VRIJE UNIVERSITEIT AMSTERDAM

A Scalable Logo Recognition Model with Deep Meta-Learning

by

Mark de Blaauw

An internship thesis submitted in partial fulfillment for the degree of Master of science

in the
Faculty of Science
Business Analytics

Supervisor: Dr. Diederik M. Roijers
Second reader: Dr. Eduard N. Belitser
Host organisation: Mobiquity Inc.
External supervisor: Dr. Vesa Muhonen

September 2019

Abstract

The goal in this thesis is to create a deep learning (DL) logo classifier that is widely applicable for conferences. We train this classifier prior to a conference for logos of organisations that will be present. However, DL models typically require many samples to learn such a classification task, whereas deep meta-learning models can learn different but related tasks with 1-10 training samples per class. This reduces data collection costs and increases deployment speed. In this thesis, we investigate if it is possible to build a scalable DL logo recognition model that is able to learn classification tasks with only five training samples per class.

Current techniques in the deep meta-learning domain focus primarily on accuracy, although for this AI-driven application, multiple factors play an important role: (i) robustness of accuracy to different tasks, (ii) robustness to in- and out-domain distribution unknown-class samples, and (iii) the flexibility to learn new tasks. We show for (i) that *deep metric learning* approaches are more robust to tasks with more classes than *initialisation learning* approaches. We show for (ii) that Gaussian prototypical network [1] is more robust to in- and out-domain distribution unknown-class samples than Reptile [2]. For (iii), we conclude that deep metric-learning approaches are more flexible to learning new tasks than initialisation and model-based learning approaches.

The conclusions we make start by creating two logo meta-learning datasets, the small- and large logo dataset. With these datasets, we compare different popular deep meta-learning models for which Gaussian prototypical networks gives the best performance on the small logo dataset. We propose a simpler version of Gaussian prototypical networks that we call *Mahalanobis prototypical networks* and use the small logo dataset to show that it performs equally well.

Deep metric-learning approaches, and especially the Mahalanobis prototypical network model, seems to be the most appropriate technique to use for this conference application based on accuracy, robustness to in- and out-domain distribution unknown-class samples, robustness to tasks with more classes, and flexibility. Although our Mahalanobis prototypical network model is a simpler method than Gaussian prototypical networks, we show it has comparable performance. Hence, we conclude that the Mahalanobis prototypical network model is able to learn new logo classification tasks with only five training samples. We further note that robustness to in- and out-domain distribution unknown-class samples and robustness to tasks with more classes can be important metrics for practical deep meta-learning such as logo classification.

Acknowledgements

During the time of writing this thesis I have received a great deal of support and assistance from people to whom I would like to express my gratitude. First, I would like to thank my supervisors Diederik Roijers and Vesa Muhonen for challenging my ideas, providing my work with feedback, and helping me to write a thesis abstract for the BENELEARN 2019 conference. Second, I would like to thank Richard Price for bouncing off ideas and proofreading some parts of my thesis. Last but not least, I would like to thank Mobiquity for facilitating this graduation project.

Contents

Abstract **i**

Acknowledgements **ii**

List of Figures **vi**

List of Tables **ix**

1 Introduction **1**
- 1.1 Main contributions . . . 2

2 Problem definition and research setup **5**
- 2.1 Definitions and model journey . . . 5
- 2.2 Quality metrics and requirements . . . 7
- 2.3 Research setup . . . 10
  - 2.3.1 Main- and sub-questions . . . 10
  - 2.3.2 Research setup . . . 10

3 Background and literature research **11**
- 3.1 Neural networks . . . 11
  - 3.1.1 Convolutional neural networks . . . 15
    - 3.1.1.1 Convolutional neural network structures . . . 17
- 3.2 Transfer learning . . . 19
- 3.3 Meta-learning . . . 21
  - 3.3.1 Introduction to meta-learning . . . 22
    - 3.3.1.1 Training a meta-learning model . . . 22
  - 3.3.2 Three directions in meta-learning . . . 24
    - 3.3.2.1 Deep metric learning . . . 25
    - 3.3.2.2 Initialisation learning . . . 27
    - 3.3.2.3 Model based learning . . . 28
  - 3.3.3 Meta-learning data and models . . . 29
- 3.4 Discussion of literature research . . . 30

4 Empirical analysis on meta-learning models **33**
- 4.1 Data . . . 33
  - 4.1.1 Omniglot dataset . . . 34
  - 4.1.2 Small logo dataset . . . 34

4.2 Methods . . . 37
4.2.1 Deep metric learning . . . 38
4.2.1.1 Prototypical networks . . . 38
4.2.1.2 Relation networks . . . 42
4.2.1.3 Proxy networks . . . 43
4.2.2 Initialisation learning . . . 45
4.2.2.1 MAML . . . 45
4.2.2.2 Reptile . . . 46
4.2.3 Model-based learning . . . 47
4.3 Experimental setup . . . 48
4.4 Results . . . 49
4.5 Discussion of empirical analysis . . . 50
4.5.1 Omniglot dataset performance results . . . 50
4.5.2 Small logo dataset performance results . . . 51
4.5.3 Chapter discussion . . . 53

5 Research Gaussian prototypical networks and Reptile 54
5.1 Introduction of large logo dataset . . . 55
5.2 Investigation of robustness and rejection accuracy . . . 56
5.2.1 Different datasets . . . 57
5.2.2 Experimental setup . . . 58
5.2.3 Results . . . 60
5.2.4 Discussion . . . 61
5.3 Deep dive into Gaussian prototypical networks . . . 63
5.3.1 Experimental setup . . . 63
5.3.2 Results . . . 65
5.3.3 Discussion of deep dive . . . 65
5.4 Chapter discussion . . . 65

6 Empirical analysis of Mahalanobis prototypical networks 68
6.1 Experimental setup . . . 68
6.2 Results . . . 70
6.3 Chapter discussion . . . 72

7 Thesis discussion 74
7.0.1 Future research . . . 77

8 Conclusion 79

A Hyperparameter settings 81
A.1 Deep metric learning . . . 81
A.2 Initialisation learning . . . 82
A.3 Model based learning . . . 83
A.4 Data augmentation . . . 83

B Backbone architectures 84
B.1 Research of robustness to quality shifts and image size . . . 84

B.2 The ResNet architecture 84
B.3 The eResNet architecture 85

C Additional experiments **86**
C.1 Looking at the representation space with MNIST 86
C.2 Using different loss functions to decrease intra-class variance and increase inter-class variance 89
C.3 Fine-tuning with the support set 91
C.4 Different settings for the softplus function 92
C.5 Using self-attention to find task specific features 94

Bibliography **99**

List of Figures

2.1 This circle flow explains the journey that the logo recognition model follows. 6

2.2 This figure shows the set of all possible queries Ω for which a model, prepared for task t_j, should accept only queries Q_{t_j} and reject $\forall \Omega \notin Q_{tj}$. . 8

3.1 Structure that represents a classifier with three classes that maps $\hat{y} = f_\theta(x)$ for which a darker green colour means a higher probability for that class. 12

3.2 A neural network structure with two hidden layers and four output nodes $(o_1, ..., o_4)$. 13

3.3 An example of a supervised dataset that can be used to train a neural network classifier. 14

3.4 Structure that shows the iterative forward- and backward propagation process to update parameters in a neural network classifier with three classes. 15

3.5 Example of a convolution and pooling operation on an 11 x 11 input grid structure. 16

3.6 VGG-16 CNN classification architecture with convolutional (conv), pooling (pool), and fully connected (fc) layers. 18

3.7 This shows one block of the ResNet architecture. Multiples of these blocks are stacked to create a ResNet CNN. 19

3.8 A CNN classifier is initialised to learn task t_2 and can be used to apply transfer learning to learn other tasks. 21

3.9 This graph visualises how a trained classifier on task t_2 can be used to apply transfer learning on different tasks. 21

3.10 Shows the distribution of training and testing datasets for meta-learning. In this image, the training and testing sets reflect one-shot, three-way learning. In every episode, one sample of the corresponding class is sampled for the query. This can be scaled up, which resembles having a larger batch size in standard classification training. 24

3.11 Structure showing an iterative forward and backward propagation process to update parameters in a neural network classifier that uses episodic training. 25

3.12 This Figure shows an example of k-NN classification with k=3, a support set with two classes with for both five examples each and one query input. 26

3.13 This picture shows intuitively how initialisation optimisation works. An initialisation is found that is close to all tasks in T, such that with only a few samples a new task can be learned. 27

4.1 Two characters from each of eight alphabets. 34

4.2 The image on the left is a classic picture that comes from the logo detection datasets that we use. We extract the ROIs of the logos on the left and present them on the right. We use the extracted ROIs for logo classification. 36
4.3 This graph shows the class distribution of the meta-train small logo dataset. The dotted line represents the median and the minimum number of samples in a class is 10. 37
4.4 The standard backbone with stride 1 for the convolutions and without a softmax layer. 38
4.5 A two-dimensional representation space of Prototypical networks with a five-shot, three-way learning task. 40
4.6 One episodic training iteration of the MAML model. 46
4.7 One episodic iteration for the Reptile model. 47
4.8 Example of model-based learning with an RNN. 48
4.9 Learning curve of Gaussian prototypical networks on a five-shot, five-way learning task. Trained on the small logo dataset. 49
4.10 Learning curve of Gaussian prototypical networks on a five-shot, 20-way learning task. Trained on the small logo-dataset. 49
4.11 Learning curve of Reptile on a five-shot, five-way learning task. Trained on the small logo dataset. 49
4.12 Learning curve of Reptile on a five-shot, 20-way learning task. Trained on the small logo-dataset. 49

5.1 Three examples of a possible logo query. 55
5.2 The decision boundary for two neural network approaches. The left image represents deep metric learning and the right image represent classification neural networks with a softmax output layer. The spaces with colour have high confidence for that colour class. The white spaces represent low confidence for every class. 57
5.3 One iteration of producing positive and negative samples for the ROC curve. 60
5.4 ROC curve of positive and negative samples collected from the mini-ImageNet dataset. 61
5.5 ROC curve of positive and negative samples collected from the small logo dataset. 61
5.6 Prototypical networks two-dimensional representation space with a four-way learning task (i.e., four prototypes) and probabilistic decision boundaries. 62

6.1 Two ResNet-17 Mahalanobis prototypical network models, that are trained on the validation meta-train set with 35- and 70-way tasks, are tested on different sets of five-shot learning tasks from the validation meta-test set. 71
6.2 Two ResNet-17 Mahalanobis prototypical network models, that are trained on the meta-train set and validation meta-train set with sets of five-shot, 35-way learning tasks, and tested on the meta-test set on different sets of learning tasks. 71

C.1 The three different representation spaces of backbone one, two, and three from 1,000 randomly sampled MNIST test set samples. 87

C.2 Representation space of a network trained on MNIST with the joint supervision of softmax and center loss. Different λ's are used to show the effect, and the white dots represent c_{y_i}. Figure from Wen et al.. 89
C.3 Architecture of masked covariance function with Prototypical networks. . 96

List of Tables

3.1 This table shows a comparison of formulas between normal neural network classifier training and episodic meta-learning training. 24
3.2 Statistics of available datasets to train logo detection models. The region of interest can be used to extract supervised logo samples. 30
3.3 Results of popular meta-learning techniques with one-shot and five-shot, five-way learning task classification accuracies on the mini-ImageNet dataset. Some scores include a 95% confidence interval. Note that more techniques are published constantly. Thus, this table does not include all techniques but is a representation of the most popular techniques of the past four years. 30
3.4 Comparative presentation of our findings between the three different meta-learning methods. 32

4.1 This table shows statistics of standard meta-learning datasets and a derived dataset by us, which is called the small logo dataset. The $*$ means that we have used the median as metric. 36
4.2 Few-shot classification accuracies of the models on the Omniglot dataset. The confidence interval is calculated by sampling 1000 tasks from the meta-test set. The bold cells indicate the best performing models within 95% confidence interval in that column. 50
4.3 Few-shot classification accuracies of the models on the small logo dataset. The confidence interval is calculated by sampling 500 tasks from the meta-test set. The bold cells indicate the best performing models within 95% confidence interval in that column. 50
4.4 Comparative findings on model scale between various models. 52

5.1 Statistics of standard meta-learning datasets and a derived dataset by us, which are called the small- and large logo dataset. The $*$ means that we have used the median as metric. 56
5.2 Datasets for robustness to quality shifts. Low/high and high datasets refer to, respectively, meta-train and meta-test datasets. The median number of pixels specifies the quality of the dataset. 58
5.3 Robustness results from Gaussian prototypical networks. Every average accuracy and confidence interval is computed from a set of 500 learning tasks. 60
5.4 Robustness results from Reptile. Every average accuracy and confidence interval is computed from a set of 500 learning tasks. 60
5.5 Results on the set of five-shot, 20-way learning tasks from the small logo dataset with and without data augmentation. 65

5.6 Comparative findings on model scale between Gaussian prototypical networks and Reptile. 67

6.1 The statistics of the training, validation and test set of the large logo dataset. Val is an abbreviation for validation. 68
6.2 Average accuracy results on the validation meta-test set with different backone architectures. 71
6.3 Results of the ResNet-17 Mahalanobis prototypical networks on the large logo dataset. The left column shows the final results on the meta-test set. The middle and right column are models trained only in the validation meta-train set and evaluated on the validation meta-test set. 72

A.1 Hyperparameter settings of deep metric learning models used throughout the research. 81
A.2 Hyperparameter settings for MAML that are used throughout the research. 82
A.3 Hyperparameter settings of Reptile used throughout the research. Train shots refer to the number of samples per support set class. 82

B.1 Architectural design of three different models, and the standard backbone model. 84
B.2 Architecture of the ResNet model that we used. 85

C.1 Accury results of the different loss function configurations. Every configuration is trained with 100 epochs. 91
C.2 Fine-tuning results on the small logo dataset with prototypical networks and standard backbone. 92
C.3 Different softplus function configuration experiment results with Mahalanobis prototypical networks. The confidence intervals are computed from 500 tasks from the small logo dataset. 93
C.4 Few-shot classification accuracies from the different Prototypical network models. The bold cells indicate the best performing model(s) within 95% confidence interval in that column. 97

Chapter 1

Introduction

Suppose you are attending a conference and you are there to acquire more information about, for example, a boat or a car you want to buy, or you are interested to know more about a certain company. Normally, you would acquire this information by collecting brochures or business cards or trying to remember as much as possible. However, what if you only need to take out your mobile phone and take a picture of the company's logo. After that, the contact and general information of the company is stored on your device. There is no hassle of collecting an extensive number of brochures and business cards in a bag or the need to remember everything.

This idea is a motivation for Mobiquity, an innovative digital consultancy agency, to create a mobile application that could help attendees make their lives easier without placing additional objects at a conference, such as QR codes. To do this, one first needs to train a deep learning (DL) model to classify a picture to a set of predefined logos from companies that are present at a conference. Training such models requires a large amount of data. However, Mobiquity is willing to collect only five logo samples per attended company to make the application scalable.

Since AlexNet won the ImageNet competition in 2012 [4], many DL models have been created that localise and/or classify logos from images [5–9]. The successes of these DL models can be attributed to an increase in computation power and data. As such, these models use on average approximately 400 to 1,000 samples per logo class [5–9]. For the mobile application, it implies that for a conference with 35 attending companies, we need to collect and annotate roughly 14,000-350,000 samples, which is far more than the 5 x 35 samples Mobiquity wants to collect. Other conferences have different sponsors and hence different logos, which impacts scalability and costs even more.

One solution would be to use logo datasets that have been collected by Su et al., Tzk et al., Joly and Buisson and Romberg et al.. Despite the fact that these datasets do not specifically match the logos of the organisations at a conference, they could contain information that can help close the gap of five samples to the required 14,000-350,000 samples.

We conduct research in the field of deep learning to create a model that is able to cope with these challenges. We also aim to find a DL solution for a practical problem and it is not sufficient to consider only accuracy:

- The model needs to be *flexible*; Mobiquity should be able to use it for different conferences without too much preparation (see Chapter 2).
- The model should be *reliable*; it should keep engagement with users by having sufficient accuracy, it should have an acceptable rejection accuracy for in- and out-domain distribution unknown-class samples and accuracy should not drop substantially when the number of logo classes at a conference deviate from other conferences (see Chapter 2).

To achieve this, the main research question is defined as follows:

Is it possible to build a flexible and reliable deep learning logo recognition model that is able to classify logo classes from pictures with only five training samples per class?

A few directions in DL are specialised in dealing with limited training data, such as transfer learning and meta-learning. Transfer learning uses a pre-trained DL network and re-trains it on a new task with a smaller dataset, and meta-learning focuses on models that are able to learn different but related tasks with only a few (i.e. 1-10) samples per class.

We primarily focus on solving the problem with techniques from the meta-learning domain. However, most of these techniques are not developed with a practical use case in mind but are trained and tested on small domain-specific datasets. Hence, the focus of this thesis is on which techniques work best, and more importantly, which techniques are appropriate considering *flexibility* and *reliability*.

1.1 Main contributions

The main contributions of this thesis are:

- Most logo datasets are created for detection problems and are not directly applicable for logo classification and meta-learning. We therefore introduce a *small-* and *large logo* classification dataset specifically for meta-learning in, respectively, Chapters 4 and 5.

- We train the *Gaussian prototypical network* model on the small logo dataset and it achieves a significantly higher accuracy than other meta-learning techniques on different sets of learning tasks, such as Prototypical networks, Relation networks, MAML, and Reptile. There are surprisingly no other accuracy results of Gaussian prototypical networks available other than the performance on the Omniglot dataset in the original paper. We present these results in Chapter 4.

- Empirical research of Gaussian prototypical networks shows that the increase in performance over prototypical network is contributed to the use of the Mahalanobis distance metric and not by adjusting the prototypes with estimated inverse covariance matrices. We therefore introduce a new, simpler model that only uses the Mahalanobis distance metric in Chapter 5, which we call *Mahalanobis prototypical networks*.

- Current techniques in the meta-learning domain focus primarily on accuracy, whereas for AI-driven applications, multiple factors play an important role: (i) robustness of accuracy to different tasks and (ii) robustness to in- and out-domain distribution unknown-class samples. We show for (i) that *deep metric learning* approaches are more robust to tasks with more classes than *initialisation learning* approaches in Chapter 4 and (ii) that Gaussian prototypical network is more robust to in- and out-domain distribution unknown-class samples than Reptile in Chapter 5.

- In the published paper of Prototypical networks, the accuracy results are obtained by learning on sets of higher K-way learning tasks than on the set of tasks the model is tested on, because Snell et al. show that doing this increases accuracy. We have tried the procedure in Chapter 6 with Gaussian prototypical networks but do not see an increase in accuracy.

- Finn and Levine have shown that MAML is better able to handle domain shifts between meta-training and meta-testing compared to recurrent meta-learners, such as SNAIL, because of within-task learning. We therefore hypothesise that Reptile is more robust to a difference in the quality of samples (i.e., number of pixels in an image) between lower qualitative meta-training samples and higher qualitative meta-testing samples than Gaussian prototypical networks. However, we see in Chapter 5 that Reptile is not more robust than Gaussian prototypical networks.

- Most deep metric learning approaches use the support classes independently to decide if a query (i.e. test sample) belongs to a certain class, which means that for every task the full set of representations are used during test time. However, depending on the task, certain dimensions (i.e. features) from the representation are more important than others. We use a self-attention block that takes the complete support set at once to learn which dimensions are most relevant for the task. We use this block together with Prototypical networks, and show that it has comparable performance to Mahalanobis prototypical networks on the five-shot, five-way learning task in Appendix C.5.

Chapter 2

Problem definition and research setup

In this chapter, we investigate properties of the research problem and give definitions that are used throughout the thesis. The logo recognition model should satisfy requirements for certain quality metrics to be called a success. We find these quality metrics by constructing a model journey and we use them to make the research quantifiable by setting certain requirements per quality metric. Finally, we explain how we setup the research to find an answer to the main research question.

2.1 Definitions and model journey

Our goal is to create a logo recognition model that is able to learn to classify logo classes from a conference with five training samples per class. For example, we teach a logo recognition model to classify 35 distinct logo classes for a conference. We define this as a learning task t, which is a set that includes the 35 logo classes the model should learn to classify. There are, however, many different conferences with different numbers of logo classes and we define a set T that includes all the conferences that exist. Now, a specific task from set T we define as t_j. Further, we can extract subsets from T, such as a set with learning tasks that have K classes per task. We define this as T^K and a specific task from this set as t_j^K. Furthermore, the classes in a task t_j and t_l can be the same for some of the classes, because different conferences can have the same organisations present. We therefore define a set L that consist of all distinct classes in T.

We define a model journey that gives insight to what quality metrics we need to focus on. In Figure 2.1, we show a model journey for which in phase (i) a generic model is

defined that should be able to learn to classify new logo classes with only five training samples per class. In other words, a generic logo recognition model that is able to learn every task $t_j \in T$ with only five training samples per logo class. In phase (ii), the generic model is prepared for the new task t_j with the five training samples that are collected for every logo class; in phase (iii), the prepared model is deployed in the mobile application and used by people at conference j. After the usage phase, the generic model can be improved with new data that are collected either during the conference from phase (iii) or via other sources. From now on, we refer to the model journey as circle flow.

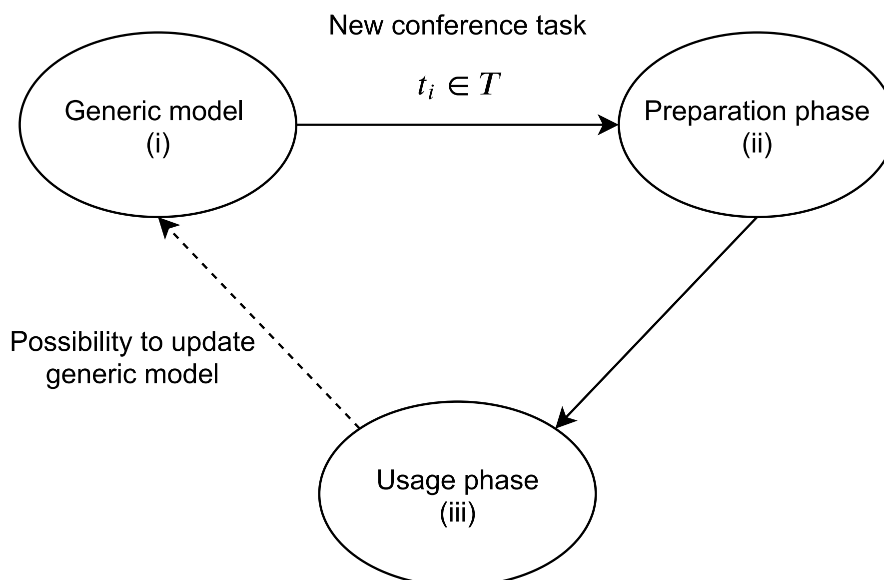


Figure 2.1: This circle flow explains the journey that the logo recognition model follows.

There are a few additional definitions in phases (ii) and (iii). These are as follows for the preparation phase:

1. Prior to a task t_j, a set of samples of logos per company are collected to prepare the generic model on task t_j; we call this the *support* set S of t_j, which include in total five training samples per logo class.
2. After the model is prepared, it is deployed and usable within the mobile application.
    - The specificity of how the model should be deployed and interact with the mobile application is out of scope of this research.

For the usage phase, we can define the following definitions:

1. An attendee takes a picture with his or her phone that can either be a logo or not; we call this picture the *query*.
    - It is assumed that the logo is the largest object in the picture.
    - Hence, the attendee performs logo detection and the model is requiring only to perform logo classification.
2. The query is forwarded to the prepared model and is rejected or accepted to a specific class in task t_j.
3. When the query is accepted for a specific class in task t_j, information from a database is sent back to the mobile application.
    - How the interaction works with the database and mobile application is out of scope of this research.

Furthermore, we define three different groups of queries: (i) in-domain distribution known-class samples, (ii) in-domain distribution unknown-class samples, and (iii) out-domain distribution unknown-class samples. Samples from (i) are queries from classes of task t_j the generic model is prepared for; we name this set of queries Q_{t_j}. The samples from (ii) are queries from logo classes the generic model is not prepared for. For example, samples from the set of classes $L \setminus t_j$; we name this set of queries $R_{L\setminus t_j}$. The samples from (iii) are queries from non-logo classes, such as a picture of a person; we name this the out-domain rejection (ODR) query set. Moreover, when a generic model is prepared for task t_j it should accept all queries from the set Q_{t_j} and reject all queries from the sets $R_{L\setminus t_j}$ and ODR. We show this in a Venn diagram in Figure 2.2.

In this thesis, we talk about accuracy when we compute the performance of a model on task t_j with queries from set Q_{t_j}. We assign a query to a class from t_j with the lowest distance or highest probability. Hence, we only measure how well the model is able to classify with this metric. We incorporate rejection when we talk about a threshold accuracy. Now the query is assigned to a class only if it is within a defined probability or distance threshold. This means that accuracy is always greater or equal than threshold accuracy. Next, we talk about the rejection accuracy when we compute the fraction of how well a model prepared for task t_j is able to reject queries from $R_{L\setminus t_j}$ or ODR.

2.2 Quality metrics and requirements

One thing to note is that many models can fit as a generic model in the circle flow of Figure 2.1, such as a vanilla DL model that is not trained on data yet. However, putting

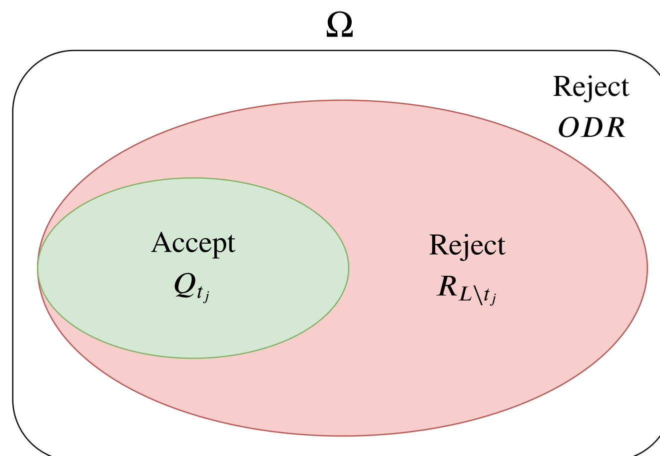


FIGURE 2.2: This figure shows the set of all possible queries Ω for which a model, prepared for task t_j, should accept only queries Q_{t_j} and reject $\forall\Omega \notin Q_{tj}$.

such models in there requires it to prepare on five samples per class for a task t_j. As discussed in the introduction, we know that five samples per class is not sufficient to train a DL logo recognition model and to expect high performance.

Quality metrics

To find the right generic model, we define that it should have four quality metrics: sufficient accuracy, high robustness, flexibility, and high rejection accuracy. The generic model must have sufficient accuracy after it is prepared on a task t_j. Otherwise, it does not solve the user's problem. Moreover, we do not want the generic model to have high accuracy for task t_j and low accuracy for task t_l. Hence, the accuracy variance across tasks should be low; we call this robustness. Furthermore, the generic model must be flexible in terms of it being easy to prepare on any task from T and time to prepare it should not fluctuate much, because it is then easier to schedule in preparation time and make the process scalable. The rejection accuracy is important, because people could take a picture of a logo or non-logo that is not in task t_j for which the generic model is prepared for. These queries should be rejected and not be assigned to any class in t_j. Finally, we talk about a *reliable* model if the generic model has sufficient accuracy, high robustness, and high rejection accuracy.

