t-AI-tans Mini-Project 2

Akshat Sharma
230101

Praneel B Satare
230774

Kanak Khandelwal
230520

Dweep Joshipura
230395

1 Task 1

1.1 Abstract

We decided to use a pre-trained Deep Neural Network to get features for our models, and applying Learning with Prototype classifier on these features. We use a weighted update rule for the prototypes to get a new set of prototypes which are used for classification on the next dataset.

1.2 Analysis & Rationale

We started out by trying simple methods on LwP, without using any feature extractor. The data was first flattened (to get 32*32*3 = 3024 dimensional features), then passed on to the classifier. We also attempted use of PCA to reduce dimensionality before applying LwP.

Model	**Training Accuracy (%)**
LwP (Euclidean)	29.04%
LwP (Mahalanobis)	09.52%
LwP (Euclidean, PCA (50))	28.56%
LwP (Mahalanobis, PCA (50))	41.20%

Table 1: Accuracies of LwP without Feature Extraction

Clearly, the models are **unable to understand the underlying nature** of the image, which is expected from such simple models. Therefore, we chose to use a **pre-trained feature extractor**

1.3 Choosing Pre-Trained Feature Extractor

As the constraints given involved not using a model trained on CIFAR, we explored models which were trained on ImageNet (which is also an image dataset, however it has sufficient deviation from CIFAR due to wider variety of classes and distribution of images).

To compare the feature extractors, we applied LwP with Euclidean distance to the features generated by the models, and compared accuracies on D_1. These are given in Table 2. We chose **BEiT-Large** due to its highest accuracy. We extracted features for all the data (train & eval, on both Task 1 and 2) on a Kaggle P100 GPU, which took 2h 38m. (Kaggle Notebook)[1]

The feature vectors from BEiT-Large $\in \mathbb{R}^{1024}$. For other details regarding BEiT, refer [1]. We implemented it using the guidelines given here.

[1]The outputs can be found in the *output/* page. The .pkl files contain all the features.
NOTE: v3 is the correct version of the notebook to replicate the results (not v4)

Model	D_1-train Accuracy (%)
ResNet	84.12%
MobileNetv3	83.72%
CaiT-M36	94.20%
ViT-Base	96.52%
Eva02-Base	96.88%
BEiT-Large	**98.72%**

Table 2: Accuracies of LwP with Feature Extractors (trained on ImageNet)

1.4 Update step

To combat forgetting, we devised an update step on the prototypes, which takes into consideration both the previous prototypes and the pseudo-labels on the new data.

On every dataset $n+1$, we make the following change to prototypes μ

$$\mu_c^{(n+1)} := \frac{N\alpha\mu_c^{(n)} + \sum\limits_{y_i^{(n+1)}=c} x_i^{(n+1)}}{N\alpha + n_c^{(n+1)}}$$

where α is a hyperparameter

N is total examples in datasets, i.e., 2500

n_c^{n+1} is the number of examples of class c in dataset $n+1$

1.4.1 Characteristics of update step

1. As it includes both the previous prototypes as well as the pseudo labels, it incorporates a combination of information
2. The tradeoff between remembering and utilizing new data is determined by α. A higher α represents a greater emphasis on the previous prototypes (as $\alpha \to \infty$, $\mu^{(n+1)} \to \mu^{(n)}$). Similarly, at $\alpha = 0$, the pseudo labels completely define the prototypes.
3. If $\alpha < 1$, then more weight is given to the newer labels, and if $\alpha > 1$, then we are emphasising more on the old prototypes. For $\alpha = 1$, we get the average of the new and old prototypes.

1.4.2 Hyperparameter Tuning: Choosing α

As per the constraints of the project, we evaluated α by comparing the accuracy of $f^{(10)}$ on D_1, and chose the α giving the highest accuracy. We limit α to be less than 2 (as $\alpha \to \infty$ would give us a prototype trained only on D_1). The peak is found to be at $\alpha = 0.2$, and we will hereafter use it.

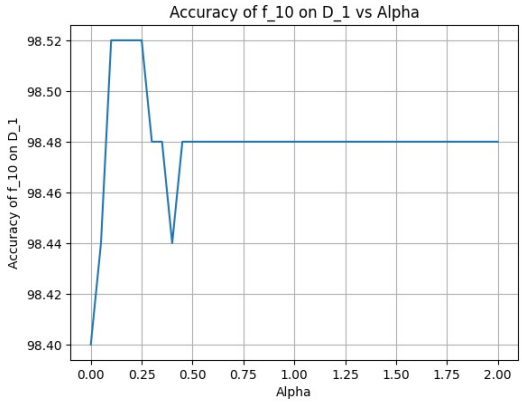

Figure 1: α v/s f^{10} accuracy

1.5 Results

The models were run on all datasets prior to the trained dataset, and the validation accuracies are shown in the table.

