

Face Recognition against My Photo Library

The iCloud photos from iOS or Mac OS provided a comprehensive way to tag faces and then all the photos in the library will be scanned, faces will be identified and tagged accordingly based on the earlier user defined tags.

I am a photography enthusiast and I have started to taking photos from year 2004. As of today, it's over 200k shots and I've kept 40k of them. There are many desktop applications can do the job of face recognition, but it's going to be super fun if I can build the solution end to end.

Transfer learning is a very useful way for extracting features from images. This report will only focus of using it to build DNN model to classify the images. It's definitely worth trying other machine learning ways to perform both supervised and unsupervised learning against the datasets.

This report will have following parts.

1. Get photos and preprocess.
2. Transfer learning and train the model.
3. Improve accuracy with more images.
4. Face recognition on sample photos.

Part 1 - Get photos and preprocessing

A glance at the folder structure

All my photos are in D:\Pictures, majority of them are in both .jpg and .nef format. The .nef is a raw image format for Nikon cameras and .jpg is the copy after image post-processing of raw file.

```
In [215]: import os
import time
cur_dir = os.getcwd()
print(cur_dir)
target_image_dir = os.path.join(cur_dir, 'images')
photo_dir = 'D:\Pictures'
os.listdir(photo_dir)
```

D:\Google Drive\Study\Deep Learning Developer\Projects\Project 4 - Face Recognition Against My Photo Library

```
Out[215]: ['.SynologyWorkingDirectory',
 '2004',
 '2005',
 '2006',
 '2007',
 '2008',
 '2009',
 '2010',
 '2011',
 '2012',
 '2013',
 '2014',
 '2015',
 '2016',
 '2017',
 'Adobe Lightroom',
 'Camera Roll',
 'desktop.ini',
 'iCloud Photos',
 'naming instruction.txt',
 'Phone Photos',
 'Photography Works',
 'Saved Pictures',
 'zbingjie',
 'zothers',
 '法蝶',
 '熊思宇和黄乐论辩论']
```

One sample of a recent photo directory. Each and every file use the time stamp as its file name. .nef is the raw image file, .xmp is generated by Adobe Lightroom, both are not applicable to this project.

```
In [216]: os.listdir(photo_dir + '/2017/2017.11.16 - Singapore Fintech Festival')
```

```
Out[216]: ['20171112-2038.jpg',
 '20171112-2038.NEF',
 '20171112-2038.xmp',
 '20171112-2039.jpg',
 '20171112-2039.NEF',
 '20171112-2039.xmp',
 '20171116-1705.jpg',
 '20171116-1705.NEF',
 '20171116-1705.xmp',
 '20171116-1707.jpg',
 '20171116-1707.NEF',
 '20171116-1707.xmp',
 '20171116-1708.jpg',
 '20171116-1708.NEF',
 '20171116-1708.xmp',
 '20171116-1709.jpg',
 '20171116-1709.NEF',
 '20171116-1709.xmp',
 '20171116-1710.jpg',
 '20171116-1710.NEF']
```

Create face images

Iteratively going through all the images and save the face to a jpg file into the working directory. The value of the scaling factor in the cascade classifier affects the number of false positives.

```
In [217]: # Import required Libraries for this section
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math
import cv2
import time
```

```
In [218]: def show_faces(file_path, display=False, save=False, scaleFactor=1.3, minNeighb=5):
    print('Image path', file_path)

    # The file path contains unicode characters, cannot use cv2.imread() directly
    file_stream = open(file_path, 'rb')
    bytes_arr = bytearray(file_stream.read())
    numpy_ar = np.asarray(bytes_arr, dtype=np.uint8)
    image = cv2.imdecode(numpy_ar, cv2.IMREAD_UNCHANGED)
    print(image.shape)

    # Convert to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Convert the RGB image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # Extract the pre-trained face detector from an xml file
    face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_frontalface_default.xml')

    # Detect the faces in image
    faces = face_cascade.detectMultiScale(gray, scaleFactor, minNeighb)

    # Print the number of faces detected in the image
    print('Number of faces detected:', len(faces))

    # Make a copy of the orginal image to draw face detections on
    image_with_detections = np.copy(image)

    # The List of detected faces
    image_faces = []
    # Get the bounding box for each detected face
    for (x,y,w,h) in faces:
        # Add a red bounding box to the detections image
        if w > 200:
            line_width = w//20
        else:
            line_width = 3
        image_faces.append(image[y:(y+h), x:(x+w)])
        cv2.rectangle(image_with_detections, (x,y), (x+w,y+h), (255,0,0), line_width)

    if save:
        save_faces(file_path, image_faces)

    if display:
        # Display the image with the detections
        fig = plt.figure(figsize=(10, 10))
        ax = fig.add_subplot(1, 1, 1, xticks=[], yticks[])
        ax.set_title('Sample Image')
        ax.imshow(image_with_detections)
        os.chdir(cur_dir)
```

```
In [219]: # pathlib available from python 3.5
from pathlib import Path
def save_faces(file_path, image_faces):
    if len(image_faces) == 0:
        return
    # Save the each face into individual files
    target_file = file_path.replace(photo_dir, target_image_dir)
    target_dir = os.path.dirname(target_file)
    target_path = Path(target_dir)

    # Create parents of directory, don't raise exception if the directory exists
    target_path.mkdir(parents=True, exist_ok=True)

    for i, face in enumerate(image_faces):
        face = cv2.resize(face, (299, 299))
        os.chdir(target_dir)
        file_name = os.path.basename(target_file)
        cv2.imwrite(file_name + '-face-' + str(i) + '.jpg', cv2.cvtColor(face, cv2.COLOR_BGR2RGB))
```

```
In [220]: # Load in color image for face detection
file_path = os.path.join(photo_dir, '2017\\2017.11.16 - Singapore Fintech Festival', '20171116-1923.jpg')
show_faces(file_path, True)
```

