第六組_DM_HW1

R07725049 吳姿君、R08725008 周若涓、R08725010 陳亦珊、R08725030 徐薇尹

一、研究動機與目的

這次的作業是對資料集降維的實作，我們希望以圖片來做降維，因為圖片經過降維後可以透過視覺來了解降維所造成的不同變化，能幫助我們更了解降維前後的差異。而我們想實作上課教的四種降維方式應用在水果的辨識，因為台灣是水果王國，身為水國王國的子民們，我們有義務分別每種水果。

二、資料描述

1. 資料來源

   Kaggle Dataset: Fruit Recognition
   https://www.kaggle.com/chrisfilo/fruit-recognition

2. 資料內容

   採用資料集中的 Apple(A&B&C)、Banana、Pitaya 資料夾的圖片作為本次作業的資料集。圖片數量分別為 Apple 2434 張、Banana 3027 張、Pitaya 2501 張，一共 7962 張圖片。

三、實驗方法

1. 資料集

   將資料集 Apple、Banana、Pitaya 分別拆分成前 80%為 training set，剩下 20% 為 testing set。

2. 降維方法

   以下分述我們實作的四種降維方法 DWT, SVD, NMF, Autoencode。

   2.1 DWT

   - 簡述

     將圖片資料相鄰的 row 和 column 各自兩兩相加，會得到壓縮 75%的圖片。

- 以下為壓縮前後比較：

Original: 480*322, 219kb

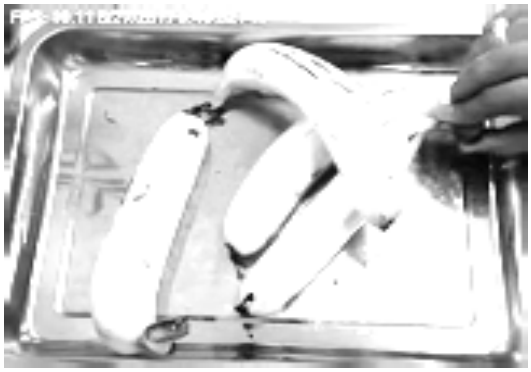
Original_gray: 480*322, 82.5kb

DWT-1: 242*163, 19.8kb

DWT-2: 123*84, 3.24kb

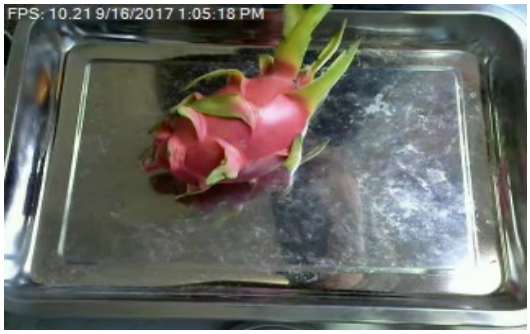
Original: 320*258, 132kb

Original_gray: 320*258, 50.5kb



DWT-1: 162*131, 11.6kb

DWT-2: 83*68, 1.57kb



Original: 320*258, 139kb

Original_gray: 320*258, 53.9kb

DWT-1: 162*131, 14.2kb

DWT-2: 83*68, 1.63kb

2.2 SVD:

(1) Singular Value Decomposition 奇異值分解：將矩陣 A 拆解為 $U \Sigma V^T$，其中 Σ 為奇異值矩陣，降維的原理是取前 k 個奇異值可以表達大部分的特徵值。

(2) 奇異值分解後取不同比例的奇異值，與原圖進行比較（以蘋果為例）。當原圖片從 k = 322 降維至 k = 31 時，圖片中的蘋果輪廓、細節依舊可以看得很清楚，而檔案大小從 225 KB 縮小至 22.9 KB。SVD 可以很有效的降維，也能根據需要選擇要保留的奇異值數量。

