research area today. So here is an example with a very simple data set but it impressively has been demonstrated that even with the simple data sets this might be very useful for learning. Below is a data set with flying chairs on random background images an that has been used training deep optical flow methods. For optical flow it is really hard to get ground truth labels because you need to know for every pixel how far it has moved. That is something that's almost impossible to annotate. So with these type of data sets there was a lot of progress despite these images that you see look very different from the images that the algorithm later was evaluated on.

[Flying things removed because too much space needed]

Because these neural networks are rather black boxes there was also a search in interest of a visualization techniques that allow for visualizing what's going on inside this neural networks. Here is one of the early examples by Zeiler and Fergus. They visualized image regions that most strongly activate various neurons at different layers of the network and found that higher levels capture more abstract semantic information and lower layers capture less abstract semantic information, etc.

[Visualization]

In 2014 it has also been demonstrated that it is very easy actually to fool deep neural networks, but they are actually not as robust as one might might have thought until then. By just adding a small tiny change (this is magnified here to the input images such that the classifier) confuses this images with a very high probability as something else. And this is this also sparked a whole research area on how to build more robust deep models and on the other hand how these more robust models can still be attacked.

[Adversial removed because too much space needed]

In 2014 another important landmark paper was the generative adversarial network paper and the variational auto encoder paper that allowed for training in an unsupervised fashion models they are able to generate new samples they haven't been seen but they are still photorealistic. In this case on faces here you can see the development over the years. On faces these images that are generated by these methods are really hard to distinguish from real images even for humans. So there is a lot of active research and image translation, domain adaptation, content and scene generation and 3D GANs that has followed up on this seminal paper.

[GAN removed because too much space needed]

O the last 10 years a lot of the recent progress has happened, where a lot of the algorithms started to work. One example is also deep phase. This was one of the first models that demonstrate performance in face recognition on par with human face recognition performance or even outperforming humans. And it was made possible by a combination of classical techniques with deep learning. And that is also maybe one of the messages I want to convey in this intro here that despite we are going to discuss a couple of very old techniques in this lecture a lot of the old techniques and ideas are still valid and still useful. Not all of them but many of them are still valid and useful and rediscovered or re-implemented today in combination with data-driven models leading to better results

[DeepFace removed because too much space needed]

Then another another line of research In 2010 was concerned with more holistic 3D scene understanding. Parsing RGB or RGBD stands for depth images into holistic 3D scene representations. There were several papers for indoors and outdoors i have chosen here one paper from our research group, where we were interested in 2013 in understanding how a traffic situation looks like just from a grayscale camera. You can see on the top the input image with some object detection. This was actually not a deep neural network object detector but a classic object detector at the time because there were no deep object detectors available. On the bottom you can see the layout of the intersection and the situation that has been inferred from that monochrome video sequence alone.

[IntersectionUunderstanding removed because too much space needed]

In 2014 there was also an impressive demonstration of 3D scanning techniques that allow now for creating really accurate replicas for instance of humans. In this case Debevec's team scanned Obama and just created a 3D presidential portrait that is now exhibited in the Smithsonian museum.

[Obama]

Then in 2015 Deepmind has published a paper how to learn human level control of video games and later also other games through deep reinforcement learning that was learning a policy directly from the image to the action. And there was also combining computer vision techniques with robotics techniques like reinforcement learning. And this was one of the first applications of these deep models demonstrating that deep models deep reinforcement learning is also possible despite the sparsity of the reward signal of reinforcement learning.

[DeepRL removed because too much space needed]

This is an example here also from Tübingen, group called image style transfer using convolutional neural networks, where they demonstrated how it is possible with a pre-trained neural network to take a real image and a painting and then produce an image that is demonstrating the content of the real image but in the style of the painting. And there's a really nice website where you can try this yourself with your own photographs.

[Style transfer]

From 2015 on semantics beyond classification of images but more fine-grained semantic was inferred using deep neural networks. One of the most famous problems in computer vision is called semantic segmentation. And these were the first deep models for semantic segmentation in 2015 here: FCN, SegNet, DeepLab, FSO, DeepLabv3. Models that have been developed in this area in semantic segmentation the task is to take an input image and we want to produce a semantic class label not for the image but for each individual pixel in that image. And as you can see it the results are quite precise and quite aligned with the object boundaries. Tthe results that were obtained are quite consistent and quite precise. So it was really demonstrated as semantic segmentation that was very brittle and didn't work very well with classical techniques classical features and classical machine learning techniques such as SVMs or structured SVM did start to work when using representation learning.

[koltun]

This is one of the still state-of-the-art object detectors and semantic instance segmenters where the goal is now to not only assign the semantic label but also the instance label to each individual pixel. It is called called mask R-CNN. And at the time produced a state-of-the-art result on a challenging data set called Microsoft common objects in context (COCO).

[Mask R CNN]

Because things started to work well people have looked at other tasks. So there was a growth in new tasks that have emerged from this. So for instance in 2017 image captioning was introduced as a new task and also visual question answering. In captioning the task is to take an image as an input and produce a sentence for that image. As you can see here in some cases works quite well and in some cases it produces some weird errors. So people were quite excited by these results but then also quickly realized that these models that have been trained were somehow also adopting the data set bias and were still lacking in a real understanding of the images and in common sense. With the growth of data sets this has changed a little bit, but but it is still a problem.

[Image captioning]

Another important area of research that has developed significantly since the 2000s is human shape and post estimation, where the goal is given a 3D point cloud or a single input image to produce a 3D model of the person or multiple people in an image. Of course this has many important applications in human computer interaction or understanding humans better in health, etc. So there were rich parametric models developed like SIMPLE and STAR here in Tubingen that even allowed for regressing these model parameters from single RGB images alone. More recent trends go towards modeling very detailed pose-dependent deformations and also clothing which can be quite challenging. So here is an example of a model already a few years ago. We can see how independently this parkour person was tracked and the pose was estimated for this person.

[humanPose removed because too much space needed]

From 2016 on deep learning has also entered the area of 3D vision. Before that deep learning was successful on classical recognition problems and 2D estimation or 2.5D estimation problems, but then from 2016 when people realized we can also apply these models for 3D vision. While in the beginning people used voxels and point clouds and meshes as representations, now there's a lot of hype around these implicit neural network based

representations. One class of this lecture is also devoted to these type of models. These allow for predicting entire 3D models even from a single image as input. And they also allow for depending on the model for predicting geometry, materials, light and motion.

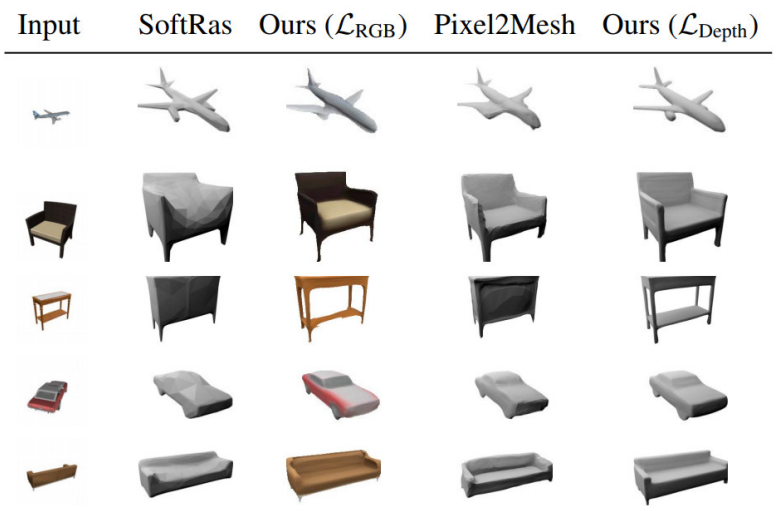


Figure 15: **3D Deep learning**

I want to conclude with a little overview. It's a very incomplete overview of applications and commercial products, where we're using computer vision technology in our everyday is live. There's for instance google's portrait mode for the smartphone, which allows to blur the background very realistically of a portrait. There is the skydio or skydio 2 drone that just came out that allows that navigates itself allows to follow a person through complicated terrain without crashing into obstacles. There are of course many companies um working on self-driving cars that haven't fulfilled the promise yet. But there is still a lot of a lot of interest and a lot of progress in this area. There is a growing interest in also virtual reality and augmented reality with new devices like virtual reality glasses but also the Hololens which is an augmented reality device and it uses a lot of computer vision for tracking the head for detecting objects in the environment and for estimating geometry and localizing itself. And here on the right is an example where Iris recognition has been used to identify a person 18 years later that otherwise wouldn't have been identified as the same person.

[applications.pdf removed because too much space needed]

This concludes the short historical review. What I want to mention in the end is that there is a lot of challenges that are still unsolved. This is an incomplete list but it gives you an idea of what are the things that people are working on. It is of course still difficult to get big data sets, in particular annotated data sets. So people look into GAN and self-supervised learning algorithms. They look also in interactive learning algorithms that more naturally interact with the environment. Accuracy is an important topic. Holistic modeling for getting self-driving cars as robust as humans or more robust than humans. Robustness and generalization is an important aspect, inductive biases are important and an understanding of the black box neural networks and the mathematics behind is an active research area. Many of these models in particular in 3D have high memory and compute requirements and so efficiency of these models and new representations are constantly developed. And there are many also ethics and legal questions that are currently investigated that are also result of the demand of commercializing some of these ideas into tools that are available to us in our daily lives.

2 Lecture 2

This lecture is about gaining an understanding of how a 3D scene is projected onto a 2D image plane.

Geometric primitives are the **basic building blocks** used to describe 3D shapes. In this unit, **points, lines and planes** are introduced. Furthermore, the **most basic transformations** are discussed. This unit covers the topics of the Szeliski book, chapter 2.1. A more exhaustive introduction can be found in the book: Hartley and Zisserman: Multiple View Geometry in Computer Vision

2.1 Primitives and Transformations

2D points can be written in inhomogeneous coordinates (as contrasted to homogeneous ones) by

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2$$

or in **homogeneous coordinates** as

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^2$$

where $\mathbb{P}^2 = \mathbb{R}^3 \setminus \{(0,0,0)\}$ is called **projective space.** Introducing a third coordinate here, we are in a 3D space (except at 0,0,0). The tilde sign is a convention for homogeneous coordinates. Homogeneous vectors are considered as equivalent when they differ only up to scale. That's why the projective space is effectively only 2D.
Homogeneous vectors allow to express vectors at infinity, intersections of parallel lines and in general it allows to express transformations very easily as concatenations of multiple ones.

An **inhomogeneous vector x** is converted to a **homogeneous vector x** as follows

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} = \bar{\mathbf{x}}$$

with **augmented vector** $\bar{\mathbf{x}}$. We say augmented vector $\bar{\mathbf{x}}$ for all homogeneous vectors which last coordinate is equal to 1.

To convert in the opposite direction we divide by the last element $\tilde{w}$:

$$\bar{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{\tilde{w}} \tilde{\mathbf{x}} = \frac{1}{\tilde{w}} \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ 1 \end{pmatrix}$$

Like this, the last element of the homogeneous vector turns into a 1 and we can read off the x and y for the inhomogeneous coordinates. Homogeneous points whose last element is $\tilde{w} = 0$ are called **ideal points** or **points at infinity.** These points can't be represented with inhomogeneous coordinates as we can't divide by $\tilde{w}$.

Comment: The transformation from homogeneous to inhomogeneous vectors resembles a lot the perspective translation that is introduced later.

2D lines can also be expressed using homogeneous coordinates $\tilde{\mathbf{l}} = (a, b, c)^\top$:

$$\{\bar{\mathbf{x}} \mid \tilde{\mathbf{l}}^\top \bar{\mathbf{x}} = 0\} \quad \Leftrightarrow \quad \{x, y \mid ax + by + c = 0\}$$

We get the line equation by the inner product of the vector $\tilde{\mathbf{l}}$ and an augmented vector. For all (x, y) for which the equation is zero, the points are located on the line. Note that we can also use another vector (not only the augmented one). The augmented vector is chosen here for convenience such that we get the line equation. We can **normalize** $\tilde{\mathbf{l}}$ so that $\tilde{\mathbf{l}} = (n_x, n_y, d)^\top = (\mathbf{n}, d)^\top$ with $\|\mathbf{n}\|_2 = 1$. In this case, $\mathbf{n}$ is the normal vector perpendicular to the line and d is its distance to the origin.
An exception is the **line at infinity** $\tilde{\mathbf{l}}_\infty = (0, 0, 1)^\top$ which passes through all ideal points.

Cross product expressed as the product of a skew-symmetric matrix and a vector:

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2\, b_3 - a_3\, b_2 \\ a_3\, b_1 - a_1\, b_3 \\ a_1\, b_2 - a_2\, b_1 \end{pmatrix}$$

The cross product is for example useful for computing line intersections.
The **intersection** of two lines is given by:

$$\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$$

Similarly, the **line joining two points** can be compactly written as:

$$\tilde{\mathbf{l}} = \bar{\mathbf{x}}_1 \times \bar{\mathbf{x}}_2$$

The symbol $\times$ denotes the cross product.

An example:

$$\tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2 = [\tilde{\mathbf{l}}_1]_\times \tilde{\mathbf{l}}_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} -2 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

The cross product for two parallel lines is therefore 0.

More complex algebraic objects can be represented using **polynomial homogeneous equations.** For example, **conic sections** (arising as the intersection of a plane and a 3D cone) can be written using quadric equations:

$$\{\bar{\mathbf{x}} \mid \bar{\mathbf{x}}^\top \mathbf{Q}\, \bar{\mathbf{x}} = 0\}$$

The intersection of a plane and a 3S cone can arise as parabola, circle, ellipse or hyperbola.
Useful for multi-view geometry and camera calibration, see Hartley and Zisserman.

3D points can be written in **inhomogeneous coordinates** as

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$$

or in **homogeneous coordinates** as

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^3$$

with **projective space** $\mathbb{P}^3 = \mathbb{R}^4 \setminus \{(0,0,0,0)\}$.
3D planes can also be represented as homogeneous coordinates $\tilde{\mathbf{m}} = (a,b,c,d)^\top$:

$$\{\bar{\mathbf{x}} \mid \tilde{\mathbf{m}}^\top \bar{\mathbf{x}} = 0\} \quad \Leftrightarrow \quad \{x,y,z \mid ax + by + cz + d = 0\}$$

Again, we can **normalize** $\tilde{\mathbf{m}}$ so that $\tilde{\mathbf{m}} = (n_x, n_y, n_z, d)^\top = (\mathbf{n}, d)^\top$ with $\|\mathbf{n}\|_2 = 1$. In this case, $\mathbf{n}$ is the normal perpendicular to the plane and d is its distance to the origin.
An exception is the **plane at infinity** $\tilde{\mathbf{m}} = (0,0,0,1)^\top$ which passes through all ideal points (= points at infinity) for which $\tilde{w} = 0$.

3D lines are less elegant than either 2D lines or 3D planes. One possible representation is to express points on a line as a **linear combination** of two points $\mathbf{p}$ and $\mathbf{q}$ on the line:

$$\{\mathbf{x} \mid \mathbf{x} = (1-\lambda)\mathbf{p} + \lambda\mathbf{q} \wedge \lambda \in \mathbb{R}\}$$

However, this representation uses 6 parameters for 4 degrees of freedom.
Alternative minimal representations are the **two-plane parameterization** or **Plücker coordinates.** See Szeliski, Chapter 2.1 and Hartley/Zisserman, Chapter 2 for details.

The 3D analog of 2D conics is a **quadric surface**. They are useful in the study of multi-view geometry. Also serves as useful modeling primitives (spheres, ellipsoids, cylinders), in terms of compact representations.

Example: Superquadrics, in order to represent geometric objects in terms of simple primitives. Allows for compression, while still preserving the dominant aspects.

2D transformations:

The simplest is a translation (2D Translation of the Input, 2 DoF) which we can again write as homogeneous representation:

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

With the homogeneous representation, we can easily chain and invert transformations (also true for the other types of transformations).

Then we have the Euclidean transformation: (2D Translation + 2D Rotation, 3 DoF)

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

$\mathbf{R} \in SO(2)$ is an orthonormal rotation matrix with $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$ and $\det(\mathbf{R}) = 1$ where SO(2) describes a special group of orthonormal matrices. Euclidean transformations preserve Euclidean distances. So points preserve the disctances in such a transformation.

Similiary introduces an arbitrary scale factor: (2D Translation + Scaled 2D Rotation, 4 DoF)

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

It preserves angles between lines, but now distances change as we have introduced a scaling factor.

Going again one level up the hierarchy, we have affine transformations: (2D Linear Transformation, 6 DoF)

$$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

Angles are now not preserved, but still parallel lines remain parallel.

The last one up the hierarchy is perspective transformation: (Homography, 8 DoF)

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\,\tilde{\mathbf{x}} \qquad \left(\bar{\mathbf{x}} = \frac{1}{\tilde{w}}\tilde{\mathbf{x}}\right)$$

where $\tilde{\mathbf{H}} \in \mathbb{R}^{3\times 3}$ is an arbitrary homogeneous 3×3 matrix (specified up to scale).

The transformation preserves straight lines.

Transformations of co-vectors (e.g. lines):

Considering any perspective 2D transformation, we can transform points using homogeneous transformations:

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$$

But what happens with co-vectors? The already transformed 2D line equation is given by:

$$\tilde{\mathbf{l}}'^\top \tilde{\mathbf{x}}' = \tilde{\mathbf{l}}'^\top \tilde{\mathbf{H}}\tilde{\mathbf{x}} = (\tilde{\mathbf{H}}^\top \tilde{\mathbf{l}}')^\top \tilde{\mathbf{x}} = \tilde{\mathbf{l}}^\top \tilde{\mathbf{x}} = 0$$

which we recognize as the line equation set to zero. Therefore, we have:

$$\tilde{\mathbf{l}}' = \tilde{\mathbf{H}}^{-\top}\tilde{\mathbf{l}}$$

Thus, the action of a projective **transformation on a co-vector** such as a 2D line or 3D normal can be represented by the transposed inverse of the matrix.

Direct Linear Transformation

We want a homography estimation using a set of 2D correspondences - which means that we want to find the transformation matrix, as to say the camera parameters. In homography, we have 8 degrees of freedom, which means that we need at least 4 correspondence pairs for estimating the transformation matrix.

Let $\mathcal{X} = \{\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}'_i\}_{i=1}^N$ denote a set of N 2D-to-2D correspondences related by $\tilde{\mathbf{x}}'_i = \tilde{\mathbf{H}}\tilde{\mathbf{x}}_i$. As the correspondence vectors are homogeneous, they have the same direction but differ in magnitude. Thus, the equation above can be expressed as $\tilde{\mathbf{x}}'_i \times \tilde{\mathbf{H}}\tilde{\mathbf{x}}_i = \mathbf{0}$. We use this expression such that we assure that the vectors are equivalent (with respect to homogeneous coordinates). Using $\tilde{\mathbf{h}}_k^\top$ to denote the k'th row of $\tilde{\mathbf{H}}$, this can be rewritten as a linear equation in $\tilde{\mathbf{h}}$:

$$\underbrace{\begin{bmatrix} \mathbf{0}^\top & -\tilde{w}'_i\tilde{\mathbf{x}}_i^\top & \tilde{y}'_i\tilde{\mathbf{x}}_i^\top \\ \tilde{w}'_i\tilde{\mathbf{x}}_i^\top & \mathbf{0}^\top & -\tilde{x}'_i\tilde{\mathbf{x}}_i^\top \\ -\tilde{y}'_i\tilde{\mathbf{x}}_i^\top & \tilde{x}'_i\tilde{\mathbf{x}}_i^\top & \mathbf{0}^\top \end{bmatrix}}_{\mathbf{A}_i} \underbrace{\begin{bmatrix} \tilde{\mathbf{h}}_1 \\ \tilde{\mathbf{h}}_2 \\ \tilde{\mathbf{h}}_3 \end{bmatrix}}_{\tilde{\mathbf{h}}} = \mathbf{0}$$

Each point correspondence yields two equations. Stacking all equations into a $2N \times 9$ dimensional matrix $\mathbf{A}$ leads to the following **constrained least squares problem**

$$\begin{aligned} \tilde{\mathbf{h}}^* &= \underset{\tilde{\mathbf{h}}}{\operatorname{argmin}} \ \|\mathbf{A}\tilde{\mathbf{h}}\|_2^2 + \lambda(\|\tilde{\mathbf{h}}\|_2^2 - 1) \\ &= \underset{\tilde{\mathbf{h}}}{\operatorname{argmin}} \ \tilde{\mathbf{h}}^\top \mathbf{A}^\top \mathbf{A}\tilde{\mathbf{h}} + \lambda(\tilde{\mathbf{h}}^\top\tilde{\mathbf{h}} - 1) \end{aligned}$$

where we have fixed $\|\tilde{\mathbf{h}}\|_2^2 = 1$ as $\tilde{\mathbf{H}}$ is homogeneous (i.e., defined only up to scale) and the trivial solution to $\tilde{\mathbf{h}} = 0$ is not of interest.

The solution to the above optimization problem is the **singular vector** corresponding to the smallest singular value of $\mathbf{A}$ (i.e., the last column of $\mathbf{V}$ when decomposing $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$, see also Deep Learning lecture 11.2). The resulting algorithm is called **Direct Linear Transformation.**

To end this subsection, here are overviews of the transformations in 2D and 3D:

2D transformations:

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix}\mathbf{I} & \mathbf{t}\end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix}s\mathbf{R} & \mathbf{t}\end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix}\mathbf{A}\end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix}\tilde{\mathbf{H}}\end{bmatrix}_{3\times3}$	8	straight lines	

3D transformations:

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix}\mathbf{I} & \mathbf{t}\end{bmatrix}_{3\times4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}_{3\times4}$	6	lengths	
similarity	$\begin{bmatrix}s\mathbf{R} & \mathbf{t}\end{bmatrix}_{3\times4}$	7	angles	
affine	$\begin{bmatrix}\mathbf{A}\end{bmatrix}_{3\times4}$	12	parallelism	
projective	$\begin{bmatrix}\tilde{\mathbf{H}}\end{bmatrix}_{4\times4}$	15	straight lines	

2.2 Geometric Image Formation

Basic camera models

Pinhole camera: 'Eye', first camera obscura (4th century B.C.) - light gets inside a room through a tiny hole.

Physical versus mathematical pinhole camera model:

In a physical pinhole camera the image is projected up-side down onto the image plane which is located **behind** the focal point, whereas in the mathematical camera model, the image plane is **in front** of the focal point.

Both models are **equivalent,** with appropriate change of image coordinates. We consider the mathematical model in the lecture because it has some advantages in the sense that the image is not projected upside down.

Projection models:

Orthographic projection: assumes light rays to travel parallel to the image plane; as distances on the image plane should correspond to the real world distances, this is used for inspection for instance.

Perspective projection: the one of the pinhole model where light rays "cross". The traditional one that is used from almost all cameras that we know from our everydays life. We get closer to the orthographic projection, if we increase the focal length or increase the distance from the object (or weak perspective one).

An **orthographic projection** simply **drops the z component** of the 3D point in camera coordinates $\mathbf{x}_c$

to obtain the corresponding 2D point on the image plane (= screen) $\mathbf{x}_s$.

$$\mathbf{x}_s = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{x}_c \quad \Leftrightarrow \quad \bar{\mathbf{x}}_s = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{x}}_c$$

Orthography is exact for telecentric lenses and an approximation for telephoto lenses. After projection the distance of the 3D point from the image can't be recovered.

In practice, world coordinates (which may measure dimensions in meters) must be scaled to fit onto an image sensor (measuring in pixels) ⇒ **scaled orthography:**

$$\mathbf{x}_s = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \end{bmatrix} \mathbf{x}_c \quad \Leftrightarrow \quad \bar{\mathbf{x}}_s = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{x}}_c$$

Remark: The unit for s is px/m or px/mm to convert metric 3D points into pixels.

Under orthography, structure and motion can be estimated simultaneously using factorization methods (e.g., via singular value decomposition which we will see later in the lecture).

For perspective projection, we can use the following relationship (which is just from the principle of equal triangles):

$$\frac{x_s}{f} = \frac{x_c}{z_c}$$

, which can also be solved for x_s of course.

In **perspective projection,** 3D points in camera coordinates are mapped to the image plane by **dividing** them **by their z component** and multiplying with the focal length:

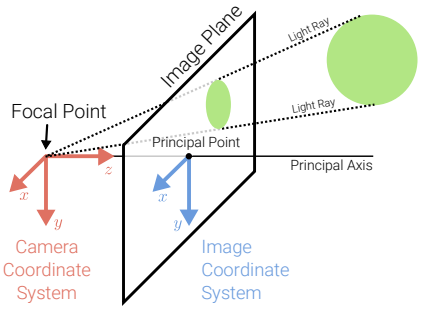
$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} f x_c / z_c \\ f y_c / z_c \end{pmatrix} \quad \Leftrightarrow \quad \tilde{\mathbf{x}}_s = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bar{\mathbf{x}}_c$$

Note that this projection is **linear** when using **homogeneous coordinates.** After the projection it is not possible to recover the distance of the 3D point from the image.

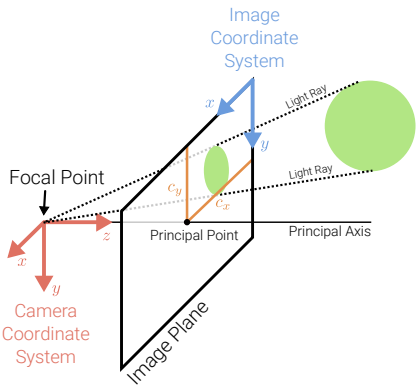
Remark: The unit for f is px (=pixels) to convert metric 3D points into pixels.

Principal point offset: Usually in practice, we compute the principal point offset to derive a coordinate system for the image plane that is much more convenient as it does not include negative values.

Without Principal Point Offset



With Principal Point Offset



To ensure positive pixel coordinates, a **principal point offset c** is usually added. This moves the image coordinate system to the corner of the image plane (see Graph).

Considering the full camera matrix, the **complete perspective projection model** is given by:

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} f_x\, x_c / z_c + s\, y_c / z_c + c_x \\ f_y\, y_c / z_c + c_y \end{pmatrix} \quad \Leftrightarrow \quad \tilde{\mathbf{x}}_s = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bar{\mathbf{x}}_c$$

The left 3×3 submatrix of the projection matrix (projection matrix is the whole 3x4 matrix) is called **calibration matrix K** The parameters of **K** are called camera intrinsics (as opposed to extrinsic pose). Here,

f_x and f_y are independent, allowing for different pixel aspect ratios. The skew s arises due to the sensor not mounted perpendicular to the optical axis. In practice, we often set $f_x = f_y$ and $s = 0$, but model $\mathbf{c} = (c_x, c_y)^\top$.

Chaining different transformations becomes very easy if we assume homogeneous coordinates:

$$\tilde{\mathbf{x}}_s = \begin{bmatrix}\mathbf{K} & \mathbf{0}\end{bmatrix} \bar{\mathbf{x}}_c = \begin{bmatrix}\mathbf{K} & \mathbf{0}\end{bmatrix} \begin{bmatrix}\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1\end{bmatrix} \bar{\mathbf{x}}_w = \mathbf{K} \begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \bar{\mathbf{x}}_w = \mathbf{P}\, \bar{\mathbf{x}}_w$$

Here, a transformation from the world coordinate system to the camera coordinate system, is changed with the camera intrinsic transformations (calibration matrix K).

Lens distortion:
The assumption of linear projection (straight lines remain straight) is violated in practice due to the properties of the camera lens which introduces distortions.
Both **radial and tangential distortion** effects can be modeled relatively easily:
Let $x = x_c/z_c$, $y = y_c/z_c$ and $r^2 = x^2 + y^2$. The distorted point is obtained as:

$$\mathbf{x}' = \underbrace{(1 + \kappa_1 r^2 + \kappa_2 r^4)}_{\text{Radial Distortion}} \begin{pmatrix} x \\ y \end{pmatrix} + \underbrace{\begin{pmatrix} 2\kappa_3 x y + \kappa_4(r^2 + 2x^2) \\ 2\kappa_4 x y + \kappa_3(r^2 + 2y^2) \end{pmatrix}}_{\text{Tangential Distortion}}$$

$$\mathbf{x}_s = \begin{pmatrix} f_x x' + c_x \\ f_y y' + c_y \end{pmatrix}$$

Images can be **undistorted** such that the perspective projection model applies.
Note here, that is practice, this is already done before the image is computed. More complex distortion models must be used for wide-angle lenses (e.g., fisheye).

2.3 Photometric Image Formation

How is an image formed in terms of intensities and colors? Light is **emitted** by one or more light sources and **reflected** or **refracted**
(once or multiple times) at surfaces of objects (or media) in the scene.

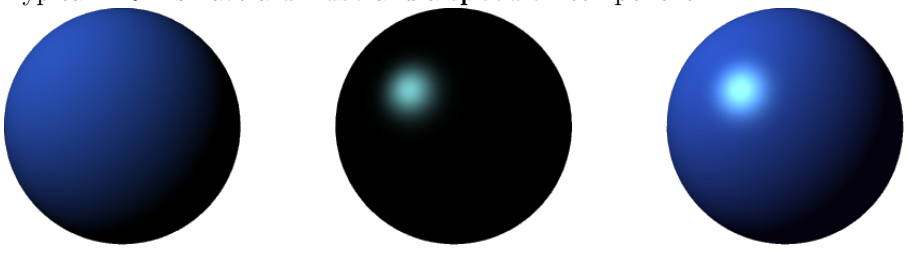
Let $\mathbf{p} \in \mathbb{R}^3$ denote a 3D surface point, $\mathbf{v} \in \mathbb{R}^3$ the viewing direction and $\mathbf{r} \in \mathbb{R}^3$ the incoming light direction. The **rendering equation** describes how much of the light L_{in}
with wavelength λ arriving at $\mathbf{p}$ is reflected into the viewing direction $\mathbf{v}$:

$$L_{\text{out}}(\mathbf{p}, \mathbf{v}, \lambda) = L_{\text{emit}}(\mathbf{p}, \mathbf{v}, \lambda) + \int_\Omega \text{BRDF}(\mathbf{p}, \mathbf{r}, \mathbf{v}, \lambda) \cdot L_{\text{in}}(\mathbf{p}, \mathbf{r}, \lambda) \cdot (-\mathbf{n}^T\mathbf{r})\, d\mathbf{r}$$

Ω is the unit hemisphere at normal $\mathbf{n}$ The bidirectional reflectance distribution function $\text{BRDF}(\mathbf{p}, \mathbf{r}, \mathbf{v}, \lambda)$ defines how light is reflected at an opaque surface. $L_{\text{emit}} > 0$ only for light emitting surfaces.

$(-\mathbf{n}^T\mathbf{r})$ - attenuation equation (if light arrives exactly perpendicular, there is no reflectance at all. or if it arrives at a shallow angle, there is less light reflected).

Typical BRDFs have a **diffuse** and a **specular** component.



Diffuse Specular Combined

A specular surface reflects more light in "ideal mirror reflection" and then it attenuates if you go further away.

The diffuse (=constant) component scatters light uniformly in all directions This leads to shading, i.e., smooth variation of intensity wrt. surface normal The specular component depends strongly on the outgoing light direction.

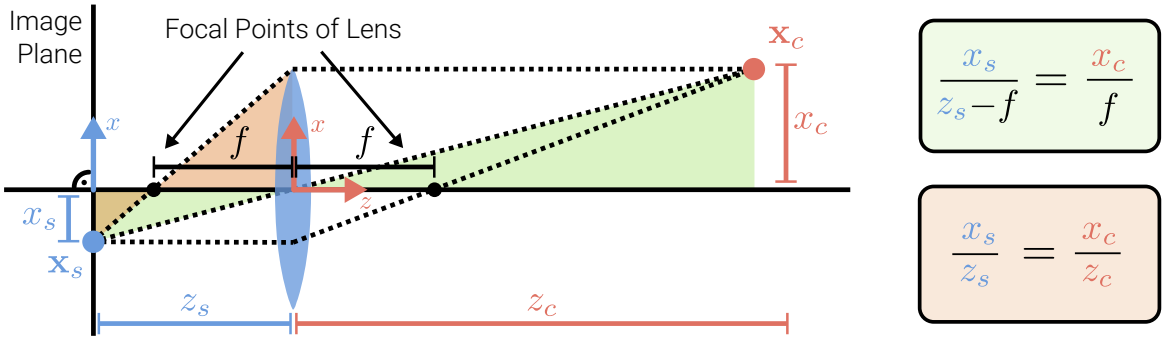
BRDFs can be very challenging in practice - very challenging distribution etc... There is a whole field specialized to this.

Fresnel effect The specular component can get stronger if the surface is further away because the viewing angle changes (example with the water reflectance). So the amount of relectance depends on the viewing angle.

Global illumination
Modeling one light bounce is insufficient for rendering complex scenes. Light sources can be shadowed by occluders and rays can bounce multiple times. textbfGlobal illumination techniques also take indirect illumination into account.

Camera lenses Why do we need camera lenses? Without them, we would often not collect enough light if we have very small pinholes (in the case that we don't want large shutter times to prevent motion blur!). So large and very small pinholes result in **image blur** (averaging, diffraction) and small pinholes require very **long shutter times** ($\Rightarrow$ motion blur). A lens system can help for example to decrease long shutter times or in the case of a large pinhole, to change focal length and to decrease image blur.
Cameras use one or multiple **lenses** to accumulate light on the sensor plane. Importantly, if a 3D point is in **focus**, all light rays arrive at the same 2D pixel. For many applications it suffices to model lens cameras with a pinhole model. However, to address **focus, vignetting and aberration** we need to model lenses.

The thin lens model



The equation in the green box describes the relationship of the red triangles, while the equation in the red box describes the relationship of the green triangles. From these relationships, we derive the following:

$$\frac{x_s}{x_c} = \frac{z_s - f}{f} \quad \wedge \quad \frac{x_s}{x_c} = \frac{z_s}{z_c} \quad \Rightarrow \quad \frac{z_s - f}{f} = \frac{z_s}{z_c} \quad \Rightarrow \quad \frac{z_s}{f} - 1 = \frac{z_s}{z_c} \quad \Rightarrow \quad \frac{1}{z_s} + \frac{1}{z_c} = \frac{1}{f}$$

The **thin lens model** with spherical lens is often used as an approximation. Properties: Axis-parallel rays pass the focal point, rays via center keep direction. From Snell's law we obtain $f = \frac{R}{2(n-1)}$ with radius R and index of refraction n

Depth of Field
The image is **in focus** if $\frac{1}{z_s} + \frac{1}{z_c} = \frac{1}{f}$ where f is the focal length of the lens. For $z_c \to \infty$ we obtain $z_s = f$ (lens with focal length $f \approx$ pinhole at distance f). If the image plane is **out of focus**, a 3D point projects to the **circle of confusion** c. The circle of confusion is a little disc - if the disc is larger than the pixel size, we get blur!

The allowable depth variation that limits the circle of confusion c is called depth of field and is a function of both the focus distance and the lens aperture. Typical DSLR cameras have a Depth of Field Indicator which describes the range where the image appears sharply.
The commonly displayed **f-number** is defined as

$$N = \frac{f}{d} \qquad \text{(often denoted as } f/N\text{, e.g.: } f/1.4)$$

To control the **size of the circle of confusion,** we change the lens **aperture**. An aperture is a hole or an opening through which light travels. The aperture limits the amount of light that can reach the image plane. Smaller apertures lead to sharper, but more noisy images (less photons).

Chromatic Aberration
The **index of refraction** for glass varies slightly as a function of wavelength. Thus, simple lenses suffer from **chromatic aberration** which is the tendency for light of different colors to focus at slightly different distances (blur, color shift). Chromatic aberration can often be observed, especially in the border regions. To reduce chromatic and other kinds of aberrations, most photographic lenses are compound lenses made of different glass elements (with different coatings)

Vignetting

Vignetting is the tendency for the brightness to fall off towards the image edge. Composition of two effects: natural and mechanical vignetting. Natural vignetting: foreshortening of object surface and lens aperture. Mechanical vignetting: the shaded part of the beam never reaches the image Vignetting can be calibrated (i.e., undone) which is also usually done.

2.4 Image Sensing Pipeline

The **image sensing pipeline** can be divided into three stages:

- **Physical light transport** in the camera lens/body
- **Photon measurement** and conversion on the sensor chip
- **Image signal processing (ISP)** and image compression

Shutter

A **focal plane shutter** is positioned just in front the image sensor / film. Most digital cameras use a combination of mechanical and electronic shutter. The shutter speed (exposure time) controls how much light reaches the sensor It determines if an image appears over- (too light)/underexposed (too dark), blurred (motion blur) or noisy.

Two main principles: CCD and CMOS for light sensors.

CCDs move charge from pixel to pixel and convert it to voltage at the output node. **CMOS** images convert charge to voltage inside each pixel and are standard today. CMOS have an advantage over CCDs nowadays, which came with advanced litography technoly (see https://meroli.web.cern.ch/lecture_cmos_vs_ccd_pixel_sensor.html for more information about this).

To measure color, pixels are arranged in a **color array,** e.g.: Bayer RGB pattern. Missing colors at each pixel are interpolated from the neighbors (demosaicing)

G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

Bayer RGB Pattern

rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb

Interpolated Pixels

Each pixel **integrates the light spectrum** L according to its spectral sensitivity S:

$$R = \int L(\lambda)\, S_R(\lambda)\, d\lambda$$

Explanation: Integral over light sensitivity of the different wave lengths, integrated over all different wave lengths coming in. The spectral response curves are provided by the camera manufacturer.

Different color spaces

RGB, L*a*b*, HSV: seperating into hue, saturation and value

As humans are more sensitive to intensity differences in darker regions, we often use **Gamma compression**. This is when we **nonlinearly transform** (left) the intensities or colors prior to discretization (left) and undo this transformation during loading.

Image compression

Often images are compressed into a format similar to JPEG. Typically luminance is compressed with higher fidelity than chrominance. Often, (8×8 pixel) patch-based discrete cosine or wavelet transforms are used. **Discrete Cosine Transform (DCT)** is an approximation to PCA on natural images. The coefficients are quantized to integers that can be stored with Huffman codes. More recently, deep network based compression algorithms are developed (improving the compression a lot compared to DCT).

3 Structure-from-Motion

This lecture builds on the concepts from the second lecture to reason about the 3D structure of a static scene from multiple images observing that scene.

3.1 Preliminaries

3.1.1 Camera Calibration

To infer 3D-information from a collection of (2D) images, it is important to know the intrinsics and extrinsics of the camera setup. The process of finding the intrinsic and extrinsic parameters is konwn as *camera calibration*. Most commonly, a known calibration target is used, such as an image or a checkerboard. First, this target is captured in different poses. Then, features, such as corners, are detected in the images. Finally, the camera intrinsics and extrinsics are jointly optimized by non-linear optimization of reprojection errors.

There exists a variety of calibration techniques that are used in different settings. These methods differ algorithmically, but also in the type of assumptions and calibration targets they use: 2D/3D targets, planes, vanishing points, etc.

3.1.2 Feature Detection and Description

Local, salient regions in an image can be described by *point features*, which are vectors. They can be used to describe and match images taken from different viewpoints and form the basis of sparse 3D reconstruction methods covered in this lecture. Features should be invariant to perspective effects and illumination and the same point should have similar vectors independent of pose or viewpoint. Plain RGB / intensity patches will not have this property, we need something better. As an example, think of the RGB matrices that describe one image and the same exact image rotated 180°. The matrices will be completely different, although they describe the exact same set of points.

One algorithm that provides us with what we need is the Scale Invariant Feature Transform (SIFT) [32]. SIFT constructs a *scale space* by iteratively filtering the image with a Gaussian and scaling the image down at regular intervals. Adjacent scales are subtracted, yielding Difference of Gaussian (DoG) images. Difference of Gaussian filters are "blob detectors"; the interest points (blobs) are detected as extrema in the resulting scale space. An example of what the scale space looks like is shown in Figure 16. It can be seen that certain features (such as the tie knot) are recognized (high intensity) at certain scales. After extracting the interest points, SIFT rotates the descriptor to align with the dominant gradient orientation. Then, gradient histograms are computed for local sub-regions of the descriptor which are concatenated and normalized to form a 128D feature vector (the keypoint descriptor). These operations make the descriptor invariant to rotation and brightness changes.

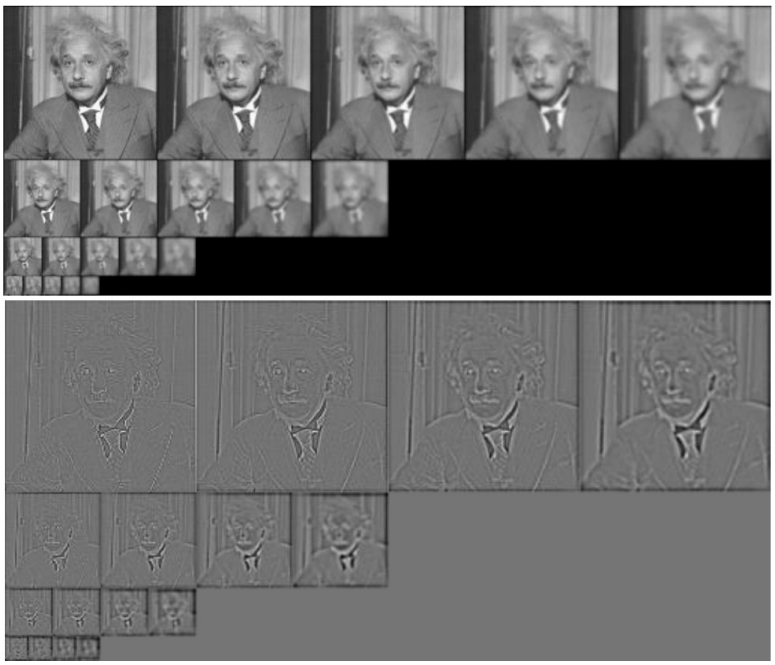


Figure 16: Illustration of the scale space constructed with the SIFT algorithm

By now, many algorithms for feature detection and description have been developed, e.g., SIFT, SURF, U-SURF, BRISK, ORB, FAST, and recently deep learning based ones. SIFT was a seminal work due to

its invariance and robustness, which revolutionized recognition and in particular matching and enabled the development of large-scale SfM techniques. Despite being over 20 years old, it is still used today (e.g., in the SfM pipeline COLMAP). The feature vectors constructed by SIFT can be used to find matching points in two or more images by efficient nearest neighbor search algorithms. Ambiguous matches are typically filtered by computing the ratio of distance from the closest neighbor to the distance of the second closest; a large ratio (> 0.8) indicates that the found match might not be the correct one. An example of this matching between two images that were taken from different viewpoints is shown in Figure 17.

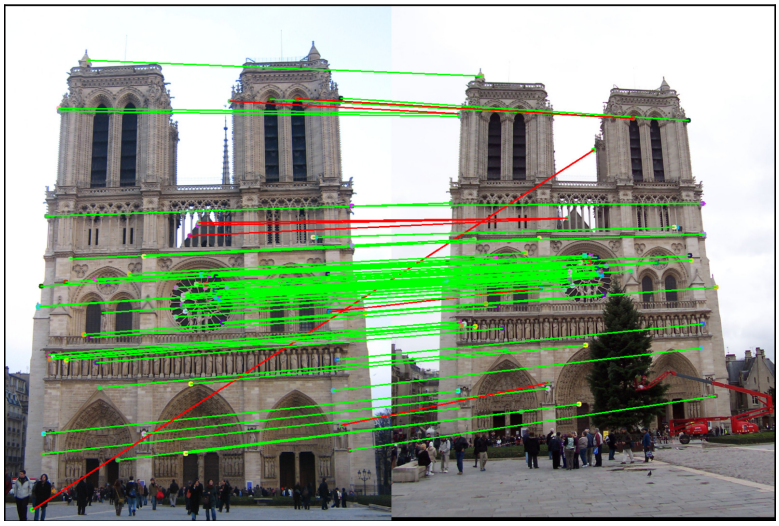


Figure 17: Feature points matched between two images, recognized using SIFT. Green means correct match, red means erroneous match.

3.2 Two-frame Structure-from-Motion

3.2.1 Epipolar Geometry

In two-frame SfM, we have two different images of an object. The goal is to recover the 3D structure of the object, as well as the camera poses from which the images were taken, solely from the images and detected features thereon. The required relationships for this task are described by the two-view epipolar geometry, illustrated in Figure 18.

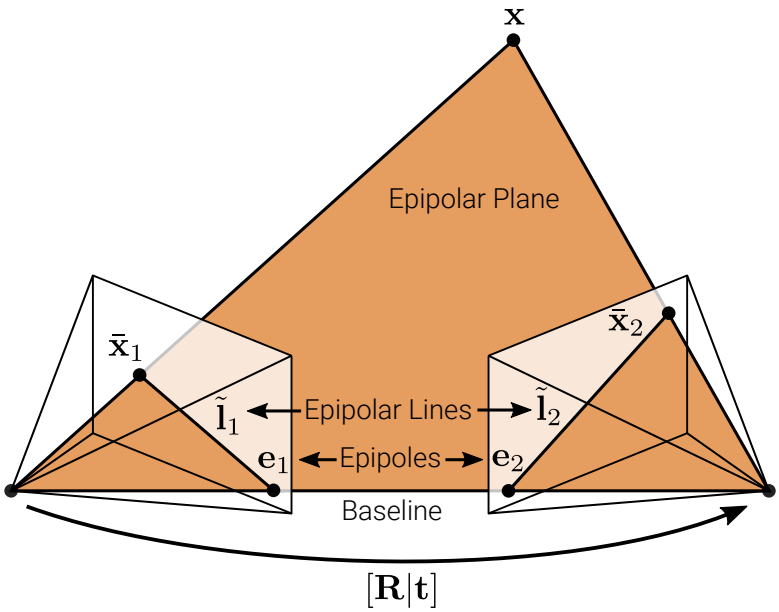


Figure 18: Epipolar geometry

In the illustration, the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ denote the relative pose between two perspective cameras. A 3D point $\mathbf{x}$ is projected to pixel $\bar{\mathbf{x}}_1$ in image 1 and to pixel $\bar{\mathbf{x}}_2$ in image 2. The 3D

point $\mathbf{x}$ and the two camera centers span the *epipolar plane*, on which also the two points $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$ lie. The correspondence of pixel $\bar{\mathbf{x}}_1$ in image 2 must lie on the *epipolar line* $\tilde{\mathbf{l}}_2$ in image 2. This means that, if the epipolar plane is known (i.e. $\mathbf{R}$ and $\mathbf{t}$ as well as camera intrinsics are known), to find the matching point $\bar{\mathbf{x}}_2$ to $\bar{\mathbf{x}}_1$, one only has to search along a one-dimensional search space (the epipolar line $\tilde{\mathbf{l}}_2$). Finally, all epipolar lines must pass though the *epipoles*, which are the points on the image planes where the baseline (the connection of the two camera centers) passes though the image planes (this point can also lie at infinity).

Estimate epipolar geometry

The challenge now is to estimate the epipolar geometry based on detected matching features in the two images. As noted above, the inverse problem (matching features when knowing the epipolar geometry) is harder. Let us assume the camera matrices $(\mathbf{K}_i \in \mathbb{R}^{3\times3})_{i=1}^2$ are known, for example through calibration (Sec. 3.1.1). Let $\tilde{\mathbf{x}}_i = \mathbf{K}_i^{-1}\,\bar{\mathbf{x}}_i$ denote the local ray direction of pixel $\bar{\mathbf{x}}_i$ in camera i. Then we have:

$$\tilde{\mathbf{x}}_2 \;\propto\; \mathbf{x}_2 \;=\; \mathbf{R}\,\mathbf{x}_1 + \mathbf{t} \;\propto\; \mathbf{R}\,\tilde{\mathbf{x}}_1 + s\,\mathbf{t}$$

Taking the cross product of both sides with $\mathbf{t}$ we obtain:

$$[\mathbf{t}]_\times \tilde{\mathbf{x}}_2 \;\propto\; [\mathbf{t}]_\times \mathbf{R}\,\tilde{\mathbf{x}}_1$$

Taking the dot product of both sides with $\tilde{\mathbf{x}}_2^\top$ yields (triple product):

$$\tilde{\mathbf{x}}_2^\top [\mathbf{t}]_\times \mathbf{R}\,\tilde{\mathbf{x}}_1 \;\propto\; \tilde{\mathbf{x}}_2^\top [\mathbf{t}]_\times \tilde{\mathbf{x}}_2 = 0 \quad\Rightarrow\quad \tilde{\mathbf{x}}_2^\top [\mathbf{t}]_\times \mathbf{R}\,\tilde{\mathbf{x}}_1 = 0$$

We arrive at the epipolar constraint:

$$\tilde{\mathbf{x}}_2^\top \tilde{\mathbf{E}}\,\tilde{\mathbf{x}}_1 = 0$$

with *essential matrix*

$$\tilde{\mathbf{E}} = [\mathbf{t}]_\times \mathbf{R}.$$

We can see that $\tilde{\mathbf{E}}$ maps a point $\tilde{\mathbf{x}}_1$ in image 1 to the corresponding epipolar line in image 2:

$$\tilde{\mathbf{l}}_2 = \tilde{\mathbf{E}}\,\tilde{\mathbf{x}}_1,$$

as $\tilde{\mathbf{x}}_2^\top \tilde{\mathbf{l}}_2 = 0$. Similarly, by transposition, we obtain the epipolar line in image 1:

$$\tilde{\mathbf{l}}_1 = \tilde{\mathbf{E}}^\top \tilde{\mathbf{x}}_2.$$

Now, for any point $\tilde{\mathbf{x}}_1$ in the first image, the corresponding epipolar line $\tilde{\mathbf{l}}_2 = \tilde{\mathbf{E}}\,\tilde{\mathbf{x}}_1$ in the second image passes through the so-called epipole $\tilde{\mathbf{e}}_2$ which therefore satisfies

$$\tilde{\mathbf{e}}_2^\top \tilde{\mathbf{l}}_2 = \tilde{\mathbf{e}}_2^\top \tilde{\mathbf{E}}\,\tilde{\mathbf{x}}_1 = 0.$$

It follows that:

$$\tilde{\mathbf{e}}_2^\top \tilde{\mathbf{E}} = \mathbf{0}.$$

Thus, $\tilde{\mathbf{e}}_2^\top$ is the left null-space (left singular vector with singular value 0) of $\tilde{\mathbf{E}}$. Further:

$$\tilde{\mathbf{E}}\,\tilde{\mathbf{e}}_1 = \mathbf{0}$$

Thus, $\tilde{\mathbf{e}}_1^\top$ is the right null-space (right singular vector with singular value 0) of $\tilde{\mathbf{E}}$.

Now we come to how we can actually estimate the epipolar geometry. We can recover the essential matrix $\tilde{\mathbf{E}}$ from N image correspondences forming N homogeneous equations in the nine elements of $\tilde{\mathbf{E}}$, using the epipolar constraint from above:

$$\begin{array}{ccccccc} x_1\,x_2\,e_{11} & + & y_1\,x_2\,e_{12} & + & x_2\,e_{13} & & \\ x_1\,y_2\,e_{21} & + & y_1\,y_2\,e_{22} & + & y_2\,e_{23} & & \\ x_1\,e_{31} & + & y_1\,e_{32} & + & e_{33} & = & 0. \end{array}$$

As $\tilde{\mathbf{E}}$ is homogeneous, we use singular value decomposition to constrain the scale. This algorithm is also called the 8-point algorithm, because to solve this system of equations, we would need 8 points (because $\tilde{\mathbf{E}}$ is a 3-by-3 matrix that is defined up to scale). However, the essential matrix actually has only 5 DoF (3 for rotation $\mathbf{R}$, 2 for translation direction $\hat{\mathbf{t}}$), and there exist algorithms that need fewer than 8 points. Note that some terms are products of two image measurements and hence amplify measurement noise asymmetrically. Thus, the *normalized 8-point algorithm* whitens the observations to have zero-mean and unit variance before the calculation and back-transforms the matrix recovered by SVD accordingly. From $\tilde{\mathbf{E}}$, we can now also recover the direction $\hat{\mathbf{t}}$ of the translation vector $\mathbf{t}$. Only the direction can be recovered, because the scale is not uniquely determined. We have:

$$\hat{\mathbf{t}}^\top \tilde{\mathbf{E}} = \hat{\mathbf{t}}^\top [\mathbf{t}]_\times \mathbf{R} = \mathbf{0}$$

Thus, $\tilde{\mathbf{E}}$ is singular and we obtain $\hat{\mathbf{t}}$ as the left singular vector associated with singular value 0. In practice the singular value will not be exactly 0 due to measurement noise, and we choose the smallest one. The other two singular values are roughly equal. The rotation matrix $\mathbf{R}$ can also be calculated, cf. [51, Section 11.3 (p. 683)].

Up to now, we assumed the calibration matrices $(\mathbf{K}_i \in \mathbb{R}^{3\times3})_{i=1}^2$ are known. What do we do in scenarios, where we know neither the extrinsics, nor the intrinsics? We cannot use the local ray directions $\tilde{\mathbf{x}}_i = \mathbf{K}_i^{-1}\,\bar{\mathbf{x}}_i$ as we only know the pixel coordinates $\bar{\mathbf{x}}_i$. The essential matrix becomes

$$\tilde{\mathbf{x}}_2^\top \tilde{\mathbf{E}}\,\tilde{\mathbf{x}}_1 = \bar{\mathbf{x}}_2^\top \mathbf{K}_2^{-\top}\tilde{\mathbf{E}}\,\mathbf{K}_1^{-1}\bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2^\top \tilde{\mathbf{F}}\bar{\mathbf{x}}_1 = 0,$$

where $\tilde{\mathbf{F}} = \mathbf{K}_2^{-\top}\tilde{\mathbf{E}}\,\mathbf{K}_1^{-1}$ is called the *fundamental matrix*. Like $\tilde{\mathbf{E}}$, $\tilde{\mathbf{F}}$ is (in absence of noise) rank two and the epipoles can be recovered in the same way. However, the intrinsic parameters cannot be directly determined, i.e., we obtain only a perspective reconstruction and not a metric one. Additional information like vanishing points, a constancy of $\mathbf{K}$ across time, zero skew or an aspect ratio can be used to upgrade a perspective reconstruction to a metric one. In Figure 19, epipolar lines of two real images based on estimated epipolar geometry are shown. Note that corresponding points lie on the corresponding epipolar lines.

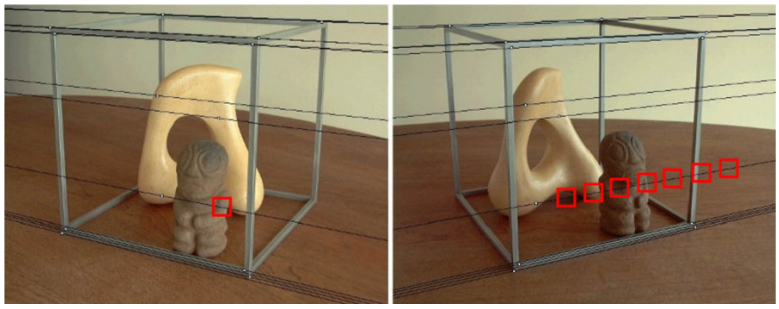


Figure 19: Demo of epipolar geometry of real images. The epipolar lines are shown in black. Features along an epipolar line in one image mus lie on the corresponding epipolar line in the other image.

Triangulation

Now that the camera intrinsics and extrinsics are known, how can we reconstruct the 3D points? In principle, this is easy: we just have to check where the two rays $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ intersect, as shown in Figure 28. However, due to measurement errors, the rays might not exactly intersect. So we want to find the point $\mathbf{x}$ that is closest to both rays.

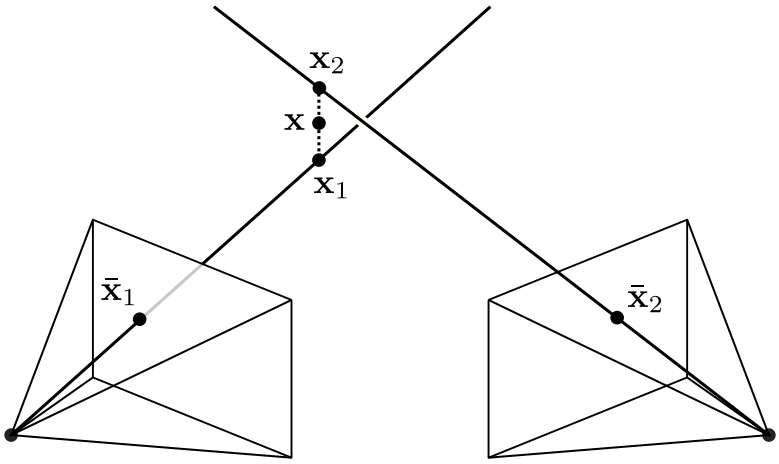


Figure 20: Triangulation setup

Let $\tilde{\mathbf{x}}_i^s = \tilde{\mathbf{P}}_i\,\tilde{\mathbf{x}}_w$ denote the projection of a 3D world point $\tilde{\mathbf{x}}_w$ onto the image of the ith camera $\tilde{\mathbf{x}}_i^s$. As both sides are homogeneous, they have the same direction but may differ in magnitude. To account for this, we consider the cross product $\tilde{\mathbf{x}}_i^s \times \tilde{\mathbf{P}}_i\,\tilde{\mathbf{x}}_w = \mathbf{0}$. Using $\tilde{\mathbf{p}}_{ik}^\top$ to denote the kth row of the ith camera's projection matrix $\tilde{\mathbf{P}}_i$, we obtain:

$$\underbrace{\begin{bmatrix} x_i^s\,\tilde{\mathbf{p}}_{i3}^\top - \tilde{\mathbf{p}}_{i1}^\top \\ y_i^s\,\tilde{\mathbf{p}}_{i3}^\top - \tilde{\mathbf{p}}_{i2}^\top \end{bmatrix}}_{\mathbf{A}_i} \tilde{\mathbf{x}}_w = \mathbf{0}$$

Stacking $N \geq 2$ observations of a point, we obtain a linear system $\mathbf{A}\tilde{\mathbf{x}}_w = \mathbf{0}$. As $\tilde{\mathbf{x}}_w$ is homogeneous this leads to a constrained least squares problem. The solution to this problem is the right singular vector corresponding to the smallest singular value of $\mathbf{A}$. This is the Direct Linear Transformation (DLT) we are already familiar with from Lecture 2. While this approach often works well, it is not invariant to perspective transformations. The gold standard is to minimize the reprojection error using numerical methods:

$$\bar{\mathbf{x}}_w^* = \underset{\bar{\mathbf{x}}_w}{\operatorname{argmin}} \sum_{i=1}^{N} \|\bar{\mathbf{x}}_i^s(\bar{\mathbf{x}}_w) - \bar{\mathbf{x}}_i^o\|_2^2 \qquad \text{with observation } \bar{\mathbf{x}}_i^o.$$

This allows to take measurement noise appropriately into account. The minimum can also be obtained in closed form as the solution of a sixth degree polynomial (see [22, Section 12.5]).

Triangulation works differently well depending on the relative camera pose. This is illustrated in Figure 21. Note that the shaded region increases as the rays become more parallel (i.e. the camera poses become more similar). This means that there is a trade-off here: Feature matching becomes easier the closer the camera poses are, but triangulation becomes harder and vice-versa.

