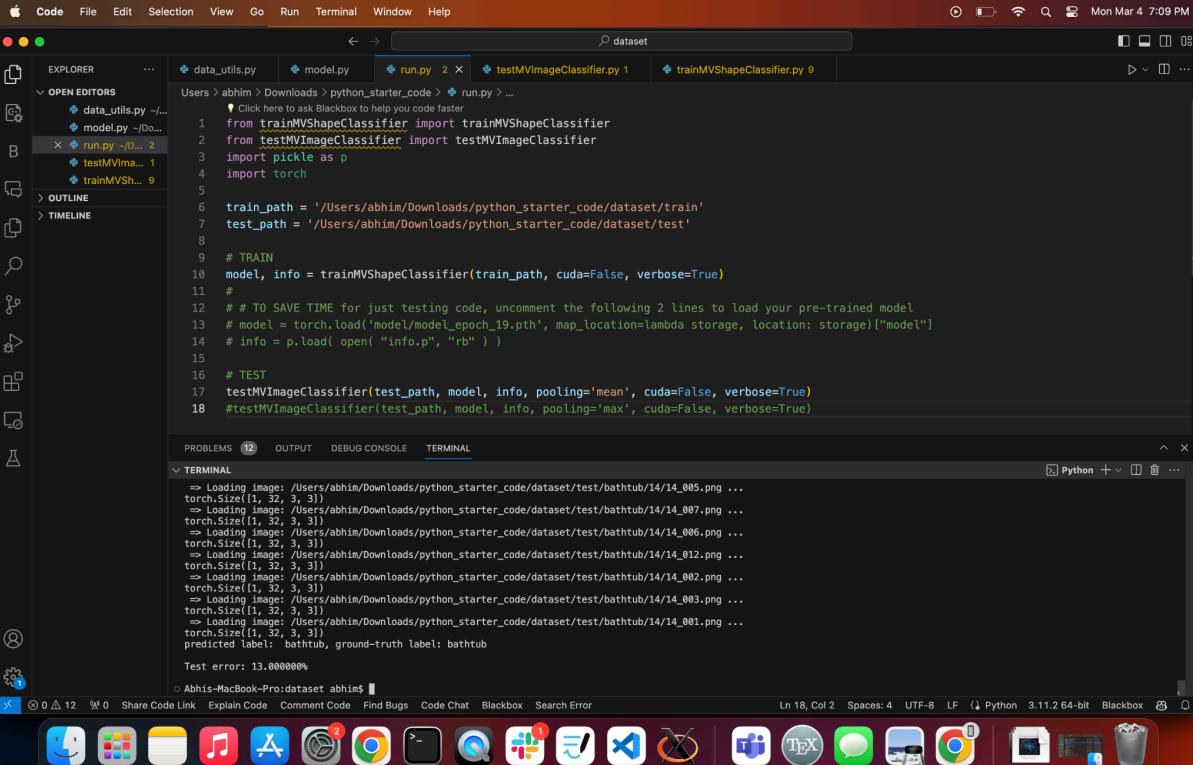


## Report

Test error for mean pooling: 13%

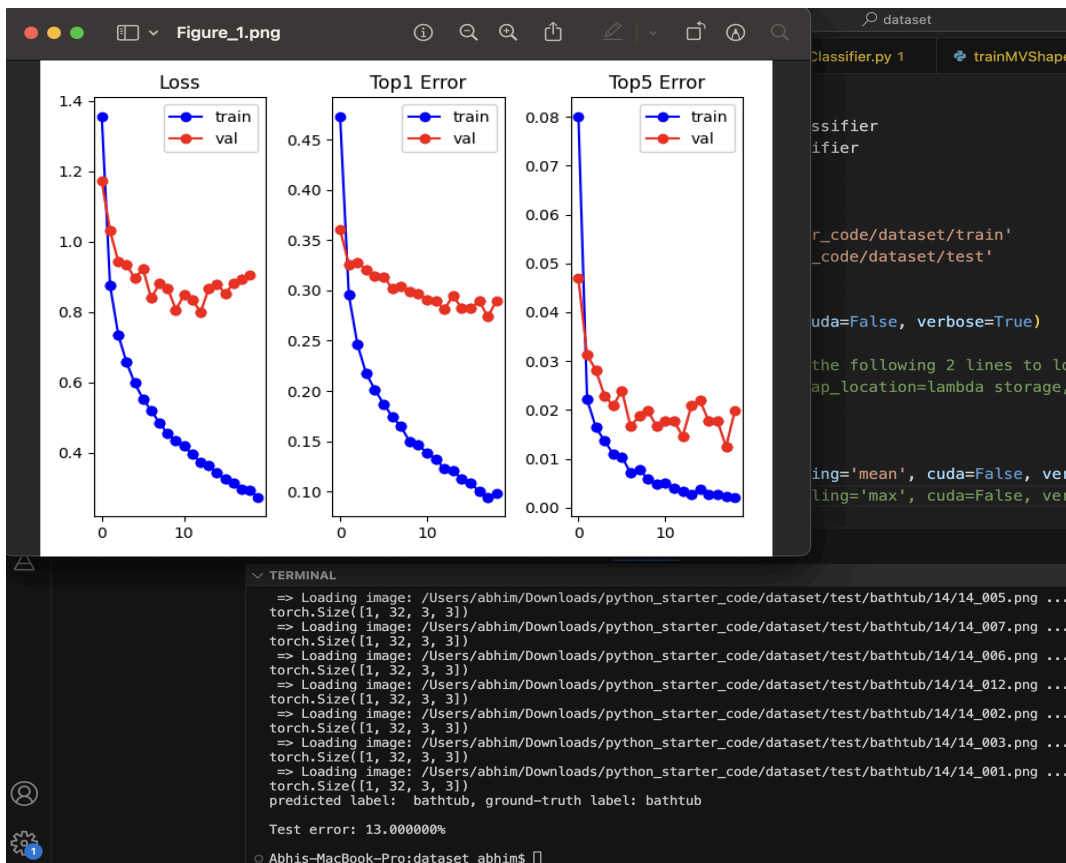


The screenshot shows a Visual Studio Code editor window with a Python script named `run.py` open. The script is located in the file explorer under `Users > abhim > Downloads > python_starter_code > dataset > run.py`. The script imports `trainMVShapeClassifier` and `testMVImageClassifier` from their respective modules, imports `pickle` as `p`, and imports `torch`. It defines `train_path` and `test_path` as `'/Users/abhim/Downloads/python_starter_code/dataset/train'` and `'/Users/abhim/Downloads/python_starter_code/dataset/test'` respectively. The script then trains a model using `trainMVShapeClassifier` with `cuda=False` and `verbose=True`. It includes commented-out lines for loading a pre-trained model and its info. Finally, it tests the model using `testMVImageClassifier` with `pooling='mean'` and `cuda=False`, and `pooling='max'` with `cuda=False`. The terminal output shows the execution of the script, loading images from the test dataset, and displaying the test error as 13.000000%.

