Implementation of

a) MLP for Classification
b) CNN backbone model using pytorch
c) Combining the MLP & CNN

Submitted by Group 7:

Sandeep Kumar Singh, Roll No. 233562487; email: sandeepks23@iitk.ac.in

Sandeep Kumar Yadav, Roll No. 233562488; email: sandeepky23@iitk.ac.in

Satendra Kumar, Roll No. 233562490 ; email: ksatendra23@iitk.ac.in

Sharad Raj, Roll No. 233562494; email: sharadraj23@iitk.ac.in

Sayantan Chakraborty, Roll No. 233562492, email: sayantanc23@iitk.ac.in

Contributions

Name	Contribution
Sandeep Kumar Singh (Roll No. 33562487)	Question 1 & 3 – CNN backbone
Sandeep Kumar Yadav (Roll No. 233562488)	Question 1 & 3 – CNN backbone
Satendra Kumar (Roll No. 233562490)	Question 1 & 2 - MLP for classification
Sharad Raj (Roll No. 233562494)	Question 1 & 2 - MLP for classification
Sayantan Chakraborty (Roll No. 233562492)	Question 4 - Report

MLP for Classification	CNN Backbone Model
Architecture : Dense Layer Feedforward Backpropagation	**Architecture:** Convolutional Model Fully Connected Layers
Hyperparameters: No. of layers & nodes Batch Size Learning Rate Epochs Activation Functions	**Hyperparameters:** Batch Size Learning Rate Epochs Number of Filters & Filter Size
Evaluation Metrics Accuracy, Precision, F1 score, Confusion Matrix	**Evaluation Metrics** Accuracy, Precision, F1 score, Confusion Matrix
Performance (Experiments) of the model Batch Size Learning Rate Epochs	**Performance (Experiments) of the model** No. of Filters and Filter Size Learning Rate Epochs Filter weight initialization

Architecture – MLP

Data Preparation:

- Download TRAINING (60000 samples) and TEST (10000) data from FashionMNIST
- Split TRAINING data into Training (48000) and Validation (12000) datasets
- Data Normalization & Flattening (input to MLP)

Dense Layer:

This class represents a single dense layer in the MLP. It takes input dimensions, output dimensions, activation function, and regularization parameters during initialization.

<u>Methods:</u>

Forward pass - forward
Back propagation : backward
activation functions and their derivatives: relu, relu_prime; sigmoid, sigmoid_prime; softmax, softmax_prime.

MLP:

This class represents the entire MLP model. It maintains a list of DenseLayer objects and a training history dictionary.

<u>Methods:</u>

Adding layers: add_layer
Forward and backward passes: forward, backward
Prediction: predict
Calculating Loss: cross_entropy_loss and training the model: train.

