Computer Science
&
Data Science

Capstone Report - Spring 2022

Few-Shot Learning in Computer Vision

Yumeng Zhang,
Mengjie Shen,
Yuhan Yao

supervised by
Professor Li Guo

Preface

As curious undergraduate students in Computer Science and Data Science, we present this capstone paper to the New York University Shanghai community with the motivation to contribute to the field of few-shot learning. We focus on the subcategory of few-shot classification in computer vision. Few-shot classification methods empower deep learning models to adapt to new classes in a fast and efficient fashion.

In this paper, we not only reproduce the test results from *Boosting Few-Shot Visual Learning with Self-Supervision*, but tune the hyper-parameters in its few-shot learning + self-supervision settings but also propose and implement two new modified models and compare performance with the original paper. One of the modified models successfully surpasses the performance of the baseline model.

All three authors of this paper are NYU Shanghai undergraduate students. Mengjie Shen majors in Computer Science; Yumeng Zhang and Yuhan Yao are both Data Science students with a concentration in AI. Mengjie Shen and Yuhan Yao are studying at the NYU New York campus while Yumeng Zhang is at the Shanghai campus during the capstone project.

Acknowledgements

This research project is supported by New York University Shanghai Data Science senior project and Computer Science capstone programs. We would like to express our deep and sincere gratitude to our research supervisor, Dr. Li Guo, Assistant Professor of Practice in Data Science at NYU Shanghai, for providing us with the opportunity to do research and guiding us every step along the way. We are also grateful to NYU Shanghai for showing community support during the Shanghai lockdown. We would also like to extend our gratitude to our family and friends who have been our consistent support throughout our undergraduate studies.

Abstract

Few-shot classification can quickly adapt to recognizing new classes, given only a few examples of each of these classes. Although existing few-shot algorithms have achieved relatively high performance on simple datasets such as Omniglot, there is still room for improvement on more complicated datasets such as miniImageNet. Therefore, to further improve upon existing algorithms, this capstone project reproduces and compares the experimental results of baseline models, Prototypical Networks and Cosine Classifier, and baseline+self-supervision models, ProtoNet+rotation prediction, and Cosine Classifier+rotation prediction. This project also tunes the hyper-parameters in baseline+rotation settings and proposes two modified models. Our contributions include reproducing and expanding the scope of existing research and proposing a new modified model that surpasses the performance of Prototypical Networks as a baseline.

Keywords

Few-shot Classification; Computer Vision; Prototypical Networks; Cosine Classifier; Self-supervision

Contents

1. Introduction

Human children can learn what a cat is from looking at one or two pictures, while a deep learning model requires tens of thousands of images as training data. This is computationally expensive and lacks scalability to new classes. Therefore, researchers are actively looking for solutions to such fast and flexible learning called few-shot learning. This project focuses on a sub-category of few-shot learning called few-shot classification in computer vision, whose aim is to recognize new image categories with only a few labeled examples in each class.

Solving such a problem requires adaptive classifiers to identify new classes, given only a few examples of each of these classes. Existing approaches such as Matching Networks, Prototypical Networks, and MAML enable the classification functions to learn and adapt quickly, as we humans have the ability to perform one-shot classification, which means learning from only a single example. These approaches also solve real-life limitations such as annotation costs and expand the applicability to emerging images that may never exist before.

Although existing few-shot classification algorithms have achieved relatively high performance on simple datasets such as Omniglot, there is still room for improvement on more complicated datasets such as miniImageNet. Therefore, the objective of this capstone project is to understand and test existing methods and then propose a new-and-improved few-shot learning model.

Our contributions include 1) reproduce and compare the test results of two metric-based methods, Prototypical Networks and Cosine Classifier, and combine them with self-supervised rotation prediction to test their performance improvement on the miniImageNet dataset, 2) tune hyper-parameter α in baseline+rotation settings and explore its effect on the final outcome, and 3) propose and implement two modified models, one of which surpasses the ProtoNet baseline.

2. Related Work

There have been significant advances in few-shot classification in computer vision using the meta-learning framework, which extracts transferable knowledge from a set of tasks while preventing overfitting and improving generalization at the same time. This section first compares and contrasts multiple state-of-the-art few-shot classification methods, two of which serve as the baseline models in the next section. Then, our inspiration for comparing and testing the combination of few-shot learning and self-supervision will be presented in detail.

2.1. Few-shot Classification Methods

Existing few-shot classification methods fall broadly into 2 categories, metric-based and optimization-based approaches.

2.1.1. Metric-based Approaches

The first category is metric-based approaches [1, 2, 3, 4, 5, 6, 7] that focus on learning generalizable embedding functions that extract image features. Cosine Classifier, Matching Networks, and Prototypical Networks are popular examples of this category.

