**KERAS를 활용하여, 얼굴의 특징 점 추출**

|  |  |
| --- | --- |
| 과목 : | 오픈소스SW프로젝트 |
| 분반 : | 01 |
| 교수 : | 나인섭 교수님 |
| 학과 : | 컴퓨터공학과 |
| 학번 : | 20165062 |
| 이름 : | 이윤혁 |

**개발 환경 : Windows 10 Home**

**개발 언어 : Jupyter Notebook(5.7.4)-Python(3.6.8)**

**오픈 소스 : OpenCV(3.4.5-BSD), TensorFlow(1.12.0-Apache2.0), Keras(2.2.4-MIT)**

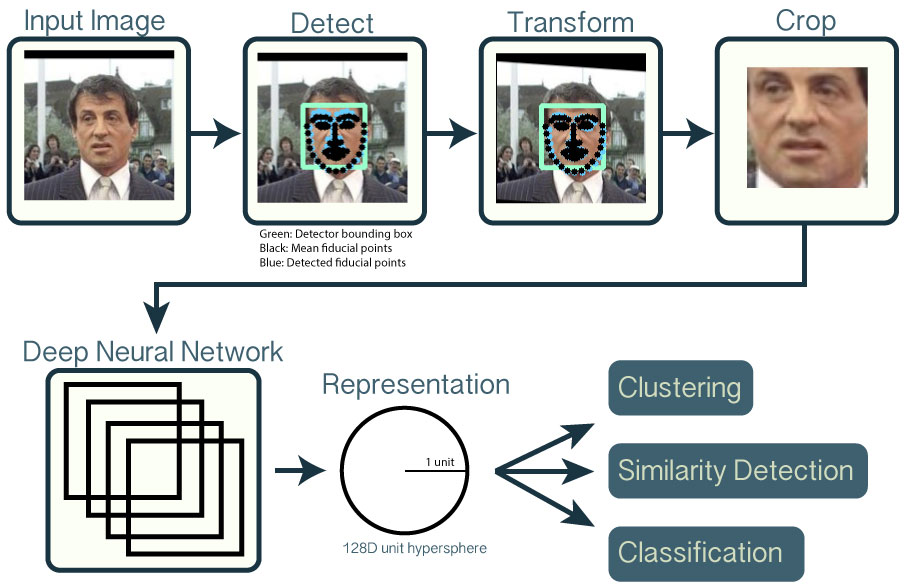
***아파치 2.0 오픈소스 라이센스****는 누구나 해당 소프트웨어에서 파생된 프로그램을 제작할 수 있으며 저작권을 양도, 전송할 수 있는 라이센스 규정을 의미한다. 아파치 라이센스에 따르면 누구든 자유롭게 아파치 소프트웨어를 다운 받아 부분 혹은 전체를 개인적 혹은 상업적 목적으로 이용할 수 있으며 재 배포시에는 원본 소스 코드 또는 수정한 소스 코드를 반드시 포함시켜야 하는 것은 아니고 아파치 라이센스 버전 2.0을 포함시켜야 하며 아파치 소프트웨어 재단에 개발된 소프트웨어라는 것을 명확하게 밝혀야 한다.*

***BSD(Berkeley Software Distribution) 라이센스****는 소프트웨어 라이선스 라고도 할 수 없을 만큼 미약하여, 해당 소프트웨어는 아무나 개작할 수 있고, 수정한 것을 제한 없이 배포할 수 있습니다. 수정본의 재배포는 의무적인 사항이 아니므로 BSD 라이센스를 갖는 프로그램은 공개하지 않아도 되는 상용 소프트웨어에서도 사용할 수 있습니다.*

***MIT 라이센스****는 소프트웨어를 누구라도 무상으로 제한없이 취급가능 하다. 저자 또는 저작권자는 소프트웨어에 관해서 아무런 책임을 지지 않는다. 수정, 배포가 자유로우며 상업적으로 이용해도 되나, 소스로 인한 피해는 사용자가 감수해야 하며, 저작권 표시 및 허가 표시를 소프트웨어의 모든 복제물 또는 중요한 부분에 기재해야 한다.*

**OpenCV(Open Source Computer Vision Library - 3.4.5)**

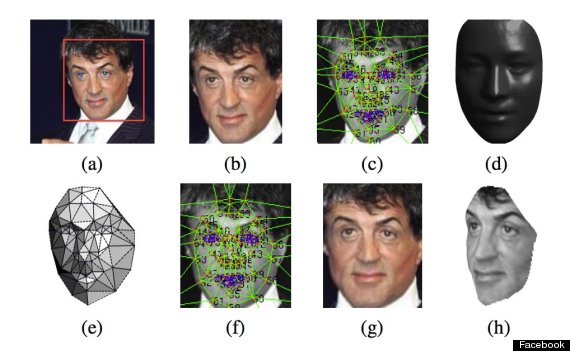
* 수백 개의 컴퓨터 비전 알고리즘(이미지 프로세싱, 비디오 분석, 카메라 캘리브레이션 및 3D 재구성, 2D 기능 프레임 워크, 객체 감지, 고수준 GUI, 비디오 I/O 등…)을 포함하는 오픈 소스 BSD 라이센스 라이브러리이다.
* **OpenFace(0.2.1 - Apache 2.0)**



**Detect (인식단계) :** 입력 이미지에서 얼굴을 찾는 단계이다. OpenCV를 통해 학습된 모델을 이용하여 얼굴을 인식한다. 이 때, 사용하는 알고리즘은 HOG(Histogram of Oriented) 알고리즘을 사용한다**.**

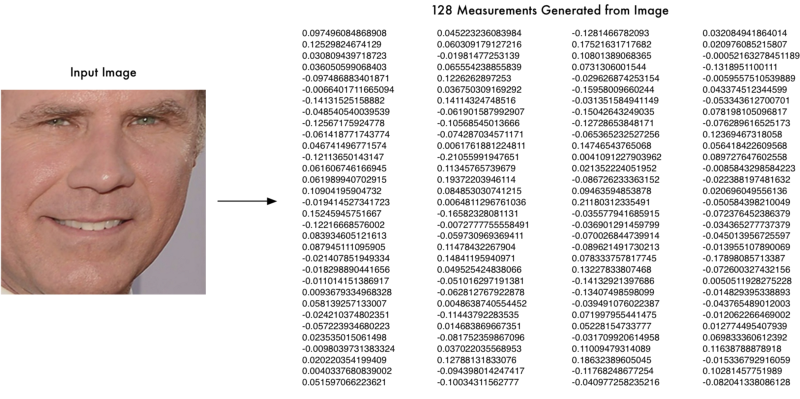
**Transform(변환) & Crop(자르기) :** 입력 이미지에서 인식된 얼굴 이미지를 이미지 분류기 학습에 사용할 수 있도록 표준화 하는 단계다.

Ex) 대표적인 표준화 예시. 페이스북의 DeepFace



동일한 사람이더라도, 정면과 측면에서 보는 모습은 다르다. 컴퓨터가 동일한 사람으로 분류하도록 학습을 하기 위해서는 눈, 코, 입이 사진의 동일한 위치로 오도록 변환해야 한다. 얼굴 특징 점(눈, 코, 입)을 찾는 알고리즘을 얼굴 특징 점 추정(Face Landmark Estimation) 이다. 올바른 배치로 변환하는 알고리즘은 OpenCV의 아핀 변환 알고리즘(2D 변환)을 이용한다.

**Representation(수치화) :** 변환된 얼굴 사진을 학습된 DNN(Deep Neural Network) 모델을 기반으로 수치화하는 단계다.



* DNN 모델 : 입력 이미지를 128차원 숫자 값으로 변환할 수 있는 모델이다.
* 분류 모델 : 128차원 숫자 값을 입력으로 사용하여, 얼굴을 분류할 수 있는 모델이다. 일반적인 분류 학습 알고리즘을 사용할 수 있다.

