Fashion-MNIST Final Report

Objective: To classify images of different pieces of clothing.

Client & Data-Set: Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of ***60,000*** examples and a test set of ***10,000*** examples. Each example is a 28x28 grayscale image, associated with a label from ***10 classes***.

Data-set is publicly available on kaggle and Zalando Fashion MNIST repository on Github.

Fashion-MNIST is intended as direct drop-in replacement for the original MNIST dataset. It shares the same image size and structure of training and testing splits.

Business Impact: E-commerce companies have lots of items for sale online which requires lots of images to be displayed on their websites, applications and on social media. And it takes lot of human power and time to separate these images into respective groups. This classifier which we are going to build helps businesses to categorize images into respective groups.

Methodology: For this project we will be going to use deep learning concepts like artificial neural networks and convolutional neural networks to build an image classification model which will learn to distinguish 10 different item images into their respective categories.

Labels: Each training and test example is assigned to one of the following labels:

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

Data Wrangling

The Dataset contains 70,000 images with 60,000 for training and 10,000 for testing in 2 different files train.csv and test.csv respectively. In both the files 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. The training and test data sets have 785 columns. The first column consists of the class labels and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

- Each row is a separate image
- Column 1 is the class label.
- Remaining columns are pixel numbers (784 total).
- Each value is the darkness of the pixel (1 to 255)

After reading the data files we convert the data-frames to array and then split the pixel data and labels by slicing the arrays and rescale the pixel data between 0 and 1 which is a kind of data normalization technique.

Then we split the original training data into 80% training and 20% validation. This helps to see whether we're over-fitting on the training data and weather we should lower the learning rate and train for more epochs if validation accuracy is higher than training accuracy or stop over-training if training accuracy shift higher than the validation.

Image Reshaping: After importing the dataset our image arrays are of shape (60000, 785) in training set. So here we reshape the array into original image size which is (60000, 28, 28, 1).

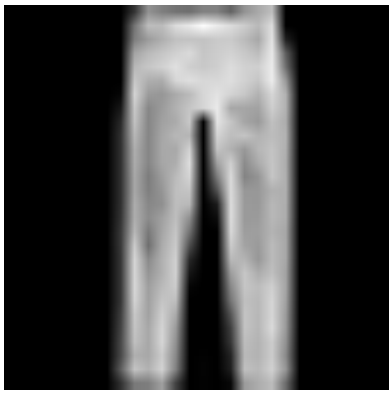
One Hot Encoding: We apply one-hot encoding to our class variables to convert them into a format that works better with machine learning algorithms.

Data Exploration

Our dataset has 10 different classes as follows:

0 - T-shirt/top

1 - Trouser



2 - Pullover

3 - Dress

4 - Coat

5 – Sandal

6- Shirt

7 - Sneaker

8 - Bag

9 - Ankle boot

Deep Learning with Jupyter Notebooks in Cloud

Here's are few brief steps on how one can run deep learning models in cloud.

While you can run deep learning on laptop or personal computer, you'll eventually find that you want to run deep learning models on a Graphical Processing Unit (GPU). Because I do not have one on my computer I choose to run one from the Amazon Web Services (AWS) Cloud.

There are quite a few steps to get your cloud computing environment set up the first time. But once you've set it up, you'll find it easy to keep using it in the future. The steps to get started are

- Getting an Amazon Web Services Account
- Setting Up Your Cloud Computing Server
- Connecting to Your Server
- Setting Up Your Jupyter Notebook
- Connecting to Jupyter in The Browser
- Using Your Notebook

Amazon offers an enormous range of cloud computing services, I have used their EC2 service (short for Elastic Cloud Compute). Each computing instance comes preloaded with different software called Amazon Machine Image (AMI). In my case, I want to use the one from the AWS Marketplace that is optimized for deep learning (*Deep Learning AMI Ubuntu Version)* which comes with all the necessary deep learning packages pre-installed. Then the final thing is to choose the instance type, each instance type has a different price, and different computational capabilities. I have used p2-xlarge instance which was fairly fast for this project. To give a quick overlook on how fast it was, while it used to take around 500 seconds on my computer it just took 6 seconds on cloud which was a huge time saver and allowed me to run more epochs and tune many hyper parameters starting from a scratch model.

Modelling

For building an image classifier here we use deep convolutional neural networks (CNN's). So, let's see what happens in each step in a CNN.