Image path D:\\Pictures\\2017\\2017.11.16 - Singapore Fintech Festival\\20171116-1923.jpg
(4760, 7132, 3)
Number of faces detected: 5



```
In [6]: all_jpg = []
for root, dirs, files in os.walk(photo_dir):
    # All the target photos are in D:\\Pictures\\20xx. Get the jpgs from them only.
    path = root.split(os.sep)
    if len(path) < 3:
        continue
    else:
        year = path[2]
        if year[:2] != '20':
            continue
    #print((len(path) - 1) * '---', os.path.basename(root))
    for file in files:
        if file[-3:].lower() == 'jpg':
            #print(len(path) * '---', file)
            all_jpg.append(os.path.join(root, file))
```

```
In [7]: print('Number of jpgs:', len(all_jpg))
all_jpg[:10]
```

Number of jpgs: 39687

```
Out[7]: ['D:\\\\Pictures\\\\2004\\\\2004.12.16 - 保存的第一张数码照片\\\\20041216-2037.JPG',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-0828.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-0829.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-1004.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-1011-2.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-1011.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-1013.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-1203.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-1219.jpg',
'D:\\\\Pictures\\\\2004\\\\2004.12.21~24 - 刚到新加坡, bukit timah hill, befriendeer, 猴子, oldham hall及附近\\\\20041221-1225.jpg']
```

```
In [ ]: #####
### RUN WITH CAUTION#####
#####

# Scan through all 40k photos and extract faces
for i in range(len(all_jpg)):
    show_faces(all_jpg[i], False, True)
```

Manually label the face images by putting them into different folders

There are 100k faces identified. Most of them are non-relevant. I hand picked over 200 of them.

```
In [221]: os.listdir('./images')
```

```
Out[221]: ['Test', 'Train', 'Validate']
```

```
In [222]: os.listdir('./images/Test')
```

```
Out[222]: ['Brother', 'Dad', 'Daughter', 'Me', 'Mum', 'Son', 'Wife']
```

Part 2 - Transfer learning and train the model

```
In [36]: from keras.applications.inception_v3 import InceptionV3
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras import backend as K
from keras.applications.imagenet_utils import preprocess_input, decode_predictions
from keras.callbacks import ModelCheckpoint, EarlyStopping, LambdaCallback, ReduceLROnPlateau
from keras.models import load_model
```

```
In [ ]: # Load the inception v3 model, exclude the dense layers
base_model = InceptionV3(weights='imagenet', include_top=False)
base_model.summary()
```

```
In [13]: # Set all the layers from inception to untrainable
for layer in base_model.layers:
    layer.trainable = False
```

```
In [21]: # Add pooling and dense Layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1000, activation = 'relu')(x)
x = Dense(200, activation = 'relu')(x)
x = Dense(50, activation = 'relu')(x)
x = Dense(7, activation = 'softmax')(x)
```

```
my_model_1 = Model(inputs = base_model.input, outputs = x)
my_model_1.summary()

activation_77 (Activation)      (None, None, None, 32 0)          batch_normalization_77[0][0]
mixed9_0 (Concatenate)         (None, None, None, 76 0)          activation_79[0][0]
                                         activation_80[0][0]
concatenate_1 (Concatenate)    (None, None, None, 76 0)          activation_83[0][0]
                                         activation_84[0][0]
activation_85 (Activation)     (None, None, None, 19 0)          batch_normalization_85[0][0]
mixed9 (Concatenate)          (None, None, None, 20 0)          activation_77[0][0]
                                         mixed9_0[0][0]
                                         concatenate_1[0][0]
                                         activation_85[0][0]
conv2d_90 (Conv2D)            (None, None, None, 44 917504)    mixed9[0][0]
batch_normalization_90 (BatchNor (None, None, None, 44 1344)    conv2d_90[0][0]
activation_90 (Activation)    (None, None, None, 44 0)          batch_normalization_90[0][0]
```

```
In [223]: # Function to load the images as a list of file names and one hot code categories
from sklearn.datasets import load_files
from keras.utils import np_utils
from glob import glob

# Read all the files and return 2 numpy arrays, 1 is the address of the files
# and 1 is the one hot encode of the category.
def load_dataset(path):
    data = load_files(path)
    face_files = np.array(data['filenames'])
    face_targets = np_utils.to_categorical(np.array(data['target']), 7)
    return face_files, face_targets
```

```
In [226]: # Load the list of images and categories
train_faces, train_targets = load_dataset('./images-fewer/Train')
test_faces, test_targets = load_dataset('./images-fewer/Test')
validate_faces, validate_targets = load_dataset('./images-fewer/Validate')

face_names = [item[21:-1] for item in glob('./images-fewer/Train/*')]

print('There are %d face categories.' % len(face_names))
print(face_names)
print('There are %d total faces.' % len(np.hstack([train_faces, test_faces, validate_faces])))
print('There are %d training faces.' % len(train_faces))
print('There are %d test faces.' % len(test_faces))
print('There are %d validate faces.' % len(validate_faces))
```

