**Department of Electronic and Telecommunication Engineering**

**University of Moratuwa, Sri Lanka**

**EN2550 - Fundamentals of Image Processing and Machine Vision**



**ASSIGMENT 4**

**Submitted By**

Nagasinghe K.R.Y 180411K

**Submitted on**

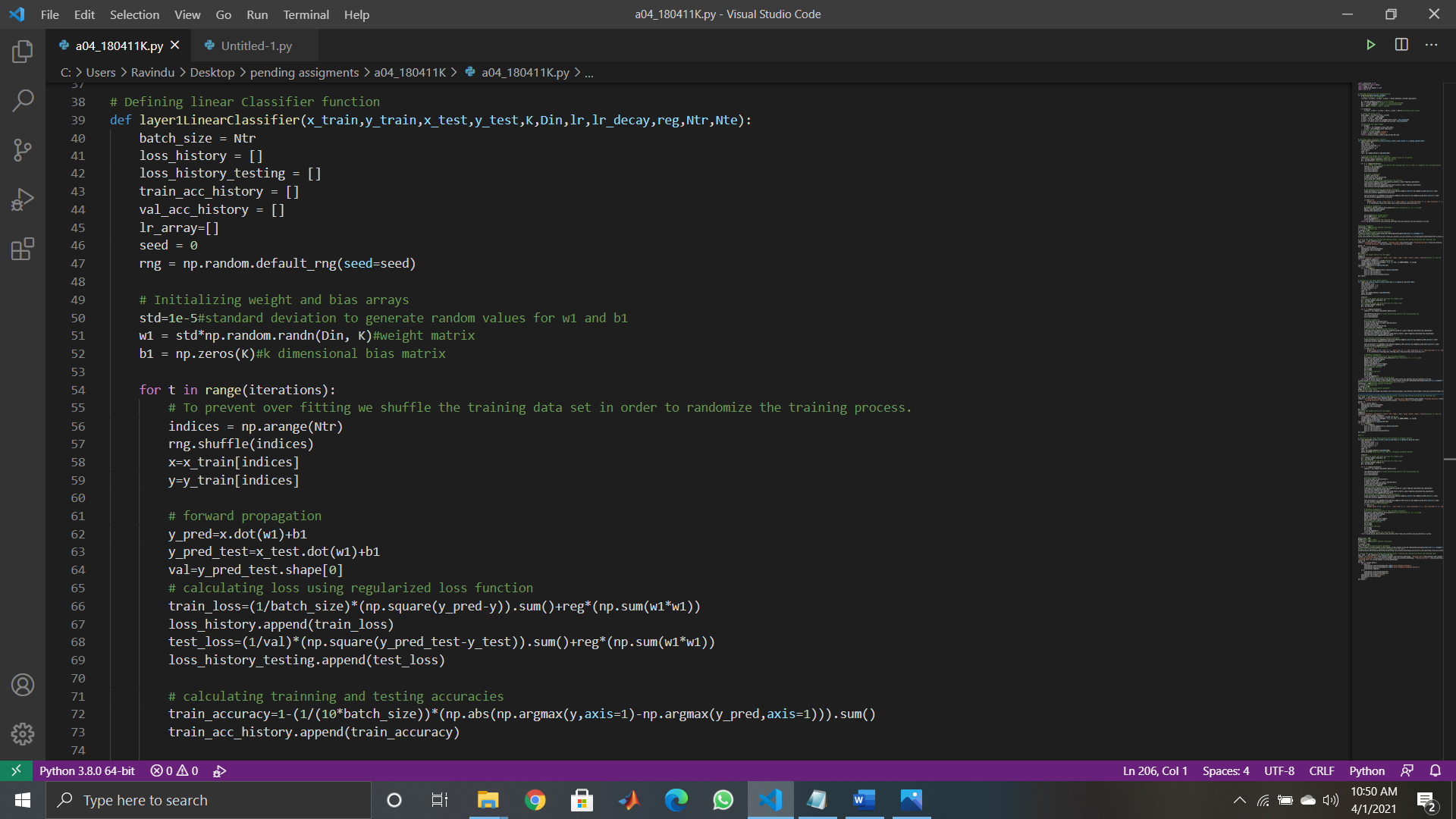
**April 2, 2021**

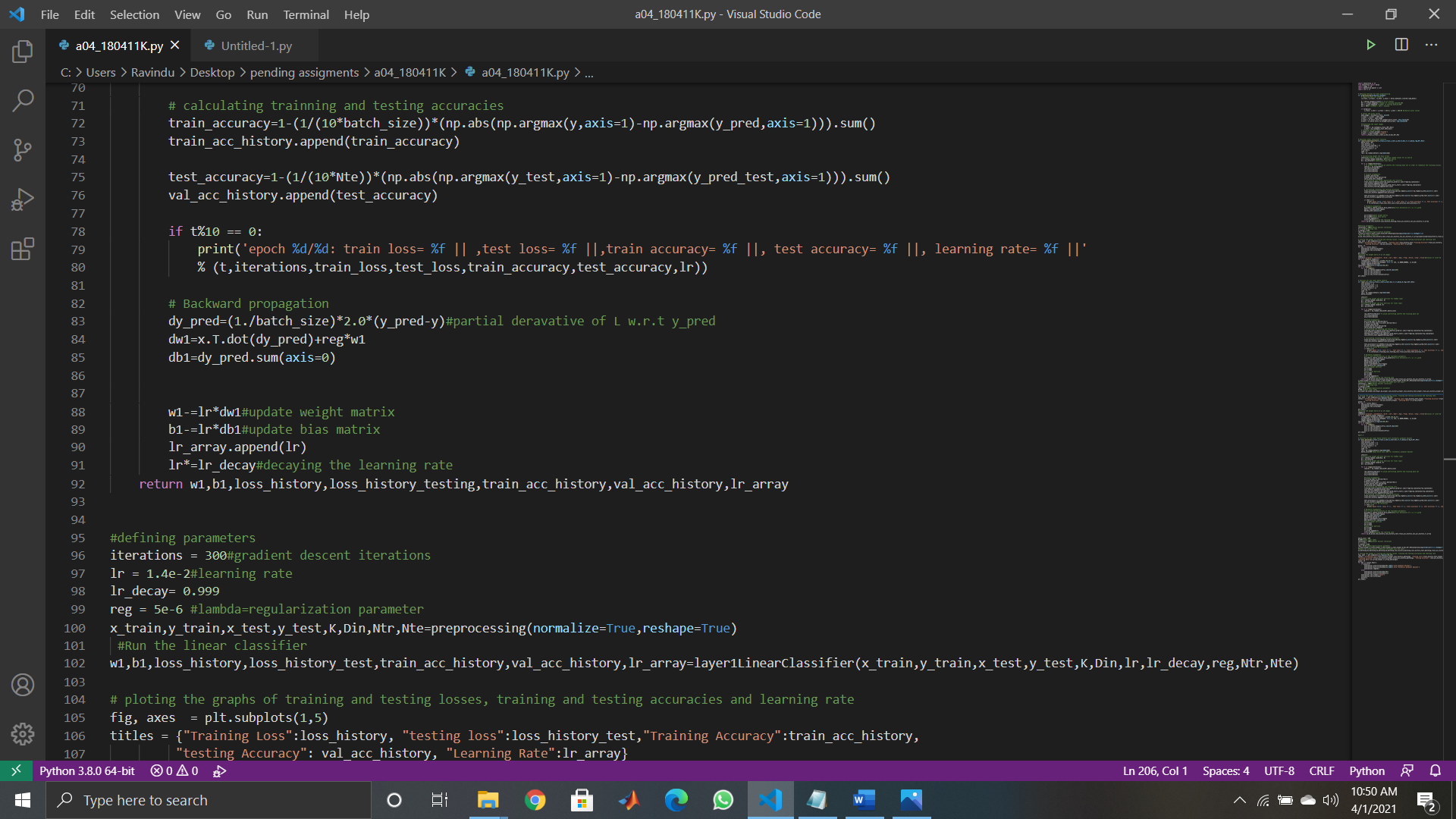
Full code: <https://github.com/Ravindu-Yasas-Nagasinghe/EN2550-Computer-Vision-and-Image-Processing-Assigments>

**1)Linear Classification using gradient descent.**

Text

Description automatically generatedHere our data set is CIFAR-10. There are 10 different classes in this data set. I use tensorflow to import the data set to python. Our score function for the linear classifier is f (x) = W x +b, and the loss function is the mean sum of squared errors function. I run for 300 epochs as instructed in the assignment. The code for 1layer linear classifier using gradient descent is as follows.





