ECE 657A Assignment 3

Submitted By :
Amandeep Kaur (21044104)
Bhupesh Dod (21046099)

Question 2 : Own Network

Solution :

To run the CNNs, we used Google Colab and Keras Library. We used Google Colab GPU to run the CNNs for faster computations. The steps followed to create and implement the own CNN netowk are as follows:

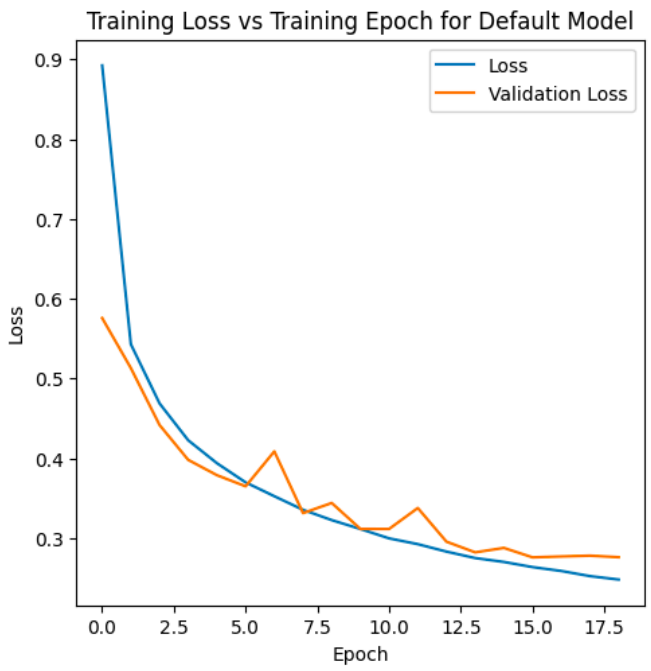
1. Load all the necessary libraries.
2. Import the Datasets.
3. Data pre-processing. : Reshape the data to 28*28
4. Create compile and summarize the CNN network.
5. Fit the CNN network.
6. Calculate Accuracy and Loss.

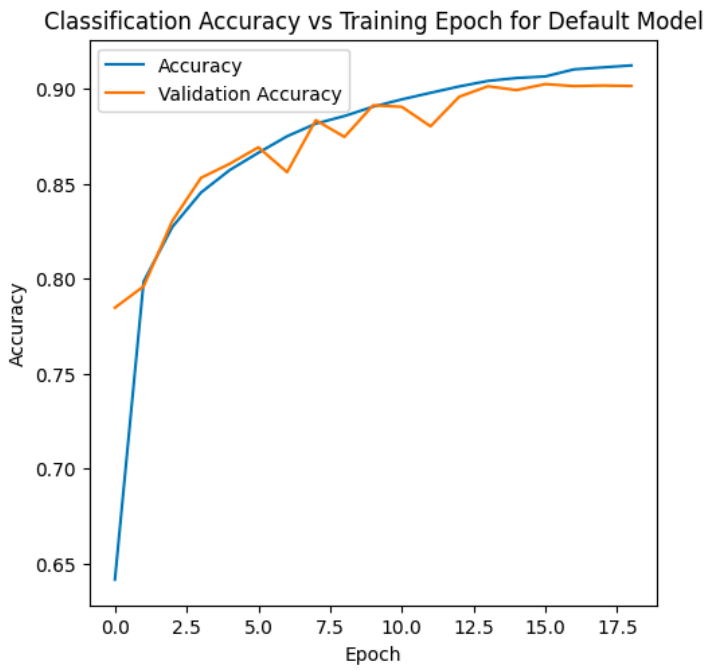
To make the best possible CNN model we tried different strategies.

Default Model:

We first started with a basic model of 2 CNN layers with 32 kernels, 3*3 filter, 1 Stride and same padding, a Maxpool layer of pool_size 2*2, a flatten function and 2 fully connected layers. The hidden layers have activation function Relu and the output layer has softmax as the activation function and optimizer as Stochastic Gradient Descent.

