HW2

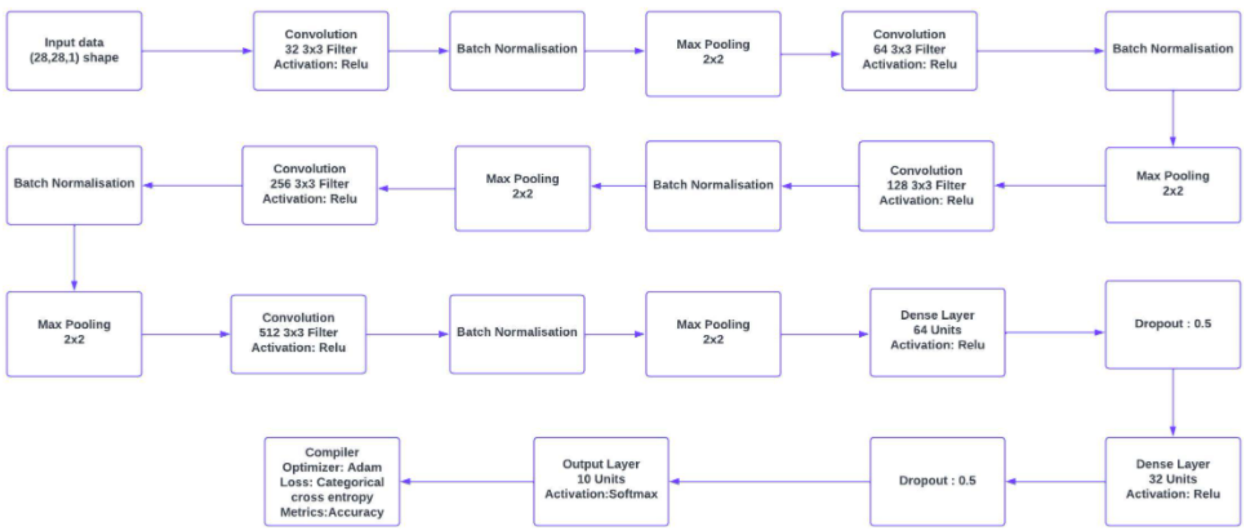
Problem 1: Apply the following models on the Fashion Mnist Dataset. Train the model with the training data and evaluate the model with the test data.

a: CNN model from scratch: Develop a CNN model with 5 convolutional layers (with kernel size= 3, stride = 1, padding = "same", activation function = "relu" (with following Max Pooling layer (Size= 2) and 3 fully connected layer (including one output layer). After each of the Convolutional layer apply Batch Normalization. In the fully connected layer apply dropout.

Ans:

<u>Model description, summary and hyper-parameters used are given below. For detailed description of the model, accuracy and prediction results, please refer to the notebook.</u>

Model Architecture



1. The input shape is (28,28,1), which means that the input image has a height and width of 28 pixels, and a depth of 1 channel (grayscale).
2. This model is a convolutional neural network (CNN) architecture. It consists of 5 convolutional layers, each followed by a batch normalization layer and a max pooling layer.
3. The output from the convolutional layers is flattened and then fed into two fully connected layers, each with a ReLU activation function and dropout regularization.
4. The final layer is a dense layer with a softmax activation function to produce the output probabilities.
5. The model uses the Adam optimizer and categorical cross-entropy loss function to train, and accuracy is used as the evaluation metric.
6. Model is fitted on the trained data with 20 epochs and batch size 32 and validation and trained data accuracy and loss is compared to see that model is not overfitting

Hyper-parameters Used

Number of Convolution Layers: 5					
Convolutions Layers	Filter Size	Stride	Number of Filters	Padding	Activation Function
Convo Layer 1	3x3	1	32	Same	Relu
Convo Layer 2	3x3	1	64	Same	Relu
Convo Layer 3	3x3	1	128	Same	Relu
Convo Layer 4	3x3	1	256	Same	Relu
Convo Layer 5	3x3	1	512	Same	Relu

Number of Hidden Layers: 2		
Hidden Layers	Neurons	Activation Function
Hidden Layer 1	64	Relu
Hidden Layer 2	32	Relu
Output Layer		
Output Layer	Neurons	Activation Function
	10	Softmax

Regularisation	
Part a	Dropout p = 0.5
Part b	Dropout p = 0.5 , Data Augmentation

Model Compilation			
Loss	Optimizer	Learning Rate	Evaluation Metric
Categorical Cross entropy	Adam	0.001	Accuracy

Training		
Epochs	Batch Size	Validation Split
20	32	0.2

Weight Initialiser	Bias Initialiser
Be default Glorot	By default zeros

Model Summary

```
1 modela.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 batch_normalization (BatchN  (None, 28, 28, 32)       128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 batch_normalization_1 (Batc  (None, 14, 14, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 128)         73856

 batch_normalization_2 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 256)         295168
```

```
 batch_normalization_3 (Batc  (None, 4, 4, 256)        1024
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 conv2d_4 (Conv2D)           (None, 2, 2, 512)         1180160

 batch_normalization_4 (Batc  (None, 2, 2, 512)        2048
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 1, 1, 512)        0
 2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 1,607,210
Trainable params: 1,605,226
Non-trainable params: 1,984
_________________________________________________________________
```

The model has a total of 1,607,210 parameters, out of which 1,605,226 are trainable and 1,984 are non-trainable. The model consists of convolutional layers, batch normalization layers, max pooling layers, fully connected layers with dropout, and an output layer with softmax activation. The architecture progressively reduces the spatial dimensions of the input, from 28x28 to 1x1, while increasing the number of channels in each convolutional layer. The model was compiled with the Adam optimizer (it computes adaptive learning rates for each parameter and performs both momentum and RMSprop style updates for faster convergence), accuracy as the evaluation metric. and categorical cross-entropy loss function, and trained with accuracy as the evaluation metric.

In total, this model has 1,607,210 parameters, out of which 1,605,226 are trainable.

```
end_timea = time.time()
```

```
total_timea = end_timea - start_timea
```

```
print("Total execution time: ", total_timea, " seconds")
```

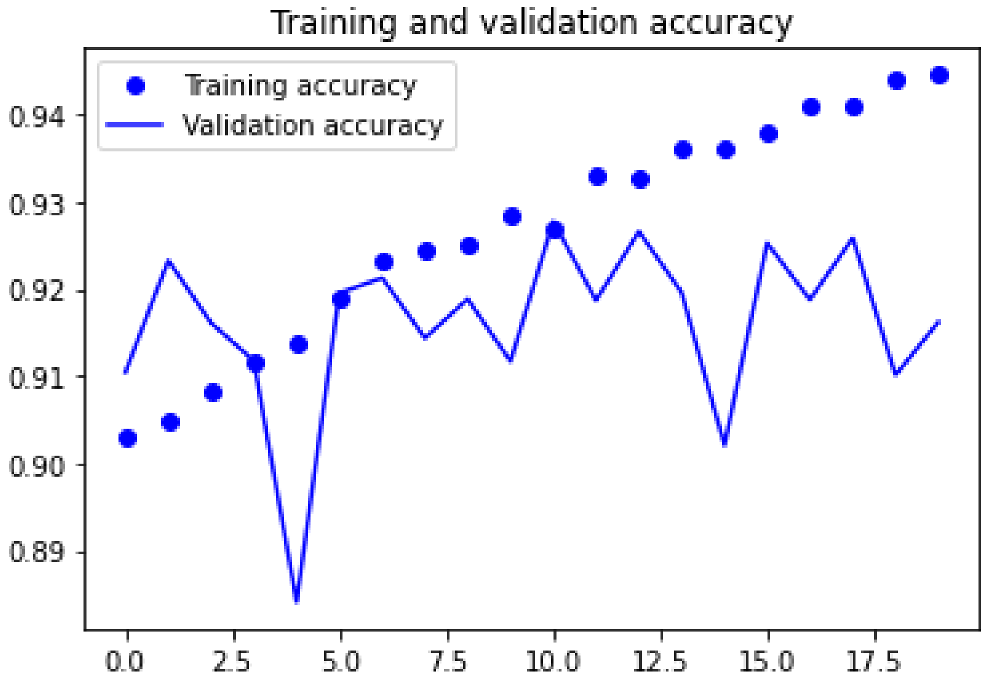
```
Total execution time:  200.49847745895386  seconds
```

