공학석사 학위논문

특징 공간에서의 데이터 증강을 이용한 퓨샷 러닝

Few-Shot Learning via Data Augmentation in Feature Space

2020년 2월

서울대학교 대학원

기계항공공학부

정 성 재

ABSTRACT

Few-Shot Learning via Data Augmentation in Feature Space

by

Seongjae Jeong

Department of Mechanical and Aerospace Engineering
Seoul National University

When applying deep learning to supervised learning problems, a large amount of labeled data is usually required to achieve good generalization performance. Using few-shot learning, however, we can train deep learning models that are capable of recognizing unseen classes even with limited labeled data. As one of the representative categories of few-shot learning, hallucination-based methods have been utilized. To address data deficiency, methods in this category learn an image generator network and make use of the generator to hallucinate new supplementary

data. The downside of the hallucination-based approach is that training the generator is itself difficult and requires a great deal of time and memory. This thesis proposes a data augmentation method for a few-shot learning task using various types of interpolation and extrapolation in feature space. Our approach does not need any trained generator for data augmentation and can be applied to any few-shot learning methods which use feature extracting networks. We conduct few-shot classification experiments using benchmark datasets as well as a visual inspection dataset. Experiments demonstrate that classification accuracy improves as the amount of data increases by the proposed method. Besides, using few-shot learning scheme and our data augmentation method at manufacturing fields that suffer from lack of defective data is expected to help reduce the labeling costs by filtering out the data roughly.

Keywords: Few-shot learning, data augmentation, feature space, interpolation and extrapolation, visual inspection.

Student Number: 2016-20714

Contents

List of Tables

List of Figures

1

Introduction

Visual recognition tasks such as image classification, object detection, and semantic segmentation have well been addressed by virtue of the development of deep learning models. However, for deep learning models to perform well in computer vision tasks, a large amount of labeled data is required and the visual diversity of the data must also be ensured. For example, we need thousands of images from each class during training even with a model pre-trained with a large-scale dataset. In practice, sufficiently collecting supervised data is burdensome due to the human annotation cost or the rarity of data. This is why using deep learning often leads to difficulties when trying to solve vision-related problems.

To cope with data deficiency issues, *few-shot learning*, or learning from a limited amount of labeled data, is receiving increased attention in the literature. Since humans are able to recognize new object categories with few instances, few-shot learning can also be an important clue to determine whether artificial intelligence carries the same cognitive capabilities as humans.

1.1 Previous Research

The visual intelligence of humans is based on their accumulated knowledge and experiences. In a similar way, few-shot learning assumes that deep learning models grasp new concepts with limited supervised information after they reach an adequate level of knowledge. In detail, deep learning models are first trained using sufficient data from base classes, and then validated to determine whether they are capable of classifying data from novel classes after seeing only a few labeled examples.

A meta-learning framework has recently been considered one of the most potential approaches to few-shot learning. Using this framework, transferrable knowledge can be obtained from multiple auxiliary tasks and propagated to the target few-shot problem to prevent overfitting. Recent representative few-shot learning algorithms based on the meta-learning framework can be categorized into two main methods: metric learning-based and gradient-based methods. Metric learning-based methods are inspired by the idea that if a model is able to determine how similar two images are, the model can classify an unlabeled image based on the similarity to a few labeled images [1]. The similarity between two images is calculated by the combination of learnable feature extractors and distance metrics. The feature extractor is mostly a convolutional neural network (CNN), as is the case with many of the recent advances in computer vision regime. Examples of distance metrics vary widely, including cosine similarity [2], Euclidean distance [3], relation module based on CNN [4], and ridge regression [5].

Gradient-based methods address the few-shot learning problem by learning good initialization of model parameters and optimizing a model with rapid adaptation. In [6], the proposed approach can learn parameter initialization for a model to be

capable of classifying unseen images with a few steps of standard stochastic gradient descent (SGD). Several variations of this method are also suggested, showing improvements in recent years [7, 8, 9, 10, 11]. Meanwhile, in [12], LSTM-based meta learner is utilized as an optimizer instead of standard SGD.

Besides the meta-learning framework, in a recent study [13], a simple classification method based on a transfer learning framework is introduced, which fine-tunes a cosine classifier on top of a pre-trained deeper backbone network. Although the cosine similarity-based classification in a few-shot learning setting is well addressed in other recent studies [14, 15], the authors of [13] demonstrate that in few-shot scenarios their method performs pretty well and even outperforms representative meta-learning-based methods when combined with deep feature extractor networks.

Hallucination-based methods address data deficiency directly by generating supplementary data. Methods in this category focus on learning a generative model using abundant data from base classes and use the trained generator to synthesize new data points for augmenting the limited amount of data. Since training a generator can be regarded as an independent task, these methods can be integrated into other few-shot learning methods to further improve performance. In [16], when given two data instances from the same class, the relative linear offset between them in feature space is assumed to represent a plausible transformation. The offset information is transferred to an unseen data point in feature space by completing the transformation analogy, leading to obtaining a hallucinated data. Another type of methods does not transfer information explicitly but combines the generator directly with a meta-learning algorithm [17].

1.2 Contributions of This Thesis

Simple but effective data augmentation method As already mentioned, existing hallucination-based methods usually focus on training of the generative model to produce the supplementary data. However, training a generator is itself difficult due to the problem of mode collapse, especially when dealing with general real-world images [18]. Furthermore, using a generator increases the time and memory costs. The memory issue makes it difficult to use a deep and powerful network for a feature extractor and also imposes restrictions on the batch size of the input data.

This thesis proposes a simple but effective data augmentation method that is exploited in feature space to improve the performance of few-shot learning instead of using generative models. We apply data augmentation in four different ways: linear interpolation between two data, triplet interpolation among three data, and adding extrapolation to each case.

Using our approach, we do not need to train the additional generative model, thus avoiding the problems of existing hallucination-based methods. Our approach is also applicable to all other few-shot learning schemes that use a feature extractor. The experimental results show that various kinds of interpolation and extrapolation in feature space can synthesize additional data meaningful enough to help improve the performance of few-shot classification when combined with the cosine classifier.

Few-shot learning for visual inspection problems This study is also motivated by the practical difficulties of applying deep learning to visual inspection problems. Existing vision inspection method has mostly been rule-based. Human experts first define detailed characteristics and thresholds for defects, and then target images are inspected according to the settings. However, when defects are complex and subjective, the rule-based inspection easily becomes difficult and inefficient. Training a deep learning network with large amounts of data is a promising alternative. If we use the deep learning approach, rules do not have to be specified explicitly, and even complex, irregular defects can also be identified with ease.

In the field of manufacturing, however, while non-defective data is abundant, defective data is often highly limited. Furthermore, the training data must all be labeled by human experts, which is laborious and time-consuming. Since the data deficiency issue makes it difficult to apply the deep learning method to visual inspection problems, we adopt the few-shot learning approach to tackle the issues.

To simulate these more practical problems, we make use of a comprehensive real-world dataset called MVTec-AD [19]. We then perform a task to distinguish defective and non-defective data under a few-shot setting. Assuming a pure few-shot classification task, we experiment with a scenario where the number of labeled data given to us is the same for the OK and NG classes. In another scenario, the number of labeled data in the OK class is much larger than the NG class, assuming a more realistic but less severe visual inspection situation. We believe that this is the first work that shows few-shot classification results in realistic visual inspection scenarios as well as popular benchmarks.

1.3 Organization

The rest of this thesis is organized as follows. In Chapter 2, we review the concept of few-shot learning and introduce the N-way K-shot problem. Two major frameworks for few-shot learning that have emerged recently are also presented. We then briefly review the traditional data augmentation methods applied at the image level. Chapter 3 covers the key methodologies used in this thesis. We introduce a few-shot learning approach based on transfer learning framework and cosine similarity, followed by the way interpolation and extrapolation can be exploited in feature space for data augmentation. Chapter 4 contains the details, results, and analysis of our experiments. It is divided into two parts: benchmark experiments and visual inspection experiments. We finally summarize the contributions and results of this paper in Chapter 5 and outline the hint about future works.

2

Preliminaries

This chapter covers the basics of few-shot learning and transfer learning, which are the basis for the methods and experiments described in the following chapters. To prevent confusion with the concept of data augmentation in feature space proposed in this thesis, we also introduce representative data augmentation methods based on image manipulations.

2.1 Few-Shot Learning

2.1.1 Problem Definition of Few-Shot Learning

Since the few-shot learning problem is a special case of general machine learning problems, it is good to first look at the definition of machine learning.

Definition 2.1.1. (*Machine learning* [20]) A computer program is said to learn from experience E with respect to some classes of task T and performance measure P if its performance can improve with E on T measured by P.

For instance, a standard image classification problem can be described using this definition. In this case, T is an image classification task and P is the accuracy based on the specified metric. E is the training procedure using large amounts of labeled data.

Few-shot learning is part of machine learning, which aims to achieve good learning performance. The only difference is that the amount of data given is very limited. Hence few-shot learning can be defined by adopting Mitchell's definition of machine learning.

Definition 2.1.2. (*Few-Shot Learning* [21]) Few-shot learning is a type of machine learning problems (specified by E, T and P) where E contains little supervised information for the target T.

Here is the image classification problem revisited, where T is the image classification task and P is the classification accuracy. What is different from the standard classification problem is there are not many labeled images in E. In practice, E not only means a limited amount of labeled data, but also all types of prior knowledge, such as images of different classes or pre-trained models. If P improves with E, we can say the few-shot learning successfully applied to the given T and expect the model to perform well in the many-shot setting.

The word "little" in this definition is not scientifically clear, but this is because "few-shot" itself contains the unclear word "few". If we set the exact number of data given per class, we may obtain a clearer name and definition of the problem (e.g., 1-shot learning when only one labeled data is given per class). In practice, most recent few-shot learning studies report the results of 1-shot and 5-shot scenarios. There is also a problem called zero-shot learning, but it is beyond the scope of this thesis.

2.1.2 Classification Problem

Unless otherwise noted, few-shot learning problems mostly mean few-shot classification problems. As the relationship between few-shot learning and machine learning, the few-shot classification problem is a special case where there is a lack of data in standard classification problems.

The goal of the classification problem is predicting the label $y \in \mathcal{C} = \{c_1, ..., c_N\}$ of a given d-dimensional input vector $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$, where $\mathcal{X}$ and $\mathcal{C}$ are a finite set of data and classes respectively. In the supervised setting, the classification problem can be viewed as estimating the function $f : \mathcal{X} \to \mathcal{C}$ using a given training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M}$ where M is the number of training data. If the function f is well trained, then we can predict the correct label y for the unseen data $\mathbf{x}$.

In most cases, the function f is just a classifier that takes the extracted features as input and predicts the labels of them. The feature extractor f_θ parametrized by parameters θ exists separately, the role of which is to transform raw data into features aiming for the features to have as much information about raw data as possible. When it comes to image classification problems, a set of 2-D image data $\mathcal{X}_{raw} \subseteq \mathbb{R}^{w \times h}$ are given as raw data and the CNN architecture followed by fully connected layers is functioning as a feature extractor f_θ if deep learning framework is used. After an image $\mathbf{x}_{raw} \in \mathcal{X}_{raw}$ passes through f_θ, it becomes a d-dimensional vector $\mathbf{x}$. This vector $\mathbf{x}$ is used as an input to the classifier f and this is where the classification process described in the previous paragraph takes place.

2.1.3 Few-Shot Classification Problem: N-way K-shot Problem

In short, the case where M is extremely small is the few-shot classification problem, which is commonly formulated as a N-way K-shot problem. It is a supervised

classification problem with the support set $\mathcal{D}_{support} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M_s}$ that is used for training and the query set $\mathcal{D}_{query} = \{\mathbf{x}_j\}_{j=1}^{M_q}$ that is used for evaluation. If K labeled examples for each of N classes exist in the support set, this problem is called N-way K-shot with $M_s = KN$. Although N can be any number, most experimentation is done in the 5-way or 20-way setting. K should be very small in the few-shot scenario, and as mentioned earlier, it is usually assumed to be 1 or 5. In Figure 2.1, we illustrate the 5-way 5-shot classification problem as an example.

