Name: Rohith Kumar Poshala

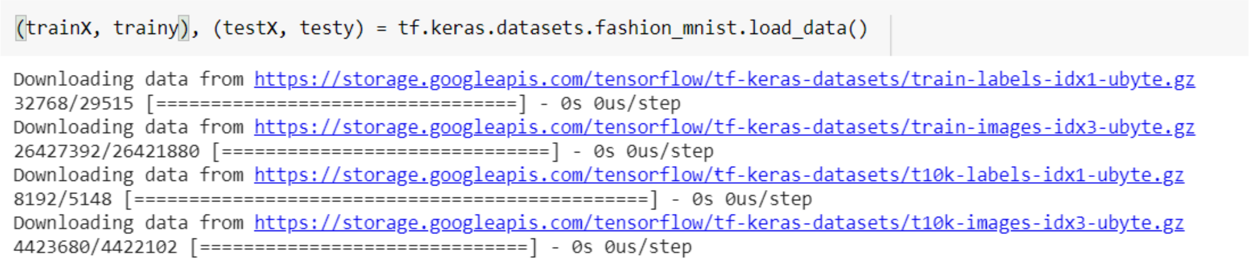
rposhala@buffalo.edu

Fashion-MNIST Image Classification

In this project, Convolutional Neural Networks were built, trained for image classification task for Fashion-MNIST dataset.

CNN models, techniques for preventing overfitting and various data augmentation tools were explored and CNN models were fine-tuned to achieve validation/test accuracy more than 92%.

Dataset: As said, Fashion-MNIST dataset was picked for this project. This dataset had been loaded from Keras API with the help of below command:

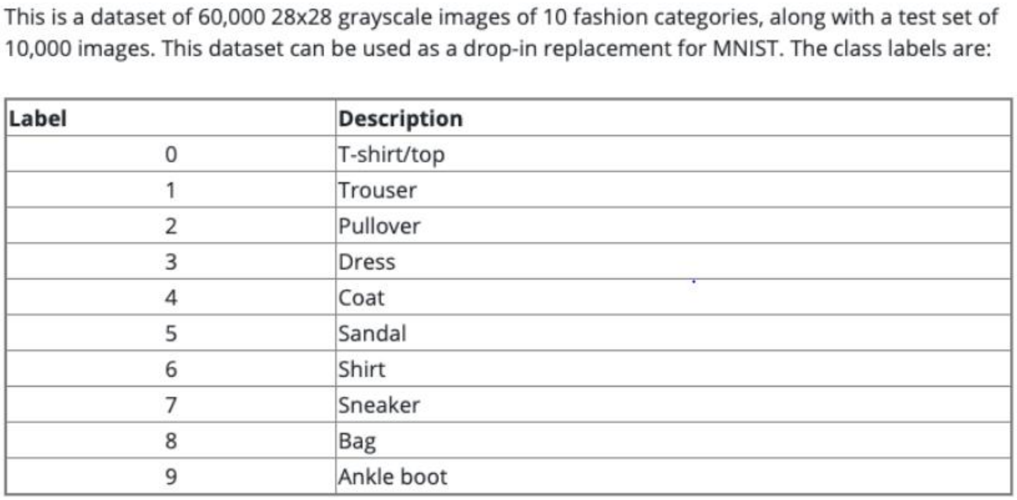


```
(trainX, trainy), (testX, testy) = tf.keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

The default train and test split had 60,000 and 10,000 examples respectively with each example of 28 X 28 grayscale image, associated with a label from 10 classes.

Description of the labels:

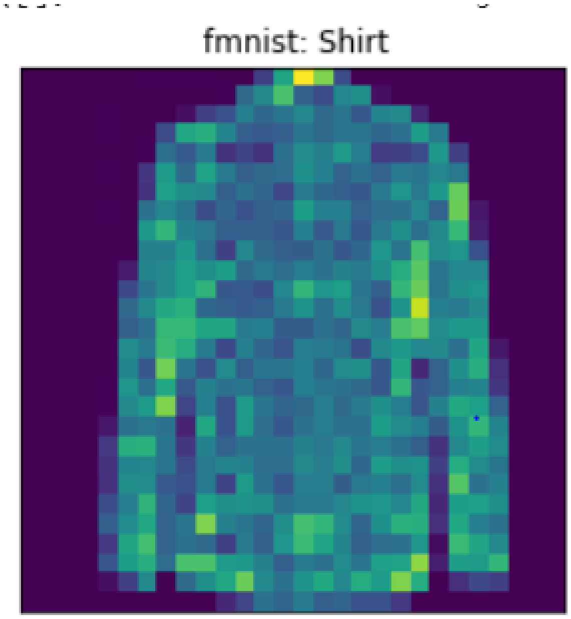


This is a dataset of 60,000 28x28 grayscale images of 10 fashion categories, along with a test set of 10,000 images. This dataset can be used as a drop-in replacement for MNIST. The class labels are:

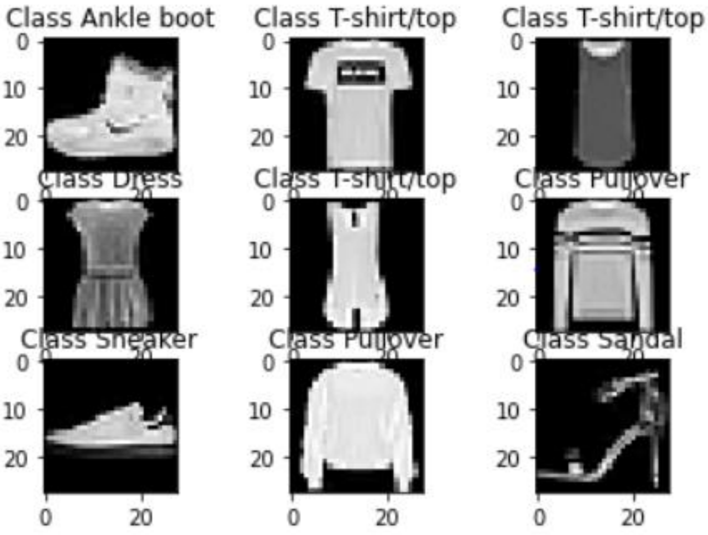
Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

A dictionary was created with the labels and their corresponding names.

Visualizing random sample from traning data with its corresponding label:

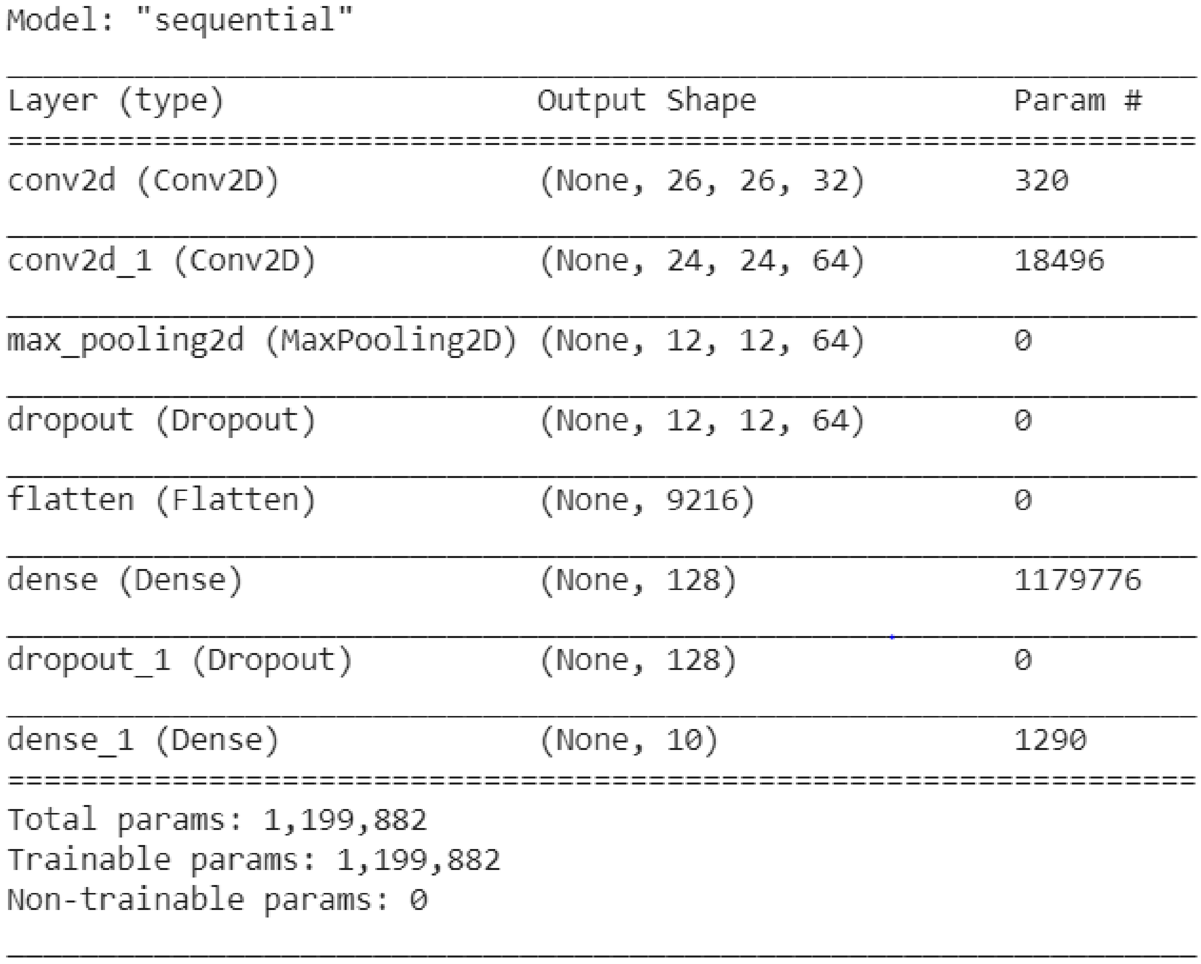


Visualizing the images in gray scale:



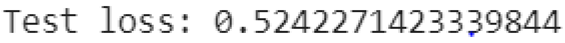
Training and test data have been reshaping and normalized. Labels were One-Hot Encoded. Reshaping the data and One-Hot encoding the labels were done to get the training/testing samples and labels shape compatible to the CNN model.

ConvNet 1: A ConvNet with 3 layers (2 convolutional layers and a MaxPooling layer) along with dropouts was built and trained.

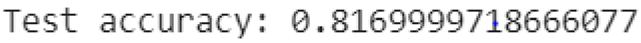


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1179776
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

The above model was fit on the training data and validated on the testing data with batch size of 128 and 64 epochs.

Validation/Test loss obtained: 

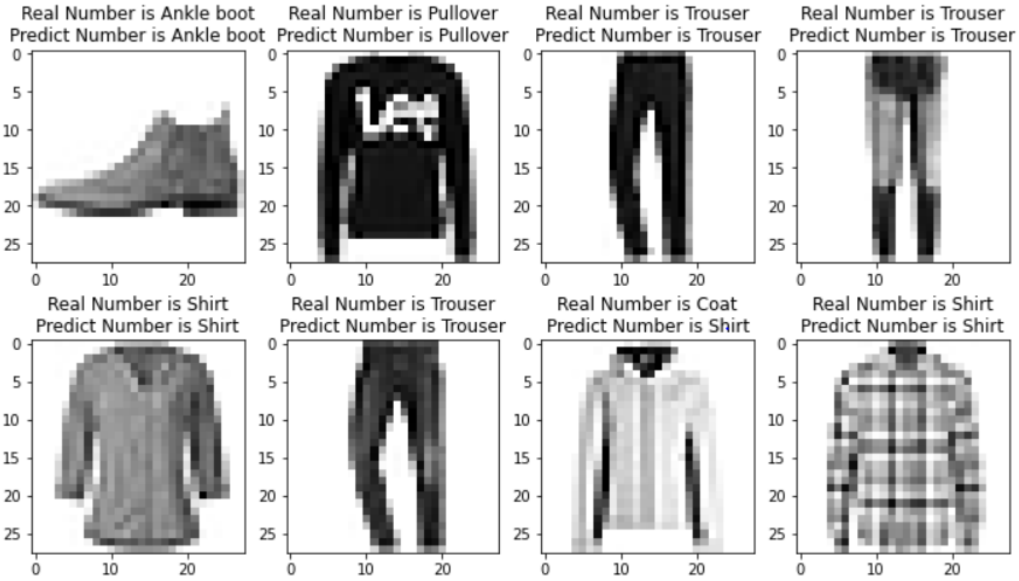
```
Test loss: 0.5242271423339844
```

Validation/Test accuracy obtained: 

