# Please refer to : “https://github.com/wany0011/BS6207/Assingment\_3”

# Q 1: Please refer to source code titled: “Q1.ipynb”

Sum of Square Error is used to evaluation the numerical difference between my own implementations and the respective torch functions:

|  |  |  |
| --- | --- | --- |
| Torch function name | My Function name | Sum of Square Error |
| nn.MaxPool2d | myMaxPool2d | 0.0 |
| nn.AvgPool2d | myAvgPool2d | 0.0 |
| nn.Conv2d | myConv2d | 4.838458065133246e-15 |
| nn.Conv2d (dilation>1) | myConv2d\_2 | 8.664802881307877e-15 |
| nn.ConvTranspose2d | myConvTranspose2d | 1.2756529552626814e-15 |

|  |  |  |
| --- | --- | --- |
| Torch function name | My Function name | Sum of Square Error |
| torch.flatten | Myflatten | 0 |
| torch.sigmoid | Mysigmoid | 3.525193e-16 |
| torchvision.ops.roi\_pool | my\_roi\_pool | 0 |
| torch.nn.functional.batch\_norm | mybatch\_norm | 4.7291874e-15 |
| torch.nn.functional.cross\_entropy | my\_cross\_entropy | 0 |
| torch.nn.functional.mse\_loss | my\_mse\_loss | 0 |

# Q 2-A-1: Please refer to source code titled: “Q2\_A1.ipynb”

It involves multiple trials in efforts to improve test-accuracy:

1. Try different scheduler, that pushed it 75.03%

from torch.optim.lr\_scheduler import ExponentialLR

from torch.optim.lr\_scheduler import StepLR

from torch.optim.lr\_scheduler import ReduceLROnPlateau

1. Try “optuna” to include both learning rate and batch\_size into optimization , by creating the below objective function:

Text, application

Description automatically generated

However, none of above could push the accuracy above the boundary of 80%.

# Q 2-A-2: Please refer to source code titled: “Q2\_A2.ipynb”

1. Construct the model as follows:

Text, letter

Description automatically generated

1. Split data into 45000 ,5000 and 10000 for training, validation, and test, respectively, by random shuffling.
2. Set training parameters: batch\_size =500, with learning rate=0.0005 and num\_epoch =30

Chart

Description automatically generatedChart

Description automatically generated with low confidenceShow validation accuracy of each epoch, as well as loss of training and validation at each epoch,

1. Show the evaluation of test data:

{'val\_loss': 0.7770928740501404, 'val\_acc': 0.738237738609314}

1. Show the mean of absolute values of activation for all training data:

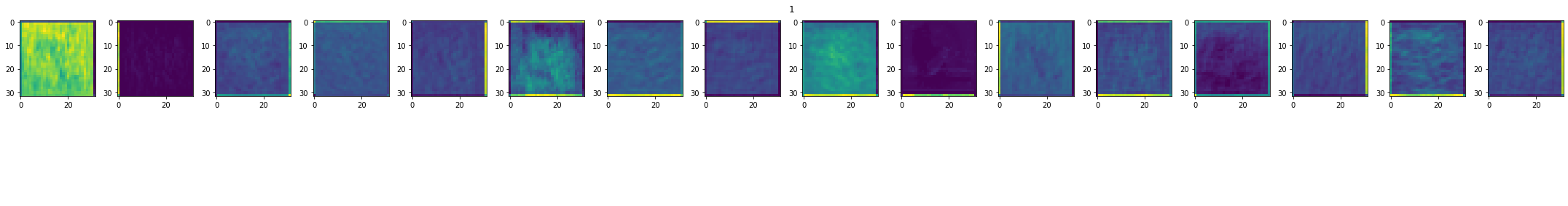
Table

Description automatically generated

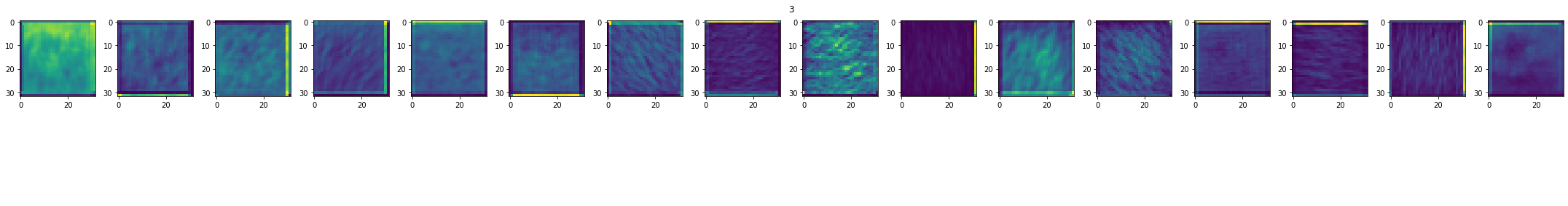
**# Observation: it has relatively high variance at 0.1393, and upon applying min-max scaler, its variance is reduced to 0.0693**

1. Show the average of absolute values of activation for each activation layer, i.e., layer\_activation = [ 1,3,6], by taking the **1st batch of test-dataset** as input.