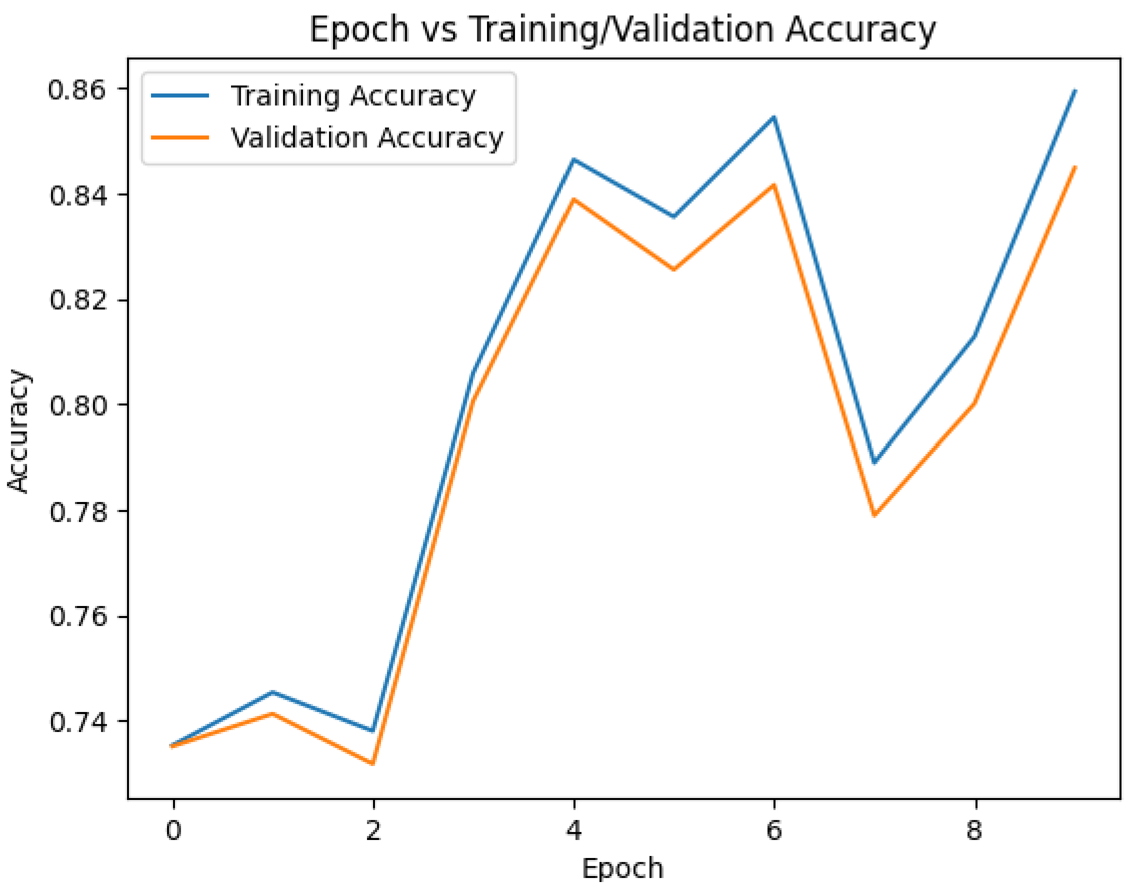
Training & Testing - MLP

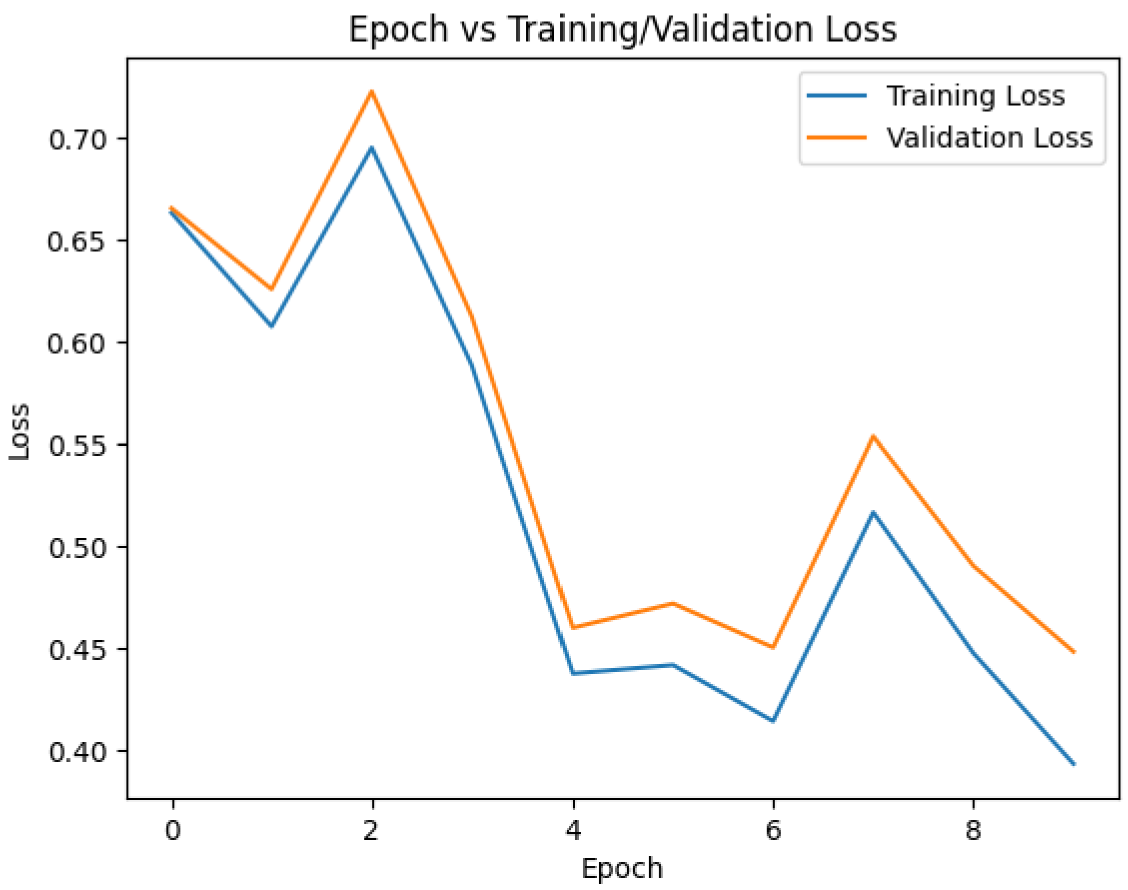
Hyperparameters:

