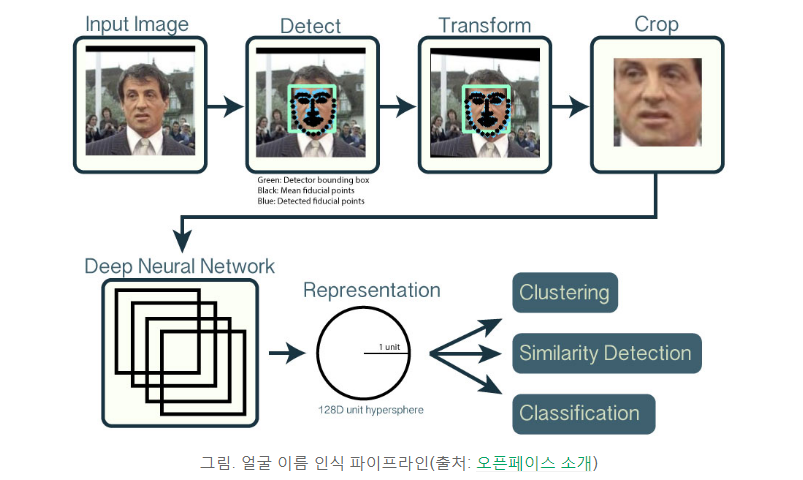
<Face Recognition in Family members>

Process.

1. Hog Algorithm to find face from images.
2. Create Cropped image using FLE Algorithm.
3. Conv-Pool layer to train.
4. Visualize neural network decision with CAM.

**1. 프로그램의 구성 및 설명**

먼저 얼굴인식 프로그램을 구현하기 위해 선택한 일련의 과정을 그림으로 나타내면 다음과 같다.



1. Input으로 Image가 들어온다.

2. Image에서 얼굴을 detect한다.

3. 기존에 학습을 통해 정해져있는 Template Landmark를 이용하여 Input-image 에서 Landmark를 찾아낸다. \* Landmark : 얼굴에 존재하는 68개의 feature point

4. Image에서 찾은 Landmark를 기준으로 Image를 Rotate 및 Scaling 을 통해

affine transform(Linear transform & translation) 을 수행한다.

5. Affine Transform을 통해 Input-Landmark를 기준으로 Image Cropping을 수행한다.

\* Cropping : Image 내에서 불필요한 영역을 잘라냄

6. Cropping된 aligned-image를 DNN(심층신경망)을 통해 학습을 시킨 후 image를 대표하는 feature 128 or 64 개의 point를 추출한다. 그리고 이 point들로 사람을 분류하기 위해 Face Classification을 수행한다.

이 프로그램에서는 위의 과정을 기반으로 얼굴인식을 구현하였으며

**(1) Face Detection 및 Cropping / (2) Face Clasffication 및 CAM** 크게 이 두 가지로 과제를 나누어 최종적으로 Image에서 얼굴을 검출한 후 어떤 사람인지 분류하는 작업을 진행하도록 하였다.

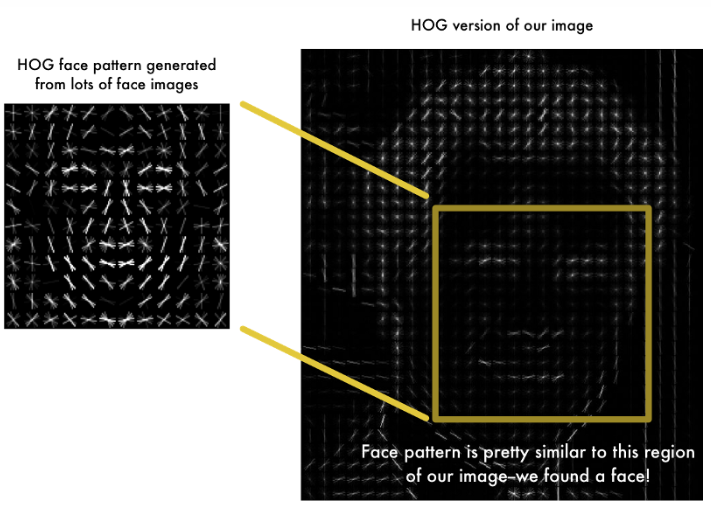
(1) Face Detection 및 Cropping

* **Face Detection**

Image 혹은 영상에서 특징을 검출하는데에 여러 알고리즘이 존재한다. (SIFT, HOG, Haar, 등..)

이때 얼굴인식을 하기위해서는 조명, Head pose 에 민감하지 않게 어떤 상황에서든 얼굴 feature을 찾아 Face Detection을 할 수 있는 알고리즘을 사용해야하는데 이 프로그램에서는 조명변화에 크게 의존하지 않는 HOG알고리즘을 선택하였다.

HOG알고리즘에서는 Image를 Gray scale Image로 변환한 후 Image에서 gradient field를 생성하여 기등록되어 있는 HOG패턴과 유사한 패턴을 찾음으로써 얼굴을 검출한다.

