Freie Universität Berlin

Master's Thesis at the Department for Informatics and Mathematics

Dahlem Center for Machine Learning and Robotics

Classification of Unseen Categories Through Few-Shot Learning for Garment Sorting Processes

Adriana Pinto

Student ID: 5261537

adriana.pinto@fu-berlin.de

Advisor: Dr. Ricardo Carrillo

Reviewer: Prof. Dr. Daniel Göhring

Berlin, October 6, 2022

Abstract

The garment industry is one of the most pollutant and prominent producers of waste on the planet. Only a low percentage of the used textile resources are reused to manufacture new clothes, and second-hand markets are becoming more saturated, causing that clothing that can still be worn to be discarded. A crucial alternative for the garments industry to reduce the environmental impact is closed-loop recycling; however, there are still challenges, such as the automation of sorting processes, that need to be tackled to enable circularity. This thesis is developed within the cooperation framework of the Freie Universitaet Berlin, the Technische Universitaet Berlin, and the circular fashion company to support CRTX. CRTX is a collaborative project that researches solutions to automate the sorting of used garments for high-quality purposes and to support human sorters to achieve a fine-grained classification. During the sorting process, previously unseen garment categories may appear that need to be classified. This work explores a meta-learning approach, which recognizes new classes from only a few labeled examples of each class, as an alternative to classify such categories. Results show that these methods are scalable to new classes and robust to imbalanced datasets, closer to real-world conditions. For the experimentation, a Machine Learning pipeline was built using state-of-the-art tools, which also contributes to the objective of an eventual system deployment for production-level serving.

Contents

List of Figures

List of Tables

1 Introduction

The first part of this section, Motivation, briefly introduces the current state of the garment industry's environmental impact, some initiatives to mitigate it, and efforts being developed to support a circular fashion economy. The second part outlines the contribution of this thesis. Furthermore, the third section, Structure of the Thesis, describes how this work is organized.

1.1 Motivation

The garment industry is one of the largest producers of waste and one of the greatest polluters on the planet. Every year, over 100 billion garments are produced, yielding over one billion tons of CO2 emissions; however, less than 1% of these textile resources are reused to manufacture new items [10]. According to Mckinsey estimations, consumers discard low-priced garments after just seven or eight wears[38]. At the same time, second-hand markets are increasingly saturated, causing clothing items to be down-cycled instead of reused.

Manufacturing clothes requires large amounts of water; part of this freshwater is used for the garments dyeing and finishing process, and the other part for growing cotton. It can take up to 200 tons of freshwater per ton of dyed fabric and up to 20.000 liters to produce 1 Kg of cotton [8]. Furthermore, the fashion industry is generating vast amounts of greenhouse gasses not only for clothing production but also for the transportation of the garments sold yearly. In addition, textile production requires large quantities of hazardous chemicals that land in the environment and worker communities. In 2011 Greenpeace launched a campaign to address the problem of toxic chemicals that aimed to achieve zero discharges of these substances [49]. Eighty companies and suppliers joined the campaign and committed to reaching zero emissions by 2020. Assessments conducted after the deadline showed that many of the involved brands managed to eliminate hazardous chemicals from over 90% of their facilities. Despite the positive results, there is still much work to be done. Since 2014, this campaign has also sought to tackle the problem of over-production and waste, encouraging brands to take responsibility for the entire lifecycle of their clothing production by "slowing the flow and closing the loop." According to [49], assessments on this aspect are not positive, and extreme overproduction results in large quantities of garments not being sold, some of which end up being destroyed, which keeps the fashion industry as a significant contributor to the global climate crisis.

The textile industry has grown significantly in the last two decades due to the rise of "fast fashion." This business model has led to "disposable fashion," which means clothes are worn only a few times before being discarded, leading to millions of tons of textile waste every year; most of this amount ends up in a landfill or is incinerated [49].

Closed-loop recycling is a crucial opportunity for the global fashion industry to reduce its environmental impact. Closed-loop systems allow the recycling of material over and over so that they can remain in constant circulation. However, challenges still need to be solved before circularity becomes a sustainable solution: less than half of used garments are collected for reuse or recycling, and less than 1% are recycled to

produce new clothes [49]. The reason behind this is that textiles are made from mixed fabrics, which complicates the separation process.

Another critical challenge is collecting and sorting used clothes [16]. Waste companies and brands have been working to move from manual to automated sorting at scale that allows the processing of more significant amounts of garments. In this regard, the Freie Universitaet Berlin, the Technische Universitaet Berlin, and the circular fashion company are working together within the framework of CRTX project to research a solution that enables automatic sorting using artificial intelligence and spectroscopy solutions. CRTX's mission is to close the gap between the collection of used garments and specific sorting for second-hand and fiber-to-fiber recycling [10].

1.2 Contribution

This thesis aims to help CRTX's efforts of allowing used garments to find their optimal channel for reuse or recycling. Concretely, this project explores an alternative to traditional classification methods using a meta-learning approach that allows classifying novel clothing categories using only a few labeled examples. Moreover, a Machine Learning pipeline was built to help automate the machine learning workflow and simplify an eventual deployment to a production-level environment.

1.3 Structure of the Thesis

The section Theoretical Background consists of six parts: the first five parts introduce the concept of Meta-Learning, and Few-Shot classification, and present the formulation and terminology adopted in this work; the sixth part, Related Work, is split into three parts to outline recent meta-learning research works in the realm of few-shot image classification and practical applications on imbalanced datasets.

The third section, Implementation, describes the dataset, model, and machine learning pipeline definition.

The fourth section, Experiments and Results, provides the experimental settings and the trained models' results.

The last section, Conclusions and Future Work, discusses the final results, gives an insight into how the results can be improved, and outlines a suggested approach that can be followed for model industrialization.

2 Theoretical Background

This chapter introduces meta-learning and few-shot learning concepts in the realm of image classification. Furthermore, some literature related to this work is reviewed, providing the salient aspects of each, and presenting the current state of the image classification through few-shot learning methods.

2.1 Meta-Learning

The meta-learning concept arises from the idea of creating a machine learning approach that resembles human learning. Humans quickly learn new concepts and skills after exposure to one or a few examples. In contrast, standard machine learning systems require a large number of examples for training in order to generalize well; furthermore, it focuses on solving one particular task.

Meta-learning or learning to learn [40] [29], is an approach that allows a system to quickly learn new tasks based on experience from previous related learning tasks [46]. For instance, a simple machine learning classifier learns a single classification task; meta-learning instead acquires knowledge of the learning process to solve a classification task by exposing itself to several similar classification tasks, assuming these tasks share the same structure.

Generally, meta-learning systems learn at two levels: within (rapid learning) and across tasks (gradual learning). First, the model learns to classify within a particular dataset, and next, it learns gradually across tasks by gathering knowledge on how the task structure changes across tasks.

Meta-learning differs from similar approaches, such as multi-task and transfer learning. In multi-task learning, a model is trained in parallel over multiple tasks while using a shared representation [7]. On the other hand, transfer learning involves training a model on a single task in the source domain where sufficient training data is available and then retrains or fine tunes on another task in the target domain. In this way, the target task leverages knowledge from the source task [33].

Meta-learning has recently increased its popularity due to its high performance in solving few-shot learning problems such as object detection [18], [51], image classification [43], [14], [9], semantic segmentation [11], [51], and reinforcement learning [13].

2.2 Meta-Learning Problem Definition

The meta-learning formulation adopted for this thesis is defined in [37] with some variations. Table 1. lists the notation and terminology used in meta-learning.

In the traditional supervised machine learning setting, the main goal is to solve a task T. Given a train and test dataset; the goal is to optimize parameters θ on the training set and evaluate the generalization performance on the test set. The learning process occurs by minimizing any loss function L. In contrast, meta-learning trains a model on a set of tasks to attempt to solve novel tasks, usually after seeing only a few annotated samples.

Meta-learning deals with three meta-sets: meta-train, meta-validation, and meta-test sets (D_{train}, D_{val}, and D_{test}). Each meta-set is a collection of tasks, where each task

is a split of labeled examples called support-set S and a set of unlabeled examples called query-set Q. Assume a task T_i from the meta-train set; the support-set is used as the training set for that particular task, and the query-set corresponds to the test data for T_i. Concretely, during training, the meta-learner uses one of its support-sets to produce a classifier or learner that achieves high classification performance on its corresponding test set query-set. More formally, given data from a distribution of tasks $P(T)$ quickly solve new tasks. Each task is associated with a dataset D_i where D_i={S_i, Q_i} which is independent and identically distributed (i.i.d.).
T_{train}={T_1, T_2, ..., T_n} and T_{test}={T_{n+1}, T_{n+2}, ..., T_{n+k}} denote training and testing tasks and their corresponding datasets: D_{train}= {D_1, D_2, ..., D_n} and D_{test} = {D_{n+1}, D_{n+2}, ..., D_{n+k}} the goal is to approximate the function f with parameters θ as follows:

$$y \approx f(S_i, x; \theta)$$

Where $(x, y) \in Q_i$. For a task $T_i \sim P(T)$, the meta-learner learns parameters θ such that its performance on its test data Q_i is optimal given its training data S_i.

2.3 Few-shot Learning

Few-shot learning is an instantiation of meta-learning in the field of supervised learning. Given a classification task T with input x and output label y, the goal is to approximate a function f with parameters θ. This is generally possible when the training dataset has sufficient samples. However, if it is not the case that the training dataset is larger enough, it becomes difficult to approximate the function f so that it has good generalization performance over a test set. A classification problem with too few examples to learn a good model is referred to as few-shot learning.

Typically, a few-shot classification task is defined as K-shot, N-way, where N is the number of classes and K refers to the number of examples per class present in the support-set. Each meta-set D_{train}, D_{val}, and D_{test} consists of K-labeled examples for N classes. The support-set comprises $K * N$ examples, and the query-set has a given number of examples for evaluation.

