Import Libraries

```
%matplotlib inline
import os, sys, glob, pickle, shutil
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageOps, ImageFilter
from keras.models import load_model
from keras.preprocessing.image import *
from keras.utils.vis_utils import plot_model
from keras.models import Model
from keras.layers import Input
```

Load the Trained Deep Learning Model

```
model = load_model('./models/vgg_face.h5')
model.summary()
```

```
Model: "vgg_face_descriptor"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1_1 (Conv2D)             (None, 224, 224, 64)      1792
_________________________________________________________________
conv1_2 (Conv2D)             (None, 224, 224, 64)      36928
_________________________________________________________________
zero_padding2d_1             (None, 226, 226, 64)      0
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 112, 112, 64)      0
_________________________________________________________________
zero_padding2d_2             (None, 114, 114, 64)      0
_________________________________________________________________
conv2_1 (Conv2D)             (None, 112, 112, 128)     73856
_________________________________________________________________
conv2_2 (Conv2D)             (None, 112, 112, 128)     147584
_________________________________________________________________
zero_padding2d_3             (None, 114, 114, 128)     0
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 56, 56, 128)       0
_________________________________________________________________
zero_padding2d_4             (None, 58, 58, 128)       0
_________________________________________________________________
conv3_1 (Conv2D)             (None, 56, 56, 256)       295936
_________________________________________________________________
conv3_2 (Conv2D)             (None, 56, 56, 256)       590080
_________________________________________________________________
conv3_3 (Conv2D)             (None, 56, 56, 256)       590080
_________________________________________________________________
zero_padding2d_5             (None, 58, 58, 256)       0
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 28, 28, 256)       0
_________________________________________________________________
zero_padding2d_6             (None, 30, 30, 256)       0
_________________________________________________________________
conv4_1 (Conv2D)             (None, 28, 28, 512)       1180160
_________________________________________________________________
conv4_2 (Conv2D)             (None, 28, 28, 512)       2359808
_________________________________________________________________
conv4_3 (Conv2D)             (None, 28, 28, 512)       2359808
_________________________________________________________________
zero_padding2d_7             (None, 30, 30, 512)       0
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 14, 14, 512)       0
_________________________________________________________________
zero_padding2d_8             (None, 16, 16, 512)       0
_________________________________________________________________
conv5_1 (Conv2D)             (None, 14, 14, 512)       2359808
_________________________________________________________________
conv5_2 (Conv2D)             (None, 14, 14, 512)       2359808
_________________________________________________________________
conv5_3 (Conv2D)             (None, 14, 14, 512)       2359808
_________________________________________________________________
zero_padding2d_9             (None, 16, 16, 512)       0
_________________________________________________________________
pool5 (MaxPooling2D)         (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 4096)              102764544
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16777216
_________________________________________________________________
dense_2 (Dense)              (None, 2622)              10739838
_________________________________________________________________
activation_1 (Activation)    (None, 2622)              0
=================================================================
Total params: 145,002,878
Trainable params: 145,002,878
Non-trainable params: 0
_________________________________________________________________
```

```
for layer in model.layers[:5]:
    print(layer.name)
    layer.trainable = False
model.summary()
```

```
conv1_1
conv1_2
zero_padding2d_1
pool1
zero_padding2d_2
```

Extract Face Function

```
def extract_face(filename, required_size=(224, 224)):
    ''' load image from file
    image = Image.open(filename)
    image = image.convert('RGB')
    pixels = np.asarrayimage)
    ''' detect faces in the image
    results = detector.detect_faces(pixels)
    ''' extract the bounding box from the first face
    x1, y1, width, height = results[0]['box']
    x2, y2 = x1 + width, y1 + height
    ''' extract the face
    face = pixels[y1:y2, x1:x2]
    ''' resize pixels to the model size
    image = Image.fromarray(face)
    image = image.resize(required_size)
    face_array = np.asarrayimage)
    return face_array
```

```
[Errorno_2_image_1_face_0]
```

Code Explanation

The extract_face function is used to extract the face from the image using the MTCNN face detector library and resize them to match the input shape of our model. This way, the image inputted to the model will always be of the same shape.

Category 1 - Basic / Rough Classification Accuracy

```
[Errorno_2_image_1_face_0]
def load_image(filename, required_size=(224, 224)):
    ''' load image from file
    image = Image.open(filename)
    image = image.convert('RGB')
    pixels = np.asarrayimage)
    ''' resize pixels to the model size
    image = image.resize(required_size)
    face_array = np.asarrayimage)
    return face_array
[Errorno_2_image_1_face_0]
def get_embeddings(filenames):
    ''' extract faces
    faces = [extract_face(f) for f in filenames]
    ''' convert into an array of samples
    samples = np.asarray(faces, 'float32')
    ''' prepare the face for the model, subtract pixel mean
    samples = preprocess_input(samples, version=2)
    ''' perform prediction
    model = vgg_face_descriptor
    yhat = model.predict(samples)
    return yhat
[Errorno_2_image_1_face_0]
def is_match(known_embedding, candidate_embedding, thresh=0.5):
    ''' calculate distance between embeddings
    score = cosine(known_embedding, candidate_embedding)
    if score <= thresh:
        print('>%.3f' % (score))
    return score
```

```
>0.000
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

Category 2 - Refined / Fine Tune Accuracy

The Members

