



Faculty of Engineering

XUEJIALU A0263762Y

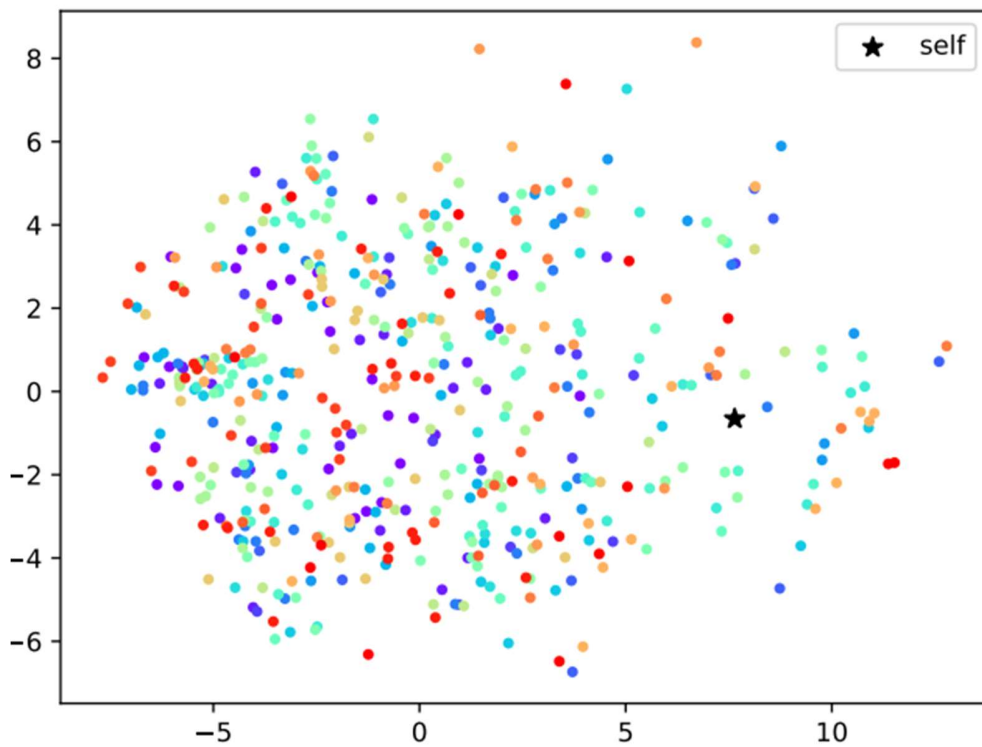
PCA

2D

Select subjects

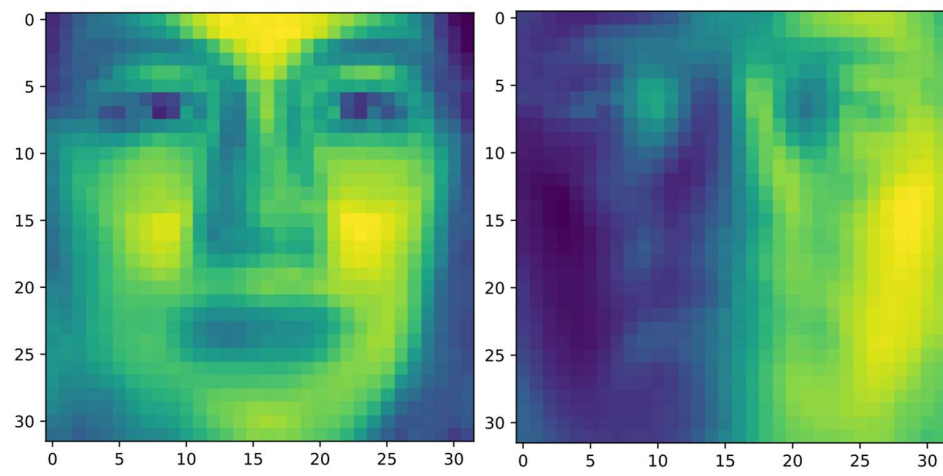
```
pca x
D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/pca.py
[36, 26, 55, 67, 45, 53, 63, 64, 1, 68, 24, 18, 34, 29, 40, 39, 61, 35, 56, 8, 15, 33, 10, 31, 54, 'selfff']
Process finished with exit code 0
```

