智慧化企業整合

深度學習為基礎之顧客情緒及商品喜好程度分析

Project 2

指導教授：邱銘傳　教授

第 6 組

110034543　戴佩怡

110034554　徐一正

110034557　方際勛

110034559　劉兆原

中 華 民 國 一 一 〇 年 十 二 月 十 七 日

目錄

一、背景介紹......1
1.1 問題背景......1
1.2 動機與目的......1
1.3 問題描述(5W1H)......1
1.4 資料來源......2
二、資料前處理......3
2.1 資料增強......3
2.2 DCGAN......3
三、深度學習模型......8
3.1 VGG16......8
3.2 模型總結......10
四、分析......13
4.1 實驗設計......13
4.2 L_9 直交表......13
4.3 結論......14
五、結果與討論......14
5.1 結論......14
5.2 未來展望......15

圖目錄

圖 1、情緒數量分布......2
圖 2、資料增強程式碼......3
圖 3、DCGAN 結構......4
圖 4、Generator 程式碼......4
圖 5、Discriminator 程式碼......5
圖 6、happy 之 Training Loss......6
圖 7、disgust 之 Training Loss......6
圖 8、fear 之 Training Loss......6
圖 9、anger 之 Training Loss......6
圖 10、DCGAN 模型之超參數......7
圖 11、資料增強前之結果......7
圖 12、資料增強後之結果......8
圖 13、VGG16 架構......9
圖 14、VGG16 之程式碼 1......9
圖 15、VGG16 之程式碼 2......10
圖 16、VGG16 之程式碼 3......10
圖 17、VGG16 之程式碼 4......10
圖 18、VGG16 原執行結果 1......11
圖 19、VGG16 原執行結果 2......11
圖 20、VGG16 執行原資料集......11
圖 21、VGG16 新執行結果 1......12
圖 22、VGG16 新執行結果 2......12
圖 23、VGG16 執行新資料集......12
圖 24、程式執行結果 1......14

圖 25、程式執行結果 2..14

表目錄

表 1、5W1H......2

表 2、三資料集比例與數量......3

表 3、VGG16 原資料集準確率......11

表 4、VGG16 新資料集準確率......12

表 5、實驗設計表格......13

表 6、L9 直交表......13

表 7、程式資料集準確率......14

一、背景介紹

1.1 問題背景

1995 年，美國麻省理工學院教授 Rosalind Picard 發表情緒運算（Affective Computing)相關研究，開創了計算機科學之重要分之；到了 2018 年，MarketWatch 之數據顯示該年全球之情緒識別產品市值已達到 123.7 億美元，且預計到 2024 年，此情緒辨識之市場將達到 916.7 億美元，五年間之複合年增長率為 40.46%；到了 2019 年，The Guardian 報導指出，「AI 臉部情緒辨識」已成為規模 200 億美元（約台幣 6,210 億元）的業產，且還在擴充。以上各事件說明了「情緒辨識」為一潛力之產業，且預計在未來擁有龐大商機。

1.2 動機與目的

企業欲瞭解顧客意見往往使用問卷調查等方式，這些方法不僅耗費時間，且顧客可能因有所顧慮而未能反應其真實想法。為解決上述問題，本小組應用 AI 辨識臉部情緒之技術以瞭解商品對顧客的吸引程度及顧客購買意願，作為產品定位、改善之參考。

1.3 問題描述(5W1H)

為釐清顧客情緒及商品喜好程度分析之問題，本小組以 5W1H 思考，進而深入了解此問題之目的、時間、人員、地點、事件、方法。透過 5W1H 分析，我們了解到此問題之目的為使企業欲調查顧客意見費時費力，且顧客未能反應真實情緒及產品評價；而此問題之時間為民眾逛街或選購產品時；而此問題之人員為企業行銷部門或欲購物之民眾；而此問題之地點為產品銷售處、貨架區、結帳等候線…等處；而此問題之事件為以顧客臉部表情判斷其對產品之喜好程度；而此問題之方法為運用 DCGAN 進行 Data Augmentation，並以 VGG16 辨識臉部表情。