Requirements

In agreement with Mobiquity, we create requirements for the defined quality metrics to make this research quantifiable. We create these requirements under the assumption

that the generic model will be mostly used for conferences with 35 logo classes, because these conferences are most common. Hence, for conferences with 35 distinct logo classes, a generic model should have at least 90% threshold accuracy, 80% rejection accuracy, a high robustness compared to the selection of selected models, and a non-expert[1] should be able to prepare the generic model on any task within 6 hours. This means that we specifically focus on the set of tasks with 35 logo classes or, in other words, at set T^{35} to compute the threshold and rejection accuracy. We also focus on sets of tasks in the range of $K \in \{5, ..., 95\}$ for robustness.

We define quantitative performance metrics for every of the four quality metrics. We use these performance metrics to evaluate if the generic model is able to match the defined requirements. We present them here:

- Accuracy
  - The *average accuracy* on a set of tasks T^K.
  - The *threshold accuracy* is the average accuracy on a set of tasks T^K with the use of a probability or distance threshold.
- Robustness[2]
  - Variance in accuracies from tasks in set T^K. We call this *within-robustness*.
  - Variance in average accuracies between different sets of tasks. For example, variance between the average accuracy of sets T^K and T^{K+1}. We call this *between-robustness*.
- Flexibility
  - The preparation time of tasks in set T^K. We call this *within-flexibility*.
  - The preparation time between sets of tasks. For example, preparation time between sets T^K and T^{K+1}. We call this *between-flexibility*.
- Rejection accuracy
  - Rejection accuracy on set $R_{L \setminus t_j}$, in other words, in-domain distribution unknown-class samples.
  - Rejection accuracy on set ODR, in other words, out-domain distribution unknown-class samples.

We use the average accuracy because it is often used in the DL and meta-learning domain. Hence, we can compare our implementation and models to published papers.

[1]We define a non-expert as someone that has no machine learning and DL experience.
[2]Note that high robustness is equal to low variance.

2.3 Research setup

In this section, we explain how we find or create a generic model that has the described quality metrics and has the potential to match the requirements. First, we define a few sub-questions that we need to answer the main research questions. Finally, we make a research setup wherein we explain how we get answers to the sub-questions.

2.3.1 Main- and sub-questions

To answer the main question, *'Is it possible to build a flexible and reliable "generic" deep learning logo recognition model that is able to classify logo classes from pictures with only five training samples per class?'*, we need to answer the following questions:

1. What techniques fit in the circle flow and have the potential to be reliable and flexible? (Chapter 3)
   (a) Which of these techniques is most flexible (Chapter 3), has the best accuracy (Chapter 4), is most robust (Chapter 4), and has the best rejection accuracy? (Chapter 5)
2. How should we collect data to make a reliable generic model? (Chapter 8)
3. How much data do we need to collect to match the requirements of reliability for the generic model? (Chapter 8)

2.3.2 Research setup

To find potential techniques that can be reliable and flexible, we first conducted background and literature research in Chapter 3. We used the techniques that we found there to acquire performance metrics with a dataset that we constructed in Chapter 4 and Chapter 5. Finally, in Chapter 6, we conducted experiments with the most potential technique to determine if or how we are able to meet the defined requirements.

Chapter 3

Background and literature research

Over the years, to achieve good performance on small datasets, many strategies on how DL can handle small datasets have been developed and researched. In this chapter, we aim to provide the reader with an overview of this work as well as discuss if techniques from that work are able to learn from only five samples per class. We conclude the chapter with a selection of some of the work that seems to be viable with respect to being flexible and reliable.

3.1 Neural networks

We focus on using DL models in this research, which are large neural networks, and first aim to provide the reader with background information on what a neural network is, how we train these models, and what for different neural networks exist.

The goal of a neural network is to approximate some function f^*. For example, in the spirit of this research, a classifier $y = f^*(x)$ maps an input x (query) to a logo class y. A neural network function tries to approximate the function f^* with $\hat{y} = f_\theta(x)$ in Figure 3.1, for which θ are learnable parameters that are tuned to get the best function approximation.

A feed forward neural network architecture can be described as a directed acyclic graph and a general structure is shown in Figure 3.2. This structure is an example that can be used in Figure 3.1 as the sub-function $g_\theta(x)$ [15]. We can represent $g_\theta(x)$ by a composition of functions, $g(x) = g^{(3)}(g^{(2)}(g^{(1)}(x)))$ [15]. These chained functions represent the different layers in Figure 3.2, as such, layer 1 is $g^{(1)}$ (first hidden layer),

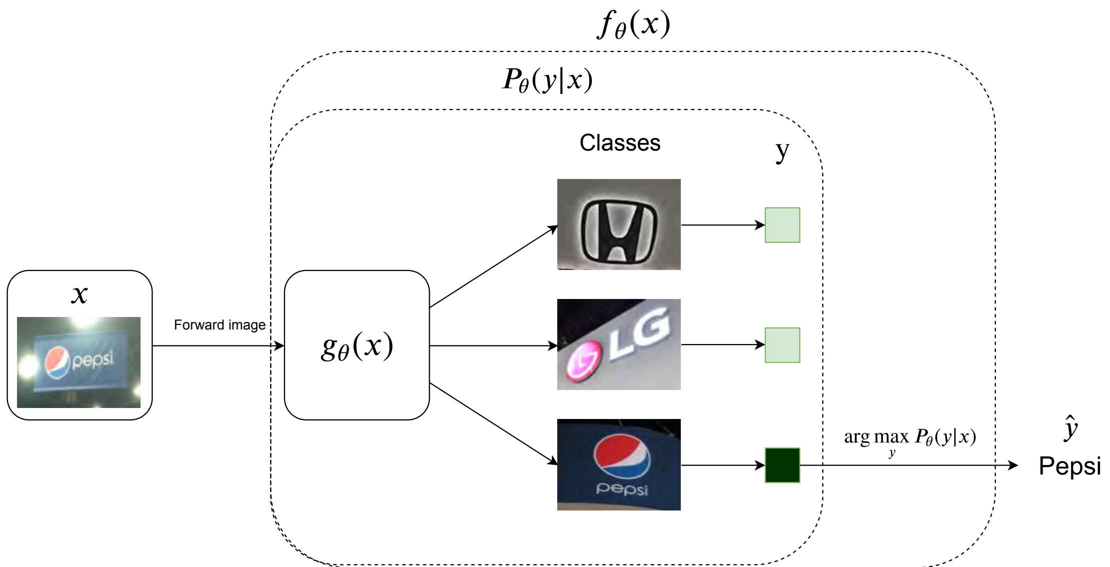


FIGURE 3.1: Structure that represents a classifier with three classes that maps $\hat{y} = f_\theta(x)$ for which a darker green colour means a higher probability for that class.

the second layer is $g^{(2)}$, etc. We talk about DL models if we have multiple layers in a neural network.

For the function $g_\theta(x)$, such as in Figure 3.1, the values from the output layer are used to compute the probability that an input belongs to a class, which we denote as random variable $\mathbf{y}$. In other words, $\mathbf{y}$ is a categorical distribution over the different classes. Hence, $P_\theta(\mathbf{y} = y|\mathbf{x} = x) = g_\theta(x)$, where P_θ is parameterised by the neural network $g_\theta(x)$. This probability distribution over classes is usually computed with the softmax function when there are more than two classes and when inputs belong to independent classes. An input x can then be assigned to a class $y \in \mathbf{y}$ that has the highest probability or, in probabilistic notation, $\hat{y} = \arg\max_y P_\theta(\mathbf{y} = y|\mathbf{x} = x)$ [1]. We show in Equation 3.1 how the softmax function works with the outputs from the neural network structure of Figure 3.2.

$$P_\theta(\mathbf{y} = y|\mathbf{x} = x) = \frac{e^{o_y(x)}}{\sum_{i=1}^{c=4} e^{o_i(x)}} \tag{3.1}$$

In deep learning, we have a task t_j for which we want to find a function that is able to map $\hat{y} = f_\theta(x)$. In line with this research, the task t_j is defined as constructing a neural network classifier that is able to map inputs to a specific number of logo classes, such as in Figure 3.1, where there are three logo classes. To find the function $f_{\theta,t_j}(x)$ that

[1]We will be using $P(\mathbf{y} = y|\mathbf{x} = x) = P(y|x)$ from now on for brevity.

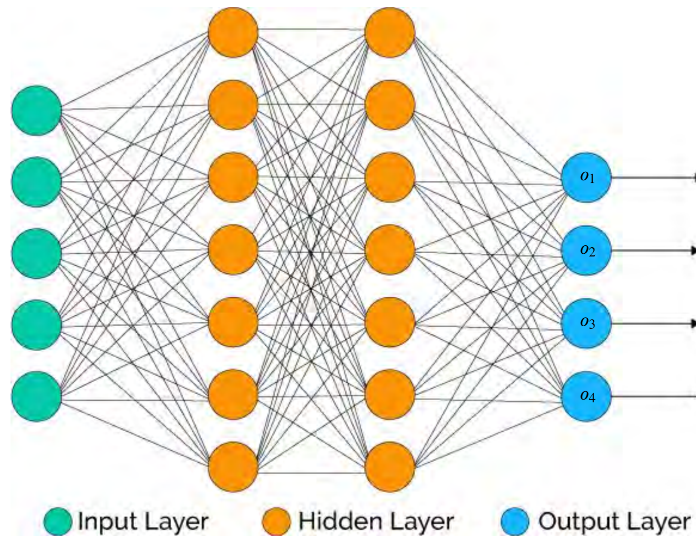


FIGURE 3.2: A neural network structure with two hidden layers and four output nodes ($o_1, ..., o_4$).

approximates $f^*_{t_j}(x)$, we use supervised data from task t_j to learn the correct parameters of the neural network function $g_\theta(x)$.

Data and loss function

Supervised data for task t_j is a set of logo pictures and corresponding logo classes, which we define as $d_{t_j} = \{(x_i, y_i), i = 1, ..., N\}$ with $y_i \in t_j \; \forall i$. For example, for the task in Figure 3.1 we have logo samples for three classes, Honda, LG and Pepsi. For every of these three classes we have a number of different samples, as we show in Figure 3.3. With these supervised data, we can evaluate how well $g_\theta(x)$ is able to map the input images x_i to the correct labels y_i. We do this with a function that is named the *loss function*. In classification, the cross-entropy loss function is often used, and we present this function in Equation 3.2. We maximise it with respect to the parameters θ to find a function that maps the inputs x_i to classes y_i as best as possible. This results in Equation 3.3a, where we maximise the loss with respect to the parameters to find the optimal parameters θ^*.

$$L(\theta, d_{t_j}) = \sum_{(x,y)\in d_{t_j}} \log(P_\theta(y|x)) \tag{3.2}$$

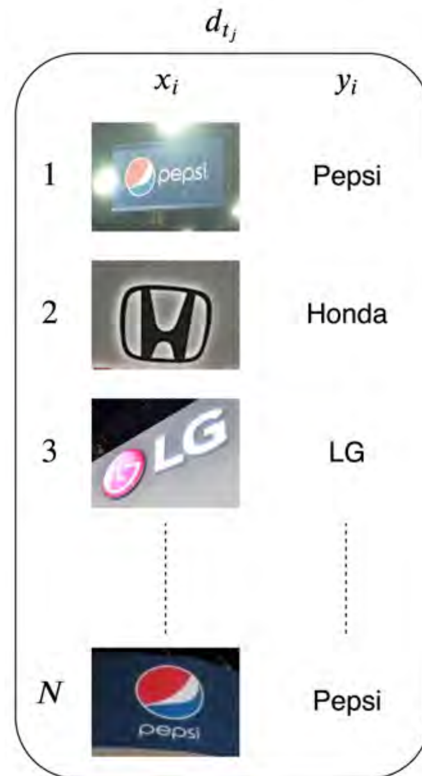


FIGURE 3.3: An example of a supervised dataset that can be used to train a neural network classifier.

$$\theta^* = \arg\max_{\theta} \sum_{(x,y)\in d_{t_j}} \log(P_\theta(y|x)) \tag{3.3a}$$

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{q\sim d_{t_j}} \left[\sum_{(x,y)\in q} \log(P_\theta(y|x)) \right] \tag{3.3b}$$

Training a DL model

In deep learning, an algorithm named backpropagation is used to find the parameters that maximise the function in Equation 3.3a [16]. Commonly, the parameters of deep learning models are learned with this algorithm using GPUs. Despite the fact that there has been an increase in computation power and memory capacity on these GPUs, we often do not have enough memory to store the complete dataset. Therefore, batches from dataset d_{t_j} are taken to use backpropagation and learn the parameters. We do

this by sampling a batch size b from d_{t_j}, which we call the query batch q. This process changes Equation 3.3a slightly into Equation 3.3b, which intuitively can be read as a double summation over batches and samples. In Figure 3.4 a process is depicted that shows how learning the parameters for $g_\theta(x)$ is executed with batch learning.

Before training a neural network, we can set many different parameters. Such as how many layers, how many nodes per layer, the size of the batch, and a learning rate that specifies with what amount the parameters θ are updated every iteration. These parameters are usually set before training and are called the hyperparameters of the neural network model.

Performance measure

To make sure the function is not overfitting on the training data, the data d_{t_j} are split into a train- and test set. We use the train set to update the parameters and we use the test set to evaluate if the model is able to generalise to unseen data.

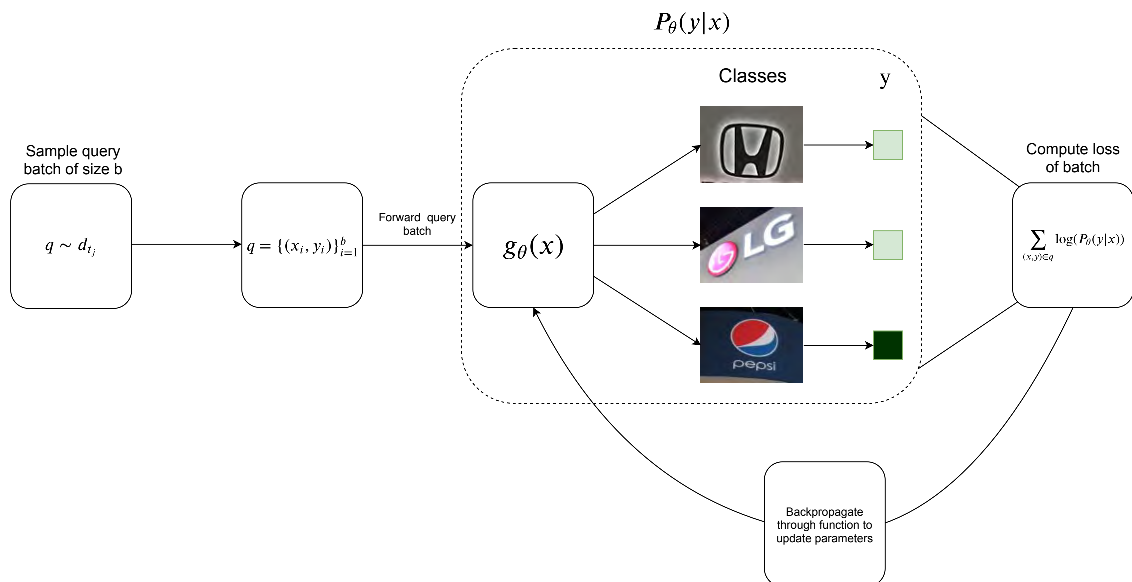


FIGURE 3.4: Structure that shows the iterative forward- and backward propagation process to update parameters in a neural network classifier with three classes.

3.1.1 Convolutional neural networks

Figure 3.2 represents a standard feed forward neural network with fully connected layers. This is not the only neural network structure but there exist many different structures, such as convolutional and recurrent neural networks. Choosing the right structure depends on the task at hand. In the case of image classification, convolutional neural

networks (CNNs) have been proven to be very successful. Such as AlexNet, which was able to get ground breaking results in the 2012 ImageNet competition, and more recently, the ResNet structure, which was able to get better performance with many convolutional layers [4, 17]. We use CNNs throughout the thesis so we explain how they work.

Images have a 2D grid structure with strong local dependencies; pixels that are spatially close often have the same pixel values [18]. Local simple features can be detected with different configurations of local strong dependencies, such as edges, corners, etc. Spatial objects can be recognised with a combination of these simple features, such as a window that is constructed out of two horizontal and vertical edges. Convolutional neural networks are designed to work with these grid-structured inputs [18]. The advantage of a domain-designed network is that significantly less data are needed to get the same performance compared to a standard feed forward neural network [19].

Convolutions

A standard convolutional neural network usually consists of convolutions, pooling schemes, and non-linear functions; the first two operations are depicted in Figure 3.5. A convolution extracts local features from an input by learning a function (filter) dependent on θ. This is done by shifting the filter over the grid structure and hence the spatial information of features stays intact. Multiple convolutions are learned to extract multiple different features. The output of a convolution is a feature map and we call a collection of these feature maps a representation.

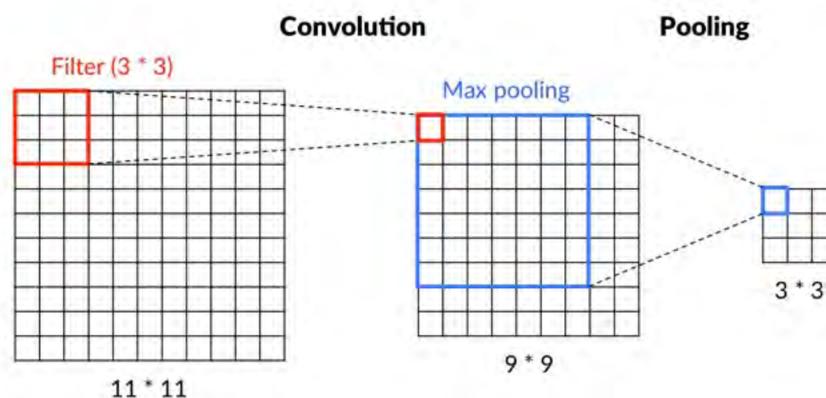


FIGURE 3.5: Example of a convolution and pooling operation on an 11 x 11 input grid structure.

Pooling layers

The goal of a neural network classifier is to map an input to an output class. An input of class Nike can be seen as a distribution, because there exist many variations of images that contain the Nike logo. The goal is thus to correctly classify every image from that distribution to the right class. Neural networks do this by learning to extract representations for a class, such that the representations between different classes are linearly separable [20]. The neural network should aim to be invariant to certain deformations to minimise the variation of the representations [20]. This depends heavily on the task, but for classifying logos from images, one can think about invariance to rotation, different lighting, and translation.

The transformation of a convolved input is mostly equivariant, which means that spatial information is the same or, precisely, $\Phi(Tx) = T\Phi(x)$, where Φ is the computation of the convolution and T is the translation [21]. The consequence is that representations are not invariant to translations and thus contradicts our previous statement that representations should be invariant. Pooling is introduced to help make CNNs more representation invariant to translations and other transformations [19]. It does this by reducing the spatial resolution of the feature maps by only forwarding the maximum input value of a specified region. An example of a pooling operation is max-pooling, which we show in Figure 3.5.

ReLU activation function

The third and last operation in a standard convolution layer is the non-linear activation function. The same as in a standard fully connected neural networks, the outputs of the feature maps are forwarded in a differential non-linear function. Nowadays, the rectified linear unit (ReLU) has been the most widely used activation function [22].

3.1.1.1 Convolutional neural network structures

Different CNN architectures exist that combine convolutions, pooling, and fully connected layers. A well-known architecture is the VGG16 CNN, depicted in Figure 3.6 [23]. In this figure, we see that nearly all layers are constructed with convolutions and pooling functions. The last four layers are fully connected (dense layers). Intuitively, we can think that the convolution layers construct a function that extracts representations from the input. The dense layers use the representations to perform the classification.

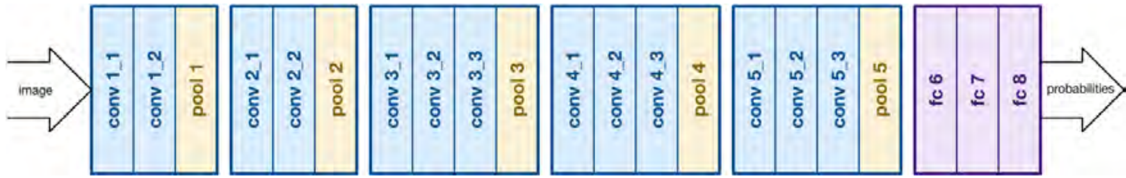


FIGURE 3.6: VGG-16 CNN classification architecture with convolutional (conv), pooling (pool), and fully connected (fc) layers.

Recently, a lot of work has focused on different architectural CNN implementations [17, 24, 25]. The trend seems to be to make very deep networks and to reduce the use of pooling operations. Springenberg et al. have shown that pooling operations are not necessary, and without them, they reach the same or even better performance than CNNs that do use pooling layers. Springenberg et al. also mention that large CNNs with enough data are able to learn the invariances without the need for pooling layers.

The ResNet model

A problem with CNNs that are based on the VGG16 architecture is that it is hard to know a priori how long the network should be to learn the function $f^*(x)$. Another problem is that very deep networks that are based on the VGG16 structure are unstable and hard to train, which is called the *degradation* problem by He et al.. Their solution is to make a skip connection, such that $g(x) = h(x) + x$ (see Figure 3.7 for a detailed picture). Multiples of these $g(x)$ CNN blocks can be attached sequentially, with which you can make very deep CNNs. The network can easily turn off blocks by using only the skip connection and thus the network can automatically find the right number of layers for a task t_j.

The ResNet model only has two pooling layers at the beginning of the network. All the other layers consist of convolutions, ReLU activation functions, and batch normalisation. Batch normalisation normalises layer inputs, which results into faster convergence, acts as a regulariser, and makes the hyperparameters more robust [27]. The final ResNet layer is a global average convolution layer.

We can try to use both VGG16 or ResNet in the flow circle from Figure 2.1 as a generic model. We then learn every task t_j from scratch. The problem is that five training samples per class for a task t_j are not enough to get good performance, as we have discussed in the introduction.

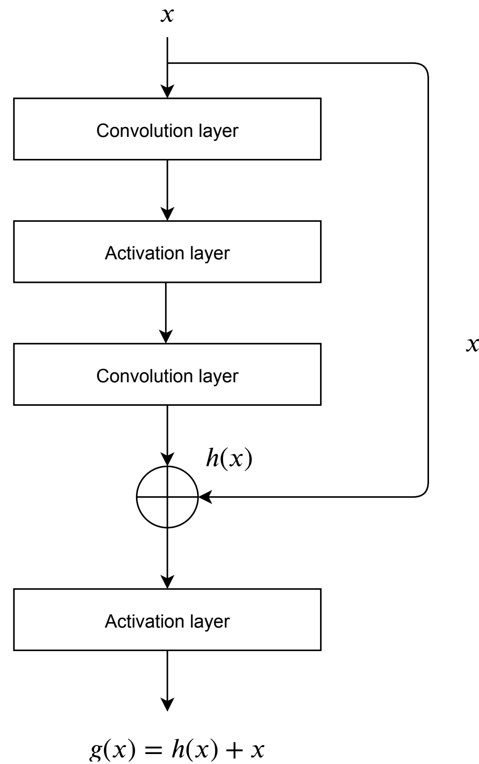


Figure 3.7: This shows one block of the ResNet architecture. Multiples of these blocks are stacked to create a ResNet CNN.

3.2 Transfer learning

Not having enough supervised data to train DL classifiers is a problem that occurs often, because much data is required for these complex function approximators. We can look to make DL models less complex by reducing the number of layers and neurons per layer. However, the strength of deep neural networks lies in the automatic generation of representations from input data [28]. It requires deep networks and a large number of samples to learn powerful representations. These representations need to capture the factors of variation in the input data, which is dependent on the task [15]. For example, the number of wheels attached to an object is an important factor of variation when one wants to classify between a car and a truck. It is then also important that the representation can disentangle these factors [15]. As such, the model is able to classify the input to a car or truck with these factors. Furthermore, translating or rotating a car does not impact its class definition and thus the presentations should also be insensitive (invariant) to task dependent transformations, such as translation, scale, and rotation [29].

Yosinski et al. have shown that convolutional neural networks learn to detect simple features (i.e., lower-level features) in early layers, such as edges and corners. In later

layers, a combination of these simple features detect more complex features (i.e., higher-level features), such as a window. Whenever multiple CNNs are trained on different but related tasks, it can be that early layers detect on the same simple features, such as the aforementioned edges and corners. However, the later layers are more specific to the task and so the complex features deviate more between the CNNs that are trained on different but related tasks. We call the early layers that detect the same features, and thus computes the same representations, general layers. Whereas we call the the later layers, the specific layers.

The general layers can be utilised to pre-process the images and extract useful representations. The higher, specific layers are then trained using the pre-processed representations. This is called network-based transfer learning, which we refer to as transfer learning from now on, and is formalised as follows [31]:

Definition 3.1. (Transfer learning). Information of a dataset d_{t_1} that is used to train a model on task t_1 is used to train a model with dataset d_{t_2} on a different task t_2, where $d_{t_1} \neq d_{t_2}$, and/or $t_1 \neq t_2$. In addition, in most cases, the size of d_{t_1} is much larger than d_{t_2}.

We can apply this by first creating a generic model. We do this by training (initialising) a logo CNN classifier on a logo task t_j with sufficient data to reach high accuracy. When we need to make a logo classifier for another task t_l, we then freeze most of the early layers and use the five training samples per class to prepare the generic model for task t_l. We do this, for example, by training only the one or two final layers of the classifier. This can work because the number of parameters that must be trained is significantly lower and hence, we need less data to learn task t_l. Initially, only a large supervised dataset must be collected for task t_j to create the generic model. However, is training on only task t_j good enough to apply transfer learning on all tasks in $T \setminus \{t_j\}$ with regards to reliability?

In Figure 3.8, we show an intuitive illustration of a classifier that learns task t_2 with a large dataset to get high accuracy on task t_2 [32]. Every circle represents the optimal parameters for that task. The solid lines refer to the learning of an initialisation classifier (generic model) and the dashed lines refer to transfer learning, which we show in Figure 3.9. We see in Figure 3.9 that the initialised model is able to learn new tasks, such as t_3 and t_5, with transfer learning. However, the difficulty to learn a task deviates, as can be seen by the different distances of the dashed lines. This means that for t_5, more data are needed than for t_3 to reach the optimal parameters for that task. Hence, the performance of transfer learning is dependent on which task we choose to initialise the model. Meaning that the performance of transfer learning is susceptible to high

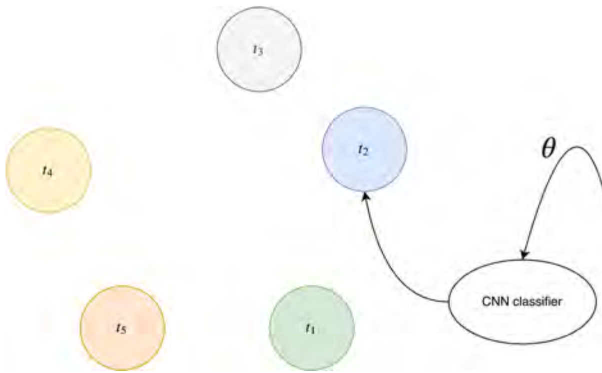


FIGURE 3.8: A CNN classifier is initialised to learn task t_2 and can be used to apply transfer learning to learn other tasks.

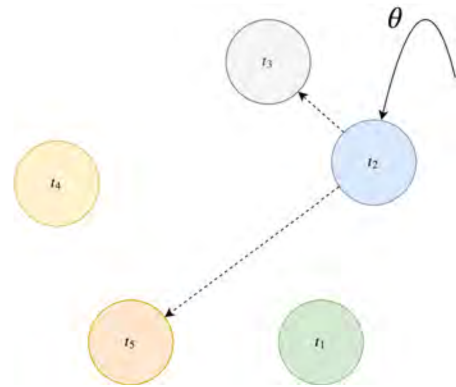


FIGURE 3.9: This graph visualises how a trained classifier on task t_2 can be used to apply transfer learning on different tasks.

variation, which is not in line with the requirement that the generic model should be robust.

