DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING, IIT MADRAS

CS6910: Programming Assignment 3

Team 37

Sheth Dev Yashpal
CS17B106

Harshit Kedia
CS17B103

1 TASK 1 - FEATURE EXTRACTION FROM VGG16 AND GOOGLENET

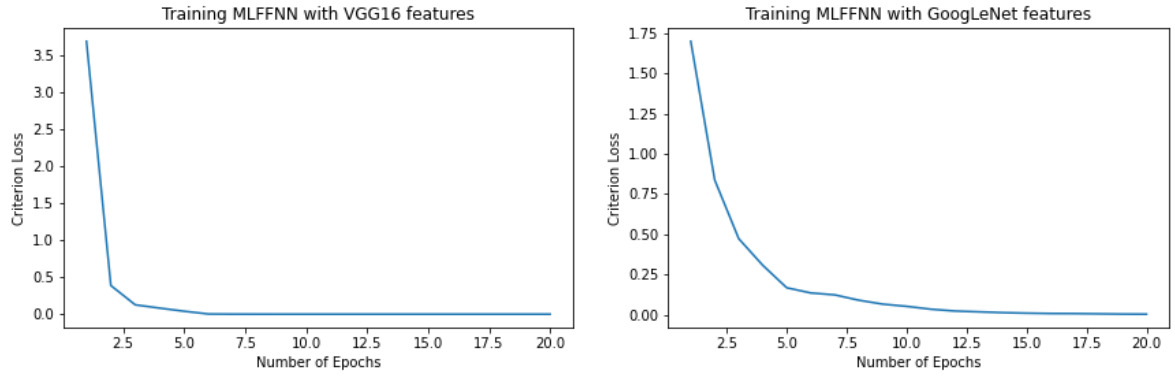


Figure 1.1: Training error curves vs epoch number

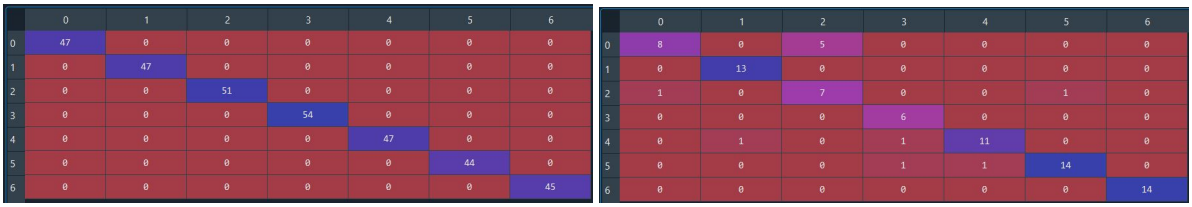


Figure 1.2: Confusion Matrix for training MLFFNN over VGG16 features, train and test accuracies being **100.0%** and **86.9%** respectively.

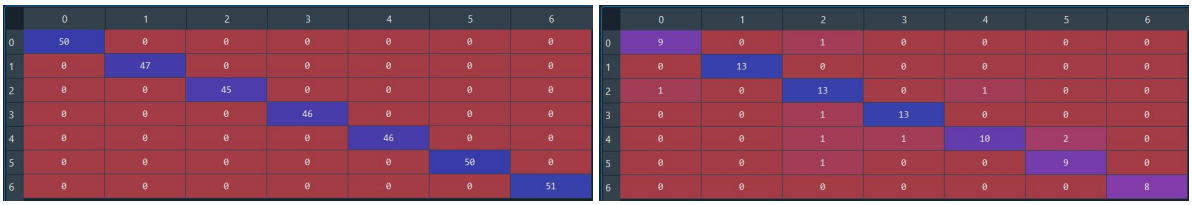


Figure 1.3: Confusion Matrix for training MLFFNN over GoogLeNet features, train and test accuracies being **100.0%** and **89.29%** respectively.

