Face Detection (Convolution) (Variable image sizes)

This piece of Python code uses *keras* to find if a face is front or side!

We have just 200 images (100 front and 100 side view) of any size, and with a *keras* CNN, we can achieve 90% accuracy rate. We fix the size of images, and then convert them to gray. However our model learns pretty well with really a small size for the training set (80% * 200 = 160 images).

Import the libraries

In [1]:

```
from os.path import join
import os

# import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.preprocessing.image import load_img, array_to_img
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import cv2            # We have installed opencv-python, this library read the images in opencv formart
from PIL import Image # We have installed Pillow
                      # and this library is needed to convert the opencv format image to PIL (color) format

from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import array_to_img

from sklearn import model_selection
```

Loading different size images

In [254]:

```
my_path = r'C:\Aarhus\Learning\Face Detection'

def rgb2gray(rgb):                               # This function converts colored images to gray ones
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

def display_img(im_pil, rgb_im, side=True):     # This function prints images
    dpi = 80
    height, width, depth = rgb_im.shape
    figsize = width / float(dpi), height / float(dpi)
    fig = plt.figure(figsize=figsize)
    ax = fig.add_axes([0, 0, 1, 1])
    ax.axis('off')
    if side == True: ax.set_title('Resized side view sample')
    else: ax.set_title('Resized front view sample')
    ax.imshow(im_pil)
    plt.show()

def display_gray(np_array, title, mis = False): # This function prints gray format images
    dpi = 80
    height, width = np_array.shape
    figsize = width / float(dpi), height / float(dpi)
    fig = plt.figure(figsize = figsize); ax = fig.add_axes([0, 0, 1, 1]); ax.axis('off'); ax.set_title(title)
    if mis == False:
        ax.imshow(np_array, interpolation='nearest', cmap=plt.get_cmap('gray'), vmin=0, vmax=255)
    else:
        ax.imshow(np_array, interpolation='nearest', cmap=plt.get_cmap('gray'), vmin=0, vmax=1)
    plt.show()

def read_files(my_path): # The function reads all images (Front and Side) and returns a numpy array of images
    im_list = []; y = []

    min_h = 1000; min_w = 1000
    read_side = False; read_front = False
    for file in os.listdir(my_path + '\Side'):
        im_pil = Image.open(join(my_path + '\Side', file))         # reading images using PIL

        if read_side == False:                                      # printing first Side image
            fig = plt.figure(); ax = fig.add_subplot(); ax.set_title('Side view sample'); plt.imshow(im_pil)
            read_side = True

        if min_w > im_pil.size[1]: min_w = im_pil.size[1]
        if min_h > im_pil.size[0]: min_h = im_pil.size[0]
```