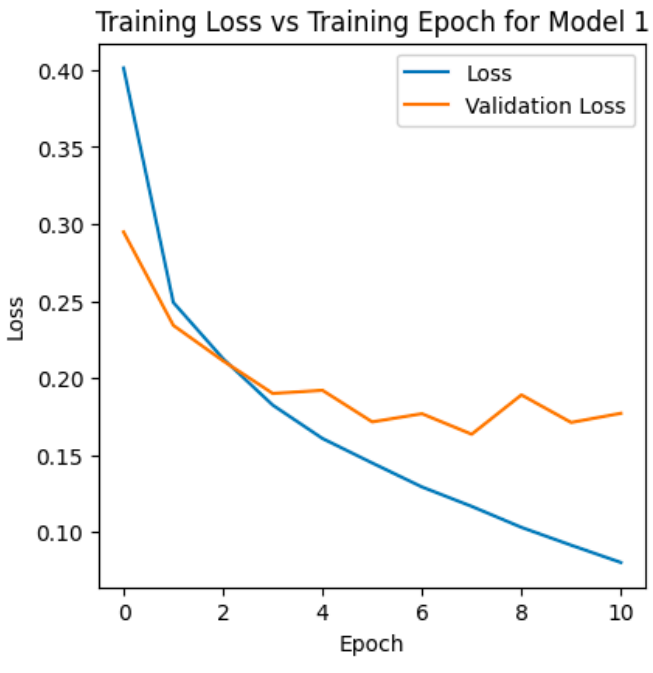
Fit the model using EarlyStopping and with 100 epochs and 128 batch size. The accuracy of this model came out to be 89.68% and loss 29.20%.

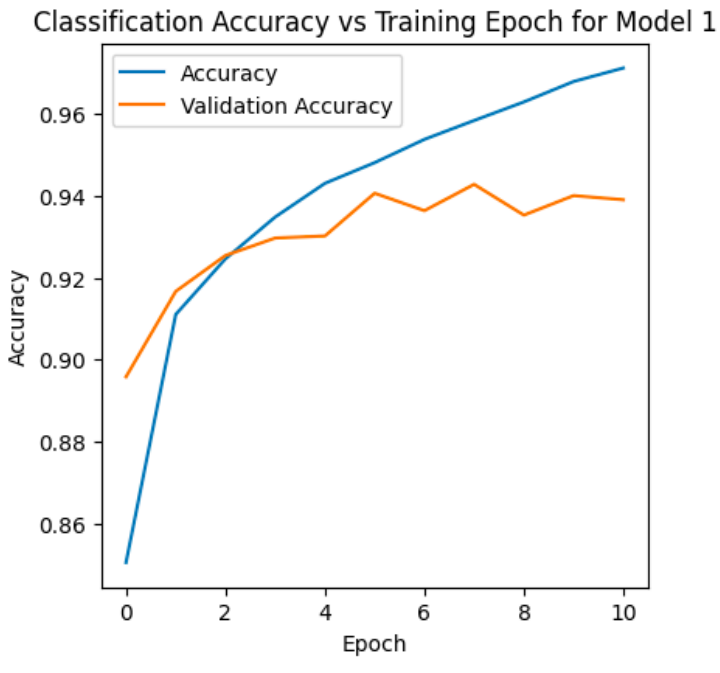




Model 1:

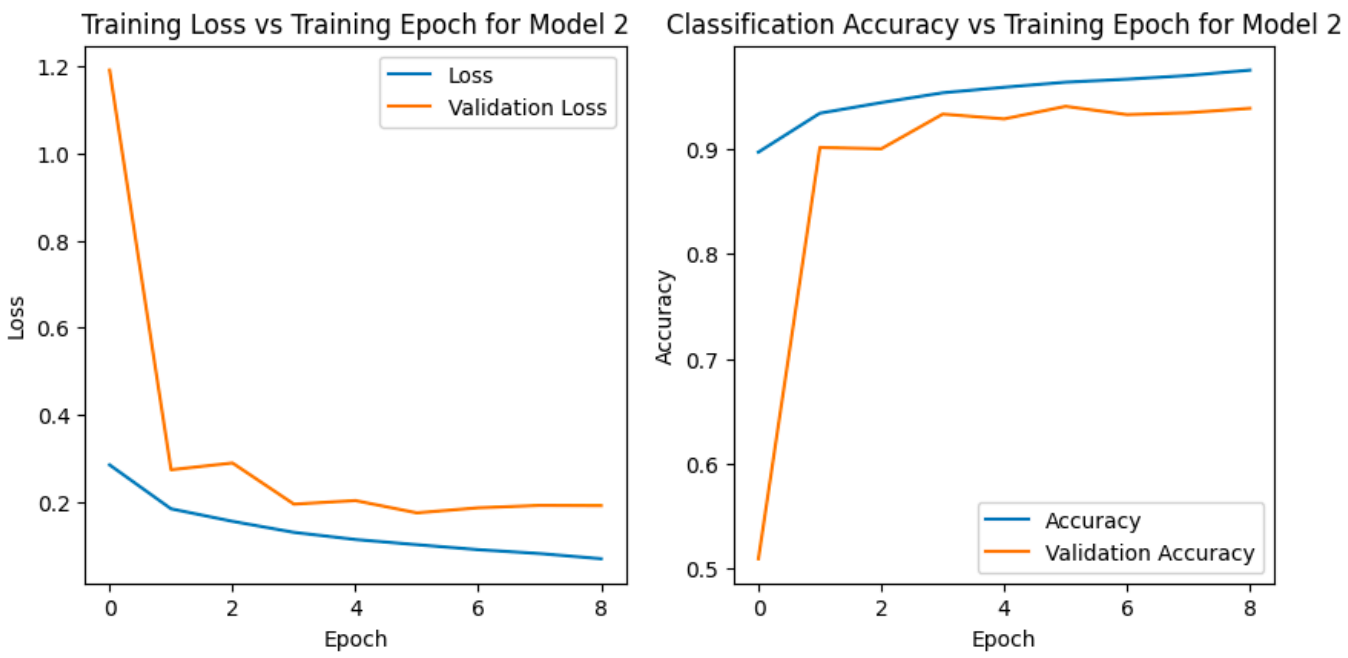
For this model, we changed the optimizer Adam as it is a widely used optimizer in CNNs, rest all the parameters are same as Default Model. The accuracy of this model increased to 93.70% and loss has decreased to 19.51%. Although the performance of this model has improved but the model is overfitting as can be inferred from learning loss curve (validation loss > training loss), so to avoid overfitting we add batch normalization and max pooling layer after second convolution layer.





