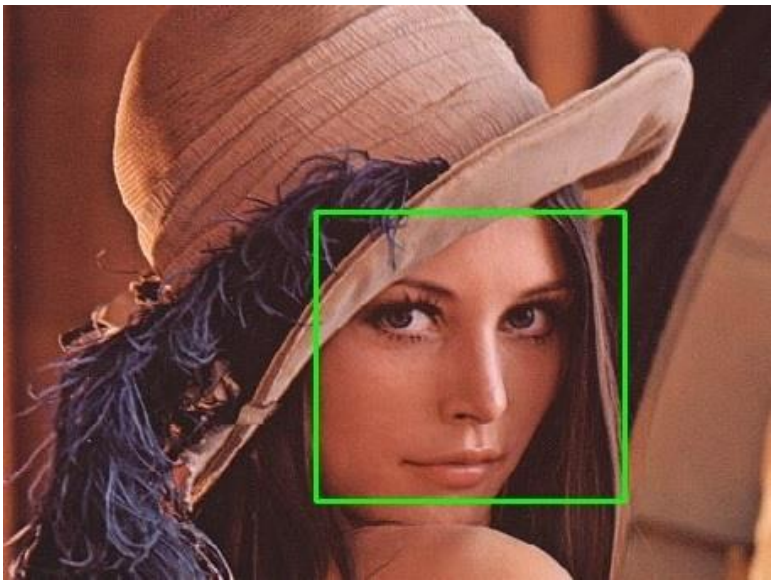


졸업작품 최종 보고서

웹 서버 업로드를 통한 얼굴 인식 및 영상 추적 분석

최종원 | 종합설계 2 | 2016-11-11

Contents

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 Output 이라는 원하는 결과를 내 놓는 자동화 방식이 아니라, 비슷한 결과를 Input 한다. 즉 Input 을 경험으로 확대하여 기계가 학습과 추측을 할 수 있게 되었다.

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 향상됨을 기대해 볼 수도 있다.

또한 대부분 웹캠이나 카메라를 통해 인식하는 경우가 많아 원하는 영상을 처리하는 서비스가 없었다. 영상을 업로드 하면 인코딩 문제로 시간이 매우 오래 걸려 서비스를 하기가 곤란하기 때문이다.

웹 서비스에 영상 처리 기술을 탑재하여 실시간 스트리밍으로 프레임을 클라이언트에 전송할 수 있기에 영상처리를 제공하는 웹 서비스를 계획하게 되었다.

- 많은 CCTV 를 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 API 를 오픈소스로 제공한다.

서비스 형태로 제공하기 위해 Flask 를 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

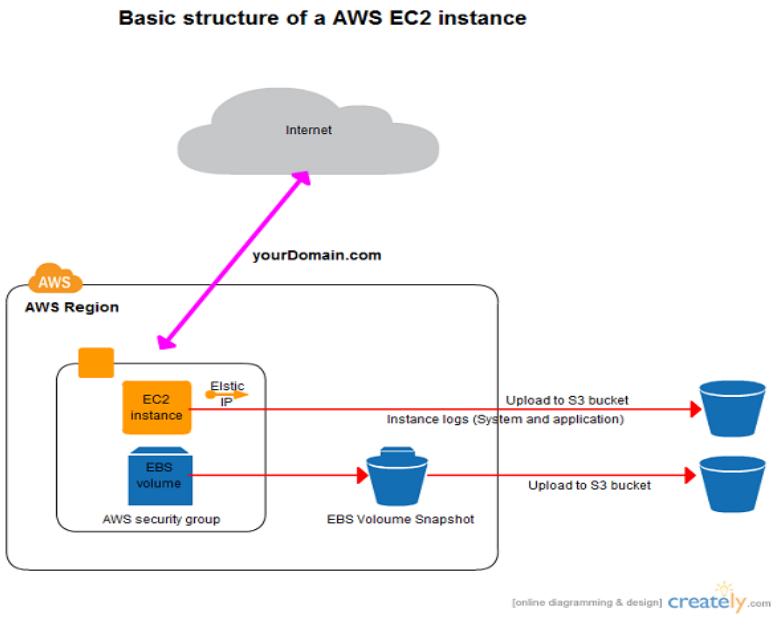
AMAZON WEB SERVICES(AWS)

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 영상처리를 하기에 적합한 인스턴스인 g2.2xlarge 인스턴스[1]를 이용

G2

G2 인스턴스는 그래픽 집약적 애플리케이션에 최적화되어 있습니다.

기능:

- 고주파수 인텔 E5-2670(샌디 브릿지) 프로세서
- 1,536개 CUDA 코어 및 4GB 비디오 메모리가 장착된 고성능 NVIDIA GPU
- 각 GPU는 최대 8개의 실시간 HD 동영상 스트림(720p@30fps) 또는 최대 4개의 실시간 풀HD 동영상 스트림(1080p@30fps)까지 지원하도록 설계된 내장 하드웨어 비디오 인코더를 제공합니다.
- 전체 운영 체제 또는 선택한 렌더링 대상에 대해 지연 시간이 짧은 프레임 캡처 및 인코딩을 지원함으로써 고품질의 대화형 스트리밍 환경 제공

모델	GPU	vCPU	메모리 (GiB)	SSD 스토리지(GB)
g2.2xlarge	1	8	15	1 x 60
g2.8xlarge	4	32	60	2 x 120

사용 사례

3D 애플리케이션 스트리밍, 비디오 인코딩 및 기타 서버 측 그래픽 워크로드.

- 테스트는 t2.micro 인스턴스[2]를 이용

- Ubuntu 14.04.3 LTS 운영체제에 설치한 핵심 오픈소스 라이브러리들

  CUDA 7.5, cuDNN 5.1

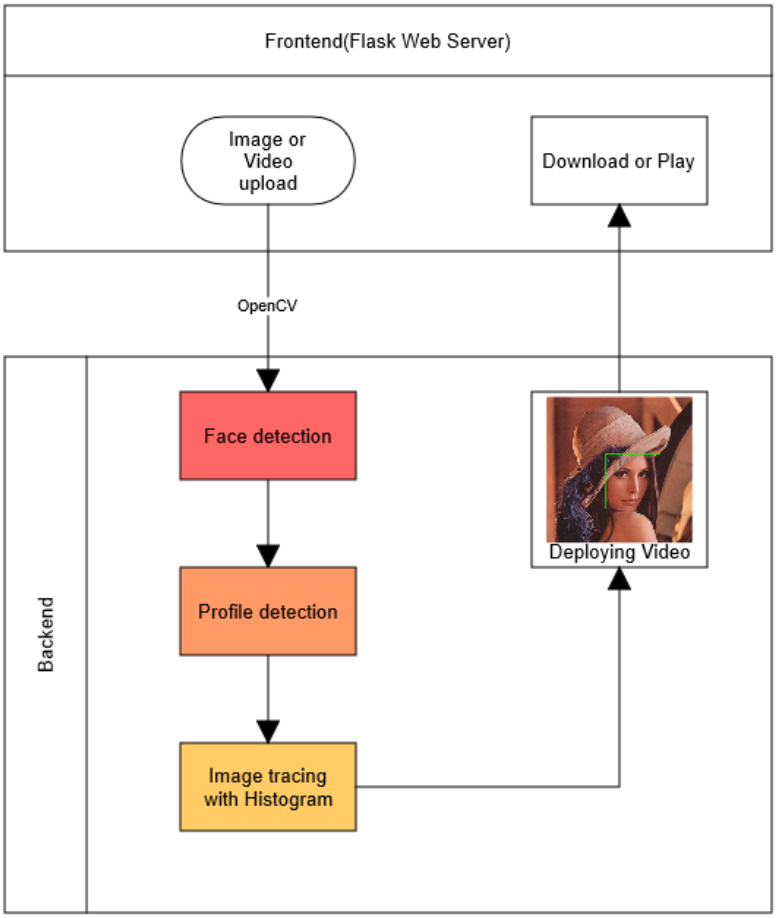
  OpenCV 3

  TensorFlow r0.10

[1] Seoul region 에서는 지원하지 않으므로 Tokyo region 의 인스턴스 사용

[2] Intel Zeon vCPU 1, Memory 1GiB

ARCHITECTURE



AWS 내부의 구조는 위 그림과 같다.

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

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

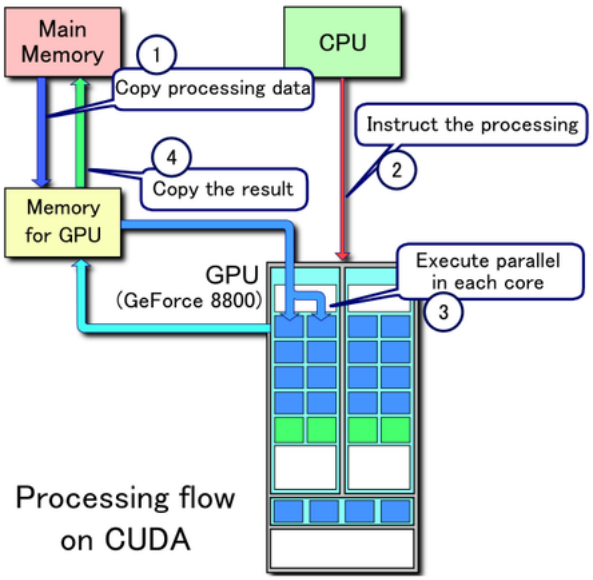
Install

AWS 클라우드 환경에서 제공하는 인스턴스는 Ubuntu 만 설치된 상태이다.

