

졸업작품 중간 보고서

얼굴 인식 및 영상 추적 분석

최종원 | 종합설계2 | 2016-10-21

Contents

[Abstract 1](#_Toc464959438)

[Goal 2](#_Toc464959439)

[Development Environment 2](#_Toc464959440)

[Amazon Web Services(AWS) 2](#_Toc464959441)

[Architecture 4](#_Toc464959442)

[Install 5](#_Toc464959443)

[CUDA 5](#_Toc464959444)

[OpenCV 7](#_Toc464959445)

[TensorFlow 9](#_Toc464959446)

[WAS 11](#_Toc464959447)

[Flask 11](#_Toc464959448)

[uWSGI 13](#_Toc464959449)

[Nginx 14](#_Toc464959450)

[Current Progress 16](#_Toc464959451)

[Main Page 16](#_Toc464959452)

[Image Upload 16](#_Toc464959453)

[Video Upload 17](#_Toc464959454)

[Details 18](#_Toc464959455)

[Frontend 18](#_Toc464959456)

[app.py 18](#_Toc464959457)

[stream.py 21](#_Toc464959458)

[Backend 23](#_Toc464959459)

[Overall 23](#_Toc464959460)

[OpenCV Frame 24](#_Toc464959461)

[Haar-like feature(deprecated) 25](#_Toc464959462)

[Optical Flow 27](#_Toc464959463)

[Histogram 30](#_Toc464959464)

[Summary 33](#_Toc464959465)

[Architecture 33](#_Toc464959466)

[Site Map 33](#_Toc464959467)

[Project Structure 34](#_Toc464959468)

[UML Cases 34](#_Toc464959469)

[Class Diagram 35](#_Toc464959470)

[Next Step 36](#_Toc464959471)

[Neural Network 36](#_Toc464959472)

[Schedule 37](#_Toc464959473)

[Reference 38](#_Toc464959474)

# Abstract

IT 기술이 얼마나 발전할 수 있을까.

컴퓨터가 발명이 되면서 인간의 불편함을 해소해 주는 도구의 하나가 되었고, 사람의 일을 하나하나 기계가 대체하기 시작하면서 궁극적으로 사람의 일을 기계가 하는 날이 얼마 남지 않은 것 같다. 그래서 스스로 판단을 할 수 있게 되어 기계와 인류가 대립하는 영화도 나오곤 했다.

이세돌과 알파고의 바둑 대결을 하면서 머신러닝이라는 기술이 떠오르게 되었다.

컴퓨터에 이미 주어졌던 결과를 이용하여 Input 명령을 내려 Output이라는 원하는 결과를 내 놓는 자동화 방식이 아니라, 비슷한 결과를 Input한다. 즉 Input을 경험으로 확대하여 기계가 학습과 추측을 할 수 있게 되었다.

알파고와 이세돌의 대국에서 알파고는 이세돌이 경기 했던 데이터를 바탕으로 하지 않았다. 단지 프로들이 했던 기본적인 기보를 스스로 학습하여 이세돌과의 대결에서 수를 추측한 것이다.

프로젝트의 주제를 고민하던 중 영상 처리 기술은 기계가 할 수 있지만 영상을 분석하는 것은 아직 사람이 할 수 밖에 없다는 것을 알게 되었다.

예를 들어 컴퓨터는 틀린 그림 찾기는 할 수 있지만 숨은 그림 찾기는 할 수 없다. 할 수 있다고 해도 미리 사람이 결과랑 동일한 결과를 먼저 찾아 코딩을 해야 할 것이다.

그러므로 머신러닝 기술을 영상에 적용하기 시작하면 영상을 스스로 판단하기 때문에 편집 등을 해야 할 경우의 생산성이 향상 됨을 기대해 볼 수도 있다.

* 많은 CCTV를 관할하는 통제 구역의 경우 설치한 모든 곳을 한꺼번에 보고 찾아내기란 쉽지 않다.
* 군중 속에서 특정한 사람을 정확히 찾아내는 일은 결국 사람이 해야한다.
* 영상 처리 시스템에서 머신러닝을 이용하여 사람의 얼굴을 스스로 학습해 추적할 수 있는 프로젝트를 계획하게 되었다.

## Goal

실시간으로 로컬 카메라를 이용한 OpenCV 영상처리 기술은 이미 많이 개발이 되어 있고, 훌륭한 API를 오픈소스로 제공한다.

서비스 형태로 제공하기 위해 Flask를 이용한 영상처리 웹페이지를 통해 영상을 업로드 하는 인터페이스를 제공한다.

영상을 업로드할 경우 영상처리를 한 다음 인코딩을 하여 새로운 영상을 생성하는 방법이 있다.

하지만 시간이 오래 걸리므로 업로드한 영상을 프레임별로 실시간 스트리밍을 통해 다시 클라이언트에게 전송하는 시스템을 택했다.

# Development Environment

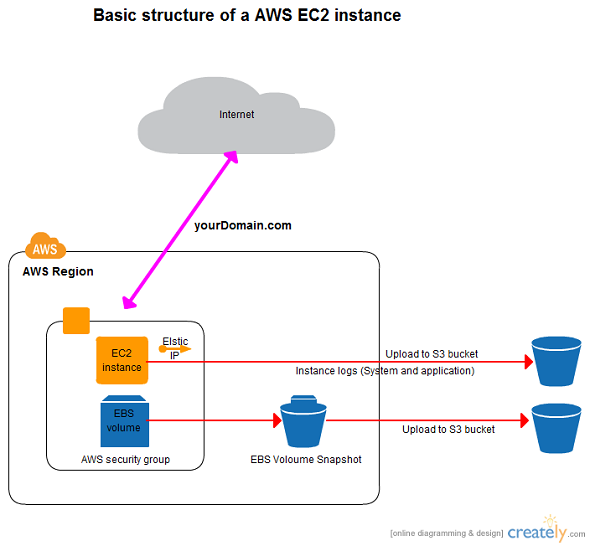
## Amazon Web Services(AWS)

여러 클라우드 컴퓨팅 서비스를 제공하는 웹 서비스 인프라이다.