Layers in a CNN:

- Convolution Layer
- Pooling Layer
- Fully Connected Layer/Dense Layer

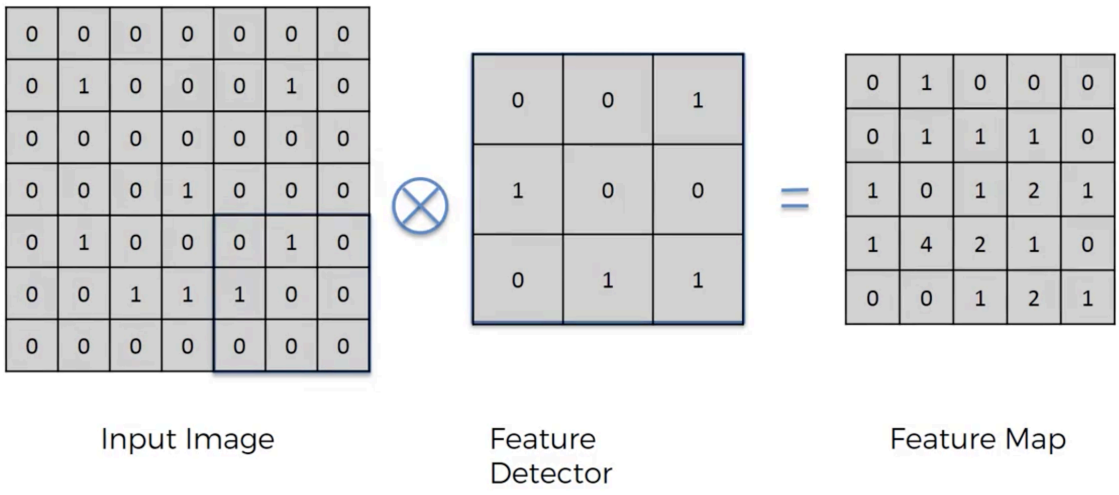
We are going to build a sequential model. The sequential model is a linear stack of layers and this is how we build a sequential model in python.

```
cnn_all = Sequential()
```

Once the sequential model is defined then we can add different layers required as explained below.

Convolution Layer: The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Every image can be considered as a matrix of pixel values. Consider a 7 x 7 image whose pixel values are only 0 and 1 (note that for a grayscale image, pixel values range from 0 to 255, the input matrix below is a special case where pixel values are only 0 and 1). Also, consider another 3 x 3 matrix as shown below. Then, the Convolution of the 7 x 7 image and the 3 x 3 matrix can be computed as shown.



Now let's understands how the computation happened above. We slide the 3 x 3 matrix over our original image (7 x 7) by 1 pixel (also called 'stride') and for every position, we compute element

wise multiplication (between the two matrices) and add the multiplication outputs to get the final integer which forms a single element of the output matrix (Feature Map). Note that the 3×3 matrix "sees" only a part of the input image in each stride.

In CNN terminology, the 3×3 matrix is called a '**filter**' or 'kernel' or '**feature detector**' and the matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the '**Feature Map**'. It is important to note that filters act as feature detectors from the original input image.

The size of the Feature Map is controlled by three parameters:

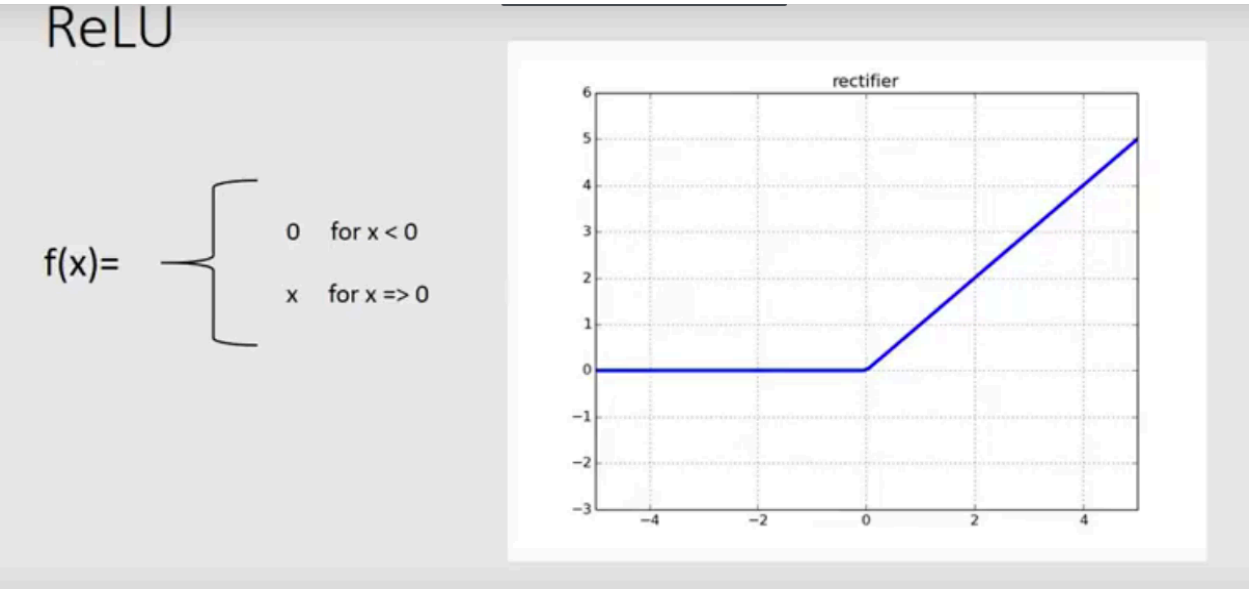
Depth: Depth corresponds to the number of filters we use for the convolution operation.

Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix.

Padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix.

Introducing Non-Linearity (ReLU):

An additional operation called ReLU has been used after every Convolution operation. ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:



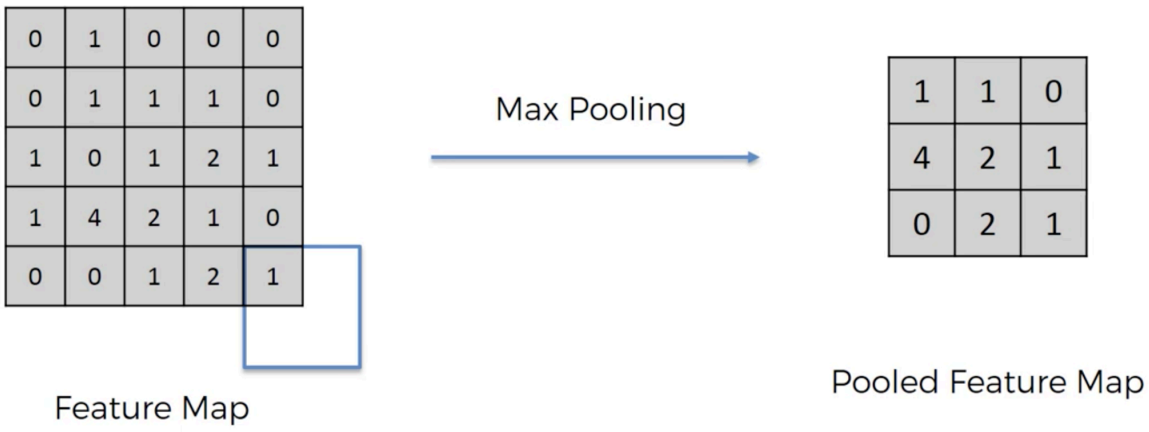
ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLu is to introduce non-linearity in CNN.

Here's how we add convolutional layers in a neural network including all the different parameters explained above.

```
cnn_all.add(Conv2D(64, 3, padding='same', activation='relu'))
```

Pooling Layer:

Pooling reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, we define a spatial neighborhood (a 2×2 window which we considered in this project) and take the largest element from the rectified feature map within that window.



The pooling layer, is used to reduce the spatial dimensions, but not depth, on a CNN, basically this is what you gain:

1. By having less spatial information you gain computation performance
2. Less spatial information also means less parameters, so less chance to over-fit
3. Makes the network invariant to small transformations, distortions and translations in the input image.

Here we add a Max-pooling layer with a pool-size = 2

```
cnn_all.add(MaxPooling2D(2))
```

Flattening: Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.

Before we give the input to fully connected layer we flatten the input as follows:

```
cnn_all.add(Flatten())
```

Fully Connected Layer/Dense Layer:

The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. Fully connected layer uses a ‘softmax’ activation function in the output layer.

SoftMax: The SoftMax activation function is useful predominantly in the output layer of a clustering system. Softmax functions convert a raw value into a posterior probability. The sum of output probabilities from the Fully Connected Layer is always 1. This provides a measure of certainty.

Now let's add a fully connected layer and this is how we do it.

```
cnn_all.add(Dense(256, activation='relu'))
```

And finally adding the output layer with 10 classes using SoftMax activation function.

```
cnn_all.add(Dense(10, activation='softmax'))
```

As explained above, the Convolution and Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier.

The overall training process of the Convolution Network is summarized as below:

- We initialize all filters and parameters / weights with random values.
- The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
- Calculate the total error at the output layer (summation over all 10 classes).
- Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error.

Once we build the sequential model now it's time to compile it. Arguments added in final compilation will control whole neural network. The choice of Optimization Algorithms and Loss Functions for a deep learning model can play a big role in producing optimum and faster results.

First argument is optimizer, this is nothing but how you want to find the optimal set of weights.

Adam: Adam stands for Adaptive Moment Estimation. It also calculates different learning rate. Adam is another method that computes adaptive learning rates for each parameter. Adam works well in practice, is faster, and outperforms other techniques.

If we go deeper in detail SGD (Adam here) depends on loss thus our second parameter is loss. Because we have 10 classes in our dependent variable we chose to use 'Categorical Cross-Entropy' as our loss function.

