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

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。

二、 執行策略

發票數字辨識之大致步驟分為以下步驟：

(一) 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。

(二) 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。

(三) 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。

三、 執行過程

(一) 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］

```
training image = (60000, 28, 28)
training label = (60000,)
testing image = (10000, 28, 28)
testing label = (10000,)
```

<--->

training 資料有60000筆，testing有10000筆

X是圖片，y是數字標籤

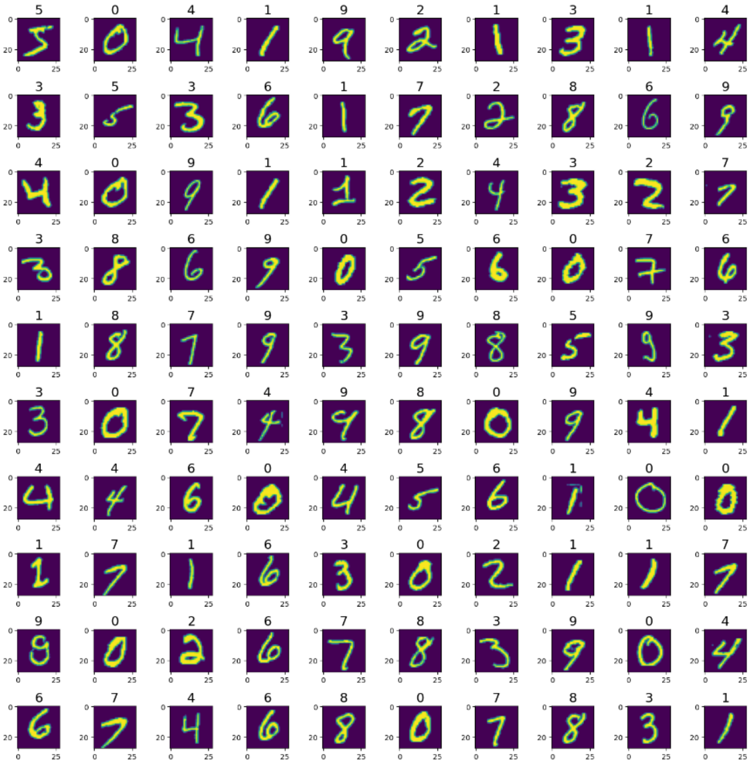
<--->

［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)。

```
training image = (60000, 784)
testing  image = (10000, 784)
```

(二)　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

```
Net_3(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (relu1): ReLU()
  (fc3): Linear(in_features=512, out_features=256, bias=True)
  (relu2): ReLU()
  (fc4): Linear(in_features=256, out_features=256, bias=True)
  (relu3): ReLU()
  (fc5): Linear(in_features=256, out_features=128, bias=True)
  (relu4): ReLU()
  (fc6): Linear(in_features=128, out_features=128, bias=True)
  (relu5): ReLU()
  (fc7): Linear(in_features=128, out_features=10, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

[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]

```
Epoch: 1 	Training Loss: 2.295582
Epoch: 2 	Training Loss: 1.420958
Epoch: 3 	Training Loss: 0.398964
Epoch: 4 	Training Loss: 0.221138
Epoch: 5 	Training Loss: 0.154034
Epoch: 6 	Training Loss: 0.118152
Epoch: 7 	Training Loss: 0.094565
Epoch: 8 	Training Loss: 0.077656
Epoch: 9 	Training Loss: 0.064311
Epoch: 10 	Training Loss: 0.053499
```

[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

```
Test Loss: 0.124218

Test Accuracy of     0: 99% (478/482)
Test Accuracy of     1: 99% (549/554)
Test Accuracy of     2: 95% (491/513)
Test Accuracy of     3: 95% (493/518)
Test Accuracy of     4: 96% (475/490)
Test Accuracy of     5: 98% (406/412)
Test Accuracy of     6: 95% (460/484)
Test Accuracy of     7: 96% (508/529)
Test Accuracy of     8: 94% (464/493)
Test Accuracy of     9: 95% (503/525)

Test Accuracy (Overall): 96% (4827/5000)
```

（三） 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]

```
ConvNeuralNet(
  (layer1): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Linear(in_features=400, out_features=120, bias=True)
  (relu): ReLU()
  (fc1): Linear(in_features=120, out_features=84, bias=True)
  (relu1): ReLU()
  (fc2): Linear(in_features=84, out_features=10, bias=True)
)
```

[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。