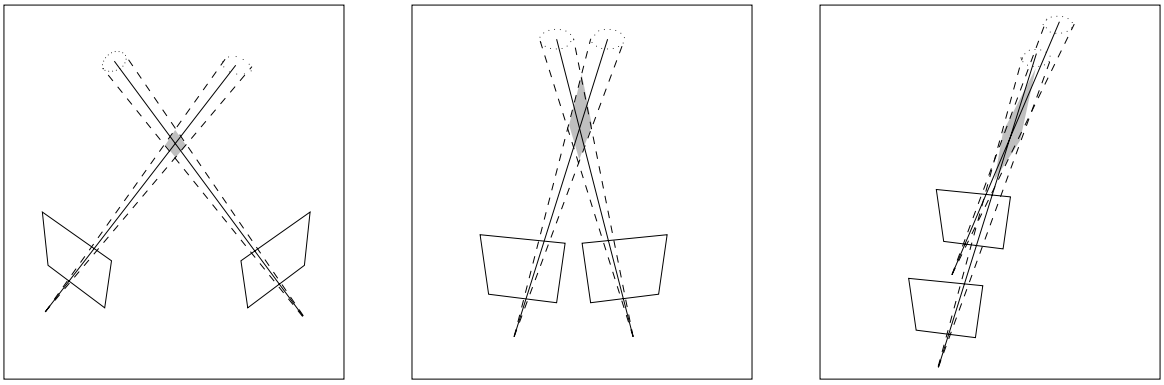


Figure 21: Triangulation is more accurate if the image planes are at a larger angle.

3.3 Factorization

Up to now, we have only talked about using two views to reconstruct 3D-geometry. Intuitively, we should get more accurate results if we use more than two views. We will talk about that approach in this section as well as the next.

Let $\mathcal{W} = \{(x_{ip}, y_{ip}) \,|\, i = 1, \ldots, N,\, p = 1, \ldots, P\}$ denote P feature points tracked over N frames. Given $\mathcal{W}$ and assuming orthographic projection our goal is to recover both camera motion (rotation) and the structure (3D points $\mathbf{x}_p$ corresponding to (x_{ip}, y_{ip})). We assume that all feature points are visible in all frames. The setup is visualized in Figure 22.

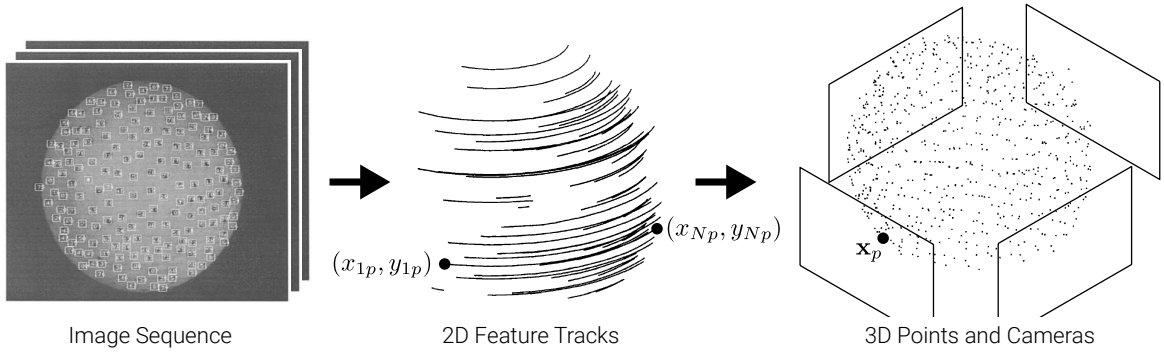


Figure 22: Orthographic projection problem setup.

Under orthographic projection, a 3D point $\mathbf{x}_p$ maps to a pixel (x_{ip}, y_{ip}) in frame i as:

$$x_{ip} = \mathbf{u}_i^\top (\mathbf{x}_p - \mathbf{t}_i),$$
$$y_{ip} = \mathbf{v}_i^\top (\mathbf{x}_p - \mathbf{t}_i).$$

This is illustrated in Figure 23. Without loss of generality, we assume that the 3D coordinate system is at the center:

$$\frac{1}{P}\sum_{p=1}^{P}\mathbf{x}_p = 0.$$

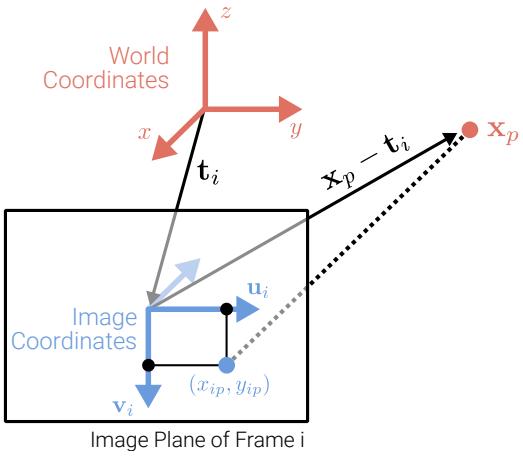


Figure 23

Let (x_{ip}, y_{ip}) denote the 2D location of feature p in frame i. Centering the features per frame (zero-mean) and collecting them yields the *centered measurement matrix* $\tilde{\mathbf{W}}$:

$$\tilde{\mathbf{W}} = \begin{bmatrix} \tilde{x}_{11} & \dots & \tilde{x}_{NP} \\ \vdots & & \vdots \\ \tilde{x}_{N1} & \dots & \tilde{x}_{NP} \\ \tilde{y}_{11} & \dots & \tilde{y}_{NP} \\ \vdots & & \vdots \\ \tilde{y}_{N1} & \dots & \tilde{y}_{NP} \end{bmatrix},$$

where

$$\tilde{x}_{ip} = x_{ip} - \frac{1}{P}\sum_{q=1}^{P} x_{iq}$$

$$\tilde{y}_{ip} = y_{ip} - \frac{1}{P}\sum_{q=1}^{P} y_{iq}.$$

Here, the $\tilde{}$ notation denotes centered, not homogeneous coordinates. As $x_{ip} = \mathbf{u}_i^\top(\mathbf{x}_p - \mathbf{t}_i)$, the centered image x-coordinate is given by:

$$\begin{aligned}
\tilde{x}_{ip} &= x_{ip} - \frac{1}{P}\sum_{q=1}^{P} x_{iq} \\
&= \mathbf{u}_i^\top(\mathbf{x}_p - \mathbf{t}_i) - \frac{1}{P}\sum_{q=1}^{P}\mathbf{u}_i^\top(\mathbf{x}_q - \mathbf{t}_i) \\
&= \mathbf{u}_i^\top(\mathbf{x}_p - \mathbf{t}_i) - \mathbf{u}_i^\top\frac{1}{P}\sum_{q=1}^{P}\mathbf{x}_q + \mathbf{u}_i^\top\mathbf{t}_i \\
&= \mathbf{u}_i^\top\left(\mathbf{x}_p - \frac{1}{P}\sum_{q=1}^{P}\mathbf{x}_q\right) = \mathbf{u}_i^\top\,\mathbf{x}_p.
\end{aligned}$$

We obtain a similar equation for centered image y-coordinate, thus we have:

$$\tilde{x}_{ip} = \mathbf{u}_i^\top\,\mathbf{x}_p, \qquad \tilde{y}_{ip} = \mathbf{v}_i^\top\,\mathbf{x}_p.$$

The centered measurement matrix $\tilde{\mathbf{W}}$ can therefore be expressed as follows:

$$\tilde{\mathbf{W}} = \mathbf{R}\,\mathbf{X} \qquad \text{with} \qquad \tilde{\mathbf{W}} = \begin{bmatrix} \tilde{x}_{11} & \dots & \tilde{x}_{1P} \\ \vdots & & \vdots \\ \tilde{x}_{N1} & \dots & \tilde{x}_{NP} \\ \tilde{y}_{11} & \dots & \tilde{y}_{1P} \\ \vdots & & \vdots \\ \tilde{y}_{N1} & \dots & \tilde{y}_{NP} \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} \mathbf{u}_1^\top \\ \vdots \\ \mathbf{u}_N^\top \\ \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_N^\top \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_P \end{bmatrix}.$$

Here, $\mathbf{R}$ represents the camera motion (rotation), and $\mathbf{X}$ the structure of the 3D scene. As $\mathbf{R} \in \mathbb{R}^{2N\times 3}$ and $\mathbf{X} \in \mathbb{R}^{3\times P}$, in the absence of noise, the centered measurement matrix $\tilde{\mathbf{W}}$ has at most rank 3. When adding noise, the matrix becomes full rank (denoted $\hat{\mathbf{W}}$. We can recover the rank-3 matrix $\tilde{\mathbf{W}}$ by doing a low-rank approximation of $\hat{\mathbf{W}}$ using singular value decomposition:

$$\hat{\mathbf{W}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top.$$

We obtain the rank 3 factorization by considering the singular vectors corresponding to the top 3 singular values (the others should be small and capture noise primarily):

$$\underbrace{\hat{\mathbf{R}}}_{\mathbb{R}^{2N\times 3}} = \underbrace{\mathbf{U}}_{\mathbb{R}^{2N\times 3}}\ \underbrace{\mathbf{\Sigma}^{\frac{1}{2}}}_{\mathbb{R}^{3\times 3}} \qquad \underbrace{\hat{\mathbf{X}}}_{\mathbb{R}^{3\times P}} = \underbrace{\mathbf{\Sigma}^{\frac{1}{2}}}_{\mathbb{R}^{3\times 3}}\ \underbrace{\mathbf{V}^\top}_{\mathbb{R}^{3\times P}}.$$

This approach minimizes the distance in Frobenius norm between the two matrices $\|\hat{\mathbf{W}} - \tilde{\mathbf{W}}\|_F$, while making sure that $\tilde{\mathbf{W}}$ has rank 3. However, this decomposition is not unique as there exists a matrix $\mathbf{Q} \in \mathbb{R}^{3\times 3}$ such that

$$\hat{\mathbf{W}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \hat{\mathbf{R}}\,\hat{\mathbf{X}} = (\hat{\mathbf{R}}\mathbf{Q})\,(\mathbf{Q}^{-1}\hat{\mathbf{X}}).$$

To find $\mathbf{Q}$ we observe that the rows of $\mathbf{R}$ are unit vectors and the first half are orthogonal to the corresponding second half of $\mathbf{R}$. We obtain the metric constraints:

$$\begin{aligned} \hat{\mathbf{u}}_i^\top \mathbf{Q}\,(\hat{\mathbf{u}}_i^\top \mathbf{Q})^\top &= \hat{\mathbf{u}}_i^\top \mathbf{Q}\,\mathbf{Q}^\top \hat{\mathbf{u}}_i = 1, \\ \hat{\mathbf{v}}_i^\top \mathbf{Q}\,(\hat{\mathbf{v}}_i^\top \mathbf{Q})^\top &= \hat{\mathbf{v}}_i^\top \mathbf{Q}\,\mathbf{Q}^\top \hat{\mathbf{v}}_i = 1, \\ \hat{\mathbf{u}}_i^\top \mathbf{Q}\,(\hat{\mathbf{v}}_i^\top \mathbf{Q})^\top &= \hat{\mathbf{u}}_i^\top \mathbf{Q}\,\mathbf{Q}^\top \hat{\mathbf{v}}_i = 0. \end{aligned}$$

This gives us a large set of linear equations for the entries in the matrix $\mathbf{Q}\,\mathbf{Q}^\top$, from which the matrix $\mathbf{Q}$ can be recovered using standard Cholesky decomposition. Thus, the final algorithm for recovering both the camera motion as well as the 3D structure is the following:

1. Take measurements $\hat{\mathbf{W}}$
2. Compute SVD $\hat{\mathbf{W}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and keep the top 3 SVs
3. Define $\hat{\mathbf{R}} = \mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}$ and $\hat{\mathbf{X}} = \mathbf{\Sigma}^{\frac{1}{2}}\mathbf{V}^\top$
4. Compute $\mathbf{Q}\,\mathbf{Q}^\top$ and from this $\mathbf{Q}$
5. Compute $\mathbf{R} = \hat{\mathbf{R}}\mathbf{Q}$ and $\mathbf{X} = \mathbf{Q}^{-1}\hat{\mathbf{X}}$

An advantage of this algorithm is that it is very fast because it has a closed-form solution (which is determined up to an arbitrary global rotation); there are no local minima. A disadvantage is that complete feature tracks are required, which means that it cannot handle occlusions of feature points. This problem can however be solved by applying this basic algorihm to subsets of features and frames and propagating the information to the other frames by a sort of matrix completion (see [53, Sec. 5]).

As mentioned above, this approach assumes orthography. Therefore there have been a couple of extensions of this algorithm, such as [8] that performs an initial orthographic reconstruction and then correct perspective in an iterative manner or [55], that performs projective factorization, iteratively updating depth. Even though these approaches are inaccurate, factorization methods can provide a good initialization for iterative techniques such as bundle adjustment. However, modern SfM approaches (e.g., COLMAP) often perform incremental bundle adjustment, initializing with a carefully selected two-view reconstruction and iteratively adding new images/cameras to the reconstruction.

3.4 Bundle Adjustment

Bundle adjustment is the gold standard algorithm for estimating 3D structure and camera pose from multiple images. It minimizes the reprojection error, which is the distance between observed feature and projected 3D point in the image plane. This yields a highly nonlinear and non-convex optimization problem which is tackled by optimizing 3D structure and camera poses jointly.

Let $\Pi = \{\pi_i\}$ denote the N cameras including their intrinsic and extrinsic parameters, let $\mathcal{X}_w = \{\mathbf{x}_p^w\}$ with $\mathbf{x}_p^w \in \mathbb{R}^3$ denote the set of P 3D points in world coordinates and let $\mathcal{X}_s = \{\mathbf{x}_{ip}^s\}$ with $\mathbf{x}_{ip}^s \in \mathbb{R}^2$ denote the image (screen) observations in all i cameras. Bundle adjustment minimizes the reprojection error of all observations, and can be formulated as follows:

$$\Pi^*, \mathcal{X}_w^* = \underset{\Pi, \mathcal{X}_w}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{p=1}^{P} w_{ip} \left\| \mathbf{x}_{ip}^s - \pi_i(\mathbf{x}_p^w) \right\|_2^2.$$

Here, w_{ip} indicates if point p is observed in image i and $\pi_i(\mathbf{x}_p^w)$ is the 3D-to-2D projection of 3D world point $\mathbf{x}_p^w$ onto the 2D image plane of the ith camera, i.e.:

$$\pi_i(\mathbf{x}_p^w) = \begin{pmatrix} \tilde{x}_p^s / \tilde{w}_p^s \\ \tilde{y}_p^s / \tilde{w}_p^s \end{pmatrix} \quad \text{with} \quad \tilde{\mathbf{x}}_p^s = \mathbf{K}_i (\mathbf{R}_i \, \mathbf{x}_p^w + \mathbf{t}_i).$$

$\mathbf{K}_i$ and $[\mathbf{R}_i|\mathbf{t}_i]$ are the intrinsic and extrinsic parameters of π_i, respectively. During bundle adjustment, we optimize $\{(\mathbf{K}_i, \mathbf{R}_i, \mathbf{t}_i)\}$ and $\{\mathbf{x}_p^w\}$ jointly. An illustration of the bundle adjustment problem involving three points and two cameras can be seen in Figure 24.

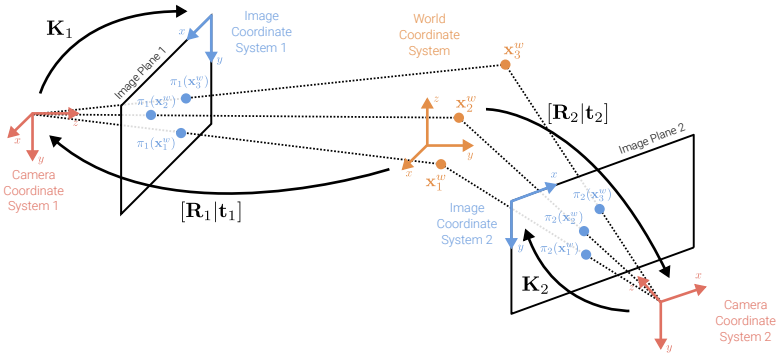


Figure 24: Bundle adjustment problem setting

As the energy landscape of the bundle adjustment problem is highly non-convex, a good initialization is crucial to avoid getting trapped in bad local minima. Initializing all 3D points and cameras jointly is difficult (occlusion, viewpoint, matching outliers), so incremental bundle adjustment is used, which means initializing with a carefully selected two-view reconstruction and iteratively adding new images/cameras. After initialization, the actual optimization is also challenging. Given millions of features and thousands of cameras, large-scale bundle adjustment is computationally demanding (cubic complexity in the number of unknowns). Luckily, the problem is sparse (not all 3D points are observed in every camera), and efficient sparse implementations (e.g., Ceres) can be exploited in practice.

3.4.1 Incremental Structure-from-Motion

In Figure 25 the SfM pipeline COLMAP [48] is shown.

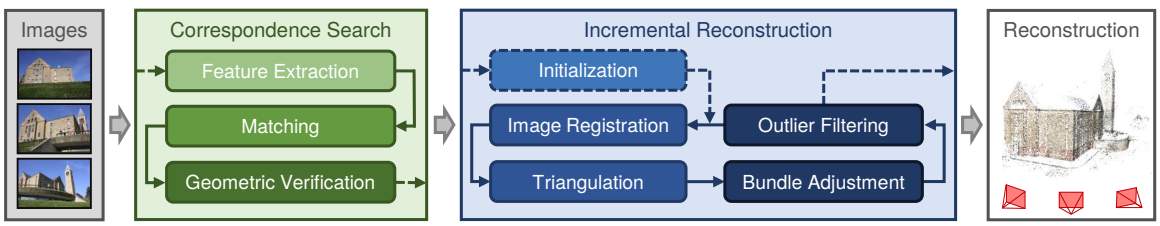


Figure 25: The COLMAP SfM pipeline

The pipeline is divided into two fundamental parts: Correspondence Search and Incremental Reconstruction. Correspondence Search is the stage where 2D features are found and matched in and between the images.

Incremental Reconstruction is the stage where these matched features are used to infer 3D structure and camera parameters. It is incremental in the sense that we start with only two cameras and then add new cameras incrementally.

The correspondence search stage begins with feature extraction in all images. For this, SIFT (discussed in Sec. 3.1.2) or another feature detector such as SURF or BRISK can be used. The next step (matching) is to find overlapping image pairs and associated feature correspondences. The first step of the incremental reconstruction stage (initialization) begins with two images (with matched features). These features are processed as described in Section 3.2 to estimate the epipolar geometry. When this is done, the next step (image registration) starts for the first time and a new image is selected that has correspondences in the current set of images. Then, the camera pose for this new image is estimated. Let $\mathcal{X} = \{\bar{\mathbf{x}}_i^s, \bar{\mathbf{x}}_i^w\}_{i=1}^N$ be a set of N 3D-to-2D correspondences related by $\bar{\mathbf{x}}_i^s = \mathbf{P}\,\bar{\mathbf{x}}_i^w$. As the correspondence vectors are homogeneous, they have the same direction but differ in magnitude. Thus, the equation above can be expressed as $\bar{\mathbf{x}}_i^s \times \mathbf{P}\bar{\mathbf{x}}_i^w = \mathbf{0}$. Using the Direct Linear Transform, this can be written as a linear equation in the entries of $\mathbf{P}$. The solution to the constrained system (fixing $\mathbf{P}$'s scale) is given by SVD. Given that $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ and $\mathbf{K}$ is upper-triangular, both $\mathbf{K}$ and $\mathbf{R}$ can be easily obtained from the front 3×3 submatrix of $\mathbf{P}$ using standard RQ factorization. If $\mathbf{K}$ is known, we can even estimate $\mathbf{P}$ from only three points (P3P algorithm). In practice, random sampling consensus (RANSAC) is additionally used to remove outliers. The next step of the pipeline is triangulation. Given the newly registered image, new correspondences can be triangulated. In COLMAP, a robust triangulation method is proposed that also handles outliers. After this, the bundle adjustment step is performed. Since incremental SfM only affects the model locally, COLMAP performs local BA on the locally connected images, and global BA only once in a while for efficiency. For solving the sparse large-scale optimization problem, COLMAP uses Ceres. After each BA, observations with large reprojection errors and cameras with abnormal field of views or large distortion coefficients are removed. COLMAP also features an additional multi-view stereo stage to obtain dense geometry, a result is shown in Figure 26.



Figure 26: Resulting dense 3D structure from COLMAP when using the additional multi-view stereo stage

4 Stereo Reconstruction

In the previous lecture we learned how to recover sparse 3D structure from two or more views. The topic of this lecture is dense stereo reconstruction: Obtaining a more dense 3D reconstruction from just two images, if the camera intrinsics and extrinsics are known. It is structured into five units.

4.1 Preliminaries

This part of the lecture covers the following two questions:

1. How to bring images in a suitable configuration such that matching is fast?
2. How to obtain depth from the actual measurements (so called *disparities*)?

How to recover 3D from an image?

In general, recovering the 3D structure from an image is an ill-posed problem, but there are several cues:

Occlusion One object being occluded by another object gives information which of them is in front and which is in the back of the scene.

Parallax If the relative position of two objects is different in a mirror reflection, the objects cannot be located at the same depth.

Perspective The behaviour of parallel structures reveals information about the depth.

Acccomodation Objects that are blurred are at a different depth than objects that are in focus.

Stereopsis By using the principle of triangulation, the 3D position of an object can be computed from the movement of the object over different images given the camera positions are known.

Why Binocular Stereopsis?
Binocular stereopsis is an established system as most species have two eyes. Conceptually, it is a minimal configuration to perceive depth relatively robustly. In some cases, this concept has been exploited by artificially enlarging the baseline, i.e. the distance between the cameras, to obtain better depth precision. Two examples for such cases are the stalk-eyed fly and the stereoscopic rangefinder.

4.1.1 Two-View Stereo Matching

Goal: Recovering the disparity for every pixel from the input images. The *disparity* (or inverse depth) is defined as the relative displacement between pixels in the two images.

Task: Construct a dense 3D model from two images of a static scene, e.g. a scene captured with synchronized cameras.

Pipeline:

1. **Calibrate cameras** intrinsically and extrinsically (Lecture 3.1)
2. **Rectify images** given the calibration
3. **Compute disparity map** for reference image
4. **Remove outliers** using consistency/occlusion test
5. **Obtain depth** from disparity using camera calibration
6. **Construct 3D model**, e.g. via volumetric fusion and meshing

4.1.2 3D Reconstruction Pipeline

Given a set of input images, we first compute the camera poses, e.g. using incremental bundle adjustment. The camera poses enable us to compute dense correspondences for each adjacent view via epipolar geometry. Correspondences can be used to recover depth maps. Using depth map fusion, these maps lead to a coherent 3D model that takes all made observations into account. This lecture covers the computation of the correspondences and the depth maps.

4.1.3 Epipolar Geometry

Recap: A 3D point $\mathbf{x}$ is projected to both images as

$$\bar{\mathbf{x}}_1 \propto \tilde{\mathbf{x}}_1 = \mathbf{K}_1\mathbf{x}$$

and

$$\bar{\mathbf{x}}_2 \propto \tilde{\mathbf{x}}_2 = \mathbf{K}_2\mathbf{x}.$$

Image correspondences are related by

$$\tilde{\mathbf{x}}_2^T\tilde{\mathbf{E}}\tilde{\mathbf{x}}_1 = 0$$

with $\tilde{\mathbf{x}}_i = \mathbf{K}_i^{-1}\tilde{\mathbf{x}}_i$. The correspondence of pixel $\mathbf{x}_1$ is located on the epipolar line

$$\tilde{\mathbf{l}}_2 = \tilde{\mathbf{E}}\tilde{\mathbf{x}}_1$$

and the correspondance of pixel $\mathbf{x}_2$ is located on the epipolar line

$$\tilde{\mathbf{l}}_1 = \tilde{\mathbf{E}}^\top\tilde{\mathbf{x}}_2.$$

All epipolar lines intersect at the epipoles. In this lecture, the property that the correspondences of the pixels must be located on epipolar lines will be exploited to reduce the correspondence search to a much simpler 1D problem. Considering a VGA image, the search space containing all pixels ($\approx 300k$ pixels) reduces to the search space containing only an epipolar line (≈ 640 pixels).

4.1.4 Image Rectification

What if both cameras face exactly the same direction?
If both cameras face exactly the same direction, their image planes will be co-planar and parallel to the baseline. This means that the epipoles lie at infinity (outside the image) and all epipolar lines are parallel to each other and horizontal on the images. In this setting, correspondence search can be done along *horizontal scanlines* which simplifies the implementation.

What if the images are not in the required setup?
We can rewarp the images through rotation, mapping both image planes to a common plane parallel to the baseline. This process is called *rectification* and does not require knowledge about the 3D structure. Similar to the homogrpahy for stitching panoramas, the rotation around the camera center can be computed by estimating a rotation homography.

How can we make epipolar lines horizontal?

For a simplified scenario, assume both calibration matrices and the rotation matrix are equal to the identity matrix,

$$\mathbf{K}_1 = \mathbf{K}_2 = \mathbf{R} = \mathbf{I},$$

and that the translation vector between the two cameras is just a vector in x-direction

$$\mathbf{t} = (t, 0, 0)^T.$$

Now, we are already in the rectified setup. In this case, the essential matrix is given by

$$\tilde{\mathbf{E}} = [t]_x R = [t]_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t \\ 0 & t & 0 \end{bmatrix}.$$

As $\mathbf{K}_i = \mathbf{I}$, two corresponding (augmented) points $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$ are related by the epipolar constraint:

$$\bar{\mathbf{x}}_2^T \tilde{\mathbf{E}} \bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t \\ 0 & t & 0 \end{bmatrix} \bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2^T \begin{pmatrix} 0 \\ -t \\ ty_1 \end{pmatrix} = ty_1 - ty_2 = 0.$$

Thus, $y_1 = y_2$, i.e. the two images of the 3D point $\mathbf{x}$ are on the same horizontal line (or same image row). In order to rectify an image, we calculate a *rectifying rotation matrix* $\mathbf{R}_{\text{rect}} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)^T$ that rotates the cameras virtually such that the above setup is fulfilled, i.e. aligning the x-axis of the camera coordinate system with the translation vector while introducing as little distortion to the other two axes as possible (these must be perpendicular to each other). One simple choice for $\mathbf{R}_{\text{rect}}$ is the following: First, we make sure the new x-axis aligns with the translation vector by setting $\mathbf{r}_1$ to its normalized form (Fig. 27a)

$$\mathbf{r}_1 = \frac{\mathbf{t}}{\|\mathbf{t}\|_2}.$$

Next, we choose $\mathbf{r}_2$ (Fig. 27b) such that the new y-axis is perpendicular to the new x-axis and perpendicular to the old z-axis (this helps to keep the distortion small)

$$\mathbf{r}_2 = [(0, 0, 1)^T] \times \mathbf{r}_1.$$

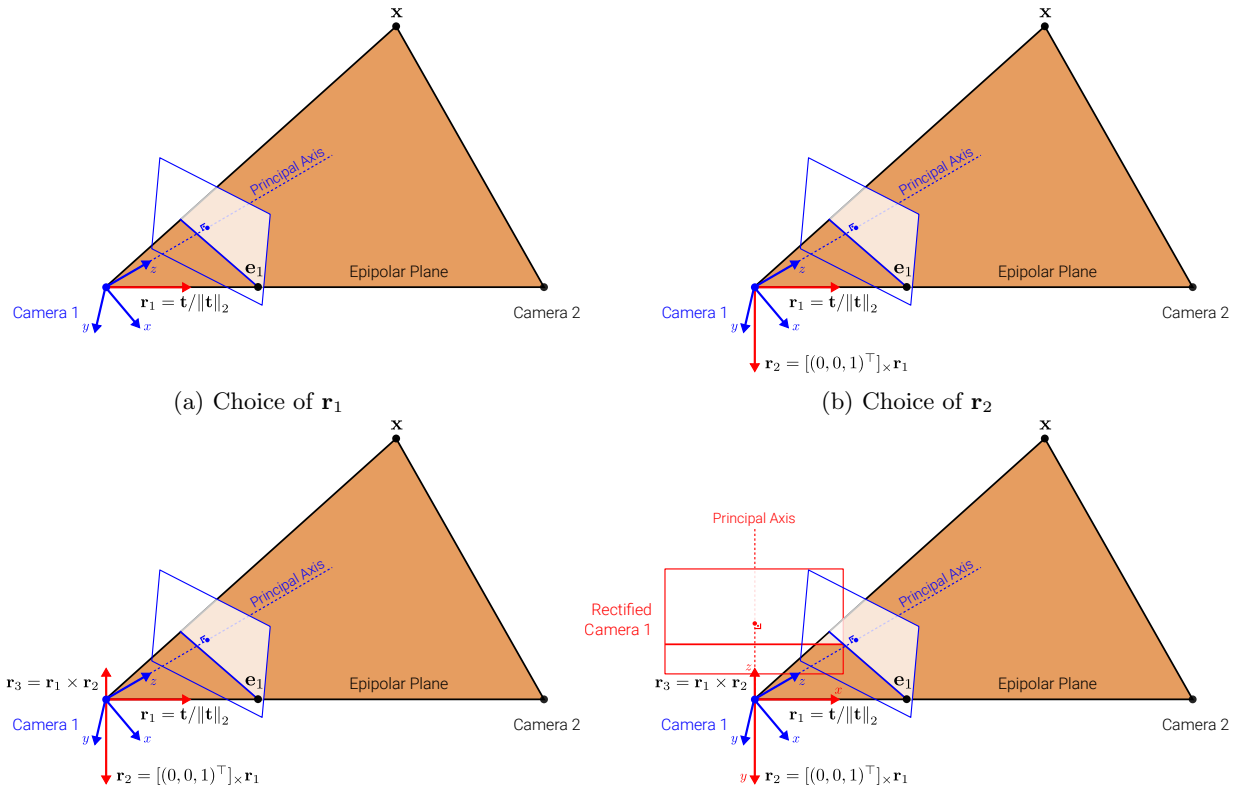
Finally, $\mathbf{r}_3$ (Fig. 27c) should be perpendicular to $\mathbf{r}_1$ and $\mathbf{r}_2$

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2.$$

As the epipole in the first image is in the direction of $\mathbf{r}_1$, it is easy to see that the rotated epipole is ideal, i.e. it lies at infinity, in the x-direction:

$$\mathbf{R}_{\text{rect}} \mathbf{r}_1 = (1, 0, 0)^T.$$

Thus, applying $\mathbf{R}_{\text{rect}}$ to the first camera leads to parallell and horizontal epipolar lines (Fig. 27d).
Rectification algorithm:



(a) Choice of $\mathbf{r}_1$

(b) Choice of $\mathbf{r}_2$

(c) Choice of $\mathbf{r}_3$

(d) Rectified Camera

Figure 27: **Image Rectification**

1. Estimate essential matrix $\tilde{\mathbf{E}}$, decompose into $\mathbf{t}$ and $\mathbf{R}$, and construct $\mathbf{R}_{\text{rect}}$ as above
2. Warp pixels in the first image as follows

$$\tilde{\mathbf{x}}_1' = \mathbf{K}\mathbf{R}_{\text{rect}}\mathbf{K}_1^{-1}\bar{\mathbf{x}}_1$$

3. Warp pixels in the second image as follows

$$\tilde{\mathbf{x}}_2' = \mathbf{K}\mathbf{R}\mathbf{R}_{\text{rect}}\mathbf{K}_2^{-1}\bar{\mathbf{x}}_2$$

Remarks:

- $\mathbf{R}$ is the rotation (extrinsics) matrix between the virtual cameras
- $\mathbf{K}$ is the shared projection matrix that can be chosen arbitrarily (e.g. $\mathbf{K} = \mathbf{K}_1$)
- In practice, the inverse transformation is used for warping. Otherwise transformed pixels might overlap or are stretched such that some pixels in the (virtual) target image are missing. Therefore, one goes over all pixels in the target image and queries the source pixels in the original image using the inverse transformation. The resulting point will not necessarily lie on an integer location. In this case interpolation is used to compute the RGB value for the desired pixel.

4.1.5 Disparity Estimation Example

The relative horizontal displacement is called *disparity* and is anti-proportional to the depth, i.e. far points move less (or have smaller disparity) than closer points. This can be exploited to estimate the depth.

4.1.6 Disparity to Depth

The left and right ray in Fig. 28 must intersect as both lie in the epipolar plane. Assuming disparity $d = x_1 - x_2$ with $x_1 > 0$ and $x_2 < 0$, we have:

$$\begin{aligned} \frac{z-f}{b-d} &= \frac{z}{b} && | \cdot (b-d)b \\ zb - fb &= zb - zd && | -zb + fb + zd \\ zd &= fb && | \cdot \frac{1}{d} \\ z &= \frac{fb}{d} \end{aligned}$$

The depth error Δz grows quadratically with the depth z.

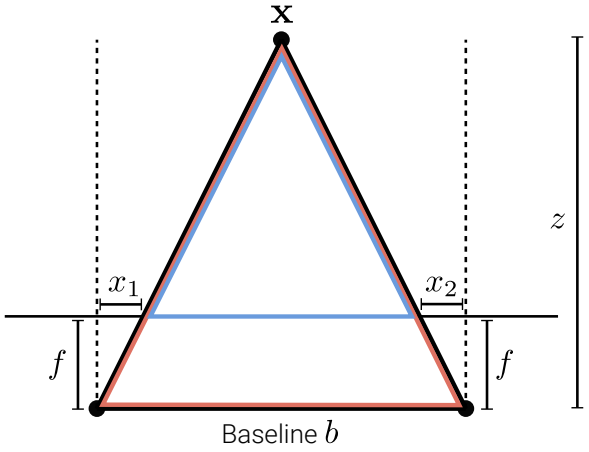


Figure 28: **Recovering Depth Via Triangulation**

4.2 Block Matching

This part of the lecture covers the question:

> How can we determine the disparity values given a rectified setup?

4.2.1 Correspondence Ambiguity

How to determine if two image points correspond?
We assume that two image points correspond to each other if they look "similar". First, we have to define what similar means. Comparing single pixels will lead to ambiguities as they do not reveal the local structure. For example, if if we try to find the correspondence of a red pixel in the left image, there will be too many pixels close to red in the right image that end up producing a noisy result. Therefore, we should compare small image regions or patches. Even then, finding the corresponding patch in the other image remains a difficult task.

4.2.2 Similarity Metrics

As shown in Fig. 29, the pixel in the right image that corresponds to the pixel at x_1 in the left image has to lie on the scanline. To find it, we query all possible patches in the right image and measure similarity to the patch in the left image. Since the disparity is always positive, we only need to consider patches that lie to the left of location x_1.
Consider two $K \times K$ windows of pixels flattened to vectors $\mathbf{w}_L, \mathbf{w}_R \in \mathbb{R}^{K^2}$ in the left and right image respectively. Let $\bar{\mathbf{w}}$ denote the mean of $\mathbf{w}$. Two possible choices for the similarity metric are:

Zero Normalized Cross-Correlation (ZNCC)

$$\mathrm{NCC}(x,y,d) = \frac{(\mathbf{w}_L(x,y) - \bar{\mathbf{w}}_L(x,y))^T (\mathbf{w}_R(x-d,y) - \bar{\mathbf{w}}_R(x-d,y))}{\|\mathbf{w}_L(x,y) - \bar{\mathbf{w}}_L(x,y)\|_2 \, \|\mathbf{w}_R(x-d,y) - \bar{\mathbf{w}}_R(x-d,y)\|_2}$$

Sum of squared differences (SSD)

$$\mathrm{SSD}(x,y,d) = \|\mathbf{w}_L(x,y) - \mathbf{w}_R(x-d,y)\|_2^2$$

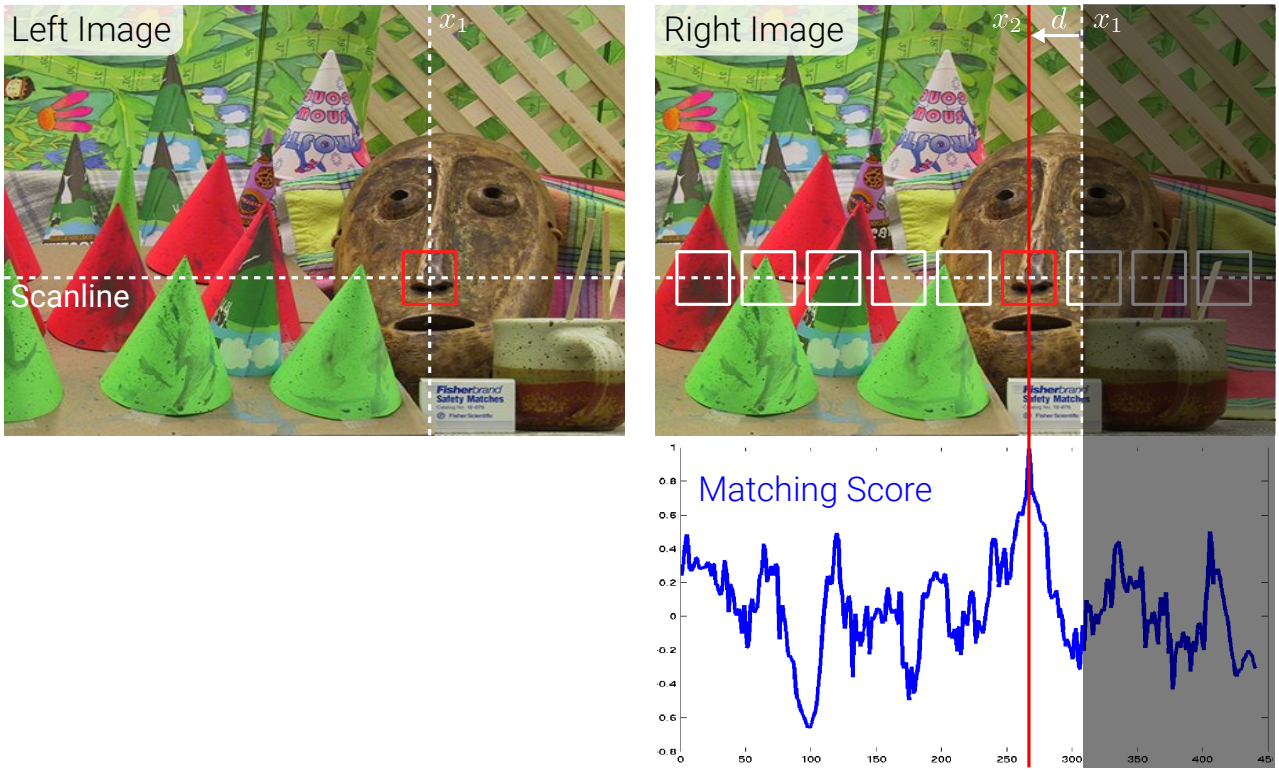


Figure 29: **Block Matching**

Note that in both cases $\mathbf{w}_R$ is moved to the left by subtracting the disparity $d \geq 0$ while keeping the image row y constant.

4.2.3 Block Matching

The block matching procedure works as follows. First, choose a disparity range $[0, D]$. Here, D is the maximum disparity we want to search for and corresponds to how close an object can be to the camera. Then, compute the best disparity for all pixels $\mathbf{x} = (x, y)$ independently (winner-takes-all). After doing this for both images a left-right consistency check is applied to remove outliers.

4.2.4 Half Occlusions

Half occlusions are regions that are only visible in the reference image but not in the target image. Thus, it is not possible to find a correspondence. The red region in Fig. 30 is visible in the left image and occluded in the right image.

4.2.5 Assumption Violations

Block matching assumes that all pixels inside the window are displaced by the same disparity d. This assumption is called *fronto-parallel assumption* and is often violated. As the name suggests, it only holds true for planes that are parallel to the image planes. Slanted surfaces will deform perspectively when the viewpoint changes. Moreover, the window content will change differently in the two images at disparity discontinuities.

4.2.6 Effect of Window Size

The choice of the window size results in a tradeoff. Smaller windows will lead to matching ambiguities and noise in the disparity maps. On the other hand, larger windows, while leading to smoother results, will cause loss of details and border bleeding. Border bleeding happens when the boundary of an object in the disparity map does not align with its boundary in the original image.

4.2.7 Left-Right Consistency Test

A left-right consistency test helps to detect outliers and half-occlusions. After computing the disparity map for both images, we can verify cycle-consistency. This means that, after moving according to the disparity value

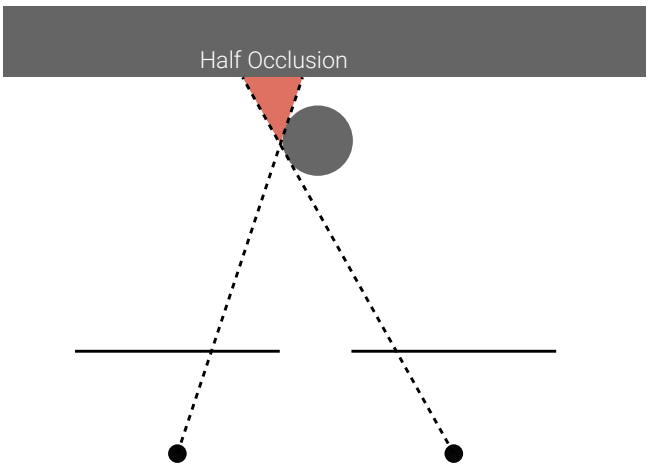


Figure 30: **Half Occlusion**

of a pixel in the first image and then moving back by the disparity value at the shifted location in the second image, we should re-arrive at the original pixel.

4.3 Siamese Networks

This unit covers learning similarity metrics from data using deep neural networks instead of using hand crafted metrics such as SSD or ZNCC.

4.3.1 Siamese Networks for Stereo Matching

Hand crafted metrics do not take into consideration relevant geometric and radiometric invariances or occlusion patterns. Unfortunately, the world is too complex to specify this by hand. Instead, matching cost computation can be treated as a classification problem where pairs of patches from both images are labeled as "good match" or "bad match".

Method Overview

1. Train CNN patch-wise based on images with ground truth disparity maps.
2. Calculate features for each of the two images using learned model.
3. Correlate features between both images (dot product).
4. - Find maximum for every pixel (winner-takes-all).
   or
   - Run global optimization algorithm that incorporates some smoothness assumptions.

4.3.2 Siamese Network Architecture

Fig. 31 shows two different architectures for the Siamese Network model. Both use convolutional layers to extract features from the two input images. The convolutions applied to the left and the right image share their parameters. The *Learned Similarity* model concatenates those features and applies several fully-connected layers (Multilayer Perceptron) to compute a similarity score. On the other hand, the *Cosine Similarity* model just normalizes the output of the convolution blocks and computes the dot product as similarity score. While the former approach has much larger computational cost, their performance is roughly on par.

4.3.3 Training

The training set is composed of patch triplets

$$\left(\mathbf{w}_L(\mathbf{x}_L^{\text{ref}}), \mathbf{w}_R(\mathbf{x}_R^{\text{neg}}), \mathbf{w}_R(\mathbf{x}_R^{\text{pos}})\right)$$

where $\mathbf{w}_L(\mathbf{x}_L)$ is an image patch from the left image centered at $\mathbf{x}_L = (x_L, y_L)$ and $\mathbf{w}_R(\mathbf{x}_R)$ is an image patch from the right image centered at $\mathbf{x}_L = (x_L, y_L)$. Negative and positive examples are created by applying different offsets to true correspondence. For the negative examples,

$$\mathbf{x}_R^{\text{neg}} = (x_L^{\text{ref}} - d + o_{\text{neg}}, y_L)$$

Learned Similarity	Cosine Similarity
Learns features and the similarity metric.	Learns features and applies dot product.
Potentially more expressive.	Features must do the heavy lifting.
Slow ($W \times H \times D$ MLP evaluations)	Fast matching (no network evaluation)

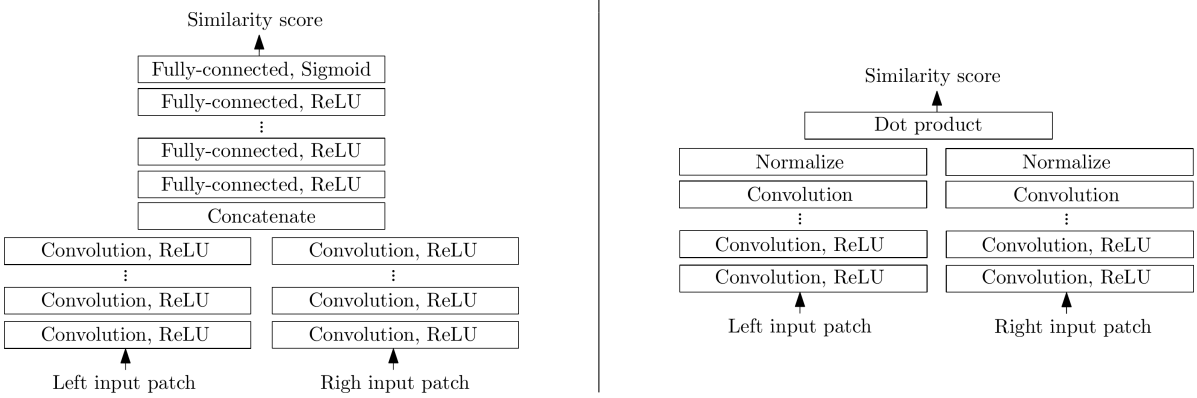


Figure 31: **Siamese Network Architectures** [62]

the reference pixel is shifted by the ground truth disparity d and an offset o_{neg} drawn from

$$\mathcal{U}(\{-N_{\text{hi}}, \ldots, -N_{\text{lo}}, N_{\text{lo}}, \ldots, N_{\text{hi}}\}).$$

For the positive examples

$$\mathbf{x}_R^{\text{pos}} = (x_L^{\text{ref}} - d + o_{\text{pos}}, y_L^{\text{ref}})$$

the offset o_{pos} is drawn from a smaller range

$$\mathcal{U}(\{-P_{\text{hi}}, \ldots, P_{\text{hi}}\}).$$

This training process is called *hard negative mining* and, typically, the hyperparameters are chosen as

$$P_{\text{hi}} = 1, N_{\text{lo}} = 3, \text{ and } N_{\text{hi}} = 6.$$

Note that the negative examples are chosen very close to the positive examples, but far enough for the classifier to classify them correctly. An example is shown in Fig. 32.

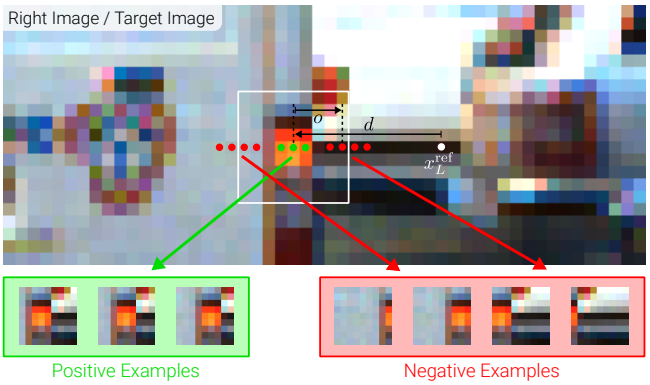


Figure 32: **Positive and Negative Examples**

4.3.4 Loss Function

The loss function used for the training process is the *Hinge Loss*:
Hinge Loss:

$$\mathcal{L} = \max(0, m + s_- - s_+)$$

Here, s_- and s_+ are the scores of the network for the negative and positive example respectively. Therefore, the loss is zero when the similarity of the positive example is greater than the similarity of the negative example by at least margin m (tunable hyperparameter). This prevents the further separation of positive and negative features that are already well separated and gives the model the capacity to focus on the hard cases.

4.3.5 Runtime

The original version of the algorithm was implemented in CUDA andd Lua/Torch 7 and run on a Nvidia GTX Titan GPU. The training took 5 hours. It consisted of 16 epochs of stochastic gradient descent (with batch size 128) over 45 million training examples. Depending on the architecture, evaluation of the neural network for a single pair of images takes 1 second (cosine similarity) or 95 seconds (learned similarity). Adding 3 seconds for semiglobal matching and 1 second for cost aggregation, the inference for as single image pair takes 6 seconds or 100 seconds.

4.4 Spatial Regularization

So far, the matching patches where chosen by winner-takes-all. This does not remove all ambiguities. In this unit we consider a global optimization approach instead.

4.4.1 The Underlying Assumption and Failure Cases

The underlying assumption of the matching process is that corresponding regions in both images should look similar while non-corresponding regions look different. There are several failure cases for this similarity constraint (see Fig. 33).

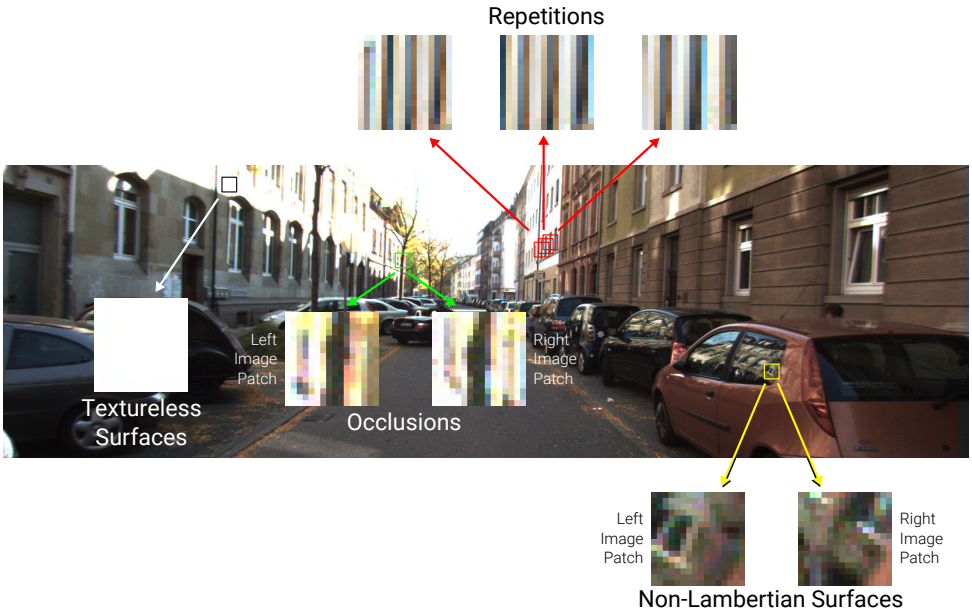


Figure 33: **Failure Cases: Similarity Constrain**

Failure Cases:

Textureless Surfaces Some patches have little or no texture at all. Matching those patches is very hard because there are many possibilities to match them in the other image.

Occlusions Consider the patch (see Fig. 33) with a tree in front of the building. Due to the viewpoint, the background changes between the two images.

Repetitions For repeating structures, many patches will look similar to each other. Thus, matching them becomes ambiguous.

Non-Lambertian Surfaces Strong specular reflections in a patch can result in a complete change of the content in the corresponding patch in the other image.

4.4.2 Spatial Regularization: Idea

The idea of spatial regularization is to overcome ambiguities by leveraging real-world statistics. Considering the Brown range database which consists of depth images of real scenes taken by a lidar sensor, certain patterns emerge. Real scenes contain a lot of smooth surfaces, i.e. the depth changes very slowly for adjacent pixels. The exception are object discontinuities where the depth can vary very quickly from a small value to a large value. But those discontinuities are sparse.

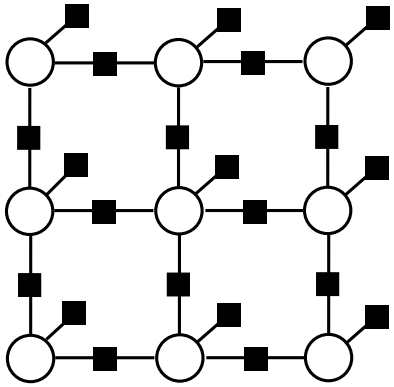


Figure 34: **Factor Graph of the MRF** The nodes represent pixels. The squares represent the constraint. Squares with one connection correspond to a unary term ϕ_{data} and squares with two connections correspond to a pairwise term ϕ_{smooth}.

4.4.3 Stereo MRF [25]

To include smoothness constraints, one can specify a loopy Markov Random Field (Lecture 5) on a grid (see Fig. 34) and solve the whole disparity map $\mathbf{D}$ at once. In this context, one can think of the task as an energy minimization problem because the probability of the disparity map is proportional to a Gibbs distribution:

$$p(\mathbf{D}) \propto \exp\left\{-\sum_i \phi_{\text{data}}(d_i) - \sum_{i\sim j} \phi_{\text{smooth}}(d_i, d_j)\right\}$$

Here, $i \sim j$ denotes all neighboring pixels on a 4-connected grid. The argument of the exponential function can be interpreted as an energy term. Thus, the minimum of energy corresponds to the maximum a posteriori solution. The energy consists of two types of term. The unary terms $\phi_{\text{data}}(d_i)$ are the matching cost of each pixel. Ideally, they should be low for correct and high for incorrect correspondences and are a negative or inverse similarity. Only using these unary terms would end up in the winner-takes-all solution as before because there are no constraints between the pixels. The pairwise terms $\phi_{\text{smooth}}(d_i, d_j)$ constrain the smoothness between adjacent pixels. One choice (Potts) is

$$\phi_{\text{smooth}}(d, d') = [d \neq d'] = \begin{cases} 1 & \text{if } d \neq d' \\ 0 & \text{if } d = d' \end{cases}$$

where we ensure low energy only if the disparity between adjacent pixels does not change. The truncated l_1 constraint takes the relative displacement into account:

$$\phi_{\text{smooth}}(d, d') = \min(|d - d'|, \tau)$$

The resulting MRF is solved approximately using belief propagation.

4.5 End-to-End Learning

This unit covers models that take entire images as input and use a deep neural network to directly output disparity maps. This approach needs much more training data and has higher computational cost than the models covered before.

4.5.1 DispNet

DispNet ([34]) is the first end-to-end trained deep neural network for stereo. It takes a pair of left and right images as input and has a U-net like architecture (see Fig. 35). This includes convolutions, downsampling and

skip-connections to retain details when increasing the resolution again. Note that there is no explicit global optimization.

Specific for this architecture:

Correlation Layer After a few convolutions and pooling layers, a correlation layer (40px displacement corresponding to 160px in input image) is applied to the feature outputs. This layer works similar to the correlation during block matching.

Multi-Scale Loss During training, a multi-scale loss (disparity errors in pixels) is applied with down-scaled versions of the ground truth at intermediate layers.

Curriculum Learning First, the model is trained on simple scenes and small resolutions. Over time, the difficulty is increased until the target dataset is reached.

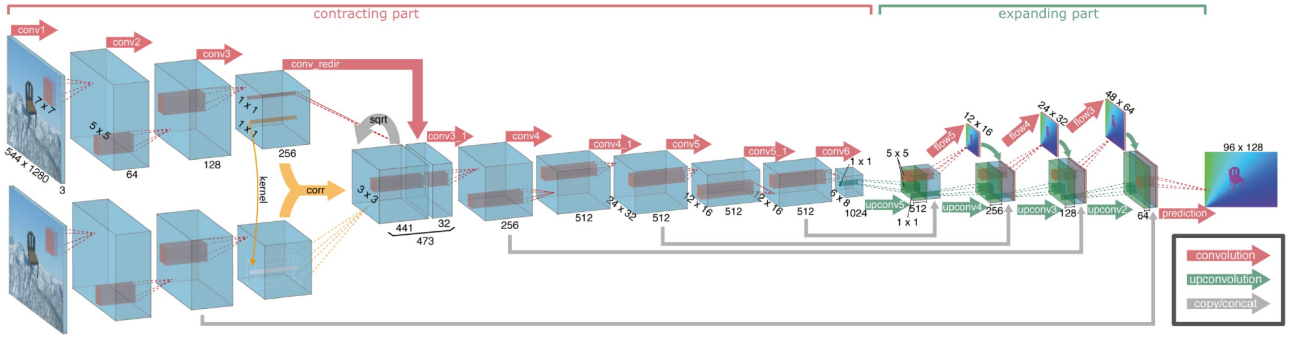


Figure 35: **DispNet Architecture**

4.5.2 Synthetic Datasets

End-to-end models need a large amount of training data and the annotation of real images is very expensive. To avoid this, the model is pretrained on large synthetic datasets (e.g. Flying Things and Monkaa) for which annotation is cheap. Afterwards, the model is finetuned on the much smaller real dataset.

4.5.3 GC-Net

The GC-Net ([28], see Fig. 36) is a follow-up work of DispNet and improves performance even further. The key idea is calculate a disparity cost volume and apply 3D convolutions on it instead of 2D convolutions. It learns a matching cost $c_\Theta(d)$ which can be converted to a disparity via the expectation:

$$d^* = \mathbb{E}[d] = \sum_{d=0}^{D} \underbrace{\text{softmax}(-c_\Theta(d))}_{p(d)} d$$

Although this model has slightly better performance, the memory requirements are larger as well (due to the 3D feature volume).

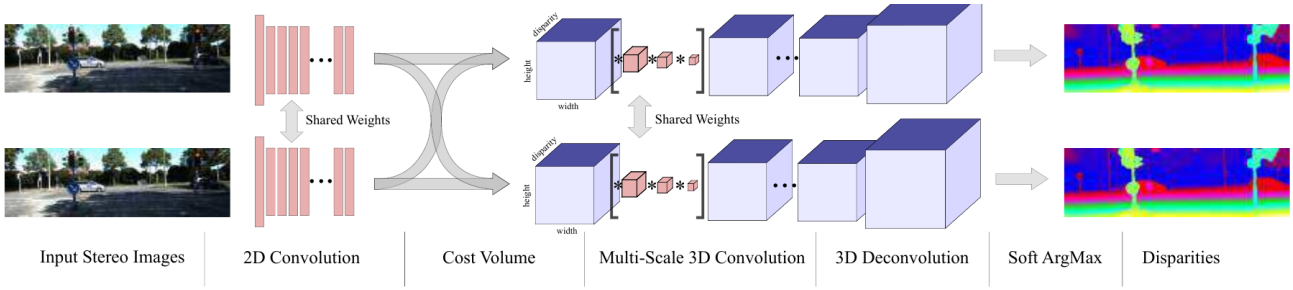
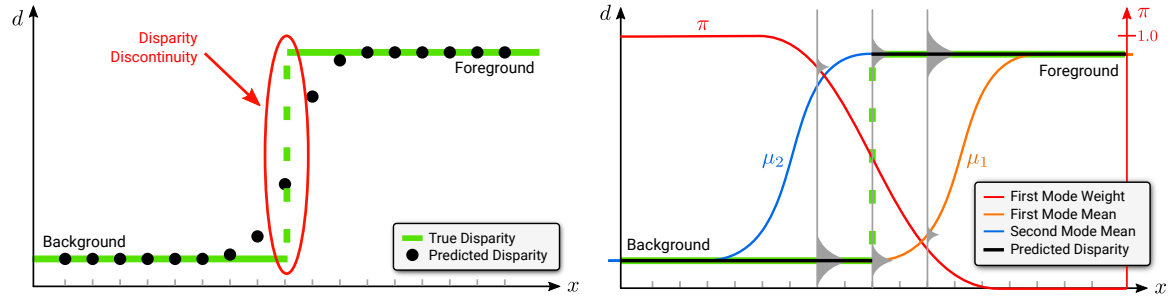


Figure 36: **GC-Net Architecture**

4.5.4 Stereo Mixture Density Networks (SMD-Nets)

In contrast to other architectures, SMD-Nets [54] predict sharper boundaries at higher resolutions. Classical deep networks for stereo regression suffer from a smoothness bias (Fig. 37a) and hence continuously interpolate object boundaries. Instead, SMD-Nets predict a bimodal (Laplacian) mixture distribution (Fig. 37b, depicted in gray) which allows to accurately capture uncertainty close to depth discontinuities. Regarding the architecture (Fig. 38), SMD-Nets use a 2D or 3D convolutional backbone in combination with a shallow MLP head that regresses the distribution parameters from interpolated features. This enables training and inference at arbitrary spatial resolutions.



(a) Classical deep networks for regression.

(b) SMD-Nets predict a bimodal mixture distribution.

