CS771 Mini-Project 2

R.Charan 230819

Vedhanth Balasubramanian 231135

Aritra Ray 230193

Sanjna S 230918

Ashwin Barnwal 230234

1 Problem :1

We are provided with 20 datasets,$D_1, D_2.....D_{20}$ consisting of CIFAR-10 images. The first ten datasets have their inputs derived from similar distributions while the other ten are obtained from ten different distributions. Despite being obtained from different distributions they possess a certain level of similarity. Only the first dataset is labeled.

1.1 Task 1

In the first task, we attempted to train a LwP classifier on D_1, using the trained LwP to predict the labels of inputs from D_2, then trained the model on both of these datasets. We repeated the process for all the ten datasets. For task 1, at large we have the following 3 steps:

1.1.1 Extracting features using pretrained models:

Here we use the EfficientNet B4 model (imported from torchvision.models) to extract features from all our datasets resulting in a 1000-dimensional vector for each image. The only change to be made is to remove the last fully connected layer with an identity layer to get all the features as they are directly.

1.1.2 Preprocessing of the data:

Next we apply our own preprocessing pipeline with the following steps:

- Standardize each dataset to have a mean of 0 and standard deviation of 1 (using StandardScaler from sklearn.preprocessing). This is a required step for the next transformation.
- Perform Yeo-Johnson transformation which transforms the data to more closely follow a Gaussian distribution (using PowerTransformer from sklearn.preprocessing). This step improves the performance of the following steps. We used Quantile-Quantile (Q-Q) plot to measure how close the data follows a theoretical distribution (in this case a Gaussian). As shown, the data resembles a Gaussian much more closely after the transformations.

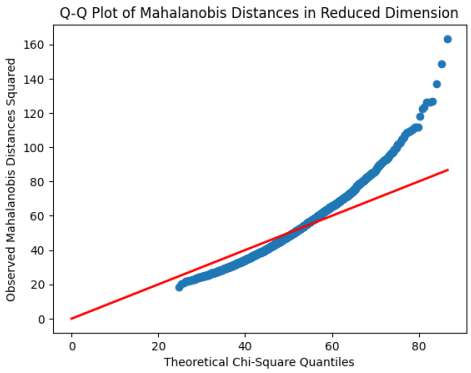

Figure 1: Before transformation

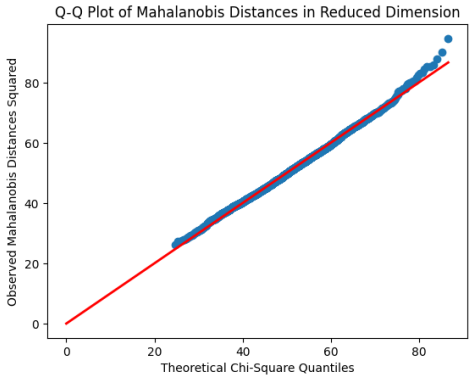

Figure 2: After transformation

- Applying PCA to reduce our feature vectors to 50 dimensions to speed up our process.
- Finally standardize the PCA-transformed data again.

During implementation we will fit all these transformations on a single dataset only (in this case D_1). Then for later set $D_2 - D_{10}$, we will use the already fitted objects to process these datasets, to ensure uniformity. Notably for the final standardization, we will use the mean and standard deviation of D_1, to transform the rest.

1.1.3 Training the model:

We have implemented Quadratic Discriminant Analysis (QDA), wherein we assume each class follows a multivariate Gaussian distribution. QDA estimates class-specific means and covariance matrices and uses Bayes' theorem to classify new data points based on posterior probabilities. We have implemented this as the class QDAClassifier with the following 3 functions:

- **Fit:** Takes a set of inputs and labels as parameters and trains the model on a dataset with labels (in this case D1). It simply calculates the means (as in an LwP model), covariance of members, counts, and priors for each class according to the given labels.
- **Predict:** Takes a set of inputs as parameters and makes predictions on it as so:
  - Compute the likelihood for each class using the multivariate Gaussian probability density function (PDF) (using scipy.stats.multivariate_normal).
  - Multiply the likelihood by the prior probability to compute the posterior probability for each class.
  - Assign the class with the highest posterior probability to the data point.

  The function uses Bayes' rule: P(y=clx) $\propto$ P(xly=c)P(y=c)
- **Update:** Takes a set of inputs as parameters and uses it to refine the model as so:
  - First make predictions on the dataset using the previously defined **predict** function.
  - Now treating the input parameters and the predicted labels as another training set, we update the class means as the weighted average of the current mean and the sum of the new members of the class.
  - New class counts are just old counts + new members, priors are calculated similarly. Notably, new covariance is calculated as the weighted average of the old covariance and the covariance of the newly classified points.
  - These updated parameters now define our new model and can be used accordingly.

