COMP 691 - Project

Waleed Afandi(40243372), Omij Mangukiya(40233479), Niki Monjazeb(27775938)

April 30, 2023

1 Introduction

In this report, we present our project on the challenging problem of image classification with very few samples or a small dataset. We have two different problem setups. In challenge 1, we cannot use pre-trained models or other datasets, while in challenge 2, we have the flexibility to use additional data sources beyond CIFAR-10.

For challenge 1, we started with a basic CNN and applied data augmentation techniques to improve the model's performance. We then explored more advanced techniques, including EvoDCNN, which involves finding the optimal architecture, as well as relational networks and prototypical networks, which leverage image similarity and class prototypes, respectively. We experimented with various hyperparameters and thoroughly evaluated each technique.

In challenge 2, we investigated the use of meta-learning and unsupervised learning approaches for image classification. Specifically, we explored the Model-Agnostic Meta-Learning (MAML) algorithm for meta-learning, and the SwAV (Unsupervised Learning of Visual Features by Contrasting Cluster Assignments) model for unsupervised learning. We implemented these models based on their original papers and evaluated their performance in our project setup.

All code can be found from here [8].

2 Literature review

The significance of few-shot learning lies in its ability to learn from only a limited amount of data, which makes it a critical area of research in the field of machine learning. In this literature review, we will explore models that can be adapted for few-shot learning, as well as models that can incorporate external data to enhance the performance of few-shot learning.

2.1 EvoDCNN[5]

Developing a Deep Convoluted Neural Network (DCNN) for image classification is a very dataset specific task as an effective network will be tuned with respect to the dataset that it is used on. Tuning the hyperparameters of a model is a time and resource consuming matter. This paper presents the implementation of Genetic Algorithm (GA) to developing a DCNN. The GA is used to evolve the network and find the best hyperparameters. 17 parameters are being tuned across 3 blocks. The convolution block contains hyperparameters: number of blocks, number of convolution layers, filter size, number of filters, dropout, pooling, activation function, short connection, and batch normalization. The classification block contains hyperparameters: type of layer, number of nodes, batch normalization, activation function, and dropout. The training block contains the optimizer, learning rate, batch normalization, and initialization.

The GA contains four staps: initialization, selection, crossover, and mutation. During the initialization, the hyperparameters for the networks are selected randomly. The Roulette wheel selection model is then used to select networks that will be used for crossover and mutation. Crossover mixes hyperparameters from two or more networks to create new networks. Mutation creates a random change in one or more of the hyperparameters of a model. This evolution can continue for as many generations as needed, in the aim of finding the best hyperparameters.

The model is evaluated on eight datasets including CIFAR10, MNIST, and six versions of EMNIST. All datasets showed an improvement in their accuracy scores as the number of generations increased, with five having error rates lower

than the state-of-the-art.

2.2 Relational Networks[6]

The field of few-shot learning has gained significant attention in the research community in recent years, as it aims to solve one of the biggest issues in the domain of machine learning which is the unavailability of huge datasets required by traditional machine learning and deep learning models. In this paper, the authors propose a new approach to few-shot learning based on Relation Networks (RNs). The authors introduce a new architecture and compare it against several methods on a variety of datasets.

The paper proposes a novel architecture called Relation Networks (RNs) for few-shot learning. RNs utilize a set of learnable transformation functions that take pairs of feature vectors as inputs and output a score representing the relationship between them. These functions can be thought of as a learnable similarity metric, allowing the network to compare different input examples and learn to recognize patterns. The RN architecture consists of multiple stages, each of which applies a set of these transformation functions to the input features. The final output of the network is a classification score based on the transformed features.

The authors evaluate the effectiveness of their proposed method on a range of few-shot learning benchmarks, including *Omniglot*, *mini-ImageNet*. They compare RNs against several other few-shot learning methods, including *Matching Networks*, and *Prototypical Networks*.