**TensorFlow(1.12.0)**

* 데이터 흐름 그래프를 사용하는 수치 연산용 오픈소스 소프트웨어 라이브러리입니다.
* 수학 계산과 데이터의 흐름을 노드(Node)와 엣지(Edge)를 사용한 방향 그래프(Directed Graph)로 표현합니다.
* 노드는 수학적 계산, 데이터 입/출력, 그리고 데이터의 읽기/저장 등의 작업을 수행한다.
* 엣지는 노드들 간 데이터의 입출력 관계를 나타낸다.
* **특징**
* 데이터 플로우 그래프를 통한 풍부한 표현력
* 코드 수정 없이 CPU/GPU 모드로 동작
* 아이디어 테스트에서 서비스 단계까지 이용 가능
* 계산 구조와 목표 함수만 정의하면 자동으로 미분 계산을 처리

**Keras(2.2.4)**

* 파이썬으로 작성된 오픈 소스 신경망 라이브러리이다.
* TensorFlow, Microsoft Cognitive Toolkit, Theano 위에서 수행할 수 있다.
* 딥 신경망과의 빠른 실험을 가능하게 설계되었으며, 최소한의 모듈 방식의 확장 가능성에 초점을 둔다.
* 쉽고 빠른 프로토 타이핑 가능
* 컨볼루션 네트워크와 반복적인 네트워크의 조합을 지원
* CPU, GPU에서 원활하게 실행
* 파이썬은 2.7~3.6 과의 호환. 주의! 현재 파이썬 버전은 3.7 버전

**Keras - CNN**

* 회선 신경망은 이미지 인식에서 큰 성공을 거두고 있는 모델이다.
* 회선 층(Convolutional Layer) 풀링 층(Pooling Layer), 전체 결합 층(Fully-Connected Layer)로 구성됨.
* 회선 층은 입력 이미지에 대해 필터를 맞추고 밀어냄.
* 풀링 층은 이미지의 크기를 줄임으로서, 매개 변수와 계산량을 감소시키고, 과도한 학습을 막습니다.
* 전체 결합 층은 보통의 신경망과 같은 구조를 하며, 뉴런은 모든 뉴런과 연결이 되어 있습니다.

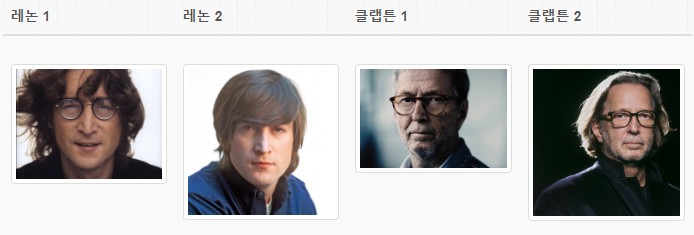
1. **필요성 및 기대효과**

얼굴을 인식하는 유용한 프로그램이 많이 출시가 되었다. 그로 인해 닮은 얼굴 찾기나 나이와 성을 예측하는 프로그램이 한 때, 출시가 되기도 하였습니다. 하지만, 얼굴의 특징점을 찾아 정교하게 작업을 해야 하는 성형외과 의사들의 말은 다르다.[[1]](#footnote-1) 표식점을 지정하는 점의 크기가 너무 크고, 정확한 표시점을 지정하는 것이 아닌 10mm~40mm 정도, 실제 얼굴의 정확한 계측점과는 차이가 많다는 말을 한다. 의사들은 자신이 수동으로 정확한 얼굴 표식점을 찍는 것이 정밀도와 정확도가 높다고 말합니다.

컴퓨터가 특징점을 표기를 하지만, 오차가 발생함에 따라 성형외과에서 자동화 3D 스캐너를 사용을 못 하고 있는 추세이다. 이런 오차 범위를 줄임으로서, 성형외과 같은 정밀한 곳에 사용할 수 있는 3D 스캐너가 사용될 수 있으며, 더 나아가 감정이 드러나는 표정의 패턴을 일어내어 바른 표정을 짓 도록 교정을 해주거나 얼굴에 순식간에 드러나는 감정 표현을 잡아내어, 상대방의 거짓말 탐지와 같은 곳에서도 사용을 할 수 있을 것이다.

1. **기존 연구**
2. 두 이미지 비교

* 비교 데모를 통해 그들의 표현 간의 제곱거리를 산출하여 양면의 예측 유사성 스코어를 출력한다. 이 때, 낮은 점수는 두 사람의 얼굴이 같은 사람의 가능성이 높다는 것을 나타낸다. 점수는 0~4로 부여.

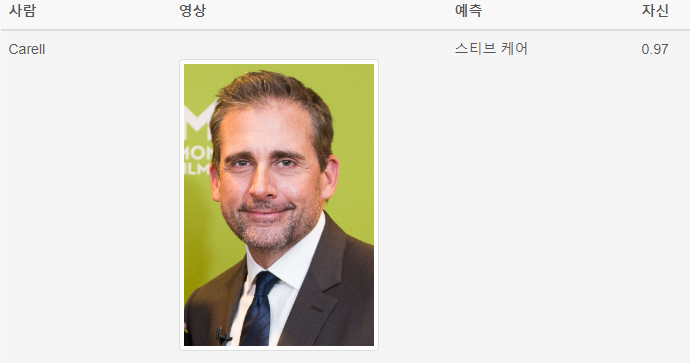




거리는 두 사람을 구별 할 수 있는 거리의 임계 값을 보여준다.

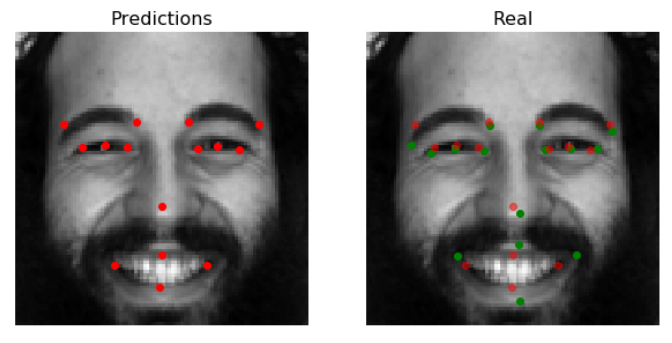
1. 분류 자 교육

* OpenFace의 코어는 얼굴의 저 차원 표현을 얻기 위한 특징 추출 방법을 제공한다. 특징 표현을 위한 심 신경 네트워크(DNN) 모델로 사람들을 분류하기 위한 모델을 훈련하는 것에는 차이가 있는데, 사전 훈련된 DNN 모델을 사용하여 분류 모델을 학습하고 사용하는 방법을 보여준다.

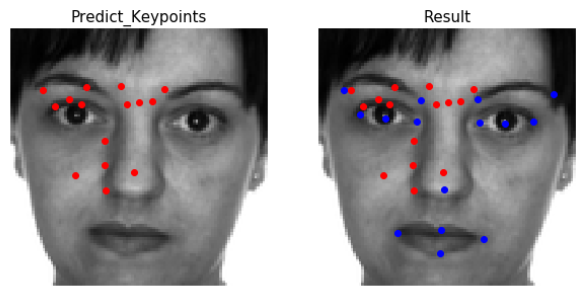


|  |  |  |
| --- | --- | --- |
| **작성자** | **점수** | **링크** |
| Karan Jakhar | 4.12930 | **https://www.kaggle.com/karanjakhar/facial-keypoint-detection?scriptVersionId=5720162** |
| Gustavo | 4.41109 | **https://www.kaggle.com/glarrea/face-detection** |
| Aktaruzzaman Aman | 4.08550 | **https://www.kaggle.com/aktaruzzaman/facial-keypoint-aman** |
| JCJ | **-** | **https://www.kaggle.com/swagmh/cnn-studying** |

1. Karan Jakhar – 데이터에서 누락된 값(NaN)에 대해 이전의 행의 값으로 채워, NaN 값을 없애고, 이미지 열 값은 문자열 형식으로 누락 된 값이 있으면, 문자열을 공백으로 분할하고 추가하여 누락된 값을 처리한다. 데이터 준비가 완료되는 과정에서 새로운 값을 예측할 때, mse(평균 제곱 오차)를 사용한다. 모델이 정의가 되어 함수를 호출하여 배치 크기와 유효성 설정 크기를 20%로 유지하면서 500회 반복 실행하는 과정을 통해 모델 검증을 한다.
2. Gustavo - 1번과 다르게 이미지 열 값에 누락된 값에 이전의 행의 값이 아닌, dropna 명령어를 통해 열 전체를 삭제를 하였습니다. 함수를 초기화 할 때, 1번과는 다르게, joblib 모듈(전처리 함수가 들어 왔을 때, CPU의 병렬처리를 도와주는 함수)을 통해 이미지의 크기는 96x96으로 만들어 주는 차이가 있다. Sklearn.model\_selection 모듈 의 서브 패키지인 tratin\_test\_split 을 통해 데이터를 학습용 데이터와 검증용 데이터로 분리한다. 이 점은 Titanic에서 test 데이터와 training 데이터를 합치는 방법과는 다른 방법이다. 후에 오픈 소스 신경망 라이브러리인 keras를 통해 필터로 특징을 뽑아주는 컨볼루션 레이어인 conv2D를 통해 눈, 코, 입의 좌표를 얻을 수 있다. 1번과는 다른 점은 keras를 사용하여, 특징점을 찾아 원본 데이터와 일치하는지 판별하는 과정이 들어가 있다.