라이브러리와 패키지들은 따로 설치를 해주어야 한다.

CUDA

CUDA (Compute Unified Device Architecture)는 NVIDIA 가 개발한 GPGPU 기술이다.



[3]

CUDA 처리 흐름의 예

1. 메인 메모리를 GPU 메모리로 복사
2. CPU 가 GPU 에 프로세스를 지시함
3. GPU 각 각 코어에 병렬 수행
4. GPU 메모리로부터의 결과물을 메인 메모리에 복사

[3] 출처 https://commons.wikimedia.org/wiki/File:CUDA_processing_flow_(En).PNG?uselang=ko

- CUDA 7.5 설치

```
sudo dpkg -i cuda-repo-ubuntu1404_7.5-18_amd64.deb

sudo apt-get update
sudo apt-get upgrade

sudo apt-get install git libatlas-base-dev python-pip python-dev
gcc g++ gfortran
sudo apt-get install linux-image-extra-`uname -r`

sudo apt-get install cuda
```

- cuDNN 5.1 설치

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

- CUDA 환경변수 설정

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

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 Tracking 을 위해 사용하였다.

- Developer tool 설치

```
sudo apt-get install build-essential cmake pkg-config
```

- 이미지 로드 라이브러리 설치

```
sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev
```

- GTK 설치

AWS EC2 로 원격 접속을 하기 때문에 이미지나 영상을 보려면 X window 를 통해 이미지를 볼 수 있다. 그러기 위해 필요한 라이브러리이다.

```
sudo apt-get install libgtk2.0-dev
```

- 비디오 프로세싱 라이브러리

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-
dev libv4l-dev
```

- pip 패키지 설치

```
sudo apt-get install python-numpy python-matplotlib
```

- OpenCV Project Clone

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

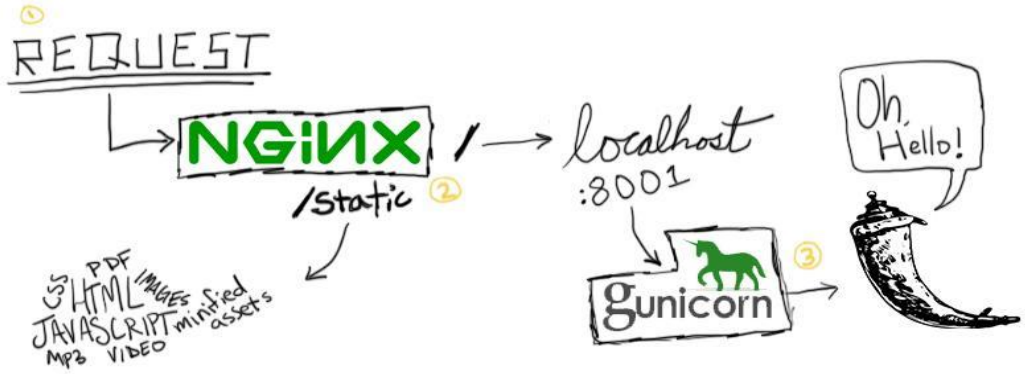
- OpenCV 빌드

```
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local \
  -D WITH_CUDA=ON \
  -D WITH_CUBLAS=1 \
  -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
  -D BUILD_EXAMPLES=ON ..

make -j8
sudo make -j8 install
sudo ldconfig
```

빌드가 완료되면 OpenCV 설치가 완료된 것이다.

WAS

업로드한 영상 또는 이미지를 처리해주는 웹 서버가 필요하다. 아래는 Http 요청을 Nginx 와 Gunicorn 을 통해 플라스크로 향하는 개략적인 그림이다.



[4]

이 프로젝트에서는 Gunicorn 대신 uWSGI 애플리케이션을 사용했다.

Flask

Python 에서 웹 서비스가 가능하도록 만들어주는 프레임워크.

OpenCV 와 TensorFlow 가 Python API 를 제공하기 때문에 하나의 프로젝트에서 Import 하기 위해서 선택했다.

- Install

```
sudo -H pip install flask
```

[4] 출처 https://realpython.com/blog/python/kickstarting-flask-on-ubuntu-setup-and-deployment/

- app.py (기본 틀)

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('upload.html')

if __name__ == "__main__":
  app.run()
```

- Structure

아래 구조는 이 프로젝트 전체적인 구조와 일치한다.

```
flask
├── app.py
│   ├── stream.py
│   ├── detection.py
│   └── infofile.py
│
├── templates/
│   ├── layout.html
│   ├── upload.html
│   ├── image.html
│   └── video.html
│
├── static/
│   ├── uploads/
│   ├── js/
│   └── css/
│
├── .gitignore
├── README.md
├── favicon.ico
└── uwsgi.ini
```

uWSGI

Python WSGI 서버 중 한가지이다.

WSGI 는 Web Service Gateway Interface 의 약자로 Python 스크립트가 웹 서버와 통신하기 위한 WSGI 의 미들웨어이다.

소켓을 열어 Flask 프레임워크와 Nginx 웹 서버의 중간 역할을 한다.

Flask 프레임워크에서 app.py 를 모듈로 하여 서비스를 한다.

- Install

```
sudo -H pip install uwsgi
```

- Configure

```
[uwsgi]
master=true
chmod-socket=666
socket=%n.sock
enable-threads=true
process=4
threads=4
manage-script-name=true
module=app
callable=app
vacuum=true
```

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]

따로 웹 서버 없이 uWSGI 만으로도 서비스가 가능하지만 Nginx 가 가진 향상된 Static content 핸들링을 통해 부하를 조금이라도 줄여줄 수 있어서 선택하게 되었다.

- Install

```
sudo apt-get install nginx
```

[5] 참고 http://blog.celingest.com/en/2013/02/25/nginx-vs-apache-in-aws/ - AWS 클라우드 컴퓨팅 환경에서의 Apache vs Nginx 속도 테스트

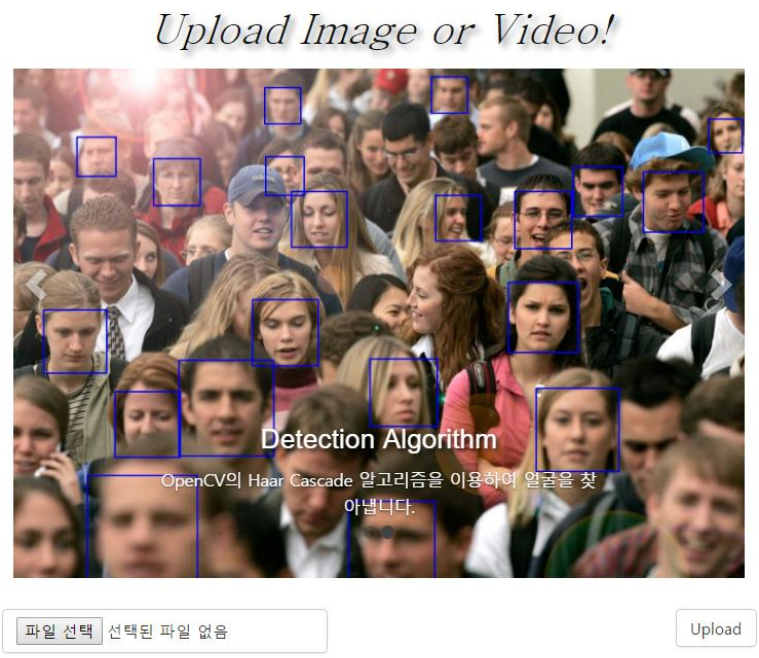
- Configure

```
upstream uwsgicluster {
  server unix:///var/www/flask/uwsgi.sock;
}


server {
  listen 80;
  server_name _;
  access_log /var/log/uwsgi/access_log;
  client_max_body_size 2G;


  location / {
    root /var/www/flask;
    uwsgi_pass uwsgicluster;
    include uwsgi_params;
    uwsgi_param UWSGI_SCRIPT app.py;
  }

  location /static {
    alias /var/www/flask/static;

    if (!-f $request_filename) {
      uwsgi_pass uwsgicluster;
    }
  }
}
```

- HTTP 80 번 포트를 통해 서버에 접속한다.
- 클라이언트가 업로드 할 수 있는 제한을 2G 로 두었다.
- uWSGI 의 소켓을 참조하여 프로젝트 루트 디렉터리를 설정했다.
- Static 경로를 Nginx 에서 핸들링하여 웹 서버의 부하를 줄였다.