原圖　k = 322　225 KB　　k = 64　24.3 KB　　k = 31　22.9 KB

k = 17　21.3 KB　　k = 10　19.3 KB　　k = 6　17.9 KB

2.3 NMF:

(1) NMF decomposition 將原本的 matrix 拆成兩個 matrices，分別為 W 以及 H。

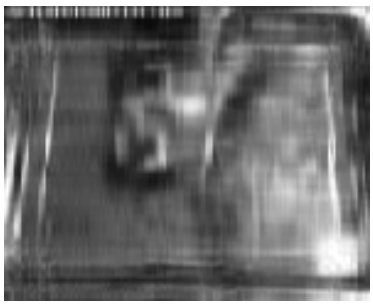
```
estimator = decomposition.NMF(n_components = n_components, init = 'random', tol=5e-3)
W = estimator.fit_transform(img)
H = estimator.components_
```

(2) 將 W 以及 H 做矩陣相乘，即可以得到降維後的 image

```
new_img = np.dot(W,H)
plt.imshow(new_img, cmap=plt.cm.gray,
                    interpolation='nearest',
                    vmin=-vmax, vmax=vmax)
```

(3) Original picture (139 KB)　　First reduction (56.5 KB)

Second reduction (42.8 KB)　Third reduction (37.0 KB)



2.4 Autoencoder:

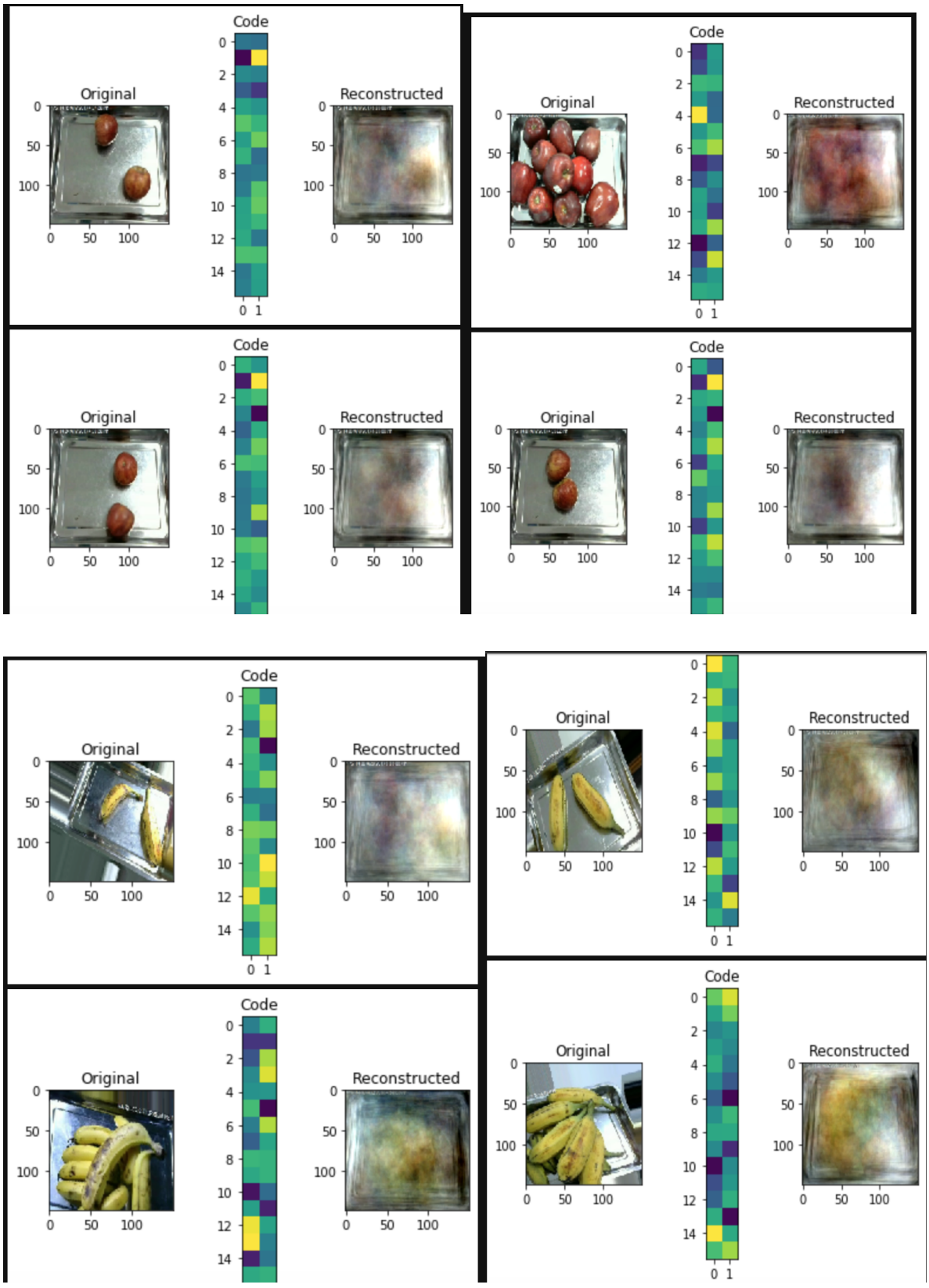
(1) 資料預處理:

將原本尺寸為 480x322 的資料及圖片，resize 至 150x150。並且將 pixel value 全部除以 255，使處理過後的 pixel value 介於 0~1 之間以便運算。

(2) 模型架構:

- Encoder: Input Layer(shape=150*150*3)-Flattening Layer-Dense(shape=32)
- Decoder: Input Layer(shape=32)-Dense(shape=32*32*3)-Reshape to 150*150*3
- optimizer='adamax'
- loss='mse'
- epochs=1000

(3) Original images v.s. Reconstructed images



四、實驗結果與分析

將前個步驟中透過 DWT, SVD, NMF, Autoencoder 四種方法降維後的圖片，使用 tensorflow keras 訓練模型，並比較訓練結果。

- Data Augmentation
  - rotation_range=20
  - width_shift_range=0.10
  - height_shift_range=0.10
  - rescale=1/255, # Rescale the image by normalzing it.
  - shear_range=0.1
  - zoom_range=0.1
  - horizontal_flip=True
- 模型架構
  - Layer

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 256, 256, 16)
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)
conv2d_1 (Conv2D)	(None, 128, 128, 32)
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 32)
conv2d_2 (Conv2D)	(None, 64, 64, 64)
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 64)
flatten (Flatten)	(None, 65536)
dense (Dense)	(None, 512)
dense_1 (Dense)	(None, 3)

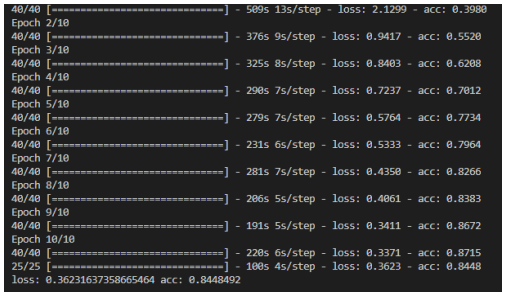
  - epoch=10

1. DWT
   a. Original

```
40/40 [==============================] - 432s 11s/step - loss: 1.3138 - acc: 0.6332
Epoch 2/10
40/40 [==============================] - 423s 11s/step - loss: 0.1502 - acc: 0.9520
Epoch 3/10
40/40 [==============================] - 460s 12s/step - loss: 0.0438 - acc: 0.9863
Epoch 4/10
40/40 [==============================] - 337s 8s/step - loss: 0.0167 - acc: 0.9937
Epoch 5/10
40/40 [==============================] - 397s 10s/step - loss: 0.0154 - acc: 0.9960
Epoch 6/10
40/40 [==============================] - 279s 7s/step - loss: 0.0139 - acc: 0.9965
Epoch 7/10
40/40 [==============================] - 239s 6s/step - loss: 0.0329 - acc: 0.9889
Epoch 8/10
40/40 [==============================] - 258s 6s/step - loss: 0.0150 - acc: 0.9965
Epoch 9/10
40/40 [==============================] - 245s 6s/step - loss: 0.0105 - acc: 0.9972
Epoch 10/10
40/40 [==============================] - 231s 6s/step - loss: 0.0109 - acc: 0.9957
25/25 [==============================] - 121s 5s/step - loss: 0.0249 - acc: 0.9906
loss: 0.024941166434437036 acc: 0.9905779
```