Figure 37: **Distinguishing Feature SMD-Nets**

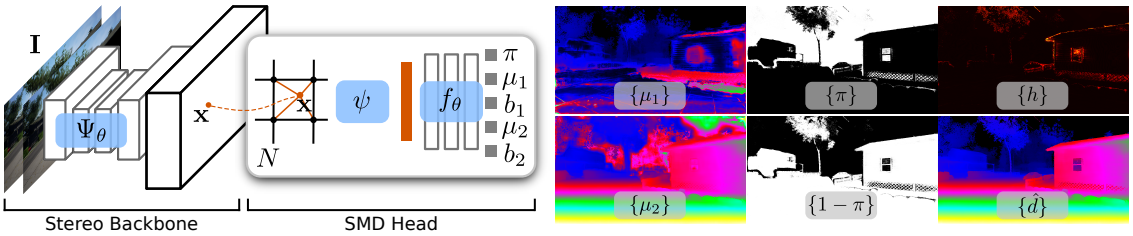


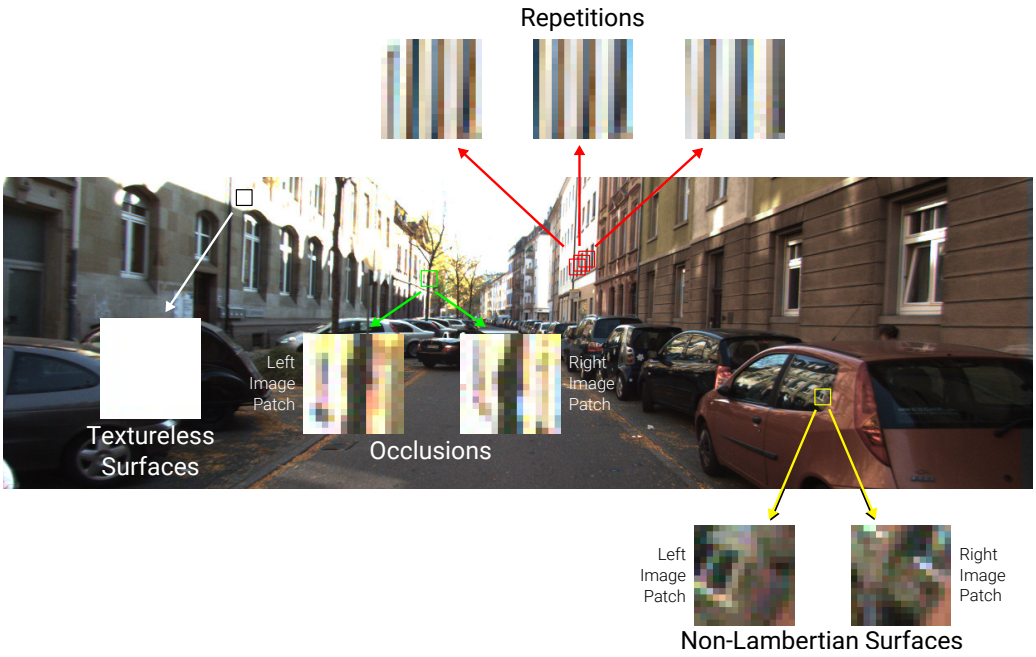
Figure 38: **SMD Architecture**

5 Probabilistic Graphical Models

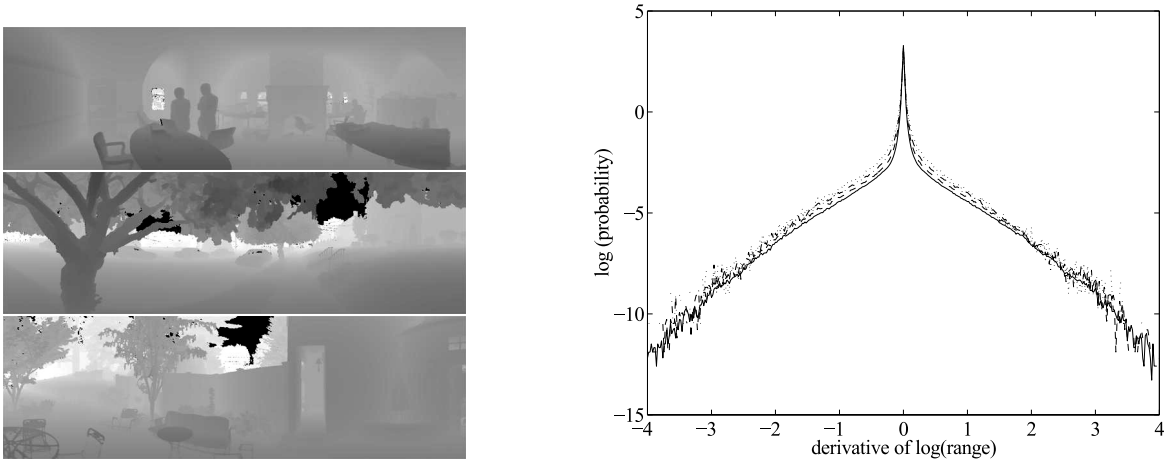
5.1 Structured Prediction

5.1.1 Block Matching Ambiguities

Block matching Assumption: Corresponding regions in both images look similar. When will this similarity constraint fail? Due to the inverse nature of the stereo matching problem there are a lot of ambiguities, that means the correct match between the left and the right image can not be determined without ambiguities. The fundamental assumption of block matching algorithms is, that corresponding regions in the two images should look similar. In a wide range of situation this similarity constraint fail. An example can be seen here:



To overcome such ambiguities we can try to integrate prior knowledge. Prior knowledge in terms of stereo matching means knowing what the world looks like in the terms of depth maps. One example is the brown range image database, from which statistics have been created which show, that depth varies slowly except at object discontinuities. As can be seen in the Distribution the highest probability is around 0 where the derivative



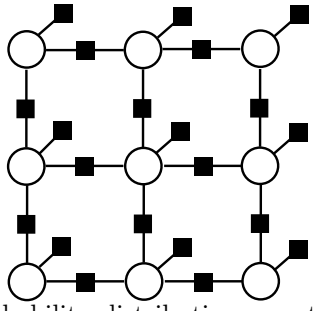
(a) depth maps from the brown range image database (b) Probability distribution of the range derivative

of the range is very small. This is our prior knowledge which we can integrate into the stereo matching.

5.1.2 Spatial Regularization

This Integration is called Spatial Regularization. To achieve this we formulate the Problem as inference in a graphical model where each node corresponds to one pixel an model interactions between adjacent pixels. The adjacent pixels are indicated in the vecor graph on the right:

$$p(\mathbf{D}) \propto \exp\left\{-\sum_i \psi_{data}(d_i) - \lambda \sum_{i\sim j} \psi_{smooth}(d_i, d_j)\right\}$$



This can also be written as an energy function which is the probability distribution over the disparity map. Where i and j are neighboring pixels on the 4 connected grid. What we now want to do is a maximum aposteori

inference over the whole space of disparity maps in order to figure out what is the most likely disparity map given this constraints. In our case the data constraint are the matching costs an the smoothness constraint is our prior knowledge.

5.1.3 Probabilistic Graphical Models

Probabilistic Graphical Models take a probabilistic viewpoint and model the dependency structure of the problem. We try to model the probability distribution over the entire disparity map. And the we want to do inference over this model. This Problem is a structured prediction Problem where we predict all the disparities at once by taking into account this local constraints from before. Graphical models have ruled computer vision before the deep learning revolution. They are Useful in the presence of little training data to integrate prior knowledge. Graphical models can be combined with or can be inform by deep learning.

Pros:

- Integration of prior knowledge
- Few parameters, limited data
- Interpretable models by design

Cons:

- Many phenomena hard to model
- Exploiting large datasets difficult
- Inference often approximate

5.1.4 Structured Prediction

Structure Prediction Models are predicting not just a single pixel disparity but predicting all the disparities at once by taking into account the local constraints and interactions. Structure prediction is in opposition to traditional classification and regression problems.
Classification/Regression: Classification and Regression models a function from complex input space (images, text, audio, sequences of amino acids, ...) and outputs a single number. This number could be either a discrete number (classification) or a continuous number (regression).

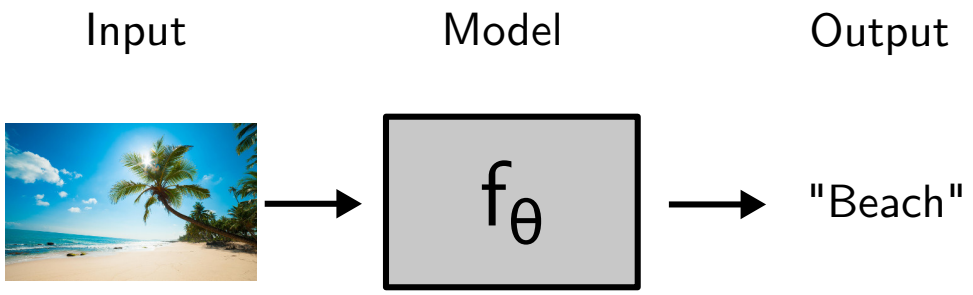


Figure 40: **Classification Example**

Structure Prediction: structure prediction models the same type of inputs as classification and regression but outputs a complex structure. Instead of predicting only a single number, the output can be for example texts, images, sparse trees or folds of a protein. In structure prediction we want to take the structure into account. In the example in Fig. 41 the output would be a three-dimensional quantity with random variables x_1, x_2 and x_3 and each of this variables can take any of three states (left, center, right). We can combine this problem with the the information that it would be unlikely that the vehicle moves in one time step from left two right, without a step in the middle. By integrating such constraints as prior knowledge into that model and by doing inference in a graphical model we become a structure prediction problem.

5.2 Markov Random Field

Probability Theory Recap: In Probability Theory we talk about random variables which are variables or which we define distributions. There are two types of variable, discrete variables and continuous variables. A Joint distribution is denoted by $p(x, y)$ which is a short notation for $p(x = c, y = c')$.
Important rules:

- Sum rule (marginal distribution): $p(x) = \sum_y p(x, y)$ or $p(x) = \int_y p(x, y)$

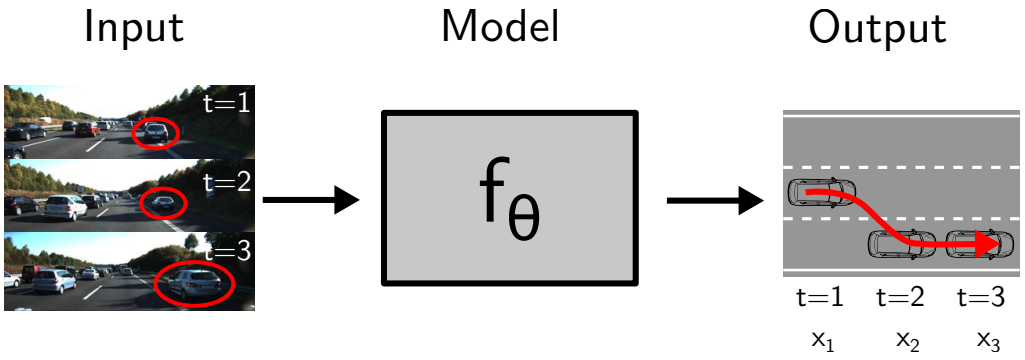


Figure 41: **Structure Prediction Example**

- Product rule: $p(x, y) = p(y|x)p(x)$
- Bayes rule: $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$

5.2.1 Markov Random Field

Potential:
A **potential** $\phi(x)$ is a non-negative function of the variable x. A **joint potential** $\phi(x_1, x_2, \dots)$ is a non-negative function of a **set** of variables.

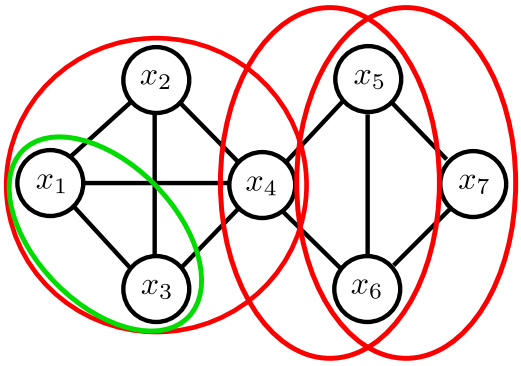
Markov Random Field (MRF) = Markov Network:
For a set of variables $\mathcal{X} = \{x_1, \dots, x_M\}$ a **Markov Random Field** is defined as a product of potentials over the **(maximal) cliques** $\{\mathcal{X}_k\}_{k=1}^K$ of the undirected graph $\mathcal{G}$:

$$p(\mathcal{X}) = \frac{1}{Z} \prod_{k=1}^{K} \phi_k(\mathcal{X}_k)$$

Special cases:

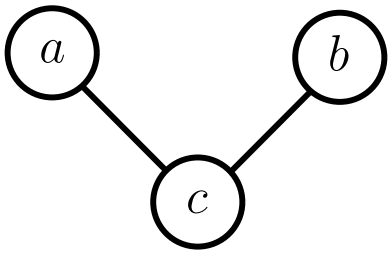
- clique of size two: **Pairwise Markov Random Field**
- If all potentials are strictly positive: **Gibbs distribution**

Undirected Graph:



An undirected graph $\mathcal{G}$ is a graph with vertices and undirected edges. A clique (green, red) is a subset of vertices that are fully connected.A maximal clique (red) is a clique that cannot be extended by any other vertex. Each of the vertices corresponds to a random variable and each edge correspond to the dependencies between this two random variables.

5.2.2 Properties of Markov Random Fields

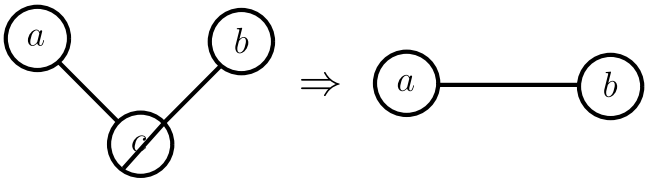


$$p(a,b,c) = \frac{1}{Z}\phi_1(a,c)\phi_2(b,c)$$

in this example we have Two maximal cliques of size two: $\mathcal{X}_1 = \{a,c\}$ and $\mathcal{X}_2 = \{b,c\}$
Z normalizes the distribution and is called **partition function**

$$Z = \sum_{a,b,c} \phi_1(a,c)\phi_2(b,c)$$

Property 1:



If we marginalizing over c, what happens is, that a and b become dependent to each other. we can proof this by showing that a and b are not independent:

$$p(a,b) \neq p(a)p(b)$$

Let's show this statement by contradiction. Assume the following holds true:

$$p(a,b) = \sum_c p(a,b,c) = \frac{1}{Z}\sum_c \phi_1(a,c)\phi_2(b,c)$$
$$= p(a)p(b) = \sum_{b,c} p(a,b,c) \sum_{a,c} p(a,b,c) = \frac{1}{Z}\sum_{b,c} \phi_1(a,c)\phi_2(b,c)\,\frac{1}{Z}\sum_{a,c} \phi_1(a,c)\phi_2(b,c)$$

Therefore, we have:

$$\sum_{a,b,c} \phi_1(a,c)\phi_2(b,c) \sum_c \phi_1(a,c)\phi_2(b,c) = \sum_{b,c} \phi_1(a,c)\phi_2(b,c) \sum_{a,c} \phi_1(a,c)\phi_2(b,c)$$

Consider binary variables $a,b,c \in \{0,1\}$ and the following choice of potentials

$$\phi_1(a,c) = [a=c] \quad \text{and} \quad \phi_2(b,c) = [b=c]$$

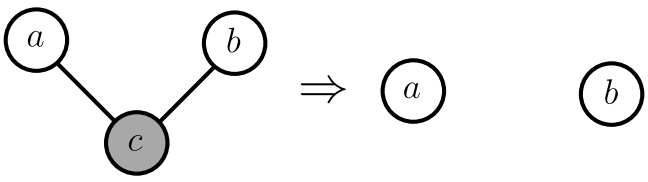
where $[\cdot]$ is the Iverson bracket which takes 1 if its argument is true and 0 otherwise.
We obtain the following contradiction:

$$\underbrace{\sum_{a,b,c} \phi_1(a,c)\phi_2(b,c)}_{2} \underbrace{\sum_c \phi_1(a,c)\phi_2(b,c)}_{[a=b]} = \underbrace{\sum_{b,c} \phi_1(a,c)\phi_2(b,c)}_{1} \underbrace{\sum_{a,c} \phi_1(a,c)\phi_2(b,c)}_{1}$$

Therefore, in general (for arbitrary choices of the potentials):

$$p(a,b) \neq p(a)p(b)$$

Property 2:



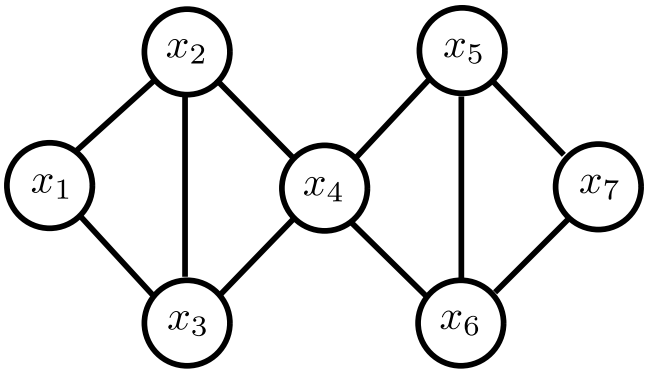
Conditioning on c makes a and b independent. We write this conditional independence statement compactly as: $a \perp\!\!\!\perp b \mid c$. This can be proven by showing:

$$p(a, b \mid c) = p(a \mid c)p(b \mid c)$$

The examples above can be generalized, as shown in the following.

Separation: A subset $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$ if every path from a member of $\mathcal{A}$ to any member of $\mathcal{B}$ passes through $\mathcal{S}$.

Global Markov Property:
For disjoint sets of variables $(\mathcal{A}, \mathcal{B}, \mathcal{S})$ where $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$, we have $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{S}$

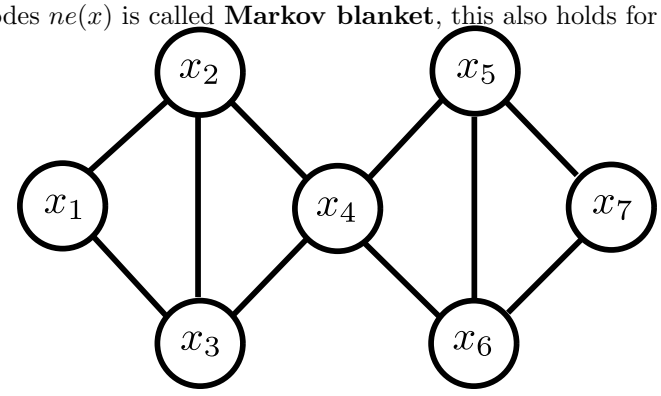


In this Graph x_4 separates (x_1, x_2, x_3) from (x_5, x_6, x_7)

Local Markov Property:
From the global Markov property, we can derive the **local Markov property:**
When conditioned on its neighbors, x becomes independent of the remaining variables of the graph:

$$p(x \mid \mathcal{X} \setminus \{x\}) = p(x \mid ne(x))$$

The set of neighboring nodes $ne(x)$ is called **Markov blanket**, this also holds for sets of variables.



In this example we have $p(x_4 \mid x_1, x_2, x_3, x_4, x_5, x_6, x_7) = p(x_4 \mid x_2, x_3, x_5, x_6)$, in other words $x_4 \perp\!\!\!\perp \{x_1, x_7\} \mid \{x_2, x_3, x_5, x_6\}$. Similarly, other independence relationships can be read off the graph.

5.2.3 Hammersley-Clifford Theorem

Hammersley-Clifford Theorem:
A probability distribution that has a strictly positive mass or density satisfies the **Markov properties** with respect to an undirected graph $\mathcal{G}$
if and only if it is a Gibbs random field, i.e., its density can be **factorized** over the (maximal) cliques of the graph.

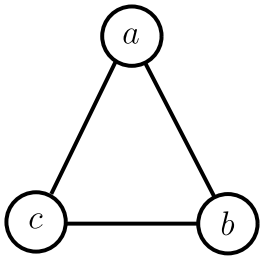
Filter View of Graphical Models:
Only distributions that factorize based on the (maximal) cliques may pass. Alternatively, only distributions that respect the Markov properties may pass. From the theorem above we know that both sets of distributions are identical

5.3 Factor Graph

Lets consider this factorization into potential functions:

$$p(a,b,c) = \frac{1}{Z}\phi(a,b)\phi(b,c)\phi(c,a)$$

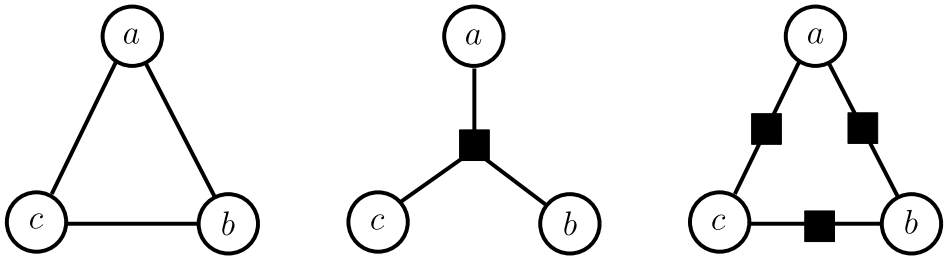
What is the corresponding Markov network / MRF? The corresponding graph is a fully connected graph with 3 nodes like this:



Obviously the maximum clique in this graph is (a,b,c). Which other factorization is represented by this network?

$$p(a,b,c) = \frac{1}{Z}\phi(a,b,c)$$

The second factorization is more general (admits larger class of distributions) But both factorizations respect the **same cond. independence assumptions** .Thus, the **factorization** into potentials is **not uniquely specified** by the graph. To disambiguate, we introduce an extra node (a square) for each factor:



On the left we have the Markov Network of $\frac{1}{Z}\phi(a,b,c)$ and $\frac{1}{Z}\phi(a,b)\phi(b,c)\phi(c,a)$. In the Middle is the Factor graph representation of $\frac{1}{Z}\phi(a,b,c)$ and on the right the Factor graph representation of $\frac{1}{Z}\phi(a,b)\phi(b,c)\phi(c,a)$.

5.3.1 Factor Graph (FG)

Given $\mathcal{X} = \{x_1, \ldots, x_M\}$, $\{\mathcal{X}_k\}_{k=1}^K$ with $\mathcal{X}_k \subseteq \mathcal{X}$ and a function

$$f(\mathcal{X}) = \prod_{k=1}^{K} f_k(\mathcal{X}_k)$$

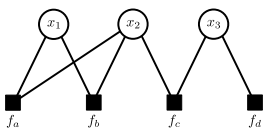
the **factor graph (FG)** is a bipartite graph with a **square node** for each factor f_k and a **circle node** for each variable x_i. By normalizing $f(\cdot)$, we obtain a distribution:

$$p(\mathcal{X}) = \frac{1}{Z}\prod_{k=1}^{K} f_k(\mathcal{X}_k)$$

As in the previous unit, $Z = \sum_{\mathcal{X}} f(\mathcal{X})$ denotes the partition function.

5.3.2 Examples

It is easy to read the distribution from an factor graph and vice versa.



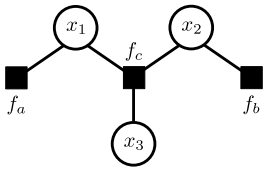
For example the distribution for this factor graph is:

$$p(x_1, x_2, x_3) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

And the other way:

$$p(x_1, x_2, x_3) = p(x_1)\, p(x_2)\, p(x_3|x_1, x_2)$$

The factor graph for this distribution is:



5.4 Belief Propagation

Lets consider a simple **chain structured vector graph**. All the nodes are arranged along a chain with factors in between and a given corresponding probability distribution.

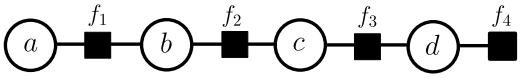


Figure 42: **Undirected Graph**

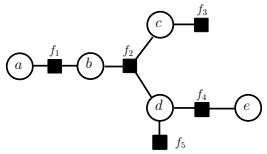
$$p(a, b, c, d) = \frac{1}{Z} f_1(a, b) f_2(b, c) f_3(c, d) f_4(d)$$

The computational complexity to compute this marginal distribution is, if the variables are all binary, we have two to the power of three equal to eight terms. So for this example its not a large Problem, but even a chain with 100 variables already has a computational complexity of 2 to the power of 100 and it becomes really intractable to do marginalization with such long chains. So instead, if we take this expression and we want to compute the marginal of a b and c which means we want to sum over d, we observe that not all of the factors involved in this expression depend on d but only the last two do. Therefore we can pull this summation operator inside that expression. This new marginalized expression (the summation over d) is called a message, where the variable d send a massage to variable c and that depends on c so it has an argument c. This simply recursion can be done further till we get this new function.

$$\begin{aligned}
p(a, b, c) &= \sum_d p(a, b, c, d) \\
&= \frac{1}{Z} f_1(a, b) f_2(b, c) \underbrace{\sum_d f_3(c, d) f_4(d)}_{\mu_{d \to c}(c)} \\
p(a, b) &= \sum_c p(a, b, c) \\
&= \frac{1}{Z} f_1(a, b) \underbrace{\sum_c f_2(b, c) \mu_{d \to c}(c)}_{\mu_{c \to b}(b)} \\
p(a) &= \sum_b p(a, b) \\
&= \frac{1}{Z} \sum_b f_1(a, b)\, \mu_{c \to b}(b) \\
&= \frac{1}{Z} \mu_{b \to a}(a)
\end{aligned}$$

With this function now the computational complexity is three times two which is six. If there are again 100 variables, the computational complexity would have been 100 times 2 which is 200 while in the previous case there is a complexity of 2 to the power of 100.

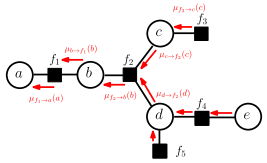
If we go to more complex structures like **tree structured graphs or branching graphs** now it becomes useful to have these vectors because now we can collect this information also at the vector level and then spread it further from the factors. Lets now consider a branching graph (tree):



with factors

$$f_1(a,b)f_2(b,c,d)f_3(c)f_4(d,e)f_5(d)$$

How can we now compute the marginal distribution $p(a,b)$?
Idea: Compute and pass messages



$$p(a,b) = \frac{1}{Z} f_1(a,b) \sum_{c,d,e} f_2(b,c,d)f_3(c)f_5(d)f_4(d,e)$$

If we want to compute the marginal of (a,b) on this tree structured graph the we do the same as before. We need to sum over c, d and e. This Message then can be split further into:

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d)f_3(c)f_5(d) \sum_{e} f_4(d,e)$$

5.4.1 Factor-to-Variable and Variable-to-Factor Messages

If we now want to make this a little bit more general, we have a message that's sent from a factor to a variable, and as an argument has the variable, is simply a summation over the state of the variables that are involved in that factor:

$$\mu_{f \to x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \to f}(y)$$

So its the summation over the state space of all the variables, except the variable that we're sending to, over the factor itself and the product of all the incoming messages. So in this case it would be the summation over bcd except for the variable b that we're sending to and the product of all the messages before b. In a last step we also see a more generalized form witch is just the product over all the messages been send from the variable to the vectors.

$$\mu_{x \to f}(x) = \prod_{g \in \{ne(x) \setminus f\}} \mu_{g \to x}(x)$$

To note is that messages can be reused and an important observation is that all the marginals can be written as function of messages. In the next step we can compute this general function with an algorithm, the so called Sum-Product Algorithm witch solves the inference Problem.

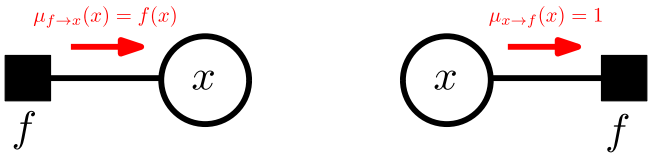
5.4.2 Sum-Product Algorithm

The algorithm works most precise on singly-connectet graphs (like chain or tree graphs). In short the algorithm first initialize then we send variable to factor messages then factor to variable messages,this is repeated until all messages have been calculated and the algorithm is converged and then we calculate the desired marginals which is the goal of the sum product algorithm from these messages. So let's look at these steps in more detail. In the first step the messages from the extremal node factors are initialized to the factor themselves. Also the messages from the extremely variable notes to the factors is set to 1. The second step of this algorithm is the variable to factor message computation with the product of all the incoming messages witch we already seen before. In the third step we now have the sum, the factor to variable message has, this form we've also already seen and this gives the name of this algorithm the sum product algorithm because we have a sum of a term that involves products. Finally once we have applied this algorithm and has converged we can read off the marginal distributions by simply taking the product of all the messages coming into a particular variable. The difference to step two is that here we calculate the product over all the neighbors.

Belief Propagation: Algorithm to compute all messages efficiently. Assumes that the graph is singly-connected (chain, tree).

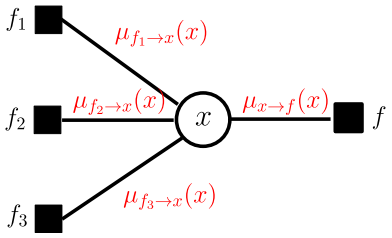
Step 1 Initialization:

- Messages from extremal node factors are initialized to factor
- Messages from extremal variable nodes is set to 1



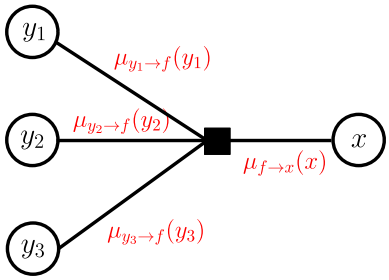
Step 2 Variable to Factor message:

$$\mu_{x\to f}(x) = \prod_{g\in\{ne(x)\setminus f\}} \mu_{g\to x}(x)$$



Step 3 Factor to Variable message:

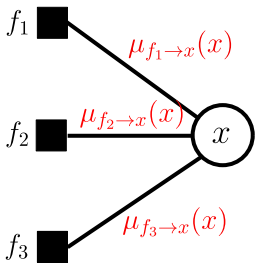
$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f\setminus x} f(\mathcal{X}_f) \prod_{y\in\{ne(f)\setminus x\}} \mu_{y\to f}(y)$$



Remark: We sum over all states in the set of variables

Step 5 Calculate the desired marginals from the messages:

$$p(x) \propto \prod_{f\in ne(x)} \mu_{f\to x}(x)$$



Remark: Unlike in step 2, here we calculate the product over all neighbors

5.4.3 Max-Product Algorithm

If we now want to find for a given distribution p of abcd the most likely state there is a very similar algorithm which is called the max product algorithm. So this solves the **Maximum-A-Posteriori (MAP) problem**. Again we use the factorization structure to distribute maximization to local computations as before with the summation. We obtain the maximal probability value, but not the most probable state. In the next step we find the optimal values be **backtracking** (dynamic programming). If the maximum is unique then the MAP solution is the maximum of the max marginals and that's much easier to compute because we need to just find the maximum of the computed vector per variable.

Problem: For a given distribution $p(a, b, c, d)$ find the most likely state:

$$a^*, b^*, c^*, d^* = \operatorname*{argmax}_{a,b,c,d} \; p(a, b, c, d)$$

This is called the **Maximum-A-Posteriori (MAP)** solution. Again use factorization structure to distribute maximisation to local computations.
Example: Chain

$$p(a, b, c, d) = \frac{1}{Z} f_1(a, b) f_2(b, c) f_3(c, d)$$

$$\begin{aligned}
\max_{a,b,c,d} p(a, b, c, d) &= \max_{a,b,c,d} f_1(a, b) f_2(b, c) f_3(c, d) \\
&= \max_{a,b,c} f_1(a, b) f_2(b, c) \underbrace{\max_{d} f_3(c, d)}_{\mu_{d\to c}(c)} \\
&= \max_{a,b} f_1(a, b) \underbrace{\max_{c} f_2(b, c)\mu_{d\to c}(c)}_{\mu_{c\to b}(b)} \\
&= \max_{a} \underbrace{\max_{b} f_1(a, b)\mu_{c\to b}(b)}_{\mu_{b\to a}(a)} \\
&= \max_{a} \mu_{b\to a}(a)
\end{aligned}$$

By this algorithm we obtain the maximum probability value, but not the most probable state.
Solution: Once messages are computed, find the optimal values:

$$\begin{aligned}
a^* &= \operatorname*{argmax}_{a} \; \mu_{b\to a}(a) \\
b^* &= \operatorname*{argmax}_{b} \; f_1(a^*, b)\, \mu_{c\to b}(b) \\
c^* &= \operatorname*{argmax}_{c} \; f_2(b^*, c)\, \mu_{d\to c}(c) \\
d^* &= \operatorname*{argmax}_{d} \; f_3(c^*, d)
\end{aligned}$$

This is called **backtracking** (dynamic programming). If maximum unique the MAP solution is the maximum of the "max-marginals". The latter is easy to compute (find maximum of computed vector per variable).

5.4.4 Log Representation

In large graphs, messages may become very small/big (due to product). This leads to numerical problems when storing them as floating point numbers. The Solution is to work with log-messages instead $\lambda = \log \mu$

Variable-to-factor messages:

$$\mu_{x\to f}(x) = \prod_{g\in\{ne(x)\setminus f\}} \mu_{g\to x}(x)$$

then becomes

$$\lambda_{x\to f}(x) = \sum_{g\in\{ne(x)\setminus f\}} \lambda_{g\to x}(x)$$

And the Factor-to-variable messages

$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f\setminus x} f(\mathcal{X}_f) \prod_{y\in\{ne(f)\setminus x\}} \mu_{y\to f}(y)$$

then become

$$\lambda_{f\to x}(x) = \log\left(\sum_{\mathcal{X}_f\setminus x} f(\mathcal{X}_f)\exp\left[\sum_{y\in\{ne(f)\setminus x\}}\lambda_{y\to f}(y)\right]\right)$$

5.4.5 Belief Propagation Algorithm

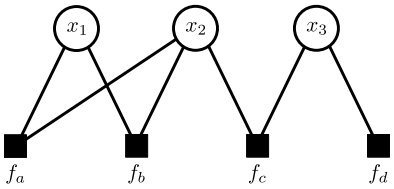
1. Input: *variables* and *factors*
2. Allocate all *messages* (log representation)
3. Initialize *messages* to 0 (=uniform distribution)
4. For N iterations do
   (a) Update all *factor-to-variable messages*
   (b) Update all *variable-to-factor messages*
   (c) Normalize all *variable-to-factor messages*:
   $\lambda_{x\to f}(x) = \lambda_{x\to f}(x) - \text{mean}\left(\lambda_{x\to f}(x)\right)$
5. Read off marginal or MAP state at each variable.

5.4.6 Loopy Belief Propagation:

We assumed that the graph is either a chain or a tree without any loops. Since in computer vision many problems have loops the assumptions we have made in order to get an exact solution actually break. If we apply the algorithms to loopy graphs we loose exactness and the guarantee of convergence.
First pass all the factor to variable messages and then pass all the variable to factor messages and repeat this for any iterations until we hopefully converge. There is no guarantee of convergence.

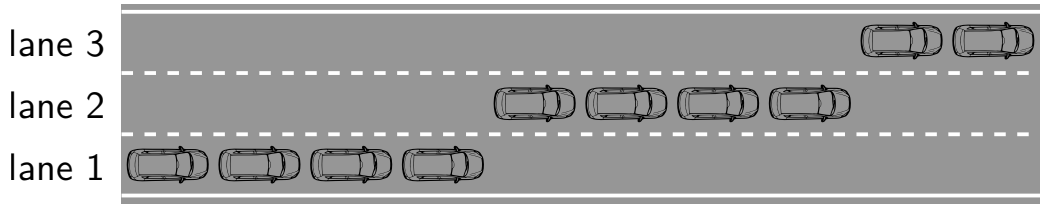
Which message passing schedule?

- Random or fixed order
- Popular choice:
  1. Factors → variables
  2. Variables → factors
  3. Repeat for N iterations
- Can be run in parallel as factor graph is bipartite:

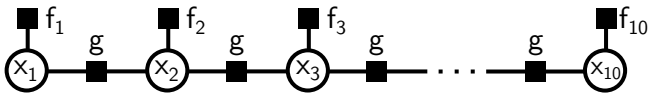


5.5 Examples

5.5.1 Example 1: Vehicle Localization



Goal: Estimate vehicle location at time $t = 1,\ldots,10$. The variables are $\mathcal{X} = \{x_1,\ldots,x_{10}\}$ with $x_i \in \{1,2,3\}$ ($C = 3$). And observations are given as $\mathcal{O} = \{\mathbf{o}_1,\ldots,\mathbf{o}_{10}\}$ $\quad \mathbf{o}_i \in \mathbb{R}^3$.



$$p_\theta(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{10} f_i(x_i) \prod_{i=1}^{9} g_\theta(x_i, x_{i+1})$$

Unary Factors: $f_1(x_1) = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.2 \end{bmatrix}, \quad f_2(x_2) = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}, \quad f_3(x_3) = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.7 \end{bmatrix}, \quad \ldots$

Pairwise Factors: $g_\theta(x_i, x_{i+1}) = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{31} & \theta_{32} & \theta_{33} \end{bmatrix} \quad g_\theta(x_i, x_{i+1}) = \begin{bmatrix} 0.8 & 0.2 & 0.0 \\ 0.2 & 0.6 & 0.2 \\ 0.0 & 0.2 & 0.8 \end{bmatrix}$

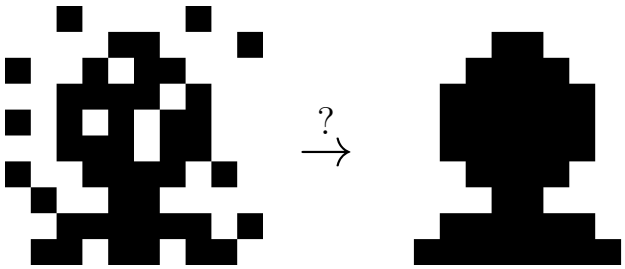
- Change 2 lanes: 0%
- Change 1 lane: 20%
- Otherwise: stay on lane

Learning Problem:

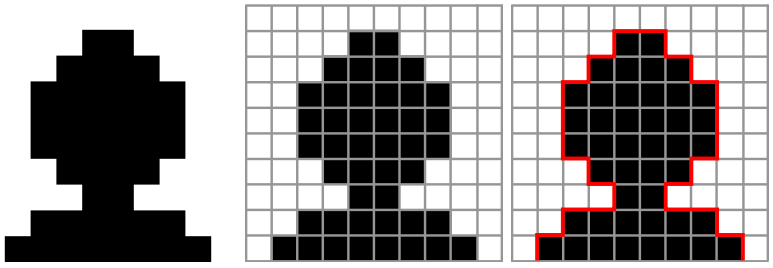
$$\theta^* = \operatorname*{argmax}_\theta \prod_n p_\theta(\mathbf{x}_n | \mathbf{o}_n)$$

5.5.2 Example 2: Image Denoising

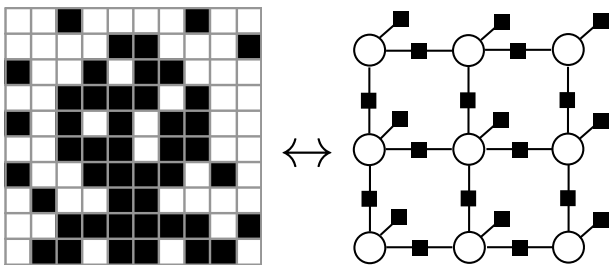
Can we recover the original image from a noisy observation?



- Variables: $x_1, \ldots, x_{100} \in \{0, 1\}$
- Unary potentials: $\psi_1(x_1), \ldots, \psi_{100}(x_{100})$
- $\psi_i(x_i) = [x_i = o_i]$ with observation o_i
- Log representation: $\psi_i(x_i) = \log f_i(x_i)$
  $p(x) = \frac{1}{Z} \prod_i f_i(x_i) = \frac{1}{Z} \exp\{\sum_i \psi_i(x_i)\}$



We like to integrate prior knowledge by adding constraints to the problem. What prior knowledge do we have about this image? Smoothness: Neighboring pixels tend to have the same label. How many neighbors share the same label? 10 x 10 x 2 - 20 = 180 neighborhood relationships in total. 34x label transition and 146x same label (factor 4.3 more).



$$p(x_1, \ldots, x_{100}) = \frac{1}{Z} \exp\left\{ \sum_{i=1}^{100} \psi_i(x_i) + \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

- Unaries: $\psi_i(x_i) = [x_i = o_i]$ with observation $o_i \in \{0, 1\}$
- Pairwise potential: $\psi_{ij}(x_i, x_j) = \lambda \cdot [x_i = x_j]$
- Parameter α controls strength of prior $\Rightarrow$ Exercise.

6 Applications of Graphical Models

Motivated by the ambiguities related to graphical models and inference algorithm called belief propagation that allows us to compute maximum posteriori solutions or map solutions and marginals, this section is focusing on some applications of graphical models to computer vision problems.

The first part is going to focus on the stereo reconstruction problem using Markov Random Fields (MRF) to perform maximum posterior inference with discrete variables. After that, we will take a brief look at applications in terms of dense multi-view stereo reconstruction, which are from more than two views in the context of the sum product algorithm for inferring marginals and probabilistic results. Finally, the optical flow estimation problem is going to be discussed, which is a estimation problem in the 2D image plane formulated using continuous variables.

6.1 Stereo Reconstruction

This section focuses on stereo reconstruction problem by performing MAP inference in Markov Random Fields with discrete variables using the max product algorithm.

6.1.1 Stereo Matching Ambiguities

The difficulties in local stereo matching or block matching are as follows:

- Textureless Surfaces (impossible to distinguish adjacent patches)
- Occlusions (background changes diffently from the foreground)
- Non-Lambertian Surfaces (in case of reflection)
- Repetitons of Pattern

In order to overcome these ambiguities we need to incorporate prior knowledge about the real world statistics (for example, using Brown range image dataset that provides a large number of depth maps). From this dataset we have found that depth vary slowly except at object discontinuities which are sparse, and we want to incorporate this prior knowledge into the model.

6.1.2 Stereo MRF

Therefore, we specify, as follows, a loopy Markov Random Field (MRF) on a grid (see Fig. 15) that models a distribution over the space of all disparity maps and solve for the whole disparity map $\boldsymbol{D}$ at once by MAP estimation i.e. by minimizing Gibbs energy:

$$p(\boldsymbol{D}) \propto \prod_i f_{data}(d_i) \times \prod_{i \sim j} f_{smooth}(d_i, d_j)$$

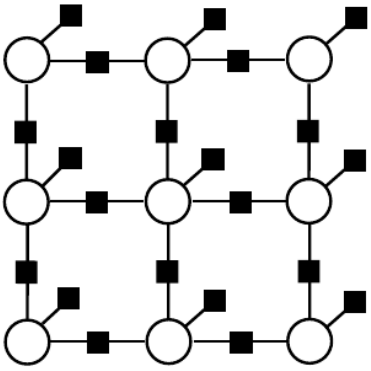
or equivalently

$$p(\mathbf{D}) \propto \exp\Big\{ \underbrace{-\Big(\sum_i \psi_{data}(d_i) + \lambda \sum_{i \sim j} \psi_{smooth}(d_i, d_j) \Big)}_{\text{Energy E}} \Big\}$$

where $\psi := -\log f$, $i \sim j$ means i and j are neighboring pixels, $\psi_{data}(d)$ are unary and $\psi_{smooth}(d, d')$ are pairwise terms.

Our objective is to maximize $p(\boldsymbol{D})$ i.e. to minimize the Gibbs energy. Notice that, the lowest energy value would be obtained in terms of the pairwise terms by just having a constant disparity map where all the d's are the same but in that case, the matching cost would be high. Thus by solving this MRF we are trading off the unity term and the pairwise term and this tradeoff is controlled by this parameter λ. If we set $\lambda = 0$, then we

fall back to the standard block matching algorithm, and as we increase λ we obtain smoother and smoother solutions.



Here we illustrate the model using a four connected grid structure graph which corresponds to a three by three pixel image. In this graph, every circle corresponds to a variable that is the disparity that we want to infer for that pixel. We have unary factors indicated by the the squares that are connected to only a single variable. These unary terms are the matching costs that we can compute using a block matching technique for example using a siamese network. We also incorporate our prior knowledge about the smoothness of disparity maps into this MRF by adding pairwise connections in this factor graph. Below are two such priors for pairwise terms:

- **Potts Model:** $\psi_{smooth}(d, d') = [d \neq d']$
- **Truncated l_1 penalty:** $\psi_{smooth}(d, d') = \min(|d - d'|, \tau)$

We can solve now this MRF approximately using the max product belief propagation algorithm. There are also other algorithms, that solve MRF approximately for the disparity map $\boldsymbol{D}$, for example using BP, graph cuts etc.

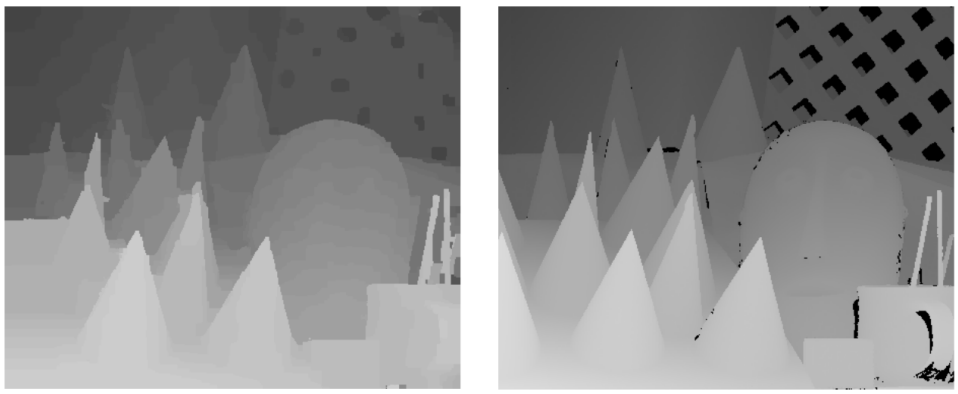
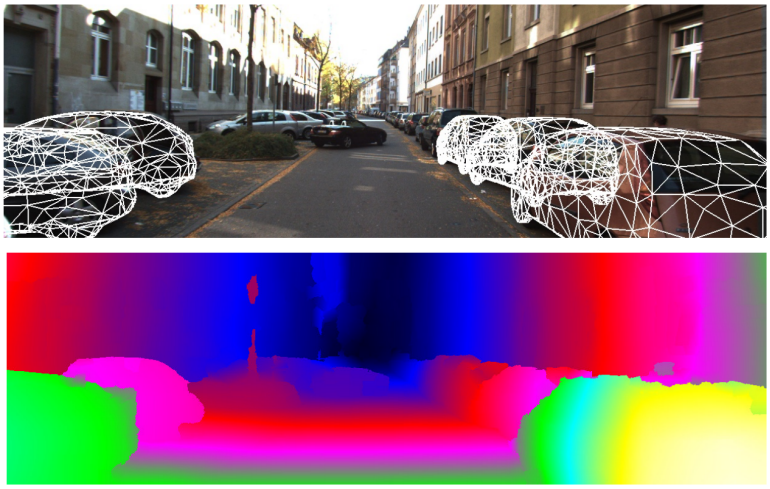


Figure 43: Stereo MRF Results: Inference Result (left) vs Ground Truth (right)

6.1.3 Example



$$E(\mathbf{D},\mathbf{O}) = \underbrace{\sum_i \psi_i^{\mathcal{A}}(d_i)}_{\text{Appearance}} + \lambda_{\mathcal{S}} \underbrace{\sum_{i\sim j} \psi_{ij}^{\mathcal{S}}(d_i,d_j)}_{\text{Smoothness}} + \lambda_{\mathcal{O}} \underbrace{\sum_k \psi_k^{\mathcal{O}}(o_k)}_{\text{Object Semantics}} + \lambda_{\mathcal{C}} \underbrace{\sum_k \sum_i \psi_{ki}^{\mathcal{C}}(o_k,d_i)}_{\text{3D Consistency}}$$

Figure 44: **Inferred scene with local structure (car shapes) included**

An example of this is shown in Fig. 44 [21]. This is the energy formulation with slightly different notation where the Gibbs energy composed of an appearance or a data term and a smoothness term. The local pairwise term cannot deal with strong violations such as reflections. Therefore, stronger assumptions are needed in this case. A solution to this is to think about the scene in terms of per pixel depth for disparities, and the objects that are present in the scene. For street scenes, there are likely a lot of cars, and the shape of cars is generally known. Thus, we can try to infer not only the disparity map, but also the objects jointly, so there are two additional terms. The semantics term tries to make the semantics of the infrared objects similar to semantics that have been inferred from the image. The consistency term tries to make the 3d objects that have been inferred consistent with the disparity map. By integrating this object level prior knowledge, further regularization of the problem provides a solution like in Fig. 44. While it is not perfect, the cross outliers at the reflections have been taken into account by the object level knowledge.

In summary, block matching easily suffers from matching ambiguities, and choosing the window size is quite problematic due to this trade-off. What can be done to improve on the problem is to integrate smoothness constraints that resolve some of these ambiguities. Furthermore, integrating recognition cues such as detecting objects and trying to infer objects jointly with the disparity map can better regularize the problem and help overcoming very strong ambiguities

6.2 Multi-View Reconstruction

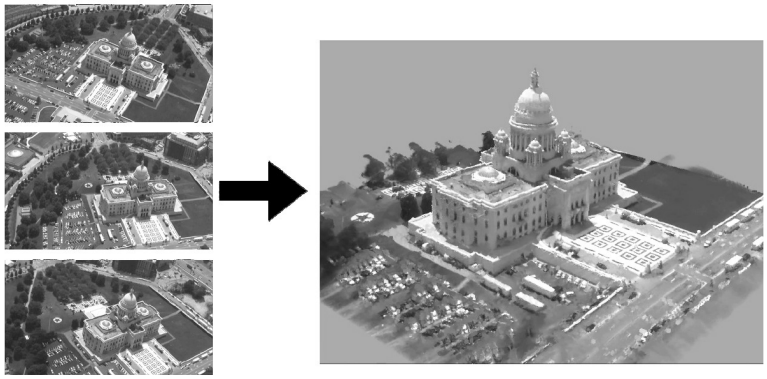


Figure 45: Reconstruction of images from multiple views

The second application of graphical models is in the context of dense multi-view stereo reconstruction which is the problem of reconstructing a 3D scene not from just two images, but from multiple images of that scene , see Fig. 45. The advantage is that there are more constraints to better resolve ambiguities, and being able to achieve a complete 3D reconstruction of the scene.

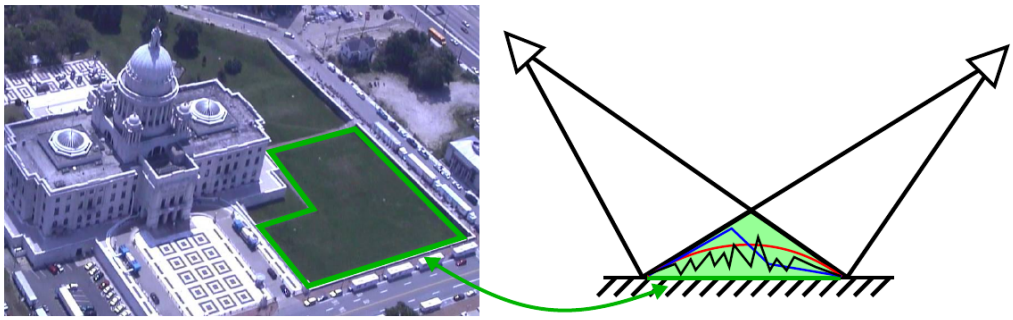
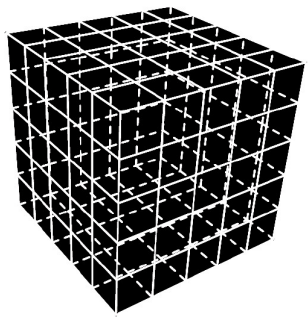


Figure 46: Ambiguities present in the image. In this case, this is a green texture-less surface

Even for using many more cameras, image-based 3d reconstruction can still be problematic. In the context of texture-less areas (Fig. 46), the green area that has almost the same color everywhere even if we look at the scene from multiple different viewpoints. Because all of these pixels are roughly of the same color or intensity, any of these other surfaces such as the blue, the red, or the black ones are also plausible surfaces. Therefore, there are still a lot of reconstruction ambiguities. Some of these ambiguities can be resolved by introducing appropriate priors such as flatness, but the stronger these priors are, the stronger our assumptions are, and there will more likely be mistakes in case these assumptions are violated.

6.2.1 Representation

Below shows the representation that we are going to use, which is to discretize the 3D volume into a discrete set of voxels. Each of these cubes is a voxel, and the number of voxels depends on the resolution applied in each dimension.



- **Voxel occupancy:**
$$o_i = \begin{cases} 1 & \text{if voxel } i \text{ is occupied} \\ 0 & \text{if voxel } i \text{ is empty} \end{cases}$$
- **Voxel appearance:**
$a_i \in \mathbb{R}$
- **Shorthand notation:**
$\mathbf{o}_r = \{o_1^r, \ldots, o_{N_r}^r\}$
$\mathbf{a}_r = \{a_1^r, \ldots, a_{N_r}^r\}$

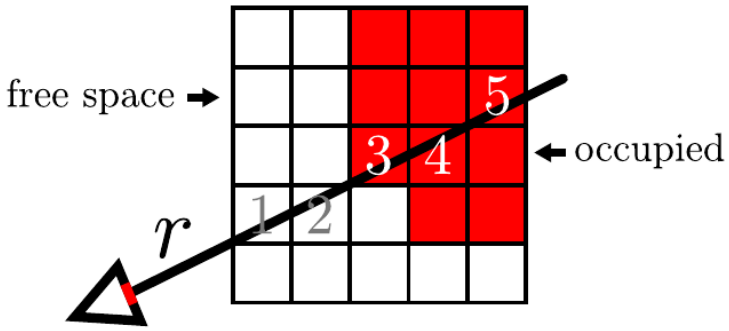
To represent the 3D geometry and appearance inside, each of the voxels is assigned with two random variables. The first random variable is the **voxel occupancy** o_i: a binary random variable that is equal to 1 if that particular voxel is occupied and 0 otherwise. The second variable is the **voxel appearance** a_i: a real number for grayscale images intensity or r^3 to represent color or more complex properties of the of the appearance (in this case, it is just a one-dimensional real number a scalar that represents how bright that voxel appears).

On the image plane of the camera, there are many pixels, and the combination of the camera center and the particular pixel defines a ray that passes through 3D space by projective geometry. For each of these rays, $\mathbf{o}_r$ and $\mathbf{a}_r$ which are collections of the set of random variables $((o_k^r,...), (a_k^r,...))$ from the entire volume that are intersected by that ray starting from the camera center

6.2.2 Image Formation Process

In order to solve a computer vision problem such as multi-view of 3D reconstruction, we first have to understand the image formation process. Intuitively, for solid objects, the color of a particular pixel corresponding to a particular ray of a particular camera is simply the color of the first occupied voxel that is hit by that ray. Let

I_r denote intensity or color at the pixel r and o_i^r and a_i^r be defined as before. The intensity at the pixel r is simply this expression as followed.



$$I_r = \sum_{i=1}^{N} o_i \prod_{j<i} (1 - o_j)\, a_i$$

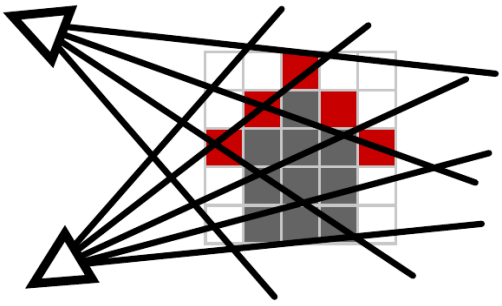
I_r: intensity at pixel r $\quad$ o_i: occupancy of voxel i $\quad$ a_i: appearance of voxel i

Figure 47: **Image formation process:** the intensity/color of at the pixel r as viewed from the first voxel hit by the ray r

This term is equal to **1** if o_i^r is the first occupied voxel (if o_i^r is the first occupied voxel, o_i^r is 1 while o_j^r is 0 so the entire product is also 1. In all other cases, this expression will be 0). We want to take the color of that voxel so we multiply with that color or intensity value a_i^r and copy it to the pixel. Because we don't know which of these voxels along the ray is the first occupied voxel, we need to sum over all possible hypotheses (but only one of this term in the sum will be a_i^r). This formula represents pixel intensity or color corresponding to the intensity or color of the first occupied voxel.

6.2.3 Probabilistic Model

As we want to formulate a probabilistic model perform inference on such probabilistic model in order to expose uncertainty, given the knowledge of the image formation process, we define a probability distribution over all the occupancy variables and all the appearance variables. These are denoted by **O** and **A**, and we are factorizing this distribution over all occupancy and appearance variables into a product of unaries - ϕ_v, and ray factors - ψ_r(these ray factors are higher order potentials because they connect all the variables along the ray), see Fig. 48.



$$p(\mathbf{O}, \mathbf{A}) = \frac{1}{Z} \prod_{v \in \mathcal{V}} \{\underbrace{\varphi_v(o_v)}_{\text{unary}} \prod_{r \in \mathcal{R}} \underbrace{\psi_r(\mathbf{o}_r, \mathbf{a}_r)}_{\text{ray}}$$

Figure 48: Joint probability distribution of the color intensity and occupancy

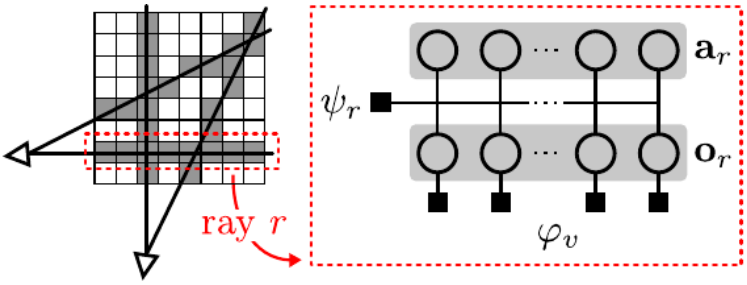
Now the unary potentials are just some simple prior knowledge that we can specify about general occupancy

of the occupancy variables i.e. we can choose a simple Bernoulli distribution with hyperparameter γ:

$$\varphi_v(o_v) = \gamma^{o_v} (1-\gamma)^{1-o_v}$$

The hyperparameter γ that controls how much we believe voxels in a scene are empty or not. As most voxels in a scene are empty, γ is typically chosen smaller than 0.5 which helps a in cleaning up some of the outliers that are inferred by this model.

The really important term is the ray potential that models the consistency of the 3D reconstruction to be inferred. The expression (Fig. 48) is is exactly the same as the expression (Fig. 49), except that $\mathbf{a}_i^r$ has replaced with the Gaussian centered at I_r) because the intensity of a particular voxel observed in all the images might be different, and because of noise and our simplistic assumptions do not hold true even in the Lambertian case. This is a hyperparameter that we can tune with the variance σ parameter. The configuration of $\mathbf{o}$(s) and $\mathbf{a}$(s) that maximizes the joint distribution is one where for all of the rays in all of the cameras is increasing the value of this potential. This is the case if the first occupied voxel is similar to the pixel that is corresponding to that ray (if a_i^r is similar to I_r then this term will be large, and if it's the first occupied voxel, then first $o_i^r \prod_{j<i}(1-o_j^r)$ will be one; so, we are getting a large value).



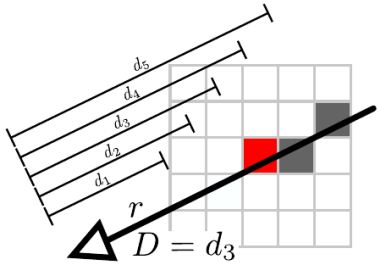
$$\psi_r(\mathbf{o}_r, \mathbf{a}_r) = \sum_{i=1}^{N_r} o_i^r \prod_{j<i} (1-o_j^r) \underbrace{\mathcal{N}(a_i^r | I_r, \sigma)}_{\text{Gaussian Noise}}$$

Figure 49: **Ray potential**

We not only have a constraint from a single image, also constraints from all the images. This illustrates that only if we have the constraints from all the images, we can on one hand infer a complete 3D reconstruction, but also reduce the uncertainty because we have many neighboring views that see the same surface. This leads to better surface reconstructions, so we want to infer a 3D reconstruction that is consistent with all of the images that have been taken from that scene.

