人工智慧期末專題

發票辨識

資管三

S0961001 陳常葳

指導老師: 陳巧旻 老師

**目錄**

[**一、** **主題動機** 3](#_Toc138801972)

[**二、** **執行策略** 3](#_Toc138801973)

[**(一)** **以mnist建立辨識模型** 3](#_Toc138801974)

[**(二)** **建立模型** 3](#_Toc138801975)

[**(三)** **蒐集資料與套入模型** 3](#_Toc138801976)

[**三、** **執行過程** 3](#_Toc138801977)

[**(一)** **以mnist建立辨識模型** 3](#_Toc138801978)

[**(二)** **建立模型:自製CNN** 6](#_Toc138801979)

[**(三)** **建立模型: LENET-5模型** 10](#_Toc138801980)

[**(四)** **處理自己蒐集的data** 12](#_Toc138801981)

[**1.** **電子發票** 12](#_Toc138801982)

[2. 傳統發票 15](#_Toc138801983)

[**四、** **結果** 16](#_Toc138801984)

[**(一)** **以自製CNN模型預測第一張發票** 16](#_Toc138801985)

[**(二)** **以自製CNN模型預測第二張發票** 17](#_Toc138801986)

[**(三)** **以LENET-5模型預測第一張發票** 18](#_Toc138801987)

[**(四)** **以LENET-5模型預測第二張發票** 19](#_Toc138801988)

[**五、** **面臨問題** 21](#_Toc138801989)

[**(一)** **圖片尺寸&維度轉換** 21](#_Toc138801990)

[**(二)** **模型建構問題** 21](#_Toc138801991)

[**(三)** **自製圖片放入模型問題** 21](#_Toc138801992)

[**六、** **結論與心得** 21](#_Toc138801993)

[**七、** **參考資料** 22](#_Toc138801994)

1. **主題動機**

由於在APP程式設計課程製作之期末專題為發票載具，欲精進其載具系統之功能，故製作發票數字辨識之模型，來增加該系統功能豐富性。

1. **執行策略**

發票數字辨識之大致步驟分為以下步驟：

* 1. **以mnist建立辨識模型**

先以mnist手寫數字資料集，將數字辨識的模型建立，供日後辨識發票號碼使用。

* 1. **建立模型**

以處裡過後的圖片資料作為辨識模型所使用的資料，用不同方法建立模型。

* 1. **蒐集資料與套入模型**

拍攝手邊發票做為資料集，經過處理後放入模型預測。

1. **執行過程**
2. **以mnist建立辨識模型**

[目的與動機]分好training 與 testing data

[程式碼]

# choose the training and test datasets

train\_data = datasets.MNIST(root='data', train=True,download=True, transform=transform)

test\_data = datasets.MNIST(root='data', train=False,download=True, transform=transform)

# prepare data loaders

train\_loader = torch.utils.data.DataLoader(train\_data, batch\_size=batch\_size,num\_workers=num\_workers)

test\_loader = torch.utils.data.DataLoader(test\_data, batch\_size=batch\_size,num\_workers=num\_workers)

[輸出與結果]



[目的與動機]查看mnist資料及圖片內容

[程式碼]

plt.figure(figsize=(15,15)) #設定圖片呈現大小

for i in range(0,100):

ax=plt.subplot(10,10,1+i)

ax.imshow(x\_train[i]) #加入cmap='gray'可以看黑白圖片

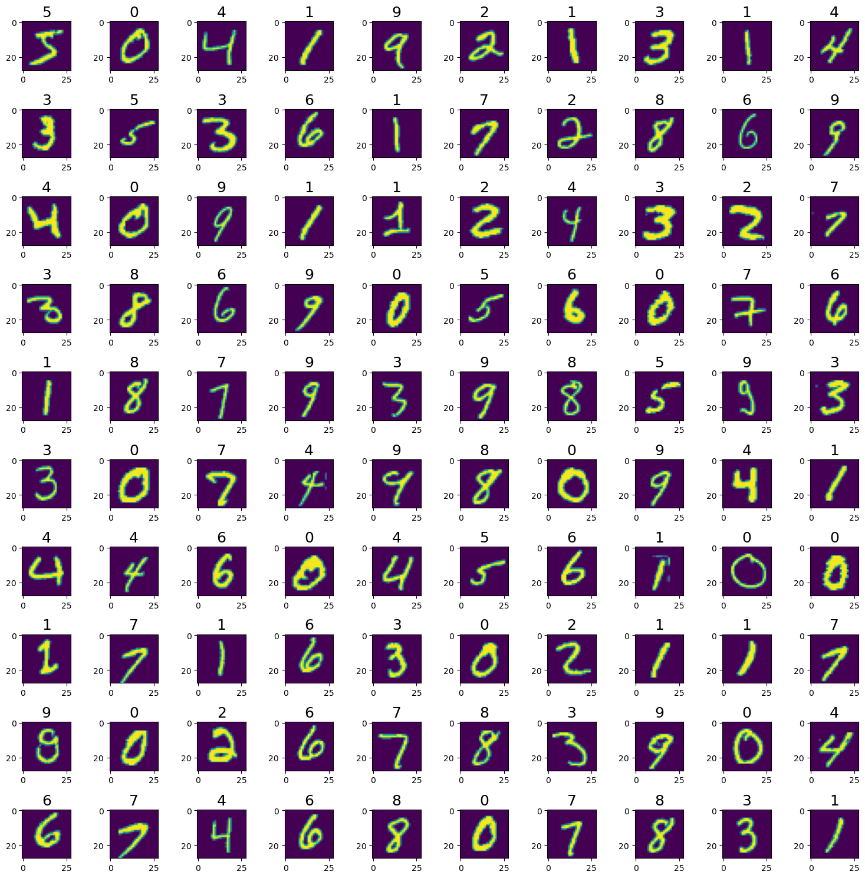
title= str(y\_train[i])

ax.set\_title(title, fontsize=18)

plt.tight\_layout()

plt.show()

[輸出與結果]



[目的與動機]圖片色彩轉換&更改維度

[程式碼]

##將圖片轉成二維的資料

x\_train= x\_train.reshape(60000, 28\*28).astype('float32')

x\_test = x\_test.reshape(10000, 28\*28).astype('float32')

print("training image =", x\_train.shape)

