Multi-Layer Neural Network:

It is a dataset comprised of 60,000 small square 28×28 pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The mapping of all 0-9 integers to class labels is listed below.

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

- I started with a simple 3 layer neural network .In the first layer the input(28*28) matrix was flattened to 784 .The second layer is a dance layer and the activation function was relu (Also tried log and tanh) . It was tries with both 64 and 128 neutrons. The last layer is a softmax activation function which clarifies it to 10 classes

- I chose the *sparse_categorical_crossentropy* loss function. Cross-entropy is the default loss function to use for a multi-class classification problem and it's sparse because our targets are not one-hot encodings but are integers
- Adam is used as the optimiser because it gave better results . Even the nor mat gradient decent can be used but its difficult to find the correct hyper-parameters.
- Now training is done using training data so we give it the input image and expect the output label
- To minimize overfitting used validation split (0.2)
- Here we can see that the test loss is 35.7 and accuracy is 88.1 for this neural network
- Next we can a deeper network give better results
- I tried with 6. And 12 layer neural network. With layer our test loss is slightly higher at 37.5 and the test accuracy is slightly lower at 86.6 than the NN-3. So our model got a little worse

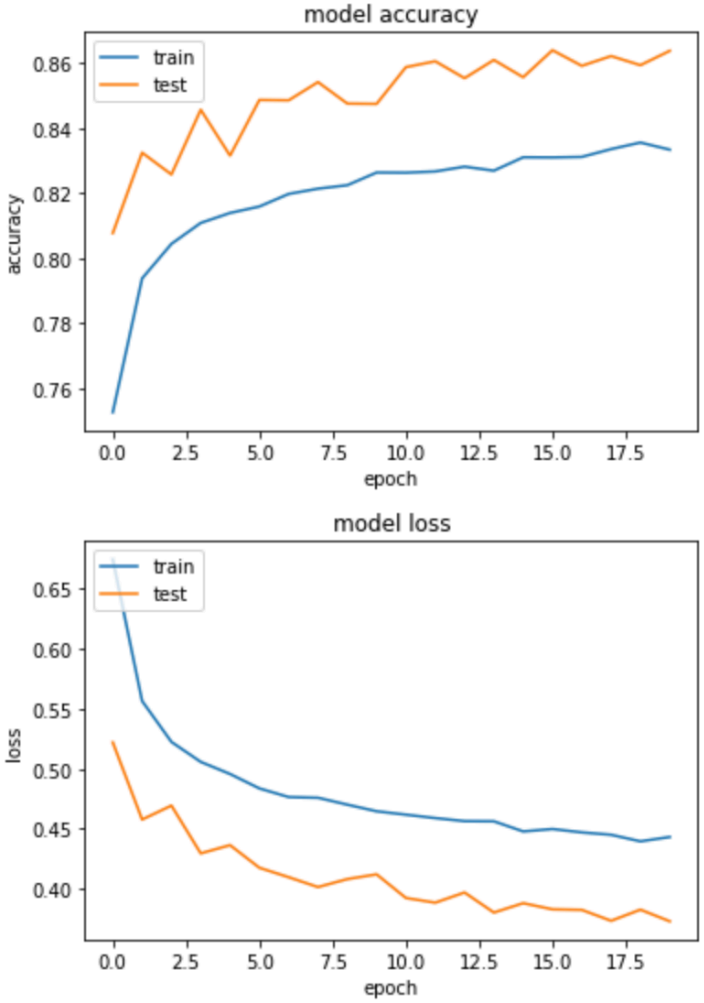
```
[14] test_loss, test_acc = model_3.evaluate(test_images, test_labels)

     10000/10000 [==============================] - 0s 37us/sample - loss: 0.3782 - acc: 0.8645
```

