Machine Learning

HW2：Neural Network

姓名：陳毅

學號：111063577

老師：孫民

日期：4/17

首先關於資料集的匯入，把老師給的 test_data 與 train_data 打開後我們發現，資料顏色為黑白且大小為 32x32bit，讀取後把圖片拉成一維的數值資料型態，而圖片標籤則存取成數字型態。

由於題目的要求，我把圖片資料利用 PCA 降維到二維，原本資料為 32x32 維，我們將其降到二維，我們先匯入 scikit-learn 中的 StandardScaler 將資料標準化，接著才做降維的動作。

Two-layer Neural Network

這次題目要求我們使用 Neural Network 去實現影像辨識的功能，其中提到說要分別實作兩層與三層的 Neural Network，輸出與輸入層節點數目為三個，由於差異在於 hidden layer 的數目不同，所以打算實作一個可控制 hidden layer 與 neuron 數量的 model，一樣參考 scikit-learn 中的 MLPClassifier。

Parameter

1.hidden_layer_sizes：用來設定 hidden layer 層數與 neuron 數量，由於我們這裡為 2NN，我們即可表示為(150,)，表示 hidden layer 一層且 neuron 數量為 150。

2.use_bias：可自行設定是否需要加入偏壓。

3.batch_size：我們會先使資料分成一個個 batch，把每個 batch 資料做 forwarding 去計算平均的 loss，再利用 loss 計算梯度更新權重。

4.epoch：當資料全部訓練過會完成一個 epoch，注意當資料無法分割完整會再下個 epoch 進行 batch 分配。

5.learning_rate：更新權重時的 learning rate。

6.random_seed：可以固定隨機數序列。

7.verbose：當 verbose 為 True 時會印出 training 時每個 epoch 的 loss 與 accuracy。

Function

1._initialize()：初始化所需的內部變數與 weight 及 bias。

2._batch_norm()：在不經過標準化就把數值丟進 sigmoid 會導致飽和無法分辨數據類別。

3._batch_norm_derivative()：batch 標準化後的導數。

4._sigmoid()：hidden layer 的 activation function。

5._sigmoid_derivative()：sigmoid 的導數。

6._softmax()：output layer 的 activation function，主要是將數值 mapping 到 0~1 區間。

7._cross_entropy()：計算 gradient 所需 loss 的 loss function。

8._softmax_and_cross_entropy_derivative()：softmax 與 cross entropy 的導數。

9._accuracy_score()：計算 prediction 的 accuracy。

10._forward_pass()：把 input data 套進整個 neural network 中，在 training 與 prediction 都會用到。