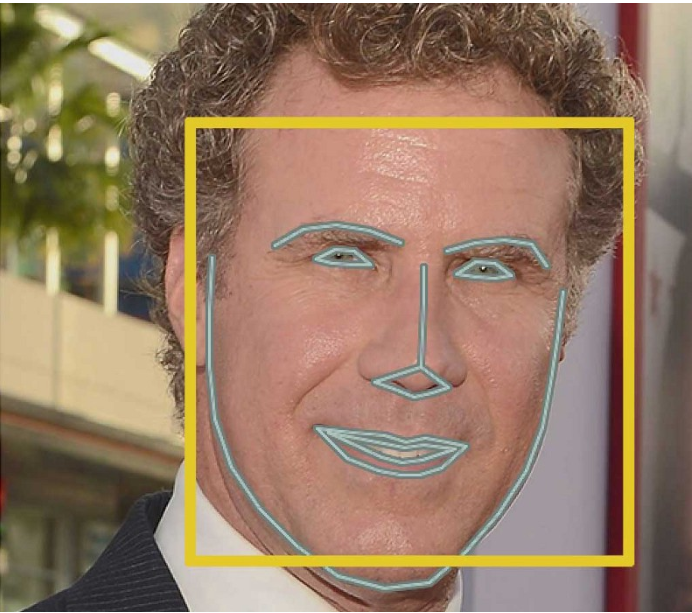
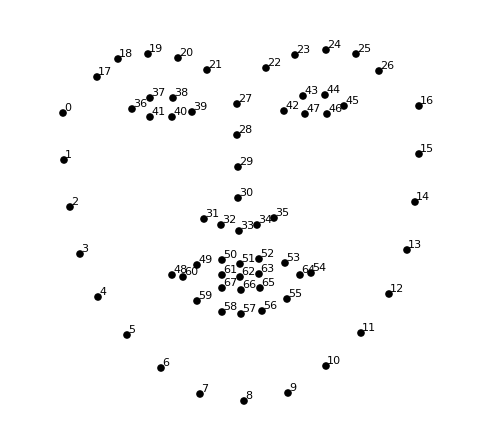


**(그림 1-a)**

* **Cropping**

검출된 얼굴에서 Cropping을 하기위해 눈코입을 검출해야한다. 이 작업을 하기 위해

Face landmark estimation(FLE) 알고리즘을 선택하였다. FLE알고리즘은 HOG 알고리즘과 유사하게 기존에 학습을 통해 만들어져있는 Template Landmark Point들을 이용해 Input-Image와 비교를 거쳐 Input-Image에서 Landmark point들을 찾아낸다.



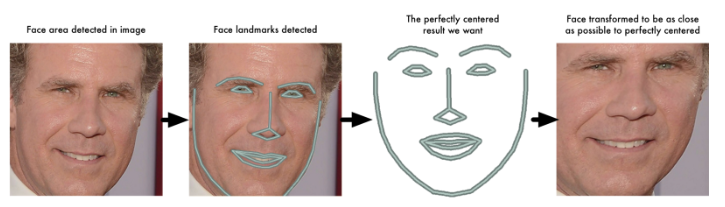
**좌) Openface 라이브러리에서 정의한 Template Landmark Point**

**우) Template Landmark Point를 이용해서 Input-Image에서 Landmark Point 검출한 결과**

**(그림2-a)**

위 그림2-a의 Template Landmark Point를 이용해서 Input-Image 에서 Landmark-point를 검출한다.

검출한 Landmark Point가 Image의 정중앙에 오게끔 rotate, scaling 을 통해 affine transform을 수행하여 Input-Image를 Crop한다. 이 과정을 통해 Input-Image에서 Aligned-Image를 생성한다.



**(그림2-b)**

(2) Face Classification 및 CAM

* **Face Classification**

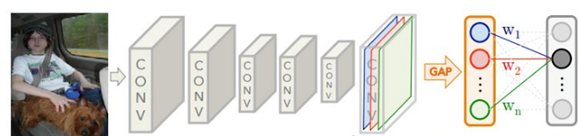
(1) Face Detection 및 Cropping 을 통해 Input-Image에서 Aligned-Image를 얻었다.

이제 이 Aligned-Image를 이용해 이 Image가 실제로 어떤 사람에 속한 Image인지 구별하는 과제를 해결해야한다.

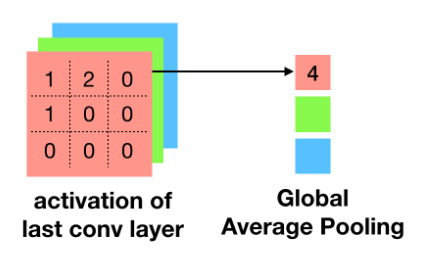
Image Classification을 위해서는 일반적으로 CNN에 의한 신경망 모델을 이용하여 학습을 진행하고 결과적으로 분류 작업을 수행한다. Face Classification을 위한 신경망 모델로는 대표적으로 Openface 라이브러리에서 제공하는 vgg16, vgg19 모델이 있다. 그러나 이 모델들은 약 1000명 이상의 사람을 구분하기 위한 모델이기 때문에 신경망 layer를 아주 깊게쌓고 방대한 Dataset을 이용하여 학습시킴으로써 사람들의 세세한 feature까지 구분하여 분류할 수 있도록 설계되어있다.