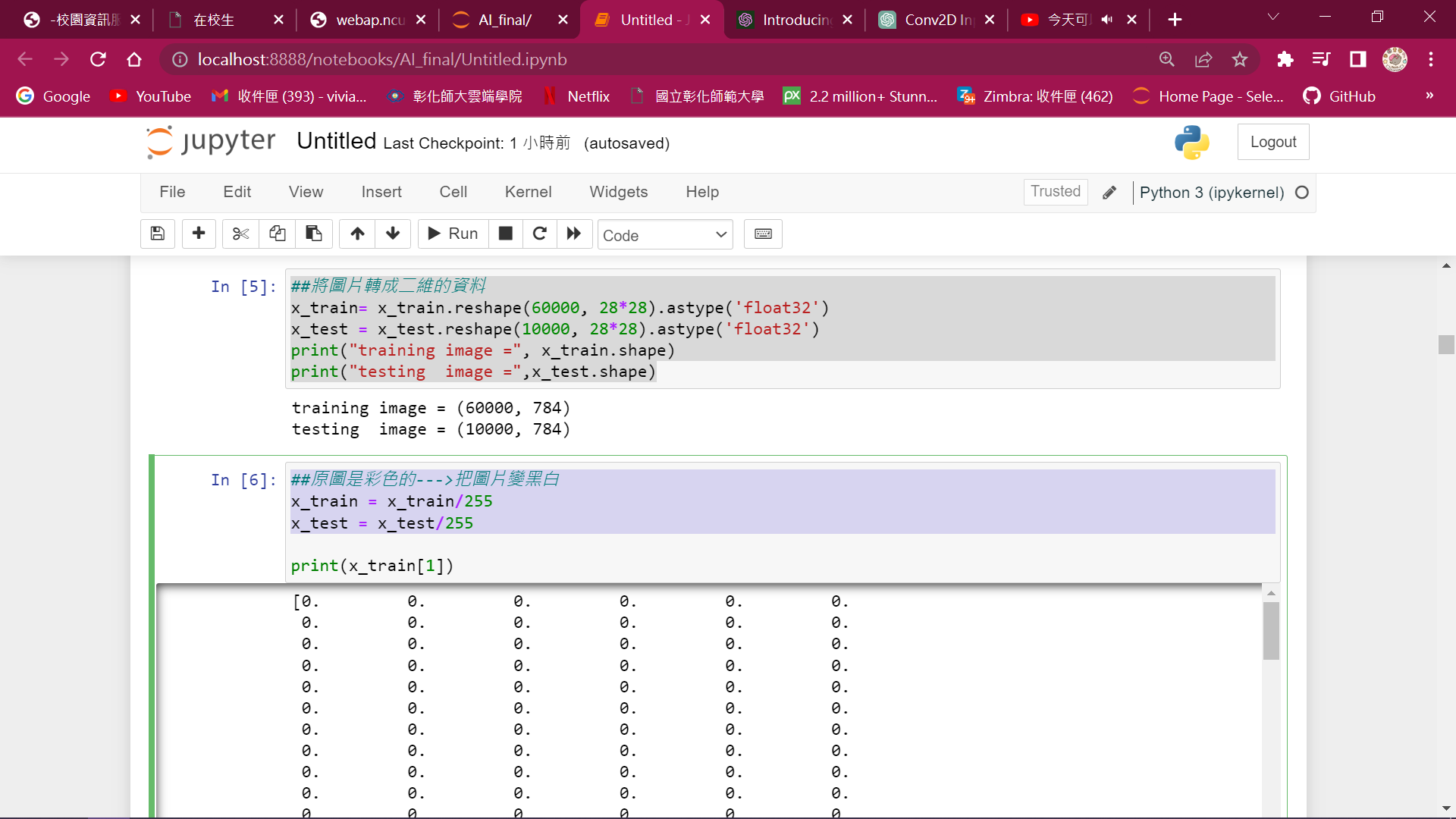
print("testing image =",x\_test.shape)

##原圖是彩色的--->把圖片變黑白

x\_train = x\_train/255

x\_test = x\_test/255

[輸出與結果]將圖片資料從二維(28\*28)改成一維(784)。



1. **建立模型:自製CNN**

[目的與動機]以pytorch建立自製CNN模型

[程式碼]

class Net\_3(nn.Module):

def \_\_init\_\_(self):

super(Net\_3, self).\_\_init\_\_()

self.fc1 = nn.Linear(28 \* 28, 512)

self.fc2 = nn.Linear(512, 512)

self.relu1 = nn.ReLU()

self.fc3=nn.Linear(512,256)

self.relu2 = nn.ReLU()

self.fc4=nn.Linear(256,256)

self.relu3 = nn.ReLU()

self.fc5=nn.Linear(256,128)

self.relu4 = nn.ReLU()

self.fc6=nn.Linear(128,128)

self.relu5=nn.ReLU()

self.fc7 = nn.Linear(128, 10)

self.dropout = nn.Dropout(0.2)

def forward(self, x):

# flatten image input

x = x.view(-1, 28 \* 28)

# add hidden layer, with relu activation function

x = self.fc1(x)

x = self.fc2(x)

x = self.relu1(x)

x = self.fc3(x)

x = self.relu2(x)

x = self.fc4(x)

x = self.relu3(x)

x = self.fc5(x)

x = self.relu4(x)

x = self.fc6(x)

x = self.relu5(x)

x = self.fc7(x)

return x

# initialize the NN

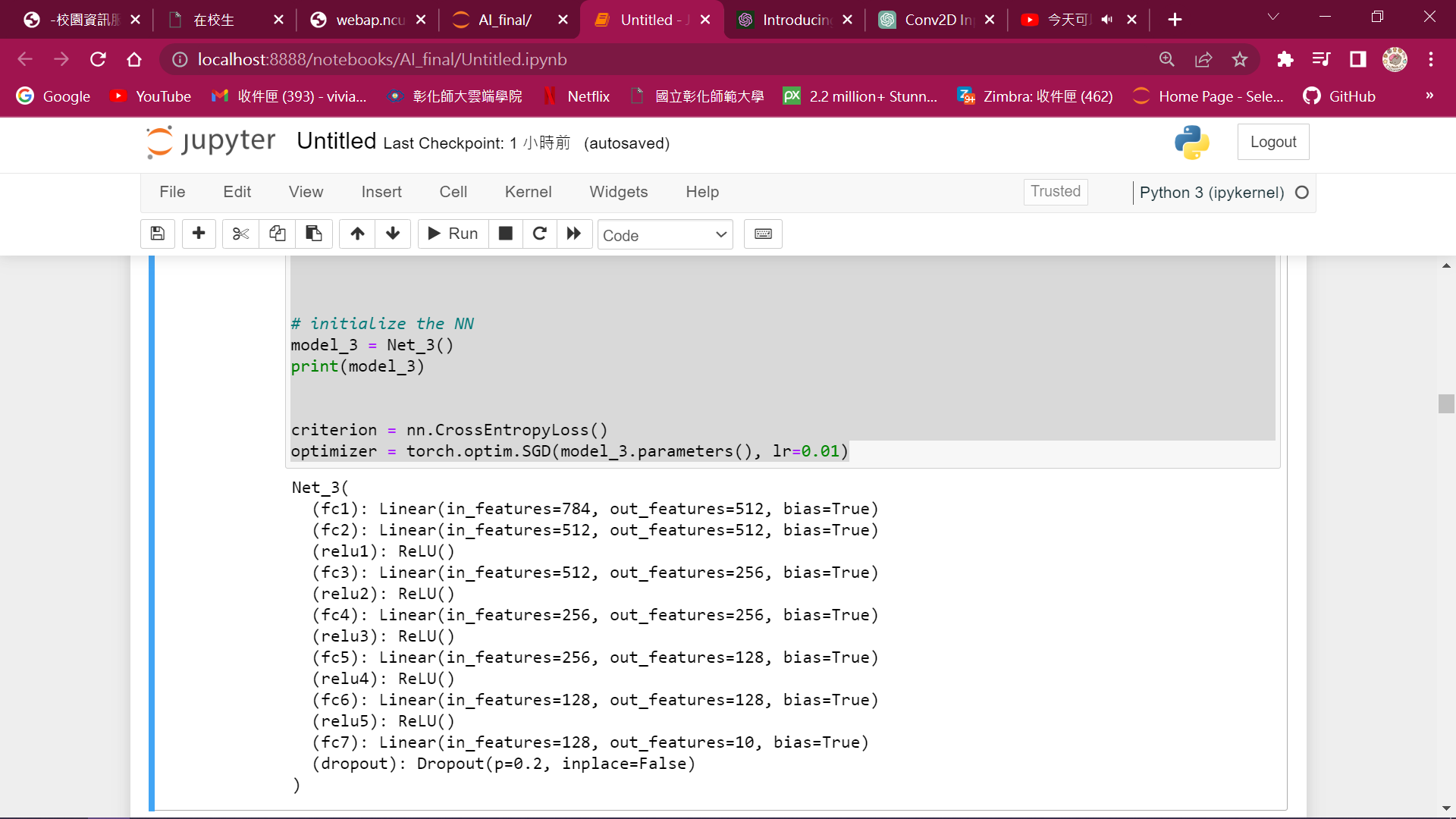
model\_3 = Net\_3()

print(model\_3)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model\_3.parameters(), lr=0.01)

[輸出與結果]逐漸減少input dimension，並在每一層加上relu() activation function



[目的與動機]查看以pytorch自製CNN模型之training loss

[程式碼]

n\_epochs = 10

model\_3.train() # prep model for training

for epoch in range(n\_epochs):

train\_loss = 0.0

for data, target in train\_loader:

# clear the gradients of all optimized variables

optimizer.zero\_grad()

# forward pass: compute predicted outputs by passing inputs to the model

output = model\_3(data)

# calculate the loss

loss = criterion(output, target)

# backward pass: compute gradient of the loss with respect to model parameters

loss.backward()

# perform a single optimization step (parameter update)

optimizer.step()

