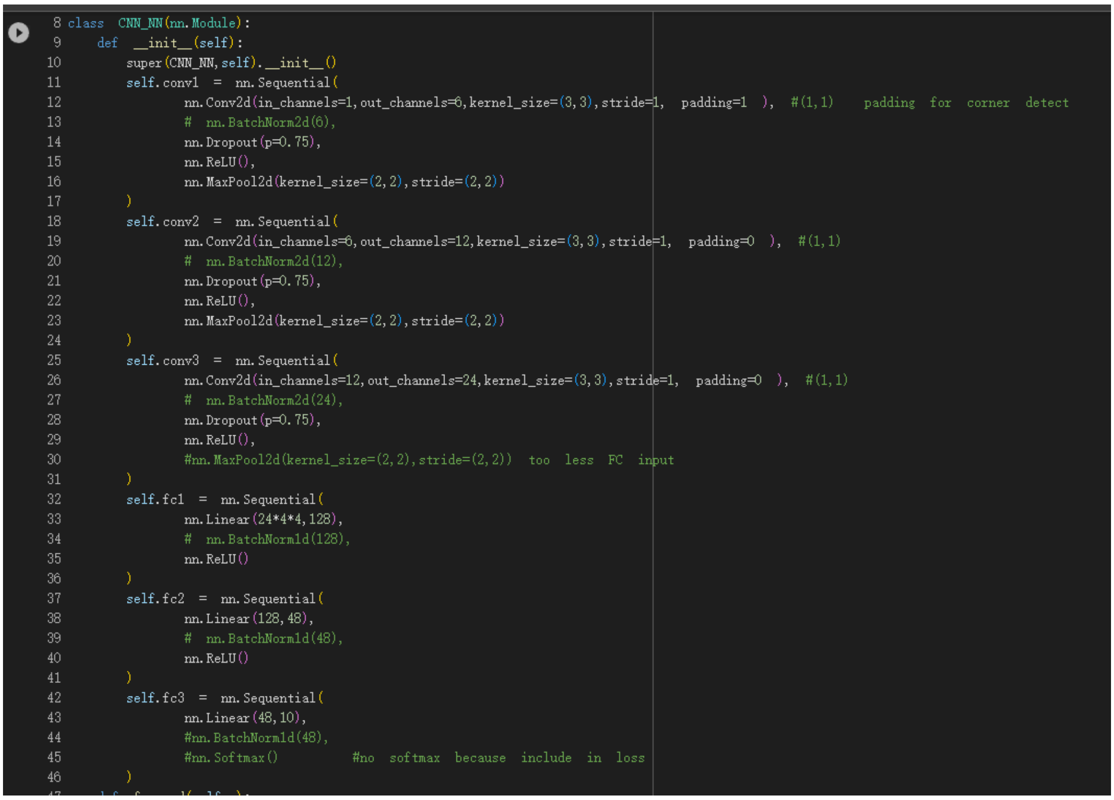
EAI

Lab 2
Report

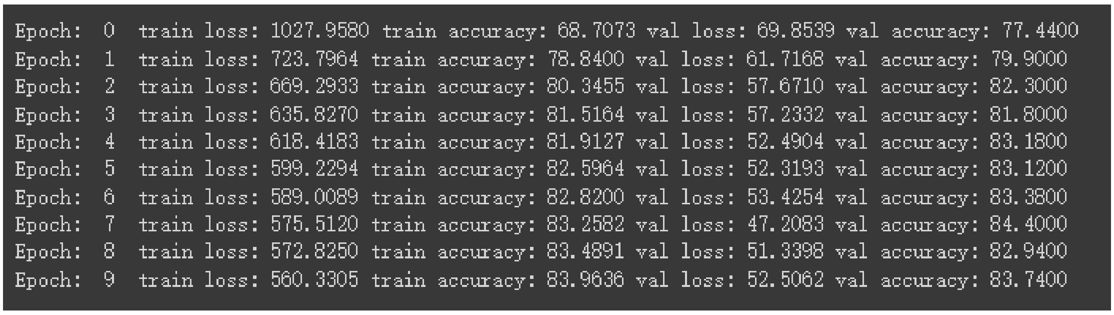
系級	113 電機乙
學號	F64096114
姓名	郭家佑

- ## Comparison w/ and w/o Batch Normalization Layer

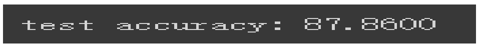


```python
class  CNN_NN(nn.Module):
    def  __init__(self):
        super(CNN_NN,self).__init__()
        self.conv1  =  nn.Sequential(
                nn.Conv2d(in_channels=1,out_channels=6,kernel_size=(3,3),stride=1,  padding=1  ),  #(1,1)   padding  for  corner  detect
                #  nn.BatchNorm2d(6),
                nn.Dropout(p=0.75),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        )
        self.conv2  =  nn.Sequential(
                nn.Conv2d(in_channels=6,out_channels=12,kernel_size=(3,3),stride=1,  padding=0  ),  #(1,1)
                #  nn.BatchNorm2d(12),
                nn.Dropout(p=0.75),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        )
        self.conv3  =  nn.Sequential(
                nn.Conv2d(in_channels=12,out_channels=24,kernel_size=(3,3),stride=1,  padding=0  ),  #(1,1)
                #  nn.BatchNorm2d(24),
                nn.Dropout(p=0.75),
                nn.ReLU(),
                #nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))  too  less  FC  input
        )
        self.fc1  =  nn.Sequential(
                nn.Linear(24*4*4,128),
                #  nn.BatchNorm1d(128),
                nn.ReLU()
        )
        self.fc2  =  nn.Sequential(
                nn.Linear(128,48),
                #  nn.BatchNorm1d(48),
                nn.ReLU()
        )
        self.fc3  =  nn.Sequential(
                nn.Linear(48,10),
                #nn.BatchNorm1d(48),
                #nn.Softmax()        #no  softmax  because  include  in  loss
        )
```

- ### With BN

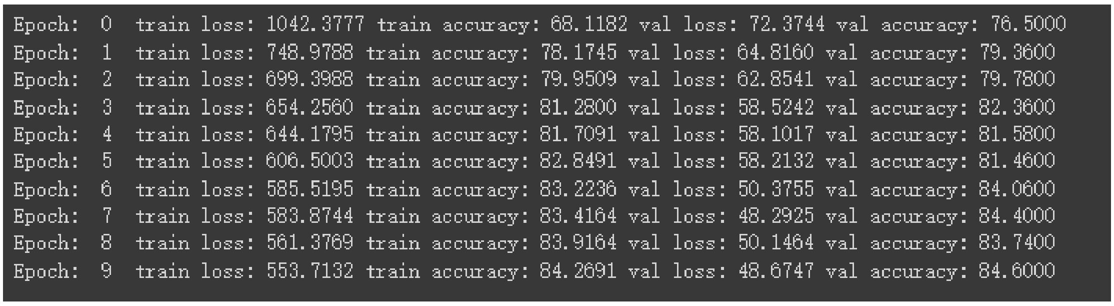


```
Epoch:  0  train loss: 1027.9580 train accuracy: 68.7073 val loss: 69.8539 val accuracy: 77.4400
