Neural network and Convolutional Neural Network on Fashion-MNIST

Meghana Vasudeva
Person#: 50290586
University at Buffalo,
The State University of New York Buffalo,
New York 14260
mvasudev@buffalo.edu

Abstract

In this project I have performed the task of classification using neural network and convolutional neural network. The task is to identify an image and to classify it as one of the 10 classes. The dataset to be worked on was the Fashion-MNIST which is that of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. The task was divided into 3 parts.

- To build a Neural Network with one hidden layer to be trained and tested on Fashion-MNIST dataset.
- To build multi-layer Neural Network with open-source neural-network library, Keras on Fashion-MNIST dataset.
- To build Convolutional Neural Network (CNN) with open-source neural-network library, Keras on Fashion-MNIST dataset.

1. Introduction

Neural networks are one among the amazing programming paradigms. They are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. Normally when we program, we will tell the computer what to do, break down huge problems into smaller problems and specify tasks to solve each of these smaller problems. But in case of neural networks we will not tell the compute how to solve the problem. Instead, it will learn from the data it observes hence figuring out the solution on its own. Neural networks interpret sensory data through some sort of machine perception, which will label or cluster raw input. The patterns it recognizes are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. Neural networks are used to cluster and classify data. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on.

Recognizing patterns and images is not easy but we humans can capture what we see so easily. All of that work is done unknowingly and without our conscious. And we will never understand how hard it is to solve such a visual problem. When we try to write a program to visualize an image we will understand that the task is extremely difficult. Writing algorithms which describe the image might take us to achieve our goal but it will surely not be precise. Neural network's approach to this problem is quite different. The basic idea behind it is to take a large number of training set of data and to develop a system which reads that data and learns from it. This way the network is learning the dataset by inferring the given dataset. Hence if we increase the number of the dataset the network learns better and predicts better on the real data in turn increasing the accuracy of the output. In this project we will be building a neural network with one hidden layer, multilayer and using CNN.

2. Dataset

Fashion MNIST Clothing Classification

The Fashion-MNIST dataset is proposed as a more challenging replacement dataset for the MNIST dataset. It is a dataset comprised of training set of 60,000 examples and a test set of 10,000 examples of small square 28×28-pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total.

Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. Fashion-MNIST is intended to serve as a direct drop-in replacement for the original MNIST dataset to benchmark machine learning algorithms, as it shares the same image size and the structure of training and testing splits. Each training and test example is assigned to one of the following labels with example images in the dataset:

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

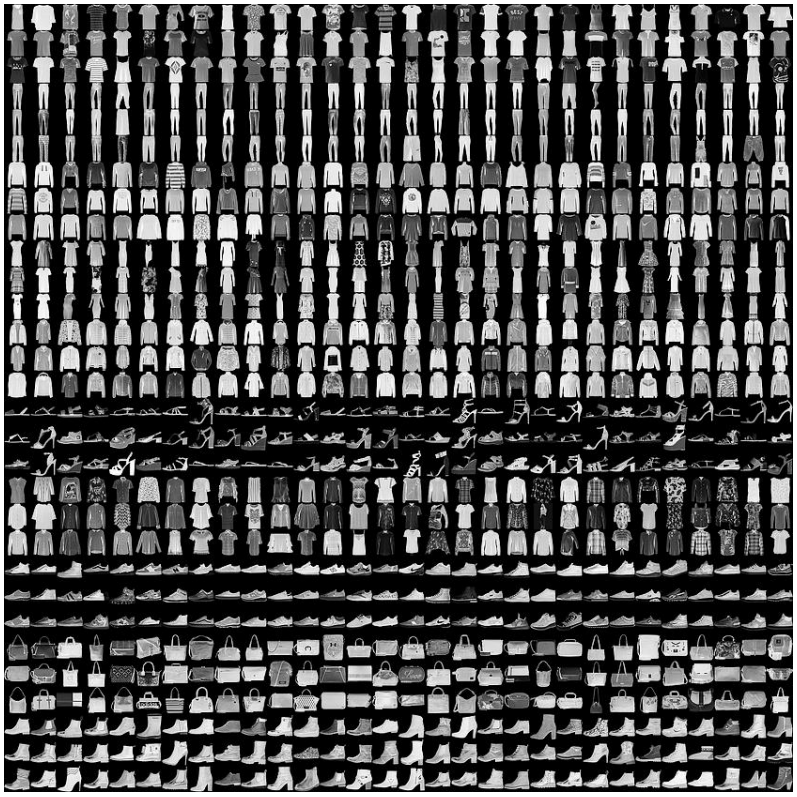


Fig: Example images in Fashion-MNIST data

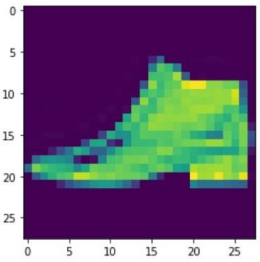
Below are the files contained in the Fashion-MNIST dataset.

