CSE 474/574: Introduction to Machine Learning (Fall 2019) Classification problem using Neural Network and Convolutional Neural Network on Fashion-MNIST

Sonali Naidu
Department of Computer Science
University at Buffalo
Buffalo, NY – 14226
sonaliye@buffalo.edu

Abstract

In this project we have worked on implementing single layer and multilayer Neural Networks and Convolutional Neural Network on Fashion MNIST for solving the classification problem. The task here is to train the classifiers using the dataset to recognize the image and identify it as one of the ten classes. The dataset on which we train the classifier is Fashion- MNIST. We have 10 classes into which each of the image from dataset gets classified.

Introduction

Neural Networks is a series of algorithm, modeled loosely after the human brain, that endeavors to recognize the underlying relationships or patterns in a set of data. Neural networks refer to system of neurons, that interpret sensory data through a kind of machine perception, labeling or clustering raw input. Neural Network can adapt to changing input hence the network generates the best possible result without needing to redesign the output criteria.

Convolutional Neural Network is a deep learning algorithm that takes in an input image and assigns the importance (i.e weight and biases) to various aspects in the image and be able to differentiate from one another. This is achieved with the local connections and the tied weights followed by some form of poolomg which results in translation of invariant features. CNNs are the regularized versions of multilayer perceptrons which usually mean fully connected networks, that is, each neuron in a layer is connected to all neurons in the next layer. As the network is fully connected to all the neurons in the next layer the model becomes prone to overfitting data.

In this project we have worked on building the

1. Single layer Neural Network to be trained and tested on Fashion-MNIST dataset.
2. Multi Layer Neural Network using Keras to be trained and tested on Fashion-MNIST dataset.
3. Convolutional Neural Network (CNN) using Keras to be trained and tested on Fashion-MNIST dataset.
4. Evaluate the results obtained by each of this classifier.

Algorithm:

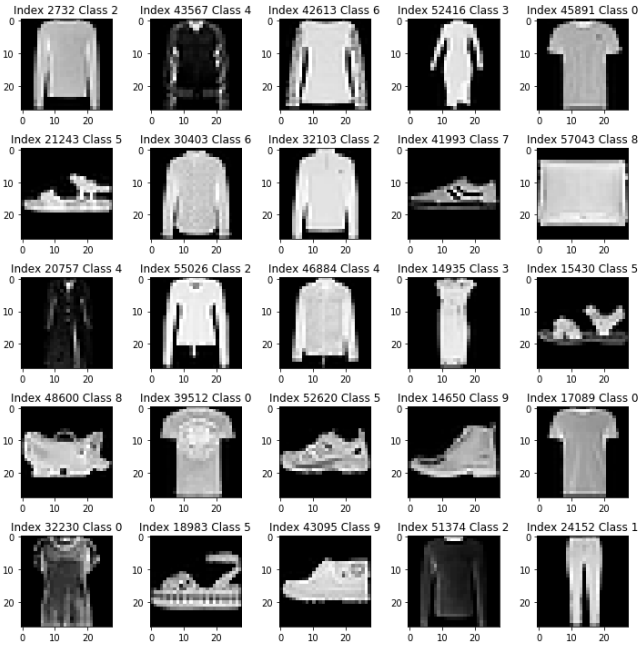
1. Extracting the features values and labels from the dataset: Fashion MNIST is processed into Numpy array using the mnist reader.
2. Data Partitioning into Training, Test and Validation set: The Fashion MNIST dataset is originally partitioned into Training and Test data each consisting of feature values and labels of count 60,000 and 10,000. The training set is further classified into **** and ***** of validation data.
3. Single Layer Neural Network:
    a. We train our model using the single layer neural network that's uses the gradient descent method
    b. The model is run on the validation set to tune the hyperparameters for better accuracy
4. Multi Layer Neural Network:

a. We train our model using the multi layer neural network with 3 and 6 layers and compare the accuracies.
b. The model is run on the validation set to tune the hyperparameters for better accuracy.

5. Convolutional Neural Network:
   a. We train our model using the convolutional neural network with 3 and 6 layers and compare the accuracies.
   b. The model is run on the validation set to tune the hyperparameters for better accuracy.

