Prototypical one-shot and k-shot learning on Omniglot using high dimensional embeddings and a mixture of Gaussians

Stanislav Fort
Stanford University
sfort1@stanford.edu

Abstract

We explore a prototypical network for k-shot classification on the Omniglot dataset. A prototypical network learns a Euclidean embeddings of images and uses their clustering for classification. This approach generalizes to different numbers of classes as well as previously unseen classes. We managed to replicate previous state of the art results on one-shot, and five-shot 20-way classification. Using a larger network, we surpassed these results by statistically significant margin. We also propose a Gaussian prototypical network which, together with a vector embedding of an image, learns its covariance matrix, and uses it to construct a direction and class dependent distance metric on the embedding space. This allows the network to reflect inherent spread of images within a class. We reach state of the art results on Omniglot with this approach as well.

1. Introduction

1.1. Few-shot learning

Deep learning has been performing well in settings with an abundance of data. In general, deep learning models have a very high functional expressivity and capacity, and rely on being iteratively trained in a supervised regime on very large dataset. An influence of a particular example within the training set is therefore small, as the training is designed to capture the general behaviour of the data set. [13]

In contrast, a lot of problems requires a very fast adaptation to new data. In particular, one-shot, few-shot, and k-shot classification refer to a regime where classes unseen during training must be incorporated using a single, a few, and k examples respectively. In this regime, the effect of a small number of examples (in one-shot classification of a single one) must be very large.

Humans are able to learn to recognize new object categories on a single or a small number of examples. This has been demonstrated in a wide range of activities from hand-written character recognition [3], and motor control [2], to acquisition of high level concepts [10].

1.2. Related work

Non-parametric models, such as k-nearest neighbours (kNN), are an ideal candidate for a few-shot classification, as they allow for incorporation of previously unseen classes. However, using the distance in the space of inputs (e.g. raw pixel values) directly does not result into a high accuracy, is the the connection between the semantic meaning of an image and its pixel values is very non-linear.

A straightforward modification in which a metric embedding is learned and then used for kNN classification has yielded good results, as demonstrated by [5], [9], and [1]. An approach using *matching networks* has been proposed in [14], in effect learning a distance metric between pairs of images. A noteworthy feature of the method is its training scheme, where each mini-batch (called an *episode*) tries to mimic the data-poor test conditions by subsampling the number of classes as well as numbers of examples in each. It has been demonstrated that such an approach improves performance on few-shot classification. [14]

Instead of learning on the dataset directly, [12] has recently proposed to train an LSTM [7] to predict updates to a few-shot classifier given an episode as its input. This approach is referred to as *meta-learning*.

Combinations of parametric and a non-parametric methods have been yielding the best few-shot learning results recently. [15][6][8]

In this paper, we explore a *prototypical network* that does not suffer from severe overfitting and according to our knowledge reaches state of the art performance on the *Omniglot* dataset. [8] By increasing the model size compared to [8], we are able to exceed their results by a statistically significant margin.

Furthermore, we propose a modification of a prototypical network, a *Gaussian prototypical network*, that learns how to embed images into a Euclidean space as well as to generate a covariance matrix for each image, reflecting a confidence about its value and importance in different directions of the embedding space. We show that a Gaussian prototypical network slightly outperforms a vanilla prototypical network, and reaches a state of the art k-shot classification result on Omniglot.

This paper is structured as follows: we describe the prototypical and Gaussian prototypical network in Section 2. The episodic training scheme is also presented there. We discuss the Omniglot dataset in 3 and our experiments in 4. Final, our conclusions are presented in 5.

2. Methods

In this paper, we first explore the *prototypical networks* described in [8]. We then continue to present our modification to the architecture, which we call a *Gaussian prototypical network*. The encoder architecture does not differ apart from its final layer, and we therefore describe both of them together. The same applies to the training scheme.

2.1. Encoder

We use a multi-layer convolutional neural network without a fully connected layer to transfer images into high-dimensional Euclidean vectors. We have been experimenting with fully connected layers, and CNNs followed by a fully-connected layer as well. The fully-connected layer on its own does not have a good spatial awareness and therefore does not perform well in image recognition. A CNN followed by a fully-connected layer does not seem to perform better than a CNN whose outputs are directly reshaped into an embedding vector. We therefore primarily used a set of stacked CNNs as our encoder.