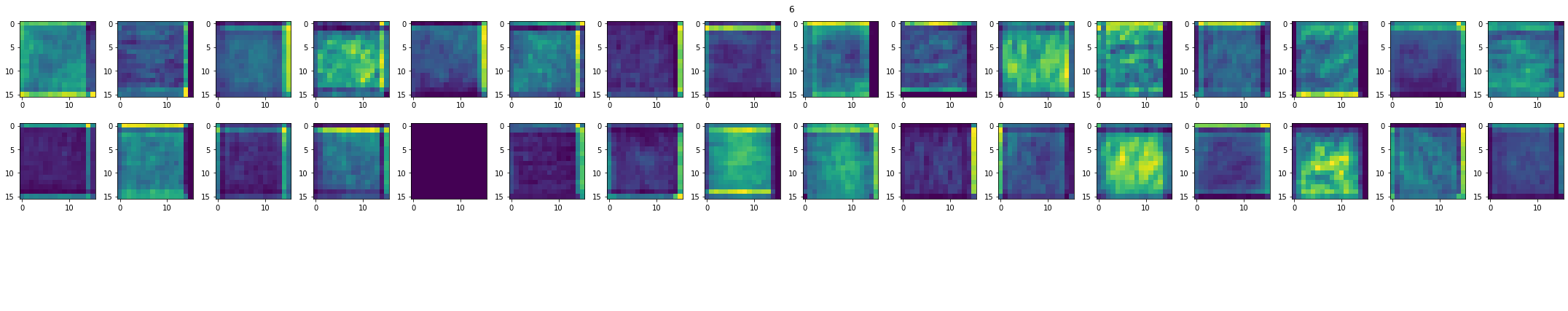
Layer 1:



Layer 3:



Layer 6 :



Layer 8:

A picture containing electronics, display

Description automatically generated

Layer 11:

A picture containing window, electronics, keyboard

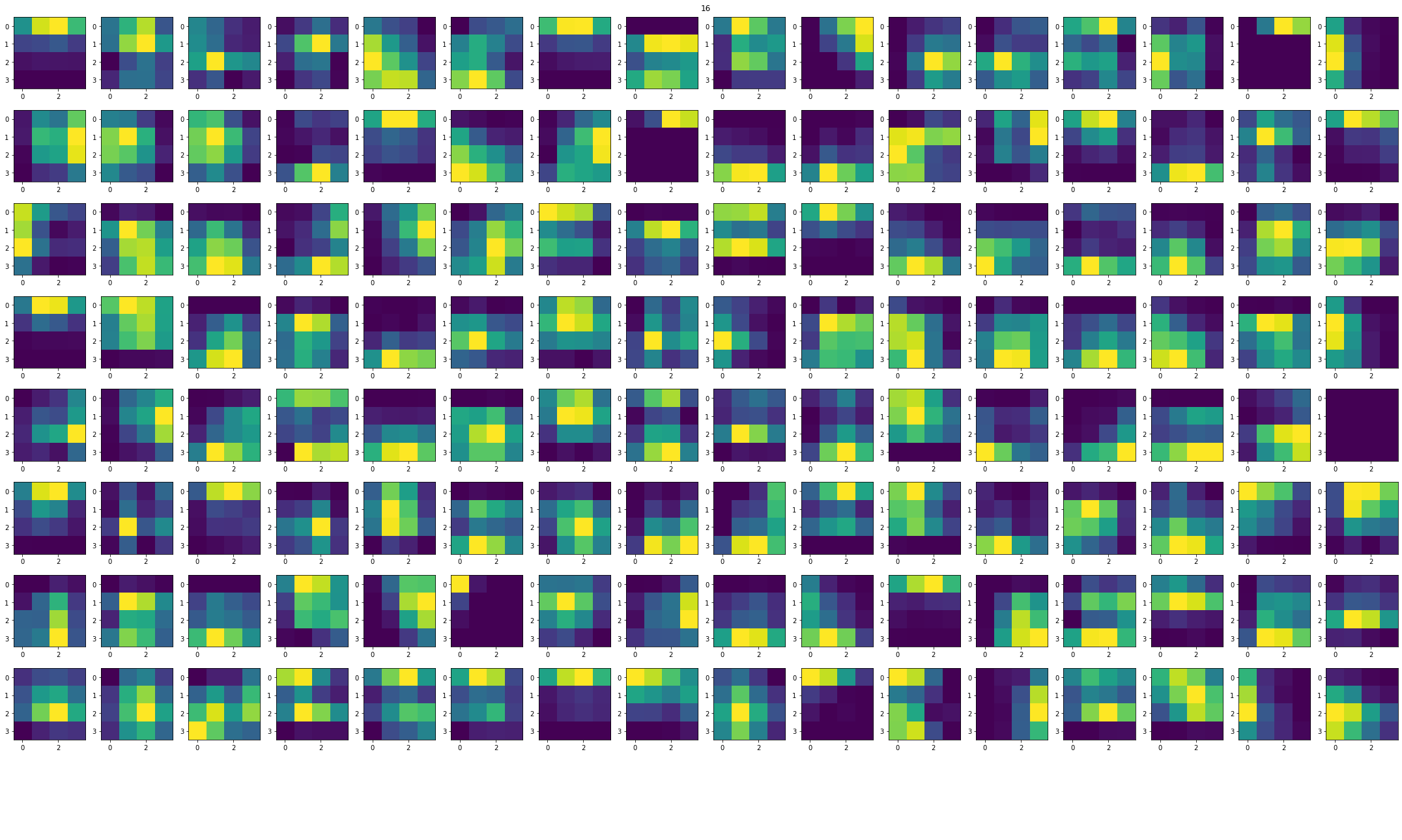
Description automatically generated

Layer 13:

A picture containing window

Description automatically generated

Layer 16:



Layer 18:

A picture containing table

Description automatically generated

# Q 2-b: Please refer to source code titled: “Q2\_B.ipynb”

Split data into 45000 ,5000 and 10000 for training, validation, and test, respectively, by random shuffling.

1. Construct the machine learning model as followings:

Text

Description automatically generated

1. Show the mean of absolute values of activation for all training data:

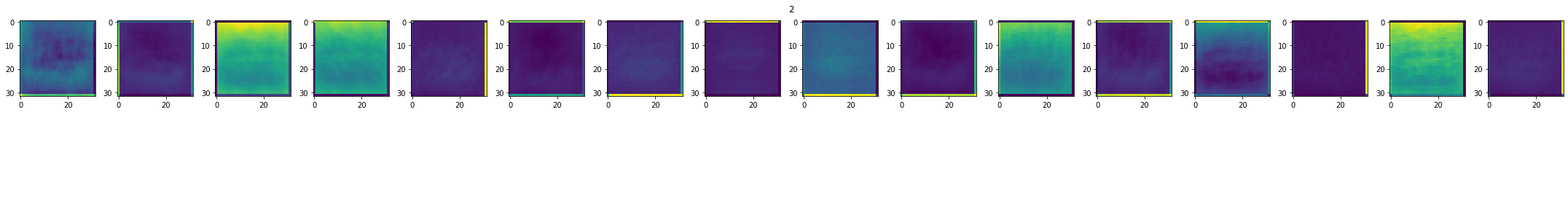
Table

Description automatically generated

**# Observation: it has very low variance at 0.0002, that explains the effect of batch-normalization placed before each activation layer.**

1. Show the average of absolute values of activation for each activation layer, i.e., layer\_activation = [ 2, 5, 9,12,16,19,23,26], by taking the **1st batch of test-dataset** as input.

Layer 2:



Layer 5:

A picture containing monitor

Description automatically generated

Layer 9:

A picture containing electronics, keyboard, dark, close

Description automatically generated

Layer 12:

A picture containing window, keyboard

Description automatically generated

Layer 16:

A picture containing window, electronics, keyboard

Description automatically generated

Layer 19:

A picture containing window, electronics, keyboard

Description automatically generated

Layer 23:

A picture containing table

Description automatically generated

Layer 26:

A picture containing keyboard

Description automatically generated

Conclusion:

1. the batch-normalization leads to very low variation across output of each activation, hence contour of loss function for all activation layers form a N-Dimensional sphere with equal radius (for example cube in 3D space). That accelerates the learning process, also makes the model less sensitive to variation of learning rates.

The figure could best explain the effect, comparing with and without batch-normalization, for the scenarios of 3-D space.

1. By visually comparing the activation of with & without batch normalization:
2. When batch normalization is not in-placed, some neurons are NOT activated,
3. When batch normalization is not in-placed, neurons are NOT uniformly activated,
4. 