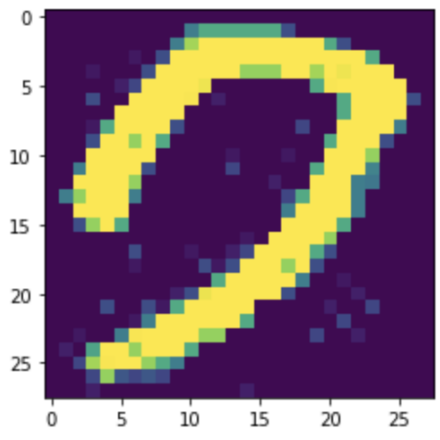
DL HW1_0513460 資財09胡明秀

Q1 Deep Neural Network for Classification

Data Preprocessing

- 原本為28* 28的圖片(如下圖)，將28x28的圖片拉成一條vector(784)，並經過normalization, one-hot encoding及reshape之後得到我們要的x_train, y_train, x_test, y_test。



- Normalization將data/255，而one-hot encoding則將label轉成0,1的dummy矩陣。

Model Design

- 因為本題為Multi-class的Classification問題，因此選用categorical cross entropy作為loss function。

$$E(w) = \sum_{n=1}^{N}\sum_{k=1}^{K} y(w) log \hat{y}(w)$$

- Output Layer前面接softmax function，將輸出的10個neurons轉換成10個0~1間的機率值。我們在prediction時便挑選機率最大的作為predicted values

$$f(z) = \frac{e^z}{\sum_{k=1}^{K} e^{z_k}}$$

- Hidden Layers間的activation則挑選sigmoid function

$$S(t) = \frac{1}{1+e^{-t}}$$

- Hyperparameters挑選
  - number of hidden layers: 3

    ▼ 原本使用2層，後來加成3層。似乎對於training error loss的收斂效果都差不多。故最後就直接挑選3層。
  - number of hidden units: 128, 64, 32
  - learning rate: 0.3

    ▼ 原本一開始挑選lr = 0.5做訓練，發現accuracy停在80%左右。後來嘗試lr = 0.1似乎又太小，model學不到什麼features。故最後決定使用lr = 0.3的效果最佳。
  - number of iterations = 3000

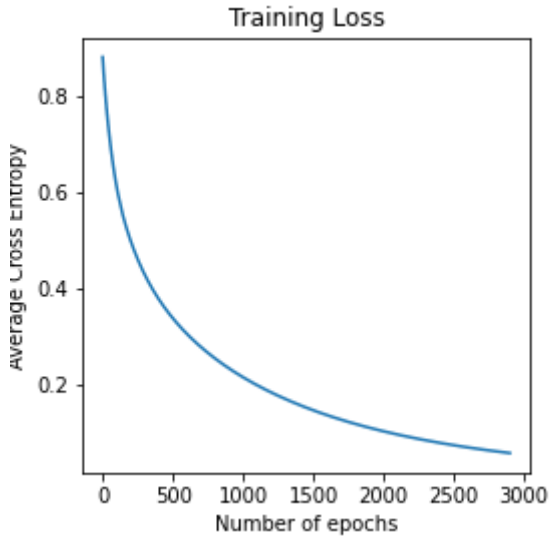
    ▼ 原本一開始先做epochs = 1000試試看，發現似乎loss不會收斂。所以嘗試epochs = 5000，在訓練的過程中發現其實差不多到3000左右loss就差不多收斂了。故最後挑選epochs = 3000
  - mini-batch size = 100

    ▼ 參考網路上面挑選mini-batch size的大小似乎都是100。先試驗mini-batch = 100，後來試驗200和300的效果差不多，故最後便選擇mini-batch = 100。