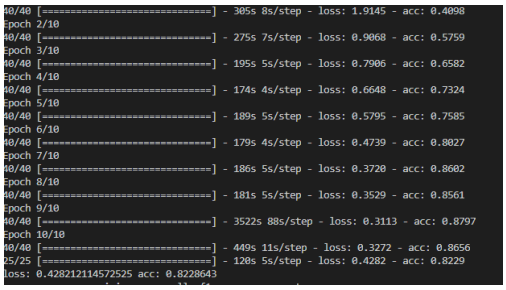
	precision	recall	f1-score	support
0	0.95	1.00	0.97	487
1	1.00	1.00	1.00	605
2	1.00	0.95	0.97	500
accuracy			0.98	1592
macro avg	0.98	0.98	0.98	1592
weighted avg	0.98	0.98	0.98	1592

   b. Original gray



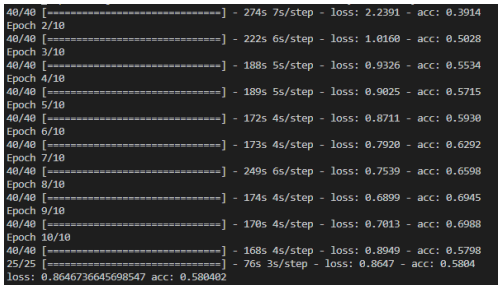
	precision	recall	f1-score	support
0	0.83	0.75	0.79	487
1	0.94	0.90	0.92	605
2	0.75	0.86	0.80	500
accuracy			0.84	1592
macro avg	0.84	0.84	0.84	1592
weighted avg	0.85	0.84	0.84	1592

c. 一次 DWT



	precision	recall	f1-score	support
0	0.74	0.88	0.80	487
1	0.93	0.83	0.88	605
2	0.81	0.77	0.79	500
accuracy			0.83	1592
macro avg	0.83	0.83	0.82	1592
weighted avg	0.84	0.83	0.83	1592

d. 兩次 DWT



	precision	recall	f1-score	support
0	0.48	0.48	0.48	487
1	0.80	0.61	0.70	605
2	0.46	0.59	0.52	500
accuracy			0.57	1592
macro avg	0.58	0.56	0.57	1592
weighted avg	0.60	0.57	0.57	1592

e. 分析

- 轉成灰階與降維後與假設基本上相符，失去的資訊越多 model 準確度越差。
- 在這些結果中有一明顯趨勢，除了原始圖片能輕易以顏色區分外，只能以形狀辨別的灰階圖片較辨認不出蘋果和火龍果；當資訊量越少越看不出火龍果的摺痕，這兩者的準確率也隨之下降，尤其第二次 DWT 後幾乎只是隨機判斷(約五成)。
- 本次實驗的分類器能將原始彩色圖片 99%分類正確，灰階也有 84%正確率。做一次 DWT 壓縮了 75%的資料量仍保有 83%準確率，結果相當不錯。觀察 precision 和 recall 發現降維後的 model

