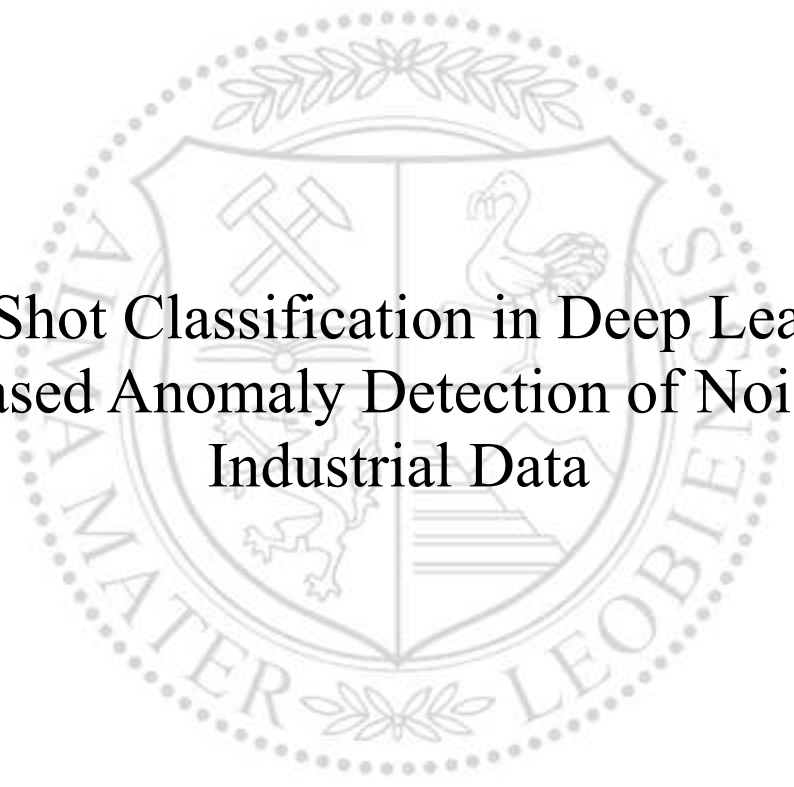




Chair of Automation

Master's Thesis

The background features a large, faint watermark of the University of Leoben seal. The seal is circular and contains a shield with various symbols, including a hammer and pickaxe, a stork, and a lion. The text 'UNIVERSITAS ANNO 1609' is visible around the top and sides of the seal.

Few-Shot Classification in Deep Learning
based Anomaly Detection of Noisy
Industrial Data

Patricia Andrea Freyler, BSc

January 2023



Master's Thesis

Few-Shot Classification in Deep Learning based Anomaly Detection of Noisy Industrial Data

written by

Patricia Freyler

Montanuniversität Leoben

Chair of Automation

Professor:

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary

Supervisor:

Mohamed Ali Thani, MSc

Leoben, January 31, 2023



EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 26.01.2023

Unterschrift Verfasser/in
Patricia Andrea Freyler

Acknowledgment

I would like to express my special thanks of gratitude to my professor O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O’Leary, who agreed to support the thesis. He took his time at every meeting and provided me with valuable ideas and suggestions for my thesis. The meetings and conversations were vital in inspiring me to think outside the box, from multiple perspectives to form a comprehensive and objective critique.

I would like to earnestly acknowledge the sincere efforts and time given by my supervisor Mohamed Ali Tnani. The guidance and feedback has helped me in completing the thesis and give me insights in producing scientific research. Further he gives me the opportunity to do my master thesis on this interesting topic. I am really thankful to him.

Many thanks also to the company Bosch Rexroth, which gave me the opportunity to work and provided my thesis.

Besides, I am also thankful to my colleagues at Bosch Rexroth and my friends. In addition, I would like to highlight the support and organization of the secretariat at the Mechanical and Automation Institute of the Montanuniversity Leoben.

Last, I want to thank my parents, who have always been there in my life and supported me through my study. Without them, I could never had completed my study.

Thank you.

Kurzfassung

Fertigungsprozesse können durch die frühzeitige Erkennung von Anomalien mithilfe von Deep Learning Methoden verbessert werden. Diese Methoden erfordern große Datenmengen, die in der Praxis oft nicht verfügbar sind. Die geringe Anzahl von Vorkommnissen und die daraus resultierende geringe Datenmenge, sowie die große Vielfalt an Prozessanomalien bei Fertigungsprozessen in modernen Produktionsanlagen, stellen eine Herausforderung für herkömmliche Deep Learning Verfahren dar.

Die großen Datenmengen, die für die Konstruktion neuronaler Netze benötigt werden, erfordern hohe Ansprüche an die Qualität und Quantität des Daten Labelings, was zu hohen Kosten führt. Abhilfe verspricht der Bereich des Few-Shot Learnings, der sich mit dem Entwurf leistungsfähiger neuronaler Netze mit begrenzten Datensätzen beschäftigt. Ziel der Arbeit ist es, das Wissen aus Computer-Vision-Methoden auf die neue Anwendungsdomäne der verrauschten Industriedaten zu übertragen und ein effizientes Labelingsystem mit fortschrittlichen Deep-Learning Few-Shot Klassifikationsmethoden für Zeitreihendaten aus der Produktion zu untersuchen. Die wichtigsten Ergebnisse dieser Arbeit sind die folgenden:

Das Prototypical Network(PN), das die euklidische Distanz verwendet, erreicht bei der Verifizierungsaufgabe einen F1-Score von 93,92 %, wenn es auf der Grundlage von 70 guten und 21 schlechten Proben (Datensatz 1) trainiert wurde, und einen F1-Score von 80,01 % mit 17 guten und 6 schlechten Proben (Datensatz 3). Das Matching Network(MN) erreichte einen F1-Score von 87,34 % und 71,81 %. Durch die Implementierung der Cosinus-Distanz erreicht PN einen F1-Score von 95,21 % und MN einen F1-Score von 91,46 % mit Datensatz 1 (Tabelle 4.2). Das Skalarprodukt erreicht eine F1-Leistung von 93,51 % für PN und 88,70 % für MN. Die Anzahl der Shots für das Support Set sollte etwa 5 bis 7 Schüsse mit einem F1-Score von 93,92 % und 94,82 % betragen. Für die Unterstützungsmenge (Support Set) und die Abfragemenge (Query Set) mit einem F1-Score von 92,92 % sind 3 Shots nicht ausreichend. Few-Shot Learning kann den Bedarf an Trainingsdaten erheblich reduzieren.

Die angewendeten Methoden liefern für die untersuchten Datensätze sehr gute Ergebnisse.

Abstract

Manufacturing processes can be improved by early detection of anomalies using Deep Learning methods. These methods require large volumes of data, however in manufacturing processes, there is usually only a small amount of information about anomalies, which leads to biased data.

These small number of occurrences and a resulting small amount of data together with the variety of process anomalies in the manufacturing processes in modern production plants pose challenges for traditional Deep Learning methods. This reduces the potential performance of neural networks in the production environment due to the lack of transferability of the individual models among each other.

The large volume of data needed to build neural networks places high demands on the quality and quantity of data labeling, which results in high costs. The field of Few-Shot Learning, which focuses on the design of high-performance neural networks with limited data sets, promises a potential remedy.

The purpose of the thesis is to transfer knowledge from state-of-the-art computer vision methods to the new application domain of noisy industrial data and investigate an efficient labeling system using advanced Deep Learning Few-Shot classification methods for data streams collected during production. The main results of this work are the following:

The Prototypical Network (PN) using the Euclidean Distance reached an F1-score of 93.92 % on the verification task when trained based on 70 good and 21 bad samples (dataset 1) and an F1-score of 80.01 % with 17 good and six bad samples (dataset 3). The Matching Network (MN) reached an F1-score of 87.34 % and 71.81 %. By implementing Cosine Distance as the final classification, PN achieves an F1-score of 95.21 % and MN an F1-score of 91.46 % with dataset 1 (Table 4.2). The DOT-product achieves an F1-performance of 93.51 % for the PN and 88.70 % for the MN. The number of shots for the support set, should be about 5 to 7 shots with an F1-score of 93.92 % and 94.82 %. Three shots are not sufficient for the support and query set with an F1-score of 92.92 %.

Few-Shot Learning in quality control can significantly reduce the need for training data. Different distance/similarity methods improve the performance of the networks. These techniques provide good results for the data used in this work.

Contents

Affidavit	I
Acknowledgment	II
Kurzfassung	III
Abstract	IV
1 Introduction	1
1.1 Definition of the Problem	1
1.2 Definition of the Objective	1
1.3 Structure of the Thesis	2
2 Background	3
2.1 Deep Learning	3
2.2 Few-Shot Learning	4
2.3 Statistical framework	4
2.3.1 Supervised learning	5
2.3.2 Expected Risk	5
2.3.3 Empirical Risk	5
2.3.3.1 Empirical Risk Minimizer	5
2.3.3.2 Unreliable Empirical Risk Minimizer	6
2.3.4 Taxonomy	7
2.3.5 Embedding Model	8
2.3.6 Meta-Learning for Few-Shot Learning	9
2.3.6.1 Learning the embedding for Few-Shot Learning	10
2.3.7 Training and Testing in Few-Shot Learning	11
2.4 Few-Shot Deep Metric Learning Methods	12
2.4.1 Metric Learning	12
2.4.2 Deep Metric Learning	14
2.4.2.1 Learning feature embeddings	15
2.4.2.2 Learning class representations	15
2.4.2.3 Learning distance/similarity measures	15

2.5	Hyperparameters	16
2.5.1	Convolution Layer	16
2.5.2	Pooling Layer	18
2.5.3	Fully Connected Layer	18
2.5.4	Activation Functions	18
2.5.4.1	Sigmoid Function	19
2.5.4.2	ReLU Function	19
2.5.4.3	Softmax Function	19
2.6	Distance/Similarity Measures	20
2.6.1	P-Norm	20
2.6.2	1-norm (Manhattan Distance)	21
2.6.3	2-norm (Euclidean Distance)	21
2.6.4	Cosine Similarity	21
2.6.5	DOT Product	21
2.6.6	Dynamic Time Warping - DTW	22
2.7	Metrics for Performance Evaluation	24
2.7.1	Classification Accuracy	24
2.7.2	Classification Loss	24
2.7.3	Confusion Matrix	25
2.7.4	F1-Score, Recall and Precision	25
2.8	Deep learning frameworks and libraries	26
3	State-Of-The-Art	27
3.1	Meta Metric-Based Few-Shot Classification	27
3.1.1	Matching Network	27
3.1.2	Prototypical Network	29
3.1.3	Relation Network	31
4	Methodology	33
4.1	Pipeline of the Few-Shot Classifications	33
4.2	Data	35
4.2.1	Data Preprocessing	35
4.2.2	Data Augmentation	36
4.2.3	Dataset - Training Set & Test Set	41
4.3	Few-Shot Classification	42
4.3.1	Training Stage	42
4.4	Testing Stage	44
4.4.1	Embedding Network Structure	44
4.4.2	Models and Metrics of Few-Shot Learning	46
4.5	Overview of the Experiments	46

4.6	Library Details	47
5	Results and Discussion	48
5.1	Results	48
5.2	Discussion	49
5.2.1	Dataset Discussion	50
5.2.2	Sufficient Shots	52
5.2.3	Learning Methods	54
5.2.4	Distance Ranking Approach	55
6	Retrospective	60
6.1	Achievements	60
6.2	Points to Improve	60
6.3	Outlook	61
7	Conclusion	62
8	Appendix	63
	Bibliography	I
	List of Figures	XI
	List of Tables	XIV

1 Introduction

The increasing quest for higher process efficiency and quality requires new and innovative ways to optimize production. Automation plays a significant role in ensuring high quality and high production throughput at the same time [1, 2].

Deep Learning has made it possible to detect faulty production with almost human accuracy. However, these approaches require a large amount of data, doing training in production costly, time-consuming, or even impossible, as the available data is simply not annotated. Few-Shot Learning (FSL) is a field of machine learning that aims to learn new concepts from a few labeled examples [2, 3, 4].

The purpose of the thesis is to present an application of FSL for anomaly detection in noisy industrial data.

1.1 Definition of the Problem

Before providing a literature review, the problem of anomaly detection is formalized here. Similar standard classification tasks, a training and a test set, are required. These sets consist of labeled examples (x, y) , where x is the noisy industrial data and y is the associated label distinguishing between a good sample and an anomaly. In the context of this work, a few data sequences are used, and the aim is to classify each of these sequences as either good or bad (representative of an anomaly). The complexity of the task can be increased by reducing the number of training examples available but keeping the number of test examples at the same level. Another point is that different processes lead to various anomaly patterns, making classification much more difficult. Besides the correct pre-processing of the data, the challenge is to extract essential and relevant features from noisy industrial data. Finally, it should also be noted that the data was collected over a more extended period, so even anomalies in the same category do not always look the same due to changes in the environment and material.

1.2 Definition of the Objective

The purpose of the thesis is to define a system that can classify noisy industrial data. To achieve this, an iterative method is used to improve the performance at each iteration step.

For this approach, networks are applied that relate directly to the concepts described in the state of the art Chapter. The main steps are:

1. Creation of the dataset through data augmentation and pre-processing and subsequent division into a training set and a test set.
2. Implementing a few-shot learning system that performs well in anomaly detection, evaluating a dataset with only a few samples from each class. Therefore, the method, metric, and dataset are varied to achieve the best output. For this reason, several few shot-learning methods and distance metrics for noisy industrial data were investigated.
3. Evaluation of the results against the performance metrics.

1.3 Structure of the Thesis

The general structure considers the problem and provides the detailed solution process in response to anomaly detection using Few-Shot Learning. Chapter 2 gives an overview of Deep Learning and different few-shot methods, focusing on the techniques exploited when designing an effective solution.

After this, the concepts, which are chosen for the work, are provided and explained in detail in Chapter 3.

In Chapter 4, the framework is introduced, and the implementation decisions are justified. The data, as well as the data preprocessing task and data augmentation, are described. Further training and testing with few-shot methods and distancing/similarity measures are explained. Additionally, a pre-training phase is proposed by using a convolutional neural network to get familiar with the noisy industrial data before tackling the few-shot learning classification.

Several possible scenarios are explored in the experiments, as each is characterized by a different combination of parameters, networks, and metrics.

In conclusion, the experiments get evaluated and discussed in Chapter 5. A review follows, highlighting both strengths and weaknesses of this work and some other interesting avenues that could not be investigated but would be worth exploring.

2 Background

2.1 Deep Learning

Deep Learning (DL) is a subset of Machine Learning (ML) and has the advantage of performing better when simple analytical models are not available. The success is based on good generalization. Generalization refers to the ability of the model to properly adapt to new, previously unseen data that comes from the same distribution as the one used to build the model [5, 6]. While ML requires less computational power and less data, DL typically requires less constant human intervention and tend to solve the problem from scratch and not breaking the problem into different parts to solve it [6, 7]. Thereby it discovers complex structures in large datasets by using the backpropagation to indicate how an algorithm should change its internal parameters to compute the representation in each layer from the one in the previous layer [8]. It extracts key internal features where the goal is to achieve higher abstraction levels when transforming raw data into a new representation [9, 8, 10].

The optimization of these deep models with dense architectures requires many iterative updates across many labeled samples and the ability to learn new concepts quickly is limited [11, 12, 13]. Such progress depends on capturing and labeling a huge amount of data, which can be often difficult and costly in practical applications [14]. Current DL approaches struggle to achieve high classification accuracy for applications with few labeled data. They tackle a single problem with a big amount of labeled data successfully but fail to break down the complex architecture of raw data with insufficient labels that have only a few labels [13]. Unlike human visual systems, which are readily able to learn new classes with extremely few labeled examples [13, 15]. Techniques such as regularization reduce overfitting for a small dataset but do not solve the inherent problem associated with fewer training samples. In addition, the large volume of datasets leads to slow learning that requires many weight updates via stochastic gradient descent [13, 3]. Therefore, reducing the required amount of data as well as generalizing to new classes with a limited amount of labeled examples for each novel class has been a growing interest. [4, 15].

To address this problem, new learning methods have been developed in recent years that use only a few labeled samples and are referred to as Few-Shot Learning [16].

2.2 Few-Shot Learning

Few-Shot Learning (FSL) is a field of machine learning that aims to enable learning with only a very limited number of samples [3]. It can be very useful in the case where data collection and annotation is costly challenging [17]. This classification predicts unlabeled samples from unseen classes given only limited labeled samples [15, 18]. A small amount of data and unseen classes make the classification of few shots very difficult and quickly lead to model overfitting to the few training samples from the novel classes [18, 14]. To tackle this challenge there are various approaches, such as Meta-Learning methods, Transfer Learning methods, and Metric Learning methods [14].

Meta-learning is a framework to leverage a large number of similar few-shot tasks to learn how to adapt a base-learner to a new task with only a few supervised examples available [4]. The classification task extracts generalizable knowledge, which enables rapid learning on a new related task with few examples [19, 20]. Transfer Learning methods promote knowledge sharing from the source domain to a target domain-containing a few labeled data with the help of a model pre-trained on a large amount of source data [21, 4]. The major difference from Meta-Learning to Transfer Learning is the existence of an outer objective. Whereas in Transfer Learning the parameters are passed from one task to the next task, in Meta-Learning the parameters, which are passed, are supposed to encode how to learn, instead of how to solve the last task [22]. Transfer Learning uses the knowledge learned in the source task into a target task and does not require a second dataset to begin a new learning process as it is needed in Meta- and Metric Learning [23, 24]. Metric Learning is about learning a representation function that maps objects in an embedded space. The distance in the embedded space should obtain the similarity of the objects, while similar objects move closer and dissimilar objects move farther apart. Feature embeddings and/or distance measures are learned to classify an unseen sample based on its distance to the labeled example. [9, 14, 25]. Samples of the same class are expected to be close together in the embedding space and samples of different classes should be far apart [14, 26].

The above methods can be applied simultaneously for instance when learning feature embeddings of Metric Learning methods by using a Meta-Learning strategy [19, 14].

2.3 Statistical framework

A finite sequence \mathbf{S} of pairs $\{z_i = (x_i, y_i)\}_{i=1}^n$ of size n is the input to the learning algorithm. The set is identically distributed along an unknown distribution P over the space of instances and labels $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. y is the corresponding label for x .

A labeling function $f : \mathcal{X} \rightarrow \mathcal{Y}$ exists for which $f(x) = y$ for all (x, y) is drawn from P .

Following the learning process, the output is a function $h : \mathcal{X} \rightarrow \mathcal{Y}'$ that aims to predict y for each x . The function h comes from a hypothesis space H that best predicts the behavior of f . It is a classifier in the classification and is used to predict the labels of new arriving instances [27, 28].

2.3.1 Supervised learning

In supervised learning, the goal is to find a modeling function $h : \mathcal{X} \rightarrow \mathcal{Y}'$, from a hypothesis class H , that predicts the value of y at x for any (x, y) from P . The objective is to ensure that the predictions are consistent with the true labels of the data. In selecting the best hypothesis to fit the data, its adequacy is determined using a loss function. Loss functions are non-negative and tend to be zero or close to zero when the prediction is correct and higher otherwise [27, 28].

2.3.2 Expected Risk

The expected risk is also known as generalization error or true risk, and intuitively measures the ability of h to predict correctly for all instances $(x, y) \in P$. The true risk E of h with respect to a loss function ℓ is the expected loss of h on the distribution P by a given hypothesis $h \in H$. The goal is to find the hypothesis h that yields the lowest true risk. The risk is an expected true value that depends on the unknown distribution P and therefore cannot be calculated directly (Equation 2.1).

$$E_P^\ell(h) = \int \ell(h(x), y) dP(x, y) = \mathbb{E}[\ell(h(x), y)] \quad (2.1)$$

Therefore, a surrogate value is minimized instead, namely the empirical value of the risk for the available sample S , also known as the empirical risk [28, 27].

2.3.3 Empirical Risk

The empirical risk E with respect to a loss function ℓ is the average loss suffered by the algorithm for the instances of \mathcal{S} given a hypothesis $h \in H$ and a sample $\mathcal{S} = \{z_i^n\}_{i=1}^n$ of size n (Equation 2.2) [28].

$$E_S^\ell(h) = \frac{1}{n} \sum_{i=1}^n \ell(h(x_i), y_i) = \mathbb{E}_n[\ell(h(x), y)] \quad (2.2)$$

2.3.3.1 Empirical Risk Minimizer

Considering a space of input-output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ associated with a probability distribution $P(x, y)$, the conditional distribution $P(y|x)$ is the unknown relationship between inputs and outputs. The loss function $\ell(\hat{y}, y)$ measures the discrepancy between

the predicted output \hat{y} and the real output y . The aim is to minimize the expected risk $E_P^\ell(h)$ (Equation 2.1) for a hypothesis h . h^* is the function (Equation 2.3) that minimizes the expected risk [29, 30, 31].

$$h^*(x) = \arg \min_{\hat{y}} \mathbb{E} [\ell(\hat{y}, y) | x] \quad (2.3)$$

In the first learning process, a family \mathcal{H} of predictive functions is selected and the function $h_n = \arg \min_{h \in \mathcal{H}} E_n(h)$ that minimizes the empirical risk is found. Since h^* is not known, one has to approximate it by using $h \in \mathcal{H}$. The best approach for $h \in \mathcal{H}$ is $h_{\mathcal{H}}^*$, whereas the best approximation in \mathcal{H} is h_n obtained by empirical risk minimization. It is supposed that h^* , $h_{\mathcal{H}}^*$ and h_n are well-defined and unique. The total error is computed out of the approximate error $\mathcal{E}_{app}(\mathcal{H})$ and the estimated error $\mathcal{E}_{est}(\mathcal{H})$ (Equation 2.4) [31, 32, 33, 16].

$$\mathbb{E} [E(h_n) - E(h^*)] = \underbrace{\mathbb{E} [E(h_{\mathcal{H}}^*) - E(h^*)]}_{\mathcal{E}_{app}(\mathcal{H})} + \underbrace{\mathbb{E} [E(h_n) - E(h_{\mathcal{H}}^*)]}_{\mathcal{E}_{est}(\mathcal{H}, n)} \quad (2.4)$$

$h^* = \arg \min_h E(h)$ function that minimizes the expected risk

$h_{\mathcal{H}}^* = \arg \min_{h \in \mathcal{H}} E(h)$ function in \mathcal{H} that minimizes the expected risk

$h_n = \arg \min_{h \in \mathcal{H}} E_n(h)$ function \mathcal{H} that minimizes the empirical risk

The $\mathcal{E}_{app}(\mathcal{H})$ measures how closely the functions in \mathcal{H} can approximate the optimal hypothesis h^* . The $\mathcal{E}_{est}(\mathcal{H}, n)$ measures, instead of the expected risk $E_P^\ell(h)$ inside \mathcal{H} , the effect of minimizing the empirical risk $E_S^\ell(h)$. The total error is related to the number of samples n in \mathcal{S} and the hypothesis space \mathcal{H} [31, 32, 16].

Learning to reduce the total error can be attempted with the following methods: data providing, model determining \mathcal{H} and algorithm searching for the optimal $h_n \in \mathcal{H}$ [16].

2.3.3.2 Unreliable Empirical Risk Minimizer

In general, the estimate error $\mathcal{E}_{est}(\mathcal{H}, n)$ can be reduced by a larger number of samples [31, 33, 34]. The empirical risk minimizer h_n will provide a good approximation $E(h_n)$ to the most efficient $E(h_{\mathcal{H}}^*)$ for the h in the hypothesis space \mathcal{H} (Figure 2.1.a). However, the number of available examples in Few-Shot Learning is small, so the empirical risk $E_S^\ell(h)$ is far from a good approximation of the expected risk $E_P^\ell(h)$, and thus the resulting empirical risk minimizer h_n overfits (Figure 2.1.b) [16].

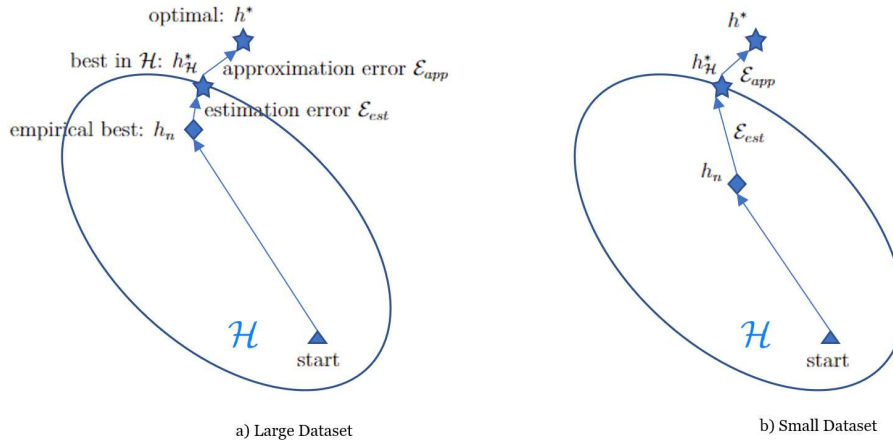


Figure 2.1: Difference between learning with enough and few training samples [16].