本小組將 5W1H 分析內容整理如表 1 所示：

表 1、5W1H

Why	企業欲調查顧客意見費時費力，且顧客未能反應真實情緒及產品評價
When	民眾逛街、選購產品時
Who	企業行銷部門、欲購物之民眾
Where	產品銷售處、貨架區、結帳等候線
What	以顧客臉部表情判斷其對產品之喜好程度
How	運用 DCGAN 進行 Data Augmentation，並以 VGG16 辨識臉部表情

1.4 資料來源

本小組於 Kaggle 下載 CK-48 資料集，其中包括開心、驚喜、嫌棄、無感、…等五種情緒之表情圖片。本資料集共有 4442 張表情圖片，而資料集五種情緒之數量分布如圖 1 所示：

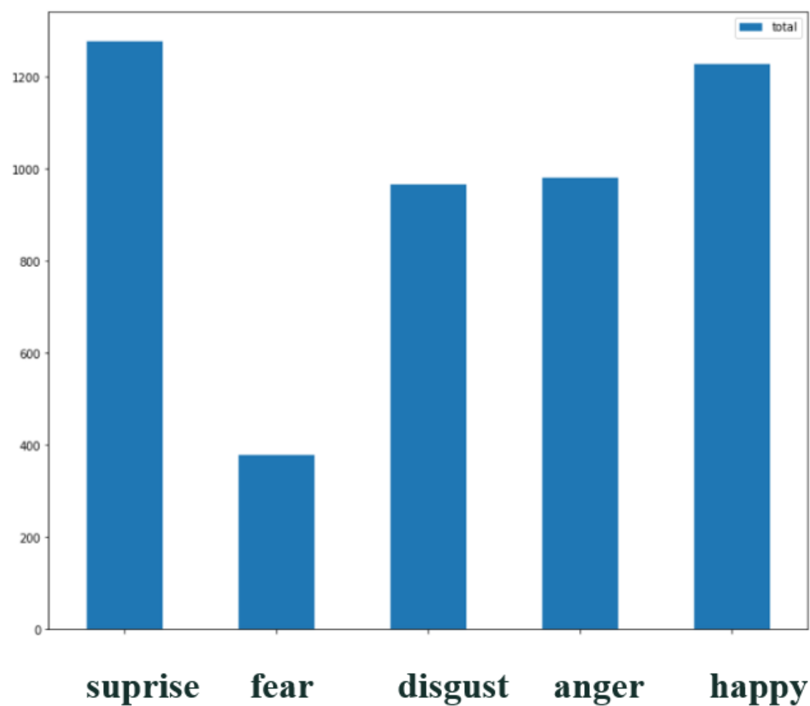

圖 1、情緒數量分布

二、資料前處理

2.1 資料增強

本小組運用之程式碼會將資料集之圖片進行旋轉、縮放、翻轉與移動，以利模型學習。使用之程式碼如圖 2 所示：

```
# Data Augmentation
train_datagen = ImageDataGenerator(
                rotation_range=5,    # randomly rotate images in the range 5 degrees
                zoom_range = 0.1,  # Randomly zoom image 10%
                width_shift_range=0.1,    # randomly shift images horizontally 10%
                height_shift_range=0.1,    # randomly shift images vertically 10%
                horizontal_flip=True,    # randomly flip images
                vertical_flip=True)    # randomly flip images
```

圖 2、資料增強程式碼

本小組也將原資料集進行資料切割，分為訓練集、驗證集、測試集，三資料集圖片之數量比例為 80%、10%、10%。三資料集之與相對比例與圖片數量以表 2 所示：

表 2、三資料集比例與數量

項目 / 資料集	比例	數量
Train set	80%	3698
Val set	10%	372
Test set	10%	372

2.2 DCGAN

本小組使用之資料前處理技術為 DCGAN（Deep Convolutional Generative Adversarial Networks），其模型結構如圖 3 所示：