Error and Loss Function: In most learning networks, error is calculated as the difference between the actual output and the predicted output. The function that is used to compute this error is known as Loss Function.

Categorical Cross-Entropy: Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Once the initial loss function is calculated then we try to optimize our loss function. Optimization Algorithms are used to update weights and biases i.e. the internal parameters of a model to reduce the error. Here we used 'Adam' optimizer in our current project.

Evaluation Metric: Evaluation metrics explain the performance of a model. An important aspects of evaluation metrics is their capability to discriminate among model results. Because we have equal number of samples from each class initially we are using 'Accuracy' to evaluate the performance of the model. Also, we will plot confusion matrix and observe different metrics in our later analysis.

Till now we have seen how we chose different parameters in model compiling step. Let's see how we do it in python.

```
cnn_all.compile(optimizer=Adam(lr = 0.0001),
                loss = 'categorical_crossentropy',
                metrics=['accuracy'])
```

Now we have built our CNN model. We will now train our data on the training data. Before that let's split our data in to train and validation sets to validate our results to observe the effects of overfitting.

Validation Set: Well, the ultimate goal of our model is to be accurate on new data, not just the data we are using to build it. Also, to observe if our model is overfitting on the train dataset meaning that the model is probably learning background noise while being fit. To observe this effect, we need a 'validation set'.

This data set is used to minimize overfitting. You're not adjusting the weights of the network with this data set, you're just verifying that any increase in accuracy over the training data set actually yields an increase in accuracy over a data set that has not been shown to the network before, or at least the network hasn't trained on it (i.e. validation data set).

In our project we have split our training set into to a 'Train' and 'Validation' sets in 80:20 ratio. So, we will train our model on 48,000 samples and validate on 12,000 samples.

We use fit method to train our model. Also, we will be optimizing the weights to improve model efficiency and now the question is when are we optimizing the weights?

Hence, we use **batch-size** to specify the number of observations after which we want to update the weights. **Epoch** is nothing but the total number of iterations. There is no specific rule or method to choose batch-size and epoch values. It's a trial and error, we had experimented different values and finally stick to use the below mentioned values.

Let's see how we fit our model in python.

```
cnn_all.fit(X_train, y_train,
            validation_data=(X_validate, y_validate),
            epochs=50,
            batch_size=300)
```

Finally, we predict the test set result to see how well our model is performing on new unseen data. Below is how we do it, which returns test-loss and test-accuracy.

```
cnn_all.evaluate(X_test, y_test, verbose=0)
```

Generalization & Regularization: Generalization in machine learning refers to how well the concepts learned by the model apply to examples which were not seen during training. The goal of most machine learning models is to generalize well from the training data, in order to make good predictions in the future for unseen data. Overfitting happens when the models learns too well the details and the noise from training data, but it doesn't generalize well, so the performance is poor for testing data.

Regularization is a key component in preventing overfitting. Here are some of the regularization techniques we have used in this project to prevent overfitting.

1. Data Augmentation: Deep networks need large amount of training data to achieve good performance. To build a powerful image classifier using very little training data, image augmentation is usually required to boost the performance of deep networks. Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear, zoom and flips, etc.

Here we will show how we have done image augmentation in this project.

```
imgen = ImageDataGenerator(rotation_range=10,
                           width_shift_range=0.1,
                           height_shift_range=0.1,
                           shear_range=0.15,
                           zoom_range=0.1,
                           horizontal_flip=True,
                           vertical_flip=False)
```

2. Dropout: Dropout is a technique where random neurons are dropped or ignored during training. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

If neurons are dropped randomly out of the network then the other neurons have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple representations being learned by the network.

The effect is that the network becomes less sensitive to specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to over fit the training data.

This is how Dropout is implemented in Keras.

```
cnn_all.add(Dropout(0.3))
```

3. Batch Normalization: Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift). To increase the stability of a neural network and to speed up learning, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

Batch Normalization in python.

```
BatchNormalization()
```

Hyper Parameters: Hyper parameters are the variables which determines the network structure and the variables which determine how the network is trained. Hyper parameters are set before training. Here are some hyper parameters which we have tuned in this project.

Hyper parameters related to Network structure:

- Number of Hidden-Layers & Units
- Dropout

Hyper parameters related to Training Algorithm:

- Learning Rate
- Number of Epochs
- Batch size

Modelling Strategy:

Because there are lots of things that can be tuned like Network Architecture, Layer parameters, Hyper parameters. There are so many different combinations to explore. So, let's do it step by step.

First, we built a basic Artificial Neural Network with a test accuracy of **88.5**.

We will now start with a basic one-layer CNN and then increase the layers and Units getting the best validation performance possible also parallelly working on the overfitting issue in the process. Our main goal is to get the best test accuracy without overfitting on the training data.