This means that we need to find a different solution to make a generic model that can learn a new task with only five samples per class. Research conducted by Scott et al. compared meta-learning techniques to standard CNN classification learning and transfer learning on the ability of learning new tasks with different numbers of training samples per class. The results show that the used meta-learning techniques are superior in accuracy to transfer learning and standard CNN classification learning when samples per class are low (i.e., less than 20 samples per class). Hence, we look if these techniques are able to function as a generic model and have the potential to meet the requirements from Chapter 2.

3.3 Meta-learning

People are efficient learners. They only need a few supervised or unsupervised examples to learn new classes or skills. Fei-Fei et al. and Lake et al. hypothesise that humans learn new classes and skills quickly on the basis of past experiences. This motivated the research community and it resulted in the field of deep meta-learning, which we refer to from now on as meta-learning. We explain in this section what meta-learning is, which techniques exist in meta-learning, and how it is useful with respect to building a generic model that can learn new tasks with five training samples per class.

3.3.1 Introduction to meta-learning

In general, a good performing meta-learning model is capable of adapting or generalising to new tasks that it has never seen before during training. To do this, an initial (generic) model is trained with a dataset that contains many different tasks. After that, only a few samples (often 1-10 samples) are needed per class to prepare the initial model for a new but related task it has never seen before. We have defined a set of tasks T in Chapter 2 from which we can sample a task t_j. Together with task t_j there exist a dataset $d_{t_j} \in D$ for which the dataset d_{t_j} is a collection of supervised data that belongs to task t_j. We can now formalise meta-learning as follows:

Definition 3.2. (Meta-learning). Information of tasks $t_j \in T$ with datasets $d_{t_j} \in D$ is used to train a model on different unseen tasks $t_j^{'} \in T^{'}$ with datasets $d_{t_j}^{'} \in D^{'}$, where $T \neq T^{'}$ and $D \neq D^{'}$. In addition, generally, the size of d_{t_j} is 1-10 samples per class.

From Definition 3.2 we can see that transfer learning is a form of meta-learning when the set of tasks T and $T^{'}$ only consist of one task. This also gives away why meta-learning is better able to learn different tasks, because the initial model is trained on a set of tasks T.

In Chapter 2, we specified that we are looking for a generic model that is able to learn every new task $t_j \in T$ with only five training samples per class. We defined a task as a collection of logo classes the model should be able to classify correctly. We can see that these constraints exactly fit Definition 3.2 and will go deeper in meta-learning to see if and how we can use it for this research.

3.3.1.1 Training a meta-learning model

In meta-learning, it is called few-shot or n-shot learning when there are n training samples available per class for learning a new task and it is called K-way for the number of classes that are learned for the new task [36]. For example, the aim of this research is that we create a reliable and flexible model that can classify 35 logo classes with five training samples per class. In meta-learning terminology, we call this a five-shot, 35-way learning task

Vinyals et al. proposed a groundbreaking learning setup designed for meta-learning to train a meta-learning model that is able to generalise to a distribution of unseen but related tasks (meta-testing) with only a few samples to learn from, also called *episodic training*. The idea is to simulate meta-testing during training (meta-training), which consequently let the model generalise better to new tasks it has never seen before. Almost

every meta-learning technique has implemented this training strategy since Vinyals et al. introduced it.

We define meta-testing as if we deploy the model in the circle flow of Chapter 2. In meta-testing, we need to prepare the model for a new learning task t_j with supervised logo data from d_{t_j}. After learning task t_j, we can test with queries from d_{t_j} how well the model has learned task t_j. We do this by sampling a support set S and query set q from d_{t_j}. We use the support set S to learn task t_j and queries from q to evaluate how well the model has learned task t_j. To simulate this in meta-training, we follow the steps in Algorithm 1 to sample support set S and query batch q for an n-shot K-way learning task.

Algorithm 1 Sample S and q for an n-shot K-way learning task

Require: A set of labels L from dataset $D = \{(x_i, y_i)\}$ with $y_i \in L \, \forall i$

1: Estimate $t^K \sim T^K$ by random sampling a task with K classes (labels) by $t^K \sim L$
2: Extract $d_{t^K} = \{(x_i, y_i), i = 1, ..., N\}$ with $y_i \in \{1, ..., K\}$ and $d_{t^K} \subset D$
3: Random sample a support set S with $S \overset{n}{\sim} d_{t^K}$ and $S = \{S_1, ..., S_K\}$ for which the total number of samples in every support class $S_1, ..., S_K$ is n
4: Random sample a query batch $q \overset{m}{\sim} d_{t^K}$ with $q = \{q_1, ..., q_K\}$ for which m is the total number of samples in every query class $q_1, ..., q_K$ and $S \cap q = \emptyset$
5: **return** Support set S and query batch q

An example of sampled support sets and query batches is visible in Figure 3.10. Every row is one iteration through Algorithm 1. We sample meta-training tasks from a dataset D but sample the meta-testing samples from a dataset with different classes from $D^{'}$.

Now, the idea is to use set S to prepare a model for a task t_j^K and evaluate how well the model has learned task t_j^K with set q. We define this as one iteration or one episode. The difference with normal neural network classifier training is that we iterate through tasks and forward set S and q to the neural network model. On the contrary, for normal neural networks classifiers, we focus only on one task and forward a training batch q. This means that with episodic training a class prediction depends on both S and q, and hence $P_\theta(y|x)$ becomes $P_\theta(y|x, S)$. This also changes some formulas we defined in the neural network section, such as Equation 3.3a and Equation 3.3b. We summarise these changes between normal neural network training and episodic training in Table 3.1. We also update Figure 3.4 with Figure 3.11 to show changes that occur with episodic training. Notice that we replaced $g_\theta(x)$ with 'A model'. We do this, because, depending on which meta-learning technique is used, the model changes substantially. Finally, depending on the technique, there can be small deviations in the episodic learning algorithm.

	Normal classifier learning	Episodic learning
Sampling task	-	$t^K \sim T^K$
Data	d_{t^K}	d_{t^K}
Sample batch	$q \sim d_{t^K}$	- $S \sim d_{t^K}$ - $q \sim d_{t^K}$
Model loss	$\sum_{(x,y)\in q} \log(P_\theta(y\|x))$	$\sum_{(x,y)\in q} \log(P_\theta(y\|x,S))$
Optimisation	$\arg\max_\theta \mathbb{E}_{q\sim d_{t^K}} \left[\sum_{(x,y)\in q} \log(P_\theta(y\|x))\right]$	$\arg\max_\theta \mathbb{E}_{t^K\sim T^K} \left[\mathbb{E}_{S\sim d_{t^K}, q\sim d_{t^K}} \left[\sum_{(x,y)\in q} \log(P_\theta(y\|x,S))\right]\right]$

TABLE 3.1: This table shows a comparison of formulas between normal neural network classifier training and episodic meta-learning training.

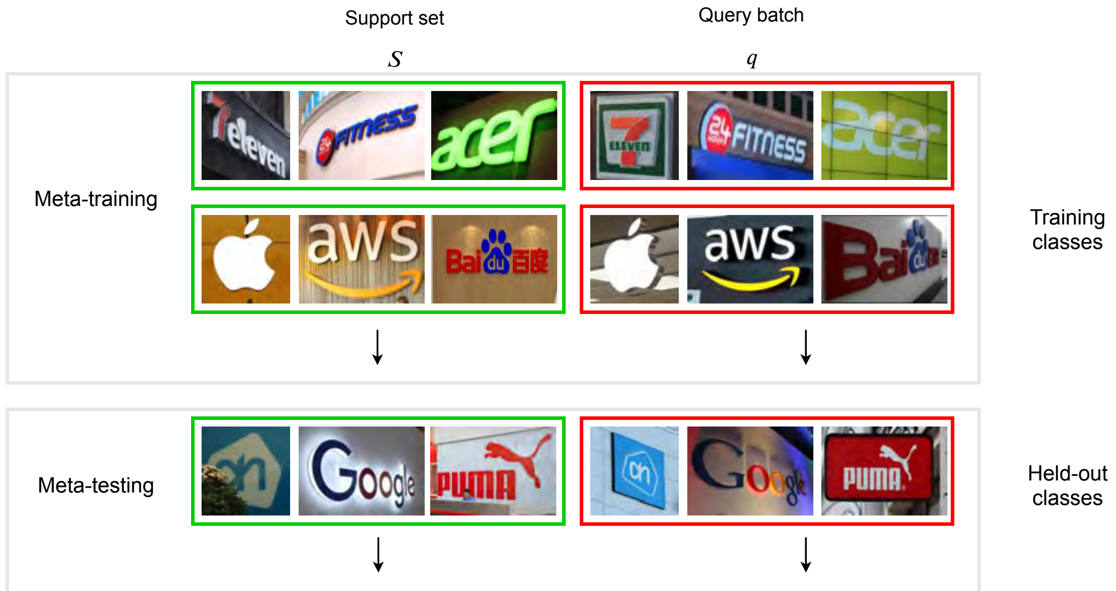


FIGURE 3.10: Shows the distribution of training and testing datasets for meta-learning. In this image, the training and testing sets reflect one-shot, three-way learning. In every episode, one sample of the corresponding class is sampled for the query. This can be scaled up, which resembles having a larger batch size in standard classification training.

3.3.2 Three directions in meta-learning

Since Vinyals et al. published their paper, many new meta-learning techniques have been developed that use the episodic training strategy. We need to investigate which of these techniques is the best fit to the flow circle from Chapter 2. Vinyals gave a presentation at NIPS 2017 wherein he roughly split the techniques in three different directions, deep metric learning, initialisation learning, and model-based meta-learning. We will use these splits to investigate what their strengths and weaknesses are and then we can better define which of these techniques have the potential to meet the requirements of the quality metrics.

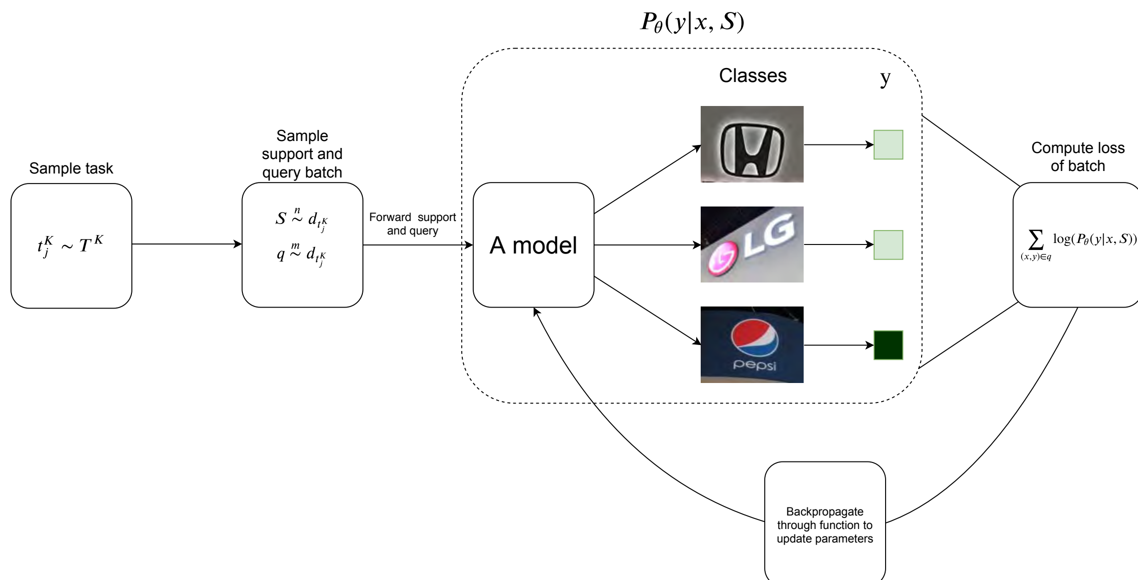


Figure 3.11: Structure showing an iterative forward and backward propagation process to update parameters in a neural network classifier that uses episodic training.

3.3.2.1 Deep metric learning

Deep metric learning is inspired by the nearest-neighbours algorithms, such as k-NN classification; k-NN is a non-parametric method and assigns a class membership to a query input by a plurality vote of its neighbours. In Figure 3.12, we show a two-dimensional representation space with a five-shot, two-way learning task. By performing the algorithm with three nearest neighbours, means that we look at the three closest representations of the query input. A plurality vote is then in favour of support set class S_2 and hence the query is assigned to the red class.

In deep metric learning, the representation of an input query is generated by a neural network function. The aim is to learn a representation generator that is generic to all $t_j \in T$ and generalises to $t_j^{'} \in T^{'}$. We can intuitively think that the neural network learns to recognise universal logo properties from logo images, such as the colour of the logo, and represent these in a representation, such that the combination of these universal properties generates a unique representation for its representative class. Representations of the same class then cluster together in the representational space and we can apply a form of k-NN.

A problem with k-NN in Figure 3.12 is that the process is discrete, which results in that any objective function based on k-NN not being smooth and hence not differentiable [39]. This means that we cannot use gradient descent to train k-NN with a neural network end-to-end. Goldberger et al. proposed a solution to make k-NN stochastic, such that gradient-based optimisation is possible. They do this with the use of the

softmax function that we described in Equation 3.1. The change is that in Figure 3.12 the distances from the query input to all other support set samples is probabilistic. This function is reflected in Equation 3.4, with $k_\theta(x, x_i)$ defined as similarity function that is parametrised by a neural network. To make a classification, we can then sum all the probabilities that assign to a specific class, which we do in Equation 3.5, and then assign the query input to the class with the highest probability (i.e. $\arg\max_y P_\theta(y|x, S)$).

$$k_\theta(x, x_i) = \frac{e^{-d_\theta(x, x_i)}}{\sum_{x_j \in S} e^{-d_\theta(x, x_j)}} \tag{3.4}$$

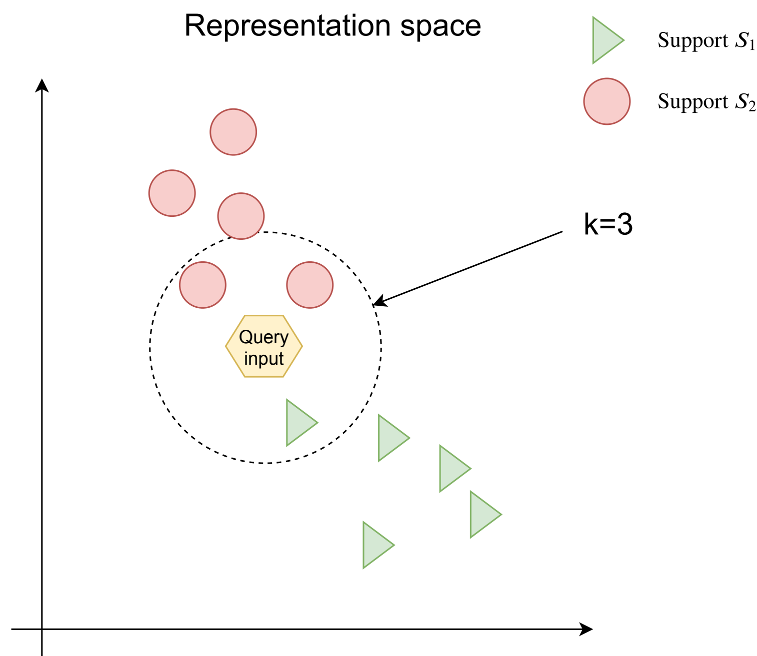


FIGURE 3.12: This Figure shows an example of k-NN classification with k=3, a support set with two classes with for both five examples each and one query input.

$$P_\theta(y|x, S) = \sum_{(x_i, y_i) \in S} k_\theta(x, x_i) y_i \tag{3.5}$$

We define some of the advantages and disadvantages of this method as follows:

- **Advantages**
  - Task t_j^K can easily be exchanged for another task by changing the support set, which means strong within-flexibility. No fine-tuning with, for example, gradient descent is necessary. This also means that we can easily replace

t_j^K with, for example, t_j^{K+1}. Thus, this approach has also strong between-flexibility.

- Memory can easily be increased by adding more samples (n) in the support set classes.

- **Disadvantages**
  - This method does only across-task learning (universal property learning) and not within-task learning (fine-tuning on the support set), which might not work for all datasets or requires large datasets to learn appropriate universal properties.
  - All the representations of the support set and the model need to be stored, which can have implications if the model is deployed on small devices.

3.3.2.2 Initialisation learning

Initialisation learning is an extreme form of transfer learning, in which, contrary to transfer learning, an initialisation from tasks $t_j^K \in T^K$ is learned. We show an intuitive image in Figure 3.13, which compared to Figure 3.9, shows the difference between initialisation and transfer learning. With initialisation learning, the tasks $t_j^K \in T^K$ are used to learn a parameter initialisation that is close to every optimal parameter manifold of tasks $t_j^K \in T^K$. Consequently, a small support set with, for example, five samples per support set class is able to learn different but related tasks.

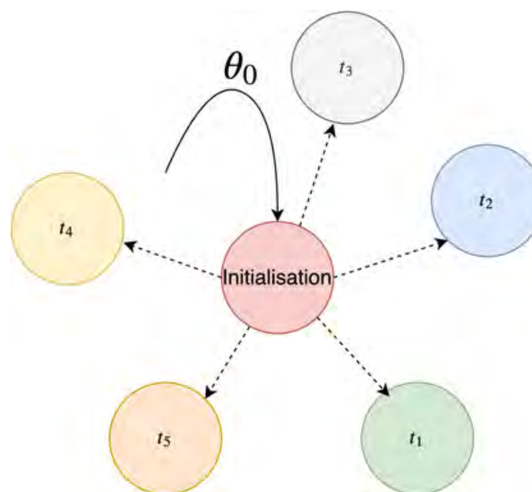


FIGURE 3.13: This picture shows intuitively how initialisation optimisation works. An initialisation is found that is close to all tasks in T, such that with only a few samples a new task can be learned.

For initialisation learning, we use a classification neural network model $g_\theta(x)$ that is able to learn a new task with an optimiser v_{θ_v}. This optimiser updates the initial

parameters θ_0 to the parameters that lie in the optimal parameter manifold of a task t_j^K with a support set S. It does this with the backpropagation algorithm. We show this optimisation step in Equation 3.6b. The model $g_\theta(x)$ then uses these updated parameters to classify queries into the K labels from class t_j^K, which we show in Equation 3.6a.

$$P_\theta(y|x, S) = g_{\theta(S)}(x) \quad (3.6a)$$

$$\theta(S) = v_{\theta_v}\left(\theta_0, \{\nabla_{\theta_0} \log(P_{\theta_0}(y_i|x_i))\}_{(x_i, y_i) \in S}\right) \quad (3.6b)$$

We define here some advantages and disadvantages of this approach:

- **Advantages**
  - Within-task training; the model specialises on a task t_j^K with fine-tuning. This can be beneficial when it needs to generalise to out-domain data.
  - This is a model agnostic approach, which means it can be applied to regression, classification, and reinforcement learning tasks [41]. However, for this research, it is not beneficial, because we only need to perform classification.
- **Disadvantages**
  - The initialisation needs to be learned from scratch if a new task, for example, t_j^{K+1}, has more classes than the original initialisation was learned on. Hence, it has a poor between-flexibility.
  - Some gradient steps need to be performed to learn a new task t_j^K, which makes this approach less flexible, because this requires some computation power.

3.3.2.3 Model based learning

Model-based approaches do not make assumptions on $P_\theta(y|x, S)$, but focus specifically on the design of the model g_θ to make it work for learning new tasks and hence $P_\theta(y|x, S) = g_\theta(x, S)$. Referred to as black boxes by Larochelle, the strength of this approach is that there are no assumptions made and the model is able to learn any function by itself. However, the downside is that this usually requires substantially more data, because the models are more complex.

We define the advantages and disadvantages of this approach here:

- **Advantages**
  - It can be very powerful when there are enough data to learn complex functions.
- **Disadvantages**
  - It is a black-box model and is often large with many parameters.
  - In many cases, the network needs to be retrained from scratch when it is optimised for learning tasks of the form t_j^K and needs to learn, for example, a new task t_j^{K+1}. Hence, just as with initialisation learning, there is poor between-flexibility.

3.3.3 Meta-learning data and models

To be able to train meta-learning techniques, we sample tasks from a set of tasks that we estimate by sampling K classes from a set of labels L. To be able to sample a diverse set of tasks, we need a large set of different logo classes in set L and as such we need to have a dataset with many different logo classes. There are a few datasets available with a large number of classes. We have summarised these datasets in Table 3.2. However, we must note that many of those datasets have overlapping classes and samples. As such, the Logos in the Wild dataset is a collection of images from WebLogo-2M, Logo-NET, and the FlickrLogos datasets. In addition, some of these datasets are very noisy or a subset of the data is unsupervised, such as WebLogo-2M and Logo-NET. We thus focus solely on the Logos in the Wild dataset. One issue, however, is that many classes in the Logos in the Wild dataset contain less than 10 samples per class, which can make it hard to train meta-learning models when there are not enough samples for a support set and query set.

Most datasets from Table 3.2 are focused on logo detection and not on logo classification. This means that these samples have many (different) logos in their picture with files that contain bounding box (x and y coordinates that specify the location of the logo in the image) information. We can extract the logos from the picture by cutting out the region of interest (ROI) using the bounding box information. The disadvantage is that the quality of the logo samples (number of pixels) will probably be less than pictures of logos taken by a mobile phone. Nowadays, the quality of pictures taken by mobile phones ranges between 8 and 21 megapixels, which is roughly equal to 3,264 x 2,448 and 5,120 x 4,096 pixels. This means that there can be a quality gap between our dataset used to train the generic model and the input queries the model must handle during deployment of the conference application.

Dataset	Number of logo classes	Number of images
BelgaLogos [11]	37	10,000
FlickrLogos-32 [12]	32	8,240
FlickrLogos-47 [12]	47	8,240
TopLogo-10 [43]	10	700
Logo-NET [6]	160	73,414
WebLogo-2M [9]	194	1,867,177
Logos in the Wild [10]	871	11,054

TABLE 3.2: Statistics of available datasets to train logo detection models. The region of interest can be used to extract supervised logo samples.

Direction	Method	5-Way Mini-ImageNet	
		1-shot	5-shot
Metric	Matching nets [37]	43.6%	55.3%
	Prototypical nets [13]	46.61 ± 0.78%	65.77 ± 0.70%
	Relation nets [44]	51.38 ± 0.82%	**68.20 ± 0.66%**
Initialisation	Meta-learn LSTM [45]	43.4 ± 0.77%	60.2 ± 0.71%
	MAML [41]	48.7 ± 1.84%	63.1 ± 0.92%
	Reptile [2]	49.97 ± 0.32%	65.99 ± 0.58%
Model based	Meta-nets [46]	49.21 ± 0.96%	-
	SNAIL [47]	**55.71 ± 0.99%**	**68.88 ± 0.92%**

TABLE 3.3: Results of popular meta-learning techniques with one-shot and five-shot, five-way learning task classification accuracies on the mini-ImageNet dataset. Some scores include a 95% confidence interval. Note that more techniques are published constantly. Thus, this table does not include all techniques but is a representation of the most popular techniques of the past four years.

Meta-learning models

Many different meta-learning models exist. We have depicted the most popular models in Table 3.3 and categorised them in one of the three directions. We see that Relation nets and SNAIL are the best performing models on the mini-ImageNet [37] dataset. However, we need to see first if this ranking based on performance stays the same when we train and test these models on a logo meta-learning dataset. Finally, we use the following naming conventions in the tables for brevity: network (net) and prototypical network (protonet).

3.4 Discussion of literature research

We have searched for a technique that is able to learn new tasks $t_j \in T$ with five training samples per class. We want this technique to be reliable and flexible enough such that we can integrate it in the flow circle in Figure 3.2 from the problem definition section.

We have found that the deep meta-learning domain consists of techniques that have the potential to match our requirements. With this conclusion, we have partly answered our first question from Chapter 2:

what technique(s) fit in the circle flow and have the potential to be reliable and flexible.

To answer this question further and the other questions, we need to investigate which meta-learning techniques are most reliable regarding the problem. We have, however, recognised that deep metric learning is within- and between-flexible, whereas initialisation and model-based learning approaches have poor between-flexibility. This means that initialisation and model-based approaches need to relearn a set of tasks from scratch if they need to prepare on a task that has more or fewer classes than that on which the current generic model is prepared. This can have a high impact on flexibility, because it requires an expert to learn a generic model again, which could take more than 6 hours to accomplish. For this reason, we conclude that deep metric learning approaches are more flexible than initialisation and model-based learning approaches.

The question remains if deep metric learning actually works on logos, because they are abstract and made specifically to be different from one another. It can thus be hard to make a general representation function that depends on universal logo properties. A recent study conducted by Fehérvári and Appalaraju showed that deep metric learning can work on logos. However, in this study, the training dataset consist of 169,400 samples, which is large compared to the Logos in the Wild dataset depicted in Table 3.2.

Looking at the sub-question of *'how should we collect data to make a reliable model?'*, there could be a difference in quality between the acquired dataset for training and evaluation, and the mobile pictures the model should make inferences on during deployment. If this is an issue, we need to collect higher qualitative data. Alternatively, it could be that initialisation-based meta-learning models are better able to handle these quality differences than models from the other two directions, because initialisation learning models apply within-task learning. Hence, we introduce an extra quality metric for the generic model, which we call robustness to *quality shift*.

We present a comparative overview of our findings in Table 3.4, where we measure on two different scales. In Table 3.4, we measure on the meta-learning direction scale. However, when we have individual meta-learning model results, we present the comparative findings on a model scale. In every row, we compare the direction or models on the basis of the quality metric or sub-quality metric for which a + means the best in that row, a $\pm$ is average, and $-$ the worst. We can see in Table 3.4 that deep metric learning is most flexible, because we can change the set of learning tasks without re-training the model. This is not the case for initialisation and model-based learning for which a change of, for

Quality metric	Sub-quality metric	Direction		
		Deep metric learning	Initialisation learning	model-based learning
Accuracy				
Rejection accuracy				
Robustness	Within-robustness			
	Between-robustness			
	Quality shift robustness			
Flexible		+	+−	+−

TABLE 3.4: Comparative presentation of our findings between the three different meta-learning methods.

example, five-shot, five-way to five-shot, 10-way learning means that we need to re-train the generic model from scratch.

We structured this research to answer the other questions and sub-questions of Chapter 2 as follows: (i) we built a small logo meta-learning classification dataset to research a selection of meta-learning models with respect to accuracy in Chapter 4; (ii) we investigated within- and between-robustness by finding accuracy metrics on different sets of learning tasks in Chapter 4; (iii) we made a model selection based on the found accuracy metrics and robustness in Chapter 4; (iv) we created a large logo dataset that resembles the use case better in Chapter 5; and (v) we investigated the rejection accuracy and robustness to quality shifts in Chapter 5.

Chapter 4

Empirical analysis on meta-learning models

Many meta-learning methods exist with their own pros and cons. We constructed a taxonomy of these methods that includes three directions in Chapter 3. In this chapter, we make a strategic selection with the use of the taxonomy to assess which methods are best applicable to be a generic model in the flow circle defined in Chapter 2. We do this in the following way: (i) construct a small logo dataset, (ii) use the small logo dataset to test out different methods from mainly Table 3.2 in Chapter 3, and (iii) make a selection of the best performing models in each direction based on accuracy, and between- and within-robustness.

4.1 Data

Generally, two datasets are used to evaluate and compare meta-learning models in the meta-learning domain, the mini-ImageNet dataset that is used to evaluate the meta-learning methods in Table 3.2 and the Omniglot [49] dataset that consists of grey scale-drawn characters from many different languages. We introduce the Omniglot dataset in this section, because we use it to evaluate if we implemented the meta-learning methods correctly. Thereafter, we introduce the small logo dataset that we create to test out which methods in the meta-learning space work well on the logo classification task. This dataset is relatively small, because we want to iterate quickly through the methods and save costs. In Chapter 5, we introduce a large logo dataset with which we focus more on collecting samples that resemble the usage phase of the conference application.