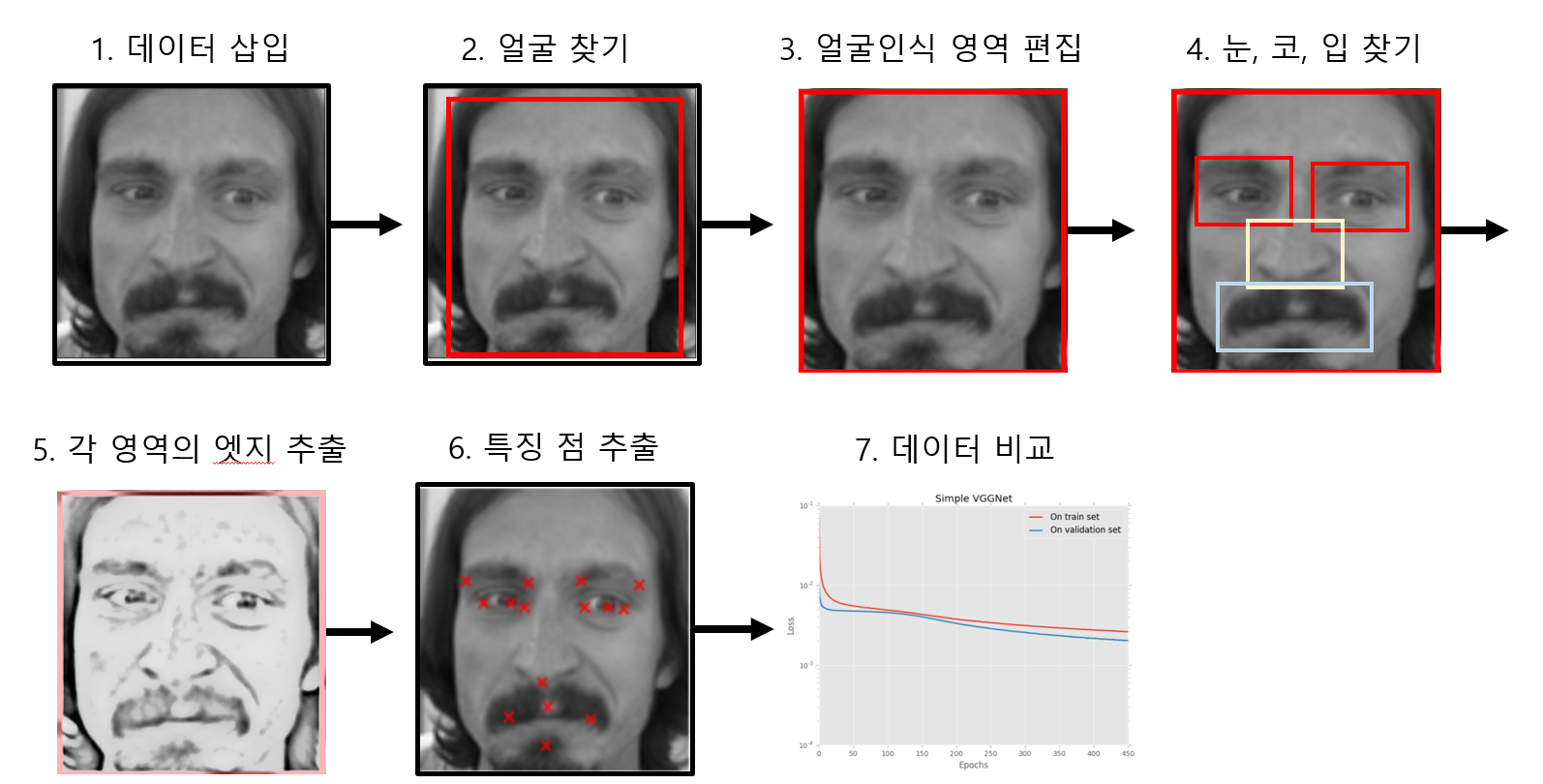
1. Aktaruzzaman Aman – 1번과 2번의 방식을 합친 방식이며, NaN 값을 처리하는 방식이 fillna를 사용하여, 마지막 유효한 유효성 검사를 전달한다는 방식을 사용한다는 점이다.
2. JCJ - 1번과 2번의 방식이다. 이 사람 경우에는 2번과 같이 특징점을 찾아 원본 데이터가 test 데이터와 일치를 하는지 판별을 하는 방식인데, 이 사람 같은 경우에는 함수의 초기화가(사진의 크기를 지정 2번의 경우 96x96) 되어 있지 않았다.



.

1. **제안 방법**

**수정 - OpenCV의 OpenFace를 사용**



1. 데이터를 삽입합니다. Test 데이터는 행 순서의 픽셀 목록으로 이루어져 있다. 이러한 데이터의 NaN 값이 들어가 있는지 확인을 합니다. 있다면, NaN 값들을 제곱근의 편차로 계산하여 값들을 집어 넣습니다.
2. OpenFace 모듈을 이용하여, 얼굴을 찾습니다. 이 때, 인식이 되는 faceID 만을 따로 모아 둡니다.
3. 얻어진 faceID를 얼굴이 인식이 되는 영역으로 편집을 하고, 영역의 크기를 96x96으로 만들어 줍니다.(80x80 이상 특징 점 을 찾음)
4. 편집된 영역에 대해 눈, 코, 입을 따로 찾습니다.
5. 각 영역의 엣지를 추출 합니다. 이 때, 엣지는 경계선을 인지하는 것을 말합니다. 즉, 이미지 안에서 픽셀의 값이 갑자기 변하는 곳이다. 엣지 추출을 하는 알고리즘은 이미지를 미분한 그레디언트 벡터의 크기로 판단을 하게 된다. 데이터의 노이즈를 Smoothing 방식으로 제거하고, 엣지가 아닌 픽셀을 지움으로서, 특징 점(코너)를 찾습니다. 특징 점, 즉 코너란 엣지가 교차되는 점이다. 코너를 찾는 방식[[2]](#footnote-2)은 Harris corner detector, Shi & Tomasi corner detection이 있다.

|  |  |
| --- | --- |
| Harris corner detector | 이미지 위에 마스크 행렬을 이동시키면서, 점(x, y) 주변에 대한 변화량을 구합니다. 변화가 있다면, 해리스 코너에서는 이를 코너로 판단을 한다. |
| Shi & Tomasi corner detection | 설정한 기준 값을 이용하여, 변화량에 대해 기준 값 보다 크면 코너라고 판단을 한다. |

1. 특징 점의 좌표를 저장합니다.
2. 최종 데이터를 비교 합니다.

**1차 수정 – CNN 사용 & 추가적인 학습 데이터를 생성하여, 학습**

1. 데이터를 삽입합니다. Training 데이터에 대해 15개의 좌표쌍과 image 픽셀 값으로 데이터가 이루어져 있으므로, image 픽셀 데이터를 따로 저장을 합니다. 이 때, 가져온 픽셀 데이터를 96x96 이미지로 변경 합니다.
2. 가져온 image 픽셀 데이터에 대해, 누락된 값(NaN)에 대해 해당 데이터 열을 삭제 합니다. (2140개)
3. 신경망을 구축 합니다. (CNN – Convolutional Neural Network)

드롭 아웃 이용 – 과도한 학습 방지, 훈련 오차가 0.00111인 반면, 검증 오류가 0.00123일 때, 훈련 오차와 검증 오류 값이 드롭 아웃을 통해 오버 학습을 막습니다.

1. 추가적인 학습 데이터 생성.

누락된 데이터를 포함하여 수평 반전을 통하여 얻을 수 유효한 데이터를 통해 학습을 합니다. 총 6가지 경우로서 좌, 우 눈의 좌표, 모든 좌표가 있는 경우 등..

1. 학습 시작 6번의 학습

학습의 개수는 500개로 정하고, overfiting이 발생하면 학습을 종료 합니다.

총 3000개의 학습량

1. 학습 곡선을 통해 학습 정도를 파악
2. Test 데이터 셋을 통해 오차 검증

**2차 수정 최종 – CNN 사용**

1. 데이터를 삽입합니다. Training 데이터 총 7049개의 데이터의 15개의 좌표와 image 픽셀값이 공백으로 이루어져 있으므로 이 값들에 대해 분리를 시켜 이미지를 저장합니다. 분리된 이미지 픽셀 값에 대해 가로, 세로의 픽셀 값인 96x96 으로 변경시키고, gray Scale 이미지 이므로 채널은 1로 설정합니다.
2. Training 데이터의 누락된 데이터의 값에 대해 fillna 명령어를 통해 Nan값이 아닌 값을 전파 시킵니다.
3. 신경망을 구축합니다. 2차원 데이터의 합성 곱 연산 이용.
4. 100개의 데이터를 학습.
5. 학습된 데이터의 test 데이터를 예측
6. IdLookupTable 데이터에서 RowId, Location 을 추출하여, 예측한 데이터 값 적재 및 제출 파일 제작.

