UNIVERSITY OF TEHRAN

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NEURAL NETWORK & DEEP LEARNING

TRANSFER LEARNING USING VGG19

SIAVASH SHAMS

MOHAMMAD HEYDARI

UNDER SUPERVISION OF:

DR. AHMAD KALHOR

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

Apr. 2022

1 CONTENTS

2 Question #1: CNN(Classification) 3
2.1 Implementing just Using Convolutional-Layers 4
2.2 Adding Max-Pooling & Batch-Normalization Layers 6
2.3 Adding Dropout Layer and Analysing Results 10
2.4 Early Stopping Approach in Neural Networks 14
3 Question #2: Transfer Learning 17
3.1 VGG19 17
3.2 Transfer Learning 18
3.3 VGG19 Pre-Trained by ImageNet 18
3.4 Transfer Learning and VGG19 19

2 CNN(CLASSIFICATION)

In this part we intend to implement a Convolutional Neural Network in case of image classification and feature extraction.

In this section We have used different convolutional layers such as max-pooling layer, batch-normalization layer and many others to satisfy our goal which is all about increasing the classifier accuracy.

As we know the most important part of our implementation is about our pipeline architecture, so further I have provided a usable figure which properly visualizes all we need throughout this part of project:

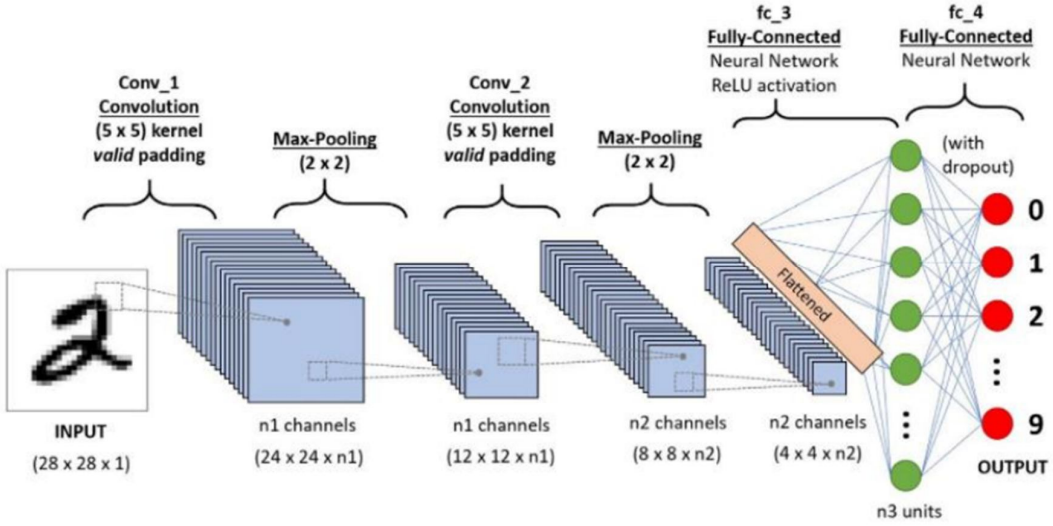


Figure1. CNN Classifier Architecture

As you can see, we are dealing with three main layers which abbreviated below:

1- 3D-Convolutional Layers in case of feature extraction.

2- Max-Pooling layers in case of image size reduction.

3- Fully-Connected Layers in case of Classification task.

Pipeline Procedure:

In the first step we pass the input image to the first Convolution layer for extracting the features and also for image size reduction, afterward we repeat filtering task one more time to obtain the size-reducing form of input image, furthermore we have implemented the flatten layer which gives us all feature information in case of a suitable vector for passing through fully-connected layers topology.

Finally, we have used softmax activation-function to do multi-class classification task as well as it could be.

Now after a brief contextualization we are dealing with four separated implementation to satisfy all of question requirements.

Note that as of question description, we have reported figure of Error and Accuracy variation among epochs for both train and validation data in two separated curves.

As of a supplementary approach we have also reported the training-time, confusion-matrix and also test final accuracy in each case.

2.1 IMPLEMENTING JUST USING CONVOLUTIONAL-LAYERS

Further you can see the training process:

```
Epoch 1/20
313/313 [==============================] - 51s 129ms/step - loss: 1.6700 - accuracy: 0.4304 - val_loss: 1.3118 - val_accuracy: 0.5264
Epoch 2/20
313/313 [==============================] - 39s 125ms/step - loss: 1.1880 - accuracy: 0.5779 - val_loss: 1.1116 - val_accuracy: 0.6088
Epoch 3/20
313/313 [==============================] - 39s 126ms/step - loss: 0.9180 - accuracy: 0.6758 - val_loss: 0.9914 - val_accuracy: 0.6528
Epoch 4/20
313/313 [==============================] - 39s 126ms/step - loss: 0.6934 - accuracy: 0.7564 - val_loss: 0.9807 - val_accuracy: 0.6724
Epoch 5/20
313/313 [==============================] - 39s 126ms/step - loss: 0.4525 - accuracy: 0.8422 - val_loss: 1.0833 - val_accuracy: 0.6653
Epoch 6/20
313/313 [==============================] - 40s 126ms/step - loss: 0.2440 - accuracy: 0.9168 - val_loss: 1.4177 - val_accuracy: 0.6548
Epoch 7/20
313/313 [==============================] - 39s 126ms/step - loss: 0.1390 - accuracy: 0.9528 - val_loss: 1.6860 - val_accuracy: 0.6334
Epoch 8/20
313/313 [==============================] - 40s 127ms/step - loss: 0.0923 - accuracy: 0.9704 - val_loss: 1.9623 - val_accuracy: 0.6498
Epoch 9/20
313/313 [==============================] - 40s 127ms/step - loss: 0.0729 - accuracy: 0.9759 - val_loss: 2.2310 - val_accuracy: 0.6407
Epoch 10/20
313/313 [==============================] - 40s 127ms/step - loss: 0.0829 - accuracy: 0.9726 - val_loss: 1.9982 - val_accuracy: 0.6377
Epoch 11/20
313/313 [==============================] - 40s 127ms/step - loss: 0.0506 - accuracy: 0.9828 - val_loss: 2.6022 - val_accuracy: 0.6347
Epoch 12/20
313/313 [==============================] - 40s 126ms/step - loss: 0.0609 - accuracy: 0.9801 - val_loss: 2.7132 - val_accuracy: 0.6343
Epoch 13/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0457 - accuracy: 0.9855 - val_loss: 2.6997 - val_accuracy: 0.6349
Epoch 14/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0348 - accuracy: 0.9883 - val_loss: 2.7218 - val_accuracy: 0.6370
Epoch 15/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0426 - accuracy: 0.9861 - val_loss: 3.0546 - val_accuracy: 0.6335
Epoch 16/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0517 - accuracy: 0.9837 - val_loss: 2.6364 - val_accuracy: 0.6260
Epoch 17/20
313/313 [==============================] - 40s 126ms/step - loss: 0.0342 - accuracy: 0.9885 - val_loss: 3.0321 - val_accuracy: 0.6445
Epoch 18/20
313/313 [==============================] - 39s 125ms/step - loss: 0.0353 - accuracy: 0.9882 - val_loss: 3.1434 - val_accuracy: 0.6166
Epoch 19/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0398 - accuracy: 0.9867 - val_loss: 2.9393 - val_accuracy: 0.6362
Epoch 20/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0359 - accuracy: 0.9882 - val_loss: 3.0170 - val_accuracy: 0.6243
```