4.1.1 Omniglot dataset

A dataset of characters was collected by Lake et al. to research applications for sets of one-shot learning tasks. The dataset consists of 50 different alphabets from around the world and even includes some alphabets from science fiction novels. The 50 alphabets have 1,623 different characters (i.e. 1,623 classes), in which every character has only 20 examples and for which all characters are drawn by 20 different persons. This means that only 20 training examples are provided for every character class. The characters are in greyscale with a size of 105 x 105 pixels. We show some of the characters from the dataset in Figure 4.1.

FIGURE 4.1: Two characters from each of eight alphabets.

Omniglot dataset preparation

In meta-learning literature, the Omniglot dataset is normally pre-processed by resizing the images to 28 x 28 pixels and using rotation to extend the number of classes even further. The dataset is extended by multiple rotations of 90°, for which every rotation generates a different character. In this way, the number of classes is multiplied by 4. We apply the same pre-processing procedure and use the standard train-test split from Lake et al. This results in 4,800 different character classes for meta-training and 1,692 character classes for meta-testing. We present more statistics about this dataset in Table 4.1.

4.1.2 Small logo dataset

We mentioned before that there is much overlap between the datasets from Table 3.2 in Chapter 3 so we focus only on constructing the small logo dataset from the Logos in the Wild dataset.

We saw in Chapter 3 that episodic training is applied to learn meta-learning models. This requires sampling a support set S and query batch q per iteration from a dataset $d_{t_j^K}$. For example, to mimic five-shot, five-way learning tasks during training, we need

to sample five logo images per support set class and 25 images in total for the complete support set. To evaluate and improve the model, we need to sample at least one logo image per class for batch q. In many published meta-learning papers, such as those by Nichol et al., Snell et al., Finn et al., and Sung et al., the authors use between 5 and 15 samples per class for batch q. This means that if we want to sample five logo images per support set class and five logo images per query batch class, we need to sample 10 logo images for each of the five classes per iteration in total. However, the number of samples per class in the Logos in the Wild dataset is often limited to 10 or fewer images per class. We therefore decide to use 10 samples per class to not limit the total number of classes in the small logo dataset too much. This means that we can both sample five logo images per support set class and per query batch class.

Further, in Figure 4.2, we present how we extract logos from the images of the Logos in the Wild dataset. There are two interesting observations: (i) the extracted ROIs are closely bound to the logo and (ii) extracted ROI number 3 is blurred compared to the other two ROIs.

Observation (i) is a problem, because mobile application users that take a picture of a logo will have a hard time bounding the logos that tightly. We need to assume that there will be much more background information in the pictures present during the usage phase of the conference application. This means that the generic model also needs to learn how to discard this background information, because this information is redundant to the task of classifying logos. We want to improve this by making the ROI larger than the bounding boxes given by the Logos in the Wild dataset. However, we cannot increase the ROI substantially, because the Logos in the Wild dataset splits logos that consist of symbols and text. For example, a combined logo of the Pepsi symbol and Pepsi text is visible in the left picture of Figure 4.2. If we increase the ROI of either the Pepsi symbol or Pepsi text too much, their ROI will overlap, which could influence the class definition. We therefore used the standard bounding box information from the Logos in the Wild dataset as our ROI and focused only on increasing the ROIs during construction of the large logo dataset in Chapter 5.

Observation (ii) shows a lack of information, because the logo is too small in the original picture or because of how the picture was taken. We see that in some occasions the extracted ROIs lack so much information that we are unable to recognise to which logo the ROI belongs. We deem these ROIs as too noisy and having a lack of information and remove these samples from the dataset. We conclude visually that a sample image needs to have at least 625 pixels in total (e.g., 25 x 25 pixels).

FIGURE 4.2: The image on the left is a classic picture that comes from the logo detection datasets that we use. We extract the ROIs of the logos on the left and present them on the right. We use the extracted ROIs for logo classification.

	Omniglot	Mini-ImageNet	Small logo
Total meta-train classes	4, 800	60	151
Total meta-test classes	1, 692	40	40
Samples per class	20	600	37*
Pixel size of images	28 x 28	84 x 84	60 x 60
Total images	129, 840	600, 000	14, 849

TABLE 4.1: This table shows statistics of standard meta-learning datasets and a derived dataset by us, which is called the small logo dataset. The $*$ means that we have used the median as metric.

Small logo dataset preparation

We extracted all ROIs from the pictures of the Logos in the Wild dataset and labelled them according to the given classes from the Logos in the wild dataset. We filtered the collected ROIs on the basis of our defined threshold, which means that any ROI below a total of 625 pixels was removed. We then looked at the number of logo images per class. The logo classes that have less than 10 ROIs were removed. This resulted in a dataset with 191 classes that include 14,849 ROIs (logo samples) in total, which is a significant decrease from the 871 classes that are present in the Logos in the Wild dataset. We split the 191 classes in a meta-train and meta-test set by, respectively, randomly sampling 151 classes and selecting the 40 leftover classes.

In Figure 4.3, we show the distribution of the number of logo samples per class in the meta-train set, which has a median of 37 logo samples per class. Note that the distribution is heavily skewed to the right. We also evaluated the median number of pixels per logo sample, which is roughly 60^2 pixels in total. To save costs and iterate quickly through various methods, we decided to resize all samples to the median size of 60^2, which we did with bilinear interpolation [50]. As such, every image has a size of 60 x 60 pixels. In Table 4.1 we show comparison statistics of the Omniglot, mini-ImageNet, and small logo datasets.

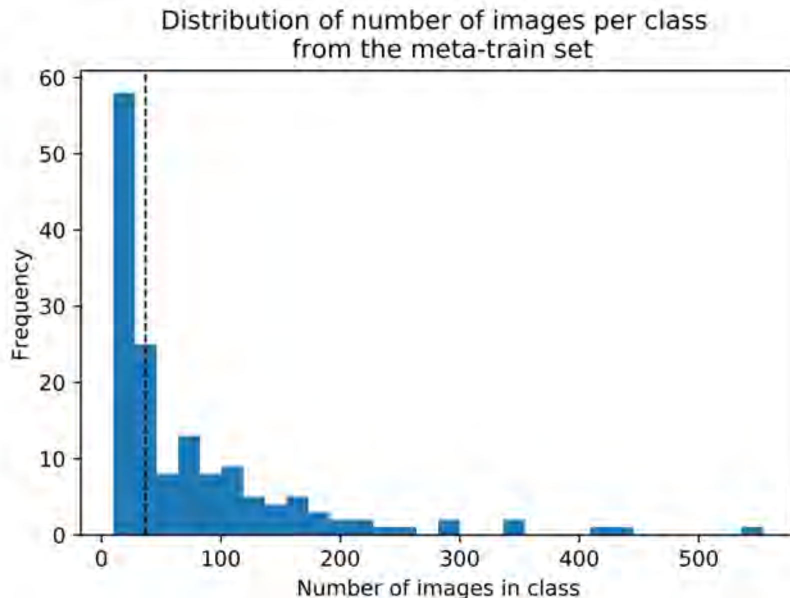


FIGURE 4.3: This graph shows the class distribution of the meta-train small logo dataset. The dotted line represents the median and the minimum number of samples in a class is 10.

4.2 Methods

In this section, we explain how the methods that we use to do our experiments work. We selected five methods on the basis of their published accuracy results and selected at least one method on the basis of our defined taxonomy in Table 3.3. We used these methods to investigate accuracy, and between- and within-robustness on the small logo dataset.

We used the following best-performing methods from Table 3.3: prototypical nets, relation nets, MAML, reptile, and SNAIL. For prototypical networks, we added an extension that is called Gaussian prototypical networks [1]. We also included a deep metric learning model that has been used by Fehervari and Appalaraju because it shows good performance on logo samples; we call it the proxy network.

In Chapter 3, we defined that meta-learning methods are trained with episodic training. This means that we sample every iteration a support set $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ and batch $q = \{(x_1, y_1), ..., (x_M, y_M)\}$ where each $x_i \in \mathbb{R}^D$ is a D-dimensional representation of an input image, and $y_i \in \{1, ..., K\}$ specifies the class. We denote S_k and q_k as the set of logo samples that belong to class k, with $k \in \{1, ..., K\}$.

For every experiment in this section, we used a backbone network (i.e. neural network architecture) $h_\theta(x)$. This network consists of a stack of modules, for which each has a 3 x 3 convolution with 64 filters followed by batch normalisation, a ReLU function,

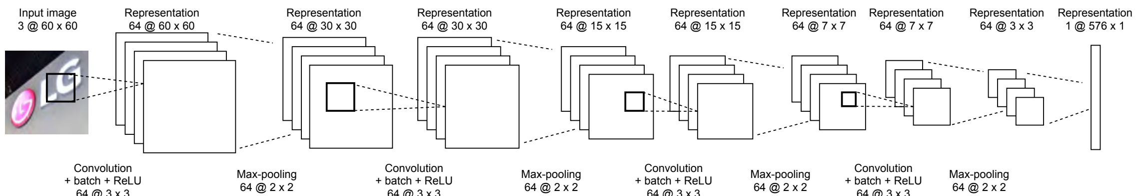


FIGURE 4.4: The standard backbone with stride 1 for the convolutions and without a softmax layer.

and 2 x 2 max-pooling. We stacked four of these modules, which resulted in an output representation of 64 channels each 3 x 3 in size when we input a logo image of 60 x 60 pixels. This input image has a dimension of 3 x 60 x 60 because of the three RGB channels, and we refer to this as the D-dimensional input query x. We vectorised the output channels, which resulted in an E-dimensional representation of 576 dimensions. We depict this architecture in Figure 4.4.

Depending on the method, the output of the standard backbone was used as a representation such as in deep metric learning, or the output was stacked with a softmax layer to compute probabilities per class such as in initialisation learning. We use $h_\theta(x)$ for the former and $g_\theta(x)$ for the latter. Finally, for SNAIL this backbone is also used but is chained together with another neural network architecture.

4.2.1 Deep metric learning

In Chapter 3, we defined that deep metric learning approaches learn a similarity function and we described this with $P_\theta(y|x, S) = \sum_{(x_i,y_i)\in S} k_\theta(x, x_i)y_i$, which gives a distribution over classes. We use this notation to explain various deep metric learning techniques.

4.2.1.1 Prototypical networks

Prototypical networks learn a representation function $h_\theta : \mathbb{R}^D \to \mathbb{R}^E$ that outputs an E-dimensional representation. It uses this representation function to create prototypes $c_k \in \mathbb{R}^E$ from support set S that represent the centroid of a cluster from class k. We create these prototypes by computing the mean with representations belonging to S_k. In Equation 4.1, we show how we compute these prototypes for which $|S_k|$ represents the number of samples in support set k. Given a distance function, we can then compute the probability distribution over the classes given a sample x from batch q and support set S. We show this in Equation 4.2 in which $p(S)$ computes the set of prototypes.

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} h_\theta(x_i) \tag{4.1}$$

$$P_\theta(y|x, S) = \sum_{(c_i, k_i) \in p(S)} \frac{e^{-d(h_\theta(x), c_i)}}{\sum_{(c_i', k_i') \in p(S)} e^{-d(h_\theta(x), c_i')}} k_i \tag{4.2a}$$

$$p(S) = \left\{ \left(\frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} h_\theta(x_i), k \right) \right\}_{k=1}^{K} \tag{4.2b}$$

We show in Figure 4.5 how Prototypical networks make prototypes and computes distances to these prototypes with a new input x in a two-dimensional representation space [13]. Snell et al. justify using distance metrics that are a Bregman divergence [52], such as the Euclidean distance. They show that for the class of Bregman divergence distance functions, prototypical network is equivalent to performing mixture density on the support set with an exponential density function. They also empirically show that the Euclidean distance performs significantly better than the cosine distance. Hence, we also use the Euclidian distance to calculate the distance of a query to a prototype.

It follows from the equivalence to performing a mixture density on the support set, that every class k from Prototypical networks can be represented in the representation space as spherical Gaussian densities with an identity covariance matrix [13]. It must also be assumed that they have the same variance across every dimension to be able to define one threshold to reject input queries that do not belong to any class. The question is if the function $h_\theta(x)$ is able to meet these assumptions for every logo class. A solution to soften these assumptions would be to use not only the mean (c_k) of a class k and assume an identity covariance matrix but to estimate a covariance matrix for every class k. We then only need to assume that clusters are Gaussian densities. Finally, in Appendix C.1, we conduct an experiment to visually show the spherical Gaussian densities in representation space with the MNIST dataset [53].

Gaussian prototypical networks

Gaussian prototypical networks try to account for some of the assumptions made in Prototypical networks, such as having the same variance across dimensions. They do

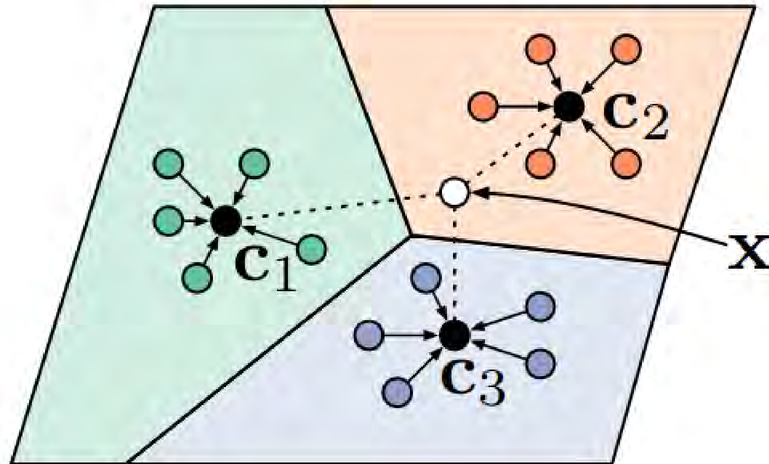


FIGURE 4.5: A two-dimensional representation space of Prototypical networks with a five-shot, three-way learning task.

this by letting the function $h_\theta(x)$ not only output representations but also output independent inverse covariance matrices. They use these matrices to adjust the prototypes by computing weighted prototypes and use them to normalise the distances. Intuitively, we now try to compute how many standard deviations an input x is from a cluster prototype c_k. To output also inverse covariance matrices, the number of channels of the convolutional layer in the last module from the standard backbone is doubled to 128. This means that an input query x of size 3 x 60 x 60, outputs an $2E$-dimensional representation of size 1152.

Gaussian prototypical networks learn a representation function $h_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^{2E}$ that outputs an 2E-dimensional representation. However, the first half of this vector $[1, ..., E]$ represents the inverse variances of the diagonals of an invariance matrix (i.e. inverse of covariance matrix). The second half (E,...,2E] is the representation of the input. We define the invariance matrix by $h_\theta(x)_1$ and the representation by $h_\theta(x)_2$. Only the inverse of the variances are estimated and not the inverse of the correlations, because also estimating correlations mean that we need to estimate $(E)^2$ values which increases the parameters of the model substantially. Furthermore, we can represent the inverted variance vectors of $h_\theta(x)_1$ to an invariance diagonal matrix, as can be seen in Equation 4.3. In our equations, we use the vector representation so we use the Hadamard product ($\circ$).

The inverse covariance matrix should be positive definite (PD) to make sure we are in compliance with the Gaussian density assumption of a PD covariance matrix. This means that an n x n matrix A is PD if the scalar $z^T Az$ is strictly positive for every

non-zero column vector $z \in \mathbb{R}^n$ [54]. To be sure that the E-dimensional outputs are PD, we use a function to transform the dimensions independently. This function needs to transform a value from $\mathbb{R}$ to $\mathbb{R}_{>0}$. For this, we use the softplus function, defined as $\log(1+e^x)$, to be sure that the vector values are between $(0, \infty)$. We use the function that gives the best result following Fort's research, which is the extended softplus function $T_\theta(x_i)$ defined in Equation 4.4.

$$h_\theta(x)_1 = \begin{bmatrix} h_\theta(x)_1^1 \\ \vdots \\ h_\theta(x)_1^E \end{bmatrix} \rightarrow \begin{bmatrix} h_\theta(x)_1^1 & & 0 \\ & \ddots & \\ 0 & & h_\theta(x)_1^E \end{bmatrix} \tag{4.3}$$

$$T_\theta(x_i) = 1 + \log(1 + e^{h_\theta(x_i)_1}) \tag{4.4}$$

Gaussian prototypical networks use the invariance matrices to adjust the means of the prototypes. The intuition is that for representations the network is unsure about, such as logos that are heavily blocked by other objects, get less weight to compute the mean of the prototype. In Equation 4.5, we show how we compute the adjusted prototypes.

$$c_k = \frac{\sum\limits_{(x,y)\in S_k} T_\theta(x_i) \circ h_\theta(x_i)_1}{\sum\limits_{(x,y)\in S_k} T_\theta(x_i)} \tag{4.5}$$

The invariance matrices from every support sample in a class k are also used to compute one invariance matrix for every class k. We do this by adding all invariance matrices from the support samples of a class k, defined in Equation 4.6. We then use this invariance matrix with the Mahalanobis distance function, defined in Equation 4.7, to compute the distance of an input query x_i to support set k.

$$\Sigma_k^{-1} = \sum_{(x_i,y_i)\in S_k} T_\theta(x_i) \tag{4.6}$$

$$d(h_\theta(x)_2, c_k, \Sigma_k^{-1}) = \sqrt{(h_\theta(x)_2 - c_k)^T \Sigma_k^{-1} \circ (h_\theta(x)_2 - c_k)} \tag{4.7}$$

We can now combine these functions to represent the Gaussian prototypical network as a probability distribution over classes in Equation 4.8. We learn this network end-to-end with episodic training.

$$P_\theta(y|x,S) = \sum_{(c_i,\Sigma_i^{-1},k_i)\in p(S)} \frac{e^{-d(h_\theta(x)_2,c_i,\Sigma_i^{-1})}}{\sum\limits_{(c_i^{'},\Sigma_i^{'},k_i^{'})\in p(S)} e^{-d(h_\theta(x)_2,c_i^{'},\Sigma_i^{'})}} k_i \tag{4.8a}$$

$$p(S) = \left\{ \left(\frac{\sum\limits_{(x_i,y_i)\in S_k} T_\theta(x_i) \circ h_\theta(x_i)_1}{\sum\limits_{(x_i,y_i)\in S_k} T_\theta(x_i)}, \sum_{(x_i,y_i)\in S_k} T_\theta(x_i), k \right) \right\}_{k=1}^{K} \tag{4.8b}$$

4.2.1.2 Relation networks

Gaussian prototypical networks and vanilla prototypical networks use standard distance metrics to compute how close a query is to a support set S_k. However, it is not always straightforward to decide what the best distance metric is. It could even be that all our defined and known distance metrics can be improved by a generated distance metric unknown for us. This is exactly what Sung et al. proposed, a neural network that replaces the distance metric such that it learns the most appropriate distance metric by itself.

The representation function is $h_{\theta_1} : \mathbb{R}^D \to \mathbb{R}^E$ that outputs an E-dimensional representation for both samples from the query batch q and the support set S. The Relation network uses this function to add the representations of the samples from support set S that belong to class k together. We compute this with $a_k = \sum_{(x_i,y_i)\in S_k} h_{\theta_1}(x_i)$ for which $a_k \in \mathbb{R}^E$. The relation function $h_{\theta_2}^{'} : \mathbb{R}^{2E} \to (0,1)$ computes how closely related the support set representation a_k is to an input query x by outputting a score between 0 and 1. The input of the relation function is a concatenation of a query representation $h_{\theta_1}(x)$ and support set representation a_k, which we denote as $\zeta(h_{\theta_1}(x), a_k)$. We use the exact same relation function that is used by Sung et al. We can now present a function that computes probabilities over classes that is parameterised by the representation function

and relation function in Equation 4.9.

$$P_{\theta_1,\theta_2}(y|x,S) = \sum_{(a_i,k_i)\in A(S)} h'_{\theta_2}(\zeta(h_{\theta_1}(x_i),a_i))k_i \tag{4.9a}$$

$$A(S) = \left\{\left(\sum_{(x_i,y_i)\in S_k} h_{\theta_1}(x_i), k\right)\right\}_{k=1}^{K} \tag{4.9b}$$

In the published paper of Sung et al., they use the mean square error as loss, because they see this as a regression problem. We use the same loss function.

$$\arg\min_{\theta_1,\theta_2} \mathbb{E}_{t^K\sim T^K}\left[\mathbb{E}_{S\sim d_{t^K},q\sim d_{t^K}}\left[\sum_{(x,y)\in q}\sum_{k\in K}(P_\theta(k|x,S) - \mathbb{1}_{(y=k)})^2\right]\right] \tag{4.10}$$

4.2.1.3 Proxy networks

The focus on deep metric learning is to make representations such that similar images are close in the representation space and dissimilar images are far apart. The aim is thus to decrease the within variance of images from the same image class and increase the between variance of dissimilar images from different classes in representation space. A challenge in deep metric learning is sampling informative samples for training such that within variance and between variance is optimised [55–57]. Research by Zhai and Wu described that similar images in a batch or dissimilar images in a batch that are too easy to classify do no contribute much to training, because the loss would be near to zero. This can decrease learning speed, because many iterations are necessary to hope that sometimes a hard batch is sampled from which the model can learn. The methods we described previously (Protonets, Gaussian protonets, and Relation nets) counter this sampling problem by sampling many different classes (i.e., K classes) in one episodic iteration and use the relation between all the classes with the similarity function from Equation 3.4 in Chapter 3. This makes the chance of sampling harder samples higher and therefore the model learns faster and has higher accuracy. However, in reality, the batch size is constrained to the limit of the GPU memory that is used. Proxy networks is a method that tries to solve this problem [59].

In proxy networks, they do not use episodic learning but learn a deep metric model prior the invention of episodic training. This means that we define a dataset $D = \{(x_i, y_i), i = 1, ..., N\}$ that includes all samples and logo classes for which $y_i \in \{1, ..., K\}$. Proxy networks learn a one-shot K-way learning task. This means that with the small logo dataset we learn to do a one-shot, 151-way learning task, because there are 151 logo classes in the small logo training dataset. We represent these classes with proxies $Z = \{1, ..., K\}$.

Proxy networks learn a representation function $h_{\theta_1} : \mathbb{R}^D \rightarrow \mathbb{R}^E$ that outputs an E-dimensional representation for an input x_i. Now for every class from Z a representation is learned that represents that class. Intuitively, you can think about this as learning a centroid for every class in Z. These representations are learned during backpropagation and are called *proxies*. We define a set of proxies by $p_{\theta_2}(Z) = \{proxy_1, ..., proxy_K\}_{\theta_2}$.

Proxy networks uses the same similarity function as is used by prototypical networks and Gaussian prototypical networks to define a probability that a query belongs to a class. However, they use all the proxies to compute the probability, which is the strength of proxy networks, because every iteration of a sample x_i is compared to all classes from the training set. The proxies are initialised and learnable representations in $\mathbb{R}^{\mathbb{E}}$ for which $D >> E$ and hence proxies take up significantly less memory compared to images.

We present in Equation 4.11 the proxy probability distribution over the classes. We learn it with the formula in Equation 4.12. During test time, we do not use the proxies and use k-NN to assign a query to a class from support set S. We use k-NN as we have discussed in Chapter 3 and call this proxy k-NN. We also try out a configuration where we compute a prototype such as in prototypical networks, and we call this proxy cluster 1-NN.

$$P_{\theta_1,\theta_2}(y|x) = \sum_{k_i \in Z} \frac{e^{-d\left(h_{\theta_1}(x), p_{\theta_2}(k_i)\right)}}{\sum_{k_i' \in Z} e^{-d\left(h_{\theta_1}(x), p_{\theta_2}(k_i')\right)}} k_i \tag{4.11}$$

$$\arg\max_{\theta_1,\theta_2} \mathbb{E}_{q \sim D}\left[\sum_{(x_i,y_i) \in q} \log(P_{\theta_1,\theta_2}(y|x))\right] \tag{4.12}$$

4.2.2 Initialisation learning

Initialisation learning is focused on finding optimal values of the parameters for a function g_θ such that we can learn a new task with a few samples and gradient steps. The first difference in learning this initialisation compared to deep metric learning is that multiple tasks are sampled per episodic iteration. The second difference is that we are not specifically learning representations, but we are using a standard neural network classifier and use the backbone with a softmax layer. Finally, we use Equation 3.6 from Chapter 3 to explain the optimisation goal of both MAML and Reptile.

4.2.2.1 MAML

We learn a neural network classifier function $g_{\theta_0} : \mathbb{R}^D \rightarrow (0,1)^K$ that has a K-dimensional output with values in the range of (0,1), where K refers to the number of classes. The goal of MAML is to first fine-tune g_{θ_0} on the support set that comes from a task t_j^K. We define this in Equation 4.13, with which we use a support set S to optimise the parameters θ_0 with a few gradient descent steps. We then initialise the function $g(x)$ with these optimised parameters.

$$P_\theta(y|x, S) = g_{\theta(S)}(x) \tag{4.13a}$$

$$\theta(S) = v_{\theta_v}\left(\theta_0, \{\nabla_{\theta_0} \log(P_{\theta_0}((y_i|x_i))\}_{(x_i,y_i)\in S}\right) \tag{4.13b}$$

We find these parameters θ_0 by using meta-learning. However, we do this a bit differently in initialisation learning than in deep metric learning. We show in Figure 4.6 how one iteration is computed with MAML. It shows that B independent tasks are sampled from T^K for which in every task we optimise the parameters θ_0 independently by the process defined in Equation 4.13. The query batches q are then used to calculate a summed loss function over the B fine-tuned models. The last step is to backpropagate over the parameters θ_0 and update them.

We note that we need to take a derivative over a derivative when we backpropagate and thus we need to compute second derivatives. Following Finn et al., a small change lets us use a first-order approximation, which means that we only need to take first-order derivatives with nearly no loss in accuracy. Finn et al. also mentions that this increases the training speed by 30%. As such, we will use only the first-order approximation.

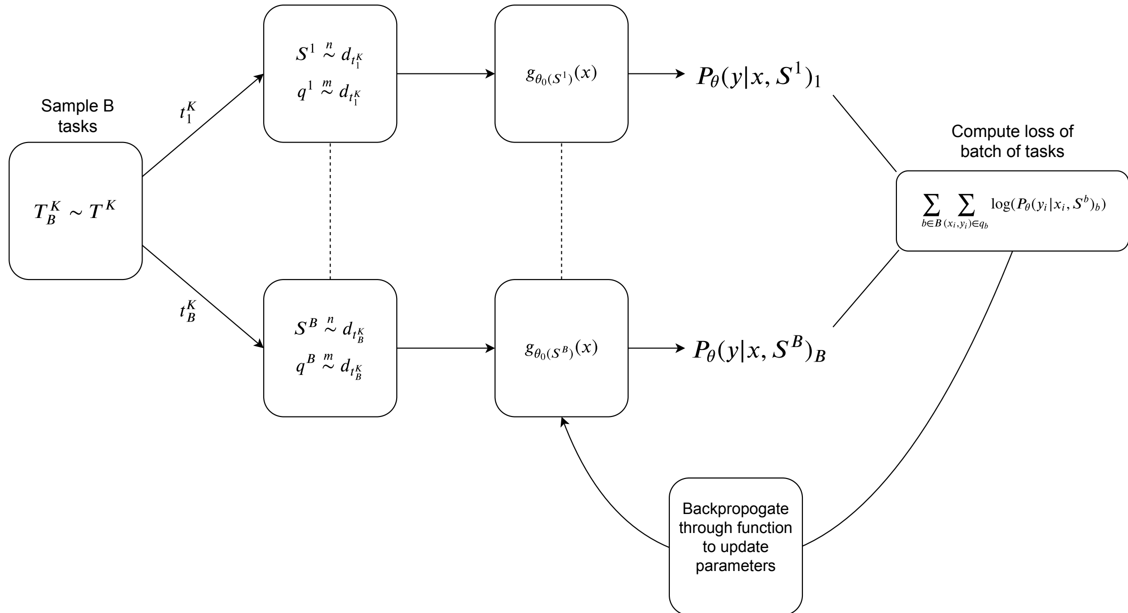


FIGURE 4.6: One episodic training iteration of the MAML model.