```
Test accuracy: 0.8169999718666077
```

But Accuracy of 81.6% on this dataset is comparatively less. Confusion matrix was plotted to look into the model performance on each class. And Train Versus Validation accuracy and loss were plotted across all epochs to observe the trend and decide on which the changes which needs to be done to the CNN model.

Checking the predicted with the true labels:



Real Number is Ankle boot
Predict Number is Ankle boot

Real Number is Pullover
Predict Number is Pullover

Real Number is Trouser
Predict Number is Trouser

Real Number is Trouser
Predict Number is Trouser

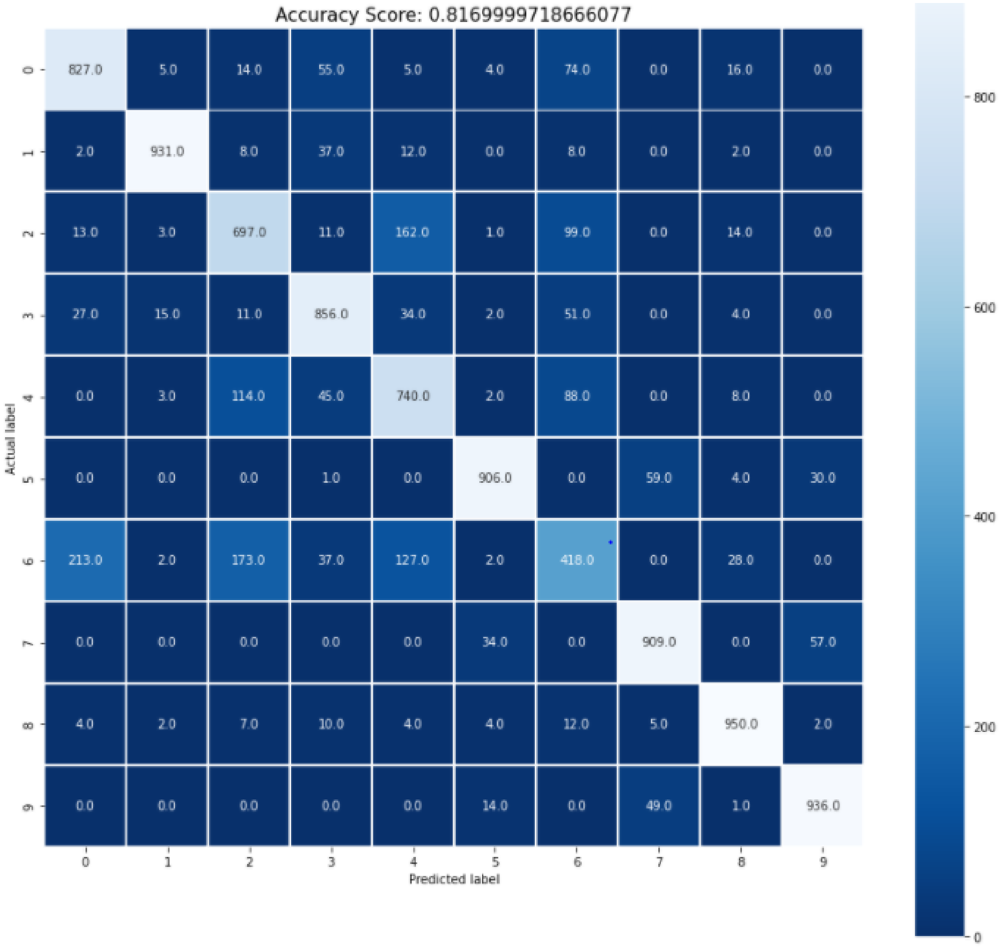
Real Number is Shirt
Predict Number is Shirt

Real Number is Trouser
Predict Number is Trouser

Real Number is Coat
Predict Number is Shirt

Real Number is Shirt
Predict Number is Shirt

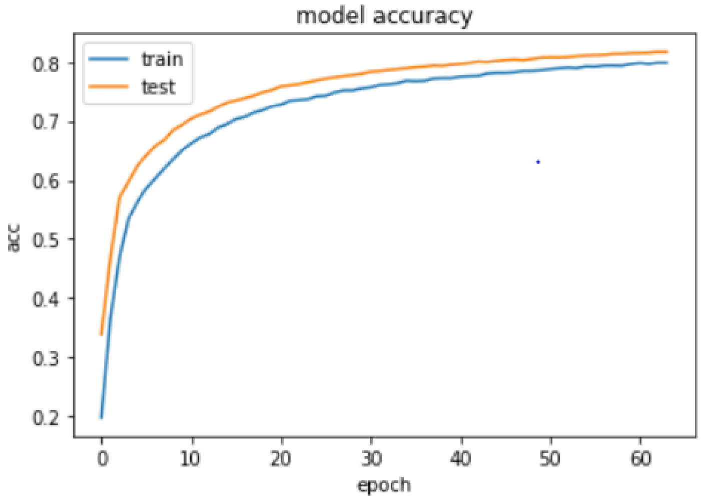
Confusion Matrix:



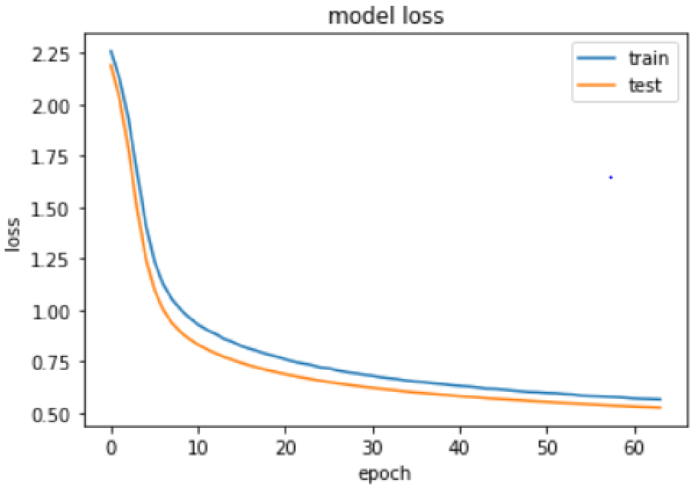
Accuracy Score: 0.8169999718666077

	0	1	2	3	4	5	6	7	8	9
0	827.0	5.0	14.0	55.0	5.0	4.0	74.0	0.0	16.0	0.0
1	2.0	931.0	8.0	37.0	12.0	0.0	8.0	0.0	2.0	0.0
2	13.0	3.0	697.0	11.0	162.0	1.0	99.0	0.0	14.0	0.0
3	27.0	15.0	11.0	856.0	34.0	2.0	51.0	0.0	4.0	0.0
4	0.0	3.0	114.0	45.0	740.0	2.0	88.0	0.0	8.0	0.0
5	0.0	0.0	0.0	1.0	0.0	906.0	0.0	59.0	4.0	30.0
6	213.0	2.0	173.0	37.0	127.0	2.0	418.0	0.0	28.0	0.0
7	0.0	0.0	0.0	0.0	0.0	34.0	0.0	909.0	0.0	57.0
8	4.0	2.0	7.0	10.0	4.0	4.0	12.0	5.0	950.0	2.0
9	0.0	0.0	0.0	0.0	0.0	14.0	0.0	49.0	1.0	936.0

Train Vs Validation Accuracy:



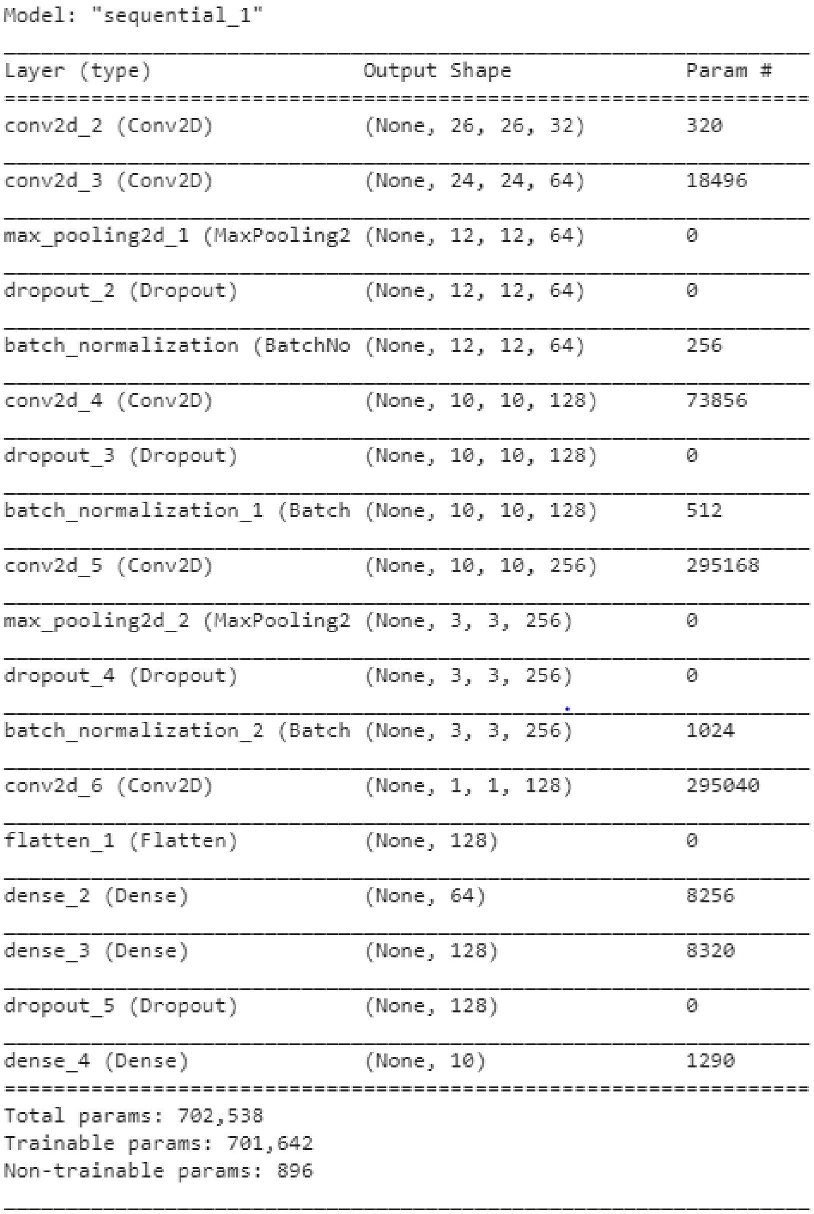
Train Vs Validation Loss:



Observation:

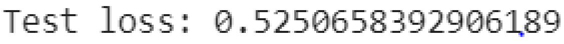
We can observe that test accuracy and training accuracy are increased over time for each epoch. Simialrly, model loss for training and testing had decreased event ually. And clearly, there is no sign of overfitting.

ConvNet 2: Few more convolutional and MaxPooling layers were added to the previous ConvNet.

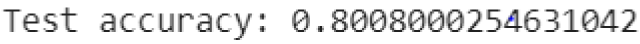


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 64)        0         
_________________________________________________________________
batch_normalization (BatchNo (None, 12, 12, 64)        256       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 128)       73856     
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 10, 128)       0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 10, 10, 128)       512       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 256)       295168    
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 256)         0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 3, 3, 256)         0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 3, 256)         1024      
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 1, 1, 128)         295040    
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_3 (Dense)              (None, 128)               8320      
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290      
=================================================================
Total params: 702,538
Trainable params: 701,642
Non-trainable params: 896
_________________________________________________________________
```

