Real-time Detection of Faces

Wenqi Wang
Haibin Hang

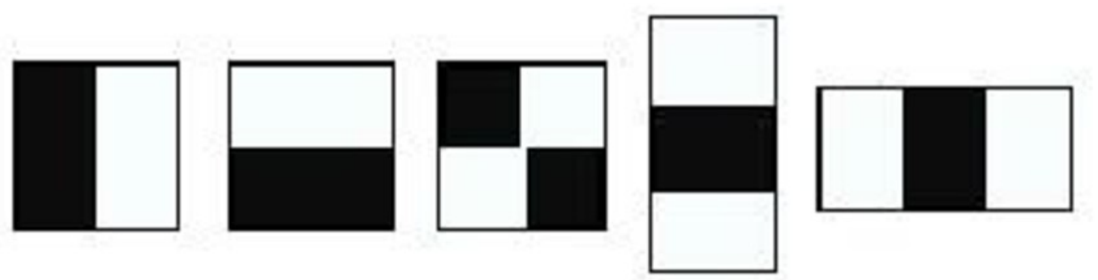
Introduction

The goal of this project is to detect faces from a given set of images. In our project, we implemented 1. Function to calculate Haar features from the training images. 2. Then from the Haar features, we selected best features to be a classifier. 3. We use this classifier to find faces in the given images.

Face Detection

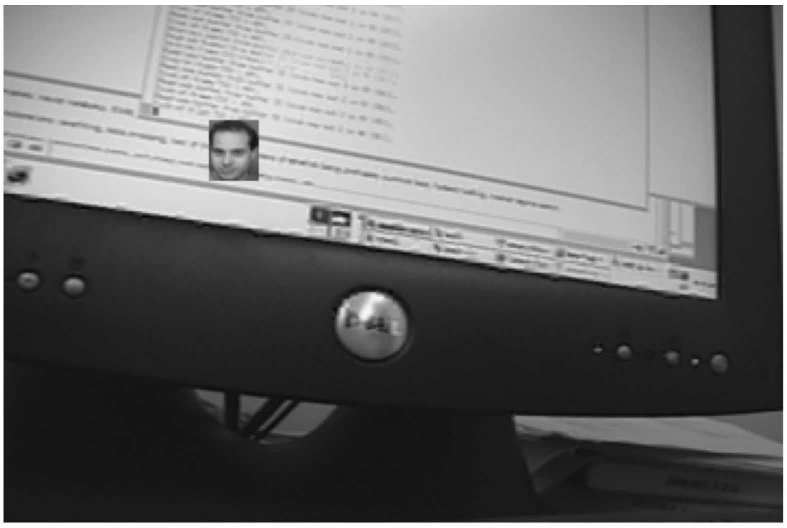
From the given face and background images. We chose 100 face images (s1-s10) and 10 background images (0-9). For each face images (112x98), we first applied Gaussian Pyramid to the images so that we got 28x23 face images. For each background images (480x720), we also applied Gaussian Pyramid make it become (120x180). Then we split each background images with 4 rows and 7 columns. As the results, for each of the background image, we got 28 non-face samples (28x23). Totally we got 100 face images and 280 non-face images.

Now we got our training set of 380 images. The next step is to extract features from it. We used Haar features. There are 5 types of Haar features. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle. Then we got the integral of images and applied all possible sizes and locations of each kernel is used to calculate plenty of features. Finally, we got 202440 features from each 28x24 images.



In the next step, we used Adaboost to select best features among these 202440 features. We applied each feature on all the training images which gave us the Haar feature value. Then we use initial weights to run the Adaboost algorithm to find the best feature with threshold which will classify the faces to positive and negative and then we update the weights for the next run. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. We selected the features with minimum error rate, which means they are the features that best classifies the face and non-face images. In the practice, we choose 10 best features to make up our classifier.