- code實作架構說明

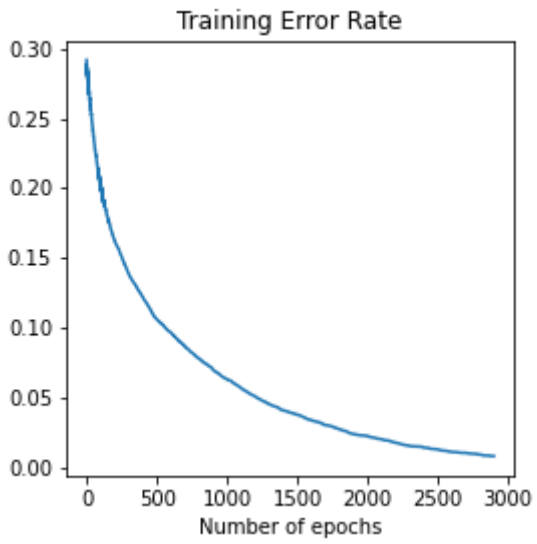
    - 建構class DNN()，SGD訓練則獨立在class外面。詳細請見051360_hw1_Q1.py檔
- Learning Curve

    ▼ Training Error Loss = 0.0564



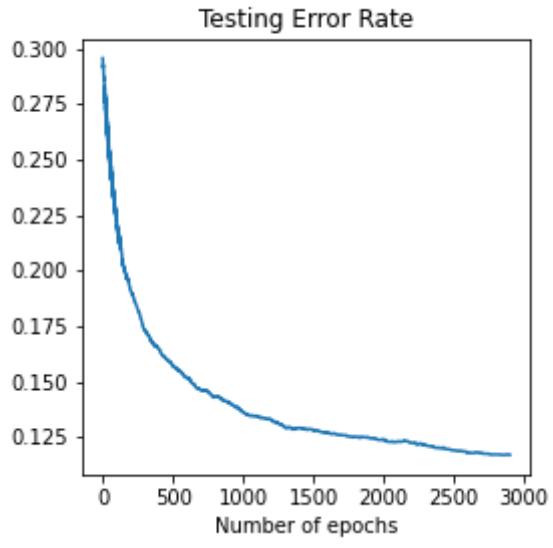
- Training Error Rate

    ▼ Training Error Rate = 0.0094



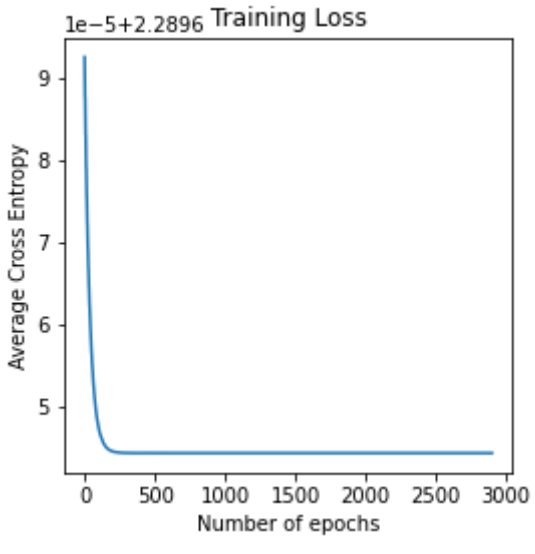
- Test Error Rate

    ▼ Test Error Rate = 0.1173



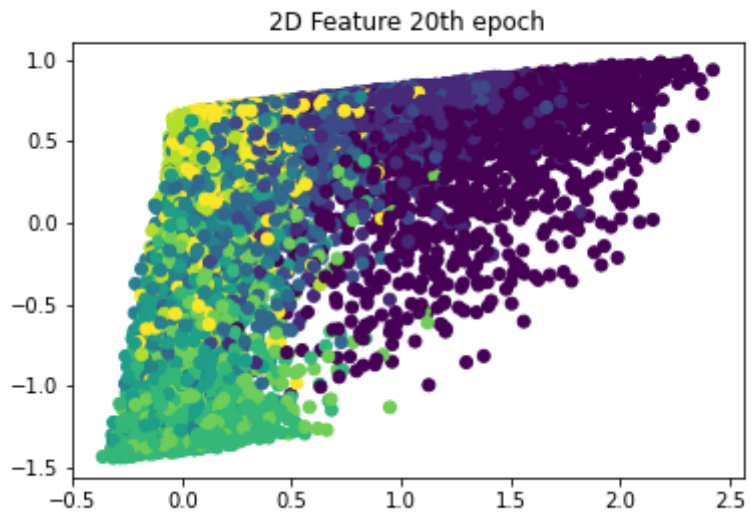
- **Zero** and **Random initializations** for model wights

