Question 2 : Classification: Convolutional Neural Networks

2.1: Design and Implementation Choices of your Model

The report is based on a classification of the Fashion MNIST dataset available on the Kaggle ECE657A competition. The dataset is consists of a training set of 60,000 images and a test set of 10,000 images. Each image is a 28x28 (784 pixels) grayscale image, associated with a label from 5 classes(the only difference between the original dataset & the dataset used in this report). 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 & test sets have 784 columns corresponding to the pixel values of the images. The training set also consists of a label column which categorizes the images in five labels: [0,1,2,3,4]

This question deals with the classification of the dataset using a convolutional neural network architecture. Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problem. The big idea behind CNNs is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model.

A convolution is a weighted sum of the pixel values of the image, as the window slides across the whole image. Turns out, this convolution process throughout an image with a weight matrix produces another image (of the same size, depending on the convention). Convolving is the process of applying a convolution.

The sliding-window shenanigans happen in the convolution layer of the neural network. A typical CNN has multiple convolution layers. The beauty of CNN is that the number of parameters is independent of the size of the original image.

The CNN architecture presented to solve classification problems is inspired by popular CNN architectures Lenet5 [2], Alexnet [3], and VGG [4] network architectures. The architecture is designed with basic patterns explained in above mentioned traditional CNN architectures.

Dataset

The dataset is plotted into train & test sets with a ratio of 80:20. The splitting is achieved to evaluate the model predictions and accuracy on unseen data.

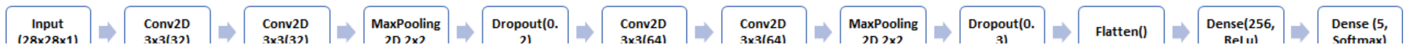
1. The CNN model Implementation and architecture

1. Data Pre-Processing:

The pixel values of images in the dataset range from 0 to 255. The higher pixel values will dominate all other pixels while predicting the class of the image. So feature scaling is applied to bring all the data to the same scale i.e. between o & 1. Min-Max Normalisation is used in this regard. Each pixel value is divided by 255 to achieve the required results. The training and testing input images are reshaped to 28x28x1 tensor as accepted by the CNN model. 28x28 represents the image size and 1 represents the number of channels. Since our image is grayscale, so we have only 1 channel.

1. Model Architecture:

A CNN model consists of many layers. The idea behind the design of model architecture is to reduce the rectangular field size with a gradual increase in channel depths resulting in the detangled feature extraction.


Input (28x28x1)
Conv2D 3x3(32)
Conv2D 3x3(32)
MaxPooling 2D 2x2
Dropout(0.2)
Conv2D 3x3(64)
Conv2D 3x3(64)
MaxPooling 2D 2x2
Dropout(0.3)
Flatten()
Dense(256, ReLu)
Dense (5, Softmax)

1. The input image is considered of size 28x82x1 as explained above.
2. Convolution Layer: The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. The convolution is applied to the input data using a convolution filter to produce a feature map. A kernel of size 3x3 is chosen and a filter size of 32 and the same padding is chosen to begin extracting useful features from the model. Initially, it is recommended to start with the small number of filters and a small kernel size to extract detailed features like edges, borders, etc. and move on with increasing the number of filters to extract globals features. Stride specifies how much we move the convolution filter at each step. A slide of 2 is used in this architecture. Two convolution layers are created in the model.
3. Max Pooling: After a convolution operation, we usually perform pooling to reduce the dimensionality. This enables us to reduce the number of parameters, which both shortens the training time and combats overfitting. Pooling layers downsample each feature map independently, reducing the height and width, keeping the depth intact. The output of the above layer is passed to a max-pooling using a 2x2 window and stride 2.
4. DropOut & Another Convolution-Pooling Layer: To get the regularising effect and avoid overfitting of the model, a drop out of 0.2 is used after this layer. The output of the above layers is passed through another convo-convo-pooling-drop layer combination to increase the channel depth and reduce the output size. The convolutional layer has a filter size 3x3 and 64 filters with the same padding which is followed by the Max Pooling layer of 2x2 window size and strides 2 and a dropout of 0.3.
5. Flatten: The output of the above layers might have extracted all the significant features required to classify the dataset. The output is passed to the Flatten layer which flattens the output into one 1x1 format.
6. Dense: The flatten output is being fed to the two dense layers of different numbers of neurons. The first layer consists of 256 numbers of neurons followed by the layer with 5 neurons. The activation function used in the first layer is a rectified linear unit (ReLu) to avoid saturation of gradients. The last dense layer of the model is using a softmax activation function to predict the classes based on the probability.