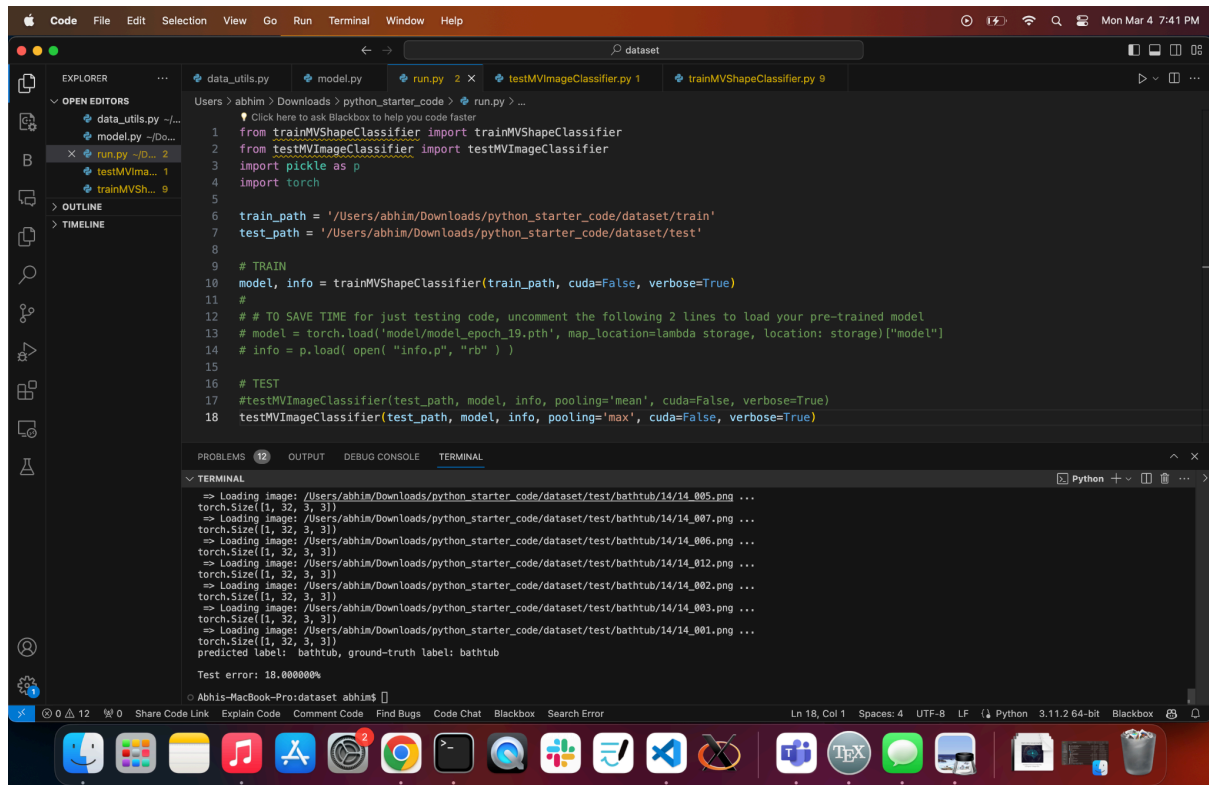
```
1 from trainMVShapeClassifier import trainMVShapeClassifier
2 from testMVImageClassifier import testMVImageClassifier
3 import pickle as p
4 import torch
5
6 train_path = '/Users/abhim/Downloads/python_starter_code/dataset/train'
7 test_path = '/Users/abhim/Downloads/python_starter_code/dataset/test'
8
9 # TRAIN
10 model, info = trainMVShapeClassifier(train_path, cuda=False, verbose=True)
11
12 # # TO SAVE TIME for just testing code, uncomment the following 2 lines to load your pre-trained model
13 # model = torch.load('model/model_epoch_19.pth', map_location=lambda storage, location: storage)["model"]
14 # info = p.load( open( "info.p", "rb" ) )
15
16 # TEST
17 testMVImageClassifier(test_path, model, info, pooling='mean', cuda=False, verbose=True)
18 #testMVImageClassifier(test_path, model, info, pooling='max', cuda=False, verbose=True)
```