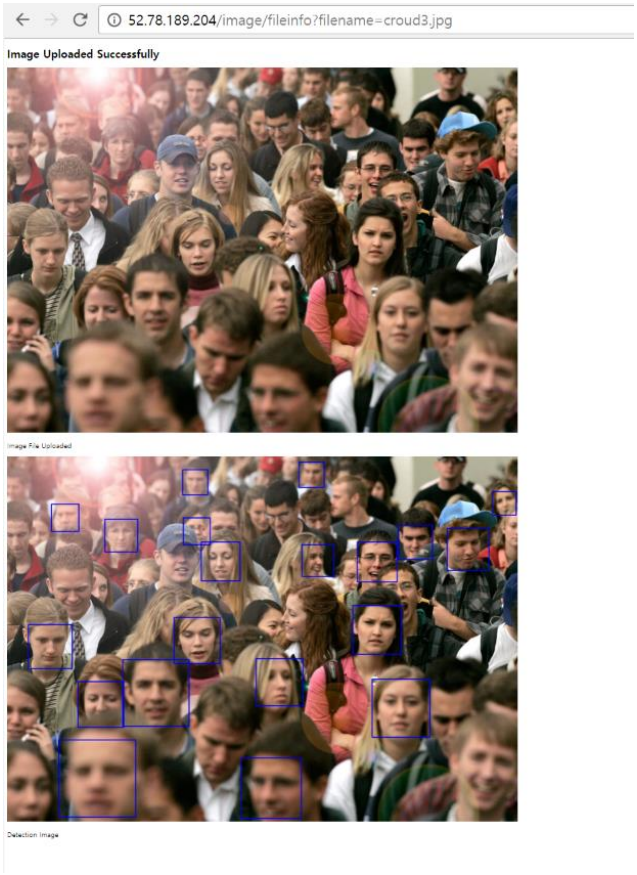
Results

MAIN PAGE



- HTML5, CSS, JavaScript, Bootstrap 을 이용하여 프론트엔드를 디자인했다.
- 메인 화면은 Carousel 을 이용하여 5 초마다 예제들이 전환되도록 전시한 모습이다.
- 이미지 또는 영상을 업로드할 수 있으며 파일 확장자에 따라 다른 웹페이지를 렌더링한다.

IMAGE UPLOAD



- 이미지를 업로드하게 되면 Haar Cascade Detection 알고리즘을 적용한 사진이 원본 사진 아래에 생성되게 된다.

VIDEO UPLOAD[6]

- 위쪽 영상이 업로드한 비디오이며 아래쪽 영상은 얼굴인식 및 추적을 하면서 스트리밍하여 프레임을 뿌려준다..
- 아래쪽 파일 업로드는 스트리밍하는 프레임에서 특정 얼굴을 추적하기 위한 쿼리 이미지를 업로드 할 수 있다. 히스토그램 기술을 통해 비슷한 쿼리를 한번 더 필터링한다.

[6] 서버에서 연산을 하고, 아직 정밀하게 FPS 를 설정하지 않았기 때문에 기존 비디오보다 약간 늦게 스트리밍 된다.

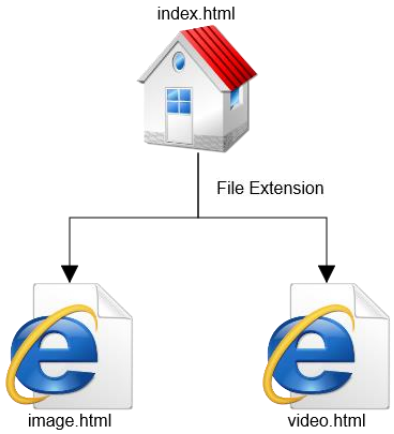
Details

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

FRONTEND[7]

- 루트 디렉터리인 flask 는 절대경로 /var/www/flask 에 위치하고 있으며 Nginx 와 uWSGI 를 통해 서비스 한다.
- uWSGI 게이트웨이 인터페이스는 app.py 를 모듈로 메인 페이지를 구성한다.
- app.py 는 stream.py 와 infofile.py, detection.py 를 import 하고 있다.

app.py



내용을 아주 개략적으로 나타내면 위 그림과 같다.

메인 페이지를 index.html 라고 했는데 템플릿에 있는 upload.html 을 가리킨다.

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

[7] WAS – Flask - Structure 참조 – page 11

- 파일을 업로드하면 절대경로 /var/www/flask/static/uploads 로 업로드 된다.
- Nginx configuration 에서 클라이언트 최대 업로드 용량을 2G 로 지정했으므로 글로벌 변수로 선언한다.
- Python 에서는 글로벌 동적 변수를 지정하기 위해 변수 앞에 global 이라고 선언을 해야하지만 Flask 에서는 글로벌 동적 변수로 app.config 딕셔너리를 지원한다.

app.py 일부 (Server side)

```
UPLOAD_PATH='/var/www/flask/static/uploads'
ALLOWED_EXTENSIONS=set(['png','jpg','jpeg','gif','mp4'])

# global variable config
app.config['MAX_CONTENT_LENGTH'] = 2 * 1024 * 1024 * 1024

# global variable
app.config['UPLOAD_PATH'] = UPLOAD_PATH
app.config['fileinfo'] = None
app.config['tfinfo'] = None
```

upload.html 에서 업로드 버튼을 누르게 되면 POST method 로 웹 서버에 요청하게 된다.

웹 서버는 업로드한 파일을 서버에 저장한 후 GET method 를 통해 자기 자신 URL 로 Redirection 한다. 이때 글로벌 동적 변수에 저장해둔 파일명으로 확장자를 분석하여 image.html 또는 video.html 템플릿을 로드할지 결정하게 된다.

templates/upload.html 일부(Client side)

Flask 에서는 Jinja2 템플릿 엔진을 이용한다. 아래와 같이 Python 코드를 사용하여 URL 을 나타낼 수 있다.

```
<form action="{{ url_for('upload', info='fileinfo') }}"
method='POST' enctype=multipart/form-data>
       <input type=file name=file>
       <input type=submit value=Upload>
</form>
```

app.py 일부 (Server side)

```
# upload module
@app.route('/upload/<info>', methods=['GET','POST'])
def upload(info):
    if request.method == 'POST':
        file = request.files['file']
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_PATH'],
filename))
            return redirect(url_for('upload', info=info,
filename=filename))

    #### GET method ####
    # filename into global class
    app.config[info] = InfoFile(app.config['UPLOAD_PATH'],
request.args.get('filename'))
    filename = app.config[info].getfilefullname()
    ext = app.config[info].getfileext()

    # redirect from extension
    if ext in ['png','jpg','jpeg','gif']:
        return redirect(url_for('image', info=info,
filename=filename))
    elif ext in ['mp4']:
        return redirect(url_for('video', info=info,
filename=filename))
    else:
        return redirect(url_for('error'))
```

Redirection 한 곳에서 image.html 또는 video.html 템플릿을 로드하게된다.

Design

HTML 을 모듈화 할 수 있는 템플릿인 jinja2 를 이용하여 기본 틀인 layout.html 을 만들었다.

Layout Design

layout.html 일부 (CDN 및 script import 생략)

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>{% block title %}{% endblock %}</title>
    {% block head %}{% endblock %}
  </head>
  <body>
    {% block body %}{% endblock %}
    {% block script %}{% endblock %}
  </body>
</html>
```

위와 같이 jinja2 템플릿 엔진 기호를 이용하면 위에서 선언한 블록을 이용할 경우 이 자리에 코드가 치환된다.

image.html 의 코드 일부 {% block body %} 만 치환하는 경우

```
{% extends 'layout.html' %}
{% block body %}
  ...
{% endblock %}
```

Carousel Design

Bootstrap 에서 제공하는 전시를 위한 자바스크립트이며 슬라이드 쇼처럼 회전목마처럼 일정 주기로 요소들을 순환시키는 플러그인이다.

Carousel 은 Main div 태그 내에 Indicator 와 메인 슬라이드, Control 컴포넌트로 구성되어 있으며 아래와 같이 나타낼 수 있다.

upload.html 의 {% block body %} 일부 (내부 코드 생략)

```
<div id="myCarousel" class="carousel slide" data-ride="carousel">

<!-- Carousel Indicators -->
<li data-target="#myCarousel" data-slide-to="1"></li>

<!-- Wrapper for slides -->
<div class="carousel-inner" role="listbox">
  <div class="item active">
  ...
  </div>
</div>

<!-- Carousel Controls -->
<a class="left carousel-control" href="#myCarousel" data-
slide="prev">
```

video.html

파일을 업로드하게 되면 image.html 과 video.html 로 페이지가 이동된다. 이때 원본 데이터와 영상처리하는 데이터가 나타나는데 한 웹 페이지에 두장의 사진은 나타낼 수 있지만 영상의 경우 영상 처리 된 파일을 생성하지 않고 스트리밍을 하는 방식이므로 html 태그를 이용하여 나타낼 수 없다.(stream.py 참조)

Backend 과정에서 **비동기로 처리된 두 URL 주소**를 하나의 웹페이지로 나타내는 방식을 취했다. 아래와 같이 iframe 태그를 사용하여 나타내었다.

video.html 의 {% block body %} 일부

```
<div id="sourcediv">
  <video id="source" style="width:90%;height:auto;" autoplay
onclick="this.play();">
    <source src="{{ url_for('static', filename='uploads/' ~
request.args.get('filename')) }}" type="video/mp4">
  </video>
</div>
<div id="trackdiv">
  <iframe style="margin:auto;width:90%;"
src="{{ url_for('stream') }}"</iframe>
</div>
```

BACKEND

stream.py

영상의 경우 Server side 에서 Response 를 나누어 전송하는 스트리밍 기법을 적용한다.

업로드한 영상을 처리 한 후 인코딩을 통해 새로운 영상을 만들어서 클라이언트에게 video 태그를 통해 전송하는 방법은 너무 느리다. 아래와 같이 영상을 프레임별로 처리하는 동시에 JPEG 로 인코딩하고 mimetype 을 'multipart/x-mixed-replace'로 하여 실시간으로 Response 를 전송한다.

이렇게 처리하면 **인코딩하여 완전한 영상을 만드는 것이 아니라 매 프레임을 인코딩하여 전송**하기 때문에 시간이 대폭 줄어들게 된다.

stream.py (OpenCV 코드 생략)