Dataset

The dataset used for training and testing the classifiers is the Fashion-MNIST dataset. It consists of 60,000 examples and testing set of 10,000 examples. Each image is a 28x28 grayscale image consisting of 28 pixels in height and 28 pixels in width thus resulting in a total of 784 pixels, associated with a label from 10 classes. The training and the test dataset have 785 columns and first columns consisting of class labels. The rest of the columns consists of the pixel-values of the associated image.



Below is the size of the training, test and validation sets.

```
print("X_train shape: " + str(X_train.shape))
print("y_train shape: " + str(y_train.shape))
print("X_val shape: " + str(X_val.shape))
print("y_val shape: " + str(y_val.shape))
print("X_test shape: " + str(X_test.shape))
print("y_test shape: " + str(y_test.shape))
print ("Number of training samples: m_train = " + str(m_train))
print ("Number of validation samples: m_validation = " + str(m_validation))
print ("Number of testing samples: m_test = " + str(m_test))
```

```
X_train shape: (59000, 784)
y_train shape: (59000,)
X_val shape: (1000, 784)
y_val shape: (1000,)
X_test shape: (10000, 784)
y_test shape: (10000,)
Number of training samples: m_train = 59000
Number of validation samples: m_validation = 1000
Number of testing samples: m_test = 10000
```

Preprocessing

Task 1: Single Layer Neural Network

1. Data Load and partitioning of the data into training, test and validation set: A brief description of the dataset is given in the dataset section of the report.
   Normalization:
   One of the steps in data preparation for neural network is the data normalization. Normalizing the input of the model helps in improving the convergence properties of the model. Applying normalization to dataset changes the values of numeric columns in the dataset to a common scale without affecting the differences in the range of values.
   Flattening the Images
   The Fashion MNIST dataset consists of images of size (60000,28,28), it is required that we transform the feature matrix of the image to size (60000,784)

```
# Flatten the images.
X_train = X_train.reshape((-1, 784))
X_test = X_test.reshape((-1, 784))

print(X_train.shape) # (60000, 784)
print(X_test.shape)  # (10000, 784)
m_train = X_train.shape[0]
m_test = X_test.shape[0]
```

```
(60000, 28, 28)
(60000, 784)
(10000, 784)
```

2. Training the Neural Network: It involves 4 steps
   a. **Build the architecture**
      We start with designing the Single Layer Neural Network by deciding the number of neurons(hidden_size) and activation functions. We start with initializing the weight and the bias terms with random values and the activation function used in this model is Relu. The Neural Network has an input dimension, a hidden layer dimension, and performs classification over classes. We train the network with a softmax loss function and L2 regularization on the weight matrices. The network uses a ReLU nonlinearity after the first fully connected layer.
      In other words, the network has the following architecture:
      input - fully connected layer - ReLU - fully connected layer – softmax .The outputs of the second fully-connected layer are the scores for each class.

      ReLu: $f(x) = max(x, 0)$

      Softmax: $$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

   b. **Compile the model**
      We calculate the loss and gradient for the Neural Network.
      Loss function: It calculates the difference between the output and the target variable. Cross-entropy is the default loss function used for a multi-class classification problem.
      Optimizer: It is used to determine how the model is updated and is based on the data and the loss function. The stochastic gradient descent is most common technique used in the Neural Network.
      Metrics: Accuracy is a common metric and it measures the fraction of images that are classified correctly.
   c. **Train the model**
      Now, we train the model by fitting it to the training data, so we input images and expected output labels into the model. We tune the hyperparameters like number of epochs, learning rate, hidden size(neurons) by running the model on validation set. This helps us in better defining the model and also minimize overfitting of the data. Once the hyperparameters are tuned the model is trained on the Training set.