There are 7 face categories.
['Brother', 'Dad', 'Daughter', 'Me', 'Mum', 'Son', 'Wife']
There are 263 total faces.
There are 148 training faces.
There are 58 test faces.
There are 57 validate faces.

```
In [40]: from keras.preprocessing import image

def path_to_tensor(img_path):
    img = image.load_img(img_path, target_size=(299,299))
    x = image.img_to_array(img)
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in img_paths]
    return np.vstack(list_of_tensors)
```

```
In [41]: # Read the images as numpy arrays
train_tensors = paths_to_tensor(train_faces).astype('float32')/255
test_tensors = paths_to_tensor(test_faces).astype('float32')/255
validate_tensors = paths_to_tensor(validate_faces).astype('float32')/255
print("Train tensor shape.", train_tensors.shape)
print('Test tensor shape.', test_tensors.shape)
print('Validate tensor shape.', validate_tensors.shape)
```

```
Train tensor shape. (148, 299, 299, 3)
Test tensor shape. (58, 299, 299, 3)
Validate tensor shape. (57, 299, 299, 3)
```

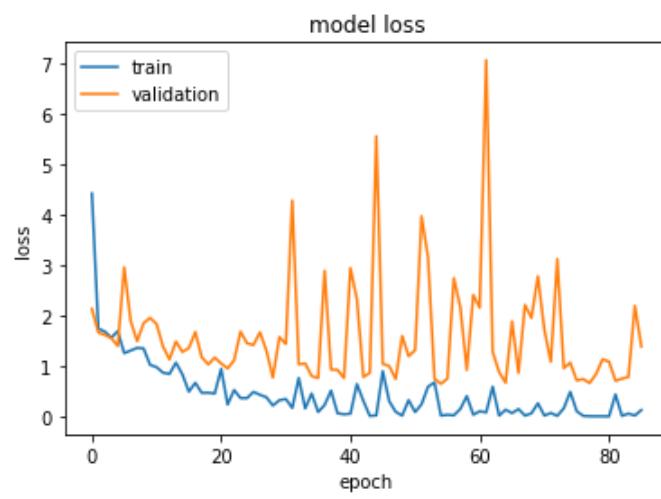
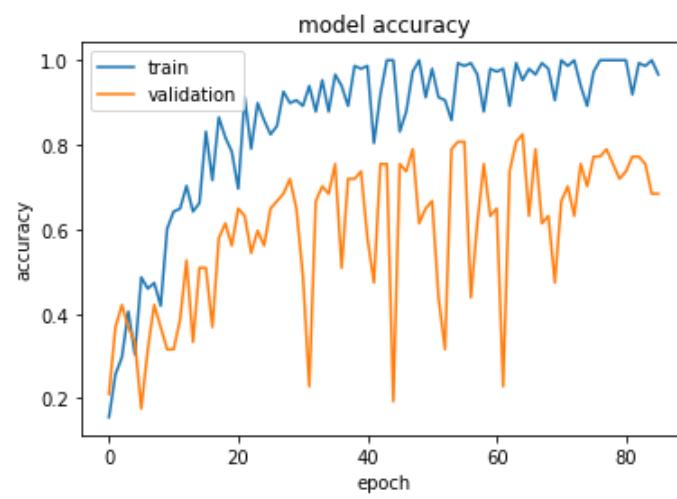
```
In [26]: my_model_1.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
checkpointer = ModelCheckpoint(filepath='my_model_1.h5',
                               verbose=1, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=30, verbose=1, mode='auto')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=10, cooldown=0, min_lr=0.00001)
lr_print = LambdaCallback(on_epoch_begin=lambda epoch, logs: print('lr:', K.eval(my_model_1.optimizer.lr)))
```

```
In [27]: hist_1 = my_model_1.fit(train_tensors, train_targets,
                           validation_data=(validate_tensors, validate_targets),
                           epochs=200, verbose=1, batch_size=20,
                           callbacks=[checkpointer, early_stopping, reduce_lr, lr_print])
```

```
In [28]: ## TODO: Visualize the training and validation Loss of your neural network
import matplotlib.pyplot as plt
def plt_hist(hist):
    print(hist.history.keys())
    # summarize history for accuracy
    plt.plot(hist.history['acc'])
    plt.plot(hist.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(hist.history['loss'])
    plt.plot(hist.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
```

```
In [29]: plt_hist(hist_1)
```

```
dict_keys(['loss', 'val_loss', 'val_acc', 'acc', 'lr'])
```



```
In [42]: my_model_1 = load_model('./my_model_1.h5')
score = my_model_1.evaluate(test_tensors, test_targets, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
58/58 [=====] - 3s
Test loss: 0.36949106126
Test accuracy: 0.844827617037
```

The testing accuracy is 84.5%, lower than I expected as there is only 7 classes. I tried to fine tune the number of layers and neurons in the dense layers, but didn't improve much. Maybe it's due to too limited number of images.