```
class VideoStream(object):
    def __init__(self, filename):
        self.video=cv2.VideoCapture(filename)
        self.face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    def __del__(self):
        self.video.release()

    def get_frame(self):
        # video frame read
        success, frame = self.video.read()
        if success:
            ######### opencv coding  #########
              ...
            ##################################
            # cv2.waitKey(rate)
            ret, jpeg = cv2.imencode('.jpg', frame)
            return jpeg.tobytes()
        else:
            return success
```

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

```
@app.route('/stream')
def stream():
    filename = app.config['fileinfo'].getfilefullpath()
    return Response(gen(VideoStream(filename)),
mimetype='multipart/x-mixed-replace; boundary=frame')
```

```
# Video Streaming
def gen(stream):
    while True:
        frame = stream.get_frame()
        if frame==False:
            break
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame +
b'\r\n\r\n')
```

위와 같이 yield 를 이용해 리턴을 하게 되면 영상 프레임이 모두 종료될 때까지 아래와 같이 multipart response 가 발생하게 된다.

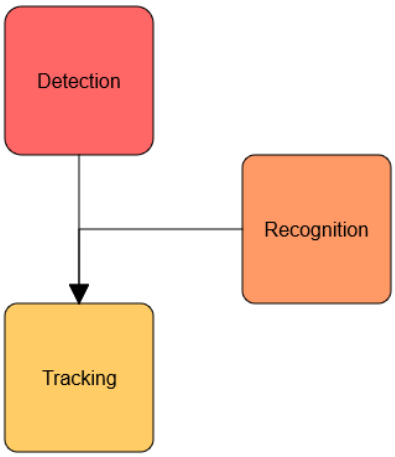
HTTP Response Stream Structure

```
HTTP/1.1 200 OK
Content-Type: multipart/x-mixed-replace; boundary=frame

--frame
Content-Type: image/jpeg

<jpeg data here>
--frame
Content-Type: image/jpeg

<jpeg data here>
...
```

이렇게 업로드 시간만으로 클라이언트에서 스트리밍된 영상을 바로 볼 수 있게 된다.

Video Processing Concepts

프로젝트의 전체적인 과정을 간단하게 나타내면 아래 그림과 같다.



- Detection

  먼저 영상에서 대상을 먼저 찾아야 한다. 현재까지 얼굴 인식을 Haar-like feature 알고리즘을 이용했다. Detection 이 되어야 Tracking 과 Recognition 이 되기 때문에 의존도가 높은 것이 단점이다.

- Recognition

  Detection 된 영역을 Queue 에 추가한 후 다음 프레임에서 다시 Detection 이 일어났을 경우 그 찾아진 얼굴이 Queue 에 등록된 얼굴인지 식별(Recognition)하는 과정이며 Histogram 를 이용하여 장면 전환이나 추적하는 대상이 달라질 경우 필터링하는 역할을 한다.

- Tracking

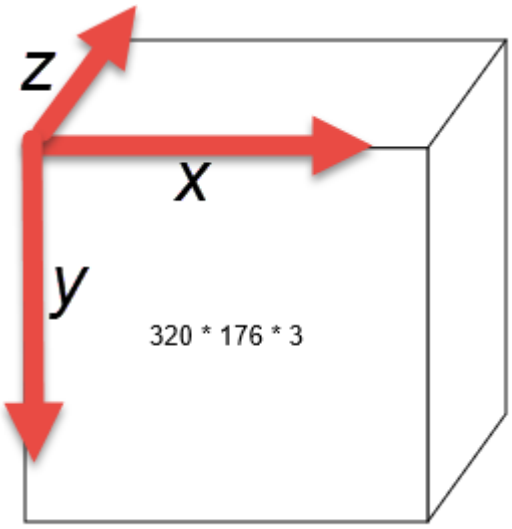
  Detection 이 하나의 영상에서 대상을 찾는 것이라면 Tracking 은 영상에서 특정 대상의 위치 변화를 추적하는 것이다. Optical Flow 기술로 위치변화를 추적한다.

OpenCV Frame

OpenCV 에서 하나의 프레임만 읽어 오는 코드이다.

```
import cv2

cap = cv2.VideoCapture(filename)
ret, frame = cap.read()
```

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.

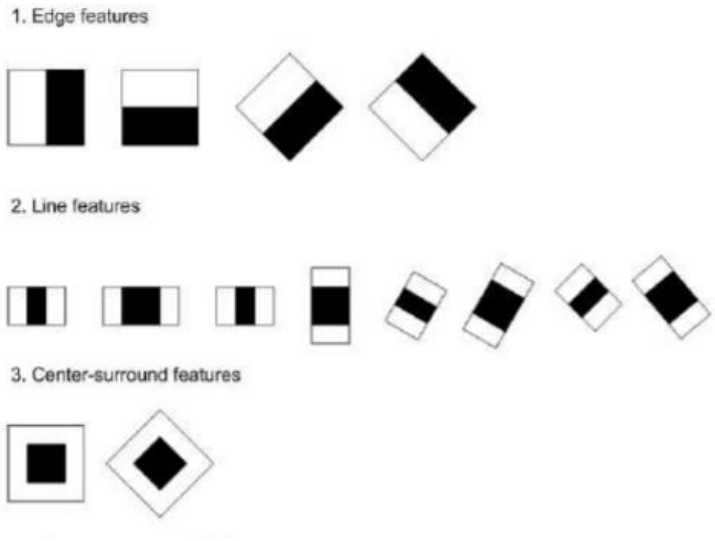
Z 축은 빨강, 초록, 파랑을 나타내며 BGR 순서로 나타낸다. 실제로 frame 의 shape 값은

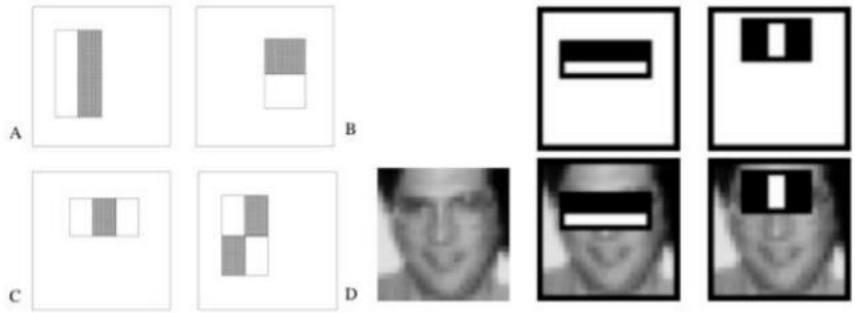
(176, 320, 3) (**row[y], col[x], BGR[3]**) 매트릭스 형태로 나타난다.

색상 프레임을 Gray scale 로 변경할 경우 shape 값은 (176, 320, 3) 에서 (176, 320)값이 된다.

Haar-like feature

얼굴의 특징을 찾는 기법중의 하나이다. 이미지를 Haar 라는 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 Feature 를 이용한 영상에서는 영역 사이의 밝기 차이를 이용한 것으로 얼굴이 아닌데도 인식을 간혹 하는 오류가 있을 수 있다.

stream.py 일부

