CSC420H1F Project Report

Ethnicity Recognition

Name: JingXi, Hao
Student Number: 1000654188

Introduction

Ethnicity recognition is a process of determining the race group to which the individual of face image belongs. This process involves the topics of linear filters, features extraction, classification model construction, object detection, affine transformation, and object recognition mentioned in the course. Current research pays more attention on developing and improving the algorithms for face detection and face recognition. In comparison, ethnicity recognition receives less attention. However, I think face classification with respect to race are essential and able to provide enhancement in the performance of a wider range of the applications such as identity authentication, access control, and etc..

Based on the early works, several algorithms are developed for ethnicity recognition. In [1], there are two methods illustrated. One is to employ Artificial Neural Network and the other one uses Convolutional Neural Network. For Artificial Neural Network algorithm, facial features are the ratios between facial areas, such as nose, mouth, left eye, and right eye, which are detected by employing cascade classifiers. Besides, there are two more features which are skin tone and normalized forehead area. The skin tone is calculated by YCbCr model and the normalized forehead area are computed with the help of Edge Detection. Then, train a feedforward Artificial Neural Network as the classifier by using backpropagation-based training algorithm based on the features extracted from each image in the training set. This algorithm produces 82.4% accuracy. In addition, a pre-trained CNN model has been used in this paper and it produces 98.6% accuracy. In [2], Web Local Descriptors (WLD) histogram is extracted from normalized face images and then employ Kruskal-Wallis feature selection technique to select the best discriminated bins. City block, Euclidean, and chi-square minimum distance classifiers are used for testing. FERET database is used in the experiment, where there are five major race groups, and this algorithm yields more than 90% accuracy for each race group.

My partner and I employ different methods to train the ethnicity recognition model, where I adopt SVM algorithm and my partner uses CNN algorithm. In this paper, I illustrate my method to detect five ethic groups that are White, Black, Asian, Indian, and others. The facial features, which is a 128-d embedding vector, for each face image in the training set are extracted by using a pre-trained neural network model. Then, I use the embeddings to train a Support Vector Machine (SVM) as the race recognition model. For each input image with one or more people presented, the program would draw a bounding box for each face detected and then find the facial landmarks for each face based on which the program would perform a face alignment with the help of the affine transformation. Then, recognize the race of each individual by using the model trained. Based on the algorithm mentioned above, I expect that the program would fail to determine race for side faces and would misclassify between Asian and Indian.

Methods

1) **Dataset**

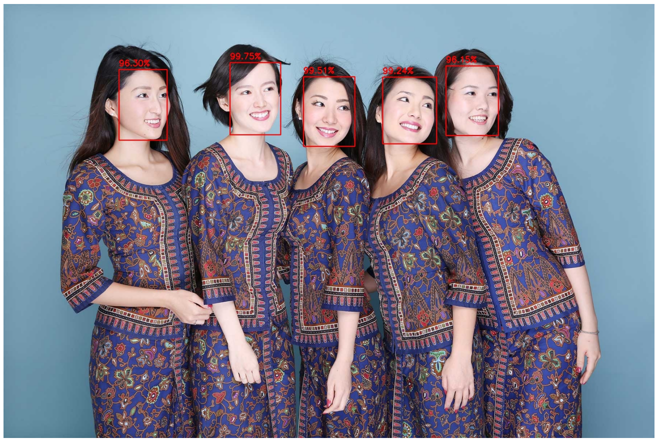
We obtain the dataset from UTKFace [3]. This dataset contains more than 20,000 face images, only one face in each image, with annotations of gender, age, and ethnicity. It covers large variation in pose, facial expression, illusion, resolution, and etc.. Also, the face images provided in this dataset are correspondingly aligned and cropped faces and labelled by age (range from 0 to 116), gender (0 or 1, where 0 indicates male and 1 indicates female), and race (range from **0 to 4, denoting White, Black, Asian, Indian, and Others**). Our goal in this step is to divide the images in the dataset into training, validation, and test set. Therefore, we first divide the dataset by five races. Under each race, divide the images into male and female categories, which is for the purpose of having the approximately same proportion of male and female when we construct the training, validation, and test set. Then, the training set consists of 60% images in each gender and race, and for validation and test set, each consists of 20% images in each gender and race. Please see the detailed code implementation of functions, **process_data()** and **split_data()**, in **data.py**. For this task, my partner and I worked together and contributed equally.