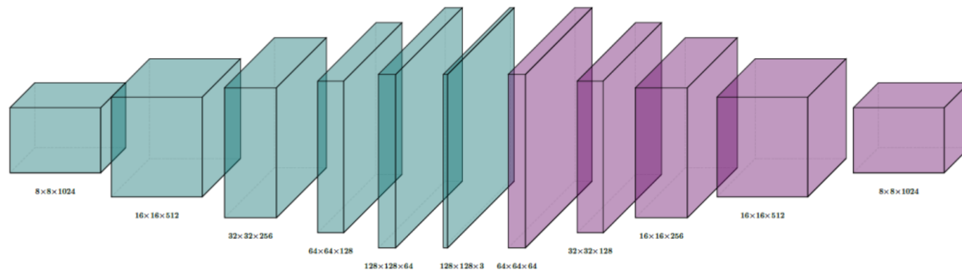

圖 3、DCGAN 結構

DCGAN 之結構可分為 Generator 與 Discriminator，其中 Generator 為圖三中綠色之結構，而 Discriminator 為圖三中紫色之結構。Generator 之程式碼以圖 4 所示，而 Discriminator 之程式碼以圖 5 所示。

```
def generator(z, output_channel_dim, training):
    with tf.variable_scope("generator", reuse= not training):

        # 8x8x1024
        fully_connected = tf.layers.dense(z, 8*8*1024)
        fully_connected = tf.reshape(fully_connected, (-1, 8, 8, 1024))
        fully_connected = tf.nn.leaky_relu(fully_connected)

        # 8x8x1024 -> 16x16x512
        trans_conv1 = tf.layers.conv2d_transpose(inputs=fully_connected, filters=512, kernel_size=[5,5], strides=[2,2], padding="SAME", kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV), name="trans_conv1")
        batch_trans_conv1 = tf.layers.batch_normalization(inputs = trans_conv1, training=training, epsilon=EPSILON, name="batch_trans_conv1")
        trans_conv1_out = tf.nn.leaky_relu(batch_trans_conv1, name="trans_conv1_out")

        # 16x16x512 -> 32x32x256
        trans_conv2 = tf.layers.conv2d_transpose(inputs=trans_conv1_out, filters=256, kernel_size=[5,5], strides=[2,2], padding="SAME", kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV), name="trans_conv2")
        batch_trans_conv2 = tf.layers.batch_normalization(inputs = trans_conv2, training=training, epsilon=EPSILON, name="batch_trans_conv2")
        trans_conv2_out = tf.nn.leaky_relu(batch_trans_conv2, name="trans_conv2_out")

        # 32x32x256 -> 64x64x128
        trans_conv3 = tf.layers.conv2d_transpose(inputs=trans_conv2_out, filters=128, kernel_size=[5,5],
                                                 strides=[2,2],
                                                 padding="SAME",
                                                 kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                                 name="trans_conv3")
        batch_trans_conv3 = tf.layers.batch_normalization(inputs = trans_conv3,
                                                          training=training,
                                                          epsilon=EPSILON,
                                                          name="batch_trans_conv3")
        trans_conv3_out = tf.nn.leaky_relu(batch_trans_conv3,
                                           name="trans_conv3_out")

        # 64x64x128 -> 128x128x64
        trans_conv4 = tf.layers.conv2d_transpose(inputs=trans_conv3_out,
                                                 filters=64,
                                                 kernel_size=[5,5],
                                                 strides=[2,2],
                                                 padding="SAME",
                                                 kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                                 name="trans_conv4")
        batch_trans_conv4 = tf.layers.batch_normalization(inputs = trans_conv4,
                                                          training=training,
                                                          epsilon=EPSILON,
                                                          name="batch_trans_conv4")
        trans_conv4_out = tf.nn.leaky_relu(batch_trans_conv4,
                                           name="trans_conv4_out")

        # 128x128x64 -> 128x128x3
        logits = tf.layers.conv2d_transpose(inputs=trans_conv4_out,
                                            filters=3,
                                            kernel_size=[5,5],
                                            strides=[1,1],
                                            padding="SAME",
                                            kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                            name="logits")
        out = tf.tanh(logits, name="out")
        return outS
```

