CS771 Mini Project II
DReaMeR - Dynamic Readaptation and Memory Retention

Mohd Mufeed Amir
220660

Mohd Sufyan
220662

Nishant Pandey
220724

Adarsh Pal
220054

1 Brief Introduction

We outline various methods and approaches employed toward incrementally learning across several subsets drawn from differing distributions of the $CIFAR-10$ dataset. `Task 1` of the project (presented here) has two main subtasks.

- **Task 1.1** Incrementally training models on datasets with a shared distribution.
- **Task 1.2** Incrementally training models on datasets with varying distributions.

Our main model is a simple `LwP` classifier. What we have tweaked - is *how the model updates* when presented with new data it may not have seen before.

2 Problem Statement

We are given 20 training datasets $D_1, D_2 \dots D_{20}$ from $CIFAR-10$

- **Task 1:** D_1 to D_{10} have inputs from a similar distribution.
- **Task 2:** D_{11} to D_{20} have inputs from different distributions.

The first dataset, D_1, is labeled; others are unlabeled. Our goal is to incrementally train a series of models, $f_1, f_2 \dots f_{20}$, where each model is trained using the predictions of its predecessor.

3 Task 1: Incremental Learning on Similar Distributions

3.1 Simple `LwP`

We started with a simple `LwP` based on Euclidean distance.

- Convert the image to grayscale (to remove dependence on color)
- Apply flattening operation to obtain a $2D$ feature representation

Observe that our feature vectors have size $32 \times 32 = 1024$.

Using this model to train over the first dataset and testing over the validation set of the first dataset, we get a very poor accuracy of approximately $16\% - 24\%$. We abandon this approach.

3.2 Mahalanobis Distance

We try a more advanced `LwP` using the *Mahalonobis Distance*, with the matrix being the *Inverse Covariance Matrix* over the training data. The rest of the procedure is similar to 1.1. The accuracy that we get on validation is again pretty low; about 15.64%

We finally abandon naive `LwP` image classifiers and move on to feature extraction.

4 `ResNet` for Feature Extraction

We try a Pretrained Neural Network: `ResNet34`. `ResNet34`, part of the `ResNet` (Residual Network) family, is a convolutional neural network with 34 layers, designed with a series of convolutional blocks interspersed with residual connections. The key innovation in `ResNet` is residual (or *skip*) connections, which allow the model to bypass certain layers, mitigating the vanishing gradient problem commonly encountered in deep networks. This enables the training of deeper networks without compromising performance or convergence. We further leverage `PyTorch` for its inbuilt functions to handle `ResNet`.

The architecture of `ResNet34` can be summarized as follows:

- The network starts with a 7×7 convolutional layer with 64 filters, followed by a 3×3 max pooling layer.

This reduces the spatial dimensions of the image and extracts initial low-level features. The network further consists of 4 groups of residual blocks with increasing filter sizes. Each block contains a series of convolutional layers with `ReLU` activations, batch normalization, and skip connections.

The residual blocks are structured as follows:

- Block 1 - 3 residual blocks, each with 3×3 convolutional layers and 64 filters.
- Block 2 - 4 residual blocks with 128 filters.
- Block 3 - 6 residual blocks with 256 filters.
- Block 4 - 3 residual blocks with 512 filters.

Each group of blocks is separated by a down-sampling convolution layer that doubles the number of filters and halves the spatial dimensions of the feature maps, capturing increasingly abstract and complex features as the layers progress. The model further features a global average pooling layer; reducing the feature map to a fixed-size 512-dimensional vector.

In the original `ResNet34`, this is followed by a fully connected layer for classification. However, in our setup, we omit this layer to retain only the feature representation.

To extract features from the images using `ResNet34`, we adopt the following approach.

- Each 32×32 $CIFAR - 10$ image was resized to 224×224, matching the input size required for the neural net.
- We applied standard normalization with mean and standard deviation values for the net, helping align the pixel distributions of $CIFAR - 10$ images with those seen by the net during pre-training. Additionally, we incorporated data augmentation via random horizontal flipping to increase model robustness.
- The outputs from the penultimate layer were flattened into 512-dimensional vectors, forming a rich feature set that served as inputs for our classifier. These 512 length vectors for each training example are ready for training.

This procedure is outlined at the official documentation for `ResNet34` [2]

4.1 Feature Extracted Naive `LwP`

We train an `LwP` classifier on feature extracted versions of D_1. At prediction time, we can choose whether to predict on Mahalanobis distance or Euclidean distance. Both give accuracies of 85.36% and 81.36% respectively. We proceed with Mahanalobis at prediction.