The few-shot learning training follows an episode-based training strategy. The term episode was defined in [48] to describe mini-batches of tasks, where each episode simulates the few-shot task by subsampling K-shot data points and N-way classes. A classifier or meta-learner takes one support-set from the meta-train set as input to produce a classifier that obtains high performance on its corresponding set of unlabeled data points or query-set. In episodic training, an epoch is composed of a fixed number of episodes.

Below is discussed the few-shot learning setting for a few-shot classification problem; however, this approach is applicable for solving other types of problems, such as regression, object detection, image segmentation, and reinforcement learning. Figure 1 shows an example of a meta-learning setting for image classification. The meta-training set D_{train} consists of a subset of tasks T_1,..., T_n. Each task or episode is a separate dataset that consists of the support-set and the query-set. The illustration depicts a 5-way 1-shot classification task. Each task contains one example from each of the five classes in the support-set and two examples for evaluation in the query-set. The meta-val and meta-test sets are defined in the same way, but with a different set

Symbol	Terminology	Details
T	task	Unit to be learned. Corresponds to a set of images from different classes
T_{train}	meta-train tasks	Set of training tasks
T_{test}	meta-test tasks	Set of testing tasks
D_{train}	meta-train set	Set of datasets pertaining to the meta training tasks. This data is used by the algorithm to learn to learn
D_{val}	meta-val set	Set of datasets corresponding to meta validations tasks
D_{test}	meta-test-set	Set of datasets corresponding to meta testing tasks
D	episode	Set containing support and query sets for training and validation
S	support-set	Training dataset; set with labeled examples. Support set for a single task
Q	query-set	Testing dataset; set with unlabeled examples. Query set for a single task
N_S	K-shot	Number of examples from one class in the support-set
N_C	N-way	Number of classes used for the classification task
N_Q	–	number of examples from one class the in query-set

Table 1: **Main Symbols.** Meta-learning terminology, symbols, and definitions used in this work

of tasks that includes classes not present in any of the tasks in the other meta-sets. Similar to the traditional machine learning paradigm, the meta-validation set is used to monitor the model's generalization performance, and the meta-test set provides an unbiased evaluation of the final model.

2.4 Meta-learning Approaches

Meta-learning methods can be categorized into optimization-based, metric-based, and model-based methods. Optimization-based models aim to find a set of optimal parameters that can generalize to a test set using fine-tuning given a small training dataset. In particular, optimization-based models attempt to optimize a function f_θ with parameters θ on limited training data to achieve a good generalization performance. Unlike a typical supervised machine learning setting, optimization-based meta-learning for few-shot learning is designed to cope with a small number of training samples preventing model overfitting [23], [27], [28]. Learning involves two phases; in the first stage, a learner model f_θ is trained for a given task and is task-specific; afterward, a meta-learner model g_ϕ is trained on a distribution of tasks and is not task-specific. During episodic training, the meta-learner learns ϕ to update the learner model's parameters θ on the training set. The meta-learner model produces updated learner model parameters θ^* such that θ^* are better than learner model parameters θ [34].

$$\theta = g_\phi(\theta, D_{train})$$

Finn et al. [13] proposed MAML, a Model-Agnostic Meta-Learning algorithm; the model is trained to generalize well on a new task with a few iterations of gradient descent steps and a small number of data points from that task. Optimization-based approaches typically require second-order optimization, hence requiring high computational resources and are memory intensive. Furthermore, supervised deep learning based on large datasets requires many weight updates, which makes the training slow due to the parametric aspect of the model. In contrast, the metric-learning-based approach allows a faster classification of novel examples as some (i.e., nearest neighbors) do not require any training but performance depends on the chosen metric. Metric-based methods use a function to embed training and test datasets and then measure the similarity between them using a distance metric [48], [42]. The goal of metric-based methods is either to learn an embedding function, usually a neural network with parameters θ_1, given a differentiable distance function d, i.e., Euclidean distance, or to learn both the embedding function and the distance function parameterized by another neural network with parameters θ_2, by harnessing meta-learning architecture.

In the third category, model-based methods make no assumptions on the form $P_\theta(y|x)$; instead, these methods involve architectures that allow fast learning, namely, parameters that are updated rapidly with few training steps [39], [3]. Depending on the model architecture, this method is further categorized into memory-based, rapid-adaptation-based, and miscellaneous models. In [39] authors proposed a model architecture MANN Memory Augmented Neural Network that uses external memory storage (a modified Neural Turing Machine, NTM) to facilitate the neural network's learning process. The memory component acts as a buffer that storages information

Figure 1: **Example of Meta-learning Setting for Image Classification.** The meta-training set D_{train}, meta-validation set D_{val} and meta-testing set D_{test} are composed of two subsets, the support-set inside the green box and query-set inside the blue box. Each meta-set consists of several episodes or tasks T_i. The illustration depicts a 5-way 1-shot classification task. Each task contains one example from each of the five classes in the support-set and two examples for evaluation from the query-set.

generated by the neural network so that it can be retrieved in the future; for example, in a few-shot classification setting, this memory allows the model to rapidly incorporate new data and leverage it to make accurate predictions after a few samples.

2.5 Learning on Class Imbalanced Datasets

In real-world settings, imbalanced datasets prevail [50]. Imbalance occurs when there is an uneven distribution of classes; the majority class contains more samples than other classes, called the minority classes. Deep learning methods typically perform well over various tasks; however, most algorithms work under the assumption of a uniform distribution over each category. Such models are trained using artificial balanced datasets, in which categories are evenly represented with numerous labeled images. Dealing with an imbalanced classification task entails that the minority class is hard to predict because there are few examples; hence, it is more challenging for a model to learn characteristics from that class and differentiate it from the majority class(es).
One naive approach to the class imbalance issue is to collect more annotated examples for the minority class; however, in some cases this approach may be complex due to privacy reasons [19]; furthermore, collecting human-annotated data is an extremely time-consuming and cost-intensive task. Classical rebalancing techniques such as oversampling and undersampling can help to overcome the problem. Alternative approaches such as weighted loss attempt to tackle class imbalance by using loss functions that assign higher or lower weights depending on the number of samples associated with the class [35]. In [31] authors evaluate the impact of imbalanced class datasets for few-shot learning methods, and results suggest that FSL methods are robust against imbalance.

2.6 Related Work

2.6.1 Few-shot Learning Classification

Few-shot learning aims to solve new tasks using only a small number of labeled examples. In the standard supervised machine learning paradigm, models are trained on large datasets and thus do not generalize well for new concepts in the presence of scarce data.

In [20] Koch et al. used a Siamese Network to approach a one-shot learning problem. The main goal is to train a Siamese Network to predict whether or not two images belong to the same class. At meta-test time, an image in the test set is compared to each image in the train set and predicts the class with the highest probability. Siamese Networks were introduced in the '90s by Bromley et al. [5] as an image matching problem to solve signature verification. Siamese Networks consist of two identical neural networks whose parameters are bound; this ensures each network maps two similar images in a near feature space. At training time, each neural network receives distinct inputs, learns to measure the similarity between pairs, and subsequently performs nearest neighbors classification using the learned metric. These discrepancies in training and testing time pose a drawback in the algorithm, as the initial embedding function is trained to maximize performance on a different task.

In [48] authors proposed a metric-learning-based framework called Matching Networks for one-shot classification. Unlike Siamese Networks, this method works under the machine learning principle: train and test conditions must match. The algorithm combines embedding and nearest neighbor classification into an end-to-end differentiable classifier and starts by mapping input-label pairs (x, y) from a small support-set of samples to a classifier that, given a test example, defines a probability distribution over outputs. In other words, the model predicts classes for the unlabeled examples in the query-set using an attention mechanism over a learned embedding of the labeled set (support-set). Matching networks can also be interpreted as an embedding space in which a weighted nearest-neighbor classifier is applied. However, for the few-shot learning setting, the algorithm will create a new different embedding vector for every example of the same class present in the support-set, and for the classification will perform comparisons independently using the cosine distance so that data points with the same class are treated as if they were different classes.

Snell et al. [42] extended the work from [48] by using Euclidean distance and proposed a method called Prototypical Networks that leverages class information across images to predict more accurately test examples. This methodology learns a metric space where a classification task can be performed by first computing class representations or prototypes through an embedding function and then computing distances to those prototypes. The algorithm uses a neural network to map all inputs in the support-set into an embedding space, then computes the mean of all support-set examples in the embedding space and assigns it as the class's prototype. Query points are embedded using the same embedding function as for the support points, then classification for those examples is carried out by finding the nearest class prototype. Unlike matching networks, prototypical networks produce a linear classifier using Euclidean distance rather than yielding a weighted nearest neighbor classifier.

Other related methodologies learn both the embeddings and the distance metric. Rather than using a fixed pre-specified distance metric such as Euclidean or cosine distance to perform classification, [44] Relation Network (RN) authors suggested learning a non-linear metric. RN consists of two modules: an embedding module, which produces feature maps for support, and a query example that is then concatenated and passed to a relation module that computes a relation score representing the similarity between the two data points.

In the presence of coarse classes with diverse variations within each class, computing the mean embedding of examples to have a single representation for each class may not be optimal. In [2] authors proposed an approach that learns a mixture of prototypes that allows the representation of more multimodal class distributions; instead of having a single prototype per class, a set of clusters represents a class.

For even more complex relationships between the different classes, authors in [14] developed a meta-learning approach that uses graph convolutions to perform message passing on embeddings to refine representations of each class.

2.6.2 Class-imbalanced Datasets

While classification performance in standard supervised deep learning methods is adversely affected by class imbalance [6], few-shot learning methods appear to over-

come the issue. In [31] Ochal et al. Studied the impact of imbalanced distributions of data on few-shot learning tasks. The results of the experiments show that few-shot learning methods are highly effective under imbalance conditions compared to other approaches whose performance decrease by up to 17%. Authors in [36] developed a few-shot learning framework for dermatological image classification using a long-tailed class distribution dataset. Results displayed strong generalization capabilities in the presence of very few training data points.