Data Distribution Visualization



Different colors represent projected points from different subjects, and my own selfie photos are marked with black star.

eigenfaces

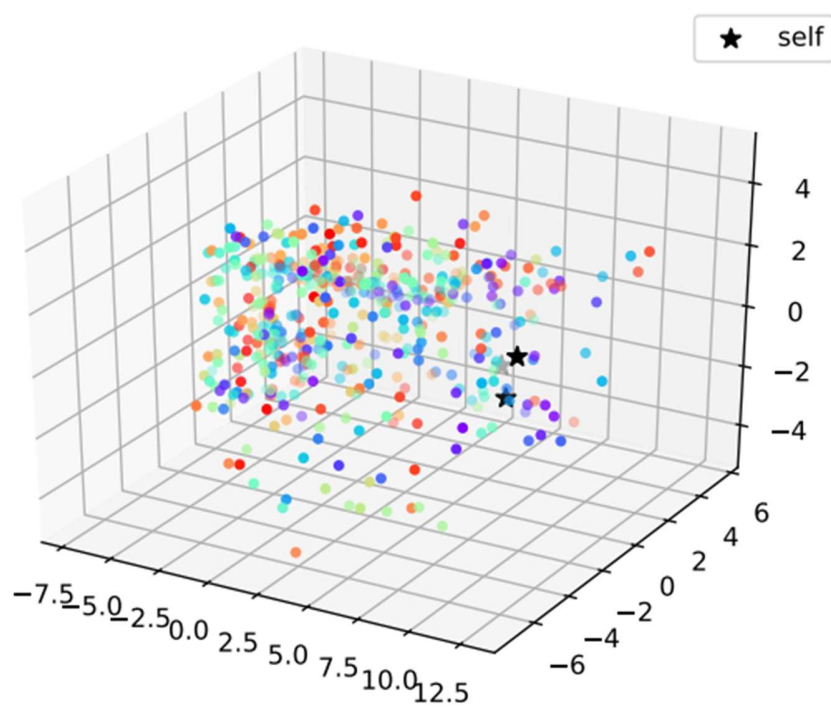


3D

Select subjects

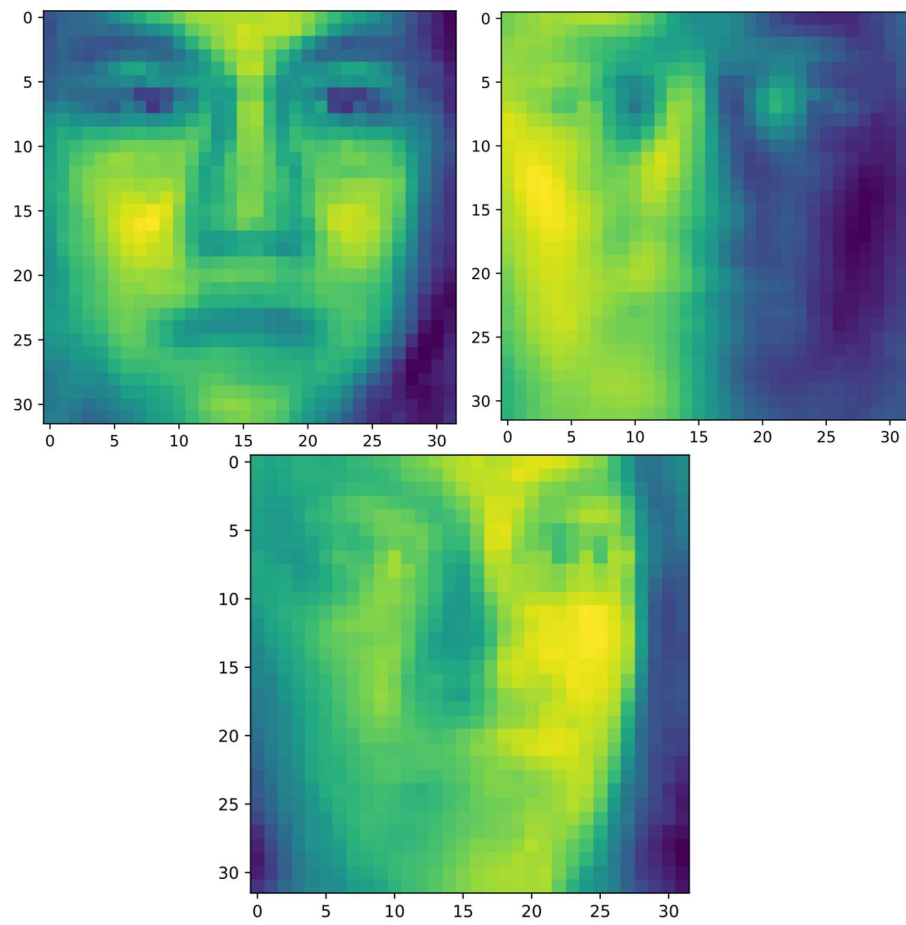
```
Run: pca x
D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/pca.py
[45, 4, 38, 18, 40, 27, 64, 33, 6, 22, 26, 8, 60, 63, 5, 50, 39, 36, 12, 62, 28, 35, 16, 2, 43, 'selfff']
Process finished with exit code 0
```

