Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Data Mining Research Group

Aspect-based Few-Shot Learning

Master Thesis

Tim van Engeland

Supervisors:
dr. V. Menkovski (TU/e)

Assessment Committee:
dr. A. Jalba (TU/e)
dr. L. Yin (TU/e)

Eindhoven, Friday 23rd February, 2024

Abstract

Image recognition with Few-Shot Learning commonly comes down to supervised learning of associating images with a label. Few-Shot Learning consists of a query image and a support set of images in which the support set only contains a small number of labels with a few examples each. The task is to map the query image to one of the support set images and assign the correct label. There is one true mapping from an image to a label with this approach. However, an approach based on one true mapping fails without an exact match within the data. Human understanding of the images is not limited by relying on a supervision signal. We do not have any trouble matching a query of a brown dog and a support set image of a brown horse. This matching with the brown horse depends on the support set surrounding the horse. We are forced to look to other features presented within the images if the support set does not contain another image of a dog to match the query of the brown dog. We ignore the intra-conceptual relationships of the support set in Few-Shot Learning with the current system of labels. Therefore, this work proposes to extend the current Few-Shot Learning with the notion of aspects and commonalities (aspect-based Few-Shot Learning). Aspects are the defining or differentiating features among the support set images. The shared features among the support set are defined as commonalities. We propose a framework that extracts the aspects and commonalities described by the support set without pre-defined labels. We create an embedding space that favors matching based on the differentiating features within the support set, while the shared features do not influence decision-making. The goal is to identify the correct assignment between the query and the instances of the support set based on the identified aspects rather than a set of labels.

To address this problem, we introduce a method of training aspect-based Few-Shot Learning. Matching between query and support set instances depends on the support set. Therefore, we ensure that the images are presented in different support sets during training to illustrate the different perceptions of the same image. We also introduce two options for a new data split for aspect-based Few-Shot learning. The standard data splits over images no longer apply because of the removal of labels. The first option is the data unique split, in which we assume that every image is unique. Training data consists of an entirely separate set of images compared to the test images. The second option is the query split. In this split, we only separate the set of images for the query images. The support sets during training consist of the same images as during testing. We introduce the Deep Set Traversal Module as a permutation-invariant machine learning model for extracting commonalities and aspects. This module forms the aspect-based representation model combined with the representation model to extract features from the initial images and deep metric learning. We evaluate our method on a generated geometric shape and a 2D sprites dataset. We demonstrate through experiments the feasibility of the approach by assessing its performance against a baseline that does not account for aspects and commonalities of the support set in the embedding space.

Contents

Contents **v**

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Research objective . . . 5
1.3 Contributions . . . 5
1.4 Research outline . . . 6

2 Research background **7**
2.1 Convolutional neural network . . . 7
2.1.1 Convolutional layers . . . 8
2.1.2 Pooling layers . . . 8
2.1.3 Non-linear activation function . . . 10
2.1.4 Fully-connected layers . . . 10
2.1.5 Advanced architectures . . . 11
2.2 Few-shot learning . . . 11
2.3 Metric learning . . . 12
2.4 Deep sets . . . 13

3 Literature review **15**
3.1 Deep metric learning . . . 15
3.2 Categorical traversal module . . . 18

4 Problem Description **21**
4.1 The effect of context on the perception of distance . . . 21
4.2 Aspect-based embedding space . . . 22
4.3 Scope of the problem . . . 23
4.4 Problem formulation . . . 23

5 Solution approach **25**
5.1 Data generation . . . 25
5.1.1 Data description . . . 25
5.1.2 Feature design . . . 27
5.1.3 Generation of support set . . . 27
5.1.4 Varying shared features with the query . . . 29
5.1.5 Data sequence . . . 30
5.1.6 Definining test data . . . 31
5.2 Deep Set Traversal Module . . . 32

5.3 Deep Metric Learning . . . 34
5.3.1 Loss function . . . 34
5.3.2 Aspect-based metric network . . . 35

6 Experiments **37**
6.1 Dataset . . . 37
6.2 Model architecture . . . 38
6.2.1 Representation learning . . . 38
6.2.2 Deep Set Traversal Module . . . 39
6.3 Experiment description . . . 39
6.4 Performance evaluation . . . 39
6.5 Environmental setup . . . 40

7 Results **41**
7.1 Experimental results and analysis of the distance ratio . . . 41
7.1.1 Geometric shapes data . . . 41
7.1.2 Sprites data . . . 44
7.2 Experimental results and analysis of the aspects . . . 47
7.2.1 Geometric shapes data . . . 47
7.2.2 Sprites data . . . 48
7.3 Discussion . . . 48

8 Conclusion and recommendations **51**

Bibliography **53**

List of Figures

1.1 Demonstration of a task in which a person has to compare a handwritten character at the top to a set of 20 different handwritten characters that are unknown to the person. . . . 1
1.2 A four-way-one-shot classification task: Classification based on the predefined label will assign correct label “horse” to the query image. The answer would be the same if there is a classification based on the differentiating animal species aspect present in the support set. . . . 3
1.3 A four-way-one-shot classification task: Each image in the support set has the label 'dog,' while the query image is a horse. The label-based classification of 'dog' is, therefore, not a correct assignment to the query image. However, a classification based on the aspect of physical activity would lead to the second image of the support set. They share the common movement of running. . . . 3
1.4 Each image in the support set has the label 'dog,' while the query image is a horse. The label-based classification of 'dog' is, therefore, not a correct assignment to the query image. However, a classification based on the aspect of surface would lead to the fourth image of the support set. They share the common surface of a grass field with flowers. . . . 3

2.1 Comparison between convolutional layers with weights sharing of CNNs and fully-connected neural network. The number of trainable parameters with the implementation of weights sharing is just 0.01% of the number of trainable parameters without weights sharing. Source from www.deeplearning.net . . . 7
2.2 The architecture of the original CNN [18] . . . 8
2.3 An example of 2-D convolution. The arrows indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor. [8] . . . 9
2.4 Max-pooling . . . 9
2.5 Curves of the Sigmoid, Tanh, and ReLu activation functions[13] . . . 10
2.6 Architecture of residual block with a skip connection [9] . . . 11
2.7 Example of 2-way-4-shot classification problem [22]. On the right, the support set consists of two classes with four examples per class. The query set on the left will be classified as either cat or dog by using the support set. . . . 12

3.1 The categorization of FSL by Pernami et al. [22] . . . 16
3.2 Taxonomy of few-shot deep metric learning methods [20] . . . 17
3.3 Detailed breakdown of components in CTM [19] . . . 18

4.1 (a) A Necker cube: a two-dimensional line drawing of a cube. (b) Additional information can resolve the perceptual ambiguity [14] . . . 21
4.2 Three images that can be perceived as similar based on a different aspect: color, thickness, and pattern . . . 22

4.3 2D aspect-based embedding space. The relative distances between the embeddings are different per aspect. 23

5.1 The effect of each parameter on the features of the geometric shapes data set . . . 26
5.2 The effect of each parameter on the features of the sprites data set 28
5.3 An example of a generated support set for each data set 29
5.4 The number of shared features dictates the aspect-based embedding space. The same image is perceived differently in another support set. 29
5.5 The generation of four different support set for the same query 30
5.6 The same support set for different queries. The features of the training query in (a) differs completely from the testing query in (b). The matching between query and support set images differs accordingly. 31
5.7 Detailed breakdown of components in DSTM 32
5.8 The permutation-equivariant deep set model 33
5.9 The permutation-invariant deep set model 34
5.10 Demonstration of the tuplet loss on the aspect of the shirt: the positive example gets closer to the anchor, while every negative example moves away. 35
5.11 The aspect-based metric network 36

6.1 Feature values seen during training do not matter as the complete image does not match. (b) has a not-seen green shirt and pants during training. So, the green hair of the training is no problem for the support set. However, the support set of (c) contains three images from training. Therefore, it does not comply with the separation. 38
6.2 The layers of each representation model 39

7.1 Example of the distance between the query and support set instances for the geometric shape data. Differentiating aspects leads to different distances between query and the same image. 41
7.2 Example of the distance between the query and support set instances for the sprites data. Differentiating aspects leads to different distances between query and the same image. 44
7.3 A support set with the aspect of thickness. 47
7.4 A support set with the aspect of stance. 48

List of Tables

3.1 Meta-Learning Approaches 15

7.1 Geometric shapes data set: The average positive and negative distance of the first support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval. 42
7.2 Geometric shapes data set: The average positive and negative distance of the second support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval. 43
7.3 Geometric shapes data set: The average positive and negative distance of the third support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval. 43
7.4 Sprites data set: The average positive and negative distance of the first support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval. 45
7.5 Sprites data set: The average positive and negative distance of the second support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval. 45
7.6 Sprites data set: The average positive and negative distance of the third support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval. 45
7.7 Sprites data set: The average positive and negative distance of the fourth support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval. 46
7.8 Geometric shapes data set: The average positive and negative distance of the first support set with a 95% interval for each aspect. 47
7.9 Sprites data set: The average positive and negative distance of the first support set with a 95% interval for each aspect. 48

Chapter 1

Introduction

1.1 Motivation

If I had to distinguish between a square and a circle, I would have no problem. The same would apply to most humans as we have been exposed to dozens of circles and squares in our lives, which give us the knowledge to perform this task. However, most of us cannot do the same between a Spix's macaw and a Madagascar pochard. Except for bird experts, we probably have never seen or been in contact with these birds. So, our perceptual system does not know how to distinguish them. In contrary to bird experts, which come in contact with different kinds of birds. Repeated exposure to certain stimuli causes perceptual learning: 'Perceptual learning involves relatively long-lasting changes to an organism's perceptual system that improve its ability to respond to its environment [7].' Our perception allows us to distinguish between features and categorize objects. To illustrate, Figure 1.1 has 20 distinct handwritten characters that are probably unknown to most. I could only categorize them as handwritten characters as they are unknown to me, while a person with knowledge of languages could already have some subcategories. Nevertheless, we can still express them in established terms and distinguish between the characters. We conclude that the character on top is the same as the most left character on the second row as both display a spiral movement with 90-degree corners.

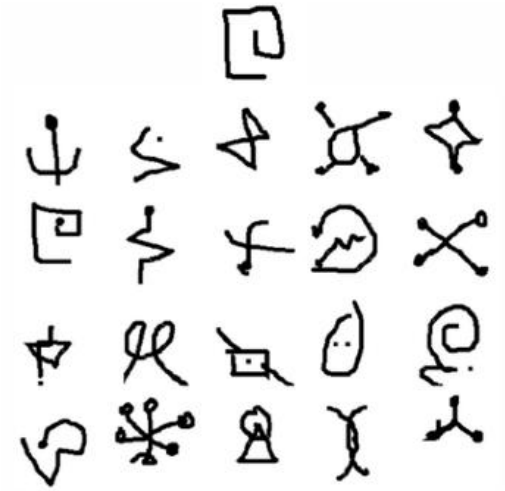


Figure 1.1: Demonstration of a task in which a person has to compare a handwritten character at the top to a set of 20 different handwritten characters that are unknown to the person.

In some cases, the goal of machine learning is to try to capture this human behavior within a model. Image classification is a well-known problem that has been studied throughout the years [9]. In this task, we assign a label to a certain picture (supervised learning), and the model's objective is to assign the correct label. Throughout the years, we have seen several advancements regarding model structure [5, 18], but also adaptations to the state-of-the-art at the time [9, 27]. However, the supervised task of image classification has its drawbacks. The first one is the data-hungry nature: it takes many iterative updates to the model to perform the required task at hand. For example, an image classification model would have to see multiple images of the character on top in Figure 1.1 before perceiving the most left character on the second row as the same. In contrast to the many images of handwritten characters that are readily available [17], many machine learning problems lack available data (data scarcity). Privacy or the specificity of the problem can limit the availability of data. Furthermore, data collection, pre-processing, and annotation are strenuous human tasks [22] regardless of data availability. The second problem is that the label approach works on a predefined set of labels. The model trained in a supervised setting will only learn to recognize the labels that occured during training. Both these problems go against the human nature of learning. We can identify objects by only seeing them once before. Furthermore, the example of Figure 1.1 demonstrated that we can recognize the handwritten character when we have never encountered it before.

The idea of Few-shot learning (FSL) was introduced to counter these problems. FSL refers to the ability of machine learning models to generalize from a few training examples [22]. The objective of the FSL model is to learn the transferable knowledge from the 'base' data (training data) with a small amount of labeled data and to be able to apply it to a 'novel' dataset with a small amount of never-seen labels. Furthermore, FSL deals with a small amount of labels per class. The model is presented with two parts to gain the knowledge: a query instance and a support set. The model has to classify the query instance based on the examples of the support set. By repeating the task on the 'base' data, it will eventually learn how to apply the same principles on the same task but with a different dataset. So, if trained accordingly, the model would be able to perform the task in Figure 1.1 without ever seeing one of those handwritten characters.

Few-shot learning has seen multiple developments over the years with great results to show for [22]. However, every development utilizes labels as a supervision signal. There exists one true mapping from an instance to a label. It fails when there is no exact match for the instance. We will demonstrate this by looking at three examples of four-way-one-shot learning. Figure 1.2 shows a standard example of the label approach. There is one image of four distinct classes: horse, dog, bunny, and lion. In this example, we would recognize that the query image is in line with the characteristics of the first image and should, therefore, be labeled a horse. However, when looking at the second set of images in Figure 1.3, there are only images of dogs. Despite this, we can still say that the second image is the most similar to the query image. The label dog is not the only thing we notice within the image. The first image is a sitting dog, the second shows running, the third one lays down, and in the last image, the dog is chewing on a bone. So, we understand that the physical activity of the dogs is different for each image. Based on that differentiating factor, we could determine that the running in the second image is also displayed within the query image. The third example in Figure 1.4 demonstrates that we can extend the notion further. Each image in the support set displays a running dog. However, each of them is running on a different surface. In order from left to right, the dogs are running in the water, on snow, on sand, and in a field. Based on my perception, I would connect the query image to the fourth dog as they are both running on grass. Each support set conveys different information, which affects your perception of the images within the support set. Every example has the same second image, but we perceive it differently. The first support set emphasizes the type of animal. Therefore, we perceive the second image as a dog. The second indicates physical activity with the resulting perception of running. Lastly, we distinguish between surfaces and see the second image as snow. The examples show that support set has a defining aspect that dictates the individual comparison between instances

of the support set and the query image.

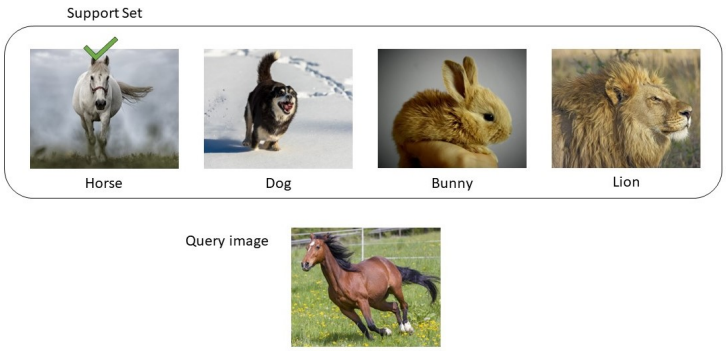


Figure 1.2: A four-way-one-shot classification task: Classification based on the predefined label will assign correct label “horse” to the query image. The answer would be the same if there is a classification based on the differentiating animal species aspect present in the support set.

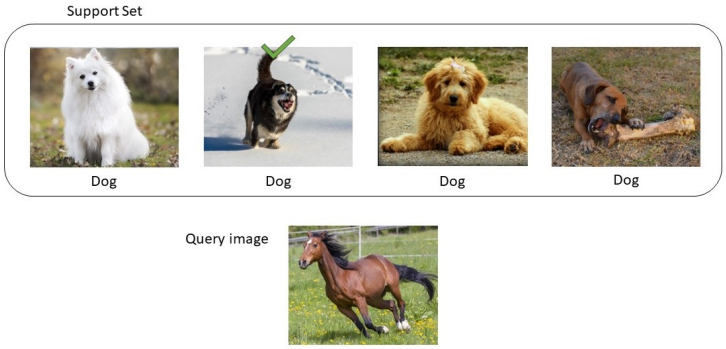


Figure 1.3: A four-way-one-shot classification task: Each image in the support set has the label ’dog,’ while the query image is a horse. The label-based classification of ’dog’ is, therefore, not a correct assignment to the query image. However, a classification based on the aspect of physical activity would lead to the second image of the support set. They share the common movement of running.