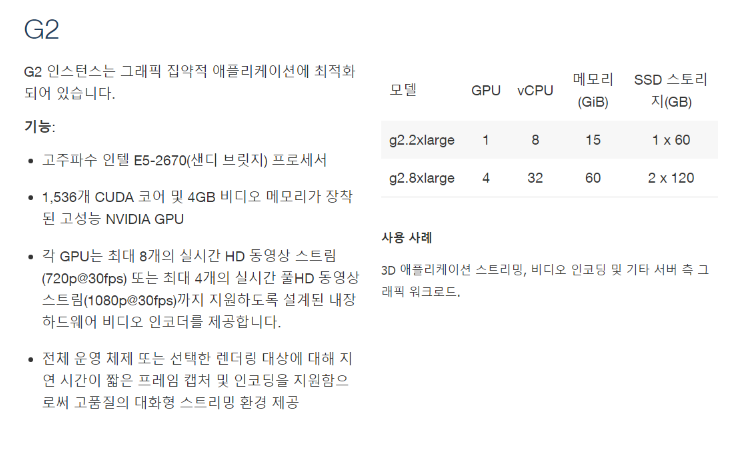
간단하게 자신의 서버를 생성할 수 있고 사용한 만큼 비용을 지불하게 된다.

EC2(Elastic Compute Cloud) 기술을 이용하여 서버로 사용할 컴퓨터가 아마존 인프라 위에 만들어진다. SSH 원격 접속을 이용하여 제어할 수 있다.

이 프로젝트에서는 웹페이지를 어디서나 액세스 할 수 있도록 AWS(Amazon Web Service) 클라우드 환경을 이용하였다.



* 머신러닝과 영상 처리를 하기에 적합한 인스턴스인 g2.2xlarge 인스턴스[[1]](#footnote-1)를 이용



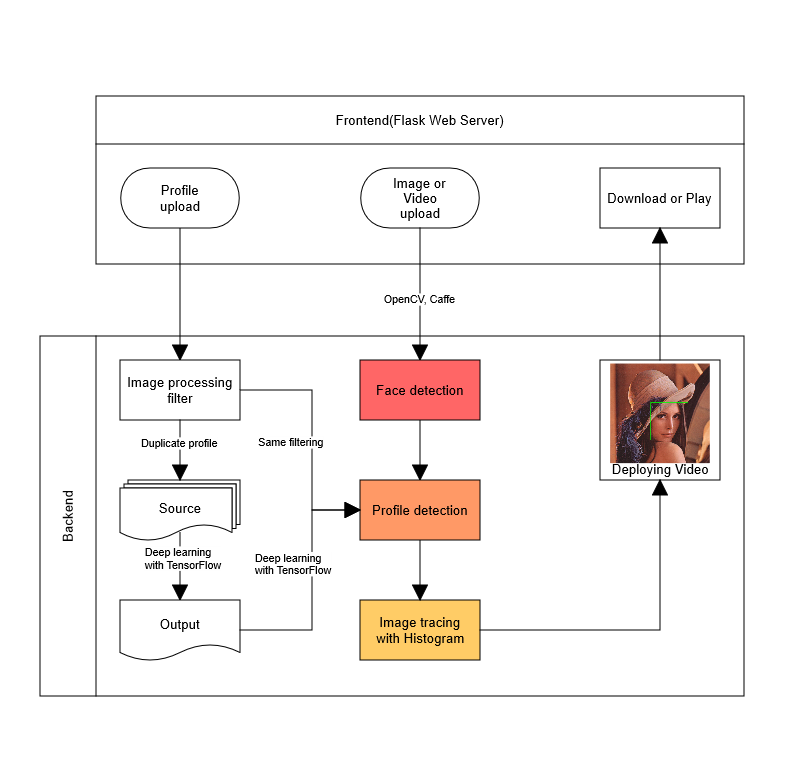
* 테스트는 t2.micro 인스턴스[[2]](#footnote-2)를 이용
* Ubuntu 14.04.3 LTS 운영체제에 설치한 핵심 오픈소스 라이브러리들

CUDA 7.5, cuDNN 5.1

OpenCV 3

TensorFlow r0.11

## Architecture



AWS 내부의 구조는 위 그림과 같다.

클라이언트가 웹 서버로 이미지나 영상을 업로드하게 되면 서버에서 OpenCV오픈 소스 라이브러리로 얼굴 인식 및 추적을 시작한다.

비디오를 업로드 했을 경우 바로 실시간 얼굴 인식을 시작하게 되며 추적할 이미지를 따로 업로드 하게되면 TensorFlow를 이용해 영상과 유사성을 판별하여 추적을 실시한다.

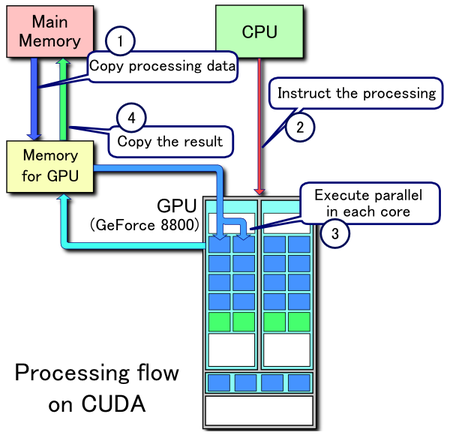
# Install

AWS 클라우드 환경에서 제공하는 인스턴스는 Ubuntu만 설치된 상태이다.

라이브러리와 패키지들은 따로 설치를 해주어야 한다.

## CUDA

**CUDA** (Compute Unified Device Architecture)는 NVIDIA가 개발한 GPGPU 기술이다.

[[3]](#footnote-3)

**CUDA 처리 흐름의 예**  
1. 메인 메모리를 GPU 메모리로 복사  
2. CPU가 GPU에 프로세스를 지시함  
3. GPU각 각 코어에 병렬 수행  
4. GPU 메모리로부터의 결과물을 메인 메모리에 복사

* CUDA 7.5 설치



* cuDNN 5.1 설치

cuDNN 5.1은 압축 파일 형태로 제공하며 CUDA 설치시에 생긴 /usr/local/cuda 디렉터리 아래에 복사한다.

* CUDA 환경변수 설정



## OpenCV

**OpenCV**(Open Computer Vision)은 오픈 소스 컴퓨터 비전 C 라이브러리이다.

**윈도우 리눅스** 등 여러 플랫폼에서 사용이 가능하고, 실시간 이미지 프로세싱에 중점을 두었다.

본 프로젝트에서는 Object Tracking을 위해 사용하였다.