그러나 첫째로 이 프로그램의 목적은 binary classification으로써 한 사람에대한 인식 혹은 인식 실패 작업만 수행하면 되는데 위의 모델들은 프로그램의 목적에 너무 과하게 설계되어있다 생각이 들었다. 둘째로 기존에 딥러닝을 공부하던 중에 있었기 때문에 이 과제를 통해 직접 신경망 모델을 설계하고 과제를 해결해봄으로써 경험과 실력을 향상시키고자, 이 두 가지 이유로 신경망 모델을 직접설계해 Face Classification을 수행해보기로 결정하였다.

직접 설계한 신경망 모델은 아래 (그림3-a) 를 기반으로 설계하였다.



**(그림3-a)**

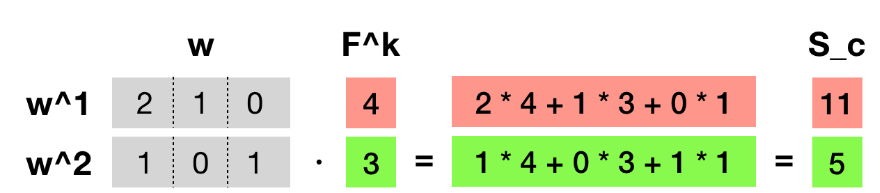


**(그림3-b)**

(128x128) Image가 input으로 들어오면 Convolution-Pooling layer 3개를 거쳐 (16x16) Image Channel 64개를 생성한다. 그 후 마지막 Convolution layer 1개를 거쳐 위와 동일한 (16x16) Image Channel 64개를 만든다. 최종적으로 (그림3-b)와 같이 (16x16) Image Chaneel value들의 평균값을 구해 1개의 value를 반환시킨다. 이렇게 64개의 GAP(Global Average Pooling) value들이 생성되면 Dense layer를 통해 2개의 node(인식 / 인식실패) 로 분류를 수행한다.

* **CAM**

이렇게 Input-Image에 대해 분류 작업이 이루어졌지만 이 과정들은 신경망 안에서 블랙박스처럼 이루어지기 때문에 우리는 이 신경망 모델이 Image에서 어떤 feature들을 통해 분류 작업을 수행했는지 알 수 없다. 따라서 위의 GAP value를 통해 CAM(Class activation map) 을 도출하여 분류하는데 사용된 Image의 feature을 시각화 하고자한다.



**(그림4-a)**

GAP을 통해 생성된 value가 F^k라 하자. 이때 F^k 와 마지막 dense layer의 weight matrix를 곱함으로써 최종 노드 value값이 결정되고 이를 통해 분류를 수행한다.

이때 F^k는 각 Image channel의 (16x16) matrix에서 256(16x16)개 value들의 평균값을 구한 값이다.

이때 GAP를 수행함으로써 Image의 특징 정보를 갖고있던 (16x16) Matrix가 1개의 value로 수렴하기때문에 Image feature 정보를 잃게된다.

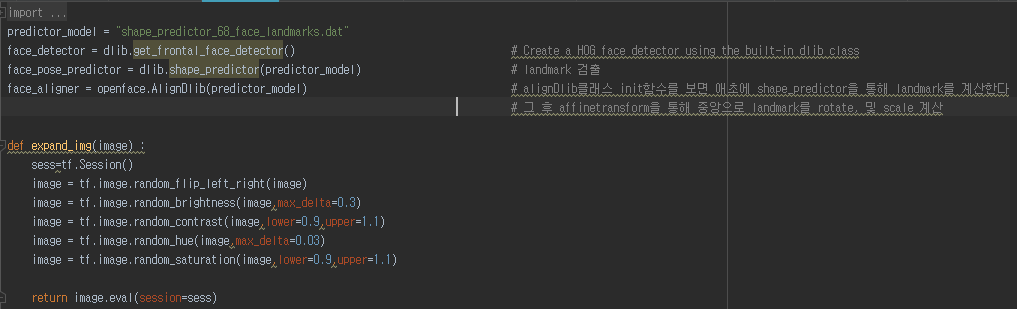
따라서 GAP를 수행하기전인 64개의 (16x16) Image Chaneel 에 weight matrix를 곱하여 최종적으로 분류할 때 이용된 Image feature 를 시각화 할 수 있다. (이 과정을 CAM이라 한다.)

**2. 오픈소스에 추가 작성한 부분 설명 및**

**완성된 프로그램 소스**

(1) Detection\_Cropping.py

* **expand\_img(image) 함수**



설명) 신경망 모델을 이용해서 딥러닝 학습을 할 때 가장 중요한건 image preprocessing이다.

복잡한 신경망 모델을 학습하기 위해서는 noise가 적은 image이여야 하며 대량의 data가 있어야 복잡하고 깊게 쌓여있는 신경만 layer들을 학습시킬 수가 있다.