2.6.3 Practical Few-shot Learning Applications

In [36] authors used a variant of Prototypical Networks called Prototypical Clustering Networks (PCN) to assist doctors in dermatological diagnosis. Diagnosing skin conditions poses several challenges; first, the amount of data available is scarce due to privacy policies, and second, the data is commonly long-tailed since rare skin conditions are poorly recorded, and other common conditions easily diagnosable are not recorded. Another major challenge is the intra-class variability, for example, the body part where a single disease occurs, the skin type, among others. PCN learns a mixture of prototypes that tackles the issue of intra-class variability. Furthermore, unlabeled support examples are incorporated via k-mean on the learned embedding.

3 Implementation

This section presents the implementation of the few-shot learning model. First, the dataset used for training and evaluation tasks is described, along with the steps required to adapt it for model use. Afterward, the model training and the developed machine learning pipeline are explained.

For this project, garments classification was formulated as a few-shot learning problem. The algorithm used to tackle this classification problem was Prototypical Networks [42] and experiments were performed on the DeepFashion dataset [26]. Prototypical Networks is a metric-based algorithm based on the concept that it exists an embedding in which several points cluster around a single prototype representation for each class [42]. Prototypical Networks algorithm uses an embedding function to encode each support input into an embedding space and compute class prototypes for every class as the mean of its embedded support-set. To perform classification for a given query point x, the point is embedded using the same function as for the support-set and then finds the nearest class prototype by computing the distance between x and all class prototypes.

3.1 Dataset

Among some fashion datasets Fashionpedia [17], ModaNet [52], DeepFashion2 [15], and DeepFashion [26], DeepFashion contains the most abundant annotated clothing categories. For meta-learning classification purposes, it is essential to have sufficient classes [12] so that at training time the algorithm leverages knowledge from enough tasks and finds common structures among them to be able to generalize.

The DeepFashion dataset is a large-scale clothes database used for category and attribute prediction, collected by the Multimedia Laboratory at The Chinese University of Hong Kong. The DeepFashion database is composed of several datasets; for this thesis, the category and attribute prediction benchmark dataset was used. This large subset of DeepFashion originally contains 50 different clothing categories, and after merging four different clothing types into a single category, this results in 46 categories with 289,222 images. The images are also split into three super categories: upper-body, lower-body, and full-body. Each image is annotated by a bounding box and a clothing type or category.

DeepFashion dataset poses one major challenge for traditional supervised machine learning classification as the data distribution is extremely long-tailed (see Figure 2). Some garment types are rare and may not have many pictures on shopping websites or other common sources for collecting clothing datasets. In the DeepFashion dataset, the number of images per category ranges from 17 to 70.000 pictures. Notably, the category Dress represents about 24% of the dataset, and Halter and Coverup categories represent less than 1% of the entire dataset. The median number of images in the dataset corresponds to 769.

Another challenge may be the intra-class variability; for example, the category Dress contains long-sleeve, short-sleeve, mini, midi, or maxi dress, among others. See Figure 3.

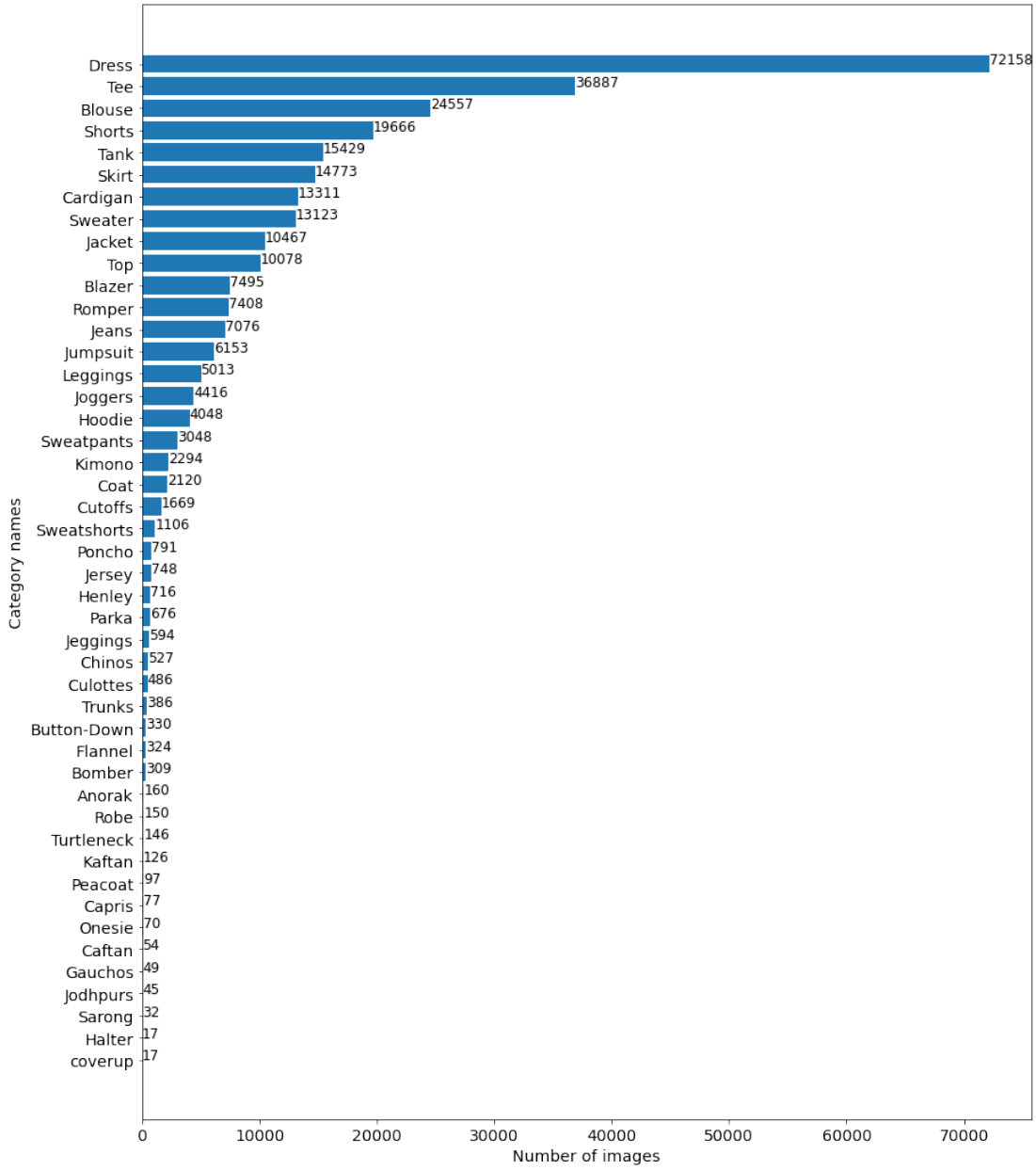


Figure 2: **Distribution of Dataset Classes.** Long-tailed class distribution of DeepFashion dataset.

Figure 3: **Example of Intra-class Variability.** The image illustrates intra-class variability for the "Dress" category.

To train the model, the dataset was converted to COCO format [25] and then to TFRecords format. This format stores a sequence of binary records, which makes large datasets take less space on disk, and less time to copy, and the lecture of files from disk is more efficient [32]. Images were cropped around the bounding boxes to discard any object or noise in the picture that was irrelevant to the learning. Moreover, the dataset is split into three sub-datasets: meta-training, meta-validation, and meta-testing. Unlike traditional machine learning methods that require splits to contain examples from all available categories in the dataset, few-shot learning classification requires disjoint splits. Categories were split into 27, 10, and 9 for meta-training, meta-validation, and meta-testing equivalent to the split used in [37]. The different categories belonging to each split can be seen in Table 3 in the Appendix. Categories containing the majority of samples were assigned to the training split, and categories with the least number were assigned to validation and testing splits.

3.2 Model

As discussed in section Meta-Learning Problem Definition, in meta-learning, each meta-set D_{train}, D_{val}, D_{test} contains multiple datasets D, where each D is split into support S and query Q sets, respectively. These datasets D are called Episodes. In particular, a training epoch consists of E episodes. A training episode contains K number of examples in the training set and N number of classes. $N_C \leq N$ is the number of classes per episode. Each episode is built by randomly selecting a subset of classes from the meta-training set, then N_S samples and N_Q samples per class are chosen uniformly at random without replacement to be the support and query sets. Given a small support-set S with N labeled examples such that, $S = (x_1, y_1), ..., (x_N, y_N)$ Where each x_i corresponds to a D-dimensional feature vector of an example and $y_i \in 1, ..., K$ is the corresponding label. S_k denotes the set of examples labeled with class k. Prototypical Networks learn a non-linear mapping of the input into an embedding space using a function f_ϕ with learnable parameters ϕ, see Figure 4. This function f_ϕ is a 4-layer convolutional neural network. A prototype feature vector c_k is computed for every class as the mean vector of the embedded support points belonging to this class.

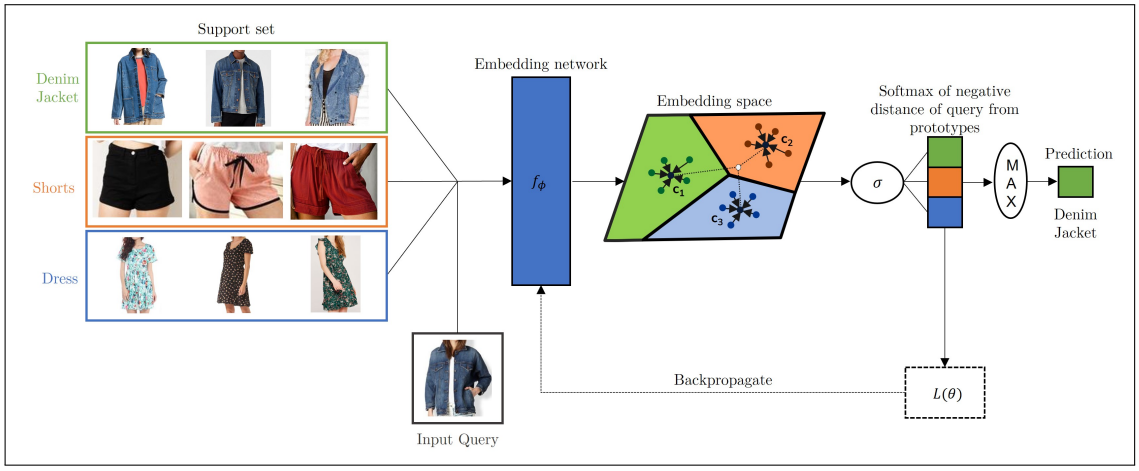


Figure 4: **Prototypical Networks.** PN computes prototypes as the mean of embedded support samples for each class. Query points are embedded using the same function and are classified via softmax over distances to the class prototypes.