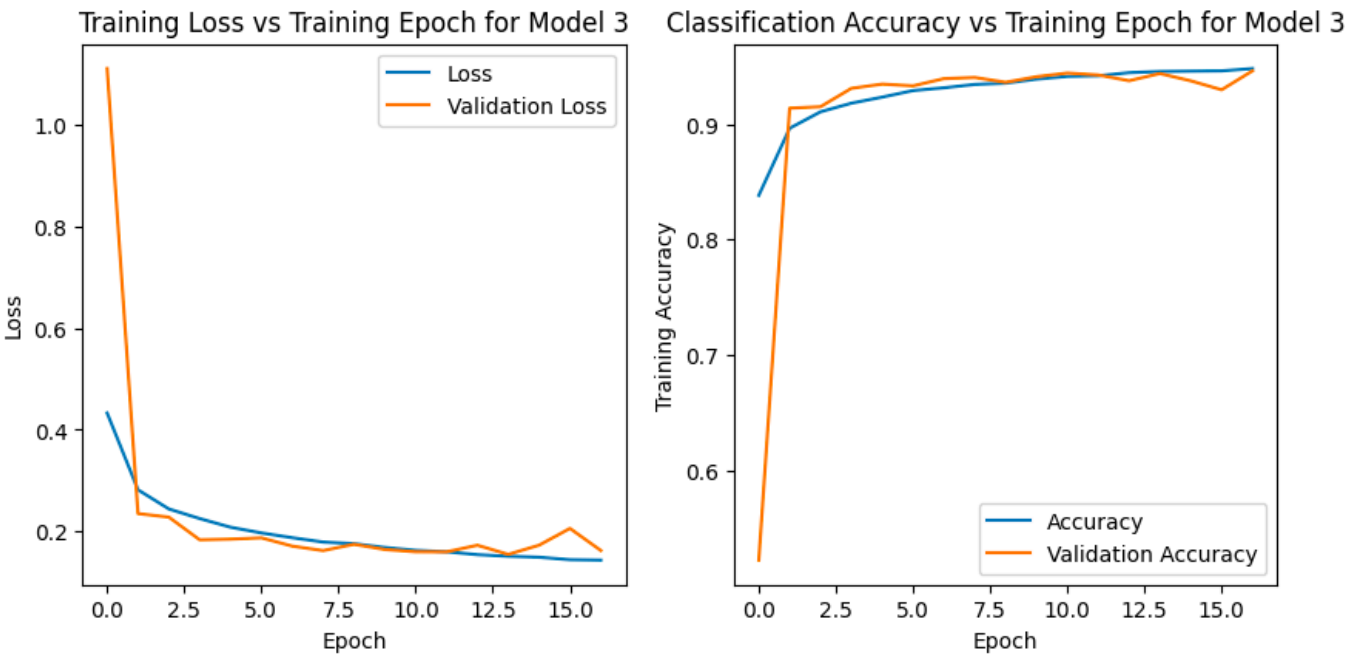
Model 2:

For this model, we add batch normalization and max pool layers to avoid overfitting of the model and reduce the validation loss. The accuracy of Model 2 is 93.52%, which is not an improvement from Model 1 and also the issue of overfitting is not resolved as it can be seen from the graphs below. So, to reduce the overfitting, we introduce the dropout layers.



Model 3:

This model adds the dropout layers on top of Model 2. By doing this, the problem of overfitting is resolved to a much extent. Also, the acuuracy has improved from Model 2 and has increased to 94.02% and the loss is reduced to 17.48%.



Now, let's build a new, better model. As previously stated, the sole addition of dropout layers could improve the test accuracy to 94.02%. There is still room to build a more robust CNN. After a few hits and trials, we include a third convolutional layer. Let's check the performance of this new model.

Own custom model

The CNN layers of this network are :

a) CNN layer with filters = 32, kernels = 3*3, strides = 1 and padding = 'same' with RelU activation function.
b) Perform BatchNormalization.
c) MaxPool layer with pool size = (2,2)
d) Dropout 25% of neuron's.
e) CNN layer with filters = 64, kernels = 3*3, strides = 1 and padding = 'same' with RelU activation function.
f) Perform BatchNormalization.
g) MaxPool layer with pool size = (2,2)

h) Dropout 25% of neuron's.
i) CNN layer with filters = 32, kernels = 3*3, strides = 1 and padding = 'same' with RelU activation function.
j) Perform BatchNormalization.
k) Dropout 25% of the neuron's.
l) Flatten layer
m) Fully connected layer with 64 neuron's and RelU activation function.
n) Fully connected output layer with 5 outputs.