이때 사용할 image로는 본인 혹은 주변 지인의 image가 필요하기 때문에 방대한 dataset을 구하기 힘들다. 따라서 expand\_img 함수를 통해 image를 random하게 밝기, 명암, pose 들을 바꿔가며 1장의 image에 대해 6장의 image를 생성하게끔 하여 작은 dataset을 극복하고자 이 함수를 정의하였다.

* **cropping\_img(img\_class) 함수**

**(오픈소스)**

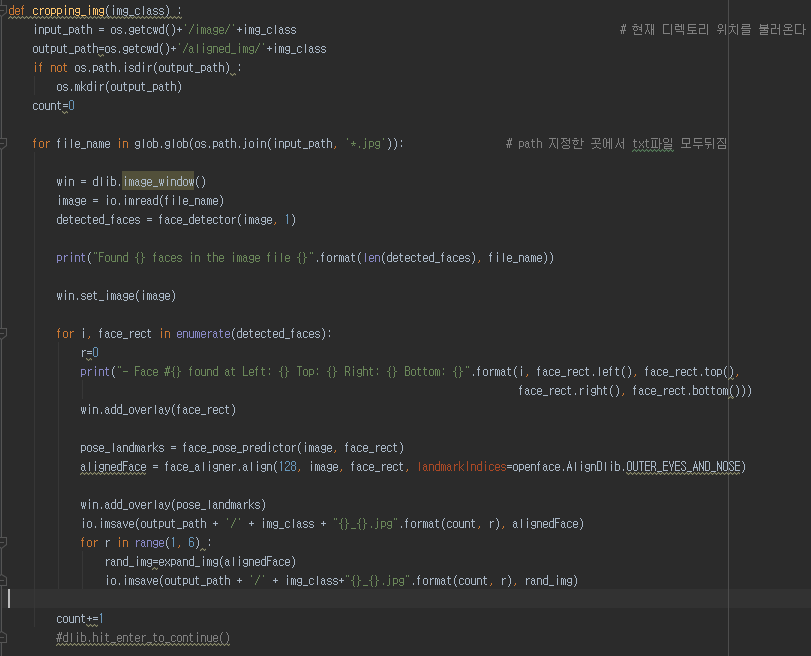


URL) <https://gist.github.com/ageitgey/82d0ea0fdb56dc93cb9b716e7ceb364b>

Image에서 Landmark Point들을 검출하고 Cropping하기 위해 이 오픈소스를 참고하였다.

이 코드에서는 주어진 input image에 대해 얼굴을 찾아내고 Openface 라이브러리에서 정의한 68개의 Template Landmark Point 정보를 이용하여 Landmark Point를 검출하고 Openface 라이브러리에 정의되어있는 AlignDlib 함수를 이용해 Image를 affine transform 및 cropping을 수행한다.

**(수정한 코드)**



설명) 먼저 위의 오픈소스에서 더 편리하게 대량의 Image를 읽고 저장하기 위해 경로를 위와 같이

현재 디렉토리 경로를 불러온 후 이 경로를 통한 상대적인 접근을 하도록 수정하였다.

또한 Landmark를 검출하여 Cropping하는 과정에 생길 error 및 noise Image를 검출해내기 위해 Cropping이 이루어질때마다 Landmark 검출과정을 시각화 하는 코드를 아래와 같이 추가하였다.

ᆞwin.set\_image(image) – Input Image 시각화

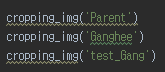
ᆞwin.add\_overlay(face\_rect) – Input Image에서 검출한 Landmark(face\_rect) 를 Input Image에 겹쳐서 보여준다.

기존에 Image를 구분하여 담아놓은 폴더이름을 인자로하여 이 함수를 호출하면 위 코드에서 설정해놓은 새로운 경로 (aligned\_img) 폴더에 Cropping 된 image가 새로 저장된다.

Ganghee - 신경망 모델을 학습하는데 사용되는 image(label=0)

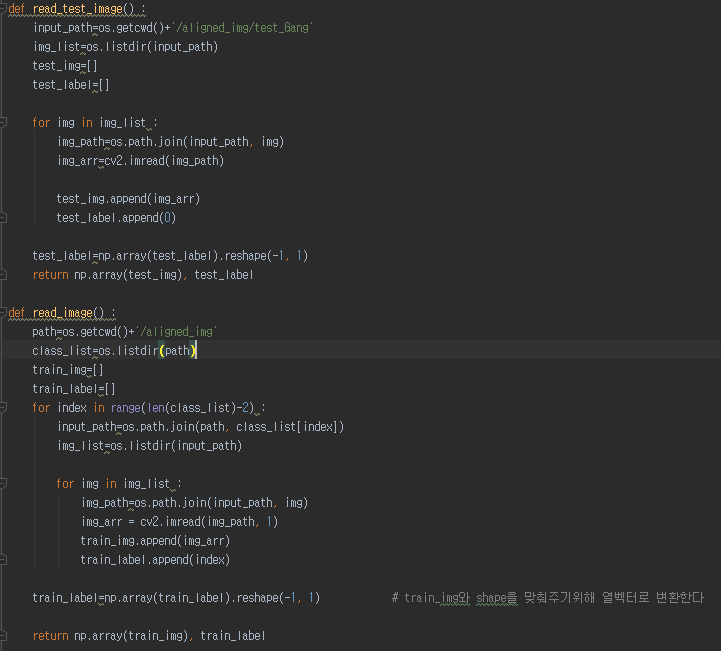
Parent – 신경망 모델을 학습하는데 사용되는 image(label=1)

