POLITECNICO
MILANO 1863

POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

STRUCTURED META-LEARNING FOR CROSS-DOMAIN FEW-SHOT CLASSIFICATION

A study on structured representations and their effectiveness when dealing with tasks from heterogeneous domains

Tesi di Laurea di:
NICOLA DE ANGELI
Matricola 914693

Relatore: PROF.MATTEO MATTEUCCI
Correlatore: DR. MARCO CICCONE
Correlatore: PROF. BARBARA CAPUTO

Anno Accademico 2019-2020

Nicola De Angeli: *Structured Meta-Learning for Cross-Domain Few-Shot Classification,* A study on structured representations and their effectiveness when dealing with tasks from heterogeneous domains

Machines take me by surprise with great frequency.

— Alan M. Turing

SOMMARIO

Nell'ambito del Few-Shot Learning (FSL) viene richiesto ai modelli di imparare nuovi task sulla base di pochi esempi. Il Cross-Domain Few-Shot Learning (CDFSL) definisce un problema ancora più difficile, imponendo anche che i task di test siano soggetti ad uno shift di dominio. Nonostante gli approcci sempre più numerosi proposti di recente nel campo, il problema di come meta-imparare efficacemente su molti domini di training e al tempo stesso evitare meta-overfitting resta un problema interessante e non banale. In quest'ottica, proponiamo Corrupted-Omniglot, un nuovo benchmark per CDFSL. Inoltre, presentiamo e analizziamo molteplici tecniche che si basano su disentanglement, agnosticismo di dominio e statistiche di batch normalization di alta qualità per affrontare il problema di CDFSL. Abbiamo riscontrato che grandi quantità di pre-training e l'uso di informazione di dominio durante la classificazione peggiora sensibilmente la performance. Più in generale, abbiamo scoperto che molti modi di estendere le architetture state-of-the-art in FSL per tenere conto della presenza di molteplici domini di training e test non ha impatti positivi sulla performance in CDFSL. Ciononostante, reputiamo che agnosticismo di dominio e statistiche di batch normalization di alta qualità siano comunque delle direzioni di ricerca ancora da esplorare opportunamente nel campo di CDFSL.

ABSTRACT

In Few-Shot Learning (FSL), models are challenged to learn new tasks based on few examples. Cross-Domain Few-Shot Learning (CDFSL) takes the FSL problem one step further, imposing test tasks to be subject to domain shift. Despite increasing efforts by recent works in the field, the problem of how to effectively meta-learn across multiple training domains while avoiding meta-overfitting remains an important challenge. To this end, we propose Corrupted-Omniglot, a novel CDFSL classification benchmark. Furthermore, we present and analyze multiple techniques that rely on disentanglement, domain agnosticism, and high-quality batch normalization statistics to tackle the CDFSL problem. We discovered that large amounts of pre-training and the usage of domain information during classification significantly worsen performance. More generally, we found that many ways of extending state-of-the-art FSL architectures to address the presence of multiple training and test domains fails to boost performance in CDFSL. Nonetheless, we believe that domain agnosticism and high-quality batch normalization statistics still represent two promising research directions that are not yet sufficiently explored in CDFSL.

ACKNOWLEDGMENTS

I would like to thank my institution, Politecnico di Milano, for the opportunity to study in one of Italy's most prestigious universities and for financial support.

I wish to show my appreciation to my advisor, Matteo Matteucci, for allowing me to work on the challenging and exciting problem of Meta-Learning, for his useful contribution to the topics discussed in this thesis, and for his constructive critiques.

I am deeply grateful to my tutor and co-advisor, Marco Ciccone, for his helpful guidance throughout my work and for teaching me important lessons on how to conduct scientific research in computer science.

I would also like to thank my other co-advisor, Barbara Caputo, for the opportunity to work on this project in collaboration with researchers from Politecnico di Torino.

I am also grateful to Dr. Luigi Malagò and Dr. Jonathan Masci for often providing me with precious insights that helped to carry this project forward.

Finally, I would like to thank my family for supporting me during the writing of this thesis and throughout these years of study. Special thanks go to my grandmother, Carla, for her enduring support and endorsement.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ERM	Empirical Risk Minimization
NN	Neural Network
FNN	Feedforward Neural Network
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
FSL	Few-Shot Learning
CDFSL	Cross-Domain Few-Shot Learning

1 INTRODUCTION

Machine Learning is an application of Artificial Intelligence whose goal is to develop methods that can automatically detect patterns in data, and then to use the uncovered patterns to predict future data or other outcomes of interest [102]. In recent years, a class of parametric machine learning techniques called *Deep Learning* has led to astonishing achievements in the field. The key aspect of Deep Learning is the ability to learn how to automatically extract relevant features from data through the employment of many neural layers, thus not requiring field-specific expertise [45]. As a consequence, the leverage of these techniques is very approachable and results in lower costs, promoting the development of artificial intelligence applications outside the academic community.

Nonetheless, Deep Learning currently faces some obstacles that still hinder the technology to be fully exploited. Humans have a remarkable capacity to quickly learn new concepts when provided with few examples. Conversely, current popular deep learning techniques need thousand of samples to be able to generalize their knowledge and make predictions on unseen data, making them extremely data inefficient. In the context of Supervised Learning, this often translates into the need to manually label thousands of samples, which is cumbersome and time-consuming. Similarly, in Reinforcement Learning, this translates into having access to a large number of training trajectories and this may be unfeasible when the experience is directly observed from real-world interactions.

Meta-Learning, also known as *"learning-to-learn"*, is a sub-field of Machine Learning that exploits previous experience to optimize learning algorithms to work well on novel tasks [41]. The experience is often formalized as a collection of tasks, upon which meta-learning techniques build general, task-agnostic knowledge that can be reused. The approach has been shown to address some of the challenges posed by Few-Shot Learning (FSL), where very few task-specific training datapoints are available and the problem of overfitting is particularly insidious [84]. The literature has recently provided promising results also thanks to the leverage of deep learning techniques, achieving human-like performance also in simple meta-learning tasks [76].

1.1 OPEN PROBLEMS IN META-LEARNING

In spite of its recent achievements, Meta-Learning currently faces many challenges. Many popular approaches struggle when scaling to more powerful learners, obtaining poor performance when dealing with complex tasks. Another challenge is the transfer of knowledge among tasks that are particularly different. Our brain builds powerful abstractions that can be used to identify an object, no matter how it is depicted, either as a natural image, a clip-art, or another visual representation. Conversely, a problem that has been observed to occur in many state-of-the-art meta-learning approaches is the inability to

generalize the knowledge when presented with a heterogeneous distribution of domains.

As also analyzed by Triantafillou et al. [138], we identified three partially overlapping issues that should be addressed to scale meta-learning. Firstly, most of the current meta-learning algorithms are designed considering the simplistic assumption that the distribution of tasks is *homogeneous*, namely that tasks are coming from a single source [76], and they share the same characteristics. In contrast, real-life learning experiences are *heterogeneous*: for instance, classification tasks may vary in terms of the number of classes or examples per class and are often unbalanced. Secondly, benchmarks in Meta-Learning only measure within-dataset generalization. However, we are interested in having models that can learn from multiple sources and generalize to entirely new distributions, namely new datasets or domains. Lastly, most of the current models and benchmarks ignore the relationships between tasks and classes, disregarding structures that could be useful to share knowledge across multiple tasks.

1.2 RESEARCH QUESTIONS AND MAIN CONTRIBUTIONS

In light of these issues, our goal is to determine whether the currently provided techniques can be extended to better scale Meta-Learning with respect to data and task heterogeneity, a problem known as Cross-Domain Few-Shot Learning (CDFSL). In particular, we attempt to tackle the problem by focusing on the quality of the embedding in our models. A model capable of operating among different data domains would be able to transfer knowledge among widely different tasks, solving the lack of training samples that is observed in certain data domains. As a practical example, the desired model would be able to generalize the recognition of malignant tumors in x-ray images to images obtained through other less popular techniques or instruments that may feature different colors and shades.

Our contribution to the problem is manifold. First of all, we propose *Corrupted-Omniglot*, a novel CDFSL benchmark, which is obtained by augmenting images from the Omniglot dataset [76] with images corruptions provided by Mu and Gilmer [99]. Furthermore, we find multiple ways to extend *Latent Embedding Optimization* (LEO) [124], a popular FSL algorithm, to address the presence of multiple domains in our problem. Initially, we examine the effects of leveraging a disentangled and interpretable embedding by pre-training a *Domain-Invariant Variational Autoencoder* (DIVA) [61] on images from Corrupted-Omniglot and subsequently reloading the obtained embedder into the LEO architecture. We also pursue disentanglement directly during meta-learning by considering a novel architecture capable of learning from both class and domain discrimination tasks. In virtue of the results obtained, we shift the focus to producing a domain agnostic embedding by leveraging gradient reversal [43]. Finally, we attempt to leverage Weighted Batch Normalization [92] to obtain reliable domain-specific statistics and a normalization procedure that can scale to unseen domains.

We discovered that large amounts of pre-training and the usage of domain information during classification significantly worsen performance. More generally, we found that many ways of extending state-of-the-art FSL architectures to address the presence of multiple training and test domains fails to boost per-

formance in CDFSL. Nonetheless, we believe that domain agnosticism and high-quality batch normalization statistics still represent two promising research directions that are not yet sufficiently explored in CDFSL.

1.3 THESIS STRUCTURE

We provide an ordered list of the chapters that constitute the rest of the thesis along with a brief description of their content.

- Chapter 2 provides an overview of the main concepts of machine learning and deep learning that are also relevant to our specific setting;
- Chapter 3 introduces Meta-Learning, providing a mathematical formalization of the problem, a taxonomy of the literature, and comparison with related fields;
- Chapter 4 examines recent works in various fields of the machine learning literature that address the presence of multiple domains in our data, such as Variational Autoencoders, Domain Adaptation, Domain Generalization, and CDFSL;
- Chapter 5 provides a complete formulation of the CDFSL problem we want to tackle and describes our first approach to the problem, which is based on pre-training a high-quality embedding;
- Chapter 6 builds upon the results obtained in the previous experiments by meta-learning the embedding from scratch and examining other interesting research directions;
- Chapter 7 briefly reports the CDFSL problem we tackled, summarizes our contributions and findings, and suggests promising future lines of work in the field based on the results we obtained.

2 MACHINE LEARNING AND DEEP LEARNING

This chapter gives a brief and self-contained introduction on the main machine learning and deep learning tools used throughout this thesis to provide the reader with the necessary background to understand the presented concepts. The resources in this chapter are adapted from the comprehensive Deep Learning textbook [45].

2.1 LEARNING FROM EXPERIENCE

Machine Learning is the study of computer algorithms that improve automatically through experience:

"A computer program is said to learn from experience E *with respect to some class of tasks* T *and performance measure P, if its performance at tasks in* T*, as measured by* P*, improves with experience* E*."*

This informal definition from Mitchell [95] can result in a variety of possibilities for the experience E, the task T, and the performance metric P, depending on the operative framework that we might want to use to build machine learning algorithms.

2.1.1 *Types of Task*

Machine Learning helps us by tackling problems that are too difficult to be solved by designing programs with a set of predefined rules. In other words, a task is a problem that we want to solve, and the process of learning the task is our means of attaining the ability to solve it. More naively, we expect that a machine learning algorithm would develop an understanding of the task to be solved by interacting with collected data and updating its knowledge of it.

Usually, a machine learning algorithm is described in terms of its ability to process an example as input. Input samples are defined, for instance, as vectors of *features* $x \in \mathbb{R}^n$. The literature provides multiple types of tasks, each one considering a different kind of mapping. Two popular examples are classification and regression.

- *Classification*: the algorithm is asked to assign a class n among a set of possible categories to a given input. This is usually achieved by learning a mapping in the form of $f : \mathbb{R}^m \to \{1, \dots, N\}$.
- *Regression*: the algorithm is asked to predict a real value, given some input. The task can be solved by learning the input-output mapping as a function $f : \mathbb{R}^m \to \mathbb{R}$.

Depending on the area of application, we can define many other types of tasks to effectively abstract the problem. In this thesis, our main focus is on classification tasks.

2.1.2 *Performance Measure*

It is necessary to define a metric to evaluate a machine learning algorithm's ability to perform on a specific task T. The performance measure is generally defined as a numerical value computed on a subset of data by comparing the prediction output of the algorithm and its ground truth value. We are generally interested in measuring the quality of the algorithm on data never observed during the learning process to analyze its generalization capability.

Unfortunately, direct optimization of the performance measure we are interested in is not always feasible. Therefore, it is often necessary to find related proxies that are easier to compute or optimize to allow the algorithm to effectively learn the task at hand. In the case of classification, the performance measure of choice usually is accuracy, with cross-entropy as a popular proxy.

2.1.3 *Experience*

The experience E is defined as the data that a machine learning algorithm can process to learn how to solve the task. Machine Learning algorithms are generally categorized under three main learning paradigms based on what kind of experience they have access to during the learning process. We define a *dataset* as the entire collection of experience, and the examples that form the dataset as *datapoints*.

- *Supervised Learning* algorithms observe a dataset where each sample x is paired with a *label* or *target* y provided from a knowledgeable external supervisor. The learning task is often formalized as estimating $p(y|x)$ by learning a mapping function from x to y.
- *Unsupervised Learning* algorithms have access to a dataset that provides only examples x and learn to extract hidden patterns in the unlabeled data by clustering together similar examples. Usually, we are interested in learning the generative model of the dataset $p(x)$.
- *Reinforcement Learning* algorithms aim to learn the optimal policy of an agent by interacting sequentially with an environment and observing a reward signal. The agent's goal is to maximize the total reward it receives over the long run by performing actions in each state he visits.

In this thesis, we primarily focus solely on the supervised learning setting.

2.2 SUPERVISED LEARNING

So far, we only gave an intuitive explanation of a machine learning algorithm and its components without providing any operative description. In this section, we focus on supervised learning, and we formalize it in a principled way using the Empirical Risk Minimization framework [141, 142].

We consider an *input space* $\mathcal{X}$ and an *output* (or *target*) *space* $\mathcal{Y}$, where typically $\mathcal{X} \subseteq \mathbb{R}^d$. The form of the output space yields different kinds of tasks: regression $\mathcal{Y} \subseteq \mathbb{R}$, binary classification $\mathcal{Y} = \{-1; +1\}$, multi-class classification $\mathcal{Y} = \{1, \dots, K\}$. The pair of random variables (x, y) is distributed according to

the unknown joint probability distribution $p(x, y)$, defined over the *data space* $\mathcal{X} \times \mathcal{Y}$.

The *training dataset* $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ is made of n identically and independently distributed (i.i.d.) input-target pairs, that we assume to be representative of the data distribution $p(x, y)$. Supervised Learning can be formalized as the process of finding the input-output mapping $f : \mathcal{X} \to \mathcal{Y}$ based on the training data $\mathcal{D}_n$ to predict via $f(x)$ the output corresponding to any new input $x \in \mathcal{X}$. This is usually achieved by searching for a good approximation of the true mapping over a class of candidate functions $\mathcal{H}$ called *hypothesis space*. More precisely, we can formulate an optimization problem:

$$\mathcal{R}^* = \min_{f \in \mathcal{H}} \quad \mathcal{R}(f), \tag{1}$$

where $\mathcal{H}$ can be any class of functions such as *linear functions*, *radial basis functions* or *deep neural networks*, and $\mathcal{R}$ is a *"risk"* functional that depends on the training data. For any fixed *loss function* $\ell : \mathcal{Y} : \mathcal{Y} \to \mathbb{R}$, the risk is defined as the expected loss over the data distribution:

$$\mathcal{R}(f) = \mathbb{E}_{p(x,y)}[\ell(y, f(x))] = \int\int \ell(y, f(x)) p(x, y) dx dy \tag{2}$$

The loss function is a point-wise measure of the error $\ell(y, f(x))$ made by predicting $f(x)$ with target value y, while the risk is a measure of the average performance of the machine learning algorithm on the task.

2.3 EMPIRICAL RISK MINIMIZATION

Given a fixed loss function $\ell(\cdot, \cdot)$, the *true* risk minimization is generally non-trivial because the underlying data distribution in unknown, making the expectation in Equation (2) non-computable and the optimization problem in Equation (1) intractable.

Instead of minimizing $\mathcal{R}(f)$ directly, one may replace the true data distribution $p(x, y)$ by its empirical distribution computed using the training dataset $\mathcal{D}_n$, and obtain the following minimization problem:

$$\hat{\mathcal{R}}_n = \min_{f \in \mathcal{H}} \quad \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)), \tag{3}$$

which is called *Empirical Risk Minimization* (ERM) [141]. Because the data distribution is unknown, instead of minimizing the risk directly, ERM uses a sample-based estimate to optimize the true risk. In particular, it can be shown that, under some conditions, ERM is *statistically consistent*, meaning that, more data is collected, the empirical risk converges in probability to its true value, and ERM corresponds to minimizing the true risk [130, 142].

The induction principle of empirical risk minimization is quite general and encompasses many popular learning methods. For instance, restricting the space of hypothesis to $\mathcal{H} = \{f(x) = \theta^T x \quad \forall \theta \in \mathbb{R}^d\}$ and taking a square loss function $\ell(y, f(x)) = (y - f(x))^2$, corresponds to the well known *least squares* estimation. *Maximum likelihood estimation (MLE)* [12, 101] is a also a special case of ERM where the loss function is the *negative log-likelihood*:

$$\theta_{MLE} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} p(y|x, \theta), \tag{4}$$

2.4 MODEL COMPLEXITY: UNDERFITTING VS OVERFITTING

Although under certain conditions the empirical risk corresponds to the true risk when the sample size n grows, in practice, the number of samples is always finite. As a result, the actual predictor might not minimize the actual risk, especially when the space of hypothesis $\mathcal{H}$ is large, and the number of samples is limited. When we have a small empirical risk but still a relatively large true risk, we say that the algorithm is *overfitting* the training data. A good learning algorithm should provide a similar behavior to the target function and perform well on new, previously unseen data, i. e., never experienced in the training set $\mathcal{D}_n$. In this case, we say that the algorithm *generalizes* well. We typically estimate the *generalization error* of a predictor by measuring the performance on a *test set* of examples that were separately collected from the training set, under the assumption of being identically distributed.

2.4.1 *Approximation and Estimation Errors*

By considering the ERM hypothesis of a predictor $\hat{f}_n$, namely a function in $\mathcal{H}$ that minimizes the empirical risk, and the best predictor among $\mathcal{H}$

$$f^* = \arg\min_{f \in \mathcal{H}} \quad \mathcal{R}(f), \tag{5}$$

we can define the *excess risk* of $\hat{f}_n$ and decompose the error of an ERM predictor in two components as follows:

$$\mathcal{R}(\hat{f}_n) - \mathcal{R}^* = \underbrace{(\mathcal{R}(f^*) - \mathcal{R}^*)}_{\text{approximation error}} + \underbrace{\left(\mathcal{R}(\hat{f}_n) - \mathcal{R}(f^*)\right)}_{\text{estimation error}}. \tag{6}$$

The *approximation error* measures the risk caused by the restriction to a specific class of hypothesis $\mathcal{H}$, also called *inductive bias*. The approximation error is deterministic and does not depend on the sample size. It can be reduced by extending the hypothesis class to other functions. The *estimation error* represents how much we are losing in terms of risk by using a finite sample approximation instead of using the true data distribution, and it gives a measure of the quality of the training set.

Similarly to the *bias-variance trade-off* for standard statistical estimation problems [90], there is also a tension between the approximation error and estimation error. The approximation error term acts as a bias square term while the estimation error behaves like the variance term. Choosing a very rich class of candidate functions $\mathcal{H}$ decreases the approximation error at the cost of the estimation error, but it might lead to *overfitting*. On the other hand, restricting $\mathcal{H}$ reduces the estimation error but might increase the approximation error because the predictor might not be expressive enough to represent the true mapping leading to *underfitting*.

2.4.2 *Regularization Techniques*

Overfitting can be mitigated by following Occam's razor principle [30] of *"parsimony of explanation"*, which states that if two models explain data equally one

should choose the simplest one, based on some notion of complexity. Model regularization is usually achieved by adding a penalization term $\lambda R(\theta)$ to the objective function that encourages the learning of simpler models over complex ones, where λ is a coefficient that regulates the level of regularization $R(\theta)$ function of the model's parameters. In the context of ERM, adding regularization terms that penalize model complexity is usually referred to as *Structural Risk Minimization*. From a Bayesian perspective, adding a regularization term can be interpreted as imposing a prior over the parameters of the model. For example, a penalty in the form of $\lambda R(\theta) = \theta^T\theta = \|\theta\|$, also called *weight decay* or simply L2 regularization corresponds to assuming that the parameters are normally distributed with zero mean [12].

2.5 STOCHASTIC GRADIENT DESCENT

Gradient descent is an iterative optimization algorithm that improves the problem's solution by taking a step in the direction of the negative of the gradient of the function to be minimized at the current point.

When the class of candidate functions and the loss function are differentiable, the risk functional is also differentiable and the minimization can be efficiently solved, albeit approximately, by applying numerical optimization techniques such as *Stochastic Gradient Descent* (SGD) [120]. When the exact gradient is not available, SGD circumvents this problem by allowing the optimization procedure to take a step along a noisy direction, as long as the expected value of the direction is the negative of the gradient. SGD provides a method to optimize directly the risk functional since the gradient of the loss function on a randomly sampled example is an unbiased estimate of the gradient of the risk.

$$\mathbb{E}_{p(x,y)}[\nabla \ell(y, f(x))] = \nabla \mathbb{E}_{p(x,y)}[\ell(y, f(x))] = \nabla \mathcal{R}(f) \tag{7}$$

Given a dataset of examples, training a model via gradient descent simply consists in repeatedly sampling data points and applying the following update rule at each i-th iteration to modify the parameters of the model θ in the direction that minimizes the loss:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla \ell(\theta^{(i)}) \tag{8}$$

where $\theta^{(0)}$ is randomly initialized, η is the learning step, $\ell : \mathbb{R}^n \to \mathbb{R}$ and the vector of partial derivatives is $\nabla \ell(\theta) = \left(\frac{\partial \ell(\theta)}{\partial \theta^1}, \ldots, \frac{\partial \ell(\theta)}{\partial \theta^n}\right)$.

Combined with automatic differentiation and recent advances in stochastic optimization [32, 53, 67, 105, 154] this provides a turnkey approach to fitting modern differentiable machine learning models such as deep neural networks (see Section 2.6.5 for more details).

2.6 NEURAL NETWORKS

A *Neural Network* (NN) is a popular machine learning model that can capture input-output relationships and complex patterns in the data. Thanks to their expressivity, NNs can be employed in many machine learning problems in supervised, unsupervised, or reinforcement learning [12, 134] settings.

The ancestors of early NNs (e. g., Perceptrons [121], Neucognitron [42]) were loosely inspired by models of information processing in human and mammals brains but soon drifted from their initial biological inspiration. Since their introduction, these models have received moderate interest from the scientific community and are considered the cornerstones of modern neural networks. Nevertheless, research on neural networks stagnated for many decades before the recent deep learning revolution. One of the reasons for the initial loss of interest in neural networks was the lack of an efficient algorithm for training complex models. Before the discovery of the *backprogation algorithm*, NNs were considered intractable models. For a detailed survey on the history of neural networks, from the first *connectionist models* to the recent developments, we refer to Schmidhuber [128]. In the rest of the chapter, we introduce the basic concepts of modern deep learning architectures preparatory for understanding the thesis.

2.6.1 *Terminology*

A Neural Network can be described as a system of basic processing units, the *neurons*, that are connected by weighted directed edges. The information flows through the computational graph, with each neuron processing its input to produce an output, also called *activation*. Connections between neurons, known as *weights*, are gradually adjusted during the learning process and determine the contribution to the output of the signals generated by the source neuron. The set of units in the graph are typically organized into sequences of *layers*; the number of units in a layer is referred to as its *width* and the total number of processing layers indicates the *depth* of the neural network. The set of neurons that processes the source information is usually referred to as *input layer*, while the ones that produce the prediction are called *output layer*. All the other layers are generally called *hidden layers*.

A Neural Network with *acyclic* computational graph is named *Feedforward Neural Network* (FNN) because the information flows only in one direction. Other types of neural networks such as recurrent neural networks are not considered in this work.

The flexibility of neural networks makes them a perfect family of candidate functions for risk minimization. Feedforward Neural Networks are known to be *universal function approximators*, i. e., a feedforward neural network with a single hidden layer can approximate any measurable function to any desired degree of accuracy on a compact set [23, 56, 57]. Although FNNs with few hidden layers can potentially represent any function, distributing the computation across multiple layers can be exponentially more efficient for some class of functions [27, 96]. Based on this observation, modern neural architectures build hierarchical data abstractions growing models in depth, improving model efficiency at the expense of trainability.

2.6.2 *Types of Layers and Networks*

We now present an overview of the types of layers and neural networks that are used in this thesis. To keep the notation tidy, the computational graph of

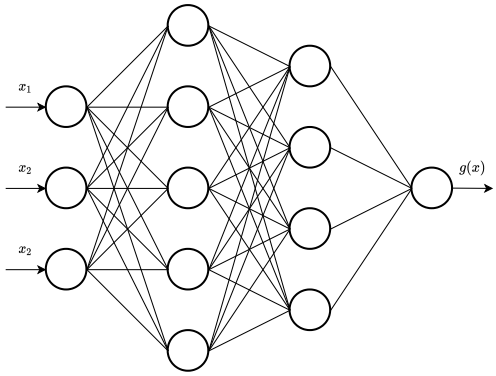


Figure 1: A Multilayer Perceptron.

each layer is described in terms of matrix multiplications rather than single-unit interactions.