# update running training loss

train\_loss += loss.item()\*data.size(0)

train\_loss = train\_loss/len(train\_loader.dataset)

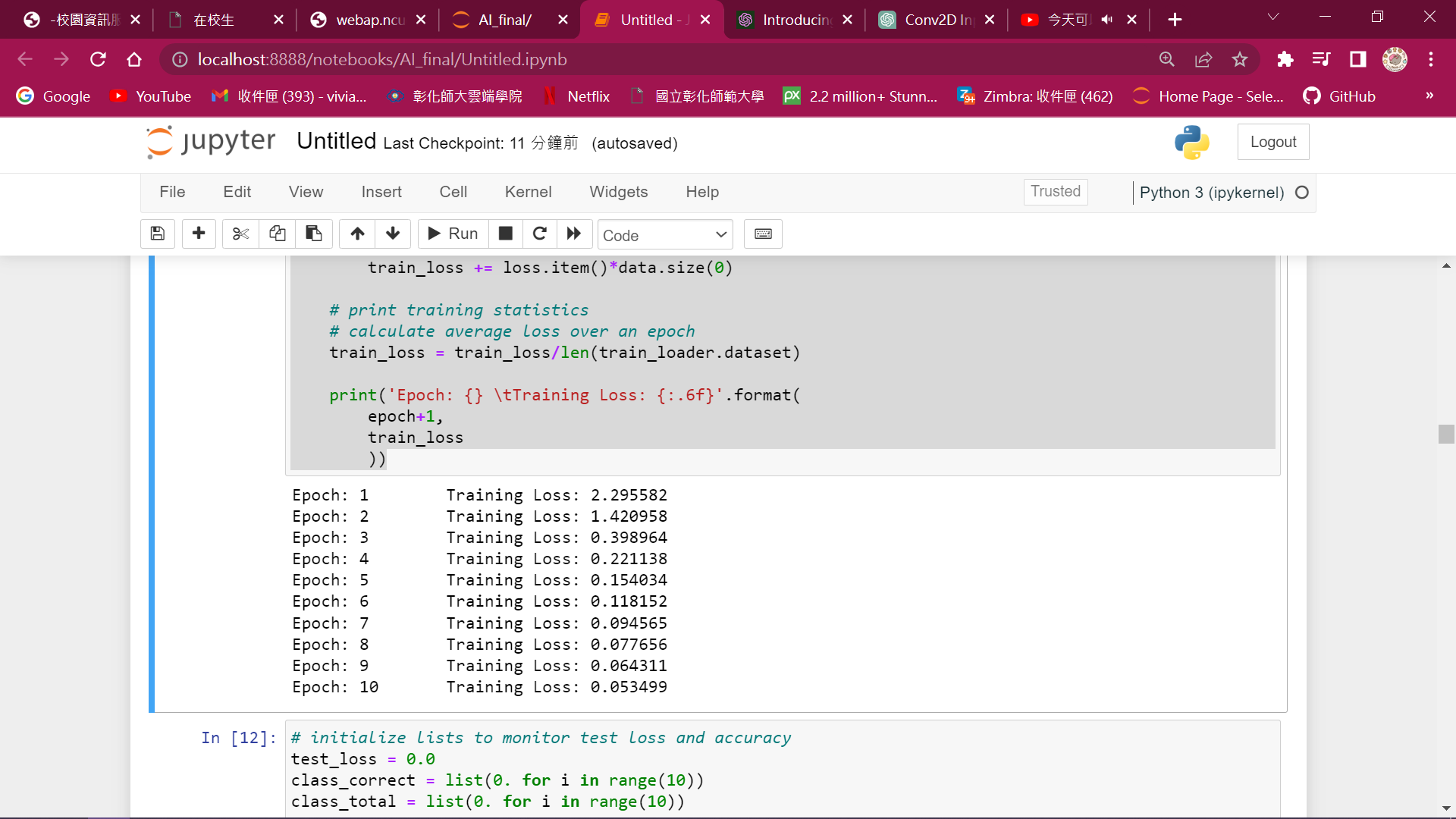
print('Epoch: {} \tTraining Loss: {:.6f}'.format(

epoch+1,

train\_loss

))

[輸出與結果]



[目的與動機]查看以pytorch自製CNN模型之accuracy

[程式碼]

test\_loss = 0.0

class\_correct = list(0. for i in range(10))

class\_total = list(0. for i in range(10))

model\_3.eval() # prep model for \*evaluation\*

for data, target in test\_loader:

# forward pass: compute predicted outputs by passing inputs to the model

output = model\_3(data)

# calculate the loss

loss = criterion(output, target)

# update test loss

test\_loss += loss.item()\*data.size(0)

# convert output probabilities to predicted class

\_, pred = torch.max(output, 1)

# compare predictions to true label

correct = np.squeeze(pred.eq(target.data.view\_as(pred)))

# calculate test accuracy for each object class

for i in range(10):

label = target.data[i]

class\_correct[label] += correct[i].item()

class\_total[label] += 1

# calculate and print avg test loss

test\_loss = test\_loss/len(test\_loader.dataset)

print('Test Loss: {:.6f}\n'.format(test\_loss))

for i in range(10):

if class\_total[i] > 0:

print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (

str(i), 100 \* class\_correct[i] / class\_total[i],

np.sum(class\_correct[i]), np.sum(class\_total[i])))

else:

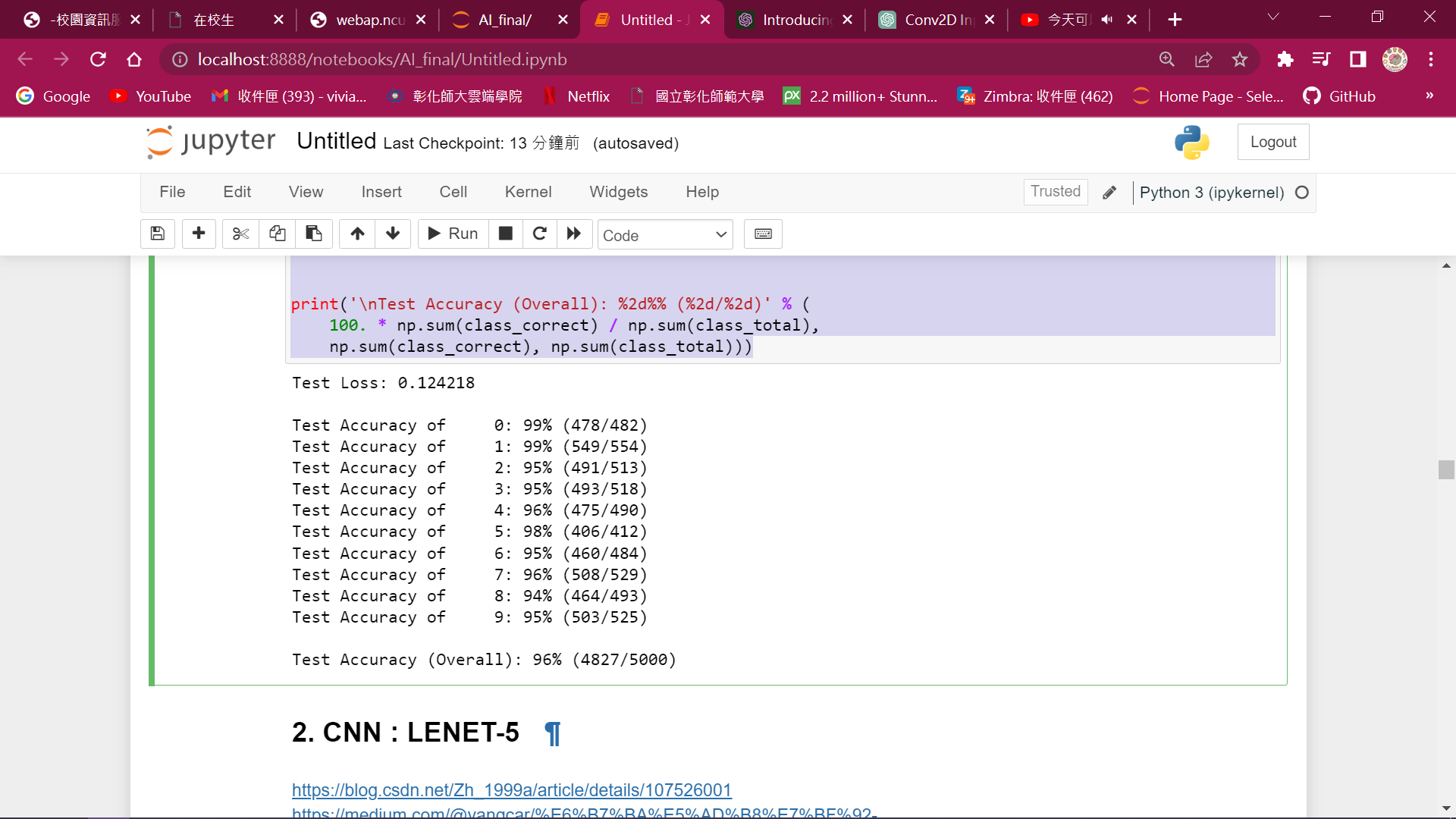
print('Test Accuracy of %5s: N/A (no training examples)' % (str(i)))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (

100. \* np.sum(class\_correct) / np.sum(class\_total),

np.sum(class\_correct), np.sum(class\_total)))

[輸出與結果]平均準確率為96%，而test loss則為0.124



1. **建立模型: LENET-5模型**

[目的與動機]以pytorch建立LENET-5模型

[程式碼]

num\_classes=10

class ConvNeuralNet(nn.Module):

def \_\_init\_\_(self, num\_classes):

super(ConvNeuralNet, self).\_\_init\_\_()

self.layer1 = nn.Sequential(

nn.Conv2d(1, 6, kernel\_size=5, stride=1, padding=0),

nn.BatchNorm2d(6),

nn.ReLU(),

nn.MaxPool2d(kernel\_size = 2, stride = 2))

self.layer2 = nn.Sequential(

nn.Conv2d(6, 16, kernel\_size=5, stride=1, padding=0),

nn.BatchNorm2d(16),

nn.ReLU(),

nn.MaxPool2d(kernel\_size = 2, stride = 2))

self.fc = nn.Linear(400, 120)

self.relu = nn.ReLU()

self.fc1 = nn.Linear(120, 84)

self.relu1 = nn.ReLU()

self.fc2 = nn.Linear(84, num\_classes)

def forward(self, x):

out = self.layer1(x)

out = self.layer2(out)

out = out.reshape(out.size(0), -1)

out = self.fc(out)

out = self.relu(out)

out = self.fc1(out)

out = self.relu1(out)

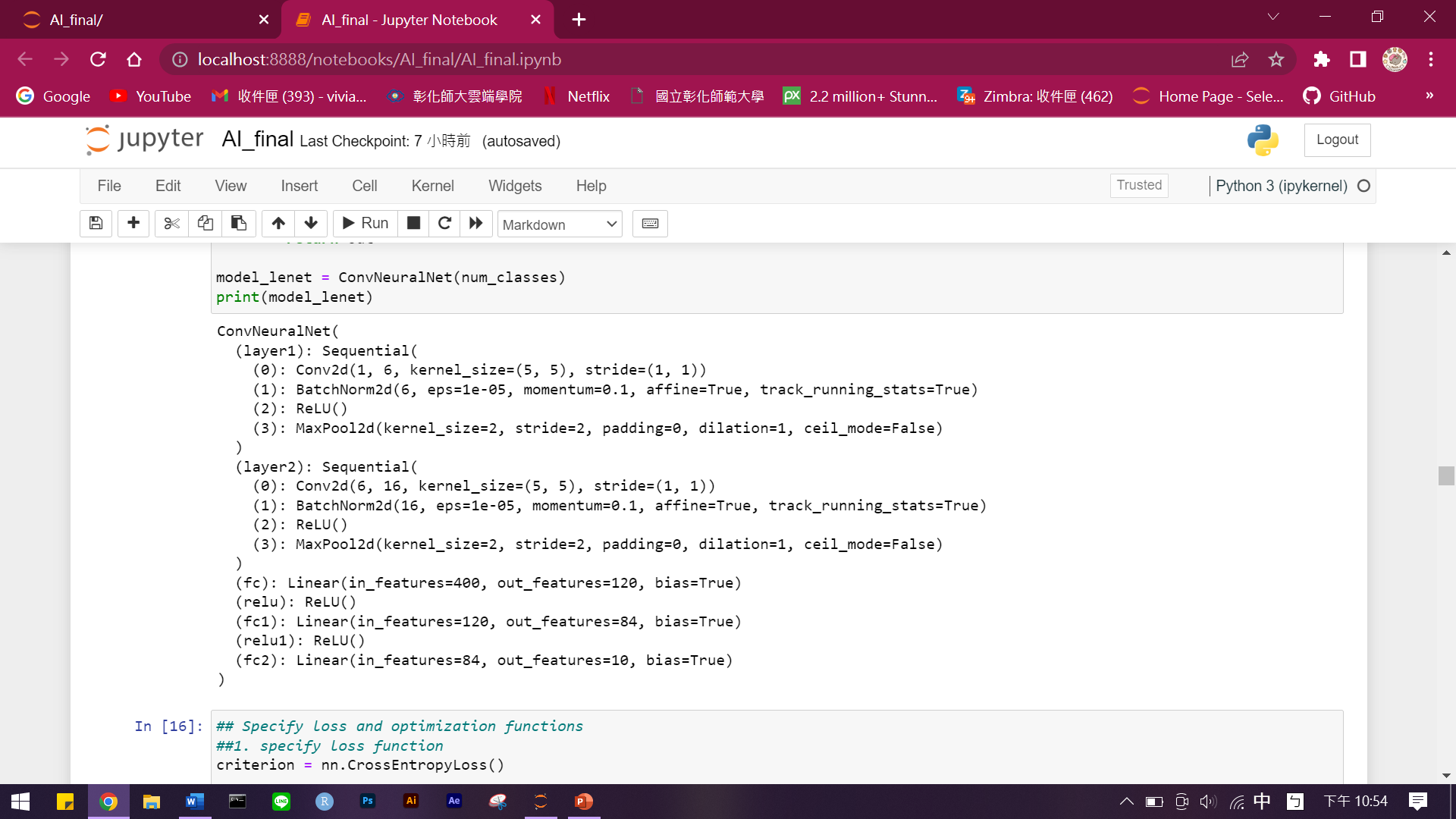
out = self.fc2(out)

return out

model\_lenet = ConvNeuralNet(num\_classes)

print(model\_lenet)