Data Distribution Visualization

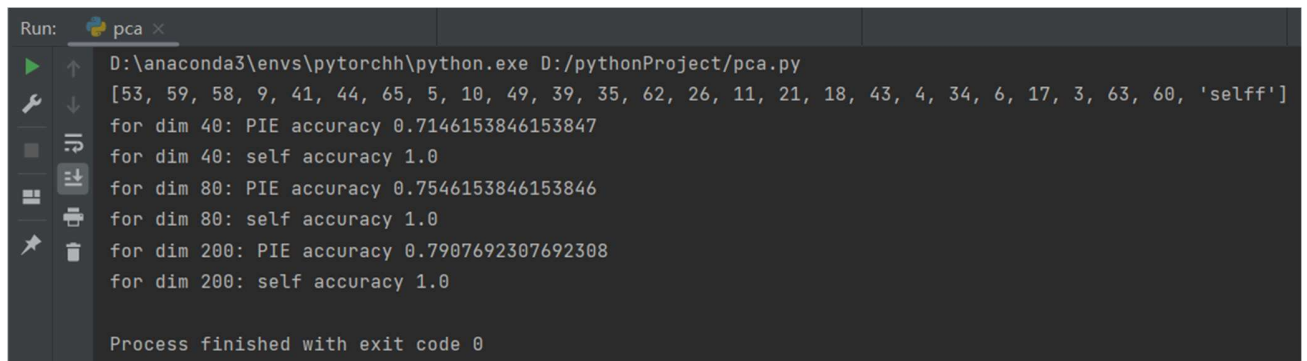


Different colors represent projected points from different subjects, and my own selfie photos are marked with black star.

eigenfaces



Classification Results

A screenshot of a Jupyter Notebook terminal window. The window title is 'Run: pca x'. The terminal output shows the execution of a Python script. It starts with the command 'D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/pca.py'. The first line of output is a list of numbers and a string: '[53, 59, 58, 9, 41, 44, 65, 5, 10, 49, 39, 35, 62, 26, 11, 21, 18, 43, 4, 34, 6, 17, 3, 63, 60, 'selfff']'. This is followed by three pairs of lines for different dimensions: 'for dim 40: PIE accuracy 0.7146153846153847' and 'for dim 40: self accuracy 1.0', 'for dim 80: PIE accuracy 0.7546153846153846' and 'for dim 80: self accuracy 1.0', and 'for dim 200: PIE accuracy 0.7907692307692308' and 'for dim 200: self accuracy 1.0'. The final line is 'Process finished with exit code 0'.

```
Run: pca x
D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/pca.py
[53, 59, 58, 9, 41, 44, 65, 5, 10, 49, 39, 35, 62, 26, 11, 21, 18, 43, 4, 34, 6, 17, 3, 63, 60, 'selfff']
for dim 40: PIE accuracy 0.7146153846153847
for dim 40: self accuracy 1.0
for dim 80: PIE accuracy 0.7546153846153846
for dim 80: self accuracy 1.0
for dim 200: PIE accuracy 0.7907692307692308
for dim 200: self accuracy 1.0

Process finished with exit code 0
```

the accuracy of my own selfie have the same accuracy of 100% on PCA 40 80 200, the accuracy of the PIE data is 71.5%, 75.5% and 79.1% for PCA 40, 80 and 200.

