Indian Institute of Technology Madras

CS6910 Deep Learning

Assignment 2

Vimal Suresh Mollyn ED17B055
Dyava Naveen Reddy ME17B140
Ramakrishnan NA18B030

March 24, 2022

Vimal Suresh Mollyn, ED17B055
Dyava Naveen Reddy, ME17B140
Ramakrishnan, NA18B030

Contents

1 Task 1

Dimension reduction on Dataset 1 using (a) PCA and (2) AANN. Use the reduced dimension representation as input to a MLFFNN based single-label multi-class classification model.

1.1 Dimensionality Reduction

For this question, we are asked to compare two different forms of dimensionality reduction - a linear one (PCA) and a non-linear one (Auto Association Neural Network). The original input data is of 48 dimensions ($\mathbb{R}^{48}$). Since this is histogram data, we normalize the inputs by dividing by the image dimensions ($210*300 = 63000$). Next we use PCA in order to find a suitable dimension to reduce the data into. We then investigate the performance of the PCA and AANN methods for dimensionality reduction on the multiclass classification problem.

1.2 PCA

The sklearn implementation of principal compenent analysis was used. We first applied PCA to the training data, and extracted features from the top N principal directions, as per the reduced dimension. A plot of the explained variance vs the number of principal components is shown in Figure 1. As we can see in this figure, to explain 95% of the variance, we need only 25 PCA components. Thus, we choose $N = 25$ as the reduced dimension size.

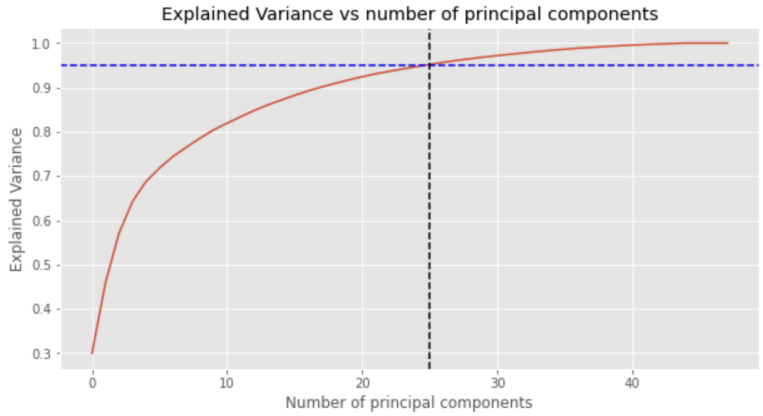

Figure 1: Explained variance from Principal Component Analysis (PCA)

1.3 AANN

Auto associative neural networks (AANNs) are a type of neural network whose output is the same as the input. Such networks can be used for dimensionality reduction. These networks consist of an *encoder*, that takes the input data and reduces it's dimension, and a *decoder*, that takes the takes the reduced dimension data and attempts to decompress it. Following our analysis from PCA, we decided to choose $N = 20$ as the bottleneck dimension size, so that we could test out the compressive power of a single AANN.
Our AANN consists of 2 parts, the encoder and the decoder. In total, there are 5 layers:

1. Encoder
   (a) Linear Input Layer (48)
   (b) Non Linear Hidden Layer (32, tanh activated)
   (c) Linear Latent Layer (20)
2. Decoder
   (a) Non Linear Hidden Layer (32, tanh activated)
   (b) Linear Output Layer (48)

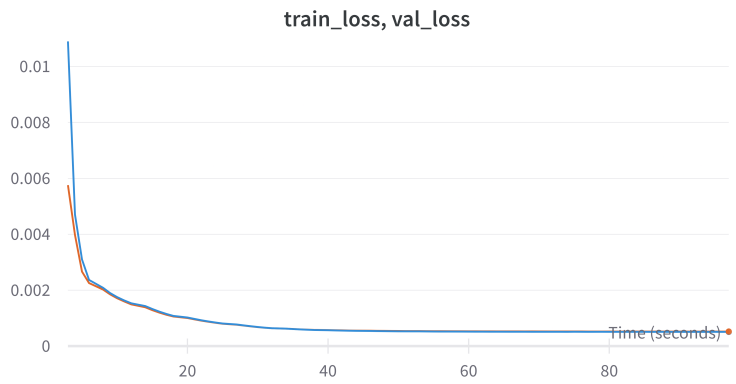

Figure 2: Training Curve of the AANN Model - Train Loss in Blue, Val loss in Orange

We used the Adam optimzer with a learning rate of $3e-4$ along with the Mean Squared Error Loss in order to train the network. Furthermore, we used an Early Stopping Callback in order prevent overfitting. Training was performed in pytorch with the pytorch-lightning package. Results are plotted in Figure **??**.

1.4 MLFFNN Classifier with Reduced dimension features

After reducing the dimensionality of the input data, we train 2 Multi Layer Feed Forward Neural Networks (2 layers, (32, 16 hidden units), tanh activated) for each of the dimensionality reduction methods. Training results are shown in Figures **??**

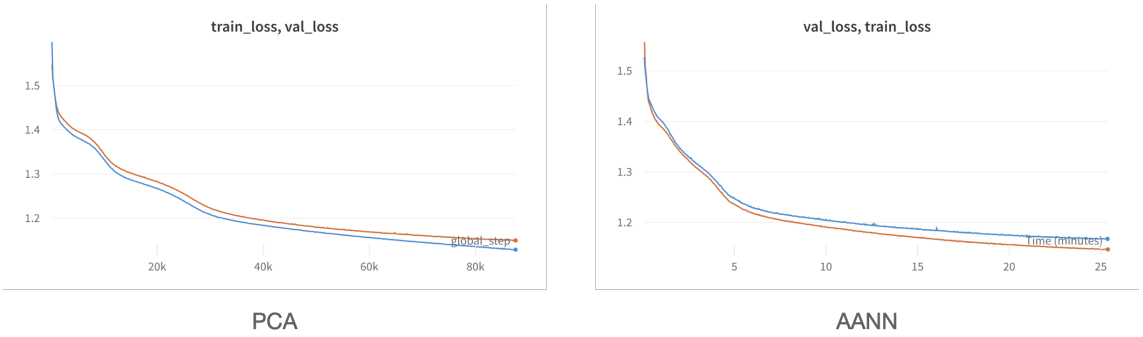

Figure 3: Training Curve of the MLFFNN Model for both dimensionality reduction methods - Train Loss in Blue, Val loss in Orange