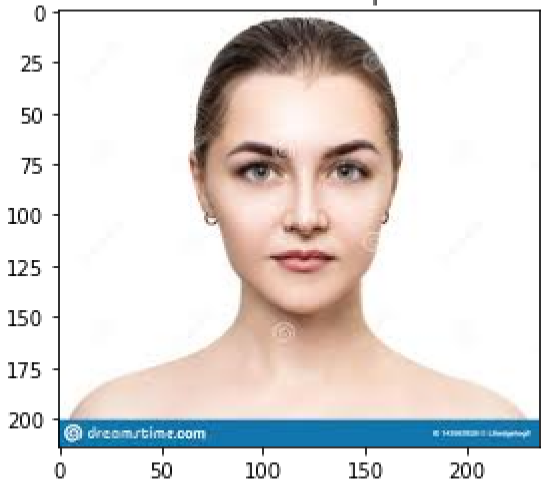
```
for file in os.listdir(my_path + '\Front'):
    im_pil = Image.open(join(my_path + '\Front', file))         # reading images using PIL

    if read_front == False:                                     # printing first Front image
        fig = plt.figure(); ax = fig.add_subplot(); ax.set_title('Front view sample'); plt.imshow(im_pil)
        read_front = True

    if min_w > im_pil.size[1]: min_w = im_pil.size[1]           # Finding the minimum image size
    if min_h > im_pil.size[0]: min_h = im_pil.size[0]

# min_w //= 4; min_h //= 4

# Resizing the images using opencv and creating image arrays
read_side = False; gray_side = False
for file in os.listdir(my_path + '\Side'):
    im = cv2.imread(join(my_path + '\Side', file))   # Reading the images using opencv
    new_im = cv2.resize(im, (min_w, min_h))          # Resizing the images (to a fixed size)
    rgb_im = cv2.cvtColor(new_im, cv2.COLOR_BGR2RGB)
    im_pil = Image.fromarray(rgb_im)                 # Converting "opencv" format to "pil" format
    if read_side == False:                           # printing the first resized side image
        display_img(im_pil, rgb_im); read_side = True

    gray_im = rgb2gray(rgb_im)                       # Converting to gray image (depth = 1)
    if gray_side == False:
        display_gray(gray_im, 'Gray side view sample'); gray_side = True

    img_array = img_to_array(gray_im)                # convert to numpy array
    im_list.append(img_array.tolist())               # Adding img to the list

    y.append(0)                                      # 0 = Side

read_front = False; gray_front = False
for file in os.listdir(my_path + '\Front'):
    im = cv2.imread(join(my_path + '\Front', file))  # Reading the images using opencv
    new_im = cv2.resize(im, (min_w, min_h))          # Resizing the images (to a fixed size)
    rgb_im = cv2.cvtColor(new_im, cv2.COLOR_BGR2RGB) # Converting BGR to RGB
    im_pil = Image.fromarray(rgb_im)                 # Converting "opencv" format to "pil" format
    if read_front == False:                          # printing the first resized front image
        display_img(im_pil, rgb_im, side = False); read_front = True

    gray_im = rgb2gray(rgb_im)                       # Converting to gray image (depth = 1)
    if gray_front == False:
        display_gray(gray_im, 'Gray front view sample'); gray_front = True

    img_array = img_to_array(gray_im)                # convert to numpy array
    im_list.append(img_array.tolist())               # Adding img to the list

    y.append(1)                                      # 1 = Front

X = np.array(im_list); y = np.array(y)               # Converting the image list to image array
return X, y
```

In [255]:

```
X, y = read_files(my_path)
```

Side view sample

Front view sample



Resized side view sample

Gray side view sample

Resized front view sample

Gray front view sample

In [169]:
```
def split_data(X, y, test_size = 0.2):
    # Split the data in train and test data
    np.random.seed(0) # the "train_test_split" function uses "np.random.seed" to split the data
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = test_size)
    print(X_train.shape)
    print(X_test.shape)
    return X_train, X_test, y_train, y_test
```

In [170]:

```
X_train, X_test, y_train, y_test = split_data(X, y, 0.2)
```

```
(160, 162, 168, 1)
(40, 162, 168, 1)
```

Reshaping data to make it ready for CNN (Pre-processing)

In [171]:

```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], X_train.shape[3])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], X_test.shape[3])
print(X_train.shape)
print(X_test.shape)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255.0
X_test /= 255.0

y_train = to_categorical(y_train, 2)
y_test = to_categorical(y_test, 2)
print(y_train.shape)
print(y_test.shape)
```

```
(160, 162, 168, 1)
(40, 162, 168, 1)
(160, 2)
(40, 2)
```

Create and compile the model

In [248]:
```
cnn = Sequential()
cnn.add(Conv2D(32, kernel_size=(5, 5), input_shape=(162, 168, 1), padding='same', activation='relu'))
cnn.add(MaxPooling2D())
cnn.add(Conv2D(64, kernel_size=(5, 5), padding='same', activation='relu'))
cnn.add(MaxPooling2D())
cnn.add(Flatten())
cnn.add(Dense(4000, activation='relu')) # I limit the number of nodeds, so it would be not a fully connected
 (dense) NN
cnn.add(Dense(2, activation='softmax')) # Two categories (Front vs. Side)

cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
cnn.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 162, 168, 32)      832       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 81, 84, 32)        0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 81, 84, 64)        51264     
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 40, 42, 64)        0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 107520)            0         
_________________________________________________________________
dense_6 (Dense)              (None, 4000)              430084000 
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 8002      
=================================================================
Total params: 430,144,098
Trainable params: 430,144,098
Non-trainable params: 0
_________________________________________________________________
```