2.3 MAML[1]

Model Agnostic Meta Learning (MAML) aims to learn a good initialization of model parameters that can be quickly adapted to new tasks with only a few examples. This paper answers three main questions regarding MAML: (1) What is the effect of the depth of a model? (2) What is the role of the model's heads in updating the bottom layers of an MAML? (3) How do the additional parameters introduced by preconditioning methods help the original method that does not contain these parameters?

The depth of a model is deemed important as its upper layers to control the lower layers's parameter updates. This can be done by increasing the depth of the network, adding layers at the output of the model, or using preconditioning methods. To explore the effects of a model's depth and linear layers, a baseline of a CNN with 4 convolutional layers is considered. This model is considered a failure scenario, meaning that it will fail to meta-learn. The depth of the model is then increased until the model becomes meta-learnable. The same principle is applied to the number of linear layers applied to the output. Layers are added to the failure scenario until it becomes meta-learnable.

Preconditioning techniques aimed at transforming models' gradients, such as First-Order Approximation are also known to make models more meta-learnable. This paper's take on preconditioning is the algorithm META-KFO. This algorithm consists of a neural network that learns to transform the gradient on models that would otherwise not be able to meta-learn, without changing the base of the model's modeling capacity.

META-KFO is tested along with MAML on Omniglot, CIFAR-FS, and mini-Imagenet datasets and shows a better performance than other meta-optimizers on both the Omniglot and CIFAR-FS datasets. However, as the number of layers of a model is increased, the accuracy score of the MAML model catches up with the MAML+KFO model. This suggests that increaseing the depth has a similar effect to adding a meta-optimizer on the meta-learning of a model.

2.4 SWaV: Unsupervised learning of visual features by contrasting cluster assignments [2]

Introduction: Self-supervised learning has become a popular technique for learning visual features without the need for labeled data. SWaV is a novel unsupervised learning approach for visual feature learning that utilizes clustering, optimal transport, and sinkhorn's theorem. In this review, we discuss the SWaV approach and its advantages over other self-supervised learning techniques.

Approach: SWaV replaces the supervised pre-training of CNN with unsupervised CNN by leveraging the property of images. The property of images that is used is that semantic informa-

tion is contained within an image, as well as its different versions, distortions, and crops. The approach uses clustering to predict the group assignment of a distortion from another distortion of the same image.

Clustering: Clustering is used in SWaV because computing all pairwise comparisons on a large dataset is not practical, and most implementations approximate the loss by reducing the number of comparisons to random subsets. However, to cluster images, a full forward pass on the entire dataset is needed, which is intractable. Therefore, the approach uses online clustering to solve this problem.

Optimal Transport: The SWaV approach enforces the constraint that all prototypes should be equally representative. Without this constraint, the network would cheat and assign all features to one prototype, making it too easy to predict the prototype. The problem of assigning features to prototypes can be solved using the sinkhorn-knopp algorithm in optimal transport. The solution in the algorithm is an ascendant matrix, which assigns a feature to a vector of probabilities of belonging to each prototype.

Soft Clustering: The vector of probabilities is then rounded using the highest probability in the vector, which is called soft clustering. Sinkhorn-knopp algorithm works on the basis of a matrix A (n x n) with strictly positive elements. There exist two diagonal matrices D1 and D2 with strictly positive elements such that D1 x A x D2 becomes doubly stochastic.

Multi-crop: The SWaV approach also utilizes the idea of multi-crop, which came from the self-supervised learning of pretext-invariant representation. The idea is that the improvement in the SWaV approach was not due to the jigsaw puzzle task, but rather due to the data augmentations used in the jigsaw objective. The approach aims to do what has been done in supervised augmentation but needs to be done in unsupervised augmentation.

Comparison with other approaches: SWaV has advantages over other self-supervised learning approaches. Cluster-based approaches are generally offline and require a full pass on the data. Methods based on noise contrastive estimation generally calculate contrastive loss using strong data augmentation techniques. These techniques become computationally expensive for large datasets. Additionally, these models maintain a large memory bank, which introduces computational expense.