```
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for x, y, w, h in faces:
          cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),3)
          roi_gray = gray[y:y+h, x:x+w]
          roi_color = frame[y:y+h, x:x+w]
```

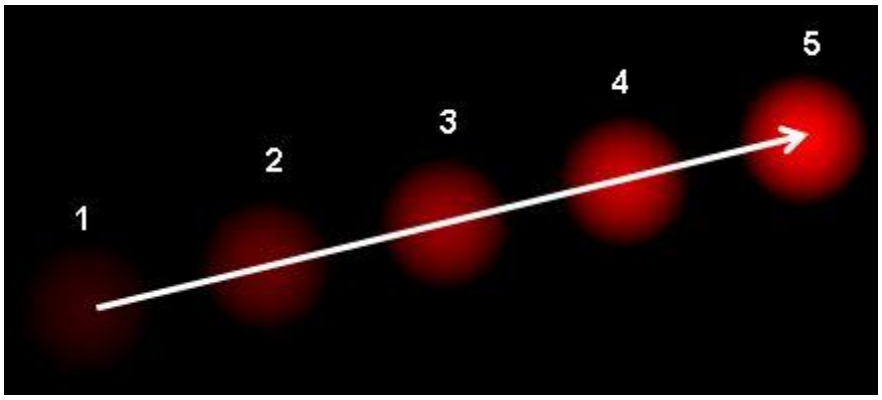
기본적으로 분류가 된 xml 파일을 사용하며 프레임을 Grayscale 로 변환한 다음 흑백 프레임의 밝기 차와 분류한 내용을 대조하여 얼굴을 검출하게 된다.

검출한 얼굴은 사각형 영역값을 리턴하며 파란색으로 프레임 위에 더했다.

Optical Flow

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

Theory



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

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

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

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

우변을 테일러 근사하면

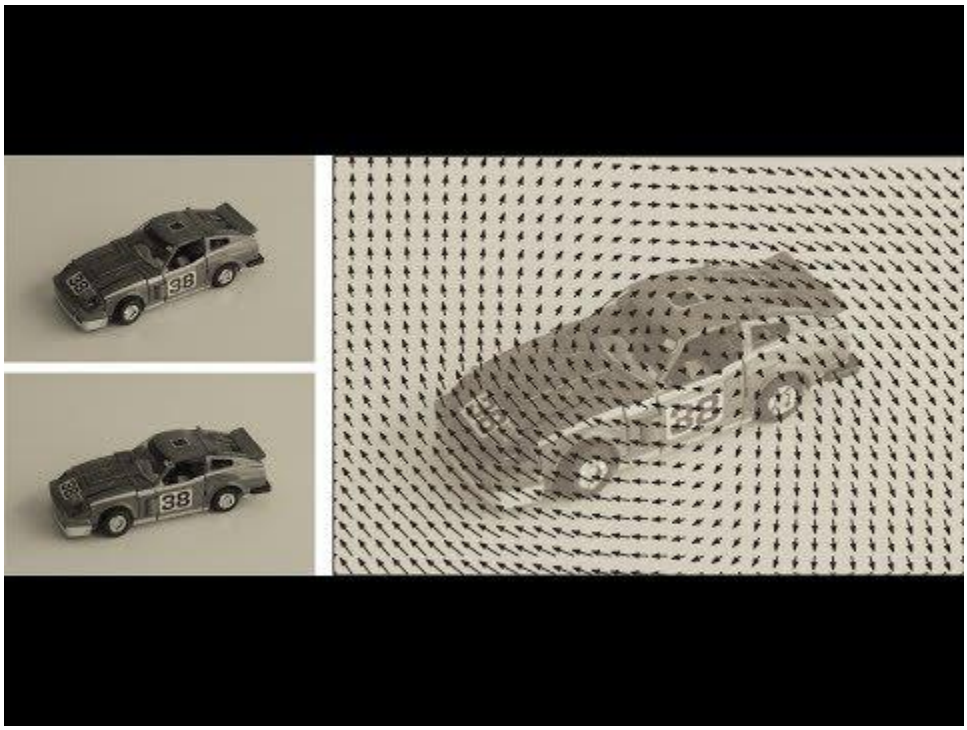
$$I(x, y, t) + \frac{\partial I}{\partial x}\frac{dx}{df} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} \cdot \frac{dt}{dt}$$

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

$$f_x = \frac{\partial f}{\partial x}\ ;\ f_y = \frac{\partial f}{\partial x}$$
$$u = \frac{dx}{dt}\ ;\ v = \frac{dy}{dt}$$
$$f_x u + f_y v + f_t = 0$$

위 식을 Optical Flow Equation 이라고 한다.

Category



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

Optical Flow 의 종류에는 Lucas-kanade Algorithm 과 Farneback Algorithm 이 있다.

- Lucas-kanade Algorithm 은 point algorithm 으로 여러 스케일 이미지 피라미드로 나누는 과정을 통해 모서리 값을 먼저 찾은 다음 이 점이 연속된 프레임에서 얼만큼 이동했는지 flow 를 보여주는 알고리즘이다.
- Farneback Algorithm 은 dense algorithm 으로 여러 스케일 이미지 피라미드로 나누는 과정을 통해 두 프레임간의 전체적인 flow 를 Polynomial Expansion Transform 을 통해 화면 전체를 일정한 간격으로 나누어 모션 벡터를 보여준다.

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

stream.py 일부

```
flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5,
1, max(w,h), 3, 5, 1.2, 0)

for j in range(h):
    for i in range(w):
        [ny, nx] = [y, x] + ((flow[y:y+h, x:x+w][j,i])/(w*h))
```

[에러처리]

cv2.calcOpticalFlowFarneback[8]이란 이름으로 OpenCV 에서 API 를 제공한다.

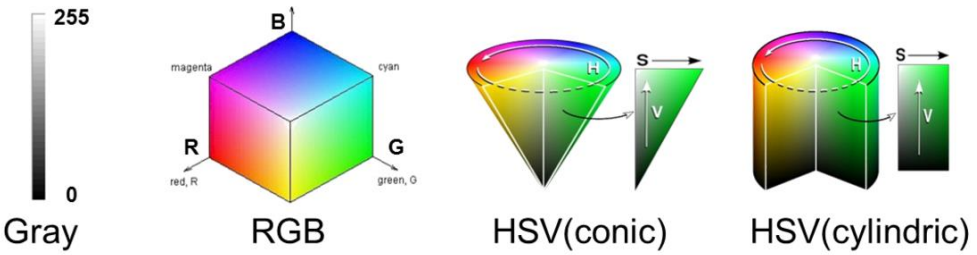
리턴된 flow 값은 프레임 각 픽셀 값에 대한 벡터를 리턴하므로 영역안의 벡터를 모두 더한 값을 주어진 영역으로 나눈 값을 이전 프레임의 좌표에서 이동 시키는 방식을 적용했다.

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

[8] API Reference http://docs.opencv.org/3.0-beta/modules/video/doc/motion_analysis_and_object_tracking.html#cv2.calcOpticalFlowFarneback

Histogram

Color Model[9]



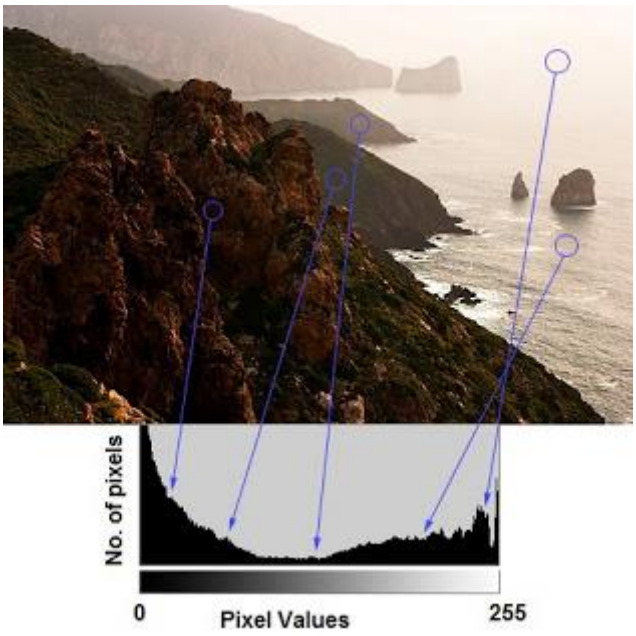
Gray 모델은 색(color) 정보를 사용하지 않고 밝기 정보만으로 영상을 표현하는 것이다. 검정색 0 부터 흰색 255 까지 총 256 단계의 밝기값(intensity)으로 영상 픽셀값을 표현한다.

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

HSV 모델은 Hue(색조), Saturation(채도), Value(명도)의 3 가지 성분으로 색을 표현한다. Hue 는 색조를, Saturation 은 그 색이 얼마나 선명한 색인지, Value 는 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 0 이면 무채색(gray 색), 255 면 가장 선명한 색임을 나타낸다. V 값은 작을수록 어둡고 클수록 밝은 색임을 나타낸다. HSV 색상 모델은 그림과 같이 원뿔(conic) 형태, 원기둥(cylindric) 형태가 있다.

[9] 출처 http://darkpgmr.tistory.com/66

Theory[10]



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

$$n = \sum_{i=1}^{k} m_i$$

[10] 출처 http://docs.opencv.org/trunk/d1/db7/tutorial_py_histogram_begins.html

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 Histogram 을 H1, H2 라고 하고 프레임을 I 라고 할 때 비교 계수를 구하는 공식은 아래와 같다.

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}} \qquad \bar{H_k} = \frac{1}{N} \sum_J H_k(J)$$

여기서 N 은 영역 내의 픽셀 전체의 수이다.

facetracking.py 일부

```
prev_hsv = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2HSV)
frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

prev_hsv_hist = cv2.calcHist([prev_hsv[ny:ny+h, nx:nx+w]], [0],
None, [256], [0,256])
roi_hsv_hist = cv2.calcHist([frame_hsv[y:y+h, x:x+h]], [0], None,
[256], [0,256])

simval = cv2.compareHist(prev_hsv_hist, roi_hsv_hist,
cv2.HISTCMP_CORREL)