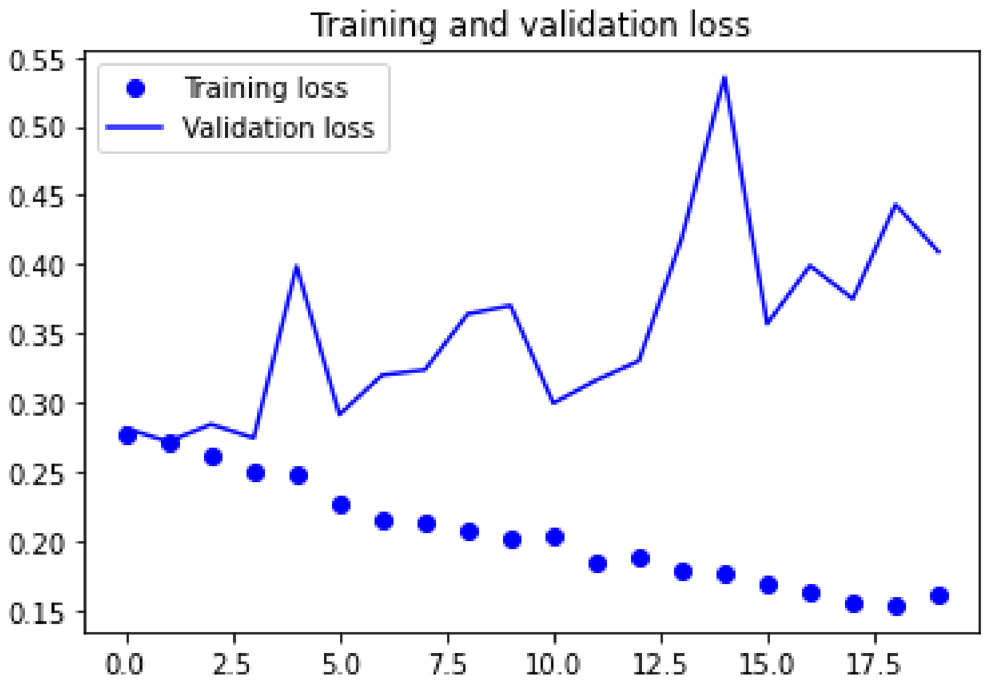
Evaluation

```
# evaluating the performance model on test data
test_evala = modela.evaluate(test_X, test_Y_one_hot, verbose=0)
```

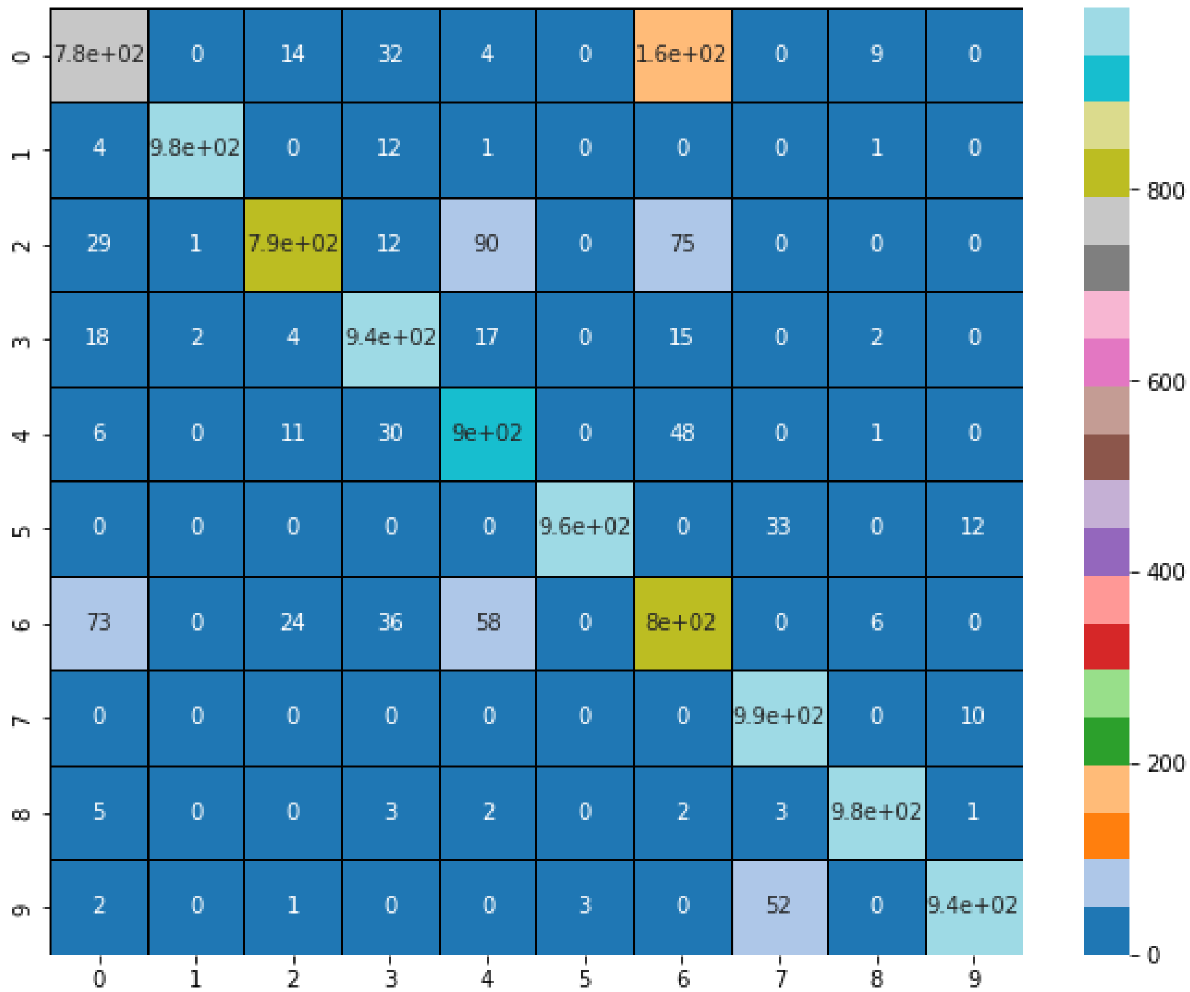
```
# printing the loss and accuracy value on test data
print('Test loss:', test_evala[0])
print('Test accuracy:', test_evala[1])
```

```
Test loss: 0.4806569218635559
Test accuracy: 0.9092000126838684
```





Confusion Matrix




```
print("Precision Score is ", precision_score(test_Y, predictiona, average = 'weighted'))
```

Precision Score is 0.9108256205924419

```
print("Recall Score is ", recall_score(test_Y, predictiona, average = 'weighted'))
```

Recall Score is 0.9077

```
print("F1 Score is " ,f1_score(test_Y, predictiona, average = 'weighted'))
```

F1 Score is 0.9079153266165798

b: Data Augmentation: Apply two image augmentation techniques on the Fashion Mnist train data to augment it and then apply the previously developed model on it.
Ans: Model architecture and Hyper-parameters used are same in previous question.
Sample showing Augmented data by rotating images up to 15 degrees, shifting images horizontally and vertically.

Model Summary

```
1 modelb.summary() # Summary of the model
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 28, 28, 32)        320

 batch_normalization_5 (Batc  (None, 28, 28, 32)       128
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 14, 14, 64)        18496

 batch_normalization_6 (Batc  (None, 14, 14, 64)       256
 hNormalization)

 max_pooling2d_6 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_7 (Conv2D)           (None, 7, 7, 128)         73856

 batch_normalization_7 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 max_pooling2d_7 (MaxPooling  (None, 4, 4, 128)        0
 2D)
```

```
 conv2d_8 (Conv2D)           (None, 4, 4, 256)         295168

 batch_normalization_8 (Batc  (None, 4, 4, 256)        1024
 hNormalization)

 max_pooling2d_8 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 conv2d_9 (Conv2D)           (None, 2, 2, 512)         1180160

 batch_normalization_9 (Batc  (None, 2, 2, 512)        2048
 hNormalization)

 max_pooling2d_9 (MaxPooling  (None, 1, 1, 512)        0
 2D)

 flatten_1 (Flatten)         (None, 512)               0

 dense_3 (Dense)             (None, 64)                32832

 dropout_2 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 32)                2080

 dropout_3 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 10)                330

=================================================================
Total params: 1,607,210
Trainable params: 1,605,226
Non-trainable params: 1,984
_________________________________________________________________
```

The model has a total of 1,607,210 parameters, out of which 1,605,226 are trainable and 1,984 are non-trainable. The model consists of convolutional layers, batch normalization layers, max pooling layers, fully connected layers with dropout, and an output layer with softmax activation. The architecture progressively reduces the spatial dimensions of the input, from 28x28 to 1x1, while increasing the number of channels in each convolutional layer. The model was compiled with the Adam optimizer (it computes adaptive learning rates for each parameter and performs both momentum and RMSprop style updates for faster convergence), accuracy as the evaluation metric. and categorical cross-entropy loss function, and trained with accuracy as the evaluation metric.

In total, this model has 1,607,210 parameters, out of which 1,605,226 are trainable.