[1] and [2] utilize the cosine similarity function in the softmax operation of the base classes to train the feature extractor. Cosine similarity based classifiers are superior to standard dot-product-based ones because Cosine Classifier learns feature extractors to reduce within-class variation so that new classes can be better generalized.

[3] follows the framework of meta-learning and is the first paper ever to match training and test conditions suitable for one-shot learning. Matching networks first embed a high dimensional sample into a low dimensional space (LSTM) and then perform a generalized form of nearest-neighbors classification using cosine similarity. This paper also creates benchmarks on ImageNet and small-scale language models for other approaches.

[4] tackles all k-shot classifications with k equal to or greater than 0. Prototypical Networks learn a non-linear mapping of the input image into an embedding space using a convolutional neural network and calculate the mean embedding which serves as a single prototype representation for each class. Then, it performs classification for an embedded query image by applying the nearest neighbor algorithm to find the nearest prototype. Notably, the distance function in Prototypical Networks is Euclidean distance, instead of the cosine similarity. Additionally, it can be generalized to zero-shot learning where the only high-level description of a class is required rather than labeled examples.

2.1.2. Optimization-based Approaches

The second category is optimization-based approaches [8, 9, 10, 11] where the classifiers are optimized with the expectation that a future update based on the new test data achieves good results on that task. Model-agnostic meta-learning, or MAML, is a representative of this category.

[8] proposes a gradient-based task-agnostic algorithm that trains a meta-learner's parameters such that a small number of gradient updates lead to fast learning on different new tasks. The

meta-learner performs gradient descent in the direction of each new task separately, but its parameters are updated by the sum of the loss of all new tasks. Other than that, its distance function is consistent with [4] and its feature extractor is the same as [3].

2.2. Self-supervision

Self-supervision [12, 13, 14, 15, 16, 17] is an intermediate form between supervised and unsupervised learning tailored to solve a lack of annotated images. It creates self-supervised tasks from annotation-free images and adapts the representations from self-supervision to actual tasks intended. Typical self-supervised tasks consist of rotation prediction [12], relative patch location prediction [13], partial removal prediction [14], cluster prediction [15], number of objects prediction [16], etc.

The main inspiration of this project is [18], which adds self-supervision loss of rotation prediction or relative patch location prediction to the few-shot classification loss of ProtoNet and Cosine Classifier as an auxiliary term. The experiment result shows an improvement in average accuracy from around 0.5 to 3.5%. [18] also proposes the idea of introducing novel classes into self-supervision, but [19] proves empirically that the performance drops when the image distributions for meta-learning and self-supervised learning are different. That is, no additional data is needed for self-supervision.

This project follows the thoughts of metric-based approaches and combines few-shot learning with self-supervision. Given the plethora of self-supervision tasks, we chose rotation prediction in our implementation because [18] shows that this task achieves the greatest performance improvement. Similarly inspired by [18], our work selects ProtoNet and Cosine Classifier as the methods to modify as well, but we further test the performance of baselines and baselines+rotation with different parameter settings. However, we do not incorporate any novel dataset, as it might hurt the classification accuracy.

3. Solution

3.1. Pre-train vs Meta-train

The feature extractor $F_\theta(\cdot)$, a ConvNet with parameters θ, is the core component of all few-shot methods. The feature extractor generates a d-dimensional feature from a picture x to $F_\theta(x)$. In the few-shot classification task, there are two learning stages, and a robust feature extractor helps

the classification model transfer the knowledge learned from the first learning stage to complete the task in the second stage. More specifically, the first learning stage uses base class images to train the feature extractor and few-shot classifier, and during the second learning stage, the model fine-tunes the classifier with samples from novel classes and performs the few-shot classification task. But please note Prototypical Network has no few-shot classifier but uses class-level average. To train a robust feature extractor during the first stage, two training frameworks are utilized: pre-training and meta-training.

Similar to a lot of researchers, the authors in [18] focus on the first stage of few-shot training and discuss two representative few-shot algorithms corresponding to two different training frameworks, Prototypical Networks (PN) for meta-training framework and Cosine Classifier (CC) for pre-training framework. In [18], the authors mainly discuss the classifier difference between CC and PN, mentioning CC learns base classifiers together with the feature extractor, whereas PN relies solely on class-level averages. However, in this paper, we shed more light on the different training frameworks where PN and CC are utilized respectively.