Part 3 - Improve accuracy with more images

More images were added to existing folder, just need to refresh the tensors and models.

```
In [79]: # Load the list of images and categories
train_faces, train_targets = load_dataset('./images/Train')
test_faces, test_targets = load_dataset('./images/Test')
validate_faces, validate_targets = load_dataset('./images/Validate')

face_names = [item[15:-1] for item in glob('./images/Train/*')]

print(face_names)
print('There are %d face categories.' % len(face_names))
print('There are %d total faces.' % len(np.hstack([train_faces, test_faces, validate_faces])))
print('There are %d training faces.' % len(train_faces))
print('There are %d test faces.' % len(test_faces))
print('There are %d validate faces.' % len(validate_faces))

['Brother', 'Dad', 'Daughter', 'Me', 'Mum', 'Son', 'Wife']
There are 7 face categories.
There are 428 total faces.
There are 257 training faces.
There are 98 test faces.
There are 73 validate faces.
```

```
In [82]: # Print a sample image
print(train_faces[0], train_targets[0])
print(np.argmax(train_targets[0]))
print(face_names[4])
```

```
./images/Train\Mum\20160805-2119.jpg-face-0.jpg [ 0.  0.  0.  0.  1.  0.  0.]
4
Mum
```

```
In [44]: # Read the images as numpy arrays, normalize
train_tensors = paths_to_tensor(train_faces).astype('float32')/255
test_tensors = paths_to_tensor(test_faces).astype('float32')/255
validate_tensors = paths_to_tensor(validate_faces).astype('float32')/255
print('Train tensor shape.', train_tensors.shape)
print('Test tensor shape.', test_tensors.shape)
print('Validate tensor shape.', validate_tensors.shape)
```

Train tensor shape. (257, 299, 299, 3)
 Test tensor shape. (98, 299, 299, 3)
 Validate tensor shape. (73, 299, 299, 3)

```
In [ ]: # It's exactly the same structure as my_model_1
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1000, activation = 'relu')(x)
x = Dense(200, activation = 'relu')(x)
x = Dense(50, activation = 'relu')(x)
x = Dense(7, activation = 'softmax')(x)

my_model_2 = Model(inputs = base_model.input, outputs = x)
my_model_2.summary()
```

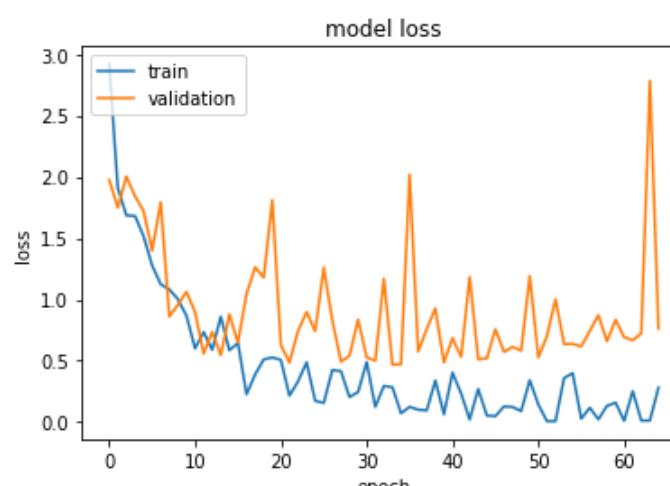
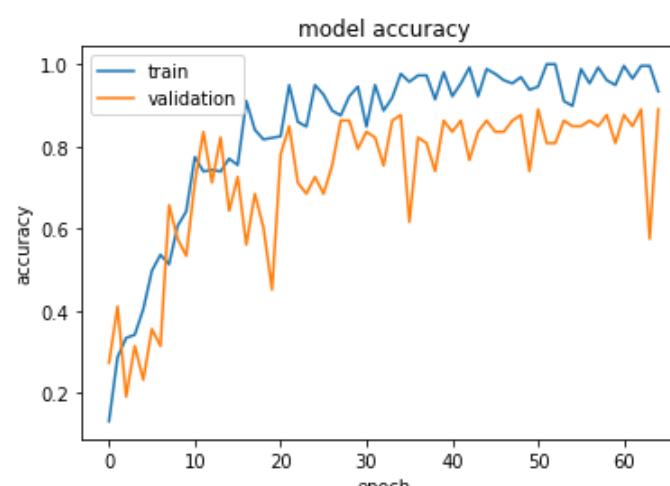
```
In [48]: my_model_2.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
checkpointer = ModelCheckpoint(filepath='my_model_2.h5',
                               verbose=1, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=30, verbose=1, mode='auto')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=10, cooldown=0, min_lr=0.00001)
lr_print = LambdaCallback(on_epoch_begin=lambda epoch, logs: print('lr:', K.eval(my_model_2.optimizer.lr)))
```

```
In [49]: hist_2 = my_model_2.fit(train_tensors, train_targets,
                           validation_data=(validate_tensors, validate_targets),
                           epochs=200, verbose=1, batch_size=20,
                           callbacks=[checkpointer, early_stopping, reduce_lr, lr_print])
```