The output layer used 'Softmax' activation function as it normalizes the output to a probability distribution over output clusters. We used Adam as the optimizer and loss as sparse categorical cross-entropy. The structure of the CNN is shown in Fig.1

Fig.1 Custom CNN structure

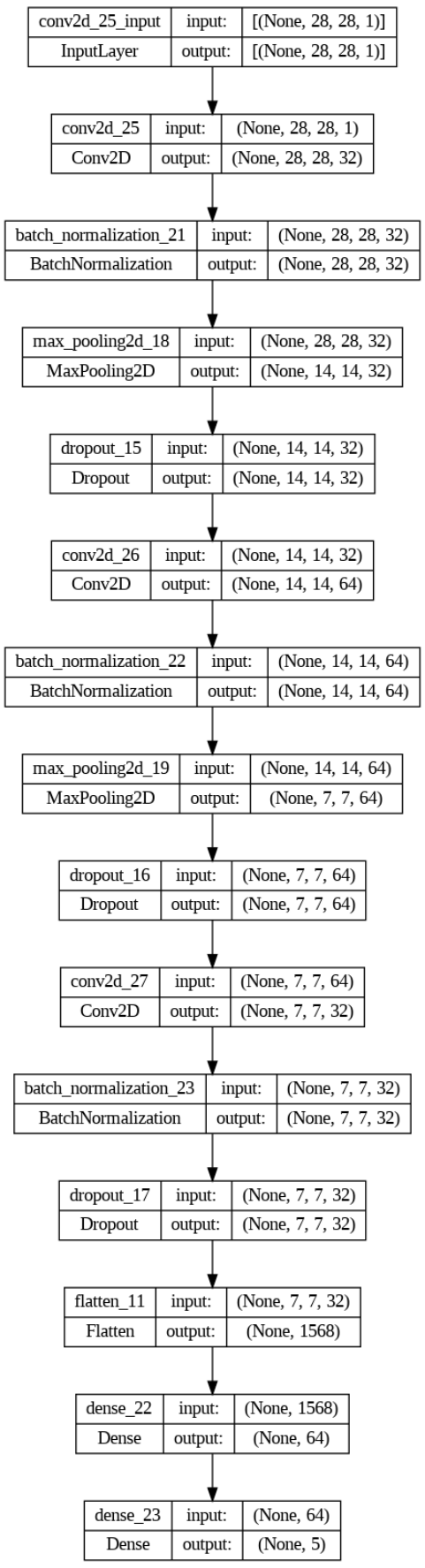


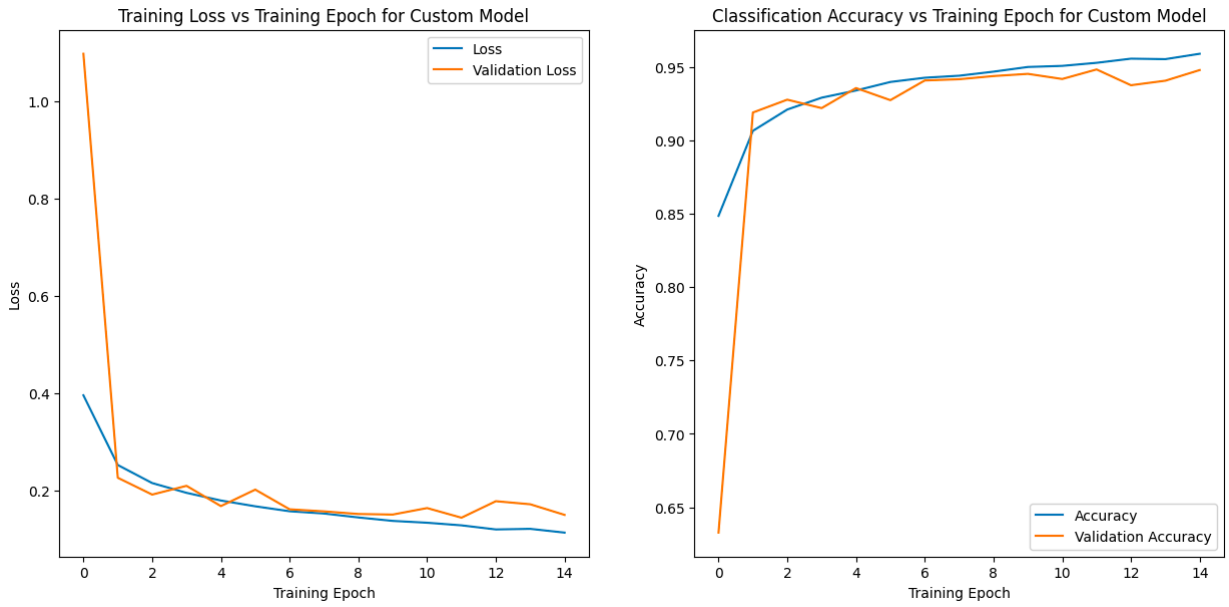
Fig.2 Custom CNN model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 28, 28, 32)        320