This is the core problem of FSL since the empirical risk minimizer h_n is no longer reliable [16].

2.3.4 Taxonomy

To reduce the risk of having an unreliable empirical risk minimizer h_n in FSL, prior knowledge must be used. Knowledge is the information that the learner possesses about the topic in question prior to the collaborative learning phase. In machine learning, knowledge is defined as prior knowledge, which refers to all the information about the problem that is available in addition to the training data. The importance of this knowledge is evident from its role in search and optimization. In order to reduce the amount of training data, it is necessary to provide prior knowledge to the learner, since the learner does not have to derive it from the data itself. The methods by which prior knowledge is used in this category can be divided into three types [16, 35, 36, 37].

1. Data, which use prior knowledge to augment the supervised experience. The number of examples is increased [16].
2. Model, which uses prior knowledge to reduce the complexity of the hypothesis space \mathcal{H} , which results in a much smaller hypothesis space $\hat{\mathcal{H}}$. As shown in Figure 2.2.b, the optimization only takes place in the white space and does not consider the grey area. The reason for this is according to prior knowledge that they are known to be unlikely to contain the optimal $h_{\mathcal{H}}^*$. For this smaller hypothesis space $\hat{\mathcal{H}}$, the number of data is enough for an efficient h_n [38, 39, 40].
3. Algorithm, which uses prior knowledge to search for the best hypothesis $h_{\mathcal{H}}^*$ in \mathcal{H} by providing a good initialization (grey triangle in Figure 2.2.c) [16].

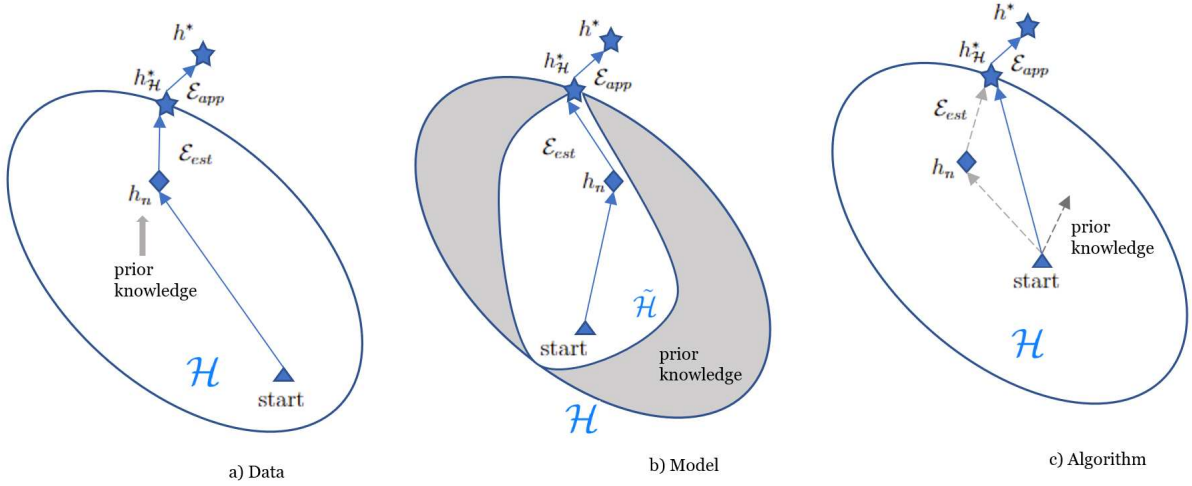


Figure 2.2: Perspectives on how FSL methods solve the few-shot problem [16].

The focus of the work is to use prior knowledge to reduce the complexity of the hypothesis space \mathcal{H} to a smaller hypothesis space $\tilde{\mathcal{H}}$. The minimization of the empirical risk is more reliable and the risk of overfitting is reduced [16].

2.3.5 Embedding Model

The purpose of this method is that samples of different classes can be well separated when samples get mapped into an embedding space, so a smaller $\tilde{\mathcal{H}}$ is needed. It embeds each sample $x_i \in \mathcal{X} \subseteq \mathbb{R}^d$ to a lower dimensional $z_i \in \mathcal{Z} \subseteq \mathbb{R}^m$. Thus similar samples are close together, while dissimilar samples are more easily differentiated. This embedding is learned from prior knowledge. The main components are a function f that embeds a test samples $x_{t,q} \in D_{t,q}$ to \mathcal{Z} from the training set D_t to form the query set $D_{t,q}$, a function g that embeds training samples $x_{t,s} \in D_{t,s}$ to \mathcal{Z} from the training set D_t to form the support set $D_{t,s}$, and a similarity function s that measures the similarity between the two functions $f(x_{t,q})$ and $g(x_{t,s})$ in \mathcal{Z} (Figure 2.3). $x_{t,q}$ is referred to the class of $x_{t,s}$, which embedding $g(x_{t,s})$ is most similar to $f(x_{t,q})$ in \mathcal{Z} according to the similarity. The number of samples in the support set and the query set can vary, so for example a support set can consist of five samples, while the query embedding can consist of only one sample.

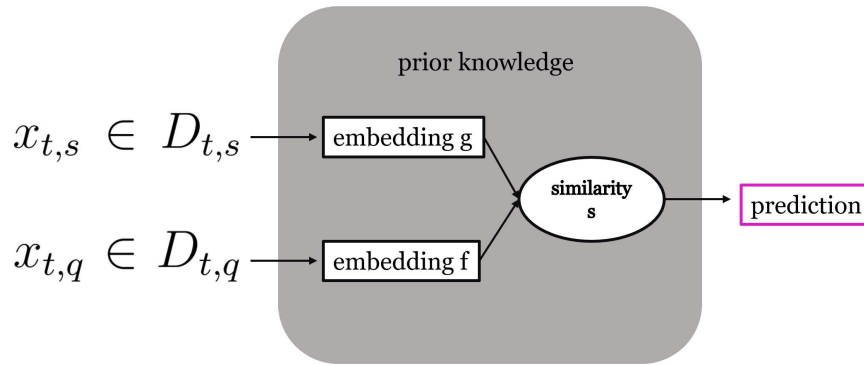


Figure 2.3: Few-Shot Learning problem solved by an embedding model [16].

The first step for the Few-Shot Learning embedding model is to embed the samples using a kernel [16, 41]. These embeddings vary in their complexity and can be learned by different convolution neural networks [42, 19, 43]. Meta-Learning can be used to create more complex embedding models such as Matching Network [19], Prototypical Network [42], and Relation Network [43].

2.3.6 Meta-Learning for Few-Shot Learning

Meta-Learning algorithms make predictions by using the results of existing ML algorithms as input and predicting a class label or a value [44]. Therefore, transferable knowledge is extracted from a task collection and shared to prevent overfitting and improve generalization [15, 45, 46, 47]. This meta-knowledge is obtained from a set of training tasks and is generalized to new test task [18] and can be an optimization strategy [12, 48], a good initial condition [20, 49, 50] or a metric space [42, 19, 43]. The basic idea is to train and adapt an algorithm on many different classification tasks to extract generalizable knowledge, which enables rapid learning on new related tasks with sparse samples [14, 4, 20]. A key feature of Meta-Learning techniques is the goal of optimizing performance by distributing it across tasks to counteract expected loss. This optimization is made by stochastic gradient descent, thereby the validation loss of the base learner is used [11, 4]. The algorithm consists of a meta-training and a meta-testing stage. During the training, the tasks usually mimic the settings in the test phase to improve the generalization ability of the model and to reduce the gap between the training and test settings [18, 15]. Reduction is achieved through multiple episodes that form the framework of episodic training in Meta-Learning [13, 4].

The algorithm in the meta-training selects at first n -ways k -shots sample from D_t within these classes. n classes are selected from the labeled data and k samples are chosen randomly in each of the n classes, which generates the base support set $D_{t,s}$ [15, 51]. On the basis of $D_{t,s}$ an embedding g is created, which generates a classifier for the given n

classes. The Few-Shot Learning tasks from the meta-training set learns the embedding g . Equation 2.5 shows the prediction \hat{y}_j of the label for a test instance $x_{t,j}$ [51].

$$\hat{y}_j = g(D_t)(x_{t,q,j}) \quad (2.5)$$

To measure the performance of the classifier that maps g to the n classes when confronted with $D_{t,s}$, a test set $D_{t,q}$ called query set, is selected with these n classes. The purpose of a good g is to achieve a low loss value after predicting the labels of the instances from the query set (Equation 2.6) [51].

$$\min_g \sum_{(D_{t,s}, D_{t,q}) \sim D_t} \left(\sum_{(x_{t,q,j}, y_{t,q,j}) \in D_{t,q}} \ell(f(D_{t,s}(x_{t,q,j}), y_{t,q,j})) \right) \quad (2.6)$$

In Equation 2.6 $(D_{t,s}, D_{t,q}) \sim D_t$ denote the enumeration of all sampled tasks from the seen training class set. The dataset $D_{t,s}$ and $D_{t,q}$ shows all the sample tasks in the meta-training. The loss function l calculates the difference between the prediction and the true label for each instance in the $D_{t,q}$ [51, 15].

2.3.6.1 Learning the embedding for Few-Shot Learning

The embedding function g , which is in the space of possible feature-vectors and classes, extracts features of the input samples and transform them into a latent space with d dimensions. This transformation can be done by ANN, RNN, CNN, etc. The Meta-Learning in Few-Shot Learning is shown in Algorithm 1 [51, 52].

Algorithm 1 Meta-Learning for Few-Shot Classification [51]

Input: Training class set D_t

Steps:

- 1: **for all** iteration=1,... **do**
 - 2: Sample n-way k-shot $(D_{t,s}, D_{t,q})$ from D_t
 - 3: **for all** $(x_{t,q,j}, y_{t,q,j}) \in D_{t,q}$ **do**
 - 4: Predict $\hat{y}_{t,j} = f(D_{t,s})(x_{t,q,j})$ based on Equation 2.5
 - 5: Compute loss $\ell(\hat{y}_{t,q,j}, y_{t,q,j})$ as Equation 2.6
 - 6: **end for**
 - 7: Compute gradient $\nabla_f \sum_{(x_{t,q,j}, y_{t,q,j}) \in D_{t,q}} \ell(\hat{y}_{t,q,j}, y_{t,q,j})$
 - 8: Mapping g get updated with the selected optimizer.
 - 9: **end for**
 - 10: **return** Few-Shot classifier mapping g
-

Few-Shot Learning classification works well when the embedding function makes similar objects appear close to each other and dissimilar ones appear far apart. The following Equation 2.7 shows that for a test case $x_{t,q,j}$ the embedding function predicts based on

an soft nearest neighbor rule.

$$\hat{y}_j = g(D_t)(x_{t,q,j}) = \sum_{(x_{t,s,i}, y_{t,q,j}) \in D_t} \mathbf{sim}(x_{t,q,j}, x_{t,s,i}) y_{t,s,i} \quad (2.7)$$

The segment $\mathbf{sim}(x_{t,q,j}, x_{t,s,i})$ measures the similarity between each training instances $x_{t,s,i}$ and each test instance $x_{t,q,j}$ [51, 15, 17].

This measurement of similarity can be done by different distancing methods, which will be discussed in the following. The important features for the few-shot classification get emphasized by learning a good embedding, which can be also used for few-shot tasks from unseen class set [51].

2.3.7 Training and Testing in Few-Shot Learning

In FSL the training dataset D_t (Equation 2.8) is split into smaller embedded support sets $D_{t,s}$ and query sets $D_{t,q}$.

$$D_t = \{(x_i, y_i); x_i \in \mathcal{X}; y_i \in Y_t\}_{i=1}^{n_t} \quad (2.8)$$

The support set consists of labeled data. Even if there is only one example from each class in the training process, it is possible to train a deep neural network. At the test time, the support set can only provide additional information. The classification task is performed on a new dataset, the test set $D_{t'}$ (Equation 2.9) [53].

$$D_{t'} = \{(x_i, y_i); x_i \in \mathcal{X}_{t'}; y_i \in Y_{t'}\}_{j=1}^{n_{t'}} \quad (2.9)$$

The classifier seeks to learn from $h : \mathcal{X} \rightarrow \mathcal{Y}'$ that can classify in $D_{t',q}$ instances correctly. The task is called n-way k-shot classification if $D_{t,s}$ contains n classes and k labeled samples per class. If the size of each class in $D_{t,s}$ is one, it is called one-shot classification, otherwise, it is denominated few-shot. The evaluation process goes over many episodes. In each episode n classes are first randomly selected from the new label set and then randomly k samples from each of the n classes to form a support set and m samples from the remaining examples of these n classes to form a query set.

$\mathcal{X}_{t,q}$ and $\mathcal{Y}_{t,q}$ indicates the set of labels or instances in the query set. Based on the learning algorithm a classifier $g(\cdot | D_t, D_{t,s}^{(e)})$ is returned, given the base dataset and the e^{th} support set, which predicts the labels of the query instances as $\hat{\mathcal{Y}}_{t,q}^{(e)} = g(\mathcal{X}_{t,q}^{(e)} | D_t, D_{t,s}^{(e)})$. The classification accuracy in the e^{th} episode is represented by a^e and measures the performance of the learning algorithm over all episodes [14].

The pseudocode of the classification using n-way k-shot is shown consequently [15, 17].

Algorithm 2 Evaluation procedure of n-way k-shot classification [14]

Input: $\mathbb{D}_t = \{(x_i, y_i); x_i \in \mathcal{X}_t; y_i \in Y_t\}_{i=1}^{n_t}$
 $\mathbb{D}_{t'} = \{(x_i, y_i); x_i \in \mathcal{X}_{t'}; y_i \in Y_{t'}\}_{j=1}^{n_{t'}}$
 E ...Number of episodes

Steps:

- 1: $e \leftarrow 0$
 - 2: **repeat**
 - 3: $e \leftarrow e + 1$
 - 4: Randomly select n classes from $\mathcal{Y}_{t'}$.
 - 5: Randomly select k samples from each class as the support set $D_{t,s}^{(e)}$.
 - 6: Randomly select m samples from the remaining samples of n classes as the query $D_s^{(e)} = \{(\mathcal{X}_{t,q}, \mathcal{Y}_{t,q})\}$.
 - 7: Save predicted labels $\hat{\mathcal{Y}}_{t,q}^{(e)} = g(\mathcal{X}_{t,q}^{(e)} | D_t, D_{t,s}^{(e)})$.
 - 8: Compute accuracy $a^{(e)} = \frac{1}{M} \sum_{j=1}^M \mathbb{1} [\hat{\mathcal{Y}}_{t,q}^{(e)} = \mathcal{Y}_{t,q}^{(e)}]^1$.
 - 9: **until** $e = E$
 - 10: **return** mean accuracy $\frac{1}{E} \sum_{e=1}^E a^{(e)}$
-

2.4 Few-Shot Deep Metric Learning Methods

The intention of Metric Learning is to learn a distance measure among instance pairs that assigns a small/large distance to semantically similar/dissimilar instances. The metric is learned related to the base dataset D_t in few-shot classification. The distance between the new query samples and the new support samples related to the learned measure is calculated. This classifies the query samples of the new classes. In contrast to the Metric Learning, which is similar to learning a linear transformation of the original features, deep Metric Learning learns the feature embeddings and the distance measure mostly separately. The result is that the non-linear data structure is captured and more discriminatory feature representations are generated [9, 14, 54, 26].

Before going into detail about the different categories of Deep Metric Learning and its methods, the difference between Metric Learning and Deep Metric Learning is described separately in the next sections.

2.4.1 Metric Learning

Metric Learning is based on a distance metric that looks after similarity/dissimilarity among samples uses an optimal distance metric for learning tasks. The term distance metric comes from mathematics and refers to a function $d: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \cup \{\infty\}$. The function d is a metric on a random set X satisfying positiveness, symmetry, and triangle inequality for all $x, y, z \in \mathcal{X}$.

1. Positiveness: $d(x, y) > 0$ if $x \neq y$, and $d(x, x) = 0$.

2. Symmetry: $d(x, y) = d(y, x)$.
3. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$.

A metric space is a set that has a metric applied to it. It is a pair (\mathcal{X}, d) , where d is a metric on \mathcal{X} . $d(x, y)$ is denominated as the distance between points x and y [55, 56].

In machine learning the metric does not retain its original definition from mathematics, but usually refers to a general measure of similarity or dissimilarity. The general concept is to learn a parametric similarity function using a set of queries where the groundtruth is available. For each query there is a set of similar and a set of dissimilar data to the query. The goal is to find the optimal parameters of the similarity function in the learning task so that the obtained similarity is as close as possible to the ground truth. A common parametric similarity function is Equation 2.10:

$$d_W(x_i, x_j) = x_i^\top W x_j \quad (2.10)$$

x_i, x_j are two vectors in \mathbb{R}^D and $\mathbf{W} \in \mathbb{R}^{D \times D}$. It is based on a projection matrix W of size $q \times r$. This represents the projection of the original data into a q -dimensional space specified by the r rows of W [57, 58, 59].

A new data representation in a transformed space is provided by the distance metric. The algorithm performance get improved by learning data for specific tasks and has a good ability to distinguish the items of different classes. As seen in Figure 2.4.c the purpose is to reduce the distance between samples of the same class (similar objects) and increase the distance among samples of the different class (dissimilar objects) [9, 60].

In general, the Metric Learning method uses a linear Metric Learning with a linear projection. This allows more flexible constraints in the transformed data space and improves learning performance. Furthermore, they are robust to overfitting, because pair or triplet-based constraints become much harder to satisfy in a nonlinear, high-dimensional kernel space [9, 54]. Nevertheless, this is not the best way to interpret and classify data. The performance of linear metric transformation over the new representation of data is not optimal, because they achieved unsatisfied results on non-linear feature structure. This limits the method in solving many real-world problems, which have mostly non-linear characteristics. A higher success can be achieved by carrying the issue to a non-linear space by using kernel approaches, which perform well on solving non-linear problems, but have a negative result against overfitting. To counteract this problem, a much more compact and complex solution was developed, the Deep Metric Learning [9, 61].

2.4.2 Deep Metric Learning

Deep Metric Learning is an interaction of Metric Learning and deep learning uses neural networks to automatically learn discriminative features. The limit of the performance in Metric Learning is the classification and clustering on raw data. Feature engineering for example preprocessing or feature extraction are useful methods to tackle this problem, but they are not directly within the classification structure. With the use of deep learning, on the other hand, the high dimensional data can be represented directly as a classification problem. Therefore the data is converted into a new feature space with higher discrimination power [9, 14].

Deep Metric Learning provides a better solution for nonlinear data by solving this problem using activation functions that add non-linearity within the architecture. Deep Metric Learning develops problem-oriented solutions by learning from raw data, using deep architectures by obtaining similarities between embedded features through non-linear learning (Figure 2.4.d) [9, 14].

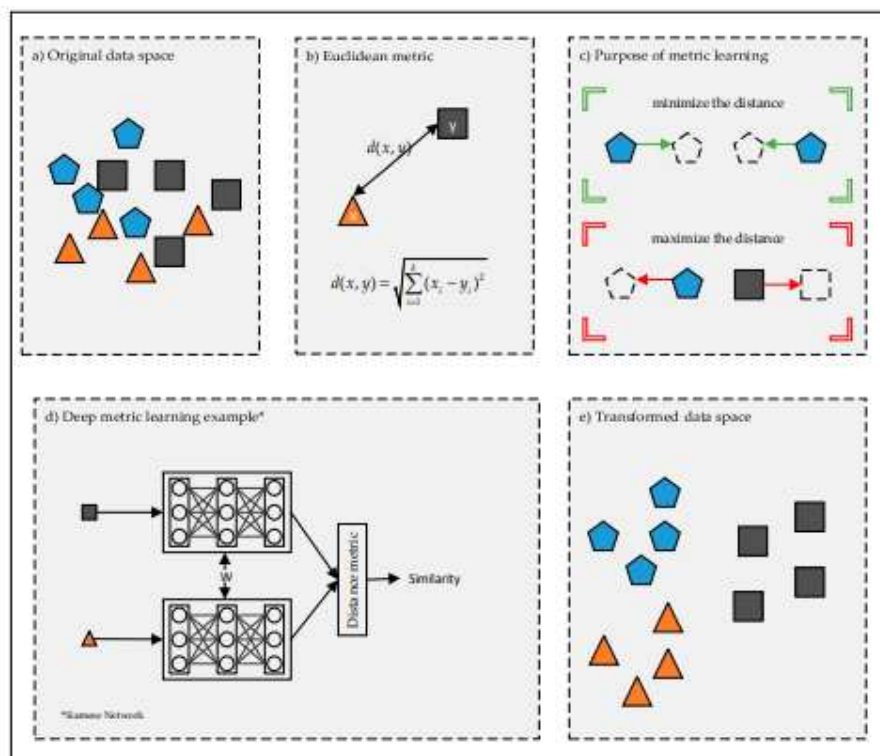


Figure 2.4: Deep Metric Learning [9]

The success of these new methods relies on the ability to effectively recognize the similarity between samples accurately. Besides this an appropriate structure of the network, a good sampling strategy and an efficient distance metric related to the task are the challenging factors to improve the performance of the network model. These are crucial factors for the success of the network in deep Metric Learning [9, 62].

Many Few-Shot Metric Learning methods calculate the distance not only between each query sample and support sample, moreover, they compare class representations like subspaces or prototypes with the query samples. In general, these methods are categorized into three main groups according to the component they are improving on: learning feature embeddings, learning class representations and learning distance/similarity measures [14].

2.4.2.1 Learning feature embeddings

Deep learning embedding methods target to learn a feature embedding (Section 2.3.6.1) from data using for example, CNN, RNN, LSTM, etc. The metric distance between two feature vectors preserves their pairwise encoded semantic relationship, e.g., in the case of contrastive loss [62, 63]. This feature embedding method (Section 2.3.5) assumes that the network extracts discriminative features and generalizes to novel classes well. To resolve the issue of overfitting and data scarcity, data augmentation techniques and multi-task learning is used [14, 62].

2.4.2.2 Learning class representations

In the classical feature learning embeddings method such as in the Matching Network [19] and the Siamese Network [64] the classification is defined over the measuring and comparing the distance between each support sample and query sample. The Matching Network compares the query feature and each support feature with the cosine distance and computes an average distance for each class. It encodes the support and the query samples in the context of the entire support set and uses episodic training for few-shot classification [42, 14, 15].

Prototypical Network [42] is a method, which compares the query features and the class mean of support features with the euclidean distance in the learned embedding space. The mean of the feature embeddings of the support samples in the class creates the centroid of each class, called prototypes. The classes learn prototypes to reduce the loss while training the episodes and serve as reference vectors for each class. For the further improvement of the representation potential, prototypes take a single sample, weighted an average, feature embeddings or learning in an end-to-end manner. The idea is that each class can be represented by a single prototype in an existing embedding space. Around this prototype of the corresponding classes all instances get clustered [42, 14]. The learning class representations have the ability to generalize on a new class even with less samples [42, 14, 15].

2.4.2.3 Learning distance/similarity measures

The methods described above learn feature embeddings or make a class representation. They have mostly a fixed distance or similarity measure for example an Euclidean distance

[42] or a Cosine similarity [19] for the few-shot classification. Another approach to improve the classification accuracy is to learn parameters in these fixed measures or make a novel measure. By using neural networks similarity scores can be learned [14, 43].

Relation Networks [43] compute the Few-Shot Learning classification with a neural network to model the similarity of feature embeddings. This network is composed of an embedding module, which acts like a feature extractor and a relation module. The embedding module consists on convolution blocks with convolution layers to map the original samples into a feature embedding space. The relation module builds on two convolutional blocks and two fully-connected layers. This module calculates the similarity between the support and query samples [14]. The Relation Network also learns a metric for comparison of the embedding. The idea is quite similar to Prototypical Network, because it averages the embeddings for each class together to a single prototype, but it replaces distance with a learnable relation module [43, 15, 17]. The flexibility of the model results from the learnable similarity measure [14].

2.5 Hyperparameters

Deep Learning converts the linear input data and models into non-linear output to support the learning of higher order polynomials beyond one degree for deeper networks. A typical diagram of deep learning is shown in Figure 2.5.



Figure 2.5: Deep learning architecture [65]

At first, the raw data gets fed into the input layer of the deep neural network. The hidden layers are created of several blocks and variations that make up the depth of the neural network. The output layer presents the network result and can be further used for special classifications or predictions with associated probabilities [66]. This convolutional neural network consists of four components, the convolution layer, the fully connected layer, the pooling layer and an activation function. In general, several convolution layers are followed by a pooling layer. At the end there is often a fully connected layer [67, 65].