lr: 0.000729
 Epoch 60/200
 240/257 [=====>..] - ETA: 0s - loss: 0.1392 - acc: 0.9542Epoch 00059: val_loss did not improve
 257/257 [=====] - 2s - loss: 0.1583 - acc: 0.9494 - val_loss: 0.8336 - val_acc: 0.8082
 lr: 0.000729
 Epoch 61/200
 240/257 [=====>..] - ETA: 0s - loss: 0.0112 - acc: 0.9958Epoch 00060: val_loss did not improve
 257/257 [=====] - 2s - loss: 0.0106 - acc: 0.9961 - val_loss: 0.6927 - val_acc: 0.8767
 lr: 0.000729
 Epoch 62/200
 240/257 [=====>..] - ETA: 0s - loss: 0.2693 - acc: 0.9625Epoch 00061: val_loss did not improve
 257/257 [=====] - 2s - loss: 0.2519 - acc: 0.9650 - val_loss: 0.6668 - val_acc: 0.8493
 lr: 0.000729
 Epoch 63/200
 240/257 [=====>..] - ETA: 0s - loss: 0.0131 - acc: 0.9958Epoch 00062: val_loss did not improve
 257/257 [=====] - 2s - loss: 0.0123 - acc: 0.9961 - val_loss: 0.7234 - val_acc: 0.8904
 lr: 0.000729
 Epoch 64/200
 240/257 [----->..] - ETA: 0s - loss: 0.0005 - acc: 1.0000Epoch 00063: val_loss did not improve

```
In [50]: plt_hist(hist_2)
```

```
dict_keys(['loss', 'val_loss', 'val_acc', 'acc', 'lr'])
```



```
In [52]: my_model_2 = load_model('./my_model_2.h5')
```

```
In [53]: score_2 = my_model_2.evaluate(test_tensors, test_targets, verbose=1)
print('Test loss:', score_2[0])
print('Test accuracy:', score_2[1])
```

96/98 [=====>.] - ETA: 0sTest loss: 0.32326669839
Test accuracy: 0.887755102041

```
In [54]: score_1 = my_model_1.evaluate(test_tensors, test_targets, verbose=1)
print('Test loss:', score_1[0])
print('Test accuracy:', score_1[1])
```

96/98 [=====>.] - ETA: 0sTest loss: 0.547816322774
Test accuracy: 0.816326530612

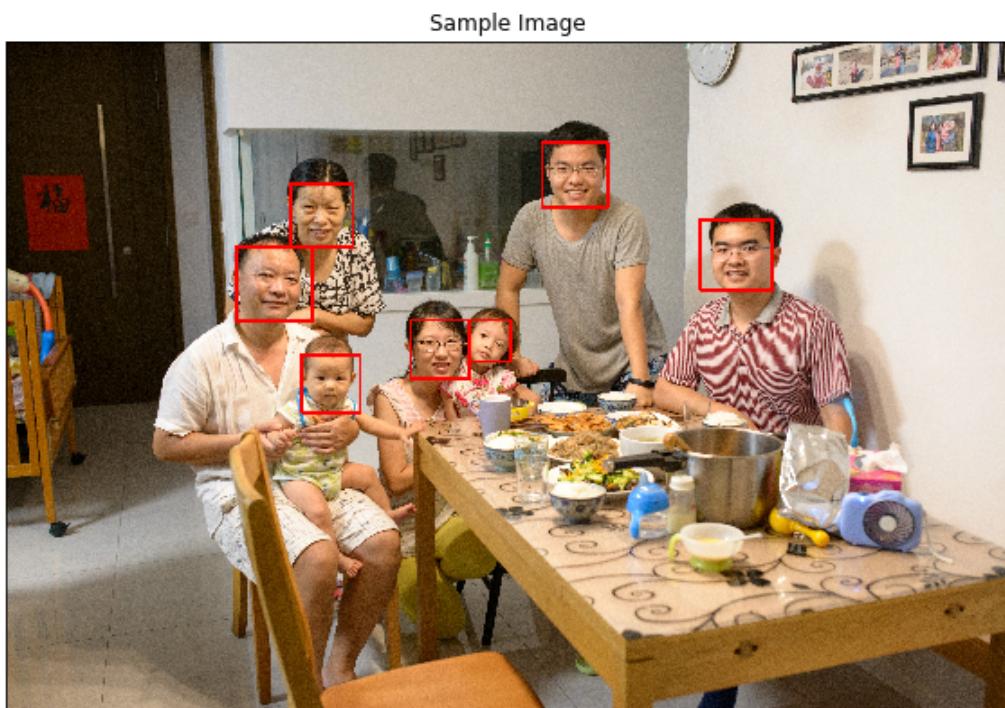
More data actually helps improve the accuracy. The second model shows 7% higher accuracy against the same set of testing data.

