

Task 1 and Task 2 (Done by Dhyanav)

Task 1

The CIFAR10 dataset is already clean, shuffled and neatly labelled. So, I directly moved to data augmentation. I divided 60,000 images of CIFAR-10 into 6 batches of 10,000 each. They were used for flipping, rotating, cropping, adding noise, cut-out and mixup.

Cut-out: I randomly cut a square of 6 by 6 from the image. Using this technique helps the model learn better since it tries to identify an image without a certain part. It reduces dependence on specific parts of the image. I decided to use 6 by 6 after some experimentation. I feel it was in the middle, not too small to be practically ineffective and not too large to remove all key features entirely.

Mixup: It selects a random image and imposes it on chosen images. The formula used is

$$X_{\text{new}} = \alpha X_1 + (1-\alpha)X_2$$

Here, X_{new} is the final result, X_1 is the base image and X_2 is the image that is being imposed. After some experimentation, I used $\alpha = 0.75$.

Then, I combined it into the original dataset, doubling its size. Then, I split data. 95,000 images for training, 10,000 for validation and 15,000 for testing.

Now, I moved to task 2

Task 2

ANN:

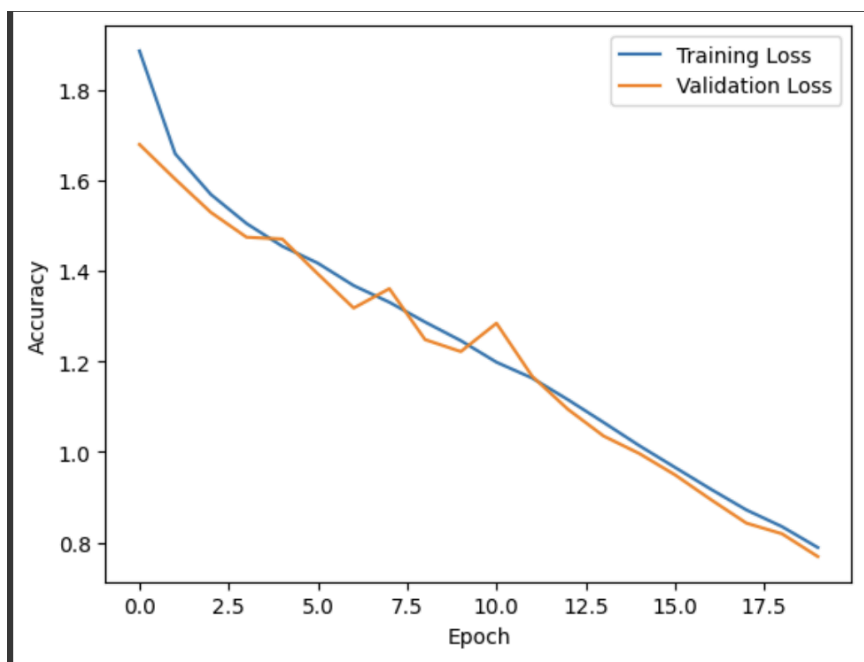
I started with 2 hidden layers, but gradually increased the layers so it was computationally viable. Finally, there are 4 layers. I also doubled the number of neurons in each layer and stuck with it since it increased accuracy without overfitting.