Optimizers

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rates to reduce the losses. Optimizers help to get results faster. Adaptive Moment Estimation (Adam) is a popular algorithm in the field of deep learning because it achieves good results fast. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. It also addresses the downside of slow convergence and the high variance of the internal parameters found in the optimization algorithm for gradient descent.

Regularisor

If you've built a neural network before, you know how complex they are. This makes them more prone to overfitting. Regularization is a technique that makes slight modifications to the learning algorithm such that the model generalizes better. This, in turn, improves the model's performance on the unseen data as well. The term “dropout” refers to dropping out units (both hidden and visible) in a neural network. Dropout is a regularization technique and is most effective at preventing overfitting.

Activation Functions

ReLU stands for the rectified linear unit and is a type of activation function. Mathematically, it is defined as y = max(0, x). Visually, ReLU is the most commonly used activation function in neural networks, especially in CNNs.ReLu activation is used to avoid the gradient saturation problem whereas softmax is used for selecting the parameter associated with a maximum probability value.

2. Other ML methods and Deep Neural Network variants

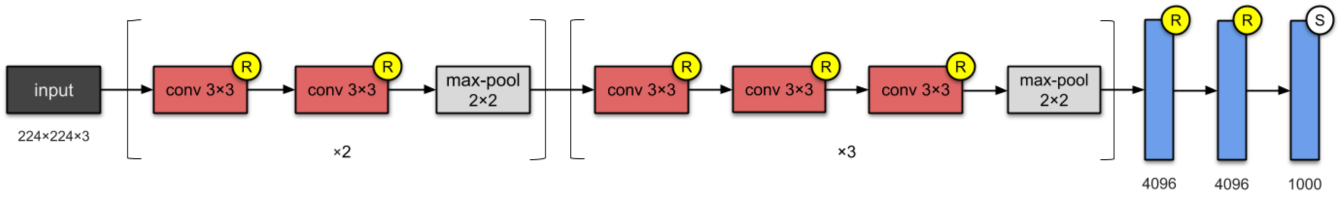
Deep neural networks and Deep Learning are powerful and popular algorithms.

1. Deep Neural Network(DNN)

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output.

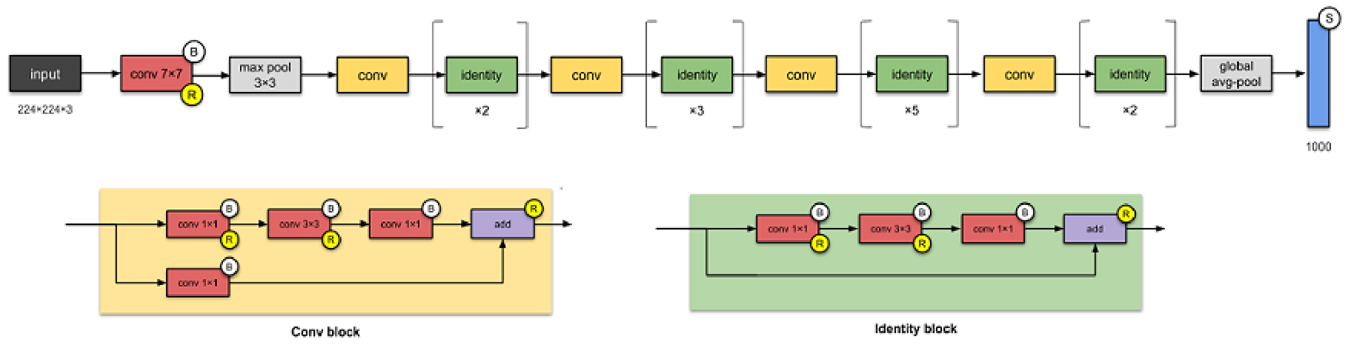
DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network

