ENEE 633 Statistical Pattern Recognition

Project 2 Digit Recognition and Transfer Learning

Pruthvi Sanghavi (116951005)

December 21, 2020

Introduction

The present project deals with the task of designing classifiers. In the first part of the project, a deep learning approach is used to perform classification. A convolutional Neural Network is built to recognize digits. The dataset used for training and testing the classifier is the classic MNIST Dataset. Other classifiers such Kernel SVM (Support Vector Machines) are also implemented using the inbuilt functions. Dimension reduction techniques such as LDA (Linear Discriminant Analysis) and PCA (Principal Component Analysis) are thereafter applied to reduce the dimension of the dataset.
For the second part of the project, transfer learning is used to perform monkey species classification.

The milestone of the project is to analyse these classifiers and present a comparative study of their accuracy.

Data set

For the first part, the dataset used is the classical MNIST Dataset which consists of training set of 60000 28x28 grayscale images of hand written digits (10 classes) and testing set with 10000 images. The example of the images are shown below.

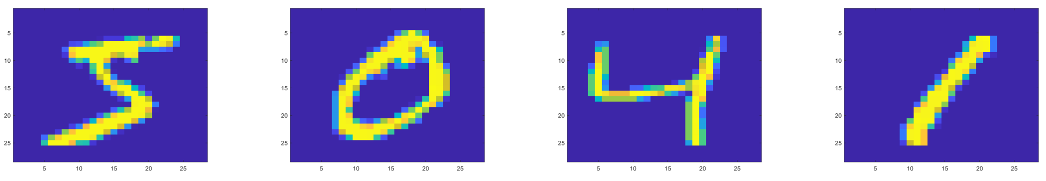


Figure 1: MNIST Dataset Hand Written Digits

For the second part, the dataset used is the kaggle monkey dataset with 10 classes and with 1400 images each with a different pixel dimension.

Part 1: Hand Written Digit Recognition

Linear and Kernel SVM

To implement linear and kernel SVM, **keras** and **sklearn** libraries are used.

Linear SVM

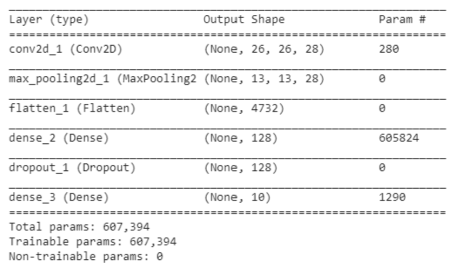
For linear SVM, inbuilt function are used to train the model, the slack parameter is fixed to 1. The testing accuracy in this case is found out to be 92.8%
With Principal Component Analysis, testing accuracy increased to 93.71% this is because using PCA the dimension of the data is reduced and as a result, features become separable allowing a better classification.
With Multi Discriminant Analysis. The value of the testing accuracy has decreased to 85.28%

Kernel SVM

For Polynomial Kernel SVM, inbuilt functions are used to train the classifier. Here the testing accuracy is found to be 96.0%
When PCA and MDA are applied to the polynomial Kernel SVM task, the testing accuracy shows a similar trend of respective increase(96%) and decrease(90.88%) for PCA and MDA. For Radial Basis Function Kernel, the testing accuracy is found out to be 96.56%. Which improved to 96.85% for PCA and decreased for MDA to 92.07%