2.6.2.1 *Multilayer Perceptron*

A *Multilayer Perceptron* (MLP), is the simplest kind of neural network and it consists of a series of dense layers where each neuron is connected to all of its input, called *fully connected* layers. An MLP can be represented by a function $g : \mathbb{R}^d \to \mathbb{R}^m$ that maps from the input to the output space via a composition of L layers. Each fully connected layer, $\ell \in \{1, \dots L\}$, involves first an affine transformation of the input, parametrized by a matrix of weights W_ℓ and a bias vector b_ℓ, followed by a non-linear *activation* function $f(\cdot)$, that warps the hidden space. The input x_ℓ of an hidden layer is the output of its predecessor y_ℓ, and each layer is applied in chain until the final output $\hat{y} = y_L$ is produced:

$$x_\ell = y_{\ell-1} \tag{9a}$$
$$z_\ell = W_\ell x_\ell + b_\ell \tag{9b}$$
$$y_\ell = f(z_\ell) \tag{9c}$$

The activation functions employed in MLPs are generally point-wise nonlinearities such as *hyperbolic tangent* $f(x) = \tanh(x)$ or logistic sigmoid $f(x) = \frac{1}{1+\exp(-x)}$, but in the recent years, non-saturating functions such as *Rectified Linear Units* (ReLU) [44, 63, 104] $f(x) = \max(0, x)$ or other variants [52] have been preferred thanks to their better trainability properties.

2.6.2.2 *Convolutional and Pooling Layers*

Similarly to Multilayer Perceptrons, a *Convolutional Neural Network* (CNN) [78, 80] is a kind of feedforward neural network where layers form a linear computational graph. Convolutional Neural Networks are particularly well suited for signals with spatial regularities thanks to special layers designed to exploit spatial patterns in data with a grid-like topology such as images. The main difference with fully connected layers consists of how the units are connected and how the input is processed to preserve its spatial structure. Rather than applying a large matrix of weights to the whole input, convolutional layers take advantage of the convolutional operator to process only a portion of the

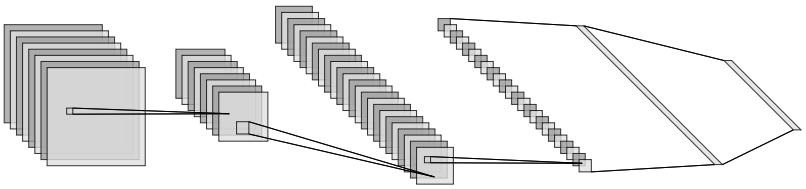


Figure 2: A series of pooling and convolutional layers followed by a fully connected layer.

input at the time and apply the same linear transformation in a sliding window fashion. This way of processing spatial data results to be beneficial for multiple reasons. Firstly, each neuron is connected only to a portion of the input, the *receptive field*, drastically reducing the number of parameters and implementing a form of *weight sharing*. Secondly, the convolution operator is equivariant to any function $g(\cdot)$ that translates the input. In other words, applying a transformation $f(\cdot)$ to a shifted version of an input x, produces the same result as applying the transformation to x and then shifting the transformed input, namely $f(g(x)) = g(f(x))$. This property is particularly useful in image recognition tasks where the same local feature should be detected regardless of its position in the image.

The weights of a convolutional layer are usually called *filters* or *kernels*, an inheritance from the signal processing community. In general, each convolutional layer learns a group of filters and the output after the activation function is a volume called *feature map*.

CNNs often reduce the spatial dimensionality of their inputs by applying the convolution operator every s pixels rather than at each location of the image, i.e., with *stride* $s > 1$, or by utilizing *pooling* or *sub-sampling* layers. Commonly, pooling layers aggregate the receptive field by taking their *mean* or *max* value. Convolutional layers can also perform the opposite operation and increase the input size via an upsampling operation [87, 131, 155]. Further details on convolutional and pooling layers can be found in the detailed guide to convolution arithmetic for deep learning [33].

2.6.3 *Representation Learning*

The success of modern supervised learning models relies on learning powerful feature extractors that build hierarchical representations also known as *embeddings*. Rather than manually designing features that might be sub-optimal, *Representation Learning* [9] delegates the feature extraction phase to several units learned explicitly for the task at hand. Neural Networks implement this mechanism efficiently via a composition of differentiable functions. If properly trained, a deep neural network can produce an internal representation of the data with several levels of abstraction [123], far more general than hand-designed features [31, 34, 91, 131, 151, 155].

2.6.4 *Backpropagation Algorithm*

The *backpropagation algorithm*, or simply *backpropagation* is an efficient method for computing the derivative of the output of neural networks with respect to its parameters. In particular, backpropagation solves the credit assignment problem in neural networks by propagating the prediction error from the top of the network through all its modules and evaluating their contributions to the final output.

The merit of proposing the backpropagation algorithm is often assigned to Rumelhart, Hinton, and Williams [123], who showed that neural networks could be successfully trained to learn useful representations; however, the first application of backpropagation to neural networks dates back to Werbos [146, 147].

Backpropagation first requires to process an input x and compute the final output $\hat{y}$. This phase is generally called *forward propagation* or *forward pass* because the information flows forward from the input layer through the network. During training, the forward pass evaluates also a scalar performance metric, called *loss* or *cost function* $\mathcal{L}(\theta)$, which has to be differentiable. Afterward, backpropagation computes the derivatives of the loss function with respect to the network's parameters, a step that takes the name of *backward pass* since the layers of the network are visited in reverse order. In the second stage, the derivatives are then used to compute the weights adjustments, usually via gradient descent. It is important to highlight that these two stages are distinct.

We now follow the derivation as presented in Bishop [11] and consider the backward pass of an MLP with $\ell = \{1, \dots, L\}$ layers described by the forward pass equations in 9. The gradient of the loss function $\mathcal{L}$ with respect to the parameters of the layer ℓ are:

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = \frac{\partial \mathcal{L}}{\partial y_L} \frac{\partial y_L}{\partial W_\ell} \tag{10a}$$

$$\frac{\partial \mathcal{L}}{\partial b_\ell} = \frac{\partial \mathcal{L}}{\partial y_L} \frac{\partial y_L}{\partial b_\ell} \tag{10b}$$

We first introduce a useful notation δ_ℓ for the layer ℓ, to indicate the *local error* of the layer with respect to the output, or simply the *delta*:

$$\delta_\ell = \frac{\partial y_L}{\partial z_\ell} \tag{11}$$

The derivatives of the output with respect to the parameters of layer ℓ can be expressed as:

$$\frac{\partial y_L}{\partial W_\ell} = \frac{\partial y_L}{\partial z_\ell} \frac{\partial z_\ell}{\partial W_\ell} \tag{12a}$$

$$\frac{\partial y_L}{\partial b_\ell} = \frac{\partial y_L}{\partial z_\ell} \frac{\partial z_\ell}{\partial b_\ell} \tag{12b}$$

From the forward pass in Equation (9c), the derivative of the pre-activation variable z_ℓ with respect to its parameters is given by the input of the layer x_ℓ, namely the activation of the previous layer:

$$\frac{\partial z_\ell}{\partial W_\ell} = x_\ell^\intercal \tag{13a}$$

$$\frac{\partial z_\ell}{\partial b_\ell} = \mathbb{1} \tag{13b}$$

By replacing Equation (11) in Equation (12) we can write:

$$\frac{\partial y_L}{\partial W_\ell} = \delta_\ell x_\ell^T \tag{14a}$$

$$\frac{\partial y_L}{\partial b_\ell} = \delta_\ell \tag{14b}$$

The key feature of backpropagation is that the activations computed in the forward pass can be stored to make the computation of the backward pass much more efficient. Thus, evaluating the derivatives only requires to calculate the value of δ_ℓ for the output and hidden layers and then apply Equation (14):

$$\delta_L = f'(z_L) \tag{15a}$$

$$\delta_\ell = W_{\ell+1}^T \delta_{\ell+1} \cdot f'(z_\ell) \tag{15b}$$

The backpropagation formulae inform us that the contribution of the units of each layer can be computed by propagating δ backward from units higher up in the network. Interestingly, backpropagation is a special case for scalar-valued functions of the *reverse accumulation mode* [48] for automatic differentiation.

2.6.5 *Gradient-based optimization*

Training models with a large number of parameters such as neural networks requires solving a complex optimization problem. In general training NNs is *NP-hard*, and there is no efficient algorithm for finding a solution to the optimization problem for general topologies and tasks [13, 51, 64].

In practice, the training complexity of NNs is overcome by exploiting their differentiability and the efficiency of the backpropagation algorithm in computing derivatives of complex compositions of functions. Indeed, under the correct hyper-parameters choice, simple gradient methods can find optimal parameter configurations that are global minimizers for a given training set [15].

Unfortunately, gradient-based learning is not free of issues, especially when it involves optimizing millions of parameters, such as in NNs. Although NNs are universal function approximators, and in principle, even the simplest MLP is capable of learning any function [56, 57], the optimization algorithm could fail in minimizing the objective in terms of performance or available time budget, resulting in underfitting the training data. Specifically, the loss landscape of NNs is in general non-convex with multiple saddle-points, local minima, and plateau, causing gradient-based optimization to be highly unstable or to stagnate in poor solutions [21, 26, 54, 83]. These difficulties have motivated specific techniques and model architectures to improve the training efficiency of gradient-based algorithms.

2.6.5.1 *Momentum*

Stochastic Gradient Descent follows the negative direction of the expected gradient to minimize a certain functional in the weights space. When the learning rate decreases with the appropriate rate, this technique converges almost

surely to the global minimum if the objective function is convex [14, 120]. On the contrary, neural networks generally provide non-convex optimization problems, limiting our ability to characterize their loss surface and to develop convergence results for the optimization algorithms. Indeed, the composition of several nonlinear functions results in loss landscapes presenting high dimensional *valleys*, *plateau*, or *ravines* [135] that requires different step sizes along different dimensions in order to be escaped. In this situation, following the actual steepest descent might not be the best strategy. For instance, the gradient direction in a valley is almost perpendicular to the flat axis and the updates oscillate back and forth in the direction of the short axis, moving very slowly along the long axis.

A widely used technique to accelerate the convergence of gradient descent is the use of a *momentum* term [106, 114] such that the weight update at the current time step depends on both the current gradient and the weight update at the previous step. The momentum term helps to average out the oscillations on the short axis while adding up contributions along the long axis, leading to faster convergence. The update rule of gradient descent with the momentum term at iteration i-th:

$$\begin{aligned} V^{(i+1)} &= \beta V^{(i)} + \nabla \mathcal{L}(\theta^{(i)}) \\ \theta^{(i+1)} &= \theta^{(i)} - \eta V^{(i+1)} \end{aligned} \tag{16}$$

where V is the momentum buffer, η is the learning step and β the amount of momentum applied. Note that with $\beta = 0$ the original gradient descent formulation is recovered.

2.6.5.2 *Second-Order Methods*

Alternatively, second-order approximations like Newton's method incorporate information about the curvature of the loss function into the optimization algorithm by rescaling the gradient with the inverse of the Hessian [107]:

$$\theta^{(i+1)} = \theta^{(i)} - H^{-1} \nabla \mathcal{L}(\theta^{(i)}) \tag{17}$$

However, to succeed, Newton's method requires the Hessian to be positive definite, namely invertible. When the loss surface is non-convex in high dimensions such as in NN, it contains many saddle points, and the application of Newton's method could be problematic [26]. Near saddle points, the eigenvalues of the Hessian are not all positive, and then Newton's method can cause updates to move in the wrong direction. This situation can be avoided by regularizing the Hessian, simply summing a diagonal term. The convergence rate of gradient descent methods is heavily influenced by the *condition number*, the ratio between the largest and smallest eigenvalues of the Hessian, $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ which should be close to one. If the condition number is high, the optimization problem is *ill-conditioned* and the convergence rate is slow. One important issue with second-order methods is that the size of the Hessian grows with $\Theta(n_\theta^2)$, quadratically with the number of parameters n_θ, quickly becoming problematic for NNs both in terms of space and the inverse time complexity. To solve this problem, one could rely on diagonal approximations, making the space complexity linear and the inverse computation trivial, trading off useful information on how different directions interact with each other. See [79] for

an earlier review on the application of second-order methods to the training of NNs, while a comprehensive treatment of numerical optimization methods can be found in Nocedal and Wright [107] and Boyd, Boyd, and Vandenberghe [15].

2.6.5.3 *Advanced First-Order Methods*

Because of the difficulties of second-order methods, various successful first-order alternatives have been proposed and popular optimizers such as AdaGrad [32], RMSProp [53], Adadelta [154] or Adam [67] are often the preferred choices for training NNs. By making use of momentum terms and automatically adapting the learning rates per dimension [127], these methods are surprisingly effective when used in conjunction with tricks that ease the optimization process. Most of these heuristics apply normalization techniques to center and decorrelate the input of each layer [29, 62, 126], inspired by standard data preprocessing methods. Most of these methods can be seen as different preconditioning of the gradient that adapts the geometry of the data to improve the rate of convergence of gradient descent.

3 META-LEARNING

We provide an overview of of Meta-Learning, including a mathematical formalization of the problem, descriptions of the most important works in the literature, and analysis of the related fields. We then discuss current limitations and open problems that are yet to be effectively addressed in Meta-Learning.

3.1 LEARNING TO LEARN

Meta-Learning, also known as *"learning-to-learn"*, is a sub-field of Machine Learning that exploits previous experience in learning to optimize learning algorithms to work well on novel tasks [41]. In the meta-learning paradigm, a machine learning model gains experience over many learning episodes, which can lead to benefits such as data and compute efficiency. Meta-Learning is better aligned with the way humans and animals learn in the real world, where learning strategies improve both on a lifetime and evolutionary timescales. Currently, the discipline is especially active in the Supervised Learning setting and it is often associated with the field of *Few-Shot Learning* (FSL), where models are challenged to quickly learn new concepts while very few data points from the task at hand are available. The problem is extremely difficult, as models operating in low data regimes are especially prone to over-fitting [84].

The field of Meta-Learning has been rising in the last few years, achieving super-human performances in simple FSL classification tasks [76]. The reasons for this are twofold. On one hand, deep learning techniques have been widely employed in Meta-Learning to achieve impressive results. On the other, critical weaknesses of Deep Learning such as the data-hungriness of deep NNs, the need for excessive computer resources, and the poor transferability of the knowledge obtained has motivated the development of meta-learning techniques to overcome its limitations. Deep Learning and Meta-Learning thus form a symbiotic relationship that encourages progress in the two fields [1].

Although we are going to focus on supervised meta-learning approaches, it is important to note that *Meta-Reinforcement Learning* to improve sample efficiency is gaining a lot of interest [50, 93]. Instead, research in *Unsupervised Meta-Learning* is still in its early stages [59, 94].

By focusing on the topic of Meta-Learning, we aspire to broaden the availability of deep learning techniques. A model that is capable to learn new, complex tasks and generalize knowledge with few training samples would prove beneficial to experts confronting niche machine learning tasks with little training data available online, which is, for instance, the case when working with clinical data. For example, in dermatology, there are many instances of rare diseases, or diseases that become rare for particular types of skin [1, 71, 122].

1 This is also reflected in the various meta-learning libraries that have been developed on top of deep learning oriented frameworks, such as Higher [47] and Torchmeta [28].

3.2 BACKGROUND

The literature provides many perspectives on Meta-Learning that are not necessarily consistent with one another. Thrun and Pratt [136] define learning-to-learn as occurring when a learner improves its ability to solve tasks drawn from a distribution of tasks improves as the number of tasks observed increases. However, this definition does not necessarily exclude paradigms that are generally not considered Meta-Learning today, such as transfer learning, which improves a learner from one domain by transferring information from a related domain [145], and multi-task learning, which leverages useful information contained in a finite set of tasks to help improve the generalization performance of all the tasks [156]. A possible way to refine our definition is by focusing on algorithms that achieve the above through end-to-end learning of an explicitly defined objective function (such as cross-entropy loss) [58].

In this section, we provide a formalization of the various concepts that are relevant in the Meta-Learning literature today.

3.2.1 *Machine Learning*

In conventional Supervised Machine Learning, we have access to a training dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $(\mathbf{x}_n, y_n)$ is a tuple containing input and desired output, e. g., input image and class label. We can train a predictive model $\hat{y} = f_\theta(\mathbf{x})$ with parameters θ by solving the minimization problem

$$\theta^* = \arg\min_\theta \mathcal{L}(\mathcal{D}; \theta; \omega) \tag{18}$$

where $\mathcal{L}$ is a loss function that measures the error between desired outputs and those predicted by our model $f_\theta(\cdot)$. The conditioning on ω denotes the dependence of this solution on assumptions about "how to learn", such as characteristics of the optimizer for θ. Generalization is then measured by evaluating the performance of the trained model on several test points whose desired output is known.

The conventional assumption is that ω is pre-specified, meaning that the optimization problem is performed *from scratch*. However, inferring ω from the data has been proven to dramatically improve performance, especially when there is little data available from the task at hand. Meta-Learning, therefore, does not assume a fixed ω and rather learns ω in an end-to-end fashion. This is done by abandoning the conventional learning from scratch and instead focusing on learning from a distribution of tasks, where ω can be inferred from general knowledge that is valid across the whole task family.

3.2.2 *Meta-Learning: Task Distribution View*

Meta-Learning is based on the concept of *episodic training*, where a learning algorithm improves its performance on *episodes* from a given *distribution of tasks* (or *task family*) $p(\mathcal{T})$ as the number of observed episodes increases. An episode is formally defined as a set of data instances sampled from the distribution associated with a specific task. During *base learning* (or *adaptation*), an *inner* (or *lower / base*) learning algorithm adapts itself to solve a task such as image clas-

sification, which is defined by a dataset and an objective. During *meta-learning*, an *outer* (or *upper / meta*) algorithm updates the inner learning algorithm such that the learned model improves an outer objective. Common outer objectives are the ability of the learned model to generalize to unseen data from the same task or the speed of the inner learning algorithm. Neural-network Meta-Learning is therefore mainly characterized by the presence of an explicit *meta-level objective* and an *end-to-end* optimization of the inner algorithm with respect to this objective.

While in a limiting case all training episodes can be sampled from a single task, we will focus on the standard case where learning episodes are sampled from a task family, leading to an inner algorithm that performs well on new tasks sampled from this family. Moreover, despite formally referring to two separate concepts, it is common in the literature to refer to episodes as tasks, thus omitting the nature of tasks as distributions over data instances. In the rest of this thesis we will also apply this convention.

In principle, a task is defined as a triple $\mathcal{T} = \langle \mathcal{D}^s, \mathcal{D}^q, \mathcal{L} \rangle$, where $\mathcal{D}^s$, $\mathcal{D}^q$ are datasets and $\mathcal{L}$ is the loss function of the task. Without loss of generality, we assume that the loss function $\mathcal{L}$ is shared across all tasks in the distribution. The two datasets contain data instances independently sampled from a common, task-specific distribution. We use the support set $\mathcal{D}^s$ jointly with ω to generate the parameters of our task-specific model

$$\theta = \arg\min_{\theta} \mathcal{L}(\theta|\omega, \mathcal{D}^s) \tag{19}$$

which is then evaluated on the query set $\mathcal{D}^q$. Finally, we can define the meta-learning problem as a minimization problem over ω:

$$\min_{\omega} \mathbb{E}_{\langle \mathcal{D}^s, \mathcal{D}^q \rangle = \mathcal{T} \sim p(\mathcal{T})} \mathcal{L}(\mathcal{D}^q|\theta(\omega, \mathcal{D}^s)) \tag{20}$$

where $\mathcal{L}(\mathcal{D}^q|\theta)$ measures the performance over the task-specific query set $\mathcal{D}^q$ with respect to the loss function $\mathcal{L}$ of the model f_θ obtained by applying our *across-task* knowledge ω on the task-specific support set $\mathcal{D}^s$.

To solve this problem in practice, we often assume to have access to a finite training set of S tasks

$$\mathscr{D}_{train} = \{\langle \mathcal{D}^s_{train}, \mathcal{D}^q_{train} \rangle^{(i)}\}_{i=1}^{S} \tag{21}$$

independently sampled from a training distribution of tasks $p_{train}(\mathcal{T})$. The *meta-training* step of "learning to learn" can then be written as

$$\omega^* = \arg\max_{\omega} \log p(\omega|\mathscr{D}_{train}). \tag{22}$$

To evaluate the obtained meta-model we also define a finite test set of Q unseen tasks

$$\mathscr{D}_{test} = \{\langle \mathcal{D}^s_{test}, \mathcal{D}^q_{test} \rangle^{(i)}\}_{i=1}^{Q} \tag{23}$$

independently sampled from a test distribution of tasks $p_{test}(\mathcal{T})$, which can be identical or simply related to $p_{train}(\mathcal{T})$. In the *meta-testing* step we use the learned meta-knowledge ω^* to train the base model on each task $\mathcal{T}^{(i)}_{test}$ in $\mathscr{D}_{test}$:

$$\theta^{(i)} = \arg\max_{\theta} \log p(\theta|\omega^*, {\mathcal{D}^s_{test}}^{(i)}). \tag{24}$$

In contrast to standard supervised learning, our training procedure is now guided by a meta-knowledge ω^* that takes into account the task family we have trained on. The meta-knowledge could be in the form of an initialization of the model's parameters, or an entire optimization strategy. Similar to many other machine learning algorithms, meta-learning procedures are prone to *meta-overfitting*, an issue whereby the meta-knowledge learned on the training tasks does not generalize well to the test tasks.

3.2.3 *Meta-Learning: Bilevel Optimization View*

The previous discussion outlines Meta-Learning as the problem of learning a training procedure to generalize across tasks from a task family but does not directly specify how to solve the optimization problem in Equation (22). This is usually accomplished by defining the meta-training step as a bilevel optimization problem. While arguably only accurate in the case of optimization-based methods (Section 3.3.3), this formalization is helpful to visualize the general functioning of any meta-learning algorithm. Bilevel optimization refers to a hierarchical optimization problem, where one optimization problem contains another as a constraint. We can therefore formalize the meta-training as:

$$\omega^* = \arg\min_{\omega} \sum_{i=1}^{S} \mathcal{L}^{meta}(\theta^{*(i)}(\omega), \omega, \mathcal{D}^{q}_{train}{}^{(i)}) \tag{25}$$

such that

$$\theta^{*(i)}(\omega) = \arg\min_{\theta} \mathcal{L}^{task}(\theta, \omega, \mathcal{D}^{s}_{train}{}^{(i)}) \tag{26}$$

where $\mathcal{L}^{meta}$ and $\mathcal{L}^{task}$ refer to the outer and inner objectives, e.g. cross entropy in the case of FSL classification. Here ω could indicate the initial parameters of the base learner, the hyper-parameters of the optimizer, or even the structure of the base model itself. The goal of the outer objective $\mathcal{L}^{meta}$ is not necessarily limited to improve the evaluation performance and can be defined to encourage the base model to improve its learning speed or robustness.

3.2.4 *Meta-Learning: Feed-Forward Model View*

Many meta-learning approaches directly synthesize the base model from the support set in a feed-forward manner rather than leveraging an inner optimization phase, a technique also known as *amortization*. While varying in degree of complexity, amortization can be easily understood through the following toy example for meta-training linear regression:

$$\min_{\omega} \mathbb{E}_{\langle \mathcal{D}^s, \mathcal{D}^q \rangle = \mathcal{T} \sim p(\mathcal{T})} \sum_{(\mathbf{x}, y) \in \mathcal{D}^q} [(\mathbf{x}^T \mathbf{g}_\omega(\mathcal{D}^s) - y)^2]. \tag{27}$$

In this case, ω represents the parameters of a network that embeds the support set $\mathcal{D}^s$ into a vector of linear regression weights to predict examples $\mathbf{x}$ from the validation set. Optimizing Equation (27) via SGD "learns to learn" by training the function g_ω to map a support set to a weight vector that essentially acts as the parameters of the base model.

3.3 TAXONOMY

Meta-learning approaches throughout the literature vary significantly in both working principles and scope. In this section, as seen in the work of Hospedales et al. [58], we categorize them into three distinct branches to form the following taxonomy:

- *Metric-Based* approaches;
- *Model-Based* approaches;
- *Optimization-Based* approaches.

3.3.1 *Metric-Based*

Metric-based or non-parametric approaches are restricted to the popular but specific FSL applications of Meta-Learning. Models in this category leverage the similarity of new data instances with the ones in $\mathcal{D}^s$ to predict their outputs. Outer-level learning corresponds to Metric Learning, i. e., learning a suitable kernel function k_ω, usually via SGD.

3.3.1.1 *Prototypical Networks*

Prototypical Networks [132] learn a metric space that enables classification by computing the distance of a datapoint to prototype representations of each class. The distance is computed in an embedding space and each prototype representation is computed as the average of the embedded datapoints from the corresponding class. The simplicity of this architecture proves to be beneficial in the context of FSL classification tasks, where overfitting is an important and critical problem.

3.3.1.2 *Relation Networks*

Relation Networks [133] are trained end-to-end to learn a deep distance metric for images. Their structure can be divided into two subsequent modules; firstly, an embedding module generates compressed image representations, secondly, a relation module determines whether two images belong to the same category based on their representations. The obtained distance metric can finally be used in FSL classification tasks without updating the network to predict the label of a new data instance by leveraging its distance with respect to the other labeled data instances.