2. VGG-16



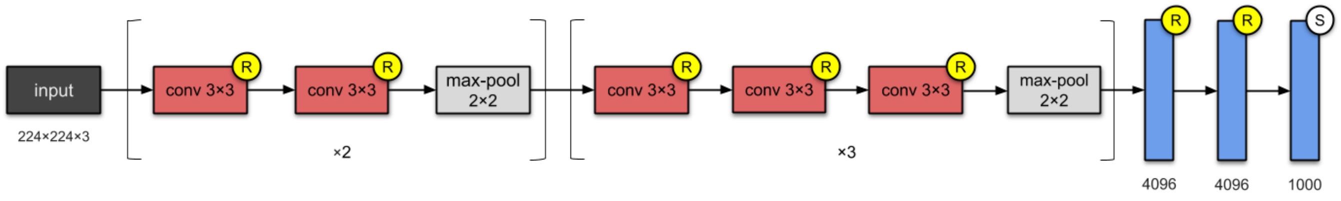
The most straightforward way of improving the performance of deep neural networks is by increasing their size (Szegedy et. al). The folks at Visual Geometry Group (VGG) invented the VGG-16 which has 13 convolutional and 3 fully-connected layers, carrying with them the ReLU tradition from AlexNet. This network stacks more layers onto AlexNet, and use smaller size filters (2×2 and 3×3). It consists of 138M parameters and takes up about 500MB of storage space. They also designed a deeper variant, VGG-19.

3. ResNet-50



The other CNNs, have an increased number of layers in the design to achieve better performance. But “with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.” Microsoft Research addressed this problem with ResNet — using skip connections (a.k.a. shortcut connections, residuals) while building deeper models. ResNet is one of the early adopters of batch normalization. Shown above is ResNet-50, with 26M parameters. The basic building block for ResNets is the conv and identity blocks.

It has even deeper CNNs (up to 152 layers) without compromising the model's generalization power. Reset used skip connections to improve the efficiency of the model. A skip connection based residual block looks like the one shown in Figure below :



Skip connections mitigate the problem of vanishing gradient by allowing this alternate shortcut path for the gradient to flow through. They allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse.

These models enable us to train very deep neural network architectures, without facing problems of degradation of gradients. The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.

2.2 Implementation of your Design Choices

In [ ]:
```
#Loading the data
X = train_data[train_data.columns[2:786]].values
Y = np.ravel(train_data[['Label']])
Y = keras.utils.to_categorical(Y,num_classes)

testing_data= pd.read_csv("/kaggle/input/cnndata/testX.csv")
testing_data=testing_data[testing_data.columns[1:785]].values

#Feature Scaling
X=X/255
testing_data =testing_data/255

#Splitting the data
X_train,X_val,y_train,y_val = train_test_split(X,Y,test_size = 0.2,random_stat
e = 42)


#Reshaping the data
testing_data = testing_data.reshape(testing_data.shape[0], 28, 28, 1)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_val = X_val.reshape(X_val.shape[0], 28, 28, 1)
```

The complete training dataset is split into the two parts i.e. Training dataset (80%) and the validation(20%) dataset. The performance on the validation set is used to compare the results of various algorithms and architecture used. The data is pre-processed by dividing the feature values by 255 and scaling it to 0 & 1. The input training & testing data is reshaped in a 4D tensor format as accepted by the CNN layer.

In [ ]:
```
cnn = Sequential([
    Convolution2D(32, kernel_size=(3, 3), activation='relu',padding='same',inp
ut_shape=(28,28,1)),
    Convolution2D(32, kernel_size=(3, 3), activation='relu',padding='same'),
    MaxPooling2D(pool_size=(2, 2),strides=2),
    Dropout(0.2),
    Convolution2D(64, kernel_size=(3, 3), activation='relu'),
    Convolution2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2),strides=2),
    Dropout(0.3),
    Flatten(),
    Dense(256, activation='relu'),
    Dense(5, activation='softmax')
])
cnn.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = [
'accuracy'])
```

The architecture of the sequential model is described above.

