CZ4042 – Neural Networks and Deep Learning

Group Project Report

11th November 2022

Name	Matriculation Number
Agarwal Gopal	U1922859F
Arora Srishti	U1922129L
Pandey Pratyush Kumar	U1923491H

1 Problem Statement: Task F - Flowers Recognition

The classification of flower species has the following problems: Firstly, the traditional feature extraction mainly uses the characteristics of color, shape and texture. The artificial selection of these features is more complicated and difficult. Secondly, the same flower at different times and in various environments showing different gestures, and the color and shape of flower species are very similar, which brings great difficulty to identification.

2 Related Works

In recent years, the important research progresses on flowers identification include:

- In 2000 Saitoh et al. [1] used a method which requires the user to place a black cloth behind the flower to recognize flowers and this method is not feasible and inconvenient.
- Maria-Elena et al. [2] [3] used different characteristics of flowers and multi-core frame combination features, using support vector machine (SVM) as the classifier, the recognition rate of flowers in the Oxford-102 dataset can reached 88.33% [4];
- Angelova and Zhu [5] proposed a segmentation approach, followed by the extraction of Histogram of Oriented Gradient (HOG) features and using Locality-constrained Linear Coding (LLC).
- In 2014, Xie used the fine-grained flower classification to improve the accuracy of flowers, the recognition accuracy was 79.1% on Oxford-102 flower dataset [6].
- Liu et al. [8] propose a novel framework based on convolution neural network (CNN) to improve the accuracy of flowers, unlike other methods using hand-crafted visual features, the method utilizes convolution neural network to automatically learn good features for flower classification, can achieves 84.02% classification accuracy in Oxford-102 flowers dataset.
- To overcome the need for having a large infrastructure for training State-Of-The-Art models, Hassani et al. (2022) introduced Compact Transformers for image classification, which is a smaller version of Vision Transformer (ViT) and achieves 99.71% accuracy on the flowers' classification task. This model uses a new sequence pooling strategy, which pools over output tokens and improves performance.

The recent efforts to reduce the computation required to train and test models inspired us to explore Few Shot Learning as a classification strategy. Recent research shows that by using few-shot learning, machines can learn rare cases. Traditional deep convolutional neural network models require large image datasets for training and thus have high computational requirements. Few shot learning overcomes this limitation by reducing data collection effort and computational costs. As few-shot learning requires less data to train a model, high costs related to data collection and labelling are eliminated. Low amount of training data means low dimensionality in the training dataset, which can significantly reduce the computational costs.

3 Meta Learning

According to the traditional paradigm, an algorithm learns whether task performance gets better with practice when given a specific task. The Meta-Learning paradigm consists of a number of tasks. An algorithm is learning to determine whether its efficiency at each task increases with practice and the quantity of tasks. This program is referred to as a meta-learning algorithm. In **Meta-Learning**, we learn how to learn to classify given a set of training data. We use one set of classification problems for other, unrelated sets.

3.1 Metric Learning

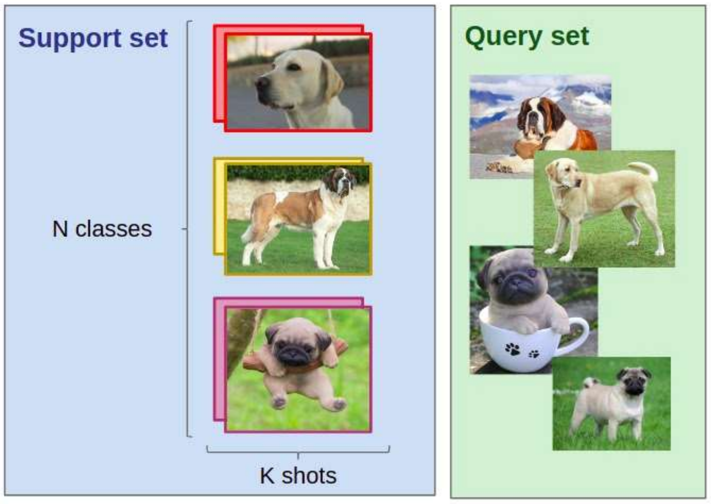
The aim of supervised metric learning is learning a distance measure between instance pairings that assigns a small (big, resp.) distance to examples that are semantically similar (dissimilar, resp.). When using few-shot classification, the metric is first learned on the base dataset. Query images of the novel class are then classified by calculating their distances from novel support images with respect to the learned measure, then using a distance-based classifier, such as the k-nearest neighbor (kNN) algorithm.