2.5.1 Convolution Layer

Convolution is a mathematical operation in which two functions are combined to produce another function. It is an integral that represents the degree of overlap of a function g

with another function f , thus "merging" one function with another. The convolution of two functions f and g over an infinite range $[-\infty, \infty]$ is shown in Equation 2.11 [68, 69]:

$$(f * g)(t) = f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (2.11)$$

The $*$ does not represent multiplication, while $(f * g)(t)$ denotes convolution of the functions f and g . τ is the formal variable for integration process and t is the physical time.

This mathematical operation forms the basis for the convolution layer. Convolution of a two-dimensional data set, can be viewed as a series of convolutions in which one function, which is referred to as the kernel, is pushed (folded) over another two-dimensional function, multiplied, and added. The Figure 2.6 shows the convolution of a function with a kernel function that results in multiple data points. The kernel slides to each position of the image and calculates a new pixel as the weighted sum of the pixels over which it slides [68, 69, 70].

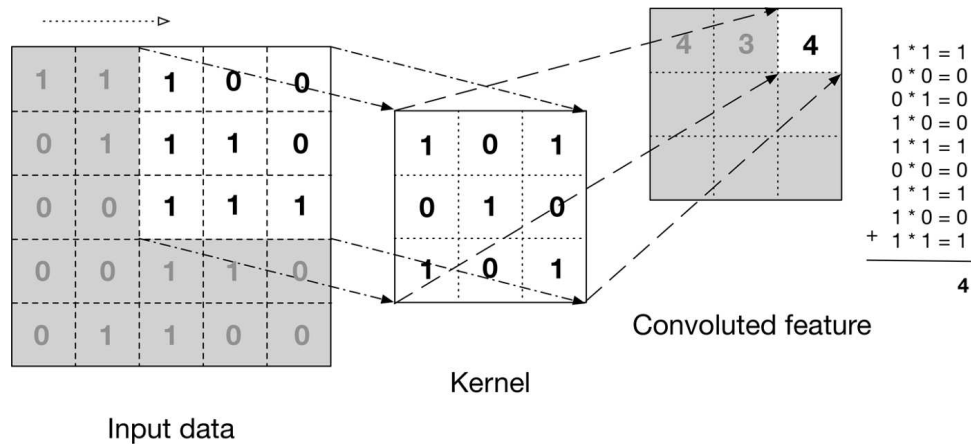


Figure 2.6: Convolution of kernel and a function [70].

The convolutional layers detect the local patterns and features in data from the previous layers. The output of a convolution layer k is $f_k(x) \in \mathbb{R}^{n_k}$ (Equation 2.12) for every $x \in \mathbb{R}^d$ [66].

$$f_k(x)_h = \sigma_k(\langle w_k^t, f_{k-1}^p(x) \rangle + (b_k)_h) \quad (2.12)$$

This output is valid for every $p \in [P_{k-1}], t \in [T_k], h := (p-1)T_k + t$. The inner product between a filter of layer k and a patch at layer $k-1$ calculates the value of each neuron. Afterward the bias is added and the activation functions are applied. Thus the number of neurons at layer k is $n_k = T_k P_{k-1}$, the width of layer k . It covers most of the used variants, because every patch can be an arbitrary subset of neurons of the identical layer [71]. These layers are composable, which means they feed the output of one convolutional layer into another. The network can detect higher-level and more abstract features with

each layer [72].

2.5.2 Pooling Layer

The pooling layers semantically combine similar features into one. To perform the pooling process as it is shown in the Figure 2.7, a window is chosen and the input elements lying in that window are passed through a pooling function. The advantage of using pooling layers is to decrease the number of trainable parameters and to introduce translation invariance [66, 73, 65, 74].

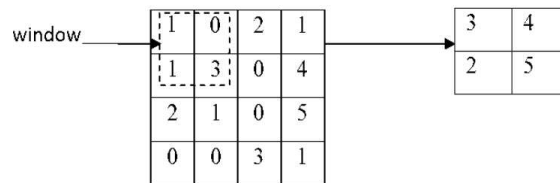


Figure 2.7: Pooling process by a 2x2 window [65]

This pooling function creates another output vector. There are several pooling techniques, for example average pooling and max-pooling. The max-pooling function has a good performance and reduces the input dimensions significantly [75]. It computes the maximum of the element over every patch from the previous layer. The number of neurons outputs at layer k is $n_k = P_{k-1}$, because there are P_{k-1} patches at layer $k - 1$. The output $f_k(x) \in \mathbb{R}^{n_k}$ (Equation 2.13) for every $x \in \mathbb{R}^d$ and $p \in [P_{k-1}]$ [66, 71].

$$f_k(x)_p = \max \left((f_{k-1}^p(x))_1, \dots, (f_{k-1}^p(x))_{l_{k-1}} \right) \quad (2.13)$$

The max-pooling layer allows the next convolutional layer to operate on the most prominent feature map as it takes only the maximum values from the previous feature map. Furthermore they make the network more invariant to small transformations of the data [72, 65].

2.5.3 Fully Connected Layer

This layer is similar to the fully connected network. After the output is passed into the fully connected layer the dot product of weight vector and input vector is calculated to get the final output [74, 65, 71].

2.5.4 Activation Functions

The primary function of the activation function is to introduce non-linearity among the hidden layers or the output by computing the weighted sum of input and biases. This

weight makes the decision to determine if a neuron gets pushed or not. In general, it changes the given data through some gradient processing, which is mostly gradient descent. In the following it creates the output for the neural network containing the parameters in the data. A linear output has no activation function [66, 71].

2.5.4.1 Sigmoid Function

The sigmoid function (Equation 2.14) is a non-linear and bounded differentiable real function. It is used for real input data, with positive derivatives everywhere and some degree of smoothness [66, 71, 76].

$$f(x) = \frac{1}{1 + \exp(-t)} \quad (2.14)$$

2.5.4.2 ReLU Function

The rectified linear unit (ReLU) function is one of the widely used activation function. It is an approximately linear function (Equation 2.15), which allows the function the abilities of linear models, which make them easy to optimize with gradient-descent methods. The ReLU applies a threshold operation to each input sample, where values less than zero are set to zero [66, 71].

$$\sigma(t) = \max(0, x) \quad (2.15)$$

2.5.4.3 Softmax Function

The softmax function is mostly used in the last output layer to make a decision and calculates the probability distribution from a vector of real numbers. It computes the value to the input variable related to their weight. The output is in a range between 0 and 1 and the sum of the probabilities is equal to 1. The softmax function (Equation 2.16) returns the probabilities of each class. The target class has the highest probability [77, 78, 66].

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (2.16)$$

2.6 Distance/Similarity Measures

The type of distance or similarity measures is the decisive factor in clustering and classification of data points. One of the biggest challenge is to choose the right distance measure for a given dataset [79, 80, 81, 82]. From a mathematical perspective the distance describes a quantitative degree of how far apart two objects are. This is often called metric, while non-metric distance measures are sometimes named divergence. The power of the relationship or the matching among two classes or features is shown by the similarity [82, 80].

Some of the popular distance and similarity methods used in few-shot classification are described in more detail below.

2.6.1 P-Norm

In mathematics, the p-norms are a class of vector norms defined for real numbers $p \geq 1$. This norm induces the Minkowski distance (Equation 2.17), which is a family of distances defined for $x, x' \in \mathbf{R}^d$ and $p \geq 1$ [28].

$$d_p(x, x') = \|x - x'\|_p = \left(\sum_{i=1}^d |x_i - x'_i|^p \right)^{\frac{1}{p}} \quad (2.17)$$

The Figure 2.8 shows that if $p < 1$ is not a proper distance, because it violates triangle inequality.

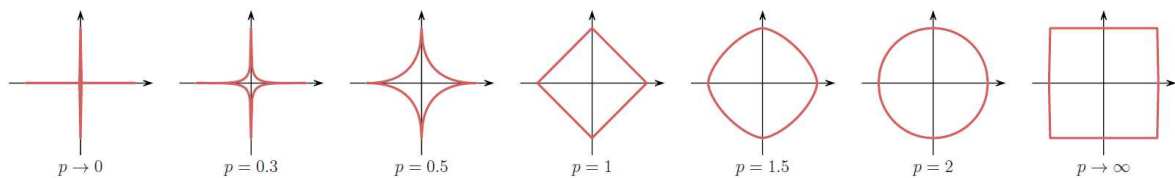


Figure 2.8: Unit circles for p-values in Minkowski distances [28]

The most widely used are the 1-norm, 2-norm, and ∞ -norm. By changing the variable p, the computational distance between the data points leads to new distance metrics [81, 28].

1. $p = 1 \rightarrow$ 1-norm (Manhattan Distance)
2. $p = 2 \rightarrow$ 2-norm (Euclidean Distance)
3. $p = \infty \rightarrow$ 3-norm (Chebychev Distance)

2.6.2 1-norm (Manhattan Distance)

The Manhattan distance (Equation 2.18), which is also called city block distance, measures the distance between two points along axis at right angles [81, 79, 82].

$$d_1(x, x') = \|x - x'\|_1 = \sum_{i=1}^d |x_i - x'_i| \quad (2.18)$$

2.6.3 2-norm (Euclidean Distance)

The Euclidean distance describes the distance between two data points with a straight line and is widely used in classification and clustering settings. The L2 distance determines the root of square differences among the coordinates of a pair of objects (Equation 2.19). 2-norm has the additional property of being rotation and translation invariant, whereas 1-norm is only translation invariant [81, 79, 82, 28].

$$d_2(x, x') = \|x - x'\|_2 = \left(\sum_{i=1}^d |x_i - x'_i|^2 \right)^{\frac{1}{2}} = \sqrt{(x - x')^\top (x - x')} \quad (2.19)$$

2.6.4 Cosine Similarity

The cosine similarity measures the cosine of the angle between two vectors of n dimensions as it is shown in Equation 2.20. This similarity compares the two vectors of attributes x and y by using a dot product and magnitude. [81, 79, 82].

$$\cos(x, x') = \frac{x^\top x'}{\|x\|_2 \|x'\|_2} \quad (2.20)$$

The generalization of the cosine similarity is parameterized by a matrix \mathbf{M} (Equation 2.21):

$$K_M(x, x') = x^\top M x' \quad (2.21)$$

2.6.5 DOT Product

The dot product is also called as the inner product of two vectors as it is shown in Equation 2.22. It yields a scalar for binary vectors, the dot product is called the number of matches or the overlap. It is the simplest linear kernel between instances in the original space \mathcal{X} [82, 28].

$$K_{lin}(x, x') = x^\top \cdot x' \quad (2.22)$$

The kernel corresponds to cosine similarity without normalization or bilinear similarity when $\mathbf{M}=\mathbf{I}$

2.6.6 Dynamic Time Warping - DTW

Dynamic Time Warping computes discrepancy measurement between two sequences and returns their optimal temporal alignment, because of their invariance to warping in the time axis (Figure 2.9). A main advantage is its ability to work with series of different lengths and phases. It finds the minimum distance between two time series, where the sequences are distorted by shrinking or stretching the time dimension. [83, 84, 28].

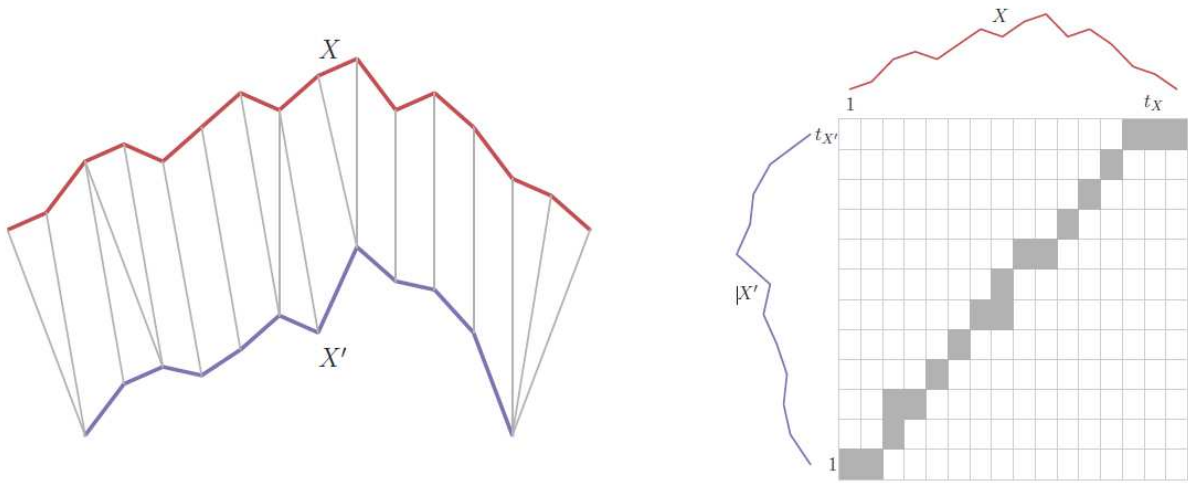


Figure 2.9: Dynamic Time Warping of two sequences [28]

Two temporal signals X and X' are given with a length of $|X|$ and $|X'|$:

$$X = x_1, x_2, \dots, x_i, \dots, x_{|X|} \quad (2.23)$$

$$X' = x'_1, x'_2, \dots, x'_i, \dots, x'_{|X'|} \quad (2.24)$$

These sequences composed a warping path W (Equation 2.25) to define the correspondence of $x_i \in X$ to $x'_i \in X'$ keeping the boundary condition, which allocates the first and last elements of X and X' to each other. k is set as the length of the alignment path [85, 86].

$$W = w_1, w_2, \dots, w_k \quad \max(|X|, |X'|) \leq k < |X| + |X'| \quad (2.25)$$

The element k^{th} is shown in Equation 2.26, i is the index for X and j from X' [86].

$$w_k = (i, j) \quad (2.26)$$

The path begins at each time series $w_1 = (1, 1)$ and ends at $w_K = (|X|, |X'|)$ to ensure the use of all indexes from the two sequences in the warping path. The optimal path is the minimum distance shown in the Equation 2.27, which is mostly calculated by the Euclidean distance [86, 87].

$$Dist(W) = \sum_{k=1} Dist(w_{ki}, w_{kj}) \quad (2.27)$$

This optimal alignment path of $|X|$ and $|X'|$ is shown in the cost matrix D . In Figure 2.10 the second path $|X'|$ is replaced by the variable name $|Y|$. The axes of D represents the time, where X is applied on the x-axis and Y on the y-axis [86].

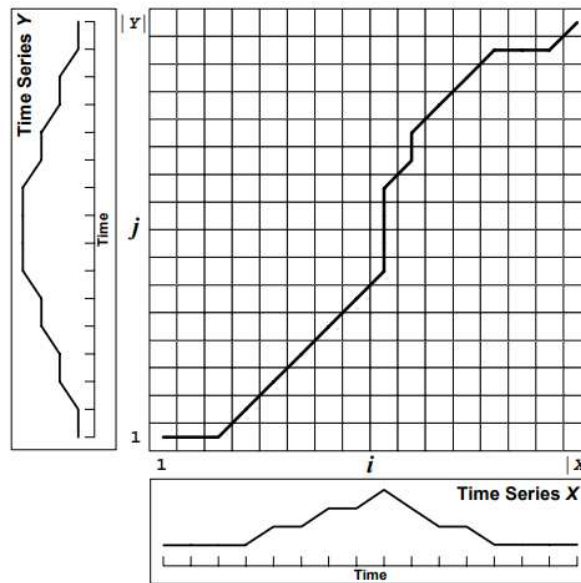


Figure 2.10: Cost matrix with the optimal warp path [86].

Each cell of $|X|$ and $|Y|$ is evaluated once in a constant time and results in an optimal time of $O(N^2)$ for the algorithm yields an improvement [85]. The space complexity and quadratic time is the limiting factor of the DTW algorithm and makes it costly in large datasets. Fast Dynamic Time Warping is a possible solution, which limits the cells that are calculated in the cost matrix and improves the optimal time to $O(N)$ in both time and space (Figure 2.11) [88, 86].

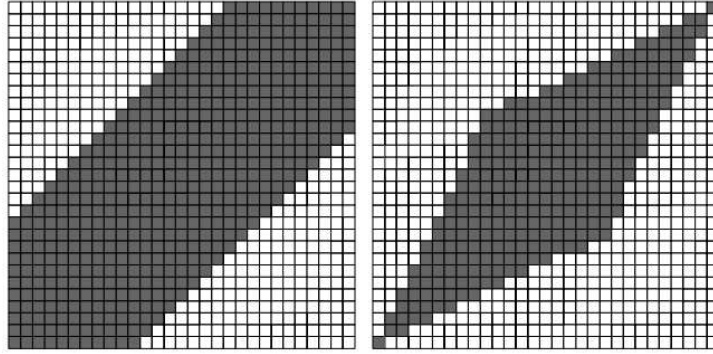


Figure 2.11: Cost Matrix in Fast Dynamic Time Warping [86]

2.7 Metrics for Performance Evaluation

In supervised learning the methods for evaluation include Classification Accuracy, Classification Loss, Confusion Matrix, F1-Score, Recall, and Precision. The metrics compare the true values with the predicted values, which evaluates the performance of the model. The expressions positive and negative relate to the prediction of the classifier, while the expressions true and false relate to whether that prediction matches the instance label.

2.7.1 Classification Accuracy

Accuracy (Equation 2.28) indicates the proportion of true results, both positives and negatives, among the total number of cases examined. It is a statistical measure of how well a classification correctly identifies or except a condition. [89, 90, 28].

$$acc = \frac{tp + tn}{tp + tn + fp + fn} \quad (2.28)$$

1. True Positive (tp): indicates how many positive class elements predicted correctly.
2. True Negative (tn): indicates how many negative class elements predicted correctly.
3. False Positive (fp): indicates how many negative class elements are predicted as positive class elements.
4. False Negative (fn): indicates how many positive class elements are falsely predicted as negative class elements.

2.7.2 Classification Loss

The loss function is a metric, which evaluates the difference between the actual labels and the predicted labels. The basic loss function is the zero-one loss that counts how many

errors a hypothesis function induces in the set. The L1 loss (absolute error loss) is the absolute difference between the actual value calculated and a prediction for each sample in the data set. The total of all these loss values is called the cost function [91, 92, 28].

2.7.3 Confusion Matrix

The Confusion Matrix (Figure 2.12) is a matrix that shows the performance of the model by visualizing the ground-truth labels versus model predictions. It demonstrates the number of correct positive (true positives), correct negative (true negatives), false positive (false positives) and false negative (false negatives) predictions [89, 90, 93].

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 2.12: Confusion Matrix [93]

2.7.4 F1-Score, Recall and Precision

The Precision (Equation 2.29) measures the reliability of the model in classifying positive samples and is the ratio of true positives and total positive elements predicted. It focuses on the error, which occurs by rejecting a true null hypothesis H^0 [89, 90, 93].

$$precision = \frac{TP}{TP + FP} \quad (2.29)$$

The Recall (Equation 2.30) measures how many positive samples were correctly classified by the model. It only takes account the correct classification of all positive samples, but not whether a negative sample is classified as positive. It focuses on the error, which occurs by accepting a false null hypothesis H^0 [89, 90, 93].

$$recall = \frac{TP}{TP + FN} \quad (2.30)$$

Recall detects all positive samples without paying attention to whether negative samples are incorrectly classified as positive. Precision is sensitive to classifying a sample as

positive, including negative examples that have been incorrectly classified as positive [89, 90, 93].

The F1-Score (Equation 2.31) is a summary metric due to the harmonic mean of precision and recall and tells you how accurate the classifier is. A high score shows a good balance between recall and precision and performs well on imbalanced classification tasks. The harmonic mean (Equation 2.32) describes the reciprocal of the arithmetic mean of the reciprocals of the data and tends to emphasize the impact of small outliers while minimizing the impact of large outliers [89, 90, 93].

$$f_1 = \frac{2TP}{2TP + FP + FN} \quad (2.31)$$

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \frac{1}{x_3} \dots + \frac{1}{x_n}} \quad (2.32)$$

n ... the number of data points in the set

2.8 Deep learning frameworks and libraries

Deep learning frameworks provide building blocks for designing, training, and validating models. Popular frameworks include PyTorch, TensorFlow, or Keras, which in particular is actually a library that can be built on top of TensorFlow. A summary of each framework is listed in the following [94, 95, 96].

1. Keras is a high-level neural network library, which runs mainly on TensorFlow backend. CNTK or Theano is a respective possibility too. This library allows rapid and easy experimentation with the ability to detect an available GPU automatically and take advantage of it [94, 95].
2. The PyTorch machine learning framework is based on Torch. The advantage is its higher flexibility and the ability to make coding more manageable and increase processing speed [95].
3. TensorFlow is a deep learning software library to define, train and deploy machine learning networks. It offers distributed training support and scalable production deployment options [96].

3 State-Of-The-Art

In recent years, classification using Few-Shot Learning has attracted considerable attention in Vinyals et al. (2016); Snell et al. (2017); Finn et al. (2017); Ravi Larochelle (2017); Sung et al. (2018); Garcia Bruna (2018); Qi et al. (2018). A well-performing approach for few-shot classification is Meta-Learning, where transferable knowledge is extracted from a set of tasks and shared to prevent overfitting and improve the generalization. In this context there are approaches based on model initialization, such as methods Ravi & Larochelle (2017); Finn et al. (2017) or Metric Learning methods Vinyals et al. (2016); Snell et al. (2017); Sung et al. (2018) [15].

The above methods are used in computer vision, but not for time series data. Within the scope of this work, the Metric Learning methods are modified to be used to classify anomalies in time series data. The theory on which the implementation is based is presented below.

3.1 Meta Metric-Based Few-Shot Classification

3.1.1 Matching Network

One method to perform Meta Metric Learning in FSL is the matching network. The need for fine-tuning to adapt to new class types is not necessary any more, because the model learns a network that maps few labelled support sets and an unlabelled sample to its label (Figure 3.1) [19, 14].

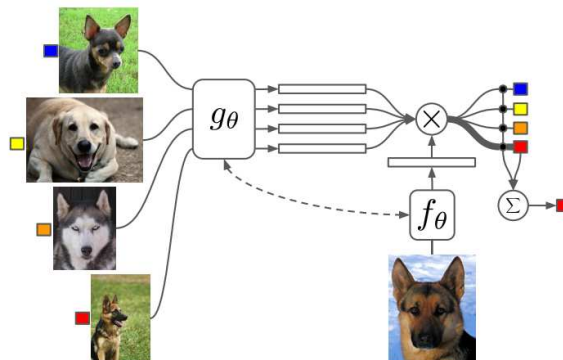


Figure 3.1: Matching network in the few-shot scenario [19]

The model defines a classifier function c_S for each support set S . More specifically, the small support set S of k examples of the labeled input $S = \{(x_i, y_i)\}_{i=1}^k$ is shown to the function $x_s(\hat{x})$. The given test sample \hat{x} is the distribution of the probability over the outputs \hat{y} . The method uses the parametric neural network, which is defined by $P(\hat{y}|\hat{x}, S')$. This network predicts the appropriate label distribution \hat{y} for each test sample \hat{x} , when given a novel support set of samples S' . The probability, the linear combination of the labels in the support set over \hat{y} is shown in Equation 3.1 [19].

$$P(\hat{y}|\hat{x}, S) = \sum_i^k = a(\hat{x}, x_i)y_i \quad (3.1)$$

The related label distribution from the support set $S = \{(x_i, y_i)\}_{i=1}^k$ and the inputs for this probability are x_i, y_i . The variable a is defined as the attention mechanism as it can be seen in Equation 3.2. It is a kernel on $\mathcal{X} \times \mathcal{X}$. [19].

$$a(\hat{x}, x_i) = \frac{e^{c(f(\hat{x}), g(x_i))}}{\sum_{j=1}^k e^{c(f(\hat{x}), g(x_j))}} \quad (3.2)$$

The classification a takes the softmax over the cosine distance c between the embeddings of \hat{x} and x_i . The class with the highest probability is selected as the predicted class [42, 14]. The matching network can be called as a weighted nearest-neighbor classifier applied within an embedding space [42]. To achieve the best accuracy through the classification, the embeddings f and g (Figure 3.1) act as a feature space \mathcal{X} .

The training process takes place in episodes, with each episode using mini-batches of support and query examples. Frequent updates of the stochastic gradient descent (SGD) are quite costly and can lead to worse results, since an approximation rather than the actual gradient is calculated at each step, leading to cost variations. Therefore, it is helpful to combine several, but not all examples into one update, which is called mini-batching. This means that n examples are taken as a subset of all data during one iteration and run updated. The batch size n can be set as a hyperparameter. This still results in multiple updates within an epoch, but not necessarily as many as SGD [97]. The idea is that each episode is designed to imitate the few-shot task by subsampling classes as well as data points. Further gradient updates are performed on episodes with c classes randomly sampled from the base label set and k samples for each class. The use of Meta-Learning with episodic training, makes the training issue more faithful to the test framework and thus improves generalization. Additionally it brings the training and test distributions together and alleviates the problem of overfitting. The testing is carried out through the presentation with a few samples of a new unlabeled task [19, 42, 14].