Inspired by [20], we consider different training frameworks cause the performance difference between PN and CC. Using the pre-training framework, CC generates a more robust feature extractor by utilizing all base classes to train the feature extractor in each episode. Meanwhile, under the meta-training framework, the Prototypical Network only samples N classes each (N as in N-way training). To investigate the two different training frameworks, we conduct experiments using the CC code in [21] and further recreated the Prototypical Network model to conduct comparison experiments. Below are the two training methods we explored:

Prototypical Networks (PN)[4]. The feature extractor $F_\theta(\cdot)$ is trained on sampled few-shot classification sub-problems that are analogs to the targeted few-shot classification issue in the first step of this approach. In other words, during the first stage of ProtoNet training, it conducts multiple few-shot classification tasks episodically. In each episode, the training data is split into Support Set and Query Set, and the goal of each episode is to classify the image from the Query Set based on the knowledge learned from the Support Set. The training data is from C_b base classes; in each episode, a subset C_{Nb} of N base classes is sampled. For each sampled class, we randomly select $k+n$ images to further create Support Set $S_b = \{(x_1, y_1)...(x_k, y_1)...(x_1, y_N)...(x_k, y_N)\}$ with k images each and Query Set $Q_b = \{(x_1, y_1)...(x_n, y_1)...(x_1, y_N)...(x_n, y_N)\}$ with n images each. For each class of the support set, a prototype is calculated using the current feature extractor F_θ

as follows:

$$p_j = \frac{1}{k} \sum_{\mathbf{x} \subset X_j} F_\theta(\mathbf{x}), X_j = \{\mathbf{x} | (x_m, y_n) \subset S_k, n = j\}$$

Classification is then performed for an embedded query point x_q from Q_b by simply finding the nearest class prototype using the following formula:

$$C^j(F_\theta(x_q), S_b) = \text{softmax}_j[d(F_\theta(x_q), p_i)_{i \subset C_{Nb}}]$$

where $d(,)$ is squared Euclidean Distance that calculates similarities between each sample from Query Set and prototypes p_j. As a result, the picture x_q is categorized into its nearest prototype's class. Finally, the initial learning stage entails the episodic drop of the following loss:

$$L_{few} = \mathbb{E}_{S,C_b}[-logC^j(F_\theta(x_q), S_b)],$$

During the second learning stage, the feature extractor $F_\theta(\cdot)$ is frozen and the new classifier is simply defined as $C(; S_n)$.

Cosine Classifiers (CC) [1, 2]. We follow the same CC setting as [18]. The feature extractor $F_\theta(\cdot)$ and the cosine-similarity-based classifier are trained on the supervised task of identifying the base classes in the first stage of learning. Different from Prototypical Network training, CC samples from the entire base classes C_b in each episode. The normalized score for an input image $\mathbf{x}$ is represented by $W_b = [w_1, ..., w_{Nb}]$, which is the matrix of the d-dimensional classification weight vectors. The weight vectors are used to generate a normalized score that reads:

$$C^j(F_\theta(\mathbf{x}), W_b) = \text{softmax}_j[\gamma cos(F_\theta(\mathbf{x}), w_i)_{i \subset b}]$$

where $cos(,)$ calculates the cosine similarity between two vectors, and γ is the inverse temperature parameter of the softmax operator. Having these normalized scores, we train the CC classifier and feature extractor by calculating the negative log-likelihood loss:

$$L_{few} = \mathbb{E}_{(x,y) \subset C_b}[-logC^y(F_\theta(\mathbf{x}), W_b)],$$

The second learning stage is similar to that in PN which involves computing one representative feature for each new class by averaging associated K samples in the sampled class and defining

the final classifier in the same way.

3.2. Self-supervision as Auxiliary Loss

[18] proposes adding a self-supervision loss to the few-shot learning loss as an auxiliary term. Figure 1 outlines the corresponding model architecture.

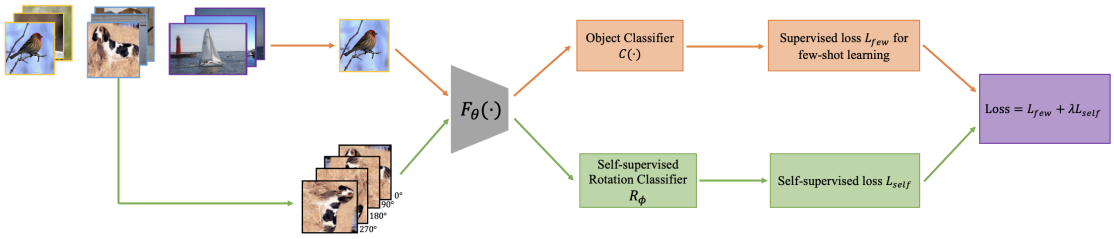

Figure 1: Self-supervision as Auxiliary Loss

The first training stage requires updating the parameters of feature extractor $F_\theta(\cdot)$ with both few-shot learning and self-supervised learning. During few-shot learning, we use the labeled images to perform few-shot classification and its loss function is

$$L_{few} = \mathbb{E}[-logC(F_\theta(x))],$$

where x is the training image and $F_\theta(x)$ is the feature embedding. The detailed calculation of the few-shot classification loss differs depending on whether the experiment uses a CC or a PN, as elaborated in Section 3.1.

