深度學習為基礎之顧客情緒及商品喜好程度分析

指導教授：邱銘傳 教授

Group 6

戴佩怡 110034543
徐一正 110034554
方際勛 110034557
劉兆原 110034559

Table of Contents

Scenario

Background

- 美國麻省理工學院教授Rosalind Picard發表情緒運算（Affective Computing）相關研究
- MarketWatch的數據顯示2018年全球的情緒識別產品市值已達到123.7億美元
- 《The Guardian》（2019）指出「AI 臉部辨識情緒」已成為規模 200 億美元（約台幣 6,210 億元）的產業，且仍持續擴充中

Motivation & Purpose

- 企業欲瞭解顧客意見往往使用問卷調查等方式，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
- 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

Problem Definition – 5W1H

Why

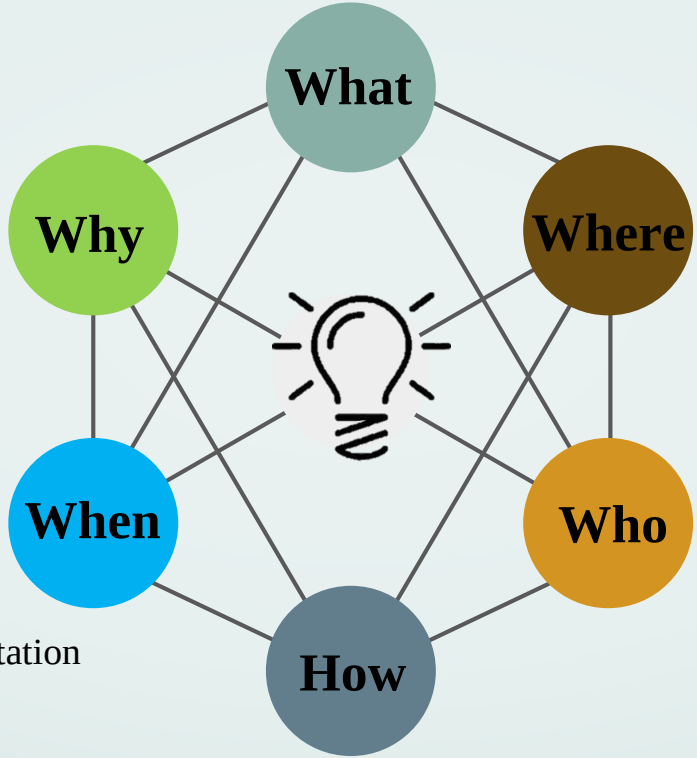
企業欲調查顧客意見費時費力
顧客未能反應其真實情緒

When

民眾逛街、選購產品時

How

運用DCGAN進行Data Augmentation
以VGG16辨識臉部表情



What

以顧客臉部表情判斷
其對產品的喜好程度

Where

產品銷售處、貨架區
結帳等候線

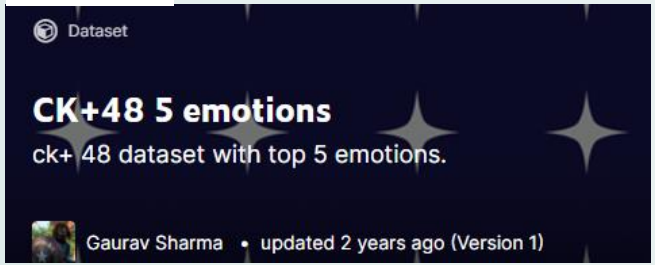
Who

企業行銷部門
欲購物之民眾

Data Resource

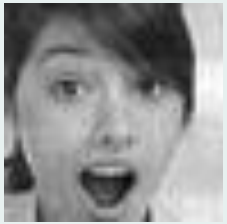
看到某產品後可能出現的表情及情緒：開心、驚喜、嫌棄、無感……

kaggle



CK-48 dataset with five emotions.