* Developer tool 설치



* 이미지 로드 라이브러리 설치



* GTK 설치

AWS EC2로 원격 접속을 하기 때문에 이미지나 영상을 보려면 X window를 통해 이미지를 볼 수 있다. 그러기 위해 필요한 라이브러리이다.



* 비디오 프로세싱 라이브러리



* pip 패키지 설치



* OpenCV Project Clone



* OpenCV 빌드



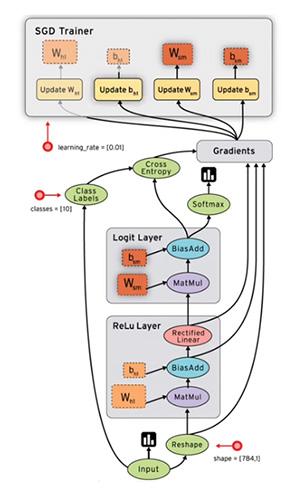
빌드가 완료되면 OpenCV 설치가 완료된 것이다.

## TensorFlow

**텐서플로우(TensorFlow)**는 구글 제품에 사용되는 머신러닝(기계학습)을 위한 오픈소스 라이브러리이다.

모바일 환경 및 64비트 리눅스, OS X 데스크톱이나 서버 시스템의 CPU와 GPU를 사용할 수 있다. 윈도우 환경에서는 Docker나 가상 머신을 통해 사용 가능하다.

텐서플로우 연산은 상태를 가지는 데이터 흐름의 **그래프 연산**이 사용된다. 이 연산을 이용하여 Convolution Neural Network(컨볼루전 신경망)을 구현하여 컴퓨터가 “스스로 판단” 할 수 있게 돕는다.



OpenCV에서도 마찬가지였지만 Python2.x 버전으로 통일한 상태이기 때문에 따로Virtualenv를 설정해 주지 않는다.



## WAS

업로드한 영상 또는 이미지를 처리해주는 웹 서버가 필요하다. 아래는 Http요청을 Nginx와 Gunicorn을 통해 플라스크로 향하는 개략적인 그림이다.

[[4]](#footnote-4)

이 프로젝트에서는 Gunicorn대신 uWSGI애플리케이션을 사용했다.

### Flask

Python에서 웹 서비스가 가능하도록 만들어주는 프레임워크.

OpenCV와 TensorFlow가 Python API를 제공하기 때문에 하나의 프로젝트에서 Import하기 위해서 선택했다.

* Install



* app.py ( 기본 틀 )



* Structure

아래 구조는 이 프로젝트 전체적인 구조와 일치한다.



### uWSGI

Python WSGI 서버 중 한가지이다.

WSGI는 Web Service Gateway Interface의 약자로 Python 스크립트가 웹 서버와 통신하기 위한 WSGI의 미들웨어이다.

소켓을 열어 Flask 프레임워크와 Nginx 웹 서버의 중간 역할을 한다.

Flask 프레임워크에서 app.py를 모듈로 하여 서비스를 한다.

* Install



* Configure



### Nginx

더 적은 자원으로 더 빠르게 데이터를 서비스 할 수 있는 차세대 웹 서버.

이미지 프로세싱과 머신 러닝이 높은 성능을 요구하기에 개발 목적에 부합하는 웹서버를 선택했다.[[5]](#footnote-5)

따로 웹 서버 없이 uWSGI만으로도 서비스가 가능하지만 Nginx가 가진 향상된 Static content 핸들링을 통해 부하를 조금이라도 줄여줄 수 있어서 선택하게 되었다.

* Install



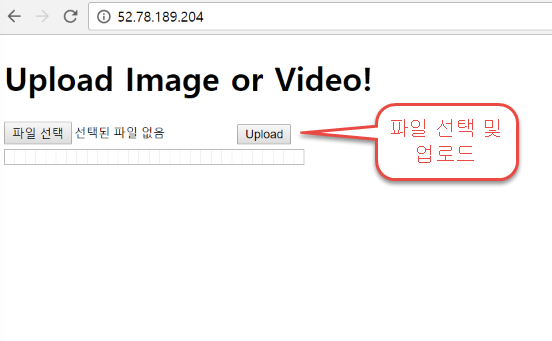
* Configure



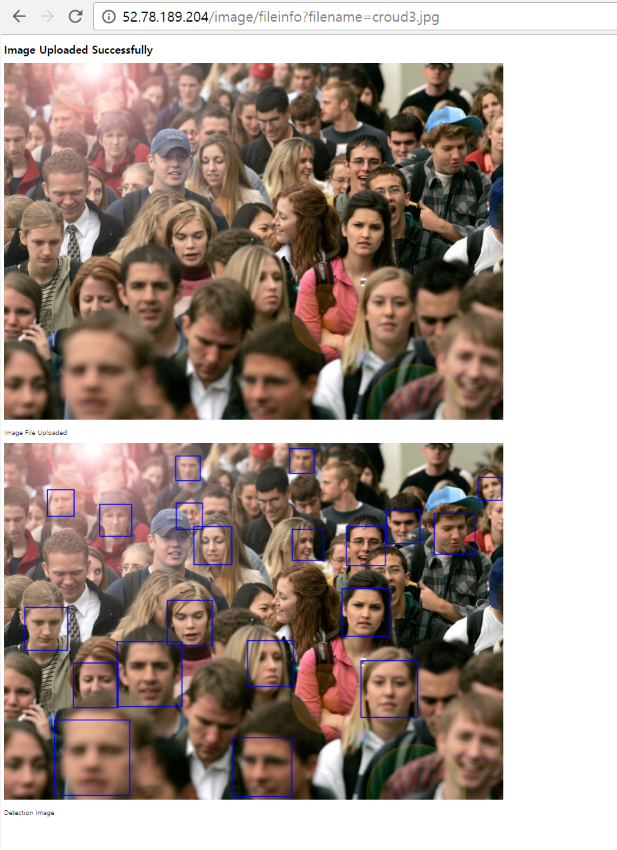
* 80번 포트를 열고 클라이언트가 업로드 할 수 있는 제한을 2G로 두었다.
* uWSGI의 소켓을 참조하여 프로젝트 루트 디렉터리를 설정했다.
* Static경로를 Nginx에서 핸들링하여 웹 서버의 부하를 줄였다.