As you can see the accuracy on the train-data is a suitable value.

```
Training time: 803.5931985378265s
test loss:  668.6348876953125
test acc:  57.190001010894775
confusion matrix:

 [[600  36  71  20  30  21  13  26 112  71]
 [ 25 738   5  11   3  13  15  20  25 145]
 [100  10 378  65  81 148  70 100  25  23]
 [ 33  25  57 326  68 268  66 106  21  30]
 [ 39  16  70  76 371 139  73 180  21  15]
 [ 13   7  51 115  40 594  31 123  11  15]
 [ 20  12  61  85  58  82 590  54  15  23]
 [ 23   5  37  33  45  91   9 718   8  31]
 [117  68  13  12   7  10   6  15 678  74]
 [ 37 110   9  17   5  19   7  49  21 726]]
```

Figure2. Test Results

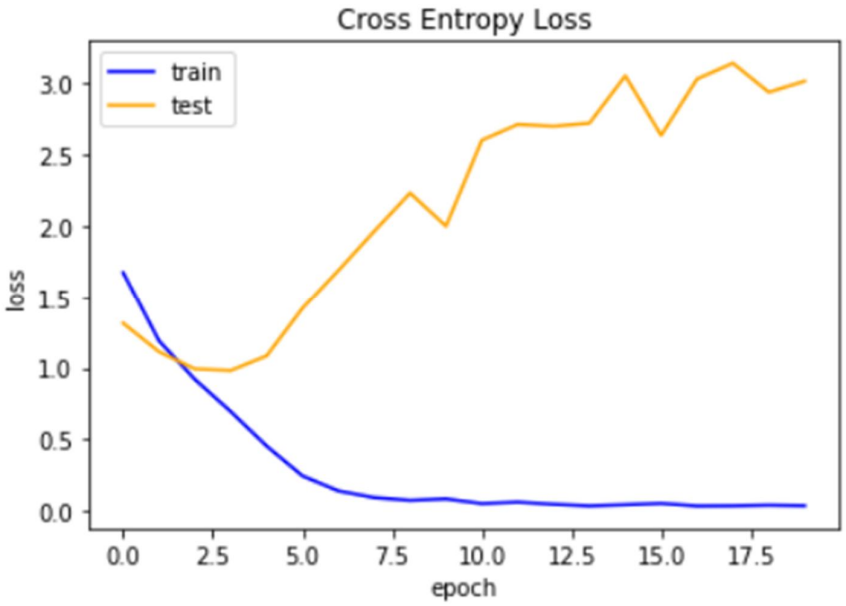


Figure3. Cross Entropy Loss

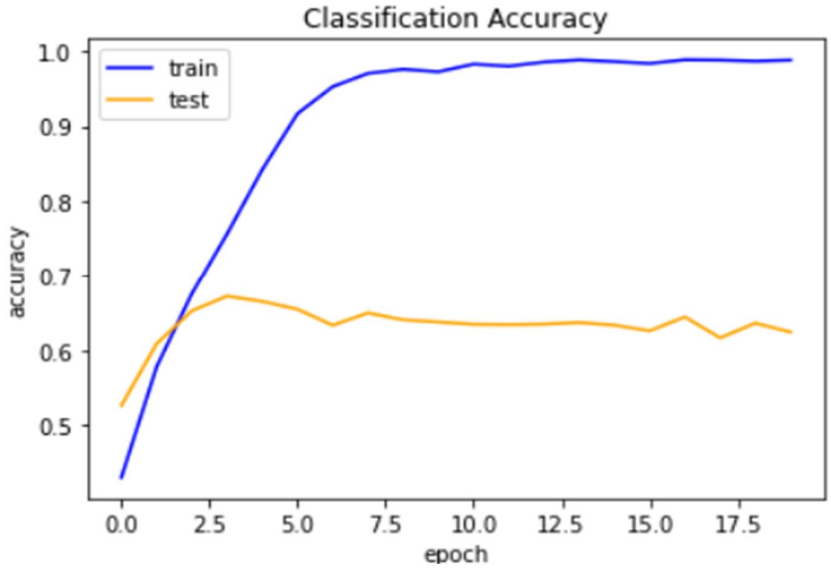


Figure4. Classification Accuracy

2.2 ADDING MAX-POOLING & BATCH-NORMALIZATION LAYERS

Batch-Normalization:

Batch normalization scales layers outputs to have mean 0 and variance 1. The outputs are scaled such a way to train the network faster. It also reduces problems due to poor parameter initialization.

Specifically, batch normalization normalizes the output of a previous layer by subtracting the batch mean and dividing by the batch standard deviation.

This is much similar to feature scaling which is done to speed up the learning process and converge to a solution.

If the distribution of the inputs to every layer is the same, the network is efficient. Batch normalization standardizes the distribution of layer inputs to combat the internal covariance shift. It controls the amount by which the hidden units shift.

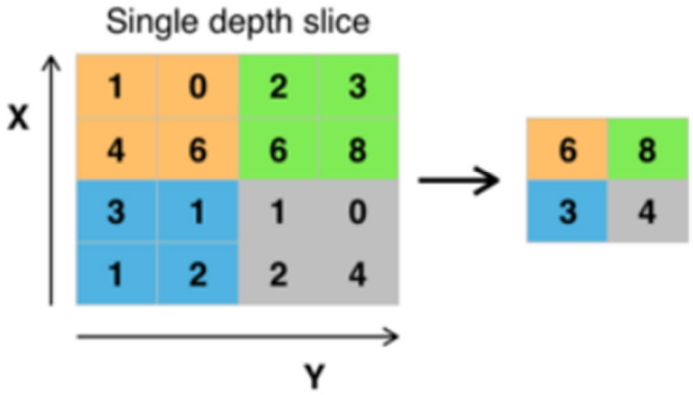
Pooling:

Pooling is nothing other than down sampling of an image. The most common pooling layer filter is of size 2x2, which discards three forth of the activations. Role of pooling layer is to reduce the resolution of the feature map but retaining features of the map required for classification through translational and rotational invariants. In addition to spatial invariance robustness, pooling will reduce the computation cost by a great deal.

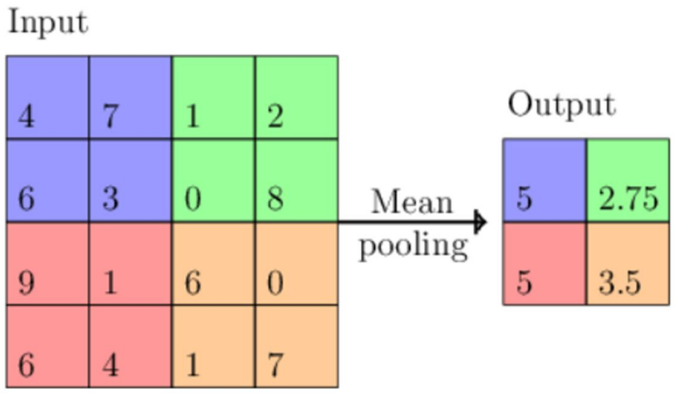
- Backpropagation is used for training of pooling operation
- It again helps the processor to process things faster.