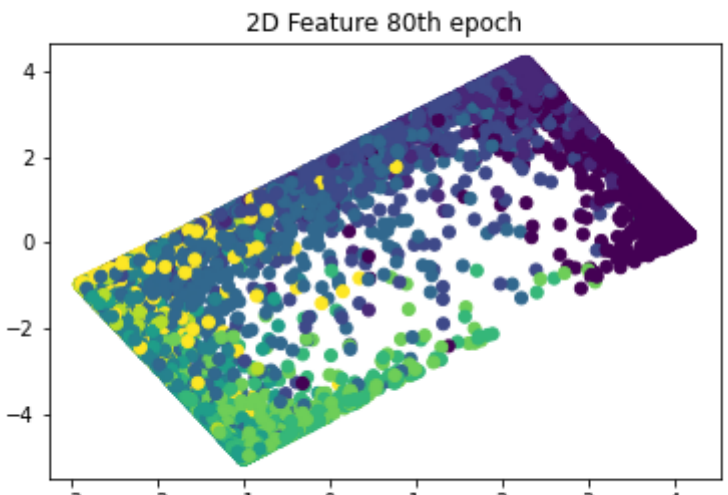
    以上三張表現很棒的loss和error rate的圖皆使用standard normal distribution來產生隨機的initial weights。但是，如果使用Zero initializations，表現會相當的差。training loss一開始會急速下降，之後就無法收斂了(請見下圖)。training error rate和test error rate則幾乎完全沒有往下降(training error = 0.6772, testing error = 0.6854）。以下為使用zero initializations的training loss結果。此結果也驗證的老師上課提到的random weights initialization可以有效地提升Model準確度。



- Distributions of latent features
  - Evolution of latent features:

    ▼ 可以觀察到20th epoch時，hidden layer的latent features仍擠在一起。到了80th epoch時，latent features便有比較分開的感覺（觀察深紫色的資料點），往自己的類別集中。由此可以見，模型預測的準確率有隨著訓練提升。





- Confusion Matrix

  以下為testing data的confusion matrix結果：

  計算對角線的結果可以得到test accuracy = 88.28016 %

```
array([[630,  21,   5,   2,   1,   0,   3,   2,   0,   0],
       [ 14, 627,  12,   5,   0,   0,   1,   0,   0,   2],
       [ 16,  14, 479,  65,   2,   0,   2,   2,   2,   2],
       [  1,   0,  75, 475,   6,   7,   7,   7,  15,   7],
       [  1,   0,   1,   4, 569,   0,  10,   5,  22,  39],
       [  0,   0,   0,   8,   6, 366,  14,   7,   3,   3],
       [  1,   0,   0,   1,   7,  12, 454,  26,   0,   1],
       [  5,   0,   2,   5,   9,  10,  24, 383,   3,   8],
       [  0,   0,   0,   9,  29,  10,   1,  11, 512,   0],
       [  3,   4,   4,  11,  40,   6,   5,   5,   3, 597]])
```

Q2 Convolutional Neural Network for Image Recognition

Data Preprocessing

- Image processing

  ▼ match label with image:

  ▼ 目標：把train.csv中的label和image配對成一個二維陣列：(image, label)

  ▼ 作法：設計兩個function，一個把images folder中的images和名字zip成一個array，另一個則利用前一個function的結果將每張照片的good / bad / none 和cropped image配對。詳細內容請見051360_hw1_Q2.py中的以下兩個function:

```
def load_images_from_folder(folder):
```

```
def get_img_train_label(images_list_with_name):
```

  ▼ image resize

  ▼ 目標：將images input的大小調整成一樣

  ▼ 作法：因為CNN模型的架構，故我們必須將input image的size調整成一樣才能輸入。使用OpenCV的resize將get_img_train_label的的照片都轉換成(3, 256, 256)的格式。原本實驗使用其他的格式大小發現training出來的效果沒有(3, 256, 256)的效果好。可能其他的大小會壓縮到照片本身的resolution，而(3, 256, 256)是對於這組dataset還不錯的resize格式選擇。

Model Design

- Model-1:
  - 設計第一個CNN model (stride=(1,1), padding = (2,2))