The above model was fit on the training data and validated on the testing data with batch size of 128 and 64 epochs.

Validation/Test loss obtained: 

```
Test loss: 0.5250658392906189
```

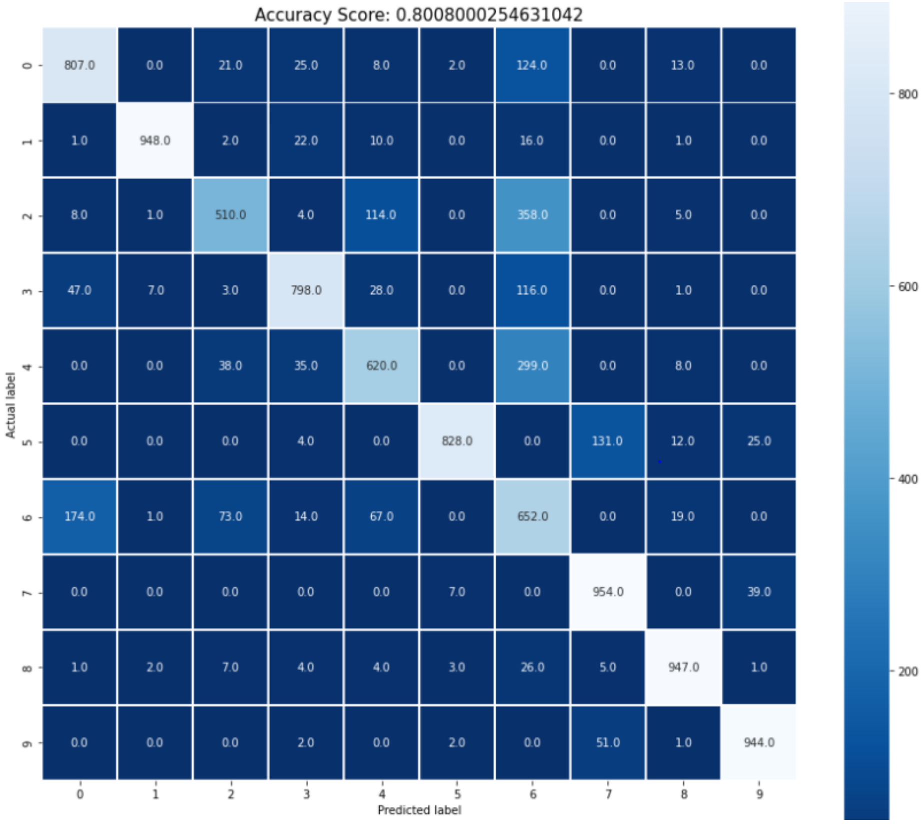
Validation/Test accuracy obtained: 

```
Test accuracy: 0.8008000254631042
```

But Accuracy of 80% on this dataset is comparatively less. Confusion matrix was plotted to look into the model performance on each class. And Train Versus

Validation accuracy and loss were plotted across all epochs to observe the trend and decide on which the changes which needs to be done to the CNN model.