3.3.2 *Model-Based*

In *model-based* (or *black-box*) methods the inner learning step is performed via the forward pass of a FNN. The model embeds the support set $\mathcal{D}^s$ into an activation state ω, based on which predictions are made for new data from $\mathcal{D}^q$. While the simpler optimization does not require second-order gradients, it has been observed that model-based approaches are less able to generalize to out-of-distribution tasks than optimization-based methods [37]. Moreover,

as ω and $\mathcal{D}^s$ directly specify ω, outer and inner-level optimizations are tightly coupled, which generally leads to poorer interpretability of the results.

3.3.2.1 *LSTM-Based Meta-Learning*

Ravi and Larochelle [118] leverage *Long short-term memory cells* (LSTM) Hochreiter and Schmidhuber [55] to learn a meta-learner model that is then used to optimize another learner neural network in a FSL classification setting. The meta-learner uses its state to represent across-task knowledge ω in the form of the initialization for the learner's parameters as well as the update rule to be applied a set amount of times during training, addressing the slow speed of convergence that traditional SGD techniques face in case of low availability of samples.

3.3.2.2 *Fast Weights*

Fast Weights architectures [100] divide weights of a FNN in two typologies. Slow weights are learned following a learning procedure on the whole task distribution and represent task-agnostic knowledge ω. In contrast, fast weights are amortized for the specific task at hand and therefore represent task-specific knowledge θ.

3.3.3 *Optimization-Based*

Optimization-based approaches adapt the inner-level task by literally solving an optimization problem, and focus on extracting meta-knowledge ω required to improve optimization performance. Many gradient-based methods of this type are subject to computing and memory challenges, especially when the inner optimization consists of multiple steps.

3.3.3.1 *Model-Agnostic Meta-Learning*

Model-Agnostic Meta-Learning (MAML) [36] is a simple meta-learning procedure that is general and model-agnostic, i. e., it can be applied to any problem and model that is trained with a gradient descent procedure. MAML is therefore extremely versatile and can be applied to a variety of different tasks, including classification, regression, and reinforcement learning.

The objective of MAML is to find the set of initial parameters θ for the model f_θ minimizing the estimate of the loss over a distribution of tasks after one (or more) gradient steps:

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}). \tag{28}$$

Many earlier meta-learning approaches learn an update function or learning rule, causing the number of learned parameters to grow. Instead, MAML learns the initial parameters via simple SGD, leading to a simpler model that is less prone to meta overfitting.

MAML is organized in two loops, as shown in Algorithm 1. The inner loop computes the loss obtained with the initialization parameters over a batch of

Algorithm 1 Model-Agnostic Meta-Learning

Require: Training meta-set $\mathscr{D}_{train}$
Require: Learning rates α, η
1: randomly initialize θ
2: **while** not done **do**
3: **for** number of tasks in batch **do**
4: Sample task $\mathcal{T}_i \in \mathscr{D}_{train}$
5: Let $(\mathcal{D}^s, \mathcal{D}^q) = \mathcal{T}_i$
6: Initialize $\theta_i' = \theta$
7: **for** number of adaptation steps **do**
8: Compute support loss $\mathcal{L}^s_{\mathcal{T}_i}\left(f_{\theta_i'}\right)$
9: Perform gradient step w.r.t. θ_i':
$\theta_i' \leftarrow \theta_i' - \alpha \nabla_{\theta_i'} \mathcal{L}^s_{\mathcal{T}_i}\left(f_{\theta_i'}\right)$
10: **end for**
11: Compute query loss $\mathcal{L}^q_{\mathcal{T}_i}\left(f_{\theta_i'}\right)$
12: **end for**
13: Perform gradient step w.r.t θ:
$\theta \leftarrow \theta - \eta \nabla_\theta \sum_{\mathcal{T}_i} \mathcal{L}^q_{\mathcal{T}_i}\left(f_{\theta_i'}\right)$
14: **end while**

tasks. The outer loop then updates the initialization parameters using the meta-gradient to minimize the average loss, until a certain condition is met. The success of MAML has contributed to the recent rise of the inner-outer loop paradigm in the meta-learning literature, which has been later reused in works such as ANIL [115] and LEO [124].

Nonetheless, MAML and its extensions also suffer from a variety of problems causing training instability, restricted generalization, reduced flexibility, increased computational overhead, and costly hyperparameter tuning. To address some of these problems, Antoniou, Edwards, and Storkey [6] propose adjustments to the original algorithm, such as the definition of a loss that is spread across multiple gradient steps, heuristics reducing the computational cost of second-order derivative, and more suitable handling of batch normalization layers inside the model.

3.3.3.2 *Latent Embedding Optimization*

Latent Embedding Optimization (LEO) [124] learns a *latent code* of the model parameters $\mathbf{z}$ in a low-dimensional latent space, avoiding the problem of the high dimensionality of model parameters when optimizing via gradient descent. The latent code is stochastic, which allows to better express the uncertainties that inevitably arise when operating in low data regimes. Furthermore, unlike many other optimization-based approaches, the support set $\mathcal{D}^s$ is used to amortize the initial latent code, making the initial state of the model task-dependent and boosting adaptation speed. The amortization is performed by including in the model architecture an encoder-decoder structure that meta-learns the parameter encoding and decoding. The encoder $g_{\phi_{enc}}$ takes as input the support set $\mathcal{D}^s$ of the task to provide a task-dependent initial latent representation. The

decoder $g_{\phi_{dec}}$ then processes the representation $\mathbf{z}$ to generate the task-specific parameters θ for the base model f_θ.

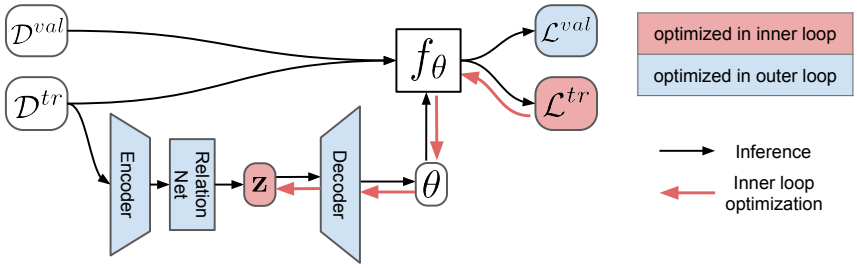


Figure 3: Overview of the architecture of LEO [124]

In the case of classification tasks, the decoder is usually followed by a relation network $g_{\phi_{rn}}$, exploiting the inter and out-of-class relations between images to generate a latent representation for each class that takes into account similarity and differences with the other classes. Each latent class n representation is then decoded to a vector of parameters θ_n, which is typically the corresponding class column of the linear classifier f_θ. Moreover, when the dimensionality of the inputs is too large (e. g., images), we leverage a pre-trained embedding to avoid Meta-Learning to be performed with respect to a high-dimensional input space.

LEO's optimization algorithm is shown in Algorithm 2. It is similar to MAML, featuring the same outer-inner loop structure. The encoding and decoding procedures are differentiable and can be learned through traditional SGD techniques in the outer loop. Furthermore, decoder differentiability allows for gradient descent in the latent embedded space by backpropagation.

Algorithm 2 Latent Embedding Optimization

Require: Training meta-set $\mathscr{D}_{train}$
Require: Learning rates α, η
Randomly initialize $\phi_{enc}, \phi_{rn}, \phi_{dec}$
Let $\phi = \{\phi_{enc}, \phi_{rn}, \phi_{dec}, \alpha\}$
while not converged **do**
  for number of tasks in batch **do**
    Sample task $\mathcal{T}_i \in \mathscr{D}_{train}$
    Let $(\mathcal{D}^s, \mathcal{D}^q) = \mathcal{T}_i$
    Encode $\mathcal{D}^s$ to $\mathbf{z}$ using $g_{\phi_{enc}}$ and $g_{\phi_{rn}}$
    Decode $\mathbf{z}$ to initial params θ_i using $g_{\phi_{dec}}$
    Initialize $\mathbf{z}' = \mathbf{z}$, $\theta_i' = \theta_i$
    for number of adaptation steps **do**
      Compute support loss $\mathcal{L}^s_{\mathcal{T}_i}(f_{\theta_i'})$
      Perform gradient step w.r.t. $\mathbf{z}'$:
      $\mathbf{z}' \leftarrow \mathbf{z}' - \alpha \nabla_{\mathbf{z}'} \mathcal{L}^s_{\mathcal{T}_i}(f_{\theta_i'})$
      Decode $\mathbf{z}'$ to obtain θ_i' using $g_{\phi_{dec}}$
    end for
    Compute query loss $\mathcal{L}^q_{\mathcal{T}_i}(f_{\theta_i'})$
  end for
  Perform gradient step w.r.t ϕ:
  $\phi \leftarrow \phi - \eta \nabla_\phi \sum_{\mathcal{T}_i} \mathcal{L}^q_{\mathcal{T}_i}(f_{\theta_i'})$
end while

Because the training procedure is based on MAML, LEO shares with the former many issues, such as heavy computational cost. The problem is however

partially addressed thanks to the low dimensionality of the latent parameter code. Moreover, empirical evidence suggests that small gradient steps in the compressed parameter representation correspond to significant changes in the model parameters, which is also implied by the relatively high curvature that the representation space exhibits with respect to the model parameter space. LEO can therefore reach a much broader region of parameters within few steps, which might lead to a further advantage over traditional optimization-based techniques in terms of flexibility and compute efficiency.

3.3.3.3 *Almost No Inner Loop*

Raghu et al. [115] claim that the effectiveness of MAML is due to feature reuse, with the meta-initialization already containing high-quality features. They propose *Almost No Inner Loop* (ANIL), a simplification of MAML that removes adaptation for all but the task-specific head of the network. ANIL often matches MAML in performance and requires substantially less computing power both during meta-learning and task adaptation.

3.3.3.4 *Multimodal MAML*

Multimodal Model-Agnostic Meta-Learning (MMAML) [144] extends MAML with the capability of identifying the type of a task from a multimodal task distribution, thus providing the optimal parameters with respect to the identified type. This approach is especially useful in the case of a complex task distribution featuring very different possible tasks, where the original MAML struggles with providing a parameter initialization that is adequate for all of them. However, the approach does not contemplate the possibility for tasks from different modes to share some relevant knowledge, which may act as a beneficial regularization.

3.3.3.5 *Hierarchically Structured Meta-Learning*

Hierarchically Structured Meta-Learning (HSML) [149] organizes tasks in a hierarchical clustering structure to improve knowledge customization among different clusters and knowledge sharing among tasks from the same cluster. The hierarchy effectively addresses the limiting assumption of transferable knowledge being globally shared that is made by many other works in the literature. The hierarchic structure of tasks is initialized at the beginning of the Meta-Learning process and is thus dynamically expanded when a task does not fit in any of the available clusters. The model learns for each cluster a representation of the knowledge shared across all tasks in the cluster, which can be the parameter initialization in the case of models trained with gradient descent. The approach is extremely general and can be applied to any model trained with gradient descent and more. As a downside, considering a single hierarchy over the distribution of tasks could be limiting, as there could exist multiple distinct hierarchies that enable effective sharing of knowledge.

3.3.3.6 *Baseline++*

A popular baseline method in FSL classification consists of a linear classifier built on top of a feature extractor. Chen et al. [19] propose baseline++, a new, simple baseline method where the classifier instead features a layer computing the cosine-distance between the feature vector and weight vector for each class. Baseline++ achieves surprisingly good performances, comparable with the state of the art. They also empirically show that current FSL classification algorithms cannot generalize well to never seen before data domains, as baseline++ compares favorably against any other method available in the literature in this case.

3.3.3.7 *Warped Gradients*

Warped Gradient Descent (WarpGrad) [38] learns a preconditioning matrix that facilitates SGD across the task distribution. This is done by interleaving warp layers between each learner's layer that precondition the layer's activation in the forward pass and gradients in the backward pass.

Unlike prior similar works, WarpGrad can be applied to FNN and the warp projections are computed by a generic NN instead of a simple linear layer. Like many other gradient-based meta-learning techniques, WarpGrad features a computational cost that is linear with respect to the number of adaptation steps, which rules out experimenting with a large number of adaptation steps in complex networks.

3.4 RELATED FIELDS

The field of Meta-Learning is not to be confused with other areas that share similarities, such as dealing with multiple learning tasks and assuming a probability distribution over the space of tasks. Here we illustrate some of the most important fields whose relation with Meta-Learning is often a source of confusion and outline their main differences.

3.4.1 *Transfer Learning*

Transfer Learning [108] uses experience from a source task to improve the performance on a target task. The term is used to refer both to the problem area and the family of solutions, which are usually in the form of parameter transfer and fine tuning [150].

In contrast, Meta-Learning is a more general paradigm and can be used to tackle both problems that are and are not Transfer Learning. Moreover, while in Transfer Learning the prior is extracted via standard learning, Meta-Learning relies on a meta-objective that evaluates the benefit of the prior when learning a new task. Finally, Meta-Learning deals with a much wider range of meta-representations than solely model parameters.

3.4.2 *Domain Adaptation and Domain Generalization*

Domain shift refers to the situation where the source and target task share the same objectives, but the target input distribution of the target task is shifted with respect to source one, affecting model performance [22, 108]. *Domain Adaptation* attempts to learn representations that alleviate this issue by adapting the source-trained model using unsupervised data from the target domain. *Domain Generalization* refers to methods that learn to be robust to domain shift without such an adaptation. Multiple approaches have been studied and proposed to address domain shift and boost target domain performance [22, 108].

However, Domain Adaptation and Generalization do not necessarily rely on optimizing a meta-objective to achieve their goal. On the other hand, Meta-Learning can and has been used to perform both Domain Adaptation and Generalization.

3.4.3 *Continual Learning*

In *Continual Learning*, models learn from a sequence of tasks from a potentially non-stationary distribution and in particular attempt to do so by rapidly extrapolating knowledge from new tasks without forgetting old tasks [20, 109, 119]. Therefore, similarly to Meta-Learning, a distribution over the tasks is considered, and the goal is to accelerate learning of a target task.

However, most continual-learning approaches do not rely on explicit optimization of a meta-objective as seen in Meta-Learning. Nonetheless, Meta-Learning can be effective in tackling continual learning problems, with a few recent studies considering a meta-objective that incorporates continual-learning performance [4, 20, 103].

3.4.4 *Multi-Task Learning*

Multi-Task Learning and Meta-Learning are very related fields of Machine Learning. Both exploit the presence of shared structures across multiple tasks to speed up the training, with a different goal in mind. Multi-Task Learning aims to jointly learn several related tasks, to benefit from regularization that arises from parameter sharing [18], as well as compute and memory savings.

Unlike Meta-Learning, conventional Multi-Task Learning is a single-level optimization and does not consider a meta-objective. Moreover, while the goal of Multi-Task Learning is to solve a fixed, finite number of seen tasks, Meta-Learning aims to generalize to and learn quicker future unseen tasks coming from a probability distribution. Nevertheless, Meta-Learning can be employed in the Multi-Task Learning setting, e. g., by learning relatedness between tasks or how to prioritize among multiple tasks.

The Multi-Task Learning literature provides several approaches to model shared information across tasks. They can be roughly separated into two different categories, i. e., methods where parameters of the model are close to each other in a geometric sense [35, 140] and approaches where the parameters of the model share a common structure [25, 77, 110, 116, 152]. This structure can be a clustering assumption [152], a (Gaussian) prior for the parameters of

all tasks [77] or some advanced structure like the Kingman's coalescent [25] which is a continuous-time partitioned prior. Argyriou, Evgeniou, and Pontil [7] propose an inductive bias on task parameters assuming them to lie in a low dimensional linear subspace. Successively, Agarwal, Gerber, and Daume [3] consider all task parameters to lie on a manifold. Based on the subspace assumption, Kumar and Daume III [74] propose a framework to selectively share the information across tasks, assuming that each task parameter vector is a linear combination of a finite number of underlying basis tasks. Other works differentiate between tasks and address the fact that some of them might be unrelated, by assuming the existence of *disjoint groups of tasks* [66] or allowing two tasks from different groups to overlap by having one or more bases in common [74].

3.4.5 *Hyperparameter Optimization*

Hyperparameter Optimization can indeed be seen as Meta-Learning since hyperparameters like learning rate and regularization coefficients effectively describe 'how to learn'. In particular, we consider as Meta-Learning the problems that define a meta-objective that is trained end-to-end, such as gradient-based hyperparameter learning [40].

In contrast, other approaches like random search [10] and Bayesian Hyperparameter Optimization [129] are rarely considered to be Meta-Learning.

3.4.6 *Hierarchical Bayesian Models*

Hierarchical Bayesian Models involve Bayesian learning of parameters θ under a prior $p(\theta|\omega)$. The prior is conditioned on some other variable ω with its own prior $p(\omega)$. Hierarchical Bayesian models are often used to in the case of grouped data $\mathcal{D} = \{\mathcal{D}_i | i = 1, 2, \ldots, M\}$, where each group i has its own θ_i. The full model therefore is

$$\left[\prod_{i=1}^{M} p(\mathcal{D}_i|\theta_i)p(\theta_i|\omega)\right] p(\omega). \tag{29}$$

The levels of the hierarchy can be increased further; in particular, ω can be parameterized, and hence $p(\omega)$ can be learned.

Bayesian hierarchical models provide a useful perspective for Meta-Learning, by providing a modeling framework to understand the meta-learning process from an algorithm-agnostic point of view. In practice, prior works in the literature usually focus on learning simple, tractable models θ. Instead, Meta-Learning often consists of complex inner-loop learning procedures involving many iterations. Nonetheless, some meta-learning approaches such as MAML [36] can be understood from the standpoint of Hierarchical Bayesian Models [46].

3.4.7 *Automated Machine Learning*

Automated Machine Learning (AutoML) [60] is a relatively broad umbrella for methods that aim to automate phases of the machine-learning process that are

typically hand-crafted, such as data preparation, algorithm selection, hyper-parameter tuning, and architecture search. AutoML often leverages multiple heuristics that are usually outside the scope of what is considered Meta-Learning. Moreover, tasks like data-cleansing are not particularly considered in the meta-learning literature. Nonetheless, some AutoML approaches do perform end-to-end optimization of a meta-objective; Meta-Learning can thus be seen as a specialization of AutoML.

3.5 DISCUSSION

The meta-learning literature has produced a large variety of interesting and performing approaches in the last few years, mainly thanks to the recent breakthroughs in Deep Learning. However, Meta-Learning, as a field, still faces many challenges that are yet to be thoroughly addressed. Many popular approaches also struggle when scaling to the complexity of the learner, which ultimately constrains them to poor performance when confronting complex tasks. In some cases, heterogeneity of the various tasks may also be the cause of counter-productive transfer learning, where irrelevant knowledge from a task is erroneously reused when dealing with another task [153]. The problem may be addressed by reinforcing the task-specific inductive bias, though it is important to introduce regularization to avoid unwanted overfitting. Finally, despite the latest efforts and achievements in the field, the literature still does not offer a method that can generalize its knowledge to tasks from unseen, out-of-distribution data domains. Most state-of-the-art techniques were designed and tested on simple benchmarks featuring samples from a single domain, such as Omniglot [76]. The lack of ability to generalize to new domains, unfortunately, hinders the ability to employ deep-learning solutions to tackle interesting and important problems when the amount of available training data is low. When it comes to discriminating new categories from data, even humans have trouble dealing with datasets that vary too greatly between examples or differ from prior experience. Because of the many applications that require high-quality learning from few datapoints, and the difficulty that both machines and humans face when learning in these circumstances, finding new methods to tackle the problem remains a challenging but desirable goal.

In light of these issues, our main objective is to determine whether the currently provided FSL techniques can be further extended to generalize previous knowledge on new tasks from unseen data domains, a problem also known as *Cross-Domain Few-Shot Learning* (CDFSL) [49]. A model capable of operating among different data domains would be able to transfer knowledge across widely different tasks, potentially solving the lack of training samples that are observed in certain data domains. As a practical example, our desired model would be able to generalize the recognition of malignant tumors in x-ray images to images obtained through other less popular techniques or instruments that may feature different colors and patterns.

4

ADDRESSING DOMAIN SHIFT

We examine various approaches in the machine learning literature that address the presence of multiple domains in our data. We first present variational autoencoders, powerful models that allow us to easily model our data. We then analyze other techniques from the fields of domain adaptation, domain generalization, and cross-domain few-shot learning.

4.1 DISENTANGLEMENT AND INTERPRETABILITY

4.1.1 *Introduction*

When dealing with data coming from multiple domains, a simple line of work consists of designing and leveraging a good embedding to overcome limitations related to heterogeneous data. In the meta-learning literature, the approach is supported by Raghu et al. [115], arguing that feature reuse is the dominant component in MAML's efficacy, with Tian et al. [137] recently confirming this hypothesis. In our CDFSL, it is thus reasonable to believe that a good embedding satisfies the three following properties:

- *Reusability*: the representation or part of it can be reapplied to other problems.
- *Disentanglement*: the representation can be easily decomposed into independent units.
- *Interpretability*: the various units have clear semantic meaning.

It could be argued that any embedding that is obtained from a meta-learning algorithm is reusable to an extent, as the goal of Meta-Learning is to perform well on new, unseen tasks. Therefore, we may want to combine existing meta-learning approaches with other machine learning techniques that guarantee the disentanglement and interpretability of the embedding.

4.1.2 *Probabilistic Graphical Models*

Probabilistic graphical models [72] are a family of models that aim to describe the conditional dependencies between a set of stochastic variables $X_1, \dots, X_N$ by representing them as nodes in a directed graph. The joint probability of a probabilistic graphical model is given by

$$p[X_1, \dots, X_N] = \prod_{n=1}^{N} p[X_n | pa(X_n)] \tag{30}$$

where $pa(X_n)$ is the set of parents of X_n. Probabilistic graphs allow us to explicitly design relations among variables in our model to partition information derived from data into latent components that are both disentangled and interpretable.

4.1.3 *Bayesian Networks*

Bayesian networks [111] are a subclass of probabilistic graphical models that require the directed graph to also be acyclic. In this case, we can understand the sampling procedure from the joint distribution as a step-by-step process where individual nodes are sampled in topological order and the distribution of each node depends on the values of its parents.

We can leverage Bayesian networks to model a latent representation of the raw data $\mathbf{x}$ given a dataset of observations $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{I}$. While in the general case the graph may feature multiple nodes corresponding to many observations and latent variables, we analyze as an example the simple case of a graph with two nodes, $\mathbf{z}$ and $\mathbf{x}$, with a single arc from $\mathbf{z}$ to $\mathbf{x}$. By considering the conditional distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$ we can treat $\mathbf{x}$ as an observable, noisy realization generated from the meaningful latent embedding $\mathbf{z}$. Therefore, we may be interested in the posterior distribution $p_{\theta^*}(\mathbf{z}|\mathbf{x})$ to infer from the observation $\mathbf{x}$ the latent embedding $\mathbf{z}$ that is likely to generate $\mathbf{x}$. A possible way to accomplish this is thus to first learn an approximation $p_\theta(\mathbf{x}, \mathbf{z})$ of the true joint distribution $p_{\theta^*}(\mathbf{x}, \mathbf{z})$ by maximizing the likelihood over the entire dataset of observations $\mathcal{D}$:

$$\max_{\theta} \prod_{\mathbf{x} \in \mathcal{D}} \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}. \tag{31}$$

After optimization we can rely on the approximated posterior $p_\theta(\mathbf{z}|\mathbf{x})$ to infer $\mathbf{z}$ from a new observation $\mathbf{x}$. In practice, obtaining the exact posterior distribution is complex and often intractable. Variational Bayesian approaches attempt to solve this problem by optimizing a *variational lower bound* to find an approximate posterior, but still require analytical solutions that may be intractable in the general case.

4.1.4 *Variational Autoencoders*

Variational Autoencoders (VAEs) [69, 70] aim to maximize a simple, unbiased estimator of the variational lower bound, which is easily differentiable and suitable to be optimized using online SGD. VAEs manage to combine the interpretability of Bayesian networks and the powerful techniques of deep learning, paving the way for a variety of new interesting approaches.

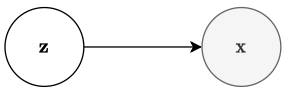


Figure 4: Generative model of a simple VAE.

Given a dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{I}$, the simplest form of VAE is based on a Bayesian network with two nodes, $\mathbf{z}$ and $\mathbf{x}$, with an arc from $\mathbf{z}$ to $\mathbf{x}$. The latent variables $\{\mathbf{z}_i\}_{i=1}^{I}$ and the true generative model of the data $p_{\theta^*}(\mathbf{x}, \mathbf{z})$ are unknown. In this case, the VAE consists of two coupled, independently parameterized models that support each other. The generative model $p_\theta(\mathbf{x}, \mathbf{z})$ learns an approximation of the true generative model of the data. At the same time, as computing the exact posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is usually intractable, we also rely on a recognition

model $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the true posterior distribution identified by θ. We assume that the generative model $p_\theta(\mathbf{x},\mathbf{z})$ and recognition model $q_\phi(\mathbf{z}|\mathbf{x})$ are differentiable with respect to the set of parameters θ and ϕ respectively. The structure of Variational Autoencoders resembles that of traditional Autoencoders. We can refer to the generative model as a *variational encoder* and the recognition model as *variational encoder*.

In principle, the objective in VAEs is to obtain $\hat{\theta}$ parameters of the generative model that maximizes the likelihood of the dataset as well as $\hat{\phi}$ parameters of the recognition model that better approximate the true posterior according to some metric. However, the problem is still computationally intractable. Therefore, some complexity is relaxed by instead maximizing the *evidence lower bound*, abbreviated as ELBO, which is a lower bound on the log-likelihood of the data.

