

Deep Learning- CSE641

Assignment - 3

Name: Akanksha Shrimal
Name: Vaibhav Goswami
Name: Shivam Sharma

Roll No: MT20055
Roll No: MT20018
Roll No: MT20121

FILES SUBMITTED : submitted .py file , .ipynb file and readme.pdf and output.pdf

UTILITY FUNCTIONS :

1. Save and Load models using Pickle

```
# Saving and Loading models using pickle

def save(filename, obj):
    with open(filename, 'wb') as handle:
        pickle.dump(obj, handle, protocol=pickle.HIGHEST_PROTOCOL)

def load(filename):
    with open(filename, 'rb') as handle:
        return pickle.load(handle)
```

2. Plotting functions

```
# function to plot double line graph
# Plot double line using X1 , Y1 and X2 , Y2
def plot_double_line_graph(X1,Y1,label1 ,X2 ,Y2,label2
,title,y_name):
    fig = plt.figure(figsize=(7,5))
    plt.subplot(111)
    plt.plot(X1,Y1 ,label=label1 ,marker = "x" , color="blue")
    plt.plot(X2, Y2 , label=label2 ,marker = "x" , color="red")
    plt.title(title)
    plt.ylabel(y_name)
    plt.xlabel('Epochs')
    plt.legend( loc='upper left',prop={'size': 13})
    plt.show()

# Plot single line using X1 , Y1
def plot_single_line_graph(X1,Y1,label1, title,name_y):
    fig = plt.figure(figsize=(7,5))
    plt.subplot(111)
    plt.plot(X1,Y1 ,label=label1 ,marker = "x" , color="blue")
    plt.title(title)

    plt.ylabel(name_y)
    plt.xlabel('Epochs')
```

```

plt.legend( loc='lower right',prop={'size': 13})
plt.show()

```

3. Plotting Confusion Matrix

```

def confusion_matrix_find(y, y_hat, nclasses):

    y = y.astype(np.int64)
    y_hat = y_hat.astype(np.int64)

    conf_mat = np.zeros((nclasses, nclasses))

    for i in range(y_hat.shape[0]):
        true, pred = y[i], y_hat[i]
        conf_mat[true, pred] += 1

    return conf_mat

# Plotting confusion matrix
def confusion_matrix_plot(cm, classes, title='Confusion matrix',
cmap=plt.cm.Blues, figsize=(7, 7), path=None, filename=None):

    cm = cm.astype(np.int64)
    plt.figure(figsize=figsize)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```

```
plt.tight_layout()
```

1. Convolutional Neural Network (CNN):

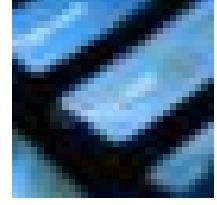
Dataset Description:

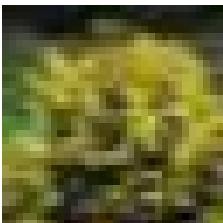
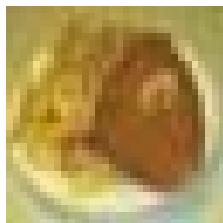
- The dataset was provided in the assignment.
- The dataset contained images, each of size (32×32) .
- There were 5000 images and 5000 corresponding labels/classes.
- There were 10 labels (0,1,2,3,4,5,6,7,8,9) to which each of the images belonged to any one.

Basic Architecture:

- CNN with 3 blocks was to be implemented as follows:
 - Block 1: $[16 \times 3 \times 3] \times X$
 - Block 2: $[32 \times 3 \times 3] \times Y$
 - Block 3: $[64 \times 3 \times 3] \times Z$
- Total 6 convolutional layers were to be used. Therefore, $X + Y + Z = 6$.
- Activation functions: ReLU or Tanh.

1.1 Visualize 10 Samples From Each Class:

Class	Image 1	Image 2
0		
1		
2		
3		

4		
5		
6		
7		
8		
9		

1.2 CNN Architecture

- Block 1 followed by FC layers and softmax.
- Block 1,2 followed by FCs and softmax.
- Block 1,2,3 followed by FCs and softmax.
- $X + Y + Z = 6$. Combination chosen: $X = 2$, $Y = 3$, $Z = 1$

1.2.1 Activation function: Tanh

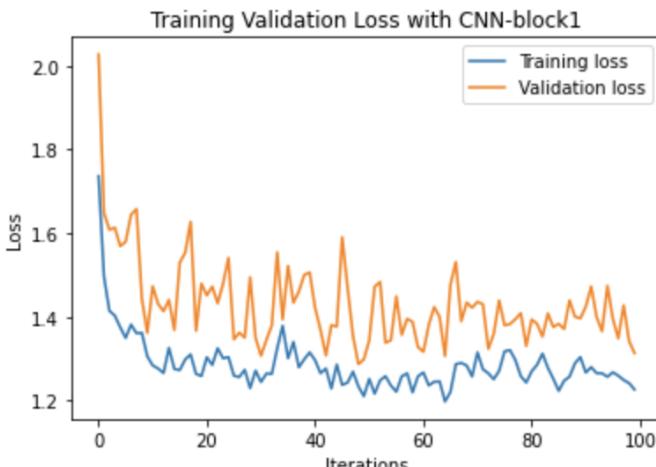
- **Block 1**

```
Model_B1(  
    (features): Sequential(  
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1),  
        padding=(1, 1))  
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,  
        track_running_stats=True)  
        (2): Tanh()  
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),  
        padding=(1, 1))  
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,  
        track_running_stats=True)  
        (5): Tanh()  
        (6): MaxPool2d(kernel_size=4, stride=3, padding=0, dilation=1,  
        ceil_mode=False)  
    )  
    (classifier): Sequential(  
        (0): Linear(in_features=1600, out_features=512, bias=True)  
        (1): Tanh()  
        (2): Linear(in_features=512, out_features=256, bias=True)  
        (3): Tanh()  
        (4): Linear(in_features=256, out_features=64, bias=True)  
        (5): Tanh()  
        (6): Linear(in_features=64, out_features=10, bias=True)  
    )  
)
```

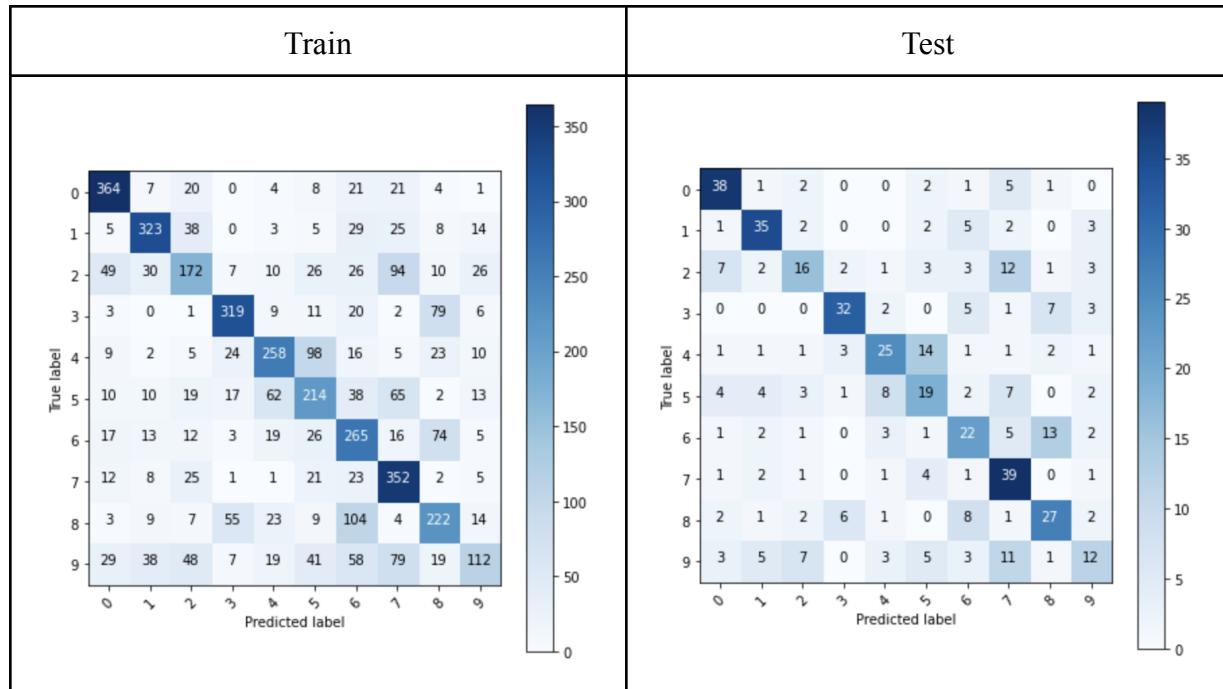
❖ Accuracy and Loss

Block 1	Train	Test
Accuracy	57%	53%
Loss	1.200912	1.292314

❖ Loss plot



❖ Confusion Matrix



- [Block 1,2](#)

```
Model_B12(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): Tanh()
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): Tanh()
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): Tanh()
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): Tanh()
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): Tanh()
        (16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
    classifier): Sequential(
        (0): Linear(in_features=512, out_features=256, bias=True)
)
```

```

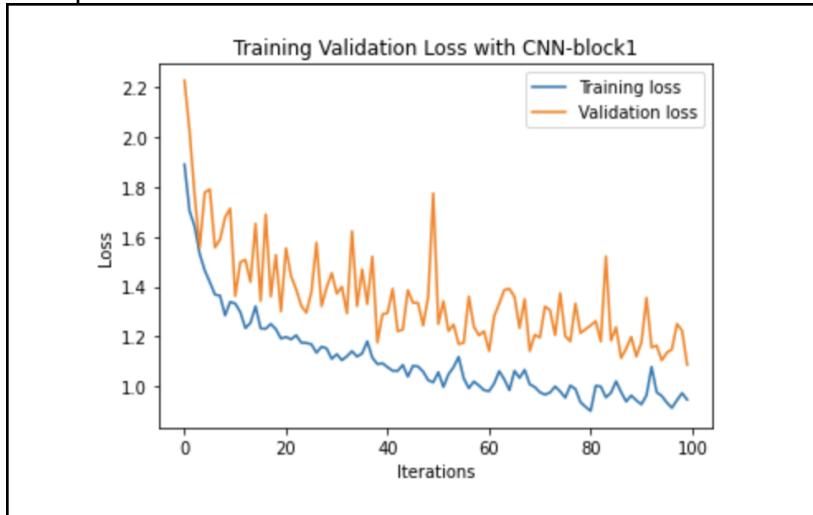
(1) : Tanh()
(2) : Linear(in_features=256, out_features=64, bias=True)
(3) : Tanh()
(4) : Linear(in_features=64, out_features=10, bias=True)
)
)

```

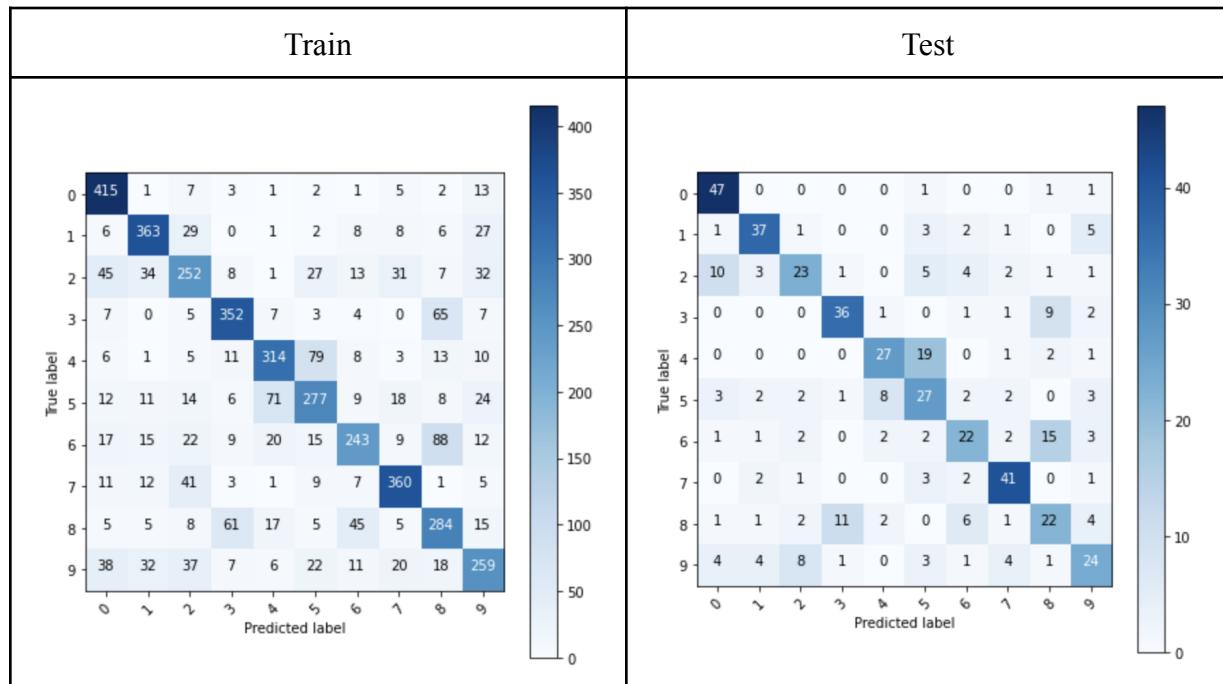
❖ Accuracy and Loss

Block 1,2	Train	Test
Accuracy	69%	61%
Loss	0.894667	1.89120

❖ Loss plot



❖ Confusion Matrix



- Block 1,2,3

```

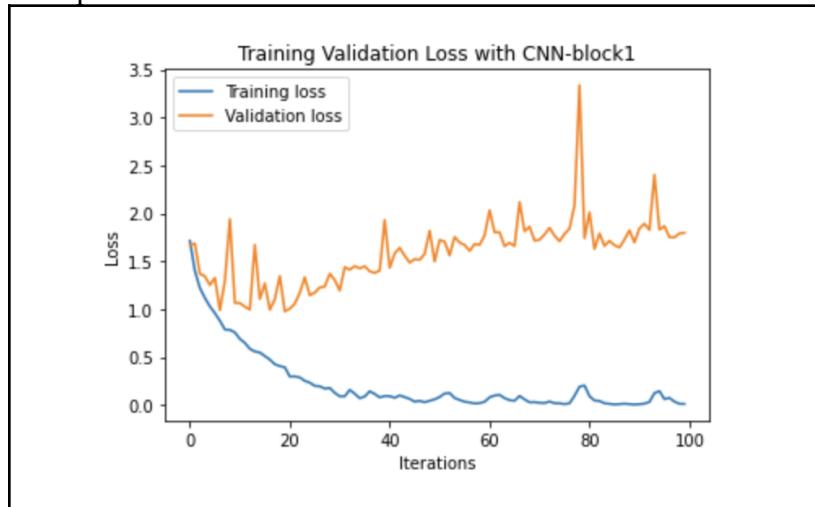
Model_B123(
    (features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): Tanh()
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): Tanh()
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): Tanh()
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): Tanh()
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): Tanh()
        (16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19): Tanh()
        (20): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)

```

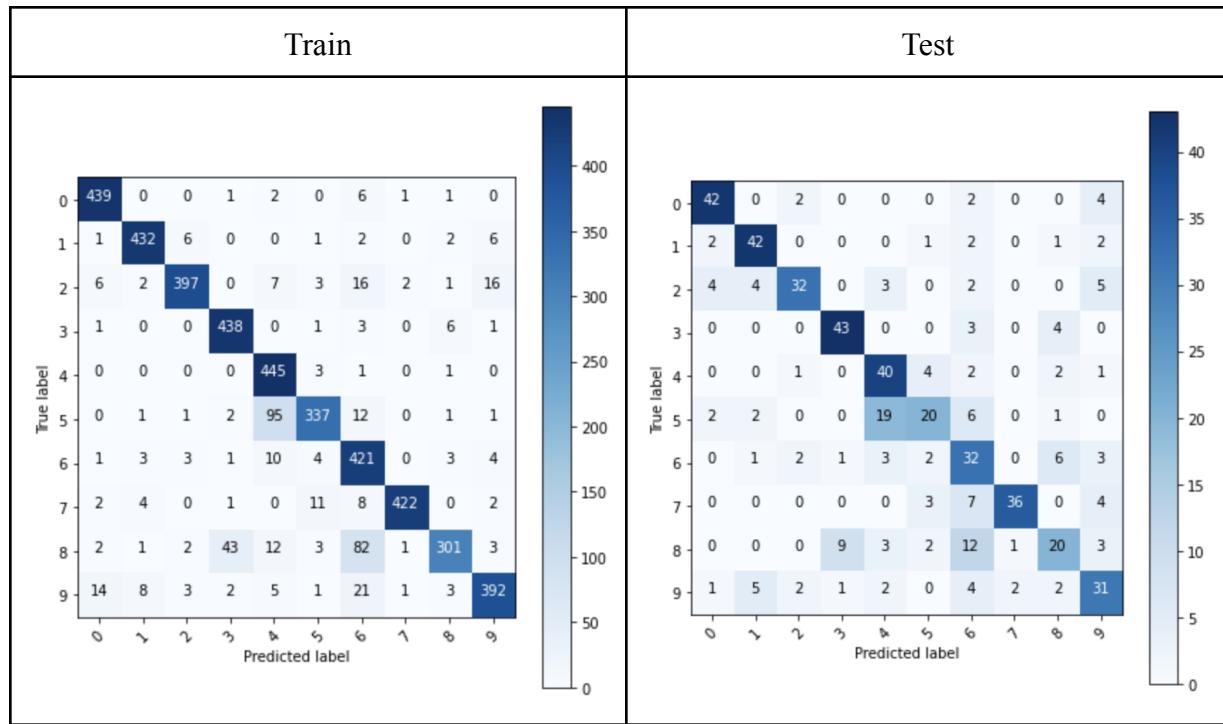
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	89%	67%
Loss	0.301188	0.963042

❖ Loss plot



❖ Confusion Matrix



1.2.2 Activation function: ReLU

- Block 1

