Meta Learning for Supervised and Unsupervised Few-Shot Learning

Antreas Antoniou

Doctor of Philosophy

Institute for Adaptive and Neural Computation

School of Informatics

University of Edinburgh

2021

Abstract

Meta-learning or learning-to-learn involves automatically learning training-algorithms such that models trained with such learnt algorithms can solve a number of tasks while demonstrating high performance in a number of predefined objectives. Meta-learning has been used successfully in a large variety of application areas. In this thesis, we present three meta-learning-based methods that can very effectively tackle the problem of *few-shot learning*, where a model needs to be learned with only a very small amount of available training samples for each concept to be learned (e.g. 1-5 samples for each concept). Two of the proposed methods are trained using supervised learning, and another involves using self-supervised learning to achieve unsupervised meta-learning.

The proposed methods build on the *Model Agnostic Meta-Learning* (MAML) framework. MAML is a gradient-based meta-learning method, where one learns a parameter initialization for a model, such that after the model has been updated a number of times towards a small training set (i.e. a support set), it can perform very well on previously unseen instances of the classes it was trained on, usually referred to as a small validation (i.e. a target set). The initialization is learned by utilizing two levels of learning. One inner-most level where the initialization is updated towards the support set and evaluated on the target set, thus generating an objective directly quantifying the generalization performance of the inner-loop model (i.e. the target-set loss), and an outer-most level where the parameter-initialization is learned using the gradients with respect to the target-set loss.

Our first method, referred to as MAML++, improves the highly unstable MAML method using a number of modifications in the batch normalization layers, the outer loop loss formulation and the formulation of the learning scheduler used on the inner loop. Not only does MAML++ enable MAML to converge more reliably and efficiently, but it also improves the model's generalization performance. We evaluate our method on Omniglot and Mini-ImageNet, where our model showcases vastly improved convergence speed, stability and generalization performance.

The second proposed method, referred to as Self-Critique and Adapt (SCA), builds on MAML++ by allowing the inner loop model to adapt itself on the unsupervised target set, by learning an unsupervised loss function parameterized as a neural network. This unsupervised loss function is learned jointly with the parameter initialization and learning scheduler of the model as done in MAML++. SCA produces SOTA results in few-shot learning, further improving the performance of MAML++. Our model is evaluated on Omniglot and Mini-ImageNet where it sets SOTA-level performance.

The third proposed method, referred to as Assume, Augment and Learn (AAL) involves sampling pseudo-supervised tasks from an unsupervised training set by leveraging random labels and data augmentation. These tasks can then be used to directly train any few-shot learning model to perform well on a given dataset. We apply our method on MAML++ and ProtoNets on two datasets, Omniglot and Mini-ImageNet where our model produces state-of-the-art (SOTA) results in Omniglot and competitive performance with SOTA methods in Mini-ImageNet.

Table of Contents

Acknowledgements

Carrying out the work in this thesis and writing the actual document were the result of admittedly a lot of luck, as everything else in the universe, which starts from my initial birth as a human and branching out all the way to sending the right emails to the right people to get admitted in this PhD program. Since it is impossible to hunt down the exact key people and moments that lead to this, I shall provide my own, admittedly, imperfect, approximation, thoroughly biased by the nature of my humanity and emotional connections. Firstly, I would like to thank my mother and family, for providing me with ample support of all types throughout the years. Furthermore, I would like to thank my grandfather, who himself being a builder, an architect, and a man of much creativity and physics intuition, inspired me at an early age to pursue my design and implementation of various crafts at a young age, as well as my mum for providing inspiration in pursuing math, cooking and music. In addition I would like to thank my home country television channels for providing ample science-based cartoons that inspired my development into a scientist, by directly showcasing that science is a super power. Furthermore, I would like to thank my grandma, Maro, who always believed in me and loved me, and helped my development into an engineer by buying tons of Lego kits for me as a kid, through sacrificing her own savings, in a purely altruistic attempt to help her grandson build his imagined robots. I would also like to thank my father for teaching me to not be shy and to take two steps forward whenever fear pushes me one step back. Furthermore, I'd like to thank my beloved partner and guardian angel, Athina Panayiotou, who, during the COVID19 isolation period, and my developing very severe depression and a multitude of other mental disorders, reached out to me, without any personal gain, and freely offered to me love and support when I had become dysfunctional, thus saving my mind and, I believe, my life. On this topic, I'd also like to thank my friend and colleague Pavlos Andreadis, who, during my PhD, offered friendship and tons of philosophical discussions, as well as emotional and mental support when my mental health took a turn for the worse, he, too, helped protect my mind from utter despair. Further, I want to thank my supervisor, Amos Storkey, who gave me the opportunity to join this PhD program, and was a true mentor and friend throughout my PhD, and supported me both emotionally, when health problems arose and academically in my continued attempts to produce good research. I would also like to thank Steve Renals, for hiring me to be his TA for the Machine Learning Practical course, which had a massive impact in my development as a teacher and a researcher, as well as his direct input in my development as an academic and what I consider to be a better human being. I should also thank Luke Darlow and his wife, Piette, who sheldered me and my partner when our flat had flooded as a result of bad waste-disposal systems, and offered us some solace in a very unstable time. Luke additionally was my gaming buddy in a very dark time during COVID19 and my subsequent development of deep depression, helping me slowly recover. I also want to thank Harrison Edwards, who supported my development as a researcher through his direct mentorship and technical support, as well as increasing fun and meaning in my life through his friendship. Furthermore, I'd like to thank Dr. Rebecca Sperling for offering ample support and guidance in adventuring within my own mind and regaining control over my mental health disorders, enabling me to increase daily serenity and functionality. Without a doubt her support helped saved my mind in a truly dark time. I also want to thank my research group members, Elliot Crowley, Paul Micaelli, Gavin Gray, Eleanour Platt, Joe Mellor, Taro Makino, Jan Brauner, Massimiliano Patacchiola and James Owers, for all the fun discussions, idea pitching and paper reviewing as well as their continual friendship during and after my PhD. I should also thank Google and StackOverflow for providing ample information on a large number of technical topics, as well as Github, for hosting and tracking my code. This thesis was supported by in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh as well as a Huawei DDMPLab Innovation Research Grant.

Chapter 1

Introduction

The human capacity to learn new concepts using only a handful of samples is immense. In stark contrast, modern deep neural networks need, at a minimum, thousands of samples before they begin to learn representations that can generalize well to unseen data-points [Krizhevsky et al., 2012a, Huang et al., 2017], and mostly fail when the data available is scarce. The fact that standard deep neural networks fail in the small data regime can provide hints about some of their potential shortcomings. Solving those shortcomings has the potential to open the door to understanding intelligence and advancing Artificial Intelligence.

Few-shot learning is a machine learning sub-field investigating methods that can learn new concepts with only a handful of data-points (usually 1-5 samples per concept). This possibility is attractive for a number of reasons. First, effective few-shot learning would reduce the need for data collection and labelling, thus reducing the time and resources needed to build robust machine learning models. Second, it would potentially reduce training and fine-tuning times for adapting systems to newly acquired data. Third, in many real-world problems there are only a few samples available per class and the collection of additional data is either remarkably time-consuming and costly or altogether impossible, thus necessitating the need to learn effectively from the available few samples.

The nature of few-shot learning makes it a very hard problem if no prior knowledge exists about the task at hand. For a model to be able to learn a robust model from a few samples, knowledge transfer [Caruana, 1995] from other similar tasks is key. However, manually crafting such knowledge priors to be used in effective few-shot learning is an arduous and often infeasible task. *Meta-learning* [Schmidhuber, 1987, Vilalta and Drissi, 2002, Hospedales et al., 2020], or *learning to learn* [Thrun and Pratt, 1998], can instead be used to automatically learn *across-task* knowledge and distill it within a learning prior such that our model can use that learning prior at inference time, to quickly acquire *task-specific* knowledge from new tasks using only a few samples.

Meta-learning can be broadly defined as the task of learning a learning process that becomes more proficient at learning with more experience, thus learning *how* to learn. Meta-learning involves abstracting learning in a number of learning levels. To describe how such abstractions work we'll use two such learning levels for simplicity, as well as because most models at the time of writing this thesis use two learning levels to achieve meta-learning. The *outer-loop* or meta level, where a *learner* or *meta-model* is learned by acquiring *across-task* knowledge. And, the *inner-loop* or *task level*, where the meta-model acquires task-specific knowledge from a task and distills it into a *base-model* that is then required perform a target task well. The performance of the base-model on a target task is then quantified using an objective function. That objective function is then used to update the learner (and often the base-model as well), such that it learns to learn more efficiently and effectively. With that being said, meta-learning at more than two levels is possible, as if one tried to meta-learn to meta-learn, they would end up with meta-meta-learning. Theoretically additional abstraction levels can be added in this same manner.

There is no strict format in what module the meta-model represents, but some explored patterns include: **1).** Parameter initialization learning [Finn et al., 2017b, Antoniou et al., 2018a, Rusu et al., 2019, Li et al., 2017b] where the meta-model is a parameter initializer that is used to initialize a base-model. **2).** Optimizer learning [Schmidhuber, 1993, Andrychowicz et al., 2016, Park and Oliva, 2019, Li and Malik, 2017, Bello et al., 2017a, Munkhdalai and Yu, 2017b, Ravi and Larochelle, 2016], where the meta-model is a learned optimizer that is then used in the inner-loop to update a base-model. **3).** Loss function learning [Houthooft et al., 2018a, Sung et al., 2017a, Zhou et al., 2020, Li et al., 2019], where the meta-model is a loss function that is used to update the base-model in the inner-loop. **4).** Architecture learning [Stanley et al., 2019, Liu et al., 2019a, Zoph and Le, 2016, Real et al., 2018, Elsken et al., 2019a], where the meta-model is a model architecture that is then applied to the base–model as a means of predisposing the model to certain training dynamics and as a result specific performance qualities. **5).** Data Augmentation learning [Cubuk et al., 2019, Li et al., 2020, Volpi and Murino, 2019, Antoniou et al., 2017], where the meta-model is a data-augmentation strategy to be applied to the data-points on the inner-loop, which are then used to update the base-model. **6).** Task learning [Doersch and Zisserman, 2017, Pathak et al., 2016], where the meta-model is an auxiliary task which when learned in the inner-loop along with the original task, can improve the original task's performance. Very relevant in unsupervised, semi-supervised, transductive, and self–supervised learning tasks. **7).** Other component learning [Bengio et al., 1990, Miconi et al., 2019, Metz et al., 2019, Prajit Ramachandran, 2017], these include learning to backpropagate more effectively, activation functions, learning rate schedulers etc. **8).** Mixed variants from this list [Munkhdalai and Yu, 2017b, Antoniou and Storkey, 2019b, Rusu et al., 2019], where two or more of the previously mentioned learning components form the meta-model.

Meta-learning has been studied for a long time, however, its potential as a driver of progress in deep-learning has generated a recent explosion of work in the area, generating state-of-the-art methods in a variety of settings spanning from few-shot learning [Finn et al., 2017a, Snell et al., 2017b] to unsupervised learning [Metz et al., 2019, Antoniou and Storkey, 2019b], data efficient learning [Duan et al., 2016, Houthooft et al., 2018a], self-directed [ale, 2020] reinforcement learning (RL), hyperparameter optimization [Franceschi et al., 2018], and neural architecture search (NAS) [Liu et al., 2019a, Real et al., 2018, Zoph and Le, 2016].

One of the most studied and useful meta-learning methods, known for its simplicity and state of the art performance, is *Model Agnostic Meta-Learning* (MAML) [Finn et al., 2017b]. In MAML, the authors propose learning a parameter initialization for a base-model such that after a few iterations of standard gradient methods on a small training set, it can achieve strong generalization performance on a validation set. Relating back to the definitions of meta-model and base-model, in MAML, the meta-model is effectively the parameter initialization. This parameter initialization is used to initialize the base-model, which is then used for few-shot learning on a support set, and then a generalization test on a target set. MAML is a simple yet elegant meta-learning framework that has achieved state of the art results in a number of settings [Finn et al., 2017a, 2019, 2018].

However, MAML suffers from a variety of problems such as: **1).** Gradient degradation issues leading to training instability. **2).** Static and manually chosen inner-loop learning rates and ineffectively formulated batch normalization restricting the model's generalization performance, reducing the framework's flexibility and increasing the amount of hyperparameter tuning required, **3).** Continuous need of second order gradient computations in each meta-training iteration leading to increased computational budgets. In Chapter 3 we propose a set of methods which can directly be added on MAML which enable robust training stability, faster convergence speed and higher generalization performance, referred to as *MAML++*.

Most few-shot meta-learning methods investigate methods that can improve learning from the labelled support-set during the inner-loop optimization process, whereas learning from the unlabelled target-set in the inner-loop optimization process has remained unexplored. In Chapter 4, we propose a mechanism we call *Self-Critique and Adapt*, or SCA that meta-learns an unsupervised loss function that can extract task-specific knowledge from the target-set to be distilled in the base-model. This unsupervised loss function is learned to work in conjuction with the supervised learning phase that takes place prior to the unsupervised phase (facilitated by this learned unsupervised loss function). The loss function is conditioned on base-model state and task information, such as base-model predictions on the target set, parameter-state and task-data. Once computed, this loss function can be used to update the base-model with target-set information to improve generalization performance. The method is very flexible and can be used on top of any existing meta-learning few-shot learning method such as MAML [Finn et al., 2017b], as well as ones that do not use gradient-based updates on the support set, such as Matching Networks [Vinyals et al., 2016]. We evaluate the proposed method on the established few-shot learning benchmarks of Mini-ImageNet [Ravi and Larochelle, 2016] and Caltech-UCSD Birds 200 (CUB) [Chen et al., 2019]. The evaluation results indicate that our method substantially boosts the performance of two separate instances of the MAML++ [Antoniou et al., 2018a] framework, setting a new state-of-the-art performance for all tasks in both benchmarks.

Most meta-learning methods are built around supervised meta-learning, that is, meta-learning where the outer-loop training requires label information about the tasks to be learned. However, the very desirable unsupervised meta-learning setting has seen very little investigation. Such unsupervised meta-learning has been demonstrated to be achievable by various forms of self-supervised learning [Hsu et al., 2018, Khodadadeh et al., 2019, Antoniou and Storkey, 2019a], where the outer-loop tasks are carefully crafted to be producible without labels, usually using synthetic labels, such that the learned learner can generalize to real-labelled datasets at test time. In Chapter 5, we propose such a self-supervised meta-learning method, which leverages random labels and data augmentation to generate self-supervised tasks, with good generalization on the test set, achieving SOTA results in Omngilot and competitive results on Mini-ImageNet.

1.1 Prerequisites: Model Agnostic Meta-Learning

Model Agnostic Meta-Learning (MAML) [Finn et al., 2017b] is a meta-learning framework for few-shot learning. MAML is elegantly simple yet can produce state of the art results in few-shot regression/classification and reinforcement learning problems. In a sentence, MAML learns good initialization parameters for a network, such that after a few steps of standard training on a few-shot dataset, the network will perform well on that few shot task.

More formally, we define the *base model* to be a neural network f_θ with meta-parameters θ. We want to learn an initial $\theta = \theta_0$ such that, after a small number N of gradient update steps on data from a support set S_b to obtain θ_N, the network performs well on that task's target set T_b. Here b is the index of a particular support set task in a batch of support set tasks. This set of N updates steps is called the *inner-loop optimization process*.

The updated base-network parameters after i steps on data from the support task S_b can be expressed recursively as:

$$\theta_i^b = \theta_{i-1}^b - \alpha \nabla_\theta \mathcal{L}_{S_b}(f_{\theta_{i-1}^b}), \tag{1.1}$$

where α is the learning rate, θ_i^b are the base-network weights after i steps towards task b, $\mathcal{L}_{S_b}(f_{\theta_{(i-1)}})$ is the loss on the support set of task b after $(i-1)$ (i.e. the previous step) update steps. Assuming that our *task batch size* (i.e. the number of tasks our learner is evaluated on prior to a meta-training iteration) is B we can define a *meta-objective*, which can be expressed as:

$$\mathcal{L}_{meta}(\theta_0) = \sum_{b=0}^{B} \mathcal{L}_{T_b}(f_{\theta_N^b(\theta_0)}) \tag{1.2}$$

where we have explicitly denoted the dependence of θ_N^b on θ_0, given by unrolling (1.1). The objective (1.2) measures the quality of an initialization θ_0 in terms of the total loss of using that initialization across all tasks. This meta objective is now minimized to optimize the initial parameter value θ_0. It is this initial θ_0 that contains the across-task knowledge. The optimization of this meta-objective is called the *outer-loop optimization process*.

The resulting update for the meta-parameters θ_0 can be expressed as:

$$\theta_0 = \theta_0 - \beta \nabla_\theta \sum_{b=0}^{B} \mathcal{L}_{T_b}(f_{\theta_N^b(\theta_0)}), \tag{1.3}$$

where β is a learning rate and $\mathcal{L}_{T_b}$ denotes the loss on the target set for task b. In this thesis we use the cross-entropy [De Boer et al., 2005, Rubinstein, 1999] loss throughout.

Chapter 2

Background

2.1 Deep Learning Fundamentals

In this section we introduce a number of core concepts, algorithms and building blocks that are used to build modern deep learning systems. The material in this section is of prime importance in implementing and understanding the work within this thesis.

2.1.1 Fully Connected Neural Networks

Fully connected neural networks were one of the first types of neural network [Schmidhuber, 2015] to be successfully applied on automatic feature learning problems. Each such network is usually composed of a cascade of a number of processing layers, referred to as *fully connected layers*, each layer composed of a linear projection, followed by a non-linear activation function such as ReLU [Xu et al., 2015] or Sigmoid [Karlik and Olgac]. They are considered feedforward as the information within them is never injected in a previous layer, as done in Recurrent Neural Networks (presented in Section 2.1.6).

They first appeared in the literature in the 1980s [Werbos, 1982, Rumelhart et al., 1985]. A notable first example of using fully connected networks to learn unsupervised representations using large-scale natural data was described in Quoc V. Le et al. [Le, 2013] where the authors successfully trained a fully connected autoencoder to reconstruct images from YouTube videos, resulting in very interesting feature detectors, including an explicit cat detector.

Today, these types of networks form the cornerstone of deep neural networks, as they form parts of most temporal processing systems such as Recurrent Neural Networks, Long-Short Term Memory Networks, Convolutional Neural Networks and

Transformers.

Working Principle: We shall focus on the working principle of a single fully connected layer, since a fully connected network is simply a cascade of such layers.

Formally, given a batch of input vectors x with dimensionality $B \times M$ where B is the batch size and M is the number of features within each vector, a weight matrix W with dimensionality $M \times K$ where M is the number of input features and K the number of output features, a vector of biases b with dimensionality K and a non-linear activation function $a(.)$ (such as ReLU or Sigmoid), a fully connected layer f computes the following operation:

$$f(x) = a(x \cdot W + b) \tag{2.1}$$

The output of such a layer $f(x)$, will have dimensionality $B \times K$.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [LeCun, 1998, Krizhevsky et al., 2012b] are considered to be one of the most important inventions in the field of deep-learning of the past decade. They are especially effective on image-related tasks, such as image classification [Krizhevsky et al., 2012b, He et al., 2016, Huang et al., 2017], localization [Sermanet et al., 2013, Johnson et al., 2016], segmentation [Badrinarayanan et al., 2017], generation [Van den Oord et al., 2016, Gregor et al., 2015, Mao et al., 2017, Zhang et al., 2018, Berthelot et al., 2017, Kim et al., 2017, Goodfellow et al., 2014, Hoffman et al., 2017, Goodfellow et al., 2014, Zhu et al., 2017, Pu et al., 2016], captioning [You et al., 2016, Anderson et al., 2018, Yao et al., 2017, Johnson et al., 2016, Rennie et al., 2017, Hossain et al., 2019], translation [Zhu et al., 2017, Hoffman et al., 2017] as well as video-related tasks such as video classification [Karpathy et al., 2014, Yue-Hei Ng et al., 2015, Wu et al., 2015b] and generation [Tulyakov et al., 2018, Denton and Fergus, 2018, Cai et al., 2018, Gygli et al., 2016, Vondrick and Torralba, 2017, Kim et al., 2018b]. In fact, convolutional neural networks have since been modified to be applicable to almost every subtopic in deep-learning, including temporal tasks such as speech recognition [Amodei et al., 2016, Deng et al., 2013, Hannun et al., 2014, Noda et al., 2015, Deng and Platt, 2014, Dong et al., 2018, Zhou et al., 2018] and generation [Ling et al., 2015, Hsu et al., 2017, Sotelo et al., 2017, Mohamed et al., 2019], music recognition [van der Wel and Ullrich, 2017, Pacha and Eidenberger, 2017, Tuggener et al., 2018, Calvo-Zaragoza and Rizo, 2018] and generation

[Briot et al., 2017], text classification [Kowsari et al., 2019] and generation [Iqbal and Qureshi, 2020].

CNNs are currently driving much of the current deep-learning systems found in both production and research settings. A CNN is usually composed of a cascade of computational blocks. Each of such blocks consists of a convolutional layer, followed by a normalization layer, such as Batch Normalization, followed by an activation function.

Working Principle: The superior performance of convolutional layers compared to fully connected layers, is usually attributed to a convolutional layer's translation equivariance properties and learning efficiencies due to large scale parameter sharing. Briefly, the convolution operator dot-products a number of learnable filters across a number of different locations in an image, while sharing the weights of those filters across the entirety of a given image. Since the dot product itself is a directional distance metric, location patches that align directionally with a given filter produce a positively signed output, while those that misalign will produce a negatively signed output. As a result, the convolutional layer detects a number of features within various spatial locations in the image. These features are then recombined, by patching each of such features to the location in the image where it was found. This patched feature representation is referred to as a *feature map*. Since these features are detected across a whole image by a shared filter, across multiple locations within the image, the operation itself is location-agnostic in that detecting a feature is not conditional on the location of that feature in an image, but rather in the pattern that the feature is searching for. As a result the convolutional layer achieves translation equivariance, in that, translating a given image will not change the feature detections within the image, while those features will be found in different locations than before.

More formally, a convolutional layer can be broken down into three main components:

1. An input patch selector mechanism $s(x)$, where x is the input image, and $s(.)$ the functional form of the selection mechanism.

2. A learnable weight matrix kernel $W_{k_{out}}^{k_{in}}$ with shape $k_h \times k_w \times k_{in} \times k_{out}$ where k_h and k_h are the pre-selected kernel height and width, k_{in} are the number of channels of the incoming volume, and k_{out} are the number of channels the convolutional layer should output.

3. The processing function of the module, the dot-product (denoted as $\cdot$) . The

dot-product compares a batch of input patches with the learnable kernels and produces a single value for each comparison describing a directional distance measure.

These three components then interact as follows: Given an input batch x_i, (containing a number of images) with shape b, k_{in}, h, w where b is the batch size, k_{in} is the number of channels of the images and h, w are the image's height and width, we select a number of sub-regions from each individual image in the batch using the selection mechanism $s(.)$ outlined in the convolution operation, usually implemented using `im2col` or image to column. More specifically, given a convolutional layer's kernel size $k_h \times k_w$, stride s, dilation d and padding size p we begin from the top left corner of the image and move along the row towards the right corner of the image, while taking steps of s pixels at a time while extracting all the patches of size $k_h \times k_w$. Dilation factors [Yu and Koltun, 2016] modify these selected patches by expanding the area (i.e. receptive field) of these patches by ignoring a number of pixels in-between the detection areas of the patch, as explained in Section 2.1.3. Once the right hand edge has been reached, we move our slider to the leftmost edge, and then move vertically downwards s pixels and repeat the process until we reach the bottom right corner. Since each image will produce a batch of such patches, and we are usually processing a whole batch of images at once, our image batch x_i has now been converted into a batch of batches of patch-vectors x_i^j where i is the data-point index, and j is the patch index. At this point, we apply the dot-product operation on our patches and our kernels using $f(x_i)_z = \sum_{j=0}^{J} x_i^j \times W_z$, to obtain a feature map over the whole image x_i, indicating the directional distance of each patch from the feature W_z, where z is the index of the feature we are searching for. This is usually efficiently done by a batch matrix multiply operation, supported by most modern deep learning libraries such as Pytorch [Paszke et al., 2017] and Tensorflow [Abadi et al., 2016]. At this point the outputs of the dot product are returned to the location of their corresponding original patches, reconstructing the original image space, now replaced by convolutional feature activations. Usually a single convolutional layer produces k_{out} feature maps, one for each feature in its kernel bank, as well as two dimensions of spatial features, whose dimensionality varies depending on s, d, p, h, w, k_h and k_w.

It is worth noting that a CNN can become, approximately, translation invariant if a global pooling operation is applied after the last convolutional layer in the model. This is because pooling feature maps on the spatial dimensions will produce a representation that remains the same even if an image has been translated.

