Q1:

Symbolic: it works for simple functions and is pretty accurate, it is as accurate as f. However, it is not efficient for functions that involve complex simulation or calculation to be evaluated.

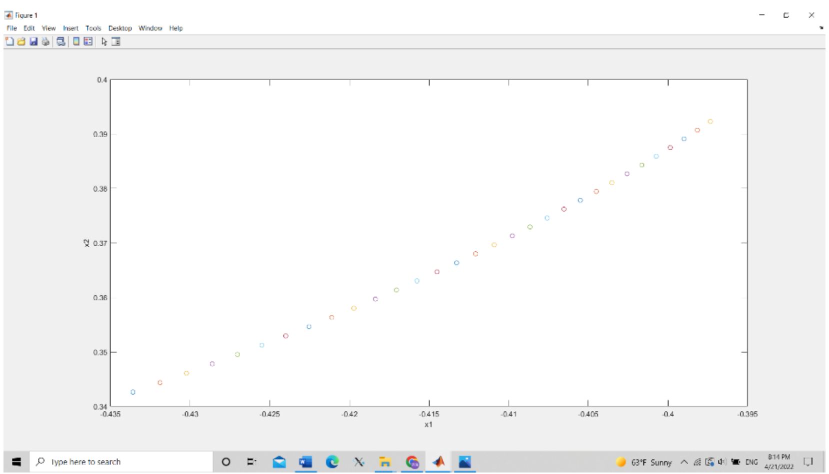
finite differences: only works if f can be evaluated with infinite precision/accuracy. Too high or too low h will yield bad derivative estimation. This is not the most efficient method.

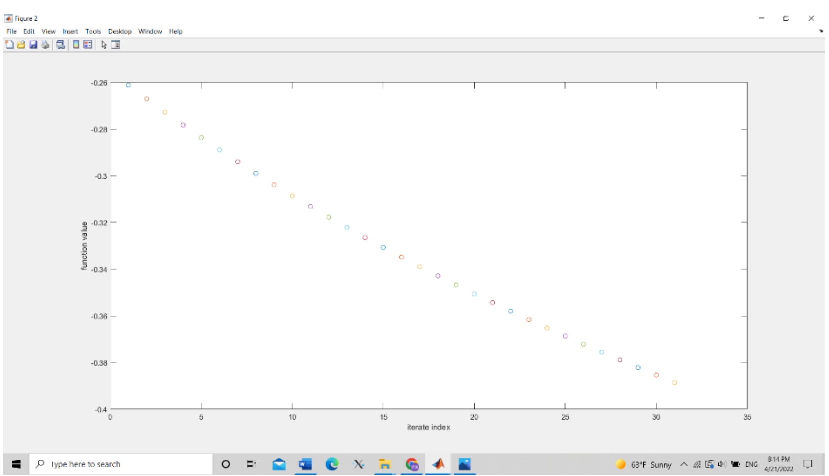
automatic differentiation: it is fast and efficient with modern implementation (fastest in these three methods), and it is very accurate.

Q2:

When initial is x1 = -0.4336 and x2 = 0.3426

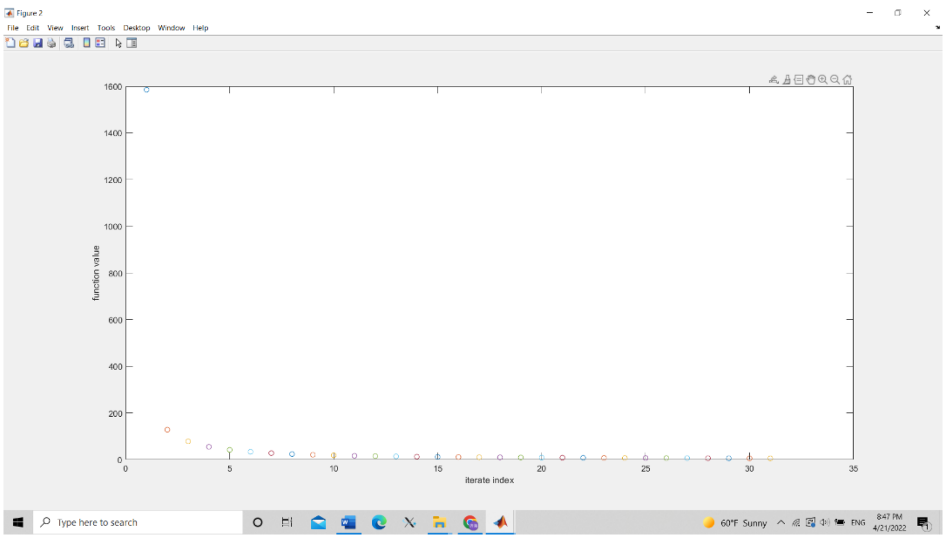
```
xResult_GradientDescent =

   -0.3973
    0.3923


xResult_fminunc =

   1.0e+13 *

   -0.0000
    3.6287


xResult_fminsearch =

   1.0e+30 *

   -0.0000
    1.3959
```

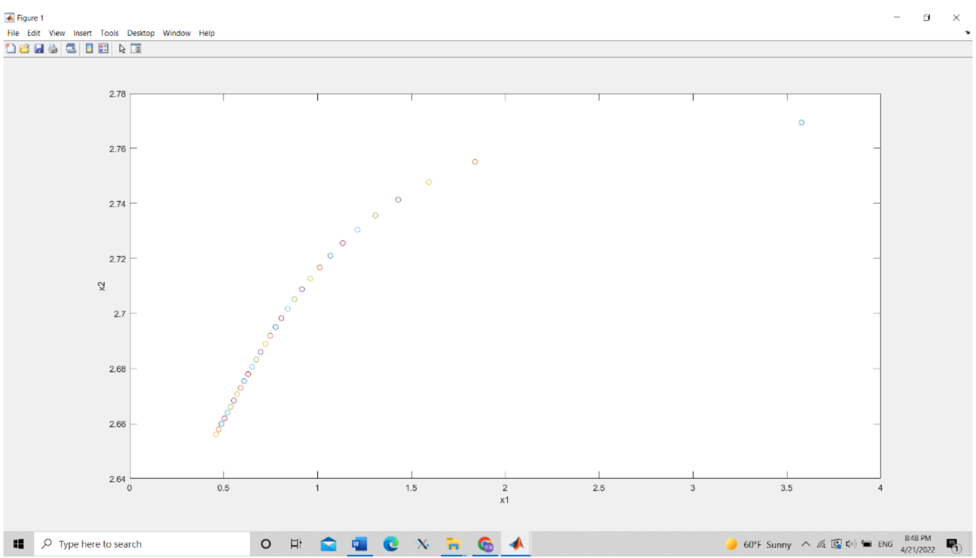




When initial is x1 = 3.5784 and x2 = 2.7694

```
xResult_GradientDescent =

    0.4600
    2.6560


xResult_fminunc =

   1.0e+12 *

    0.0000
   -4.4665


xResult_fminsearch =

   1.0e+28 *

   -0.0000
    3.0796

fx >>
```





Q4:

+ Code + Text

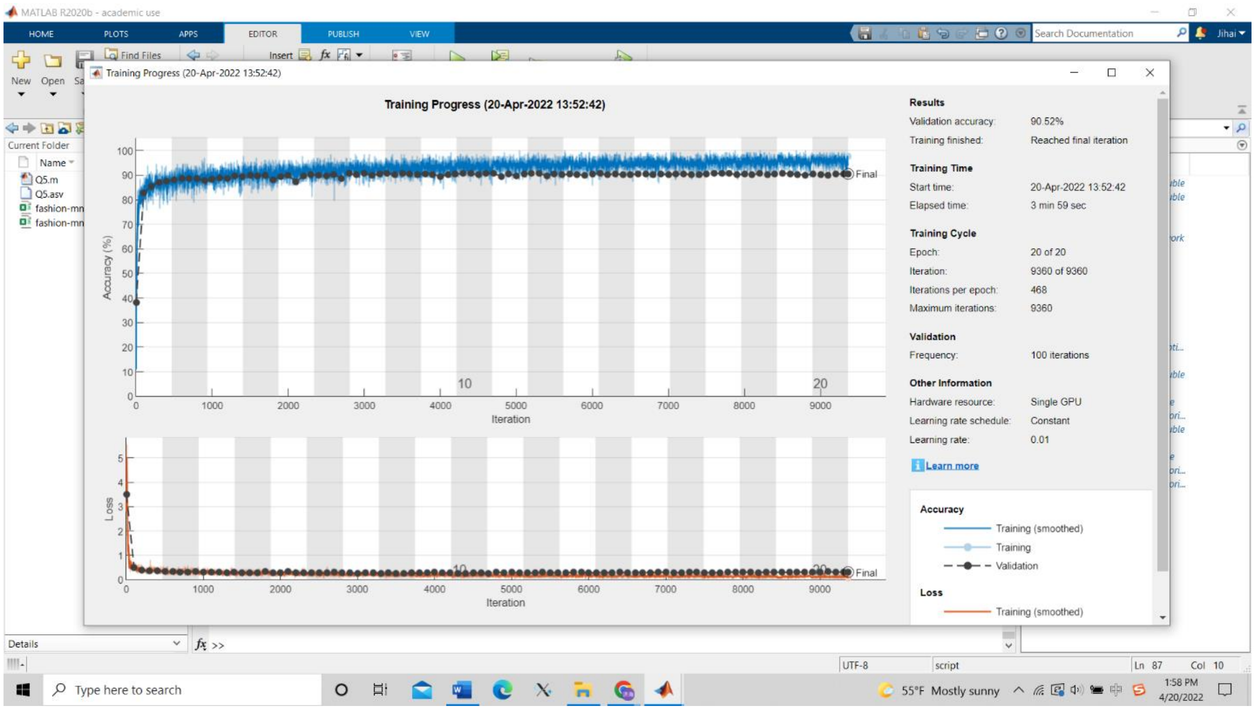
```
import numpy as nd
import sympy as sp
x1, x2 = sp.symbols('x1 x2')
f = (2-x1)**2+10*(x2-x1**2)**2
print(f)
F1 = sp.diff(f,x1)
F2 = sp.diff(f,x2)
F = sp.lambdify([x1,x2], [F1,F2])

F(3,4)
```