# Current Progress[[6]](#footnote-6)

## Main Page



## Image Upload



## Video Upload[[7]](#footnote-7)



* 위쪽 영상이 업로드한 비디오이며 아래쪽 영상은 얼굴인식 및 추적을 하면서 스트리밍하는 프레임.
* 아래쪽 파일 업로드는 스트리밍하는 프레임에서 특정 얼굴을 추적하기 위한 쿼리 이미지이며 아직 기능 구현은 되지 않음.
* 스트리밍하는 영상은 되감기를 할 수 없다는 것이 단점.

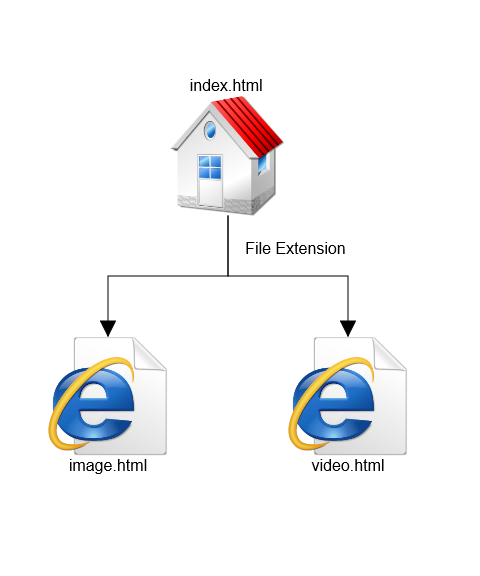
# Details

모든 소스코드는 <https://github.com/lastone9182/flask>에 있다.

## Frontend[[8]](#footnote-8)

* 루트 디렉터리인 flask는 절대경로 /var/www/flask에 위치하고 있으며 Nginx와 uWSGI를 통해 서비스 한다.
* uWSGI 게이트웨이 인터페이스는 app.py 를 모듈로 메인 페이지를 구성한다.
* app.py는 stream.py와 infofile.py, detection.py를 import 하고 있다.

### app.py



내용을 아주 개략적으로 나타내면 위 그림과 같다.

메인 페이지를 index.html라고 했는데 템플릿에 있는 upload.html을 가리킨다.

여기서 어떤 파일을 업로드 했는지에 따라 각자 다른 곳으로 Redirection 하게되는데 그 기준은 파일 확장자로 구분한다.

파일을 업로드하면 절대경로 /var/www/flask/static/uploads로 업로드 된다.

Nginx configuration에서 클라이언트 최대 업로드 용량을 2G로 지정했으므로 글로벌 변수로 선언한다.

Python에서는 글로벌 동적 변수를 지정하기 위해 변수 앞에 global이라고 선언을 해야하지만 Flask에서는 글로벌 동적 변수로 app.config 딕셔너리를 지원한다.

**app.py 일부 ( Server side )**



upload.html에서 업로드 버튼을 누르게 되면 POST method로 웹 서버에 요청하게 된다.

웹 서버는 업로드한 파일을 서버에 저장한 후 GET method를 통해 자기 자신 URL로 Redirection한다. 이때 글로벌 동적 변수에 저장해둔 파일명으로 확장자를 분석하여 image.html 또는 video.html 템플릿을 로드할지 결정하게 된다.

**templates/upload.html 일부( Client side )**

Flask에서는 Jinja2 템플릿 엔진을 이용한다.



**app.py 일부 ( Server side )**

****

Redirection한 곳에서 image.html 또는 video.html 템플릿을 로드하게된다.

### stream.py

영상의 경우 Server side에서 Response를 나누어 전송하는 스트리밍 기법을 적용한다.

업로드한 영상을 처리 한 후 인코딩을 통해 새로운 영상을 만들어서 클라이언트에게 video 태그를 통해 전송하는 방법은 너무 느리며 매우 큰 Response가 발생한다.

영상을 프레임별로 처리하는 동시에 JPEG로 변환하고 mimetype을 ‘multipart/x-mixed-replace’로 하여 실시간으로 Response를 전송한다.

여기서 OpenCV 코드는 생략하였다.

**stream.py**



**app.py 일부 ( Server side ) stream.py class import**





위와 같이 yield를 이용해 리턴을 하게 되면 프레임이 종료될 때 까지 아래와 같이 multipart response가 발생하게 된다.

**HTTP Response Stream Structure**

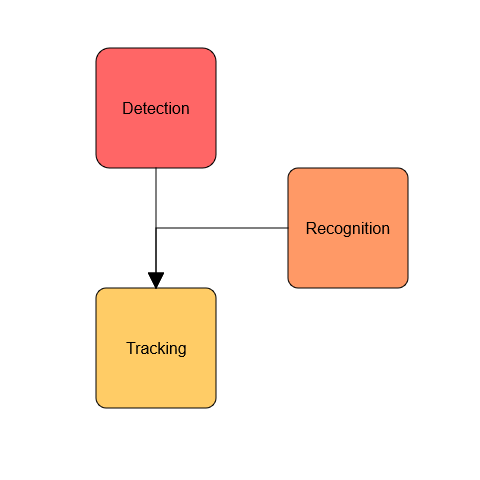


## Backend

* 테스트 소스코드가 facetracking.py에 구현되어 있다.

### Overall

프로젝트의 전체적인 과정을 간단하게 나타내면 아래 그림과 같다.



* Detection

먼저 영상에서 대상을 먼저 찾아야 한다. 현재까지 얼굴 인식을 Haar-like feature 알고리즘을 이용했으며 추후 TensorFlow를 이용한 머신 러닝을 이용하여 학습된 이미지를 바탕으로 Detection을 대체할 예정이다.

* Recognition

Detection된 얼굴 또는 오브젝트를 Database에 추가한 후 다음 프레임에서 다시 Detection이 일어났을 경우 그 찾아진 얼굴이 Database에 등록된 얼굴인지 식별(Recognition)하는 과정을 나타낸다.

Tensorflow를 이용한 Detection이 완료되면 차후 구현할 예정이다.

* Tracking

Detection이 하나의 영상에서 대상을 찾는 것이라면 Tracking은 영상에서 특정 대상의 위치 변화를 추적하는 것이다.