Data augmentation has been done using Keras ImageGenerator. This helps to increase the size of the training dataset by generating new images by applying various transformations like rotation, zooming, shifting. This helps to improve the generalization of the model by reducing overfitting.

```
end_timeb = time.time()
```

```
total_timeb = end_timeb - start_timeb
```

```
print("Total execution time: ", total_timeb, " seconds")
```

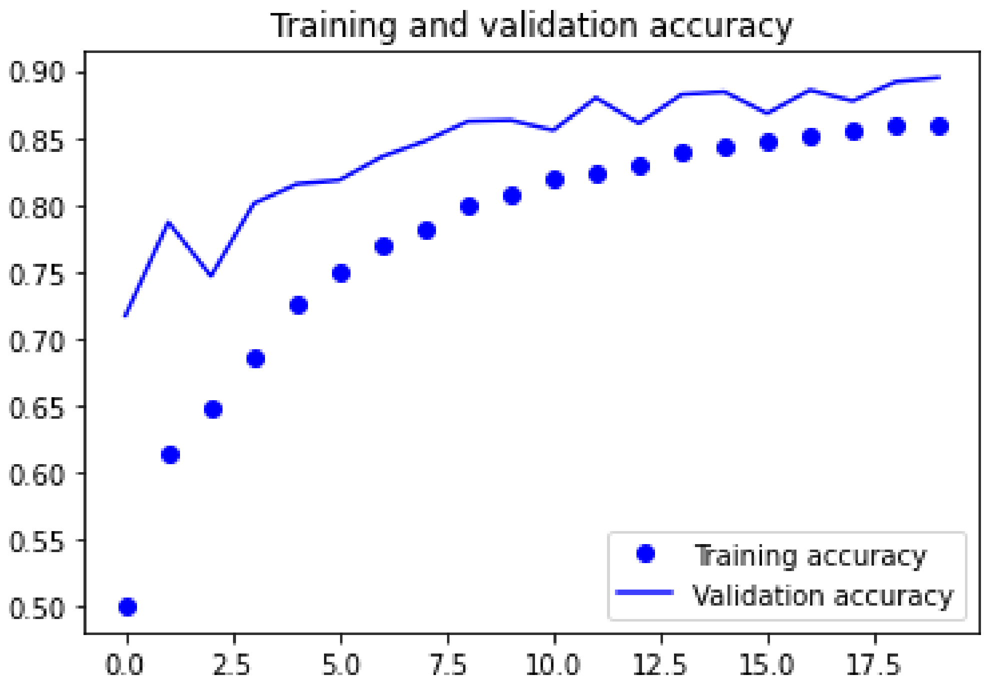
```
Total execution time:  362.29075384140015  seconds
```

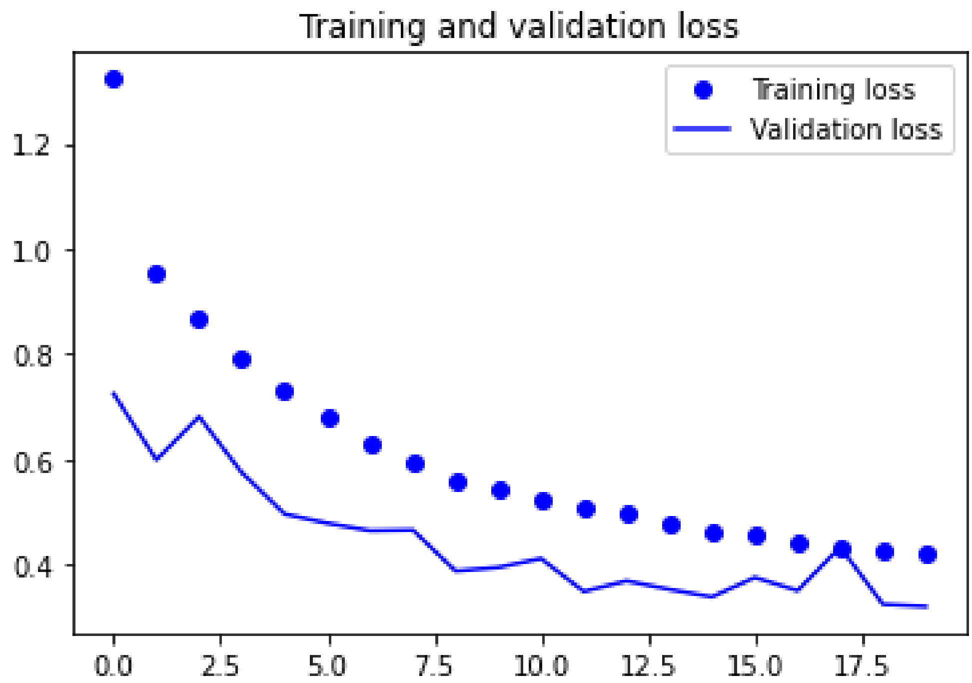
Evaluation

```
# evaluating the performance of model on test data
test_evalb = modelb.evaluate(test_X, test_Y_one_hot, verbose=0)
```

```
print('Test loss:', test_evalb[0])
print('Test accuracy:', test_evalb[1])
```

```
Test loss: 0.34032368659973145
Test accuracy: 0.8927999973487854
```





Confusion Matrix and Performance Score

```
cm = confusion_matrix(test_Y, predictionb)
#printing the confusion matrix to see corrrect and wrong predicted values
print(cm)
```

```
[[897   0   6  10   1   2  78   0   6   0]
 [  6 985   0   7   0   0   1   0   1   0]
 [105   1 844   2  25   0  21   0   2   0]
 [122  10   2 823  11   1  23   0   8   0]
 [121   0  74  30 755   0  20   0   0   0]
 [  0   0   0   0   0 994   0   6   0   0]
 [312   0  52   8  67   0 556   0   5   0]
 [  5   0   0   0   0  34   0 938   0  23]
 [  9   1   0   1   0   1   1   0 987   0]
 [  6   0   0   0   0   8   0  34   0 952]]
```

```
print("Precision Score is ", precision_score(test_Y, predictionb, average = 'weighted'))
```

```
Precision Score is  0.8894451534174008
```

```
print("Recall Score is ", recall_score(test_Y, predictionb, average = 'weighted'))
```

```
Recall Score is  0.8731
```

```
print("F1 Score is " ,f1_score(test_Y, predictionb, average = 'weighted'))
```

```
F1 Score is  0.874542856446507
```

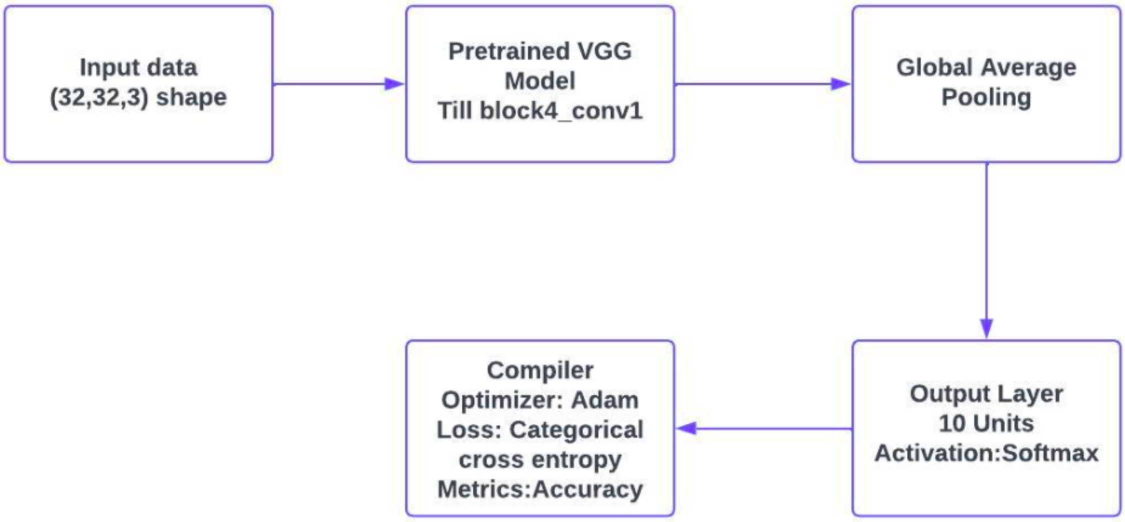
c : Transfer Learning: Load the VGG-19 model. Drop after the block4 conv1 layer (highlighted in the image below) and on top of it add one global average pooling and one final output layer. Keep the base model layers (VGG19) freeze.

Model: "vgg19"

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv4 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808

Ans:

Model Architecture



The architecture used on Fashion MNIST dataset starts by reshaping the input image to 28x28x1 and then passing it through a pre-trained VGG network. The model is only used until the conv1 layer of block4. Following this, a Global Average Pooling layer is added to reduce the dimensionality of the feature maps to 512. Finally, a dense output layer is added to predict the class label of the input image.