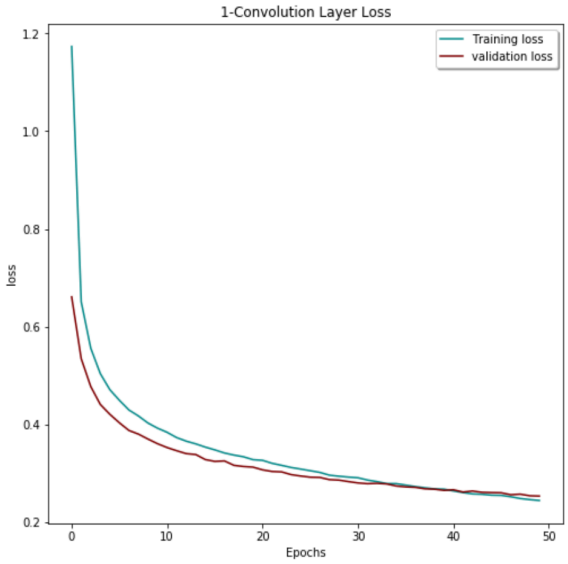
Now Let's do it!!!

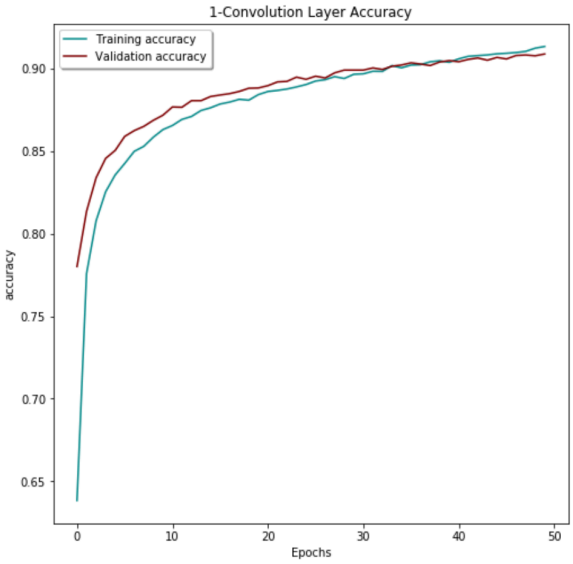
Basic 1-Layer CNN:

- One Convolution Layer with 32 3x3 filters.
- One Max Pooling Layer with 2x2 pooling.
- Flatten Layer.
- Dropout (0.25).
- Fully Connected Layer with 128 neurons.
- Dropout (0.3)
- Output Layer with 10 neurons.

Test Accuracy: **91.25**

Loss & Accuracy Plots



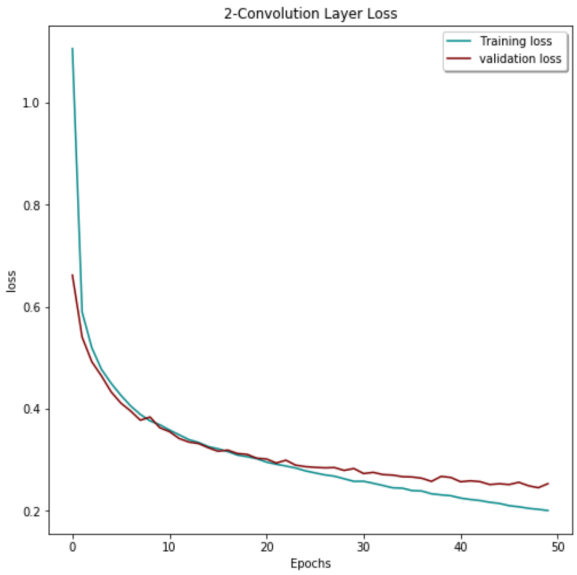


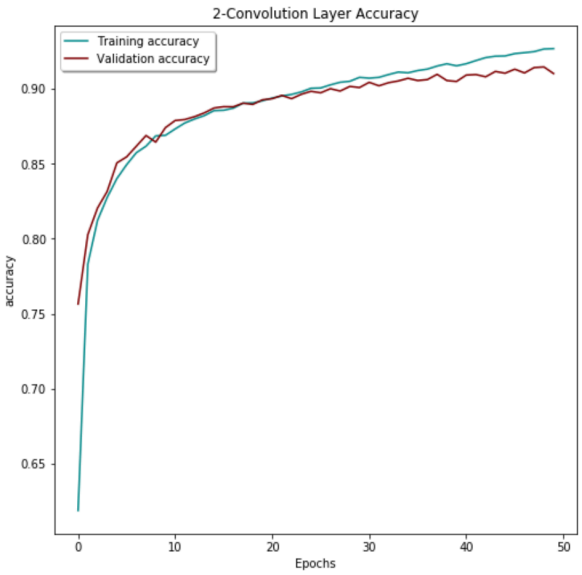
2-Layer CNN:

- Two Convolution Layers with 32 & 64 3x3 filters.
- Dropout (0.2).
- One Max Pooling Layer with 2x2 pooling.
- Dropout (0.3).
- Flatten Layer.
- Two Fully Connected Layer with 64 & 128 neurons.
- Two Dropout's (0.3).
- Output Layer with 10 neurons.