These techniques minimize overfitting to few training images in few-shot image classification and have attained the best performance by categorizing unseen samples according to their distances to few seen samples in an embedding space learnt by potent deep neural networks.

4 Few-Shot Learning Image Classification Task

Few-shot learning is a meta-learning framework. In few-shot classification, the goal is to reduce the prediction error on data samples with unknown labels given a small support set for "fast learning". To begin with, first we need to understand certain terminologies of a general meta-learning framework:

1. **Label Set:** This is the sample set of all possible categories. For example, if we have 1,000 categories and as part of sampling from the database, we only use only 5 categories.
2. **Support Set:** This is the sampled input data points (for example, images) of label set categories.



3. **N-way K-Shot Method:** Here, N is the size of the support set. In simpler terms, the number of possible categories in the training set. K is the number of examples of each category.
4. **Query Set:** A set of Q images used to query the model.
5. **Classical Training:** As the embedding needs to learn essential features for the classes to be discriminated, traditional Neural Network training is used to train the backbone on the actual image classification task (in our case identify the 102 flowers from the dataset). Then this trained backbone is used to preprocess the support and query set in the few-shot classifier
6. **Episodic Training:** Each episode corresponds to an N-way K-shot classification task. After the model solved every episode (*i.e.* it labelled the images of every query set), its parameters are updated. This is done by **backpropagating** the loss resulting from its classification inaccuracy on the query sets.

*Few-shot learning involves metric learning methods like prototypical networks, matching network and relation networks. For this project, we primarily explore the first two metric-learning techniques***.**

5 Methodology

The raw data was transformed using the following pre-processing techniques:

1. Resize the image to 128 x 128 pixels by cropping it
2. Random horizontal Flip
3. Normalization was performed on image data using ResNet50 pretrained values

The following meta-algorithms are used for the image classification task using few shot learning

1. Prototypical Networks
2. Matching Networks

5.1 Prototypical Network

We use Prototypical Networks for the problem of few-shot classification, where a classifier must generalize to new classes not seen in the training set, given only a small number of examples of each new class. Prototypical Networks learn a metric space in which classification can be performed by computing distances to prototype representations of each class. Compared to recent approaches for few-shot learning, they reflect a simpler inductive bias that is beneficial in this limited-data regime and achieve excellent results. Prototypical networks require a backbone network to create embeddings for different images.

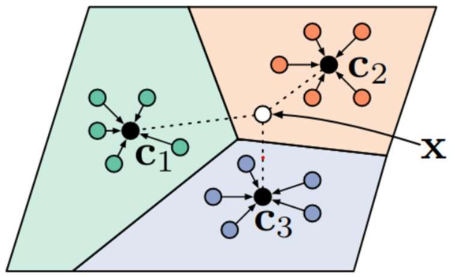


Figure 1: Prototypes for different clusters in the network

These networks differ from Matching Networks in the few-shot case with equivalence in the one-shot scenario.

5.1.1 Convolutional Autoencoder as backbone

Previously, we used pre-trained models like ResNet18 and ResNet12 to form the backbone of the network and provide embedding for the support and query sets.

Classical training according to us defeats the purpose of using metric learning or meta learning in the first place. Besides, using heavy-weight networks like ResNet18 and VGG19 which take up huge number of computational resources as well. Hence, we decided to experiment with lightweight auto-encoder to learn a latent feature space from the image. The following is the methodology we followed for this experiment:

1. Describe a convolutional autoencoder with the following architecture (refer figure 1). The architecture was borrowed from [10]. The authors used it to learn latent space for dogs and cat classification problem:

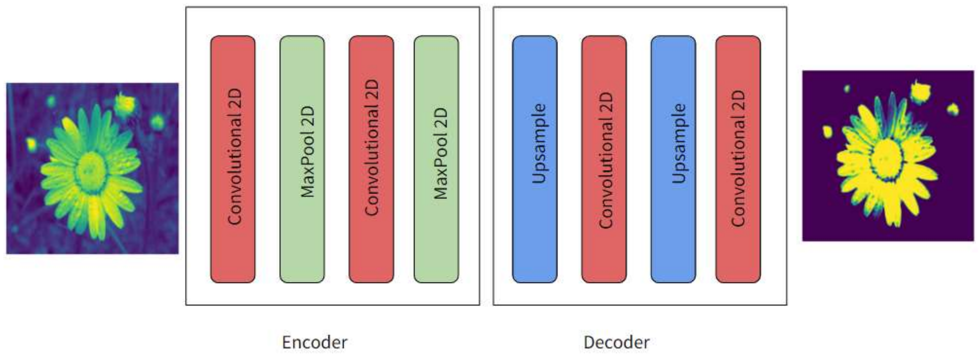


Figure 2: Convolutional Autoencoder architecture

2. Train the autoencoder to reduce MSE while reconstructing the image
3. Now use flatten layer in place of the decoder to form the backbone for a prototypical network. The flatten layer serves as the bottleneck for the autoencoder.

4. Now train the new network using episodical training for it to learn a better and generalized embedding for the meta learning task involving support and query sets.

The intuition behind such an approach is to initialize the weights and gradients of the backbone network with weights and gradients learnt after training it for a lightweight task i.e., in our case Image reconstruction using autoencoders. Since, the task is not cumbersome, we decided to experiment with a sparse convolutional autoencoder with just 2 convolutional and 2 pooling layers for encoder and decoders. This is very easy to train when compared to complex transfer learning models. Hence, we do not need to decide which layers to freeze and unfreeze for learning the latent space/embedding. This eliminates a hyperparameter.

After training the autoencoder and after performing episodical training on the prototypical networks, we realized a lot of limitations of our approach and the network did not perform as expected. This is further discussed in Section 6 of the report.

5.1.2 Modified Prototypical networks using KNN

According to the authors of [11], prototypical networks can be represented as simple linear classifiers which use a distance function belonging to the class of regular Bregman divergences to classify labels for a query set. We decided to combine the idea of using Euclidean distance from Prototypical networks and K-nearest neighbor idea from Matching networks to form a new distance /Score function for finding the class labels:

$$Score(Query, Label) = \frac{Euclidean\ Distance\ (Query,\ Prototype_{label})}{1 + Count(KNN, label)}$$

The **Count (KNN, label)** function counts the number of support images belonging to a specific label which are among the K-Nearest neighbors of the query images. We used 1 as a delta to avoid division by zero error. After this SoftMax function is applied to the score vector for each label.

$$\Pr(Query, label) = Softmax(Score(Query, label))$$

The intuition behind using such a score function is that we also use information about the K-nearest neighbors to compute a better and much more sophisticated score for each label in the query set. Greater the count of support images for the label among the K-nearest neighbors and lower the distance of the prototype of the label, higher the probability of query image belonging to that label/class.

In effect we also believe that the score could be made much better by adding additional hyper-parameters μ and α to the equation. The new score would look like the following.

$$Score(Query, Label) = \frac{Euclidean\ Distance\ (Query,\ Prototype_{label})}{1 + \mu\ Count(KNN, label)^{\alpha}}$$

These hyperparameters can be tuned depending on the task. The reason we chose Euclidean distance to be the numerator, is that it belongs to Bregman divergence class of distance functions. The class of functions plays an important role in deciding which prototype of a class is closer to the embedding of the query image (according to [11]).

5.1.3 Prototypical networks with Triplet Loss

5.1.3.1 Custom training using Triplet Marginal Loss

As we know, prototypical networks use distance function to compute the closest label to a query image based on a support set. In the process, the network compares the embedding of the query set with centroid of the embeddings of support set belonging to a specific class.

Using standard loss functions like Negative log likelihood, may help us learn a complex feature space for the network. However, since prototypical network leverage distance functions to discriminate between classes, it seems more intuitive to train a Siamese network using Triplet loss to learn a latent space representation for the classes. The feature space after training the Siamese network would tend to be much more clustered due to the contrastive nature of the triplet loss function.