test\_Gang – 신경망 모델을 테스트하는데 사용되는 image



(2) model\_real.py

* **read\_test\_image() / read\_image() 함수**



설명) (1)Detection\_Cropping.py 소스코드를 통해 Raw-Image를 Cropping하여 Aligned-Image를 새로 생성한 후 이 Aligned-Image를 load하기 위한 함수들이다.

신경망 모델을 학습 및 테스트하는데 서로 다른 Image를 이용하기 때문에 두 함수로 구분하였다.

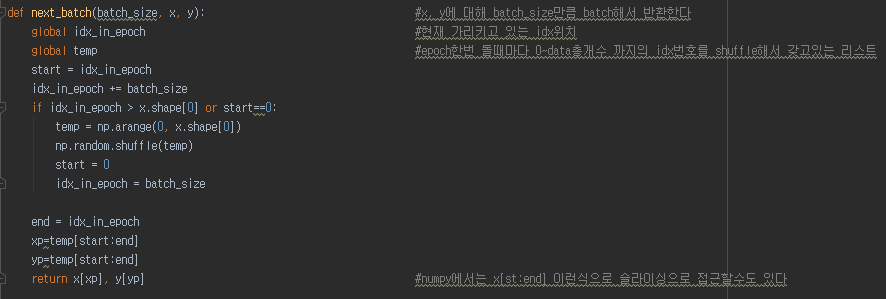
read\_image() – (1)Detection\_Cropping.py를 통해 Aligned-Imaged를 load하며 신경망 모델을 학습시킨다. 해당 Image의 label을 폴더의 index를 통해 구분하여 값을 지정해주었다.

(Ganghee & Parent 폴더를 인자로 넘겨준다)

read\_test\_image() – 신경망 모델을 학습시킨 후 테스트할 때 사용할 test용 Image를 load한다.

(test\_Gang 폴더를 인자로 넘겨준다)

* **net\_batch(batch\_size, x, y) 함수**



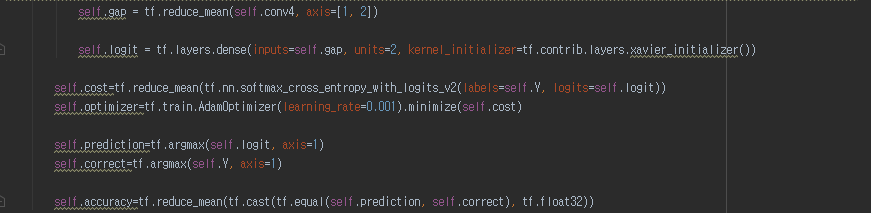
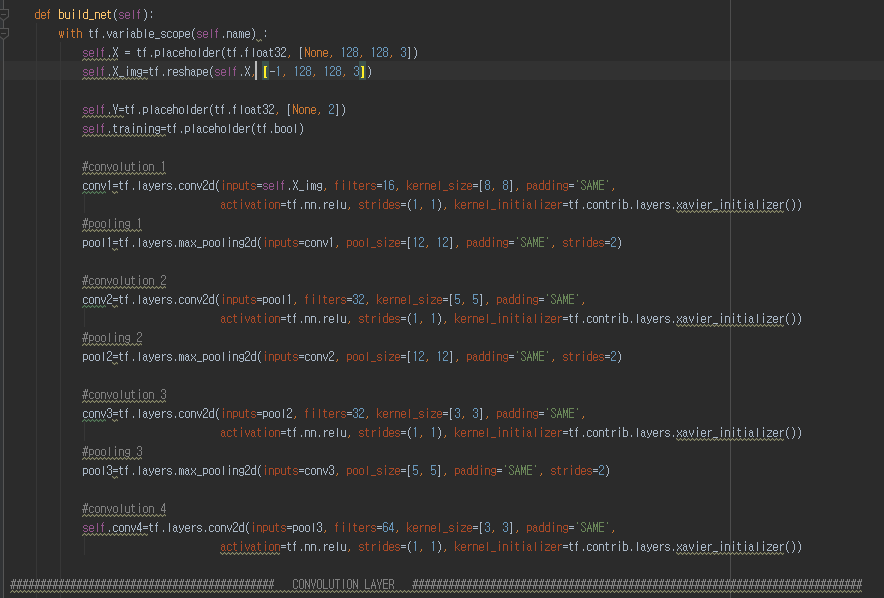
설명) 신경망 모델을 train 할때 1장의 input\_image 마다 신경망 가중치를 수정한다면 가중치가 불안정적으로 변하는 경향이있다. 이를 극복하기 위해 batch 학습을 이용하는데 1장이 아닌 여러장의 image-set를 한번에 input으로 넘겨주고 이 image-set에 적용되는 가중치 변화량을 평균낸 후 그 값을 이용하여 가중치를 수정한다. 이를 통해 좀 더 안정적으로 가중치를 수정하면서 신경망 모델을 학습시킬 수가 있다.