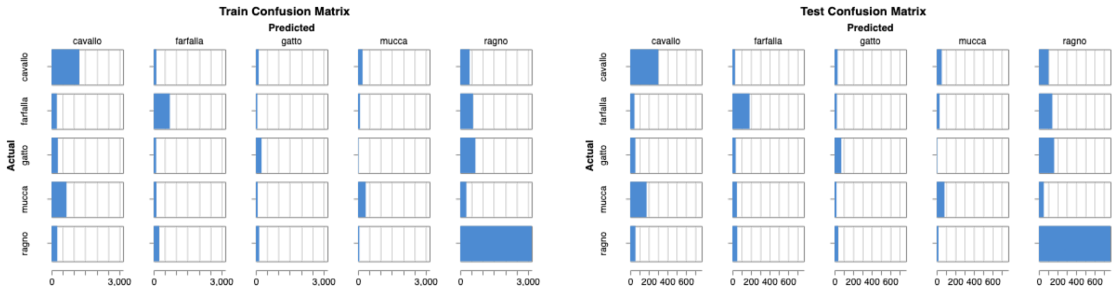

Figure 4: Confusion Matrices for PCA Based Dimensionality Reduction

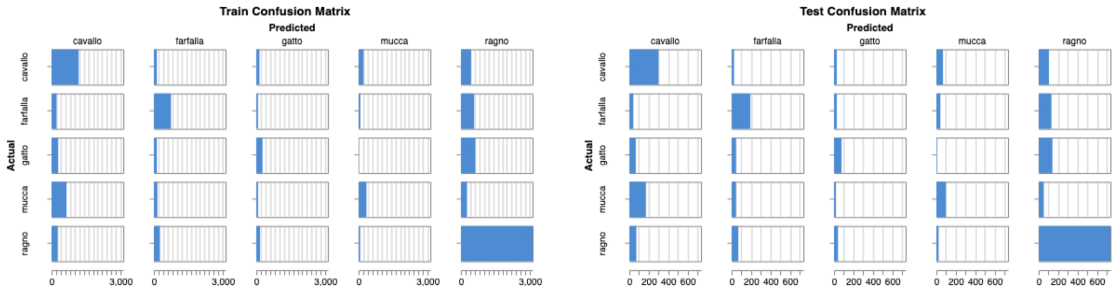

Figure 5: Confusion Matrices for AANN Based Dimensionality Reduction

1.5 Observations

Overall, we observe that both methods achieve comparable validation accuracies of 54.4% and 53.8% respectively. Crucially, the AANN was able to compress the data

to only 20 dimensions, whereas the PCA based method was only able to compress the data to 25 dimensions. Thus, with 20% less data, the AANN is able to still able capture the disciminatory features of each of the classes. What's also interesting is that from the Confusion Matrices (Figures 5, 4) we can see that both the models seemed to learn how to classify the *ragno* class really well, which accounts for most of their performance.

2 Task 2

2.1 Description about the dataset -1

2.2 Training Process

The objective of the task 2 is to perform Stacked Autoencoder (with 3 autoencoders) based pre-training of a DNN based classifier for Dataset 1. In order to perform stacked autoencoder based pre-training the follow procedure is adopted:

- Autoencoder-1 model is trained first using the dataset-1 directly.
- Autoencoder-2 model is trained using the features output by the encoder part of Autoencoder-1. In this case the parameters of the Autoencoder-1 model are frozen.
- Autoencoder-3 model is using the features output by the encoder parts of Autoencoder-1 and Autoencoder-2 stacked together. The parameters of first and second autoencoders are frozen during this process
- Encoders modules of Autoencoder-1, Autoencoder-2 & Autoencoder-3 are stacked together. The features extracted from this stack is fed into a Deep Neural Network Classifier and the whole model is fine-tuned for classification task. Note that in this case the parameters of Autoencoder-1, Autoencoder-2 & Autoencoder-3 are not frozen.

We have performed experiments on two different sized autoencoders. We have chosen autoencoders with 1 hidden layer and 2 hidden layers. To reduce the complexity of the experiment we have performed experiments either by stacking autoencoders with 1 hidden layers or autoencoders with 2 hidden layers together. Though mix of both can be used, to reduce the complexity of the experiment we have stacked only autoencoders with same number of hidden layers together.
We have performed total 6 sets of hyperparameter tuning experiments. Hyperparameter tuning is done using grid search. The following is the list of the hyperparameter tuning experiments:

- Autoencoder-1 with 1 hidden layer. Hyperparameters are number of units in the hidden layer & no. of units in the bottle neck layer.

- Autoencoder-1 with 2 hidden layers. Hyperparameters are number of units in the hidden layers & no. of units in the bottle neck layer.
- Autoencoder-2 with 1 hidden layer. Hyperparameters are number of units in the hidden layer & no. of units in the bottle neck layer.
- Autoencoder-2 with 2 hidden layers. Hyperparameters are number of units in the hidden layers & no. of units in the bottle neck layer.
- Autoencoder-3 with 1 hidden layer. Hyperparameters are number of units in the hidden layer & no. of units in the bottle neck layer.
- Autoencoder-3 with 2 hidden layers. Hyperparameters are number of units in the hidden layers & no. of units in the bottle neck layer.

Note:

- The given dataset is split into training and validation in the ratio of 70% : 30% respectively
- PyTorch-lightning is used as a frame work for building the models.
- Weights Biases tool is used to keep track visualize various model parameters and performance metrics during the hyperparameter tuning.
- The data from various csv files is merged into one data frame and custom labels were given
- The mean & variance of the merged data is calculated it is used for normalizing the data before it is fed into the model
- While selecting the the number of units in each layer for the auto encoders, the values are carefully chosen so that compression is achieved and the value of the number of layers in the subsequent autoencoders is dependent on the best best value of the bottleneck units achieved during hyperparameter tuning of the current autoencoders. For this reason to reduce the complexity of the the process either autoencoder models with 1-layers and 2-layers are stacked separately for the process.