11._backprop：利用 forward 後得到的 loss 計算 gradient，並更新對應的 weight 與 bias。

12.fit(X, y)：持續將 batch data 計算出 loss 並且更新權重，以達到指定的 epoch 數。

13.predict(X)：將 data 餵進_forward_pass 預測各個類別的機率，再利用 np.argmax 以轉換成預測類別的 index。

14.predict_proba(X)：將 data 餵進_forward_pass 預測各個類別的機率。

test accuracy

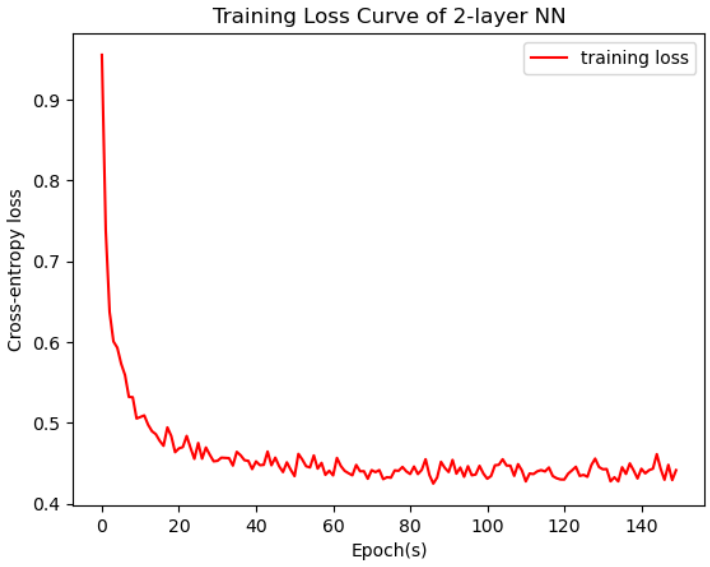
由結果可以發現 Testing accuracy 比 Training accuracy 還好，推測為在測試資料中的特徵集中導致 overfiting 的結果。

```
In [5]: from sklearn.metrics import accuracy_score

        nn2 = MYNN(random_seed=0)
        nn2.fit(X_train, y_train)

        print('Accuracy of 2NN:')
        y_pred = nn2.predict(X_train)
        print(f'Training accuracy: {accuracy_score(y_train, y_pred):f}')

        y_pred = nn2.predict(X_test)
        print(f'Testing accuracy: {accuracy_score(y_test, y_pred):f}')

        Accuracy of 2NN:
        Training accuracy: 0.834014
        Testing accuracy: 0.921687
```

Testing accuracy：0.92

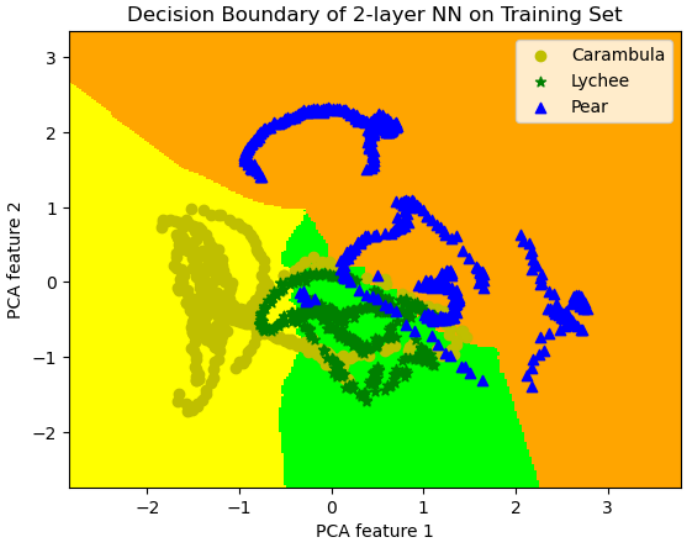
Plot training loss curves

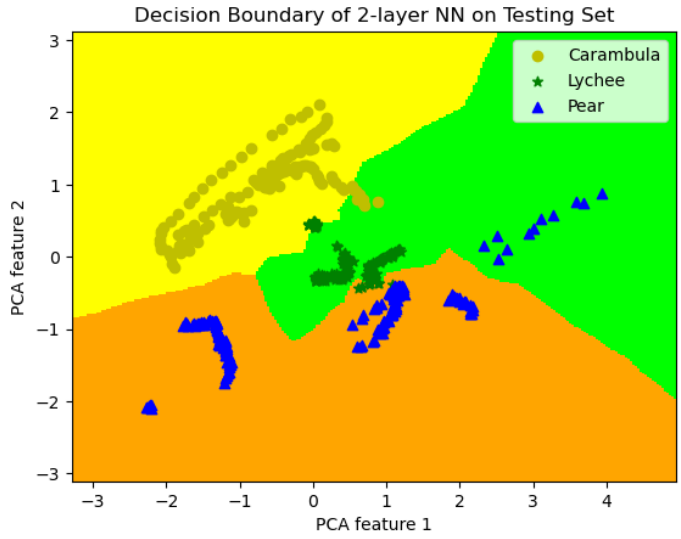
在做 training 階段時我們有記錄每個 epoch，從圖中可以看出大概在 40 個 epoch 後 loss 的下降幅度微乎其微，往後的 epoch 趨近於一水平線。