The self-supervised learning involves image rotation prediction as our self-supervision task with the same dataset. These images are randomly sampled from the labeled images. Given an image x, we need to first rotate it four times and save the images in $\{x^r | r \in R\}$ where $R = \{0°, 90°, 180°, 270°\}$. Then, the self-supervised rotation classifier R_ϕ predicts the rotation degree r according to the feature embedding $F_\theta(x)$. The loss function is

$$L_{self} = \mathbb{E}[\sum_{\forall r \in R} -logR_\phi^r(F_\theta(x^r))],$$

where $R_\phi^r(\cdot)$ is the predicted score for rotation degree r.

The final loss is the weighted sum of the few-shot loss and the self-supervision loss. Therefore,

the model aims to

$$\min_{\theta,\phi} Loss = L_{few} + \alpha L_{self}.$$

[18] uses $\alpha = 1$ for all experiments, which means that few-shot classification and self-supervision rotation tasks are equally weighted. But in this capstone project, we experiment with $\alpha = 0, 0.5, 1$ to not only recreate tests in [18] but also extend the scope of experimentation. Please note that if $\alpha = 0$, we have $Loss = L_{few}$, which is the same as testing baseline models. Therefore, controlling the value of α allows us to test baseline versus baseline+rotation and rotation with different weights.

3.3. Modified Models

[22] proposes the idea of combining features extracted from different feature extractors. It suggests that combining a library of pre-trained feature extractors with a basic feed-forward network learned with an L2-regularizer can be a good way to solve cross-domain few-shot image categorization. Thus, following this idea, our model architecture introduces another feature extractor and then combines the feature inputs.

3.3.1. Modified Model 1.0

It is worth noticing that the number and sampling method of the input images are the same as our original model. During the first stage of ProtoNet learning, it still conducts multiple few-shot classification tasks episodically. In the first learning stage, the training data is from base classes C_b; in the second learning stage, the training data is from novel classes C_n.

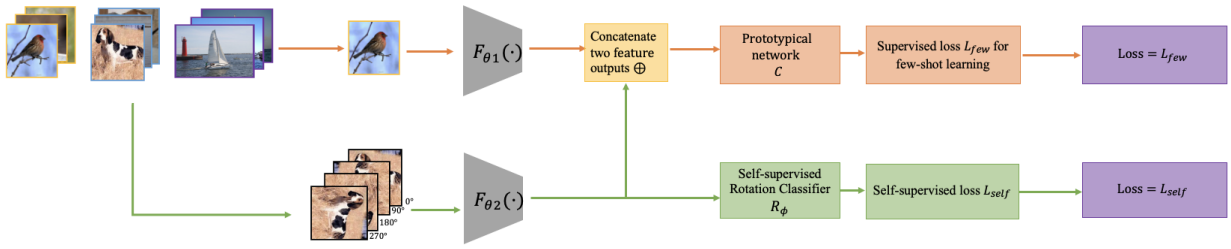

Figure 2: Modified Network Architecture 1.0

According to Figure 2, we have two feature extractors parallel to each other in the first learning

stage. These two sets of ConvNets have parameters θ_1 and θ_2 respectively. We train the feature extractor $F_{\theta 1}(\cdot)$ and the $F_{\theta 2}(\cdot)$ using the same data input (both original images and rotated images) in a multi-task setting. The output of both feature extractors $F_{\theta 1}$ and $F_{\theta 2}$ is two d-dimensional features. In the top branch, we concatenate these two features together by their channel and input them into the Prototypical Networks. Therefore, the current formula for computing the prototypes has turned into:

$$p_j = \frac{1}{k} \sum_{\mathbf{x} \subset X_j} [F_{\theta_1}(\mathbf{x}) \oplus F_{\theta_2}(\mathbf{x})],$$

where $X_j = \{\mathbf{x} | (x_m, y_n) \subset S_k, n = j\}$.