3.1.2 Prototypical Network

Another network that has performed efficiently in few-shot classification is the prototypical network, which conducts classification by learning the distance distribution among relations. Because of their simpler inductive bias, it addresses overfitting as key issue of Few-Shot Learning [42, 98, 99].

In the prototypical network exists an embedding, where the points cluster around one centroid embedding for each class. It learns a non-linear mapping of the input into an embedding space with the use of a neural network [42]. The prototypical network averages the vector of a few support instances as the class prototype. This vector is the mean vector of the embedded support instances belonging to its class [98]. It compares the distance between all prototype vectors and a target query vector by performing nearest neighbor classification with learned class features. These prototypes c_k are the mean of the embedded support samples for each class k in few-shot as Figure 3.2 shows [18, 98, 42].

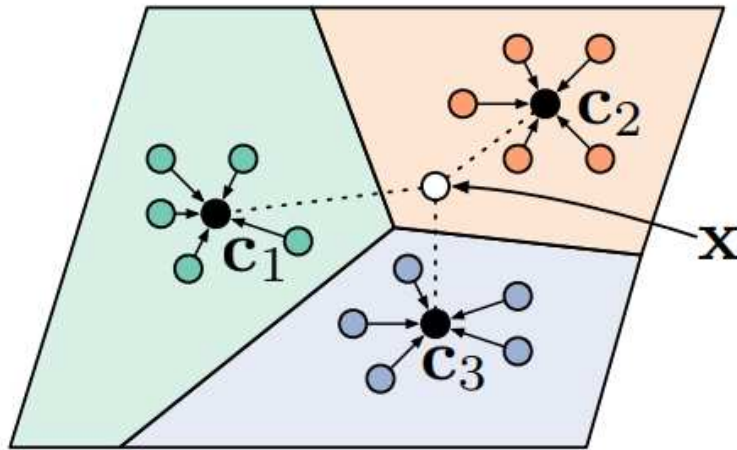


Figure 3.2: Prototypical networks in the few-shot scenario [42].

Due to the few-shot setting, only a support set of N labeled samples $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$ is given. Each $x_i \in \mathbb{R}^D$ is the D -dimensional feature vector of a sample. The related label of this sample is $y_i \in \{1, \dots, K\}$. The set of labeled samples with class k is called S_k [42].

There is a support and a query set for each possible class. A prototypical network computes a prototypical representation of each class k with a M -dimensional representation $c_k \in \mathbb{R}^M$ by an embedding function $f_\phi : \mathbb{R}^D \leftarrow \mathbb{R}^M$ with learnable parameters ϕ . c_k estimates the prototypes for each possible class k .

$$c_1, c_2, c_3 \quad \longrightarrow \quad c_k \quad \text{are the class prototypes}$$

This estimation works with a forward pass in our network $f_\phi(x_i)$. If it is a 5-Shot Learning, it has given 5 x_i and the division by 5 (N_C). The prototype, which is the mean vectors

of the embedded support points belonging to its class, is created (Equation 3.3) [42].

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\phi(x_i) \quad (3.3)$$

The distribution over classes for a given query input x is a softmax over the inverse of distances between the query data embedding and prototype vectors. The Equation 3.4 represents the probability [42, 100].

$$p_\phi(y = k|z) = \frac{\exp(-d(f_\phi(x), c_k))}{\sum_{k'} \exp(-d(f_\phi(x), c_{k'}))} \quad (3.4)$$

y is the output of the network, k is the true label and x is the input. The classifier is the probability of the output belonging to a specific true class given the input. The distance gives more weight to a point further apart. By taking the negative of the distance, the distance is converted into a similarity. The higher the value is, the more close the point is to the specified centroid. By applying a standard softmax, these distances are turned inside probability distributions. The sum of the values is 1. The higher the value of a specific class is, the higher the probability is that the input x_Q belongs to that class [42].

The learning is done by minimizing the negative log-probability with the loss function shown in Equation 3.5. The loss is initialized to zero and then it is updated with a certain number of iterations(N_C) [42].

$$J \leftarrow J + \frac{1}{N_C N_Q} \left[d(f_\phi(x), c_k) + \log \sum_{k'} \exp(-d(f_\phi(x), c_{k'})) \right] \quad (3.5)$$

Instead of comparing the query samples with all the support samples, Prototypical Network only compares the query sample with the class prototype (or mean class embedding). The key assumption is that there exists an embedding for each class where samples cluster around a single prototypical representation. The advantage of the prototypical network is that the network has fewer or equal class prototypes than the number of samples in the support set and therefore the amount of required pairwise comparison decreases. This saves computational costs [101, 42]. The following pseudocode 3 shows the training procedure of a prototypical network to produce the loss $J(\phi)$ [42].

Algorithm 3 Algorithm of the Prototypical Network [42]

Input: Training class set \mathcal{D}_{train} **Steps:**

```

1: for  $k$  in  $(1, \dots, N_C)$  do
2:    $S_k \leftarrow$  random Sample  $(D_{V_k}, N_s)$ 
3:    $Q_k \leftarrow$  random Sample  $(D_{V_k}, \mathcal{S}_k, N_s)$ 
4:    $c_k \leftarrow \frac{1}{|N_C|} \sum_{(x_i, y_i) \in S_k} f_\phi(x_i)$ 
5: end for
6:  $J \leftarrow 0$ 
7: for  $k$  in  $(1, \dots, N_C)$  do
8:   for  $(x, y)$  in  $Q_k$  do
9:
10:     $J \leftarrow J + \frac{1}{N_C N_Q} [d(f_\phi(x), c_k) + \log \sum_{k'} \exp(-d(f_\phi(x), c_{k'}))]$ 
11:   end for
12: end for

```

The training set $\mathcal{D}_{train} = \{(x_1, y_1), \dots, (x_N, y_N)\}$, where each $y_i \in \{1, \dots, K\}$ is the input of the algorithm. \mathcal{D}_k is the subset of \mathcal{D} containing all elements (x_i, y_i) so that $y_i = k$. The output of the code is the loss \mathcal{J} for a randomly generated training episode.

The Prototypical Network and the Matching Network in the few-shot classification is equal in the one-shot scenario. The Matching Network merges the embedding and classification to form an end-to-end differentiable weighted nearest neighbors classifier with the cosine distance. The weights are the pairwise similarity between each query sample and each support sample [19]. The Prototypical Network first computes a prototype for each class and then combines these centroids with the query samples using the metric distance of Euclidean distance [101, 42].

3.1.3 Relation Network

Another method for few-shot classification is the end-to-end trained relation network (Figure 3.3). The network performs the classification in two stages by learning to compare query samples against few labeled samples.

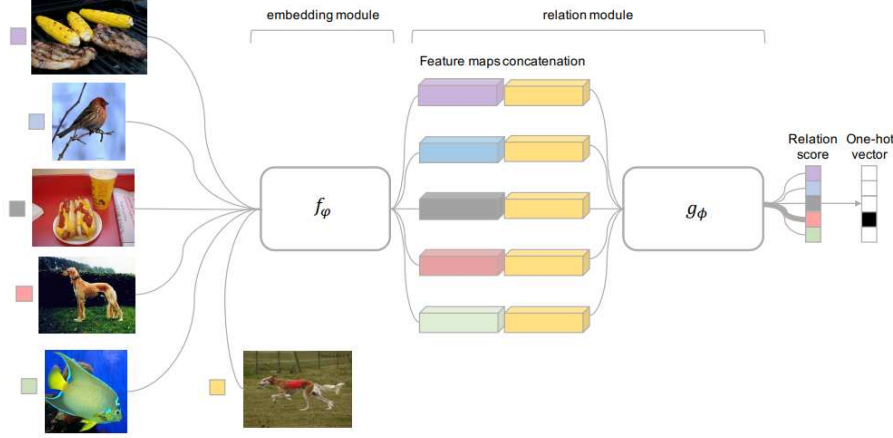


Figure 3.3: Relation network in the few-shot scenario [43].

The first stage is an embedding module f_ϕ , which compresses and creates a sort of concentration of different layers and generates representations of the query and training samples. Since the aim is to compare the query data with all the classes, they concatenate the query feature with all the other encoded classes. The meta-training proceeds to learn a deep distance metric to differentiate few amount of samples in the episodes. Each episode reproduces the few-shot setting. Once this feature maps concatenation is created by training, they are passed through another neural network, which is the second stage. This second branch is the classifier, represented by the relation network g_ϕ . It analyzes the given layers and N generates relation scores $r_{i,j}$ for the relation between one query input x_j and training sample set tasks x_i . The relation score for each query x_j is shown in Equation 3.6.

$$r_{i,j} = g_\Phi(N(f_{\phi_1}(v_c), f_{\phi_2}(x_j))) \quad i = 1, 2, \dots, N \quad (3.6)$$

In general, the output is a score between $[0, 1]$, which represents the similarity between x_i and x_j . In the following equation the relation score is compared to the labelled one-hot vector. This training is done with the mean square error loss, regressing the relation score $r_{i,j}$ to the ground truth (Equation 3.7).

$$\phi, \Phi \leftarrow \arg \min_{\phi, \Phi} \sum_{i=1}^m \sum_{j=1}^n (r_{i,j} - 1(y_1 == y_j))^2 \quad (3.7)$$

The indicator function is equal to 1 if the $y_i == y_j$ is satisfied, otherwise it would be equal to zero. The loss function minimizes a specific loss such that these parameters of the neural network are moved towards to a minimum [43].

RelationNet also learns a metric for comparison of the embedding. The idea is quite similar to Prototypical Network, because it averages the embeddings for each class together to a single prototype, but it replaces distance with a learnable relation module [43, 15, 17].

4 Methodology

This chapter addresses the various steps involved in creating an efficient labeling system for noisy industrial data using a few labeled samples with DL Few-Shot classification methods.

In addition to a training set and a test set, a support set and a query set are required for each new episodic training in the learning phase (Section 2.3.7). In the definition phase of the embedding, a discriminative feature space is created for each set. Therefore, the input of the support set or the query set is fed into a deep neural network that generates the embedding. Further the network can be pre-trained and improved by a feature extractor, which is not evaluated in this work. The metric is learned on the embeddings between the support and the query set. The goal of the learning process is to show the similarity and separate the differences. If a sample is similar to another sample of the same class, a feature representation is derived from the sample. Conversely, if a sample is similar to another class, it should be as dissimilar as possible. The network can be trained with the use of a loss function based on the distance between the representation of pairs, which is assimilated into the classification phase [9, 14, 54, 26, 94].

4.1 Pipeline of the Few-Shot Classifications

The flow chart Diagram 4.1 represents the pipeline of the experiments chronologically. Subsequently, the critical factors for the flow are pointed out and discussed further.

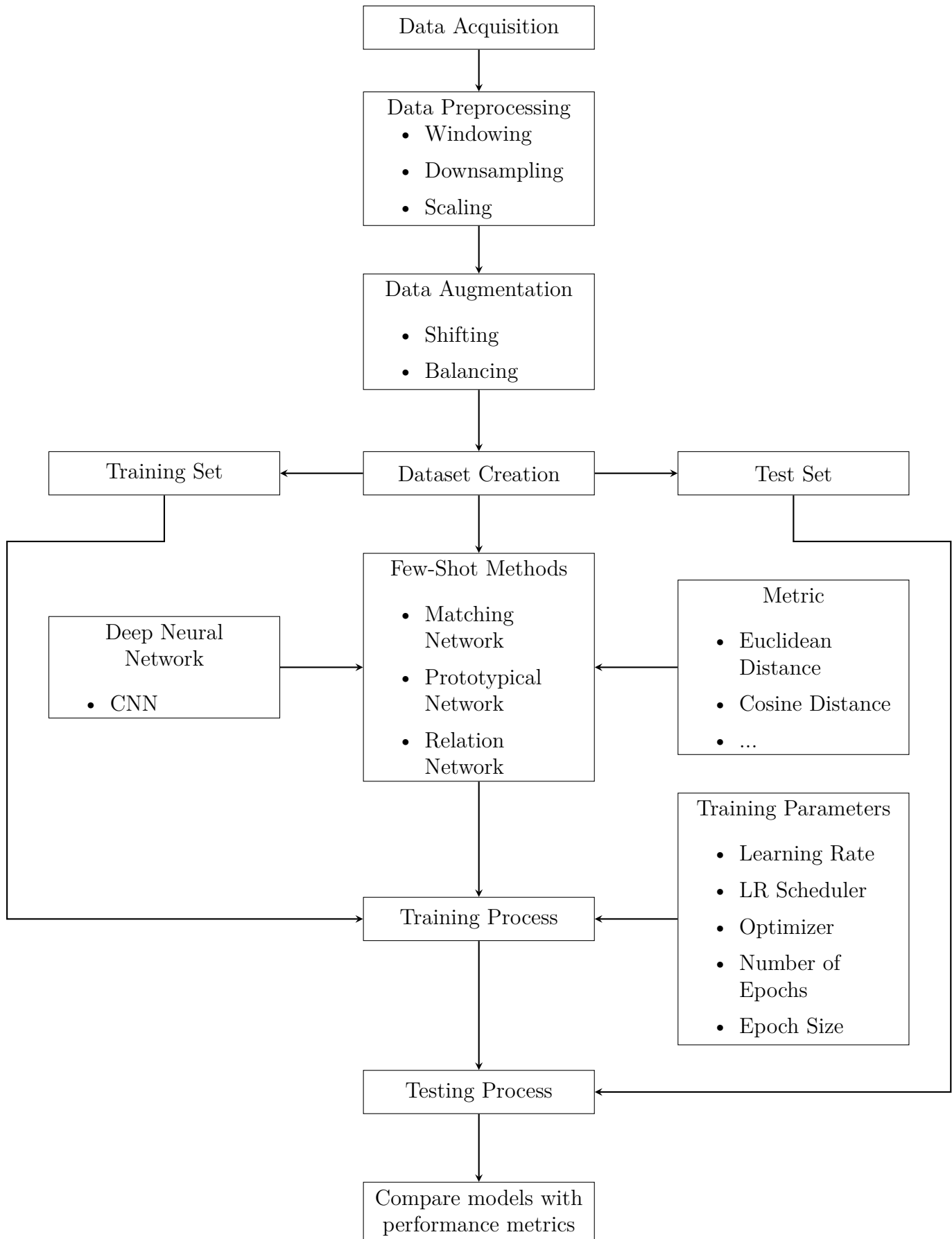


Figure 4.1: The pipeline of the Few-Shot Classification

4.2 Data

The data used for this work have been collected within two different time periods, each over 5 months. During these periods, data were taken from different milling machines of the same type. While there were many good examples in the collection, few anomalies were noted. As a result, the data set is imbalanced, having many examples for one class but few for the other. The collection of the data, as well as the subsequent base code for preprocessing, has already been described in earlier work [102], and will not be further explained here. The respective parameters for the correct preprocessing are evaluated by myself and further research [103].

4.2.1 Data Preprocessing

Data preprocessing techniques have been used to reduce the complexity of the data and to detect or remove irrelevant and noisy parts from the data to improve the overall quality of the data. For the collected data three different preprocessing methods are used in this work: Downsampling, Scaling and Windowing [104, 105].

It is important to note that these described techniques are mainly used for image processing and not for time series data. In the context of this work, these techniques produce good results for the used data. However, it should be noted that this cannot necessarily be generalized and still needs to be investigated further.

Downsampling is an efficient method for speeding up classification time. It reduces the amount of data in a specific sequence according to a specified downsample factor, which is selected by the user in the preprocessing task [106, 107]. The downsampling factor is set to 2 for all experiments after applying an anti-aliasing filter. The factor 2 means that 2 points describe each upward and downward movement of the curve. Although the factor is low, the curve can be represented sufficiently [108].

The scaling is done in a function [102] designed specifically for the data in this project. In the data there is a difference between the maximum and the minimum. Therefore, normalization of the value magnitudes is carried out and scaled to appreciably low values. Standardization assumes that the observations correspond to a Gaussian distribution with a well-formed mean and standard deviation. The most common approaches are min-max normalization and z-score normalization. In the scope of the thesis, the z-score (Equation 4.1) is used because it offers a better normalization for industrial data containing outliers [105, 109, 102]. v is the old feature value, v' is the new one, and σ is the standard deviation of v .

$$v' = \frac{v - \text{mean}(v)}{\sigma} \quad (4.1)$$

Windowing is a common preprocessing technique in computer vision that was also used here. It is important to note that while this technique gives good results in this context, it should not be generalized to time series data. In this work the data is split into smaller patches by dividing the time series into smaller numerous windows, which is exemplary shown in Figure 4.2. The algorithm picks a random sample of a user-selected size from the full set of the whole sample. The window size should be neither too small, thus containing less information, nor too large, which can lead to overloading in the neural network [110, 111]. The window size for all experiments is set to 4096, has already been established in research [103] and was not part of this work.

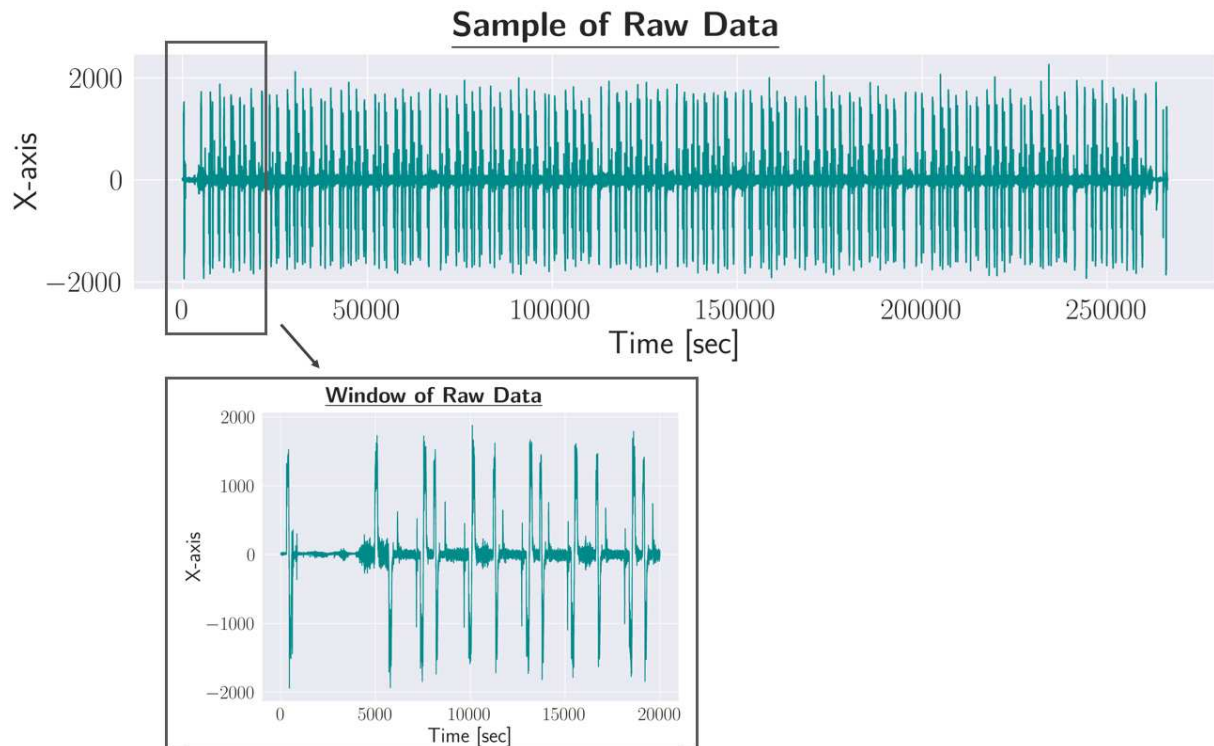


Figure 4.2: Shows the technique of windowing a data sample. Windows are extracted from the data sample to create multiple samples from a single sample [103]. In general this method is used for computer vision models and not for time series data. Nevertheless this technique generates valuable results and can be used for the data set

4.2.2 Data Augmentation

Data augmentation is a technique to increase the diversity of the training set by applying random, but realistic transformations. This is a way to proceed generating new data from the current ones, where those new ones are variants of the data included in the initial databases. One approach is explained in the following.

Shifting is a sliding window technique that divides the time series data (Figure 4.3). The

time series is not only divided into individual segments, but the number of samples can be increased by adjusting the overlap rate, which in this work is called the shift distance [112]. The parameter for the shifting distance is set to 2096 [103] for all experiments.

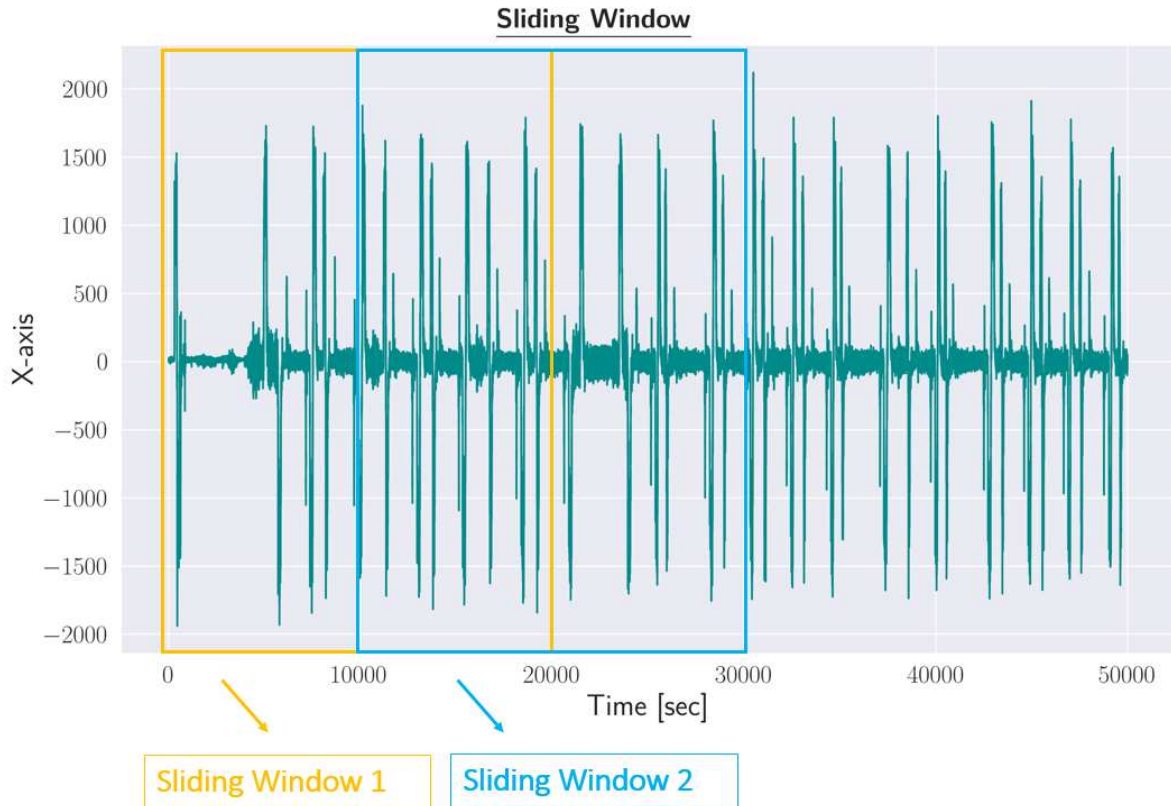


Figure 4.3: Shows the technique of the sliding window mechanism. Windows are created at a distance of a certain overlap factor [103]. The window size must be selected beforehand to create segments with the same size. The higher the overlap factor, the more window segments can be created from the existing time series data.

In the shift technique, the balancing factor can reduce the proportion of imbalanced data. This makes it possible to generate x more data for one class than others and thus counteract the imbalance [113]. In this thesis, a factor of 2 was assumed so that the balance goes in the direction of bad examples. A factor of 2 means that the overlap rate for defect data is 2 times higher than for good data, so more defect samples are generated to achieve a more balanced data set.

In the following Table 4.1, the selected parameters are given.

Window Size	Downsampling	Scaling	Shifting Distance	Balancing Factor
4096	2	True	2096	2

Table 4.1: The chosen preprocessing and augmentation parameters to generate the final data set for the experiments

Finally, it should be noted that although I did the preprocessing, both the functions and the parameters were not researched and evaluated as part of this thesis; I refer only to the results of the master's theses [102, 103]. Figures 4.4 and 4.5 illustrate the difference between the raw data and the preprocessed data. The characteristics machine, process and period time of the recorded data are identical. After preprocessing, the data is visualized, where a time window was taken out due to the used technique windowing.