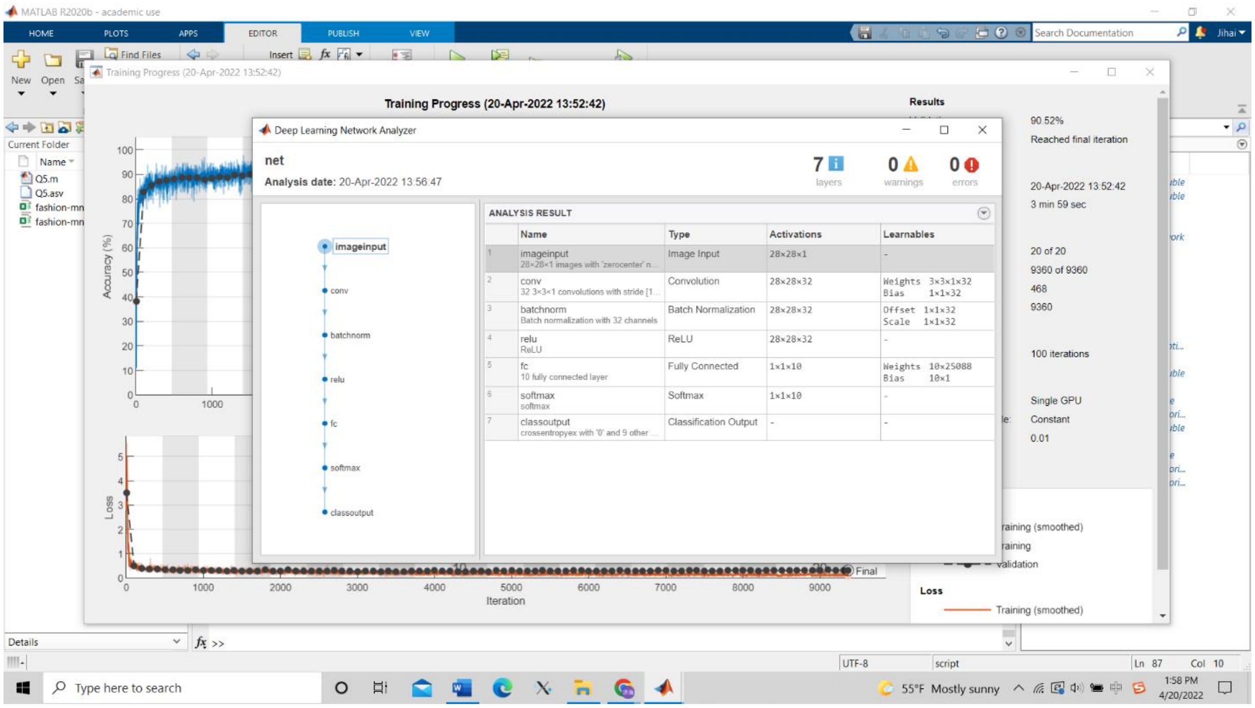
```
(2 - x1)**2 + 10*(-x1**2 + x2)**2
[602, -100]
```

Q5:

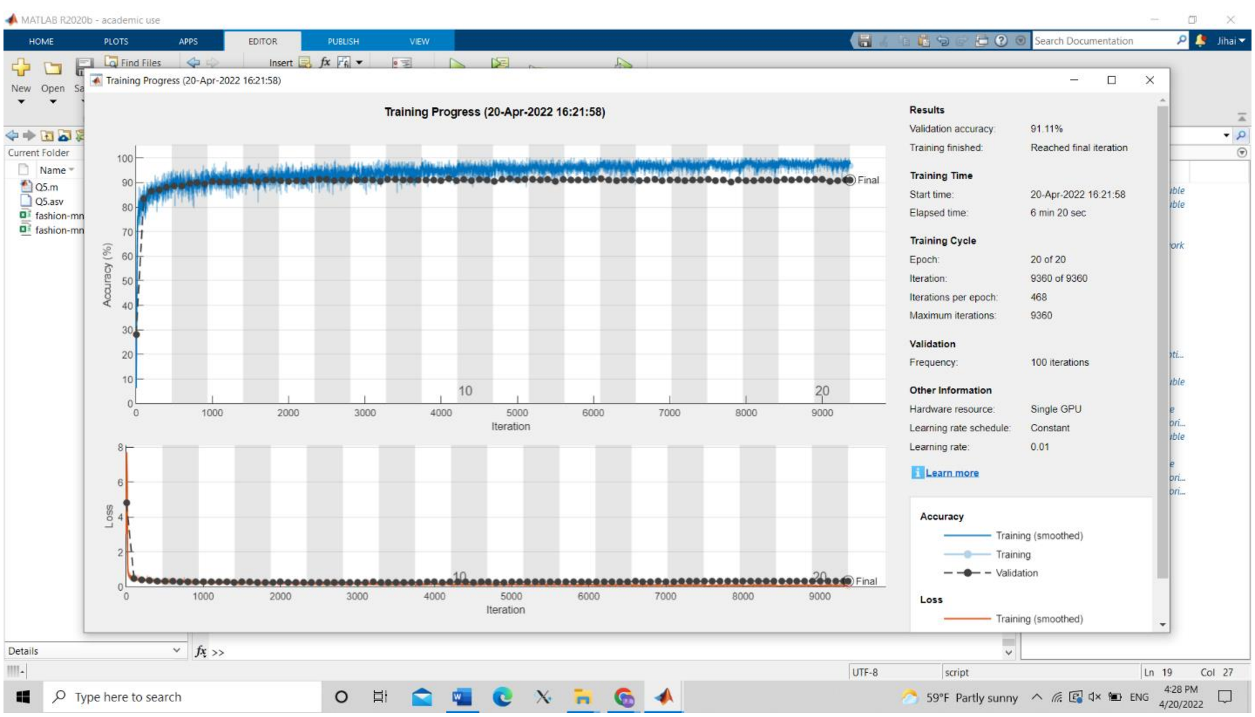
In this problem, I used a large number of epochs which is 20. First, run with default L2Regularization which is 1*10^-4.

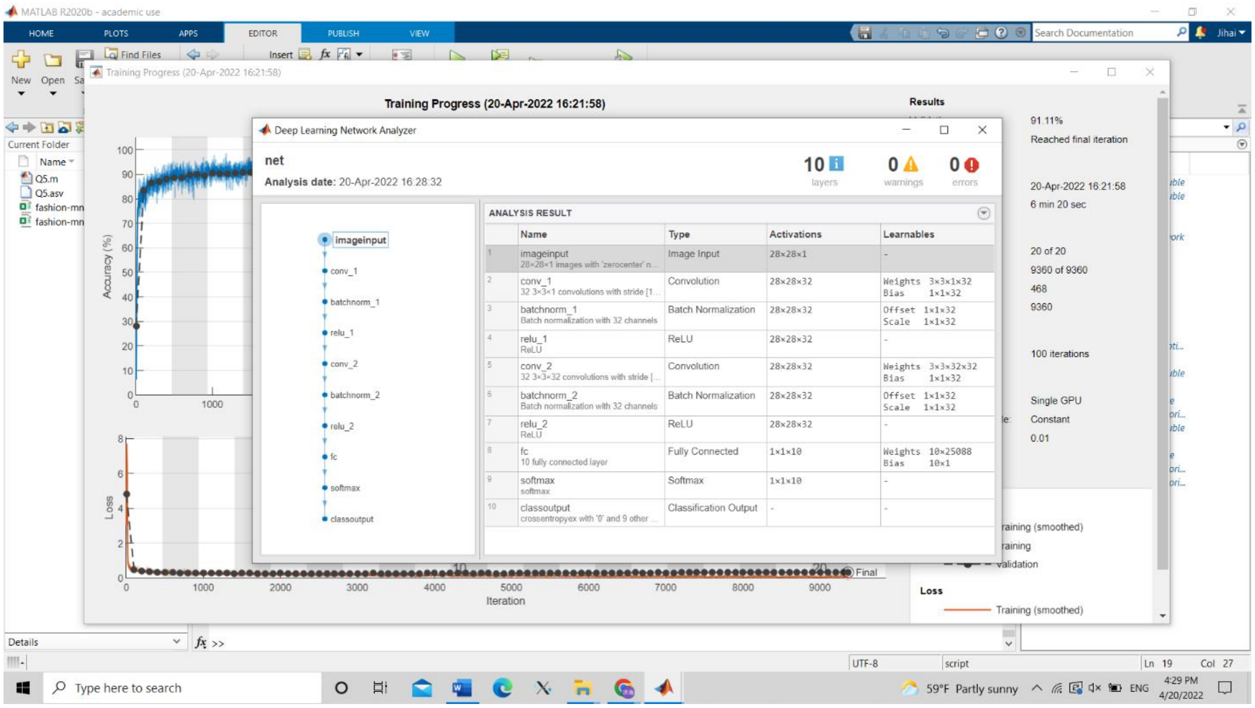
a) the program runs pretty slow which costs 3 mins and 59 seconds. The accuracy is 90.52%.