We still use Euclidean distance to calculate similarities between each sample from Query Set and prototypes p_j and we adopt a softmax classifier for classification. For each new image x_q from Q_b in each class j, we calculate the cross-entropy loss after outputting from the prototypical network. What's different is that the few-shot classification loss L_{few} is used to train the feature extractor $F_{\theta 1}(\cdot)$.

$$L_{few} = \mathbb{E}_{S,C_b}[-\log C^j([F_{\theta_1}(x_q), F_{\theta_2}(x_q)], S_b)],$$

For the self-supervised task, the process is very much similar to the original one. We sample images from the annotated set and we generate four rotations for each input image, process them with $F_{\theta 2}$ and train the rotation classifier R_ϕ with the self-supervised loss L_{self}. What's different is that the loss L_{self} is only used to update $F_{\theta 2}$ and R_ϕ.

In the second learning stage, the feature extractor $F_{\theta 1}$ and $F_{\theta 2}$ are both frozen, and the classifier of novel classes is based on the output of the Prototypical Networks with combined features input.

3.3.2. Modified Model 2.0

After running experiments with model 1.0, we find the problem that the convergence speeds for the two feature extractors are different. This leads to the result that one feature extractor is far from convergence while the other one has already been overfitted. Therefore, we propose the Modified Model 2.0, as shown in Figure 3. The difference from model 1.0 is that instead of using the losses L_{few} and L_{self} to update separately. We add up the loss with the weight assigned and then use the total loss to update the parameters in both feature extractors $F_{\theta 1}$, $F_{\theta 2}$, and the rotation classifier R_ϕ. Using a single loss to calculate the gradient and update parameters can effectively prevent the problem of different convergence speeds.

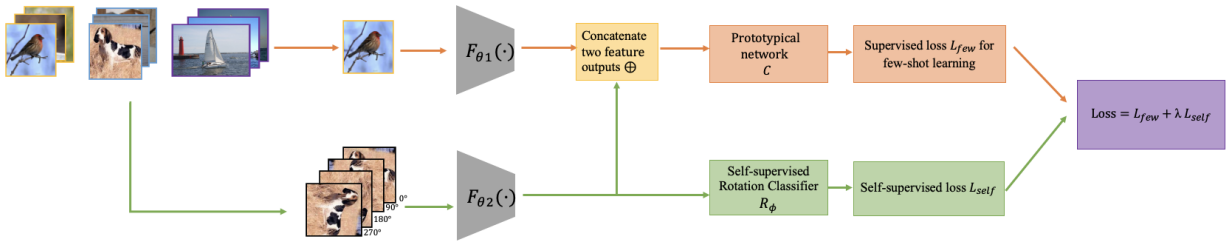

Figure 3: Modified Network Architecture 2.0

4. Experimental Results

This section presents the training performance difference between pre-training and meta-training models as well as their performance improvement after adding self-supervision as auxiliary loss in 4.1 and 4.2. Furthermore, we adjust the coefficient of the auxiliary loss and compare the performance between different model versions in 4.3. Finally, we modify the ProtoNet+Rotation model by introducing an extra feature extractor that is only updated by the auxiliary loss in 4.4. The codebase of the experiments is in GitHub repository [23].

Dataset. The dataset used in this project is miniImageNet [24], which consists of 100 classes randomly sampled from ImageNet. And in this project, we follow the setting of [18], there are 64 base classes, 16 validation classes, and 20 novel test classes.

Model evaluation. Few-shot classifications are evaluated based on the classification during the second stage. For each test, we used 2000 episodes and each episode is an N-way K-shot few-shot classification task of the novel class images. In this project, we set all the tasks to be 5-way few-shot classification tasks and conducted experiments on 5-shot and 1-shot respectively. Additionally, during the second stage, we sample 15 test images from each novel class to construct the Query Set. To sum up, the classification accuracy is measured on the 5x15 test images and is averaged over all the sampled few-shot tasks and all the experiments follow this setting.

Implementation Details. In this work, we use WRN-28-10 [25] as the backbone network of the feature extractor. We use mini-batch stochastic gradient descent (SGD) to optimize the training loss and we use both recognition L_{few} and self-supervised L_{self} losses on the labeled data.

4.1. Baseline Model Comparison

ProtoNet VS Cosine Classifier. We first compare the training result between the meta-training baseline algorithm Prototypical Network (PN) to the pre-training baseline model Cosine Classifier (CC). We perform this study using the miniImageNet dataset and report our comparison results in Table 1. For the PN case, based on a 5-way few-shot setting, in the first learning stage, we sample 5 images from 5 randomly selected base classes in every training episode. In comparison, for CC, we sample all base classes in a total of 64 images in each episode. Note that both models have been taught to detect all four rotated variations of an image. The justification for employing these baselines is that the model "sees" the same type of data as the model with rotation prediction self-supervision. Similarly, during the second stage, we also apply rotation augmentations.

