

One-Shot Learning for Handwritten Character Recognition

Giancarlo Ricci
Stanford University
gricci@stanford.edu

Abstract

I investigate one-shot learning for handwritten character recognition. Though current state-of-the-art approaches come from meta-learning frameworks, many primitive deep learning models still provide a reasonable approach to this task. In particular, I explore Convolutional Siamese Neural Networks (CSN) – including various architectures and design choices. Through implementing a CSN, I experiment with various hyperparameters, regularization techniques, and the effect of appending a memory module. In addition, I explore the effect of data-augmentation techniques – including affine distortions and right-angle rotations. Through this exploration, I achieve 93.4 accuracy for 1-shot 20-way classification on the Omniglot dataset. Though shadowed by more modern approaches, this project reveals that even relatively simple deep learning models provide compelling results in many domains – including one-shot learning.

1. Introduction

1.1. Background and Motivation

For my project, I investigate the task of one-shot learning. While humans are remarkably adept at generalizing patterns from few training examples, the same cannot be said about many modern machine learning algorithms. One standard machine learning paradigm entails training a model on large amounts of data before generalizing the learned structure to testing data. However, the requirement for vast amounts of data is not feasible in settings where data acquisition and annotation presents financial or logistical challenges. To accommodate such settings, one-shot learning restricts training to a single example for each unique class. Given my motivation to understand and develop more robust and human-like learning algorithms, I am interested in investigating one-shot learning to understand the advances made in this domain.

Specifically, I approach one-shot learning from the domain of handwritten character recognition. Handwritten character recognition is a classification problem, in which

handwritten characters of a pre-defined alphabet must be correctly categorized. Handwritten recognition interests me due to its far reaching applications – from prescription digitization in healthcare, to processing insurance claims, to banking. These numerous applications render handwritten character recognition an important problem setting.

Furthermore, handwritten character recognition naturally lends itself to the task of one-shot learning. Given the incredible range of linguistic diversity on this planet, there exist many different scripts and writing systems – some with little to no training data. To operationalize machine learning approaches in such settings, approaches like one-shot learning become necessary. I hope that through this project I can better understand and apply techniques towards under-represented and under-resourced scripts – increasing accessibility for users of these languages.

1.2. Overview of Approach

Given the premise of one-shot learning requires models to operate with little training data, it may seem that modern data-driven deep learning approaches are fundamentally incompatible with one-shot learning. Indeed, traditional neural network architectures generally over-fit with little training data – even with extensive regularization techniques such as batchnorm and norm regularization. As such, to learn meaningful semantic information from little data, an alternate neural network architecture is required. Notably, Siamese Neural Networks (SNN), produce impressive results using very little data. The specific design choices, and justification for SNN’s high-performance in this setting, will be discussed further in Section 2.

As such, for my final project, I will apply Siamese Neural Networks for one-shot learning on handwritten character recognition. My input will be characters, which come from the Omniglot dataset (discussed further in Section 4). More specifically, the input for my algorithm is an grayscale image (105 x 105 pixels) of a handwritten character from a given alphabet. The output of my algorithm is the predicted class of the each image. I quantify my classifiers performance using 1-shot 20-way accuracy.

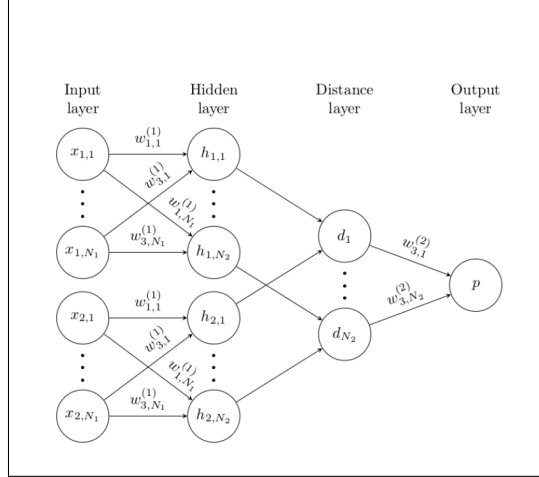


Figure 1. Schematic of Siamese Neural Network

2. Related Work

2.1. Meta-Learning Approaches

The current state of the art for one-shot learning applied to Omniglot comes from meta-learning approaches. In particular, the meta-curvature (MC) framework by Park et al. [1] claims state-of-the-art performance with an impressive 99.97% accuracy on 1-one 20-shot classification of Omniglot. MC sits alongside many other meta-learning approaches including MAML++ [2], TapNET [3], Global Class Representations [4], and Relational Nets [5] that perform comparably well on the Omniglot dataset. While these meta-learning approaches are incredibly interesting, and deserve thorough attention, I choose to focus on deep learning techniques more aligned with the course scope of CS231N for my final project.