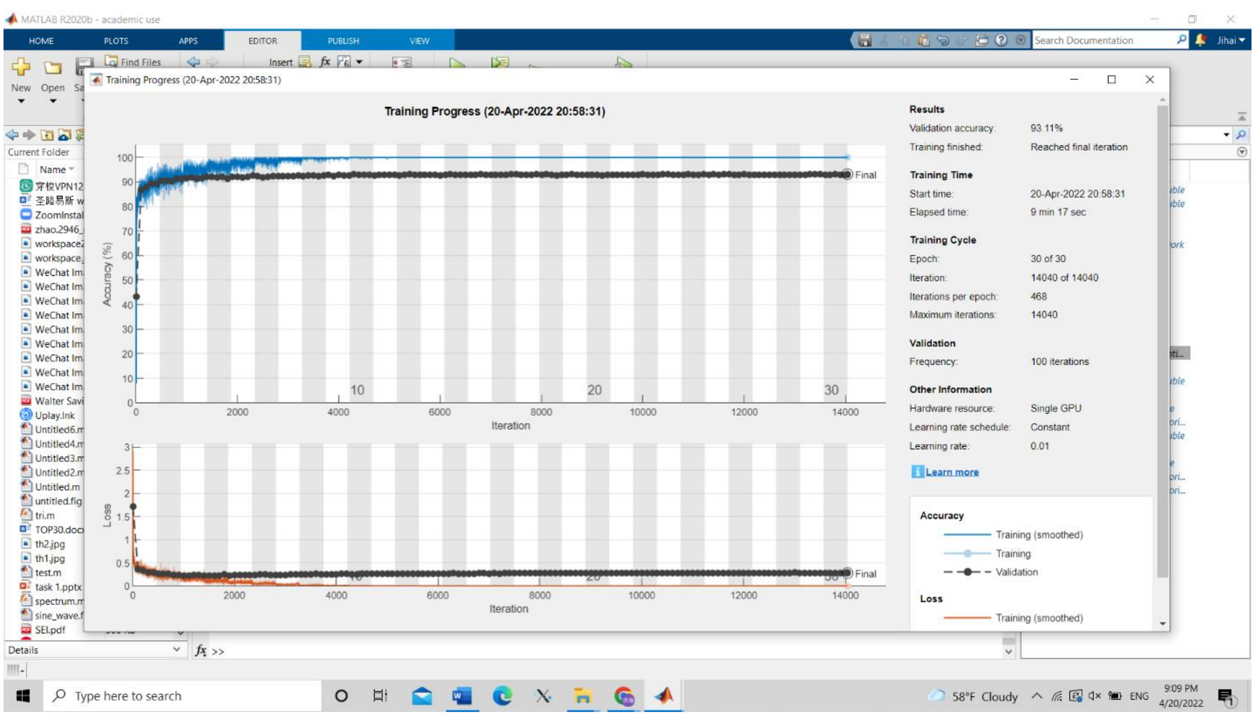


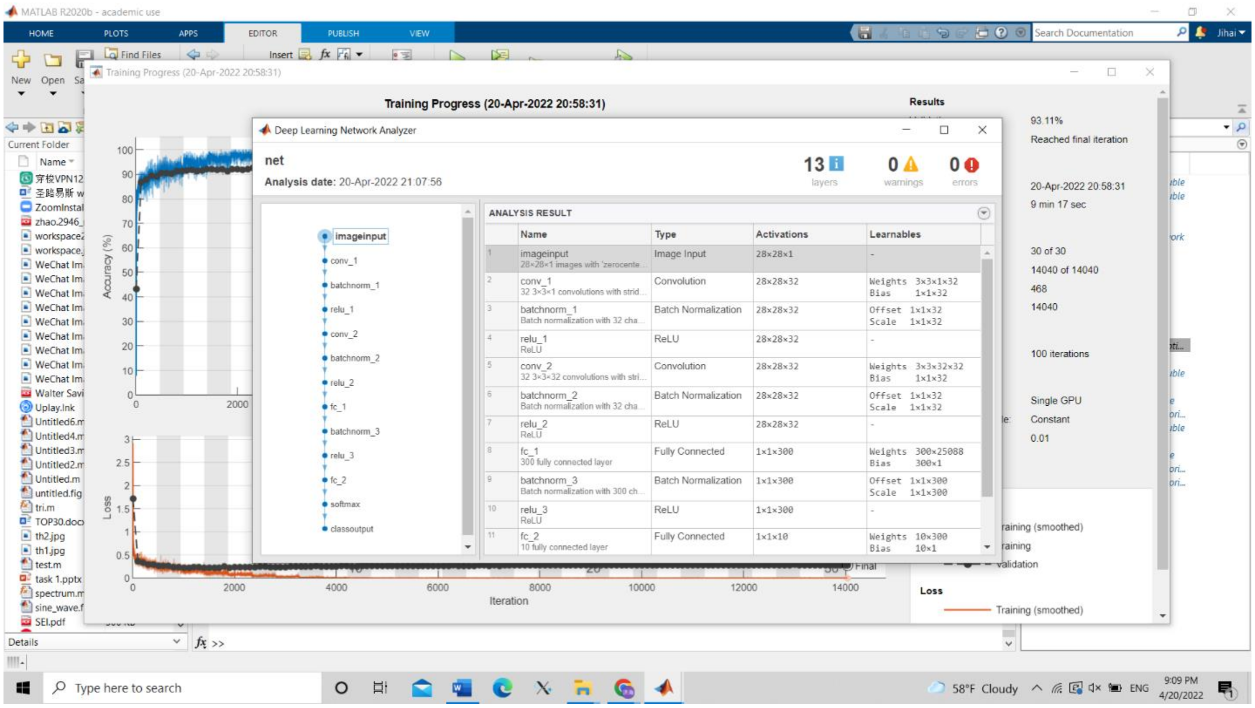
b) After adding another convolution2DLayer, bathchnormalization, and reLULayer, the program runs even slower which costs 6 mins and 20 seconds. The accuracy increases a litter bit which becomes 91.11%.





c) I add the trio of layers: fully connected + reLu + batchnorm and increase epochs to 30. The running time keeps increasing to 9 min 17 sec. The accuracy increases a lot and becomes 93.11%.

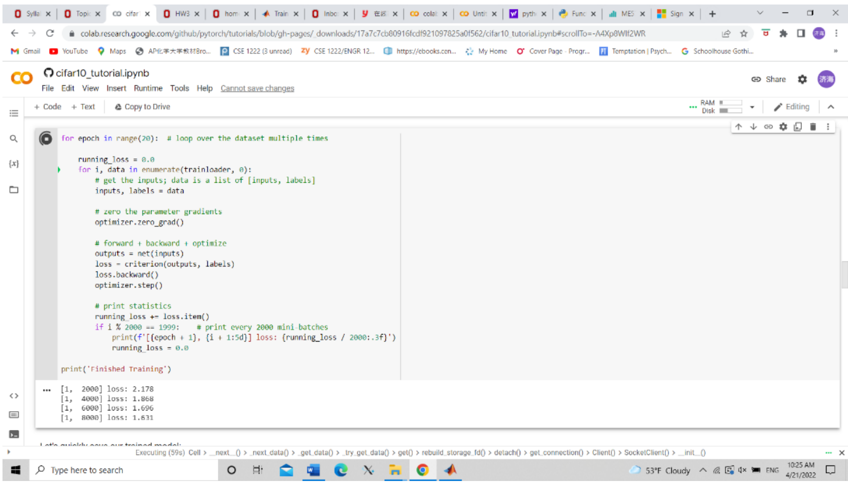


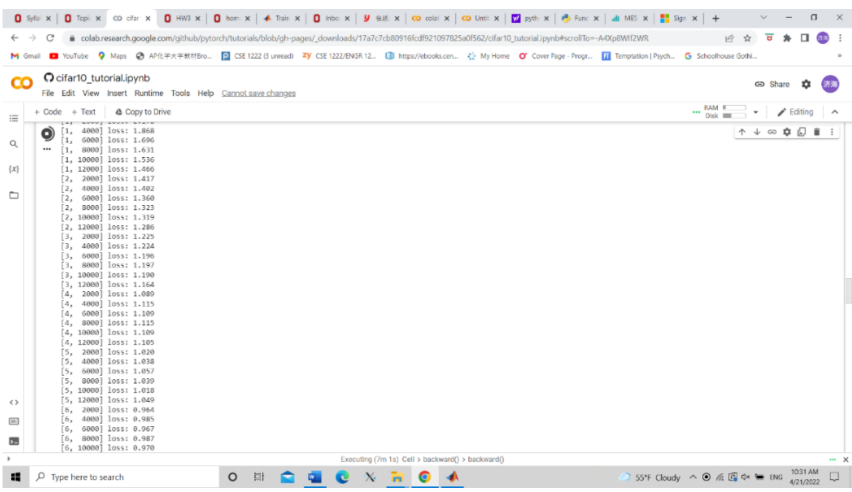


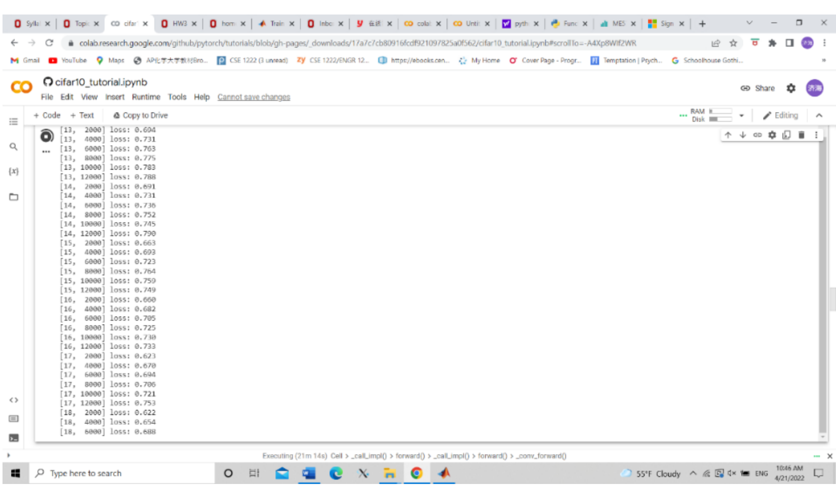
MATLAB R2020b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Training Progress (20-Apr-2022 20:58:31)
Results
93.11%
Reached final iteration
20-Apr-2022 20:58:31
9 min 17 sec
30 of 30
14040 of 14040
468
14040
100 iterations
Single GPU
Constant
0.01
Deep Learning Network Analyzer
net
Analysis date: 20-Apr-2022 21:07:56
13 layers
0 warnings
0 errors
imageinput
conv_1
batchnorm_1
relu_1
conv_2
batchnorm_2
relu_2
fc_1
batchnorm_3
relu_3
fc_2
softmax
classoutput
ANALYSIS RESULT
Name
Type
Activations
Learnables
imageinput
Image Input
28×28×1
conv_1
Convolution
28×28×32
Weights 3×3×1×32
Bias 1×1×32
batchnorm_1
Batch Normalization
28×28×32
Offset 1×1×32
Scale 1×1×32
relu_1
ReLU
28×28×32
conv_2
Convolution
28×28×32
Weights 3×3×32×32
Bias 1×1×32
batchnorm_2
Batch Normalization
28×28×32
Offset 1×1×32
Scale 1×1×32
relu_2
ReLU
28×28×32
fc_1
Fully Connected
1×1×300
Weights 300×25088
Bias 300×1
batchnorm_3
Batch Normalization
1×1×300
Offset 1×1×300
Scale 1×1×300
relu_3
ReLU
1×1×300
fc_2
Fully Connected
1×1×10
Weights 10×300
Bias 10×1
Accuracy (%)
Loss
Iteration
Loss
Training (smoothed)
Validation
Current Folder
Details
Type here to search
58°F Cloudy
ENG
9:09 PM
4/20/2022