For a vanilla prototypical network, the encoder is a function taking and image I and transforming it into a vector $\vec{x}$ as

$$\mathrm{encoder}(W) : I \in \mathbb{R}^{H\times W\times C} \to \vec{x} \in \mathbb{R}^{D} \,, \qquad (1)$$

where H and W are the height and width of the input image, and C the number of its channels. D is the embedding dimension of our vector space and it is a hyperparameter of the model. W are the trainable weights of the encoder. In our models, the encoder was always made of 4 layers of CNN.

For a Gaussian prototypical network, the output of the encoder has a dimension $2D$, as it is to be understood as a concatenation of an embedding vector $\vec{x} \in \mathbb{R}^D$ and the diagonal of a covariance matrix $\vec{\sigma} \in \mathbb{R}^D$. Therefore

$$\mathrm{encoder}_{\mathrm{Gauss}}(W) : I \in \mathbb{R}^{H\times W\times C} \to [\vec{x}, \vec{\sigma}] \in \mathbb{R}^{2D} \,. \quad (2)$$

We used down-sampled gray-scale Omniglot images of the dimension $28 \times 28 \times 1$ as an input. A 4-layer CNN architecture with 2×2 max pooling therefore resulted into a volume of shape $1\times 1\times D$, where the embedding dimension D is equal to the number of filters in the last later. We were using the `TensorFlow` *SAME* padding and stride 1. Our filters were 3×3 in spatial extent.

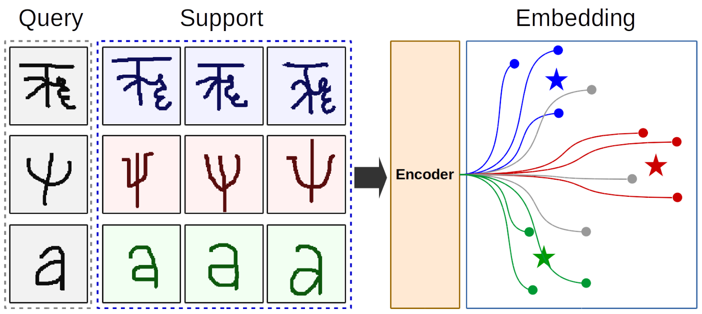

Figure 1. A diagram of a function of a prototypical network. An encoder maps an image into a vector in the embedding space. Support images are used to define the centroids of the particular classes (shown as stars). The distances between these centroids and encoded query images are used to classify them.

We were using 2 encoder architectures: 1) a **small** architecture, and 2) a **big** architecture. The small architecture corresponded to the one used in [8] and we used to validate our own experiments with respect to the state of the art results. The big architecture was used to see the effect of increasing the model capacity on accuracy. As a basic building block, we used a series of a 3×3 CNN, batch normalization, dropout, ReLU, and 2×2 max pooling (in this order). Both architectures were composed of 4 of these blocks stacked together. The details of the architectures are as follows:

1) **Small architecture:** 3×3 filters, numbers of filters $[64, 64, 64, 64]$ ($[64, 64, 64, 128]$ for a Gaussian model). Embedding vector dimension $D = 64$.

2) **Big architecture:** 3×3 filters, numbers of filters $[128, 256, 512, 256]$ ($[128, 256, 512, 512]$ for a Gaussian model). Embedding vector dimension $D = 256$.

2.2. Episodic training

A key component of the prototypical model is the episodic training regime. During training, a subset of N_c classes is chosen from the total number of classes in the training set without replacement. For each of these classes, N_s of *support* examples are chosen at random, as well as N_q of *query* examples. The encoded embeddings of the support examples are used to define where a particular class c lies in the embedding space. The distances between the query examples and positions of classes determined from the support points are then used to classify and calculate a loss. A diagram of the process is shown in Figure 1.

For a Gaussian prototypical network, the diagonal of a covariance matrix is output together with the embedding vector. These are then used to weight the embedding vectors corresponding to support points of a particular class, as well as to calculate a total covariance matrix for the class.