Historical Context: Historically, some of the foundations for CNNs were set by Hubel's and Wiesel's work [Hubel and Wiesel, 1962] where they experimented on a cat [1] by connecting electrodes in the animal's visual cortex. The resulting discovery was that activations in the visual cortex work in a hierarchical fashion. Earlier neurons firing for simpler features such as edges, and later neurons firing for increasingly more complex features. This idea, of hierarchical features lead to Fukushima's *Neo-Cognitron* [Fukushima and Miyake, 1982] that was an important precursor to the development of the convolutional neural network [LeCun et al., 1989, 1998] in a form almost identical to modern CNNs. However due to computational constraints and gradient-degradation-related optimization problems, the original system's training time and stability were highly problematic as it required approximately two weeks on a simple character recognition task called the *MNIST* (Modified National Institute of Standards and Technology) image classification benchmark [LeCun, 1998], while the optimization itself was unstable when more depth was introduced. Furthermore, for a model to learn more general features, a large, high quality dataset with class and sample diversity was also necessary. As research in the area continued, early solutions were proposed for the gradient degradation problem [Glorot and Bengio, 2010, Nair and Hinton, 2010], while larger and richer datasets became possible because of the internet facilitating data labelling crowd-sourcing services. At this point, the main bottleneck for training such deep neural networks was the computational one, making deep neural networks trainable only by massive machine clusters [Le, 2013], made of thousands of compute nodes. However, in 2012 Krizhevsky et al. [2012a] trained a deep convolutional neural network using only two modern gaming GPUs, on the large ImageNet dataset [Krizhevsky et al., 2012a] to achieve unprecedented results in image classification. As a result, deep-learning quickly became a research area of prime importance in academia and industry alike. Leading to increasingly better results in a variety of fields such as machine vision [Sermanet et al., 2013, Simonyan and Zisserman, 2014, Ioffe and Szegedy, 2015, He et al., 2016, Szegedy et al., 2017, Huang et al., 2017], speech recognition [Mikolov et al., 2010, Graves et al., 2013, Sak et al., 2014, Zhang et al., 2017] and synthesis [Wu et al., 2015a, Van Den Oord et al., 2016], music generation [Van Den Oord et al., 2016], natural language processing [Young et al., 2017] and reinforcement learning [Lillicrap et al., 2015, Mnih et al., 2015].

[1] The authors of this report do not endorse animal cruelty.

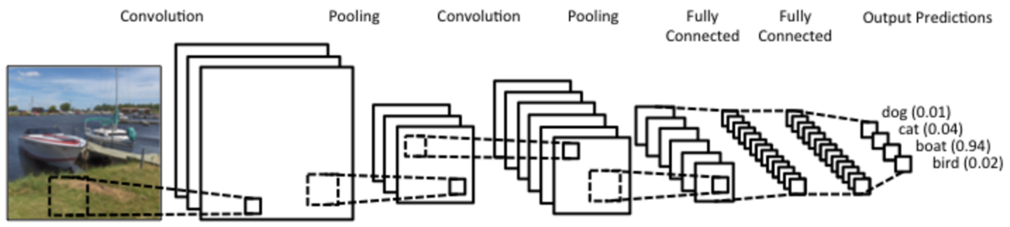


Figure 2.1: Standard CNN: A series of convolutional layers, followed by an activation function such as ReLU, followed by some dimensionality reducing function such as average pooling or max pooling. Once convolutional features are produced, a series of fully connected layers are used to map the features into classes. From [LeCun et al., 1989, 1998]

2.1.3 Dilated Convolutions

Dilated convolutions are effectively a way of expanding the receptive field of a convolutional layer without increasing the number of learnable parameters needed, as would be the case for a convolutional layer with a larger kernel size. For example, in a standard convolution with 3×3 kernels and a stride of 1, the filters scan the image in 3×3 regions of adjacent pixels. This filter has *dilation* 1: centre-to-centre, there is a 1-pixel distance between each filtered pixel and its nearest neighbour. Now, consider the case where there is a 2-pixel distance: the filter is only applied to pixels that are in both odd-numbered rows and columns in each 5×5 region. This is dilation 2. For dilation 3, the filter is applied to only pixels in every third row and column in each 7×7 region, and so on. These are illustrated in Figure 2.2. These dilations allow a model to learn higher order abstractions without the need for dimensionality reduction. They are frequently used in segmentation networks [Yu and Koltun, 2016, Romera et al., 2017a,b], but can also be used for model compression [Crowley et al., 2018], and audio generation [van den Oord et al., 2016] among other things.

2.1.4 Normalization Layers

Normalizing the inter-layer features of a neural network has been demonstrated to allow faster convergence (in terms of training iterations), more stable training and higher generalization performance in neural networks [Ioffe and Szegedy, 2015, Ba et al., 2016]. Normalizing features has a noticeable improvement on the gradient degradation problem discussed in Section 2.1.5. Some of the most notable normalization schemes

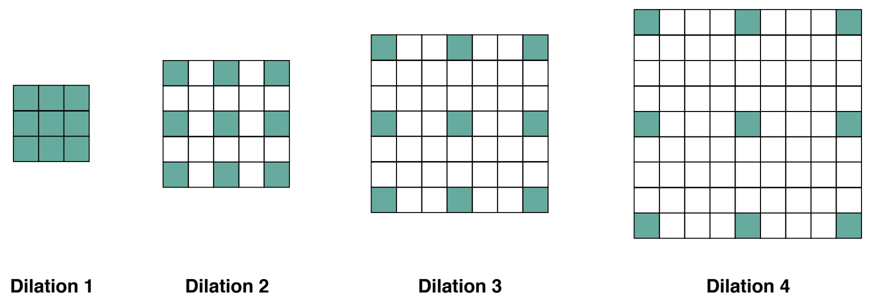


Figure 2.2: An illustration showing the effect of increasing the dilation of a 3×3 filter. For a given image patch, each coloured square corresponds to the locations at which the filter is placed. Notice that this has the effect of increasing the receptive field of the filter as dilation is incremented.

are those of *Batch Normalization* [Ioffe and Szegedy, 2015], *Layer Normalization* [Ba et al., 2016] and *Instance Normalization* [Ulyanov et al., 2016].

Batch normalization is a technique that normalizes inter-layer features by using a given batch's mean and standard deviation as the normalization statistics. More specifically, given a data-point with shape (b, c, h, w), where b is the batch size, c is the number of channels, and h, w are the height and width respectively, the statistics will be computed across the batch and spatial dimensions, therefore producing c means and standard deviations, one for each channel. Batch normalization also employs a learnable scaling parameter referred to as γ and a learnable bias parameter referred to as β. Using these two terms batch normalization can learn the appropriate scaling and shifting for a given task and model. The reasons behind batch normalization's effectiveness is a topic of ongoing debate [Santurkar et al., 2018]. The original authors argued that batch normalization achieved this effect because of reduction of internal covariate shift. However, recent work in Santurkar et al. [2018] has demonstrated that internal covariate shift does not, in fact, decrease. Instead, the authors of [Santurkar et al., 2018] propose that the reason that batch normalization helps speed up and improve neural network training is because it substantially smoothens the optimization landscape (therefore smoothing the gradients computed in backward pass). Which, as a result, produces a more stable step-wise gradient surface. And this, they argue is what produces the improved speed, stability and generalization performance.

Layer Normalization normalizes inter-layer features of a neural network using the sample-wise means and standard deviations. More specifically, when the data points are vectors, then means and standard deviations are computed for each such vector, while when the data is of matrix form, they are first flattened and then the means and standard deviations computed. So, in our earlier example, this would produce b means and standard deviations, one for each sample in a batch. Layer normalization also learns scaling and shifting parameters in the same way batch normalization does. Layer normalization is a great fit for problems where the batch size is close or equal to one, since batch statistics from small batches or even single data-points contain a large amount of noise and are usually very far from the true mean and standard deviation of a dataset.

Instance Normalization was first introduced in an attempt to improve style transfer with neural networks. It uses statistics computed across spatial dimensions only. As a result each image has a mean and standard deviation for each of its channels, that is the method computes b,c means and standard deviations, this is in contrast to batch normalization's c means and standard deviations. In more detail, the difference between batch normalization and instance normalization is that in instance normalization each image within a batch of images has distinct channel-wise statistics whereas in batch normalization all images within a batch use the same channel-wise statistics. If one is trying to generate images, having distinct statistics for each image allows more expressiveness for the generator network since it is no longer restricted to batch statistics.

2.1.5 Gradient Degradation: Causes and Solutions

2.1.5.1 Causes

Deep neural networks have always suffered from the *gradient degradation* problem. That is, the fact that as a model becomes deeper, the probability that the gradients propagated to the earlier - shallower layers tends to either diminish to zero, or explode exponentially to infinity. This happens for the simple reason that the weights in any given layer will have values within the space representable by floating point numbers. If such a value is smaller or larger than one in a number of cascading layers, then the gradients backpropagated in a network utilizing only fully connected or convolutional layers followed by non-linearities will decrease according to the function x^N where $x < 1.0$ or $x > 1.0$ and N is the number of layers in the neural network, in the first instance causing diminishing gradients and in the second exploding gradients.

2.1.5.2 Pretraining

Early attempts to reduce the levels of gradient degradation included supervised [Bengio et al., 2007] and unsupervised [Erhan et al., 2009] layer-wise pretraining. Briefly, for a deep network of N layers one would pretrain the first layer by training a single layer neural network on a given task, then freezing the weights of the pretrained layer, and stacking the next layer, and training as a two layer neural network. This process would continue until the Nth layer was reached. This method could be applied on both supervised tasks, by always appending the prediction layer after the layer currently being trained, as well as unsupervised tasks, such as image reconstruction via autoencoders. The reason for the effectiveness for these two methods, was that the initialization of the network when it begins its training on the full model, would be in a much better distribution than the completely random ones that were employed in the early days of neural networks.

However, we have since learned that computational blocks within a neural network can themselves ensure that the gradients remain within a suitable range to allow efficient training of decently deep neural networks up to 1000 layers whereas originally anything larger than 25 was problematic.

2.1.5.3 Feature Normalization

Instead of attempting to improve downstream gradients by pretraining a good initialization, one can also ensure that the gradients of a randomly initialized neural network during training remain within a sensible distribution.

The first proposed method that could alleviate some gradient degradation problems was Batch Normalization [Ioffe and Szegedy, 2015], explained in Section 2.1.4. Batch normalization would ensure that the inter-layer feature distributions within a deep neural network remain normalized, with only allowing the shift and scale terms to be learned and thus changed during training, which stabilizes training. Furthermore, batch normalization was demonstrated to smoothen the optimization landscape for a given task as a direct consequence of the mathematical operations describing batch norm's backward pass. These two effects allowed deeper networks to be successful trained without any requirement for pretraining.

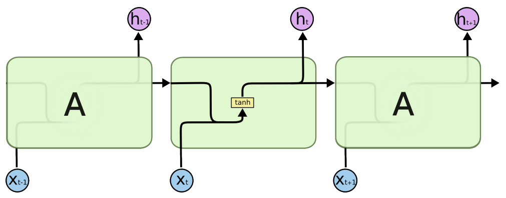
2.1.5.4 Multi-Stage Loss Optimization

Furthermore, a number of training methods have also been proposed to improve gradient degradation problems in the form of *multi-stage loss* optimization, where a neural network of depth N, will predict the desirable outputs at multiple stages within the network, such as $N/4$, $N/2$ and N, and then compute a separate loss for each output layer, which then can be used to compute gradients for what are effectively shallower networks and thus keep the gradients of the model within a suitable range for training. Such methods were used to train the original Inception model [Szegedy et al., 2015]. This method can be considered a form of simultaneous pretraining and training that combines ideas from layer-wise pretraining with vanilla joint training.

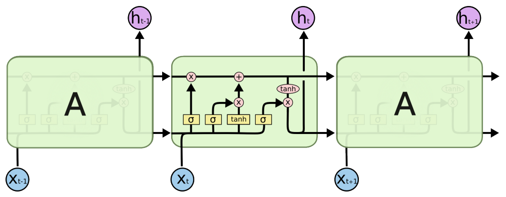
2.1.5.5 Skip-Connections

In [He et al., 2016] the authors carried out a study on the training dynamics of both shallow and deep neural networks. They found that models that exceeded 20 layers in depth, began to showcase a curious behaviour. The training losses of deeper networks were higher than that of shallower ones. This went in direct contrast to the established theoretical knowledge that deeper models with more parameters should be able to fit a function better, at least in the training phase, compared to ones with less depth and parameters. A good explanation for this would be that training is somehow hindered by the extra depth. A careful study of the gradients of such model reveals gradient-degradation issues. The authors proposed alleviating such gradient problems by introducing a *skip-connection* which adds to the output of every layer the input that went into that layer. Therefore introducing what was effectively a direct path for gradients to flow through, without having to go through the weights of a given convolutional layer. The results were positive, the problematic behaviour was reduced and the model, named a *ResNet* (from residual and network), was able to benefit from the extra depth, not only during training, but also at test time, achieving better generalization performance. The introduction of skip-connections started a new direction in model architecture research, and enabled training of networks thousands of layers deep.

In an attempt to enable even better gradient propagation throughout a neural network the authors in [Huang et al., 2017] introduced *DenseNets*. In this model architecture layers are not just skip-connected to their preceding layer, but instead, they are skip-connected to *every* layer that came prior to them, using a concatenation instead of a summation that the original ResNets used. Via this method, not only do DenseNets



(a) Recurrent Neural Network Cell



(b) Long Short Term Memory Cell

Figure 2.3: RNN vs LSTM

improve gradient degradation, but they are also more parameter efficient, as features required for many layers downstream can simply be accessed from early layers, rather than having to be relearned.

Inception, ResNets and DenseNets were empirically shown to achieve the top state of the art results in a number of image classification benchmarks including CIFAR10/100 and ImageNet at their time of development. ResNets and DenseNets have remained the the most popular model architectures in vision-related tasks for the last four years.

2.1.6 Recurrent Neural Networks

Recurrent Neural Networks [funahashi and nakamura, 1993] or RNNs are a neural network component that can model temporal data streams. They consist of what is effectively a fully connected layer, but with the distinction that it can receive information from past hidden states referred to as H. RNNs demonstrate a type of weight-sharing across time steps. However, since gradients have to be back-propagated through what can be a large number of fully connected layers, RNNs suffer from gradient degradation problems as discussed in [Hochreiter et al., 2001] and Section 2.1.5. The left picture in Figure 2.3 illustrates the internal structure of an RNN.

To improve the gradient propagation issues in RNNs, the *Long Short Term Memory* network LSTM [Schmidhuber et al., 1997] was proposed. LSTMs improve the gradient propagation issues by introducing what is effectively a form of skip-connection in between steps. The output of that skip-connection in an LSTM is called the *cell-state*, and it differs from identity skip-connections used in ResNets [He et al., 2015] in that the network can choose which data be thrown away and which data should be added to the the cell state. The choice of which data are thrown away is made using a layer called the *forget-gate* which consists of a multiplicative interaction with few learnable parameters and a sigmoid gate. The addition of new information in the cell state is achieved using a layer called the *input gate layer* which is composed of another multiplicative interaction between a variety of input sources. Once the network chooses which data to throw away and which new data to add to the cell state, it is ready to output its new predictions. However, it also needs to output a hidden state that keeps the context of the network. To choose what the context is, the current cell state is passed through a tanh activation and then multiplicatively interacted with the previous hidden state's information. At this stage the network decides what past context should be kept and which thrown away as well as which new context should be stored. At this point the network outputs the results. An illustration of an LSTM vs an RNN is shown in Figure 2.3.

2.1.7 Relational Layers

Most CNNs are built by synthesizing a number of processing blocks, usually consisting of a convolutional layer with carefully selected hyperparameters, followed by batch normalization, followed by a non-linearity, such as ReLU. Following such blocks is usually a fully connected network, which computes an output prediction whose type varies by task.

A fully connected layer applies the operation $f(x) = x_i \cdot W$ where x_i is a batch of vectors of dimensionality $B \times K$ where B is the batch size, indexed by i, and K are the number of features in the vectors, and W is a learnable weight matrix of dimensionality $K \times F$ where K is the number of input features, and F is the number of output features. Furthermore, the operation $\cdot$ for matrices is defined as $a \cdot b = \sum_{i=1}^{n} a_i b_i$. Once the operation of the fully connected layer is completed, the output volume has dimensionality $B \times F$.

Even though fully connected layers have proven to be integral parts of neural network architectures, they also have many shortcomings. For one, they produce significant generalization issues when faced with relational reasoning tasks (that is, tasks requiring awareness of the relational positioning of objects in a given image), among others. A *relational network* is a drop-in replacement for fully connected layers, that is far more robust when it comes to relational tasks, as well as tasks where relative positions of objects are important to the task at hand.

Given an input batch of convolutional features x_i with dimensionality b, k_{out}, h, w where b is the batch size, h and w are the height and width of the features, and k_{out} is the number of channels, a relational network considers the channel-wise features located at each of the spatial locations as *objects*, and then collects all possible combinations of such objects. For example, the channel-wise features of dimensionality b, k_{out} located at the position (i, j) where i is a row on the width dimension and j is a column on the height dimension, are one such object. And a concatenation of two such objects along with their coordinates, is considered a pair. Once all possible pairs are collected, we have a volume with dimensionality $(b, (h \times w)^2, 2 \times k_{out} + 2)$, where $(h \times w)^2$ is the number of pairs. These pairs are then propagated through a series of fully connected layers with non-linear activations, after which, we compute the sum of the resulting features on the pair dimension, resulting in an output volume of dimensionality (b, r), where r is the number of relational features that we have collected.

Once the relational layer is added following a CNN, it can be trained via gradient-based methods and backpropagation. The results showcased in [Santoro et al., 2017], show dramatic improvement of the performance of the system on relational reasoning tasks.

A relational network represents a new type of structural inductive bias for a network, that has some very desirable properties. Firstly, it only assumes the bare minimum of two objects for each relational feature to be detected, contrary to fully connected networks that are trying to find relationships across the objects at all spatial locations, therefore forming a type of, which might represent a more generalizable learning style. A simple example to motivate this would be to imagine an image classifier that detects various pets, trained only on images showcasing animals whose whole body is shown in full. If such a model equipped with a fully connected layer would then be tasked on generalizing on images of pets whose body is only partially shown, e.g. only faces or tails etc, then the model's generalization performance could suffer, as the model might have associated a cat with the presence of specific legs, tails and faces, while the absence of one might confuse the classifier. Whereas, a model trained with a relational network, might be able to perform well since it only cares about associations between pairs of objects.

Secondly, a relational network cares about the comparison of two feature vectors and their relative positions. Such an inductive bias would generalize better in the face of rotational domain shift, and potentially other linear and even non-linear domain shifts, assuming they are somewhat present in the data. The possibility for better sample efficiency for such domain shifts and stronger generalization is a very strong premise of relational networks.

It is also worth mentioning that a relational network can also be considered a convolutional layer with all possible dilation rates applied to it, if the convolution had a kernel shape of 2×1. Some attempts have been made to use dilated convolutional networks in relational reasoning with good success [Antoniou et al., 2018b].

2.1.8 Adversarial Attacks

A trained deep neural network, that has been evaluated to make correct predictions for a given training set, can easily be forced into producing erroneous predictions on those same data points by (what are most of the time) human imperceptible perturbations on those samples. Such perturbations can easily be generated by a suite of methods referred to as *Adversarial Attacks* [Goodfellow et al., 2015]. A notable type of adversarial attack is the gradient-based adversarial attack that directly optimizes an input region such that the loss of a model on a given sample increases, therefore causing the performance of the model to degrade. These types of attacks can also be used to explore a model and understand more about its decision boundaries. We use it in this thesis as a tool for acquiring adhoc information on meta-learned loss functions in Chapter 4.

2.1.9 Activation Functions

Utilizing depth in neural networks requires the usage of non-linear activation functions. If one where to try training a multi layer neural network within linear space then no matter how many layers one would add the network would be equivalent to a single layer linear network. In addition, activation functions are a powerful way of shaping the inductive bias of a model and thus choosing the types of representations it can learn.

There are a large number of activation functions that have been proposed in the last few years, and fewer proposed in the early days of neural networks. One of the first non-linear activation functions to be used was the sigmoid activation function. Formally, sigmoid can be defined as:

$$S(t) = \frac{1}{1+e^{-t}} \tag{2.2}$$

However, the sigmoid activation function was proven to cause diminishing and exploding gradients, mainly due to the saturating gradients in the higher positive and lower negative values that the function naturally provides.

To fix this problem, the *Rectified Linear Unit* [Xu et al., 2015] or ReLU was proposed. ReLU activation functions are cheaper to compute and greatly alleviate gradient propagation issues, as gradient saturation can now only be found in the negative part of the spectrum, while the positive part follows a linear function. ReLU can be expressed as:

$$f(x) = \max(0,x) \tag{2.3}$$

Where x are the inputs to an activation layer and $f(x)$ are the outputs.

Since the gradients of this function are zero when the input is zero or less the gradients do not saturate when in positive ranges and thus the network is optimized faster with less chances of gradient-degradation. Furthermore, the inductive bias enforced by a ReLU allows for easier information compression, as simply entering the negative region is enough to turn a whole unit off and throw away that information.

However ReLUs are not perfect. When at zero, they can cause gradient computation to be zero and thus result in *dead units* which will no longer update. Restricting a network in such a way will slow down optimization and the modeling capacity of the network. In an attempt to solve this problem, the *Leaky ReLU*[Xu et al., 2015] was introduced. The leaky ReLU can be expressed as:

$$f(x) = \left\{ \begin{array}{cc} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{array} \right\} \tag{2.4}$$

Where x are the inputs to an activation layer, α is the *leak rate* and $f(x)$ are the outputs.

Leaky ReLUs have the same outputs as ReLUs when the inputs are positive, but when they are negative instead of setting them to zero, they set them to be a fraction of the inputs using the leak rate. The leak rate, usually chosen to be 0.01 helps alleviate the issue of dead units by enabling gradient propagations in the negative input region. As a result, leaky ReLUs allow for improved convergence speed and increased modeling capacity for a neural network, compared to one with ReLU activation functions. The modeling capacity does not increase in the same sense as when we have more parameters. Instead, leaky ReLUs increase the *efficiency* or probability of use of the available units. It is also worth noting that α can also be set as learnable as demonstrated in [He et al., 2015].

Leaky ReLUs still have problems however, one of which is the fact that they are not zero-centered. It has been observed that inputting zero-centered or other normalized data into a neural network layer will produce a network with improved stability, faster convergence and increased generalization. Normally one can use a normalization layer to normalize activations, however attempts to include zero-centering within an activation function have also been made. The *Exponential Linear Unit* (ELU) [Clevert et al., 2015] is one such attempt. Expressed as:

$$f(x) = \left\{ \begin{array}{cc} x & \text{if } x >= 0 \\ a(e^x - 1) & \text{otherwise} \end{array} \right\} \tag{2.5}$$

In addition the authors in Clevert et al. [2015] claim that ELUs can work very well with CNNs and provided some empirical evidence for their effectiveness compared to other activation functions. However the debate for the best activation function is still ongoing.

Another activation function worth mentioning is the *tanh* activation function. It is precisely the same as functional form of sigmoid but ranges from -1 to 1. It is often used as an activation function on the final layer of generative models or in the circuitry of LSTMs and RNNs. Furthermore, both sigmoid and tanh activation functions are excellent as differentiable selection mechanisms in neural attention methodologies.

Tanh can be expressed as:

$$f(x) = tanh(x) = \frac{2}{1 + e^{(-2x)}} - 1 \tag{2.6}$$

Tanh suffers from the same saturating gradient problems as the sigmoid does.

2.1.10 Dropout

Dropout is a simple regularization technique for neural networks. The idea of dropout is to randomly choose a fraction of neural activations and set them to zero. By doing so, the model is forced to learn more features that are less reliant on one another, which produces more robust features as a result.

The fraction of units dropped is controlled by a variable called *dropout-rate*. At training time dropout routinely drops units, while at evaluation time dropout keeps all units and instead scales the units by a factor of $s = r^{-1}$ where s is the scale coefficient and r is the dropout rate used. The intuition behind the evaluation time modification is that we require deterministic behaviour from a model and thus to achieve that, the existing features are not dropped, but instead scaled by a coefficient such that their magnitudes take a form that the network has previously seen in training.

Dropout can also be viewed as a form of noise injection into the neural network, as it is effectively a type of bernoulli noise multiplied by a network's intermediate activations. Furthermore, it can also be considered a form of model ensembling as we are continuously sampling subnetworks of a given network and making sure that they performs well. The test time scaling applied on dropout, can be considered as a form of ensembling over all possible subnetworks generated by the method. From this perspective, applying the scaling coefficient at test time, effectively equals taking the mean of all possible subnetworks, further providing mathematical intuition as to why scaling at test time is a good practice.

2.1.11 Data Augmentation

Deep neural networks, are known for requiring large amounts of data to be successfully trained. In general, more data usually leads to better generalization performance. One way to increase the amount of data without additional data collection is data augmentation [Krizhevsky et al., 2012b, He et al., 2016, Cubuk et al., 2018]. Data augmentation covers a range of methods that can be applied to existing data points in order to create new data-points. For example, simply cropping parts of an image can create new data-points with the same class as the original. Some of the most notable data augmentation methods include randomly rotating an image, randomly cropping an image, adding random noise to an image, applying dropout to an image, mirroring an image, horizontal or vertical translation (shifts) of an image, as well as *warping* an image. In addition to manual data augmentation methods, recently, work by various authors produced *learned* data augmentation models. Learning data augmentations given a dataset can be very beneficial since the augmentations learned are no longer restricted to few possible data manipulations, instead they can be data-specific and highly complex.

2.2 Meta-Learning

2.2.1 Preface

In this section we will define and motivate meta-learning as a tool for problem solving, and then present some historical context, before we move on to reviewing related meta-learning work. However, the literature review is not exhaustive, as its purpose is to provide the reader with all the information they need to understand and follow the work within this document, hence we will keep our coverage of meta-learning methods to those that deal with neural networks. For an exhaustive survey of meta-learning in neural networks, please refer to [Hospedales et al., 2020], a recent survey paper, co-authored by the author of this thesis.