Epoch:  1  train loss: 723.7964 train accuracy: 78.8400 val loss: 61.7168 val accuracy: 79.9000
Epoch:  2  train loss: 669.2933 train accuracy: 80.3455 val loss: 57.6710 val accuracy: 82.3000
Epoch:  3  train loss: 635.8270 train accuracy: 81.5164 val loss: 57.2332 val accuracy: 81.8000
Epoch:  4  train loss: 618.4183 train accuracy: 81.9127 val loss: 52.4904 val accuracy: 83.1800
Epoch:  5  train loss: 599.2294 train accuracy: 82.5964 val loss: 52.3193 val accuracy: 83.1200
Epoch:  6  train loss: 589.0089 train accuracy: 82.8200 val loss: 53.4254 val accuracy: 83.3800
Epoch:  7  train loss: 575.5120 train accuracy: 83.2582 val loss: 47.2083 val accuracy: 84.4000
Epoch:  8  train loss: 572.8250 train accuracy: 83.4891 val loss: 51.3398 val accuracy: 82.9400
Epoch:  9  train loss: 560.3305 train accuracy: 83.9636 val loss: 52.5062 val accuracy: 83.7400
```

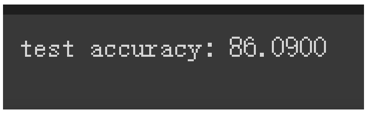


```
test accuracy: 87.8600
```

- ### Without BN



```
Epoch:  0  train loss: 1042.3777 train accuracy: 68.1182 val loss: 72.3744 val accuracy: 76.5000
Epoch:  1  train loss: 748.9788 train accuracy: 78.1745 val loss: 64.8160 val accuracy: 79.3600
Epoch:  2  train loss: 699.3988 train accuracy: 79.9509 val loss: 62.8541 val accuracy: 79.7800
Epoch:  3  train loss: 654.2560 train accuracy: 81.2800 val loss: 58.5242 val accuracy: 82.3600
Epoch:  4  train loss: 644.1795 train accuracy: 81.7091 val loss: 58.1017 val accuracy: 81.5800
Epoch:  5  train loss: 606.5003 train accuracy: 82.8491 val loss: 58.2132 val accuracy: 81.4600
Epoch:  6  train loss: 585.5195 train accuracy: 83.2236 val loss: 50.3755 val accuracy: 84.0600
Epoch:  7  train loss: 583.8744 train accuracy: 83.4164 val loss: 48.2925 val accuracy: 84.4000
Epoch:  8  train loss: 561.3769 train accuracy: 83.9164 val loss: 50.1464 val accuracy: 83.7400
Epoch:  9  train loss: 553.7132 train accuracy: 84.2691 val loss: 48.6747 val accuracy: 84.6000
```