For the PIE data, we can see that PCA at higher dimensions, the accuracy increases gradually.

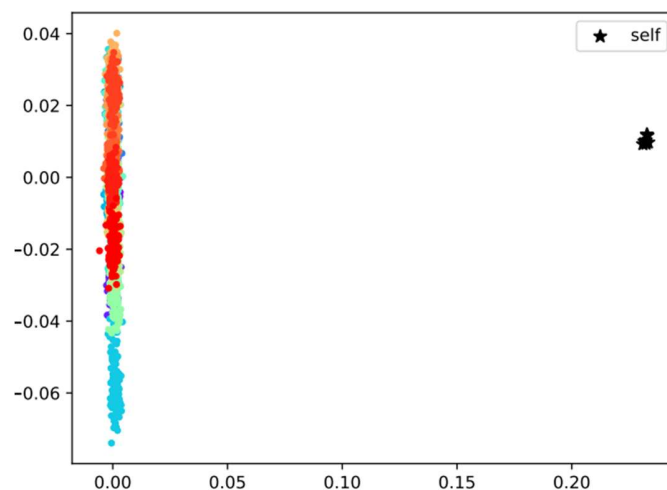
LDA

2D

Select subjects

```
Run: LDA x
D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/LDA.py
[43, 44, 54, 5, 62, 25, 23, 20, 57, 14, 65, 18, 41, 39, 3, 21, 26, 49, 36, 48, 27, 12, 19, 45, 35, 'selfff']
Process finished with exit code 0
```

Data Distribution Visualization



Different colors represent projected points from different subjects, and my own selfie photos are marked with black star.

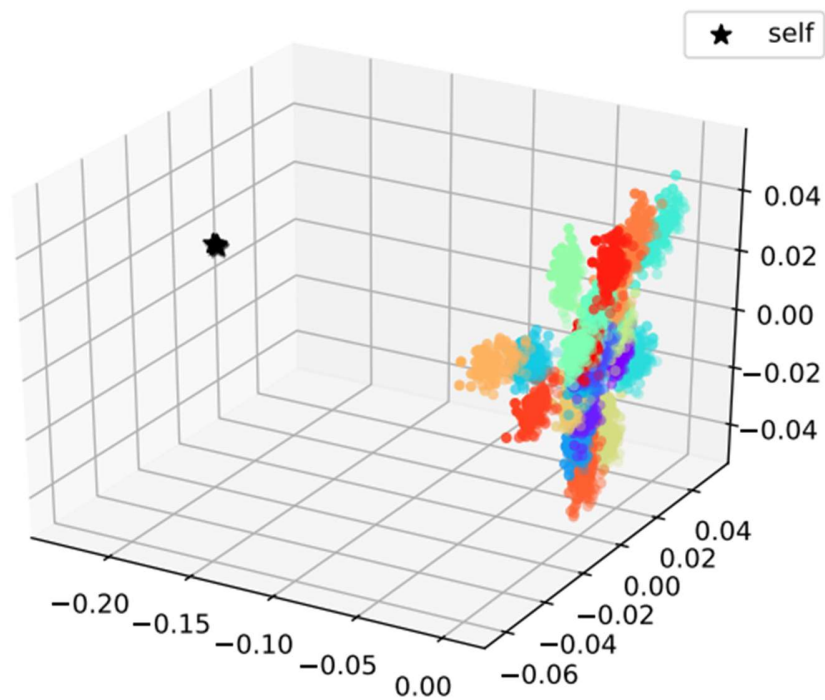
LDA realizes the distinction between classes by maximizing the "inter-class distance" and minimizing the "intra-class distance". Because the shooting environment of my selfie is quite different from that of the data set, it is easier to distinguish, and it is farther away from other data after projection.