圖 4、Generator 程式碼

```
def discriminator(x, reuse):
    with tf.variable_scope("discriminator", reuse=reuse):

        # 128*128*3 -> 64x64x64
        conv1 = tf.layers.conv2d(inputs=x,filters=64, kernel_size=[5,5], strides=[2,2], padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                 name='conv1')
        batch_norm1 = tf.layers.batch_normalization(conv1, training=True,epsilon=EPSILON,name='batch_norm1')
        conv1_out = tf.nn.leaky_relu(batch_norm1, name="conv1_out")

        # 64x64x64-> 32x32x128
        conv2 = tf.layers.conv2d(inputs=conv1_out,filters=128,kernel_size=[5, 5],strides=[2, 2],padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                 name='conv2')
        batch_norm2 = tf.layers.batch_normalization(conv2,training=True,epsilon=EPSILON,name='batch_norm2')
        conv2_out = tf.nn.leaky_relu(batch_norm2,name="conv2_out")

        # 32x32x128 -> 16x16x256
        conv3 = tf.layers.conv2d(inputs=conv2_out,filters=256,kernel_size=[5, 5],strides=[2, 2],padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                 name='conv3')
        batch_norm3 = tf.layers.batch_normalization(conv3,training=True,epsilon=EPSILON,name='batch_norm3')
        conv3_out = tf.nn.leaky_relu(batch_norm3,name="conv3_out")

        # 16x16x256 -> 16x16x512
        conv4 = tf.layers.conv2d(inputs=conv3_out,filters=512,kernel_size=[5, 5],strides=[1, 1],padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                 name='conv4')
        batch_norm4 = tf.layers.batch_normalization(conv4,training=True,epsilon=EPSILON,name='batch_norm4')
        conv4_out = tf.nn.leaky_relu(batch_norm4,name="conv4_out")

        # 16x16x512 -> 8x8x1024
        conv5 = tf.layers.conv2d(inputs=conv4_out,filters=1024,kernel_size=[5, 5],strides=[2, 2],padding="SAME",
                                kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                name='conv5')
        batch_norm5 = tf.layers.batch_normalization(conv5,training=True,epsilon=EPSILON,name='batch_norm5')
        conv5_out = tf.nn.leaky_relu(batch_norm5,name="conv5_out")

        flatten = tf.reshape(conv5_out, (-1, 8*8*1024))
        logits = tf.layers.dense(inputs=flatten,
                                 units=1,
                                 activation=None)
        out = tf.sigmoid(logits)
        return out, logits
```