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

For a given query point x the similarity is measured by computing the distance d of the embedded query point x and each class prototype. The distance function d can be any differentiable distance function. The output probability over classes is calculated by taking a softmax over the negative distances.

$$P(y = c|x) = softmax(-d(f_\phi(x), c_k)) = \frac{exp(-d(f_\phi(x), c_k))}{\sum c' \in C exp(-d(f_\phi(x), c_{k'}))}$$

Learning proceeds by minimizing the negative log-likelihood of the true class k via Stochastic Gradient Descent.

$$L(\theta) = -logP_\phi(y = k|x)$$

The embedding function consists of 4 convolutional modules. Each module is composed of a 2-D convolutional layer, with 64 filters of 3x3 size, a batch normalization layer, activated by a ReLU nonlinearity, and a max-pooling layer of 2x2 size. See Figure 5.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(100, 84, 84, 64)	1792
batch_normalization_4 (Batch Normalization)	(100, 84, 84, 64)	256
re_lu_4 (ReLU)	(100, 84, 84, 64)	0
max_pooling2d_4 (MaxPooling 2D)	(100, 42, 42, 64)	0
conv2d_5 (Conv2D)	(100, 42, 42, 64)	36928
batch_normalization_5 (Batch Normalization)	(100, 42, 42, 64)	256
re_lu_5 (ReLU)	(100, 42, 42, 64)	0
max_pooling2d_5 (MaxPooling 2D)	(100, 21, 21, 64)	0
conv2d_6 (Conv2D)	(100, 21, 21, 64)	36928
batch_normalization_6 (Batch Normalization)	(100, 21, 21, 64)	256
re_lu_6 (ReLU)	(100, 21, 21, 64)	0
max_pooling2d_6 (MaxPooling 2D)	(100, 10, 10, 64)	0
conv2d_7 (Conv2D)	(100, 10, 10, 64)	36928
batch_normalization_7 (Batch Normalization)	(100, 10, 10, 64)	256
re_lu_7 (ReLU)	(100, 10, 10, 64)	0
max_pooling2d_7 (MaxPooling 2D)	(100, 5, 5, 64)	0
flatten_1 (Flatten)	(100, 1600)	0

Total params: 113,600
Trainable params: 113,088
Non-trainable params: 512

Figure 5: **Embedding Architecture**. The architecture is composed of four convolutional modules. Each module consists of a 64-filter 3x3 convolution, a batch normalization layer, followed by a ReLU nonlinearity, and a 2x2 max pooling layer.

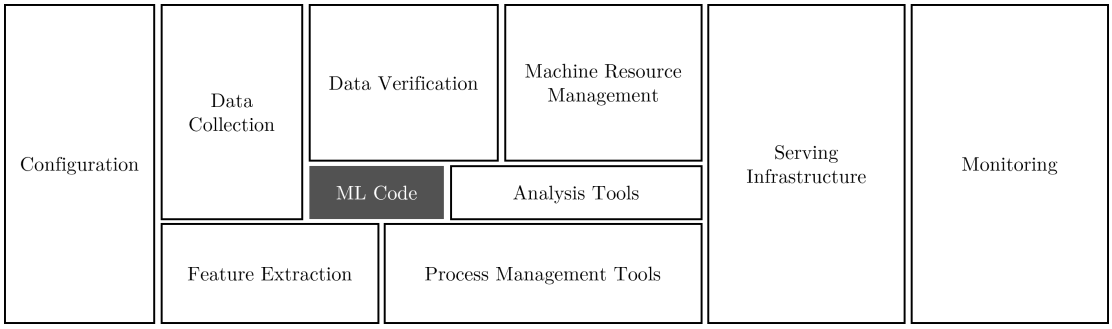


Figure 6: **Machine Learning System Components.** Representation of ML code and surrounding activities in the biggest context of a production-ready solution [41].

3.3 Machine Learning Pipeline

3.3.1 MLOps Workflow

MLOps stands for Machine Learning Operations, which is the combination of Machine Learning and a set of practices that aim to reduce the gap between software development and operations [45]. MLOps attempts to automate Machine Learning processes and brings ML models to production.

For the purpose of the project, state-of-the-art platforms were harnessed to automate and optimize the Development and Training lifecycles. Traditionally, only a fraction of the ML systems pertains to training routines, surrounded by the "plumbing" [24] of core capabilities required to bring the resulting models to a successful stage see Figure 6.

According to [4] the development of industry-ready Machine Learning solutions follows an iterative process consisting of four major stages:

- Data Management, focusing on dataset retrieval and preparation
- Model Learning, executing the training activities, and hyperparameter optimization
- Model Verification, validating the model with an evidence-based approach
- Model deployment, integrating, and operating the model within a fully-fledged system.

In [30] the author proposes a different machine learning lifecycle, forked from traditional software development and defines the following stages:

- Project Scope, defining the problem and the project
- Data Collection, preparing, processing, and establishing a data baseline
- Model Training, training the model and executing error analysis
- Production Deployment, industrializing, and monitoring of the model

This project follows the lifecycle definition shown in Figure 7 and tackles the classification problem with a model-centric AI approach, where the performance improvement effort focuses on model and hyperparameter tuning. In contrast, a data-centric approach explores data modifications to improve performance [30].

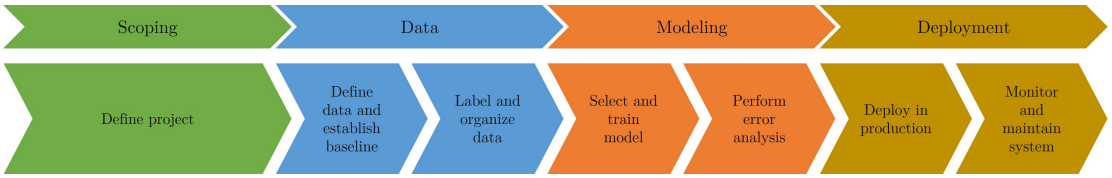


Figure 7: **Machine Learning Lifecycle.** Four major stages in the machine learning lifecycle as defined in [30].

3.3.2 Pipeline Definition

Based on a model-centric approach, a training and experimentation lifecycle was defined, executing hyperparameter and model optimizations iteratively, aiming to improve the model results while having a frozen dataset. The experimentation and iterative execution of different configurations harnessed the workflow and hardware orchestration capabilities of Kubeflow. Kubeflow is an open-source platform running on top of Kubernetes that enables the execution of ML pipelines in a portable and scalable manner [21].

Kubeflow enables developers to deploy containerized ML code successfully and to execute the different steps or stages of the ML lifecycle in distributed hardware. It offers a central dashboard, the primary way to interact with the Kubernetes cluster resources. Within this dashboard, the main platform capabilities can be found: Jupyter notebooks dynamically launched for exploration and experimentation, tensorboards for live monitoring of runs and experiments, a model repository to control versions and manage resulting models, experiments to group and analyze a set of training routines for comparison and refinement, and pipelines to manage and execute ML workflows.

Different configurations of pipeline executions can be grouped into what Kubeflow calls experiments. This structure enables comparing settings and performance of different runs and hyperparameters using Kubeflow's UI. For the project's scope, several experiments were created, each with different major versions and refinements of the meta-learning training routines.

The ML codebase takes advantage of GitHub version control and continuous integration capabilities. A GitHub workflow was configured to automatically build and push a docker container to a GitHub private container registry. This workflow is triggered on every change to the main branch. This was the first step to successfully developing and deploying an automated and portable training workflow.

The generated container is later downloaded and executed by the Kubeflow cluster, as defined in a pipeline YAML file. This file specifies how to orchestrate different container routines and storage resources to run an experiment, taking advantage of Kubeflow's component objects. A component is a self-contained routine in charge of

executing a single step in the Machine Learning Workflow [22]. These components are affected by input values and output artifacts. The input values might be user-defined and can be leveraged to specify different configurations or hyperparameters during the pipeline execution. The output artifacts help monitor the model performance and identify weaknesses in the training routines.

The resulting pipeline is classified as a DAG or Directed Acyclic Graph, and it can be executed with multiple configurations for refinement and enhancement. The ML workflow DAG consists of four steps executed in four components, as seen in Figure 8:

- Data download, which provisions the cluster local storage with the latest dataset available, downloaded directly from a cloud location
- Data processing and optimization, which processes the source data and stores the resulting TFrecords files in the artifact repository
- Model Training, which executes the TensorFlow fit routines
- Model evaluation, which executes the evaluation of the resulting model with the test split

By the end of the workflow execution, a list of artifacts is available within the cluster storage and can be downloaded for further analysis.