```
Epoch: 1        Training Loss: 1.231563
Epoch: 2        Training Loss: 0.638276
Epoch: 3        Training Loss: 0.458971
Epoch: 4        Training Loss: 0.322604
Epoch: 5        Training Loss: 0.305124
Epoch: 6        Training Loss: 0.123450
Epoch: 7        Training Loss: 0.055459
Epoch: 8        Training Loss: 0.047972
Epoch: 9        Training Loss: 0.042294
Epoch: 10       Training Loss: 0.037691
```

```
Test Loss: 0.034972

Test Accuracy of     0: 100% (152/152)
Test Accuracy of     1: 100% (179/179)
Test Accuracy of     2: 97% (154/158)
Test Accuracy of     3: 99% (146/147)
Test Accuracy of     4: 100% (154/154)
Test Accuracy of     5: 97% (136/139)
Test Accuracy of     6: 99% (159/160)
Test Accuracy of     7: 98% (161/164)
Test Accuracy of     8: 98% (142/144)
Test Accuracy of     9: 95% (166/173)

Test Accuracy (Overall): 98% (1549/1570)
```

（四） 處理自己蒐集的 data

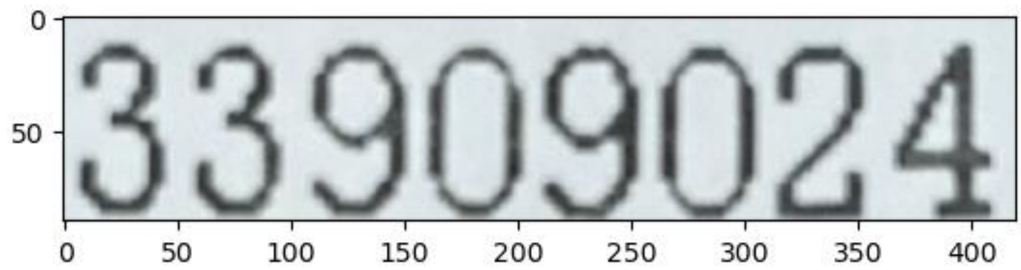
1. 電子發票

［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］

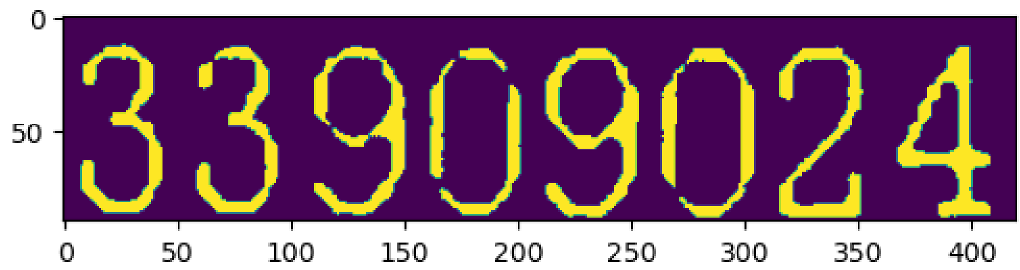


［目的與動機］ 將資料灰階與二值化

［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］



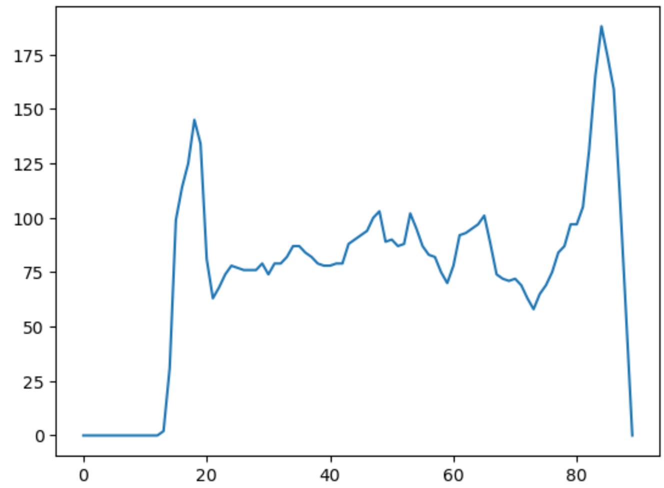
［目的與動機］ 準確描出數字確切的範圍

[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

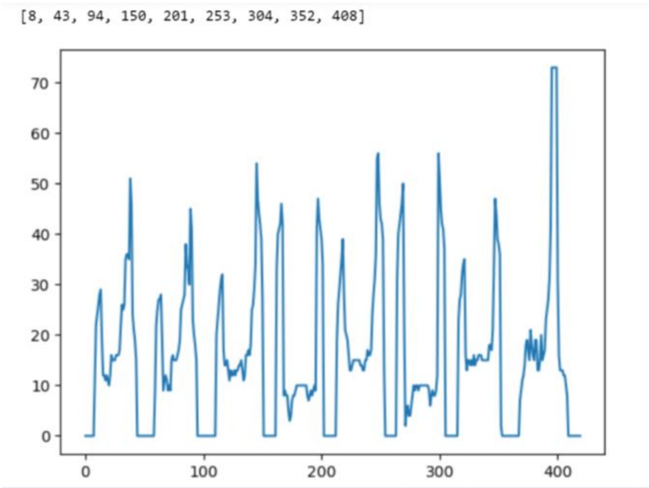
[程式碼]