A picture containing text, gallery, room, colorful

Description automatically generatedAfter that I plot the weight matrix W, as 10 images and plot the training loss, testing loss, training accuracy, testing accuracy and learning rate. W1 weight array is of the shape 3072 x 10. (Code for plotting is not included due to page constraints. Full code is available at the link on cover page).

Weight matrix as 10 images

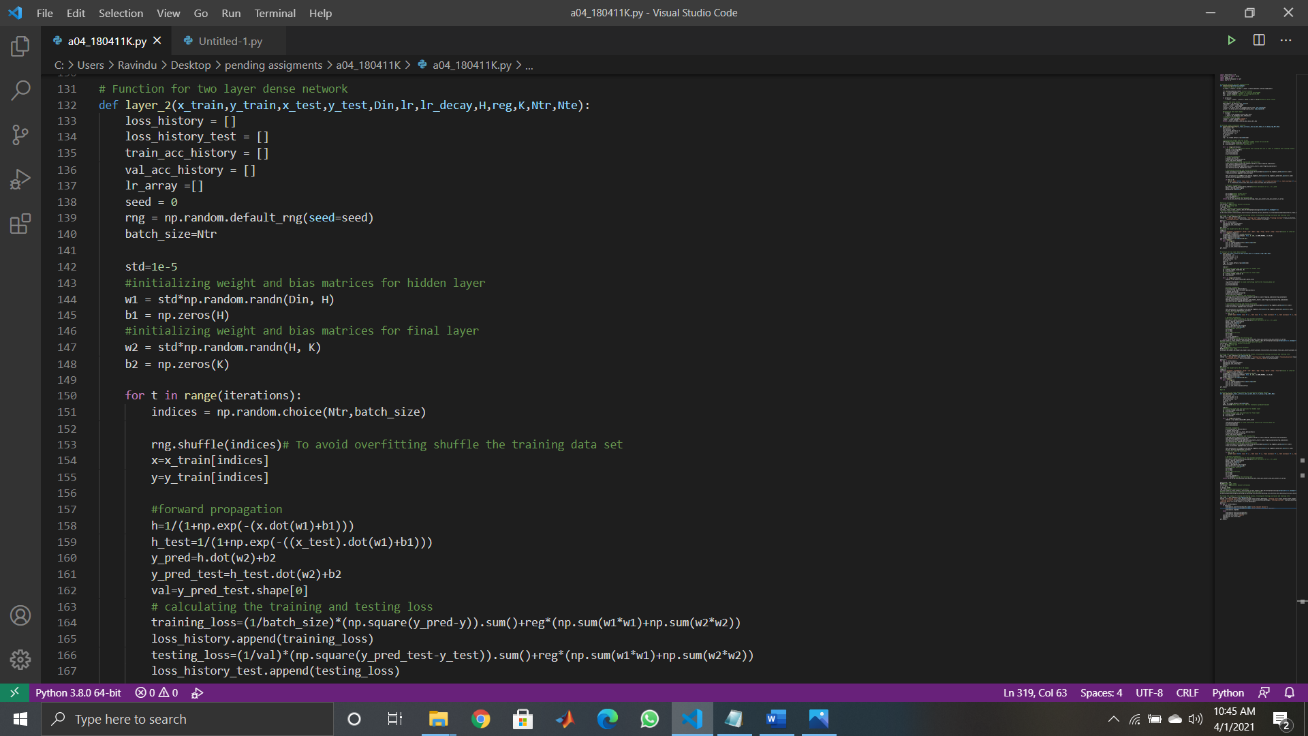