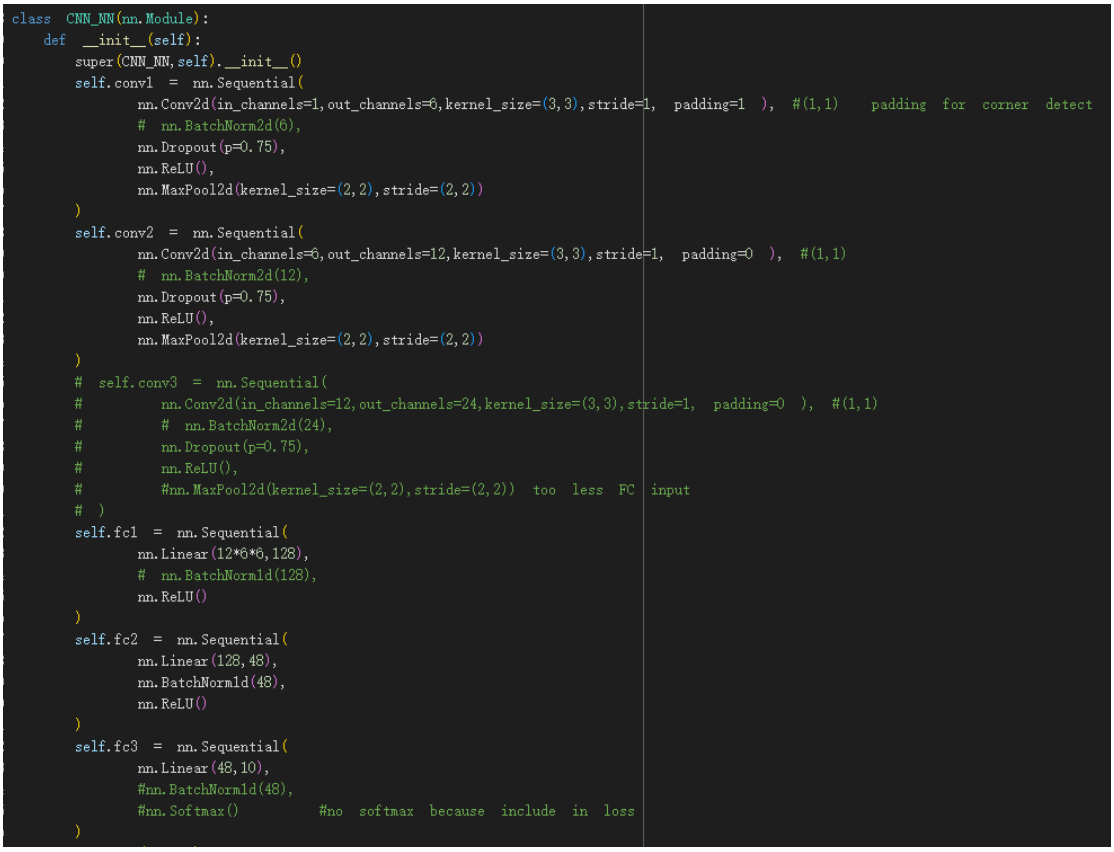


```
test accuracy: 86.0900
```

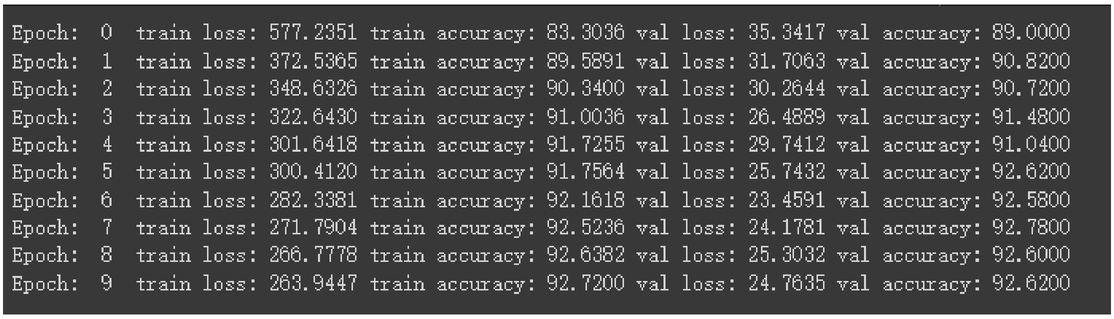
由上述結果可看出有 Batch normalization 後，要進去 activation 的 data 會往中心點靠並壓縮，對 activation function 出來的結果會越準確，因此整體的 accuracy 較高 loss 也較低。