```
[Errorno_2_image_1_face_0]
# load faces
members = dir('./data/members/*.jpg')
print('Loaded %d members' % len(members))
members = [load_image(f) for f in members]
print('Loaded %d members faces' % len(members), members[0].shape)
# calculate embeddings for faces
members = get_embeddings(members)
# enforce unit norm
members = [vec / np.linalg.norm(vec) for vec in members]
# perform one-shot classification for each face
threshold = 0.5
for i in range(len(members)):
    print('>%.3f' % is_match(members[i], members[i], threshold))
```

```
Loaded 3 members
Loaded 3 members faces (224, 224, 3)
>0.000
>0.000
>0.000
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

Test Case 1: Background Environment

```
[Errorno_2_image_1_face_0]
# load faces
members = dir('./data/members/*.jpg')
print('Loaded %d members' % len(members))
members = [load_image(f) for f in members]
print('Loaded %d members faces' % len(members), members[0].shape)
# calculate embeddings for faces
members = get_embeddings(members)
# enforce unit norm
members = [vec / np.linalg.norm(vec) for vec in members]
# perform one-shot classification for each face
threshold = 0.5
for i in range(len(members)):
    print('>%.3f' % is_match(members[i], members[i], threshold))
```

```
Loaded 3 members
Loaded 3 members faces (224, 224, 3)
>0.000
>0.000
>0.000
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```



```
[Errorno_2_image_1_face_0]
```

Findings

As shown by the output above, the model can recognize the images with different background environments as the same person. This is due to the nature of the pre-trained face models which will always capture the face and ignore the background environment.

Test Case 2: Size Variations

```
[Errorno_2_image_1_face_0]
# load faces
members = dir('./data/members/*.jpg')
print('Loaded %d members' % len(members))
members = [load_image(f) for f in members]
print('Loaded %d members faces' % len(members), members[0].shape)
# calculate embeddings for faces
members = get_embeddings(members)
# enforce unit norm
members = [vec / np.linalg.norm(vec) for vec in members]
# perform one-shot classification for each face
threshold = 0.5
for i in range(len(members)):
    print('>%.3f' % is_match(members[i], members[i], threshold))
```

```
Loaded 3 members
Loaded 3 members faces (224, 224, 3)
>0.000
>0.000
>0.000
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

Findings

As shown by the output above, the model can recognize the images with different image sizes as the same person. This is due to the nature of the model. The model is used to a fixed size, so all of the inputs will always be resized to the same input shape.

Test Case 3: Orientation Perspective

```
[Errorno_2_image_1_face_0]
# load faces
members = dir('./data/members/*.jpg')
print('Loaded %d members' % len(members))
members = [load_image(f) for f in members]
print('Loaded %d members faces' % len(members), members[0].shape)
# calculate embeddings for faces
members = get_embeddings(members)
# enforce unit norm
members = [vec / np.linalg.norm(vec) for vec in members]
# perform one-shot classification for each face
threshold = 0.5
for i in range(len(members)):
    print('>%.3f' % is_match(members[i], members[i], threshold))
```

```
Loaded 3 members
Loaded 3 members faces (224, 224, 3)
>0.000
>0.000
>0.000
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

Findings

As shown by the output above, the model does hamper in not able to recognize faces in top-this and the model is able to identify the person correctly.

Test Case 4: Age

```
[Errorno_2_image_1_face_0]
# load faces
members = dir('./data/members/*.jpg')
print('Loaded %d members' % len(members))
members = [load_image(f) for f in members]
print('Loaded %d members faces' % len(members), members[0].shape)
# calculate embeddings for faces
members = get_embeddings(members)
# enforce unit norm
members = [vec / np.linalg.norm(vec) for vec in members]
# perform one-shot classification for each face
threshold = 0.5
for i in range(len(members)):
    print('>%.3f' % is_match(members[i], members[i], threshold))
```

```
Loaded 3 members
Loaded 3 members faces (224, 224, 3)
>0.000
>0.000
>0.000
```

```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```



```
[Errorno_2_image_1_face_0]
```



```
[Errorno_2_image_1_face_0]
```

Findings

A person who passes a 2016 during high school graduation ceremony 2 years ago is used here. The model is still able to identify the correctly.

Test Case 5: Hairstyle

```
[Errorno_2_image_1_face_0]
# load faces
members = dir('./data/members/*.jpg')
print('Loaded %d members' % len(members))
members = [load_image(f) for f in members]
print('Loaded %d members faces' % len(members), members[0].shape)
# calculate embeddings for faces
members = get_embeddings(members)
# enforce unit norm
members = [vec / np.linalg.norm(vec) for vec in members]
# perform one-shot classification for each face
threshold = 0.5
for i in range(len(members)):
    print('>%.3f' % is_match(members[i], members[i], threshold))
```

```
Loaded 3 members
Loaded 3 members faces (224, 224, 3)
>0.000
>0.000
>0.000
```



```
[Errorno_2_image_1_face_0]
```



```
[Errorno_2_image_1_face_0]
```

```
[Errorno_2_image_1_face_0]
```

Findings

Unfortunately, we don't have images of us with beards (mainly because we don't really have beard, so we ended with a picture of Sirio who's Instagram based filter. The pictures of members on different hairstyles is also tested here and the same is tested for the accuracy.