Model	$\hat{D}_1$	$\hat{D}_2$	$\hat{D}_3$	$\hat{D}_4$	$\hat{D}_5$	$\hat{D}_6$	$\hat{D}_7$	$\hat{D}_8$	$\hat{D}_9$	$\hat{D}_{10}$
f_1	98.32%	—	—	—	—	—	—	—	—	—
f_2	98.36%	97.84%	—	—	—	—	—	—	—	—
f_3	98.16%	97.76%	98.16%	—	—	—	—	—	—	—
f_4	98.16%	97.76%	98.04%	97.92%	—	—	—	—	—	—
f_5	98.20%	97.68%	97.96%	98.00%	97.92%	—	—	—	—	—
f_6	98.12%	97.84%	98.00%	97.92%	97.96%	98.40%	—	—	—	—
f_7	98.16%	97.72%	97.92%	97.92%	98.00%	98.36%	97.40%	—	—	—
f_8	98.00%	97.76%	97.80%	97.84%	97.88%	98.36%	97.36%	97.56%	—	—
f_9	98.08%	97.72%	97.84%	97.96%	97.84%	98.28%	97.36%	97.60%	97.68%	—
f_{10}	98.04%	97.76%	97.88%	97.96%	97.84%	98.28%	97.36%	97.60%	97.64%	97.88%

Table 3: Accuracies of models f_1 to f_{10} on held-out datasets

2 Task 2

2.1 Introduction

In this task, we are to generalize the LwP prototypes, so that they are able to tackle data from a different distribution than the training data. We will use the same feature extractor and pipeline as in Task 1, as the initial model.

2.2 Idea

In the first task, the adaptation for the new datasets involved the use of pseudo labels, which were also generated from the same prototypes that were used before. Therefore, there is heavy dependence and bias on the previous data (and hence its distribution) when attempting to classify the next dataset.

Given the need for a reduced-bias model, we see that the task has the following objectives

1. Understand the new distribution of images, and perform well on it.
2. Take into account the learned classification from the previous set of images.

The above task is known as **Unsupervised Domain Adaptation**, as we do not have the ground truth labels for the new distribution of classes. There are several methods in literature that we considered implementing for this, such as Subspace Alignment (SA) [2], Sliced Wasserstein Distance (SWD) [3], Nyström Subspace Override (NSO) [4] etc.

However, [2] and [4] involve dimensionality reduction (which both alters the distribution and leads to loss in accuracy (we got $\approx 20\%$ accuracy using a simple implementation of SA), and [3] requires solving an optimization problem (through gradient based updates), which is restricted by the constraints of the project.

We took inspiration from papers such as [5] (and the methods cited in it), to come up with our **Clustering-based Prototype Updation** method.

2.3 Clustering-based Prototype Updation

This method involves two major steps -

1. Class-aware clustering of new dataset
2. Adapting previous prototypes to newly found cluster centroids

2.3.1 Class-aware Clustering

We implement standard K-means for the clustering, with the important change that the cluster centres are initialised to the prototypes from the previous dataset. This step is critical in enabling correlation between clusters and the classes. Therefore, the cluster centre initialised to the class c prototype would represent, post-clustering, a possible candidate for the prototype of that class. The clustering process would change the position of the centre to better suit the new dataset.

2.3.2 Adapting previous prototypes

To ensure that forgetting does not occur, we consider a weighted sum of the previous prototype along with the converged cluster centre. Therefore, we set

$$\mu_c^{(n+1)} := \frac{\beta\mu_c^{(n)} + M_c^{(n+1)}}{\beta + 1}$$

where β is a hyperparameter

and M_c is the centroid for class c

This method ensures the balance between remembering the previous information about class discrimination as well as the shift in distribution for the new classes. It is also easy to implement and interpret.

2.3.3 Hyperparameter Tuning: Choosing β

We chose a β based on pure heuristics. For an effective choice of β, we would need labelled data from at least two domains, from which we're allowed to tune. As we are not allowed to tune our model on the validation set, our choice of β is based on an attempt to equally weigh new and old information. Hence we set $\beta = 1$.