1. **실험 및 결과(비교실험)**
2. 데이터베이스

training.csv :  교육을 하기 위한 7049 이미지 목록을 제공합니다. 이 이미지 목록은 픽셀 형식으로 되어 있습니다.

각 행은 15 개의 키포인트에 대한 (x, y) 좌표와 행 순서의 픽셀 목록으로 된 이미지 데이터를 포함하고 있습니다.

* 사용 용도로 기계를 학습 시키는 데이터로 사용합니다.
* 15개의 키 포인트 : left\_eye\_center, right\_eye\_center, left\_eye\_inner\_corner, left\_eye\_outer\_corner, right\_eye\_inner\_corner, right\_eye\_outer\_corner, left\_eyebrow\_inner\_end, left\_eyebrow\_outer\_end, right\_eyebrow\_inner\_end, right\_eyebrow\_outer\_end, nose\_tip, mouth\_left\_corner, mouth\_right\_corner, mouth\_center\_top\_lip, mouth\_center\_bottom\_lip

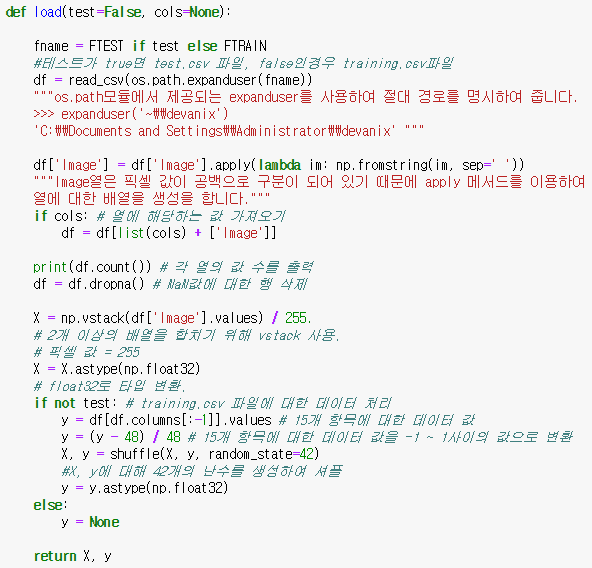
요약 : training 데이터는 7,049개, 96x96 픽셀의 gray 스케일 이미지로 구성됩니다.

Test.csv :  1783 개의 테스트 이미지 목록. 각 행에는 ImageId 및 이미지 데이터가 행 순서의 픽셀 형식으로 되어 있습니다.

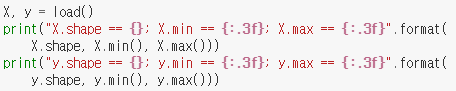
* Training.csv 파일의 학습한 데이터를 토대로 test 파일의 특징점을 찾습니다.

제출용 파일의 형식은 각 행에 RowId, ImageId, FeatureName, Location 내용이 들어 갑니다. Location에 해당하는 것은 학습을 통해 예측된 위치가 들어 간다.

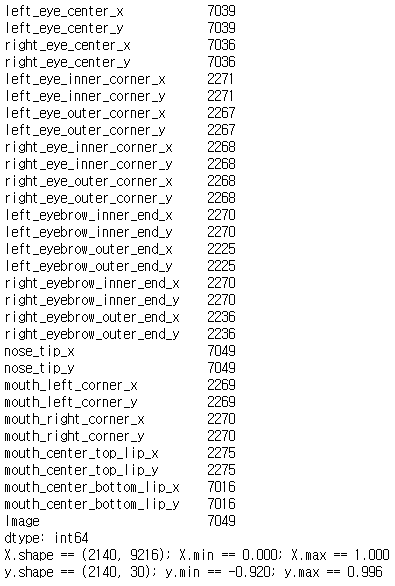
대회의 목적은 left\_eye\_center, right\_eye\_outer\_corner, mouth\_center\_bottom\_lip 같은 keypoints (얼굴의 특징적인 위치) 15 개 (각각 x, y 좌표를 가진)을 학습하고 추정합니다



픽셀의 밝기 값을 255로 나눔으로서 [0,1] 간격으로 조정을 합니다.



CSV 파일에서 데이터를 읽어들여 X, y에 대해 값을 수정합니다.



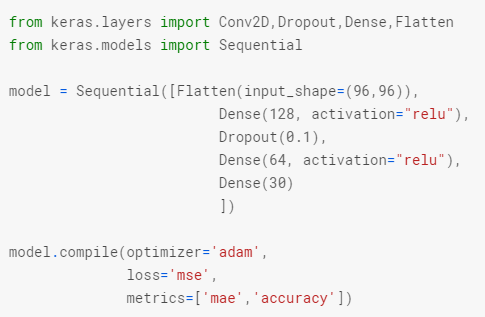
15개의 항목에 대해 데이터의 존재 여부를 파악 할 수 있습니다. Dropna()명령을 통해 NaN값이 들어간 keypoint 값은 제거 되어 존재 하는 샘플의 데이터는 2140개 인 것을 알 수 있습니다.

**상위 방법**

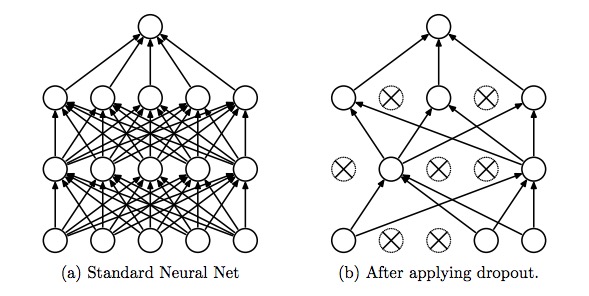
Karan Jakhar : score - 4.12930

**https://www.kaggle.com/karanjakhar/facial-keypoint-detection?scriptVersionId=5720162**

Kernel의 작성자의 데이터 준비 단계는 데이터의 null 값을 찾아 해당하는 데이터를 삭제 시키며, 해당하는 사진의 타입을 gray, 크기를 96x96으로 변경 시켰습니다. 데이터 준비가 완료되어 아래와 같이 모델을 정의 합니다.



Keras를 이용합니다. 케라스는 파이썬으로 작성된 오픈 소스 신경망 라이브러리 입니다. 여기서는 keras의 drop out을 사용합니다.

(Nitish SrivastavaGeoffrey, 2014.11.13)

Drop out은 과도한 학습을 방지하고 매우 효과적이고 간단한 방법으로 사용이 됩니다. 그림과 같이 무작위로 뉴런을 무시하고 학습을 진행하는데 있어 매우 다양한 신경망을 학습시키고 있다.

신경망 구현 순서는

1. Sequential : 모형 클래스 객체 생성
2. Add : 메소드로 레이어 추가
3. Compile : 메소드로 모형 완성
4. Fit : 메소드로 트레이닝



배치크기와 유효성 설정의 크기를 20%로 유지하면서 500회 반복 실행을 합니다.

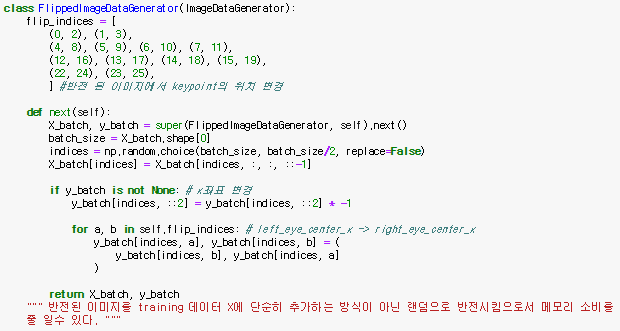
**현재 상위 방법과 비교**

1. Harris corner detector, Shi & Tomasi corner detection 알고리즘을 통해 특징점을 찾았습니다. 다른 사람들과 방법을 비교해 보면, 테스트 데이터를 학습하여, 특징점을 찾은 것에 반해 저의 경우에는 알고리즘을 통해 추출한 데이터를 이용하여, 특징점을 찾았습니다.
   1. 결과 : 오차가 32이라는 점에서 다른 사람들의 오차 범위의 8배에 달하는 숫자이므로, 적절하지 못하다는 판단.
2. 7049개의 이미지 데이터 중 15개 항목의 데이터가 없어 사용하지 못하는 데이터를 제외 하고, 2140개의 데이터를 가지고 학습을 시킨다. 그런데, 2140개의 데이터를 dropout 하여, 학습을 시키는데 있어 학습량을 증가를 시키면, 좀더 정확한 좌표를 예측을 하게 되어 오차를 줄일 수 있을 것이라는 생각을 가졌습니다.