```
Model_B1(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
```

```

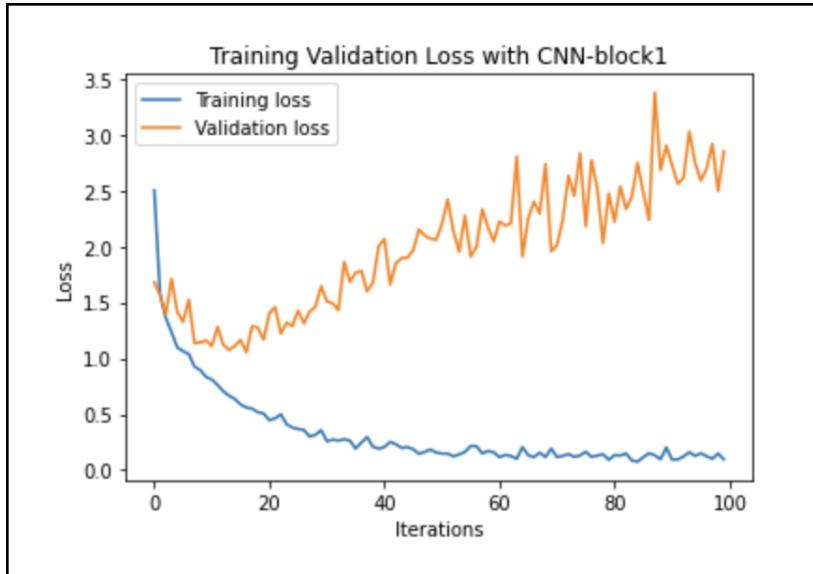
        track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=4, stride=3, padding=0, dilation=1,
ceil_mode=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=1600, out_features=512, bias=True)
        (1): ReLU(inplace=True)
        (2): Linear(in_features=512, out_features=256, bias=True)
        (3): ReLU(inplace=True)
        (4): Linear(in_features=256, out_features=64, bias=True)
        (5): ReLU(inplace=True)
        (6): Linear(in_features=64, out_features=10, bias=True)
    )
)

```

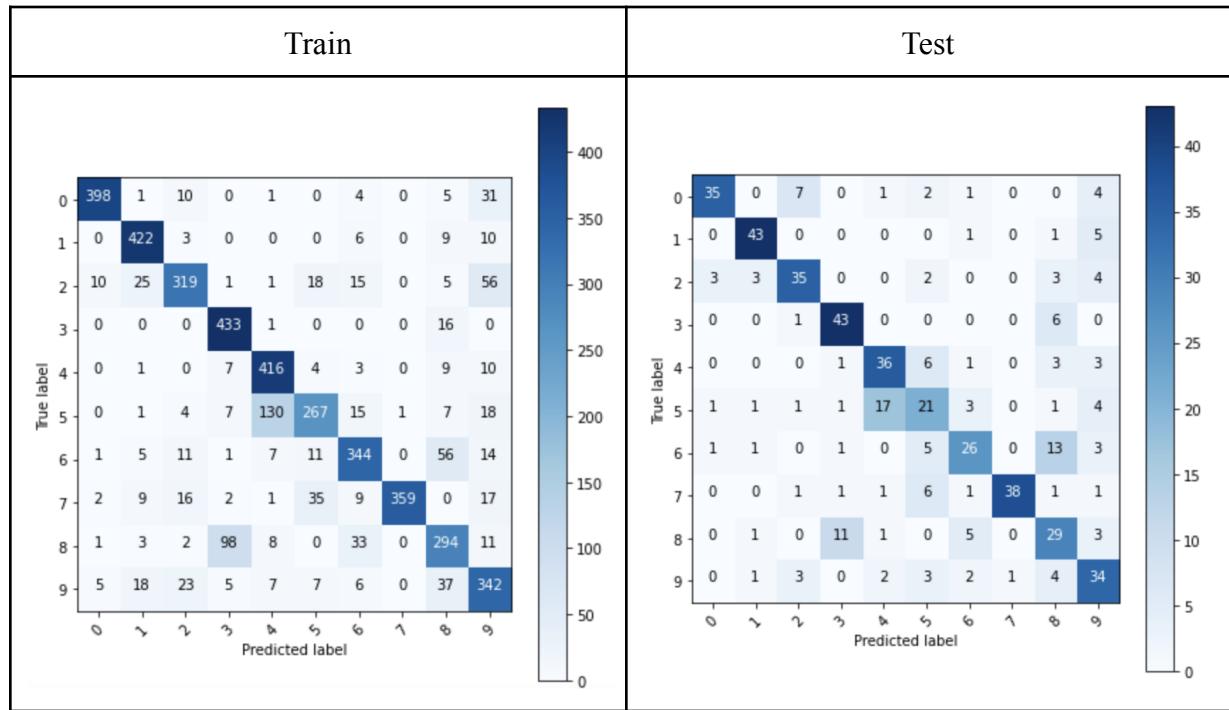
❖ Accuracy and Loss

Block 1	Train	Test
Accuracy	79%	68%
Loss	0.549036	1.057941

❖ Loss plot



❖ Confusion Matrix



- **Block 1,2**

```
Model_B12(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): ReLU(inplace=True)
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): ReLU(inplace=True)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
    classifier): Sequential(
        (0): Linear(in_features=512, out_features=256, bias=True)
    )
)
```

```

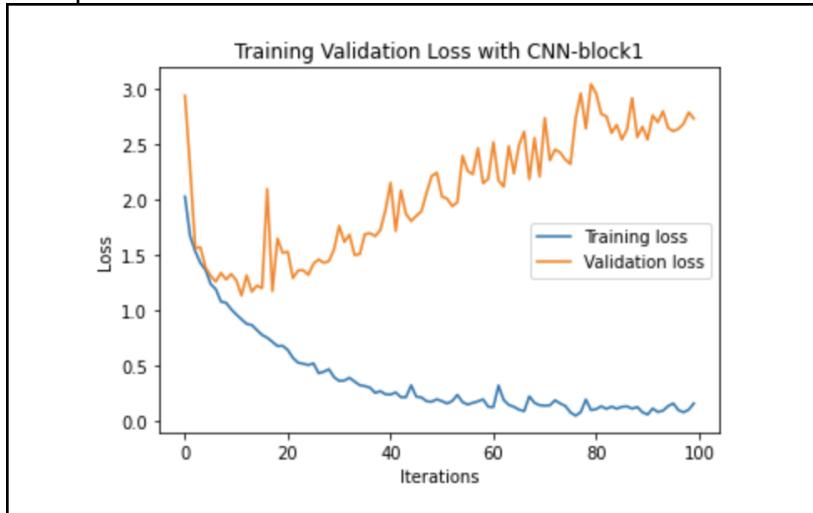
(1) : ReLU(inplace=True)
(2) : Linear(in_features=256, out_features=64, bias=True)
(3) : ReLU(inplace=True)
(4) : Linear(in_features=64, out_features=10, bias=True)
)
)

```

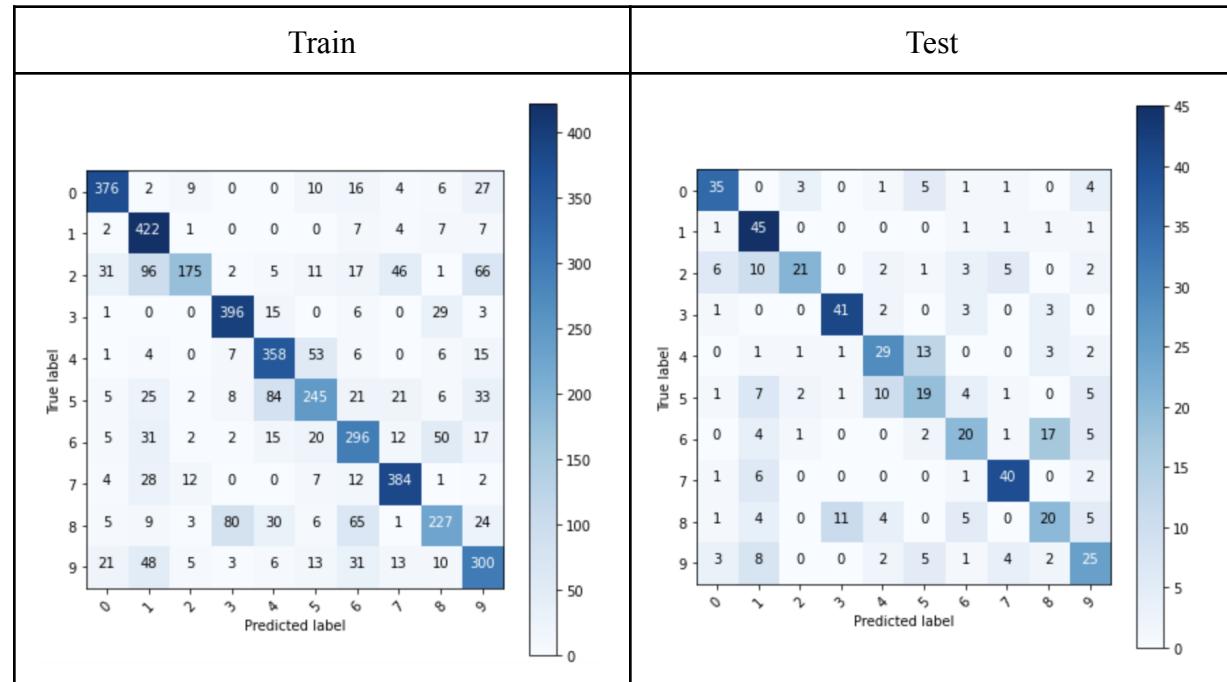
❖ Accuracy and Loss

Block 1,2	Train	Test
Accuracy	70%	59%
Loss	0.837771	1.137523

❖ Loss plot



❖ Confusion Matrix



- Block 1,2,3

```

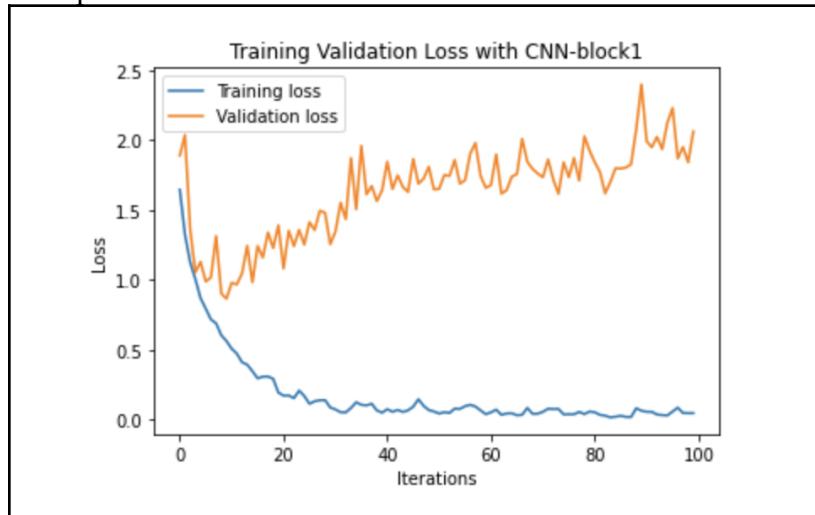
Model_B123(
    (features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): ReLU(inplace=True)
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): ReLU(inplace=True)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19): ReLU(inplace=True)
        (20): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)

```

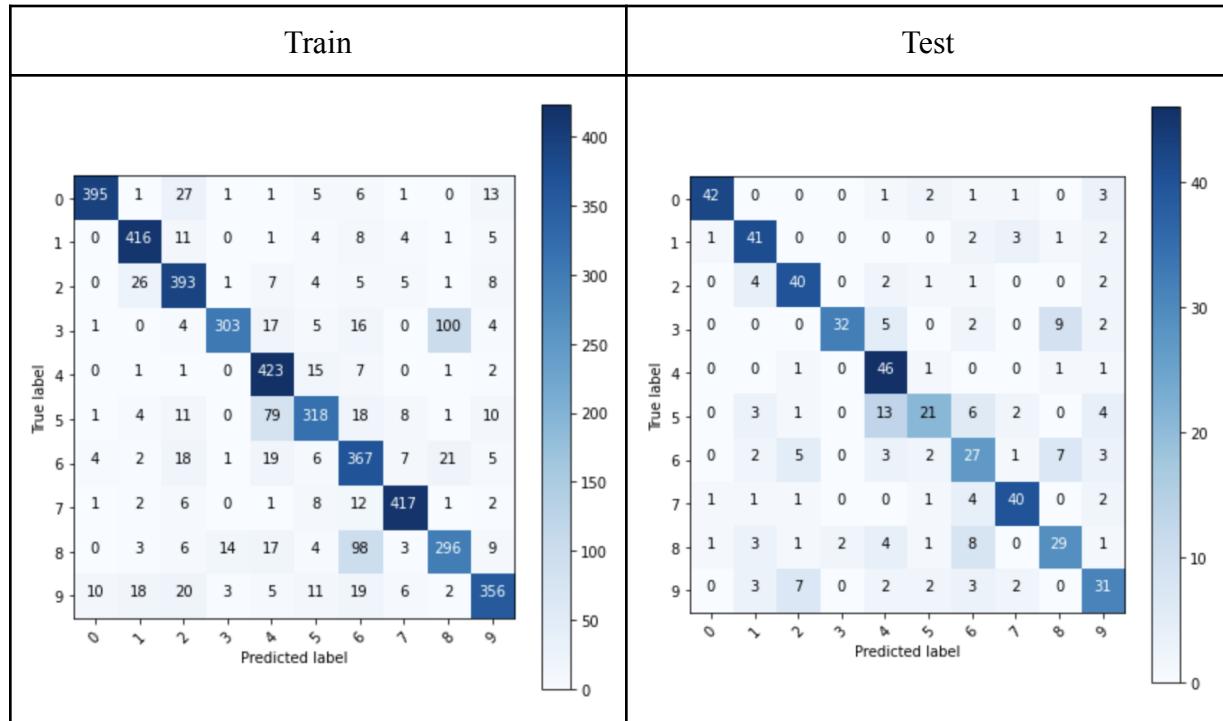
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	81%	69%
Loss	0.512706	0.868855

❖ Loss plot



❖ Confusion Matrix



Analysis:

- Blocks were added one at a time and performance was evaluated for each model.
- With the addition of blocks, the performance increased, i.e. accuracy increased and loss decreased in both cases (tanh and ReLU).
- ReLU performed better on average than tanh. Therefore, for the rest of the question, ReLU activation function is used.

1.3 Dropout Probability

- Dropout was to be implemented in the model. The model was to be tested for 3 values of dropout probability.
- 3 values chosen were - 0.3, 0.5 and 0.7
- 3 ways of implementing dropout were used:
 - After convolutional layers

- Between fully connected layers
- After each block

1.3.1 Dropout after Convolutional layers

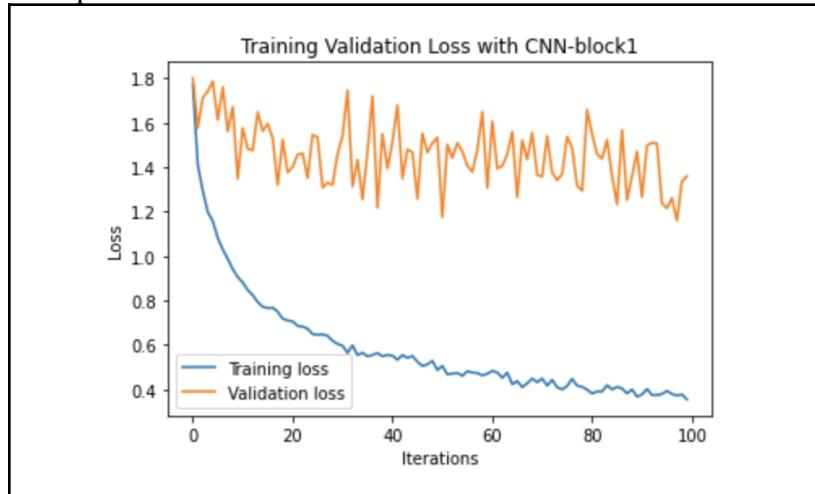
1.3.1.1 Dropout = 0.3

```
Model_B123(
    (features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Dropout(p=0.3, inplace=False)
        (4): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (5): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (6): ReLU(inplace=True)
        (7): Dropout(p=0.3, inplace=False)
        (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (9): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (11): ReLU(inplace=True)
        (12): Dropout(p=0.3, inplace=False)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): Dropout(p=0.3, inplace=False)
        (17): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (18): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19): ReLU(inplace=True)
        (20): Dropout(p=0.3, inplace=False)
        (21): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (22): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (23): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (24): ReLU(inplace=True)
        (25): Dropout(p=0.3, inplace=False)
        (26): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)
```

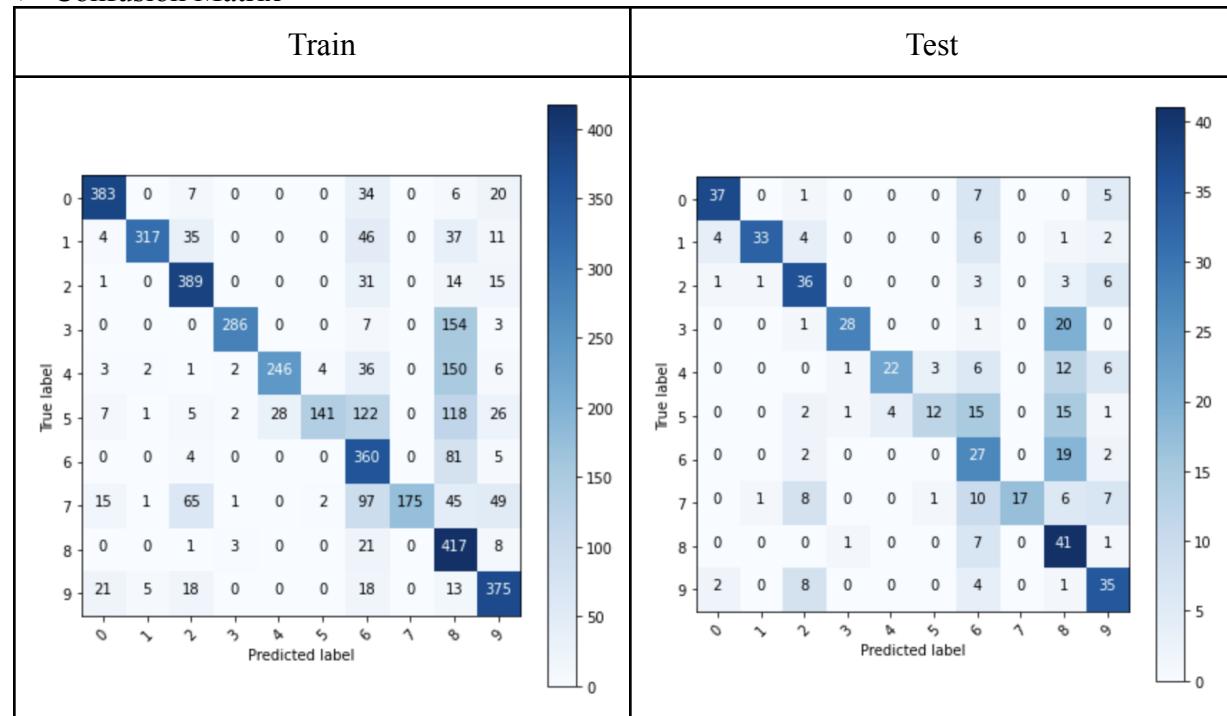
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	68%	57%
Loss	0.903236	1.165431

❖ Loss plot



❖ Confusion Matrix



1.3.1.2 Dropout = 0.5