6.2.4 Bayes Optimal Depth Estimation

Now to do probabilistic inference with this model we will use the sum product algorithm and the simplest thing we can do in terms of an inference question is to ask, for any ray in the scene what is the depth along that ray i.e. what is the first occupied voxel along the ray. If we can get the depth along each potential ray in the scene then we can we have the 3D reconstruction.



Consider a single ray r in space and let d_k be the distance from the camera to voxel k along ray r and let

depth $D \in \{d_1, \ldots, d_N\}$ be the distance to the closest occupied voxel. Then the optimal depth estimate is

$$\begin{aligned} D^* &= \text{argmin}_{D'} \, \text{Risk}(D') \\ &= \text{argmin}_{D'} \, \mathbb{E}_{p(D)}[\Delta(D, D')] \\ &= \begin{cases} \text{mean}(p(D)) & \text{if } \Delta(D, D') = (D - D')^2 \\ \text{median}(p(D)) & \text{if } \Delta(D, D') = |D - D'| \, . \end{cases} \end{aligned}$$

But this requires the marginal depth distribution $p(D)$ along each ray.

6.2.5 Depth Distribution for Single Ray

Let

$$o_1 = 0, \ldots, o_{k-1} = 0, o_k = 1$$

Then,

$$\begin{aligned} p(D = d_k) &= p(o_1, \ldots, o_k) \\ &= \sum_{\boldsymbol{o}_{>k}} \int_{\boldsymbol{a}} p(\boldsymbol{o}, \boldsymbol{a}) \\ &\propto \sum_{\boldsymbol{o}_{>k}} \int_{\boldsymbol{a}} \psi(\boldsymbol{o}, \boldsymbol{a}) \prod_i \mu(o_i) \prod_i \mu(a_i) \quad \ldots (*) \\ &= \sum_{\boldsymbol{o}_{>k}} \int_{\boldsymbol{a}} \left(\underbrace{\sum_{i=1}^{N} o_i \prod_{j<i} (1 - o_j) \mathcal{N}(a_i | I, \sigma)}_{= \mathcal{N}(a_k | I, \sigma)} \right) \prod_i \mu(o_i) \prod_i \mu(a_i) \\ &= \sum_{\boldsymbol{o}_{>k}} \int_{\boldsymbol{a}} \mathcal{N}(a_k | I, \sigma) \prod_i \mu(o_i) \prod_i \mu(a_i) \\ &= \sum_{\boldsymbol{o}_{>k}} \int_{\boldsymbol{a}} \mathcal{N}(a_k | I, \sigma) \prod_{i>k} \mu(o_i) \prod_i \mu(a_i) \times \prod_{i \leqslant k} \mu(o_i) \\ &= \underbrace{\sum_{\boldsymbol{o}_{>k}} \int_{\boldsymbol{a}_{\neq k}} \prod_{i>k} \mu(o_i) \prod_{i \neq k} \mu(a_i)}_{=1} \times \prod_{i \leqslant k} \mu(o_i) \int_{\boldsymbol{a}_k} \mathcal{N}(a_k | I, \sigma) \mu(a_k) \\ \Rightarrow p(D = d_k) &= \prod_{i \leqslant k} \mu(o_i) \int_{\boldsymbol{a}_k} \mathcal{N}(a_k | I, \sigma) \mu(a_k) \end{aligned}$$

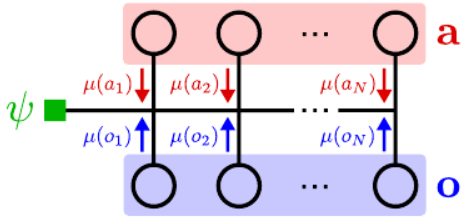
where at $(*)$, we obtain the expression using the sum product belief propagation algorithm, in which the marginal is given as the product of factors and incoming messages. We can obtain μ as follows:

$$\mu_{\psi_v \to o_v}(o_v) = \psi_v(o_v)$$

$$\mu_{x \to \psi_r}(x) = \prod_{f \in \mathcal{F} \setminus \psi_r} \mu_{f \to x}(x)$$

and $\mu_{\psi_r \to x}(x)$ can be computed in linear time. Intuitionally last expression conveys the message that,

Depth $D = d_k \Leftrightarrow$ Voxel k is occupied and visible and explains the observed pixel value.



6.2.6 Inference

Some results for a dataset that has been captured by flying around a city capturing images of that city, and at the same time, there was a lighter that was measuring depth so that relatively accurate ground truth of depth was available for evaluation. This algorithm compared to previous local and global algorithms that were not exposed to uncertainty improves performance in particular in texture-less areas.

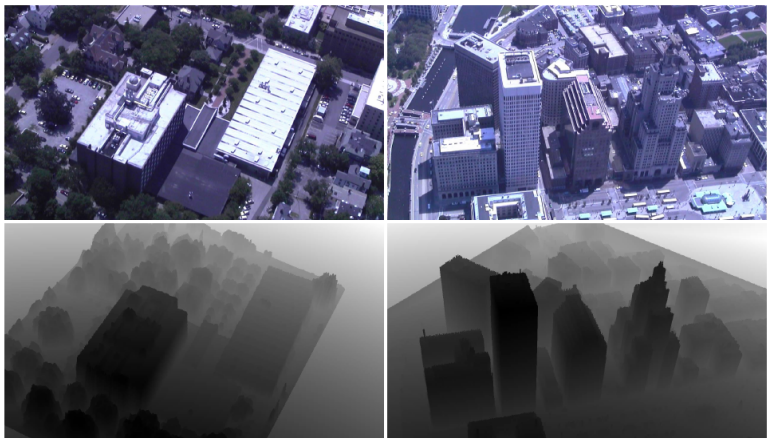


Figure 50: Results for dataset that has been captured by flying around a city

The depth map that has been inferred by a previous algorithm that was using maximum posteriori inference that was not exposed to the uncertainty. The error map is a colored visualization of where errors are large (red) or small (blue).

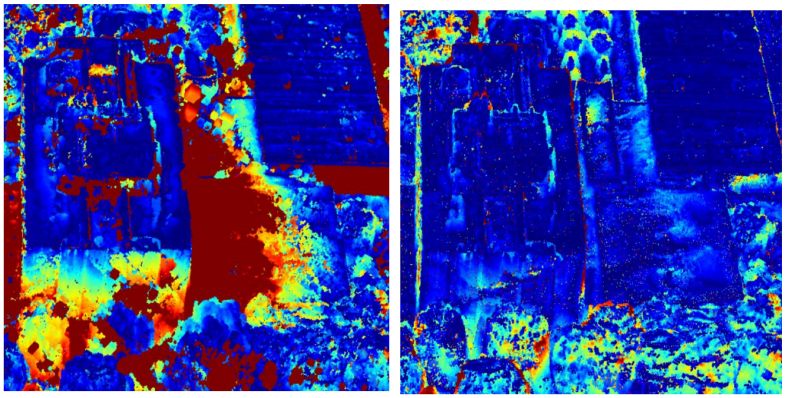


Figure 51: Depth map inferred with (right) vs without uncertainty (left)

Challenges :

1. MRF comprises discrete and continuous variables
2. Ray potentials are high-order
3. Many factors: each pixel defines a factor

Solutions :

1. Approximate continuous belief propagation
   $\Longrightarrow$ Update MoG's via importance sampling
2. Messages can be calculated in linear time
3. Octree implementation & GPGPU parallelization

6.2.7 Integrating 3D Shape Priors

Similar to the with 2D stereo matching, higher order constraints in terms of objects can also be utilized. Such shape prior knowledge for many scenes like the downtown scene is available. For example, the GPS tag of the location of where the images have been taken, you can simply query that in 3d warehouse and you obtain 3d

models for that scene such as rough building outlines. For indoor scenes, many rooms contain particular type of furniture like IKEA furniture, so for this type of furniture, there are also a lot of CAD models available online that can be used as prior knowledge. However, there are a lot of challenges also involved. These 3d models often only contain coarse and inaccurate structure, and the orientation and the location are not known. There might be occlusions, or the retrieve models might actually not be present in the scene, and the object size might not be correct.

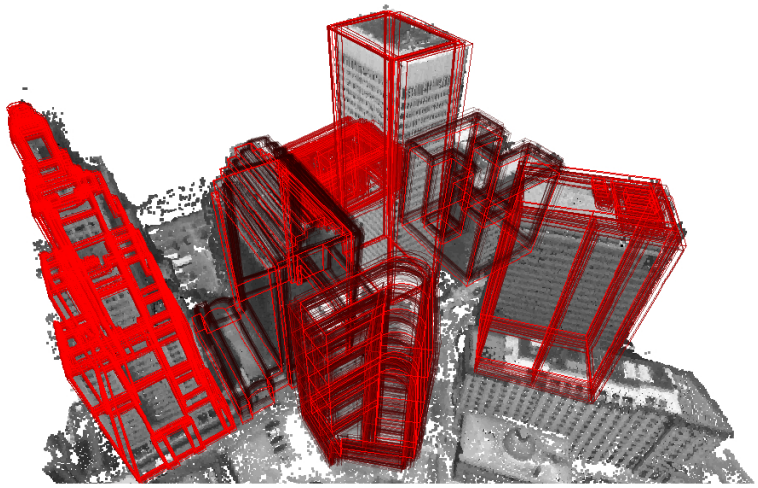


Figure 52: Integration of 3D priors to the image (buiding shapes)[56]

6.2.8 Summary

Probabilistic Multi-View Reconstruction has the following pros and cons:

Pros :

- Probabilistic formulation using Graphical Models is tractable as ray factors decompose
- Non-local constraints via joint inference in 2D and 3D
- CAD priors can help disambiguate textureless regions
- Using octrees, reconstructions up to 10243 voxels are possible

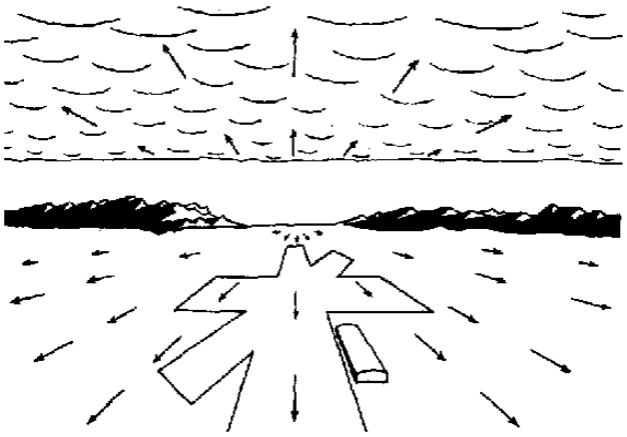
Cons :

- Only approximate inference possible (highly loopy)
- Relatively slow: several minutes per scene on a GPU
- Appearance term very simplistic and not robust (does not take into account Non-Lambertian appearance, noise and outliers)
- Resolution limited by discrete voxels (as opposed to meshes)

6.3 Optical Flow

In this section we will discuss the optical flow problem and how it can be formulated as an inference problem with a Markov Random Field.

6.3.1 Optical Flow



In the above scene where a pilot landing an airplane on this airfield, the arrows indicate how much a particular pixel moves in this apparent motion. Notice that the points that are nearby the observer shows large apparent motion than those are further away. This motion of how the pixels move in an image is caused by either objects moving or the observer moving. In this case the scene is static but the camera is moving with respect to the scene causing this motion field.

6.3.2 Stereo vs Optical Flow

Optical flow has a relationship to stereo. Both are 2D estimation problems, but in stereo we take two images at the same time, or of a static scene, and in optical flow, no such assumptions is made. In stereo, we only have camera motion while optical flow has both camera and object motions. Therefore, stereo is a 1D estimation problem that has to be searched along the epipolar line. Optical flow is a full 2D estimation problem as a pixel in the first image can be located at any pixel in the second image.

6.3.3 Motion Field and Optical Flow

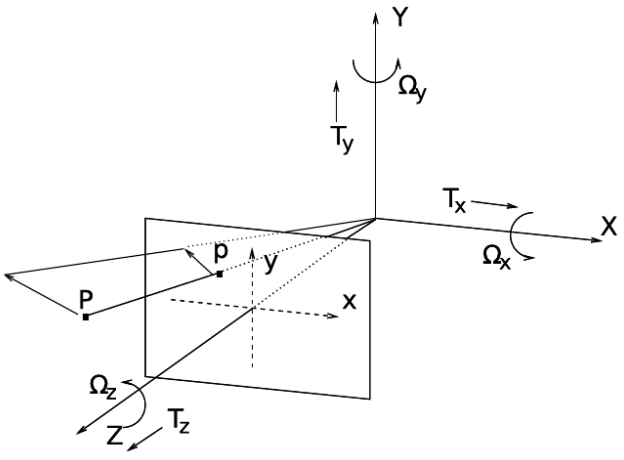


Figure 53: Simple visualization of motion field on 2D image

A **motion field** is the 2d motion that represents the projected motion of the actual 3D motion of an object or points in the scene onto the image plane. This motion field can be the result of camera motion or object motion (or both). The **optical flow** is the 2D velocity field describing the apparent motion in the image. It is the displacement of pixels looking similar.

Thus, optical flow and motion field are not the same. Lets do two little thought experiments to understand why it is the case.

Imagine we have a Lambertian ball that is rotating around the z-axis. Since we have a rotating ball, in the 2D motion field we will have the actual projected 3D motion i.e. we will have small arrows near the poles and

large arrows near the equator. But since the ball is rotating the ball actually is not changing the appearance in the image, so the optical flow field will be zero.

Now if we instead take a specular ball and move the light source across, then 2D motion field will be actually zero since the ball actually static. But the optical flow field will not be zero since in the image the highlight on the ball will be moving.

6.3.4 Optical Flow Field

The optical flow tells us something (maybe ambiguous) about: the 3d structure of the world, the motion of objects in the viewing area, and the motion of the observer if any. It is kind of a substitute for the motion field that we like to estimate.

6.3.5 Applications of Optical Flow Field

Video Interpolation / Frame Rate Adaption: Optical flow has many important applications in computer vision such as video interpolation of frame rate adaption if we know the image motion between two frames. If we know the relative motion by the flow field, we can take the first frame and warp it only halfway from time t to time $t_0.5$. If from these two images, we have estimated the flow, we can reduce the flow vectors or scale the flow vectors by one half, which allows us to synthesize an artificial frame in between.

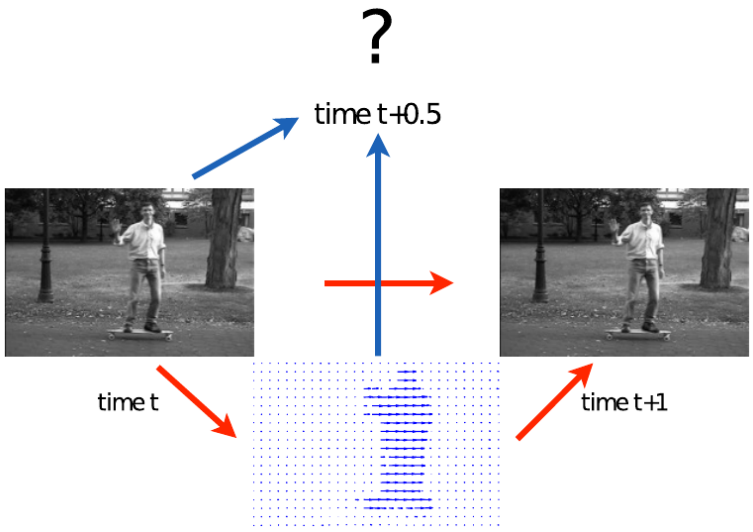


Figure 54: Can we synthesize new artificial frame if the optical flow is known?

Video Compression: Optical flows can also be used for video compression. For example, to compress an image sequence, new frames can be predicted using the optical flow field, and only store the optical flow field. Once the optical flow field is stored, the predictions are made by taking image at time t and warp it using the motion field or the optical flow field. Then, we fix the prediction by changing some of the pixels only. Because the flow fields are smooth, they are much easier to compress and store than storing the second image separately from the first image.

Autonomous Driving: In autonomous driving, combining disparity with optical flow produces 3D motion that is called a scene flow, which allows for tracking of which 3D points moving in which directions if the depths of the 3D points are known.

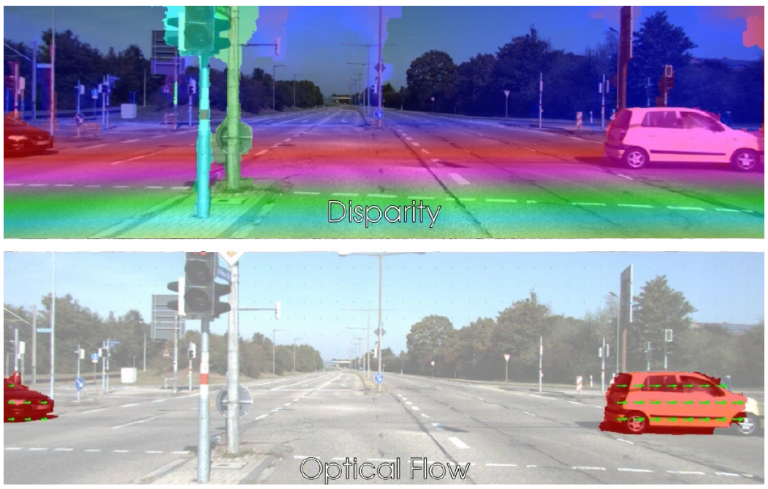


Figure 55: Optical flow in autonomous driving

6.3.6 Aperture Problem

The most famous problem in the context of optical flow is called the aperture problem. The aperture problem is such that if a single observation like a single pixel through a hole/aperture, is it possible to determine how things move? A single observation is not enough to determine optical flow, and that also makes sense mathematically. If only changes in a single pixel intensity is observed, it cannot determine the optical flow because optical flow has two unknowns (flow in the x-coordinate of the image plane, and then the y-coordinate of the image plane).

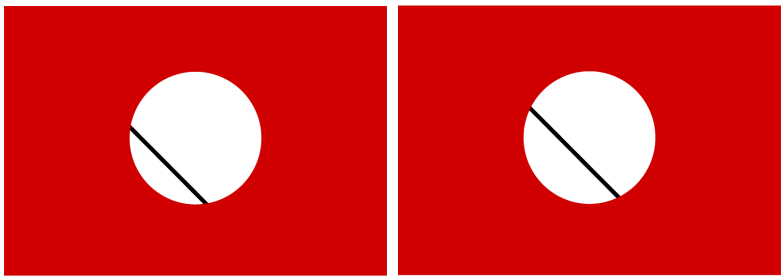


Figure 56: The aperture problem: in which direction is the line moving?

6.3.7 Determining Optical Flow: Horn-Schunck Optical Flow [24]

Consider the image I as a function of continuous variables x, y, t, x, y is the image domain and t is the time domain. Consider $u(x, y)$ and $v(x, y)$ as continuous flow fields (functions) in the image space and to determine u, v we minimize the following energy functional:

$$E(u,v) = \int\int \underbrace{(I(x+u(x,y), y+v(x,y), t+1) - I(x,y,t))^2}_{\text{quadratic penalty for brightness change}} + \lambda \cdot \underbrace{(\|\nabla u(x,y)\|^2 + \|\nabla v(x,y)\|^2)}_{\text{quadratic penalty for flow change}} dxdy$$

with regularization parameter λ and gradient $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$.

The first term is also called the brightness constancy assumption since we want the image or the intensity at x, y at time t to be the same as the intensity at $x + u(x, y), y + v(x, y)$ at time $t + 1$. Then because we have the aperture problem we also need a penalty for the flow change. Thus the second term is a regularizer without which the problem could not be optimized. Now minimizing this directly is a hard problem because the energy is highly non-convex and has many local optima. To solve this issue, we linearize the brightness constancy assumption. Therefore using first-order Taylor approximation, we have:

$$\begin{aligned} & I(x+u(x,y), y+v(x,y), t+1) \\ \overset{x,y,t}{\approx} \; & I(x,y,t) + I_x(x,y,t)(x+u(x,y)-x) + I_y(x,y,t)(y+v(x,y)-y) + I_t(x,y,t)(t+1-t) \\ = \; & I(x,y,t) + I_x(x,y,t)u(x,y) + I_y(x,y,t)v(x,y) + I_t(x,y,t) \end{aligned}$$

Thus, the optical flow energy functional is approximated by the following linearized equation:

$$E(u,v) = \int\int (I_x(x,y,t)u(x,y) + I_y(x,y,t)v(x,y) + I_t(x,y,t))^2 + \lambda \cdot (\|\nabla u(x,y)\|^2 + \|\nabla v(x,y)\|^2) dxdy.$$

Therefore, spatial discretization of the equation leads to the following discretized objective:

$$\begin{aligned} E(\mathbf{U},\mathbf{V}) &= \sum_{x,y}(I_x(x,y,t)u_{x,y} + I_y(x,y,t)v_{x,y} + I_t(x,y,t))^2 \\ &\quad +\lambda\cdot((u_{x,y}-u_{x+1,y})^2 + (u_{x,y}-u_{x,y+1})^2 + (v_{x,y}-v_{x+1,y})^2 + (v_{x,y}-v_{x,y+1})^2) \end{aligned}$$

This objective is quadratic in the flow maps $\boldsymbol{U},\boldsymbol{V}$ and thus has a unique optimum. But we cannot simply differentiate E w.r.t. $\boldsymbol{U},\boldsymbol{V}$ and set the gradient to 0 and solve for it, since this results in a huge but sparse linear system. Thus this problem can be solved using standard techniques such as Gauss-Seidel, SOR etc. However, this linearization works only for small motions, and as a solution we use iterative estimation & warping and coarse-to-fine estimation: Iteratively estimate, and make a step. If we have not converged, we make another step and relinearize around the current estimate. Sometimes, because even that is not enough for recovering large motions, we also typically do a coarse to fine estimation strategy where we start with a small image resolution and then perform the optical flow estimation there as the optical flow is not larger than a few pixels. After that, we proceed to the next higher resolution, and warp the target image based on the optical flow estimated at the lower resolution. This is used as an initialization because the starting point is better, so the optical flow has only the delta to be estimated, which is smaller and can be done using these iterative techniques.



Figure 57: **The result of the Horn-Schunck algorithm:** the right is a color-coded flow field where the color denotes the orientation, and the intensity denotes the strength of the optical flow.

The Horn & Schunck optical flow estimation are quite plausible. However, the flow is very smooth even if the transition between the two objects is not. The reason for over-smoothing is because we have to overcome the ambiguities by setting λ to a relatively high value to overcome the aperture problem. This trade-off poses a problem that we are assuming that the terms are all quadratic for convenience and easy to solve purpose. If there is a quadratic penalty for the change in optical flow between two objects that have different optical flows, the model is penalized heavily for making a sharp transition, and that causes the over-smoothing artifacts. Therefore, there has been a whole line of research that tried to formulate optical flow more robustly by using better penalties that are more aligned with the statistics of the real world.

6.3.8 Robust Estimation of Optical Flow

Probabilistic Interpretation

The HS optimization problem can be interpreted as MAP inference in a MRF:

$$p(\boldsymbol{U},\boldsymbol{V}) = \frac{1}{Z}\exp\left(-E(\boldsymbol{U},\boldsymbol{V})\right)$$

with the following Gibbs energy:

$$\begin{aligned} E(\boldsymbol{U},\boldsymbol{V}) &= \sum_{x,y}(I_x(x,y,t)u_{x,y} + I_y(x,y,t)v_{x,y} + I_t(x,y,t))^2 \\ &\quad +\lambda\cdot((u_{x,y}-u_{x+1,y})^2 + (u_{x,y}-u_{x,y+1})^2 + (v_{x,y}-v_{x+1,y})^2 + (v_{x,y}-v_{x,y+1})^2) \end{aligned}$$

Therfore, performing map inference on this distribution is the same as minimizing this energy. Also, since $\boldsymbol{U},\boldsymbol{V}$ are continuous, we solve inference with gradient descent (not BP).

Now, notice that the quadratic penalties translate to Gaussian distributions, i.e.:

$$\begin{aligned} p(\boldsymbol{U},\boldsymbol{V}) \quad \propto \quad & \prod_{x,y} \exp\{-(I_x(x,y,t)u_{x,y} + I_y(x,y,t)v_{x,y} + I_t(x,y,t))^2\} \\ & \times \exp\{-\lambda(u_{x,y} - u_{x+1,y})^2\} \times \exp\{-\lambda(u_{x,y} - u_{x,y+1})^2\} \\ & \times \exp\{-\lambda(v_{x,y} - v_{x+1,y})^2\} \times \exp\{-\lambda(v_{x,y} - v_{x,y+1})^2\} \end{aligned}$$

But these assumptions are invalid both for the brightness consistency constraint as well as for the smoothness constraint as we've seen gaussian distributions correspond to squared loss functions which are not robust to outliers or to this flow discontinuities. These outliers occur at object boundaries (violation of smoothness/regularizer) and at specular highlights (violation of photoconsistency/data term).

Robust Regularization

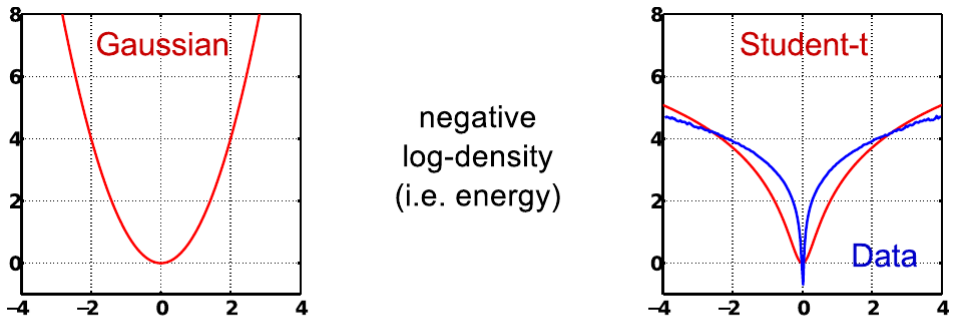
The solution to the above problem, can be mitigated by introducing a robust data term and smoothness penalties $\rho(\cdot)$:

$$\begin{aligned} p(\boldsymbol{U},\boldsymbol{V}) \quad \propto \quad & \prod_{x,y} \exp\{-\rho_D(I_x(x,y,t)u_{x,y} + I_y(x,y,t)v_{x,y} + I_t(x,y,t))\} \\ & \times \exp\{-\lambda\rho_S(u_{x,y} - u_{x+1,y})\} \times \exp\{-\lambda\rho_S(u_{x,y} - u_{x,y+1})\} \\ & \times \exp\{-\lambda\rho_S(v_{x,y} - v_{x+1,y})\} \times \exp\{-\lambda\rho_S(v_{x,y} - v_{x,y+1})\} \end{aligned}$$

Now we want a prior that allows for discontinuities in the optical flow and a likelihood that allows for outliers and occlusions. Therefore, we replace the Gaussian distributions with (heavy-tailed) Student-t distribution (=Lorentzian penalty):

$$p(x) \propto \left(1 + \frac{x^2}{2\sigma^2}\right)^{-\alpha} \implies \rho(x) = -\log p(x) = \alpha \log\left(1 + \frac{x^2}{2\sigma^2}\right).$$

Below we illustrate a comparison between Gaussian and robust Student-t penalty:



$$p(x) \propto \left(1 + \frac{x^2}{2\sigma^2}\right)^{-\alpha} \qquad \rho(x) = \alpha \log\left(1 + \frac{x^2}{2\sigma^2}\right)$$

The results show that the boundaries are much sharper compared to the original algorithm primarily because of the more precise assumptions.



Figure 58: Final inferred optical flow resulted from student t-distribution

6.3.9 End-to-End Deep Learning

Finally, optical flow has also been approached with end-to-end deep learning, which has recently succeeded to overcome classical methods. One of the reasons for this is that it is very challenging to generate ground truth data that could be used for deep models to learn from, which is why deep learning has taken a while in order to outperform the classical techniques.

The first technique **FlowNet**[12] which has not been able to outperform the classical techniques, but it demonstrates that optical flow estimation using supervised learning with large synthetic dataset is possible. It contains an encoder with strided convolutions, and a decoder with up-convolutions and skip connections. It uses a multi-scale loss curriculum learning, and a lot of synthetic training data because of many parameters.

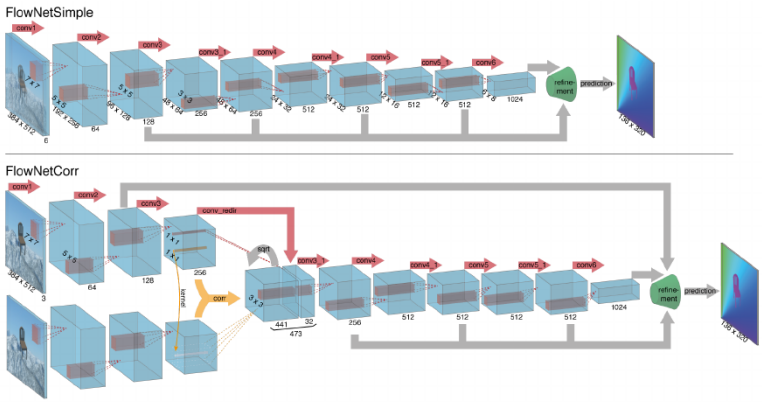


Figure 59: **FlowNet** Structure

FlowNet2[27]: an extension of FlowNet that is basically stacking multiple units togetherin order to handle different problems separately like large and small displacements, small flow vectors and fuses them together.

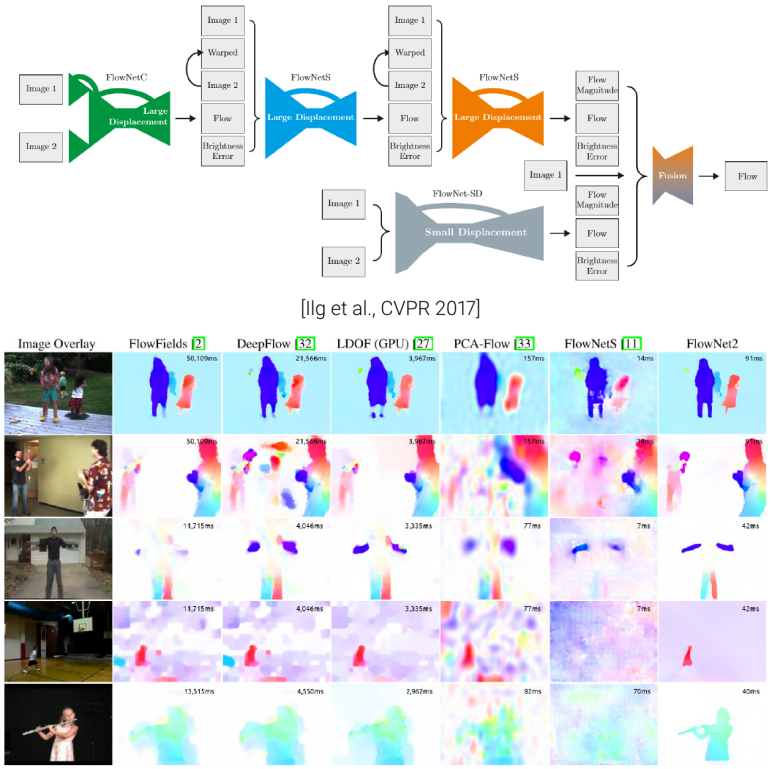


Figure 60: **FlowNet 2** structure and result

UnFlow[35]: Recently, it has been demonstrated that optical flow can also be trained in an unsupervised fashion by leveraging the principles and models that have been classical optical flow estimation techniques by predicting with a deep network the forward and backward flow followed by consistency checks and warping the images into each other using these estimated flow fields. By putting data loss and smoothness loss terms on the flow fields, we can train this model using just images without any optical flow ground truth which makes it possible to learn optical flow without supervision

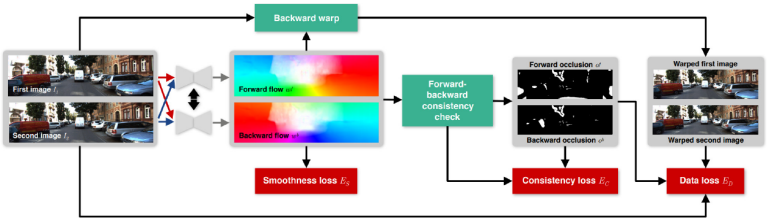


Figure 61: **UnFlow** Structure

In summary, classical optical flow approaches have been state of the art until 2016-2017, and deep learning based methods became on par or even better since 2017. However, they require big models with a lot of parameters, enormous amount of synthetic training data, a lot of GPU power, and sophisticated curriculum learning schedules where you train with simple data first and then go progressively harder. Nowadays, the state of the art is clearly deep learning because the datasets became better and more readily available including real data with optical flow ground-truth. Interestingly, the top performing deep learning methods borrow many elements from classical methods such as the idea of warping and iterative estimation, cost volume, coarse-to-fine estimation, and also similar loss functions.

7 Learning in Graphical Models

This chapter is about learning in graphical models (i.e. estimating the parameters given a data set). In Section 7.1, we introduce the Conditional Random Field. In Section 7.2, we discuss the actual learning problem - first in a simple setting where the parameters appear linearly or log linearly in the equations and where learning is a convex problem. Then, in Section 7.3, we talk about deep structured models where the parameters appear non-linearly in the model.

The following variable conventions are used throughout this chapter:

- N: number of training samples
- M: number of (output) variables
- C: number of classes/labels
- K: number of cliques/factors
- D: dimensionality of feature space

7.1 Conditional Random Fields

So far, we talked about Markov Random Fields or factor graphs. A quick reminder: In a Markov Random Field, we have a distribution over random variables:

$$p(x_1, \dots, x_{100}) = \frac{1}{Z} \exp \left\{ \sum_i \psi_i(x_i) + \lambda \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

In this example, there are 100 random variables. There is the exponential of a sum of log factors (or potentials). The naming "potentials" might be a bit confusing as it is used both in the logarithmic and in the original domain: Often, we call the factors the potentials; but we also call the log factors the potential. In the literature, these potentials are sometimes defined as the negative log factors. In this lecture, we define them as the log factors and omit the sign. This means that a high value of a potential will lead to a high value in the probability function.

So far, we discussed inference in Markov Random Fields. We were interested in estimating marginal distributions (for example, estimating $p(x_1) = \sum_{x \setminus x_1} p(x_1, \dots x_{100})$) or the Maximum-A-Posteriori (MAP) solution (that is, given a particular model, what configuration of x_1 to x_{100} is actually maximizing the probability: $x_1^*, \dots, x_{100}^* = \text{argmax}_{x_1, \dots, x_{100}} p(x_1, \dots x_{100})$).

Now, we talk about the learning problem: How can we estimate the parameters? In the example above, there is just one parameter λ. How can we estimate this λ from a data set?

What are Conditional Random Fields and why do we need them?

If we just define Markov Random Fields, we are only looking at one particular model instantiation. In the example of image denoising, we look at one particular image that we want to denoise. In this image, we associate each pixel with a random variable which yields the set of random variables $\mathcal{X}$. But if we want to do parameter learning, we want to learn from a larger data set - not just for a particular instance. This is why we need to formulate the problem conditionally.

This leads us to so-called structured output learning where we want to learn a function / a mapping $f_{\mathbf{w}} : \mathbb{X} \to \mathbb{Y}$. In structured output learning, the inputs $\mathcal{X} \in \mathbb{X}$ can be any kind of objects. Importantly, the outputs $\mathcal{Y} \in \mathbb{Y}$ are complex structured objects.

In a Conditional Random Field, we make the conditioning of the output $\mathcal{Y}$ on the input $\mathcal{X}$ and the parameters $\mathbf{w}$ explicit:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z} \exp \left\{ \sum_i \psi_i(\mathcal{X}, y_i) + \lambda \sum_{i \sim j} \psi_{ij}(\mathcal{X}, y_i, y_j) \right\}$$

Instead of writing just the distribution on $\mathcal{X}$, we now write a distribution of $\mathcal{Y}$ where the $\mathcal{Y}$ is now the output. (Note: Be careful with the variable names! MRF notation: outputs $\mathcal{X} \in \mathbb{X}$. CRF notation: inputs $\mathcal{X} \in \mathbb{X}$, outputs $\mathcal{Y} \in \mathbb{Y}$)

So we have a model that, given a particular $\mathcal{X}$ (could be a noisy image), produces an output $\mathcal{Y}$ (the denoised image). We make this input image $\mathcal{X}$ and the parameters $\mathbf{w}$ explicit in order to be able to apply this model on an entire data set where we want to learn these mappings from $\mathbb{X}$ to $\mathbb{Y}$ (from noisy images to denoised images).

As a reminder: In the MRF notation

$$p(\mathcal{X}) = \frac{1}{Z} \exp \left\{ \sum_i \psi_i(x_i) + \lambda \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

we haven't explicitly encoded a noisy image. This was implicitly encoded in the definition of, for example, the unary potentials.

Learning means: Estimating the parameters of the model (in this case, just λ) given a data set of input-output-pairs: $\mathcal{D} = \{(\mathcal{X}^1, \mathcal{Y}^1), \ldots, (\mathcal{X}^N, \mathcal{Y}^N)\}$. Examples: 1) Denoising: noisy inputs and denoised outputs. 2) Semantic segmentation: input images $\mathcal{X}$ and semantic segmentation maps $\mathcal{Y}$. So it is a supervised learning problem where we always consider input-output-pairs. We want to infer what is the optimal parameter such that, given a novel input that we haven't seen during training, we do the best possible prediction under that model.

Coming from this specific example of image denoising, we can write it more general. We can concatenate all the features in a long vector and multiply this with the parameters. Therefore, the general form can simply be written with the inner product:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z(\mathcal{X}, \mathbf{w})} \exp\{\langle \mathbf{w}, \psi(\mathcal{X}, \mathcal{Y}) \rangle\}$$

This general form is much more flexible than just having a single parameter because we can have one parameter for each feature that is defined by the graphical model. The goal is to learn or estimate the parameters $\mathbf{w}$ from a given annotated data set with input-output-pairs $\mathcal{X}^1, \mathcal{Y}^1), \ldots, (\mathcal{X}^N, \mathcal{Y}^N)$.

Clarifying the components:

- **Feature function:** $\psi(\mathcal{X}, \mathcal{Y}) : \mathbb{X} \times \mathbb{R}^M \to \mathbb{R}^D$ (concatenates potentials/features)
  Graphical model specifies decomposition of ψ into potentials (=log factors) ψ_k:

$$\psi(\mathcal{X}, \mathcal{Y}) = \big(\psi_1(\mathcal{X}, \mathcal{Y}_1), \ldots, \psi_K(\mathcal{X}, \mathcal{Y}_K)\big)$$

- **Parameter vector:** $\mathbf{w} \in \mathbb{R}^D$ (M: number of output nodes, D: dimensionality of feature space)
- **Partition function:** $Z(\mathcal{X}, \mathbf{w}) = \sum_{\mathcal{Y}} \exp\{\langle \mathbf{w}, \psi(\mathcal{X}, \mathcal{Y}) \rangle\}$

Note: Naming "Conditional Random Field" because we model conditional distributions. We model the distribution of the output conditioned on the input $\mathcal{X}$ (and the weights ($\mathbf{w}$)).

7.2 Parameter Estimation

How can we estimate the parameters of a Conditional Random Field? Our goal is to maximize the likelihood of the outputs $\mathcal{Y}$ conditioned on the inputs $\mathcal{X}$ with respect to the parameter vector $\mathbf{w}$. As often the case in machine learning, we assume that the data is independent and identically distributed (IID) - in order for the likelihood to factorize. Mathematically, we want to find the parameter vector $\hat{\mathbf{w}}_{ML}$ that maximizes the probability of $\mathcal{Y}$ given $\mathcal{X}$ and $\mathbf{w}$:

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w} \in \mathbb{R}^D}{\operatorname{argmax}} \; p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) \quad \text{with} \quad p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \prod_{n=1}^{N} p(\mathcal{Y}^n|\mathcal{X}^n, \mathbf{w})$$

(Index n runs over the entire data set. N is the number of annotated images that we have.) For the log linear models that we consider in this unit, this optimization problem is convex or semiconvex. So we find a global optimum.

In other words, we want to find the parameter vector $\hat{\mathbf{w}}_{ML}$ such that the model distribution is as similar as possible to the data distribution: $p_{model}(\mathcal{Y}|\mathcal{X}, \hat{\mathbf{w}}_{ML}) \approx p_{data}(\mathcal{Y}|\mathcal{X})$.

This is equivalent to minimizing the negative conditional-log-likelihood:

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w} \in \mathbb{R}^D}{\operatorname{argmin}} \; \mathcal{L}(\mathbf{w}) \quad \text{with} \quad \mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N} \log p(\mathcal{Y}^n|\mathcal{X}^n, \mathbf{w})$$

Note: The loss function $\mathcal{L}(\mathbf{w})$ is the negative log probability. Because the negative logarithm of the product is the same as the negative sum of the logarithm, we have rewritten this here in terms of summations instead of products.

Rewriting the loss function such that we can minimize it wrt. $\mathbf{w}$:

$$
\begin{aligned}
\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N} \log p(\mathcal{Y}^n|\mathcal{X}^n, \mathbf{w}) && \text{| substitute distr. with CRF def} \\
&= -\sum_{n=1}^{N} \left[\log \frac{1}{Z(\mathcal{X}^n, \mathbf{w})} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle\right\} \right] && \text{| } \log(a \cdot b) = \log(a) + \log(b) \\
&= -\sum_{n=1}^{N} \left[\log \frac{1}{Z(\mathcal{X}^n, \mathbf{w})} + \log(\exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle\right\}) \right] && \text{| } \log \frac{1}{a} = \log a^{-1} = -\log(x) \\
&= -\sum_{n=1}^{N} \left[-\log Z(\mathcal{X}^n, \mathbf{w}) + \log(\exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle\right\}) \right] && \text{| } \log(\exp\{a\}) = a \\
&= -\sum_{n=1}^{N} \left[-\log Z(\mathcal{X}^n, \mathbf{w}) + \langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle \right] && \text{| reorder terms} \\
&= -\sum_{n=1}^{N} \left[\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle - \log Z(\mathcal{X}^n, \mathbf{w}) \right] && \text{| substitute } Z(\mathcal{X}^n, \mathbf{w}) \text{ with def} \\
&= -\sum_{n=1}^{N} \left[\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle - \log \sum_{\mathcal{Y} \in \mathbb{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\} \right]
\end{aligned}
$$

What did we do here? We simply substituted the conditional probability through its CRF form that we have defined in 7.1. Then, we applied the logarithm rules and wrote the partition function out.

How can we optimize this? It is clear that there is no closed form solution. It is clearly not quadratic. So we are going to use gradient descent.

How does gradient descent work? In its simplest form, we pick a step size η and a tolerance ϵ. Then we initialize the parameters $\mathbf{w}^0$. Then we repeat until $\|\mathbf{v}\| < \epsilon$ (convergence): we compute the gradient of the loss function $\mathbf{v} = \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ and then we do an update on the parameters $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \mathbf{v}$. There are various variants. There is, for example, line search where, instead of just picking a step size, we try to query the optimal step size by going into the direction of the gradient until we find the minimum. There is also conjugate gradients which provides a better direction for the next gradient step. What is common to all these algorithms is that they all require gradients. Some of them even require the function evaluation, like the line search. So, we have to be able to evaluate the loss function. And we also have to be able to compute the gradient of that loss function in order to execute the gradient descent algorithm. So we have to compute two things: the loss function / the negative conditional log likelihood

$$
\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N} \left[\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle - \log \sum_{\mathcal{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\} \right]
$$

and the gradient wrt. the parameters $\mathbf{w}$

$$
\begin{aligned}
\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N} \left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \frac{\sum_{\mathcal{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\} \psi(\mathcal{X}^n, \mathcal{Y})}{\sum_{\mathcal{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\}} \right] \\
&= -\sum_{n=1}^{N} \left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}} \frac{\exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\}}{\sum_{\mathcal{Y}'} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}')\rangle\right\}} \psi(\mathcal{X}^n, \mathcal{Y}) \right] \\
&= -\sum_{n=1}^{N} \left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w}) \psi(\mathcal{X}^n, \mathcal{Y}) \right] \\
&= -\sum_{n=1}^{N} \left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \mathbb{E}_{\mathcal{Y} \sim p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})} \psi(\mathcal{X}^n, \mathcal{Y}) \right]
\end{aligned}
$$

When is the loss function minimal? If the expectation of the features under the model is equal to the features for the labeled example, the difference in the square brackets will vanish and the gradient will be zero:

$$
\mathbb{E}_{y \sim p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})} \psi(\mathcal{X}^n, \mathcal{Y}) = \psi(\mathcal{X}^n, \mathcal{Y}^n) \Rightarrow \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = 0
$$

In this case, we have found a critical point. The interpretation is that we aim at expectation matching. We aim at matching the expectation of the features under the model to the features of the observation (informally

abbreviated with $\mathcal{Y}^{\text{obs}}$). We try to do this discriminatively only for the $\mathcal{X}$ that are available in the training set $\{\mathcal{X}^1, \ldots, \mathcal{X}^N\}$.

Sidenote: The loss function $\mathcal{L}(\mathbf{w})$ is convex because the Hessian of it is positive semi-definite. This means that if for this particular model / this particular CRF definition, the gradient is zero $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 0$ (for example by starting at some random initial guess and performing gradient descent and arriving at a critical point where the gradient is zero), then we have reached the global optimum. However, this is only true if we use this very special definition of the Conditional Random Field; in particular if this probability $p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$ is log linear in $\mathbf{w}$. In 7.3, we will also see non-linear models where this is not the case. If we apply gradient descent in those cases, we do not necessarily arrive at the global optimum but just at a local minimum.

For gradient descent we must evaluate $\mathcal{L}(\mathbf{w})$ and $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$. The problem is that the state space $\mathbb{Y}$ is typically very (exponentially) large. If we want to just naively compute the sums over the entire state space of $\mathbb{Y}$, this is completely intractable. As an example, if we consider binary image segmentation of a VGA resolution image (640 pixel wide and 480 pixel high), then there is $|\mathbb{Y}| = 2^{640\times480} \approx 10^{92475}$ possible solutions. So the sum is a sum over 10^{92475} terms. So this is not possible! We need to exploit the structure of graphical models in $\mathbb{Y}$ to make this tractable - similar to what we did for the intractable inference problem in graphical models in chapter 5.

In order better understand the computational complexity of computing these two quantities $\mathcal{L}(\mathbf{w})$ and $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$, let's write down the computational complexity precisely: $O(NC^M D)$

- N: number of samples in dataset ($\approx$ 100 to 1,000,000).
  In order to evaluate the loss or the gradient, we have to compute the sum over N many terms. So we have N appearing linearly.
- M: number of output nodes ($\approx$ 100 to 1,000,000).
  In the previous case, we had 640 by 480 pixels, so 640 by 480 variables or output nodes. This is the main problem in the computation.
- C: maximal number of labels per output node ($\approx$ 2 to 100).
  In the case of binary segmentation, this would be $C = 2$. But it could be much more; for instance, for semantic segmentation: if we have 100 different semantic classes, this would be $C = 100$.
- D: dimensionality of feature space ψ.
  This occurs in the complexity term because we have to compute the sum (in the gradient expression) always in terms of all the features. The larger the feature space, the more we have to compute. In the case of the computation of the loss function, it also appears because: with a larger dimensionality D, also the inner product becomes more and more expensive to compute because it's a product of larger vectors.

How can we reduce this computational complexity? The first point that we are going to address is the most important one. This is the one that makes the entire problem either tractable or intractable. It is the complexity that is introduced by summing over the entire state space $\mathbb{Y}$. In the case of binary segmentation, we have $2^{640\times480}$ possible combinations in these two sums:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N}\left[\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle - \log \sum_{\mathcal{Y}\in\mathbb{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\}\right] \\
\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N}\left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}\in\mathbb{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y})\right]
\end{aligned}$$

In order to solve this problem, we exploit the structure of graphical models. Remember, in a graphical model, the features and weights decompose:

$$\psi(\mathcal{X}, \mathcal{Y}) = (\psi_1(\mathcal{X}, \mathcal{Y}_1), \ldots, \psi_K(\mathcal{X}, \mathcal{Y}_K)) \qquad \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_K)$$

So, this long feature vector is just a concatenation of smaller feature vectors. Therefore, the partition function simplifies:

$$\begin{aligned}
\sum_{\mathcal{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\} &= \sum_{\mathcal{Y}} \exp\left\{\sum_k \langle \mathbf{w}_k, \psi_k(\mathcal{X}^n, \mathcal{Y}_k)\rangle\right\} \\
&= \sum_{\mathcal{Y}} \prod_k \underbrace{\exp\left\{\langle \mathbf{w}_k, \psi_k(\mathcal{X}^n, \mathcal{Y}_k)\rangle\right\}}_{\text{k'th factor}}
\end{aligned}$$

Now, we have local potentials that do not depend anymore on the entire output space $\mathbb{Y}$ but just on a subset of that. In the case of pairwise potentials, that's just two variables (two adjacent pixels) instead of all the pixels in the image. This is indicated by this subscript k: $\mathcal{Y}_k$. We don't make any restriction or assumption here in terms of the input. The input is still $\mathcal{X}^n$. This potential could still depend on all the input variables. This doesn't matter. What matters here is the complexity with respect to the output variables.

Now we know: If these factors are of tractable order (such as in the case of the stereo problem we've discussed with pairwise and unary factors), then we can efficiently calculate this entire expression, using message passing. Using message passing, we can compute marginals. By summing over any of these unnormalized marginals, we get the partition function.

Similarly, the feature expectation simplifies as well:

$$\begin{aligned}
\sum_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y}) &= \mathbb{E}_{\mathcal{Y}\sim p(\mathcal{Y}|\mathcal{X}^n,\mathbf{w})}\psi(\mathcal{X}^n, \mathcal{Y}) && \text{| feature decomposition} \\
&= \mathbb{E}_{\mathcal{Y}\sim p(\mathcal{Y}|\mathcal{X}^n,\mathbf{w})} \sum_k \psi(\mathcal{X}^n, \mathcal{Y}) && \text{| feature only depend on k} \\
&= \sum_k \mathbb{E}_{\mathcal{Y}_k\sim p(\mathcal{Y}_k|\mathcal{X}^n,\mathbf{w})}\psi_k(\mathcal{X}^n, \mathcal{Y}_k) && \text{| substitute expectation depending only on k} \\
&= \underbrace{\sum_k}_{\text{K}} \underbrace{\sum_{\mathcal{Y}_k}}_{C^F} \underbrace{p(\mathcal{Y}_k|\mathcal{X}^n, \mathbf{w})}_{\text{marginal}} \underbrace{\psi_k(\mathcal{X}^n, \mathcal{Y}_k)}_{\text{feature k}}
\end{aligned}$$

where capital K is the number of potentials, C the maximum number of labels which remains unchanged and F is the largest order of the potentials. So the marginals can be calculated in polynomial time using the belief propagation algorithm (BP).

$\mathcal{Y}_k$ is the subset of the variables that the feature or potential depends upon. We see that the computation has dramatically simplified in terms of complexity: we have to sum just over this state space $\mathcal{Y}_k$ which is much smaller than the entire state space of all the variables because this is only some of the variables (one, two or three; not all the output variables). Before summing this up, we have to compute the marginals for a particular configuration of nodes. We know that these marginals $p(\mathcal{Y}_k|\mathcal{X}^n, \mathbf{w})$ can be computed for tractable graphical models with not too high order cliques efficiently (e.g. using belief propagation). So we have an efficient way of computing these marginals. Then we need to sum up over the product of these marginals with the features. And we have to do this for all the potentials in the graphical model. In the sum over $\mathcal{Y}_k$, we only have C^F terms (C: max. number of labels, F: largest order).

What does that mean in terms of the computational complexity? Compared to the computational complexity that we had before ($O(NC^M D)$ where M was very large), now we have $O(NKC^F D)$ where (this is important) F (the order of the largest factor; typically $2-3$) is much smaller than the number of all the output nodes. So this becomes tractable because K (the number of factors) is just linear and F is small. In other words, we have exploited efficient inference algorithms in graphical models in order to calculate the loss function and the gradient of the loss function much more efficiently.

Since N also occurs linearly in the complexity term, learning on large data sets becomes problematic because processing all N training samples for one gradient update is slow and often it doesn't even fit into memory (e.g. if you use GPUs as in deep learning). How can we estimate the parameters in this setting? One way would be to simplify the model to make the gradient updates faster. But if we simplify the model, then the model is less accurate. So the results get worse. That's not what we want to sacrifice. Another strategy would be to train the model on a subsampled data set. But this clearly ignores some of the information in the data set and is not ideal either. We could also parallelize across CPUs and GPUs, but that doesn't really save computation. It just makes things run in parallel. So what we can do (what we also did in deep learning) is to use stochastic gradient descent.

In each gradient step of the stochastic gradient descent, we create a random subset $\mathcal{D}' \subset \mathcal{D}$. This random subset is typically $\mathcal{D}' \leq 256$; so, relatively small compared to the size of the entire data set. Then, we follow the approximate gradient where the approximation comes in by summing not over all the data samples, but summing over only this small subset of the data samples:

$$\nabla_{\mathbf{w}} \approx - \sum_{(\mathcal{X}^n,\mathcal{Y}^n)\in\mathcal{D}'} \left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \mathbb{E}_{\mathcal{Y}\sim p(\mathcal{Y}|\mathcal{X}^n,\mathbf{w})}\psi(\mathcal{X}^n, \mathcal{Y})\right]$$

In this case, line search is no longer possible. So we have to introduce this extra step-size hyper-parameter η that allows us to go forward along the gradient for a fixed step-size in stochastic gradient descent. It can be shown that SGD converges to (at least) a local minimum of the loss function $\operatorname{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ if η is chosen right.

In our particular case here of these log linear models, it even converges to a global minimum. SGD needs more iterations but requires less memory, and each of these iterations is faster.

The final term in the complexity expression is the D (the dimensionality of the feature space). If the dimensionality of the feature space is extremely large (e.g. 1000000), then this also contributes significantly to the runtime of computing the loss and the gradient.

Summary of computational optimizations

Problem	Solution	Method
$\lvert\mathbb{Y}\rvert$ too large	exploit structure (inference)	belief propagation
N too large	mini-batches	stochastic gradient descent
D too large	trained unary classifiers ψ	piece-wise training

Applications

In order to get a feeling of what these feature spaces typically look like, we look at some concrete applications in the following.

The first application is **semantic segmentation** [49] where the input is an image and the output is a label map. This could be just binary segmentation (foreground versus background) or it could be multi-class segmentation. In this case, if we specify this problem as a Conditional Random Field, we associate with every spatial variable y_i (let's say, every pixel) a local image feature $\psi_i(\mathcal{X}, y_i) \in \mathbb{R}^{\approx 1000}$ that describes how this pixel looks or how it relates to a certain semantic class label. For example, if there is a green region, that is unlikely to be a horse. But if there's a brown region around, that pixel is more likely to be a horse. So these are local features that can be extracted using hand-engineered features (such as bag of words or deep features) and are typically in the order of a few thousand. So we have to compute $\langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i) \rangle$. You can think of this as a local classifier like in logistic regression where we also take this inner product. Then, we have another term here that is a test (in this particular example here) for the same label: $\psi_{ij}(y_i, y_j) = [y_i = y_j] \in \mathbb{R}^1$. So it could, for example, take value 1 if the labels are the same (if you want to maximize these) and 0 if they are not the same. In Fig. 118, you can see what happens if we do just local classification versus introducing the smoothness term: A lot of this noise that is present by the local classifiers gets removed: We penalize for label changes (if $w_{ij} > 0$): $\langle w_{ij}, \psi_{ij}(y_i, y_j) \rangle$. But at the same time, the foreground region gets over-smoothed a little. So this combination of local and smoothness terms ($\text{argmax}_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$) leads to a smooth version of what we would classify with the local features alone.

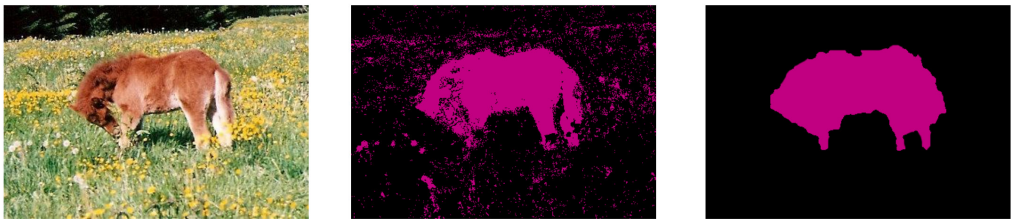


Figure 62: **Semantic segmentation.** Comparing the results for semantic segmentation with and without inducing a smoothness term.

Another example is **handwriting recognition** where we have a sequence of characters (images) and we want to classify these characters. An image is represented by $\psi_i(\mathcal{X}, y_i) \in \mathbb{R}^{\approx 1000}$ (e.g., pixels, gradients); using $\langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i) \rangle$ as local classifier for letters. Maybe we have certain prior knowledge about which characters occur after each other. That could be encoded by such pairwise potentials: $\psi_{ij}(y_i, y_j) = \mathbf{e}_{y_i} \mathbf{e}_{y_j}^\top \in \mathbb{R}^{26 \times 26}$ (a matrix with one element set to 1). $\langle \mathbf{w}_{ij}, \psi_{ij}(y_i, y_j) \rangle$ then encourages/suppresses letter combinations. Of course, if we have such knowledge extracted from a big text database, then we can more robustly solve this problem. We can overcome some of the uncertainties that might be present in the local representations if the character cannot be uniquely identified by just its local surrounding (in terms of its appearance). So, the combination of the local and the pairwise term delivers a corrected version of the local cues: $\text{argmax}_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$. A possible result is illustrated in Fig. 63.

Another example is **pose estimation**. Again, the input is an image but now we want to infer the human body pose of a person. So again, we associate with each pixel a random variable y_i and that random variable y_i indicates the body part that is associated with that pixel. So again, an image is represented by $\psi_i(\mathcal{X}, y_i) \in \mathbb{R}^{\approx 1000}$. We use either HoG or deep features in order to extract local features at each pixel, obtaining $\langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i) \rangle$ as local confidence map. But if we do this alone and we use this for classification, we get a lot

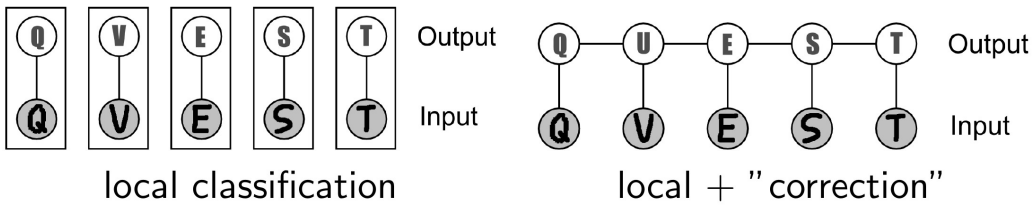


Figure 63: **Handwriting Recognition.** Comparing the results for handwriting recognition with and without prior knowledge about character sequences.

of ambiguities. This is illustrated in Fig. 64, middle. But if we know how certain body parts are aligned with respect to each other (for example, it is more likely that the head of a person is on top of the body), then we can use this and include this into the Conditional Random Field in order to obtain a sanitized version of the local cues. $\psi_{ij}(y_i, y_j) = \text{fit}(y_i, y_j) \in \mathbb{R}^1$ can be used as a test for geometric fit / as a pose prior, with $\langle w_{ij}, \psi_{ij}(y_i, y_j)\rangle$ penalizing for unrealistic poses. If we combine them by argmax $_{\mathcal{Y}}\, p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$, this yields a sanitized version of local cues. This is shown in Fig. 64 on the right where a lot of this uncertainty has been removed and plausible classification into body parts has been established.