2) **Classification Model Construction**

I first build the training and test set embedding matrices along with its corresponding label vectors. When read in every face image from each set, preprocess the image by using Gaussian blur to reduce the noise. Then, we employ a pre-trained embedding extraction model to compute a 128-d feature vector for each face image passed into the model [4]. This pre-trained model is from OpenFace project and it is called FaceNet [5]. In order to train FaceNet model, the triplet loss function is used and for each batch of the input data, three images are included, which are anchor, positive image, and negative image. The anchor and the positive image are from the same face which is not contained by the negative image. When the neural network computes the embedding for each face, the weights of the network are tweaked by triplet loss function. This process would push the negative image embeddings further away and put the anchor and the positive image embeddings closer, which is able to produce a suitable embedding vector to quantify each face image. In order to use this model, load it from openface_nn4.samll2.v1.t7 file. Then, pass each preprocessed image into the model to find its embedding vector. Then, combine all embedding vectors of the images in each set and build the embedding matrix for that set. Then, I use training set embedding matrix and its labels to train a Support Vector Machine (SVM) as the recognition model. After I attain the recognition model, use the test set to test the accuracy of the model. Please see the detailed code implementation of the function **extract_embeddings_and_labels()**, **train_race_recognition_model()**, and **test_accuracy()** in **model.py**. I wrote this part of the code and adopted the idea from [4].

3) **Face Detection**

In this step, we employ an OpenCV pre-trained deep learning face detector model [6] to complete the face detection task. The pre-trained deep learning face detector is called Caffe-based face detector which is under the OpenCV dnn module. First, we load and store the face detection model by reading in its architecture (from .prorotxt file) and its weights (from .caffemodel file). Then, we read in and store our query image and construct a blob which is the output after resizing (300 by 300) and normalizing the query image. Then, we complete the pre-process step before we pass the query image into the face detection model. Then, we pass the blob created into the face detection model and set a threshold, 0.5, to get rid of the weak detections. Thus, for each detection passed the threshold, we draw a bounding box around each face and label the confidence value of being a face on the top of the box. We show an example below and please see the detailed code implementation of function **detect_faces()** in **face.py**. For this step, my partner and I worked together and contributed equally.