Thus to get our outputs we first extract the data from the 10 datasets, then pass them through our preprocessing pipeline to get the final datasets to be used. Then we fit the model on D_1, and for later iterations update them on $D_2 - D_{10}$ to get 10 models.

```
[[85.88  0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [84.96 85.8   0.    0.    0.    0.    0.    0.    0.    0.  ]
 [84.24 85.6  85.04  0.    0.    0.    0.    0.    0.    0.  ]
 [83.8  85.4  84.24 84.28  0.    0.    0.    0.    0.    0.  ]
 [83.52 85.4  84.04 83.52 84.4   0.    0.    0.    0.    0.  ]
 [83.24 84.72 83.4  83.44 84.2  85.2   0.    0.    0.    0.  ]
 [83.2  84.56 83.44 83.12 84.04 84.8  83.08  0.    0.    0.  ]
 [83.16 84.12 82.92 82.88 83.64 84.36 82.6  84.08  0.    0.  ]
 [82.96 84.08 82.64 82.84 83.44 84.2  82.6  83.88 82.72  0.  ]
 [82.84 84.04 82.52 82.72 83.32 83.8  82.44 83.64 82.72 83.8 ]]
```

Figure 3: Final Output

1.2 Task 2

This task is similar to the first one, the difference is that the inputs are drawn from different distributions and therefore we need more powerful methods to tackle these distribution shifts. For Task 2, we majorly tried 4 methods :

- Pseudo-dataset generation and then applying Gaussian Mixture Models using Expectation Maximization
- KL-Divergence minimization between new datasets and pseudo-generated dataset
- Loss function minimization (as described in Paper 2)
- Weighted QDA approach, using modified update function

We finally settled for the Weighted QDA approach as we obtained the best results from it compared to the other two.

Here our process involves the exact same steps for **feature extraction** and **preprocessing** of the data.

1.3 Weighted QDA approach

Here our process involves the exact same steps for **feature extraction** and **preprocessing of the data**. The difference from the first task's method is outlined below.

Modified update (mod_update):

The need for this function arises from the fact that the model will struggle to adapt to the newer datasets (say D_{11}) once it has trained on the previous datasets D_1-D_{10} using our old update function. This is because D_1-D_{10} belong to the same distribution and once the model has trained on all 10 of them, it is ineffective to try and skew its parameters using a single dataset with a different distribution. Thus the modified approach is to instead weight and average the class prototypes using parameters independent of the number of datasets currently already tested, this allows the later datasets from different distributions to have more influence on the final model. We have computed weighted average of the new models' parameters and the old parameters, giving weight h = 0.1 (value chosen by manual cross validation) to the new parameters and 0.9 to the old.

The workflow is as so:

- Extract data using **EfficientNet 4B** and preprocess it using our **PreprocessingPipeline**.
- **Fit** on D_1, and then further **update** on D_2-D_{10}
- Use **update** on sets D_{11}-D_{20} then use the produced models to test on the evaluations sets accordingly.

1.3.1 Final Output:

(each row represents a model, each column represents the dataset)

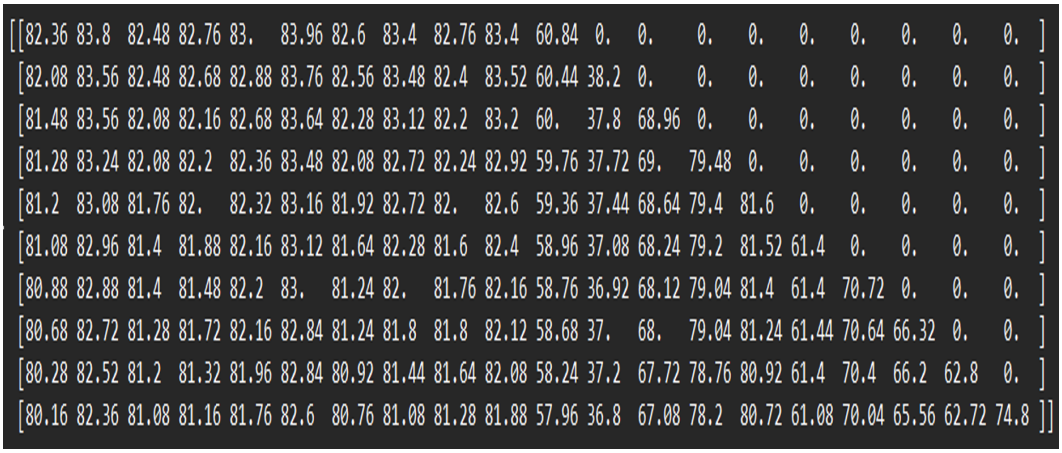

Figure 4: Final Output

1.3.2 Other Approaches:

Expectation-Maximization and GMMs: This approach involved the following steps:

- Generating n samples from the current parameters in the model, assuming each class to be a Gaussian. The augmented set will be comprised of these generated samples, with the number belonging to each class being proportional to the class's prior.
- Concatenating these samples with the new dataset to be updated (say D_{11} when the current model is f_{10}), then estimating the means, covariances and priors of this new dataset.
- Within the expectation_maximization function, we have initialized the parameters as the values found out in the previous step (an educated guess), then alternatively optimised the class labels assigned and the parameters.
- Finally we updated the final model parameters weighing with the number of past examples and new examples, and tested it on the dataset.