4.2.2.2 Reptile

We have seen that MAML uses parallel episodic iterations to learn general parameters that can be used to learn a new task with a few samples and a few gradient descent steps. Reptile was published by Nichol et al. as a scalable and faster method than MAML with comparable accuracy.

We still learn a network classifier function $g_{\theta_0} : \mathbb{R}^D \rightarrow (0,1)^K$ that is able to fine-tune g_{θ_0} on a support set from a task t_j^K and for which the distribution over classes is defined in Equation 4.13. The difference compared to MAML is that the initial parameters are updated directly with the derived parameters from the different tasks, which is denoted as Equation 4.14, with B the number of tasks and ϵ the learning rate. This means that there is no second backpropagation iteration over the function and we do not need to sample a query batch q to evaluate every fine-tuned task. In Figure 4.7, we show one iteration of learning the classifier with the Reptile algorithm.

$$\theta_0 \leftarrow \theta_0 + \epsilon \frac{1}{B} \sum_{b=1}^{B} (\theta_b - \theta_0) \tag{4.14}$$

The advantage over MAML is that we do not need to backpropagate for a second time, which gives an additional training speed increase compared to the first- and second-order MAML algorithm without a loss of accuracy on the Omniglot and mini-ImageNet

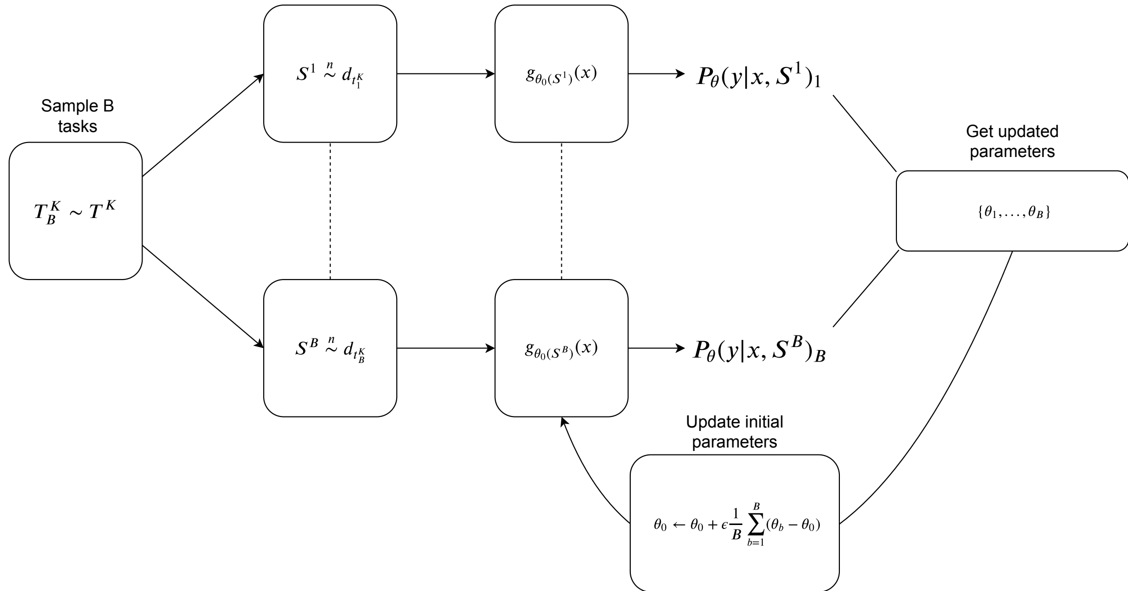


FIGURE 4.7: One episodic iteration for the Reptile model.

datasets. Nichol et al. published findings that the hyperparameters of Reptile are robust, information with which Baklid and Barlaug also agreed.

4.2.3 Model-based learning

In model-based learning, the focus lies explicitly on the function $g_\theta(x, S)$ that does the learning of new tasks on its own and outputs a probability distribution over categorical classes. A popular method is to treat the support set S as a sequential dataset and use this to train model structures as in Figure 4.8. Here, a recurrent neural network (RNN) is trained to take in a support set with labels and a query. The output is then a distribution over classes.

The goal of RNNs is to construct a meaningful representation of past datapoints from a sequence. However, this also seems to be the bottleneck of RNNs, because they have trouble passing long-term information through the network [61]. Recent publications that aim to use attention and causal convolutions (i.e. temporal convolutions) seem to improve on this problem [62, 63]. Mishra et al. was the first to apply these concepts in the meta-learning domain with state-of-art results on the Omniglot and mini-ImageNet dataset with the Simple Neural Attentive Meta-Learner, or SNAIL.

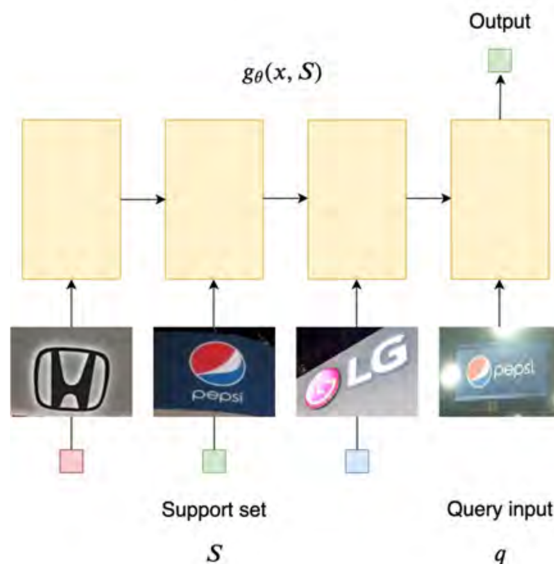


FIGURE 4.8: Example of model-based learning with an RNN.

4.3 Experimental setup

We selected seven methods based on accuracy results that are published in papers, on preferable properties and on the three meta-learning directions. These include all the discussed methods in this chapter, but we used only first-order MAML to speed up training, and we tested the Gaussian prototypical networks model only on the small logo dataset.

We evaluated if the implementation of the models was correct by training and testing on the Omniglot dataset. Thereafter, we compared these results to the published results if they were available. When we are close to these published results, we are confident enough that we had implemented the model correctly and used the model implementation to test on the small logo dataset. On the contrary, if we were not confident enough, we did not use the model further.

We trained the implemented models on the constructed small logo dataset to evaluate how good the accuracy, and between- and within-robustness are. We did this with three different sets of learning tasks: one-shot, five-way; five-shot, five-way; and five-shot, 20-way. We computed the mean accuracy on the meta-test set with a 95% confidence interval by sampling 1000 and 500 tasks from the Omniglot and small logo meta-test sets, respectively. We computed the confidence interval by assuming that the sets of accuracies are normally distributed, which follows from the central limit theorem [64]. This allowed us to derive a confidence interval with the t-distribution for a set of learning tasks, because we estimated the mean and variance of a set of accuracies [64]. We used

FIGURE 4.9: Learning curve of Gaussian prototypical networks on a five-shot, five-way learning task. Trained on the small logo dataset.

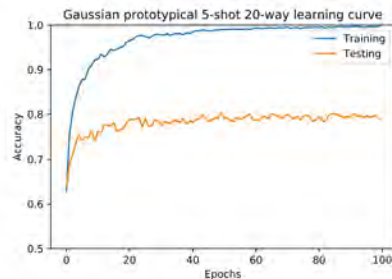


FIGURE 4.10: Learning curve of Gaussian prototypical networks on a five-shot, 20-way learning task. Trained on the small logo-dataset.

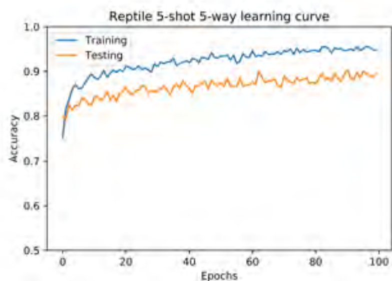


FIGURE 4.11: Learning curve of Reptile on a five-shot, five-way learning task. Trained on the small logo dataset.

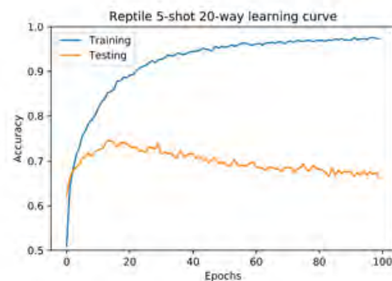


FIGURE 4.12: Learning curve of Reptile on a five-shot, 20-way learning task. Trained on the small logo-dataset.

the confidence intervals to evaluate if models are more within-robust and if a model has significantly more accuracy than other models. We investigated between-robustness by looking at the mean accuracy fluctuation between the five-shot, five-way and five-shot, 20-way learning tasks. Initially, we trained every model with 100 epochs. However, if we saw that the learning curve did not flatten at 100 epochs, we retrained the model with more epochs to see if the model was able to learn more. Finally, we present the hyperparameters that we used in Appendix A.

4.4 Results

We present the results that we use to evaluate our implementation of the models in Table 4.2 and we present the results of the models on the small logo dataset in Table 4.3. In Figure 4.9, 4.10, 4.11, and 4.12, we show the learning curves of both Gaussian prototypical networks and Reptile on the sets of five-shot, five-way and five-shot, 20-way learning tasks.

	1-shot 5-way	5-shot 5-way	Epochs
Protonet	**96.72 ± 0.31%**	**99.06 ± 0.14%**	100
Relation net	**96.77 ± 0.29%**	98.78 ± 0.15%	100
Proxy cluster 1-NN	94.68 ± 0.42%	91.61 ± 0.39%	100
Proxy k-NN	94.68 ± 0.42%	95.39 ± 0.32%	100
MAML first order	93.02 ± 0.76%	**99.04 ± 0.13%**	100
Reptile	95.07 ± 0.36%	**99.04 ± 0.13%**	100
SNAIL	81.50 ± 0.51%	93.54 ± 0.30%	100
SNAIL	93.31 ± 0.31%	97.36 ± 0.20%	500

Table 4.2: Few-shot classification accuracies of the models on the Omniglot dataset. The confidence interval is calculated by sampling 1000 tasks from the meta-test set. The bold cells indicate the best performing models within 95% confidence interval in that column.

	1-shot 5-way	5-shot 5-way	5-shot 20-way	Epochs
Protonet	70.26 ± 1.11%	87.96 ± 0.68%	76.25 ± 0.42%	100
Gaussian protonet	**76.48 ± 0.74%**	**89.68 ± 0.67%**	**79.97 ± 0.40%**	100
Relation net	74.55 ± 1.09%	87.82 ± 0.70%	**80.22 ± 0.39%**	100
Proxy k-NN	49.84 ± 1.16%	-	-	100
MAML first order	69.62 ± 1.23%	82.22 ± 0.65%	62.25 + 0.48%	100
Reptile	74.08 ± 0.79%	**88.56 ± 0.70%**	66.88 ± 0.50%	100

Table 4.3: Few-shot classification accuracies of the models on the small logo dataset. The confidence interval is calculated by sampling 500 tasks from the meta-test set. The bold cells indicate the best performing models within 95% confidence interval in that column.

4.5 Discussion of empirical analysis

We discuss here the performance results of the models that are trained and tested on the Omniglot and small logo dataset. Furthermore, we answer two research questions from Chapter 2 and discuss with which models we continued our research.

4.5.1 Omniglot dataset performance results

We infer from Table 4.2 that proxy cluster 1-NN performs significantly worse on the five-shot learning task than on the one-shot learning task. We assume this is due to computing the mean of a support set class, because the mean can be skewed by one or more representations in the support set class that are outliers. What is interesting is that this does not seem to be the case with prototypical networks. However, a difference is that prototypical networks are trained end-to-end, including taking the mean of the support set classes, which is not the case with proxy networks. Therefore, we decided to run proxy networks again but with k-NN to see if this works better. We see that this gives significantly better results. However, the improvement is still minor compared to

the other methods. This seems to show that episodic learning gives an advantage when we deal with learning tasks with more than one-shot samples.

We ran SNAIL for both 100 and 500 epochs, because the learning curve showed that the model could learn more. However, we were still not able to get close to the reported results in the paper of Mishra et al., and training the SNAIL model is even slower than MAML and Reptile. We therefore do not use this model further in our research. For the other models, we were confident about our results compared to the published results, so we trained these models on the small logo dataset.

4.5.2 Small logo dataset performance results

In Table 4.3, the performance results on the small logo meta-test set are presented, which also includes Gaussian prototypical networks. We see that Proxy networks with k-NN have poor performance on the meta-test set and on the meta-train set the accuracy is approximately 68% on the set of five-shot, five-way learning tasks. We think that a combination of a skewed meta-train class distribution, as we have seen in Figure 4.3, and many logo classes that have only 10 logo samples per class contribute to the bad performance, because it makes it more difficult for the proxies to learn good class representations. Further research should be conducted to understand why proxy models do not perform in this situation.

We see in Table 4.3 that Gaussian prototypical networks significantly beat Prototypical networks. We also tried the same backbone structure with Prototypical networks to make sure this is not due to an increase in the number of parameters in the last layer. For this, we get accuracy results in the confidence interval of the Prototypical networks model. Hence, Gaussian prototypical networks have significantly better accuracy results. We also see that Gaussian prototypical networks beat Relation networks on two out of three of the learning tasks.

When we compare the learning curves of Gaussian prototypical networks for the sets of five-shot, five-way and five-shot 20-way learning tasks in respectively Figure 4.9 and Figure 4.10, we see some interesting patterns. For the five-shot, 20-way learning tasks, every episodic iteration is harder than with a five-way task, because there are more classes to which an input can be assigned. In addition, a larger batch of images is sampled and learned on every iteration. We see that this is reflected in the training accuracy of Figure 4.10 that increases faster and is higher than in Figure 4.9. However, there is a huge gap between the meta-train and meta-test curve in Figure 4.10, which shows that more data are needed to train good universal logo representations.

For the initialisation method, we see that Reptile is able to outscore MAML significantly. However, both models overfit on the set of five-shot, 20-way learning tasks. We see this in the learning curve of Reptile in Figure 4.12. It shows that Gaussian prototypical networks and Relation networks have better between robustness. On the other hand, Reptile performs well within the confidence interval of Gaussian prototypical networks on the set of five-shot, five-way learning tasks. We see in Figure 4.11 that the train- and test curves are close to each other and maybe more epochs would have benefited Reptile in a higher accuracy.

We see in Table 4.3 that the within-robustness is approximately the same for all methods. This is reflected by the confidence interval for which we cannot say that one method has considerably tighter confidence intervals in all of the three different sets of learning tasks.

We present the comparative results in Table 4.4 and we conclude that Gaussian prototypical networks have the highest accuracy or shares this on every different set of learning tasks and is therefore the best on the accuracy row. For the between robustness, we compared the decrease of accuracy between the five-shot, five-way and five-shot, 20-way set of learning tasks. Here we see that both Gaussian prototypical networks and Relation networks have a decrease of roughly 8-10% accuracy, whereas there is a 22% decrease in accuracy for Reptile. We used the results of flexibility for learning new tasks from Table 3.4 in Chapter 3.

This means we can answer two sub-research questions, *which of the selected techniques have the best accuracy, and are most robust.* We conclude from Table 4.4 that Gaussian prototypical networks have the best accuracy, share the best between-robustness with Relation networks, and have the same within-robustness as the other models.

quality metrics	Sub-quality metrics	Method				
		Protonet	Gaussian protonet	Relation net	MAML first order	Reptile
Accuracy		−	+	±	−	±
Rejection accuracy						
Robustness	between	±	+	+	−	−
	within	+	+	+	+	+
	Quality shift					
Flexible		+	+	+	±	±

TABLE 4.4: Comparative findings on model scale between various models.

4.5.3 Chapter discussion

The goal of the research in this chapter is to make a strategic selection of methods from the three directions. On the basis of accuracy and robustness, we chose one method in every direction to investigate the other quality metrics further. With this information, we are better able to recommend a method that will work in the circle flow and that is able to match the requirements. We did not, however, investigate a model from the model-based learning direction on the logo dataset, because SNAIL did not perform as expected. We therefore chose one model from both the deep metric and initialisation learning direction.

We hypothesised in the previous chapter that deep metric learning can have problems learning universal logo properties for good logo representations compared to initialisation learning. However, we have seen that Gaussian prototypical networks especially beat all other methods with respect to accuracy in all three sets of learning tasks. We also see that Gaussian prototypical networks reach 90% accuracy on the five-shot, five-way set of learning tasks. For the threshold accuracy, the requirement is to reach 90% on a set of five-shot, 35-way learning tasks. Hence, we need to find ways to increase the accuracy.

A problem we have not investigated yet is the robustness to quality shifts between our dataset and data on which the model will make inferences on during deployment of the application. We have seen that the image quality of the small logo dataset has a median of 60^2 pixels, which is much lower than mobile phones are able to make nowadays. Finn and Levine have shown that MAML is better able to handle domain shifts between meta-training and meta-testing compared to recurrent meta-learners, such as SNAIL. We think that Reptile is also more robust to quality shifts than deep metric learning. We have seen that Reptile is more competitive than MAML with respect to accuracy and thus we will investigate if Reptile is able to handle the quality shifts better. This means that we proceed our research with Gaussian prototypical networks and Reptile.

Finally, we still need to investigate the rejection accuracy. There is research conducted by Yang et al. that is strongly in favour of deep metric learning approaches compared to standard neural network classifiers when the rejection accuracy is an important factor. We used standard neural network classifiers with MAML and Reptile. Hence, the question is if this also relates to Gaussian prototypical networks and Reptile.

Chapter 5

Research Gaussian prototypical networks and Reptile

We have seen from our selection of methods that deep metric learning, and specifically Gaussian prototypical networks, has high accuracy on the small logo dataset with better-between robustness than initialisation methods. Gaussian prototypical networks seem most promising with respect to accuracy on all sets of learning tasks. However, we still need to investigate the performance on the rejection accuracy and robustness to quality shifts. We resized the input queries to 60 x 60 pixels; this could remove some features that are important to classifying the logos correctly, so we also investigate if larger input queries have a positive effect on accuracy. When we have answers to these questions, we can answer which direction and method has the most potential to be used as generic model. With this information, we can then obtain estimates of the accuracy and rejection accuracy on a large logo dataset that represents the use case of the mobile application better than the small logo dataset. We do this in the following way: (i) construct a large logo dataset that better represents the use case than the small logo dataset; (ii) investigate if deep metric learning has a better rejection accuracy than standard neural network classifiers by comparing Gaussian prototypical networks with Reptile; (iii) investigate if Reptile is more robust to quality shifts than Gaussian prototypical networks; and (iv) research if larger input queries results in a higher accuracy. Finally, at the end of the chapter, we conduct research into different properties of Gaussian prototypical networks and try to find ways to improve it.

5.1 Introduction of large logo dataset

In Chapter 4, we introduced the small logo dataset to iterate quickly through various methods without too many costs. However, this dataset is limited for our next experiments, such as investigating robustness to quality shifts, and the small logo dataset contains logo samples with tight ROIs that do not reflect the use case, because in reality more background noise will be present in the picture of a logo. We thus focused on constructing a large logo dataset that reflects the use case better.

We define three general logo structures: symbols, text, and a symbol-text combination. In Figure 5.1, we show examples of these three structures. For the small logo dataset, we used ROIs only from the Logos in the Wild dataset, which splits symbol-text combinations into symbol and text samples. This increases the number of classes but does not reflect real use cases. Together with the issue of the tight ROIs, we focused on collecting samples with more background information and samples that include symbol-text combinations.

FIGURE 5.1: Three examples of a possible logo query.

Large logo dataset collection

The first set of samples we collected manually from the internet. Most of these samples originate from conferences in the domain of motor bikes, cars, the cloud, gaming, and logos from buildings. Second, we excluded many classes from the Logos in the Wild dataset because many classes contain fewer than 10 samples. We added more samples for those classes such that they have at least 10 samples. Finally, we included logo classes from the BelgaLogos dataset that do not overlap with the Logos in the Wild dataset. This means that we collected 18,296 logo samples manually and used 2,354 logo ROIs from the BelgaLogos dataset. We present the statistics of the large logo dataset in Table 5.1.

	Omniglot	Mini-ImageNet	Small logo	Large logo
Meta-train classes	4,800	60	151	642
Meta-test classes	1,692	40	40	161
Samples per class	20	600	37*	23*
Resized pixel size of images	28 x 28	84 x 84	60 x 60	60 x 60
Total images	129,840	600,000	14,849	35,499

Table 5.1: Statistics of standard meta-learning datasets and a derived dataset by us, which are called the small- and large logo dataset. The $*$ means that we have used the median as metric.

5.2 Investigation of robustness and rejection accuracy

The model should reject input queries when users of the mobile application take pictures of logos for which the generic model is not prepared for or they make pictures of a scenery that has nothing to do with logos. We, respectively, named these input queries in- and out-domain distribution unknown-class samples. We specified this as the samples from the set $R_{L\setminus t_j}$ and set ODR in Chapter 2. We called the rejection accuracy the proportion of correct rejections of these samples. In this section, we investigate if there are differences between the Gaussian prototypical networks and Reptile method with respect to the rejection accuracy.

The motivation to conduct research on the rejection accuracy is that standard neural network classifiers partition the *whole* representation space and therefore in- and out-domain distribution unknown-class samples can be assigned to a class with very high confidence. We depict what these decision boundaries look like for standard classification networks in the right image of Figure 5.2. We see that there is only low confidence at or close to the white decision boundaries and high confidence in the rest of the representation space. Research conducted by Yang et al. showed that deep metric learning is more robust when distances to cluster are taken into account to define if a query belongs to a class or not, because this does not partition the *whole* representation space. We show this in the left image of Figure 5.2, where the white space represents low confidence scores. Hence, we hypothesise that Gaussian prototypical networks with distance thresholds have a better rejection accuracy than Reptile to both in- and out-domain distribution unknown-class samples.

The second investigation we conduct in this section is robustness to quality shifts. The problem is that our collected logo datasets have different quality samples than will be used during the usage phase of the mobile application. Finn and Levine conducted research to see if initialisation learning is more robust to out-of-domain tasks compared to model-based methods, such as SNAIL. They found that initialisation learning is more robust to these tasks, because they apply within-task learning. SNAIL does not perform

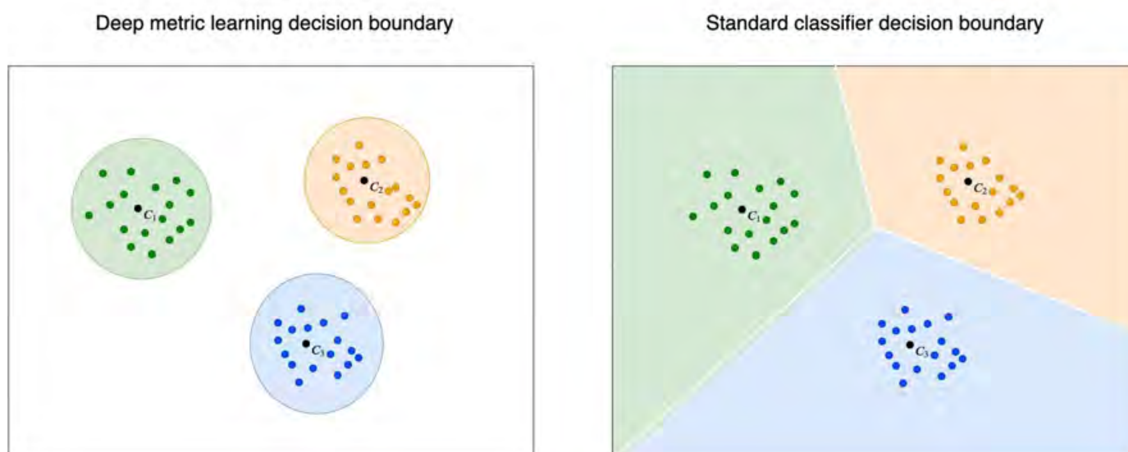


FIGURE 5.2: The decision boundary for two neural network approaches. The left image represents deep metric learning and the right image represent classification neural networks with a softmax output layer. The spaces with colour have high confidence for that colour class. The white spaces represent low confidence for every class.

within-task learning such as deep metric learning approaches. As such, we hypothesise that Reptile is more robust to quality shifts than Gaussian prototypical networks.

Finally, in the previous chapter, we resized the input queries to 60 x 60 pixels, because this is the median quality of the small logo dataset. However, it could be that we are removing important information to classify queries correctly. As such, we research if resizing the training images to a higher quality than 60 x 60 pixels gives a significant accuracy increase. We do this research together with the research on robustness to quality shifts.

5.2.1 Different datasets

To conduct both experiments, we needed different datasets. We have seen in Table 4.3 from Chapter 4 that Reptile is competitive together with Gaussian prototypical networks on the set of five-shot, five-way learning tasks, so we conducted the experiments with this learning task. We used two datasets for the rejection accuracy experiment: we sampled in-domain distribution unknown-class samples from the small logo dataset, and for the out-domain distribution unknown-class samples, we used the mini-ImageNet dataset. For the robustness to quality shifts, we constructed two new datasets from the large logo dataset: a dataset that has high-quality images in the meta-test set and a dataset that has high-quality images in both the meta-train- and meta-test set. For both datasets, the meta-test sets are the same. We present the statistics of both datasets in Table 5.2.

	low and high datasets		high and high datasets	
	meta-train	meta-test	meta-train	meta-test
Samples	14,071	522	9,129	522
Classes	173	40	424	40
Median number of pixels	59.3^2	380.7^2	122.6^2	380.7^2

TABLE 5.2: Datasets for robustness to quality shifts. Low/high and high datasets refer to, respectively, meta-train and meta-test datasets. The median number of pixels specifies the quality of the dataset.

5.2.2 Experimental setup

Rejection accuracy

We used both Gaussian prototypical networks and Reptile models that we trained with the small logo dataset on the set of five-shot, five-way learning tasks in Chapter 4 to investigate performance of the rejection accuracy. To research this performance, we prepared the model first on a task t_j^5 from the set of five-shot, five-way learning tasks. We then sampled a query batch q from $d_{t_j^5}$ that must be accepted (i.e., in-domain distribution known-class samples) and sampled a rejection set from the small logo dataset (i.e., in-domain distribution unknown-class samples) and mini-ImageNet dataset (i.e., out-domain distribution unknown-class samples) that include samples that must be rejected. We called the samples that must be accepted the positive samples and samples that must be rejected negatives samples. We used a receiver operating characteristic (ROC) curve to see which method is more robust to false positives [66].

In Figure 5.3, we show the procedure to collect positive and negative samples. We used a model that has learned task t_j^5 that we denote as H_θ. For Gaussian prototypical networks, this outputs the distances over classes. We replaced this with $P_\theta(y|x_i, S)$ to output the probability distribution over classes for both Gaussian prototypical networks and Reptile. Further, in Figure 5.3, we first sample two distinct tasks t_j^5 and t_k^5 from a distribution of tasks T^5. We then sampled a support set S and query batch q from task t_j^5. We also sampled a rejection set R from the other task t_k^5. We selected five samples per class for sets S, q, and R. Thereafter, we used set S to prepare H_θ to learn task t_j^5 and used batch q to get the distances of every query in batch q to the correct class y. We defined this with function $A(x, y)$ that uses the minimum distance of a query to assign it to a class y. When this assigned class was in agreement with the actual true class, the distance was accepted in the accept set (positive samples). We did this because we only wanted queries that are predicted correctly by the trained model such that the threshold is not influenced by misclassifications.