Epochs = 10
input_size = 28 * 28

Batch Size = 64
hidden_size = 128

Learning Rate = 0.01
output_size = 10



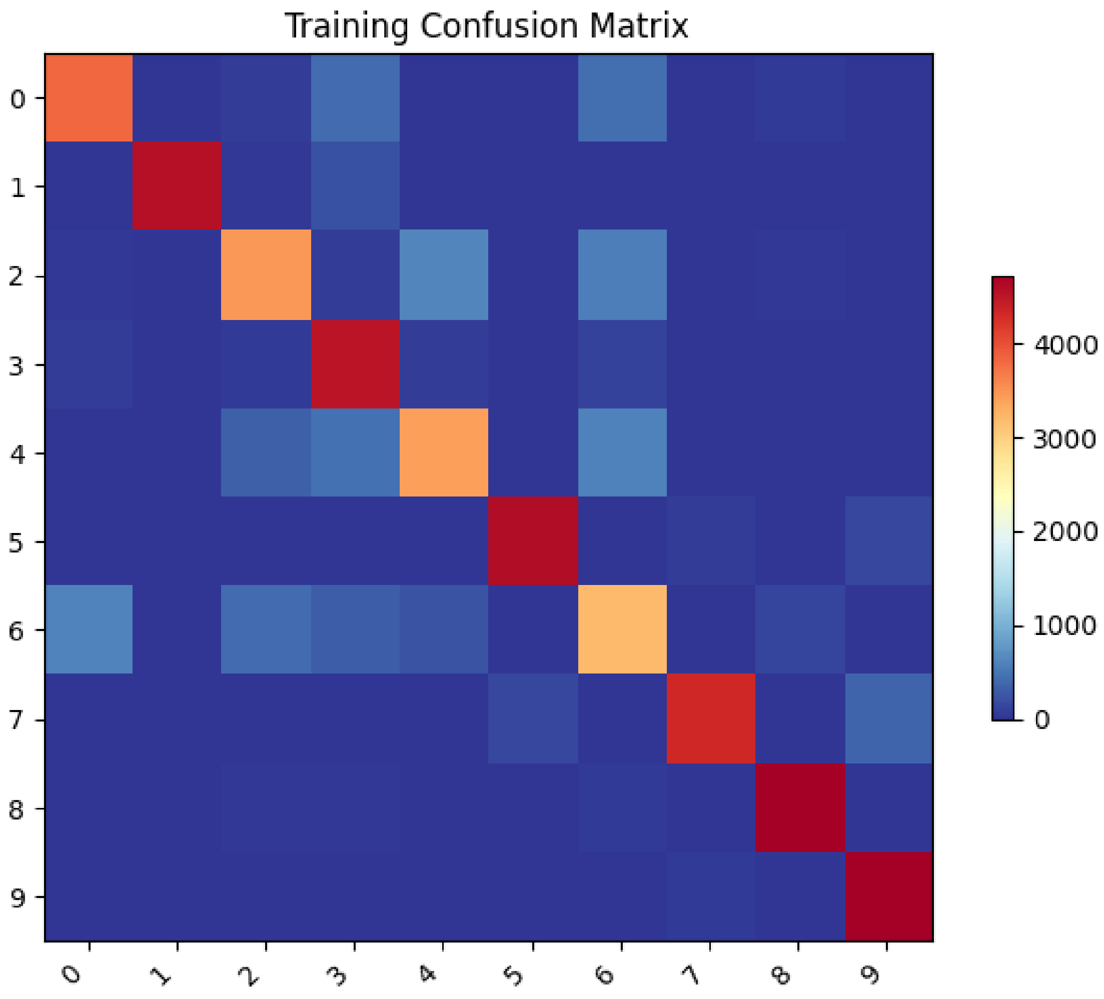


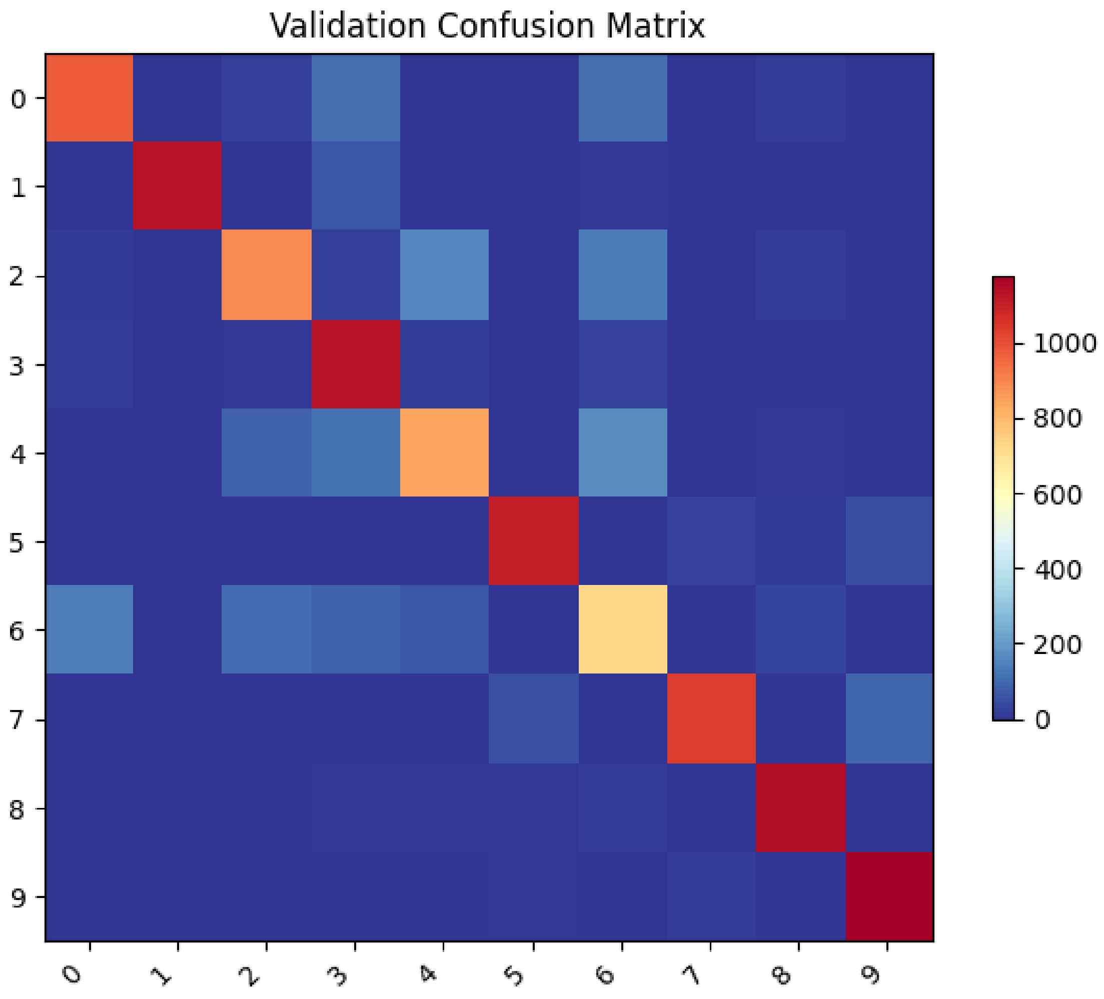
Training & Testing - MLP

Evaluation Metrics:

Training Accuracy: 0.8594
Training Recall : 0.8594
Training F1 Score: 0.8588

Validation Accuracy: 0.8448
Validation Recall : 0.8448
Validation F1 Score: 0.8443





Training & Testing - MLP

Evaluation Metrics:

Training Accuracy: 0.8594
Training Recall : 0.8594
Training F1 Score: 0.8588

Validation Accuracy: 0.8448
Validation Recall : 0.8448
Validation F1 Score: 0.8443

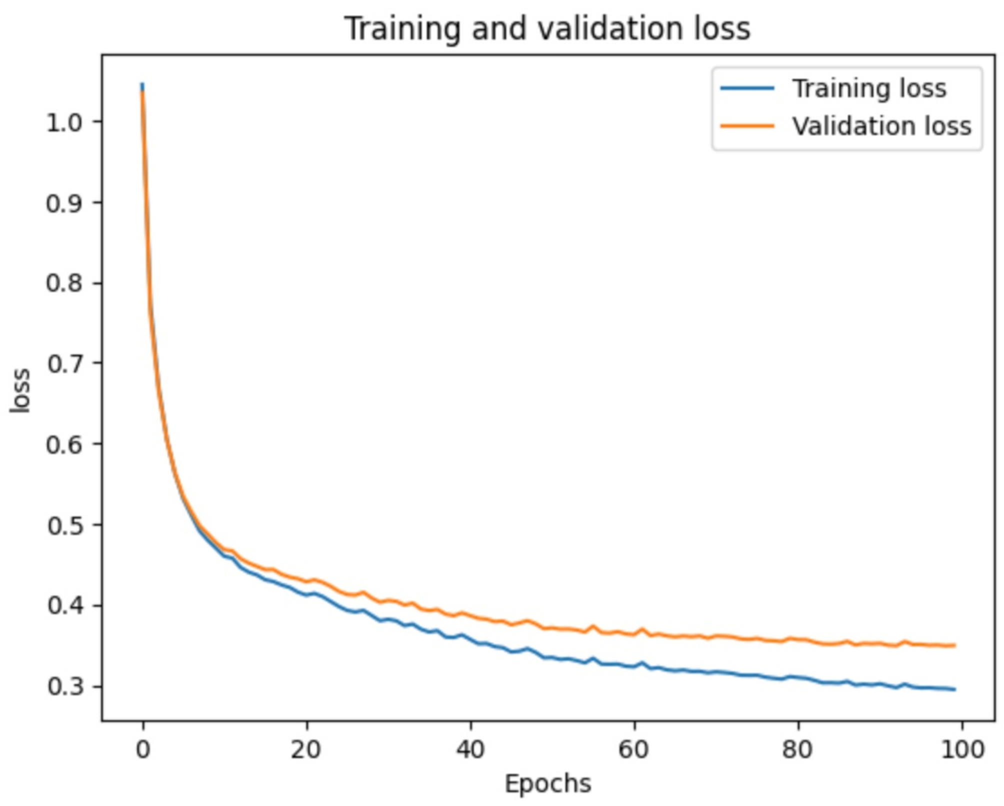
```
Confusion Matrices:
Training:
       Predicted
Actual     0   1   2   3   4   5   6   7   8   9
0   [3804    4   63  406    8    0  426    0   53    0]
1   [   3 4543   24  208    5    0    8    0    3    0]
2   [  25    6 3462   73  609    0  565    0   28    0]
3   [  73    8   37 4513   56    0   98    0   11    0]
4   [   3    6  329  449 3409    0  579    0   10    0]
5   [   1    3    0    6    0 4579    0   61   15  141]
6   [ 604    4  409  299  227    0 3191    0  117    0]
7   [   0    0    0    0    0  134    0 4322    9  355]
8   [   7    2   19   21   10    5   48    3 4704    1]
9   [   0    2    0    2    0   15    0   48    7 4722]

Validation:
       Predicted
Actual     0   1   2   3   4   5   6   7   8   9
0   [975   0  23 109   4   0 107   0  18   0]
1   [   2 1129    4   64    0    0    5    0    2    0]
2   [ 12   0 884  20 160   0 138   0  18   0]
3   [  16    1    9 1130   17    0   27    0    3    1]
4   [  0   4  85 113 839   0 168   0   6   0]
5   [   3    1    0    1    0 1104    1   26   10   48]
6   [142   1 103  84  63   1 724   0  31   0]
7   [   0    0    0    0    0   54    0 1033    0   93]
8   [   4    0    1    5    5    6   15    2 1142    0]
9   [   0    1    0    0    0    7    0   17    1 1178]
```