Even still, feature extraction *and* updating covariance matrices upon learning new examples failed to be effective on the new datasets $(D_{11} \ldots D_{20})$. We are now forced to change our approach from the ground up.

5 Task 2: The Actual Model

Before we go into the final results obtained, let's have a look at the model itself. Indeed, while our model heavily builds upon the work of Liu et al. (2023) [3], it addresses a narrower scope with a simpler framework.

Our model is quite simple, actually. Let f_i denote the model trained on datasets $D_1, \ldots, D_i$. For our model to learn on dataset D_{i+1}, we implement the following functionalities.

5.1 Top-k Confidence Selection

Let f_i be trained and let f_i encounter D_{i+1} for the first time. Using f_i, we label the training examples in D_{i+1} and assign each a *confidence* score. The score is inversely proportional to the softmax of the distances of this training point from the prototypes learned by f_i (ie, points further away from a prototype get assigned lower scores). We then sort each prediction by its confidence value, and return the predictions that have a confidence within *top-k* of all confidence values.

5.2 Sample Duplication

Given a labeled sample, we make the natural assumption that samples that lie *close* to it must also have the same label. So to increase our training space, we take some labeled samples and apply `AdaIN2D` spatial transformations to augment those samples with random Gaussian noise with 0 mean and low variance (we once again refer to [3] and [4]). This provides us with *more training examples from the dataset* ***without exhausting it***.

5.3 The Model

Having established these procedures, our model is, once again, simple. For datasets $D_1 \ldots D_{10}$, we *double* the training data using *sample duplication*. This makes our model more robust on the original dataset and more confident on *noisy* data. To train f_{i+1} from f_i, we use the simple update procedure with the doubled samples.

Now, for datasets $D_{11} \ldots D_{20}$, we make a slight modification. Because these datasets are drawn from a different distribution, we first apply the *top-k confidence selection* to obtain some samples that the model is sure about. On these new samples, we apply *sample duplication* so the model can learn more about this sample before making a prediction.

We finally update f_i to f_{i+1} by training it on the top-k and duplicated samples, and using this to label the leftover data.

6 Results

6.1 Results on $D_1 \ldots D_{10}$

For each D_i with $1 \leq i \leq 10$, we double the entire sample set to $2500 \times 2 = 5000$ samples and naively train on these new samples just like a regular `LwP` model, by using f_i to predict the labels of all of D_{i+1} and updating f_i on these samples. We get the following accuracies on the respective validation sets.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
f_1	85.36%									
f_2	84.68%	84.08%								
f_3	84.28%	83.64%	83.40%							
f_4	83.88%	83.24%	83.32%	83.12%						
f_5	83.44%	83.32%	83.72%	82.84%	83.56%					
f_6	82.76%	83.08%	83.36%	82.96%	82.60%	83.24%				
f_7	83.72%	82.76%	83.60%	82.04%	83.12%	82.76%	82.32%			
f_8	83.04%	82.40%	83.56%	82.56%	83.12%	83.48%	82.52%	82.68%		
f_9	82.20%	83.28%	82.16%	82.12%	83.44%	82.48%	82.20%	83.04%	82.20%	
f_{10}	83.92%	83.04%	83.36%	82.44%	84.04%	83.76%	83.16%	82.72%	82.28%	82.44%

Table 1: Accuracy on $D_1 \dots D_{10}$ trained with Sample Duplication

6.2 Results on $D_{11} \dots D20$

For each D_i with $11 \leq i \leq 20$ we update f_i to f_{i+1} by first performing a top-k confidence selection on D_{i+1}, and then performing sample duplication on these newly generated labeled samples. We then use this model to label the rest of the training points and finally update the model normally on these new labels. What follows is the performance of f_i with $11 \leq i \leq 20$

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
f_{11}	83.64%	83.20%	83.36%	82.52%	84.04%	83.80%	83.24%	83.28%	82.44%	82.80%
f_{12}	83.64%	83.32%	83.08%	82.28%	83.72%	83.40%	83.08%	82.84%	82.40%	82.60%
f_{13}	83.84%	83.52%	83.28%	82.52%	84.16%	83.76%	83.16%	82.68%	82.60%	82.72%
f_{14}	84.00%	83.92%	83.48%	82.64%	83.88%	83.88%	83.36%	82.80%	82.80%	82.88%
f_{15}	84.12%	84.12%	83.68%	83.80%	83.96%	83.16%	83.32%	83.00%	82.92%	83.92%
f_{16}	84.20%	84.28%	84.04%	83.36%	83.92%	84.08%	83.48%	83.60%	83.44%	83.16%
f_{17}	84.12%	84.44%	84.40%	83.44%	83.96%	84.20%	83.68%	83.56%	83.40%	83.28%
f_{18}	84.04%	84.36%	84.44%	83.52%	83.88%	84.40%	83.72%	83.72%	83.56%	83.32%
f_{19}	84.12%	84.36%	84.32%	83.60%	83.92%	84.20%	83.72%	83.60%	83.24%	83.56%
f_{20}	84.08%	84.56%	84.44%	83.64%	84.00%	84.44%	83.72%	83.64%	83.44%	83.76%