1. The initial convolutional layer has a kernel size of 3 and depth 32 with the same padding. This filter is used to extract relationships between each pixel & its neighboring pixels. The activation used in all the convolutional layers is a rectified linear unit(ReLu). To extract disentangled features, a second similar convolutional layer of 3x3 window and filter depth 32 is added with the same padding followed by the max-pooling layer of pool size (2,2) and stride 2 resulting in the output size of 14x14x32.
2. To reduce overfitting a dropout layer of 0.2 is applied. To increase the channel depth further, two convolutional layers of 3X3 filter and an increased size of 64 filters are added to the model with the same padding. The output of this layer is of size 10x10x64. Another max-pooling layer of pool size (2,2) and stride 2 is added and thus reducing the output size to 5x5x64. Another drop out of 0.3 is used.
3. The features are extracted and are then flatten before passing it to the dense layers for classification. It outputs the shape of size of 1600. The output is then passed to the fully connected layers.
4. The flatten output is passed through the dense layer of 256 neurons with the ReLu activation function. The 256 neurons are passed to the second dense layer consisting of 5 number of neurons with Softmax activation as output layer. Softmax activation is used because the data is to be classified in more than 2 classes.

Model Compilation

The model made is compiled using Adam optimizer and categorical cross-entropy loss. These are selected so that our model gets trained with minimum loss while classifying multi-label data. A commonly used metrics: Accuracy is used to validate the performance of the model.

Model Training:

In [ ]:
```
cnn.fit(X_train, y_train,
              batch_size=256,
              epochs=40,
              verbose=1,
              validation_data=(X_val,y_val))
```

The model is trained with a batch size of 256 and epochs of 40. The model trains over 48,000 images of the training set and simultaneously validates the training on the 12,000 images of the test set. The model trains for over 40 epochs while fitting 256 images at a time.

2.3 Kaggle Competition

90.62 % Accuracy is achieved on the leaderboard implementing the CNN based model. The model is trained on the complete training data set provided to achieve this accuracy.

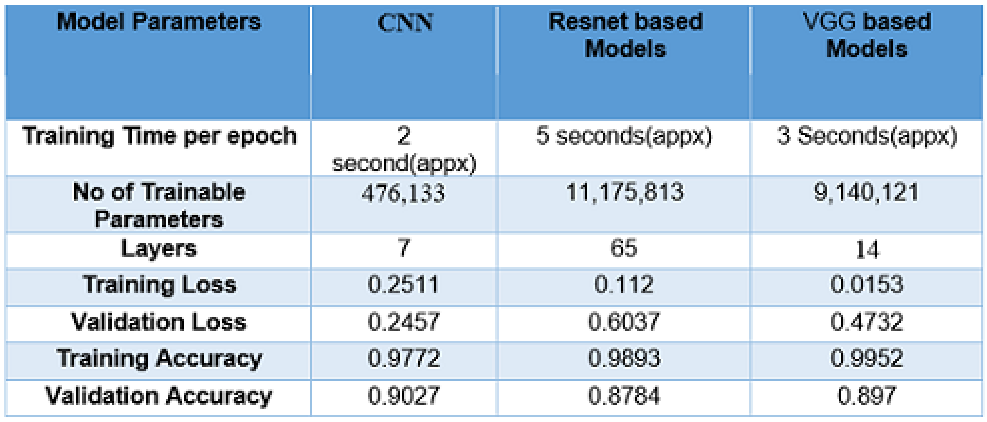
The variation in the accuracy is due to the various parameters being used by a CNN model. The type of optimizers used, several layers, activation functions, filter size, kernel size, etc highly influence the accuracy of the model. These parameters are changed to improve accuracy scores. Different teams might have used different optimizers like SGD, ADM, etc and some might have designed different architectures like Resnet, Googlenet to achieve different results. since the number of parameters to tweak is larger, it becomes highly complex and difficult to find the best combination of these parameters. Therefore, the variation in different performances can be attributed to the choice of hyperparameters and the architecture of CNN models.

2.4 Results Analysis

1. Runtime performance for training and testing:

The time taken to complete one epoch was 2 seconds while the validation was tested within 32 microseconds. Since we had 40 epochs it took 80 seconds to complete the training and validation process. This time is platform dependent. We achieved this time by using GPU and CPU available on Kaggle. GPU tends to simulate the training process and thus saving a lot of time.