3D

Select subjects

```
Run: LDA ×
D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/LDA.py
[26, 43, 64, 47, 22, 15, 45, 39, 65, 17, 3, 9, 40, 44, 31, 11, 10, 52, 20, 32, 53, 62, 21, 23, 68, 'self']
Process finished with exit code 0
```

Data Distribution Visualization



Classification Results

```
LDA x
D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/LDA.py
[9, 41, 4, 56, 54, 25, 3, 34, 39, 18, 15, 49, 65, 7, 46, 27, 44, 36, 47, 6, 64, 32, 29, 22, 26, 'selfff']
for dim 2: PIE accuracy 0.14692307692307693
for dim 2: self accuracy 1.0
for dim 3: PIE accuracy 0.32384615384615384
for dim 3: self accuracy 1.0
for dim 9: PIE accuracy 0.6730769230769231
for dim 9: self accuracy 1.0

Process finished with exit code 0
```

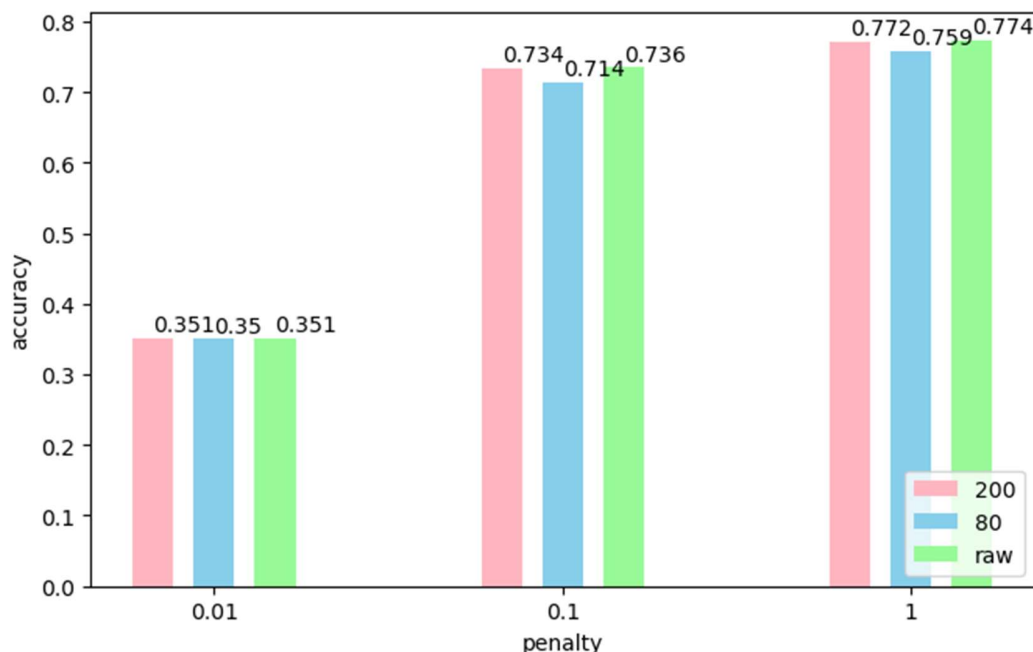
the accuracy of my own selfie have the same accuracy of 100% on LDA2, 3, 9, the accuracy of the PIE data is 14.7%, 32.4% and 67.3% for LDA 2, 3, 9.

For the PIE data, we can see that PCA at higher dimensions, the accuracy increases gradually.

SVM

Select subjects

```
D:\anaconda3\envs\pytorchh\python.exe D:/pythonProject/svm.py
[16, 61, 65, 8, 67, 22, 54, 38, 68, 2, 62, 5, 56, 24, 48, 17, 26, 39, 46, 50, 32, 64, 47, 1, 30, 'selfff']
Accuracy with penalty(0.01) for raw images: 0.35126825518831667
Accuracy with penalty(0.1) for raw images: 0.7355880092236741
Accuracy with penalty(1) for raw images: 0.7740199846272099
Accuracy with penalty(0.01) at 80 dimensions: 0.34973097617217525
Accuracy with penalty(0.1) at 80 dimensions: 0.7140661029976941
Accuracy with penalty(1) at 80 dimensions: 0.7586471944657955
Accuracy with penalty(0.01) at 200 dimensions: 0.35126825518831667
Accuracy with penalty(0.1) at 200 dimensions: 0.7340507302075326
Accuracy with penalty(1) at 200 dimensions: 0.7717140661029976
```



Influence of dimension and penalty parameter

The penalty parameter C represents the extent of SVM can avoid misclassification of each training data. For larger C values, the optimization will select a hyperplane with smaller boundaries if the training points can be better classified correctly. Conversely, a smaller value of C causes the optimizer to look for a larger bounded hyperplane, at the cost of potentially misclassifying more points.