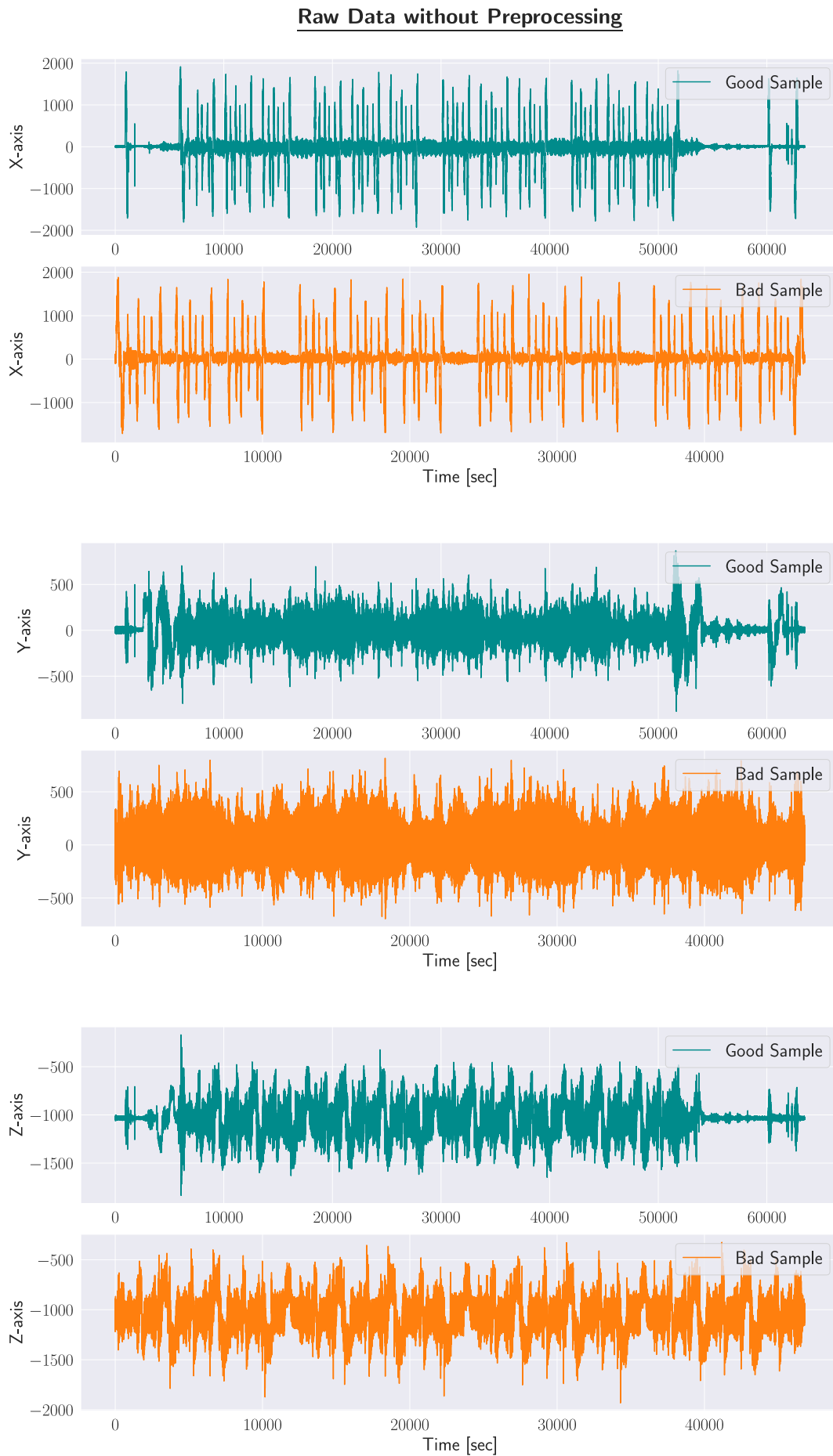


Figure 4.4: The raw time series data of 3 different orientation X-Y-Z axis.

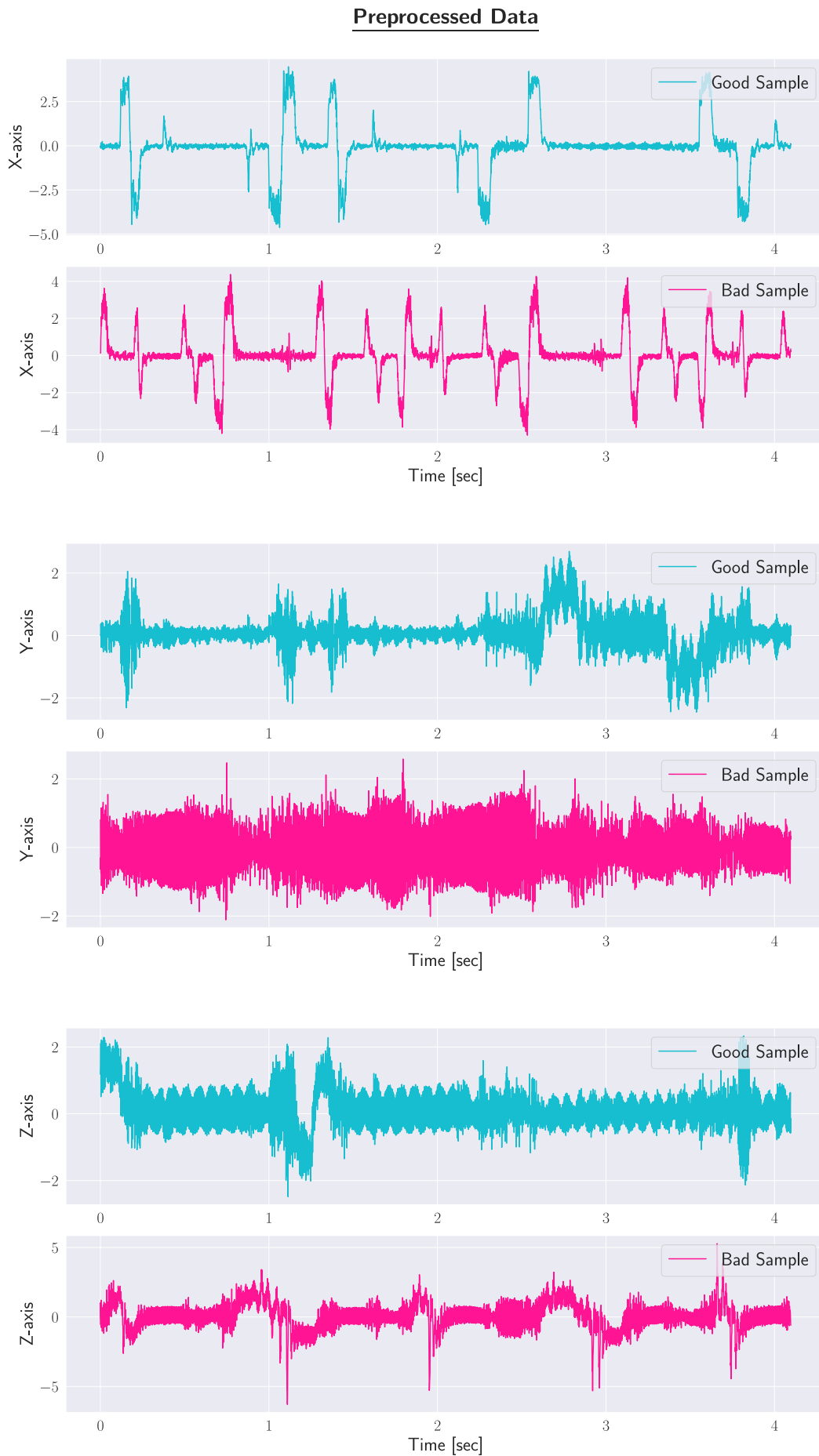


Figure 4.5: The preprocessed time series data of 3 different orientation X-Y-Z axis.

4.2.3 Dataset - Training Set & Test Set

The data is divided into a training and a test set for the ML classification task. To ensure uniform evaluation and comparison, the data sets for each experiment are the same. Regarding the Few-Shot Learning technique, the aim is to train on as few samples as possible. Therefore, in contrast to most classification tasks, the training set is small, whereas the test set is large. The purpose is that the network achieves a good performance even with fewer samples.

The dataset is collected from three different milling machines (M1, M2, M3). Each machine can operate 15 different operation processes, which are referred to as OP00-OP14 in the following work. In the context of the work, the data is divided process by process, whereas it contains examples from all three machines. The data is split process-wise but not machine-wise to make the training more challenging. Additionally, there is variability in the set due to the time period of the data collection, around 2.5 years. The number of samples for each process and machine highly varies, making the data imbalanced. The models for the training set are user-selected to achieve an overview of the process and machines, although there are just a few samples. In the case of few samples, e.g. 6, it is impossible to consider examples from each process from every machine. A more detailed list of the datasets can be found in the appendix.

The number of examples chosen for each data set was determined, with a gradation towards a few examples. It should be considered that each training is performed with data samples from the operation process OP00-OP04. However, there are two scenarios when testing: In the first scenario all operation processes from OP00-OP14 are evaluated. In the second one only the data from OP05-OP14, which the training set has never seen, are used. This makes the network evaluation more challenging if the testing is done on unseen samples.

In this work, the classification is based on two classes, good and bad samples. The labeling of good and bad samples is predefined in another work and will not be further discussed in the context of this thesis. Before the final data sets were selected and the evaluation for the thesis was performed, good and bad examples were also divided by procedure, resulting in similarly good results. However, only good and bad examples are classified independently of the processes to ensure good comparability and discussion of the results and not to address an additional issue.

In the following Table 4.2, the splitting of the created dataset is shown.

Data Set	Classes	Train Process	Test Process	Train good Samples	Train bad Samples	Test good Samples	Test bad Samples
DS1	good/bad	OP 00-04	OP 00-14	70 pc.	21 pc.	1632 pc.	70 pc.
DS2				35 pc.	11 pc.	1632 pc.	70 pc.
DS3				17 pc.	6 pc.	1632 pc.	70 pc.
DS4			OP 05-14	70 pc.	21 pc.	1902 pc.	49 pc.
DS5				35 pc.	11 pc.	1902 pc.	49 pc.
DS6				17 pc.	6 pc.	1902 pc.	49 pc.

Table 4.2: The table shows the created datasets for the evaluation of the FSL-Methods with two classes (good and bad). The training set consist of data from operation 00 to 04 (5 different process). The test set consist of data from operation processes 00-14 or 05-14. The number of the training data decreases in the dataset.

4.3 Few-Shot Classification

The Few-Shot learning and the subsequent classification occur in two stages: Training and Testing.

4.3.1 Training Stage

In this section, the setting of the optimizer, the learning rate, the number of epochs, epoch size, and the number of samples for each support and query set are described. To find the optimal setting of the combination of all the hyperparameters (Section 2.5), several experiments are required. In this evaluation, the values vary within a specific range, with the final parameters as follows.

The model is trained with an initial learning rate of 1×10^{-3} , applied with an additional scheduler to adjust the learning rate as the training progress to a pre-defined schedule. This can be performed by learning rate schedules, including time-based decay, exponential decay, or step decay. In this approach, the STEPLR is applied, which decays the learning rate of each parameter by every step size epoch. The step size is set to 5×10^{-1} [114].

The Adaptive Moment Estimation (Adam) is used for optimization, a general DL training application. It improves the training by adapting the learning rate to different parameters automatically, based on the statistics of the gradient. The aim is to handle sparse gradients on noisy problems [115, 116].

The optimal number of epochs depends on the size of the training and the support/query set.

To avoid overfitting, the number of epochs has to be reduced according to the decreasing number of samples in the training set. With a small number of samples in the training set, as in this case, an epoch number of 6 is a good parameter for this use case. The goal was to select an epoch number that remains the same for each experiment and does not require as much computation time, but still allows an efficient training.

Each epoch is composed of several batches, which represents the epoch size. The epoch size is set for all experiments to 1000.

In each training epoch, a subset is randomly selected.

Depending on the application, these subsets are called support sets or query sets. The size of the subsets can be varied, which will be discussed later.

All parameters described were used consistently for each experiment to allow better comparison of results. Table 4.3 gives an overview.

Learning Rate	1×10^{-3}
LR Step Size	5×10^{-1}
Optimizer	ADAM
Number of Epochs	6
Epoch Size	1000

Table 4.3: The table shows the chosen training parameters.

In the context of the work, the classification is done on 2 classes (good and bad), which is known in the few-shot term as 2-way.

In addition to the fixed parameters, the number of shots is varied in the support and query set. A 2-way k -shot classification is proposed, which represents 2 number of classes and k number of shots in the support set, and the query set in each epoch. In addition, experiments were conducted with a different number of examples in the support and query set. Table 4.4 shows the different sizes of the support and query set.

n-way Classes	k-shot	
	Support Set	Query Set
2	3	3
2	3	5
2	5	3
2	5	5
2	5	7
2	7	5
2	7	7

Table 4.4: The table shows the selected values for the n -way k -shot classification. Each data set contains 2 classes (good and bad). The shots for the support and query set vary between 3 and 7.

Due to the episodic sampling approach, the class imbalance in the data samples is reduced by randomly selected class pairings with an equal number of class samples.

4.4 Testing Stage

The training set consists of several epochs with a number of mini-batches. In contrast, the testing set is evaluated episodically on the mini-batches in the test episodes.

The testing parameters and the size of the support and query set are equal; only the number of the test episodes is set to 1000.

4.4.1 Embedding Network Structure

The deep neural network architecture to form the embedding for the Few-Shot classification is a Convolutional Neural Network (CNN). A simple CNN was chosen as the embedding function to evaluate the best performing Few-Shot Learning method and metric.

The model consists of one convolutional block, consisting of a convolutional layer with ReLU activation as well as a max-pooling layer. The architecture of the CNN model is shown in table 4.5 and is generated by several trial-and-error experiments.

Layer	Feature Map	Size	Kernel Size	Stride	Activation
Input Data	3	$20 \times 3 \times 4096$	/	/	/
Convolutional	32	$20 \times 32 \times 4096$	3	2	ReLU
Maxpool	32	$20 \times 32 \times 2047$	3	2	/
Convolutional	64	$20 \times 64 \times 2047$	3	2	ReLU
Maxpool	64	$20 \times 64 \times 1023$	3	2	/
Convolutional	64	$20 \times 128 \times 1023$	3	2	ReLU
Maxpool	3	$20 \times 128 \times 511$	3	2	/
Convolutional	32	$20 \times 32 \times 511$	3	2	ReLU
Maxpool	3	$20 \times 32 \times 255$	3	2	/
Flattening	32	20×8160	/	/	/

Table 4.5: The table shows the architecture of the Convolutional Neural Network(CNN). The properties of the different layers are described in more detail in section 2.5.

The same CNN is used for both support and query sets in the training and test process.

4.4.2 Models and Metrics of Few-Shot Learning

The performance of the classifications depends mainly on the chosen Few-Shot Learning method and the distance/similarity measure. Three different learning methods are chosen for the experiments, described in Section 3.2 in more detail.

In addition, different distance/similarity measures explained in Section 3.1 are used for the classification task.

The challenge was to make the chosen networks and metrics applicable to the noisy industrial data, due to different dimensions.

During the experiments, some metrics and the relation network were found to be inappropriate for the process data because they either had poor evaluation performances, or did not train at all. Therefore, the training was not finalized on these metrics.

4.5 Overview of the Experiments

The final performed experiments are listed in Table 4.6.

Experiment	Dataset	Model	Metric	Shots (Support & Query)	Epoch
Exp1	DS1	PN	L2	5 & 5	6
Exp2	DS2	PN	L2	5 & 5	6
Exp3	DS3	PN	L2	5 & 5	6
Exp4	DS4	PN	L2	5 & 5	6
Exp5	DS5	PN	L2	5 & 5	6
Exp6	DS6	PN	L2	5 & 5	6
Exp7	DS1	MN	L2	5 & 5	6
Exp8	DS2	MN	L2	5 & 5	6
Exp9	DS3	MN	L2	5 & 5	6
Exp10	DS4	MN	L2	5 & 5	6
Exp11	DS5	MN	L2	5 & 5	6
Exp12	DS6	MN	L2	5 & 5	6
Exp13	DS1	PN	L2	3 & 5	6
Exp14	DS1	PN	L2	7 & 5	6
Exp15	DS1	PN	L2	3 & 3	6
Exp16	DS1	PN	L2	7 & 7	6
Exp17	DS1	PN	L2	5 & 3	6
Exp18	DS1	PN	L2	5 & 7	6
Exp19	DS1	PN	Cosine	5 & 5	6

Experiment	Dataset	Model	Metric	Shots (Support & Query)	Epoch
Exp20	DS1	PN	DOT	5 & 5	6
Exp21	DS1	PN	DTW	5 & 5	6
Exp22	DS1	MN	Cosine	5 & 5	6
Exp23	DS1	MN	DOT	5 & 5	6
Exp24	DS1	MN	DTW	5 & 5	6
Exp25	DS1	MN	L2	3 & 3	6
Exp26	DS1	MN	L2	7 & 7	6
Exp27	DS6	PN	L2	5 & 5	6

Table 4.6: The table listed the different experiments on which the FSL method is evaluated in scope of this work.

4.6 Library Details

The whole pipeline was written in Python and supported by the machine learning framework PyTorch and the library Keras (Section 2.8). For the main part, the training and testing, PyTorch was chosen because of its flexibility and low abstraction, which allows better adaptation of parameters during the learning process. The preprocessing part is done with Keras, because some used functions, which were not programmed within the scope of this work, are based on Keras [94, 95, 96].

5 Results and Discussion

In this chapter, FSL method and Metric Learning method are evaluated for the given time series data. In the first section (5.2.1) we evaluate the impact of the number of samples per class. In section 5.2.2 the number of shots per support set is changed. In addition, the network method is changed in section 5.2.3. Finally, in section 5.2.4, the classification is done on different distance/similarity measures.

5.1 Results

To determine the performance of the models, several evaluation metrics (Test Accuracy, Test Loss, F1-Score, Precision, Recall, Confusion Matrix), which are described in chapter 2.6, are used. The recall and precision metrics can be interesting to highlight and understand the model's behaviour better. Depending on the final use case, the focus is on a high recall or precision. While precision refers to the percentage of relevant results, recall refers to the percentage of all relevant results that were correctly classified by an algorithm. A high recall is usually preferred to detect and filter any anomaly, focusing on anomaly detection.

Considering two metrics and given the unbalanced dataset in terms of good and bad examples, the F1 measure is used throughout the evaluation of the different models. In conjunction with accuracy, the methods can be evaluated efficiently. The parameters for the composition of the experiments would go beyond the scope of the table. They are described in the methodology chapter Table 4.5. In subsequent, I will mention only the experiment numbers. Finally, the following results in Table 5.1 are the average of all the performance metrics over each epoch.

Experiment	Train Loss	Train Accuracy	Test Loss	Test Accuracy	F1 Score	Precision	Recall
Exp1	0.0004	1.0000	0.5989	0.9412	0.9392	0.9548	0.9376
Exp2	0.0003	0.9998	0.8149	0.8870	0.8817	0.9120	0.8808
Exp3	0.0000	1.0000	1.2798	0.8061	0.8009	0.8358	0.8150
Exp4	0.0004	1.0000	0.5825	0.9422	0.9413	0.9524	0.9428
Exp5	0.0003	0.9998	0.7439	0.8928	0.8893	0.9119	0.8926
Exp6	0.0000	1.0000	1.2922	0.7758	0.7640	0.8059	0.7794
Exp7	0.0076	0.9970	0.7395	0.8879	0.8735	0.8937	0.8778
Exp8	0.0002	1.0000	1.5152	0.8121	0.7957	0.8355	0.8026
Exp9	0.0001	1.0000	2.4469	0.7399	0.7181	0.7755	0.7454
Exp10	0.0076	0.9970	0.7570	0.8920	0.8865	0.9028	0.8938
Exp11	0.0002	1.0000	1.5813	0.7994	0.7840	0.8221	0.7968
Exp12	0.0001	1.0000	2.6403	0.6882	0.6600	0.7131	0.7036
Exp13	0.0017	0.9994	0.7725	0.9303	0.9293	0.9413	0.9340
Exp14	0.0003	1.0000	0.6176	0.9490	0.9482	0.9558	0.9530
Exp15	0.0020	0.9993	0.5503	0.9402	0.9375	0.9555	0.9413
Exp16	0.0001	1.0000	0.8983	0.9376	0.9370	0.9456	0.9409
Exp17	0.0018	0.9995	0.6066	0.9388	0.9368	0.9549	0.9387
Exp18	0.0007	0.9997	0.7045	0.9437	0.9432	0.9490	0.9474
Exp19	0.3407	1.0000	0.4173	0.9531	0.9521	0.9619	0.9530
Exp20	0.0026	0.9992	0.7776	0.9351	0.9334	0.9482	0.9362
Exp21	2.5340	0.8548	2.8910	0.8303	0.8345	0.8268	0.8914
Exp22	0.3627	0.9975	0.4603	0.9176	0.9147	0.9274	0.9186
Exp23	0.0047	0.9988	0.7948	0.8870	0.8850	0.8979	0.8930
Exp24	8.3297	0.7235	8.6038	0.6908	0.6460	0.6647	0.7292
Exp25	0.0314	0.9898	0.5305	0.8812	0.8707	0.8892	0.8843
Exp26	0.0048	0.9984	1.0743	0.8642	0.8556	0.8789	0.8610
Exp27	0.0004	1.0000	0.7834	0.9218	0.9190	0.9270	0.9216

Table 5.1: The table shows the results of the performance metric for the experiments listed in Table 4.6.

5.2 Discussion

This section provides a detailed comparison and evaluation of the results. There are four types of approaches in which different parameters have been varied, which will be explained in more detail in the following sections. However, the changes must not be considered individually, as it is always an interaction of several variants. The same training

and testing sets are used in the following experiments (Table 4.2).

5.2.1 Dataset Discussion

The two networks are evaluated on datasets where the number of examples in the set was reduced. Dataset 1 is the largest one with 91 pieces (pc.), followed by DS2 with 46 pc. and DS3 with 23 pc. for training samples. As shown in Diagrams 5.1 and 5.2, the model trains faster with few examples (Exp. 3) than many (Exp. 1). This applies to both the PN and the MN, whereby the PN reaches 100 % after fewer epochs. The training is completed in about 6 epochs after the testing takes place. For the evaluation in Figures 5.1 and 5.2, experiments 1, 2, 3, 7, 8 and 9 are used (Table 5.1).

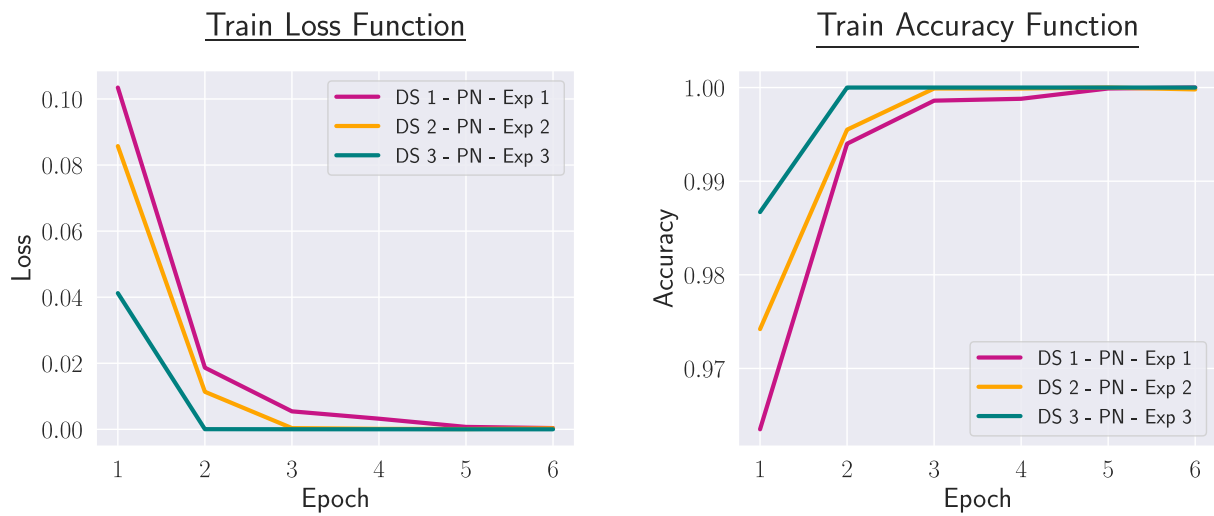


Figure 5.1: Train loss and accuracy of the Prototypical Network trained on datasets 1, 2 and 3. The parameters for the experiments are listed in detail in Table 4.6.

The training loss for exp 3 in PN is $1.3e^{-5}$, while it is for exp 9 in MN $5e^{-5}$.