이 프로그램에서는 batch 학습을 위한 image-set size를 50으로 설정하였고 next\_batch 함수를 호출함으로써 학습하는데에 input으로 50장의 image를 사용할 수 있게끔한다.

**<Class Model>**

신경망 모델을 정의하고 학습 및 테스트, 그리고 CAM을 위한 함수들을 정의해놓은 함수이다.

* **build\_net() 함수**



설명) 신경망 모델을 정의한 함수이다.

처음에 설명한 바와 같이 오직 binary classification을 목적으로 하는 프로그램이기에 layer를 깊게 쌓을 필요가 없다 판단하였다. 따라서 Conv-Pool layer을 3단계로 쌓았고 GAP를 위해 Conv layer를 하나 더 쌓은후 최종적으로 GAP를 계산한 후 dense\_layer를 통해 0, 1로 분류한다.

(0 = Ganghee, 인식 / 1 = X Ganghee(Parent), 인식 실패)

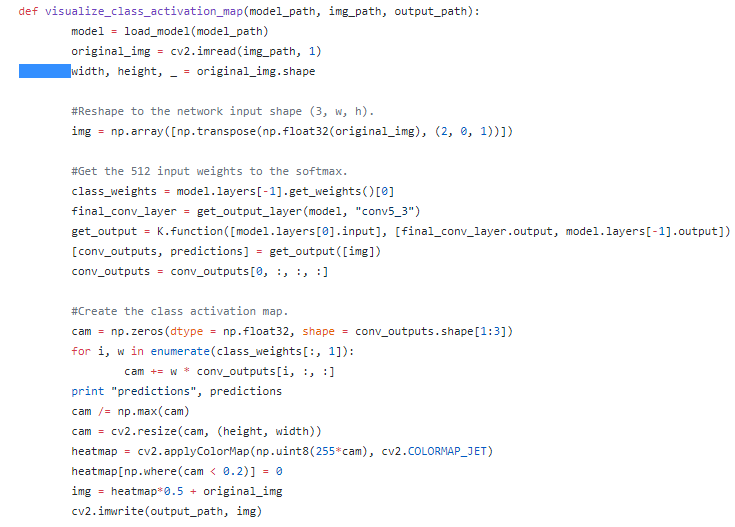
학습하는 과정에서 gradient 손실을 방지하기 위해 각 layer의 활성함수로 relu 함수를 선택하였으며

가중치를 수정하기위한 optimizer로는 momentum을 함께 사용하여 더욱 안정적으로 가중치를 수정하여 좋은 효율을 보이는 AdamOptimizer를 이용하였다.

또한 신경망 모델 학습에서 중요시 여기는 초기값 설정으로는 간단하지만 꽤 좋은 효율을 보이는 xavier\_initializer를 이용하였다.

* **visualize\_class\_activation\_map() 함수**

**(오픈소스)**



URL) <https://github.com/jacobgil/keras-cam/blob/master/cam.py>

위 오픈소스에서는 vgg16 모델을 신경망 모델을 선택하였다. 따라서 vgg16 class에 정의되어있는 get\_weights() – 해당 layer에 종속되어있는 가중치 matrix를 반환

get\_output\_layer(model, ‘name’) – ‘name’과 일치하는 layer의 output matrix를 반환

이 두 함수를 이용하여 마지막 Convolution 을 거친 후의 Image Channel들과 마지막 dense-layer에 해당하는 weight matrix를 반환받은 후 두 matrix를 곱하는 과정을 통해 CAM을 구현하였다.

그러나 이 프로그램에서는 직접 설계한 신경망 모델을 사용하고 있기 때문에 두 함수를 직접 구현해야했다.

**(수정한 코드)**



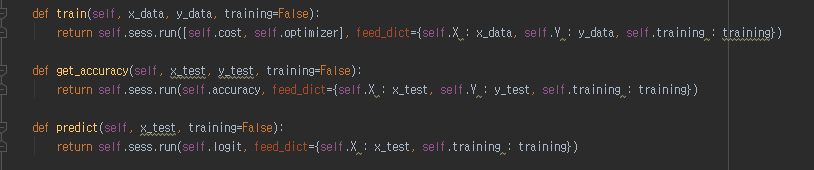
설명) 위의 두함수와 같은 기능을 수행하고자 따로 코드를 구현하였다.

tf.get\_default\_graph().get\_tensor\_by\_name(‘name’) – tensorflow 에서 제공하는 함수로써 직접 설계한 신경망 모델이 tensorflow로 구현되어있기 때문에 이 함수를 이용하여 ‘name’에 종속된 weight matrix를 반환받았다.

sess.run(conv4, …) – 마지막 Convolution 계층을 거친 후의 output matrix를 얻기 위해 직접 sess.run을 통해 output matrix를 반환받았다.