I trained for 15 epochs. Since, loss is steadily decreasing, implementing early stopping was not useful. Increasing epochs led to an increase in difference between training accuracy and test accuracy. I finally decided to use 15 epochs since the difference was only 5%.

```

Epoch 1/20
743/743 ————— 10s 8ms/step - accuracy: 0.2536 - loss: 2.0862 - val_accuracy: 0.3981 - val_loss: 1.679
Epoch 2/20
743/743 ————— 4s 5ms/step - accuracy: 0.3936 - loss: 1.6859 - val_accuracy: 0.4287 - val_loss: 1.6026
Epoch 3/20
743/743 ————— 4s 5ms/step - accuracy: 0.4359 - loss: 1.5695 - val_accuracy: 0.4551 - val_loss: 1.5297
Epoch 4/20
743/743 ————— 4s 5ms/step - accuracy: 0.4602 - loss: 1.5097 - val_accuracy: 0.4736 - val_loss: 1.4743
Epoch 5/20
743/743 ————— 5s 5ms/step - accuracy: 0.4803 - loss: 1.4559 - val_accuracy: 0.4801 - val_loss: 1.4703
Epoch 6/20
743/743 ————— 4s 5ms/step - accuracy: 0.4901 - loss: 1.4178 - val_accuracy: 0.4946 - val_loss: 1.3936
Epoch 7/20
743/743 ————— 4s 5ms/step - accuracy: 0.5117 - loss: 1.3672 - val_accuracy: 0.5313 - val_loss: 1.3179
Epoch 8/20
743/743 ————— 4s 5ms/step - accuracy: 0.5262 - loss: 1.3283 - val_accuracy: 0.5214 - val_loss: 1.3607
Epoch 9/20
743/743 ————— 5s 5ms/step - accuracy: 0.5345 - loss: 1.2936 - val_accuracy: 0.5600 - val_loss: 1.2482
Epoch 10/20
743/743 ————— 5s 5ms/step - accuracy: 0.5547 - loss: 1.2391 - val_accuracy: 0.5637 - val_loss: 1.2216
Epoch 11/20
743/743 ————— 5s 5ms/step - accuracy: 0.5720 - loss: 1.1946 - val_accuracy: 0.5372 - val_loss: 1.2843
Epoch 12/20
743/743 ————— 4s 5ms/step - accuracy: 0.5868 - loss: 1.1579 - val_accuracy: 0.5885 - val_loss: 1.1672
Epoch 13/20
743/743 ————— 4s 5ms/step - accuracy: 0.6027 - loss: 1.1118 - val_accuracy: 0.6075 - val_loss: 1.0944
Epoch 14/20
743/743 ————— 5s 5ms/step - accuracy: 0.6242 - loss: 1.0528 - val_accuracy: 0.6281 - val_loss: 1.0353
Epoch 15/20
743/743 ————— 5s 5ms/step - accuracy: 0.6451 - loss: 0.9930 - val_accuracy: 0.6412 - val_loss: 0.9966
Epoch 16/20
743/743 ————— 5s 5ms/step - accuracy: 0.6584 - loss: 0.9508 - val_accuracy: 0.6611 - val_loss: 0.9495
Epoch 17/20
743/743 ————— 4s 5ms/step - accuracy: 0.6805 - loss: 0.9027 - val_accuracy: 0.6784 - val_loss: 0.8952
Epoch 18/20
743/743 ————— 4s 5ms/step - accuracy: 0.6954 - loss: 0.8527 - val_accuracy: 0.6981 - val_loss: 0.8429
Epoch 19/20
743/743 ————— 5s 5ms/step - accuracy: 0.7094 - loss: 0.8127 - val_accuracy: 0.7044 - val_loss: 0.8193
Epoch 20/20
743/743 ————— 5s 5ms/step - accuracy: 0.7290 - loss: 0.7683 - val_accuracy: 0.7202 - val_loss: 0.7691
469/469 ————— 2s 3ms/step - accuracy: 0.6772 - loss: 0.9278
[1.1586171388626099, 0.6033333539962769]

```

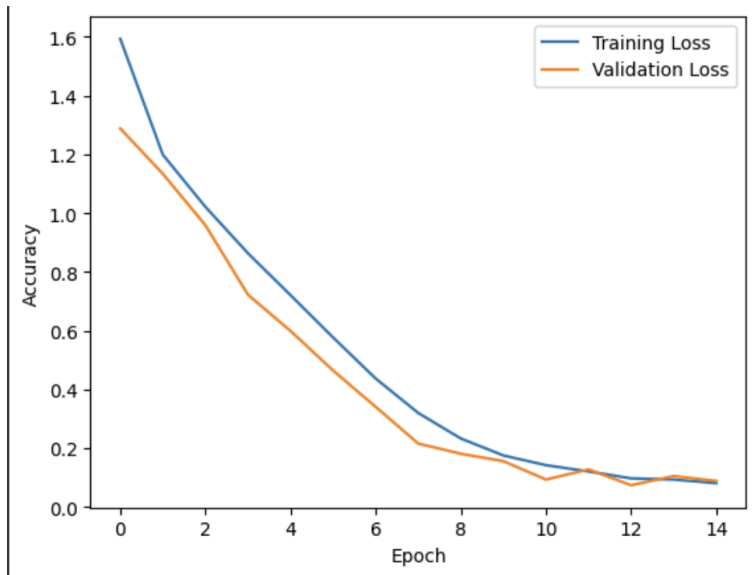


CNN:

I used a similar process as in ANN. Gradually increasing the number of layers and number of neurons in each layer till I was satisfied with the result. I tried implementing early stopping but it was useless as only the last three layers were showing signs of saturation and early stopping evaluates accuracy of three layers before stopping training.