Name	Description	Number of Examples	Size
train-images-idx3-ubyte.gz	Training set images	60, 000	25 MBytes
train-labels-idx1-ubyte.gz	Training set labels	60, 000	140 Bytes
t10k-images-idx3-ubyte.gz	Test set images	10, 000	4.2 MBytes
t10k-labels-idx1-ubyte.gz	Test set labels	10, 000	92 Bytes

Let us have a look at one instance (an article image), say at index 2, of the training dataset. So, we see that image at index (instance no.) 2 is an Ankle boot.

```
In [151]: plt.imshow(X_train[2, :].reshape((28, 28)))

Out[151]: <matplotlib.image.AxesImage at 0x22a447fb548>
```



3. Setting up Data and Preprocessing

Importing the libraries for Task 1,2,3. The first task was implemented completely on python from scratch and the second and third task used Keras built on tensorflow.

```
# importing Libraries for task 1.
import os # processing file path
import util_mnist_reader
import gzip # unzip the .gz file, not used here
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
```

```
#Library Import for task 2 and 3.
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Lambda
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import pandas as pd
import util_mnist_reader
np.random.seed(12345)
%matplotlib inline
```

Load the data and split the data into training and testing set of data. Then we normalize the data dimensions so that they are of the same scale.

```
# Read Fashion MNIST dataset

X_train, y_train = util_mnist_reader.load_mnist('fashion', kind='train')
X_test, y_test = util_mnist_reader.load_mnist('fashion', kind='t10k')
```

4. Implementation

Task 1: Single Layer Neural Network

This task was completely coded in python from scratch. These are the following steps to be performed for a single layer neural network.

- Initializing the Model by initializing the weights and biases. The biases are initialized to 0 and the weights to small random value.

```
def __init__(self, input_size, hidden_size, output_size, std=1e-4):
    """
    Initialize the model. Weights are initialized to small random values and
    biases are initialized to zero.
    """
    self.params = {}
    self.params['W1'] = std * np.random.randn(input_size, hidden_size)
    self.params['b1'] = np.zeros((1, hidden_size))
    self.params['W2'] = std * np.random.randn(hidden_size, output_size)
    self.params['b2'] = np.zeros((1, output_size))
```

- Compute the loss function by defining the forward and backward pass.
- Train the neural network using stochastic gradient descent to optimize the parameters. Create a mini batch of training set and compute loss and gradients on this mini batch. Next update the parameters. Finally, for every epoch calculate the accuracy for training and validation set. Predict and print the values.
- Activation functions are used at the end of a hidden unit is ReLU. The code snippet for the same is given below.

```
def loss(self, X, y=None, reg=0.0):
    """
    Compute the loss and gradients for a single layer neural
    network.

    """
    # Unpack variables from the params dictionary
    W1, b1 = self.params['W1'], self.params['b1']
    W2, b2 = self.params['W2'], self.params['b2']
    N, D = X.shape

    # Compute the forward pass
    scores = None
    h1 = ReLU(np.dot(X, W1) + b1)
    out = np.dot(h1, W2) + b2
    scores = out

    # If the targets are not given then jump out, we're done
    if y is None:
        return scores

    # Compute the loss
    scores_max = np.max(scores, axis=1, keepdims=True)    # (N,1)
    exp_scores = np.exp(scores - scores_max)              # (N,C)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)    # (N,C)
    correct_logprobs = -np.log(probs[range(N), y])        # (N,1)
    data_loss = np.sum(correct_logprobs) / N
    reg_loss = 0.5 * reg * np.sum(W1*W1) + 0.5 * reg * np.sum(W2*W2)
    loss = data_loss + reg_loss

    # Backward pass: compute gradients
    grads = {}
    dscores = probs                                       # (N,C)
    dscores[range(N), y] -= 1
    dscores /= N
    dW2 = np.dot(h1.T, dscores)                           # (H,C)
    db2 = np.sum(dscores, axis=0, keepdims=True)          # (1,C)
    dh1 = np.dot(dscores, W2.T)                           # (N,H)
    dh1[h1 <= 0] = 0
    dW1 = np.dot(X.T, dh1)                                # (D,H)
    db1 = np.sum(dh1, axis=0, keepdims=True)              # (1,H)
    dW2 += reg * W2
    dW1 += reg * W1

    grads['W1'] = dW1
    grads['b1'] = db1
    grads['W2'] = dW2
    grads['b2'] = db2

    return loss, grads
```

```
num_train = X.shape[0]
iterations_per_epoch = max(int(num_train / batch_size), 1)

# Use SGD to optimize the parameters in self.model
v_W2, v_b2 = 0.0, 0.0
v_W1, v_b1 = 0.0, 0.0
loss_history = []
train_acc_history = []
val_acc_history = []

for it in range(1, num_epochs * iterations_per_epoch + 1):
    X_batch = None
    y_batch = None

    # Create a random minibatch of training data and labels
    sample_index = np.random.choice(num_train, batch_size, replace=True)
    X_batch = X[sample_index, :]
    y_batch = y[sample_index]

    # Compute loss and gradients using the current minibatch
    loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
    loss_history.append(loss)

    # Use the gradients to update the parameters of the network
    v_W2 = mu * v_W2 - learning_rate * grads['W2']
    self.params['W2'] += v_W2
    v_b2 = mu * v_b2 - learning_rate * grads['b2']
    self.params['b2'] += v_b2
    v_W1 = mu * v_W1 - learning_rate * grads['W1']
    self.params['W1'] += v_W1
    v_b1 = mu * v_b1 - learning_rate * grads['b1']
    self.params['b1'] += v_b1

    if verbose and it % iterations_per_epoch == 0:
    # Every epoch, check train and val accuracy and decay learning rate.
        epoch = it / iterations_per_epoch
        train_acc = (self.predict(X_batch) == y_batch).mean()
        val_acc = (self.predict(X_val) == y_val).mean()
        train_acc_history.append(train_acc)
        val_acc_history.append(val_acc)
        print("epoch %d / %d: loss %f, train_acc: %f, val_acc: %f" %
                    (epoch, num_epochs, loss, train_acc, val_acc))

        # Decay learning rate
        learning_rate *= learning_rate_decay
        mu *= mu_increase

return {
    'loss_history': loss_history,
    'train_acc_history': train_acc_history,
    'val_acc_history': val_acc_history,
```