Figure 64: **Pose estimation.** [14] Comparing the results for pose estimation with and without prior knowledge about body geometry.

In summary, typical feature functions for CRFs in computer vision include unary terms $\psi_i(\mathcal{X}, y_i)$ (which are local representation and often high dimensional; so you can think of this as local classifiers: $\langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i)\rangle$) and pairwise terms $\psi_{ij}(y_i, y_j)$ (which encode some prior knowledge, but are typically rather low dimensional). With the pairwise terms, we can penalize inconsistencies: $\langle w_{ij}, \psi_{ij}(y_i, y_j)\rangle$. Not only the unary terms can depend on $\mathcal{X}$, but also the pairwise terms can depend on $\mathcal{X}$: $\psi_{ij}(\mathcal{X}, y_i, y_j)$.

During learning, we want to adjust the parameters. We want to learn these local linear classifiers (the unary weights $\mathbf{w}_i$) and we want to learn the pairwise weights w_{ij} (i.e. learning the importance of, for example, smoothing in terms of obtaining a smooth semantic segmentation as an output). The argmax $_{\mathcal{Y}}\, p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$ is a cleaned up version of the local prediction by utilizing our prior knowledge that is encoded in these pairwise or higher order terms.

However, sometimes training this entire model is not easy because of the feature dimensionality. For example, often it is not so easy to specify features or terms where the parameters are linear in the features. Therefore, we often need very high dimensional feature vectors. If we have high dimensional feature vectors, learning can be very slow. In order to overcome this problem of high dimensionality of this D, we can use piecewise training where we first pre-train classifiers $p(y_i|\mathcal{X})$ and then set the potential as a one-dimensional function $\psi_i(\mathcal{X}, y_i) = \log p(y_i|\mathcal{X}) \in \mathbb{R}$ and learning a one-dimensional weight per classifier: $\langle w_i, \psi_i(\mathcal{X}, y_i)\rangle$. The advantage of this is that lower dimensional feature vectors, during training, lead to faster training and also faster inference. The second advantage is that these classifiers $\log p(y_i|\mathcal{X})$ can be stronger than just the linear classifier as we have to find it in the Conditional Random Field (e.g. it can be a non-linear SVM or a CNN etc.). However, the disadvantage of this strategy is that if these local classifiers are actually bad, the CRF training can also not fix this. In other words, the features that we extract from this must still be meaningful in order to solve our problem.

Summary of this chaper. Given a training set $\mathcal{D} = \{(\mathcal{X}^1, \mathcal{Y}^1), \ldots, (\mathcal{X}^N, \mathcal{Y}^N)\}$ with IID drawn data $(\mathcal{X}^n, \mathcal{Y}^n) \overset{\text{i.i.d.}}{\sim} p_{data}(\mathcal{X}, \mathcal{Y})$ and a feature function from input and output space to some feature space $\psi(\mathcal{X}, \mathcal{Y}) : \mathbb{X} \times \mathbb{R}^M \to \mathbb{R}^D$, the task is to find the parameter vector $\hat{\mathbf{w}}_{ML}$ such that the model distribution becomes similar to the data distribution under that training set: $p_{model}(\mathcal{Y}|\mathcal{X}, \hat{\mathbf{w}}_{ML}) = \frac{1}{Z(\mathcal{X}, \hat{\mathbf{w}}_{ML})} \exp\{\langle \hat{\mathbf{w}}_{ML}, \psi(\mathcal{X}, \mathcal{Y})\rangle\} \approx p_{data}(\mathcal{Y}|\mathcal{X})$.

In order to do so, we minimize the negative conditional log likelihood:

$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left[\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y}^n)\rangle - \log \sum_{\mathcal{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\}\right]$$

In the case of log linear models, this is a convex optimization problem. So we know that gradient descent leads to a global optimum. We have also seen that training needs repeated runs of probabilistic inference. Therefore, this inference must be fast because we have to do it at every iteration of the gradient descent algorithm. We have also seen how we can make it fast. For example, how we can solve the problem of this exponentially large state space $|\mathbb{Y}|$ that we have to sum over by exploiting the structure of the problem using graphical models and, for example, using the belief propagation for computing marginal distributions. We have seen that we can overcome the problem of very large data sets (i.e. N being to large) by exploiting mini batches and stochastic gradient descent. Additionally, we have seen that very large features (i.e. D being too large) can be overcome by piece-wise training.

7.3 Deep Structured Models

So far, we discussed log-linear models:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z(\mathcal{X}, \mathbf{w})} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}, \mathcal{Y})\rangle\right\}$$

Log-linear models are models where the parameters $\mathbf{w}$ appear in a log-linear fashion in the model equation / in the probability distribution. This assumption severely limits these type of models because the features must already be very powerful. They must already do the heavy lifting. The parameter vector can only adjust the influence of these different features in a linear fashion. It cannot do anything sophisticated other than linearly re-weighting these features. What we ideally like to have is the feature functions themselves being parameterized with parameters, where these parameters do not have to appear in a linear fashion, but these feature functions could, for example, be little neural networks where the parameters don't appear linearly. We want to update not only the relative weighting of these features as in these log-linear models but we also want to update these parameters of the feature functions themselves jointly by training (by maximizing the likelihood). This is called deep structured models where the parameters don't need to appear just linearly in the expression:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z(\mathcal{X}, \mathbf{w})} \exp\left\{\psi(\mathcal{X}, \mathcal{Y}, \mathbf{w})\right\}$$

Here, the potential functions are directly parameterized via $\mathbf{w}$. This results in a much more flexible model (ψ, can represent e.g., a neural network). (Note: The representation for deep structured models above is just a super-set of the representation for log-linear models above because if we specify the potential functions of the (more general) deep structured model, they could be just linear ones. But in general, we're going to be interested now in non-linear dependencies on $\mathbf{w}$.)

Similarly to the derivation of the negative log-likelihood and the gradient of the negative log-likelihood in 7.2 for log-linear models, here, for this more general class of deep structured models, we can also derive the negative log-likelihood and its gradient:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N}\left[\psi(\mathcal{X}^n, \mathcal{Y}^n, \mathbf{w}) - \log \sum_{\mathcal{Y}} \exp\left\{\psi(\mathcal{X}^n, \mathcal{Y}, \mathbf{w})\right\}\right] \\
\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N}\left[\nabla_{\mathbf{w}}\psi(\mathcal{X}^n, \mathcal{Y}^n, \mathbf{w}) - \sum_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\nabla_{\mathbf{w}}\psi(\mathcal{X}^n, \mathcal{Y}, \mathbf{w})\right]
\end{aligned}$$

(Differences to the log-linear model are highlighted in red.)

Again, the sum can be efficiently computed as the features decompose:

$$\psi(\mathcal{X}, \mathcal{Y}, \mathbf{w}) = \big(\psi_1(\mathcal{X}, \mathcal{Y}_1, \mathbf{w}), \ldots, \psi_K(\mathcal{X}, \mathcal{Y}_K, \mathbf{w})\big)$$

Sidenote: Here, we included the dependency on the parameters as an argument of the function. Often, when we talk about neural networks or parameterized functions in general, we include the dependency as a subscript. But we mean the same. It is just more convenient here, because we have another subscript of these potentials, to include the dependency as an argument as well.

So we do assume that there is a graphical model that makes inference (which is required for training) tractable and that the potentials decompose into a sum of potentials where each of them can depend on the parameters.

In order to perform a gradient update step in the context of a gradient descent or stochastic gradient descent algorithm, we need to compute the gradient and the negative log-likelihood. To this end, we can exploit again the efficiency of inference in graphical models if the graphical model is tractable to compute these expressions efficiently. However, differently from before, in order to compute these two quantities, we also have to do inference in the neural networks (if we assume that these feature functions are specified in neural networks) in order to compute the value of these neural networks for a particular input and parameter configuration as well as the gradient of these neural networks with respect to the parameters $\mathbf{w}$. From deep learning we know that in order to compute the output for a particular input, we have to apply the forward algorithm / the forward pass. In order to compute the gradients, we can compute the backpropagation algorithm. These two algorithms are tractable because they heavily utilize the principle of dynamic programming (reusing computation and storing intermediate information in the forward pass to compute the quantities in the backward pass efficiently). However, it is more complex now. Before, we just had to compute the inner product between the features and the parameters. Now, we have to do inference in, potentially, very deep neural networks in order to compute the features and the gradients of these features.

The algorithm: We compute a forward pass in order to get the feature values $\psi_k(\mathcal{X}, \mathcal{Y}_k, \mathbf{w})$. Then we compute a backward pass to obtain the gradients $\nabla_{\mathbf{w}}\psi(\mathcal{X}^n, \mathcal{Y}, \mathbf{w})$. Then we can compute the marginals using message passing. With those, the gradient of the negative log-likelihood and the negative-log likelihood, we update the parameters $\mathbf{w}$. The problem of this approach is that this becomes even slower than before. Learning in a Conditional Random Field already is not super fast. But now, instead of just multiplying a weight vector with a feature vector, we have to compute a forward and a backward pass through these deep neural networks at every iteration of the gradient descent algorithm and for every potential (unless they have shared weights). In each iteration, the forward and the backward pass are required to calculate the features and the gradients for this graphical model inference. Therefore, this becomes really slow. However, there are some alternatives. One alternative is to interleave learning and inference. This has been discussed in [6]. It makes learning faster but it's still comparably slow. The applications that have been tackled in this line of works are still relatively simple compared to the applications we are typically interested in. So it is a line of work that hasn't been as fruitful. Another alternative that is more heavily used today is so-called unrolled inference. This is much simpler (leads also to a much simpler algorithm) but we loose the probabilistic interpretation. We are going to discuss this in the remainder of this chapter.

Inference Unrolling

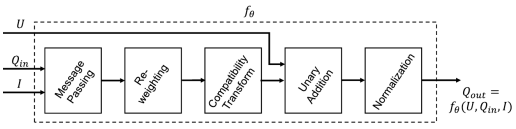
The idea in unrolled inference is to consider the inference process (for example, sum product belief propagation) in a graphical model as a sequence of small computations for which we can calculate a computation graph (as we write down a computation graph for a particular deep learning architecture). Then, we unroll a fixed number of inference iterations (similar to unrolling of an RNN during learning). When we do inference in a graphical model, typically, we don't know how many iterations we have to do until convergence, depending on the algorithm and the approximations that we have assumed. Nevertheless, here we say: for a fixed number of iterations, we want to train a model that, for this number of iterations, produces the best output. When unrolling this inference algorithm for a fixed number of iterations, we can compute the gradients simply by using automatic differentiation.

Some remarks: With this idea of inference unrolling for a fixed number of iterations, we are in the regime of empirical risk minimization. So it's a purely deterministic approach where we give up the probabilistic viewpoint. We cannot trust that our quantities that we compute really do have a probabilistic interpretation. So, for this inference algorithm that we unroll for a fixed number of iterations, we want to update the parameters in a way that for this particular instantiation of this algorithm, on expectation, we get the best prediction (the best maximum a posteriori prediction), under that model given the trainings data set that we have. The advantage of this is that this is often fast enough for efficient training in deep models. Because compared to belief propagation where we have to do the forward-backward-pass in order to compute the gradients for stochastic gradient descent, here, we just have to do a training of a deep neural network where the architecture of the deep neural network is defined by the inference algorithm that we have unrolled. So you can think about this as having defined a novel deep neural network architecture which integrates some of the knowledge that we have about the problem that is encoded in these potentials and constraints of the graphical model - but now unrolled into a chain of computations which has some parameters that yields a certain specific architecture of a deep neural network where we can just do learning / empirical risk minimization by applying stochastic gradient descent, using the back propagation algorithm for computing the gradients. So what we do is: we effectively integrate the structure of the problem (that we have encoded in the graphical model) into the architecture of a deep neural network. We derived a special new architecture through this unrolling process. This can be thought of as a form of regularization (hard constraint). We're not softly regularizing (e.g., using an L2 loss) but we have a hard constraint on the space of possible functions that can be expressed that hopefully improves generalization performance and learning from the data that we have.

Examples. Now, we are going to consider two little examples where these unrolled inference ideas have been used. One of the first instances where this has occurred is in the paper called "**Conditional Random Fields as Recurrent Neural Networks**" [63] where the goal is semantic segmentation. We have a graphical model (see Fig. 65, equation 1). This is copied from the paper and uses a different notation. It is an energy. So we have $exp(-E)$ as the Gibbs distribution. This energy is composed of unary and pairwise terms where these pairwise terms connect all possible combination of pixels in the image. They are specified through the Gaussian kernels which are defined in Fig. 65, equation 2.

$$E(\mathbf{x}) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j), \qquad (1)$$

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{M} w^{(m)} k_G^{(m)}(\mathbf{f}_i, \mathbf{f}_j), \qquad (2)$$



Algorithm 1 Mean-field in dense CRFs [29], broken down to common CNN operations.

$Q_i(l) \leftarrow \frac{1}{Z_i} \exp\left(U_i(l)\right)$ for all i ▷ Initialization

while not converged **do**

$\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all m ▷ Message Passing

$\check{Q}_i(l) \leftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ Weighting Filter Outputs

$\hat{Q}_i(l) \leftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \check{Q}_i(l')$ ▷ Compatibility Transform

$\breve{Q}_i(l) \leftarrow U_i(l) - \hat{Q}_i(l)$ ▷ Adding Unary Potentials

$Q_i \leftarrow \frac{1}{Z_i} \exp\left(\breve{Q}_i(l)\right)$ ▷ Normalizing

end while

Figure 65: **CRF as RNN model.** Illustration of the Conditional Random Field as Recurrent Neural Network model.

There is a special inference algorithm for this so-called densely connected CRF that, despite the fact that we have a lot of these potentials, gives us results in comparably little time (efficient). It is based on the so-called mean-field algorithm. So it is a different inference algorithm than the belief propagation algorithm that we have discussed before. However, the general gist of this is the same: There is a graphical model. There is some inference algorithm that gives us a solution. We unroll this inference algorithm for a couple of iterations and then, we do empirical risk minimization. Given the data set, we try to make as little error in the predictions for the inputs in the data set compared to the annotated examples in that data set.

Fig. 66 depicts some results. Three input images and the corresponding semantic segmentations are shown. In the semantic segmentations, each pixel is classified into a particular class label, such as rider, bicycle, sofa, or horse. FCN-8s and DeepLab are two deep learning architectures. We can see that they make certain errors. FCN-8s is a very simple architecture that leads to very smooth results. DeepLab is a more complicated architecture, but it produces quite a bit of noise. Using the smoothness constraints (the priors) that are encoded in the CRF by combining the CRF with deep potentials and back propagating, training this end to end, the segmentation boundaries have improved for these examples.

Our second example is **RayNet** [41] which tries to apply this idea of unrolling to the multi-view reconstruction model that we introduced in chapter 6. You will recognise the distribution over voxel occupancies $p(\mathbf{o}) = \frac{1}{Z} \prod_{i \in \mathcal{X}} \underbrace{\varphi_i(o_i)}_{\text{unary}} \prod_{r \in \mathcal{R}} \underbrace{\psi_r(\mathbf{o}_r)}_{\text{ray}}$ as well as $\varphi_i(o_i) = \gamma^{o_i}(1-\gamma)^{1-o_i}$ from before. What is different from before is that now, we have a little neural network at every pixel that produces a probability score for a particular depth value and that relates then to the occupancy of a particular voxel in that volume: $\psi_r(\mathbf{o_r}) = \sum_{i=1}^{N_r} o_i^r \prod_{j<i}(1-o_j^r) s_i^r$. So we have neural networks that can estimate roughly the probability for a particular depth that is observed for every pixel in every image that participates in the reconstruction. And we optimize the parameters of these neural networks as well as the parameters of the graphical model for inference that encodes this image formation process by applying a forward and backward pass through his unrolled computation graph. The corresponding factor graph is depicted in Fig. 67:

Fig. 68 is a high-level picture of this method. Some 2D convolutional neural networks extract features from the images. Then, we take a reference view and adjacent views. From those, using multi-view reconstruction, we can obtain surface probabilities (the probability over the depth at a particular pixel). So we get a volume because we have this for every pixel in the input images. We do this for all the reference views. Then, three unrolled iterations of this sum product belief propagation algorithm are depicted where we have ray potentials that encode the image formation process and the unary potentials φ_i that encode a prior about the occupancy of the space. We iteratively update these occupancy variables (a variable that is associated with every voxel) in order to obtain a depth map at every view. Then, we assume that ground truth depth maps are available. We minimize the expected loss of the inferred depth distribution with respect to the ground truth depth map. Fig. 69 depicts a result of this process. The corresponding input image is shown. You can see the result of just local predictions, using a CNN, and then the result of the combination of the local features with the constraints that are defined by the graphical model by doing unrolling of this graphical model and training the CNN features

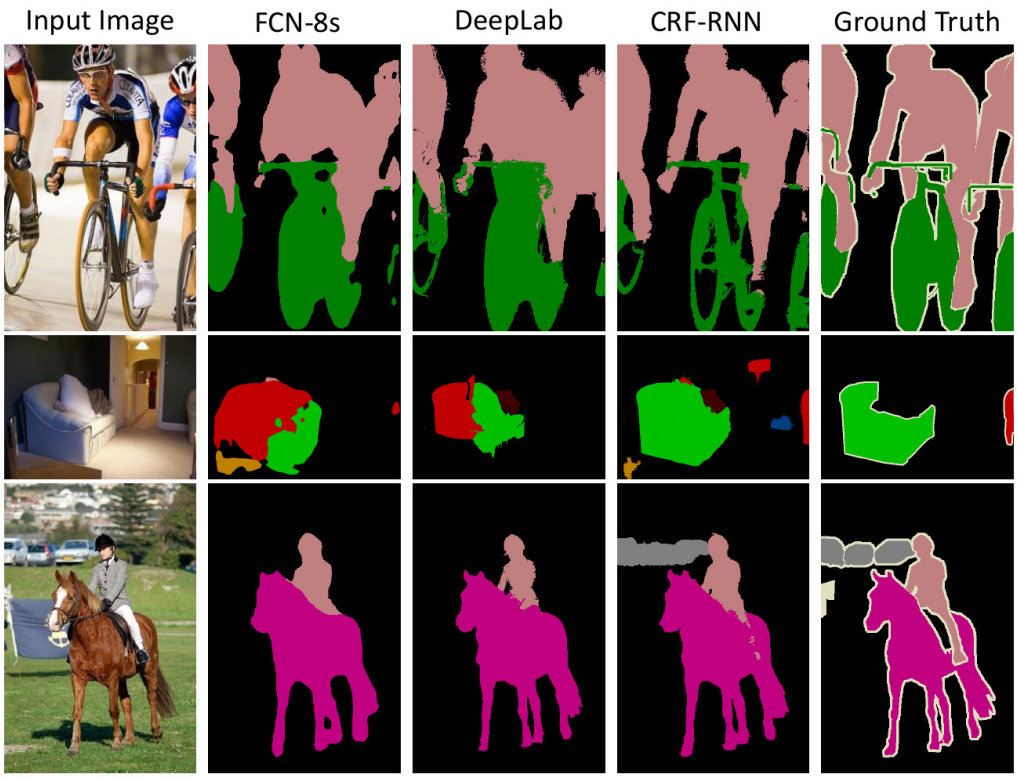


Figure 66: **Results of CRF as RNN.** Comparing the results of the Conditional Random Field as Recurrent Neural Network to other approaches.

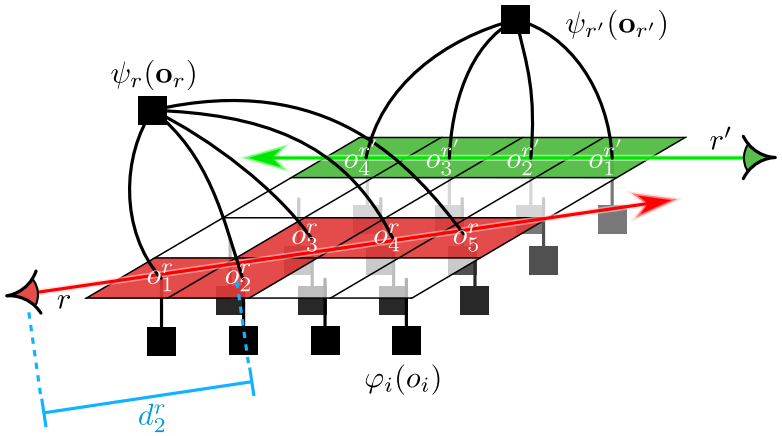


Figure 67: **RayNet factor graph.** Depicting the factor graph behind RayNet.

and the weights in this graphical model jointly (Fig. 69, c).

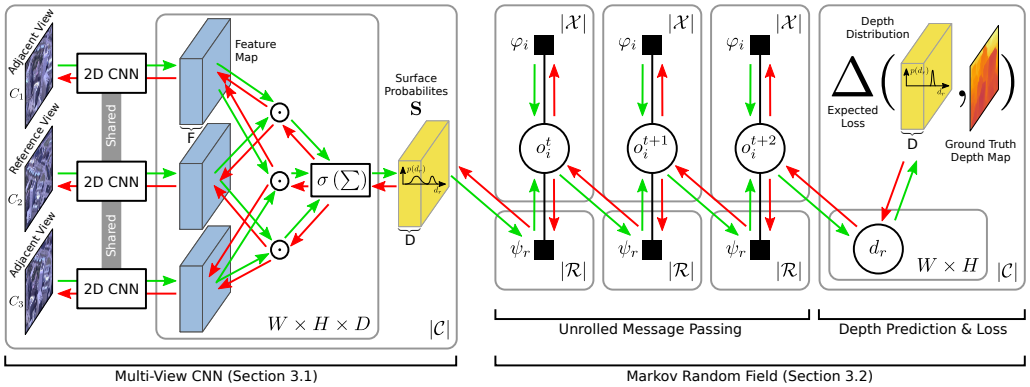
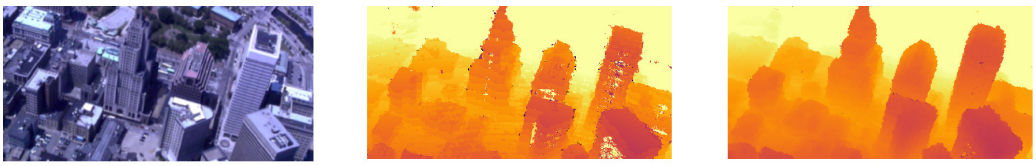
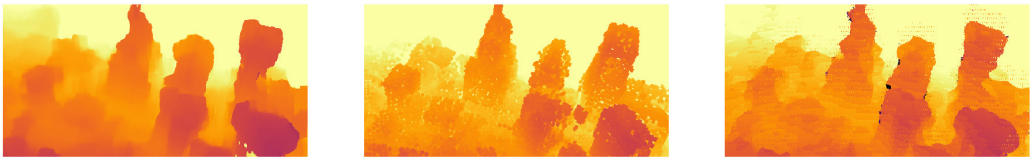


Figure 68: **RayNet model.** Depicting the architecture of RayNet.



(a) Image (b) Ours (CNN) (c) Ours (CNN+MRF)



(d) ZNCC (e) Ulusoy et al. [35] (f) Hartmann et al. [14]

Figure 69: **RayNet results.** Comparing the RayNet results to other approaches.

8 Shape-from-X

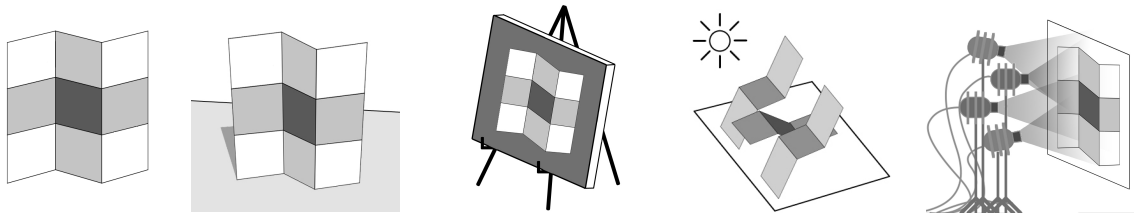
In this lecture the topic is "Shape-from-X", which means reconstructing 3D-geometry from images. Previously we already discussed two techniques: One was binocular stereo, reconstructing from two images using stereo matching techniques. The other was a multi-view reconstructing approach. Here we are going to learn how to use other cues for reconstructing, in particular shading cues.

8.1 Shape-from-Shading

Shape-from-Shading is a technique that tries to reconstruct the 3D-geometry from as little as a single image.

Shading is the relative intensity that one can observe looking at an image. In other words: We want to uncover the relation between the intensity observed(e.g. darker and lighter regions in an image) and the underlying shape. We will see that this is only possible under strong assumptions, even though intuitively it should be possible, as humans generally are able to estimate the light direction and geometry from an image.

But in general this problem can be very challenging, in particular if we restrict ourselfs to only a single image. This can be seen in the famous Adelson and Pentland's Workshop Metaphor [Fig. 70].



(a) an image (b) a likely explanation (c) painter's explanation (d) sculptor's explanation (e) gaffer's explanation

Figure 70: **Adelson and Pentland's Workshop Metaphor** Multiple different scenes can result in the same image.

Solving the inverse problem is hard because the space of shapes, paint and light that produce a particular image is vast. In the workshop metaphor (see Figure 70) on the left the underlying image is shown, while on the right multiple explanations(with different geometry and illumination) that result in exactly the image are shown. We are going to be looking for the simplest, most-likely explanation, using prior knowledge.

First let's quickly recap the rendering equation: Let $\mathbf{p} \in \mathbb{R}^3$ denote a 3D surface point, $\mathbf{v} \in \mathbb{R}^3$ the viewing direction and $\mathbf{s} \in \mathbb{R}^3$ the incoming light direction. The **rendering equation** describes how much of the light L_{in} with wavelength λ arriving at the patch $\mathbf{p}$ is reflected into the viewing direction $\mathbf{v}$:

$$L_{\text{out}}(\mathbf{p}, \mathbf{v}, \lambda) = L_{\text{emit}}(\mathbf{p}, \mathbf{v}, \lambda) + \int_{\Omega} \text{BRDF}(\mathbf{p}, \mathbf{s}, \mathbf{v}, \lambda) \cdot L_{\text{in}}(\mathbf{p}, \mathbf{s}, \lambda) \cdot (-\mathbf{n}^\top \mathbf{s}) \, d\mathbf{s}$$

Here Ω denotes the unit hemisphere at normal $\mathbf{n}$, the bidirectional reflectance distribution function BRDF defines how light is reflected at an opaque surface. $L_{\text{emit}}(\mathbf{p}, \mathbf{v}, \lambda)$ is positive only for light emitting surfaces. Please also refer to [Fig. 71].

Usually we can discard the L_{emit} term and only consider the integral over the hemisphere at the patch. The term $(-\mathbf{n}^\top \mathbf{s})$ is modelling the foreshortening effect, the intensity of the reflection depends on the angle of the incoming light. We take the integral to account for all light sources in a scene.

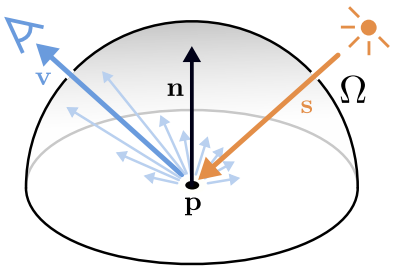


Figure 71: Definition of the 3D surface point $\mathbf{p} \in \mathbb{R}^3$, the viewing direction $\mathbf{v} \in \mathbb{R}^3$ and the incoming light direction $\mathbf{s} \in \mathbb{R}^3$ as used in the **Rendering Equation**

The BRDFs usually have a diffuse and a specular component, which model different behaviors of the light at the surface depending on the material. Most materials are both specular and diffuse.

We will now simplify the rendering equation by dropping the dependency on the wavelength λ and $\mathbf{p}$ for notational simplicity and also considering only a single point light source at direction $\mathbf{s}$. The rendering equation can now be stated much simpler in the following way:

$$L_{\text{out}}(\mathbf{v}) = \text{BRDF}(\mathbf{s}, \mathbf{v}) \cdot L_{\text{in}} \cdot (-\mathbf{n}^\top \mathbf{s})$$

In particular the integral has vanished, as we only have a singular point light source that illuminates the scene.

If we further assume a purely diffuse material everywhere in the image with an albedo(diffuse reflectance) $BRDF(\mathbf{s}, \mathbf{v}) = \rho$, we get the following:

$$L_{\text{out}} = \rho \cdot L_{\text{in}} \cdot (\mathbf{n}^\top \mathbf{s})$$

Most notably, L_{out} is now independent of $\mathbf{v}$. Additionally we also dropped the minus sign by reversing the definition of the orientation of the light ray $\mathbf{s}$. This way we arrive at a much simpler expression for the rendering equation(under certain assumptions).

$$L_{\text{out}} = \rho \cdot L_{\text{in}} \cdot (\mathbf{n}^\top \mathbf{s}) = \mathcal{R}(\mathbf{n})$$

We are going to call this simpler expression the reflectance map $\mathcal{R}(\mathbf{n})$. This is a function over all possible surface normal unit vectors $\mathbf{n}$. If we would know $\mathbf{n}$ at each surface point we could integrate the geometry. We cannot directly infer the geometry from the intensity, but rather from the normal $\mathbf{n}$, which is the gradient of the depth. Therefore our goal is to determine $\mathbf{n}$ from the observation L_{out} for every pixel in the image. This is the so called Shape-from-Shading(Berthold Horn,1970) problem.

8.1.1 Shape-from-Shading(SfS)

Let us recap the assumptions we made so far: First, we assume a diffuse material with spatially constant albedo ρ. We now have only a single parameter that describes the material properties of the entire object. Secondly, we consider a single point light source at infinity. This allows us to keep the light direction $\mathbf{s}$ constant for all pixels in the image, independent of the geometry or depth. And lastly, we assume we consider a known camera at infinity(orthographic projection). This similarly enables us to keep the view direction $\mathbf{v}$ constant across all pixels and assume it to be independent of the geometry and depth as well. These assumptions are necessary for the following simple algorithm to work, but there are also extensions to the algorithm that can work with weaker assumptions.

The first thing that we have to address is the parametrization of the normal $\mathbf{n}$. While $\mathbf{n} \in \mathbb{R}^3$ it only has 2 degrees of freedom. We are going to use the so called gradient space representation: Instead of specifying the normal directly, we parametrize $\mathbf{n}$ by the negative gradients of the depth map: $(p, q) = \left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}\right)$.

The surface normal $\mathbf{n}$ at a pixel (x, y) is now given by:

$$\mathbf{n} = \frac{(p, q, 1)^\top}{\sqrt{p^2 + q^2 + 1}}$$

With this formula we can calculate the normal given the gradients of the depth map and inversely we can infer the gradient given the normal.

Assuming this representation and assuming the following relation for the albedo: $\rho \cdot L_{in} = 1$, we can formulate the reflectance $\mathcal{R}(\mathbf{n})$ in the following way:

$$R(\mathbf{n}) = \mathbf{n}^\top \mathbf{s} = \frac{p\, s_x + q\, s_y + s_z}{\sqrt{p^2 + q^2 + 1}} = R(p, q)$$

Note that we don't normalize over the vector $\mathbf{s}$, as we already assume it to be normalized. Also note the change of variables, the now defined reflectance depends only on the gradients of the depth map.

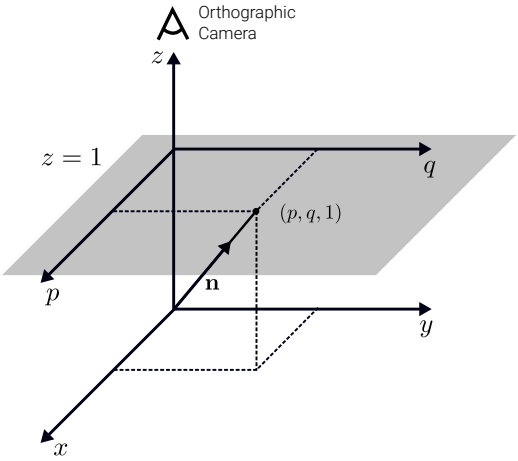


Figure 72: **Gradient space representation** Visualization of the gradient space (grey plane) and the normal $\mathbf{n}$

In figure [Fig. 72] a visualization of the gradient space (gray) is shown. If we extend the normal $\mathbf{n}$ it intersects the $z = 1$ plane exactly at $(p, q, 1)^\top$. The unit normal is therefore given by normalizing this intersecting vector:

$$\mathbf{n} = \frac{(p, q, 1)^\top}{\sqrt{p^2 + q^2 + 1}}$$

Note that the normals with a negative z component are not important for this problem, as they would imply negative reflectance. But normal vectors with a component $z = 0$ or z close to zero indeed pose a problem, as the intersection is infinitely or arbitrarily far away from the origin of the plane.

The gradient space can also be used to represent the direction of the light source $\mathbf{s}$ by finding the intersection with the $z = 1$ plane at $(p_s, q_s, 1)^\top$. To calculate the normal $\mathbf{n}$ for a given light ray $\mathbf{s}$ and observed reflectance R we can also use the gradient space representation. Because $\mathbf{n}$ and $\mathbf{s}$ are unit vectors we have $R = \mathbf{n}^\top \mathbf{s} = cos(\theta)$, were θ denotes the angle between $\mathbf{n}$ and $\mathbf{s}$. Fixing R and $\mathbf{s}$, all normals with angle θ to $\mathbf{s}$ lie on a circle around $\mathbf{s}$. Again we can take the intersection of all the candidate $\mathbf{n}$ on the circle with the $z = 1$ plane. This results in a so called Iso-Reflectance Contour, all possible solutions in gradient space therefore lie on a conic section(ellipse in figure [Fig. 72]). This non-uniqueness is obvious: We only observe one intensity, but have to estimate two unknowns (with the exception of $R(p, q) = 1$) [Fig. 73]. For $R(p, q) = 0$ the space of solutions even degenerates to a line! To infer unique normals we thus need further constraints.

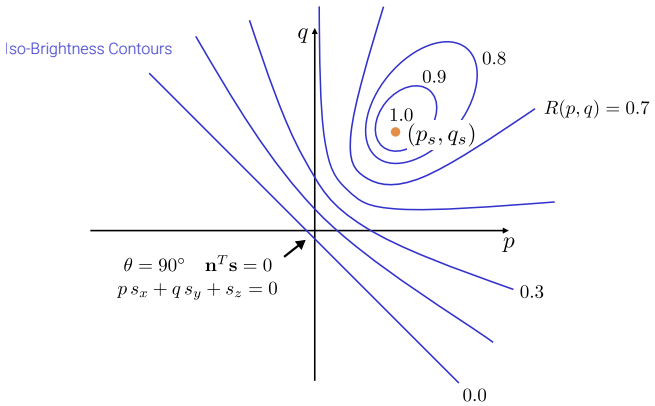


Figure 73: **Reflectance Map** The normals are not unique, they rather lie on Iso-Contours

But first let us discuss a solution to the problem that occurs if $\mathbf{n}$ becomes orthogonal or almost orthogonal to the viewing direction. In this case it is impossible to find a point in the gradient space, as the point will be infinitely far away. This can be resolved by changing the representation by a stereographic mapping[Fig. 74]. To do this we bound the area in the gradient space representation, for example by restricting the norm to 2. To now map from the normal to the restricted gradient space we consider a line from $(0, 0, -1)$ that intersects with the 'tip' of the normal vector and retrieve the coordinates that intersect the $(f, g, 1)$ plane. The following

relationship now holds:

$$f = \frac{2p}{1 + \sqrt{p^2 + q^2 + 1}}$$
$$g = \frac{2q}{1 + \sqrt{p^2 + q^2 + 1}}$$

The reflectance R now depends on f and g: $R(f(x, y), g(x, y))$.

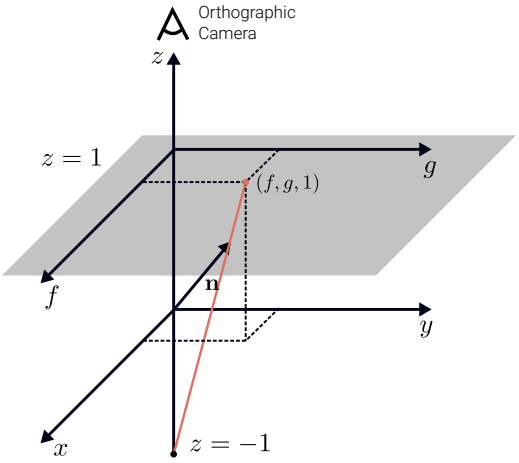


Figure 74: **Stereographic Mapping** Solution by restricting the area of the gradient space and mapping the normal to the restricted area

The first solution to finding a unique normal is using additional regularization (Shape-from-Shading). We are going to assume the following relation:

$$I(x, y) = R(f(x, y), g(x, y))$$

The image radiance(intensity) should be the same as the reflectance map. Under this assumption SfS now minimizes the following error term over the gradient field parametrized by f and g:

$$E_{image}(f, g) = \int \int (I(x, y) - R(f, g))^2 \, dx \, dy$$

Here we try to minimize the squared error between the observed intensity and the reflectance at f and g for every x and y. As we observed earlier this problem is ill-posed, so to constrain it further SfS exploits two additional constraints:

The first constraint is a simple smoothness constraint that penalizes the magnitude of the surface gradients of f and g:

$$E_{smooth}(f, g) = \int \int f_x^2 + f_y^2 + g_x^2 + g_y^2 \, dx \, dy$$

The second constraint is the so called Occluding Boundaries constraint. It's goal is to constrain normals to occluding boundaries that are known (e.g. via a different algorithm), because the occluding boundary must be orthogonal to the viewing direction by definition. At the boundary the normal is given: $\mathbf{n} = \mathbf{e} \times \mathbf{v}$ with $\mathbf{e}$ the direction of the edge.

In summary, we optimize the variational energy which comprises the image and the weighted smoothness term wrt. the gradient fields $f(x, y)$ and $g(x, y)$ subject to occluding boundary constraints:

$$E(f, g) = E_{image}(f, g) + \lambda E_{smooth}(f, g)$$

Because images are spatially discrete, this problem is transferred to a spatially discrete problem similar to the Horn-Schunk algorithm discussed in lecture 6. For this problem iterative optimization is required again, as R depends non-linearly on both f and g. On the occluding boundary we can use the known normals to fix the values for f and g, all other variables are initalized to 0. After that the iterative optimization is ran until convergence. See also https://fpcv.cs.columbia.edu/. We now know how to infer the surface gradients (p, q).

How do we get the underlying 3D-surface and depth map? Assuming a sufficently smooth surface we can solve this variational problem using the discrete Fast Fourier Transform(Frankot and Chellappa,1988):

$$E(z) = \int\int \left(\frac{\partial z}{\partial x} + p\right)^2 + \left(\frac{\partial z}{\partial y} + q\right)^2 dx\, dy$$

Here we integrate over the image domain. We sum p and q to obtain the difference, as they store the negative surface gradients.

Let's look at some results[15]:

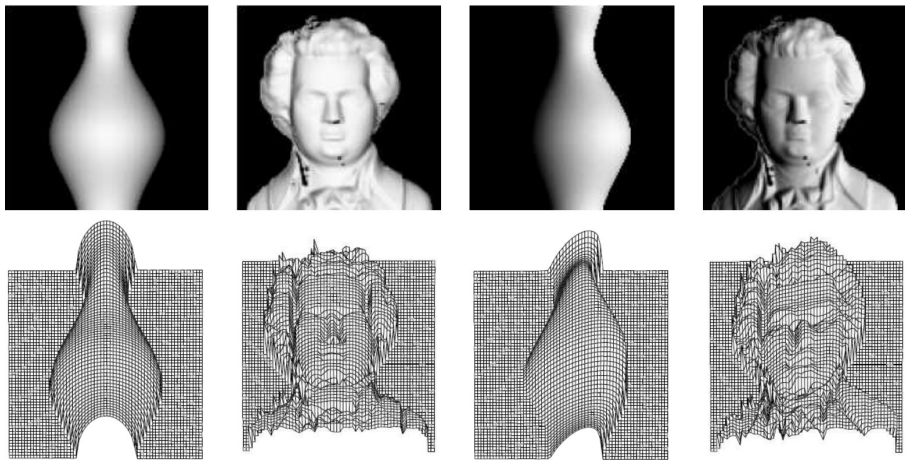


Figure 75: **SfS results** In the top row the underlying images are shown, in the bottom row the infered meshes.

There are also more modern solutions to a more difficult problem. For example SIRFS(Shape, Illumination, adn Reflectance from Shading) estimates depth, normals, reflectance, shading and lighting from a single masked RGB image. This method doesn't assume a known point light or constant color.

8.2 Photometric Stereo

In the previous section we saw how the ambiguities arising in the SfS problem can be addressed to some degree using strong smoothness assumptions. We will show that these ambiguities can also be addressed differently: Photometric Stereo generalises Shape-from-Shading to multiple observations. Here the camera is kept fixed, but the illumination of the scene is adjusted to reconstruct the 3D-geometry. Instead of the smoothness constraints we obtain more observations per pixel. To achive this we take K(e.g. 3) images of a static object from the same viewpoint, but with a different known point light source each. This way we can recover the surface normal and the albedo uniquely, even for just a single pixel. Assuming a lambertian surface the number of parameters we need to determine is three (2 for surface normal, 1 for albedo), so we are able to uniquely determine them with just three observations per pixel. For non lambertian materials we might need more observations. In principle we don't need smoothness constraints for this to work, but we still assume a far away light source and camera.

8.2.1 Reflectance Maps

For one single image we obtain certain iso-contours for a particular light source direction (p_s^1, q_s^1)[Fig. 76] at a certain pixel. Doing the same thing for a second image from the same viewpoint with a different light source direction (p_s^2, q_s^2) we obtain different iso-contours. By intersecting the two iso-contours corresponding to the observed intensities the normal can only obtain two values. By taking a third measurement we can uniquely determine the location of the normal.

This intuitive understanding can also be derived more formally: Again we are going to use Lambertian reflectance and $L_{\text{in}} = 1$, the image intensity / reflected light is: $I = L_{out} = \rho \cdot \mathbf{n}^\top \mathbf{s} = \rho \cdot \mathbf{s}^\top \mathbf{n}$. Our goal is to estimate both ρ and $\mathbf{n}$. We will avoid the gradient space representation to use the fact that the unnormalized normal encodes the albedo ρ as the magnitude of the normal. If we observe three images(with same view direction $\mathbf{v}$ and different light sources $\mathbf{s}$) we can express this relationship in the following matrix form:

$$\underbrace{\begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix}}_{\mathbf{I}} = \underbrace{\begin{pmatrix} \mathbf{s}_1^\top \\ \mathbf{s}_2^\top \\ \mathbf{s}_3^\top \end{pmatrix}}_{\mathbf{S}} \underbrace{\rho\, \mathbf{n}}_{\tilde{\mathbf{n}}}$$

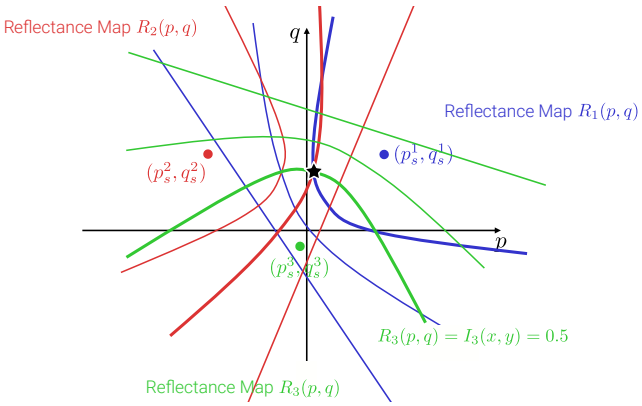


Figure 76: **Reflectance Maps for multiple images** By combining the reflectance maps of multiple images a unique solution can be determined

This matrix form can simply be solved with $\tilde{\mathbf{n}} = \mathbf{S}^{-1}\mathbf{I}$, $\rho = \|\tilde{\mathbf{n}}\|_2$ and $\mathbf{n} = \tilde{\mathbf{n}}/\rho$.

This seems simple enough, but when does photometric stereo not work? As we need to invert $\mathbf{S}$ to solve the matrix form we need it to have full rank. If we have a linear dependency between the light sources(e.g. all light sources and the origin $\mathbf{p}$ lie on the same 3D plane) we cannot invert $\mathbf{S}$ and cannot find a unique solution $\tilde{\mathbf{n}}$. There are also other challenges, the observed intensities might be noisy, the material might not be truly lambertian or the camera might not be truly orthographic. To tackle these challenges we can add more constraints or add more observations. With more observations we can solve the matrix form using a least squares solution.



Figure 77: **Photometric Stereo Algorithm** From left to right: base image, surface normals, albedo, depth map, relit image with uniform albedo[59]

The full algorithm [Fig. 77] now consists of the following three steps: First we compute the surface normals and albedo values for each pixel in the image. Then we integrate the surface normals to obtain the depth map. Lastly we relight the object(e.g. with uniform albedo).

For color images we apply Photometric Stereo to each color channel individually to obtain the color albedo. Also note that deviations from the lambertian assumption could cause artifacts.

One can also apply the Photometric Stereo algorithm to outdoor images, as the sun is a perfect point light source for our assumptions. This can be utilized by observing a scene at different days and times of day[1].

One problem with Photometric Stereo is that we assume the light sources to be known, which is often not the case. This is a topic still under investigation but some approaches have been explored, e.g. using deep neural networks to infer both the surface normals and the light sources.

There is lots of other ongoing research in Photometric Stereo(PS), topics include Volumetric Multi-View PS, Perspective PS, Uncalibrated PS, Deep Models, mobile setups and problems involving non-lambertian materials.

8.3 Shape-from-X

This unit provides a short overview of different Shape-from-X techniques.

In previous lectures we discussed how to do binocular stereo by using the epipolar geometry to find point correspondences in two images along a scanline. For this we could use smoothness constraints to overcome ambiguities. If we use more than two images this process is called multi-view stereo (e.g. using CNNs).

In this lecture we have seen how to obtain geometry from even just a single image using shading cues with certain smoothness assumptions. We have also seen how to circumvent these assumptions by just using more images from the same viewpoint to determine a unique solution.

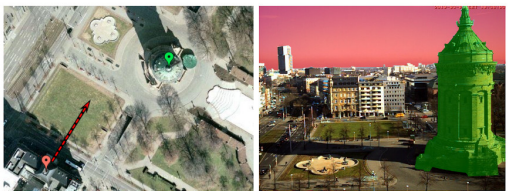


Figure 1. The geographic context of the scene with camera (red marker) and target object (green marker) on the left, and an example image from the camera with sky mask (red region) and object mask (green region) on the right. Sat image © by Google Inc.



Figure 6. One input image, detected shadow regions, selected points for intensity estimation and the recovered object albedo.

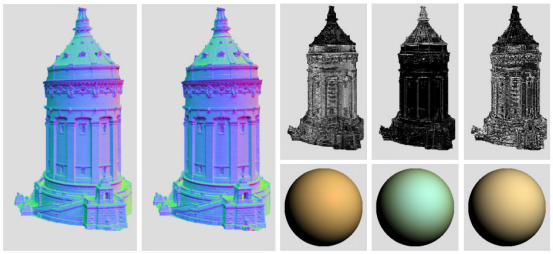


Figure 7. The initial normal map, the final normal map, and the four recovered BRDFs with corresponding material map.

One could also infer the geometry by considering the observed texture. This is called Shape-from-Texture and assumes that the scene has some regularity in local textural elements. But in general this is not applicable.

Another more broadly used technique is so called Structured Light estimation. Here a pattern projector is used to illuminate the scene using a stripe or dot pattern. Because the pattern and the position of the projector is known the geometry of the scene can be infered using a single camera by triangulation. One advantage of this setup is that it can overcome e.g. textureless regions that usually are very hard to uniquely estimate with the other algorithms. There are multiple devices that have been developed for this application, but the first product that really revolutionized research in this area is Microsoft's Kinect V1, which was developed for the gaming industry(wasn't a big success there) but was a huge success in robotics and computer vision research. But there are multiple follow-up devices, also from different companies. One problem of the Structured Light approach is that the light emitted by the projector usually is too weak compared to the sunlight and therefore has limited applications outdoors.

Another technique for estimating geometry is called Monocular Depth Estimation, that in contrast to Shape-from-Shading doesn't make any assumptions about the materials in the scene, despite also using just a singular image. This method tries to directly estimate the geometry by using a large deep neural network that produces the depth map from an input image. With this approach state-of-the-art results[29] can be achieved[Fig. 78]. This is a very versatile algorithm that works on many different scenes.



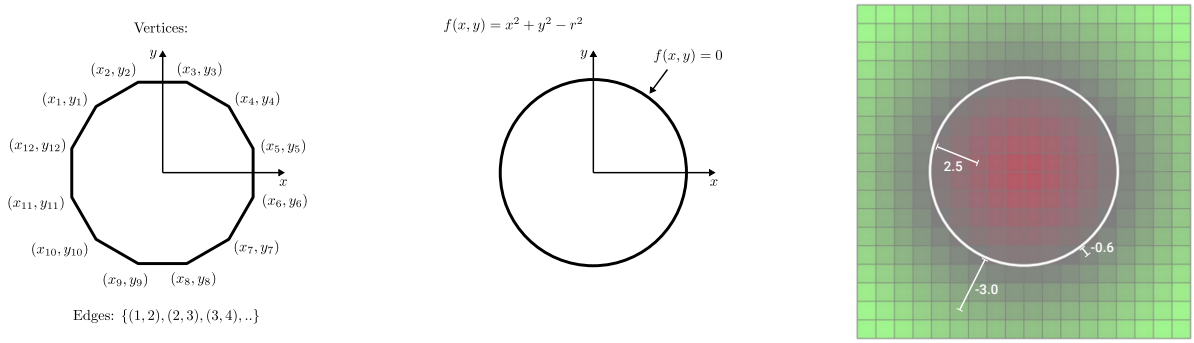
Figure 78: Monocular Depth Estimation, infering the underlying geometry from a singular image using deep neural networks

Additionaly to the visible 3D-geometry one can also try estimate the complete shape of the objects in a scene, even from just a single image.

8.4 Volumetric Fusion

So far we have seen multiple approaches that mostly just considered a single object and only produced a single depth map. Often we might want to reconstruct larger scenes that are composed of multiple objects. In this last section we will discuss how to fuse/join multiple (incomplete) reconstructions into a more global, larger scale model of the world. This technique is called Volumetric Fusion.

The traditional 3D reconstruction pipeline is composed the following way: We get a set of images as input and first estimate the camera poses for each one. From these poses we can compute dense correspondences using binocular or multiview stereo techniques but also the techniques we discussed in this unit. This results in a depth map for each image in the set of input images. The last step is so called depth map fusion. Here we try to combine all of the depth maps into one consistent 3D reconstruction.



(a) **Mesh** (b) **Implicit (Continuous)** (c) **Implicit (Discrete SDF)**

To understand how depth map fusion works we first need to understand the representation used: In [Fig. 79a] we can see a mesh based representation. It is a circle discretised into multiple vertices and edges. This is an explicit representation of the geometry, storing all information in vertices and faces. But this representation is not ideal for fusion, in particular it is very hard to change topology in this representation.

Instead we will consider implicit representations. The simplest implicit representation is a simple parametric form [Fig. 79b]. We can represent a circle as the level set $f(x, y) = 0$ of the parabola function $f(x, y) = x^2 + y^2 - r^2$. This way we implicitly defined the shape as the level set of a function. In the next lecture we will explore this concept more, in particular how to represent these functions using neural networks.

In this lecture we will focus on the discretization [Fig. 79c] of the implicit space. We are mainly going to look at so called signed distance functions that measure distances from the center of each of the voxels to the surface. To do this we discretize the space to a chosen resolution and insert a value at each voxel to the distance to the closest surface. Using the signed distance we indicate that a voxel is inside the object if the distance is positive, outside if the distance is negative. The surface is represented as the zero level set again. One crucial advantage of implicit models is that they allow for representing arbitrary topologies and topology changes (e.g. adding and removing objects, or modifying existing objects in the scene).

We discussed the underlying representation, let us discuss the high-level steps of Volumetric Fusion: First we convert the depth map to the implicit discrete SDF representation we just outlined. Then we perform Volumetric Fusion. The last step is to extract the mesh from the fused implicit model.

8.4.1 Depth-to-SDF Conversion

Unfortunately the problem is not as easy as outlined before, as the true distance to the surface is often unknown. Usually we only know the depth of a surface along a particular ray. Therefore SDF fusion methods often assume that the distance to the surface can be approximated with the distance along the ray[Fig. 80]. Note that this is not necessarily the closest distance to the surface for each voxel, but rather an upper bound to the distance. We often consider only distance values close to the surface, as we assume that the reconstruction is pretty good already and only fuse in the vincinity of the surface. We assign the ray distance values to the corresponding voxels as the signed distance for a single ray. For every viewpoint there are many different rays that we use to estimate the distance to the surface. For voxels that are not captured yet we can interpolate the distance value. In an implementation we would implement it reversely: We would project all voxels into the image and find the corresponding depth map this way. In the end we have a discretised SDF representation for each different camera.

8.4.2 Volumetric Fusion

Let us now illustrate how to do the actual Volumetric Fusion. We are going to consider the more simple orthographic case, where the cameras are orthographic and all rays are parallel to each other. But this algorithm

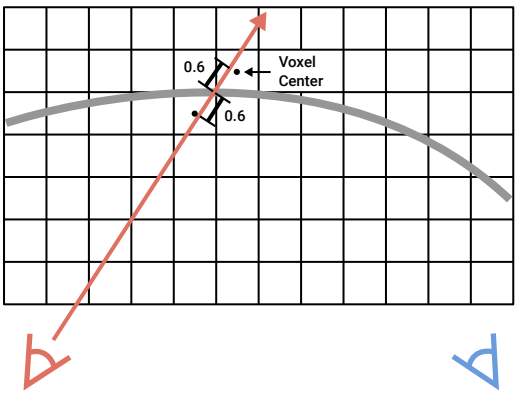
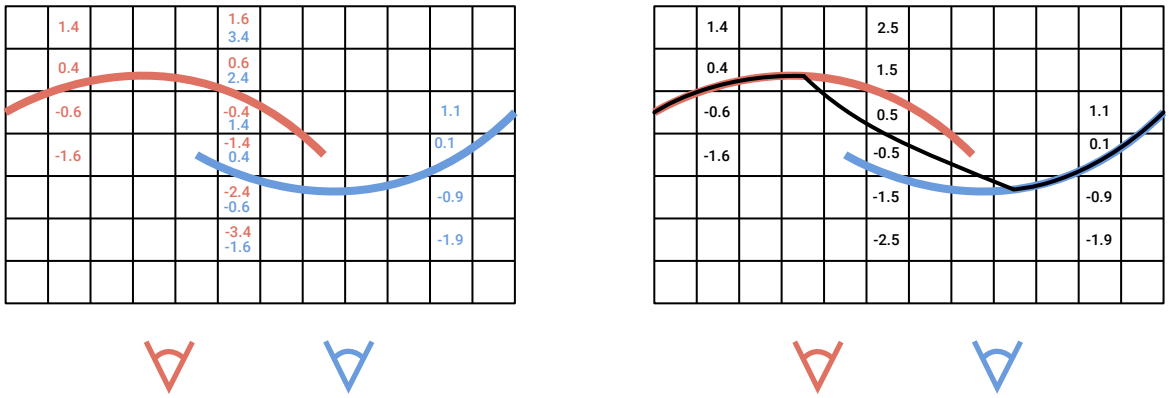


Figure 80: **Depth-to-SDF Conversion** Estimating the distance to the surface by using the distance along a ray



(a) Before averaging. There is an overlap between the two SDF fields.

(b) After averaging the values. Implicit surface shown as black line.

Figure 81: **Volumetric Fusion**

also works for perspective cameras. After the conversion to the discretised SDF representation we now calculate the average of the different SDF fields[Fig. 81a], this gives us and average of the implicit surface. By calculating the average we therefore also average over the implicit surface that we are trying to extract. In [Fig. 81a] we can see that in the middle column we have overlapping information for the two surfaces for both the red and the blue camera. By averaging these values over the whole domain [Fig. 81b] Volumetric Fusion also averages/interpolates the implicit surface (black line). The resulting implicit surface is smooth. This Fusion can be seen as a weighted average(with weights w_i) per voxel:

$$D(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x})\, d_i(\mathbf{x})}{\sum_i w_i(\mathbf{x})}$$
$$W(\mathbf{x}) = \sum_i w_i(\mathbf{x})$$

The fused distance is given by $D(\mathbf{x})$, where i denotes the indexed camera, $W(\mathbf{x})$ denotes the fused weight. This can be conveniently expressed with an incremental update rule:

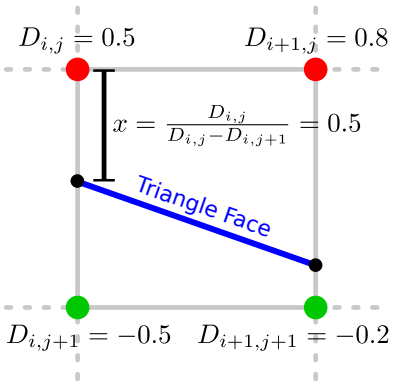
$$D_{i+1}(\mathbf{x}) = \frac{W_i(\mathbf{x})\, D_i(\mathbf{x}) + w_{i+1}(\mathbf{x})\, d_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})}$$
$$W_{i+1}(\mathbf{x}) = W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})$$

This way we can add more images with a constant computational overhead. We need these weights to downweight SDF values behind the surface, as these basically are just guesses about the distance. But we need to assign some value to the close voxels behind the surface to obtain a level set in the fused surface. We can also use weights to downweight rays we are more uncertain about, e.g. an observation taken from a large distance with large measurement noise.

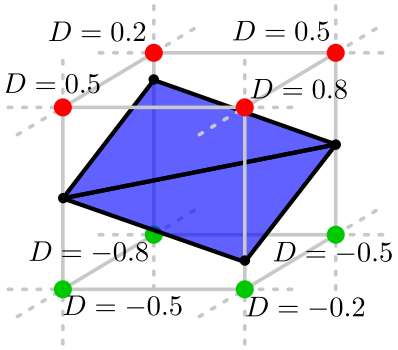
But why is weighted averaging in general a good idea here? It can be seen as the solution to the following weighted least squares problem: $D^* = \text{argmin}_D \sum_i w_i\,(d_i - D)^2$. Therefore D can be thought of as the optimal distance in the sense that it minimizes the least squares distance for every $\mathbf{x}$.

8.4.3 Mesh Extractions

Let's now consider how to extract a mesh from the resulting averaged SDF using the marching cubes algorithm: To do this we try to find a triangular mesh that best approximates the zero-level set. This is achived by "marching" through all the grid cells(can be done independently for all grid cells) and inserting triangle faces whenever a sign change is detected. In this process we first estimate the topology and then we predict the vertex location. For example in the 2D case[Fig. 82a] the topology describes the way the triangle face is inserted into the pixel. If all corners (voxel centers in the dual grid) shared a color no triangle face would be inserted, if only one corner was a different color the triangle face would be very skewed. In 2D there are 2^4 possible topologies. In the 3D case [Fig. 82b] there are 2^8 possible topologies (that can be grouped into 15 equivalence classes due to rotational symmetry).



(a) 2D case. Topology shown in blue, location of the left vertex given by offset x.



(b) 3D case. Topology in blue.

Figure 82: Mesh Extractions

The next step is to predict the location of the vertices. Because we have signed distance values a resonable thing to assume is that the location of the vertices is at the zero crossing of a linear interpolation $f(x) = D + x(D' - D)$ between the values. This zero crossing implies the following solution for the vertices: $f(x) \stackrel{!}{=} 0 \Rightarrow x = D/(D - D')$.