2.2.2 Motivation

Machine learning models were originally learned as simple linear models or *Support Vector Machines* (SVMs), on top of human-crafted features. Model performance depended heavily on the quality and relevance of such human-crafted features. Deep learning allowed the automation of feature discovery, by directly learning good features given a task, and in doing so improving the performance of machine learning systems dramatically, while substantially speeding up the discovery of good models for a given task. This also freed up the time of many human experts, previously working on feature discovery, which could then begin to work on improving feature-learning and therefore speeding up the progress in the field of machine learning even further. Meta-learning presents a new learning paradigm where the learning algorithm that learns the models can be automatically learned. Models learned via meta-learning have already achieved various notable improvements over human invented counterparts, however, meta-learning is still at its infancy. Further work in this field might enable further advancement of machine learning as well as increasing the pace of that progress by replacing much of the need for research in learning algorithms and thus allowing machine learning researchers to focus on work that will have increasingly wider impact.

Furthermore, there are many hard problems in machine learning that are naturally hierarchical that naturally lend themselves to meta-learning. For example, few-shot learning will almost always require a strong prior built in to enable rapid and effective learning. Therefore, meta-learning such a prior that can facilitate few-shot learning with high generalization power can be a natural way to attempt to solve such a problem.

2.2.3 Definition

Meta-learning [Schmidhuber, 1987, Thrun and Pratt, 1998, Hospedales et al., 2020] can be defined as a learning paradigm where a learning algorithm or *learner* is automatically learned using end-to-end differentiable optimization. Such a meta-learned algorithm is learned such that models that are then trained with that algorithm can solve a predefined set of tasks in a way that maximizes the performance of those models on a number of predefined objectives (e.g. model classification accuracy).

More formally, given a plentiful set of training tasks $\mathcal{D} = \{d_{task}, \mathcal{L}\}$, where d_{task} denotes a task's data, and $\mathcal{L}$ denotes a task's objective and a learner (or meta-model) $g(d, W)$ where g is the functional form of the learner, d are task-data to be processed by the learner and W are learnable parameters, the optimization problem to be solved by a meta-learning algorithm would be:

$$\min_{W} \mathbb{E}\ \mathcal{L}(g(d_{task}, W)) \tag{2.7}$$

Equation 2.7 is also often referred to as the *outer-loop optimization process*. In more detail, during this outer-loop optimization process, the meta-learning algorithm should extract *across-task* knowledge and distill it within the learner's parameters W, while discarding task-specific knowledge, to keep the learner as task-agnostic as possible. At each meta-training iteration, the learner g, will receive task-data d_{task} and from that extract *task-specific* knowledge that should then be distilled in a base-model $f(x, \theta)$ where f is the functional form of the model, θ are the learnable parameters and x are input-data points that are used to produce a prediction. This process, where the learner receives task-data and produces a base-model is referred as the *inner-loop* optimization process. At this point the base-model will then be evaluated on a given task's objective. The resulting performance metric can then be used to compute gradients for the learner's parameters W.

Furthermore, meta-learning does not strictly involve only two learning levels. In theory, there can be any number of learning abstraction levels. Such an N-level meta-learning system would apply learners learned at a higher level of abstraction, to train learners at the level directly below that of the upper-level learner. This process is usually terminated when the lower-most level is reached and the model produced at that level is evaluated on some target task. Once the performance metrics on such a target-task are computed, back-propagation can be used to update all the learners starting from the lower-level ones and moving all the way to the top-most ones. Meta-learning is usually reserved for models that learn using two such levels, therefore learning a single learner (i.e. a meta-model) that can learn a lower level model (base-model) such that the base-model can perform well on some target task. However, additional learning hierarchy is also possible and should be included in the term meta-learning.

2.2.4 Brief History of Meta-Learning

Schmidhuber [1987] set the theoretical framework for a new family of methods that can learn how to learn, using *self-referential* learning. Self-referential learning involves training neural networks that can receive as inputs their own weights and predict updates for said weights. Schmidhuber further proposed that the model itself can be learned using evolutionary algorithms. In the same year, Hinton and Plaut [1987] proposed the usage of *fast-weights*, previously framed by v.d. Malsburg [1981], where two weights per neural network connection instead of one as a means of recovering older parameter states. The first weight is the standard *slow-weight* which acquires knowledge slowly (called *slow-knowledge*) over optimizer updates, whereas the second weight or *fast-weight* acquires knowledge quickly (called *fast-knowledge*) during inference. The fast weight's responsibility is to be able to *deblur* or recover slow weights learned in the past, that have since been forgotten due to optimizer updates.

After the introduction of meta-learning, one can see a rapid increase in the usage of the idea in multiple different areas. Bengio *et al.*[Bengio et al., 1990, 1995] proposed systems that attempt to meta-learn biologically plausible learning rules. Schmidhuber *et al.*continued to explore self-referential systems and meta-learning in subsequent work [Schmidhuber et al., 1996, Schmidhuber, 1993]. S. Thrun *et al.*took care to clearly define the term *learning to learn* in [Thrun and Pratt, 1998] (which was already used by in [Schmidhuber, 1987]) as an alternative to meta-learning and proceeded to explore and dissect available literature in meta-learning in search for a general meta-learning definition. Proposals for training meta-learning systems using gradient descent and backpropagation were first made in 1992 [urgen Schmidhuber, Schmidhuber, 1991] followed by more extensions in 2001 [Hochreiter et al., 2001, Younger et al., 2001]. Additional overviews of the meta-learning literature shortly followed [Vilalta and Drissi, 2002]. Meta-learning was first used in reinforcement learning in [Storck et al., 1995, Wiering and Schmidhuber, 1998a, Schmidhuber et al., 1998, Wiering and Schmidhuber, 1998b, Schweighofer and Doya, 2003] after which came the first usage of meta-learning in zero-shot learning by Larochelle *et al.*[Larochelle et al., 2008].

2.2.5 Meta-Learning for Few-Shot Learning

Benchmarks and Setting: Progress in any scientific field is often quantified and directed by thoroughly framed benchmarks [Russakovsky et al., 2015]. In machine learning, such benchmarks usually consist of a dataset and a task that a model should perform well on, while generalizing well after the test set. In meta-learning, benchmark design is more complex as one is (meta) training a learner by learning to learn on a set of training tasks, such that the learner can learn well on previously unseen tasks in the test set. In this section, we will introduce some of the more established few-shot learning benchmarks suitable for meta-learning.

The *set-to-set* few-shot learning setting [Vinyals et al., 2016] has been vital in framing few-shot learning as a meta-learning problem. In set-to-set few-shot learning, we have a number of tasks, which a model seeks to learn. Each task is composed of two small sets. A small training-set or *support-set* used for acquiring task-specific knowledge, and a small validation-set or *target-set*, which is used to evaluate the model once it has acquired knowledge from the support-set. The tasks are generated dynamically from a larger dataset of input-output pairs. The dataset is split into 3 subsets beforehand, the *meta-training*, *meta-validation* and the *meta-test* sets, used for training, validation and testing respectively.

The above setting is usually applied on some established few-shot learning datasets, such as Mini-ImageNet [Vinyals et al., 2016, Ravi and Larochelle, 2016], Tiered-ImageNet [Ren et al., 2018], SlimageNet [Antoniou et al., 2020], CUB-200 [Chen et al., 2019] and Omniglot [Vinyals et al., 2016]. These benchmarks restructure the previous large number of classes and samples into small few-shot learning tasks therefore building the meta-train and meta-test sets.

Dataset Diversity, Bias and Generalization: The above is effective for generating a large number of tasks, but often lacks diversity which makes the performance of these systems on real world dataset shifts harder to estimate. For example, shifting between different animal categories in Mini-ImageNet or birds in CUB is a rather weak test of transferability. Ideally we'd like to test generalization in more extreme shifting (satellite, medical, agricultural, underwater, etc), or perhaps going from simulations to real data.

There is much work to be done in few-shot learning, as even in the many-shot setting fitting a model with a wide range of distributions is non-trivial [Rebuffi et al., 2017], as is generalization in previously unseen distributions [Li et al., 2019, Balaji et al., 2018]. Meta-learned few-shot learners have been shown to have a large generalization problem when the distribution of training and test is very different [Chen et al., 2019]. New datasets with a wide range of tasks have been suggested in Meta-Dataset [Triantafillou et al., 2020] and CVPR cross-domain few-shot challenge [Guo et al., 2019]. Meta-Dataset combines a large array of existing vision tasks, to thoroughly evaluate a few-shot learner's ability to generalize in a variety of domain shifts. Meanwhile, [Guo et al., 2019] showcases more extreme domain shifts that task a learner to generalize from the everyday images of ImageNet to medical, satellite and agricultural images. Recent work has begun to try and address these issues by meta-training for domain-shift robustness as well as sample efficiency [Tseng et al., 2020]. Generalization issues also arise in applying models to data from under-represented countries [de Vries et al., 2019]. Another recent dataset that could facilitate research in few-shot learner generalization is [Gondal et al., 2019], which offers samples across environments from simulation, to high definition simulation and real-world.

Models: Once meta-learning was shown to be very effective in learning to few-shot learn, a multitude of methods showcasing unprecedented performance in few-shot learning surfaced. Matching Networks [Vinyals et al., 2016], and their extension Prototypical Networks [Snell et al., 2017a] were some of the first methods to showcase strong few-shot performance, using learnable embedding functions parameterized as neural networks in combination with distance metrics, such as cosine and euclidean distance. Unsupervised and supervised set-based embedding methods were also developed for the few-shot setting [Edwards and Storkey, 2017].

Further advancements were made by gradient-based meta-learning methods that explicitly optimized themselves for fast adaptation with a few data-points. The first of such methods, Meta-Learner LSTM [Ravi and Larochelle, 2016] attempted to learn a parameter-initialization and an optimizer, parameterized as neural networks for fast adaptation. Subsequently, additional improvements came from *Model Agnostic Meta-Learning* (MAML) [Finn et al., 2017b] and its improved versions *Meta-SGD* [Li et al., 2017b] and *MAML++* [Antoniou et al., 2018a], where the authors proposed learning a parameter-initialization for a base-model that is adapted with standard SGD for a number of steps towards a task.

Relational Networks [Santoro et al., 2017], that were originally invented for relational reasoning, demonstrated very strong performance in few-shot learning tasks [Santurkar et al., 2018]. The effectiveness of relational networks in a large variety of settings made them a module often found in meta-learning systems.

In Reinforcement Learning, meta-learning has been used to learn an unsupervised loss function for sample-efficient adaptation in [Yu, 2018]. Furthermore, in [Houthooft et al., 2018b], the authors propose a method that learns a state and reward conditional loss function to train a policy network, using RL inner phase and an evolutionary algorithm outer loop. Furthermore, in [Sung et al., 2017b] the authors propose learning a supervised loss function for few-shot learning, as well as a state and reward conditional loss function for training RL agents.

Shortly after, substantial progress was made using a hybrid method utilizing embeddings, gradient-based methods and dynamic parameter generation called Latent Embedding Optimization [Rusu et al., 2018].

Semi-supervised learning via learning label-free functions were also attempted in [Rinu Boney, 2018]. Transductive learning for the few-shot learning setting has previously been attempted by learning to propagate labels [Liu et al., 2018].

Ideas by on self-referential learning [Schmidhuber, 1987, Schmidhuber et al., 1996] and memory networks[Santoro et al., 2016] are revisited in [Munkhdalai and Yu, 2017a] where authors propose a framework for few-shot learning that combines ideas from HyperNetworks [Ha et al., 2016], differentiable external memory [Santoro et al., 2016] and LSTM Meta Learner [Ravi and Larochelle, 2016] to train an end-to-end few-shot learning system. The system works by first applying a base-network on a support set, and computing the support set crossentropy loss and the gradients with respect to that loss. Once acquired the gradient information are sent to the meta learner along with the support set data. The meta-learner then generates new weights for the base-network and for itself (i.e. the meta-learner) thus updating itself in the process. Once the new weights are acquired they are saved into a differentiable memory bank, and then using yet another network they apply attention over the memory bank which then selects which weights should be copied over to the base and meta network. This process can continue for a number of inner loops. The results show SOTA results across all one-shot learning schemes on both Omniglot and Mini-Imagenet.

2.2.6 Meta-Learning for Initialization Learning

Static Initializations: Neural network initializations have always been an important factor on both training and generalization performance. In early days of neural networks, ineffective initializations lead to gradient-degradation issues which kept the training performance of deep neural networks very poor. Various approaches were proposed to counter that as explained in Section 2.1.5. However, the importance of initializations does not stop at gradient degradation. A network pretrained on ImageNet will, most of the time, be able to perform better on new domains that have say, limited data or heavy class imbalance than a randomly initialized one. For that reason meta-learning approaches have been proposed into learning initializations that enable a number of different learning paradigms to be achieved. Few-shot learning being one of the key ones, by learning an initialization for fast adaptation and strong generalization. Other notable examples include applications in RL using MAML [Finn et al., 2017a, 2018, 2019] for fast adaptation to new tasks, as well as *editable networks* [Sinitsin et al., 2020] trained such that when a learned model is optimized on a single new data item, the model will retain all past knowledge and add the knowledge of that sample to its knowledge bank. Once such methods were proven to work well there was an explosion of work focussing on meta-learning initialization parameters for specific layers. In addition, work was carried out on learning a large embedding function that remains fixed in the inner loop, while only allowing just a couple of layers to be updated to improve generalization [Antoniou and Storkey, 2019b, Qiao et al., 2018a, Rusu et al., 2019], as well as by separating out weights and biases [Sun et al., 2018]. Furthermore, work was also done that focused on optimizing a vector embedding itself on the inner loop [Yoonho Lee, 2018, Rusu et al., 2019]. Work in this thesis also presents methods to stabilize and improve MAML as in the published paper [Antoniou et al., 2018a].

Dynamic Initialization: In addition to learning static initializations for a neural network, one can also learn a function that generates weights given some task information (e.g. data, architecture etc.). This might be desirable as learning a single static initialization may force solutions that can only work well for a fraction of the target tasks, while failing on the remainder of the tasks, whereas with a dynamic initialization, a model can be initialized far from another, therefore achieving far wider solutions.

The idea of training networks that can predict weights for another network can first be found in the literature in [Schmidhuber, 1992] where the authors propose an alternative to LSTM networks. A standard LSTM, receives an input x_i and a hidden state h_i which uses to produce an output y_i and the next hidden state h_{i+1}. Hidden state h_{i+1} is then fed into the LSTM along with the next input in order to produce the next step in the process. The hidden states hold information from all previous steps and act as short-term memory for the network, which then uses its internal gates to change the current state and compute the future one. One shortcoming of the standard LSTM is that once learned, the weights within its cell are kept static and shared across all time steps. This seems very unintuitive since future operations might require different weights for optimal results. Thus the authors in [Schmidhuber, 1992] propose using a *fast-weights* network that given a hidden state and an input can predict the weights of the cell that will process that state and input. By doing so they argue the network can achieve greater flexibility and better generalization.

In a modern re-enactment of [Schmidhuber, 1992] the authors in [Ha et al., 2016] propose a model and framework for learning to generate weights. They train a neural network, called a *Hyper-Network*, to predict the weights of another network (i.e. a base-network). In standard neural network training the weights of the base-network are learned directly, whereas they propose to learn to predict the weights using another network. Their approach allows for dynamic generation of weights conditioned on some input. They both explore static networks (conditioned only on the layer index) and dynamic networks (conditioned on input data). The dynamic variants are used in the context of recurrent neural networks and allow for recurrent networks that can change their own weights at each timestep. The authors empirically demonstrate state of the art results on the Penn Treebank[Marcus et al., 1993] and Hutter Prize Wikipedia [Hutter] Language Modeling tasks. They also attempted to learn static hyper-networks for a variety of state of the art image classification architectures but their results were inferior to training the network weights directly. Strictly speaking Hyper-Networks themselves are not a meta-learning model, but with very few modifications, they can become very powerful components for meta-learning dynamic initializations.

Building on Hyper-Networks the authors in [Brock et al., 2017] propose learning to generate weights for arbitrarily structured neural network architectures. Once trained, the model can then be used to instantly generate weights for a variety of previously unseen architectures. The performance of the generated networks is shown to be lower than the performance of a fully trained counterpart. However, the relative performance of the generated networks generalizes to their fully trained counterparts. Because of this, the model can be used to efficiently explore architectures on a given dataset, speeding up architecture search by a significant margin.

More recently, work by [Qiao et al., 2018a, Rusu et al., 2019] demonstrated how a system could learn dynamic initializations given task data. They do so by first learning an embedding for data-points using standard CNNs, whose outputs are further processed by a Relational Network, to generate a task-level embedding. Then that task level embedding is send into a weight generator network that generates weights for a given task. Those weights can then be further updated within a given task's inner-loop. The outer loop can learn the sample-level and task-level embeddings as well as the parameter generator. The results showcased SOTA performance in few-shot learning.

2.2.7 Meta-Learning for Learning Schedulers

A learning scheduler is an important function whose impact on all aspects of training and generalization has been shown to be significant [Loshchilov and Hutter, 2016]. Meta-learning provides new directions for directly learning such learning schedulers, that can be targeted to have any desirable properties as long as those can be framed into an objective.

Some initial work in this direction was done in [Antoniou et al., 2018a] where the authors proposed learning per-layer, per-step learning rates, to achieve quick adaptation for a few-shot learning task, with much improvement in generalization performance and convergence speed. Work that tried to learn a per-parameter learning rate was also done in [Li et al., 2017b], demonstrating improved generalization and fewer inner steps needed to get excellent performance.

With this being said, there is still much work to be done towards learning learning schedulers for many-shot problems as well as moving from learning of static learning rates to data and step conditional learning schedulers that can dynamically generate the right learning rate for the right task.

2.2.8 Meta-Learning for Architecture-Component Learning

Learning neural network architecture components directly instead of manually building them is highly desirable. This is because it generally speeds up discovery of useful architectures when compared to manual discovery by humans, automates the process therefore making it more repeatable and cheap, but also because those architectures can be learned to exhibit a number of desirable properties, such as faster convergence, better generalization, better domain-transfer, reduced computational cost among others. The reality here is that as long as a trait can be cast as an objective, then it can be used to meta-learn an architecture that can optimize that.

Neural Architecture Search (NAS) is especially challenging for a number of reasons: (i) The inner loop usually involves a full training session of a model on a large dataset, which makes such evaluations and backpropagation through it very expensive, which leads to shortcuts such as sub-sampling the train set, less iterations on the inner loop, and ultimately greedy interleaved optimization on both outer and inner loop parameters [Liu et al., 2019a] as in online meta-learning. (ii.) Defining the search space itself is often a very complicated process, because the space of architectures itself is most of the time non-differentiable and discrete. As such, solutions usually involve either cell-level search [Zoph and Le, 2016, Liu et al., 2019a] to restrict the search space at the block-level while relying on RL [Zoph and Le, 2016] to learn, discrete gradient estimators using gates like softmax to select from a number of potential layer types within a block [Liu et al., 2019a, Xie et al., 2019], and evolution [Real et al., 2018, Stanley et al., 2019].

Architecture search [Stanley et al., 2019, Liu et al., 2019a, Zoph and Le, 2016, Real et al., 2018, Elsken et al., 2019a], can be framed as a meta-learning task when an architecture learned on the outer loop, is used in the inner loop to achieve a predefined type of learning that can perform well on a predefined set of tasks. Some notable examples include: (i) NASNet [Zoph and Le, 2016, Zoph et al., 2018] where the search space is restricted to cell-level learning, and defined as a string generated by an RNN which indicates what operations should be at what parts of the cell-tree, optimized using RL. (ii) Reqularized Evolution [Real et al., 2018] where the authors use NASNet's search space but optimize it using regularized evolution, i.e. standard tournament based evolution with removal of oldest individuals after every iteration. (iii.) *DARTS* [Liu et al., 2019a] where the authors carefully cast the space of cell architectures as a sequence of softmax selections over a number of pre-selected operations, thus making the search space differentiable. Learning the architecture then corresponds to jointly learning the softmax weights with the network parameters. This allows architecture learning to be sped up by 2-3 levels of magnitude both in computational overheads and wall-clock time. (iv) T-NAS [Lian et al., 2020] where the authors utilize the DARTS search space, but train it using a data-flow that enforces the architecture to be learned using very few data-points and very few updates, while keeping the generalization performance high. As a result of learning such softmax weights, they achieve few-shot architecture search. Once trained, these weights can be adapted to new tasks within seconds rather than days.

Activation function learning can also be viewed as a NAS problem [Prajit Ramachandran, 2017]. Even though ReLU and its subtypes are currently the most used in the deep learning literature, meta-learning has been able to learn succesful alternatives such as the Swish activation function [Prajit Ramachandran, 2017] with RL in a space of symbolic activation functions. Swish has been demonstrated to be successful in a number of new architectures and other deep learning applications [Tan and Le, 2019, Howard et al., 2019].

Few-shot learning models usually use manually-crafted architectures to form its learnable components [Finn et al., 2017a, Snell et al., 2017b, Sung et al., 2018], however, learning an architecture directly for the task of few-shot learning is also very effective [Kim et al., 2018a, Elsken et al., 2019b]. Furthermore, similarly to fast-adating initializations in MAML [Finn et al., 2017a], one can learn architectures that can achieve fast adaptation [Lian et al., 2020] or architecture priors [Shaw et al., 2019] that can quickly learn architectures for a given task.

2.2.9 Meta-Learning for Optimizer Learning

Optimizers are yet another neural network component whose effects on training and generalization are immense. Meta-learning can again be used to learn such optimizers instead of using manually-invented ones. Learning to learn optimizers in the form of a neural network has been a long standing idea in machine learning, with some early work as early as 1993 [Schmidhuber, 1993]. In [Andrychowicz et al., 2016] the authors trained an LSTM, that given a set of gradients to predict updates for a neural network. The algorithm was trained on small neural networks and then used on larger neural networks with varying degrees of success. Issues started to arise when the optimizer was used on very deep networks or networks with non-linearities the optimizer was not trained on. This highlighted the importance of training the optimizer on a wider range of tasks and architecture types for future work. More recent attempts at optimizer learning exists in [Wichrowska et al., 2017a] where the authors present a framework and model for learning an optimizer. The optimizer neural network is based on a Hierarchical RNN augmented with architectural features that mirror the structure of optimization tasks. This optimizer network is trained on non-neural network problems, which include convex loss functions, logistic regressions, multi-dimensional oscillating valley problems, noisy gradient toy problems and sparse gradient toy problems. Once trained, the optimizer recurrent neural network can generalize very well and exceed both Adam and RMS-Prop on many types of neural networks, including InceptionV3 and ResNet.

Learning optimization algorithms can be framed as a reinforcement learning task as recent work in [Li et al., 2017b] where the authors propose a framework driven by reinforcement learning in order to learn optimization algorithms using a neural network. The optimization algorithms are framed as a number of actions, executed by a neural network, given a state. Experiments were ran on linear regression, robust linear regression and neural network classifiers. The results indicate that the automatically learned optimization policy outperforms all the manual alternatives.

Learning optimizers was recently attempted in [Bello et al., 2017a], where the authors learn a recurrent neural network that can generate a string that defines the update for the network. After sampling an update rule they train a neural network to convergence on a variety of datasets. Once finished they use the final test accuracy as the reward signal and repeat the process multiple times. Once converged the system produced 2 notable update rules called *Power Sign* and *Add Sign* that are shown to outperform all commonly used optimizers, including Adam [Kingma and Ba, 2014], RMSprop [Tieleman and Hinton, 2012] and SGD [Krizhevsky et al., 2012a].

Optimizers have also been learned in the form of a neural network, such as an LSTM in [Ravi and Larochelle, 2016, Li and Malik, 2017] by casting the LSTM as an inner loop optimizer that receives gradients, losses and other state information and generate parameter updates. In addition, instead of directly generating gradients, other ideas have also been proposed in the form of fixed learning rates [Antoniou et al., 2018a, Li et al., 2017a] to more advanced conditioning curvature gradient information [Park and Oliva, 2019].

Parameter-learning can also be combined with optimizer learning very succesfully as shown in [Ravi and Larochelle, 2016, Li et al., 2017b]. Optimizer learning has been effectively applied on few-shot learning [Ravi and Larochelle, 2016] as well as many-shot learning [Andrychowicz et al., 2016, Wichrowska et al., 2017b, Bello et al., 2017b].

2.2.10 Meta-Learning for Loss Function Learning

Just as with any other learning-related mechanism in neural networks, loss functions can also be meta-learned. This can be achieved by learning a loss function on the outer loop that is used on the inner loop in conjuction with an optimizer to update the base-model. Such loss functions are usually parameterized as a neural network that receives a number of inputs, such as predictions, parameters, task-data and labels, and return a single scalar value that is considered the loss value. That value can then be used to compute gradients with respect to a base-model's parameters which can then be fed into an optimizer to update the network. Such loss functions can have learn to exhibit a number of useful qualities such as being more efficient and effective on the given task [Houthooft et al., 2018a, Sung et al., 2017a, Zhou et al., 2020], achieving faster adaptation and generalization [Denevi et al., 2018, 2019, Gonzalez and Miikkulainen, 2019], or more effective domain shift [Li et al., 2019]. In addition, such losses can be learned to work without labels, in both unsupervised and transductive settings.

