

졸업작품 중간 보고서

얼굴 인식 및 영상 추적 분석

최종원 | 종합설계2 | 2016-10-21

# 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 동영상을 업로드 하는 인터페이스를 제공한다.

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

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

# Development Environment

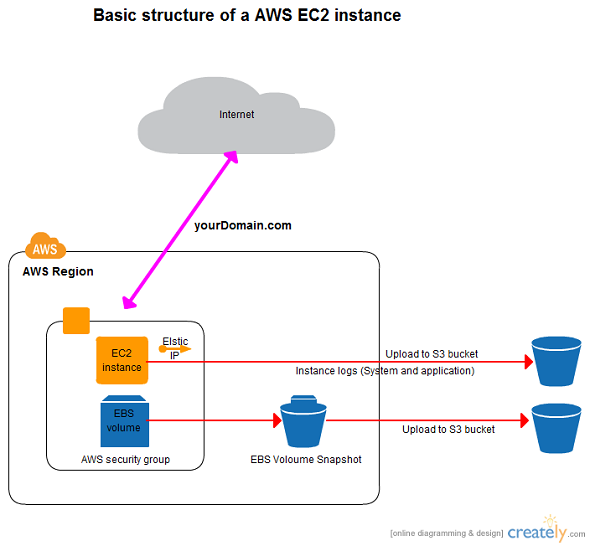
## Amazon Web Services(AWS)

여러 클라우드 컴퓨팅 서비스를 제공하는 웹 서비스 인프라이다.

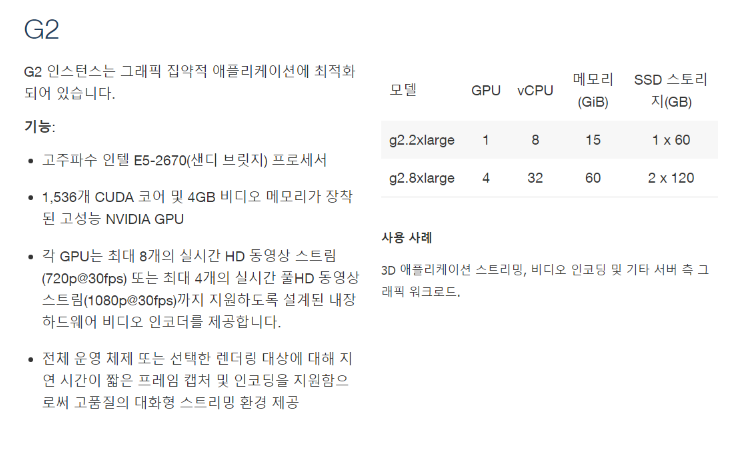
간단하게 자신의 서버를 생성할 수 있고 사용한 만큼 비용을 지불하게 된다.

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



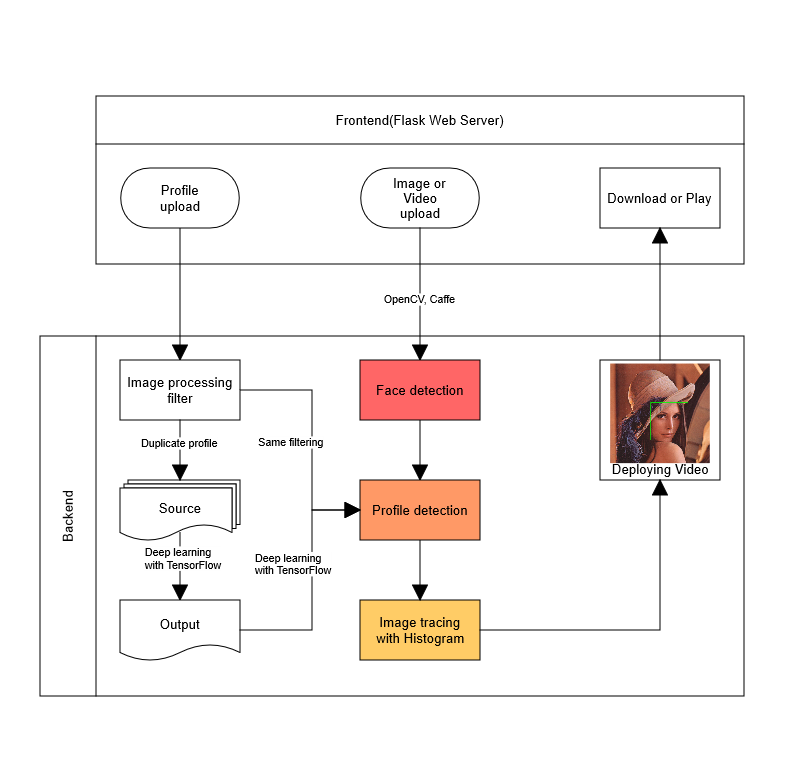
* 테스트는 t2.micro 인스턴스[[2]](#footnote-2)를 이용
* Ubuntu 14.04.3 LTS 운영체제에 설치한 핵심 오픈소스 라이브러리들