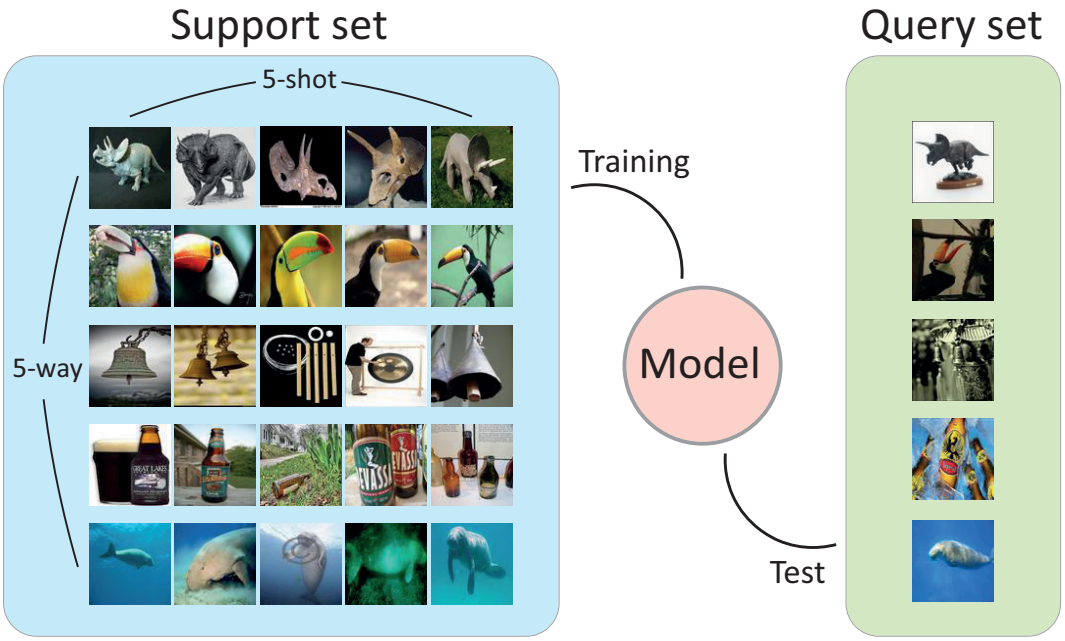


Figure 2.1: 5-way 5-shot classification example.

2.2 Two Main Frameworks of Few-Shot Learning

Strictly speaking, in few-shot learning problems, the learning model does not learn with just one or five labeled examples per class in general. The model is pre-trained with copious data from the source classes that are disjoint with the target classes. After that, the model is used to solve the few-shot learning problem with limited labeled data of target classes. This learning process is the same as that of humans that use their accumulated knowledge when solving new problems or learning new concepts faster and better.

In the few-shot learning literature, there are two main frameworks to allow the learning model to accumulate knowledge: *meta-learning* and *transfer learning*. Both frameworks aim to convey knowledge from the source tasks to the target task. To distinguish the source tasks from the target task, we should first split the dataset into *base classes* and *novel classes*. The few-shot classification setting can be described by a whole dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M}$, where $\mathbf{x} \in \mathbb{R}^d$, $y \in \mathcal{C}$, and $\mathcal{C}$ is a finite set of all given classes. We have labeled data abundant enough to be used for the source tasks, and the abundant data are in the set of base classes $\mathcal{C}_b \subset \mathcal{C}$. Meanwhile, the amount of labeled data for the target tasks is limited, and the data are in the set of novel classes $\mathcal{C}_n \subset \mathcal{C}$. If validation process is required, *validation classes* also need to be assigned separately. These subsets must be completely disjoint with each other.

The difference between the two frameworks is how knowledge is accumulated. We describe each of these in detail below.

2.2.1 Meta-Learning for Few-Shot Learning

The method proposed in [2], called *set-to-set* learning or *episodic* learning, plays a key role in this framework. The main idea is exploiting the N-way K-shot problem repeatedly in the training stage, mimicking the tasks of the test stage. In each episode, we have N classes randomly selected from $\mathcal{C}_b$. From the selected classes, the support set $\mathcal{D}_{support}$ for training is given with K labeled examples per class. The query set $\mathcal{D}_{query}$ is used to determine if the model has acquired knowledge from the support set correctly. The model is trained with tens of thousands of episodes. Repeating the N-way K-shot problem in multiple episodes, we expect the model to learn how to learn a new concept from limited amounts of labeled data. This is why we call this approach "meta-learning".

2.2.2 Transfer learning for Few-Shot Learning

Transfer learning is a traditional way to use when we do not have sufficient labeled data by transferring knowledge from source tasks to a target task. In the Few-shot learning literature, the transfer learning approach has been revisited in recent years, and it is demonstrated that the performance is competitive compared to the conventional meta-learning methods.

The whole learning process consists of two learning phases. The first phase can be thought of as a pre-training stage, where a feature extractor and a base classifier are trained using data in base classes $\mathcal{D}_b = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M_b}$ with $y_i \in \mathcal{C}_b \subset \mathcal{C}$ aiming to perform a standard classification task. During this stage, the feature extractor adjusts itself to be capable of extracting robust features that generalize well even with very few data. In the second phase, the pre-trained feature extractor is fixed and a new classifier is fine-tuned with the examples in the support set from

novel classes $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M_n}$ with $y_i \in \mathcal{C}_n \subset \mathcal{C}$. The fine-tuned classifier then performs the N-way K-shot classification task for the unlabeled examples of the query set. A few-shot learning process based on the transfer learning framework is illustrated in Figure 2.2.

2.3 Data Augmentation by Image Manipulations

Data augmentation is a simple but effective method to avoid overfitting. In this section, we cover several well-known data augmentation methods in computer vision literature with reference to [22]. This kind of augmentation is applied at the image level, so it should be distinguished from the data augmentation in feature space.

Flipping is one of the easiest data augmentation methods. It is usually applied horizontally, not vertically. The important thing is whether the flipped image belongs to the same class as the original image. This makes it difficult to apply the method to character datasets such as MNIST and Omniglot.

Cropping means to crop a portion of the original image. There are two main types of cropping: center cropping and random cropping. Center cropping is a process that crops a central patch of an image, which can be useful for images with different height and width. Random cropping is a method that randomly crops a part of the original image to the desired size. The cropped image might not retain the label depending on the desired crop size.

Color jittering denotes randomly manipulating brightness, contrast, and saturation of an input image. It can also be thought of as making random changes to the RGB values of an image. Even though the label of images rarely changes after applying color jittering, sometimes color is important in classifying images.

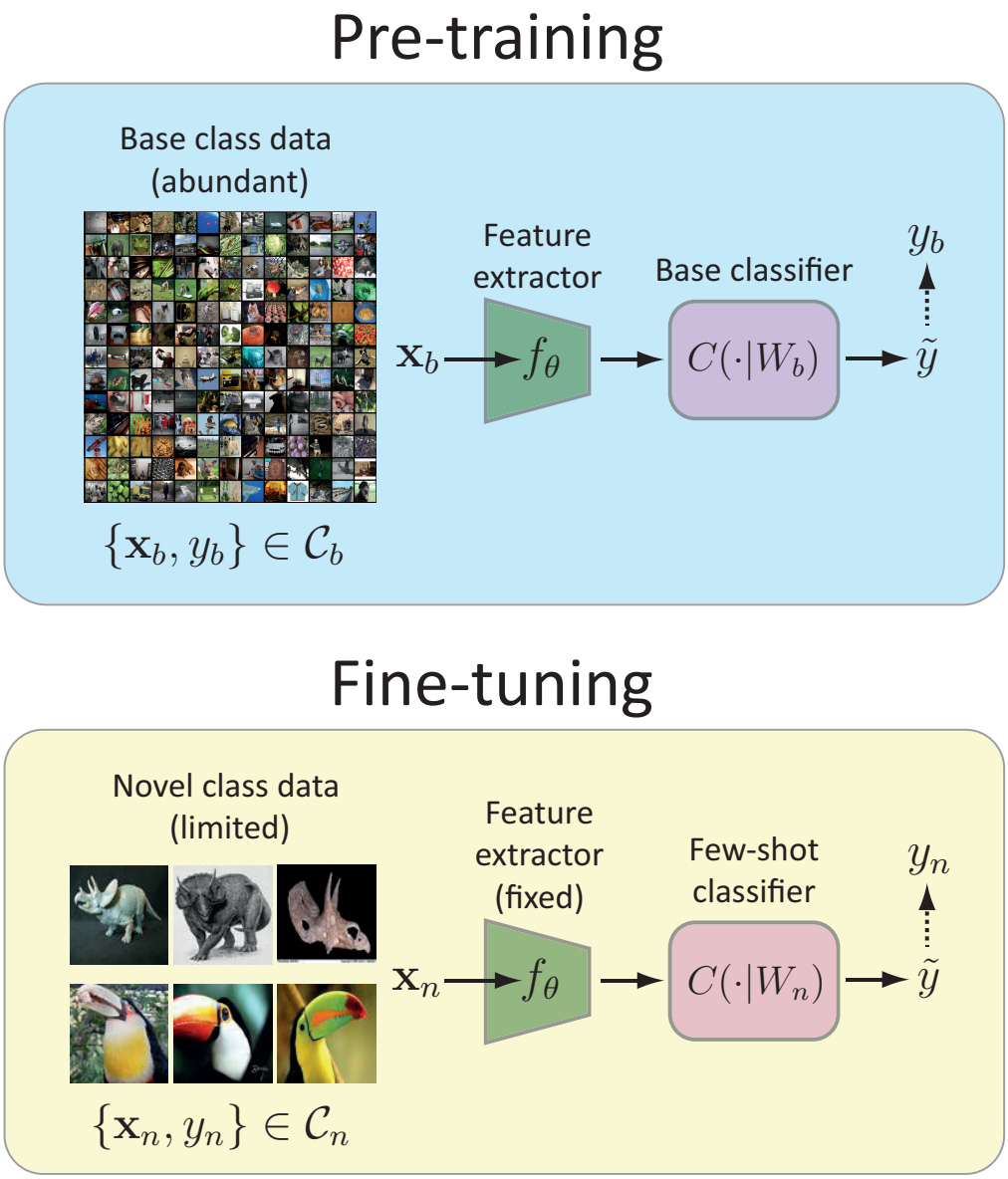


Figure 2.2: Flowchart for few-shot learning based on transfer learning framework.

The red color, for example, is the key to distinguishing blood from other liquids. In such cases, it is necessary to apply the color jittering method carefully.

3

Few-Shot Classification via Data Augmentation in Feature Space

In this chapter, we describe in more detail how the transfer learning framework is exploited in the few-shot classification setting based on cosine similarity. We also subsequently address how a data augmentation method using interpolation and extrapolation in feature space can be integrated into the few-shot classification problem.

3.1 Few-Shot Classification Based on Cosine Similarity

In this section, we sketch the details of a few-shot classification method using a cosine classifier. This concept of cosine similarity-based classification has recently been studied for the purpose of using in the few-shot classification setting. Particularly, in [13], the authors show that learning a cosine classifier achieves competitive performance with the state-of-the-art meta-learning methods by using deeper

networks as feature extractors.