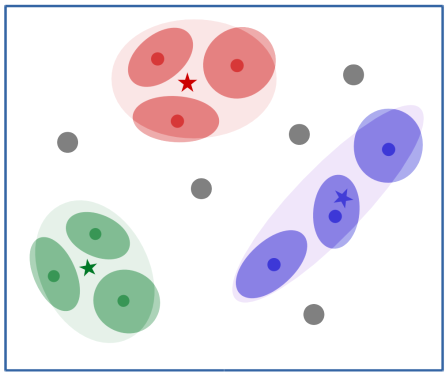

Figure 2. A diagram showing the embedding space of a Gaussian prototypical network. An image is mapped to its embedding vector (dark dot) by the encoder. Its covariance matrix (small ellipses) is also output by the encoder. An overall covariance matrix for each class is then computed(large light ellipses), as well as centroids of the classes. The covariance matrix of a class is used to locally modify the distance measure to query points (shown in gray).

Distances from class c are then calculated as

$$d_c(i) = (\vec{x_i} - \vec{p_c})^T \frac{1}{\Sigma_c} (\vec{x_i} - \vec{p_c}) \ , \tag{3}$$

where $\vec{p_c}$ is the centroid, or *prototype* of the class c and Σ_c its covariance matrix. The Gaussian prototypical network is therefore able to learn class, and direction-dependent distance metric in the embedding space. A diagram of the embedding space for a Gaussian network is shown in Figure 2. In our case, we set the covariance matrix of an example i to be

$$\Sigma_i = \text{diag}\left(\vec{\sigma}_i\right) \ , \tag{4}$$

where $(\vec{\sigma}_i$ is the second half of the output of the encoder for a Gaussian network. This is done to make the model faster, as a full covariance matrix would be $D \times D$ in size. In our experiments, we also try setting the $\Sigma_i = sI$, where $s \in \mathbb{R}$ is just a scalar. Effectively, this means that each point only carries a sphere of radius s with it.

The training algorithm is described as follows:

1) Choose a subset C of N_c classes from all possible training classes (without replacement)

2) Choose N_s *support* examples for each class $c \in C$.

3) Choose N_q *query* examples for each class $c \in C$.

4) Calculate embedding vectors $\vec{x}_i$ of all chosen images (and their $\vec{\sigma}_i$ if using the Gaussian network).

5V) If using a **vanilla** prototypical network, evaluate distances of query points to classes as:

   a) For each class c, calculate the position of the class prototype (centroid)

$$\vec{p}_c = \frac{1}{N_s} \sum_{i \in c(\text{support})} \vec{x}_i$$

   b) Calculate the distance between prototype c and query point i $d_c(i)$.

5G) If using a **Gaussian** prototypical network, evaluate distances of query points to classes as:

   a) For each class c, calculate the position of the class prototype (centroid)

$$\vec{p}_c = \left(\sum_{i \in c(\text{support})} \frac{\vec{x}_i}{\sigma_i^2} \right) / \left(\sum_{i \in c(\text{support})} \frac{1}{\sigma_i^2} \right)$$

   b) For each class c, calculate the class covariance matrix Σ_c as

$$\Sigma_c = \left(\sum_{i \in c(\text{support})} \frac{1}{\sigma_i^2} \right)^{-1/2}$$

   c) Calculate the locally defined distance between prototype c and query point i

$$d_c(i) = (\vec{x_i} - \vec{p_c})^T \frac{1}{\Sigma_c} (\vec{x_i} - \vec{p_c}) \ .$$

6) Classify query points based on the closest class as $\hat{y}(i) = \text{argmin}_c d_c(i)$.

7) Calculate the cross-entropy loss as $\text{loss} = \text{CE}(y, \hat{y})$ over the query points, where y are the known correct classes.

2.3. Distance metric

The distance metric used does not change the result of classification, as long as it is monotonously increasing with the Euclidean distance. The key difference lies in the ease of training. We have explored the cosine and L_2 norms, and found that the latter outperforms the former. We also explored powers of the L_2 norm, and found that the norm itself works the best.

2.4. Gaussian local metric

For the Gaussian prototypical network, the distance between a query point i and a class c is class dependent, as well as direction dependent. Intuitively, this allows classes that would more easily assume a non-spherical shape to do so, while using their size in the direction in question as a standard ruler.

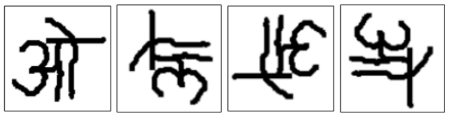
Figure 3. An example of class number augmentation by rotations. An original character (on the left) is rotated by 90°, 180°, and 270°. Each rotation is then defined as a new class.