Configuration and Observations:

- VGG16 outputs a feature vector of dimension 25088. We use a MLFFNN with 2 hidden layers of sizes 4096 and 1024 for the classifier network. We tried using a smaller network and a network with less number of layers both resulted in poorer performance than the current configuration.
- GoogLeNet outputs a feature vector of dimension 1024 and we again use a two layer network with hidden layer sizes 512 and 128. For both the cases, we have a learning rate of 0.001 and a batch size of 16 and trained the networks for 100 epochs. A higher batch size would have been preferable however due to the small size of the training data we did not tune that parameter further. For all our experiments we split the given dataset into 80% train and 20% test and used the Adam Optimizer with default parameters.
- The MLFFNN classifier is able to perfectly fit on the training data and gives perfect accuracy but it also doesn't abruptly fail over the test data giving decent performance on the blind test set as well. This shows that the features extracted from the pretrained networks are of very high-quality.

2 Task 2 - Convolutional Neural Network

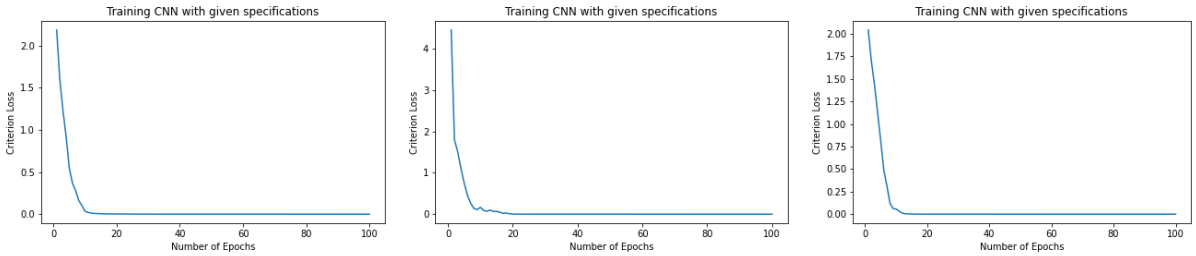


Figure 2.1: Training error curves vs epoch number with 1, 2 and 3 fully connected layers after the CNN output.

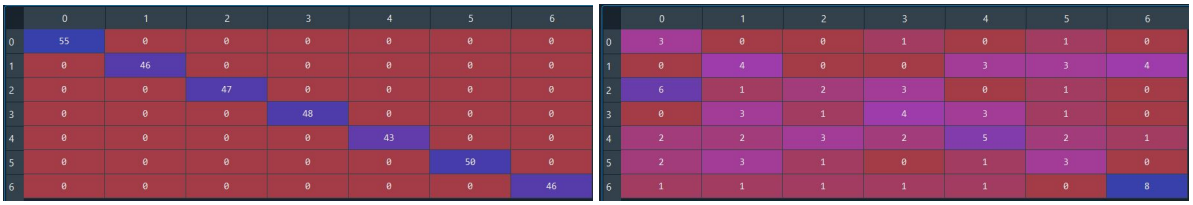


Figure 2.2: Confusion Matrix for best CNN configuration, with 3 fully connected layers following the convolution layer output, train and test accuracies being **100.0%** and **34.52%** respectively.

Configuration and Observations:

- The CNN network is kept as given in the problem statement i.e. convolution layers of size 3 with stride 1 and 4 & 16 channel outputs (we also put single padding to maintain sizes), pooling layers of size 2 and stride 2. We pre-processed the data such that all images were of size 256 x 256 as input to our network. Hence after the convolution and pooling layers we got a feature vector of 16 x 64 x 64 = 65536.
- For the MLFFNN following this we tried three different approaches - single layer (directly connecting the output nodes), double layer (1 hidden layer with 1000 nodes) and three layers (2 hidden layers of size 1000 and 100 followed by output nodes). The final test accuracy for the network with 1 fc layer was **26.19%**, with 2 fc layers it was **32.14%** and with 3 fc layers it was **34.52%**. As we see the best scenario is to use 3 fc layers like most historical networks (ex.