2.2. Hierarchical Bayesian Program Learning

Inspired by human approaches in visual recognition, Hierarchical Bayesian Program Learning (HBPL) [6] explains images by combining elementary parts and their spatial relations. Through leveraging and decomposing the structure of characters, this approach parallels human psychological processes and performs quite well. Though not a deep learning approach, I find HBPL incredibly clever due to its ability to leverage pre-existing structure and human-like approach to learning.

2.3. Deep Learning and Siamese Networks

Siamese Neural Networks rely upon contrastive loss to learn. Unlike traditional neural networks that take a single input, the Siamese network takes in two inputs and feeds them into two neural networks with the same structure (that is, share the same parameters and architecture). The output of these networks is combined together and fed into fully

connected layers – outputting one number representing the similarity of the two images. Intuitively, instead of learning which image belongs to which class, the Siamese network learns how to determine the similarity between two images. After training, given a completely new image, the network compares the image to an image from each of the categories to determine which category is the most similar to the given image.

Consequently, the underlying contrastive approach of SNN lends itself well to low-data settings. In particular, the SNN does not require large amounts of training data to learn a robust, general representation of each class. Rather, it leverages the little data it has available to learn how to differentiate classes through relative similarity – such an approach works given a pre-defined number of classes. As such, the fundamental contrastive approach of SNN lends itself well to one-shot learning.

2.4. Convolution Siamese Neural Network Architecture

Recall that a Siamese Neural Network consists of two twin neural networks. These constituent neural networks can be fully connected, as shown in Figure 1. However, there is a strong incentive to make these twin networks Convolutional Neural Networks (CNN). Through using twin CNN, the overall Siamese Network can better preserve spatial information – lending itself well to tasks like character recognition. A Siamese Network consisting of twin CNN's is referred to as a Convolutional Siamese Neural Network (CSN).

Though the general CSN model provides a reasonable approach to operating in low-data settings, there exists considerable design flexibility for the specific architecture of each twin CNN. Chakrapani et al. [7], Santoro et al. [8], and Koch et al. [9] all describe different architectures for the twin CNN constituents of the larger CSN. These architectures all leverage certain similar design features – including the use of max-pooling and ReLU activation between convolutional layers, Siamese L1 distance as the contrastive loss function between twin CNN, and a regularized cross-entropy loss on the feature vector.

Despite these overall similarities, each architecture presents unique design features. Each CSN varies in its depth, respective filter size, and stride length. Furthermore, the usage of certain regularization techniques such as batch-norm and data-augmentation vary across sources. In my experiment section, I further explore these design choices and their impact on my model.

2.5. Memory Module

Kaiser et al. [10] describe a memory module that may be appended to any contrastive neural network to increase its performance in one-shot learning. This memory mod-

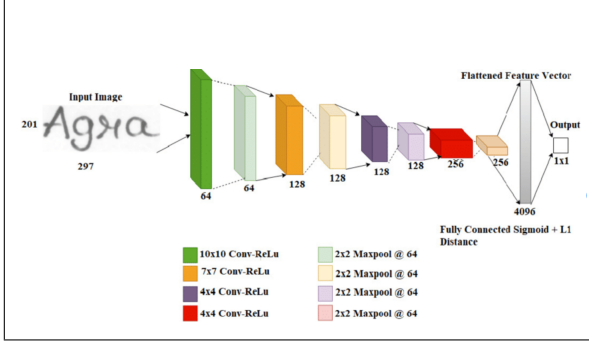


Figure 2. Chakrapani et al. Convolutional Neural Network Architecture

20-way 1-shot	Accuracy
1-Nearest Neighbor	21.7
Siamese Neural Net	58.3
Convolutional Siamese Neural Net	92.0
ConvNet w/ Memory	95.0
Hierarchical Bayesian Program Learning	95.2
MC2	99.63

Table 1. Summary of pre-existing approaches

ule consists of key-value pairs, where the keys are activations for a given neural network layer and values are the ground-truth targets of an example. More formally, memory consists of a matrix K of memory keys, a vector V of memory values, and an additional vector A that tracks the age of items stored in memory. Given $M = (K, V, A)$, Kaiser et al. leverage fast nearest-neighbor algorithms to find the value corresponding to a given normalized query. This module enhances the network’s ability to learn long-term relationships – remembering training examples shown many thousands of steps in the past and successfully generalizing from them. Furthermore, this module enables the ability to do life-long (i.e. without retraining) one-shot learning.