A picture containing histogram

Description automatically generatedInitial learning rate = 1.4 x 10

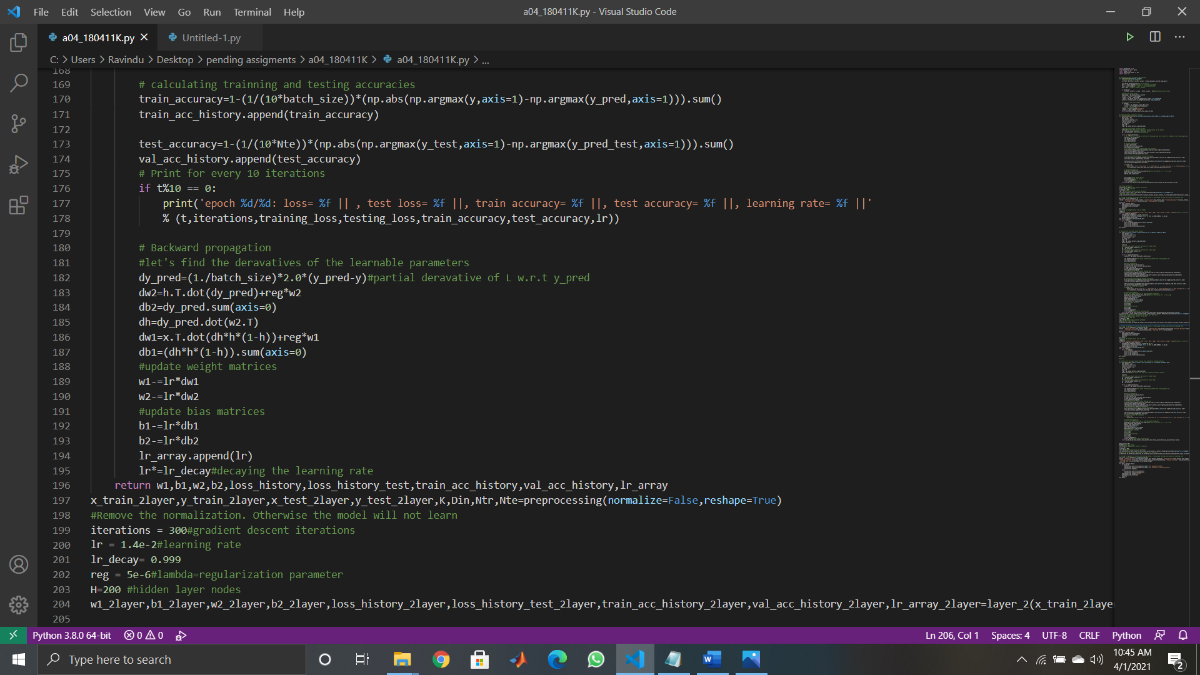
Loss, testing loss, training accuracy, testing accuracy, learning rate of the linear classifier for 300 epochs.

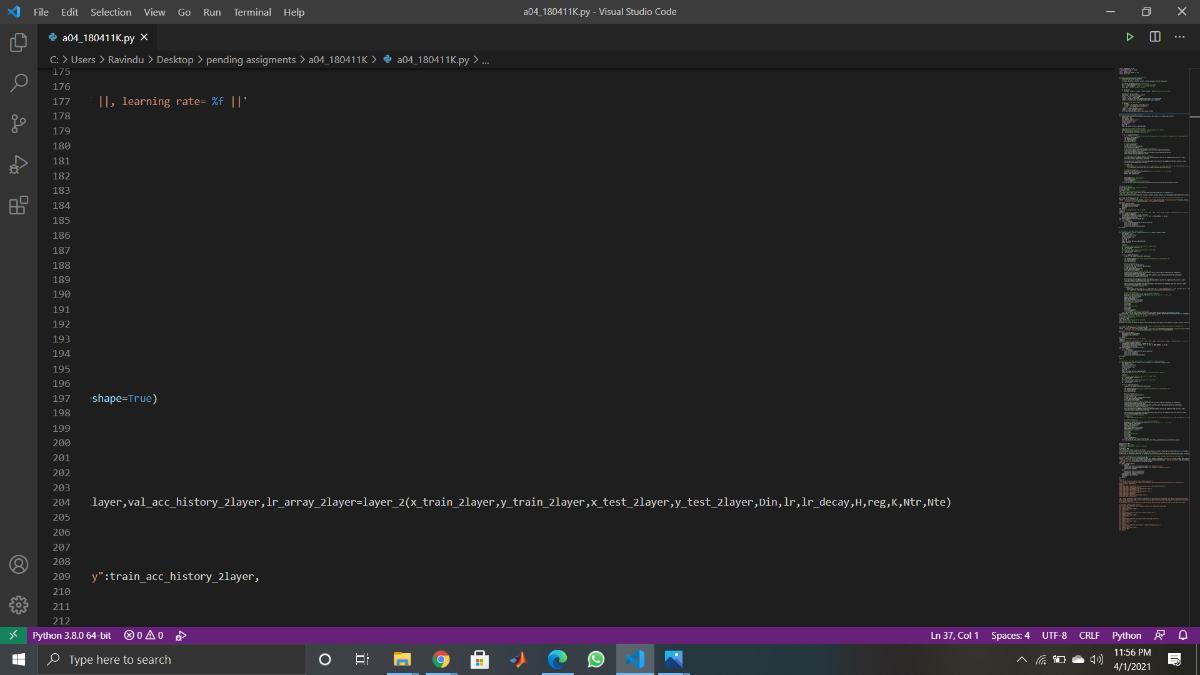
After 300 epochs loss= 0.783117, test loss= 0.787730, train accuracy= 0.779958, test accuracy= 0.774120, learning rate= 0.010474.