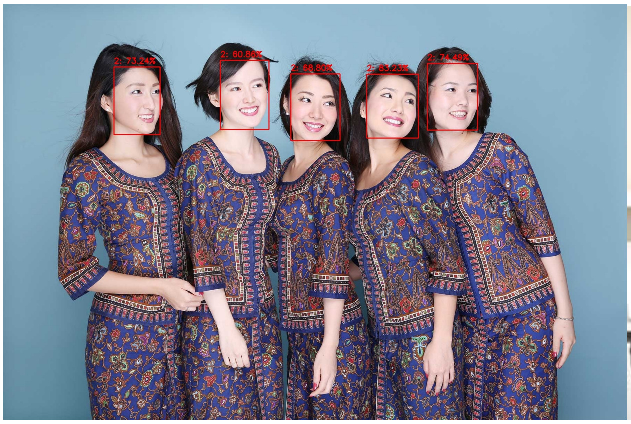


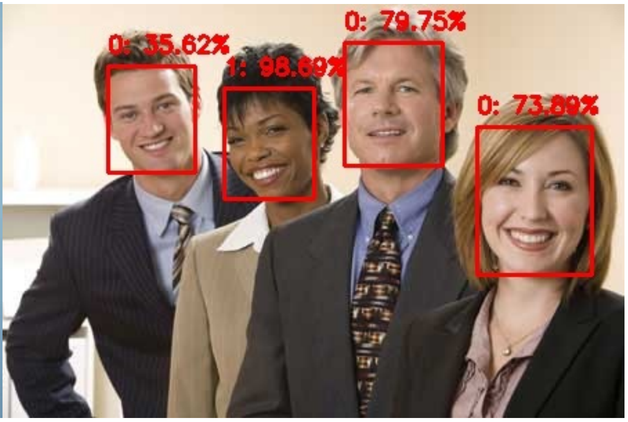
4) **Face Alignment**

In order to do the face alignment, we first need to find facial landmarks which needs the **dlib** library [7]. This library offers a facial landmark detector, which is able to estimate 68 facial landmark positions on the face. The model is obtained by training an ensemble of regression trees based on a training set of labeled facial landmarks on each image, where the facial landmarks are manually labeled. Given the probability on distance between pairs of input pixels, the trained model is able to predict the facial landmark positions from the pixel intensities. Thus, we employ the facial landmark model mentioned above to find the coordinates of each facial landmark for every detected face on the query image. First, we need to load the facial landmark detector from shape_predictor_68_face_landmarks.dat file. Then, for each face detected in **step 3),** we pass the gray scaled query image and the top, left, bottom, and right position of the face into the facial landmark detector. Thus, we obtain a list of 68 coordinates of facial landmarks. Then, we start to align each detected face by following the idea in [7]. Based on the landmarks found by the facial landmark detector, we first extract the left and right eye position and then compute the central of mass of each eye. Then, take the x-difference and y-difference between eye centroids and compute the arc tangent between the x-difference and y-difference to find the the rotation angle between eyes. Since we have desired left eye x-coordinate defined, then we compute the desired right eye x-coordinate based on it. Thus, we are able to find the scale of the face by calculating the ratio of the distance between the eye enters of the current image and the distance between the desired eye positions. Then, we compute the centre coordinates between two eyes in the face image. By passing eye centre coordinates, rotation angle, and scale of face into OpenCV function, we are able to find the rotation matrix, here is our affine matrix. Then, employ affine transformation, we obtain the aligned faces. We show examples below (for each pair of images, **left image: aligned, right image: original**) and please see the detailed code implementation of the function **find_landmarks()** and **align_face()** in **face.py**. For this task, my partner and I both worked on it and my partner made the final modification.

5) **Face Recognition**

In this step, I only need to use the model trained in **step 2)** to predict each aligned faces obtained from **step 4)**. Then, for each face in the query image, draw a bounding around it and label with the ethnicity (**0: White, 1: Black, 2: Asian, 3: Indian, 4: Others**) and its confidence value on the top of the box. Show some examples below. Please see the detailed code implementation of function **recognize_race()** in **model.py**. I wrote this part of the code.

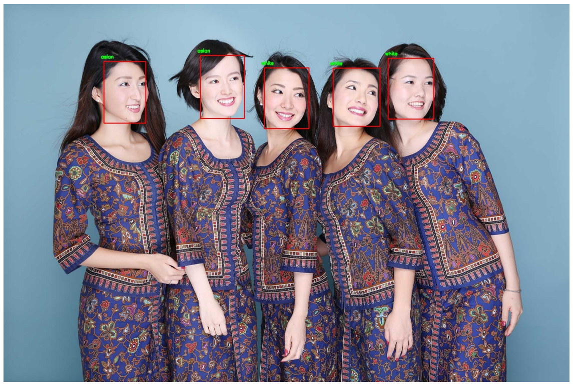


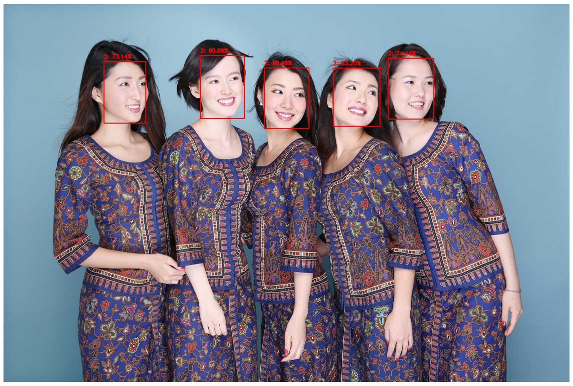


Results and Discussions

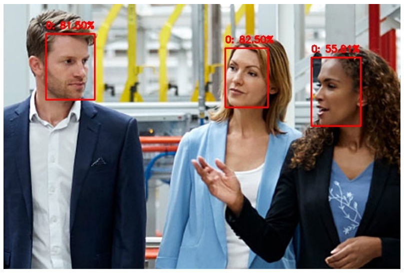
By testing the accuracy on the test set, my program produces 75% accuracy. I have done several steps in order to achieve higher accuracy. First of all, every image in the training set is blurred by using OpenCV GaussianBlur function before passing it into the feature extraction model to compute its embedding vector. This process improves the model prediction accuracy a little bit (not much, only 0.2% ~ 0.3%). Also, since I prefer to have relatively even number of images for each race in the training set, so we first run with the least number of images for all the race, which is approximately 1000 images for each race. Then, I found that the recognition model trained by using 5000 images produces 71% accuracy, which is lower than the accuracy, 75%, of the model trained by using more than 14, 000 images. Therefore, I decide to use the large dataset to produce more accuracy.