There are many pooling techniques. They are as follows:

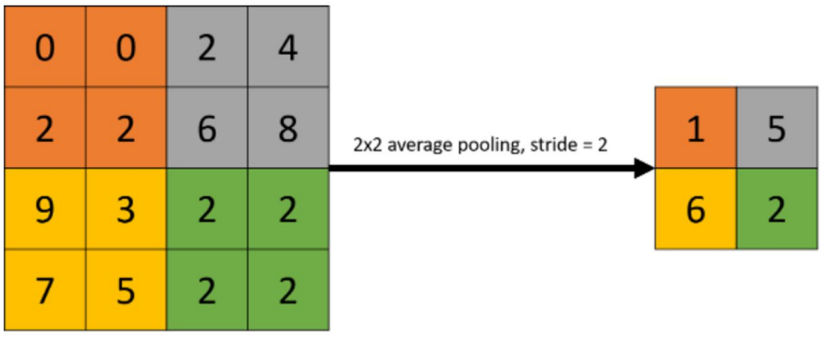
Max pooling where we take largest of the pixel values of a segment.



Mean pooling where we take largest of the pixel values of a segment.



Avg-pooling where we take largest of the pixel values of a segment.



Further you can see the training process in this section:

```
Epoch 1/20
313/313 [==============================] - 23s 40ms/step - loss: 1.2929 - accuracy: 0.5407 - val_loss: 1.4887 - val_accuracy: 0.4736
Epoch 2/20
313/313 [==============================] - 12s 37ms/step - loss: 0.8480 - accuracy: 0.6978 - val_loss: 0.9594 - val_accuracy: 0.6708
Epoch 3/20
313/313 [==============================] - 12s 37ms/step - loss: 0.6496 - accuracy: 0.7711 - val_loss: 0.8767 - val_accuracy: 0.7086
Epoch 4/20
313/313 [==============================] - 12s 37ms/step - loss: 0.5109 - accuracy: 0.8205 - val_loss: 0.7921 - val_accuracy: 0.7326
Epoch 5/20
313/313 [==============================] - 12s 37ms/step - loss: 0.4009 - accuracy: 0.8583 - val_loss: 0.7481 - val_accuracy: 0.7587
Epoch 6/20
313/313 [==============================] - 12s 37ms/step - loss: 0.2969 - accuracy: 0.8958 - val_loss: 0.8478 - val_accuracy: 0.7431
Epoch 7/20
313/313 [==============================] - 12s 37ms/step - loss: 0.2239 - accuracy: 0.9201 - val_loss: 0.8741 - val_accuracy: 0.7479
Epoch 8/20
313/313 [==============================] - 12s 37ms/step - loss: 0.1774 - accuracy: 0.9375 - val_loss: 0.9341 - val_accuracy: 0.7610
Epoch 9/20
313/313 [==============================] - 12s 38ms/step - loss: 0.1247 - accuracy: 0.9566 - val_loss: 0.9643 - val_accuracy: 0.7654
Epoch 10/20
313/313 [==============================] - 12s 39ms/step - loss: 0.1045 - accuracy: 0.9627 - val_loss: 1.0302 - val_accuracy: 0.7708
Epoch 11/20
313/313 [==============================] - 12s 38ms/step - loss: 0.0956 - accuracy: 0.9658 - val_loss: 1.1223 - val_accuracy: 0.7602
Epoch 12/20
313/313 [==============================] - 12s 38ms/step - loss: 0.0919 - accuracy: 0.9677 - val_loss: 1.1365 - val_accuracy: 0.7532
Epoch 13/20
313/313 [==============================] - 12s 37ms/step - loss: 0.1047 - accuracy: 0.9628 - val_loss: 1.1254 - val_accuracy: 0.7653
Epoch 14/20
313/313 [==============================] - 12s 37ms/step - loss: 0.0738 - accuracy: 0.9740 - val_loss: 1.1855 - val_accuracy: 0.7712
Epoch 15/20
313/313 [==============================] - 12s 37ms/step - loss: 0.0498 - accuracy: 0.9833 - val_loss: 1.3631 - val_accuracy: 0.7611
Epoch 16/20
313/313 [==============================] - 12s 37ms/step - loss: 0.0542 - accuracy: 0.9818 - val_loss: 1.3634 - val_accuracy: 0.7486
Epoch 17/20
313/313 [==============================] - 12s 37ms/step - loss: 0.0787 - accuracy: 0.9722 - val_loss: 1.2194 - val_accuracy: 0.7690
Epoch 18/20
313/313 [==============================] - 12s 37ms/step - loss: 0.0742 - accuracy: 0.9743 - val_loss: 1.2792 - val_accuracy: 0.7674
Epoch 18/20
313/313 [==============================] - 39s 125ms/step - loss: 0.0353 - accuracy: 0.9882 - val_loss: 3.1434 - val_accuracy: 0.6166
Epoch 19/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0398 - accuracy: 0.9867 - val_loss: 2.9393 - val_accuracy: 0.6362
Epoch 20/20
313/313 [==============================] - 39s 126ms/step - loss: 0.0359 - accuracy: 0.9882 - val_loss: 3.0170 - val_accuracy: 0.6243
```

As you can see the accuracy on the train-data is a suitable value.

```
Training time: 263.9280664920807s
test loss:  562.2454833984375
test acc:  27.790001034736633
confusion matrix:

 [[635  44   2 263   0   0   0   7  16  33]
 [213 702   2  41   0   0   0   4   9  29]
 [453  41  87 345   0   1   0  21  19  33]
 [351  29  15 538   0   1   0  38  10  18]
 [458  83  18 333   0   0   0  36  29  43]
 [240  17  14 560   0  52   0  79  14  24]
 [434  98  21 330   0   0   6  11  84  16]
 [460  97   2 148   0   3   0 167  14 109]
 [328  80   0 331   0   0   0   6 137 118]
 [212 251   1  74   0   1   0   5   1 455]]
```