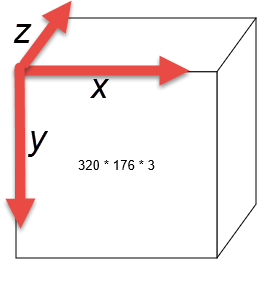
이 프로젝트에서는 Optical Flow와 Histogram 방식을 사용했다.

### OpenCV Frame

OpenCV에서 하나의 프레임만 읽어 오는 코드이다.



위와 같이 cv2.VideoCapture().read()를 호출 하면 하나의 프레임을 불러오는데 이 기본적인 하나의 프레임 구조는 다음과 같다.



( X, Y )픽셀이 ( 320, 176 ) 이라고 가정했을 때 하나의 프레임을 나타낸 것이다.

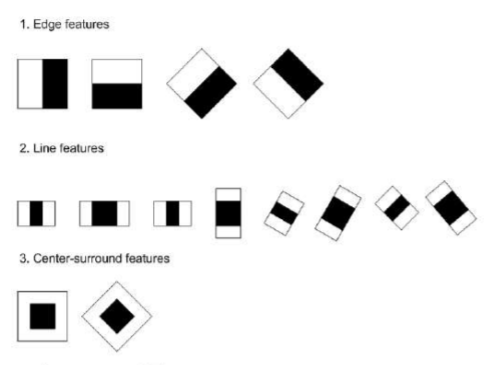
Z축은 빨강, 초록, 파랑을 나타내며 BGR 순서로 나타낸다. 실제로 frame의 shape값은

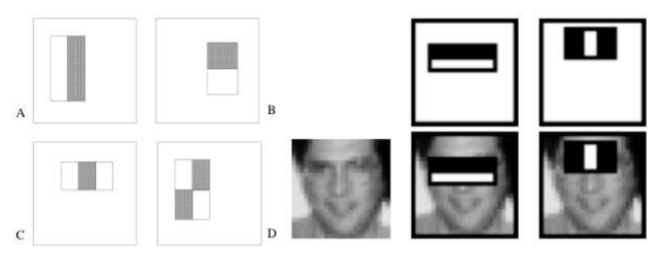
( 176, 320, 3 ) **( row[y], col[x], BGR[3] )** 매트릭스 형태로 나타난다.

색상 프레임을 Gray scale로 변경할 경우 shape값은 ( 176, 320, 3) 에서 ( 176, 320 )값이 된다.

### Haar-like feature(deprecated[[9]](#footnote-9))

얼굴의 특징을 찾는 기법중의 하나이다. 이미지를 Haar라는 특수 기저로 변환한 다음 아래 그림과 같은 사각형 흑백 영역에 대한 픽셀값의 평균 차에 의한 임계치 구분에 의해 특징을 판단한다.





다른 방법보다 연산이 간단하여 빠른 얼굴 검출에 적합하고 정면 얼굴 이미지에선 강력하고 빠른 알고리즘이지만 영상이 특정한 각도로 회전이 되거나 대조변화, 광원 밝기 등 이미지에 변형이 일어나게 되면 검출이 어려워지는 단점이 있다.

또한 Haar Feature를 이용한 영상에서는 영역 사이의 밝기 차이를 이용한 것으로 얼굴이 아닌데도 인식을 간혹 하는 오류가 있을 수 있다.

현재 얼굴 추적을 위해 Detection 알고리즘이 필요하여 선택하였다.

facetracking.py 일부



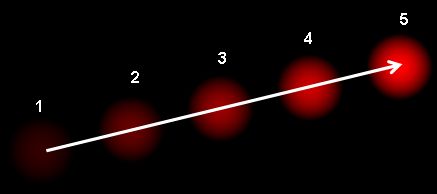
기본적으로 분류가 된 xml파일을 사용하며 프레임을 Grayscale로 변환한 다음 흑백 프레임의 밝기 차와 분류한 내용을 대조하여 얼굴을 검출하게 된다.

검출한 얼굴은 사각형 영역값을 리턴하며 파란색으로 프레임 위에 더했다.

### Optical Flow

Optical Flow란 개체 또는 카메라의 이동에 의한 두 연속된 프레임 사이의 움직임의 패턴을 벡터로 나타낸 것이다.

#### Theory



위 그림은 5개의 연속 프레임으로 움직이는 것을 나타내며 화살표는 변위 벡터로 나타낸다.

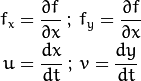
프레임을 I라 하고 시간에 따라 변하는 프레임이므로 각 픽셀에 대해 아주 짧은 시간에 대한 식은 아래와 같다.

http://docs.opencv.org/3.0-beta/_images/math/ae0a1708bb3b09848aab506592be34a7217f3306.png

이전 프레임에서 델타값을 더한 값이 다음 프레임의 위치 및 시간과 같다는 식이다.

우변을 테일러 근사하면

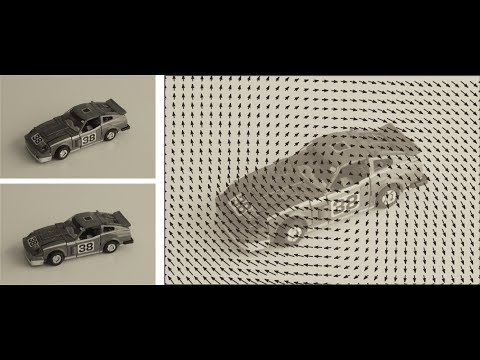
위 식을 얻고 양 변이 소거가 되어 아래와 같이 변수를 치환하게 되면



http://docs.opencv.org/3.0-beta/_images/math/0583b3af8a557dfd03d768cc97ca7bb76673b368.png

위 식을 Optical Flow Equation이라고 한다.

#### Category



이 이미지는 Optical Flow의 한 예로 시계방향으로 회전한 프레임을 위 식에서 도출된 (u, v) 벡터장으로 나타낸 것이다.

Optical Flow의 종류에는 Lucas-kanade Algorithm과 Farneback Algorithm이 있다.

* Lucas-kanade Algorithm은 point algorithm으로 여러 스케일 이미지 피라미드로 나누는 과정을 통해 모서리 값을 먼저 찾은 다음 이 점이 연속된 프레임에서 얼만 큼 이동했는지 flow를 보여주는 알고리즘이다.
* Farneback Algorithm은 dense algorithm으로 여러 스케일 이미지 피라미드로 나누는 과정을 통해 두 프레임간의 전체적인 flow를 Polynomial Expansion Transform을 통해 화면 전체를 일정한 간격으로 나누어 모션 벡터를 보여준다.