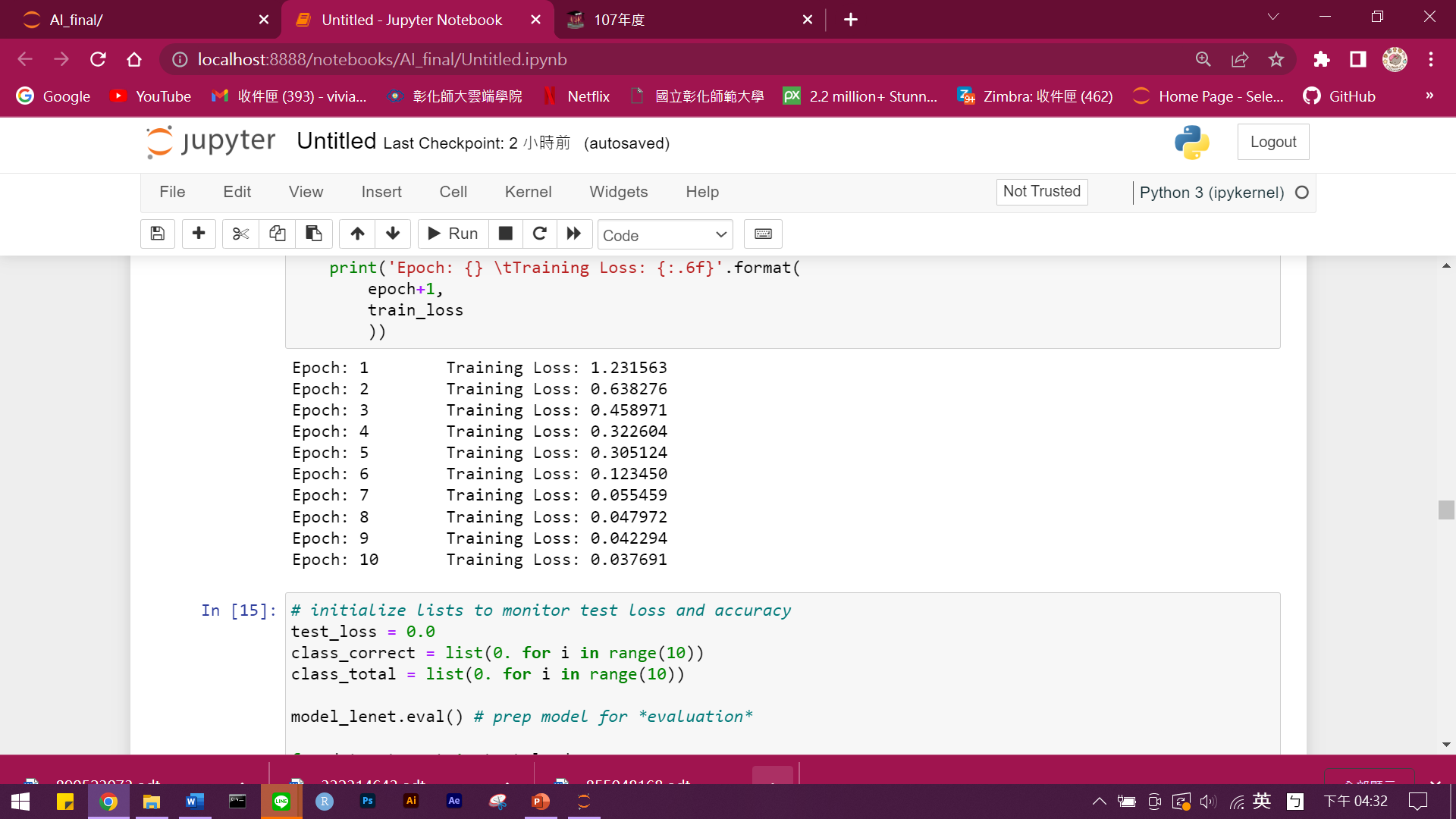
[輸出與結果]



[目的與動機]查看LENET-5模型之testing loss 與 accuracy

[程式碼] 與自製CNN模型做法相同

[輸出與結果]可得知lenet-5的testing data loss 為0.035，準確率為98%，較自製CNN高一些。





1. **處理自己蒐集的data**
2. **電子發票**

[目的與動機]開啟欲處理相片並適當裁切

[程式碼]

os.chdir('D:\invoice')#change directory

rawimg0 = cv2.imread("S\_\_28573882.jpg")

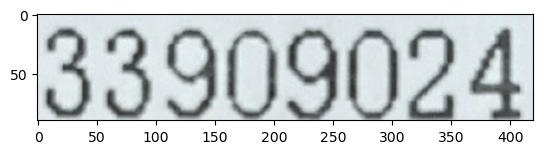
# 對照片進行定位後裁切

cropped = rawimg0 [460:550,430:860]

# 查看裁切後的照片

plt.imshow(cropped)

[輸出與結果]



[目的與動機]將資料灰階與二值化

[程式碼]

# 圖片灰階

grayscaleimg = cv2.cvtColor(cropped,cv2.COLOR\_BGR2GRAY)

# 圖片二值化

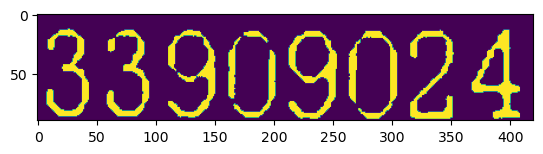
ret, binary = cv2.threshold(grayscaleimg, 130, 255, cv2.THRESH\_BINARY)

plt.imshow(binary,cmap='Greys',interpolation='None')

rawimg = binary - binary[0,1] #圖的最低就會變成0 & 黑底白字

plt.imshow(rawimg)

[輸出與結果]



[目的與動機]準確描出數字確切的範圍

[程式碼]

# counting non-zero value by row , axis y

row\_nz = []

for row in rawimg.tolist():

row\_nz.append(len(row) - row.count(0))

plt.plot(row\_nz)

idx=np.array(row\_nz)>(max(row\_nz)/4) #截出上下的範圍

np.where(idx==1)[0][0],np.where(idx==1)[0][-1]

up\_y=np.where(idx==1)[0][-1] #上界

down\_y=np.where(idx==1)[0][0] #下界

plt.imshow(rawimg)

[輸出與結果]



[目的與動機]切割8個發票數字成8個圖片

[程式碼]

# counting non-zero value by column, x axis

col\_nz = []

for col in rawimg1.T.tolist():

col\_nz.append(len(col) - col.count(0))

plt.plot(col\_nz)

idy=np.not\_equal(col\_nz,0)

record\_y=[] #如果有八個數字，裡面應該要有九個格子(一開始找出七個，前後插入變九個)

for i in range(0,(len(np.where(idy==1)[0])-1)):

# 如果下一個數是0就略過，直到找到下一個數不是0的位置

if(np.where(idy==1)[0][i+1]-np.where(idy==1)[0][i]==1):

pass

else:

record\_y.append(np.where(idy==1)[0][i])

