圖像分類 CNN face

- 這個學習包是要利用 CNN 進行臉部表情的分類，
- 共有 ["angry", "disgust","fear","happy","neutral","sad","surprise"] 七個表情

匯入自己的圖像資料

- 圖像檔由像素所構成，圖片有大小，精密度差別很大
- 要統一成固定的規格，且要數位化，亦即轉為電腦可以讀得懂的數字格式
- 範例檔 face 解壓縮後，請置於 d:\my python\corpus\face\
- 資料夾下，有七個次資料夾，裡面有為數眾多的圖像檔，代表七個分類。

Import 必要模組

In [1]:

```
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing import image
from tensorflow.keras import Sequential

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
import pickle
```

data

- 注意檔案所在的目錄

In [2]:

```
# 輸入檔名檢查圖像
path = "c:/111/natural_images"
# 類別名稱即為資料夾的名稱
CATEGORIES = ["airplane", "car","cat","dog","flower","motorbike","person"]
# dogs 類別
directory = os.path.join(path,CATEGORIES[0])
# 第一筆資料
img = image.load_img(directory+"/airplane_0000.jpg")
plt.imshow(img)
```

Out[2]:

```
<matplotlib.image.AxesImage at 0x1d736d55280>
```

In [3]:

```
# CV2 讀取圖像檔轉為數字
```

```
img_array=cv2.imread(directory+"/airplane_0000.jpg")
# 縱向 499 個像素
print (len(img_array))
 # 橫向 327 個像素
print (len(img_array[0]))
# 每個像素有三個數值
print (len(img_array[0][0]))
# 第一個像素的三個數值，代表 R G B 三個顏色
print (img_array[0][0])
# shape
print (img_array.shape)
```

```
104
300
3
[232 240 223]
(104, 300, 3)
```

In [4]:

```
# 各像表數值代表顏色組成
img_array
```

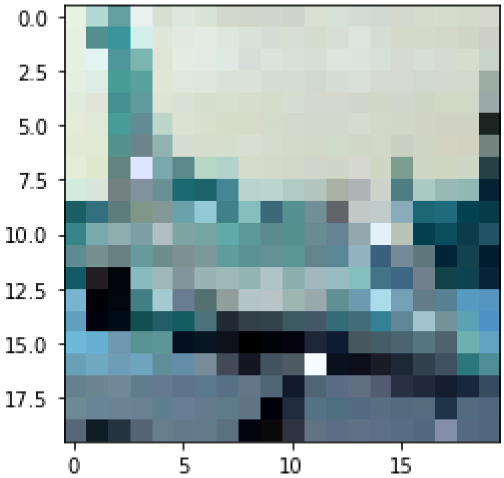
Out[4]:

```
array([[[232, 240, 223],
        [232, 240, 223],
        [233, 241, 224],
        ...,
        [210, 217, 204],
        [211, 218, 205],
        [209, 216, 203]],

       [[230, 238, 221],
        [231, 239, 222],
        [232, 240, 223],
        ...,
        [213, 220, 207],
        [208, 215, 202],
        [210, 217, 202]],

       [[228, 238, 222],
        [229, 239, 223],
        [229, 239, 223],
        ...,
        [209, 214, 199],
        [210, 217, 202],
        [211, 219, 202]],

       ...,

       [[104, 128, 148],
        [103, 127, 147],
        [106, 128, 146],
        ...,
        [ 78, 100, 125],
        [ 79,  99, 124],
        [ 78,  98, 123]],

       [[ 90, 111, 132],
        [ 82, 103, 124],
        [ 75,  94, 115],
        ...,
        [ 88, 109, 136],
        [ 89, 110, 137],
        [ 87, 108, 135]],

       [[ 96, 117, 138],
        [ 85, 106, 127],
        [ 73,  92, 113],
        ...,
        [ 92, 113, 140],
        [ 94, 115, 142],
        [ 92, 113, 140]]], dtype=uint8)
```

資料前處理

In [5]:

```python
img = image.load_img(directory+"/airplane_0000.jpg")
new_array = cv2.resize(img_array, (20, 20))
plt.imshow(new_array, cmap='gray')
plt.show()
```