이후 오픈소스와 마찬가지로 두 matrix의 곱을 계산한 후 최종적으로 더해줌으로써 Image Feature 정보를 갖고있는 matrix을 계산하였다. 그리고 이 matrix를 기존의 Image size로 맞춰준 후 두 이미지를 겹침으로써 Image위에 feature들을 볼 수 있도록 시각화하였다.

* **train / get\_accuracy / predict 함수**

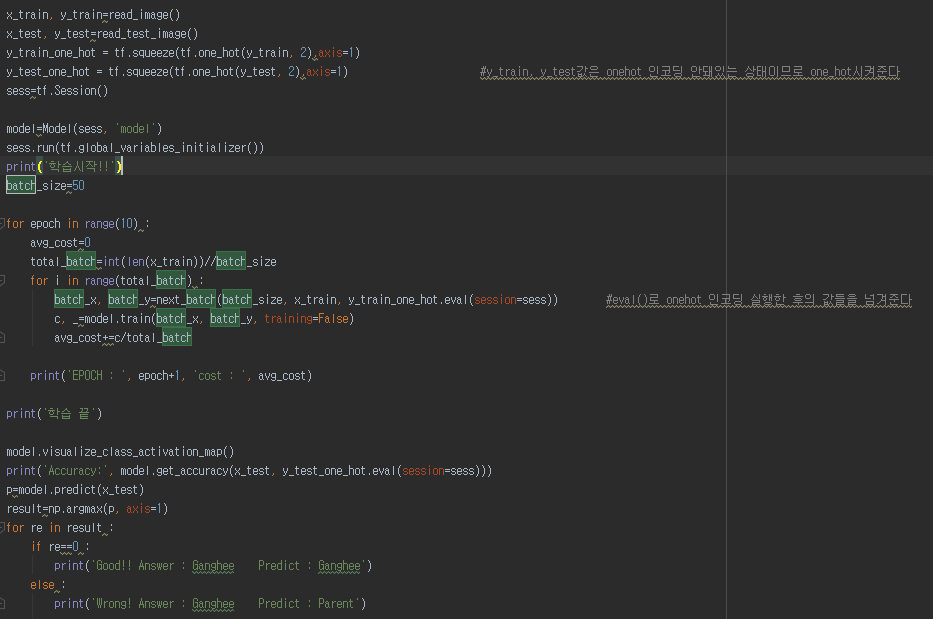


설명) train() - 신경망 모델에 Input-Image와 label를 인자로 하여 train 함수를 호출함으로써 신경망 모델을 학습시킨다.

get\_accuracy() – test-Image를 인자로 신경망 모델을 돌려봄으로써 정확도를 계산한다.

predict() – test-Image를 인자로 신경망 모델을 돌려봄으로써 모델이 도출한 결과값을 반환한다.

* **학습 / 테스트 수행코드**



설명) 신경만 모델을 생성한 후 aligned-image를 인자로 하여 모델을 학습시킨다.

epoch=10으로 하여 전체 image dataset을 10번 학습시키고 학습을 완료한 후에 CAM을 시각화하여 보여준다. 마지막으로 모델이 학습이 잘됐는지 임의의 10장의 사진을 동일인물로 판단하는지 확인하기 위해 accuracy와 predict 함수를 이용한다.

**3. 단계별 매뉴얼**

**(그림4-a)**

**(그림4-b) – image폴더 내부**

설명) 1. 위와같이 코드와 위 폴더 및 파일들을 같은 디렉토리에 위치시킨다.

2. (그림4-b) 와같이 image 폴더내에 Ganghee, Parent, test\_Gang 폴더를 위치시킨다.

3. 각 코드파일에서 import하고있는 라이브러리들을 설치한다.

- pip install scipy

- pip install openface

- pip install tensorflow

- pip install dlib

4. Openface 라이브러리에서 제공하는 Template Landmark Point 정보를 저장하고 있는

shape\_predictor\_68\_face\_landmarks.dat 파일을 같은 디렉토리에 위치시킨다.

5. Detection\_Cropping.py 를 컴파일 함으로써 image폴더에 있는 raw-image의 Croppoing결과가 aligned\_img폴더에 저장되게끔 한다.

6. model\_real.py를 컴파일 함으로써 학습 및 테스트를 진행한다.

7. model\_real.py 컴파일 결과로 cam\_img에 CAM 결과 image가 저장되게 된다. cam\_img내에 image들을 확인함으로써 CAM이 정상적으로 이루어졌는지 확인한다.

\* 직접 설계한 신경망 모델이기에 매번 컴파일할 때 마다 학습이 새롭게 이루어져서 컴파일하는데 약 1~2시간의 시간이 소요됩니다.