Conclusion: In conclusion, SWaV is a novel approach for self-supervised visual feature learning that utilizes clustering, optimal transport, and sinkhorn's theorem. The approach is advantageous over other self-supervised learning approaches as it is online, does not require strong data augmentation techniques, and does not maintain a large memory bank. The approach could potentially be useful in downstream tasks such as classification, where inter-image similarities are important.

3 Methods applied for Challenge 1

3.1 EvoDCNN

3.1.1 Implementation

EvoDCNN's implementation has been modified to better fit the requirements of the project. The focus of the experiments below have been on the convolutions and training blocks as the training and testing functions in the original code were to remain intact. The function select_best_model splits the training set into training-train and training-test sets and uses the training-train set to train the various hyperparameter combinations which are then tested on the training-test set. The hyperparameter combination with the best performance in the training-test set is chosen as the best parameter and used to train the entirety of the training set before being tested on the test set.

The function *select_best_model* uses the function *generate_random_population* to generate a random combination of hyperparameters. The number of hyperparameters generated is called the population. All members of the population are evaluated via the function *evaluate_population* where for each hyperparameter, the model is set up, trained, and tested. After each generation, crossover and mutation are applied to the members based on their accuracy results, creating a new population. This cycle is repeated for as many generations as needed.

Epoch	Accuracy
5	64.44+-7.07
10	64.44+-7.07
20	64.44+-7.07

Table 1: Accuracy with changes in Epochs with Population = 30, Generation = 2

Generation	Accuracy
1	68.00 +- 6.65
2	67.06 +- 11.25
5	62.38 +- 4.03

Table 2: Accuracy with changes in Generation with Train epoch = 5, Population = 20

3.1.2 Results

This model is very rich in variability, however, several variations were left untested in the original paper: The range of hyperparameters used, the number of generations, the number of members at each generation, and the number of training epochs. The results for these tests can be seen in Tables 3.1.2, 3.1.2, 3.1.2. It's observed that the accuracy increases with an increase in the population but decreases with an increase in the number of generations. This could be due to overfitting for the training set. No changes in the accuracy is observed as the training accuracy increases.

3.2 Relational Networks

3.2.1 Implementation

As per the requirements of the project, the implementation of the Relational Networks (RNs) has been slightly modified to suit the testing bed. In the original paper [6] the training sample first goes through a convolutional neural network (CNN) encoder, which creates embeddings for the sample, and then another model is trained on these embeddings to identify the proper class type.

While this approach has been followed in this project, we thought it would be interesting to train RN on raw images (without using CNN Encoder and embeddings). Therefore, in our setting, we had to make some changes to the original implementation to make it work with our data. In this modified version, the input samples are fed directly into the RNs, and the RNs learn to extract relevant features and classify the samples based on these features. We made this modification as our dataset was relatively small and simple, and we wanted to explore the effectiveness of the RNs without adding the additional complexity of the CNN encoder.

Population	Accuracy
5	69.00 +- 6.11
10	66.88 +- 3.77
15	67.69 +- 3.61
20	68.00 +- 6.65

Table 3: Accuracy with changes in Population with Train epoch = 5, Generation = 1

In the case of the CNNE+RN implementation and standalone RN implementation, we have fine-tuned the hyperparameters to achieve the best possible results.

3.2.2 Model Architecture

CNN Encoder The first model is a CNN encoder, which is designed to take in images with 3 color channels and produce a 64-dimensional feature vector as output. The architecture consists of 4 convolutional layers with batch normalization, ReLU activation, and max pooling. The first three layers have a kernel size of 3 and a padding of 1, while the last layer has a padding of 1 but no max pooling. The output feature vector has a size of 64.

Relational Network (RN) The second model is a neural network that takes in the 64-dimensional feature vector produced by the CNN encoder and produces a classification output. The architecture consists of 2 convolutional layers with batch normalization, ReLU activation, and max pooling, followed by a fully connected layer with a ReLU activation function. The output of the fully connected layer is passed through a sigmoid activation function to produce a probability distribution over the 10 possible classes. The input size and hidden size of the fully connected layer are configurable parameters of the model.