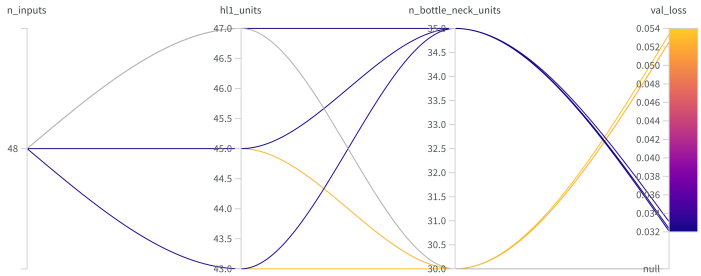

Figure 6: Parallel Coordinates plot-hyperparameter tuning of Autoencoder-1 with 1 hidden layer

2.3 Autoencoder - 1

2.3.1 Autoencoder - 1 with 1 hidden layer

Grid search parameters:

- Hidden layer-1: [43, 45, 47]
- Bottleneck layer units: [30, 35]

Best parameters found:

- Hidden layer-1: 47
- Bottleneck layer units: 35
- Validation Loss: 0.03207

2.3.2 Autoencoder - 1 with 2 hidden layers

Grid search parameters:

- Hidden layer-1: [43, 45, 47]
- Hidden layer-2: [37, 39, 41]
- Bottleneck layer units: [30, 35]

Best parameters found:

- Hidden layer-1: 47
- Hidden layer-2: 41

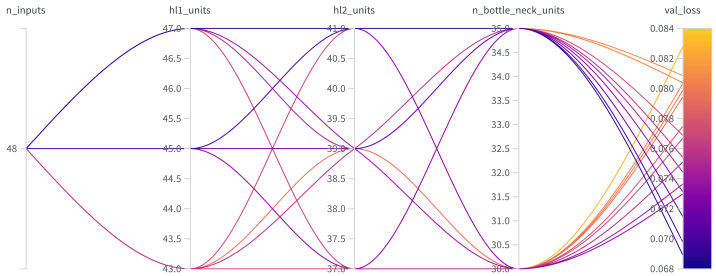

Figure 7: Parallel Coordinates plot-hyperparameter tuning of Autoencoder-1 with 2 hidden layers

- Bottleneck layer units: 35
- Validation Loss: 0.06889

2.4 Autoencoder - 2

2.4.1 Autoencoder - 2 with 1 hidden layer

Grid search parameters:

- Hidden layer-1: [27, 29, 31, 33]
- Bottleneck layer units: [20, 23, 25]

Best parameters found:

- Hidden layer-1: 33
- Bottleneck layer units: 25
- Validation Loss: 0.003005

2.4.2 Autoencoder - 2 with 2 hidden layers

Grid search parameters:

- Hidden layer-1: [30, 31, 33]
- Hidden layer-2: [26 27, 29]
- Bottleneck layer units: [20, 23, 25]

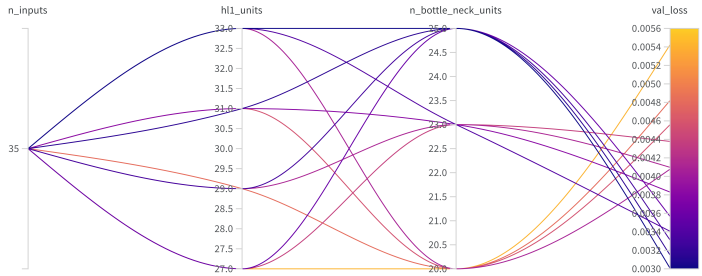

Figure 8: Parallel Coordinates plot-hyperparameter tuning of Autoencoder-2 with 1 hidden layer

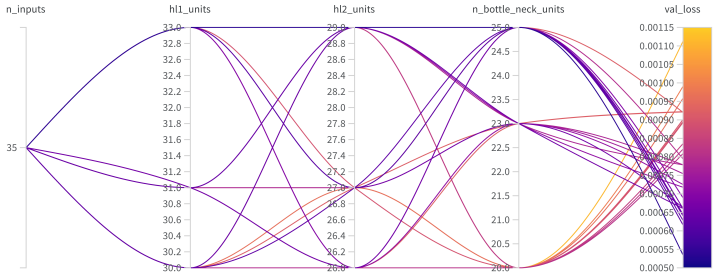

Figure 9: Parallel Coordinates plot-hyperparameter tuning of Autoencoder-2 with 2 hidden layers

Best parameters found:

- Hidden layer-1: 33
- Hidden layer-2: 29
- Bottleneck layer units: 25
- Validation Loss: 0.0005344

2.5 Autoencoder - 3

2.5.1 Autoencoder - 3 with 1 hidden layer

Grid search parameters:

- Hidden layer-1: [17, 19, 21, 23]
- Bottleneck layer units: [10, 13, 15]

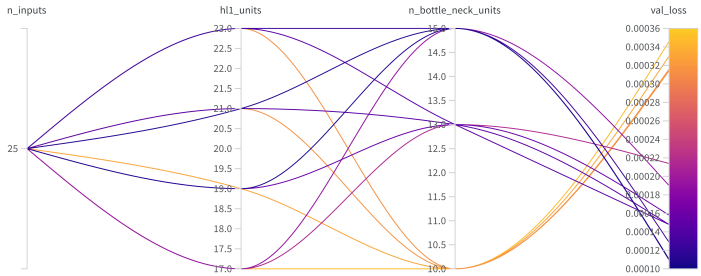

Figure 10: Parallel Coordinates plot-hyperparameter tuning of Autoencoder-3 with 1 hidden layer

Best parameters found:

- Hidden layer-1: 19
- Bottleneck layer units: 15
- Validation Loss: 0.0001096

2.5.2 Autoencoder - 3 with 2 hidden layers

Grid search parameters:

- Hidden layer-1: [19, 21, 23]
- Hidden layer-2: [15, 16, 18]
- Bottleneck layer units: [10, 13, 18]

Best parameters found:

- Hidden layer-1: 19
- Hidden layer-2: 18
- Bottleneck layer units: 13
- Validation Loss: 0.00002404

