Face Recognition
using Local Binary Pattern features

Final Project

- 패턴인식 시스템 -

2018.05.17

제출 관련 공지

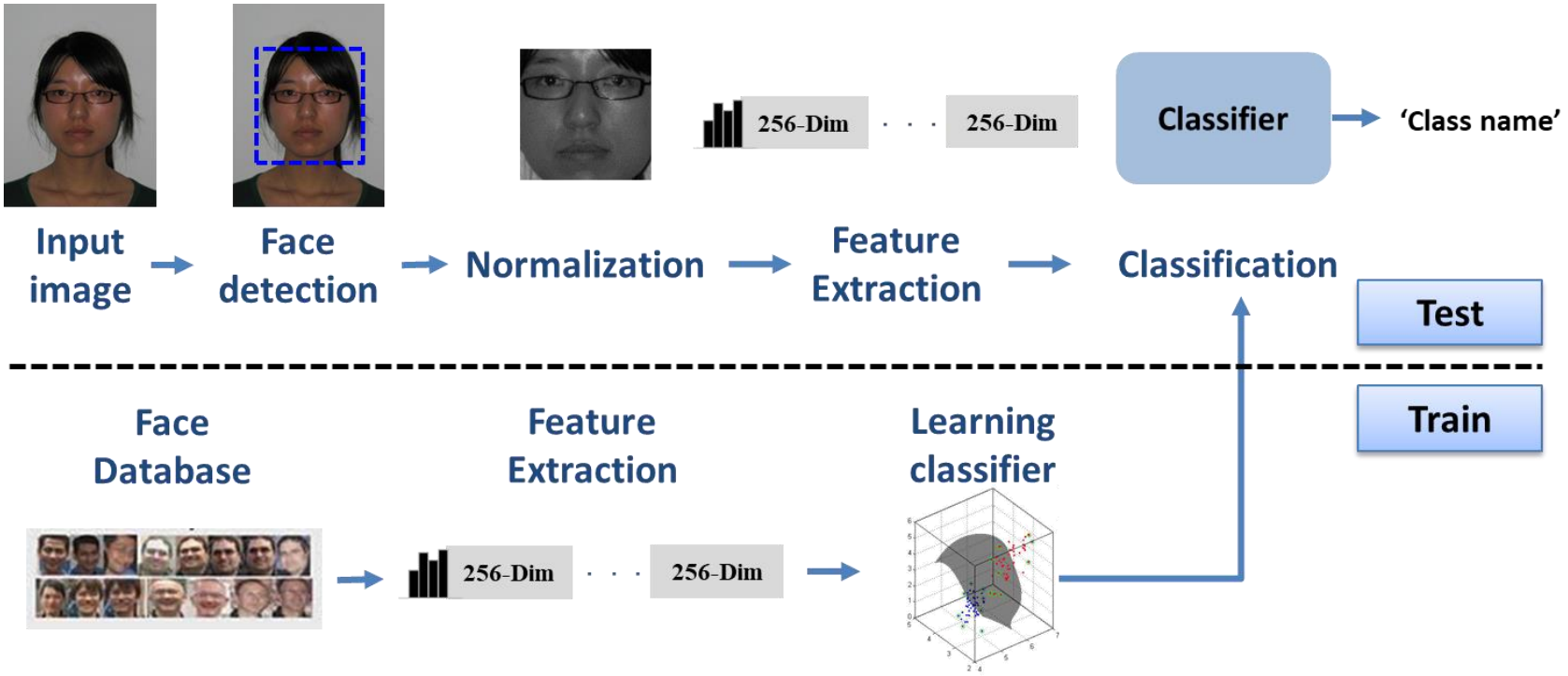
- **제출 마감일: 6월 10일 24:00 까지**

- **제출 목록**
  - 소스 코드 압축 파일
  - 결과 분석 보고서

- **결과 분석 보고서 관련 추가 설명**
  - 필수 사항
    - ✓구현한 얼굴 인식 과정 설명 (특징 추출, 정규화 방법 등)
    - ✓주어진 데이터 셋을 이용하여 결과 분석 및 성능 평가 결과
  - 선택 사항
    - ✓인식 성능 향상을 위한 별도의 과정을 추가할 경우 작성
    - ✓다른 공개 데이터를 통해 추가 실험한 경우 작성

Goal

- **Input: 얼굴 이미지 (Single face image)**
- **Output: 입력 이미지의 인식 클래스 결과 출력 (Class label)**



<Overview>

To-do list

- **얼굴 검출: OpenCV 제공 검출기 또는 Lib face 검출기 사용**
  - 제공된 5가지 종류의 얼굴 검출기 학습 파일 중 택 1 후 로드하여 사용
- **얼굴 크기 정규화**
- **얼굴 특징 추출: Local Binary Pattern 특징 사용**
  - LBP 특징 추출 및 히스토그램 생성

    (함수 사용하지 않고 논문 참고하여 직접 구현)
- **분류기 학습 및 테스트 (2가지 모두 구현)**
  - Support Vector Machine 분류기 (LibSVM 라이브러리 사용)
  - Nearest Neighbor 분류기