3.2.3 Experiments

All experiments were conducted using a fixed epoch size. Specifically, we set the default epoch number to 100 and conducted tests based on this fixed value.

Dimension of Hidden Layers In the initial experiment, our aim is to determine the most suitable dimension for the hidden layers.

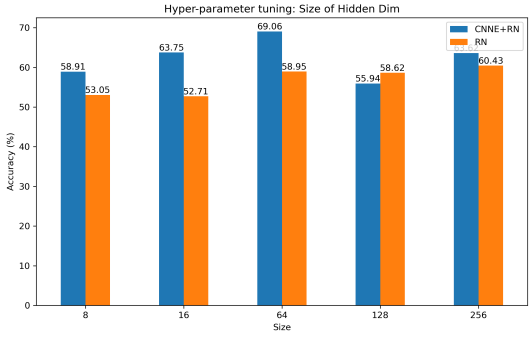

Figure 1: Optimizing hidden layers

From the experiments in Figure 1 , we can observe that for CNNE+RN the best Hidden Dim is 64 whereas for RN 256 seems to provide the best results. However the accuracy with 128 and 64 dimensions are close in case of RN.

Optimizer In this experiment, we aim to compare the performance of Adam and Stochastic Gradient Descent (SGD) optimizers in our setting. Our experiments in Figure 2 indicate that for both models, SGD yields better results. In the original paper, the authors had used the Adam optimizer.

Learning Rate We conduct experiments to evaluate the performance of the models under different learning rates. We tried increasing and decreasing the learning rate from the baseline 0.01. Our findings in Figure 3 indicate that a learning rate of 0.01 yielded the most favorable results for both models.

Batch Size Experimenting with batch size is an important part of optimizing deep learning

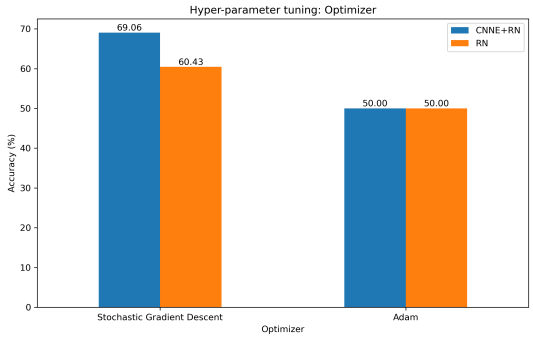

Figure 2: Comparing Optimizers

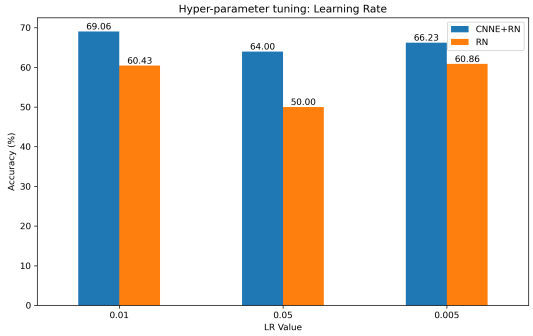

Figure 3: Adjusting Learning Rate

models. Batch size refers to the number of training samples used in one iteration of gradient descent. The experiments in Figure 4 show that setting the batch size to 32 provides best results for both models.

Additional Sequential Layers Finally, we added additional sequential layers to see if it has any impact on the model's accuracy. From the results in Figure 5, we observe that in CNNE + RNN, the baseline configuration works the best. However, for the standalone RN, adding two additional sequential layers improved the performance of the model.