Loss function learning forms parts of work in meta-learning self-supervised learning [Doersch and Zisserman, 2017, Pathak et al., 2016], as well as auxiliary task learning, where an auxiliary task is set up as a secondary loss function to optimize [Jaderberg et al., 2017] or by predicting alternative labels for a given dataset in [Liu et al., 2019b]. Furthermore, in cases of multi-task learning, the weighting of each task loss can also be meta-learned as in [Lin et al., 2019], when parameterized as a set of weights to be learned on the outer-loop and then used on the inner-loop to maximize the performance on the target objective.

2.2.11 Meta-Learning for Data Augmentation Learning

Data augmentations are data transformations that preserve class semantics but challenge the model to learn more robust representations, which aids generalization performance. Meta-learning can also be used to learn such data-augmentation strategies, by casting the data augmentation function as an inner loop editor of a given inner-loop training dataset with the goal of improving generalization on the inner-loop validation set [Cubuk et al., 2019]. Methods used to learn such data-augmentations often involve RL as many such transformations are non differentiable [Cubuk et al., 2019], as well as discrete gradient-estimators [Li et al., 2020], or evolutionary [Volpi and Murino, 2019] methods. sGAN-based data augmentation methods have also been applied in such inner-loops adhoc with some evidence to support that they can improve few-shot learning [Antoniou et al., 2017], however, more work is needed to further explore this hypothesis, with perhaps joint learning of the GAN via both GAN losses and meta-learning losses.

2.2.12 Unsupervised Meta-Learning and Meta-Learning Unsupervised Learning

In the meta-learning literature, there exist two main variants of meta-learning involving unsupervised learning. In the first one the meta-objective of the outer loop is unsupervised, and therefore the learner itself is learned without any labels available. We refer to this case as *Unsupervised Meta-Learning*. In the second variant, meta-learning is used as a means to learn an unsupervised inner loop task. The outer objective in this case can be anything from supervised, unsupervised or reinforcement based. We refer to this as *Meta-Learning Unsupervised Learning*.

Unsupervised Meta-Learning [Hsu et al., 2018, Khodadadeh et al., 2019, Antoniou and Storkey, 2019a] aims to relax the conventional assumption of an annotated set of source tasks for meta-training, while still producing good downstream performance for supervised few-shot learning, by utilizing self-supervised learning methods for task building and outer loop training. Typically such synthetic source tasks are constructed without supervision via clustering or class-preserving data augmentation.

Meta-Learning Unsupervised Learning aims to use meta-learning to train unsupervised learning algorithms that work well for downstream supervised learning tasks. One can train unsupervised clustering algorithms [Metz et al., 2019, Garg and Kalai, 2018, Jiang and Verma, 2019] or losses [Antoniou and Storkey, 2019b, Rinu Boney, 2018] such that downstream supervised learning performance is optimized. This helps to deal with the ill-definedness of the unsupervised learning problem by transforming it into a problem with a clear (meta) supervised objective.

2.2.13 Meta-Learning Continual and Online Learning

Continual Learning refers to the ability to learn tasks presented in a sequence, with minimal or no forgetting of past task information. Such learning should ideally be done without requiring to store all previously seen task information for retraining against forgetting [Chen and Liu, 2018]. Deep Neural Networks as they stand today, struggle to perform this type of learning, often forgetting very important information from older tasks, a phenomenon referred to as *catastrophic forgetting*. Meta-learning is naturally an ideal tool for such a problem, and as such have began being used to improve continual learning models. The key requirements for continual learning can be framed as a task and objective, for example by defining a sequence of learning episodes in which the support set contains one new task, but the query set contains examples drawn from all tasks seen until now [Vuorio et al., 2018, Flennerhag et al., 2020]. With this meta-objective design, various meta-representations can be trained so as to improve continual learning performance. For example: weight priors [Ren et al., 2019], gradient descent preconditioning matrices [Flennerhag et al., 2020], or RNN learned optimizers [Vuorio et al., 2018], or feature representations [Javed and White, 2019].

Of relevance here are Editable Networks [Sinitsin et al., 2020] where a model is learned such that when it has been trained on a large dataset, it can be retrained on single sample tasks, without losing any past information while performing the new task very well.

Online Learning also focuses on sequential learning, but puts more emphasis on adapting to new samples as an agent is exposed to them. Meta-learning methods that can be learned to perform well on such tasks include [Finn et al., 2019].

Benchmarks There exist a number of benchmarks for continual learning that work quite well with standard deep learning methods. However, most of those benchmarks cannot readily work with meta-learning approaches. Most of them would require adjustments to their task generation routines to include a large number of explicit training sets and an explicit evaluation sets. Some early steps were made towards defining meta-learning ready continual benchmarks in [Finn et al., 2019, Vuorio et al., 2018, Javed and White, 2019], mainly composed of Omniglot and perturbed versions of MNIST. However, most of those were simply tasks built to demonstrate a proposed method. More explicit benchmark work can be found in Antoniou et al. [2020], where Continual Few-Shot Learning is defined as a new type of setting to be tackled, and a benchmark suite is proposed for meta and non meta-learning approaches alike. In this setting, a task is composed by a number of small training sets, each potentially made of different classes, after which the learned model should generalize well on previously unseen samples from all the tasks it learned from. The benchmark proposes the usage of Omniglot and SlimageNet [Antoniou et al., 2020] as the datasets to be used.

2.2.14 Meta-Learning for Backpropagation Learning and Biologically Plausible Learning

Deep neural networks are almost always trained via means of backpropagation based on gradient-descent and the chain rule. An interesting direction for meta-learning would be to parameterize the backward pass elements themselves as neural networks, and to learn the backpropagation rules. Some existing work in this domain includes learning unsupervised rules [Metz et al., 2019], biologically plausible learning [Bengio et al., 1990, Miconi et al., 2018, 2019] and neuromodulation [Miconi et al., 2019].

2.2.15 Non Meta-Learning Few-shot Learning

In the past year, there have been a number of very effective non meta-learning few-shot learning methods that have been able to achieve top state-of-the-art results. This has began to raise doubts about the need of meta-learning, both within the few-shot learning setting as well as in general. In [Chen et al., 2019] the authors showcased that pretraining a model on the meta-training dataset using data-augmentation and fine tuning on a given meta-test task, produces very good performance, contrary to original papers that did not include data-augmentation. Moreover, deeper architectures with such simple training regimes have shown to reduce the performance difference between meta-learning and non meta-learning methods [Triantafillou et al., 2020]. Most recently the authors in [Wang et al., 2019] have empirically showcased that simple L2-normalization, applied on the features of a nearest neighbour classifier can achieve top SOTA results across all few-shot learning tasks. Because of this there is an ongoing debate about whether the issue here is that the few-shot learning benchmarks we currently use are not appropriate to properly showcase the effectiveness of meta-learning, or whether simply put, the classification task itself, when coupled with a relatively weak transfer dataset such as Mini-ImageNet and CUB, can best be solved by such simple methods. Various meta-learning methods, including the one summarized in Chapter 4 have also converged to the conclusion that a feature reqularizer is a very effective means of improving the generalization performance of a meta-learning system.

Chapter 3

How to train your MAML: Improving Model Agnostic Meta-Learning

3.1 Introduction

In this chapter we propose MAML++ , an improved meta-learning framework that offers the flexibility of MAML along with many improvements to its significant training and tuning problems, such as robust and stable training, automatic learning for most of the system's hyperparameters, greatly improved computational efficiency both during inference and training and significantly improved generalization performance. MAML++ is evaluated in the few-shot learning setting where the system is able to set a new state of the art across all established few-shot learning tasks on both Omniglot and Mini-Imagenet, performing as well as or better than all established meta learning methods on both tasks.

This chapter's contributions are:

1. Reduction of gradient-degradation issues in MAML using Multi-Step Loss Optimization. This leads to improved training stability for MAML.
2. Reduction of hyperparameters and improvement of generalization via learning of per-step, per-layer learning rates for MAML.
3. Reduction of computational overheads of MAML via derivative-order annealing.
4. Proposal of a new type of Batch Normalization, specially designed for meta-learning. This leads to improved convergence speed and generalization performance.

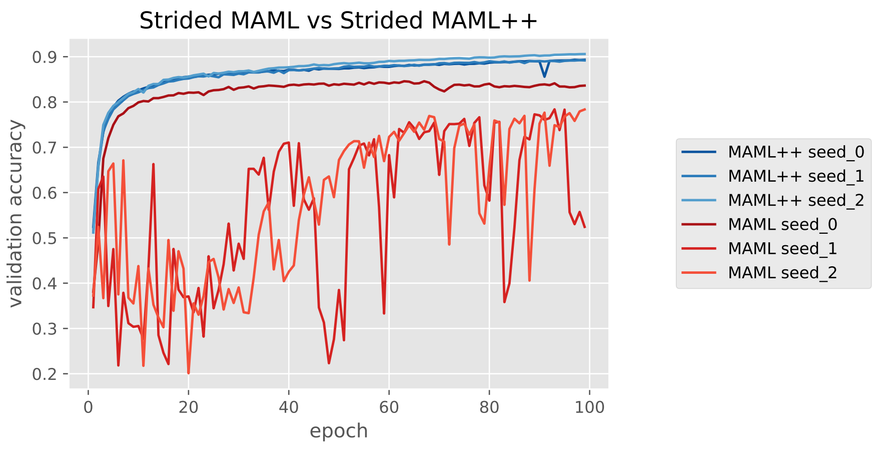


Figure 3.1: Stabilizing MAML: This figure illustrates 3 seeds of the original strided MAML vs strided MAML++. One can see that 2 out of 3 seeds with the original strided MAML seem to become unstable and erratic, whereas all 3 of the strided MAML++ models seem to consistently converge very fast, to much higher generalization accuracy without any stability issues.

Note: The reader's awareness of the definition of MAML in Section 1.1 is assumed in this Chapter.

3.2 Related Work

The work in this chapter builds directly on MAML [Finn et al., 2017a], showcasing various methods that can improve the model.

Furthermore, there are some similarities between this work and MetaSGD [Li et al., 2017b] where the authors meta-learned a learning rate magnitude and direction for each network parameter. Instead we learned a learning rate for each layer at each step; thus requiring to learn only ten learning rates compared to thousands for meta-SGD while producing similar generalization improvements.

Other work that attempts to improve aspects of MAML includes MAML with Warped Gradient Descent [Flennerhag et al., 2020] where the authors propose an alternative way for optimizing MAML that is computationally cheaper and reduces gradient degradation issues, as well as Implicit Gradients [Rajeswaran et al., 2019] where the authors introduce a new objective for MAML which allows further reduction of gradient degradation problems and cheaper optimization.

Furthermore, MAML is also related to Meta-Learner LSTM [Ravi and Larochelle, 2016] where the authors propose meta-learning an optimizer for quick adaptation of a base-model. Other related work has been covered in Section 2.2.10. Meta-Learner LSTM was of special interest here as it was one of the very first to have meta-learned a learner in the context of gradient-based meta-learning.

3.3 Model Agnostic Meta-Learning Problems

This chapter builds on the notation and concepts introduced in Section 1.1.

The simplicity, elegance and high performance of MAML make it a very powerful framework for meta-learning. However, MAML has also many issues that make it problematic to use. These are identified here.

Training Instability: Depending on the neural network architecture and the overall hyperparameter setup, MAML can be very unstable during training as illustrated in Figure 3.1. Optimizing the outer loop involves backpropagating derivatives through an unfolded inner loop consisting of the same network multiple times. Model depth alone could be cause for gradient degradation problems. However, the gradient issues are further compounded by the model architecture, which is a standard 4-layer convolutional network without skip-connections. The lack of skip-connections means that every gradient must be passed through each convolutional layer many times; effectively the gradients will be multiplied by the same sets of parameters multiple times. After multiple back-propagation passes, the large depth structure of the unfolded network and lack of skip connections can cause gradient explosions and diminishing gradient problems respectively.

Second Order Derivative Cost: Optimization through gradient update steps requires the computation of second order gradients which are very expensive to compute. The authors of MAML proposed using first-order approximations to speed up the process by a factor of three, however using these approximations can have a negative impact on the final generalization error. Further attempts at using first order methods have been attempted in Reptile [Nichol et al., 2018b] where the authors apply standard SGD on a base-model and then take a step from their initialization parameters towards the parameters of the base-model after N steps. The results of Reptile vary, in some cases exceeding MAML, and in others producing results inferior to MAML. Approaches to reduce computation time while not sacrificing generalization performance had not been proposed prior to the publication of the work described in this chapter.

Absence of Batch Normalization Statistic Accumulation: A further issue that affects the generalization performance is the way that batch normalization is applied in the experiments in the original MAML paper. Instead of accumulating running statistics, the statistics of the current batch were used for batch normalization. This results in batch normalization being less effective, since the biases learned have to accommodate for a variety of different means and standard deviations instead of a single mean and standard deviation. On the other hand, if batch normalization uses accumulated running statistics it will eventually converge to some global mean and standard deviation. This leaves only a single mean and standard deviation to learn biases for. Using running statistics instead of batch statistics, can greatly increase convergence speed, stability and generalization performance as the normalized features will result in smoother optimization landscape [Santurkar et al., 2018].

Shared (across step) Batch Normalization Bias: An additional problem with batch normalization in MAML stems from the fact that batch normalization biases are not updated in the inner-loop; instead the same biases are used throughout all iterations of base-models. Doing this implicitly assumes that all base-models are the same throughout the inner loop updates and hence have the same distribution of features passing through them. This is a false assumption to make, since, with each inner loop update, a new base-model is instantiated that is different enough from the previous one to be considered a new model from a bias estimation point of view. Thus learning a single set of biases for all iterations of the base-model can restrict the modeling capacity and generalization capability of the model.

Shared Inner Loop (across step and across parameter) Learning Rate: One issue that affects both generalization and convergence speed (in terms of training iterations) is the issue of using a shared learning rate for all parameters and all update-steps. Doing so introduces two major problems. Having a fixed learning rate requires doing multiple hyperparameter searches to find the correct learning rate for a specific dataset; this process can be very computationally costly, depending on how search is done.

The authors in Li et al. [2017b] propose to learn a learning rate and update direction for each parameter of the network. Doing so solves the issue of manually having to search for the right learning rate, and also allows individual parameters to have smaller or larger learning rates. However this approach brings its own problems. Learning a learning rate for each network parameter means increased computational effort and increased memory usage since the network contains between 40K and 50K parameters depending on the dimensionality of the data-points.

Fixed Outer Loop Learning Rate: In MAML the authors use Adam with a fixed learning rate to optimize the meta-objective. Annealing the learning rate using either step or cosine functions has proven crucial to achieving state of the art generalization performance in a multitude of settings [Loshchilov and Hutter, 2017, He et al., 2016, Larsson et al., 2016, Huang et al., 2017]. Thus, we theorize that using a static learning rate reduces MAML's generalization performance and might also be a reason for slower optimization. Furthermore, having a fixed learning rate might mean that one has to spend more (computational) time tuning the learning rate.

3.4 Stable, automated and improved MAML

In this chapter we propose methods for solving the issues with the MAML framework, described in Section 3.3. Each solution has a reference identical to the reference of the issue it is attempting to solve.

Gradient Instability $\rightarrow$ **Multi-Step Loss Optimization (MSL)**: MAML works by minimizing the target-set loss computed by the base-network after it has completed **all** of its inner-loop updates towards a support set task. Instead we propose minimizing the target-set loss computed by the base-network after **every** step towards a support set task. More specifically, we propose that the loss minimized is a weighted sum of the target-set losses after every support set loss update. More formally:

$$\theta = \theta - \beta \nabla_{\theta} \sum_{b=0}^{B} \sum_{i=0}^{N} v_i \mathcal{L}_{T_b}(f_{\theta_i^b}) \tag{3.1}$$

Where β is a learning rate, $\mathcal{L}_{T_b}(f_{\theta_i^b})$ denotes the target-set loss of task b when using the base-network weights after i steps towards minimizing the support set task and v_i denotes the importance weight of the target-set loss at step i, which is used to compute the weighted sum.

By using the *multi-step* loss proposed above we improve gradient propagation, since now the base-network weights receive gradients both directly (for the current step loss) and indirectly (from losses coming from subsequent steps) at every step. With the original methodology described in Section 1.1 the base-network weights at every step except the last one were optimized implicitly as a result of backpropagation, which caused many of the instability issues MAML had. However using the multi-step loss alleviates this issue as illustrated in Figure 3.1. Furthermore, we employ an annealed weighting for the per step losses. Initially all losses have equal contributions towards the loss, but as iterations increase, we decrease the contributions from earlier steps and slowly increase the contribution of later steps. The contributions from the first $N-1$ steps is linearly interpolated towards zero equally among all these steps, while step N importance is linearly interpolated to 100%. We arrived at this configuration after some preliminary experiments, that included keeping equal weighting across the full training, as well as fixed weighting that increased as the N of a step increased, as well as the reverse, decreasing with N. No specific temperature weightings were empirically evaluated. This is done to ensure that as training progresses the final step loss becomes more dominant in the overall training loss thus ensuring that the additional steps done by our model (which represent significant added capacity and computational overhead) are optimized in a way that is maximally used to reduce the inner loop loss. If the annealing is not used, we found that the final loss might be higher than with the original formulation.

Second Order Derivative Cost $\rightarrow$ Derivative-Order Annealing (DA): One way of making MAML more computationally efficient is reducing the number of inner-loop updates needed, which can be achieved with some of the methods described in subsequent sections of this report. However, in this paragraph, we propose a method that reduce the per-step computational overhead directly. The authors of MAML proposed the usage of first-order approximations of the gradient derivatives. However they applied the first-order approximation throughout the whole of the training phase. Instead, we propose to anneal the derivative-order as training progresses. More specifically, we propose to use first-order gradients for the first 50 epochs (chosen after preliminary experiments) of the training phase, and to then switch to second-order gradients for the remainder of the training phase. We empirically demonstrate that doing so greatly speeds up the first 50 epochs, while allowing the second-order training needed to achieve the strong generalization performance the second-order gradients provide to the model. An additional interesting observation is that derivative-order annealing experiments showed no incidents of exploding or diminishing gradients, contrary to second-order only experiments which were more unstable. Using first-order before starting to use second-order derivatives can be used as a strong *pretraining* method that learns parameters less likely to produce gradient explosion/diminishment issues.

Absence of Batch Normalization Statistic Accumulation $\rightarrow$ Per-Step Batch Normalization Running Statistics (BNRS): In the original implementation of MAML [Finn et al., 2017b] the authors used only the current batch statistics as the batch normalization statistics. This, we argue, caused a variety of undesirable effects described in Section 3.3. To alleviate the issues we propose using running batch statistics for batch normalization. A naive implementation of batch normalization in the context of MAML would require sharing running batch statistics across all update steps of the inner-loop fast-knowledge acquisition process. However doing so would cause the undesirable consequence that the statistics stored be shared across all inner loop updates of the network. This would cause optimization issues and potentially slow down or altogether halt optimization, due to the increasing complexity of learning parameters that can work across various updates of the network parameters. A better alternative would be to collect statistics in a per-step regime. To collect running statistics per-step, one needs to instantiate N (where N is the total number of inner-loop update steps) sets of running mean and running standard deviation for each batch normalization layer in the network and update the running statistics respectively with the steps being taken during the optimization. The per-step batch normalization methodology should speed up optimization of MAML whilst potentially improving generalization performance.

Shared (across step) Batch Normalization Bias $\rightarrow$ Per-Step Batch Normalization Weights and Biases (BNWB): In the MAML section the authors trained their model to learn a **single** set of biases for each layer. Doing so assumes that the distributions of features passing through the network are similar. However, this is a false assumption since the base-model is updated for a number of times, thus making the feature distributions increasingly dissimilar from each other. To fix this problem we propose learning a set of biases **per-step** within the inner-loop update process. Doing so, means that batch normalization will learn biases specific to the feature distributions seen at each set, which should increase convergence speed, stability and generalization performance.

Shared Inner Loop Learning Rate (across step and across parameter) $\rightarrow$ Learning Per-Layer Per-Step Learning Rates and Gradient Directions (LSLR): Previous work in [Li et al., 2017b] demonstrated that learning a learning rate and gradient direction for each parameter in the base-network improved the generalization performance of the system. However, that had the consequence of increased number of parameters and increased computational overhead. So instead, we propose, learning a learning rate and direction for each layer in the network as well as learning different learning rates for each adaptation of the base-network as it takes steps. Learning a learning rate and direction for each layer instead for each parameter should reduce memory and computation needed whilst providing additional flexibility in the update steps. Furthermore, for each learning rate learned, there will be N instances of that learning rate, one for each step to be taken. By doing this, the parameters are free to learn to decrease the learning rates at each step which may help alleviate overfitting. Experimentally, we found that the learned learning rates tend to converge across multiple initial seeds to very similar values for a given task, but tend to vary quite a bit across different tasks. One characteristic that remains constant across all tasks and seeds, is the fact that all early layers in the network receive near zero learning rate, while the last 2-3 layers receive very high learning rates. In a way, meta-learning has converged to the conclusion that the best way to few-shot learn is to use a pretrained embedding and fine tune only the last couple of layers, which has also been discovered and empirically supported in other works [Wang et al., 2019, Antoniou and Storkey, 2019b, Rusu et al., 2019]. Furthermore, in 5-shot tasks, the trend of the learning rates seems to be linearly decreasing, while in 1-shot tasks the trend seems to be a bit more variable, and can often have a linearly decreasing pattern. A full report on the learned learning rates, and deeper insights about them, can be found at Antoniou [2018].

Fixed Outer Loop Learning Rate $\rightarrow$ Cosine Annealing of Meta-Optimizer Learning Rate (CA): In MAML the authors use a static learning rate for the optimizer of the meta-model. Annealing the learning rate, either by using step-functions [He et al., 2016] or cosine functions [Loshchilov and Hutter, 2017] has proved vital in learning models with higher generalization power. The cosine annealing scheduling has been especially effective in producing state of the art results whilst removing the need for any hyper-parameter searching on the learning rate space. Thus, we propose applying the cosine annealing scheduling on the meta-model's optimizer (i.e. the *meta-optimizer*). Annealing the learning rate allows the model to fit the training set more effectively and as a result might produce higher generalization performance.

3.5 Datasets

The datasets used to evaluate our methods were the Omniglot ([Lake et al., 2015]) and Mini-Imagenet ([Vinyals et al., 2016, Ravi and Larochelle, 2016]) datasets. Each dataset is split into 3 sets, a training, validation and test set. The Omniglot dataset is composed of 1623 characters classes from various alphabets. There exist 20 instances of each class in the dataset. For Omniglot we shuffle all character classes and randomly select 1150 for the training set and from the remaining classes we use 50 for validation and 423 for testing. In most few-shot learning sections the first 1200 classes are used for training and the remaining for testing. However, having a small validation set to choose the best model is crucial, so we choose to use a small set of 50 classes as validation set. For each class we use all available 20 samples in the sets. Furthermore for the Omniglot dataset, data augmentation is used on the images in the form of rotations of 90 degree increments. Class samples that are rotated are considered new classes, e.g. a 180 degree rotated character C is considered a different class from a non rotated C, thus effectively having 1623 x 4 classes in total. However the rotated classes are generated dynamically after the character classes have been split into the sets such that rotated samples from a class reside in the same set (i.e. the training, validation or test set). The Mini-Imagenet dataset was proposed in [Ravi and Larochelle, 2016], it consists of 600 instances of 100 classes from the ImageNet dataset, scaled down to 84x84. We use the split proposed in [Ravi and Larochelle, 2016], which consists of 64 classes for training, 12 classes for validation and 24 classes for testing.

3.6 Experiments

To evaluate our methods we adopted a hierarchical hyperparameter search methodology. First we began with the baseline MAML experiments, which were ran on the 5/20-way and 1/5-shot settings on the Omniglot dataset and the 5-way 1/5-shot setting on the Mini-Imagenet dataset. Then we added each one of our 6 methodologies on top of the default MAML and ran experiments for each one separately. Once this stage was completed we combined the approaches that showed improvements in either generalization performance or convergence speed (both in terms of number of epochs and clock-time) and ran a final experiment to establish any potential gains from the combination of the techniques.

An experiment consisted of training for 150 epochs, each epoch consisting of 500 iterations. At the end of each epoch, we evaluated the performance of the model on the validation set. Upon completion of all epochs the best performing model on the validation set was applied on the test set, thus producing the final test performance of the model. An evaluation ran consisted of 500 iterations on their targeted set. A distinction between the training and evaluation tasks, was that the training tasks were generated dynamically continually without repeating previously sampled tasks, whilst the 500 evaluation tasks generated were identical across epochs. This practice ensured that the comparison between models was fair, from an evaluation set viewpoint. Every experiment was repeated for three independent runs.

The models were trained using the Adam optimizer with a learning rate of 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.99$. Furthermore, all Omniglot experiments used a task batch size of 16, whereas for the Mini-Imagenet experiments we used a task batch size of 4 and 2 for the 5-way 1-shot and 5-way 5-shot experiments respectively.

Omniglot 5-way Few-Shot Classification		
	Accuracy	
Omniglot	**1-shot**	**5-shot**
Siamese Nets	97.3%	98.4%
Matching Nets	98.1%	98.9%
Neural Statistician	98.1%	99.5%
Memory Mod.	98.4%	99.6%
Meta-SGD	99.53±0.26%	**99.93±0.09%**
Meta-Networks	98.95%	-
MAML (original)	98.70±0.4%	99.9±0.1%
MAML (local replication)	98.57%	99.82%
MAML++	**99.47%**	99.85%

Table 3.1: MAML++ Omniglot 5-way Results: *MAML++* indicates MAML + all the proposed fixes. We report our own base-lines on MAML to provide better relative intuition on how each method impacted the test accuracy of the model. We can see that MAML++ matches or improves on MAML across all cases. MAML++ also has performance very close to Meta-SGD which uses double the amount of parameters that MAML requires (about 40K extra parameters), whilst only using 522 extra parameters (one for each layer). Having less parameters also means smaller training and testing times.