**2) 2 layer fully connected network**

A computer screen capture

Description automatically generated with low confidenceHere I use a two-layer dense network with H=200 hidden nodes. Code for this part is as follows.





Here gradients are computed in the direction from output to input layers and combined using chain rule.

* Input layer to hidden layer
* Hidden layer to the output layer
* Total number of learnable parameters in the network = (200 x 3072+200) + (10 x 200+10) = 616,610

We preprocess the input data set without pixel normalization to avoid underfitting.

As we can see from the below results when the number of iterations increase the loss decreases and accuracy increases. But the rate of the loss decreasing and accuracy increasing reduces with the iterations.

As we can see from the results, when we use 2 layer fully connected network instead of single layer as in part 1, we can reduce the train and test loss and increase training and testing accuracy. So, if we increase the number of layers further, we can achieve more and more accuracy and reduce loss.

Chart, histogram

Description automatically generatedInitial learning rate = 1.4 x 102

Loss, testing loss, training accuracy, testing accuracy, learning rate of the 2 layer fully connected network for 300 epochs.

After 300 epochs, loss= 0.735866, test loss= 0.759436, train accuracy= 0.795186, test accuracy= 0.784170 and learning rate= 0.01047|.

**3) Stochastic gradient descent with a batch size of 500**.

**A screenshot of a computer

Description automatically generated with medium confidence**Here instead of using 50,000 samples I only uses 500 samples which are selected randomly. The code for this part is as follows. (only the parts different from the code in part 2 is included. The full code is available in the link at cover page).

**A picture containing text, screenshot, monitor, computer

Description automatically generatedText

Description automatically generated**

Here we compute the gradients off the loss function w.r.t to the functions constructed only using 500 samples. It is advantageous to use stochastic gradient descent instead of gradient descent because we can avoid being stuck at a local minimum instead of global minimum when finding loss. By using SGD we can compute errors and updates weights much faster as batch size is low. SGD often converges much faster compared to GD. One disadvantage of SGD is that loss function can be much noisier than GD.