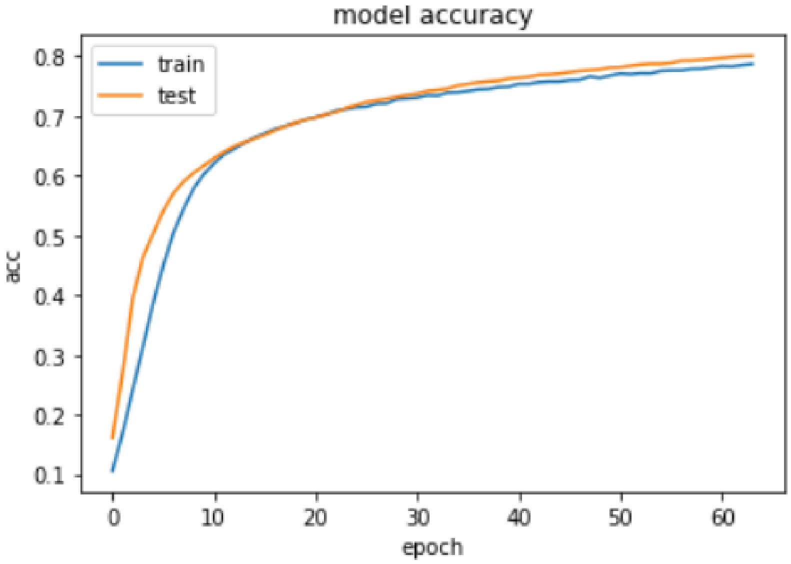
Confusion Matrix:



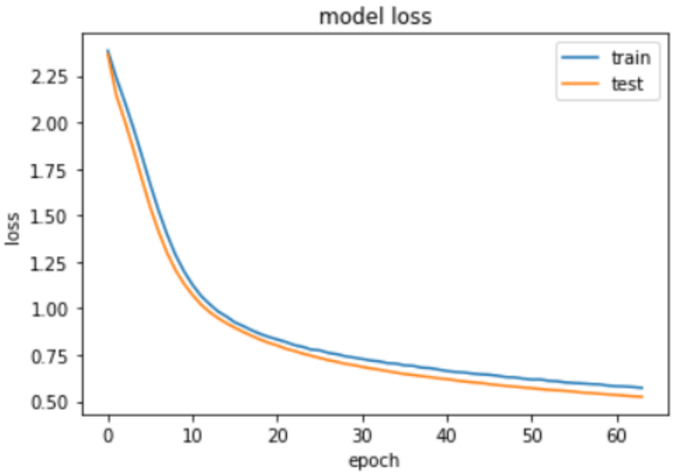
Accuracy Score: 0.8008000254631042

	0	1	2	3	4	5	6	7	8	9
0	807.0	0.0	21.0	25.0	8.0	2.0	124.0	0.0	13.0	0.0
1	1.0	948.0	2.0	22.0	10.0	0.0	16.0	0.0	1.0	0.0
2	8.0	1.0	510.0	4.0	114.0	0.0	358.0	0.0	5.0	0.0
3	47.0	7.0	3.0	798.0	28.0	0.0	116.0	0.0	1.0	0.0
4	0.0	0.0	38.0	35.0	620.0	0.0	299.0	0.0	8.0	0.0
5	0.0	0.0	0.0	4.0	0.0	828.0	0.0	131.0	12.0	25.0
6	174.0	1.0	73.0	14.0	67.0	0.0	652.0	0.0	19.0	0.0
7	0.0	0.0	0.0	0.0	0.0	7.0	0.0	954.0	0.0	39.0
8	1.0	2.0	7.0	4.0	4.0	3.0	26.0	5.0	947.0	1.0
9	0.0	0.0	0.0	2.0	0.0	2.0	0.0	51.0	1.0	944.0

Train Vs Validation Accuracy:



Train Vs Validation Loss:



Observation:

We can observe an increase in the performance of the model when the model was made deep by adding additional convolutional layers. But the difference was not that significant. Model was not able to learn at a greater rate as expected and there is no overfitting visibility (no variance).

It is a clear case of avoidable bias. For this we can try either Bigger network or different optimizers or increasing training dataset size.

The most efficient way is to increase the training dataset size by doing data Augmentation.

Improving CNN model and implementing Data Augmentation techniques:

As said above in the observation of the previous ConvNet model, the best way to counter the problem of avoidable bias is by increasing size of the training dataset, this can be done by data augmentation.

As part of data Augmentation, a function was defined with the help NumPy function like random choice and roll to shift the images up to 10 pixels to the right or left or up or bottom randomly (number of pixels to be shifted parameter was tuned).

Along with this Image Data Generator class from tensorflow was also used to do the data augmentation.

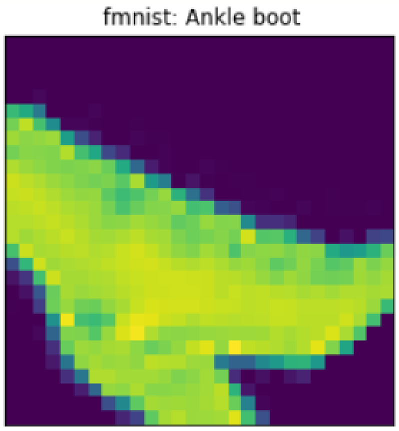
But as the image shift in the image data generator couldn’t be micro tuned, we had used the above defined customized image shift function in it.

Using Image Data Generator for rotation, random cropping (zoom), horizontal flip and vertical flip. This is data augmentation function is then fitted to the recently formed training data (of length 120000).

Each of the above training image is passing to the function with batch size 1 to get the generated images which can be included into training data.

These Augmented images are generated by shift up to 10 pixels, rotation up to 60 degrees, random zoom till 0.1, constant fill mode and random horizontal and vertical flips.