3.7 Results

Our proposed methodologies are empirically shown to improve the original MAML framework. In Table 3.2 one can see how our proposed approach performs on Omniglot. Each proposed methodology can individually outperform MAML, however, the most notable improvements come from the learned per-step per-layer learning rates and the per-step batch normalization methodology. In the 5-way 1-shot tasks it achieves 99.47% and in the 20-way Omniglot tasks MAML++ achieves 97.76% and 99.33% in the 1-shot and 5-shot tasks respectively. MAML++ also showcases improved convergence speed in terms of training iterations required to reach the best validation performance. Furthermore, the multi-step loss optimization technique substantially improves the training stability of the model as illustrated in Figure 3.1. In Table 3.2 we also include the results of our own implementation of MAML, which reproduces all results except the 20-way 1-shot Omniglot case. Difficulty in replicating the specific result has also been noted before in [Jamal et al., 2018]. We base our conclusions on the relative performance between our own MAML implementation and the proposed methodologies.

Table 3.3 showcases MAML++ on Mini-Imagenet tasks, where MAML++ sets a new state of the art in both the 5-way 1-shot and 5-shot cases where the method achieves 52.40% and 67.15% respectively. More notably, MAML++ can achieve very strong 1-shot results of 51.05% with only a single inner loop step required. Not only is MAML++ cheaper due to the usage of derivative order annealing, but also because of the much reduced number of inner loop steps. Another notable observation is that MAML++ converges to its best generalization performance much faster (in terms of iterations required) when compared to MAML as shown in Figure 3.1.

Table 3.4 showcases how the computational expense of both MAML and MAML++ compare to each other as number of inner loop steps increases on MiniImagenet.. Despite the fact that the multi-step loss (MSL) modification makes MAML++ slower, the additional usage of gradient order annealing ensures that MAML++ remains overall cheaper in terms of wall-clock time, per update iteration.

3.8 Conclusion

In this chapter we delve deep into what makes or breaks the MAML framework and propose multiple ways to automate most of the hyperparameter searching required, improve the generalization error, stabilize and speed up MAML. The resulting approach, called MAML++ sets a new state of the art across all few-shot tasks, across Omniglot and Mini-Imagenet. The results of the approach indicate that learning per-step learning rates, batch normalization parameters and optimizing on per-step target losses appears to be key for fast, highly automatic and strongly generalizable few-shot learning.

After publication, this work has received a lot of attention from the community, partly as an entryway into understanding MAML and meta-learning, through the paper, as well as an entry-way into building meta-learning systems, using the MAML++

open-source implementation[2] which was specifically designed with high readability, modarity and extensibility in mind. In fact, in our knowledge, it was the very first implementation of MAML which succeeded in modularizing inner-loop adaptable layers, which previously had to be fully hardcoded, hence making the design of inner loop architectures of all types much more efficient, and intuitive.

[2]The open source implementation can be found at `https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch`

Omniglot 20-way Few-Shot Classification		
	Accuracy	
Approach	**1-shot**	**5-shot**
Siamese Nets	88.2%	97.0%
Matching Nets	93.8%	98.5%
Neural Statistician	93.2%	98.1%
Memory Mod.	95.0%	98.6%
Meta-SGD	95.93±0.38%	98.97±0.19%
Meta-Networks	97.00%	–
MAML (original)	95.8±0.3%	98.9±0.2%
MAML (local replication)	91.27±1.07%	98.78%
MAML++	**97.65±0.05%**	**99.33±0.03%**
MAML + MSL	91.53±0.69%	-
MAML + LSLR	95.77±0.38%	-
MAML + BNWB + BNRS	95.35±0.23%	-
MAML + CA	93.03±0.44%	-
MAML + DA	92.3±0.55%	-

Table 3.2: MAML++ Omniglot 20-way Few-Shot Results: Our reproduction of MAML appears to be replicating all the results except the 20-way 1-shot results. Other authors have come across this problem as well [Jamal et al., 2018]. We report our own baselines to provide better relative intuition on how each method impacted the test accuracy of the model. We showcase how our proposed improvements individually improve on the MAML performance. Our method improves on the existing state of the art. The ablation study showcases how each proposed modification affects the performance independently, as well as when all modifications are combined. **Note**: **MSL**: Multi Step Loss, **LSLR**: per Layer, per Step Learning Rate, **BNWB**: learnable per step per layer Batch Normalization Weights and Biases, **BNRS**: per layer, per step, Batch Normalization Running Statistics, **CA**: Cosine Annealing, **DA**: Derivative order Annealing

Mini-Imagenet 5-way Few-Shot Classification			
	Inner Steps	**Accuracy**	
Mini-Imagenet		**1-shot**	**5-shot**
Matching Nets	-	43.56%	55.31%
Meta-SGD	1	50.47±1.87%	64.03±0.94%
Meta-Networks	-	49.21%	-
MAML (original section)	5	48.70±1.84%	63.11±0.92%
MAML (local reproduction)	5	48.64%[1]	64.78%
MAML++	1	51.05±0.31%	-
MAML++	2	51.49±0.25%	-
MAML++	3	51.11±0.11%	-
MAML++	4	51.65±0.34%	-
MAML++	5	**52.40±1.13%**	**67.15±0.26%**

Table 3.3: MAML++ Mini-Imagenet Results. *MAML++* indicates MAML + all the proposed fixes. Our reproduction of MAML appears to be replicating all the results of the original. Our approach sets a new state of the art across all tasks. It is also worth noting, that our approach, with only 1 inner loop step can already exceed all other methods. Additional steps allow for even better performance.

Inner Loop Steps	**1**	**2**	**3**	**4**	**5**
MAML ++ (ms/iter)	275.319	433.8172	579.314	786.3278	947.0376
MAML (ms/iter)	294.373	475.1218	658.4436	859.1158	1028.1656

Table 3.4: Mini-Imagenet Training Iteration Timing table. In this table, one can see per training iteration wall-clock timings for MAML vs MAML++. We provide timings for variants of the model spanning 1 to 5 inner loop steps. It can be observed that MAML++ needs less time per training iteration, even though it needs more parameters and has more computations needed. We can see that more steps require more computation in return for better generalization performance as evidenced in Table 3.3.

Chapter 4

Learning to Learn via Self-Critique: Meta-Learning Unsupervised Loss Functions Suitable for Supervised Learning Tasks

4.1 Introduction

Humans can learn from a few data-points and generalize very well, but also have the ability to adapt in real-time to information from an incoming task. Given two training images for a two class problem, one with a cat on a white sofa, and one with a dog next to a car, one can come up with two hypotheses as to what each class describes in each case. A test image that presents a dog with a cat would be ambiguous. However, having other unlabelled test images with cats in other contexts and cars in other contexts enables these cases to be disambiguated, and it is possible to learn to focus on features that help this separation. We wish to incorporate this ability to adapt into a meta-learning context.

Few-shot learning is a learning paradigm where only a handful of samples are available to learn from. It is a setting where deep learning methods previously demonstrated weak generalization performance. In recent years, by framing the problem of few-shot learning as a meta-learning problem [Vinyals et al., 2016], we have observed an advent of meta-learning methods that have demonstrated unprecedented performance on a number of few-shot learning benchmarks [Snell et al., 2017a, Finn et al., 2017b, Rusu et al., 2018].

Most few-shot meta-learning methods have focused on either learning static [Finn et al., 2017b, Antoniou et al., 2018a, Li et al., 2017b] or dynamic parameter initializations [Rusu et al., 2018], learning rate schedulers [Antoniou et al., 2018a], embedding

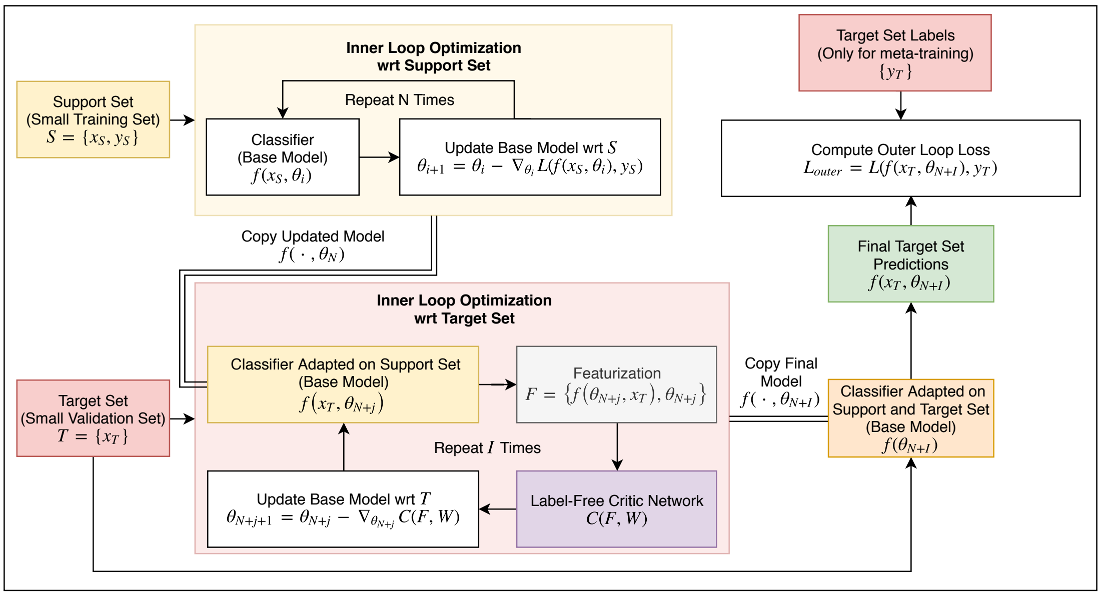

Inner Loop Optimization wrt Support Set
Support Set (Small Training Set) $S = \{x_S, y_S\}$
Repeat N Times
Classifier (Base Model) $f(x_S, \theta_i)$
Update Base Model wrt S $\theta_{i+1} = \theta_i - \nabla_{\theta_i} L(f(x_S, \theta_i), y_S)$
Target Set Labels (Only for meta-training) $\{y_T\}$
Compute Outer Loop Loss $L_{outer} = L(f(x_T, \theta_{N+I}), y_T)$
Copy Updated Model $f(\cdot, \theta_N)$
Final Target Set Predictions $f(x_T, \theta_{N+I})$
Inner Loop Optimization wrt Target Set
Classifier Adapted on Support Set (Base Model) $f(x_T, \theta_{N+j})$
Featurization $F = \{f(\theta_{N+j}, x_T), \theta_{N+j}\}$
Copy Final Model $f(\cdot, \theta_{N+I})$
Classifier Adapted on Support and Target Set (Base Model) $f(\theta_{N+I})$
Target Set (Small Validation Set) $T = \{x_T\}$
Repeat I Times
Update Base Model wrt T $\theta_{N+j+1} = \theta_{N+j} - \nabla_{\theta_{N+j}} C(F, W)$
Label-Free Critic Network $C(F, W)$

Figure 4.1: Proposed Method. Starting from the top-left, task-specific knowledge from the support-set is used to train the base model, updating θ_0 to θ_N. At this point, standard meta-learning methods return predictions from this learnt model to complete their inference. Instead, we use an unsupervised loss from a critic network C applied to the unlabelled target set to make further updates. To do this we collect a set of features F that summarise the model applied to the target set T; these features are sent to the critic C, a neural network with parameters W. Using the loss from this critic network, and the model with parameters θ_N, we make further updates to get θ_{N+I}. We use the predictions from this model as our predictions for the target set. During training, an *outer-loop* loss comparing target-set predictions to target set labels is used to update the initial parameters θ_0 and the critic model parameters W.

functions [Vinyals et al., 2016, Snell et al., 2017a], optimizers [Ravi and Larochelle, 2016] and other internal components. However all of these methods explore learning from the labelled support-set, whereas learning (at inference time) from the unlabelled target-set has remained unexplored.[1] In situations where the labelled training set is small, and the unlabelled target set is very large, such a capability could provide a lot of benefit.

In this chapter, we propose a mechanism we call *Self-Critique and Adapt* or SCA that enables meta-learning-based few-shot systems to learn not only from the support-set input-output pairs, but also from the target-set inputs, by learning a label-free loss function, parameterized as a neural network. Doing so grants our models the ability to learn from the target-set input data-points, simply by computing a loss, conditioned on base-model predictions of the target-set. The label-free loss can be used to compute gradients with respect to the model, and the gradients can then be used to update the base-model at inference time, to improve generalization performance. Furthermore, the proposed method can be added on top of any modern meta-learning method, including both methods that utilize gradient updates on the support set, such as MAML [Finn et al., 2017b], as well as ones that do not use gradient-based updates on the support set, such as Matching Networks [Vinyals et al., 2016].

Self-Critique and Adapt is a *transductive learning* approach [Vapnik, 2006]. Transductive learning uses training data and test input data-points to learn a model that is **specifically** tuned to produce predictions for the given test-set. Transductive learning benefits from *unsupervised information* from the test example points, and *specification* by knowing where we need to focus model capability. In stark contrast, *inductive learning*, can be defined as a learning paradigm where given training input-output pairs, a model is learned consisting of **general** rules, that can then be used on any test-set without refinement to produce predictions. Given that, in a meta-learning context, additional learning needs to be done for each new setting anyway, and given the importance of making the most of every piece of information, transductive learning is a natural learning paradigm for the few-shot learning setting.

Transductive learning can be considered a form of semi-supervised learning. Standard semi-supervised learning is purely inductive, as it basically uses two training sets, one labelled and one unlabelled to derive general rules from a previously unseen test set, whereas transductive learning receives a labelled training set, and an unlabelled

[1]Existing techniques like MAML [Finn et al., 2017b] utilize target-set information by computing means and standard deviations for the batch normalization layers within their base models. However, we don't consider that as explicit learning, but instead, as a minimal adaptation routine.

test set, and the learner is allowed to use the unlabelled test set to tune the model such that it can do well on those specific data points. Transduction is purely contextual learning, and can, therefore, find a solution for a given context without having to first derive general rules for a much larger space of problems, rather than what is needed.

We evaluate the proposed method on the established few-shot learning benchmarks of Mini-ImageNet [Ravi and Larochelle, 2016] and Caltech-UCSD Birds 200 (CUB) [Chen et al., 2019]. The evaluation results indicate that our method substantially boosts the performance of two separate instances of the MAML++ [Antoniou et al., 2018a] framework, setting a new state-of-the-art performance for all tasks in both benchmarks.

This chapter's contributions are:

1. An approach that gives state-of-the-art performance in the Mini-Imagenet and Caltech-UCSD Birds 200 (CUB) benchmark tasks by using both support and target set information through a transductive approach.

2. The ability to learn to learn a flexible parameterized loss function appropriate for a supervised problem but defined on unlabelled data; this loss function can be used to enhance training on semi-supervised data.

3. A set of ablation studies on different conditioning features for the critic network, revealing which features are most useful to the few-shot learning benchmarks.

4.2 Related Work

Much of the related work can be found in Section 2.2.10; however, there are a few methods that are of special note. The first of such methods, Meta-Learner LSTM [Ravi and Larochelle, 2016] that learned an optimizer parameterized as a neural network, is relevant as it was one of the first to attempt gradient-based meta-learning. Model Agnostic Meta-Learning (MAML) [Finn et al., 2017b] and its enhanced versions *Meta-SGD* [Li et al., 2017b] and *MAML++* [Antoniou et al., 2018a], are of high relevance as this work is built on top of MAML++.

Furthermore, Relational Networks [Santoro et al., 2017] were also used in this work to build task-level embeddings for our base-model.

Work such as LEO [Rusu et al., 2018] is related as well in that both pieces of work attempt to learn meta-learning models that rely on combined representation types. LEO

Algorithm 1 SCA Algorithm combined with MAML

```
1: Required : Base model function f and initialisation parameters θ, critic network function C and parameters W, a batch of tasks $\{\mathbf{S^B} = \{x_S^B, y_S^B\}, \mathbf{T^B} = \{x_T^B, y_T^B\}\}$ (where B is the number of tasks in our batch) and learning rates $\alpha, \beta, \gamma$
2: $L_{outer} = 0$
3: for b in range(B) do
4:     $\theta_0 = \theta$    ▷ Reset $\theta_0$ to the learned initialization parameters
5:     for i in range(N) do ▷ N indicates total number of inner loop steps wrt support set
6:         $\theta_{i+1} = \theta_i - \alpha\nabla_{\theta_i} L(f(x_S^b, \theta_i), y_S^b)$   (4.1)
               ▷ Inner loop optimization wrt support set
7:     for j in range(I) do ▷ I indicates total number of inner loop steps wrt target set
8:         $F = \{f(x_T^b, \theta_{N+j}), \theta_{N+j}, g(x_S, x_n)\}$   (4.2)
               ▷ Critic feature-set collection
9:         $\theta_{N+j+1} = \theta_{N+j} - \gamma\nabla_{\theta_{N+j}} C(F, W)$   (4.3)
               ▷ Inner loop optimization wrt target set
10:    $L_{outer} = L_{outer} + L(f(x_T^b, \theta_{N+I}), y_T^b)$
11: $\theta = \theta - \beta\nabla_\theta L_{outer}$   (4.4)
               ▷ Joint outer loop optimization of θ
12: $W = W - \beta\nabla_W L_{outer}$   (4.5)
               ▷ Joint outer loop optimization of W
```

learned a sample and task embedding and a weight generator while our work learned a sample and task embedding, and an unsupervised loss function.

In the Reinforcement Learning domain, meta-learning has been used to learn an unsupervised loss function for sample-efficient adaptation in [Yu, 2018]. Their work differs from ours, in that our model adapts a meta-learned initialization by both labelled and unlabelled examples; we do use a learnt unsupervised model, but the combination is critical. Their work is purely unsupervised in the inner loop, using RL as the outer loop, whereas ours consists of both supervised and unsupervised inner loops and a supervised outer loop. Furthermore, in [Houthooft et al., 2018b], the authors propose a method that learns a state and reward conditional loss function to train a policy network, using RL inner phase and an evolutionary algorithm outer loop. Our work, instead targets a different problem, that is few-shot learning, using a supervised outer loop, and a transductive inner loop (composed by supervised and unsupervised inner loop phases). Furthermore, in [Sung et al., 2017b] the authors propose learning a supervised loss function for few-shot learning, as well as a state and reward conditional loss function for training RL agents.

Semi-supervised learning via learning unsupervised loss functions were also proposed in [Rinu Boney, 2018, Ren et al., 2018]. Transductive learning for the few-shot learning setting was concurrently attempted in [Liu et al., 2018], where the authors effectively learn to generate new labels for a given task, which can be considered a form of auxiliary task learning. Their work differs from ours in that we learn to transduce using a learned unsupervised loss function whereas they instead learn to generate labels for the test set.

4.3 Self-Critique and Adapt

For a model to learn and adapt in a setting where only input data-points are available (e.g. on given task's few-shot target-set), one needs a label-free loss function. For example, many unsupervised learning approaches try to maximize the generative probability, and hence use a negative log-likelihood (or bound thereof) as a loss function. In general though, much of the generative model will be task-irrelevant. In the context of a particular set of tasks there are likely to be more appropriate, specialised choices for a loss function.

Manually inventing such a loss function is challenging, often only yielding loss functions that might work in one setting but not in another. Understanding the full implications of a choice of loss-function is not easy. Instead, we propose a Self-Critique and Adapt approach which *meta-learns* a loss function for a particular set of tasks. It does this by framing the problem using the set-to-set few-shot learning framework and using end-to-end differentiable gradient-based meta-learning as our learning framework.

SCA is model-agnostic, and can be applied on top of **any** end-to-end differentiable, gradient-based, meta-learning method that uses the inner-loop optimization process to acquire task-specific information. Many such approaches [Ravi and Larochelle, 2016, Finn et al., 2017b, Li et al., 2017b, Antoniou et al., 2018a, Finn et al., 2018, Qiao et al., 2018b, Rusu et al., 2018, Grant et al., 2018] are currently competing for state-of-the-art in the few-shot learning landscape.

Self-Critique and Adapt, summarised in Figure 4.1, takes a base-model, updates it with respect to the support-set with an existing gradient-based meta-learning method (e.g. MAML [Finn et al., 2017b], MAML++ [Antoniou et al., 2018a] or LEO [Rusu et al., 2018]), and then infers predictions for the target-set. Once the predictions have been inferred, they are concatenated along with other base-model related information (e.g. model parameters, a task embedding etc.), and are then passed to a learnable *critic loss network*, the output of which should be interpreted as a loss value for the given inputs. This critic network computes and returns a loss with respect to the target-set. The base-model is then updated with any stochastic gradient optimization method (such as SGD) with respect to this critic loss; updates can be done a number of times if necessary. This *inner-loop* optimization produces a predictive model specific to the support/target set information.

The inner loop process is used directly at inference time for the task at hand. However, as in other meta-learning settings, we optimize the inner loop using a collection of training tasks (these are different from the test tasks as explained in Section 4.7). The quality of the inner-loop learned predictive model is assessed using ground truth labels from the training tasks. The *outer loop* then optimizes the initial parameters and the critic loss to maximize the quality of the inner loop predictions. As with other meta-learning methods, the differentiability of the whole inner loop ensures this outer-loop can be learnt with gradient based methods.

In this chapter we use MAML++ as the base method. We denote the model, which is parameterized as a neural network, by $f(\cdot,\theta)$, with parameters θ, and the critic loss $C(\cdot,W)$, also a neural network with parameters W. We want to learn good parameters θ and W, such that when the model f is optimized a number of steps N with respect to the loss L on the support-set $S_b = \{x_S, y_S\}$, and then additionally another I steps towards the target-set $T_b = \{x_T\}$, using the critic loss C, it can achieve good generalization performance on the target-set. Here, b is the index of a particular task in a batch of tasks. The full algorithm is described in Algorithm 1.

Equation 4.2 in Algorithm 1 defines a potential set of conditioning features F that summarise the base-model and its behaviour. These features are what the unsupervised critic loss C can use to tune the target set updates. Amongst these possible features, $f(\theta_N, x_T)$ are the predictions of the base-model f, using parameters θ_N (that is parameters updated for N steps on the support-set loss) and $g(x_S, x_n)$ is a task embedding, parameterized as a neural network, which is conditional on the support and target input data-points.

4.4 Example: A Prediction-Conditional Critic Network

Model Inference and Training: As explained in Sections 4.3 and 4.5 our critic network can be given access to a wide variety of conditioning information. Here, we describe a specific variant, which uses the predictions of the base-model on the target-set as the critic network's conditioning information; this simple form enables visualisation of what the critic loss returns.

In this example, SCA is applied directly to MAML++. Given a support set, S and target set T for a task, and initial model $f(\cdot, \theta_0)$, five SGD steps are taken to obtain the updated model $f(\cdot, \theta_5)$ using the support set's loss using Equation 4.1. The updated model is then applied to the target set to produce target-set predictions $f(x_T, \theta_5)$; these predictions are passed to the critic network, which returns a loss-value for each element of the target set. Aggregated, these loss-values form the critic loss. The base model $f(\cdot, \theta_5)$ is then updated for 1 step using the gradients with respect to the critic loss using Equation 4.3 to obtain the final model $f(\cdot, \theta_6)$. We use gradients of a whole batch of target set images to update the base model since this produces better performance. The results in the chapter are from experiments where we used target sets of size 75 to learn the loss function. However one could instead randomly choose the batch size of the target set to train a loss function that generalizes on a wide range of batch sizes. At this stage our final model has learned from both the support and target sets and is ready to produce the final predictions of the target set. These are produced, and, at training time, evaluated against the ground truth. This final error is now the signal for updating the whole process: the choice of initial θ_0 and the critic loss update. The gradients for both the base and critic model parameters can be computed using backpropagation as every module of the system is differentiable.

Interrogating the Learned Critic Loss:

We investigate how the predictions of the base-model change after updates with respect to the critic loss. We generate target-set predictions using the base-model $f(\cdot, \theta_5)$ and pass those predictions to the learnt critic network to obtain a loss value for each individual prediction. Five steps of SGD are performed with respect to the the individual critic loss. The results are shown in Figure 4.2.

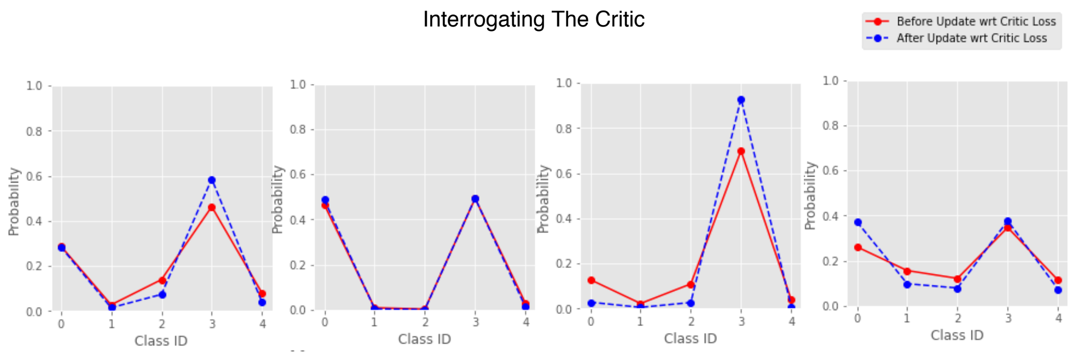


Figure 4.2: The target-set predictions of the base-model before (red lines) and after (green lines) it has been updated wrt the learned critic loss. Starting from the left, the first chart showcases an instance where the probabilities of two classes dominate the rest, with one class having approximately 20% higher probability than the other. The critic network proposes that the dominant class with the lower probability be left as is, whilst the higher probability class be increased. In the second chart, we present an example where two dominant classes have equal probabilities, and are thus considered optimal and left unchanged. In the third case, we present an example where a single class dominates all others. The critic network proposes that the probability of that class is pushed to almost the maximum possible value. Finally, in the fourth case we present a very interesting case where the critic, faced with two dominant classes, which had approximately 10% difference in their probability, proposed that the two classes have their probabilities matched to improve the resulting generalization performance. Note how some top classes change, e.g. in the rightmost chart. The effect of these changes is not in the class-labels alone but also allows better model initialization: training a critic on a pretrained initialization does not provide as much benefit. Even though the differences appear small, this *risk minimizer* function produces significant generalization improvements.