Task 2: Multi-Layer Neural Network using Keras

Training a Neural Network (NN) requires 4 steps:

- Build the architecture

The first layer is the ReLU activation function and has 512 neurons. Second has 128 neurons. The last layer is dense layer with a softmax activation function that classifies the 10 categories of data and has 10 neurons. Model.summary will visualizes the network architecture.

- Compile the model
  Loss function — This calculates the difference between the output and the target variable In this example, we chose the *sparse_categorical_crossentropy* loss function.
  Optimizer — This tells how the model is updated and is based on the data and the loss function. *Adam* is an extension to the classic stochastic gradient descent**.**
  Metrics — monitors the training and testing steps. *Accuracy* is a common metric and it measures the fraction of images that are correctly classified
- Train the model
- Evaluate the model

```
model = Sequential([
    Dense(512, input_shape=(784,), activation='relu'),
    Dense(128, activation = 'relu'),
    Dense(10, activation='softmax')
])
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_11 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer=Adam(lr=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=50,
                    verbose=1,
                    validation_data=(X_val, y_val))
```

Task 2: CNN using Keras

I have created a 3-layer CN network. There are 3 major parts to CNN. The convolutional layer, max pooling layer and the Dense layer. All layers will use the ReLU activation function and the He weight initialization scheme. Similar to the multi-layer model the compilation steps are taken place.

```
name = '1-layer'
cnn_model_1 = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape,name='Conv2D'),
    MaxPooling2D(pool_size=(2, 2),name='MaxPool'),
    Dropout(0.2,name='Dropout'),
    Flatten(name='flatten'),
    Dense(32, activation='relu', name='Dense'),
    Dense(10, activation='softmax',name='Output')
], name=name)

name = '2-layer'
cnn_model_2 = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape,name='Conv2D-1'),
    MaxPooling2D(pool_size=(2, 2),name='MaxPool'),
    Dropout(0.2,name='Dropout-1'),
    Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=input_shape,name='Conv2D-2'),
    Dropout(0.25,name='Dropout-2'),
    Flatten(name='flatten'),
    Dense(64, activation='relu', name='Dense'),
    Dense(10, activation='softmax',name='Output')
], name = name)

name = '3-layer'
cnn_model_3 = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape,kernel_initializer='he_normal', name='Conv2D-1'
    MaxPooling2D(pool_size=(2, 2),name='MaxPool'),
    Dropout(0.25,name='Dropout-1'),
    Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=input_shape,name='Conv2D-2'),
    Dropout(0.25,name='Dropout-2'),
    Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=input_shape,name='Conv2D-3'),
    Dropout(0.4,name='Dropout-3'),
    Flatten(name='flatten'),
    Dense(128, activation='relu', name='Dense'),
    Dropout(0.4,name='Dropout'),
    Dense(10, activation='softmax',name='Output')
], name = name)

cnn_models = [cnn_model_1,cnn_model_2,cnn_model_3]
```

```
history_dict_cnn = {}

for model in cnn_models:
    model.compile(
        loss = 'sparse_categorical_crossentropy',
        optimizer = Adam(),
        metrics=['accuracy']
    )

    history_cnn = model.fit(
        X_train_cnn,y_train_cnn,
        batch_size = batch_size,
        epochs=50,
        verbose=1,
        validation_data=(X_val_cnn,y_val_cnn)
    )

    history_dict_cnn[model.name] = history_cnn
```