Hyperparameter in Transfer Learning

1) Layers frozen from pretrained model VGG-16= 13

2) Layers to tune from pretrained model = 0

3) New layers added (Global Average Pooling and Output Layer) = 2

Model Summary

```
1 modelc.summary() # summary of the model
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0         
                                                                 
 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0         
                                                                 
 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080    
                                                                 
 block3_conv4 (Conv2D)       (None, 8, 8, 256)         590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0         
                                                                 
 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160   
                                                                 
 global_average_pooling2d (G  (None, 512)              0         
 lobalAveragePooling2D)                                          
                                                                 
 dense_8 (Dense)             (None, 10)                5130      
                                                                 
=================================================================
Total params: 3,510,858
Trainable params: 5,130
Non-trainable params: 3,505,728
_________________________________________________________________
```

The model is a VGG pretrained model that has been used for classification. It consists of an input layer that takes in an image of shape (32,32,3). The input layer is followed by 2 convolutional layers (block1_conv1 and block1_conv2) with 64 filters each, and a max pooling layer (block1_pool) that reduces the feature map size by half.

Next, there are 2 more convolutional layers (block2_conv1 and block2_conv2) with 128 filters each, followed by another max pooling layer (block2_pool) that reduces the feature map size by half again. The next 4 convolutional layers (block3_conv1, block3_conv2, block3_conv3, and block3_conv4) have 256 filters each, and are followed by a max pooling layer (block3_pool) that reduces the feature map size by half again.

The final convolutional layer (block4_conv1) has 512 filters, and is followed by a global average pooling layer that averages the feature maps to a single vector. This is then followed by a dense layer (dense_8) with 10 units for classification.

The model has a total of 3,510,858 parameters, but only 5,130 of them are trainable, as the convolutional layers are pretrained and frozen.

```
end_timec = time.time()
```

```
total_timec = end_timec - start_timec
```

```
print("Total execution time: ", total_timec, " seconds")
```

```
Total execution time:  121.30633425712585  seconds
```

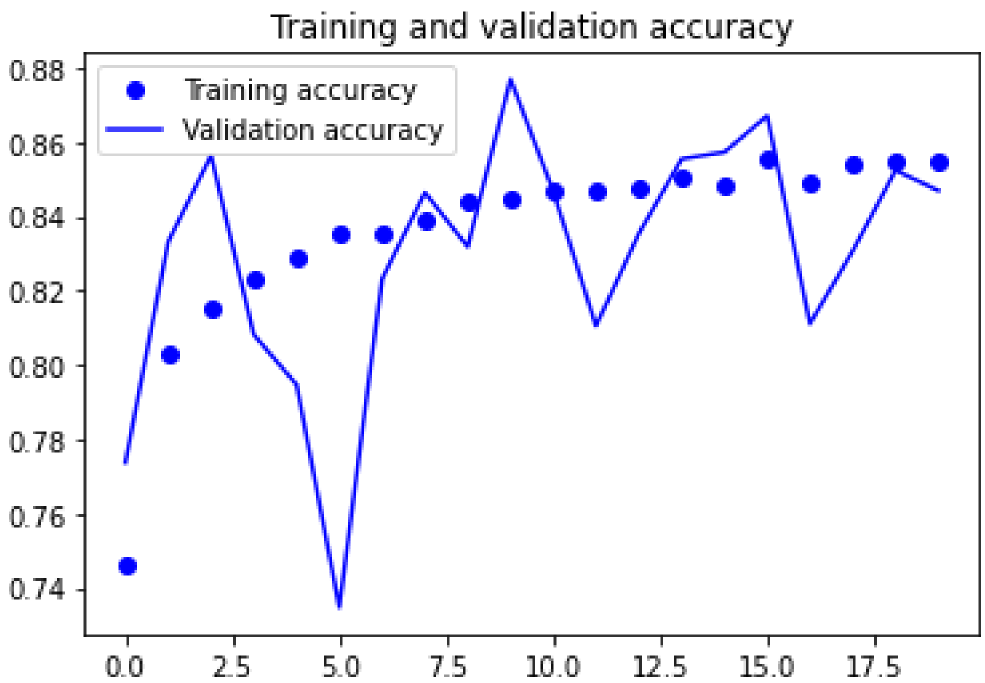
Evaluating the Model

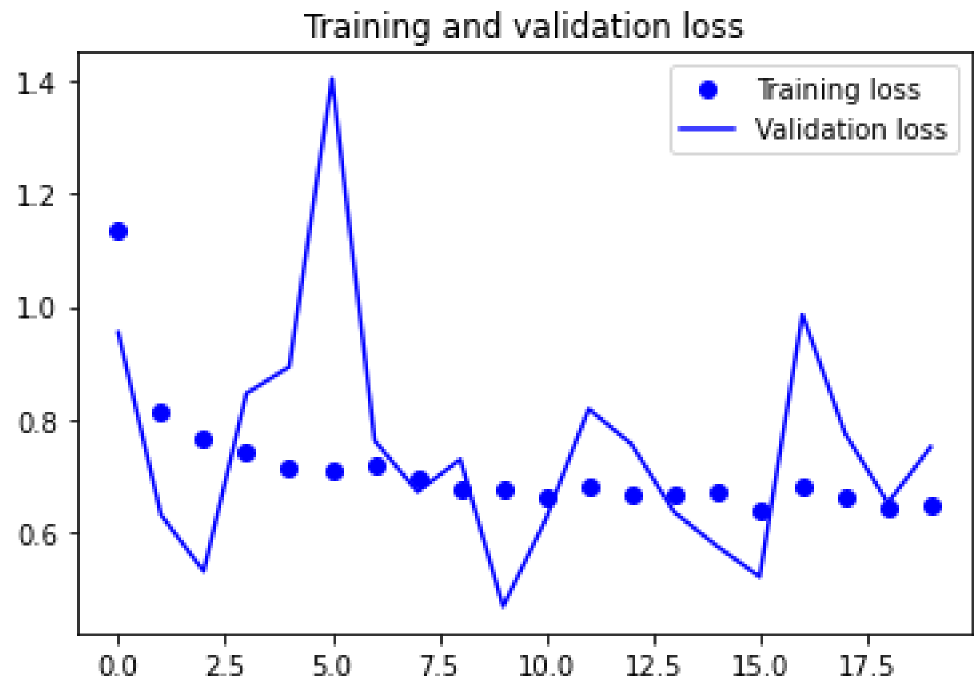
```
# evaluating the performance of model in test data

test_evalc = modelc.evaluate(test_images, test_Y_one_hot, verbose=0)
```

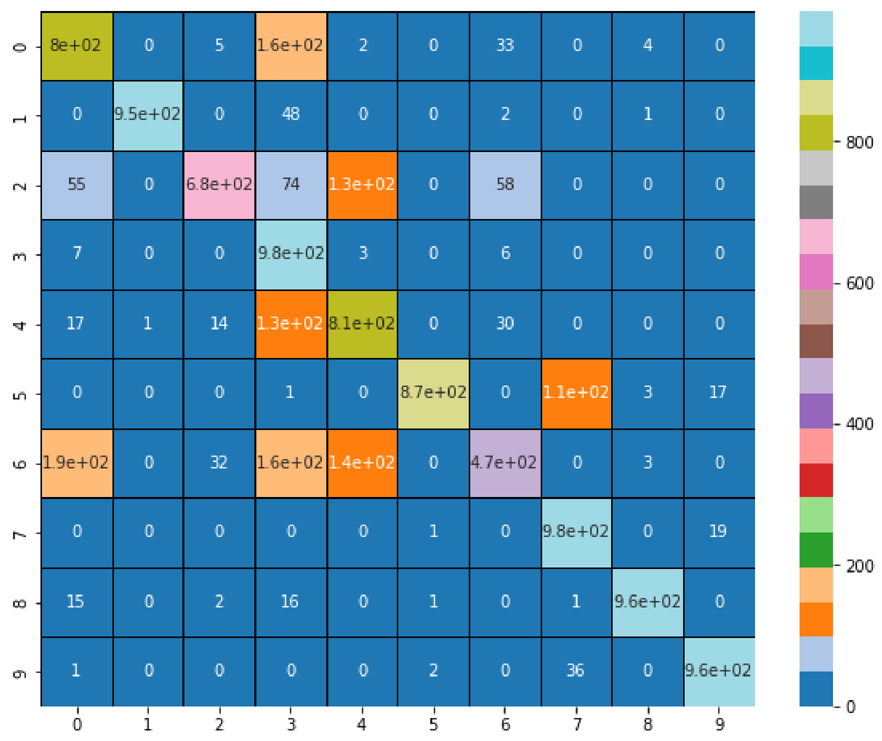
```
print('Test loss:', test_evalc[0])
print('Test accuracy:', test_evalc[1])
```

```
Test loss: 0.7428037524223328
Test accuracy: 0.8489000201225281
```