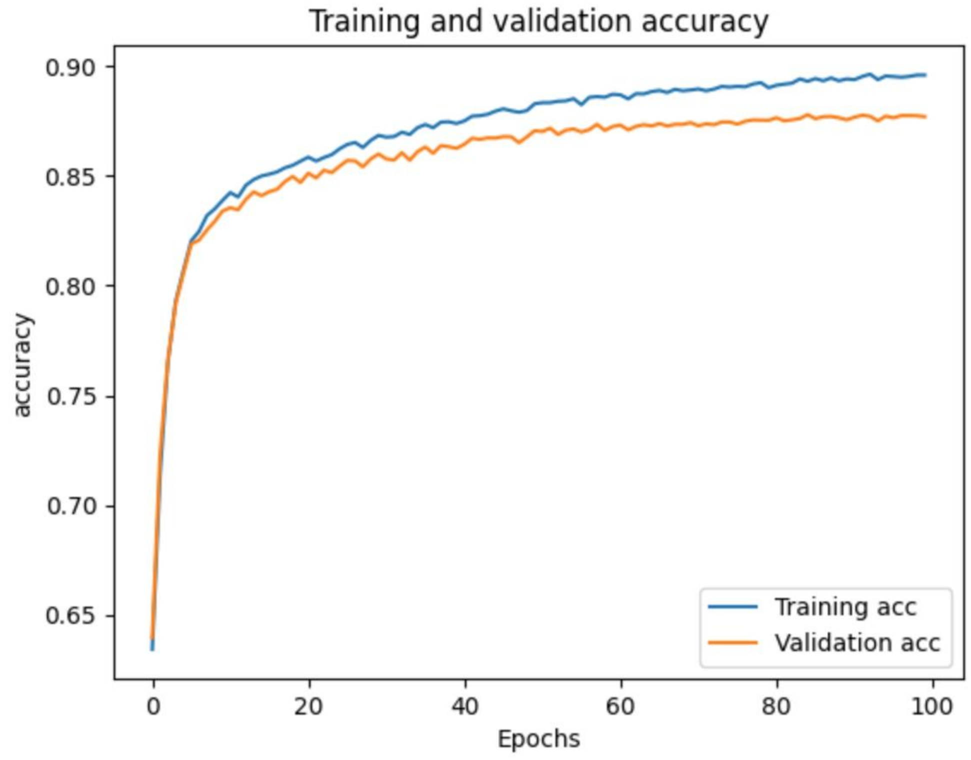
Experiments with – MLP (Decay Rate)

Hyperparameters:

input_size = 28 * 28
output_size = 10
learning_rate = 0.01
Decay_rate= 0.02

hidden_size = 128
epochs = 100
dropout_rate=none



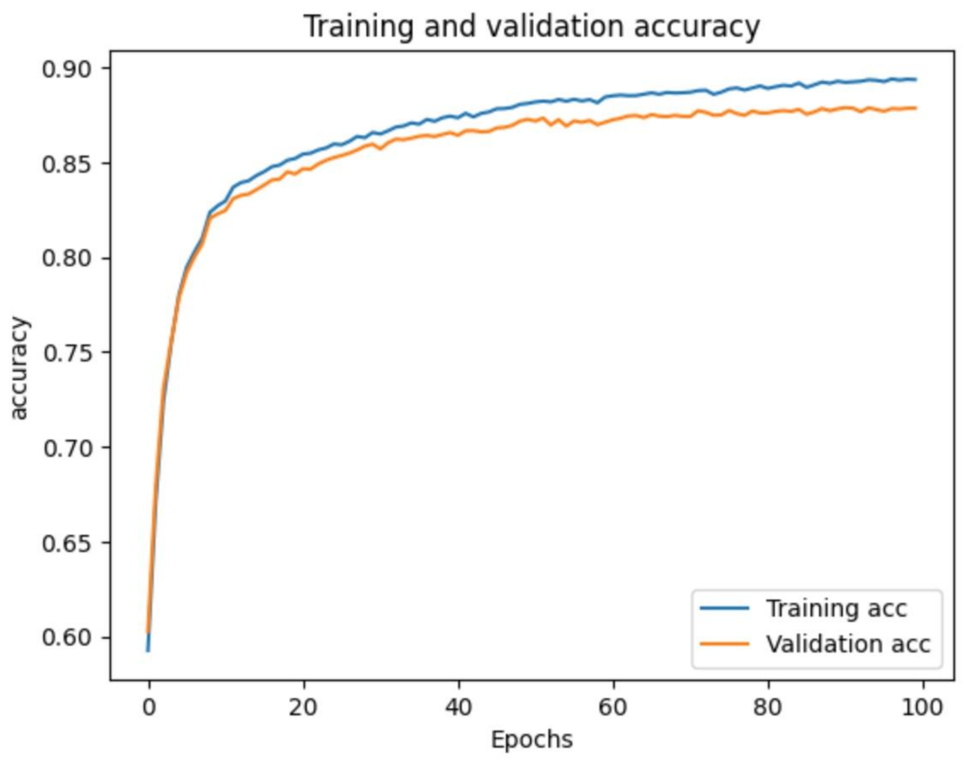


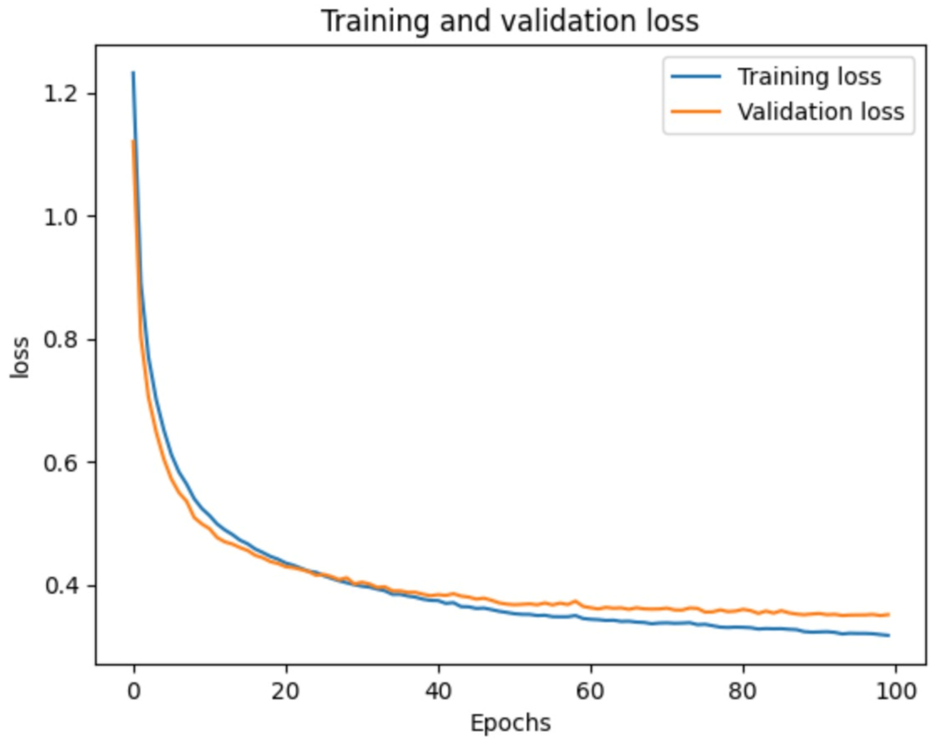
Experiments with – MLP (Dropout Rate)