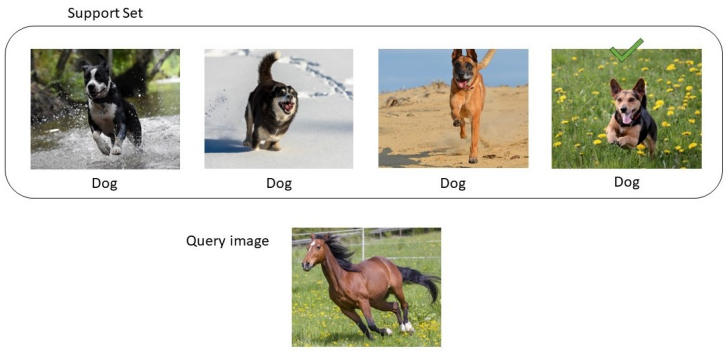


Figure 1.4: Each image in the support set has the label ’dog,’ while the query image is a horse. The label-based classification of ’dog’ is, therefore, not a correct assignment to the query image. However, a classification based on the aspect of surface would lead to the fourth image of the support set. They share the common surface of a grass field with flowers.

The current state-of-the-art FSL methods ignore those relationships between the instances of the support set by limiting the information to a label. Often, the focus lies on multiple one-on-one comparisons between the query and instances of the support set. Due to these limitations, we create a model only capable of finding the exact match, which is not in line with human understanding of the content of the data. In the current FSL setting, Figure 1.3 would only work with the exact match if we only highlight the physical activity represented in the images. Rather than the class 'dog,' we would assign 'sit,' 'run,' etc. However, it is not efficient to do so and requires unnecessary amounts of annotation. So, to expand the possibilities within FSL, we have to move away from relying on labels as a supervision signal. Rather, the new methods already have to account for the intra-conceptual relationships of the support set.

This work proposes to extend the FSL model to an aspect-based approach. Rather than learning the pre-defined labels, the model learns the differentiating feature of the support set. In our work, we will denote the aspect as the varying feature in the support set. The shared features among instances of the support set are defined as commonalities. With these definitions in place, we propose a framework that learns the aspect described by the support set without pre-defined labels. The developed framework's goal is to identify the correct assignment between the query and the instances of the support set by determining the aspect.

1.2 Research objective

The current direction of FSL is about the correct association of a query object to an instance of a support set. In this work, we address the case of identifying the changing feature (aspect) within the support set to associate between the query and the support set instead of relying on a fixed mapping of predefined labels. Similar to FSL, our research will also handle learning from a limited amount of available examples to illustrate human behavior. We refer to this problem as aspect-based Few-Shot learning. The main objective of this work is to solve the problem of aspect-based Few-Shot learning. We need to extend the current state-of-the-art FSL methods to include the intra-conceptual relationships of the support set. Therefore, our problem formulation is the following:

Develop a method capable of extracting the commonalities and aspects from a support set to accurately associate a query object to a support set instance given the aspect.

To find a solution to this problem, we need to solve three research questions. The first one regards the capability of extracting commonalities and aspects from the support set. The extracted aspects and commonalities are not dependable on the order of the instances in the support set. If we shuffled the support set, we would still perceive the same support set. We define this as permutation invariance within the model. So, our first research question is the following:

1. *How to build a permutation invariant machine learning model for extracting aspects and finding commonalities within a support set?*

The second and third questions are about the training of the model. The training- and test split of the data in the FSL setting are straightforward. In training, we have a set of classes that will not appear in the test data ($C_{training} \cap C_{test} = \emptyset$). Within this work, there are no longer any classes available. The standard FSL split does not apply any longer to this problem. Also, there is the standard image classification split: images in the training set will not appear in the testing set. However, an image could represent a different aspect in another context. So, we could argue to include images from training in testing. It becomes clear that the standard splitting procedures do not completely align with the task of aspect-based FSL. Therefore, we construct a new way of splitting the data according to the task of aspect-based FSL.

Furthermore, we must ensure that during training, the model learns to apply the extracted commonalities and aspects of the support set. Figure 1.2, 1.3, and 1.4 show how the same dog can be perceived differently. The training data has to reflect the different perception of the same image to ensure properly capturing the commonalities and aspects. Additionally, we provide another option besides the triplet network setup of training. In the triplet setup, the model is presented with the query, a positive-, and a negative sample. Our training is not limited to one negative example by including the intra-conceptual relationships of the support set. So, the second and third research questions are the following:

2. *How to separate the training data from the test data for the task of aspect-based Few-Shot learning?*
3. *How to develop a procedure of data points for training the model to capture and utilize the commonalities and aspects presented by the support set?*

1.3 Contributions

We highlight that the listed contributions in this section currently hold when the data generation process is known. Currently, there are no realistic data sets that adhere to the task of aspect-based FSL. We realize the difficulty in constructing such a data set due to the subjective nature of aspect-based FSL. For example, this work highlighted the aspect of physical movement in Figure

1.3. Physical movement was one of the aspects that first occurred when starting this work. Therefore, physical movement was perceived as more important. However, there is also the aspect of the color of dogs' coats. Depending on the person's perception, the answer could differ. Real-life data for aspect-based FSL suffers from ambiguity as it is impossible to control the individual features. Therefore, real-life data requires extensive human feedback.

This work contributes to the literature by providing insights in the following ways:

- We propose an alternate method for training the novel problem of aspect-based few-shot learning. We provide an extension to the training of FSL that includes n number of negative samples rather than one. We also offer the first setup for splitting the data.
- We analyze the limitations of standard few-shot metric learning in a setting where the aspects of the entire support set determine the correct assignment and show the need for a solution that moves away from the prevalent label matching between instances of the support set and query.
- We propose a solution method and evaluate our method based on the performance against other few-shot learning methods in the setting of aspect-based few-shot learning.

1.4 Research outline

This section outlines an overview of the following chapters. Chapter 2 introduces the relevant background information for understanding this work's solution approach with the necessary notations and definitions. Chapter 3 provides an overview of the relevant literature in the current field of few-shot learning. Chapter 4 discusses the effect of context on perception and the translation of this effect to an aspect-based embedding space. We formally define the problem within this chapter with the necessary notation. Chapter 5 explains our solution approach for data generation, deep metric learning for aspect-based FSL, and an aspect-based representation model. Chapter 6 provides the experimental setup with the necessary evaluation metric. Chapter 7 presents the outcome of the results, followed by a discussion. Chapter 8 summarizes and concludes the work with recommendations for future work.

Chapter 2

Research background

We provide the necessary background information in combination with mathematical notations and definitions in this chapter. First, convolutional neural networks are explained. Second, we introduce Few-shot learning formally with the needed mathematical notation. The third section introduces the definition of metric functions and metric learning. Finally, we conclude this chapter with the introduction of Deep Sets.

2.1 Convolutional neural network

Convolutional neural networks (CNNs) were first introduced in 1998[18]. However, it took a while before it started to gain momentum within the machine learning community due to the success of AlexNet [16]. The model design works well for the class of spatially distributed data. Within spatially distributed data, we have correlations of the features based on their spatial position relative to other features. For example, images have correlations localized in small regions. Pixels that are positioned close to each other are highly correlated. The structure of the CNN exploits the assumptions or preconceptions (inductive bias) about the spatially distributed data by utilizing local receptive fields (convolutional layers). The same local receptive field is reused over the complete image (weights sharing). The re-usage of the parameters drastically reduces the number of parameters, which helps during training with high-dimensional data (see figure 2.1). The parameter reduction has two main advantages: lowers training times and prevents over-fitting [3]. Over-fitting is a modeling error that occurs when the model corresponds too closely to the training data and may fail to perform the same task on unseen test data.

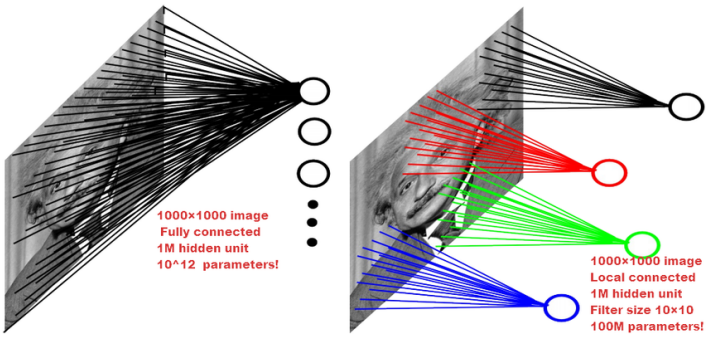


Figure 2.1: Comparison between convolutional layers with weights sharing of CNNs and fully-connected neural network. The number of trainable parameters with the implementation of weights sharing is just 0.01% of the number of trainable parameters without weights sharing. Source from www.deeplearning.net

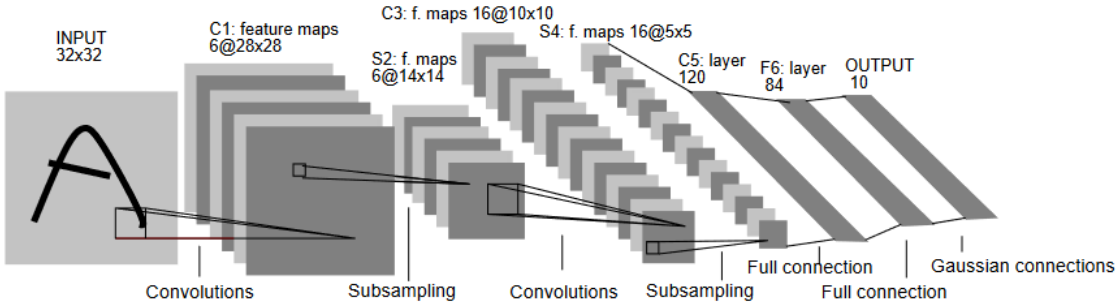


Figure 2.2: The architecture of the original CNN [18]

Feature extraction is done by two types of layers in a CNN: convolutional and pooling layers. Often, the model structure stacks multiple convolutional and pooling layers. The stacking helps the model to recognize high-level features within the spatially distributed data. The first layers learn low-level features, such as edges and lines. In the following layers, the low-level features combine to produce simple shapes (circles, squares, etc.). Each additional layer provides more specific information. Each convolutional layer also utilizes a non-linear activation function. The non-linear activation functions allow the CNN to model complex and non-linear relationships between inputs and outputs. Traditionally, we also have fully-connected layers after the convolutional and pooling layers to process the extracted features to the final output. Figure 2.2 shows the different layers in the first iteration of the CNN.

2.1.1 Convolutional layers

The convolutional layer performs a dot product between two matrices or vectors. However, to illustrate, we will use images as our spatially distributed data in this section. One matrix is the kernel consisting of trainable parameters, while the other is a restricted part of the input with the same size as the kernel. The kernel is spatially smaller than the input (width and height) but extends over the complete depth. The kernel slides across the input during the forward pass, producing a single value representation of each region (see Figure 2.3). The following formula determines the output dimension after applying the kernel:

$$d_{out} = \lfloor \frac{d_{in} + 2p - k}{s} \rfloor + 1 \tag{2.1}$$

There are several components to the formula. First, d_{in} is the number of input features. The p stands for padding, which is the process of adding layers of zeros or other values on the outside of the input matrix. It is a technique to manage the spatial dimensions of the input. The k represents the kernel size (local receptive field). The last parameter s means the stride, which dictates the movement of the kernel across the input. In other words, the stride determines how many pixels we shift the filter after each step.

To illustrate each component, we look at the example of Figure 2.3. The image has an input width of four and a height of three. No padding is added to the outside of the image, and the kernel size is two by two. Last, the stride is set to one, so the filter will only move one pixel at a time. By applying Equation 2.1, we get an output width $d_{out} = \lfloor \frac{4+2\times0-2}{1} \rfloor + 1 = 2 + 1 = 3$ and an output height $d_{out} = \lfloor \frac{3+2\times0-2}{1} \rfloor + 1 = 1 + 1 = 2$, which is in accordance with Figure 2.3.

2.1.2 Pooling layers

The feature maps after a convolutional layer capture the exact position of the local patterns. This exact position opposes a problem as the model is susceptible now to small shifts in the

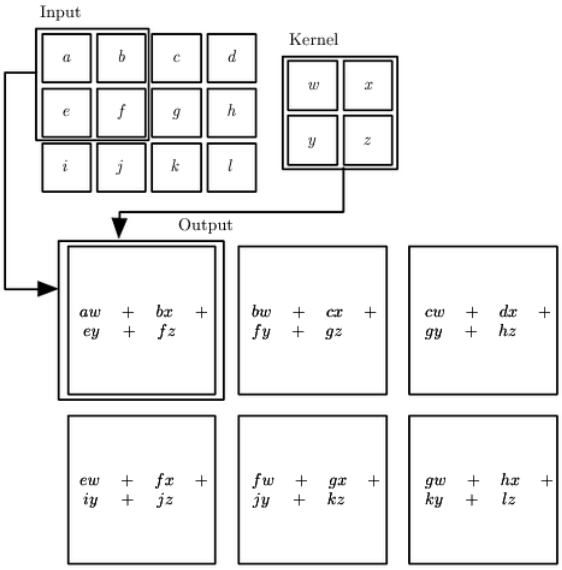


Figure 2.3: An example of 2-D convolution. The arrows indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor. [8]

input. The solution within the CNN structure is the addition of a local or global pooling layer after the convolutional layers. A pooling layer replaces the networks' output at certain locations by deriving a summary statistic of the nearby outputs. It reduces the spatial size of the representation to introduce translation invariance to tiny shifts. Figure 2.4 shows an example of max-pooling. Pooling layers also utilize a kernel, as seen within the convolutional layers. However, the output is the max value seen within the kernel instead of the dot product of the convolutional layers. Typically, pooling layers use a 2x2 kernel with a stride set to two. Another well-known pooling method is Average pooling. Rather than taking the max value, we take the average over the values seen within the kernel.

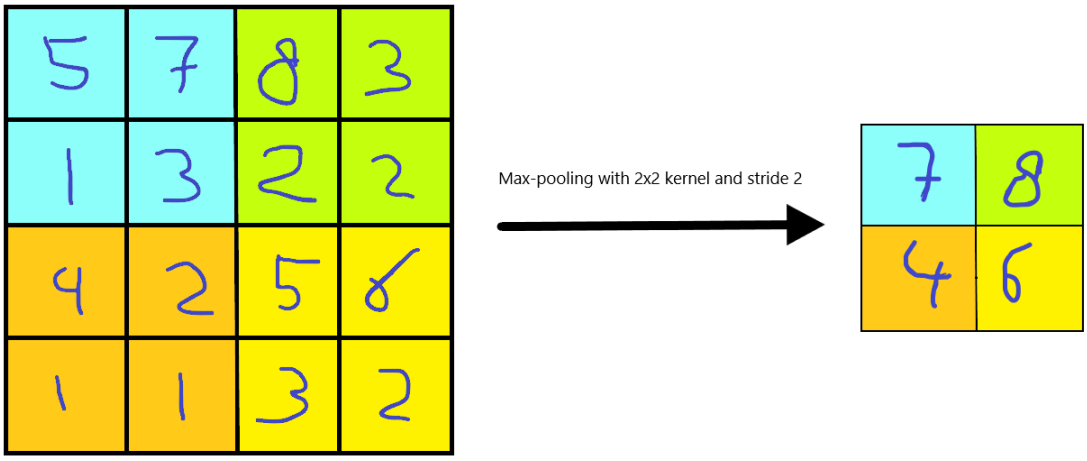


Figure 2.4: Max-pooling

2.1.3 Non-linear activation function

A non-linear activation function is required to convert the linear inputs to non-linear outputs. The dot product with kernels in CNNs are linear in nature and are, therefore, only able to model a linear relationship between inputs and outputs. However, adding non-linear activation functions allows the CNN to model non-linear relationships between inputs and outputs. Furthermore, the non-linear activation functions allow the stacking of multiple layers. If we would stick with linear functions, the last layer will still be a linear function of the first layer regardless of the number of layers in the network.