2.4 Results

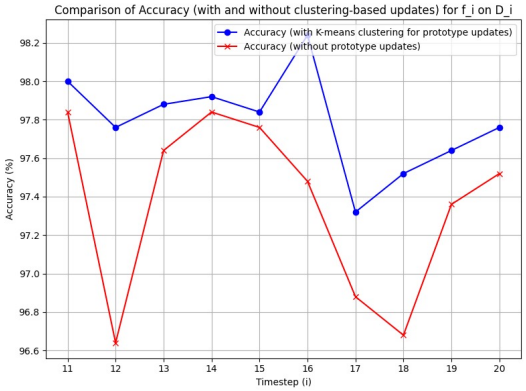

Figure 2: Increase in accuracy due to Prototype Updates

2.5 Scope for Improvement

While our results show significant domain adaptation in general, it fails to work well for some datasets (ex: $\hat{D}_{17}$), which would likely occur due to lack of correlation between cluster centres and class-split in the dataset. Improving this can be done by performing more sophisticated clustering algorithms, and also using a more powerful update-rule rather than weighted combinations.

Model	$\hat{D}_1$	$\hat{D}_2$	$\hat{D}_3$	$\hat{D}_4$	$\hat{D}_5$	$\hat{D}_6$	$\hat{D}_7$	$\hat{D}_8$	$\hat{D}_9$	$\hat{D}_{10}$
f_{11}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{12}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{13}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{14}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{15}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{16}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{17}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{18}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{19}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%
f_{20}	98.32%	97.88%	98.36%	98.0%	98.12%	98.36%	97.56%	97.88%	97.92%	98.28%

Table 4: Accuracies of models f_{11} to f_{20} on datasets D_1 to D_{10}.

Model	$\hat{D}_{11}$	$\hat{D}_{12}$	$\hat{D}_{13}$	$\hat{D}_{14}$	$\hat{D}_{15}$	$\hat{D}_{16}$	$\hat{D}_{17}$	$\hat{D}_{18}$	$\hat{D}_{19}$	$\hat{D}_{20}$
f_{11}	90.36%	-	-	-	-	-	-	-	-	-
f_{12}	90.36%	75.92%	-	-	-	-	-	-	-	-
f_{13}	90.36%	75.92%	93.56%	-	-	-	-	-	-	-
f_{14}	90.36%	75.92%	93.56%	97.28%	-	-	-	-	-	-
f_{15}	90.36%	75.92%	93.56%	97.28%	97.92%	-	-	-	-	-
f_{16}	90.36%	75.92%	93.56%	97.28%	97.92%	94.56%	-	-	-	-
f_{17}	90.36%	75.92%	93.56%	97.28%	97.92%	94.56%	94.56%	-	-	-
f_{18}	90.36%	75.92%	93.56%	97.28%	97.92%	94.56%	94.56%	91.32%	-	-
f_{19}	90.36%	75.92%	93.56%	97.28%	97.92%	94.56%	94.56%	91.32%	76.48%	-
f_{20}	90.36%	75.92%	93.56%	97.28%	97.92%	94.56%	94.56%	91.32%	76.48%	97.24%

Table 5: Accuracies of models f_{11} to f_{20} on datasets D_{11} to D_{20}.

3 Conclusion

We demonstrated methods to prevent catastrophic forgetting and domain adaptation, using simple prototype updates for LwP models. These methods led to a gain of about 5% in accuracy across datasets.

References

[1] H. Bao, L. Dong, S. Piao, and F. Wei, "Beit: Bert pre-training of image transformers," 2022.

[2] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars, "Subspace alignment for domain adaptation," 2014.

[3] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," 2018.

[4] C. Raab and F.-M. Schleif, "Low-rank subspace override for unsupervised domain adaptation," in *KI 2020: Advances in Artificial Intelligence* (U. Schmid, F. Klügl, and D. Wolter, eds.), (Cham), pp. 132–147, Springer International Publishing, 2020.

[5] J. Dridi, M. Amayri, and N. Bouguila, "Unsupervised clustering-based domain adaptation for estimating occupancy and recognizing activities in smart buildings," *Journal of Building Engineering*, vol. 85, p. 108741, 2024.

4 Problem 2

Link to YouTube video: "Deja Vu: Continual Model Generalization for Unseen Domains" (Research paper explained)