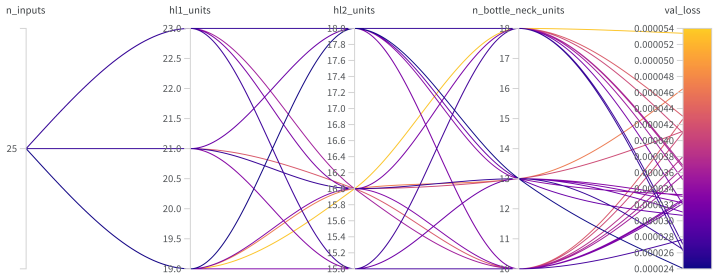

Figure 11: Parallel Coordinates plot-hyperparameter tuning of Autoencoder-3 with 2 hidden layers

2.6 Full Model

After training the 3 autoencoder models they are stacked together and combined with the classifier Feed Forward Neural Network to perform the classification task. Now the whole model is trained as a classifier model.
Note:

- Various architectures were tried for the final classifier model. It is observed that very similar results were achieved for different number of layers and different number of units in each of these layers.
- All the 3 stacked autoencoders are either having 1 hidden layers or 2 hidden layers.

Finally the stacked Autoencoder model with 2 hidden layers is used. The three autoencoders with 2 hidden layers in each are stacked and a final classifier model is used. This decision is made based on the accuracy achieved.
The final model is summarised as follows:

- Input - 48 units
- AE-1 HL-1 - 47 units - Tanh
- AE-1 HL-2 - 41 units - Tanh
- AE-1 Bottle Neck Layer - 35 units - Linear
- AE-2 HL-1 - 33 units - Tanh
- AE-2 HL-2 - 29 units - Tanh

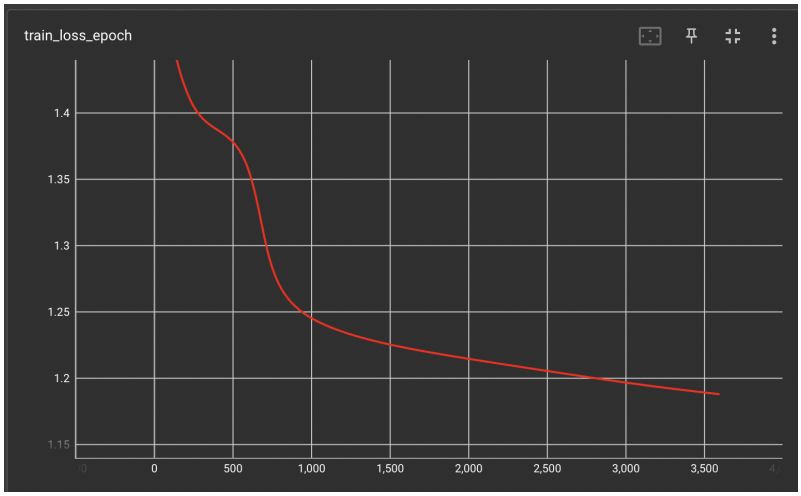

Figure 12: Training loss vs steps

- AE-2 Bottle Neck Layer - 25 units - Linear
- AE-3 HL-1 - 19 units - Tanh
- AE-3 HL-2 - 18 units - Tanh
- AE-3 Bottle Neck Layer - 13 units - Linear
- Classifier Hidden Layer - 30 units - Tanh
- Output Layer - 5 units - Linear

2.7 Results

The validation accuracy achieved: 52
The training accuracy achieved: 51
Figures 10 - 15 will show the metrics of the final model

2.8 Conclusion

- The accuracies achieved are very low using image histograms as features
- Another interesting observation is that the model classified most of the images to one particular class - "rango". Most of the images in "rango class" are classified correctly so there is a substantial increase in accuracy.