Convolutional Neural Network

For this task **Google's Tensorflow Library** is used for building the Convolutional Neural Network. The CNN is trained on the MNIST Dataset to classify hand written Digits. The accuracy of the classifier depends upon a lot of parameters such as Number of layers in the network, types of layer in the network, Number of epochs etc.
A custom architecture is designed by tweaking the parameters to get the best accuracy.
The summary of the custom built model is given as:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 28)        280
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 28)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4732)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               605824
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 607,394
Trainable params: 607,394
Non-trainable params: 0
```

Figure 2: Model Summary

Variation of Accuracy with number of epochs

Number of epochs is the number of times the training data is shown to the network while training. With an increase in the number of epochs the training accuracy increases. The number of epochs is increased until the validation accuracy starts decreasing even when training accuracy is increasing(overfitting). The ideal number of epochs here is 15 since at 20 epochs the test accuracy is less.

Number of Epochs	Training Accuracy	Testing Accuracy
5	98.89%	98.43%
10	99.57%	98.32%
15	99.71%	98.57%
20	99.65%	98.48%

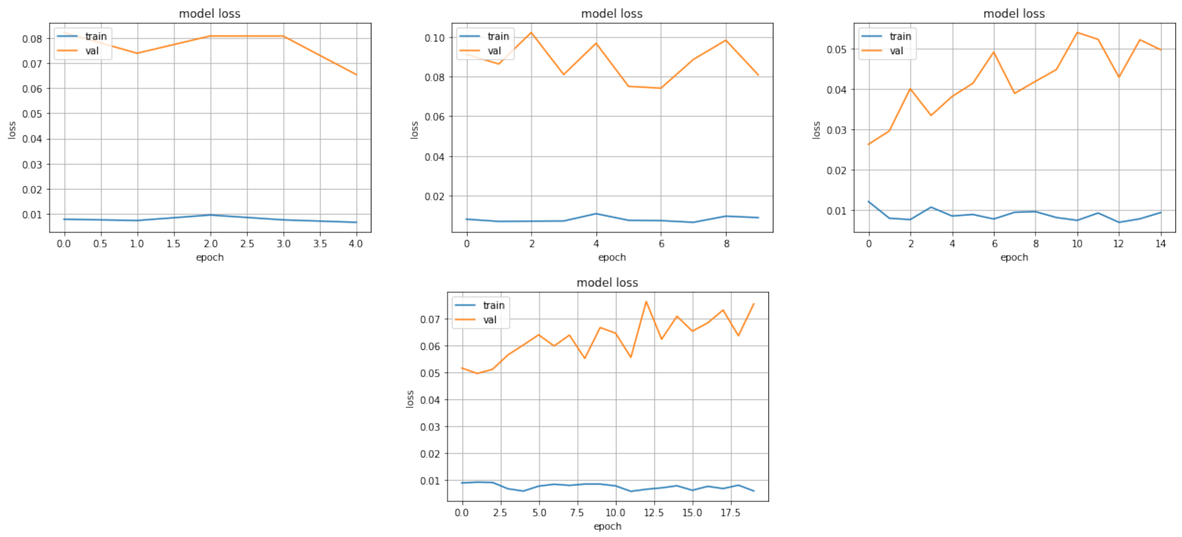


Figure 3: Number of Epoch vs Loss

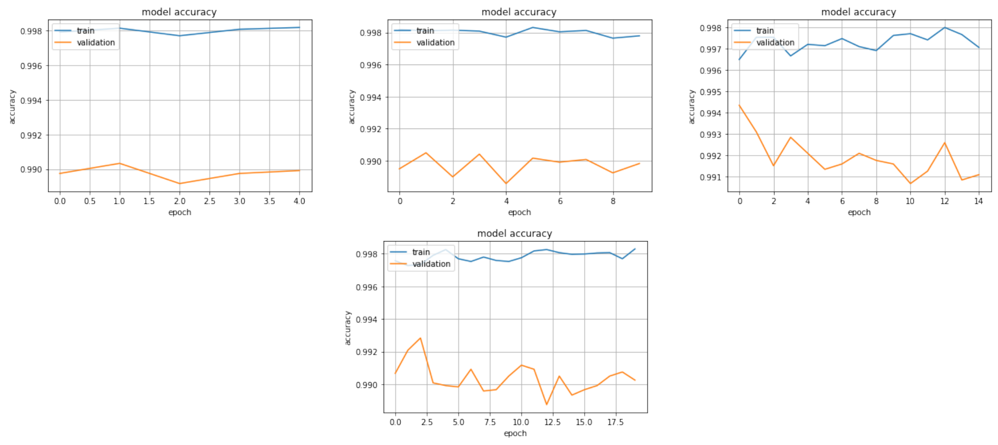


Figure 4: Number of Epoch vs Accuracy

Variation of Accuracy with Batch Size

It is the number of sub samples given to the network after which the parameter update happens.Experiments are done 5 epochs. Larger Batch size has a negative effect on the accuracy of the network. Larger batch size helped in speeding up the computation.

Batch Size	Training Accuracy	Testing Accuracy
32	97.06%	98.00%
64	98.75%	98.27%
128	99.71%	98.41%
256	99.11%	99.11%
1024	99.31%	98.29%

Variation of Accuracy with Dropout

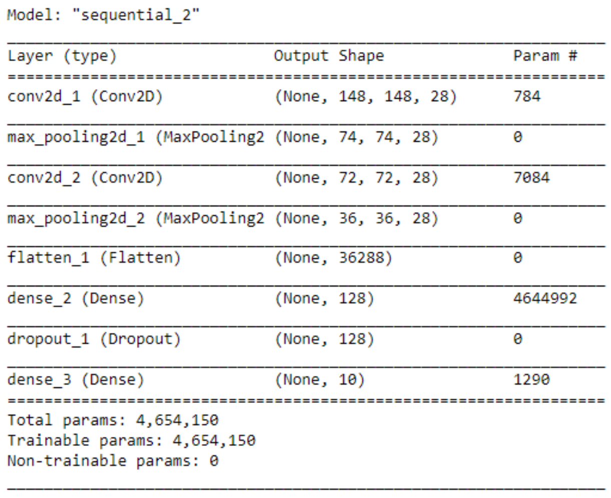
Dropout is regularization technique to avoid overfitting (increase the validation accuracy) thus increasing the generalizing power. These experiments are done for 10 epochs.

Batch Size	Training Accuracy	Validation Accuracy	Testing Accuracy
0.1	99.62%	98.37%	98.19%
0.2	99.38%	98.55%	98.43%
0.3	99.14%	98.39%	98.09%

Part 2: Transfer Learning

Transfer Learning helps in boosting up the accuracy of the neural networks when the size of the dataset is small.

Initially, a custom model is built which consists of 2 convolutional layer to check how it performs when the number of samples are less. The architecture of this model is given as follows,

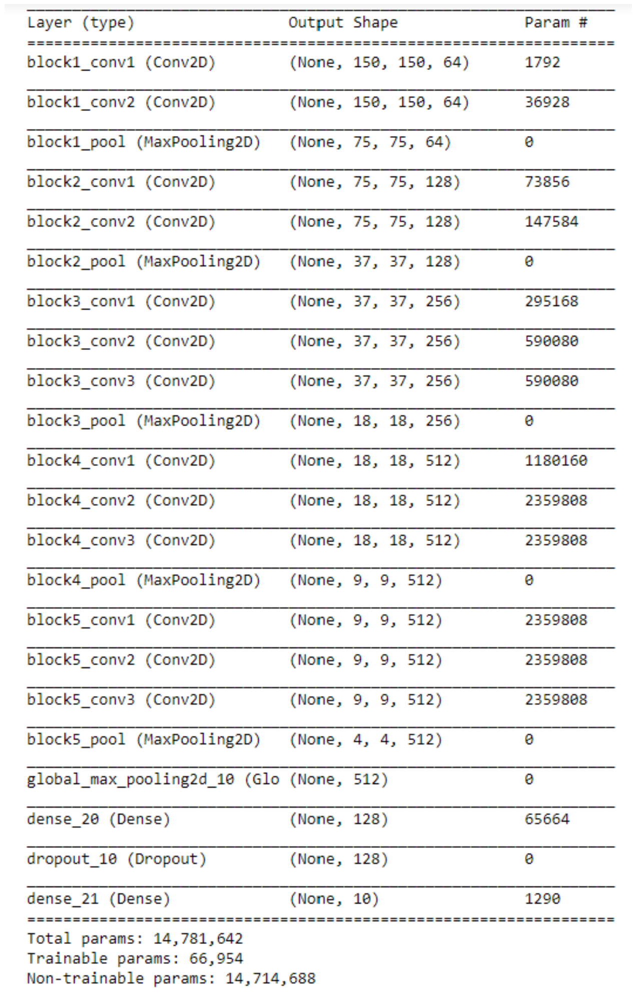