A Sample image from the Augmented data:

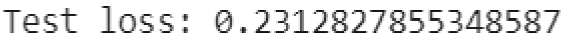


Finally, the size of training dataset has been increased to 2,40,000 samples with the help of data augmentation.

ConvNet 3: Previous ConvNet 1 was fit on Augmented data.

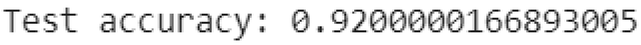
This ConvNet 3(altered ConvNet 1) model was fit on the training data and validated on the testing data with batch size of 128 and 64 epochs.

Validation/Test loss obtained:



```
Test loss: 0.2312827855348587
```

Validation/Test accuracy obtained:

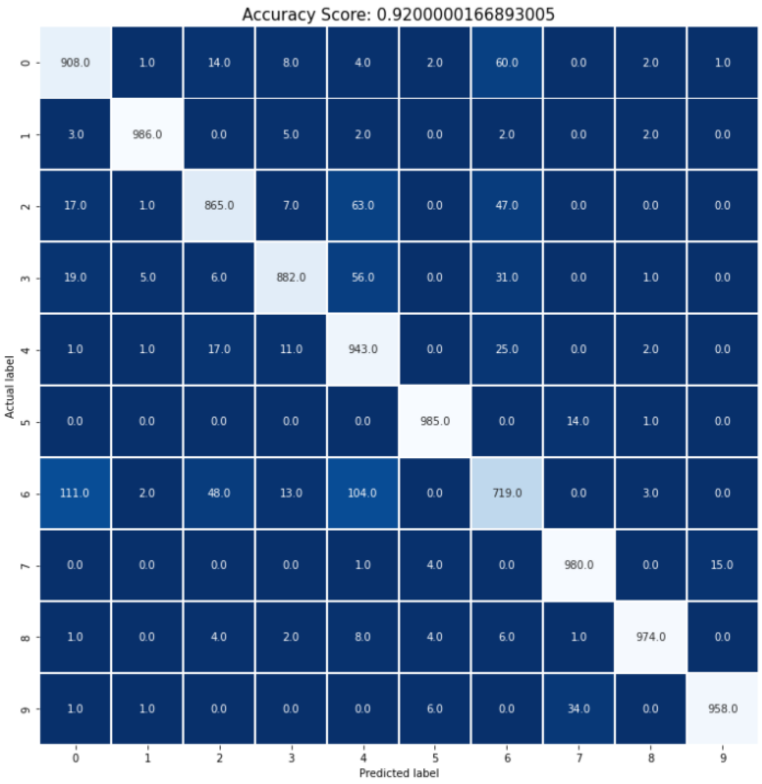


```
Test accuracy: 0.9200000166893005
```

Our main goal was reached by achieving 92% with the help data augmentation.

The Testing accuracy had crosses 92% after the 5th epoch and settled at exactly 92 % by the end of 8th epoch.

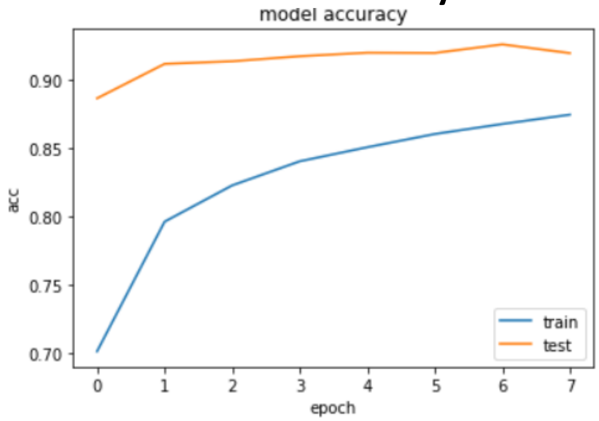
Confusion Matrix:



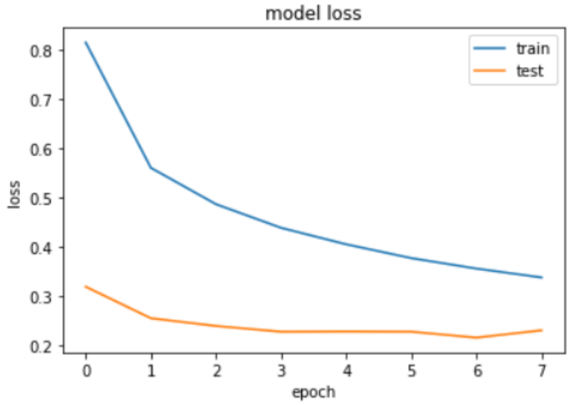
Accuracy Score: 0.9200000166893005

Actual label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	908.0	1.0	14.0	8.0	4.0	2.0	60.0	0.0	2.0	1.0
1	3.0	986.0	0.0	5.0	2.0	0.0	2.0	0.0	2.0	0.0
2	17.0	1.0	865.0	7.0	63.0	0.0	47.0	0.0	0.0	0.0
3	19.0	5.0	6.0	882.0	56.0	0.0	31.0	0.0	1.0	0.0
4	1.0	1.0	17.0	11.0	943.0	0.0	25.0	0.0	2.0	0.0
5	0.0	0.0	0.0	0.0	0.0	985.0	0.0	14.0	1.0	0.0
6	111.0	2.0	48.0	13.0	104.0	0.0	719.0	0.0	3.0	0.0
7	0.0	0.0	0.0	0.0	1.0	4.0	0.0	980.0	0.0	15.0
8	1.0	0.0	4.0	2.0	8.0	4.0	6.0	1.0	974.0	0.0
9	1.0	1.0	0.0	0.0	0.0	6.0	0.0	34.0	0.0	958.0

Train Vs Validation Accuracy:



Train Vs Validation Loss:



Observation:

The Testing accuracy had crosses 92% after the 5th epoch and settled at exactly 92% by the end of 8th epoch.

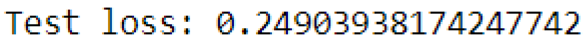
In this ConvNet we have used Dropouts to put the effect of regularization, we can see that the network was still learning and trying to get good accuracies.

And we can also observe that model had fit test accuracy more efficiently than training data.

ConvNet 4: ConvNet 3 with increased epochs to let the model learn more

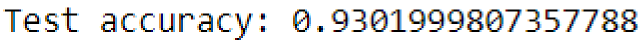
This ConvNet 4 model was fit on the training data and validated on the testing data with batch size of 128 and 96 epochs.

Validation/Test loss obtained:



```
Test loss: 0.24903938174247742
```

Validation/Test accuracy obtained:



```
Test accuracy: 0.9301999807357788
```

We can observe a decrease in the loss function and improved training accuracies. We also observe testing accuracy crossing 92% and reaching 93% by the end 96 epochs.