Database

- **제공 데이터 셋: Train / Test 폴더**
  - 인원: 총 7명
  - 해상도: 768 X 576

- **Train 폴더: 분류기 학습에 사용**
  - 1인 당 20장 포함

- **Test 폴더: 분류기 테스트에 사용**
  - 1인 당 약 15장 포함

1) Face Detection: Python

- **얼굴 검출 단계**
  - 조건: OpenCV 라이브러리에서 제공하는 얼굴 검출기 사용
- **Viola-Jones 검출 알고리즘**
  - Haar-like 특징: 영상에서 영역과 영역의 밝기 차를 이용
  - detectMultiScale 함수의 각 파라미터 별 의미 파악 후, 적절히 조절

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

얼굴 검출기 학습 파일 제공

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

<Example>

1) Face Detection: C++

- **얼굴 검출 단계**
  - 조건: OpenCV 라이브러리에서 제공하는 얼굴 검출기 사용
- **Viola-Jones 검출 알고리즘**
  - Haar-like 특징: 영상에서 영역과 영역의 밝기 차를 이용
  - detectMultiScale 함수의 각 파라미터 별 의미 파악 후, 적절히 조절

```
Mat input = imread("./face.jpg");
Mat gray;
cvtColor(input, gray, CV_RGB2GRAY);

CascadeClassifier face_classifier;
face_classifier.load("./haarcascade_frontalface_default.xml");
vector<Rect> faces;
face_classifier.detectMultiScale(img, faces,1.2, 10, 0, cvSize(80, 80), cvSize(200, 200));
```

얼굴 검출기
학습 파일 제공

<Example>

1) Face Detection

- **OpenCV 라이브러리에서 제공하는 기본 검출기는 얼굴 회전에 취약함. 기본 검출기를 이용해서 얼굴 검출이 잘 되지 않을 경우**

- **하단 사이트를 참고하여 별도 검출기 사용 가능**
  - DLL/lib/header 파일 다운로드 후 로드하여 사용
  - 코드 적용 방법은 사이트 내 libfacedetect-example.cpp 파일 참조

  - https://github.com/ShiqiYu/libfacedetection

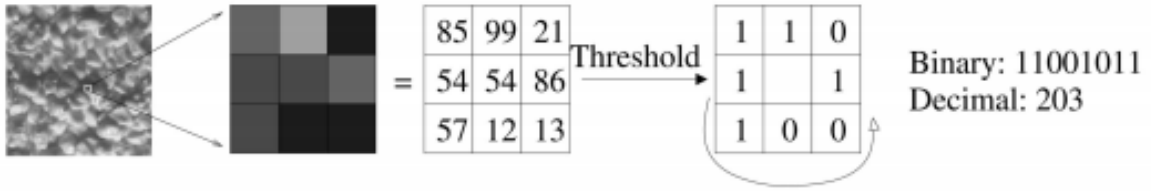
2) Face Normalization

- **검출된 얼굴 크기가 모두 다르므로, 정규화 과정 필요**

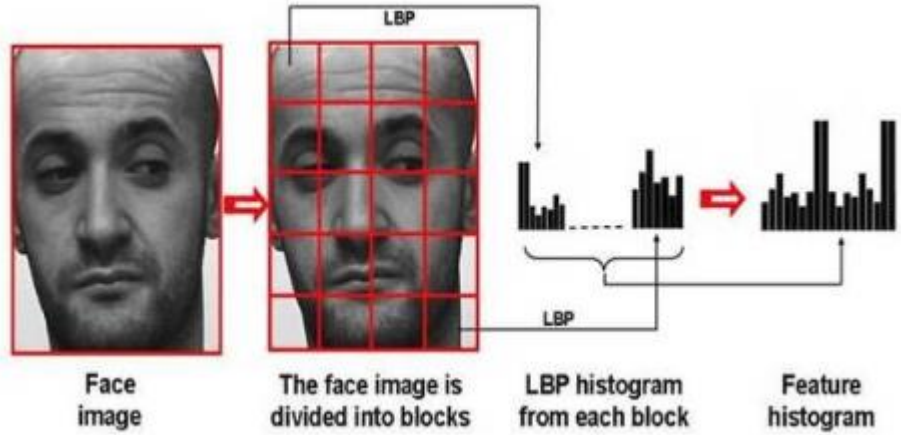
- **테스트 이미지들의 검출된 얼굴이 고정된 얼굴 크기가 될 수 있도록 적절한 과정을 통해 얼굴 크기 정규화**
  - 얼굴 크기를 정규화 하지 않을 경우, 9p 특징 추출 과정에서 각 히스토그램들을 정규화(Normalization)하는 과정 필요

3) Feature Extraction

- **얼굴 특징 추출 단계**
  - 정규화된 얼굴 영상을 일정한 크기의 셀로 분할
  - 각 셀 마다 Local Binary Pattern에 대한 히스토그램 구함
    (해당 셀에 속하는 LBP 인덱스 값들에 대한 히스토그램을 의미)
  - 구한 히스토그램들을 일렬로 연결한 벡터를 최종 특징으로 사용