The three most common non-linear activation functions are rectified linear unit (ReLU), sigmoid, and hyperbolic tangent (tanh) [26]:

$$\text{ReLU: } f(x) = \max(0, x), \text{ sigmoid: } f(x) = \frac{1}{1 + e^{-x}}, \text{ tanh: } f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

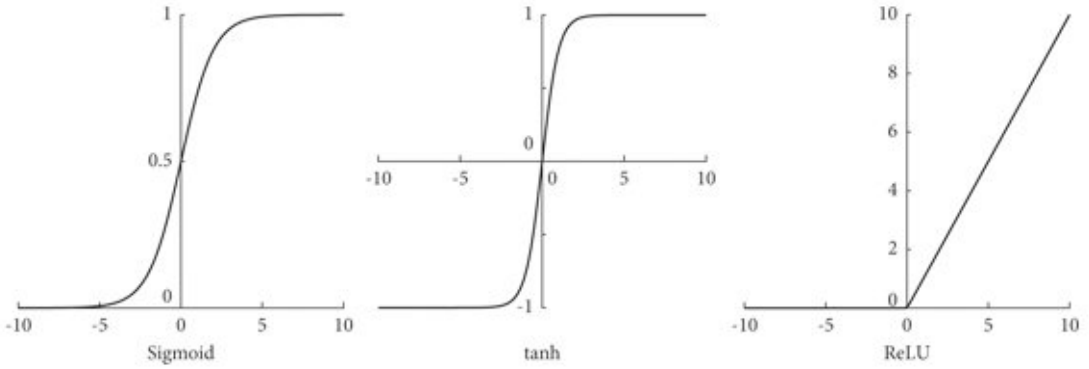
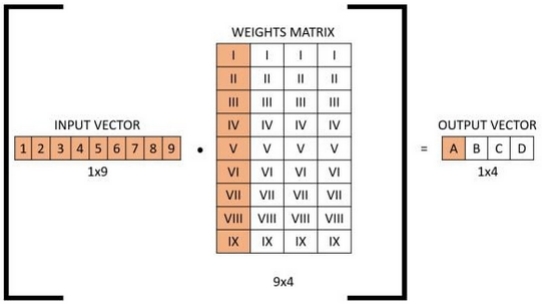


Figure 2.5: Curves of the Sigmoid, Tanh, and ReLu activation functions[13]

2.1.4 Fully-connected layers

Fully-connected layers or linear layers connect every input neuron to every output neuron. Each connection has its own trainable weight. The fully-connected layer performs the dot product between the weight matrix W and the input x. Typically, with the addition of a bias term W_0[1]:

$$y_{jk}(x) = \sum_{i=1}^{N} w_{jk} x_i + w_{j0}$$



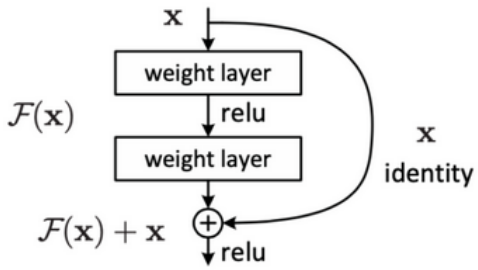


Figure 2.6: Architecture of residual block with a skip connection [9]

2.1.5 Advanced architectures

The VGG structure utilizes the smallest possible receptive field of 3 by 3 pixels. The smallest possible receptive field refers to the size that allows capturing information on each side of the middle pixel(up, down, left, and right). The convolutional layer uses a stride and padding of 1 to preserve the spatial resolution. After each convolutional layer, there is a ReLU component to reduce training time. The combined convolutional layers with ReLU activation are repeated several times with the same number of filters for the convolutions. A pooling layer follows the stacking of convolutions with ReLU to reduce the dimensionality and the number of parameters of the feature maps. This process of stacking convolutions with a final pooling layer is repeated several within the VGG structure.

ResNet architecture was an introduction to CNNs with an increased amount of layers. The VGG method of stacking convolutional layers with ReLU seems to have a maximum of around 19 layers. The opted solution of the ResNet architecture was to add an intermediate input to the output of a series of convolution blocks (residual blocks). The intuition behind the residual blocks is that the model learns the difference between the input and output. Consider H(x) to be the mapping of x after multiple convolutional layers. Rather than expecting stacked layers (such as VGG) to approximate H(x), the residual blocks only approximate the residual function F(x) = H(x) - x. The original mapping is recast into F(x) + x and can be realized by feed-forward neural networks with skip connections (Figure 2.6). The ResNet architecture can consist of many more layers than the VGG architecture by only modeling the residual function.

2.2 Few-shot learning

Humans translate previously learned information across tasks to new information. It is called meta-learning within machine learning, or learning to learn, and aims to learn new concepts or adapt to new environments efficiently from previous experiences. A well-known application of meta-learning is few-shot learning (FSL): the ability of machine learning models to generalize from a few training examples. The objective of the FSL model is to learn the transferable knowledge from the 'base' data (training data) with a small amount of labeled data and to be able to apply it to a 'novel' dataset with a small amount of never-seen labels. Formally, we have a few-shot classification learning task T with a data set $D = \{\mathrm{D}_{train}, \mathrm{D}_{test}\}$ with n data samples, where $\mathrm{D}_{train} = \{(\mathrm{x}_i, \mathrm{y}_i)\}_{i=1}^{t}$ and $\mathrm{D}_{test} = \{(\mathrm{x}_i, \mathrm{y}_i)\}_{i=t+1}^{n}$ with t being the number of training samples. Let $p(x, y)$ be the ground-truth joint probability distribution of input x and output y, and $\hat{f}$ be the optimal hypothesis from x to y. FSL tries to approximate $\hat{f}$ by determining $f(x;\theta)$ by optimizing for the parameters θ on the training data D_{train} and evaluating on D_{test}:

$$y = f(x;\theta) \text{ where } (x, y) \in D_{test} \tag{2.2}$$

[1]Figure from https://builtin.com/machine-learning/fully-connected-layer

and

$$\theta = \arg\min_{\theta} \sum_{(x,y)\in D_{train}} L(f(x;\theta), y) \tag{2.3}$$

where L is the loss function between prediction $f(x;\theta)$ and true label y to measure the error of the model.

The few-shot classification task is often referred to as a standard N-way-K-shot task, where N is the number of classes and K denotes the number of samples per class present. Each task of N-way-K-shot classification T_i contains the following components:

- The support set S_i consisting of N class labels and K labeled instances for each class.
- The query set Q_i containing Q query samples.

Figure 2.7 illustrates the different components. In each task, the objective is to correctly assign a class label to each query, where classes are specified by the support set consisting of only a few labeled examples. There are three main approaches to solving this problem. Metric-based methods aim to learn a shared metric (distance function) in feature space for few-shot prediction. Examples of this would be Convolutional Siamese Networks [15], Matching Networks [33], and Relation Networks [31]. Optimization-based methods follow the idea of modifying the gradient-based optimization to adapt to novel tasks, such as Model-Agnostic Meta-Learning [6], and Latent Embedding Optimization [23]. At last, there are memory-based methods [24, 4], which adopt extra memory components for novel concept learning.

Figure 2.7: Example of 2-way-4-shot classification problem [22]. On the right, the support set consists of two classes with four examples per class. The query set on the left will be classified as either cat or dog by using the support set.

2.3 Metric learning

A metric or distance function is a function that defines the distance between elements of a set. A distance function tells us the similarity between two objects. A small distance indicates high similarity, while a large distance means high dissimilarity. There are some properties to which a distance function has to adhere. Therefore, we have to specify the definition of distance [30]:

Definition 1. Let X be a non-empty set. A distance over X is a map d: $X \times X \to \mathbb{R}$ that satisfies the following properties:

1. Non-negativity: $d(x, y) \geq 0$ for every $x, y \in X$
2. Coincidence: $d(x, y) = 0$ if and only if $x = y$ for every $x, y \in X$
3. Symmetry: $d(x, y) = d(y, x)$ for every $x, y \in X$

4. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$ for every $x, y, z \in X$

We call the ordered pair (X, d) a metric space.

Within this work, we will focus on the Euclidean distance, also referred to as the L_2-norm. L_2-norm stems from the generalization of the Euclidean and Manhattan distance called the Minkowski distance or L_p-norm, where p = 2 for the Euclidean distance. It has a unified definition:

$$d(x, y) = \sum_{i=1}^{N} (\|x_i - y_i\|^p)^{\frac{1}{p}} \tag{2.4}$$

where N denotes the dimensionality of the space and p has to be an integer. For p $\geq$ 1, the Minkowski distance is a distance or metric as the specified properties of definition 1 hold.

Metric learning is the task of learning distances from a dataset. Consider a training set D_{train} = $\{(x_i, y_i)\}_{i=1}^{t}$ with t being the number of training samples. Between the examples, we can construct same-class pairs (y is the same) and different-class pairs. To illustrate, x_i and x_j are similar if $y_i = y_j$ and should have a small distance between them. To translate this notion the the idea of FSL, our support set contains two images x_1 and x_2 with their respective classes y_1 and y_2. We have a third image x_3 as the query, which we would like to classify. The computed distance between x_1 and x_3 is smaller than the distance between x_1 and x_2. Thus, we would assign the class of x_1 to x_3, as the smallest distance indicates the highest similarity.

Typically, the distance is computed over the embeddings of the input, rather than the high-dimensional input. The embeddings are obtained using an embedding function f to transform the input to a lower dimension before calculating the distances:

$$f : \mathbb{R}^n \to \mathbb{R}^m, \text{where n} > \text{m} \tag{2.5}$$

Metric-based FSL has two main approaches. The first one is to learn an embedding function $f(;\ \theta)$, which is parameterized by a neural network with parameters θ. The distance over the embeddings is then calculated with a defined distance metric, such as the Euclidean distance, The second option is to add another neural network $g(;\ \lambda)$ after the embedding function, which will function as the distance function.

2.4 Deep sets

In mathematics, a set is defined as a collection of distinct, well-defined objects forming a group. [12] 'Distinct' indicates that each element must be unique, meaning that no element can occur more than once. Each element is considered once, regardless of how many times it may be mentioned. 'Well-defined' means that it is possible to readily determine whether an object is a member/element of a given set. So, either a is an element of set A ($a \in A$) or a is not in set A ($a \notin A$). A significant property of a set is that the order does not matter. So, the set A = 4, 7, 1 and set B = 7, 4, 1 are the same.

In neural networks, the order property translates to the notion of permutation invariance. Often, our inputs can be seen as sets: sequences of items, where the ordering of items does not influence the task at hand. A permutation-invariant function exploits the built-in knowledge of discarding the order of the elements. So, applying the function $f(;\ \theta)$ to the input with the order of x_1, x_2, and x_3 ($f[x_1, x_2, x_3;\ \theta]$) would yield the same outcome as $f(x_3, x_1, x_2;\ \theta)$. A simple example would be when asked the sum of 5 different numbers: 13, 4, 6, 8, 2. The answer would be 33, regardless of the order of the numbers.

An approach to the permutation-invariant neural networks is to utilize an operation P, which is known to be permutation-invariant. We embed each element separately through a neural network $f(;\,\theta)$ and apply the operation P to the set of embeddings. The permutation-invariant property of P destroys the information regarding the ordering. Deep Sets [35] applies this technique by setting P to a summation of the embeddings (sum-decomposition). The architecture of deep sets is decomposed into two: The equivariant and the invariant model. The equivariant model utilizes the permutation-invariant operation P to enrich the embedding of each element with the information of the other members of the set:

$$g_i = g(x_i, \underset{j \in N(i)}{P} [f(x_j; \theta)]\,; \lambda) \tag{2.6}$$

where N is the neighborhood.

In Equation 2.6, we notice that the neighboring elements are first embedded through neural network $f(;\,\theta)$. Afterward, the operation P is applied to the embeddings, which results in a permutation-invariant representation of the neighborhood. The element and its neighborhood representation are then pushed through another neural network $g(;\,\lambda)$. Thus, in the end, we end up with an updated embedding for each element. The permutation-equivariant part indicates that the ordering of the input will be the same as the order of the output. However, the element's position within the set does not change output embedding for an element. The next part is the permutation-invariant model. Now we apply an operation P over the updated embeddings to destroy the information regarding the ordering of the set:

$$\text{output} = h(\underset{i \in S}{P} [g_i]\,; \mu) \tag{2.7}$$

where S denotes the entire set. The sum-decomposition of the set is then processed by another neural network $h(;\,\mu)$ to have a final output embedding that represents the complete set, regardless of the original ordering in which the data is presented.

Chapter 3

Literature review

In this chapter, we first discuss current state-of-the-art deep metric learning techniques that are relevant to our research and provide an overview of the existing solution approaches to problems of a similar nature to our challenge.

3.1 Deep metric learning

There are two main groups within FSL: Non-Meta-Learning and Meta-learning [22]. Non-meta mainly focuses on transfer learning. In the FSL setting, the data is often too limited to train a deep network from scratch. Therefore, we transfer the knowledge of a pre-trained model to a new task instead of training from scratch. However, the main direction is Meta-Learning or Learning-to-Learn. These methods aim to approximate the function f with parameters θ such that the performance on any few-shot task T_i sampled randomly from the task distribution $p(T)$ is optimal. Consider an N-way-K-shot classification task, the meta-learner learns a prior θ over the distribution of N-way-K-shot classification tasks during training. After training, we would be

	Metric-based	**Optimization-based**	**Model-based**
Key idea	Metric Learning	Gradient Descent	Memory; RNN
How is modeled?	$\sum_{(x_j,y_k)\in \mathrm{S}} k_\theta(x,x_k)y_k$	$P_{\theta'}(y\|x)$, where $\theta' = g_\phi(\theta, \mathrm{S})$	$f_\theta(x, \mathrm{S})$
Advantages	Faster Inference.	Offers flexibility to optimize in dynamic environments.	Faster inference with memory models
	Easy to deploy.	S can be discarded post-optimization	Eliminates the need for defining a metric or optimizing at test
Disadvantages	Less adaptive to optimization in dynamic environments.	Optimization at inference is undesirable for real-world deployment.	Less efficient to hold data in memory as S grows
	Computational complexity grows linearly with size of S at test	Prone to overfitting.	Hard to design.

Table 3.1: Meta-Learning Approaches

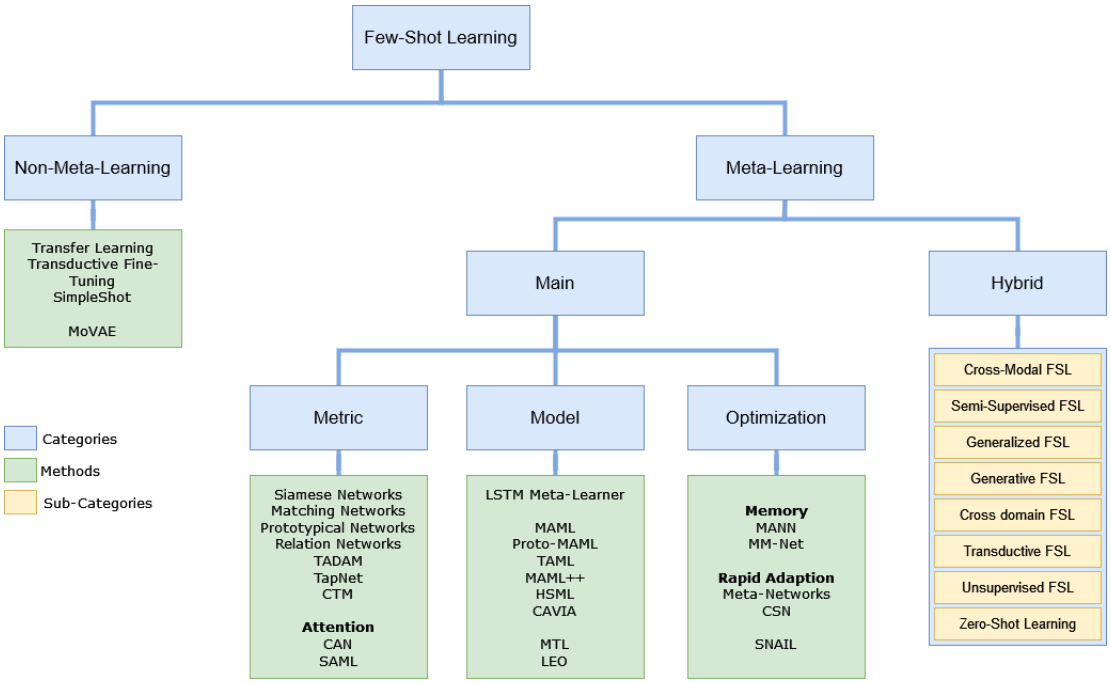


Figure 3.1: The categorization of FSL by Pernami et al. [22]

able to solve a new task at testing.

Within the literature, Meta-leaning is categorized into three approaches: metric-based, optimization-based, and model- based. Furthermore, there exist hybrid approaches to handle problems such as cross-domain FSL, generalized FSL, etc. However, since we are dealing with an N-way-K-shot classification setting, there is no need to explain these approaches further. Consider a few-shot task T with support set S and a query set Q to differentiate the three different approaches. Let f be a few-shot classification model with the parameters θ. Each approach has a different manner in which they model output probability $P_\theta(y|x)$ for $(x, y) \in$ Q, where x is the input image and y is the true class label. Table 3.1 displays the differences between the approaches [22] and Figure 3.1 provides a complete overview of FSL approaches.

In this work, we focus on the metric-based approach. As explained in Section 2.3, Metric learning is the task of learning distances from a dataset. Pernami et al. [22] stop the categorization at the point of metric. However, metric-based approaches within few-shot image classification have multiple directions. Figure 3.2 displays a categorization and its respective sub-categories [20]. 'Learning feature embeddings' refers to the method in which we assume that extracting discriminative features is sufficient for few-shot image classification. We expect that the two images are similar if they contain the same features. So, the ability to discriminate between features is the main priority for generalizing to a novel task. A well-known example is the convolutional Siamese neural network [15] for one-shot image classification. The Siamese neural network consists of a pair of identical CNNs with shared weights. By pushing two images through the CNNs, we extract the high-level features from the two images. We could then calculate the L_1-norm (Equation 2.4) between the two feature maps. With the calculated distance, the model can determine if the images belong to the same class or a different class. This approach is extended to three identical CNNs in triplet networks [10]. Instead of two images, we put three different images through the CNN: the query, a positive sample (same class), a negative sample (different class). The triplet networks minimizes the distance between the embedding of the query to the positive sample, while it maximizes the distance between the query and negative sample. Due to this technique,

Figure 3.2: Taxonomy of few-shot deep metric learning methods [20]

triplet models embed the data in such a way that a pair of samples with identical labels are closer than those with different labels. The Siamese neural network and triplet network fall under the 'learning task-agnostic features' sub-category. Most other sub-categories are not relevant for our research, except for 'learning task-specific features.' More specifically, Section 3.2 introduces the Categorical Traversal Module.

