Final project: Facial Detection

Group member: Hanxiao Yan N16242797, Zijun Huang N11300608, Huai Lin N12507166 ¶

In [91]:

```
import keras
```

In [92]:

```
import tensorflow as tf
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense, Conv2D
import numpy as np
import os
import pickle
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline
```

Introduction:

Background: In recent years, face recognition problem has been more and more popular and attractive to nearly every single person in modern society. Face recognition system is a computer application capable of identifying or verifying a person from a digital image or a video frame from a video source. One of the ways to do this is by comparing selected facial features from the image and a face database. Since we have completed this amazing introduction to machine learning course, we would like to challenge ourselves to accomplish facial detection problem in this project.

Object of this project: Accomplish facial detection implementation by using machine learning knowledges, especially convolutional neural network.

Our idea: At first, we want to create a fully connected CNN(no FC layer at all), and then train this network on small images and set up CNN to output a 1x1x1 output. Then run the CNN on larger image, it will output the score for each box in the input image. However, we don't know how to create multi-dimensional labels (y), thus we used FC layers after CNN layers as instead. In this way, we can generate 1-D label very easy. So, following this idea, we begin to build our model as the following steps:

- step1: Transfer learning from well-known model (VGG-16), this could save our time to train model from scrach.
- step2: Add our own layers (FC layers) to VGG-16 and obtain our model.
- step3: Download human pictures and generate face images as part of training samples.
- step4: Training model

After model being built, we try to do:

- step1: Single face detection
- step2: Multi face detection

From the obtained results of singel/multi face dectection, we found our model is not accurate and efficient enough, thus we studied additional knowlege from Coursera deep learning course, and this course introduce YOLO model to us.

Data details:

In our project, we use:

- The input of training data is a batch of images of shape (m, 64, 64, 3), m samples and each is 64(Height)x64(Width)x3(RGB). Label y=1, non-face. Label y=0, face.
- The output of model is a score(decimal number) which represent the probabilty that whether the corresponding input is a face or an non-face.
- The input of testing data is one image of shape (512, 512, 3)

Ⅰ - Build model

1-Transfer learning

Limited to our computation power, we need to use pre-trained model, and we select well-known model VGG-16.

In [142]:

```
# Clear session at first
keras.backend.clear_session()
# Load the VGG16 network
input_shape = (64, 64, 3)
base_model = applications.VGG16(weights='imagenet',
                                include_top=False, input_shape=input_shape)
```

In [143]:

```
# Create a new model, but freeze the weights in VGG-16
model = Sequential()
# Loop over base_model.layers and add each layer to model
for layer in base_model.layers:
    model.add(layer)
for layer in model.layers:
    layer.trainable=False
```

2 - Add additional layers to VGG-16

In [144]:

```
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

3 - Load training data

Use flicker API and Google downloading human images, then generate face images. In training images, there are 215 face class images and 265 non-face images. We then upload our own pictures to test our facial recognition.

In [145]:

```
train_data_dir = './train'
batch_size = 50
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
                        train_data_dir,
                        target_size=(64,64),
                        batch_size=batch_size,
                        class_mode='binary')
```

```
Found 480 images belonging to 2 classes.
```

In [146]:

```
test_data_dir = './test'
batch_size = 1
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
                        test_data_dir,
                        target_size=(512,512),
                        batch_size=batch_size,
                        class_mode='binary')
```

```
Found 1 images belonging to 1 classes.
```

Extract testing data

In [ ]:

```
Xtest = test_generator.next()[0]
```

3.1 -Display Image

In [147]:

```
# Display the image
def disp_image(im):
    if (len(im.shape) == 2):
        # Gray scale image
        plt.imshow(im, cmap='gray')
    else:
        # Color image.
        im1 = (im-np.min(im))/(np.max(im)-np.min(im))*255
        im1 = im1.astype(np.uint8)
        plt.imshow(im1)