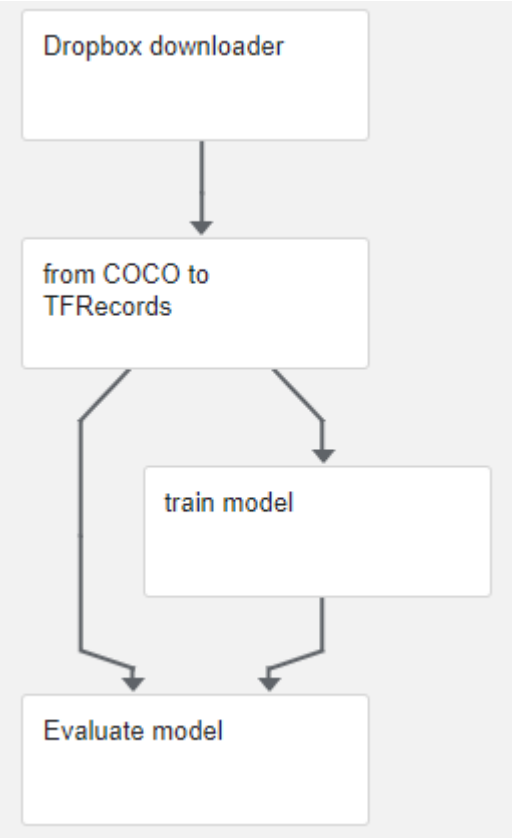


Figure 8: **Machine Learning Workflow.** Pipeline describing the machine learning workflow implemented. The figure shows the pipeline's components and how they are related to each other in a graph. This image is an extract from Kubeflow.

4 Experiments and Results

This section presents the experimental settings and the detailed experiment results.

Experiments were conducted on an on-premises Kubernetes cluster with Kubeflow ML computing capabilities built on top, owned by CRTX. The cluster hardware uses an Nvidia RTX 3090 Graphics card, a unit with 10496 CUDA cores, a 1.7GHz boost clock, and 24GB of available GDDR6X VRAM. The project harnesses this hardware to efficiently execute training routines, enabling faster iterations and model refinements.

Few-shot classification models were trained with several combinations of components, such as the number of training classes (N-way) per episode, number of images per class (K-shot), and number of query points. The performance of the model is measured in terms of accuracy. At test time, classification accuracy was averaged over 1,000 randomly generated episodes from the meta-test set.

The embedding architecture described in section Model was used to embed support and query points. Models were trained using Stochastic Gradient Descent (SGD) with Adam as the optimization method. The number of training episodes for all experiments was set to 100 episodes per epoch.

Euclidean and cosine distances were used as distance metrics; however, models show significantly better results using Euclidean distance. In [42], authors found that results improved greatly using Euclidean distance compared to cosine distance and claim this might be due to the cosine distance not being a Bregman divergence. For this reason, the analysis is centered around experiments using Euclidean distance. Furthermore, the number of images per class or K-shot at testing time was fixed at 5-shot.

Several scenarios were set up to analyze the effect of the number of training classes, the number of support and query images. Training starts with an initial learning rate of 10^{-3}, and then a step decay schedule drops the learning rate by a *drop_rate* factor every *epochs_drop* number of epochs. Several values for *drop_rate* a *epochs_drop* were tested to find the optimal learning rate. The number of training epochs is determined by performing early stopping on validation loss.

4.1 Effect of Class Size (N-way)

First experiments aimed to analyze the effect of the number of training classes controlled by the parameter N while keeping the test N-way fixed at 5, the number of training images per class at 5, and the number of query points at 15. Fixed values were chosen to follow the experimental setup defined in [42]. Experiments were run varying train N-way from 4-way up to 10-way values. 10-way was the maximum number acceptable to train the model since the meta-validation split comprises ten classes, limiting the number of training classes as both training and validation procedures require matching episodic settings.
From the experiments can not be concluded that the more classes per episode at training time, the better the model's performance, as no upward trend is seen in Figure 9a. This figure shows a significant improvement in classification accuracy when training the model with six classes per episode and then fluctuates, reaching the highest

accuracy at eight classes.

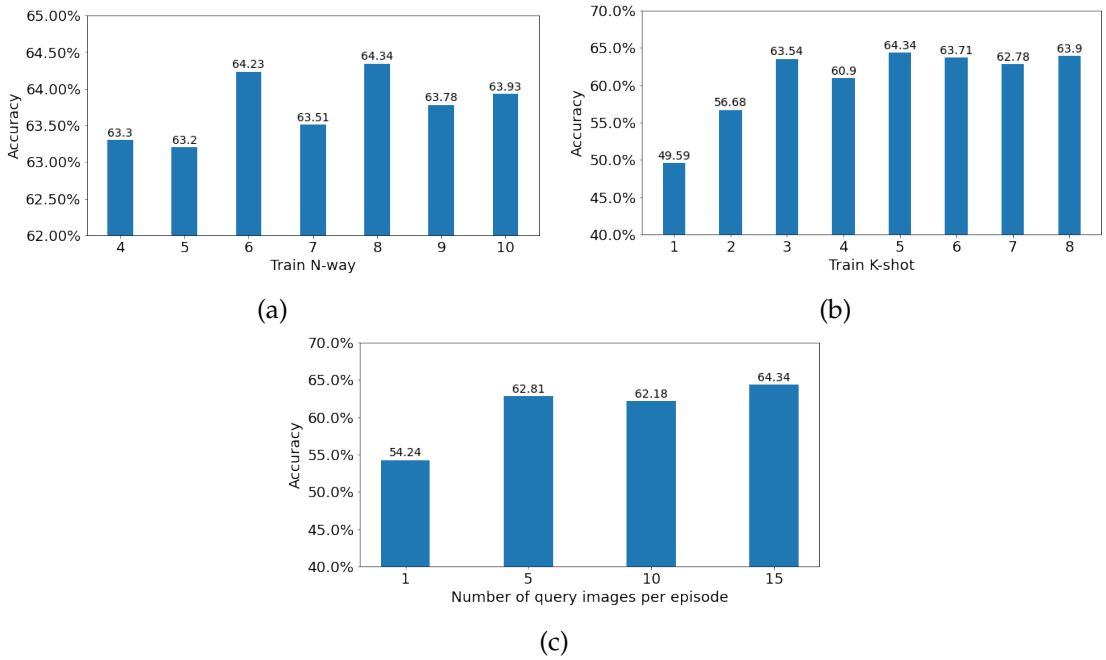


Figure 9: **Performance Comparison with Respect to Different N-way, K-shot, and Query-set Size.** Experiments to analyze the effect of the training class size N-way, support-set, and query-set size on the DeepFashion dataset. All models were tested on 5-way, 5-shot classification tasks. Performance is reported in terms of accuracy using the meta-test set, which contains previously unseen categories. The accuracy is averaged over 1,000 episodes. **(a)** Effect of the number of training classes N-way per episode with support-set size of 5 or 5-shot and 15 query points per class. **(b)** Effect of the number of images per class or K-shot with 8-way episodes and 15 query points per class. **(c)** Effect of the number of query images per episode using 8-way and 5-shot episodes.

4.2 Effect of Support Size (K-shot)

Similarly, to investigate the impact of the support-set size or number of images per class, several experiments were conducted considering 1-shot up to 8-shot classification settings. The train N-way was set to 8, test N-way at 5, and query points at 15, as per the performance in above experiments. Due to the limited number of images for some categories, the maximum number of K-shot values tested was 8.

From the figure 9b, it can be observed that model performance, presented in terms of accuracy, improves drastically when increasing from 1-shot to 3-shot values; after 5-shot, the accuracy remains almost constant. This indicates that a larger support-set can produce better prototypes for few-shot classification. From the experiments, 5-shot shows the best accuracy, achieving comparable results with the Prototypical Network's baseline [42] and 1-shot results in higher accuracy than the achieved in [42] on miniImageNet dataset [48].

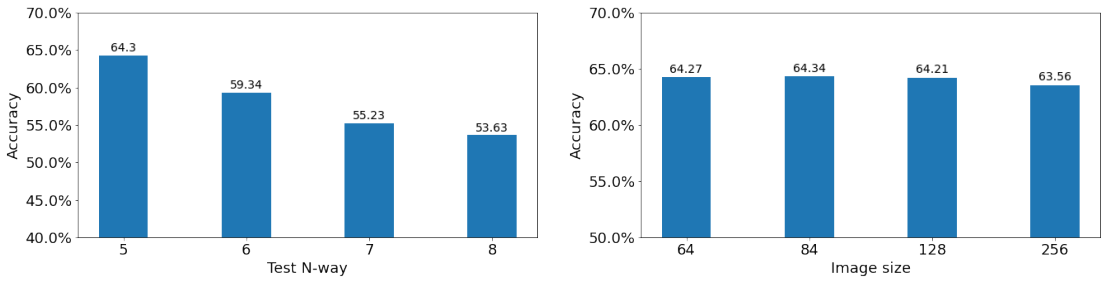
4.3 Effect of Query Size

Furthermore, the influence of the query-set size per episode was examined by training several models with a different number of query points. Here, models were trained using 8-way 5-shot tasks while changing the number of query points. The query-set size is matched at training and testing times. Results show that the larger the query-set, the higher the performance gain; hence, it can be conjectured that a higher number of query points will help the model to adapt the knowledge from the meta-training tasks, which also improves generalization ability, Figure 9c.

4.4 Further Experiments

While in Effect of Class Size (N-way) the test N-way remained fixed at 5 classes, figure 10a displays the model's performance for different test N-way values using 8-way 5-shot tasks at training. Models evidence a higher performance when the number of classes at testing time is lower than the ones used at training time. A reason may be that a larger number of test classes entails a wider variety of categories to be predicted, hence augmenting the complexity of the few-shot classification.

Moreover, variations in the image size were also tested using the 8-way 5-shot classification setting, with 15 query points, see figure 10b. Surprisingly, the size of the image does not demonstrate significant improvements to the model's performance in terms of accuracy.



(a) (b)

Figure 10: **Performance Comparison with Respect to Different Test N-way and Images Sizes.** Experiments to examine the effect of the number of test classes and image size. Performance is reported in terms of accuracy using the meta-test set which includes previously unseen categories. The accuracy is averaged over 1,000 episodes. **(a)** Effect of variations in the number of test classes while keeping the number of training classes fixed at eight or 8-way, support-set size of 5 or 5-shot, and 15 query points. **(b)** Effect of image size on model's performance. The training uses 8-way and 5-shot episodes with 15 query points.