Local Binary Pattern example



Feature Extraction example

4) Classifier

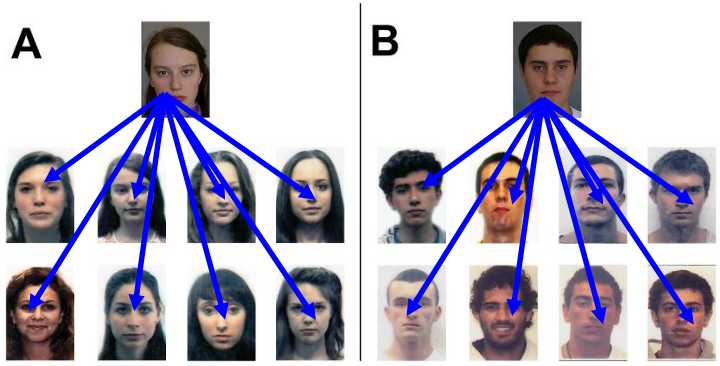
- **추출한 LBP 특징을 이용하여 2가지 분류기로 최종 인식**

Support Vector Machine(SVM) 분류기

- LibSVM 라이브러리 사용하여 클래스 별 확률값 출력 후 확률이 높은 클래스로 최종 인식

- **Nearest Neighbor(NN) 분류기**

- 학습 과정 없이 테스트 이미지 1장에 대해 전체 학습 폴더 내의 모든 이미지와 각각 매칭(거리 계산)하여 가장 가까운 클래스로 최종 인식



4) Classifier

- **SVM 분류기 학습 방법 (1회 학습)**
  - Train 폴더 내 7명에 대해 1인 당 20장 씩 학습에 이용
  - 성능 평가 시 Train 폴더 내 아이디 번호를 정답 Class label(=Ground truth)로 사용 (ID_1, ID_2..)
  - 학습 후, SVM 학습 파일 생성됨(.xml)

- **SVM 분류기 테스트 방법**
  - Test 폴더 내 이미지들을 이용하여 테스트
  - 테스트 시, 학습한 SVM 파일 로드하여 인식 수행
  - 학습에 사용한 클래스 별로 확률값 및 최종 인식 결과 출력 (Class label 출력)

성능 평가 방법

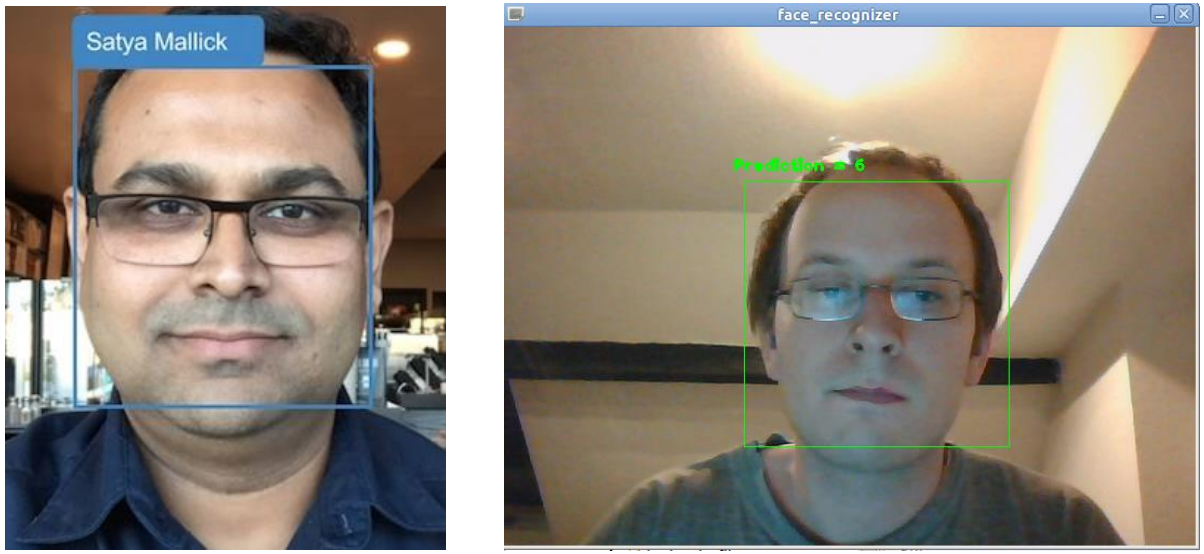
- **전체 테스트 셋에 대하여 인식률 측정**

- **인식률(Recognition rate) 측정 방법**

$$\frac{\text{해당 클래스로 옳게 인식한 이미지 수}}{\text{전체 테스트 이미지 수}} \times 100$$

5) Final Results

- 얼굴 위치 bounding box 출력 (검출 결과)
- 인식된 클래스 레이블 출력 (인식 결과)



<Example>