```

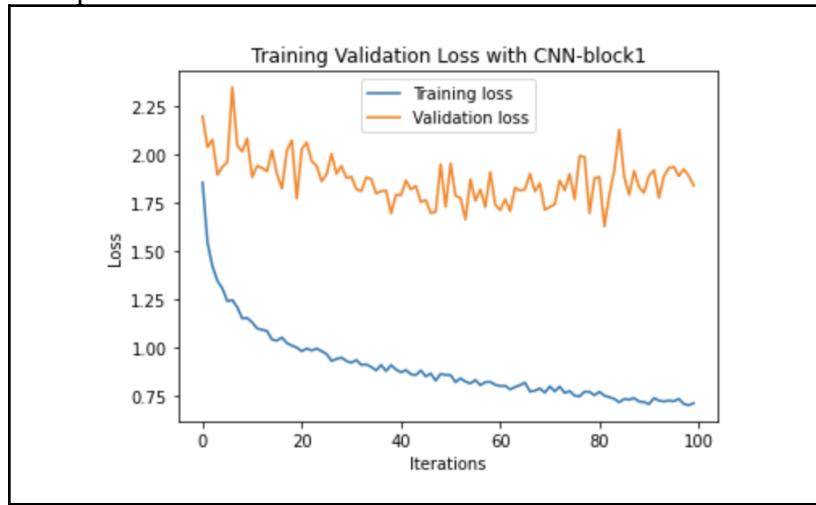
Model_B123(
    (features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Dropout(p=0.5, inplace=False)
        (4): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (5): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (6): ReLU(inplace=True)
        (7): Dropout(p=0.5, inplace=False)
        (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (9): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (11): ReLU(inplace=True)
        (12): Dropout(p=0.5, inplace=False)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): Dropout(p=0.5, inplace=False)
        (17): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (18): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19): ReLU(inplace=True)
        (20): Dropout(p=0.5, inplace=False)
        (21): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (22): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (23): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (24): ReLU(inplace=True)
        (25): Dropout(p=0.5, inplace=False)
        (26): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)

```

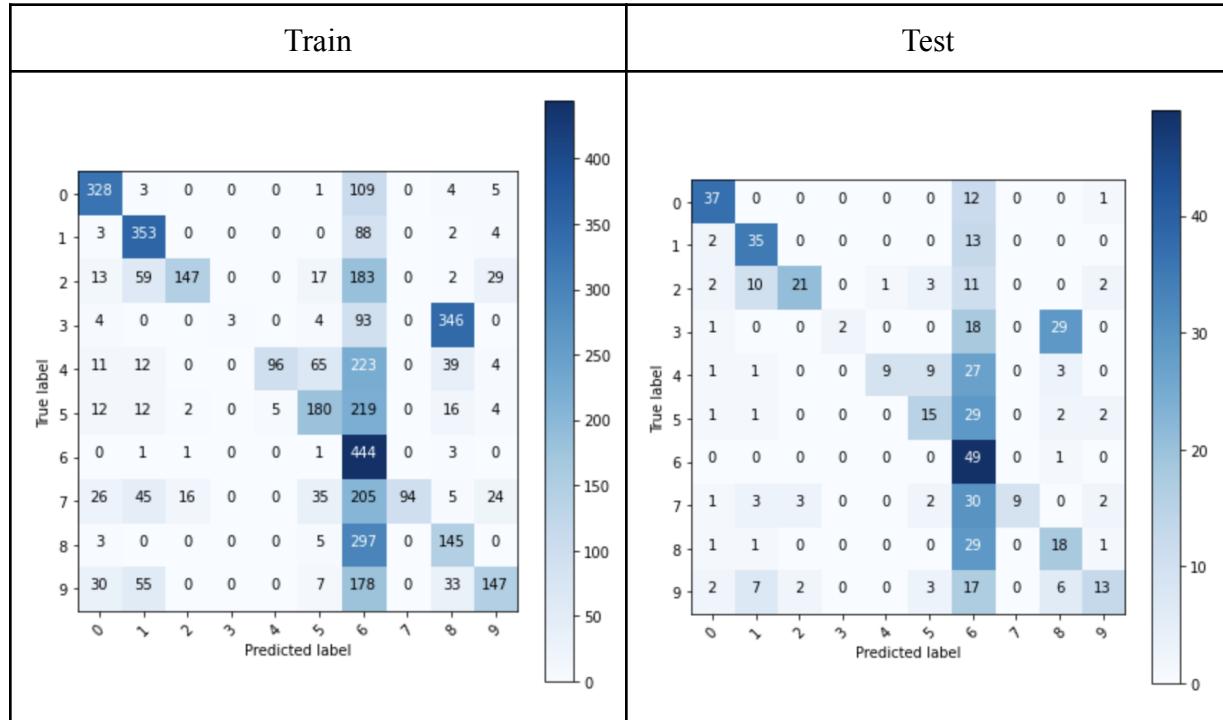
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	43%	41%
Loss	1.542332	1.630709

❖ Loss plot



❖ Confusion Matrix



1.3.1.2 Dropout = 0.7

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Dropout(p=0.7, inplace=False)
        (4): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (5): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (6): ReLU(inplace=True)
        (7): Dropout(p=0.7, inplace=False)
        (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
```

```

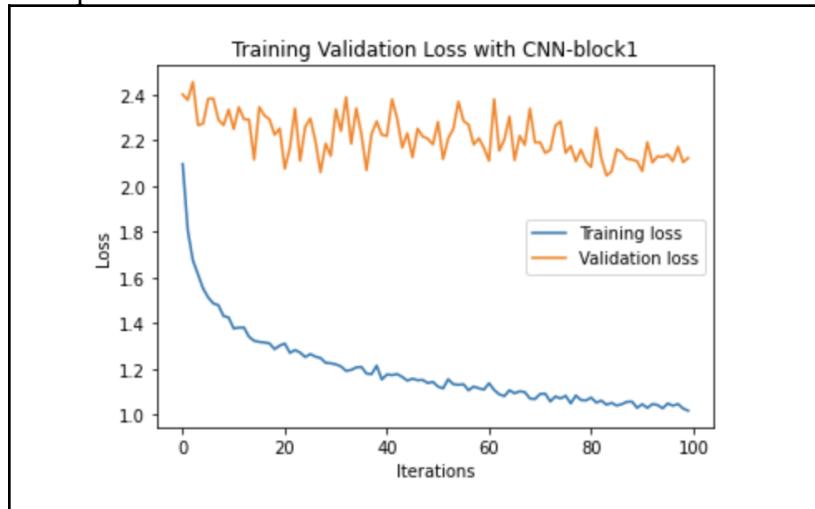
ceil_mode=False)
(9): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
(10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(11): ReLU(inplace=True)
(12): Dropout(p=0.7, inplace=False)
(13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(15): ReLU(inplace=True)
(16): Dropout(p=0.7, inplace=False)
(17): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(18): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(19): ReLU(inplace=True)
(20): Dropout(p=0.7, inplace=False)
(21): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(22): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(23): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(24): ReLU(inplace=True)
(25): Dropout(p=0.7, inplace=False)
(26): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

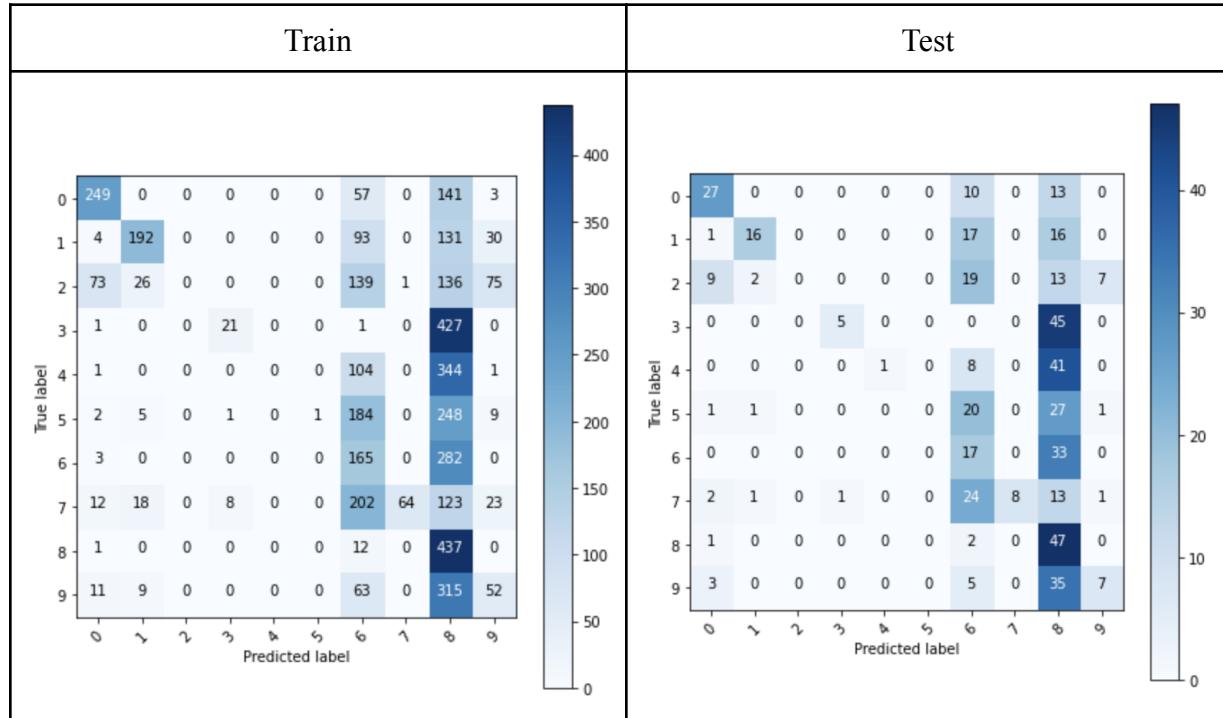
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	26%	25%
Loss	2.020246	2.043498

❖ Loss plot



❖ Confusion Matrix



1.3.2 Dropout between fully-connected layers

1.3.2.1 Dropout = 0.3

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
```

```

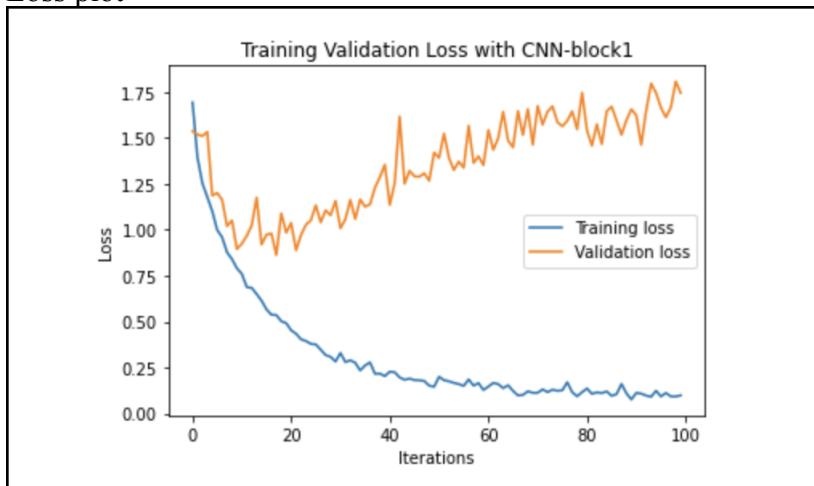
        ceil_mode=False)
(7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
(8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(9): ReLU(inplace=True)
(10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(12): ReLU(inplace=True)
(13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(15): ReLU(inplace=True)
(16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(19): ReLU(inplace=True)
(20): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Dropout(p=0.3, inplace=False)
(1): Linear(in_features=64, out_features=10, bias=True)
)
)
)

```

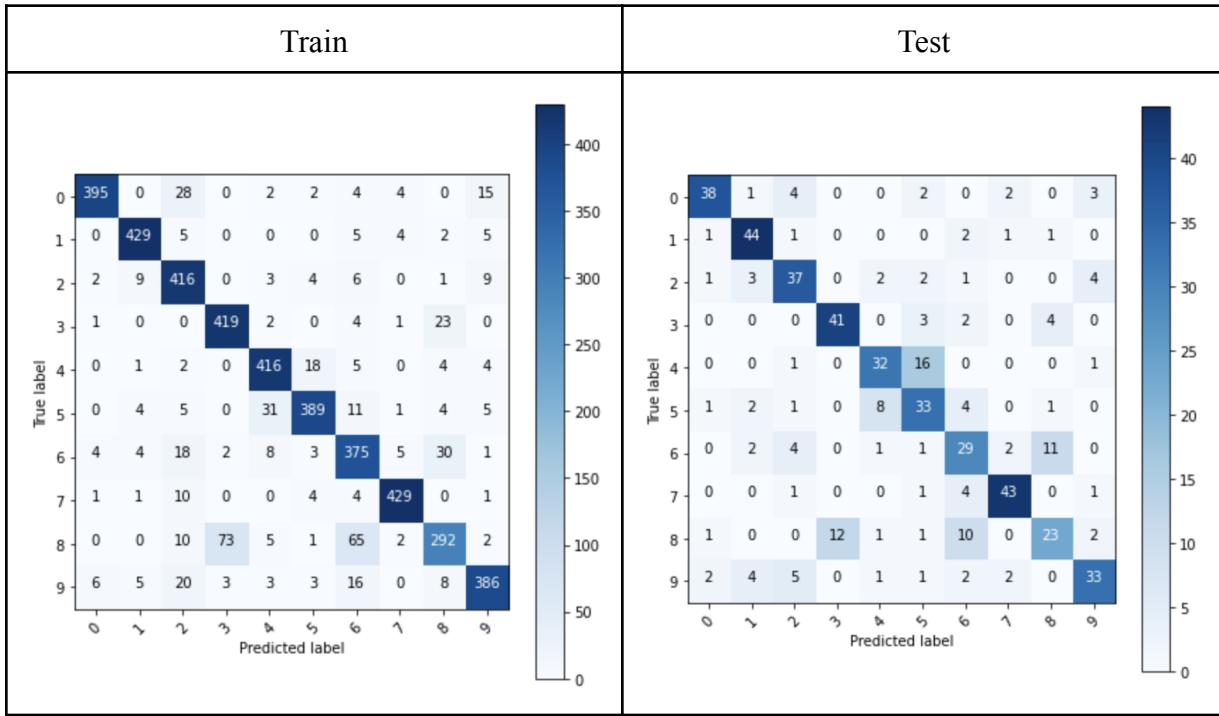
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	87%	70%
Loss	0.355316	0.854087

❖ Loss plot



❖ Confusion Matrix



1.3.2.2 Dropout = 0.5

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): ReLU(inplace=True)
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): ReLU(inplace=True)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```

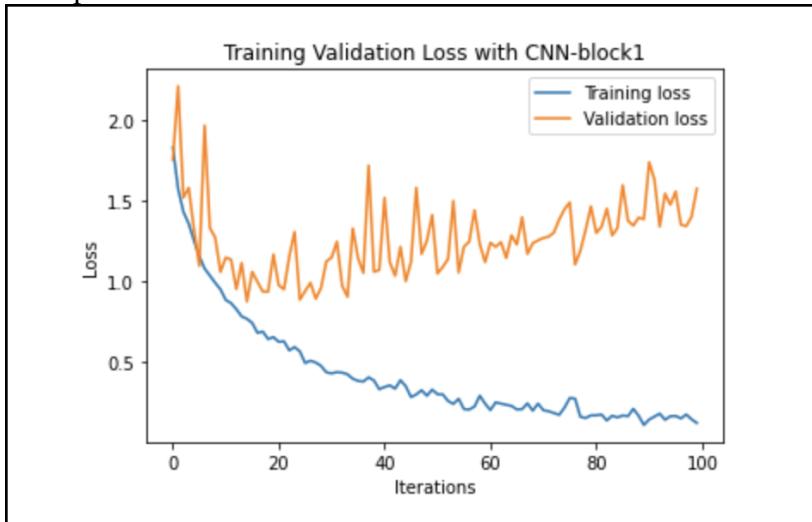
(19): ReLU(inplace=True)
(20): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Dropout(p=0.5, inplace=False)
(1): Linear(in_features=64, out_features=10, bias=True)
)
)

```

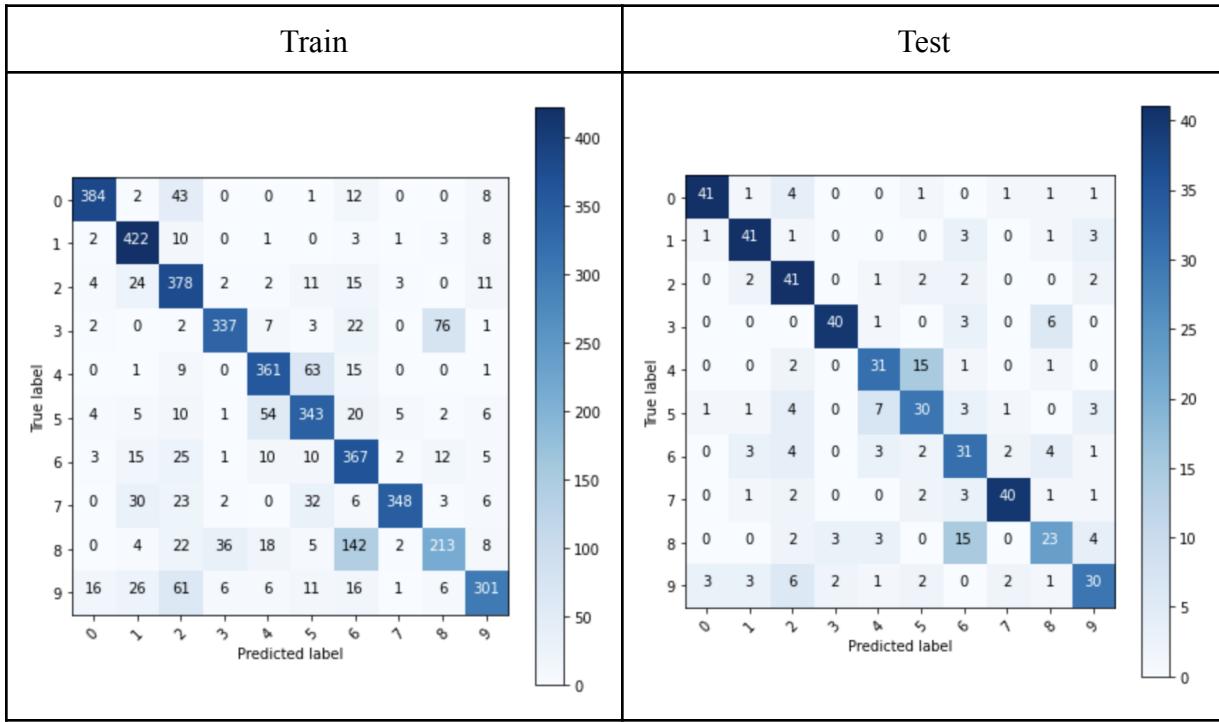
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	76%	69%
Loss	0.660881	0.876783

❖ Loss plot



❖ Confusion Matrix



1.3.2.2 Dropout = 0.7

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): ReLU(inplace=True)
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): ReLU(inplace=True)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19): ReLU(inplace=True)
```

```

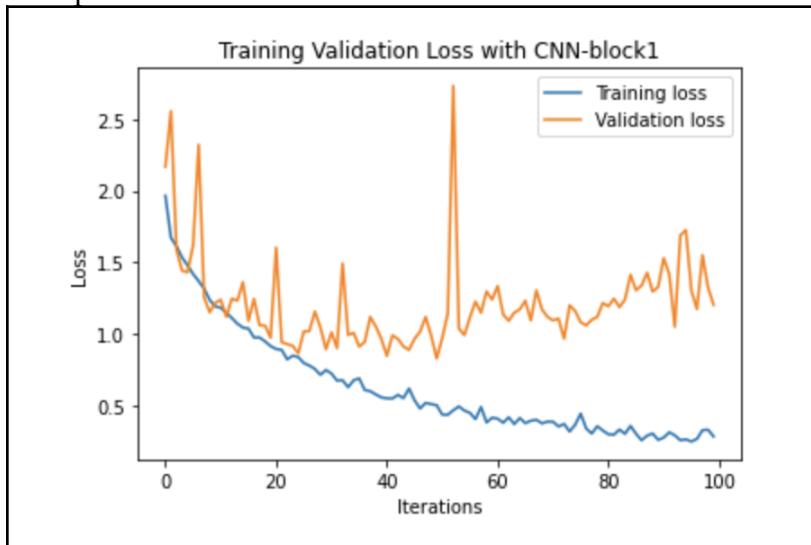
(20): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Dropout(p=0.7, inplace=False)
(1): Linear(in_features=64, out_features=10, bias=True)
)
)

```

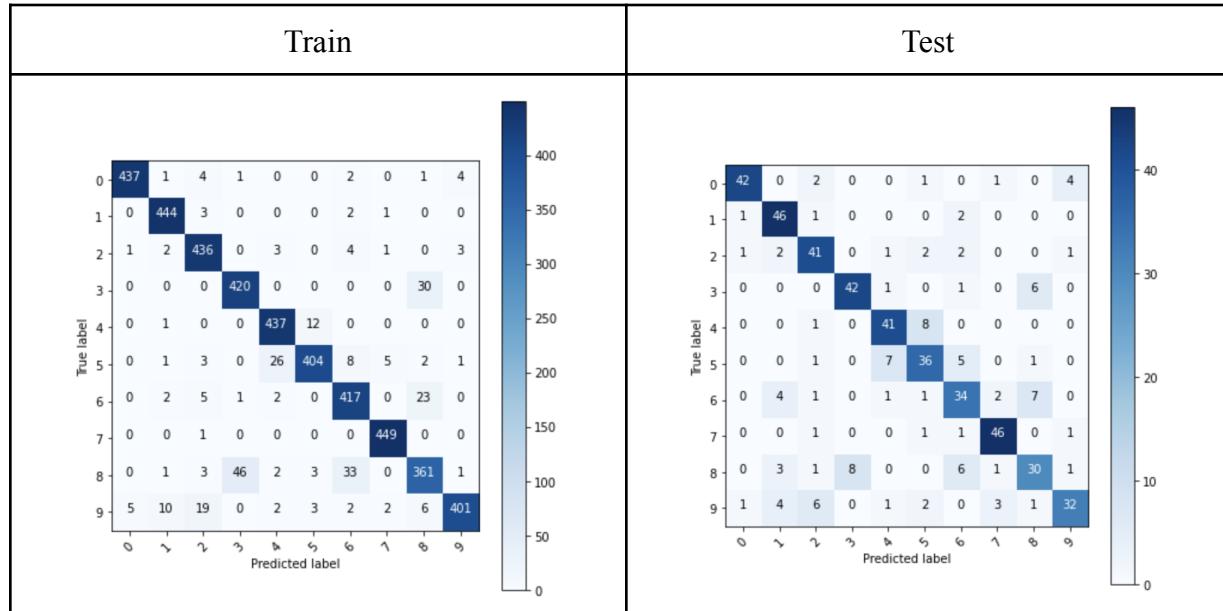
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	93%	78%
Loss	0.213234	0.833252