Part 4 - Face recognition on sample photos

There are two family photos 'test_1.jpg' and 'test_2.jpg' in the root of working directory. Both of them were not used for training or testing in the earlier sections. Let's first perform a face detection using the `show_faces()`.

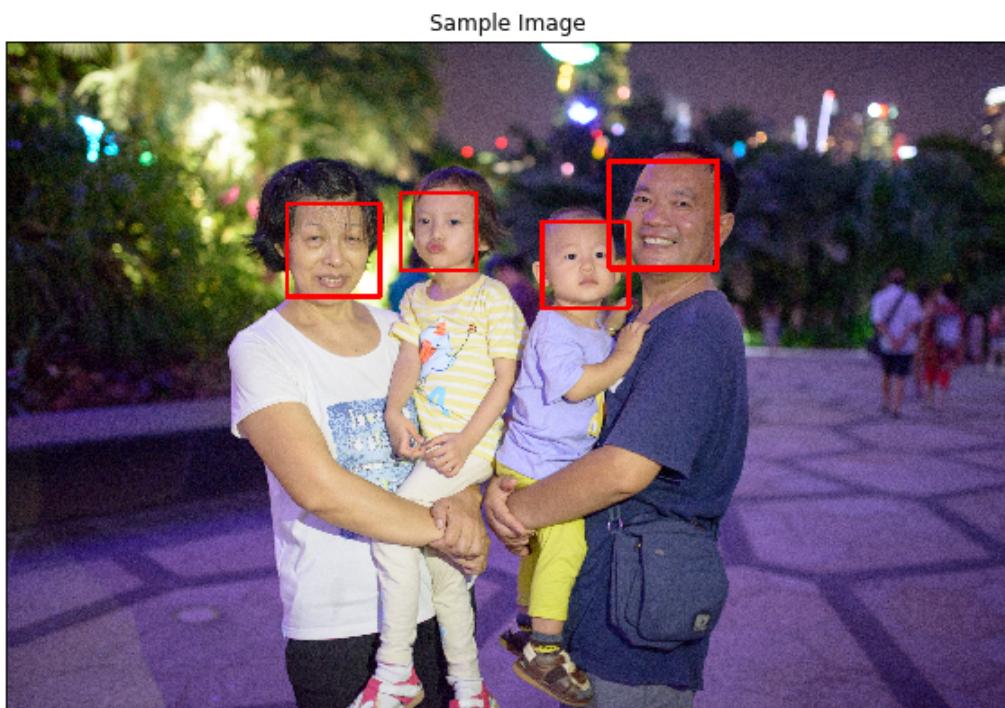
```
In [76]: show_faces('./test_1.jpg', True, False, 1.3, 7)
```

Image path ./test_1.jpg
(4912, 7360, 3)
Number of faces detected: 7



```
In [77]: show_faces('./test_2.jpg', True, False, 1.35, 5)
```

Image path ./test_2.jpg
(4766, 7141, 3)
Number of faces detected: 4



Use one photo from the testing dataset to predict and return the result as index of the `face_names`.

```
In [147]: one_test = test_tensors[56].reshape(-1, 299, 299, 3)
predict_idx = np.argmax(my_model_2.predict(one_test))
print(test_faces[56])
print(face_names)
print('The predicted face is: ', face_names[predict_idx])
```

```
./images/Test\ Dad\20110204-1825.jpg-face-0.jpg
['Brother', 'Dad', 'Daughter', 'Me', 'Mum', 'Son', 'Wife']
The predicted face is: Dad
```