To implement this feature, in the simplest case, I use the final layer of my CSN network as a query to the memory module – and the output of the module is directly used for classification.

3. Methods

3.1. Model Architecture

My model is a Convolutional Siamese Neural Network. Formally, my CSN has L layers, with N_l units, where $\mathbf{h}_{1,l}$ represents the hidden vector in layer l for the first twin, and $\mathbf{h}_{2,l}$ denotes the same for the second twin. Between each layer in the first $L - 2$ layers, I use a ReLU activation func-

tion; for the remaining layers, I use sigmoid units.

The CSN consists of a sequence of convolutional layers. Each convolution uses filters with fixed stride of 1. The CSN first applies ReLU activation to the output feature, and then a maxpooling layer with a filter size and stride of 2. Consequently, the k -th filter in each layer is defined as:

$$a_{1,m}^{(k)} = \text{maxpool}(\max(0, \mathbf{W}_{1-1,1}^{(k)} \star \mathbf{h}_{1,(l-1)} + b_l), 2)$$

$$a_{2,m}^{(k)} = \text{maxpool}(\max(0, \mathbf{W}_{1-1,1}^{(k)} \star \mathbf{h}_{2,(l-1)} + b_l), 2)$$

Here, $\mathbf{W}_{1-1,1}$ represents the map for layer l , and $\star$ denotes the convolutional operation. The units in the final convolutional layer are flattened into a single vector. This convolutional layer is followed by a fully-connected layer, and then a final layer which computes the Siamese L1 distance between each twin, which is fed through the sigmoid function. More formally, the prediction vector is given as:

$$\rho = \sigma(\sum_j \alpha_j |\mathbf{h}_{1,(L-1)}^{(j)} - \mathbf{h}_{2,(L-1)}^{(j)}|)$$

This final layer scores the similarity between the two feature vectors. The α_j are additional parameters that weight component-wise distance. The final L -th fully-connected layer for the network joins the two Siamese twins.

3.2. Weight Decay

I also experimented with adding weight decay, where learning rates are decayed uniformly across the network by 1 percent per epoch. That is, where:

$$\eta(T_j) = 0.99\eta(T - 1)_j$$

3.3. Loss Function

Denote the minibatch size as M , where i indexes the i th minibatch. Now let $\mathbf{y}(x_1^{(i)}, x_2^{(i)})$ be a length- M vector which contains the labels for the minibatch, where I assume $\mathbf{y}(x_1^{(i)}, x_2^{(i)}) = 1$ whenever x_1 and x_2 are from the same character class and $\mathbf{y}(x_1^{(i)}, x_2^{(i)}) = 0$ otherwise. I impose a regularized cross-entropy objective on my binary classifier of the following form:

$$L(x_1^{(i)}, x_2^{(i)}) = \mathbf{y}(x_1^{(i)}, x_2^{(i)}) \log p(x_1^{(i)}, x_2^{(i)}) + 1 - \mathbf{y}(x_1^{(i)}, x_2^{(i)}) \log(1 - p(x_1^{(i)}, x_2^{(i)})) + \lambda^T |w|^2$$

where λ represents the penalty term applied to the weights, w represents the weights.