We used the support set S again to get a set of rejection distances (negative samples). We input a query from set R in the trained model and assigned it to a class in set S for which it has the smallest distance. We did this for every query in R and store the distances in the reject set. We did the exact same procedure for probabilistic sets. However, we then used the maximum probability to assign a query to a class. We repeated the procedure in Figure 5.3 250 times, which means that we sampled 250 tasks and collected a reject set with 7,500 datapoints and an accept set with 7,500 or fewer datapoints, because this depends on how correctly the model assigns every batch q.

Now we can define the true positive rate (TPr) and false positive rate (FPr)[1] for which both are exactly defined in Equation 5.1. A 100% TPr means that all queries from the accept set are accepted, and a 0% FPr means that every query from the reject set is rejected. By varying the threshold, we can inspect with the ROC curve which method is most robust to false positives.

We performed two investigations, one where we sampled reject sets from the small logo dataset and one investigation where we sampled the reject set from the mini-ImageNet dataset. This means that t_k^5 from Figure 2.2 is sampled from a distribution of tasks of the mini-ImageNet dataset. We still prepared the model on a support set and collected the accept set from the small logo dataset.

$$\text{TPr} = \frac{\#\{x|x \in \text{ accept}, x \leq \text{ threshold}\}}{\#\{x|x \in \text{ accept}\}} \tag{5.1a}$$

$$\text{FPr} = \frac{\#\{x|x \in \text{ reject}, x \leq \text{ threshold}\}}{\#\{x|x \in \text{ reject}\}} \tag{5.1b}$$

Robustness to quality shifts and image size

We trained and tested both Gaussian prototypical networks and Reptile on the three defined meta-train- and meta-test sets to investigate the robustness to quality shifts. We did this with sets of five-shot, five-way learning tasks. Thereafter, we compared the results between the different datasets and between both models.

We also trained and tested the Gaussian prototypical and Reptile model on different image sizes. We did this on three different image sizes: 60 x 60, 96 x 96, and 192 x 192 pixels. We used different backbone architectures to keep the dimensions of the representations roughly the same, because larger input queries result in larger representations

[1] Note that the rejection accuracy is $1 - \text{FPr}$.

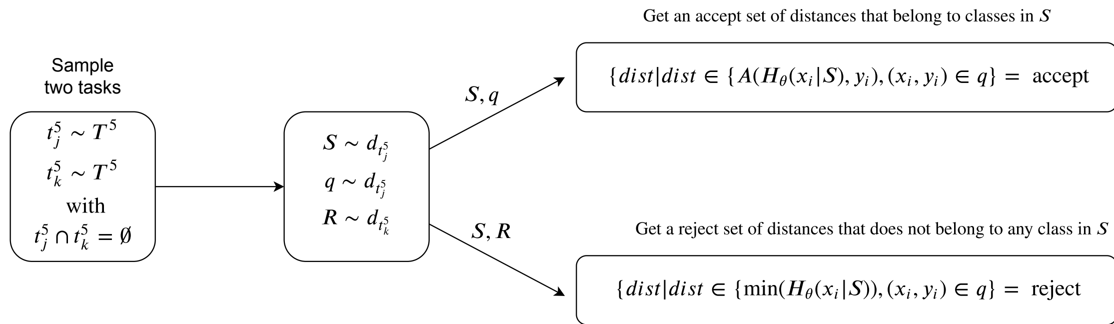


FIGURE 5.3: One iteration of producing positive and negative samples for the ROC curve.

with the standard CNN backbone. We name these architectures 1,2, and 3 and we present the exact details about these structures in Appendix B.1. Finally, we also used the small logo dataset to infer if the choice of architecture has an effect on the accuracy.

5.2.3 Results

In Figure 5.4 and Figure 5.5, the ROC curves of the rejection accuracy experiment are shown with, respectively, the mini-ImageNet and small logo dataset as reject set. In Table 5.3 and Table 5.3, results of the quality shift and image size investigation of, respectively, Gaussian prototypical networks and Reptile are presented.

Quality	Architecture	5-shot 5-way accuracy results		
		Small logo	Low-quality train and high-quality test	High-quality train and test
60x60	1	$89.68 \pm 0.67\%$	$84.83 \pm 0.77\%$	$90.93 \pm 0.43\%$
96x96	2	$87.97 \pm 0.52\%$	$83.14 \pm 0.57\%$	$89.14 \pm 0.61\%$
192x192	3	$88.66 \pm 0.68\%$	$83.14 \pm 0.83\%$	$89.42 \pm 0.64\%$

TABLE 5.3: Robustness results from Gaussian prototypical networks. Every average accuracy and confidence interval is computed from a set of 500 learning tasks.

Quality	Architecture	5-shot 5-way accuracy results		
		Small logo	Low-quality train and high-quality test	High-quality train and test
60x60	1	$88.56 \pm 0.70\%$	$85.44 \pm 0.78\%$	$89.66 \pm 0.61\%$
96x96	2	$87.94 \pm 0.73\%$	$84.85 \pm 0.78\%$	$90.13 \pm 0.58\%$
192x192	3	$89.08 \pm 0.70\%$	$85.70 \pm 0.74\%$	$90.67 \pm 0.59\%$

TABLE 5.4: Robustness results from Reptile. Every average accuracy and confidence interval is computed from a set of 500 learning tasks.

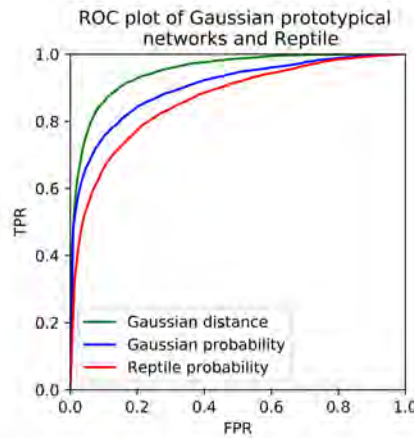


FIGURE 5.4: ROC curve of positive and negative samples collected from the mini-ImageNet dataset.

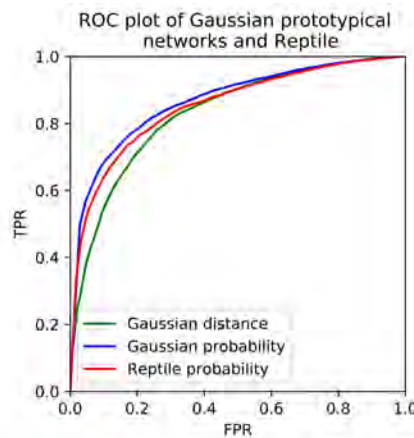


FIGURE 5.5: ROC curve of positive and negative samples collected from the small logo dataset.

5.2.4 Discussion

In Figure 5.4, we can see that the ROC curve with the Gaussian distance has a higher area under the curve and is more robust to false positives than Gaussian probability and Reptile. We also see that Gaussian probability is more robust to false positives than Reptile. This shows that Gaussian prototypical networks are more robust to false positives from out-domain distribution unknown-class samples, such as samples from the mini-ImageNet dataset. However, when we look at Figure 5.5, we see a slightly different trend. Every ROC curve is close together, but we can see that Gaussian distance performs less than both Reptile and the Gaussian probability. This means that for in-domain distribution unknown-class samples, Gaussian probability is the most robust to false positives.

We provide here an intuitive explanation about our previous findings. In Figure 5.6, we show a probabilistic decision plot of a Prototypical networks model with four prototypes;

the decision plot for Gaussian prototypical network looks roughly the same. We see that the uncertainty at the boundaries is wider than with standard classification neural network models. We speculate that in-domain distribution unknown-class samples will be represented somewhere close to the low confidence region (i.e., close or in the red region) and together with the wider uncertainty boundaries it is more robust to false positives from in-domain distribution unknown-class samples. However, we see that this model also partitions the whole representation space. We speculate that out-domain distribution unknown-class samples will be represented more away from the prototypes where there is a larger space with high confidence. Hence, a crude method, such as a distance threshold, is better able to reject out-domain distribution unknown-class samples. Finally, when we compare the robustness in of both figures in Figure 5.5, we see that the Gaussian distance threshold produces relatively more robustness to false positives if we give equal importance to in- and out-domain distribution unknown-class samples.

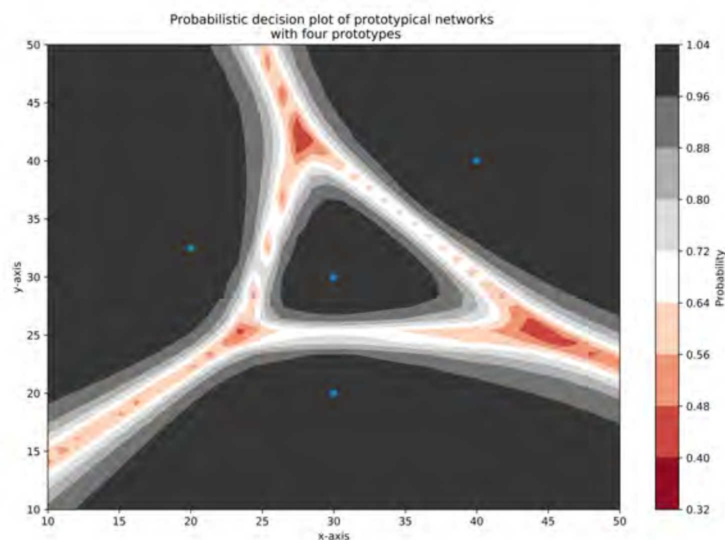


FIGURE 5.6: Prototypical networks two-dimensional representation space with a four-way learning task (i.e., four prototypes) and probabilistic decision boundaries.

We present Table 5.3 and Table 5.4 to research the effect of quality differences between meta-train- and meta-test sets. At first, we investigate the influence of the different architectures. We see that for both models the accuracy is approximately 5-6% lower between the low- and high-quality training sets across the different architectures. This means that we cannot conclude with certainty that Reptile performs better in this case and therefore we reject our hypothesis that Reptile is better able to handle shifts in image quality between meta-train and meta-test datasets. However, we must note that there is a substantial difference between the two datasets in the number of classes if we

look at Table 5.2. Whether this has an effect on our conclusion is research that can be conducted in the future by collecting a larger logo dataset.

For our next experiment, we are interested in seeing if resizing both meta-train and meta-test set samples have an impact on the accuracy. In the previous chapter, we took 60 x 60 pixels per image to speed up training and save costs. We are interested in whether higher image quality is able to increase accuracy. Intuitively, this could be the case, because more information can be used by the model to make a decision. However, in Table 5.3, we see that some datasets perform less and others just as good when we use 192 x 192 pixels and compare it to 60 x 60 pixels. We do see a similar behaviour in Table 5.4. Hence, we conclude that larger image sizes do not increase the accuracy significantly for this meta-learning task.

5.3 Deep dive into Gaussian prototypical networks

We have seen that Reptile is not more robust to quality shifts and it does not have a better rejection accuracy than Gaussian prototypical networks. We therefore focus on improving Gaussian prototypical networks. On the small logo dataset, we obtain 89% accuracy on the set of five-shot, five-way learning tasks for Gaussian prototypical networks, and this accuracy decreases substantially with the set of five-shot, 20-way learning tasks. The requirements state that we have a reliable model when it has 90% threshold accuracy and an 80% rejection accuracy for a set of five-shot, 35-way learning tasks. To do this, we need to investigate if there are improvements to be made for Gaussian prototypical networks. We conduct this research as follows: (i) we investigate which of the two properties of Gaussian Prototypical networks contribute to the performance increase over vanilla Prototypical networks and (ii) we research if data augmentation is able to increase performance. Finally, we conducted more experiments and we present the results of these experiments in Appendix C. They focus on the usage of different loss functions, fine-tuning with the support set, tuning the extended softplus function, and using self-attention to find task specific features.

5.3.1 Experimental setup

There are two properties that define Gaussian prototypical networks: the adjustment of the prototypes with estimated inverse covariance matrices, and using these inverse covariance matrices in the Mahalanobis distance metric. We have seen that Gaussian Prototypical networks have significantly better performance than Prototypical networks on the small logo dataset. We are interested in which of these two properties contribute

most to this performance increase or if they complement each other. Moreover, we investigated if data augmentation helps to increase the number of samples in the dataset and if this improves accuracy significantly.

Model architectures

We conducted research on three different architectures to investigate which properties of Gaussian prototypical networks is actually contributing to the performance increase we have seen in Table 4.3 from Chapter 4. We trained the Gaussian prototypical networks model with only the estimated inverse covariance matrices that we use to adjust the prototypes, so we used the Euclidean distance metric. We named this model the weighted prototypical networks. We then trained the Gaussian prototypical networks only with the Mahalanobis distance metric and did not adjust the prototypes. We named this model the Mahalanobis prototypical networks.

Data augmentation

We applied data augmentation to see if this has a positive effect on the accuracy. Normally, data augmentation is applied to increase the initial training dataset or, in other words, increase the within-class samples, such that test accuracy increases. We previously talked about the importance of invariant representations in Chapter 3; the representation of a class should be as constant as possible, which means that under different lighting, rotation, zoom, background, etc., the class representations from the neural network function are tightly clustered (i.e., low cluster intra-variance). We applied data augmentation to improve the invariant nature of representations. We got our inspiration of data augmentation transformations from Chen et al.. They applied random crop, left-right flip, and colour jitter only in the training phase. We did not apply left-right transformation (mirroring), because this is not how logos are usually presented, and we did not include random crop, because this did not improve performance. We did include random rotation. The exact settings can be found in Appendix A.

For all of the experiments, we used the same hyperparameters that are used in Chapter 4 for the Gaussian prototypical networks on the set of five-shot, 20-way learning tasks, and we used the small logo dataset for every experiment. Last, we also trained vanilla Prototypical networks with the backbone architecture of Gaussian prototypical networks to compare results. We call this network the vanilla+ Prototypical networks.

5.3.2 Results

In Table 5.5, we show the accuracy results of the different experiments.

Method	Augmentation	5-shot 20-way	epochs
Vanilla+ protonet	no	76.41 ± 0.44%	100
	yes	79.79 ± 0.42%	100
Gaussian protonet	no	79.97 ± 0.40%	100
	yes	**81.77 ± 0.39%**	100
Weighted protonet	no	76.37 ± 0.43%	100
	yes	80.16 ± 0.41%	100
Mahalanobis protonet	no	79.74 ± 0.43%	100
	yes	**81.61 ± 0.37%**	100

Table 5.5: Results on the set of five-shot, 20-way learning tasks from the small logo dataset with and without data augmentation.

5.3.3 Discussion of deep dive

We see in Table 5.5 the results of the experiments. When we only look at the results without data augmentation, we see that the vanilla+ and the weighted model have the same accuracy results. This is also the case for the Gaussian and the Mahalanobis model, but they have significantly higher accuracy. It seems that we can conclude that using the estimated inverse covariance matrices with the Mahalanobis distance metric contributes solely to the improvements of Gaussian prototypical networks over vanilla Prototypical networks.

Our next experiment focused on data augmentation. We see that with data augmentation, the accuracies significantly increase compared to not using data augmentation for every model architecture. We see that the Mahalanobis and Gaussian prototypical networks have the same performance with data augmentation, which strengthen our conclusion that the Mahalanobis distance metric solely contributes to the increase in performance over vanilla Prototypical networks. We therefore introduce the Mahalanobis prototypical networks, which are a simpler model than Gaussian prototypical networks but with similar performance. We also conclude that our data augmentation strategy significantly increases performance.

5.4 Chapter discussion

The goal of this chapter was to create a larger logo dataset that resembles the use case better, investigate robustness to shifts in quality, to investigate the rejection accuracy

between Gaussian prototypical networks and Reptile, and research if we can improve Gaussian prototypical networks.

We increased the small logo dataset from 191 classes to the large logo dataset with 803 logo classes and with a total of 35,499 samples. We specifically focused on collecting logo samples with more background information compared to the small logo dataset. However, the quality of the number of pixels per image is still low compared to the pictures mobile phones are able to make. Thus, there is still a quality shift between the large logo dataset and pictures that mobile phones can take.

We have seen that Reptile does not significantly offer more robustness to shifts in quality of images compared to Gaussian prototypical networks. We have seen that training with a lower quality set and testing on a higher quality set decreases the accuracy of both networks with approximately 5-6% compared to training and testing on a higher quality set of samples. We can conclude that it is important to also collect samples that have comparable image quality to what the conference application will see during deployment.

We can answer the question of *'which of the selected techniques have the best rejection accuracy'* with the research in this chapter. We have seen that deep metric learning, and specifically Gaussian prototypical networks, is more robust than Reptile when we use the distance metric to assign queries to classes. We see this behaviour when we query samples that must be rejected from out-domain distribution unknown-classes, such as the mini-ImageNet. However, we do not see the same behaviour when we sample queries that must be rejected from in-domain distribution unknown-classes. In this case, we see that using a probabilistic threshold for Gaussian prototypical networks gives the most robustness against false positives. However, we deem the rejection rate for in- and out-domain distribution unknown-class samples as equally important, because data-driven applications, such as the mobile application of this research, will encounter both, in- and out-domain distribution unknown-class samples, during deployment. Finally, we can conclude that deep metric learning approaches and specifically Gaussian prototypical networks are more robust to false positives and have a higher rejection accuracy than Reptile.

We create Table 5.6 with the collected information from Chapter 4 and this chapter. We can conclude that Gaussian prototypical networks have a higher accuracy, a better rejection accuracy, and are more between-robust and more flexible than Reptile. We investigated Gaussian prototypical networks and we have found that the simpler Mahalanobis prototypical networks has similar performance. In the next chapter, we used the Mahalanobis prototypical networks to see if we can reach the requirements that we defined in Chapter 2 with the large logo dataset. Finally, we investigated if resizing the images to a higher quality for training increases accuracy. We found that resizing to 96

Quality metrics	Sub-quality metrics	Method	
		Gaussian Protonet	Reptile
Accuracy		+	±
Rejection accuracy		+	−
Robustness	between	+	−
	within	+	+
	Quality shift	±	±
Flexible		+	±

TABLE 5.6: Comparative findings on model scale between Gaussian prototypical networks and Reptile.

x 96 or 192 x 192 does not increase accuracy compared to resizing the images to 60 x 60. Hence, we keep resizing the meta-train- and meta-test set samples to 60 x 60 pixels with bilinear interpolation.

Chapter 6

Empirical analysis of Mahalanobis prototypical networks

We have seen that Gaussian prototypical networks perform best compared to a few selected meta-learning methods with respect to accuracy, rejection accuracy, flexibility and robustness. Thereafter, we introduced a simpler method for Gaussian prototypical networks called the Mahalanobis prototypical networks. In this chapter, we investigate if the Mahalanobis prototypical networks are able to meet the requirements we set in Chapter 2, which are a 90% threshold accuracy and 80% rejection accuracy on a set of five-shot, 35-way learning tasks.

6.1 Experimental setup

We used the large logo dataset to get performance metrics on the meta-test set. In this section, we explain how we set up the experiments to estimate these metrics. We first investigated four different backbone CNN architectures and two different learning approaches. We then used a validation set to tune a rejection threshold that maximises the accuracy. We show the statistics of the datasets in Table 6.1. Finally, we estimated the accuracy, threshold accuracy, and rejection accuracy on the meta-test set.

	Meta-train		Meta-test
	Val meta-train	Val meta-test	
Samples	24,168	5,219	6,102
Classes	514	128	161

Table 6.1: The statistics of the training, validation and test set of the large logo dataset. Val is an abbreviation for validation.

Network architecture

In previous experiments, we constantly used a simple backbone that consists of four convolution layers. We saw that this backbone is complex enough to learn the small logo meta-training set well (i.e. the loss goes to zero). However, with the large logo dataset and including data augmentation, we might need a more complex CNN architecture to learn a better representation function. Hence, we tested more complex architectures on the set of five-shot, 35-way learning tasks to see if this produces higher accuracy results.

We trained the models with 200 epochs for every architecture using the same hyperparameters. First, we trained with the backbone (standard) structure we used throughout the previous sections. We then trained on an extended backbone (standard plus 2 conv) that has two extra convolution layers attached at the end with 3 x 3 filters, no max-pooling layers, the first convolution using 64 channels, and the second using 128 channels. We talked in Chapter 3 about some of the advantages of the ResNet model with which we are able to make very deep networks. We tried out the standard ResNet-18 model without the first layer, and name it ResNet-17 [17]. We excluded this layer because its function is to decrease the input size of an input image of 224 x 224 pixels. We were already working with 60 x 60 pixels so did not need to use this layer. Last, we tried out an adjusted ResNet-18 model. This model is called eResNet and was first introduced by Yang et al.. The motivation is that the ResNet models have too many parameters and therefore overfit easily on small training sets. The major difference between the ResNet-18 and eResNet model is that the eResNet model uses pooling layers to decrease the size of the representations. In Appendix A, we give a detailed overview of the hyper-parameters used, and in Appendix B, we give an overview of the ResNet and eResNet architectures.

Learning approach

Snell et al. mentions that training prototypical networks with higher classes (i.e., higher 'K-way') than the model will be used for is beneficial, or in other words, gives higher accuracy. We hypothesise that higher K-way training increases the chance of more difficult tasks. The consequence is that the model converges faster but is also able to learn more which results in higher accuracy. We tested this by training the Mahalanobis prototypical networks on the meta-train set twice but on a set of five-shot, 35-way and five-shot, 70-way learning tasks. Thereafter, we tested both networks on five-shot, 5-, 20-, 35-, 50-, 65-, 80-, and 90-way learning tasks, which allows us to investigate between-robustness.

Threshold research

We used the CNN architecture with the best accuracy from the previous two experiments to define an optimal distance threshold. We used the same procedure in Chapter 5 to find this optimal distance threshold. We defined an optimal distance threshold for the Mahalanobis prototypical networks model with the validation meta-test set of the large logo dataset. We did this by creating a set of positive distances and a set of negative distances. The exact procedure is outlined in Figure 5.3 from Chapter 5. We derived the positive and negative distance sets from a set with 250 five-shot, 35-way learning tasks. This gave us 43,750 data points for every iteration and the same or less for the positive distance set. We tuned the distance threshold such that the FPr is 20%. This is equal to a rejection accuracy of 80%. This means that we maximised the accuracy such that we met the rejection accuracy requirement.

Performance metrics on the meta-test set

At this stage, we used the model with the highest accuracy from the architecture research and trained this model on a dataset that is a combination of both the validation meta-train- and validation meta-test set. We then obtained performance metrics scores on the meta-test set.

6.2 Results

In Table 6.2, we see the accuracy results of the different architectures on the validation meta-test set. In Figure 6.1, we show the plot where we trained two models on a set of five-shot, 35-way and five-shot, 70-way learning tasks from the validation meta-train set, and tested on different sets of learning tasks from the validation meta-test set. In Figure 6.2, we show a plot for which one model is trained on the meta-train set and the other on the validation meta-train set and both are tested on different sets of learning tasks from the meta-test set. We defined an optimal threshold with the validation meta-test set of 25.65 with a model trained on the validation meta-train set on a five-shot, 35-way set of tasks. We used this to get the performance statistics in the second-from-right column in Table 6.3 for the threshold accuracy and rejection accuracy.

Architecture	5-shot 35-way	Epochs
Standard	83.14 ± 0.29%	200
Standard plus 2 conv	79.72 ± 0.31%	200
eResnet	85.75 ± 0.27%	200
Resnet-17	**88.02 ± 0.24%**	200

TABLE 6.2: Average accuracy results on the validation meta-test set with different backone architectures.

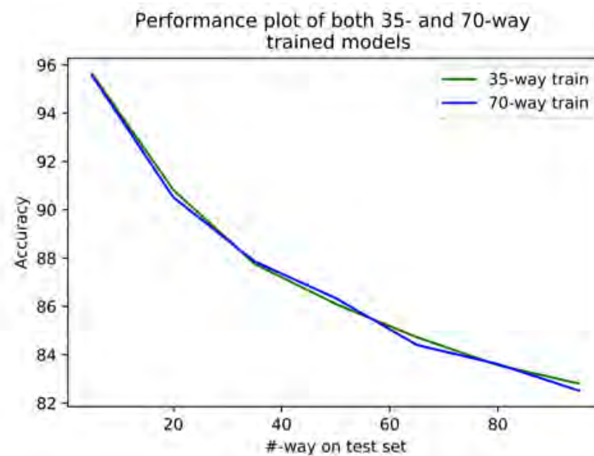


FIGURE 6.1: Two ResNet-17 Mahalanobis prototypical network models, that are trained on the validation meta-train set with 35- and 70-way tasks, are tested on different sets of five-shot learning tasks from the validation meta-test set.

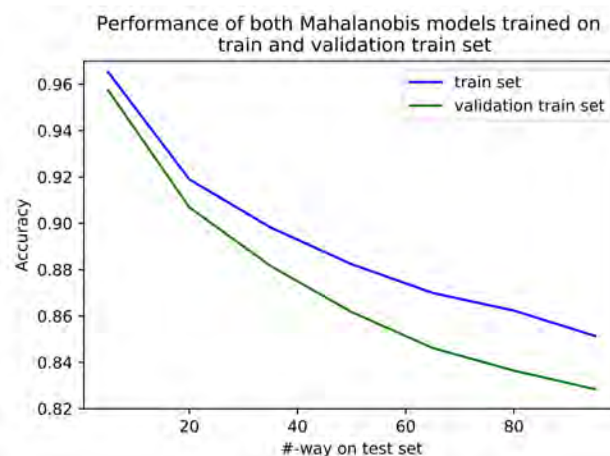


FIGURE 6.2: Two ResNet-17 Mahalanobis prototypical network models, that are trained on the meta-train set and validation meta-train set with sets of five-shot, 35-way learning tasks, and tested on the meta-test set on different sets of learning tasks.

Metrics	5-shot 35-way	5-shot 35-way	5-shot 35-way
Training dataset	Meta-train	Val meta-train	Val meta-train
Testing dataset	Meta-test	Meta-test	Meta-test
Data augmentation	yes	yes	no
Accuracy	89.70 ± 0.32%	88.16 ± 0.41%	84.66 ± 0.37%
Threshold accuracy	77.88 ± 3.62%	-	-
Rejection accuracy	88.21 ± 0.40%	-	-

Table 6.3: Results of the ResNet-17 Mahalanobis prototypical networks on the large logo dataset. The left column shows the final results on the meta-test set. The middle and right column are models trained only in the validation meta-train set and evaluated on the validation meta-test set.

6.3 Chapter discussion

We see clearly that the ResNet-17 model performs significantly better on the validation meta-test set in Table 6.2 than the other architectures. We therefore use the ResNet-17 as backbone for Mahalanobis prototypical networks in the other implementations. It could be that larger ResNet structures are even better, such as ResNet-34, -50, or -101. However, the limitation and costs of GPU memory prohibit us from using these architectures.

Besides training the Mahalanobis model on the set of five-shot, 35-way learning tasks from the validation meta-train set, we also trained the model on a set of five-shot, 70-way learning tasks. Thereafter, we tested the models on different sets of learning tasks from the validation meta-test set. We depict the results from both models in Figure 6.1. We see that both models have comparable performance. This means that there is no increase in accuracy by training the Mahalanobis model on higher K-way learning tasks, and the results from Snell et al. that higher K-way learning increases accuracy is not applicable in this setup.