4.5 Choosing Conditioning Information for the Critic Network

We find that the performance of the critic network C, is unsurprisingly dependent on the quality of critic features it is given access to. In this section, we will outline a set of critic features we used in experiments, from simplest to most complicated; we also discuss the intuition behind their selection. To evaluate the usefulness of each set of critic features we use these in ablation studies in Section 4.7 and summarize the results in Tables 4.1 and 4.2.

Base-Model Predictions: The predictions on the base model indicate the certainty about any input; this can be a powerful driver for an unsupervised loss. As in Section 4.4, given the model f, the support-set updated weights θ_N and some target-set data-points x_T, we can generate predictions $f(x_T, \theta_N)$. These predictions can be passed to our critic model C to compute the loss function.

Base-model Parameters: Another key source of information about our base-model is the model parameters. In this context, the inner loop optimized parameters θ_N are passed to our loss network. For example this might enable it to learn *penalty* terms, similar to manually invented penalties such as *L1* and *L2*.

Task Embeddings: Giving the critic model direct access to task information can enable model assessment in the context of a particular task; we find this further improves performance as empirically observed in Tables 4.1 and 4.2. To generate a task embedding, we learn an embedding function g, parameterized as a neural network such as a DenseNet.[2] The embedding function g receives the support-set images and returns a batch of embedding vectors. Taking the mean of the vectors can serve as a strong task embedding. However, we found that using a relational network to relate all embeddings with each other and then taking the sum of those embeddings produced superior results. Similar embedding functions have previously been used in [Rusu et al., 2018].

4.6 Baselines

The proposed model is, by virtue of its design, applicable to any few-shot meta-learning system that follows the set-to-set [Vinyals et al., 2016] few-shot learning set-

[2]Here we use a DenseNet, with growth rate 8, and 8 blocks per stage, for a total of 49 layers. The DenseNet is reqularised using dropout after each block (with drop probability of 0.5) and weight decay (with decay rate 2e-05).

ting. Thus, to investigate its performance, we require baseline meta-learning systems which inner-loop optimize their parameters only on the support-set. We chose the MAML++ system, for it's simplicity and strong generalization performance.

To thoroughly evaluate our method, we experimented using two separate instances of the MAML++ framework, each differing only in the neural network architecture serving as its backbone.

Low-End MAML++: The backbone of our low-end baseline model follows the architecture proposed in [Antoniou et al., 2018a], which consists of a cascade of 4 convolutional layers, each followed by a batch normalization layer and a ReLU activation function. We optimize the entirety of the backbone during the inner loop optimization process. The low-end baseline is chosen to be identical to an existing model (MAML++), such that we could: 1. Confirm that our implementation replicates the results of the original method (i.e. makes sure that our framework does not over/under perform relatively to an existing method, and thus reduces the probability that any improvements in our results are there due to a bug in the data provider or learning framework) and 2. Investigate how our proposed method performs when added adhoc to an existing, non-tuned, architecture, therefore showcasing performance unbiased wrt architectures.

High-End MAML++: Methods that provide significant performance improvements on low-capacity models, often fail to provide the same level of improvement on high-capacity models and vice versa. Furthermore, meta-learning methods, have been demonstrated to be very sensitive to neural network architecture changes. Thus, to evaluate both the consistency and sensitivity of our method, we designed a high (generalization) performance MAML++ backbone. It uses a shallow, yet wide DenseNet architecture, with growth-rate 64, utilizing 2 dense-stages each consisting of two bottleneck blocks as described in [Huang et al., 2017] and one transition layer. Before each bottleneck block, we apply squeeze-excite style convolutional attention as described in [Hu et al., 2018], to further regularize our model. To improve the training speed and generalization performance, we restrict the network components optimized during the inner loop optimization process. In more detail, we share a deep static embedding representing the majourity of the learnable neural network across the inner loop steps, and only optimize within the inner loop the penultimate convolutional layer (including the squeeze excite attentional block preceding it) as well as the final linear layer. An efficient way of sharing components across steps, is to treat them as a *feature* embedding, whose features are passed to the components that will be inner loop optimized. Recent work [Rusu et al., 2018, Qiao et al., 2018b] followed a similar approach. Motivations behind these design choices can be found in the Section 1. 4.6.

Critic Network Architecture: The critic network consists of two majour components. First, a selection of conditioning features as described in Section 4.5, which are then reshaped into a batch of one-dimensional features vectors and concatenated on the feature dimension. Second, an *Information Integration* network, which consists of a sequence of five one-dimensional dilated convolutions with kernel size 2, and 8 kernels per convolutional layer. Further, we employ an exponentially increasing dilation policy where a given convolutional layer has dilation $d = 2^i$ where i is the index of the convolutional layer in the network, starting from 0 for the first layer and increasing up to 4 for the fifth layer. We use Dense-Net style connectivity for our convolutional layers, more specifically, the inputs for a given layer consist of a concatenation of the output features of all preceding convolutional layers. Finally we apply a sequence of two fully connected layers using ReLU non-linearities before the final linear layer which outputs the loss value.

High-End Backbone Design Motivations: The motivations behind each of the design choices can be found below.

1. Using DenseNet as the backbone, which decreases probability of gradient degradation problems and by allowing feature-reuse across all blocksm improves parameter/training efficiency. MAML is highly vulnerable to gradient degradation issues, and thus ensuring that our backbone decreases probability of such issues is of vital importance.

2. Using a shallow, yet wide backbone: Previous works [Qiao et al., 2018b, Rusu et al., 2018] have demonstrated that using features from the 20th layer of a pretrained ResNet produces superior generalization performance. The authors made the case that using features from shallower parts of the network decreases the probability that the features are too class-specific, and thus allow for better generalization on previously unseen classes. In both [Qiao et al., 2018b, Rusu et al., 2018] the authors did not train their meta-learning system end-to-end, and instead trained the feature backbone and the meta-learning components separately. However, in preliminary experiments we found that ResNet and DenseNet backbones tend to overfit very heavily, and in pursuit of a high-generalization end-to-end trainable meta-learning system, we experimented with explicit reduction of the effective input region of the layers in a backbone. Doing so, ensures that

features learned will be local. We found that keeping the effective input region of the deepest layer to approximately 15x15/20x20 produced the best results for both Mini-ImageNet and CUB. Furthermore, we found that widening the network produced additional generalization improvements. We theorize that this is because of a higher probability for a randomly initialized feature to lie in just the right subspace to produce a highly generalizable feature once optimized.

3. Using bottleneck blocks, preceded by squeeze-excite-style [Hu et al., 2018] convolutional attention: We empirically found that this improves generalization performance.

4. Inner-Loop optimize only the last squeeze excite linear layer, as well as the last convolutional layer and the final linear layer, whilst sharing the rest of the backbone across the inner loop steps. This design choice was hinted by the learned per-layer, per-step learning rates learned by MAML++ on the low-end baseline. More specifically, the learned learning rates where close to zero, for all layers, in all steps, except the last convolutional and last linear layers. Thus, we attempted to train a MAML++ instance where only those two layers where optimized in the inner loop, while the rest of the layers where shared across steps. In doing so, we found that doing so makes no difference to generalization performance, whilst increasing the training and inference speeds by at least 15 times.

4.7 Experiments

To evaluate the proposed methods we first establish baselines on both the low-end and high-end variants of MAML++ on the Mini-ImageNet and Caltech-UCSD Birds 200 (CUB) 5-way 1/5-shot tasks. Then, we add the proposed critic network, which enables the adaptation of the base-model on a given target-set. To investigate how the selection of conditional information we provide to the critic affect its performance we conduct ablation studies. Due to the large number of combinations possible, we adopt a hierarchical experimentation style, where we first ran experiments with a single source of conditional information, and then tried combinations of the best performing methods. Finally, we ran experiments where we combined every proposed conditioning method to the critic, as well as experiments where we combined every conditional method except one. Tables 4.1 and 4.2 show ablation and comparative studies on Mini-Imagenet and CUB respectively.

4.8 Results

The results of our experiments showcased that our proposed method was able to substantially improve the performance of both our baselines across all Mini-ImageNet and CUB tasks.

More specifically, the original MAML++ architecture (i.e. Low-End MAML++), yielded superior performance when we provided additional conditional information such as the task embedding and the model parameters. The only source of information that actually decreased performance were the base-model's parameters themselves. Furthermore, it appears that a very straight-forward strategy that worked the best, was to simply combine all proposed conditioning sources and pass them to the critic network.

However, in the case of the High-End MAML++ architecture, performance improvements were substantial when using the predictions and the task embedding as the conditioning information. Contrary to the results on MAML++, providing the critic with the matching network and relational comparator features (in addition to the prediction and task embedding features) did not produce additional performance gains.

Model	**Test Accuracy**			
	Mini-Imagenet		**CUB**	
	1-shot	5-shot	1-shot	5-shot
MAML++ (Low-End)	52.15±0.26%	68.32±0.44%	62.19±0.53%	76.08±0.51%
MAML++ (Low-End) with SCA (preds)	52.52±1.13%	70.84±0.34%	66.13±0.97%	77.62±0.77%
MAML++ (Low-End) with SCA (preds, params)	52.68±0.93%	69.83±1.18%	-	-
MAML++ (Low-End) with SCA (preds, task-embedding)	**54.84±1.24%**	70.95±0.17%	65.56±0.48%	77.69±0.47%
MAML++ (Low-End) with SCA (preds, task-embedding, params)	54.24±0.99%	**71.85±0.53%**	-	-
MAML++ (High-End)	58.37±0.27%	75.50±0.19%	67.48±1.44%	83.80±0.35%
MAML++ (High-End) with SCA (preds)	**62.86±0.70%**	77.07±0.19%	70.33±0.78%	85.47±0.40%
MAML++ (High-End) with SCA (preds, task-embedding)	62.29±0.38%	**77.64±0.40%**	**70.46±1.18%**	**85.63±0.66%**

Table 4.1: SCA Ablation Studies on Mini-ImageNet and CUB: All variants utilizing the proposed SCA method perform substantially better than the non-SCA baseline variant. Interestingly, the best type of critic conditioning features varies depending on the backbone architecture. Based on our experiments, the best critic conditioning features for the Low-End MAML++ is the combination of predictions, task-embedding and network parameters, whereas on High-End MAML++, using just the target-set predictions appears to be enough to obtain the highest performance observed in our experiments.

Model	**Test Accuracy**			
	Mini-ImageNet		**CUB**	
	1-shot	5-shot	1-shot	5-shot
Matching networks [Vinyals et al., 2016]	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$	$61.16 \pm 0.89\%$	$72.86 \pm 0.70\%$
Meta-learner LSTM [Ravi and Larochelle, 2016]	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$	-	-
MAML [Finn et al., 2017b]	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$	$55.92 \pm 0.95\%$	$72.09 \pm 0.76\%$
LLAMA [Grant et al., 2018]	$49.40 \pm 1.83\%$	-	-	-
REPTILE [Nichol et al., 2018a]	$49.97 \pm 0.32\%$	$65.99 \pm 0.58\%$	-	-
PLATIPUS [Finn et al., 2018]	$50.13 \pm 1.86\%$	-	-	-
Meta-SGD (our features)	$54.24 \pm 0.03\%$	$70.86 \pm 0.04\%$	-	-
SNAIL [Mishra et al., 2017]	$55.71 \pm 0.99\%$	$68.88 \pm 0.92\%$	-	-
[Gidaris and Komodakis, 2018]	$56.20 \pm 0.86\%$	$73.00 \pm 0.64\%$	-	-
[Munkhdalai and Yu, 2017a]	$57.10 \pm 0.70\%$	$70.04 \pm 0.63\%$	-	-
TADAM [Oreshkin et al., 2018]	$58.50 \pm 0.30\%$	$76.70 \pm 0.30\%$	-	-
[Qiao et al., 2018b]	$59.60 \pm 0.41\%$	$73.74 \pm 0.19\%$	-	-
LEO [Rusu et al., 2018]	$61.76 \pm 0.08\%$	$77.59 \pm 0.12\%$	-	-
Baseline [Chen et al., 2019]	-	-	$47.12 \pm 0.74\%$	$64.16 \pm 0.71\%$
Baseline ++ [Chen et al., 2019]	-	-	$60.53 \pm 0.83\%$	$79.34 \pm 0.61\%$
MAML (Local Replication)	$48.25 \pm 0.62\%$	$64.39 \pm 0.31\%$	-	-
MAML++ (Low-End)	$52.15 \pm 0.26\%$	$68.32 \pm 0.44\%$	$62.19 \pm 0.53\%$	$76.08 \pm 0.51\%$
MAML++ (Low-End) + SCA	$54.84 \pm 0.99\%$	$71.85 \pm 0.53\%$	$66.13 \pm 0.97\%$	$77.62 \pm 0.77\%$
MAML++ (High-End)	$58.37 \pm 0.27\%$	$75.50 \pm 0.19\%$	$67.48 \pm 1.44\%$	$83.80 \pm 0.35\%$
MAML++ (High-End) + SCA	$\mathbf{62.86 \pm 0.79\%}$	$\mathbf{77.64 \pm 0.40\%}$	$\mathbf{70.46 \pm 1.18\%}$	$\mathbf{85.63 \pm 0.66\%}$

Table 4.2: Comparative Results on Mini-ImageNet and CUB: The proposed method appears to improve the baseline model by over 4 percentage points, allowing it to set a new state-of-the-art result on both the 1/5-way Mini-ImageNet tasks.

4.9 Conclusion

In this chapter we propose adapting few-shot models not only on a given support-set, but also on a given target-set by learning a label-free critic model. In our experiments we found that such a critic network can, in fact, improve the performance of two instances of the well established gradient-based meta-learning method MAML. We found that some of the most useful conditional information for the critic model were the base-model's predictions, a relational task embedding and a relational support-target-set network. The performance achieved by the High-End MAML++ with SCA is the current best across all SOTA results. The fact that a model can learn to update itself with respect to an incoming batch of data-points is an intriquing one. Perhaps deep learning models with the ability to adapt themselves in light of new data, might provide a future avenue to flexible, self-updating systems that can utilize incoming data-points to improve their own performance at a given task.

This work was undertaken in late 2018, early 2019, and since then, there has been a lot of interest in meta-learning loss functions [Huang et al., 2019, Li et al., 2019], rewards [Houthooft et al., 2018b, Jaderberg et al., 2017], and other types of risk minimization methods [Zhang et al., 2020, Kye et al., 2020]. In particular, the authors in Zhang et al. [2020] learn an explicit risk minimization function, which works in a very similar way to the work in this section, but further expands by directly generating new parameters given an unlabelled target set, without any labelled data. Furthermore, the authors of Kye et al. [2020] meta-learn a *confidence* function that weights each one of the target-set samples in terms of usefulness before they are used in a transductive objective to update the model.

Chapter 5

Assume, Augment and Learn: Self-Supervised Learning for Unsupervised Few-Shot Learning using Data Augmentation and Random Labels

5.1 Introduction

Much of the success of modern deep learning relies on supervised learning. Supervised learning describes a family of learning algorithms that, given a set of input-output pairs, learns a model that maps a set of input data to a set of output labels. A good supervised model should not only be able to predict the correct output for the data it was trained on (i.e. the training set) but also have high generalization power (i.e. to be able to predict the correct outputs for previously unseen data points).

In general, training a deep neural network model via supervised learning requires a large number of input-output data-pairs. Labels are usually acquired by human annotators, and the labelling process is laborious. There have been attempts at semi-automating the process [Parkhi et al., 2015] but even those require a strong pretrained model to be available, which needs substantial data in order to be trained in the first place. The data labeling component of data acquisition is considered by many to be the most costly and time consuming part of the machine learning pipeline. In stark contrast, for many problems, unlabeled data is freely available in large quantities.

In the context of few-shot learning, the fact that a few-shot learner using just 24K handwritten digits that can achieve over 95%+ on a variety of tasks is impressive [Finn et al., 2017b, Snell et al., 2017a, Ravi and Larochelle, 2016, Antoniou et al., 2018a]. However, a well designed machine learning algorithm should be able to utilize other larger unlabeled datasets, related to but not identical to the test scenario, to meta-learn robust few-shot learning models. Such an unsupervised few-shot learner, once trained, would acquire task-specific knowledge from a test-time support set of different data and generalize well on the test-time target set.

Meta-learning has been demonstrated to be an effective method for learning models that can achieve effective few-shot learning [Finn et al., 2017a, Antoniou et al., 2018a, Rusu et al., 2019, Qiao et al., 2018b]. However, most of the work in meta-learning relies on a supervised outer-loop, for which task-related labels need to be available for successful learning. Instead, we are interested in investigating unsupervised learning as the outer-loop optimization's learning style. Such unsupervised meta-learning can be achieved via self-supervised methods that generate artificial tasks that emulate real tasks, such that when a learning prior is meta-learned with it, it can generalize to real tasks.

In this chapter we propose such a method for leveraging unsupervised data to generate tasks for few-shot learners. In summary, we create support sets by assigning random labels to subsets of the unsupervised data and generate corresponding target sets through data-augmentation transformations of the support set data points; these transformed data points have the same labels. Our unsupervised meta-learner must learn to learn networks that consistently classify correctly both the randomly labelled support set and the data augmented examples (target-set). Such a learnt model can then be applied to real labelled data in the test setting (see Figure 5.1). We call our method *Assume, Augment and Learn* (AAL).

The approach is evaluated on two separate types of models, the Model Agnostic Meta Learning framework [Finn et al., 2017b] and the Prototypical Networks framework [Snell et al., 2017a]. Once a few-shot learner is trained using the tasks generated with the proposed method, it can then be used at test time with a few shot support set with real labels, without any fine tuning, and then generalize to the target task.

A support and target set are built using only unlabeled data. For the support set, we sample a number of data-points from our training dataset and then, since no labels are available, we *assume* labels by randomly generating a label for each image. In a supervised setting the target set contains new instances of the same classes found in the support set. In our unsupervised case, we apply data augmentation methods to the support set (keeping the labels consistent), hence creating a target set with semantically the same classes as the support set, but different enough to serve as a generalization evaluation.

This chapter's contributions are:

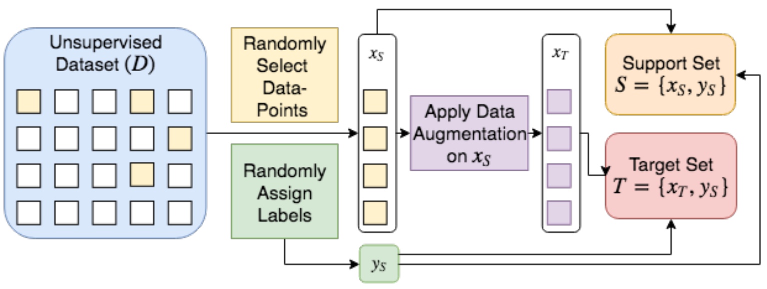

Unsupervised Dataset (D)
Randomly Select Data-Points
Randomly Assign Labels
x_S
Apply Data Augmentation on x_S
x_T
Support Set $S = \{x_S, y_S\}$
Target Set $T = \{x_T, y_S\}$
y_S

Figure 5.1: Proposed method. We follow the set-to-set [Vinyals et al., 2016] few-shot learning framework, where during inference time, a learner acquires task-specific information from a **support set**, and is then evaluated on a **target set** for that task. Building support and target sets for few-shot tasks ordinarily requires labelled information. However, to produce such tasks in the absence of labelled data, we randomly select some data points from a label-free dataset and assign random labels to them, in order to form the support set. Then, the support set data-points are augmented, and the label information kept the same to form the target set.

1. Proposal of new self-supervised meta-learning algorithm for few-shot learning using Random Labels and Data Augmentation, producing state-of-the-art results in Omniglot and competitive performance on Mini-ImageNet.
2. Thorough evaluation of possible data augmentation strategies via a large ablation study.

In the following sections, we first describe our setting, then the MAML and ProtoNets frameworks on which we built our method. Then we proceed to describe our method and the evaluation experiment types used. Finally we present our empirical results and draw our conclusions.

5.2 Related Work

Unsupervised deep learning has been extensively investigated in the context of generative models [Doersch, 2016, Vincent et al., 2008, Goodfellow et al., 2014], yet, have only been briefly attempted in classification models [Caron et al., 2018, Bojanowski and Joulin, 2017].

Concurrent work in unsupervised few-shot learning was introduced in the form of CACTUs [Hsu et al., 2018, Huang et al., 2020], where the authors propose the generation of support and target sets using k-means clustering in combination with semantically meaningful representations. Furthermore, work has also been carried out in the semi-supervised setting, which attempts to learn an unsupervised loss function [Rinu Boney, 2018], such that a MAML-based model can learn from an unlabeled support task and generalize well on a target task. In work developed concurrent to this [Khodadadeh et al., 2019], the authors meta-learn few-shot models to maintain labels under data augmentation. In comparison we conducted extensive ablation studies on data-augmentation strategies, ranging from very simple crops all the way to warping [Wong et al., 2016] and cutout [DeVries and Taylor, 2017]. We also evaluate our method on the MAML model and our results outperform UMTRA on the Omniglot tasks. UMTRA also utilizes data-augmentations learned in the AutoAugment [Cubuk et al., 2018]. However, the augmentations in AutoAugment were learned over the whole of ImageNet; utilizing AutoAugment might be considered as transfering knowledge from a model trained on a vastly larger labelled dataset. Which, in a way, defeats the purpose of unsupervised learning.

Contrastive learning Chen et al. [2020], Chuang et al. [2020] is another family of unsupervised (and self-supervised) methods, which revolve around the idea of training a model by *pushing* similar samples closer together, and *pulling* dissimilar samples further apart in some representation space. Similarity and dissimilarity is usually what sets apart the different types of methods in this family. In some, similar data points might be those coming from the same instance, but have been augmentented, while dissimilar samples are those from two separate instances. The transforms that make samples similar or dissimilar are usually referred to as *views* Tian et al. [2020].

The work in this paper is similar to contrastive learning in that it creates pseudo-labels using data-augmentations, but is different in that contrastive learning pushes and pulls around similar and dissimilar data-points, whereas our work focusses on learning a learner that can quickly adapt to a new dataset, and generalize well to data augmented versions of that dataset, hence no explicit pulling is happening.

5.3 Setting

Given an unsupervised dataset we want to learn a few-shot learning model that can generalize very well on small few-shot labeled dataset-tasks. We make use of three sets. The first, the *meta-training* set, is label-free and used to train a few-shot learner using the proposed unsupervised method. The second and third sets, are called the *meta-validation* and *meta-test* sets respectively, and contain labeled data, to be used to create evaluation few-shot tasks. Using a validation set to pick the best train model and a test set to produce the final test errors removes any potential unintended (human-derived/implicit) over-fitting. Intuitively, the meta-validation and meta-test sets are generated in the exact same manner that the validation and test tasks are generated in any other few-shot learning methodologies such as MAML [Finn et al., 2017b] and Prototypical Networks [Snell et al., 2017a].

Training meta-learning models requires using a large number of *tasks*. To generate a few-shot task we use the *set-to-set* few-shot learning scheme proposed in [Vinyals et al., 2016]. In the set-to-set few-shot learning setting, a task is composed by a training or *support* set and a validation or *target* set. The support set consists of a number of classes (N), and a small number of data-samples per class (K). The target set consists of samples from the same classes as the support set, but using different instances of those classes. A setting of N classes per set and K samples per class can alternatively be referred using the shortened notation of *N-way, K-shot learning* setting.

5.4 Motivation

Initial motivation for the work came from the human ability of being able to find features that can accurately describe a set of randomly clustered data-points, even when the clusters are created randomly. Transferring this to a machine learning setting could produce strong representations for classification.

However, using random clusters in a standard deep-learning setting would not work well since the model is trying to learn a mapping from x to y, that holds for all seen x to y pairs. If one randomly reset the labels (y) as training progresses to create new tasks, the model would not be able to formulate a solution that will generalize across all x to y pairs since many of them will be contradictory. However, intuitively, it should work in a meta-learning setting. In a meta-learning setting the goal is to be able to learn a model that at inference time can acquire task-specific knowledge from a support set such that it can do well on the target set for the same task. Intuitively, learning a meta-model that captures cross-task similarities but throws away task-specific information (such as the specific class clusters for a given task) would still work well. Exploiting semantic similarities between data points to learn robust across-task representations that can then be used in a setting where supervised labels are available.