이 프로젝트에서는 주어진 영역 전체에 대한 추적을 해야 하므로 Farneback Algorithm을 선택했다.

facetracking.py 일부



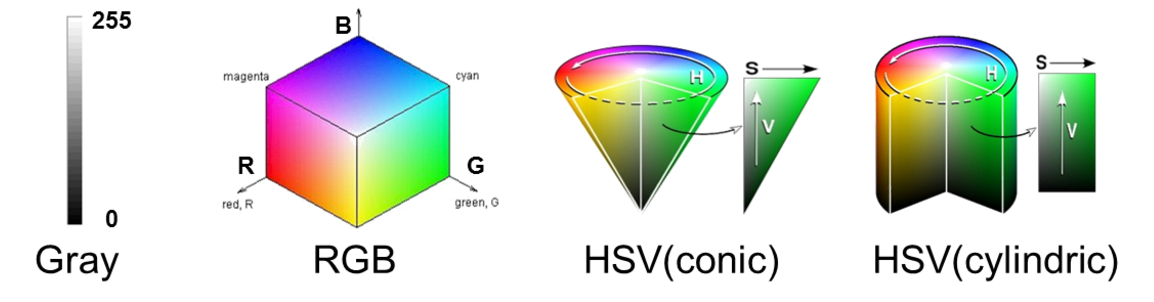
cv2.calcOpticalFlowFarneback[[10]](#footnote-10)이란 이름으로 OpenCV에서 API를 제공한다.

리턴된 flow값은 프레임 각 픽셀 값에 대한 벡터를 리턴하므로 영역안의 벡터를 모두 더한 값을 주어진 영역으로 나눈 값을 이전 프레임의 좌표에서 이동 시키는 방식을 적용했다.

( row, col ) 순이기 때문에 ( x, y )가 뒤바뀐 상태이고 flow 매트릭스의 특정 영역에 대한 벡터를 추출하는 방법은 Numpy를 사용했다.

### Histogram

#### Color Model[[11]](#footnote-11)

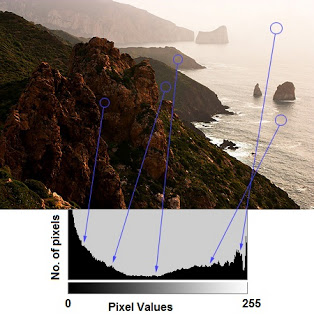


Gray 모델은 색(color) 정보를 사용하지 않고 밝기 정보만으로 영상을 표현하는 것이다. 검정색 0부터 흰색 255까지 총 256단계의 밝기값(intensity)으로 영상 픽셀값을 표현한다.

RGB 모델은 가장 기본적인 색상모델로서 색(color)을 Red, Green, Blue의 3가지 성분의 조합으로 생각하는 것이다. R, G, B 각각은 0 ~ 255 사이의 값을 가질 수 있기 때문에 RGB 색상 모델을 사용하면 총 256\*256\*256 = 16,777,216가지의 색을 표현할 수 있다.

HSV 모델은 Hue(색조), Saturation(채도), Value(명도)의 3가지 성분으로 색을 표현한다. Hue는 색조를, Saturation은 그 색이 얼마나 선명한 색인지, Value는 밝기(Intensity)를 나타낸다. HSV 모델은 색을 가장 직관적으로 표현할 수 있는 모델이며 또한 머리속에서 상상하는 색을 가장 쉽게 만들어낼 수 있는 모델이다. 영상처리/영상인식에서 HSV 모델을 사용할 때, H, S, V 각각은 0 ~ 255 사이의 값으로 표현된다. H 값은 색의 종류를 나타내기 때문에 크기는 의미가 없으며 단순한 인덱스(index)를 나타낸다. S 값은 0이면 무채색(gray 색), 255면 가장 선명한 색임을 나타낸다. V 값은 작을수록 어둡고 클수록 밝은 색임을 나타낸다. HSV 색상 모델은 그림과 같이 원뿔(conic) 형태, 원기둥(cylindric) 형태가 있다.

#### Theory[[12]](#footnote-12)



영상처리에서의 Histogram은 위 이미지에 대한 그래프와 같이 ( X, Y ) 이미지의 픽셀 수에 대응하는 화소 값에 대한 그래프이다. 총 픽셀 수를 이라 하고 각 픽셀에 대한 적절한 색상 모델에 대응하는 값을 라 하면 아래 조건을 만족하는 것이다.

#### Calculate

이 프로젝트에서는 주어진 두 영역에 대한 HSV 색상 모델에서의 Histogram 계수를 계산했다. 흑백 이미지보다는 영상의 색상 인덱스를 매트릭스로 나타내는게 영상 추적에서는 정확도가 높았다.

두 Histogram을 H1, H2라고 하고 프레임을 I라고 할 때 비교 계수를 구하는 공식은 아래와 같다.

http://docs.opencv.org/3.0-beta/_images/math/a97f7b5144b192bd40b04c165665c9eb1f5cceef.png http://docs.opencv.org/3.0-beta/_images/math/438152c292378fc2e75dfa048289aaefe29756d2.png

여기서 N은 영역 내의 픽셀 전체의 수이다.