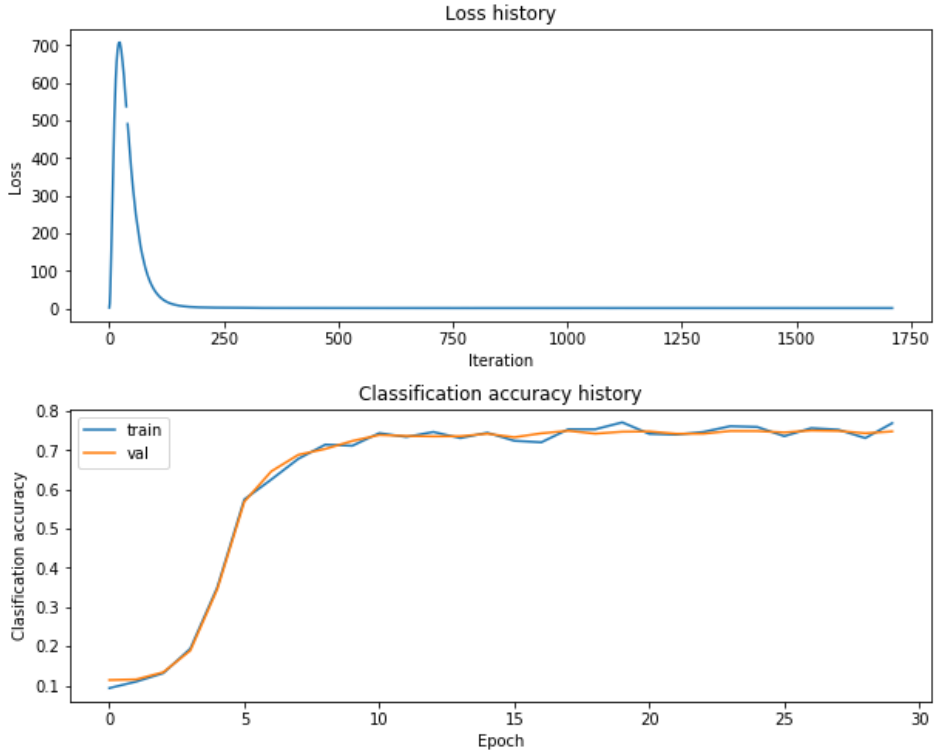
d. Evaluating the model

Once the model is trained, we evaluate its performance. This is done on the test set which the models hasn't been trained on yet. We evaluate the model based on the loss and accuracy.

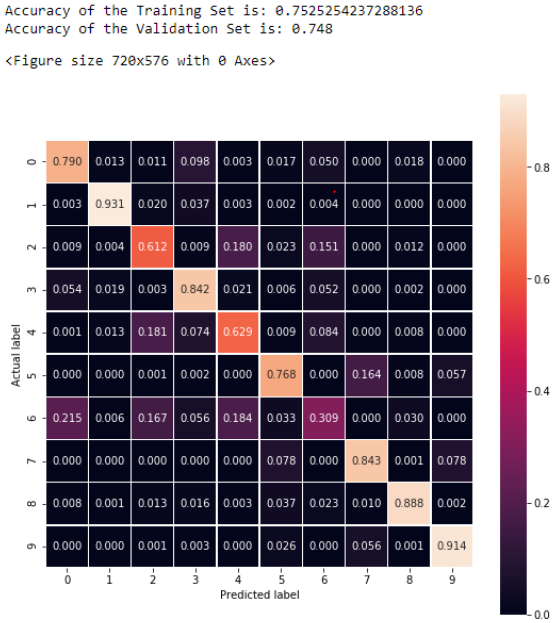
Results:

Below are the graphs for a. Loss vs Number of Iterations on Test Data Set

b. Accuracy vs Epoch on both Training and Validation Sets



Confusion Matrix on Training Set and Accuracies for Single Layer Neural Network



Task 2: Multi Layer Neural Network

1. Data Load and partitioning of the data into training, test and validation set, Normalizing the data and Flattening of images is same as that of Single Layer Neural Network.
2. Training the Neural Network: It involves 4 steps
   a. **Build the architecture**