We propose the following methodology for training Prototypical Network using Triplet Marginal loss(refer to Figure 2)

1. Select equal number of query images and support images for training task
2. Sample tasks using a 2-way train sampler and create a function to output the values for the Siamese network for the original pre-trained model (also referred as the backbone network)
3. Choose one label from the support set (at random) to be the anchor label or data
4. Choose the query images with same label as the positive label or data
5. The query images with a different label are chosen as the negative label or data

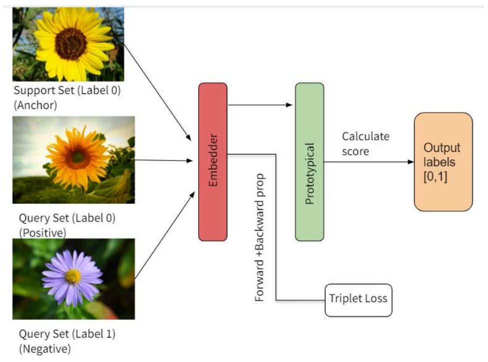


Figure 3: Architecture for Proto-Trip: While training the embedding network we use a Siamese network like function in the prototypical network

6. The embedding for the respective data is passed through the Triplet Marginal loss function and the model is evaluated on a 5-way validation sampler using the Prototypical Network.

After the model has been trained for enough episodes, the Prototypical Network is expected to test on 5-way test data sampler. The intuition is that since we use triplet loss for training, the embedding space of the dataset should be transferrable to a larger number of classes for evaluation as we train using pairwise distances of (anchor, positive) and (anchor, negative).

5.1.3.2 Semi-hard Triplet Loss

We also included a triplet semi-hard loss defined in [13]. The loss function encourages the positive distances (between a pair of embeddings with the same labels) to be smaller than the minimum negative distance among which are at least greater than the positive distance plus the margin constant (called semi-hard negative) in the mini batch. This implementation does not require us to define Siamese architecture as well.

5.2 Matching networks

Matching networks [14], in general, proposes a framework which learns a network that maps a small dataset and an unlabeled example to its label. It is a neural network model that implements an end-to-end training procedure that combines feature extraction and differentiable k-NN with cosine similarity. Matching networks are designed to be two-fold:

1. **Modeling level:** At Modeling level, Matching nets uses advances made in attention and memory that enable fast and efficient learning.
2. **Training procedure:** At Training Level, they have one condition that distribution of training and test set must be same. For example: show a few examples per class, similar to how it will be tested when presented with a few examples of a new task.

5.2.1 Model Architecture

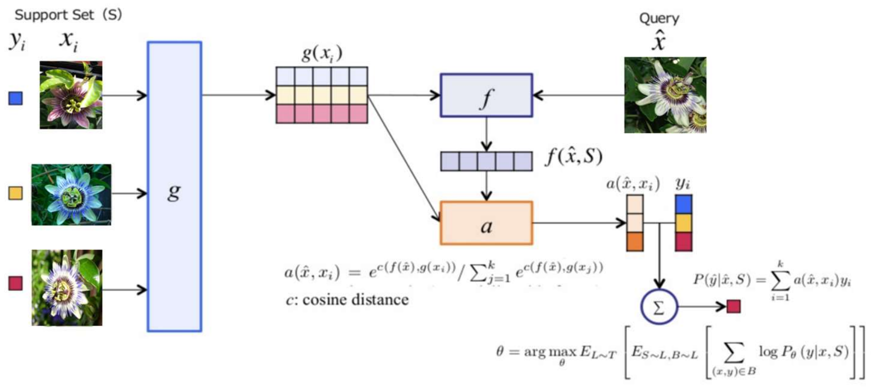

Support Set (S)
y_i
x_i
g
$g(x_i)$
f
Query
$\hat{x}$
$f(\hat{x}, S)$
a
$a(\hat{x}, x_i)$
y_i
$a(\hat{x}, x_i) = e^{c(f(\hat{x}), g(x_i))} / \sum_{j=1}^{k} e^{c(f(\hat{x}), g(x_j))}$
c: cosine distance
$P(\hat{y}|\hat{x}, S) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$
Σ
$\theta = \arg\max_\theta E_{L\sim T}\left[E_{S\sim L, B\sim L}\left[\sum_{(x,y)\in B} \log P_\theta(y|x, S)\right]\right]$

Figure 4: In the following sample architecture for matching networks, input to g(θ) are randomly sampled data (support set) from already existing examples, and input to f(θ) is input test data point

Let x_i, y_i be the samples and labels from the support set $S = \{(\text{xi}, \text{yi})\}_{i=1}^{k}$, and ***a*** be **an attention mechanism.** Let $\hat{x}$ be the query set. The function used to compute $\hat{y}$ in Matching networks is defined as follows:

$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

Equation 1

5.2.2 Data Processing

1. **Data Processing Backbone:** The support set x_i is passed to the backbone of the FewShotLearning classifier. In this study we have experimented by using pretrained ResNet50 and classically trained ResNet50 architectures as the backbone.
2. **Extracting Fully Contextual Embeddings**: The Support set x_i and query set $\hat{x}$, are passed through a standard feature extraction layer (g) and (f) respectively, which consists of a bidirectional LSTM architecture. It helps us learn the probabilistic distribution of labels present in support set.
3. **Attention Kernel:** After obtaining the outcomes from both supports set and query image, they get fed into an attention (a) kernel which uses the Softmax function over the cosine distance c.

5.2.3 Training Methodology

Matching Networks learns its parameters through training-set by creating small subsets [8] as that of a test-sets; that is, it is actually training the model as a few-shot learning model.

6 Experiments

While discussing experiments, we will be using the following naming convention for the different methods of Few-Shot learning we have tried:

1. Prototypical Networks using Convolutional auto-encoder: *Proto-ConvAE*
2. Prototypical Networks using KNN with Semi-hard Triplet Loss: *Proto-KNN-Trip-Semihard*
3. Prototypical Networks with Custom Triplet Loss: *Proto-Trip-Custom*
4. Prototypical Networks with Semi-hard Triplet Loss: *Proto-Trip-Semihard*
5. Matching Networks: *Match*

6.1 Proto-ConvAE

- The test results of this variation do not compete well with the benchmark results. We observe a significant drop in accuracy. ***The validation accuracy drops to 64.40% and test accuracy drops to 53.33%.***
- It can also be noted there is a huge difference in accuracy between validation and test. This implies the model is heavily overfitted.
- One of the major reasons for this is the lack of regularization when training our autoencoder. One of the most common problems of using an autoencoder is the learning of useless and redundant embeddings due to overfitting. Since, we also want to generalize over 102 classes using Few-Shot Learning, it proves to counter-intuitive to use an overfitted embedder as a backbone.
- From the loss vs epoch curve, it can be observed that the loss saturates after 11-12 epochs.

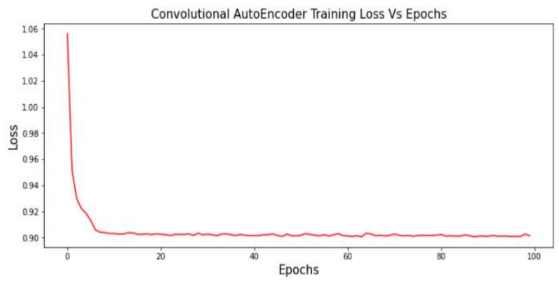


Figure 5: MSE Loss curve for ConvAE

- This saturation is very quick and implies that the autoencoder is not able to learn an effective and discriminatory latent representation. We hypothesize that the encoder and decoder networks are too lightweight to generalize over 102 classes. Hence, we need to train a much more complicated and denser autoencoder with some level of regularization and batch normalization.
-

6.2 Proto-KNN-Trip-Semihard

- We also tried out the semi-hard triplet loss function with Proto-KNN model. Upon trying multiple optimizers – Adam and SGD; multiple learning rates – 0.01, ,0.001, 0.0001; and multiple backbones, following are the results.

Backbone	Optimizer	Learning Rate	Validation Accuracy	Test Accuracy
Resnet50 Custom Trained	Adam	0.001	96.95	91.90
Resnet50 Custom Trained	SGD	0.001	97.75	92.08
Resnet50 pretrained	Adam	0.0001	86.70	80.15
Resnet50 pretrained	SGD	0.001	85.40	80.15

- **Best K = 10 ; Best $\alpha = 2$**
- This variant, while performing well custom trained Resnet50 on the test, does not provide the same level of accuracy as the normal Prototypical model with semi-hard triplet loss does above.
- This could be attributed to the reasoning provided in [11]. The authors show for Bregman divergences that the cluster representative achieving minimal distance to its assigned points is the cluster mean. Hence, when we use a prototypical network (involves calculation of cluster means), Euclidean distance is the optimal distance function to use. With experimental data, they also show that it outperforms cosine similarity.
- Hence, it is possible that the incorporation of KNN count in the new scoring function will not contribute much to improving the model.