圖 5、Discriminator 程式碼

資料增強後，各情緒之 Training Loss 以圖 6 至圖 9 所示，而 DCGAN 模型之超參數如圖 10 所示。

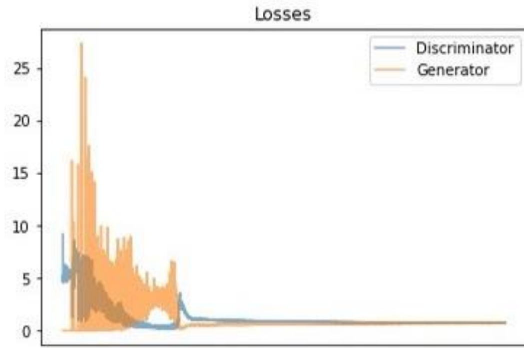

圖 6、Happy 之 Training Loss

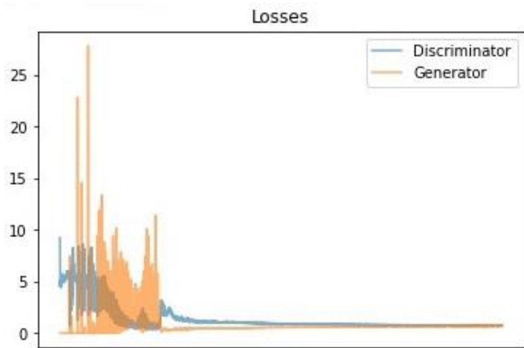

圖 7、Disgust 之 Training Loss

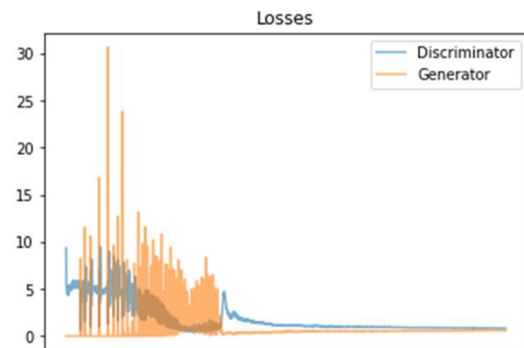

圖 8、Fear 之 Training Loss

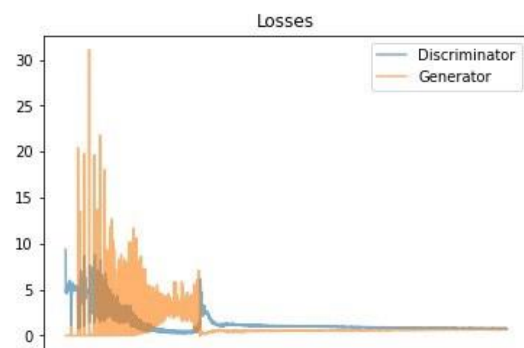

圖 9、Anger 之 Training Loss

```
#  Hyperparameters
IMAGE_SIZE  =  128
NOISE_SIZE  =  100
LR_D  =  0.00001
LR_G  =  0.0004
BATCH_SIZE  =  64
EPOCHS  =  500
BETA1  =  0.5
WEIGHT_INIT_STDDEV  =  0.02
EPSILON  =  0.00005
```