   We start with building a simple 3-layer Neural Network. In the first layer the data is flatted from 28x28 to 784. The second layer is dense layer that consist of the ReLu activation function and has 128 neurons. The last layer is a dense layer that consists of softmax activation function which classifies the 10 categories of data and consists of around 10 neurons.

```
# Model a simple 3-layer neural network
model_3 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
model_3.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_10 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_26 (Dense)             (None, 128)               100480
_________________________________________________________________
dense_27 (Dense)             (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

   We have also built a 6-layer Neural Network. Below is the summary of the model.

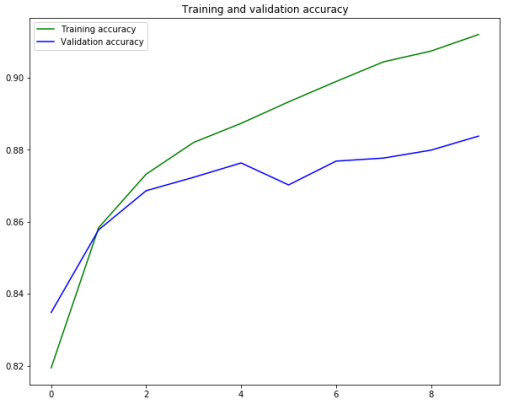
```
# Model a simple 6-layer neural network
model_6 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
model_6.summary()
model_6.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_11 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_28 (Dense)             (None, 128)               100480
_________________________________________________________________
dense_29 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_30 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_31 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_32 (Dense)             (None, 10)                1290
=================================================================
Total params: 151,306
Trainable params: 151,306
Non-trainable params: 0
_________________________________________________________________
```

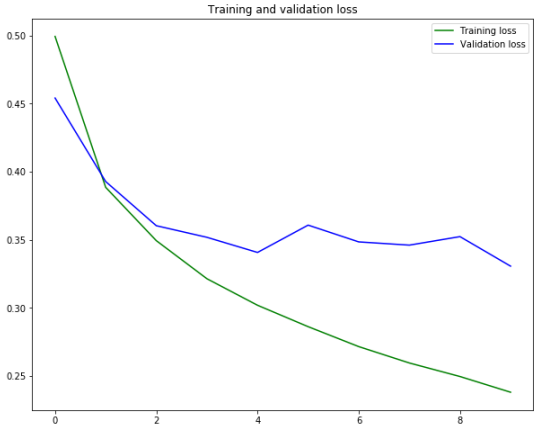
   b. Model is Compiled
   c. Train the Model
   d. Evaluate the Model

Results:

Graph for Training vs Validation Accuracy for 3- Layer NN



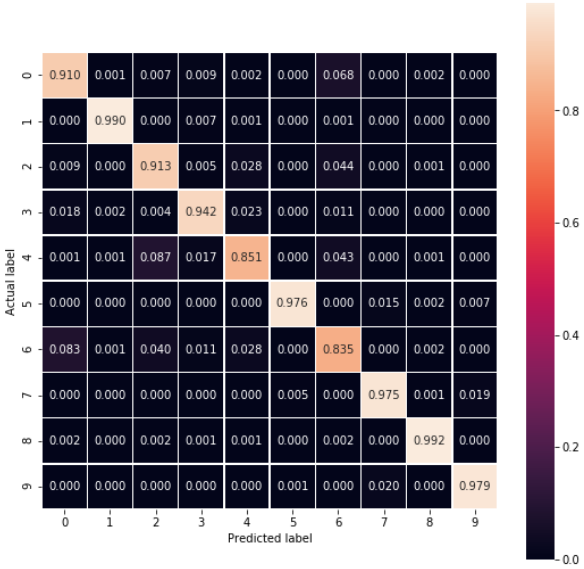
Graph for Training vs Validation Loss for 3 Layer NN



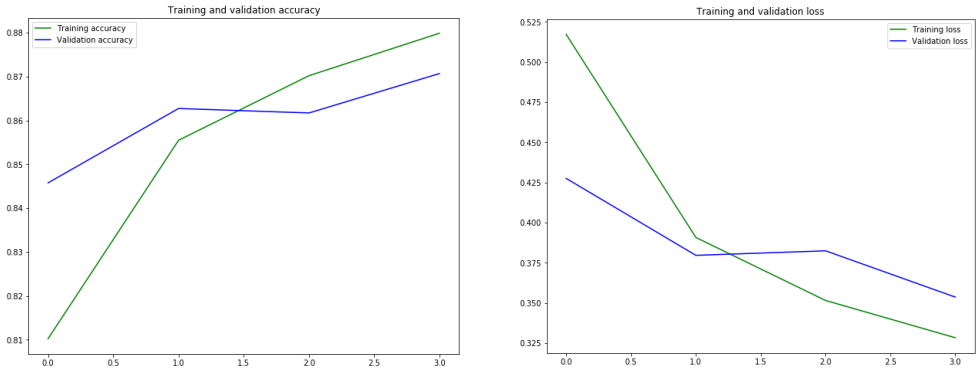
Confusion Matrix on Training Set and Accuracies for Multilayer 3- Layer Neural Network

