DIPLOMARBEIT

One-Class Domain Adaptation via Meta-Learning

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Technische Mathematik

eingereicht von

Stephanie Holly
Matrikelnummer 11703485

ausgeführt am Institut für Analysis und Scientific Computing
der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien
in Zusammenarbeit mit Siemens Technology

Betreuung
Betreuer: Assoz. Prof. Dr. techn. Dipl.-Ing. Clemens Heitzinger

Wien, am 6. Oktober 2024

Stephanie Holly	Clemens Heitzinger

Kurzfassung

Die Integration von IoT (Internet of Things) Sensorplattformen in industrielle Anlagen eröffnet neue Möglichkeiten für den Einsatz von Modellen des maschinellen Lernens, insbesondere in der Anomalie-Klassifizierung. Der erfolgreiche Einsatz dieser Modelle in solchen Systemen ist jedoch mit erheblichen Herausforderungen verbunden. Insbesondere weichen die in kontrollierten Laborumgebungen gesammelten Trainingsdaten häufig erheblich von den Echtzeitdaten in Produktionsumgebungen ab. Zudem ist es oft nicht möglich, für jede Anomalie-Klasse in neuen Umgebungen eine ausreichende Menge an Daten zu beschaffen. Daher ist es von entscheidender Bedeutung, anpassungsfähige Modelle zu entwickeln, die sich effektiv von einer Umgebung in eine andere übertragen lassen und eine schnelle Anpassung unter Verwendung normaler Daten ermöglichen. Diese Arbeit erweitert diese Problemstellung auf allgemeine Klassifikationsaufgaben und formuliert die Fragestellung der "One-Class Domain Adaptation" (OC-DA). Nach meinem Kenntnisstand handelt es sich bei dieser Arbeit um die erste, die sich mit OC-DA befasst. Im Rahmen dieser Arbeit wird eine Task-Sampling-Strategie vorgeschlagen, die es ermöglicht, jeden zweistufigen Meta-Learning-Algorithmus an die Anforderungen von OC-DA anzupassen. Konkret wird der "Model-Agnostic Meta-Learning" (MAML) Algorithmus modifiziert und der OC-DA MAML Algorithmus entwickelt. Eine theoretische Analyse zeigt, dass OC-DA MAML Meta-Parameter findet, die es dem Modell ermöglichen, sich mit nur wenigen Trainingsdaten einer einzigen Klasse an eine neue Umgebung anzupassen. Der OC-DA MAML wird empirisch anhand des Rainbow-MNIST Meta-Learning-Benchmarks und eines realen Datensatzes aus vibrationsbasierten Sensormesswerten evaluiert. Die Ergebnisse zeigen, dass OC-DA MAML signifikant bessere Leistungen erbringt als MAML mit der herkömmlichen Task-Sampling-Strategie.

Abstract

In recent years, the integration of IoT (Internet of Things) sensor platforms into industrial plants has opened up new opportunities for applying machine learning models to various tasks, such as anomaly classification. However, the deployment of these models in such systems poses significant challenges due to distribution shifts, as the training data acquired in controlled laboratory settings may significantly differ from real-time data in production environments. Furthermore, many real-world applications cannot provide a substantial number of labeled examples for each anomalous class in every new environment. It is therefore crucial to develop adaptable machine learning models that can be effectively transferred from one environment to another, enabling rapid adaptation using normal data. I extended this problem setting to an arbitrary multi-class classification task and formulated the one-class domain adaptation (OC-DA) problem setting. To the best of my knowledge, my work is the first to address one-class adaptation across domains. I took a meta-learning approach to tackle the challenge of OC-DA, and proposed a task sampling strategy to adapt any bi-level meta-learning algorithm to OC-DA. I modified the well-established model-agnostic meta-learning (MAML) algorithm and introduced the OC-DA MAML algorithm. I provided a theoretical analysis showing that OC-DA MAML optimizes for meta-parameters that enable rapid one-class adaptation across domains. I empirically evaluated OC-DA MAML on the Rainbow-MNIST meta-learning benchmark and on a real-world dataset of vibration-based sensor readings. The results show that OC-DA MAML significantly outperforms MAML using the standard task sampling strategy.

Acknowledgement

First and foremost, I would like to express my deep gratitude to my supervisor, Prof. Clemens Heitzinger.

I also wish to acknowledge the support of the Distributed AI Systems Research Group at Siemens Technology. My sincere thanks go to all my colleagues in the research group for making this experience both enriching and memorable.

A special thanks is deserved by Jana Eder for her consistent support, insightful discussions and encouragement throughout this work.

I would like to extend my appreciation to Thomas Bierweiler and Stefan van Dosky for their efforts in acquiring the real-world dataset. Interdisciplinary collaboration with them has been essential for interpreting the results.

I am very grateful to Ahmed Frikha for the interesting and helpful discussions.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 6. Oktober 2024

Stephanie Holly

Contents

1 Introduction

In recent years, the integration of Internet of Things (IoT) sensor platforms into industrial plants has opened up new avenues for the application of machine learning models in various industrial systems [14, 9, 18, 30], offering significant opportunities for enhancing efficiency in industrial processes. However, deploying machine learning models in such systems poses significant challenges, particularly in ensuring consistent performance across diverse environments within an industrial plant. In many industrial machine learning applications, models are developed in controlled laboratory settings before being deployed into critical production environments [15]. However, real-world data is inherently complex and diverse, and thus, the training data may significantly differ from the real-time data in production. Distribution shifts – where the training distribution differs from the test distribution – pose significant challenges in the application of machine learning, as they can lead to a substantial decline in model performance [17]. Therefore, it is crucial to develop robust machine learning models that can be effectively transferred from laboratory settings to real-world deployments, as well as from one environment to another within an industrial system.

The generalization ability of machine learning models has been significantly driven by the wealth and diversity of available data. This is evident in computer vision, where datasets like ImageNet [26] have improved the performance of image classification models. Although these pre-trained models can be effectively applied to related computer vision tasks, such extensive datasets are often unavailable in many specialized domains [12]. Common approaches to directly addressing the challenge of distribution shifts are domain adaptation and domain generalization. Domain adaptation methods leverage unlabeled or sparsely labeled data from the test environment for adaptation, while domain generalization methods are specifically trained to achieve invariance across diverse environments, enabling them to generalize to the test environment without requiring further adaptation [12]. While domain adaptation and generalization methods require a sufficiently large amount of data [1, 13], real-world applications often face significant limitations in data availability. This scarcity is often due to intrinsic factors (e.g., rare medical conditions, special industrial failure types) or the time-consuming and costly nature of data acquisition processes. In contrast, meta-learning is specifically tailored to few-shot learning settings and can effectively address both domain adaptation and domain generalization tasks [12]. Meta-learning is commonly understood as 'learning to learn' and leverages prior experience across multiple tasks with limited datasets to develop models capable of rapid adaptation to new tasks [12].

However, a notable shortcoming of existing meta-learning algorithms is that the adaptation process still requires access to examples of every class, either for ranking similarity between examples [16, 32, 29] or for fine-tuning classifiers [25, 3, 24, 22]. This dependence constrains

the applicability of current meta-learning methods. In many real-world applications, the adaptation data in new domains is constrained not only by limited volume but also by the scarcity of certain classes. For illustration, machine learning models in industry are typically pre-trained in laboratory settings before being transferred to real-world deployments. While normal operational data is often readily available for model adaptation, acquiring anomalous data within large and complex industrial systems is both expensive and time-consuming [5]. This raises the question of whether and how the information present in normal data can be leveraged for adaptation to a new domain.

I extended this problem setting to an arbitrary multi-class classification task and formulated the *One-Class Domain Adaptation* (OC-DA) problem setting. To the best of my knowledge, my work is the first to address one-class adaptation across domains. The goal of OC-DA is to learn a model that is able to quickly adapt to a new domain using only a few examples of *one* class. I take a meta-learning approach to tackle the challenge of OC-DA and propose a task sampling strategy to adapt any bi-level meta-learning algorithm to the OC-DA setting. While recent meta-learning algorithms are designed to effectively address generalization across *diverse* learning tasks, i.e., learning tasks with varying label spaces [16, 32, 29, 25, 3, 24, 22], OC-DA focuses on generalization from one class to the other classes of a *single* learning task across diverse domains.

In Chapter 2, I discuss the challenges involved in transferring machine learning models to real-world deployments, with a focus on distribution shifts. Chapter 3 provides an overview of recent meta-learning approaches, covering metric-based, model-based, and optimization-based algorithms. In Chapter 4, I introduce the OC-DA problem setting and propose a task sampling strategy to adapt any bi-level meta-learning algorithm to the OC-DA setting. I focus on modifying the well-established model-agnostic meta-learning (MAML) algorithm [3] and introduce the OC-DA MAML algorithm. Furthermore, I provide a theoretical analysis showing that the OC-DA MAML algorithm explicitly optimizes for meta-parameters that enable rapid one-class adaptation across domains. In Chapter 5, I discuss the empirical experiments and results. I demonstrate the effectiveness of the OC-MAML algorithm on the Rainbow-MNIST dataset [4] and on a real-world dataset of vibration-based sensor readings recorded by centrifugal pumps within diverse environments. I show the performance gap between source and target domains, and compare the MAML algorithm to the OC-DA MAML algorithm. In Chapter 6, I summarize my results.

In summary, my contributions are:

(1) I formulate a new problem setting in the context of domain adaptation (OC-DA) motivated by real-world challenges and requirements.

(2) I show that the well-established meta-learning algorithm MAML fails in the OC-DA setting.

(3) I propose a sampling strategy to adapt any bi-level meta-learning algorithm to OC-DA. I focus on modifying the MAML algorithm and introduce the OC-DA MAML algorithm.

(4) I provide a theoretical analysis showing that the OC-DA MAML algorithm optimizes for meta-parameters that enable rapid one-class adaptation across domains.

(5) I empirically evaluate the OC-MAML algorithm on a meta-learning benchmark, the Rainbow-MNIST dataset [4], and on a dataset of sensor readings recorded by centrifugal pumps within diverse environments, demonstrating its robustness for real-world applications.

2 Distribution Shifts

A common assumption in machine learning is that data is identically and independently distributed (iid), and both training and testing examples are drawn from the same distribution [17]. However, this assumption often fails to hold in real-world applications. In real-world applications, data is inherently complex and diverse, often collected across multiple environments [1]. In machine learning, these variations are known as distribution shifts, where the training distribution differs from the test distribution [17]. In many real-world deployments, distribution shifts naturally arise in applications such as animal species categorization, tumor identification, poverty mapping [17], remote sensing [27] and industrial systems [15], caused by different cameras, hospitals, demographics, continents, countries, seasons and time periods. For instance, in biomedical applications, machine learning models are frequently trained on data from a limited number of hospitals, yet the goal is to generalize their deployment across a broader range of hospitals [17]. In remote sensing, the increasing number of satellite constellations facilitates the monitoring of global issues such as deforestation, wildfires, road flooding, and urban development [27]. However, the representations of the Earth's surface can differ significantly between continents and across seasons, posing challenges for the effective transfer of machine learning models to previously unseen geographic regions [27].

2.1 Distribution Shifts in Industrial Systems

In industrial machine learning, model robustness plays a crucial role. In many applications, models are often developed in laboratory settings before being transferred to critical production environments. However, the transfer of these models to real-world deployments poses significant challenges due to distribution shifts, as the training data acquired in controlled laboratory settings may significantly differ from real-time data in production environments. Moreover, even if the training data is initially collected from production environments, it is often outdated by the time the model is developed and ready for deployment [15]. Furthermore, an increasing number of industrial assets is equipped with IoT sensor platforms, presenting significant opportunities for applying machine learning models in industrial systems (e.g., predictive maintenance, condition monitoring, anomaly classification, etc.) [9]. Industrial plants often contain multiple industrial assets of the same type; for instance, it is common to find hundreds of identical filling pumps within a single food and beverage plant [14]. Additionally, these industry-standard assets are frequently deployed across multiple industrial plants world-wide. However, the model transfer from one industrial asset to another, even if they are of the same type and highly standardized,

is often difficult. Various real-world factors, such as slight variations in production, wear and tear, and environmental factors, can influence the sensor recordings [14, 9]. For reliable performance, it is crucial to develop robust models that can be effectively transferred from one industrial asset to another across diverse environments.

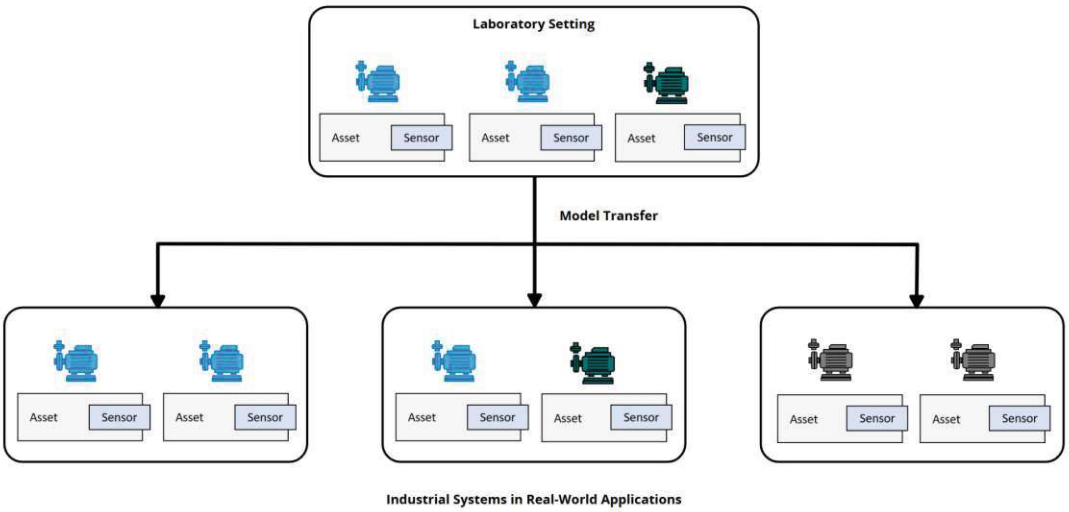

Figure 2.1: Process in industrial machine learning: (1) Model training in laboratory setting. (2) Model transfer to real-world deployments.

As illustrated in Fig. 2.1, the process of designing a machine learning model for an industrial application typically involves the following steps:

1. Model Training in a Laboratory Setting: Initially, data is collected from industrial assets within a controlled laboratory environment. This dataset is then utilized to train the machine learning model.

2. Model Deployment in Real-World Environments: After training, the model is deployed across various real-world settings. These environments often differ significantly from the laboratory setup, encompassing a wider range of asset and environmental characteristics.

The data acquisition process in controlled laboratory settings often follows standardized measurement protocols. While this controlled setting facilitates the collection of detailed sensor data under consistent conditions, it fails to produce the diverse dataset necessary for training a robust model. To address this limitation, it may be beneficial to adjust the laboratory setup to simulate distribution shifts, thereby generating a more varied dataset. However, a key challenge in this process is that the laboratory setting can only simulate a limited spectrum of the environmental influences that the model will encounter in actual deployment. Real-world environments can significantly differ from the laboratory setup, presenting a broader range of asset and environmental characteristics.

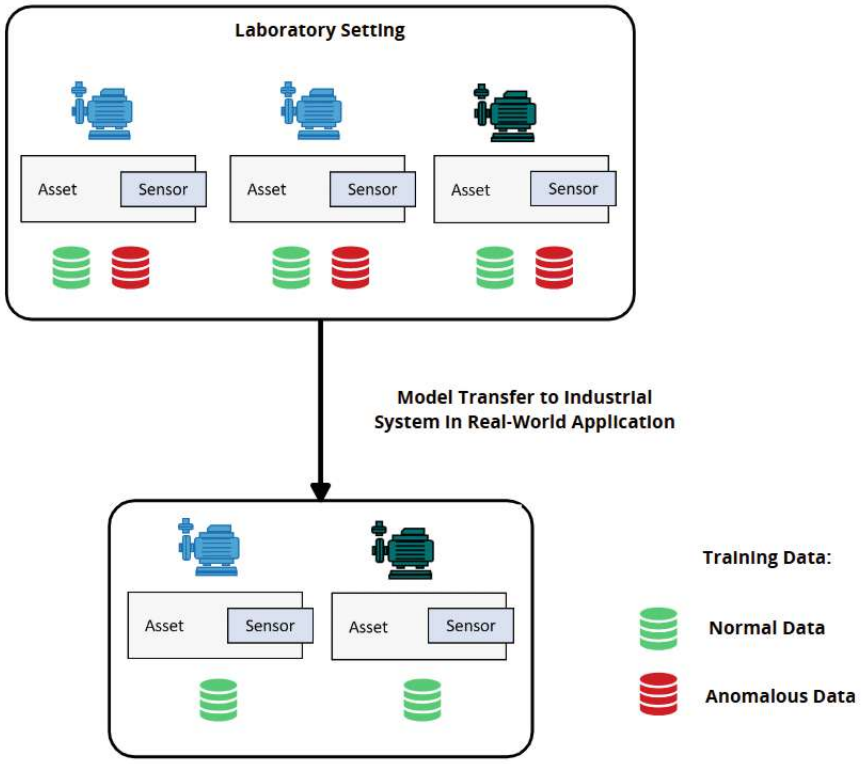

Figure 2.2: Illustration of training data distribution in industrial system: (1) Model training in laboratory setting. (2) Model transfer. (3) Model adaptation in real-world environments.