Hyperparameters:

input_size = 28 * 28
output_size = 10
learning_rate = 0.01
Decay_rate= 0.02

hidden_size = 128
epochs = 100
dropout_rate= 0.2





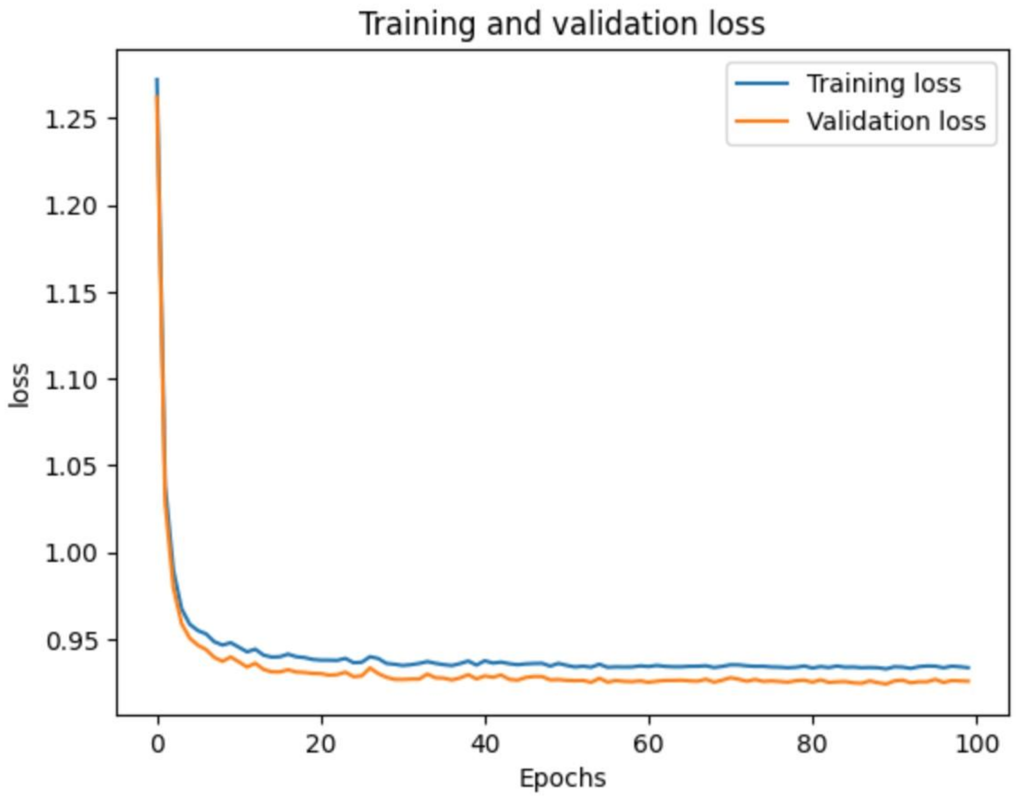
Experiments with – MLP (L2 Regularization)

Hyperparameters:

input_size = 28 * 28
output_size = 10
learning_rate = 0.01
Decay_rate= 0.02

hidden_size = 128
epochs = 100
dropout_rate=None
regularization=L2

```
plt.show()
```



Architecture - CNN

Data Preparation:

- Download TRAINING (60000 samples) and TEST (10000) data from FashionMNIST
- Split TRAINING data into Training (48000) and Validation (12000) datasets
- Data Normalization & Flattening after convolutional layers

Convolution and Pooling Layers:

First Convolutional Block:

nn.Conv2d(1, 49, (3, 3), stride=(1, 1), padding=(1, 1)), # input = (1,28,28), output = (49, 28, 28)

A convolutional layer with 1 input channel (grayscale image), 49 output channels, a 3x3 kernel size, stride of 1, and padding of 1.

nn.ReLU(), # A ReLU activation function applied element-wise.

nn.MaxPool2d((2, 2)), # input = (49, 28, 28), output = (49, 14, 14)

A max-pooling layer with a 2x2 window, reducing the spatial dimensions by a factor of 2.

Architecture - CNN

Convolution and Pooling Layers:

Second Convolutional Block:

```
nn.Conv2d(49, 98, (2, 2), stride=(1, 1), padding=(1,1)),    # input = (49, 14, 14), output = (98, 15, 15)
nn.ReLU(),
nn.MaxPool2d((2, 2)),    # input = (98, 15, 15), output = (98, 7, 7)
```

Third Convolutional Block:

```
nn.Conv2d(98, 196, (2, 2), stride=(1, 1), padding=(1,1)),   # input = (98, 7, 7), output = (196, 8, 8)
nn.ReLU(),
nn.MaxPool2d((2, 2)),    # input = (196, 8, 8), output = (196, 4, 4)
```

Fourth Convolutional Block:

```
nn.Conv2d(196, 392, (2, 2), stride=(1, 1), padding=(1, 1)), # input = (196, 4, 4), output = (392, 5, 5)
nn.ReLU(),
nn.MaxPool2d((2, 2)),                                       # input = (392, 5, 5), output = (392, 2, 2)
```

Fifth Convolutional Block:

```
nn.Conv2d(392, 784, (2, 2), stride=(1, 1), padding=(1, 1)), # input = (392, 2, 2), output = (784, 3, 3)
nn.ReLU(),
nn.MaxPool2d((2, 2))                                         # input = (784, 3, 3), output = (784,1,1)
```

Architecture - CNN

Flatten Layer: input = (784,1,1), output = (784)

Fully Connected Layers (Only in standalone CNN) :

Hidden Layer: input = 784, output = 128, ReLU activation

Output Layer: input = 128, output = 10

Training & Testing – CNN (Standalone)