    # Remove axis ticks
    plt.xticks([])
    plt.yticks([])
```

3.2 - Check training images

In [148]:

```
X = train_generator.next()
plt.figure(figsize=(10,10))
nplot = 8
for i in range(nplot):
    plt.subplot(1,nplot,i+1)
    disp_image(X[0][i,:,:,:])
    if X[1][i].astype(int)==1:
        plt.xlabel('nonface')
    elif X[1][i].astype(int)==0:
        plt.xlabel('face')
```

4 - Train model

4.1 - Select optimizer and compile model

In [149]:

```
model.compile(optimizer=optimizers.Adam(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

4.2 - Use training data train our model

In [150]:

```
model.fit(x=X[0], y=X[1])
```

```
Epoch 1/10
50/50 [==============================] - 4s - loss: 0.7098 - acc: 0.6000
Epoch 2/10
50/50 [==============================] - 3s - loss: 0.4516 - acc: 0.9000
Epoch 3/10
50/50 [==============================] - 3s - loss: 0.3001 - acc: 0.8800
Epoch 4/10
50/50 [==============================] - 3s - loss: 0.1929 - acc: 0.9400
Epoch 5/10
50/50 [==============================] - 3s - loss: 0.1587 - acc: 0.9400
Epoch 6/10
50/50 [==============================] - 3s - loss: 0.0973 - acc: 0.9800
Epoch 7/10
50/50 [==============================] - 3s - loss: 0.0661 - acc: 0.9800
Epoch 8/10
50/50 [==============================] - 3s - loss: 0.0537 - acc: 1.0000
Epoch 9/10
50/50 [==============================] - 3s - loss: 0.0225 - acc: 1.0000
Epoch 10/10
50/50 [==============================] - 4s - loss: 0.0222 - acc: 1.0000
```

Out[150]:

```
<keras.callbacks.History at 0x2d6c05ac6a0>
```

Ⅱ - Single Facial Detection

Since we has already trained our model with face image, which has size 64x64x3, and also we appended FC layers instead of CNN layers, we can't put the test image (which has size 512x512x3) derectly into our model and do convolution(if we didn't use FC layers as the last layer in our model but CNN layer, we can just directly put the test image, no matter what size it is, and obtain output scoce very fast through convolution). Now that we have to crop the test image (512x512x3) into many parts, each part has size 64x64x3. When cropped image, we use stride=4.Then put each part of test image into model and get scores. Finally, we select the part of test image we cropped (which we can also call "bounding box", because we are going to use this box to highlight face area) of the lowest probability (since we label face as 0), and draw this box on original test image.

- Step1: get the score of each bounding box
- Step2: draw the bounding box with best probability being a face on original test image
- Step3: draw heatmap

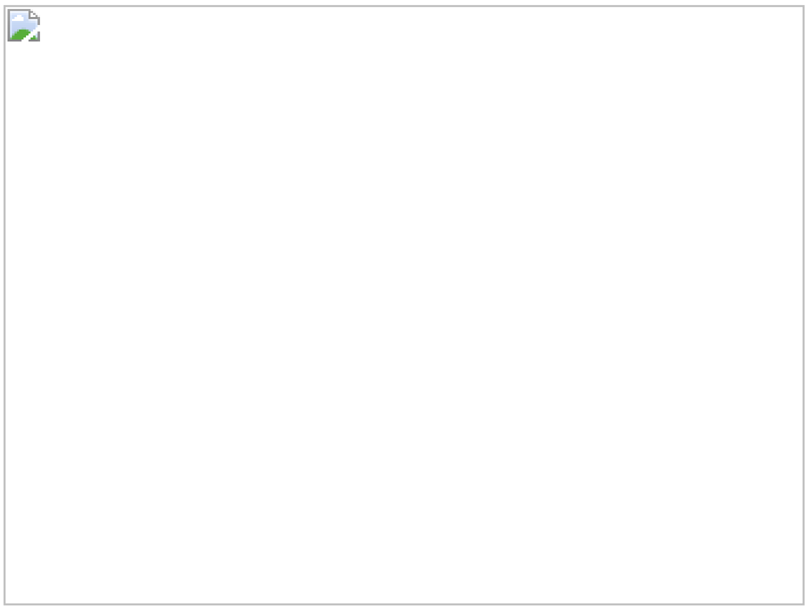
1 - Get the score of each bounding box.