```
Accuracy of the Training Set is: 0.936322033898305
Accuracy of the Validation Set is: 0.893

<Figure size 720x576 with 0 Axes>
```



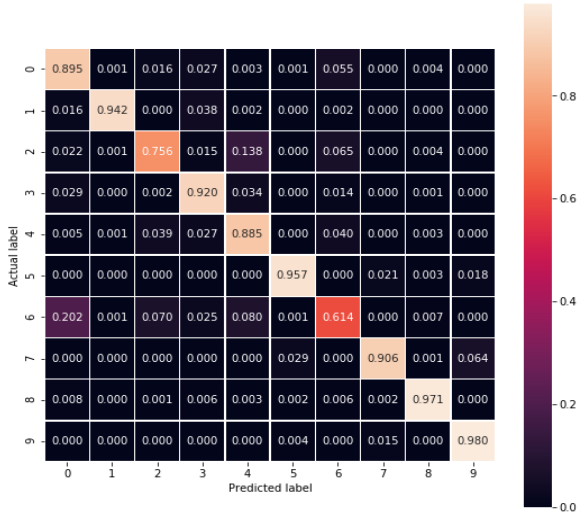
Graph for Training vs Validation Accuracy and Training vs Validation loss for 6- Layer NN



Confusion Matrix on Training Set and Accuracies for Multilayer 6- Layer Neural Network

```
Accuracy of the Training Set is: 0.882406779661017
Accuracy of the Validation Set is: 0.865

<Figure size 720x576 with 0 Axes>
```



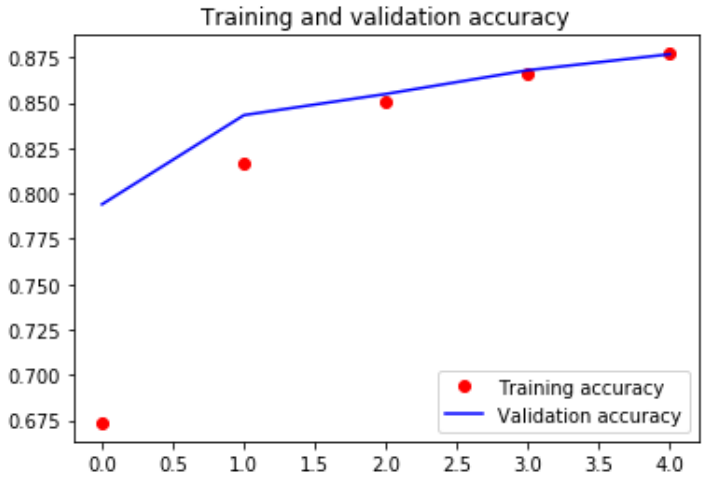
Task 3: Convolutional Neural Network:

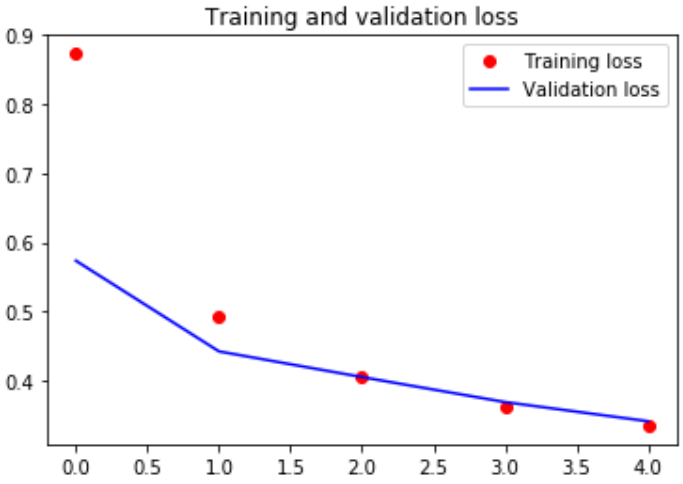
1. Data Load and partitioning of the data into training, test and validation set, Normalizing the data and Flattening of images is same as that of Single Layer Neural Network.
   One additional step in preprocessing the data in CNN is to 'one-hot-encode' the target variable.
2. Training the Neural Network: It involves 4 steps
   a. **Build the architecture**
      The type of model used in CNN is Sequential in Keras. It allows us to build the model by adding layers by layers using the add(). The first three layers are the Conv2Dlayers. We use a kernel of size 3x3 filter matrix.