Confusion Matrix



```
1 print("Precision Score is ", precision_score(test_Y,  predictionc, average = 'weighted'))
```

```
Precision Score is  0.8638325170358898
```

```
1 print("Recall Score is ", recall_score(test_Y,  predictionc, average = 'weighted'))
```

```
Recall Score is  0.847
```

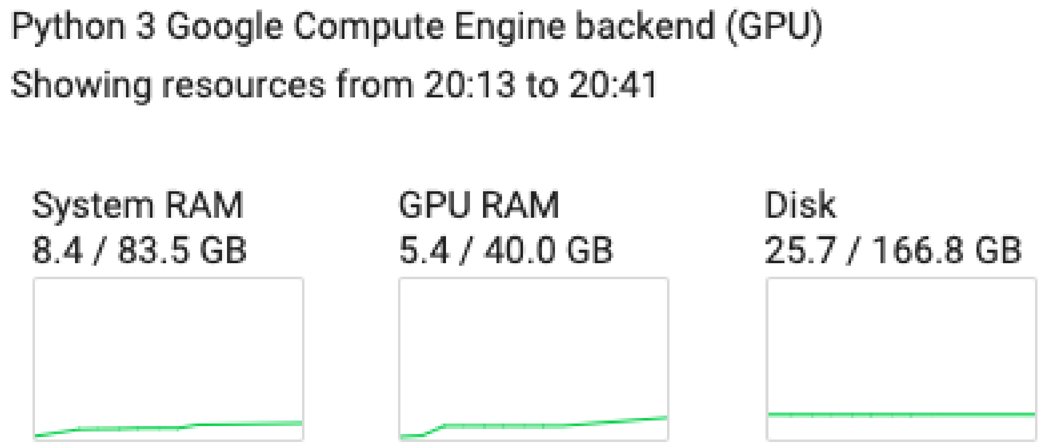
```
1 print("F1 Score is ", f1_score(test_Y,  predictionc, average = 'weighted'))
```

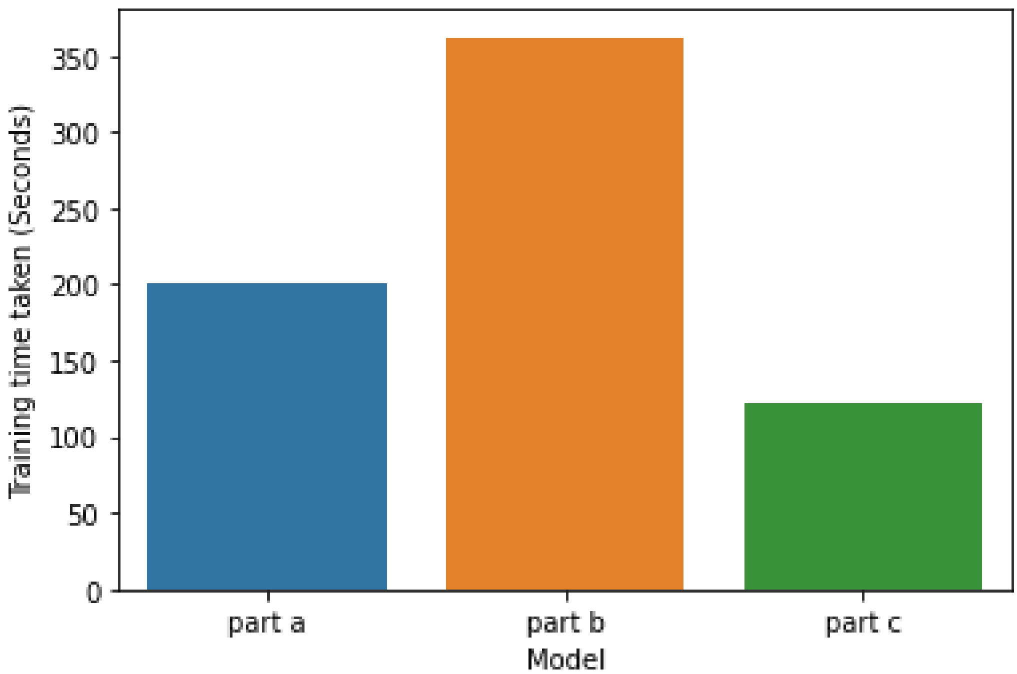
```
F1 Score is  0.8445080875402742
```

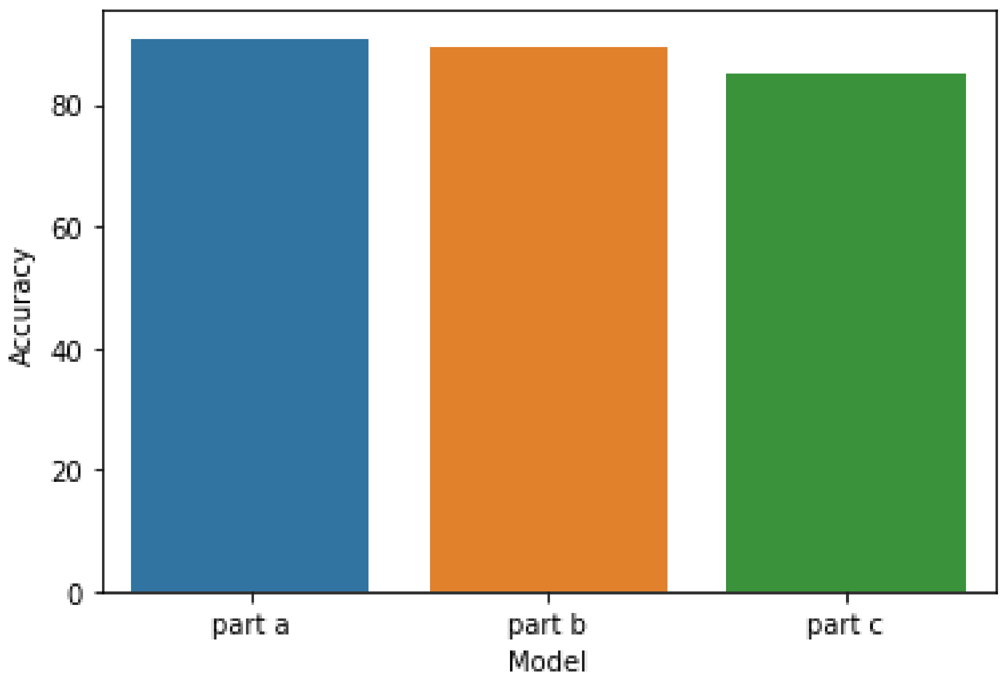
d: Make a comparison table including the above three model's performance on the test data (accuracy, number of trainable parameters and execution time).

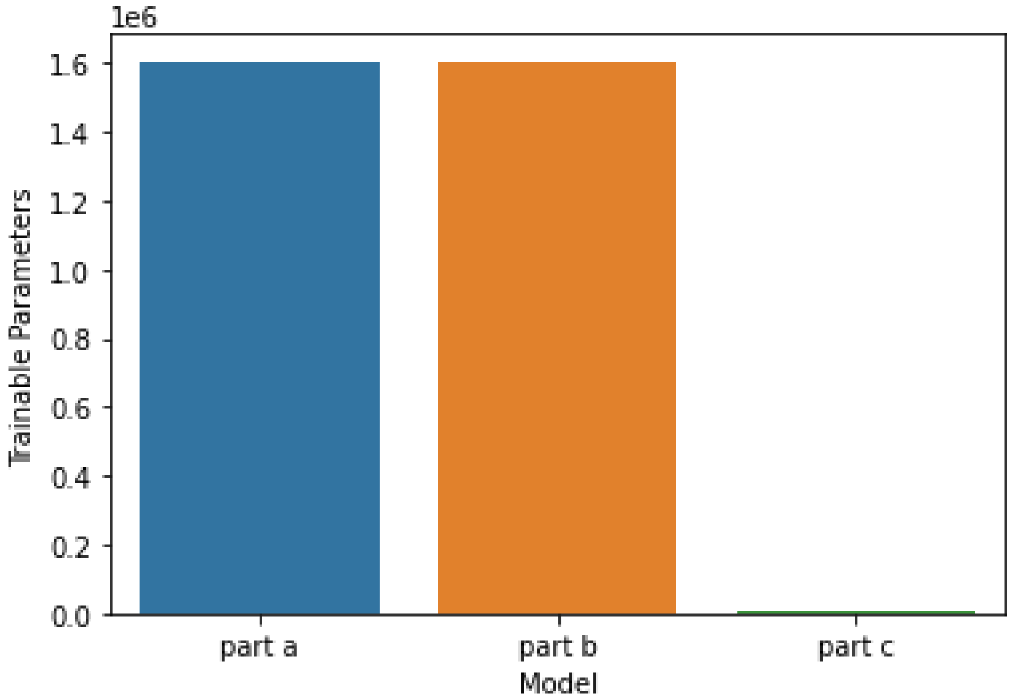
Ans: All the model are trained for batch size = 32 and epoch = 20