LeNet). But as we can see in terms of the amount of training data we have, the network is highly over-parameterized and we need to use some form of regularization to ensure similar performance across the train and test sets.

- As visible in the error vs epoch plots, the network converges very fast, around 20 epochs itself. We found the optimum batch size to be 16 and learning rate to be 0.001. As mentioned above due to the high number of parameters in the network, it is effectively learning to memorize the training examples rather than learn meaningful features and therefore converges very fast as it has to learn a really small dataset.

3 Task 3 - CNN with NetVLAD layer

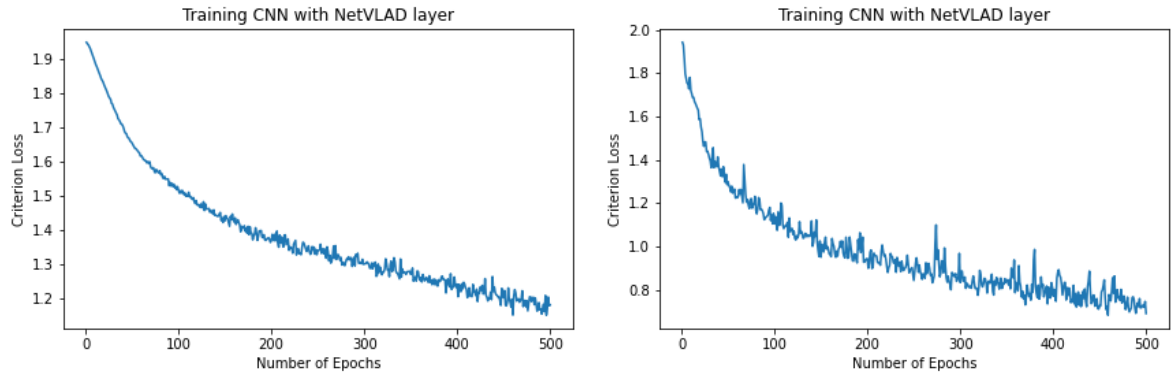


Figure 3.1: Training error vs epoch number curves for learning rates 0.001 and 0.01 respectively for 500 epochs.

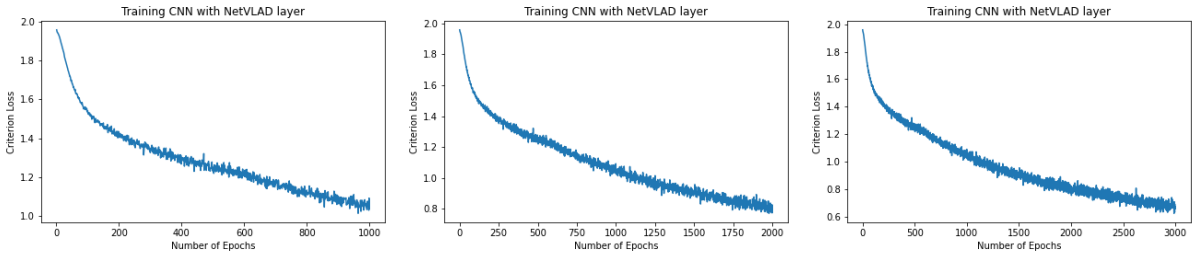


Figure 3.2: Training error vs epoch curves for 1000, 2000 and 3000 epochs respectively for NetVLAD model, with learning rate 0.001.