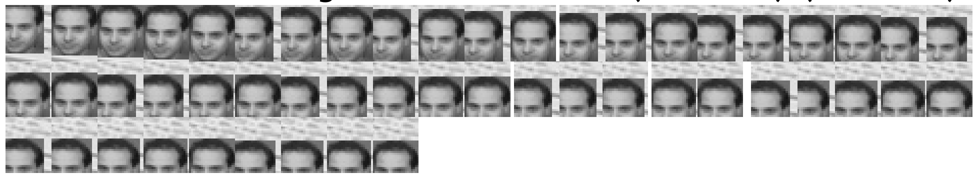
Finally, we used the C program to generate an example image that is combined with one randomly selected background image and one randomly selected face image. Then we applied our classifier to the output images, the section below shows the results. (Since my computer cannot view the .pgm image, when I write this report I use some other tools to change it to .jpg below)



Result:

Time spent in total: 26370s
Time spent on calculating the Haar feature value matrix, which is a 380 (number of
    training samples)*202440 (number of Haar features): 832s
Time spent on Adaboost algorithm to find a classifier consists of 10 good features:
    25026s
Time spent on searching faces on a randomly generated image: 512s

The number of sub-images returned: 1013 (it contains the targeted face image)
The number of faces returned: 51
The number of non-faces returned: 962
The number of sub-images searched: 63474=(240-28+1)*(320-23+1)



Code (We wrote all the related functions using Python in this project):