Derivation of the ELBO is as follows:

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right] && (32)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] && (33)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] && (34)\\
&= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right]}_{\substack{=\mathcal{L}_{\theta,\phi}\\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right]}_{=D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))} && (35)
\end{aligned}$$

The first term in eq. (35) is the ELBO:

$$\mathcal{L}_{\theta,\phi} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{z}|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]. \tag{36}$$

The second term in eq. (35) is the *Kullback-Leibler* (KL) divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z}|\mathbf{x})$, which is non-negative:

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \geqslant 0 \tag{37}$$

and zero if, and only if, $q_\phi(\mathbf{z}|\mathbf{x})$ equals the true posterior distribution. Due to the non-negativity of the KL divergence, the ELBO is a lower bound on the log-likelihood of the data.

$$\begin{aligned}
\mathcal{L}_{\theta,\phi} &= \log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))\\
&\leqslant \log p_\theta(\mathbf{x})
\end{aligned}$$

Interestingly, the KL divergence $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$ determines both the quality of the approximated posterior q_ϕ (by definition) and the gap between ELBO and log-likelihood of the data (according to eq. (35)). Therefore, the benefits of maximizing the ELBO are twofold. On one hand, it approximately maximizes the marginal likelihood $p_\theta(\mathbf{x})$, meaning that our generative model becomes better. On the other, it minimizes the KL divergence of q_ϕ from the true posterior, therefore, our recognition model also becomes better.

While the actual value of the ELBO and its gradient $\nabla_{\theta,\phi}\mathcal{L}_{\theta,\phi}$ is in general intractable to compute, we can resort to an unbiased estimator of the gradient so that we can still perform minibatch SGD. In particular, in the case of

$\nabla_\theta \mathcal{L}_{\theta,\phi}$, we employ a simple Monte Carlo estimator by random sampling $\mathbf{z}$ from $q_\phi(\mathbf{z}|\mathbf{x})$:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\theta,\phi} &= \nabla_\theta \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (38)\\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\right] && (39)\\ &\simeq \nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})) && (40)\\ &= \nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z})). && (41)\end{aligned}$$

However, unbiased gradient with respect to the parameters ϕ are harder to obtain, since the ELBO's expectation is taken with respect to the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ which is a function of ϕ. In fact, in general it holds that

$$\begin{aligned}\nabla_\phi \mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (42)\\ &\neq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\phi(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))\right]. && (43)\end{aligned}$$

Nevertheless, in the case of continuous latent variables we can resort to the reparameterization trick, i. e., we can express the random variable $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ as some differentiable, invertible transformation $\mathbf{z} = g(\epsilon, \phi, \mathbf{x})$ of another random variable ϵ whose distribution is independent from $\mathbf{x}$ or θ. Under the reparameterization, we can rewrite the ELBO as

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (44)\\ &= \mathbb{E}_{p(\epsilon)}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] && (45)\end{aligned}$$

where $\mathbf{z} = g(\epsilon, \phi, \mathbf{x})$. With the ELBO in this form, we can easily compute an unbiased estimate of the gradient for the parameters ϕ analogously to what is done for θ. We can then jointly learn the parameters θ, ϕ using online SGD.

A popular choice for VAEs is to model the generative and recognition model as NNs. Moreover, the marginal prior over the latent variable is often defined as $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. In this case, we can rewrite our generative model as $p_\theta(\mathbf{z}|\mathbf{x})p(\mathbf{z})$ and only learn the conditional distribution.

As they are based on probabilistic graphical models, VAEs allow us to come up with models that ensure a disentangled and interpretable representation. For example, when classifying hand-written digits, we may be able to capture the notion of writing styles, which is independent of the notion of digit number. In the following, we illustrate and analyze various architectures that deal with classification tasks and employ multiple latent variables to obtain an embedding satisfying our desired properties. Indeed, a richer Bayesian network also leads to higher complexity of the model's architecture.

4.1.4.1 *SSVAE*

The *Semi-supervised Variational Autoencoder* (SSVAE) [68] is a hierarchical VAE that operates in semi-supervised classification tasks. SSVAE encourages disentanglement in its latent representation by leveraging a latent variable that captures residual, non-class information.

In SSVAE, the dataset consists of both labeled and unlabeled samples:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i \in I_{sup}} \cup \{(\mathbf{x}_i, \cdot)\}_{i \in I_{unsup}}. \quad (46)$$

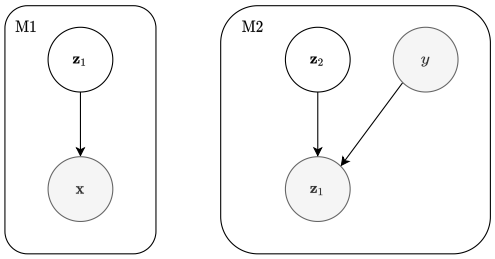


Figure 5: Generative model in SSVAE.

Therefore, the trivial VAE graphical model with two nodes $\mathbf{x}$, $\mathbf{z}_1$ is extended with two new nodes y and $\mathbf{z}_2$, with prior factorization is

$$p(\mathbf{x}, y, \mathbf{z}_1, \mathbf{z}_2) = p(\mathbf{x}|\mathbf{z}_1)p(\mathbf{z}_1|y, \mathbf{z}_2)p(y)p(\mathbf{z}_2). \tag{47}$$

The hierarchical architecture consists of:

- M1: a standard VAE encoding raw data $\mathbf{x}$ in a latent embedding variable $\mathbf{z}_1$.
- M2: a novel VAE encoding $\mathbf{z}_1$ obtained from M1 in two independent variables $\mathbf{z}_2$ and y.

We first train M1 separately and freeze its parameters during the training of M2. The goal in M2 is to disentangle the information in $\mathbf{z}_1$ to store class information in the label variable y and residual, class-invariant information in $\mathbf{z}_2$. Two different losses are considered when learning M2 to account for the partial observability of y. The variational encoding process in M2 also acts as a class predictor for unlabeled datapoints.

4.1.4.2 *VFAE*

The *Variational Fair Autoencoder* (VFAE) [88] is a VAE that can operate in a semi-supervised setting and improves upon SSVAE by extracting features that are more relevant to the task at hand. VFAE can capture and filter nuisance information from the data and better guide the flow of class information inside the model compared to SSVAE, which helps in guaranteeing high-quality disentanglement and interpretability of the latent representation.

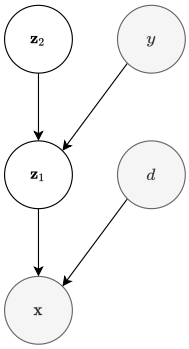


Figure 6: Generative model in VFAE.

In VFAE, each datapoint in the dataset $\mathbf{x}_i$ is paired with a domain label d_i, while the class label may be missing for some tuples:

$$\mathcal{D} = \{(\mathbf{x}_i, d_i, y_i)\}_{i \in I_{sup}} \cup \{(\mathbf{x}_i, d_i, \cdot)\}_{i \in I_{unsup}}. \tag{48}$$

VFAE extends the SSVAE graphical model to feature an additional observed variable d in the unsupervised feature extraction that is assumed to be independent from the latent representation $\mathbf{z}_1$. One of the goals in VFAE is to obtain from $\mathbf{x}$ a latent representation $\mathbf{z}_1$ that is maximally informative about the partially observed variable y while minimally informative about the observed variable d. In the context of domain adaptation, d is treated as a nuisance variable, i. e., by filtering information related to the domain d from our representations we are likely to obtain improved performance.

The generative model of VFAE can be factorized as

$$p(\mathbf{x}, d, y, \mathbf{z}_1, \mathbf{z}_2) = p_{\theta_x}(\mathbf{x}|d, \mathbf{z}_1)p(d)p_{\theta_{z_1}}(\mathbf{z}_1|y, \mathbf{z}_2)p(y)p(\mathbf{z}_2). \tag{49}$$

The priors of the domain variable d and label y can be modeled as multinomial distributions whose statistics are inferred from the dataset and thus do not need parameterization. We can also avoid to parameterize the prior of $\mathbf{z}_2$ by assuming $p(\mathbf{z}_2) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Concerning the recognition model, the posterior is factorized as

$$q(\mathbf{z}_1, \mathbf{z}_2, y|\mathbf{x}, d) = q_{\phi_{z_1}}(\mathbf{z}_1|\mathbf{x}, d)q_{\phi_y}(y|\mathbf{z}_1)q_{\phi_{z_2}}(\mathbf{z}_2|\mathbf{z}_1, y) \tag{50}$$

where the posteriors of $\mathbf{z}_1$ and $\mathbf{z}_2$ are modeled as normal distributions. The model learns from both labeled and unlabeled data and the predictive posterior $q_{\phi_y}(y|\mathbf{z}_1)$ can be used as a class predictor for unlabeled datapoints, analogously to SSVAE. A regularization term that estimates maximum mean discrepancy is also considered in the loss to guarantee the independence between $\mathbf{z}_1$ and d.

Unlike SSVAE, all the parameters of VFAE are jointly learned to promote a representation $\mathbf{z}_1$ that is strongly correlated to y. This also prevents situations where the model erroneously recognizes useful information as coming from d rather than $\mathbf{z}_1$ due to possible correlations between d and y, though this is usually not a problem in the domain adaptation setting as we assume the two variables to be independent.

4.1.4.3 *DIVA*

The *Domain Invariant Variational Autoencoder* (DIVA) [61] extends the standard VAE model by partitioning the latent space in three *latent subspaces* (or *latents*) representing domain, class, and residual information of the raw data $\mathbf{x}$, to learn a disentangled and interpretable representation. Furthermore, the continuity and stochasticity of the latents can potentially be reusable, i. e., generalize to images from unseen classes and domains, which is particularly useful in the CDFSL setting.

In DIVA, each datapoint in the dataset $\mathbf{x}_i$ is paired with a domain label d_i, while the class label y_i may be missing for some tuples:

$$\mathcal{D} = \{(\mathbf{x}_i, d_i, y_i)\}_{i \in I_{sup}} \cup \{(\mathbf{x}_i, d_i, \cdot)\}_{i \in I_{unsup}}. \tag{51}$$

DIVA's latent space $\mathbf{z}$ is partitioned in three subspaces $\mathbf{z}_d$, $\mathbf{z}_x$, and $\mathbf{z}_y$, with prior distributions over $\mathbf{z}_d$ and $\mathbf{z}_y$ conditioned on the domain label d and the class label y respectively. In the graphical model, d and y are independent, which is consistent with the standard definition of the domain as a nuisance variable that is commonly adopted in the cross-domain setting.

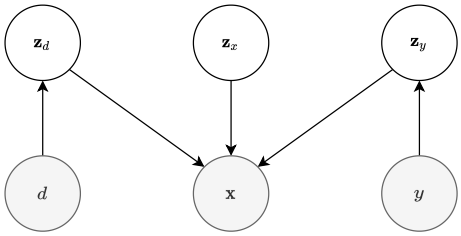


Figure 7: Generative model in DIVA.

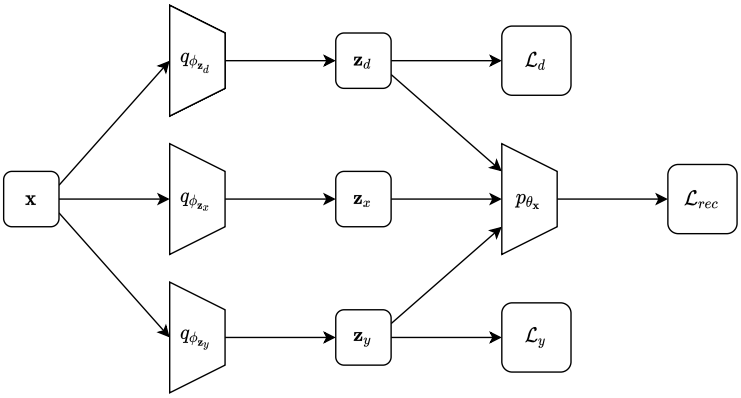


Figure 8: DIVA architecture.

The generative model $p(d, \mathbf{x}, y, \mathbf{z}_d, \mathbf{z}_x, \mathbf{z}_y)$ can be factored as

$$p_{\theta_x}(\mathbf{x}|\mathbf{z}_d, \mathbf{z}_x, \mathbf{z}_y) p_{\theta_d}(\mathbf{z}_d|d) p(\mathbf{z}_x) p_{\theta_y}(\mathbf{z}_y|y) p(d) p(y). \tag{52}$$

The priors over the labels $p(d)$ and $p(y)$ can be modeled as multinomial distributions whose statistics are inferred from the dataset and thus do not need parameterization. We can also avoid to parameterize $p(\mathbf{z}_x)$ by assuming $p(\mathbf{z}_x) = \mathcal{N}(\mathbf{o}, \mathbf{I})$.

The recognition model of DIVA is learned by considering the approximate factorization

$$q_\phi(\mathbf{z}_d, \mathbf{z}_x, \mathbf{z}_y|\mathbf{x}) = q_{\phi_{\mathbf{z}_d}}(\mathbf{z}_d|\mathbf{x}) q_{\phi_{\mathbf{z}_x}}(\mathbf{z}_x|\mathbf{x}) q_{\phi_{\mathbf{z}_y}}(\mathbf{z}_y|\mathbf{x}) \tag{53}$$

where $q_{\phi_{\mathbf{z}_d}}(\mathbf{z}_d|\mathbf{x})$, $q_{\phi_{\mathbf{z}_x}}(\mathbf{z}_x|\mathbf{x})$, and $q_{\phi_{\mathbf{z}_y}}(\mathbf{z}_y|\mathbf{x})$ are modeled as normal distributions, as well as the two posterior distributions $q_{\phi_d}(d|\mathbf{z}_d)$ and $q_{\phi_y}(y|\mathbf{z}_y)$ which act as auxiliary discriminators of domain and class label respectively.

The loss function considered when training DIVA with labeled samples is

$$\mathcal{F}_{\text{DIVA}}(d, \mathbf{x}, y) := \mathcal{L}(d, \mathbf{x}, y) \tag{54}$$

$$+ \alpha_d \mathbb{E}_{q_{\phi_{\mathbf{z}_d}}(\mathbf{z}_d|\mathbf{x})} \left[[\log q_{\phi_d}(d|\mathbf{z}_d)] \right] \tag{55}$$

$$+ \alpha_y \mathbb{E}_{q_{\phi_{\mathbf{z}_y}}(\mathbf{z}_y|\mathbf{x})} \left[[\log q_{\phi_y}(y|\mathbf{z}_y)] \right] \tag{56}$$

where

$$\mathcal{L}(d, \mathbf{x}, y) = \mathbb{E}_{q_{\phi_{\mathbf{z}_d}}(\mathbf{z}_d|\mathbf{x}) q_{\phi_{\mathbf{z}_x}}(\mathbf{z}_x|\mathbf{x}) q_{\phi_{\mathbf{z}_y}}(\mathbf{z}_y|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z}_d, \mathbf{z}_x, \mathbf{z}_y)] \tag{57}$$

$$- \beta_d D_{KL}(q_{\phi_{\mathbf{z}_d}}(\mathbf{z}_d|\mathbf{x}) \| p_{\theta_d}(\mathbf{z}_d|d)) \tag{58}$$

$$- \beta_x D_{KL}(q_{\phi_{\mathbf{z}_x}}(\mathbf{z}_x|\mathbf{x}) \| p(\mathbf{z}_x)) \tag{59}$$

$$- \beta_y D_{KL}(q_{\phi_{\mathbf{z}_y}}(\mathbf{z}_y|\mathbf{x}) \| p_{\theta_y}(\mathbf{z}_y|y)) \tag{60}$$

and where α_d, α_y, β_d, β_x, and β_y are tunable weighting terms. Indeed, similarly to other VAEs, the loss of the model can be extended to operate in a semi-supervised setting where the label y may be missing. This brings significant performance improvements when a large quantity of domain-labeled data is available.

Lastly, we can perform class discrimination of an unlabeled sample $\mathbf{x}$ by using the mean of the distribution $q_{\phi_{\mathbf{z}_y}}(\mathbf{z}_y|\mathbf{x})$ as input to the auxiliary class discriminator $q_{\phi_y}(y|\mathbf{z}_y)$. The entire class discrimination process does not require knowledge on the domain label d of the sample. Instead, many previous methods such as VFAE [88] and CVIB [98] do require domain information at inference time, which makes employing them in unseen domains unreliable or impossible.

By learning the disentanglement of domain, class, and residual the model addresses and prevents the impossibility of learning a disentangled representation in an unsupervised fashion for arbitrary generative models, a claim that is made by Locatello et al. [85] and Dai and Wipf [24].

4.2 DOMAIN ADAPTATION AND GENERALIZATION

Domain Adaptation aims to boost cross-domain performance by combining supervised learning on source domains and unsupervised learning on target domains.

Ganin and Lempitsky [43] propose Gradient Reversal. The approach leverages a novel gradient reversal layer that multiplies the gradient by a certain negative constant during backpropagation, resembling the behavior of adversarial training. The parameters of a deep feature extractor are thus encouraged to be non-informative about the domain membership of the input, which in turn promotes feature alignment between domains and boosts performance in the target domain.

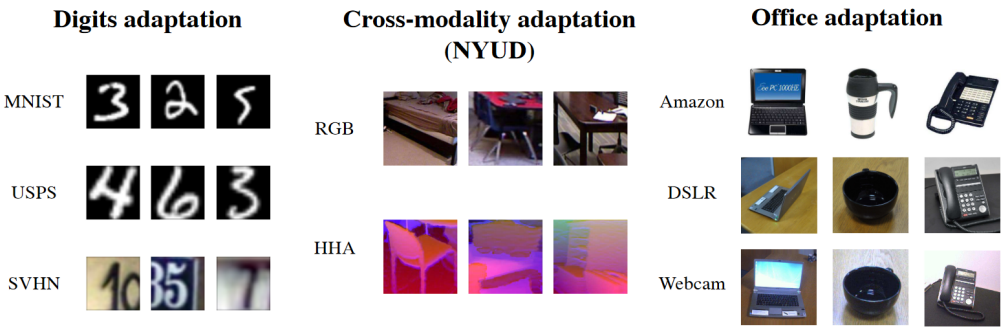


Figure 9: Examples of domain shift in images from three different settings [43].

Motivated by the observation that activations within a NN follow domain-dependent distributions, Mancini et al. [92] revisit batch normalization to provide domain-specific statistics and normalization. When dealing with target domains, they leverage a domain prediction network to find a suitable interpolation of the various domain-specific normalizations of the input. The approach can potentially provide reliable statistics that can generalize to unseen domains.

Domain Generalization approaches do not assume access to a large collection of unlabeled samples from the target domains and instead learn from multiple source domains to extract a domain-agnostic model that can then be applied to an unseen domain. Li et al. [82] propose a domain generalization method based on low-rank parameterized CNNs to obtain domain-agnostic features while avoiding overfitting.

4.3 CROSS-DOMAIN FEW-SHOT LEARNING

In FSL classification, models are challenged to learn new tasks based on few examples. Cross-Domain Few-Shot Learning (CDFSL) takes the FSL problem one step further, assuming that test tasks are subject to domain shift. Multiple interpretations of CDFSL are available in the literature. Works such as [97] assume that images from source and target domains share the same class distribution. In our work, we instead focus on the more challenging problem where the distribution of both classes and domains are shifted. Others such as [125] presume access to a large quantity of unlabeled data from the target domain. Instead, we aim to adapt to the unseen domain considering only a few data from the support set of the task, which is much more difficult.

While the literature has produced many interesting approaches in the trivial FSL setting [36, 117, 143], Chen et al. [19] recently highlight that meta-learning based FSL algorithms fail to outperform traditional pre-training and fine-tuning methods when dealing with test data that is subject to significant domain shift. Consequently, several new approaches have been recently developed to tackle the CDFSL problem.

Peng, Song, and Ester [112] propose *Combining Domain-Specific Meta-Learners* (CosML), where a set of meta-learners are trained to be later combined via a task-dependent weighted average in the parameter space, generating the initial parameters of a task-specific meta-learner which is then adapted to the novel FSL classification task at hand. The architecture of CosML consists of a set of domain-specific meta-learners on top of a shared, pre-trained feature extractor. During meta-training, some source domains are never observed during base learning to simulate unseen domains and are instead used to meta-optimize the quality of the generalization to unseen domains.

Representation fusion is the concept of unifying and merging information from different levels of abstraction within the layers of a deep NN. Adler et al. [2] recognize the effectiveness of representation fusion in the CDFSL setting and propose *Cross-domain Hebbian Ensemble Few-shot learning* (CHEF), which achieves representation fusion by employing an ensemble of Hebbian learners acting on different layers of a deep NN.

Kwon et al. [75] repurpose MAML checkpoints to solve new out-of-domain FSL classification tasks by acting on the gradient update rule during task adaptation based on information extracted from the support set.

Zhao et al. [157] argue that a naïve combination of existing domain adaptation and FSL methods fails to offer an effective solution to the CDFSL problem: existing domain adaptation methods assume that the target and source domains have identical label space, and global or per-class distribution alignment would have a detrimental effect on class separation and discriminativeness. Consequently, they propose a *Domain-Adversarial Prototypical Network* (DAPN), where the adversarial domain confusion objective is complemented by new losses that enforce source/target class discriminativeness, leading to both globally aligned distributions and well-separable class representations.

Tseng et al. [139] employ feature-wise transform during meta-training to simulate various distributions of image features that encourage learning representations with improved ability to generalize. The hyperparameters of the feature-wise transformation layers are meta-learned.

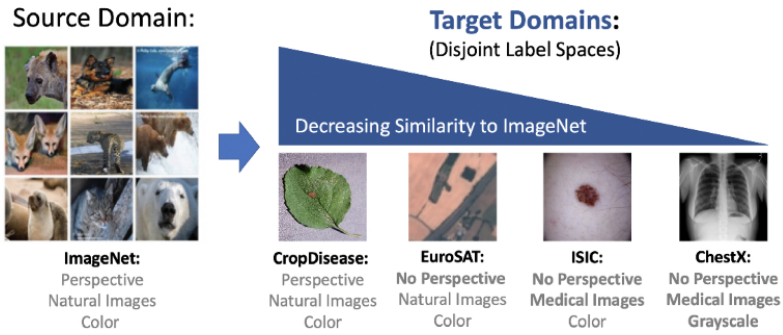


Figure 10: Images from BSCD-FSL [49].

Nevertheless, despite increasing efforts by recent works in the field, the problem of how to effectively meta-learn across multiple source domains while avoiding meta-overfitting remains an important challenge. Furthermore, in the case of CDFSL, the literature currently does not feature well-established evaluation benchmarks, with many existing ones only involving natural images [138, 139]. When evaluating on BSCD-FSL, a benchmark that contains both natural and non-natural images, Guo et al. [49] conclude that state-of-the-art meta-learning approaches are outperformed by earlier, shallow learning methods such as fine-tuning, with recent CDFSL techniques degrading performance.

5

PRE-TRAINING THE EMBEDDING

We provide a formulation of the CDFSL problem we want to tackle. In particular, we define a new benchmark for our experiments and build upon LEO to improve cross-domain performance. We then attempt to tackle the problem by leveraging a pre-trained embedding with useful properties.

5.1 PRELIMINARIES

To test our hypotheses, we first need to define a solid foundation on which to base our experiments. Therefore, we provide a rigorous formalization of the CDFSL problem we want to tackle, as well as define the model we want to build upon.

5.1.1 *Background*

To formally define our Meta Domain Adaptation problem, we expand on the task distribution view illustrated in Section 3.2.2. Like always, our objective is to learn a training procedure that can generalize to many tasks from a distribution of tasks. Without loss of generality, we assume that the loss function $\mathcal{L}$ is shared across all tasks in the distribution. To simplify the problem even further, we assume that the tasks we are dealing with are FSL classification tasks. An N-way, K-shot classification task is a tuple of two datasets $\mathcal{T} = \langle \mathcal{D}^s, \mathcal{D}^q \rangle$, where both the support set $\mathcal{D}^s$ and query set $\mathcal{D}^q$ contain pairs of inputs and desired outputs:

$$\mathcal{D}^s = \{\langle \mathbf{x}^s_{n,k}, n \rangle | 1 \leqslant n \leqslant N, 1 \leqslant k \leqslant K\}; \tag{61a}$$

$$\mathcal{D}^q = \{\langle \mathbf{x}^q_{n,k'}, n \rangle | 1 \leqslant n \leqslant N, 1 \leqslant k' \leqslant K'\}. \tag{61b}$$

The model adapts to the task by adjusting its parameters based on the support set, while the query set is used to evaluate the performance of the model on the task after the inner adaptation. Consequently, the difficulty of the task mainly depends on the values of N and K, i. e., the number of classes and the number of examples per class provided for the task at hand. The value of K' mostly affects the reliability of the estimation of the model's performance on a task.