The deployment of machine learning models into real-world industrial systems poses challenges beyond distribution shifts. Another major challenge is the limited availability of anomalous data in real-world deployments. Acquiring anomalous data in large and complex industrial systems is both expensive and time-consuming. Normal operational data is typically available, while data on anomalous conditions is scarce in critical production environments [11]. Simulating failure types requires intervention from domain experts, a process that is extremely time-consuming for a large number of industrial assets and poses a substantial risk of material damage or complete system breakdown [14]. While it is possible to simulate anomalous conditions of industrial assets in laboratory settings with manageable risk, collecting data for every possible failure type from each individual asset within a large industrial system is infeasible.

For illustration, consider the Centrifugal-Pumps dataset detailed in Section 5.1. Normal state data for centrifugal pumps in industrial systems can be readily collected. However, simulating common failure types, such as hydraulic blockage, dry running, and cavitation, requires interruptions in continuous operation, resulting in unwanted downtime and potentially leading to the breakdown of the entire system [30]. This raises the question

whether it is possible to leverage normal operational data for efficient adaptation to a new environment, bridging the gap between the laboratory setting and the actual deployment environment, see Fig. 2.2.

2.2 Problem Statement

Consider a collection of datasets $(D_i)_{i\in\mathcal{E}}$ acquired in multiple domains $\mathcal{E}$. Each dataset $D_i := (\mathcal{X}_i, \mathcal{Y}_i)$ contains pairs of feature vectors $\mathcal{X}_i \subset \mathcal{X}$ and labels $\mathcal{Y}_i \subset \mathcal{Y}$. The domain set $\mathcal{E}$ is split into a collection of source and target domains $\mathcal{E}^{\text{source}} \cap \mathcal{E}^{\text{target}} = \emptyset$. The goal is to use $\mathcal{E}^{\text{source}}$ to learn a robust prediction model f_θ with parameters θ that can generalize to unseen domains $\mathcal{E}^{\text{target}}$. The performance on a domain is specified by a loss function $\mathcal{L}$ measuring the error between correct labels and those predicted by f_θ. Thus, the objective is

$$\theta^\star := \arg\min_\theta \mathop{\mathbb{E}}_{\substack{i\sim\mathcal{E} \\ D_i=(\mathcal{X}_i,\mathcal{Y}_i)}} \mathcal{L}\big(f_\theta(\mathcal{X}_i), \mathcal{Y}_i\big). \tag{2.1}$$

There are two common approaches to tackle the challenge of distribution shifts: domain generalization (DG) and domain adaptation (DA). DG methods train a model on the source domains, aiming to ensure the model performs well in a new target domain without further adaptation [21]. Unlike standard algorithms that focus on minimizing the empirical risk (ERM), DG methods frequently incorporate a penalty term to promote invariance across domains [17]. For instance, invariant risk minimization (IRM) aims to learn an invariant model across multiple domains with the ability to generalize to new domains by learning a data representation, typically through a neural network, such that the optimal classifier built upon this data representation is consistent across all source domains [1].

However, DG may not accurately reflect realistic scenarios [8]. In many real-world applications, the trained model is not expected to perform well out of the box in a new domain. Typically, the model has access to at least a small dataset from the target domain, allowing for some degree of domain-specific adaptation.

In contrast to DG, DA methods utilize unlabeled or sparsely labeled data in the target domain to adapt the model. Transfer learning is a form of DA, and leverages prior experience from source domains to improve learning on a target domain in terms of data efficiency, speed and performance [12].

In transfer learning, a model is initially pre-trained on source domains using standard learning techniques and subsequently fine-tuned on a smaller dataset from the target domain. However, the effectiveness of fine-tuning diminishes significantly when the target domain dataset is very small [13]. This dependency on the data volume of adaptation datasets can be a significant limitation in scenarios where acquiring a large amount of target domain data is impractical or impossible.

Meta-learning, on the other hand, is specifically tailored to few-shot learning settings and can effectively address both DA and DG tasks [12, 21, 20]. This is particularly useful in

scenarios where only a few examples from the target domain are available. Thus, meta-learning algorithms offer an effective approach for overcoming the challenges associated with distribution shifts in few-shot learning settings.

3 Meta-Learning

A defining trait of human intelligence is the ability to quickly learn new skills and quickly adapt to new tasks, such as recognizing objects from only a few examples [3]. The goal of meta-learning is the design of machine learning systems with similar properties [35]. In traditional machine learning, machine learning models are trained from scratch for a specific task [12]. This paradigm faces three significant challenges: data and computation bottlenecks, as well as generalization gaps [12].

First, the training of machine learning models requires large amounts of data and compute resources, excluding many applications where data is rare or expensive and compute resources are limited [12]. In medical diagnosis, data acquisition itself is expensive, e.g., in medical imaging where data annotation requires domain expertise [5]. Also, acquiring a large labeled dataset for every possible medical condition is infeasible, either due to privacy concerns or the rarity of certain conditions. Similarly, the operation of an industrial system under anomalous conditions poses the risk of expensive material damage, arising from wear and tear or even complete system breakdown [14]. Hence, the collection of data in industrial systems and many other real-world applications is difficult.

Second, the generalization ability of machine learning systems has been driven by the wealth of available data. Large and diverse datasets are crucial for achieving broad generalization in machine learning models, enabling the model to learn robust patterns [12]. This is evident in computer vision, where datasets like ImageNet [26] have improved the performance of image classification models. Although these pre-trained models can be effectively applied to related computer vision tasks, such extensive datasets are often unavailable in many specialized domains, such as time-series domains.[12]. The scarcity of data and limited computational resources make it impractical to train models from scratch for each medical condition, industrial system, or even individual assets within an industrial system. Meta-learning aims to address these challenges by leveraging prior experience across multiple tasks with limited datasets to develop models capable of quick adaptation and generalization to new tasks. Meta-learning is commonly understood as 'learning to learn', referring to the process of improving a learning algorithm [12].

The classification of handwritten characters has received significant attention in machine learning research [19]. While traditional machine learning models have achieved high performance on the standard MNIST dataset, which provides thousands of examples, they still fall short of human-level competence [19]. To address this gap, meta-learning datasets have been introduced, offering a setting that challenges machine learning models to learn from only a few examples. The Omniglot dataset [19] contains 1623 handwritten characters from 50 different alphabets, with each character being drawn by one of 20 different

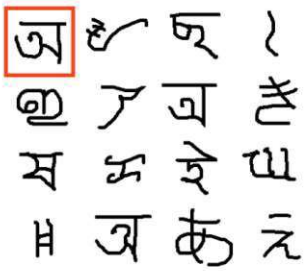

Figure 3.1: Illustration of one-shot learning on the Omniglot dataset [19].

individuals, see Fig. 3.1. Humans can effortlessly label a new character from an unknown alphabet after seeing only one example of that character [19]. This capability is rooted in the human brain's remarkable ability to generalize from limited examples, leveraging prior knowledge and learning new patterns rapidly. In contrast, traditional machine learning models struggle with this task, typically requiring extensive training data to achieve high performance and lacking the ability to naturally generalize well from a few examples.

3.1 Problem Statement

In supervised meta-learning, a dataset $D := \{(x_i, y_i)\}$ contains pairs of feature vectors $x_i \in \mathcal{X} \subset \mathbb{R}^n$ and labels $y_i \in \mathcal{Y} \subset \mathbb{N}$. The training dataset D^{train} is divided into a collection of meta-training tasks $\mathcal{T}_i^{\text{train}}$, the test dataset D^{test} is divided into a collection of meta-testing tasks $\mathcal{T}_i^{\text{test}}$. Each task $\mathcal{T}_i$ is associated with two disjoint sets, a support set S_i and a query set Q_i [24]. In meta-training, the objective is to learn a model f_θ, parameterized by θ, that can generalize across tasks [24]. The performance on a task is specified by a loss function $\mathcal{L}$ which measures the error between correct labels and those predicted by f_θ. The support set S_i^{train} is used for adapting the model on a task $\mathcal{T}_i^{\text{train}}$, while the query set Q_i^{train} is used to evaluate this adaptation [24]. The generalization performance on a meta-testing task $\mathcal{T}_i^{\text{test}}$ is assessed by the loss on the query set Q_i^{test} after adaptation on the support set S_i^{test} [24].

In traditional machine learning, the objective is to learn a model f_θ, parameterized by θ, that maximizes the log-probability of the correct labels,

$$\theta^\star := \arg\max_\theta \mathop{\mathbb{E}}_{(x,y)\sim D^{\text{train}}} \log p_\theta(y|x). \tag{3.1}$$

In meta-learning, however, the goal is to optimize the model's performance across various tasks, rather than a single training dataset. The objective shifts to learning a model f_θ that utilizes the support set for adaptation to a task,

$$\theta^\star := \arg\max_\theta \mathop{\mathbb{E}}_{\substack{\mathcal{T}\sim p(\mathcal{T}) \\ \mathcal{T}=(S,Q)}} \sum_{(x,y)\in Q} \log p_\theta(y|x, S), \tag{3.2}$$

where $p(\mathcal{T})$ denotes a distribution over tasks. A standard assumption in meta-learning is that both meta-training and meta-testing tasks are drawn from the same task distribution [3]. During meta-training, the model is explicitly trained to quickly adapt to new tasks, effectively learning 'how to learn'. Here, 'quickly' refers to the ability to learn from a small number of examples. In N-way K-shot learning, the support and query sets of a task consist of K feature-label pairs per class, with N randomly sampled classes.

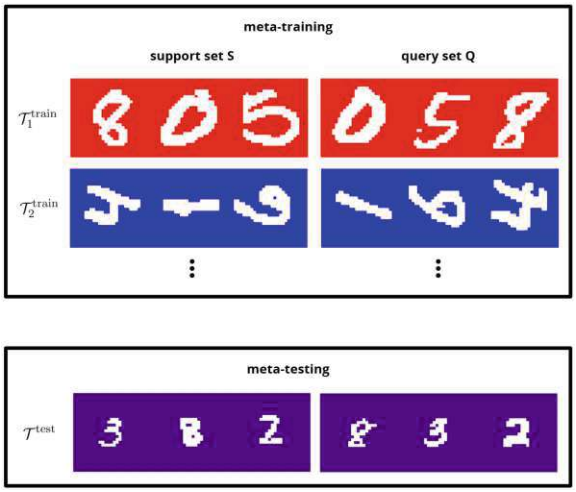

Figure 3.2: Example of 3-way 1-shot learning on the Rainbow-MNIST dataset [4].

The Rainbow-MNIST dataset [4] is created from the MNIST dataset. It provides 56 domains, each domain corresponding to a background color ('red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet'), degree of rotation (0°, 90°, 180°, 270°) and scale size ('full', 'half'). Fig. 3.2 illustrates 3-way 1-shot learning on the Rainbow-MNIST dataset. The domains are divided into meta-training and meta-testing domains. Meta-training tasks are sampled from the meta-training domains, meta-testing tasks are sampled from the meta-testing domains. In meta-training, the model learns to adapt to a new task utilizing only one example per class (meta-training support). In meta-testing, the model is given one example per class from a previously unseen task (meta-testing support), and its performance is evaluated based on its ability to generalize to unseen data from that task (meta-testing query).

There are three common approaches to meta-learning: metric-based, model-based and optimization-based meta-learning [31]. These approaches differ in how they leverage the support set S to model the probability distribution over labels $p_\theta(y|x, S)$, see Table 3.1. Metric-based meta-learning focuses on learning a similarity measure $k_{\theta,S}$ to compare new examples with those in the support set, enabling classification based on proximity in an embedding space [31]. Model-based meta-learning, on the other hand, directly trains models to represent $p_\theta(y|x, S)$, using architectures specifically designed for rapid adaptation to new tasks [31]. Finally, optimization-based meta-learning aims to optimize the learning process itself by learning an adaptation procedure $\mathcal{A}lg(\theta, S)$ [24]. The adaptation procedure computes effective task-specific parameters using the examples from the support set.

Each approach offers a unique method for improving the model's ability to learn from a small number of examples and generalize to new tasks effectively.

	Metric-Based	Model-Based	Optimization-Based
$p_\theta(y\|x,S)$	$\sum_{(x_i,y_i)\in S} k_{\theta,S}(x,x_i)$	$f_\theta(x,S)$	$p_{\mathcal{A}lg(\theta,S)}(y\|x)$

Table 3.1: Meta-learning approaches leverage the support set to model the probability distribution over labels $p_\theta(y|x,S)$ [31].

3.2 Metric-Based Meta-Learning

The key idea of metric-based meta-learning is learning a robust embedding function, parameterized by θ, which maps features to an embedding space such that similar features are situated close to each other [31]. The similarity of these embedded features is measured using an appropriate distance function. Given tasks of the form $\mathcal{T} := (S, Q)$, the probability distribution over labels y of a query feature x given a support set S is determined by the distance between x and the support features in the learned embedding space. By normalizing these distances, a probability distribution is obtained. This probability distribution can be modeled as

$$p_\theta(y|x,S) = \sum_{(x_i,y_i)\in S} k_{\theta,S}(x,x_i), \tag{3.3}$$

where $k_{\theta,S}(x,x_i)$ is a learned similarity measure comparing x with the support feature x_i, based on the learned embedding function and the distance function [31].

3.2.1 Siamese Neural Network

The key idea of Siamese Neural Networks for one-shot learning [16] is to learn feature representations via an embedding function and then use a distance function to rank similarity between embedded features. The Siamese Neural Network consists of twin networks $f_{\theta'}$ with shared parameters θ'. The embedding function $f_{\theta'} : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$ is a sequence of convolutional layers and maps a feature from the n-dimensional feature space into a m-dimensional embedding space.

In meta-training, the Siamese Neural Network is trained for a verification task, learning to identify feature pairs based on the probability that they share the same label [16]. Given a feature pair (x_i, x_j), the pair is labeled $y := \chi_{y_i=y_j} \in \{0,1\}$ to indicate whether the features are similar or dissimilar. The feature pair is passed through the twin networks, i.e., the embeddings $f_{\theta'}(x_i)$ and $f_{\theta'}(x_j)$ are computed. These embeddings are then compared using the L^1 element-wise distance $d : \mathbb{R}^m \times \mathbb{R}^m \to [0,+\infty)^m$,

$$d\big(f_{\theta'}(x_i), f_{\theta'}(x_j)\big) := \Big(|f_{\theta'}(x_i)_k - f_{\theta'}(x_j)_k|\Big)_{k=1}^{m}. \tag{3.4}$$

This distance is converted to a probability $p_\theta(x_i, x_j)$ by a linear layer with weights $W \in \mathbb{R}^m$ and a sigmoid layer σ,

$$p_\theta(x_i, x_j) := \sigma\Big(W^\top d\big(f_{\theta'}(x_i), f_{\theta'}(x_j)\big)\Big) \in (0, 1), \tag{3.5}$$

where $\theta := (\theta', W)$. Using the cross entropy loss, the loss on a batch B of feature-label pairs is given by

$$\mathcal{L}_\theta(B) := \sum_{(x_i,y_i),(x_j,y_j)\in B} \chi_{y_i=y_j} \log p_\theta(x_i, x_j) + (1 - \chi_{y_i=y_j}) \log\big(1 - p_\theta(x_i, x_j)\big). \tag{3.6}$$

Then, the training objective becomes

$$\theta^\star := \arg\min_\theta \mathop{\mathbb{E}}_{B\sim D^{\text{train}}} \mathcal{L}_\theta(B),$$

where B is a subset of D^{train} and consists of feature-label pairs that include only a subset of the label space $(y_i)_{i=1}^N \subset \mathcal{Y}$. The Siamese Neural Network is trained by minimizing the loss with respect to the parameters θ via stochastic gradient descent. It is important to note that the meta-training process aligns with the conventional machine learning objective, as outlined in Eq. 3.1, and does not integrate the task-specific setup characteristic of meta-learning. This is based on the assumption that networks trained for verification can generalize to one-shot classification [16]. Provided that the network has been exposed to a variety of data, the network can be used for measuring the similarity between unknown labels.