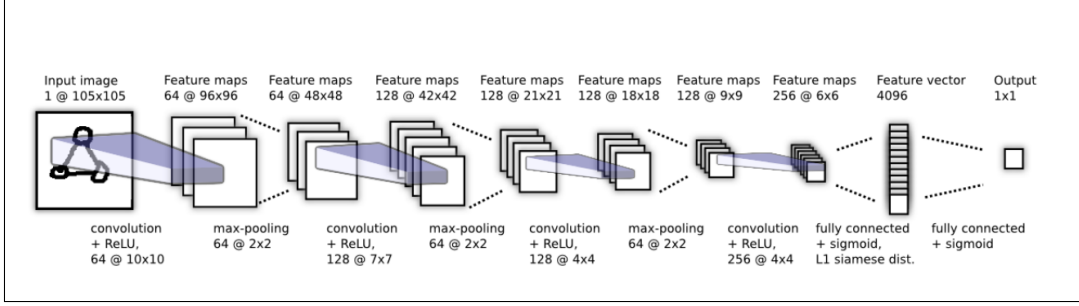


Figure 3. Koch et al. Convolutional Neural Network Architecture

3.4. Optimization

I explore various optimizers for my CSN. However, I found most success with minibatch stochastic gradient descent, with size 128. This objective can be combined with the standard back-propagation algorithm, where the gradient is additive across the twin networks. I define learning rate η_j , momentum μ_j , and L_2 regularization weights λ_j so that my update rule at epoch T is as follows

$$w_{kj}^{(T)}(x_1^{(i)}, x_2^{(i)}) = w_{kj}^{(T)} + \Delta w_{kj}^{(T)}(x_1^{(i)}, x_2^{(i)}) + 2\lambda_j |w_{kj}|$$

$$\Delta w_{kj}^{(T)}(x_1^{(i)}, x_2^{(i)}) = \eta_j \Delta w_{kj}^{(T)} + \mu_j \Delta w_{kj}^{(T-1)}$$

3.5. Weight Initialization

I initialized the network weights in the convolutional layers from a normal distribution with zero-mean and a standard deviation of 10^{-2} , as recommended from lecture. Biases weights also initialized from a normal distribution, but with mean 0.5 and standard deviation 10^{-2} . In the fully-connected layers, the weights were drawn from a wider normal distribution with zero-mean and standard deviation $2 \cdot 10^{-1}$.

3.6. Memory Module

My memory M consists of a matrix K of memory keys, a vector V of memory values, and an additional vector A that tracks the age of items stored in memory; that is: $M = (K, V, A)$. I use the final layer of the SCN as the query q , returning the value from memory that matches directly for classification (found through fast-nearest neighbors).

4. Dataset and Features

4.1. Omniglot Dataset

I are interested in applying my results to the Omniglot dataset [11]. The Omniglot data set was collected by Brenden Lake and his collaborators at MIT via Amazon's Mechanical Turk to produce a standard benchmark for learning from few examples in the handwritten character recognition

domain. The Omniglot contains 1623 characters (classes) from 50 different alphabets, including Latin and Korean. Each class contains 20 samples drawn by different people.

The number of letters in each alphabet varies from about 15 to upwards of 40 characters. All characters across these alphabets are produced a single time by each of 20 drawers. Lake split the data into a 30 alphabet background set and a 20 alphabet evaluation set. I preserve these two terms in order to distinguish from the normal training, validation, and test sets that can be generated from the background set in order to tune models for verification. The background set is used for developing a model by learning hyperparameters and feature mappings. Conversely, the evaluation set is used only to measure the one-shot classification performance.

4.2. Data Augmentation

Given that there exist such few examples in the training data, various techniques have been explored to augment the dataset. This includes modifying the training data in a way to preserve the underlying semantic information, but make the classifier more robust to slight perturbations or noise in the data.

4.2.1 Rotation

Santoro et al. describe one form of data-augmentation through randomly rotating and translating images. These methods are intended to prevent over-fitting. This approach will be explored in my experiment section.

4.2.2 Affine Distortions

Koch et al. describes a procedure for affine distortions of training data. More formally, for each image pair x_1, x_2 , I generated a pair of affine transformations T_1, T_2 to yield.