Q6:

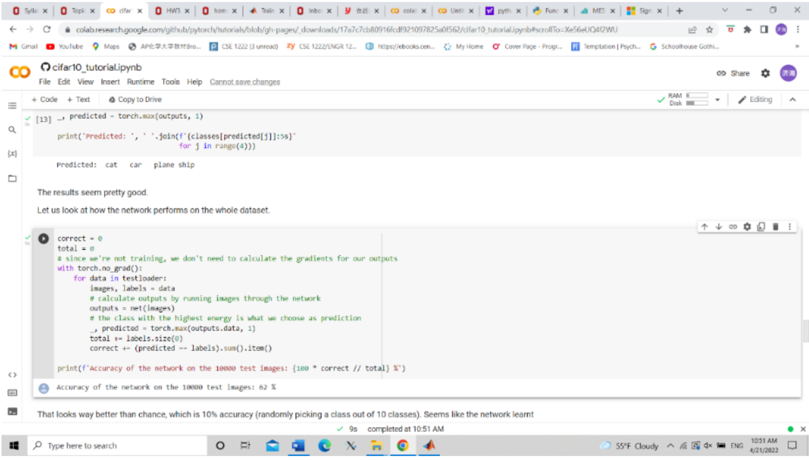
a)

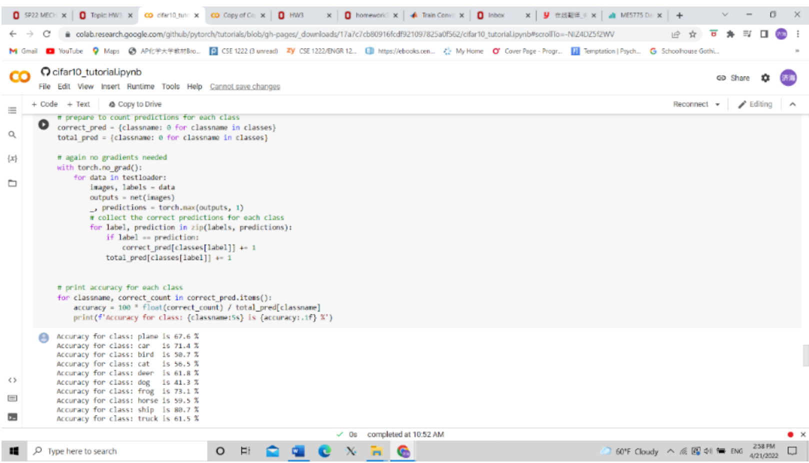

cifar10_tutorial.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
for epoch in range(20): # loop over the dataset multiple times
running_loss = 0.0
for i, data in enumerate(trainloader, 0):
# get the inputs; data is a list of [inputs, labels]
inputs, labels = data
# zero the parameter gradients
optimizer.zero_grad()
# forward + backward + optimize
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
# print statistics
running_loss += loss.item()
if i % 2000 == 1999:   # print every 2000 mini-batches
print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
running_loss = 0.0
print('Finished Training')
[1, 2000] loss: 2.178
[1, 4000] loss: 1.868
[1, 6000] loss: 1.696
[1, 8000] loss: 1.631
Type here to search


cifar10_tutorial.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
[1, 4000] loss: 1.868
[1, 6000] loss: 1.696
[1, 8000] loss: 1.631
[1, 10000] loss: 1.536
[1, 12000] loss: 1.466
[2, 2000] loss: 1.417
[2, 4000] loss: 1.402
[2, 6000] loss: 1.360
[2, 8000] loss: 1.323
[2, 10000] loss: 1.319
[2, 12000] loss: 1.286
[3, 2000] loss: 1.225
[3, 4000] loss: 1.224
[3, 6000] loss: 1.196
[3, 8000] loss: 1.197
[3, 10000] loss: 1.190
[3, 12000] loss: 1.164
[4, 2000] loss: 1.089
[4, 4000] loss: 1.115
[4, 6000] loss: 1.109
[4, 8000] loss: 1.115
[4, 10000] loss: 1.109
[4, 12000] loss: 1.105
[5, 2000] loss: 1.020
[5, 4000] loss: 1.038
[5, 6000] loss: 1.057
[5, 8000] loss: 1.039
[5, 10000] loss: 1.018
[5, 12000] loss: 1.049
[6, 2000] loss: 0.964
[6, 4000] loss: 0.985
[6, 6000] loss: 0.967
[6, 8000] loss: 0.987
[6, 10000] loss: 0.970
Type here to search

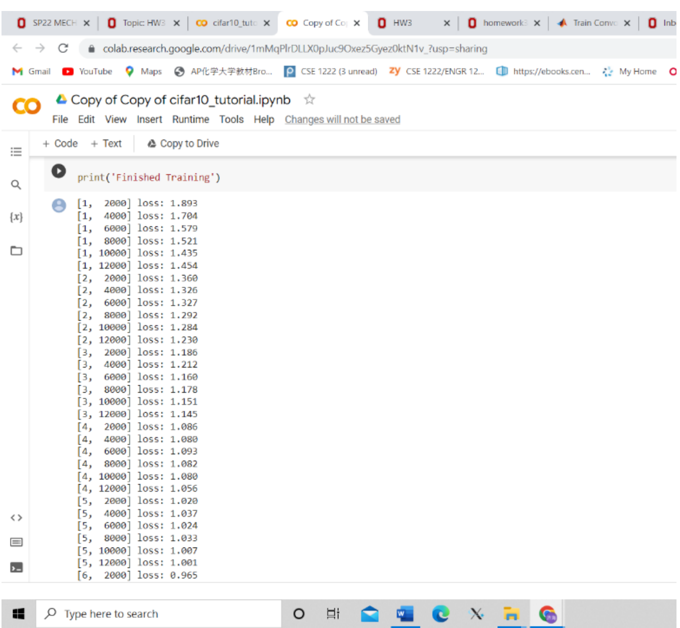

cifar10_tutorial.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
[13, 2000] loss: 0.694
[13, 4000] loss: 0.731
[13, 6000] loss: 0.763
[13, 8000] loss: 0.775
[13, 10000] loss: 0.783
[13, 12000] loss: 0.788
[14, 2000] loss: 0.691
[14, 4000] loss: 0.731
[14, 6000] loss: 0.736
[14, 8000] loss: 0.752
[14, 10000] loss: 0.745
[14, 12000] loss: 0.790
[15, 2000] loss: 0.663
[15, 4000] loss: 0.693
[15, 6000] loss: 0.723
[15, 8000] loss: 0.764
[15, 10000] loss: 0.759
[15, 12000] loss: 0.749
[16, 2000] loss: 0.660
[16, 4000] loss: 0.682
[16, 6000] loss: 0.705
[16, 8000] loss: 0.725
[16, 10000] loss: 0.730
[16, 12000] loss: 0.733
[17, 2000] loss: 0.623
[17, 4000] loss: 0.670
[17, 6000] loss: 0.694
[17, 8000] loss: 0.706
[17, 10000] loss: 0.721
[17, 12000] loss: 0.753
[18, 2000] loss: 0.622
[18, 4000] loss: 0.654
[18, 6000] loss: 0.688
Type here to search