Figure5. Test Results

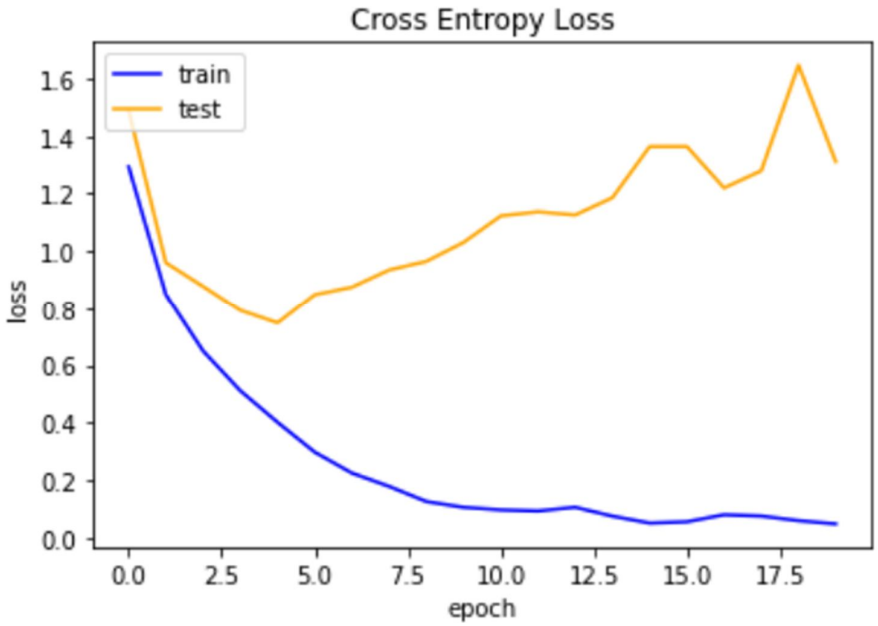


Figure6. Cross Entropy Loss

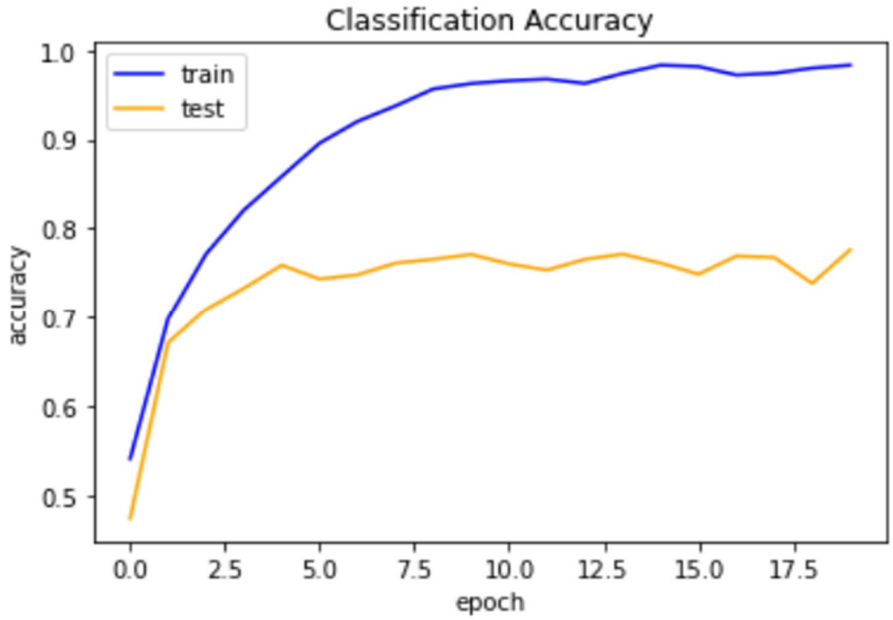


Figure7. Classification Accuracy

2.3 ADDING DROPOUT LAYER AND ANALYSING RESULTS

Why we use dropout?

Dropout is a regularization technique for reducing overfitting in artificial neural networks by preventing complex co-adaptations on training data. It is an efficient way of performing model averaging with neural networks. The term dilution refers to the thinning of the weights