$$x'_1 = T_1(x_1)$$

$$x'_2 = T_2(x_2)$$

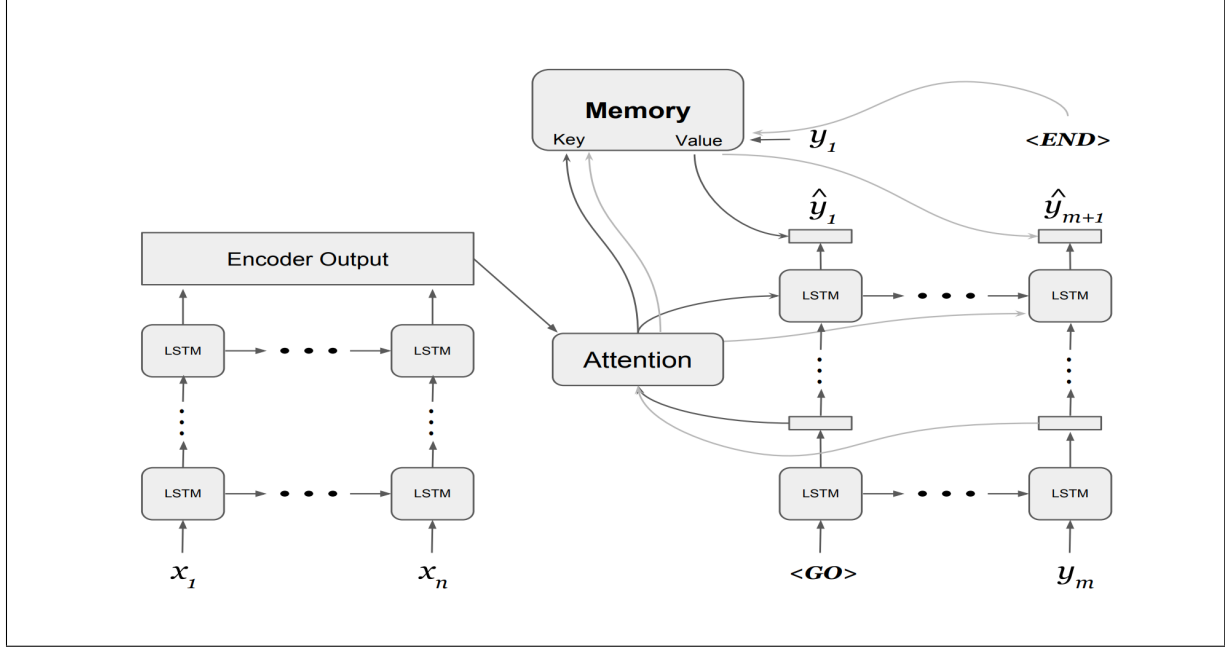


Figure 4. Memory Module from Kaiser et al. added to GNMT

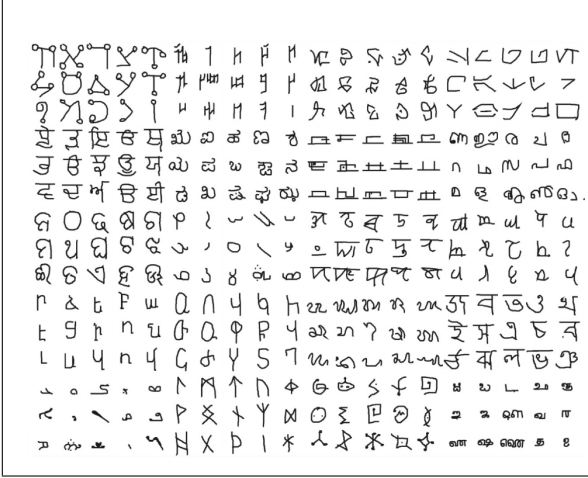


Figure 5. Example Characters from the Omniglot dataset

where T_1, T_2 are determined stochastically from multi-dimensional uniform distribution. So for an arbitrary transform T , I have:

$$T = (\theta, \rho_x, \rho_y, s_x, s_y, t_x, t_y) \text{ with } \theta \in [-10.0, 10.0] \\ \rho_x, \rho_y \in [-0.3, 0.3] \quad s_x, s_y \in [0.8, 1.2] \quad t_x, t_y \in [-2, 2]$$

Each of these components of the transformation is included with probability 0.5.