In both, I used ADAM optimiser, so learning rate was not a part of experimentation as ADAM adjusts learning rate accordingly. I increased batch size from 32 to 256 to speed up training while also maintaining accuracy. (After 256, accuracy started to drop.)

```
Epoch 1/15  
372/372 — 22s 36ms/step - accuracy: 0.3192 - loss: 1.8257 - val_accuracy: 0.5380 - val_loss: 1.2884  
Epoch 2/15  
372/372 — 6s 17ms/step - accuracy: 0.5547 - loss: 1.2457 - val_accuracy: 0.5978 - val_loss: 1.1342  
Epoch 3/15  
372/372 — 9s 15ms/step - accuracy: 0.6289 - loss: 1.0507 - val_accuracy: 0.6544 - val_loss: 0.9589  
Epoch 4/15  
372/372 — 5s 14ms/step - accuracy: 0.6856 - loss: 0.8931 - val_accuracy: 0.7445 - val_loss: 0.7224  
Epoch 5/15  
372/372 — 11s 17ms/step - accuracy: 0.7452 - loss: 0.7242 - val_accuracy: 0.7935 - val_loss: 0.5991  
Epoch 6/15  
372/372 — 6s 15ms/step - accuracy: 0.8011 - loss: 0.5730 - val_accuracy: 0.8386 - val_loss: 0.4643  
Epoch 7/15  
372/372 — 6s 15ms/step - accuracy: 0.8474 - loss: 0.4388 - val_accuracy: 0.8813 - val_loss: 0.3407  
Epoch 8/15  
372/372 — 10s 14ms/step - accuracy: 0.8931 - loss: 0.3132 - val_accuracy: 0.9268 - val_loss: 0.2155  
Epoch 9/15  
372/372 — 10s 14ms/step - accuracy: 0.9237 - loss: 0.2235 - val_accuracy: 0.9379 - val_loss: 0.1811  
Epoch 10/15  
372/372 — 10s 14ms/step - accuracy: 0.9434 - loss: 0.1683 - val_accuracy: 0.9467 - val_loss: 0.1558  
Epoch 11/15  
372/372 — 10s 14ms/step - accuracy: 0.9570 - loss: 0.1266 - val_accuracy: 0.9681 - val_loss: 0.0932  
Epoch 12/15  
372/372 — 5s 14ms/step - accuracy: 0.9616 - loss: 0.1156 - val_accuracy: 0.9552 - val_loss: 0.1277  
Epoch 13/15  
372/372 — 10s 14ms/step - accuracy: 0.9689 - loss: 0.0928 - val_accuracy: 0.9745 - val_loss: 0.0739  
Epoch 14/15  
372/372 — 10s 14ms/step - accuracy: 0.9655 - loss: 0.1021 - val_accuracy: 0.9640 - val_loss: 0.1051  
Epoch 15/15  
372/372 — 10s 14ms/step - accuracy: 0.9759 - loss: 0.0698 - val_accuracy: 0.9692 - val_loss: 0.0885  
469/469 — 2s 3ms/step - accuracy: 0.9271 - loss: 0.2719  
[0.5354151129722595, 0.8613333106040955]
```