● Comparison w/ arbitrary layer of abovementioned CNN network

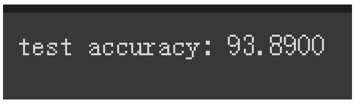
■ 2CNN+3NN



```
class  CNN_NN(nn.Module):
    def  __init__(self):
        super(CNN_NN,self).__init__()
        self.conv1  =  nn.Sequential(
                nn.Conv2d(in_channels=1,out_channels=6,kernel_size=(3,3),stride=1,   padding=1  ),  #(1,1)    padding  for  corner  detect
                #  nn.BatchNorm2d(6),
                nn.Dropout(p=0.75),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        )
        self.conv2  =  nn.Sequential(
                nn.Conv2d(in_channels=6,out_channels=12,kernel_size=(3,3),stride=1,   padding=0  ),  #(1,1)
                #  nn.BatchNorm2d(12),
                nn.Dropout(p=0.75),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        )
        #  self.conv3  =  nn.Sequential(
        #          nn.Conv2d(in_channels=12,out_channels=24,kernel_size=(3,3),stride=1,   padding=0  ),  #(1,1)
        #          #  nn.BatchNorm2d(24),
        #          nn.Dropout(p=0.75),
        #          nn.ReLU(),
        #          #nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))  too  less  FC  input
        #  )
        self.fc1  =  nn.Sequential(
                nn.Linear(12*6*6,128),
                #  nn.BatchNorm1d(128),
                nn.ReLU()
        )
        self.fc2  =  nn.Sequential(
                nn.Linear(128,48),
                nn.BatchNorm1d(48),
                nn.ReLU()
        )
        self.fc3  =  nn.Sequential(
                nn.Linear(48,10),
                #nn.BatchNorm1d(48),
                #nn.Softmax()          #no  softmax  because  include  in  loss
        )
```



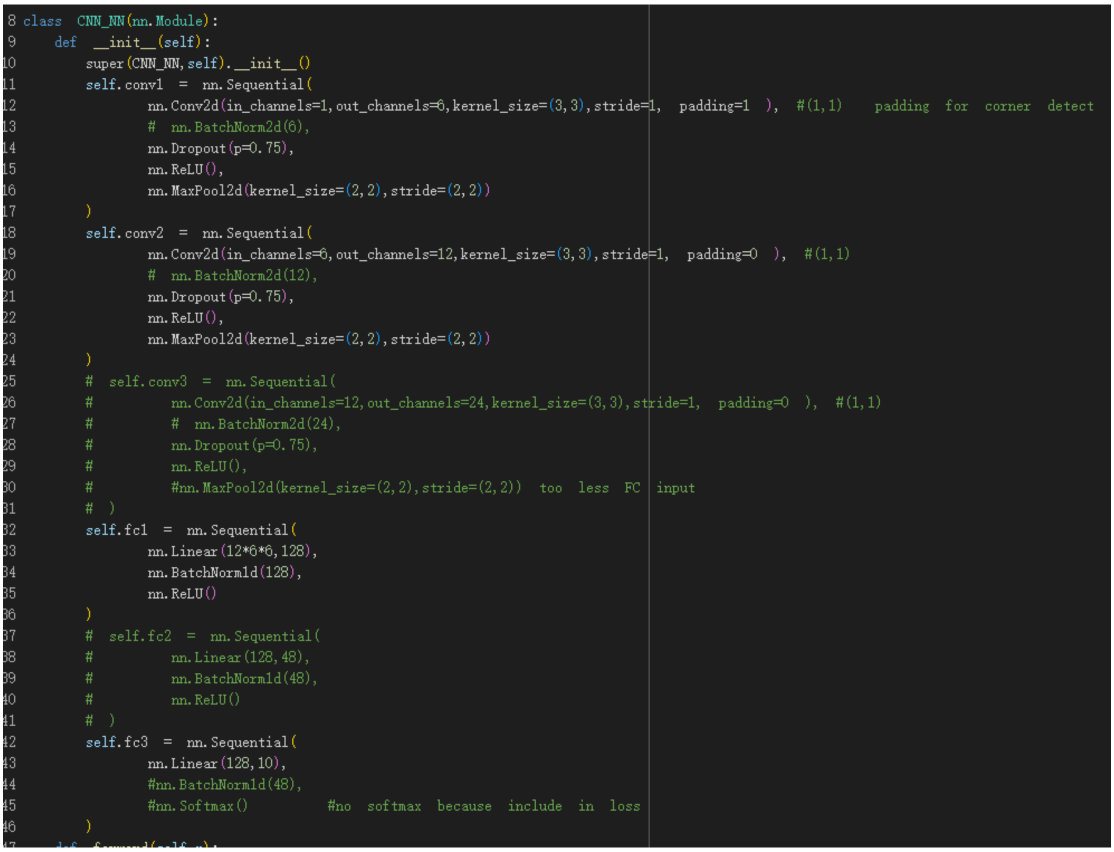
```
Epoch:  0  train loss: 577.2351 train accuracy: 83.3036 val loss: 35.3417 val accuracy: 89.0000
Epoch:  1  train loss: 372.5365 train accuracy: 89.5891 val loss: 31.7063 val accuracy: 90.8200
Epoch:  2  train loss: 348.6326 train accuracy: 90.3400 val loss: 30.2644 val accuracy: 90.7200
Epoch:  3  train loss: 322.6430 train accuracy: 91.0036 val loss: 26.4889 val accuracy: 91.4800
Epoch:  4  train loss: 301.6418 train accuracy: 91.7255 val loss: 29.7412 val accuracy: 91.0400
Epoch:  5  train loss: 300.4120 train accuracy: 91.7564 val loss: 25.7432 val accuracy: 92.6200
Epoch:  6  train loss: 282.3381 train accuracy: 92.1618 val loss: 23.4591 val accuracy: 92.5800
Epoch:  7  train loss: 271.7904 train accuracy: 92.5236 val loss: 24.1781 val accuracy: 92.7800
Epoch:  8  train loss: 266.7778 train accuracy: 92.6382 val loss: 25.3032 val accuracy: 92.6000
Epoch:  9  train loss: 263.9447 train accuracy: 92.7200 val loss: 24.7635 val accuracy: 92.6200
```