3.1.1 Details of Few-Shot Learning Based on Transfer Learning

In the **training phase**, we train a feature extractor f_θ and the classifier $C(\cdot|\mathbf{W}_b)$ parametrized by the network parameters θ and the weight matrix $\mathbf{W}_b \in \mathbb{R}^{d\times c}$ respectively, where d is the dimension of the extracted feature and c is the number of output classes. Using the training data in the base classes $\mathcal{D}_b = \{(\mathbf{x}_i, y_i)\}_{i=1}^{M_b}$, the training procedure is performed in a way that minimizes a standard cross-entropy loss L_{pred}. The classifier $C(\cdot|\mathbf{W}_b)$ is typically composed of a linear layer $\mathbf{W}_b^\top f_\theta$ followed by a softmax function σ. The softmax function is defined as follows:

$$\sigma(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{d}\exp(x_j)}, \tag{3.1.1}$$

where $i = 1, 2, ..., d$ and $\mathbf{x} = (x_1, x_2, ..., x_d) \in \mathbb{R}^d$. After a vector of d real numbers passes through the softmax function, each element is mapped to be in the interval of $(0, 1)$ and the elements add up to 1, which is to say, the input vector is normalized into a probability distribution of d probabilities proportional to the exponentials of the components in the input vector. In the classification setting, the softmax function maps the output of a neural network to a probability distribution over output classes.

In the **fine-tuning phase**, the network parameter θ is fixed in the pre-trained feature extractor f_θ. With the fixed feature extractor, we can train a new classifier $C(\cdot|\mathbf{W}_n)$ which is parametrized by the weight matrix $\mathbf{W}_n$. The new classifier adapts itself to recognize novel classes by fine-tuning with only a small amount of labeled data in the novel classes. The N-way K-shot problem emerges in the fine-tuning phase. The support set $\mathcal{D}_{support} = \{(\mathbf{x}_{ij}, c_i), i = 1, 2, ..., N \quad \text{and} \quad j = 1, 2, ..., K\}$ is the training data and the query set $\mathcal{D}_{query} = \{\mathbf{x}_i\}_{i=1}^{M_q}$ is the test data.

3.1.2 Cosine Classifier

In deep metric learning and few-shot classification problems, reducing intra-class variations of features is known to be highly important [14, 23]. In [13], the classifier design using cosine similarity is proposed to achieve the goal of reducing intra-class variations of features. Most of the contents in this section refer to [13].

Before describing the cosine classifier, let us recall what the cosine similarity is. The cosine similarity between two non-zero d-dimensional vectors $\mathbf{a}$ and $\mathbf{b}$ of an inner product space is defined as follows:

$$s = \cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = \frac{\sum_{i=1}^{d} a_i b_i}{\sqrt{\sum_{i=1}^{d} a_i^2} \sqrt{\sum_{i=1}^{d} b_i^2}}, \tag{3.1.2}$$

where a_i and b_i are components of a and b respectively. The similarity value ranges from -1 to 1, where -1 means two vectors are exactly opposite to each other and 1 means two vectors are the same. If the value is zero, two vectors are orthogonal.

Except for the design of the classifier, the training process is the same as what is described in Section 3.1.1 when a cosine classifier is used. Although weight matrices $\mathbf{W}_b, \mathbf{W}_n \in \mathbb{R}^{d \times c}$ still exist in the training phase and fine-tuning phase respectively, the matrices are used to calculate cosine similarity scores between weight vectors and an input feature. Viewing a matrix as a collection of column vectors, the weight matrix can be rewritten as $[\mathbf{w}_1, ..., \mathbf{w}_c]$, where a d-dimensional weight vector $\mathbf{w}_i \in \mathbb{R}^d$ exists for each class. For a given input feature $f_\theta(\mathbf{x})$, the cosine similarity scores $[s_1, s_2, ..., s_c]$ can be obtained for all classes, where s_i is computed using the cosine similarity as follows:

$$s_i = \frac{f_\theta(\mathbf{x})^\top \mathbf{w}_i}{\|f_\theta(\mathbf{x})\| \, \|\mathbf{w}_i\|}. \tag{3.1.3}$$

The softmax function subsequently takes the similarity scores as input so that these scores are normalized and functioning like a probability distribution. The

classifier can then predict the label of input by selecting the class with the highest probability value. Learning the classifier is equivalent to learning weight vectors $[\mathbf{w}_1, ..., \mathbf{w}_c]$, each of which represents the corresponding class. Hence, in a similar manner as [2, 3] propose, we can think of the weight vectors as prototypes for each class and the cosine similarity as a measure of how close the learned prototypes are to the input feature.

3.2 Data Augmentation in Feature Space

We hypothesize that high-quality interpolation or extrapolation between data in feature space can produce additional data meaningful enough to be incorporated into the training set. However, intuitively, pixel-wise interpolation or extrapolation between images does not produce realistic images. In addition, high-level representations usually live in a space with much lower dimensions where interpolation and extrapolation of features would more likely to navigate through relevant regions.

In this section, we outline some of the recent studies that have reported noticeable results about regularization using only simple interpolation methods between features. Furthermore, we also describe how we use various types of interpolation and extrapolation in this thesis to improve few-shot classification performance.

3.2.1 Previous Research on Interpolation in Feature Space

According to [24], "high-quality" interpolation should have two main properties. First, data points created by interpolation look realistic enough to be indistinguishable from real data. Secondly, a natural shift occurs between the intermediate data, resulting in a semantically smooth transition between the two original data.

To achieve these two properties, the authors of [24] propose Adversarially Constrained Autoencoder Interpolation (ACAI) algorithm. The main idea is to train a critic network to identify whether the input image is from interpolation or not, while the autoencoder is trained to trick the critic network and reconstruct the input image at the same time. Under this adversarial regularization strategy, the autoencoder is expected to generate more realistic interpolated outputs.

Another line of research on interpolation is Manifold Mixup [25], which leverages linear interpolations in deep hidden layers as additional training examples. Using this algorithm, we can obtain smoother decision boundaries at multiple layers of neural networks and class-representations which have fewer directions of variance. Experimental results demonstrate that neural networks trained with the algorithm show better generalization performance in terms of error and log-likelihood.

3.2.2 Various Types of Interpolation in Feature Space

In this thesis, we exploit various types of interpolation in feature space to augment data in the few-shot classification setting. To be more specific, the data augmentation approach is applied to the support set in the fine-tuning stage where the labeled data are limited. As a result, the original N-way K-shot problem can be transformed into the N-way $K^{'}$-shot problem with $K^{'} > K$.

Linear interpolation When using ACAI algorithm, it appears to be hard to train the neural networks with realistic datasets like CUB, *mini*ImageNet, and MVTec-AD. More precisely, it is difficult to properly train both the autoencoder and the critic network. Often only one of them converges well while the other does not. This is a common problem in using adversarial strategy. The result of the Manifold Mixup is, however, quite encouraging in that simple linear interpolation

can make sense without any adversarial processes or additional neural networks. Moreover, even intermediate data generated by simple linear interpolation without any critic network can represent sufficiently realistic data when close to the endpoints (i.e., mixing coefficient is close to zero). We thus conjecture that simple linear interpolation between data in feature space can potentially improve the performance of few-shot learning.

Let f_θ be a trained feature extractor. For two raw data $\mathbf{z}_1$ and $\mathbf{z}_2$, we first get features $\mathbf{x}_1 = f_\theta(\mathbf{z}_1)$ and $\mathbf{x}_2 = f_\theta(\mathbf{z}_2)$ from the raw data. The interpolated data point $\mathbf{x}_\alpha$ in feature space can then be calculated as follows:

$$\mathbf{x}_\alpha = (1-\alpha)f_\theta(\mathbf{z}_1) + \alpha f_\theta(\mathbf{z}_2) = (1-\alpha)\mathbf{x}_1 + \alpha\mathbf{x}_2, \tag{3.2.4}$$

where the mixing coefficient α is sampled from the uniform distribution $U(0, k_s)$. The hyperparameter k_s is a spreading factor of interpolation, which means how far $\mathbf{x}_\alpha$ exists from $\mathbf{x}_1$ in the feature space. k_s must be less than 0.5 to ensure that $\mathbf{x}_\alpha$ belongs to the same class as $\mathbf{x}_1$ while keeping $\mathbf{x}_\alpha$ close to the data manifold. At the same time, to enforce some extent of spreading, k_s should not be too close to zero. The detailed process of using linear interpolation for data augmentation is described in Algorithm 1.

Triplet interpolation using De Casteljau's algorithm Data created by linear interpolation of two data exists only on the straight line connecting the endpoint data. It may not be a problem if the data is abundant, but the amount of data given is so small that linear interpolation alone cannot properly cover feature space in the few-shot setting. We thus are motivated to use triplet interpolation for data augmentation, leading to exploring inside the triangle that consists of the three data as vertices.

Let f_θ be a trained feature extractor again. For three raw data $\mathbf{z}_1$, $\mathbf{z}_2$, and $\mathbf{z}_3$, we obtain features $\mathbf{x}_1 = f_\theta(\mathbf{z}_1)$, $\mathbf{x}_2 = f_\theta(\mathbf{z}_2)$, and $\mathbf{x}_3 = f_\theta(\mathbf{z}_3)$ respectively through f_θ. The intermediate data from triplet interpolation $\mathbf{x}_\alpha$ in feature space can be obtained as follows:

$$\mathbf{x}_{12} = (1-\alpha_1)\mathbf{x}_1 + \alpha_1\mathbf{x}_2, \tag{3.2.5}$$

$$\mathbf{x}_{23} = (1-\alpha_2)\mathbf{x}_2 + \alpha_2\mathbf{x}_3, \tag{3.2.6}$$

$$\mathbf{x}_\alpha = (1-\alpha_3)\mathbf{x}_{12} + \alpha_3\mathbf{x}_{23}. \tag{3.2.7}$$

where the mixing coefficients α_1 and α_3 are sampled from the uniform distribution $U(0, k_s)$, while α_2 is from $U(0, 1)$. In triplet interpolation, k_s should be less than 0.25 for the same reason of labeling as in linear interpolation. The sampling range of α_2 is specified to make the most of the space between x_2 and x_3. This method of calculating x_{alpha} in triplet interpolation is a variation of De Casteljau's algorithm [26], which is a recursive algorithm to obtain a Bézier curve. Although augmentation can be achieved using more than three data (e.g., quadruplet interpolation), in this thesis only triplet interpolation is applied because of the computational cost. We describe the process of triplet interpolation in Algorithm 2.

Adding extrapolated data The limitation of the two methods described above is that the augmented data exist within the convex hull of the given data. We can overcome this limitation by considering extrapolation in feature space for data augmentation. It is not difficult to obtain extrapolated data with a little modification of the previous two interpolation algorithms: changing the sampling range of the mixing coefficients. We use the uniform distribution $U(-k_s, k_s)$ for sampling α in the linear case and α_3 in the triplet case.

Algorithm 1 Data augmentation via linear interpolation in N-way K-shot setting

initialization Support set data in N-way K-shot setting $\mathcal{D}_{support} = \{(\mathbf{x}_{11}, c_1), (\mathbf{x}_{12}, c_1), ..., (\mathbf{x}_{NK}, c_N)\}$, a repeat count of augmentation n_{aug}, and a spreading factor k_s.

$\mathcal{D}_{aug} = \mathcal{D}_{support}$

for $k = 1$ **to** n_{aug} **do**

  Sample a set of mixing coefficient $A = \{\alpha_i\}_{i=1}^{KN}$, where $\alpha_i \sim U(0, k_s)$.

  $\mathcal{D}'_{support} = ReorderRandomly(\mathcal{D}_{support})$

  $\mathcal{D}_{interp} = LinearInterpolation(\mathcal{D}_{support}, \mathcal{D}'_{support}, A)$

  $\mathcal{D}_{aug} = Concatenate(\mathcal{D}_{aug}, \mathcal{D}_{interp})$

end for

$\mathcal{D}_{support} = \mathcal{D}_{aug}$

Algorithm 2 Data augmentation via triplet interpolation in N-way K-shot setting

initialization Support set data in N-way K-shot setting $\mathcal{D}_{support} = \{(\mathbf{x}_{11}, c_1), (\mathbf{x}_{12}, c_1), ..., (\mathbf{x}_{NK}, c_N)\}$, a repeat count of augmentation n_{aug}, and a spreading factor k_s.