8.4.4 Applications

KinectFusion is a real time fusion and pose estimation method that uses the Kinect sensor we discussed earlier. It employs a real time pose estimation using a version of the iterated closest point algorithm and then updates the reconstruction using the SDF fusion techniques we just discussed.

One extension to this work is given by so called DynamicFusion that addresses the problem of a non-static scene by mapping a captured cannonical frame to the live frame at each time step. This approach is able to reconstruct non-rigid scenes by warping the estimated geometry.

Another work is OctNetFusion, that integrates a 3D convolutional network into the fusion process using efficient adaptive data structures. This enables us to output a denoised reconstruction from features from the fusion process.

Another line of work is approaches that try to solve the learning problem end-to-end. One example is Deep Marching Cubes, which employs a differentiable variant of the Marching Cubes algorithm to be able to backpropagate gradients end-to-end.

9 Coordinate-based Networks

9.1 Implicit Neural Representations

A traditional 3D reconstruction pipeline, where the input is a set of images, infers the camera poses for instance using bundle adjustment. Then it is possible to establish dense correspondences, stereo or multiview stereo, resulting in depth maps. And finally, the depth maps can be fused into a more coherent global 3D reconstruction of the scene.

Although some parts of this pipeline might utilize learning, it is not end-to-end learning.
It might be beneficial to directly go from just one or multiple images to a complete 3d reconstruction of an object.
The next part covers this question:

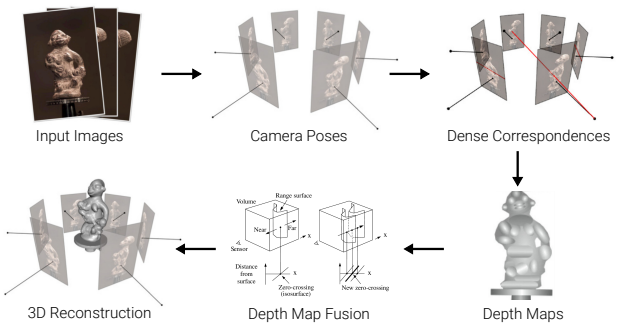


Figure 83: Traditional 3D Reconstruction Pipeline

Can we **learn** 3D reconstruction **from data**?



[Newcombe et al., 2011]



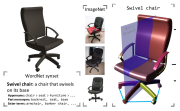
[Choi et al., 2011]



[Dai et al., 2017]



[Wu et al., 2015]



[Chang et al., 2015]



[Chang et al., 2017]

Figure 84: 3D Datasets and Repositories

Learning requires data, but luckily over the last 10 years or so a lot of 3D data sets have become publicly available. For example through scanning techniques such as active light scanning systems [Newcombe et al., 2011] that were discussed in a previous lecture. [Choi et al., 2011] is another data set that has been scanned for both the object level as well as at the room level. [Wu et al., 2015] is a dataset comprised of CAD-models that have been designed by artists. Also there are a lot of these CAD-models now available online for free. A more recent dataset is [Chang et al., 2017] which was created with motivation in mind to not just reconstruct single rooms but entire buildings.

To build a model that learns to predict shapes directly from one or multiple input images, it is necessary to answer the question:

What is a good **output** representation?

The input representation is pretty clear. It could be, in the simplest case, a single 2D image that gets processed with a standard 2D convolutional network as introduced in the deep-learning lecture. But what should be the decoder and in particular what is a good output representation?

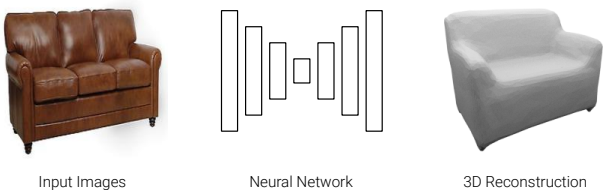


Figure 85: 3D Reconstruction from 2D image

9.1.1 3D output represetations

- **Voxels**
  The most simple extension of a pixel-based representation is a voxel-based representation. A voxel is a 3D equivalent of a pixel.

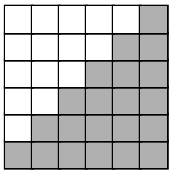
  – **Discretization** of 3D space into grid

  – Easy to process with neural networks

  – Cubic memory $O(n^3)$ ⇒ limited resolution

  – Manhattan world bias

  [Maturana et al., IROS 2015]





- **Points**
  Another representation is a point-set. Points are a discretization of the surface into 3D points. It is not a volumetric representation but it directly a represents the surface.

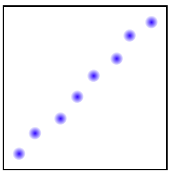
  – **Discretization** of surface into 3D points

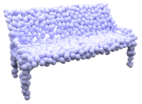
  – Does not model connectivity / topology

  – Limited number of points

  – Global shape description

  [Fan et al., CVPR 2017]





- **Meshes**
  Meshes are yet another output representation. They discretize the 3D space into vertices and faces to represent the object's surface.

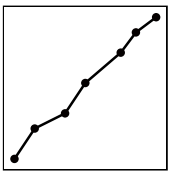
  – **Discretization** into vertices and faces

  – Limited number of vertices / granularity

  – Requires class-specific template – or –

  – Leads to self-intersections

  [Groueix et al., CVPR 2018]





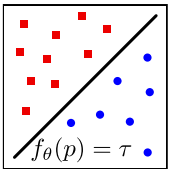
- **Implicit Representation**
  With implicit representations, the goal is to go away from these discretized representations and instead represent the surface with an implicit function.

  – Implicit representation ⇒ **No discretization**

  – Arbitrary topology & resolution

  – Low memory footprint

  – Not restricted to specific class





9.1.2 Occupancy Networks

The key idea is to not represent the 3D shape explicitly, but instead to consider the surface implicitly as the decision boundary of a non-linear classifier. A linear neural network is used to separate two classes, the red points in Fig 87 are the points inside the object which take occupancy value one, and the blue points are outside the object which take occupancy value zero.

To train a classification model from which a decision boundary can be extracted, which most optimally sep-

$$f_\theta : \mathbb{R}^3 \times \mathcal{X} \rightarrow [0, 1]$$

3D Location — Condition (eg, Image) — Occupancy Probability

Figure 86: Occupancy-Network formula

arate the inside from the outside points, is what is considered as representing the surface implicitly by the parameters of the model. It is an implicit representation because it needs to be extracted in a post-processing step because the only thing that is accessible is the classifier that separates the inside from the outside. This classifier is called the Occupancy Network f_θ because it's a neural network with parameters θ, which maps from a 3D location and a condition which could be an encoding of an input image or a point cloud representing the structure of the 3D shape to an occupancy. This condition is needed in order to represent multiple shapes. The output of the neural network is then an occupancy probability that is between 0 and 1. The probability can of course be modelled by a simple Sigmoid.

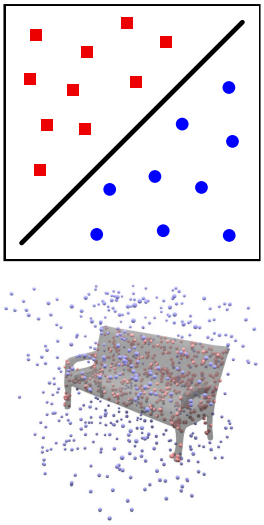


Figure 87: Decision-Boundary

In Fig. 87 the 3D case at the bottom shows an example of a non-linear decision boundary for a bench. The red points are sampled inside the bench and the blue points are sampled outside the bench. The goal is to train a classifier that most optimally separates the blue from the red points. This can be done by repeatedly sampling points from the volume, for which it is known whether they are inside or outside of the volume. Points closer to the surface can be sampled to become more precise. This of course requires full supervision, as it is necessary to know for every 3D point whether it is inside or outside. So a watertight mesh is required in order to generate ground truth data.

As a remark, the function f_θ models an occupancy field as it operates in continuous space. It is also possible to model the signed distance, as seen in previous lectures. The signed distance is useful as it allows for example to directly find the surface from a point close to the surface by just following the gradient for as long as the signed distance value is greater than zero. This can be done by a very similar model, except that the occupancy probability is exchanged with a SDF output.

The Occupancy-Network is designed as follows. A very simple multi-layer perceptron is defined, which is a residual network composed of five blocks (Fig. 88 the yellow block is repeated 5 times). An encoder network is used that takes an image or a point set which produced a fixed-length vector that is the condition. The condition is then injected at multiple locations into the residual network using so-called conditional batch normalization.

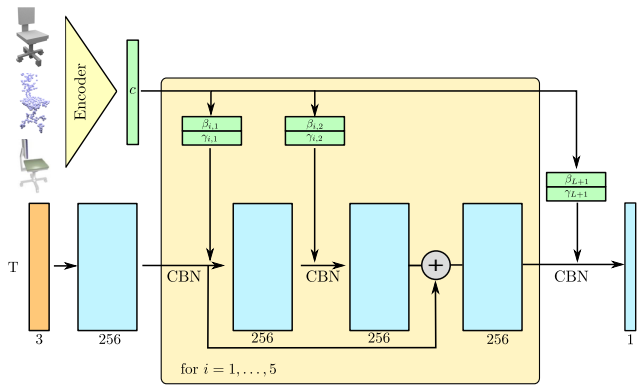


Figure 88: Occupancy-Network Architecture

The other input to the residual network is a batch of a set of 3D points. The output for these T points is the occupancy probability. As this is a very simple classification task, the model can be trained with the standard binary cross-entropy of the prediction of the model f_θ given a point p_{ij} , the input condition z_i and the ground truth o_{ij}.

$$\mathcal{L}(\theta, \psi) = \sum_{j=1}^{K} \text{BCE}(f_\theta(p_{ij}, z_i), o_{ij})$$

- K: Randomly sampled 3D points ($K = 2048$)
- BCE: Cross-entropy loss

It is also possible to build a variational occupancy-encoder which is just the variational auto-encoder applied to the idea of implicit shape representations. In addition to the reconstruction term, there is another KL term between some prior and the encoder distribution q_ψ. For this, another encoder is added to the model that goes from the point set to the latent code.

$$\mathcal{L}(\theta, \psi) = \sum_{j=1}^{K} \text{BCE}(f_\theta(p_{ij}, z_i), o_{ij}) + KL\left[q_\psi(z|(p_{ij}, o_{ij})_{j=1:K}) \,\|\, p_0(z)\right]$$

- q_ψ: Encoder

9.1.3 Extraction a mesh

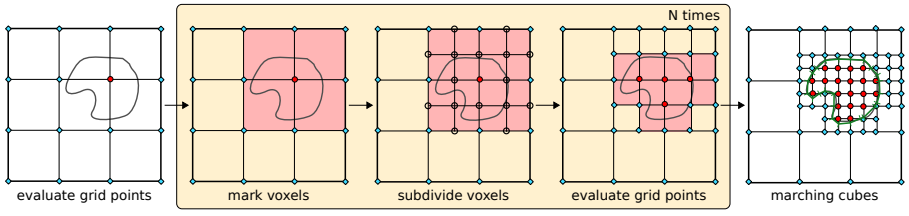


Figure 89: Multiresolution IsoSurace Extraction

Because the Occupancy Network is an implicit model, we don't directly get a mesh. So it is necessary to extract the mesh from the implicit representation. This is done by a technique called Multi-Resolution IsoSurface Extraction (MISE). MISE incrementally builds an octree by querying the occupancy network. It starts with a $n \times n \times n$ grid. In Fig. 89 an easy example with a 4×4 2D grid is shown which is a simplified case of the 3D case. It can be observed that one point is inside the object given the parametrization of the neural network. Then the adjacent cells of the inside point will be looked at and subdivided. The subdivided cells are then again queried whether they lie inside or outside. This process can be repeated a number of times to get a finer and finer representation until a level of granularity is reached that is desired. In the end, the Marching-cubes-algorithm, as discussed in the last lecture, can be used to extract a triangular mesh from this indicator grid. The whole process requires about one to three seconds depending on the size of the scene, so it's not super fast, but it's not super slow either. The main time requirement of this process comes from querying the neural network many times. But it is done in a clever way by sampling in this hierarchical course to fine manner.
Experiments with this approach outperform previous works, as seen in Fig. 90.

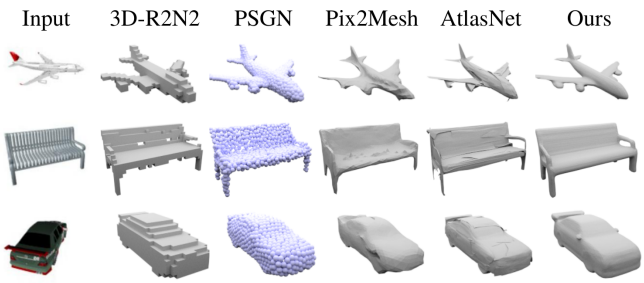


Figure 90: Occupancy results

9.1.4 Materials and Lighting

With implicit models not only occupancies can be modelled but it is pretty straightforward to extend this to appearance. What is done in Fig. 91 is called implicit surface light fields. The goal is to model view dependent

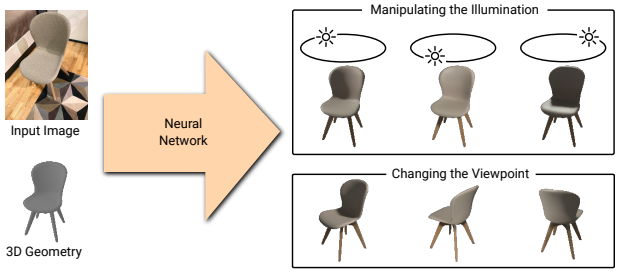


Figure 91: Appearance Teaser

appearances by conditioning a neural network on an input image just as with the occupancy networks but also on some 3D geometry that could be a CAD model that represents this object in the image. So afterwards not only the viewpoint can be changed, but also the illumination can be manipulated.
For this, instead of modelling the standard rendering equation, the conditional surface light field is modelled by a neural network.

$$L_{\text{cSLF}}(\mathbf{p}, \mathbf{v}, \mathbf{l}) : \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^M \rightarrow \mathbb{R}^3$$

In Fig. 92 is an overview of the model. The image encoder is the same as in the occupancy network. Additionally, the input shape is passed into the network and encoded by a geometry encoder into a global shape code s. Then any point on the surface can be queried. First the appearance field will be quarried. This gives a feature vector for the appearance. Then the lighting model that conditions also on the light setting and the view direction produces the color of the pixel.

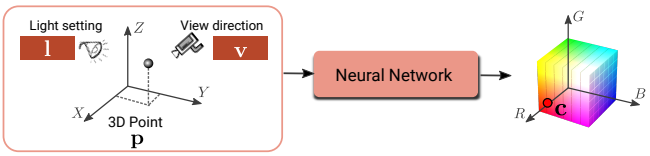


Figure 92: Single-Image Appearance Prediction

9.1.5 Representing Motion

Representing 3D motion is to simply extend this idea of occupancy networks to 4D (3D+Time). But unfortunately this is hard due to the curse of dimensionality.

Therefore, only the shape at a particular time instance t is represented. The motion is then represented by a temporally and spatially continuous vector field. Even though this is still a 4D function, in contrast to the shape evolving over time this is a much more continuous object because this vector field can be continuous over the entire duration of the sequence. This is much easier for the model to learn. The relationship between the 3D trajectory of the 3D location s and the velocity v, is given by a simple ordinary differential equation $\frac{\partial \mathbf{s}(t)}{\partial t} = \mathbf{v}(\mathbf{s}(t), t)$. And because ODE's are differentiable, it can be used inside the neural network and back-propagate gradients through it. The model can be seen in Fig. 94. Given ground truth shapes at discrete time instances for which we know inside and outside we can take any point and warp it through the ODE, that is

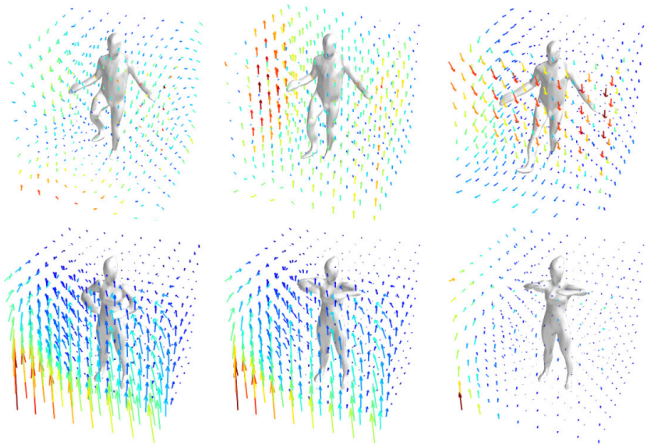


Figure 93: Occupancy Flow

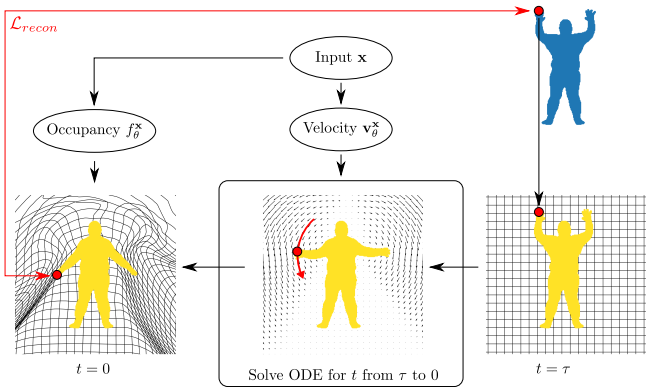


Figure 94

conditioned on the velocity field $\mathbf{v}^x + \theta$, to the location at time step zero and then the Occupancy Network can predict whether the point lies inside or outside. The advantage of the model is that it is end-to-end learnable as the ODE is differentiable, so the gradients can be backpropagated through the model to the parameters of the Occupancy Network and the parameters of the velocity network. The advantage of this approach is that no correspondences are required, as they are established by the model.

9.1.6 Representing Scenes

There are two major limitations of the implicit neural representations. The first limitation has to do with the prediction of the global latent code. But it is very difficult to represent the entire distribution of all the possible scenes in such a low dimensional vector, compared to a single object where the shape variability is not as large. And because there is no local information in the global latent code, this leads to overly smooth geometry. Another limitation is the fully connected architecture after the latent code encoding. Compared to convolutions, fully connected layers can't exploit translation equivalence.
Both drawback result in very poor results on complex scenes. So to mitigate these limitations Peng et al. have introduced a variation which uses a 2D or 3D encoder which locally encodes a 2D or 3D latent code. Then a 2D or 3D convolutional network can be used, which can exploit translation equivalence. Finally, the occupancy value can be computed with a shallow occupancy network which uses a bilinear or trilinear interpolation of the features.

9.2 Differentiable Volumetric Rendering (DVR)

With implicit representations it is assumed that full 3D supervision is available for training the model. However, in practice often this is not the case. In the lecture of 3D reconstruction we discussed how to do 3D reconstruction only from 2D images. Now the question is, can this also be done with implicit models? This is the idea of differentiable volumetric rendering, which is a technique which allows to backpropagate gradients from image based reconstruction losses to the parameters of implicit neural networks.
The architecture of such a model is seen in Fig. 96. As input the network passes a 2D image through an image encoder to the conditioned latent code. Additionally, it takes a set of 3D points which are passed together

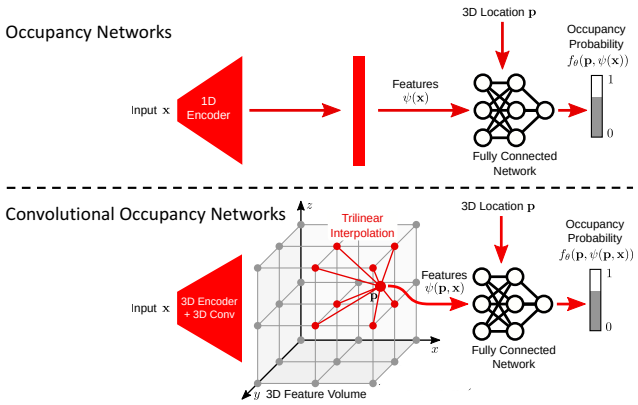


Figure 95: Comparison of Occupancy-Net Variation

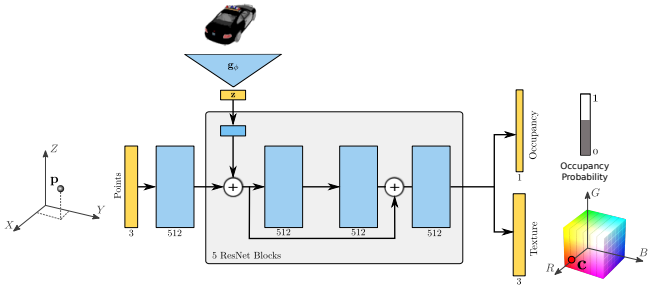


Figure 96: DVR: Differentiable Volumetric Rendering

with the conditioned latent code through the residual blocks. But in contrast to the occupancy network, this network now has two shallow heads which predict occupancy and texture or color respectively.

To train the parameters of the model, it is necessary to define rendering operations to render the representation into images.

9.2.1 Foward Pass

The method is shown in Fig. 97. Form all pixels u, a ray is cast through the 3D volume and intersect the object's surface at some point. Ray marching is used to find an estimate of the surface point $\hat{\mathbf{p}}$ for all pixels u along the ray w, by checking where the occupancy value changes from below threshold to above the occupancy threshold. Then in this interval the exact location of the surface can be found by numerical root finding with the secant method. Once the optimal point $\hat{\mathbf{p}}$ is found on the surface, the texture field can be evaluated at that position to get the color which is mapped to the pixel u. **Secant Method**

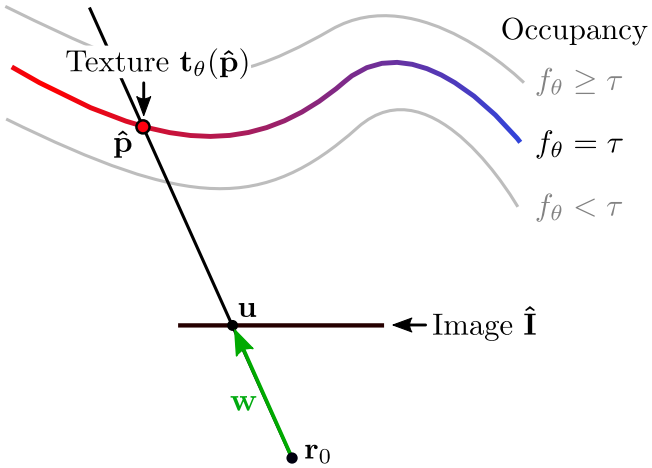


Figure 97

The secant method can be used to find the root of the function without calculating gradients. Given two points

x_0 and x_1, one at either side of the root (these points have been found with ray marching, as discussed before), a line can be constructed between the function evaluations. This line is given by:

$$y_2 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_1) + f(x_1)$$

The root can be found by setting $y_2 = 0$ and solving for x_2:

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

This method is then iteratively repeated with the newly found point. As seen in Fig. 98 this is a good approximation of the actual root if the initial guesses x_0 and x_1 were close to the actual root.

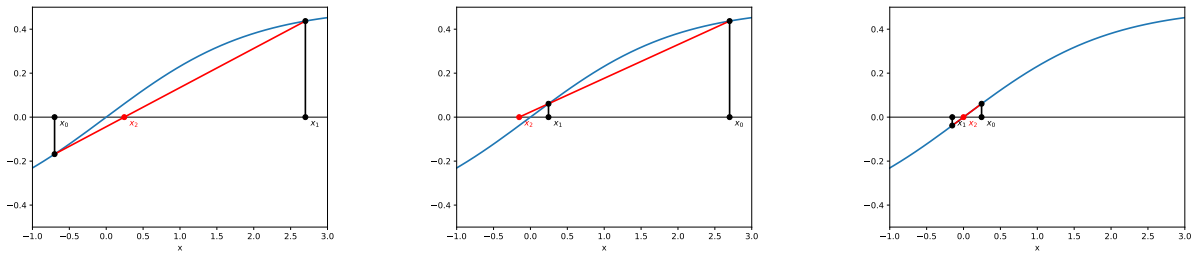


Figure 98: Secant Method

9.2.2 Backward Pass

For the understanding of the backward pass, an understanding of the total derivative is needed. A quick recap is shown here:
Let $f(x, y)$ be a function where $y = y(x)$ depends on x.

The **partial derivative** is defined as:

$$\frac{\partial f(x,y)}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial x}$$

Example:

$$f(x, y) = xy$$
$$\frac{\partial f(x,y)}{\partial x} = y$$

The **total derivative** is defined as:

$$\frac{df(x,y)}{dx} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial x} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial x}$$

Example:

$$f(x, y) = xy \ \wedge \ y = x$$
$$\frac{df(x,y)}{dx} = y + x = 2x$$

Another concept that is needed is the concept of implicit differentiation. A quick recap is shown here:
An **implicit equation** is a relation

$$f(x, y) = 0$$

where $y(x)$ is defined only implicitly.

Implicit differentiation computes the total derivative of both sides wrt. x

$$\frac{\partial f}{\partial x}\frac{\partial x}{\partial x} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} = 0$$

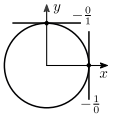
and solves for $\frac{\partial y}{\partial x}$.

Example: Let's assume

$$x^2 + y^2 = 1$$

Implicit differentiation yields

$$2x + 2y\frac{\partial y}{\partial x} = 0$$
$$\frac{\partial y}{\partial x} = -\frac{x}{y}$$



Note the presence of y in this term, i.e., the implicit "function" is a curve.

In the backward pass, the goal is to minimize the difference of the predicted RGB image $\hat{\mathbf{I}}$ and the image observation $\mathbf{I}$. The loss can be defined as follows:

$$\mathcal{L}(\hat{\mathbf{I}}, \mathbf{I}) = \sum_{\mathbf{u}} \|\hat{\mathbf{I}}\mathbf{u} - \mathbf{I}\mathbf{u}\|$$

The gradient of the loss with respect to the parameters of the model which includes the shared backbone, the shape and texture heads can be computed with the chain rule.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta} &= \sum_{\mathbf{u}} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{I}}\mathbf{u}} \cdot \frac{\partial \hat{\mathbf{I}}\mathbf{u}}{\partial \theta} \\ \frac{\partial \hat{\mathbf{I}}\mathbf{u}}{\partial \theta} &= \frac{\partial \mathbf{t}\theta(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial \mathbf{t}\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \frac{\partial \hat{\mathbf{p}}}{\partial \theta}\end{aligned}$$

Noteworthy is that the total derivative is used, as the texture $\mathbf{t}_\theta$ depends on the parameters of the color network but also on the parameters of the optimal point location $\hat{\mathbf{p}}$. Because when the occupancy network changes $\hat{\mathbf{p}}$, this would also change the color value. To calculate $\frac{\partial \hat{\mathbf{p}}}{\partial \theta}$ implicit differentiation is needed. Let's consider the

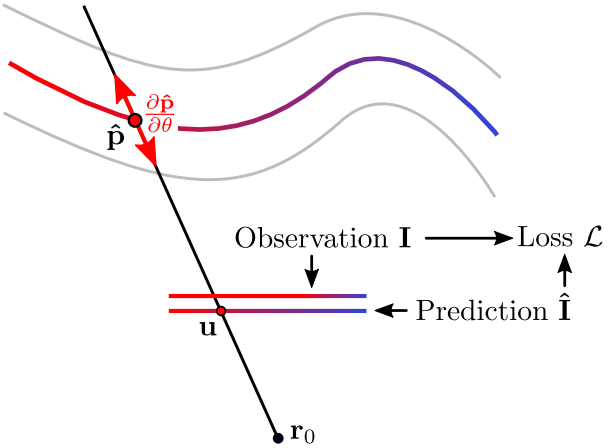


Figure 99

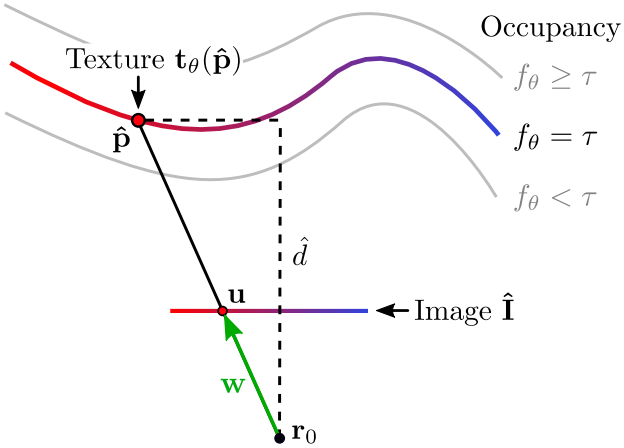


Figure 100

ray: $\hat{\mathbf{p}} = \mathbf{r}_0 + \hat{d}\mathbf{w}$ seen in Fig. 100. By implicitly differentiating $f_\theta(\hat{\mathbf{p}}) = \tau$, where τ is the occupancy threshold, on both sides with respect to θ yields the following:

$$\begin{aligned}&\frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \frac{\partial \hat{\mathbf{p}}}{\partial \theta} = 0 \\ \Leftrightarrow\ &\frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \mathbf{w}\frac{\partial \hat{d}}{\partial \theta} = 0\end{aligned}$$

By solving this expression for $\frac{\partial \hat{d}}{\partial \theta}$ yields:

$$\frac{\partial \hat{\mathbf{p}}}{\partial \theta} = \mathbf{w}\frac{\partial \hat{d}}{\partial \theta} = -\mathbf{w}\left(\frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \mathbf{w}\right)^{-1} \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta}$$

With this, the parameters θ can be updated such that the prediction becomes closer to the observation. This update of the parameter changes both the shape as well as the color.
In contrast to other rendering techniques that work on voxel based representations, there is no need to store intermediate results along the ray. This method can also be used for simple 3D reconstruction by just optimizing the parameters to optimally represent a particular 3D object. Exemplary results are shown in Fig. 101

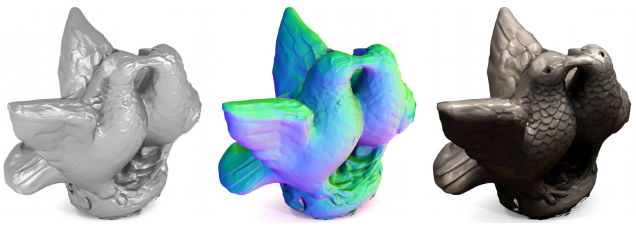


Figure 101: Reconstruction Results

9.3 Neural Radiance Fields

A popular follow-up work of representing shape and appearance implicitly is called Neural Radiance Fields or NeRF for short. The idea of NeRF is to synthesize novel views of sparsely sampled images of a scene. The focus lies not in an accurate 3D reconstruction, but to render novel viewpoints as realistically as possible. One major advantage of this method is that it works for general scenes. In contrast, the DVR method requires pixel aligned masks of the object. One reason why this method is more general is that the representation is slightly

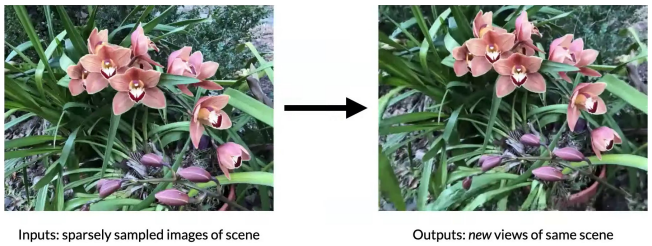


Figure 102: NeRF

different. This makes it easier to optimize but more time-consuming. NeRF also uses a ReLU MLP that maps location and view direction to color and density. The density σ describes how solid or transparent a 3D point is. This way it is possible model e.g. fog. Conditioning on the view direction allows for modeling view-dependent

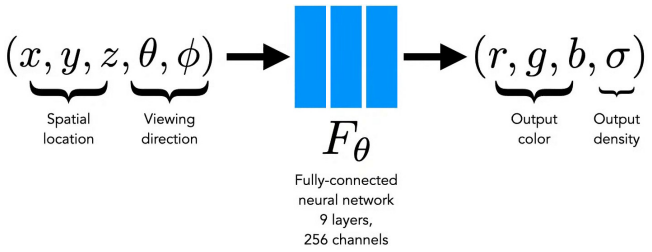
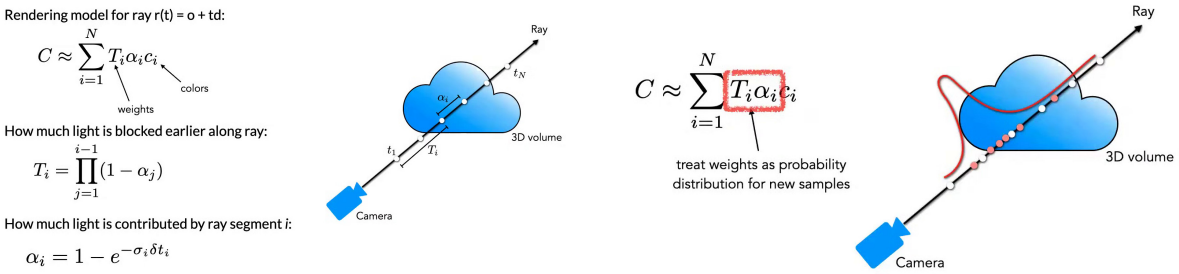


Figure 103: NeRF Model

effects like reflections. Which is important for real world modelling as real world objects are non-lambertian. In practice the view direction is not fed into the network as two separate angles but as one normalized 3D vector.

NeRF works very similar to traditional ray tracing. First a ray is shot through the scene, then points along the ray are sampled. For the sampled points the radiance field is queried to obtain color and density values. Finally aplha composition is applied to obtain the pixel color. (see Fig. 105)

The math behind NeRF is quite simple. The color is estimated along the ray $r(t)$. The color term consists of a weight factor T_i which indicates how much light is blocked earlier along the ray, and a α factor which indicates how much light is contributed by the ray segment i and the color value c_i.



The training is then done by shooting rays, rendering the rays to pixel and minimizing the reconstruction error via backpropagation.

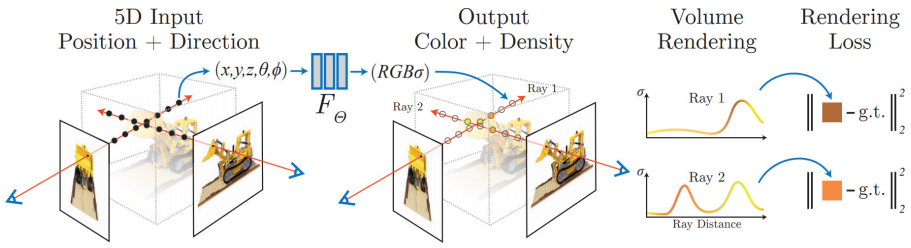


Figure 105: NeRF Training

Although results achieved with NeRF are very impressive, NeRF has some drawbacks. For once, to optimize NeRF's parameters, many different views of a single scene are required. Another remark, is that the sampling is quite slow. If a lot of points are sampled along the rays, many queries of the radiance field are required. To mitigate this, NeRF first allocates the samples more sparsely and equidistantly. Once NeRF is more certain where the surface is located it allocates the samples more densely. Through these two-pass sampling operation seen in Fig. 105 some time and compute is saved.

9.3.1 View Dependent Effects

NeRF can model view-dependent effect, e.g. for non-lambertian materials. This is possible as the input view direction is provided to the radiance field. To not entangle the viewpoint condition with the geometry, the viewpoint conditioning happens later in the MLP. (For more details how this is done see the NeRF paper). Fig. 106 highlights how the same point location can be correctly modelled with specular highlights with NeRF. To

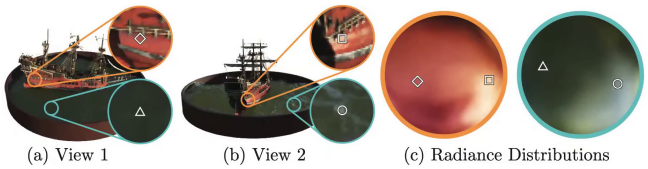


Figure 106: Modelling View-dependent effects

get even better results, NeRF utilizes a so-called *positional encoding* for the input point x and direction d. To show the benefit of this positional encoding the authors looked at a more simple task which is memorizing a simple image in RGB space. But standard fully-connected nets surprisingly produced oversmooth results with ReLU artifacts. To solve this problem, the authors did not pass the position into the network but a positional encoding with random Fourier features of varying frequencies. These features let networks learn high-frequency

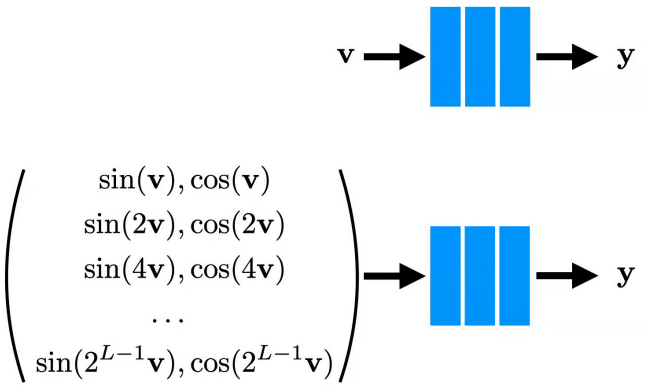


Figure 107: Fourier Encoding

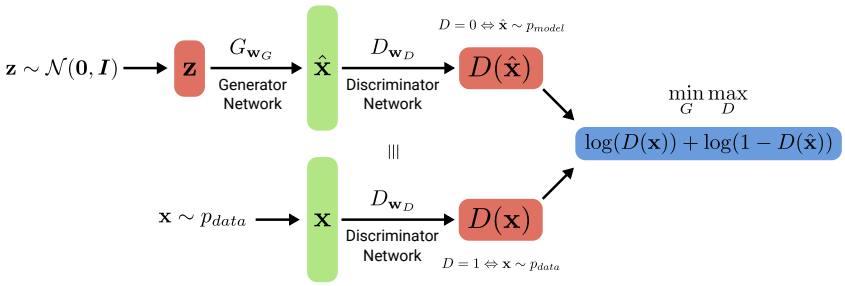
functions in low-dimensional domains. With the positional encoding the results were much sharper and more detailed. The convergence was also much faster.

9.4 Generative Radiance Fields

In contrast to the last section, the goal is now to build a generative model with Neural Radiance Fields that can generate photorealistic 3D objects. The difference to before is that NeRF is always optimized for a particular

scenes given many viewpoints where the camera poses have been known. This assumption does not hold anymore. The only given data is a collection of images of objects of a particular category. The camera pose is also not known. This method is called Generative Radiance Fields or GRAF in short. For better understanding, a quick recap of Generate Adversarial Networks (GANs).

- The term **generative model** refers to any model that takes a dataset drawn from p_{data} and learns a probability distribution p_{model} to represent p_{data}
- In some cases, the model estimates p_{model} **explicitly** and therefore allow for evaluating the (approximate) likelihood $p_{model}(\mathbf{x})$ of a sample $\mathbf{x}$
- In other cases, the model is only able to **generate samples** from p_{model}
- GANs are prominent examples of this family of **implicit models**
- They provide a framework for training models without explicit likelihood



D and G play the following **two-player minimax game** with value function $V(G, D)$:

$$G^*, D^* = \underset{G}{\operatorname{argmin}} \ \underset{D}{\operatorname{argmax}} \ V(D, G)$$

$$V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

- **Theoretical analysis** shows that this minimax game recovers $p_{model} = p_{data}$ if G and D are given enough capacity and assuming that D^* can be reached
- In practice, however, we must use iterative numerical optimization and optimizing D in the inner loop to completion is computationally prohibitive and would lead to overfitting on finite datasets
- Therefore, we resort to **alternating optimization:**
  - k steps of optimizing D (typically $k \in \{1, \ldots, 5\}$)
  - 1 step of optimizing G (using a small enough learning rate)
- This way, we maintain D near its optimal solution as long as G changes slowly

For further information of GANs see the Deep Learning Lecture.

Generative Radiance Fields are the idea of GANs applied to radiance fields. A latent code can be sampled which contains a 3D point location $\mathbf{x}$, a viewing direction $\mathbf{d}$, information about the shape and appearance of the object $\mathbf{z}$ as well as the camera pose. With the radiance field the color and density can be predicted. Then with volumetric rendering the scene can be rendered. The goal is that the discriminator applied on that scene cannot tell the difference between the generated scene and a real 2D image. (The process is shown in Fig 108) The

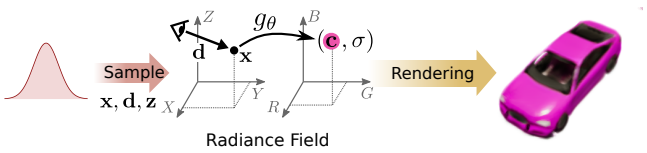


Figure 108: GRAF Overview

training is done with only unstructured and unposed 2D images. (Unposed meaning the pose is not known). The challenges with this approach is that the discriminator cannot be applied in 3D as only 2D images are available. So the slow volumetric rendering step is mandatory.

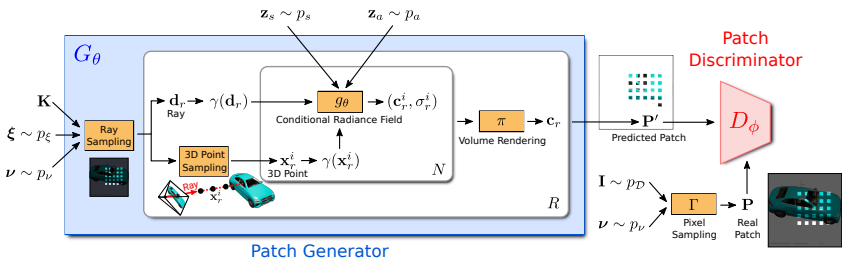


Figure 109: GRAF Network

Let's examine the model which is shown in detail in Fig. 109. A radiance field g_θ maps a 3D point $\mathbf{x}_r^i$ and viewing direction $\mathbf{d}_r$ to a color and a density. By sampling N points along the ray, it is possible to do volumetric rendering π to get the pixel's color $\mathbf{c}_r$. Additionally, GRAF conditions the radiance file g_θ on additional latent codes $\mathbf{z}$. Here, $\mathbf{z}_s$ is a shape latent code and $\mathbf{z}_a$ is the appearance latent code.
Because rendering the entire image would be too time-consuming, only a few pixels will be rendered instead. The pixels are chosen randomly from a small patch. The generator repeats this process for R rays. The camera intrinsics K, the extrinsics ξ and the 2D sampling grid are sampled randomly. The generated patch is then of the same size as the image patch. For the real image the same patch sampling is applied. A Patch Discriminator which is a simple 4 layer ConvNet then discriminates whether the sample is real or a generated image patch. This approach reduces the problem quite a lot as it is not necessary anymore to render the entire image, instead sparse random sampling is done. This is fast enough to being trained in the context of a generative adversarial network objective.

The predicted volume density σ depends solely on the 3D point x and the shape code bz_s. To make sure that the shape code $\mathbf{z}_s$ and the appearance code $\mathbf{z}_a$ are not entangled in the Conditional Radiance Field g_θ, the appearance code $\mathbf{z}_a$ is introduced later in the network.

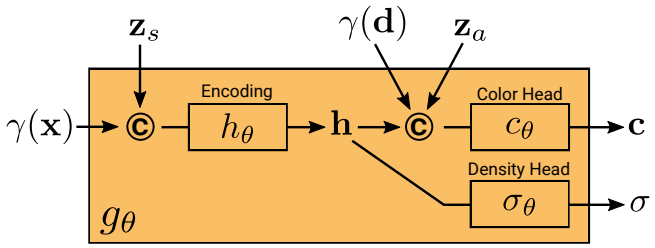


Figure 110: Contitional Radiance Field Network Architecture

9.5 Summary

In this lecture, neural networks as continuous shape/appearance/motion representations were introduced.
Occupancy Networks were one of the first networks to do this. They were published at the same conference with **Scene Representation Networks** and **DeepSDF** which follow a very similar principle by taking the same input which is the 3D position. **Neural Radiance Fields** take as input the 3D position and the view direction and predicts color and density. In **Differentiable Volumetric Rendering** predict the color and occupancy from a 3D location. Finally **Generative Radiance Fields** take the 3D location, view direction and a latent shape and appearance code to predict novel color and density.
But all models share a pretty naive MLP as a simple predictor for the implicit representation. In summary, coordinate-based networks form an effective output representation for shape, appearance, material and motion. They don't require any discretization and are able to model arbitrary topology. They can be learned from images via differentiable rendering and there are many applications in 3D reconstruction, motion estimation view synthesis robotics etc.
However, the output representation is implicit that means certain properties such as geometry must be extracted in a post-processing step which takes some time. The extension to higher dimensions is not necessarily straightforward due to the curse of dimensionality. Also, fully connected architectures and global conditioning lead to over smooth results.
But there are also promising directions such as using local features as well as better input encoding as proposed in NeRF.

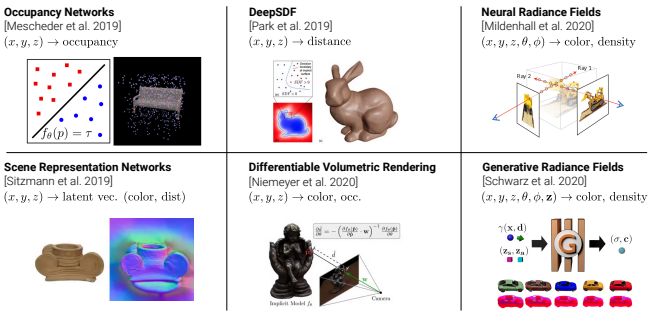


Figure 111: Summary Neural Networks as Continuous Shape Representations

10 Recognition

In the previous lectures we have mainly focused our attention on the *geometric* aspects of computer vision. In this lecture, we are going to touch on another important sub-field of computer vision – **recognition**. This is a very data-driven field which has benefited a lot from the advances of *computational resources* along with the great scientific strides that *deep learning* has made over the last 10 years. In the first unit of the lecture we are going to cover **image classification** – one of the fundamental grand goals of computer vision. Next, we are going to cover an equivalently important problem known as **semantic segmentation**. Last but not least, we are going to discuss **object detection and segmentation**.



Figure 112: **Image classification**

10.1 Image classification

Simply put, the goal of **image classification** is to assign a single class label to an image (see Fig. 112). Now that we have defined what image classification is, let us provide further *motivation*. First of all, it is an extremely *useful* task on its own (eg. it allows for effective image search from a large database). Furthermore, it is a very *standardized task* – it embodies simple input/output specification (eg. inputs are 224x224 images, and outputs are singular labels such as "cat", "dog", etc.) and can be trained using standard loss functions (such as categorical cross entropy). In fact, for a long time this has been one of the difficult *gold-standard* tasks in computer vision. Finally, it is a very useful high-level proxy-task for learning good representations which can be *transfered* to a completely new task/domain (this process is also known as *transfer learning*).
Early attempts of **hand-engineering** the features and algorithms for image classification failed, as object shapes and appearances are simply too hard to describe. For this reason, we need **machine learning** and big labeled datasets to learn these relationships!



Figure 113: **The MNIST dataset**

10.1.1 Datasets

The **MNIST dataset** (see Fig. 113) is one of the most popular labeled datasets in Machine Learning today. There are many variants, and interestingly, is still in use today. This dataset is based on data from the National Institute of Standards and Technology. It consists of *hand written digits* from Census Bureau employees and high school students. The images are of 28 x 28 pixels resolution, and is usually split between 60k training samples, and 10k test samples.

The **Caltech101** dataset was the first major object recognition dataset, dating back to 2003. The images are hand-curated from Google Image Search, amounting to 101 object categories. However, this is a highly *biased* dataset, since the objects are of canonical size and location.

Lastly, we mention **ImageNet**. Today, this is one of the most widely used datasets in the machine learning community. It contains 15,000,000 images amounting to 22,000 categories. In 2012, the AlexNet results on the ImageNet challenge started the **deep learning revolution**. Furthermore, it is one of the primary datasets for **pre-training** generic representations. The main idea of **transfer learning** is to pretrain a large model (such as a Convolutional Neural Network) on a big dataset (such as ImageNet) for a generic task. Then, one fine-tunes the last (or all) layers on a smaller dataset concerned with a new task. This paradigm more often than not leads to state-of-the-art performance.

Nevertheless, image classification has been, and still is, a challenging task. Some of the difficulties related to this learning task are: large number of categories (10,000-30,000), intra-class and viewpoint variations, illumination changes, background clutter, deformation, and occlusion.

10.1.2 Simple models

One of the simplest ideas for an image classification model is to use the **nearest neighbor paradigm**. In particular, we first define a distance function:

$$d(\mathbf{I}_1, \mathbf{I}_2) = \sum_{\mathbf{p}} |I_1(\mathbf{p}) - I_2(\mathbf{p})|_1$$

Then, given a query image $\mathbf{I}$, we find its nearest neighbor in our dataset:

$$\mathbf{I}^* = \underset{\mathbf{I}' \in \mathcal{D}}{\operatorname{argmin}} \; d(\mathbf{I}, \mathbf{I}')$$

Finally, we classify the query image $\mathbf{I}$ with the class of $\mathbf{I}^*$. Note that this algorithm is very **slow**, since we have to perform a nearest-neighbor search at test time (although, there is no training phase). Furthermore, this is a **bad** classifier, since pixel distances are extremely uninformative. In order to demonstrate the weakness of the classifier, consider two checkerboard patterns, where one is just a horizontal displacement of the other by one field. Even though, semantically, they are the same object, according to our method described above, the two images have maximal distance.

The main motivation behind the **Bag-of-Words (BoW)** classifier is to obtain spatial invariance by comparing **histograms of local features**. The idea of this model originates from Natural Language Processing, where these models have been used to build an **orderless** document representation through the **frequencies of words** from a given dictionary.



Figure 114: **Extracted features**

In the computer vision setting, the algorithm could be described in few basic steps:

1. **Extract features**, for example by using SIFT detector.
2. **Learn visual vocabulary**, for example by running K-means on the SIFT feature vectors. The center of each cluster $\{1, \ldots, K\}$ will be used as the visual dictionary's vocabularies.

3. **Quantize features** into visual words using the learnt vocabulary, for example by running nearest neighbors. In particular, we assign each feature to a cluster $\{1, \ldots, K\}$ based on the nearest neighbors in the SIFT feature space.

4. Represent the images by **histograms** of visual word frequencies.

5. **Train a classifier** (for example, k-NN, SVM, random forest, neural network) using these histograms as features.

Nevertheless, the problem of many **hand-designed components** still persists in the Bag-of-Words model. A solution for this is to learn the entire model **end-to-end** from data.

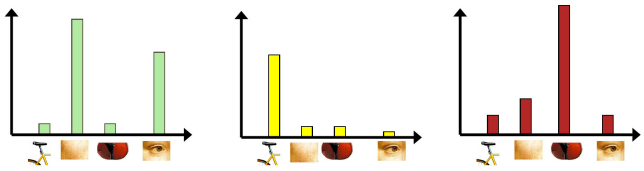


Figure 115: **Histogram of visual words**

Convolutional Neural Networks are a type of an Artificial Neural Network model particularly suitable for image recognition. As we discussed in the previous section, these models can be learnt end-to-end, thus solving the problem of having to hand-engineer features and components. These models typically have three **types of layers**: convolutional layers, downsampling layers, and fully connected layers. The model takes an input image (for ex. 224x224x3), and successively decrease the spatial dimensions as the image is passed through the network.

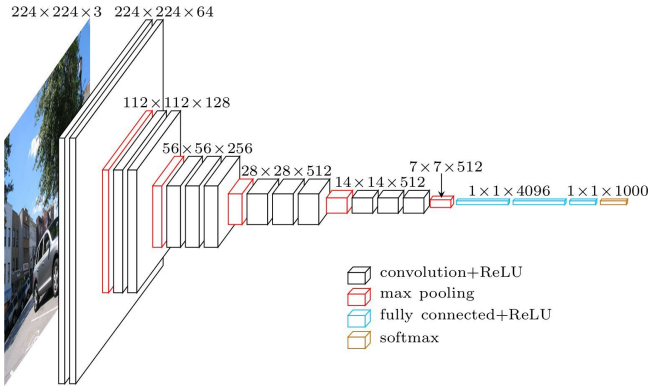


Figure 116: **Convolutional neural network**

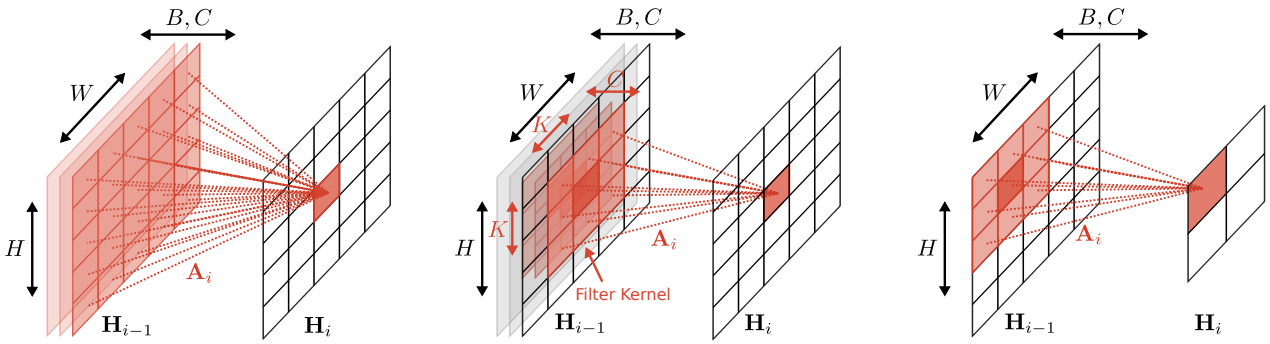
At the same time, the model increases the number of feature channels in order to improve the **expressiveness**. At the end, typically there is a 1xK dimensional vector (where K is the number of classes to be recognized), upon which we utilize the softmax operator in order to obtain a valid probability distribution over the classes.

In fully connected layers (see Fig. 117a) each neuron in the successive layer is connected to all neurons in the previous layer. On the other hand, in convolutional layers (see Fig. 117b) each neuron in the successive layer is connected only to a small neighborhood in the previous layer. Furthermore, each neuron in a given layer (and a given channel) **shares the weights**, thus drastically reducing the number of parameters. Convolutional layers are **translation equivariant**:

$$\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$$

meaning that the order of applying a translation $\mathcal{T}_\theta$ and convolution f doesn't matter. In other words, if we translate the input to the convolutional layer, then the output of the convolution gets translated in the same way.

As we discussed before, downsampling **reduces the spatial resolution** in effort to **increase the receptive field** (i.e. which pixels in the input influence a neuron). A common downsampling paradigm is pooling (see Fig. 117c), though it has several variants: max, min, mean. Typically, pooling is performed with a stride $s = 2$ and kernel size 2×2, amounting to **reduction in the spatial dimension** by a factor of 2. It is worth noting that pooling has **no parameters**, and is applied to **each channel separately**.



(a) **Fully connected layers** (b) **Convolutional layers** (c) **Pooling layers**

Figure 117: **Neural Network Layers**

Finally, the last layers of the network are typically fully connected. These layers are the most **memory intensive** part of the model. Let us now discuss a suitable choice for the output activations and the loss function, in order to successively train models for image classification.

In order to be able to predict classes (or categories), we need to make use of the **categorical** distribution. This is a discrete distribution, which for each class c assigns a probability value μ_c:

$$p(y = c) = \mu_c$$

A more useful alternative notation is to represent the target vector $\mathbf{y}$ as a one hot encoding, such that $y_c \in \{0, 1\}$ (ex. $\mathbf{y} = (0, \dots, 0, 1, 0, \dots, 0)^\top$ with all zeros except for one (the true class)):

$$p(\mathbf{y}) = \prod_{c=1}^{C} \mu_c^{y_c}$$

Now, let $p_{model}(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \prod_{c=1}^{C} f_{\mathbf{w}}^{(c)}(\mathbf{x})^{y_c}$ be a **categorical distribution**. From probability theory, we know that maximizing this (log-) likelihood leads to minimizing the **cross-entropy (CE)** loss:

$$\begin{aligned}
\hat{\mathbf{w}}_{ML} &= \operatorname*{argmax}_{\mathbf{w}} \sum_{i=1}^{N} \log p_{model}(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \sum_{i=1}^{N} \log \prod_{c=1}^{C} f_{\mathbf{w}}^{(c)}(\mathbf{x}_i)^{y_{i,c}} \\
&= \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{N} \underbrace{\sum_{c=1}^{C} -y_{i,c} \log f_{\mathbf{w}}^{(c)}(\mathbf{x}_i)}_{\text{CE Loss}}
\end{aligned}$$

Finally, we need to ensure that $f_{\mathbf{w}}^{(c)}(\mathbf{x})$ predicts a **valid categorical distribution**. In particular, we must guarantee that:

1. $f_{\mathbf{w}}^{(c)}(\mathbf{x}) \in [0, 1]$
2. $\sum_{c=1}^{C} f_{\mathbf{w}}^{(c)}(\mathbf{x}) = 1$

The **softmax** function guarantees both of these properties:

$$\text{softmax}(\mathbf{x}) = \left(\frac{\exp(x_1)}{\sum_{k=1}^{C} \exp(x_k)}, \dots, \frac{\exp(x_C)}{\sum_{k=1}^{C} \exp(x_k)} \right)$$

Let the score vector $\mathbf{s}$ denote the network output after the last affine layer. Then:

$$f_{\mathbf{w}}^{(c)}(\mathbf{x}) = \frac{\exp(s_c)}{\sum_{k=1}^{C} \exp(s_k)} \quad \Rightarrow \quad \log f_{\mathbf{w}}^{(c)}(\mathbf{x}) = s_c - \log \sum_{k=1}^{C} \exp(s_k)$$

Let us build intuition for the log-softmax formulation. Assume c is the correct class. Then our goal is to **maximize** the above mentioned criterion $\log f_{\mathbf{w}}^{(c)}(\mathbf{x})$. The first term encourages the score s_c for the correct class c to increase. On the other hand, the second term (which can be approximated by: $\log \sum_{k=1}^{C} \exp(s_k) \approx \max_k s_k$ as $\exp(s_k)$ is insignificant for all $s_k < \max_k s_k$) penalizes the most confident predictions. So, if the correct class already has the largest score (i.e., $s_c = \max_k s_k$), both terms roughly cancel and the example will contribute little to the overall training cost. On the other hand, if the correct class doesn't have the largest score, then we incur loss.

It is worth noting that softmax only responds to differences between the inputs, hence it is invariant to adding the same scalar to all of the inputs:

$$\text{softmax}(\mathbf{x}) = \text{softmax}(\mathbf{x} + c)$$

We can therefore derive a numerically more stable variant:

$$\text{softmax}(\mathbf{x}) = \text{softmax}(\mathbf{x} - \max_{k=1..L} x_k)$$

which allows for accurate computation even when $\mathbf{x}$ is large.

Let us observe the behavior of the softmax function and the Cross-Entropy loss through a simple example:

Input $\mathbf{x}$	Label $\mathbf{y}$	Predicted scores $\mathbf{s}$	softmax($\mathbf{s}$)	CE Loss
	$(1,0,0,0)^\top$	$(+3,+1,-1,-1)^\top$	$(0.85,0.12,0.02,0.02)^\top$	0.16
	$(0,1,0,0)^\top$	$(+3,+3,+1,+0)^\top$	$(0.46,0.46,0.06,0.02)^\top$	0.78
	$(0,0,1,0)^\top$	$(+1,+1,+1,+1)^\top$	$(0.25,0.25,0.25,0.25)^\top$	1.38
	$(0,0,0,1)^\top$	$(+3,+2,+3,-1)^\top$	$(0.42,0.16,0.42,0.01)^\top$	4.87

Observe that for the first sample, the classifier is most confident in the correct label, thus we incur only a small loss. On the other hand, for the fourth sample, the classifier is least confident in the correct label, thus incurring a very high loss.