Ques No.	Time Taken in Training	Number of Trainable Parameters	Accuracy	F1 score	Recall	Precision
Ques 1 a	200.49 Seconds	1605226	90.92%	90.79%	90.77%	91.08%
Ques 1 b	362.29 Seconds	1605226	89.27%	87.45%	87.31%	88.94%
Ques 1 c	121.30 Seconds	5130	84.89%	84.45%	84.70%	86.38%




Training time taken (Seconds)
0
50
100
150
200
250
300
350
part a
part b
part c
Model


Accuracy
0
20
40
60
80
part a
part b
part c
Model



Results

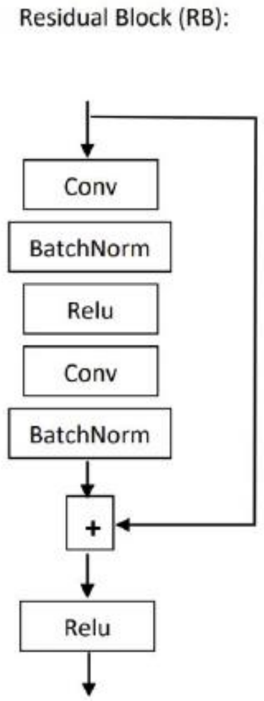
Model was comparison was done on the basis of accuracy , training time taken and trainable parameters for all three models ie. simple convolution model (part a) , convolution model with data augmentation (part b) and pretrained vgg model (part c). Trainable parameters was less for part c i.e 5130 model where other two models have 1605226 trainable parameters. Accuracy is more for part a i.e simple convolution model (90.92%) whereas other two models have accuracy 89% and 84% respectively. Training time taken was highest for part b where data augmentation was done on the fly i.e. it took around 363 seconds whereas for part c (transfer learning) training time was less ie. it took around 122 seconds. With reference to accuracy, part a has highest accuracy ie. 90% whereas accuracy was comparatively less for partb and part c.

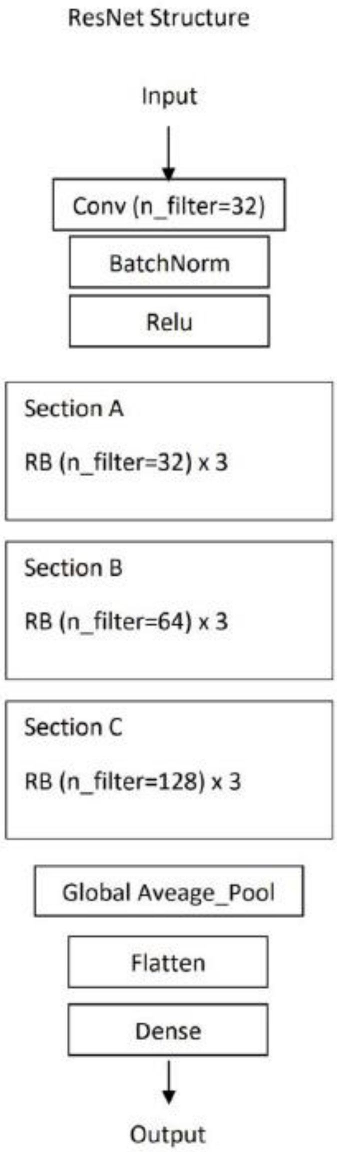
Ques 2: Developing ResNet model from scratch
Apply a residual network specified in the following architecture. All convolutional layers use kernel size 3, stride = 1, and padding = “same”.
Ans:

Link for Saved Model is given below.

https://drive.google.com/drive/u/2/folders/1anKLUX8RfCPfxEoU0_P16baZzBksvP3g

Model Architecture





The ResNet model architecture consists of the following layers:

Input layer: A 3D tensor of shape (32, 32, 3) representing the input image.

Conv2D layer: A 2D convolutional layer with 32 filters, kernel size of 3, and padding of "same".

Batch Normalization layer: Normalizes the activations of the previous convolutional layer.

Activation layer: Applies the ReLU activation function to the previous layer.

Residual unit: Consists of two 2D convolutional layers with 32 filters, kernel size of 3, and padding of "same", followed by Batch Normalization layers and ReLU activation functions. Concatenates the output of the first layer with the shortcut connection and applies the ReLU activation function.

Residual units: Two more residual units consisting of convolutional, Batch Normalization, and activation layers with 64 and 128 filters, respectively.

Each Residual Block is repeated thrice so that the model can learn more features

Global Average Pooling 2D layer: Computes the spatial average of the previous layer's output across all channels.

Dense layer: A fully connected layer with 10 neurons, representing the output classes.

Softmax activation layer: Applies the softmax function to the output of the previous layer.

Hyperparameter Used

Number of Convolution Layers: 1					
Convolutions Layers	**Filter Size**	**Stride**	**Number of Filters**	**Padding**	**Activation Function**
Convo Layer 1	3x3	1	32	Same	Relu
RSU Block 1					
Convolutions Layers	**Filter Size**	**Stride**	**Number of Filters**	**Padding**	**Activation Function**
Convo Layer 1	3x3	1	32	Same	Relu
Convo Layer 1	3x3	1	32	Same	Relu
RSU Block 2					
Convolutions Layers	**Filter Size**	**Stride**	**Filters**	**Padding**	**Function**
Convo Layer 1	3x3	1	64	Same	Relu
Convo Layer 1	3x3	1	64	Same	Relu
RSU Block 3					
Convolutions Layers	**Filter Size**	**Stride**	**Filters**	**Padding**	**Function**
Convo Layer 1	3x3	1	128	Same	Relu
Convo Layer 1	3x3	1	128	Same	Relu
Output Layer					
Output Layer	**Neurons**			**Activation Function**	
	10			Softmax	
Categorical Cross	Adam	0.001	Accuracy		
Training					
Epochs	**Batch Size**			**Validation Split**	
10	32			0.2	
Weight Initialiser		**Bias Initialiser**			
Be default Glorot		By default zeros			

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 32, 32, 3)]  0           []

 conv2d_10 (Conv2D)             (None, 32, 32, 32)   896         ['input_2[0][0]']

 batch_normalization_10 (BatchN  (None, 32, 32, 32)  128         ['conv2d_10[0][0]']
 ormalization)

 activation (Activation)        (None, 32, 32, 32)   0           ['batch_normalization_10[0][0]']

 conv2d_11 (Conv2D)             (None, 32, 32, 32)   9248        ['activation[0][0]']

 batch_normalization_11 (BatchN  (None, 32, 32, 32)  128         ['conv2d_11[0][0]']
 ormalization)
activation_1 (Activation)      (None, 32, 32, 32)   0           ['batch_normalization_11[0][0]']

conv2d_12 (Conv2D)             (None, 32, 32, 32)   9248        ['activation_1[0][0]']

batch_normalization_12 (BatchN  (None, 32, 32, 32)  128         ['conv2d_12[0][0]']
ormalization)

concatenate (Concatenate)      (None, 32, 32, 64)   0           ['activation[0][0]',
                                                                 'batch_normalization_12[0][0]']

activation_2 (Activation)      (None, 32, 32, 64)   0           ['concatenate[0][0]']

conv2d_13 (Conv2D)             (None, 32, 32, 32)   18464       ['activation_2[0][0]']

batch_normalization_13 (BatchN  (None, 32, 32, 32)  128         ['conv2d_13[0][0]']
ormalization)

activation_3 (Activation)      (None, 32, 32, 32)   0           ['batch_normalization_13[0][0]']
conv2d_15 (Conv2D)             (None, 32, 32, 32)   2080        ['activation_2[0][0]']

conv2d_14 (Conv2D)             (None, 32, 32, 32)   9248        ['activation_3[0][0]']

batch_normalization_15 (BatchN  (None, 32, 32, 32)  128         ['conv2d_15[0][0]']
ormalization)

batch_normalization_14 (BatchN  (None, 32, 32, 32)  128         ['conv2d_14[0][0]']
ormalization)

concatenate_1 (Concatenate)    (None, 32, 32, 64)   0           ['batch_normalization_15[0][0]',
                                                                 'batch_normalization_14[0][0]']

activation_4 (Activation)      (None, 32, 32, 64)   0           ['concatenate_1[0][0]']

conv2d_16 (Conv2D)             (None, 32, 32, 32)   18464       ['activation_4[0][0]']

batch_normalization_16 (BatchN  (None, 32, 32, 32)  128         ['conv2d_16[0][0]']
ormalization)
activation_5 (Activation)      (None, 32, 32, 32)   0           ['batch_normalization_16[0][0]']

conv2d_18 (Conv2D)             (None, 32, 32, 32)   2080        ['activation_4[0][0]']

conv2d_17 (Conv2D)             (None, 32, 32, 32)   9248        ['activation_5[0][0]']

batch_normalization_18 (BatchN  (None, 32, 32, 32)  128         ['conv2d_18[0][0]']
ormalization)

batch_normalization_17 (BatchN  (None, 32, 32, 32)  128         ['conv2d_17[0][0]']
ormalization)

concatenate_2 (Concatenate)    (None, 32, 32, 64)   0           ['batch_normalization_18[0][0]',
                                                                 'batch_normalization_17[0][0]']

activation_6 (Activation)      (None, 32, 32, 64)   0           ['concatenate_2[0][0]']

conv2d_19 (Conv2D)             (None, 32, 32, 64)   36928       ['activation_6[0][0]']
```