In the case of single-domain FSL classification, the probability distribution over tasks $p(\mathcal{T})$ can be expressed as a Bayesian network by introducing latent variables $\mathbf{z}^c_n$, representing the classes of the tasks, sampled from a distribution over the latent class variables p^c:

$$\mathbf{z}^c_n \sim p^c(\mathbf{z}^c), \quad 1 \leqslant n \leqslant N. \tag{62}$$

Then, each data instance in the support and query set is sampled from a distribution over the data instances p^x that is conditioned on the corresponding class latent variable:

$$\mathbf{x}^s_{n,k} \sim p^x(\mathbf{x}|\mathbf{z}^c_n) \quad 1 \leqslant n \leqslant N, 1 \leqslant k \leqslant K; \tag{63a}$$
$$\mathbf{x}^q_{n,k'} \sim p^x(\mathbf{x}|\mathbf{z}^c_n) \quad 1 \leqslant n \leqslant N, 1 \leqslant k' \leqslant K'. \tag{63b}$$

However, since we are interested in Cross-Domain Few-Shot Learning, our FSL classification tasks must be from multiple different domains. In particular, we want to learn across-task knowledge ω that can generalize to unseen, possibly out-of-distribution domains. The definition of the task probability distribution $p(\mathcal{T})$ is therefore enriched with a new latent variable $\mathbf{z}^d$ representing the domain of the task that is sampled from a distribution over the latent domain variables p^d:

$$\mathbf{z}^d \sim p^d(\mathbf{z}^d). \tag{64}$$

The distribution of each data instance in the task is now also conditioned on the task-specific domain latent variable:

$$\mathbf{x}^s_{n,k} \sim p^x(\mathbf{x}_n|\mathbf{z}^c_n, \mathbf{z}^d) \quad 1 \leqslant n \leqslant N, 1 \leqslant k \leqslant K; \tag{65a}$$
$$\mathbf{x}^q_{n,k'} \sim p^x(\mathbf{x}_n|\mathbf{z}^c_n, \mathbf{z}^d) \quad 1 \leqslant n \leqslant N, 1 \leqslant k' \leqslant k'. \tag{65b}$$

Thus, the hyper-parameters N, K, K' along with the probability distributions p^c, p^d, p^x completely define the distribution of tasks:

$$p(\mathcal{T}) = p_{N,K,K'}(\mathcal{T}|p^c, p^d, p^x). \tag{66}$$

We can now define our training and test distributions of tasks. For simplicity, we assume that the hyper-parameters N, K, K' as well as the probability distribution over data instances $p^x(\mathbf{x}|\mathbf{z}^c, \mathbf{z}^d)$ are shared across all the possible task distributions:

$$p(\mathcal{T}) = p(\mathcal{T}|p^c, p^d). \tag{67}$$

Therefore, training and test distributions of tasks differ in either the shape of the distribution over class latents p^c, the distribution over domain latents p^d, or both. In particular, if we consider a training distribution p^c_{train} and test distribution p^c_{test} over class latents as well as a training distribution p^d_{train} and test distribution p^d_{test} over domain latents, we can identify four possible distributions:

	$p^c = p^c_{train}$	$p^c = p^c_{test}$
$p^d = p^d_{train}$	$p_{train,train}(\mathcal{T})$	$p_{test,train}(\mathcal{T})$
$p^d = p^d_{test}$	$p_{train,test}(\mathcal{T})$	$p_{test,test}(\mathcal{T})$

Our training distribution is $p_{train,train}(\mathcal{T})$, while we are particularly interested in the performance obtained by our model in $p_{test,test}(\mathcal{T})$, which features both unseen out-of-distribution classes and domains. The test distribution $p_{test,train}(\mathcal{T})$ arguably identifies the test distribution of traditional meta-

learning problems, where the only change from the training distribution is observed from the point of view of classes, while the domain distribution remains unchanged.

An interesting direction for us is to attempt to minimize the difference in performance between the in-domain distribution $p_{test,train}(\mathcal{T})$ and the out-of-domain distribution $p_{test,test}(\mathcal{T})$. Indeed, the test distributions over class and domain latents should cover areas of the latent spaces that, while potentially unseen during training, are nonetheless related to the areas covered by the training distributions, so that the meta-knowledge acquired during training has a chance of generalizing to the new distributions.

5.1.2 *Transductive vs Non-Transductive Meta-Learning*

In the FSL literature, we can identify two mutually exclusive settings: *transductive* and *non-transductive* learning. Non-transductive learning prohibits any type of access to query set information during adaptation, meaning that, for each prediction of a generic unlabeled query input $\mathbf{x}^q_{n,k'}$, the model can only rely on that query input, the meta-knowledge ω, and the support set $\mathcal{D}^s$ of the task. On the other hand, in transductive learning, we also have access to the inputs of the query set $\mathcal{D}^q$ of the task during adaptation. For instance, when computing statistics for normalization in the transductive setting, we can leverage the unlabeled inputs in the query set to obtain more accurate statistics. The non-transductive setting is more challenging, as we have access to less information.

Transductive learning features some arguably undesired properties, such as dependence on the size and distribution of the query set. Furthermore, the assumption of having access to a large set of unlabeled samples may be unrealistic when considering many real-world problems, e. g., it may violate privacy constraints. In light of these considerations, in our work, we decide to focus on the non-transductive setting.

5.1.3 *Addressing Meta-Batch Normalization*

In standard supervised learning, batch normalization layers behave differently during training and evaluation phase. In particular, during training, first and second order statistics $\hat{\mu}$ and $\hat{\sigma}^2$ of the activations x_i are computed across the batch $\{x_i\}_{i=1}^B$:

$$\hat{\mu} = \frac{1}{B}\sum_{i=1}^{B} x_i; \quad \hat{\sigma}^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \hat{\mu})^2. \tag{68}$$

The statistics are then used to normalize and match those of the standard normal distribution:

$$\hat{x}_i = \frac{x_i - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}} \tag{69}$$

where ϵ is a small constant value to avoid division by zero. Meanwhile, at evaluation time activations are normalized by leveraging estimation of the statistics computed at training time over multiple batches, which is usually more reliable.

Adapting the rationale of batch normalization to meta-learning is not trivial. In meta-learning, a batch contains a set of tasks, and each task, in turn, contains a set of images. Since we focus on the non-transductive setting, we can compute statistics using only images from the support sets of the tasks. For simplicity, we consider the case where each image produces a single activation for the normalization. Furthermore, we refer to the activation corresponding to the k-th image of class n in the support set of the b-th task in the batch as x^s_{bnk}.

A naïve way to implement meta-batch normalization would be to ignore the separation in tasks of the images and compute the statistics over all the images of the entire meta-batch, independently from the task they belong to:

$$\hat{\mu} = \frac{1}{BNK} \sum_{b=1}^{B} \sum_{n=1}^{N} \sum_{k=1}^{K} x^s_{bnk}; \quad \hat{\sigma}^2 = \frac{1}{BNK} \sum_{b=1}^{B} \sum_{n=1}^{N} \sum_{k=1}^{K} (x^s_{bnk} - \hat{\mu})^2. \tag{70}$$

The statistics can then be used to normalize the activation x of a generic image in the meta-batch:

$$\hat{x} = \frac{x - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}}. \tag{71}$$

In this case, we can also keep track of the statistics in the same way we do in supervised learning to later use them at evaluation time. We refer to this type of meta-batch normalization as *naïve normalization*.

However, when adapting batch normalization to meta-learning, we should take into account that, unlike traditional supervised learning, images are not i.i.d. globally, but only with respect to a specific task. We address the above by leveraging *task-specific normalization*. In task-specific normalization, we normalize the activations by only considering the statistics over the images of the task at hand:

$$\hat{\mu}_b = \frac{1}{NK} \sum_{n=1}^{N} \sum_{k=1}^{K} x^s_{bnk}; \quad \hat{\sigma}^2_b = \frac{1}{NK} \sum_{n=1}^{N} \sum_{k=1}^{K} (x^s_{bnk} - \hat{\mu}_b)^2. \tag{72}$$

The statistics can then be used to normalize the activation x_b of a generic image in the b-th task in the meta-batch:

$$\hat{x}_b = \frac{x_b - \hat{\mu}_b}{\sqrt{\hat{\sigma}^2_b + \epsilon}}. \tag{73}$$

We also leverage task-specific normalization at evaluation time, which means that we do not need to keep track of any statistics during the training phase. Task-specific normalization is a popular way to implement batch normalization in the meta-learning literature [16, 138]. In our meta-learning experiments, we leverage this type of normalization unless specified otherwise.

5.1.4 *Our Main Benchmark: Corrupted-Omniglot*

Following the mathematical formalization of our CDFSL classification problem, we now need to define a concrete benchmark to evaluate our models on. Initially, we are interested in a benchmark that both represents a moderate cross-domain challenge and is easy to handle in terms of computing power so that

we can comfortably evaluate our ideas on a simpler problem before moving on to harder ones. The *Omniglot* dataset [76] is a popular benchmark in the FSL literature. It consists of a collection of black-and-white square images displaying handwritten characters from 50 different alphabets. The classification problem is significantly easier than the ones that feature natural images. Moreover, by resizing the images to a resolution of 28x28, we manage to considerably speed up our experiments.

Each character in the dataset is represented by 20 different images portraying the same character written in different styles. When performing N-way FSL classification with Omniglot, tasks are usually dynamically generated by first sampling N different characters out of all the ones available in the dataset, thus determining the classes of the task. Then, for each character selected, some of its corresponding images are randomly selected to form the support set $\mathcal{D}^s$ of the task. Meanwhile, the images that end up not being selected are instead included in the query set $\mathcal{D}^q$ of the task. When splitting the dataset into training and test classes, we resort to the harder setup provided by Vinyals et al. [143], where the split is randomly performed at alphabet level, rather than character level. In this way, some alphabets are never seen during meta training, guaranteeing the out-of-distribution property of test classes.

Despite providing a reasonable benchmark for standard FSL classification, the Omniglot dataset is not an example of CDFSL classification. Furthermore, the latest performances obtained from state-of-the-art models on Omniglot are tremendously high [81], suggesting possible obsolescence. However, we can augment the dataset by applying many different *image corruptions* to the images. Image corruptions transform images by introducing some kind of perturbation to the image itself. A classic example of image corruption is Gaussian blur, which blurs the image by convolving the image with a Gaussian function. In the case of black-and-white images, Mu and Gilmer [99] provide us with a set of 15 readily implemented corruptions. By considering each corruption as a domain, we can augment the Omniglot dataset and repurpose it as *Corrupted-Omniglot* (or briefly C-Omniglot), a new benchmark for CDFSL classification.

C-Omniglot features 16 domains; one of them is the domain of uncorrupted images, which we refer to as *identity*. When sampling a task from C-Omniglot, the domain is randomly selected among the available ones and the images in both the support and query set of the task are corrupted accordingly, which conforms to the formal definition of the CDFSL classification problem we provide. The tasks in C-Omniglot also include the domain label d:

$$\mathcal{T} = \langle \mathcal{D}^s, \mathcal{D}^q, \mathrm{d} \rangle. \tag{74}$$

The information of the domain label may or may not be leveraged depending on the algorithm.

Finally, we also split the available domains into 10 training domains and 6 test domains, intending to generalize meta-knowledge extracted from the former to the latter. In our experiments, we consider 20-way, 1-shot classification tasks, which is a popular and challenging setting in the literature. Our query set always contains 15 images per class to provide a robust evaluation of the performance on the task after adaptation. The batch size during meta-learning, i. e., the number of tasks per meta-gradient step, is 8 unless specified otherwise.

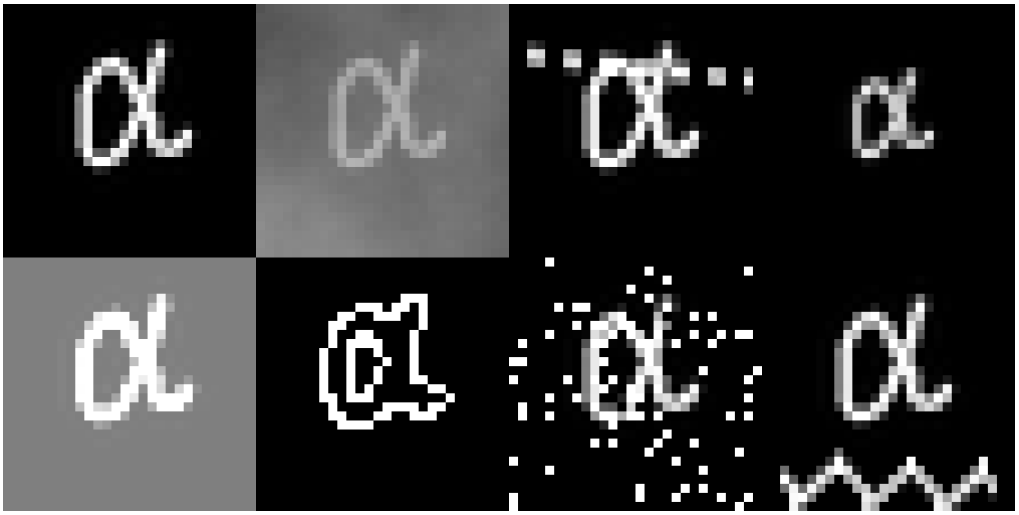


Figure 11: Examples of corruptions in C-Omniglot. The top row represents the training domains identity, fog, dotted line, and scale, while the bottom row represents the test domains brightness, canny edges, shot noise, and zigzag.

5.1.5 *Our Baseline Model: LEO*

To avoid starting from scratch, we decide to build upon LEO (Section 3.3.3.2), which features attractive qualities such as reduced computation costs compared to MAML. In particular, we extend LEO's architecture by including a convolutional embedder $g_{\phi_{emb}}$ that we use as input for the original LEO meta-learning pipeline[1]. Given a generic input image $\mathbf{x}$, we refer to its embedding $g_{\phi_{emb}}(\mathbf{x})$ as simply $\bar{\mathbf{x}}$.

Initially, we carry out numerous other experiments to gather insights on what affects the performance of our model. In particular, since the original LEO originally features a very complex architecture and many non-trivial regularizations, we analyze their impact by comparing it with a version of LEO that has a simpler architecture and fewer regularizations. Surprisingly, we find that many choices in the architectures such as the presence of a relation network g_{ϕ_r} right after the encoder g_{ϕ_e} and the stochastic nature of the latent parameter code $\mathbf{z}$ are not needed to achieve top performance on our benchmark. We can often even remove the entire encoding process from the architecture and initialize $\mathbf{z}$ with a constant value and still obtain results that are comparable with the original architecture. Furthermore, we observe that besides L2 regularization we can ignore all other forms of regularizations in our network without hurting performance. Finally, we find that a single gradient step in the parameter code space and no fine-tuning of the parameters after latent code optimization is enough when adapting to a new task. Thanks to these discoveries, we can consider a simpler LEO architecture, shown in Fig. 12, that is faster to train, is easier to understand, and better lends itself to further extensions. We define *Baseline LEO* as the model obtained by meta-training this simpler LEO architecture from scratch.

When considering 20-way, 1-shot tasks with test classes from Corrupted-Omniglot, Baseline LEO achieves a classification accuracy of 0.95 in training do-

1 In contrast, Rusu et al. [124] consider a pre-trained embedder that is external to the model.

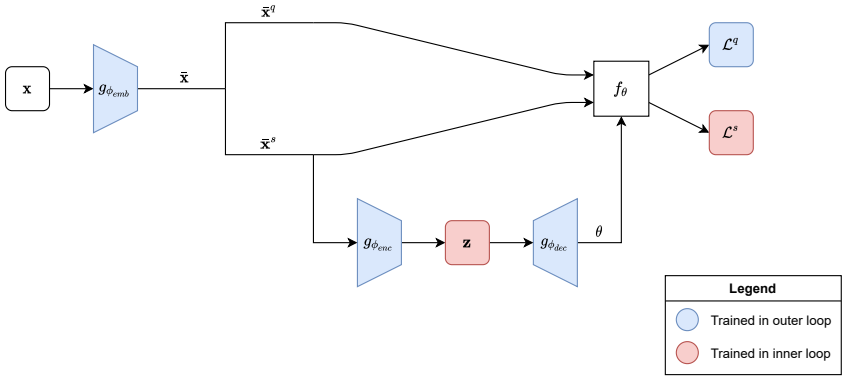


Figure 12: Baseline LEO architecture.

mains and 0.76 in test domains. Since our goal is to find a novel solution that is capable of achieving comparable training and test domains performances, we realize that there is still large room for improvement. In the following sections, we compare the performance of new models with respect to Baseline LEO to quantify their capabilities when generalizing to unseen domains.

5.2 A GOOD EMBEDDING

As seen in Section 4.1, Raghu et al. [115] and Tian et al. [137] argue that the performance of meta-learning procedures heavily depends on the quality of the embedding. We thus focus on obtaining a high-quality embedding by leveraging existing techniques and pre-training on images from the training dataset. We also evaluate disentanglement and interpretability in the embeddings in multiple ways.

5.2.1 *Pre-Training DIVA*

One of the most trivial ways to implement a disentangled and interpretable embedding in LEO is to first learn an embedder by using an existing approach in the literature that promotes our desired properties. The parameters of the embedder can then be reloaded as the parameters of LEO's embedder so that the original LEO architecture can meta-learn on the high-quality embedding.

Our go-to disentangling model is DIVA (Section 4.1.4.3), since it offers continuous, stochastic latent spaces representing class and domain information that can potentially generalize to unseen classes and domains, something that is particularly useful in the cross-domain meta-learning setting. In DIVA, disentanglement and interpretability are provided by the variational encoders $q_{\phi_{z_d}}(\mathbf{z}_d|\mathbf{x})$, $q_{\phi_{z_x}}(\mathbf{z}_x|\mathbf{x})$, and $q_{\phi_{z_y}}(\mathbf{z}_y|\mathbf{x})$, partitioning the embedding space in domain, residual, and class information. By leveraging DIVA's embedding during meta-learning, we can effectively evaluate the quality of the generalization for our scope.

Nonetheless, since DIVA is trained via standard (semi) supervised learning, we first need to define a dataset to perform our pre-training on, as well as identify other baseline pre-training models to compare with. Moreover, we want to observe a high-quality disentanglement and interpretability of the embedding

on this new dataset before proceeding with the meta-learning phase. Finally, we might decide to extend the DIVA model to improve the quality of the embedding we obtain during pre-training.

5.2.2 *Corrupted-Omniglot PT*

We define our supervised-learning pre-training task by considering a dataset of labeled images from C-Omniglot, thus assuming that features that are useful for traditional supervised classification tasks are also suitable in FSL classification. Indeed, we do not consider images coming from test classes or domains during pre-training, preserving their validity as unseen samples when evaluating our meta-learning model. Our pre-training dataset *Corrupted-Omniglot Pre-Training* (briefly C-Omniglot PT) is in the form

$$\mathcal{D}^{\text{PT}}_{\texttt{train}} = \{(\mathbf{x}_i, y_i, d_i)\}_{i \in I_{\texttt{train}}} \tag{75}$$

where y_i and d_i are class and domain labels of the image $\mathbf{x}_i$. For each training class and domain, 19 of the 20 available images in C-Omniglot end up in $\mathcal{D}^{\text{PT}}_{\texttt{train}}$, whereas the remaining one is instead included in an evaluation dataset $\mathcal{D}^{\text{PT}}_{\texttt{test}}$ to assess the performance of the pre-trained model after pre-training. Moreover, since DIVA does not rely on the domain label when performing class discrimination, we also include in $\mathcal{D}^{\text{PT}}_{\texttt{test}}$ images from training classes and test domains to evaluate the ability of DIVA to generalize to unseen domains.

In the end, we obtain a pre-training discrimination task where each image is assigned to one of 1028 possible class labels and one of 10 possible domain labels.

5.2.3 *Other Pre-Training Baselines*

To thoroughly evaluate the advantages provided by DIVA's disentangled and interpretable representation, we also consider a variety of baseline models to compare with. Firstly, we consider a simple NN that is pre-trained to discriminate the class label of the images in C-Omniglot PT. We refer to the model in question as *Classifier*. The architecture of Classifier consists of a linear classifier positioned on top of a deep convolutional feature extractor. Unlike DIVA, Classifier does not leverage the information of domain label d provided in C-Omniglot PT. Furthermore, its sole objective is to provide accurate class discrimination, while DIVA also discriminates the domain labels d and learns to reconstruct the images from the latents.

Secondly, we consider a standard VAE (see 4.1.4) that is pre-trained to maximize the likelihood of the images in C-Omniglot PT. We refer to the model in question as simply *VAE*. Indeed, VAE does not make any use of class labels y and domain labels d and instead only focuses on reconstructing the input image $\mathbf{x}$ as accurately as possible. Consequently, unlike DIVA and Classifier, VAE does not perform class discrimination.

Concerning the architecture of our models, we obtain a fair comparison between them by assuming the following:

- in DIVA, each variational encoder features the same structure, which is a fully connected layer on top of a deep CNN;
- in Classifier, the feature extractor is a deep CNN with the same structure as the one found in DIVA's variational encoders;
- in VAE, the single variational encoder shares the same structure as the ones found in DIVA.

5.2.4 *Quality Evaluation of the Embedding*

One of the simplest ways to evaluate the quality of the obtained embedding is considering the class discrimination accuracy obtained during pre-training. When confronting the performance of DIVA and Classifier in Table 1, we can observe a small improvement of the former with respect to the latter in both training and test domains images. This suggests that DIVA is indeed able to make good use of the additional information provided by the domain label in our dataset. Furthermore, the performance gap accentuates in the case of test domains, which hints at a better generalization of DIVA to unseen domains with respect to the Classifier baseline. However, it is important to note that the performances of both models drop dramatically in the case of test domains, suggesting that there is still much room for improvement when generalizing to unseen domains.

Model	Training Domains	Test Domains
DIVA	0.85	0.33
Classifier	0.83	0.31

Table 1: *C-Omniglot PT.* Classification accuracy for DIVA and Classifier models on test images and training/test domains.

Nonetheless, focusing only on performances may not provide us with a correct perception of the quality of our embedding. In particular, it is hard to evaluate the quality of the disentanglement and interpretability when only looking at these metrics. Another valuable way to evaluate the quality of the disentanglement in the embedding is by inspecting the values of correlation between pairs of latents. More specifically, we can compute a correlation matrix among the various latent activations. Since DIVA's graphical model assumes statistical independence between all three possible latent pairs, we expect to observe correlation values that are very close to 0 in the entries that correspond to pair of activations that reside in different latent spaces. In Fig. 13, we observe that this is indeed the case for most pairs of activations considered, with a few exceptions that exhibit moderate correlation.

As a Variational Autoencoder, DIVA also features a variational decoder

$$p_{\theta_x}(\mathbf{x}|\mathbf{z}_d, \mathbf{z}_x, \mathbf{z}_y) \tag{76}$$

to reconstruct the input image from the latents. We can therefore inspect the reconstructions obtained by considering various subsets of latents (e. g., $\{\mathbf{z}_x, \mathbf{z}_y\}$)

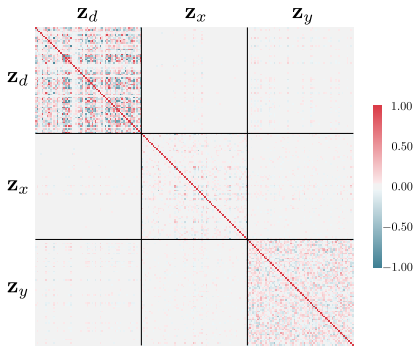


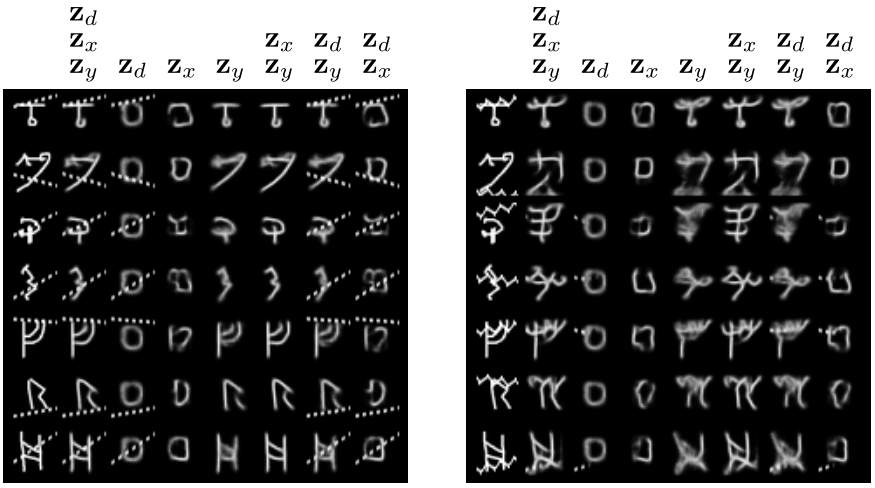
Figure 13: *C-Omniglot PT.* Correlation matrix of latent activations in DIVA. The blocks on the diagonal represent intra-latent correlation and are not matter of concern when pursuing disentanglement

to verify whether the embedding is disentangled and interpretable. The value of latents that are not considered during reconstruction is set to the mode of their prior distribution before being fed as input to the variational decoder. We thus expect to observe reconstructed images that, while different from the original, still look like something that would reasonably appear in our dataset. In our reconstructions, we expect the reconstructed image

- to be very similar to the original when reconstructing from $\{\mathbf{z}_d, \mathbf{z}_x, \mathbf{z}_y\}$, since all types of information are considered in this case;
- to feature the same scribble of the original when reconstructing from $\{\mathbf{z}_x, \mathbf{z}_y\}$, but also to display a different image corruption since $\mathbf{z}_d$ is not considered;
- to feature the same character and image corruption of the original when reconstructing from $\{\mathbf{z}_d, \mathbf{z}_y\}$, but also present a different writing style since $\mathbf{z}_x$ is not considered;
- to feature the same image corruption (and, in principle, the same writing style, but this may be difficult to interpret in this case) when reconstructing from $\{\mathbf{z}_d, \mathbf{z}_x\}$, but also display a different character since $\mathbf{z}_y$ is not considered.