10.1.3 Network architectures

The first prominent example to demonstrate the ability of Convolutional Neural Networks working on (relatively small) images was the **LeNet-5 model**. It only contains 2 convolution layers with (5×5) kernels, 2 pooling layers with (2×2) kernels, and 2 fully connected layers. This model achieved state-of-the-art accuracy on MNIST.

The first prominent example of a Convolutional Neural Network working on the large ImageNet dataset was **AlexNet**. It triggered the deep learning revolution by demonstrating that CNNs can work well in practice. This model consists of 8 layers, and makes use of ReLU activations, dropout layers and data augmentation – paradigms still widely popular today.

The **VGG model** is another popular model, which demonstrated that deeper networks work better. It uses convolutions with a 3×3 kernel everywhere, which yielded the same expressiveness, albeit fewer parameters. In particular, three 3×3 layers have the same receptive field as one 7×7 layer, but have less parameters. There are two variants with 16 and 19 layers respectively.

The **Inception network** is another large model which demonstrated competitive performance. It consists of 22 layers, grouped in inception *modules* which utilize convolution/pool operations with varying filter size. Moreover, this model uses multiple *intermediate* classification heads in order to improve *gradient flow*. Since it doesn't make use of fully connected layers (which are exchanged with global average pooling), it has 27 times less parameters than VGG-16!

ResNet demonstrated that it is possible to train very deep models (up to 152 layers), by the use of *residual connections*. Standard deep networks of such sizes suffer from vanishing gradients. The residual connections resolve this problem by allowing for more efficient gradient flow to the earlier layers. The ResNet model is in fact a very simple and regular network structure with 3×3 convolutions. Moreover, instead of the standard max pool operation, it uses strided convolutions for downsampling. Today, ResNet-like architectures are dominating.

10.2 Semantic Segmentation

The goal of **semantic segmentation** is to assign a semantic label to every pixel in the image – both objects and scenes (see Fig. 118). Even though there have been many attempts to solve this problem in the past

using standard techniques from computer vision, in this section we only stick to deep models, as they have demonstrated great improvements over the former models.

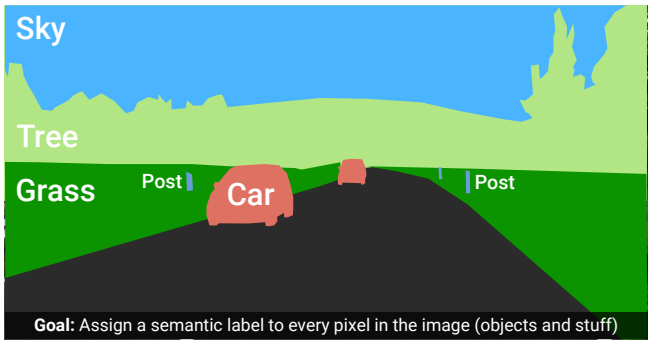


Figure 118: **Semantic segmentation**

10.2.1 Fully Convolutional Networks

In 2015, the authors of [30] proposed a new approach for semantic segmentation by applying a classification network in convolutional fashion in order to obtain class heatmaps. The networks are trained similarly as before, by using cross entropy loss at every pixel of the output. However, due to the downsampling operations, the output heatmaps are **low resolution** (see Fig. 119).

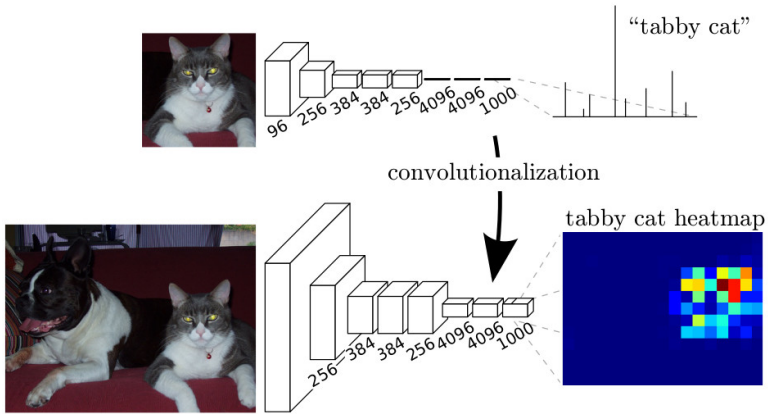


Figure 119: **Fully Convolutional Networks**

On the other hand, using only convolutional filters doesn't work due to the small receptive field (one would need many more layers to achieve the same receptive field using exclusively convolutional filters instead of occasionally employing pooling layers). The idea was then to **learn an upsampling** operation and combine low and high level information. Even though fusing information from layers with different strides improved details, the results at the time were still relatively coarse.

10.2.2 SegNet

SegNet is another model that uses this idea of down and upsampling (see Fig. 120). The idea is similar – downsampling provides strong features with large receptive field; upsampling yields outputs at the same resolution as the input.

For the upsampling operation, the authors have used **Max Unpooling**. This operation is performed by **remembering** which element was maximum during the pooling operation, and then injecting the new element during the upsampling step in the same position (see Fig. 121). Naturally, this requires corresponding pairs of downsampling and upsampling layers.

10.2.3 Dilated convolutions

Dilated convolutions are an alternative to combining downsampling and upsampling operations in order to reach a large receptive field size quickly. This operator increases the receptive field of standard convolutions **without**

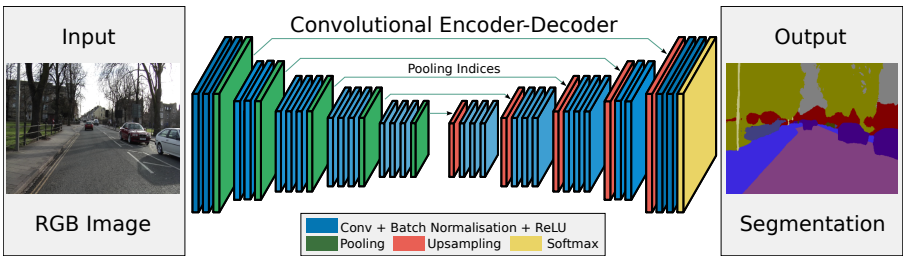


Figure 120: **SegNet**

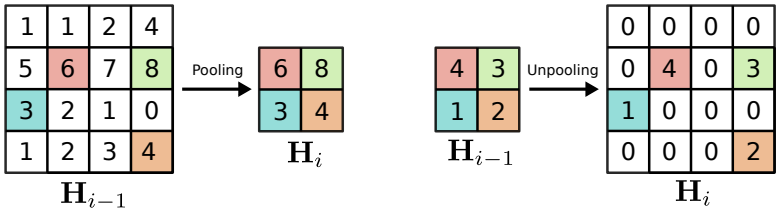


Figure 121: **Max Unpooling**

increasing the number of parameters (see Fig. 122).

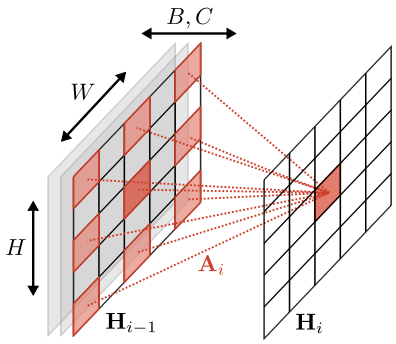


Figure 122: **Dilated convolution**

Thus, a network with dilated convolutions is able to perform image-level predictions, such as semantic segmentation, without upsampling and downsampling. Nevertheless, in practice dilated convolutions are often combined with standard backbone architecture.

10.2.4 U-Networks

Somewhat similar to the previous approaches, U-Nets use max-pooling, up-convolutions and skip-connections that perform concatenation. This architecture has become the defacto standard for many tasks with image output, such as depth and segmentation.

10.2.5 Cityscapes

Cityscapes is a standard dataset for semantic segmentation in the field of autonomous driving. The difficulty here is that, other than the fact that the model should perform accurate segmentation, the outputed results should be teomporally consistent across the sequences. There are several evaluation metrics, such as: per-pixel accuracy, per-class class Jaccard Index / Intersection-over-Union (IoU), IoU weighted by inverse instance size, etc.

10.2.6 Panoptic segmentation

The idea is to combine semantic and instance segmentation – predict semantic category for both objects and scenes, as well as instance label for each object (see Fig. 124). Note that semantic segmentation associates every pixel of an image with a class label such as: person, flower, car and so on. It treats multiple objects of the same class as a single entity. In contrast, instance segmentation treats multiple objects of the same class as distinct individual instances.

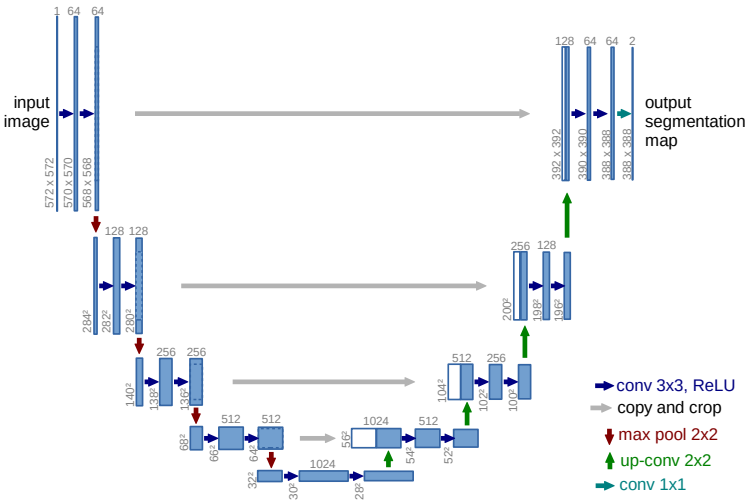


Figure 123: **U-Nets**

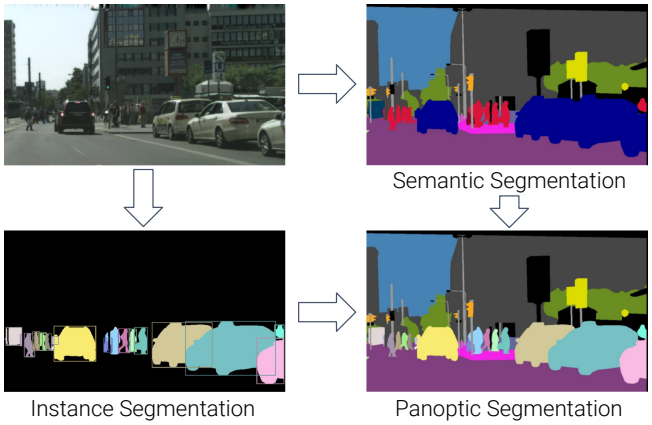


Figure 124: **Panoptic segmentation**

10.3 Object Detection and Segmentation

The goal of **object detection** is to localize (put bounding boxes) and classify all objects in the image (see Fig. 125). The problem setting is as follows: as inputs, the algorithm receives RGB images or laser range scans; as outputs, the algorithm produces 2D/3D bounding boxes with category label and confidence. Note that the number of objects and the object sizes are not known a priori.

10.3.1 Performance evaluation

While in the case of image classification and sematic segmentation the evaluation of the performance of the models is pretty straightforward, that is not the case for object detection. Among several candidates, the one most widely used in practice is **IoU - Intersection over Union** (see Fig. 126). In general, a detection is considered successful only if $IoU > 0.5$.

However, the problem remains – how to fairly measure detection performance in case of multiple objects? For this purpose, a fairly standard evaluation is the **Average Precision Metric**. One can calculate this metric in a couple of steps:

1. Run the detector with varying thresholds (typically, we run the detector only once, and then consider several thresholds)
2. Assign detections to closest object
3. Count **True Positives TP** (number of objects correctly detected, i.e. $IoU > 0.5$), **False Negatives FN** (number of objects not detected, i.e. $IoU < 0.5$) and **False Positives FP** (wrong detections)

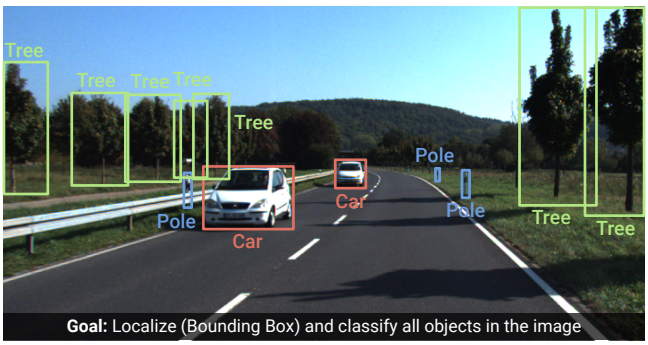


Figure 125: **Object detection**

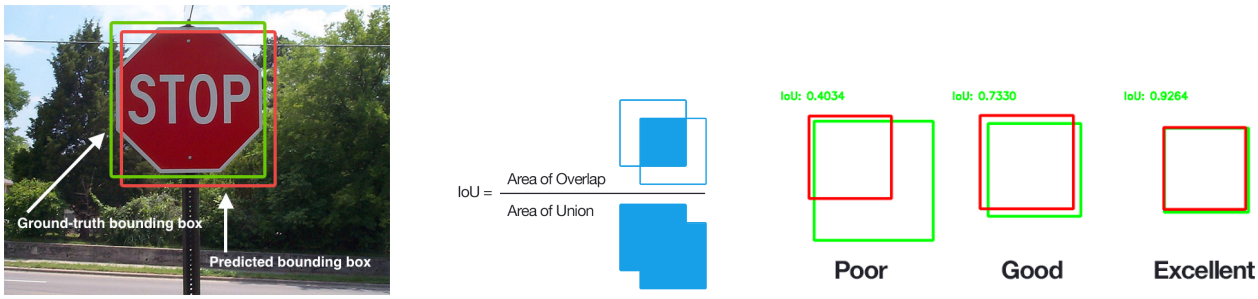


Figure 126: **Intersection over Union**

4. Compute Precision (P), recall (R), and finally – **Average Precision (AP)**:

$$\text{Precision } P = \frac{TP}{TP + FP}$$

$$\text{Recall } R = \frac{TP}{TP + FN}$$

$$\text{Avg. Prec. } AP = \frac{1}{n} \sum_{R \in \{1,\ldots,n\}} \max_{R' \geq R} P(R')$$

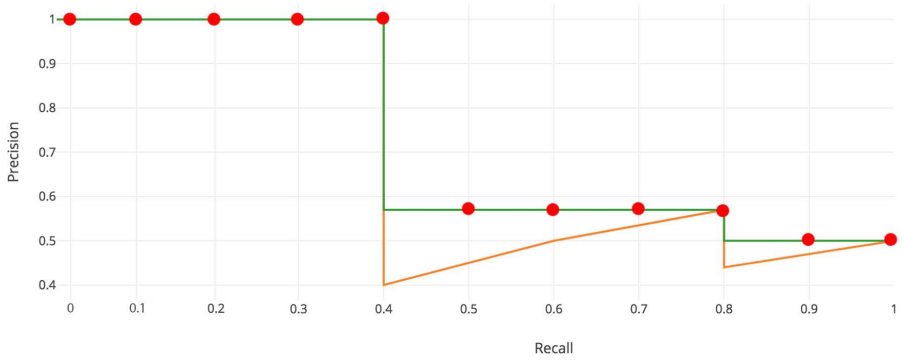


Figure 127: **Precision Recall curve**

10.3.2 Object detection using sliding windows

The idea behind this model is relatively simple – one runs **sliding window** of fixed size over the image and extracts features for each window. Then, each crop is **classified** (object vs. background) using SVM, random forest, boosting, etc. In order to improve performance, one should iterate over multiple aspect ratios/scales and perform **non-maxima suppression**

As for which feature space to use, there are a couple of options. While directly using the RGB pixel space is relatively easy and straightforward, it does not yield the best performance since it is neither viewpoint nor

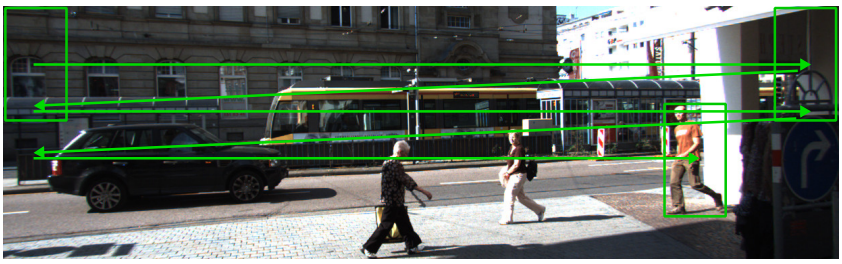


Figure 128: **Object detection using Sliding windows**

illumination invariant. A better approach is to use **Histograms of Oriented Gradients (HoG)**, where one represents patches of the image with histograms of gradient angles weighted by the respective magnitude (see Fig. 133). Due to its invariance to small deformations such as translation, scale, rotation, and perspective, HoG has been the defacto standard for over 10 years.

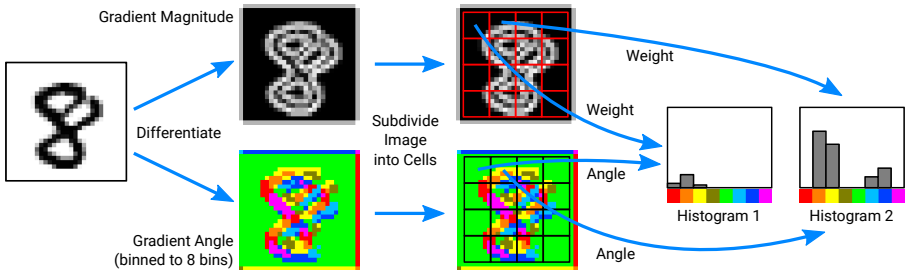


Figure 129: **Histograms of Oriented Gradients**

Part-based models is another class of algorithms for object detection. The main idea is to model an object based on its parts, by modeling a distribution of part configurations. It allows for even more invariance, such as non-rigid deformations. However, it is characterized with an even slower inference, and there is not much gain with respect to multi-view HoG models.

Classical approaches haven't lead to major breakthroughs. One of the reasons is because computer vision features are very hard to hand-engineer – it is unclear how a good representation should look like. Furthermore, sliding window approaches are not fast enough for today's applications in practice. Lastly, computation is often not efficiently re-used. However, with the recent advent of deep learning, things have turned for the better.

10.3.3 Object detection with deep neural networks

The **Region-based Convolutional Neural Network (R-CNN)** was one of the first successful deep networks that could perform object detection. We define the algorithm pipeline in several steps

1. First we use an off-the-shelf region/object detection proposal algorithm (ex. Selective search, Edge Boxes, MCG) to generate roughly 2000 proposals per image.
2. Next, we crop and warp each proposal image window to obtain a fixed-size network input.
3. Then, we forward propagate each fixed-size input through a ConvNet in order to get a feature representation.
4. Using the same backbone, we construct two heads: one is a *linear classifier* which labels the proposed region; and the other is a *box regressor* which refines the proposed localization.

Each of the steps of the algorithm are summarized in Fig. 130. Some of the drawbacks of the R-CNN algorithm are as follows:

- Heavy per-region computation (e.g., 2000 full network evaluations)
- There is no computation/feature sharing
- Slow region proposal method increases the overall runtime of the algorithm
- Generic region proposal techniques have limited recall

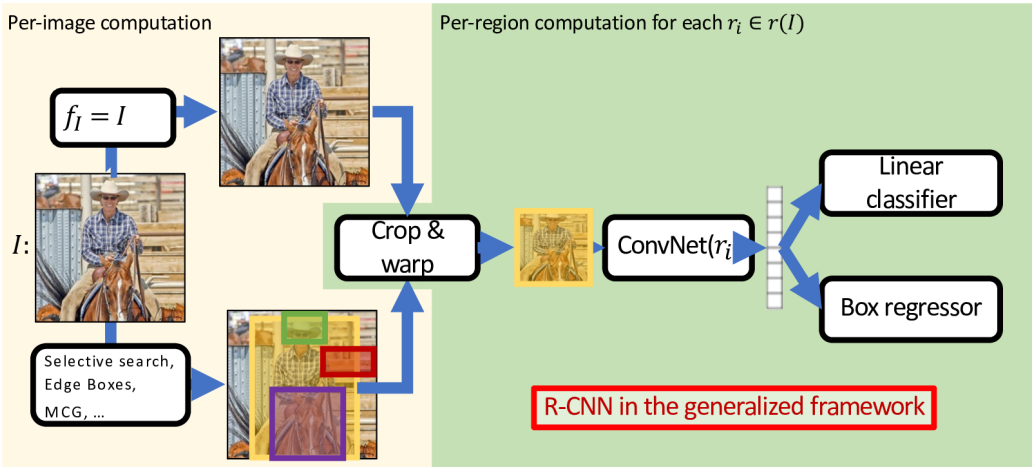


Figure 130: **R-CNN**

In order to resolve (some of the) problems mentioned above, **Fast R-CNN** was introduced. The main changes to the above pipeline are as follows:

1. **Fully convolutional network (FCN)** maps the image to a lower resolution spatial feature map. One can use any standard convolutional network as "backbone" architecture (e.g. AlexNet, VGG, ResNet). However, the global pooling operators should be removed, since the output spatial dimensions should be proportional to the input spatial dimensions. Naturally, the better the backbone, the better the performance.
2. **Region of Interest (RoI) pooling** converts each region into a fixed dimensional representation. This is done by first snapping the Region-of-Interest to the nearest grid cell structure, and then performing a max-pool operation onto a fixed dimensional representation.
3. Then, a lightweight **MLP** processes each region feature map. Recall, this is possible since we have standardized our inputs to a fixed size.
4. Finally, same as before, a classification head and box regression head are applied at the end.

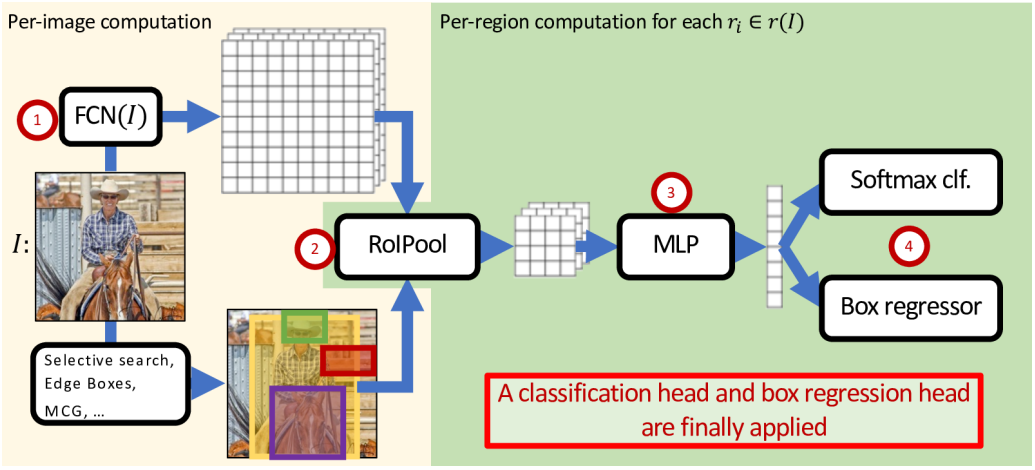


Figure 131: **Fast R-CNN**

The updated model can be seen on Fig. 131. It is worth noting that this model resolves the first two issues of the previous model:

- ~~Heavy per-region computation (e.g., 2000 full network evaluations)~~
- ~~There is no computation/feature sharing~~
- Slow region proposal method increases the overall runtime of the algorithm
- Generic region proposal techniques have limited recall

However, the drawbacks are still obvious – the model strongly relies on the region proposal method.

In this iteration, we introduce **Faster R-CNN**, which gets rid of the traditional region proposal method. The main idea is to construct a **Region Proposal Network** (RPN) which works on the output feature map of the Fully convolutional network (FCN), instead of the image directly. This way the proposals are learned, and the computations are shared.

In this new pipeline, regions are proposed by sliding a 3×3 window over the convolutional feature map of the FCN, which embodies an objectness classifier predicting the probability scores $[0, 1]$, and a box regressor predicting the box shape and shape parametrized by dx, dy, dh, dw. A small caveat is that these predictions are not w.r.t. the 3×3 sliding window, but w.r.t an **anchor box** (see Fig 132).

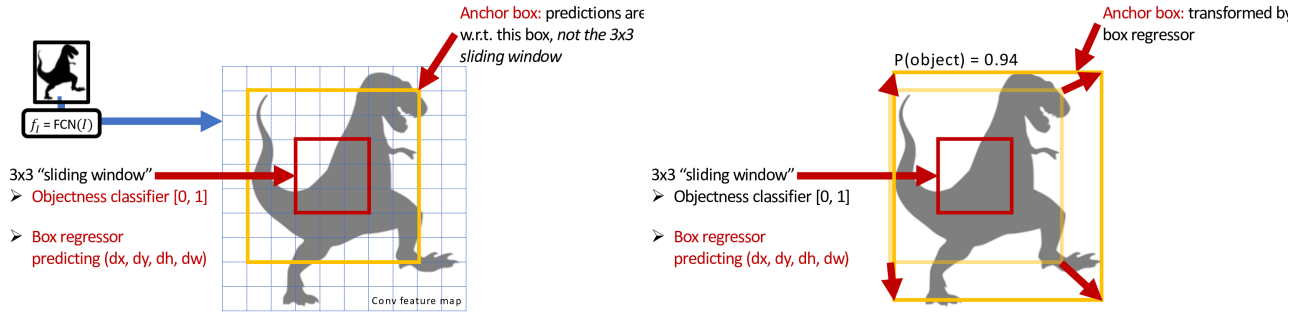


Figure 132: **Region proposal network**

In order to be able to detect multiple objects per location, one can use K anchors per location with different scales and aspect ratios. At this point, it is worth noting that there are methods, such as YOLO and SSD, which predict the 2D bounding boxes in a single stage instead of using region proposal techniques. Even though they are usually faster, they often perform worse in practice.

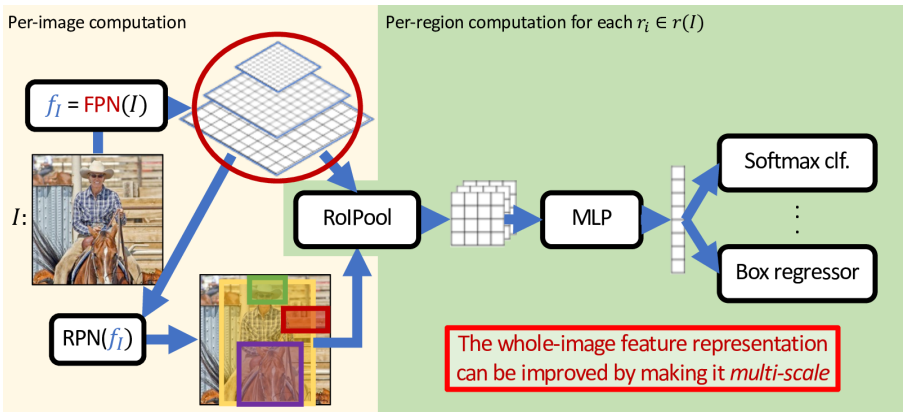


Figure 133: **Faster R-CNN** with a Region Proposal Network (RPN) and a Feature Pyramid Network (FPN)

The whole-image feature representation can be improved by making it *multi-scale*. For this purpose, we further update our model by exchanging the previous Fully Convolutional Network (FCN) with a **Feature Pyramid Network (FPN)** .

As we have already discussed, the goal of this network is to improve the *scale equivariance*. In fact, there are several possible approaches to allow for multi-scale detection. One possibility is to use a featurized image pyramid (e.g. Viola and Jones, HOG, SPP-net) – however this is very slow. Another approach is to use a single feature map from the image (e.g. Fast/er R-CNN, YOLO) – fast, but suboptimal. A third approach is to use pyramidal feature hierarchy (e.g. SSD) – fast, but suboptimal since the features are strong at low resolutions and weak at higher ones. Lastly, the **Feature Pyramid Network** allows for top-down enrichment of high-resolution features – fast, and less suboptimal. It uses a U-net-like architecture in order to allow for strong features both in the high and low resolution setting.

Given our current architecture, it is easy to add **additional network heads**. This leads us to the last class of

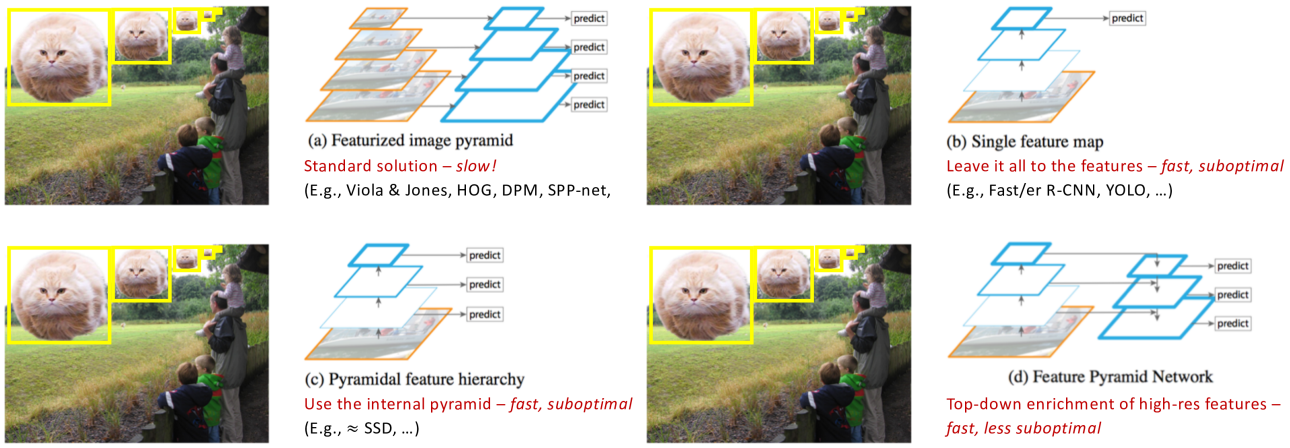


Figure 134: **Feature pyramids**

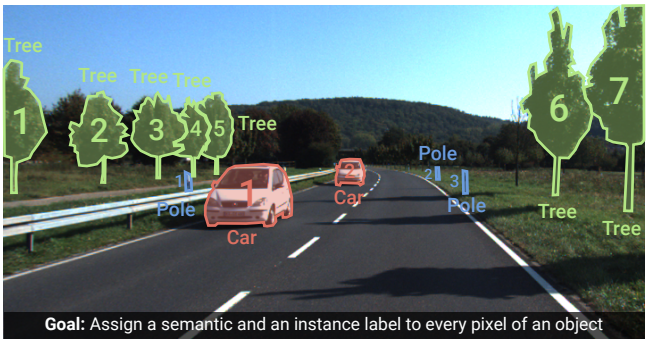


Figure 135: **Instance segmentation**

algorithms for this lecture.

10.4 Instance Segmentation

The goal of **instance segmentation** is to assign a semantic and an instance label to every pixel of an object.

Mask R-CNN is a Faster R-CNN adaptation which predicts binary instance segmentation (foreground vs background) for each detection (see Fig. 136). It introduces a slight modification to the RoI pooling component: instead of the standard max operation, it makes use of *bilinear interpolation*. Evaluation is performed by measuring Intersection-over-Union at mask level (rather than bounding box level).

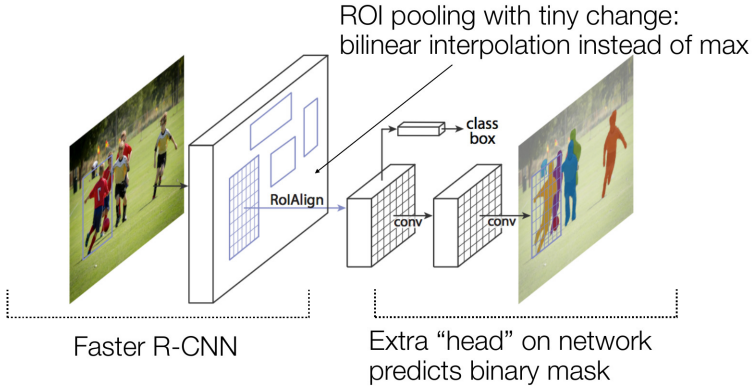


Figure 136: **Mask R-CNN**

DensePose is a Faster R-CNN adaptation for Dense Human Pose Estimation. The goal is to map all human pixels of an RGB image to the 3D surface of the human body. The main contribution was the DensePose-COCO

large-scale dataset with manually annotated image-to-surface correspondences.

Mesh R-CNN is a Faster R-CNN adaptation which predicts vertices and faces of a 3D mesh for each 2D bounding box.

PointRCNN is a Faster R-CNN adaptation for predicting 3D bounding boxes from point clouds.

11 Self-Supervised Learning

11.1 Preliminaries

11.1.1 Data Annotation

Most of the approaches that were covered so far in this course on Computer Vision were so called "supervised learning" techniques. This means that in order to estimate the parameters of these models, we need a very large labeled data set for training. Generally, the more parameters need to be estimated, the more data is needed. Labeling data sets is a sophisticated process, which requires a lot of human labor. As a result, obtaining these large annotated data sets is very tedious and time-consuming. Fig. 137 shows the amount of annotation time needed for some well-known image data sets in relation to their size. Of course the annotation time also varies a lot depending on the specific task that the data set is used for. The `ImageNet` data set is relatively large yet requires only little annotation time due to the fact that only per-image class labels need to be annotated. On the other, the `CamVid` and `CityScapes` data sets are relatively small but require a lot of annotation time, due to the fact that they are used for semantic segmentation, which requires per-pixel annotation of class labels. Fig. 137 therefore visualizes the trade-off between (1) annotating a large amount of data only coarsely and (2) annotating a small amount of data more precisely.

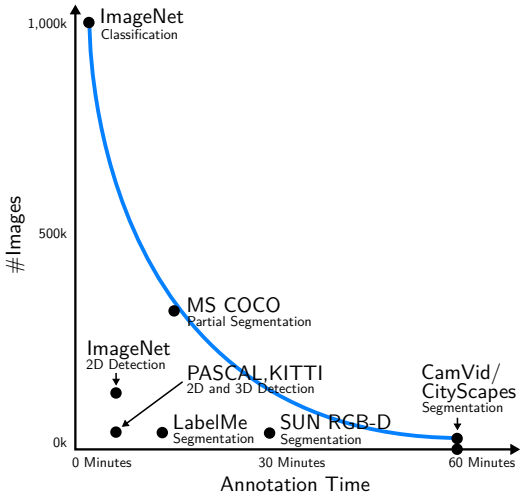


Figure 137: Overview of annotation time in relation to number of images for different data sets in computer vision

11.1.2 Image Classification

Image classification and more specifically `ImageNet` can be considered the "gold standard" image annotation problem. `ImageNet` has been annotated by humans in a peer-reviewed process to make the annotations consistent and noise-free. Apart form the cost of the annotation itself, the following human errors can occur during the process [1]:

Fine-grained recognition `ImageNet` contains images of over 120 species of dogs and an estimated 37% of human errors that occurred during the annotation of `ImageNet` fall into this category, as not all annotators are dog specialists.

Class unawareness Du to the large amount of possible classes, annotators can be unaware of the fact that a certain class even exists as ground truth. This type of error is estimated to constitute 24% of all errors.

Insufficient training data The annotator is only presented with 13 examples for every class which is insufficient for generalization. This type of error is estimated to constitute 5% of all errors.
Overall, the annotation of `ImageNet` took an estimated 22 human years of full-time work.

[1] https://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

11.1.3 Dense Semantic and Instance Annotation

Moving from image classification to semantic segmentation, the problem becomes even harder, because distant objects in an image only take up a few pixels and therefore are hard to recognize. Because for this task images have to be densely annotated, the process takes much more time. For the `CityScapes` data set the annotation of a single image took an estimated 60 - 90 minutes and required multiple annotators [9]

11.1.4 Stereo, Monodepth and Optical Flow

For problems like stereo reconstruction, monocular depth estimation and optical flow it is nearly impossible to obtain annotations manually through human labor. This is due to the fact that correspondences and depth are really difficult to annotate for humans. In case of the `KITTI` data set a light detection and ranging (LiDAR) laser scanner mounted on top of a vehicle was used to obtain ground truth for depth. Furthermore, manual object segmentation and tracking was used. However, for optical flow ground truth is hard to obtain. In case of the `KITTI` data set it was obtained by estimating the motion of the ego-vehicle and compensating for the shutter effect of the LiDAR scanner in order to convert the depth map into an optical flow map for the static parts of a scene. For dynamic objects in the scene (e.g. cars) CAD models were retrieved and fitted to multiple frames manually.

11.1.5 Self-Supervision

The idea of self-supervision is training a lot of the parameters needed for the task that we want to solve while avoiding the use of large annotated data sets. After this initial training, the network then only has to be fine-tuned on a smaller amount of labeled data in the end. This is achieved by obtaining labels from the raw unlabeled data itself, which can be thought of predicting parts of the data from other parts of the same data. For example, you can think of predicting an masked part of an image based on the rest of the unmasked image. This process removes the need for the costly acquisition of a labeled data set by supervising with the data itself, which is much less costly to obtain. The name of this process is called "pretext learning".
A classic example of pretext leaning is the Denoising Autoencoder (DAE) [58]. This kind of architecture is comprised of an encoder and a decoder which are both neural networks. In the example shown in Fig. 138 the

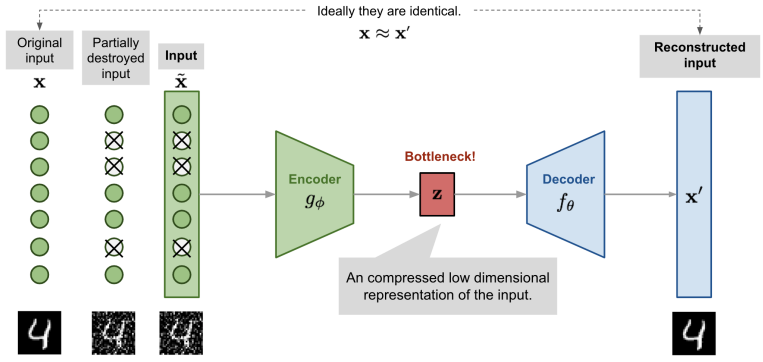


Figure 138: Illustration of the denoising autoencoder (DAE) architecture [58].

`MNIST` data set is used. Instead of using the original image as input to the DAE, a corrupted version of the original is used. The task of the DAE is then to predict the original image from the corrupted input. After training has taken place, only the encoder is kept and is used for prediction, by adding a classification head to the trained encoder. The model can then be fine-tuned for a classification task. Note that the initial training of the DAE was not yet for classification, but just for the auxiliary task of denoising a noisy input image.

11.1.6 Learning Problems

To put self-supervised learning into perspective, the following list contains some other well-known learning problems:

Reinforcement Learning The aim of reinforcement learning is to learn model parameters (e.g. of a neural network) by using active exploration from sparse rewards in the environment. Based on these rewards, adaptation of the model parameters takes place. Examples for this are deep q learning, gradient policy and actor critique. In terms of the amount of information that is given to the machine during learning, this learning problem requires only a few bits per sample, as the machine only predicts a scalar reward every once in a while.

Unsupervised Learning Here the goal is to learn the model parameters using a data set $\{x_i\}_{i=1}^N$ without any labels. Classical examples are clustering, dimensionality reduction (e.g. PCA) and generative models.

Supervised Learning The aim here is to learn the model parameters based on a data set $\{(x_i, y_i)\}_{i=1}^N$ of data-label pairs. Examples for this are all of the problems that have been covered in this course so far, including classification, regression and structured prediction. This learning problem requires approx. 10 to 10000 bits per sample to be given to the machine during learning, due to the fact that the machine predicts a category or a few numbers per input.

Self-Supervised Learning The goal here is to learn the model parameters using a data set $\{(x_i, x_i')\}_{i=1}^N$ of data-data pairs. And this is the topic of this section. This leaning problem requires millions of bits to be given to the machine during learning, as the machine has to predict any part of its input for any observed part (i.e. predicting future frames of a video).

11.2 Task-Specific Models

Task-specific models are self-supervised models that are specific to particular target tasks. This type of models is different from self-supervised models that are trying to learn generic representations independent of any downstream task.

11.2.1 Unsupervised Leaning of Depth and Ego-Motion

The first problem that we consider is unsupervised learning of depth and ego-motion (see Fig. 139) . The input at training time are monocular videos that can be downloaded from the internet or can be recorded by a camera mounted on top of a vehicle. The aim is then to train two neural networks (indicated in green and red in Fig. 139) The neural network highlighted in green (see Fig. 139, `Depth CNN`) is a CNN that takes a single

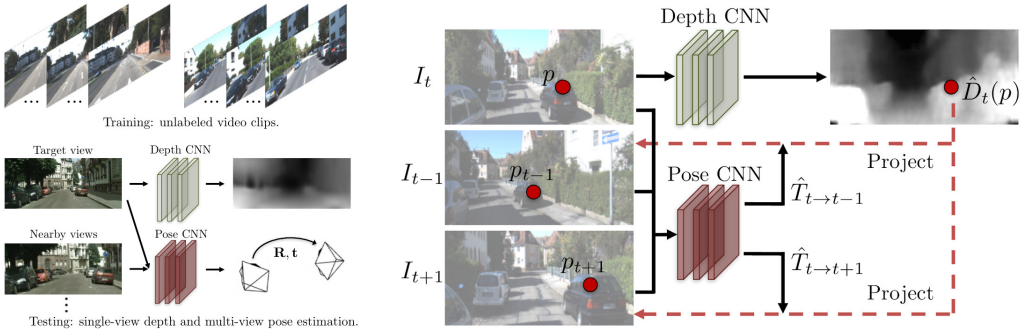


Figure 139: Illustration of unsupervised learning of depth and ego-motion [64].

image as input and produces a dense depth map as output. Note that unlike the settings we have considered in this course so far, we do not have access to the ground truth depth map for this problem. The neural network highlighted in red (see Fig. 139, `Pose CNN`) is a CNN that takes in two adjacent frames as input and predicts the relative camera motion, consisting of the rotation matrix $\mathbf{R} \in \mathbb{R}^3$ and the translation vector $\mathbf{t} \in \mathbb{R}^3$ (i.e. six d.o.f). The goal of the architecture is to jointly predict the depth and relative pose from two or three adjacent video frames.

The way this task is approached is heavily inspired by the way classical problems like optical flow are tackled. Also here, the photoconsistency assumption that has been used for optimizing the flow field is used to optimize the parameters of the `Depth CNN` as well as the `Pose CNN`. The model shown in Fig. 139 uses three adjacent frames as input: the "target view" at time t (indicated by I_t) as well as two "source views" at times $t-1$ and $t+1$ (indicated by I_{t-1} and I_{t+1}, respectively). The `Depth CNN` predicts a depth $\hat{D}_t(p)$ for every pixel location p in the target view. The `Pose CNN` predict the relative motion between (1) the target view I_t and the source view I_{t-1} (indicated by $\hat{T}_{t\to t-1}$) and (2) the target view I_t and the source view I_{t+1} (indicated by $\hat{T}_{t\to t+1}$). Note that each $\hat{T}$ here consists of $\mathbf{R}$ and $\mathbf{t}$. Because we now know every depth of every pixel location p in the target view, we can project it into 3D. Furthermore, because we know the relative pose between the all three views (or frames), we can project point p in target image I_t onto I_{t-1} and I_{t+1}, based on Eq.(1) (see Fig. 140, first arrow).

$$\tilde{\mathbf{p}}_s = \mathbf{K}((\mathbf{R}\,\mathbf{D}(\bar{\mathbf{p}}_t)\,\mathbf{K}^{-1}\,\bar{\mathbf{p}}_t) + \mathbf{t}) \tag{1}$$

In Eq.(1), the predicted depth $\mathbf{D}(\bar{\mathbf{p}}_t)$ and the relative pose $\mathbf{R}$ and $\mathbf{t}$ depend of the parameters of the `Depth CNN` and the `Pose CNN` as well as on the input frames. $\mathbf{K}$ is the intrinsic matrix of the camera.

Assuming the scene to be static, we therefore know the correspondence of pixel p between I_t and I_{t-1} as well as between between I_t and I_{t-1}, which is given implicitly through the depth map $\hat{D}_t$ and the relative pose $\hat{T}_{t\to t-1}$ and $\hat{T}_{t\to t+1}$. Given this information, we can effectively warp the two source views I_{t-1} and I_{t+1} into the target image I_t (see Fig. 140, second arrow). Due to the fact that after the projection based on Eq.(1), the estimated location might not match an actual, discrete pixel location, bilinear interpolation of RGB color

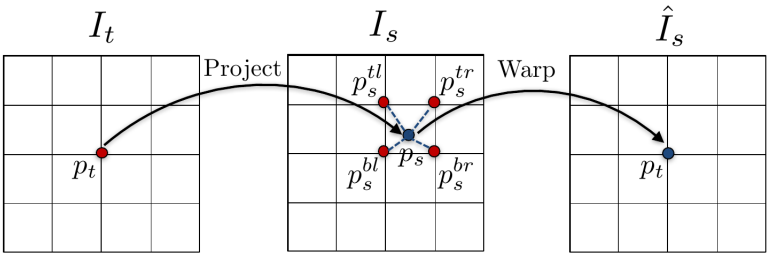


Figure 140: Illustration of projection and warping. Point p_t in target view I_t is projected into source view I_s based on Eq.(1). $\tilde{\mathbf{p}}_s$ represents the projected point in homogeneous coordinates. $\hat{I}_s$ represents the source view that has been warped to the target view, based on bilinear interpolation.

values of adjacent pixels is used during the warping process. After the warping, we can then compare the pixels of the original target view to the warped source views based on the photoconsistency loss, i.e. Eq.(2).

$$\mathcal{L} = \sum_p |I_t(p) - \hat{I}_s(p)| \tag{2}$$

Note that the potoconsistency loss is computed for all warped source images $\hat{I}_s$ and is then averaged across them. Assuming a static scene, Lambertian surfaces and photoconsistency, we would ideally expect the difference between the RGB pixel values to be zero, given that the predicted depth and the predicted relative pose is correct. However, if the predicted depth and the predicted pose are incorrect, then there will be a discrepancy between the RGB values of the compared pixels. This discrepancy - or photoinconsistency - can be used to back-propagate gradients to the parameters of the `Depth CNN` as well as the `Pose CNN`, in order to update these parameters. This is possible, because the projection as well as the bilinear interpolation are differentaible. All in all, we try to minimize the potoconsistency for a lot of monocular sequences of frames without requiring ground truth for depth or relative pose.
The particular architecture that is used for the model by Zhou et al. [64] is a U-Net (DispNet) architecture with a multi-scale prediction loss. This means that the depth as well as the pose are predicted at multiple scales and that the source views are warped at multiple scales. This improves gradient flow and thereby also optimization. The final objective includes the before mentioned photoconsistency loss, as well as a smoothness and an explainability loss. It is crucial to have a smoothness loss on the disparity map for the same reasons that we had to introduce smoothness constraints in methods that were covered earlier in the course, i.e. stereo and optical flow estimation.
Already back in 2017, the performance of the self-supervised method by Zhou et al. [64] was nearly on par with depth- or pose-supervised methods. Compared to depth-supervised methods, which use LiDAR scanners for ground truth that is not available for the whole image, the self-supervised model is able to produce results for all regions of a given image. Nevertheless, a downside of the model is that it assumes static scenes, so it can fail in the presence of dynamic objects.

11.2.2 Unsupervised Depth Estimation from Stereo

Another way of training a monocular depth estimation network in a self-supervised fashion is based on stereo (i.e. based on images that have been taken at the same time from adjacent viewpoints). The advantage is that one can get sharper results and that dynamic scenes are of no concern, because images have been taken simultaneously. In their paper from 2017, Godard, Aodha and Brostow [20] propose an idea which is similar to that presented in the last section (i.e. 11.2.1), just applied to the pair of stereo images instead of consecutive images. They proposed three different kinds of models as shown in Fig. 141. The "naïve" model (Fig. 141, left) takes the left input image (i.e. I^l) and runs it through a CNN that produces a disparity map (i.e. d^r). Because the left image is in the end compared to the right target image (i.e. I^r), it has to be first represented in the target image coordinate frame. This is due to the fact that the "sampler" (i.e. the bilinear interpolation) can only interpolate points in one direction. In other words, one can only warp an image into another image for which the disparity map is given. Therefore, the left image I^l has to be warped using the disparity map that is defined in the right image frame (i.e. d^r). However, our aim is not to predict the disparity map for the right target image d^r.
The "No LR" model (see Fig. 141, middle) tries to correct for this. Here, the left image I^l is run through the CNN to produce the left disparity map d^l. Using this disparity map the right image (i.e. I^r) is then warped into the coordinate frame of the left image (i.e. $I^r \rightarrow \tilde{I}^l$) so that it can be compared to the original left image I^l based on a photoconsistency loss. While this approach works better, it suffers from unwanted artefacts.

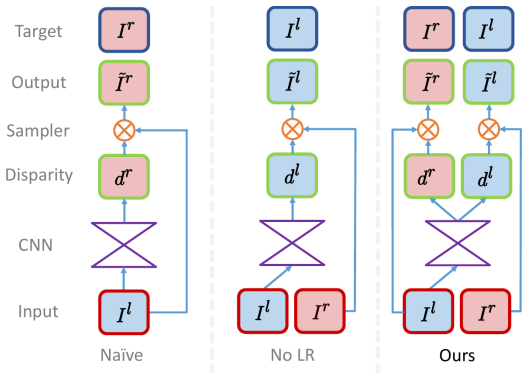


Figure 141: Illustration of unsupervised depth estimation from stereo [20].

The approach that worked best (see Fig. 141, right) is to run the left image (i.e. I^l) through a monocular depth estimation network that outputs not only the disparity map for the left image (i.e. d^l), but also for the right image (i.e. d^r). Based on both these disparity maps, we can then take the left image and warp it to the right image coordinate frame (i.e. $I^l \rightarrow \tilde{I}^r$) based on d^r. Similarly, we can take the right image and warp it to the left image coordinate frame (i.e. $I^r \rightarrow \tilde{I}^l$) based on d^l. The advantage of this is that we can now apply a consistency constraint between the two disparity maps d^l and d^r, which helps to eliminate outliers and improves training. Therefore, the losses that are used in this method are a photoconsistency loss, a disparity smoothness loss as well as a disparity left-right consistency loss.

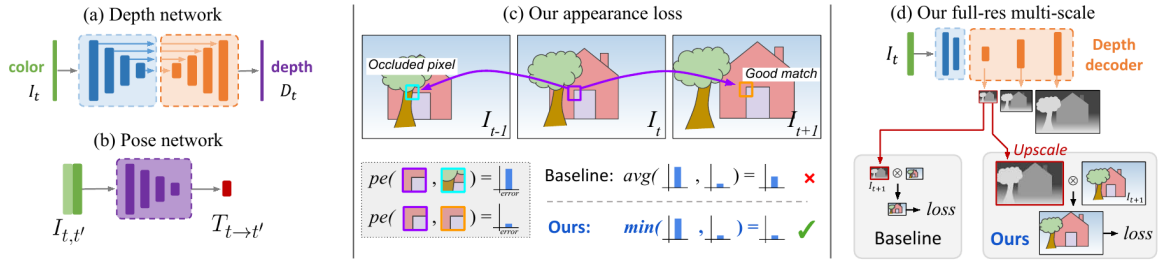


Figure 142: Illustration of improved unsupervised depth estimation from stereo [20].

In a followup work, the same authors more generally looked into self-supervised monocular depth estimation methods [19] and they introduced some improvements (see Fig. 142). For example, a per-pixel minimum photoconsistency loss was introduced to handle occlusions (see Fig. 142, middle). The baseline approach averages the photoconsistency loss across the comparisons of the target view to the warped source view(s) (see 11.2.1). This leads to a relatively high photoconsistency loss in the case that in one of the source views a pixel is occluded. In contrast, the method proposed by the authors takes the minimum photoconsistency loss over the comparisons. Because the model assumes three consecutive frames as input, the occluded pixel in one of the source views will likely not be occluded in the other source view. Therefore, the model effectively ignores the occlusion and still leads to a small overall photoconsistency loss. Another contributions of the authors is to compute all losses at the resolution of the input images (see Fig. 142, right). To this end the multi-scale predictions of the model are scaled up using bilinear interpolation before computing the losses. The authors have found that doing this can avoid texture-copy artefacts. Lastly, the authors introduced an auto-masking loss which helps to ignore input images in which the camera does not move, as for this kind of input the problem becomes ill-posed and cannot be solved, which harms the training of the network. By implementing the above mentioned improvements, the authors could produce significantly sharper depth boundaries and more details.

11.2.3 Unsupervised Learning of Optical Flow

The same ideas that have been outlined in the above sections can also be applied to the estimation of optical flow. In their paper form 2018, Meister, Hur and Roth [35] introduced so called bidirectional training (see Fig. 143). They used a `FlowNetC` (Dosovitskiy et al., 2015 [12]) and trained the same network (i.e. shared weights) once with input (I_1, I_2) and once with input (I_2, I_1) to predict optical flow in the forward direction and the backward direction, respectively. Just like the techniques explained in earlier sections, the authors also use a smoothness loss on the predicted optical flow maps. Given the estimated forward and backward flow, the input images can then be warped into the respective other image. Using a so called forward-backward consistency check, occluded areas can be retrieved which can then be masked when computing the data loss. Furthermore,

the occluded areas contribute to a consistency loss. The data/photo loss compares the images that have been warped based on estimated optical flow to the original input images.

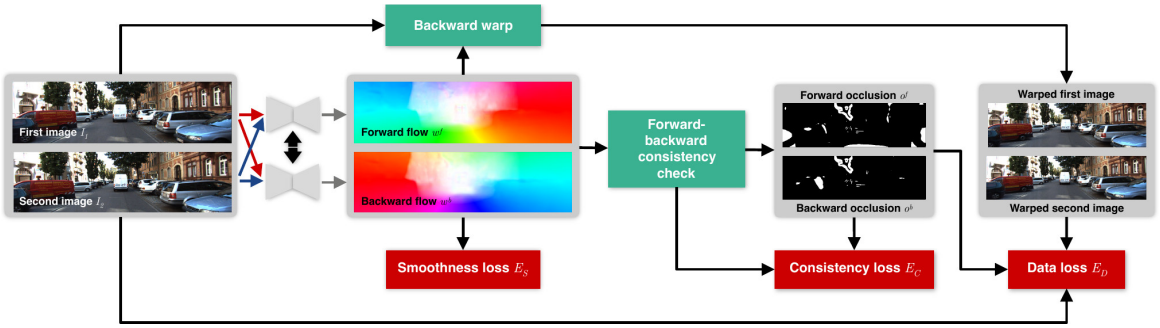


Figure 143: Illustration of unsupervised learning of optical flow [35].

11.2.4 Self-Supervised Monocular Scene Flow Estimation

The methods outlined in the previous sections can also be applied to monocular scene flow estimation, which combines monocular depth and optical flow estimation [26]. Here the task is to predict (1) the depth at each frame and (2) the 3D motion of each point in a frame, based on a monocular video sequence as input. The method proposed by Hur and Roth [26] is supervised (i.e trained) using stereo videos, however, at inference time only a monocular camera is used. Also here a smoothness loss and a photoconsistency loss are used.

11.3 Pretext Tasks

The last section (i.e. 11.2) has focused on self-supervised models which are tailored to a specific downstream task. This section covers the learning of more general neural representations using self-supervision. To achieve this, we have to define an auxiliary task - or pretext task. An example of this is predicting the rotation of an image which has been randomly rotated. By doing this, the hope is that we learn useful features which will be helpful for some other downstream task. This kind of pre-training can be preformed on lots of unlabeled data which in most cases is easily accessible via the internet. After the pre-training has taken place, we can then remove the last layers of the pre-trained model and instead add new layers to perform the target task (e.g. classification). We then train the whole network to perform the target task. Because pre-training has (hypothetically) already produced strong and useful features, we now need much less data to train the model to perform the target task.
The following sections introduce some pretext tasks that have been proposed in the literature. All of these might appear quite hand-engineered. This is due to the fact that there is not yet a theoretical basis for designing pretext tasks. As a result, the design of pretest tasks still heavily relies on empirical data which indicates that they work well.

11.3.1 Visual Representation Learning by Context Prediction

In this task the goal is, given two image patches, to predict their relative location in the image [11]. The locations of the patches per image are discretized into eight possible patch locations (see Fig. 144). This makes the task an eight-way classification task. The hope underlying this approach is that the model learns to recognize objects and their parts. By design, the task forces the model to learn something about the composition of objects.
Importantly, special care has to be taken in order to avoid that the network takes trivial short cuts to solve the task. For example, it has become clear in practice, that the network simply uses edge continuity. To prevent these short cuts, the authors proposed to leave gaps between the center patch and its eight neighbouring patches and to spatially jitter the locations of the eight neighboring patches. However, the authors realized that this does no fully solve the problem. They noticed that the network was able to predict the absolute location of randomly sampled patch in an image, from which the relative location of patches can be inferred easily. The authors found out, that the model was able to do so due to the so called abberation of camera lenses. Aberration describes the systematic shift of color channels with respect to the image location. Based on the information of this systematic color shift alone, the model was able to predict the absolute location of patches. A solution to this short cut is to randomly drop color channels or to project the whole image to grayscale.
With these measures to prevent trivial short cuts implemented, the model was actually able to learn useful features to perform downstream tasks such as object detection. The authors found that pre-training an `R-CNN` on this context prediction task without labels followed by fine-tuning the fully-connected layers for classification on a small data set led the resulting model to perform much better, compared to a model without pre-training.

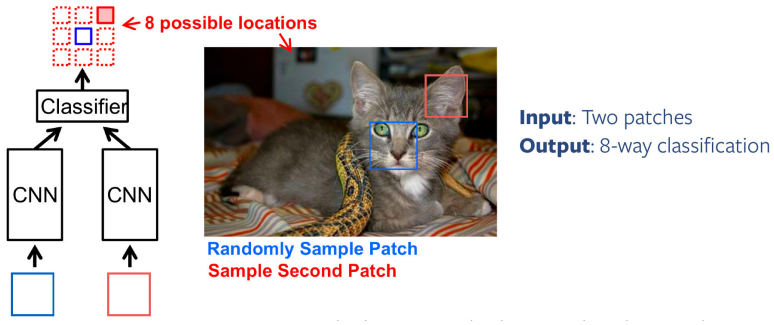


Figure 144: Illustration of visual representation learning by context prediction [11].

Surprisingly, the same approach has also been shown to work for surface-normal estimation, which is a completely different task.
The authors [11] also investigated the visual representations of what the network has learned. To this end, they used nearest neighbors search in the feature space of on of the last fully-connected layers of the architecture. The authors compared results from (1) random initialization of the network, (2) form `AlexNet` trained on `ImageNet` and (3) their pre-trained method. While random initialization (as expected) does not lead to meaningful results, the results of the pre-trained network are as semantically meaningful, as the results of the network trained on `ImageNet`.

11.3.2 Visual Representation Learning by Solving Jigsaw Puzzles

In a jigsaw puzzle pre-test task the idea is to randomly permute a version of the image that is separated into a 3×3 array of patches (see Fig. 145) and train a network to recover the configuration of patches that corresponds to the original image. Because theoretically, there are very many (i.e. 9!) possible configuration, the authors chose to restricted the number to 1000 possible permutations. This makes the problem a 1000-way classification task. In addition, these permutations were chosen based on the Hemming distance to increase the level of difficulty (i.e. the authors chose particularly hard permutation in order to make the model learn better representations).
Also for this jigsaw puzzle task, one has to make sure to prevent the model from learning trivial short cuts which may be useful for solving the pre-text task, but not for solving the downstream target task.