5.5 Generating Tasks for Unsupervised Few-Shot Learning

To enable unsupervised learning in a few-shot classification setting, one must find a way to generate tasks. As defined in Section 5.3, a task is composed of a support and target set. A support set consists of samples from a number of distinct classes and their associated labels. Since no label information is available, we instead *assume* labels by randomly assigning labels for a randomly sampled set of data-points. This will effectively cluster the data items into groups. Since the model takes update steps on the support set, it should be able to acquire fast-knowledge in its parameters that classify that support set very well. Generating a semantically related (to the support set) target set is where much of the difficulty lies. A target set should have *different* instances of the *same* classes appearing in the support set. Since we randomly assigned classes to the data-points we have no information on the semantics connecting the samples together in a class. To overcome this problem we generate a target set by *augmenting* the support set data-points using data augmentation. Classes in the support set then perfectly match the classes in the target set, yet the samples in the two sets are different enough to allow the target set to serve as a good evaluation set. In Algorithms 2 and 3 one can see how a supervised and unsupervised sampling procedure works.

Algorithm 2 Supervised MAML Sampling Strategy

```
1: **Require:** Dataset $\mathcal{D}$ with $\mathcal{C}$ number of classes and $\mathcal{M}$ samples per class
2: Sample $N$ class indexes from $\mathcal{D}$, where $N \leqslant \mathcal{C}$
3: Sample the support set $S$ by sampling $K$ samples from each selected class where $K \leqslant \mathcal{M}$
4: Sample the target (evaluation) set $E$ by sampling $J$ samples from each selected class, making sure that these samples do not contain any support set images where $E \leqslant \mathcal{M}$ and $E \cap K = \emptyset$
5: **Return** $S$, $E$
```

[1]One should choose the same number of classes and number of samples per class as for the supervised setting the model will be tested on at inference time.

Algorithm 3 Unsupervised MAML Sampling Strategy

```
1: Require: Dataset $\mathcal{D}$ with $I$ number of data-points
2: Sample $N \times K$ data-points from $\mathcal{D}$, where $N$ is the number of classes per set[1] and $K$ is the number of samples per class $(N \times K) \leqslant I$
3: Build the support set $S$ by assigning random labels to the previously $N \times K$ sampled data-points
4: Build the target (evaluation) set $E$ by augmenting the support set $S$ samples and keeping the labels identical
5: Return $S$, $E$
```

5.6 Meta-Learning Frameworks used for Evaluation

AAL + MAML can be applied on any existing few-shot meta-learning technique, as long as the method can be trained using the set-to-set few-shot learning framework. To demonstrate this, we apply our method on two very popular and effective few-shot meta-learning frameworks, the Prototypical Networks framework and the MAML framework.

Prototypical Networks: Prototypical Networks [Snell et al., 2017a] were introduced in an attempt to improve performance and reduce computational and storage requirements of Matching Networks. In Prototypical Networks, the authors propose using per-class or *prototype* embeddings instead of per-sample embeddings. More specifically they propose computing the mean of per-class embeddings, hence obtaining a single embedding vector that summarizes a particular class. Once the prototypical embeddings are computed, they are compared with a target sample's embedding, using a suitable distance metric, such as euclidean or cosine distance, to obtain a probability distribution over the prototypical classes predicting the target sample's class membership. Not only does the usage of prototypical embeddings improve the generalization performance of the system, but it also reduces both the computational and space complexity of the system. In particular, using prototypical embeddings renders the computational complexity of computing the similarity vectors linearly proportional to number of classes (N) instead of number of samples ($N \times K$), hence reducing the computational complexity in all cases, except the case when $K = 1$. Furthermore, the storage complexity is also reduced from $N \times K$ to N.

Model Agnostic Meta-Learning [Finn et al., 2017b] or (MAML) is a meta-learning framework for few-shot learning. It approaches few-shot learning by learning an initialization for the parameters of a base-model. Given a task, MAML computes the loss and gradients of a base-model with respect to the support set and takes a step in the direction of the gradients, thus acquiring task-specific knowledge usually refered to as the *inner-loop* optimization process. This process is repeated a number of times. Once the inner loop update steps have been completed, the base model is evaluated on the task's target set to obtain the target set loss. Once computed, the target loss, is used to update the base-model's initialization, by taking a step in the direction that minimizes the target loss. Effectively, this trains a base-model parameter initialization, that after a number of SGD updates wrt a support set, the model can generalize very well on a target set.

5.7 Selecting Target Set Data Augmentations

Selecting the right combination of data augmentation techniques is crucial for our proposed approach. Selecting a data augmentation method which makes minimal changes to the data-points could produce samples that are too similar to the originals (thus being ”too easy”) for the model and thus causing reduced generalization performance. On the other hand using a data augmentation that changes the data-points too much could change semantics within the image that are important for the class of the image. Removing semantically important features from the image can make learning very hard, and in addition, guide the network to learn a function that does not generalize well to real-labelled tasks.

Name	Omniglot Hyperparameters	Mini-Imagenet Hyperparameters
Random Crops (C)	Padding: 7 pixels	Padding: 21 pixels
Random Horizontal Flips (H)	Flip probability 50%	Flip probability 50%
Random Vertical Flips (V)	Flip probability 50%	Flip probability 50%
Random Rotations (R)	1 - 30 degrees	1 - 270 degrees
Image-Pixel Dropout (DROP)	Drop probability: 30%	Drop probability: 70%
Image-Pixel Cutout (CUT)	Holes: 5, Side Length: 4 - 14 pixels	Holes: 5, Side Length: 11 - 42 pixels
Image-Warping (W)	Warping area: 14 + U(0, 6) x 14 + U(0,6)	Warping area: 42 + U(0, 41) x 42 + U(0,41)
Random Grayscale (G)	Not used	Grayscale probability: 50%

Table 5.1: This table presents the types and settings of data augmentation used to generate target sets in our experiments.

To mitigate any requirement for excessive compute, we chose to search through combinations of rotations, cutout, dropout, warping and grayscaling of images, whilst keeping random crops, horizontal and vertical flips shared across all experiments. Doing so keeps the number of required experiments manageable. Preliminary experiments with crops, horizontal and vertical flips demonstrated that their combination improved the generalization performance of models across all few-shot tasks across both Omniglot and Mini-Imagenet, hence their usage as shared (and baseline) augmentations.

	5-way Accuracy (Test)		20-way Accuracy(Test)	
Algorithm	1-shot	5-shot	1-shot	5-shot
Randomly initialized	$70.3 \pm 0.36\%$	$86.9 \pm 0.27\%$	$44.7 \pm 0.45\%$	62.1 ± 0.85
AAL + MAML + CHV	$75.8 \pm 0.63\%$	$96.6 \pm 0.32\%$	$42.0 \pm 1.16\%$	76.6 ± 1.14
AAL + MAML + CHVW	$\mathbf{88.4 \pm 0.75}\%$	$\mathbf{98.0 \pm 0.32}\%$	$\mathbf{70.2 \pm 0.86}\%$	$\mathbf{88.3 \pm 1.22}$
AAL + MAML + CHVR	$78.0 \pm 2.39\%$	$96.6 \pm 0.04\%$	$58.9 \pm 1.16\%$	81.6 ± 0.82
AAL + MAML + CHV + DROP	$85.7 \pm 1.09\%$	$97.1 \pm 0.12\%$	$65.1 \pm 0.62\%$	83.8 ± 0.62
AAL + MAML + CHV + CUT	$74.0 \pm 1.60\%$	$95.9 \pm 0.44\%$	$54.0 \pm 1.59\%$	75.2 ± 0.62
AAL + MAML + CHV + DROP + CUT	$85.7 \pm 1.09\%$	$95.7 \pm 0.38\%$	$68.3 \pm 0.63\%$	85.8 ± 0.62
AAL + MAML + CHVR + DROP	$86.16 \pm 1.09\%$	$96.5 \pm 0.87\%$	$64.6 \pm 0.39\%$	82.0 ± 0.19
AAL + MAML + CHVR + CUT	$77.8 \pm 1.00\%$	$95.1 \pm 1.05\%$	$60.2 \pm 0.66\%$	80.6 ± 0.53
AAL + MAML + CHVR + CUT + DROP	$85.6 \pm 1.05\%$	$96.7 \pm 0.45\%$	$66.5 \pm 0.27\%$	83.1 ± 0.64
MAML (supervised upper bound)	$99.4 \pm 1.05\%$	$99.9 \pm 0.45\%$	$97.7 \pm 0.27\%$	99.1 ± 0.64

Table 5.2: AAL + MAML Omniglot Ablation studies. Abbreviation explanations can be found in Table 5.1 One can clearly see that the best results 1-shot were obtained with the mixture of crops, vertical and horizontal flips while the best results for 5-shot were obtained with the same combo as in the 1-shot optimal configuration but with extra use of cutout.

5.8 Datasets

To evaluate our methods we used the Omniglot and Mini-Imagenet datasets. As discussed in Section 5.3 we split our datasets in a background (or meta-training) set and an evaluation set (which is then further split into meta-validation and meta-test sets). The Omniglot dataset is composed of 1623 character classes from various alphabets. There exist 20 instances of each class in the dataset. For meta-training, meta-validation and meta-test sets we use classes 1-1150, 1150-1200 and 1200-1623 respectively. The Mini-Imagenet dataset was proposed in [Ravi and Larochelle, 2016], it consists of 600 instances of 100 classes from the ImageNet dataset, scaled down to 84x84. We use the split proposed by the authors, which consists of 64 classes for training, 12 classes for

	5-way Accuracy (Test)		20-way Accuracy(Test)	
Algorithm	1-shot	5-shot	1-shot	5-shot
Randomly initialized	25.72 ± 0.22%	28.03 ± 0.28%	07.31 ± 0.41%	08.04 ± 0.36%
AAL + ProtoNets + CHV	**84.23 ± 0.93%**	**89.14 ± 0.36%**	64.34 ± 1.07%	**74.28 ± 1.76%**
AAL + ProtoNets + CHV + CUT	82.78 ± 2.25%	**89.04 ± 0.09%**	64.23 ± 1.06%	71.37 ± 1.18%
AAL + ProtoNets + CHV + CUT + DROP	83.38 ± 0.60%	**88.98 ± 0.88%**	66.26 ± 0.96%	**73.94 ± 1.18%**
AAL + ProtoNets + CHVR + CUT	82.06 ± 0.53%	87.78 ± 0.21%	64.77 ± 1.46%	70.66 ± 0.53%
AAL + ProtoNets + CHV + DROP	**84.66 ± 0.70%**	88.41 ± 0.27%	**68.79 ± 1.03%**	**74.05 ± 0.46%**
AAL + ProtoNets + CHVR + DROP + CUT	81.24 ± 0.62%	87.15 ± 0.86%	65.31 ± 0.76%	72.04 ± 0.33%
AAL + ProtoNets + CHVR + DROP	82.65 ± 0.36%	88.13 ± 0.75%	**67.10 ± 1.13%**	71.92 ± 0.41%
AAL + ProtoNets + CHVR	82.12 ± 0.97%	86.04 ± 1.09%	64.39 ± 1.27%	70.39 ± 0.99%
ProtoNets (supervised upper bound)	98.92 ± 0.07%	99.56 ± 0.07%	97.87 ± 0.11%	98.90 ± 0.10%

Table 5.3: AAL + ProtoNets Omniglot ablation studies. Abbreviation explanations can be found in Table 5.1 One can clearly see that the best results 1-shot were obtained with the mixture of crops, vertical and horizontal flips while the best results for 5-shot were obtained with the same combo as in the 1-shot optimal configuration but with extra use of cutout.

validation and 24 classes for testing. We use the training, validation and testing sets as the meta-training, meta-validation and meta-test respectively for our experiments.

5.9 Experiments

To evaluate the generalization performance of the unsupervised trained models on supervised few-shot tasks we trained our model using the AAL + ProtoNets method and after each unsupervised meta-training epoch we applied the resulting model without any additional fine tuning on the supervised meta-validation set. Once all epochs were completed, the best validation model was then applied on the test set which produced the final test performance of the model.

We applied AAL + ProtoNets on two separate model types. The first, was the MAML++ [Antoniou et al., 2018a] model (an improved variant of MAML), more specifically, we used the LSLR (Per-Layer Per-Step Learning Rates), along with the per-step batch norm modifications (BNWB/BNRS) and the multi-step loss optimization (MSL) to both stabilize training and improve the generalization performance. The second, was the prototypical networks [Snell et al., 2017a] model.

To apply AAL + ProtoNets on a given model, we simply sampled training tasks

	5-way Accuracy (Test)	
Algorithm	1-shot	5-shot
Randomly initialized	$28.55 \pm 0.22\%$	$36.68 \pm 0.23\%$
AAL + MAML + CHV	$33.06 \pm 0.49\%$	$40.75 \pm 1.03\%$
AAL + MAML + CHVR	$33.21 \pm 0.90\%$	$40.34 \pm 1.64\%$
AAL + MAML + CHV + CUT	$33.34 \pm 0.54\%$	$39.44 \pm 0.94\%$
AAL + MAML + CHV + DROP	$30.86 \pm 0.64\%$	$40.41 \pm 0.68\%$
AAL + MAML + CHVW	$33.30 \pm 0.31\%$	$46.98 \pm 0.16\%$
AAL + MAML + CHVWG	$\mathbf{34.57 \pm 0.74}\%$	$\mathbf{49.18 \pm 0.47}\%$
AAL + MAML + CHVR + CUT	$33.09 \pm 0.51\%$	$40.11 \pm 0.76\%$
AAL + MAML + CHVR + DROP	$31.70 \pm 0.54\%$	$39.38 \pm 0.57\%$
AAL + MAML + CHV + DROP + CUT	$31.55 \pm 0.85\%$	$38.76 \pm 0.83\%$
AAL + MAML + CHVR + DROP + CUT	$31.44 \pm 0.78\%$	$39.87 \pm 0.89\%$
MAML (supervised upper bound)	$52.15 \pm 0.26\%$	$67.87 \pm 0.42\%$

Table 5.4: AAL + MAML Mini-ImageNet ablation studies. Abbreviation explanations can be found in Table 5.1 One can clearly see that the best results for both 1-shot and 5-shot can be obtained with the mixture of crops, vertical and horizontal flips, warps and gray scaling.

using the AAL + ProtoNets sampling scheme, and then evaluated our model on validation and test tasks using the supervised meta-learning formulation as done in [Finn et al., 2017b]. As a result, the evaluation metrics would effectively be quantifying the transferability of the representations learned using unsupervised data, on a real-labelled supervised meta-learning task, rendering our evaluation results directly comparable to supervised meta-learning models.

5.10 Neural Network Architectures

The architecture used for the base classifier was the one proposed in [Finn et al., 2017b]. A convolutional neural network with 4 layers. Each layer is composed of a convolutional layer followed by batch normalization followed by a ReLU non-linearity followed by max-pooling. Every convolutional layer, had 64 filters, a padding of 1, a

	5-way Accuracy (Test)	
Algorithm	1-shot	5-shot
Randomly initialized	$22.96 \pm 0.05\%$	$24.73 \pm 0.10\%$
AAL + ProtoNets + CHV	$\mathbf{37.67 \pm 0.39}\%$	$\mathbf{40.29 \pm 0.68}\%$
AAL + ProtoNets + CHV + CUT	$36.38 \pm 0.91\%$	$\mathbf{40.89 \pm 1.30}\%$
AAL + ProtoNets + CHV + CUT + DROP	$33.13 \pm 0.95\%$	$36.64 \pm 0.79\%$
AAL + ProtoNets + CHVR + CUT + DROP	$31.93 \pm 0.28\%$	$36.45 \pm 1.33\%$
AAL + ProtoNets + CHVR + CUT	$33.92 \pm 0.60\%$	$39.87 \pm 0.53\%$
AAL + ProtoNets + CHV + DROP	$32.12 \pm 0.34\%$	$36.12 \pm 1.11\%$
AAL + ProtoNets + CHVR + DROP	$31.13 \pm 0.48\%$	$36.83 \pm 0.57\%$
AAL + ProtoNets + CHVR	$34.28 \pm 0.21\%$	$39.83 \pm 0.85\%$
ProtoNets (supervised upper bound)	$50.16 \pm 0.15\%$	$65.56 \pm 0.13\%$

Table 5.5: AAL + ProtoNets Mini-ImageNet ablation studies. Abbreviation explanations can be found in Table 5.1 One can clearly see that the best results 1-shot were obtained with the mixture of crops, vertical and horizontal flips while the best results for 5-shot were obtained with the same combo as in the 1-shot optimal configuration but with extra use of cutout.

Algorithm (way, shot)	(5, 1)	(5, 5)	(20, 1)	(20, 5)
Training from scratch	52.50%	74.78%	24.91%	47.62%
ACAI CACTUs-MAML	68.84%	87.78%	48.09%	73.36%
ACAI CACTUs-ProtoNets	68.12%	83.58%	47.75%	66.27%
BiGAN CACTUs-MAML	58.18%	78.66%	35.56%	58.62%
BiGAN CACTUs-ProtoNets	54.74%	71.69%	33.40%	50.62%
UMTRA	77.80%	92.74%	62.20%	77.50%
AAL + ProtoNets (ours)	84.66%	89.14%	68.79%	74.28%
AAL + MAML (ours)	**88.40%**	**97.96%**	**70.21%**	**88.32%**
MAML (supervised upper bound)	94.46%	98.83%	84.60%	96.29%
MAML (supervised upper bound)	99.42%	99.93%	97.76%	99.12%
ProtoNets (supervised upper bound)	98.92%	99.56%	97.87%	98.90%

Table 5.6: Comparative Omniglot results. In Omniglot our methods achieve the top state-of-the-art performance across the board.

stride of 1 and a filter size of 3.

In the case of MAML, following the convolutional network, we applied a linear layer and a softmax activation function to produce the network's predictions. The size of the last layer is equal to the number of classes per set.

In the case of Prototypical Networks, the networks predictions are the flattened outputs of the fourth convolutional layer, exactly as done in the original chapter.

5.11 Results

Tables 5.7 and 5.6 present comparative results between a variety of baseline and SOTA methods for unsupervised few shot learning. Our proposed method outperforms all other methods across all Omniglot tasks, and it remains competitive in the Mini-Imagenet 5-way 1-shot task. In the 5-way 5-shot Mini-Imagenet our method is out-performed by the CACTUs methods. Furthermore, to investigate the effect of the data augmentation strategies on the generalization performance of our method we conducted ablation studies. Tables 5.2, 5.3, 5.4 and 5.5 showcase the results of the ablation studies. Our ablation studies revealed that for the Omniglot task, the most

Algorithm (way, shot)	(5, 1)	(5, 5)
Training from scratch	27.59%	38.48%
DeepCluster CACTUs-MAML	**39.90%**	**53.97%**
DeepCluster CACTUs-ProtoNets	39.18%	53.36%
UMTRA + AutoAugment	**39.93%**	50.73%
AAL + ProtoNets	37.67%	40.29%
AAL + MAML	33.30%	49.18%
MAML (supervised upper bound)	46.81%	64.13%
MAML (supervised upper bound)	52.15%	67.87%
ProtoNets (supervised upper bound)	50.16%	65.56%

Table 5.7: Comparative Mini-Imagenet results. In Mini-ImageNet our method achieves performance competitive to the top methods when utilized with ProtoNets in the 1-shot case and when used with MAML in the 5-shot case.

useful data augmentation methods are pixel-dropout and warping. Warping produces state of the art results in Omniglot, we hypothesize that this is due to the augmented samples matching very closely natural variations of the source images. In the Mini-Imagenet tasks we observed that, surprisingly, crops and flips outperformed all other methods. Much of the success of our Omniglot results, came from using augmentations that actually generate a target set that is meaningful yet initially difficult for the underlying meta-learning method to perform on. In Mini-Imagenet, we observed that all attempted models would converge within 5 epochs. This indicates that, perhaps, the data augmentation strategies utilized were not *challenging* enough, hence fitting quickly and not generalizing very well, which is contrary to Omniglot, where models take longer to converge and generalize a lot better. Therefore, it is likely that with more work towards discovery or automatic learning of the right data augmentation strategies might yield even better results.

5.12 Conclusion

Learning robust representations using label-free data is crucial task for any truly intelligent agent. In this chapter, we propose a methodology that allows a machine to create its own tasks using unlabelled datasets by leveraging randomly clustered data-points and augmentation. Models trained with our method, are able to generalize well to real-labelled inference tasks. We compared our method against a variety of recent unsupervised few-shot methods, evaluating the method on Omniglot and Mini-Imagenet. Our model outperformed all other competitors in Omniglot, whilst remaining competitive in the 5-way 1-shot Mini-ImageNet task. However, in the 5-way 5-shot task, our method was outperformed by CACTUs. Finally, a very noteworthy observation is that the performance of our method is heavily dependent on the data augmentation strategies employed. Hence, finding the most beneficial choice is of some importance. Automating the search for such a data augmentation function could produce fully unsupervised systems with stronger performance.

This work was undertaken in late 2018, early 2019, and since then, the deep learning research landscape has seen many works that utilize clever schemes to achieve self-labelling, on which a model can be trained. Such methods have been dubbed *self-supervised* methods, and have been applied both in standard deep learning, under the form of contrastive learning methods [Chen et al., 2020, Tian et al., 2020, Chuang et al., 2020], as well as *self-training* methods [Zhang et al., 2019]. Furthermore, self-supervised meta-learning has also become a popular research direction, producing increamentally better performing models [Hsu et al., 2018, Liu et al., 2019b, Gidaris et al., 2019, Gupta et al., 2018].

Chapter 6

Conclusion

6.1 Summary and Key Points

In this document, we presented a thesis on meta-learning few-shot learning algorithms. The first project, *How to train your MAML*, enabled the follow up projects by stabilizing and improving the convergence speed and generalization performance of the MAML framework. The results from this project, clearly showcased how gradient degradation can be reduced within a MAML model by employing the multi-step loss method alone, however, with some additional modifications, such as the per-layer, per-step batch normalization method and the per-layer, per-step learning rates method the model stability can be further improved along with the generalization and convergence speed of the model.

In the second project, we demonstrated that casting unsupervised loss functions as small neural networks, and learning them within a gradient-based meta-learning framework, can generate substantial improvements in generalization performance by leveraging the available unsupervised information contained within a target set, along with the already available supervised information within the support set. Understanding what such a learned loss function is intuitively doing is quite complicated, however, by employing adversarial attack-style methods, we were able to acquire some information about what the loss function considers as an acceptable input and what it does not. We were able to infer that the simplest variant of the model, the one only receiving as inputs the predictions of the classifier on the target set, learned some type of prediction reqularizer, which aims to ensure that the confidence of the model remains within a range that, through across-task training, produces better generalization performance. At the time of publishing this project as a paper, the performance of the models trained in this project, achieved top-in-the-world state-of-the-art results on Omniglot, Mini-

ImageNet and CUB-200. While this is no longer the case, this model still remains within the top-5 models after more than 15 months after its development.

In the third project, we turned our attention towards learning few-shot learning models via unsupervised meta-learning. Achieving such unsupervised meta-learning requires building training tasks that are similar to the meta-test tasks that the model should perform well on. Performing well on a few-shot learning task, means that given a labelled support set, a system must produce a model, that can generalize well on a previously unseen target set. To generate tasks that imitate the task we care for, we build a labelled support set by randomly sampling data-points from a large dataset, and assigning random labels to them. Then, we apply data augmentation on the support set to generate a target set that contains samples within the same class distribution as a given support set, but that are different enough to be good at emulating a previously unseen sample distribution. Once a model is meta-learned to perform well on thousands of such self-supervised tasks, it can generalize well to real few-shot tasks, containing human labelled examples very well. Our model produces state of the art results on Omniglot, and remains competitive with state of the art models on Mini-ImageNet.

6.2 Broader Conclusions

In this thesis we have demonstrated that meta-learning can indeed be used as a framework for learning to few-shot learn. In all instances, we have demonstrated that meta-learning increasingly more components has allowed increasingly better results, as evidenced from moving from meta-learning a parameter initialization in MAML, to joint learning of learning schedulers and parameter initialization in MAML++ and then joint learning of initialization, learning scheduler and loss functions.

Related work has provided further evidence to this, by meta-learning a variety of components, from optimizers, to auxiliary tasks, to dynamic parameter initializers among others. However, as described in Section 2.2.15, manually invented learners have recently began to be just as effective as meta-learning. Therefore, one might begin to wonder here whether the issue lies in that we need better meta learning algorithms, better/harder benchmarks or whether we need something else completely different to properly understand what meta-learning can do well and where it becomes necessary to use in comparison with non meta-learning methods. Recent work has attempted to enrich the data distribution of the few-shot image classification tasks in Triantafillou et al. [2020], which provides evaluation signal with a lot more degrees of variation, and as such can produce more robust conclusions for a given model.

Even in the face of existing non-meta-learning methods, being able to just meta-learn a component on 8 hours on a single GPU, remains more efficient than spending weeks, at best, manually inventing a single learner. However, one may argue that then a researcher spends more time inventing the meta-learner for a given system. While that is true, one could replace that step with yet another meta-learning step, and allowing the very top level optimizer to be as general and robust as possible. In any case, more work is needed to understand what is really the best way to robust learning algorithm discovery.

6.3 Relevance of Work in Reinforcement Learning

This thesis covered supervised and unsupervised meta-learning for few-shot learning methods extensively, however, it did not delve much into how such meta-learning methods may be relevant to another very important family of methods – that is – Reinforcement Learning (RL) [Sutton and Barto, 2018]. In reinforcement learning there usually exists an agent, an environment and a reward signal, and the task is to train the agent such that it behaves in a way in its environment such that it can maximize the reward signal.