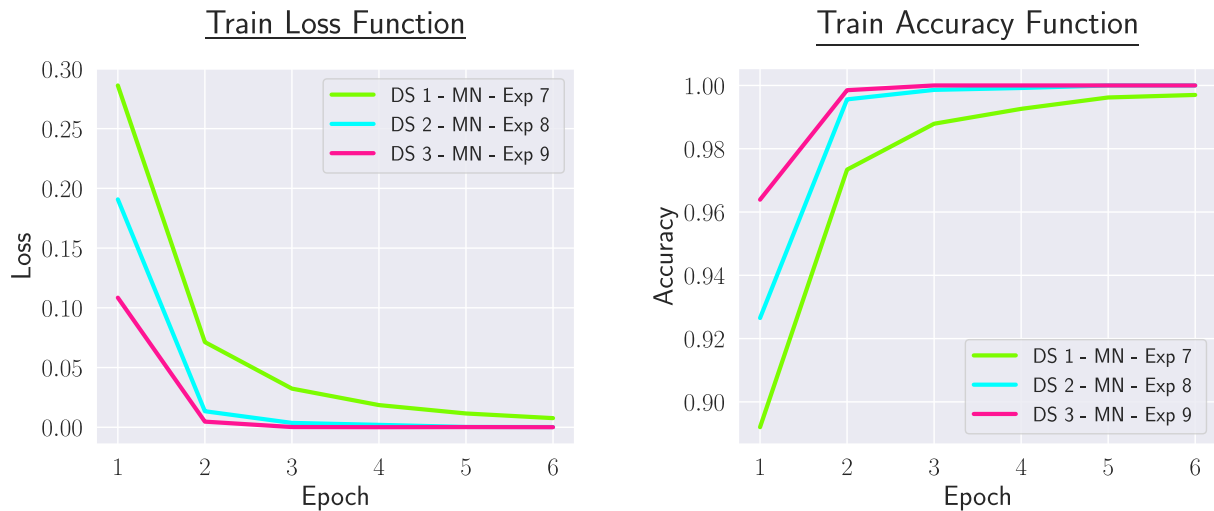


Figure 5.2: Train loss and accuracy of the Matching Network trained on datasets 1, 2 and 3. The parameters for the experiments are listed in detail in Table 4.6.

The performance decreases with fewer examples, as shown in Diagram 5.3. However, good results with a F1-score of 80.09 % can be achieved with 17 good and 6 bad examples (dataset 3). The prototypical network performs better than the matching network on all performance metrics (Test Accuracy, F1-Score, Precision, Recall), as Figure 5.3 shows.

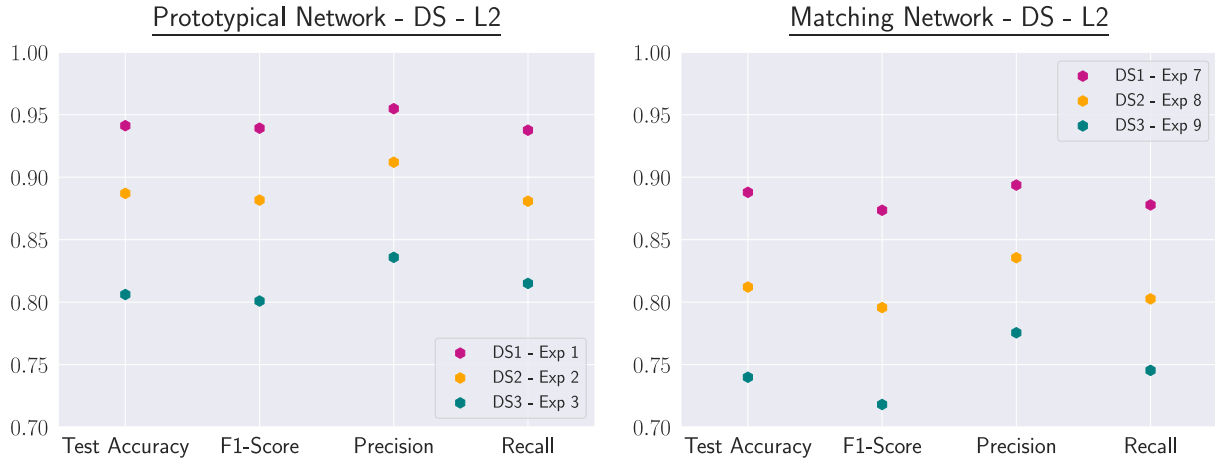


Figure 5.3: Metrics for the performance evaluation of the Prototypical and Matching Network. The parameters for the experiments are listed in detail in Table 4.6.

In datasets 4,5 and 6, the training set and the test set consist of completely different samples. The performance of experiment 4, 5, 6 approximately has the same performance as experiments 1, 2, 3. The exact composition of the data sets can be found in Table 4.2. For evaluation 5.4, experiments 1, 2, 3, 4, 5, 6 for PN and 6, 7, 8, 9, 10, 11 for MN are used (Table 5.1).

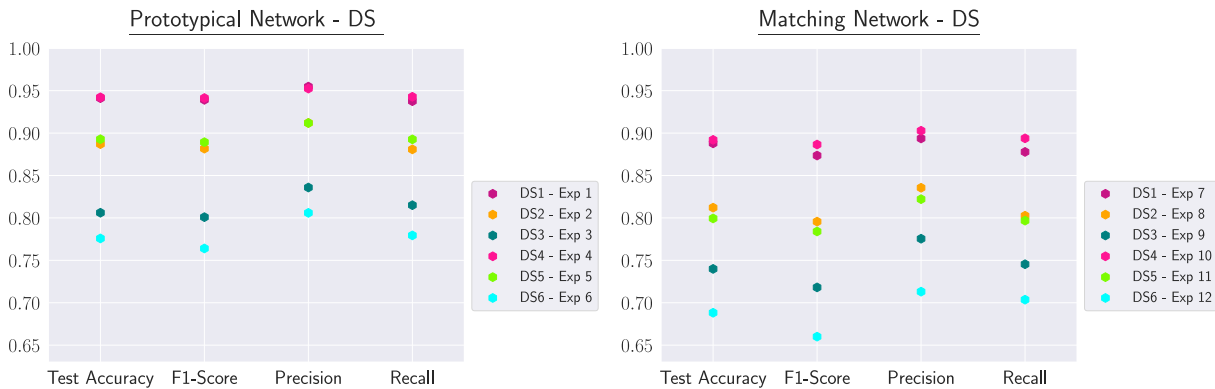


Figure 5.4: Metrics for the performance evaluation of the Prototypical and Matching Network for all datasets. The parameters for the experiments are listed in detail in Table 4.6.

The error rate in the confusion matrix (Figure 5.5) increases proportionally to a lower number of samples. Even in dataset 1, with the initial situation of 70 good and 21 bad examples, the F1-score is 80.09%. It should be noted that the number of samples is increased by data augmentation and preprocessing, which leads to better performance.

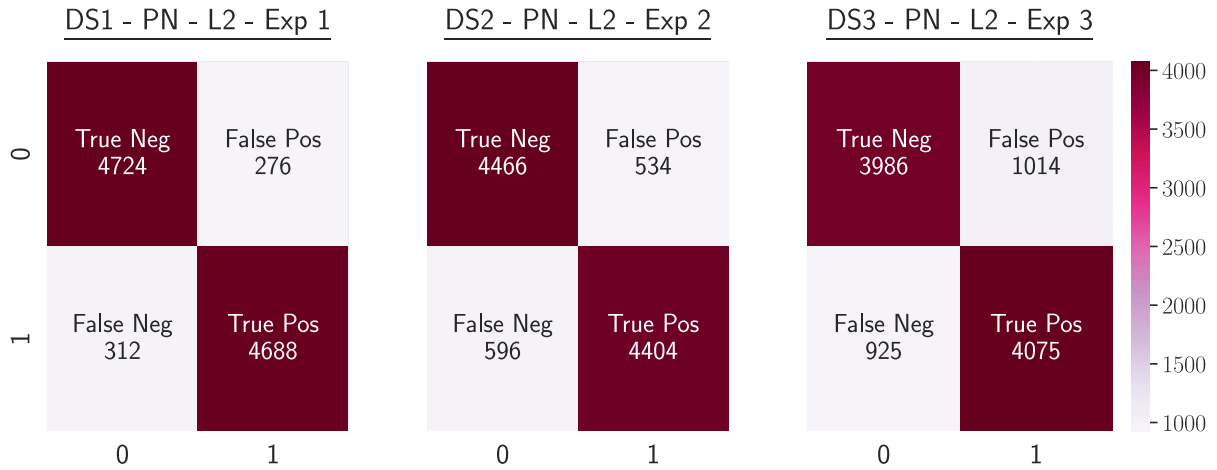


Figure 5.5: Confusion Matrix of different datasets. The parameters for the experiments are listed in detail in Table 4.6.

In summary, both the prototypical and matching networks are excellent methods for Few-Shot Classification.

5.2.2 Sufficient Shots

Another proven method is considered, where the shots for support and query set are varied. The number of examples for creating the prototypes concerning the prototypical network and the shots of each epoch size is changed.

The correlation between training loss/accuracy and the performance metrics varies for the number of support and query examples. While the highest number of support and query examples (Exp. 16 - 7 Support/7 Query) is the most efficient for training, which can be seen in Figure 5.6, experiment 16 has not the highest performance (Figure 5.7). Better performance can be achieved by a higher number of support examples, which is necessary for a good F1-score around 95 % (Exp. 1, 14, 18). The larger the support sets are, the more samples can be trained. A query set with 7 samples will decrease the performance. Testing performance is better on fewer query sets because the probability of classifying correctly with fewer examples is higher.

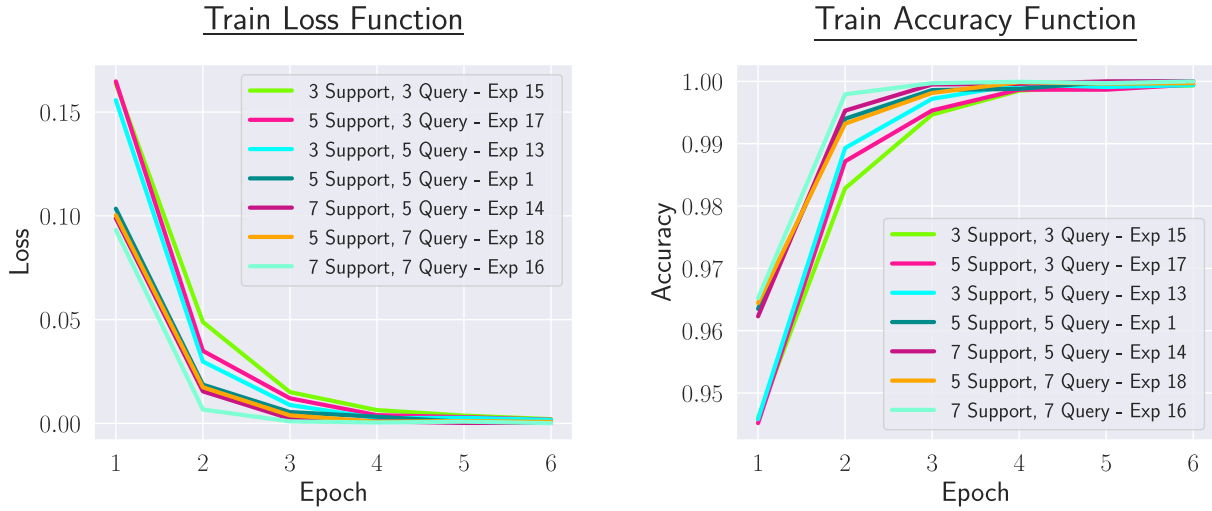


Figure 5.6: Training loss and accuracy on different shots for the support and query set. The parameters for the experiments are listed in detail in Table 4.6.

The training takes longer with few shots per set as in experiments 15 and 17, and the performance is significantly worse. The best results with a F1-score of 94.82 % are achieved with 7 shots for the support set and 5 for the query set in both the training and testing phase (Exp.14). In addition, the recall of 95.30 % is high, which can be seen in figure 5.7.

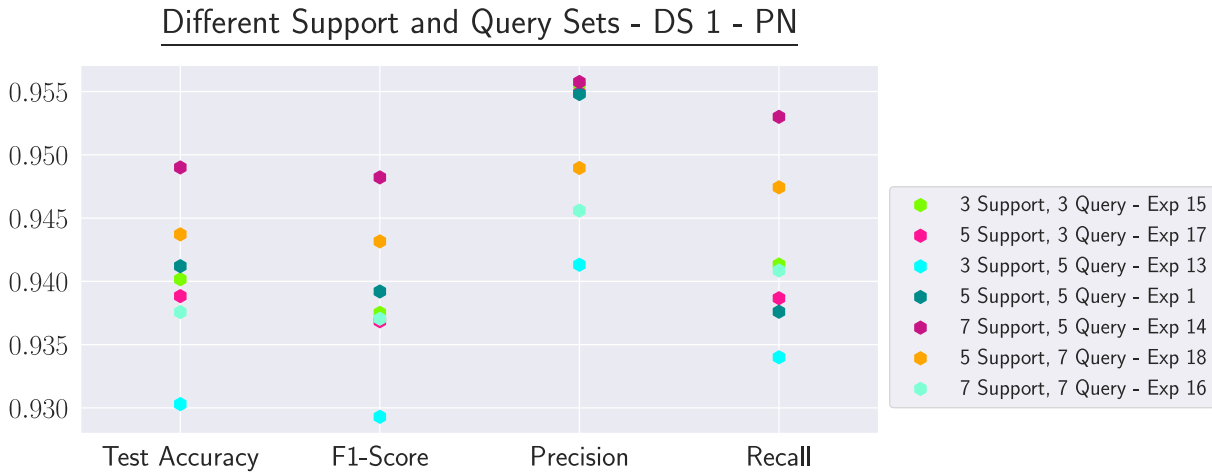


Figure 5.7: Metrics for the performance evaluation on different shots for the support and query set. The parameters for the experiments are listed in detail in Table 4.6.

The precision of 95.48 % with 5/5 shots (Figure 5.7) is also high. However, experiment 1 has a lower recall of 93.76 %. In the case of anomaly detection, it is essential to have a high recall to detect and filter anomalies out. It is more acceptable to deal with good samples considered as bad. In contrast, anomalies should not be classified as good.

The Confusion Matrix (Figure 5.8) shows that experiment 14 classifies most efficient with 4725 as *True Negative* and 1765 as *True Positive*.

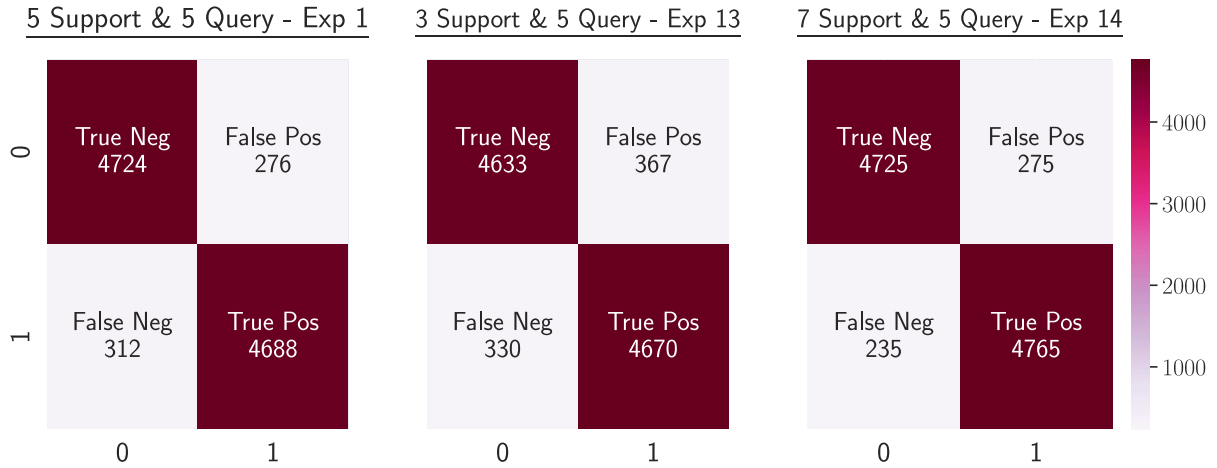


Figure 5.8: Confusion Matrix on different shots for the support and query set. The parameters for the experiments are listed in detail in Table 4.6.

5.2.3 Learning Methods

Three Few-Shot Learning methods were presented within this work: Matching Network, Prototypical Network, and Relation Network. While the first two showed good results, the Relation Network proved unsuitable for process data, so an evaluation was neglected. In the following Figure 5.9, the training of the PN is more efficient with a loss of $1.3e^{-5}$ than the MN with $5e^{-5}$. The formation of a prototype leads to better results than the matching network, as Figure 5.10 shows with 234 more *True Negative* and 299 *True Positive* samples. In addition, the formation of this centroid significantly increases the runtime, which was not part of the task and, therefore, will not be discussed further.

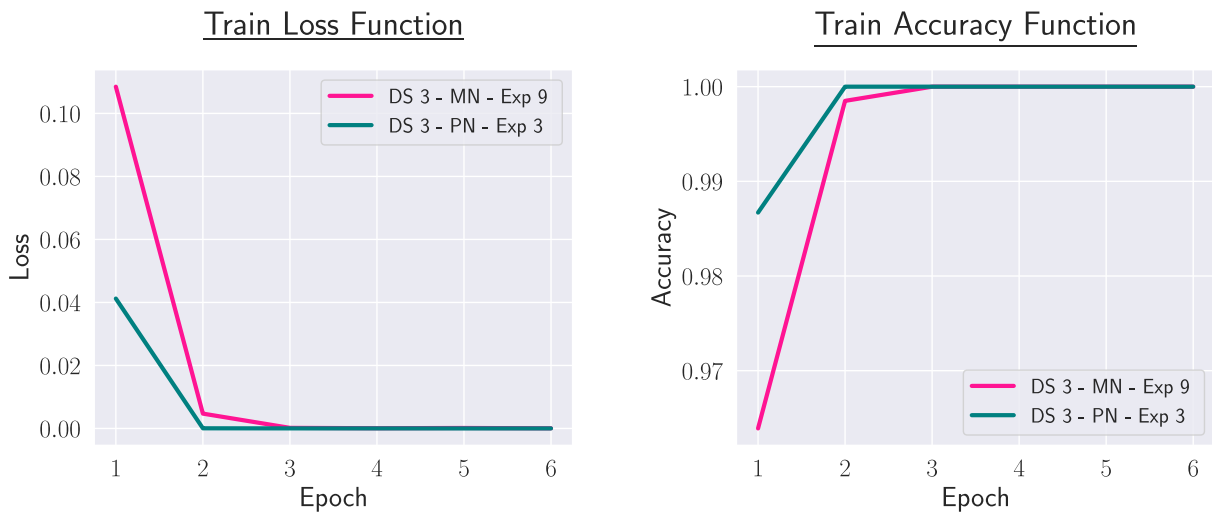


Figure 5.9: Train loss and accuracy trained on different learning methods. The parameters for the experiments are listed in detail in Table 4.6.

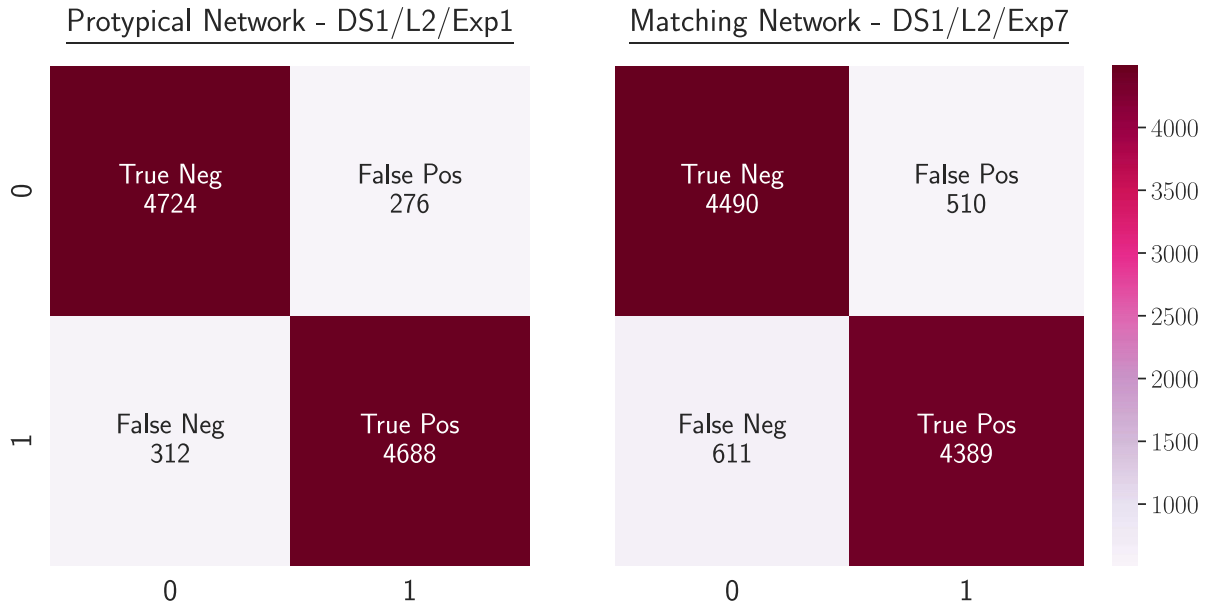


Figure 5.10: Confusion Matrix of different learning methods. The parameters for the experiments are listed in detail in Table 4.6.

5.2.4 Distance Ranking Approach

When the classification task is based on a model whose training has been driven by a distance-based loss, both the distance-ranking and the voting approaches can calculate the embeddings of the two classes and derive the distance between them. The difference is that this distance itself determines the final ranking in the distance ranking approach. In contrast, in the voting system, the elements of the distance vector are averaged and the result is compared to the threshold t calculated at the end of the training phase to output a boolean response. It is reasonable to assume that there is no advantage to using the voting system. The threshold does not help identify anomalies since it only leads to a loss of information by turning a distance into a boolean value resulting from a comparison. So the direct use of the distance is better [94].

Figures 5.11 and 5.12 show that the training performs better using Euclidean distance over Cosine distance. This effect is even more pronounced for prototypical networks, in which computing the class prototype as the mean of embedded support points is more naturally suited to Euclidean distances since cosine distance is not a Bregman divergence [117], which measures the difference between two points defined in terms of a strictly convex function.

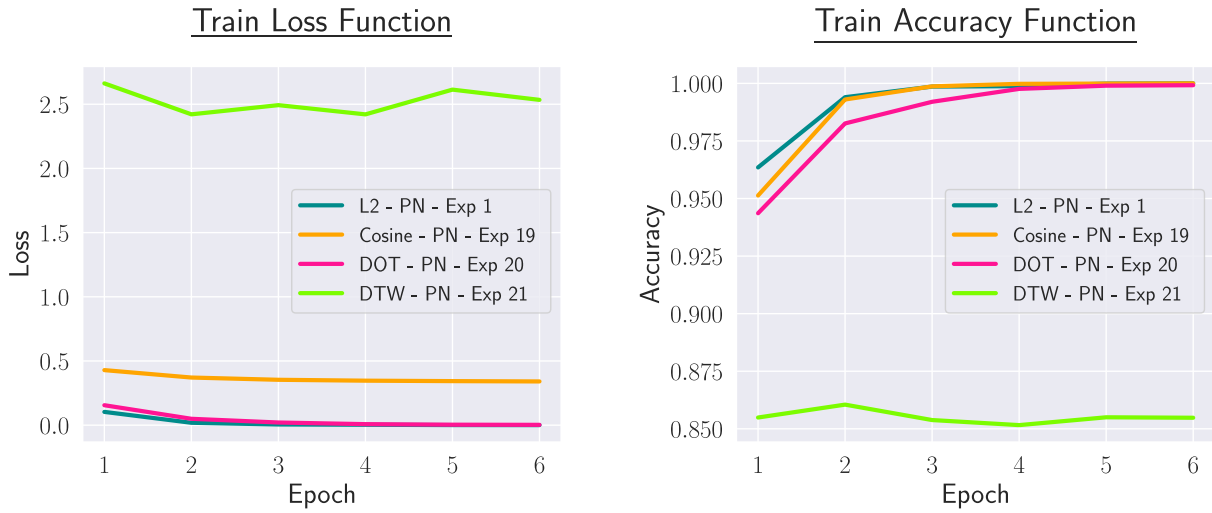


Figure 5.11: Train loss and accuracy were evaluated on 4 different distance/similarity metrics on the Prototypical Network. The parameters for the experiments are listed in detail in Table 4.6.

Figure 5.12 gives a deeper insight into the performance of the metrics, where the DTW is not considered, thus not performing well with a F1-score of 64.60 % (Table 5.1). One assumption why DTW does not work is that the data used is cyclic, leading to worse performance in classification.