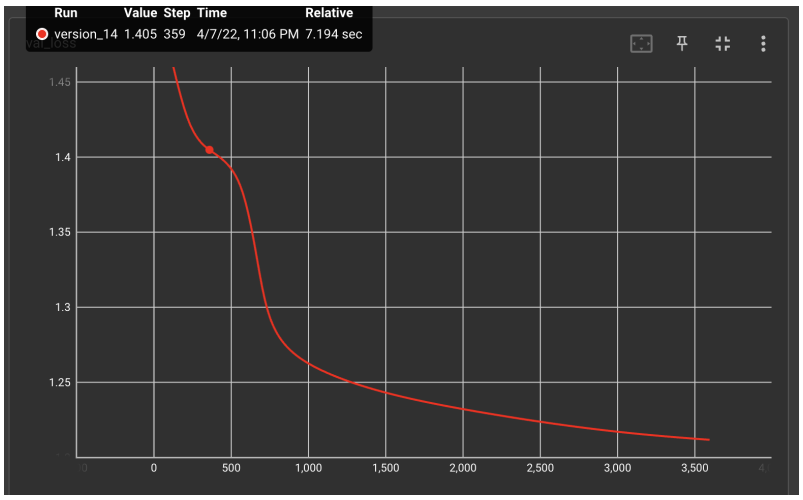

Figure 13: validation loss vs steps

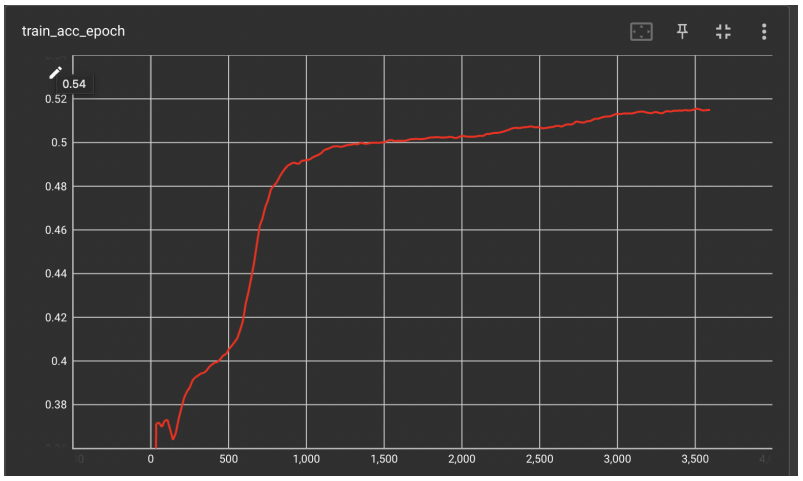

Figure 14: Training accuracy vs steps

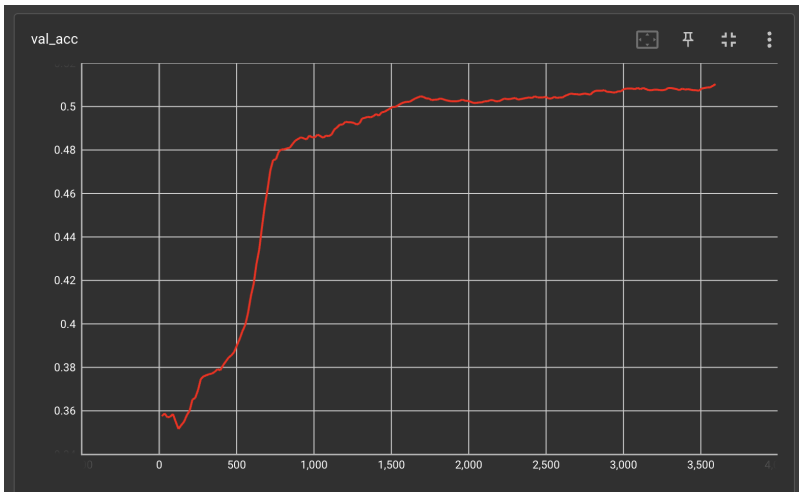

Figure 15: Validation accuracy vs steps

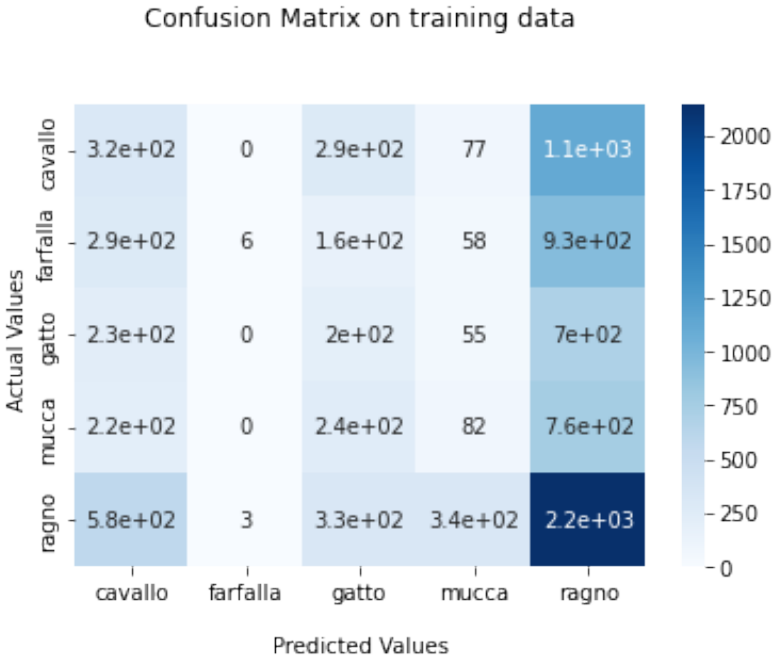

Figure 16: Confusion matrix on training data

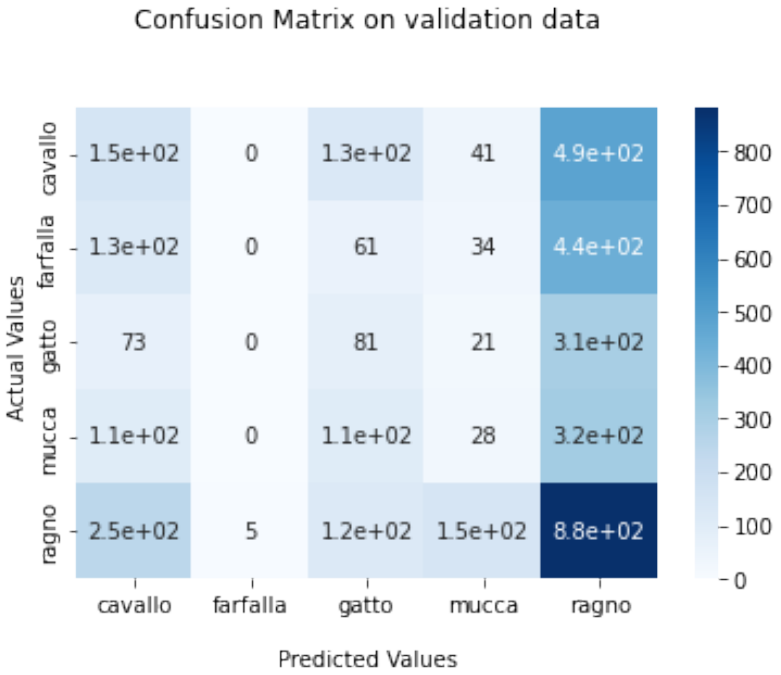

Figure 17: Confusion matrix on validation data

- But the number of images which are wrongly classified are more than number of images which are correctly classified.
- There is a lot of imbalance in the classification, which can be verified from the confusion matrices.
- So my conclusion is that the actual accuracy can be much lower that what we got here using this method.
- It can be concluded that using image histograms to classify images is not a very suggestible method.

3 Task 3

Image classification for Dataset 2, using a MLFFNN with Deep CNN features for an image as the input to the MLFFNN, with (a) VGGNet as Deep CNN and (b) GoogLeNet as Deep CNN.

3.1 Using Deep CNNs as Feature Extractors

In recent years, many Deep CNNs have been trained to classify images. A CNN trained for image classification on large datasets will have learned generic image filters in it's layers - whose outputs can be used as inputs to another Multi Layer Feed Forward Neural Network (MLFFNN). Figure 18 shows a couple of example filters from the first few layers of AlexNet - including generic filters such as various edge detectors, color detectors, etc.
In this assignment, we tested out 2 such Deep CNNs – VGG16 & GoogLeNet – as feature extractors.