2.5. Evaluating models

To estimate the accuracy of a model on the test set, we let it go through the whole test set multiple times with a number of support points $N_s = k$ for $k \in (1, ..19)$. Then, the accuracies are aggregated. Since we are not using a designated validation set, we then take the test results for 3 or 5 (3 for small models, and 5 for big) highest training accuracies and calculate their mean and standard deviation of the mean. By doing that, we prevent overfitting on the test set.

3. Datasets

We were working with the Omniglot dataset in this paper. [11] Omniglot contains 1623 character classes from 50 alphabets (real and fictional) and 20 hand-written examples of each. We were using the recommended split to 30 training alphabets, and 20 test alphabets, as suggested by [11] and used by [8]. The training set included overall 964 unique character classes, and the test 659 of them. There was no class overlap between the training and test datasets.

To extend the number of classes, we augmented the dataset by rotating each character by 90°, 180°, and 270°, and defined each rotation to be a new character class on its own. The same approach is used in [8] and [14]. An example of an augmented character is shown in Figure 3. This increased the number of classes 4-fold. In total, the training set therefore included 77,120 images, and the test set 52,720 images. Due to our rotational augmentation, characters that have a rotational symmetry will be classified as multiple classes. As even a human is not able to recognize the charactor ”O” from a mirror ”O”, $100\,\%$ accuracy will not be reachable.

Each Omniglot character comes as a gray-scale $105 \times 105 \times 1$ image. We downsample them to $28 \times 28 \times 1$, and subtract the mean of each image from itself. Omniglot also provides characters with background value 1.0 and writing vale 0.0. We invert this to have the character at a higher value than the background.

4. Experiments

We conducted a large number of experiments with vanilla as well as Gaussian prototypical networks. In general, we explored the size of the encoder (small, and big, as described in Section 2), the vanilla/Gaussian prototypical method, the distance metrics (cosine, $\sqrt{L_2}$, L_2, and L_2^2, and the number of degrees of freedom for the covariance matrix in the Gaussian networks.

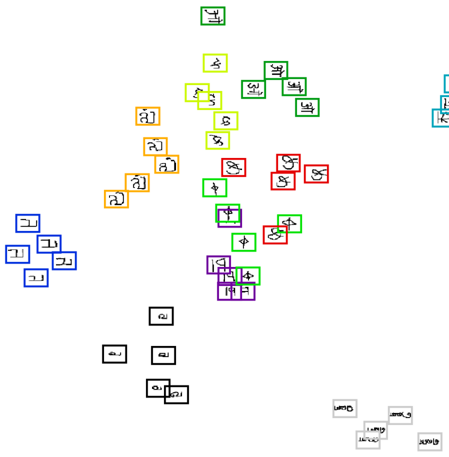
Figure 4. A PCAed two dimensional slice of the embedding space during training. Clustering of similar characters is apparent in the project.

We were using *Adam* as our optimizer with an initial learning rate of 10^{-3}. We then halved the learning rate each 2000 episodes $\approx$ 30 epochs. All our models were implemented in `TensorFlow` and ran on a single NVidia K80 GPU on Google Cloud.

We implemented the prototypical network described in [8] and managed to reach very comparable results. We trained with $N_c = 60$ classes, and tested on $N_{ct} = 20$, i.e. 20-way classification. During training each class present in the mini-batch comprised $N_s = 1$ support point. The results can be found in Table 1.

A PCAed sample of the embedding space during training is shown in Figure 4. It illustrates the clustering of similar characters.

We conducted the same experiments with the large encoder architecture described in Section 2. We managed to outperform [8] on both 1 and 5-shot 20-way classification by a statistically significant margin. The results are summarized in Table 1.

We them used the Gaussian prototypical network with its covariance matrix fixed to a single scalar and used a small encoder. The network outperformed its small vanilla counterparts in 1-shot learning and reached equivalent results for 5-shot.

To study the effect of capacity on the model performance, we experimented with the big encoder architecture for the vanilla model, Gaussian model with covariance fixed to a scalar, and a Gaussian model with covariance fixed to a diagonal of the covariance matrix. The results are shown in Table 1.