❖ Loss plot



❖ Confusion Matrix



1.3.3 Dropout after each block

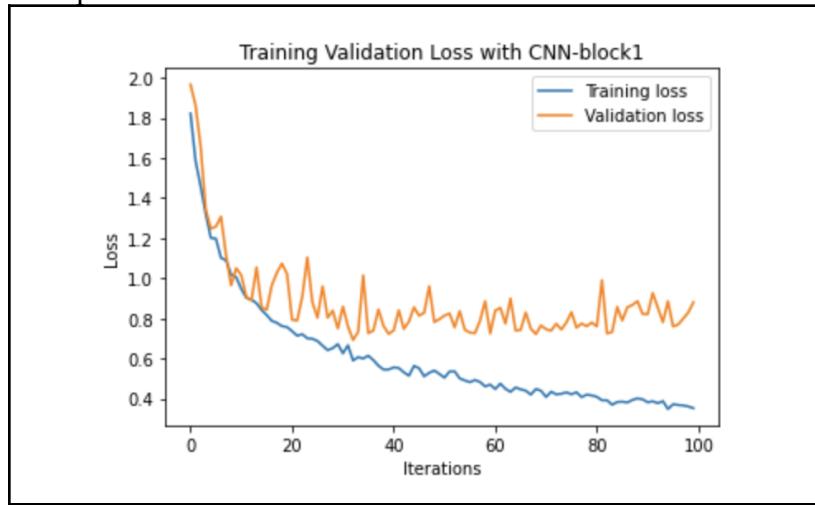
1.3.3.1 Dropout = 0.3

```
Model_B123(
    (features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Dropout(p=0.3, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (13): ReLU(inplace=True)
        (14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (16): ReLU(inplace=True)
        (17): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (18): Dropout(p=0.3, inplace=False)
        (19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (21): ReLU(inplace=True)
        (22): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (23): Dropout(p=0.3, inplace=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)
```

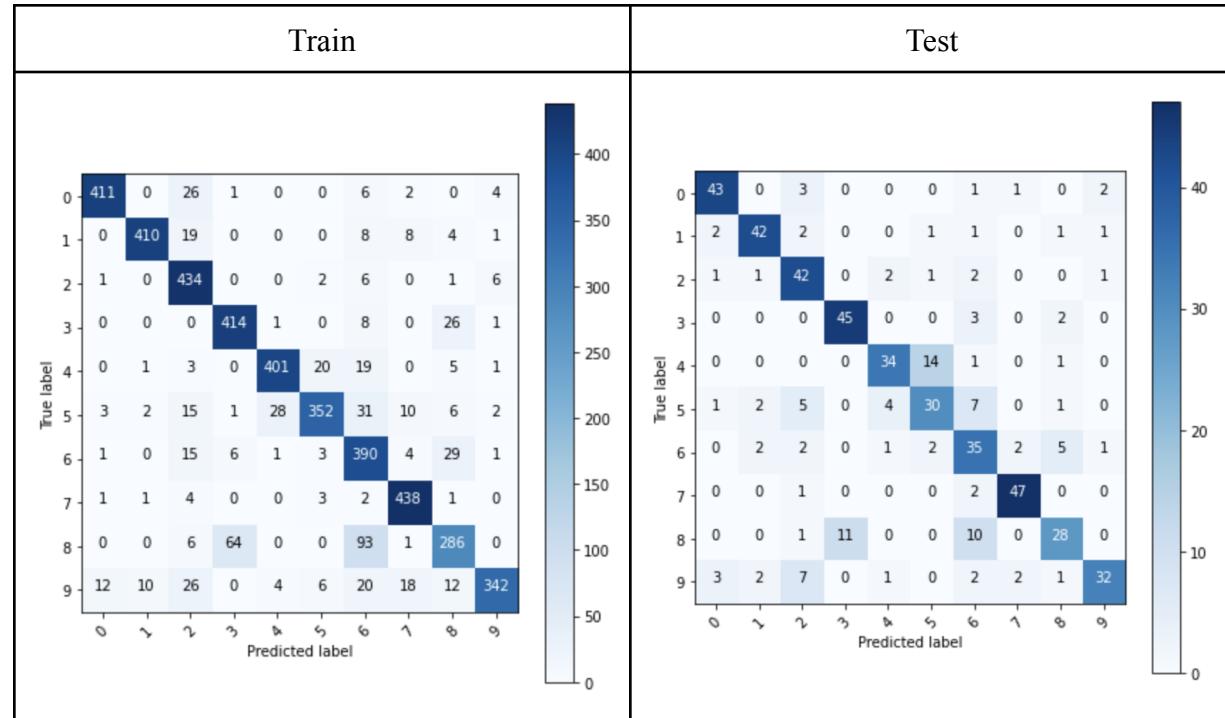
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	86%	75%
Loss	0.415009	0.690495

❖ Loss plot



❖ Confusion Matrix



1.3.3.2 Dropout = 0.5

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```

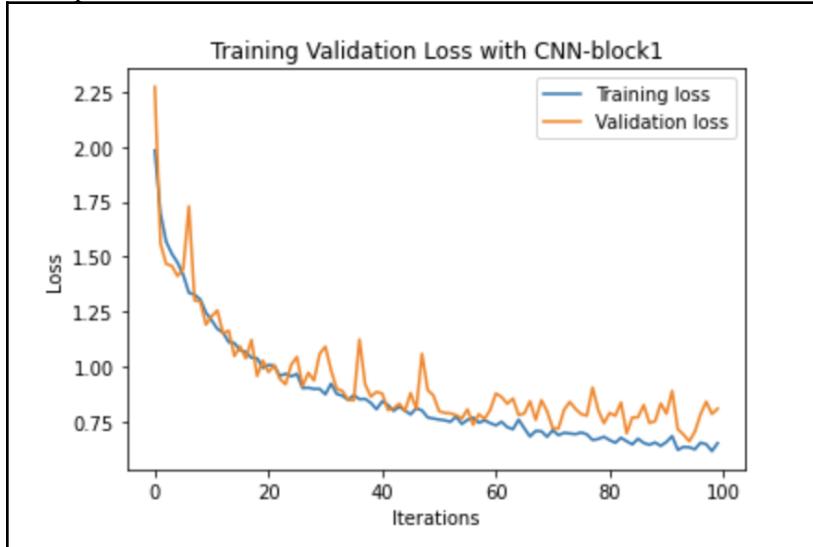
(7): Dropout(p=0.5, inplace=False)
(8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
(9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(10): ReLU(inplace=True)
(11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(13): ReLU(inplace=True)
(14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(16): ReLU(inplace=True)
(17): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(18): Dropout(p=0.5, inplace=False)
(19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(21): ReLU(inplace=True)
(22): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(23): Dropout(p=0.5, inplace=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

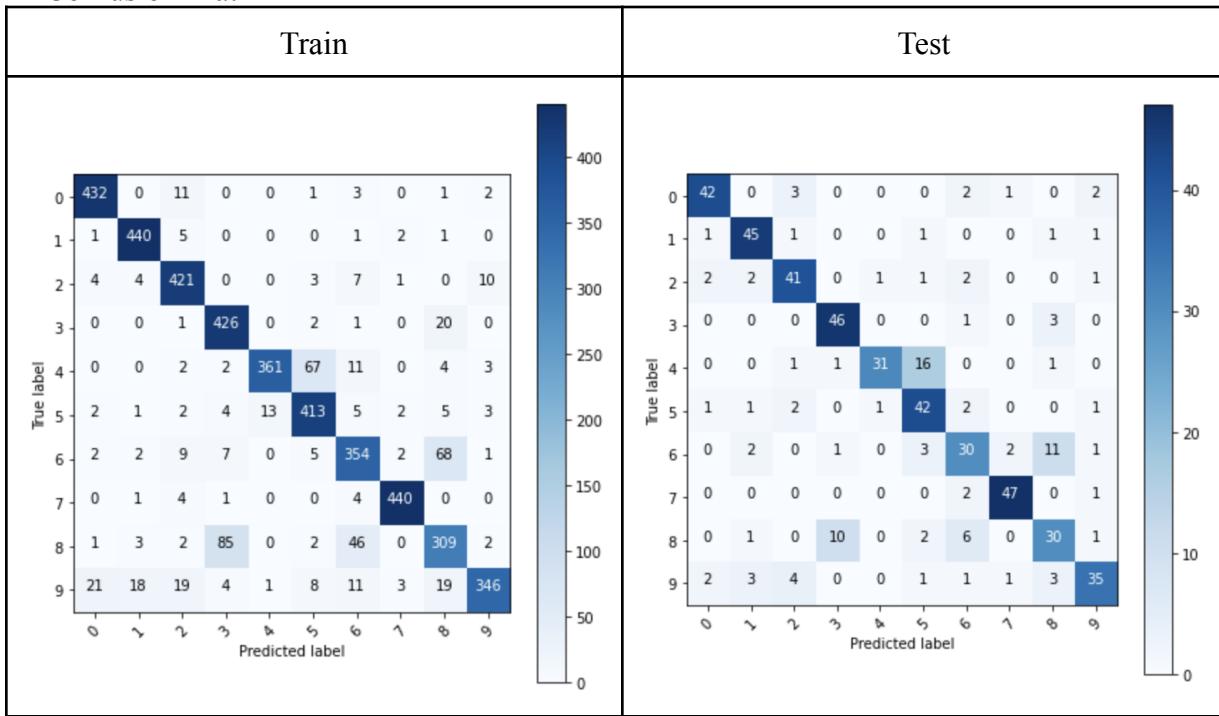
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	87%	77%
Loss	0.380031	0.662487

❖ Loss plot



❖ Confusion Matrix



1.3.3.3 Dropout = 0.7

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Dropout(p=0.7, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (13): ReLU(inplace=True)
        (14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (16): ReLU(inplace=True)
        (17): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (18): Dropout(p=0.7, inplace=False)
        (19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
```

```

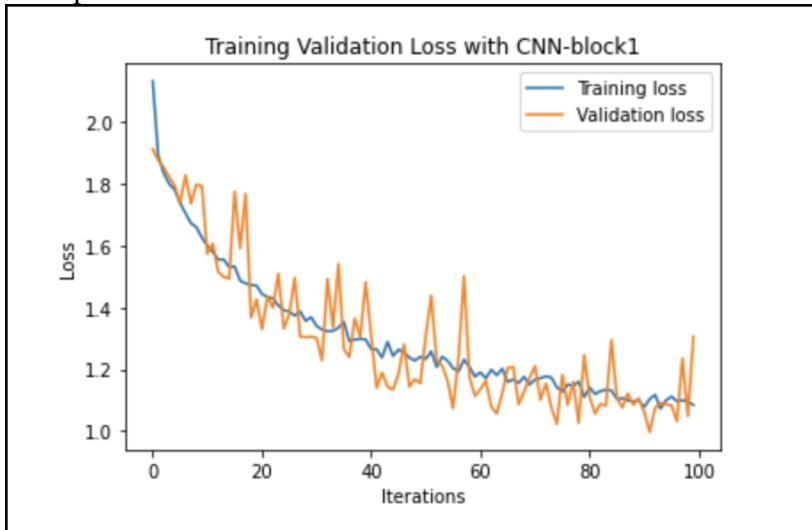
        (20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (21): ReLU(inplace=True)
        (22): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (23): Dropout(p=0.7, inplace=False)
    )
(classifier): Sequential(
    (0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

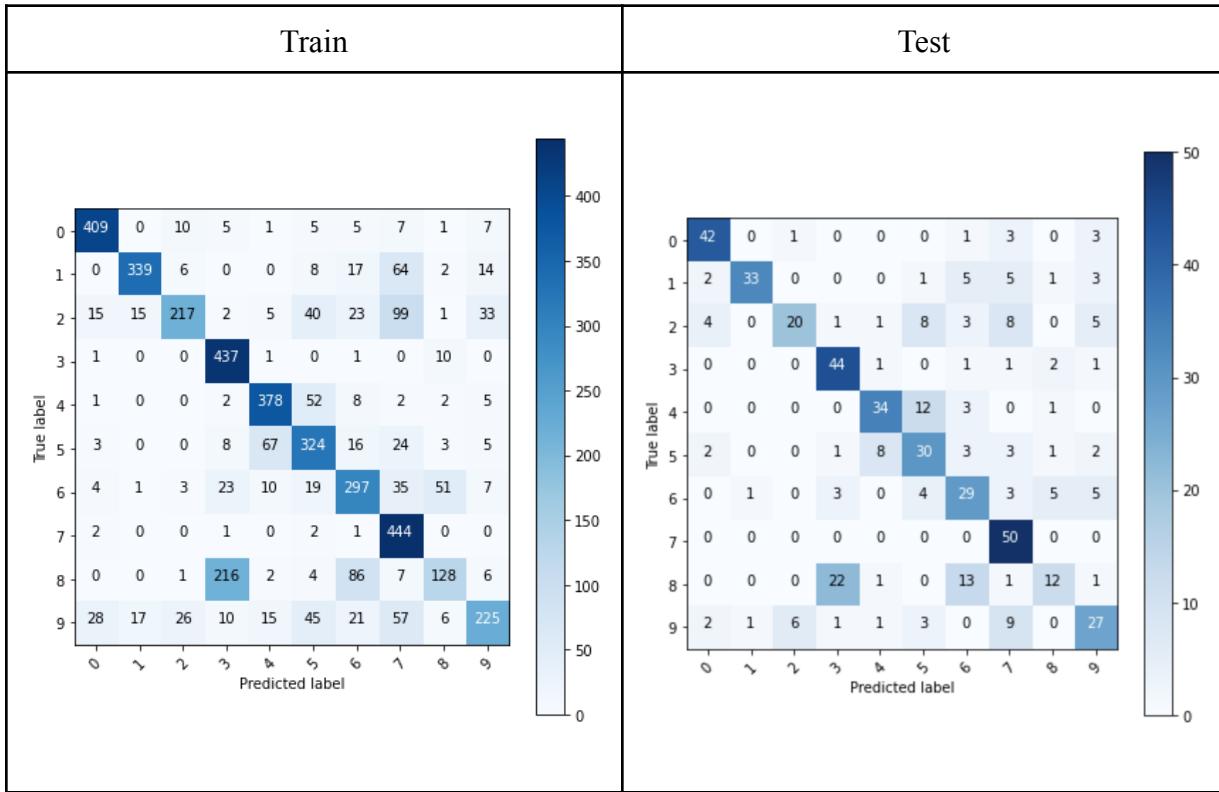
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	71%	64%
Loss	0.834854	0.997905

❖ Loss plot



❖ Confusion Matrix



Analysis:

- Dropout was implemented in all three cases with 3 dropout probabilities in each case.
- Inclusion of dropout prevented overfitting. The extent of the prevention depended on the dropout probability.
- In every case, 0.3 dropout probability still tends to overfit while 0.7 dropout probability tends to underfit. Therefore, 0.5 dropout probability is the best out of the 3 probabilities tested.
- Out of the 3 architectures, best results are obtained if dropout is applied after each block.
- Dropout applied after convolutional layers is not successful, as we might lose important information in images by dropping out some nodes also CNN already has shared weights so dropping out can have adverse effects.
- From the graphs it is clearly visible at 0.3 after some point test loss keeps on increasing while train loss keeps on increasing i.e overfitting. In our case 0.5 works the best with between the blocks.

1.4 Hyperparameters and best architecture

1.4.1 Number of layers (X,Y,Z)

1.4.1.1. X = 2, Y = 2, Z = 2

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
```

```

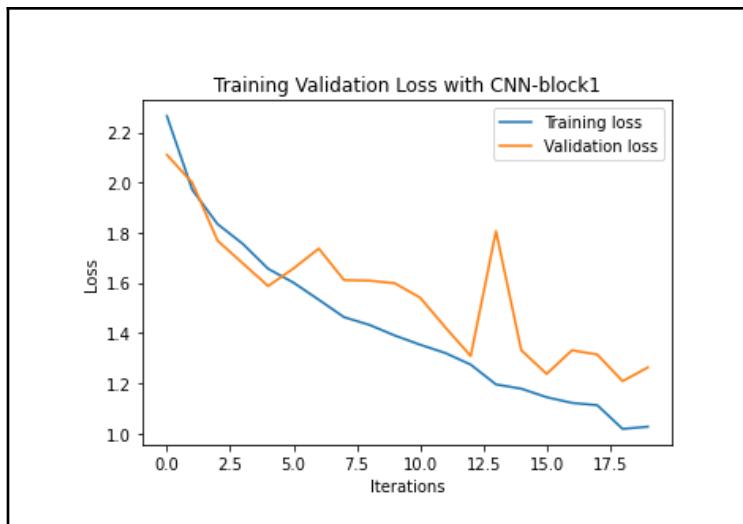
track_running_stats=True)
(5): ReLU(inplace=True)
(6): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=1,
ceil_mode=False)
(7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
(8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(9): ReLU(inplace=True)
(10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(12): ReLU(inplace=True)
(13): MaxPool2d(kernel_size=2, stride=1, padding=0,
dilation=1, ceil_mode=False)
(14): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(15): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(16): ReLU(inplace=True)
(17): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
(18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(19): ReLU(inplace=True)
(20): MaxPool2d(kernel_size=4, stride=3, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Linear(in_features=1600, out_features=512, bias=True)
(1): ReLU(inplace=True)
(2): Linear(in_features=512, out_features=256, bias=True)
(3): ReLU(inplace=True)
(4): Linear(in_features=256, out_features=64, bias=True)
(5): ReLU(inplace=True)
(6): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	65%	55%
Loss	1.011410	1.203690

❖ Loss plot



1.4.1.2. X = 3, Y = 2, Z = 1

```

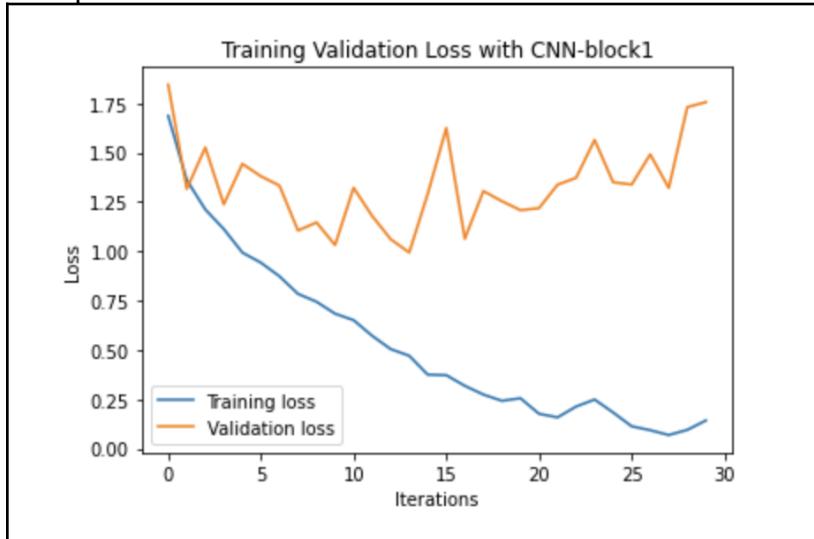
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (7): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (8): ReLU(inplace=True)
        (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (10): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): ReLU(inplace=True)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
        (17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19): ReLU(inplace=True)
        (20): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	87%	66%
Loss	0.376737	0.995357

❖ Loss plot



1.4.1.3. X = 2, Y = 3, Z = 1

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): ReLU(inplace=True)
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): ReLU(inplace=True)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True))
```

```

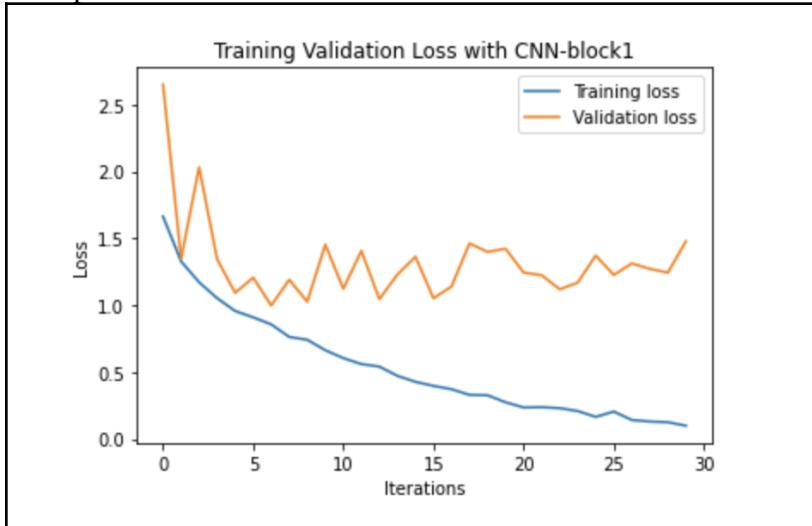
(16) : MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(17) : Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(18) : BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(19) : ReLU(inplace=True)
(20) : MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	74%	64%
Loss	0.718399	0.995629

❖ Loss plot



Best value: X = 2, Y = 3, Z = 1

- The architecture gave best testing accuracy and minimum loss.
- X = 2, Y = 3, Z = 1 and X = 3, Y = 2, Z = 1 are almost similar so we chose first one.
- Keeping higher X value worked better as it helped in better feature extraction. Hence, X = 1 gave low accuracy while X = 2 and X = 3 gave better accuracy.

1.4.2 Batch Normalization (With Max Pooling)

1.4.2.1. Without Batch Normalization