圖 10、DCGAN 模型之超參數

資料增強前後之結果如圖 11 與圖 12 所示：

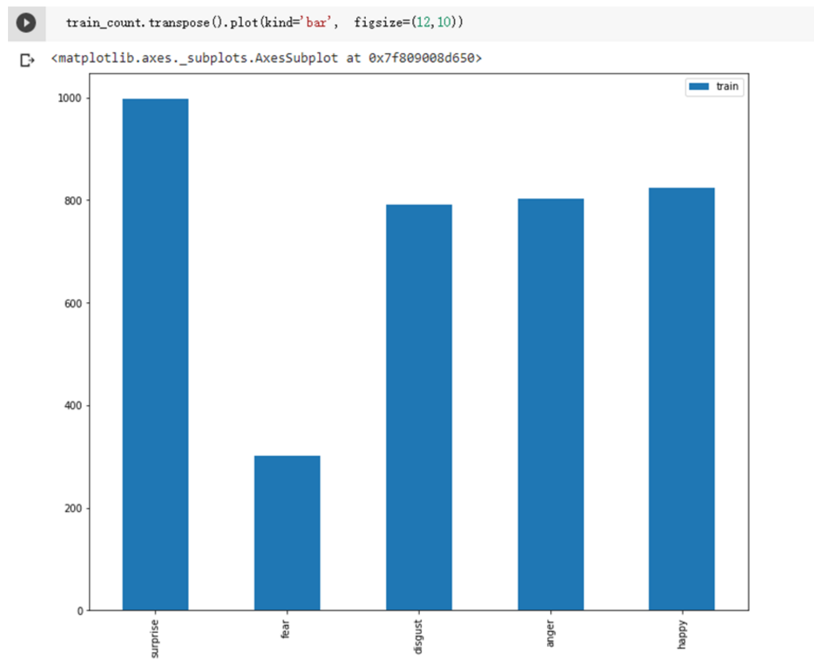

圖 11、資料增強前之結果

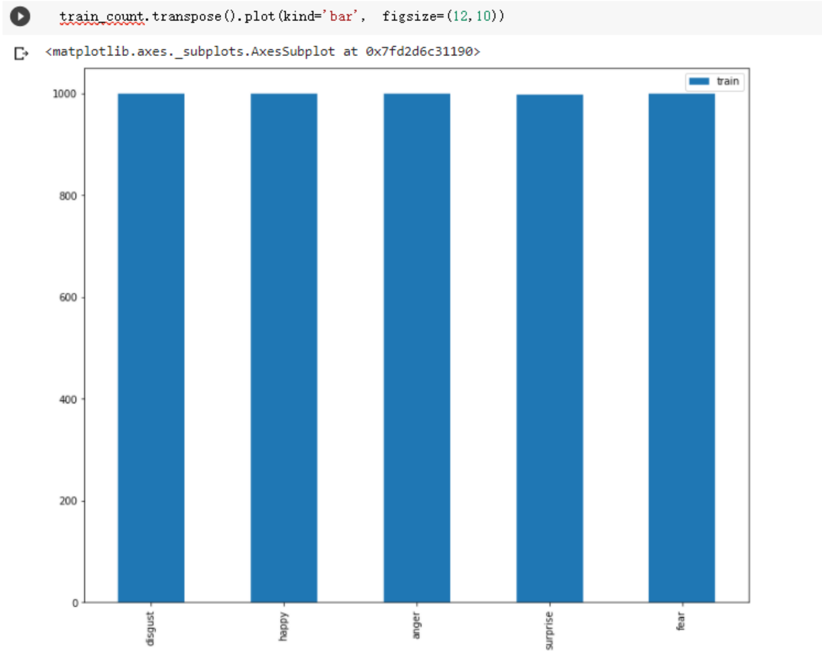

圖 12、資料增強後之結果

GAN 生成器生成之圖片

三、深度學習模型

3.1 VGG16

本小組使用之深度學習模型為 VGG16，由 13 層卷積層與 3 層全連接層組成，其模型架構如圖 13 所示：

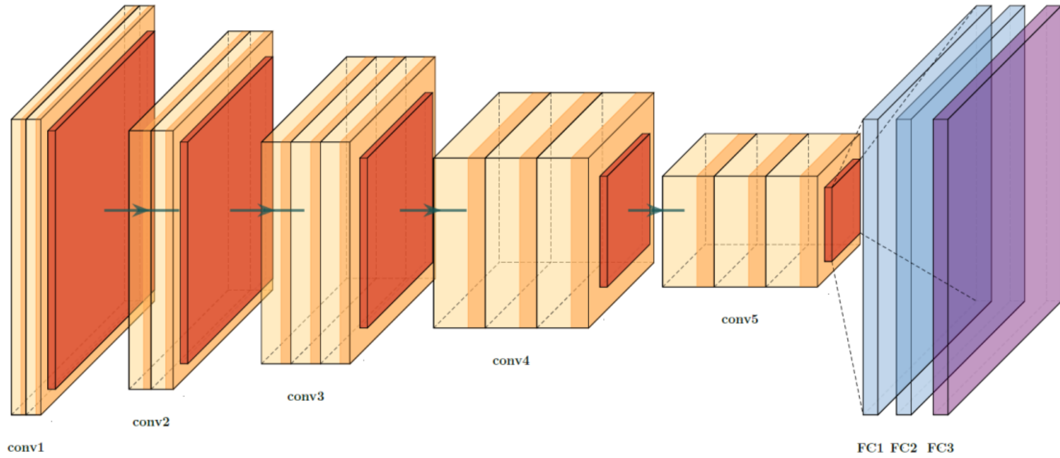

圖 13、VGG16 架構

VGG16 之程式碼以圖 14 至圖 17 所示：

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 48, 48, 3)]       0

 block1_conv1 (Conv2D)       (None, 48, 48, 64)        1792

 block1_conv2 (Conv2D)       (None, 48, 48, 64)        36928

 block1_pool (MaxPooling2D)  (None, 24, 24, 64)        0

 block2_conv1 (Conv2D)       (None, 24, 24, 128)       73856

 block2_conv2 (Conv2D)       (None, 24, 24, 128)       147584

 block2_pool (MaxPooling2D)  (None, 12, 12, 128)       0

 block3_conv1 (Conv2D)       (None, 12, 12, 256)       295168

 block3_conv2 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv3 (Conv2D)       (None, 12, 12, 256)       590080

 block3_pool (MaxPooling2D)  (None, 6, 6, 256)         0

 block4_conv1 (Conv2D)       (None, 6, 6, 512)         1180160

 block4_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 3, 3, 512)         0

 block5_conv1 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv2 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv3 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

圖 14、VGG16 之程式碼 1

```
model = models.Sequential()

model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(4096, activation='relu', name='FC1'))
model.add(layers.Dense(4096, activation='relu', name='FC2'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1000, activation='softmax', name='predictions'))


model.summary()
```

圖 15、VGG16 之程式碼 2

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten_3 (Flatten)         (None, 512)               0

 FC1 (Dense)                 (None, 4096)              2101248

 FC2 (Dense)                 (None, 4096)              16781312

 dropout_3 (Dropout)         (None, 4096)              0

 predictions (Dense)         (None, 5)                 20485

=================================================================
```

圖 16、VGG16 之程式碼 3