Terminal Output:

```
=> Loading image: /Users/abhim/Downloads/python_starter_code/dataset/test/bathtub/14/14_005.png ...
torch.Size([1, 32, 3, 3])
=> Loading image: /Users/abhim/Downloads/python_starter_code/dataset/test/bathtub/14/14_007.png ...
torch.Size([1, 32, 3, 3])
=> Loading image: /Users/abhim/Downloads/python_starter_code/dataset/test/bathtub/14/14_006.png ...
torch.Size([1, 32, 3, 3])
=> Loading image: /Users/abhim/Downloads/python_starter_code/dataset/test/bathtub/14/14_012.png ...
torch.Size([1, 32, 3, 3])
=> Loading image: /Users/abhim/Downloads/python_starter_code/dataset/test/bathtub/14/14_002.png ...
torch.Size([1, 32, 3, 3])
=> Loading image: /Users/abhim/Downloads/python_starter_code/dataset/test/bathtub/14/14_003.png ...
torch.Size([1, 32, 3, 3])
=> Loading image: /Users/abhim/Downloads/python_starter_code/dataset/test/bathtub/14/14_001.png ...
torch.Size([1, 32, 3, 3])
predicted label: bathtub, ground-truth label: bathtub
Test error: 13.000000%
```

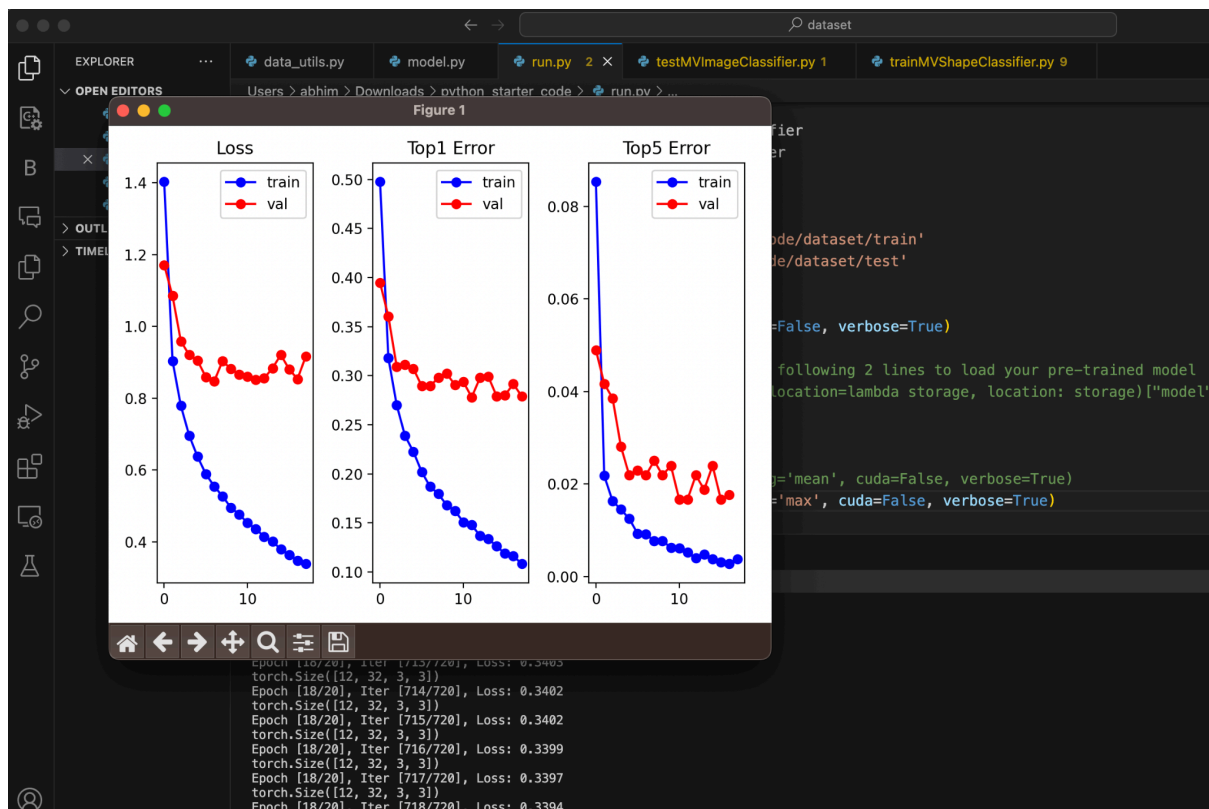


Test error for max pooling: 18%



```
1 from trainMVShapeClassifier import trainMVShapeClassifier
2 from testMVImageClassifier import testMVImageClassifier
3 import pickle as p
4 import torch
5
6 train_path = '/Users/abhim/Downloads/python_starter_code/dataset/train'
7 test_path = '/Users/abhim/Downloads/python_starter_code/dataset/test'
8
9 # TRAIN
10 model, info = trainMVShapeClassifier(train_path, cuda=False, verbose=True)
11 #
12 # # TO SAVE TIME for just testing code, uncomment the following 2 lines to load your pre-trained model
13 # model = torch.load('model/model_epoch_19.pth', map_location=lambda storage, location: storage)["model"]
14 # info = p.load( open( "info.p", "rb" ) )
15
16 # TEST
17 #testMVImageClassifier(test_path, model, info, pooling='mean', cuda=False, verbose=True)
18 testMVImageClassifier(test_path, model, info, pooling='max', cuda=False, verbose=True)
```

Test error: 18.000000%



```
1 import torch
2 import torch.nn as nn
3
4 class CNN(nn.Module):
5     def __init__(self, num_classes):
6         super(CNN, self).__init__()
7         # Layer 1: Convolutional layer with 8 filters applying 7x7 filters on the input image.