Happy



Surprise

Disgust

Fear

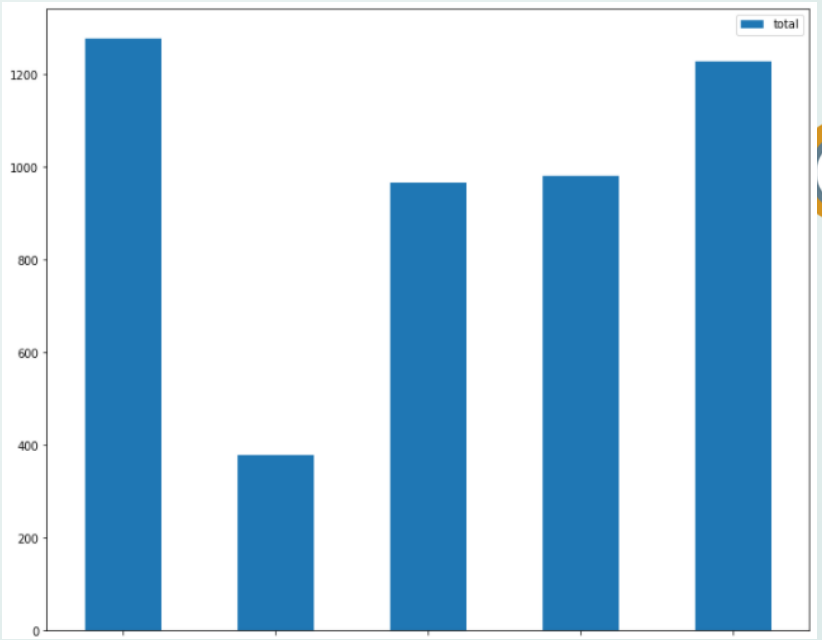
Anger

Original Data

CK-48 total number of images:

4442

原始資料集5種表情之數量分佈



Data Preprocessing

Data Augmentation

Keras ImageDataGenerator class

Random Rotations
Random Zoom
Random Shifts
Random Flips

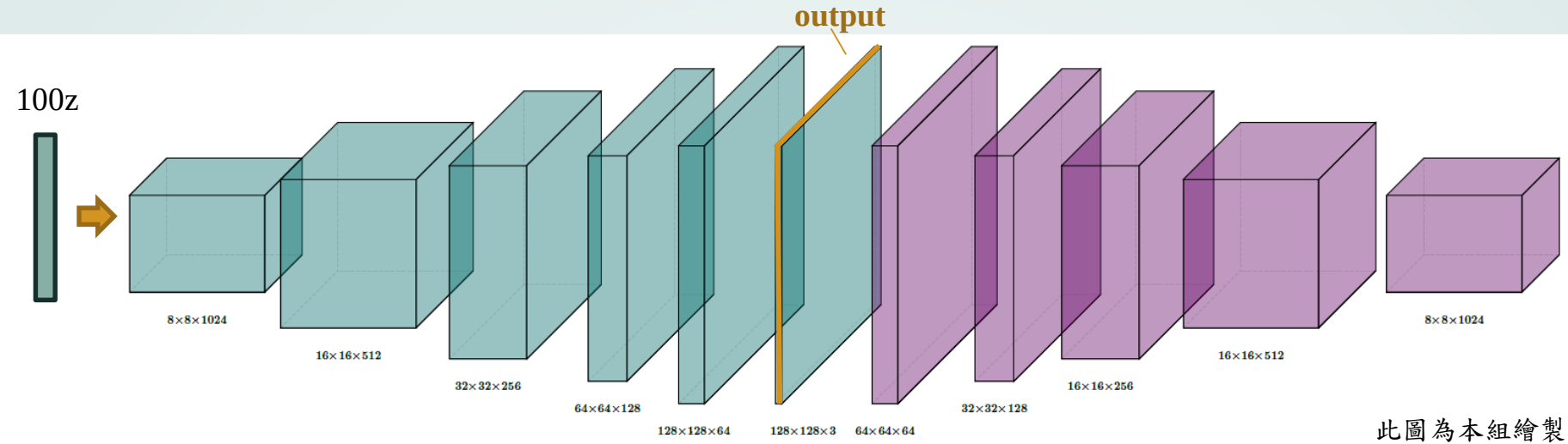
```
# Data Augmentation
train_datagen = ImageDataGenerator(
                rotation_range=5,    # randomly rotate images in the range 5 degrees
                zoom_range = 0.1, # Randomly zoom image 10%
                width_shift_range=0.1,    # randomly shift images horizontally 10%
            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

資料切割

Train set	**80%**	**3698**
Val set	**10%**	**372**
Test set	**10%**	**372**

DCGAN

Input DeConv1 DeConv2 DeConv3 DeConv4 DeConv5 Conv1 Conv2 Conv3 Conv4 Conv5 Output



此圖為本組繪製

Generator

Discriminator

Generator