```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 28)      784
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 28)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 28)        7084
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 28)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 36288)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               4644992
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 4,654,150
Trainable params: 4,654,150
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Model Summary

Number of Epochs	Training Accuracy	Validation Accuracy	Testing Accuracy
5	60.62%	56.620%	55.882%
10	65.90%	59.930%	58.088%
20	73.75%	60.29%	58.088%

It can be seen from the table that the testing accuracy of the model comes out to be less. Now we will use Transfer Learning to increase the accuracy. A pre-trained VGG16 model is used and the last three layers of the network are activated and the remaining layers are deactivates(freeze) so that the parameters are not updated in those layers. Further, Convolutional layer is unfreezed to see the effects. The transfer learning model can be summarized as follows,



```
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
global_max_pooling2d_10 (Glo (None, 512)               0
_________________________________________________________________
dense_20 (Dense)             (None, 128)               65664
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1290
=================================================================
Total params: 14,781,642
Trainable params: 66,954
Non-trainable params: 14,714,688
```

Figure 6: VGG Model Summary

It can be referred from the above table, that most of the parameters are un-trainable and only the parameters which belongs to the last three layers are trainable.

Number of Epochs	Training Accuracy	Validation Accuracy	Testing Accuracy
5	60.60%	68.38%	68.382%
10	85.95%	82.35%	83.824%
20	96.86%	84.93%	84.926%

As can be seen from the table, after transfer learning the testing accuracy of the model has increased.

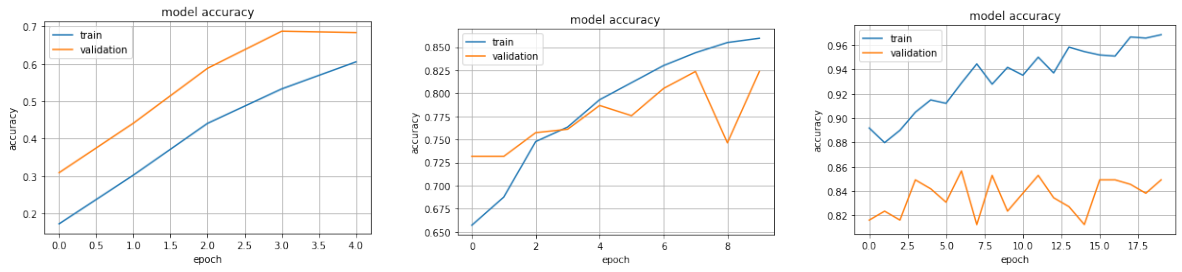


Figure 7: Number of Epoch vs Accuracy

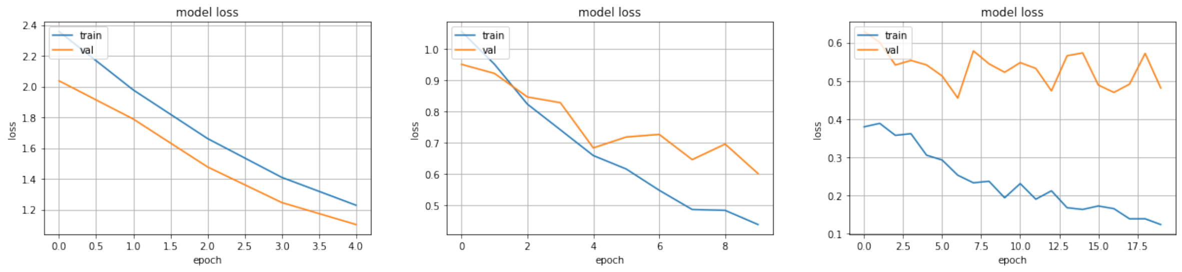


Figure 8: Number of Epoch vs Loss

Conclusions

In conclusion, SVM and CNN are implemented for the recognition of digits and then transfer learning is used to increase the accuracy of the pre trained network in case when the number of samples are not sufficient. PCA and LDA are also applied on SVM to anayse their accuracy. CNN implemented on the MNIST dataset showed a good testing accuracy after the training because of a large number of samples. The time required while training the SVM classifier is also high.

Custom CNN applied on monkey dataset did not show a good accuracy improvement. The training time required while performing the Transfer Learning was high and that seems quite contradictory since we are freezing the VGG16 and training only the dense layer, the training should take less time.