I also trained the same models, ANN and CNN on raw CIFAR10 data and the difference in accuracy was significant.

```
super (7) _init_ (backwards)
Epoch 1/20
391/391 _____ 6s 8ms/step - accuracy: 0.1764 - loss: 192.5728
Epoch 2/20
391/391 _____ 2s 4ms/step - accuracy: 0.3281 - loss: 1.9206
Epoch 3/20
391/391 _____ 2s 4ms/step - accuracy: 0.3685 - loss: 1.7916
Epoch 4/20
391/391 _____ 3s 4ms/step - accuracy: 0.4040 - loss: 1.6968
Epoch 5/20
391/391 _____ 3s 4ms/step - accuracy: 0.4141 - loss: 1.6505
Epoch 6/20
391/391 _____ 3s 5ms/step - accuracy: 0.4289 - loss: 1.6159
Epoch 7/20
391/391 _____ 2s 4ms/step - accuracy: 0.4376 - loss: 1.5935
Epoch 8/20
391/391 _____ 3s 4ms/step - accuracy: 0.4269 - loss: 1.6178
Epoch 9/20
391/391 _____ 2s 4ms/step - accuracy: 0.4427 - loss: 1.5682
Epoch 10/20
391/391 _____ 2s 4ms/step - accuracy: 0.4447 - loss: 1.5621
Epoch 11/20
391/391 _____ 2s 4ms/step - accuracy: 0.4545 - loss: 1.5468
Epoch 12/20
391/391 _____ 3s 5ms/step - accuracy: 0.4561 - loss: 1.5256
Epoch 13/20
391/391 _____ 2s 4ms/step - accuracy: 0.4631 - loss: 1.5007
Epoch 14/20
391/391 _____ 3s 5ms/step - accuracy: 0.4721 - loss: 1.4838
Epoch 15/20
391/391 _____ 3s 4ms/step - accuracy: 0.4644 - loss: 1.4894
Epoch 16/20
391/391 _____ 2s 4ms/step - accuracy: 0.4696 - loss: 1.4905
Epoch 17/20
391/391 _____ 3s 5ms/step - accuracy: 0.4823 - loss: 1.4414
Epoch 18/20
391/391 _____ 2s 4ms/step - accuracy: 0.4845 - loss: 1.4481
Epoch 19/20
391/391 _____ 2s 4ms/step - accuracy: 0.4853 - loss: 1.4352
Epoch 20/20
391/391 _____ 3s 4ms/step - accuracy: 0.4903 - loss: 1.4312
313/313 _____ 2s 3ms/step - accuracy: 0.4469 - loss: 1.5485
[1.5541332960128784, 0.44449999928474426]
```

```
Epoch 1/15
196/196 _____ 9s 22ms/step - accuracy: 0.1260 - loss: 5.1147
Epoch 2/15
196/196 _____ 7s 13ms/step - accuracy: 0.2919 - loss: 1.8043
Epoch 3/15
196/196 _____ 5s 13ms/step - accuracy: 0.4495 - loss: 1.4742
Epoch 4/15
196/196 _____ 2s 12ms/step - accuracy: 0.5496 - loss: 1.2520
Epoch 5/15
196/196 _____ 3s 12ms/step - accuracy: 0.6090 - loss: 1.0987
Epoch 6/15
196/196 _____ 3s 13ms/step - accuracy: 0.6700 - loss: 0.9320
Epoch 7/15
196/196 _____ 5s 13ms/step - accuracy: 0.7162 - loss: 0.7970
Epoch 8/15
196/196 _____ 2s 12ms/step - accuracy: 0.7590 - loss: 0.6782
Epoch 9/15
196/196 _____ 3s 13ms/step - accuracy: 0.8057 - loss: 0.5491
Epoch 10/15
196/196 _____ 3s 13ms/step - accuracy: 0.8435 - loss: 0.4444
Epoch 11/15
196/196 _____ 3s 13ms/step - accuracy: 0.8769 - loss: 0.3582
Epoch 12/15
196/196 _____ 3s 13ms/step - accuracy: 0.8971 - loss: 0.2993
Epoch 13/15
196/196 _____ 3s 13ms/step - accuracy: 0.9286 - loss: 0.2087
Epoch 14/15
196/196 _____ 3s 13ms/step - accuracy: 0.9367 - loss: 0.1879
Epoch 15/15
196/196 _____ 2s 13ms/step - accuracy: 0.9466 - loss: 0.1638
313/313 _____ 2s 4ms/step - accuracy: 0.6336 - loss: 1.7517
[1.766332983970642, 0.6341000199317932]
```

On the CNN model, I implemented one-pixel attack. This is a type of attack where the attacker randomly selects a pixel and manipulates it. It is known to reduce accuracy of the model by a significant margin. But since I implemented cutout in data augmentation, model accuracy only dropped from 91.24% to 83.34

```
one_pixel_attack(test_images, test_labels, model_cnn)
```

```
0.843
```

Task 3-6(Sumit and Shikhar)