'Learning class representations' builds further on this idea of Siamese neural networks. However, it states that the discrimination between features of single-class instances, is likely to overfit. A class representation or distribution is rather more representative than a single-class instance. We no longer calculate the distance between two feature maps of the images, but rather the distance between a feature map and the class representation. Prototypical Networks [28] compute the class representation as the mean of the feature maps of the support examples of each class. However, these techniques are not the right fit for our research. As explained in Chapter 1, our research moves away from label/class approach.

These previously mentioned methods originally opted for already existing distance measures to measure the similarity. Minkowski distance or cosine similarity are default options. The last category provided another solution to distance computation. 'Learning distance or similarity measures' offered the idea of learning additional parameters to act as similarity/distance to improve performance. Relation Networks [31] consists of two modules: an embedding and a relation module. The embedding module has the same purpose as the Siamese network: recognition of high level feature. The relation module processes the query and support feature maps through additional layers to compute the similarity.

3.2 Categorical traversal module

The function of the Categorical Traversal Module (CTM) is to traverse across the entire support set at once, identifying task-relevant features based on both intra-class commonality and inter-class uniqueness in the feature space [19]. As an example, we consider a 2-way-4-shot with the classes *tiger* and *cheetah*. Each tiger has its recognizable coat of reddish-orange with dark stripes. Therefore, it appears to be an intra-class commonality or a feature presented within each image. The same applies to the cheetahs with their tawny coats covered in black spots. The coat for both animals is unique for their species. So, the difference in coats is the inter-class uniqueness. Both animals belong to the family of Felidea (cats) and have their commonalities. So, identifying task-relevant features could make a difference in correctly identifying a tiger or cheetah.

Figure 3.3 has one component that aligns with intra-class commonality and one component for inter-class uniqueness. $f_\theta(\mathrm{S})$ Refers to an embedding function $f(;\theta)$ applied to each instance of the support set, which is parameterized by a neural network with parameters θ. This embedding function extracts the high-level features from each image. The concentrator attempts to find universal features shared by all instances for one class. In the original work, each image is independently fed through a CNN to perform a dimension reduction. The next step is to average out over the classes. So, consider the output dimensions of $f_\theta(\mathrm{S})$ to be (NK, m_1, d_1, d_1), where m_1, d_1 are the number of channels and width. So, the concentrator does the following steps:

$$f_\theta(\mathrm{S}) : (\mathrm{NK},\, m_1, d_1, d_1) \xrightarrow{\mathrm{CNN}} o : (\mathrm{NK},\, m_2, d_2, d_2) \xrightarrow{\mathrm{mean}} o : (\mathrm{N},\, m_2, d_2, d_2)$$

The second component is the projector to mask out irrelevant features and select the ones most discriminative for the current few-shot task. It takes the features of the concentrator simultaneously into account. The first step in the original work was to reshape the dimensions of the features of the concentrator. The class prototypes in the first dimension (N) are concatenated to the channels (m_2). The next step is to apply another CNN over this reshaped data. The class prototypes are simultaneously processed as channels include the class prototypes. As the last step, we apply a softmax over the channel dimension of the output of CNN. So, the projector does the following steps:

$$o : (\mathrm{N},\, m_2, d_2, d_2) \xrightarrow{\mathrm{reshape}} \hat{o} : (1,\, \mathrm{N}m_2, d_2, d_2) \xrightarrow{\mathrm{CNN}} \hat{o} : (1,\, m_3, d_3, d_3) \xrightarrow{\mathrm{softmax}} p : (1,\, m_3, d_3, d_3)$$

The purpose of the reshaper is to align the dimension of the feature map of the support set with the dimensions of the projector:

$$f_\theta(\mathrm{S}) : (\mathrm{NK},\, m_1, d_1, d_1) \xrightarrow{\mathrm{CNN}} r(\mathrm{S}) : (\mathrm{NK},\, m_3, d_3, d_3)$$

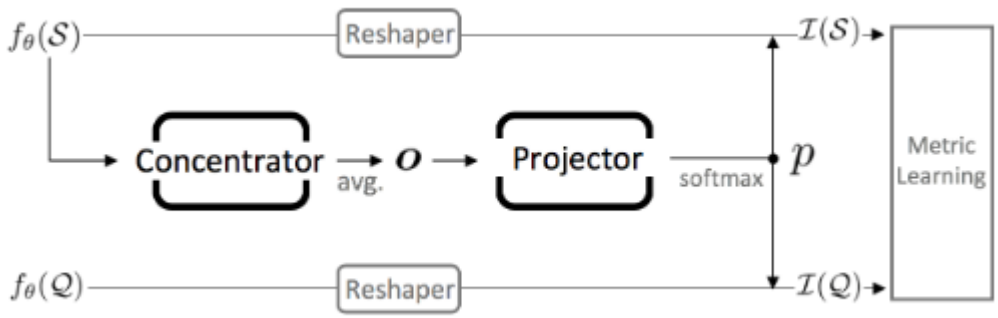


Figure 3.3: Detailed breakdown of components in CTM [19]

The last step of the CTM is to apply the output p of the projector to the support set and query. The best option in the original paper was to execute an element-wise multiplication between the projector output p and the reshaper output:

$$\mathrm{I(S)} = p \odot r(\mathrm{S}) : (\mathrm{NK},\, m_3, d_3, d_3)$$
$$\mathrm{I(Q)} = p \odot r(\mathrm{Q}) : (1,\, m_3, d_3, d_3)$$

Chapter 4

Problem Description

4.1 The effect of context on the perception of distance

Ambiguous visual stimuli provide the brain with sensory information that contains conflicting evidence for multiple mutually exclusive interpretations [14]. Klink et al. provide the example of the Necker cube in Figure 4.1. The brain interprets a two-dimensional line drawing as a three-dimensional cube. However, the cube's three-dimensional orientation provides ambiguity due to no explicit depth cues. Without any cues, we can perceive the second black vertical line from the left to be in front or at the back of our projected three-dimensional structure in Figure 4.1(a). The added context of the grey pipe in Figure 4.1(b) eliminates the ambiguity.

We could extend this notion of ambiguity in visual stimuli to differentiate between images. No explicit context or task surrounding two images leads to ambiguous answers. Consider the following example of the three images in Figure 4.2. Three different shapes are presented with three distinct aspects: thickness, color, and pattern. We will ask a group of participants to find the two most similar images. Each image has something in common with one other image. So, multiple answers present themselves. One person's perception might emphasize the importance

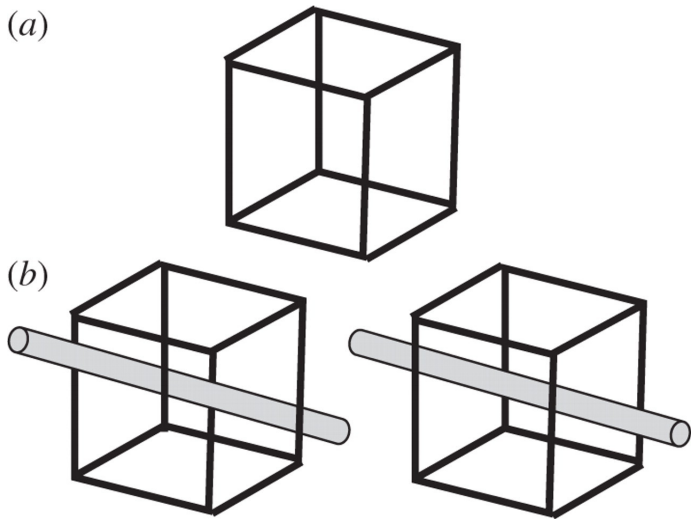


Figure 4.1: (a) A Necker cube: a two-dimensional line drawing of a cube. (b) Additional information can resolve the perceptual ambiguity [14]

(a) Thick orange triangle with diagonal stripe pattern

(b) Thin grey hexagon with diagonal stripe pattern

(c) Thin orange rhombus with rhombus pattern

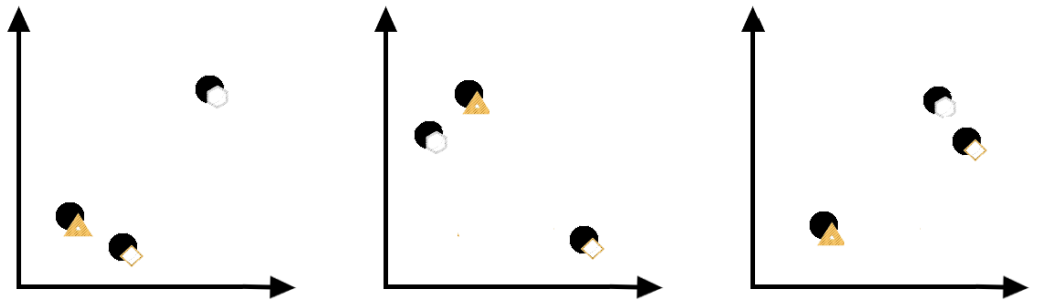
Figure 4.2: Three images that can be perceived as similar based on a different aspect: color, thickness, and pattern

of pattern over color, and another person's perception vice versa. Instead of adding a grey pipe (Figure 4.1), we offer the participants an aspect-based task. From now on, they have to find the most similar images based on color, which leads to the unambiguous answer of Figure 4.2(a) and Figure 4.2(c). We perceive those images as relatively close to each other based on the color. The answer changes when asked to differentiate based on the pattern. Figure 4.2(a) and Figure 4.2(b) have the same kind of diagonal stripe pattern. We perceive the distance between Figure 4.2(a) and Figure 4.2(c) as a lot larger than when asked about the color. The same applies to the aspect of thickness between Figure 4.2(b) and Figure 4.2(c). The perceived distance differs based on the presented context/aspect around the image. Each has a mutually exclusive or unambiguous answer if the correct aspect is present. However, removing the task alongside Figure 4.2 leads to ambiguous answers. Therefore, it is instrumental to incorporate such aspects into the design of metric-based FSL frameworks to find the relative distance.

4.2 Aspect-based embedding space

We want to translate this aspect-based relative distance into a machine learning model. To this extent, we propose an aspect-based embedding space. In deep learning, an embedding is a relatively low-dimensional representation (vector) of values or real-life objects, such as images, text, etc. Ideally, data points with similar semantic concepts are represented by a similar vector within the embedding space. With the computation of Metric distance (Euclidean distance, cosine similarity, etc.) over embedding vectors, we can estimate similarity between data points. However, Section 4.1 demonstrated that the perceived similarity/distance depends on the aspect. Therefore, one embedding space does not provide enough information to capture the semantic similarity between pairs of images. This issue leads to the introduction of aspect-based embedding space.

We hypothesize that the determined aspect can be taken into account during the embedding of the input data to capture only the relevant features concerning the aspect. Differentiating the aspect would lead to a different embedding of the same input data. We refer to this embedding space where the aspects alter the embedding of data points as the aspect-based embedding space. We can compute similarities in the aspect-based embedding space with a metric distance, similar to one embedding space. However, the distances are no longer fixed due to different manners of embedding concerning the aspect. Figure 4.3 demonstrates the aspect-based embedding space. We notice the embedding of the orange rhombus and orange triangle are close together in the color-based embedding space. At the same time, they are far apart when we compare them in the pattern-based- and thickness-based embedding space. Each aspect leads to a different similarity based only on the relevant feature.



(a) Color-based embedding space: the distance between the embedding of the orange triangle and the embedding of the orange rhombus is small, while they have a larger distance to the embedding of the grey hexagon

(b) Pattern-based embedding space: the distance between the embedding of the triangle with diagonal stripe pattern and the embedding of the hexagon with diagonal stripe pattern is small, while they have a larger distance to the embedding of the rhombus with rhombus pattern

(c) Thickness-based embedding space: the distance between the embedding of the thin hexagon and the embedding of the thin rhombus is small, while they have a larger distance to the embedding of the thick triangle

Figure 4.3: 2D aspect-based embedding space. The relative distances between the embeddings are different per aspect.

4.3 Scope of the problem

This work addresses the aspect-based few-shot learning problem where the novel challenge is to determine commonalities and aspects shared among the data in the support set. We want to learn an embedding that highlights the features concerning the determined aspect. To solve this challenge, we address the following points:

- This project solely focuses on the domain of image recognition, where there are different factors based upon which comparison for similarities can be made such as thickness, color, pattern, or stance.
- The data generation process of the data is completely known. The data in this work does not represent real-life examples. The main point of this research is to demonstrate the possibilities of shifting the direction of FSL to an aspect-based approach.

4.4 Problem formulation

Consider the standard FSL setting in which we have a base dataset D_{base}(Equation 4.1) and a novel dataset D_{novel}(Equation 4.2). There is no overlap of labels between the base dataset and novel dataset, i.e., $y_{base} \cap y_{novel} = \emptyset$. The premise of FSL is to leverage the limited data observed in y_{base} to learn a good classification model for the unseen labels y_{novel}. We present the model with a support set S of N distinct, previously unseen labels, with K examples of each class and a query Q to evaluate performance. The goal is to find a match between the query and a data point of the support to assign the correct label to the query.

$$D_{base} = \{(x_i, y_i)\}_{i=1}^{t}, \tag{4.1}$$

$$D_{novel} = \{(x_i, y_i)\}_{i=t+1}^{n}, \tag{4.2}$$

where x_i are the data points, y_i the labels, and t number of training samples.

The current FSL assumes that the single class assignment y_i is the generating factor where we want to match for this one class assignment. This work proposes to change the assumption of one generating factor to a set of multiple generating factors for each data point. We remove the direct mapping from x to single factor y. Furthermore, we assume that extracting the set of generating factors is viable without any mapping. Equations 4.3 and 4.4 portray the data for aspect-based FSL. Each of these generating factors could lead to a match between various data points. If one of the factors is the same for each support set data point, it does not give any information regarding the matching of the query. The similarity between the query and support set would be the same for this factor. These shared factors/features among instances of the support set are the previously mentioned *commonalities*. We would rather need to look at the factors that differentiate the support set instances from each other. These defining factors/features are denoted as aspects. Aspects dictate the similarity between query and data points of the support set because the similarity between query and support set instances would differ from this factor. So, the objective behind the aspect-based few-shot classification task is to learn a model capable of exploiting the training examples provided in each support set S to extract commonalities and aspects.

$$D_{base} = \{(x_i)\}_{i=1}^{t}, \text{ where } x_i \tag{4.3}$$

$$D_{novel} = \{(x_i)\}_{i=t+1}^{n}, \tag{4.4}$$

where x_i are the data points, and t number of training samples.

Let $f(;\theta)$ denote an embedding function to extract high-level features or the generating factors, which is parameterized by a neural network with parameters θ. This function is used to extract the embeddings of the images x, represented by $f(x;\theta)$. These high-level features are utilized to determine the commonalities and aspects within the supports. Formally, given the set of embeddings $\{f(x_0;\theta)$, ..., $f(x_N;\theta)\}$ for x_0, ..., $x_N \in$ S, we need to learn an additional function that extracts the commonalities and aspects from this set. The representation of the set is utilized to compute the aspect-based embeddings for the query images and the images of the support set. We can calculate the distance between the aspect-based embeddings. The extraction combined with computation of the aspect-based embedding space is denoted as I. Formally, the aspect-based embedding of the query and support sets are denoted as I(f_θ(S)) and I(f_θ(Q)).

The objective is to move away from label-based supervision. Therefore, we do not have the commonly available label information in an FSL task. However, as stated in Section 4.3, we know the underlying data generation process. Therefore, the data is sampled based on their pre-defined commonalities and aspects to create a support set. In this work, our support set conforms to the N-way 1-shot setting. There are no classes present in this research. Therefore, we acknowledge each image as its class (K=1).