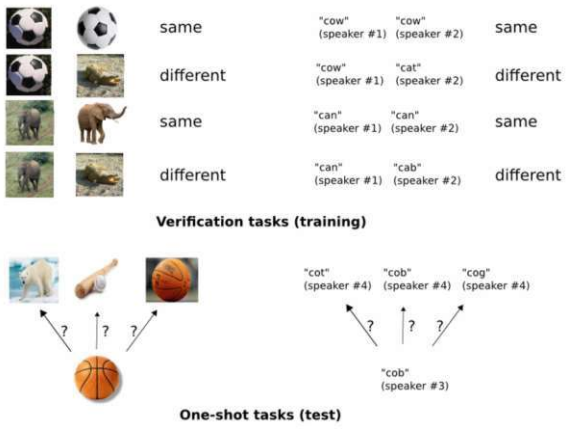

Figure 3.3: Diagram of Siamese Neural Network strategy. The model is first trained to discriminate between same/different pairs and then used to rank similarity between given support examples and an unknown example [16].

At meta-testing time, a N-way 1-shot learning setup is considered. Thus, the support set of a task $\mathcal{T} = (S, Q)$ contains one feature-label pair per class, $S = (x_i, y_i)_{i=1}^N$. The Siamese Neural Network compares a query feature x to every support feature x_i. Note that $p_\theta(x, x_i)$

corresponds to the probability $p_\theta(y = y_i|x, S)$. The predicted label $\hat{y}$ of a query feature x is then given by the highest probability,

$$i^\star := \arg\max_{i=1,\dots,N} p_{\theta^\star}(x, x_i),$$
$$\hat{y} := y_{i^\star}.$$

3.2.2 Matching Networks

Matching Networks [32] are based on the simple machine learning principle that training and testing conditions must match, leveraging principles of metric learning and attention mechanisms. In meta-training, a mapping $h : S \mapsto h_S$ is learned that maps a support set S to a classifier h_S. The classifier h_S defines a probability distribution over labels.

Given tasks of the form $\mathcal{T} := (S, Q)$ with a set of task labels $C \subset \mathcal{Y}$, the key idea is to learn an embedding function g for the support features and an embedding function f for the query features mapping the support and query features into a shared embedding space, as illustrated in Fig. 3.4. Let $f_{\theta_1}, g_{\theta_2} : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$ denote these embedding functions. f_{θ_1} is an LSTM and g_{θ_2} is a bidirectional LSTM.

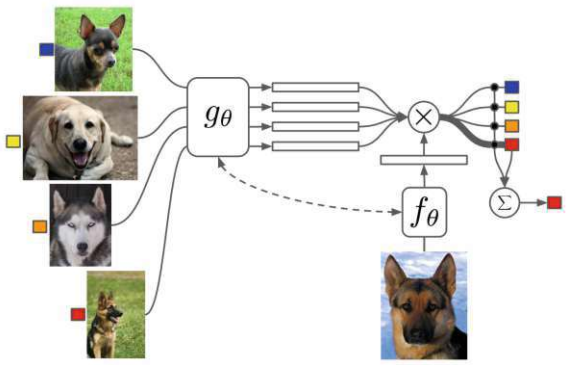

Figure 3.4: Diagram of the Matching Network. The model learns an embedding function that maps support examples and a query example into a shared embedding space [32].

An attention mechanism a is used to assign weights to support examples. Let $a_\theta : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ denote an attention mechanism with parameters $\theta := (\theta_1, \theta_2)$. In the simplest form, the attention mechanism is a softmax function over distances between support and query features in the embedding space measuring the similarity between the query feature x and each support feature x_i. Let $d : \mathbb{R}^m \times \mathbb{R}^m \to [0, +\infty)$ denote a distance function, such as the cosine similarity function $d(x, x_i) := \frac{x \cdot x_i}{\|x\|\|x_i\|}$. The attention mechanism is then given by

$$a_\theta(x, x_i) := \frac{\exp\Big(d\big(f_{\theta_1}(x), g_{\theta_2}(x_i)\big)\Big)}{\sum_{(x_j, y_j) \in S} \exp\Big(d\big(f_{\theta_1}(x), g_{\theta_2}(x_j)\big)\Big)}.$$

The classifier $h_S : \mathcal{X} \to [0,1]^{|C|}, h_S(x) := \big(p_\theta(y = c|x, S)\big)_{c \in C}$ is fully specified by the attention mechanism a_θ [32], where the probability of a query feature x belonging to class $c \in C$ is computed as the sum of the corresponding attention weights,

$$p_\theta(y = c|x, S) := \sum_{(x_i, c) \in S} a_\theta(x, x_i).$$

In meta-training, the classifier is trained by maximizing the log-probability of the correct label of the query examples via stochastic gradient descent,

$$\theta^\star := \arg\max_\theta \mathop{\mathbb{E}}_{\substack{\mathcal{T} \sim p(\mathcal{T}) \\ \mathcal{T} = (S,Q)}} \sum_{(x_i, y_i) \in Q} \log p_\theta(y = y_i|x_i, S). \tag{3.7}$$

In meta-testing, given a task $\mathcal{T} := (S, Q)$ with a set of task labels C, the predicted label of a query feature x is computed as

$$\hat{y} := \arg\max_{c \in C} p_{\theta^\star}(y = c|x, S).$$

Long Short-Term Memory (LSTM)

The LSTM network [10] is a recurrent network architecture with an efficient, gradient-based learning algorithm that enables learning long-term dependencies and overcomes the exploding and vanishing gradient problems [10]. The recurrent hidden layers include so-called memory blocks, each containing an input gate, an output gate and a forget gate [28]. The LSTM network maps an input sequence $x := (x_t)_{t=1}^T$ of elements $x_t \in \mathbb{R}^n$ to an output sequence $y := (y_t)_{t=1}^T$ of elements $y_t \in \mathbb{R}^n$. The key idea of LSTM networks is based on a cell state c_t controlling the flow of information [23]. A memory block processes the current sequence element x_t, the cell output h_{t-1} and the cell state c_{t-1} from the previous memory block [23], see Fig. 3.5. It then updates the cell state to c_t and computes the cell output h_t [23].

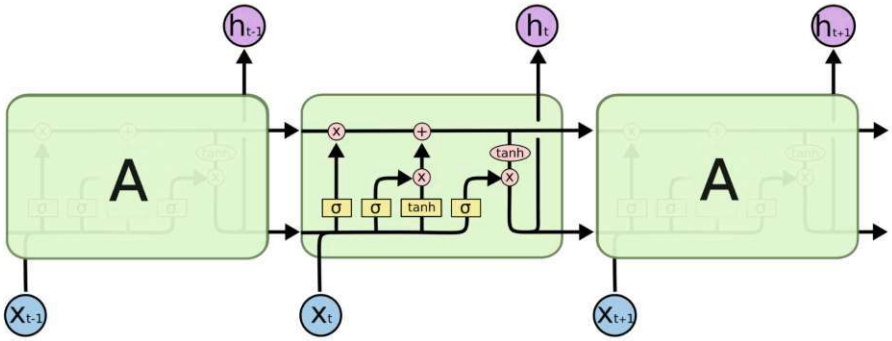

Figure 3.5: Memory block in a recurrent LSTM layer [23].

The forget and input gates regulate the flow of information from the previous cell state c_{t-1} to the next cell state c_t [23]. The forget gate enables the cell state to reset itself, while

the input gate enables the cell state to store new information [6]. The forget gate vector f_t and the input gate vector i_t are computed by applying a sigmoid activation function σ to a linear combination of the current sequence component x_t and the previous cell output h_{t-1} [23], see Eq. 3.8, 3.9. The candidate cell state g_t represents the new information that could be added to the current cell state c_{t-1} [23]. It is computed by applying the hyperbolic tangent function tanh to a linear combination of the current sequence element x_t and the previous cell output h_{t-1} [23], see Eq. 3.10. The forget gate vector f_t determines which information from the previous cell state c_{t-1} should be retained or discarded, while the input gate vector i_t determines how much of the newly computed candidate cell state g_t will be added to the next cell state c_t [23], see Eq. 3.12. The output gate controls the extend to which the updated cell state c_t contributes to the current cell output h_t [23], see Eq. 3.13. In summary, the LSTM network maps the input sequence x to the output sequence y by computing iteratively

$$i_t := \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i), \tag{3.8}$$
$$f_t := \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f), \tag{3.9}$$
$$g_t := \tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g), \tag{3.10}$$
$$o_t := \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o), \tag{3.11}$$
$$c_t := f_t \odot c_{t-1} + i_t \odot g_t, \tag{3.12}$$
$$h_t := o_t \odot \tanh(c_t), \tag{3.13}$$
$$y_t := W_y h_t + b_y. \tag{3.14}$$

The $W_{(\cdot)x} \in \mathbb{R}^{m\times n}$, $W_{(\cdot)h} \in \mathbb{R}^{m\times m}$, $W_y \in \mathbb{R}^{n\times m}$ terms denote weight matrices, and the $b_{(\cdot)} \in \mathbb{R}^m$, $b_y \in \mathbb{R}^n$ terms denote bias vectors where m is the number of hidden units [28]. $\odot$ is the element-wise product. The bidirectional LSTM network processes the input sequence in both forward and backward directions [7].

3.2.3 Prototypical Networks

Prototypical Networks [29] are based on the idea that there exists an embedding in which features cluster around a prototype representation for each class. Therefore, a non-linear mapping $f_\theta : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$ from the n-dimensional feature space into a m-dimensional embedding space is learned using a neural network with parameters θ. Given tasks of the form $\mathcal{T} := (S, Q)$ with a set of task labels $C \subset \mathcal{Y}$, the prototype v_c of class $c \in C$ is the mean of the embedded support features,

$$v_{S,c} := \frac{1}{|S_c|} \sum_{(x_i,y_i)\in S_c} f_\theta(x_i),$$

where $S_c := \{(x_i, y_i) \in S | y_i = c\}$.

Let $d : \mathbb{R}^m \times \mathbb{R}^m \to [0, +\infty)$ denote a distance function, such as the Euclidean distance. A softmax function is used to convert the distances to the prototypes in the embedding space

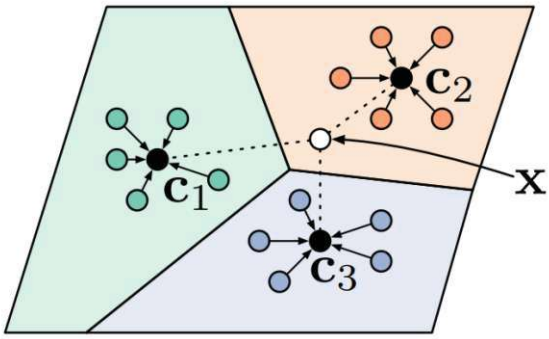

Figure 3.6: Illustration of prototypes in the embedding space [29].

to a probability distribution over labels. Specifically, the probability distribution p_θ for a query feature x is given by

$$p_\theta(y = c|x, S) := \frac{\exp\Big(-d\big(f_\theta(x), v_{S,c}\big)\Big)}{\sum_{k\in C}\exp\Big(-d\big(f_\theta(x), v_{S,k}\big)\Big)}.$$

In meta-training, the neural network is trained by minimizing the negative log-probability of the correct label of the query examples via stochastic gradient descent,

$$\theta^\star := \arg\min_\theta \mathop{\mathbb{E}}_{\substack{\mathcal{T}\sim p(\mathcal{T})\\ \mathcal{T}=(S,Q)}} \sum_{(x_i,y_i)\in Q} -\log p_\theta(y = y_i|x_i, S).$$

In meta-testing, given a task $\mathcal{T} := (S, Q)$ with a set of task labels $C \subset \mathcal{Y}$, the predicted label of a query feature x is computed as

$$\hat{y} := \arg\max_{c\in C} p_{\theta^\star}(y = c|x, S).$$

This corresponds to finding the nearest prototype in the embedding space [29].

3.3 Model-based Meta-Learning

Model-based meta-learning directly trains models f_θ, parameterized by θ, to represent the probability distribution $p_\theta(y|x, S)$. f_θ can be any model capable of learning in a few-shot learning setting [31]. Given N-way K-shot learning tasks $\mathcal{T} = (S, Q)$ with task labels $(k)_{k=1}^N \subset \mathcal{Y}$, the model f_θ processes the entire support set S and a query feature x, outputting a probability distribution over labels, as illustrated in Fig. 3.7. The probability distribution is given by $p_\theta(y|x, S) := f_\theta(x, S) \in [0, 1]^N$. During meta-training, the log-probability of the correct labels for the query examples is maximized,

$$\theta^\star := \arg\max_\theta \mathop{\mathbb{E}}_{\substack{\mathcal{T}\sim p(\mathcal{T})\\ \mathcal{T}=(S,Q)}} \sum_{k=1}^{N} \sum_{(x,k)\in Q} \log f_\theta(x, S)_k. \tag{3.15}$$

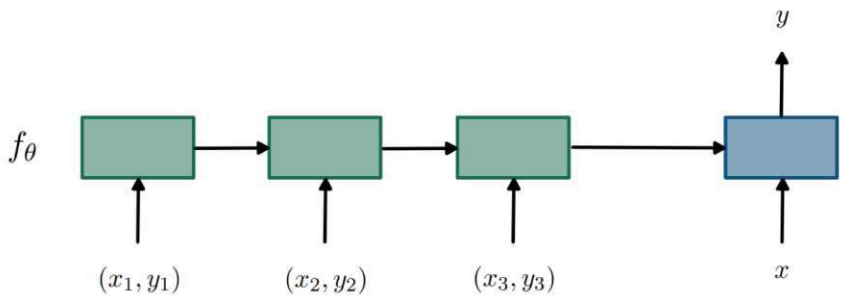

Figure 3.7: The meta-learner f_θ takes as input the entire support set $S := (x_i, y_i)_{i=1}^3$ along with a query feature x [31].

3.4 Optimization-Based Meta-Learning

The key idea of optimization-based meta-learning is to optimize the learning process itself [12]. While gradient-based optimization has seen great success in a variety of learning problems, these achievements have relied on many iterative updates across many training examples [25]. Gradient-based optimization algorithms are not tailored to few-shot learning, as they are not designed to learn with a small number of training examples or converge within a small number of updates [25]. Optimization-based meta-learning aims to adjust gradient-based optimization to a small data regime.

The optimization-based meta-learning approach frames the meta-training process as a bi-level optimization problem. The goal is to learn meta-parameters that produce good task-specific parameters after an adaptation procedure $\mathcal{A}lg$ [24]. $\mathcal{A}lg$ corresponds to an algorithm that computes task-specific parameters ϕ_i using a set of meta-parameters θ and the support set S_i of a task $\mathcal{T}_i$ [24]. Given a meta-batch $(\mathcal{T}_i)_{i=1}^N$ of meta-training tasks $\mathcal{T}_i = (S_i, Q_i)$, the meta-learning problem in Eq. 3.2 can be formalized as

$$\theta^\star := \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(Q_i | \phi_i^\star(\theta)\big) \tag{3.16}$$

$$\text{s.t. } \phi_i^\star(\theta) := \mathcal{A}lg(\theta, S_i). \tag{3.17}$$

Since $\mathcal{A}lg(\theta, S_i)$ is typically interpreted as solving explicitly [3] or implicitly [25] an underlying optimization problem, this is viewed as a bi-level optimization problem [24]. The learned model outputs a probability distribution over labels,

$$p_\theta(y|x, S) = p_{\mathcal{A}lg(\theta,S)}(y|x),$$

where the adaptation procedure $\mathcal{A}lg$ utilizes the support set S to find good task-specific parameters given a set of meta-parameters θ [24].

This framework satisfies the machine learning principle that training and testing conditions must match: In meta-training, the support set is used for the adaptation procedure on a

task and the query set is used for evaluating this adaptation. Specifically, the goal for a task $\mathcal{T}_i = (S_i, Q_i)$ is to learn task-specific parameters utilizing the support set S_i such that the loss on the query set Q_i is minimal [24]. Consequently, at meta-testing time, good generalization performance can be achieved using the meta-learned parameters $\theta^\star$. The generalization performance on a meta-testing task $\mathcal{T}_i^{\text{test}}$ is measured by the loss on the query set Q_i^{test} after performing the adaptation procedure on the support set S_i^{test} with meta-learned parameters $\theta^\star$ [24].