```
batch_normalization_19 (BatchN  (None, 32, 32, 64)  256         ['conv2d_19[0][0]']
ormalization)

activation_7 (Activation)      (None, 32, 32, 64)   0           ['batch_normalization_19[0][0]']

conv2d_20 (Conv2D)             (None, 32, 32, 64)   36928       ['activation_7[0][0]']

batch_normalization_20 (BatchN  (None, 32, 32, 64)  256         ['conv2d_20[0][0]']
ormalization)

concatenate_3 (Concatenate)    (None, 32, 32, 128)  0           ['activation_6[0][0]',
                                                                 'batch_normalization_20[0][0]']

activation_8 (Activation)      (None, 32, 32, 128)  0           ['concatenate_3[0][0]']

conv2d_21 (Conv2D)             (None, 32, 32, 64)   73792       ['activation_8[0][0]']

batch_normalization_21 (BatchN  (None, 32, 32, 64)  256         ['conv2d_21[0][0]']
ormalization)
```

```
activation_9 (Activation)      (None, 32, 32, 64)   0           ['batch_normalization_21[0][0]']

conv2d_23 (Conv2D)             (None, 32, 32, 64)   8256        ['activation_8[0][0]']

conv2d_22 (Conv2D)             (None, 32, 32, 64)   36928       ['activation_9[0][0]']

batch_normalization_23 (BatchN  (None, 32, 32, 64)  256         ['conv2d_23[0][0]']
ormalization)

batch_normalization_22 (BatchN  (None, 32, 32, 64)  256         ['conv2d_22[0][0]']
ormalization)

concatenate_4 (Concatenate)    (None, 32, 32, 128)  0           ['batch_normalization_23[0][0]',
                                                                 'batch_normalization_22[0][0]']

activation_10 (Activation)     (None, 32, 32, 128)  0           ['concatenate_4[0][0]']

conv2d_24 (Conv2D)             (None, 32, 32, 64)   73792       ['activation_10[0][0]']
```

```
batch_normalization_24 (BatchN  (None, 32, 32, 64)  256         ['conv2d_24[0][0]']
ormalization)

activation_11 (Activation)     (None, 32, 32, 64)   0           ['batch_normalization_24[0][0]']

conv2d_26 (Conv2D)             (None, 32, 32, 64)   8256        ['activation_10[0][0]']

conv2d_25 (Conv2D)             (None, 32, 32, 64)   36928       ['activation_11[0][0]']

batch_normalization_26 (BatchN  (None, 32, 32, 64)  256         ['conv2d_26[0][0]']
ormalization)

batch_normalization_25 (BatchN  (None, 32, 32, 64)  256         ['conv2d_25[0][0]']
ormalization)

concatenate_5 (Concatenate)    (None, 32, 32, 128)  0           ['batch_normalization_26[0][0]',
                                                                 'batch_normalization_25[0][0]']

activation_12 (Activation)     (None, 32, 32, 128)  0           ['concatenate_5[0][0]']
```

```
 conv2d_27 (Conv2D)             (None, 32, 32, 128)  147584      ['activation_12[0][0]']

 batch_normalization_27 (BatchN  (None, 32, 32, 128)  512        ['conv2d_27[0][0]']
 ormalization)

 activation_13 (Activation)     (None, 32, 32, 128)  0           ['batch_normalization_27[0][0]']

 conv2d_28 (Conv2D)             (None, 32, 32, 128)  147584      ['activation_13[0][0]']

 batch_normalization_28 (BatchN  (None, 32, 32, 128)  512        ['conv2d_28[0][0]']
 ormalization)

 concatenate_6 (Concatenate)    (None, 32, 32, 256)  0           ['activation_12[0][0]',
                                                                  'batch_normalization_28[0][0]']

 activation_14 (Activation)     (None, 32, 32, 256)  0           ['concatenate_6[0][0]']

 conv2d_29 (Conv2D)             (None, 32, 32, 128)  295040      ['activation_14[0][0]']

 batch_normalization_29 (BatchN  (None, 32, 32, 128)  512        ['conv2d_29[0][0]']
 ormalization)

 activation_15 (Activation)     (None, 32, 32, 128)  0           ['batch_normalization_29[0][0]']

 conv2d_31 (Conv2D)             (None, 32, 32, 128)  32896       ['activation_14[0][0]']

 conv2d_30 (Conv2D)             (None, 32, 32, 128)  147584      ['activation_15[0][0]']

 batch_normalization_31 (BatchN  (None, 32, 32, 128)  512        ['conv2d_31[0][0]']
 ormalization)

 batch_normalization_30 (BatchN  (None, 32, 32, 128)  512        ['conv2d_30[0][0]']
 ormalization)

 concatenate_7 (Concatenate)    (None, 32, 32, 256)  0           ['batch_normalization_31[0][0]',
                                                                  'batch_normalization_30[0][0]']

 activation_16 (Activation)     (None, 32, 32, 256)  0           ['concatenate_7[0][0]']

 conv2d_32 (Conv2D)             (None, 32, 32, 128)  295040      ['activation_16[0][0]']

 batch_normalization_32 (BatchN  (None, 32, 32, 128)  512        ['conv2d_32[0][0]']
 ormalization)

 activation_17 (Activation)     (None, 32, 32, 128)  0           ['batch_normalization_32[0][0]']

 conv2d_34 (Conv2D)             (None, 32, 32, 128)  32896       ['activation_16[0][0]']

 conv2d_33 (Conv2D)             (None, 32, 32, 128)  147584      ['activation_17[0][0]']

 batch_normalization_34 (BatchN  (None, 32, 32, 128)  512        ['conv2d_34[0][0]']
 ormalization)

 batch_normalization_33 (BatchN  (None, 32, 32, 128)  512        ['conv2d_33[0][0]']
 ormalization)

 concatenate_8 (Concatenate)    (None, 32, 32, 256)  0           ['batch_normalization_34[0][0]',
                                                                  'batch_normalization_33[0][0]']

 activation_18 (Activation)     (None, 32, 32, 256)  0           ['concatenate_8[0][0]']

 global_average_pooling2d_1 (Gl  (None, 256)         0           ['activation_18[0][0]']
 obalAveragePooling2D)

 dense_7 (Dense)                (None, 10)           2570        ['global_average_pooling2d_1[0][0
                                                                 ]']