Test Accuracy: **91.73**

Loss & Accuracy Plots



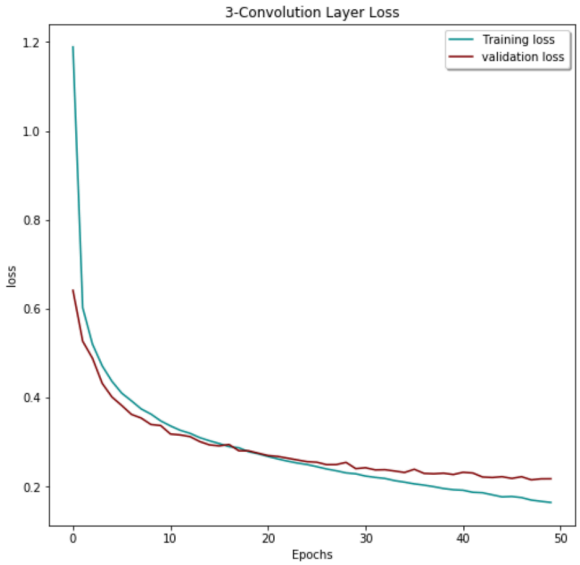


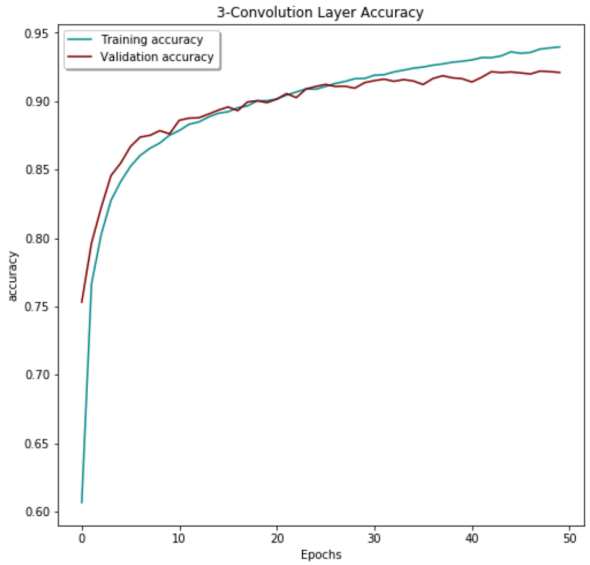
3-Layer CNN:

- Three Convolution Layers with 32, 64 & 128 3x3 filters.
- Two Max Pooling Layer with 2x2 pooling.
- Flatten Layer.
- Two Fully Connected Layer with 128 & 256 neurons.
- Three Dropout's (0.5).
- Output Layer with 10 neurons.