- Even with 12 layer the model got little worse
- Also applied data Augmentation . Used Gaussian Noise(0.5) (Also tried with 0.1 , 0.01 as parameter for Gaussian Noise and 0.5 gave the

```
model_3 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),keras.layers.GaussianNoise(0.5),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

best results



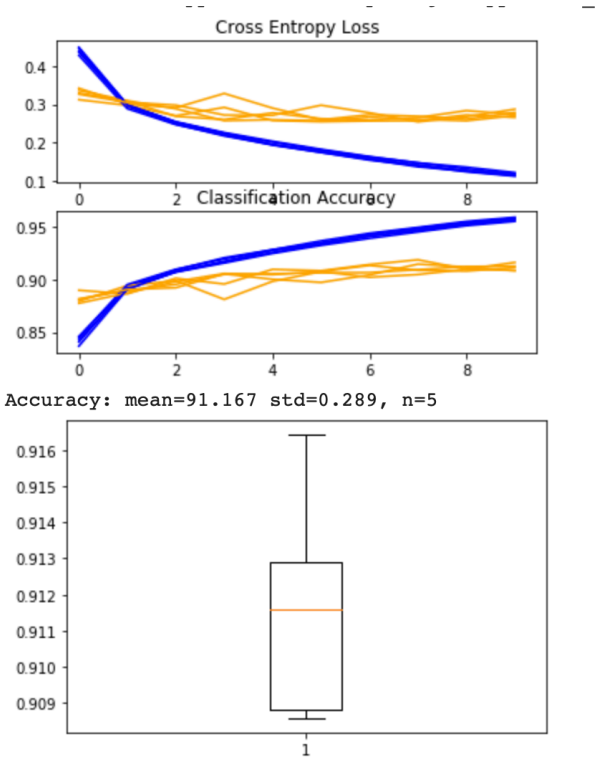

```
sess = tf.Session()
with sess.as_default():
        print(sess.run(con_mat))
```

```
[[840   1  23  19   4   0  95   0  17   1]
 [  5 969   0  18   3   0   3   0   2   0]
 [ 13   0 758  13 119   0  88   0   8   1]
 [ 33   3   8 882  40   0  27   0   6   1]
 [  1   0  77  25 814   0  77   0   6   0]
 [  0   0   0   0   0 861   0  81   2  56]
 [152   0  87  28  78   0 624   0  31   0]
 [  0   0   0   0   0   4   0 942   0  54]
 [  0   0   2   4   3   0   1   4 985   1]
 [  0   0   0   0   0   1   0  28   1 970]]
```

- Tried to use dropout but there were no much notable change in accuracy

Convolution Neural Network

- Started with a single convolutional layer with filter size (3,3) and a modes no of filters (32) followed by a max pooling layer .the filter maps are then flattened to give features to the classifier .
- For all the layers RELU activation is used and for classification s0ft max is used .And HE weight initialisation is used
- Stochastic gradient decent with learning rate 0.009 and a momentum go 0.9 used and categorical Cross entropy function is minimised
- The model is evaluated using k fold Kross validation

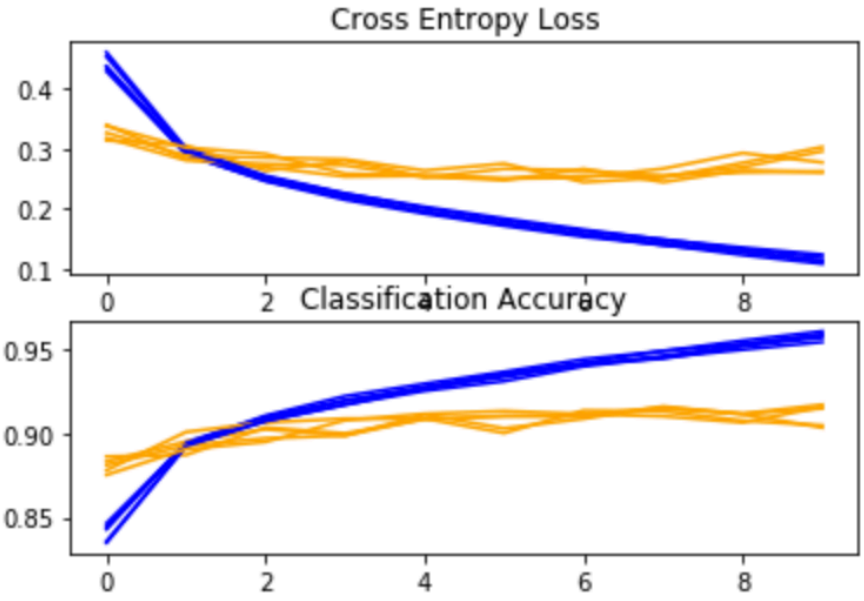


- The training dataset is shuffled prior to being split and the sample shuffling is performed each time so that any model we evaluate will have the same train and test datasets in each fold
- Accuracy for the first model

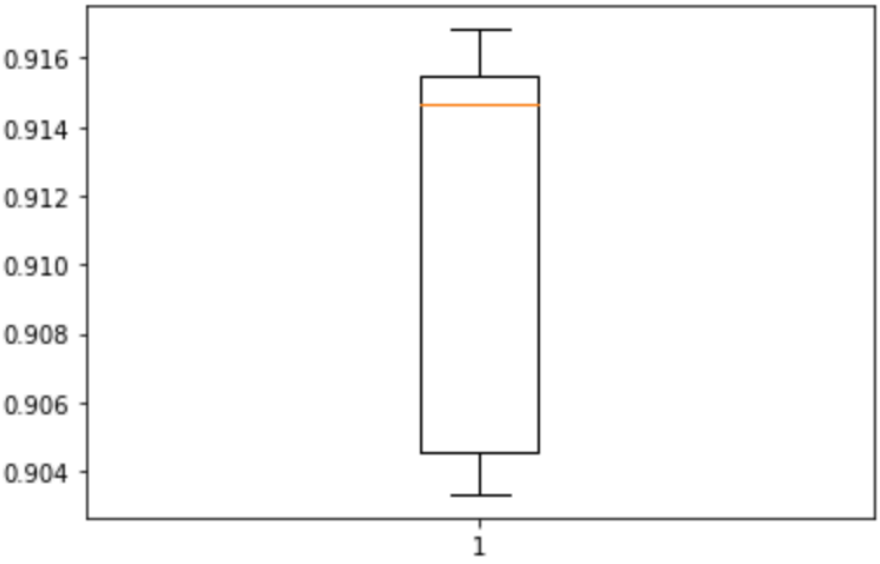
```
> 91.292
> 91.642
> 90.883
> 91.158
> 90.858
```

- The second model is improved from the first model by adding padding . Padding often results in better model performance .as more of the input image of feature maps are given an opportunity to participate or contribute to the output
- Results of the second model