CUDA 7.5, cuDNN 5.1

OpenCV 3

TensorFlow r0.11

## Architecture



AWS 내부의 구조는 위 그림과 같다.

클라이언트가 웹 서버로 사진이나 동영상을 업로드하게 되면 서버에서 OpenCV오픈 소스 라이브러리로 얼굴 인식 및 추적을 시작한다.

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

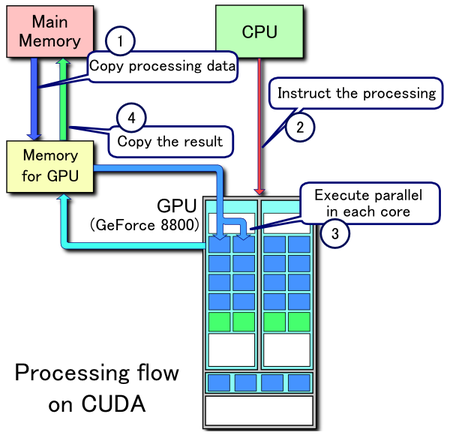
# Install

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.

### 개발도구 설치



### 이미지 로드 라이브러리 설치



### GTK 설치

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



### 비디오 프로세싱 라이브러리



### pip 패키지 설치



### OpenCV Project Clone



### OpenCV 빌드



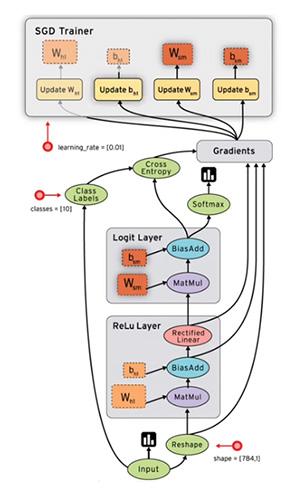
빌드가 완료되면 OpenCV 설치가 완료된 것이다.

## 텐서플로우

**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 동영상 또는 사진을 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.



#### app.py ( 기본 틀 )



#### Structure



### 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.



#### Configure



### Nginx

더 적은 자원으로 더 빠르게 데이터를 서비스 할 수 있는 차세대 웹서버.

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.