4.5 Results

Table 2 shows the configuration of training and testing episodes that achieved the highest classification accuracy on the meta-test dataset. To see further experiments

conducted, refer to the Table 4 in the Appendix. Onwards, the model obtained with this configuration will be referred to as the best model.

Train episodes			Test episodes			Accuracy
N-way	K-shot	Query-set	N-way	K-shot	Query-set	
8	5	15	5	5	15	65,20%

Table 2: **Few-shot Configuration with the Highest Classification Accuracy.** Configuration of training and testing episodes with the best classification accuracy on the meta-test dataset.

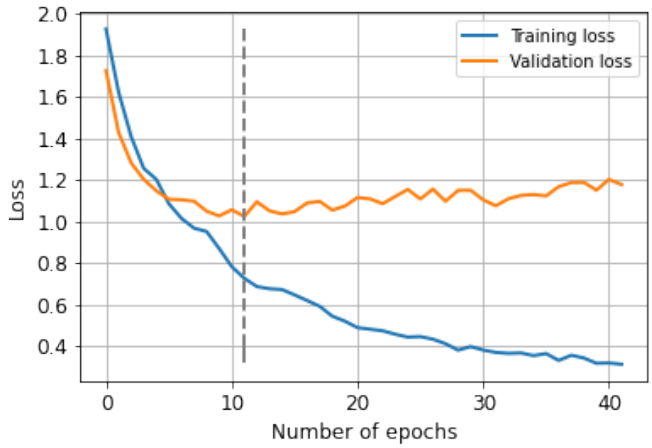
The model quickly converged to the minimum validation loss and reached its maximum validation accuracy at 11 epochs, as seen in figures 11a and 11b. Early stopping was configured on the validation loss to avoid overfitting on the meta-train set, with a patience value of 30 epochs to check for no further improvement. The weights for the best model were saved at 11 epochs.

Figure 12 showcases a sample image of the support-set for five different classes in a random episode and the five nearest images of the query-set to the class prototype. This is an extract of a random test episode. The images with labels in green represent garments correctly classified, and images with red labels indicate misclassified garments. The accuracy achieved in this episode was 72%.

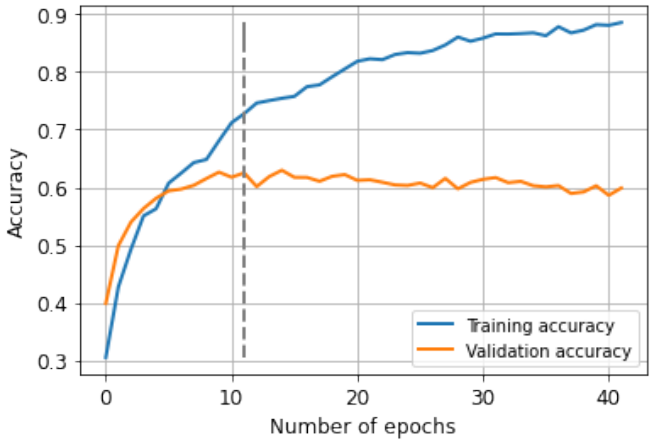
Examining the misclassified images for this episode, it can be conjectured that the model misrecognizes the garments with similar printed fabrics to the samples in the support-set. These examples may also be challenging for a human person to label correctly.

The model is tested using the meta-test set, which contains classes the model has not seen during the training or validation stages. The total classification accuracy is averaged over the total amount of episodes. The performance of the model along 1,000 episodes is summarized in Figure 13.

Figure 14 provides examples of the classes present in one of the episodes with the highest classification accuracy and samples of classes in one of the episodes with the lowest accuracy. In 14a, it is noticed that the model is good at discriminating when the episode contains a variety of classes from both the upper and lower body, all with well-defined structural features, which makes the distinction easier. In contrast, in 14b, the episode sees categories with very similar structural attributes that make the distinction harder for the model. An algorithm with the ability to represent variability in classes may capture critical features that allow a better classification.



(a)



(b)

Figure 11: **Training and Validation History of the Best Model.** **(a)** Training and validation loss of the model. **(b)** Training and validation accuracy of the model along the epochs. The dashed line represents the lowest validation loss point, where trained weights have been saved for the test results.

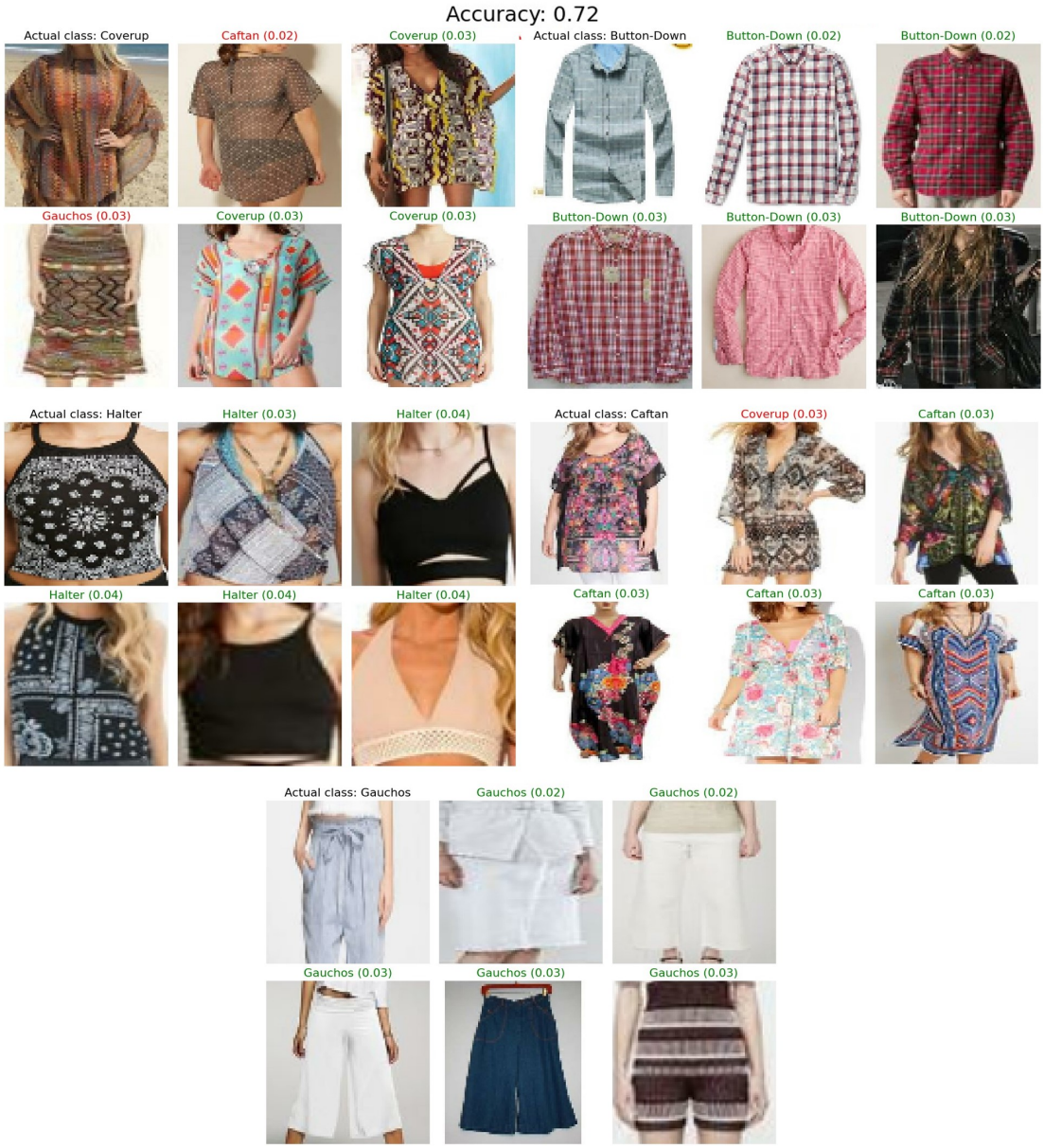


Figure 12: **Accuracy Result for a Random Test Episode on the Meta-test Set.** The figure shows a sample image (actual class) of the support-set for five different classes in a random episode and the five nearest images of the query-set to the class prototype. The images with labels in green represent garments correctly classified, and images with red labels indicate misclassified garments. Images in meta-test set were never seen by the model during training and validation stages. In parentheses, it is shown the normalized distance to the class prototype.

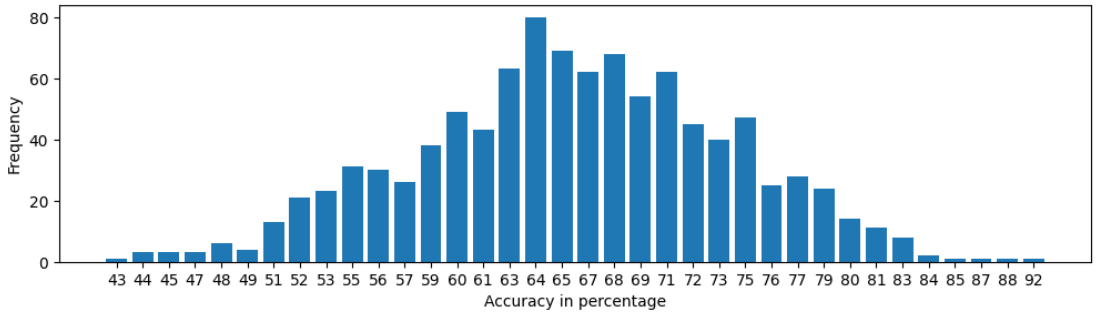


Figure 13: **Accuracy results distribution along 1,000 Randomly Generated Episodes**. The model is tested using previously unseen categories by the model. At testing time, episodes are randomly generated from the meta-test set, and classification accuracy is averaged at over 1,000 episodes.