6.3 Proto-Trip-Custom

- Our variation of using Triplet loss in training embedding for few-shot learning does not improve the accuracy. ***The validation accuracy drops to 77.50% and test accuracy drops to 48.65%.***
- The model shows huge sign of overfitting the train and validation. This implies that the model does not generalize well over the different classes seen in the support set.

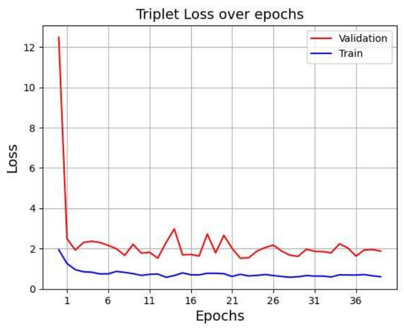


Figure 6: Loss vs Epochs curve

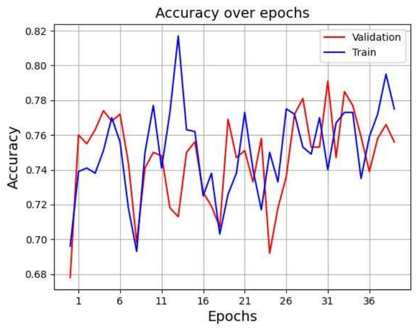


Figure 7: Accuracy vs Epochs Curve

- After analyzing the training loss over epochs (shown in the figure above), we realized that the loss saturates very quickly and keeps fluctuating in a very small range.. This can be attributed to the number of classes in the dataset. Since triplet loss analyzes the pairwise distance of an anchor label to a negative label, it may not be able to learn for all pairwise combinations. Also, the number of samples per class is 10 in the training dataset. This again will not be enough to learn good parameters using advanced loss functions like triplet loss.

6.4 Proto-Trip-Semihard

- The results after using the semi hard triplet loss defined in [13] are as follows:

Backbone	**Optimizer**	**Learning Rate**	**Validation Accuracy**	**Test Accuracy**
Resnet50 Custom Trained	Adam	0.001	97.30	94.70
Resnet50 Custom Trained	SGD	0.001	97.55	94.82
Resnet50 pretrained	Adam	0.01	92.00	89.27
Resnet50 pretrained	SGD	0.0001	92.6	90.55

- We continually observed that for vanilla Prototypical Network and modified version using KNNs, the accuracy achieved with semi-hard triplet loss function always outperformed the baseline models with Categorical Cross Entropy loss. Thus, the results for Categorical Cross Entropy loss have not been reported here.

6.5 Match

Matching networks return log-probabilities and thus we need to use negative log likelihood as the loss function.

6.5.1 Results: Using Pre-trained Resnet50 as backbone

The pretrained Resnet50 architecture was used as the convolutional network. A fully connected flattened layer was added to the architecture and trained on the training flowers-102 dataset. This model performed better than other few shot learning

Num of classes in a task (N_WAY)	Num of images per class in the support set (N_SHOT)	Number of images per class in the query set (N_QUERY)	Best Validation Accuracy	Test Accuracy
5	4	6	93.72%	92.33%
4	4	5	95.25%	91.74%

Table 1: Results of matching network using pre-trained resnet50 as backbone

As shown in table 1, this model outperforms prototypical networks and performs better than majority of the models discussed in literature review.