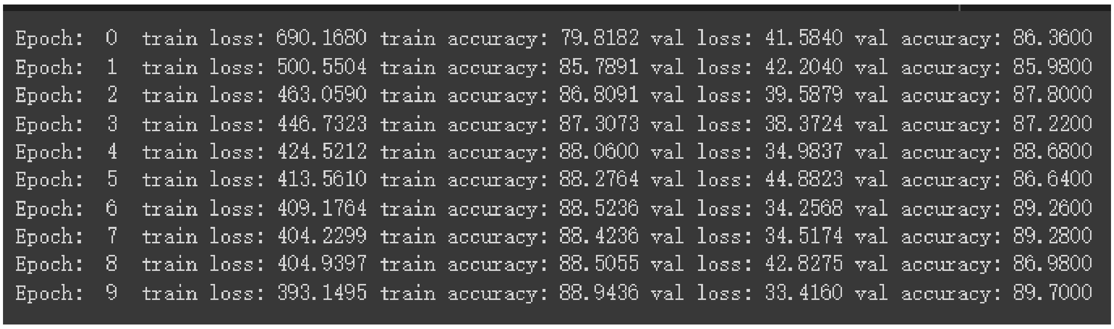
```
test accuracy: 93.8900
```

可看出將前面 CNN 的 layer 減少一層可使準確率上升不少，我推測原因是 3 層 CNN layer 對 MNIST 這種輕量化的 dataset 來說太過了，而且若做三次 Maxpooling 後會讓 feature size 變成 2*2 而已，會使的 feature 太小而影響之後分類層判斷的準確度，所以兩層出來的 6*6 會有更好的結果。

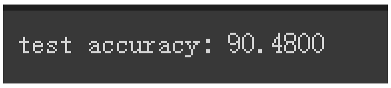
■ 2CNN+2NN



```
class  CNN_NN(nn.Module):
    def  __init__(self):
        super(CNN_NN,self).__init__()
        self.conv1  =  nn.Sequential(
                nn.Conv2d(in_channels=1,out_channels=6,kernel_size=(3,3),stride=1,  padding=1  ),  #(1,1)   padding  for  corner  detect
                #  nn.BatchNorm2d(6),
                nn.Dropout(p=0.75),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        )
        self.conv2  =  nn.Sequential(
                nn.Conv2d(in_channels=6,out_channels=12,kernel_size=(3,3),stride=1,  padding=0  ),  #(1,1)
                #  nn.BatchNorm2d(12),
                nn.Dropout(p=0.75),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        )
        #  self.conv3  =  nn.Sequential(
        #          nn.Conv2d(in_channels=12,out_channels=24,kernel_size=(3,3),stride=1,  padding=0  ),  #(1,1)
        #          #  nn.BatchNorm2d(24),
        #          nn.Dropout(p=0.75),
        #          nn.ReLU(),
        #          #nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))  too  less  FC  input
        #  )
        self.fc1  =  nn.Sequential(
                nn.Linear(12*6*6,128),
                nn.BatchNorm1d(128),
                nn.ReLU()
        )
        #  self.fc2  =  nn.Sequential(
        #          nn.Linear(128,48),
        #          nn.BatchNorm1d(48),
        #          nn.ReLU()
        #  )
        self.fc3  =  nn.Sequential(
                nn.Linear(128,10),
                #nn.BatchNorm1d(48),
                #nn.Softmax()         #no  softmax  because  include  in  loss
        )
    def  forward(self,x):
```



```
Epoch:  0  train loss: 690.1680 train accuracy: 79.8182 val loss: 41.5840 val accuracy: 86.3600
Epoch:  1  train loss: 500.5504 train accuracy: 85.7891 val loss: 42.2040 val accuracy: 85.9800
Epoch:  2  train loss: 463.0590 train accuracy: 86.8091 val loss: 39.5879 val accuracy: 87.8000
Epoch:  3  train loss: 446.7323 train accuracy: 87.3073 val loss: 38.3724 val accuracy: 87.2200
Epoch:  4  train loss: 424.5212 train accuracy: 88.0600 val loss: 34.9837 val accuracy: 88.6800
Epoch:  5  train loss: 413.5610 train accuracy: 88.2764 val loss: 44.8823 val accuracy: 86.6400
Epoch:  6  train loss: 409.1764 train accuracy: 88.5236 val loss: 34.2568 val accuracy: 89.2600
Epoch:  7  train loss: 404.2299 train accuracy: 88.4236 val loss: 34.5174 val accuracy: 89.2800
Epoch:  8  train loss: 404.9397 train accuracy: 88.5055 val loss: 42.8275 val accuracy: 86.9800
Epoch:  9  train loss: 393.1495 train accuracy: 88.9436 val loss: 33.4160 val accuracy: 89.7000
```