We use a simple example to demonstrate our idea:

1. To select a 2x2 part of original test image at the upper left corner of a matrix "test_img" (shape (5,5,3)), we would do:

   ```
   pic = test_img[0:2,0:2,:]
   ```

   This will be useful when we define `test_img` below, using the `start/end` indexes we define.
2. To find a particular part of the test image we need first define its corners `vert_start`, `vert_end`, `horiz_start` and `horiz_end`.



In [158]:

```
def get_box(test_img, output_size, stride):
    import time

    probs = [] # Record score for each part of test image
    coordinates = [] # Record location for each part of test image

    start = time.time()
    for row in range(output_size):
        for col in range(output_size):

            horiz_star = col*stride
            horiz_end = col*stride+64
            vert_star = row*stride
            vert_end = row*stride+64

            pic = test_img[:, vert_star:vert_end, horiz_star:horiz_end, :]
            probability = model.predict(pic)
            coordinates.append([row*4, col*4])
            probs.append(probability)
        if row%10==0:
            # since it's going to take a long time, we print out the process.
            print('In process-------:{0:d}/{1:d}'.format(row, output_size))
    end = time.time()
    running_time = end-start

    return probs, coordinates, running_time
```

In [153]:

```
# Save the output probabilities with corresponding coordinates.
def save_data(img_name, probs, coordinates):
    with open("{0:s}_probs.p".format(img_name), "wb") as fp:
        pickle.dump(probs, fp)
    with open("{0:s}_coordinate.p".format(img_name), "wb") as fp1:
        pickle.dump(coordinates, fp1)
```

2 - Draw bounding box

In [154]:

```
def draw_box(probs, coord, img):
    # Get the index of the box which has the highest probability to be a face
    # Since 0 represent face class, thus we use min() to get index
    imin = probs.index(min(probs))
    row, col = coord[imin]
    
    fig, ax = plt.subplots(figsize=(5, 5))
    ax.imshow(img[:,:,:])
    rect = mpatches.Rectangle((col, row), 64, 64, fill=False, 
                              edgecolor='red', linewidth=2)
    ax.add_patch(rect)
    plt.show()
```

3 - Draw heatmap

In [155]:

```
def draw_heatmap(probs, output_size):
    arr = np.asarray(probs)[:,0,0]
    Arr = np.reshape(arr, (output_size,output_size))
    plt.imshow(Arr,interpolation='none')
    plt.colorbar()
```

4 - Wrap up all our previous methods and try single face recognition

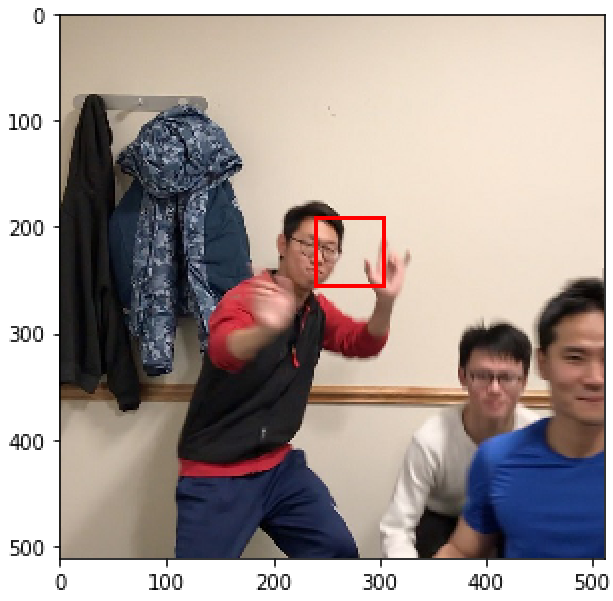
In [156]:

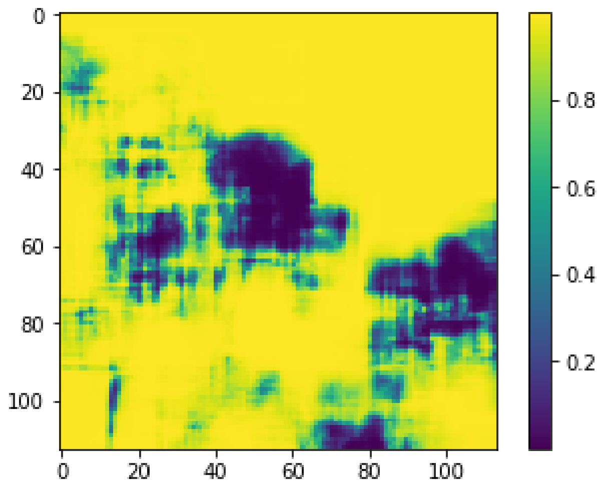
```
def single_face_detection(img, img_name, img_size, stride, Do_facial=False):
    if Do_facial==True:
        output_size = int((img_size - 64)/stride) + 1
    
        # Step1, get the bounding box
        probs, coordinates, running_time = get_box(img, output_size, stride)
    
        # Step2, save data to disk
        save_data(img_name, probs, coordinates)
        
        # Step3, draw the bounding box
        draw_box(probs, coordinates, img[0])
    
        # Step4, draw the heapmap
        draw_heatmap(probs, output_size)
```

In [157]:

```
single_face_detection(Xtest, '3 members 10', 512, 4, Do_facial=True)
```

```
In process-------:0/113
In process-------:10/113
In process-------:20/113
In process-------:30/113
In process-------:40/113
In process-------:50/113
In process-------:60/113
In process-------:70/113
In process-------:80/113
In process-------:90/113
In process-------:100/113
In process-------:110/113
```





Now, we have our single face detection and corresponding heatmap

Ⅲ - Multi-face Detection

Since the method above only picks the minimum probability, thus it only draws one face, can't apply to image with more than one face. In order to improve our facial detection ability, we would introduce IoU technique in the following code.

1 -Filtering bounding box with a threshold on probability

We are going to apply a first filter by thresholding. We would like to get rid of any box for which the probability is less than a chosen threshold (in our case, we chose threashold=0.2).

In our project, we define:

- a box using its two corners (upper left and lower right): (x1, y1, x2, y2)
- box_probability: the probability to be a face of this box

In [160]:

```
# Since we define a way to represent box location(using its upper left and lower right cornor) r
ather than
# upper left cornor which we got from coordinate list, we define a method to convert one from ot
her.
def coord_convert_box(coord):
    box = []
    for c in coord:
        box.append((c[1], c[0], c[1]+64, c[0]+64))
    return box
```

In [161]:

```
# Filters boxes by threasholding on faces and its probability.
def filter_boxes(box_probability, boxes, threshold=0.2):

    filtered_box_probability = []
    filtered_boxes = []

    for i, prob in enumerate(box_probability):
        if prob <= threshold:
            filtered_box_probability.append(prob)
            filtered_boxes.append(boxes[i])

    return filtered_box_probability, filtered_boxes
```

2- Non-max suppression

Even after filtering by thresholding over the probability, we still end up a lot of overlapping boxes. A second filter for selecting the right boxes is called non-maximum suppression.

Non-max suppression uses the function called **"Intersection over Union"**, or IoU.