import numpy as np
import matplotlib.pylab as plt
import numpy as np

```
import cv2
import time

class ComputerVision(object):
    # size is the image shape.
    def img_write(image,name,path_save,size):
        """
        divide the image into small part of image with the given size
        :param image: original image
        :param size: the size of small image that we want to divide from the
original image
        :return: no return
        """

        image = cv2.pyrDown(image)
        image = cv2.pyrDown(image)
        i=0
        for m in range(int(image.shape[0]/size[0])):
            for n in range(int(image.shape[1]/size[1])):
                crop_img = image[m*size[0]:(m+1)*size[0], n*size[1]:
(n+1)*size[1]]
                cv2.imwrite(path_save+name+str(i) + ".jpg", crop_img)
                i+=1

    def integral_image(img):

        """


        :param img:original image
        :return: the integral_image
        """
        height, width = img.shape[0],img.shape[1]
        blank_image1 = np.zeros((height, width), int)
        blank_image2 = np.zeros((height, width), int)

        for m in range(height):
            for n in range(width):
                for z in range(m+1):
                    blank_image1[m][n] += img[z][n]

        for m in range(height):
            for n in range(width):
                for z in range(n+1):
                    blank_image2[m][n] += blank_image1[m][z]

        return blank_image2

    def haar_feature(size):
        """
        Given a size of image, generate all its haar features based on the
```

```
five basic unit features
    :param size: the size to generate haar features
    :return: total haar features
    """
    # image_x and image_y represent the dimension of the image, which
is the first and second index of size.
    image_x=size[0]
    image_y=size[1]
    # use the following five kind of basic feature types
    features_num=5
    features_type=[[2,1],[1,2],[3,1],[1,3],[2,2]]
    # haar_feature is to store all the features of the image
    haar_feature=[]
    count=0

    for i in range(features_num):
      size_x=features_type[i][0]
      size_y=features_type[i][1]

      for width in np.arange(size_x,image_x+1,size_x):
        for height in np.arange(size_y,image_y+1,size_y):
          for x in range(image_x-width+1):
            for y in range(image_y-height+1):
              haar_feature.append([i,x,y,width,height])
              count=count+1
    print("the total number of haar feature is:",count)
    return haar_feature

  def feature_value(integral_image, feature):
    """
    Given integral_image and a specific Haar feature, returns the
difference of the corresponding squares
    indicated by this Haar feature in original image

    """
    # read the the type, location, scale of Haar feature
    t=feature[0]
    x=feature[1]
    y=feature[2]
    w=feature[3]
    h=feature[4]

    if t==0:
      a_11=a_12=a_13=a_21=0
      a_22=integral_image[x-1+w/2][y-1+h]
      a_23=integral_image[x-1+w][y-1+h]
      if (x-1>0):
        a_21=integral_image[x-1][y-1+h]
        if y-1>0:
          a_11=integral_image[x-1][y-1]
```

```
            a_12=integral_image[x-1+w/2][y-1]
            a_13=integral_image[x-1+w][y-1]
    elif y-1>0:
        a_12=integral_image[x-1+w/2][y-1]
        a_13=integral_image[x-1+w][y-1]

    return a_11-2*a_12+a_13-a_21+2*a_22-a_23
elif t==1:
    a_11=a_12=a_21=a_31=0
    a_22=integral_image[x-1+w][y-1+h/2]
    a_32=integral_image[x-1+w][y-1+h]
    if x-1>0:
        a_21=integral_image[x-1][y-1+h/2]
        a_31=integral_image[x-1][y-1+h]
        if y-1>0:
            a_11=integral_image[x-1][y-1]
            a_12=integral_image[x-1+w][y-1]
    elif y-1>0:
        a_12=integral_image[x-1+w][y-1]

    return -a_11+a_12+2*a_21-2*a_22-a_31+a_32
elif t==2:
    a_11=a_12=a_13=a_14=a_21=0
    a_22=integral_image[x-1+w/3][y-1+h]
    a_23=integral_image[x-1+w/3+w/3][y-1+h]
    a_24=integral_image[x-1+w][y-1+h]
    if x-1>0:
        a_21=integral_image[x-1][y-1+h]
        if y-1>0:
            a_11=integral_image[x-1][y-1]
            a_12=integral_image[x-1+w/3][y-1]
            a_13=integral_image[x-1+w/3+w/3][y-1]
            a_14=integral_image[x-1+w][y-1]
    elif y-1>0:
        a_12=integral_image[x-1+w/3][y-1]
        a_13=integral_image[x-1+w/3+w/3][y-1]
        a_14=integral_image[x-1+w][y-1]

    return a_11-2*a_12+2*a_13-a_14-a_21+2*a_22-2*a_23+a_24
elif t==3:
    a_11=a_12=a_21=a_31=a_41=0
    a_22=integral_image[x-1+w][y-1+h/3]
    a_32=integral_image[x-1+w][y-1+h/3+h/3]
    a_42=integral_image[x-1+w][y-1+h]
    if x-1>0:
        a_21=integral_image[x-1][y-1+h/3]
        a_31=integral_image[x-1][y-1+h/3+h/3]
        a_41=integral_image[x-1][y-1+h]
        if y-1>0:
            a_11=integral_image[x-1][y-1]
            a_12=integral_image[x-1+w][y-1]