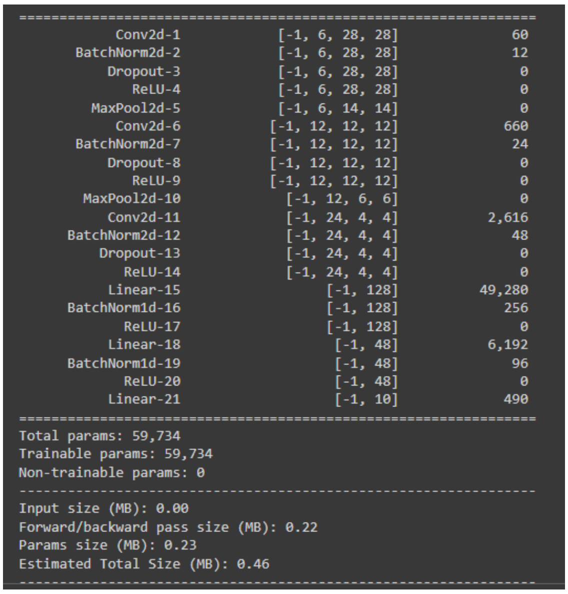


```
test accuracy: 90.4800
```

相比上一個 2CNN+3NN 的模型，這個少了一層 Fully connected layer 的模型的準確率比較低，我推測是分類問題越深的網路會有越準確的預測結果。