# histogram 80% same
if simval < 0.80:
    trackzone = np.delete(trackzone, itertrack, 0)
```

Histogram 에 대한 calcHist[11]와 compareHist[12]라는 이름의 OpenCV 의 API 가 제공된다.

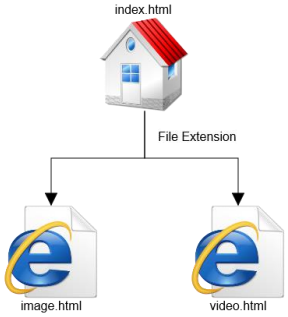
먼저 RGB 색상 모델을 이전 프레임과 현재 프레임에 대하여 HSV 모델로 변경한다.

그 후 이전 프레임에서의 Optical Flow 로 이동한 추적 영역에 대한 Histogram 과 현재 프레임에서의 추적 영역에 대한 Histogram 을 위 관계식으로 계산하여 나온 계수가 0.8 이상일 경우 80% 이상의 유사성을 나타내므로 그 이하의 계수가 나올 경우는 추적 영역에서 제외하는 코드이다.

[11] API Reference http://docs.opencv.org/3.0-beta/modules/imgproc/doc/histograms.html#cv2.calcHist
[12] API Reference http://docs.opencv.org/3.0-beta/modules/imgproc/doc/histograms.html#comparehist

Summary

SITE MAP

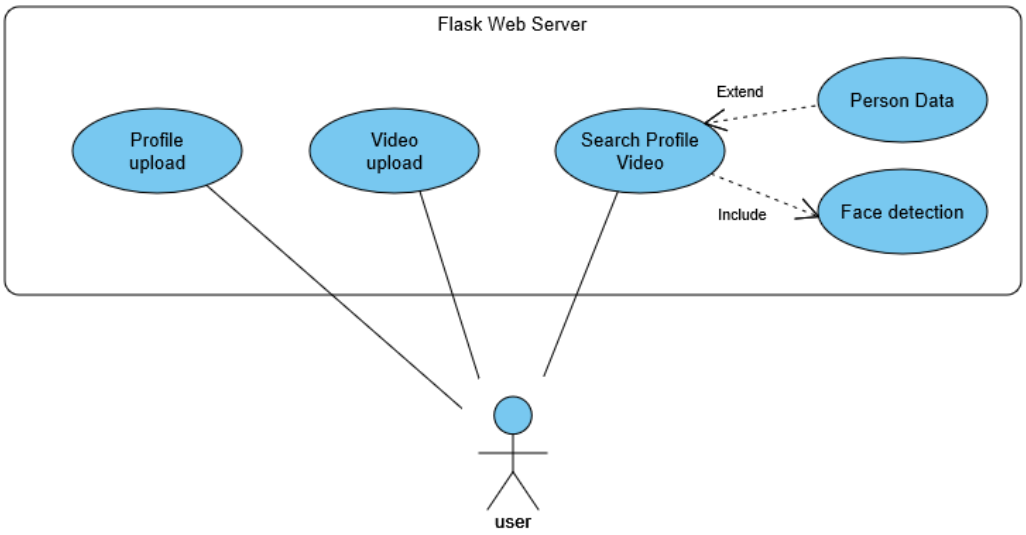


DEVELOPMENT

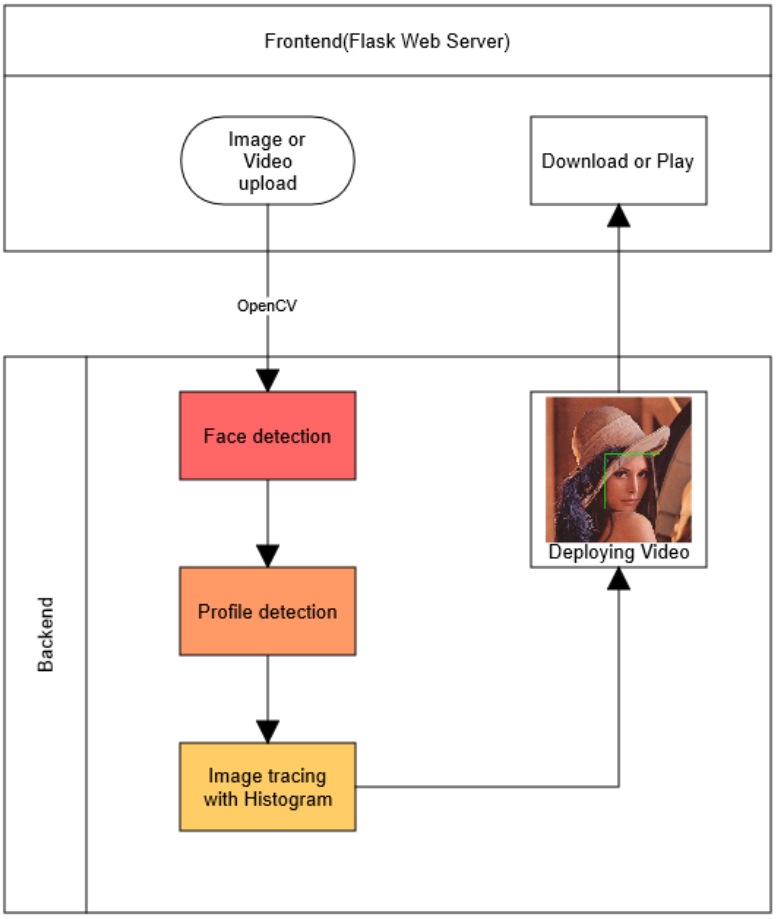
PROJECT STRUCTURE