3.4.1 Meta-Learner LSTM

The key idea is to train a *meta-learner* to optimize a *learner* classifier [25]. The learner M_ϕ is a neural network parameterized by ϕ, mapping a feature $x \in \mathcal{X}$ to a probability distribution over labels, $M_\phi(x) := p_\phi(y|x)$. The meta-learner, R_θ, is based on an LSTM network parameterized by θ and trained to optimize the learner M_ϕ. Therefore, the LSTM-based meta-learner receives a series of support set gradients and losses from the learner and proposes new learner parameters. The objective is to minimize the loss of the proposed learner on a query set with respect to the meta-learner parameters [25]. Note that a task $\mathcal{T} = \left((S_t)_{t=1}^T, Q\right)$ is associated with a series of support sets $(S_t)_{t=1}^T$ and a single query set Q. In the following, I will write $S_t = (X_t, Y_t)$ and $Q = (X, Y)$.

In the bi-level optimization view, the inner-level algorithm $\mathcal{A}lg(\theta, (S_t)_{t=1}^T)$ in Eq. 3.17 refers to a forward pass of the meta-learner R_θ and computes task-specific parameters ϕ for the learner M_ϕ. The outer-level problem in Eq. 3.16 is solved via gradient descent. The loss function is the negative log-probability of the correct label [25]. The loss in Eq. 3.16 is defined as the query set loss of the learner,

$$\mathcal{L}(Q|\phi) := \mathcal{L}(M_\phi(X), Y). \tag{3.18}$$

The learner M_ϕ is not explicitly trained via gradient descent; instead, its parameters ϕ are updated by the LSTM-based meta-learner R_θ. This is motivated by the similarity between gradient descent updates and cell state updates [25]. The gradient descent algorithm uses updates of the form

$$\phi_t := \phi_{t-1} - \alpha_t \nabla_{\phi_{t-1}} \mathcal{L}_t, \tag{3.19}$$

where ϕ_{t-1} are the current parameters, α_t is the current learning rate, $\mathcal{L}_t$ is the loss of the classifier $M_{\phi_{t-1}}$, $\nabla_{\phi_{t-1}}\mathcal{L}_t$ is the gradient of that loss, and ϕ_t are the updated parameters [25]. This update resembles the cell state update within a memory block of an LSTM network, see Eq. 3.12, if $f_t = 1$, $c_{t-1} = \phi_{t-1}$, $i_t = \alpha_t$, and $g_t = -\nabla_{\phi_{t-1}}\mathcal{L}_t$ [25]. However, fixing the input gate vector i_t and the forget gate vector f_t might not be the optimal choice [25]. Thus, they are learned as well by the meta-learner,

$$i_t := \sigma(W_i \cdot [\nabla_{\phi_{t-1}}\mathcal{L}_t, \mathcal{L}_t, \phi_{t-1}, i_{t-1}] + b_i), \tag{3.20}$$
$$f_t := \sigma(W_f \cdot [\nabla_{\phi_{t-1}}\mathcal{L}_t, \mathcal{L}_t, \phi_{t-1}, f_{t-1}] + b_f). \tag{3.21}$$

The initial value of the cell state c_0 is also treated as a parameter of the meta-learner R_θ, corresponding to the initialization of the learner classifier M_ϕ [25]. Given a task $\mathcal{T} = \big((S_t)_{t=1}^T, Q\big)$, the learner parameters are passed through the meta-learner,

$$\mathcal{L}_t := \mathcal{L}(M_{\phi_{t-1}}(X_t), Y_t), \tag{3.22}$$

$$c_t := R_\theta(\nabla_{\phi_{t-1}}\mathcal{L}_t, \mathcal{L}_t), \tag{3.23}$$

$$\phi_t := c_t. \tag{3.24}$$

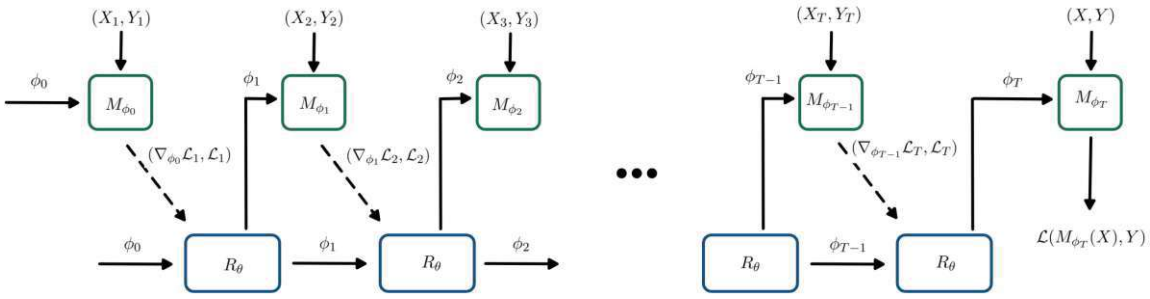

Figure 3.8: Computational graph for the forward pass of the meta-learner [25]. The dashed arrows indicate steps where back-propagation is omitted during the training of the meta-learner R_θ [25].

The learner loss $\mathcal{L}_t$ and the gradient $\nabla_{\phi_{t-1}}\mathcal{L}_t$ of that loss are computed, see Eq. 3.22. In Eq. 3.23, the meta-learner receives the learner gradient and loss, and computes the cell state c_t using Eq. 3.12. The learner parameters are updated to the cell state, see Eq. 3.24. After T updates, the proposed learner M_{ϕ_T} is evaluated on the query set Q, $\mathcal{L} := \mathcal{L}(M_{\phi_T}(X), Y)$. The meta-learner parameters θ are updated via gradient descent using $\nabla_\theta\mathcal{L}$. Note that the series of gradients and losses $(\nabla_{\phi_{t-1}}\mathcal{L}_t, \mathcal{L}_t)_{t=1}^T$ is dependent on the meta-learner parameters θ. Thus, the meta-learner gradient $\nabla_\theta\mathcal{L}$ involves second-order derivatives. In order to avoid these second-order derivatives, the dependence of the learner gradients and losses on the meta-learner parameters is ignored in the computation of the meta-learner gradients [25]. The corresponding computation graph is shown in Fig. 3.8.

3.4.2 Model-Agnostic Meta-Learning

MAML

The key idea of model-agnostic meta-learning (MAML) [3] is to learn an initialization for the parameters θ of a neural network f_θ. The neural network f_θ maps a feature $x \in \mathcal{X}$ to a probability distribution over labels, $f_\theta(x) := p_\theta(y|x)$. In contrast to the LSTM meta-learner [25], MAML explicitly optimizes for initial parameters that enable rapid adaptation to a new task. MAML ensures that with only a few gradient descent steps on a few examples of a new task, the model can achieve good task-specific parameters [3], as illustrated in Fig. 3.9.

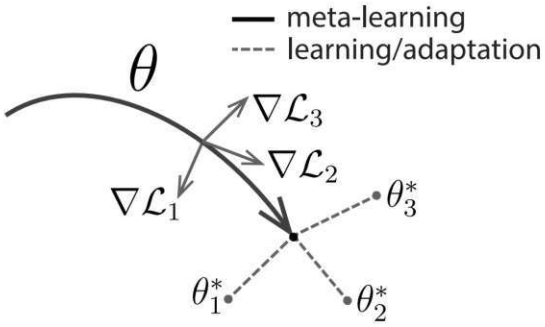

Figure 3.9: Diagram of the MAML algorithm [3].

The notation introduced here will be used throughout the work. Let $\mathcal{L}$ represent a loss function, such as the categorical cross-entropy loss,

$$\mathcal{L}(f_\theta(x), y) := -\sum_{y' \in \mathcal{Y}} \Big(\chi_{y'=y} \ \log p_\theta(y = y'|x)\Big). \tag{3.25}$$

Then, $\mathcal{L}_D(\theta)$ denotes the loss on dataset D based on f_θ, defined as a function of the parameters θ,

$$\mathcal{L}_D(\theta) := \frac{1}{|D|} \sum_{(x,y)\in D} \mathcal{L}(f_\theta(x), y). \tag{3.26}$$

In the bi-level optimization view, the inner-level and outer-level problems are both minimized via gradient descent. The inner-level algorithm $\mathcal{A}lg(\theta, S_i)$ in Eq. 3.17 corresponds to a few gradient descent steps on the support set S_i initialized at meta-parameters θ. For simpler notation, only one gradient descent step is used in the following. By this, MAML explicitly optimizes for parameters θ such that one gradient descent step on a small number of examples from a new task $\mathcal{T}_i$ produces good task-specific parameters $\phi_i(\theta)$,

$$\mathcal{A}lg(\theta, S_i) := \phi_i(\theta) := \theta - \alpha \ \nabla\mathcal{L}_{S_i}(\theta). \tag{3.27}$$

The outer-level problem in Eq. 3.16 is solved by performing a gradient descent step on the corresponding query sets,

$$\theta^\star := \theta - \beta \ \nabla\Big(\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{Q_i}\big(\phi_i(\cdot)\big)\Big)(\theta) = \theta - \frac{\beta}{N}\sum_{i=1}^{N} \nabla\big(\mathcal{L}_{Q_i} \circ \phi_i\big)(\theta). \tag{3.28}$$

Note that the outer-level loss in Eq. 3.16 is computed based on the task-specific parameters ϕ, however, the outer-level optimization is performed over the meta-parameters θ [3]. Therefore, MAML can be viewed as a form of transfer learning with many tasks, where parameters are differentiated through the fine-tuning process, i.e., the adaptation procedure $\mathcal{A}lg$ [22]. In classical transfer learning, a model is first pre-trained on a large dataset and subsequently fine-tuned on a smaller dataset. However, this approach does not guarantee that the pre-trained model will be well-suited for fine-tuning [22]. In contrast, MAML directly optimizes for initial parameters so that the model performs well on the query set after fine-tuning on the support set [22].

First-Order MAML (FOMAML)

Since the meta-gradient $\nabla\big(\mathcal{L}_{Q_i} \circ \phi_i\big)(\theta)$ in Eq. 3.28 is computed with respect to the meta-parameters θ, the meta-gradient update involves second-order derivatives. First-order model-agnostic meta-learning (FOMAML) is a first-order approximation of MAML where these second-order derivatives are omitted [3],

$$\begin{aligned}
\nabla\big(\mathcal{L}_{Q_i} \circ \phi_i\big)(\theta) &= d(\mathcal{L}_{Q_i} \circ \phi_i)(\theta)^\top && (3.29)\\
&= \Big(d\mathcal{L}_{Q_i}\big(\phi_i(\theta)\big)\; d\phi_i(\theta)\Big)^\top && (3.30)\\
&= \Big(d\mathcal{L}_{Q_i}\big(\phi_i(\theta)\big)\big(I - \alpha\,\nabla^2\mathcal{L}_{S_i}\big(\phi_i(\theta)\big)\big)\Big)^\top && (3.31)\\
&\approx d\mathcal{L}_{Q_i}\big(\phi_i(\theta)\big)^\top && (3.32)\\
&= \nabla\mathcal{L}_{Q_i}\big(\phi_i(\theta)\big). && (3.33)
\end{aligned}$$

The meta-gradient update then becomes

$$\theta^\star = \theta - \frac{\beta}{N}\sum_{i=1}^{N}\nabla\mathcal{L}_{Q_i}\big(\phi_i(\theta)\big). \tag{3.34}$$

Implicit MAML (iMAML)

Let ϕ^k denote k gradient descent steps,

$$\phi^k := \underbrace{\phi \circ \cdots \circ \phi}_{k \text{ steps}}. \tag{3.35}$$

In the general case, MAML solves the inner-level problem in Eq. 3.17 by taking k gradient descent steps on the support set,

$$\mathcal{A}lg(\theta, S_i) := \phi_i^k(\theta) = \theta - \alpha\sum_{j=0}^{k-1}\nabla\mathcal{L}_{S_i}\big(\phi_i^j(\theta)\big). \tag{3.36}$$

The computation of the meta-gradient $\nabla(\mathcal{L}_Q \circ \phi^k)(\theta)$ in Eq. 3.28 then involves the computation of k second-order derivatives,

$$\begin{aligned}
\nabla(\mathcal{L}_Q \circ \phi^k)(\theta) &= d(\mathcal{L}_Q \circ \phi^k)(\theta)^\top && (3.37)\\
&= \Big(d\mathcal{L}_Q\big(\phi^k(\theta)\big)d\phi^k(\theta)\Big)^\top && (3.38)\\
&= d\phi^k(\theta)^\top d\mathcal{L}_Q\big(\phi^k(\theta)\big)^\top && (3.39)\\
&= \prod_{j=0}^{k-1}\Big(I - \alpha\,\nabla^2\mathcal{L}_S\big(\phi^j(\theta)\big)\Big)\nabla\mathcal{L}_Q\big(\phi^k(\theta)\big). && (3.40)
\end{aligned}$$

The derivative $d\phi^k(\theta)$ is derived in Section 4.2. Thus, for a large number of inner gradient descent steps, the meta-gradient computation requires significant computational and memory resources [24]. Implicit MAML (iMAML) [24] solves this problem by approximating the inner-level derivative $d\phi^k(\theta)$ with a term that requires the computation of only one second-order derivative.

In order to prevent over-fitting in the inner-level problem in Eq. 3.27, $\mathcal{A}lg(\theta, S)$ needs to incorporate some form of regularization [24]. While MAML uses a small number of gradient descent steps as a form of regularization, iMAML explicitly adds a proximal regularization term with regularization strength $\lambda > 0$ [24]. Let $\phi^\star(\theta)$ denote the exact solution to the following optimization problem,

$$\phi^\star(\theta) := \arg\min_{\phi} \mathcal{L}_S(\phi) + \frac{\lambda}{2}\|\phi - \theta\|^2. \tag{3.41}$$

Rajeswaran et al. [24] prove that if $\left(I + \frac{1}{\lambda}\nabla^2\mathcal{L}_S\big(\phi^\star(\theta)\big)\right)$ is invertible, the derivative $d\phi^\star(\theta)$ is given by

$$d\phi^\star(\theta) = \left(I + \frac{1}{\lambda}\nabla^2\mathcal{L}_S\big(\phi^\star(\theta)\big)\right)^{-1}. \tag{3.42}$$

By approximating $d\phi^k(\theta) \approx d\phi^\star(\theta)$, the computation of the meta-gradient in Eq. 3.37 – 3.40 can be significantly simplified in terms of second-order derivatives. However, note that inverting the matrix in Eq. 3.42 may be computationally prohibitive for large deep neural networks [24].

3.4.3 Reptile

Reptile [22] is a simple first-order meta-learning algorithm. Similar to MAML [3], Reptile learns an initialization for the parameters θ of a neural network f_θ such that a small number of gradient descent steps on a new task will produce good task-specific parameters, enabling quick adaptation to a new task. The inner-level algorithm in Eq. 3.17 corresponds to k gradient descent steps on the support set,

$$\mathcal{A}lg(\theta, S_i) := \phi_i^k(\theta) = \theta - \alpha\sum_{j=0}^{k-1}\nabla\mathcal{L}_{S_i}\big(\phi_i^j(\theta)\big). \tag{3.43}$$

In contrast to MAML, Reptile does not differentiate through the adaptation procedure $\mathcal{A}lg$, thereby avoiding the computation of second-order derivatives. Instead, the outer-level update in Eq. 3.16 is given by

$$\theta^\star := \theta + \frac{\beta}{N}\sum_{i=1}^{N}(\phi_i^k(\theta) - \theta). \tag{3.44}$$

The key idea is to update the parameters θ in the direction of $(\phi_i^k(\theta) - \theta)$, effectively adjusting the parameters θ towards the task-specific parameters $\phi_i^k(\theta)$ [22]. Note that

Reptile does not assess the adaptation performance of the task-specific parameters using a separate query set. Instead, it updates the meta-parameters exclusively based on the support set in meta-training.

4 Domain-Adaptation via Meta-Learning

The success of machine learning has been driven by the wealth and diversity of available datasets. This is evident in computer vision, where datasets like ImageNet [26] have improved the performance of image classification models. However, in many real-world applications, acquiring a large labeled dataset is often impractical or impossible. Moreover, training data is often distributed across multiple sources due to privacy concerns, limited computational resources, or the natural distribution of data across different production environments. A significant challenge in such settings is the presence of distribution shifts [17, 14]. Meta-learning algorithms are specifically designed for few-shot learning settings, where data is scarce or expensive to obtain. By interpreting the domains as a collection of related tasks, meta-learning algorithms can be effectively applied to DG and DA tasks [12, 21, 20].