Chart, histogram

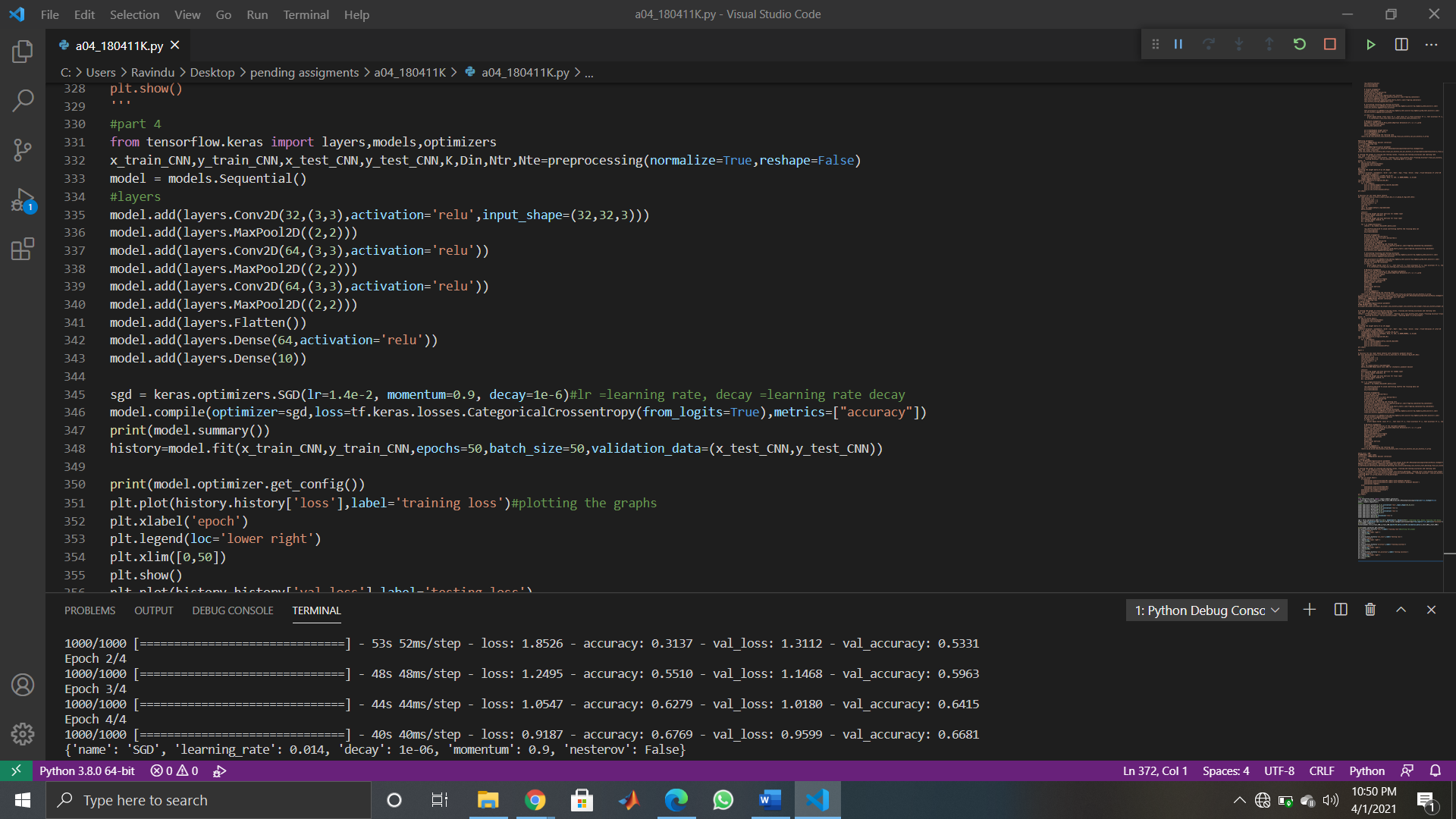
Description automatically generated

Loss, testing loss, training accuracy, testing accuracy, learning rate of the 2 layer fully connected network using gradient descent and stochastic gradient descent for 300 iterations.

After 300 iterations for stochastic gradient descent with batch size =500, loss= 0.737830, test loss= 0.765587, train accuracy= 0.808400, test accuracy= 0.783190 and learning rate = 0.010474.

By comparing the plots from part2(GD) and SGD, we can state that accuracy has been slightly increased when using SGD instead of GD. By observing the loss plots we can state that SGD has reached the convergence much faster than GD, but SGD is much noisier and has higher loss than GD.

**4) CNN**

Here I preprocess data without normalization of pixels and without reshaping the images. As seen from the results, we can clearly state that CNN is overfitting. Here I used sgd optimizer with momentum. The code is as follows. (In plotting code part, only code for plotting loss is included)

Here I used,

learning rate = 0.014

momentum = 0.9.

learning rate decay =1e-6

There are 73,418 total learnable parameters in this network.

After 50 epochs loss: 0.2491, accuracy: 0.9164, testing loss: 1.7655, testing accuracy: 0.6914

**Chart, line chart

Description automatically generated**Chart

Description automatically generated

Testing loss

Training loss

**Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated**

Testing accuracy

Training accuracy