- To calculate the area of a rectangle we need to multiply its height (y2 - y1) by its width (x2 - x1)
- We'll also define coordinates (xi1, yi1, xi2, yi2) of the intersection of two boxes.
    - xi1 = maximum of the x1 coordinates of the two boxes
    - yi1 = maximum of the y1 coordinates of the two boxes
    - xi2 = minimum of the x2 coordinates of the two boxes
    - yi2 = minimum of the y2 coordinates of the two boxes

In [80]:

```
def IoU(box1, box2):
    xi1 = np.maximum(box1[0], box2[0])
    yi1 = np.maximum(box1[1], box2[1])
    xi2 = np.minimum(box1[2], box2[2])
    yi2 = np.minimum(box1[3], box2[3])
    inter_area = (yi2-yi1)*(xi2-xi1)
    
    box1_area = (box1[3]-box1[1])*(box1[2]-box1[0])
    box2_area = (box2[3]-box2[1])*(box2[2]-box2[0])
    union_area = box1_area + box2_area - inter_area
    
    iou = inter_area/union_area
    return iou
```

Now we have IoU and then ready to implement non-max suppression. The key steps are:

1. Select the box that has the highest probability(lowest box_probability in our case) if list is not empty.
2. Compute its overlap with all other boxes, and remove boxes that overlap it more than iou_threshold.
3. Go back to step 1 and iterate until there's no more boxes with a lower probability than the current selected box.

In [162]:

```
# This will remove all boxes that have a large overlap with the selected boxes.

def face_boxes(box_probability, boxes):
    facial_boxes = []
    facial_probability = []
    
    while box_probability:
        # Get the value and index of the minimum box probability.
        min_prob = min(box_probability)
        imin = box_probability.index(min_prob)
        curr_box = boxes[imin]
        
        # Append corresponding box and probability to facial boxes 
        # waiting to be drawn in the future.
        facial_boxes.append(boxes[imin])
        facial_probability.append(min_prob)
        
        # Remove this probability and box at its index in related lists.
        box_probability.remove(min_prob)
        boxes.remove(boxes[imin])
        
        low_iou_boxes = []
        low_iou_probs = []
        for i, box in enumerate(boxes):
            iou = IoU(curr_box, box)
            if iou<0.1:
                low_iou_boxes.append(box)
                low_iou_probs.append(box_probability[i])
        
        # Update box probability and boxes
        box_probability = low_iou_probs
        boxes = low_iou_boxes
    
    return facial_probability, facial_boxes
```

3 - Multi-face draw method

Implement a method displaying the output(512x512x3) of the deep CNN and filtering through all the boxes using the methods we've just implemented.

In [163]:

```python
def multi_face_draw_box(probs, coord, img, threshold=0.2, Do_multi_face=False):

    if Do_multi_face==True:

        # Step1: convert coordinates into boxes, which has upper left and lower right corner
        boxes = coord_convert_box(coord)

        # Step2: eliminate any boxes which has low probability to be a face
        filtered_box_probability, filtered_boxes = filter_boxes(box_probability=probs,
                                                                boxes=boxes,
                                                                threshold=threshold)

        # Step3: get rid of overlapped boxes and output the clean face boxes
        facial_probability, facial_boxes = face_boxes(filtered_box_probability,
                                                      filtered_boxes)

        fig, ax = plt.subplots(figsize=(5, 5))
        ax.imshow(img[:,:,:])

        for face in facial_boxes:
            rect = mpatches.Rectangle((face[0], face[1]), 64, 64, fill=False,
                                      edgecolor='red', linewidth=2)
            ax.add_patch(rect)
        plt.show()
```

In [164]:

```python
with open("3_members_10_probs.p", "rb") as fp:
    two_probs = pickle.load(fp)
with open("3_members_10_coordinate.p", "rb") as fp1:
    two_coordinate = pickle.load(fp1)
```

In [168]:

```python
multi_face_draw_box(two_probs, two_coordinate, Xtest[0], 0.003, True)
```