$\mathcal{D}_{aug} = \mathcal{D}_{support}$

for $k = 1$ **to** n_{aug} **do**

  Sample sets of mixing coefficient $A_1 = \{\alpha_{1,i}\}_{i=1}^{KN}$, $A_2 = \{\alpha_{2,i}\}_{i=1}^{KN}$, and $A_3 = \{\alpha_{3,i}\}_{i=1}^{KN}$, where $\alpha_{1,i}, \alpha_{3,i} \sim U(0, k_s)$ and $\alpha_{2,i} \sim U(0, 1)$.

  $\mathcal{D}'_{support} = ReorderRandomly(\mathcal{D}_{support})$

  $\mathcal{D}''_{support} = ReorderRandomly(\mathcal{D}_{support})$

  $\mathcal{D}_{interp1} = LinearInterpolation(\mathcal{D}_{support}, \mathcal{D}'_{support}, A_1)$

  $\mathcal{D}_{interp2} = LinearInterpolation(\mathcal{D}'_{support}, \mathcal{D}''_{support}, A_2)$

  $\mathcal{D}_{interp} = LinearInterpolation(\mathcal{D}_{interp1}, \mathcal{D}_{interp2}, A_3)$

  $\mathcal{D}_{aug} = Concatenate(\mathcal{D}_{aug}, \mathcal{D}_{interp})$

end for

$\mathcal{D}_{support} = \mathcal{D}_{aug}$

4

Experiments and Results

Two types of few-shot classification experiments are performed in this thesis. The first uses a benchmark dataset to demonstrate that our data augmentation method works, and the second assumes a more practical and novel scenario using a visual inspection dataset. In this chapter, we describe the details of used datasets and the experimental setup of few-shot classification using transfer learning-framework and cosine classifier. Experimental results are also reported, followed by analysis and discussion.

4.1 Benchmark Experiments

4.1.1 Datasets

There are several representative benchmark datasets in the few-shot learning literature: Omniglot [27], EMNIST [28], CUB [29], and *mini*ImageNet [12]. Figure 4.1 shows some example images of these datasets. Unless otherwise mentioned, we follow the setup of [13] in the preparation of the benchmark datasets.

Omniglot is a dataset that contains 1623 handwritten characters from 50 languages. 20 examples exist for each character, where an individual person draws each example. We augment the character classes with rotations in 90, 180, 270 degrees. Consequently, we obtain a total of 6492 classes and the classes are divided into 4112 base classes, 688 validation classes, and 1692 novel classes.

EMNIST dataset contains 10 digits from 0 to 9, with English alphabetic characters in both upper and lower case letters, resulting in 62 classes in total. EMNIST dataset is not used alone, but is used for domain adaptation experiments between different character datasets following the experiments in [13]. We train a model with Omniglot dataset and evaluate the few-shot classification performance on EMNIST dataset. Therefore, we distribute a total of 62 classes evenly between validation classes and novel classes. For base classes, we first exclude 26 Latin characters from Omniglot dataset to avoid sharing information between base classes and novel classes. Since no data augmentation is applied to the rest of Omniglot dataset, 1597 classes belong to base classes in total.

CUB and ***mini*ImageNet** datasets are frequently used in the few-shot learning regime. CUB dataset contains 11,788 images from 200 bird species. Since the CUB dataset consists only of bird photos, it can be used for fine-grained recognition experiments. 200 classes are randomly divided into 100 base classes, 50 validation classes, and 50 novel classes for our experiments.

*mini*ImageNet is a dataset first introduced by [2]. It is a subset of ImageNet dataset [30] and consists of 100 classes with 600 color images per class. We use the split proposed in [12] which is widely adopted in the few-shot literature. We then have 64 base classes, 16 validation classes, and 20 novel classes.

The data belonging to the validation classes are used to verify the performance during the training stage when using the meta-learning framework. When using

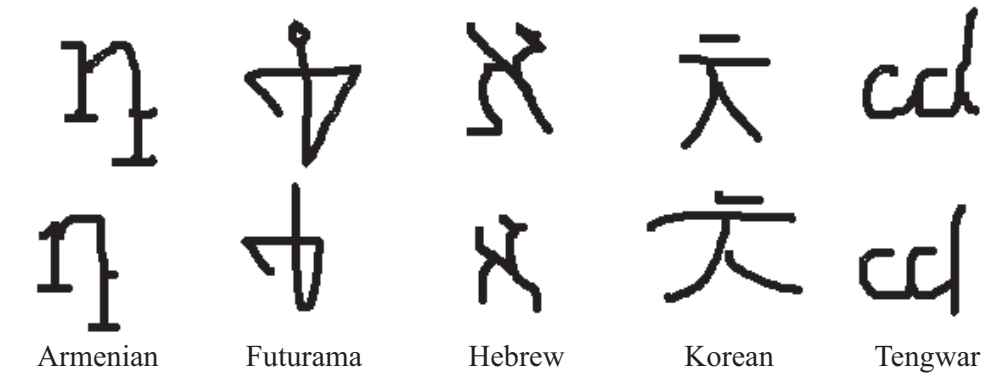
the transfer learning framework, however, there is no validation stage, so the validation data are not used at all.

4.1.2 Experiment Details

The common process of experiments with the benchmark datasets is as follows. In the pre-training stage, We train a model for the standard classification task using data from the base classes. The model consists of a CNN-based feature extractor followed by a cosine classifier. The Adam optimizer with learning rate 10^{-3} and weight decay 10^{-5} is used to train the model.

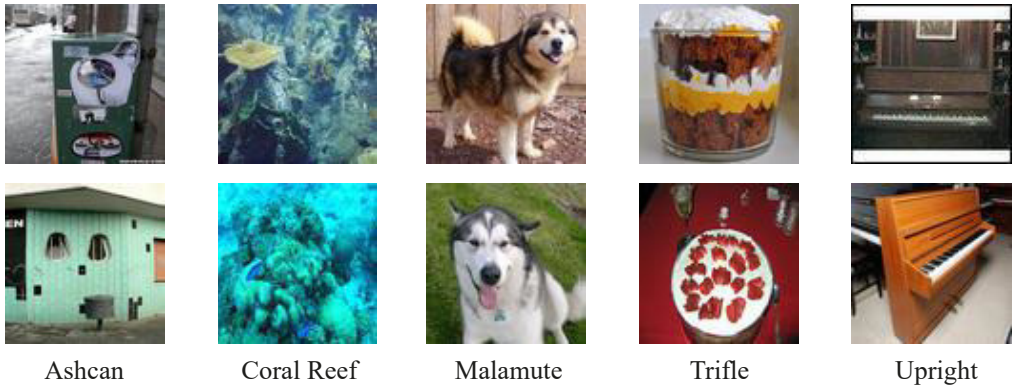
In the fine-tuning stage, we set a support set and a query set respectively using the novel class data to conform to the N-way K-shot problem. We evaluate the few-shot classification performance on 5-way 1-shot and 5-way 5-shot settings. A new cosine classifier is trained for 100 iterations using only the support set data with a batch size of 4. In addition, we conduct experiments with data augmentation via various types of interpolation and extrapolation in feature space. We compare the results from the proposed data augmentation against those without the augmentation. When data is augmented by our approach, there may be hundreds of data in the support set. We accordingly set the batch size to the smaller number between 64 and the total number of data in the support set. In the query set, there are 15 data for each class, leading to measuring classification performance using 75 data.

In the Omniglot-only experiment and domain adaptation experiment (Omniglot $\rightarrow$EMNIST), we adopt 4-layer CNN as a feature extractor. The learning model is pre-trained for 5 epochs without any image-manipulating data augmentation.



(a) Examples of Omniglot.

(b) Examples of CUB.



(c) Examples of *mini*ImageNet.

Figure 4.1: Examples of benchmark datasets.

On the other hand, CUB and *mini*ImageNet datasets that consist of real photos are much more complicated than character datasets, which means neural networks with larger capacity should be used as a feature extractor. We therefore adopt ResNet-18 [31] as our feature backbone and pre-train the model for 350 epochs. In the pre-training stage, we exploit several data augmentation methods at the image level, including random crop, horizontal flip, and color jitter.

4.1.3 Evaluation Results

We conduct experiments using the benchmark datasets on 5-way 1-shot and 5-way 5-shot classification setting: Omniglot, character domain adaptation (Omniglot $\rightarrow$ EMNIST), CUB, and *mini*ImageNet. We report the average of the classification results from 1200 test episodes and the 95% confidence intervals in Table 4.1 and Table 4.2.

The hyperparameters that can be adjusted when applying our data augmentation method are the repeat count of augmentation n_{aug} and the spreading factor k_s as described in Algorithm 1. For $n_{aug} > 0$, the number of data in the support set increases from KN to $(n_{aug} + 1)KN$. k_s controls how close the intermediate data created by interpolation is from the original endpoint data.

We report the results from the augmentation setting of $k_s = 1/3$. The repeat count n_{aug} varies depending on the dataset and how many "shots" we have. In Table 4.1, the results of Omniglot and character domain adaptation experiments are reported with $n_{aug} = 100$ for 1-shot setting and $n_{aug} = 30$ for 5-shot setting. In Table 4.2, we report the results of CUB and *mini*ImageNet experiments with $n_{aug} = 50$ for 1-shot setting and $n_{aug} = 25$ for 5-shot setting.

As we can observe from tables, few-shot classification performance improves with data augmentation in feature space. Among many experiment settings, the

character domain adaptation experiment with 5-shot classification shows significant improvement when data augmentation is applied. In the CUB experiments with 5-shot, on the other hand, the classification accuracy increased just slightly. Using triplet interpolation is better than using linear interpolation, but the difference is not so significant. The results of including extrapolation also show no noticeable performance improvement.

Table 4.1: 5-way classification results on Omniglot and character domain adaptation experiments ($\dagger$ = linear interpolation, $*$ = triplet interpolation, $\star$ = linear interpolation with extrapolation, $\sharp$ = triplet interpolation with extrapolation).

	Omniglot		Omniglot→EMNIST	
	1-shot	5-shot	1-shot	5-shot
With aug.†	96.16 ± 0.25	99.15 ± 0.08	70.31 ± 0.56	88.05 ± 0.38
With aug.*	96.19 ± 0.25	99.16 ± 0.09	70.51 ± 0.56	87.84 ± 0.39
With aug.*	96.18 ± 0.25	99.13 ± 0.09	70.47 ± 0.57	87.85 ± 0.39
With aug.$^{\sharp}$	96.06 ± 0.27	99.12 ± 0.08	70.74 ± 0.56	87.81 ± 0.40
Without aug.	95.41 ± 0.29	98.41 ± 0.14	69.01 ± 0.56	82.83 ± 0.44

Table 4.2: 5-way classification results on CUB and *mini*ImageNet experiments ($\dagger$ = linear interpolation, $*$ = triplet interpolation, $\star$ = linear interpolation with extrapolation, $\sharp$ = triplet interpolation with extrapolation).

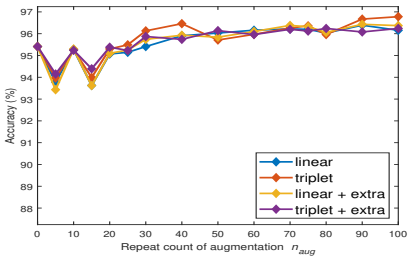
	CUB		*mini*ImageNet	
	1-shot	5-shot	1-shot	5-shot
With aug.†	68.58 ± 0.64	83.78 ± 0.37	50.10 ± 0.55	69.73 ± 0.45
With aug.*	68.60 ± 0.63	83.51 ± 0.37	50.55 ± 0.54	69.67 ± 0.46
With aug.*	68.65 ± 0.63	83.94 ± 0.36	50.46 ± 0.57	69.32 ± 0.47
With aug.$^{\sharp}$	69.07 ± 0.63	84.13 ± 0.36	49.84 ± 0.56	69.50 ± 0.45
Without aug.	67.38 ± 0.63	83.32 ± 0.37	48.59 ± 0.55	68.83 ± 0.46

4.1.4 Discussion

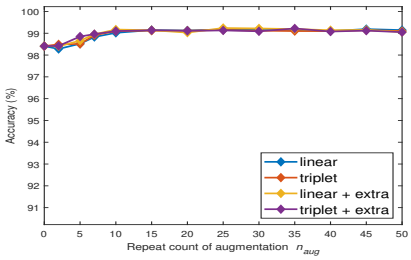
Effect of varying n_{aug} with fixed k_s We measure the accuracy by varying n_{aug} with fixed k_s to see how n_{aug} affects the few-shot classification. The results are in Figure 4.2. Regardless of the dataset and "shot", classification performance with small n_{aug} is not evidently better than that without augmentation and even worse in most cases. In contrast, when n_{aug} becomes large enough, as the n_{aug} value increases the performance also improves. Once n_{aug} reaches a certain level or more, the performance change according to n_{aug} is no longer significant.