이 딕셔너리는 누락된 데이터에 대해 overfiting[[3]](#footnote-3)의 중요성을 설명합니다. Overfiting은 특정 데이터 세트에 가깝거나 정확하게 일치하는 데이터를 추가 생성 하는 것을 말 합니다.

Load\_data() 함수가 추출 할 열 목록에 대한 데이터 입니다.



앞서 만들어진 딕셔너리의 flip\_indices 튜플의 정의 입니다.

제공 받은 훈련데이터에서 반전, 회전을 통해 이미지를 생성하고, training 데이터로 사용을 할 것인데,

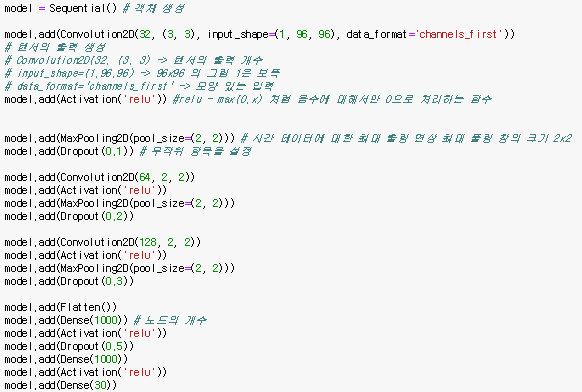


왼쪽 : 원본, 오른쪽 : 수평 반전

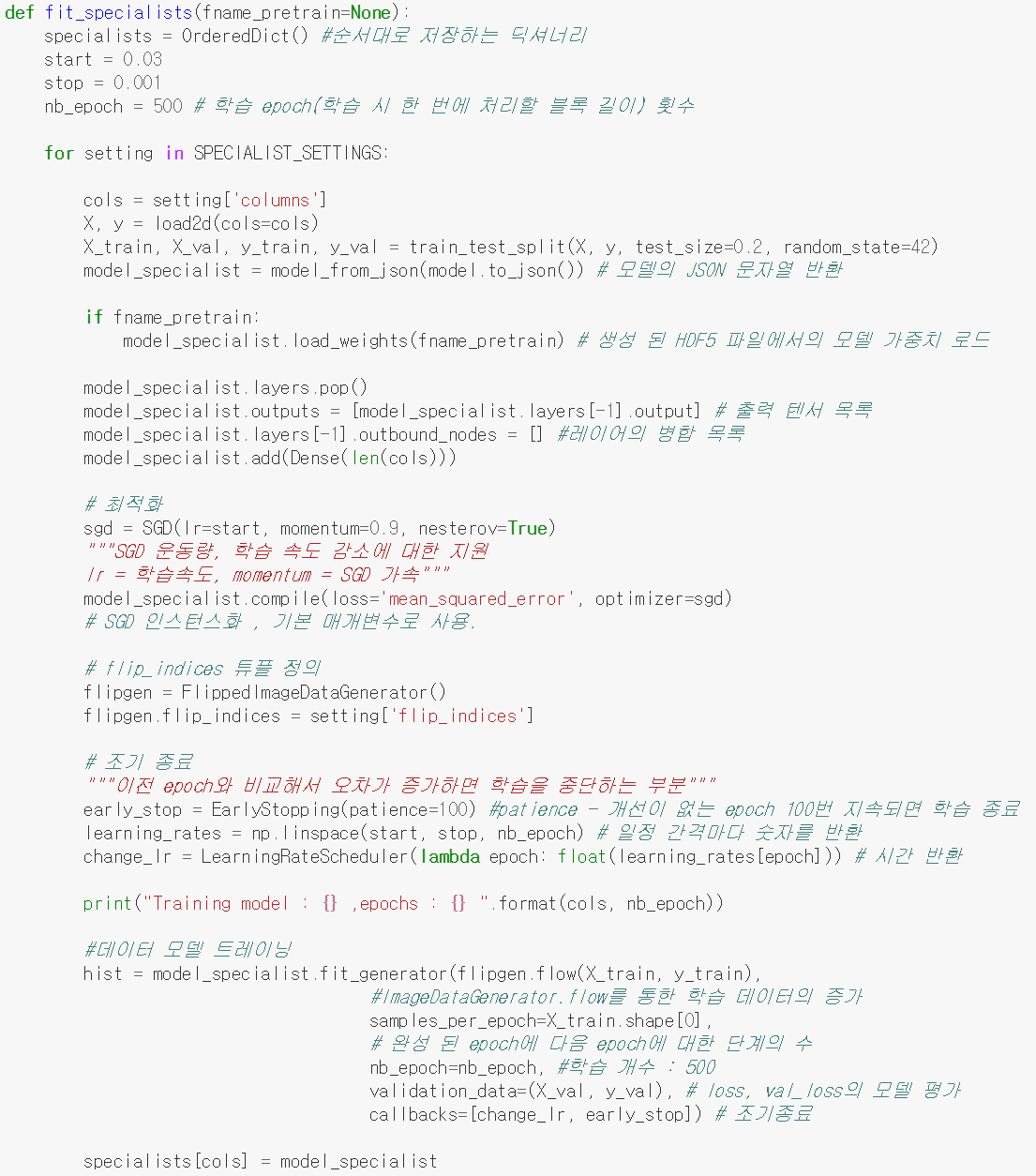
반전 된 이미지에서 keypoint의 위치가 어긋난 것을 확인할 수 있습니다. 이미지와 keypoint의 위치도 변경을 해야 하는데, 단순히 생각하면 left\_eye\_center\_x와 righit\_eye\_center\_x의 값을 변경을 하고 nose\_tip\_y(코) 값은 변경을 할 필요가 없다는 것을 알 수 있습니다. 그래서 이 flip\_indices튜플을 통해 이미지를 수평으로 반전을 시켰을 때, 해당하는 배열 요소 번호의 어떤 영역이 변경이 되는지에 대한 정보를 담고 있습니다.

Ex) 튜플 (0, 2) = left\_eye\_center\_x, right\_eye\_center\_x

flip\_indices튜플과 딕셔너리를 통해 일부keypoint를 예측하는 모델의 분리를 통해 데이터를 생성하여 활용을 할 수 있습니다.



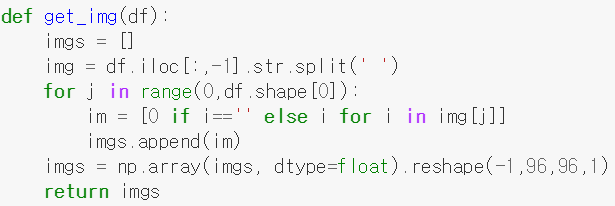
상위 방법과 마찬가지로 신경망을 구축 합니다.



좋은 결과를 얻지 못하였습니다. 6번의 학습을 진행하는 동안, 조기 종료를 택하였는데, 실제로는 3000개 보다 더 적은 수로 학습을 완료(CPU – 40시간, GPU – 1시간)를 하였습니다.

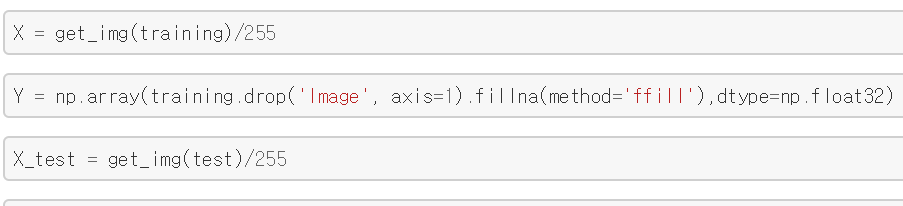
예측된 결과에 대해, 정확한 좌표를 얻을 수는 있었지만(통계상으로) 문제는 제출 파일 형식으로 변환할 때 발생하였습니다. 수평 반전으로 생성된 데이터에 한해 음수의 좌표 값이 발생하였습니다. Math 모듈의 abs()함수를 통해 해결을 하였지만, 그것은 잘 못된 결과로 이어졌습니다.

1. 최종적으로 7049개의 training 데이터에서 Nan값들에 대해 fillna 명령어를 통해 이전의 값들로 대체를 시키는 것으로 시작되었습니다.