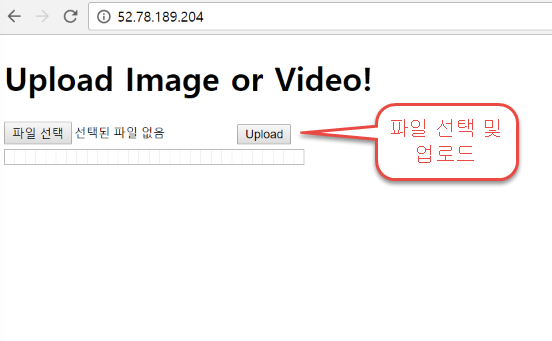
#### configure



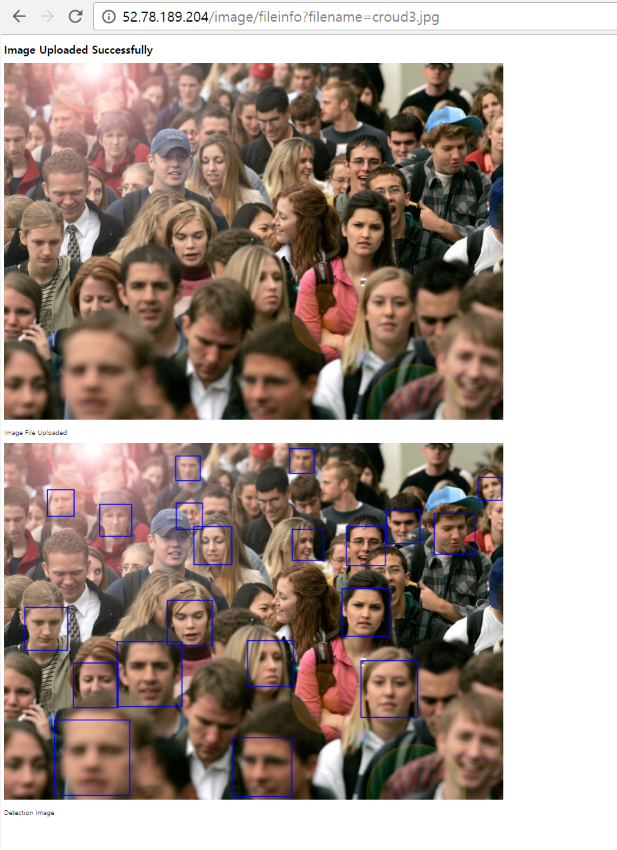
* 80번 포트를 열고 클라이언트가 업로드 할 수 있는 제한을 2G로 두었다.
* uWSGI의 소켓을 참조하여 프로젝트 루트 디렉터리를 설정했다.
* Static경로를 Nginx에서 핸들링하여 웹 서버의 부하를 줄였다.

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

## 메인화면



## 이미지를 업로드 했을 경우



## 동영상을 업로드 했을 경우[[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 사진이며 아직 기능 구현은 되지 않음.
* 스트리밍하는 영상은 되감기를 할 수 없다는 것이 단점.

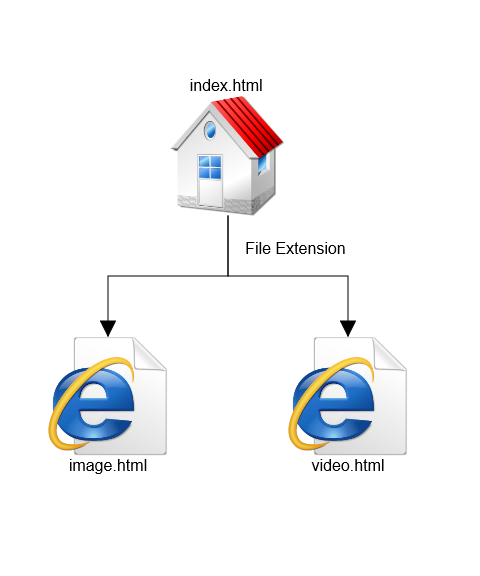
# Details

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

동영상의 경우 Server side에서 Response를 나누어 전송하는 스트리밍 기법을 적용한다.

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

동영상을 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

Opencv

Haar-like feature(deprecate)

Optical Flow

Histogram

앞으로의 계획

Backend :

Tensorflow

Convolusion neural network

개발 일정

참고문헌 및 영상

<https://blog.miguelgrinberg.com/post/video-streaming-with-flask>

<http://flask-docs-kr.readthedocs.io/ko/latest/ko/patterns/streaming.html>

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

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

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

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

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

<https://www.youtube.com/watch?v=EB2A2ej70Pw&list=WL&index=27>

<https://www.youtube.com/watch?v=QfNvhPx5Px8&list=WL&index=28>

<https://www.youtube.com/watch?v=KoMTYnlNNnc>

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. CSS와 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)