5. Architecture

Single Layer and Multi-Layer Neural Network:

The most common structure of connecting neurons into a network is by layers. The nodes on the left are called the input layers. The rightmost layer is the output layer. The middle layer is the hidden layer. The meaning of hidden layer is that it is neither an input layer nor an output layer. So, the design of the layers is simple. Similarly, a neural network can have multiple hidden layers which we term as multi-layer neural network.

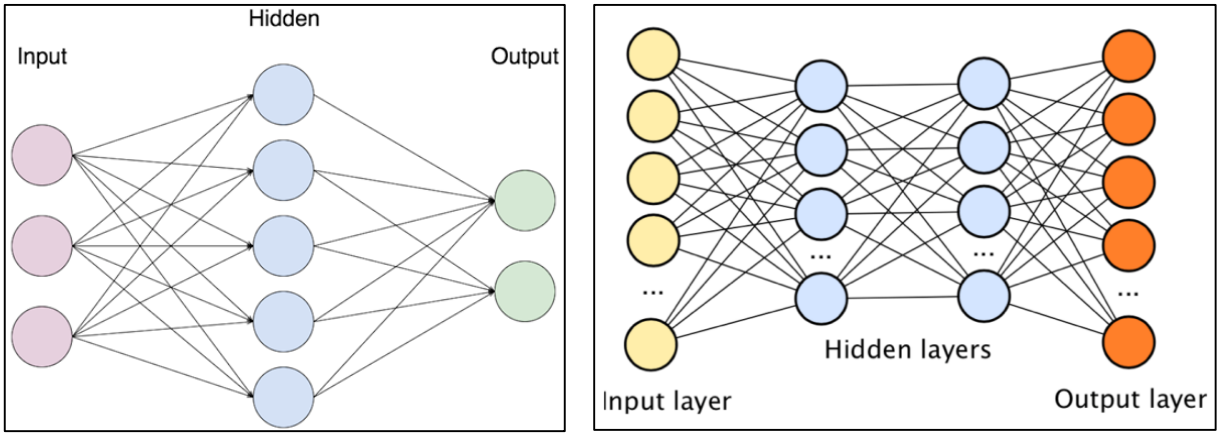
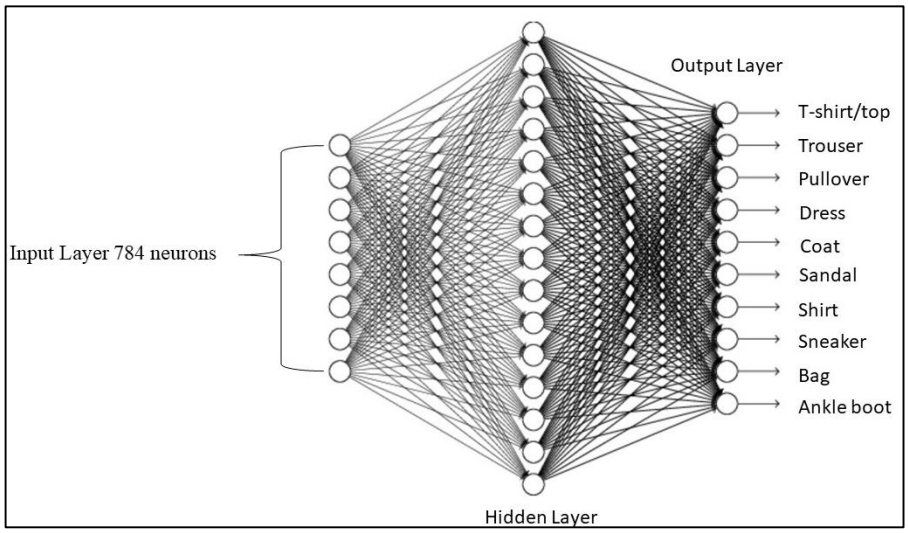


Fig: Single and Multi-Layer Neural Network (images from google)

Now we want to determine if the image is a shirt or now then we will encode the intensity of the image into input neurons. The image is a greyscale image of size 28x28 pixels then we have 784 input neurons with a value of 0.0 representing white, a value of 1.0 representing black, and in between values have shades of grey. The second layer of the network is a hidden layer and we experiment with different values for n. The output layer will contain 10 neurons which indicate one for each of our type of clothing. So if the label is 0 then the clothing will represent T-shirt/top.