3.2.4 Model Comparison on Challenge Leaderboard

From most experiments, it is evident that the CNNE + RN model works much better. However, running the model on the challange leader-

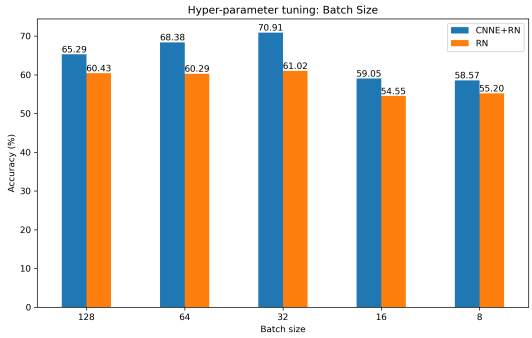

Figure 4: Setting Batch Size

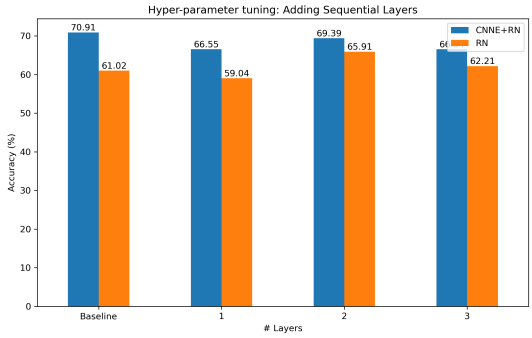

Figure 5: Adding sequential layers to the model

board shows in 6 that standalone RN model is **9%** better than the CNNE + RN model.

3.3 Protypical Network and Data Augmentation

3.3.1 Data Augmentation

In order to demonstrate the value of data augmentations, we employed a strategy of applying random transformations to images to create additional training data. To do this, we utilized a multi-crop strategy that utilizes a mix of views, as introduced in the SWaV framework. Specifically, we defined the growth-rate as the number of augmentations per image in the training data.

By using this approach, we were able to leverage the semantic information stored in the images, ultimately generating more data from our initial set of 50 training examples. This allowed us to explore the usefulness of data augmentations in a practical setting, providing valuable insights into how this technique can improve model performance in a variety of applications.

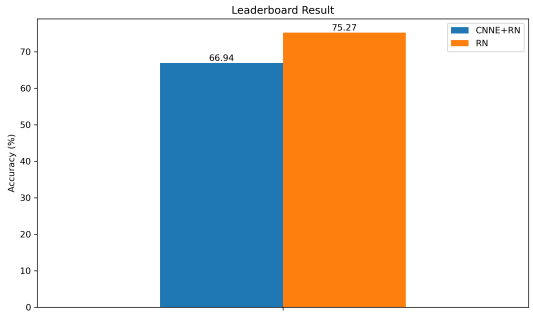

Figure 6: Accuracy of models on scoreboard

3.3.2 Prototypical Network Implementation

Prototypical networks [4] are a type of machine learning model that learns a metric space to perform classification tasks by computing distances to prototype representations of each class. However, in order to implement this network for our specific project requirements, certain modifications had to be made. We used the implementation from [3] and modified.

In the original paper [4], the training and testing of the network involved using independent transformations as separate classes (rotating at 90, 180, and 270 degrees), with the support set used for calculating the prototypes for each class and the query set classified using the embedding distance. However, since our setup did not allow for the **5-way and 20-way** scenarios of the original paper, we had to make some changes (**2-way**).

To implement the prototypical network for our project, we used the same encoder architecture as the original paper. **However, to generate the training data (support and query sets), we used random image transformations to create additional data to sample from**. Additionally, instead of using the (Q,S) set for testing and prediction as in the original paper, **we saved the detached class prototypes after averaging over each epoch and iteration, and used the Euclidean distance to classify**

lr	epochs	batch size	growth rate	Mean Acc	+-
0.005	100	32	2	70.05	8.73
0.010	100	64	2	69.40	7.51
0.010	100	64	8	69.25	10.49
0.010	150	32	4	68.20	7.22
0.005	100	32	1	67.95	7.43

Table 4: Data Augmentation: Accuracies over different hyper parameters, growth-rate: number of augmentation per image

epochs	epoch size	growth rate	mean accuracy	+-
3	500	2	55.67	0.077
3	100	8	55.49	0.078
3	500	4	55.34	0.076
3	500	8	55.32	0.076
3	100	4	55.18	0.077

Table 5: Prototypical Network Accuracies: over different hyper parameters, growth-rate: number of augmentaion per image

images during prediction.