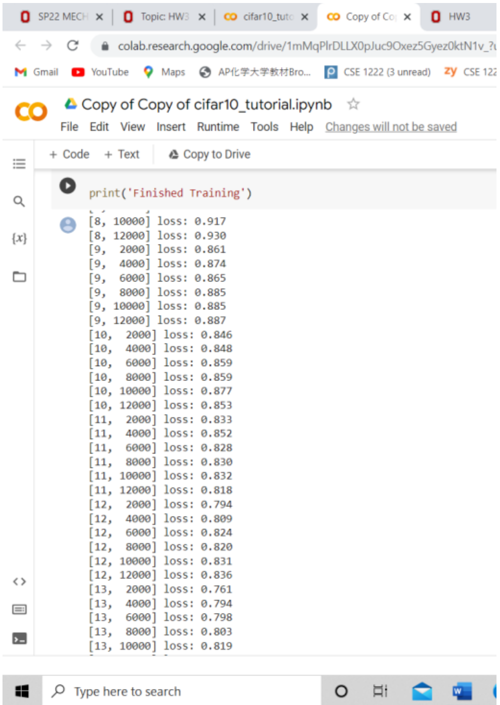
The accuracy is 62%

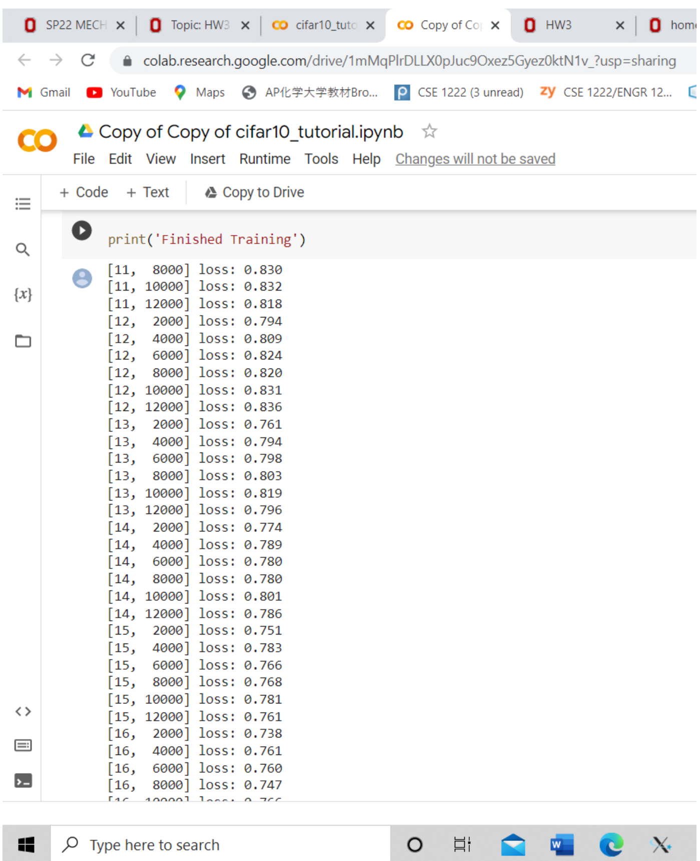


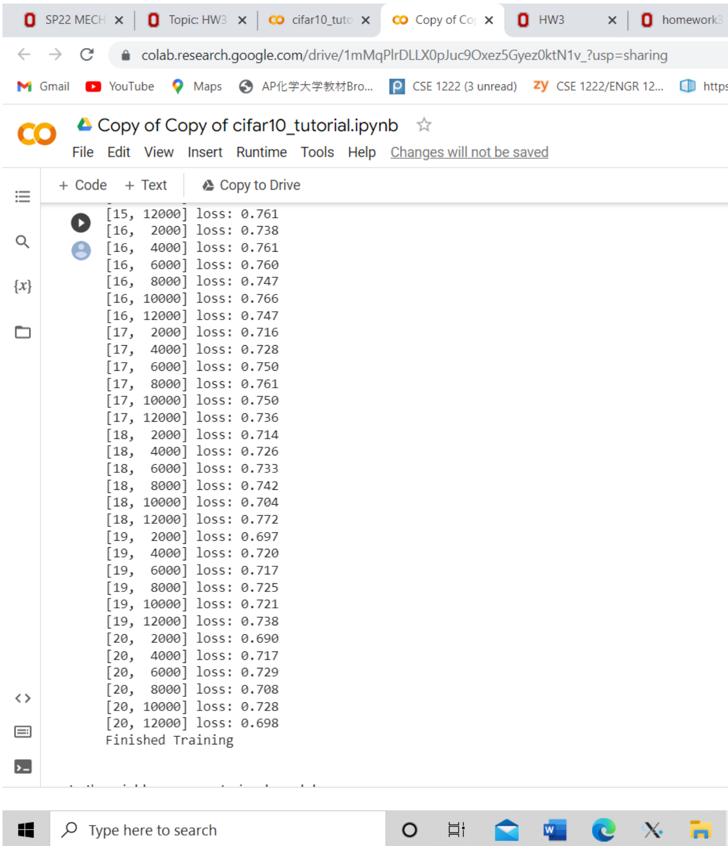


b)




colab.research.google.com/drive/1mMqPlrDLLX0pJuc9Oxez5Gyez0ktN1v_?
Copy of Copy of cifar10_tutorial.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text Copy to Drive
print('Finished Training')
[8, 10000] loss: 0.917
[8, 12000] loss: 0.930
[9, 2000] loss: 0.861
[9, 4000] loss: 0.874
[9, 6000] loss: 0.865
[9, 8000] loss: 0.885
[9, 10000] loss: 0.885
[9, 12000] loss: 0.887
[10, 2000] loss: 0.846
[10, 4000] loss: 0.848
[10, 6000] loss: 0.859
[10, 8000] loss: 0.859
[10, 10000] loss: 0.877
[10, 12000] loss: 0.853
[11, 2000] loss: 0.833
[11, 4000] loss: 0.852
[11, 6000] loss: 0.828
[11, 8000] loss: 0.830
[11, 10000] loss: 0.832
[11, 12000] loss: 0.818
[12, 2000] loss: 0.794
[12, 4000] loss: 0.809
[12, 6000] loss: 0.824
[12, 8000] loss: 0.820
[12, 10000] loss: 0.831
[12, 12000] loss: 0.836
[13, 2000] loss: 0.761
[13, 4000] loss: 0.794
[13, 6000] loss: 0.798
[13, 8000] loss: 0.803
[13, 10000] loss: 0.819
Type here to search


colab.research.google.com/drive/1mMqPlrDLLX0pJuc9Oxez5Gyez0ktN1v_?usp=sharing
Copy of Copy of cifar10_tutorial.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text Copy to Drive
print('Finished Training')
[11, 8000] loss: 0.830
[11, 10000] loss: 0.832
[11, 12000] loss: 0.818
[12, 2000] loss: 0.794
[12, 4000] loss: 0.809
[12, 6000] loss: 0.824
[12, 8000] loss: 0.820
[12, 10000] loss: 0.831
[12, 12000] loss: 0.836
[13, 2000] loss: 0.761
[13, 4000] loss: 0.794
[13, 6000] loss: 0.798
[13, 8000] loss: 0.803
[13, 10000] loss: 0.819
[13, 12000] loss: 0.796
[14, 2000] loss: 0.774
[14, 4000] loss: 0.789
[14, 6000] loss: 0.780
[14, 8000] loss: 0.780
[14, 10000] loss: 0.801
[14, 12000] loss: 0.786
[15, 2000] loss: 0.751
[15, 4000] loss: 0.783
[15, 6000] loss: 0.766
[15, 8000] loss: 0.768
[15, 10000] loss: 0.781
[15, 12000] loss: 0.761
[16, 2000] loss: 0.738
[16, 4000] loss: 0.761
[16, 6000] loss: 0.760
[16, 8000] loss: 0.747
Type here to search



The accuracy is 66%

The results seem pretty good.

Let us look at how the network performs on the whole dataset.

```
[ ] correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Accuracy of the network on the 10000 test images: %d %%' % (
        100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 66 %
```

```python
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 72 %
Accuracy of   car : 81 %
Accuracy of  bird : 58 %
Accuracy of   cat : 50 %
Accuracy of  deer : 55 %
Accuracy of   dog : 52 %
Accuracy of  frog : 71 %
Accuracy of horse : 74 %
Accuracy of  ship : 78 %
Accuracy of truck : 68 %
```

c)

To increase the accuracy, I choose to increase the epochs from 20 to 30. The accuracy only increase a lit bit but running for a much longer time.

File Edit View Insert Runtime

+ Code + Text Copy to Drive

```
[1,  8000] loss: 1.544
[1, 10000] loss: 1.469
[1, 12000] loss: 1.432
[2,  2000] loss: 1.350
[2,  4000] loss: 1.338
[2,  6000] loss: 1.324
[2,  8000] loss: 1.269
[2, 10000] loss: 1.256
[2, 12000] loss: 1.251
[3,  2000] loss: 1.167
[3,  4000] loss: 1.198
[3,  6000] loss: 1.168
[3,  8000] loss: 1.170
[3, 10000] loss: 1.144
[3, 12000] loss: 1.126
[4,  2000] loss: 1.073
[4,  4000] loss: 1.096
[4,  6000] loss: 1.100
[4,  8000] loss: 1.083
[4, 10000] loss: 1.065
[4, 12000] loss: 1.055
[5,  2000] loss: 1.013
[5,  4000] loss: 1.018
[5,  6000] loss: 1.032
[5,  8000] loss: 1.046
[5, 10000] loss: 0.978
[5, 12000] loss: 1.025
[6,  2000] loss: 0.963
[6,  4000] loss: 0.962
[6,  6000] loss: 0.979
[6,  8000] loss: 0.970
[6, 10000] loss: 0.978
[6, 12000] loss: 0.968
[7,  2000] loss: 0.913
[7,  4000] loss: 0.931
```

File Edit View Insert Runtime Tools Hel

+ Code + Text Copy to Drive

```
[16,  6000] loss: 0.756
[16,  8000] loss: 0.753
[16, 10000] loss: 0.762
[16, 12000] loss: 0.789
[17,  2000] loss: 0.704
[17,  4000] loss: 0.727
[17,  6000] loss: 0.732
[17,  8000] loss: 0.760
[17, 10000] loss: 0.758
[17, 12000] loss: 0.761
[18,  2000] loss: 0.704
[18,  4000] loss: 0.734
[18,  6000] loss: 0.720
[18,  8000] loss: 0.742
[18, 10000] loss: 0.748
[18, 12000] loss: 0.737
[19,  2000] loss: 0.695
[19,  4000] loss: 0.710
[19,  6000] loss: 0.718
[19,  8000] loss: 0.700
[19, 10000] loss: 0.733
[19, 12000] loss: 0.744
[20,  2000] loss: 0.679
[20,  4000] loss: 0.721
[20,  6000] loss: 0.709
[20,  8000] loss: 0.711
[20, 10000] loss: 0.711
[20, 12000] loss: 0.736
[21,  2000] loss: 0.682
[21,  4000] loss: 0.688
[21,  6000] loss: 0.689
[21,  8000] loss: 0.697
```

Let's quickly save our trained model:

File Edit View Insert Runtime Tools

+ Code + Text Copy to Drive

```
[26,  2000] loss: 0.627
[26,  4000] loss: 0.636
[26,  6000] loss: 0.661
[26,  8000] loss: 0.655
[26, 10000] loss: 0.689
[26, 12000] loss: 0.672
[27,  2000] loss: 0.619
[27,  4000] loss: 0.641
[27,  6000] loss: 0.649
[27,  8000] loss: 0.642
[27, 10000] loss: 0.664
[27, 12000] loss: 0.670
[28,  2000] loss: 0.606
[28,  4000] loss: 0.621
[28,  6000] loss: 0.650
[28,  8000] loss: 0.658
[28, 10000] loss: 0.662
[28, 12000] loss: 0.651
[29,  2000] loss: 0.614
[29,  4000] loss: 0.620
[29,  6000] loss: 0.638
[29,  8000] loss: 0.647
[29, 10000] loss: 0.651
[29, 12000] loss: 0.649
[30,  2000] loss: 0.605
[30,  4000] loss: 0.615
[30,  6000] loss: 0.640
[30,  8000] loss: 0.638
[30, 10000] loss: 0.625
[30, 12000] loss: 0.622
Finished Training
```