```
# 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]

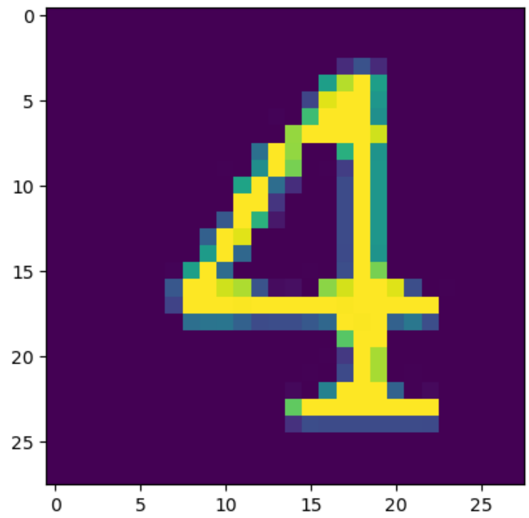


[目的與動機] 將數字存成圖檔

[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。



2. 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

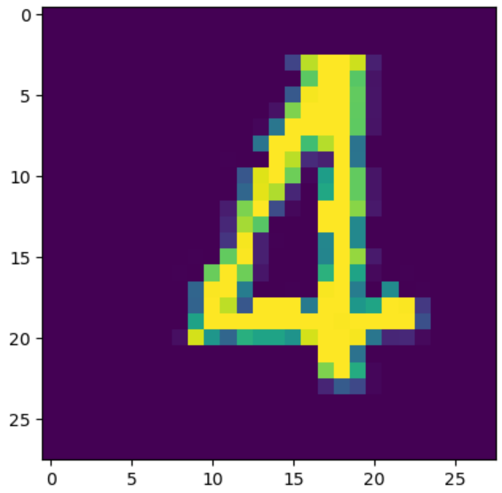
[程式碼]

```
# 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。



四、 結果

(一) 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

```
Predicted label 1: 3
Predicted label 2: 3
Predicted label 3: 5
Predicted label 4: 0
Predicted label 5: 9
Predicted label 6: 0
Predicted label 7: 2
Predicted label 8: 4
```

(二) 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。

```
Predicted label 1: 3
Predicted label 2: 5
Predicted label 3: 7
Predicted label 4: 8
Predicted label 5: 0
Predicted label 6: 3
Predicted label 7: 5
Predicted label 8: 6
```

（三） 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。

```
Predicted label 1: 3
Predicted label 2: 3
Predicted label 3: 3
Predicted label 4: 0
Predicted label 5: 3
Predicted label 6: 0
Predicted label 7: 2
Predicted label 8: 4
```

(四) 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。

```
Predicted label 1: 7
Predicted label 2: 8
Predicted label 3: 7
Predicted label 4: 8
Predicted label 5: 0
Predicted label 6: 3
Predicted label 7: 8
Predicted label 8: 2
```

五、 面臨問題

(一) 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。

(二) 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。

(三) 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。

六、 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。

七、 參考資料

1. https://ithelp.ithome.com.tw/articles/10193469
2. https://www.wpgdadatong.com/blog/detail/46474
3. https://github.com/gradient-ai/LeNet5-Tutorial/blob/main/LeNet5.ipynb
4. https://blog.paperspace.com/writing-lenet5-from-scratch-in-python/
5. https://blog.csdn.net/Zh_1999a/article/details/107526001
6. 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
7. https://github.com/ChawDoe/LeNet5-MNIST-PyTorch/blob/master/train.py
8. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
9. https://finnysteps.blogspot.com/2019/02/python.html
10. 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