The obtained reconstructions are shown in Fig. 14. The quality of disentanglement and interpretability in the reconstructions seems good when observing images from training domains. We can observe how reconstructing from $\{\mathbf{z}_x, \mathbf{z}_y\}$ has the only effect of completely removing the corruption from the image, which is particularly evident in the case of domains such as dotted line. Therefore, $\mathbf{z}_d$ seems successful in capturing all and only domain information, which is exactly what we want. Moreover, the absence of corruption suggests that the identity domain is encoded by DIVA near the mode of the latent space. The behavior is not surprising, as the identity domain is a special type of domain that does not corrupt the image, making it a suitable "latent neighbor" for many other domains. Furthermore, we notice that reconstructing from $\{\mathbf{z}_d, \mathbf{z}_x\}$ only modifies the character part of the image by transforming the scribble into a shape resembling a rounded edge box shape. Consequently, even in this case, we observe the correct behavior as $\mathbf{z}_y$ captures class information. The resulting box-like character can be explained by considering its

similarity to many existing characters, e. g., the letter "o" in the Latin alphabet. Finally, when reconstructing from $\{\mathbf{z}_d, \mathbf{z}_y\}$, we observe that changing the value of the latent variable does not alter the type of corruption and class displayed in the reconstruction. Instead, what seems to change is the writing style of the character. Indeed, this is exactly the behavior we were hoping for in this case, since it proves that $\mathbf{z}_x$ contains residual information that does not represent class nor domain.



(a) Training domain dotted line. (b) Test domain zigzag.

Figure 14: *C-Omniglot PT.* Reconstructions of the images in DIVA when considering various subsets of latents. Original images are shown in the leftmost column.

Unfortunately, the disentangling and interpretability of the embedding do not generalize to the test domains. In particular, domain information from unseen domains is often encoded as class information, making the original character unrecognizable. This is especially noticeable in the case of the domain zigzag. Nevertheless, the disentangled reconstructions of images from the brightness and rotate domains seem to be satisfactory, perhaps thanks to the similarity of such domains to fog and rotate respectively. Indeed, the poor generalization capabilities were to be expected. Like many variational autoencoders, DIVA is also likely to struggle when dealing with out-of-distribution samples, which is confirmed by considering the poor class discrimination performance and reconstruction quality in test domains we have obtained in our experiments.

For completeness, we also analyze the embedding provided by leveraging a convolutional latent space in DIVA, i. e., by replacing the fully connected layer found at the end of the variational encoders with a convolution. We refer to this type of model as *DIVA Conv*. Comparing the accuracy obtained in class prediction with our original DIVA architecture in Table 2 suggests an improvement in the quality of the embedding, as we obtain marginally better performance for test domains images. However, in this case, the correlation matrix in Fig. 15 exhibits a high amount of correlation between latents, which is undesirable as it is a sign of the absence of disentanglement. Concerning the reconstructions in Fig. 16, we can observe some imperfections in the disentangling capability of DIVA Conv, even when only looking at images from training domains. For

instance, we can observe how information about the dotted line is distributed between $\mathbf{z}_d$ and $\mathbf{z}_x$, while in the case of correct behavior we would expect $\mathbf{z}_d$ to fully capture the information. Moreover, we also observe a perturbation in the background of the image when not considering $\mathbf{z}_d$ in the reconstruction. This may be caused by the model encoding the fog domain near the latent space origin, which should not be a source of concern. Interestingly, not considering $\mathbf{z}_y$ in the reconstruction produces a series of horizontal lines in the reconstruction itself. Many characters include multiple horizontal lines in their shape, such as the letter "E".

Model	Training Domains	Test Domains
DIVA	0.85	0.33
DIVA Conv	0.85	0.35

Table 2: *C-Omniglot PT.* Classification accuracy for DIVA and DIVA Conv models on test images and training/test domains.

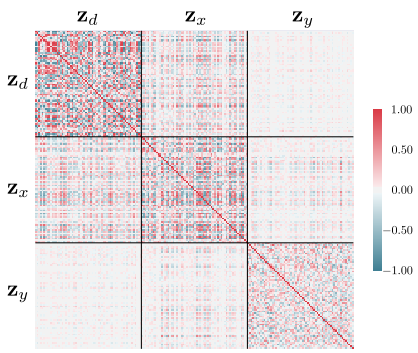


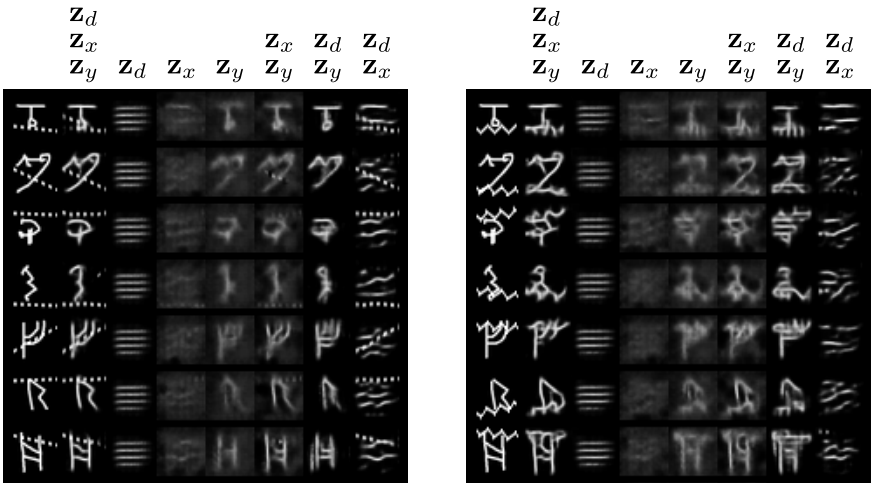
Figure 15: *C-Omniglot PT.* Correlation matrix of DIVA Conv's latent activations. The blocks on the diagonal represent intra-latent correlation and are not matter of concern when pursuing disentanglement

5.2.5 *Refining the Quality of the Embedding: MI Minimization*

DIVA has provided us with an embedding that is disentangled and interpretable when dealing with images from training domains. However, it also struggles when dealing with data from test domains, treating domain information as class information. We thus aim to refine our disentanglement by introducing further regularization into our model. In particular, we do so by minimizing the *Mutual Information* (MI) between pairs of activations from different latents. Mutual Information is a fundamental quantity for measuring the relationship between two variables. It quantifies the dependence of two random variables X_1, X_2 with the formula

$$I(X_1, X_2) = D_{KL}(\mathbb{P}_{X_1X_2} \| \mathbb{P}_{X_1} \otimes \mathbb{P}_{X_2}) \tag{77}$$

where $\mathbb{P}_{X_1X_2}$ is the joint probability distribution and where $\mathbb{P}_{X_1}$ and $\mathbb{P}_{X_2}$ are the marginals. One of the most interesting properties of MI is that it is equal to



(a) Training domain dotted line. (b) Test domain zigzag.

Figure 16: *C-Omniglot PT.* Reconstructions of the images in DIVA Conv when considering various subsets of latents. Original images are shown in the leftmost column.

0 if and only if the two variables are independent. In contrast, correlation, another popular metric for modeling dependence, does not satisfy the above: two variables can have zero correlation while still being dependent. Unfortunately, MI is notoriously difficult to compute, with exact computation only tractable for a limited family of problems. Therefore, we rely on the *Mutual Information Neural Estimator* (MINE) [8], a parametric approach to estimate MI using deep NNs. The method exploits the bound $I(X, Z) \geqslant I_\Theta(X_1, X_2)$, where $I_\Theta(X_1, X_2)$ is the *neural information measure* defined as

$$I_\Theta(X_1, X_2) = \sup_{\theta \in \Theta} \mathbb{E}_{\mathbb{P}_{X_1 X_2}} [T_\theta(X_1, X_2)] - \log \left(\mathbb{E}_{\mathbb{P}_{X_1} \otimes \mathbb{P}_{X_2}} \left[e^{T_\theta(X_1, X_2)} \right] \right) \quad (78)$$

with T_θ deep NN parameterized by θ. In practice, samples from the product of marginals $\mathbb{P}_{X_1} \otimes \mathbb{P}_{X_2}$ are obtained by shuffling the samples from the joint distribution along the batch axis. The objective can then be maximized via gradient ascent to find a set of parameters θ^* that provides a sharp lower bound of the true MI:

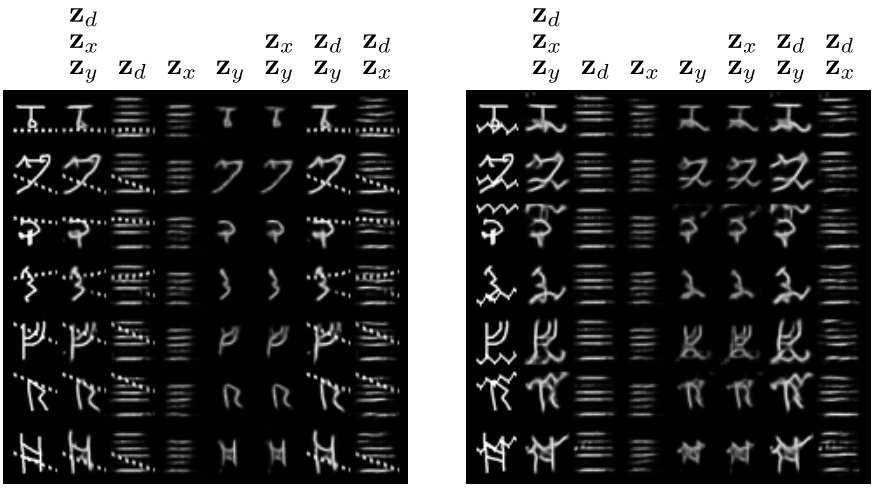
$$\widehat{I(X_1, X_2)} = \mathbb{E}_{\mathbb{P}_{X_1 X_2}} [T_{\theta^*}(X_1, X_2)] - \log \left(\mathbb{E}_{\mathbb{P}_{X_1} \otimes \mathbb{P}_{X_2}} \left[e^{T_{\theta^*}(X_1, X_2)} \right] \right). \quad (79)$$

We therefore employ three different NNs $T_{\theta_{dx}}$, $T_{\theta_{dy}}$, and $T_{\theta_{xy}}$ to estimate the mutual information between the pairs $(\mathbf{z}_d, \mathbf{z}_x)$, $(\mathbf{z}_d, \mathbf{z}_y)$, and $(\mathbf{z}_x, \mathbf{z}_y)$ respectively. While MI as a KL divergence is potentially unbounded, we encounter problems when training the MI estimators using an unbounded final output function, e. g., linear. In particular, the networks tend to find a local optimum that leads to the uninformative lower bound 0. For this reason, we decide to bound the outputs of the networks using a sigmoid activation function. In this case, though it is unfortunately impossible for us to obtain lower bounds greater than 1, we at least manage to surpass 0. Indeed, all three networks reach the maximum possible lower bound, or close to it. This is a clear indication that none of our latents are truly independent from the others, despite correlation values and reconstructions seemingly suggesting otherwise. This is

not surprising, as zero correlation and disentangled image reconstructions are necessary but not sufficient conditions for independence.

Nonetheless, by having access to a MI estimator, we can now extend the loss in DIVA to explicitly minimize MI between the three pairs of latents. For each pair of latent $(\mathbf{z}_a, \mathbf{z}_b)$, we add a weighted term $\widehat{I(\mathbf{z}_a, \mathbf{z}_b)}$ to the loss. We then proceed to train both DIVA and the MI estimators in an alternating manner, effectively rendering the MI minimization problem a minimax problem. When maximizing MINE to obtain a sharp bound, we keep DIVA's parameters fixed and train the MI estimators, while the reverse happens when minimizing MINE with the intent of obtaining independent latents. We find that for each gradient step performed by DIVA, 5 gradient steps are needed by MINE to properly address the shift in the distribution of the latents and re-adapt. In our experiments, introducing MINE minimization significantly reduces the lower bound of the MI computed by MINE, which is the desired behavior.

Unfortunately, we find that the regularization generally does not improve class accuracy in DIVA. However, leveraging MINE minimization in DIVA Conv exhibits higher quality reconstructions of training domains images, as shown in Fig. 17.



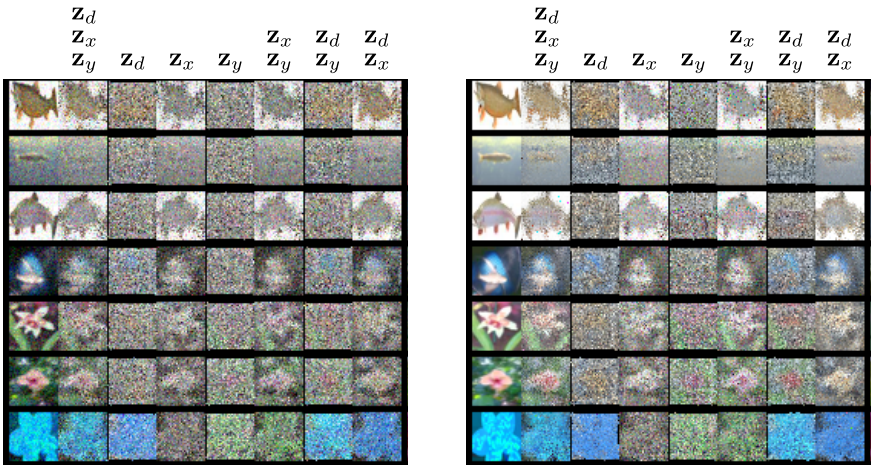
(a) Training domain dotted line.

(b) Test domain zigzag.

Figure 17: *C-Omniglot PT.* Reconstructions of the images in DIVA Conv with MINE minimization when considering various subsets of latents. Original images are shown in the leftmost column.

5.2.6 *DIVA and Colored Images*

Before transitioning to testing the obtained embeddings in meta-learning, we decide to evaluate DIVA's disentanglement capabilities on colored images, i. e., images that feature 3 input channels representing red, green, and blue light. The cross-domain natural images are obtained by augmenting the CIFAR dataset [73] with a collection of corruptions from various online repositories[17, 65, 99], obtaining *C-CIFAR* and, by extension, *C-CIFAR PT*. The reconstructions are shown in Fig. 18.



(a) Training domain gaussian noise. (b) Test domain brightness.

Figure 18: *C-CIFAR PT.* Reconstructions of natural images in DIVA. Original images are shown in the leftmost column.

Unfortunately, DIVA is not able to disentangle domain and class information in the natural image case. For instance, we can observe how both $\mathbf{z}_d$ and $\mathbf{z}_x$ erroneously store a large quantity of class information, such as the shapes and colors of the original image. DIVA's failure in this field strongly suggests that rethinking how we obtain disentanglement and interpretability of the embedding will be necessary when scaling to harder problems.

5.3 LEVERAGING THE EMBEDDING IN LEO

We have managed to obtain a disentangled and interpretable embedding by pre-training DIVA on images from C-Omniglot. We now want to verify whether such embedding is also reusable, i. e., generalizes well to unseen classes and domains. To do so, we reload in our LEO architecture the embedders found in DIVA and the other baseline pre-training models to observe their effects in our CDFSL classification setting. In particular, if our hypothesis is true, we expect to observe significantly higher performance for test domain tasks when leveraging DIVA's variational encoders.

5.3.1 *Identifying the Embedding in the Pre-Trained Models*

Firstly, we need to identify the part of the model we are going to reload as LEO's embedder for each one of our pre-trained models. In the case of DIVA, there are various possible options to choose from when defining the embedding. For instance, we may decide to reload all three variational encoders and consider their concatenated output as our embedding, or only consider the variational encoder that captures class information. We opt for the second alternative if not specified otherwise, since in principle domain and residual information are completely useless when discriminating the class of the images. Moreover, we might choose to reload the entire variational encoder or only a portion of it, such as its deep convolutional feature extractor. This is because the harsh dimensionality reduction performed by the final fully con-

nected layer could act as a disadvantageous bottleneck when meta-learning. We refer to the embedding provided by the variational encoder as a whole as *DIVA Latents* and to the embedding provided by the convolutional feature extractor as *DIVA Features*. In contrast, our VAE model only includes a single variational encoder. Since the structure of the variational encoder is the same as the ones found in DIVA, we use the same rationale to refer to define *VAE Latents* and *VAE Features*. Finally, concerning the Classifier model, we easily identify the embedding as the deep convolutional feature extractor preceding the fully connected classifier layer.

5.3.2 *Pre-Training Normalization*

Concerning the type of meta-batch normalization in the embedding, we have seen that two possible choices are naïve and task-specific normalization. However, in this case, since our embeddings are pre-trained, we can decide to rely on the running statistics $\bar{\mu}$, $\bar{\sigma}$ tracked during the supervised learning phase:

$$\hat{x} = \frac{x - \bar{\mu}}{\sqrt{\bar{\sigma}^2 + \epsilon}}. \tag{80}$$

We refer to this type of batch normalization as *pre-training normalization*. In this instance, the embedding obtained from an image is always the same, independently from the other images in the batch of tasks. Moreover, since changing the parameters of the embedding would skew the distribution of the activations, thus rendering the pre-training running statistics meaningless, we cannot train the embedding further in this case.

5.3.3 *Experiments and Results*

We identify 5 possible options for our embeddings and 3 possible ways to implement batch normalization in meta-learning, leading to 15 possible unique configurations for our experiments. In practice, some of them are discarded as we believe they do not provide meaningful insight and address our limited compute resources. To provide a fair comparison with the other experiments, the meta-trained embedding is a deep convolutional feature extractor and its structure is the same as the reloaded feature extractors found in the pre-trained models.

When meta-training LEO with a pre-trained embedding, we decide to keep the parameters of the embedding fixed. The reason for this is manifold. Firstly, modifying the parameters of the embedding is undesirable when using pre-training normalization. Furthermore, we currently do not have access to a meta-training procedure that preserves the unique characteristics of our embeddings. The standard inner-outer loop meta-learning algorithm would most definitely neutralize the stochastic nature of the variational encoders or the disentanglement and interpretability provided by DIVA.

The results are available in Table 3. We can observe that the embeddings provided by VAE perform far worse than the other models. This is perhaps to be expected since VAE does not make any use of the class and domain information when learning the embedding, which leads to both sub-optimal capture of

class information and unnecessary presence of domain information. Furthermore, we observe a sharp general drop in performance when comparing DIVA Latents to DIVA Features. This suggests that the dimensionality bottleneck caused by the last fully connected layer in the variational encoder erroneously filters useful class information.

When comparing the various types of batch normalization, task-specific normalization stands out with respect to the naïve and pre-training normalization. In the case of performance in training domains, we notice a sizeable drop in performance. This is probably because that the running statistics computed over the entire training dataset of images are much more reliable than the ones obtained by only considering the support set of the task. Nonetheless, in the case of test domains, there is a general improvement. Indeed, while running statistics may be more reliable, they are only suitable when dealing with images from training domains. Instead, when dealing with images from test domains, task-specific statistics are far superior. Interestingly, the improvement is particularly noteworthy when using the DIVA embeddings. Perhaps this is a sign that a disentangled embedding capturing only class information benefits greatly from high-quality batch normalization statistics.

Unfortunately, when comparing the pre-trained embeddings with Baseline LEO, the latter significantly outperforms the former, especially when it comes to tasks from test domains. Interestingly, this suggests that the inner-outer loop algorithm exhibits at least some domain generalization capabilities.

We also decide to examine the effects of meta-learning the embedding starting from a pre-trained initialization. Interestingly, it seems that while most embeddings reach a common plateau in terms of train domains performance, the performance in test domains is negatively affected by the amount of pre-training performed. For instance, reloading earlier epochs or considering models only trained on the identity domain seems to boost performance in test domains. This suggests that pre-training actually hurts domain generalization capabilities in meta-learning and that we should instead strive to meta-learn the embedding from scratch. We thus shift our focus away from the pre-training of a good embedding to concentrate on possible extensions of the LEO algorithm that encourages disentanglement and interpretability of the embedding during meta-learning.

5.4 ORACLES

We employ oracle models, i. e., models trained on images and tasks from test domains. Our goal is to understand if and how we should leverage a latent representation of the domain of the task and to determine an upper bound on the performance in test domains.

5.4.1 *Oracle Pre-Training and Evaluation*

Before focusing our attention on new architecture ideas that encourage disentanglement and interpretability during meta-learning, we would first like to gather additional information on what seems to work best when dealing with CDFSL tasks. We could argue that the reason DIVA fails in providing us with

Embedding	Naïve BN		Pre-Training BN		Task-Specific BN	
	Training Domains	Test Domains	Training Domains	Test Domains	Training Domains	Test Domains
Classifier	0.94	0.66	0.95	0.63	0.88	0.64
DIVA Features	0.94	0.66	0.95	0.65	0.91	0.72
DIVA Latents	0.89	0.60	0.89	0.55	0.84	0.65
VAE Features	0.67	0.45	—	—	—	—
VAE Latents	0.41	0.35	—	—	—	—
Meta-Trained	—	—	—	—	0.95	0.76

Table 3: *C-Omniglot, 20-ways, 1-shot.* Classification accuracy using different embeddings with various batch normalization methods on test classes and training/test domains.

a good embedding for CDFSL is due to its poor reusability, which is suggested by many factors such as the insufficient quality of the reconstructions in test domains. However, coming up with a method that can provide an embedding that is at the same time reusable, disentangled, and interpretable is a very challenging task that, to our best of knowledge, is yet to be properly addressed. Therefore, we want to understand how and whether we should leverage such a powerful embedding. To do so, we resort to a simplified setting where we are allowed to train our models on images and tasks from test domains. In this way, we can effortlessly obtain such an embedding by pre-training DIVA on both training and test domains images without spending time on coming up with a model that can generalize to unseen domains. We refer to a DIVA trained on images from both training and test domains as *Oracle DIVA*. For completeness, we also consider a Classifier model trained on images from both training and test domains, which we refer to as *Oracle Classifier*.

Pre-Training performances of the oracles are shown in Table 4. We can observe how Oracle DIVA manages to outperform Oracle Classifier in most domains. Moreover, we want to make sure that Oracle DIVA provides us with a disentangled and interpretable embedding, especially when dealing with the test domains. The reconstructions in Fig. 19 display the correct behavior, with $\mathbf{z}_d$ and $\mathbf{z}_y$ capturing domain and class information respectively.

We also analyze the latents using *t-distributed Stochastic Neighbor Embedding* (t-SNE) [89], a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two-dimensional map. In t-SNE, datapoints that are close to each other in the original high-dimensional space are encouraged to end up close to each other in the two-dimensional space as well. Similarly, datapoints that are far from each other in the original space are also likely to be far from each other in the two-dimensional space.

In our case, the data we want to visualize is a collection of latents obtained from Oracle DIVA when encoding images from one of 20 possible classes and one of the 16 available domains. When visualizing the latents of Oracle DIVA with t-SNE, we expect to observe different behaviors depending on the type of the latent we consider. In particular, if we want our latents to be disentangled

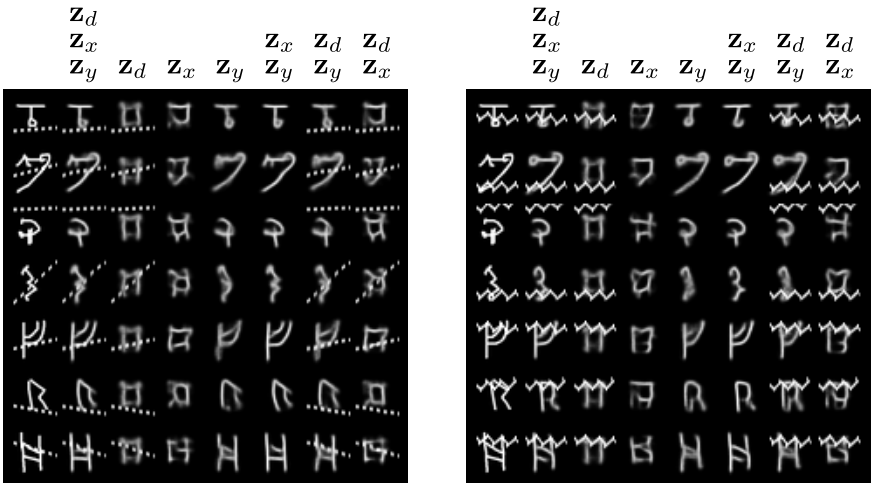
Domain	Oracle DIVA	Oracle Classifier
Dotted Line	0.85	0.84
Fog	0.85	0.84
Glass Blur	0.76	0.70
Identity	0.87	0.86
Scale	0.84	0.82
Shear	0.84	0.83
Shot Noise	0.83	0.81
Spatter	0.85	0.84
Stripe	0.85	0.85
Translate	0.82	0.84
Avg Training Domains	0.84	0.82
Brightness	0.85	0.85
Canny Edges	0.82	0.78
Impulse Noise	0.83	0.80
Motion Blur	0.83	0.81
Rotate	0.79	0.77
Zigzag	0.85	0.83
Avg Test Domains	0.83	0.81

Table 4: *C-Omniglot PT.* Domain-Specific classification accuracy for the oracle models during pre-training.

and interpretable, then the visualization of $\mathbf{z}_d$ should show 16 discernible clusters, with each one containing all the datapoints from a certain domain, and no same-class clusters. Analogously, the visualization of $\mathbf{z}_y$ should display 20 different clusters, each one containing all the datapoints from a certain class, and no same-domain clusters. The t-SNE visualizations are available in Fig. 20. In the case of $\mathbf{z}_d$, we can observe that the behavior is generally correct, with some misclustered datapoints and few overlapping clusters. Interestingly, we observe that some domains such as dotted line and zigzag are particularly near to each other, which is reasonable considering their similarity. Regarding $\mathbf{z}_y$, the behavior is mostly the desired one, though unfortunately, some small domain clusters appear, such as dotted line, zigzag, and stripe.