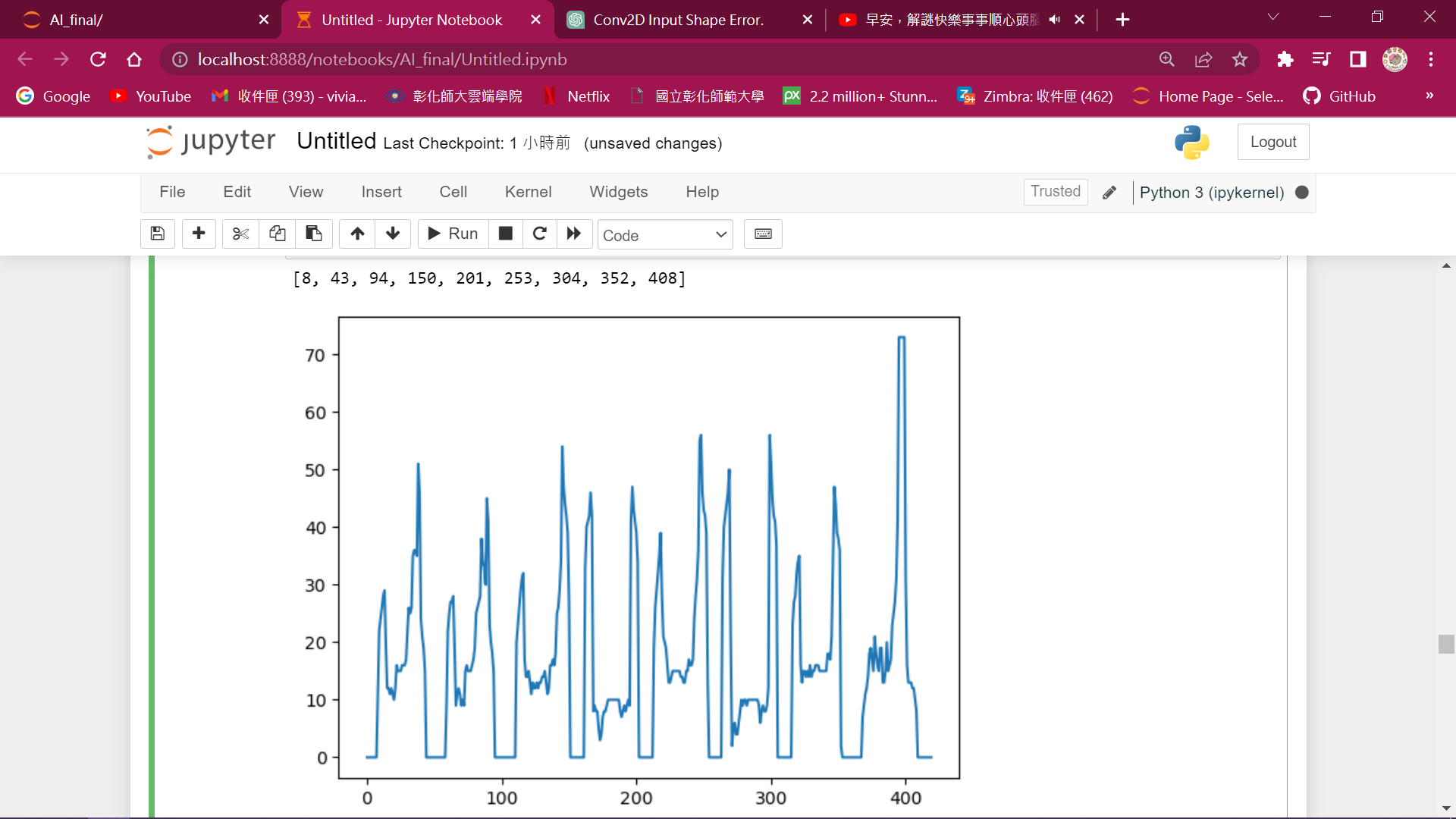
# 插入第一個非0位置跟最後一個非0的位置

record\_y.insert(0,np.where(idy==1)[0][0])

record\_y.append(np.where(idy==1)[0][-1])

print(record\_y)

[輸出與結果]



[目的與動機]將數字存成圖檔

[程式碼]

for i in range(0,len(record\_y)-1):

a=binary[down\_y:up\_y,record\_y[i]+5:record\_y[i+1]+5]

a=cv2.resize(a, (28, 28), interpolation=cv2.INTER\_CUBIC)

a = cv2.bitwise\_not(a)

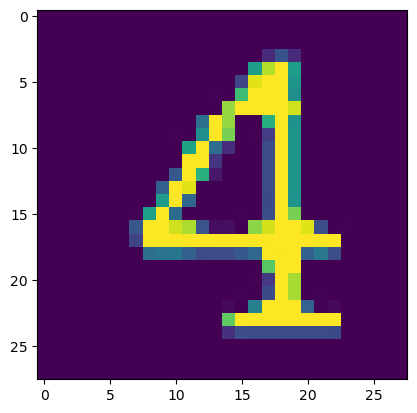
a = cv2.copyMakeBorder(a,5,5,5,5,cv2.BORDER\_CONSTANT,value=0)#加上邊框，不要讓數字太靠進圖片邊緣

img\_name='%s-%s.png'%(1,i+1)

cv2.imwrite(img\_name,a)

plt.imshow(a)

[輸出與結果]可以看到數字置中於圖像中，並且在邊框保留空白。



1. **傳統發票**

傳統發票處理方式與電子發票相同，但由於傳統發票雜訊較多，因此加上清除雜訊之程式碼。

[目的與動機]清除雜訊並輸出結果

[程式碼]

# 檢查數字

rm\_id=[]

if len(record\_y)>9:

for j in range(0,len(record\_y)-1):

temp=np.array(col\_nz[record\_y[j]:record\_y[j+1]])

#如果只是雜訊，就刪掉

if sum(temp>(max(col\_nz)/4))==0:

rm\_id.append(record\_y[j+1])

for x in rm\_id:

record\_y.remove(x)

for i in range(0,len(record\_y)-1):

b=binary2[down\_y:up\_y,record\_y[i]+5:record\_y[i+1]+5]

b=cv2.resize(b, (28, 28), interpolation=cv2.INTER\_CUBIC)

b = cv2.bitwise\_not(b)

b = cv2.copyMakeBorder(b,4,4,4,4,cv2.BORDER\_CONSTANT,value=0)#加上邊框，不要讓數字太靠進圖片邊緣

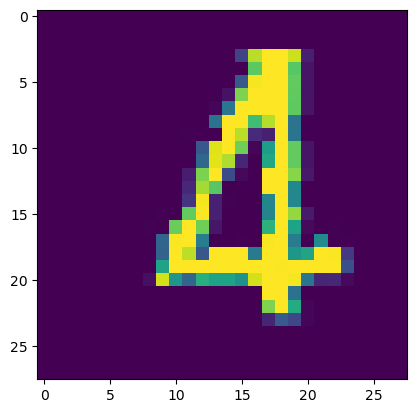
b=cv2.resize(b, (28, 28), interpolation=cv2.INTER\_CUBIC)

img\_name='%s-%s.png'%(2,i+1)

cv2.imwrite(img\_name,b)

plt.imshow(b)

[輸出與解果]由於傳統發票本身數字顏色非黑色，因此較為模糊，加上雜訊清除也可以增加數字的清晰程度。



1. **結果**
2. **以自製CNN模型預測第一張發票**

[目的與動機]將處理好的圖片資料放入自製模型中預測

[程式碼]

model\_3.eval()

# define圖像的處裡轉換

transform = transforms.Compose([

transforms.Grayscale(), # 轉灰階

transforms.ToTensor(), # 轉張量

transforms.Normalize((0.1307,), (0.3081,)) # 標準化

])

for i in range(0,8):

img2= Image.open(r'D:\invoice\1-%s.png'%(str(i+1)))

plt.show(img2)

# 用剛剛定義的預處理圖像定義

img2 = transform(img2)

# 轉一維向量

image\_vector = img2.view(1, -1)

# 模型推理

output = model\_3(image\_vector)

predicted\_label = torch.argmax(output, dim=1)

# 看結果

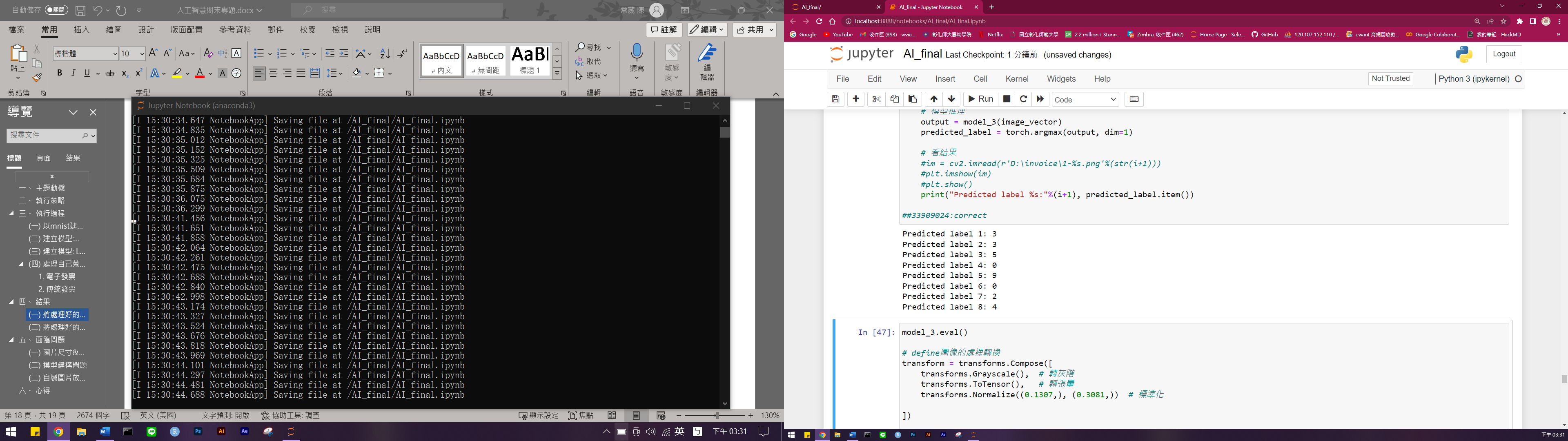
im = cv2.imread(r'D:\invoice\1-%s.png'%(str(i+1)))

plt.imshow(im)

plt.show()

print("Predicted label %s:"%(i+1), predicted\_label.item())

[輸出與結果]在電子發票預測上自製模型的判斷大致上正確，只有一個數字錯誤



1. **以自製CNN模型預測第二張發票**

[目的與動機]將處理好的圖片資料放入自製模型中預測

[程式碼]

model\_3.eval()