Overall, these modifications allowed us to implement the prototypical network in a way that was tailored to our specific project requirements, while still maintaining the fundamental principles of the original paper.

3.3.3 Experiments

Data Augmentation: Table 4 3.3.3: Top 5 mean accuracy Based on the experiment, it is evident that the accuracy is affected to a certain extent when data augmentations are applied. This is primarily because the augmentations are only performed during the training phase and not during the testing phase. Additionally, the accuracy is influenced by the number of epochs and the batch size used during the training process.

Prototypical Network: Table 5 3.3.3: Top 5 mean accuracy According to the experiment, the prototypes of both classes are calculated by taking the average of their respective embeddings. The results indicate that better performance is achieved when the images from both classes are less similar. However, the increase in performance is not significant.

4 Methods applied for Challenge 2

4.1 MAML

4.1.1 Implementation

Based on [1], the best model would be a deep architecture with several linear layers at the end. The model in the MAML class is composed of 6 convolution layers with 4 linear layers at the end. This class also contains functions to keep track of the metaparameters such as *meta_named_pars* and *meta_params*.

The training is done on two tasks: the stl10 and svhn datasets. The function *maml_train* trains the model and contains an inner loop and an outer loop. In the inner loop, where for each task, the model is trained and tested. The gradients are updated and the losses for all tasks are added up. Once all tasks are completed, the outer loop backpropagates the outer loss to update the model parameters using the optimizer.

Althought the meta-optimizer introduced in [1], META-KFO, was not implemented, a simpler one, first-order approximation was implemented in *maml_train_FOA*.

The model will train on the SVHN, STL10 and CIFAR10 datasets. The function *maml_test* finetunes the model by training it on the test set before testing it on the entire test set.

4.1.2 Results

A combination of datasets were used where the model was trained on only the SVHN dataset, only the STL10 dataset, and on both the SVHN and STL10 datasets. Due to the poor results obtained, the CIFAR dataset was used for the taining as well, yielding a better result.The results for these datasets are shown in Table 4.1.2, 4.1.2.

In Table 4.1.2, we can see that the similarity of the training dataset to the test dataset is the most important factor as the CIFAR10 dataset has the best results followed by the STL10 dataset as it has a closer format to CIFAR and SVHN. However, having multiple datasets helps decrease the variance in the results.

Table 4.1.2 shows the effects of first order approximation. While our model does not show much better accuracy with the first order approximation, its variance has halved, proving that

Training Datasets	Accuracy
STL10	13.38 +- 7.50
SVHN	10.38 +- 6.65
STL10 and SVHN	12.31 +- 6.59
CIFAR10	66.03 +- 4.37

Table 6: Accuracy with changes in Datasets

Training Datasets	Accuracy
With First Order Approximation	12.31 +- 6.59
Without First Order Approximation	11.62 +- 3.44

Table 7: Prototypical Network Accuracies: over different hyper parameters, growth-rate: number of augmentaion per image

even in its limited capacity, the first order approximation has been beneficial.

4.2 SWaV

4.2.1 Implementation

The SWaV [2] framework is utilized for pre-training image representations through unsupervised learning, with the resulting pre-trained models able to be applied to various downstream classification tasks. However, the original model was trained on the large-scale Imagenet challenge dataset, which can be computationally intensive.

To adapt the SWaV framework for our purposes, we modified the paper's implementation [7] and we pre-trained our model on the SVHN and STL-10 datasets, making certain modifications in the process. However, we found that training the model on the full dataset was not feasible given our limited computational resources, so we instead utilized only 10% of the total dataset. Additionally, while the original paper trained models over 800 epochs, we experimented with shorter training periods of 50 and 100 epochs.

To implement our modified SWaV framework, we set up a full pipeline by adapting the original code to pre-train the model on our chosen dataset, and then fine-tuned it for challenge 2. However, due to our computational constraints, we were only able to achieve accuracies of 54% and 57% for 50 and 100 epochs, respectively.