```python
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)
```

DeConv1

```python
        # 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")
```

DeConv2

```python
        # 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")
```

DeConv3

```python
        # 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")
```

DeConv4

```python
        # 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")
```

DeConv5

```python
        # 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")
```

Output

```python
        out = tf.tanh(logits, name="out")
        return outs
```

Discriminator

Conv1

```
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")
```

Conv2

```
        # 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")
```

Conv3

```
        # 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")
```

Conv4

```
        # 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")
```

Conv5

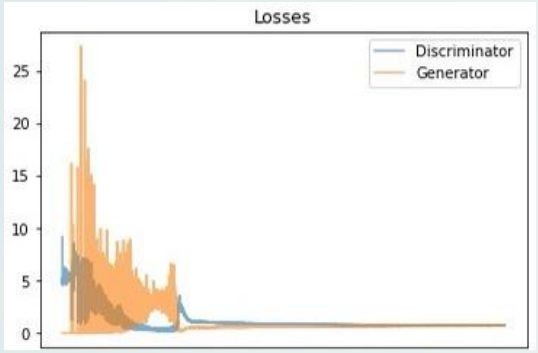
```
        # 16x16x512 -> 8x8x1024
        conv5 = tf.layers.conv2d(inputs=conv4_out,filters=1024,kernel_size=[5, 5],strides=[2, 2],padding="SAME",
                                 kernel_initializer=tf.truncated_normal_initializer(stddev=WEIGHT_INIT_STDDEV),
                                 name='conv5')
        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")
```

Output

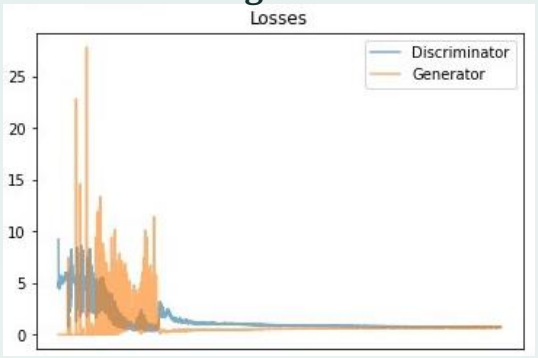
```
        flatten = tf.reshape(conv5_out, (-1, 8*8*1024))
        logits = tf.layers.dense(inputs=flatten,
                                 units=1,
                                 activation=None)
        out = tf.sigmoid(logits)
        return out, logits
```

Training Loss

happy

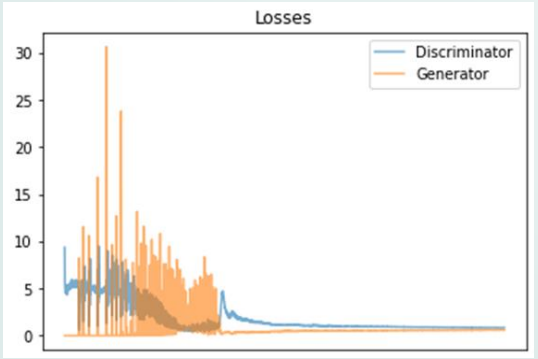


disgust

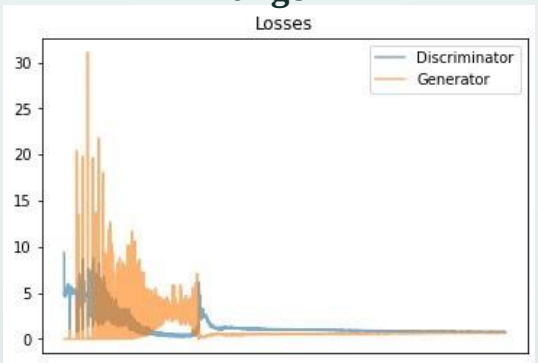