Figure 14: **Sample Images of the Classes Present in Episodes with High and Low Performance. (a)** Sample images of 5 classes seen by the model in one of the 1,000 episodes along testing with high accuracy. **(b)** Sample images of 5 classes seen by the model in one of the 1,000 episodes along testing with low accuracy.

5 Conclusions and Future Work

This section summarizes the main results of this work and presents some ideas on how the results can be further improved. In addition, an approach for model industrialization is proposed in the future work subsection.

5.1 Discussion of Results

Emerging trends in the fashion industry constantly create new garment categories that need to be classified and for which the amount of labeled pictures is limited. Meta-learning approaches, concretely the Prototypical Networks algorithm, proves to be a promising alternative for garment classification that can be used during the sorting process due to its ability to classify previously unseen categories from a few labeled images.

Despite the model being trained under challenging conditions closer to real-world scenarios such as a class-imbalanced dataset with a limited number of categories, the model performance is comparable to the baseline (Prototypical Networks trained on miniImageNet dataset [42]) results. However, a higher performance would be required to allow the sorting process to be fully automated without human intervention.

Furthermore, the computational resource usage for meta-learning methods is much lower than traditional deep learning techniques due to the episodic nature of the training process. Also, these algorithms require smaller datasets, which reduce costs associated to data collection and data labeling. Lastly, the usage of modern tools for end-to-end machine learning execution and container-based solutions simplifies the experimentation process and output management.

5.2 Future Work

While few-shot learning techniques demonstrate to work effectively on datasets with limited classes, a dataset with a higher number of classes available for training may help the model gain more knowledge to generalize to new tasks more effectively. Jointly, a larger number of images for the minority classes might also be beneficial for the model to adapt the knowledge of these classes to improve model's generalization capacity.

The model can be further improved to perform a more fine-grained classification of image categories by learning multiple prototypes per class. Having several prototypes for a single class would allow learning a more accurate representation and tackle the intra-class variability of classes.

Furthermore, in the future, this project may be used in a production setup. Below is briefly outlined the cycle to deploy, monitor, and maintain the model.

One of the biggest challenges of the proposed model is to industrialize the training and classification process to serve CRTX's mission and vision. The principles of MLOps (MLOps Workflow) serve as a guide for further implementing an iterative methodology to gather data, execute the training and optimization of hyperparameters, and release an incremental version of the working model to CRTXs operations. It is important to emphasize that machine learning has been most successful when devised to fit narrow boundaries, specifically when the test inputs are close to the

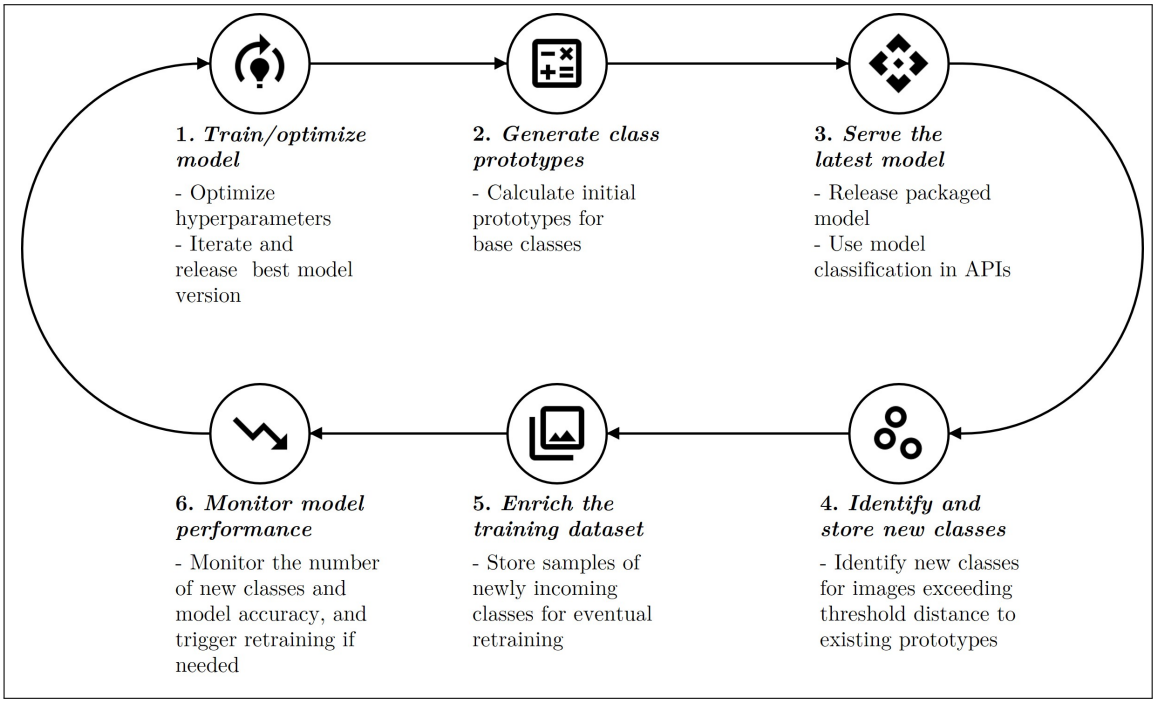


Figure 15: **Iterative Approach Suggested for Model Industrialization.** The figure shows six suggested steps to release and maintain a few-shot learning based solution in a production-level environment.

training inputs [47]. Therefore, the production model should be trained with images as close as the ones the model will see in the target setup. For this reason, continuous input and output data monitoring are essential in the industrialization process to ensure high performance and overcome challenges presented by likely data drift [1].

Figure 15 illustrates an iterative approach that can be adopted for the model industrialization. In the first stage, an optimized model is trained. Afterward, using the current dataset, generate initial prototypes for the existing classes, and store them for their use in production classification processes. Then, serve the model through APIs to execute real-time classification on sorting plant garment pictures. When a given picture has a distance greater than a predefined threshold from each prototype, request user validation, and store it as a new class prototype for further examples to be classified. Keep storing new examples and classes to enrich the current dataset. Lastly, monitor performance regularly, and when misclassification exceeds a specific limit, trigger a model retraining process. After some iterations of this cycle, the model will be robust enough to perform accurate garment classification.

References

[1] Samuel Ackerman, Eitan Farchi, Orna Raz, Marcel Zalmanovici, and Parijat Dube. *Detection of data drift and outliers affecting machine learning model performance over time*. 2020. DOI: 10.48550/ARXIV.2012.09258. URL: https://arxiv.org/abs/2012.09258.

[2] Kelsey R. Allen, Evan Shelhamer, Hanul Shin, and Joshua B. Tenenbaum. *Infinite Mixture Prototypes for Few-Shot Learning*. 2019. DOI: 10.48550/ARXIV.1902.04552. URL: https://arxiv.org/abs/1902.04552.

[3] 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*. 2016. DOI: 10.48550/ARXIV.1606.04474. URL: https://arxiv.org/abs/1606.04474.

[4] Rob Ashmore, Radu Calinescu, and Colin Paterson. *Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges*. 2019. DOI: 10.48550/ARXIV.1905.04223. URL: https://arxiv.org/abs/1905.04223.

[5] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature Verification Using a "Siamese" Time Delay Neural Network". In: *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS'93. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 737–744.

[6] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. "A systematic study of the class imbalance problem in convolutional neural networks". In: *Neural Networks* 106 (Oct. 2018), pp. 249–259. DOI: 10.1016/j.neunet.2018.07.011. URL: https://bit.ly/3xNhHty.

[7] Rich Caruana. "Multitask Learning". In: *Machine Learning* 28 (July 1997). DOI: 10.1023/A:1007379606734.

[8] Mathilde Charpail. *Fashion & environment*. Visited on 28/06/2022. URL: https://www.sustainyourstyle.org/en/whats-wrong-with-the-fashion-industry#anchor-environmental-impact.

[9] Haoxing Chen, Huaxiong Li, Yaohui Li, and Chunlin Chen. *Multi-scale Adaptive Task Attention Network for Few-Shot Learning*. 2020. DOI: 10.48550/ARXIV.2011.14479. URL: https://arxiv.org/abs/2011.14479.

[10] CRTX.ai. Visited on 06/07/2022. URL: https://crtx.ai/.

[11] Nanqing Dong and Eric P. Xing. "Few-Shot Semantic Segmentation with Prototype Learning". In: *BMVC*. 2018.

[12] Chelsea Finn. *The Meta-Learning Problem and Black-Box Meta-Learning*. Visited on 10/05/2022. URL: http://cs330.stanford.edu/fall2020/slides/cs330_metalearning_bbox_2020.pdf.

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. 2017. DOI: 10.48550/ARXIV.1703.03400. URL: https://arxiv.org/abs/1703.03400.

[14] Victor Garcia and Joan Bruna. *Few-Shot Learning with Graph Neural Networks*. 2017. DOI: 10.48550/ARXIV.1711.04043. URL: https://arxiv.org/abs/1711.04043.

[15] Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang, and Ping Luo. *DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images*. 2019. DOI: 10.48550/ARXIV.1901.07973. URL: https://arxiv.org/abs/1901.07973.

[16] Amed Imran, Berg Achim, Balchadani Anita, Hedrich Saskia, Jensen Jakob, Straub Michale, Rolkens Felix, Younga Robb, Brown Pamela, and Le Merle Leila. *The State of Fashion 2022*. Visited on 06/07/2022. URL: https://www.mckinsey.com/industries/retail/our-insights/state-of-fashion.

[17] Menglin Jia, Mengyun Shi, Mikhail Sirotenko, Yin Cui, Claire Cardie, Bharath Hariharan, Hartwig Adam, and Serge Belongie. *Fashionpedia: Ontology, Segmentation, and an Attribute Localization Dataset*. 2020. DOI: 10.48550/ARXIV.2004.12276. URL: https://arxiv.org/abs/2004.12276.

[18] Bingyi Kang, Zhuang Liu, Xin Wang, Fisher Yu, Jiashi Feng, and Trevor Darrell. *Few-shot Object Detection via Feature Reweighting*. 2018. DOI: 10.48550/ARXIV.1812.01866. URL: https://arxiv.org/abs/1812.01866.