**4. 프로그램 결과 화면 및 데이터**

**[1]** 먼저 Detection\_Cropping.py를 실행하여 Image를 Cropping된 Aligned-Image로 변환하며 이 과정에서 expand\_img() 함수 호출로 인해 1장의 Image에 대해 각각 random하게 조명, 명암, 각도가 변환된 6장의 Aligned-Image가 aligned\_img 폴더에 저장된다. 또한 Cropping 과정에서 Landmark image가 화면에 출력되도록 하였다.

**(기존의 Image폴더)** – 각각 Ganghee / Parent / test\_Gang 폴더

Train을 위한 Ganghee Image로 168장의 Image를 이용했다.

Train을 위한 Parent Image로 59장의 Image을 이용했다.

최종적으로 test를 위한 test Image로 장소, 조명, 포즈를 다르게한 21장의 Image을 이용했다.

**(Detection\_Cropping.py 컴파일 후 aligned\_img폴더)**

**(aligned\_img내에 Ganghee폴더)**

얼굴의 눈코입을 기점으로 기존 Image를 Cropping 하였으며 위 사진들을 보면 동일 image에 대해 각각 각도, 조명, 밝기가 random 하게 바뀐 image 6장이 생성되었음을 확인할 수 있다.

168장의 Image가 936개의 Image로 확장되었다.

실질적으로 신경망 모델 label=0을 학습할 때 이용할 image들이다.

**(aligned\_img내에 Parent폴더)**

59장의 Image가 324장의 Image로 확장되었다. 신경망 모델 label=1을 학습하기 위한 Image이다.

**(aligned\_img내에 test\_Gang폴더)**

21장의 Image가 126장의 Image로 확장되었다. 신경망 모델을 최종적으로 test하기 위한 Image로 사용된다. 각각 조명, 포즈, 각도를 다르게 한 사진들이다.

(각기 다른 head pose에 대해 landmark를 검출하고 있는 Detection\_Croppoing.py의 결과화면이다.)

**[2]** 신경망을 학습한 후 test-image에 대해 결과를 보여주며 CAM 결과 image를 저장하는

model\_real.py를 컴파일한 결과화면이다.

**(test-image에 대한 정확도와 어떤사람인지 추측한 결과 predict를 보여주는 결과화면)**

결과화면을 보면 정확도 비율이 1.0으로써 test-image로 사용된 126장의 test용 Ganghee image에 대해 모두 같은 인물로 추측하였음을 보이고있다. 이때 항상 정확도가 1.0으로 나오는 것은 아니다.

대략 0.93~1.0을 보이고있는데 정확도가 1.0이 아닐때는 아래 결과화면에서

**Wrong! Answer : Ganghee Predict : Parent** 라 출력되는 문장이 다소 있음을 확인할 수 있다.

**(model\_real.py를 컴파일 한 후 cam\_img 폴더에 저장된 CAM image)**

위와같이 model\_real.py를 컴파일 하면 cam\_img 폴더에 각 test-image에 대해 feature을 추출하여 image위에 겹쳐 표시한 CAM image가 저장된다. 위 사진들을 보면 얼굴의 외곽이 아닌 얼굴내에 Colormap이 겹쳐져 표시됨을 볼 수 있는데 이는 신경망 모델이 분류할 때 추출한 feature을 image위에 겹쳐서 시각화한 것이다.

**5. 결론**

신경망 모델을 학습하기 위해 약 250장의 사진을 1500장의 사진으로 확장시켰다.

이는 신경망 layer을 여러게 쌓은 Deep Neural networks을 학습시킬 때 dataset이 충분히 크지 않으면 학습이 잘 되지 않기때문이다. 보통 오픈소스에서 흔히 있는 vgg16 / vgg19 모델 같은 경우 image에 대해 1000개의 분류를 수행해야하므로 dataset을 적게는 몇만장 많게는 몇십만장의 image를 이용한다. 그러나 이 프로그램에서 설계한 모델은 16, 19개의 layer을 쌓은 vgg16 / vgg19 모델과 달리 3개의 Conv-Pool layer을 쌓은 간단한 모델이기 때문에 1500장의 사진으로도 binary classfication을 어느정도 잘 수행함을 확인할 수 있다.

이렇게 binary classification에 한해서 좋은 성능을 보이고 있는 모델이지만 CAM에서 기대한만큼의 CAM-image를 보여주지는 않는다. 신경망을 이용해서 Face classification을 수행할 때 신경망은 image의 얼굴정보나 눈코입 정보를 통해 모델을 학습하고 예측하는 것이 아닌 오직 pixel의 value 정보만을 가지고 학습하고 예측하는 것이기 때문이다. 따라서 image를 충분히 Cropping하여 얼굴만 보이도록 image를 preprocessing 하였음에도 모델이 feature을 추출할 때 머리카락 색깔, 배경 색깔에 영향을 받아 이들을 feature로 선택하기도 한다. 이 모델을 더욱 확장시켜서 layer을 깊게쌓고 더 방대한 dataset을 이용하여 학습시킨다면 더욱 섬세하게 feature을 추출하기 때문에 이와 같은 noise와 error들을 줄여서 눈코입의 pixel value만을 기준으로 face classification을 할 수 있을거라 예상한다.