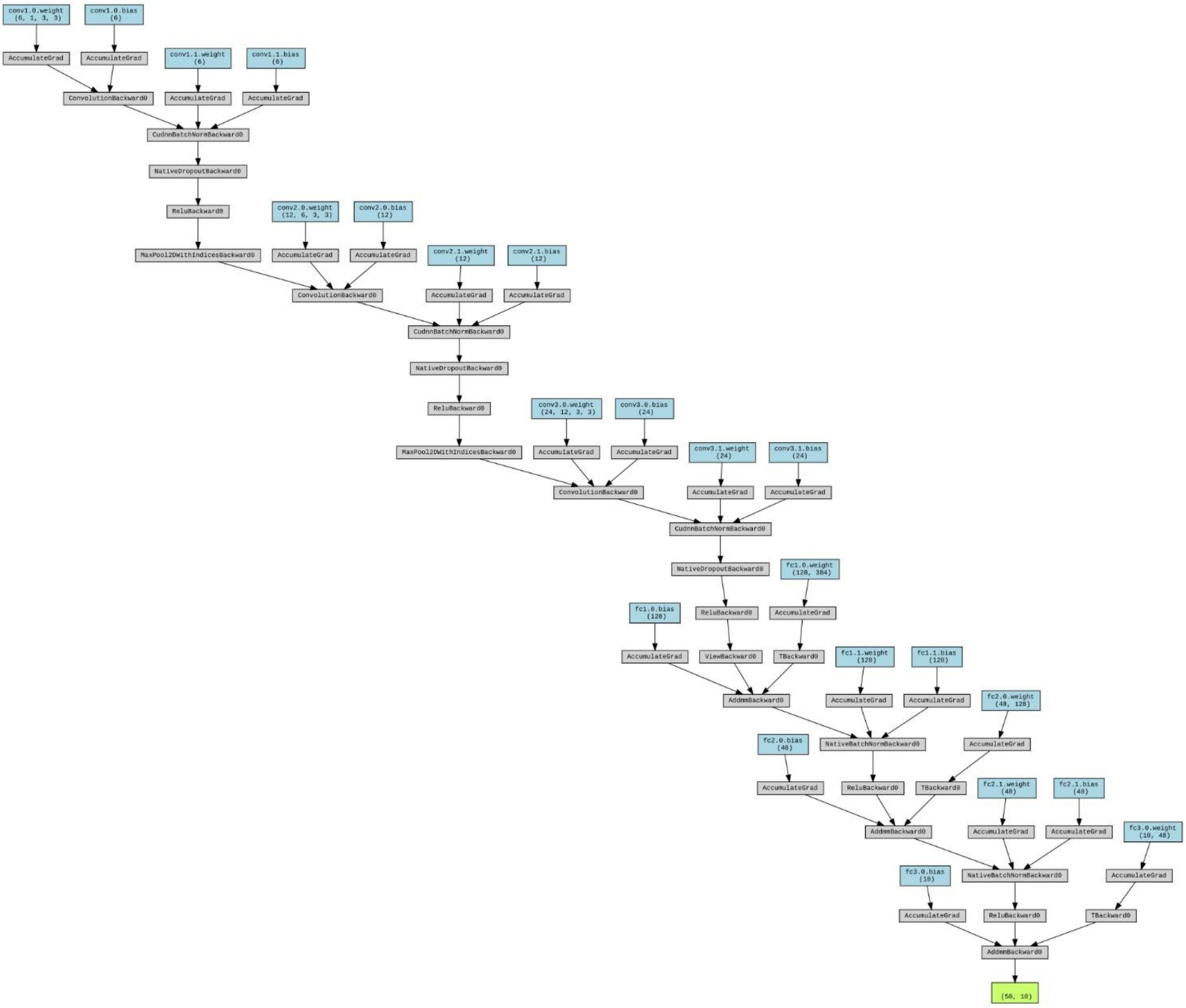
● 截圖

■ model summary

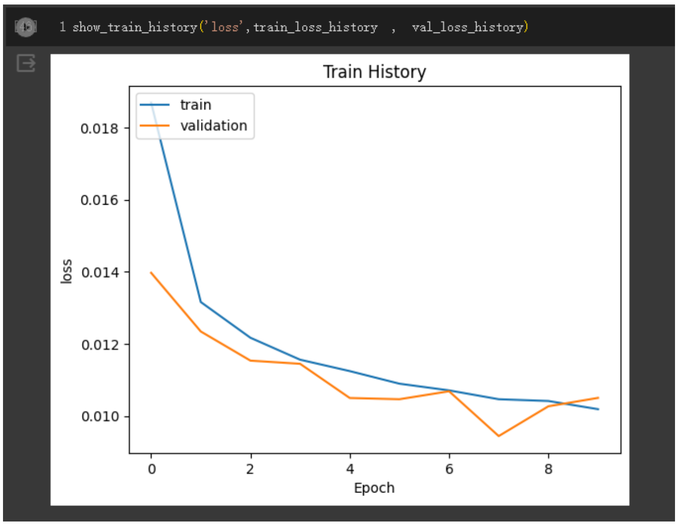


```
================================================================
            Conv2d-1            [-1, 6, 28, 28]              60
       BatchNorm2d-2            [-1, 6, 28, 28]              12
           Dropout-3            [-1, 6, 28, 28]               0
              ReLU-4            [-1, 6, 28, 28]               0
         MaxPool2d-5            [-1, 6, 14, 14]               0
            Conv2d-6           [-1, 12, 12, 12]             660
       BatchNorm2d-7           [-1, 12, 12, 12]              24
           Dropout-8           [-1, 12, 12, 12]               0
              ReLU-9           [-1, 12, 12, 12]               0
        MaxPool2d-10             [-1, 12, 6, 6]               0
           Conv2d-11             [-1, 24, 4, 4]           2,616
      BatchNorm2d-12             [-1, 24, 4, 4]              48
          Dropout-13             [-1, 24, 4, 4]               0
             ReLU-14             [-1, 24, 4, 4]               0
           Linear-15                  [-1, 128]          49,280
      BatchNorm1d-16                  [-1, 128]             256
             ReLU-17                  [-1, 128]               0
           Linear-18                   [-1, 48]           6,192
      BatchNorm1d-19                   [-1, 48]              96
             ReLU-20                   [-1, 48]               0
           Linear-21                   [-1, 10]             490
================================================================
Total params: 59,734
Trainable params: 59,734
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.22
Params size (MB): 0.23
Estimated Total Size (MB): 0.46
----------------------------------------------------------------
```

- **plot model**

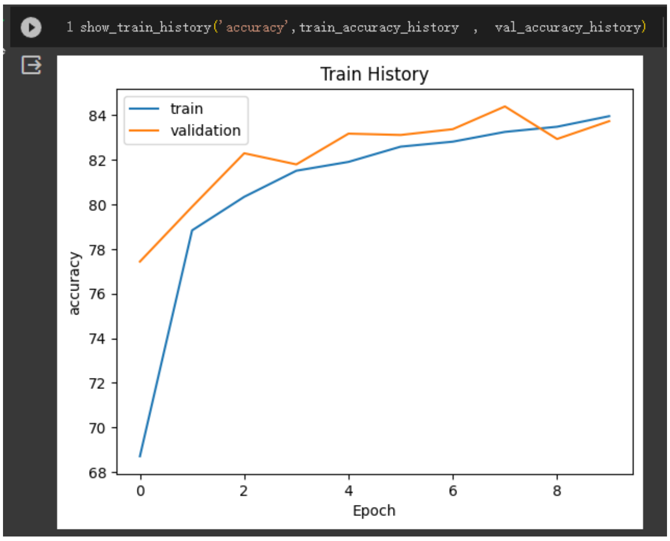


- **Loss plot**



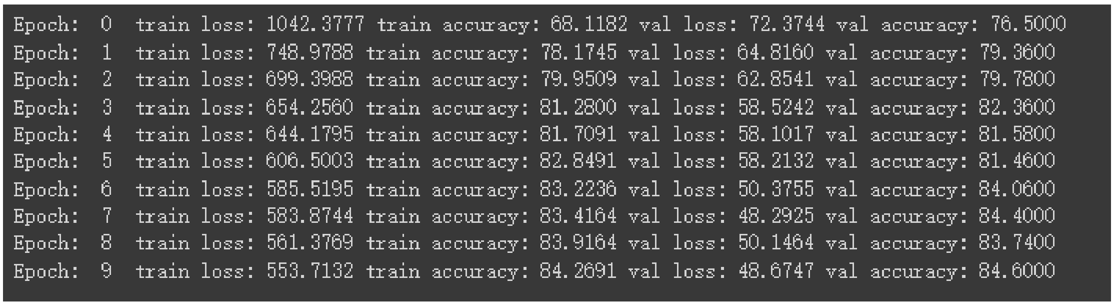
```
show_train_history('loss',train_loss_history , val_loss_history)
```