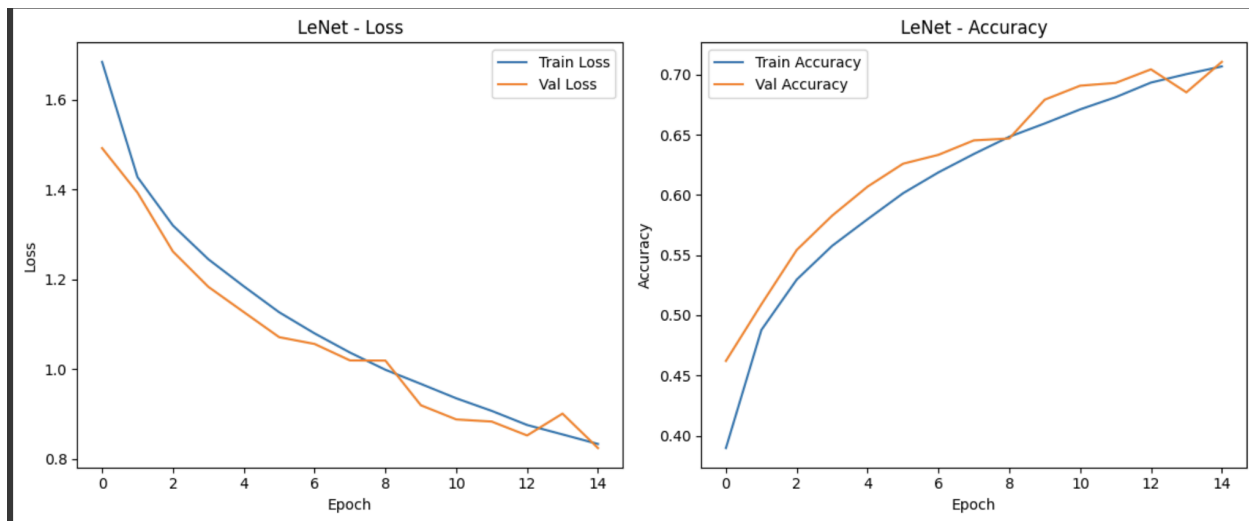
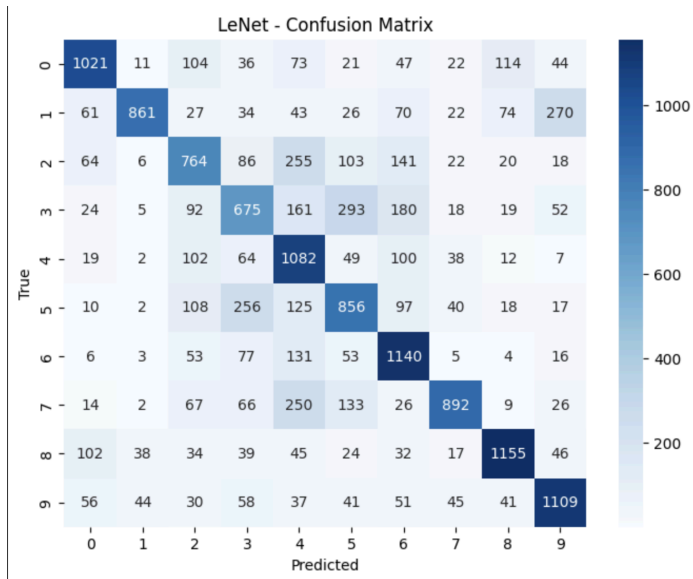
AlexNet, LeNet, VGG16(Sumit)

LeNet: ~60K parameters. A shallow network with 2 convolutional layers.

Epochs-15

Batch Size-128

LeNet - Classification Report					
	precision	recall	f1-score	support	
0	0.74	0.68	0.71	1493	
1	0.88	0.58	0.70	1488	
2	0.55	0.52	0.53	1479	
3	0.49	0.44	0.46	1519	
4	0.49	0.73	0.59	1475	
5	0.54	0.56	0.55	1529	
6	0.61	0.77	0.68	1488	
7	0.80	0.60	0.68	1485	
8	0.79	0.75	0.77	1532	
9	0.69	0.73	0.71	1512	
accuracy			0.64	15000	
macro avg	0.66	0.64	0.64	15000	
weighted avg	0.66	0.64	0.64	15000	
ROC AUC Score (0vA): 0.9401					