傾向減少判斷為蘋果，造成火龍果和蘋果互相消長，而香蕉幾乎不受影響因而保留了降維前的正確率。

- 然而壓縮第二次雖只剩下六成正確率，但壓縮了近 90%，相較其他降維方法也是相對高的準確率。推測因為 DWT 的降維資訊皆來自壓縮前的原始資訊組合，而非捨棄某些部分的資訊(如使用 inerpolation 計算或降低權重等)，故作結 DWT 在 image compression 上是相對能保留較高資訊量的壓縮方式。
- 相較前述，香蕉的辨識率始終不錯，因此如果類別形狀差異夠大，DWT 可以提供相當有效的壓縮率。

2. SVD

a. 當只保留前 10 % 的奇異值，降維至 k =31 時相較於原圖已降低了很多維度，但圖片的輪廓還很清楚，因此這邊選用 k =31 來訓練模型測試結果。並比較彩色圖與灰階圖的結果差異。

i. Original picture, k = 322 (原圖模型訓練結果與 DWT 相同)

```
40/40 [==============================] - 432s 11s/step - loss: 1.3138 - acc: 0.6332
Epoch 2/10
40/40 [==============================] - 423s 11s/step - loss: 0.1502 - acc: 0.9520
Epoch 3/10
40/40 [==============================] - 460s 12s/step - loss: 0.0438 - acc: 0.9863
Epoch 4/10
40/40 [==============================] - 337s 8s/step - loss: 0.0167 - acc: 0.9937
Epoch 5/10
40/40 [==============================] - 397s 10s/step - loss: 0.0154 - acc: 0.9960
Epoch 6/10
40/40 [==============================] - 279s 7s/step - loss: 0.0139 - acc: 0.9965
Epoch 7/10
40/40 [==============================] - 239s 6s/step - loss: 0.0329 - acc: 0.9889
Epoch 8/10
40/40 [==============================] - 258s 6s/step - loss: 0.0150 - acc: 0.9965
Epoch 9/10
40/40 [==============================] - 245s 6s/step - loss: 0.0105 - acc: 0.9972
Epoch 10/10
40/40 [==============================] - 231s 6s/step - loss: 0.0109 - acc: 0.9957
25/25 [==============================] - 121s 5s/step - loss: 0.0249 - acc: 0.9906
loss: 0.024941166434437036 acc: 0.9905779
```

	precision	recall	f1-score	support
0	0.95	1.00	0.97	487
1	1.00	1.00	1.00	605
2	1.00	0.95	0.97	500
accuracy			0.98	1592
macro avg	0.98	0.98	0.98	1592
weighted avg	0.98	0.98	0.98	1592

ii. k = 31, 彩色圖 (22.9 KB)

```
Epoch 1/10
40/40 [==============================] - 128s 3s/step - loss: 1.1481 - accuracy: 0.5124
Epoch 2/10
40/40 [==============================] - 134s 3s/step - loss: 0.3233 - accuracy: 0.8783
Epoch 3/10
40/40 [==============================] - 134s 3s/step - loss: 0.0892 - accuracy: 0.9735
Epoch 4/10
40/40 [==============================] - 132s 3s/step - loss: 0.0481 - accuracy: 0.9855
Epoch 5/10
40/40 [==============================] - 127s 3s/step - loss: 0.0311 - accuracy: 0.9898
Epoch 6/10
40/40 [==============================] - 134s 3s/step - loss: 0.0275 - accuracy: 0.9910
Epoch 7/10
40/40 [==============================] - 129s 3s/step - loss: 0.0237 - accuracy: 0.9925
Epoch 8/10
40/40 [==============================] - 129s 3s/step - loss: 0.0342 - accuracy: 0.9905
Epoch 9/10
40/40 [==============================] - 125s 3s/step - loss: 0.0267 - accuracy: 0.9910
Epoch 10/10
40/40 [==============================] - 132s 3s/step - loss: 0.0228 - accuracy: 0.9941
```

	precision	recall	f1-score	support
0	0.94	1.00	0.97	486
1	1.00	1.00	1.00	605
2	0.99	0.94	0.96	500
accuracy			0.98	1591
macro avg	0.98	0.98	0.98	1591
weighted avg	0.98	0.98	0.98	1591