```
#  Hyperparameters
IMAGE_SIZE  =  128
NOISE_SIZE  =  100
LR_D  =  0.00001
LR_G  =  0.0004
BATCH_SIZE  =  64
EPOCHS  =  500
BETA1  =  0.5
WEIGHT_INIT_STDDEV  =  0.02
EPSILON  =  0.00005
```

fear



anger

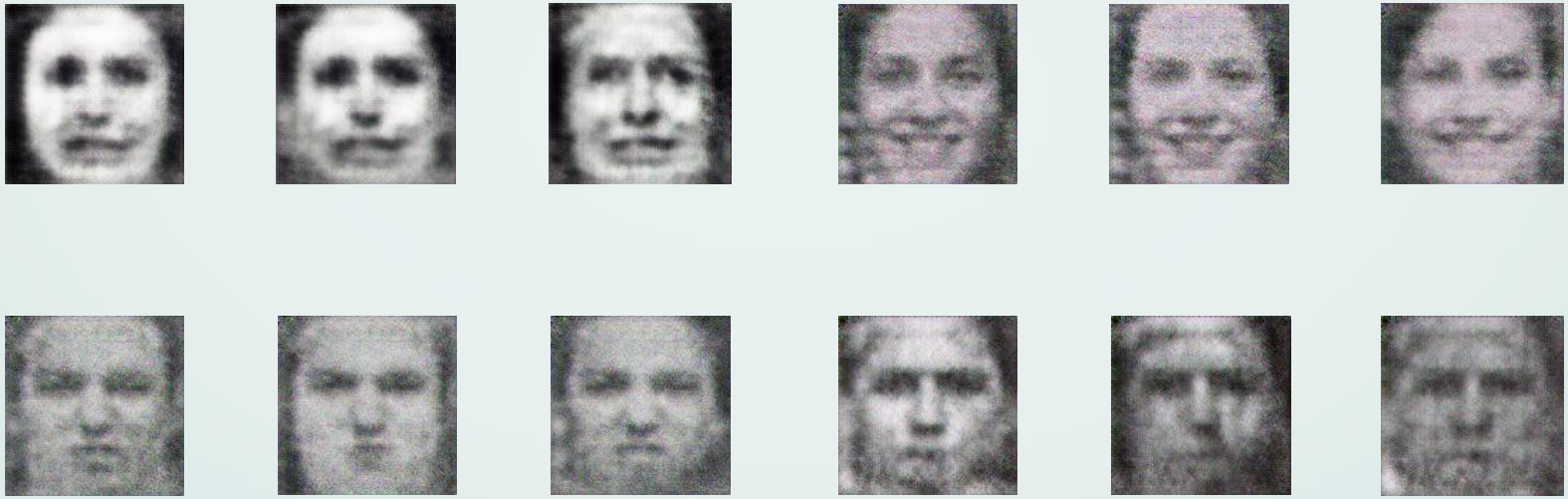