3.2 Extracting Outputs from the Final Layer

We used the pretrained models for VGG16 and GoogLeNet from the *torchvision* library. Each of these models is structured differently, so as the first step, we had to search for the penultimate layer in order to extract the featurized outputs. This is well documented in our code (search for "backbone model"). Furthermore, since we are using these models solely as feature extractors, we do not want to propagate gradients back into those layers. Hence, during forward propagation, we don't compute gradients for those layers.

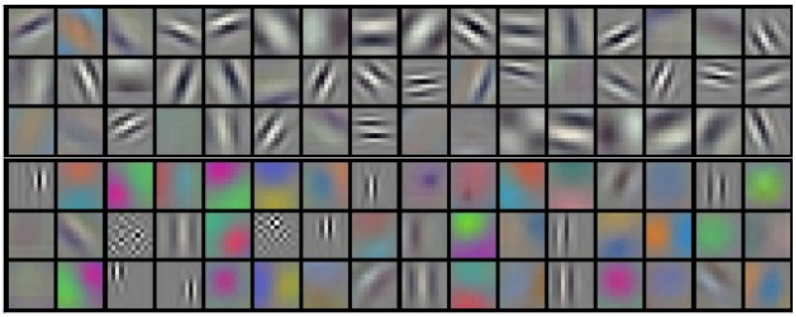

Figure 18: Example filters learned by Krizhevsky et al.

3.3 Model and Performance on classification

We then fed these extracted features into a MLFFNN with 2 hidden layers of 1000 nodes each, activated by Tanh. The Adam Optimizer with a learning rate of $3e-4$ was used in conjunction with an Early Stopping callback in order to prevent overfitting. Results are shown in Figures 19 & 20.

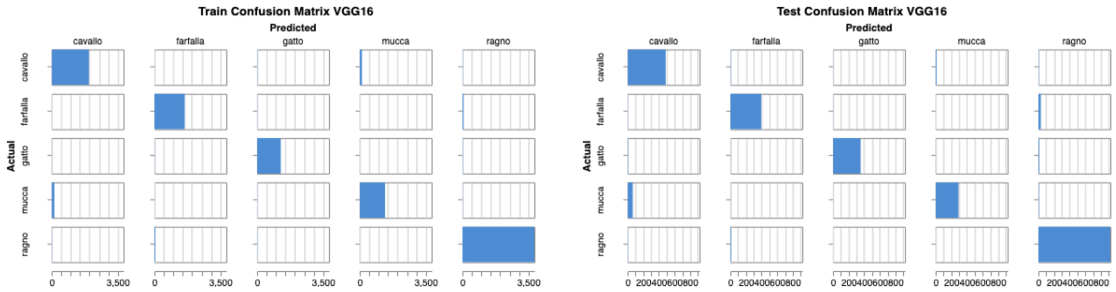

Figure 19: Confusion Matrix for VGG16 Backbone

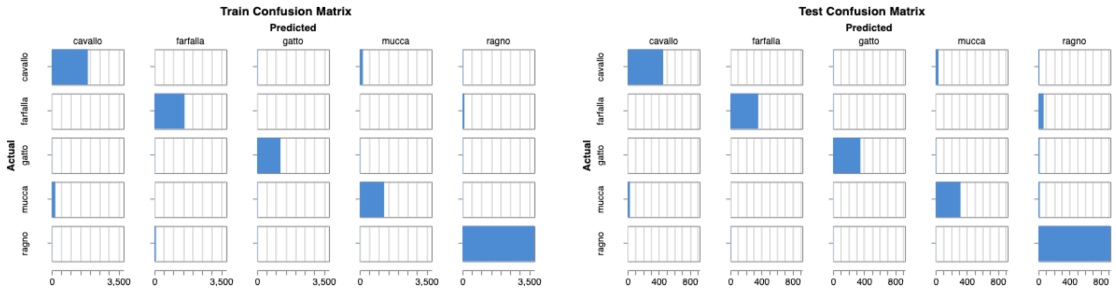

Figure 20: Confusion Matrix for GoogLeNet Backbone

Overall, we were able to achieve a 93.2% and 94.3% accuracy respectively using GoogLeNet and VGG16 as the backbones.

4 Task 4

Image classification for Dataset 2, using a CNN with CL1, PL1, CL2 and PL2 as the layers. Use kernels of size 3x3, stride of 1 in the convolutional layers. Use the mean pooling with a kernel size of 2x2 and stride of 2 in the pooling layers. Use 4 feature maps in CL1. The number of feature maps in CL2 is a hyperparameter.

4.1 CNN

Convolutional neural networks is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery. In our CNN architecture, we used three types of layers

- Convolutional layer
- Mean Pooling layer
- Fully-connected (FC) layer

4.2 Description

- This section describes the implementation of CNN for classification task of image data into one of the five classes, namely ('cavallo','farafalla','gatto','mucca','ragno')
- We have used 80:20 split to split the training and validation data into similar distribution based on the output labels
- The early stopping is set such that when the validation error stops decreasing we stop the training of model
- The CNN architecture is as follows
  - Input of dimension (3, 210, 300)
  - Convolution layer 1 of (inchannels=3, outchannels=4, kernelsize=3, stride=1)
  - Relu activation applied on output of convolution layer 1
  - Mean pooling of (kernelsize=2, stride=2)
  - Convolution layer 2 of (inchannels=4, outchannels=hp1, kernelsize=3, stride=1)

  - Relu activation applied on output of convolution layer 2
  - Mean pooling of (kernelsize=2, stride=2)
  - Fully-connected layer 1 with (input dimension = hp1*3723, output dimension = hp2)
  - Relu activation applied on output of fully-connected layer 1
  - Fully-connected layer 2 with (input dimension = hp2, output dimension = hp3)
  - Relu activation applied on output of fully-connected layer 2
  - Fully-connected layer 1 with (input dimension = hp3, output dimension = 5)
- Loss function used : Cross-entropy loss
- Optimizer used : Stochiastic gradient descent with learning rate (lr) and momentum (α)
- Hyperparameters :
  - hp1 : outchannels of convolution layer 2
  - hp2 : number of nodes in fully-connected layer 1
  - hp3 : number of layer in fully-connected layer 2
  - hp4 : learning rate (lr)
  - hp5 : momentum (α)