Model	K-Shots	Accuracy	Prior Work Accuracy
PN	1-shot	50.53 ± 0.46%	55.85 ± 0.48%
CC	1-shot	62.05 ± 0.46%	61.09 ± 0.44%
PN	5-shot	65.63 ± 0.38%	68.72 ± 0.36%
CC	5-shot	79.45 ± 0.32%	78.43 ± 0.33%

Table 1: **Comparison of Baseline Models and Prior Work.** Average 5-way classification accuracies for the novel classes with 95% confidence intervals (using 2000 episodes)

Here we can find that both in our experiment and in the original paper, the pre-training model CC outperforms the meta-training model PN. We believe this result proves that the pre-training model by seeing more categories contribute to training a more robust feature extractor $F_\theta(\cdot)$. Additionally, the performance dropped for our PN model compared to [18]. It is because this work utilizes rotation augmentations to align with the input data of the CC algorithm for a fair comparison. However, [18] uses non-augmented images for ProtoNet. As a result, this performance degradation is expected and has also been mentioned in [18]. Furthermore, this performance degradation does not affect our conclusion that the pre-training model generates better results than the meta-training model under this setting.

4.2. Self-supervision as auxiliary loss function

We further study the impact of adding rotation prediction as self-supervision to PN and CC respectively. We train the model using the same setting as for baseline models but added auxiliary loss of rotation prediction task aiming to boost model performance by minimizing the images' intra-class difference of extracted features. The results in Table 2 demonstrate that adding rota-

tion prediction into a few-shot classification task indeed improves the performance of recognizing novel classes during the second stage.

Model	K-Shots	Accuracy	Prior Work Accuracy
PN	1-shot	50.53±0.46%	55.85±0.48%
PN+rot	1-shot	**56.82±0.47%**	**58.28±0.49%**
CC	1-shot	62.05 ± 0.46%	61.09 ± 0.44%
CC+rot	1-shot	**62.81±0.46%**	**62.93±0.45%**
PN	5-shot	65.63±0.38%	68.72±0.36%
PN+rot	5-shot	**71.93±0.38%**	**72.13±0.38%**
CC	5-shot	79.45±0.32%	78.43±0.33%
CC+rot	5-shot	**80.00±0.39%**	**79.87±0.33%**

Table 2: **Rotation prediction as auxiliary loss.** Average 5-way classification accuracies for the novel classes with 95% confidence intervals (using 2000 episodes).

And this result is consistent with both meta-training and pre-training models. Whereas, clearly the performance improvement is more significant in terms of the PN with auxiliary loss when compared to the CC model, which further justifies the contribution of 'seeing' more classes boost experiments' results during the first training stage.

4.3. Adjusting Hyper-parameter Alpha

In this section, we focus on adjusting the hyper-parameter in the model architecture shown in figure 1, with either Cosine Classifier or Prototypical Network as the Object Classifier C. We focus on the hyper-parameter α which occurs in the loss function:

$$\min_{\theta,\phi} Loss = L_{few} + \alpha L_{self}.$$

α is the weight for the self-rotation loss. It determines the influence of self-rotation in the model. With $\alpha = 0$ means that it's a simple few-shot classification model without self-supervision playing any part. We set α to be 0, 0.5, and 1.0 and run the test in 1-shot and 5-shot cases with average 5-way classification accuracies for the novel classes with 95% confidence intervals (using 2000 episodes).

From the table 3, we can observe that there is not much of a difference for the pre-trained model Cosine Classifier that [18] proposes. However, for the meta-trained model Prototypical Network, a larger α achieves better results for both 1-shot scenario and 5-shot scenario.

Model	WaysShots	Alpha	Accuracy
CC + rot	1-shot	0	62.05±0.46%
CC + rot	1-shot	0.5	62.90±0.45%
CC + rot	1-shot	1.0	62.81±0.46%
CC + rot	5-shot	0	79.45±0.32%
CC + rot	5-shot	0.5	80.01±0.33%
CC + rot	5-shot	1.0	80.00±0.39%
PN + rot	1-shot	0	50.53±0.46%
PN + rot	1-shot	0.5	54.70±0.47%
PN + rot	1-shot	1.0	56.82±0.47%
PN + rot	5-shot	0	65.63±0.38%
PN + rot	5-shot	0.5	70.06±0.39%
PN + rot	5-shot	1.0	71.93±0.38%

Table 3: **Comparison of choice of alpha between models.**Average 5-way classification accuracies for the novel classes with 95% confidence intervals (using 2000 episodes).

4.4. Modified models

In the last section, we test the performance of the two model structures we proposed. We set the hyper-parameter α to be 1.0 and test the model in the 1-shot scenario and 5-shot scenario. Table 4 shows the result of our experiments. Original Model refers to the model architecture in Figure1, with the rotation classifier for the self-supervision task and prototypical network for the object classification task. Modified Model 1.0 refers to the model architecture in section 3.3.1 and Modified Model 2.0 refers to the model architecture in section 3.3.2.