```
batch  =  np.asarray(augmented_images)
# Normalize images from 0 / 255 to -1 / 1
normalized_batch  =  (batch  /  127.5)  -  1.0
batches.append(normalized_batch)
```

```
d_train_opt  =  tf.train.AdamOptimizer
g_train_opt  =  tf.train.AdamOptimizer
```

Data Augmentation Result

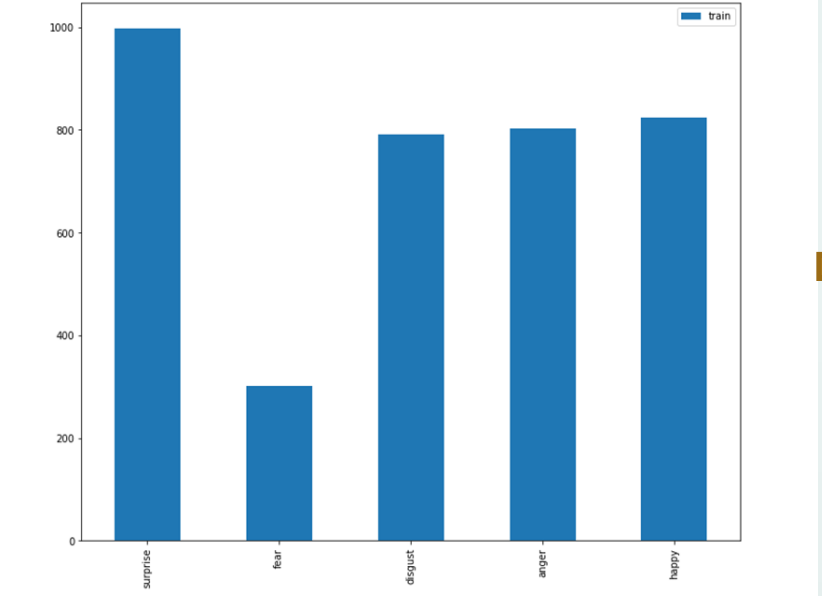
GAN 生成器生成之圖片



Data Augmentation Result

```
train_count.transpose().plot(kind='bar',  figsize=(12,10))
```

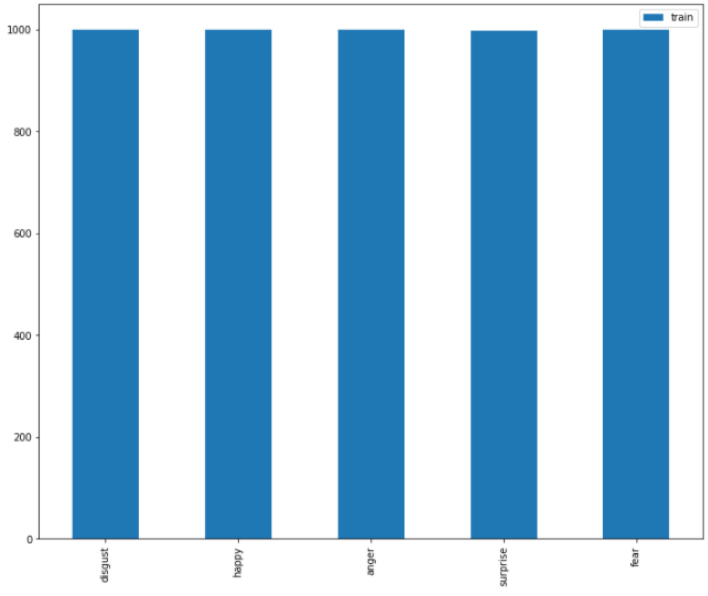
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f809008d650>
```



Before data augmentation

```
train_count.transpose().plot(kind='bar',  figsize=(12,10))
```