4.3 Observations

- We explored the following set of hyperparameters to find the best fit based on validation loss
- Number of filter layers in convolution layer 2 : [4, 8, 16]
- Number of nodes in fully connected layer 1 : [1024, 512]
- Number of nodes in fully connected layer 2 : [100, 50]
- Learning rate : [0.005, 0.01]
- Momentum : [0.9]

- The following plots are the evolution of training loss and validation loss with epochs for different combination of hyperparameters

4.4 Plot of training loss and validation loss with epochs

Results for 512 nodes in FC1 and 100 nodes in FC2 and 4 filter layers in CL1 and learning rate 5e-3

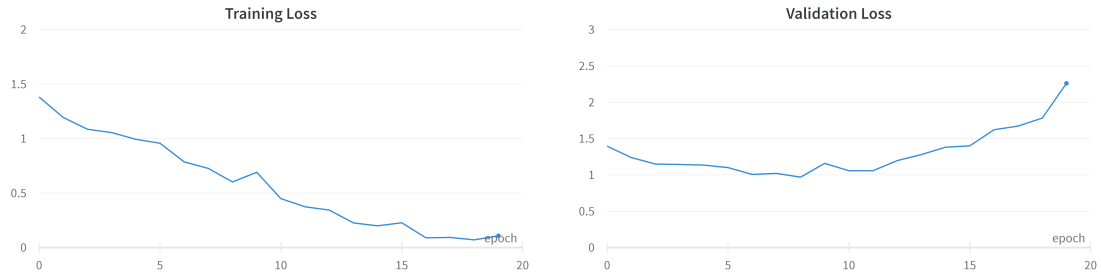

Figure 21: Average loss vs. epoch

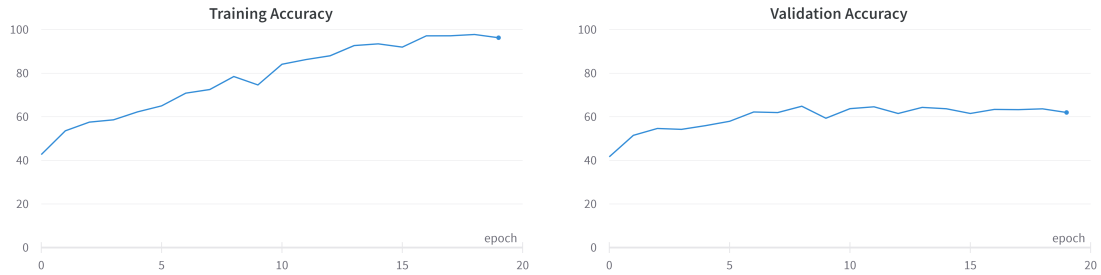

Figure 22: Accuracy vs. epoch

Results for 1024 nodes in FC1 and 50 nodes in FC2 and 4 filter layers in CL1 and learning rate 5e-3

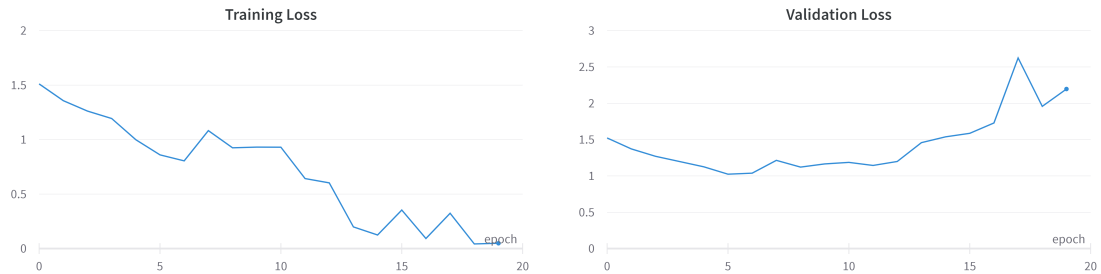

Figure 23: Average loss vs. epoch

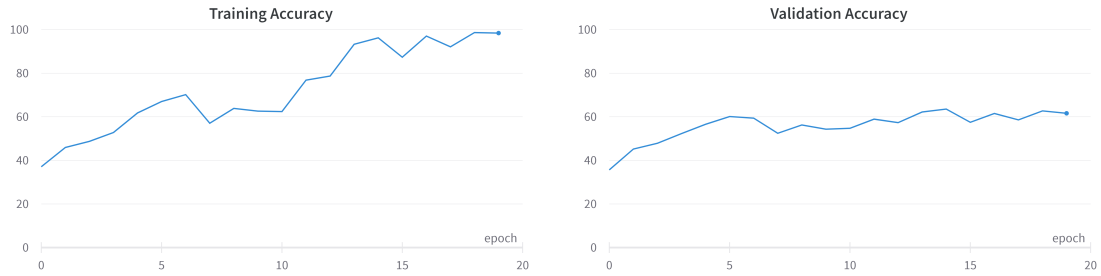

Figure 24: Accuracy vs. epoch

Results for 512 nodes in FC1 and 50 nodes in FC2 and 16 filter layers in CL1 and learning rate 10e-3