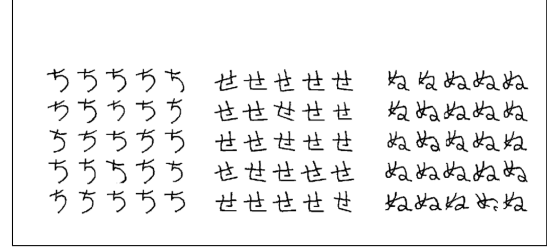


Figure 6. Affine Distortions

5. Experiments

5.1. Baseline Architecture

To establish baseline performance results, I implemented CSN with various architectures. Chakrapani et al., Santoro et al., and Koch et al. all describe different CSN architectures. However, only Koch et al. is designed for the Omniglot dataset, i.e. inputs of size 105 x 105 pixels. As such, the specific architecture presented by Chakrapani et al. and Santoro et al. had to be adapted to fit the reduced input size. I sought to preserve the relative network depth, changing the filter size of each convolutional layer instead. Though these architectures perform in similar ranges, the best empirical model comes from Koch et al. This result may not be surprising, given that Koch et al. developed their architecture on the Omniglot dataset, so it may be more fine-tuned than the other approaches.

It should also be noted that my baseline accuracy is

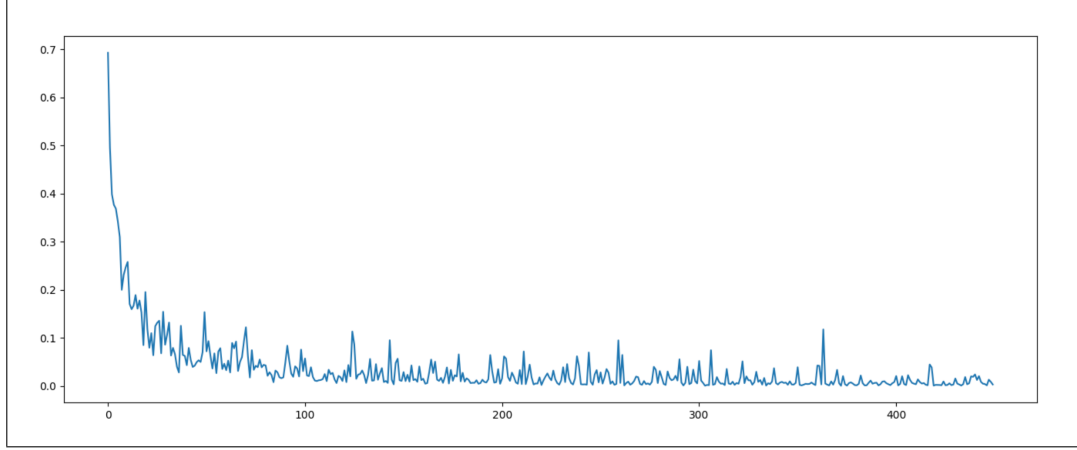


Figure 7. Loss while Training Koch et. al Architecture using Vanilla SGD

CSN Architecture	Accuracy
Koch et al.	89.9
Inspired by Chakrapani et al.	87.7
Inspired by Santoro et al.	88.8

Table 2. Summary of different CSN Architectures

Batchnorm	Accuracy
Vanilla	89.9
P = 0.5	89.8
P = 0.8	88.6

Table 3. Impact of Batchnorm

Data Sugmentation	Accuracy
Vanilla	89.9
Rotation	83.5
Affine Distortion	90.5

Table 4. Data Augmentation

Optimizers	Accuracy
SGD	90.3
Adam	90.2
SGD + Weight Decay	90.5

Table 5. Optimizer

slightly lower than Koch et al., who achieve 92.0% accuracy. This difference likely comes from hyperparameter tuning. Koch ran a Bayesian optimization framework to fine-tune, whereas I tuned my hyperparameters through random search. This difference may contribute to my decrease accuracy.

5.2. Regularization Techniques

I also explore a variety of regularization techniques. To begin, I explore the impact of batchnorm in each CNN, with varying dropout probabilities.

I found the batchnorm had little to no-effect on the overall accuracy. This may be due to the regularization approaches already in place, or due to the fact that the network is sensitive to overfitting due to its little training data. As such, dropout did not seem to generate meaningfully new structure dependencies.

5.3. Data Augmentation

Various procedures for data augmentation were proposed. I explore the result of right-angle rotation (90, 180, 270 degree) and affine distortion.

As with Koch et al. my implementation of affine distortion improved accuracy considerably. This makes intuitive sense, as affine distortions slightly perturbs the data while keeping semantic information intact.

Conversely, rotation slightly worsened my accuracy. This may be explained by the fact that sometimes orientation provides meaningful semantic information. For example, the English letters "M" and "W" and "E" may be easily confused through rotation. Perhaps further work done need to be done fine-tuning the exact range of these rotations for meaningful results.