覺得還是不要簡化，保留它原始的尺寸，IMG_SIZE=48

In [6]:

```python
# training_data
path = "c:/111/natural_images"
# 類別，與資料夾名稱同
CATEGORIES = ["airplane", "car","cat","dog","flower","motorbike","person"]
# 圖片大小設定
IMG_SIZE=48
training_data = []
def create_training_data():
    for category in CATEGORIES:  # do dogs and cats
        directory = os.path.join(path,category)  # create path to dogs and cats
        class_num = CATEGORIES.index(category)  #  0=dog 1=cat
        # 讀取圖像，數字轉換，調整大小，只取灰階
        for img in tqdm(os.listdir(directory)):
            try:
                # os.path.join(directory,img) 即 directory+"/"+img
                img_array = cv2.imread(os.path.join(directory,img) ,cv2.IMREAD_GRAYS
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
                # training data 是一個list, 每個元素，
                # 由 features 與 label 構成，label 是數字代碼
                training_data.append([new_array, class_num])
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()
# 總共筆數
print(len(training_data))
```

```
100%|██████████████████████████████████████████████████████████████████████████
| 727/727 [00:04<00:00, 170.49it/s]
100%|██████████████████████████████████████████████████████████████████████████
| 968/968 [00:05<00:00, 179.25it/s]
```

```
100%|████████████████████████████████████████
| 885/885 [00:05<00:00, 149.30it/s]
100%|████████████████████████████████████████
| 702/702 [00:04<00:00, 149.42it/s]
100%|████████████████████████████████████████
| 843/843 [00:06<00:00, 125.67it/s]
100%|████████████████████████████████████████
| 788/788 [00:04<00:00, 173.57it/s]
100%|████████████████████████████████████████
| 986/986 [00:05<00:00, 164.78it/s]
5899
```

In [7]:

```
# 重新排序
import random
random.shuffle(training_data)
```

In [8]:

```
# features and label
X = []
y = []
for features,label in training_data:
    X.append(features)
    y.append(label)

# 重組 張量的 shape, -1 代表任意數，由電腦計算得之
# X 是 features 注意 input shape 須一致 the last one is new and represent the color
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE,1)
# y 是label, 數字代碼
y=np.array(y).reshape(-1, 1)
```

In [9]:

```
# 注意 input shape
X.shape
```

Out[9]:

```
(5899, 48, 48, 1)
```

In [10]:

```
X.shape[1:]
```

Out[10]:

```
(48, 48, 1)
```

In [11]:

```
y.shape
```

Out[11]:

```
(5899, 1)
```

In [12]:

```
# 標準化
X = X/255.0
```

save

- 注意，我們存到 d:/my python/data/ 目錄下

In [13]:

```
import pickle

pickle_out = open("c:/111/data/face_X.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("c:/111/data/face_y.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

In [14]:

```
pickle_in = open("c:/111/data/face_X.pickle","rb")
```

```
X = pickle.load(pickle_in)

pickle_in = open("c:/111/data/face_y.pickle","rb")
y = pickle.load(pickle_in)
```

Model, DNN

In [15]:

```
model_DNN = keras.Sequential([
    # input layer (1) 像素攤平，成為 一個維度
    # 注意 input_shape 與 X feature 的 shape 要一致 (previous output)
    keras.layers.Flatten(input_shape=(48, 48,1)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    # output layer
    keras.layers.Dense(7, activation='softmax')
])
model_DNN.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 2304)              0         
_________________________________________________________________
dense (Dense)                (None, 128)               295040    
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 455       
=================================================================
Total params: 303,751
Trainable params: 303,751
Non-trainable params: 0
_________________________________________________________________
```

compile

In [16]:

```
model_DNN.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

fit

In [17]:

```
history =model_DNN.fit(X,y, batch_size=32, epochs=10,validation_split=0.1)   # ,vali
```

```
Epoch 1/10
166/166 [==============================] - 2s 10ms/step - loss: 1.6626 - accuracy:
0.3768 - val_loss: 1.1819 - val_accuracy: 0.5763
Epoch 2/10
166/166 [==============================] - 0s 3ms/step - loss: 1.0633 - accuracy: 0.
6181 - val_loss: 1.1020 - val_accuracy: 0.6000
Epoch 3/10
166/166 [==============================] - 0s 3ms/step - loss: 0.9174 - accuracy: 0.
6663 - val_loss: 0.9865 - val_accuracy: 0.6203
Epoch 4/10
166/166 [==============================] - 0s 3ms/step - loss: 0.8551 - accuracy: 0.
6756 - val_loss: 0.9898 - val_accuracy: 0.6407
Epoch 5/10
166/166 [==============================] - 0s 3ms/step - loss: 0.7516 - accuracy: 0.
7306 - val_loss: 1.0332 - val_accuracy: 0.6000
Epoch 6/10
166/166 [==============================] - 0s 3ms/step - loss: 0.6988 - accuracy: 0.
7467 - val_loss: 0.9319 - val_accuracy: 0.6644
```

```
Epoch 7/10
166/166 [==============================] - 0s 3ms/step - loss: 0.6949 - accuracy: 0.
7375 - val_loss: 0.9891 - val_accuracy: 0.6424
Epoch 8/10
166/166 [==============================] - 1s 3ms/step - loss: 0.6732 - accuracy: 0.
7526 - val_loss: 0.8662 - val_accuracy: 0.6780
Epoch 9/10
166/166 [==============================] - 0s 3ms/step - loss: 0.6504 - accuracy: 0.
7564 - val_loss: 0.8950 - val_accuracy: 0.6678
Epoch 10/10
166/166 [==============================] - 0s 3ms/step - loss: 0.5968 - accuracy: 0.
7746 - val_loss: 0.8596 - val_accuracy: 0.6695
```

結果不好，看來要另闢奚徑了，就用 CNN 試試看。

CNN

- 卷積層的數目，自己決定，一個或數個均可。
- 最後一個卷積層的後面要加 Flatten() layer
- 要特別注意 input shape 否則無法建模成功

In [18]:

```
# 重組 張量的 shape
# 配合 input layer
IMG_SIZE=48
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE,1)
y=np.array(y).reshape(-1, 1)
```

In [19]:

```
X.shape[1:]
```

Out[19]:

```
(48, 48, 1)
```

In [20]:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Activation,MaxPooling2D,Conv2D,Flatten,Dense
# sequential 宣告
model = Sequential()
# 卷積層 256 個 filter, filter size (3,3), 注意要有 input_shape 48>>>46 有損失
model.add(Conv2D(256, (3, 3),activation='relu', input_shape=(48, 48,1)))
# Maxpooling2D
model.add(MaxPooling2D(pool_size=(2, 2)))
# 第二個卷積層
model.add(Conv2D(256, (3, 3),activation='relu'))
# Maxpooling2D
model.add(MaxPooling2D(pool_size=(2, 2)))
# 攤平變成一個維度(一定要，不能省略)10*10*256
model.add(Flatten())
# 全連接層，神經元 64
model.add(Dense(64))
# 全連接層，神經元 7，作為 output layer, activation=softmax 傳出各類別的機率
model.add(Dense(7, activation='softmax'))
# loss='sparse_categorical_crossentropy'
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

In [21]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 46, 46, 256)       2560      
```

```
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 256)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 256)       590080    
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 256)       0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 25600)             0         
_________________________________________________________________
dense_3 (Dense)              (None, 64)                1638464   
_________________________________________________________________
dense_4 (Dense)              (None, 7)                 455       
=================================================================
Total params: 2,231,559
Trainable params: 2,231,559
Non-trainable params: 0
_________________________________________________________________
```

- 共有兩個 conv2d, 第一個 output shape 為 (None, 46, 46, 256) 46 是 48-2 因為 filter 是 (3, 3)，邊缘損耗 2, 256 是 filter 的深度（depth），可視為重覆多次的 filter
- max_pooling2d 為 (2, 2) 的視窗簡化圖形，Output Shape 變成 (None, 23, 23, 256)
- 第二個 output shape 為 (None, 21, 21, 256) 21 為 23-2 原理與前同
- 第二個 max_pooling2d 為 (2, 2) 的視窗簡化圖形，Output Shape 變成 (None, 10, 10, 256)
- Flatten 為攤平成為一個維度，Output Shape 為 (None, 25600)，25600=10*10*256
- 第一個 dense, nuron 為64
- 第二個 dense, nuron 為7

CNN2

- 改變一些參數設定

In [22]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Activation,MaxPooling2D,Conv2D,Flatten,Dense

# 宣告 Sequential() 若不想取代掉先前的 要改名
model = Sequential()
# Conv2D
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
# MaxPooling2D
model.add(MaxPooling2D((2, 2)))
# Conv2D
model.add(Conv2D(64, (3, 3), activation='relu'))
# MaxPooling2D
model.add(MaxPooling2D((2, 2)))
# Conv2D
model.add(Conv2D(64, (3, 3), activation='relu'))
# Flatten
model.add(Flatten())
# Dense
model.add(Dense(64, activation='relu'))
# Dense
model.add(Dense(7, activation='softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 46, 46, 32)        320       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 64)        18496     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928     
_________________________________________________________________
flatten_2 (Flatten)          (None, 4096)              0         
_________________________________________________________________
dense_5 (Dense)              (None, 64)                262208    
_________________________________________________________________
dense_6 (Dense)              (None, 7)                 455       
=================================================================
Total params: 318,407
Trainable params: 318,407
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

fit (這要花一點時間)

In [24]:

```
history =model.fit(X,y, batch_size=32, epochs=10,validation_split=0.1,verbose=2)
```

```
Epoch 1/10
166/166 - 10s - loss: 1.1356 - accuracy: 0.5779 - val_loss: 0.8193 - val_accuracy: 
0.6915
Epoch 2/10
166/166 - 9s - loss: 0.6391 - accuracy: 0.7595 - val_loss: 0.6612 - val_accuracy: 0.
7424
Epoch 3/10
166/166 - 9s - loss: 0.5097 - accuracy: 0.8088 - val_loss: 0.5526 - val_accuracy: 0.
7881
Epoch 4/10
166/166 - 9s - loss: 0.4177 - accuracy: 0.8452 - val_loss: 0.5052 - val_accuracy: 0.
8017
Epoch 5/10
166/166 - 9s - loss: 0.3555 - accuracy: 0.8631 - val_loss: 0.5551 - val_accuracy: 0.
7831
Epoch 6/10
166/166 - 9s - loss: 0.3055 - accuracy: 0.8851 - val_loss: 0.4986 - val_accuracy: 0.
8153
Epoch 7/10
166/166 - 9s - loss: 0.2565 - accuracy: 0.9041 - val_loss: 0.5313 - val_accuracy: 0.
8034
Epoch 8/10
166/166 - 9s - loss: 0.2345 - accuracy: 0.9090 - val_loss: 0.5173 - val_accuracy: 0.
8203
Epoch 9/10
166/166 - 10s - loss: 0.1940 - accuracy: 0.9248 - val_loss: 0.4730 - val_accuracy: 
0.8390
Epoch 10/10
166/166 - 9s - loss: 0.1528 - accuracy: 0.9452 - val_loss: 0.5663 - val_accuracy: 0.
8305
```

evaluation

In [34]:

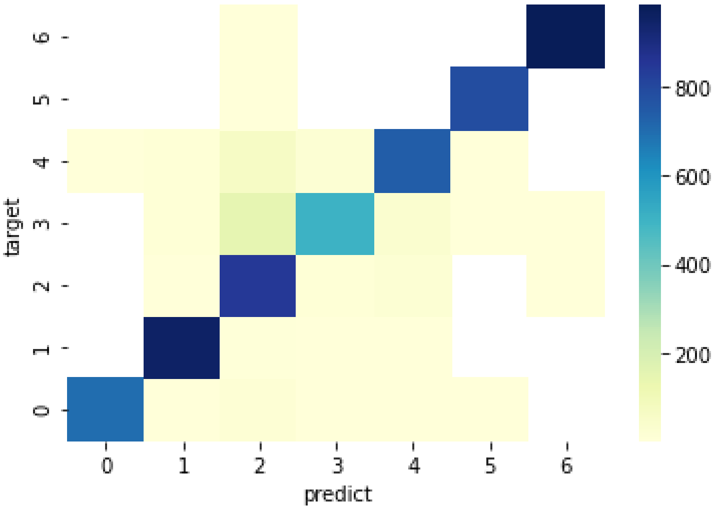
```
##-- 評估模型表現 (各類別 accuracy) --##
import seaborn as sns
def acc_matrix(model, test_x, test_y):
    pred = model.predict(test_x)
    pred_y = np.argmax(pred, axis=1)
    acc = pd.DataFrame({'target': test_y.squeeze(1), 'predict': pred_y})
    acc['corr'] = acc['target'] == acc['predict']
    acc = acc.groupby(['target', 'predict']).count().unstack('predict')
    acc.columns = acc.columns.droplevel(0)
    display(acc)

    sns.heatmap(acc, cmap="YlGnBu").invert_yaxis()
    plt.show()
```

In [35]:

```
# 針對 training set, 如果有 test set 可比較 x & y are tranning set 對角線深就是對 旁邊
print (CATEGORIES)
acc_matrix(model,X, y)
```

```
['airplane', 'car', 'cat', 'dog', 'flower', 'motorbike', 'person']
```

predict	0	1	2	3	4	5	6
target							
0	703.0	1.0	19.0	1.0	2.0	1.0	NaN
1	NaN	957.0	8.0	1.0	2.0	NaN	NaN
2	NaN	3.0	848.0	12.0	21.0	NaN	1.0
3	NaN	11.0	149.0	507.0	32.0	1.0	2.0
4	1.0	10.0	66.0	22.0	741.0	3.0	NaN
5	NaN	NaN	1.0	NaN	NaN	787.0	NaN
6	NaN	NaN	1.0	NaN	NaN	NaN	985.0



In [36]:

```
import pandas as pd
# history 轉為 dataframe 格式
hist = pd.DataFrame(history.history)
# 新增 epoch 欄位
hist['epoch'] = history.epoch
# 顯示 epoch, loss, val_loss
hist.tail()
```

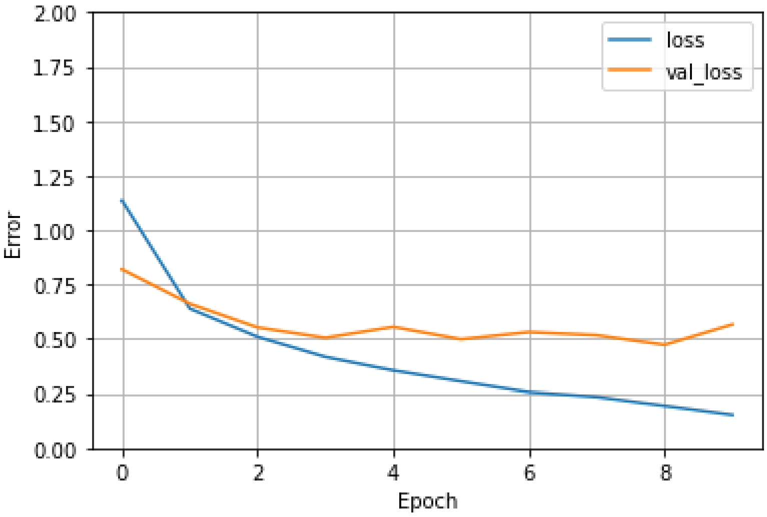
Out[36]:

	loss	accuracy	val_loss	val_accuracy	epoch
5	0.305480	0.885101	0.498622	0.815254	5
6	0.256487	0.904125	0.531340	0.803390	6
7	0.234484	0.909022	0.517272	0.820339	7

	loss	accuracy	val_loss	val_accuracy	epoch
8	0.194036	0.924845	0.472981	0.838983	8
9	0.152831	0.945187	0.566326	0.830508	9

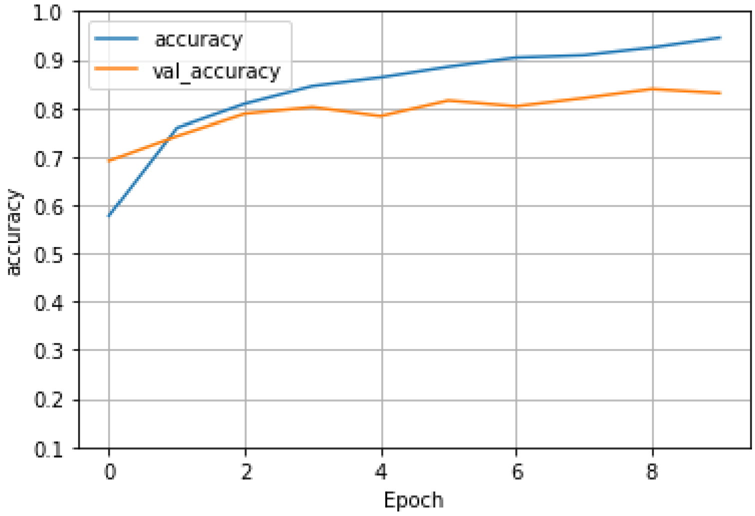
In [37]:

```
# 繪圖，顯示損失函數下降的趨勢
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 2])
  plt.xlabel('Epoch')
  plt.ylabel('Error')
  plt.legend()
  plt.grid(True)
plot_loss(history)
```



In [40]:

```
# 繪圖，顯示正確率上升的趨勢
def plot_acc(history):
  plt.plot(history.history['accuracy'], label='accuracy')
  plt.plot(history.history['val_accuracy'], label='val_accuracy')
  plt.ylim([0.1, 1.0])
  plt.xlabel('Epoch')
  plt.ylabel('accuracy')
  plt.legend()
  plt.grid(True)
plot_acc(history)
```



save and load

In [39]:

```
model.save('model_face_CNN')
```

```
INFO:tensorflow:Assets written to: model_face_CNN\assets
```

In [29]:

```
# Load model
model = tf.keras.models.load_model('model_face_CNN')
```

predict

隨機選取圖片預測

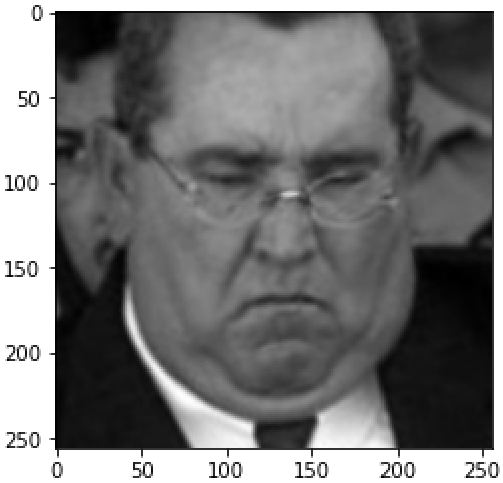
In [30]:

```
import random
```

In [25]:

```
path = "c:/111/natural_images"
CATEGORIES = ["airplane", "car","cat","dog","flower","motorbike","person"]
# 隨機選取目錄
cat=random.choice(CATEGORIES)
imgs=os.listdir(path+"/"+cat)
# 隨機選取檔案
img=random.choice(imgs)
# 完整路徑
filepath=path+"/"+cat+"/"+img
# 讀取影像檔
img_array=cv2.imread(filepath,cv2.IMREAD_GRAYSCALE)
# resize
new_array = cv2.resize(img_array, (48, 48))
# reshape, 要與 model 訓練時的 input shape 符合
new_array = np.array(new_array).reshape(IMG_SIZE, IMG_SIZE,1)
# float
new_array=new_array.astype(float)
# tensor, creat a batch 目的是與 model 訓練時的 input shape 一致
img_tf = tf.expand_dims(new_array, 0)
# prediction
predictions = model.predict(img_tf)
# score，顯示各類別的機率
score =predictions[0]

print (filepath)
print (score)
print("這應該是 {} 有百分之 {:.0f} 的信心 "
    .format(CATEGORIES[np.argmax(score)], 100 * np.max(score)))
plt.imshow(img_array, cmap='gray')
plt.show()
```

```
c:/111/natural_images/person/person_0885.jpg
[0. 0. 0. 0. 0. 0. 1.]
這應該是 person 有百分之 100 的信心
```



In [26]:
```
判斷不是很準，其實主要原因是訓練集，本身就不好判斷，到底是 "angry", "disgust" 根本分不清。
如果分成正面、負面、中性，三個類別，就容易判斷多了。
```

```
  File "<ipython-input-26-1f08263bd8ea>", line 1
    判斷不是很準，其實主要原因是訓練集，本身就不好判斷，到底是 "angry", "disgust" 根本分不清。
                                   ^
SyntaxError: invalid character in identifier
```

input 格式轉換，前面程式的解釋

- 注意要轉為 tensorflow 格式之後，才能 predict

In [ ]:
```
# cv2 數據化以後的結果
img_array
```

In [ ]:
```
# 轉為 tensorflow 格式 shape
img_tf.shape
```

In [ ]:
```
# tensorflow 變數值
img_tf
```

In [ ]:
```
```