However, a notable shortcoming of existing meta-learning algorithms is that the adaptation procedure still requires access to examples of every class, either for ranking similarity between a new example and each example in the support set [16, 32, 29] or for fine-tuning classifiers using the support set [25, 3, 24, 22]. This dependency limits the applicability of current meta-learning methods, preventing them from fully addressing the challenges posed by DA in scenarios where access to every class is unavailable during the adaptation phase, as discussed in Section 2.1.

In many real-world applications, the adaptation data in new domains is constrained not only by limited volume but also by the scarcity of certain classes. For illustration, machine learning models in industry are typically pre-trained in laboratory settings before being transferred to real-world deployments. While normal operational data is often readily available for model adaptation, acquiring anomalous data in large and complex industrial systems is both expensive and time-consuming. This raises the question of whether and how the information present in normal data can be leveraged for adaptation to a new domain. Effectively utilizing the structure and patterns in normal data could potentially facilitate the model's ability to quickly adapt to new environments in real-world deployments.

Figure 4.1: Illustration of three Rainbow-MNIST domains [4].

This problem setting is not limited to anomaly classification tasks in industrial systems but can be extended to an arbitrary multi-class classification problem that allows for in-

ferring domain-specific information from one class to the other classes. In other words, the source and target domains have to share a common structure, enabling one class to encode sufficient information about the corresponding domain to make inferences about the other classes within that domain. I introduce this problem as *One-Class Domain Adaptation* (OC-DA). For example, the Rainbow-MNIST handwritten digit classification problem [4] involves 56 domains, each defined by a unique combination of background color, degree of rotation, and scale size. Fig. 4.1 illustrates different domains, where one class encodes information regarding background color, degree of rotation, and scale size, thereby providing sufficient domain-specific information for the model's adaptation to the new domain.

4.1 Problem Statement

In the One-Class Domain Adaptation (OC-DA) setting, data is distributed across multiple domains. The goal of OC-DA is to learn a model that is able to quickly adapt to a new domain using a few examples from only *one* class. I will refer to this class as the normal class n. The model is a multi-class classifier, capable of differentiating between normal class examples and examples of multiple other classes using only a few normal class examples. I take a meta-learning approach to tackle the challenge of OC-DA. While recent meta-learning algorithms are designed to effectively address generalization across *diverse* learning tasks [16, 29, 25, 3], i.e., learning tasks with varying label spaces, OC-DA focuses on generalization from normal class examples to examples of multiple other classes of a *single* learning task across diverse domains. To the best of my knowledge, my work is the first to address one-class adaptation across domains.

My approach is related to previous work by Li et al. [21] that proposes a meta-learning method for domain generalization (MLDG). Rather than explicitly training for rapid adaptation, MLDG explicitly trains a model for good generalization ability to new domains. Therefore, MLDG simulates domain shifts by splitting the source domains into training and virtual testing domains. The key idea is that gradient descent steps on training domains should also improve performance on virtual testing domains.

The most related work is the task sampling strategy in few-shot one-class classification (FSOC) by Frikha et al. [5], studying the intersection of few-shot learning and one-class classification. One-class classification refers to learning a binary classifier that can differentiate between in-class and out-of-class examples using only in-class data [5]. The goal of FSOC is to learn a binary classifier such that fine-tuning on a few in-class examples achieves the same performance as doing so with a few in-class and out-of-class examples. The class-imbalance rate indicates the ratio of in-class and out-of-class examples. FSOC modifies the task sampling strategy in bi-level meta-learning algorithms such that the class-imbalance rate in the support set of meta-training tasks matches the one in the support set of meta-testing tasks, while the query set of meta-training and testing tasks is class-balanced.

Rather than training a binary classifier, I am interested in learning a multi-class classifier

that is able to differentiate between normal class examples and examples of multiple other classes using only a few normal class examples. I propose a task sampling strategy to adapt any bi-level meta-learning algorithm to OC-DA, thereby extending the use of meta-learning to one-class adaptation across domains.

Consider a collection of datasets $(D_i)_{i\in\mathcal{E}}$ acquired in multiple domains $\mathcal{E}$. Each dataset $D_i := (\mathcal{X}_i, \mathcal{Y}_i)$ contains pairs of feature vectors $\mathcal{X}_i \subset \mathcal{X}$ and labels $\mathcal{Y}_i \subset \mathcal{Y}$. The domain set $\mathcal{E}$ is split into a collection of source and target domains $\mathcal{E}^{\text{source}} \cap \mathcal{E}^{\text{target}} = \emptyset$. The goal is to learn a robust prediction model f_θ on the source domains such that it can quickly adapt to an unseen target domain using a few normal class examples. This training objective can be formulated as

$$\theta^\star := \arg\max_\theta \mathop{\mathbb{E}}_{i\sim\mathcal{E}} \mathop{\mathbb{E}}_{\substack{\mathcal{T}_i\sim D_i \\ \mathcal{T}_i=(S_i,Q_i)}} \sum_{(x,y)\in Q_i} \log p(y|x, S_i), \tag{4.1}$$

where $S_i \subset \mathcal{X}_i \times \{n\}$ is a subset of a few normal class examples in D_i.

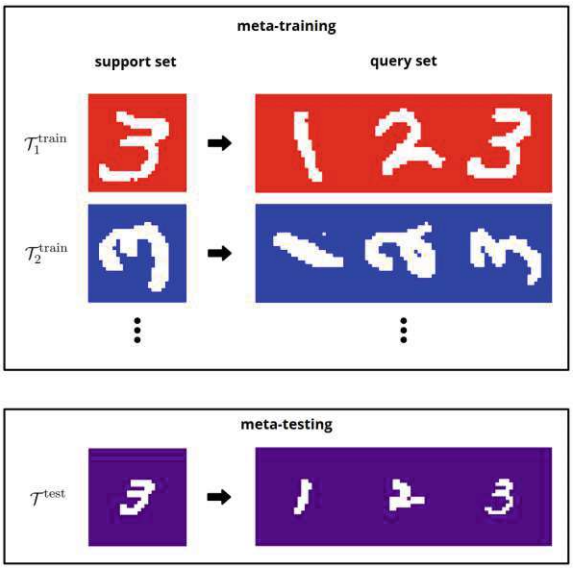

Figure 4.2: Example of a 3-way 1-shot learning task in the OC-DA setting.

The bi-level optimization framework of optimization-based meta-learning, as detailed in Section 3.4, enables direct optimization for one-class adaptation across domains. Meta-training tasks are sampled from the source domains, while meta-testing tasks are sampled from the target domains. In N-way K-shot learning, classical meta-learning algorithms typically sample tasks where both the support and query sets consist of class-balanced datasets, each containing NK examples, with K examples per class. I aim to train a model on the source domains in the same manner it will be utilized in the target domains. Therefore, I propose a task sampling strategy that aligns the task setup during meta-training with that of meta-testing. Thus, the support set is restricted to normal class examples, while the query set remains class-balanced in meta-training tasks. This strategy

requires the domain-specific parameters obtained by performing the adaptation procedure $\mathcal{A}lg$ using a few normal class examples to improve performance on a class-balanced dataset within the domain.

Fig. 4.2 illustrates 3-way 1-shot learning in the OC-DA setting on the Rainbow-MNIST dataset. During meta-training, the model learns to adapt to a new domain utilizing only one normal class example ($n = 3$). At meta-testing time, the model is given one normal class example from a previously unseen domain, and its performance is evaluated on a class-balanced dataset within the domain.

OC-DA allows addressing distribution shift challenges in industrial machine learning applications, where only normal data is available for model adaptation in new environments. Fig. 4.3 illustrates the training of a machine learning model in the OC-DA setting for an industrial application. The model is trained in the laboratory setting in the same way it will be utilized in real-world systems, learning to leverage domain-specific information present in normal operational data (shown in green) for adaptation to new domains. At deployment time, the model can effectively leverage the available information to maintain robust performance when encountering failure types (shown in red) in the system.

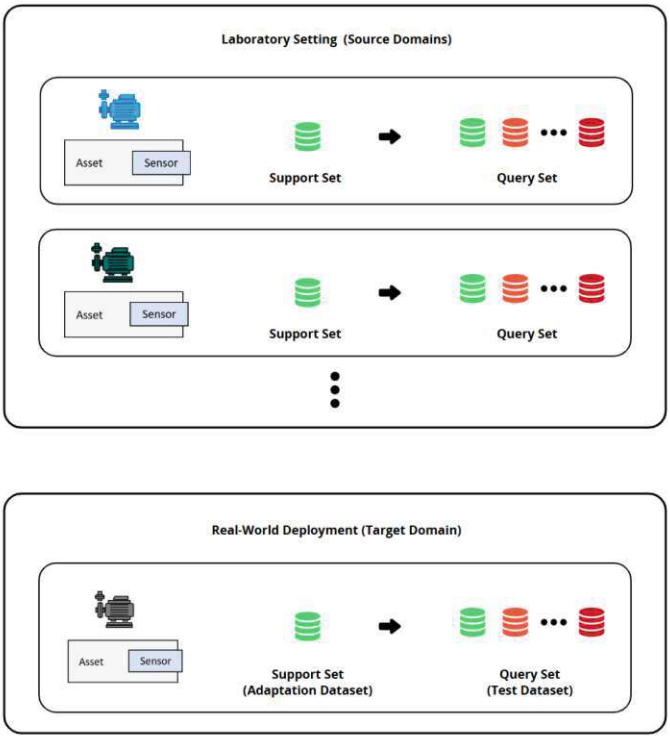

Figure 4.3: Illustration of OC-DA in industrial machine learning.

4.2 One-Class Domain Adaptation via Model-Agnostic Meta-Learning

I apply my proposed task sampling strategy to the MAML algorithm [3], using the notation introduced in Section 3.4.2. The OC-DA MAML algorithm for the meta-training process is described in Algorithm 1. The neural network f_θ is randomly initialized. In each iteration step, a random batch of domains $I \subset \mathcal{E}$ is selected. For each domain $i \in I$, a N-way K-shot learning task is sampled according to our task sampling strategy: the support set S_i contains K normal class examples, while the query set Q_i is a class-balanced dataset with K examples per class. Then, the domain-specific parameters $\phi_i(\theta)$ are computed by performing k gradient descent steps on the support set. Finally, the meta-parameters θ are updated via a gradient descent step on the query set. Note that this includes differentiating through the fine-tuning process on the support set. By this, the meta-parameters θ are explicitly optimized such that performing k gradient descent steps on a few normal class examples yields effective domain-specific parameters $\phi_i^k(\theta)$.

Algorithm 1: MAML for One-Class Domain Adaptation (OC-DA MAML)

Required: model f_θ parameterized by θ, loss function $\mathcal{L}$, inner learning rate α, outer learning rate β, set of class labels C, normal class $n \in C$, number of inner gradient descent steps k

Randomly initialize θ_0

while *not done* **do**

  Sample batch of domains $I \subset \mathcal{E}$

  for $i \in I$ **do**

    Sample K examples $S_i := \{(x_j, n)\}_{j=1}^K$ from D_i

    for $c \in C$ **do**

      Sample K examples $Q_i^c := \{(x_j, c)\}_{j=1}^K$ from D_i

    end

    $Q_i := \bigcup_{c \in C} Q_i^c$

    Compute domain-specific parameters via k gradient descent steps:
    $\phi_i^k(\theta_t) := \theta_t - \alpha \sum_{j=0}^{k-1} \nabla \mathcal{L}_{S_i}\left(\phi_i^j(\theta_t)\right)$

  end

  Perform meta-update via gradient descent:
  $\theta_{t+1} := \theta_t - \frac{\beta}{|I|} \sum_{i \in I} \nabla\left(\mathcal{L}_{Q_i} \circ \phi_i^k\right)(\theta_t)$

end

Through iterative application of the chain rule, the derivative of ϕ^k is given by

$$d\phi^k(\theta) = d(\phi \circ \cdots \circ \phi)(\theta) \tag{4.2}$$

$$= d\phi\big(\phi^{k-1}(\theta)\big)\; d\phi\big(\phi^{k-2}(\theta)\big)\; \ldots\; d\phi(\theta) \tag{4.3}$$

$$= \Big(I - \alpha\, \nabla^2 \mathcal{L}_S\big(\phi^{k-1}(\theta)\big)^\top\Big)\Big(I - \alpha\, \nabla^2 \mathcal{L}_S\big(\phi^{k-2}(\theta)\big)^\top\Big)\; \ldots\; \Big(I - \alpha\, \nabla^2 \mathcal{L}_S\big(\theta\big)^\top\Big) \tag{4.4}$$

$$= \prod_{j=0}^{k-1} \Big(I - \alpha\, \nabla^2 \mathcal{L}_S\big(\phi^j(\theta)\big)^\top\Big). \tag{4.5}$$

Then, the meta-gradient $\nabla(\mathcal{L}_Q \circ \phi^k)(\theta)$ can be written as

$$\nabla(\mathcal{L}_Q \circ \phi^k)(\theta) = d(\mathcal{L}_Q \circ \phi^k)(\theta)^\top \tag{4.6}$$

$$= \Big(d\mathcal{L}_Q\big(\phi^k(\theta)\big)\; d\phi^k(\theta)\Big)^\top \tag{4.7}$$

$$= d\phi^k(\theta)^\top\; d\mathcal{L}_Q\big(\phi^k(\theta)\big)^\top \tag{4.8}$$

$$= \prod_{j=0}^{k-1} \Big(I - \alpha\, \nabla^2 \mathcal{L}_S\big(\phi^j(\theta)\big)^\top\Big)^\top \nabla \mathcal{L}_Q\big(\phi^k(\theta)\big) \tag{4.9}$$

$$= \prod_{j=0}^{k-1} \Big(I - \alpha\, \nabla^2 \mathcal{L}_S\big(\phi^j(\theta)\big)\Big) \nabla \mathcal{L}_Q\big(\phi^k(\theta)\big). \tag{4.10}$$

Since the computation of the meta-gradient performs a gradient descent step with respect to the meta-parameters θ, the meta-update involves the computation of second-order derivatives,

$$\theta_{t+1} = \theta_t - \frac{\beta}{|I|} \sum_{i \in I} \nabla\big(\mathcal{L}_{Q_i} \circ \phi_i^k\big)(\theta_t) \tag{4.11}$$

$$= \theta_t - \frac{\beta}{|I|} \sum_{i \in I} \prod_{j=0}^{k-1} \Big(I - \alpha \nabla^2 \mathcal{L}_{S_i}\big(\phi_i^j(\theta_t)\big)\Big) \nabla \mathcal{L}_{Q_i}\big(\phi_i^k(\theta_t)\big). \tag{4.12}$$

4.3 Analysis of OC-DA MAML

In the following section, I present an analysis to provide a deeper understanding of my proposed task sampling strategy and its underlying motivation. Specifically, I will use a Taylor series expansion to approximate the gradient of the OC-DA MAML algorithm, $g(\theta) := \nabla(\mathcal{L}_Q \circ \phi^k)(\theta)$ [22]. This analysis will focus on two key components: the first term minimizes the loss on the query set, while the second term maximizes the inner product between the gradients of the support and query loss [22]. For simplicity, I will initially consider a single inner gradient descent step ($k = 1$) before generalizing to multiple steps.