Existing reinforcement learning methods are notorious for their data inefficiency and extreme sensitivity to hyperparameters [Hospedales et al., 2020, Schulman et al., 2017]. Even if good hyperparameters are found, training runs with different random number generators can yield drastically different results. This combination of inefficiency and instability, makes RL research progress and utility in real life applications a lot less impactful.

Meta-learning offers a potential way to tackle these issues. By learning a trainer for such agents, such that certain desirable properties such as sample efficiency and stability are maximized, one can effectively automatically learn RL algorithms that may be able to surpass human-invented ones.

Such work is already under way, from learning initial conditions, to optimizers [Houthooft et al., 2018a], to intrinsic rewards/losses [Zhou et al., 2020, Veeriah et al., 2019, Jaderberg et al., 2019, Zheng et al., 2018], exploration strategies [Xu et al., 2018a], hyperparameters [Xu et al., 2018b, Young et al., 2019] and even augmentations [Cubuk et al., 2019]. The work in this thesis is also relevant to RL, since MAML++

(Section 3) methods can be directly applied to RL without any modifications other than the supervised optimization being changed into an RL one, similar to how Finn et al. [2017a] originally adapted MAML to work in RL settings. Furthermore, the Self-Critique and Adapt method (Section 4, may also be similarly applied to either learn intrinsic losses/rewards for an RL optimizer, or, may be used to learn a lifelong learning loss that allows an agent to learn while being online in an RL environment. Finally, Assume, Augment and Learn (AAL) (Section 5, can also be applied to learn from replay memory of an RL system i.e. once some experience has been collected by an RL exploration module, one could use such experience with AAL to learn a useful representation that can then be tuned into a policy or a value function.

6.4 Future Work

6.4.1 Few-Shot Learning

Work from this thesis and the few-shot learning research landscape, clearly reveals that both meta-learning and non-meta-learning few-shot learning methods, have a clear generalization problem, that is, once a learner has been learned on the meta-training set, and applied on the meta-test set, the performance degradation is quite significant, especially on natural data such as Mini-ImageNet and CUB-200. Thus, understand why that is happening is an important next step. The community has attempted meta-learning a large variety of different components, such as parameter initilizers, loss functions, learning schedulers, optimizers and auxiliary tasks. Many of those have shown significant gains in generalization performance, but as of yet, none have been able to achieve upwards of 80% of training accuracy on the one shot tasks. Ofcourse, since the human performance for few-shot learning has not been established, we don't really know what is a good upper bound that we should be aiming for, thus, it is always best to aim for the maximum possible score.

An interesting direction would be to attempt meta-learning component types that have not been tried yet, such as a form of learnable backpropagation or a data-conditional learning rate scheduler, as a sort of neural plasticity network, that can perhaps choose better learning rates for a given filter bank, compared to layer-level and step conditional learning schedulers.

On the other hand, perhaps there is also a need for more flexible architecture searching. Learning an architecture starting from basic components like the dot product and building up to complex neural networks might allow more flexibility in what computational components can be learned, compared the the existing architecture search starting from existing layer types and trying to synthesize a single computational block to be repeated.

Furthermore, a good next direction might be to investigate new outer loop meta-learning algorithms. Evolutionary algorithms might be a good algorithm type to investigate as they tend to make minimal assumptions about the task structure, require no gradients to be computed for the outer loop update, and are easily parallelizable.

6.4.2 Meta-Learning

Meta-learning is an exciting and promising new learning paradigm, that if utilized correctly could lead to the next wave of innovation in artificial intelligence. However, at the current state, meta-learning methods are mostly focused on few-shot learning which has a significant impact on both the types of models that work well enough to be published, as well as the utility of those models to the deep learning community.

A very important area to consider is that of benchmark design for meta-learning. If we want to maximize the information available to meta-learning researchers about the models they spend their valuable time on, we need new, more informative benchmarks. We need to know how a given meta-learning model works on a variety of different domain shifts [Triantafillou et al., 2020], task types (i.e. few-shot and many-shot image classification, relational reasoning, regression, segmentation, generation, abstract reasoning, reinforcement learning, continual and continual few shot learning etc). Having access to more metric signal would be vital in identifying good meta-learning models, both during the research stage and at the peer-review stage. Benchmarks are effectively the objective that optimizes a whole subfield. Should those be chosen poorly, a lot of resources will be wasted. Thus, while working on improving meta-learning methods themselves is important, it's equally or even more important to improve current meta-learning benchmarks.

A more specific direction for such benchmark design, would be to create a benchmark for image classification where the meta-learner learns a model on simulated data, such that it can transfer on real data. This would be a high utility goal as it would allow training of deep nets on simulated data, which are much cheaper to produce, with the ability to generalize well on their real counterparts. There was at least one paper that produced a simulated to real classification and regression dataset that could be used to design such benchmarks called [Gondal et al., 2019]. Furthermore, meta-learning should also be researched more as a replacement of existing reinforcement learning methods, as that can serve as both a benchmark and a high utility direction as well.

Another important research direction is that of allowing meta-learning to be applied on models unrolling for thousands of inner loop update steps. Currently, most meta-learning models focus on a few steps, on a few-shot task. Extending those steps is not trivial, as gradient-degradation and computational expense gets in the way. To that end, MAML can be augmented with multi step loss to improve gradient issues, but, the computational expense remains a very large issue. Hence, we propose that more research should be done on meta-learning algorithms that utilize evolutionary algorithms for the outer loop [Fernando et al., 2021]. That would be beneficial as it would reduce the need for any gradients and as such make a massive inner loop feasible, while also solving the gradient degradation issues. At the same time, evolutionary algorithms are far more noisy than stochastic gradient based methods, and as such, there would need to be work done on finding the right population sizes, as well as the right type of algorithm that can work well with as many meta-learning problems as possible.

Furthermore, current architecture search methods are too restricted, both in their search space and in their *resolution* of component search. More specifically, searching for an architecture using a number of known layers, and trying to learn blocks to be added in a network, means that you never have the opportunity to discover lower-level components such as new layers, from the bottom up. To this end, we propose learning new architectures by starting from the very basic unit of a dot product, and learning how to connect such dot products together to build the right architecture for a task. This would enable automatic discovery of new layer types as well.

In terms of new tasks, one could consider transfer after modality shift. For example, training a learner that can learn on image data such that the output classifier can perform well on audio data from the same categories as those in the images (e.g. instruments images to sounds, natural images to sounds etc.). This would enable research into meta-learning that can transfer after modality shift. Which could serve as another good benchmark. In an attempt to begin introducing new meta-learning task types, I was involved in designing benchmarks [Antoniou et al., 2020] suitable for evaluating systems capable of continual few-shot learning, wherein a learner is required to learn continually from small tasks composed of only a few samples each.

Inventing better meta-learning algorithms can be a very complex process, but perhaps, one possible solution to automating this step, would be to attempt meta-meta-learning. Perhaps with more layers of learning abstraction, the generalization of the below learners will improve due to the continual transfer dynamics of such a process.

Overall, meta-learning is at its infancy, and there is a lot of work that needs to be done. More immediately, however, if we can extent the number of steps for current meta-learning systems, improve the transfer generalization (i.e. meta-train to meta-test), design better benchmarks and improve architecture search, it is very likely that we will be able to produce much more powerful results with meta-learning.

Bibliography

Meta-learning curiosity algorithms. *ICLR*, 2020.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182, 2016.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6077–6086, 2018.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016.

Antreas Antoniou. How to train your maml: A step by step approach, Nov 2018. URL `https://www.bayeswatch.com/2018/11/30/HTYM/`.

Antreas Antoniou and Amos Storkey. Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation. *arXiv e-prints*, 2019a.

Antreas Antoniou and Amos Storkey. Learning to learn by self-critique. *NeurIPS*, 2019b.

Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv e-prints*, 2017.

Antreas Antoniou, Harrison Edwards, and Amos J. Storkey. How to train your MAML. In *ICLR*, 2018a.

Antreas Antoniou, Agnieszka Słowik, Elliot J Crowley, and Amos Storkey. Dilated densenets for relational reasoning. *arXiv preprint arXiv:1811.00410*, 2018b.

Antreas Antoniou, Massimiliano Patacchiola, Mateusz Ochal, and Amos Storkey. Defining benchmarks for continual few-shot learning. *arXiv e-prints*, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv e-prints*, 2016.

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. Metareg: Towards domain generalization using meta-regularization. In *NeurIPS*, 2018.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. *arXiv preprint arXiv:1709.07417*, 2017a.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In *ICML*, 2017b.

Samy Bengio, Yoshua Bengio, and Jocelyn Cloutier. On the search for new learning rules for ANNs. *Neural Processing Letters*, 1995.

Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning A Synaptic Learning Rule*. 1990.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

David Berthelot, Thomas Schumm, and Luke Metz. BEGAN: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. *arXiv preprint arXiv:1704.05310*, 2017.

Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation – a survey. *arXiv preprint arXiv:1709.01620*, 2017.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Haoye Cai, Chunyan Bai, Yu-Wing Tai, and Chi-Keung Tang. Deep video generation, prediction and completion of human action sequences. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 366–382, 2018.

Jorge Calvo-Zaragoza and David Rizo. End-to-end neural optical music recognition of monophonic scores. *Applied Sciences*, 8(4):606, 2018.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. *arXiv preprint arXiv:1807.05520*, 2018.

Rich Caruana. Learning many related tasks at the same time with backpropagation. In *NeurIPS*, 1995.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *ICLR*, 2019.

Zhiyuan Chen and Bing Liu. Lifelong machine learning, second edition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2018.

Ching-Yao Chuang, Joshua Robinson, Lin Yen-Chen, Antonio Torralba, and Stefanie Jegelka. Debiased contrastive learning. *arXiv preprint arXiv:2007.00224*, 2020.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units. *arXiv e-prints*, 2015.

Elliot J. Crowley, Gavin Gray, and Amos Storkey. Moonshine: Distilling with cheap convolutions. In *Advances in Neural Information Processing Systems*, 2018.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. AutoAugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. AutoAugment: Learning augmentation policies from data. *CVPR*, 2019.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Terrance de Vries, Ishan Misra, Changhan Wang, and Laurens van der Maaten. Does object recognition work for everyone? In *CVPR*, 2019.

Giulia Denevi, Carlo Ciliberto, Dimitris Stamos, and Massimiliano Pontil. Learning to learn around a common mean. In *NeurIPS*. 2018.

Giulia Denevi, Dimitris Stamos, Carlo Ciliberto, and Massimiliano Pontil. Online-within-online meta-learning. In *NeurIPS*, 2019.

Li Deng and John C Platt. Ensemble deep learning for speech recognition. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603. IEEE, 2013.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *arXiv preprint arXiv:1802.07687*, 2018.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Carl Doersch. Tutorial on variational autoencoders. *arXiv e-prints*, 2016.

Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *ICCV*, 2017.

Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference* *on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888. IEEE, 2018.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. In *ArXiv E-prints*, 2016.

Harrison Edwards and Amos Storkey. Towards a neural statistician. In *ICLR*, 2017.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 2019a.

Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning, 2019b.

Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160, 2009.

Chrisantha Fernando, SM Eslami, Jean-Baptiste Alayrac, Piotr Mirowski, Dylan Banarse, and Simon Osindero. Generative art using neural visual grammars and dual encoders. *arXiv preprint arXiv:2105.00162*, 2021.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017a.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017b.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. *ICML*, 2019.

Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. In *ICLR*, 2020.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, 2018.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition And Cooperation In Neural Nets*. 1982.

ken-ichi funahashi and yuichi nakamura. approximation of dynamical systems by continuous time recurrent neural networks. *neural networks*, 6(6):801–806, 1993.

Vikas Garg and Adam T Kalai. Supervising unsupervised learning. In *NeurIPS*, 2018.

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

Spyros Gidaris, Andrei Bursuc, Nikos Komodakis, Patrick Pérez, and Matthieu Cord. Boosting few-shot visual learning with self-supervision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8059–8068, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings Of The Thirteenth International Conference On Artificial Intelligence And Statistics*, 2010.

Muhammad Waleed Gondal, Manuel Wuthrich, Djordje Miladinovic, Francesco Locatello, Martin Breidt, Valentin Volchkov, Joel Akpo, Olivier Bachem, Bernhard Schölkopf, and Stefan Bauer. On the transfer of inductive bias from simulation to the real world: a new disentanglement dataset. In *Advances in Neural Information Processing Systems*, pages 15714–15725, 2019.

Santiago Gonzalez and Risto Miikkulainen. Improved training speed, accuracy, and data utilization through loss function optimization. *arXiv e-prints*, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech And Signal Processing (icassp), 2013 Ieee International Conference On*, 2013.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Yunhui Guo, Noel C. F. Codella, Leonid Karlinsky, John R. Smith, Tajana Rosing, and Rogerio Feris. A new benchmark for evaluation of cross-domain few-shot learning, 2019.

Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

Michael Gygli, Yale Song, and Liangliang Cao. Video2GIF: Automatic generation of animated gifs from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1001–1009, 2016.

David Ha, Andrew Dai, and Quoc V Le. HyperNetworks. *arXiv e-prints*, 2016.

Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv e-prints*, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. 1987.

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *ICANN*, 2001.

Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. CyCADA: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213*, 2017.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv e-prints*, 2020.

MD Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *NeurIPS*, 2018a.

Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems*, pages 5400–5409, 2018b.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetV3. In *ICCV*, 2019.

Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv e-prints*, 2018.

Wei-Ning Hsu, Yu Zhang, and James Glass. Learning latent representations for speech generation and transformation. *arXiv preprint arXiv:1704.04222*, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

Chen Huang, Shuangfei Zhai, Walter Talbott, Miguel Bautista Martin, Shih-Yu Sun, Carlos Guestrin, and Josh Susskind. Addressing the loss-metric mismatch with adaptive loss alignment. In *ICML*, 2019.

Gabriel Huang, Hugo Larochelle, and Simon Lacoste-Julien. Are few-shot learning benchmarks too simple ?, 2020. URL `https://openreview.net/forum?id=SygeY1SYvr`.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 1962.

Marcus Hutter. Hutter prize: Human knowledge compression contest. URL `http://prize.hutter1.net/`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv e-prints*, 2015.

Touseef Iqbal and Shaima Qureshi. The survey: Text generation models in deep learning. *Journal of King Saud University-Computer and Information Sciences*, 2020.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 2019.

Muhammad Abdullah Jamal, Guo-Jun Qi, and Mubarak Shah. Task-agnostic meta-learning for few-shot learning. *arXiv preprint arXiv:1805.07722*, 2018.

Khurram Javed and Martha White. Meta-learning representations for continual learning. In *NeurIPS*, 2019.

Yibo Jiang and Nakul Verma. Meta-learning to cluster, 2019.

Justin Johnson, Andrej Karpathy, and Li Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4565–4574, 2016.

Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

Siavash Khodadadeh, Ladislau Boloni, and Mubarak Shah. Unsupervised meta-learning for few-shot image classification. In *NeurIPS*, 2019.

Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Young-duck Choi, Yongseok Choi, Dong-Yeon Cho, and Jiwon Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*, 2018a.

Tae Hyun Kim, Mehdi SM Sajjadi, Michael Hirsch, and Bernhard Schölkopf. Spatio-temporal transformer network for video restoration. In *European Conference on Computer Vision*, pages 111–127. Springer, 2018b.

Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1857–1865. JMLR. org, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv e-prints*, 2014.

Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012a.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012b.

Seong Min Kye, Hae Beom Lee, Hoirin Kim, and Sung Ju Hwang. Meta-learned confidence for few-shot learning. *arXiv preprint arXiv:2002.12017*, 2020.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. Zero-data learning of new tasks. In *AAAI*, 2008.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. FractalNet: Ultra-deep neural networks without residuals. *arXiv e-prints*, 2016.

Quoc V Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten Zip code recognition. *Neural computation*, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Deeper, broader and artier domain generalization. In *ICCV*, 2017a.

Ke Li and Jitendra Malik. Learning to optimize. In *ICLR*, 2017.

Yiying Li, Yongxin Yang, Wei Zhou, and Timothy M. Hospedales. Feature-critic networks for heterogeneous domain generalization. In *ICML*, 2019.

Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M. Robertson, and Yongxing Yang. DADA: Differentiable automatic data augmentation, 2020.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-SGD: Learning to learn quickly for few-shot learning. In *ArXiv E-prints*, 2017b.

Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *ICLR*, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv e-prints*, 2015.

Xingyu Lin, Harjatin Baweja, George Kantor, and David Held. Adaptive auxiliary task weighting for reinforcement learning. In *NeurIPS*. 2019.

Zhen-Hua Ling, Shi-Yin Kang, Heiga Zen, Andrew Senior, Mike Schuster, Xiao-Jun Qian, Helen M Meng, and Li Deng. Deep learning for acoustic modeling in parametric speech generation: A systematic review of existing techniques and future trends. *IEEE Signal Processing Magazine*, 32(3):35–52, 2015.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019a.

Shikun Liu, Andrew Davison, and Edward Johns. Self-supervised Generalisation With Meta Auxiliary Learning. In *NeurIPS*, 2019b.

Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, and Yi Yang. Transductive propagation network for few-shot learning. *arXiv preprint arXiv:1805.10002*, 2018.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv e-prints*, 2017.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference On*, 2017.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The Penn treebank. *Computational linguistics*, 1993.

Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. *ICLR*, 2019.

Thomas Miconi, Jeff Clune, and Kenneth O Stanley. Differentiable plasticity: Training plastic neural networks with backpropagation. In *ICML*, 2018.

Thomas Miconi, Aditya Rawal, Jeff Clune, and Kenneth O. Stanley. Backpropamine: Training self-modifying neural networks with differentiable neuromodulated plasticity. In *ICLR*, 2019.

Tomáš Mikolov, Martin Karafi át, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference Of The International Speech Communication Association*, 2010.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Abdelrahman Mohamed, Dmytro Okhonko, and Luke Zettlemoyer. Transformers with convolutional context for asr. *arXiv preprint arXiv:1904.11660*, 2019.

Tsendsuren Munkhdalai and Hong Yu. Meta networks. *arXiv preprint arXiv:1703.00837*, 2017a.

Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *ICML*, 2017b.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018a.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. In *ArXiv E-prints*, 2018b.

Kuniaki Noda, Yuki Yamaguchi, Kazuhiro Nakadai, Hiroshi G Okuno, and Tetsuya Ogata. Audio-visual speech recognition using deep learning. *Applied Intelligence*, 42(4):722–737, 2015.

Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. TADAM: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 719–729, 2018.

Alexander Pacha and Horst Eidenberger. Towards a universal music symbol classifier. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 35–36. IEEE, 2017.

Eunbyung Park and Junier B. Oliva. Meta-curvature. In *NeurIPS*, 2019.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, 2015.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

Quoc V. Le Prajit Ramachandran, Barret Zoph. Searching for activation functions. In *ArXiv E-prints*, 2017.

Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L. Yuille. Few-shot image recognition by predicting parameters from activations. *CVPR*, 2018a.

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7229–7238, 2018b.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *NeurIPS*, 2019.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NeurIPS*, 2017.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

Mengye Ren, Renjie Liao, Ethan Fetaya, and Richard Zemel. Incremental few-shot learning with attention attractor networks. In *NeurIPS*, 2019.

Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7008–7024, 2017.

Alexander Ilin Rinu Boney. Semi-supervised few-shot learning with maml. *ICLR2018, Meta-Learning Workshop*, 2018.

Eduardo Romera, José M. Álvarez, Luis M. Bergasa, and Roberto Arroyo. Efficient convnet for real-time semantic segmentation. *IEEE Intelligent Vehicles Symposium*, 2017a.

Eduardo Romera, José M. Álvarez, Luis M. Bergasa, and Roberto Arroyo. ERFNet: Efficient residual factorized ConvNet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 2017b.

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *ICLR*, 2019.

Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference Of The International Speech Communication Association*, 2014.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv e-prints*, 2016.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?(no, it is not about internal covariate shift). *arXiv e-prints*, 2018.

Juergen Schmidhuber, Jieyu Zhao, and MA Wiering. Simple principles of meta-learning. *Technical report IDSIA*, 1996.

Jurgen Schmidhuber. Evolutionary principles in self-referential learning. *On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1987.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 1992.

Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *Neural Networks, 1993., IEEE International Conference On*, 1993.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Jürgen Schmidhuber, Jieyu Zhao, and Marco Wiering. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 1997.

Jürgen Schmidhuber, Jieyu Zhao, and Nicol N Schraudolph. Reinforcement learning with self-modifying policies. In *Learning to learn*, pages 293–309. Springer, 1998.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. *Neural Networks*, 2003.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv e-prints*, 2013.

Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *NeurIPS*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv e-prints*, 2014.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. In *ICLR*, 2020.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017a.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017b.

Jose Sotelo, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2wav: End-to-end speech synthesis. 2017.

Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 2019.

Jan Storck, Sepp Hochreiter, and Jürgen Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the international conference on artificial neural networks, Paris*, volume 2, pages 159–164. Citeseer, 1995.

Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *CVPR*, 2018.

Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv e-prints*, 2017a.

Flood Sung, Li Zhang, Tao Xiang, Timothy M. Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *CoRR*, 2017b.

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. In *CVPR*, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-V4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning To Learn*. 1998.

Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? *arXiv preprint arXiv:2005.10243*, 2020.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *ICLR*, 2020.

Hung-Yu Tseng, Hsin-Ying Lee, Jia-Bin Huang, and Ming-Hsuan Yang. Cross-domain few-shot classification via learned feature-wise transformation. *ICLR*, January 2020.

Lukas Tuggener, Ismail Elezi, Jurgen Schmidhuber, and Thilo Stadelmann. Deep watershed detector for music object recognition. *arXiv preprint arXiv:1805.10548*, 2018.

Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. MocoGAN: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The missing ingredient for fast stylization. *arXiv e-prints*, 2016.

J urgen Schmidhuber. Adaptive confidence and adaptive curiosity. Technical report, Citeseer.

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. In *SSW*, 2016.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

Eelco van der Wel and Karen Ullrich. Optical music recognition with convolutional sequence-to-sequence models. *arXiv preprint arXiv:1707.04877*, 2017.

Vladimir Vapnik. 24 transductive inference and semi-supervised learning. 2006.

C. v.d. Malsburg. Technical report 81-2. 1981.

Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Richard Lewis, Janarthanan Rajendran, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks. In *NeurIPS*, 2019.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 2002.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. 2008.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, 2016.

Riccardo Volpi and Vittorio Murino. Model vulnerability to distributional shifts over image transformation sets. In *ICCV*, 2019.

Carl Vondrick and Antonio Torralba. Generating the future with adversarial transformers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1020–1028, 2017.

Risto Vuorio, Dong-Yeon Cho, Daejoong Kim, and Jiwon Kim. Meta continual learning. *arXiv e-prints*, 2018.

Yan Wang, Wei-Lun Chao, Kilian Q. Weinberger, and Laurens van der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning, 2019.

Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. *arXiv preprint arXiv:1703.04813*, 2017a.

Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *ICML*, 2017b.

MA Wiering and Jürgen Schmidhuber. Learning exploration policies with models. 1998a.

Marco Wiering and Jürgen Schmidhuber. Efficient model-based exploration. 1998b.

Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? *arXiv preprint arXiv:1609.08764*, 2016.

Zhizheng Wu, Cassia Valentini-Botinhao, Oliver Watts, and Simon King. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *Acoustics, Speech And Signal Processing (ICASSP), 2015 IEEE International Conference On*, 2015a.

Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 461–470, 2015b.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *ICLR*, 2019.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv e-prints*, 2015.

Tianbing Xu, Qiang Liu, Liang Zhao, and Jian Peng. Learning to explore with meta-policy gradient. *ICML*, 2018a.

Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *NeurIPS*, 2018b.

Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. Boosting image captioning with attributes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4894–4902, 2017.

Seungjin Choi Yoonho Lee. Gradient-based meta-learning with learned layerwise metric and subspace. In *ICML*, 2018.

Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4651–4659, 2016.

Kenny Young, Baoxiang Wang, and Matthew E. Taylor. Metatrace actor-critic: Online step-size tuning by meta-gradient descent for reinforcement learning control. In *IJCAI*, 2019.

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *arXiv e-prints*, 2017.

A Steven Younger, Sepp Hochreiter, and Peter R Conwell. Meta-learning with backpropagation. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference On*, 2001.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations*, 2016.

Yang Yu. Towards sample efficient reinforcement learning. 2018.

Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

Marvin Mengxin Zhang, Henrik Marklund, Nikita Dhawan, Abhishek Gupta, Sergey Levine, and Chelsea Finn. Adaptive risk minimization: A meta-learning approach for tackling group shift. 2020.

Xinyu Zhang, Jiewei Cao, Chunhua Shen, and Mingyu You. Self-training with progressive augmentation for unsupervised cross-domain person re-identification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8222–8231, 2019.

Ying Zhang, Mohammad Pezeshki, Philémon Brakel, Saizheng Zhang, Cesar Laurent Yoshua Bengio, and Aaron Courville. Towards end-to-end speech recognition with deep convolutional neural networks. *arXiv e-prints*, 2017.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *NeurIPS*, 2018.

Shiyu Zhou, Linhao Dong, Shuang Xu, and Bo Xu. Syllable-based sequence-to-sequence speech recognition with the transformer in mandarin chinese. *arXiv preprint arXiv:1804.10752*, 2018.

Wei Zhou, Yiying Li, Yongxin Yang, Huaimin Wang, and Timothy M. Hospedales. Online meta-critic learning for off-policy actor-critic methods, 2020.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv e-prints*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.