```

Model_B123(
(features): Sequential(

```

```

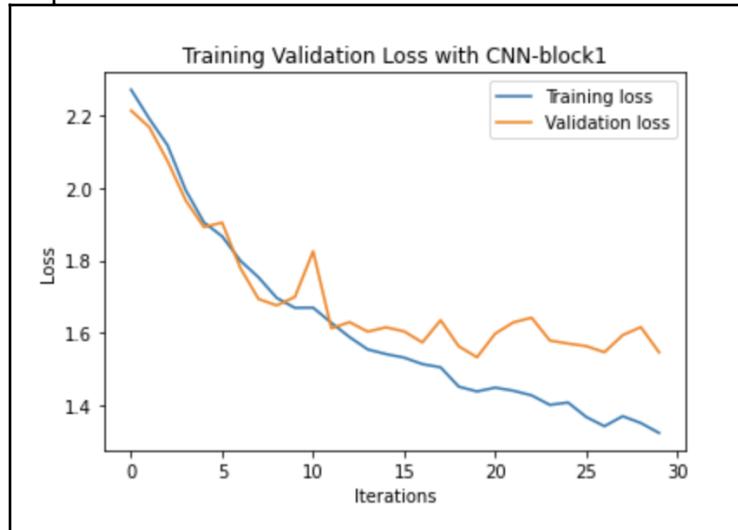
(0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
(1): ReLU(inplace=True)
(2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
(5): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
(6): ReLU(inplace=True)
(7): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(8): ReLU(inplace=True)
(9): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(10): ReLU(inplace=True)
(11): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(12): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(13): ReLU(inplace=True)
(14): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	52%	45%
Loss	1.349049	1.532184

❖ Loss plot



1.4.2.2. With Batch Normalization

```

Model_B123(
(features): Sequential(

```

```

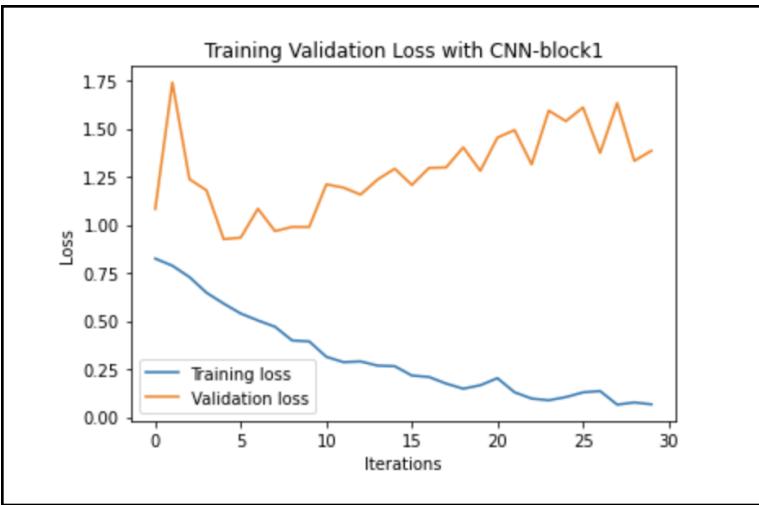
(0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
(1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(2): ReLU(inplace=True)
(3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
(4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(5): ReLU(inplace=True)
(6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
(7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
(8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(9): ReLU(inplace=True)
(10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(12): ReLU(inplace=True)
(13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(15): ReLU(inplace=True)
(16): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(18): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(19): ReLU(inplace=True)
(20): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	81%	69%
Loss	0.521781	0.919671

❖ Loss plot



Best value: The model performed better with batch normalization as it normalizes value in a range (0,1) which helps the model learn faster and perform better.

1.4.3 Pooling

1.4.3.1. Max Pooling

The model is the same as that in section 1.4.2.2 (with batch normalization and max pooling).

1.4.3.2. Average Pooling

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (7): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (9): ReLU(inplace=True)
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): ReLU(inplace=True)
        (13): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (15): ReLU(inplace=True)
        (16): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (17): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
```

```

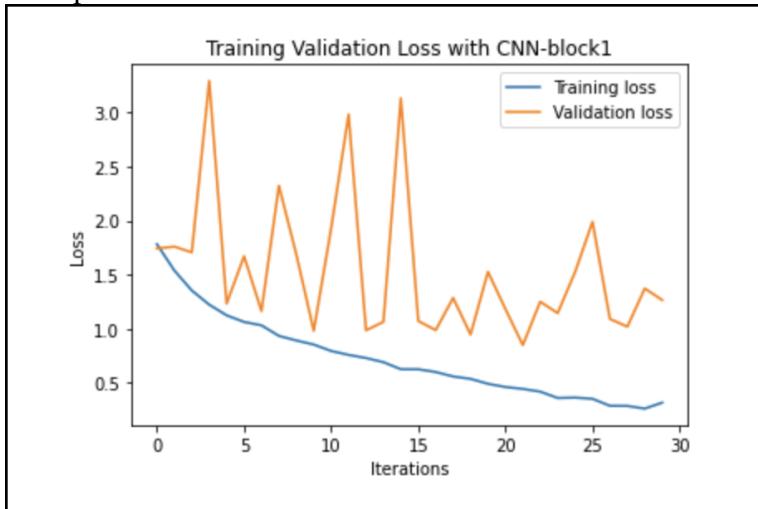
        (18) : BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (19) : ReLU(inplace=True)
        (20) : AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
(classifier): Sequential(
    (0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	88%	72%
Loss	0.347223	0.853640

❖ Loss plot



Best value: The model performed better with Average Pooling because Average Pooling considers every feature of the image. Looking at the dataset images, the sharpness and intensity in different areas of images is different. Therefore, average pooling takes into account all of them rather than the highest (like Max Pooling does).

1.4.4 Dropout

From the analysis of section 1.3, the optimal dropout probability was chosen as 0.5 and the dropout was applied after each block for best performance.

```

Model_B123(
(features): Sequential(
(0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
(1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
)

```

```

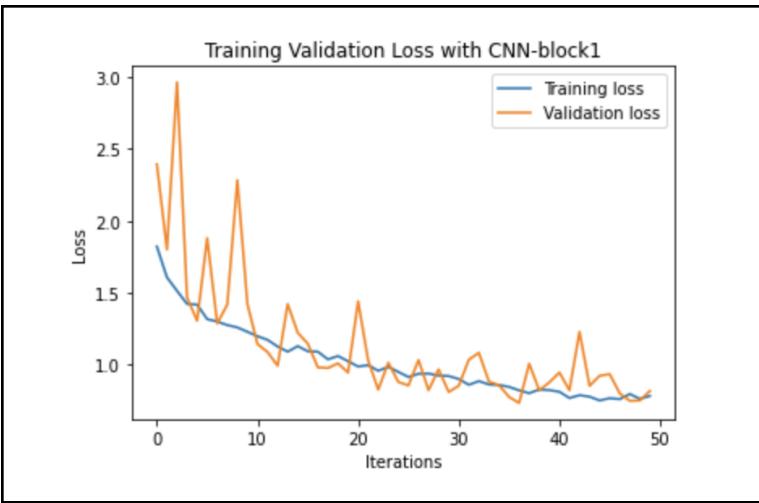
(2): ReLU(inplace=True)
(3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
(4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(5): ReLU(inplace=True)
(6): AvgPool2d(kernel_size=2, stride=2, padding=0)
(7): Dropout(p=0.5, inplace=False)
(8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
(9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(10): ReLU(inplace=True)
(11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(13): ReLU(inplace=True)
(14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(16): ReLU(inplace=True)
(17): AvgPool2d(kernel_size=2, stride=2, padding=0)
(18): Dropout(p=0.5, inplace=False)
(19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(21): ReLU(inplace=True)
(22): AvgPool2d(kernel_size=2, stride=2, padding=0)
(23): Dropout(p=0.5, inplace=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	81%	73%
Loss	0.529796	0.738360

❖ Loss plot



1.4.5 Optimizer and weight decay

Adam optimizer has been chosen for the model as Adam generally is the optimal choice because of its adaptive learning rate property.

1.4.5.1 With weight decay

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (7): Dropout(p=0.5, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (13): ReLU(inplace=True)
        (14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (16): ReLU(inplace=True)
        (17): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (18): Dropout(p=0.5, inplace=False)
        (19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (21): ReLU(inplace=True)
        (22): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (23): Dropout(p=0.5, inplace=False)
)
```

```

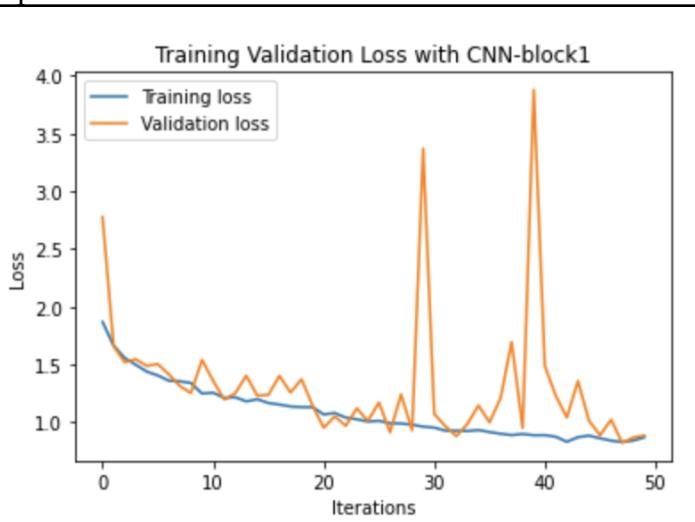
(classifier): Sequential(
    (0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	78%	71%
Loss	0.598614	0.823383

❖ Loss plot



1.4.5.2. Without Weight Decay

```

Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (7): Dropout(p=0.5, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (13): ReLU(inplace=True)
        (14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
)
)

```

```

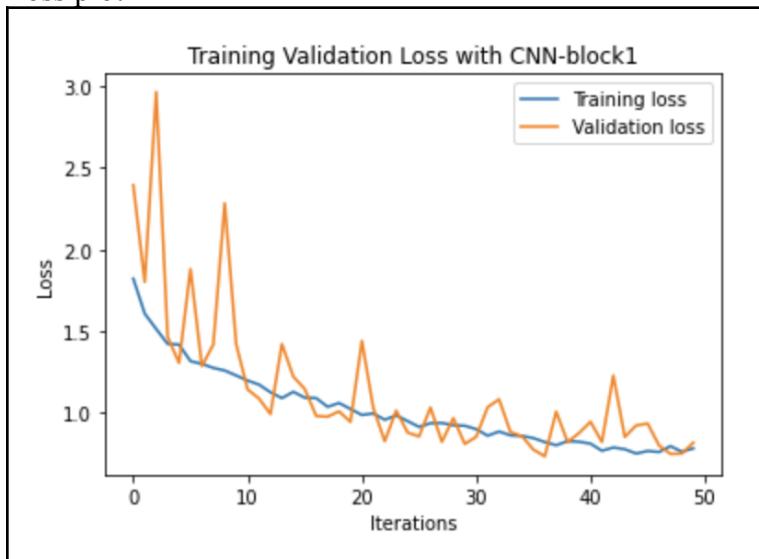
(15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(16): ReLU(inplace=True)
(17): AvgPool2d(kernel_size=2, stride=2, padding=0)
(18): Dropout(p=0.5, inplace=False)
(19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(21): ReLU(inplace=True)
(22): AvgPool2d(kernel_size=2, stride=2, padding=0)
(23): Dropout(p=0.5, inplace=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	81%	73%
Loss	0.529796	0.738360

❖ Loss plot



Best value: Adam optimizer performed well. Using weight decay did not improve performance. If the value of weight decay was kept small, no improvement was seen whereas if weight decay was kept large then the model underfit. Therefore, no weight decay was done.

1.4.6 Learning Rate

1.4.6.1 Learning rate = 0.01

The results are the same as section 1.4.5.2 as the learning rate was 0.01 by default in all of the above results.

1.4.6.2. Learning rate = 0.001

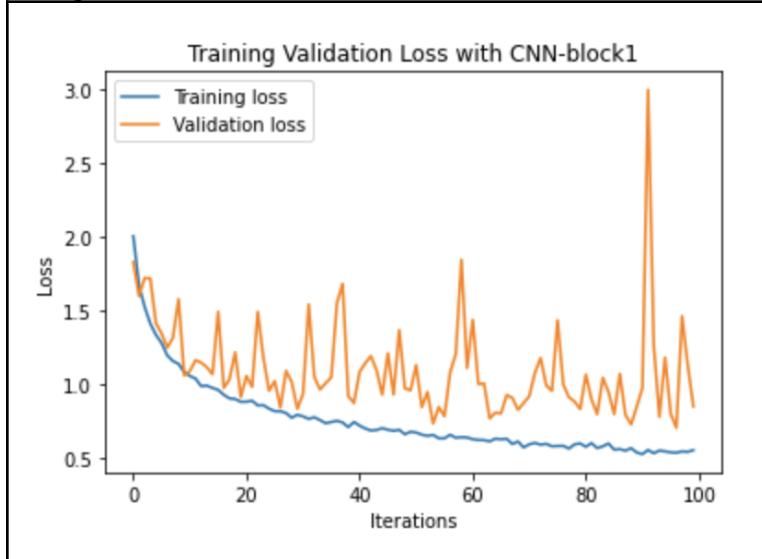
```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (7): Dropout(p=0.5, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (13): ReLU(inplace=True)
        (14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (16): ReLU(inplace=True)
        (17): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (18): Dropout(p=0.5, inplace=False)
        (19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (21): ReLU(inplace=True)
        (22): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (23): Dropout(p=0.5, inplace=False)
    )
    classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)
```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
-------------	-------	------

Accuracy	86%	76%
Loss	0.370939	0.703236

❖ Loss plot



Best value: Learning rate = 0.001 performed better as, at value 0.01, the model oscillated around the minima and could not reach it which indicated that the value had to be decreased.

1.4.7 Data Augmentation

1.4.7.1 Without Data Augmentation

The results are the same as section 1.4.6.2 as the model in the section did not use data augmentation and had optimal hyperparameters.

1.4.7.2. With Data Augmentation

Random Horizontal flips were used for data augmentation.

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (7): Dropout(p=0.5, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True),
```

```

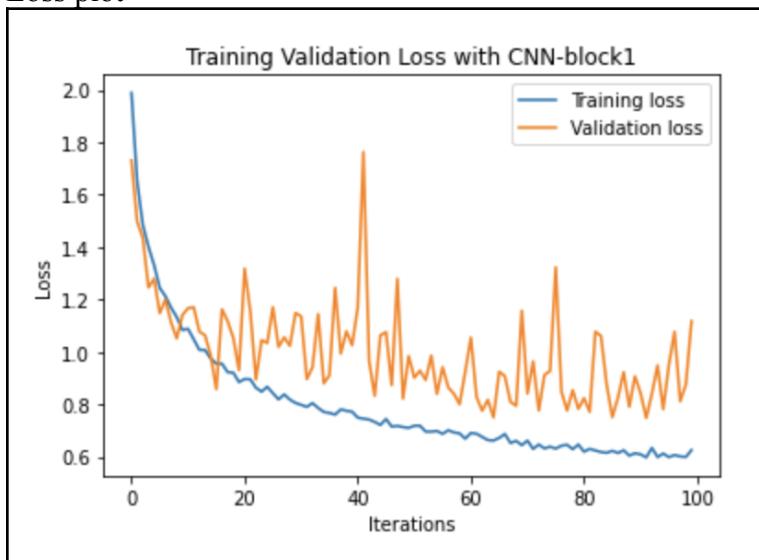
track_running_stats=True)
(13): ReLU(inplace=True)
(14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(16): ReLU(inplace=True)
(17): AvgPool2d(kernel_size=2, stride=2, padding=0)
(18): Dropout(p=0.5, inplace=False)
(19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(21): ReLU(inplace=True)
(22): AvgPool2d(kernel_size=2, stride=2, padding=0)
(23): Dropout(p=0.5, inplace=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)
)

```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	83%	74%
Loss	0.461166	0.749098

❖ Loss plot



Best value: Data augmentation improved the performance of the model by making it more generalised. It made the model more generic and helped the model make more accurate predictions.

1.4.8 Epochs

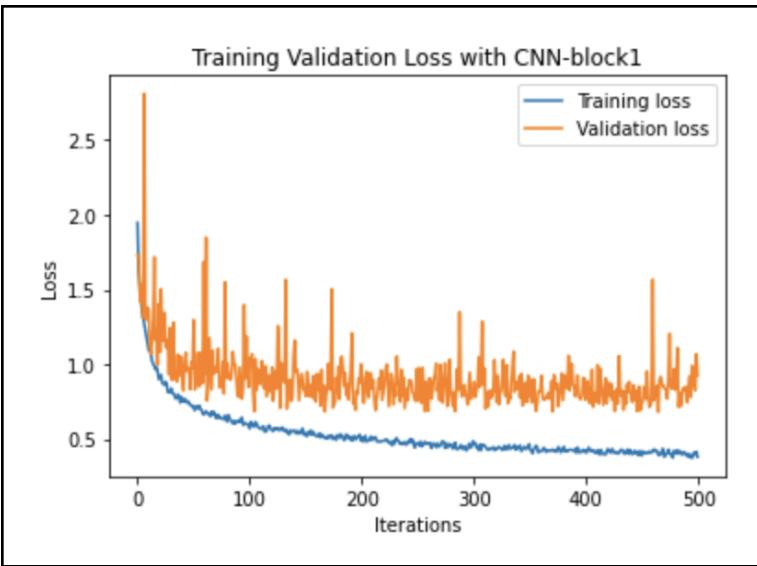
The model was run for 500 epochs.

```
Model_B123(
    (features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (7): Dropout(p=0.5, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (13): ReLU(inplace=True)
        (14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (16): ReLU(inplace=True)
        (17): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (18): Dropout(p=0.5, inplace=False)
        (19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (21): ReLU(inplace=True)
        (22): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (23): Dropout(p=0.5, inplace=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)
```

❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	92%	78%
Loss	0.207575	0.683457

❖ Loss plot



Best value: From the plot and model performance, it was found that optimal epoch value was 100. After that, the model learning saturates i.e there is no significant change in test loss.