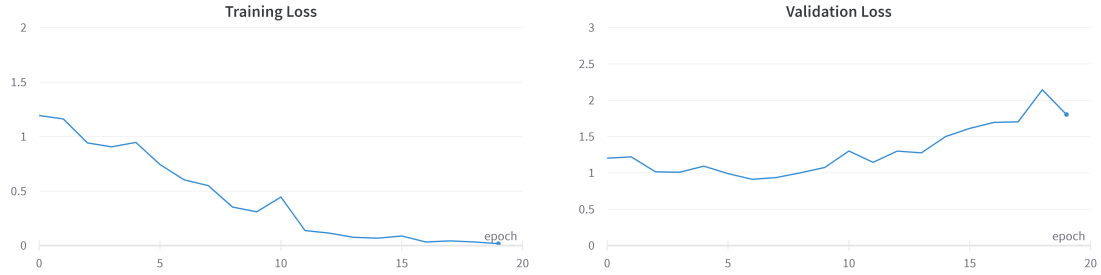

Figure 25: Average loss vs. epoch

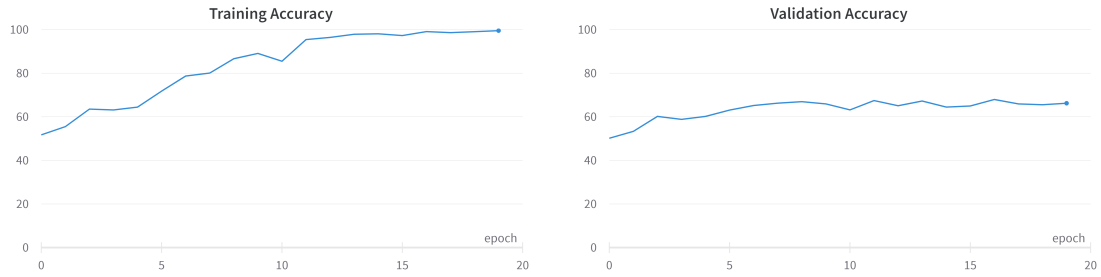

Figure 26: Accuracy vs. epoch

Results for 512 nodes in FC1 and 100 nodes in FC2 and 8 filter layers in CL1 and learning rate 5e-3

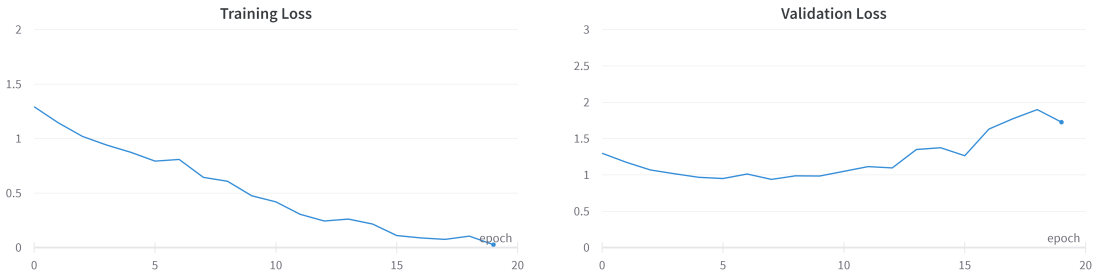

Figure 27: Average loss vs. epoch

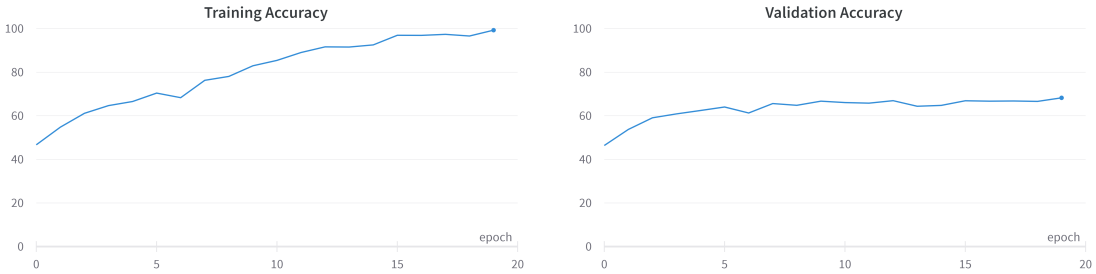

Figure 28: Accuracy vs. epoch

4.5 Summary

FC1	FC2	FM2	lr	Train error	Train accuracy	Val error	Val accuracy	Epochs
512	100	4	0.005	0.60	78.48	0.96	64.84	9
1024	50	4	0.005	0.65	78.95	1.013	62.09	9
512	50	16	0.01	0.54	80.06	0.93	66.25	7
512	100	8	0.005	0.79	70.41	0.95	64.04	6

4.6 Conclusion

Based on the observations we conclude that the following set of implementation gives the best result

- Number of filter layers in convolution layer 2 : 16
- Number of nodes in fully connected layer 1 : 512
- Number of nodes in fully connected layer 2 : 50
- Learning rate : 0.005

- Momentum : 0.9

Results:

- Training error : 0.54
- Training accuracy : 80.06
- Validation error : 0.93
- Validation accuracy : 66.25

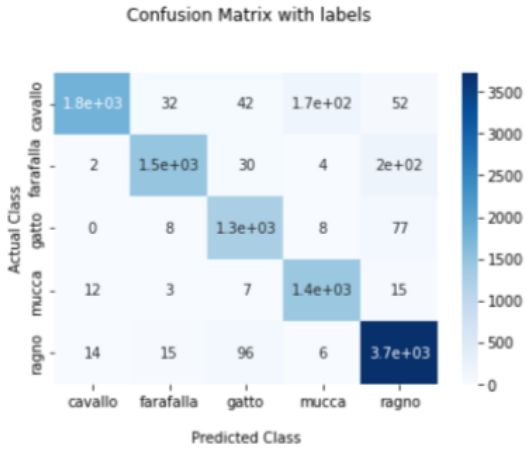

Figure 29: Confusion matrix on training data

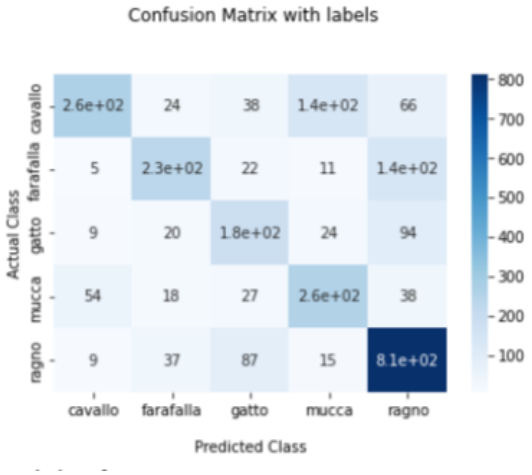

Figure 30: Confusion matrix on validation data

In comparison to the model trained with a Deep CNN Backbone (Task3), we notice much worse performance. This is probably due to the fact that:

1. Those backbones were trained on massive image datasets (ImageNet, CIFAR10, etc) where