batch_normalization_21 (Bat  (None, 28, 28, 32)        128
chNormalization)

max_pooling2d_18 (MaxPoolin  (None, 14, 14, 32)        0
g2D)

dropout_15 (Dropout)         (None, 14, 14, 32)        0

conv2d_26 (Conv2D)           (None, 14, 14, 64)        18496

batch_normalization_22 (Bat  (None, 14, 14, 64)        256
chNormalization)

max_pooling2d_19 (MaxPoolin  (None, 7, 7, 64)          0
g2D)

dropout_16 (Dropout)         (None, 7, 7, 64)          0

conv2d_27 (Conv2D)           (None, 7, 7, 32)          18464

batch_normalization_23 (Bat  (None, 7, 7, 32)          128
chNormalization)

dropout_17 (Dropout)         (None, 7, 7, 32)          0

flatten_11 (Flatten)         (None, 1568)              0

dense_22 (Dense)             (None, 64)                100416

dense_23 (Dense)             (None, 5)                 325

=================================================================
Total params: 138,533
Trainable params: 138,277
Non-trainable params: 256
_________________________________________________________________
```

Summary of the model

- We reshaped the data to 28*28 pixels and input it to the CNN layer 1, this layer has 32 filters with 3*3 kernel, stride of 1 and padding as 'same'. This layer has converted the data to a shape of 28*28 with 32 activation maps. This layer uses RelU as the activation function. Output is (28,28,32)
- Next we normalize the output using BatchNormalization. Output (28,28,32)
- Maxpool layer with pool_size = 2,2. This layer has converted the input to (14,14,32).
- Dropout 25% of neuron's. (14,14,32)
- Next hidden layer is the second CNN layer with 64 filters, 3*3 kernel size, stride of 1 and padding as 'same' with RelU as activation function The output of this layer is (14,14,64).
- Normalize the output using BatchNormalization. Output (14,14,64).

- Maxpool layer with pool_size = 2,2. This layer has converted the input to (7,7,64).
- Dropout 25% of neuron's. (7,7,64).
- Next hidden layer is the third CNN layer with 32 filters, 3*3 kernel size, stride of 1 and padding as 'same' with RelU as activation function The output of this layer is (7,7,32).
- Next we normalize the output using BatchNormalization. Output (7,7,32).
- Dropout layer drops the 25% of the neuron's and leaves (7,7,32). as output.
- Next layer flattens the output into one column of 1568 rows with activation function RelU.
- Fully connected layer with 64 neuron's, we used the 'RelU' activation function.
- Fully connected output layer with 5 neuron's, we used the 'Softmax' activation function.
- Parameters are weights that are learnt during training. For own CNN model, there are 138,277 trainable parameters.



The new model reached a test **accuracy of 94.16%,** beating the default model.

Runtime Information

For training the CNN model we used EarlyStopping with monitor variable as validation loss and patience = 2, this helps in stopping the training of the model when the change in monitor variable is not significant. The **accuracy** of the model is **94.16**% and the **loss** is **16.48** %. The time lapsed in training the mode with 100 epochs and batch size of 128 is 49.588 seconds. The time taken in testing the model is 1.369 seconds.