We have anticipated that, if there is a performance improvement via our data augmentation, the improvement will be greater in character datasets like Omniglot than in datasets composed of real photos such as CUB or *mini*ImageNet. This anticipation is evident in the 5-shot experiments, where the real photo datasets have a competitive accuracy when $n_{aug} = 0$. This may be because the complexity of the image affects how meaningful interpolation and extrapolation in feature space are. A character image consists only of a handwritten character with a simple white or black background. In real photos, however, subjects have a variety of poses, and the background is also different for each image. We should also consider the angle and lighting at which the picture is taken.

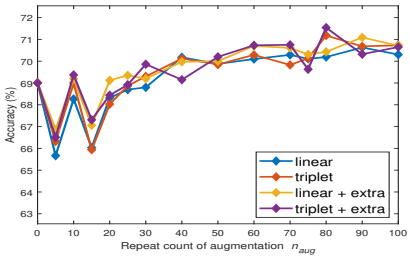
Effect of varying k_s with fixed n_{aug} In Figure 4.3, the results from varying k_s with fixed n_{aug} are reported. We fix the n_{aug} following the setting of Section 4.1.3. The results with changing k_s show no particular tendency. When k_s is 1, the data generated through our data augmentation could be mislabeled. We thus have expected the accuracy would be greatly degraded. The actual results with $k_s = 1$ are, however, not worse than we expected and even competitive to the results with $k_s \leq 0.5$. On the other hand, when k_s becomes too small, we have not anticipated



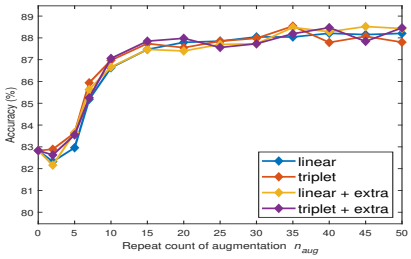
(a) Omniglot 1-shot.



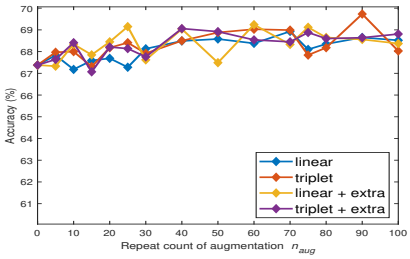
(b) Omniglot 5-shot.



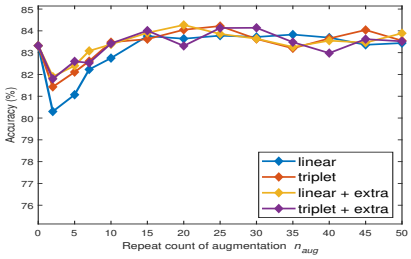
(c) Omniglot → EMNIST 1-shot.



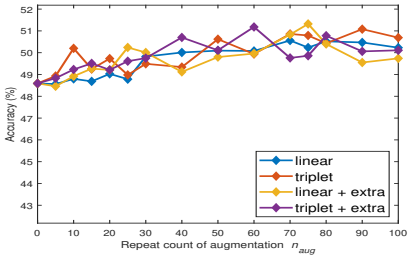
(d) Omniglot → EMNIST 5-shot.



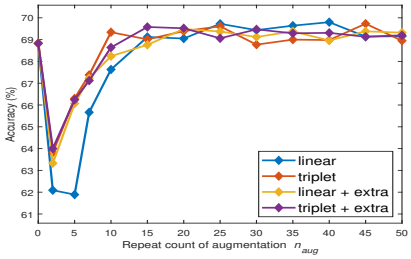
(e) CUB 1-shot.



(f) CUB 5-shot.

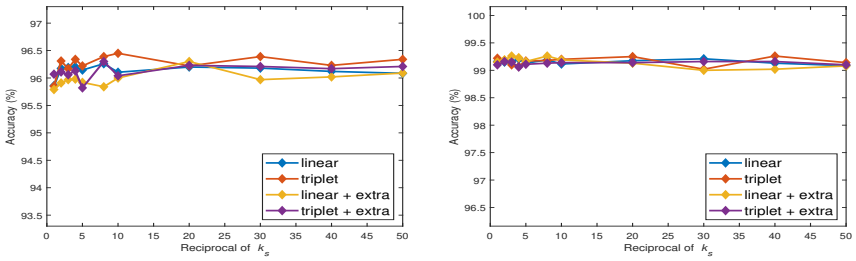


(g) *mini*ImageNet 1-shot.



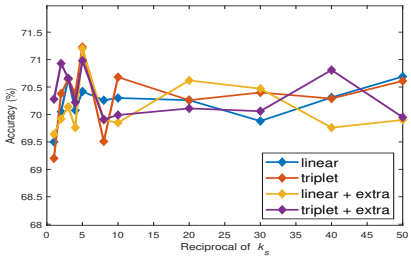
(h) *mini*ImageNet 5-shot.

Figure 4.2: Classification accuracy based on the change of n_{aug} with fixed $k_s = 1/3$.

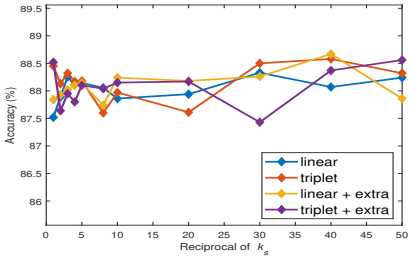


(a) Omniglot 1-shot.

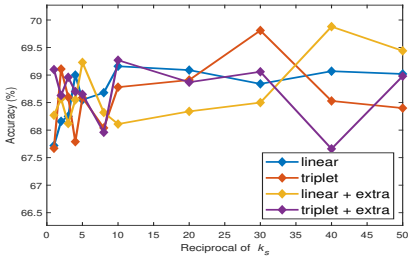
(b) Omniglot 5-shot.



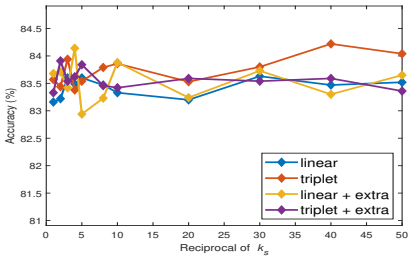
(c) Omniglot → EMNIST 1-shot.



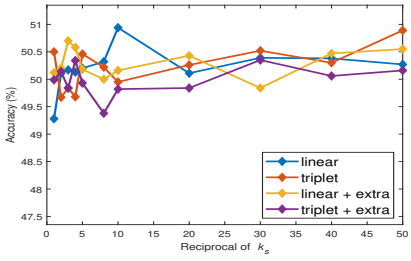
(d) Omniglot → EMNIST 5-shot.



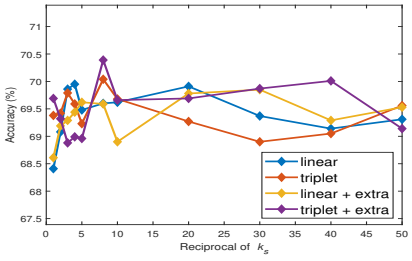
(e) CUB 1-shot.



(f) CUB 5-shot.



(g) *mini*ImageNet 1-shot.



(h) *mini*ImageNet 5-shot.

Figure 4.3: Classification accuracy based on the change of k_s with fixed n_{aug}.

that our data augmentation would work because the generated data must be concentrated around the endpoints. Nevertheless, even the performance with small k_s (e.g., $k_s = 1/50$) turns out to be competitive. It can thus be concluded that the effect of k_s appears insignificant if n_{aug} is large enough. The difference according to the type of interpolation or the inclusion of extrapolation is not obvious as well.

4.2 Visual Inspection Experiments

4.2.1 MVTec-AD Dataset

MVTec-AD (MVTec Anomaly Detection) dataset is invented for the task of anomaly detection [19]. Mimicking real-world visual inspection scenarios, it contains 5354 high-resolution color images of 15 product categories: 10 objects and 5 textures. There are 88 classes in total, including 15 non-defect classes for each product category and 73 different types of defect classes (e.g., scratch, dent, or contamination) across the 15 categories. Several examples of the dataset are illustrated in Figure 4.4 with one non-defective and one defective data for every category. For each category, images on the top row represent non-defective data, whereas the middle row shows defective ones. The bottom row shows a close-up view of the defective region [19]. Following the original intention of anomaly detection, only non-defective data should be used as training data and the trained model must become capable of classifying unseen defective data. Therefore, the training set of one category consists only of non-defective data, while the test set contains both defective and non-defective data.

We use the dataset for 2-way few-shot classification to determine if an unseen image is defective or non-defective. We thus manipulate the dataset to better suit the transfer learning-based approach. What we first do is to combine the data for

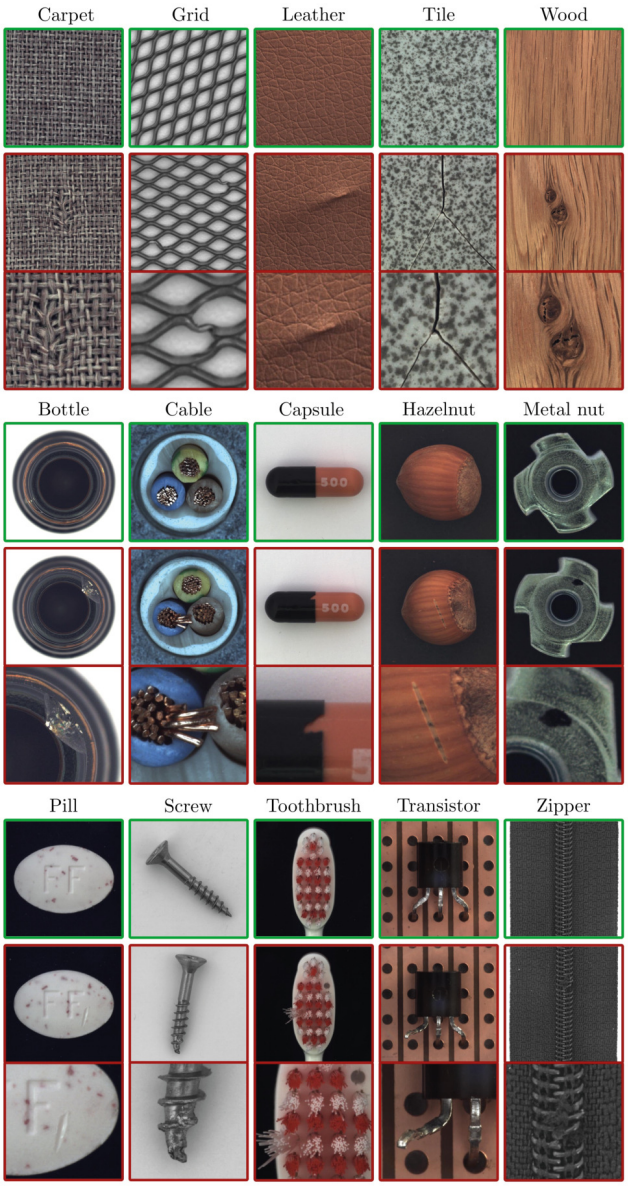


Figure 4.4: Examples of all ten object and five texture categories in the MVTec-AD dataset.

each category that are originally divided into the training set and test set. Afterward, one of 15 product categories is designated as a novel category and the remaining 14 categories become base categories. For the novel category, we integrate all kinds of defect classes into NG classes, so that there are only two classes, OK and NG. Consequently, more than 80 base classes and 2 novel classes exist with 15 different cases depending on what the novel category is. We split again the data from base classes into the training set and validation set at a ratio of 8:2. The validation set is used to monitor the model's generalization performance during training for the standard classification task.