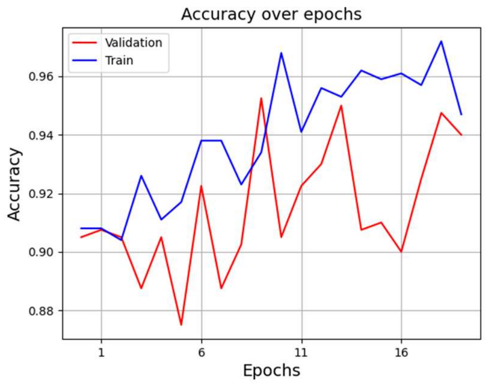


Figure 8 8: Accuracy for 4-way, 4-shot, 5 query queries

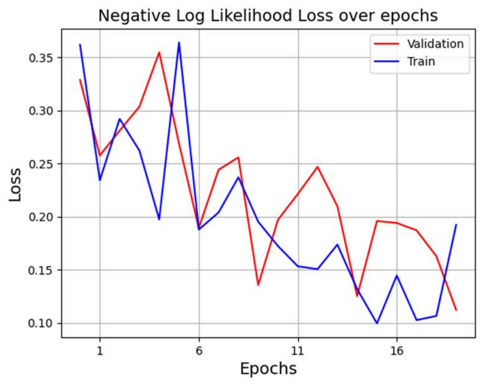


Figure 99: Loss for 4-way, 4-shot, 5

Fig 4 shows that accuracy follows an increasing trend overall. However, we observe some drops in both train and validation accuracies because the set of classes evaluated in every epoch vary and hence the ***accuracy scores fluctuates because the network takes time to generalize over 102 classes using a 4-way technique.***

6.5.2 Results: Using Resnet50 trained using Transfer Learning as the backbone

The architecture of Resnet50 (excluding the final fully connected layer) was classically trained on the training set of the flowers-102 dataset using transfer learning. These trained weights were used to initialize the matching network model. A fully connected linear layer was added to the matching network model and trained.

Num of classes in a task (N_WAY)	Num of images per class in the support set (N_SHOT)	Number of images per class in the query set (N_QUERY)	Best Validation Accuracy	Test Accuracy
4	4	5	94.38%	93.96%
10	4	5	93.66%	92.90%

Table 2: Results of matching network using classically trained resnet50 as backbone

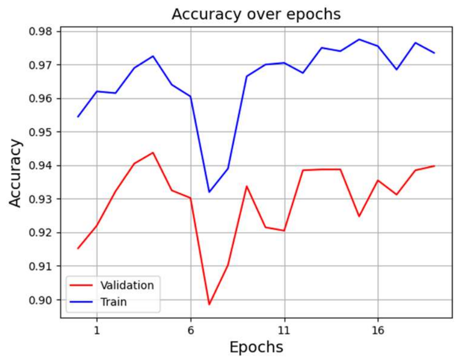


Figure 1010: Accuracy for 4-way, 4-shot, 5 query

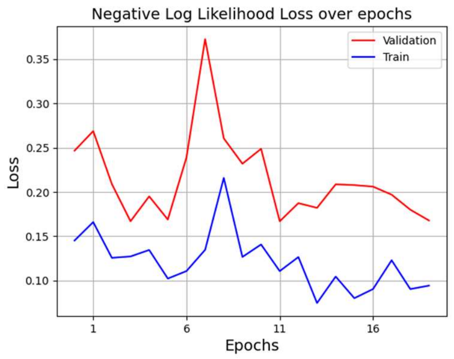


Figure 1111: Loss for 4-way, 4-shot, 5 query

Training the ResNet50 architecture helped us achieve slightly higher results as compared to pretrained model. We observed that we can train a 10 Way few shot classifier without much degradation of accuracy using matching networks.

7 Discussion

Overall, for the chosen Oxford Flower 102 dataset, since the training data samples per class are limited to 10, we chose the method of Few-Shot Learning. We have tried two metric meta learning algorithms – Prototypical networks and Matching networks.

In addition to implementing the normal prototypical metric learning based network, we experimented with myriad modifications of the standard prototypical model. We experimented with both – a pretrained ResNet50 backbone and a classically trained ResNet50 architecture. We also attempted using a Convolutional Autoencoder for our backbone. Additionally, we also implemented a modified version of the prototypical network using KNN. Thereafter, we developed a matching network and experimented with different backbones to process data for the same. Finally, we tuned our hyperparameters to enhance our results. 94.82% is the best test accuracy was achieved with the help of a prototypical network using a semi-hard triplet loss (refer Fig 12).