Table 2: Accuracy on $D_1 \dots D_{10}$ trained with Sample Duplication and Top-k Selection

	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}	D_{17}	D_{18}	D_{19}	D_{20}
f_{11}	68.04%									
f_{12}	68.68%	40.68%								
f_{13}	68.60%	41.76%	69.00%							
f_{14}	69.00%	42.16%	69.08%	77.36%						
f_{15}	69.32%	42.84%	69.00%	77.28%	83.28%					
f_{16}	69.56%	42.96%	69.08%	77.32%	83.52%	66.48%				
f_{17}	70.04%	43.60%	69.48%	77.92%	83.44%	66.76%	76.04%			
f_{18}	69.68%	44.92%	69.32%	77.76%	83.52%	66.92%	75.96%	69.08%		
f_{19}	69.60%	44.32%	69.68%	77.64%	83.52%	66.92%	75.72%	68.04%	61.00%	
f_{20}	69.64%	44.92%	69.32%	77.60%	83.76%	67.52%	75.76%	68.00%	60.92%	78.96%

Table 3: Accuracy on $D_{11} \dots D_{20}$ trained with Sample Duplication and Top-k Selection

6.3 Some Observations

Notice how the incremental model is more or less stable at its accuracies across datasets. Further, it even improves significantly on D_{12} as well as $D_1 \dots D_{10}$. We also remark here that the model's performance on the newer datsets $D_{11} \dots D_{20}$ is far, *far* better than just naive updates.

7 Closing Remarks

7.1 Why didn't you try xyz?

We probably did and it didn't work. Other than naive updates, we tried every possible permutation of 5.1, 5.2, as well as other filtering methods. The result we have presented is *the* best we have so far across all datasets.

7.2 What's Going on with D_{12}?

The short answer is that we do not know. The long answer is that we suspect it to be adversarial. During hyperparameter tuning with cosine similarity, we found something intriguing. Some hyperparameters significantly degraded the performance of f_{11} across all previous datasets, but the same hyperparameters *signijicantly improved* the performance of f_{12} across all its previous datasets. This only furthers our belief that the dataset itself is adversarial. Further, our presented model achieves an incrementally increasing accuracy on D_{12}, which is a lot more progress than we hoped for. Indeed, we have other models which achieve higher accuracies of up to 47%, but we choose to present this due to the incremental nature.

7.3 Horizons - Centroid $\cos$ Similarity

After applying 5.1 and 5.2, we have possibility of *further* improving our quasi-model before training on it. So, for now, entertain the "equation" $f_i \oplus \mathcal{M}_i \to f_{i+1}$, where $\mathcal{M}_i$ represents the model trained on confident predictions of f_i after sample duplication.

Suppose we have $2 \cdot k$ many labeled samples in $\mathcal{M}_i$. What we do next is take some k' many *unlabeled* samples from D_{i+1} that have the highest $\cos$ similarity with the class prototypes of $\mathcal{M}_i$. This ensures some more newer samples are *aligned well* with some class prototype. We can then update $\mathcal{M}_i$ on these samples with a $k-\texttt{NN}$ classifier and then use $f_i \oplus M_i$ to make further predictions. We have indeed implemented this, but couldn't find the right hyperparameters. We again refer back to [3]

7.4 Task 2

Find the video submission on [1]

References

[1] Video submission. `https://youtu.be/3ghjPtz-34E`.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun; PyTorch Vision Library. Resnet-34 documentation. `https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet34.html#torchvision.models.resnet34`, 2015.

[3] Chenxi Liu, Lixu Wang, Lingjuan Lyu, Chen Sun, Xiao Wang, and Qi Zhu; Northwestern University; SonyAI. Deja vu: Continual model generalization for unseen domains. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. Published as a conference paper at ICLR 2023.

[4] Sony Research. Ratp: Real-time and temporal processing for image generation. `https://github.com/SonyResearch/RaTP`, 2024.