The accuracy is 67%

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

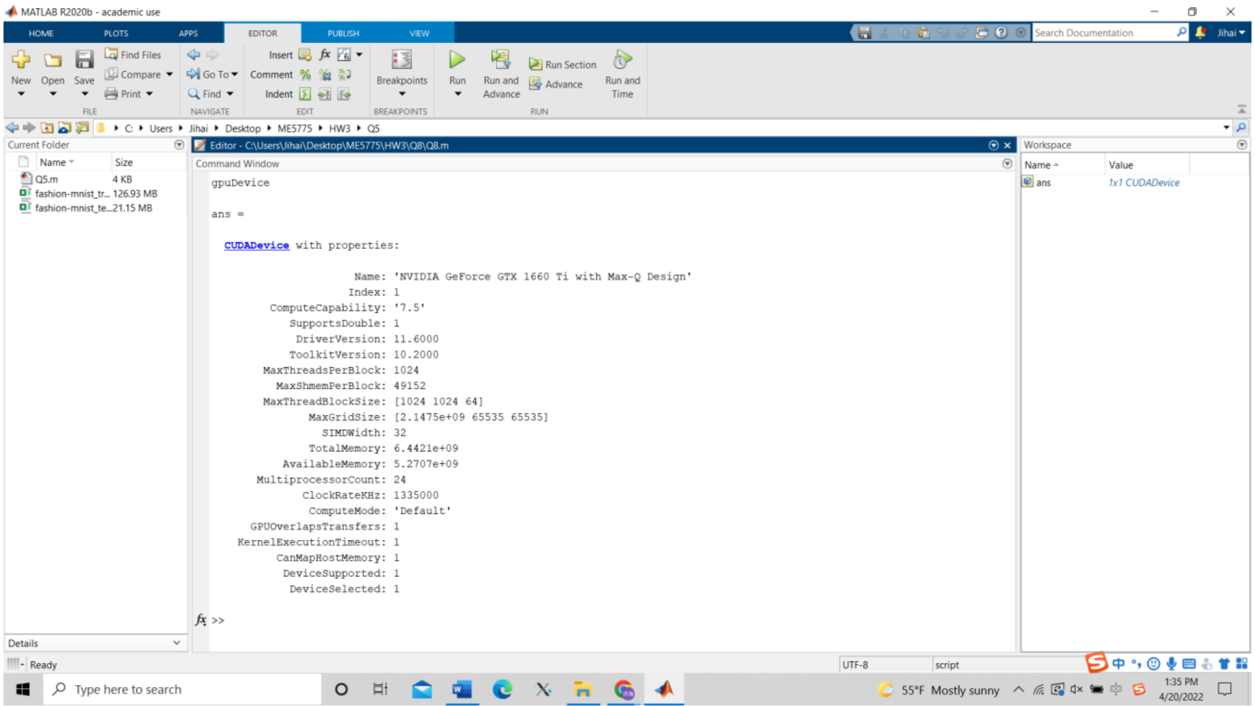
```
Accuracy of the network on the 10000 test images: 67 %
```

```
for i in range(10):
    print('Accuracy of %5s : %2d %%'
        classes[i], 100 * class_corr
```

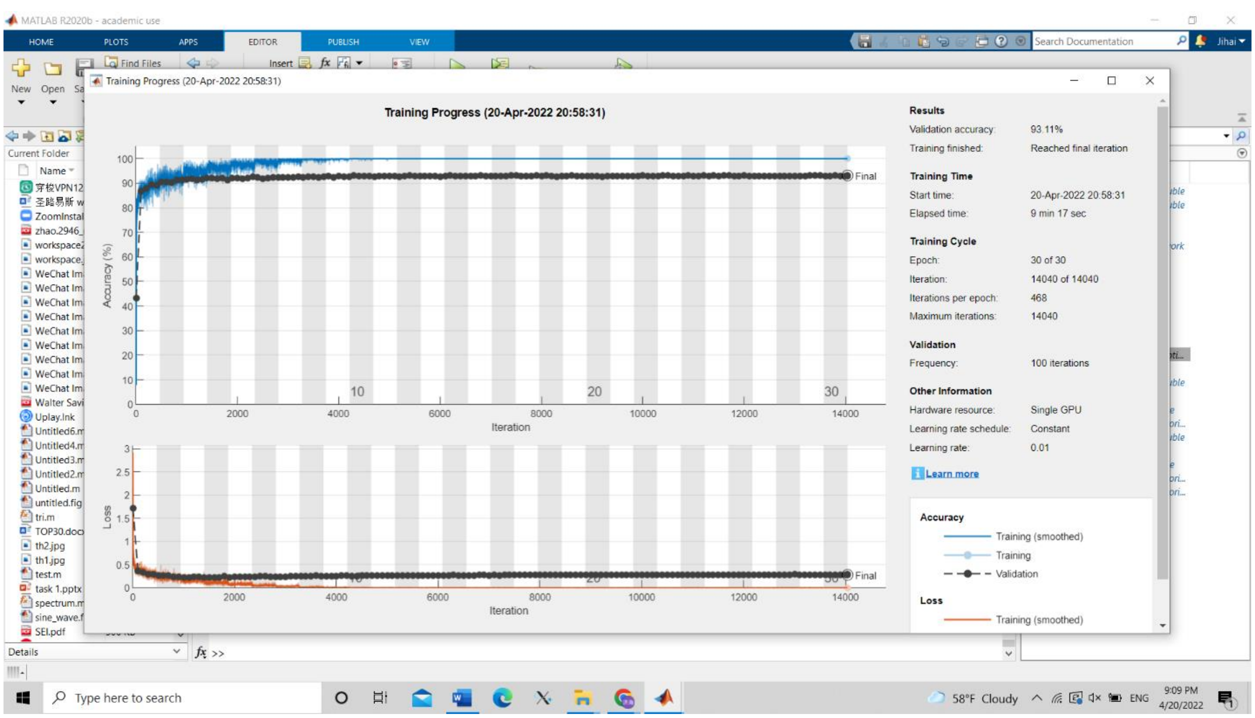
```
Accuracy of plane : 74 %
Accuracy of   car : 75 %
Accuracy of  bird : 58 %
Accuracy of   cat : 42 %
Accuracy of  deer : 68 %
Accuracy of   dog : 61 %
Accuracy of  frog : 70 %
Accuracy of horse : 70 %
Accuracy of  ship : 77 %
Accuracy of truck : 73 %
```

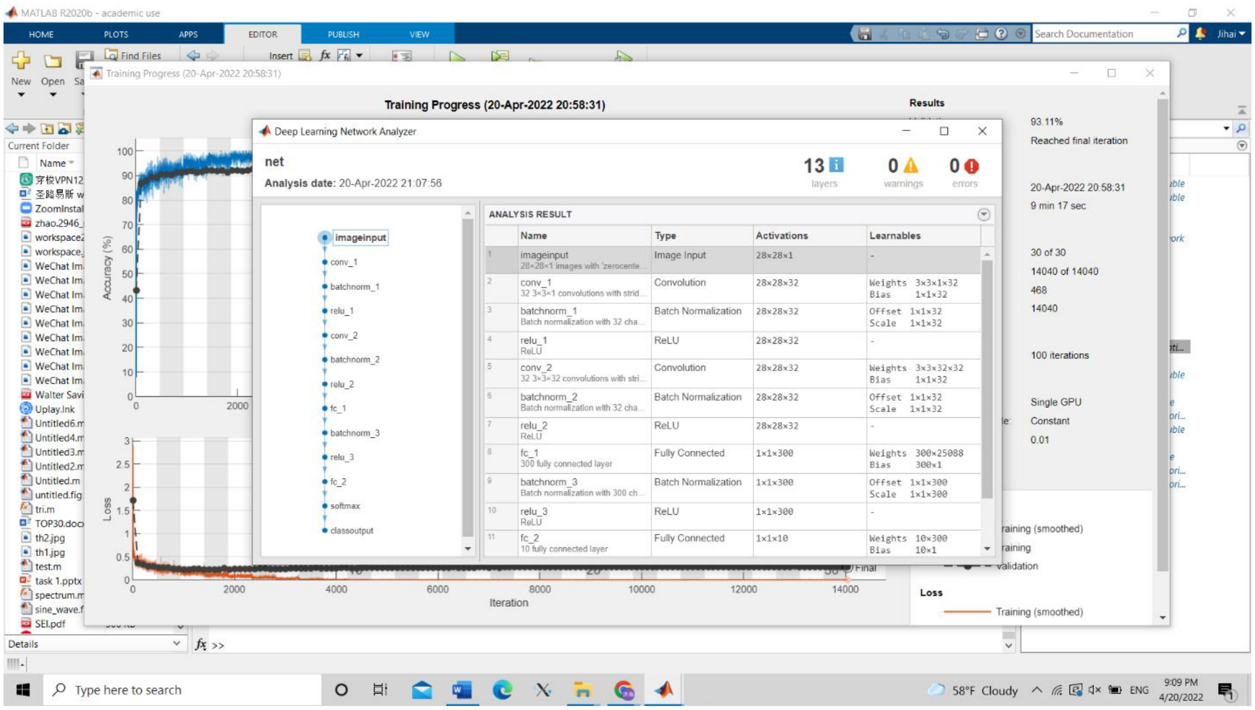
Q8:

a) My computer has a GPU

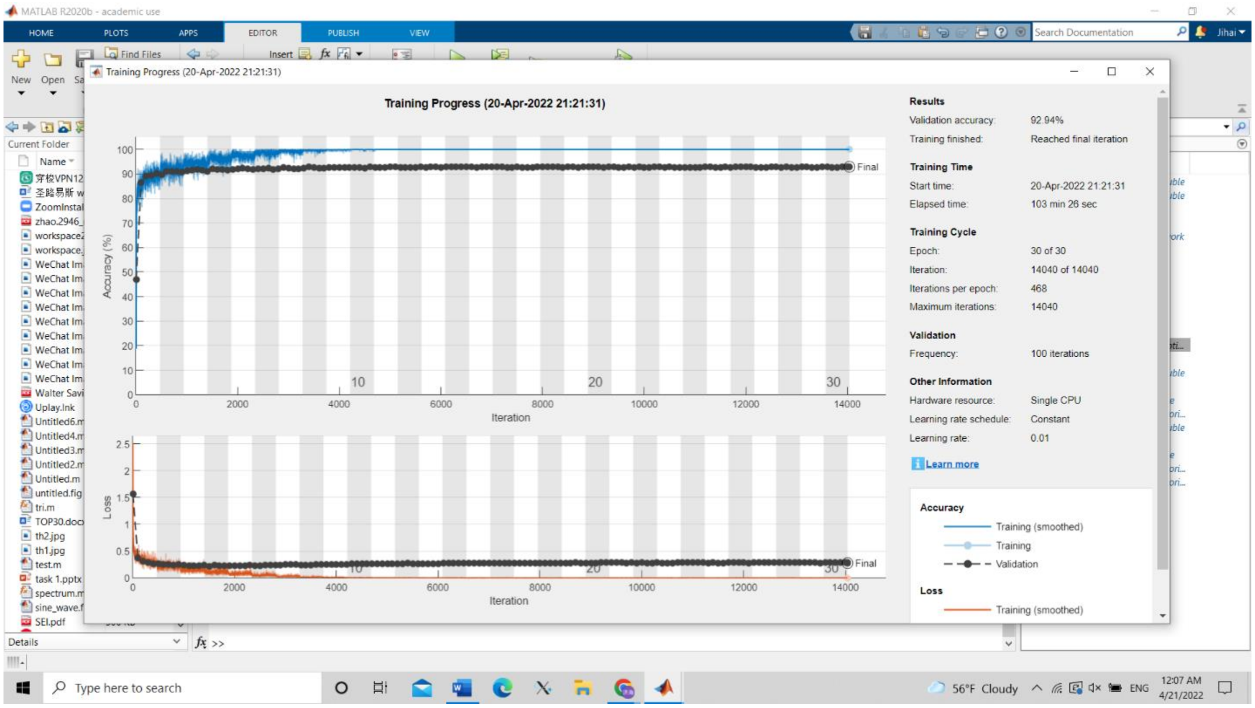


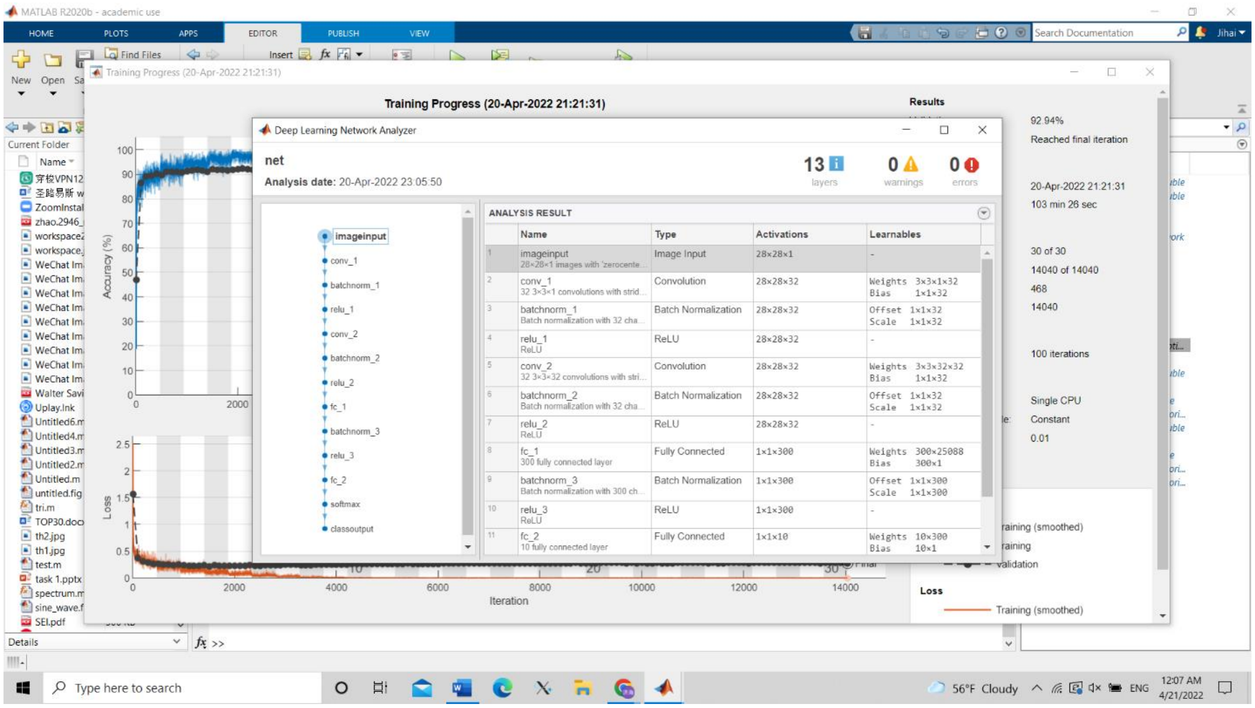
b) My computer will be training on single GPU as default. With double convolution2DLayer, bathchnormalization, and reLULayer, and trio of layers: fully connected + reLu + batchnorm, the running time is 9 min 17 sec and the accuracy is 93.11%.





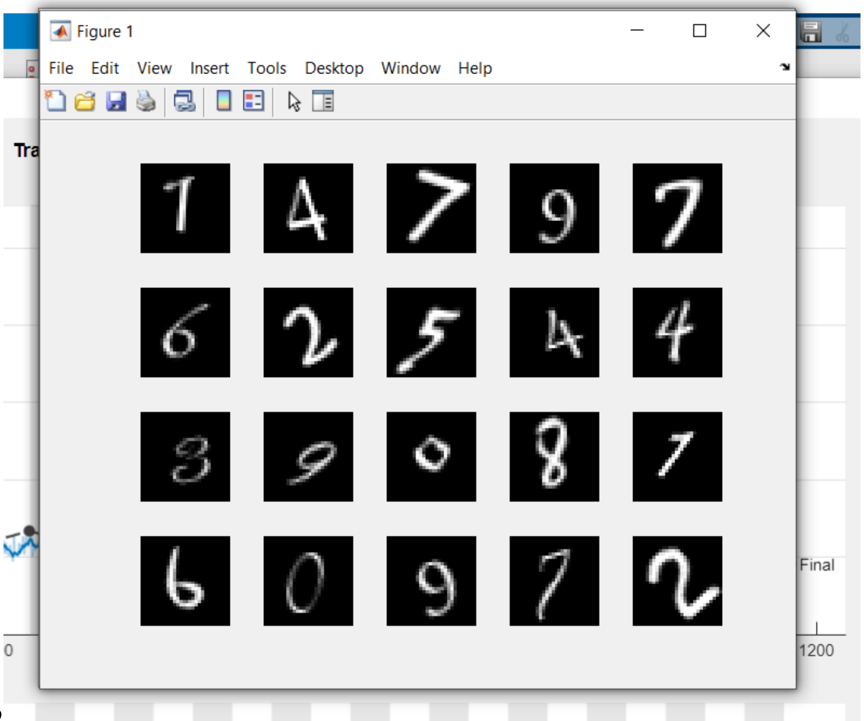
Then I force my computer to run with cpu. In the same condition, CPU is much slower than GPU. The running time is 103 min 26 sec and the accuracy is 92.94%.




MATLAB R2020b - academic use
Training Progress (20-Apr-2022 21:21:31)
Results
Deep Learning Network Analyzer
net
Analysis date: 20-Apr-2022 23:05:50
13 layers
0 warnings
0 errors
ANALYSIS RESULT
Name Type Activations Learnables
imageinput Image Input 28×28×1 -
conv_1 Convolution 28×28×32 Weights 3×3×1×32 Bias 1×1×32
batchnorm_1 Batch Normalization 28×28×32 Offset 1×1×32 Scale 1×1×32
relu_1 ReLU 28×28×32 -
conv_2 Convolution 28×28×32 Weights 3×3×32×32 Bias 1×1×32
batchnorm_2 Batch Normalization 28×28×32 Offset 1×1×32 Scale 1×1×32
relu_2 ReLU 28×28×32 -
fc_1 Fully Connected 1×1×300 Weights 300×25088 Bias 300×1
batchnorm_3 Batch Normalization 1×1×300 Offset 1×1×300 Scale 1×1×300
relu_3 ReLU 1×1×300 -
fc_2 Fully Connected 1×1×10 Weights 10×300 Bias 10×1
92.94%
Reached final iteration
20-Apr-2022 21:21:31
103 min 26 sec
30 of 30
14040 of 14040
468
14040
100 iterations
Single CPU
Constant
0.01
Accuracy (%)
Loss
Iteration
Training (smoothed)
Training
Validation
Type here to search
56°F Cloudy
12:07 AM
4/21/2022

Q9:

This example shows how to fit a regression model using convolutional neural networks to predict the angles of rotation of handwritten digits. The example constructs a convolutional neural network architecture, trains a network, and uses the trained network to predict angles of rotated handwritten digits.