From the results of the second left column in Table 6.3, we see that we are able to acquire an accuracy of 89.70% with the Mahalanobis prototypical networks model on the meta-test set. Using the distance threshold that we tuned with the validation meta-test set, we are able to get a threshold accuracy of 78%. We also see that the 95% confidence interval increases substantially compared to the accuracy metric. This means that the optimal distance threshold varies substantially per task. Further research can be conducted to see if more data or estimating the distance threshold per task or class is better. We probably would have been able to get higher accuracy by performing a hyperparameter search. However, we did not do this, because of resources and costs. Otherwise, a Bayesian hyperparameter search would have been our choice [68].

In Figure 6.2 shows the meta-test set results on different sets of learning tasks for two models that are trained on either the validation meta-train set or meta-train set. We first looked at the model trained on the meta-train set. What is interesting is that we achieve $91.90 \pm 0.46\%$ accuracy on the set of five-shot, 20-way learning tasks, which is 10% higher than the Mahalanobis model we trained and tested with the small logo dataset with data augmentation in Chapter 5. Hence, our choice of architecture and increase of data has benefited the accuracy tremendously. Moreover, we see a 4.63% decrease in accuracy from five-way to 20-way learning tasks, which is more than a 50% improvement of between-robustness compared to Gaussian prototypical networks in Chapter 4, where we attained a 10% difference. Furthermore, we see that at 95-way tasks, the model achieves $85.14 \pm 0.24\%$, which is a decrease from five-way to 95-way of 11.38% accuracy. Hence, the between-robustness is substantially increased. Second, we look at the effect of the size of training data between both models in Figure 6.2. We see that the difference in accuracy increases between both models when the models need to learn higher K-way learning tasks. Hence, the effect of adding 128 classes to the meta-train set compared to the validation meta-train set makes the model more robust for higher K-way learning tasks.

In Table 6.3, we also show the difference between three model configurations. We included the two right-most columns primarily to investigate the effect of data augmentation and addition of more classes to the meta-train set. We can view data augmentation as a form of increasing the sample size in the already-existing classes, which resembles increasing the within-class samples. Furthermore, we see an increase of 84.66-88.16% accuracy by applying data augmentation, whereas adding more classes to the validation meta-train set gives an increase of 88.16-89.60% accuracy. Hence, it seems that increasing within-class samples has a stronger effect on the accuracy than adding more classes to the large logo dataset.

In the second-from-right column in Table 6.3, we see that the rejection accuracy on the meta-test set is 88%. This means that the tuned validation threshold does not translate perfectly to the meta-test set, because we tuned the threshold at 80% rejection accuracy on the validation meta-test set. We need to conduct further research to see if we can make this more robust if more data are collected. Despite this, we can conclude that the Mahalanobis prototypical networks model is able to achieve the rejection-accuracy quality metric of 80% but is not able to achieve the threshold accuracy of 90%.

Chapter 7

Thesis discussion

In this chapter, we discuss the results obtained so far and reflect back on the quality metrics and requirements we formulated in Chapter 2. We defined four quality metrics with a model journey: accuracy, rejection accuracy, flexibility, and robustness. In agreement with Mobiquity, we respectively defined the following requirements per quality metric: at least 90% threshold accuracy, 80% rejection accuracy, a non-expert is able to prepare the generic model within 6 hours, and the methods should be as robust as possible. We defined these requirements under the situation that the generic model is prepared for conferences with 35 logo classes and using 5 training samples per logo class or, in other words, a five-shot, 35-way learning task.

From our background and literature research, we concluded that meta-learning is the perfect approach for this research problem. We split the techniques from this domain into three directions to select the best performing models in every direction according to literature. We did this to make a better decision of which technique has the potential to meet the quality metrics and requirements we defined. An absence in our approach is that we did not research the 'model-based' direction thoroughly, because our implementation of SNAIL was not able to meet the performances published in literature. However, we hypothesise that model-based approaches are not flexible enough to meet our flexibility requirement, because the models we have observed have the same issue of poor between-flexibility that initialisation learning approaches also have.

Our empirical and literature research on deep metric learning and initialisation learning is strongly in favour of deep metric learning. We see that Gaussian prototypical networks and our introduced Mahalanobis prototypical networks model are performance wise the best across the selected deep metric learning and initialisation learning models based on the quality metrics.

Accuracy

The model improvements, more-complex architecture, data augmentation, and more training data shows in Figure 6.2 that accuracy increases 12% on the set of five-shot, 20-way learning tasks compared to the results of the Gaussian prototypical networks in Chapter 4, where we had 79% accuracy on the same set of learning tasks. However, we only managed to reach a threshold accuracy of 77.88% accuracy on the large logo test set, which means we have not been able to achieve the threshold requirement of 90% accuracy. Despite this, we used only 29,387 samples to acquire these results, which in deep-learning terms is a small dataset.

One extra point of attention is the case of multiple logos for one company. It sometimes occurs that a company has multiple, visually different logos. We are treating these logos as different classes and therefore a situation can occur wherein an input is wrongly classified but is still classified to the correct company. For the mobile application, this is seen as a correct classification, whereas with our defined performance metrics it is not. This means that the acquired accuracy results could be a bit pessimistic compared to the true performance results.

Another point of attention that we have noticed during the collection of the large logo dataset is that logos from the same conferences often share the same properties, or in other words, there are logos that look alike. This means that the task of classifying logos to the correct companies at a conference could be harder than the tasks we randomly sampled from our dataset. This means that the accuracy results could reflect more optimistic results than we have stated for the conference application.

Rejection accuracy

We hypothesised that deep metric learning is more robust to false positives and therefore has a higher rejection accuracy than standard softmax neural network classification models. We proved this hypothesis empirically for Gaussian prototypical networks versus Reptile. We see this in Chapter 5, where Gaussian prototypical networks are more robust to false positives than Reptile and it consequently has a higher rejection accuracy. Interestingly, there is nearly no research of robustness to false positives and rejection accuracies in the meta-learning domain, although this property can be of utmost importance for AI-driven applications.

We have also shown that the distance threshold for Gaussian prototypical networks is more robust to false positives for out-domain distribution unknown-class samples, such

as images from the mini-ImageNet dataset. However, it seems like a crude method, because for in-domain distribution unknown-class samples, the distance threshold does not perform as equally well. We still use the distance threshold, because we show with the ROC curves that the relative improvement in rejection accuracy compared to in- and out-domain distribution unknown-class samples is in favour of the latter.

Robustness

We came across three properties of robustness: within-robustness, between-robustness, and quality-shift robustness. We have empirically shown that all of our selected meta-learning models have roughly the same within-robustness using the 95% confidence intervals in Chapter 4. For the between-robustness, we have empirically shown that deep metric learning approaches are more robust than initialisation learning approaches in Chapter 4. Moreover, we see in Figure 6.2 that the performance on the set of five-shot, five-way to five-shot, 95-way learning tasks decreases from 96.52% to 85.14% accuracy. When we compare the accuracy results to the Gaussian prototypical networks results on the set of five-shot, 20-way learning tasks in Chapter 4, we see that our final implementation has higher accuracy results on the set of five-shot, 95-way learning tasks. Hence, the collection of more data, usage of data augmentation, introduction of the Mahalanobis prototypical networks, and usage of the ResNet-17 model has made the final model substantially more between-robust.

We have observed that most of the samples in the small and large logo dataset do not have the same quality as images that can be taken by a mobile phone. The hypothesis that we formed on the basis of Finn and Levine's work is that Reptile is more robust to learn new tasks with out-domain distribution known-class samples. This would mean that we could prepare Reptile with a support set that exist of out-domain distribution known-class samples and is able to learn to classify these samples well. However, we were unable to agree with our hypothesis, because Reptile did not show more robustness to shifts in quality of images than Gaussian prototypical networks. We saw a constant decrease in performance of 5-6% accuracy with different qualitative datasets for Reptile and Gaussian prototypical networks, showing that Reptile and Gaussian prototypical networks share the same robustness to quality shifts. It should be noted that there are some differences between the datasets that can have an influence on our results, such as the number of classes. Future research could focus on collecting a larger dataset from which more equal datasets can be sampled. Furthermore, the issue of robustness to quality shifts can also have an impact to our performance results, because we have not been able to test on a dataset that exactly matches input queries that occur during

deployment of the conference application. We expect that there will be a decrease in performance because of our research in robustness to quality shifts. However, we do not know exactly how influential this will be.

Flexibility

Flexibility is important to make the application more scalable. We found in Chapter 2 that deep metric learning has the best between-flexibility and shares the same performance with the other techniques on the within-flexibility quality metric. We hypothesise that deep metric learning is flexible enough to meet the requirements to prepare a generic model for any task within 6 hours by a non-expert. We have not conducted a formalised research to test this out, but from our training runs, we are confident that deep metric learning methods, such as Mahalanobis prototypical networks, have enough within- and between-flexibility to be within the proposed requirements.

7.0.1 Future research

The first item for future work originates from the shortcoming of the threshold accuracy. We have seen from Table 6.3 that increasing the within-class samples with data augmentation gives an accuracy increase of 3.5% and adding 128 additional classes increases accuracy by 1.5%. Collecting more within-class samples seems to be more beneficial than collecting more classes at this stage. Research by Fehervari and Appalaraju with a deep metric learning model on classification of tight ROI samples showed a 97.16% accuracy on a set of five-shot, 49-way learning tasks with a dataset of 242 logos and 700 tight ROI samples per logo, which is 169,400 samples in total. This shows the importance of within-class samples and shows that increasing our dataset with more within-class samples from 38,000 to, for example, 380,000 samples provides a great deal of potential for higher performance. Further research could be conducted on what the effect is of more data on the threshold accuracy.

Second, we noticed a negative impact on accuracy by the absence of high qualitative training images that reflect the input queries during deployment of the application. Therefore, future research should not only be aiming to increase the large logo dataset with more within-class samples but also focus on collecting qualitative data that represents the input queries during deployment of the application.

Last but not least, we have not been able to conduct precise research on the flexibility of the Mahalanobis prototypical networks approach. We hypothesise from our acquired

experience during this research that deep metric learning approaches are capable of preparing any learning task within 6 hours by a non-expert. However, an official research could be designed in the future to investigate if our hypothesis is correct. A way to do this is, for example, by creating a demo with Amazon Web Services[1].

[1] https://aws.amazon.com

Chapter 8

Conclusion

The goal of this thesis is to answer the following main research question: *Is it possible to build a flexible and reliable 'generic' deep learning logo recognition model that is able to classify logo classes from pictures with only five training samples?* We have stated the following requirements for the terms *reliable* and *flexible* under the assumption that the generic model is mostly prepared for conferences with 35 logos from organisations that will be present: at least 90% threshold accuracy, 80% rejection accuracy, as robust as possible, and a non-expert is able to prepare the generic model within 6 hours.

We investigated the research question by breaking it down into three sub-questions: (i) what technique(s) fit in the circle flow and have potential to be reliable and flexible, (ii) how should we collect data to make the generic model reliable, and (iii) how much data do we need to collect to match the requirements of reliability for the generic model. We answer these sub-questions first in the next three paragraphs before we answer the main research question.

We observed from the literature that meta-learning techniques have the most potential to fit in the circle flow that we defined in Chapter 2 and to meet the requirements of flexibility and reliability. However, many different techniques exist in the meta-learning domain and accuracy is often the only quality metric that is used, although multiple factors play an important role in AI-driven applications, such as rejection accuracy, within- and between-robustness, and within- and between-flexibility. We therefore looked at these quality metrics to decide which techniques in the meta-learning domain are a good fit by constructing a small and large logo meta-learning dataset. We concluded that, from our selected techniques, deep metric learning and especially Gaussian prototypical networks have better accuracy, between-robustness, rejection accuracy, and between-flexibility. Moreover, we empirically showed that the improvement in Gaussian prototypical networks accuracy over prototypical networks is attributed to the use of

the Mahalanobis distance metric. Hence, we introduce the Mahalanobis prototypical networks and we trained and tested it on the large logo dataset to investigate its potential. This resulted in a 77.88% threshold accuracy and 88.21% rejection accuracy on a set of five-shot, 35-way learning tasks from the large logo test set. This shows that the Mahalanobis prototypical network model has a higher rejection accuracy than the required 80%, has the best robustness or shares this with the other selected techniques, and is hypothetically flexible enough to learn logo classification tasks within 6 hours.

We created a small logo meta-learning classification dataset from using already existing logo datasets. These datasets are purposely built for logo detection and we observed that extracting ROIs that match the use case of the conference application is hard, such as having enough background information, within-class samples, and classes. We therefore created a large logo dataset in which we succeeded to increase the total number of classes from 191 to 803 and we introduced many samples with more background information. We recognised, however, that there is a quality shift between the large logo data samples and query images that will be processed during deployment of the conference application, which can negatively impact the performance of the model. Furthermore, we see that the large logo dataset especially benefits from having more within-class samples, whereas increasing the number of classes also has a positive but less of an impact on the performance metrics. Hence, increasing our created large logo dataset to improve reliability of the generic model further, one should focus on adding highly qualitative samples that resemble images from mobile phones and collecting more within-class samples.

The large logo dataset consists of only 35,499 samples from which we trained the final Mahalanobis model with 29,387 samples. We have not been able to reach the 90% threshold accuracy, although we have reached this with a score of 89.70 $\pm$ 0.32% for the accuracy metric. The question is, how much more data needs to be collected to reach the requirement of 90%? We have argued that our large logo dataset is still small in comparison with standard deep learning datasets, such as the Omniglot and mini-ImageNet datasets that contain, respectively, 129,840 and 600,000 samples. We already obtain good results with the large logo dataset and conclude that there is potential to reach the 90% threshold accuracy when a magnitude more within-class samples is collected for the large logo dataset.

On the basis of the above answers and research, we have seen that deep metric learning approaches and especially the Mahalanobis prototypical network model can match every defined requirement and performs well on the defined quality metrics. We can therefore conclude that it is possible to make a flexible and reliable generic model that is able to learn different logo classification tasks with only five training samples per class.

Appendix A

Hyperparameter settings

For every implementation in the training stage where we conducted n-shot, K-way learning, we used the meta-train set to select n support samples for each class K, and we sampled n query samples per iteration for each class K. We used the same sample-collection procedure for the meta-test set during testing.

A.1 Deep metric learning

The hyperparameters for every deep metric learning model are shown in Table A.1, and they are the same for the Omniglot and both logo datasets. Moreover, for every model, we used the Adam optimiser [69] with standard beta hyperparameters (i.e., $\beta_1 = 0.9$ and $\beta_2 = 0.999$), and used a discrete learning rate (LR) adjustment protocol. This means that we adjusted the learning rate every 20 epochs by a half.

Model	Hyperparameters			
	Distance	LR	Loss function	Batch size
Protonet	Euclidian	0.001	Cross-entropy	-
Gaussian protonet	Mahalanobis	0.001	Cross-entropy	-
Mahalanobis protonet	Mahalanobis	0.001	Cross-entropy	-
Relation network	Euclidian	0.001	Mean squared error	-
Proxy network	Euclidian	0.001	Cross-entropy	32

Table A.1: Hyperparameter settings of deep metric learning models used throughout the research.

A.2 Initialisation learning

MAML

We used for all implementations of MAML the first-order approximation, and the Adam optimiser for the outer step (meta-step) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We used stochastic gradient descent for the inner step. The hyperparameters are shown in Table A.2. Moreover, we used the same hyperparameters across learning tasks for both Omniglot and logo datasets with one exception. For Omniglot, we used the hyperparameters of the one-shot, five-way column from Table A.2 on both one-shot, five-way and five-shot, five-way learning tasks. Finally, we used the same learning rate scheduler for the outer LR with the same settings which we used for deep metric learning.

Hyperparameters	1-shot 5-way	5-shot 5-way	5-shot 20-way
Adam LR	0.001	0.001	0.001
Inner iterations	1	5	5
Inner LR	0.4	0.01	0.01
Meta-batch size	32	8	3
Test inner iterations	3	10	10

Table A.2: Hyperparameter settings for MAML that are used throughout the research.

Reptile

In Table A.3, we show the hyperparemeters of Reptile. We used the same hyperparameters per n-shot, K-way learning task for training with the Omniglot and logo datasets. We linearly annealed the outer step LR to zero, just as is published by Nichol et al.. Furthermore, we used the same beta hyperparameters for the Adam optimiser across all learning tasks and datasets with $\beta_1 = 0$ and $\beta_2 = 0.999$. Finally, we only used the Adam optimiser for the inner step.

Hyperparameters	1-shot 5-way	5-shot 5-way	5-shot 20-way
Adam LR	0.001	0.001	0.001
Inner batch size	10	10	20
Inner iterations	5	5	10
Train shots	9	9	9
Outer step LR	1	1	1
Meta-batch size	5	5	5
Test inner iterations	50	50	50
Test inner batch size	5	5	10

Table A.3: Hyperparameter settings of Reptile used throughout the research. Train shots refer to the number of samples per support set class.

A.3 Model based learning

SNAIL

We used the Adam optimiser with standard beta hyperparameters to learn the parameters together with a LR of 0.001. For training, we used a training batch size of 32, and for testing, we used a batch size of 25. Finally, we did not use a step scheduler to adjust the learning rates.

A.4 Data augmentation

For every training procedure, we resized the logo images to 60 x 60 pixels with bilinear interpolation. We also normalised every logo image before forwarding it into the model to a range between $[-1, 1]$.

Data augmentation settings

When we used data augmentation, we added jittering and random rotation. For image jittering, we used the exact same settings and code from this link, which is code that was used by Chen et al.. For random rotation, we rotated the images randomly between $\pm 20°$.

Appendix B

Backbone architectures

B.1 Research of robustness to quality shifts and image size

We used three different CNN architectures during the research of robustness to quality shifts and image size. We explain in this section the architectural design of architecture 1, 2, and 3.

We used a standard CNN backbone that consists of four modules throughout the research. We explained the structure of it in Chapter 4 and it consists of four stacked modules, for which each has a 3 x 3 convolution with 64 filters, and one padding. We abbreviate this as follows: 4 x module[3], and padding=0. Now, we show the three different architectures in Table B.1.

Architecture	Design
Standard	4 x module[3], padding=1
1	4 x module[3], padding=1
2	1 x module[7] + 3 x module[3], padding=0
3	1 x module[7] + 4 x module[3], padding=0

TABLE B.1: Architectural design of three different models, and the standard backbone model.

B.2 The ResNet architecture

We used the ResNet-17 architecture as backbone in Chapter 6. We adjusted the ResNet-18 architecture from He et al. to fit the logo sizes of 60 x 60 pixels better. This means that we excluded the first layer only and so we created a ResNet-17 model. We show the architecture in Table B.2. A block consists of two convolution layers, for which both

convolution layers are followed by batch normalisation. Thereafter, it follows with a ReLU activation function in the activation layer. In Figure 3.7, we show how a block is precisely sorted with these layers.

Layer	Filter size	Channels
Block 1 and 2	3 x 3	64
Block 3 and 4	3 x 3	128
Block 5 and 6	3 x 3	256
Block 7 and 8	3 x 3	512
Average pool	7 x 7	512

Table B.2: Architecture of the ResNet model that we used.

B.3 The eResNet architecture

The major difference between ResNet and eResNet is the use of max-pooling. This minimises the number of parameters to prevent overfitting. We used the exact same structure that Zheng et al. proposed.

Appendix C

Additional experiments

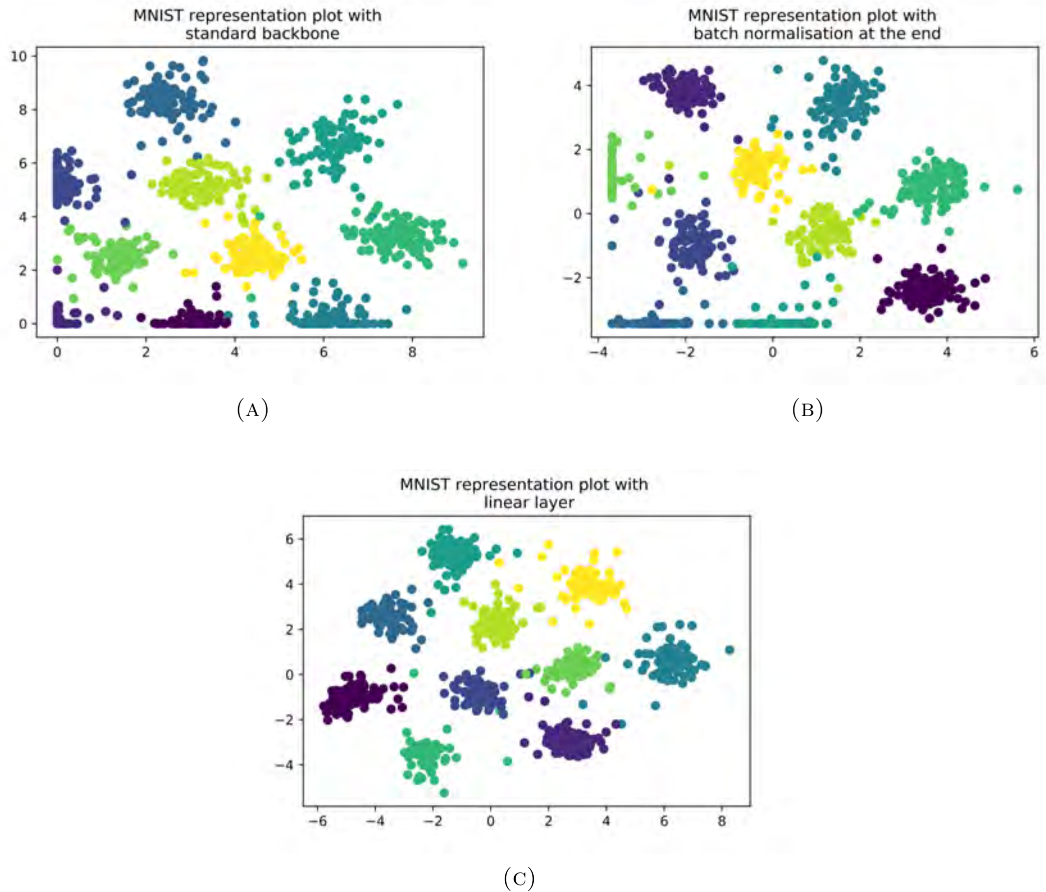
In this appendix, we explain and show results of additional experiments we conducted.

C.1 Looking at the representation space with MNIST

Prototypical networks maps task depended classes to a mixture of spherical Gaussians in an n-dimensional space or, it tries to do this. We use MNIST to represent its classes in a two-dimensional representation space with Prototypical networks. There is a peculiar observation when we look at the standard backbone structure that is used throughout this thesis, and is constantly used in the meta-learning domain. A module of this backbone consist of a convolution, batch normalisation, ReLU and max-pooling. The ReLU function maps every input to a positive quadrant in representation space, which can influence the structure of the Gaussian densities the model tries to create. There is some debate about the ordering of these layers by Mishkin et al., Ioffe and Szegedy. Kruspe moves the batch normalisation layer to the end of every module hoping this improves the Gaussian representations. Hence, we conduct an experiment to assess the behaviour of Prototypical networks and of different backbone architectures.

Experimental setup

We used the MNIST dataset that consist of 10 number classes ranging from 1-10. We trained a prototypical network model with three different backbones on the five-shot, 10-way learning task with 100 epochs. Every backbone maps the output to a 2-dimensional representation. As such, we can investigate the class representations from the MNIST test set. We do this by sampling 1,000 samples randomly. Furthermore, we used the standard train- and test-set split from MNIST that consist of grey-scale samples that



(A) (B) (C)

FIGURE C.1: The three different representation spaces of backbone one, two, and three from 1,000 randomly sampled MNIST test set samples.

have 28 x 28 pixels. Moreover, the first backbone is the standard backbone but with the last layer only having two output channels, the second backbone is almost the same but moves the batch normalisation layer to the end of every module, the third backbone is the same as the standard backbone including a linear layer at the end with a 2-dimensional output. Finally, we used the same Prototypical networks hyperparameters presented in Appendix A.

Results

We show three figures wherein we show randomly sampled representations from the 10 classes of the MNIST test set. In Figure C.1a, we show the representation space of the first backbone which reached a train accuracy of 99.7%, and a test accuracy of 97.5%. In Figure C.1b, we show the representation space of the second backbone which reached a train accuracy of 99.8%, and test accuracy of 97.7%. Last, In Figure C.1c, we present the representation space of the third backbone which reached a train accuracy of 99.7%, and test accuracy of 98.7%.

Discussion

We aim to see how Prototypical networks behaves in the representations space with different backbones. We also hypothesised that the order of layers in the standard backbone module can have an impact on the assumption of Gaussian densities. In Figure C.1a, we see the representation of the 10 MNIST classes in representation space. We observe that the classes are represented in a positive quadrant which influences the shape of certain classes at the edge dramatically. We conclude visually that these classes are not Gaussian shapes, whereas we conclude that the other classes do represent Gaussian shapes. Furthermore, we look if switching the order has an impact on the shapes of the representations. In Figure C.1b, we show the representation space for which we used a backbone that has the batch normalisation layer at the end of every module. We see that this change does not improve the Gaussian behaviour of certain classes, and we can even argue that it looks worse than the standard backbone. It seems that batch normalisation has only shifted and squeezed the representations, which is actually the intended purpose of batch normalisation. Finally, in Figure C.1c, we see an improvement compared to the other two backbones. We visually conclude that every class is represented as a Gaussian density. It also seems that the intra-class variance is lower compared to the other two backbones, although the backbone we used is more complex by adding a linear layer, which can have an impact on creating better invariant representations.

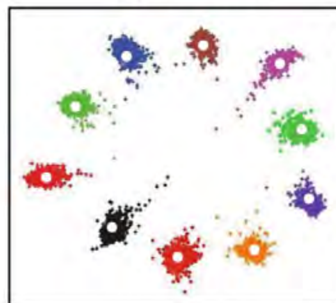
Using the discussion in the previous paragraph, we can conclude that adding a linear layer at the end of the backbone improves the assumption of Gaussian densities for Prototypical networks. We can also conclude that re-ordering batch normalisation to the end of every module does not have any impact compared to the Gaussian shapes of the standard backbone. However, if we compare accuracy results between the three backbones, we do not see much fluctuation. This could be due to using an easy computer vision task, such as MNIST. Hence, further research should be conducted to see what the effect is on harder tasks, such as the logo classification task in this thesis. However, we keep using the standard backbone in this thesis, because it is used by the meta-learning community, so we, and the community, are able to compare performance results between various techniques.

C.2 Using different loss functions to decrease intra-class variance and increase inter-class variance

One of the first deep metric learning model was introduced by Koch et al. and uses a siamese CNN structure to do one-shot, two-way learning. They used a standard sigmoid output function with the cross-entropy loss (i.e., the softmax loss). However, Wen et al. mentioned the importance of discriminative representations (i.e., low intra-class variation and high inter-class variation.) and they proposed the center loss. In Equation C.1, we show this loss for which d is the Euclidian distance, and it is jointly constructed with the softmax loss. The W's and b's are trainable weights. The c in the center loss computes a within-class representation mean, such as we calculate with Prototypical networks (see Chapter 4 for more details). The λ is a hyperparemeter that can be tuned to balance the softmax and center loss. In Figure C.2, we show the comparative behaviour of the joint supervision of the softmax and center loss using different λ's. We clearly see that the intra-class variance is decreasing when λ is increased to 0.1.

$$\frac{1}{mK}\left[-\sum_{i=1}^{mK}\log\left(\frac{e^{W_{y_i}^T h_\theta(x_i)+b_{y_i}}}{\sum_{k=1}^{K} e^{W_k^T h_\theta(x_i)+b_k}}\right)+\frac{\lambda}{2}\sum_{i=1}^{mK} d(h_\theta(x_i)-c_{y_i})\right] \tag{C.1}$$

(a) $\lambda = 0.001$



(c) $\lambda = 0.1$

FIGURE C.2: Representation space of a network trained on MNIST with the joint supervision of softmax and center loss. Different λ's are used to show the effect, and the white dots represent c_{y_i}. Figure from Wen et al..