The original image and the reduction of dimension to 80 and 200 have no significant effect on the result. It is proved that the performance of SVM may be independent of the dimension of feature space.

CNN

Select subjects

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 'selfff']
```

Network Architecture 1

two convolutional layers and one fully connected layer, with the architecture specified as follows: number of nodes: 20-50-500-26.

```
import torch.nn as nn
import torch.nn.functional as F

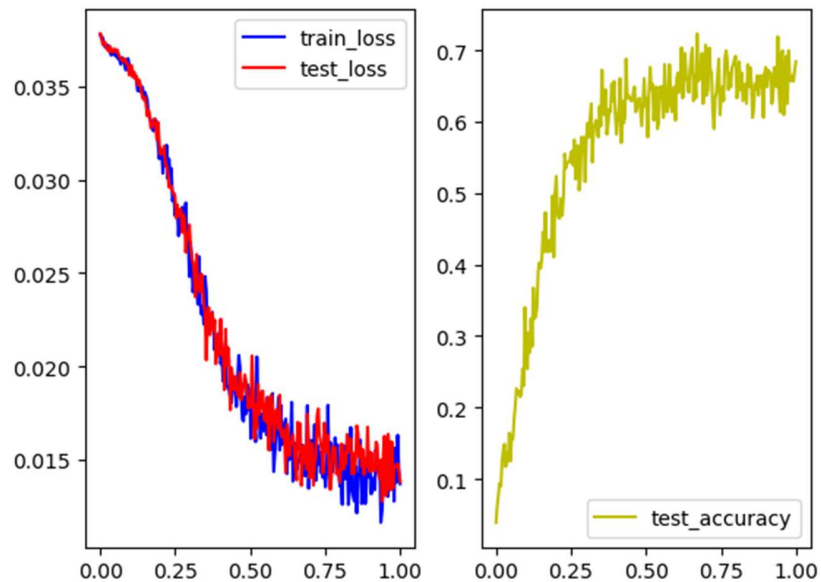
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.con1 = nn.Conv2d(1, 20, kernel_size=5)
        self.pool = nn.MaxPool2d(2, stride=2)
        self.con2 = nn.Conv2d(20, 50, kernel_size=5)
        self.fc = nn.Linear(1250, 500)
        self.fc1 = nn.Linear(500, 26)

    def forward(self, x):
        x = self.pool(self.con1(x))
        x = self.pool(self.con2(x))
        x = F.relu(self.fc(x.reshape(256, -1)))
        x = F.relu(self.fc1(x))

        return x
```

Result

```
epoch: 249 train loss: 0.013691765256226063
test accuracy: 0.68359375
test loss: 0.013810474425554276
finish training
```



Finally, the test loss drop to 0.013, and the test accuracy rise to 68.4%

Network Architecture 2

I've removed the second convolution layer on the base of the first architecture.

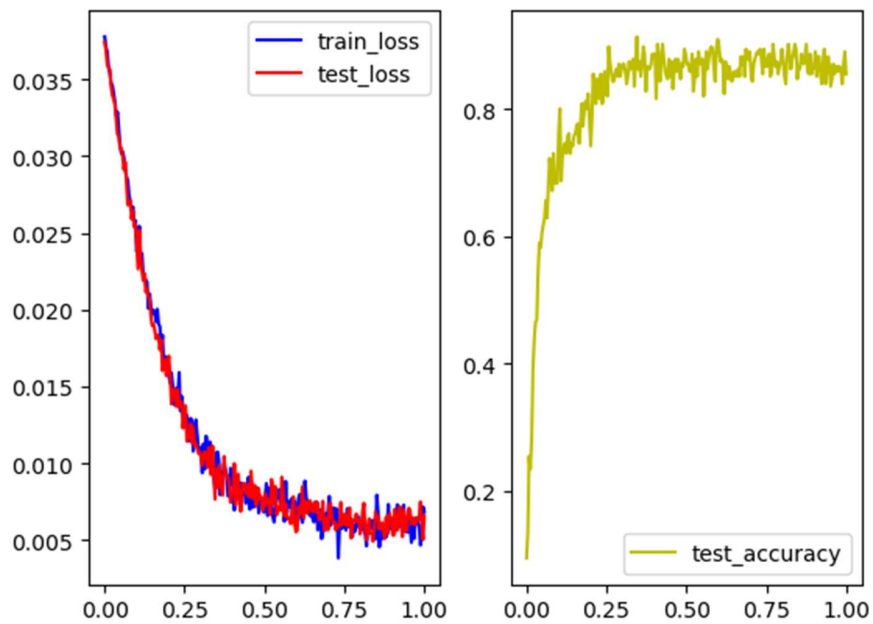
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.con1 = nn.Conv2d(1, 20, kernel_size=5)
        self.pool = nn.MaxPool2d(2, stride=2)
        self.fc = nn.Linear(20*14*14, 500)
        self.fc1 = nn.Linear(500, 26)

    def forward(self, x):
        x = self.pool(self.con1(x))
        x = F.relu(self.fc(x.reshape(256, -1)))
        x = F.relu(self.fc1(x))

        return x
```