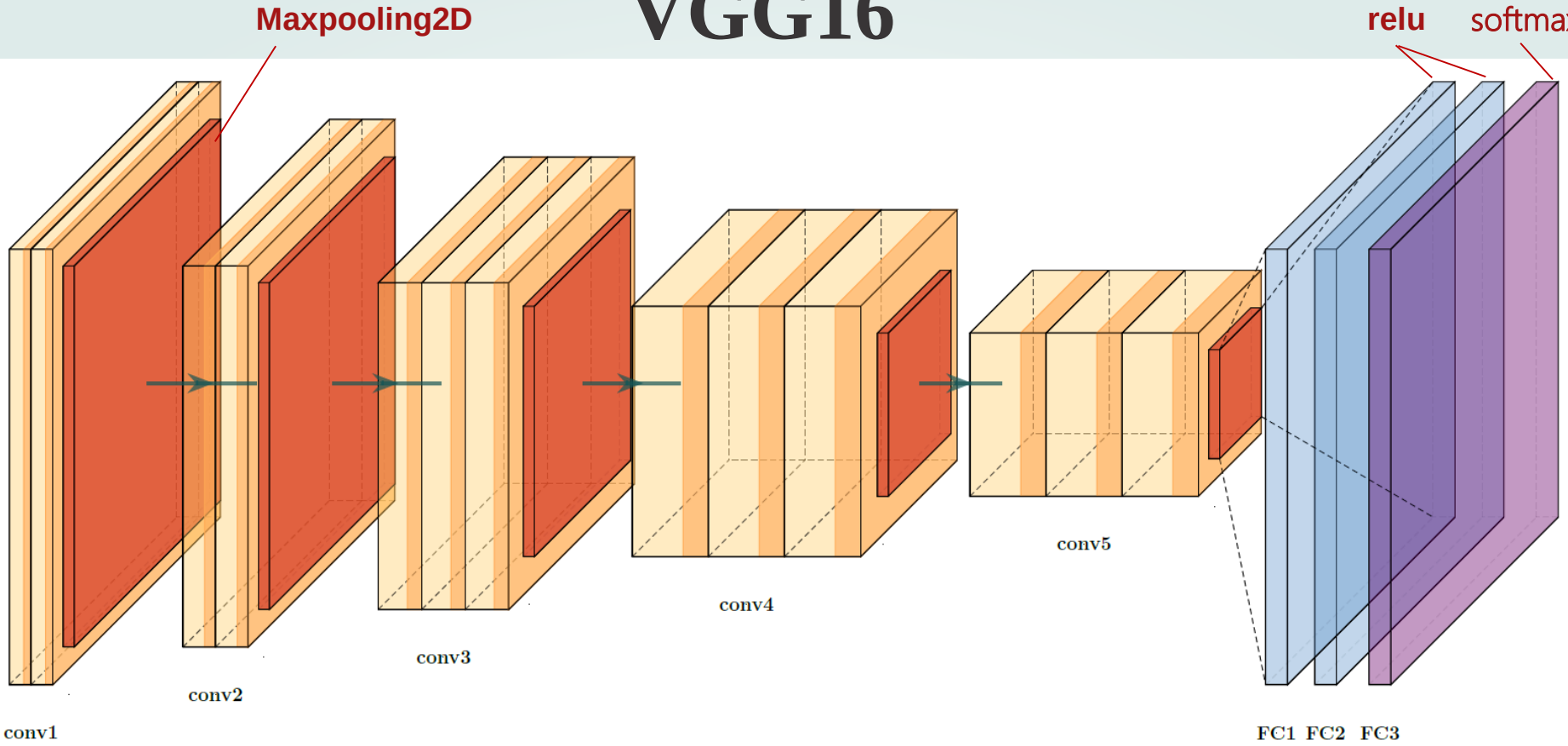
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd2d6c31190>
```



After data augmentation

Deep Learning Model

VGG16



此圖為本組繪製

Model Summary-VGG16