facetracking.py 일부



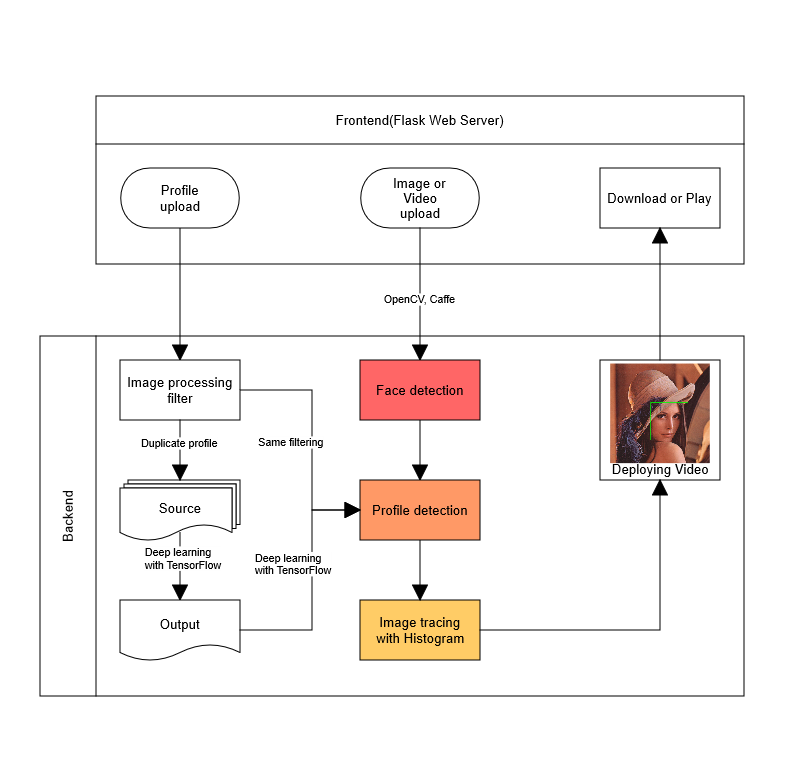
Histogram에 대한 calcHist[[13]](#footnote-13)와 compareHist[[14]](#footnote-14)라는 이름의 OpenCV의 API가 제공된다.

먼저 RGB 색상 모델을 이전 프레임과 현재 프레임에 대하여 HSV모델로 변경한다.

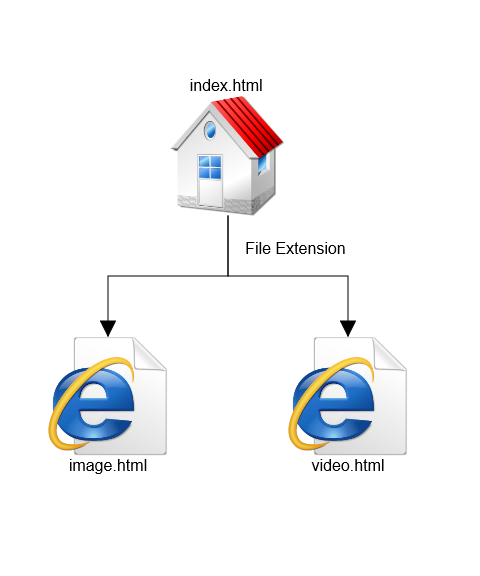
그 후 이전 프레임에서의 Optical Flow로 이동한 추적 영역에 대한 Histogram과 현재 프레임에서의 추적 영역에 대한 Histogram을 위 관계식으로 계산하여 나온 계수가 0.8 이상일 경우 80% 이상의 유사성을 나타내므로 그 이하의 계수가 나올 경우는 추적 영역에서 제외하는 코드이다.

# Summary

## Architecture



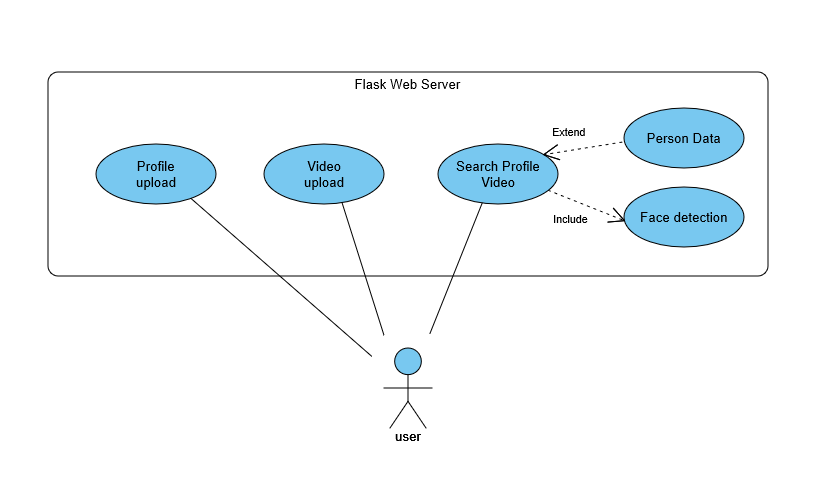
## Site Map



## Project Structure



## UML Cases

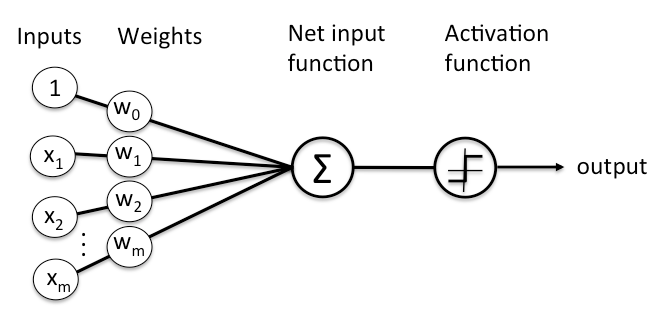


# Next Step

## Neural Network

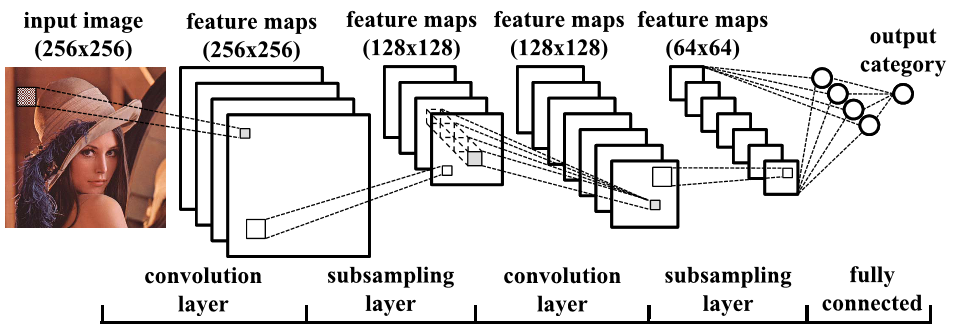
인공신경망(Neural Network)은 뇌가 패턴을 인식하는 방식을 모사한 알고리즘이다. 이미지, 소리, 문자 등의 데이터를 분류, 군집을 이용하여 해석하면 데이터 안의 패턴을 인식하는 것이 가능하다. 이 과정을 간단히 표현하자면, 데이터를 위에 여러 층(layer)을 얹어 각 층의 라벨링 되어있는 데이터를 기반으로 분류기를 ‘학습’하여 자동으로 데이터를 분류시키는 것이다.