Train the model

In [249]:

```
history_cnn = cnn.fit(X_train, y_train, epochs=5, verbose=1, validation_data=(X_train,y_train))
```

```
Train on 160 samples, validate on 160 samples
Epoch 1/5
160/160 [==============================] - 33s 206ms/sample - loss: 11.7790 - accuracy: 0.4938 - val_loss: 0.
6867 - val_accuracy: 0.5813
Epoch 2/5
160/160 [==============================] - 26s 162ms/sample - loss: 0.5537 - accuracy: 0.7625 - val_loss: 1.1
439 - val_accuracy: 0.5375
Epoch 3/5
160/160 [==============================] - 24s 149ms/sample - loss: 0.7897 - accuracy: 0.6875 - val_loss: 0.3
012 - val_accuracy: 0.9250
Epoch 4/5
160/160 [==============================] - 26s 161ms/sample - loss: 0.2949 - accuracy: 0.9312 - val_loss: 0.2
055 - val_accuracy: 0.9812
Epoch 5/5
160/160 [==============================] - 24s 151ms/sample - loss: 0.1588 - accuracy: 0.9812 - val_loss: 0.1
037 - val_accuracy: 0.9750
```

Evaluation the model

In [250]:

```
score = cnn.evaluate(X_test, y_test)
```

```
40/1 [=======================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=============================================================================================================
=======] - 1s 24ms/sample - loss: 0.1791 - accuracy: 0.9000
```

In [251]:

```
score
```

Out[251]:

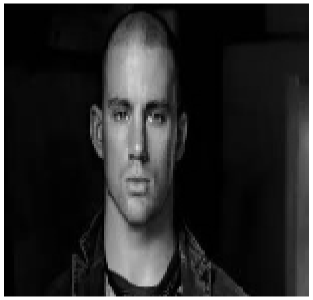
```
[0.2557919770479202, 0.9]
```

Misclassified samples

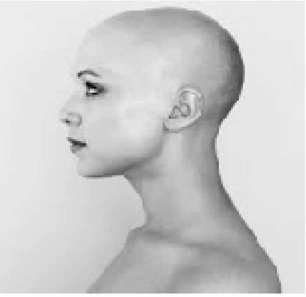
In [252]:

```
pred = cnn.predict(X_test)
mis_side = 0; mis_front = 0
for i in range(len(pred)):
    pred_side = pred[i][0]; pred_front = pred[i][1]; real_side = y_test[i][0]; real_front = y_test[i][1]
    if (pred_side > pred_front) and real_side < real_front and mis_front < 3:
        my_col = []
        for j in range(len(X_test[i])):
            my_row = []
            for k in range(len(X_test[i][j])):
                my_row.append(X_test[i][j][k])
            my_col.append(my_row); del my_row
        my_array = np.asarray(my_col)
        my_array.shape = (len(X_test[i]), len(X_test[i][0]))
        display_gray(my_array, 'Misclassified front view sample', mis = True); mis_front += 1
    if (pred_side < pred_front) and real_side > real_front and mis_side < 3:
        my_col = []
        for j in range(len(X_test[i])):
            my_row = []
            for k in range(len(X_test[i][j])):
                my_row.append(X_test[i][j][k])
            my_col.append(my_row); del my_row
        my_array = np.asarray(my_col)
        my_array.shape = (len(X_test[i]), len(X_test[i][0]))
        display_gray(my_array, 'Misclassified side view sample', mis = True); mis_side += 1
    if mis_side + mis_front >= 6: break
```

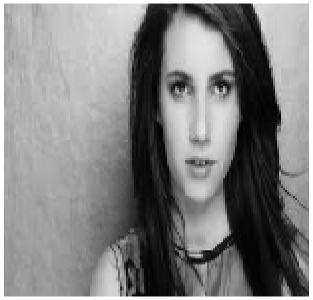
Misclassified front view sample



Misclassified side view sample



Misclassified front view sample



Misclassified side view sample

```
In [ ]:
```