우선 데이터를 일반화 시키는 함수를 만들었습니다. -> test 데이터와 training 데이터의 image에 대한 일반화

픽셀로 이루어진 image 값들은 공백으로 이루어 져 있는 것을 위에서부터 확인을 해왔습니다. 그에 따라, 공백으로 나누어 리스트에 저장을 합니다. 이에 따라 이미지를 출력하기 위해 reshape함수를 통해 배열의 모양을 변경합니다.

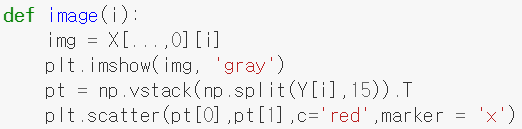


X 변수에는 training 데이터의 이미지 픽셀 값이 0과 1로 이루어져 있는 값으로 변경합니다.

Y 변수에는 training 데이터의 15개의 좌표 쌍에 대해 Nan값들을 fillna함수를 통해 이전의 값으로 변경을 해주고 float형으로 타입을 지정하여 줍니다.

X\_test 변수에는 test 데이터의 이미지 픽셀 값에 대해 0과 1로 이루어진 값으로 변경합니다.

0과 1로 변경을 하는 이유는 우리는 gray 스케일의 이미지를 사용하기 때문입니다.

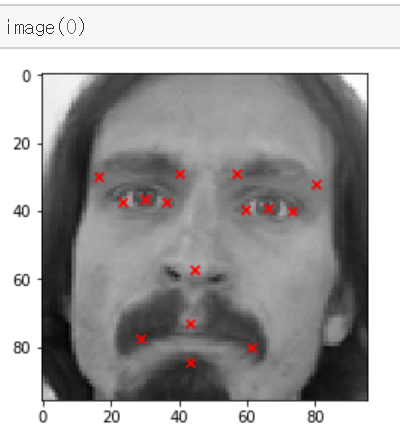


Training 데이터의 이미지를 출력하는 함수를 정의 합니다.

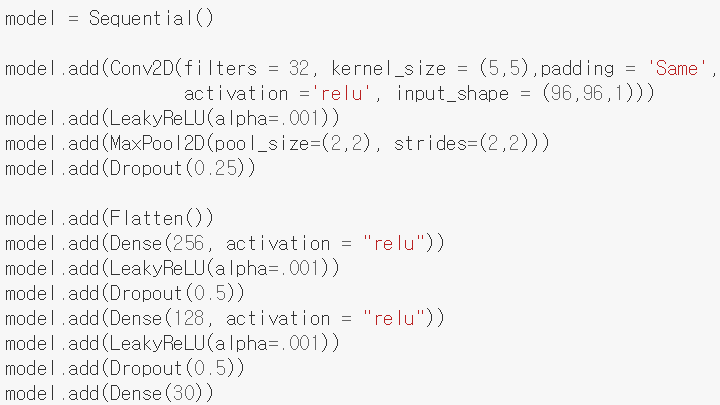
Imshow 함수를 통해 배열의 값과 gray 스케일 이미지를 출력합니다.

그리고 좌표들을 흩 뿌려주기 위해 scatter함수를 사용합니다. 이 때, Y에는 15개의 좌표쌍이 저장되어 있는데, 열의 수가 같기 때문에 vstack 명령어를 통해 값들을 묶어 주게 됩니다. 마커는 x 색상은 빨강으로 이미지를 보여 주게 됩니다.

이에 따른 결과는 아래와 같습니다.



가장 중요한 부분은 신경망을 구축하는 데 있다고 생각합니다. 신경망이 어떤 모양을 가지고 있는지에 따라 학습 능력이 변경이 되기 때문입니다.



우선 객체를 생성을 합니다. 객체가 생성됨에 따라 Convolution2D 레이어를 정의 합니다. 입력은 96x96으로 배열을 정의 해 주었으므로 input\_shape를 정의 해 줍니다. 이 때, gray Scale이미지이기 때문에 채널은 1로 설정하여 줍니다.

Activation 활성화 함수를 설정합니다. 종류는 linear, relu, sigmoid, softmax로 이루어져 있습니다.

|  |  |
| --- | --- |
| Linear | 입력 뉴런과 가중치로 계산된 결과값이 그대로 출력으로 나옵니다. |
| Relu | 은닉층에 주로 사용. |
| Sigmoid | 이진 분류 문제에서 출력층에서 사용 |
| softmax | 다중 클래스 분류 문제에서 출력층에 사용. |

일반적으로 sigmoid를 사용하지만, **Gradient Vanishing**문제로 인해 Relu가 사용됩니다.

패딩을 통해 출력 이미지 사이즈가 입력 이미지와 동일하게 지정을 해줍니다.

후에 32개의 필터를 가진 5x5 의 커널을 생성합니다.

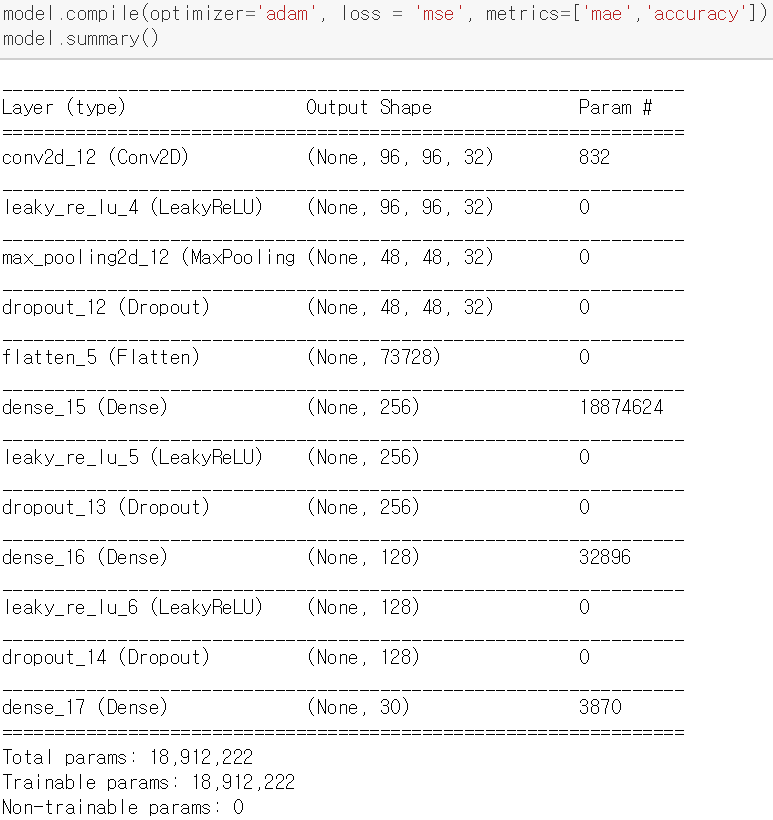
사소한 변화가 출력 이미지에 영향을 미치지 않도록 하기 위해 MaxPooling을 지정합니다. 이 때 pool\_size를 지정해 주어 수직, 수평 축소 비율을 지정합니다.

2차원 자료를 전결합층에 전달하기 위해 1차원 자료로 변환을 해주기 위해 Flatten()를 사용합니다.

Dense를 이용해 입력된 뉴런 수에 상관 없이 출력 뉴런 수를 자유롭게 설정할 수 있습니다.

X가 음수이면 Gradient가 무조건 0이 된다는 단점이 있습니다. 이를 해결하기 위해 Leaky ReLu를 이용하여 가중치를 0.001로 설정 합니다.

Overfiting을 줄이기 위해 dropout를 이용합니다.



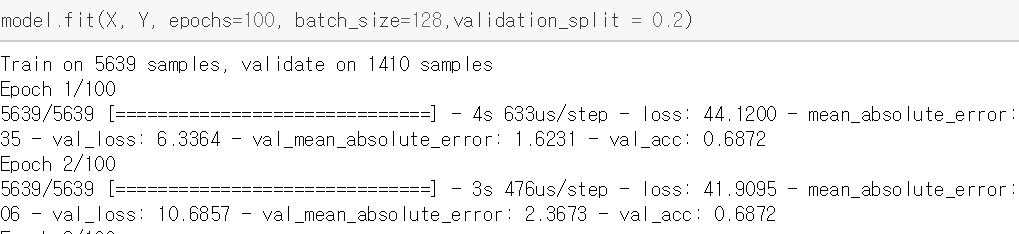
지정한 모델이 효과적으로 구현이 될 수 있도록 환경을 설정을 해주고 컴파일을 합니다.

keras모델을 컴파일을 하기 위해 optimizer을 설정을 해주어야 합니다. 여러 optimizer을 설정할 수 있지만, 계산 상 효율적이며, 메모리에 요구가 적은 Adam을 사용하였습니다.