```
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
```

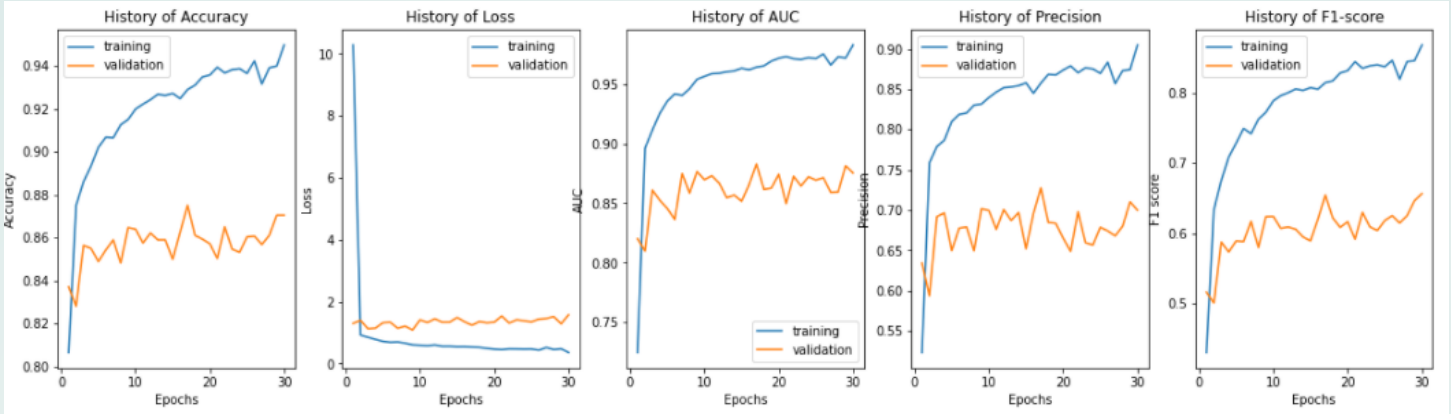
```
model = models.Sequential()

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()
```

```
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

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

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

- ReduceLROnPlateau：當模型連續訓練n次沒有更好，就進行一次學習率衰降

VGG16 model with Original Dataset



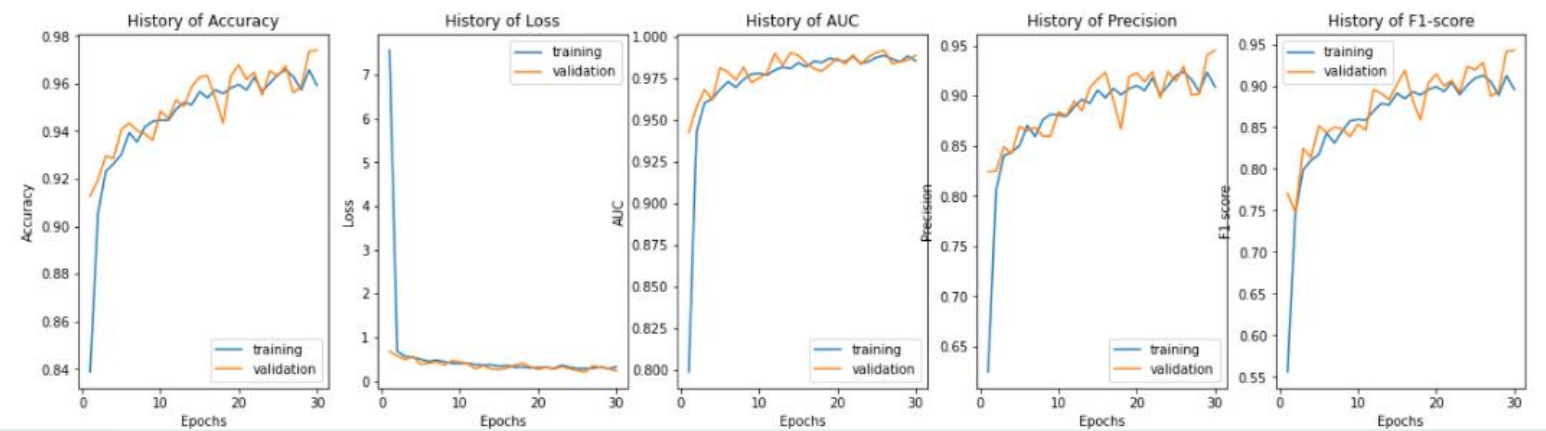

```
test_datagen    =  ImageDataGenerator()

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
```