4.2.2 AUC Score

Receiver operating characteristics (ROC) curves and area under curve (AUC) values are frequently used to measure the performance of binary classification models [32]. A ROC curve plots true positive rate (TPR) against false positive rate (FPR) under various threshold values of the classifier. TPR (also called *recall*) is the proportion of positive samples correctly categorized as positive among total positive samples. FPR represents the ratio between the number of negative samples wrongly predicted as positive and the total number of actual negative samples. Lowering the classification threshold means the classifier becomes less conservative, and it increases both FPR and TPR because more samples are classified as positive.

The AUC is the area underneath the ROC curve. AUC = 1 means we get a perfect classifier, while AUC = 0.5 indicates that the classifier makes a purely random prediction. A larger AUC value means a better classifier. When there exists a class imbalance, AUC is a good measure of classification performance. Using the classification accuracy may not reflect the actual performance of the trained

model if the data is imbalanced. Besides, since the cost of a false positive often differs from the cost of a false negative, a threshold-independent score needs to be considered.

As has been noted, one of the practical difficulties of visual inspection is that defective data is often difficult to collect while non-defective data is plentiful. Let us assume that we are given 95 non-defective data and 5 defective data as an illustration. If all the data are predicted to be non-defective, the classification accuracy is 95%, which seems pretty good. Nevertheless, since all the defective data are wrongly classified as non-defective, a big problem arises that the defective product cannot be filtered out. In conclusion, the AUC score is definitely suitable for the visual inspection scenario.

4.2.3 Experiment Details

Two main scenarios are considered in visual inspection experiments depending on the presence of base class data. The first scenario assumes that labeled data of the novel category is not given much, but 14 base categories are rich in labeled data. This simulates the situation where a new product has started to be manufactured in a line that has already produced other products. We can thus pre-train the model from scratch using abundant data of about 80 base classes in this scenario. On the other hand, the second scenario is when data of base categories is not given at all, and only a small amount of labeled data from the novel category is available. It simulates manufacturing products completely for the first time. Since the base class data is assumed to be absent, the experiment was performed using a model pre-trained with the ImageNet dataset instead.

ResNet-18 is used as a feature extractor in visual inspection experiments. The pre-training epoch is set to 200, but the model used for fine-tuning is the one

that shows the best validation performance. The optimizer and data augmentation at the image level during pre-training and batch size setting for fine-tuning are identical to the CUB and *mini*ImageNet experiments.

Balanced support set experiment The pre-trained model is fine-tuned to solve the 2-way 1-shot and 2-way 5-shot classification problems for the novel category data. In the novel category dataset, one (1-shot) or five (5-shot) randomly selected examples from OK and NG class respectively are incorporated into the support set, whereas the query set contains imbalanced data with much more non-defective examples than the defective counterparts. To construct the query set, we first consider data that is not included in the support set as query set candidates. Since the number of non-defective candidates is larger than defective candidates, we exclude some non-defective data at random to ensure that the number of non-defective candidates and defective candidates can be the same. After that, the query set finally consists of 5 defective data extracted from the defective candidates and all of the remaining non-defective candidates. The number of OK class data in the query set depends on what the novel category is. Like the benchmark experiments, we compare the few-shot classification performance depending on whether or not the support set data is augmented in feature space.

Imbalanced support set experiment In addition, we conduct experiments with a lot of OK class data in the support set. This scenario is, strictly speaking, about an imbalanced setting rather than a few-shot setting. It applies the class imbalance setting of visual inspection to the support set as well as the query set. If we configure the support set and query set as described above, non-defective data that is not included in either set still exists. We use the remaining non-defective data as part of the support set. Therefore, "K-shot" in this case means there are

K defective examples and large amounts of non-defective examples in the support set. How to augment data through interpolation or extrapolation in the imbalanced setting is slightly different from the few-shot setting. Since non-defective data is sufficient, only defective data is the target of data augmentation, making the number of defective data equal to the number of non-defective data.

4.2.4 Evaluation Results

We conduct experiments with the MVtec-AD dataset to evaluate the 2-way 1-shot and 2-way 5-shot classification performance. The average value of AUC from 1200 randomly selected test episodes is reported with 95% confidence interval in Table 4.3 and 4.4. We also display side-by-side the results of different pre-trained models, each pre-trained with MVTec-AD and ImageNet respectively. Furthermore, following the way the authors of [19] split the entire dataset into objects and textures, we divide the table into two parts: the upper part for 10 object categories and the lower part for 5 texture categories.

In Table 4.3, the results with the balanced support set are shown. The repeat count of augmentation n_{aug} is set to 50 and 25 for 1-shot and 5-shot setting respectively. The spreading factor k_s is fixed to 1/4. The results with the imbalanced support set are reported in Table 4.4, where our augmentation is applied only to defective data so that the number of defective data is the same as the number of non-defective ones. Spreading factor k_s is set to 1/4 as well in the imbalanced case. In both tables, the category name with †, $*$, $\star$, and ♯ means that the corresponding row shows the results of linear interpolation, triplet interpolation, linear interpolation with extrapolation, and triplet interpolation with extrapolation respectively. In the 2-way 1-shot setting, the triplet method is not evaluated since there are only two data in the support set.

4.2.5 Discussion

Balanced support set experiment Overall, no significant performance improvement is observed when our data augmentation is applied. Some categories show worse performance, while those with better AUC improved only slightly. Furthermore, like the benchmark results, there is no significant improvement or deterioration in performance depending on the interpolation and extrapolation types. These observations lead to the conclusion that our augmentation method has no meaningful effect in the balanced support set experiment. However, there are several categories where performance improvement is noticeable in 5-shot setting: bottle, hazelnut, and metal nut. What they have in common is that the object itself occupies a large portion of the image and most defects are relatively apparent.

From a practical perspective, since AUC values are not even over 95% at all, it seems difficult to apply the few-shot learning scheme to visual inspection problems. The only categories that show the potential are leather and bottle, with AUC values above 90% in the 5-shot setting. Comparing the type of base class data, the performance of the model pre-trained with ImageNet dataset is better except for fabric categories such as leather and carpet. This indicates that large datasets such as ImageNet dataset, albeit not intended for visual inspection, may be more effective for transfer learning in visual inspection.

Imbalanced support set experiment This time, we analyze imbalanced support set experiments that simulate a more realistic scenario of visual inspection. In our experiments, there are hundreds of non-defective data in the support set, while there is only one or five defective data. Since non-defective data dominates in the support set, the fine-tuned model predicts most query images as non-defective.

This tendency can be observed in Table 4.4; the results of 1-shot and 5-shot are not very different, and sometimes the result of 1-shot is even better.

Using our data augmentation method, we increase the amount of defective data to a level similar to the amount of non-defective ones. As shown in Table 4.4, our method leads to the substantial improvement of AUC in many categories, particularly under a 5-shot condition. Results of our method in 1-shot setting, however, are not as meaningful as in 5-shot except when evaluating texture categories using the pre-trained model with ImageNet dataset. This is probably due to the variety of defects in each category. There are as many as eight different kinds of defects in one category. Therefore, in the case of 1-shot experiment, defect types unseen in the fine-tuning process are more likely to exist in the query set than in the 5-shot case. Even if we increase the amount of NG data by interpolation with OK data, it may not be easy to cover unseen defect types.

Similar to the results of the balanced support set experiments, our augmentation method is effective in bottle, hazelnut, and metal nut among the object categories. In addition, performance is improved by using our data augmentation method in the texture categories except for grid, and the effect is more dramatic when using the pre-trained model with ImageNet dataset. Our method does not work for the grid category only, probably because the grid has the shape of entangled wires, while the other texture categories have solid surfaces. Besides, when our augmentation method is applied, the model pre-trained with ImageNet dataset outperforms the model pre-trained with MVTec-AD dataset in most categories except for the leather and carpet. Even in these two categories, the performance of both models is competitive in the 5-shot setting. What kind of interpolation and extrapolation is used does not make any meaningful difference. In conclusion, using the ImageNet-driven model and our data augmentation strategy is likely to

achieve good classification performance under the imbalanced support set condition regardless of the type of interpolation and extrapolation.

Table 4.3: 2-way classification results on MVTec-AD experiments with the **balanced** support set ($\dagger$ = linear interpolation, $*$ = triplet interpolation, $\star$ = linear interpolation with extrapolation, $\sharp$ = triplet interpolation with extrapolation).

		MVTec-AD pre-trained		ImageNet pre-trained	
	Category	1-shot	5-shot	1-shot	5-shot
Object	Bottle†	68.06 ± 0.94	80.45 ± 0.68	77.00 ± 0.86	91.96 ± 0.44
	Bottle*	-	79.87 ± 0.68	-	91.52 ± 0.44
	Bottle*	68.23 ± 0.91	80.50 ± 0.69	77.10 ± 0.85	92.14 ± 0.43
	Bottle$^{\sharp}$	-	80.25 ± 0.67	-	91.57 ± 0.44
	Bottle	67.47 ± 0.95	80.22 ± 0.66	77.87 ± 0.85	91.06 ± 0.47
	Cable†	54.31 ± 0.87	61.85 ± 0.82	57.43 ± 0.81	66.55 ± 0.76
	Cable*	-	60.25 ± 0.85	-	65.95 ± 0.77
	Cable*	55.17 ± 0.88	61.49 ± 0.81	57.17 ± 0.82	66.38 ± 0.78
	Cable$^{\sharp}$	-	61.69 ± 0.80	-	66.28 ± 0.74
	Cable	56.38 ± 0.85	61.32 ± 0.84	57.39 ± 0.84	66.63 ± 0.78
	Capsule†	52.57 ± 0.82	54.86 ± 0.82	54.22 ± 0.7	59.82 ± 0.78
	Capsule*	-	55.30 ± 0.80	-	59.52 ± 0.78
	Capsule*	52.62 ± 0.81	56.01 ± 0.79	54.71 ± 0.78	60.01 ± 0.74
	Capsule$^{\sharp}$	-	56.34 ± 0.80	-	60.04 ± 0.77
	Capsule	52.11 ± 0.84	55.64 ± 0.79	54.64 ± 0.74	59.47 ± 0.76
	Hazelnut†	56.18 ± 0.86	63.53 ± 0.77	69.32 ± 0.84	85.15 ± 0.55
	Hazelnut*	-	63.06 ± 0.79	-	84.28 ± 0.56
	Hazelnut*	55.87 ± 0.83	63.07 ± 0.79	69.20 ± 0.86	85.10 ± 0.56
	Hazelnut$^{\sharp}$	-	63.41 ± 0.77	-	85.02 ± 0.58
	Hazelnut	57.05 ± 0.85	62.33 ± 0.80	69.80 ± 0.84	83.81 ± 0.58