Best model:

Layers: X = 2, Y = 3, Z = 1

Batch Normalization

Average Pooling

Dropout = 0.5 after each block

Adam optimizer

No weight decay

Learning rate = 0.001

Data Augmentation - Random Horizontal Flips

Epochs = 100

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (7): Dropout(p=0.5, inplace=False)
        (8): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): ReLU(inplace=True)
        (11): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (12): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (13): ReLU(inplace=True))
```

```

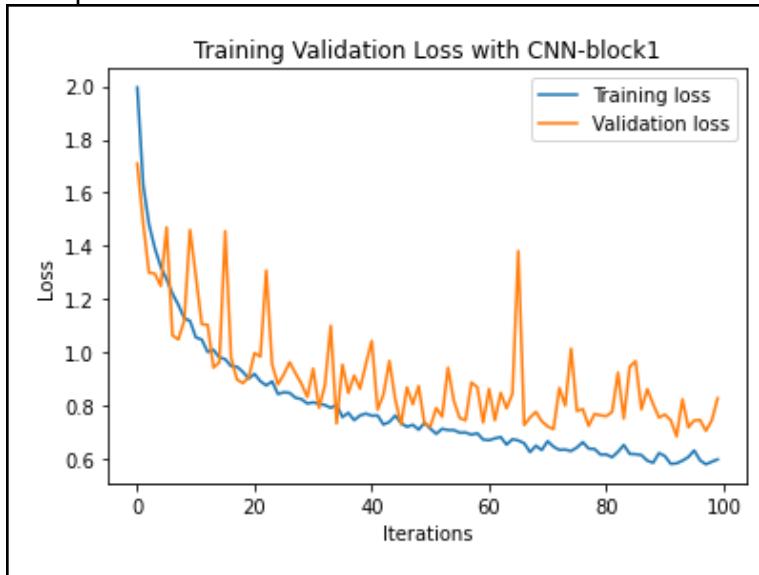
(14): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
(15): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(16): ReLU(inplace=True)
(17): AvgPool2d(kernel_size=2, stride=2, padding=0)
(18): Dropout(p=0.5, inplace=False)
(19): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
(20): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(21): ReLU(inplace=True)
(22): AvgPool2d(kernel_size=2, stride=2, padding=0)
(23): Dropout(p=0.5, inplace=False)
)
(classifier): Sequential(
(0): Linear(in_features=64, out_features=10, bias=True)
)
)

```

❖ Accuracy and Loss

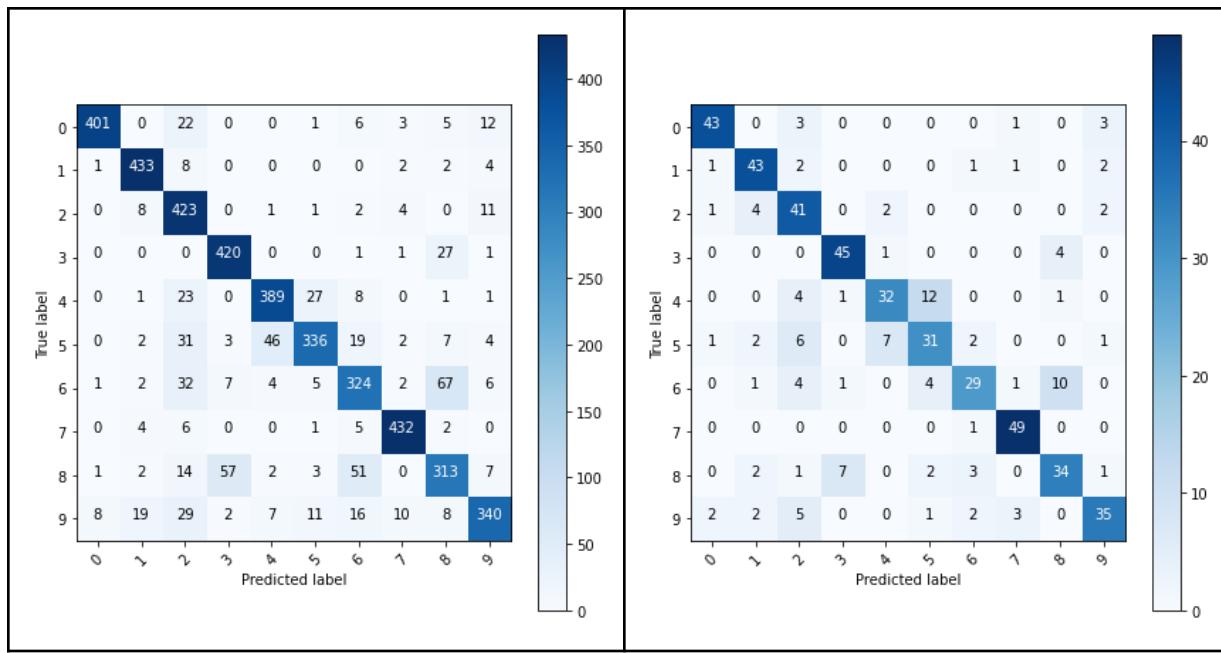
Block 1,2,3	Train	Test
Accuracy	84%	76%
Loss	0.420687	0.682963

❖ Loss plot



❖ Confusion Matrix

Train	Test



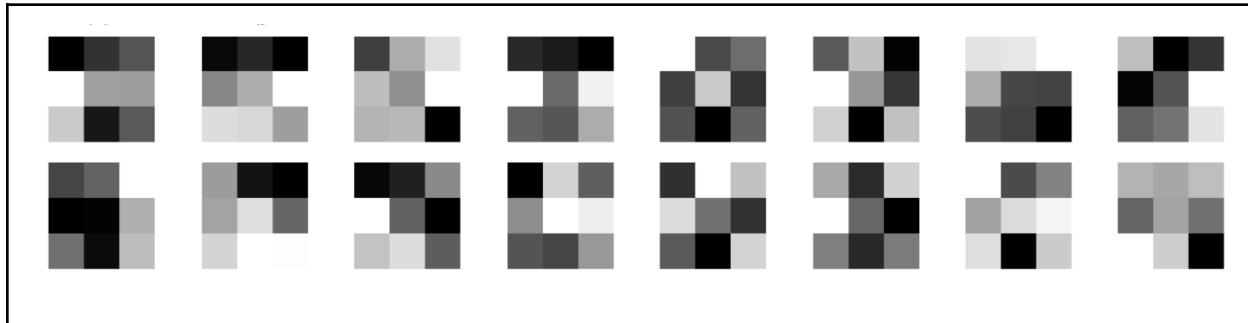
1.5 Visualize Convolutional Filters and Feature Maps

1.5.1 Visualize Convolutional Filters

- Block 1 filter:

```
CONV: Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1)) =====> SHAPE:  
torch.Size([16, 16, 3, 3])
```

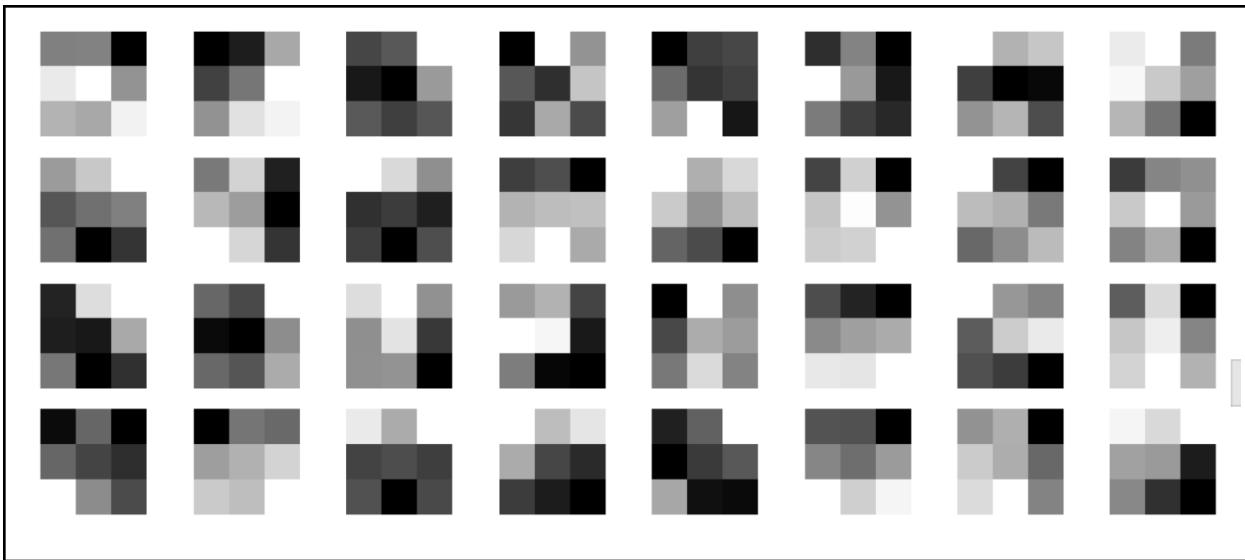
- ❖ There will be 16 filters for block 1 of size (3x3).
- ❖ The filters are as follows:



- Block 2 filter:

```
CONV: Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1)) =====> SHAPE:  
torch.Size([32, 32, 3, 3])
```

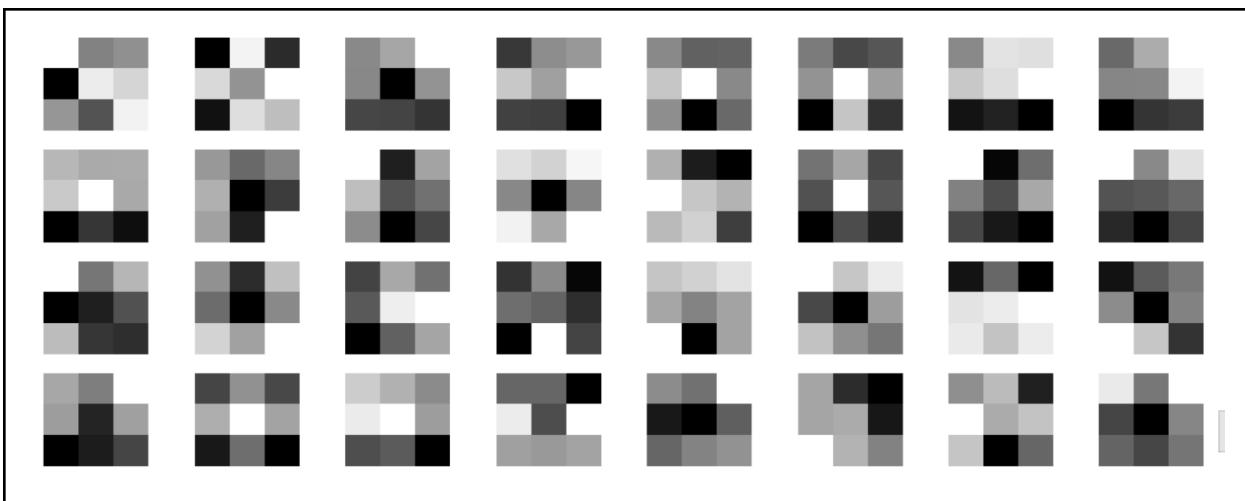
- ❖ There will be 32 filters of size (3x3)
- ❖ The filters are as follows:



- Block 3 filter:

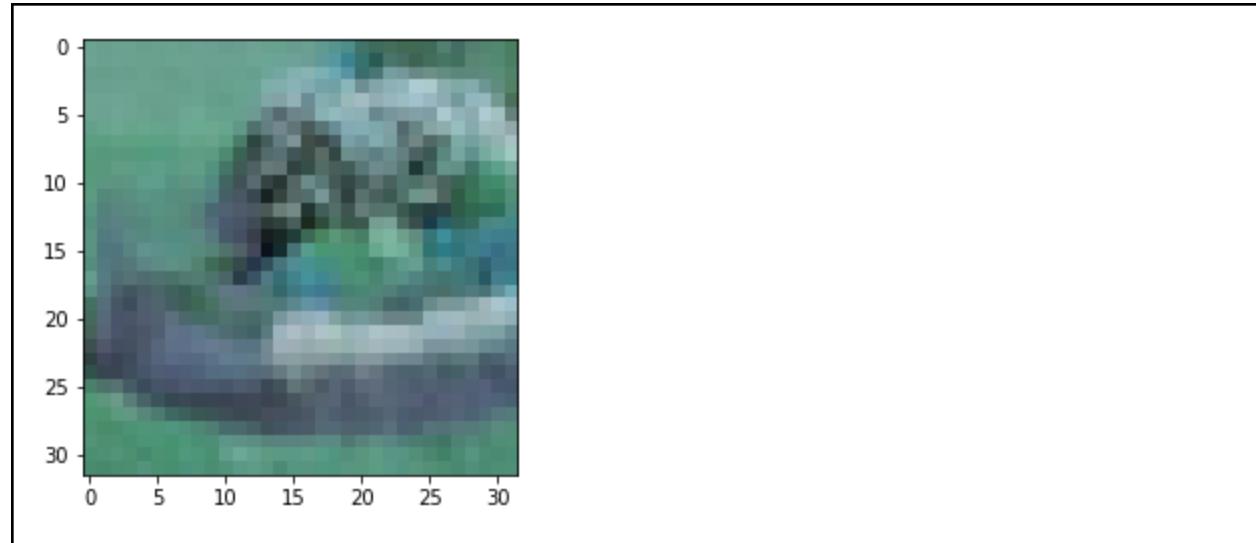
CONV: Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1)) =====> SHAPE:
torch.Size([64, 32, 3, 3])

- ❖ There will be 64 filters of size (3x3)
- ❖ Some of the filters are as follows:

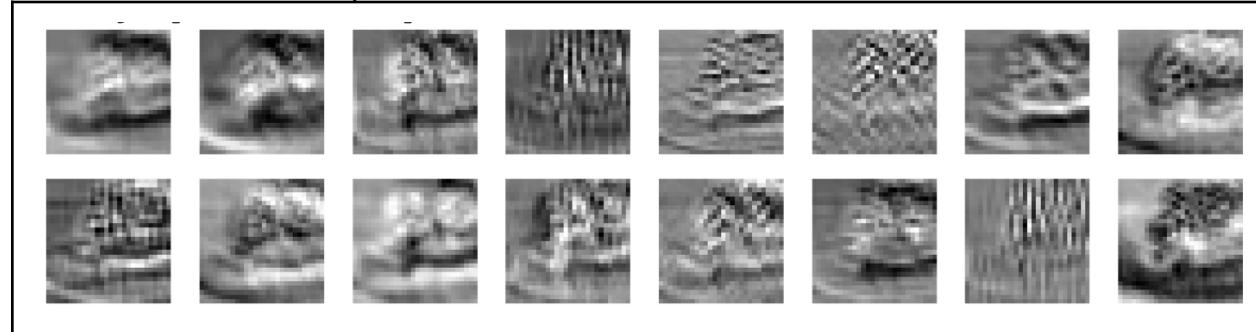


1.5.2 Visualize Feature Maps

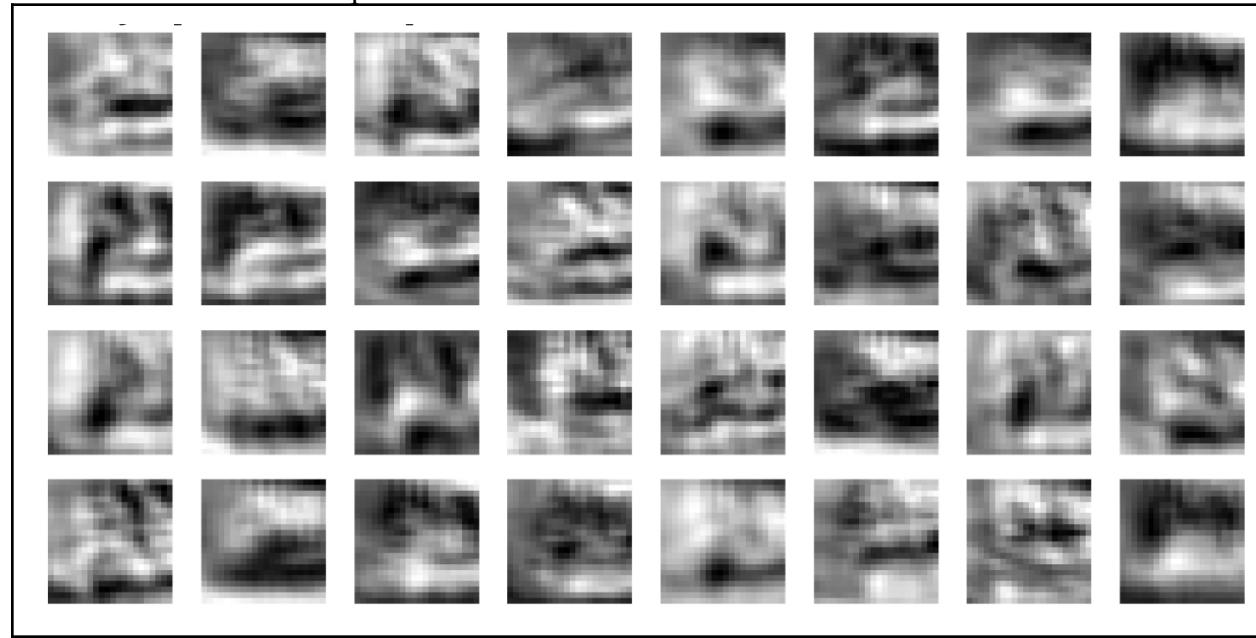
For the following image feature maps visualised are:



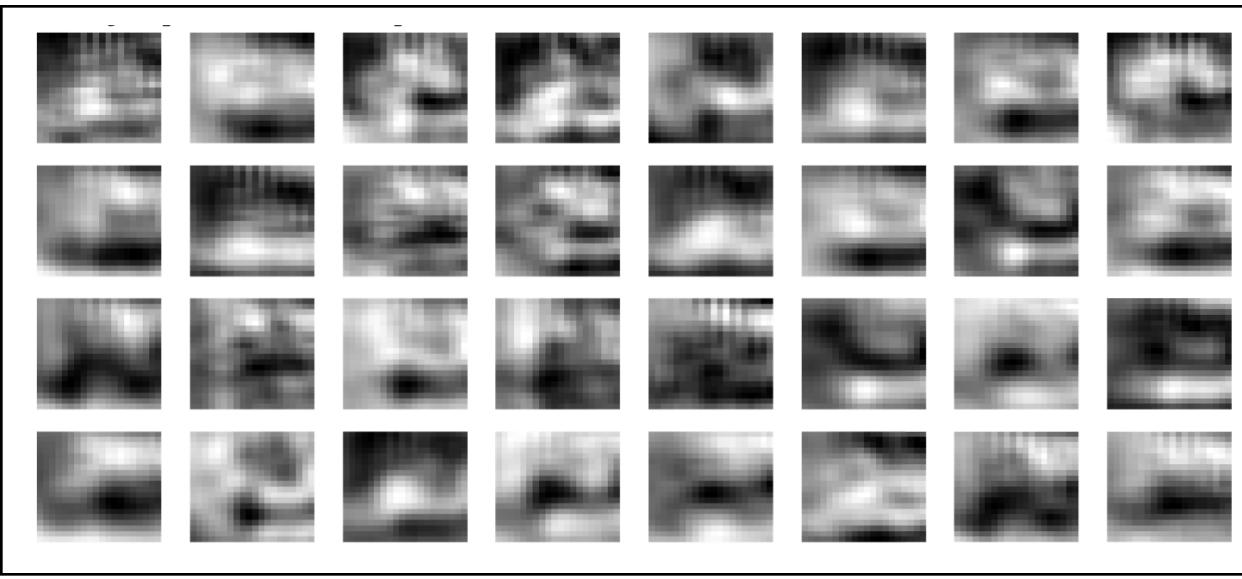
- Block 1 feature maps:



- Block 2 feature maps:



- Some of the block 3 feature maps:



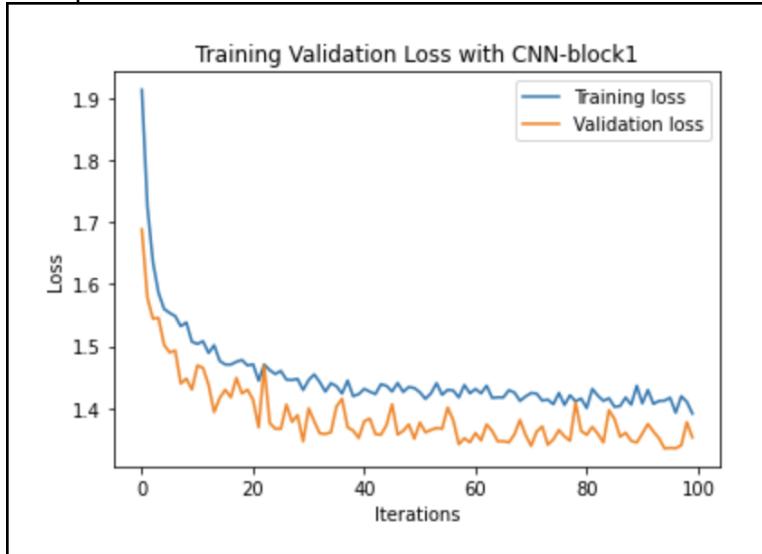
1.6 Best Model Without Activation Function