Chapter 5

Solution approach

This chapter explains the requirements and their respective solutions for the task of aspect-based FSL. The first requirements address data generation that aligns with aspect-based FSL. Currently, there are no data sets available. The images must have multiple features to illustrate different aspects of the support set. Furthermore, we want complete control over the data generation process without any noise in the data to assess the results correctly. The solutions to these requirements are provided in Sections 5.1.1 and 5.1.2. The second requirements concentrate on the creation of the support sets. We enforce the no exact match between the query and support set images. The images in the support set will differ from the query. The matching between support set images and query must be unambiguous. Our support set should not allow for multiple answers based on personal perception. Moreover, we want to include the same images in several support sets to be perceived differently during training. Figures 1.2, 1.3, and 1.4 have different perceptions of the same second dog. Our support set generation process has to capture the same behavior. The solutions to these requirements are found in Sections 5.1.3, 5.1.4 and 5.1.5. The third requirement in Section 5.1.6 addresses the problem of the standard splitting procedures not aligning with the task of aspect-based FSL. Another requirement regards the capacity to extract commonalities and aspects from the support set. We need a permutation-invariant machine learning model that is capable of this task. We discuss the architecture of this machine learning model in Section 5.2. The last requirement in Section 5.3 regards the training with multiple negative examples to account for the intra-conceptual relationships of the support set.

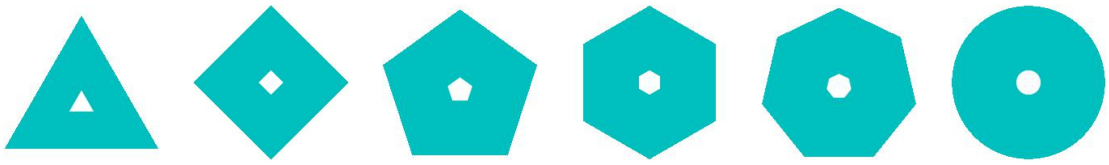
5.1 Data generation

5.1.1 Data description

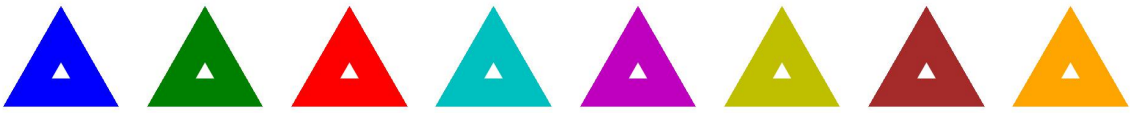
The experiments of Chapter 6 are implemented using two different generated data sets. We have complete control over the underlying data generation process with our own data generation. The control does not allow for any ambiguity, as we know what the data entails. The generated data sets must consist of multiple features to illustrate different aspects of the support set. Furthermore, the data requires a lack of noise. The clean data strictly consists of relevant information for the task of aspect-based FSL. Therefore, we eliminate any inconsistencies that could influence our performance.

These requirements lead to the generation of the first data set. This data includes two-dimensional geometric shapes with an image size of 112 by 112 pixels. Each image is one of the following single-holed polygon shapes: triangle, rhombus, pentagon, hexagon, heptagon, or circle. Each shape has a different filling color, thickness, and pattern.

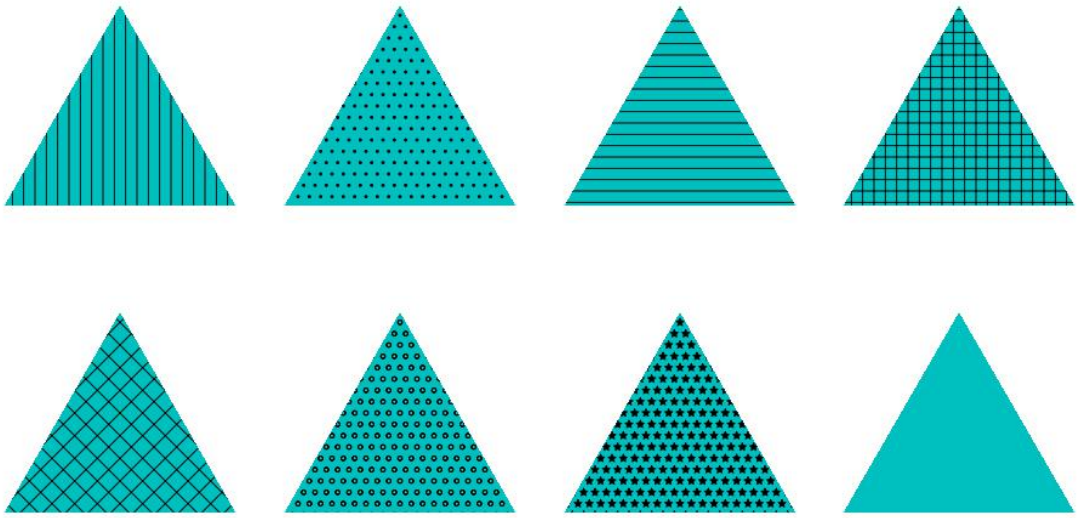
The second generated dataset is derived from generated data for the disentanglement of sequential data. The sprites data set [21] utilizes the sequence of frames (animation) from 2D sprites



(a) Every shape in geometric shape data (color = 'cyan,' thickness = 0.16, pattern = 'No')



(b) Every color in geometric shape data (shape = 'triangle,' thickness = 0.16, pattern = 'No')



(c) Every pattern in geometric shape data (shape = 'triangle,' thickness = 0.16, color = 'cyan')

(d) Every thickness in geometric shape data (color = 'cyan,' shape = 'triangle,' pattern = 'No')

Figure 5.1: The effect of each parameter on the features of the geometric shapes data set

from the open-source video game project called Liberated Pixel Cup [1]. Available sprite sheets [2] offer the possibility to create customized characters. In this work, we are not interested in the complete sequence, only individual frames. Selected frames represent the physical stance aspect. Each image contains a different (fictional) body type: human, orc, alien, skeleton, pumpkin man, pig, goblin, and mouse. Furthermore, the images involve a difference in shirt-, pants, and hair color.

5.1.2 Feature design

The previous features are considered parameters during the generation of the two data sets. First, we generate the geometric shapes data using the MatPlotlib Library [11, 34] in Python. The generation involves the following features:

- Shape: triangle, rhombus, pentagon, hexagon, heptagon, and circle. (Figure 5.1(a))
- Color: blue, green, red, cyan, magenta, yellow, brown, and orange. (Figure 5.1(b))
- Pattern: vertical line, dots, horizontal line, horizontal and vertical line, crossed diagonal pattern, circles, stars, and no pattern. (Figure 5.1(c))
- Thickness: 0.0, 0.11, 0.22, 0.33, 0.44, 0.55, 0.66, and 0.77. A higher value indicates a thinner shape. (Figure 5.1(d))

The complete data set contains one image for each combination of parameters. So, our first data consists of $6 \times 8 \times 8 \times 8 = 3{,}072$ different images. The parameters are selected based on the idea that we can differentiate between the features. For example, a tiny change in thickness is difficult to notice. Therefore, we limit the number of values per feature to have enough change between images.

Second, we generate the 2D sprites with hand-selected sprite sheets [2]. The generation involves the following features:

- Body: human, orc, alien, skeleton, pumpkin man, pig, goblin, and mouse. (Figure 5.2(a))
- Stance: standing, praying, mountain pose, spreading arms, unsheathing sword, dancing, extending left arm, fighting stance, and drawing bow.(Figure 5.2(b))
- Hair: red, light brown, white, black, blue, green, yellow, dark brown, pink, and purple. (Figure 5.2(c))
- Shirt: purple, orange, red, white, turquoise, yellow, green, navy, dark brown, and black. (Figure 5.2(d))
- Pants: navy, dark brown, green, purple, orange, red, white, turquoise, yellow, and black. (Figure 5.2(e))

This data set contains $8 \times 9 \times 10 \times 10 \times 10 = 72{,}000$ different images. Within this data set, we also want enough difference between the images. For example, the subsequent frames of each animation were often close. So, many stances were left out of the data to achieve the variation.

5.1.3 Generation of support set

A support set has several requirements to account for. First, the query image and images of the support set do not share the same object: shapes in the geometric shapes data set and bodies in the sprites data set. We do not recognize objects as an aspect. Object recognition requires the simultaneous recognition of multiple features at once. For example, a rhombus has several properties that hold to be classified as a rhombus. All sides of a rhombus are equal, diagonals bisect each other at 90°, opposite sides are parallel in a rhombus, opposite angles are equal in a rhombus, and adjacent angles add up to 180°. Additionally, we enforce the idea of no exact match between support and query by excluding objects as an aspect.

Second, each support set needs to represent only one aspect (defining feature), and every other feature is a commonality (shared feature). Representing multiple aspects in a support set leads to ambiguity, while this work aims to have a controlled environment without ambiguity. Additionally, we know that the model's decision could only be based on the aspect by reducing every other feature to a commonality. Thus, there are no differences between the images of the supports except for the aspect.

(a) Every body in sprites data (hair = 'red,' shirt = 'purple,' pants = 'navy,' stance='standing')

(b) Every stance in sprites data (hair = 'red,' shirt = 'purple,' pants = 'navy,' body = 'human')

(c) Every hair in sprites data (stance='standing,' shirt = 'purple,' pants = 'navy,' body = 'human')

(d) Every shirt in sprites data (stance='standing,' hair = 'red,' pants = 'navy,' body = 'human')

(e) Every pants in sprites data (stance='standing,' hair = 'red,' shirt = 'purple,' body = 'human')

Figure 5.2: The effect of each parameter on the features of the sprites data set

Third, the number of features that are shared between the images of the support set and the query's features differ per generated support set. One feature is always an aspect. So, only one image of the support set matches the value of that feature with the query image. The commonalities of the support set do not necessarily have to match the query. The second point states that we control the support set to decide based on the aspect. So, two features that match the query should not give a different answer than zero shared features given the same value. However, Section 5.1.4 explains the necessity of varying the number of shared features of the support set with the query

Figure 5.3 displays an example of a support set(N=3) for each data set. We notice the query is a human, while the support set only consists of orcs in Figure 5.3(b). So, there is no exact match, and we must look at the features. The commonalities among the instances of the support set are the hair, shirt, and pants. The hair matches the query, while the pants and shirt differ from the query. The displayed aspect is the stance. Each image of the support set has a different stance. Therefore, the query matches with the second image as they are the same according to the aspect of this specific support set.

(a) No exact match due to the different shapes. The pattern of each image of the support set is the same as the query, while the thickness differs from the query. The aspect of the support set is the color.

(b) No exact match due to the different bodies. The hair of each image of the support set is the same as the query, while the pants and shirt differ from the query. The aspect of the support set is the stance.

Figure 5.3: An example of a generated support set for each data set

5.1.4 Varying shared features with the query

It is necessary to vary the number of shared features with the query. Without it, we cannot include the context in the embedding space of an image. Figure 5.4 shows that we perceive the same image differently due to the various shared features. Figures 5.4(a) and 5.4(b) have the same query and third image of the support set. Figure 5.4(a) has the aspect of stance and one shared feature (hair), while Figure 5.4(b) has the hair aspect and no shared feature. In the first example, we do not match the third image to the query, but we do for the second image. The third image's embedding of the first example should differ from the second example. So, varying the shared

(a) No exact match due to the different bodies. The hair of each image of the support set is the same as the query, while the pants and shirt differ from the query. The aspect of the support set is the stance.

(b) No exact match due to the different bodies. There is no shared feature between the support set and the query. The aspect of the support set is the hair.

Figure 5.4: The number of shared features dictates the aspect-based embedding space. The same image is perceived differently in another support set.

(a) No exact match due to the different bodies. The shirt, stance, and pants of each image of the support set are the same as the query. The aspect of the support set is the hair.

(b) No exact match due to the different bodies. The shirt and pants of each image of the support set are the same as the query, while the hair differs from the query. The aspect of the support set is the stance.

(c) No exact match due to the different bodies. The pants of each image of the support set are the same as the query, while the hair and stance differ from the query. The aspect of the support set is the shirt.

(d) No exact match due to the different bodies. There is no shared feature between the support set and the query. The aspect of the support set is the pants.

Figure 5.5: The generation of four different support set for the same query

feature forces the inclusion of the support set into the embedding space of the individual images.

5.1.5 Data sequence

The sequence of data points for training the model is semi-randomized. The randomization happens at three different parts of the generation of the support sets. The first part is to determine the order of aspects. We cover each aspect per query. Figure 5.5 shows an example of the sprites dataset where the order is the following: the hair as the first aspect, the stance as the second, the shirt as the third, and the pants as the last. The next randomization step of support set generation removes the exact match. It randomly selects a different body in the sprites data set

or the shape in the geometric data set (to the query). Figure 5.5 has the query of a human, while only pigs in the support sets. Then, we randomly select N(size of support set) - 1 values for the aspect to complete the support set. The position of the matching image of the support set is also randomized. These steps lead to the first support set of Figure 5.5(a).

One of the negative examples in the first support set is then picked to be the match in the next support set. The predetermined next match explains why the procedure is semi-randomized. We want to ensure the same image presents itself in different contexts/support sets during training. Additionally, we represent the same image as a negative example and a match, which could only be explained by accounting for the support set. The next step is the same as the generation of the first support set: randomly select N - 1 values for the aspect, determine the position of the matching image, and select the match for the next support set. Figure 5.5(b) is the result of repeating the procedure with a negative example of Figure 5.5(a). We repeat this procedure until the negative examples do not match any feature of the query (Figure 5.5(c) and 5.5(d)).

There are four different generated support sets in the case of the sprite data set (Figure 5.5). We have four possible aspects presented within the data. The same procedure can be applied to the geometric shapes data set. However, this data set only contains three possible aspects. Therefore, there are only three generated support sets per query.

5.1.6 Definining test data

The common approach of training- and test split of the data in the FSL setting is no longer viable. In the traditional setting, each image is expected to belong to a class where the number of classes is smaller than the number of images in the data. In this work, that is no longer the case. However, we can classify each image as a separate class. In the controlled setting of this work, each image differs on at least one feature value. So, each image can be classified as a separate class due to

(a) No exact match due to the different bodies. The pants and shirt of each image of the support set are the same as the query, while the hair differ from the query. The aspect of the support set is the stance. is the pants.

(b) No exact match due to the different bodies. The hair of each image of the support set are the same as the query, while the shirt and pants differ from the query. The aspect of the support set is the stance.

Figure 5.6: The same support set for different queries. The features of the training query in (a) differs completely from the testing query in (b). The matching between query and support set images differs accordingly.

the guaranteed difference. Real-life data does not have this guarantee due to the unknown data generation. Nonetheless, it is still a viable option without the guarantee. Real-life image data is unlikely to match each of the many different features. So, the first solution is to treat each image as a class, the so-called data unique split. By classifying each image uniquely, we can do the common FSL training- and test split. The training data consists of classes that do not appear in the test data ($C_{training} \cap C_{test} = \emptyset$) as there are differing images for training compared to testing.

The second approach would be a so-called query split. The images in the support sets can be every image from our data, while we have two sets of queries: training queries and testing queries. Such an approach highlights two important parts. First, the perception of similarity differs based on the support set. So, it should not matter that an image seen during training occurs in a support set. It would show the model's adaptability in dealing with the same image in different support sets. There is the possibility of having the same support set in training and testing. However, that is where the query separation matters. The query separation is meant to evaluate the correctly extracted aspect within each image. Figure 5.6 shows the same support set. The query in Figure 5.6(a) differs completely from the query in Figure 5.6(b). The distance from the first support set image to the second query should be significantly lower than the distance to the first query. We can evaluate if the support set extracts the value of the aspect for each image within the support set by separating training- and testing query images. Especially when the first image of the support set in Figure 5.6(a) will always be seen as a negative example of the aspect of stance during training. Furthermore, the testing queries have never been seen in the aspect-based embedding space for this support set during training. So, the overlap in images in the support set for training and testing does not pose a problem.

Chapter 6 explains how both splits look in a controlled setting.

5.2 Deep Set Traversal Module

The proposed model in this section takes inspiration from the CTM. However, the module does not meet one of the requirements: permutation-invariance. Before the projector, the data is concatenated to the channel dimension to feed it through a CNN module. Convolutional layers do not provide invariance over the channels. So, the order of examples in the support set affects the final embedding. We propose the Deep Set Traversal Module (DSTM) to include permutation invariance within the model.