02

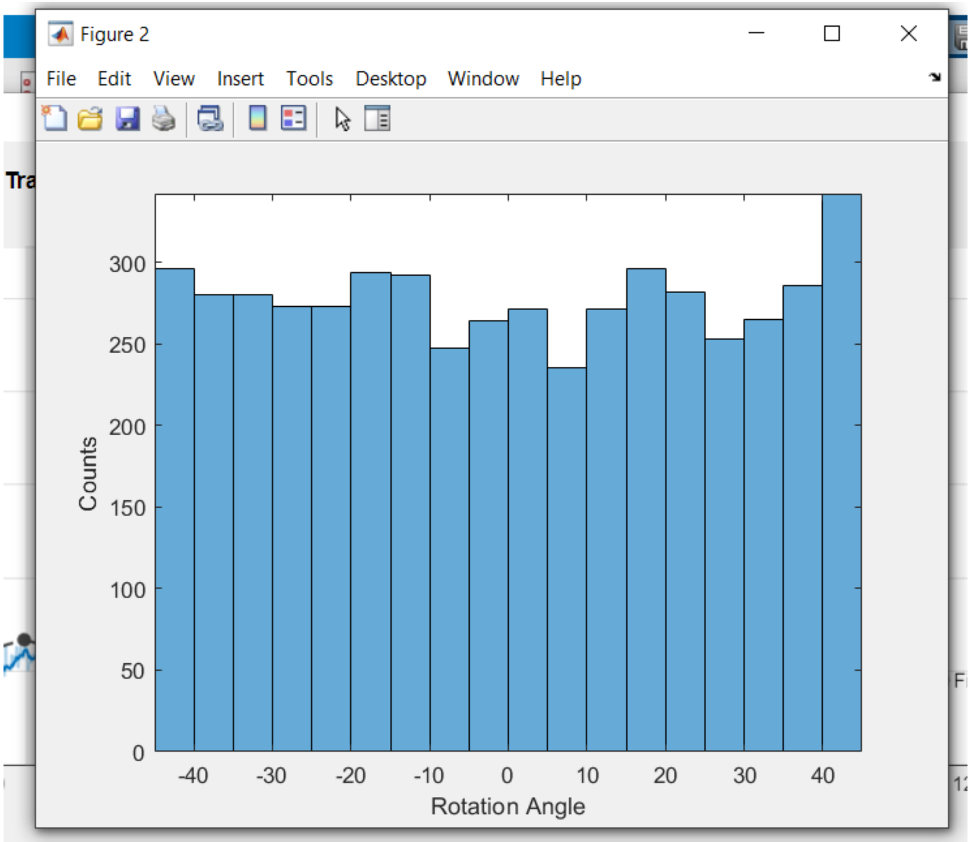


Figure 2
File Edit View Insert Tools Desktop Window Help
Counts
Rotation Angle
0
50
100
150
200
250
300
-40 -30 -20 -10 0 10 20 30 40

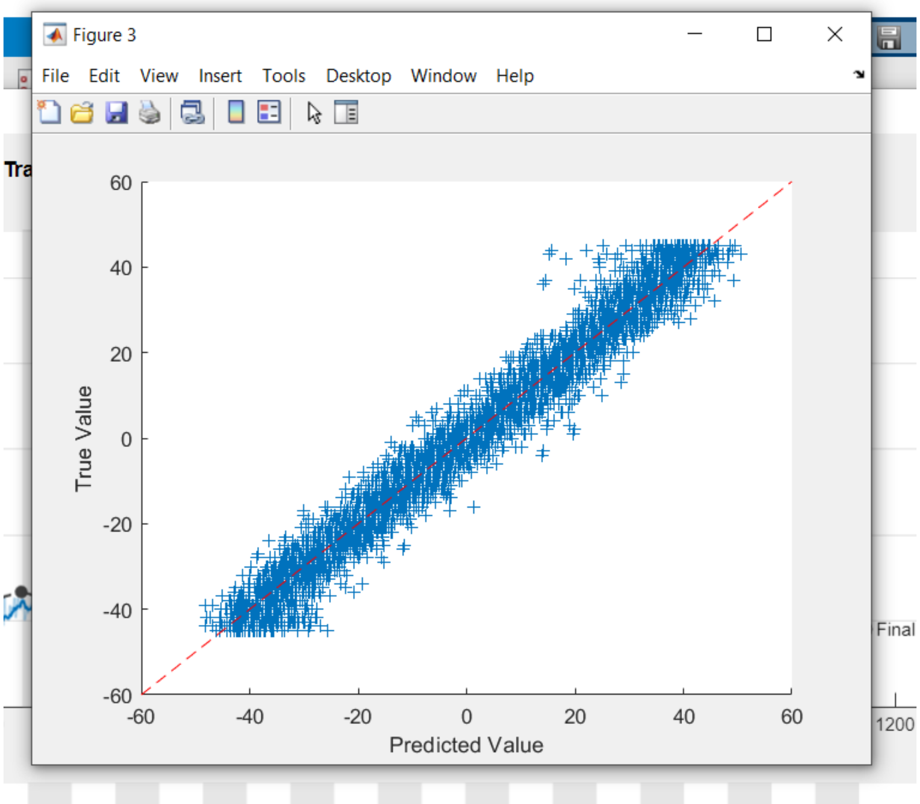


Figure 3
File Edit View Insert Tools Desktop Window Help
True Value
Predicted Value
60
40
20
0
-20
-40
-60
-60 -40 -20 0 20 40 60

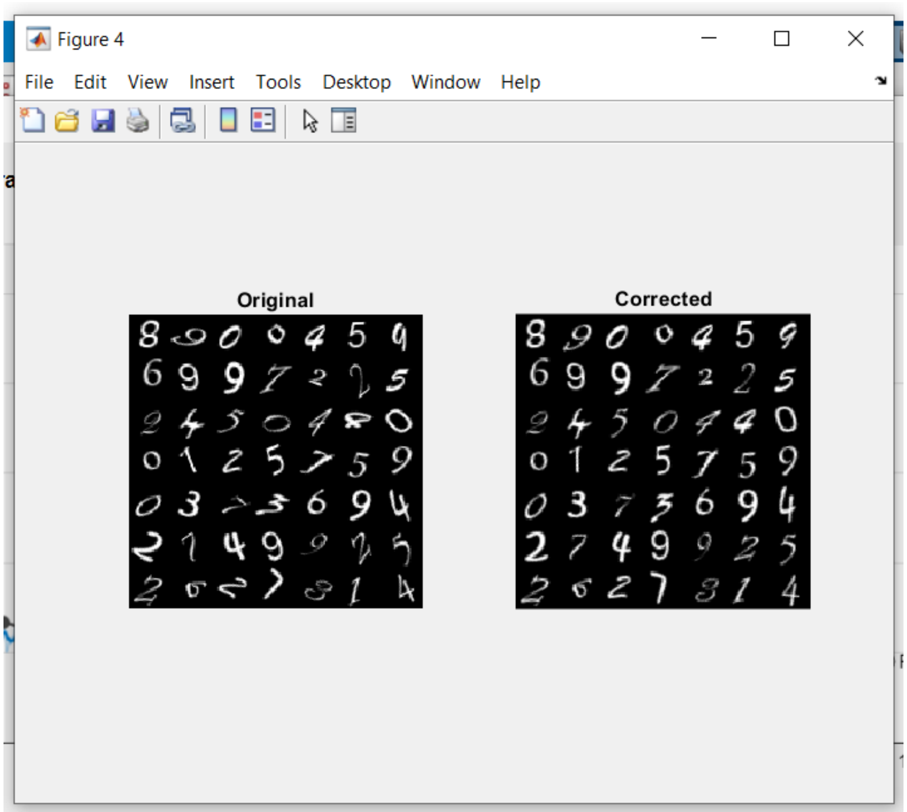
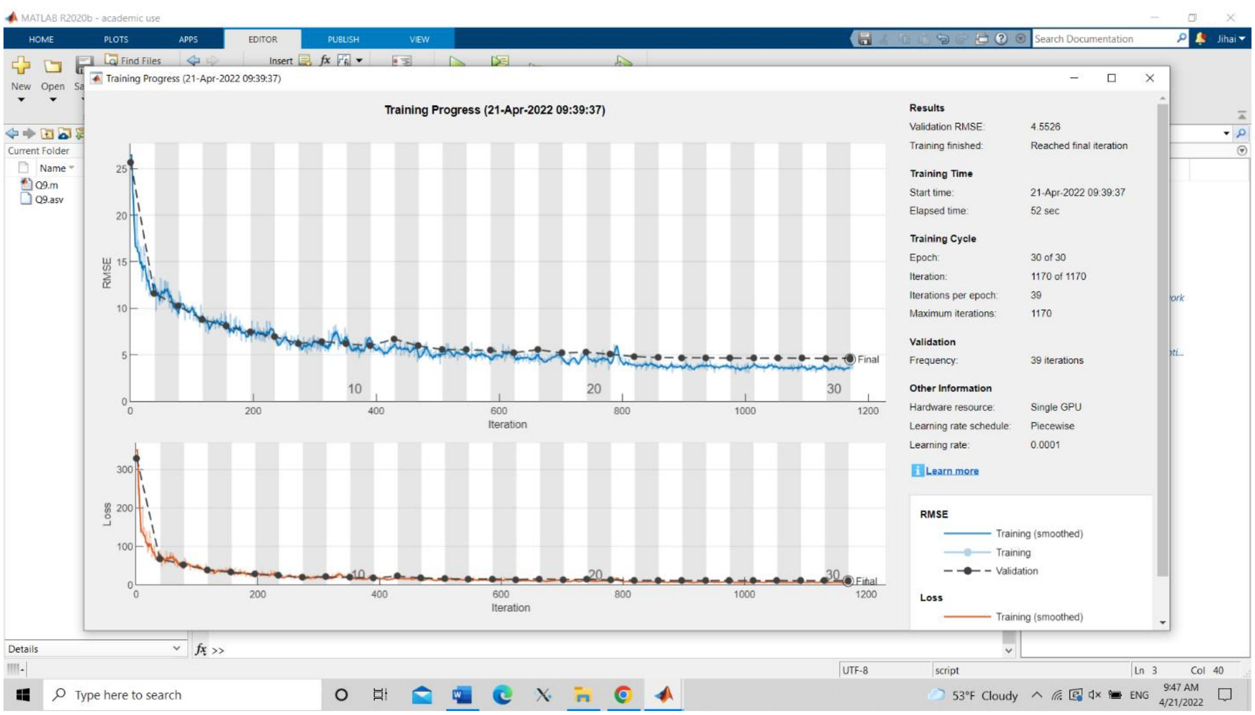


Figure 4
File Edit View Insert Tools Desktop Window Help
Original
Corrected


MATLAB R2020b - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Search Documentation
Training Progress (21-Apr-2022 09:39:37)
RMSE
Iteration
Loss
Final
Results
Validation RMSE: 4.5526
Training finished: Reached final iteration
Training Time
Start time: 21-Apr-2022 09:39:37
Elapsed time: 52 sec
Training Cycle
Epoch: 30 of 30
Iteration: 1170 of 1170
Iterations per epoch: 39
Maximum iterations: 1170
Validation
Frequency: 39 iterations
Other Information
Hardware resource: Single GPU
Learning rate schedule: Piecewise
Learning rate: 0.0001
Learn more
RMSE
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Current Folder
Name
Q9.m
Q9.asv
Details
Type here to search
53°F Cloudy
ENG
9:47 AM
4/21/2022