        elif y-1>0:
            a_12=integral_image[x-1+w][y-1]

        return a_11-a_12-2*a_21+2*a_22+2*a_31-2*a_32-a_41+a_42
    else:
        a_11=a_12=a_13=a_21=a_31=a_41=0
        a_22=integral_image[x-1+w/2][y-1+h/2]
        a_23=integral_image[x-1+w][y-1+h/2]
        a_32=integral_image[x-1+w][y-1+h]
        a_33=integral_image[x-1+w][y-1+h]
        if x-1>0:
            a_21=integral_image[x-1][y-1+h/2]
            a_31=integral_image[x-1][y-1+h]
            if y-1>0:
                a_11=integral_image[x-1][y-1]
                a_12=integral_image[x-1+w/2][y-1]
                a_13=integral_image[x-1+w][y-1]
        elif y-1>0:
            a_12=integral_image[x-1+w/2][y-1]
            a_13=integral_image[x-1+w][y-1]

        return a_11-2*a_12+a_13-2*a_21+4*a_22-2*a_23+a_31-
2*a_32+a_33


def weak_classifier(features,weights,classes):
    """
    this function is to realize the weak classifier in the paper " Robust
    Real-time Object Detection".
    features: input of haar features value array, list or n*1 ndarray
    weights: the weights of input haar features, list or n*1 array
    classes: the classes of each input feature,list or n*1 array
    return: the minimum error of the classifier, and the corresponding
threshold
    """
    features=np.array(features)
    weights=np.array(weights)
    classes=np.array(classes)

    #feature_matrix combines the haar features, weights and classes
together.

feature_matrix=np.concatenate((features.reshape(features.shape[0],1),
weights.reshape(weights.shape[0],1),classes.reshape(classes.shape[0],1)
),1)
    # sorted the feature matrix by the haar features
    feature_matrix=feature_matrix[np.argsort(feature_matrix[:,0])]
    T_p=np.sum(feature_matrix[feature_matrix[:,2]==1][:,1])
    T_n=np.sum(feature_matrix[feature_matrix[:,2]==0][:,1])
```

```
        _error=[]
        for i in range(1,feature_matrix.shape[0]-1):
            S_p=np.sum(feature_matrix[:i,:][feature_matrix[:i,2]==1][:,1])
            S_n=np.sum(feature_matrix[:i,:][feature_matrix[:i,2]==0][:,1])
            _error.append(min(S_p+(T_n-S_n),S_n+(T_p-S_p)))

        return min(_error),feature_matrix[_error.index(min(_error)),0]

    def ada_boost(features,weights,classes,iter_num):
        """
        :param: features:input of haar features array, can be nxm np array
matrix
        :param weights: the weights of input haar features
        :param classes: the classes of each input feature
        :param iter_num: total number of iterations, that is the number of
classifiers
        :return: alpha,index of features and threshold for each classifier
        """

        # normalize the weights
        weights=np.array(weights)/sum(weights)
        features=np.array(features)
        classes=np.array(classes)


feature_matrix=np.concatenate((features,weights.reshape(weights.shap
e[0],1),classes.reshape(classes.shape[0],1)),1)


        # loop to find iter_num classifiers

_alpha_list,_feature_index_list,_error_classifer_list,_threshold_classifie
r_list=[],[],[],[]
        for i in range(0,iter_num):
            print("the model number",i)

            # loop for all the features
            _error_list,_threshold_list,_class_predict_list=[],[],[]
            for j in range(features.shape[1]):
                # sort the feature_matrix by the j feature
                #feature_matrix=feature_matrix[np.argsort(feature_matrix[:,j])]


_threshold=ComputerVision.weak_classifier(feature_matrix[:,j],feature_
matrix[:,-2],feature_matrix[:,-1])[1]
                _threshold_list.append(_threshold)
                _class_predict=[0 if x <=_threshold else 1 for x in
feature_matrix[:,j]]
                _class_predict_list.append(_class_predict)
                _error=np.sum(feature_matrix[:,-2][_class_predict!
=feature_matrix[:,-1]])
                _error_list.append(_error)

            _error_min=min(_error_list)
            _feature_index=_error_list.index(_error_min)
            _threshold_choose=_threshold_list[_feature_index]
            _class_choose=_class_predict_list[_feature_index]

            if _error_min==0:
                return [1],[_feature_index],[_threshold_choose]

            # update the weights
            _beta=_error_min/(1+_error_min)
            _alpha=np.log(1/(_beta))

            for i in range(feature_matrix[:,_feature_index].shape[0]):
                if _class_choose[i]==feature_matrix[i,-1]:
                    feature_matrix[i,-2]=feature_matrix[i,-2]*_beta
            feature_matrix[:,-2]=feature_matrix[:,-2]/sum(feature_matrix[:,-2])

            _feature_index_list.append(_feature_index)
            _error_classifer_list.append(_error_min)
            _threshold_classifier_list.append(_threshold_choose)
            _alpha_list.append(_alpha)

        return _alpha_list,_feature_index_list,_threshold_classifier_list
```