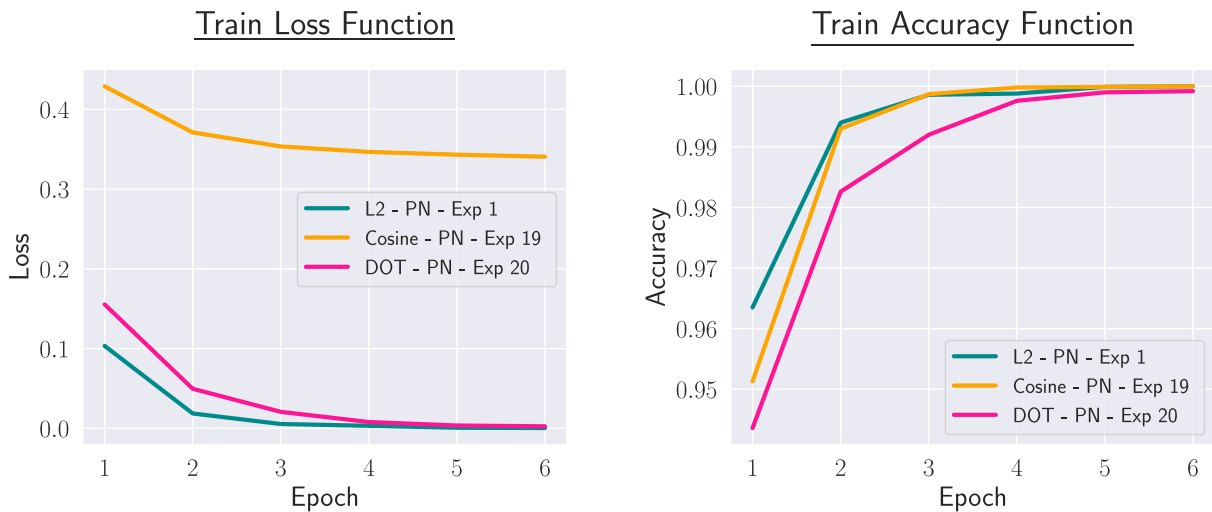


Figure 5.12: Train loss and accuracy were evaluated on 3 different distance/ similarity metrics on the Prototypical Network. The parameters for the experiments are listed in detail in Table 4.6.

Using Euclidean distance improves the training performance substantially with a training loss of $4e^{-4}$ over cosine distance with a loss of $3.407e^{-1}$. However, the F1-score and recall of L2 are 93.92 % and 93.76 %, for the cosine distance, 95.21 % and 95.30 %.

Diagrams 5.13, 5.14, and 5.15 representative describe the Matching Network and its metrics.

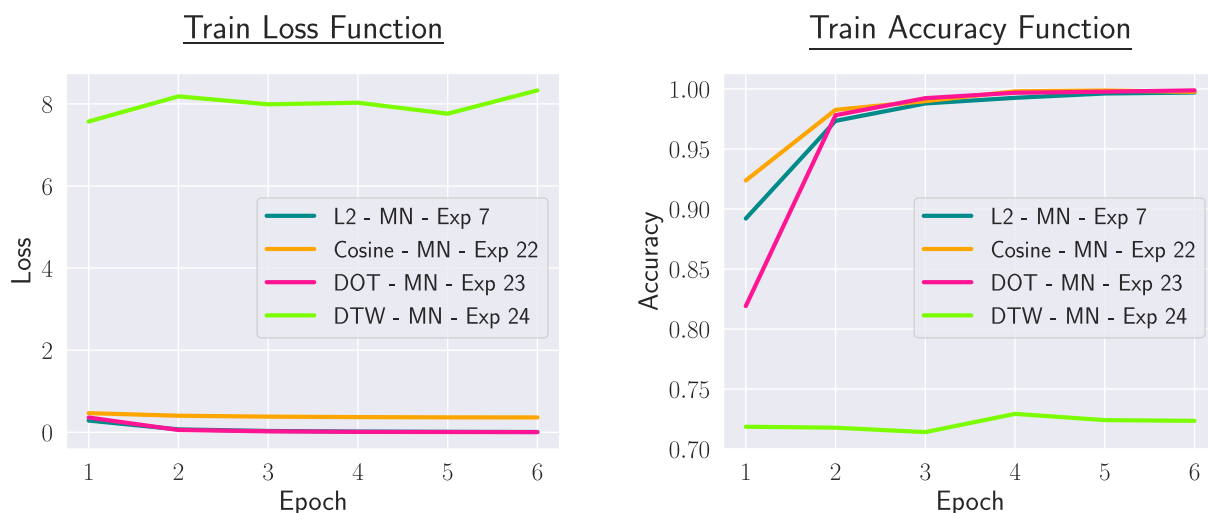


Figure 5.13: Train loss and accuracy were evaluated on 4 different distance/ similarity metrics on the Matching Network. The parameters for the experiments are listed in detail in Table 4.6.

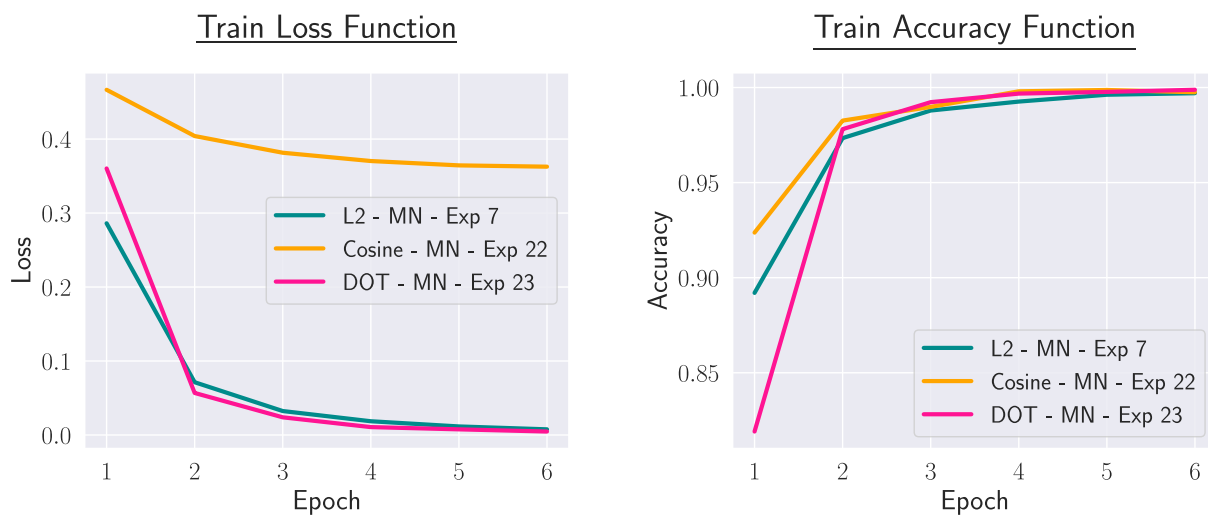


Figure 5.14: Train loss and accuracy were evaluated on 3 different distance/ similarity metrics on the Matching Network. The parameters for the experiments are listed in detail in Table 4.6.

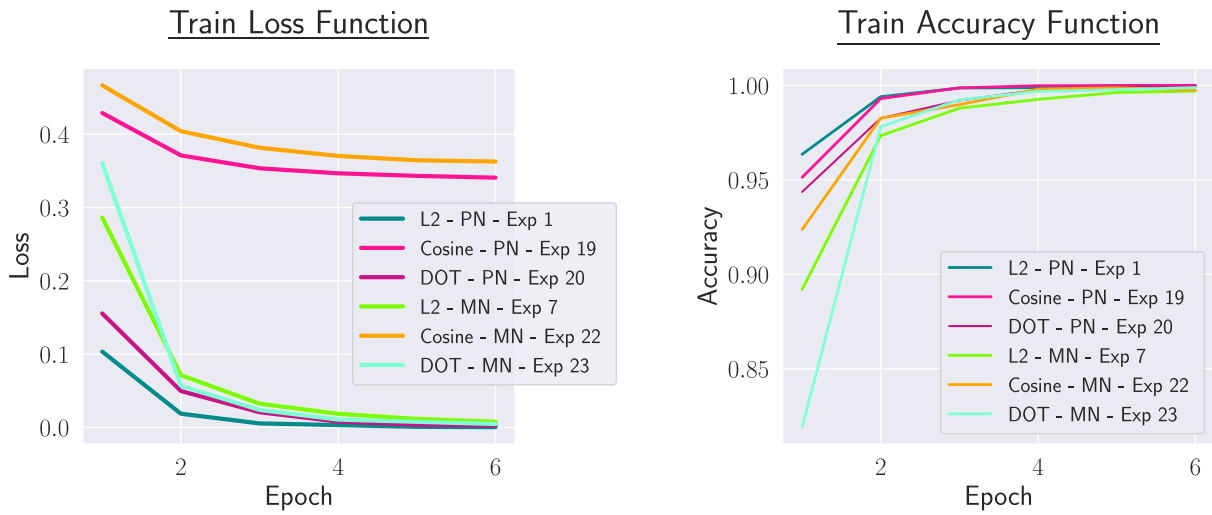


Figure 5.15: Comparison between the train loss and accuracy with different metrics used in Prototypical Network and Matching Network. The parameters for the experiments are listed in detail in Table 4.6.

In summary, while L2 trains with at least epochs, it classifies with a F1-score of 93.92 %. At the same time, Cosine Distance needs more epochs to train but classifies with a F1-score of 95.21 %. For the Prototypical Network, the metrics L2 and Cosine perform well for the Matching Network, Cosine and the DOT product are preferred.

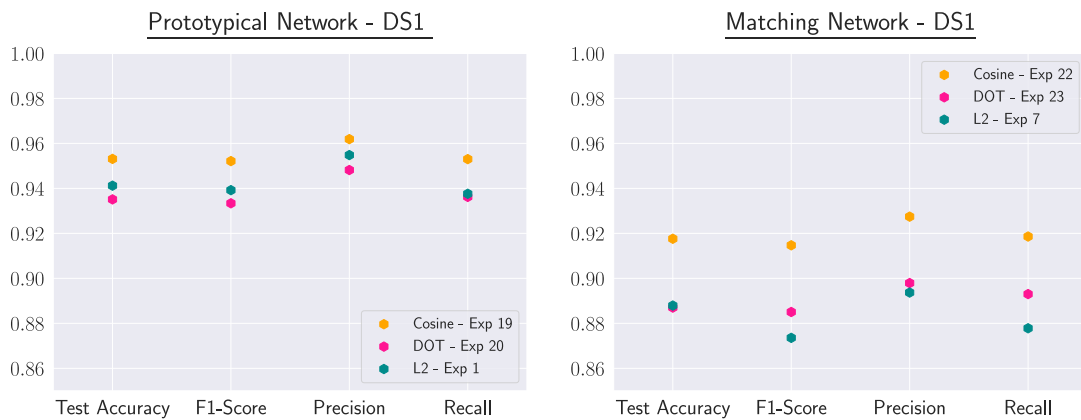


Figure 5.16: Metrics for the performance evaluation on different distance/similarity measures. The parameters for the experiments are listed in detail in Table 4.6.

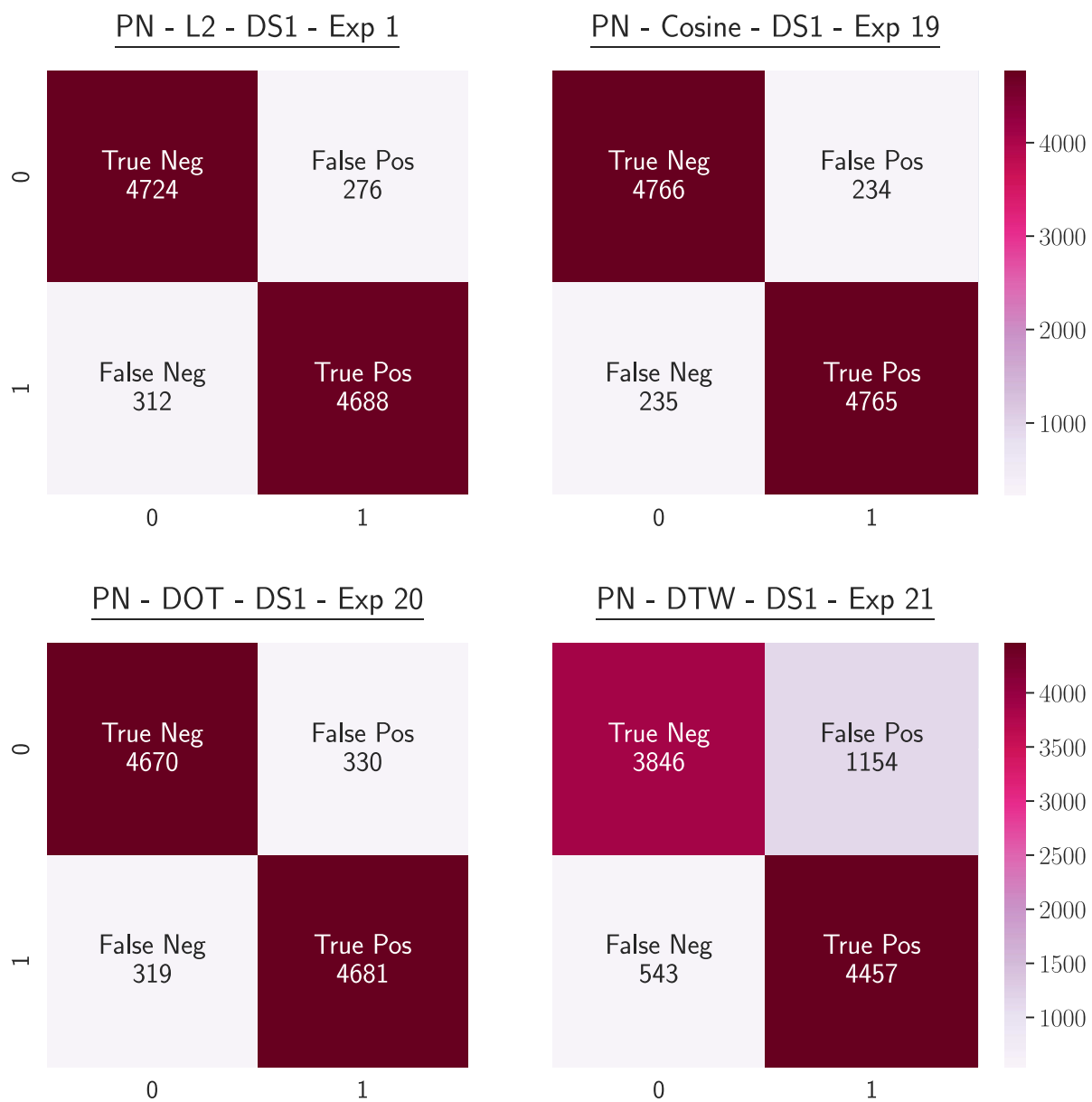


Figure 5.17: Confusion Matrix of different distance/similarity measures. The parameters for the experiments are listed in detail in Table 4.6.

6 Retrospective

Before summarizing, some words are dedicated to highlighting all the positive aspects and difficulties that have been encountered. Through the literature review in the field of Meta-Learning and Few-Shot Learning, FSL methods were selected that can meet the requirements for the classification of noisy industrial data. There has always been much freedom regarding how and with what techniques to explore. This led to a broader exploration of different methods and approaches, which were not discussed or mentioned in the course of this work, as this would go beyond the scope. Only the approaches used and final procedures were formulated and explained in more detail in the chapter background and state-of-the-art. Thereby, a certain diversity arose in the implementation of the solution. Many different modules had to be designed to tackle particular subtasks. These include the data processing, data augmentation, the optimization of the network architecture, the classification part, and the experimentation phase based on the definition of scenarios.

6.1 Achievements

A final well-performing solution framework has been designed.

6.2 Points to Improve

In addition to the positive points, some difficulties and points, which could be improved, are mentioned:

1. A large number of parameters and implementation options leads to many possible scenarios. Therefore, a good organization was necessary to overview all parameter settings and the corresponding scenarios to cope with this.
2. Due to the long experimentation phase, the number of final trials was limited in time. In this context, further investigations would be a good option.
3. Related to the freedom already mentioned, the reverse side is that clarifying the objectives was not straightforward. However, this phase was very progressive, and the goals were slightly adjusted in terms of performance expectations to see how

the effective work went. My supervisor was very supportive in asking the right questions, but at the same time, he gave me enough freedom to figure out which direction I wanted to go.

6.3 Outlook

Finally, this work is only a subtask of a more considerable scientific investigation and can be integrated.

Some other points that could be relevant to inspect are listed here:

1. In the evaluation phase, the experiments could run more often with the same settings and data sets and then average the results to get an average and thus a better generalization. However, due to lack of time, this could not be done.
2. When creating the support and query sets, it would be possible to specify the same samples for each experiment so that there is no random selection in the support/query set. The training and test set, on the other hand, is always the same and does not arise by chance.
3. A more complex performing feature extractor can replace the CNN architecture.
4. Additionally, a hyperspherical coordinate system can be considered.

7 Conclusion

This work proposed a strategy for anomaly detection in a specific usecase using Few-Shot Learning techniques achieving a good final performance with a limited amount of data. The preprocessed and augmented data provided good input parameters for learning Meta-Metric-based neural networks, which were evaluated against the verification task. The evaluation of the best performing model was based on the correct classification of a sample based on a labeled instance. For this purpose, this sample is compared to each time series instance by calculating the distance between their embeddings. The selected distance and shots were varied for the representative embeddings.

Regarding the performance that was obtained, the Prototypical Network using the Euclidean Distance reached a F1-score of 93.92 % on the verification task when trained based on 70 good and 21 bad samples (data set 1) and a F1-score of 80.01 % with 17 good and six bad samples (data set 3). The Matching Network reached a F1-score of 87.34 % and 71.81 %. By implementing Cosine Distance as the final classification, PN achieves an F1-score of 95.21 % and MN a F1-score of 91.46 % with dataset 1 (Table 4.2). The DOT-product achieves a F1-performance of 93.51 % for the PN and 88.70 % for the MN. When selecting the number of samples for the support and query set, the number of shots, especially for the support set, should be about 5 to 7, with a F1-score of 93.92 % and 94.82 %. Three shots are insufficient for each support and query set with a F1-score of 92.92 %.

With 17 good examples and six bad examples, a network trains successfully and performs well in the explored datasets. It should be noted that the number of samples is increased by data augmentation and preprocessing, which leads to better performance. The Prototypical Network and Matching Network are excellent methods to train a neural network with few data.

Finally, many ways can still be explored, especially regarding the retraining always on specific and not random chosen support and query sets and the evaluation phase, which could lead to a more average result. Additionally it should be noted that the described techniques are mainly used for computer vision and not for time series data. In the context of this work, they produced good results for the used data. However, it should be considered that this cannot necessarily be generalized and still needs to be investigated further.

8 Appendix

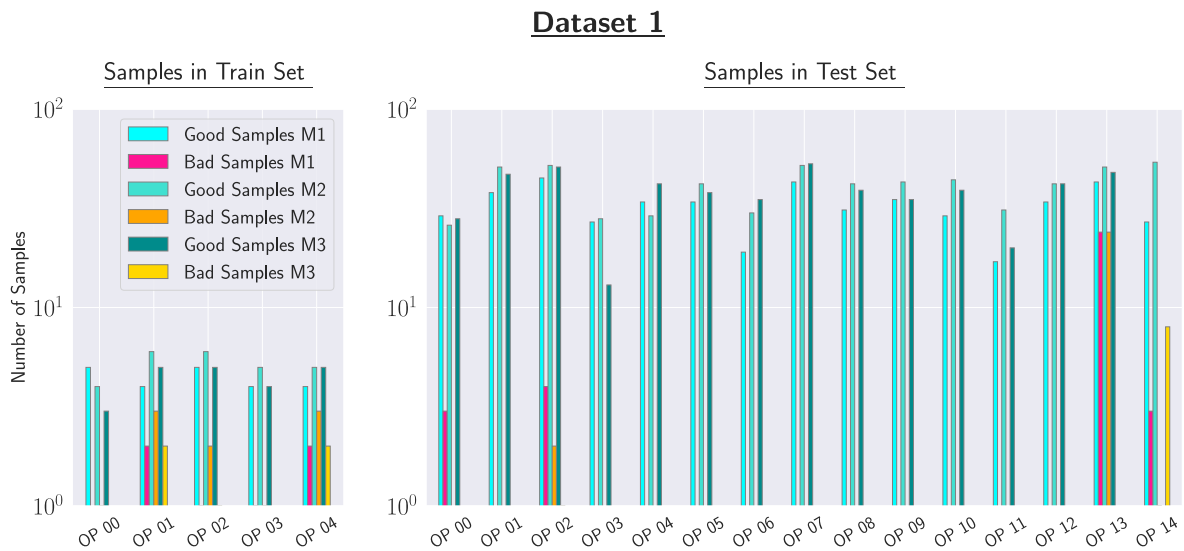


Figure 8.1: Shows the sample split for data set 1, which is split into a training set(left) and a test set(right). The training set includes samples from process 0 to 4, the test set from process 0 to 14. It can be seen that the dataset is unbalanced even though data augmentation was applied, which is a challenge for training. For some processes there were only good examples but no bad ones.



Figure 8.2: Shows the sample split for data set 2, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 1, and the test set contains fewer samples from processes 0 to 14 than data set 1.

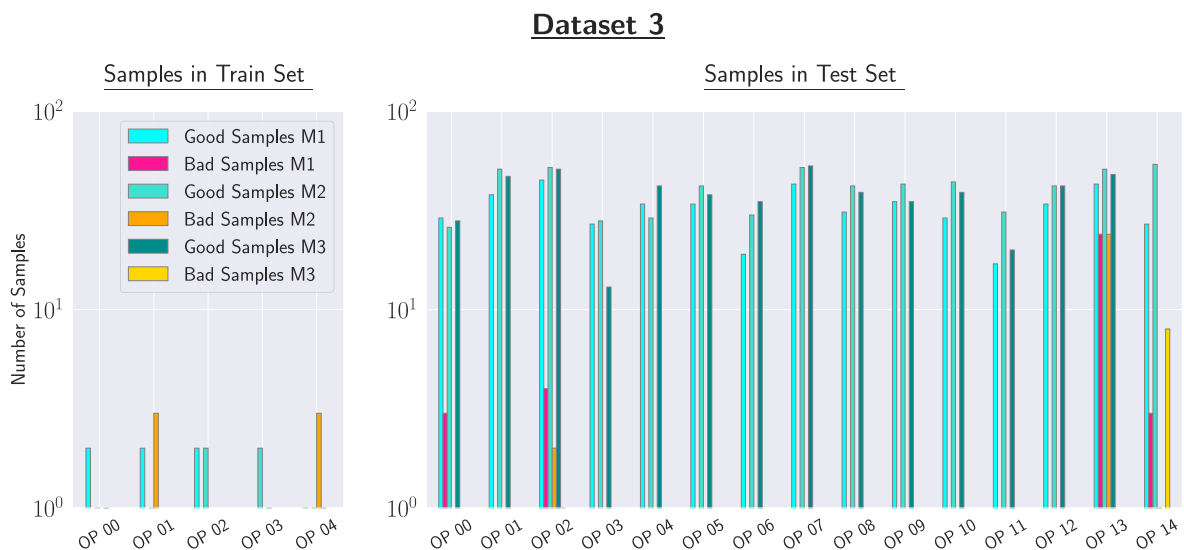


Figure 8.3: Shows the sample split for data set 3, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 2, and the test set contains fewer samples from processes 0 to 14 than data set 2.

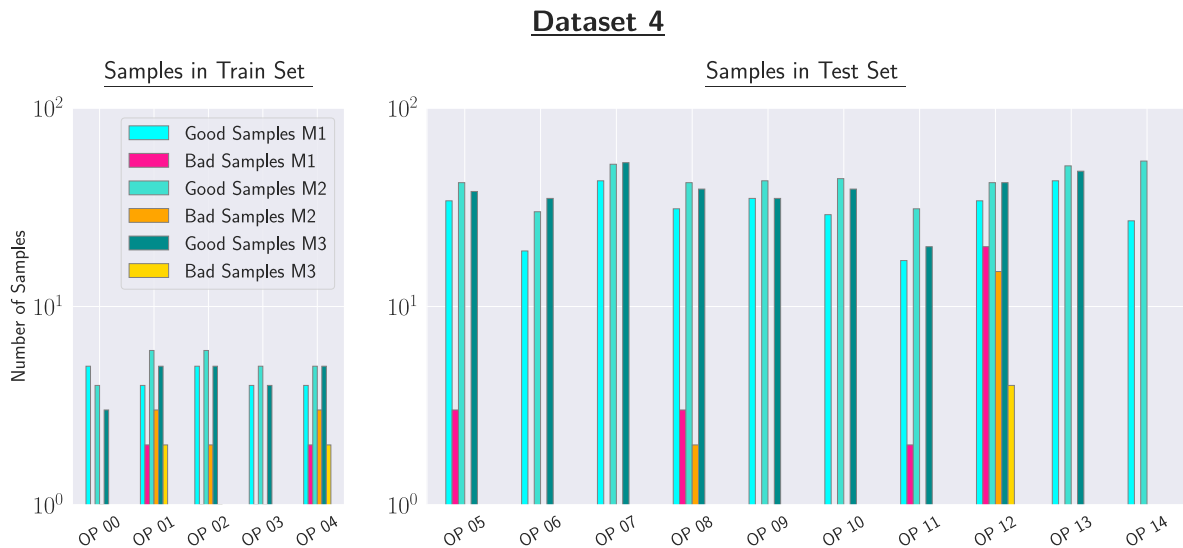


Figure 8.4: Shows the sample split for data set 4, which is split into a training set(left) and a test set(right). The training set includes samples from process 0 to 4, the test set from process 5 to 14.