```
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=(28,28,1),padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

b. Model is Compiled: Model is to be compiled before training it.
c. Train the Model: Model is trained using the fit() function. The hyper parameters are tuned on the validation set to better define our model and also to minimize the overfitting on the model.
d. Evaluate the Model
   Predictions on X_test to evaluate the model for accuracy.

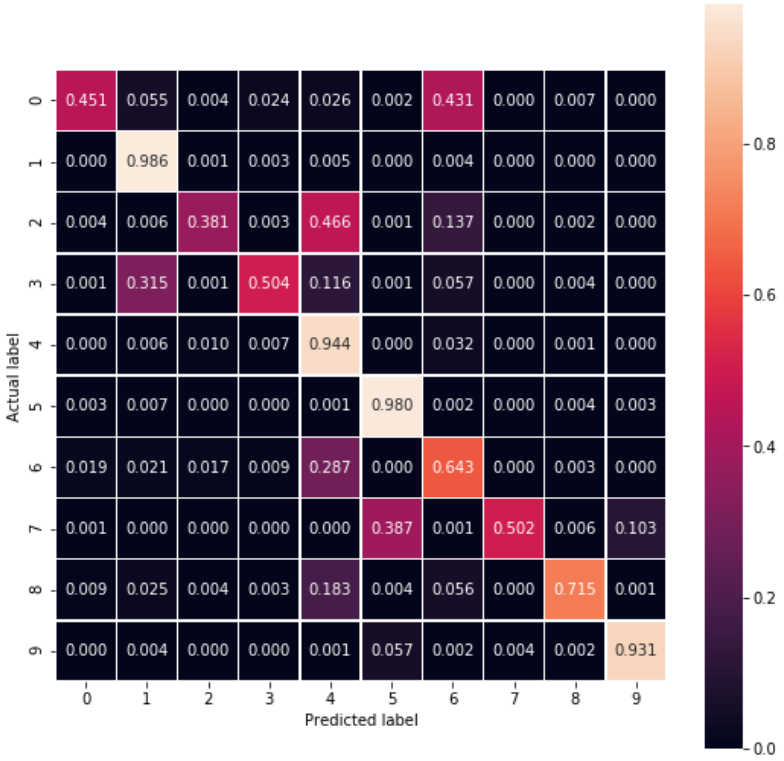
Results:





```
Accuracy of the Training Set is: 0.7036666666666667

<Figure size 432x288 with 0 Axes>
```



Conclusion

Neural Networks have the ability to learn and model non-linear and complex relationships. It learns from the initial inputs and their relationships, and can infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data. They can recognize patterns and predictions; and model highly volatile data and variances needed to predict rare events. In our problem we hav used the Neural Network to solve the problem of Classification with multiple classes. We have trained the dataset on Single Layer Neural Network, Multi Layer namely 3 and 6 and Convolutional Neural Network.

References:

https://medium.com/@ipylypenko/exploring-neural-networks-with-fashion-mnist-b0a8214b7b7b
https://www.kaggle.com/scratchpad/kernel765689136f/edit
https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python
https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5