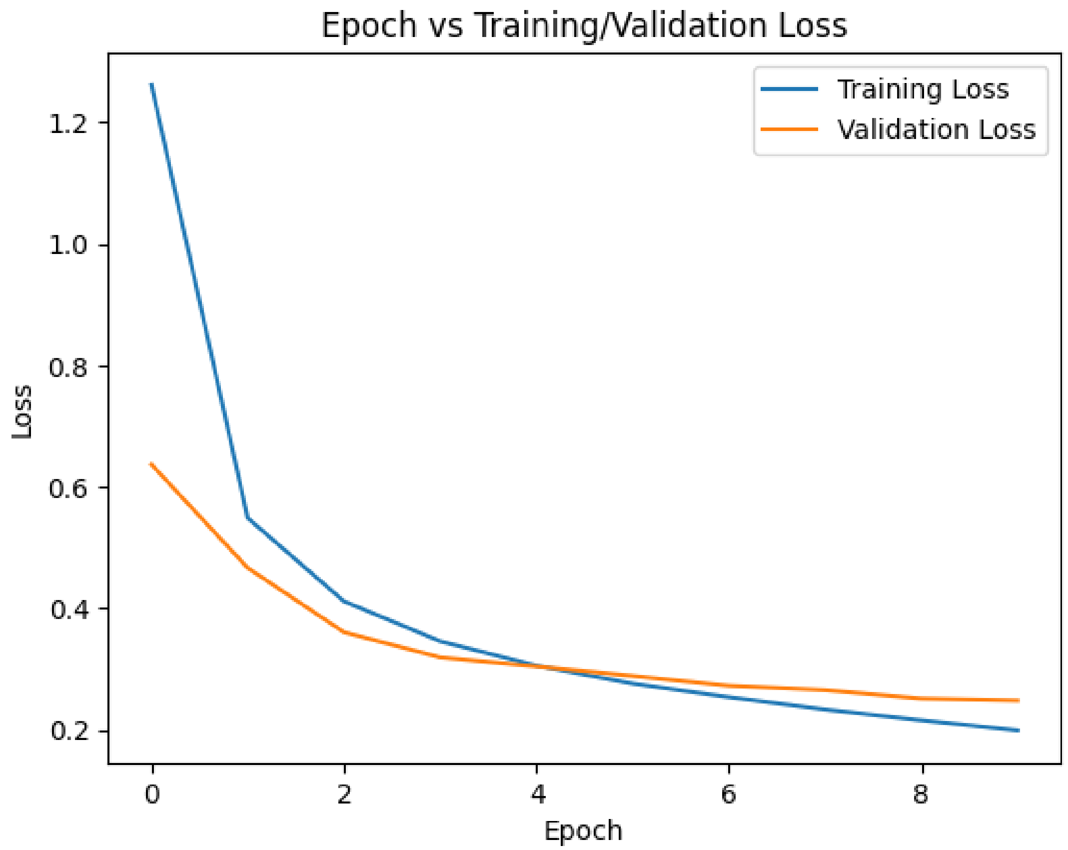
Hyperparameters:

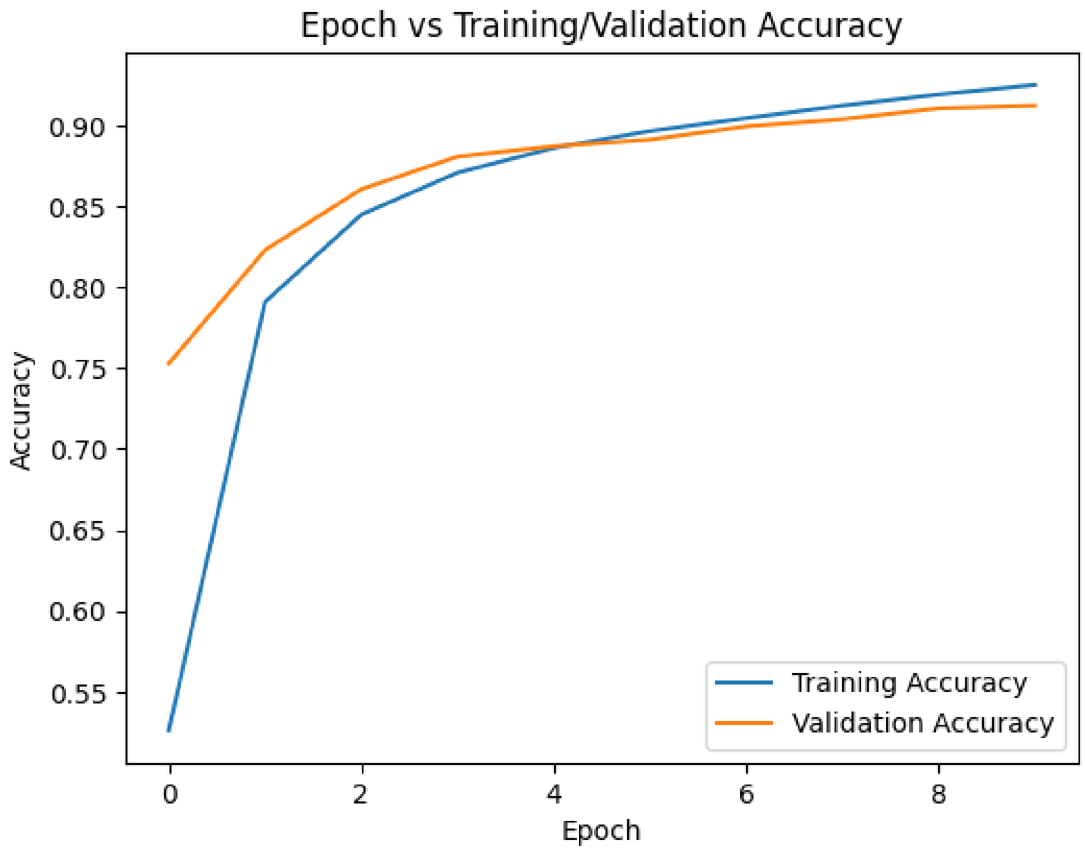
Epochs = 10
No. of Convolution/Maxpool Layer = 5/5
No. of Dense Layer = 2

Batch Size = 64
Input Size = (1,28,28)
Input=(784,1,1)

Learning Rate = 0.01
Output Size = (784,1,1)
Output Size = 10

Observation: Test Phase: Accuracy: 88.5%, Avg loss: 0.322936





Performance – Experiments (Standalone)

Hyperparameters:

Epochs = 10 Batch Size = 64

Learning Rate = 0.1

No. of Filters:

a) Same across all layers : Accuracy: 90.1% ; Avg loss: 0.299253

b) Double across successive layers: Accuracy: 89.4% ; Avg loss: 0.290769

Size of Convolution Layer Filters:

a) (3,3) in first layer, (2,2) in next four layers : Accuracy: 89.2% ; Avg loss: 0.293090

b) (3,3) in first three layers, (2,2) in next two layers : Accuracy: 90.4%; Avg loss: 0.274256

Note: In order to maintain dimension of input features to standalone MLP i.e. (28 *28) was matched with the output of the backbone CNN which is 784.

Stride:

a) (2,2) in first layer
b) (1,1) in first layer

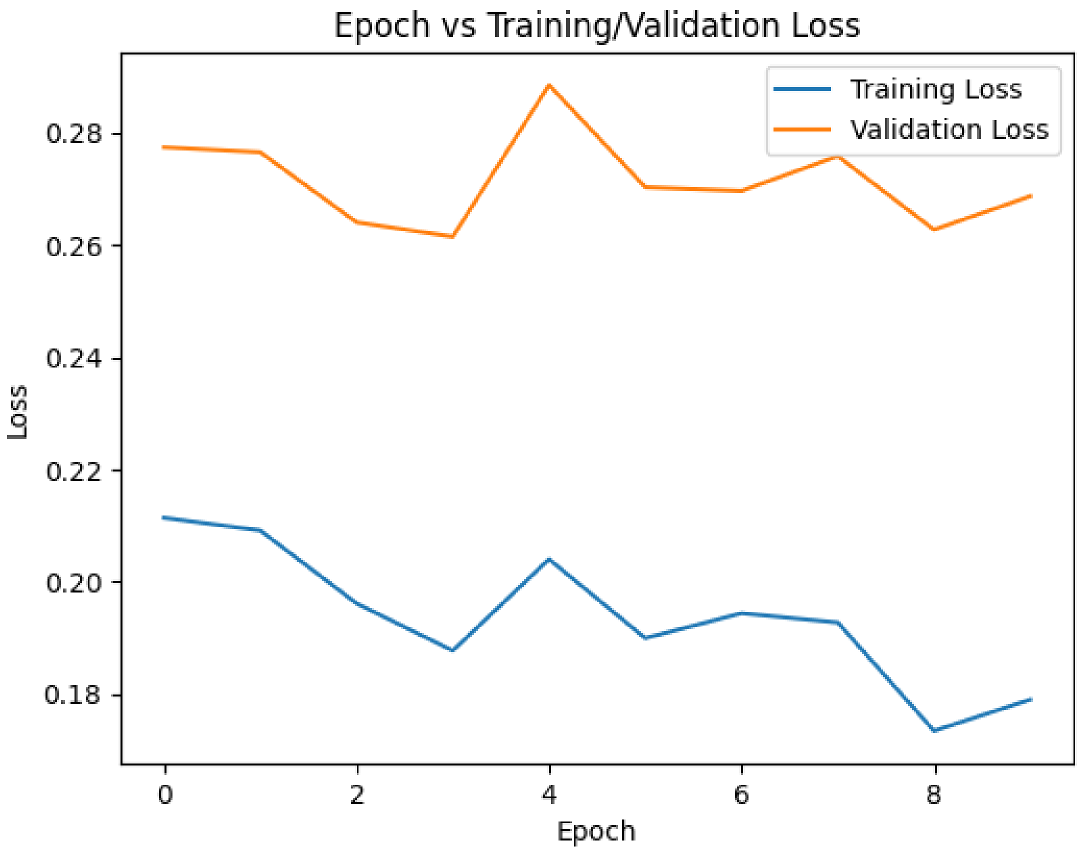
<u>Observation:</u> Larger stride resulted in poor accuracy

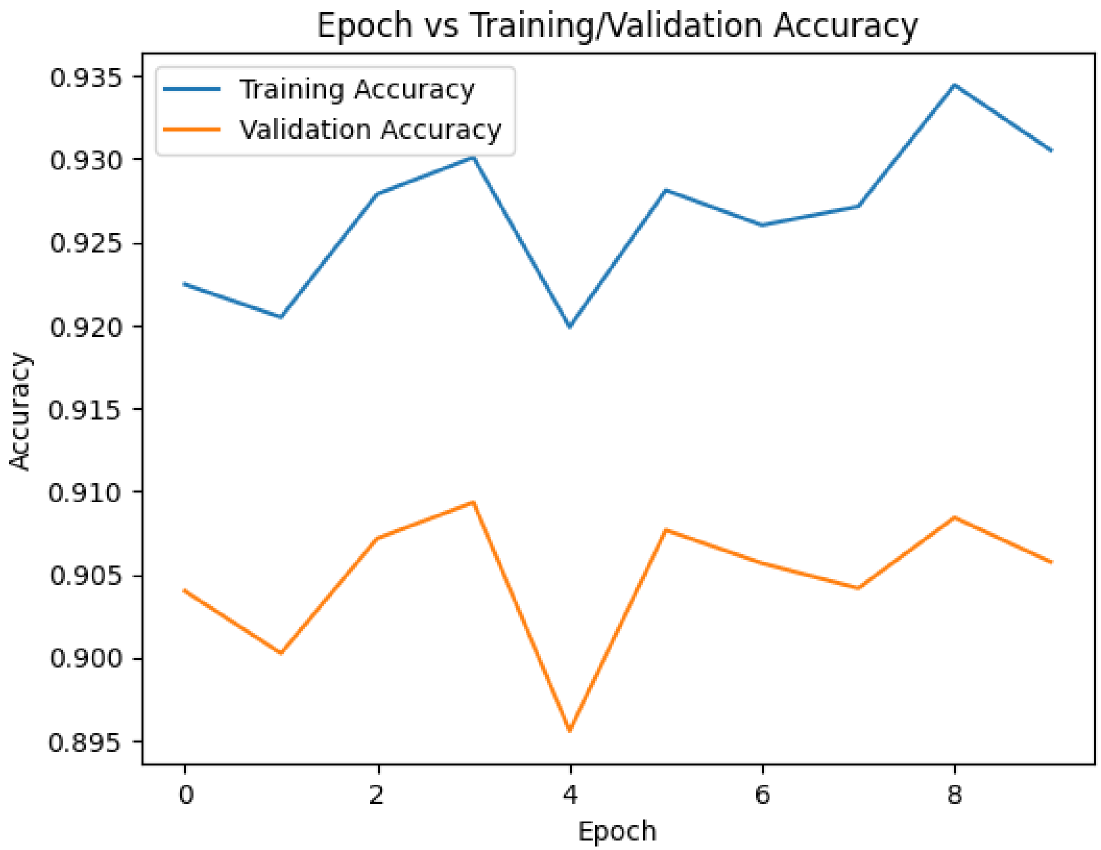
Evaluation Metrics (Combined)

Hyperparameters:

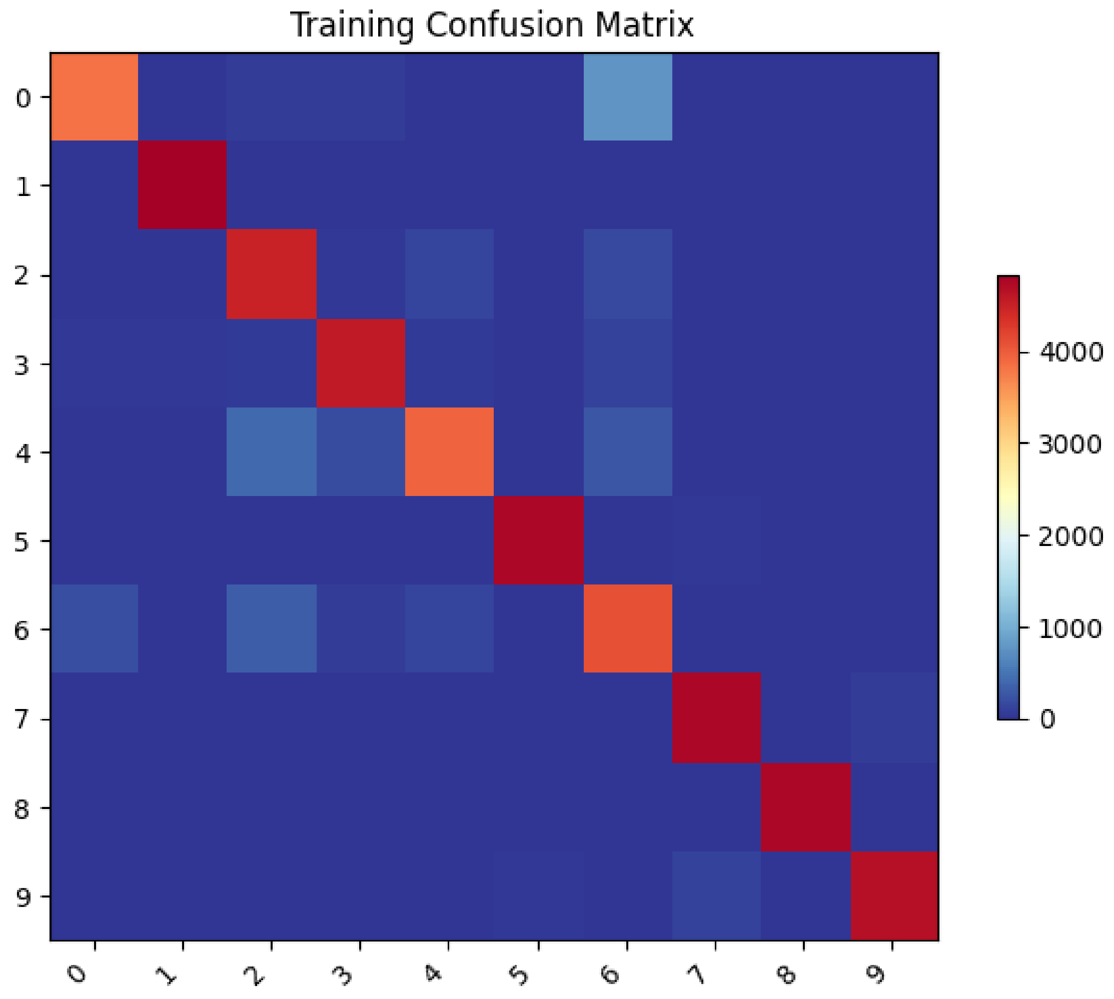
Epochs = 10 Batch Size = 64

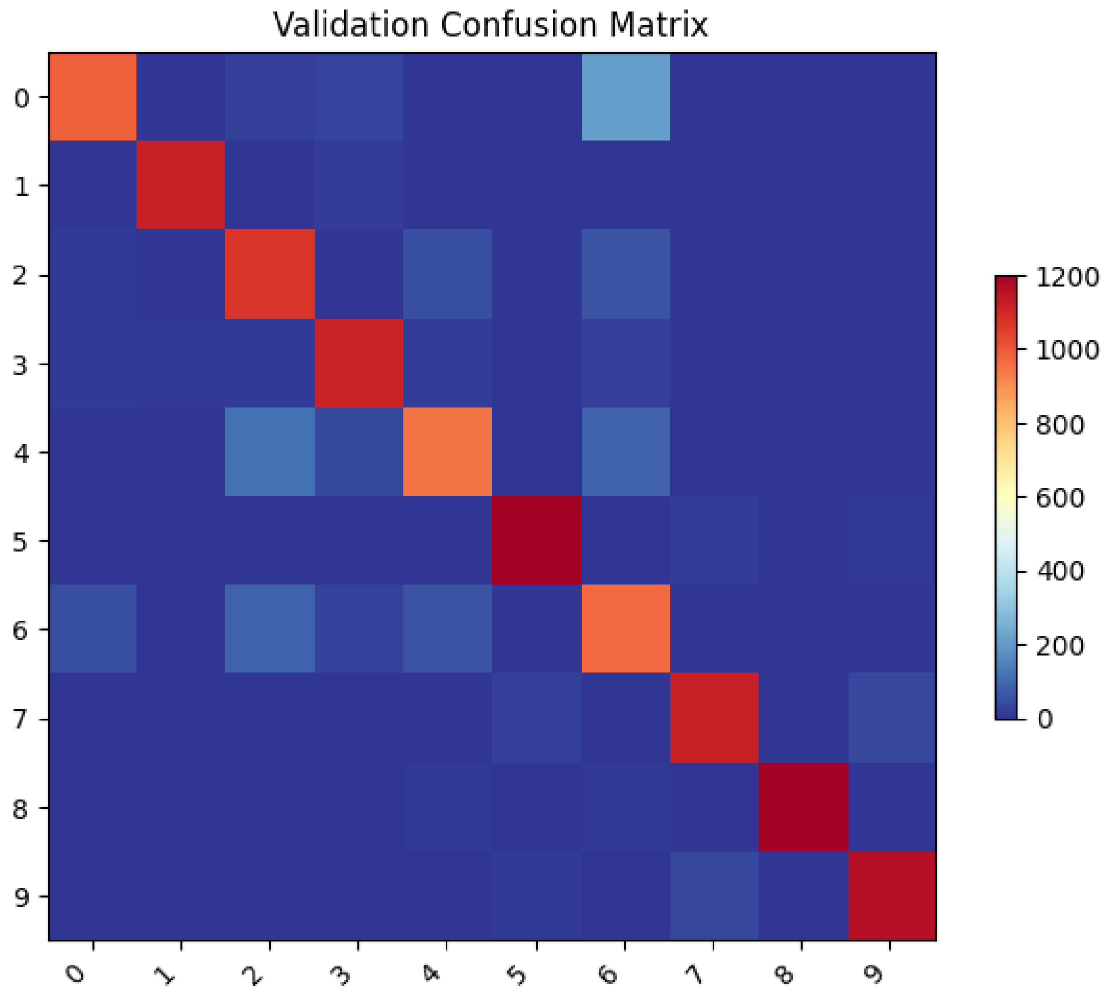
Learning Rate = 0.1





Evaluation Metrics (Combined)





Evaluation Metrics (Combined)

Evaluation Metrics:

Training Accuracy: 0.9782
Training Recall : 0.9782
Training F1 Score: 0.9782

Validation Accuracy: 0.9047
Validation Recall : 0.9047
Validation F1 Score: 0.9051

```
Confusion Matrices:
Training:
      Predicted
Actual    0  1  2  3  4  5  6  7  8  9
0   [3837    2   66   63    3    0  759    0   14    0]
1   [   0 4839    0   15    0    0    3    0    0    0]
2   [  13    1 4485   22  126    0  155    0    3    0]
3   [  25   21   52 4559   45    1  101    0    7    0]
4   [   1    1  410  186 3940    0  247    0   11    0]
5   [   0    0    0    0    0 4748    0   27    0    2]
6   [ 199    1  308   67  126    0 4091    0   13    0]
7   [   0    0    0    0    0   14    0 4746    0   67]
8   [   0    0    6    1    2    1   10    0 4761    0]
9   [   0    0    0    0    0   23    0  112    2 4660]

Validation:
      Predicted
Actual    0  1  2  3  4  5  6  7  8  9
0   [982   0  23  31   4   1 212   0   3   0]
1   [   1 1119    0   18    1    0    3    0    1    0]
2   [  12    2 1069    3   50    0   58    0    1    0]
3   [  13    9   13 1111   18    0   22    0    3    0]
4   [  3   2 122  41 948   0  85   0   3   0]
5   [   0    0    0    0    0 1202    0   15    1    5]
6   [ 50   3  89  27  57   0 966   0   3   0]
7   [   0    0    0    0    0   19    0 1120    1   33]
8   [   4    0    4    4    5    1    5    1 1195    0]
9   [   0    1    0    0    0   10    0   35    0 1157]
```

Reference:

- *Pytorch:* *https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html*
- *Derivative of Softmax:* *https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1*
- *CNN backbone model:* *https://www.baeldung.com/cs/neural-network-backbone*
- *Weight initialization -* *https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/*

Question