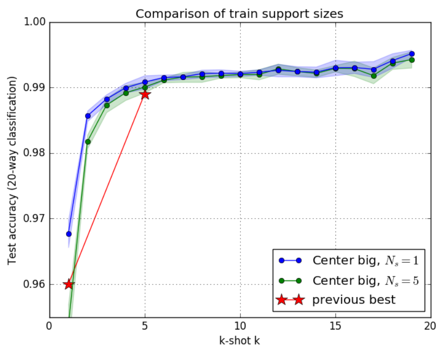

Figure 5. The effect of support size on test accuracy. The plot shows the test accuracy for k-shot 20-way classification for our vanilla big model training with support size $N_s = 1$, and support size $N_s = 5$. The smaller support size improves the accuracy. The previous state of the art is shown in red.

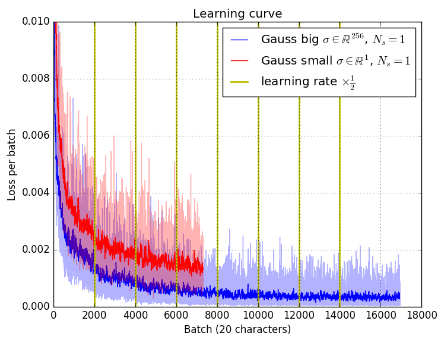

Figure 6. The effect of model size on loss. The bigger model trains faster and reaches a smaller loss overall. The yellow vertical lines show where the learning rate was halved.

The effect of the number of support examples during training on test accuracy is shown in Figure 5. The $N_s = 1$ does better on all k. A comparison of training loss as a function of epoch for a small and a large model is shown in Figure 6. The training accuracy and its comparison to test performance for our model is shown in Figure 7. There is a significant gap between the training and test accuracy which shows that we having a large enough model.

The performance of our best small and large models is shown in Figure 8 with respect to the current state of the art from [8]. The performance of our best big models is compared in Figure 9. The model with a trainable covariance matrix diagonal performs the best overall, however, the differences are not as significant.

The Table 2 compares our best small and big models to the current state of the art and previous attempts at k-shot learning.

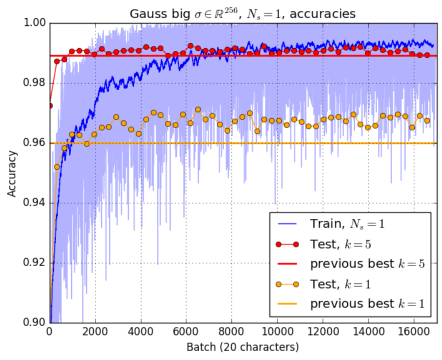

Figure 7. The training accuracy as compared to the test accuracy. The plot shows the training accuracy of a large Gaussian prototypical model and compares it to its 1-shot and 5-shot test performance (20-way classification). It also compares the results to the current state of the art. [8]

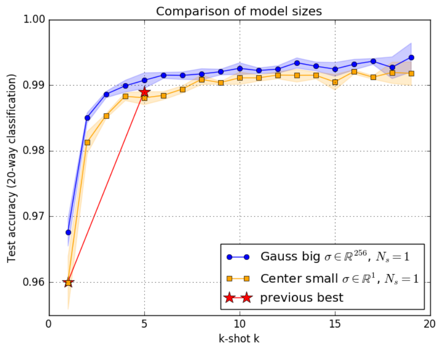

Figure 8. Comparsion of k-shot test accuracy of our best big and small models.

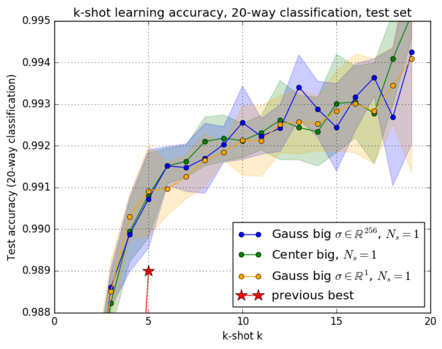

Figure 9. The performance of different versions of the big encoder model and their uncertainties. The Gaussian model with a trainable covariance matrix diagonal performs the best overall, however, the differences are not very significant.