Low-level statistics The first such short cut is using low-level statistics. Adjacent patches in an image often share similar low-level statistics, i.e. mean and variance. A solution to avoid this kind of short cut is to normalize the patches based on their mean and variance.

Edge continuity This is the same short cut as mentioned in the previous section (i.e. 11.3.1).

Chromatic Aberration This is the same short cut as mentioned in the previous section (i.e. 11.3.1).

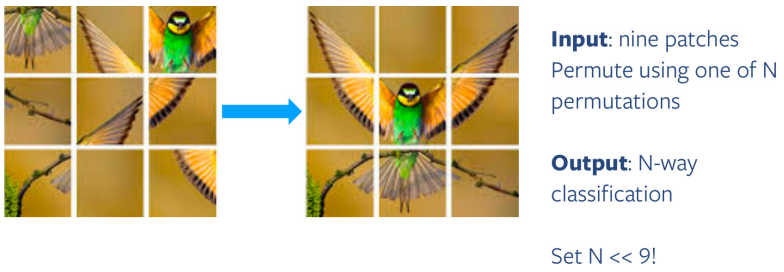


Figure 145: Illustration of visual representation learning by solving jigsaw puzzles [38].

By inspecting the visual representations that the model learns, the authors found that the model including these counter-measures for short cut learning indeed was able to learn meaningful representations throughout the layers of the architecture. The authors further compared their pre-trained model with fine-tuning for the `PASCAL VOC` challenge (i.e. classification, detection and segmentation) to other established methods such as fully-supervised pre-training on `ImageNet` and fine-tuning on `PASCAL VOC`, which was the standard approach at the time. They found that the performance of their model was nearly at par with the standard approach. This shows that self-supervised pre-training on the jigsaw puzzle task is a powerful method which produces results that can keep up quite well with standard, fully-supervised methods, at least for some tasks.

11.3.3 Feature Learning by Image Inpainting

Another pretext task is image inpainting, which is related to the idea of a DAE (see 11.1.5). In image inpainting, entire regions - or patches - of an image are masked out, which requires the model to learn some kind of semantic knowledge to recover the removed region from the context (i.e. the intact area of the image) [42].

11.3.4 Pretext Tasks - Summary

In summary, pretext tasks focus on "visual common sense", e.g. rearrangement, predicting rotation, inpaintin, colorization et cetera. To this end, the models are forced to learn useful features about natural images, e.g. semantic representation of an object category, in order to solve the pretext tasks. This is how the pretext tasks have been designed. We don't care about pretext task performance, but rather about the utility of the learned features for downstream target tasks, such as classification, detection, segmentation, depth estimation or normal prediction. As was already discussed above, the problem with this is that designing good pretext tasks is tedious and more of an "art", because there is not yet a theory behind it. Furthermore, it is usually not totally clear how the pretext task relates to the downstream task. Therefore, the features learned from the pretext task may not be general.

11.4 Contrastive Learning

Formally, the content of this section also covers pretext tasks, but as they have developed into their own field - so called contrastive learning - an individual section is dedicated to them.
As we have seen in the previous section (i.e. 11.3) the problem with pretext tasks is that for learning the majority of parameters during pre-training, we are considering a task that is completely decoupled from the downstream target task, i.e. solving jigsaw puzzles (pretext task) and image classification (target task). Therefore, we can only hope that the pretext task and the target task are somewhat aligned. In fact, it can be observed that the performance of of image classification heads that are attached to models that have been pre-trained on a jigsaw puzzle task saturates for deeper layers. This is somewhat counter-intuitive, because one would think that applying the classification heads to deeper layers in the pre-trained architecture should lead to better results, because deeper layers usually contain more semantically meaningful features compared to early layers. This indicates that the last layers of the pre-trained model have specialized in solving the pre-text task, i.e. the last layers are very specific to solving context prediction problems such as jigsaw puzzles.
The question is now, can we find a more general pretext task? To answer this question, we first need to specify what is desirable for a pretext task. Firstly, the pre-trained features should represent how images relate to each other. This means that images of object that closely relate to each other should also be close by in the feature space, while images that are very dissimilar should be further away in feature space. At the same time, learned features should be invariant to nuisance factors, such as the location of the object, lighting conditions or color. So the aim is to build a model, where for a particular reference image, different "views" of that reference image are close in feature space, while any view of any other object is further away. The term "views" here refers to augmentations generated from a reference image.

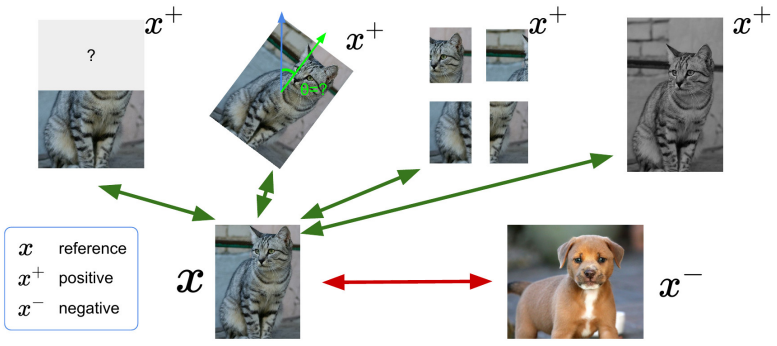


Figure 146: Illustration of different "views" of a reference image.

Given a chosen score function $s(\cdot, \cdot)$, we want to learn an encoder f that yields a high score for positive pairs (x, x^+) and a low score for negative pairs (x, x^-):

$$s(f(x), f(x^+)) \gg s(f(x), f(x^-)) \tag{3}$$

11.4.1 Contrastive Learning Objective

Assume we have one reference (x), one positive (x^+) and $N-1$ negative (x_j^-) examples. Consider the following multi-class cross entropy loss function:

$$\mathcal{L} = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+))) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right] \tag{4}$$

Note the use of the scoring function $s(\cdot, \cdot)$ in Eq. (4). This is commonly known as the `InfoNCE loss` and its negative (i.e. $-\mathcal{L}$) is a lower bound on the mutual information between $f(x)$ and $f(x^+)$ (see [57] for details):

$$MI[f(x), f(x^+)] \geq \log(N) - \mathcal{L} \tag{5}$$

Importantly, the larger the negative sample size (N), the tighter the bound, which is why we need a large N for this to work. The key idea behind this loss is to maximize the mutual information between features extracted from multiple "views", which forces the features to capture information about higher-level factors. In other words, by minimizing the loss, we maximize the mutual information between multiple views of the reference image.

11.4.2 Design Choices for Contrastive Learning

There are a number of design choices for these contrastive methods, that one can consider. The first design choices is concerned with the scoring function $s(\cdot, \cdot)$. The most common choice for the scoring function is the cosine similarity. i.e Eq. (6), which is the inner product between the features divided by their norm. As a side note, here we are looking at a "Siamese network" because the features are computed with the same network.

$$s(\mathbf{f}_1, \mathbf{f}_2) = \frac{\mathbf{f}_1^\top \mathbf{f}_2}{\|\mathbf{f}_1\| \, \|\mathbf{f}_2\|} \tag{6}$$

The second design choice is concerned with the choice of the positive and negative examples - or views. In `Contrastive Predictive Coding` the examples are chosen from the same image in a way that related (positive) examples/views are taken from nearby image regions, while unrelated (negative) examples/views are taken from further away image regions. A method that is more common nowadays is so called `Instance Discrimination`. Here all related examples come from a single image and all unrelated examples come from a different image. This method assumes that the image from which the related examples are taken is a simple image that depicts only a single object.
The third design choice is concerned with the augmentations that are used on the related examples. Possible augmentations are to crop, resize or flip images, to apply rotation or cutout, to drop or jitter colors, to apply Gaussian noise or blur or to apply other kinds of filters.

11.4.3 A Simple Framework for Contrastive Learning

`SimCLR` is currently one of the top-performing methods for contrastive leanring, which also uses the cosine similarity as score function. Fig. 147 shows an overview of `SimCLR`.

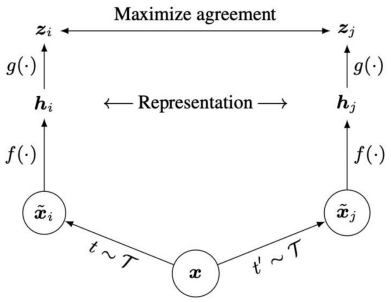


Figure 147: Illustration of `SimCLR` [7].

`SimCLR` takes the input reference image x and produces two different views $\tilde{x}_i$ and $\tilde{x}_j$ through randomly sampled augmentation operations t and t', respectively. It has been shown that having a large set of possible augmentations is crucial for this method to perform well. Then it runs the network $f(\cdot)$ (that we are interested in) to produce representations h_i and h_j. Additionally, `SimCLR` uses a projection network $g(\cdot)$ (which we are not interested in) to project features to a space (i.e. z_i and z_j) where contrastive learning is applied. Note that the

representation of z_i and z_j is used to maximize agreement based on the cosine similarity in the following way (cf. Eq.(6)):

$$s(\mathbf{z}_i, \mathbf{z}_j) = \frac{\mathbf{z}_i^\top \mathbf{z}_j}{\|\mathbf{z}_i\| \, \|\mathbf{z}_j\|}$$

However, the representation of h_i and h_j is what we are actually interested in. As such, $g(\cdot)$ and the representations of z_i and z_j only serve for better training of the model. Network $g(\cdot)$ will be discarded after training. Another advantage of using the projection network $g(\cdot)$ is that we can project into arbitrarily large dimensions, which provides useful flexibility.

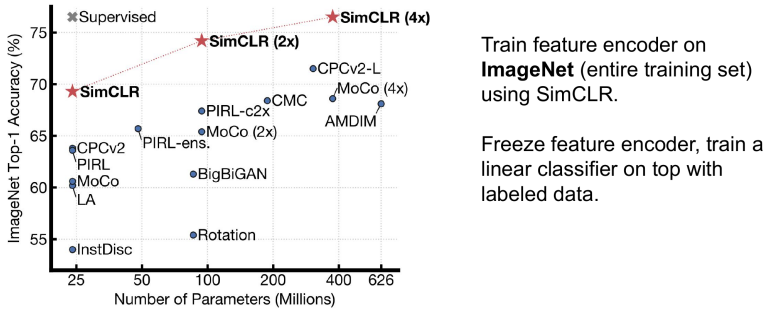


Figure 148: Performance of `SimCLR` compared to other self-supervised methods [7]

When looking at Top-1 accuracy on `ImageNet`, the `SimCLR` method performs surprisingly well compared to other self-supervised methods (see Fig. 148). When increasing the number of parameters (Fig. 148, x-axis) it is even possible to achieve results that are on par with supervised pre-trained models. When the feature encoder is pre-trained on the entire `ImageNet` data set without labels using `SimCLR` and subsequently fine-tuned only on 1% or 10% of the `ImageNet` data set with labels, the method even performs significantly better than a supervised model, that was trained on only 1% or 10% of the data set with labels.
A drawback of `SimCLR` is that it requires large batch sizes (between 2048 and 4096) to perform well. Due to the kind of architecture (i.e. Siamese network) this leads to a huge memory requirement. Therefore, it requires distributed training on TPUs (i.e. training on GPUs not possible).

11.4.4 Momentum Contrast

`Momentum Contrast` is a method that is similar to `SimCLR`, but includes major innovations that aim to alleviate the large memory requirements of `SimCLR` [23]. Fig. 149 shows an overview of this method. The crucial thing about `Momentum Contrast` is to keep a running queue - or dictionary - of keys for negative samples (i.e. a ring buffer of mini-batches). Gradients are then only back-propagated to the query encoder (i.e. Fig. 149 left, in green) and not to the queue of keys (i.e. Fig. 149 right).

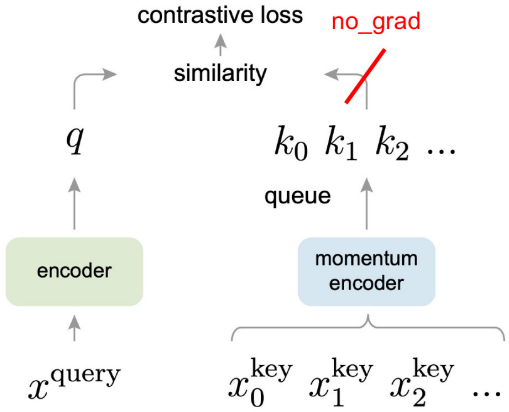


Figure 149: Illustration of `Momentum Contrast` [23].

Therefore, we do not have to store intermediate representations, but we only have to store the final result (i.e. the keys). For this reason, the dictionary can be much larger than the mini-batch size. However, if the same encoder that is used for the query would also be used for the keys, then this would lead to inconsistencies due to the fact that the query encoder changes during training (i.e. through backpropagation). To improve the

consistency of the keys, the authors proposed a momentum update rule:

$$\theta_k = \beta\,\theta_k + (1-\beta)\,\theta_q \tag{7}$$

The momentum encoder thus is a linear combination of the query encoder θ_q and the previous momentum encoder θ_k. This is a key improvement to make the method work also with smaller mini-batch sizes.
An improved version of this method - called `MoCo v2` - uses stronger data augmentation techniques as well as a non-linear projection head. These two additions turned out to be crucial as well. Using these two additions `MoCo v2` outperforms `SimCLR` while using much smaller mini-batch sizes (i.e. 256). Furthermore, `MoCo v2` can be trained on a regular $8 \times V100$ GPU node, instead of TPUs [7].

11.4.5 Barlow Twins

All of the methods that were discussed in this section so far were classical contrastive learning approaches, and there exist many more. A method called `Barlow Twins` that was proposed quite recently deviates somewhat from the classical paradigm and makes things even simpler [61]. It is based on information theory and tries to reduce the redundancy between neurons, instead of samples. In other words: instead of looking at distances between samples it looks at distances between neurons. The idea behind this is that neurons should be invariant to data augmentations, but independent of others.

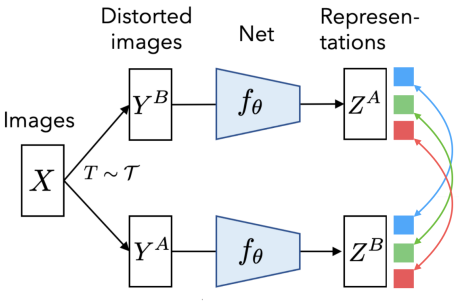


Figure 150: [61]

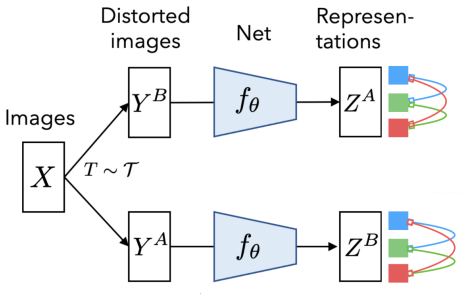


Figure 151: [61]

In the example given in Fig. 150 an image X is augmented into two different views Y^A and Y^B and then passed through a network $f(\cdot)$ that we want to train in order to produce feature representations Z^A and Z^B. For now we will imagine that both Z^A and Z^B are three-dimensional feature vectors, one feature per channel for the RGB-channels of the image. We now want to achieve that the features for each of the channels are similar between the two views (see Fig. 150, colored arrows) because the views are two different augmentations of the same object/image (i.e. X). However, at the same time we want the features of each of the representations Z to be different from each other (see Fig. 151). The idea to achieve this mathematically is to compute the empirical cross-correlation matrix (i.e. Eq.(8)) and to encourage it to become the identity matrix (see Fig. 152). So we need a loss $\mathcal{L}_{\mathcal{BT}}$ which enforces the off-diagonal elements of the empirical cross-correlation matrix to be zero (minimuze redundancy) and the diagonal elements to be one (i.e. neurons across augmentations should be correlated) (i.e. Eq.(9)).

$$\mathcal{C}_{ij} \triangleq \frac{\sum_b z^A_{b,i} z^B_{b,i}}{\sqrt{\sum_b (z^A_{b,i})^2}\sqrt{\sum_b (z^B_{b,j})^2}} \tag{8}$$

$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i (1-\mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2}_{\text{redundancy reduction term}} \tag{9}$$

The crucial difference to `SimCLR` is that no negative samples are needed, because the contrastive learning happens in the neuron space, rather than the sample space.
The results of the method are on par with state-of-the-art techniques such as `SimCLR`, even though it is much simpler to implement. Furthermore, it is only mildly affected by the mini-batch size. Similarly to `SimCLR`, the authors also found that projection into higher-dimensional space during pre-training leads to better results (see Fig. 153). By increasing the dimensionality of the projection MLP (i.e. Fig. 153) the performance of the method keeps on increasing.

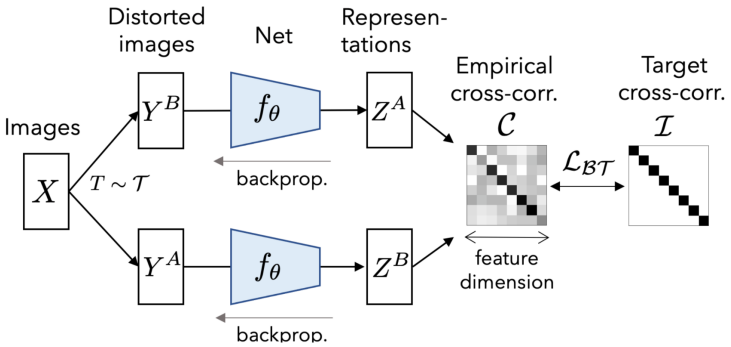


Figure 152: Illustration of the `Barlow Twins` [61]

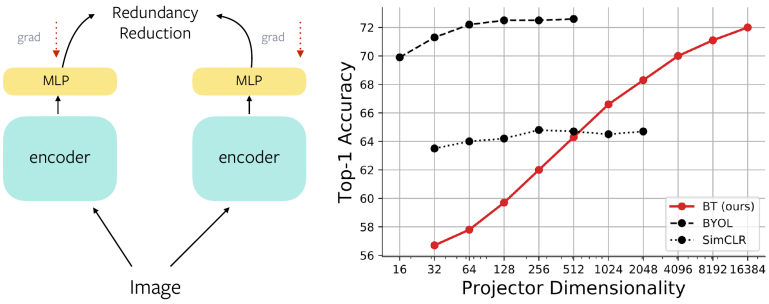


Figure 153: Performance of the `Barlow Twins` dependent on projector dimensionality [61].

12 Diverse Topics in Computer Vision

12.1 Input Optimization

Optimization with Neural Networks usually works by back-propagating gradients of the loss through the model. During weight optimization these gradients (with respect to the parameters) are used to minimize the loss by adjusting the weights.

However, the gradients can also be calculated with respect to the input itself. Consequently, the input of a Neural Network can be modified to cause a desired behavior in the fixed model. These techniques can be called **Input Optimization**. The lecture deals with to instances, namely Adversarial Attacks and Neural Style Transfer.

12.1.1 Adversarial Attacks

Adversarial Attacks are techniques to deliberately cause malfunctions in prediction model. More specific, Adversarial Examples are intentionally created instances to deceive the model. One way to create them is using Input Optimization.

One of the first examples was a technique called **L-BFGS-Attack** [50]. The idea is to take a fixed classifier $f : \mathbb{R} \to \{1, ..., L\}$ and an image x that way correctly classified by f. The adversarial example is generated by adding a small portion of noise to the image x, so that

$$x + \underset{\Delta x}{\operatorname{argmin}} \{\|\Delta x\|_2 : f(x + \Delta x) = y_t\} \tag{10}$$

where y_t denotes the target class. The new image is then classified to any y_t with a high confidence in the softmax distribution (see Fig. 154). Furthermore, the noise is quasi-imperceptible for a human observer. L-BFGS-Attack demonstrated that the classification of images (with CNNs) is remarkably fragile.
Additionally, it has been demonstrated that adversarial perturbations are also able to deceive semantic segmentation models [36].

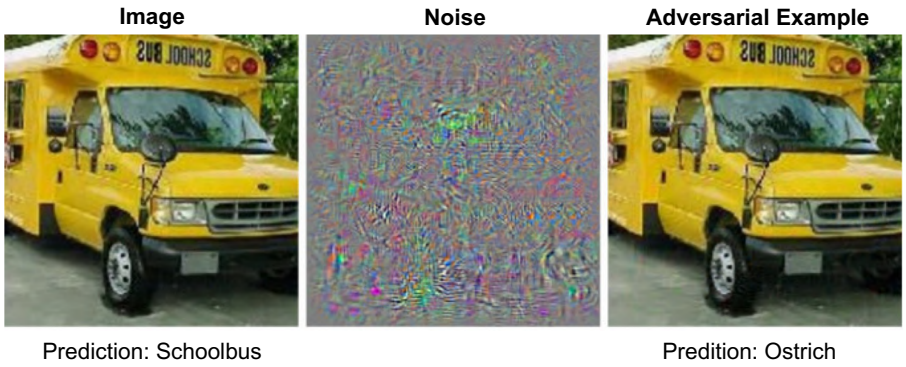


Figure 154: **L-BFGS-Attack**

Generally, these attacks are using direct access to the system and the exact input image. Consequently, the danger of these methods was not broadly recognized [33]. The attacks were underrated they only work models-specific and when the gradients information is available. Furthermore, the attack only works when concrete input pixels are manipulated. An attack in a real world environment does not have such constructed circumstances.

Robust Adversarial Attacks

However, newer research demonstrates that so called **Robust Adversarial Examples** exist [2]. It has been shown that classifiers in the physical world can be attacked, by constructing examples that not only work for static images. This is done by maximizing the **expectation over transformation** τ (EOT)

$$\underset{x'}{\operatorname{argmax}} \ \mathbb{E}_{t\sim\tau}[\log P(y_t|t(x')) - \lambda\|(t(x') - t(x))\|_2] \tag{11}$$

with a set of possible transformations t and for a target class y_t. Intuitively, the attack is constructed to cause malfunctions for different perspectives and viewing ankles of the adversarial example. However, it is notable that a larger distribution of transformations t require larger permutations which eventually can be recognised by humans.
Another possibility is the usage of *unsuspicious* robust attacks that mimic "graffiti" like structures that cause malfunctions in the classifier [13].

Adversarial Patch Attacks

Another popular technique are **Adversarial Patch Attacks** [5]. In practice they are easy to apply, because the attack consists of a patch that is added to a scene (see Fig. 155). These patches are optimized across many images to classify the scene as any target class. The attack is robust because the patches were optimized to be effective under a wide variety of transformations.

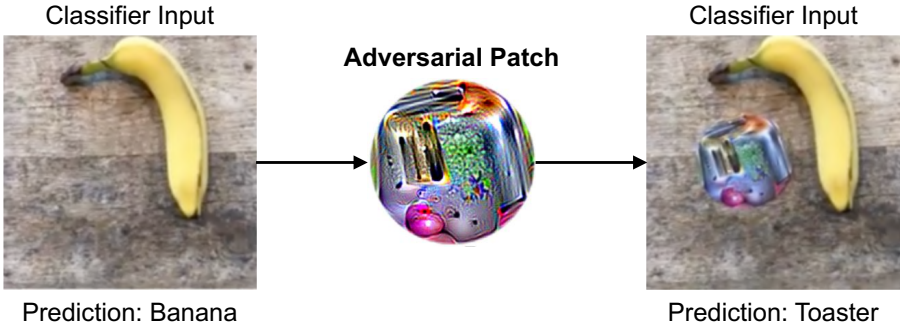


Figure 155: **Adversarial Patch Attacks**

It as been demonstrated that patch attacks also work for optical flow networks [44]. Let $F(I, I)$ be a optical flow network and $\mathcal{I}$ be the data set (I, I') of an image and the optical flow. Then $A(I, p, t, l)$ denotes the insertion of a patch p (transformed by t) into a the image I at location l. Furthermore, $\mathcal{T}$ is the distribution of affine 2D transformations of the patch and $\mathcal{L}$ denotes the uniform distribution where the patch is placed in the

image. The goal is to find a patch $\hat{p}$ so that

$$\begin{aligned} \hat{p} &= \underset{p}{\operatorname{argmin}} \ \mathbb{E}_{(I,I')\sim\mathcal{I},t\sim\mathcal{T},l\sim\mathcal{L}} \left[\frac{(u,v)\cdot(\tilde{u},\tilde{v})}{\|(u,v)\|_2 \cdot \|(\tilde{u},\tilde{v})\|_2} \right], \quad \text{with} \\ (u,v) &= F(I,I') \\ (\tilde{u},\tilde{v}) &= F(A(I,p,t,l),A(I',p,t,l)) \end{aligned} \tag{12}$$

Intuitively, the patch $\hat{p}$ is optimized to reverse the direction of the predicted optical flow.

When the patches are optimized for a specific network, it is called a **White-Box Attack** (see Fig. 156). Generally, larger patches cause a higher disturbance of the predicted optical flow. However, even small patches disturb the prediction far beyond the attacked area. Furthermore, a patch can be trained on a discrete set of networks (e.g. FlowNet2 and PWCNet) and is still able to disturb the prediction of unseen networks. This is called a **Black-Box Attack**.

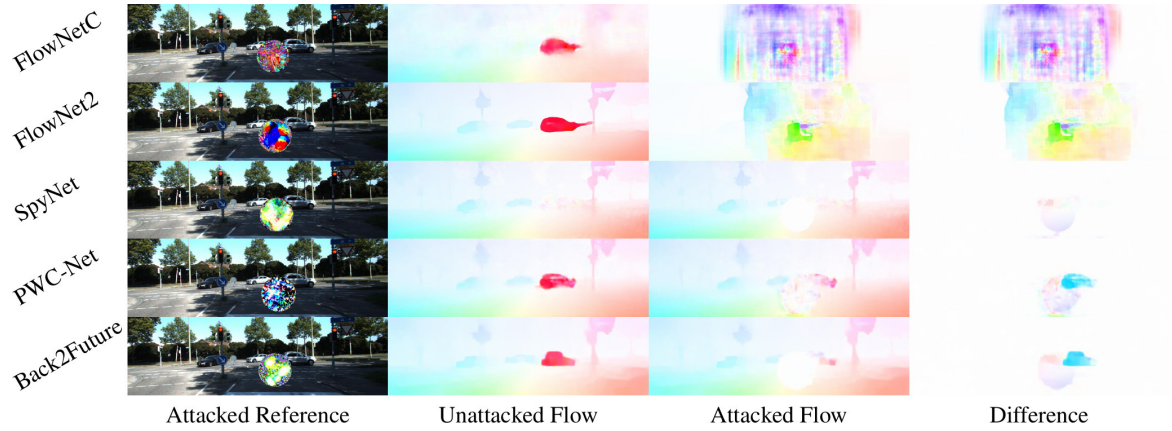


Figure 156: **Patch Attacks of Optical Flow Networks.** White box attacks for several networks with a large adversarial patch.

Defenses against Adversarial Attacks

Adversarial Attacks created a whole separate filed that deals to defend these attacks. A review from 2019 summarized these defenses into three distinct categories [60].

- **Gradient Masking/Obfuscation:** Hiding or masking the gradient of the model because most attack algorithms utilize the gradient information to exploit vulnerabilities of the classifier.
- **Robust Optimization:** A manner of learning, so that the model is more robust against adversarial attacks. By relearning and providing adversarial examples, the network thereby becomes more immune against constructed attacks.
- **Adversarial Example Detection:** Methods which first distinguish whether the input is benign or adversarial. Consequently, adversarial instances are rejected in the first place.

12.1.2 Neural Style Transfer

Input optimization can be used to find vulnerabilities in classifiers. However, the same technique is used to generate new data itself. **Neural Style transfer** utilized input optimization to synthesize new images [16]. The method receives a content image and a style image. The goal is to optimize a random image so that it reproduces the content of content image and transfers the characteristics of a style image (see Fig. 157). The loss of the network consists of a separate content and style loss $\mathcal{L}_{\text{content}} = \mathcal{L}_{\text{content}} + \mathcal{L}_{\text{style}}$. Neural Style Transfer uses a VGG network as feature extractor that is pre-trained on ImageNet.



Figure 157: **Neural Style transfer.** The stylized result combines a photograph from the *Neckarfront* in Tübingen with *The Starry Night* by Vincent van Gogh. It is beautiful.

Content Reconstruction

The content image is given to the fixed and pre-trained VGG Net. In the forward pass, the network will generate feature maps $\mathbf{P}_l$ at any layer l for the given content image. Separately, a random image $\vec{x}$ (with the same size) is given to the pre-trained VGG net, to generate feature maps $\mathbf{F}_l$. The content loss

$$\mathcal{L}_{\text{content}}(l) = \|\mathbf{F}_l - \mathbf{P}_l\|_2^2 \tag{13}$$

quantifies the difference of the extracted features in a given layer l. When the content loss is back-propagated, the random image is optimized so that the extracted feature maps are similar in the VGG net. Thereby, the content is reconstructed in the random image (see Fig. 158).

Style Reconstruction

The style loss is more difficult but follows a similar principle. The goal of style reconstruction is to minimize feature correlations of the generated image and the style image. The reconstruction utilizes the Gram matrices of $\mathbf{G}_l$ and $\mathbf{A}_l$, for the generated image and style image, respectively. The Gram matrices represent the pairwise correlation of features channels at the given layer. The complete style loss is given by

$$\mathcal{L}_{\text{content}}(l) = \|\mathbf{G}_l - \mathbf{A}_l\|_2^2 \qquad G_{ij}^l = \sum_{\mathbf{p}\in\Omega} F_i^l(\mathbf{p}) F_j^l(\mathbf{p}) \tag{14}$$

The style loss is combined over several layers of the network. By using the Gram matrices, the content of the style image becomes secondary and almost unrecognizable. However the artistic characteristics are preserved.

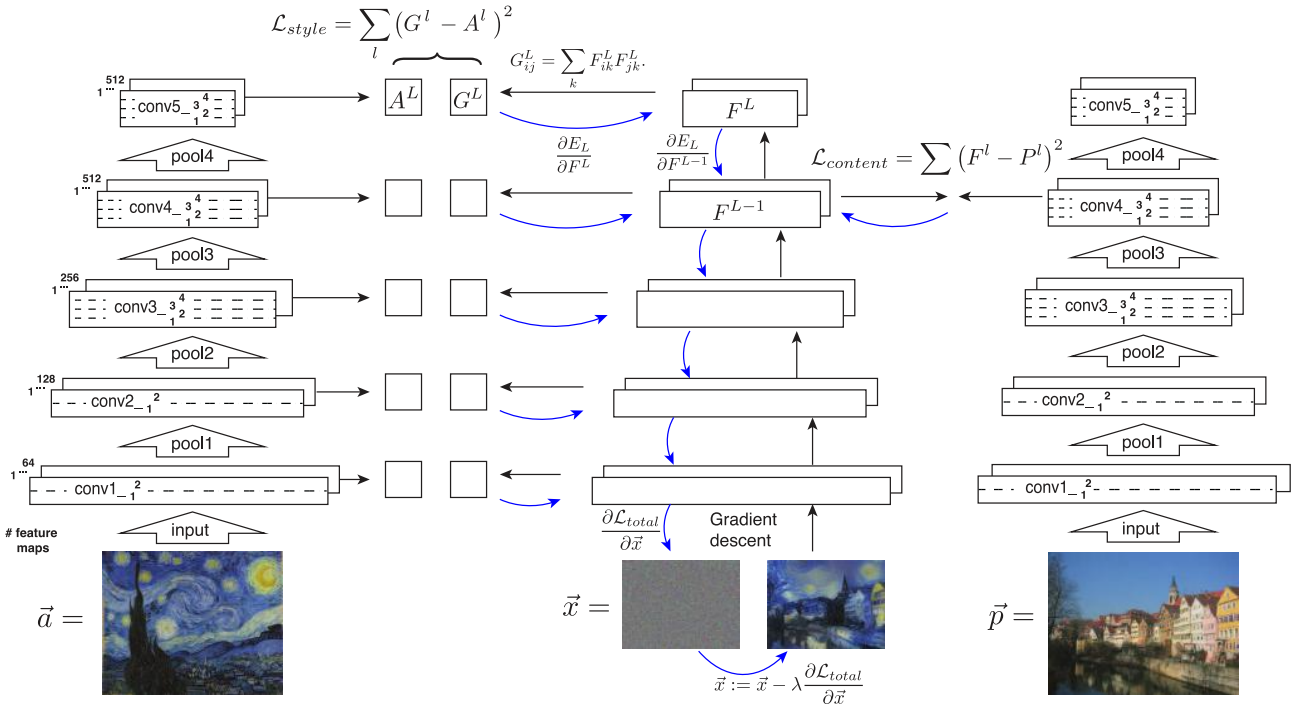


Figure 158: **Complete Neural Style transfer.**

In total, Neural Style transfer uses the gradients of the content loss $\mathcal{L}_{\text{content}}$ and the style loss $\mathcal{L}_{\text{style}}$ to optimize the white noise image $\vec{x}$. The optimized image combines the content and style of the given images and yields truly beautiful results.

12.2 Compositional Models

12.2.1 Shape Abstraction

There are several traditional methods to encode 3D objects and scenes. These include voxels, point clouds, meshes, and implicit representations. However, traditional methods have limitations. They encode a large number of unstructured geometric elements. There is no semantic information about the scene, e.g. the individual parts or the composition of an object.

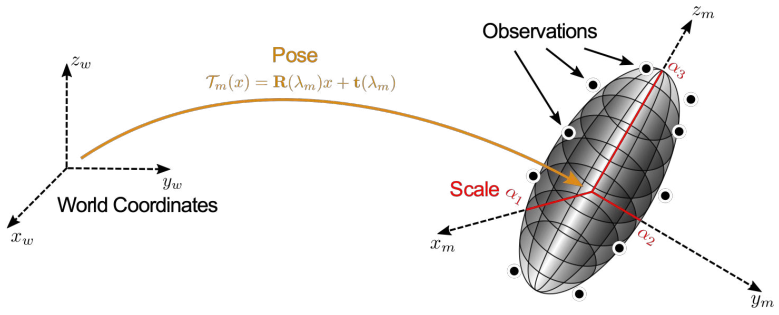


Figure 159: **The Superquadratic**

Humans generally perceive an object as a composition of several atomic parts. Because of that, **Primitive-based 3D Representations** are a quite natural for the human perception. A 3D object is encoded using a small number simple geometric shapes, called **primitives**. This representation has benefit's by construction. The primitives express semantically meaningful parts of an object and additionally offer memory-efficient storage.

Nevertheless, learning the right abstraction is challenging. First, there are no annotated large-scale data sets that allow supervision for primitive-based methods. Therefore, images or point clouds must be used as an input, which means the primitive representation must be learned unsupervised. Second, objects can have a variable number of primitives. Consequently, a network has to estimate the number (or the probability of existence) of primitives during reconstruction.

In 1986, researchers proposed to use multiple superquaratics to present a 3D scenes [43]. A superquadratic a parameterized geometrical surface that can describe a large variety of structures (e.g. cubes, cylinders, spheres)(see Fig. 159). Furthermore, a superquadratic can be efficiently describes by only 11 parameters: 6 for the pose, 3 for the size, and 2 for the shape. However, early approaches failed to robustly fit the parameters to the input shapes.

Recent work revisited the primitive representation via super quadratics [40]. The continuous parameters of a super quadratic can easily be estimated with Neural Networks. Additionally, the fitting of the parameters can be regularized by utilizing large data sets for (unsupervised) learning.
The network architecture is fairly straightforward (see Fig. 160). A encoder is used to extract high level features from an image or point cloud. These features are used to estimate the parameters for a fixed number of super quadratics. Additionally, the network has to estimate a probability of existence for each primitive.

The Network can't be optimized by using supervised primitives. Nevertheless, a loss function can still be constructed by measuring the difference between the primitive representation and the true object:

$$\mathcal{L}(\mathbf{P}, \mathbf{X}) = \mathcal{L}_{P\to X}(\mathbf{P}, \mathbf{X}) + \mathcal{L}_{X\to P}(\mathbf{X}, \mathbf{P}) + \mathcal{L}_{\gamma}(\mathbf{P}) \tag{15}$$

- $\mathcal{L}_{P\to X}(\mathbf{P}, \mathbf{X})$: Difference between a primitive $\mathbf{P}$ and a point cloud $\mathbf{X}$. Enforces precision.
- $\mathcal{L}_{X\to P}(\mathbf{X}, \mathbf{P})$: Difference between a point cloud $\mathbf{X}$ and a primitive $\mathbf{P}$. Enforces coverage.
- $\mathcal{L}_{\gamma}(\mathbf{P})$: Ensures existence of at least one primitive and enforces parsimony primitive representation.

In practice, the loss is evaluated by sampling points on the surface of each super quadratic. These points are used to approximate the expected loss.

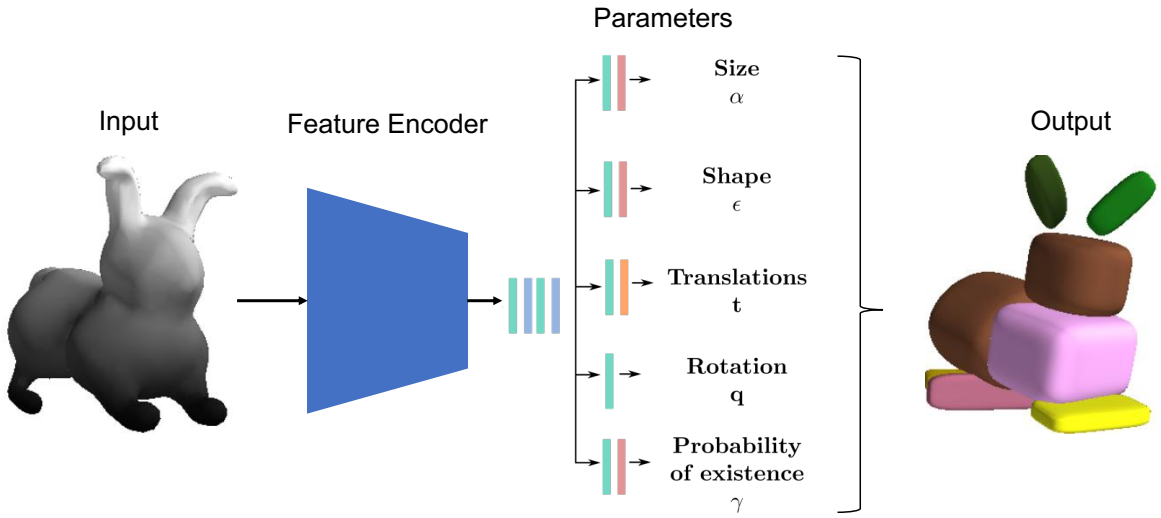


Figure 160: **The Network Architecture for primitive-based 3D reconstruction**

Nevertheless, a super quadratic is often too limited to model detailed primitives. Newer research propose Invertible Neural Networks (INNs), which bijectively map a sphere and to a target object [39]. As a result, the INN can efficiently learn and predict more complex shapes which result in semantically more meaningfully parts.

12.2.2 Causal Reasoning

Neural Networks can learn complex features from images. Nevertheless, they are also prone to learn spurious correlation and to take shortcuts. For example, a classifier for cows and camels can suffer from biases in the data set. Images of camels are expected to have a desert background whereas images of cows are mostly captured on green pastures. Consequently, such a classifier is likely to take short cuts and classify according to the vegetation of the background.
The goal of **Counterfactual Generative Networks** (CGN) is to decompose an image and to generate new *counterfactual examples*, which have new combinations of texture, shape, and background [47]. The generative model is structured into several sub-networks, which utilize pre-trained GANs and salient object detection networks (BigGAN, U2-Net). The shape **m**, foreground **f**, and background **b** are separately extracted and recomposed for a final image (see Fig. 161).

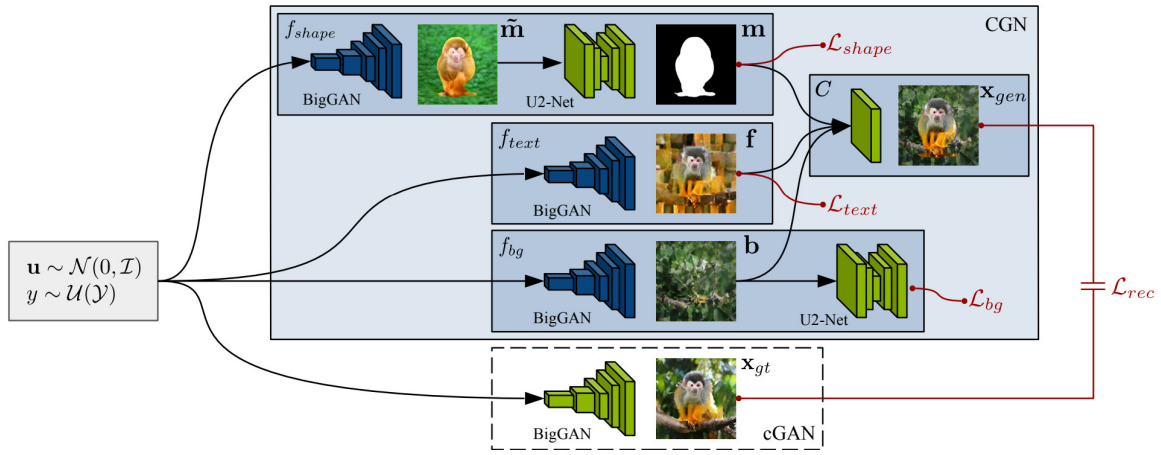


Figure 161: **Counterfactual Generative Networks.**

Counterfactual examples can be beneficial when testing the robustness of a classifier. Retraining with counterfactual examples can increase the performance on out-of-distribution data.

12.2.3 Holistic 3D Scene Understanding

In holistic 3D scene understanding, the goal is to infer the composition of a 3D scene. Examples are Indoor Scene Understanding, where the goal is to predict the layout and the objects of indoor scene from a single

RGB-D image of with a Kinect camera [18]. Another example is Urban scene understanding, where the goal is to detect multiple objects in a outdoor traffic scene and to predict their layout [17].

12.3 Human Body Models

Humans interacted with their environment though their bodies. Activities of humans are mainly defined by particular movements and gestures. Modelling a human body is interesting for several disciplines, e.g. the video game or movie industry.
The goal is to model a realistic looking human, which deforms and moves like a real person. Ideally, the model can be fitted with real world data and be represented by small number of parameters. This is crucial to efficiently store the model and to reduce dimensionality.
This subsection mainly deals about **SMPL**, which is a modelling technique that was trained on thousands of 3D body scans to create realistic human models [31].

Human body modelling has a long history in Computer Vision. However, the task is difficult due to variety of shape, physique and proportions of human bodies. SMPL is inspired by a morphable 3D face modelling method from 1999 [3]. In this method, a data set of 200 colored 3D images of faces is utilized to derive a single morphable reference face. As a result, a face can be represented by a vector space in relation to the reference face. Furthermore, the continuous parameters can be modified to change to perceived gender, fullness, or expression of the face.

SMPL: A Skinned Multi-Person Linear Model

Similarly, SMPL utilizes thousands of human 3D body scans. The output representation of a human body is a 3D mesh M:

$$M(\theta, \beta, \delta, A; P) \tag{16}$$

where θ, β, and δ are parameters for the pose, shape and dynamics of the mesh, respectively (see Fig. 162). A is the texture and P parameters of the body mesh. A reference mesh was designed by an artist to match a wide variety of human bodies and consist of 6890 vertices and 23 joint.

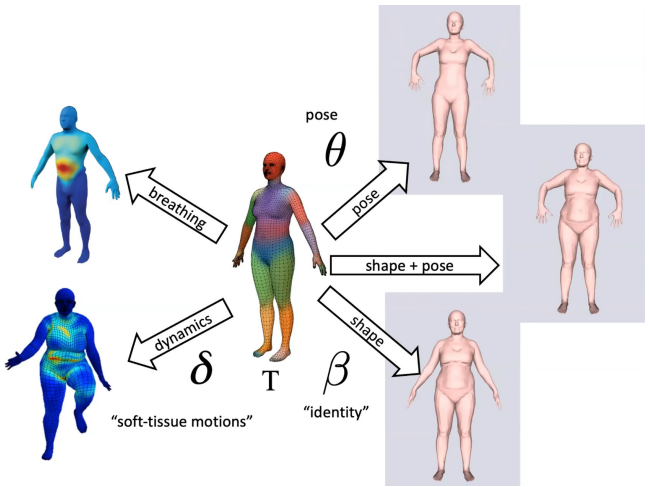


Figure 162: **The 3D Mesh of SMPL** with its parameters

A hard problem is the data pre-processing and mesh registration in the first place. In total, 1786 raw 3D scans were available as unordered 3D points with various artefacts. The goal is to align the reference mesh with each scan, to get the correspondences to the base mesh. However, a true body model would be needed to know how to deform the reference model towards the new body. This result in a *chicken-and-egg problem*. SMPL solves this by predicting the human model and performing the registration jointly.

In the following, it is a assumed that the true meshes are available. Subsequently, the scan can be converted into a standard pose of each person. This pose is called the *A-pose* due to the similarity with the shape of the letter A. Consequently, all the mesh vertices can be vectorized into the structure T^i where the mean of the meshes are subtracted. These vectors can be mapped into a large matrix $\mathbf{T}$ (with the columns T^i). This is done to performing Principal Component Analysis (PCA):

$$\mathbf{T} = \mathbf{S}\boldsymbol{\beta} + \boldsymbol{\mu} \tag{17}$$

where PCA yields a linear model which transfers the meshes into a low dimensional subspace. As a result, the 2100 dimensional space is captured with 10-300 dimensions, while maintaining most variance of the data.

As discussed earlier, SMPL also has to solve the mesh registration. A traditional method to obtain how the base vertices are related to an structure is **Linear Blend Skinning** (LBS). In LBS, the goal is to find a transformed vertex $\mathbf{t}'_i$ (in a target pose) each point $\mathbf{t}_i$:

$$\mathbf{t}'_i = \sum_k w_{ki} \mathbf{G}_k(\theta, \mathbf{J}) \mathbf{t}_i$$

where w_{ki} are the blend skinning weights, $\mathbf{G}_k$ is the rigid bone transformation, θ the pose parameters, and $\mathbf{J}$ the joint locations. Intuitively, $\mathbf{G}_k$ determines how the points on a bone k are transformed according to new location of k (see Fig. 163). In practice, the skinning weights w_{ki} are manually assigned by artists to ensure realistic models.

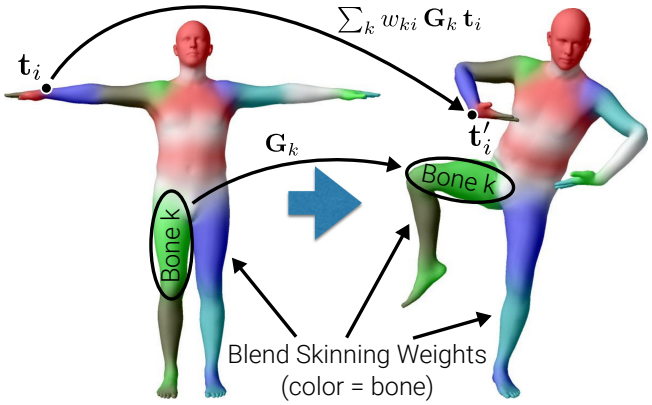


Figure 163: **Illustration of Linear Blend Skinning.**

Unfortunately LBS is often too simple and produces unrealistic deformations at joints. Because of that, SMPL uses an extension of LBS:

$$\mathbf{t}'_i = \sum_k w_{ki} \mathbf{G}_k(\theta, \mathbf{J}(\boldsymbol{\beta}))(\mathbf{t}_i + \mathbf{s}_i(\boldsymbol{\beta}) + \mathbf{p}_i(\boldsymbol{\beta}))$$

where the joints $\mathbf{J}(\boldsymbol{\beta})$ depend on the shape of the body $\boldsymbol{\beta}$, and a shape corrective $\mathbf{s}(\boldsymbol{\beta}) = \mathbf{S}\boldsymbol{\beta}$ is added. This allows to take the shape of a body into account when deforming the model, e.g. if a person is heavy or thin. Furthermore, SMPL adds a pose corrective $\mathbf{p}(\boldsymbol{\theta}) = \mathbf{P}\boldsymbol{\theta}$, to accommodate the pose $\boldsymbol{\theta}$ when deforming the mesh.

SMPL formed the basis for various follow-up work. An example is SoftSMPL, which focuses on modelling realistic soft-tissue dynamics for body shape and motion [46]. Another variant is SMPLify where a human body model is estimated from a single image [4]. Current research focuses to create more complex human body models, which for example include clothing, appearance or interaction of a person.

12.4 Deepfakes

Deepfakes (combination of *Deep Learning* and *fake*) are a form of synthesized media that were created using Deep Learning techniques. The name originates from a GitHub repository[2] that focuses on swapping face in photos and videos. Deepfakes can also be used create other forms of media, e.g. the audio of a persons voice [37].

Face Swap and Reenactment

The original **Face Swap** method uses a auto-encoder model which was trained on the footage and faces of the two people. A shared encoder is trained on the original footage of both persons, to extract latent features (see Fig. 164). As a result, the encoder is forced to learn the same latent space for both faces. The encoder learns the similarity between two sets of face images and is more robust for pose, expression, lighting etc.

[2] https://github.com/deepfakes/faceswap

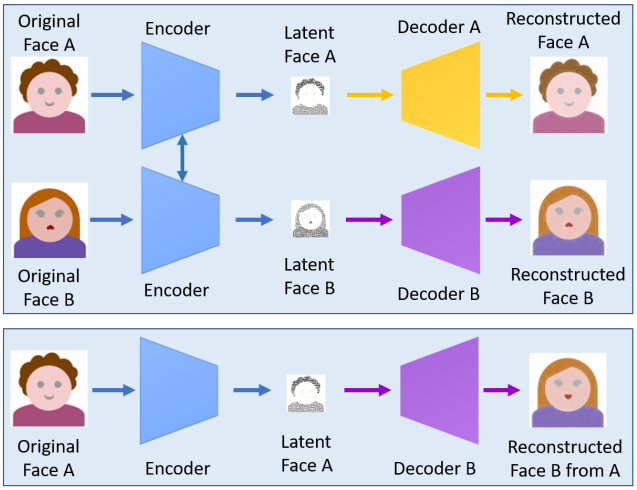


Figure 164: **Architecture of Face Swap.** (Illustration form [37])

A separate decoder is trained for each face individually to reconstruct the footage from the latent space. At inference time, these decoders are swapped, so that the face of a person is reconstructed with the appearance of another face.



Figure 165: **Face Swap vs. Facial Reenactment**

In contrast, **Facial Reenactment** is a technique which takes an expression from a video or image and animates it on another person (see Fig. 165). The output is a manipulation of the expression of a person. Reenactment techniques yield large potential in the animation and movie industry. Classical techniques require tedious 3D model creation and manual editing. Machine Learning could automate this process and enable richer editing, or updating scenes without re-shooting them.

Another example of facial reenactment is **Face2Face** [52] (see Fig. 166). The goal is capture the facial expression from a source RGB-input and map it on a person in a target video. All of this is done in real time with a commodity webcam for the source input. The method uses a face model to track the expression of the faces in the target and source video. The parameters of this face model are recovered by dense non-rigid alignment. The expressions in the target and source video are tracked which enables the transfer of the facial expressions.

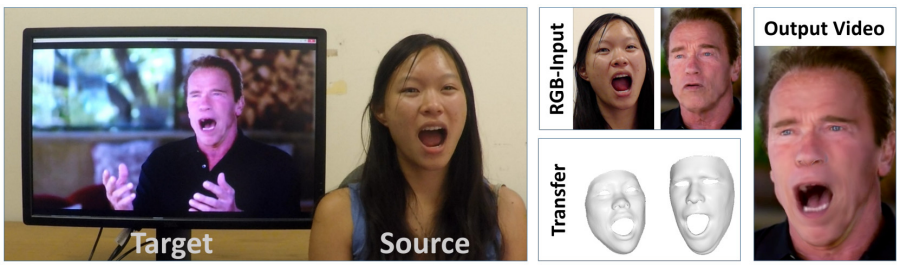


Figure 166: **An illustration of Face2Face**

Deep Fake Detection

Deep fake methods can be misused to spread disinformation and cause harm. Methods to detect deep fakes have been proposed which train a classifier to distinguish between authentic and tampered footage [45]. Convolutional Neural Networks (CNNs) can easily be utilized for binary classification such as deep fake detection (see Fig. 167). CNNs are powerful classifiers when trained on large diverse data sets. The **FaceForensics** data set offers a publicly available collection of 1.8 million images from 1000 manipulated videos. The data set contains footage from with various techniques (e.g. Face2Face, FaceSwap) of over 1000 research groups.

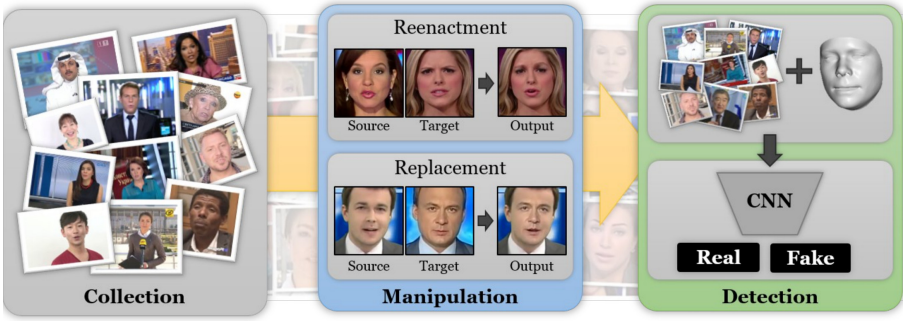


Figure 167: **FaceForensics.** Deep Fake Detection with Convolutional Neural Networks

Deep fake detection reliably able detect manipulated instances on in-domain data. The model learns the features and artefacts of the explicit forgery method and is able to detect them in a test scenario. However, it has been shown that deep fake detection yields poor performance on out-of-distribution data [10]. In other words, the detection fails to generalize and only work for deepfake methods it was trained with. Therefore, the detection requires more advanced methods like self-supervised, transfer and few-shot learning for better generalization.

References

[1] Jens Ackermann, Fabian Langguth, Simon Fuhrmann, and Michael Goesele. Photometric stereo for outdoor webcams. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[2] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *Proc. of the International Conf. on Machine learning (ICML)*, 2018.

[3] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *ACM Trans. on Graphics*, 1999.

[4] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter V. Gehler, Javier Romero, and Michael J. Black. Keep it SMPL: automatic estimation of 3d human pose and shape from a single image. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2016.

[5] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv.org*, 1712.09665, 2017.

[6] Liang-Chieh Chen, Alexander G. Schwing, Alan L. Yuille, and Raquel Urtasun. Learning deep structured models. In *Proc. of the International Conf. on Machine learning (ICML)*, pages 1785–1794, 2015.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *Proc. of the International Conf. on Machine learning (ICML)*, 2020.

[8] Stéphane Christy and Radu Horaud. Euclidean shape and motion from multiple perspective views by affine iterations. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 18(11):1098–1104, 1996.

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] Davide Cozzolino, Justus Thies, Andreas Rössler, Christian Riess, Matthias Nießner, and Luisa Verdoliva. Forensictransfer: Weakly-supervised domain adaptation for forgery detection. *arXiv.org*, abs/1812.02510, 2018.

[11] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2015.

[12] A. Dosovitskiy, P. Fischer, E. Ilg, P. Haeusser, C. Hazirbas, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2015.

[13] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models. *arXiv.org*, 1707.08945, 2017.

[14] Vittorio Ferrari, Manuel J. Marín-Jiménez, and Andrew Zisserman. Progressive search space reduction for human pose estimation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[15] Robert T. Frankot and Rama Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 10(4):439–451, 1988.

[16] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] Andreas Geiger, Martin Lauer, Christian Wojek, Christoph Stiller, and Raquel Urtasun. 3D traffic scene understanding from movable platforms. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 36(5):1012–1025, 2014.

[18] Andreas Geiger and Chaohui Wang. Joint 3d object and layout inference from a single rgb-d image. In *Proc. of the German Conference on Pattern Recognition (GCPR)*, 2015.

[19] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth estimation. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019.

[20] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[21] Fatma Güney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[22] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[23] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence (AI)*, 17(1-3):185–203, 1981.

[25] Jinggang Huang, Ann B. Lee, and David Mumford. Statistics of range images. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2000.

[26] Junhwa Hur and Stefan Roth. Self-supervised monocular scene flow estimation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[27] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[28] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, and Peter Henry. End-to-end learning of geometry and context for deep stereo regression. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2017.

[29] Katrin Lasinger, René Ranftl, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *arXiv.org*, 1907.01341, 2019.

[30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[31] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. on Graphics*, 2015.

[32] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 1999.

[33] Jiajun Lu, Hussein Sibai, Evan Fabry, and David A. Forsyth. NO need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv.org*, 1707.03501, 2017.

[34] N. Mayer, E. Ilg, P. Haeusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[35] Simon Meister, Junhwa Hur, and Stefan Roth. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. In *Proc. of the Conf. on Artificial Intelligence (AAAI)*, 2018.

[36] Jan Hendrik Metzen, Mummadi Chaithanya Kumar, Thomas Brox, and Volker Fischer. Universal adversarial perturbations against semantic image segmentation. *arXiv.org*, 1704.05712, 2017.

[37] Thanh Thi Nguyen, Quoc Viet Hung Nguyen, Cuong M. Nguyen, Dung Nguyen, Duc Thanh Nguyen, and Saeid Nahavandi. Deep learning for deepfakes creation and detection: A survey. *arXiv.org*, 1909.11573, 2019.

[38] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Proc. of the European Conf. on Computer Vision (ECCV)*, 2016.

[39] Despoina Paschalidou, Angelos Katharopoulos, Andreas Geiger, and Sanja Fidler. Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. *arXiv.org*, 2103.10429, 2021.

[40] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[41] Despoina Paschalidou, Ali Osman Ulusoy, Carolin Schmitt, Luc van Gool, and Andreas Geiger. Raynet: Learning volumetric 3d reconstruction with ray potentials. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[42] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[43] Alex Pentland. Parts: Structured descriptions of shape. In *Proc. of the Conf. on Artificial Intelligence (AAAI)*, 1986.

[44] Anurag Ranjan, Joel Janai, Andreas Geiger, and Michael Black. Attacking optical flow. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019.

[45] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images. *arXiv.org*, 1901.08971, 2019.

[46] Igor Santesteban, Elena Garces, Miguel A. Otaduy, and Dan Casas. Softsmpl: Data-driven modeling of nonlinear soft-tissue dynamics for parametric humans. *arXiv.org*, 2004.00326, 2020.

[47] Axel Sauer and Andreas Geiger. Counterfactual generative networks. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2021.

[48] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[49] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2006.

[50] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2014.

[51] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

[52] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of RGB videos. *arXiv.org*, 2007.14808, 2020.

[53] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision (IJCV)*, 9(2):137–154, 1992.

[54] Fabio Tosi, Yiyi Liao, Carolin Schmitt, and Andreas Geiger. Smd-nets: Stereo mixture density networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[55] Bill Triggs. Factorization methods for projective structure and motion. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 845–851. IEEE, 1996.

[56] Ali Osman Ulusoy, Michael Black, and Andreas Geiger. Semantic multi-view stereo: Jointly estimating objects and voxels. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[57] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv.org*, 1807.03748, 2018.

[58] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proc. of the International Conf. on Machine learning (ICML)*, 2008.

[59] Robert J. Woodham. Analysing images of curved surfaces. *Artificial Intelligence (AI)*, 17(1-3):117–140, 1981.

[60] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil K. Jain. Adversarial attacks and defenses in images, graphs and text: A review. *arXiv.org*, 1909.08072, 2019.

[61] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In Marina Meila and Tong Zhang, editors, *Proc. of the International Conf. on Machine learning (ICML)*, 2021.

[62] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research (JMLR)*, 17(65):1–32, 2016.

[63] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. Conditional random fields as recurrent neural networks. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2015.

[64] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.