Loss 인자를 포함합니다. 비용함수라고 하는데, 이는 예측 값과 원래 값에 대한 차이의 합을 나타낸다. 이때 계산하는 방식으로 mse를 사용하였습니다.

마지막으로 metrics 인자를 통해평가 기준을 지정합니다. 일반적인 평가 기준인 accuracy를 설정해 주고, mae를 추가로 설정을 해주어, 결과 값과 예측 값이 얼마나 비슷한지 평가를 해주게 됩니다.

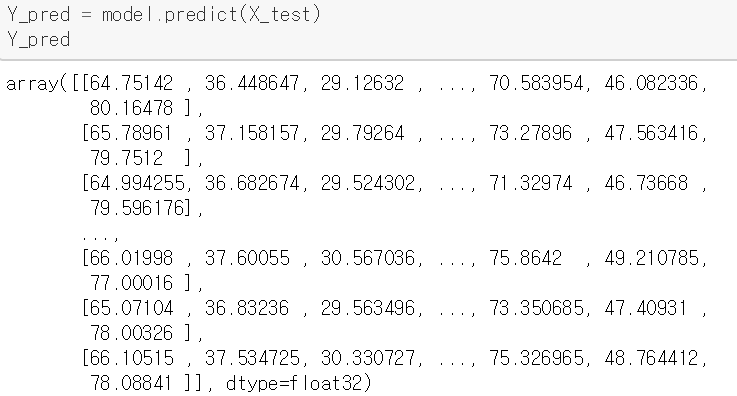
Summary 함수를 통해 모델의 요약된 표현을 출력합니다.



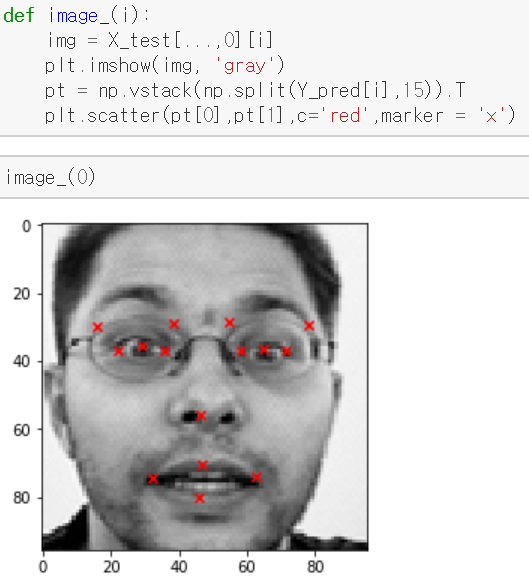
모델 학습을 위해 fit 함수를 사용합니다.

학습데이터는 100개로 지정을 하고, 한 번에 사용하는 데이터 개수를 128개로 지정합니다.

Validation\_split 인자를 통해 데이터 셋을 얼마의 비율로 나눠 트레이닝 데이터셋과 테스트 데이터 셋으로 사용할지 결정 할 수 있습니다.



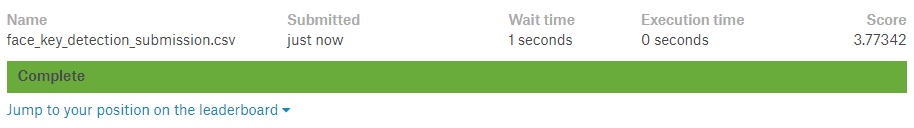
Test 데이터를 집어 넣고 모델을 예측해 봅니다. 수치를 보면 우선 0~96 사이의 값으로 잘 표현되어 있는 것을 확인할 수 있었습니다.



Training 데이터에서 이미지를 출력하기 위해 사용한 함수와 동일하게 함수를 정의 하고 학습된 데이터에 대해 키 포인트를 예측해 봅니다.

나름 예측이 잘 되었다고 생각 합니다.

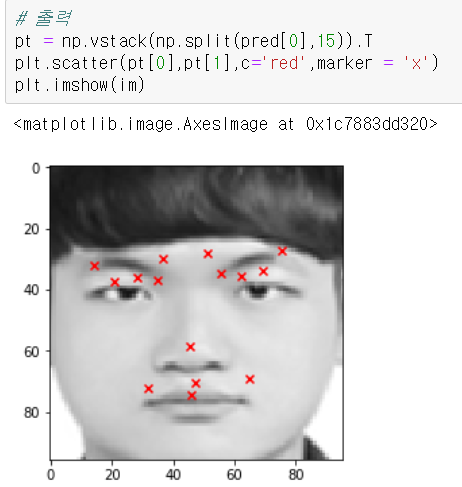
제출할 csv 파일을 만들고 데이터를 삽입하여 줍니다. 그런 후, kaggle에서 테스트를 해봅니다.



점수는 3.77342를 받았습니다. 앞에서 kernel 들에 대한 점수를 확인해 보았을 때 최대 0.7 최소 0.3 정도 성능이 개선이 되었습니다.

신경망을 구축할 때, 방식만 잘 선택한다면, 좀 더 개선된 점수를 받을 수 있을 것 같습니다.

***기능 추가***



학습된 데이터를 h5로 만들어 두어 예측 값을 쉽게 구할 수 있었습니다. 점이 많이 어긋난 이유는 픽셀로 데이터를 변환하는 과정에서 배열을 재정의 할 때, 기존의 training 데이터와 크기와 위치가 미세하게 다르기 때문입니다.

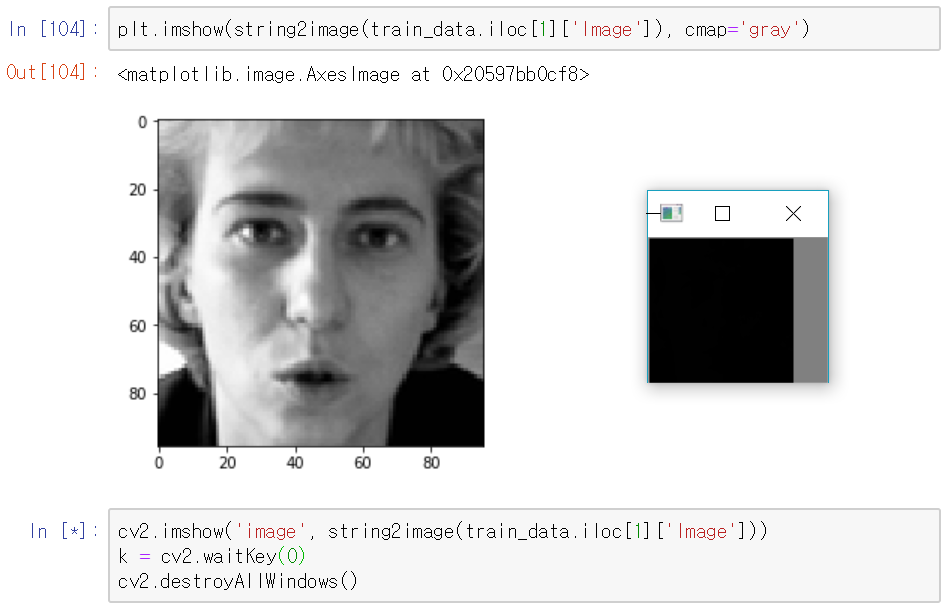
다음으로 이미지에 다른 이미지를 합성을 해 보았습니다.



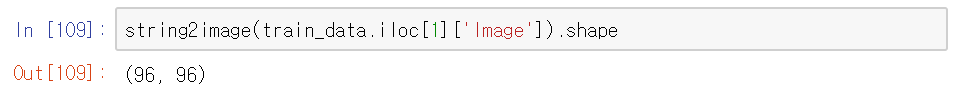
콧수염 이미지 파일은 구글을 통해 얻을 수 있었습니다.

그에 따라 예측된 지점과 콧수염의 중앙 값을 통해 이미지를 합성을 할 수 있었습니다.

1. **고찰(토론)**
2. opennCV를 이용하여, 읽어 온 데이터의 얼굴을 인식하기위해 시도를 해보았습니다. 일이 순조 롭게 해결이 될 줄 알았지만, 그것은 실패로 이어졌습니다.



marplotlib으로 확인해 보면 픽셀 데이터를 그려주지만, OpenCV를 이용하면 출력이 되지 않습니다. 왜 그런지 이유를 찾아 보았습니다.



Shape 명령어를 통해 사진의 형태를 확인 할 수 있습니다. 96x96 크기라는 의미 인데, 이 과정은 함수를 통해 재정의를 해주었기 때문입니다.