Observation:

We can observe a decrease in the loss function and improved training accuracies. We also observe testing accuracy crossing 92% and reaching 93% by the end 32 epochs.

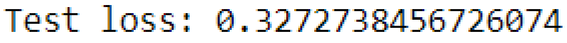
Apparently ReduceLROnPlateau had been affective after increasing the epochs and allowing the model to learn more. We can also observe that with the help of ReduceLROnPlateau, model reduced its learning rate as it neared the optimal solution.

The increase of training accuracy was more significant and evident than the increase of the testing accuracy. By the end of final epoch, training accuracy had crossed 93% and looked like it may overfit if we train further.

ConvNet 5: Applying ConvNet3, by adding regularizers to convolutional layers for the previous convnet model

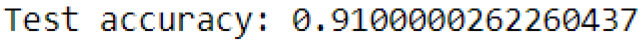
This ConvNet 5 model was fit on the training data and validated on the testing data with batch size of 128 and 32 epochs.

Validation/Test loss obtained:



```
Test loss: 0.3272738456726074
```

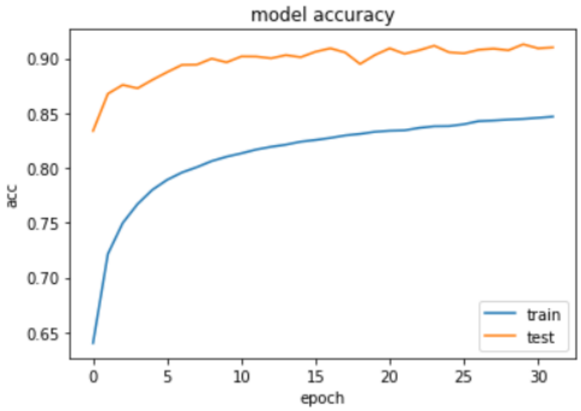
Validation/Test accuracy obtained:



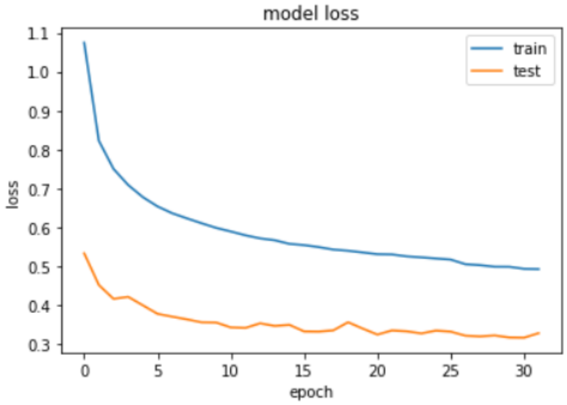
```
Test accuracy: 0.9100000262260437
```

We can observe that adding regularizers has stopped training data from overfitting. But Testing accuracy can be seen settling at 91% which are a little lesser than previous convnet model.

Train Vs Validation Accuracy:



Train Vs Validation Loss:



Observation:

We can observe that adding regularizers has stopped training data from overfitting. But Testing accuracy can be seen settling at 91% which are a little lesser than previous convnet model.

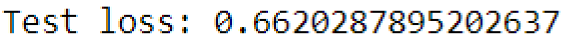
And we can clearly see from the above model loss plot and the accuracies plot that the model got has stopped improving its training accuracy and reducing training loss as effectively as the previous models where regularization was not used.

Early stopping was also implemented to the current model to observe its effect on the validation loss and accuracy.

ConvNet 6: Applying ConvNet4, by adding early stopping to convolutional layers for the previous convnet model.

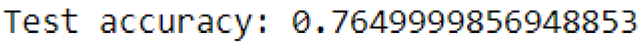
This ConvNet 6 model was fit on the training data and validated on the testing data with batch size of 128 and 32 epochs.

Validation/Test loss obtained:



```
Test loss: 0.6620287895202637
```

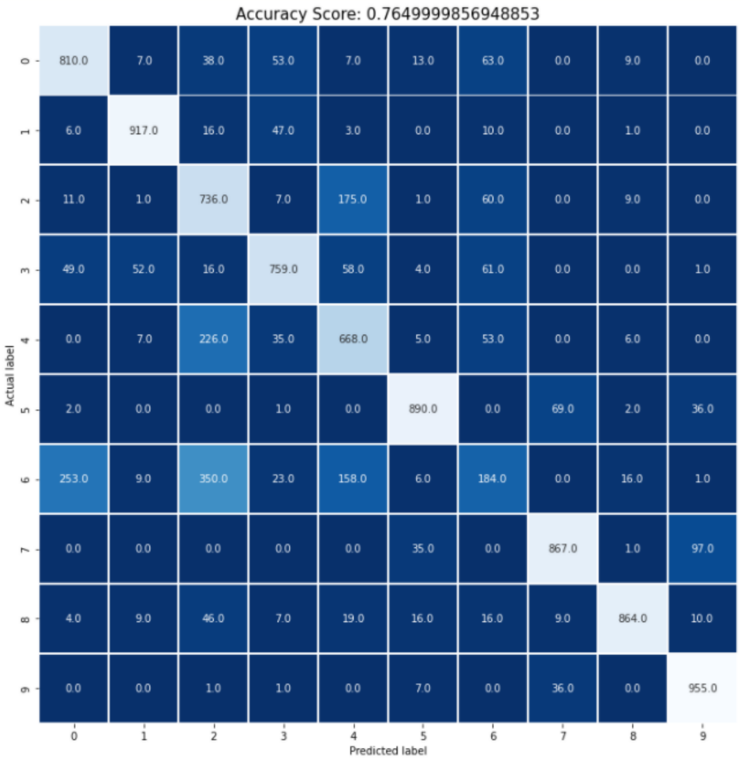
Validation/Test accuracy obtained:



```
Test accuracy: 0.7649999856948853
```

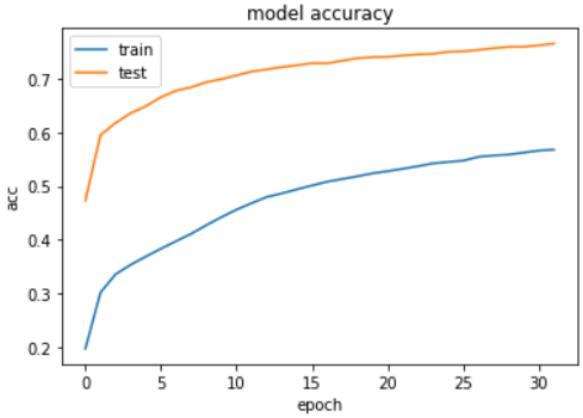
We can observe that early stopping defined to control the training loss to the default 'auto' mode, had made the model limit the reduction of the training loss, resulting in the lesser accuracies than the previous models.