8         self.conv1 = nn.Conv2d(in_channels=1, out_channels=8, kernel_size=7, stride=1, padding=0, bias=False)
9         self.norm1 = nn.LayerNorm((8, 106, 106))
10        self.leaky_relu1 = nn.LeakyReLU(0.01)
11        self.maxpool1 = nn.MaxPool2d(kernel_size=2, stride=2)
12
13        # Layer 2: Depthwise+pointwise convolution layer --> depthwise separable convolution
14        self.depthwise_conv1 = nn.Conv2d(in_channels=8, out_channels=8, kernel_size=7, stride=2, padding=0, groups=8, bias=False)
15        self.norm2 = nn.LayerNorm((8, 24, 24))
16        self.leaky_relu2 = nn.LeakyReLU(0.01)
17        self.maxpool2 = nn.MaxPool2d(kernel_size=2, stride=2)
18        self.pointwise_conv1 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=1, stride=1, padding=0, bias=False)
19
20        # Layer 4: Depthwise+pointwise convolution layer --> depthwise separable convolution
21        self.depthwise_conv2 = nn.Conv2d(in_channels=16, out_channels=16, kernel_size=7, stride=1, padding=0, groups=16, bias=False)
22        self.norm3 = nn.LayerNorm((16, 6, 6))
23        self.leaky_relu3 = nn.LeakyReLU(0.01)
24        self.maxpool3 = nn.MaxPool2d(kernel_size=2, stride=2)
25        self.pointwise_conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=1, stride=1, padding=0, bias=False)
26
27        # Layer 6: Fully connected layer implemented as convolutional layer
28        self.fc = nn.Conv2d(in_channels=32, out_channels=num_classes, kernel_size=3, stride=1, padding=0, bias=True)
29
30        # Initialize weights
31        self.initialize_weights()
```

Bias set false for convolutional layers before ReLU.

Additionally we are using LayerNorm with `elementwise_affine` parameter defaulting to True (and bias True too) meaning bias variables are initially 0 and then learnt as training progresses so as to avoid having points at origin (0,0..) and to thus learn patterns better from the data without missing any points after the ReLU transformation.

Bias set false for pointwise convolutional layers.

Bias set true for final fully connected layer since even if the model learns that it isn't required it will just set it to zero at the end of training.