Using the second-order Taylor series expansion of $\nabla \mathcal{L}_Q$ around θ, for small $\alpha > 0$, the OC-DA MAML gradient can be approximated as [22]

$$
\begin{aligned}
g(\theta) &= \nabla(\mathcal{L}_Q \circ \phi)(\theta) \\
&= \big(I - \alpha\nabla^2\mathcal{L}_S(\theta)\big)\, \nabla\mathcal{L}_Q\big(\phi(\theta)\big) \\
&= \big(I - \alpha\nabla^2\mathcal{L}_S(\theta)\big)\, \Big(\nabla\mathcal{L}_Q(\theta) + \nabla^2\mathcal{L}_Q(\theta)\, \underbrace{\big(\phi(\theta) - \theta\big)}_{=-\alpha\nabla\mathcal{L}_S(\theta)} + \underbrace{\mathcal{O}\big(||\phi(\theta) - \theta||^2\big)}_{=\mathcal{O}(\alpha^2)}\Big) \\
&= \big(I - \alpha\nabla^2\mathcal{L}_S(\theta)\big)\, \big(\nabla\mathcal{L}_Q(\theta) - \alpha\nabla^2\mathcal{L}_Q(\theta)\nabla\mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2)\big) \\
&= \nabla\mathcal{L}_Q(\theta) - \alpha\nabla^2\mathcal{L}_Q(\theta)\nabla\mathcal{L}_S(\theta) - \alpha\nabla^2\mathcal{L}_S(\theta)\nabla\mathcal{L}_Q(\theta) + \mathcal{O}(\alpha^2) \\
&= \nabla\mathcal{L}_Q(\theta) - \alpha\Big(\nabla^2\mathcal{L}_Q(\theta)\nabla\mathcal{L}_S(\theta) + \nabla^2\mathcal{L}_S(\theta)\nabla\mathcal{L}_Q(\theta)\Big) + \mathcal{O}(\alpha^2) \\
&= \nabla\mathcal{L}_Q(\theta) - \alpha\, \nabla\big(\nabla\mathcal{L}_Q \cdot \nabla\mathcal{L}_S\big)(\theta) + \mathcal{O}(\alpha^2) \\
&= \nabla\big(\mathcal{L}_Q - \alpha\nabla\mathcal{L}_Q \cdot \nabla\mathcal{L}_S\big)(\theta) + \mathcal{O}(\alpha^2).
\end{aligned}
$$

Then, the meta-update can be written as

$$
\theta_{t+1} = \theta_t - \frac{\beta}{|I|}\sum_{i\in I}\nabla\big(\mathcal{L}_{Q_i} \circ \phi_i^k\big)(\theta_t) \tag{4.13}
$$

$$
= \theta_t - \frac{\beta}{|I|}\sum_{i\in I}\nabla\Big(\mathcal{L}_{Q_i} - \alpha\nabla\mathcal{L}_{Q_i} \cdot \nabla\mathcal{L}_{S_i}\Big)(\theta_t) + \mathcal{O}(\alpha^2). \tag{4.14}
$$

While the first term minimizes the loss on the query set, the second term maximizes the inner product between the gradients of the support and query loss [22]. When this inner product is positive, performing a gradient descent step on one dataset improves performance on the other [22]. The MAML algorithm maximizes this inner product, thereby finding update directions that minimize both the support and query loss [21]. Consequently, a gradient descent step on the support set improves performance on the query set, and thus, enables within-task generalization [22]. In the context of OCDA-MAML, this implies that gradient descent steps on normal class examples not only improve performance on normal data but also improve performance on data including other classes. At meta-testing time, when the model only has access to normal class examples, fine-tuning on a few normal class examples will produce good domain-specific parameters while avoiding over-fitting.

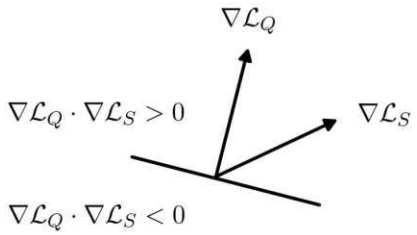

Figure 4.4: MAML maximizes the inner product between the gradients $\nabla\mathcal{L}_Q$ and $\nabla\mathcal{L}_S$.

It is straightforward to extend this to k inner gradient descent steps. For sufficiently small

αk, the first-order Taylor series expansion of $\nabla\mathcal{L}_S$ around θ yields

$$\nabla\mathcal{L}_S\big(\phi^k(\theta)\big) = \nabla\mathcal{L}_S(\theta) + \mathcal{O}\big(\| \underbrace{\phi^k(\theta) - \theta}_{=-\alpha\sum_{j=0}^{k-1}\nabla\mathcal{L}_S\left(\phi^j(\theta)\right)} \|\big) \tag{4.15}$$

$$= \nabla\mathcal{L}_S(\theta) + \mathcal{O}(\alpha k). \tag{4.16}$$

Analogously, I have

$$\nabla^2\mathcal{L}_S\big(\phi^k(\theta)\big) = \nabla^2\mathcal{L}_S(\theta) + \mathcal{O}(\alpha k). \tag{4.17}$$

For sufficiently small αk, the gradient $\nabla\mathcal{L}_Q\big(\phi^k(\theta)\big)$ can be approximated by the second-order Taylor expansion as [22]

$$\begin{aligned}
\nabla\mathcal{L}_Q\big(\phi^k(\theta)\big) &= \nabla\mathcal{L}_Q(\theta) + \nabla^2\mathcal{L}_Q(\theta)\big(\phi^k(\theta) - \theta\big) + \mathcal{O}\big(\|\phi^k(\theta) - \theta\|^2\big) \\
&= \nabla\mathcal{L}_Q(\theta) + \nabla^2\mathcal{L}_Q(\theta)\Big(-\alpha\sum_{j=0}^{k-1}\nabla\mathcal{L}_S\big(\phi^j(\theta)\big)\Big) + \mathcal{O}(\alpha^2k^2) \\
&= \nabla\mathcal{L}_Q(\theta) - \alpha\nabla^2\mathcal{L}_Q(\theta)\sum_{j=0}^{k-1}\nabla\mathcal{L}_S\big(\phi^j(\theta)\big) + \mathcal{O}(\alpha^2k^2) \\
&= \nabla\mathcal{L}_Q(\theta) - \alpha\nabla^2\mathcal{L}_Q(\theta)\sum_{j=0}^{k-1}\big(\nabla\mathcal{L}_S(\theta) + \mathcal{O}(\alpha j)\big) + \mathcal{O}(\alpha^2k^2) \\
&= \nabla\mathcal{L}_Q(\theta) - \alpha k\nabla^2\mathcal{L}_Q(\theta)\nabla\mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2k^2) + \mathcal{O}(\alpha^2k^2) \\
&= \nabla\mathcal{L}_Q(\theta) - \alpha k\nabla^2\mathcal{L}_Q(\theta)\nabla\mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2k^2),
\end{aligned} \tag{4.18}$$

where I used the first-order Taylor series expansion of $\nabla\mathcal{L}_S\big(\phi^j(\theta)\big)$ from Eq. 4.16 in Eq. 4.18. Therefore, the OC-DA MAML gradient can be approximated as [22]

$$
\begin{aligned}
g(\theta) &= \nabla(\mathcal{L}_Q \circ \phi^k)(\theta) \\
&= \prod_{j=0}^{k-1} \Big(I - \alpha\, \nabla^2 \mathcal{L}_S\big(\phi^j(\theta)\big)\Big) \nabla \mathcal{L}_Q\big(\phi^k(\theta)\big) \\
&= \Big(I - \alpha \sum_{j=0}^{k-1} \nabla^2 \mathcal{L}_S\big(\phi^j(\theta)\big) + \mathcal{O}(\alpha^2)\Big)\, \nabla \mathcal{L}_Q\big(\phi^k(\theta)\big) \\
&= \Big(I - \alpha \sum_{j=0}^{k-1} \nabla^2 \mathcal{L}_S\big(\phi^j(\theta)\big) + \mathcal{O}(\alpha^2)\Big)\, \big(\nabla \mathcal{L}_Q(\theta) - \alpha k \nabla^2 \mathcal{L}_Q(\theta) \nabla \mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2 k^2)\big) \\
&= \Big(I - \alpha \sum_{j=0}^{k-1} \big(\nabla^2 \mathcal{L}_S(\theta) + \mathcal{O}(\alpha j)\big) + \mathcal{O}(\alpha^2)\Big)\, \big(\nabla \mathcal{L}_Q(\theta) - \alpha k \nabla^2 \mathcal{L}_Q(\theta) \nabla \mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2 k^2)\big) \\
&= \big(I - \alpha k \nabla^2 \mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2 k^2) + \mathcal{O}(\alpha^2)\big)\, \big(\nabla \mathcal{L}_Q(\theta) - \alpha k \nabla^2 \mathcal{L}_Q(\theta) \nabla \mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2 k^2)\big) \\
&= \big(I - \alpha k \nabla^2 \mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2 k^2)\big)\, \big(\nabla \mathcal{L}_Q(\theta) - \alpha k \nabla^2 \mathcal{L}_Q(\theta) \nabla \mathcal{L}_S(\theta) + \mathcal{O}(\alpha^2 k^2)\big) \\
&= \big(I - \alpha k \nabla^2 \mathcal{L}_S(\theta)\big)\, \big(\nabla \mathcal{L}_Q(\theta) - \alpha k \nabla^2 \mathcal{L}_Q(\theta) \nabla \mathcal{L}_S(\theta)\big) + \mathcal{O}(\alpha^2 k^2) \\
&= \nabla \mathcal{L}_Q(\theta) - \alpha k \nabla^2 \mathcal{L}_Q(\theta) \nabla \mathcal{L}_S(\theta) - \alpha k \nabla^2 \mathcal{L}_S(\theta) \nabla \mathcal{L}_Q(\theta) + \mathcal{O}(\alpha^2 k^2) \\
&= \nabla \mathcal{L}_Q(\theta) - \alpha k \nabla\big(\nabla \mathcal{L}_Q \cdot \nabla \mathcal{L}_S\big)(\theta) + \mathcal{O}(\alpha^2 k^2) \\
&= \nabla\big(\mathcal{L}_Q - \alpha k \nabla \mathcal{L}_Q \cdot \nabla \mathcal{L}_S\big)(\theta) + \mathcal{O}(\alpha^2 k^2).
\end{aligned}
$$

Again, the first term minimizes the loss on the query set and the second term maximizes the inner product between the gradients of the support and query loss [22].

5 Experiments

In the next chapter, I present the experiments on distribution shifts and one-class domain adaptation. I implemented both standard learning and meta-learning methods. In the context of standard learning, I illustrate the performance gap between source and target domains. For meta-learning, I implemented the MAML algorithm and the OC-DA MAML algorithm in the OC-DA setting. I aim to show that the MAML algorithm is not tailored to OC-DA settings and demonstrate the effectiveness of my task sampling strategy in OC-DA MAML, enabling rapid one-class adaptation. To ensure the reliability of my results, all results were averaged over three runs with different seeds. In the implementation of the MAML and OC-DA MAML algorithm, I used the 'learn2learn' library [2], a library for meta-learning research that provides low-level routines built on top of PyTorch for few-shot learning and differentiable optimization (e.g., automatic differentiation through the meta-updates in MAML).

I empirically evaluate my approach on two datasets. First, I utilize a well-established meta-learning benchmark, the Rainbow-MNIST dataset [4] and adapt it to the OC-DA setting. Second, I assess my approach using a real-world dataset of vibration-based sensor readings recorded by four centrifugal pumps across multiple domains (Centrifugal-Pumps dataset), demonstrating its robustness and applicability in real-world applications. The goal of my experimental evaluation is to answer the following questions:

(1) Is it possible to leverage domain-specific information present in one class for efficient domain adaptation?

(2) Can OC-DA MAML enable rapid one-class adaptation in the context of domain adaptation tasks?

(3) How does OC-DA MAML compare to the original MAML algorithm in the OC-DA setting?

5.1 Data

For the Rainbow-MNIST dataset, I followed the approach of Finn et al. [4] and split the MNIST dataset into 56 class-balanced sub-datasets, each containing 1000 examples, and applied the corresponding domain transformation to each sub-dataset. Support and query tasks are sampled randomly from these domain datasets. To adapt the meta-learning benchmark to the OC-DA setting, I designate one class in the original dataset as the normal class. I split the domains into 40/8/8 domains for meta-training/validation/testing

and list their corresponding combinations of domain transformations as follows:

Source domains:

Meta-training: (‘orange’, 0°, ‘full’), (‘orange’, 0°, ‘half’), (‘orange’, 90°, ‘full’), (‘orange’, 90°, ‘half’), (‘orange’, 180°, ‘full’), (‘orange’, 180°, ‘half’), (‘orange’, 270°, ‘full’), (‘orange’, 270°, ‘half’), (‘yellow’, 0°, ‘full’), (‘yellow’, 0°, ‘half’), (‘yellow’, 90°, ‘full’), (‘yellow’, 90°, ‘half’), (‘yellow’, 180°, ‘full’), (‘yellow’, 180°, ‘half’), (‘yellow’, 270°, ‘full’), (‘yellow’, 270°, ‘half’), (‘blue’, 0°, ‘full’), (‘blue’, 0°, ‘half’), (‘blue’, 90°, ‘full’), (‘blue’, 90°, ‘half’), (‘blue’, 180°, ‘full’), (‘blue’, 180°, ‘half’), (‘blue’, 270°, ‘full’), (‘blue’, 270°, ‘half’), (‘indigo’, 0°, ‘full’), (‘indigo’, 0°, ‘half’), (‘indigo’, 90°, ‘full’), (‘indigo’, 90°, ‘half’), (‘indigo’, 180°, ‘full’), (‘red’, 180°, ‘half’), (‘indigo’, 270°, ‘full’), (‘indigo’, 270°, ‘half’), (‘violet’, 0°, ‘full’), (‘violet’, 0°, ‘half’), (‘violet’, 90°, ‘full’), (‘violet’, 90°, ‘half’), (‘violet’, 180°, ‘full’), (‘violet’, 180°, ‘half’), (‘violet’, 270°, ‘full’), (‘violet’, 270°, ‘half’)

Meta-validation: (‘red’, 0°, ‘full’), (‘red’, 0°, ‘half’), (‘red’, 90°, ‘full’), (‘red’, 90°, ‘half’), (‘red’, 180°, ‘full’), (‘red’, 180°, ‘half’), (‘red’, 270°, ‘full’), (‘red’, 270°, ‘half’)

Target domains:

Meta-testing: (‘green’, 0°, ‘full’), (‘green’, 0°, ‘half’), (‘green’, 90°, ‘full’), (‘green’, 90°, ‘half’), (‘green’, 180°, ‘full’), (‘green’, 180°, ‘half’), (‘green’, 270°, ‘full’), (‘green’, 270°, ‘half’)

The Centrifugal-Pumps dataset comprises sensor readings recorded from four identical centrifugal pumps, each equipped with IoT sensors recording vibration data, in a controlled laboratory environment. Centrifugal pumps are the most widely used industrial assets in a broad range of industrial systems [18]. The continuous use of centrifugal pumps within an industrial system makes it prone to failure, with the risk of material damage or complete system breakdown [14]. Therefore, maintenance work is crucial in the deployment of these pumps to ensure their reliable operation and longevity [30].

The widespread installation of IoT sensor platforms on industrial assets offers significant opportunities for applying machine learning models to predictive maintenance [14]. The goal of predictive maintenance in industry is to assess the condition of industrial assets to predict the optimal timing for maintenance interventions [34]. This strategy offers potential cost savings compared to periodic or time-based maintenance by ensuring that maintenance tasks are carried out only when necessary. Moreover, identifying failure types is crucial for facilitating efficient maintenance services [15]. Typical failure types in centrifugal pumps include blockage, where the liquid flow is obstructed, and cavitation, which occurs when the liquid pressure drops too low [30].

In order to create a diverse dataset with multiple domains, I simulated varying environmental influences by placing the pumps on different surfaces. For each measurement round, a pump was placed either on a concrete surface or within a steel framework and operated under multiple conditions, including normal operational data, idle state data and three anomalous conditions: hydraulic blockage, dry running and cavitation. For hydraulic blockage, the flow of liquid was blocked in the outlet pipe. For dry running, the flow of

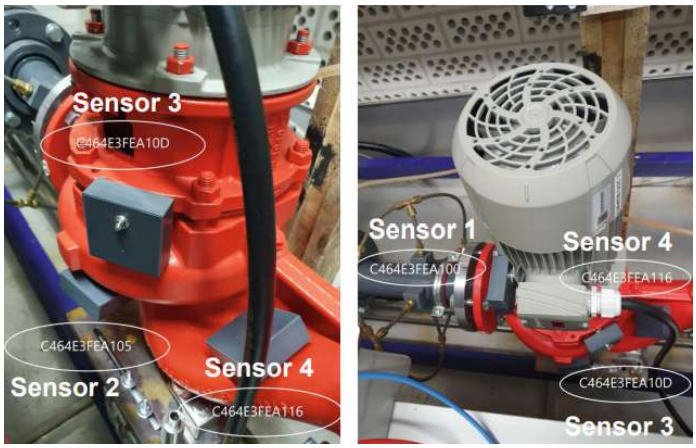

Figure 5.1: Centrifugal pump in laboratory environment.

liquid was blocked in the inlet pipe. Cavitation was simulated by reducing the liquid pressure. The IoT sensors were consistently positioned on the same location of each pump, see Fig. 5.1. Each sensor recorded vibration data at a frequency of 6644 Hz, collecting 512 data points per minute. By the fast Fourier transform (FFT), the raw signal in the time domain is converted to its representation in the frequency domain.

The dataset provides 32 domains: Each domain corresponds to a measurement round involving one of four centrifugal pumps placed on a concrete surface or within a steel framework and equipped with two sensors positioned at different locations of the pump. Each experiment was repeated on another day. Each domain contains approximately 100 normal examples, 60 idle state examples and 25 examples per anomalous class.