2. Comparison of the different algorithms



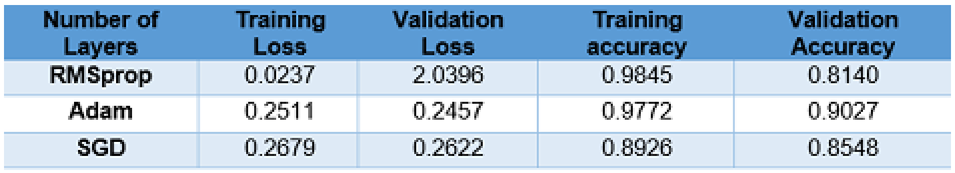
Model Parameters	CNN	Resnet based Models	VGG based Models
Training Time per epoch	2 second(appx)	5 seconds(appx)	3 Seconds(appx)
No of Trainable Parameters	476,133	11,175,813	9,140,121
Layers	7	65	14
Training Loss	0.2511	0.112	0.0153
Validation Loss	0.2457	0.6037	0.4732
Training Accuracy	0.9772	0.9893	0.9952
Validation Accuracy	0.9027	0.8784	0.897

The table above depicts the accuracy using different types of models. It is observed that

1. Maximum validation accuracy of 90.27% was achieved by CNN model with the architecture explained in Section 2.1. Although Resnet & VGG-16 model are effective enough to classify data, they were unsuccessfull in classifying this dataset.
2. The validation loss achieved by the model is least with CNN followed by VGG and Resnet. The VGG model seems to have be overfitted due to a large gap in its training and validation accuracy. Moreover, there is a large gap between its training & validation loss as well.
3. The traditinal CNN model has a training accuracy of 7% more than its validation accuracy. There is a small difference in its two losses indicating a low chance of overfitting. This might have been the due to the fact of using 2 dropout layers after each pooling layer.
4. Since Resnet & VGG model are highly dense networks, they took more time in training an epoch as compared to the traditional CNN model.
5. Training and validation losses are negatively related with training and validation accuracies respectively. As the accuracy increases the losses decrease and vice versa.

2. Comparison of the parameters

Following table shows performance of different optimizers with activation function ReLu approaches, epochs = 40 and a batch size of 256.

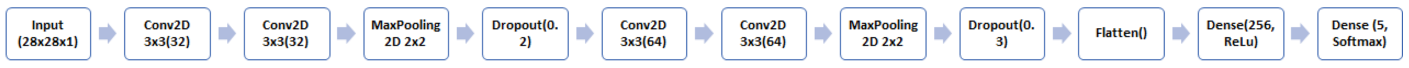


Number of Layers	Training Loss	Validation Loss	Training accuracy	Validation Accuracy
RMSprop	0.0237	2.0396	0.9845	0.8140
Adam	0.2511	0.2457	0.9772	0.9027
SGD	0.2679	0.2622	0.8926	0.8548

The table depicts that:

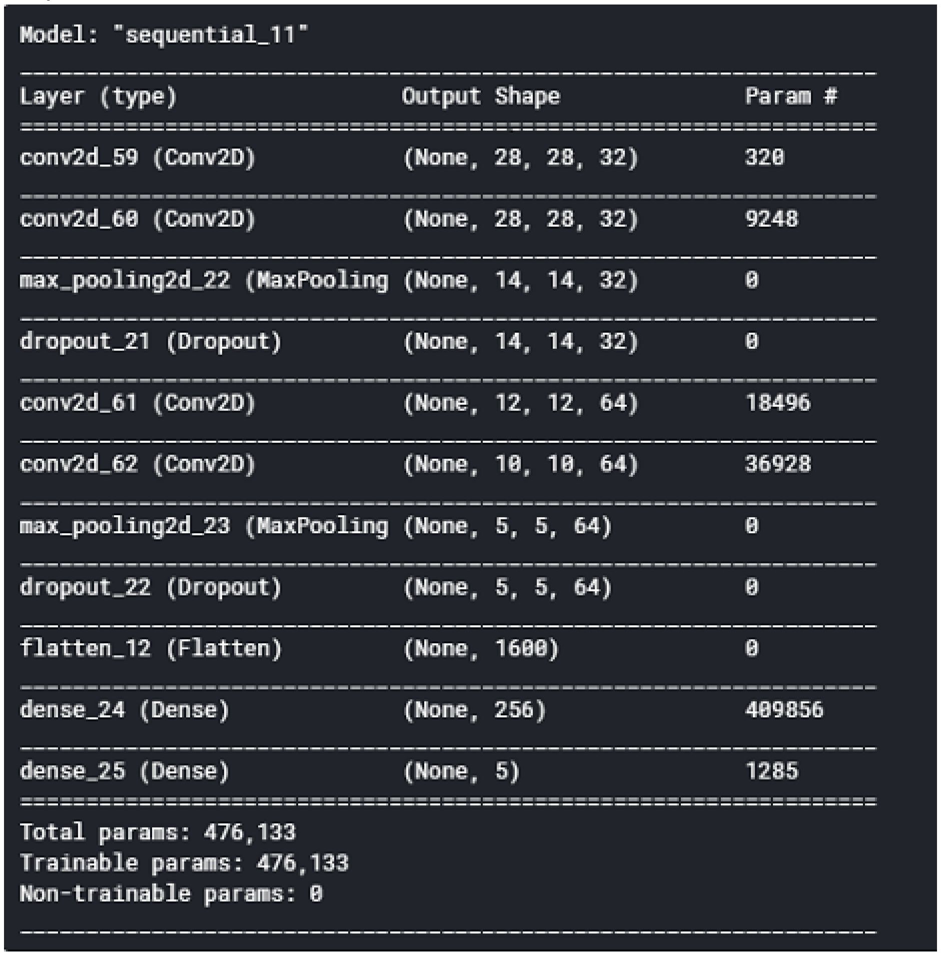
1. Minimization of loss function with RMSprop optimizer overfits the model more compared to Adam and SGD optimization algorithms.
2. SGD optimizer training accuracy is 89.26 which depicts that the model is not yet rained properly and thus will take more epochs to train it.
3. Adam is preferred as a Optimizer in classifying this dataset, since it gave the least overfitted model amd gave a pretty good accuracy.

3. Explanation of your model (algorithms, network architecture, optimizers,regularization, design choices, numbers of parameters)



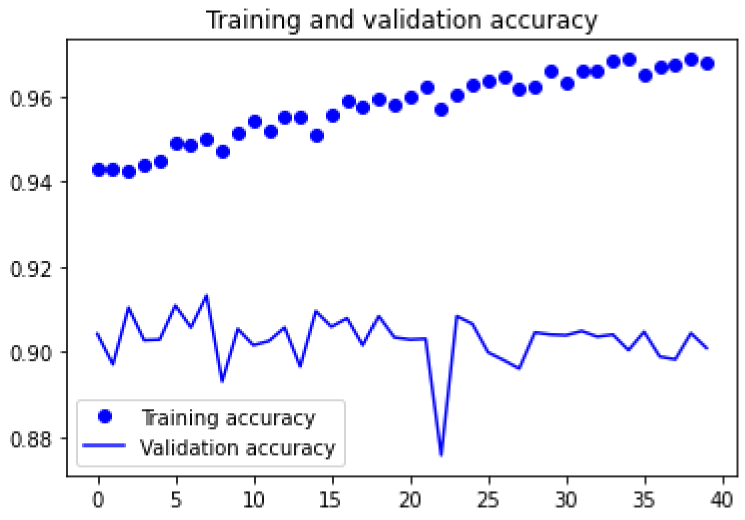
The model architecture is explained in Section 2 & 3. The following parametrs are used:

1. Adaptive Moment Estimation (Adam) optimizer is used since it gave it didn't overfitted the model.
2. Binary cross-entropy is for multi-label classifications, whereas categorical cross entropy is for multi-class classification where each example belongs to a single class. Therefore, categorical cross entropy loss function was used.
3. Droput layer with increasing value was added to reduce the overfitting.
4. Two successive convolution layer pattern revealed a better result than convo-pool pattern by 3.57%. The model layers are depicted below:

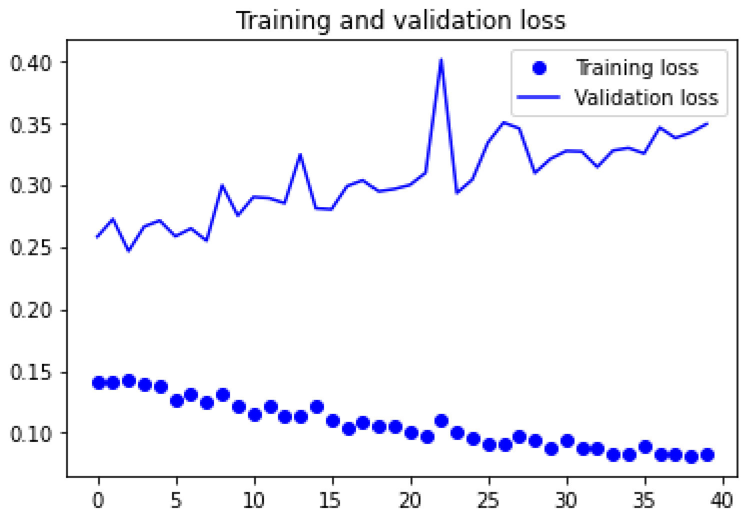


```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_59 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_21 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
dropout_22 (Dropout)         (None, 5, 5, 64)          0
_________________________________________________________________
flatten_12 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_24 (Dense)             (None, 256)               409856
_________________________________________________________________
dense_25 (Dense)             (None, 5)                 1285
=================================================================
Total params: 476,133
Trainable params: 476,133
Non-trainable params: 0
_________________________________________________________________
```

4. Plots



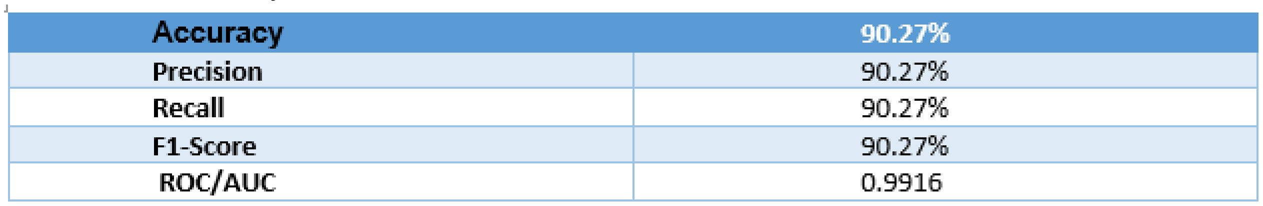
Above plot shows the values of the accuracy of both the training and validation dataset for each number of the epoch.The model is run for 40 number of epochs. With increase in number epochs, training accuracy follows the increasing trend till the last number of epochs. On the other side, the accuracy of the validation test changes while the model gets trained better. However, the model gets to a validation accuracy of approximately 90.27% which is pretty decent.



Above plot shows the values of the loss function of both the training and validation dataset for each number of the epoch.The model is run for 40 number of epochs. It is seen that the training loss decreases with increase in the number of epochs as the model gets trained with new features in each epoch. However, the validation loss tends to increase with time. Although there is light difference in both the losses after the ending of 40 epochs, the validation loss reached 0.3357.

5. Metrics for Validation:

The model can be evaluated on the different accuracy matrices such as accuracy, F1-score, ROC AUC etc. Values for different accuracy matrices are shown in the table below:



Accuracy	90.27%
Precision	90.27%
Recall	90.27%
F1-Score	90.27%
ROC/AUC	0.9916

To understand the model well its performance have been validated based on the metrics mentiioned in the table. In our model the precision is 90.27% which tells us aabout the sensitivity of the model. The Reacall(90.27%) tells the true positive rate whichconfirms the model's excellent ability to distinguish between different classes (true positives, false positive and false negative values are taken into consideration).F1-score(90.27%) shows the balance between precision and recall. At last high value (.9916, i.e. close to 1) of area under the ROC (Receiver Operating Curve) shows high true positive rate in the model's ability to classify images into respective categories.

References:

1. https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d (https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d)
2. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998
3. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.
4. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
5. https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e276 (https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e276)
6. https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks (https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks)

Libraries Used:

1. tensorflow
2. Keras
3. Matplotlib
4. numpy