5.4. Optimization

Koch et al. paper suggests Stochastic Gradient Descent. Empirically, SGD with weight-decay performed better than

Memory Module	Accuracy
Vanilla	90.5
Memory	93.4

Table 6. Impact of Memory Module

Adam and SGD without weight decay. However, all of these optimizers produced relatively similar performances – with training time being the primary difference between these models.

5.5. Memory Module

My primary intervention to the pre-existing CSN architecture was the addition of a memory module. Though Kaiser et al. created open-source tensorflow code that made this addition very simple, the memory module provided a suprisingly profound impact on my results.

It is interesting that memory module works so well even with a relatively standard architecture, and not the more completed Matching Net [12] that Kaiser compared to.

5.6. Discussion

This section allowed me to explore various design choices for my deep learning model. Through experimenting with various CSN architectures, hyperparameters, and regularization techniques, I was able to marginally increase performance. However, the greatest increases came from data-augmentation using affine distortion and the additional of the memory module.

6. Conclusion

6.1. Recap

Overall, this project has been a great way to apply deep learning approaches to an important problem domain. On one hand, the current state-of-the-art techniques rely on meta-learning approaches that fall outside of the scope of the deep learning techniques presented in 231N. That said, it is quite impressive that even relatively primitive deep learning approaches can be modified to perform well into low-data settings.

This project provided me with the opportunity to empirically test various CSN architectures, parameters, hyperparameters, and overall design choices. The overall design flexibility led to some dubious design decisions, and affirmed that training and creating deep learning models is not always easy to do.

6.2. Future Directions

Overall, many modern meta-learning techniques have surpassed these techniques in performance. As such, explo-

ration of different model architectures and design choices within this paradigm will likely not lead to fruitful results.

That said, these results still perform well-enough to achieve decent results on the Omniglot dataset. It would be an interesting measure of robustness to see how these models transfer to other Datasets. Koch et al. performed a similar experiment, testing Omniglot on a reduced "one-shot" version of MNIST. Similar experiments could be performed with other datasets.

I also think that the method of CSN applied to one-shot learning is more general than just handwritten character recognition. In future projects, I would like to explore how my model performs on other one-shot learnings tasks – in computer vision and other non-CV related tasks.

7. Contributions

I, Giancarlo Ricci, wrote this project in entirety. Though I was originally in a partnership with Kaiyu Ren, he did not contribute to this final project in any way. I quite literally wrote every single line of code, and every single word in this document. I also fully researched and planned out the intellectual work. He did not contribute to any intellectual work or our authorship in this project.

I am not comfortable pretending that this project was a group effort. To do so goes against Stanford's Honor Code and my own code of ethics. I am happy to cooperate with the teaching time in any way to verify any of these claims. This has been an incredibly rewarding project for me, however, the supposed group aspect has been incredibly frustrating. Thank you for understanding.

References

- [1] Eunbyung Park and Junier B. Oliva. Meta-curvature. *CoRR*, abs/1902.03356, 2019. 2
- [2] Antreas Antoniou, Harrison Edwards, and Amos J. Storkey. How to train your MAML. *CoRR*, abs/1810.09502, 2018. 2
- [3] Sung Whan Yoon, Jun Seo, and Jaekyun Moon. Tapnet: Neural network augmented with task-adaptive projection for few-shot learning. *CoRR*, abs/1905.06549, 2019. 2
- [4] Tiange Luo, Aoxue Li, Tao Xiang, Weiran Huang, and Liwei Wang. Few-shot learning with global class representations. *CoRR*, abs/1908.05257, 2019. 2
- [5] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. *CoRR*, abs/1711.06025, 2017. 2
- [6] Brenden M Lake, Russ R Salakhutdinov, and Josh Tenenbaum. One-shot learning by inverting a compositional causal process. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural*

Information Processing Systems, volume 26. Curran Associates, Inc., 2013. 2

- [7] Asish Chakrapani Gv, Sukalpa Chanda, Umapada Pal, and David Doermann. One-shot learning-based handwritten word recognition. pages 210–223, 02 2020. 2
- [8] Adam Santoro, Sergey Bartunov, Matthew M. Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065, 2016. 2
- [9] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015. 2
- [10] Lukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *CoRR*, abs/1703.03129, 2017. 2
- [11] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 4
- [12] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016. 7