Plot decision regions

其中可以觀察到訓練集有部分重疊導致無法劃分區域。

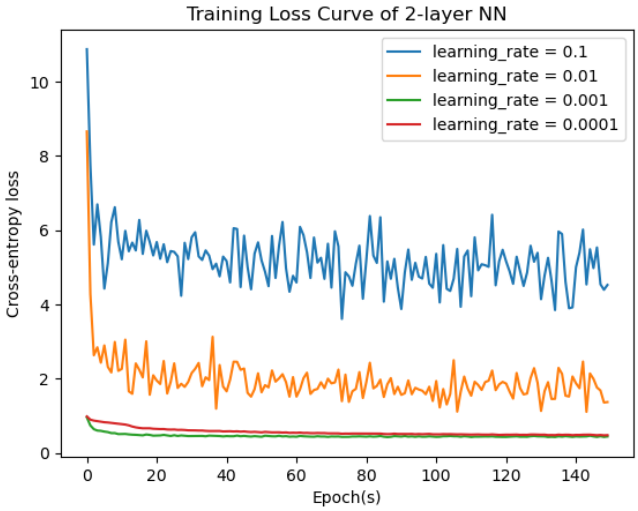




Discussion

我們使用不同的 learning_rate 大小去比較對 loss 的影響，從圖中可以發現，當 epoch 數量較少時，對於 loss 的影響較明顯，當 epoch

數量較多時，會發現 loss 會形成震盪的分布，而 learning_rate 越小越不受到影響。



Three-layer Neural Network

test accuracy

由結果可以發現 Testing accuracy 比 Training accuracy 還好，在 2NN 時也有這個現象，而 2 層比 3 層還好，推測為樣本的特徵數目過少，導致基於 2 層訓練就快接近飽和。

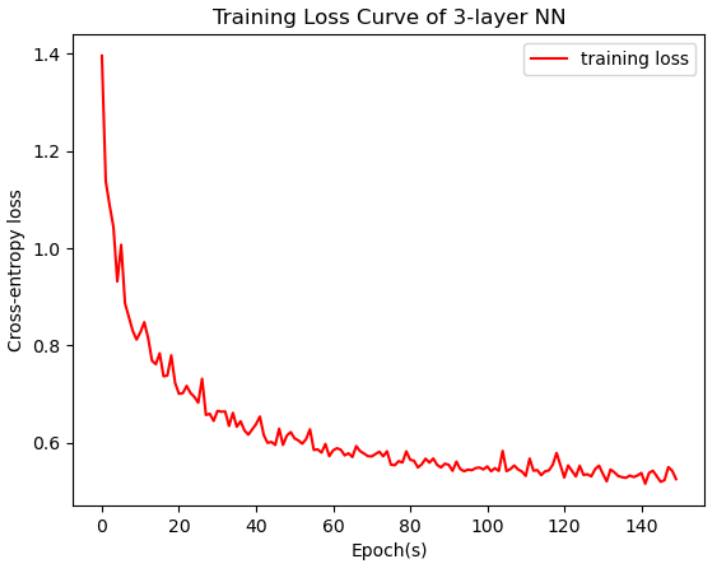
In [22]:
```
from sklearn.metrics import accuracy_score

nn3 = MYNN(random_seed=0, hidden_layer_sizes=(150, 100))
nn3.fit(X_train, y_train)

print('Accuracy of 3NN:')
y_pred = nn3.predict(X_train)
print(f'Training accuracy: {accuracy_score(y_train, y_pred):f}')

y_pred = nn3.predict(X_test)
print(f'Testing accuracy: {accuracy_score(y_test, y_pred):f}')
```

```
Accuracy of 3NN:
Training accuracy: 0.812245
Testing accuracy: 0.845382
```