```
lrd = ReduceLROnPlateau(monitor = 'val_loss',patience = 20
                        ,verbose = 1,factor = 0.50, min_lr = 1e-10)
```

圖 17、VGG16 之程式碼 4

使用 ReduceLROnPlateau 之情形為當模型連續訓練 n 次沒有更好，就進行一次學習率衰降。

3.2 模型總結

VGG16 執行原資料集之結果如圖 18 至圖 20 所示，而訓練、驗證、測試之準確率如表 3 所示。

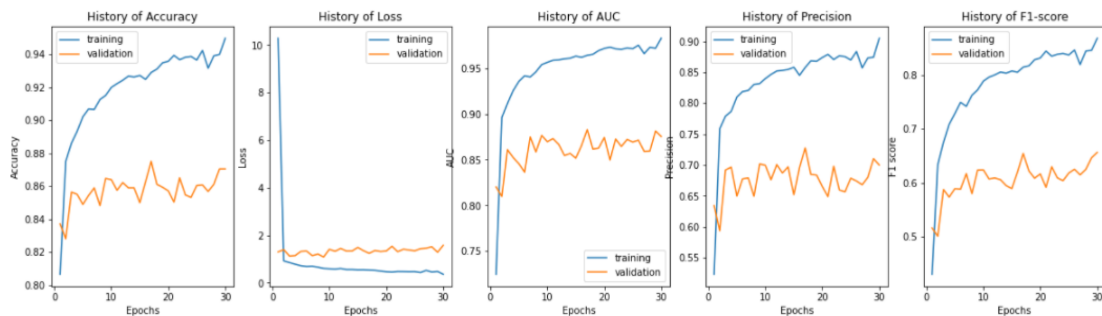

圖 18、VGG16 原執行結果 1

```
test_datagen   = ImageDataGenerator()

test_generator  = test_datagen.flow_from_directory(test_dir,  batch_size=64,
                                                   target_size=(48,  48),   class_mode='categorical')
model.evaluate(test_generator,  use_multiprocessing=True)

Found 557 images belonging to 5 classes.
9/9 [==============================] - 71s 9s/step - loss: 1.3014 - accuracy: 0.8639 - precision: 0.6824 - recall: 0.5978 - auc: 0.8699 - f1_score: 0.6377
```

圖 19、VGG16 原執行結果 2

```
train_dir  =  '/content/drive/MyDrive/project2/CK-48-5/train/'
test_dir  =  '/content/drive/MyDrive/project2/CK-48-5/test/'
val_dir  =  '//content/drive/MyDrive/project2/CK-48-5/val/'
row,  col  =  48,  48
classes  =  3
```

圖 20、VGG16 執行原資料集

表 3、VGG16 原資料集準確率

準確率 / 資料集	準確率
Train	0.9497
Val	0.8704
Test	0.8639

VGG16 執行新資料集之結果如圖 21 至圖 23 所示，而訓練、驗證、測試之準確率如表 4 所示。

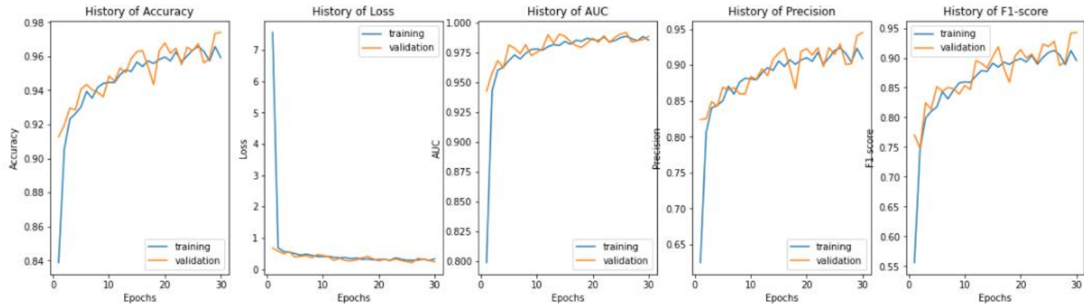

圖 21、VGG16 新執行結果 1

```
test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow_from_directory(test_dir, batch_size=64, target_size=(48, 48), class_mode='categorical')
model.evaluate(test_generator, use_multiprocessing=True)
#print('Test loss:', score[0])
#print('Test accuracy:', score[1])

Found 372 images belonging to 5 classes.
6/6 [==============================] - 48s 9s/step - loss: 1.0589 - accuracy: 0.9274 - precision: 0.8319 - recall: 0.7984 - auc: 0.9272 - f1_score: 0.8130
```

圖 22、VGG16 新執行結果 2