Further you can see the training process in this section:

```
Epoch 1/60
313/313 [==============================] - 7s 21ms/step - loss: 1.5027 - accuracy: 0.4593 - val_loss: 1.2691 - val_accuracy: 0.5590
Epoch 2/60
313/313 [==============================] - 6s 18ms/step - loss: 1.1274 - accuracy: 0.6014 - val_loss: 1.0832 - val_accuracy: 0.6231
Epoch 3/60
313/313 [==============================] - 6s 18ms/step - loss: 0.9754 - accuracy: 0.6582 - val_loss: 1.0270 - val_accuracy: 0.6463
Epoch 4/60
313/313 [==============================] - 6s 18ms/step - loss: 0.8822 - accuracy: 0.6891 - val_loss: 0.9944 - val_accuracy: 0.6590
Epoch 5/60
313/313 [==============================] - 6s 18ms/step - loss: 0.8086 - accuracy: 0.7162 - val_loss: 0.9402 - val_accuracy: 0.6754
Epoch 6/60
313/313 [==============================] - 6s 18ms/step - loss: 0.7573 - accuracy: 0.7343 - val_loss: 0.9171 - val_accuracy: 0.6803
Epoch 7/60
313/313 [==============================] - 6s 18ms/step - loss: 0.7025 - accuracy: 0.7522 - val_loss: 0.9387 - val_accuracy: 0.6741
Epoch 8/60
313/313 [==============================] - 6s 18ms/step - loss: 0.6473 - accuracy: 0.7713 - val_loss: 0.9412 - val_accuracy: 0.6838
Epoch 9/60
313/313 [==============================] - 6s 18ms/step - loss: 0.6019 - accuracy: 0.7882 - val_loss: 0.9349 - val_accuracy: 0.6898
Epoch 10/60
313/313 [==============================] - 6s 18ms/step - loss: 0.5562 - accuracy: 0.8043 - val_loss: 0.9460 - val_accuracy: 0.6882
Epoch 11/60
313/313 [==============================] - 6s 18ms/step - loss: 0.5168 - accuracy: 0.8175 - val_loss: 0.9665 - val_accuracy: 0.6877
Epoch 12/60
313/313 [==============================] - 6s 18ms/step - loss: 0.4748 - accuracy: 0.8307 - val_loss: 0.9795 - val_accuracy: 0.6873
Epoch 13/60
313/313 [==============================] - 6s 18ms/step - loss: 0.4439 - accuracy: 0.8408 - val_loss: 0.9831 - val_accuracy: 0.6832
Epoch 14/60
313/313 [==============================] - 6s 18ms/step - loss: 0.4135 - accuracy: 0.8512 - val_loss: 1.0424 - val_accuracy: 0.6858
Epoch 15/60
313/313 [==============================] - 6s 18ms/step - loss: 0.3768 - accuracy: 0.8651 - val_loss: 1.0490 - val_accuracy: 0.6907
Epoch 16/60
313/313 [==============================] - 6s 18ms/step - loss: 0.3542 - accuracy: 0.8734 - val_loss: 1.0792 - val_accuracy: 0.6834
Epoch 17/60
313/313 [==============================] - 6s 18ms/step - loss: 0.3258 - accuracy: 0.8839 - val_loss: 1.1026 - val_accuracy: 0.6863
Epoch 18/60
313/313 [==============================] - 6s 18ms/step - loss: 0.3096 - accuracy: 0.8908 - val_loss: 1.1470 - val_accuracy: 0.6852
Epoch 19/60
313/313 [==============================] - 6s 18ms/step - loss: 0.2913 - accuracy: 0.8978 - val_loss: 1.1388 - val_accuracy: 0.6841
Epoch 20/60
313/313 [==============================] - 6s 18ms/step - loss: 0.2688 - accuracy: 0.9048 - val_loss: 1.1906 - val_accuracy: 0.6863
Epoch 21/60
313/313 [==============================] - 6s 18ms/step - loss: 0.2539 - accuracy: 0.9105 - val_loss: 1.1892 - val_accuracy: 0.6883
Epoch 22/60
313/313 [==============================] - 6s 18ms/step - loss: 0.2384 - accuracy: 0.9161 - val_loss: 1.2492 - val_accuracy: 0.6867
Epoch 23/60
313/313 [==============================] - 6s 18ms/step - loss: 0.2328 - accuracy: 0.9184 - val_loss: 1.2604 - val_accuracy: 0.6846
Epoch 24/60
313/313 [==============================] - 6s 18ms/step - loss: 0.2228 - accuracy: 0.9212 - val_loss: 1.2854 - val_accuracy: 0.6864
Epoch 25/60
313/313 [==============================] - 6s 18ms/step - loss: 0.2048 - accuracy: 0.9278 - val_loss: 1.2841 - val_accuracy: 0.6869
Epoch 26/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1928 - accuracy: 0.9310 - val_loss: 1.3079 - val_accuracy: 0.6858
Epoch 27/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1897 - accuracy: 0.9343 - val_loss: 1.3008 - val_accuracy: 0.6883
Epoch 28/60
313/313 [==============================] - 6s 19ms/step - loss: 0.1836 - accuracy: 0.9365 - val_loss: 1.3579 - val_accuracy: 0.6932
Epoch 29/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1743 - accuracy: 0.9385 - val_loss: 1.3405 - val_accuracy: 0.6898
Epoch 30/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1651 - accuracy: 0.9431 - val_loss: 1.3792 - val_accuracy: 0.6896
Epoch 31/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1619 - accuracy: 0.9433 - val_loss: 1.4196 - val_accuracy: 0.6899
Epoch 32/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1533 - accuracy: 0.9454 - val_loss: 1.4132 - val_accuracy: 0.6879
```

```
Epoch 33/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1495 - accuracy: 0.9476 - val_loss: 1.4916 - val_accuracy: 0.6852
Epoch 34/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1464 - accuracy: 0.9483 - val_loss: 1.4218 - val_accuracy: 0.6868
Epoch 35/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1371 - accuracy: 0.9529 - val_loss: 1.4977 - val_accuracy: 0.6846
Epoch 36/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1334 - accuracy: 0.9534 - val_loss: 1.4972 - val_accuracy: 0.6924
Epoch 37/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1331 - accuracy: 0.9549 - val_loss: 1.5362 - val_accuracy: 0.6911
Epoch 38/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1273 - accuracy: 0.9559 - val_loss: 1.5684 - val_accuracy: 0.6860
Epoch 39/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1262 - accuracy: 0.9576 - val_loss: 1.5248 - val_accuracy: 0.6893
Epoch 40/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1240 - accuracy: 0.9574 - val_loss: 1.5972 - val_accuracy: 0.6804
Epoch 41/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1253 - accuracy: 0.9567 - val_loss: 1.5865 - val_accuracy: 0.6847
Epoch 42/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1211 - accuracy: 0.9582 - val_loss: 1.6051 - val_accuracy: 0.6862
Epoch 43/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1149 - accuracy: 0.9603 - val_loss: 1.5130 - val_accuracy: 0.6890
Epoch 44/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1115 - accuracy: 0.9618 - val_loss: 1.5589 - val_accuracy: 0.6933
Epoch 45/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1093 - accuracy: 0.9628 - val_loss: 1.6301 - val_accuracy: 0.6897
Epoch 46/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1063 - accuracy: 0.9639 - val_loss: 1.6799 - val_accuracy: 0.6857
Epoch 47/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1038 - accuracy: 0.9638 - val_loss: 1.6592 - val_accuracy: 0.6880
Epoch 48/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1039 - accuracy: 0.9648 - val_loss: 1.7027 - val_accuracy: 0.6818
Epoch 49/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1061 - accuracy: 0.9628 - val_loss: 1.6557 - val_accuracy: 0.6894
Epoch 50/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0961 - accuracy: 0.9677 - val_loss: 1.7069 - val_accuracy: 0.6868
Epoch 51/60
313/313 [==============================] - 6s 18ms/step - loss: 0.1010 - accuracy: 0.9658 - val_loss: 1.7056 - val_accuracy: 0.6882
Epoch 52/60
313/313 [==============================] - 6s 19ms/step - loss: 0.0953 - accuracy: 0.9673 - val_loss: 1.7383 - val_accuracy: 0.6850
Epoch 53/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0917 - accuracy: 0.9682 - val_loss: 1.6973 - val_accuracy: 0.6882
Epoch 54/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0920 - accuracy: 0.9679 - val_loss: 1.7820 - val_accuracy: 0.6889
Epoch 55/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0900 - accuracy: 0.9687 - val_loss: 1.8124 - val_accuracy: 0.6867
Epoch 56/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0904 - accuracy: 0.9696 - val_loss: 1.7334 - val_accuracy: 0.6902
Epoch 57/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0935 - accuracy: 0.9682 - val_loss: 1.7282 - val_accuracy: 0.6924
Epoch 58/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0814 - accuracy: 0.9718 - val_loss: 1.8293 - val_accuracy: 0.6889
Epoch 59/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0883 - accuracy: 0.9699 - val_loss: 1.7023 - val_accuracy: 0.6919
Epoch 60/60
313/313 [==============================] - 6s 18ms/step - loss: 0.0841 - accuracy: 0.9716 - val_loss: 1.7559 - val_accuracy: 0.6898
```

As you can see the accuracy on the train-data is a suitable value.

```
Training time: 383.22622776031494s
test loss:  474.4646911621094
test acc:  59.42000150680542
confusion matrix:

 [[771  31  37  12   7   6   6   5  82  43]
 [ 48 801   0   6   1   6   0   3  35 100]
 [127  20 460  66  56 107  21  71  43  29]
 [ 94  38  58 362  29 238  30  51  50  50]
 [103  20  96  74 379  85  17 165  38  23]
 [ 44  12  63 135  17 576  15  75  32  31]
 [ 54  46  79 122  45  59 463  47  49  36]
 [ 52  22  33  31  36  71   3 694  15  43]
 [131  56  10   7   2   8   0   5 738  43]
 [ 68 169   3   4   2   5   1   9  41 698]]
```