The modifications we made included utilizing pre-trained weights from SWaV and fine-tuning the model for classification using a smallest pre-trained model with a linear layer. We also performed grid searches over different hyperparameters, such as learning rate, number of epochs, and batch size, to optimize the model's performance given our limited resources.

LR	Epochs	Batch Size	Mean Acc	+-
0.010	150	32	81.64	8.14
0.005	150	32	81.35	8.24
0.010	100	32	81.18	8.47
0.010	100	64	81.05	8.30
0.010	150	8	80.66	8.48

Table 8: SWaV + fc Fine tuning Accuracy: over different hyper parameters

4.2.2 Results

We used this set of hyper parameters range to find out best fine-tuned model using SWaV. lr = [1e-2, 1e-3, 5e-3, 1e-4, 3e-4, 5e-4], epochs = [10, 20, 30, 40, 50, 100, 150], batch size = [8, 16, 32, 64]. The Table 8 (4.2.2), shows the top 5 accuracies. These accuracies show that the batch size of 32 and learning rate of 0.01 are favorable for this dataset.

5 Discussion

5.1 Challange 1

- **Performance**: RelationalNetwork(CNNE+RN) performs better with reliability compare to other models. This is mainly due to the fact that the Relational models are designed to identify similar and dissimilar items which makes them work better on small dataset compared to other models. and ProtoTypicalNetwork performes worst we are limited to only two prototypes that gets averaged over training data and the variance of each image lowers the performance.

- **Training Time**: EvoDCNN has the longest training time because of grid search to find the best architecture given the training data. And ProtoTypicalNetwork and DataAug uses the image transformation to create the Support and Query set samples

that will increase the training time than RN and DataAug.

- **Inference Time**: Since the different models does have differnt number of layers in the architecture, the inference time depends on it, RelationalNetwork(CNNE+RE) does have the most number of layers and it takes the highest inference time compare to others while DataAug only uses 3 layers and takes the least amount of time.

5.2 Challange 2

- **Performance**: MAML performs better when the number of tasks from training and testing set are related. On the other hand, SWaV achieves far better results (80) than MAML due to pre-trained weights and fine-tuning the linear classifier.

- **Training Time**: For MAML, an increase in the number of tasks will result in a polynomial increase in the training time. In the case of SWaV, the training time is affected by various factors such as batch size, number of crops, and number of prototypes during pre-training. During fine-tuning, the encoder is always the same (resnet50), so the training time depends on the learning rate, epochs, and batch size.

- **Inference Time**: MAML has a higher inference time with an increase in the number of layers. SWaV, on the other hand, has a higher inference time compared to MAML due to the resnet50 encoder.

6 Contributions

6.0.1 Challenge 1:

EvoDCNN : Niki
Relational Networks: Waleed
SWaV: Omij

6.0.2 Challenge 2:

MAML : Implementation(Niki), Testing(All)
SWaV: Omij

References

[1] Sébastien M. R. Arnold, Shariq Iqbal, and Fei Sha. *When MAML Can Adapt Fast and How to Assist When It Cannot*. 2021. arXiv: 1910.13603 [cs.LG].

[2] Mathilde Caron et al. *Unsupervised Learning of Visual Features by Contrasting Cluster Assignments*. 2021. arXiv: 2006.09882 [cs.CV].

[3] *prototypical-networks*. https://towardsdatascience.com/few-shot-learning-with-prototypical-networks-87949de03ccd.

[4] Jake Snell, Kevin Swersky, and Richard S. Zemel. *Prototypical Networks for Few-shot Learning*. 2017. arXiv: 1703.05175 [cs.LG].

[5] Yanan Sun et al. *Evolving Deep Convolutional Neural Networks for Image Classification*. 2019. arXiv: 1710.10741 [cs.NE].

[6] Flood Sung et al. *Learning to Compare: Relation Network for Few-Shot Learning*. 2018. arXiv: 1711.06025 [cs.CV].

[7] *SWaV-Github*. https://github.com/facebookresearch/swav.

[8] *TinyData-Classification*. https://github.com/ozx1812/TinyData-Classification.