In Figure 5.7, we replace the concentrator with a permutation-equivariant deep set model. The purpose of the model is to enrich the embedding of each element with the information of the other members of the support set:

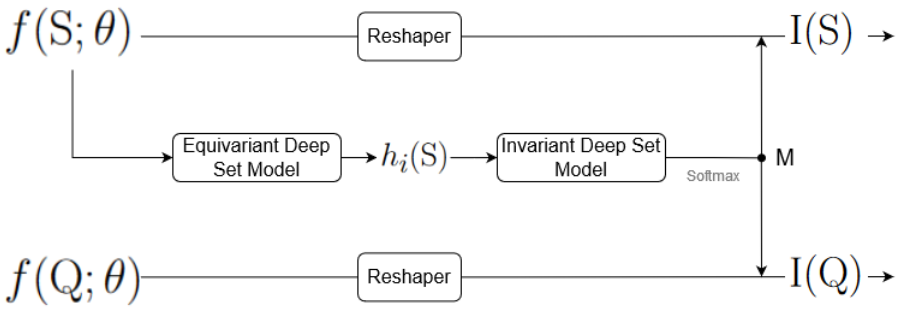


Figure 5.7: Detailed breakdown of components in DSTM

$$h_i = h\Big[f(x_i;\theta),\ g_i;\lambda\Big] \text{ where } g_i = \underset{j\in N(i)}{P}\Big(g\big[f(x_j;\theta);\mu\big]\Big),$$

N is the neighborhood and $f(;\ \theta)$ is an embedding function before the permutation-equivariant model. This model attempts to extract the values of the aspect of each support set image by communicating the representation of the neighboring images. First, it process the information of each embedding in $f(\mathrm{S};\ \theta)$ individually through embedding function $g(;\mu)$. Afterward, we apply the permutation-invariant operation P over the neighboring embeddings. We denote the collection of permutation-invariant representations as $g_i(\mathrm{S})$. $g_i(\mathrm{S})$ is concatenated to the original embedding $f(\mathrm{S};\ \theta)$ and put through a final embedding function $h(;\lambda)$, which leads to the final representation $h_i(\mathrm{S})$. Each embedding function can be a simple convolutional layer with ReLU activation or a ResNet block. Consider the output dimensions of $f(\mathrm{S};\theta)$ to be $(\mathrm{N},\ m_1, d_1, d_1)$, where m_1, d_1 are the number of channels and width. So, the permutation-equivariant model does the following steps with additional visual representation in Figure 5.8:

$$f(\mathrm{S};\theta) : (\mathrm{N},\ m_1, d_1, d_1) \xrightarrow{\text{CNN}} g(\mathrm{S};\mu) : (\mathrm{N},\ m_2, d_1, d_1) \xrightarrow[\text{neighbours}]{\text{P over}} g_i(\mathrm{S}) : (\mathrm{N},\ m_2, d_1, d_1)$$
$$\xrightarrow[\text{concatenation}]{\text{with } f(\mathrm{S};\theta)} f(\mathrm{S};\theta), g_i(\mathrm{S}) : (\mathrm{N},\ m_1 + m_2, d_1, d_1) \xrightarrow{\text{CNN}} h_i(\mathrm{S}) : (\mathrm{N},\ m_3, d_2, d_2)$$

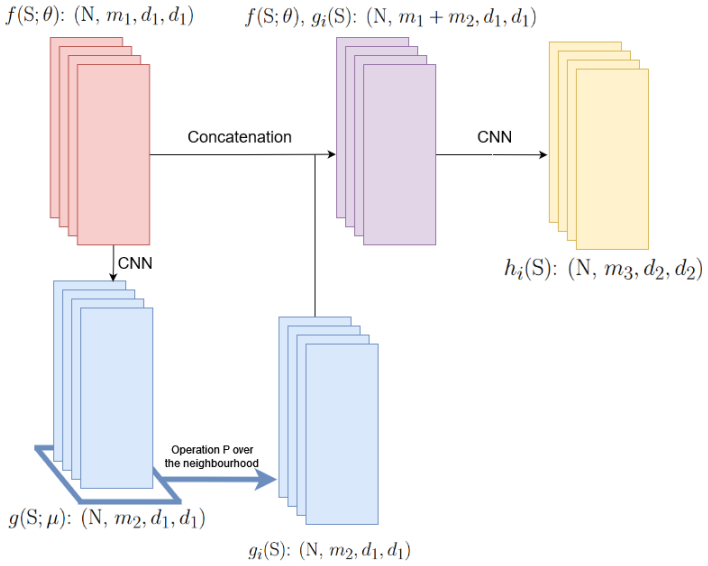


Figure 5.8: The permutation-equivariant deep set model

The second component is the permutation-invariant deep set model. We apply an additional operation P over updated embeddings $h_i(\mathrm{S})$ to destroy the information regarding the ordering of the set:

$$O = k(P\left[h_i(\mathrm{S})\right];\eta)$$

The permutation-invariant representation of the set is processed by another neural network $k(;\ \eta)$ to have a final output embedding representing the complete set. As the last step, we apply a Softmax over the channel dimension of the output of CNN as with the CTM. So, the permutation-invariant model does the following steps with additional visual representation in Figure 5.9

$$h_i(\mathrm{S}) : (\mathrm{N},\, m_3, d_2, d_2) \xrightarrow[\text{embeddings}]{\text{P over}} P\left[h_i(\mathrm{S})\right] : (1,\, m_3, d_2, d_2) \xrightarrow{\text{CNN}} O : (1,\, m_4, d_3, d_3)$$
$$\xrightarrow{\text{Softax}} M : (1,\, m_4, d_3, d_3)$$

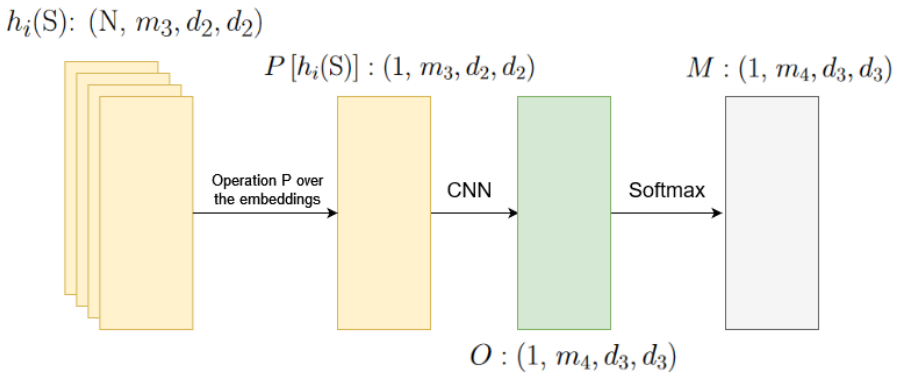


Figure 5.9: The permutation-invariant deep set model

The purpose of the reshaper is to align the dimension of the feature map of the support set and query with the dimensions of M:

$$f_\theta(\mathrm{S}) : (\mathrm{NK},\, m_1, d_1, d_1) \xrightarrow{\text{CNN}} r(\mathrm{S}) : (\mathrm{NK},\, m_3, d_3, d_3)$$

The last step of the CTM is to apply the output p of the projector to the support set and query. The best option in the original paper was to execute an element-wise multiplication between the output permutation-variant deep set model M and the reshaper output:

$$\mathrm{I(S)} = p \odot r(\mathrm{S}) : (\mathrm{NK},\, m_3, d_3, d_3)$$
$$\mathrm{I(Q)} = p \odot r(\mathrm{Q}) : (1,\, m_3, d_3, d_3)$$

5.3 Deep Metric Learning

5.3.1 Loss function

A support set must contain at least two images to extract an aspect: a positive and a negative example. A positive example in this work is the image closest to the query given the aspect. In the case of two support set images, a triplet network [10] would suffice. The triplet network minimizes the distance between the embedding of the query to the positive sample while maximizing the distance between the query and the negative sample. Triplet models embed a pair of samples with identical labels closer together compared to pairs with different labels. The triplet loss [25] (Equation 5.1) enforces the anchor embedding to be closer to the positive example than to the negative example by some margin $m > 0$.

$$L_{triplet}(x, x^+, x^-) = \max(0, m + \|f(x) - f(x^+)\|_2^2 - \|f(x) - f(x^-)\|_2^2) \tag{5.1}$$

However, the support set contains at least two support set images. The triplet loss does not suffice for support sets with more than one negative example. So, we propose to use the tuplet loss [29]. The tuplet loss generalizes the triplet loss to explore multiple negative examples simultaneously. Instead of the triplet (x, x^+, x^-), we have a tuplet defined as $(x_a, x^+, x_1^-, .., x_{N-1}^-)$, where N is the size of the support set. The tuplet loss (Equation 5.2) enforces the anchor embedding

to be closer to the positive example, as does the triplet. However, the loss pushes away all negative samples simultaneously instead of only one negative example. Figure 5.10 demonstrates the idea for an example of the sprites data set. Based on the aspect of the shirt, we want to push the image with an orange shirt toward the anchor while forcing every other shirt away from the anchor.

$$L_{tuplet}(x, x^+, x_1^-, .., x_{N-1}^-) = \log\left(1 + \sum_{j=1}^{N-1} e^{\|f(x)-f(x^+)\|_2^2 - \|f(x)-f(x_j^-)\|_2^2}\right) \tag{5.2}$$

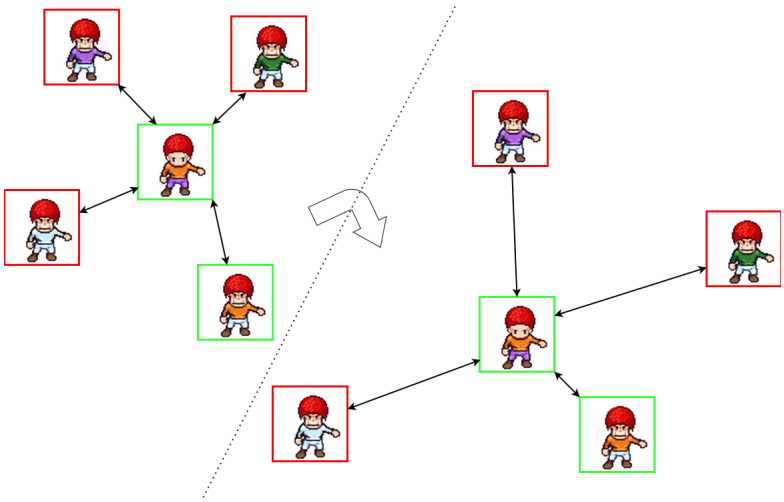


Figure 5.10: Demonstration of the tuplet loss on the aspect of the shirt: the positive example gets closer to the anchor, while every negative example moves away.

The training of aspect-based FSL contains multiple support sets for one query, as stated in Section 5.1.5. Therefore, the total loss is the sum of the tuplet losses of each support set for a query:

$$L_{total} = \sum_{j=1}^{|S|} L_{tuplet}^j \text{ where } |S| \text{ is the amount of support sets}$$

5.3.2 Aspect-based metric network

The aspect-based metric network consists of three components. The first step of representation learning extracts the features from the input data. In this work, we are dealing with image data. Therefore, the representation learning model is a CNN. After feature extraction, the DSTM transforms the embedding space into an aspect-based embedding space. The embedding of support set embeddings is put through a permutation-equivariant deep set model to highlight the value of the aspect and filter out the commonalities. The next step is to process the highlighted values by a permutation-invariant deep set model to create a mask. This mask emphasizes the relevant features within the original embeddings from the representation learning to end with an aspect-based embedding space. The last step is to use metric learning over the aspect-based embedding space. The tuplet loss offers the possibility to deal with multiple negative examples simultaneously. Figure 5.11 shows the complete overview of the aspect-based metric network.

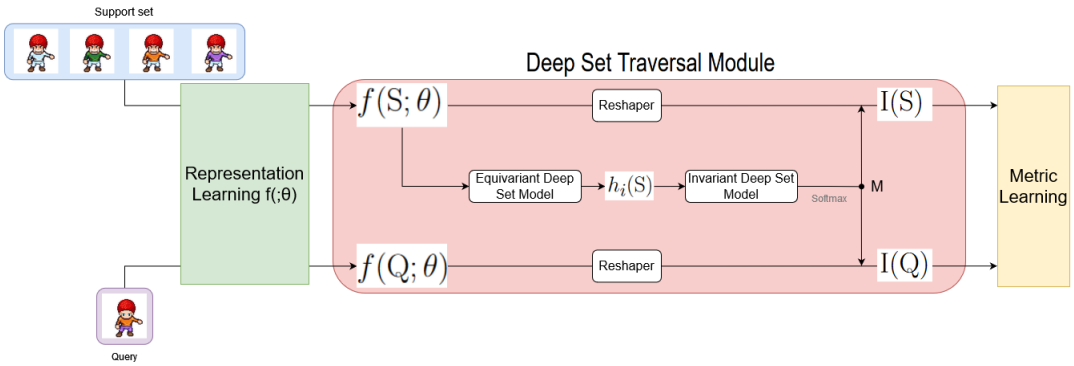


Figure 5.11: The aspect-based metric network

Chapter 6

Experiments

6.1 Dataset

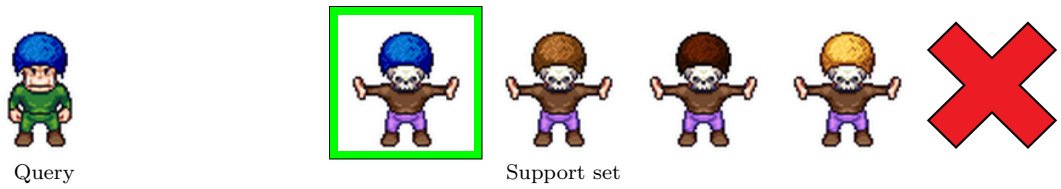
Section 5.1 described both data sets extensively. Theoretical separation of the data was given in two manners in Section 5.1.6. Both splits utilize no overlap in feature values of the query images ($Q_{training} \cap Q_{validation} \cap Q_{test} = \emptyset$). This way, the answer to the query is always a different image of the support set, even if the support set is the same (example of Figure 5.6). The main purpose of completely independent queries is to demonstrate that the model correctly highlights the aspect values of the complete support set. The model has never seen that feature value be the correct answer during training. We assign five values of each feature to the training, one feature value to the validation, and the remaining values to the test in the sprite data set. The training contains four feature values in the case of the geometric shape data set due to fewer feature values. The support set of the sprite data set consists of 4 images, while the geometric shape support set consists of 3 images.

Now, we have to differentiate between the two separation methods. The construction of the support set is based on the feature values of the training set when we treat every image as a separate class in the data unique split. For example, a support set of the sprite data set during training only sees the five different values for each feature. It does not mean training features are not present within support set images of the validation and test set. The individual feature values can appear combined with other not-seen features of the validation and test. Figure 6.1 shows a possible training and testing example. The feature value of green hair appears during testing (Figure 6.1(b)) but is not seen in combination with the green outfit (Figure 6.1(a)). Therefore, it follows the separation of the data. The only downside of this approach is the evaluation when no feature values match between the query and support set images (last support set of training sequence). The remaining features in the test and validation set are too few to construct a complete support set. So, there is a chance that images from training appear in the last support set due to the completely different feature values of the query. Figure 6.1(c) shows what could happen in that case. Therefore, we have to exclude that support set during evaluation.

In the case of only separating between queries, we do not have any limitations on the training support sets. The features of the training appear during the testing phase and vice versa. The support sets of training could happen to be the same support set during testing. However, it should not be an issue, as stated earlier. The answer would differentiate due to the independence of feature values of testing and training. We refer back to Section 5.1.6 for an example.

(a) A training example: no exact match due to the different bodies. The shirt, movement, and pants of each image of the support set are the same as the query. The aspect of the support set is the hair.

(b) A test example: no exact match due to the different bodies. The shirt and pants of each image of the support set are the same as the query, while the hair differs from the query. The aspect of the support set is the movement.



(c) A test example: No exact match due to the different bodies. There is no commonality within the support set that matches the query, while the movement, pants, and shirt differ from the query. The aspect of the support set is the hair.

Figure 6.1: Feature values seen during training do not matter as the complete image does not match. (b) has a not-seen green shirt and pants during training. So, the green hair of the training is no problem for the support set. However, the support set of (c) contains three images from training. Therefore, it does not comply with the separation.

6.2 Model architecture

In this section, we discuss the architecture of the aspect-based metric network.

6.2.1 Representation learning

We provide three options for representation learning: a shallow representation model, a small VGG architecture, and a small ResNet architecture. The shallow representation model is a simple architecture of convolutional layer + batch normalization + ReLU followed by a pooling layer. Each layer has an increasing amount of filters. VGG16[27] and ResNet-34[9] contains too many layers for this dataset. So, it would likely over-fit in the case of these large structures. However, the techniques of both architecture could have their merit. Therefore, we opt for a small VGG and a small ResNet model. Figure 6.2 shows each structure in more detail. Each model is trained from scratch.