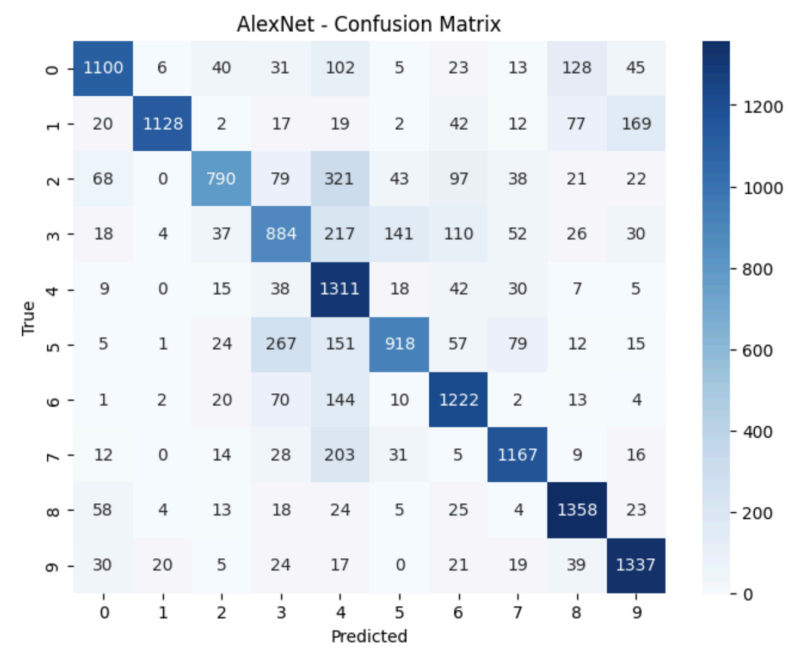


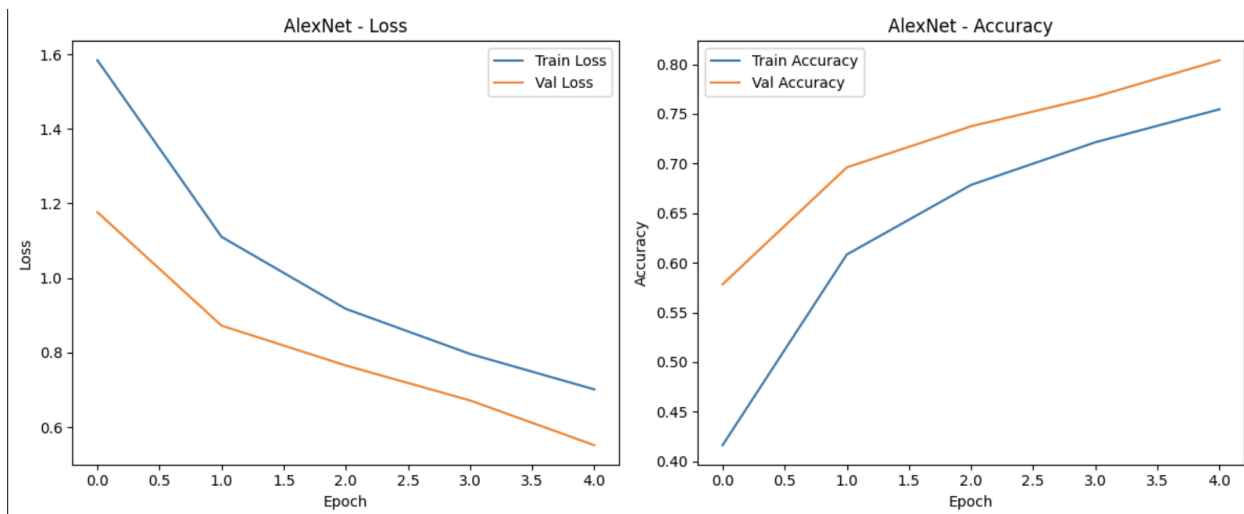
AlexNet: ~60M parameters. A deeper architecture with 5 convolutional layers. It handles larger input variations better.

Epochs: Initially tried 15 but it was taking much longer than LeNet so switched to 5.

Batch Size-128

AlexNet - Classification Report					
	precision	recall	f1-score	support	
0	0.83	0.74	0.78	1493	
1	0.97	0.76	0.85	1488	
2	0.82	0.53	0.65	1479	
3	0.61	0.58	0.59	1519	
4	0.52	0.89	0.66	1475	
5	0.78	0.60	0.68	1529	
6	0.74	0.82	0.78	1488	
7	0.82	0.79	0.80	1485	
8	0.80	0.89	0.84	1532	
9	0.80	0.88	0.84	1512	
accuracy			0.75	15000	
macro avg	0.77	0.75	0.75	15000	
weighted avg	0.77	0.75	0.75	15000	
ROC AUC Score (OvA): 0.9706					