Figure8. Test Results

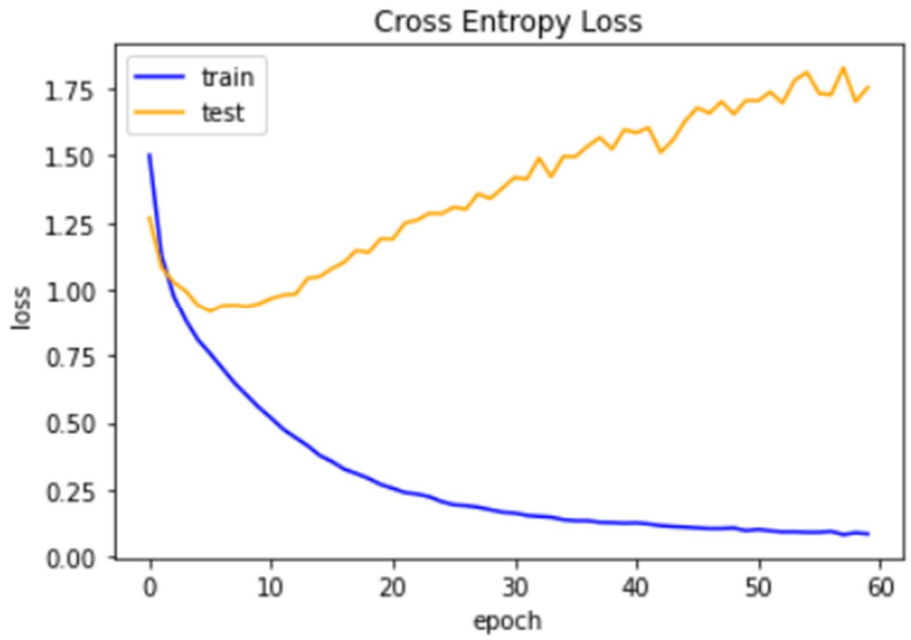


Figure9. Cross Entropy Loss

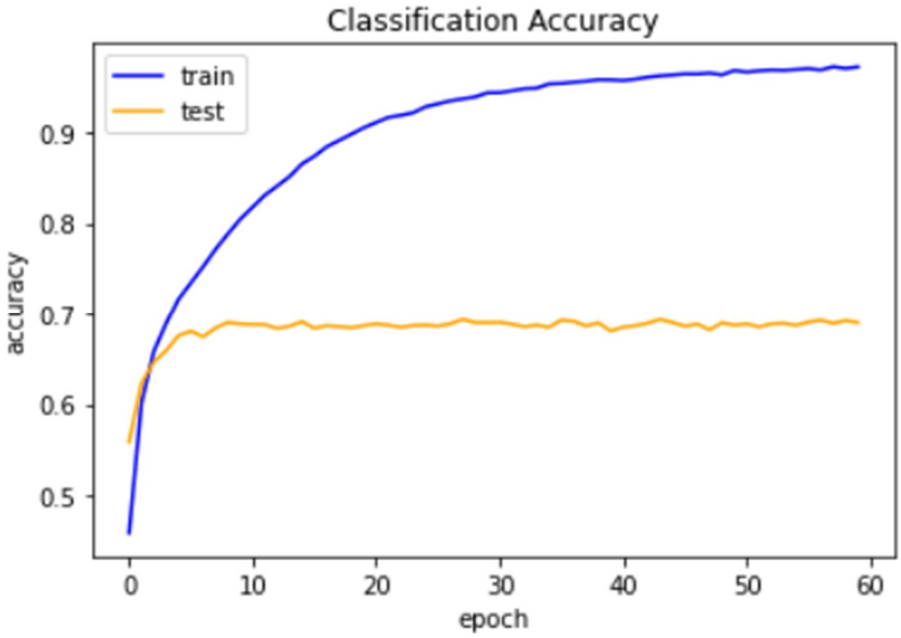


Figure10. Classification Accuracy

2.4 EARLY STOPPING APPROACH IN NEURAL NETWORKS

A problem with training neural networks is in the choice of the number of training epochs to use. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model.

Early stopping is a method that allows you to specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on the validation dataset.

Monitoring Performance: (metrics can be used)

The performance of the model must be monitored during training.

This requires the choice of a dataset that is used to evaluate the model and a metric used to evaluate the model.

It is common to split the training dataset and use a subset, such as 30%, as a validation dataset used to monitor performance of the model during training. This validation set is not used to train the model. It is also common to use the loss on a validation dataset as the metric to monitor, although you may also use prediction error in the case of regression, or accuracy in the case of classification.

The loss of the model on the training dataset will also be available as part of the training procedure, and additional metrics may also be calculated and monitored on the training dataset.

Performance of the model is evaluated on the validation set at the end of each epoch, which adds an additional computational cost during training. This can be reduced by evaluating the model less frequently, such as every 2, 5, or 10 training epochs.

Some of the most important metrics can be monitored during training:

1) Validation-Loss (most common usage)

2) Training-loss (as mentioned above)

3) Precision

4) Recall

5) F-measure

Implementation Early-Stopping:

in this section we are going to implement Early-Stopping on the best model which obtained in the previous section.

Further you can see that the training process has been stopped at the special number of epochs to prevent overfitting:

```
Epoch 1/20
313/313 [==============================] - 13s 10ms/step - loss: 1.6406 - accuracy: 0.4162 - val_loss: 1.2972 - val_accuracy: 0.5397
Epoch 2/20
313/313 [==============================] - 3s 8ms/step - loss: 1.1798 - accuracy: 0.5843 - val_loss: 1.1426 - val_accuracy: 0.6014
Epoch 3/20
313/313 [==============================] - 3s 8ms/step - loss: 1.0313 - accuracy: 0.6361 - val_loss: 1.0180 - val_accuracy: 0.6484
Epoch 4/20
313/313 [==============================] - 3s 9ms/step - loss: 0.9380 - accuracy: 0.6724 - val_loss: 1.0129 - val_accuracy: 0.6470
Epoch 5/20
313/313 [==============================] - 3s 9ms/step - loss: 0.8700 - accuracy: 0.6951 - val_loss: 0.9667 - val_accuracy: 0.6651
Epoch 6/20
313/313 [==============================] - 3s 8ms/step - loss: 0.8120 - accuracy: 0.7169 - val_loss: 0.9863 - val_accuracy: 0.6618
Epoch 7/20
313/313 [==============================] - 3s 9ms/step - loss: 0.7621 - accuracy: 0.7338 - val_loss: 0.9505 - val_accuracy: 0.6738
Epoch 8/20
313/313 [==============================] - 3s 9ms/step - loss: 0.7200 - accuracy: 0.7463 - val_loss: 0.9586 - val_accuracy: 0.6728
Epoch 9/20
313/313 [==============================] - 3s 9ms/step - loss: 0.6793 - accuracy: 0.7605 - val_loss: 0.9538 - val_accuracy: 0.6776
Epoch 10/20
313/313 [==============================] - 3s 9ms/step - loss: 0.6451 - accuracy: 0.7728 - val_loss: 0.9581 - val_accuracy: 0.6799
Epoch 11/20
313/313 [==============================] - 3s 9ms/step - loss: 0.6129 - accuracy: 0.7836 - val_loss: 0.9604 - val_accuracy: 0.6828
Epoch 12/20
307/313 [============================>.] - ETA: 0s - loss: 0.5810 - accuracy: 0.7970Restoring model weights from the end of the best epoch: 7.
313/313 [==============================] - 3s 9ms/step - loss: 0.5807 - accuracy: 0.7969 - val_loss: 0.9855 - val_accuracy: 0.6805
Epoch 12: early stopping
313/313 [==============================] - 1s 3ms/step - loss: 163.3202 - accuracy: 0.5986
```

The train and validation error have been attached below:

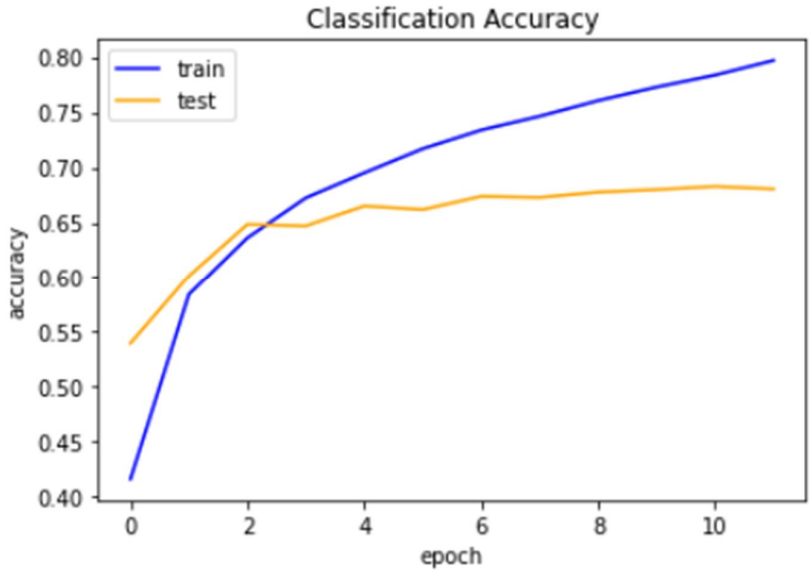


Figure11. Classification Accuracy

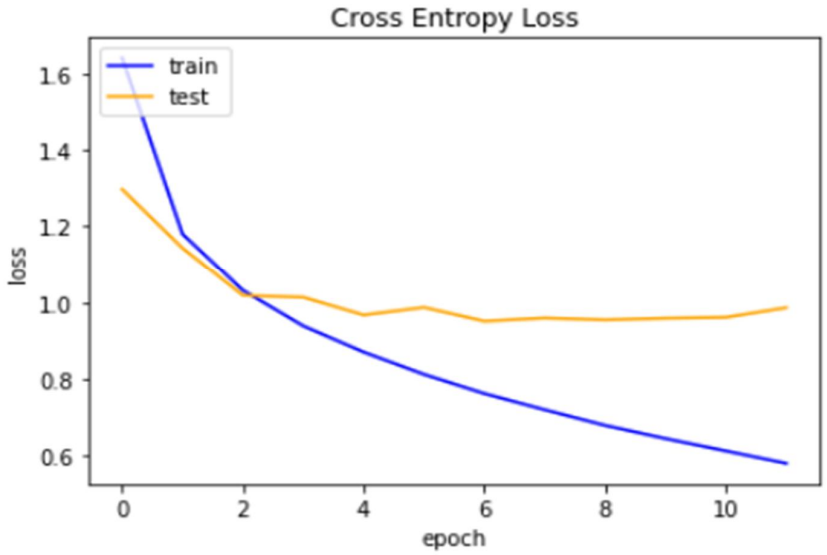


Figure12. Cross Entropy Loss

3 TRANSFER LEARNING

3.1 VGG19

Architecture:

Visual Geometric Group or VGG is a CNN architecture that was introduced 2 years after AlexNet in 2014. The main reason for introducing this model was to see the effect of depth on accuracy while training models for image classification/recognition.

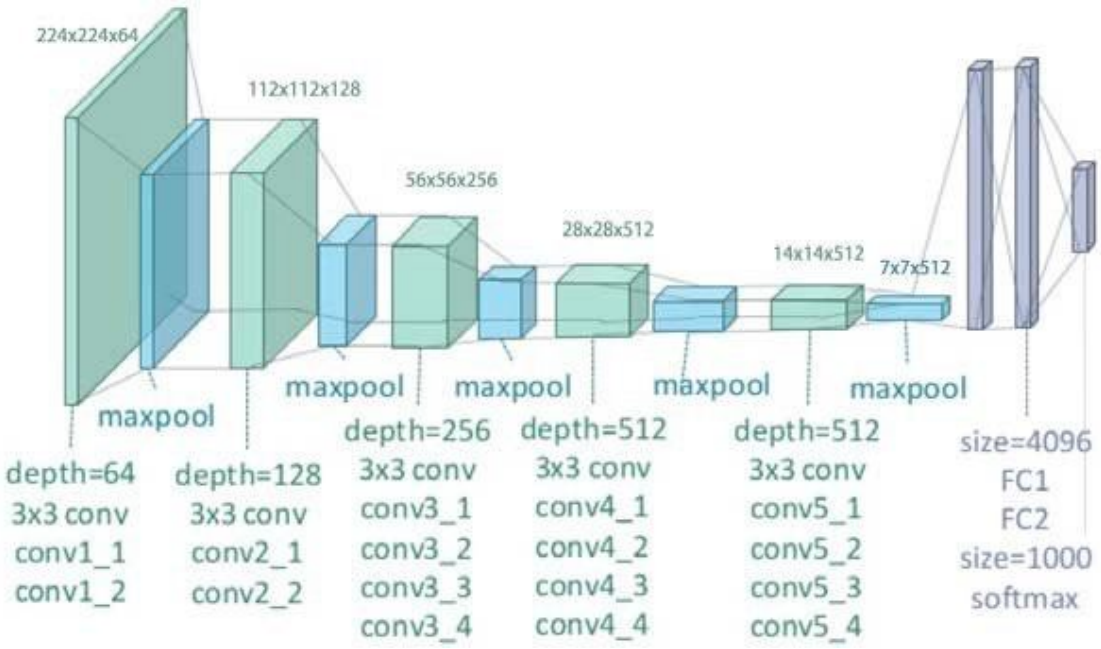


Figure13. VGG-19 Architecture

2Conv — 1Maxpool — 2Conv — 1Maxpool — 4Conv — 1Maxpool — 4Conv — 1Maxpool — 4Conv — 1Maxpool — 1FC — 1FC — 1FC

VGG-19 architectures, due to its depth is slow to train and produce models of very large size. Though the architectures we see here are different, we can create a simple template to perform transfer learning from these models with few lines of code.

Advantages:

- VGG19 brought with it a massive improvement in accuracy and an improvement in speed as well. This was primarily because of improving the depth of the model and also introducing pretrained models.
- The increase in the number of layers with smaller kernels saw an increase in non-linearity which is always a positive in deep learning.

- VGG19 brought with it various architectures built on the similar concept. This gives more options to us as to which architecture could best fit our application.

Disadvantages:

- One major disadvantage that they found was that this model experiences the vanishing gradient problem. This wasn't the case with any of the other models. The vanishing gradient problem was solved with the ResNet architecture.
- VGG19 is slower than the newer ResNet architecture that introduced the concept of residual learning which was another major breakthrough.

Preprocessing:

For VGG19, we need to convert the input images from RGB to BGR, then zero-center each color channel with respect to the ImageNet dataset without scaling as preprocessing.

3.2 TRANSFER LEARNING

Transfer Learning is a machine learning method where we reuse a pre-trained model as the starting point for a model on a new task. To put it simply—a model trained on one task is repurposed on a second, related task as an optimization that allows rapid progress when modeling the second task.

The key idea here is to leverage the pre-trained model's weighted layers to extract features, but not update the model's weights during training with new data for the new task.