iii. k = 31, 灰階圖 (12.3 KB)

```
Epoch 1/10
40/40 [==============================] - 128s 3s/step - loss: 1.1960 - accuracy: 0.4062
Epoch 2/10
40/40 [==============================] - 123s 3s/step - loss: 0.8862 - accuracy: 0.5918
Epoch 3/10
40/40 [==============================] - 124s 3s/step - loss: 0.8102 - accuracy: 0.6420
Epoch 4/10
40/40 [==============================] - 121s 3s/step - loss: 0.6782 - accuracy: 0.7112
Epoch 5/10
40/40 [==============================] - 123s 3s/step - loss: 0.5531 - accuracy: 0.7695
Epoch 6/10
40/40 [==============================] - 137s 3s/step - loss: 0.4790 - accuracy: 0.8062
Epoch 7/10
40/40 [==============================] - 127s 3s/step - loss: 0.4247 - accuracy: 0.8270
Epoch 8/10
40/40 [==============================] - 125s 3s/step - loss: 0.4132 - accuracy: 0.8395
Epoch 9/10
40/40 [==============================] - 150s 4s/step - loss: 0.3179 - accuracy: 0.8728
Epoch 10/10
40/40 [==============================] - 126s 3s/step - loss: 0.3380 - accuracy: 0.8652
```

	precision	recall	f1-score	support
0	0.89	0.90	0.89	486
1	0.96	0.73	0.83	605
2	0.75	0.96	0.84	500
accuracy			0.85	1591
macro avg	0.87	0.86	0.86	1591
weighted avg	0.87	0.85	0.85	1591

b. 結果分析

i. 比較 k = 31 的彩色圖訓練結果準確度很高，雖然維度降了很多，圖片大小也與原圖相差了大約十倍，但訓練結果卻與原圖訓練沒有太大的差異，結果依舊非常好，accuracy 皆達 0.99，precision 亦達 0.98。可以看出透過 SVD 取合適奇異值個數降維可以達到很不錯的效果。

ii. 當 k =31 的彩色圖轉為灰階後再拿去訓練，雖然圖片大小又再縮小了二分之一左右，但訓練結果與彩色圖、原圖相比差距就比較大。 accuracy 約在 0.86，precision 為 0.87，整體而言結果也不算太差。

3. NMF (Take Pitaya picture as example)

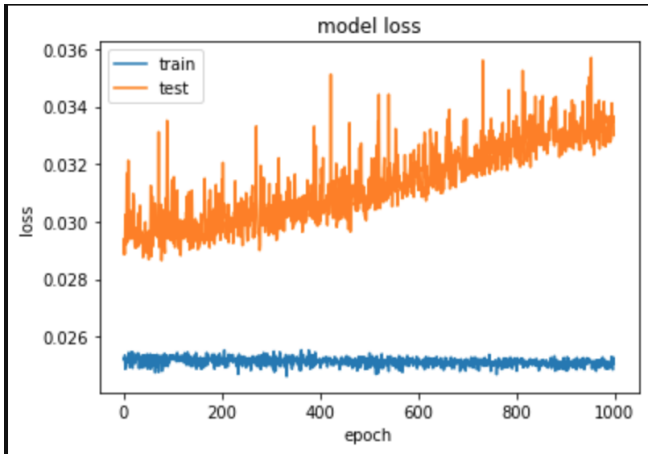
a. Result of classifier

```
Train for 40 steps
Epoch 1/10
40/40 [==============================] - 63s 2s/step - loss: 1.3813 - accuracy: 0.4855
Epoch 2/10
40/40 [==============================] - 62s 2s/step - loss: 0.3090 - accuracy: 0.8746
Epoch 3/10
40/40 [==============================] - 64s 2s/step - loss: 0.0984 - accuracy: 0.9730
Epoch 4/10
40/40 [==============================] - 67s 2s/step - loss: 0.1050 - accuracy: 0.9605
Epoch 5/10
40/40 [==============================] - 69s 2s/step - loss: 0.0259 - accuracy: 0.9944
Epoch 6/10
40/40 [==============================] - 67s 2s/step - loss: 0.0330 - accuracy: 0.9909
Epoch 7/10
40/40 [==============================] - 68s 2s/step - loss: 0.0256 - accuracy: 0.9945
Epoch 8/10
40/40 [==============================] - 68s 2s/step - loss: 0.0285 - accuracy: 0.9922
Epoch 9/10
40/40 [==============================] - 69s 2s/step - loss: 0.0103 - accuracy: 0.9976
Epoch 10/10
40/40 [==============================] - 66s 2s/step - loss: 0.0100 - accuracy: 0.9969
WARNING:tensorflow:From C:/Users/user/Data mining#1/classifier.py:65: Model.evaluate_generator (from
tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.evaluate, which supports generators.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
125/125 [==============================] - 124s 994ms/step - loss: 2.2217 - accuracy: 0.3057
loss: 2.2216819887161257 acc: 0.3057021
    predictions_b:  [0.88747436 0.06134693 0.0511787 ]
    predictions_p: [0.7955717  0.1384983  0.06593002]
```