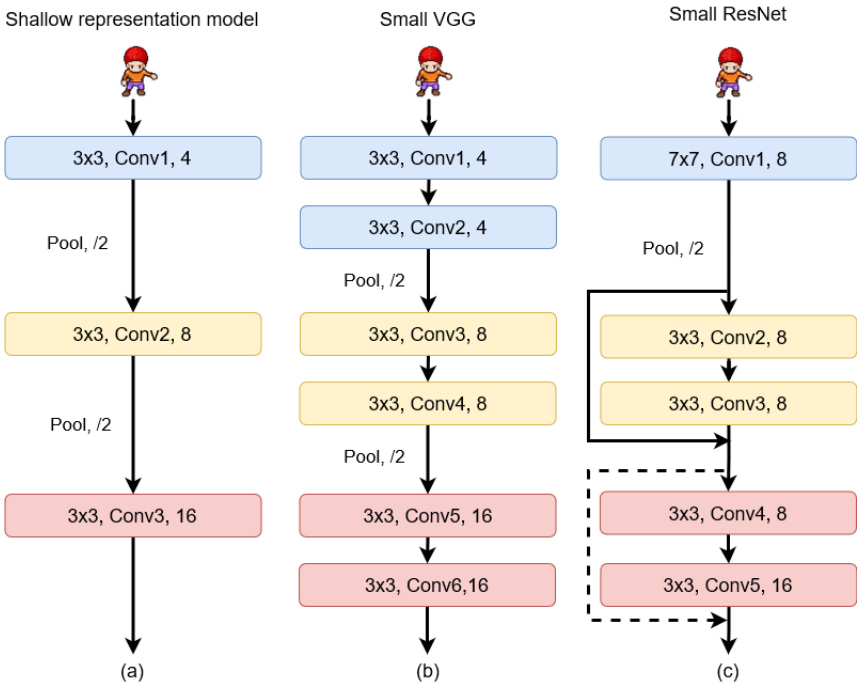


Figure 6.2: The layers of each representation model

6.2.2 Deep Set Traversal Module

Within the deep set traversal module, we experiment with two types of layers in the permutation-equivariant and -variant deep set models. The first option is to have a single convolutional layer + batch normalization + ReLU for the embedding function $g(;,\mu)$ in the permutation-equivariant model and a single layer for the embedding function $k(;\eta)$ in the permutation-invariant model. The second option is to have a single residual block at those places in the model instead.

6.3 Experiment description

We compare the different structures of the aspect-based model versus a baseline model. The baseline model would be the shallow representation model without the DSTM. Without the DSTM, we can see the effect of the module on the embedding space. We also compare the different architectures. We expect that the architectures with the implemented DSTM outperform the baseline.

Furthermore, we want to study the viability of both splitting methods. Due to the training times, we do not consider every iteration of the aspect-based model for this experiment. We also study how different aspects affect the performance. We expect that different aspects are embedded differently by the representation model. Therefore, the performance per aspect will differ.

Additionally, we look into the possible effect of the varying shared features with the query image. The best-case scenario shows no difference in performance when the number of shared features differs.

6.4 Performance evaluation

The accuracy is not the correct evaluation method for these experiments. The positive examples in our support sets always have one feature more comparatively with the query than our negative

examples. Therefore, a non-aspect-based embedding space would also capture the smallest distance. Rather, the distance between the query and positive examples and between the query and negative examples is a relevant evaluation method. For simplicity, we refer to the distance between the positive example and query as the positive distance from now on. The same applies to the negative examples as negative distance. The positive distance within the aspect-based embedding space is expected to be smaller than within a single embedding space due to the focus on one feature rather than all features simultaneously.

We could compare the baseline and the DSTM in an absolute manner. The baseline is compared with the average positive and negative distance. The advantage of this comparison is the clear effect of the DSTM on the perceived distance. If the model performs accordingly, we expect a smaller positive distance for the aspect-based embedding space. However, the disadvantage of this comparison is that the negative distance also goes down due to the aspect.

Therefore, we opt also for a relative comparison: the ratio of positive distance to the negative distance. A larger ratio portrays a better distinction between the positive and negative example. So, we propose to use a distance ratio:

$$\text{distance ratio} = \frac{|\text{average positive distance - average negative distance}|}{\text{average positive distance}}$$

6.5 Environmental setup

The experiments are implemented in PyTorch(version 2.1.0+cu121). For training, the learning rate is set to 0.0007. We use the ADAM optimizer with L2 regulization set to 0.01. Each model trains for 50 epochs without any early stopping. The model is saved when the validation loss decreases compared to the best model up to this point. During testing, we generate ten different support sets for each query. Only one support set does not offer enough evidence, as the support sets are generated randomly. Therefore, we generate multiple support sets.

Chapter 7

Results

This chapter assesses the performance of the proposed solution method. We expect to see a better distance ratio for the model architecture with the DSTM compared to the baseline without. Furthermore, we predict that the number of shared features between support set images and query should not influence the performance. The aspect should determine the distance between the query and the support set images. The two data sets are investigated separately in Section 7.1. Last, we expect to notice a difference in positive and negative distance for each aspect. The results for each data set can be found in Section 7.2

7.1 Experimental results and analysis of the distance ratio

7.1.1 Geometric shapes data

Figure 7.1 shows the three different support sets for the same query. The upper support set is referred to as the first support set. This support set has two shared features with the query. In

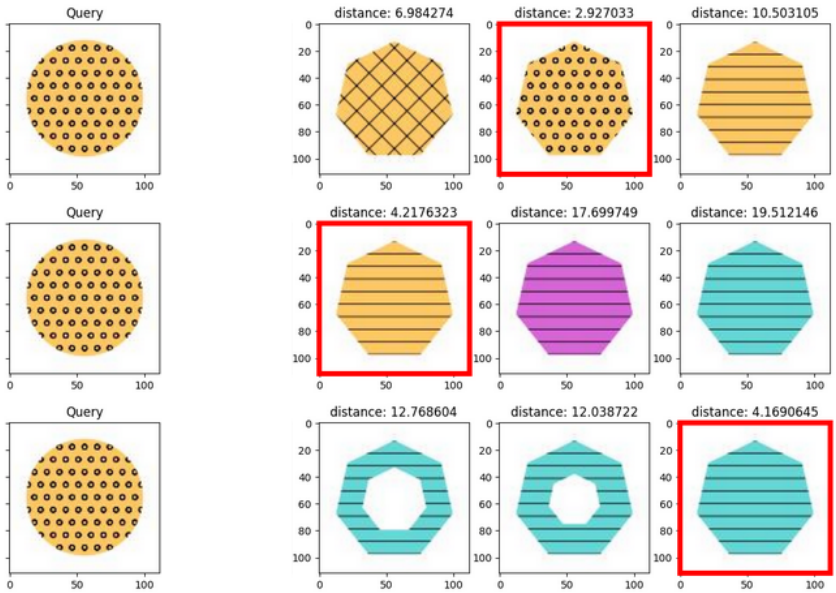


Figure 7.1: Example of the distance between the query and support set instances for the geometric shape data. Differentiating aspects leads to different distances between query and the same image.

	Data unique split			Query split		
Model	Positive distance	Negative distance	Distance Ratio	Positive distance	Negative distance	Distance Ratio
Baseline	-	-	-	40.5 ± 0.56	49.0 ± 0.32	0.19-0.24
Shallow + single layer DSTM	4.03 ± 0.07	8.96 ± 0.14	1.15-1.3	2.66 ± 0.06	6.67 ± 0.14	**1.4-1.63**
VGG + single layer DSTM	3.75 ± 0.08	9.76 ± 0.21	1.49-1.72	3.08 ± 0.07	7.61 ± 0.15	1.36-1.57
ResNet + single layer DSTM	3.25 ± 0.08	8.25 ± 0.15	1.43-1.65	2.71 ± 0.05	6.26 ± 0.11	1.23-1.4
Shallow + residual block DSTM	2.46 ± 0.06	5.90 ± 0.11	1.3-1.51	3.69 ± 0.06	6.13 ± 0.08	0.62-0.71
VGG + residual block DSTM	2.26 ± 0.06	6.19 ± 0.13	**1.6-1.85**	2.95 ± 0.07	7.84 ± 0.16	**1.54-1.79**
ResNet + residual block DSTM	2.54 ± 0.06	8.32 ± 0.19	**2.13-2.42**	3.24 ± 0.07	7.25 ± 0.12	1.15-1.32

Table 7.1: Geometric shapes data set: The average positive and negative distance of the first support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval.

this example, the shared features are color and thickness. In the first support set, we see that the second image matches the query on the aspect of the pattern. The third image of the first support set has a distance of 10.5 from the query image. However, we see the same image being much closer to the query in the middle or second support set, given the aspect of color. This second support set has only one shared feature in thickness. The distance to the query depends on the aspect of the support set aligning with the goal of this work. We see the same happening with the third image of the second support set transferred to the lower or third support set. In this support set, we do not have any shared features.

Table 7.1 shows the average positive distance and negative distance with a 95% interval of the first support set for each model configuration and the baseline. The distance ratio boundaries are calculated based on the 95% interval. We see that the distance ratio of the baseline is small. So, the baseline model cannot differentiate between positive and negative examples well. Each trained model with the DSTM has a better distance ratio. So, the model does a better job of differentiating between positive and negative examples. The VGG and ResNet representation model with the residual block DSTM perform best for the data unique split. The VGG representation model with the residual block DSTM also performs well in the case of a query split. However, the shallow representation model with a single-layer DSTM performs better than the ResNet representation model with the residual block DSTM. We only consider these model architectures in Sections 7.1.2 and 7.2. Due to the longer training times of the sprites data set, we cannot train everything. These models have the highest upper bound of the distance ratio and, therefore, the most potential.

Table 7.2 displays the average positive distance and negative distance with a 95% interval of the second support set for each model configuration and the baseline. The distance ratios are smaller than the first support set in Table 7.2. There is an increase in the positive distance and the negative distance. It indicates that the DSTM cannot completely filter out the differentiating features between the support set and the query. However, the best model architectures can still differentiate relatively well between negative and positive examples.

Table 7.3 shows the average positive distance and negative distance with a 95% interval of the

	Data unique split			Query split		
Model	Positive distance	Negative distance	Distance Ratio	Positive distance	Negative distance	Distance Ratio
Baseline	-	-	-	49.0 ± 0.46	54.9 ± 0.28	0.1-0.14
Shallow + single layer DSTM	5.46 ± 0.08	9.72 ± 0.13	0.73-0.84	3.96 ± 0.09	7.57 ± 0.14	**0.84-0.99**
VGG + single layer DSTM	5.19 ± 0.1	10.4 ± 0.2	0.93-1.09	4.32 ± 0.09	8.21 ± 0.15	0.82-0.97
ResNet + single layer DSTM	4.92 ± 0.11	9.30 ± 0.15	0.82-0.96	3.96 ± 0.07	7.15 ± 0.1	0.75-0.87
Shallow + residual block DSTM	3.82 ± 0.08	6.78 ± 0.11	0.71-0.84	5.51 ± 0.09	7.44 ± 0.08	0.31-0.39
VGG + residual block DSTM	3.23 ± 0.08	6.80 ± 0.12	**1.02-1.2**	4.03 ± 0.09	8.14 ± 0.15	**0.94-1.1**
ResNet + residual block DSTM	3.58 ± 0.08	8.85 ± 0.18	**1.36-1.58**	4.59 ± 0.09	8.06 ± 0.12	0.69-0.81

Table 7.2: Geometric shapes data set: The average positive and negative distance of the second support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval.

third support set for each model configuration and the baseline. The same happens with the third support set compared to the second support set as the second support set compared to the first support set. The positive distance and negative distance increase for every model. This increase results in a lower distance ratio. The data unique split does not disclose any results because of the reason mentioned in Section 6.1. We still see better differentiating between positive and negative examples with the DSTM compared to the baseline without.

	Data unique split			Query split		
Model	Positive distance	Negative distance	Distance Ratio	Positive distance	Negative distance	Distance Ratio
Baseline	-	-	-	55.0 ± 0.39	59.1 ± 0.25	0.06-0.09
Shallow + single layer DSTM	-	-	-	4.82 ± 0.09	8.35 ± 0.13	**0.67-0.79**
VGG + single layer DSTM	-	-	-	5.15 ± 0.1	8.71 ± 0.13	0.63-0.75
ResNet + single layer DSTM	-	-	-	4.96 ± 0.08	8.02 ± 0.1	0.57-0.66
Shallow + residual block DSTM	-	-	-	6.88 ± 0.1	8.55 ± 0.08	0.21-0.27
VGG + residual block DSTM	-	-	-	4.69 ± 0.09	8.40 ± 0.13	**0.73-0.85**
ResNet + residual block DSTM	-	-	-	5.38 ± 0.09	8.72 ± 0.11	0.57-0.67

Table 7.3: Geometric shapes data set: The average positive and negative distance of the third support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval.

7.1.2 Sprites data

Figure 7.2 has an example of the sprites data set with four support sets. We refer to the support sets the same way as previously with the geometric shape data set. However, the first support set has three shared features with the query. The second has two, the third has one, and the fourth support set has zero shared features. The support sets in the sprites data display the same behavior as the support sets of the geometric shape data set. The distance of the first image in the first support is significantly larger than the same image in the second support set. We see the same behavior in the remaining support sets. The identified aspects and commonalities influence the distance between query and support set instances. A different aspect leads to a different distance for the same image due to the aspect-based embedding space.

Table 7.4 displays the average positive distance and negative distance with a 95% interval of the first support set for each model configuration and the baseline. The distance ratio of the first support set displays the best results in the case of the query split. Each DSTM architecture with a query split performs better than every model with a data unique split. However, the data unique split still outperforms the baseline by a large margin. The distance ratios in Table 7.4 are also an improvement compared to the results of the geometric shapes data set in Table 7.1. The models do a better job of differentiating between positive and negative examples in the sprites data set. Tables 7.5, 7.6, and 7.7 show the results for the second, third, and fourth support sets. The data unique split is excluded in the results of the fourth support set. We see the same pattern as with the geometric shape data set. The distance ratio decreases as the matching features between the query and support set decrease. The model still has difficulty completely filtering out these feature values.

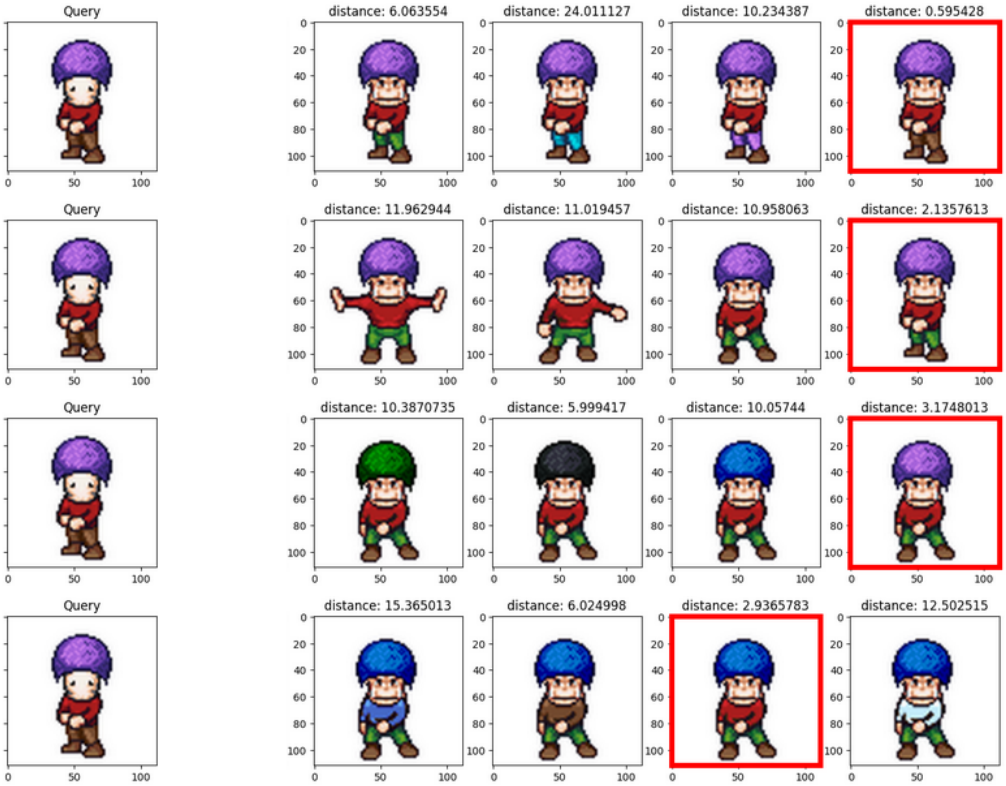


Figure 7.2: Example of the distance between the query and support set instances for the sprites data. Differentiating aspects leads to different distances between query and the same image.

	Data unique split			Query split		
Model	Positive distance	Negative distance	Distance Ratio	Positive distance	Negative distance	Distance Ratio
Baseline	-	-	-	14.2 ± 0.05	34.6 ± 0.1	1.42-1.45
Shallow + single layer DSTM	1.24 ± 0.01	7.73 ± 0.03	5.16-5.31	1.19 ± 0.01	10.8 ± 0.05	8.01-8.24
VGG + residual block DSTM	0.91 ± 0.01	6.67 ± 0.04	**6.2-6.37**	0.76 ± 0.01	10.1 ± 0.04	**12.0-12.46**
ResNet + residual block DSTM	1.03 ± 0.01	6.75 ± 0.05	5.39-5.6	1.05 ± 0.01	11.3 ± 0.04	9.58-9.86

Table 7.4: Sprites data set: The average positive and negative distance of the first support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval.