5. Conclusion

In this paper we investigated the usage of prototypical networks for few-shot classification on Omniglot. We also

Method (20-way classification)	1-shot test	5-shot test
Best results from [8]	$96.0\,\%$	$98.9\,\%$
Small, $N_s = 5$	$94.28 \pm 0.26\,\%$	$98.61 \pm 0.08\,\%$
Small, $N_s = 1$	$95.64 \pm 0.22\,\%$	$98.87 \pm 0.05\,\%$
Small Gauss $\sigma \in \mathbb{R}^1$, $N_s = 1$	$95.99 \pm 0.28\,\%$	$98.81 \pm 0.07\,\%$
Big, $N_s = 5$	$95.36 \pm 0.16\,\%$	$99.01 \pm 0.04\,\%$
Big, $N_s = 1$	$96.77 \pm 0.10\,\%$	$99.08 \pm 0.05\,\%$
Big Gauss $\sigma \in \mathbb{R}^{256}$, $N_s = 1$	$96.76 \pm 0.10\,\%$	$99.07 \pm 0.06\,\%$
Big Gauss $\sigma \in \mathbb{R}^1$, $N_s = 1$	$96.78 \pm 0.09\,\%$	$99.09 \pm 0.06\,\%$

Table 1. Results of our experiments. State of the art for 20-way classification 1-shot was $96.0\,\%$ and for 5-shot $98.9\,\%$. Small means the small encoder architecture, big is the big encoder archicture. N_s is the number of support points per class during training. All training done in $N_c = 60$ (60-way classification) regime. For the Gaussian prototypical model $\sigma \in \mathbb{S}$ shows the dimensionality of the estimated covariance matrix.

Method (20-way classification)	1-shot test	5-shot test
Matching networks [14]	$93.8\,\%$	$98.5\,\%$
Matching networks [14]	$93.5\,\%$	$98.7\,\%$
Neural statistician [4]	$93.2\,\%$	$98.1\,\%$
Prototypical network [8]	$96.0\,\%$	$98.9\,\%$
Small Gauss $\mathbb{R}^1$, $N_s = 1$	$96.0 \pm 0.3\,\%$	$98.8 \pm 0.1\,\%$
Big Gauss $\mathbb{R}^1$, $N_s = 1$	**96.8** $\pm$ **0.1** $\%$	**99.1** $\pm$ **0.1** $\%$

Table 2. The best results of our experiments. N_s is the number of support points per class during training. All training done in $N_c = 60$ (60-way classification) regime. For the Gaussian prototypical model $\sigma \in \mathbb{S}$ shows the dimensionality of the estimated covariance matrix. We manage to outperform previous work with our large network, and perform similarly to the previous state of the art with our small network.

proposed an improved version of a prototypical network which we call a Gaussian prototypical network. The prototypical networks proved to be very useful for k-shot classification and we managed to replicated state of the art results on Omniglot with them. [8] By increasing the model size, we outperformed the results by a statistically significant margin. The Gaussian prototypical networks typically performed better then the vanilla networks, although with a smaller margin. Contrary to [8], we found that the best results are obtained if one trains the network in the 1-shot regime.

References

[1] H. A. Bellet, Aurlien and M. Sebban. A survey on metric learning for feature vectors and structure data. *arXiv:1306.6709*, 2013.

[2] A. A. W. D. M. Braun, Daniel A and C. Mehring. Motor task variation induces struc- tural learning. *Current Biology, 19(4):352357*, 2009.

[3] J. G. Brenden M. Lake, Ruslan Salakhutdinov and J. B. Tenenbaum. One shot learning of simple visual concepts, 2011.

[4] H. Edwards and A. Storkey. Towards a neural statistician. *International Conference on Learning Representations*, 2017.

[5] H. G. E. R. S. T. Goldberger, Jacob and R. Salakhutdinov. Neighbourhood components analysis. *Advances in Neural Information Processing Systems*, 2004.

[6] R. S. Gregory Koch, Richard Zemel. Siamese neural networks for one-shot image recognition. 2015.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation, 9(8)*, 1997.

[8] R. S. Z. Jake Snell, Kevin Swersky. Prototypical networks for few-shot learning, 2017.

[9] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning, 5(4)*, 2012.

[10] S. R. Lake, Brenden M and J. B. Tenenbaum. Human-level concept learning through prob- abilistic program induction. *Science, 350(6266):1332 1338*, 2015.

[11] J. B. T. M. Lake1, Ruslan Salakhutdinov. Human-level concept learning through probabilistic program induction. 2015.

[12] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2017.

[13] e. a. Santoro A. One-shot learning with memory-augmented neural networks. 2016.

[14] B. C. L. T. W. D. e. a. Vinyals, Oriol. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 2016.

[15] J. Z. K. H. Weiqiang Ren, Yinan Yu. Learning convolutional nonlinear features for k nearest neighbor image classification. 2014.