Now let's make customized detect_faces() and recognize_faces()

```
In [158]: def recognize_face(image_face):
    # Convert the 3 channel RGB to 4-d tensor and normalize it
    image_face = image_face.reshape(-1, 299, 299, 3)/255
    predict_idx = np.argmax(my_model_2.predict(image_face))
    return face_names[predict_idx]
```

```
In [213]: def detect_faces(file_path, display=False, recognize=False, scaleFactor=1.3, minNeighb=5):
    print('Image path', file_path)

    # The file path contains unicode characters, cannot use cv2.imread() directly
    file_stream = open(file_path, 'rb')
    bytes_arr = bytearray(file_stream.read())
    numpy_ar = np.asarray(bytes_arr, dtype=np.uint8)
    image = cv2.imdecode(numpy_ar, cv2.IMREAD_UNCHANGED)
    print(image.shape)

    # Convert to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Convert the RGB image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # Extract the pre-trained face detector from an xml file
    face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_frontalface_default.xml')

    # Detect the faces in image
    faces = face_cascade.detectMultiScale(gray, scaleFactor, minNeighb)

    # Print the number of faces detected in the image
    print('Number of faces detected:', len(faces))

    # Make a copy of the orginal image to draw face detections on
    image_with_detections = np.copy(image)

    # The list of detected faces
    image_faces = []
    # Get the bounding box for each detected face
    for (x,y,w,h) in faces:
        # Add a red bounding box to the detections image
        if w > 200:
            line_width = w//20
        else:
            line_width = 3
        cur_face = image[y:(y+h), x:(x+w)]
        image_faces.append(cur_face)

        # Draw the red rectangle on the image
        cv2.rectangle(image_with_detections, (x,y), (x+w,y+h), (255,0,0), line_width)
        if recognize:
            cur_face = cv2.resize(cur_face, (299,299))
            name = recognize_face(cur_face)

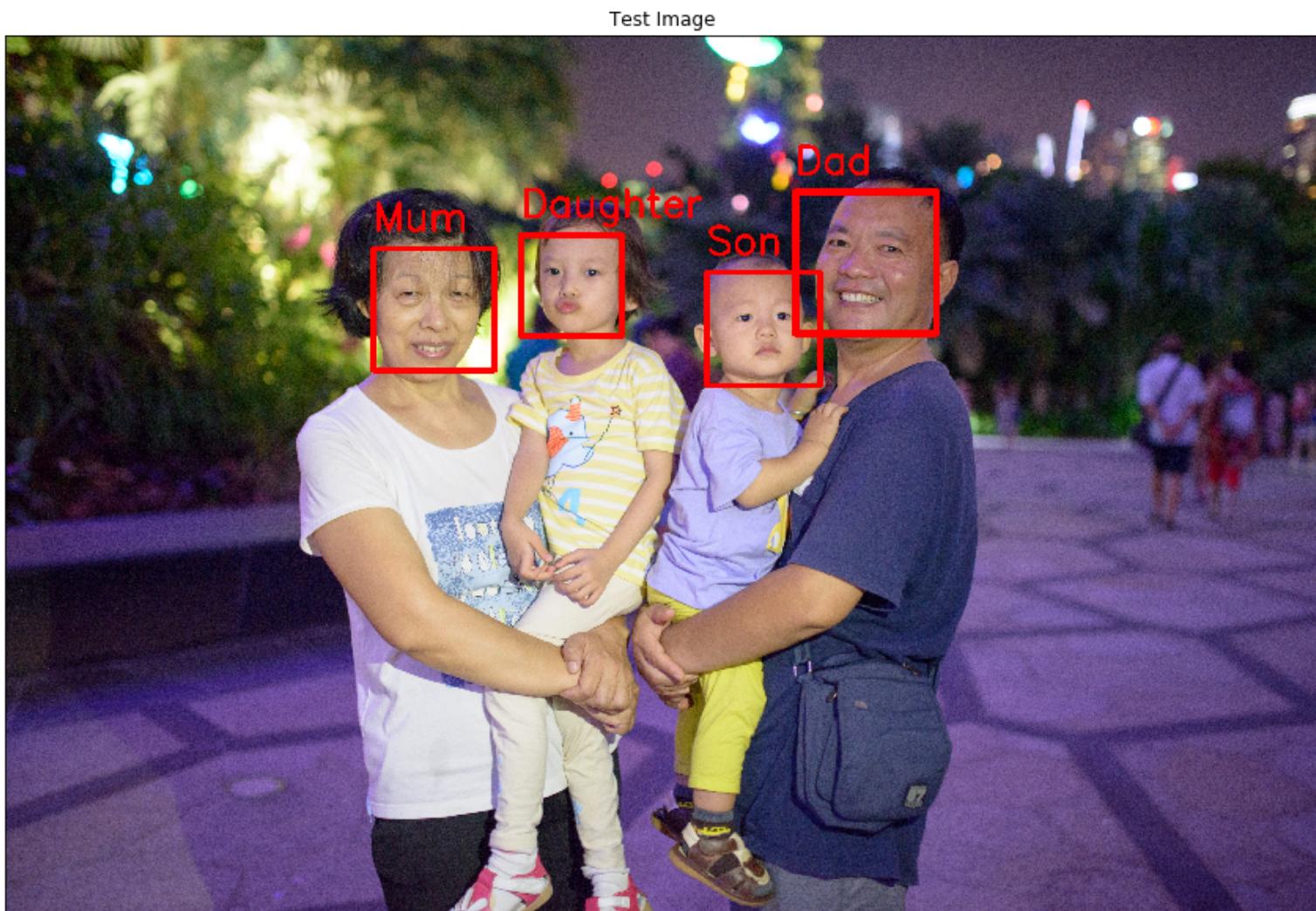
            print('x,y,w,h', x,y,w,h)
            print(name)

            # Write the returned name on the image
            cv2.putText(image_with_detections, name,
                       (x,y-100),cv2.FONT_HERSHEY_SIMPLEX, 7, (255,0,0),20,cv2.LINE_AA)

    if display:
        # Display the image with the detections
        fig = plt.figure(figsize=(15, 15))
        ax = fig.add_subplot(1, 1, 1, xticks=[], yticks[])
        ax.set_title('Test Image')
        ax.imshow(image_with_detections)
        os.chdir(cur_dir)
```