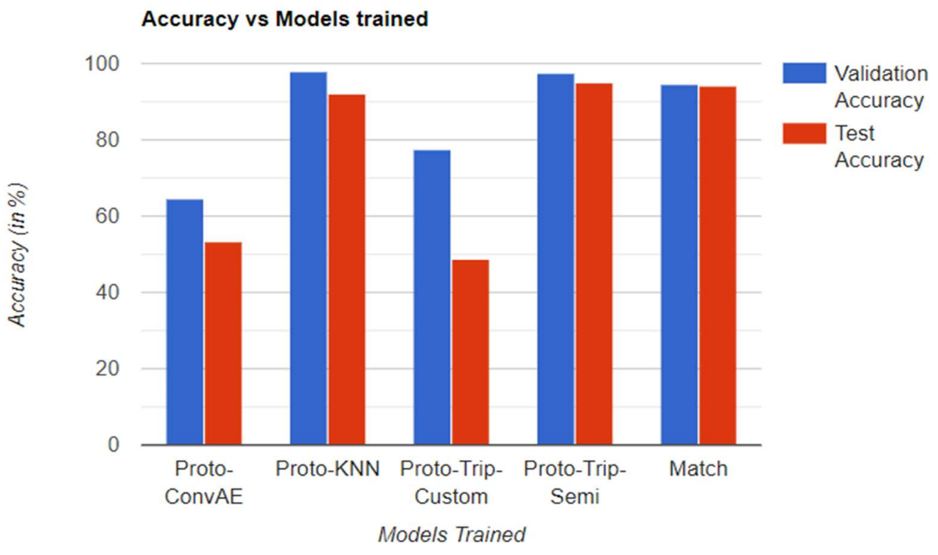


Figure 1212: Summary of the experiment results

There is a potential to improve the results obtained from these models. We recommend the following future improvements:

- We can try using variational autoencoders with block units resembling those in complicated architectures like Inception and ResNet50. This will serve as a replacement for the convolutional autoencoders we have trained.
- We can try to form a metric using trainable parameters for the distance function and KNN Count. This would imply that they would be fed to multi-layer perceptron for further training and optimization for the Prototypical model with KNN.
- Due to paucity of time, we only tried Random search technique for finetuning α. It is possible that an exhaustive grid search for the hyperparameters for μ and α may gave slightly better results for the Prototypical model with KNN.
- In matching networks, we can try using a multi-head attention layer with large number of heads. This will help us to better extract features from the large images resulting in possibly increased accuracy.
- We can experiment with exponents for the Minkowski distance instead of using Euclidean distance while calculating classification score in Prototypical networks.

8 References

[1] Saitoh, T.; Kaneko, T., “Automatic recognition of wild Flowers”, Pattern Recognition, Proceedings. 15th International Conference on, vol.2, no.,pp.507-510 vol.2, 2000.

[2] Maria-Elena Nilsback, Andrew Zisserman: A Visual Vocabulary for Flower Classification. CVPR (2) 2006.pp.1447-1454,2006.

[3] Maria-Elena Nilsback, Andrew Zisserman: Delving into the Whorl of Flower Segmentation. BMVC 2007.pp. 1-10,2007.

[4] MarMaria-Elena Nilsback, Andrew Zisserman: Automated Flower Classification over a Large Number of Classes. ICVGIP ,pp.722- 729,2008.

[5] Angelova A, Zhu S. Efficient object detection and segmentation for fine-grained recognition. In Computer Vision and Pattern Recognition(CVPR), 2013 IEEE Conference on, pp.811-818, 2013.

[6] Xiaodong Xie, Research on Fine-Grained Classification for Visual Flower Image. Xiamen University, 2014.

[7] Liu, W., Rao, Y., Fan, B., Song, J., & Wang, Q. Flower classification using fusion descriptor and SVM. International Smart Cities Conference.pp.1-4. 2017

[8] Liu, Y., Tang, F., Zhou, D., Meng, Y., & Dong, W. “Flower classification via convolutional neural network.”,International Conference on Functional-Structural Plant Growth Modeling, Simulation, Visualization and Applications IEEE,pp.110-116. 2017

[9] SHEN Ping, ZHAO Bei. “Automatic Classification of Flowers Based on Deep Learning Model”, Bulletin of Science and Technology, 33(3):pp.115-119, 2017

[10] https://github.com/RutvikB/Image-Reconstruction-using-Convolutional-Autoencoders-and-PyTorch/blob/main/Conv_AE_Pytorch.py

[11] Snell, J., Swersky, K., & Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in neural information processing systems*, *30*.

[12] Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. Journal of Machine Learning Research, 6(Oct):1705–1749, 2005.

[13] https://github.com/alfonmedela/triplet-loss-pytorch

[14] Mustière, S., & Devogele, T. (2008). Matching networks with different levels of detail. *GeoInformatica*, *12*(4), 435-453.