- **accuracy plot**



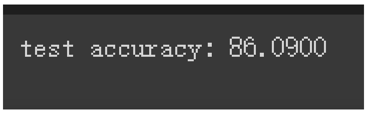
```
show_train_history('accuracy',train_accuracy_history , val_accuracy_history)
```

- **Training accuracy**



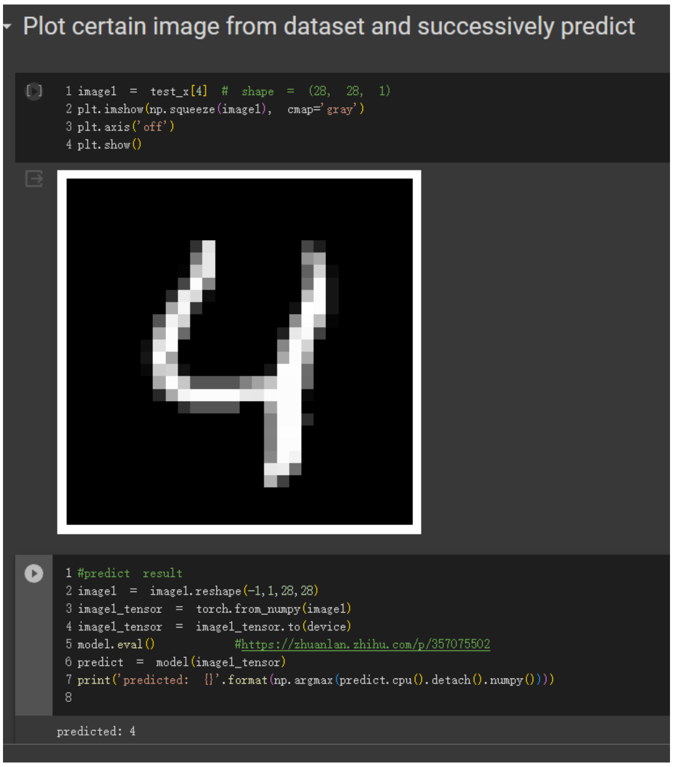
```
Epoch:  0  train loss: 1042.3777 train accuracy: 68.1182 val loss: 72.3744 val accuracy: 76.5000
Epoch:  1  train loss: 748.9788 train accuracy: 78.1745 val loss: 64.8160 val accuracy: 79.3600
Epoch:  2  train loss: 699.3988 train accuracy: 79.9509 val loss: 62.8541 val accuracy: 79.7800
Epoch:  3  train loss: 654.2560 train accuracy: 81.2800 val loss: 58.5242 val accuracy: 82.3600
Epoch:  4  train loss: 644.1795 train accuracy: 81.7091 val loss: 58.1017 val accuracy: 81.5800
Epoch:  5  train loss: 606.5003 train accuracy: 82.8491 val loss: 58.2132 val accuracy: 81.4600
Epoch:  6  train loss: 585.5195 train accuracy: 83.2236 val loss: 50.3755 val accuracy: 84.0600
Epoch:  7  train loss: 583.8744 train accuracy: 83.4164 val loss: 48.2925 val accuracy: 84.4000
Epoch:  8  train loss: 561.3769 train accuracy: 83.9164 val loss: 50.1464 val accuracy: 83.7400
Epoch:  9  train loss: 553.7132 train accuracy: 84.2691 val loss: 48.6747 val accuracy: 84.6000
```

- **testing accuracy**



```
test accuracy: 86.0900
```

- **Plot certain image from dataset and successively predict**



Plot certain image from dataset and successively predict

```
image1 = test_x[4] # shape = (28, 28, 1)
plt.imshow(np.squeeze(image1), cmap='gray')
plt.axis('off')
plt.show()
```

```
#predict result
image1 = image1.reshape(-1,1,28,28)
image1_tensor = torch.from_numpy(image1)
image1_tensor = image1_tensor.to(device)
model.eval()           #https://zhuanlan.zhihu.com/p/357075502
predict = model(image1_tensor)
print('predicted: {}'.format(np.argmax(predict.cpu().detach().numpy())))

```

```
predicted: 4
```

- # 遇到的困難及你後來是如何解決的

1. Numpy 與 tensor 轉換 & cpu 與 cuda 轉換

   Ans:

   Numpy➔ tensor: torch.from_numpy()

   tensor ➔Numpy: .numpy()

   一般要用 GPU 運算要用: .to(device)

   device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

   而有些要用 numpy 才有的功能必須從 tensor 轉回 numpy，因為使用的是 cuda，必須先用.cpu()再用.numpy()轉。

2. 切 Batch 是如何運作?

   Ans:要把 input 想成多一維的 matrix，而後續的運作方式

皆與一筆資料相同，只是多一維 batch size 的維度。

3. 最後一層一定要用到 BN?? 不確定是不是跟 Adam optimizer 有關