Metal nut†	58.85 ± 0.98	69.86 ± 0.82	59.51 ± 0.82	73.12 ± 0.77
Metal nut*	-	68.26 ± 0.85	-	72.11 ± 0.80
Metal nut*	60.42 ± 0.98	68.57 ± 0.84	59.42 ± 0.86	72.86 ± 0.72
Metal nut$^{\sharp}$	-	69.54 ± 0.83	-	72.63 ± 0.76
Metal nut	59.25 ± 0.99	67.23 ± 0.84	59.29 ± 0.82	71.73 ± 0.75
Pill†	54.77 ± 0.82	59.74 ± 0.80	56.12 ± 0.81	62.56 ± 0.76
Pill*	-	59.33 ± 0.79	-	62.39 ± 0.74
Pill*	54.18 ± 0.83	59.80 ± 0.78	56.15 ± 0.78	61.72 ± 0.77
Pill$^{\sharp}$	-	60.03 ± 0.79	-	62.31 ± 0.77
Pill	54.26 ± 0.80	58.97 ± 0.77	55.76 ± 0.79	62.45 ± 0.75
Screw†	51.08 ± 0.78	51.00 ± 0.77	53.06 ± 0.80	56.91 ± 0.79
Screw*	-	50.82 ± 0.76	-	56.83 ± 0.78
Screw*	50.71 ± 0.77	51.50 ± 0.75	53.21 ± 0.81	56.94 ± 0.77
Screw$^{\sharp}$	-	52.29 ± 0.73	-	56.62 ± 0.78
Screw	51.20 ± 0.75	51.37 ± 0.77	52.60 ± 0.81	56.70 ± 0.77
Toothbrush†	51.40 ± 0.78	52.76 ± 0.75	55.27 ± 0.84	62.00 ± 0.82
Toothbrush*	-	52.11 ± 0.76	-	61.14 ± 0.83
Toothbrush*	51.48 ± 0.77	52.44 ± 0.75	55.12 ± 0.84	61.90 ± 0.83
Toothbrush$^{\sharp}$	-	52.63 ± 0.78	-	61.62 ± 0.80
Toothbrush	51.34 ± 0.78	52.65 ± 0.74	55.50 ± 0.86	61.13 ± 0.81
Transistor†	58.10 ± 0.94	68.13 ± 0.80	59.37 ± 0.92	69.81 ± 0.78
Transistor*	-	67.63 ± 0.80	-	70.04 ± 0.78
Transistor*	58.65 ± 0.97	68.11 ± 0.80	55.12 ± 0.84	70.23 ± 0.79
Transistor$^{\sharp}$	-	67.93 ± 0.81	-	70.41 ± 0.79
Transistor	58.72 ± 0.94	67.34 ± 0.78	60.34 ± 0.91	70.04 ± 0.74
Zipper†	61.77 ± 1.07	71.23 ± 0.77	62.37 ± 0.88	74.99 ± 0.70
Zipper*	-	70.98 ± 0.77	-	74.59 ± 0.73
Zipper*	62.26 ± 1.03	71.40 ± 0.76	62.89 ± 0.88	74.71 ± 0.70
Zipper$^{\sharp}$	-	71.06 ± 0.76	-	75.02 ± 0.69
Zipper	62.91 ± 1.04	70.91 ± 0.82	62.42 ± 0.88	74.27 ± 0.69

Texture	Leather†	81.94 ± 0.89	93.70 ± 0.31	77.60 ± 0.89	90.80 ± 0.40
	Leather*	-	93.28 ± 0.38	-	90.21 ± 0.41
	Leather*	82.69 ± 0.86	94.16 ± 0.29	77.78 ± 0.85	90.59 ± 0.41
	Leather$^{\sharp}$	-	93.83 ± 0.34	-	91.00 ± 0.37
	Leather	81.46 ± 0.90	93.76 ± 0.34	77.97 ± 0.85	90.97 ± 0.37
	Carpet†	74.32 ± 1.20	84.68 ± 0.65	68.22 ± 0.94	81.06 ± 0.64
	Carpet*	-	84.10 ± 0.69	-	80.33 ± 0.65
	Carpet*	73.73 ± 1.20	84.39 ± 0.64	67.93 ± 0.94	81.37 ± 0.62
	Carpet$^{\sharp}$	-	84.68 ± 0.64	-	80.71 ± 0.62
	Carpet	74.84 ± 1.17	84.33 ± 0.94	69.14 ± 0.90	81.15 ± 0.62
	Grid†	51.61 ± 0.76	54.15 ± 0.75	52.76 ± 0.80	56.23 ± 0.79
	Grid*	-	55.74 ± 0.80	-	56.54 ± 0.81
	Grid*	50.89 ± 0.74	53.78 ± 0.77	52.03 ± 0.84	55.94 ± 0.80
	Grid$^{\sharp}$	-	56.37 ± 0.79	-	56.42 ± 0.79
	Grid	51.41 ± 0.75	54.11 ± 0.77	52.10 ± 0.79	55.13 ± 0.79
	Tile†	60.11 ± 0.93	69.97 ± 0.73	67.95 ± 0.99	80.68 ± 0.66
	Tile*	-	70.19 ± 0.71	-	81.07 ± 0.66
	Tile*	59.69 ± 0.97	69.81 ± 0.76	69.42 ± 0.95	80.48 ± 0.67
	Tile$^{\sharp}$	-	69.93 ± 0.72	-	81.60 ± 0.64
	Tile	59.81 ± 0.96	69.37 ± 0.73	67.84 ± 0.99	80.77 ± 0.64
	Wood†	54.74 ± 0.83	62.48 ± 0.79	60.74 ± 0.91	73.12 ± 0.72
	Wood*	-	63.05 ± 0.77	-	71.73 ± 0.70
	Wood*	55.10 ± 0.80	63.19 ± 0.79	61.07 ± 0.89	72.84 ± 0.69
	Wood$^{\sharp}$	-	62.80 ± 0.80	-	73.53 ± 0.72
	Wood	55.40 ± 0.82	62.45 ± 0.77	61.26 ± 0.87	72.76 ± 0.73

Table 4.4: 2-way classification results on MVTec-AD experiments with the **imbalanced** support set († = linear interpolation, ∗ = triplet interpolation, ⋆ = linear interpolation with extrapolation, ♯ = triplet interpolation with extrapolation).

		MVTec-AD pre-trained		ImageNet pre-trained	
	Category	1-shot	5-shot	1-shot	5-shot
Object	Bottle†	76.63 ± 0.78	85.58 ± 0.56	89.15 ± 0.55	95.54 ± 0.32
	Bottle∗	-	84.73 ± 0.59	-	95.74 ± 0.30
	Bottle⋆	77.27 ± 0.78	85.53 ± 0.56	89.48 ± 0.51	95.81 ± 0.30
	Bottle♯	-	85.46 ± 0.55	-	96.07 ± 0.28
	Bottle	90.67 ± 0.41	90.38 ± 0.45	88.46 ± 0.35	87.73 ± 0.36
	Cable†	61.58 ± 0.82	64.51 ± 0.77	63.45 ± 0.89	72.96 ± 0.74
	Cable∗	-	64.73 ± 0.79	-	73.38 ± 0.76
	Cable⋆	61.73 ± 0.80	66.11 ± 0.75	63.17 ± 0.88	73.57 ± 0.74
	Cable♯	-	66.02 ± 0.75	-	73.45 ± 0.76
	Cable	75.08 ± 0.67	75.88 ± 0.64	56.58 ± 0.76	56.47 ± 0.73
	Capsule†	54.28 ± 0.81	59.17 ± 0.79	59.86 ± 0.78	64.54 ± 0.74
	Capsule∗	-	58.80 ± 0.80	-	64.17 ± 0.78
	Capsule⋆	53.93 ± 0.80	59.37 ± 0.77	60.92 ± 0.74	65.05 ± 0.69
	Capsule♯	-	58.91 ± 0.80	-	64.49 ± 0.73
	Capsule	56.24 ± 0.69	56.98 ± 0.67	68.44 ± 0.74	68.66 ± 0.75
	Hazelnut†	60.27 ± 0.82	68.04 ± 0.73	80.46 ± 0.67	92.32 ± 0.38
	Hazelnut∗	-	68.61 ± 0.72	-	92.40 ± 0.39
	Hazelnut⋆	60.27 ± 0.78	68.23 ± 0.71	80.17 ± 0.66	91.63 ± 0.43
	Hazelnut♯	-	68.33 ± 0.71	-	92.32 ± 0.39
	Hazelnut	57.54 ± 0.70	57.13 ± 0.74	66.16 ± 0.65	65.56 ± 0.63
	Metal nut†	63.53 ± 0.94	73.00 ± 0.72	69.66 ± 0.77	80.33 ± 0.68
	Metal nut∗	-	72.81 ± 0.73	-	80.97 ± 0.67
	Metal nut⋆	62.98 ± 0.92	72.20 ± 0.73	70.22 ± 0.77	80.63 ± 0.67

	Metal nut$^{\sharp}$	-	73.09 ± 0.75	-	80.38 ± 0.64
	Metal nut	72.08 ± 0.67	71.28 ± 0.68	69.36 ± 0.61	68.86 ± 0.59
	Pill†	57.06 ± 0.80	63.32 ± 0.76	61.99 ± 0.77	67.81 ± 0.72
	Pill*	-	63.56 ± 0.73	-	67.83 ± 0.73
	Pill*	57.64 ± 0.79	63.57 ± 0.73	62.49 ± 0.79	68.01 ± 0.70
	Pill$^{\sharp}$	-	63.66 ± 0.71	-	67.34 ± 0.74
	Pill	56.88 ± 0.74	57.29 ± 0.73	65.71 ± 0.68	66.00 ± 0.68
	Screw†	52.43 ± 0.74	53.47 ± 0.76	57.96 ± 0.75	61.08 ± 0.77
	Screw*	-	53.28 ± 0.76	-	60.82 ± 0.75
	Screw*	51.74 ± 0.77	52.55 ± 0.75	58.04 ± 0.76	61.04 ± 0.76
	Screw$^{\sharp}$	-	52.84 ± 0.76	-	60.68 ± 0.79
	Screw	51.78 ± 0.72	51.83 ± 0.73	63.00 ± 0.69	62.85 ± 0.71
	Toothbrush†	53.70 ± 0.78	52.23 ± 0.79	59.56 ± 0.82	64.29 ± 0.81
	Toothbrush*	-	52.40 ± 0.78	-	64.05 ± 0.81
	Toothbrush*	54.09 ± 0.77	52.54 ± 0.79	60.61 ± 0.80	65.68 ± 0.79
	Toothbrush$^{\sharp}$	-	52.77 ± 0.77	-	65.21 ± 0.80
	Toothbrush	64.62 ± 0.73	65.50 ± 0.77	70.56 ± 0.73	70.34 ± 0.71
	Transistor†	64.86 ± 0.84	73.53 ± 0.68	70.33 ± 0.76	76.87 ± 0.66
	Transistor*	-	73.91 ± 0.70	-	76.74 ± 0.64
	Transistor*	64.66 ± 0.85	73.52 ± 0.68	71.46 ± 0.73	77.19 ± 0.66
	Transistor$^{\sharp}$	-	73.27 ± 0.69	-	76.51 ± 0.65
	Transistor	67.18 ± 0.69	67.07 ± 0.72	75.16 ± 0.71	75.48 ± 0.68
	Zipper†	69.11 ± 0.86	75.47 ± 0.61	73.97 ± 0.69	81.30 ± 0.55
	Zipper*	-	74.90 ± 0.64	-	81.40 ± 0.55
	Zipper*	70.30 ± 0.81	75.00 ± 0.63	74.17 ± 0.69	81.87 ± 0.54
	Zipper$^{\sharp}$	-	75.61 ± 0.63	-	81.98 ± 0.54
	Zipper	75.35 ± 0.68	74.94 ± 0.69	73.43 ± 0.55	73.87 ± 0.53
Texture	Leather†	91.56 ± 0.38	96.24 ± 0.19	87.02 ± 0.59	95.13 ± 0.23
	Leather*	-	96.14 ± 0.20	-	95.34 ± 0.22
	Leather*	92.07 ± 0.36	96.30 ± 0.20	86.41 ± 0.62	95.40 ± 0.21