```
Model_B123(
    features): Sequential(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
        (3): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (4): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (5): Dropout(p=0.5, inplace=False)
        (6): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (8): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (9): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (10): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
        (11): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (12): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (13): Dropout(p=0.5, inplace=False)
        (14): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
        (15): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (16): AvgPool2d(kernel_size=2, stride=2, padding=0)
        (17): Dropout(p=0.5, inplace=False)
    )
    (classifier): Sequential(
        (0): Linear(in_features=64, out_features=10, bias=True)
    )
)
```

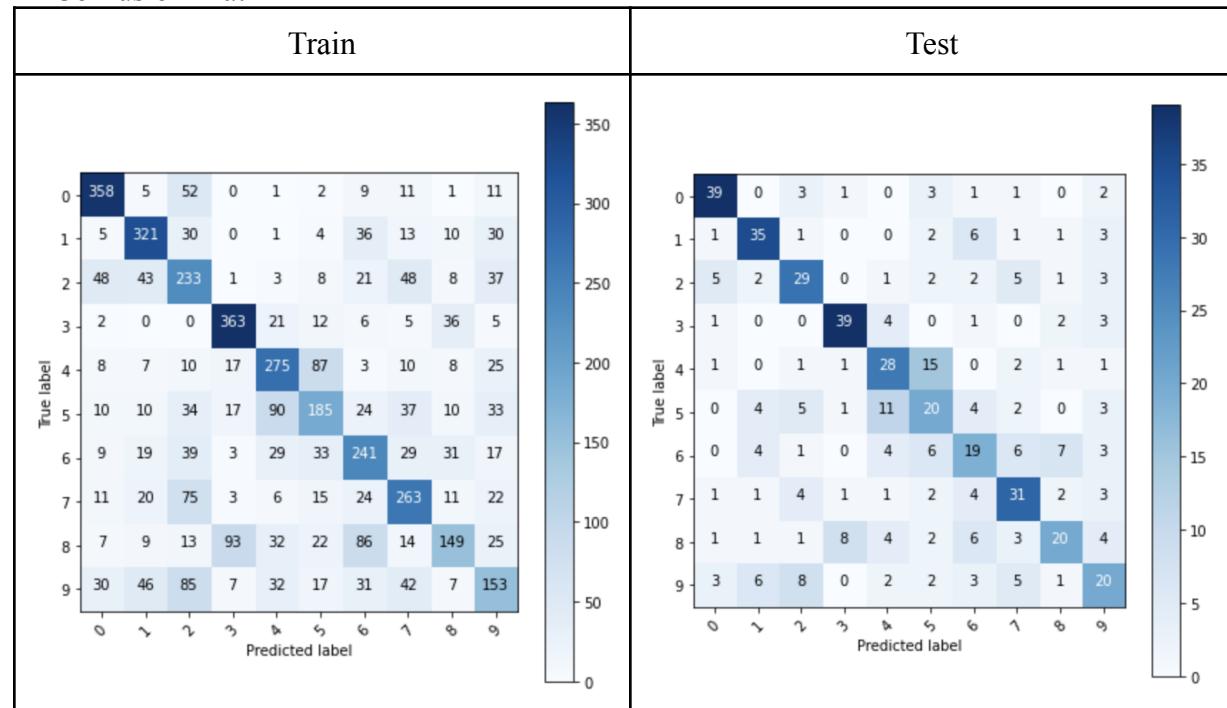
❖ Accuracy and Loss

Block 1,2,3	Train	Test
Accuracy	56%	56%
Loss	1.278515	1.338608

❖ Loss plot



❖ Confusion Matrix



Analysis:

- Removal of activation function has a significant drop in accuracy (from 76% to 56%) on test data.
- The drop in performance is because, without activation function, the model cannot capture non-linearity in the data.
- The model also might have a vanishing/exploding gradient problem without activation function.

2. Sentence Classification

Dataset: TREC dataset

- The dataset is based on task of question classification.
- It contains 6-coarse level classes and 47 fine-level classes.
- Format: `<coarse - label : fine-label<space><question_text>>`.

2.2 Pre-processing

- The data was pre processed and only 6 coarse-level classes were extracted along with the text data.
- The labels were encoded using LabelEncoder. The resultant classes obtained were {0,1,2,3,4,5}.
- The text from the document was pre-processed as follows:
 - Text was converted to lowercase.
 - Punctuations were removed.
 - Stopwords were removed.
 - Single character words and '\x' type words were removed.
- After doing the above steps, the sentences were padded to equal length for consistency in data.
- Vocabulary of words was generated from the obtained data after doing the above steps.

2.3 Train-Test Split

- Data was split into 80-20 ratio of train-test using `train_test_split` from sklearn.

2.4 Effect of Input Word Vector

- Out of word2vec and GloVe word embedding, the latter was chosen.
- GloVe embedding of 300-dimension was used.
- GloVe is a model based on global word-word co-occurrence statistics.

2.5 Effect of Pooling Strategy

- To check the effect of pooling strategy, pooling region size and different pooling strategies were explored.
- 1-max pooling was implemented and performed significantly well, giving an accuracy of 88.6% on test data.
- Further, effects of local regions, different equal sized local regions were implemented. Therefore, k -local pooling was done for $k = \{3,5,10,15\}$.
- It was inferred from the results that for lower values of local region sizes, accuracy was almost the same as 1-max, but 1-max pooling outperformed them.
- It could be said that 1 local region of feature vector holds enough information for the model to perform well.
- Next, max pooling was performed for different region sizes. The region size $k = \{3,5,10,15\}$.
- The results showed that 1-max pooling again outperformed the k -max pooling approach for larger values ($k > 1$).
- Finally, average pooling was done instead of max pooling. The local region sizes were again taken as $\{3,5,10,15\}$.
- The results indicated that local average pooling performed worse than the max pooling.

- Overall, 1-max pooling performed best out of all the strategies and local pooling had comparable results to 1-max pooling.

Best Models

As discussed in the paper, 1-max pooling has shown the best results. We have studied the effect of pooling using different strategies. We have fixed the values for local-regional, k-max and local averaging pooling and drawn a comparison using:

- Accuracy
- Confusion Matrix
- ROC Curve
- F1-Measure

Below are the best results for each type of pooling strategies mentioned above.

Local Regional Pooling

5-local regional pooling has shown the best results

K-Max Pooling

k-max pooling has not worked quite well on the given dataset and among the k values 3, 5, 10and 15: 3-max regional pooling has shown the best results.

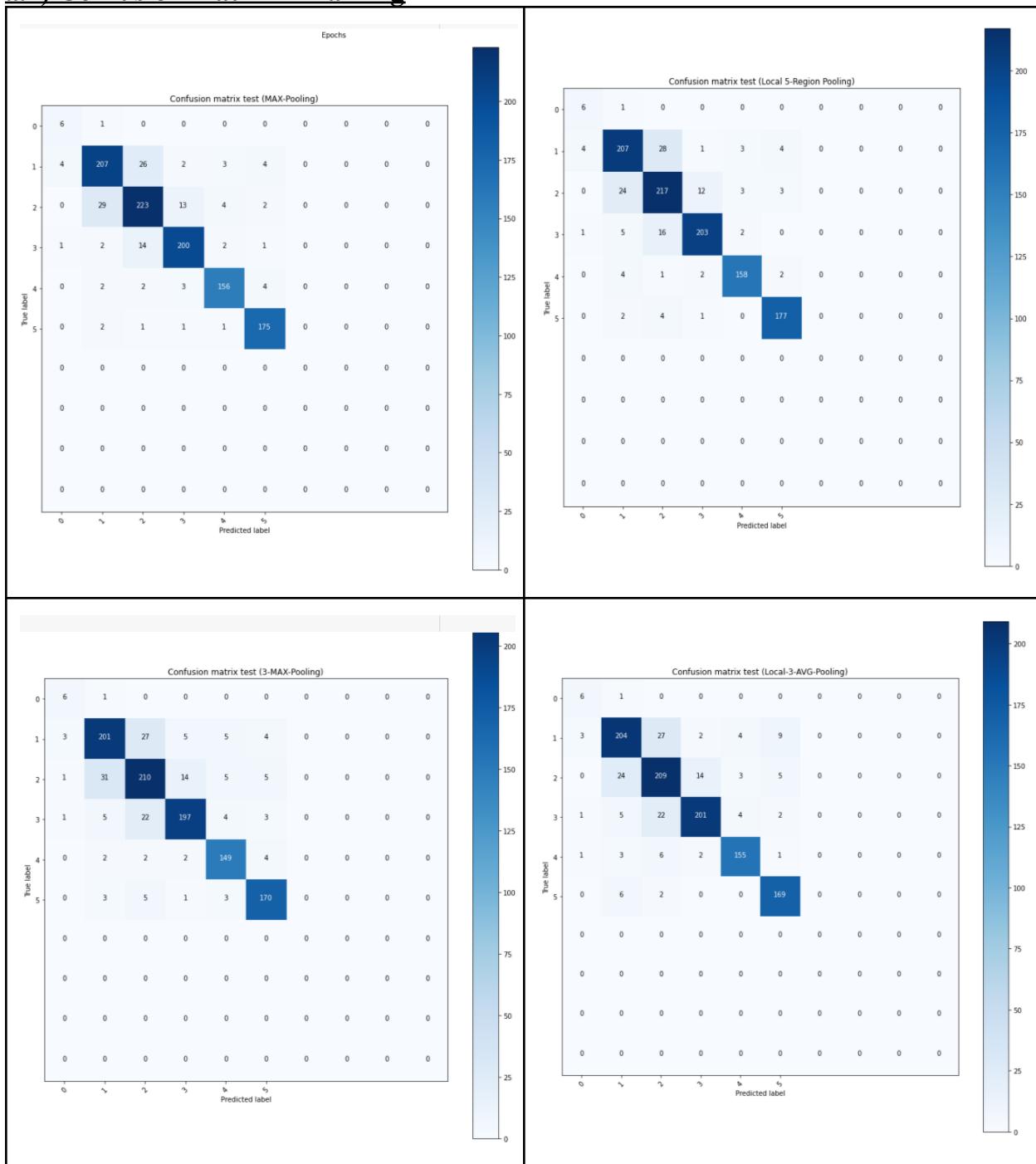
Local Average pooling

Like k-max pooling it has also not fared as well as 1-max pooling and local regional pooling.

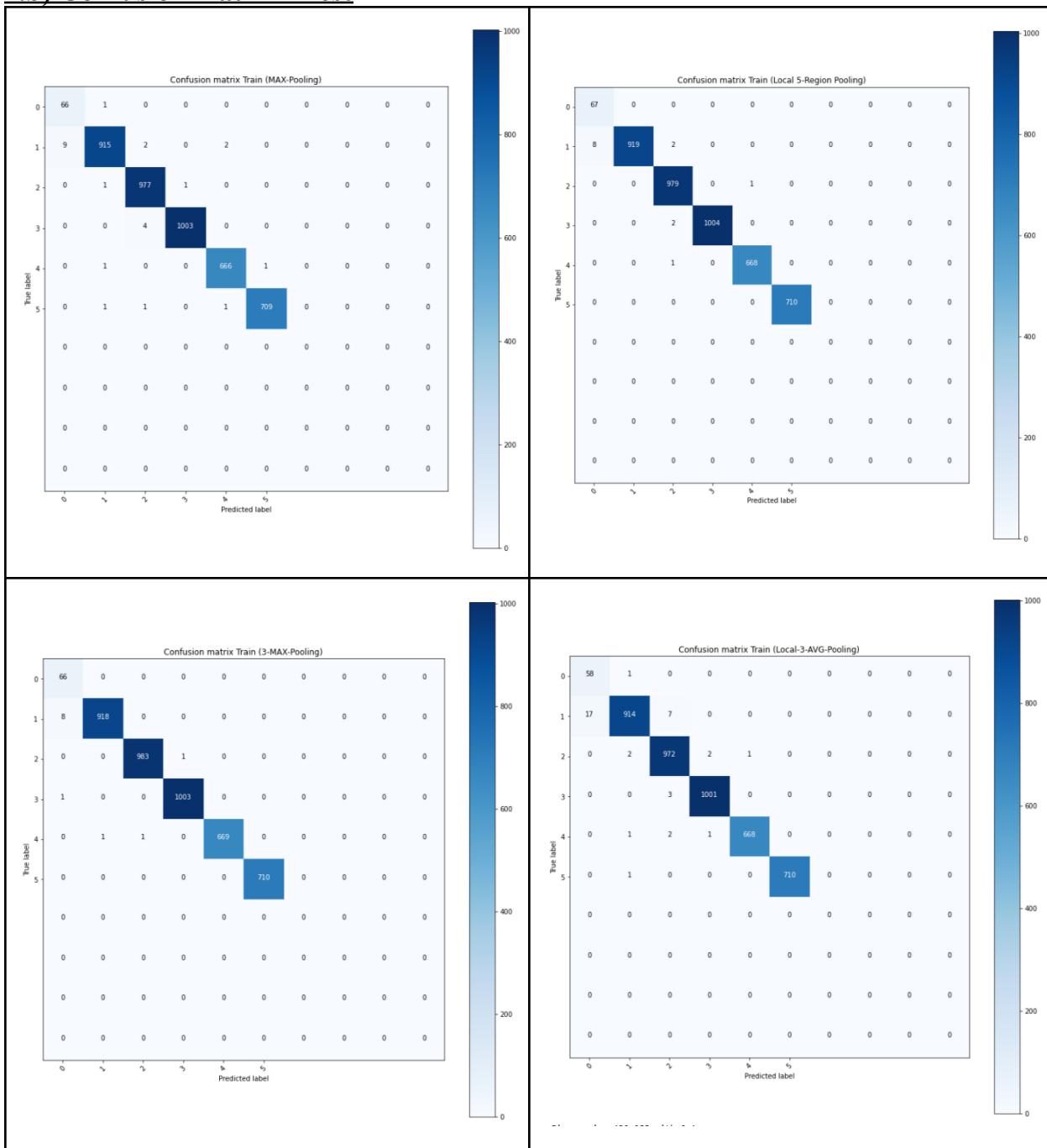
We have fixed the below as described in the paper while studying the effect of different pooling strategies.

Description	Values
input word vectors	Google word2vec
filter region size	(3,4,5)
feature maps	100
activation function	ReLU
pooling	1-max pooling
dropout rate	0.5
l_2 norm constraint	3

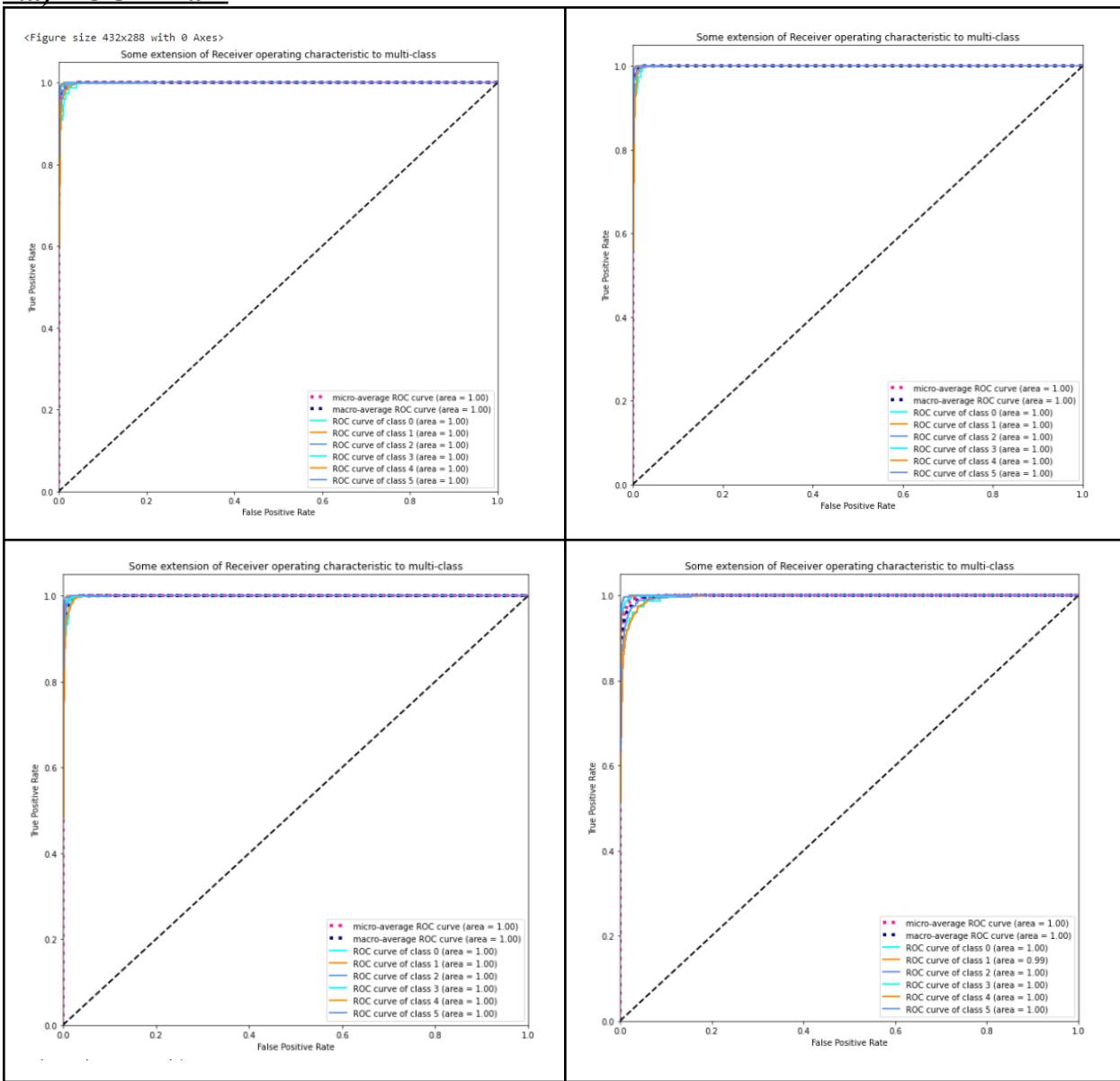
a.1) Confusion matrix - Training



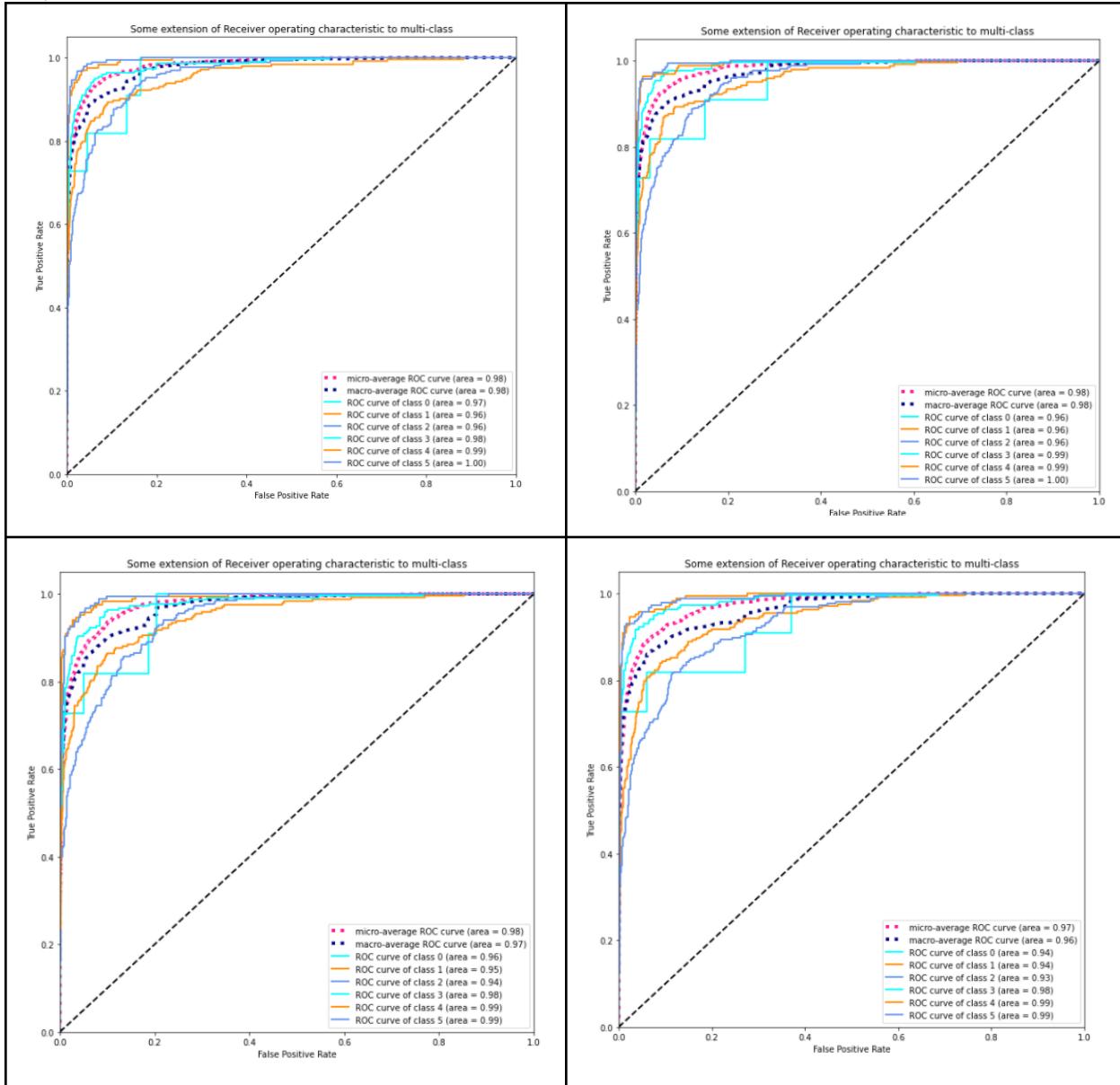
1.b) Confusion matrix - Test



2.a) ROC - Train



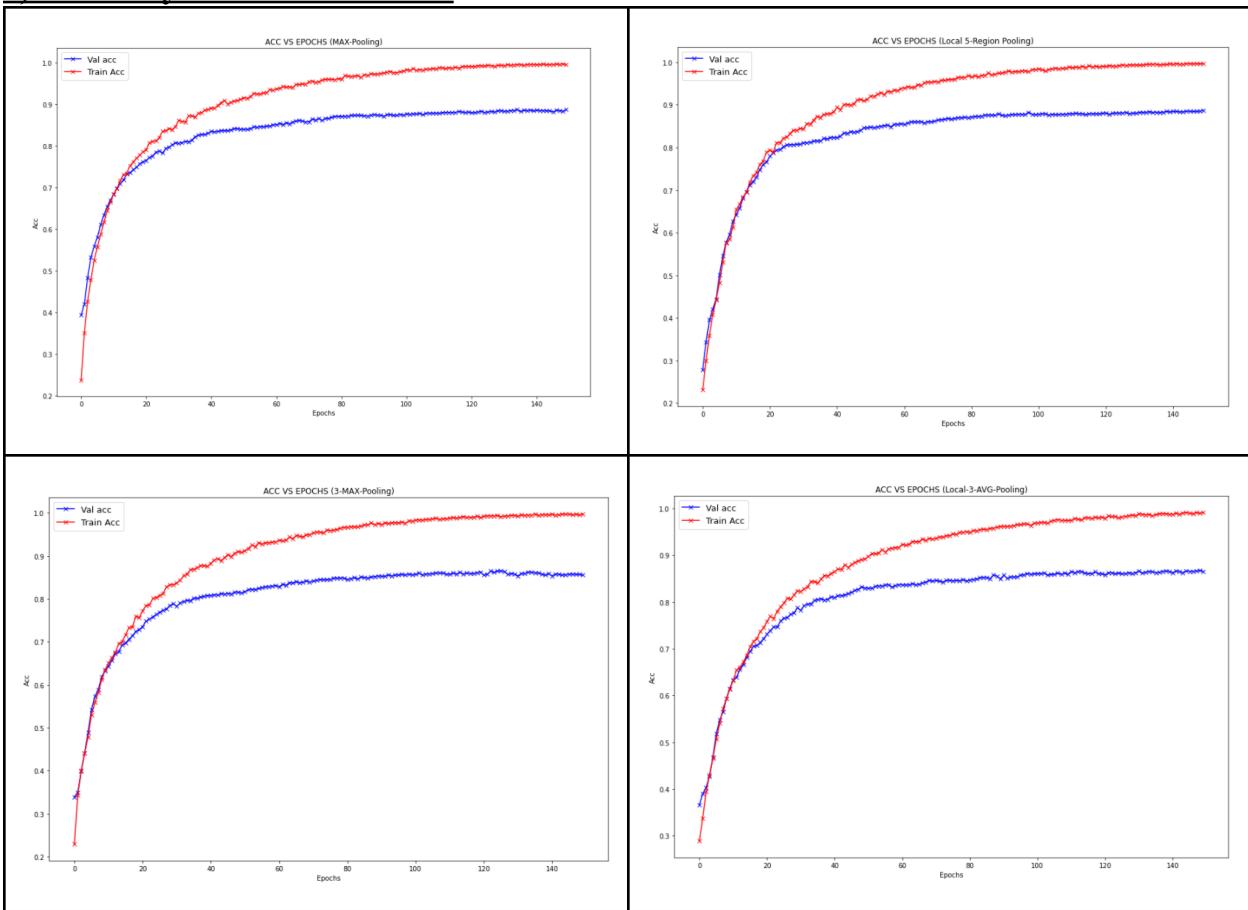
2.b) ROC - Test



3) F1 Measure - train vs test

precision recall f1-score support					precision recall f1-score support				
0	0.857	0.545	0.667	11	0	0.857	0.545	0.667	11
1	0.841	0.852	0.847	243	1	0.838	0.852	0.845	243
2	0.823	0.838	0.831	266	2	0.838	0.816	0.827	266
3	0.909	0.913	0.911	219	3	0.894	0.927	0.910	219
4	0.934	0.940	0.937	166	4	0.946	0.952	0.949	166
5	0.972	0.941	0.956	186	5	0.962	0.952	0.957	186
accuracy			0.886	1091	accuracy			0.887	1091
macro avg	0.889	0.838	0.858	1091	macro avg	0.889	0.841	0.859	1091
weighted avg	0.887	0.886	0.886	1091	weighted avg	0.887	0.887	0.887	1091
precision recall f1-score support					precision recall f1-score support				
0	0.985	0.880	0.930	75	0	1.000	0.893	0.944	75
1	0.986	0.996	0.991	919	1	0.989	1.000	0.995	919
2	0.998	0.993	0.995	984	2	0.999	0.995	0.997	984
3	0.996	0.999	0.998	1004	3	0.998	1.000	0.999	1004
4	0.997	0.996	0.996	669	4	0.999	0.999	0.999	669
5	0.996	0.999	0.997	710	5	1.000	1.000	1.000	710
accuracy			0.994	4361	accuracy			0.997	4361
macro avg	0.993	0.977	0.984	4361	macro avg	0.997	0.981	0.989	4361
weighted avg	0.994	0.994	0.994	4361	weighted avg	0.997	0.997	0.997	4361
precision recall f1-score support					precision recall f1-score support				
0	0.857	0.545	0.667	11	0	0.857	0.545	0.667	11
1	0.819	0.840	0.829	243	1	0.820	0.827	0.824	243
2	0.820	0.786	0.802	266	2	0.789	0.789	0.789	266
3	0.855	0.918	0.885	219	3	0.849	0.900	0.874	219
4	0.923	0.934	0.928	166	4	0.937	0.898	0.917	166
5	0.955	0.909	0.931	186	5	0.934	0.914	0.924	186
accuracy			0.865	1091	accuracy			0.855	1091
macro avg	0.871	0.822	0.840	1091	macro avg	0.865	0.812	0.832	1091