# define圖像的處裡轉換

transform = transforms.Compose([

transforms.Grayscale(), # 轉灰階

transforms.ToTensor(), # 轉張量

transforms.Normalize((0.1307,), (0.3081,)) # 標準化

])

for i in range(0,8):

img2= Image.open(r'D:\invoice\2-%s.png'%(str(i+1)))

plt.show(img2)

# 用剛剛定義的預處理圖像定義

img2 = transform(img2)

# 轉一維向量

image\_vector = img2.view(1, -1)

# 模型推理

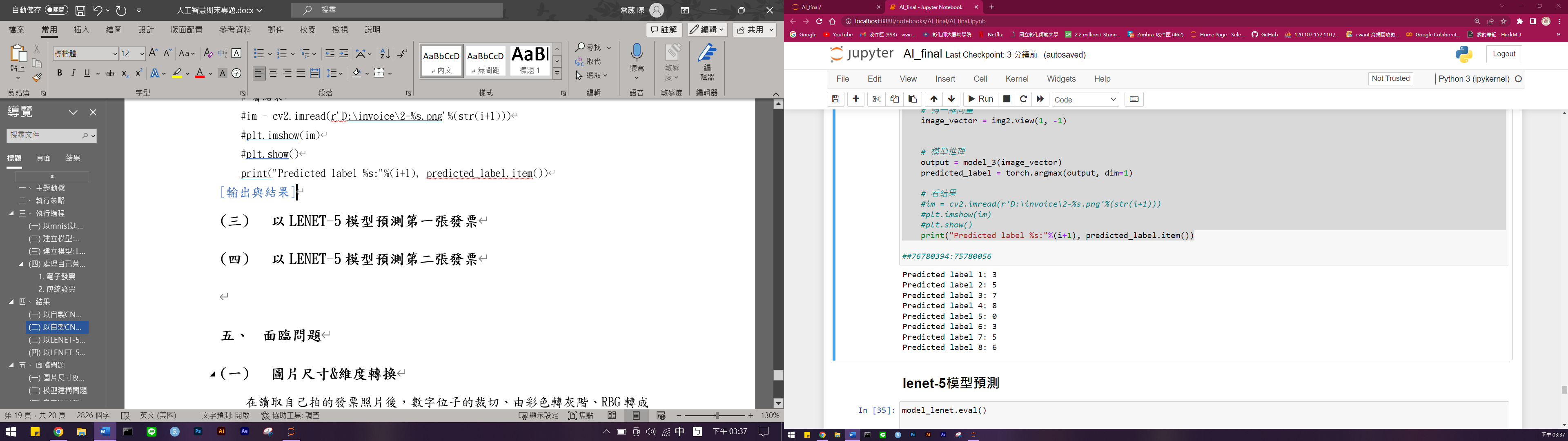
output = model\_3(image\_vector)

predicted\_label = torch.argmax(output, dim=1)

# 看結果

print("Predicted label %s:"%(i+1), predicted\_label.item())

[輸出與結果]可以得知傳統發票在自製模型的預測上較為不準確，約一半為錯誤判斷。



1. **以LENET-5模型預測第一張發票**

[目的與動機]由於傳統發票在自製模型上預測較為不準確，因此用lenet-5模型來預測做觀察兩張發票的情況

[程式碼]

model\_lenet.eval()

# define圖像的處裡轉換

transform = transforms.Compose([

#transforms.Grayscale(), # 轉灰階

transforms.ToTensor(), # 轉張量

transforms.Resize((32,32)),

transforms.Normalize((0.1307,), (0.3081,)) # 標準化

])

for i in range(0,8):

img= Image.open(r'D:\invoice\1-%s.png'%(str(i+1)))

# 用剛剛定義的預處理圖像定義

img = transform(img)

img2 = torch.unsqueeze(img, dim=0)

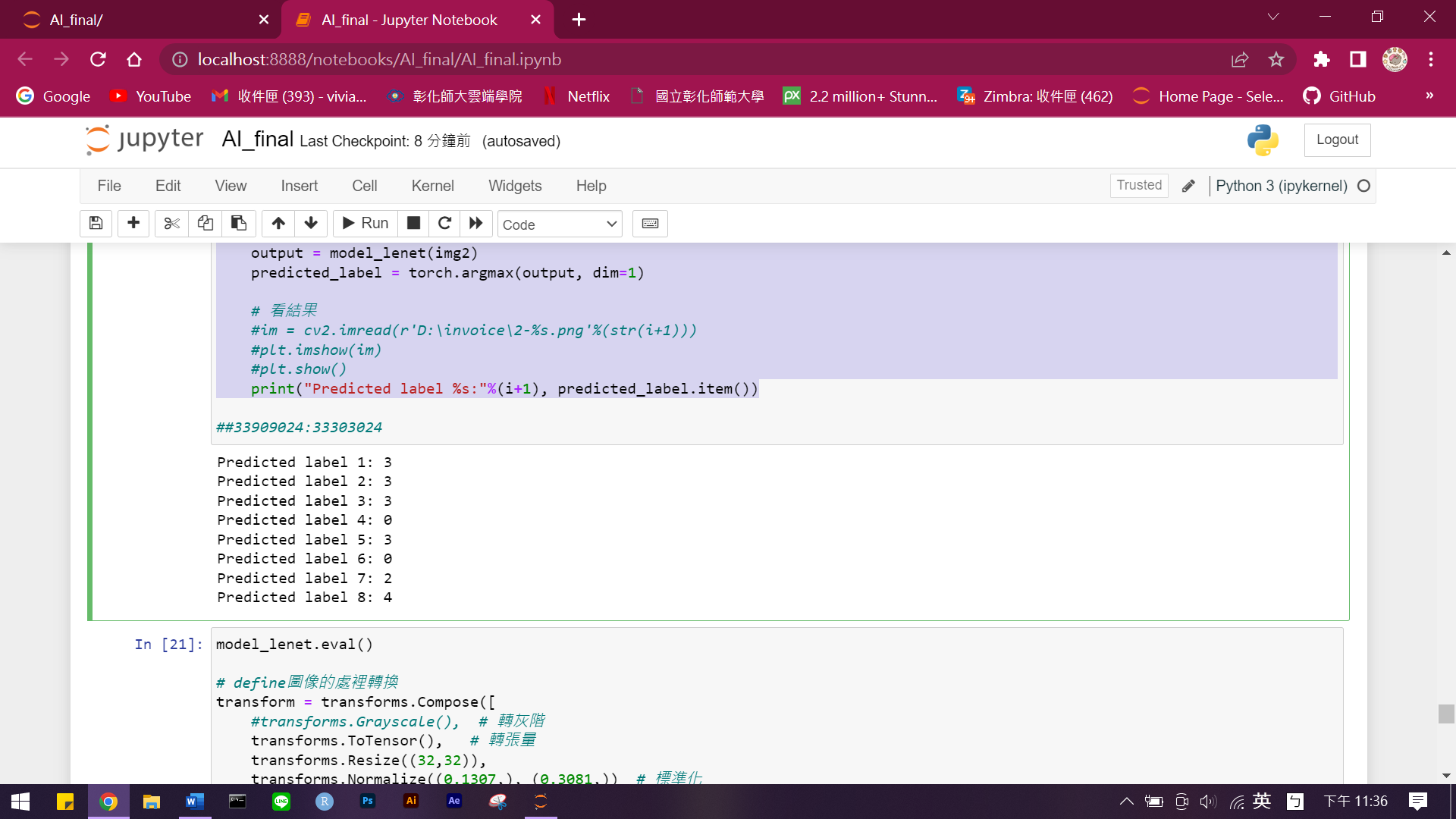
output = model\_lenet(img2)

predicted\_label = torch.argmax(output, dim=1)

# 看結果

print("Predicted label %s:"%(i+1), predicted\_label.item())

[輸出與結果]可以得知並不是每個數字都能判斷正確，原本號碼為33909024，而數字9在lenet-5的模型被判斷錯誤。



1. **以LENET-5模型預測第二張發票**

[目的與動機]由於傳統發票在自製模型上預測較為不準確，因此用lenet-5模型來預測做觀察兩張發票的情況

[程式碼]

model\_lenet.eval()