Although this was a standard and promising approach, we obtained about 60 percent accuracy on the new datasets, hence we abandoned it.

Minimising KL Divergence: In this approach we first generated a pseudo-dataset using the stored Gaussian parameters, then we sought to minimize the KL divergence between the generated dataset and the new dataset belonging to a different probability distribution, finally transforming the new dataset to a distribution similar as the source domain. Then we applied the normal QDA model as in Task 1. Unfortunately, despite the novelty of this approach, we obtained just about 45 percent accuracy, hence we rejected it.

Loss Function Minimization (Paper 2) : Steps are as follows :

- **Pseudo-Label Computation:**Pseudo-labels are assigned using the Euclidean distance between data points and class means:

$$\text{pseudo\_labels}[i] = \arg\min_k \|X_i - \mu_k\|_2$$

- **Loss Function Design:**The loss function optimizes class means with three components: log-scaled distance, combined distances, and mean separation.

– **Log-Scaled Distance:**Minimizes the negative log probability of aligning data points with pseudo-labels:

$$\text{LCE} = \mathbb{E}_{x_i \sim P_{X_t}} \left[-\log \left(\frac{\exp\left(-\|p_k^t - x_i\|_2^2\right)}{\sum_{c=1}^{K} \exp\left(-\|p_c^t - x_i\|_2^2\right)} \right) \right],$$

– **Combined Distances:** Brings updated means closer to older ones while maximizing inter-class distances:

$$\text{LPCA} = \mathbb{E}_{x_i \sim P_{X_t}} \left\{ -\log \left[\frac{\mathbb{I}_{\hat{y}=k} \cdot \left(\exp\left(-\|p_k^t - x_i\|_2^2\right) + \exp\left(-\|p_k^{t-1} - x_i\|_2^2\right)\right)}{\Delta} \right] \right\},$$

where:

$$\Delta = \sum_{c=1}^{K} \exp\left(-\|p_c^t - x_i\|_2^2\right) + \sum_{c=1}^{K} \exp\left(-\|p_c^{t-1} - x_i\|_2^2\right) + \sum_{x_j \in \mathcal{T}, j \neq i} \mathbb{I}_{\hat{y} \neq \hat{y}'} \exp\left(-\|x_i - x_j\|_2^2\right).$$

– **Regularization Term**: Ensures separation of updated means:

$$\text{Regularization} = \sum_{i \neq j} \frac{1}{\|\mu_i - \mu_j\|_2 + \epsilon},$$

- **Optimization Strategy :** Stochastic Gradient Descent (SGD) is used with normalization after each step for stability.
- **Results:** Despite optimization, accuracy on the evaluation dataset remained low at 35

2 Task 2:

Link for the youtube video explaining "Deja-Vu Continual Model Generalization for unseen domains": `https://youtu.be/YQliswQ5oKQ?si=S31BVc9kLy17x9nB` The link for the feature extraction model: `https://github.com/AshwinBarnwal/CS771-mini-project-2`

Acknowledgments

We would like to express our gratitude to Prof. Piyush Rai for the guidance and valuable insights provided throughout this project. We would also like to thank the teaching assistants for their assistance with the datasets and experiments.

References

[1] Jason Brownlee. How to transform data to fit the normal distribution. `https://machinelearningmastery.com/how-to-transform-data-to-fit-the-normal-distribution/`, 2019.

[2] Lingjuan Lyu Chen Sun Xiao Wang Qi Zhu1 Chenxi Liu, Lixu Wang. Deja vu: Continual model generalization for unseen domains. `https://arxiv.org/pdf/2301.10418`, 2023.

[3] SciPy community. scipy.stats.yeojohnson. {https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.yeojohnson.htm 2024.

[4] USC Information Sciences Institute Mohammad Rostami. Lifelong domain adaptation via consolidated internal distribution. `https://proceedings.neurips.cc/paper_files/paper/2021/file/5caf41d62364d5b41a893adc1a9dd5d4-Paper.pdf`, 2021.

[5] Scikit-learn Developers. Scikit-learn: User guide. `https://scikit-learn.org/stable/user_guide.html`, 2024.

[5] [1] [3] [2] [4].