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=262144, out_features=3, bias=True)
)
```

  - Performance:

    ▼ Number of epochs: 50

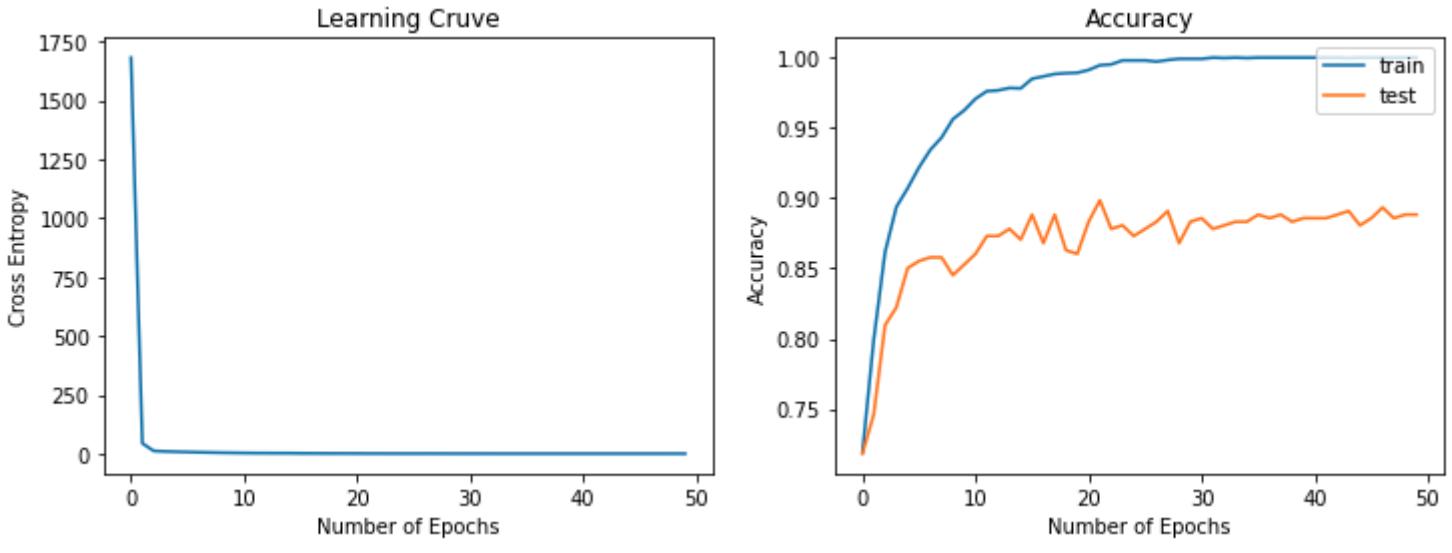
    ▼ Learning Rate: 0.0001

    ▼ Training Loss: 0.017

    ▼ Training Accuracy: 99.944%

    ▼ Testing Accuracy: 86.832%

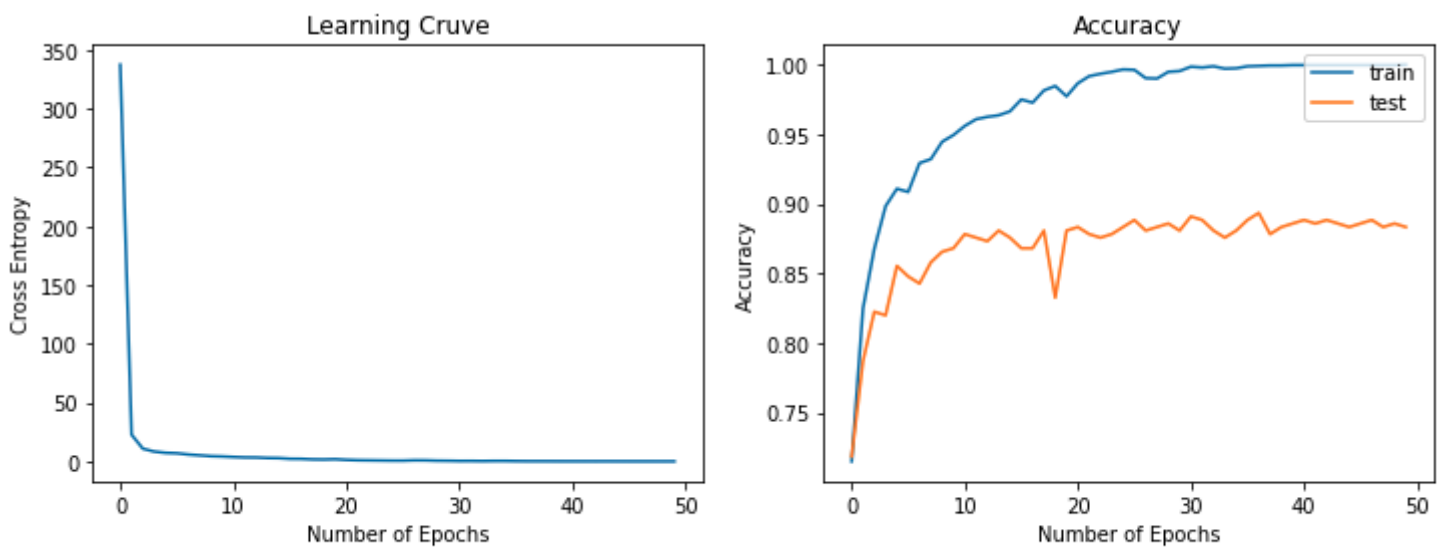
    ▼ Training Accuracy的表現很棒，但是Test Accuracy到後面的epcohs就掉下來了。故推測Model-1可能有over-fitting的問題。



- Model-2:
  - 設計第二個CNN model (stride=(1,1), padding = (3,3))。這次調整了max-pooling，希望training的時候不要抓那麼多features，Model可以在testing data的時候表現比Model-1好。