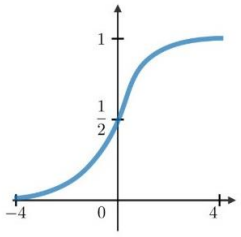
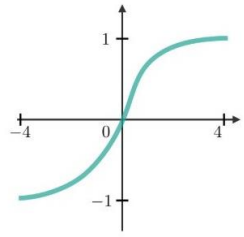
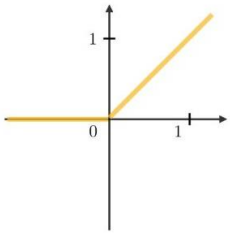
Equations and Expressions of neural networks:

Linear models for regression and classification can be represented as shown below which are linear combinations of basis functions. In a neural network the basis functions depend on parameters. During training we will allow these parameters to be adjusted along with the coefficients w_j.

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}) \right)$$

Activation Function f:

Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model.

Sigmoid	Tanh	ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

Cross Entropy Error Function:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right\}$$

We use linear outputs and a sum-of-squares error. For a multiple independent binary classification, we use logistic sigmoid outputs and a cross-entropy error function. In case of a multiclass classification, softmax outputs with corresponding multiclass cross-entropy error is used. Finally, for a classification problems involving only two classes, we can use a single logistic sigmoid output, or alternatively we can use a network with two outputs having a softmax output activation function.

Convolutional Neural Network:

Convolutional neural network is most commonly used to visualize an image. Convolution is a special kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. CNNs apply a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model can then use for classification. CNNs contains three components:

- Convolutional layers, which apply a specified number of convolution filters to the image. For each sub region, the layer performs a set of mathematical operations to produce a single value in the output feature map. Convolutional layers then typically apply a ReLU activation function to the output to introduce nonlinearities into the model.
- Pooling layers, which down-sample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. A commonly used pooling algorithm is max pooling, which extracts sub regions of the feature map (e.g., 2x2-pixel tiles), keeps their maximum value, and discards all other values.

- Dense layers, which perform classification on the features extracted by the convolutional layers and down-sampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

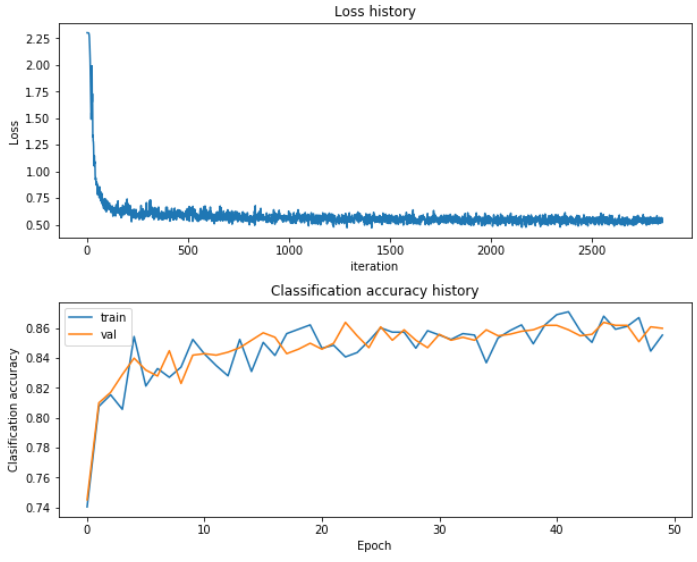
6. Analysis and Results

Task 1: Single Layer Neural Network

After tuning the parameters this is the final values I have used to get the best accuracy.
Parameters:
Number of epochs=**50**,
batch_size=**1024**,
learning_rate= **0.00075**,
learning_rate_decay=**0.95**,
reg=**1.0**,
verbose=**True**

```
epoch 30 / 50: loss 0.537970, train_acc: 0.858398, val_acc: 0.847000
epoch 31 / 50: loss 0.534268, train_acc: 0.855469, val_acc: 0.856000
epoch 32 / 50: loss 0.596191, train_acc: 0.852539, val_acc: 0.852000
epoch 33 / 50: loss 0.563458, train_acc: 0.856445, val_acc: 0.854000
epoch 34 / 50: loss 0.531306, train_acc: 0.855469, val_acc: 0.852000
epoch 35 / 50: loss 0.576230, train_acc: 0.836914, val_acc: 0.859000
epoch 36 / 50: loss 0.531799, train_acc: 0.853516, val_acc: 0.855000
epoch 37 / 50: loss 0.534296, train_acc: 0.858398, val_acc: 0.856000
epoch 38 / 50: loss 0.536905, train_acc: 0.862305, val_acc: 0.858000
epoch 39 / 50: loss 0.575171, train_acc: 0.849609, val_acc: 0.859000
epoch 40 / 50: loss 0.518943, train_acc: 0.862305, val_acc: 0.862000
epoch 41 / 50: loss 0.514905, train_acc: 0.869141, val_acc: 0.862000
epoch 42 / 50: loss 0.519241, train_acc: 0.871094, val_acc: 0.859000
epoch 43 / 50: loss 0.522600, train_acc: 0.858398, val_acc: 0.855000
epoch 44 / 50: loss 0.554469, train_acc: 0.850586, val_acc: 0.856000
epoch 45 / 50: loss 0.508249, train_acc: 0.868164, val_acc: 0.864000
epoch 46 / 50: loss 0.540576, train_acc: 0.859375, val_acc: 0.862000
epoch 47 / 50: loss 0.527925, train_acc: 0.861328, val_acc: 0.862000
epoch 48 / 50: loss 0.542674, train_acc: 0.867188, val_acc: 0.851000
epoch 49 / 50: loss 0.557939, train_acc: 0.844727, val_acc: 0.861000
epoch 50 / 50: loss 0.558309, train_acc: 0.855469, val_acc: 0.860000
```