Testing accuracy：0.84

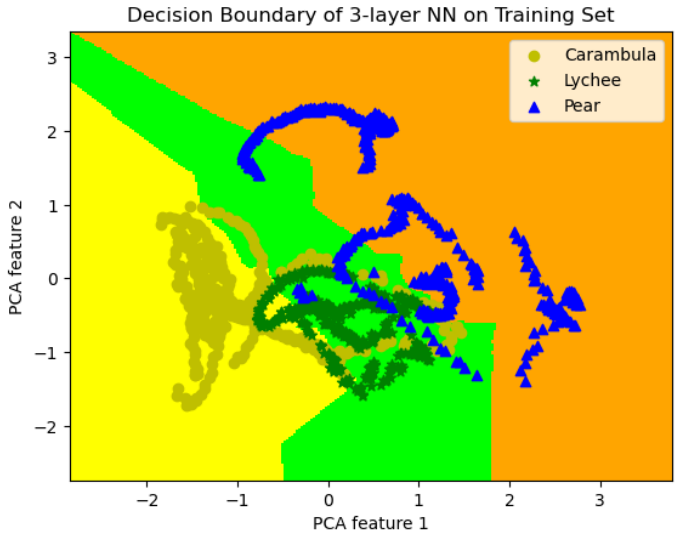
Plot training loss curves

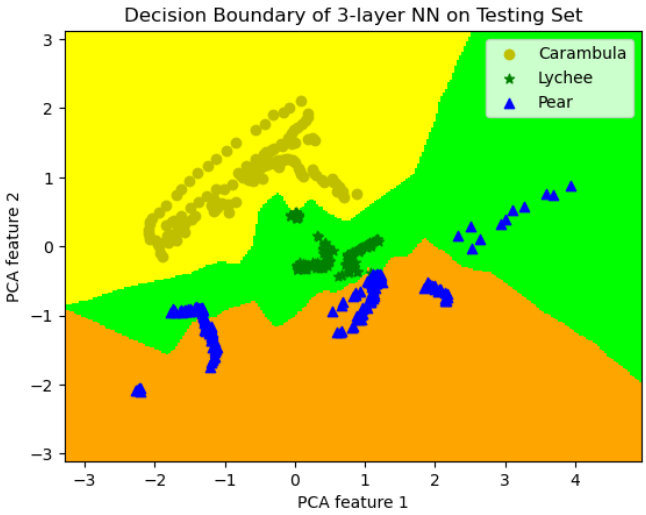
在做 training 階段時我們有記錄每個 epoch，從圖中可以看出大概在 80 個 epoch 後 loss 的下降幅度微乎其微，與 2NN 相比收斂速度較慢。



Plot decision regions

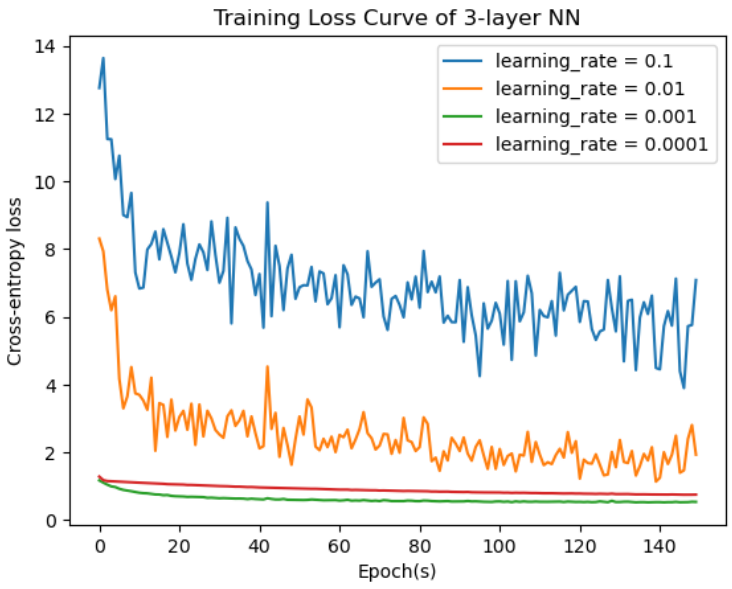
其中可以觀察到訓練集有部分重疊導致無法劃分區域。





Discussion

我們使用不同的 learning_rate 大小去比較對 loss 的影響，從圖中可以發現，當 epoch 數量較少時，對於 loss 的影響較明顯，當 epoch 數量較多時，會發現 loss 會形成震盪的分布，大致與 2NN 趨勢相同。



參考資料：

https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?hl=zh-tw
https://www.kaggle.com/code/arbazkhan971/image-classification-using-cnn-94-accuracy