Furthermore, in comparison with my partner's CNN model, which achieves approximately 75% accuracy as well, sometimes my SVM model produces more accurate prediction and sometimes her CNN model generates more accurate predictions. Here are the examples displayed below. In this case, the left one is my partner's output and the right one is my output. As you can see, my algorithm woks better since the right threes ladies on the left image are labeled as White. There are cases that her model works better than mine.

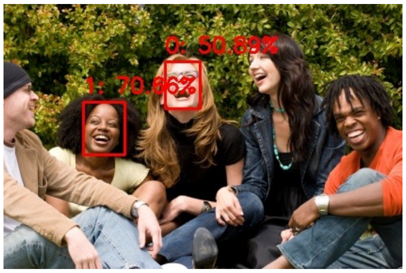




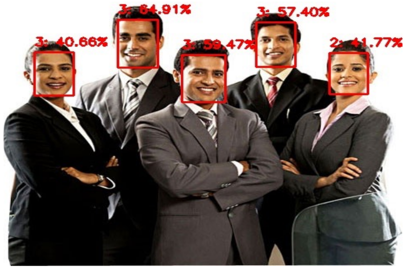
There are some issues with my program that would cause failure of the model. Firstly, my algorithm for face alignment only works fine with frontal faces in the query image. If side faces appeared on the query image, the aligned face would still be side face and this would cause the predication results to be wrong and provide failure cases for race recognition. The **example (1)** is shown below. Lady at right is misclassified as White (labeled as 0). The other issue related to face detection since some faces (i.e. side, covered, far away from the camera, and etc.) would not be detected or dropped as weak detections. Then, our program would not be able to produce a race recognition on it. Show **example (2)** below, where only two faces are detected and recognized with respect to ethnicity. Moreover, my program would misclassify between Asian and Indian. The **example (3)** is displayed below. As you can see, the lady on the right side is classified as Asian (labeled with 2).



Example (1)



Example (2)



Example (3)

In addition, there are dataset issues that would fail the program. Firstly, I do not have a very large dataset. Thus, my program may be less robust to variations and cause the failure in race recognition. Secondly, we have more images for White, therefore, more face would be misclassified as White. Thirdly, there are many baby faces inside the dataset. Since baby faces have less racial characteristics. Thus, by randomly pick images to construct training set for each race from the database, if more baby face images are selected, then the features extracted for that race would not be accurate and this would also cause the failure in recognition.

Main Challenges

1. Extract Face Features, initially using SIFT + SVM

At first, I tried to use SIFT to extract features for each image in the training set and employ KMEANS to cluster the features and find the visual words. Then, use the bag-of-words vector along with TF-IDF as the image embedding to train a Support Vector Machine (SVM) as the race recognition model. However, this method provides low accuracy. Therefore, I decide to utilize the FaceNet deep learning model as our recognition model at the end.

2. Face Alignment

Our face alignment algorithm works pretty well on frontal faces. However, it can not solve the side face issue. When have side faces on the query image, the face would stay as side face after alignment and our algorithm can not predict the face structure in order to covert the face into its frontal face. Also, there are no side faces in the training set, thus it is hard to determine the race for side faces.

3. Train Ethnicity Recognition Model, initially using Convolutional Neural Network (CNN)

I spent too long time to train a CNN model with parameter tuning. Thus, I choose to use Support Vector Machine (SVM) which produces a quick way to train a recognition model and offers fairly good results in comparison with my partner's CNN model.

Reference

1. Sarfaraz Masood1, Shubham Gupta2,Abdul Wajid3, Suhani Gupta4, "Prediction of human ethnicity from facial images using neural networks "
2. Ghulam Muhammad, Muhammad Hussain, Fatimah Alenezy, George Bebis, Anwar M. Mirza, Hatim Aboalsamh, "Race recognition from face images using Weber local descriptor", 2012 19th International Conference on Systems, Signals and Image Processing (IWSSIP)
3. UTKFace, https://susanqq.github.io/UTKFace/
4. OpenCV Face Recognition, https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/
5. Florian Schroff, Dmitry Kalenichenko, James Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering", 2015 CVPR publication
6. Face detection with OpenCV and deep learning, https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning
7. Facial landmarks with dlib, OpenCV, and Python, https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/
8. Face alignment with OpenCV and Python, https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/