The pre-trained models are trained on a large and general enough dataset and will effectively serve as a generic model of the visual world.

3.3 VGG19 PRE-TRAINED BY IMAGENET

We used VGG19 Architecture pre-trained by ImageNet dataset which has more than 14 million images. ImageNet contains more than 20,000 categories with a typical category, such as "balloon", "strawberry" and "animals", consisting of several hundred images.

We pre-processed a picture of a bald eagle shown in Figure? And the fed it to the pre-trained VGG19 model to see the results.

Figure14. Photo of bald eagle used for testing the network

```
bald_eagle (99.81%)
kite (0.16%)
vulture (0.02%)
```

Figure15. Top 3 classes with highest probabilities

3.4 TRANSFER LEARNING AND VGG19

To solidify these concepts, let's walk you through a concrete end-to-end transfer learning & fine-tuning example. We will load the VGG19 model, pre-trained on ImageNet, and use it on the Kaggle "cats vs. dogs" classification dataset.

Some random images of the dataset are shown in figure16

Figure16. Top 3 classes with highest probabilities

As for preprocessing we reshaped the images to 224*224 and converted them from RGB to BGR, then zero-centered each color channel with respect to the ImageNet dataset without scaling.

For transfer learning we froze the weights of all convolutional layers of the model and replaced the last 3 fully connected layers with one fully connected layer with 1 neuron (since we have 2 classes).

Then we trained the network on the dataset.

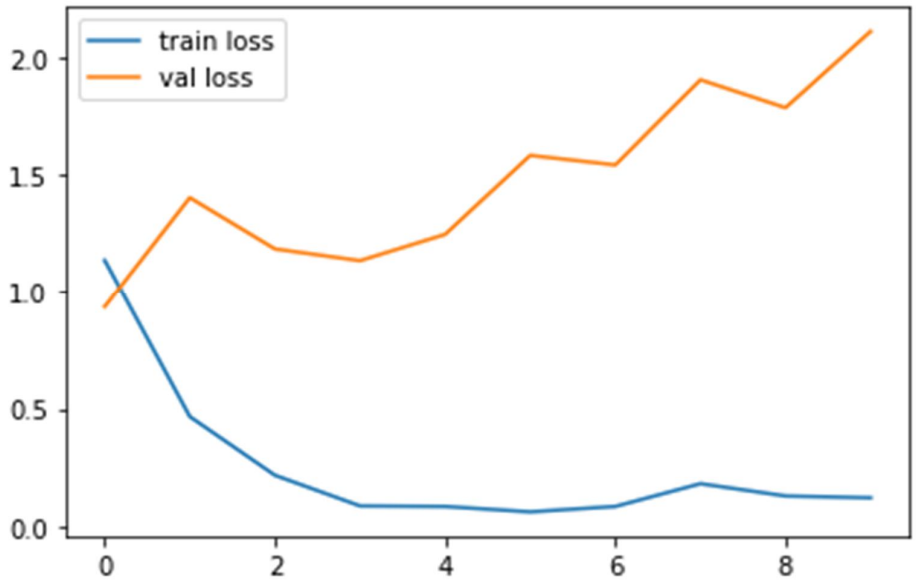


Figure17. Top 3 classes with highest probabilities

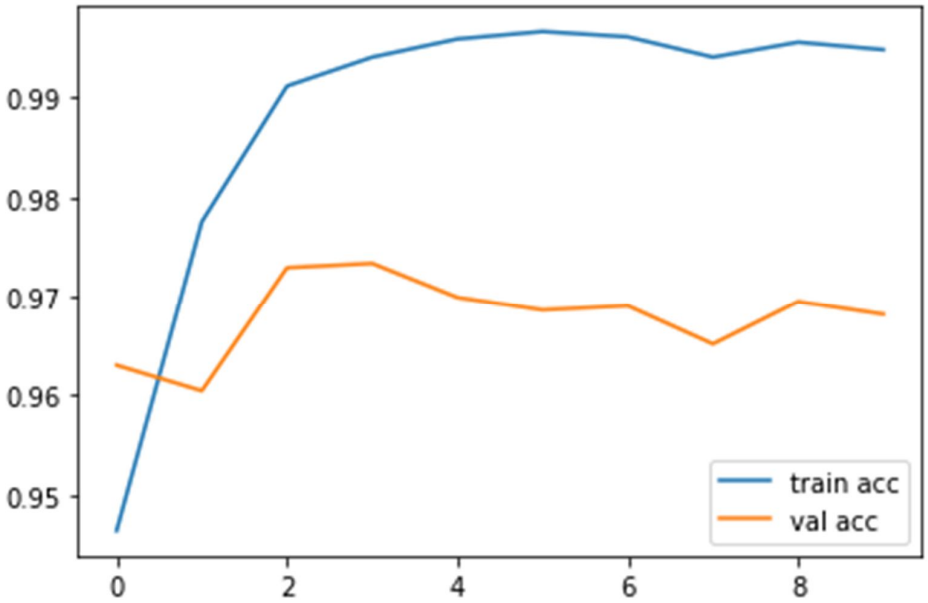


Figure18. Top 3 classes with highest probabilities

By looking at the accuracy and loss plots we can see that 3 to 4 epoch is enough for training our model and more epochs will cause overfitting.

```
Accuracy on test data is: 0.9729148753224419
```

Figure19. Top 3 classes with highest probabilities

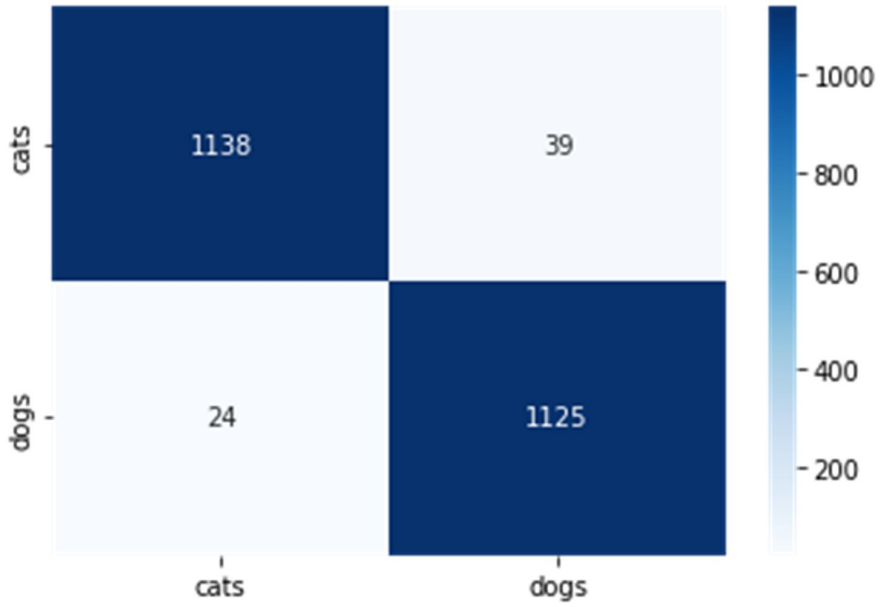


Figure20. Top 3 classes with highest probabilities