# define圖像的處裡轉換

transform = transforms.Compose([

#transforms.Grayscale(), # 轉灰階

transforms.ToTensor(), # 轉張量

transforms.Resize((32,32)),

transforms.Normalize((0.1307,), (0.3081,)) # 標準化

])

for i in range(0,8):

img= Image.open(r'D:\invoice\2-%s.png'%(str(i+1)))

# 用剛剛定義的預處理圖像定義

img = transform(img)

img2 = torch.unsqueeze(img, dim=0)

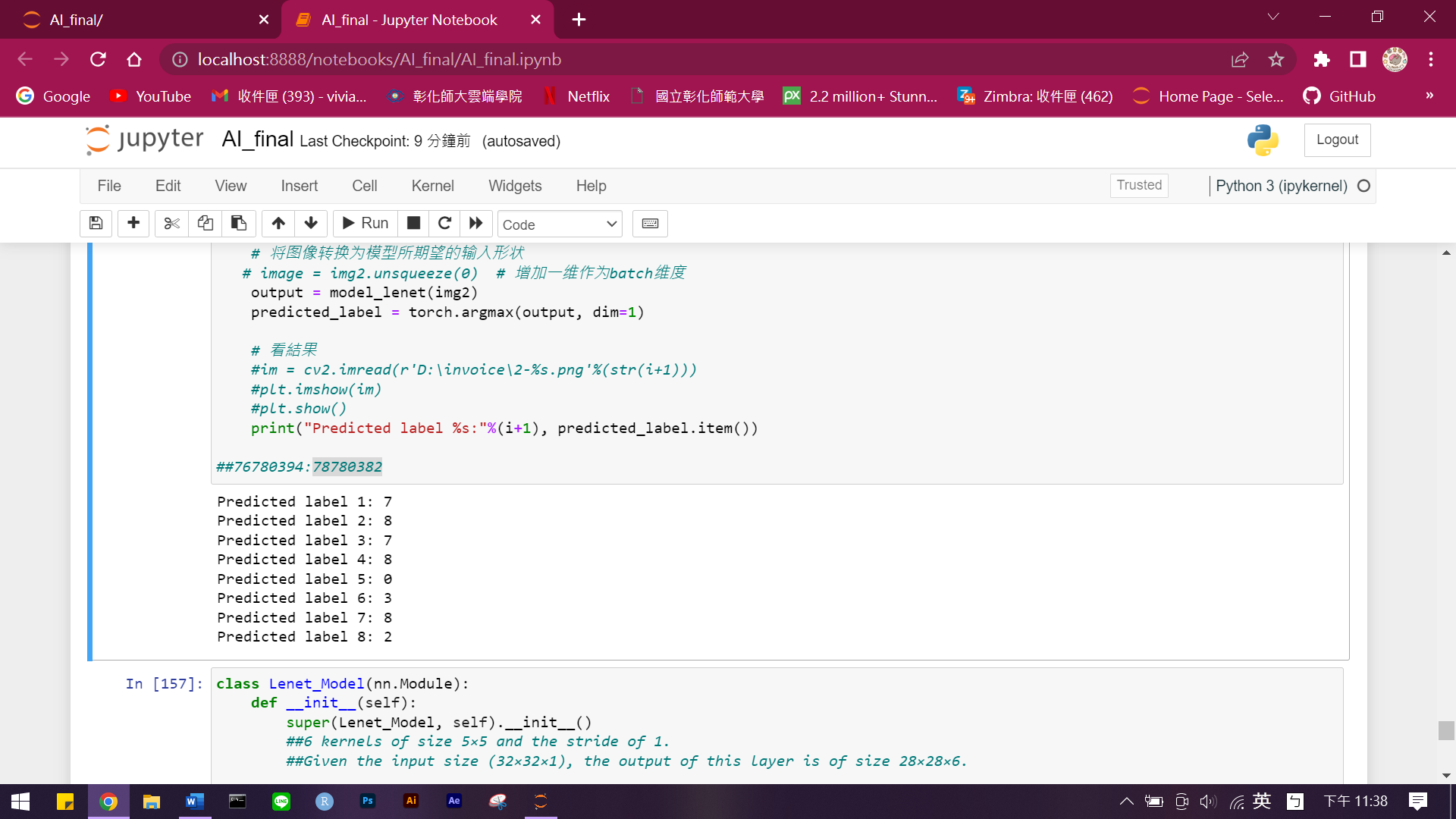
output = model\_lenet(img2)

predicted\_label = torch.argmax(output, dim=1)

# 看結果

print("Predicted label %s:"%(i+1), predicted\_label.item())

[輸出與結果]可以得知並不是每個數字都能判斷正確，原本號碼為76780394，而數字6、9、4在lenet-5的模型預測中被判斷錯誤。



1. **面臨問題**
2. **圖片尺寸&維度轉換**

在讀取自己拍的發票照片後，數字位子的裁切、由彩色轉灰階、RBG轉成一維array花不少時間上網查一些資料。最終數字位子裁切以cv2.copyMakeBorder(cv2.BORDER\_CONSTANT,value=0)來解決，讓數字可以集中在圖片中央而不會因太靠進邊緣預測失敗。

1. **模型建構問題**

原本打算用tensorflow做模型，最後因為還是比較熟pytorch所以又重新將lenet-5的模型以pytorch寫了一次。

1. **自製圖片放入模型問題**

由於模型與圖片格式的限制，也花了一些時間把處理好的自製資料丟進模型內進行預測，至於lenet-5的模型預測則是卡在維度array轉換與資料扁平化的問題，最終也得以在batch\_size的部份解決。

1. **結論與心得**

在兩個模型中可以看到，雖然準確率與梯度下降後的結果都很不錯，單在實際拍攝的照片資料上仍然不一定可以準確判斷出數字，而圖片拍攝手法以及真實資料處理可能為主要影響因素。

由於在準確率上兩個模型差不多，因此將兩個模型都放入預測，另外加上當時在預測前的發票影像處理上前期處理得比較粗糙，造成在報告時的預測結果較差，維度轉換問題也一直無法解決，因此在這兩點花上較多時間。而在後續繼續完成專題時這些問題雖然依舊花了點時間查詢資料，但最終也都得以順利解決。

另外雖然上學期已經修過機器學習的課程，不過因為我以前沒有碰過python語言，在這學期的摸索上也花了一些時間，不過整體而言是有收獲也滿有趣的。

1. **參考資料**
2. [https://ithelp.ithome.com.tw/articles/10193469](https://ithelp.ithome.com.tw/articles/10193469%20%20)
3. <https://www.wpgdadatong.com/blog/detail/46474>
4. [https://github.com/gradient-ai/LeNet5-Tutorial/blob/main/LeNet5.ipynb](https://github.com/gradient-ai/LeNet5-Tutorial/blob/main/LeNet5.ipynb%20%20)
5. [https://blog.paperspace.com/writing-lenet5-from-scratch-in-python/](https://blog.paperspace.com/writing-lenet5-from-scratch-in-python/%20%20)
6. [https://blog.csdn.net/Zh\_1999a/article/details/107526001](https://blog.csdn.net/Zh_1999a/article/details/107526001%20%20)
7. [https://medium.com/@yangcar/%E6%B7%BA%E5%AD%B8%E7%BF%92-%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-ebc673ec97f7](https://medium.com/@yangcar/%E6%B7%BA%E5%AD%B8%E7%BF%92-%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-ebc673ec97f7%20%20)
8. [https://github.com/ChawDoe/LeNet5-MNIST-PyTorch/blob/master/train.py](https://github.com/ChawDoe/LeNet5-MNIST-PyTorch/blob/master/train.py%20%20)
9. <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html>
10. <https://finnysteps.blogspot.com/2019/02/python.html>
11. <https://chih-sheng-huang821.medium.com/%E6%A9%9F%E5%99%A8-%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-%E5%9F%BA%E7%A4%8E%E4%BB%8B%E7%B4%B9-%E6%90%8D%E5%A4%B1%E5%87%BD%E6%95%B8-loss-function-2dcac5ebb6cb>