Test Accuracy: **92.55**

Loss & Accuracy Plots



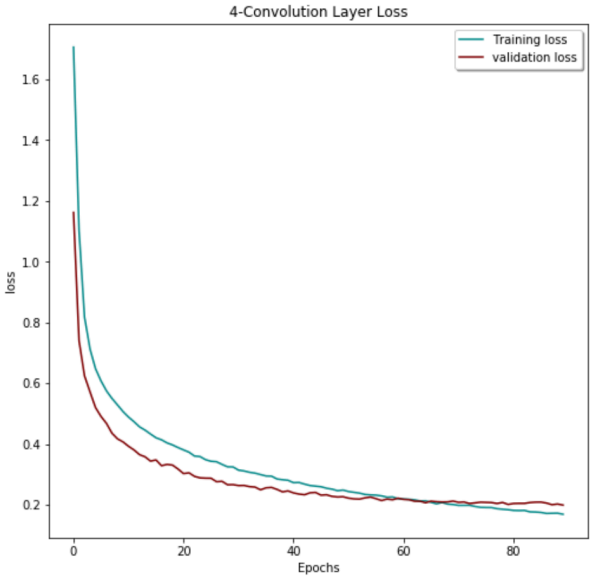


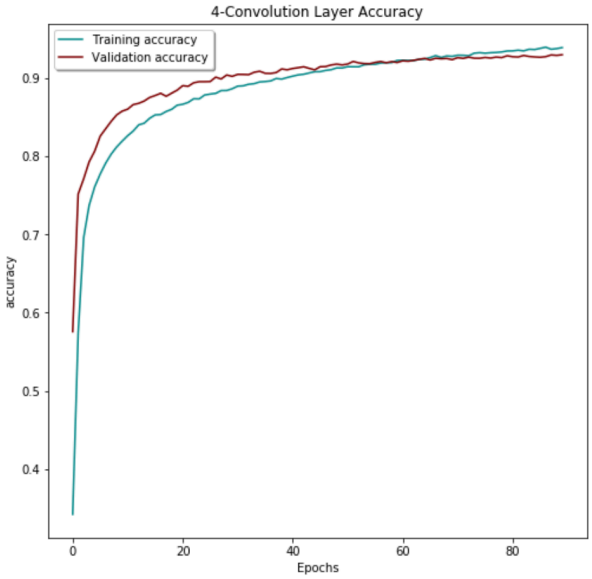
4-Layer CNN with Padding: As we add more and more Convolution & Pooling Layers the spatial dimensions of the image decrease rapidly so to avoid this we have implemented padding here.

- Four Convolution Layers with 32, 64, 128 & 256 3x3 filters.
- Two Max Pooling Layer with 2x2 pooling.
- Three Dropout's (0.3).
- Flatten Layer.
- Three Fully Connected Layer with 128, 256 & 512 neurons.
- Three Dropout's (0.5).
- Output Layer with 10 neurons.

Test Accuracy: **93.52**

Loss & Accuracy Plots



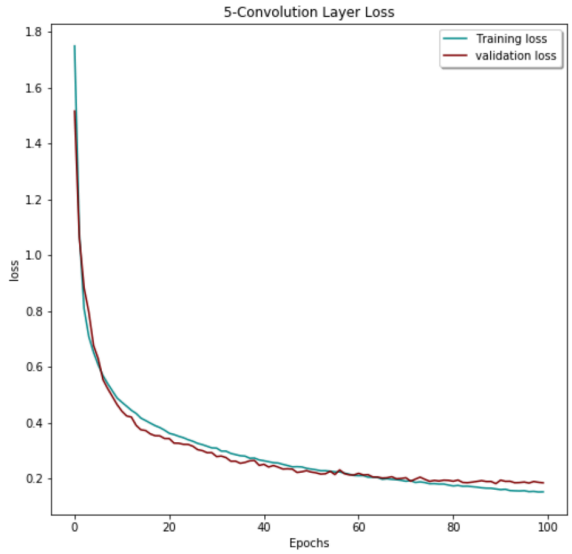


5-Layer CNN with Padding & Batch Normalization:

- Five Convolution Layers with 32, 64, 128, 256, 512 & 1024 3x3 filters with padding.
- Three Max Pooling Layer with 2x2 pooling.
- Four Dropout's (0.5).
- Flatten Layer.
- Three Fully Connected Layer with 256, 512 & 1024 neurons.
- Three Dropout's (0.5).
- Output Layer with 10 neurons.
- Added Batch Normalization to the inputs of the activation layers in both convolutional layers and fully connected layers.

Test Accuracy: **93.80**

Loss & Accuracy Plots



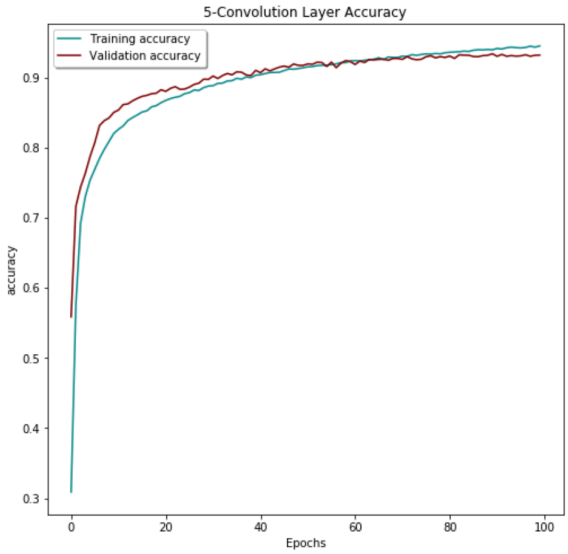


Image Augmentation: Later we have implemented image augmentation on the 5-Layer CNN to boost the performance of the classifier.