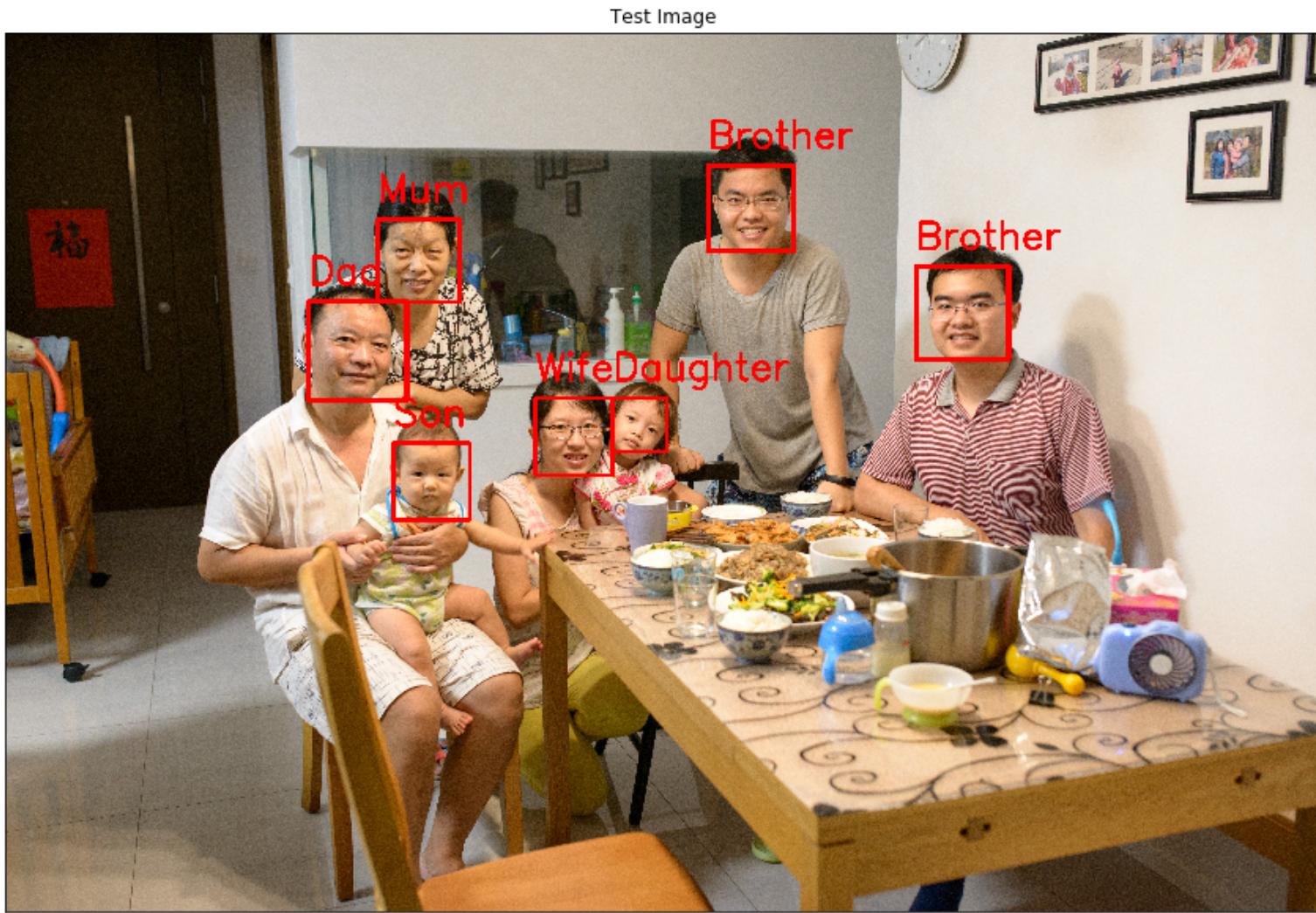
```
In [214]: detect_faces('./test_2.jpg', True, True, 1.35, 5)
```

```
Image path ./test_2.jpg  
(4766, 7141, 3)  
Number of faces detected: 4  
x,y,w,h 1997 1163 658 658  
Mum  
x,y,w,h 2798 1090 546 546  
Daughter  
x,y,w,h 3805 1292 614 614  
Son  
x,y,w,h 4285 857 765 765  
Dad
```



```
In [212]: detect_faces('./test_1.jpg', True, True, 1.3, 7)
```

```
Image path ./test_1.jpg  
(4912, 7360, 3)  
Number of faces detected: 7  
x,y,w,h 2080 1051 459 459  
Mum  
x,y,w,h 3923 750 477 477  
Brother  
x,y,w,h 2961 2047 430 430  
Wife  
x,y,w,h 1693 1509 550 550  
Dad  
x,y,w,h 2166 2300 427 427  
Son  
x,y,w,h 5083 1313 518 518  
Brother  
x,y,w,h 3392 2045 302 302  
Daughter
```



And now I finally have a 'theoretical' evidence that how much I look like my brother. Despite we're not twins, many friends say we look 'exactly' like twins!!!

```
In [ ]:
```