Model	WaysShots	Alpha	Accuracy
Original Model(PN)	1-shot	1.0	50.53±0.46%
Modified Model 1.0	1-shot	1.0	46.00±0.45%
Modified Model 2.0	1-shot	1.0	53.82±0.47%
Original Model(PN + rot)	1-shot	1.0	56.82±0.47%
Original Model(PN)	5-shot	1.0	65.63±0.38%
Modified Model 1.0	5-shot	1.0	59.10±0.37%
Modified Model 2.0	5-shot	1.0	68.42±0.40%
Original Model(PN + rot)	5-shot	1.0	71.93±0.38%

Table 4: **Comparison of the performance between our proposed modified models and the original model.**Average 5-way classification accuracies for the novel classes with 95% confidence intervals (using 2000 episodes).

From table 4, we can infer that the original model still outperforms the model architectures we proposed. Our modified models' performance turned out to be worse than we expected. The potential reason might be the insufficient training of the feature extractors. The original paper [22], which practices the combination of multiple feature extractors, uses a large dataset to train its parameters. Traditionally, the model uses a large number of unlabelled data to conduct

unsupervised learning.

However, in our few-shot scenario, the number of input images is limited. The small sample size could make the training of the feature extractors less efficient. Since we are not sure about the quality of the feature extractors, it might be problematic when we try to concatenate the features before we input them into the prototypical network.

Our modified model 2.0 performs better than model 1.0. As we mentioned earlier in section 3.3.2, our model 1.0 might face the issue that the convergence rates for two feature extractors are different, since we are updating the parameters of two feature extractors separately using different losses. However, in model 2.0, we aligned the convergence rate by using a weighted total loss. The testing result also suggests that we have effectively solved the problem of unmatched convergence rate for different feature extractors.

5. Discussion

Some of the challenges and issues we encounter are:

1. Few-shot learning is a rapidly developed field where there exist many new ideas and new directions to pursue, so it took us a while to understand the mechanisms of the state-of-the-art models and narrow down to one direction to dig deeper: few-shot classification + self-supervision.

2. The GitHub repository [21] from the main inspiration of this research project [18] is poorly documented. The authors fail to explain many implementation details and do not provide proper code documentation for their ProtoNet implementation, which complicates our experiments for the original paper.

3. Since this research project involves a lot of computationally expensive experimentation, the GPU hardware limit renders our experimentation process incredibly difficult. However, we manage to use Google Colab instead of local machines or NYUSH HPC.

We overcome these obstacles and conduct successful experiments on 2 baseline models, 2 baseline+rotation models, and 2 modified models, each with 5-way 1-shot and 5-way 5-shot settings respectively. Through these experiments, this paper

1. reproduces two baseline models, Cosine Classifier as a representative of pre-training algorithms and Prototypical Networks as a representative of meta-training algorithms, and

concludes that CC outperforms PN as a baseline;

2. reproduces two baseline+rotation models, CC+rotation and PN+rotation, and concludes that adding self-supervised rotation learning achieves higher accuracy than baseline models and that the performance improvement is higher on PN, rather than CC;

3. tunes the hyper-parameter α and concludes that a bigger α that is closer to 1 allows PN+rotation to achieve better performance but different α values have a homogeneous effect on the performance improvement on CC;

4. proposes and implements two modified models and successfully creates Modified Model 2.0, which surpasses the performance of ProtoNet baseline;

5. compares two modified models and concludes that Modified Model 2.0 outperforms 1.0 because using one weighted sum of losses to update two ConvNets, instead of two separate losses, is more conducive for the convergence of both networks.

Although Modified Model 2.0 outperforms the ProtoNet baseline, the baseline+rotation proposed in [18] remains on top in all experiments conducted. This indicates that training one ConvNet instead of two increases the classification accuracy, so there are limitations to our approach. [18] already proposes a model that mitigates the limitations, but further improvement is still left desired. Some future work may include:

1. Experiment on combining other self-supervision benchmarks with few-shot learning and compare their respective performance in improving few-shot classification task performance.

2. Add large scale non-annotated images to the input data of self-supervised network training. This expects to improve the extracted features' quality that might have better potential to boost the few-shot classification performance.

3. Conduct more experiments on different datasets like CUB which is a fine-grained dataset and run experiments under domain shift problem setting. For example, we can train the model with base classes from miniImageNet and test performance using novel classes from CUB.