4. Autoencoder
    a. Training Phase

```
Epoch 992/1000
4507/4507 [==============================] - 20s 4ms/step - loss: 0.0250 - accuracy: 0.6017 - val_loss: 0.03
27 - val_accuracy: 0.5861
Epoch 993/1000
4507/4507 [==============================] - 18s 4ms/step - loss: 0.0249 - accuracy: 0.6011 - val_loss: 0.03
35 - val_accuracy: 0.5500
Epoch 994/1000
4507/4507 [==============================] - 19s 4ms/step - loss: 0.0250 - accuracy: 0.6012 - val_loss: 0.03
28 - val_accuracy: 0.5202
Epoch 995/1000
4507/4507 [==============================] - 19s 4ms/step - loss: 0.0253 - accuracy: 0.5985 - val_loss: 0.03
37 - val_accuracy: 0.5619
Epoch 996/1000
4507/4507 [==============================] - 19s 4ms/step - loss: 0.0251 - accuracy: 0.6014 - val_loss: 0.03
41 - val_accuracy: 0.5457
Epoch 997/1000
4507/4507 [==============================] - 29s 6ms/step - loss: 0.0249 - accuracy: 0.6012 - val_loss: 0.03
34 - val_accuracy: 0.5135
Epoch 998/1000
4507/4507 [==============================] - 29s 6ms/step - loss: 0.0248 - accuracy: 0.6034 - val_loss: 0.03
35 - val_accuracy: 0.5489
Epoch 999/1000
4507/4507 [==============================] - 23s 5ms/step - loss: 0.0253 - accuracy: 0.6002 - val_loss: 0.03
37 - val_accuracy: 0.5313
Epoch 1000/1000
4507/4507 [==============================] - 20s 4ms/step - loss: 0.0251 - accuracy: 0.6000 - val_loss: 0.03
30 - val_accuracy: 0.5182
```

    b. Test loss: 0.03299946734618865
    c. Test accuracy: 0.5182107090950012
    d. Model Loss



    e. 分析與小結

    可以明顯看出與前面三個方法相比，模型學習的效果十分不好，Overall accuracy on training set 僅有 0.6，Testing set accuracy 約為 0.52。我們認為這可能與 Autoencoder **高度依賴訓練集生成圖片的特性**有關，由於 training set 和 testing set 中皆含有一些 noise，像是有異物(例如人類的手)入鏡，或是拍攝不完全的水果樣貌、晃動鏡頭造成的模糊影像等。受限於時間與人力，在資料清理的部分較無法做得比較仔細透徹，導致 training set 和 testing set 相似度較不高、構圖也較不一致，進而造成 Autoencoder 在這樣的 dataset 上面表現得比較不出色。