The accuracy that I have got for a single layer neural network is 86% with a loss of 0.55. Below is the graph for loss vs number of iteration and accuracy vs epochs.



Task 2: Multi-Layer Neural Network

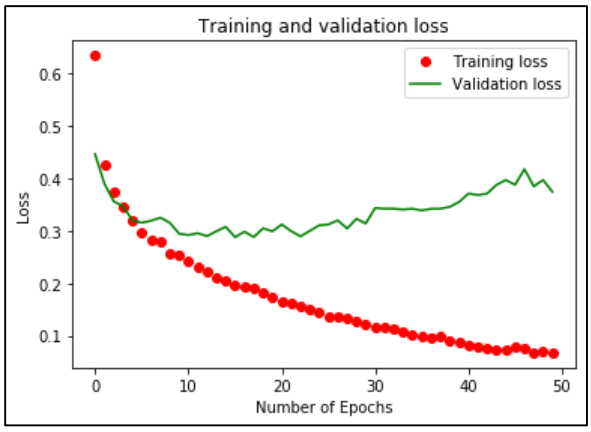
Parameters:
epochs=50,
verbose=1,
lr=0.001

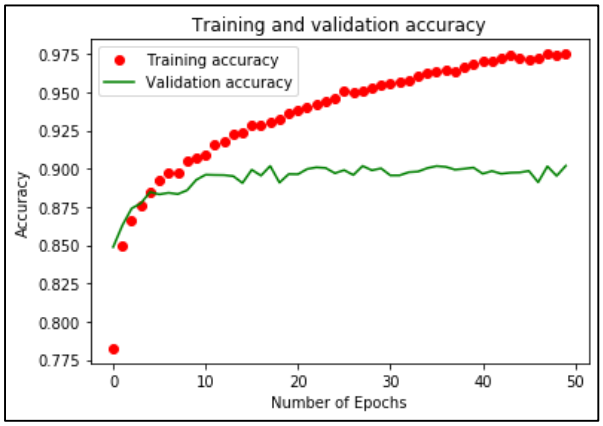
The accuracy obtained after tuning the parameters is 96% with loss 0.118

```
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

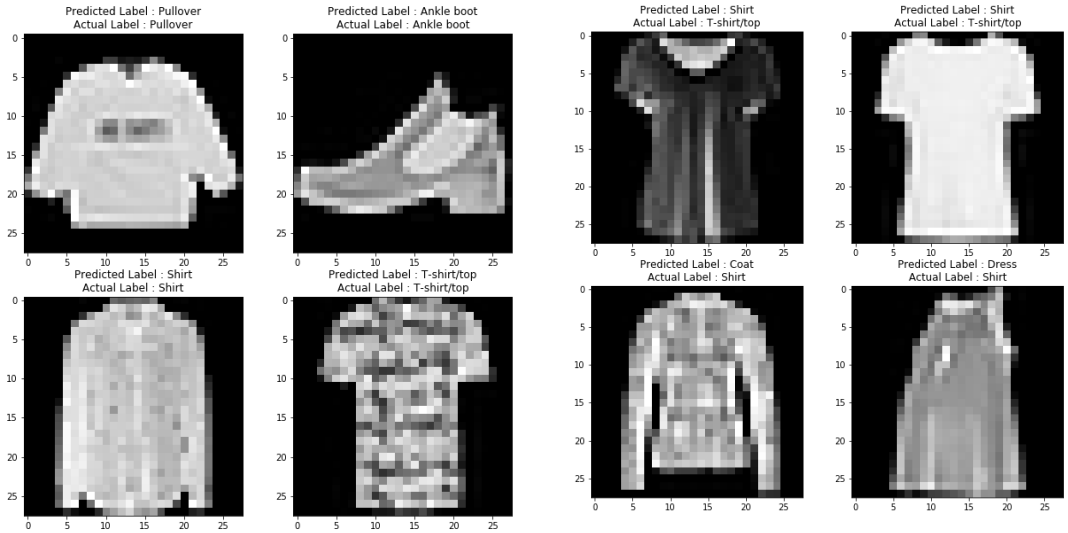
```
Test loss: 0.11822070961284141
Test accuracy: 0.9654333333333334
```

Below is the graph for the loss vs epoch and accuracy vs epoch for training and validation set. We can notice that as the accuracy increases the loss reduces.

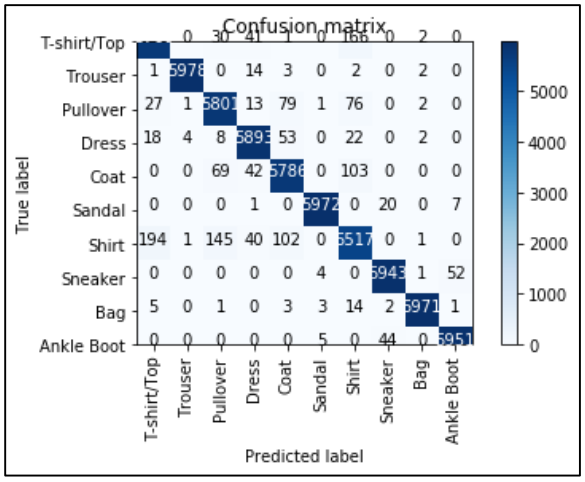




Below is the visualization of correctly predicted classes and incorrectly predicted classes.



Classification Report:
The classification report visualizer displays the precision, recall, F1, and support scores for the model.



	precision	recall	f1-score	support
T-shirt/Top	0.96	0.96	0.96	6000