```
train_dir = '/content/drive/MyDrive/CK-48-AUG/train/'
test_dir = '/content/drive/MyDrive/CK-48-AUG/test/'
val_dir = '/content/drive/MyDrive/CK-48-AUG/val/'
row, col = 48, 48
classes = 3
```

圖 23、VGG16 執行新資料集

表 4、VGG16 新資料集準確率

準確率 / 資料集	準確率
Train	0.9561
Val	0.9566
Test	0.9274

四、分析

4.1 實驗設計

本小組設計 4 因子 3 水準之實驗設計，其中選定之因子為 Optimizer、Epoch、Dropout、Batch Size。實驗設計之表格如表 5 所示：

表 5、實驗設計表格

Factor / Level	Optimizer	Epoch	Dropout	Batch Size
Level 1	Adam	30	0.4	32
Level 2	AdaGrad	40	0.5	64
Level 3	Adadelta	50	0.6	128

4.2 L_9 直交表

實驗設計之結果以表 6 之 L_9 直交表所示，最終由於第 3 次實驗之準確率最高，因此依照第 3 次實驗以設定程式之因子。

表 6、L9 直交表

Factor / Experimant	Optimizer	Epoch	Dropout	Batch Size	Accuracy
1	Adam	30	0.4	32	0.8995
2	Adam	40	0.5	64	0.9065
3	Adam	50	0.6	128	0.9280
4	AdaGrad	30	0.5	128	0.9199
5	AdaGrad	40	0.6	32	0.9113
6	AdaGrad	50	0.4	64	0.9274
7	Adadelta	30	0.6	64	0.9108
8	Adadelta	40	0.4	128	0.9258
9	Adadelta	50	0.5	32	0.9237

4.3 結論

最終程式之執行結果如圖 24、圖 25 所示，最終訓練、驗證、測試之準確率如表 7 所示。

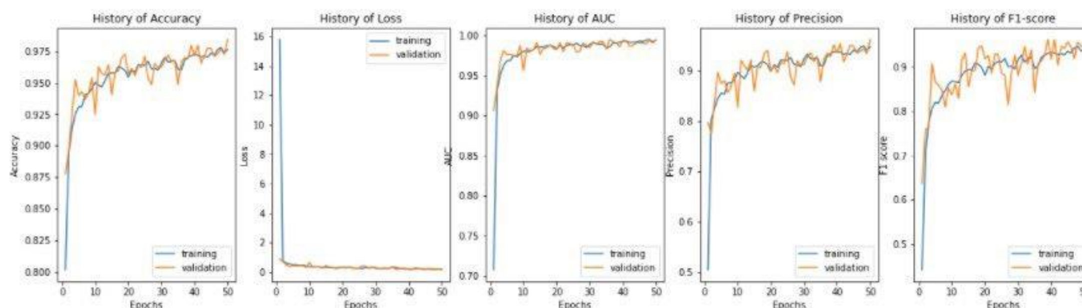

圖 24、程式執行結果 1

```
Found 372 images belonging to 5 classes.
3/3 [==============================] - 3s 1s/step - loss: 1.3792 - accuracy: 0.9280 - precision: 0.8251 - recall: 0.8118 - auc: 0.9293 - f1_score: 0.8206
```

圖 25、程式執行結果 2

表 7、程式資料集準確率

準確率 / 資料集	準確率
Train	0.9561
Val	0.9566
Test	0.9280

五、結果與討論

5.1 結論

- 於輸入資料數不對稱之情況下，可透過 DCGAN 增強資料，其能補足資料數不對稱之缺陷。
- 於輸入資料數不對稱之情況下，透過 DCGAN 增強現有資料後，將增強後之資料與原始資料透過 VGG16 訓練，結果顯示經過 DCGAN 增強過後之資料

與原始資料相比，資料增強後之表情辨識結果有更高之準確率。

- 可透過實驗設計等方式，根據欲改變的因子及水準選擇適當的直交表並完成實驗，求得模型所需參數之最佳組合。

5.2 未來展望

- **套用真實顧客之表情**

  若使用真實顧客之表情圖片於模型內進行優化，可預期模型之呈現結果將更加準確。

- **泛化性**

  透過不同之 CNN 模型驗證 DCGAN 將原資料增強後之資料集的泛化能力。