result

```
epoch: 249 train loss: 0.006817028392106295  
test accuracy: 0.85546875  
test loss: 0.00657217251136899  
finish training  
  
Process finished with exit code 0
```



the test loss drop to 0.0065, and the test accuracy rise to 85.5%

Network Architecture 3

I've removed all the maxpool layers on the base of the first architecture.

```

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.con1 = nn.Conv2d(1, 20, kernel_size=5) # 256 20 28 28
        self.con2 = nn.Conv2d(20, 50, kernel_size=5) # 256 50 24 24
        self.fc = nn.Linear(50*24*24, 500)
        self.fc1 = nn.Linear(500, 26)

    def forward(self, x):
        x = self.con1(x)
        x = self.con2(x)
        x = F.relu(self.fc(x.reshape(256, -1)))
        x = F.relu(self.fc1(x))

        return x

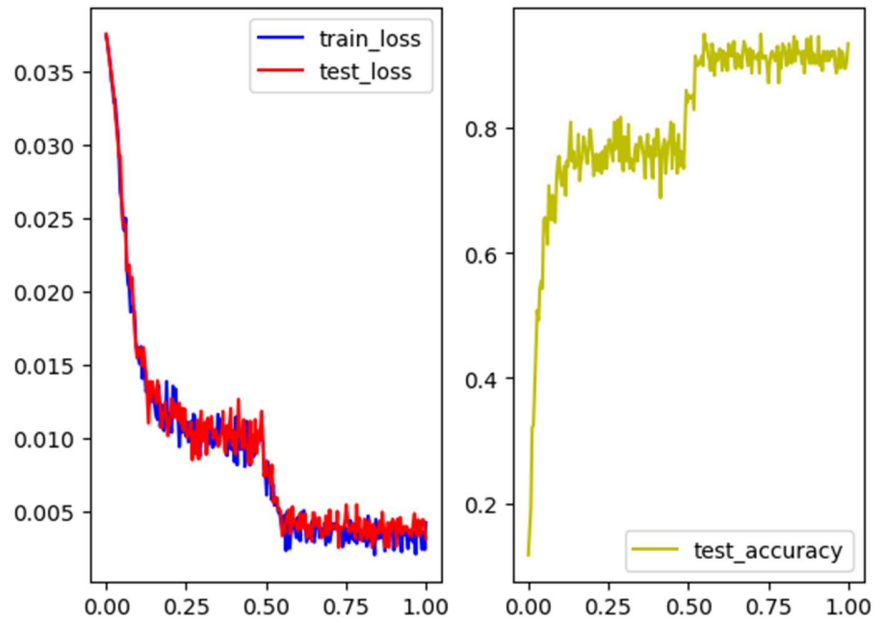
```

Result

```

epoch: 249 train loss: 0.004228631965816021
test accuracy: 0.93359375
test loss: 0.003141300054267049
finish training

```



The test loss drop to 0.0031, and the test accuracy rise to 93.3%. compare to the result of the first one, it can be seen that the existence of maxpool layer will reduce the performance of convolutional neural network.