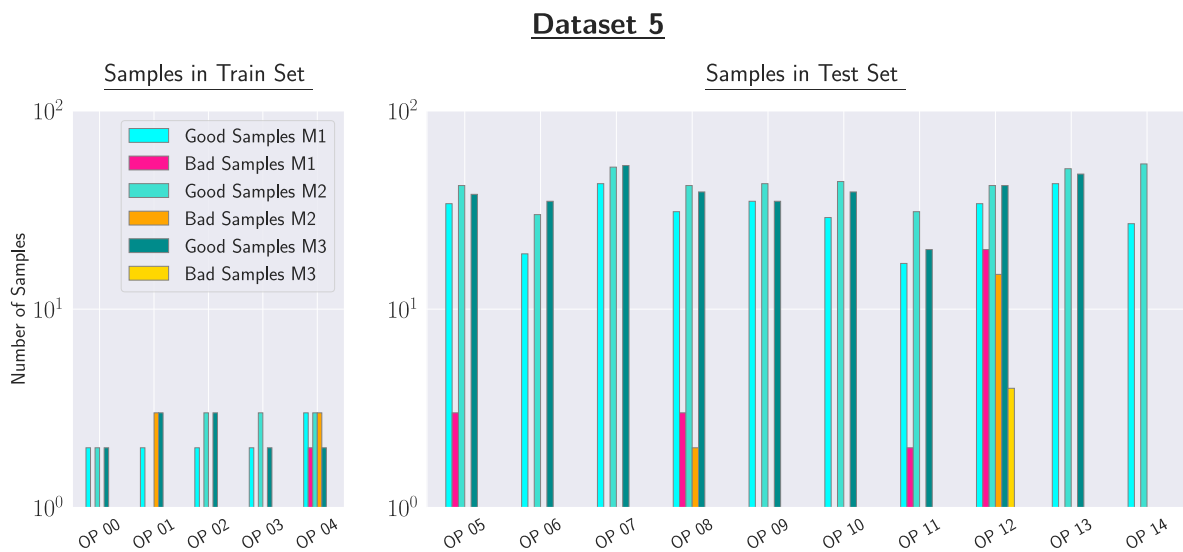


Figure 8.5: Shows the sample split for data set 5, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 4, and the test set contains fewer samples from processes 5 to 14 than data set 4.

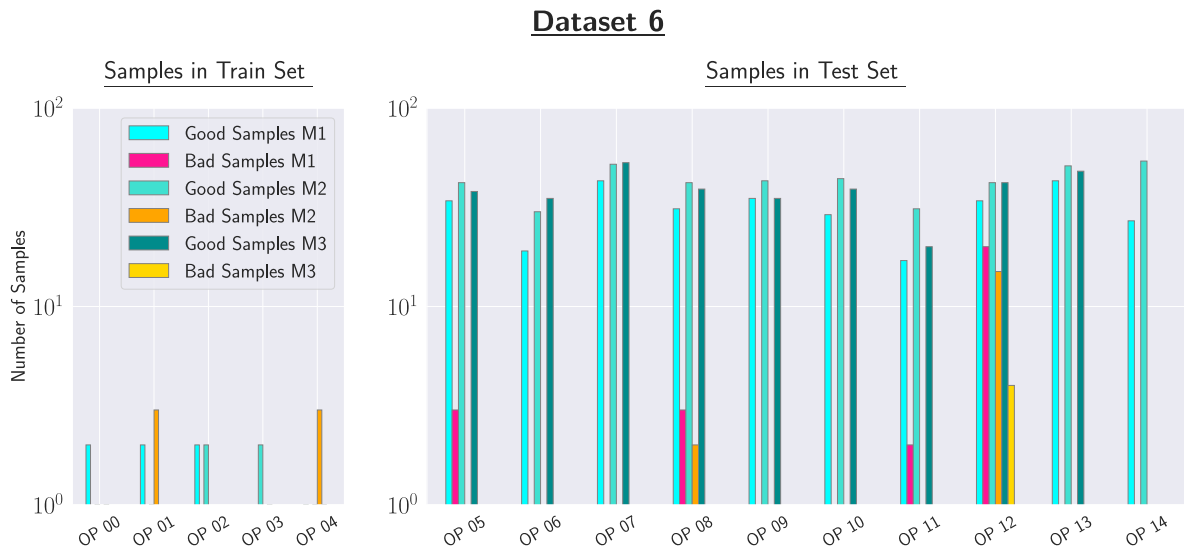


Figure 8.6: Shows the sample split for data set 6, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 5, and the test set contains fewer samples from processes 5 to 14 than data set 5.

Bibliography

- [1] Thorsten Wuest, Christopher Irgens, and Klaus-Dieter Thoben. An approach to monitoring quality in manufacturing using supervised machine learning on product state data. *Journal of Intelligent Manufacturing*, 25(5):1167–1180, 2014.
- [2] Aditya M Deshpande, Ali A Minai, and Manish Kumar. One-shot recognition of manufacturing defects in steel surfaces. *Procedia Manufacturing*, 48:1064–1071, 2020.
- [3] Shruti Jadon. An overview of deep learning architectures in few-shot learning domain. *arXiv preprint arXiv:2008.06365*, 2020.
- [4] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 403–412, 2019.
- [5] Machine Learning. Generalization. <https://developers.google.com/machine-learning/crash-course/generalization/video-lecture>, august 2022.
- [6] S Mahapatra. Why deep learning over traditional machine learning? <https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063>, august 2022.
- [7] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [9] Mahmut Kaya and Hasan Şakir Bilge. Deep metric learning: A survey. *Symmetry*, 11(9):1066, 2019.
- [10] Thien Khai Tran and Tuoi Thi Phan. Deep learning application to ensemble learning—the simple, but effective, approach to sentiment classifying. *Applied Sciences*, 9(13):2760, 2019.
- [11] Mike Huisman, Jan N van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, pages 1–59, 2021.

-
- [12] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.
- [13] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.
- [14] Xiaoxu Li, Xiaochen Yang, Zhanyu Ma, and Jing-Hao Xue. Deep metric learning for few-shot image classification: A selective review. *arXiv preprint arXiv:2105.08149*, 2021.
- [15] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.
- [16] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020.
- [17] W. Zi. Tutorial 2: few-shot learning and meta-learning i. <https://www.borealisai.com/en/blog/tutorial-2-few-shot-learning-and-meta-learning-i/>, september 2019.
- [18] Ruibing Hou, Hong Chang, Bingpeng Ma, Shiguang Shan, and Xilin Chen. Cross attention network for few-shot classification. *arXiv preprint arXiv:1910.07677*, 2019.
- [19] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29:3630–3638, 2016.
- [20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.
- [21] Marcus Rohrbach, Sandra Ebert, and Bernt Schiele. Transfer learning in a transductive setting. *Advances in neural information processing systems*, 26:46–54, 2013.
- [22] Eunjung Lee and Wonjong Rhee. Individualized short-term electric load forecasting with deep neural network based transfer learning and meta learning. *IEEE Access*, 9:15413–15425, 2021.
- [23] Jiajun Pan. Review of metric learning with transfer learning. In *AIP Conference Proceedings*, volume 1864, page 020040. AIP Publishing LLC, 2017.

-
- [24] Yong Luo, Yonggang Wen, Ling-Yu Duan, and Dacheng Tao. Transfer metric learning: Algorithms, applications and outlooks. *arXiv preprint arXiv:1810.03944*, 2018.
- [25] Yiwen Sun, Kun Fu, Zheng Wang, Changshui Zhang, and Jieping Ye. Road network metric learning for estimated time of arrival. In *2020 25th International Conference On Pattern Recognition (ICPR)*, pages 1820–1827. IEEE, 2021.
- [26] Eric Xing, Michael Jordan, Stuart J Russell, and Andrew Ng. Distance metric learning with application to clustering with side-information. *Advances in neural information processing systems*, 15:521–528, 2002.
- [27] Michael M Wolf. *Mathematical foundations of supervised learning*, 2018.
- [28] Maria-Irina Nicolae. *Learning similarities for linear classification: theoretical foundations and algorithms*. PhD thesis, Lyon, 2016.
- [29] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.
- [30] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.
- [31] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.
- [32] L Bottou and O Bousquet. The tradeoffs of large scale learning advances in neural information processing systems 20, 2008.
- [33] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.
- [34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. springer series in statistics. In \therefore . Springer, 2001.
- [35] Yunxiao Qin, Weiguo Zhang, Chenxu Zhao, Zezheng Wang, Xiangyu Zhu, Jingping Shi, Guojun Qi, and Zhen Lei. Prior-knowledge and attention based meta-learning for few-shot learning. *Knowledge-Based Systems*, 213:106609, 2021.
- [36] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2018.
- [37] Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 920–923. IEEE, 2017.

-
- [38] Huy L Nguyen and Lydia Zakyntinou. Improved algorithms for collaborative pac learning. *arXiv preprint arXiv:1805.08356*, 2018.
- [39] Pascal Germain, Francis Bach, Alexandre Lacoste, and Simon Lacoste-Julien. Pac-bayesian theory meets bayesian inference. *arXiv preprint arXiv:1605.08636*, 2016.
- [40] Sridhar Mahadevan and Prasad Tadepalli. Quantifying prior determination knowledge using the pac learning model. *Machine Learning*, 17(1):69–105, 1994.
- [41] Michael Fink. Object classification from a single example utilizing class relevance metrics. *Advances in neural information processing systems*, 17:449–456, 2005.
- [42] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.
- [43] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.
- [44] J. Brownlee. What is meta-learning in machine learning? [WhatIsMeta-LearninginMachineLearning?](#), december 2018.
- [45] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.
- [46] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.
- [47] Devang K Naik and Richard J Mammone. Meta-neural networks that learn by learning. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 437–442. IEEE, 1992.
- [48] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.
- [49] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.
- [50] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

-
- [51] Han-Jia Ye, Xiang-Rong Sheng, and De-Chuan Zhan. Few-shot learning with adaptively initialized task optimizer: a practical meta-learning approach. *Machine Learning*, 109(3):643–664, 2020.
- [52] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.
- [53] D. Thailappan. An introduction to few-shot learning. <https://www.analyticsvidhya.com/blog/2021/05/an-introduction-to-few-shot-learning/>, july 2021.
- [54] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.
- [55] Peipei Yang, Kaizhu Huang, and Amir Hussain. A review on multi-task metric learning. *Big Data Analytics*, 3(1):1–23, 2018.
- [56] Dmitri Burago, Yuri Burago, and Sergei Ivanov. *A course in metric geometry*, volume 33. American Mathematical Society, 2022.
- [57] Liu Yang and Rong Jin. Distance metric learning: A comprehensive survey. *Michigan State University*, 2(2):4, 2006.
- [58] Romain Negrel, David Picard, and Philippe-Henri Gosselin. Efficient metric learning based dimension reduction using sparse projectors for image near duplicate retrieval. In *2014 22nd International Conference on Pattern Recognition*, pages 738–743. IEEE, 2014.
- [59] Amir Globerson and Sam Roweis. Metric learning by collapsing classes. *Advances in neural information processing systems*, 18, 2005.
- [60] Yueqi Duan, Jiwen Lu, Jianjiang Feng, and Jie Zhou. Deep localized metric learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2644–2656, 2017.
- [61] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of machine learning research*, 10(2), 2009.
- [62] Chen Huang, Chen Change Loy, and Xiaoou Tang. Local similarity-aware deep feature embedding. *Advances in neural information processing systems*, 29:1262–1270, 2016.

-
- [63] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM transactions on graphics (TOG)*, 34(4):1–10, 2015.
- [64] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.
- [65] Sakshi Indolia, Anil Kumar Goswami, Surya Prakesh Mishra, and Pooja Asopa. Conceptual understanding of convolutional neural network-a deep learning approach. *Procedia computer science*, 132:679–688, 2018.
- [66] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.
- [67] G.Di. Pooling layers. <https://guandi1995.github.io/Pooling-Layers/>, july 2020.
- [68] A Kumar. Convolutional neural network (cnn) – simply explained. https://vitalflux.com/convolutional-neural-network-cnn-simply-explained/#Whats_Convolution_Whats_intuition_behind_Convolution, november 2020.
- [69] Mathworld Wolfram. Convolution. <https://mathworld.wolfram.com/Convolution.html>, july 2022.
- [70] G Kashyap. How to use conv2d layers as fully connected layers. <https://medium.com/@knighthawkk/how-to-use-conv2d-layers-as-fully-connected-layers-b0a82eb8a408>, sept. 2021.
- [71] Quynh Nguyen and Matthias Hein. Optimization landscape and expressivity of deep cnns. In *International conference on machine learning*, pages 3730–3739. PMLR, 2018.
- [72] C. Olah. Conv nets: A modular perspective. <https://colah.github.io/posts/2014-07-Conv-Nets-Modular/>, july 2014.
- [73] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.
- [74] Yu Zhou, Haipeng Wang, Feng Xu, and Ya-Qiu Jin. Polarimetric sar image classification using deep convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, 13(12):1935–1939, 2016.

-
- [75] Ki Bum Lee, Sejune Cheon, and Chang Ouk Kim. A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, 30(2):135–142, 2017.
- [76] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks*, pages 195–201. Springer, 1995.
- [77] C. Versloot. Relu, sigmoid and tanh: today’s most used activation functions. <https://www.machinecurve.com/index.php/2019/09/04/relu-sigmoid-and-tanh-todays-most-used-activation-functions/>, july 2019.
- [78] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [79] Shraddha Pandit, Suchita Gupta, et al. A comparative study on distance measuring approaches for clustering. *International Journal of Research in Computer Science*, 2(1):29–31, 2011.
- [80] Marina Adriana Mercioni and Stefan Holban. A survey of distance metrics in clustering data mining techniques. In *Proceedings of the 2019 3rd International Conference on Graphics and Signal Processing*, pages 44–47, 2019.
- [81] Jasmine Irani, Nitin Pise, and Madhura Phatak. Clustering techniques and the similarity measures used in clustering: A survey. *International journal of computer applications*, 134(7):9–14, 2016.
- [82] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.
- [83] Xiaobin Chang, Frederick Tung, and Greg Mori. Learning discriminative prototypes with dynamic time warping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8395–8404, 2021.
- [84] Xingyu Cai, Tingyang Xu, Jinfeng Yi, Junzhou Huang, and Sanguthevar Rajasekaran. Dtwnet: a dynamic time warping network. *Advances in neural information processing systems*, 32, 2019.
- [85] Pavel Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23):40, 2008.

-
- [86] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.
- [87] Rohit J Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, 30(2):283–312, 2016.
- [88] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter*, 12(1):40–48, 2010.
- [89] Aditya Mishra. Metrics to evaluate your machine learning algorithm. *Towards Data Science*, January 2022.
- [90] Aayush Baja. Performance metrics in machine learning. *Neptune Blog*, January 2022.
- [91] J. Durán. Everything you need to know about gradient descent applied to neural networks. <https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14>, july 2019.
- [92] A. Sharma. Understanding activation functions in deep learning. <https://learnopencv.com/understanding-activation-functions-in-deep-learning/>, july 2017.
- [93] Aatish Kayyath. Confusion matrix : Let’s clear this confusion. *Medium*, January 2022.
- [94] Géraldine Brieven et al. Master thesis: One-shot learning for face recognition. 2019.
- [95] Ivan Vasilev, Daniel Slater, Gianmario Spacagna, Peter Roelants, and Valentino Zocca. *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*. Packt Publishing Ltd, 2019.
- [96] Peter Goldsborough. A tour of tensorflow. *arXiv preprint arXiv:1610.01178*, 2016.
- [97] Machine Learning Deep Learning. Stochastic gradient descent versus mini batch gradient descent versus batch gradient descent. <https://programmatically.com/stochastic-gradient-descent-versus-mini-batch-gradient-descent-versus-batch-gradient-descent/#:~:text=Mini%20batch%20gradient%20descent%20is,to%20a%20smoother%20learning%20curve.>, september 2021.
- [98] Shengli Sun, Qingfeng Sun, Kevin Zhou, and Tengchao Lv. Hierarchical attention prototypical networks for few-shot text classification. In *Proceedings of the 2019*

- Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 476–485, 2019.
- [99] Haopeng Ren, Yi Cai, Xiaofeng Chen, Guohua Wang, and Qing Li. A two-phase prototypical network model for incremental few-shot relation classification. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1618–1629, 2020.
- [100] Alice Baird, Silvan Mertes, Manuel Milling, Lukas Stappen, Thomas Wiest, Elisabeth André, and Björn W Schuller. A prototypical network approach for evaluating generated emotional speech. *Proc. Interspeech 2021*, pages 3161–3165, 2021.
- [101] G. Polat. Deep learning architectures that you can use with a few data. <https://medium.com/swlh/deep-learning-architectures-that-you-can-use-with-a-very-few-data-8e5b4fa1d5da>, july 2020.
- [102] Mohamed Ali Thani. Design and validation of iot architectures for deep learning application in fault diagnosis of rotating machinery. *Master’s thesis*, 2019.
- [103] Subarnaduti Paul. An autoencoder based efficient feature extractor for noisy industrial data. *Master’s thesis*, 2021.
- [104] A Famili, Wei-Min Shen, Richard Weber, and Evangelos Simoudis. Data preprocessing and intelligent data analysis. *Intelligent data analysis*, 1(1):3–23, 1997.
- [105] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. Data preprocessing for supervised leaning. *International journal of computer science*, 1(2):111–117, 2006.
- [106] Edwin Onuonga. Downsampling. *sequentia*, January 2022.
- [107] Machine Learning. Imbalanced data. <https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data>, august 2022.
- [108] SciPy documentation. `scipy.signal.decimate`. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.decimate.html>, august 2022.
- [109] S Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.
- [110] S Benchaou, M Nasri, O El Melhaoui, and B Bouali. New approach of preprocessing for numeral recognition. *Int. Journal of Engineering Research and Applications*, 4(7):26–30, 2014.

- [111] Johannes Fürnkranz. More efficient windowing. In *AAAI/IAAI*, pages 509–514. Citeseer, 1997.
- [112] Xin Zhang, Liangxiu Han, Lianghao Han, and Liang Zhu. How well do deep learning-based methods for land cover classification and object detection perform on high resolution remote sensing imagery? *Remote Sensing*, 12(3):417, 2020.
- [113] Xiangtao Chen, Zhouzhou Liu, and SH Zhu. Finding contrast patterns in imbalanced classification based on sliding window. In *Proceedings of the 4th International Conference on Mechanical Materials and Manufacturing Engineering (MMME 2016), Advances in Engineering Research*, volume 10, pages 161–166, 2016.
- [114] Pytorch: Steplr. *PyTorch Documentary*, January 2022.
- [115] Sebastian Bock and Martin Weiß. A proof of local convergence for the adam optimizer. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.
- [116] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2. IEEE, 2018.
- [117] Michael Gutmann and Jun-ichiro Hirayama. Bregman divergence as general framework to estimate unnormalized statistical models. *arXiv preprint arXiv:1202.3727*, 2012.

List of Figures

2.1	Difference between learning with enough and few training samples [16]. . .	7
2.2	Perspectives on how FSL methods solve the few-shot problem [16].	8
2.3	Few-Shot Learning problem solved by an embedding model [16].	9
2.4	Deep Metric Learning [9]	14
2.5	Deep learning architecture [65]	16
2.6	Convolution of kernel and a function [70].	17
2.7	Pooling process by a 2x2 window [65]	18
2.8	Unit circles for p-values in Minkowski distances [28]	20
2.9	Dynamic Time Warping of two sequences [28]	22
2.10	Cost matrix with the optimal warp path [86].	23
2.11	Cost Matrix in Fast Dynamic Time Warping [86]	24
2.12	Confusion Matrix [93]	25
3.1	Matching network in the few-shot scenario [19]	27
3.2	Prototypical networks in the few-shot scenario [42].	29
3.3	Relation network in the few-shot scenario [43].	32
4.1	The pipeline of the Few-Shot Classification	34
4.2	Shows the technique of windowing a data sample. Windows are extracted from the data sample to create multiple samples from a single sample [103]. In general this method is used for computer vision models and not for time series data. Nevertheless this technique generates valuable results and can be used for the data set	36
4.3	Shows the technique of the sliding window mechanism. Windows are created at a distance of a certain overlap factor [103]. The window size must be selected beforehand to create segments with the same size. The higher the overlap factor, the more window segments can be created from the existing time series data.	37
4.4	The raw time series data of 3 different orientation X-Y-Z axis.	39
4.5	The preprocessed time series data of 3 different orientation X-Y-Z axis. . .	40

5.1	Train loss and accuracy of the Prototypical Network trained on datasets 1, 2 and 3. The parameters for the experiments are listed in detail in Table 4.6.	50
5.2	Train loss and accuracy of the Matching Network trained on datasets 1, 2 and 3. The parameters for the experiments are listed in detail in Table 4.6.	50
5.3	Metrics for the performance evaluation of the Prototypical and Matching Network. The parameters for the experiments are listed in detail in Table 4.6.	51
5.4	Metrics for the performance evaluation of the Prototypical and Matching Network for all datasets. The parameters for the experiments are listed in detail in Table 4.6.	51
5.5	Confusion Matrix of different datasets. The parameters for the experiments are listed in detail in Table 4.6.	52
5.6	Training loss and accuracy on different shots for the support and query set. The parameters for the experiments are listed in detail in Table 4.6.	53
5.7	Metrics for the performance evaluation on different shots for the support and query set. The parameters for the experiments are listed in detail in Table 4.6.	53
5.8	Confusion Matrix on different shots for the support and query set. The parameters for the experiments are listed in detail in Table 4.6.	54
5.9	Train loss and accuracy trained on different learning methods. The parameters for the experiments are listed in detail in Table 4.6.	54
5.10	Confusion Matrix of different learning methods. The parameters for the experiments are listed in detail in Table 4.6.	55
5.11	Train loss and accuracy were evaluated on 4 different distance/similarity metrics on the Prototypical Network. The parameters for the experiments are listed in detail in Table 4.6.	56
5.12	Train loss and accuracy were evaluated on 3 different distance/ similarity metrics on the Prototypical Network. The parameters for the experiments are listed in detail in Table 4.6.	56
5.13	Train loss and accuracy were evaluated on 4 different distance/ similarity metrics on the Matching Network. The parameters for the experiments are listed in detail in Table 4.6.	57
5.14	Train loss and accuracy were evaluated on 3 different distance/ similarity metrics on the Matching Network. The parameters for the experiments are listed in detail in Table 4.6.	57
5.15	Comparison between the train loss and accuracy with different metrics used in Prototypical Network and Matching Network. The parameters for the experiments are listed in detail in Table 4.6.	58

5.16	Metrics for the performance evaluation on different distance/similarity measures. The parameters for the experiments are listed in detail in Table 4.6.	58
5.17	Confusion Matrix of different distance/similarity measures. The parameters for the experiments are listed in detail in Table 4.6.	59
8.1	Shows the sample split for data set 1, which is split into a training set(left) and a test set(right). The training set includes samples from process 0 to 4, the test set from process 0 to 14. It can be seen that the dataset is unbalanced even though data augmentation was applied, which is a challenge for training. For some processes there were only good examples but no bad ones.	63
8.2	Shows the sample split for data set 2, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 1, and the test set contains fewer samples from processes 0 to 14 than data set 1.	64
8.3	Shows the sample split for data set 3, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 2, and the test set contains fewer samples from processes 0 to 14 than data set 2.	64
8.4	Shows the sample split for data set 4, which is split into a training set(left) and a test set(right). The training set includes samples from process 0 to 4, the test set from process 5 to 14.	65
8.5	Shows the sample split for data set 5, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 4, and the test set contains fewer samples from processes 5 to 14 than data set 4.	65
8.6	Shows the sample split for data set 6, which is split into a training set(left) and a test set(right). The training set contains fewer samples from processes 0 to 4 than data set 5, and the test set contains fewer samples from processes 5 to 14 than data set 5.	66

List of Tables

4.1	The chosen preprocessing and augmentation parameters to generate the final data set for the experiments	37
4.2	The table shows the created datasets for the evaluation of the FSL-Methods with two classes (good and bad). The training set consist of data from operation 00 to 04 (5 different process). The test set consist of data from operation processes 00-14 or 05-14. The number of the training data decreases in the dataset.	42
4.3	The table shows the chosen training parameters.	43
4.4	The table shows the selected values for the n -way k -shot classification. Each data set contains 2 classes (good and bad). The shots for the support and query set vary between 3 and 7.	44
4.5	The table shows the architecture of the Convolutional Neural Network(CNN). The properties of the different layers are described in more detail in section 2.5.	45
4.6	The table listed the different experiments on which the FSL method is evaluated in scope of this work.	47
5.1	The table shows the results of the performance metric for the experiments listed in Table 4.6.	49