Instead of fixing a single source and target domain setup, I investigate multiple combinations of source and target domains. Previous work by Kemnitz et al. [14] demonstrated that transferring data from one pump to another, even within the same environment, presents significant challenges. In real-world deployment, industry partners are typically interested in transferring a model not only to another pump within the same environment but also to another pump within an unknown environment. Therefore, I split the domains into multiple combinations of source and target domains, each combination corresponding to a transfer to a new pump within an unknown environment.

Specifically, I split the domains into 12/4/4 domains for meta-training/validation/testing. The meta-training domains correspond to three pumps operated in the same environment, including the repeated measurement round. The meta-validation domains correspond to the new pump within the known environment. The meta-testing domains correspond to the new pump within the unknown environment. This setup results in a total of eight source and target domain combinations. For illustration, I list one such source and target domain combination as follows:

Source domains:

Meta-training: (P_1, S_1, 'steel'), (P_1, S_1, 'steel', 'repetition'), (P_2, S_1, 'steel'), (P_2, S_1, 'steel', 'repetition'), (P_3, S_1, 'steel'), (P_1, S_1, 'steel', 'repetition'), (P_1, S_2, 'steel'), (P_1, S_2, 'steel', 'repetition'), (P_2, S_2, 'steel'), (P_2, S_2, 'steel', 'repetition'), (P_3, S_2, 'steel'), (P_3, S_2, 'steel', 'repetition')

Meta-validation: (P_4, S_1, 'steel'), (P_4, S_1, 'steel', 'repetition'), (P_4, S_2, 'steel'), (P_4, S_2, 'steel', 'repetition')

Target domains:

Meta-testing: (P_4, S_1, 'concrete'), (P_4, S_1, 'concrete', 'repetition'), (P_4, S_2, 'concrete'), (P_4, S_2, 'concrete', 'repetition')

5.2 Experimental Setup

In this section, I detail my experimental setup. I first introduce the metrics used to measure the performance gap resulting from distribution shifts between source and target domains, and explain their significance for my evaluation. I then outline my proposed task sampling strategy for bi-level meta-learning algorithms in OC-DA settings. I contrast this strategy with the task sampling approach used in classical bi-level meta-learning algorithms, with a particular focus on comparing it to the task setup employed by the MAML algorithm. Finally, I provide the model architecture for the Rainbow-MNIST and the Centrifugal-Pumps dataset.

In the context of distribution shifts, accurately measuring the performance drop caused by these shifts is challenging. The machine learning literature distinguishes between in-distribution (ID) and out-of-distribution (OOD) performance [17]. ID performance is measured by training a model on the training distribution and evaluating its performance on held-out data from the training distribution [17]. This metric provides insight into how well the model generalizes on the training distribution. In contrast, ODD performance is measured by the model's performance on data from the test distribution [17]. This metric is crucial for assessing the model's robustness and provides insight into how well the model generalizes to data outside its training distribution.

However, the comparison of the ID and OOD metrics alone might not effectively isolate the impact of the distribution shift [17]. Since the ID metric is evaluated on the training distribution and the OOD metric on the test distribution, the observed performance gap may be influenced by factors beyond the distribution shift [17]. The test distribution might present greater challenges for model training; for example, classification could be more difficult in the test domain due to domain-specific factors [17]. For illustration, consider an IoT sensor-based dataset. In real-world deployments, industrial assets are often integrated into large and complex industrial systems. Unlike training data obtained in controlled laboratory settings, the real-world environment might introduce additional noise to the recorded sensor signals, thereby posing greater challenges for model training. Consequently, the gap between ID and OOD performance might be larger than the actual distribution

shift gap. In addition to the ID and OOD metrics, it is therefore crucial to measure the model's performance by training and evaluating it directly on the test distribution (ID-test metric) [17]. The ID-test performance is measured by training a model on the test distribution and evaluating its performance on held-out data from the test distribution [17]. This metric provides a performance baseline in the test domain. However, a major challenge of the ID-test metric is that the test domain often lacks sufficient data to train a robust model [17].

To demonstrate the performance gap between the source and target domains, I train a model on the source domains and evaluate its performance both on held-out data from each source domain (ID performance) and on the target domains (OOD performance). I then evaluate the effectiveness of my adapted OC-DA MAML variant within the OC-DA setting and compare it to the original MAML algorithm. The ID and ODD metrics not only provide insights into the distribution shift gap but also serve as baselines for my OC-DA MAML algorithm.

The ODD metric can be viewed as a lower bound for performance, indicating that model adaptation on target domain data should outperform standard learning on the source domains and direct model transfer to the target domains. Additionally, the ID metric serves as a reference for predicting performance on the target domains, establishing a baseline expectation for how well the model should perform given similar training conditions to the source domains. The ID-test metric also provides a baseline expectation for performance in the target domains. However, it is important to consider the data and computational resources required in the target domains. Although training a model directly on the target domains may produce better results, it demands considerable amounts of target domain data and significant computational resources. For example, the domains in the Centrifugal-Pumps dataset are severely data-constrained, providing only 25 examples per anomalous class. Furthermore, it is not feasible to train a new model for each new centrifugal pump within a new environment.

		Shots per class c (support)		Shots per class c (query)
		$c = n$	$c \in C \setminus \{n\}$	$c \in C$
MAML	Meta-training	K	K	K
	Meta-validation	K	K	K
	Meta-testing	K	0	$\min_{c \in C} N_c^i$
OC-DA MAML	Meta-training	K	0	K
	Meta-validation	K	0	K
	Meta-testing	K	0	$\min_{c \in C} N_c^i$

Table 5.1: Task sampling strategy of MAML vs. OC-DA MAML: Meta-testing tasks are sampled in terms of the OC-DA setting, where the support set contains K normal examples and the query set is class-balanced. In MAML, meta-training and meta-validation tasks are K-shot learning tasks. In OC-DA MAML, the support set only contains K normal examples. N_c^i denotes the number of examples in D_i that belong to class c.

I divide the source domains into training and validation domains. During meta-training, meta-training tasks are sampled from the training domains, meta-validation tasks are sampled from the validation domains. Meta-training is stopped either by reaching the maximal number of iterations or through early stopping. In MAML, the meta-training and validation tasks are K-shot learning tasks, including K examples per class for both the support and query sets. In the OC-DA MAML sampling strategy, only K examples of the normal class are sampled for the support set, see Table 5.1. In each target domain $i \in \mathcal{E}^{\text{target}}$, the model is adapted on K normal examples (meta-testing support set) and evaluated on a class-balanced dataset (meta-testing query set).

In Rainbow-MNIST, the domains are class-balanced by design. In Centrifugal-Pumps, I down-sampled the target domain datasets D_i by reducing the number of examples per class to match the size of the minority class. Here, N_c^i denotes the number of examples in D_i that belong to class c, $N_c^i := |\{j|(x_j, y_j) \in D_i, y_j = c\}|$. Note that the MAML algorithm is designed for classical meta-learning settings, where the task setup typically differs from the task setup in the OC-DA setting, as outlined in Table 5.1. In classical meta-learning settings, the support and query sets of meta-training, validation, and testing tasks are K-shot learning tasks. In contrast, in the OC-DA setting, the support set of meta-testing tasks is limited to K shots of the normal class, as detailed in Table 5.1. Furthermore, evaluating the model's performance using only K-shots per class in the target domain is impractical for domain adaptation. Therefore, I assess the model's performance on a down-sampled dataset, rather than a dataset with K-shots per class.

In Rainbow-MNIST, I adopt the approach of Yao et al. [33] by employing a convolutional neural network that consists of four convolutional blocks, followed by a linear layer and softmax activation function. Each convolutional block is designed with a two-dimensional convolutional layer with 32 filters of size 3×3, a batch normalization layer, a ReLU activation function, and a two-dimensional max-pooling layer of size 2×2. In the Centrifugal-Pumps dataset, I use the same model architecture as Frikha et al. [5], that is, a convolutional neural network with three convolutional blocks, a linear layer and a softmax activation function. Each block consists of a one-dimensional convolutional layer with 32 filters of size 5, a one-dimensional max-pooling layer of size 2, and a ReLU activation function.

5.3 Results

Fig. 5.2 illustrates differences in the data distribution across two different domains in the Centrifugal-Pumps dataset. The plots show the average amplitude of FFT vibration signals for each class (1-normal, 5-cavitation, 6-hydraulic blockage, 7-dry running). The data was recorded by the same pump placed on two different surfaces: within a steel framework and on a concrete surface. There are significant differences in the data distribution. Empirical experiments will later confirm that these visual differences in the data distribution present challenges for classification models and lead to a substantial performance drop, see Table 5.4.

Fig. 5.3 illustrates domain-specific patterns in the Centrifugal-Pumps dataset. The plots show the 95% confidence interval of the FFT vibration signals in the frequency spectrum. The normal class encodes domain-specific information, in the sense that the vibration signals of the anomalous classes follow the pattern of the normal class. This observation motivates the question whether the information present in normal data can be leveraged for adaptation to another domain.

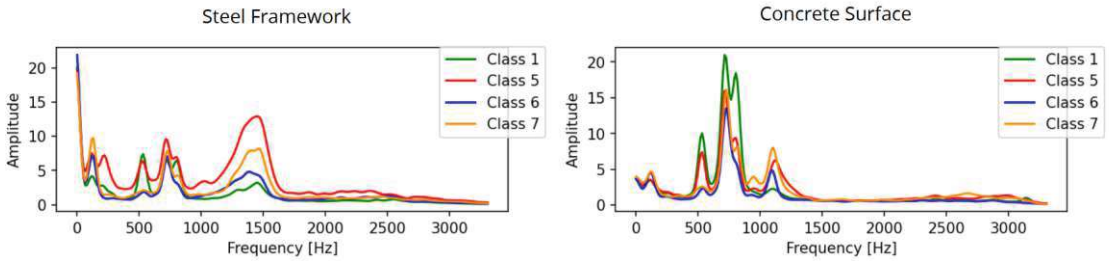

Figure 5.2: Visualization of distribution shifts in the Centrifugal-Pumps dataset, showing the average amplitude [mm] of FFT vibration signals per frequency [Hz] for each class (1-normal, 5-cavitation, 6-hydraulic blockage, 7-dry running), recorded by the same pump operated within a steel framework vs. on a concrete surface.

Metric	Training dataset	Test dataset	Rainbow-MNIST
ID	source	source	83.93
ODD	source	target	13.5
ID-test	target	target	72.77

Table 5.2: Experimental results of standard learning on source and target domains on Rainbow-MNIST. Note that the ID metric is evaluated on held-out data from each source domain, the ID-test metric is evaluated on held-out data from each target domain.

Table 5.2 presents the experimental results on the Rainbow-MNIST dataset, demonstrating the performance gap between source and target domains. In model training, I split the data into training and validation data and applied early stopping. For the ID metric, I collected, shuffled and split the data in the source domains into a training, validation and test dataset. The ID metric is computed as the accuracy [%] on this test dataset. For the ODD metric, I collected, shuffled and split the data in the source domains into a training and validation dataset. The ODD metric is computed as the average accuracy [%] over the class-balanced target domain datasets. For the ID-test metric, I collected, shuffled and split the data in the target domains into a training, validation and test dataset. The ID-test metric is computed as the accuracy [%] on this test dataset. The results show high ID performance and low ODD performance. Additionally, the ID-test performance significantly outperforms the ODD performance, indicating that the target domains are not intrinsically more challenging and the performance gap between source and target domains is indeed due to distribution shifts. Since the corresponding training dataset is substantially larger, the ID performance is higher than the ID-test performance.

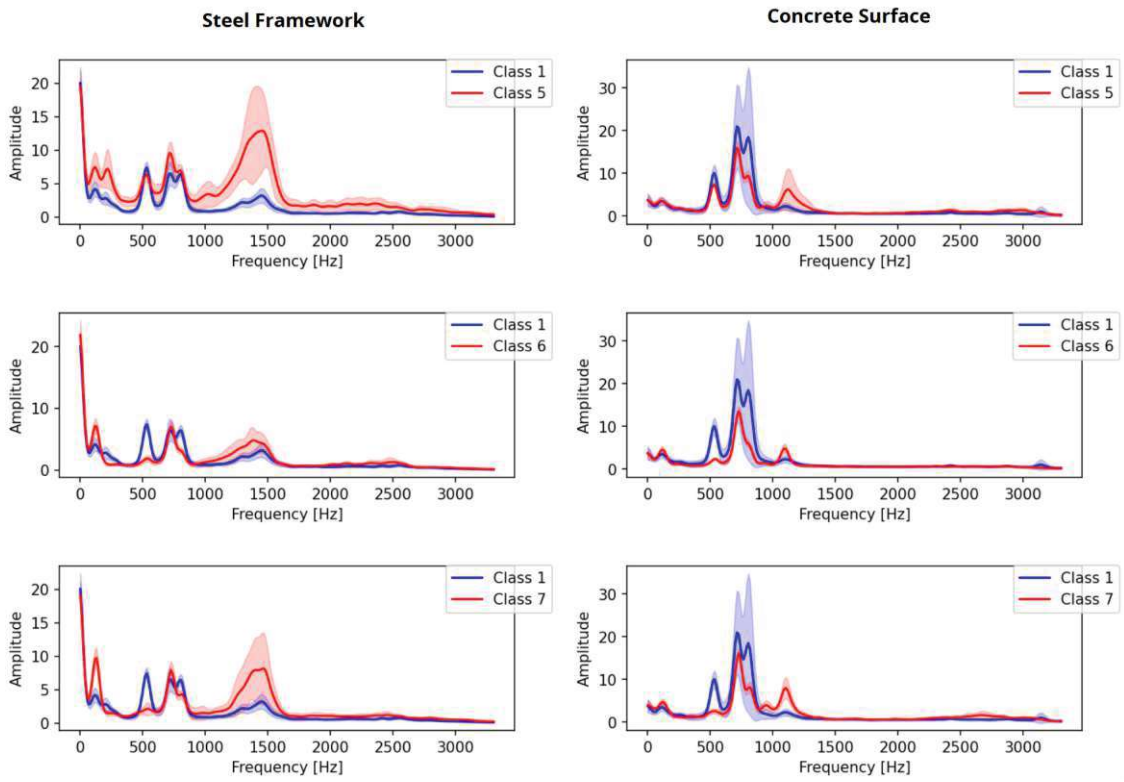

Figure 5.3: Visualization of domain-specific information in the Centrifugal-Pumps dataset, showing the average amplitude [mm] of the FFT vibration signals per frequency [Hz] with a 95% confidence interval. Each anomalous class (5-cavitation, 6-hydraulic blockage, 7-dry running) is compared to the normal class (1-normal class), recorded by same pump operated within a steel framework vs. on a concrete surface.

Table 5.3 presents the experimental results of the meta-learning approaches on the Rainbow-MNIST dataset in the OC-DA setting, showing the average accuracy [%] computed on the target domain datasets. To adapt the Rainbow-MNIST dataset to the OC-DA setting, I designate one class as the normal class. I conducted experiments for each class, using two different values for K. The hyperparameters are presented in more detail in Table 5.5. Performance improves with larger values of K. Furthermore, the results demonstrate the effectiveness of my task sampling strategy in OC-DA MAML, significantly outperforming MAML with classical K-shot learning tasks. Note that OC-DA MAML achieves results on the target domains that are comparable to the ID performance on the source domains, as shown in Table 5.2.

Table 5.4 shows the experimental results on the Centrifugal-Pumps dataset. In each source and target domain combination, note that there are 16 source domains and 4 target domains, as outlined in Section 5.1. For the distribution shift metrics, I down-sampled each domain dataset by reducing the number of examples per class to match the size of the

	$K = 1$		$K = 3$	
Class n	MAML	OC-DA MAML	MAML	OC-DA MAML
0	41.33	**82.17**	44.2	**86.07**
1	36.9	**80.87**	41.63	**81.77**
2	32.33	**80.77**	39.53	**89.57**
3	20.03	**84.93**	31.1	**92.67**
4	37.07	**84.63**	43.87	**91.1**
5	32.37	**82.3**	31.3	**87.87**
6	37.23	**86.1**	31.4	**92.47**
7	35.87	**86.23**	39.23	**93.07**
8	41.73	**82.73**	40.1	**86.73**
9	41.4	**78.87**	41.43	**82.33**
Average	35.6	**82.9**	38.4	**88.4**

Table 5.3: Experimental results of meta-learning approaches on Rainbow-MNIST in OC-DA setting.

minority class. For the ID metric, I collected, shuffled and split the data in the source domains into a training, validation and test dataset. The ID metric is computed as the accuracy [%] on this test dataset. For the ODD metric, I collected, shuffled and split the data in the source domains into a training and validation dataset. The ODD metric is computed as the average accuracy [%] over the class-balanced down-sampled target domain datasets. In meta-learning, the source domains include 12 meta-training domains and 4 meta-validation domains, the target domains include 4 meta-testing domains. The table shows the average accuracy [%] computed on class-balanced down-sampled target domain datasets (meta-testing query sets), as described in Table 5.1. While the OC-DA MAML algorithm outperforms the standard MAML algorithm, the performance gap between MAML and OC-DA MAML is smaller compared to the Rainbow-MNIST dataset. This can be attributed to the well-defined structure and deliberate design of the domains in the benchmark. In contrast, the Centrifugal-Pumps dataset provides less domain-specific structure, making it more challenging for the OC-DA MAML algorithm to fully leverage its advantages. Additionally, the performance drop is notably smaller on the Centrifugal-Pumps dataset compared to the benchmark, suggesting that the distribution shifts are less pronounced. Given that the original MAML algorithm is optimized for within-domain generalization, I attribute its higher performance to these less severe distribution shifts between domains, although they remain significant.