	Data unique split			Query split		
Model	Positive distance	Negative distance	Distance Ratio	Positive distance	Negative distance	Distance Ratio
Baseline	-	-	-	34.5 ± 0.17	46.9 ± 0.1	0.35-0.37
Shallow + single layer DSTM	2.40 ± 0.02	8.10 ± 0.03	**2.34-2.41**	2.78 ± 0.02	11.1 ± 0.04	2.94-3.03
VGG + residual block DSTM	2.47 ± 0.03	7.46 ± 0.04	1.97-2.09	2.03 ± 0.02	10.3 ± 0.04	**3.98-4.14**
ResNet + residual block DSTM	2.54 ± 0.02	7.51 ± 0.04	1.91-1.99	2.53 ± 0.02	11.4 ± 0.04	3.44-3.56

Table 7.5: Sprites data set: The average positive and negative distance of the second support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval.

	Data unique split			Query split		
Model	Positive distance	Negative distance	Distance Ratio	Positive distance	Negative distance	Distance Ratio
Baseline	-	-	-	46.9 ± 0.17	56.1 ± 0.1	0.19-0.2
Shallow + single layer DSTM	3.10 ± 0.02	8.43 ± 0.03	**1.69-1.74**	3.81 ± 0.02	11.3 ± 0.04	1.93-1.99
VGG + residual block DSTM	3.51 ± 0.04	8.18± 0.04	1.28-1.36	2.96 ± 0.03	10.4 ± 0.04	**2.49-2.59**
ResNet + residual block DSTM	3.54 ± 0.02	8.23 ± 0.04	1.29-1.35	3.53 ± 0.03	11.5 ± 0.04	2.23-2.3

Table 7.6: Sprites data set: The average positive and negative distance of the third support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval.

	Data unique split			Query split		
Model	Positive distance	Negative distance	Distance Ratio	Positive distance	Negative distance	Distance Ratio
Baseline	-	-	-	56.2 ± 0.17	63.5 ± 0.08	0.12-0.13
Shallow + single layer DSTM	-	-	-	4.59 ± 0.02	11.4 ± 0.04	1.46-1.51
VGG + residual block DSTM	-	-	-	3.75 ± 0.03	10.6 ± 0.04	**1.8-1.87**
ResNet + residual block DSTM	-	-	-	4.32 ± 0.03	11.6 ± 0.04	1.67-1.72

Table 7.7: Sprites data set: The average positive and negative distance of the fourth support set with a 95% interval. The distance ratio interval is calculated based on the 95% interval.

7.2 Experimental results and analysis of the aspects

7.2.1 Geometric shapes data

Table 7.8 shows the positive and negative distance of the first support set for each aspect. There is not much difference in the positive distance of each model architecture. Each aspect has a similar distance to the positive example. However, each aspect differs in the negative distance. The negative distance of the color aspect is significantly larger than the other two aspects in our data. Most likely, this happens due to the close similarity of the feature values. For example, Figure 7.3 displays a support set with the aspect of thickness. The second and third image are relatively close in thickness. Therefore, the distance between the second image and the query is significantly lower than the distance to the first image of the support set. These results support that the model successfully extracts the aspect as it explains the minor difference when the thickness is closer to the correct answer. The same applies to the aspect of pattern. The embeddings of certain patterns are close together, which results in a lower negative distance.

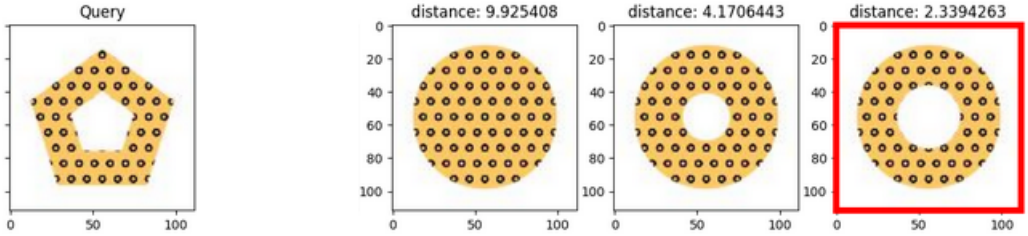


Figure 7.3: A support set with the aspect of thickness.

		Data unique split		Query split	
Model	Aspect	Positive distance	Negative distance	Positive distance	Negative distance
Baseline	Color	-	-	40.3 ± 0.99	51.3 ± 0.54
	Pattern	-	-	40.3 ± 0.97	46.4 ± 0.48
	Thickness	-	-	40.7 ± 0.97	49.3 ± 0.62
Shallow + single layer DSTM	Color	3.70 ± 0.12	10.5 ± 0.24	2.77 ± 0.13	8.18 ± 0.3
	Pattern	3.94 ± 0.11	7.09 ± 0.18	2.68 ± 0.11	6.34 ± 0.24
	Thickness	4.44 ± 0.13	9.28 ± 0.24	2.51 ± 0.08	5.48 ± 0.16
VGG + residual block DSTM	Color	2.35 ± 0.1	8.21 ± 0.26	3.04 ± 0.12	10.6 ± 0.31
	Pattern	2.36 ± 0.09	5.28 ± 0.19	2.86 ± 0.12	6.26 ± 0.26
	Thickness	2.08 ± 0.1	5.05 ± 0.16	2.92 ± 0.12	6.57 ± 0.19
ResNet + residual block DSTM	Color	2.54 ± 0.09	11.2 ± 0.4	2.96 ± 0.1	8.02 ± 0.23
	Pattern	2.65 ± 0.11	6.63 ± 0.23	3.11 ± 0.12	6.86 ± 0.18
	Thickness	2.44 ± 0.09	7.09 ± 0.25	3.64 ± 0.16	6.82 ± 0.19

Table 7.8: Geometric shapes data set: The average positive and negative distance of the first support set with a 95% interval for each aspect.

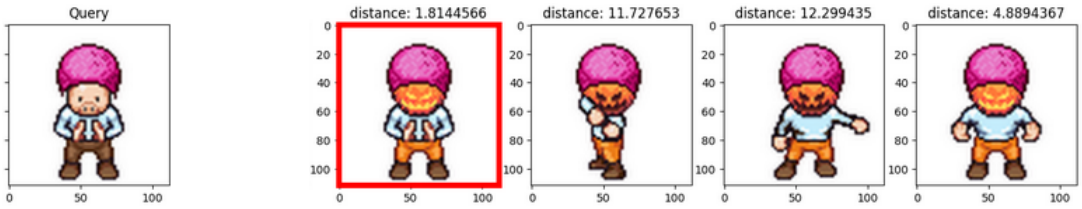


Figure 7.4: A support set with the aspect of stance.

7.2.2 Sprites data

The sprites data set has a smaller negative distance for the aspect of stance in Table 7.9. Stance has the similar characteristic that it shares similarity among feature values. The stance of the query in Figure 7.4 has the equal position of the feet as the fourth image. The only difference is within the position of the hands. This difference is reflected in the distance between the image and the query. The fourth image is closer than the second and third image, where almost everything differs in the stance. So, the sprite data shows the same behavior as the trained models of the geometric shapes data. This data set further reinforces the model correctly extracting the aspect from the support set.

7.3 Discussion

We provide a summary of the insights that are gained from the experimental results. A baseline model without the DSTM has difficulty discriminating between positive and negative examples. It shows a large positive and negative distance without a large relative distance between them. The

		Data unique split		Query split	
Model	Aspect	Positive distance	Negative distance	Positive distance	Negative distance
Baseline	Bottom	-	-	14.2 ± 0.1	26.0 ± 0.11
	Hair	-	-	14.3 ± 0.1	45.2 ± 0.21
	Stance	-	-	14.1 ± 0.1	34.9 ± 0.11
	Top	-	-	14.2 ± 0.1	32.6 ± 0.14
Shallow + single layer DSTM	Bottom	1.09 ± 0.02	7.23 ± 0.06	0.87 ± 0.01	13.3 ± 0.08
	Hair	1.25 ± 0.02	10.4 ± 0.07	1.39 ± 0.02	11.8 ± 0.11
	Stance	1.02 ± 0.02	7.01 ± 0.06	1.05 ± 0.02	7.03 ± 0.06
	Top	1.59 ± 0.02	6.23 ± 0.04	1.44 ± 0.01	11.1 ± 0.07
VGG + residual block DSTM	Bottom	0.77 ± 0.01	7.38 ± 0.09	0.44 ± 0.01	12.5 ± 0.11
	Hair	0.93 ± 0.01	8.07 ± 0.08	0.83 ± 0.02	8.77 ± 0.08
	Stance	0.86 ± 0.01	4.88 ± 0.05	0.76 ± 0.01	8.81 ± 0.05
	Top	1.08 ± 0.01	6.32 ± 0.09	1.00 ± 0.01	10.6 ± 0.07
ResNet + residual block DSTM	Bottom	0.89 ± 0.01	4.61 ± 0.06	0.56 ± 0.01	10.4 ± 0.07
	Hair	1.14 ± 0.02	11.6 ± 0.12	1.5 ± 0.02	13.3 ± 0.09
	Stance	0.93 ± 0.01	6.65 ± 0.06	0.92 ± 0.01	9.52 ± 0.06
	Top	1.19 ± 0.01	4.11 ± 0.04	1.27 ± 0.02	12.2 ± 0.09

Table 7.9: Sprites data set: The average positive and negative distance of the first support set with a 95% interval for each aspect.

introduction of the DSTM to the model architecture removes this problem. Each model architecture demonstrates a better differentiating between positive and negative examples in our support sets. Both data sets show increased performance in distance ratio compared to the baseline model. Furthermore, the amount of data affects the performance of aspect-based FSL. First, we notice the data unique split works better for smaller data sets. We have a lot fewer images in the case of the geometric shapes data set. The best model for this data set is a ResNet representation model with residual block DSTM. However, every model with query split outperforms the data unique split when we have much more data in the case of the sprites data set. The VGG representation model with the residual block DSTM outperforms every model architecture from the data unique split. Second, there is an overall performance improvement for the models trained on the sprites data set. In that setting, the distance ratio for each model is significantly larger than the setting of the geometric shapes data.

The analysis of the aspects illustrates that the model enriches the embeddings with the correct information. Figure 7.3 has a smaller distance to a thickness closer to the appropriate answer, while a completely different thickness is far away from the query. Figure 7.4 demonstrates the same behavior in the sprites data set. The stance with similarity to the correct stance is perceived as closer to the query than entirely different stances. So, the model identifies the aspects and highlights the relevant feature values.

Chapter 8

Conclusion and recommendations

This work addressed the intricate problem of label-based supervision approaches in Few-Shot learning. Existing Few-Shot learning approaches assume a unique mapping of all data points to a class label in a given set of labels. The class label of the query must precisely match one of the support set class labels. It does not reflect the human understanding of the data. We do not consider only the images separately. The context of the presented support set also conveys information to us. The different and shared information among support set instances matters in our decision-making. Therefore, we can still identify a similar image without any exact match by accounting for intra-conceptual relationships of the support set. In this work, we proposed to extend the Few-shot methods. The model moves away from learning a mapping to pre-defined labels. Instead, the model learns the differentiating feature (aspects) and common features (commonalities) of the support set in identifying a correct assignment of the query image.

An aspect-based embedding space was introduced towards this goal. An embedding space without the aspect leads to ambiguous answers. The distance/similarity between images is difficult to determine with ambiguous answers. Therefore, an embedding space conditioned on the aspect removes the ambiguity from the model. We proposed the Deep Set Traversal Module to incorporate the aspect-based embedding space into FSL. This model architecture learns a permutation-invariant representation of the support set. This representation highlights the feature values of the aspects in the embedding space while filtering out the values of the commonalities. In other words, the standard embedding space is transformed into an aspect-based embedding space by applying the permutation-invariant representation. It results in identifying the correct assignment based on only the aspect.

The typical split for Few-Shot learning does not apply to aspect-based Few-Shot learning. The removal of labels removes a data separation based on the labels. We offered two solutions to this problem. The first option was to split still based on classes. However, each image is classified as a separate class. Each image is identified as a unique set of feature values. Therefore, each image can be classified as a separate label. The second option was to have a split for the query images. There are no issues with seeing the same single image during testing and training, as the images are now part of a context. The same image conveys different information within different support sets.

The experimental results with the DPTM showed better differentiating between positive and negative examples when there is no exact match between query and support set instances. The baseline model without the DPTM had a significantly large distance to a positive and a negative example. However, they were relatively close together. It indicates that the baseline model was not able to make a lot of distinction between the correct and wrong answers. The proposed model architectures with the DPTM provided a reduced distance to the positive and negative examples. The distance ratio improved much compared to the baseline model. Nonetheless, we saw a decrease in performance when the support set had fewer feature values in common with

the query. This decrease indicates that the DPTM cannot completely filter out the commonalities from the embedding space. The evaluation of the individual aspects on the distances from the query to support set instances indicated a correct highlight of the aspect by the DPTM. The distance between a query and a negative example was lower when the aspect value of a negative example was similar to the query's value.

The drawn conclusions in this work stem from the data where the underlying data-generating process is known. This work mainly intended to illustrate the possibilities of changing the label-based supervision to an aspect-based approach in the domain of Few-shot learning. The idea has to be extended to cases with real-life data to build further upon this work. It requires a lot of human feedback as our perception of support sets can differ from person to person. We need multiple opinions to construct the support set with the correct answer. Furthermore, the model performs significantly better when switching from a small data set in the geometric shape data to the larger sprites data. This improvement suggests that aspect-based Few-Shot learning requires variety during training to correctly identify aspects and commonalities among the support set.

Additionally, the current model structure utilizes Convolutional Neural Networks and Deep Sets. Recently, tranformer-based models [5] for image analysis have advanced the state-of-the-art. The state-of-the-art representation learning architecture could further improve the performance in aspect-based Few-Shot learning. The DPTM utilizes Deep Sets for the permutation-invariant representation. However, other model structures also include the idea of permutation-invariance within the model architecture. For example, Graph Neural Networks [32] offer the same possibilities as Deep Sets. The proposed model architecture in this work is not the only possibility for the task of aspect-based Few-Shot learning. It was a design choice to test the viability of the method.

Bibliography

[1] Liberated pixel cup. http://lpc.opengameart.org/. Accessed: 2024-01-01. 26

[2] Universal lpc spritesheet character generator. https://github.com/sanderfrenken/Universal-LPC-Spritesheet-Character-Generator. Accessed: 2024-01-01. 26, 27

[3] Ossama Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Interspeech*, volume 2013, pages 1173–5, 2013. 7

[4] Qi Cai, Yingwei Pan, Ting Yao, Chenggang Yan, and Tao Mei. Memory matching networks for one-shot image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4080–4088, 2018. 12

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2, 52

[6] 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. 12

[7] Robert L Goldstone. Perceptual learning. *Annual review of psychology*, 49(1):585–612, 1998. 1

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. vii, 9

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. vii, 2, 11, 38

[10] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, pages 84–92. Springer, 2015. 16, 34

[11] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007. 27

[12] Pawan Kumar Jain, Khalil Ahmad, and Om P Ahuja. *Functional analysis*. New Age International, 1995. 13

[13] Rui Jin and Qiang Niu. Automatic fabric defect detection based on an improved yolov5. *Mathematical Problems in Engineering*, 2021:1–13, 2021. vii, 10

[14] PC Klink, Richard Jack Anton van Wezel, and Raymond van Ee. United we sense, divided we fail: context-driven perception of ambiguous visual stimuli. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1591):932–941, 2012. vii, 21

[15] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, page 0. Lille, 2015. 12, 16

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 7

[17] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 2

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. vii, 2, 7, 8

[19] Hongyang Li, David Eigen, Samuel Dodge, Matthew Zeiler, and Xiaogang Wang. Finding task-relevant features for few-shot learning by category traversal. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1–10, 2019. vii, 18

[20] Xiaoxu Li, Xiaochen Yang, Zhanyu Ma, and Jing-Hao Xue. Deep metric learning for few-shot image classification: A review of recent developments. *Pattern Recognition*, page 109381, 2023. vii, 16, 17

[21] Yingzhen Li and Stephan Mandt. Disentangled sequential autoencoder. In *International Conference on Machine Learning*, 2018. 25

[22] Archit Parnami and Minwoo Lee. Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291*, 2022. vii, 2, 12, 15, 16

[23] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018. 12

[24] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016. 12

[25] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 34

[26] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017. 10

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2, 38

[28] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017. 17

[29] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. *Advances in neural information processing systems*, 29, 2016. 34

[30] Juan Luis Suárez, Salvador García, and Francisco Herrera. A tutorial on distance metric learning: Mathematical foundations, algorithms, experimental analysis, prospects and challenges. *Neurocomputing*, 425:300–322, 2021. 12

[31] 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. 12, 17

[32] Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, 2023. 52

[33] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016. 12

[34] Lu Yin, Vlado Menkovski, and Mykola Pechenizkiy. Knowledge elicitation using deep metric learning and psychometric testing. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part II*, pages 154–169. Springer, 2021. 27

[35] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017. 14