Leather$^{\sharp}$	-	96.02 ± 0.21	-	95.08 ± 0.22
Leather	85.51 ± 0.48	84.44 ± 0.50	50.84 ± 0.63	48.22 ± 0.62
Carpet†	79.04 ± 1.04	86.51 ± 0.59	73.96 ± 0.86	86.85 ± 0.50
Carpet*	-	86.37 ± 0.59	-	86.28 ± 0.52
Carpet*	80.01 ± 0.97	86.43 ± 0.61	73.13 ± 0.88	86.70 ± 0.49
Carpet$^{\sharp}$	-	86.54 ± 0.60	-	86.38 ± 0.51
Carpet	79.11 ± 0.58	78.45 ± 0.61	47.83 ± 0.68	46.20 ± 0.68
Grid†	55.44 ± 0.75	56.26 ± 0.75	55.20 ± 0.79	58.14 ± 0.80
Grid*	-	56.20 ± 0.75	-	58.58 ± 0.78
Grid*	55.72 ± 0.78	56.73 ± 0.76	55.31 ± 0.79	58.82 ± 0.78
Grid$^{\sharp}$	-	56.57 ± 0.73	-	58.93 ± 0.77
Grid	58.94 ± 0.78	59.03 ± 0.78	53.97 ± 0.71	53.29 ± 0.72
Tile†	66.95 ± 0.83	75.49 ± 0.59	74.87 ± 0.89	85.97 ± 0.54
Tile*	-	75.38 ± 0.58	-	86.30 ± 0.50
Tile*	67.86 ± 0.77	75.60 ± 0.59	75.26 ± 0.87	85.73 ± 0.52
Tile$^{\sharp}$	-	75.72 ± 0.58	-	85.62 ± 0.53
Tile	65.79 ± 0.75	65.71 ± 0.73	51.65 ± 0.73	49.85 ± 0.75
Wood†	62.77 ± 0.78	68.76 ± 0.72	64.96 ± 0.85	79.59 ± 0.60
Wood*	-	69.04 ± 0.71	-	79.88 ± 0.59
Wood*	63.16 ± 0.74	69.19 ± 0.74	64.91 ± 0.84	79.88 ± 0.60
Wood$^{\sharp}$	-	68.74 ± 0.72	-	79.61 ± 0.61
Wood	65.05 ± 0.76	65.01 ± 0.75	44.92 ± 0.61	44.48 ± 0.65

4.2.6 Toward Reducing Labeling Costs

Based on the experimental results so far, it seems hard to obtain satisfactory performance entirely only through the few-shot learning scheme in visual inspection problems due to low accuracy. However, even though we cannot classify a whole query set very accurately, we can reduce labeling costs by roughly filtering out non-defective data. Let's say we adjust the threshold of the classifier so that the query data is more easily predicted as defective data. If all actual defective data can be classified as defective data, data predicted as non-defective data can simply be labeled as non-defective data.

In Table 4.5, we report the results of 2-way 5-shot classification experiments with the imbalanced support set and query set in the form of a confusion matrix. The rows with †, ∗, ⋆, and ♯ correspond to the results of linear interpolation, triplet interpolation, linear interpolation with extrapolation, and triplet interpolation with extrapolation respectively. By adding 1 to the score for predicting defective data, the query data is classified more as the defective one. Since the number of non-defective data in a query set is different for each category, to facilitate comparison, the confusion matrix is scaled as if there are 95 non-defective data. We also set that there are 5 defective data in query set regardless of category. We conduct experiments only with the 4 categories that show high AUC values in Table 4.4.

As shown in Table 4.5, the actual defective data is mostly classified as non-defective data without our augmentation method. However, when interpolation or extrapolation is applied, the defective query data is properly predicted. Although non-defective data may be misclassified as defective one, this is not a problem for the purpose of roughly filtering out non-defective data. A closer analysis shows

that there is no significant difference between the results of applying linear interpolation and triplet interpolation. On the other hand, when extrapolated data is included, the input data is classified more as non-defective. Since this tendency acts more strongly on actual non-defective data, when extrapolation is applied, the efficiency of filtering increases while the reliability deteriorates slightly. Based on the AUC results in the imbalanced setting, however, we can conjecture that the addition of extrapolated data does not greatly affect the filtering performance. It's a matter of how much weight is given to the prediction score. If we add extra weight to the prediction score when extrapolation is included, we can get confusion matrices similar to those with only interpolation.

The use of the model pre-trained with ImageNet dataset increases the rate at which defective data is correctly classified in all four categories, which means the filtering can be more reliable. When using the ImageNet-driven model, the classification performance of non-defective data is better in bottle and hazelnut, but worse in the other two fabric categories. Thus, in the fabric categories, there exists a trade-off between the reliability and efficiency of filtering depending on the dataset for pre-training.

Table 4.5: 2-way 5-shot classification results on MVTec-AD experiments with the imbalanced support set and weighted prediction score († = linear interpolation, ∗ = triplet interpolation, ⋆ = linear interpolation with extrapolation, ♯ = triplet interpolation with extrapolation).

		MVTec-AD pre-trained		ImageNet pre-trained	
	Category	Predicted-OK	Predicted-NG	Predicted-OK	Predicted-NG
Bottle	Actual-OK†	18.77	76.23	52.73	42.27
	Actual-NG†	0.13	4.87	0.11	4.89
	Actual-OK*	17.95	77.05	52.94	42.06
	Actual-NG*	0.10	4.89	0.11	4.89
	Actual-OK*	29.87	65.13	60.17	34.83
	Actual-NG*	0.26	4.74	0.18	4.82
	Actual-OK$^{\sharp}$	29.63	65.37	60.15	34.85
	Actual-NG$^{\sharp}$	0.25	4.75	0.17	4.83
	Actual-OK	95.00	0.00	95.00	0.00
	Actual-NG	5.00	0.00	4.77	0.23
Hazelnut	Actual-OK†	33.37	61.63	62.16	32.84
	Actual-NG†	0.67	4.33	0.36	4.64
	Actual-OK*	34.10	60.90	62.61	32.39
	Actual-NG*	0.78	4.22	0.32	4.68
	Actual-OK*	42.67	52.33	67.10	27.90
	Actual-NG*	1.04	3.96	0.41	4.59
	Actual-OK$^{\sharp}$	42.63	52.37	68.47	26.53
	Actual-NG$^{\sharp}$	0.99	4.01	0.45	4.55
	Actual-OK	95.00	0.00	94.41	0.59
	Actual-NG	5.00	0.00	5.00	0.00

Leather	Actual-OK†	67.93	27.07	42.08	52.92
	Actual-NG†	0.14	4.86	0.01	4.99
	Actual-OK*	68.31	26.69	42.36	52.64
	Actual-NG*	0.10	4.90	0.01	4.99
	Actual-OK*	71.14	23.86	47.69	47.31
	Actual-NG*	0.13	4.87	0.02	4.98
	Actual-OK$^{\sharp}$	71.80	23.20	48.77	46.23
	Actual-NG$^{\sharp}$	0.17	4.83	0.02	4.98
	Actual-OK	95.00	0.00	93.64	1.36
	Actual-NG	4.53	0.47	4.95	0.05
Carpet	Actual-OK†	55.14	39.86	35.37	59.63
	Actual-NG†	0.55	4.45	0.22	4.78
	Actual-OK*	55.16	39.84	35.59	59.41
	Actual-NG*	0.53	4.47	0.23	4.76
	Actual-OK*	63.31	31.69	39.84	55.16
	Actual-NG*	0.59	4.41	0.27	4.73
	Actual-OK$^{\sharp}$	64.87	30.13	40.49	54.51
	Actual-NG$^{\sharp}$	0.61	4.38	0.26	4.74
	Actual-OK	95.00	0.00	93.79	1.21
	Actual-NG	5.00	0.00	5.00	0.00

5
Conclusion

In this thesis, we proposed a data augmentation method in feature space using interpolation and extrapolation. The proposed data augmentation is applied to few-shot learning problems. Most existing few-shot learning methods that address the data deficiency directly, which are called hallucination-based methods, employ a generator network to generate more data. Our approach, however, deals with the problem of data shortage in a simple but effective way without the need to train additional networks.

In our benchmark experiments, we found that our data augmentation can improve few-shot classification accuracy. Our results also show that for the augmentation to work, the amount of generated data should reach a certain level. If the amount of data obtained through augmentation is sufficient, the spreading factor does not seem to have a significant impact on performance. Moreover, we applied a few-shot setting to visual inspection problems using the MVTec-AD dataset, unlike most studies that have only focused on benchmark experiments. For a more realistic scenario, the query set is always assumed to be imbalanced, and we experiment

separately for the situation where the support set is balanced and imbalanced. It turns out that the proposed data augmentation is effective for the imbalanced support set, especially when using the ImageNet dataset for pre-training. However, according to our results, it seems difficult to apply a few-shot learning scheme and our augmentation method to actual visual inspection problems. We thus introduced rough filtering as a less ambitious goal. Experimental results demonstrate that our augmentation method is useful for roughly filtering out non-defective data with the imbalanced support set, which means our approach has potential to help reduce the labeling costs.

Our method of data augmentation via interpolation and extrapolation in feature space is so simple that there must surely be methods that better exploit the structure of the data manifold. However, since the performance of few-shot classification can improve by our approach, interpolation and extrapolation cannot be ruled out just because it is simple. Recent studies on mixup have focused on the validity of intermediate data by adversarial training of a critic network rather than simply applying interpolation. If we add a critic network to our method, it would be important to investigate the trade-off between performance and cost through a lot of experiments because we cannot overcome the disadvantages of the existing hallucination-based methods by introducing a critic network.

Bibliography

[1] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015.

[2] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[3] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[4] 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.

[5] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

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

[8] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

[9] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2, 2018.

[10] 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.

[11] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

[12] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[13] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

[14] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

[15] Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5822–5830, 2018.

[16] Bharath Hariharan and Ross Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3018–3027, 2017.

[17] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7278–7286, 2018.

[18] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[19] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad–a comprehensive real-world dataset for unsupervised anomaly detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9592–9600, 2019.

[20] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[21] Yaqing Wang and Quanming Yao. Few-shot learning: A survey. *arXiv preprint arXiv:1904.05046*, 2019.

[22] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[23] Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Deep transfer metric learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 325–333, 2015.

[24] David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. Understanding and improving interpolation in autoencoders via an adversarial regularizer. *arXiv preprint arXiv:1807.07543*, 2018.

[25] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, Aaron Courville, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. *arXiv preprint arXiv:1806.05236*, 2018.

[26] Gerald Farin and Dianne Hansford. *The essentials of CAGD*. AK Peters/CRC Press, 2000.

[27] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011.

[28] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[29] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[30] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[31] 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.

[32] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

국문초록

지도학습 문제에 딥러닝을 사용할 때, 일반적으로 라벨링된 데이터가 충분히 많아야 좋은 성능을 기대할 수 있다. 하지만 퓨샷 러닝 기법을 통해서라면 아주 적은 양의 라벨링된 데이터로 학습한 딥러닝 모델도 뛰어난 분류 성능을 보일 수 있다. 퓨샷 러닝의 대표적인 방법 중 하나로 이미지 생성 네트워크를 학습하여 데이터의 양을 늘리는 방법이 제시되어왔다. 그러나 이미지 생성 네트워크는 그 자체로 학습하기가 어려울 뿐만 아니라 시간과 메모리 소요가 크다는 단점이 있다. 본 논문에서는 보다 간단하게 특징 공간에서 보간과 외삽을 통해 데이터의 양을 늘리는 방법을 제안한다. 이러한 방식을 사용하면 이미지 생성 네트워크를 추가적으로 학습할 필요도 없으며, 특징 추출 네트워크를 사용하는 방법이라면 어디에든 적용할 수 있다는 장점이 있다. 본 연구에서는 퓨샷 러닝에 쓰이는 대표적인 데이터셋, 그리고 비전 검사용 데이터셋을 이용해 퓨샷 분류 실험을 수행하였다. 결과적으로 특징 공간에서 보간과 외삽을 이용해 데이터의 양을 늘리면 분류 정확도가 상승하는 것을 확인하였다. 또한 정상 데이터에 비해 결함 데이터가 부족하여 문제를 겪는 제조업 현장에서 퓨샷 러닝과 우리의 데이터 증강 방법을 이용한다면 대략적으로 데이터를 필터링하여 라벨링 비용을 줄이는 데 도움이 될 것으로 기대된다.

주요어: 퓨샷 러닝, 데이터 증강, 특징 공간, 보간 및 외삽, 비전 검사

학번: 2016-20714