Test Accuracy: **94.30**

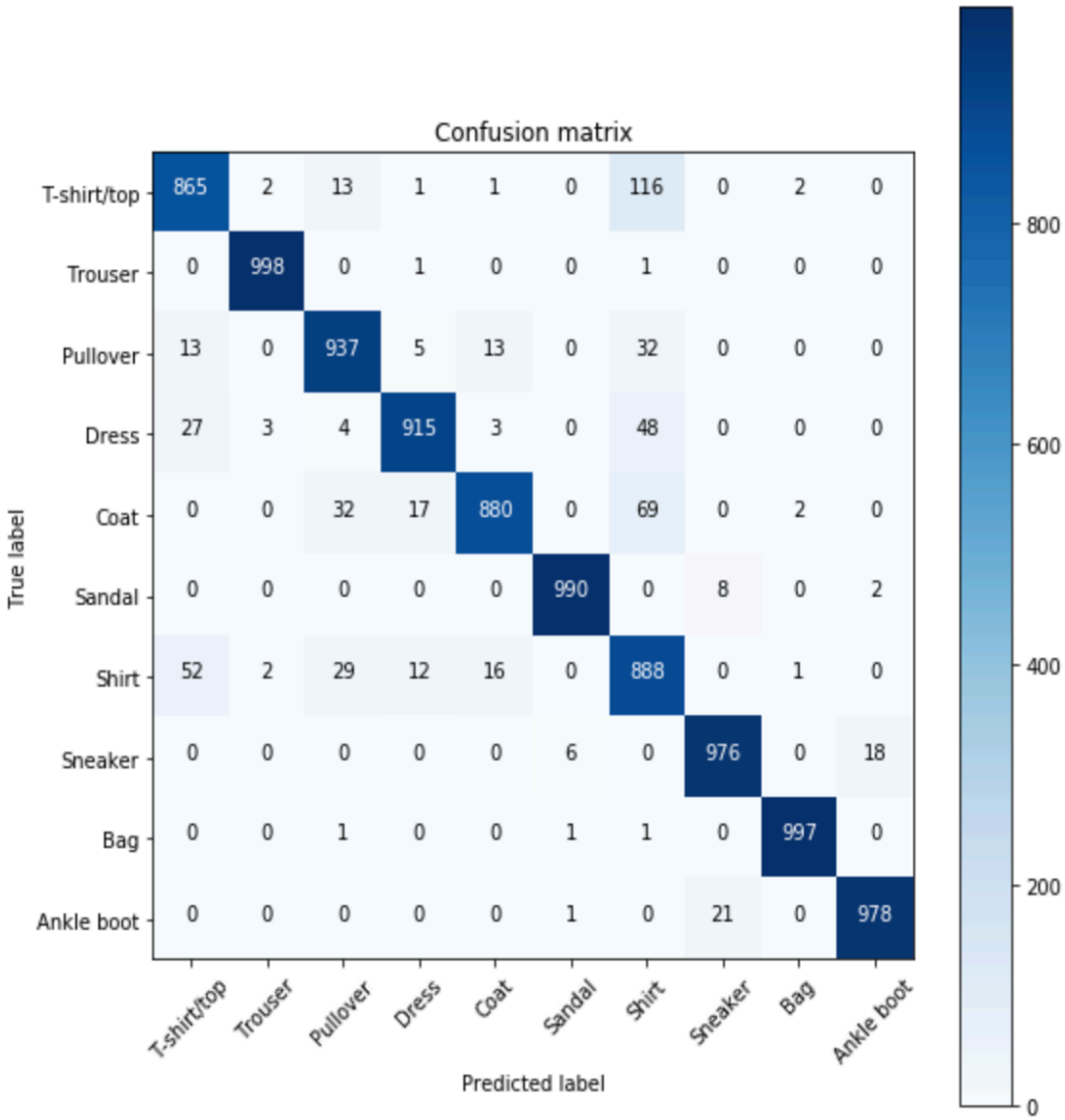
Now that we have observed the accuracy of each classifier let's take a look on other metrics like precision, recall and f1-score.

	precision	recall	f1-score	support
0.0	0.90	0.86	0.88	1000
1.0	0.99	1.00	1.00	1000
2.0	0.92	0.94	0.93	1000
3.0	0.96	0.92	0.94	1000
4.0	0.96	0.88	0.92	1000
5.0	0.99	0.99	0.99	1000
6.0	0.77	0.89	0.82	1000
7.0	0.97	0.98	0.97	1000
8.0	1.00	1.00	1.00	1000
9.0	0.98	0.98	0.98	1000
avg / total	0.95	0.94	0.94	10000

Different CNN's & Accuracies

CNN	Classifier Accuracy
1-Layer	91.25
2-Layer	91.73
3-Layer	92.55
4-Layer with Padding	93.52
5-Layer with padding	93.69
5-Layer with Padding & Batch Normalization	93.89
Image Augmentation on 5-Layer	94.30

Confusion Matrix:



And now we can observe the misclassifications by plotting the confusion matrix. Confusion Matrix helps us to describe the performance of the classifier visually by plotting the true labels and predicted labels.

Conclusion

Finally, we can say that by seeing the above plotted confusion matrix that most of the misclassifications are happening between the classes Shirt, T-shirt/top, Pullover and Coat which are majorly impacting the performance of the classifier.

And Deep Networks require large amount of training data to achieve good performance which we observed in our analysis how image augmentation boosted the classifier performance. To further improve the classifier performance, we should collect more samples and give more images from these 4 classes to the model so the classifier can learn more features or patterns even better.