6. Conclusion

We are inspired by the idea that an auxiliary loss based on self-supervision can boost the result of few-shot recognition models. Given the experiment results using the pre-training framework Cosine Classifier as the object classifier, we further recreate the meta-training Prototypical Network model to investigate the two different training frameworks and conduct comparison experiments on miniImagenet. We find that adding self-supervision to the meta-training framework also leads to significant improvements in the few-shot classification performance. Furthermore, we adjust the hyper-parameters in the model to see whether it makes any changes to the results. Our experiments show that adjusting the weight parameter does not make any significant difference for the pre-trained model, but influences the model performance in the meta-train framework. Finally, we propose two modified models, inspired by the idea that concatenating features from different feature extractors could achieve better performance. Our results show that concatenating features in the few-shot classification scenario improves upon the ProtoNet baseline but is not effective to improve upon the baseline+rotation classification performance. In the future, we can further research combining other self-supervision benchmarks into the model and add large-scale non-annotated images to input, which can potentially improve the quality of training feature extractors.

References

[1] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4367–4375.

[2] H. Qi, M. Brown, and D. G. Lowe, "Low-shot learning with imprinted weights," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5822–5830.

[3] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

[4] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," *Advances in neural information processing systems*, vol. 30, 2017.

[5] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2. Lille, 2015, p. 0.

[6] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1199–1208.

[7] Y. Liu, J. Lee, M. Park, S. Kim, E. Yang, S. J. Hwang, and Y. Yang, "Learning to propagate labels: Transductive propagation network for few-shot learning," *arXiv preprint arXiv:1805.10002*, 2018.

[8] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[9] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," 2016.

[10] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-learning with latent embedding optimization," *arXiv preprint arXiv:1807.05960*, 2018.

[11] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," *arXiv preprint arXiv:1707.09835*, 2017.

[12] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," *arXiv preprint arXiv:1803.07728*, 2018.

[13] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1422–1430.

[14] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," in *European conference on computer vision*. Springer, 2016, pp. 577–593.

[15] M. Caron, P. Bojanowski, J. Mairal, and A. Joulin, "Unsupervised pre-training of image features on non-curated data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2959–2968.

[16] M. Noroozi, H. Pirsiavash, and P. Favaro, "Representation learning by learning to count," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5898–5906.

[17] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *European conference on computer vision*. Springer, 2016, pp. 69–84.

[18] S. Gidaris, A. Bursuc, N. Komodakis, P. Pérez, and M. Cord, "Boosting few-shot visual learning with self-supervision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 8059–8068.

[19] J.-C. Su, S. Maji, and B. Hariharan, "When does self-supervision improve few-shot learning?" in *European Conference on Computer Vision*. Springer, 2020, pp. 645–666.

[20] C.-Y. Chen, H.-T. Lin, G. Niu, and M. Sugiyama, "On the role of pre-training for meta few-shot learning," 2021. [Online]. Available: https://openreview.net/forum?id=TwkEGci1Y-

[21] valeoai, "Bf3s," 2019. [Online]. Available: https://github.com/valeoai/BF3S

[22] A. Chowdhury, M. Jiang, S. Chaudhuri, and C. Jermaine, "Few-shot image classification: Just use a library of pre-trained feature extractors and a simple classifier," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 9445–9454.

[23] AlisonYao, "few-shot-learning-cv," 2022. [Online]. Available: https://github.com/AlisonYao/few-shot-learning-cv

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[25] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

A. Appendix

Model	WaysShots	Alpha	Accuracy
Original Model(CC)	1-shot	0.0	62.05±0.46%
Original Model(CC)	5-shot	0.0	79.45±0.32%
Original Model(CC+rot)	1-shot	0.5	62.90±0.45%
Original Model(CC+rot)	5-shot	0.5	80.01±0.33%
Original Model(CC+rot)	1-shot	1.0	62.81±0.46%
Original Model(CC+rot)	5-shot	1.0	80.00±0.39%
Original Model(PN)	1-shot	0.0	50.53±0.46%
Original Model(PN)	5-shot	0.0	65.63±0.38%
Original Model(PN+rot)	1-shot	0.5	54.70±0.47%
Original Model(PN+rot)	5-shot	0.5	70.06±0.39%
Original Model(PN+rot)	1-shot	1.0	56.82±0.47%
Original Model(PN+rot)	5-shot	1.0	71.93±0.38%
Modified Model 1.0	1-shot	1.0	46.00±0.45%
Modified Model 1.0	5-shot	1.0	59.10±0.37%
Modified Model 2.0	1-shot	1.0	53.82±0.47%
Modified Model 2.0	5-shot	1.0	68.42±0.40%

Table 5: **Test results of all experiments conducted.** Average 5-way classification accuracies for the novel classes with 95% confidence intervals (using 2000 episodes).