weighted avg	0.866	0.865	0.865	1091	weighted avg	0.856	0.855	0.855	1091
precision recall f1-score support					precision recall f1-score support				
0	0.983	0.773	0.866	75	0	1.000	0.880	0.936	75
1	0.974	0.995	0.984	919	1	0.991	0.999	0.995	919
2	0.995	0.988	0.991	984	2	0.999	0.999	0.999	984
3	0.997	0.997	0.997	1004	3	0.999	0.999	0.999	1004
4	0.994	0.999	0.996	669	4	0.997	1.000	0.999	669
5	0.999	1.000	0.999	710	5	1.000	1.000	1.000	710
accuracy			0.991	4361	accuracy			0.997	4361
macro avg	0.990	0.959	0.972	4361	macro avg	0.998	0.979	0.988	4361
weighted avg	0.991	0.991	0.991	4361	weighted avg	0.997	0.997	0.997	4361

4) Accuracy curve train vs test



5) Accuracy curve train vs test (Epochs)

1-max	5-local
Epoch: 10 Epoch Time: 0m 0s Train Loss: 1.154 Train Acc: 66.46% Val. Loss: 1.147 Test. Acc: 66.91%	Epoch: 10 Epoch Time: 0m 0s Train Loss: 1.263 Train Acc: 61.31% Val. Loss: 1.251 Test. Acc: 62.69%
Epoch: 20 Epoch Time: 0m 0s Train Loss: 0.748 Train Acc: 78.48% Val. Loss: 0.781 Test. Acc: 76.17%	Epoch: 20 Epoch Time: 0m 0s Train Loss: 0.785 Train Acc: 78.89% Val. Loss: 0.819 Test. Acc: 76.63%
Epoch: 30 Epoch Time: 0m 0s Train Loss: 0.530 Train Acc: 84.60% Val. Loss: 0.602 Test. Acc: 80.66%	Epoch: 30 Epoch Time: 0m 0s Train Loss: 0.546 Train Acc: 84.51% Val. Loss: 0.610 Test. Acc: 80.75%
Epoch: 40 Epoch Time: 0m 0s Train Loss: 0.402 Train Acc: 88.63% Val. Loss: 0.508 Test. Acc: 82.95%	Epoch: 40 Epoch Time: 0m 0s Train Loss: 0.402 Train Acc: 88.36% Val. Loss: 0.507 Test. Acc: 82.31%
Epoch: 50 Epoch Time: 0m 0s Train Loss: 0.317 Train Acc: 91.10% Val. Loss: 0.455 Test. Acc: 83.96%	Epoch: 50 Epoch Time: 0m 0s Train Loss: 0.311 Train Acc: 91.23% Val. Loss: 0.449 Test. Acc: 84.60%
Epoch: 60 Epoch Time: 0m 0s Train Loss: 0.254 Train Acc: 93.32% Val. Loss: 0.418 Test. Acc: 84.97%	Epoch: 60 Epoch Time: 0m 0s Train Loss: 0.238 Train Acc: 93.86% Val. Loss: 0.413 Test. Acc: 85.52%
Epoch: 70 Epoch Time: 0m 0s Train Loss: 0.204 Train Acc: 94.77% Val. Loss: 0.393 Test. Acc: 85.70%	Epoch: 70 Epoch Time: 0m 0s Train Loss: 0.189 Train Acc: 95.54% Val. Loss: 0.387 Test. Acc: 86.16%
Epoch: 80 Epoch Time: 0m 0s Train Loss: 0.167 Train Acc: 96.10% Val. Loss: 0.375 Test. Acc: 86.98%	Epoch: 80 Epoch Time: 0m 0s Train Loss: 0.147 Train Acc: 96.83% Val. Loss: 0.370 Test. Acc: 86.98%
Epoch: 90 Epoch Time: 0m 0s Train Loss: 0.133 Train Acc: 97.28% Val. Loss: 0.361 Test. Acc: 87.35%	Epoch: 90 Epoch Time: 0m 0s Train Loss: 0.117 Train Acc: 97.44% Val. Loss: 0.358 Test. Acc: 87.53%
Epoch: 100 Epoch Time: 0m 0s Train Loss: 0.103 Train Acc: 97.89% Val. Loss: 0.353 Test. Acc: 87.35%	Epoch: 100 Epoch Time: 0m 0s Train Loss: 0.094 Train Acc: 98.26% Val. Loss: 0.349 Test. Acc: 87.72%
Epoch: 110 Epoch Time: 0m 0s Train Loss: 0.084 Train Acc: 98.46% Val. Loss: 0.348 Test. Acc: 87.81%	Epoch: 110 Epoch Time: 0m 0s Train Loss: 0.072 Train Acc: 98.91% Val. Loss: 0.344 Test. Acc: 87.90%
Epoch: 120 Epoch Time: 0m 0s Train Loss: 0.065 Train Acc: 99.07% Val. Loss: 0.345 Test. Acc: 88.08%	Epoch: 120 Epoch Time: 0m 0s Train Loss: 0.060 Train Acc: 99.09% Val. Loss: 0.342 Test. Acc: 87.99%
Epoch: 130 Epoch Time: 0m 0s Train Loss: 0.055 Train Acc: 99.30% Val. Loss: 0.344 Test. Acc: 88.36%	Epoch: 130 Epoch Time: 0m 0s Train Loss: 0.045 Train Acc: 99.39% Val. Loss: 0.341 Test. Acc: 87.99%
Epoch: 140 Epoch Time: 0m 0s Train Loss: 0.045 Train Acc: 99.48% Val. Loss: 0.343 Test. Acc: 88.36%	Epoch: 140 Epoch Time: 0m 0s Train Loss: 0.036 Train Acc: 99.59% Val. Loss: 0.342 Test. Acc: 88.36%
Epoch: 150 Epoch Time: 0m 0s Train Loss: 0.038 Train Acc: 99.43% Val. Loss: 0.345 Test. Acc: 88.63%	Epoch: 150 Epoch Time: 0m 0s Train Loss: 0.029 Train Acc: 99.68% Val. Loss: 0.344 Test. Acc: 88.73%
3-max	3-local-average
Epoch: 10 Epoch Time: 0m 0s Train Loss: 1.196 Train Acc: 63.47% Val. Loss: 1.201 Test. Acc: 63.24%	Epoch: 10 Epoch Time: 0m 0s Train Loss: 1.220 Train Acc: 61.29% Val. Loss: 1.216 Test. Acc: 61.50%
Epoch: 20 Epoch Time: 0m 0s Train Loss: 0.778 Train Acc: 75.65% Val. Loss: 0.832 Test. Acc: 72.87%	Epoch: 20 Epoch Time: 0m 0s Train Loss: 0.814 Train Acc: 74.47% Val. Loss: 0.846 Test. Acc: 72.14%
Epoch: 30 Epoch Time: 0m 0s Train Loss: 0.552 Train Acc: 83.31% Val. Loss: 0.645 Test. Acc: 78.83%	Epoch: 30 Epoch Time: 0m 0s Train Loss: 0.573 Train Acc: 82.40% Val. Loss: 0.661 Test. Acc: 78.74%
Epoch: 40 Epoch Time: 0m 0s Train Loss: 0.420 Train Acc: 87.56% Val. Loss: 0.550 Test. Acc: 80.75%	Epoch: 40 Epoch Time: 0m 0s Train Loss: 0.449 Train Acc: 86.11% Val. Loss: 0.562 Test. Acc: 81.12%
Epoch: 50 Epoch Time: 0m 0s Train Loss: 0.319 Train Acc: 90.80% Val. Loss: 0.494 Test. Acc: 81.39%	Epoch: 50 Epoch Time: 0m 0s Train Loss: 0.351 Train Acc: 89.20% Val. Loss: 0.506 Test. Acc: 82.95%
Epoch: 60 Epoch Time: 0m 0s Train Loss: 0.251 Train Acc: 93.23% Val. Loss: 0.458 Test. Acc: 83.13%	Epoch: 60 Epoch Time: 0m 0s Train Loss: 0.284 Train Acc: 91.64% Val. Loss: 0.471 Test. Acc: 83.68%
Epoch: 70 Epoch Time: 0m 0s Train Loss: 0.198 Train Acc: 94.90% Val. Loss: 0.434 Test. Acc: 83.87%	Epoch: 70 Epoch Time: 0m 0s Train Loss: 0.232 Train Acc: 93.41% Val. Loss: 0.448 Test. Acc: 84.42%
Epoch: 80 Epoch Time: 0m 0s Train Loss: 0.159 Train Acc: 96.63% Val. Loss: 0.418 Test. Acc: 84.78%	Epoch: 80 Epoch Time: 0m 0s Train Loss: 0.188 Train Acc: 95.02% Val. Loss: 0.434 Test. Acc: 84.42%
Epoch: 90 Epoch Time: 0m 0s Train Loss: 0.122 Train Acc: 97.55% Val. Loss: 0.409 Test. Acc: 85.33%	Epoch: 90 Epoch Time: 0m 0s Train Loss: 0.154 Train Acc: 96.08% Val. Loss: 0.426 Test. Acc: 84.88%
Epoch: 100 Epoch Time: 0m 0s Train Loss: 0.096 Train Acc: 98.07% Val. Loss: 0.402 Test. Acc: 85.61%	Epoch: 100 Epoch Time: 0m 0s Train Loss: 0.129 Train Acc: 96.83% Val. Loss: 0.420 Test. Acc: 85.98%
Epoch: 110 Epoch Time: 0m 0s Train Loss: 0.074 Train Acc: 98.62% Val. Loss: 0.400 Test. Acc: 85.79%	Epoch: 110 Epoch Time: 0m 0s Train Loss: 0.103 Train Acc: 97.37% Val. Loss: 0.419 Test. Acc: 85.88%
Epoch: 120 Epoch Time: 0m 0s Train Loss: 0.062 Train Acc: 98.91% Val. Loss: 0.403 Test. Acc: 86.25%	Epoch: 120 Epoch Time: 0m 0s Train Loss: 0.087 Train Acc: 98.08% Val. Loss: 0.422 Test. Acc: 85.98%
Epoch: 130 Epoch Time: 0m 0s Train Loss: 0.046 Train Acc: 99.37% Val. Loss: 0.406 Test. Acc: 85.79%	Epoch: 130 Epoch Time: 0m 0s Train Loss: 0.069 Train Acc: 98.44% Val. Loss: 0.424 Test. Acc: 86.25%
Epoch: 140 Epoch Time: 0m 0s Train Loss: 0.039 Train Acc: 99.55% Val. Loss: 0.412 Test. Acc: 85.88%	Epoch: 140 Epoch Time: 0m 0s Train Loss: 0.059 Train Acc: 98.75% Val. Loss: 0.431 Test. Acc: 86.53%
Epoch: 150 Epoch Time: 0m 0s Train Loss: 0.029 Train Acc: 99.73% Val. Loss: 0.419 Test. Acc: 85.52%	Epoch: 150 Epoch Time: 0m 0s Train Loss: 0.049 Train Acc: 99.14% Val. Loss: 0.438 Test. Acc: 86.53%

Contribution of Each Member

1. Shivam Sharma

Implemented part(2) and report

2. Akanksha Shrimal

Implemented part(1) and report

3. Vaibhav Goswami

Implemented part(2) and report