Table 5.5 shows the hyerparameter settings. I used the Adam optimizer with corresponding learning rates α, β and a weight decay of $1e-5$. In all experiments, I used the categorical cross entropy loss.

Source domains	Target domains	ID	ODD	K=2 MAML	K=2 OCDA-MAML
P_1, P_2, P_3 'steel'	P_4 'concrete'	96.2	80.3	87.73	**89.63**
P_1, P_2, P_4 'steel'	P_3 'concrete'	94.77	89.56	74.03	**93.7**
P_1, P_3, P_4 'steel'	P_2 'concrete'	94.86	79.73	76.17	**91.2**
P_2, P_3, P_4 'steel'	P_1 'concrete'	94.23	77.57	91.53	**92.43**
P_1, P_2, P_3 'concrete'	P_4 'steel'	92.63	71.4	92.47	**93.5**
P_1, P_2, P_4 'concrete'	P_3 'steel'	93.93	88.9	86.9	**93.1**
P_1, P_3, P_4 'concrete'	P_2 'steel'	95.07	89.53	81.4	**98.6**
P_2, P_3, P_4 'concrete'	P_1 'steel'	94.67	80.83	83.8	**90.47**
Average		95.5	82.2	84.3	**92.8**

Table 5.4: Experimental results on the Centrifugal-Pumps dataset.

Hyperparameter	Rainbow-MNIST	Centrifugal-Pumps
Input size	28×28	1×256
Meta-batch size $\|I\|$	4	2
Meta-training iterations	30,000	20,000
Inner gradient descent steps k	1	1
Inner learning rate α	0.01	0.01
Outer learning rate β	0.001	0.001
N (Classes per task)	10	5
K (Shots per class)	$\{1, 3\}$	2

Table 5.5: Hyperparameters in MAML and OC-DA MAML.

6 Conclusion

In recent years, the integration of IoT sensor platforms into industrial plants has opened new opportunities for the application of machine learning in various industrial systems. In many real-world applications, machine learning models are developed in controlled laboratory settings before being deployed into critical production environments. However, distribution shifts pose significant challenges in deploying these models effectively. Furthermore, real-world data is often constrained not only by limited volume but also by the scarcity of certain classes (e.g., rare medical conditions, specific industrial failure types). I extended this problem setting to an arbitrary multi-class classification task and introduced the OC-DA problem setting. A major assumption in OC-DA is that the source and target domains share a common structure, allowing one class to encode sufficient domain-specific information to make inferences about the other classes within the domain.

In this work, I proposed a task sampling strategy to adapt any bi-level meta-learning algorithm to the OC-DA setting. Based on this strategy, I modified the MAML algorithm and introduced the OC-DA MAML algorithm. I provided a theoretical analysis of the OC-DA MAML meta-update, demonstrating that OC-DA MAML explicitly optimizes for meta-parameters that enable generalization from one class to the other classes within a domain, and thus, one-class adaptation across domains. My empirical results support these theoretical observations.

I evaluated the OC-DA MAML algorithm on a meta-learning benchmark and demonstrated its robustness in real-world applications using a dataset of vibration-based sensor readings recorded by centrifugal pumps in diverse environments. On the benchmark, OC-DA MAML significantly outperforms MAML with the standard task sampling strategy. In the real-world dataset, I illustrated differences in the data distribution across domains and confirmed that these visual differences cause a substantial performance drop. However, the performance drop is notably smaller compared to the benchmark, suggesting that the distribution shifts are less pronounced. The OC-DA MAML algorithm consistently outperforms the standard MAML algorithm for all source and target domain combinations. I conclude that it is possible to leverage domain-specific information present in one class for efficient domain adaptation. The proposed task sampling strategy in bi-level meta-learning enables generalization from one class to other classes within a domain, and thus, one-class adaptation across domains.

Despite the growing demand for machine learning in industry, transferring models from laboratory settings to real-world deployments is an open challenge. While I achieved promising results on a real-world dataset, the centrifugal pumps were operated within a laboratory setting. It would be interesting to explore the performance of OC-DA MAML in actual

deployments, including a large number of pumps in diverse environments. Additionally, I only considered centrifugal pumps of the same type and size. In reality, it is common to encounter pumps assembled according to a modular system. In such systems, pumps are composed of separate components that can be combined, interchanged, or replaced to create different pump configurations. Future research could therefore explore generalizing the models not only to different pumps in new environments but also to entirely new pump configurations.

List of Figures

List of Tables

Bibliography

[1] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. "Invariant Risk Minimization". In: *arXiv preprint arXiv:1907.02893*. 2019. URL: https://arxiv.org/abs/1907.02893 (visited on 06/10/2024).

[2] Sébastien M.R. Arnold, Praateek Mahajan, Debajyoti Datta, Ian Bunner, and Konstantinos Saitas Zarkias. "Learn2learn: A Library for Meta-Learning Research". In: *arXiv preprint arXiv:2008.12284*. 2020. URL: https://arxiv.org/abs/2008.12284 (visited on 06/10/2024).

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *The Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. Sydney, Australia: PMLR, 2017, pp. 1126–1135.

[4] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. "Online Meta-Learning". In: *The Proceedings of the 36th International Conference on Machine Learning (ICML)*. Vol. 97. Long Beach, USA: PMLR, 2019, pp. 1920–1930.

[5] Ahmed Frikha, Denis Krompaß, Hans-Georg Köpken, and Volker Tresp. "Few-Shot One-Class Classification via Meta-Learning". In: *The Proceedings of the 35th AAAI Conference on Artificial Intelligence*. Vol. 35. 8. AAAI press, 2021, pp. 7448–7456. DOI: 10.1609/aaai.v35i8.16913.

[6] Felix Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Computation*. Vol. 12. 10. MIT press, 2000, pp. 2451–2471. DOI: 10.1162/089976600300015015.

[7] Alex Graves and Jürgen Schmidhuber. "Framewise Phoneme Classification with Bidirectional LSTM and other Neural Network Architectures". In: *Neural Networks*. Vol. 18. 5. IJCNN 2005. 2005, pp. 602–610. DOI: 10.1016/j.neunet.2005.06.042.

[8] Ishaan Gulrajani and David Lopez-Paz. "In Search of Lost Domain Generalization". In: *arXiv preprint arXiv:2007.01434*. 2020. URL: https://arxiv.org/abs/2007.01434 (visited on 06/10/2024).

[9] Clemens Heistracher, Anahid Jalali, Indu Strobl, Axel Suendermann, Sebastian Meixner, Stephanie Holly, Daniel Schall, Bernhard Haslhofer, and Jana Kemnitz. "Transfer Learning Strategies for Anomaly Detection in IoT Vibration Data". In: *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*. Toronto, Canada, 2021, pp. 1–6. DOI: 10.1109/IECON48115.2021.9589185.

[10] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation*. Vol. 9. 8. MIT press, 1997, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[11] Stephanie Holly, Robin Heel, Denis Katic, Leopold Schoeffl, Andreas Stiftinger, Peter Holzner, Thomas Kaufmann, Bernhard Haslhofer, Daniel Schall, Clemens Heitzinger, and Jana Kemnitz. "Autoencoder Based Anomaly Detection and Explained Fault Localization in Industrial Cooling Systems". In: *The Proceedings of the 7th European Conference of the Prognostics and Health Management Society 2022*. Vol. 7. 1. 2022, pp. 200–210. DOI: 10.36001/phme.2022.v7i1.3349.

[12] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. "Meta-Learning in Neural Networks: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 44. 9. 2022, pp. 5149–5169. DOI: 10.1109/TPAMI.2021.3079209.

[13] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *arXiv preprint arXiv:1801.06146*. 2018. URL: https://arxiv.org/abs/1801.06146 (visited on 06/10/2024).

[14] Jana Kemnitz, Thomas Bierweiler, Herbert Grieb, Stefan van Dosky, and Daniel Schall. "Towards Robust and Transferable IIoT Sensor based Anomaly Classification using Artificial Intelligence". In: *Data Science – Analytics and Applications*. Ed. by Peter Haber, Thomas J. Lampoltshammer, Helmut Leopold, and Manfred Mayr. Proceedings of the 4th International Data Science Conference – iDSC2021. Wiesbaden, Germany: Springer Fachmedien Wiesbaden, 2022, pp. 14–19. DOI: 10.1007/978-3-658-36295-9_2.

[15] Jana Kemnitz, Axel Weissenfeld, Leopold Schoeffl, Andreas Stiftinger, Daniel Rechberger, Bernhard Prangl, Thomas Kaufmann, Thomas Hiessl, Stephanie Holly, Clemens Heistracher, and Daniel Schall. "An Edge Deployment Framework to Scale AI in Industrial Applications". In: *2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC)*. Bangalore, India, 2023, pp. 24–32. DOI: 10.1109/ICFEC57925.2023.00012.

[16] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-Shot Image Recognition". In: *Deep Learning Workshop at the 32nd International Conference on Machine Learning (ICML)*. Vol. 2. 1. Lille, France, 2015, pp. 1–30.

[17] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. "WILDS: A Benchmark of in-the-Wild Distribution Shifts". In: *The Proceedings of the 38th International Conference on Machine Learning (ICML)*. Vol. 139. PMLR, 2021, pp. 5637–5664.

[18] Dhiraj Kumar, Aakash Dewangan, Rajiv Tiwari, and D.J. Bordoloi. "Identification of Inlet Pipe Blockage Level in Centrifugal Pump over a Range of Speeds by Deep Learning Algorithm using Multi-Source Data". In: *Measurement*. Vol. 186. 110146. Elsevier, 2021. DOI: 10.1016/j.measurement.2021.110146.

[19] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. "One Shot Learning of Simple Visual Concepts". In: *The Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 33. 2011, pp. 2568–2573.

[20] Da Li and Timothy Hospedales. "Online Meta-Learning for Multi-Source and Semi-Supervised Domain Adaptation". In: *Computer Vision – ECCV 2020*. Vol. 12361. Lecture Notes in Computer Science. Springer, 2020, pp. 382–403. DOI: 10.1007/978-3-030-58517-4_23.

[21] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. "Learning to Generalize: Meta-Learning for Domain Generalization". In: *The Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. Vol. 32. 1. New Orleans, USA: AAAI Press, 2018, pp. 3490–3497. DOI: 10.1609/aaai.v32i1.11596.

[22] Alex Nichol, Joshua Achiam, and John Schulman. "On First-Order Meta-Learning Algorithms". In: *arXiv preprint arXiv:1803.02999*. 2018. URL: https://arxiv.org/abs/1803.02999 (visited on 06/10/2024).

[23] Christopher Olah. *Understanding LSTM Networks*. Colah's Blog. Aug. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 06/10/2024).

[24] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. "Meta-Learning with Implicit Gradients". In: *The Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 32. 11. Vancouver, Canada: Curran Associates Inc., 2019, pp. 113–124.

[25] Sachin Ravi and Hugo Larochelle. "Optimization as a Model for Few-Shot Learning". In: *The 5th International Conference on Learning Representations (ICLR), Conference Track Proceedings*. Toulon, France: OpenReview.net, 2017. URL: https://openreview.net/pdf?id=rJY0-Kcll (visited on 06/10/2024).

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. "Imagenet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision*. Vol. 115. 2015, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[27] Marc Russwurm and Devis Tuia. "Instance Norm improves Meta-Learning in Class-Imbalanced Land Cover Classification". In: *Workshop on Distribution Shifts: Connecting Methods and Applications at the 36th Conference on Neural Information Processing Systems (NeurIPS)*. 2022. URL: https://openreview.net/pdf?id=vYOGXML19sr (visited on 06/10/2024).

[28] Haşim Sak, Andrew Senior, and Françoise Beaufays. "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition". In: *arXiv preprint arXiv:1402.1128*. 2014. URL: https://arxiv.org/abs/1402.1128 (visited on 06/10/2024).

[29] Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical Networks for Few-Shot Learning". In: *The Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 30. Long Beach, USA: Curran Associates Inc., 2017, pp. 4080–4090.

[30] Rajiv Tiwari, D.J. Bordoloi, and Aakash Dewangan. "Blockage and Cavitation Detection in Centrifugal Pumps from Dynamic Pressure Signal using Deep Learning Algorithm". In: *Measurement*. Vol. 173. 108676. Elsevier, 2021. DOI: 10.1016/j.measurement.2020.108676.

[31] Oriol Vinyals. "Contrasting Model vs Optimization Meta Learning". In: *The Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*. Symposium Meta-Learning. Long Beach, USA, 2017. URL: https://neurips.cc/Conferences/2017/Schedule?type=Symposium (visited on 06/10/2024).

[32] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. "Matching Networks for One Shot Learning". In: *The Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 29. Barcelona Spain: Curran Associates Inc., 2016, pp. 3637–3645.

[33] Huaxiu Yao, Linjun Zhang, and Chelsea Finn. "Meta-Learning with Fewer Tasks through Task Interpolation". In: *The 10th International Conference on Learning Representations (ICLR)*. Virtual Event: OpenReview.net, 2022. URL: https://openreview.net/pdf?id=ajXWF7bVR8d (visited on 06/10/2024).

[34] Tiago Zonta, Cristiano André Da Costa, Rodrigo da Rosa Righi, Miromar Jose de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. "Predictive Maintenance in the Industry 4.0: A Systematic Literature Review". In: *Computers & Industrial Engineering*. Vol. 150. 106889. Elsevier, 2020. DOI: 10.1016/j.cie.2020.106889.

[35] Nicolas Zucchet, Simon Schug, Johannes von Oswald, Dominic Zhao, and João Sacramento. "A Contrastive Rule for Meta-Learning". In: *The Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 35. 1880. New Orleans, USA: Curran Associates Inc., 2022, pp. 25921–25936.

My Publications

[1] Clemens Heistracher, Anahid Jalali, Indu Strobl, Axel Suendermann, Sebastian Meixner, Stephanie Holly, Daniel Schall, Bernhard Haslhofer, and Jana Kemnitz. "Transfer Learning Strategies for Anomaly Detection in IoT Vibration Data". In: *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*. Toronto, Canada, 2021, pp. 1–6. DOI: 10.1109/IECON48115.2021.9589185.

[2] Stephanie Holly, Robin Heel, Denis Katic, Leopold Schoeffl, Andreas Stiftinger, Peter Holzner, Thomas Kaufmann, Bernhard Haslhofer, Daniel Schall, Clemens Heitzinger, and Jana Kemnitz. "Autoencoder Based Anomaly Detection and Explained Fault Localization in Industrial Cooling Systems". In: *The Proceedings of the 7th European Conference of the Prognostics and Health Management Society 2022*. Vol. 7. 1. 2022, pp. 200–210. DOI: 10.36001/phme.2022.v7i1.3349.

[3] Stephanie Holly, Thomas Hiessl, Safoura Rezapour Lakani, Daniel Schall, Clemens Heitzinger, and Jana Kemnitz. "Evaluation of Hyperparameter-Optimization Approaches in an Industrial Federated Learning System". In: *Data Science – Analytics and Applications*. Ed. by Peter Haber, Thomas J. Lampoltshammer, Helmut Leopold, and Manfred Mayr. Proceedings of the 4th International Data Science Conference – iDSC2021. Wiesbaden, Germany: Springer Fachmedien Wiesbaden, 2022, pp. 6–13. DOI: 10.1007/978-3-658-36295-9_1.

[4] Jana Kemnitz, Axel Weissenfeld, Leopold Schoeffl, Andreas Stiftinger, Daniel Rechberger, Bernhard Prangl, Thomas Kaufmann, Thomas Hiessl, Stephanie Holly, Clemens Heistracher, and Daniel Schall. "An Edge Deployment Framework to Scale AI in Industrial Applications". In: *2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC)*. Bangalore, India, 2023, pp. 24–32. DOI: 10.1109/ICFEC57925.2023.00012.