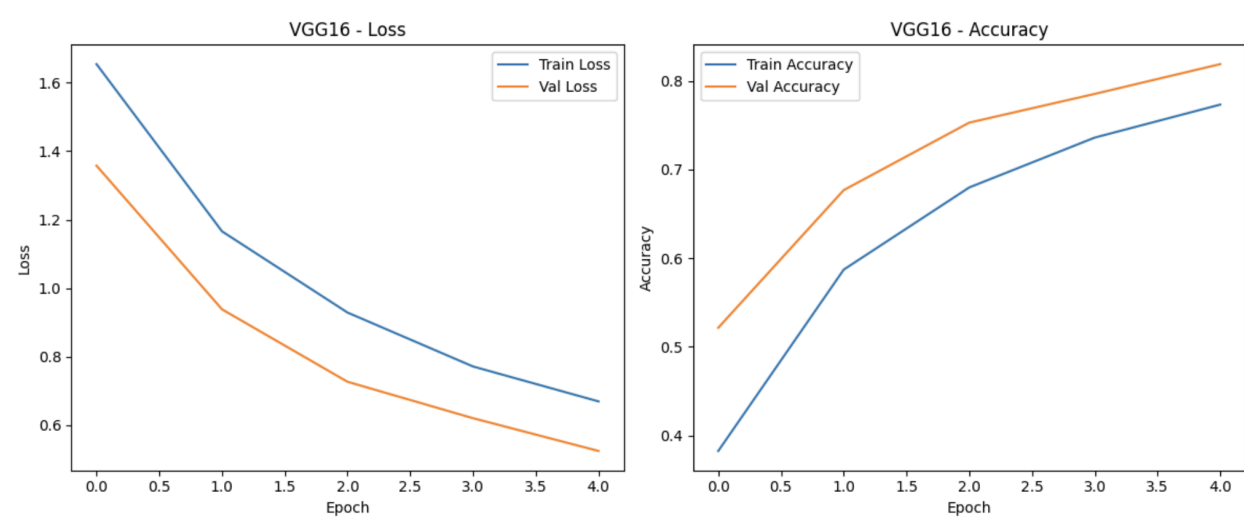
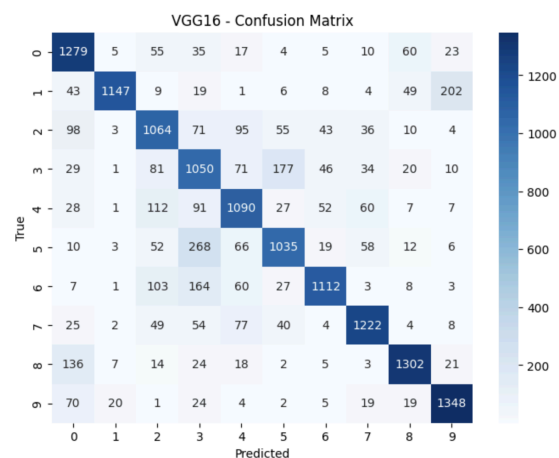


VGG16: ~138M parameters. A very deep model with 13 convolutional layers.

Epochs: 5(same problem as AlexNet)

Batch Size: 128

VGG16 - Classification Report				
	precision	recall	f1-score	support
0	0.74	0.86	0.79	1493
1	0.96	0.77	0.86	1488
2	0.69	0.72	0.70	1479
3	0.58	0.69	0.63	1519
4	0.73	0.74	0.73	1475
5	0.75	0.68	0.71	1529
6	0.86	0.75	0.80	1488
7	0.84	0.82	0.83	1485
8	0.87	0.85	0.86	1532
9	0.83	0.89	0.86	1512
accuracy			0.78	15000
macro avg	0.79	0.78	0.78	15000
weighted avg	0.79	0.78	0.78	15000
ROC AUC Score (0vA): 0.9746				



Model	Accuracy	Precision	Recall	F1-score	ROC-AUC(OvA)
LeNet	0.64	0.66	0.64	0.64	0.9401
AlexNet	0.75	0.77	0.75	0.75	0.9706
VGG16	0.78	0.79	0.78	0.78	0.9746

(here precision, recall, and f1 score are the macro avg)

Factor	LeNet	AlexNet	VGG16
Overfitting	High	Medium	Low
Training Time	Low	Medium	High
Generalization	Poor	Decent	Good

Conclusion:

- VGG16 gave the best results in terms of accuracy, F1-score, and AUC.
- LeNet was unsuitable for CIFAR-10 due to its simplicity.
- AlexNet was a good balance between complexity and performance.