```
flask
├── app.py
│   ├── stream.py
│   ├── detection.py
│   ├── facetracking.py
│   └── infofile.py
│
├── templates/
│   ├── layout.html
│   ├── upload.html
│   ├── image.html
│   └── video.html
│
├── static/
│   ├── js/
│   ├── uploads/
│   └── css/
│
├── .gitignore
├── README.md
├── favicon.ico
└── uwsgi.ini
```

UML CASES



ARCHITECTURE

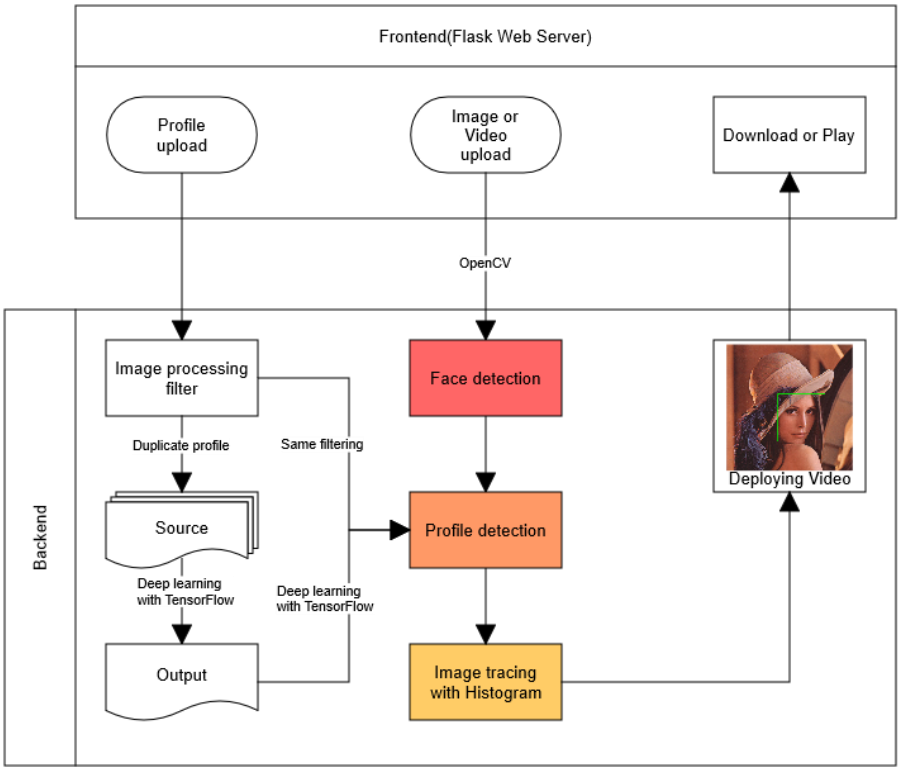


Appendix

본래 이 프로젝트는 실시간 영상에서의 머신러닝을 통해 이미지를 스스로 학습하고 추적하는 방향이었다. CCTV 등을 통해 특정한 사람을 구별하기 위해 머신러닝을 이용한 얼굴 학습을 통해 구별하기 위해 TensorFlow 를 이용하여 특정 얼굴을 찾아내는 모델을 구현했었다. 하지만 제대로 결과를 얻지 못해 원인 분석을 해보았다.

다음은 하고자 했던 최종 Architecture 이며 이 알고리즘을 적용했을 때의 결과와 현재 머신러닝의 한계 및 문제점을 분석해보았다.

ARCHITECTURE



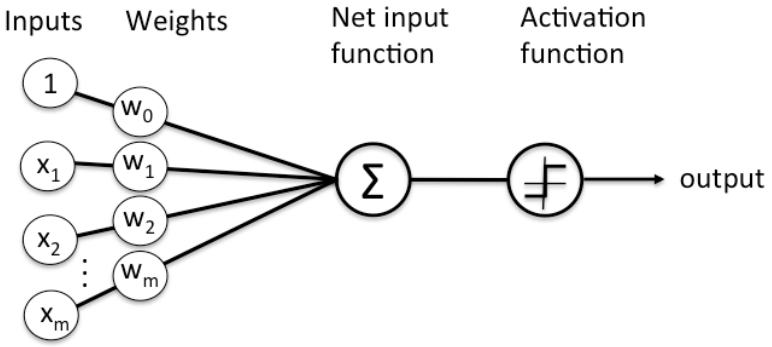
위와 같이 이미지 쿼리를 추가적으로 업로드 하게 되면 비슷한 이미지 프레임을 학습하여 추적하는 방향으로 설계하였다.

NEURAL NETWORK

머신러닝을 통한 이미지 학습은 Neural Network 라는 인공신경망 기술이 필요하다.

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.

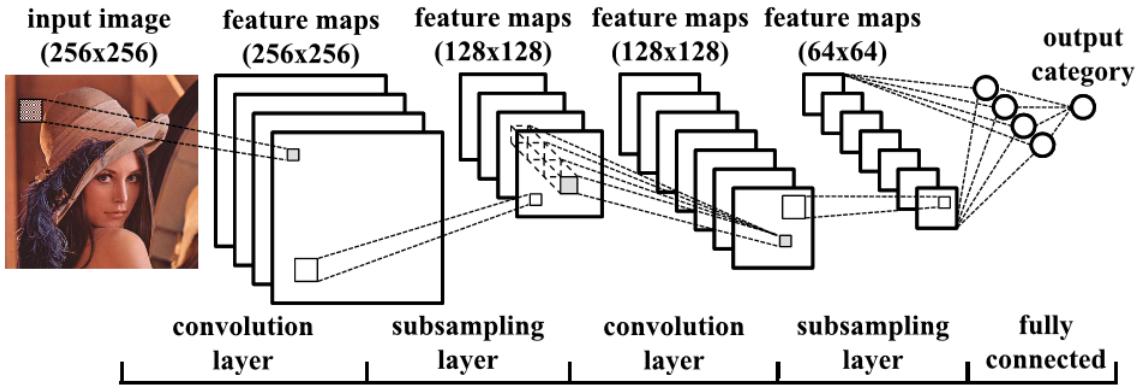
이 분류기는 마치 인간이 학습하듯이 인공신경망으로 특징을 추출하고, 그 특징을 다시 다른 학습 알고리즘의 입력으로 사용하여 분류와 회귀를 하는 '강화학습'을 하게 된다. 또 이 '학습'이 바른 알고리즘으로 반복됐다면 더욱 정확한 분류와 회귀를 할 수 있다.



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[13]

Intro



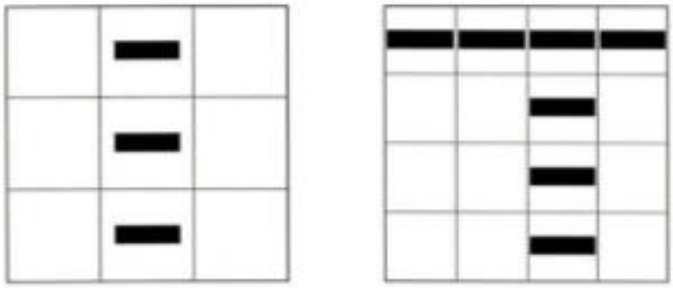
컨볼루전 신경망(Convolution Neural Network)이라 하는 기술은 전체적으로 앞서 서술한 Neural Network 를 따르는데 위 그림과 같이 이미지의 고유한 구조가 있다고 가정하고 사진을 벡터로 입력 받아 히든 레이어를 통해 가중치를 계산하여 Output 을 추측하게 되는 방식이다.

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 중요하다.

[13] 출처 www.semanticscholar.org/

Concepts



위의 그림을 보면 왼쪽을 filter, 오른쪽을 이미지라고 가정했을 때 이미지를 filter 에 대해 convolution 을 적용하여 자질을 추출할 수 있다. Convolution 의 정의는 아래와 같다.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$

거창해 보이는 식이지만 그냥 각각 pixel 의 dot product 를 구한 후 합을 취하는 것이다. 예를 들어 위의 그림의 Convolution 을 계산하면 $\begin{pmatrix} 1 & 3 \\ 0 & 3 \end{pmatrix}$ 이라는 결과가 나올 것이다.

이러한 feature 를 Neural Network 에 적용한 알고리즘을 Convolutional Neural Network 라고 하며 몇가지 Hidden layer 층으로 이루어져있고 기본적으로 3 가지의 다른 층을 가지고 있다.

- Convolution layer : 위에서 보인 것처럼 유의미한 자질을 추출하는 층이다.
- Pooling layer : 일반적으로 CNN 은 이미지에 적용된다. 이미지 특성상 픽셀의 갯수가 너무 많아 자질을 줄이기 위해 sub-sampling 하는 과정을 거치는데 이를 Pooling 이라고 한다.
- Feedforward layer : 위의 두 층에서 나온 자질들을 이용해서 분류를 할 때 사용하는 층이며 일반적인 Neural Network 처럼 행동한다.

Theory

모든 layer 에 대해 추측을 할 때는 forward propagation 모델, weight 을 학습할 때는 backpropagation 모델로 학습한다.

Neural Network 는 거대한 선형대수 모델을 연산하는 것이며 가장 기본적인 구조는 아래와 같다.

$$\mathbf{Ax + b = y}$$

여기서 A 는 Weight Parameter Matrix, b 는 bias 를 나타내며 x 가 데이터셋의 학습데이터 또는 은닉노드의 출력 값, y 가 결과 label 을 나타낸다.

Forward propagation

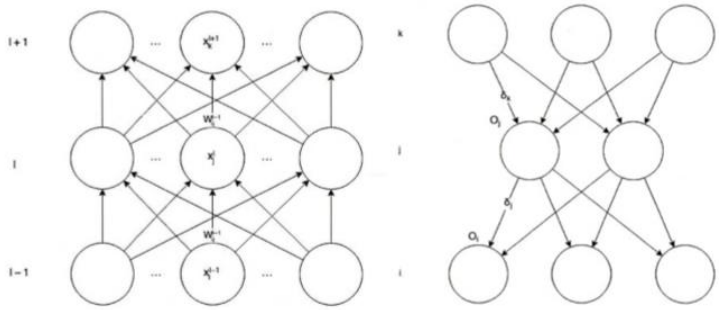
데이터를 처리할 때 학습 데이터 x 가 주어지면 미리 학습된 가중치 파라미터 A 를 통해 추측값 **y 를 계산하는 방향**이며 y 를 추측하여 계산한 것이므로 정확한 계산 데이터를 위해서는 학습된 가중치 파라미터가 검증된 값이어야 한다.

Back propagation

Back propagation 은 학습 데이터 x 와 label y 가 주어지면 forward propagation 을 하여 추측한 y'(error function)을 정의한 후에 이를 이용하여 A 를 업데이트하게 된다. 각 층을 반대방향으로 계산하기 때문에 역전파(back propagation)이라는 이름을 붙인다.

위에서 A 라고 단순하게 나타냈지만 실제로 layer 가 복합적으로 구성되므로 각 층별로 있는 weight 을 모두 업데이트 해야하며 이는 미분을 이용하여 해를 구하는 과정을 거쳐야한다.

대표적으로 gradient decent 알고리즘을 적용하여 모든 weight, bias 의 gradient 를 계산한다.



위 그림의 왼쪽은 화살표를 따라 가장 마지막의 노드를 계산하는 것이며(forward)

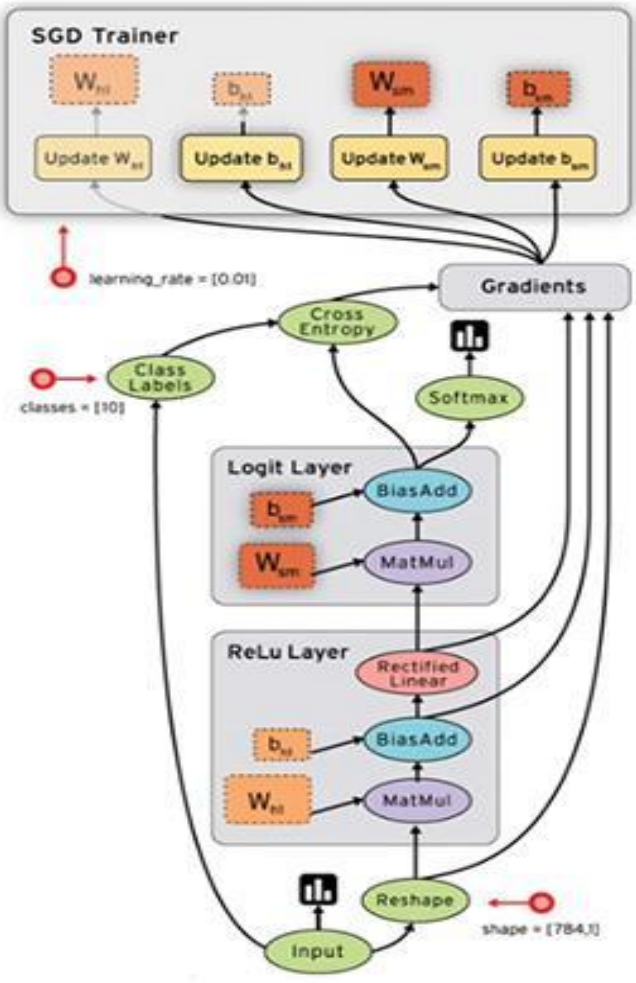
오른쪽은 결과(마지막 노드)로부터 화살표의 계수를 업데이트 하는 것이다(back)

TENSORFLOW

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

모바일 환경 및 64 비트 리눅스, OS X 데스크톱이나 서버 시스템의 CPU 와 GPU 를 사용할 수 있다. 윈도우 환경에서는 Docker 나 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.



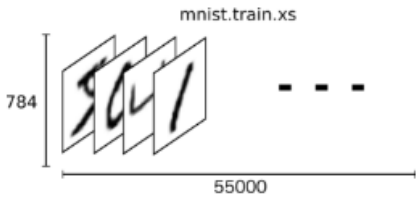
Install

OpenCV 에서도 마찬가지였지만 Python2.x 버전으로 통일한 상태이기 때문에 따로 Virtualenv 를 설정해 주지 않았다.