	ACCURACY
Train	**0.9497**
Validation	**0.8704**
Test	**0.8639**

```
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
```

VGG16 model with New dataset




```
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
```

	ACCURACY
Train	**0.9764**
Validation	**0.9842**
Test	**0.9274**

```
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
```

Analysis

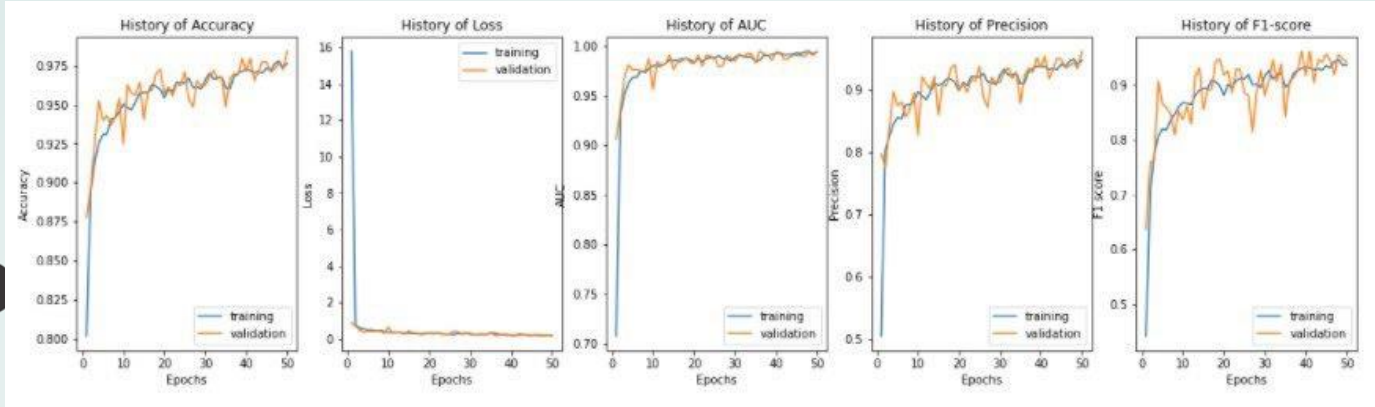
Design of Experiments

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

Orthogonal Array $L_9\ (\ 3^4\ )$

Experiment \ Factor	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

Results




```
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
```

	ACCURACY
Train	**0.9561**
Validation	**0.9566**
Test	**0.9280**

```
model.compile(optimizer='Adam',
model.add(layers.Dropout(0.6))
(train_dir,  target_size=(48,  48),  batch_size=128,
(val_dir,  batch_size=128,
.(train_generator, validation_data=val_generator, epochs  =  50,
```

Conclusion

Conclusion

在輸入資料數不平衡之情況下，可透過DCGAN增強資料，補足資料數不平衡之缺陷。

透過DCGAN數據增強後的資料，進行VGG16訓練，結果提高面部表情分類結果的準確率。

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。

Further Improvement

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，或許模型呈現之結果會更加準確。

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。

Reference

1. https://www.sciencedirect.com/science/article/pii/S0020025520306484
2. https://medium.com/swlh/image-generation-using-keras-framework-eb7480f7847a
3. https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
4. https://iter01.com/515554.html#google_vignette
5. https://ithelp.ithome.com.tw/articles/10273302
6. https://kknews.cc/zh-tw/tech/5ajljgl.html

Thank you!