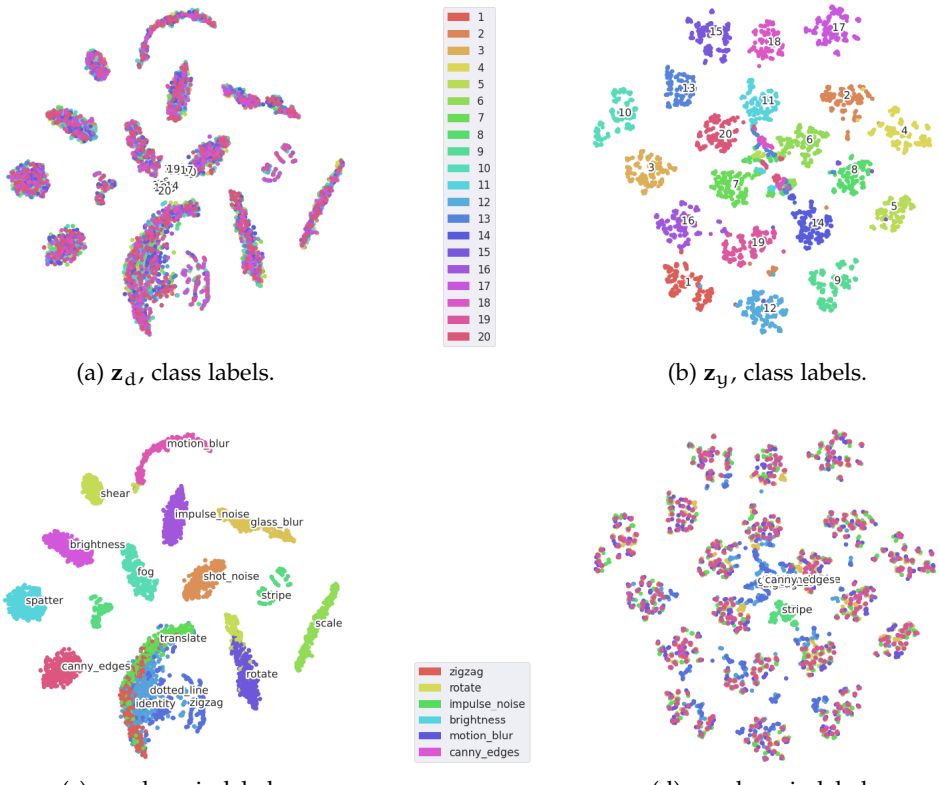
5.4.2 *Experiments and Results*

After pre-training our oracle models, we first reload the embedder they provide in our LEO architecture as already seen in Section 5.3. Because we want to observe the benefits of the oracles solely from the point of view of disentanglement, we still meta-train LEO on train domains only. We thus keep the parameters of our embedder fixed when meta-learning to preserve the domain



(a) Train domain dotted line. (b) Test domain zigzag.

Figure 19: Reconstructions of the images in Oracle DIVA. Original images are shown in the leftmost column. The other columns contain image reconstructions from the subset of latents denoted above.



(a) $\mathbf{z}_d$, class labels. (b) $\mathbf{z}_y$, class labels.

(c) $\mathbf{z}_d$, domain labels. (d) $\mathbf{z}_y$, domain labels.

Figure 20: *C-Omniglot PT.* t-SNE plots of $\mathbf{z}_d$ and $\mathbf{z}_y$ in Oracle DIVA with class/domain labels. Centroids are identified in the plot with the name of the label.

generalization capabilities of the oracles. Moreover, we also consider a Meta-Oracle, i. e., LEO meta-trained from scratch on both training and test domains.

The results are shown in Table 5. Interestingly, leveraging pre-training normalization in oracles also improves performance in test domains. This is be-

Embedding	Pre-Training BN		Task-Specific BN	
	Training domains	Test domains	Training domains	Test domains
Oracle Classifier	0.94	0.90	0.86	0.83
Oracle DIVA Features	0.91	0.87	—	—
Oracle DIVA Latents	0.88	0.85	0.79	0.77
Meta-Oracle	—	—	0.94	0.93
Classifier	0.95	0.63	0.88	0.64
DIVA Features	0.95	0.65	0.91	0.72
DIVA Latents	0.89	0.55	0.84	0.65
Meta-Trained	—	—	0.95	0.76

Table 5: *C-Omniglot, 20-ways, 1-shot.* Classification accuracy using different oracle and non-oracle embeddings with various batch normalization methods on test classes and training/test domains.

cause the statistics are now also suitable for test domains. Generally, we can observe that Oracle DIVA performs much better than non-oracle DIVA in test domains, which is another proof that DIVA is not able to generalize to unseen domains. Surprisingly, Oracle DIVA is outperformed by Oracle Classifier in both training and test domains. This suggests that forcing disentanglement, at least during pre-training, is not the correct way to generalize to unseen domains in meta-learning.

Finally, the Meta-Oracle performance provides us with an upper bound on the test domains performance we can hope to achieve. The fact that all the pre-trained oracle embeddings perform worse than Meta-Oracle in test domains suggests that focusing our research on pre-training models is not the correct way to achieve top performance. Instead, encouraging domain generalization directly when meta-learning seems to be a promising direction to explore.

5.4.3 *Leveraging Domain Information*

Oracle DIVA also provides us with a high-quality representation of the domain that is also suitable for the test domains. We thus want to find a way to leverage this domain information effectively in order to boost performance in our model. Discovering a method that successfully employs domain information would make the search for a suitable domain embedding likely worth the effort.

Initially, we consider a model where Oracle DIVA's $\mathbf{z}_d$ and $\mathbf{z}_y$ latents are concatenated and considered as LEO's embedding. We hope that LEO can learn to exploit the domain information of the task to better guide the adaptation process. Interestingly, we find that this causes LEO to tremendously overfit on training domains, causing a huge drop in test domains performance. This suggests that domain information is something that we want to avoid at embedding level. Therefore, we leverage *Feature-wise Linear Modulation* (FiLM) [113]

between the layers of the embedder to filter domain information from the embedding via affine transformations. The parameters of the transformations are dynamically computed by meta-learned NNs that receive as input a domain representation of the task provided by Oracle DIVA's $\mathbf{z}_d$. Unfortunately, we find that even in this case performance in test domains does not improve, suggesting that domain information is not something that we want to capture anywhere in our model.

6

META-TRAINING THE EMBEDDING

Our experiments in Chapter 5 have suggested that pre-training hurts the performance of our model. We thus focus on meta-learning the embedding from scratch while leveraging techniques that address the CDFSL setting defined in Section 5.1. We try many different approaches, such as pursuing disentanglement and interpretability of the embedding during meta-training, enforcing domain agnosticism of the embedding, and refining the quality of batch normalization statistics in the embedder.

6.1 META-LEARNING DOMAIN INFORMATION

Succesfully leveraging domain information from pre-training has proven to be particularly difficult in our CDFSL problem. However, since pre-training was shown to hurt performance, one could argue that meta-learning how to capture domain information may instead bring significant improvement. Therefore, to encourage disentanglement and interpretability during meta-learning, we come up with the concept of domain-discrimination task. We then present Disentangling Meta-Encoder, an original architecture that meta-learns a disentangled and interpretable latent representation of the task.

6.1.1 *Domain Discrimination Tasks*

Until now, we have only considered tasks where images from the same domains are to be correctly discriminated based on their class. We refer to a task of this type as a *class discrimination task*. We now propose the dual version of this type of task, the *domain discrimination task*.

In domain discrimination tasks, images from the same class are to be correctly discriminated based on their domain. The structure of an N-way, K-shot domain discrimination task $\mathcal{T}$ remains unchanged with respect to the class discrimination case:

$$\mathcal{D}^s = \{\langle \mathbf{x}^s_{n,k}, n\rangle | 1 \leqslant n \leqslant N, 1 \leqslant k \leqslant K\}; \tag{81a}$$

$$\mathcal{D}^q = \{\langle \mathbf{x}^q_{n,k'}, n\rangle | 1 \leqslant n \leqslant N, 1 \leqslant k' \leqslant k'\}. \tag{81b}$$

Like in the case of class discrimination, the distribution over domain discrimination tasks can also be expressed as a Bayesian network. In particular, N domain latent variables $\mathbf{z}^d_n$ are sampled from a distribution over the domain latent variables p^d:

$$\mathbf{z}^d_n \sim p^d(\mathbf{z}^d), \quad 1 \leqslant n \leqslant N. \tag{82}$$

At the same time, a single class latent variable $\mathbf{z}^c$ is sampled from a distribution over the class latent variables p^c:

$$\mathbf{z}^c \sim p^c(\mathbf{z}^c). \tag{83}$$

Each image in the task is thus sampled from a distribution over the images p^x that is conditioned on the values of the domain latent variables and class latent variable:

$$\mathbf{x}^s_{n,k} \sim p^x(\mathbf{x}_n|\mathbf{z}^c_n, \mathbf{z}^d) \quad 1 \leqslant n \leqslant N, 1 \leqslant k \leqslant K; \tag{84a}$$

$$\mathbf{x}^q_{n,k'} \sim p^x(\mathbf{x}_n|\mathbf{z}^c_n, \mathbf{z}^d) \quad 1 \leqslant n \leqslant N, 1 \leqslant k' \leqslant k'. \tag{84b}$$

With the concept of domain discrimination tasks, we aim to learn domain information in a FSL setting, which could provide more reusable representations.

6.1.2 *Disentangling Meta-Encoder*

We propose a new model, *Disentangling Meta-Encoder* (DME), to leverage both class and domain discrimination tasks. DME is based on an encoder $g_{\phi_{dme}}$ encoding the embedding of each support set image $\bar{\mathbf{x}}^s_{n,k}$ into class and domain latent codes $\mathbf{z}^c_{n,k}$ and $\mathbf{z}^d_{n,k}$. The behavior of the model afterward depends on the type of discrimination task considered. The obtained codes are averaged grouping by label or across all the images in the tasks depending on the type of the task.

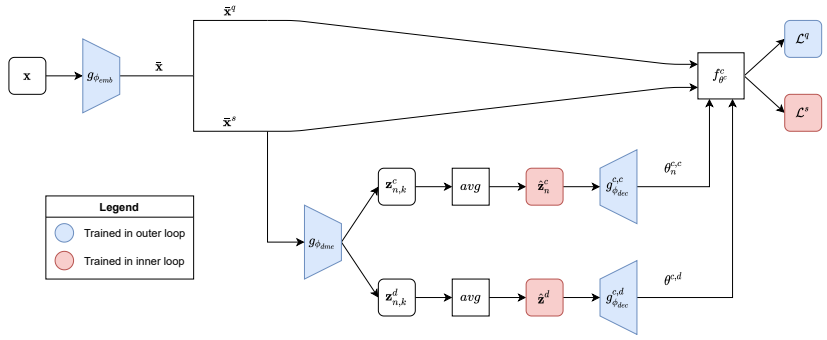


Figure 21: Disentangling Meta-Encoder pipeline for class-discrimination tasks.

For instance, in class discrimination tasks, we have:

$$\hat{\mathbf{z}}^c_n = \frac{1}{K}\sum_{k=1}^{K} \mathbf{z}^c_{n,k}, \quad \hat{\mathbf{z}}^d = \frac{1}{NK}\sum_{n=1}^{N}\sum_{k=1}^{K} \mathbf{z}^d_{n,k}. \tag{85}$$

The obtained codes are then decoded individually to obtain various different subset of parameters for the discriminator of the task. In particular, we leverage the class discrimination decoders $g^{c,c}_{\phi_{dec}}$ and $g^{c,d}_{\phi_{dec}}$ to obtain the set of class-specific parameters $\{\theta^{c,c}_n\}_{n=1}^N$ and the domain parameter $\theta^{c,d}$ respectively:

$$\theta^{c,c}_n = g^{c,c}_{\phi_{dec}}(\hat{\mathbf{z}}^c_n), \quad 1 \leqslant n \leqslant N; \tag{86a}$$

$$\theta^{c,d} = g^{c,d}_{\phi_{dec}}(\hat{\mathbf{z}}^d). \tag{86b}$$

In practice, we consider a class discriminator $f^c_{\theta^c}$ where we use the class-specific parameters $\theta^{c,c}_n$ as the n-th column for the final fully connected layer and $\theta^{c,d}$ as parameters for a feature extraction with the intent of filtering domain information.

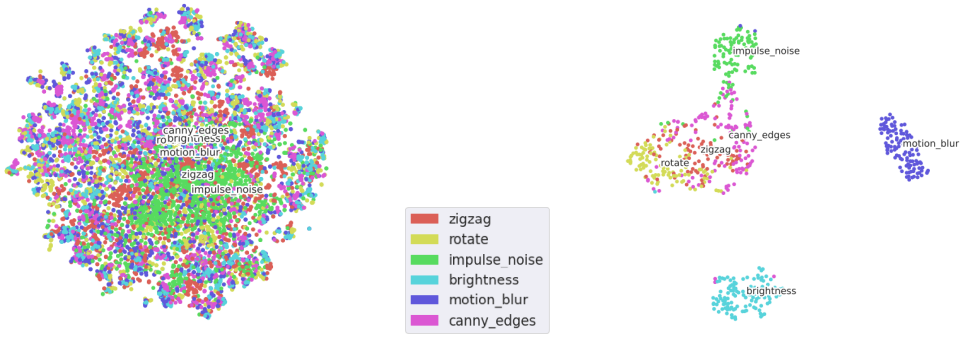
In the case of domain discrimination tasks, dual reasoning is applied by switching the role of class and domain. Indeed, here we consider two domain

discrimination decoders $g^{d,c}_{\phi_{dec}}$ and $g^{d,d}_{\phi_{dec}}$ to generate the parameters of a domain discriminator $f^{d}_{\theta^{d}}$.

The codes are also adapted via gradient descent during adaptation to optimize the performance on the support set. The parameters of the encoder $g_{\phi_{dme}}$ and of the four decoders $g^{c,c}_{\phi_{dec}}$, $g^{c,d}_{\phi_{dec}}$, $g^{d,c}_{\phi_{dec}}$, and $g^{d,d}_{\phi_{dec}}$ are meta-learned in the outer loop.

6.1.3 *Experiments and Results*

We meta-train DME providing a balanced number of class and domain discrimination tasks for each batch. The loss we consider is a weighted sum of the average loss obtained in the two types of tasks. Furthermore, we consider two types of feature extraction layers of the discriminators, namely attention and fully connected layer, thus obtaining *DME Attention* and *DME FC* respectively. A t-SNE analysis of domain information in the codes is shown in Fig. 22, while the performances obtained are available in Table 6.



(a) Class code. (b) Domain code.

Figure 22: *C-Omniglot, 20-way, 1-shot.* t-SNE plots of post-adaptation class and domain codes in DME with domain labels. Images are from test classes and test domains. Centroids are identified in the plot with the name of the label.

Model	Training Domains	Test Domains
Baseline LEO	0.95	0.76
DME Attention	0.92	0.71
DME FC	0.94	0.74

Table 6: *C-Omniglot, 20-ways, 1-shot.* Classification accuracy of Baseline LEO and DME variations on test classes and training/test domains.

We observe that DME does not provide any performance advantage over Baseline LEO in class-discrimination tasks. A possible reason for this is that the model is not able to fully filter domain information from the class code, as otherwise the centroids computed for each domain in the t-SNE plot would be much closer to each other. Nonetheless, another likely explanation is that domain information is not useful at all in our CDFSL classification problem, which is something that we already observed in Section 5.4. This would not

be entirely surprising, since the domain is modeled as a perturbation variable and shares no mutual information with the class of the image in our setting. Consequently, in our future works, we decide not to pursue a high-quality domain representation of the task and instead rely on other promising directions, namely domain agnosticism and reliable batch normalization statistics.

6.2 PURSUING DOMAIN AGNOSTICISM

Our oracle analysis in Section 5.4 and experiments in Section 6.1 have suggested that the presence of domain information anywhere in our model is something that we want to avoid. Furthermore, performances obtained when reloading a pre-trained embedding in our model have been disappointing when compared to meta-learning the embedder from scratch. Consequently, we focus our attention on ways to meta-learn an embedding that is *domain agnostic*, i. e., that is invariant to domain shifts in the image. We attempt to achieve domain agnosticism by leveraging gradient reversal.

6.2.1 *Gradient Reversal*

Gradient Reversal is a technique originally proposed by Ganin and Lempitsky [43] for performing unsupervised domain adaptation from a single training to a single test domain. The approach leverages a novel gradient reversal layer that multiplies the gradient by a certain negative constant during backpropagation. An auxiliary domain discriminator is thus connected via a gradient reversal layer to the embedding of a standard deep NN. The domain discriminator is then trained jointly with the original model to discriminate the domain label of a datapoint based on the corresponding embedding. In this way, while the parameters of the domain discriminator learn to discriminate the domain label, at the same time the parameters of the embedder learn to produce an embedding that fools the domain discriminator, resembling the behavior of adversarial training.

Empirical evidence shows that gradient reversal encourages the embedding to be domain agnostic, which is exactly what we are looking for. Indeed, the domain alignment obtained also results in a substantial improvement in standard classification performance in the target domain.

6.2.2 *Meta Gradient Reversal*

Implementing gradient reversal in our LEO model is not trivial, as our setting vastly differs from the one tackled by Ganin and Lempitsky [43]. Indeed, we are interested in solving a FSL problem, which presents many differences compared to standard machine learning tasks. We are also dealing with multiple training and test domains, which makes alignment more difficult. Furthermore, in our case, we do not have access to a large collection of unlabeled images from the target domains, which makes domain adaptation much more challenging, since we can at most rely on the few data available in the support set of the task. Moreover, we have to decide where to position the auxiliary domain discriminator within our model.

In principle, we want to encourage domain agnosticism in an area of our LEO model that is otherwise particularly abundant with domain information. To discover possible intervention zones, we first meta-train Baseline LEO along with a set of auxiliary domain discriminators that learn to discriminate the domain of a task based on quantities in the model such as the output of the embedder and post-adaptation value of the latent parameter code. We only update the discriminators during the meta-learning outer loop, and we do not backpropagate the gradient to the parameters of our original model as we do not want to explicitly encourage extraction of domain information from the images. Surprisingly, we find that the latent parameter code autonomously achieves domain agnosticism, as the accuracy we obtain when discriminating domain from its value is rather low. In contrast, the auxiliary discriminator in charge of discriminating the domain from the embedding manages to achieve accuracy very near to the perfect guess, which is a clear sign that domain information is present in the embedding.

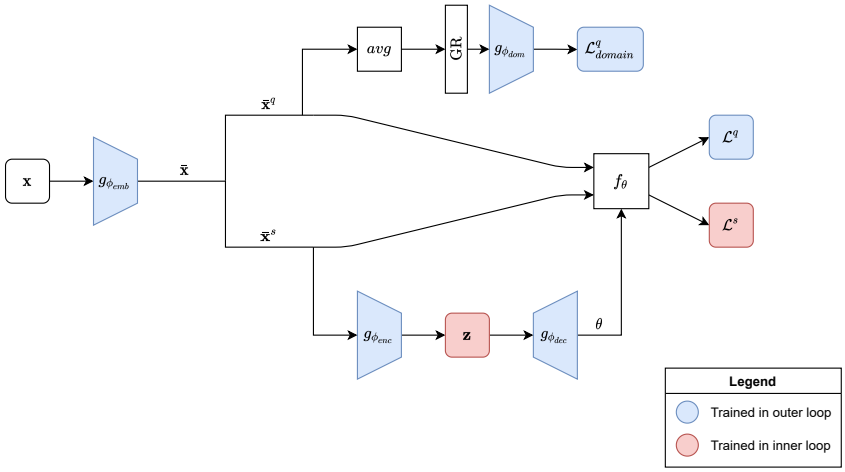


Figure 23: Gradient Reversal LEO architecture.

After identifying our intervention zone, we proceed to leverage a gradient-reversed domain discriminator $g_{\phi_{dom}}$ to encourage domain agnosticism in LEO's embedding, obtaining *GR LEO*. In particular, during meta-learning, we consider the average of the embedding across all query images

$$\bar{\mathbf{x}}^q_{avg} = \frac{1}{NK'} \sum_{n=1}^{N} \sum_{k'=1}^{K'} \bar{\mathbf{x}}^q_{nk'} \tag{87}$$

to discriminate the domain of the task. We leverage orthonormal regularization for the weights of the discriminators since otherwise the gradients obtained when introducing gradient reversal are too noisy. When updating the parameters in the outer loop, we reverse the gradient for the parameter in the embedder to encourage our embedding to be domain agnostic. Introducing gradient reversal to our model reduces the performance obtained by the domain discriminator, which is a possible sign of domain agnosticism. Analyzing the embedding via t-SNE visualizations also seems to confirm this hypothesis.

Unfortunately, when comparing the performance of GR LEO with respect to Baseline LEO in Table 7, we observe a marginal drop in overall performance in both training and test domains. Though our approach does not guarantee

Model	Training Domains	Test Domains
Baseline LEO	0.95	0.76
GR LEO	0.94	0.75

Table 7: *C-Omniglot, 20-ways, 1-shot.* Classification accuracy of Baseline LEO and GR LEO on test classes and training/test domains.

the ability to generalize to unseen domains in principle, it is surprising that even training domains performance is negatively affected. By explicitly forcing our model to discard domain information during meta-training, we would expect LEO to be better at extracting class information and consequently achieve higher performance when confronted with training domain tasks. A possible explanation for this could reside in the objective implicitly defined by reversing the domain discrimination gradient, which is essentially maximization of the cross-entropy of the discriminator. Domain agnosticism always leads to random guess, which is not the optimal solution for the maximization objective when considering the possibility of guesses being worse than random. Unfortunately, fooling the discriminator into performing worse than random requires some type of domain information. Therefore, it could be the case that our embedder learns to worsen the performance of the discriminator not by providing a domain agnostic embedding, but by introducing fictitious domain information in the embedding to deceive the discriminator. This is something that we would want to avoid since it does not bring any advantage for classification. Nonetheless, as we do not observe worse-than-random performance in the domain discriminator, we cannot verify whether this is effectively the case. Possible future lines of work in this direction could instead consider adversarial training and entropy maximization of the discriminator guess, which should help in preventing this pathological case.

6.2.3 *Discriminator Prediction Alignment*

We further experiment on Gradient Reversal LEO by considering an alignment of the embedding at adaptation time when dealing with out-of-distribution domains. The approach is based on the observation that tasks from out-of-distribution domains tend to exhibit a slight difference in the statistics of the prediction made by the domain discriminator when compared to tasks from training domains. We thus act on this discrepancy by implementing an embedder adaptation phase that precedes the original LEO inner loop. The goal of the adaptation is to fine-tune the last layers of the embedder to obtain a domain discrimination prediction on the embedding that is similar to the average prediction observed during meta-training. We manage to do so by minimizing the mean square error via gradient descent to match the average mean square error observed in the case of training domains. Unfortunately, even when aligning the embedding via this method, the performance in our model seems to be unaffected.

6.2.4 *Other Experiments*

We leverage gradient reversal in several ways to understand whether the technique can be useful in our setting. In particular, we also attempt to use gradient reversal by defining a domain-match pair discrimination problem, where a discriminator is asked to recognize whether two tasks $\mathcal{T}_1$, $\mathcal{T}_2$ share the same domain based on the value of the pair $\langle \bar{\mathbf{x}}^q_{avg,1}, \bar{\mathbf{x}}^q_{avg,2} \rangle$. A domain agnostic embedding would lead to optimal performance for the problem that is equal to random guess. Unfortunately, we do not manage to successfully train a discriminator that achieves performance better than random guess in this case, making the application of gradient reversal futile.

We also attempt to recreate a setting that is similar to the one found in the original gradient reversal approach by introducing pseudo-unseen domains, i. e., training domains that are not encountered when performing classification but are instead used to align the distribution of the embedding across the domains. In this case, we find that most pseudo-unseen domains benefit from the alignment compared to the case in which alignment does not occur, though the same is not true for some domains whose performance is instead negatively affected by the alignment. Nonetheless, the performance in test domains generally worsens.

6.3 REFINING BATCH NORMALIZATION

Previous experiments (Section 5.3) have suggested that the quality of batch normalization statistics significantly impacts the final performance of our model. However, since non-transductive meta-learning presumes no access to query set data during adaptation, the statistics we can obtain from the small support set alone are often quite unreliable and noisy. An interesting research direction thus becomes to refine the reliability of batch normalization statistics.

6.3.1 *Weighted Batch Normalization*

Domain adaptation layers are motivated by the observation that activations within a NN follow domain-dependent distributions. Domain shift is thus addressed by normalizing activations in a domain-dependent way, usually by matching the first and second-order moments to those of the standard normal distribution. The main idea can be easily extended to multiple source domains, as long as the domain of each source sample is known. However, in some cases domain of most or even all the samples in the dataset is unknown.

Mancini et al. [92] address the problem by introducing *Weighted Batch Normalization* (WBN). They define the global input distribution $q_{d,x}$ as a mixture of k Gaussians, one for each domain d. The statistics μ_d, σ^2_d can then estimated for each domain d from a batch of samples $\mathcal{B} = \{\mathbf{x}_i\}_{i=1}^b$ as

$$\mu_d = \sum_{i=1}^{b} \alpha_{i,d} x_i, \quad \sigma^2_d = \sum_{i=1}^{b} \alpha_{i,d} (x_i - \mu_d)^2, \tag{88}$$

where

$$\alpha_{i,d} = \frac{q_{d|x}(d|\mathbf{x}_i)}{\sum_{i=1}^{b} q_{d|x}(d|\mathbf{x}_i)}. \tag{89}$$

The value of $q_{d|x}(d|\mathbf{x}_i)$ is known for all samples $\mathbf{x}_i$ for which domain information is available and is otherwise inferred from data using a domain prediction network. These probabilities become an additional input for the alignment layer, denoted as $w_{i,d}$ for the predicted probability of $\mathbf{x}_i$ belonging to d. By substituting $w_{i,d}$ for $q_{d|x}(d|\mathbf{x}_i)$ they can finally obtain the empirical estimates for the mixture parameters $\hat{\mu}_d$, $\hat{\sigma}_d^2$ that are then used to normalize the layer's inputs according to the formula:

$$\text{WBN}(\mathbf{x}_i, w_i; \hat{\mu}, \hat{\sigma}) = \sum_{d \in \mathcal{D}} w_{i,d} \frac{\mathbf{x}_i - \hat{\mu}_d}{\sqrt{\hat{\sigma}_d^2 + \epsilon}}. \tag{90}$$

Interestingly, the weights provided by the domain prediction network can be leveraged when dealing with unseen domains to find a suitable interpolation of the various domain-specific normalizations that employ running statistics collected at training time. This potentially provides normalization statistics that are reliable and can generalize to unseen domains, which is welcome in our model.