본론으로 들어가. openCV 모듈을 사용 할 때 cv2.imread()함수를 이용하여 이미지 파일을 읽어 들이게 됩니다. 이 때, 들어오는 파일은 우리가 흔히 사용하는 jpg, png등의 이미지 포맷 형태 입니다. 함수가 실행 되면, Numpy의 ndarray type을 반환 합니다. 배열을 함수로 처리를 해주었기 때문에 openCV를 일반적으로 사용을 할 수 있을 것 같다는 생각이 들었습니다.

openCV에서는 BGR로 색을 표현을 하게 되는데, 이는 3차원 행렬로 반환이 되는 것에 해당 됩니다. shape 명령어를 통해 확인을 했듯이 이미지의 사이즈 가 96x96 라는 이외의 정보를 얻지 못하였습니다.

그러면, 3차원 행렬로 정의를 해주면 되는 것이 아닙니까 ?

* cannot reshape array of size 9216 into shape (96,96,3)
* reshape를 통해 정의를 해주는 과정에서 조건이 발생합니다. 원래의 배열에 있는 원소가 즉, 픽셀 데이터 값들이 배열의 차원에 빠짐 없이 분배가 되어야 하는데, 재구조화 과정에서 호환이 되지 않기 때문에 오류가 발생합니다.

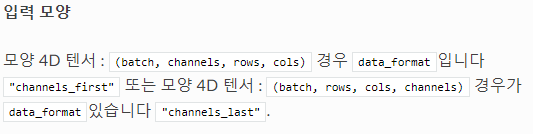
2019.01.09 해결 : dtype=np.uint8으로 지정.

2) model.add(Convolution2D(32, 3, 3, input\_shape=(1, 96, 96)))

오류 발생

ValueError: Negative dimension size caused by subtracting 3 from 1 for 'conv2d\_2/convolution' (op: 'Conv2D') with input shapes: [?,1,28,28], [3,3,28,32].

원인 : Convolution2D는 입력 값이 샘플, 행, 열, 채널로 구성.



2019.01.14 해결 : Convolution2D 레이어를 선언할 때, 선택적 키워드를 사용하여 문제를 해결할 수 있다.

model.add(Convolution2D(32, (3, 3), input\_shape=(1, 96, 96), data\_format='channels\_first'))

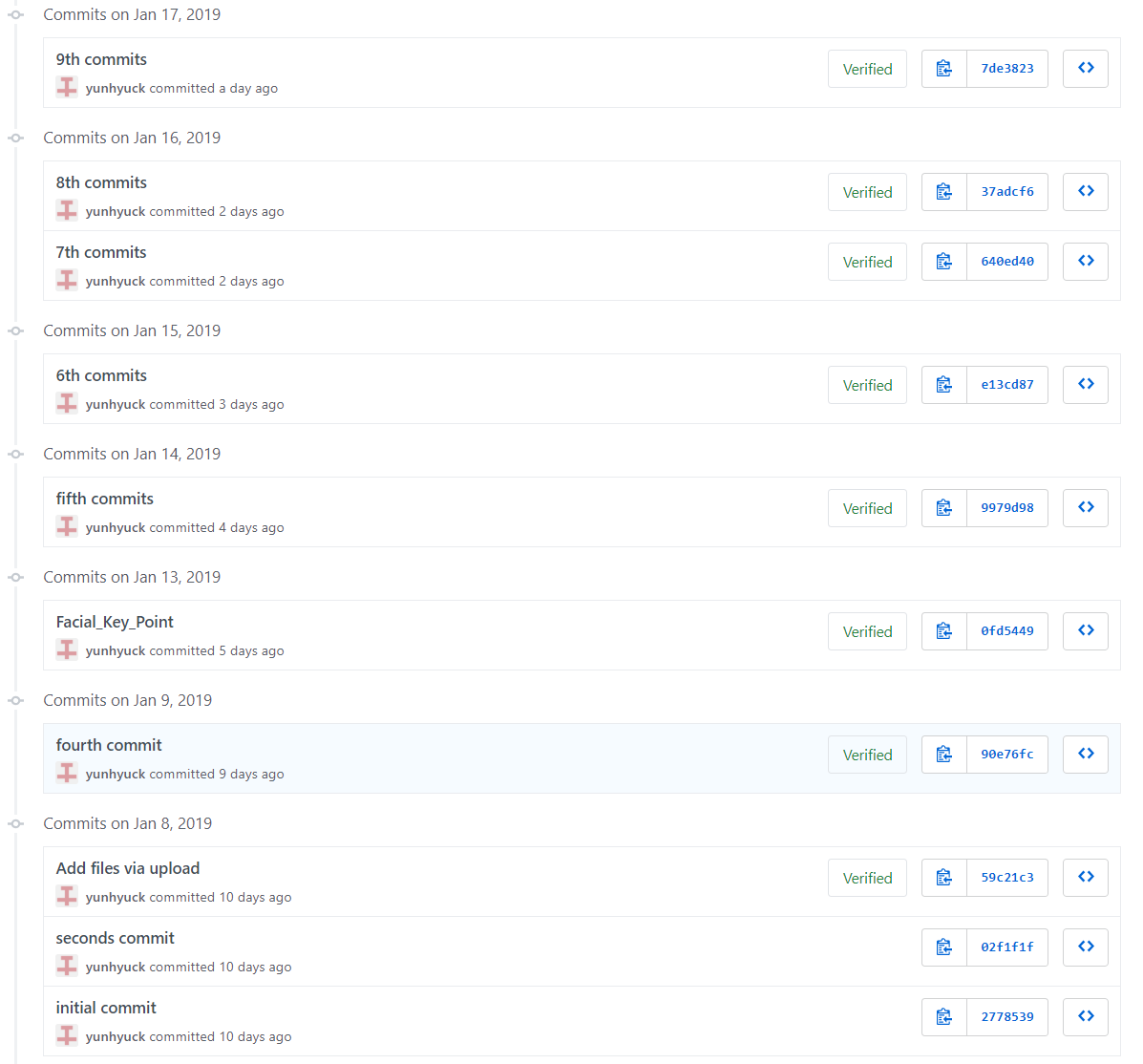
**참고 문헌 :**

[1] Facial Keypoint Detection Competition. Kaggle, 7 May 2013. Web. 31 Dec. 2016. https://www.kaggle.com/c/facial-keypointsdetection.

[2] Nouri, Daniel. 2016. Github. Kaggle Facial Keypoints Detection tutorial. Available from https://github.com/dnouri/kfkd-tutorial/blob/master/kfkd.py

[3] Facial Keypoint Detection Competition. Shayne Longpre, Ajay Sohmshetty Stanford University, 13 March 2016.

1. **느낀 점**



[**https://github.com/yunhyuck/OpenSourceProject/commits/master**](https://github.com/yunhyuck/OpenSourceProject/commits/master)

프로젝트를 진행하면서, 오픈소스의 중요성에 대해 많이 느껴 졌습니다. 전자제품에 사용되는 소프트웨어부터 상용 소프트웨어까지 오픈소스가 사용되는 곳이 정말 많다고 느껴졌습니다. 이번 머신 러닝을 통해 특징점을 추출하는 데, 어떻게 학습을 하는지에 따라 결과가 많이 다르다는 것을 느꼈습니다. 처음부터 생각대로 해보고, 결과가 나와 시도를 해 보았을 때, 오차가 32배나 차이가 난다는 것에 절망감에 빠졌습니다. 하지만, 이런 경험과 다양한 사람들의 소스 분석을 통해, 여러 데이터를 학습시키는 방식을 보고, 오픈소스는 넓은 세계라는 것을 알게 되었습니다. 가장 중요한 점은 라이선스에 대해 알고 있어야 상용 프로그램을 만들 때, 발생하는 저작권 문제에 대처 할 수 있다는 것입니다. 가장 기억에 남는 것은 학습할 데이터가 많으면 많을수록 오차는 줄어 들지만, 어느 지점에 발생하는 overfiting에 대처하는 방식이 가장 중요한 점이라고 생각합니다. 이번 프로젝트를 마치면서, 새로 배운 지식을 통해 앞으로 할 프로젝트에 응용을 한다면 좋을 것 같다는 생각을 가졌습니다.

학습 데이터 셋을 늘리는 것은 오차를 줄일 수 있을 것이라는 생각을 가졌습니다. 그렇지만, 개인적인 생각으로 신경망의 설계를 잘 못 했기에 발생한 결과라고 생각 합니다. 학습 시간이 많이 걸린 만큼 개선될 가능성이 높기 때문에, 이번 프로젝트를 마치고 마저 실험을 해 보고 싶습니다.

1. https://blog.naver.com/artprs/220993600148 [↑](#footnote-ref-1)
2. https://datascienceschool.net/view-notebook/7eb4b2a440824bb0a8c2c7ce3da7a4e2/ [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/Overfitting [↑](#footnote-ref-3)