Trouser	1.00	1.00	1.00	6000
Pullover	0.96	0.97	0.96	6000
Dress	0.98	0.98	0.98	6000
Coat	0.96	0.96	0.96	6000
Sandal	1.00	1.00	1.00	6000
Shirt	0.94	0.92	0.93	6000
Sneaker	0.99	0.99	0.99	6000
Bag	1.00	1.00	1.00	6000
Ankle Boot	0.99	0.99	0.99	6000
accuracy			0.98	60000
macro avg	0.98	0.98	0.98	60000
weighted avg	0.98	0.98	0.98	60000

We can see that some of the clothes were completely predicted correctly and we can notice that shirts and T-shirt/Top have a little lesser as they have been confused with each other. But then the results show that the classification is very accurate and successfully determined the clothes into different categories. The accuracy is 98% in this case.

Task 3: Convolutional Neural Network
I have used 3 layers in my CNN.
Parameters:
epochs=50,
verbose=1

Accuracy Layer1: 91.77%
Loss Layer1: 0.22

```
Epoch 50/50
48000/48000 [==============================] - 18s 370us/step - loss: 0.1502 - acc: 0.9458 - val_loss: 0.2393 - val_acc:
0.9177
```

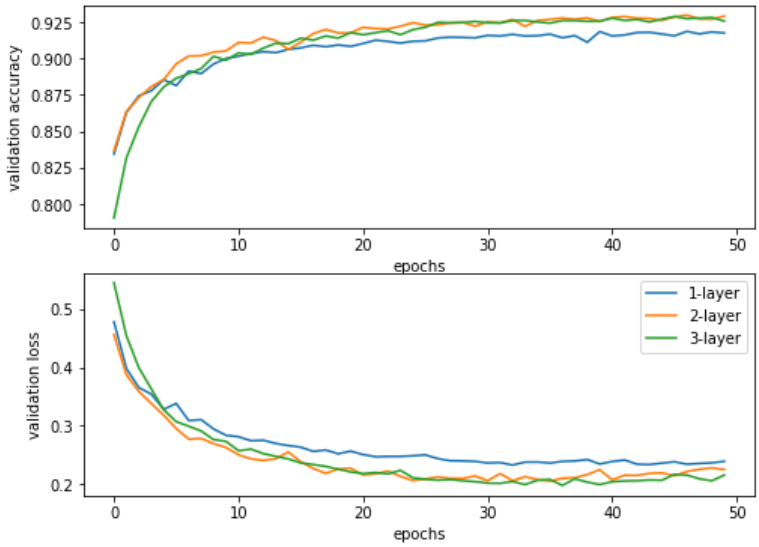
Accuracy Layer2: 92.9%
Loss Layer2: 0.225