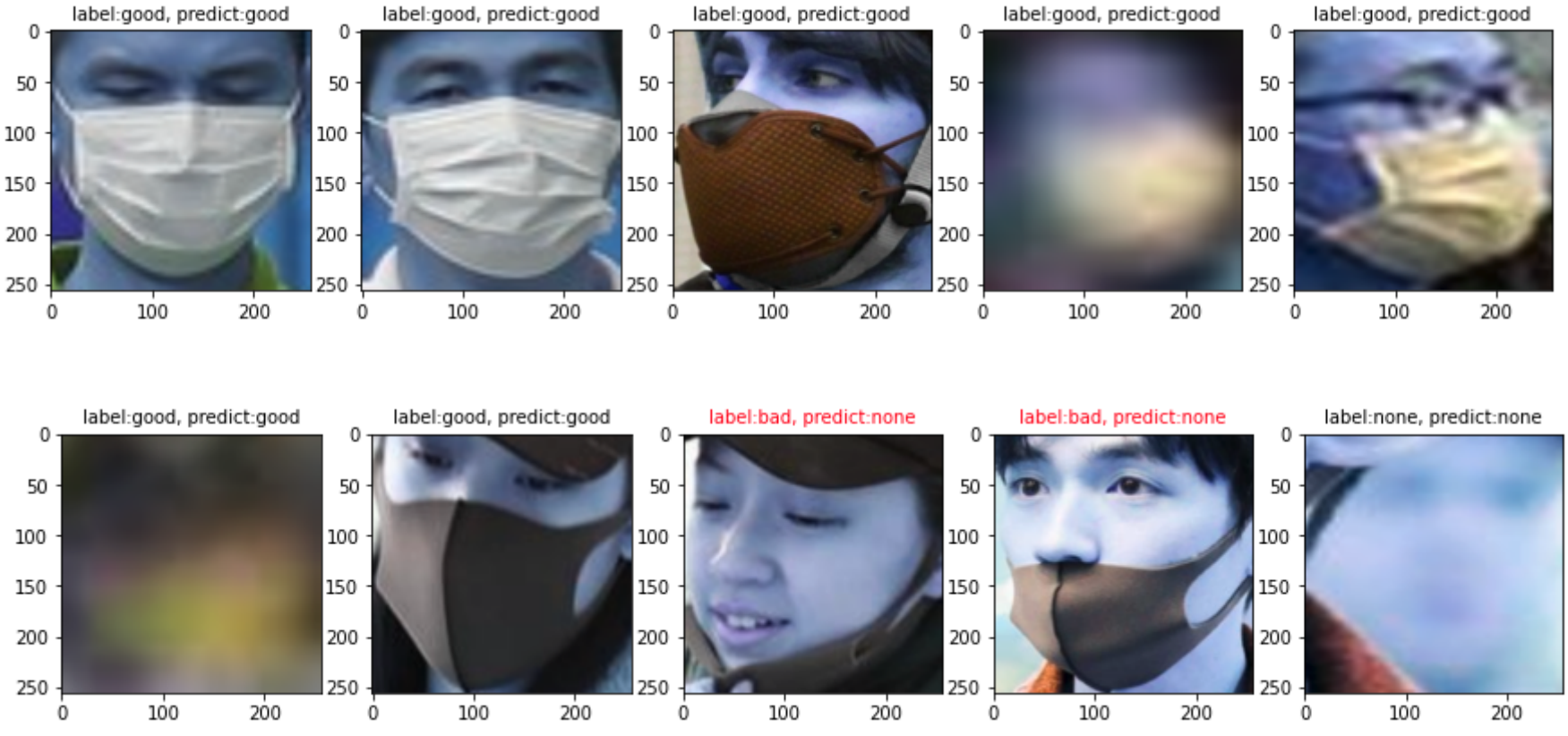
```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=65536, out_features=3, bias=True)
)
```

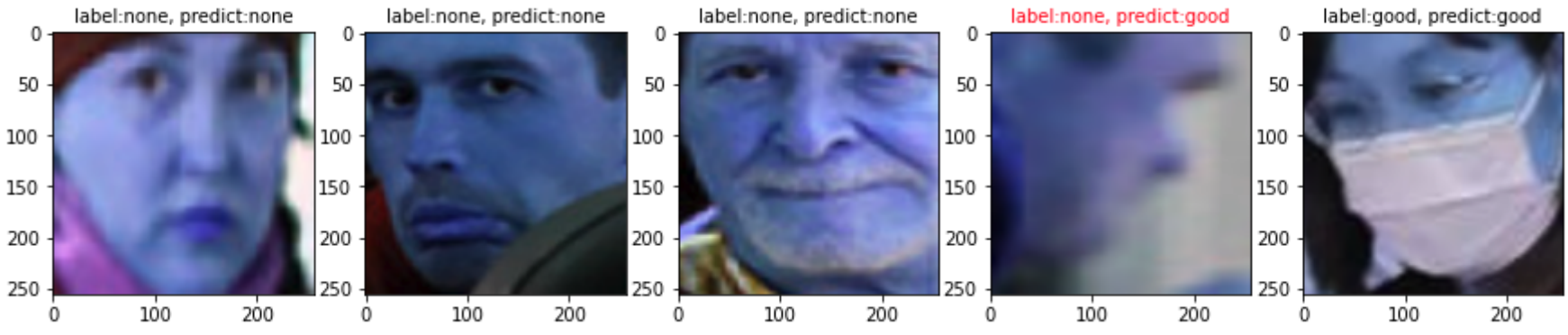
- Performance:
  - ▼ Number of epochs: 50
  - ▼ Learning Rate: 0.0001
  - ▼ Training Loss: 0.073
  - ▼ Training Accuracy: 98.112%
  - ▼ Testing Accuracy: 87.322%
  - ▼ 稍微改善一點Model-1over-fitting的問題



Classification Result

- 以下為用CNN model預測完轉回原始圖片的結果





- 以下為根據labels各別分類的ACC的表格

Classification Result by labels

Class	Train Acc	Test Acc
good	99.75%	97.51%
bad	95.50%	91.00%
none	54.54%	31.81%

可以發現none的類別，無論是training acc或是testing acc都表現得特別差。我們可以從原始資料的三個類別的比例來推測是否是原始資料的imblance導致這樣的結果。觀察原始資料的類別分佈：

- number of labels by category (train/test):
  - ***good*** (wearing mask) : 3129 (2846/283)
  - ***none*** (wrongly wearing mask) : 126 (104/22)
  - ***bad*** (no wearing mask) : 667 (578/89)

可以發現none的原始資料本來就較good和bad少很多，故CNN訓練時對於none的features較不熟悉。我採取data augmentation來把none的類別數量補到跟good和bad差不多。具體來說，把同一張none的圖片旋轉，就可以補足none類別資料數量比較少的問題。以下為經過data augmentation後的ACC表格：

Classification Result by labels Revised

Class	Train Acc	Test Acc
good	98.90%	95.50%
bad	92.50%	91.00%
none	78.33%	58.81%