Confusion Matrix:

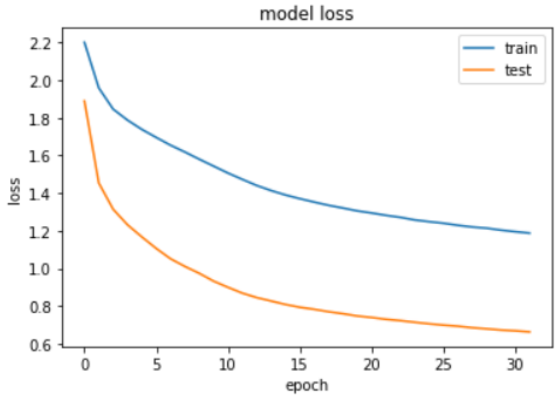


	0	1	2	3	4	5	6	7	8	9
0	810.0	7.0	38.0	53.0	7.0	13.0	63.0	0.0	9.0	0.0
1	6.0	917.0	16.0	47.0	3.0	0.0	10.0	0.0	1.0	0.0
2	11.0	1.0	736.0	7.0	175.0	1.0	60.0	0.0	9.0	0.0
3	49.0	52.0	16.0	759.0	58.0	4.0	61.0	0.0	0.0	1.0
4	0.0	7.0	226.0	35.0	668.0	5.0	53.0	0.0	6.0	0.0
5	2.0	0.0	0.0	1.0	0.0	890.0	0.0	69.0	2.0	36.0
6	253.0	9.0	350.0	23.0	158.0	6.0	184.0	0.0	16.0	1.0
7	0.0	0.0	0.0	0.0	0.0	35.0	0.0	867.0	1.0	97.0
8	4.0	9.0	46.0	7.0	19.0	16.0	16.0	9.0	864.0	10.0
9	0.0	0.0	1.0	1.0	0.0	7.0	0.0	36.0	0.0	955.0

Train Vs Validation Accuracy:



Train Vs Validation Loss:

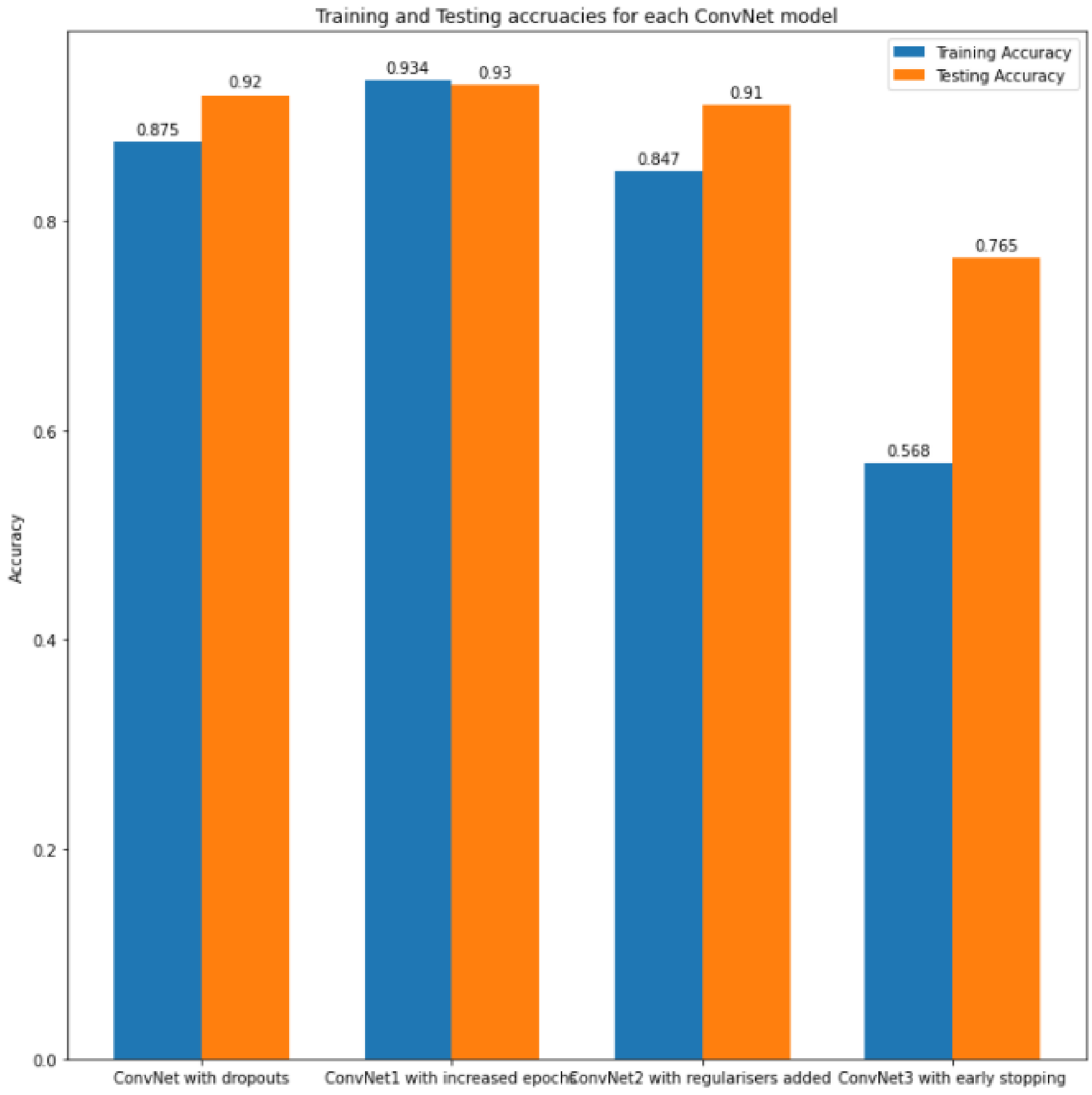


Observation:

We can observe that early stopping defined to control the training loss to the default 'auto' mode, had made the model limit the reduction of the training loss, resulting in the lesser accuracies than the previous models.

May be early stopping would be more effective for very deep models where the model doesn't learn after certain epochs. It's a time and resource saver in that case.

Comparing the performance of each ConvNet model fit on Augmented training dataset:



Observation:

From the above plot, we can understand that we have achieved good testing accuracies which neared training accuracies. This implies that the models are not overfitted to the training data. In my opinion, to increase the testing accuracy, we can afford to let the model learn more on training data. But with the existing training data, it may take a lot of epochs and runtime to achieve better results.

Instead we can do more data augmentation and provide the model with training data, to achieve good training and testing accuracies.