이 분류기는 마치 인간이 학습하듯이 인공신경망으로 특징을 추출하고, 그 특징을 다시 다른 학습 알고리즘의 입력으로 사용하여 분류와 회귀를 하는 ‘강화학습’을 하게 된다. 또 이 ‘학습’이 바른 알고리즘으로 반복됐다면 더욱 정확한 분류와 회귀를 할 수 있다.



인공신경망은 한 층이 여러 개의 노드로 이루어져 있으며, 이 노드 안에서 실제로 연산이 일어난다. 연산은 인간의 신경을 구성하는 뉴런에서 일어나는 처리 과정을 모사하도록 설계되어 있는데, 노드가 일정 크기(인간 신경의 역치) 이상의 자극을 받으면 반응한다. 이 반응의 크기는 입력 값과 노드의 계수(역치, 가중치)를 곱한 값과 비례한다. 따라서 이 계수를 조절하여 여러 입력에 다른 가중치를 부여할 수 있다. 이렇게 얻은 값은 전부 더해져 활성함수(Activation function)의 입력으로 들어가고, 활성 함수의 결과가 노드의 출력이 되어 분류나 회귀 분석에 쓰인다.

## Convolution Neural Network[[15]](#footnote-15)



컨볼루전 신경망(Convolution Neural Network)이라 하는 기술을 프로젝트에 적용할 계획이다.

전체적으로 앞서 서술한 Neural Network를 따르는데 위 그림과 같이 이미지의 고유한 구조가 있다고 가정하고 사진을 벡터로 입력 받아 히든 레이어를 통해 가중치를 계산하여 Output을 추측하게 되는 방식이다.

CNN 방식이라고 불리는 이 방식은 ImageNet 이미지 분류 시합에서 다른 경쟁자들을 이기고 우승을 차지한 방식으로 이미지 머신 러닝 분야에서 유용한 툴이 되었다.

신경망은 머신 러닝 계산 방식의 하나일 뿐이며 실제로 행렬 연산의 가중치를 구하는 것이 가장 중요하다고 한다.

# Schedule

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Schedule | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 설치 및 환경 구축 |  |  |  |  |  |  |  |  |  |
| OpenCV 알고리즘 적용 및 구현 |  |  |  |  |  |  |  |  |  |
| TensorFlow 이미지 학습 및 구현 |  |  |  |  |  |  |  |  |  |
| Debugging 및 테스트 |  |  |  |  |  |  |  |  |  |

# Reference

<https://blog.miguelgrinberg.com/post/video-streaming-with-flask> - Flask 를 이용한 비디오 스트리밍

<http://flask-docs-kr.readthedocs.io/ko/latest/ko/patterns/streaming.html> - Flask 를 이용한 비디오 스트리밍

<https://www.tensorflow.org/versions/r0.11/get_started/os_setup.html#installing-from-sources> – TensorFlow Install

<http://flask-docs-kr.readthedocs.io/ko/latest/quickstart.html> - Flask Quick Start

<http://www.pyimagesearch.com/2015/06/22/install-opencv-3-0-and-python-2-7-on-ubuntu/> - OpenCV Installation on Ubuntu

<http://docs.opencv.org/3.0-beta/modules/refman.html> - OpenCV API Reference

<http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html> - OpenCV Python Tutorial

<https://www.youtube.com/watch?v=EB2A2ej70Pw&list=WL&index=27> – TensorFlow를 이용한 얼굴 인식 구현

<https://www.youtube.com/watch?v=QfNvhPx5Px8&list=WL&index=28> – Build a TensorFlow Image Classifier

<https://www.youtube.com/watch?v=KoMTYnlNNnc> – Optical Flow Lecture

<https://tensorflowkorea.wordpress.com/2016/04/28/first-contact-with-tensorflow/> - First Contact Tensorflow

<http://aikorea.org/cs231n/convolutional-networks-kr/> - Convolution Neural Network

1. Seoul region에서는 지원하지 않으므로 Tokyo region의 인스턴스 사용 [↑](#footnote-ref-1)
2. Intel Zeon vCPU 1, Memory 1GiB [↑](#footnote-ref-2)
3. 출처<https://commons.wikimedia.org/wiki/File:CUDA_processing_flow_(En).PNG?uselang=ko> [↑](#footnote-ref-3)
4. 출처 <https://realpython.com/blog/python/kickstarting-flask-on-ubuntu-setup-and-deployment/> [↑](#footnote-ref-4)
5. 참고<http://blog.celingest.com/en/2013/02/25/nginx-vs-apache-in-aws/> - AWS클라우드 컴퓨팅 환경에서의 Apache vs Nginx 속도 테스트 [↑](#footnote-ref-5)
6. CSS및 Javascript를 이용한 웹 디자인 추후 적용 [↑](#footnote-ref-6)
7. 서버에서 연산을 하고, 아직 정밀하게 FPS를 설정하지 않았기 때문에 기존 비디오보다 약간 늦게 스트리밍 된다. [↑](#footnote-ref-7)
8. WAS – Flask - [Structure](#_Structure) 참조 – page 11 [↑](#footnote-ref-8)
9. 이 항목은 Tensorflow classifier로 대체할 예정. [↑](#footnote-ref-9)
10. API Reference <http://docs.opencv.org/3.0-beta/modules/video/doc/motion_analysis_and_object_tracking.html#cv2.calcOpticalFlowFarneback> [↑](#footnote-ref-10)
11. 출처 <http://darkpgmr.tistory.com/66> [↑](#footnote-ref-11)
12. 출처 <http://docs.opencv.org/trunk/d1/db7/tutorial_py_histogram_begins.html> [↑](#footnote-ref-12)
13. API Reference <http://docs.opencv.org/3.0-beta/modules/imgproc/doc/histograms.html#cv2.calcHist> [↑](#footnote-ref-13)
14. API Reference <http://docs.opencv.org/3.0-beta/modules/imgproc/doc/histograms.html#comparehist> [↑](#footnote-ref-14)
15. 출처 [www.semanticscholar.org/](http://www.semanticscholar.org/) [↑](#footnote-ref-15)