For Prototypical networks, we have the loss function defined in Equation C.2. We rewrite this loss with some basic algebra into two parts in Equation C.2. Interestingly, the first part is exactly the center loss function. Furthermore, a thought experiment indicates that minimising this loss decreases the intra-class variance by the first part, and consequently increases inter-class variance by the second part. Thus, this loss function that we used throughout the experiments in the various deep metric learning models seems to be a good choice. However, over the years, many loss functions are introduced in deep metric learning, such as the triplet loss [56]. We use some of them together with Gaussian prototypical networks to see if we are able to get even better results. We do not go into detail of every loss function but we refer to literature instead.

$$-\frac{1}{mK}\sum_{i=1}^{mK}\log\left(\frac{e^{-d(h_\theta(x_i),c_{y_i})}}{\sum_{k=1}^{K}e^{-d(h_\theta(x_i),c_k)}}\right)=\frac{1}{mK}\sum_{i=1}^{mK}\left[d(h_\theta(x_i),c_{y_i})+\log\left(\sum_{k=1}^{K}e^{-d(h_\theta(x_i),c_k)}\right)\right] \tag{C.2}$$

Experimental setup

We used Gaussian prototypical networks and three different loss functions: triplet loss, mixture loss [70], and max mixture loss [70]. We added the triplet loss to the standard Prototypical network loss. The mixture loss function was introduced to be a better function than the standard prototypical network loss function.

We tested the three different loss configurations with Gaussian prototypical networks on the sets of five-shot, five-way and five-shot, 20-way learning tasks. We derived these tasks from the small logo dataset. We used the standard hyperparameters for Gaussian prototypical networks defined in Appendix A. Finally, we computed a 95% confidence interval by sampling 500 tasks for both learning sets.

Results

We show the results of the different configurations on the small logo test set in Table C.1.

Discussion

In Table C.1, the different results are presented. We see that adding the triplet loss gives worse results than standard Gaussian prototypical networks. For both mixture

Loss function	5-shot 5-way	5-shot 20-way
Standard	**89.68 ± 0.67%**	**79.97 ± 0.40%**
Triplet	87.77 ± 0.75%	-
Mixture	**89.80 ± 0.64%**	**79.49 ± 0.42%**
Max mixture	**90.30 ± 0.63%**	**79.35 ± 0.42%**

TABLE C.1: Accury results of the different loss function configurations. Every configuration is trained with 100 epochs.

losses, we see comparable performance. This means that we have not found evidence that the mixture loss provides better performance than the standard loss. Thus, none of the investigated losses actually improves performance. Hence, we kept using the original loss in all of our experiments.

C.3 Fine-tuning with the support set

In Chapter 3, we discussed that initialisation learning performs within-task learning, and deep metric learning does not do this. However, there is literature were they do within-task learning with the support set, which is called fine-tuning. Vinyals et al. use the support set to tune the parameters of a deep metric learning backbone with a few gradient steps. They did not, however, see a significant increase but another study, conducted by Zheng et al., had a significant increase on the mini-ImageNet dataset with 9% accuracy on the set of five-shot, 20-way learning tasks. They say that most of this increase is contributed to using the eResNet model (see Appendix B.3) but they do not specifically explain how they used fine-tuning. We used the standard backbone to see if we can also get a significant increase in accuracy on the small logo dataset.

Experimental setup

Zheng et al. used the prototypical networks model as a foundation for their research. We used Gaussian prototypical networks for our research together with the eResNet backbone. They are, however, not completely clear how they fine-tuned the model. What is clear, is that they used the support set S to perform one gradient step to tune the backbone parameters.

In our experiment, we trained prototypical networks on the small logo dataset with the same hyperparameters in Appendix A. When we tested the model, we first fine-tuned the backbone. We did this by using the support set S to create the centroids per class K, and then we also used the support set S as query batch. We then performed gradient

descent with one or three steps to update the parameters in the backbone. Thereafter, we used the centroids and the normal query batch to get evaluation metrics for the task.

We used a few different configurations such as freezing a few layers, which means that these layers are not updated by fine-tuning. We also changed the learning rate of the fine-tuning a bit to see if this has effect. Finally, we tested this approach on the set of five-shot, five-way learning tasks.

Results

In Table C.2, we show the results of different configurations from the fine-tuning experiment.

Settings	Learning rate	5-shot 5-way
Freeze zero layers	0.001	89.91 ± 0.61%
Freeze zero layers	0.0001	89.70 ± 0.70%
Freeze all but last two layers	0.001	86.69 ± 0.77%
Freeze zero layers and three gradient steps	0.001	89.70 ± 0.67%

TABLE C.2: Fine-tuning results on the small logo dataset with prototypical networks and standard backbone.

Discussion

In Table C.2, we see the results of fine-tuning with the standard backbone. Comparing to the Gaussian prototypical networks results without fine-tuning, we see that none of the configurations have a significant increase in accuracy. Hence, we can conclude that fine-tuning with our approach does not work. However, we did not use the eResNet backbone, and it could be that we perform the fine-tuning differently than by Zheng et al.. Thus, future research should be conducted to see if results will be different using the eResNet backbone.

C.4 Different settings for the softplus function

Fort proposes to use the softplus function to ensure a PD matrix for the Mahalanobis distance metric. They specifically propose to use the function $S = 1 + \text{softplus}(x)$ to limit the domain to $S > 1$ and thus guaranteeing that the data points can only be less important. However, in the sense of a PD matrix that maps the inverse variance, the

domain should be $S > 0$, which can be achieved by only using the softplus function. Another way is to use trainable parameters, such as $S = \text{offset} + \text{scale} * \text{softplus}(x/\text{div})$. Hence, in this experiment, we tried out different implementation of the softplus function for Mahalanobis prototypical networks.

Experimental setup

We used the Mahalanobis prototypical networks model with the same hyperparameters in Appendix A to train with different configurations of the softplus function. We saw in Appendix C.1 that the ReLU function maps every output from the representation into a positive quadrant, and thus it is not necessary to use the softplus function to get a PD matrix. Hence, we replace the softplus function with $x + \epsilon$, for which $\epsilon = 1e - 10$. To still make sure that data points are only less important, we can replace the softplus function with $1 + x$, and this is the second configuration we tried. Finally, we used the trainable softplus function. We initialised the trainable parameters with 1.0 and trained it end-to-end with the Mahalanobis prototypical networks.

We trained the model on the set of five-shot, 20-way learning tasks from the small logo dataset. We used data augmentation for training, which is the same data augmentation and settings we used in Chapter 5.

Results

In Table C.3 the results of the experiments are shown.

Function	5-shot 20-way	Epochs
Trainable	$80.14 \pm 0.38\%$	100
$x + \epsilon$	$80.91 \pm 0.41\%$	100
$1 + x$	$81.79 \pm 0.41\%$	100

Table C.3: Different softplus function configuration experiment results with Mahalanobis prototypical networks. The confidence intervals are computed from 500 tasks from the small logo dataset.

Discussion

The Mahalanobis prototypical networks model on the set of five-shot, 20-way learning tasks achieved an accuracy of 81.60%, with data augmentation and the standard softplus function. In Table C.3, we see that only the $1+x$ function has comparable results within the confidence interval. Curiously, the trainable function has the worst results. It should, however, be able to converge to a least the $1+x$ function. It could be that longer training

is required. However, we do not see any improvement compared to the standard softplus function, and thus we kept using it throughout the experiments.

C.5 Using self-attention to find task specific features

The Mahalanobis and Gaussian prototypical networks construct an inverse variance matrix by adding support set representations from class K. This seems to be working well, although Oreshkin et al. show that scaling the Euclidian distance metric by a learnable parameter increases the performance of Prototypical networks significantly. This raises the question if Mahalanobis and Gaussian prototypical networks are truly estimating an inverse covariance matrix for every class density or, only learn a constant set of parameters that scale the distance metric. We conducted an experiment to estimate the inverse covariance matrices from the support set representations directly. This gave some challenges, such as values that approaches infinity after taking the inverse of the matrix, because some values lay close to zero. Another idea was to create a neural network function that computes the inverse covariance matrices by taking in the complete set of support set representations.

The published work by Li et al. is another way of thinking that inspired us. Most deep metric learning approaches use the support set classes independently to decide if a query belongs to a certain class. However, depending on the task, certain dimensions (features) from representations are more important than others. For example, a car can be represented by the following features: number of wheels, number of doors, colour, and shape. When a meta-learning model must prepare on a task in which it should distinguish between a Mercedes or Toyota, the dimension that specifies the number of wheels does not add any additional information for the task at hand, because both cars have four wheels. However, when it must distinguish between a truck and a car, the number of wheels dimension does add additional information for the task, because a truck has more than four wheels. Hence, in Li et al., they proposed an additional neural network function that finds important dimensions using CNNs.

We propose a neural network function that finds important dimensions and computes a covariance matrix that can be used in the Mahalanobis distance metric. We name this function the covariance function, and use it together with prototypical networks. The main idea is to use self-attention and feed forward neural networks to, respectively, mask non-relevant dimensions and compute the variances per dimension.

Method

In Chapter 3 and Chapter 4, we defined that meta-learning methods are trained with episodic training. This means that we sample every iteration a support set $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ and batch $q = \{(x_1, y_1), ..., (x_M, y_M)\}$ where each $x_i \in \mathbb{R}^D$ is a D-dimensional representation of an input image, and $y_i \in \{1, ..., K\}$ specifies the class. We denote S_k and q_k as the set of logo samples that belong to class k, with $k \in \{1, ..., K\}$. We denote n, from n-shot, as the number of samples in every set S_k.

Prototypical networks learn a representation function $h_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^E$ that outputs an E-dimensional representation. It uses this representation function to create prototypes $c_k \in \mathbb{R}^E$ from support set S that represent the centroid of a cluster from class k. More information about prototypical networks can be found in Chapter 4.

For a n-shot, K-way learning task, we have a support set $S : (nK, 3, d_1, d_1)$ to prepare the task, in which d_1 is the dimension of the input image with $3 * d_1 * d_1 = D$ and three represent the RGB dimensions. The nK value represents the total number of samples. We put the support set through function g_θ, and get the representation set $R : (nK, d_2)$, in which d_2 is an E-dimensional vector for all samples. We use the representation set to compute both the masks, and covariance matrix with the covariance function.

Self-attention [63] creates three vectors from an input vector: the query (q), key (k), and value (v). These are created by forwarding the input vector in three distinct feed forward neural networks. The key and query are used to see what dimension is relevant by computing a dot product or, element-wise multiplication. These scores per dimension are forwarded into a softmax function to get relevant scores between $(0, 1)$. These scores are then multiplied by the value vector to give relevancy to certain dimensions.

We use feed forward neural networks to compute vectors q, k, and v. For the within-mask function, the input and output should be n-dimensional vectors so an output layer with n nodes. For the between-mask function, the input and output should be K so an output layer with K nodes. One or more hidden layers can be used in every neural network.

In Figure C.3, we show the architecture of the covariance function. First, the representations are forwarded into the within-mask function. In Algorithm 2, we show the procedure. Here it computes the dimensions that are important for the class. Second, the output from the within-mask function is forwarded to the between-mask function, for which we show in Appendix 3 how it works. Here it computes the dimensions that are relevant between the classes, and it outputs a mask per dimension. Third, the representations are also forwarded into a covariance function $cov_{\theta_5}(x)$, which is

a feed forward neural network, and outputs inverse covariance matrices per class by $R : (K, n, d_2) \xrightarrow{cov_{\theta_5}(x)} c : (K, d_2)$. The output is element-wise multiplied with the masks from the output of the between-mask function by $c \circ f$. As such, only relevant dimensions for the task are selected. Finally, we use the masked inverse covariance matrices in the Mahalanobis distance metric.

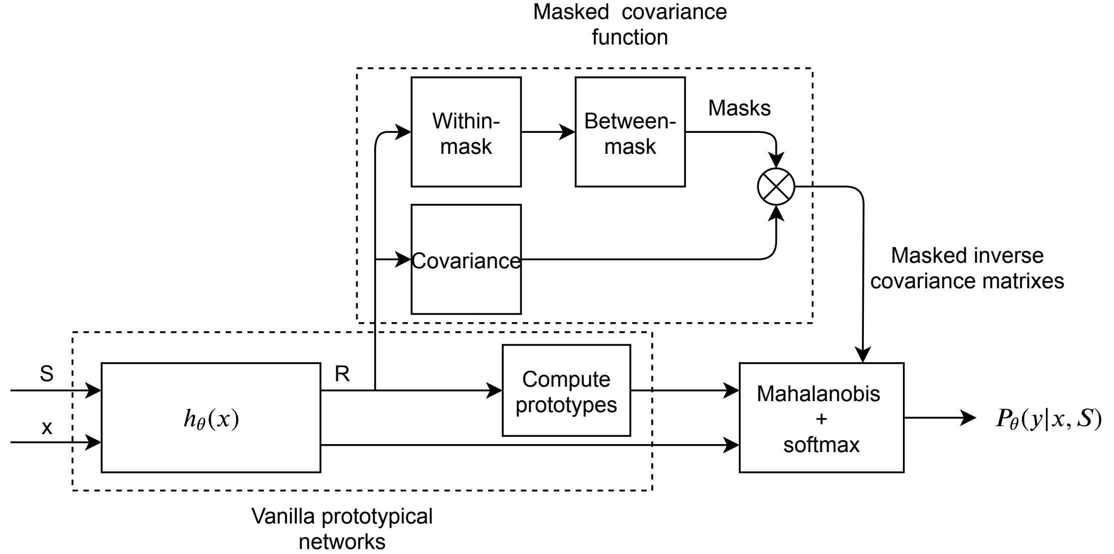


FIGURE C.3: Architecture of masked covariance function with Prototypical networks.

Algorithm 2 Within-mask

Require: $q^1_{\theta_1}(x)$, $k^1_{\theta_2}(x)$, $v^1_{\theta_3}(x)$, and $S : (nK, 3, d_1, d_1) \xrightarrow{g_\theta(x)} R : (nK, d_2)$

1: Compute $R : (nK, d_2) \xrightarrow{q^1_{\theta_1}(x), k^1_{\theta_2}(x), v^1_{\theta_3}(x)} q, k, v : (K, n, d_2)$
2: Compute element-wise multiplication $q \circ k$, and transpose, which gives $s : (K, d_2, n)$

3: Sum across dim=3, which gives $\hat{s} : (K, d_2, 1)$
4: Compute softmax of $\hat{s}$ over dim=2 to get masks per class over the dimensions. This gives $f : (K, d_2, 1)$
5: Compute element-wise multiplication $f \circ v$, and broadcast, which gives $R_2 : (K, d_2, n)$

6: **return** R_2

Experimental setup

We used vanilla prototypical networks with the standard backbone and hyperparameters settings from Appendix A. The difference is that we did not use the Euclidian distance but the Mahalanobis distance and the covariance function. We trained this model on the set of five-shot, five-way and five-shot, 20-way learning tasks, with 100 epochs on

Algorithm 3 Between-mask

Require: $q^2_{\theta_1}(x)$, $k^2_{\theta_2}(x)$, $conv1d_{\theta_4}(x)$, and $R_2 : (K, d_2, n)$

1: Average over n samples per class with $R_2 : (K, d_2, n) \xrightarrow{conv1d_{\theta_4}(x)} R_3 : (K, d_2)$
2: Compute $R_3 : (k, d_2) \xrightarrow{q^2_{\theta_1}(x), k^2_{\theta_2}(x)} q, k : (k, d_2)$
3: Compute element-wise multiplication $q \circ k$, and transpose, which gives $s : (K, d_2)$
4: Sum across dim=1, which gives $\hat{s} : (d_2)$
5: Compute softmax of $\hat{s}$ to get masks between classes. This gives $f : (d_2)$
6: **return** $f : (1, 1, d_2)$

the small logo dataset. We computed the accuracy and 95% confidence intervals with 500 sets of tasks from the meta-test set.

For the functions $q^1_{\theta_1}(x)$, $k^1_{\theta_2}(x)$, and $v^1_{\theta_3}(x)$, we used feed forward neural networks with an input size of n nodes, one hidden layer with 10 nodes, and an ouput layer with n nodes. For the functions $q^2_{\theta_1}(x)$ and $q^2_{\theta_1}(x)$, we used also feed forward neural networks with an input size of K, hidden layer with 10 nodes, and an output layer with K nodes. Furthermore, we used a one-dimensional convolutional layer with filter one, and stride one for $conv1d_{\theta_4}$. For the covariance layer $cov_{\theta_5}(x)$, we used a feed forward neural network with n nodes as input, hidden layer with 10 nodes, and an output layer with one node as output. Finally, for every layer in every network, we used a ReLU layer but not after the output layer.

Results

In Table C.4, we show the results from the Prototypical networks model with covariance function. We compare the results with different prototypical networks specifications.

	5-shot 5-way	5-shot 20-way
Protonet with covariance function	**89.41 ± 0.66%**	77.96 ± 0.41%
Protonet	87.96 ± 0.68%	76.25 ± 0.40%
Gaussian protonet	**89.68 ± 0.67%**	**79.97 ± 0.40%**
Mahalanobis protonet	-	**79.74 ± 0.43%**

Table C.4: Few-shot classification accuracies from the different Prototypical network models. The bold cells indicate the best performing model(s) within 95% confidence interval in that column.

Discussion

In Table C.4, we see the results of Prototypical networks with the covariance function. We see that it has comparable performance on the set of five-shot, five-way learning

tasks, although it performs less on the set of five-shot, 20-way learning tasks compared to Gaussian and Mahalanobis prototypical networks. The number of parameters that we used is comparable to Gaussian and Mahalanobis prototypical networks but we do have poor between-flexibility, because higher K-way learning tasks require to use different sizes of feed forward neural networks. In the future, we plan to try out different configuration, check if the covariance function behaves as expected, and train on different datasets, such as mini-ImageNet. We can then see how it compares to state-of-the-art meta-learning models.

Bibliography

[1] Stanislav Fort. Gaussian prototypical networks for few-shot learning on omniglot. August 2017. URL https://arxiv.org/abs/1708.02735.

[2] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. October 2018. URL https://arxiv.org/abs/1803.02999.

[3] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV (7)*, volume 9911 of *Lecture Notes in Computer Science*, pages 499–515. Springer, 2016. ISBN 978-3-319-46477-0. URL http://dblp.uni-trier.de/db/conf/eccv/eccv2016-7.html#WenZL016.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. 2012. URL https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[5] Goncalo Oliveira, Xavier Frazao Andre Pimentel, and Bernardete Ribeiro. Automatic graphic logo detection via fast region-based convolutional networks. April 2016. URL https://arxiv.org/pdf/1604.06083v1.pdf.

[6] Steven C.H. Hoi, Xiongwei Wu, Hantang Liu, Yue Wu, Huiqiong Wang, Hui Xue, and Qiang Wu. Logo-net: Large-scale deep logo detection and brand recognition with deep region-based convolutional networks. November 2015. URL https://arxiv.org/pdf/1511.02462.pdf.

[7] Forrest N. Landola, Anting Shen, Peter Gao, and Kurt Keutzer. Deeplogo: Hitting logo recognition with the deep neural network hammer. October 2015. URL https://arxiv.org/pdf/1510.02131v1.pdf.

[8] Simone Bianco, Marco Buzzelli, Davide Mazzini, and Raimondo Schettini. Deep learning for logo recognition. May 2017. URL https://arxiv.org/pdf/1701.02620.pdf.

[9] Hang Su, Shaong Gong, and Xiatian Zhu. Weblogo-2m: Scalable logo detection by deep learning from the web. October 2017. URL https://ieeexplore.ieee.org/document/8265251/authors#authors.

[10] Andras Tzk, Christian Herrmann, Daniel Manger, and Jrgen Beyerer. Open set logo detection and retrieval. October 2017. URL https://arxiv.org/pdf/1710.10891.pdf.

[11] Alexis Joly and Olivier Buisson. Logo retrieval with a contrario visual query expansion. October 2009. URL https://dl.acm.org/citation.cfm?id=1631361&dl=ACM&coll=DL.

[12] Stefan Romberg, Lluis Garcia Pueyo, and Rainer Lienhart. Scalable logo recognition in real-world images. April 2011. URL http://www.multimedia-computing.de/mediawiki/images/3/34/ICMR2011_Scalable_Logo_Recognition_in_Real-World_Images.pdf.

[13] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. June 2017. URL https://arxiv.org/abs/1703.05175.

[14] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *CoRR*, abs/1710.11622, 2017. URL http://arxiv.org/abs/1710.11622.

[15] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[16] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[18] Charu C. Aggarwal. *Neural Networks and Deep Learning*. 2018.

[19] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[20] Alessandro Achille and Stefano Soatto. On the emergence of invariance and disentangling in deep representations. *CoRR*, abs/1706.01350, 2017. URL http://arxiv.org/abs/1706.01350.

[21] Jasper Linmans, Jim Winkens, Bastiaan S. Veeling, Taco S. Cohen, and Max Welling. Sample efficient semantic segmentation using rotation equivariant convolutional networks. *CoRR*, abs/1807.00583, 2018. URL http://arxiv.org/abs/1807.00583.

[22] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018. URL http://arxiv.org/abs/1811.03378.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. April 2015. URL https://arxiv.org/pdf/1409.1556.pdf.

[24] Shuyang Sun, Jiangmiao Pang, Jianping Shi, Shuai Yi, and Wanli Ouyang. Fishnet: A versatile backbone for image, region, and pixel level prediction. *CoRR*, abs/1901.03495, 2019. URL http://arxiv.org/abs/1901.03495.

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[26] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015. URL http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[29] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *Journal of Machine Learning Research*, 19(3):1–34, December 2018. URL http://www.jmlr.org/papers/volume19/17-646/17-646.pdf.

[30] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? November 2014. URL https://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

[31] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. August 2018. URL https://arxiv.org/pdf/1808.01974.pdf.

[32] Jiatao Gu, Yong Wang, Yun Chen, Kyunghyun Cho, and Victor O. K. Li. Meta-learning for low-resource neural machine translation. *CoRR*, abs/1808.08437, 2018. URL http://arxiv.org/abs/1808.08437.

[33] Tyler R. Scott, Karl Ridgeway, and Michael C. Mozer. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. *CoRR*, abs/1805.08402, 2018. URL http://arxiv.org/abs/1805.08402.

[34] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28 (4):594–611, April 2006. URL https://authors.library.caltech.edu/5407/1/LIFieeetpam06.pdf.

[35] Brenden M. Lake, Ruslan, Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. One shot learning of simple visual concepts. 2011. URL https://cims.nyu.edu/~brenden/LakeEtAl2011CogSci.pdf.

[36] Sudharsan Ravichandiran. *Hands-On Meta Learning with Python*. 2018.

[37] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. December 2017. URL https://arxiv.org/abs/1606.04080.

[38] Oriol Vinyals. Model vs optimization meta learning, December 2018. URL http://metalearning-symposium.ml/files/vinyals.pdf.

[39] Daniel Jiwoong Im and Graham W. Taylor. Learning a metric for class-conditional KNN. *CoRR*, abs/1607.03050, 2016. URL http://arxiv.org/abs/1607.03050.

[40] Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *NIPS*, pages 513–520, 2004. URL http://dblp.uni-trier.de/db/conf/nips/nips2004.html#GoldbergerRHS04.

[41] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaption of deep networks. July 2017. URL https://arxiv.org/abs/1703.03400.

[42] Hugo Larochelle. Few shot learning with meta learning progress made and challenges ahead, Juli 2018. URL https://twitter.com/hugo_larochelle/status/1018971531140091907.

[43] Hang Su, Xiatian Zhu, and Shaong Gong. Deep learning logo detection with data expansion by synthesising context. March 2018. URL https://arxiv.org/pdf/1612.09322.pdf.

[44] 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. March 2018. URL https://arxiv.org/abs/1711.06025.

[45] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=rJY0-Kcll.

[46] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *CoRR*, abs/1703.00837, 2017. URL http://arxiv.org/abs/1703.00837.

[47] Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. A simple neural attentive meta-learner. February 2018. URL https://arxiv.org/abs/1707.03141.

[48] István Fehérvári and Srikar Appalaraju. Scalable logo recognition using proxies. *CoRR*, abs/1811.08009, 2018. URL http://arxiv.org/abs/1811.08009.

[49] Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. One shot learning of simple visual concepts. January 2011. URL https://cims.nyu.edu/~brenden/LakeEtAl2011CogSci.pdf.

[50] Wikipedia contributors. Bilinear interpolation — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Bilinear_interpolation&oldid=906625267. [Online; accessed 30-July-2019].

[51] Istvan Fehervari and Srikar Appalaraju. Scalable logo recognition using proxies. November 2018. URL https://arxiv.org/abs/1811.08009.

[52] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *J. Mach. Learn. Res.*, 6:1705–1749, December 2005. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1046920.1194902.

[53] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[54] Wikipedia contributors. Multivariate normal distribution — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Multivariate_normal_distribution&oldid=910530169. [Online; accessed 21-August-2019].

[55] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *CoRR*, abs/1703.07737, 2017. URL http://arxiv.org/abs/1703.07737.

[56] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015. URL http://arxiv.org/abs/1503.03832.

[57] Chao-Yuan Wu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. Sampling matters in deep embedding learning. *CoRR*, abs/1706.07567, 2017. URL http://arxiv.org/abs/1706.07567.

[58] Andrew Zhai and Hao-Yu Wu. Making classification competitive for deep metric learning. *CoRR*, abs/1811.12649, 2018. URL http://arxiv.org/abs/1811.12649.

[59] Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. August 2017. URL https://arxiv.org/pdf/1703.07464.pdf.

[60] Martin Baklid and Nils Barlaug. Towards faster development of deep learning models using meta-learning. 2018.

[61] Trieu H. Trinh, Andrew M. Dai, Thang Luong, and Quoc V. Le. Learning longer-term dependencies in rnns with auxiliary losses. *CoRR*, abs/1803.00144, 2018. URL http://arxiv.org/abs/1803.00144.

[62] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction. *CoRR*, abs/1808.03867, 2018. URL http://arxiv.org/abs/1808.03867.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[64] Fetsje Bijma, Marianne Jonker, and Aad van der Vaart. *Inleiding in de Statistiek*. 2016.

[65] Hong-Ming Yang, Xu-Yao Zhang, Fei Yin, and Cheng-Lin Liu. Robust classification with convolutional prototype learning. *CoRR*, abs/1805.03438, 2018. URL http://arxiv.org/abs/1805.03438.

[66] Nicolas Lachiche and Peter Flach. Improving accuracy and cost of two-class and multi-class probabilistic classifiers using roc curves. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 416–423. AAAI Press, 2003. ISBN 1-57735-189-4. URL http://dl.acm.org/citation.cfm?id=3041838.3041891.

[67] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *CoRR*, abs/1904.04232, 2019. URL http://arxiv.org/abs/1904.04232.

[68] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[69] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[70] Yan Zheng, Ronggui Wang, Juang Yang, Lixia Xue, and Min Hu. Principal characteristics networks for few-shot learning. *Journal of Visual Communication and Image Representation*, 59(1):563–573, February 2019. URL https://www.sciencedirect.com/science/article/pii/S1047320319300574.

[71] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 2017. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2017.05.007. URL http://www.sciencedirect.com/science/article/pii/S1077314217300814.

[72] Anna M. Kruspe. One-way prototypical networks. *CoRR*, abs/1906.00820, 2019. URL http://arxiv.org/abs/1906.00820.

[73] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015.

[74] Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. *CoRR*, abs/1805.10123, 2018. URL http://arxiv.org/abs/1805.10123.

[75] Hongyang Li, David Eigen, Samuel Dodge, Matthew Zeiler, and Xiaogang Wang. Finding task-relevant features for few-shot learning by category traversal. *CoRR*, abs/1905.11116, 2019. URL http://arxiv.org/abs/1905.11116.