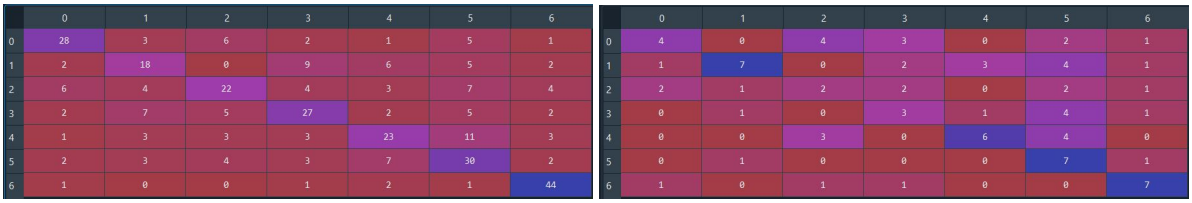
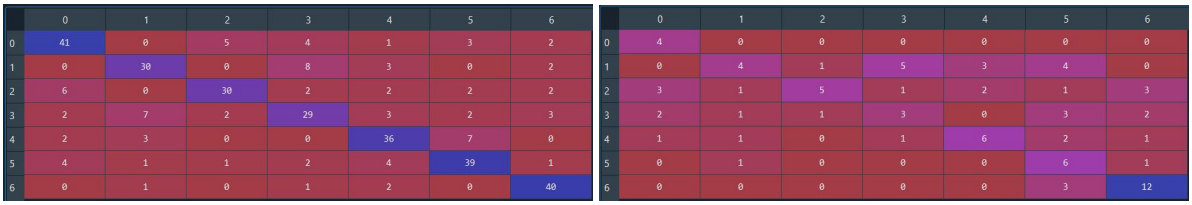


Figure 3.3: Confusion Matrix for CNN with NetVLAD layer, trained for 500 epochs. Train and Test accuracy's are **57.31%** and **42.86%** respectively.

Configuration and Observations:

- The CNN with same configuration is used as before, followed by a NetVLAD layer. This layer receives input of size 16 x 64 x 64 and uses soft-clustering to generate 4 feature vectors, each



	0	1	2	3	4	5	6
0	41	0	5	4	1	3	2
1	0	30	0	8	3	0	2
2	6	0	30	2	2	2	2
3	2	7	2	29	3	2	3
4	2	3	0	0	36	7	0
5	4	1	1	2	4	39	1
6	0	1	0	1	2	0	40

	0	1	2	3	4	5	6
0	4	0	0	0	0	0	0
1	0	4	1	5	3	4	0
2	3	1	5	1	2	1	3
3	2	1	1	3	0	3	2
4	1	1	0	1	6	2	1
5	0	1	0	0	0	6	1
6	0	0	0	0	0	3	12

Figure 3.4: Confusion Matrix for CNN with NetVLAD layer, trained for 2000 epochs. Train and Test accuracy's are **73.13% 47.62%** respectively.

of 16 dimension (4 x 16). This is then normalized, flattened and again normalized to generate output of 64 features vectors per input image. This vector is fed directly into the output layer for classification (having more fully connected layers decreased the accuracy in this case).

- For this particular task we show plots with two different learning rates - 0.01 and 0.001 trained for 500 epochs each. The higher learning rate of 0.01 led to over-fitting and the final train and test accuracies were **73.13%** and **34.52%** respectively. For the case of lr=0.001 the train and test accuracies were **57.31%** and **42.86%**. As we can see modifying the optimal hyperparameters can lead to severe misbehaviour in the training process. For this case, though increasing the learning rate improved train accuracy, the model was not able to generalize well over the blind test set. We found the batch size of 16 to be the best case again as decreasing it was destabilizing the training process and increasing was not very lucrative due to the low training data size.

- As we can see from the error vs epoch plot for 500 epochs, the overall error is still decreasing though it is showing some instability at the tail end. Therefore, we extended our training process to further 1000, 2000 and 3000 epochs. Our train accuracies went from **57.31%** to **62.69%**, **73.13%** and **77.91%** respectively. Meanwhile, our test accuracy went from **42.86%** to **44.05%**, **47.62%** and **47.62%** respectively.

- Though the model seems to show improvement over the train set by increasing the number of epochs, it is actually starting to overfit as we observe no improvement from the test set result. The training graph also shows saturation from 2500 to 3000 epochs. Therefore our best model is for the case when we train for 2000 epochs. We could have achieved the same train accuracy with a higher learning rate or a lower batch size but that way our model wouldn't generalize well. On the other hand, if we reduce the learning rate or increase the batch size we would have to train for more and more number of epochs to achieve similar train and test accuracies which is computationally very expensive.