6.3.2 *Meta WBN*

Generalizing the idea behind WBN to our setting is not straightforward. Since the images within a task are all from the same domain, we opt for normalization at the meta-batch level, thus sharing information on image statistics across tasks during meta-training. Considering the problem of normalizing tasks across the meta-batch would raise the question of how to generalize the concept of standard batch normalization to meta-batches, where tasks are datapoints. Therefore, we consider instead the simpler problem of normalizing images within the meta-batch independently from task membership, to reposition ourselves in a familiar, supervised-learning case and leverage the theory behind standard WBN. Nonetheless, we still perform some small changes in the algorithm to adapt to our case. In particular, since we know that a task contains images from a single domain, we discriminate the domain of a task $\mathcal{T}_i$ to obtain domain membership values $q_{d|x}(d|\mathcal{T}_i)$ that are valid for all images $\mathbf{x}_{ij} \in \mathcal{T}_i$.

The domain discrimination of the task is performed via a stand-alone deep CNN $g_{\phi_{dom}}$ that takes as input the images in the support set of the task to return the array of domain weights for the task w. Analogously to standard WBN, we can leverage the predictions of the domain discriminator to interpolate various domain-specific normalizations when dealing with tasks from unseen domains.

We thus test our implementation of Meta WBN by substituting the batch normalization layers in Baseline LEO's embedder with WBN layers, obtaining *WBN LEO*. For completeness, we also consider WBN a convex combination of task-specific normalization and WBN, with a meta-learned interpolation parameter, which we refer to as *TWBN LEO*. Unfortunately, our implementation

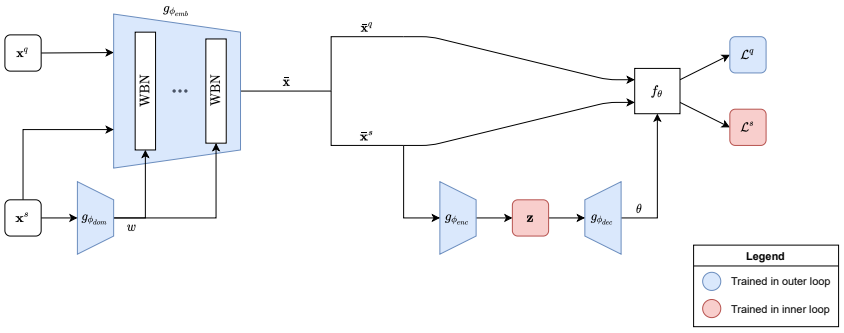


Figure 24: Weighted Batch Normalization LEO architecture.

of WBN is very demanding on the memory resources, which forces us to further reduce the batch size in our experiments from 8 to 4. The results are shown in Table 8.

Model	Training Domains	Test Domains
Baseline LEO	0.95	0.76
WBN LEO	0.94	0.62
TWBN LEO	0.94	0.74

Table 8: *C-Omniglot, 20-ways, 1-shot.* Classification accuracy of Baseline LEO, WBN LEO, and TWBN LEO on test classes and training/test domains.

When leveraging WBN LEO we find that while training domains performance is barely affected, test domains performance experiences a sharp drop. There are multiple possible reasons for this occurrence, such as poor quality of the interpolation provided by the domain discriminator or even the non-existence of a suitable interpolation. Nonetheless, since the entropy displayed by the domain discriminator is extremely low, we tend to favor the first explanation. An interesting future line of work could therefore involve coming up with domain discriminators that provide high-quality interpolation of training domains when presented with images from unseen domains. Furthermore, since WBN may benefit from larger batch size, an optimized version of the algorithm would be needed to fully evaluate the approach.

Leveraging TWBN LEO also seems to degrade test domains performance, though much less severely. Interestingly, the parameters defining the convex interpolation between the two BN techniques tend to favor weighted BN in the low-level layers and task-specific BN in the high-level layers, suggesting that domain information is especially present in low-level features. Indeed, this is in line with the observation that features tend to become increasingly more domain-invariant when going deeper into the network [5].

7 CONCLUSION

In this thesis, we tackled the challenging problem of CDFSL classification, where models are challenged to learn classification tasks while dealing with both data scarcity and domain shift. The problem is often ignored in the literature, with many recent works in Meta-Learning only considering simple benchmarks featuring samples from a single domain. Despite the latest efforts and achievements in the field, the literature still does not offer a method that can generalize its knowledge to tasks from unseen, out-of-distribution data domains. Finding a way to generalize meta-knowledge to new domains would enable deep-learning solutions to tackle interesting and important problems when the amount of available training data is low.

7.1 ORIGINAL CONTRIBUTION

To effectively address the CDFSL problem, we first proposed Corrupted-Omniglot, a novel CDFSL benchmark for classification. The dataset is obtained by augmenting the images found in Omniglot [76] with the image corruptions provided by Mu and Gilmer [99]. We thus built upon LEO [124] to address the presence of multiple domains in our dataset.

Initially, we leveraged a disentangled and interpretable embedding provided by a pre-trained DIVA [61]. We observed that, despite being the best among the other pre-trained embeddings considered, DIVA's embedding is outperformed by a simple meta-trained embedding in test domains. Moreover, in this phase, we also found that poor quality of batch normalization statistics and large amounts of pre-training significantly worsens the model's performance.

We then employed oracle models, i. e., models trained on test domains, to understand how and whether we should focus on learning a domain representation. Interestingly, we learned that including domain information from pre-training causes performance to drop in test domains. Furthermore, we observed that, in terms of embedding performance, Oracle DIVA is surprisingly outclassed by a simple Oracle Classifier, casting doubts on whether disentanglement and interpretability of the embedding are beneficial.

We thus shifted our focus to meta-learning the embedding from scratch. Initially, we tried to encourage disentanglement and interpretability during meta-learning by proposing Disentangling Meta Encoder, a novel architecture that meta-learns class and domain representations of the task. We found that even in this case the obtained domain representation is not helpful to boost performance in both training and test domains.

We then developed other promising ideas, namely meta-learning a domain agnostic embedding via gradient reversal [43] and obtaining high-quality, reusable normalization statistics by rethinking weighted batch normalization [92] in the meta-learning setting. Unfortunately, both approaches failed to boost test domains performance.

7.2 FUTURE WORK

Because of our findings, we believe that future works in CDFSL should not consider a pre-training phase and instead should focus on meta-learning from scratch. Furthermore, we believe that, despite the initial failures encountered with our experiments, a domain agnostic embedding and reliable batch normalization statistics are two promising research directions that are not yet sufficiently explored in CDFSL.

Moreover, we think that a promising line of work is to focus on the role of adaptation. Compared to the static nature of the embedding, adaptation can correct the model based on data from the task at hand, which leads to a reasonable assumption of improvement in unseen domains. To this end, warped gradients [39] may help in achieving the desired effect. Rethinking domain adaptation techniques available in the literature [86, 148] for the adaptation phase of meta-learning may also be effective in generalizing to unseen domains.

BIBLIOGRAPHY

[1] Adewole S Adamson and Avery Smith. "Machine learning and health care disparities in dermatology." In: *JAMA dermatology* 154.11 (2018), pp. 1247–1248.

[2] Thomas Adler, Johannes Brandstetter, Michael Widrich, Andreas Mayr, David Kreil, Michael Kopp, Günter Klambauer, and Sepp Hochreiter. "Cross-Domain Few-Shot Learning by Representation Fusion." In: *arXiv preprint arXiv:2010.06498* (2020).

[3] Arvind Agarwal, Samuel Gerber, and Hal Daume. "Learning multiple tasks using manifold regularization." In: *Advances in neural information processing systems*. 2010, pp. 46–54.

[4] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. "Continuous adaptation via meta-learning in nonstationary and competitive environments." In: *arXiv preprint arXiv:1710.03641* (2017).

[5] Rahaf Aljundi and Tinne Tuytelaars. "Lightweight unsupervised domain adaptation by convolutional filter reconstruction." In: *European Conference on Computer Vision*. Springer. 2016, pp. 508–515.

[6] Antreas Antoniou, Harrison Edwards, and Amos Storkey. "How to train your maml." In: *arXiv preprint arXiv:1810.09502* (2018).

[7] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. "Convex multi-task feature learning." In: *Machine learning* 73.3 (2008), pp. 243–272.

[8] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. "Mine: mutual information neural estimation." In: *arXiv preprint arXiv:1801.04062* (2018).

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 35.8 (2013), pp. 1798–1828.

[10] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).

[11] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[12] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[13] Avrim L. Blum and Ronald L. Rivest. "Training a 3-node Neural Network is NP-complete." In: *Neural Networks* 5.1 (1992), pp. 117–127.

[14] Léon Bottou. "Online learning and stochastic approximations." In: *Online learning in Neural Networks* 17.9 (1998), p. 142.

[15] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[16] John Bronskill, Jonathan Gordon, James Requeima, Sebastian Nowozin, and Richard Turner. "Tasknorm: Rethinking batch normalization for meta-learning." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1153–1164.

[17] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. "Albumentations: Fast and Flexible Image Augmentations." In: *Information* 11.2 (2020). ISSN: 2078-2489. DOI: 10.3390/info11020125. URL: https://www.mdpi.com/2078-2489/11/2/125.

[18] Rich Caruana. "Multitask learning." In: *Machine learning* 28.1 (1997), pp. 41–75.

[19] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. "A closer look at few-shot classification." In: *arXiv preprint arXiv:1904.04232* (2019).

[20] Zhiyuan Chen and Bing Liu. "Lifelong machine learning." In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 12.3 (2018), pp. 1–207.

[21] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. "The loss surfaces of multilayer networks." In: *Artificial intelligence and statistics*. 2015, pp. 192–204.

[22] Gabriela Csurka. *Domain adaptation in computer vision applications*. Springer, 2017.

[23] George Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[24] Bin Dai and David Wipf. "Diagnosing and enhancing VAE models." In: *arXiv preprint arXiv:1903.05789* (2019).

[25] Hal Daumé III. "Bayesian multitask learning with latent hierarchies." In: *arXiv preprint arXiv:0907.0783* (2009).

[26] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014, pp. 2933–2941.

[27] Olivier Delalleau and Yoshua Bengio. "Shallow vs. deep sum-product networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2011, pp. 666–674.

[28] Tristan Deleu, Tobias Würfl, Mandana Samiei, Joseph Paul Cohen, and Yoshua Bengio. "Torchmeta: A meta-learning library for pytorch." In: *arXiv preprint arXiv:1909.06576* (2019).

[29] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and Koray Kavukcuoglu. "Natural Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015, pp. 2071–2079.

[30] Pedro Domingos. "The role of Occam's razor in knowledge discovery." In: *Data mining and knowledge discovery* 3.4 (1999), pp. 409–425.

[31] Alexey Dosovitskiy and Thomas Brox. "Inverting visual representations with convolutional networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4829–4837.

[32] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of Machine Learning Research* 12.7 (2011).

[33] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning." In: *ArXiv e-prints* (2016). eprint: 1603.07285.

[34] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Visualizing Higher-Layer Features of a Deep Network." In: *Workshop on Learning Feature Hierarchies at International Conference on Machine Learning (ICML)* (June 2009).

[35] Theodoros Evgeniou and Massimiliano Pontil. "Regularized multi–task learning." In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 109–117.

[36] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1126–1135.

[37] Chelsea Finn and Sergey Levine. "Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm." In: *arXiv preprint arXiv:1710.11622* (2017).

[38] Sebastian Flennerhag, Andrei A Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. "Meta-learning with warped gradient descent." In: *arXiv preprint arXiv:1909.00025* (2019).

[39] Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. "Meta-Learning with Warped Gradient Descent." In: *International Conference on Learning Representations (ICLR)*. 2020. URL: https://openreview.net/forum?id=rkeiQlBFPB.

[40] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. "Forward and reverse gradient-based hyperparameter optimization." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1165–1173.

[41] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. "Bilevel programming for hyperparameter optimization and meta-learning." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1568–1577.

[42] Kunihiko Fukushima. "Neocognitron: A self-organizing Neural Network model for a mechanism of Pattern Recognition unaffected by shift in position." In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[43] Yaroslav Ganin and Victor Lempitsky. "Unsupervised domain adaptation by backpropagation." In: *International conference on machine learning*. PMLR. 2015, pp. 1180–1189.

[44] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier Neural Networks." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.

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

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

[47] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. "Generalized inner loop meta-learning." In: *arXiv preprint arXiv:1910.01727* (2019).

[48] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[49] Yunhui Guo, Noel C Codella, Leonid Karlinsky, James V Codella, John R Smith, Kate Saenko, Tajana Rosing, and Rogerio Feris. "A broader study of cross-domain few-shot learning." In: *European Conference on Computer Vision*. Springer. 2020, pp. 124–141.

[50] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. "Meta-reinforcement learning of structured exploration strategies." In: *arXiv preprint arXiv:1802.07245* (2018).

[51] Barbara Hammer and Thomas Villmann. "Mathematical Aspects of Neural Networks." In: *ESANN*. 2003, pp. 59–72.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: (2015), pp. 1026–1034.

[53] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural Networks for machine learning lecture 6a overview of mini-batch gradient descent." In: *Cited on* 14.8 (2012).

[54] Sepp Hochreiter and Jürgen Schmidhuber. "Flat minima." In: *Neural Computation* 9.1 (1997), pp. 1–42.

[55] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[56] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks." In: *Neural Networks* 4.2 (1991), pp. 251–257.

[57] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5 (1989), pp. 359–366.

[58] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. "Meta-learning in Neural Networks: A survey." In: *arXiv preprint arXiv:2004.05439* (2020).

[59] Kyle Hsu, Sergey Levine, and Chelsea Finn. "Unsupervised learning via meta-learning." In: *arXiv preprint arXiv:1810.02334* (2018).

[60] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[61] Maximilian Ilse, Jakub M Tomczak, Christos Louizos, and Max Welling. "Diva: Domain invariant variational autoencoders." In: *Medical Imaging with Deep Learning*. PMLR. 2020, pp. 322–348.

[62] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2015.

[63] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. "What is the best multi-stage architecture for object recognition?" In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE. 2009, pp. 2146–2153.

[64] J Stephen Judd. *Neural Network design and the complexity of learning*. MIT Press, 1990.

[65] Alexander B. Jung et al. *imgaug*. https://github.com/aleju/imgaug. Online; accessed 06-Apr-2021. 2020.

[66] Zhuoliang Kang, Kristen Grauman, and Fei Sha. "Learning with whom to share in multi-task feature learning." In: *ICML*. 2011.

[67] Diederik P. Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *International Conference on Learning Representations (ICLR)* (2015).

[68] Diederik P Kingma, Danilo J Rezende, Shakir Mohamed, and Max Welling. "Semi-supervised learning with deep generative models." In: *arXiv preprint arXiv:1406.5298* (2014).

[69] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (2013).

[70] Diederik P Kingma and Max Welling. "An introduction to variational autoencoders." In: *arXiv preprint arXiv:1906.02691* (2019).

[71] Newton M Kinyanjui, Timothy Odonga, Celia Cintas, Noel CF Codella, Rameswar Panda, Prasanna Sattigeri, and Kush R Varshney. "Estimating skin tone and effects on classification performance in dermatology datasets." In: *arXiv preprint arXiv:1910.13268* (2019).

[72] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[73] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10 and CIFAR-100 datasets*. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2021-04-06.

[74] Abhishek Kumar and Hal Daume III. "Learning task grouping and overlap in multi-task learning." In: *arXiv preprint arXiv:1206.6417* (2012).

[75] Namyeong Kwon, Hwidong Na, Gabriel Huang, and Simon Lacoste-Julien. "Repurposing Pretrained Models for Robust Out-of-domain Few-Shot Learning." In: *arXiv preprint arXiv:2103.09027* (2021).

[76] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "The Omniglot challenge: a 3-year progress report." In: *Current Opinion in Behavioral Sciences* 29 (2019), pp. 97–104.

[77] Alessandro Lazaric and Mohammad Ghavamzadeh. "Bayesian multi-task reinforcement learning." In: *ICML-27th International Conference on Machine Learning*. Omnipress. 2010, pp. 599–606.

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[79] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop." In: *Neural Networks: Tricks of the trade*. Ed. by G. Orr and Muller K. Springer, 1998, pp. 9–48.

[80] Yann LeCun. "Generalization and network design strategies." In: *Connectionism in perspective* 19 (1989), pp. 143–155.

[81] Aoxue Li, Tiange Luo, Tao Xiang, Weiran Huang, and Liwei Wang. "Few-shot learning with global class representations." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9715–9724.

[82] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. "Deeper, broader and artier domain generalization." In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5542–5550.

[83] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. "Visualizing the loss landscape of neural nets." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018, pp. 6389–6399.

[84] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. "Meta-sgd: Learning to learn quickly for few-shot learning." In: *arXiv preprint arXiv: 1707.09835* (2017).

[85] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. "Challenging common assumptions in the unsupervised learning of disentangled representations." In: *international conference on machine learning*. PMLR. 2019, pp. 4114–4124.

[86] Mohammad Reza Loghmani, Luca Robbiano, Mirco Planamente, Kiru Park, Barbara Caputo, and Markus Vincze. "Unsupervised Domain Adaptation through Inter-modal Rotation for RGB-D Object Recognition." In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6631–6638.

[87] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440.

[88] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. "The variational fair autoencoder." In: *arXiv preprint arXiv:1511.00830* (2015).

[89] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[90] David JC MacKay. *Information theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[91] Aravindh Mahendran and Andrea Vedaldi. "Understanding deep image representations by inverting them." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 5188–5196.

[92] Massimiliano Mancini, Lorenzo Porzi, Samuel Rota Bulo, Barbara Caputo, and Elisa Ricci. "Boosting domain adaptation by discovering latent domains." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3771–3780.

[93] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Guided meta-policy search." In: *arXiv preprint arXiv:1904.00956* (2019).

[94] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. "Meta-learning update rules for unsupervised representation learning." In: *arXiv preprint arXiv:1804.00222* (2018).

[95] Tom Mitchell. *Machine Learning*. New York, USA: McGraw-Hill Higher Education, 1997.

[96] Guido F. Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. "On the Number of Linear Regions of Deep Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014, pp. 2924–2932.

[97] Saeid Motiian, Quinn Jones, Seyed Mehdi Iranmanesh, and Gianfranco Doretto. "Few-shot adversarial domain adaptation." In: *arXiv preprint arXiv:1711.02536* (2017).

[98] Daniel Moyer, Shuyang Gao, Rob Brekelmans, Greg Ver Steeg, and Aram Galstyan. "Invariant representations without adversarial training." In: *arXiv preprint arXiv:1805.09458* (2018).

[99] Norman Mu and Justin Gilmer. "Mnist-c: A robustness benchmark for computer vision." In: *arXiv preprint arXiv:1906.02337* (2019).

[100] Tsendsuren Munkhdalai and Adam Trischler. "Metalearning with hebbian fast weights." In: *arXiv preprint arXiv:1807.05076* (2018).

[101] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[102] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[103] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning." In: *arXiv preprint arXiv:1803.11347* (2018).

[104] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted Boltzmann machines." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2010.

[105] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)." In: *Doklady an ussr*. Vol. 269. 1983, pp. 543–547.

[106] Yurii Nesterov. "A method of solving a convex programming problem with convergence rate $O(1/k^2)$." In: *Soviet Mathematics Doklady* 27.2 (1983), pp. 372–376.

[107] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer verlag, 2006.

[108] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning." In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[109] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. "Continual lifelong learning with neural networks: A review." In: *Neural Networks* 113 (2019), pp. 54–71.

[110] Alexandre Passos, Piyush Rai, Jacques Wainer, and Hal Daume III. "Flexible modeling of latent task structures in multitask learning." In: *arXiv preprint arXiv:1206.6486* (2012).

[111] Judea Pearl. "Bayesian networks." In: (2011).

[112] Shuman Peng, Weilian Song, and Martin Ester. "Combining Domain-Specific Meta-Learners in the Parameter Space for Cross-Domain Few-Shot Classification." In: *arXiv preprint arXiv:2011.00179* (2020).

[113] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. "FiLM: Visual Reasoning with a General Conditioning Layer." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2018.

[114] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods." In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[115] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. "Rapid learning or feature reuse? towards understanding the effectiveness of maml." In: *arXiv preprint arXiv:1909.09157* (2019).

[116] Piyush Rai and Hal Daumé III. "Infinite predictor subspace models for multitask learning." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 613–620.

[117] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning." In: (2016).

[118] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning." In: *International Conference on Learning Representations (ICLR)*. 2017.

[119] Mark Bishop Ring et al. "Continual learning in reinforcement environments." PhD thesis. University of Texas at Austin Austin, Texas 78712, 1994.

[120] Herbert Robbins and Sutton Monro. "A stochastic approximation method." In: *The annals of mathematical statistics* (1951), pp. 400–407.

[121] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[122] Veronica Rotemberg, Allan Halpern, Steven Dusza, and Noel CF Codella. "The role of public challenges and data sets towards algorithm development, trust, and use in clinical practice." In: *Seminars in cutaneous medicine and surgery*. Vol. 38. 1. 2019, E38–E42.

[123] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *Nature* 323.6088 (1986), pp. 533–536.

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

[125] Doyen Sahoo, Hung Le, Chenghao Liu, and Steven CH Hoi. "Meta-learning with domain adaptation for few-shot learning under domain shift." In: (2018).

[126] Tim Salimans and Durk P Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)* 29 (2016), pp. 901–909.

[127] Tom Schaul, Sixin Zhang, and Yann LeCun. "No more pesky learning rates." In: *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR. 2013, pp. 343–351.

[128] Jürgen Schmidhuber. "Deep learning in Neural Networks: An overview." In: *Neural Networks* 61 (2015), pp. 85–117.

[129] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. "Taking the human out of the loop: A review of Bayesian optimization." In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[130] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[131] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *International Conference on Learning Representations (ICLR)* (2015).

[132] Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 4077–4087.

[133] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. "Learning to compare: Relation network for few-shot learning." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1199–1208.

[134] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 28. MIT press, 1998.

[135] Richard Sutton. "Two problems with back propagation and other steepest descent learning procedures for networks." In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*. 1986, pp. 823–832.

[136] Sebastian Thrun and Lorien Pratt. "Learning to learn: Introduction and overview." In: *Learning to learn*. Springer, 1998, pp. 3–17.

[137] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. "Rethinking few-shot image classification: a good embedding is all you need?" In: *arXiv preprint arXiv:2003.11539* (2020).

[138] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. "Meta-dataset: A dataset of datasets for learning to learn from few examples." In: *arXiv preprint arXiv:1903.03096* (2019).

[139] Hung-Yu Tseng, Hsin-Ying Lee, Jia-Bin Huang, and Ming-Hsuan Yang. "Cross-domain few-shot classification via learned feature-wise transformation." In: *arXiv preprint arXiv:2001.08735* (2020).

[140] Aleš Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. "Task-specific generalization of discrete and periodic dynamic movement primitives." In: *IEEE Transactions on Robotics* 26.5 (2010), pp. 800–815.

[141] Vladimir Vapnik. "Principles of risk minimization for learning theory." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1992, pp. 831–838.

[142] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[143] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, and Daan Wierstra. "Matching Networks for One Shot Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 3630–3638.

[144] Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim. "Multimodal Model-Agnostic Meta-Learning via Task-Aware Modulation." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 1–12.

[145] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning." In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[146] Paul J. Werbos. "Beyond regression: New tools for prediction and analysis in the behavioral sciences." PhD thesis. Cambridge, MA, USA: Harvard University, 1974.

[147] Paul J. Werbos. "Applications of Advances in Nonlinear Sensitivity Analysis." In: *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*. 1981, pp. 762–770.

[148] Ruijia Xu, Guanbin Li, Jihan Yang, and Liang Lin. "Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1426–1435.

[149] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. "Hierarchically structured meta-learning." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7045–7054.

[150] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *arXiv preprint arXiv:1411.1792* (2014).

[151] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. "Understanding Neural Networks through deep visualization." In: *Deep Learning Workshop at International Conference on Machine Learning (ICML)* (2015).

[152] Kai Yu, Volker Tresp, and Anton Schwaighofer. "Learning Gaussian processes from multiple tasks." In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 1012–1019.

[153] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. "Gradient surgery for multi-task learning." In: *arXiv preprint arXiv:2001.06782* (2020).

[154] Matthew D Zeiler. "Adadelta: an adaptive learning rate method." In: *arXiv preprint arXiv:1212.5701* (2012).

[155] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2014, pp. 818–833.

[156] Yu Zhang and Qiang Yang. "A survey on multi-task learning." In: *arXiv preprint arXiv:1707.08114* (2017).

[157] An Zhao, Mingyu Ding, Zhiwu Lu, Tao Xiang, Yulei Niu, Jiechao Guan, and Ji-Rong Wen. "Domain-adaptive few-shot learning." In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 1390–1399.