[19] Aleksandr Kesa and Tanel Kerikmäe. "Artificial Intelligence and the GDPR: Inevitable Nemeses?" In: *TalTech Journal of European Studies* 10 (Dec. 2020), pp. 68–90. DOI: 10.1515/bjes-2020-0022.

[20] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-shot Image Recognition". In: 2015.

[21] Kubeflow. *An introduction to Kubeflow*. Visited on 25/07/2022. URL: https://www.kubeflow.org/docs/started/introduction/.

[22] Kubeflow. *Conceptual overview of components in Kubeflow Pipelines*. Visited on 26/07/2022. URL: https://www.kubeflow.org/docs/started/introduction/.

[23] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. *Meta-SGD: Learning to Learn Quickly for Few-Shot Learning*. 2017. DOI: 10.48550/ARXIV.1707.09835. URL: https://arxiv.org/abs/1707.09835.

[24] Jimmy Lin and Dmitriy Ryaboy. "Scaling Big Data Mining Infrastructure: The Twitter Experience". In: *SIGKDD Explor. Newsl.* 14.2 (Apr. 2013), pp. 6–19. ISSN: 1931-0145. DOI: 10.1145/2481244.2481247. URL: https://doi.org/10.1145/2481244.2481247.

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[26] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. "DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1096–1104. DOI: 10.1109/CVPR.2016.124.

[27] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. *A Simple Neural Attentive Meta-Learner*. 2017. DOI: 10.48550/ARXIV.1707.03141. URL: https://arxiv.org/abs/1707.03141.

[28] Tsendsuren Munkhdalai and Hong Yu. *Meta Networks*. 2017. DOI: 10.48550/ARXIV.1703.00837. URL: https://arxiv.org/abs/1703.00837.

[29] D.K. Naik and R.J. Mammone. "Meta-neural networks that learn by learning". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 1. 1992, 437–442 vol.1. DOI: 10.1109/IJCNN.1992.287172.

[30] Andrew Ng. *MLOps: From Model-centric to Data-centric AI*. Visited on 01/07/20. 2021. URL: https://www.deeplearning.ai/wp-content/uploads/2021/06/MLOps-From-Model-centric-to-Data-centric-AI.pdf.

[31] Mateusz Ochal, Massimiliano Patacchiola, Amos Storkey, Jose Vazquez, and Sen Wang. *Few-Shot Learning with Class Imbalance*. 2021. DOI: 10.48550/ARXIV.2101.02523. URL: https://arxiv.org/abs/2101.02523.

[32] Dimitre Oliveira. *Creating TFRecords*. Visited on 18/08/2022. URL: https://keras.io/examples/keras_recipes/creating_tfrecords.

[33] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[34] Archit Parnami and Minwoo Lee. *Learning from Few Examples: A Summary of Approaches to Few-Shot Learning*. 2022. DOI: 10.48550/ARXIV.2203.04291. URL: https://arxiv.org/abs/2203.04291.

[35] Huy Phan, Martin Krawczyk-Becker, Timo Gerkmann, and Alfred Mertins. *DNN and CNN with Weighted and Multi-task Loss Functions for Audio Event Detection*. 2017. DOI: 10.48550/ARXIV.1708.03211. URL: https://arxiv.org/abs/1708.03211.

[36] Viraj Prabhu, Anitha Kannan, Murali Ravuri, Manish Chablani, David Sontag, and Xavier Amatriain. *Prototypical Clustering Networks for Dermatological Disease Diagnosis*. 2018. DOI: 10.48550/ARXIV.1811.03066. URL: https://arxiv.org/abs/1811.03066.

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

[38] Nathalie Remy, Eveline Speelman, and Steven Swartz. *Style that's sustainable: A new fast-fashion formula*. Visited on 24/07/2022. URL: https://www.mckinsey.com/business-functions/sustainability/our-insights/style-thats-sustainable-a-new-fast-fashion-formula.

[39] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. *One-shot Learning with Memory-Augmented Neural Networks*. 2016. DOI: 10.48550/ARXIV.1605.06065. URL: https://arxiv.org/abs/1605.06065.

[40] Jurgen Schmidhuber. "Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook". Diploma Thesis. Technische Universitat Munchen, Germany, 14 May 1987. URL: http://www.idsia.ch/~juergen/diploma.html.

[41] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. "Hidden Technical Debt in Machine Learning Systems". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'15. Montreal, Canada: MIT Press, 2015.

[42] Jake Snell, Kevin Swersky, and Richard S. Zemel. *Prototypical Networks for Few-shot Learning*. 2017. DOI: 10.48550/ARXIV.1703.05175. URL: https://arxiv.org/abs/1703.05175.

[43] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. *Meta-Transfer Learning for Few-Shot Learning*. 2018. DOI: 10.48550/ARXIV.1812.02391. URL: https://arxiv.org/abs/1812.02391.

[44] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales. "Learning to Compare: Relation Network for Few-Shot Learning". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2018).

[45] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas. *MLOps – Definitions, Tools and Challenges*. 2022. DOI: 10.48550/ARXIV.2201.00162. URL: https://arxiv.org/abs/2201.00162.

[46] Sebastian Thrun and Lorien Pratt, eds. *Learning to Learn*. USA: Kluwer Academic Publishers, 1998. ISBN: 0792380479.

[47] Dustin Tran et al. *Plex: Towards Reliability using Pretrained Large Model Extensions*. 2022. DOI: 10.48550/ARXIV.2207.07411. URL: https://arxiv.org/abs/2207.07411.

[48] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. *Matching Networks for One Shot Learning*. 2016. DOI: 10.48550/ARXIV.1606.04080. URL: https://arxiv.org/abs/1606.04080.

[49] Wohlgemuth Viola, Kopp Mirjam, and Cobbing Madeleine. *Self-regulation: a fashion fairytale / Part One*. Visited on 20/07/2022. URL: https://www.greenpeace.de/sites/default/files/publications/20211122-greenpeace-detox-fashion-fairytale-engl-pt1.pdf.

[50] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. "Learning to Model the Tail". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/147ebe637038ca50a1265abac8dea181-Paper.pdf.

[51] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. "Meta-Learning to Detect Rare Objects". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9924–9933. DOI: 10.1109/ICCV.2019.01002.

[52] Shuai Zheng, Fan Yang, M. Hadi Kiapour, and Robinson Piramuthu. *ModaNet: A Large-Scale Street Fashion Dataset with Polygon Annotations*. 2018. DOI: 10.48550/ARXIV.1807.01394. URL: https://arxiv.org/abs/1807.01394.

A Appendix

Category name	Super category	Split
Dress Romper Jumpsuit Kimono Coat	Full-body	Meta-train
Shorts Skirt Jeans Leggings Joggers Sweatpants Cutoffs Sweatshorts Jeggings	Lower-body	
Tee Blouse Tank Cardigan Sweater Jacket Top Blazer Hoodie Poncho Jersey Henley Parka	Upper-body	
Onesie Robe Cape	Full-body	Meta-val
Trunks Chinos Sarong	Lower-body	
Bomber Flannel Anorak Turtleneck	Upper-body	
Coverup Caftan	Full-body	Meta-test
Gauchos Capris Jodhpurs Culottes	Lower-body	
Halter Peacoat Button-Down	Upper-body	

Table 3: **Category Splits.** List of category names and super categories on DeepFashion dataset. The split column indicates if the garment class was used for model training, validation, or testing.

			Meta-training parameters			Meta-testing parameters			
lr_drop_rate	**epochs_drop**	**image_size**	**N-way**	**K-shot**	**Query-set**	**N-way**	**K-shot**	**Query-set**	**Accuracy**
0,5	10	84	5	5	15	5	5	15	63,20%
0,5	10	84	6	5	15	5	5	15	64,23%
0,5	10	84	7	5	15	5	5	15	63,51%
0,5	**10**	**84**	**8**	**5**	**15**	**5**	**5**	**15**	**65,20%**
0,5	10	84	8	1	15	5	5	15	49,59%
0,5	10	84	8	2	15	5	5	15	56,68%
0,5	10	84	8	3	15	5	5	15	63,54%
0,5	10	84	8	4	15	5	5	15	60,90%
0,5	10	84	8	4	15	5	5	15	62,76%
0,5	10	84	8	6	15	5	5	15	63,71%
0,5	10	84	8	7	15	5	5	15	62,78%
0,5	10	84	8	8	15	5	5	15	63,90%
0,5	10	84	8	5	1	5	5	15	54,24%
0,5	10	84	8	5	5	5	5	15	62,81%
0,5	10	84	8	5	10	5	5	15	62,18%
0,5	10	84	8	5	15	6	5	15	59,34%
0,5	10	84	8	5	15	7	5	15	55,23%
0,5	10	84	8	5	15	8	5	15	53,63%
0,5	10	128	8	5	15	5	5	15	64,21%
0,5	10	64	8	5	15	5	5	15	64,27%
0,5	10	256	8	5	15	5	5	15	63,56%
0,1	10	84	8	5	15	5	5	15	64,86%
0,2	10	84	8	5	15	5	5	15	64,07%
0,3	10	84	8	5	15	5	5	15	64,04%
0,4	10	84	8	5	15	5	5	15	65,19%
0,6	10	84	8	5	15	5	5	15	63,61%
0,7	10	84	8	5	15	5	5	15	64,11%
0,8	10	84	8	5	15	5	5	15	62,66%

Table 4: **Classification Accuracy of the Model for Different Hyperparameters Settings**. Configurations of training and testing episodes are indicated as meta-training and meta-testing parameters. "N-way" refers to the number of classes per episode, "K-shot" is the number of support points per class, and "Query-set" is the number of query points per class. "lr_drop_rate" and "epochs_drop" correspond to the factor and frequency the learning rate drops during training. The size of the images in pixels is "image_size". Classification accuracy was averaged over 1,000 randomly generated episodes from the meta-test set. The best result is shown in bold.