```
# Ubuntu/Linux 64-bit, CPU only
export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.11.0rc0-cp27-none-linux_x86_64.whl

# Ubuntu/Linux 64-bit, GPU enabled
# Requires CUDA toolkit 7.5 and CuDNN v5
export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.11.0rc0-cp27-none-linux_x86_64.whl

sudo -H pip install --upgrade $TF_BINARY_URL
```

Dataset

TensorFlow 에서 사용하는 Dataset 은 batch 형태로 전달한다. TensorFlow 의 MNIST 튜토리얼 데이터 셋을 들여다보면 데이터와 label 로 나누어지는데 아래와 같이 숫자를 (28, 28) Matrix 로 구성한 학습 데이터 55000 개를 묶어서 batch 형태로 만든다.



TensorFlow 에서는 위의 데이터를 100 개씩 묶어서 아래와 같이 전달한다.

```
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

위 코드로 batch 는 학습 데이터 xs 와 label ys 의 튜플 형태로 데이터를 전달하는 것을 알 수 있다. OpenCV 에서 xs 를 일관적으로 Resize 한 이미지를 만들고 구별하기위한 labeling 을 0 과 1 로 나타내는 벡터를 생성하여 만들 수 있었다.

(code 는 flask 프로젝트 내부의 static/uploads/backup 경로에 있다.)

tfsample.py 일부 함수화

```
def make_batch(sample, n):
    batchar = sample.reshape((1, 784))
    batchar = batchar.astype(np.float32)
    batchar = np.multiply(batchar, 1.0 / 255.0)

    batchlb = np.zeros((1,10))
    batchlb[0,n] = 1

    savbatch = (batchar, batchlb)

    return savbatch
```

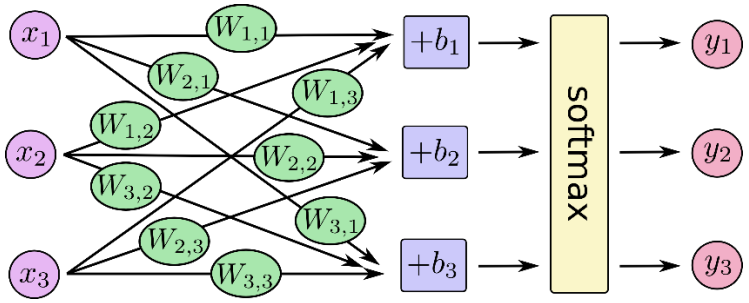
위와 같이 본 프로젝트에 적용하기 위해서 같은 (28, 28) Matrix 로 만들었으며 튜플 형태로 묶어 프레임마다 label 을 태그하여 Tensorflow 로 전달했다.

Softmax regressions

TensorFlow 를 이용하여 Convolution Neural Network 를 적용하기 위해서는 데이터를 판별하기 위한 회귀 함수가 필요하다. 역치를 가진 뉴런을 표현하기 위해 대표적인 함수가 Softmax 함수이다. Softmax 는 각각의 결과값의 편차를 확대시켜 큰 값은 상대적으로 더 크게, 작은값은 상대적으로 더 작게 만든다음 Normalization 시키는 함수이며 식은 다음과 같다.

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

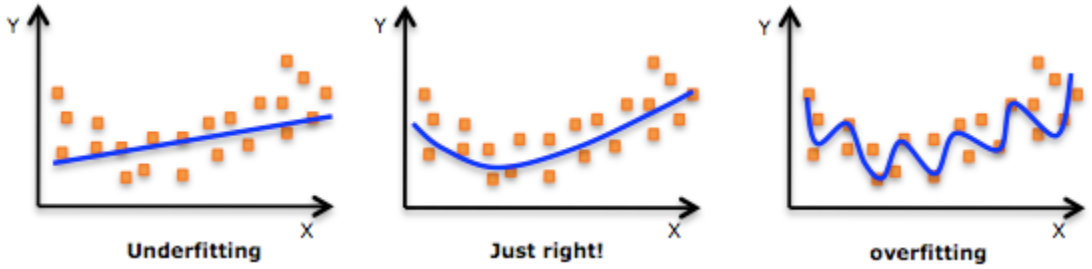
이 식을 아래와 같이 적용하여 Ax + b = y 의 모델을 만드는 것이다.

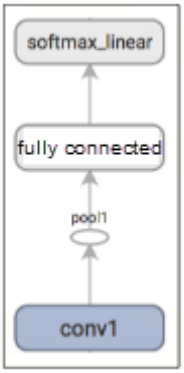


mymodel.py

여기까지가 하나의 뉴런을 이용하여 Neuron 을 구성한 것이다. 다수의 Neuron 으로 Neural Network 를 구성하기 위해서는 다중 Layer 를 통한 Model 구성이 필요하다.

앞서 Convolution Neural Network Concepts 에서 세가지 다른 Hidden Layer 층이 있다고 했다. 모델이 복잡할 수록 학습하는데 시간도 오래 걸릴 뿐더러 Overfitting 현상이 나타날 수 있으므로 한번의 Convolution 과 Polling 을 거친 모델을 만들어 보았다.





자세한 코드는 flask/static/uploads/backup/mymodel.py 참조.

MyTfModel 클래스에 위 모델이 구현되어 있으며 backpropa_train 함수로 데이터를 학습하고 feedforward 함수로 결과를 추측한다.

LIMITS OF PROJECT

Detection Algorithm Dependence

TensorFlow 는 이미지 프로세싱을 하지 않는다. 두 이미지 간의 유사도를 측정할 뿐이다. OpenCV 의 Detection 알고리즘에 대한 의존도가 매우 큰 것이 문제였다.

앞서 언급한 바와 같이 이미지 프로세싱을 Detection, Tracking, Recognition 세가지로 나누었다. Tracking 과 Recognition 도 Detection 알고리즘의 영향을 매우 많이 받는데, TensorFlow 역시 이미지에서 얼굴 검출이 되어야 해당 얼굴에 대한 유사도 측정을 할 수 있었다.

이에 비해 Haar Cascade 알고리즘은 밝기 차를 비교하여 얼굴을 검출해 내는 속도에 초점을 맞추었기 때문에 강력하다고는 할 수 없는 알고리즘이다.

또한 Convolution 은 각 픽셀의 밝기에 대한 특징을 추출하는 것이다.

Haar Cascade 알고리즘을 거친 Convolution 은 다음에 추측하기위한 Forward propagation 알고리즘을 적용할 때도 얼굴 검출이 되어야 하므로 거의 비슷한 특징이 나올 수 밖에 없어 다른 얼굴 사진이더라도 99% 이상의 유사도가 계속 나올 수 밖에 없었다.

Real time Streaming

학습 데이터가 전혀 없는 머신은 갓 태어난 아이의 상태와 같다.

실시간 스트리밍에서의 학습을 초점으로 했기 때문에 학습 데이터의 수가 현저히 부족한 것이 문제였다. 기본적으로 비슷한 사람을 골라서 구별하는 것에 목표를 두었지만 유일성이 보장되는 해당 데이터가 10 억개 이상이 있어야 99% 이상의 정확도를 낸다고 한다.

스스로 데이터를 증폭시켜 학습을 할 수도 있지만 실시간 스트리밍에서의 해당작업은 이미지 프로세싱을 포함하여 부하가 너무 큰 작업이었다.

Conclusion

AWS 부터 Flask, OpenCV, TensorFlow 까지 많은 분야를 공부했지만 많은 것들을 소화하기 위해 최대한 프로젝트를 완성하려고 하다가 제대로 된 결과를 얻기위한 시간이 부족했던 것 같다.

배경지식이 부족해서 충분한 테스트를 해 보지 않고 프로젝트를 하다가 실현가능성에 문제가 생겨서 방향이 바뀌었다.

졸업작품은 작품이기 때문에 완성품을 만들어야 한다는 생각 때문에 두마리 토끼를 놓친 것이다.

하지만 위와 같은 경험들로 별로 프로젝트를 해보지 못한 자신에게 큰 경험이 되었고 다양한 분야를 빠르게 배우면서 성장할 수 있는 계기가 되었다.

또한 수학과 컴퓨터 공학을 복수전공하면서 수학이 컴퓨터 공학 분야에 매우 많이 쓰이게 된다는 것을 알게 되었다.

인공지능 분야를 공부하면서 수학을 배울 때와 약간 달랐던 점은 이렇게 모델을 세웠을 경우 최고의 결과를 낸다는 귀납적인 결과는 있지만 어떻게 모델을 세워야 하는지에 대한 방향은 제시해 주지 않아 한계를 느꼈다.

아직 사람의 일을 온전히 기계로 대체하려면 많은 과제가 남았음을 이 프로젝트를 통해 알게 되었다.

Schedule

Schedule	0	1	2	3	4	5	6	7	8
설치 및 환경 구축 (GPU)									
OpenCV 알고리즘 적용 및 구현									
TensorFlow 이미지 학습 및 구현									
Frontend Design									
Debugging 및 테스트									

Reference

WEB SITES

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 – TensorFlow 를 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=K0MTYnlNNnc – 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

http://www.w3schools.com/ - Frontend design

http://bootstrapk.com/ - Bootstrap Homepage

BOOKS

Machine Learning to Deep Learning 이론편 – Deepcumen AI Research Group 곽동민 박세원 이한남

Python Cookbook 3rd edition – O'Relly

Flask Web Development – O'Relly

OpenCV Computer Vision with Python – Joseph Howse

My Source Code : https://github.com/lastone9182/flask