```
> 90.333
> 91.467
> 91.683
> 90.458
> 91.550
```

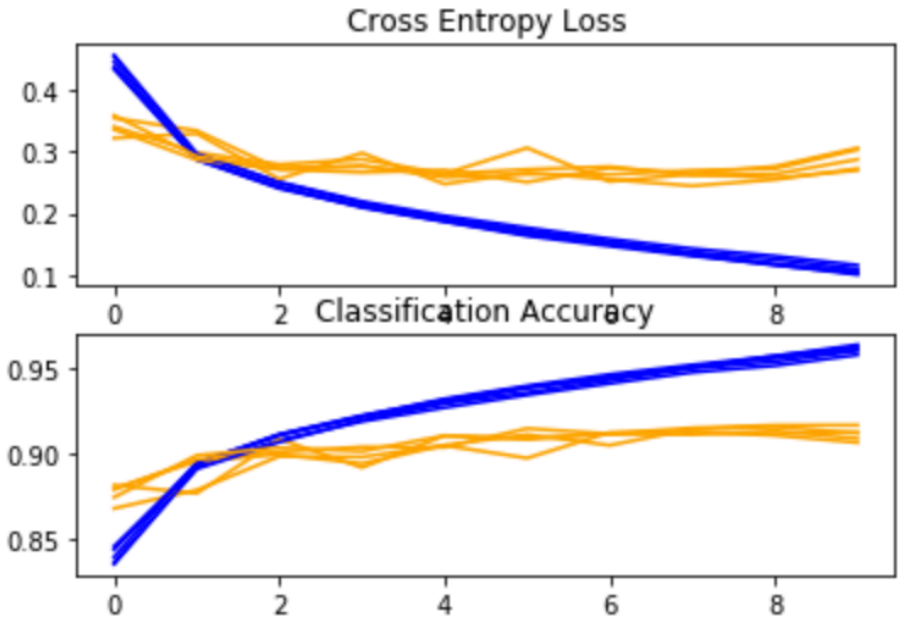



```
Accuracy: mean=91.098 std=0.579, n=5
```



- An increase in the number of filters used in the convolutional layer can often improve performance, as it can provide more opportunity for extracting simple features from the input images. So we increase it from 32 to 64
- Results of this model

```
> 91.292
> 91.692
> 91.217
> 90.675
> 90.908
```




```
Accuracy: mean=91.157 std=0.347, n=5
```