==================================================================================================
Total params: 1,646,858
Trainable params: 1,643,210
Non-trainable params: 3,648
```

This is a custom-built RESNET model (convolutional neural network model with skip connections). It takes an input of size (32, 32, 3) and has a total of 392,906 parameters, out of which 391,946 are trainable and 960 are non-trainable.

The model has several convolutional layers with batch normalization and activation functions, and skip connections are added between them. It ends with a global average pooling layer followed by a dense layer with 10 units, representing the 10 classes in the dataset.

Overall, the model aims to learn hierarchical representations of the input images through the convolutional layers and use the skip connections to preserve important features from the earlier layers. Finally, the global average pooling layer condenses the learned representations into a fixed-length vector, which is fed to the output dense layer for classification.

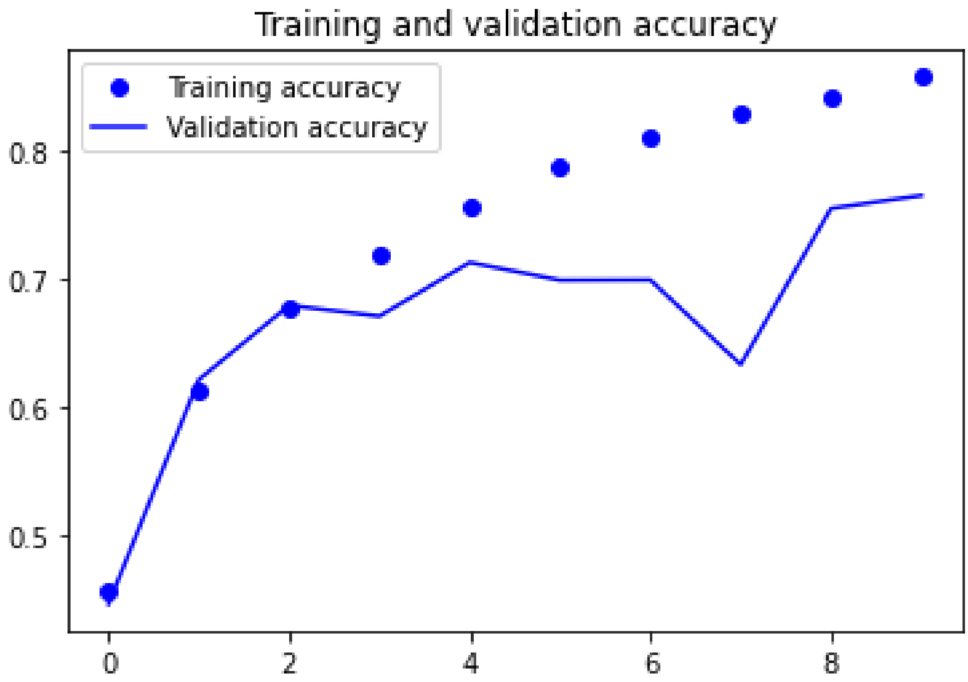
Evaluating the Model

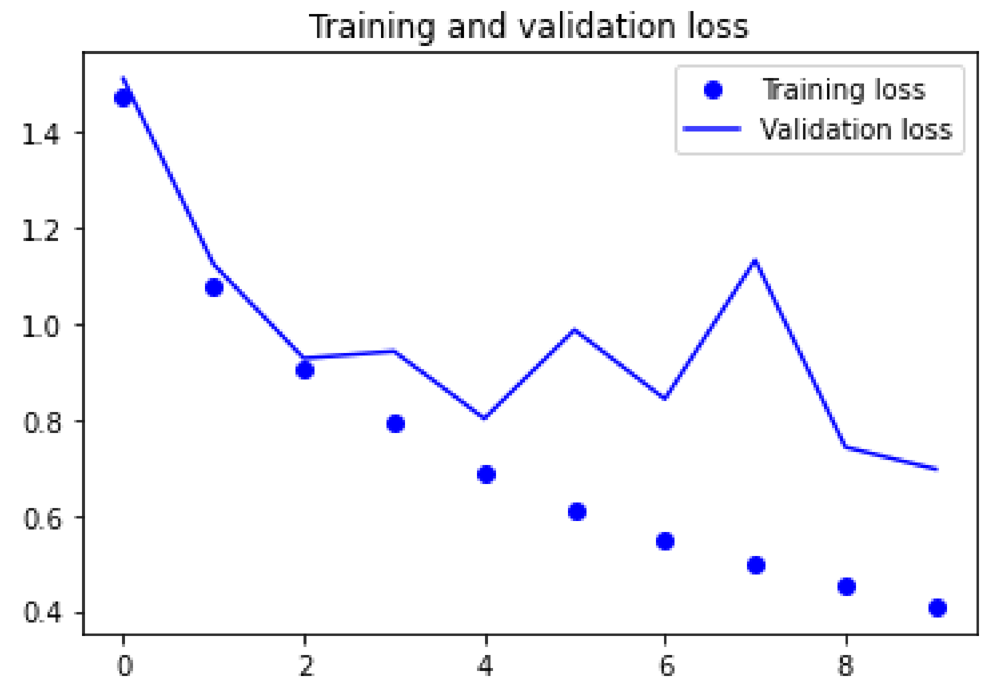
```
# evaluating the performance of model in test data

test_evald = modeld.evaluate(x_testcifar,y_testcifar_one_hot, verbose=0)
```

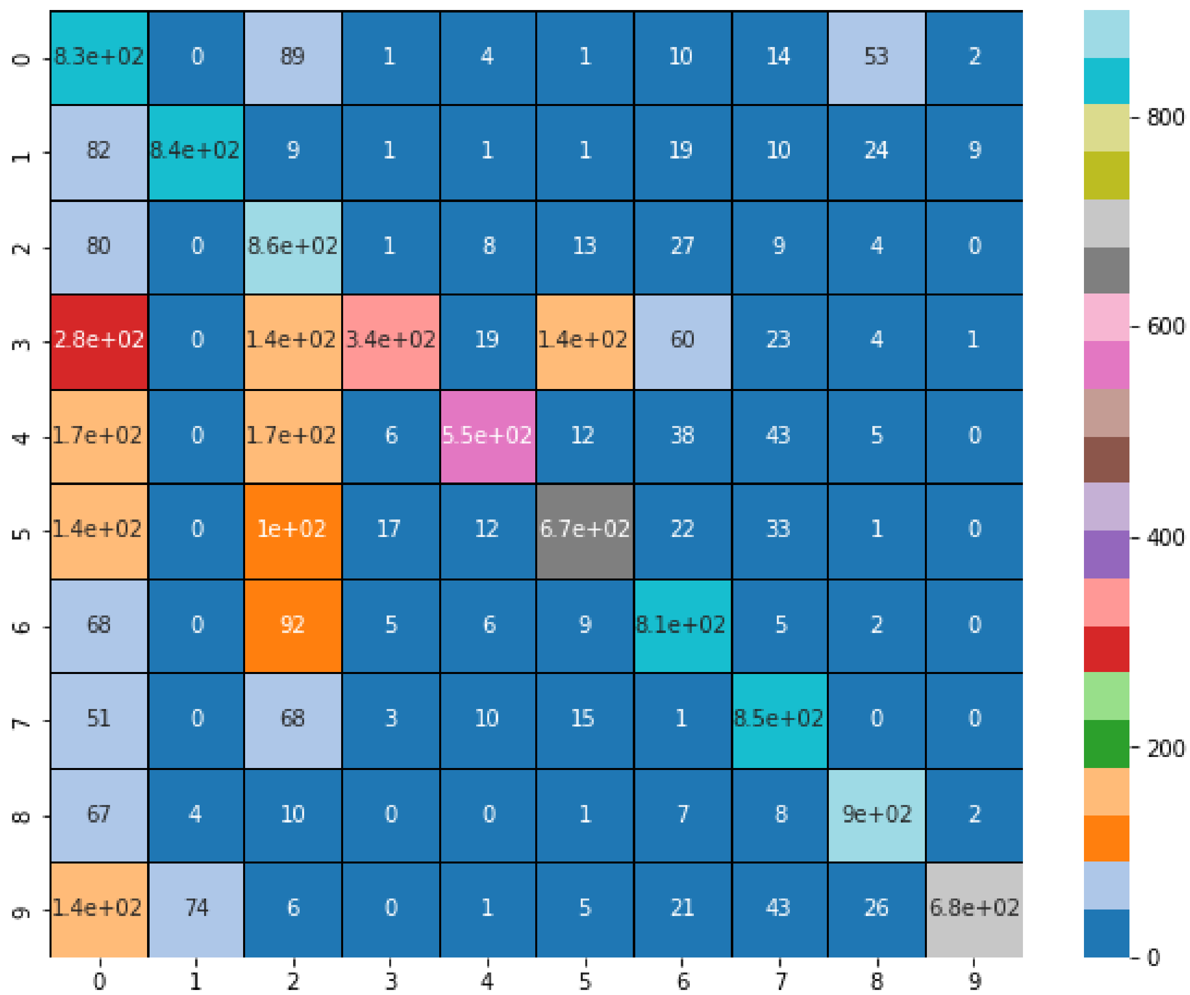
```
print('Test loss:', test_evald[0])
print('Test accuracy:', test_evald[1])
```

```
Test loss: 0.6992190480232239
Test accuracy: 0.763700008392334
```





Confusion Matrix



```
print("Precision Score is ", precision_score(y_testcifar, predictiond, average = 'weighted'))
```

```
Precision Score is  0.7970126970511039
```

```
print("Recall Score is ", recall_score(y_testcifar, predictiond, average = 'weighted'))
```

```
Recall Score is  0.7339
```

```
print("F1 Score is ", f1_score(y_testcifar, predictiond, average = 'weighted'))
```

```
F1 Score is  0.735701149804657
```

Results

RESNET model has a total of 1,646,858 parameters, out of which 1,643,210 are trainable and 3,648 are non-trainable.Model was trained for 10 epochs and accuracy for the model was 70%. Recall and Precision score for the model is less as compared to above three models. Recall and Precision score is 73% and 79% respectively. This may be due to the complexity of the model as the learning curve shows that training accracy is more than validation accuracy. Confusion matrix, recall and precision score is given above.