```
Epoch 50/50
48000/48000 [==============================] - 45s 931us/step - loss: 0.0833 - acc: 0.9680 - val_loss: 0.2254 - val_acc:
0.9292
```

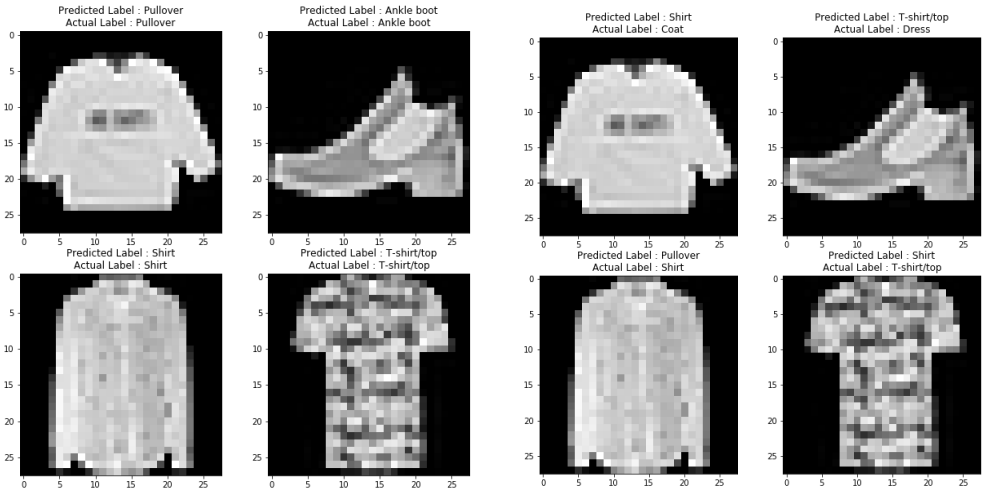
Accuracy Layer3: 92.5%
Loss Layer3: 0.149

```
Epoch 50/50
48000/48000 [==============================] - 63s 1ms/step - loss: 0.1492 - acc: 0.9417 - val_loss: 0.2159 - val_acc: 0.
9258
```

The graph of accuracy vs epoch and loss vs epoch is given below. We can see that the accuracy increased in each layer and loss got reduced.

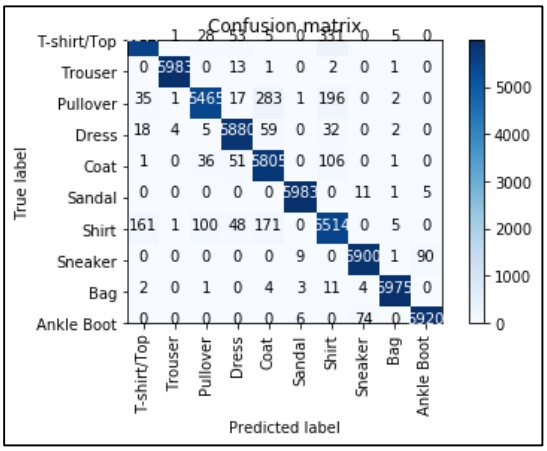


Below is the visualization of correctly predicted classes and incorrectly predicted classes.



Classification Report:

The classification report visualizer displays the precision, recall, F1, and support scores for the model



	precision	recall	f1-score	support
T-shirt/Top	0.96	0.93	0.95	6000
Trouser	1.00	1.00	1.00	6000
Pullover	0.97	0.91	0.94	6000
Dress	0.97	0.98	0.97	6000
Coat	0.92	0.97	0.94	6000
Sandal	1.00	1.00	1.00	6000
Shirt	0.89	0.92	0.90	6000
Sneaker	0.99	0.98	0.98	6000
Bag	1.00	1.00	1.00	6000
Ankle Boot	0.98	0.99	0.99	6000
accuracy			0.97	60000
macro avg	0.97	0.97	0.97	60000
weighted avg	0.97	0.97	0.97	60000

From the confusion matrix we can see that the precision of coat shirt and is less as these have been misclassified a few times as they are very similar. The rest have been classified pretty accurately and hence we have an accuracy of 97% in this case.

7. Conclusion

As we have seen from the results the single layer neural network has the least accuracy when compared to the other two yet it classified images to an accuracy of 86%. Multi-layer neural network and CNN were very accurate and have classified mostly all of the clothes correctly. Hence I can conclude by saying that neural network is one of the best algorithms for prediction and detection of images.

8. References

MNIST to CSV conversion : https://pjreddie.com/projects/mnist-in-csv/

Fashion MNIST dataset: https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/

Neural Network: https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning

Class Notes Neural Networks: https://ublearns.buffalo.edu/bbcswebdav/pid-5108173-dt-content-rid-25438091_1/courses/2199_23170_COMB/Chap5.2-Network%20Training.pdf