

작성자
박상준
이명진

문서관리자
박상준

검토
박상준
이명진

문서종류
캡스톤 디자인 결과보고서

제 목

개인정보 유포 방지를 위한 얼굴식별 기반 데이터 크롤링, 비식별화 및 신고서비스

캡스톤 디자인 결과보고서

“개인정보 유포 방지를 위한 얼굴식별 기반 데이터 크롤링, 비식별화 및 신고서비스”

컴퓨터정보통신공학과
19 박상준
191370 이명진

2023년 12월 12일

목 차

1. 서론

- 1.1 기술 개발에 대한 사회적, 기술적 필요성
- 1.2 기술 개발의 개요, 범위

2. 관련 연구 배경 및 이론

- 2.1 배경지식
- 2.2 관련 연구

3. 시스템 설계 및 구현

- 3.1 사용자 요구사항
- 3.2 시스템 설계의 목표
- 3.3 시스템 설계
- 3.4 구현

4. 실험 및 분석

- 4.1 실험 환경
- 4.2 실험 결과 및 분석

5. 결론 및 향후 과제

- 5.1 연구 과제 기여
- 5.2 향후 연구 및 개발 과제

6. 참고문헌

1. 서론

1.1 기술 개발에 대한 사회적, 기술적 필요성

사람들은 SNS와 커뮤니티 등을 일상적으로 사용한다. 자주 볼 수 없는 사람들과 소통 할 수 있게 하는 창구가 되기도 하지만 그만큼 어두운 면 또한 가지고 있다. SNS와 커뮤니티를 사용함에 있어 사람들은 의도적이든 그렇지 않은 개인정보를 노출하고 있다. 그 정보들이 개인 정보가 포함된 사진이나 동영상의 형태로 무단으로 유포되는 경우가 적지 않다. 이러한 불법 유포된 정보들로 인해 피해자는 프라이버시 침해뿐만 아니라, 사회적, 심리적 피해를 겪는다. 이처럼 오늘날 디지털 시대에서 개인정보의 보호는 매우 중요한 이슈로 자리 잡았다. 특히, 사진이나 동영상 속 개인의 얼굴은 개인을 식별할 수 있는 중요한 정보이다. 본 연구는 이러한 문제를 식별하고 방지할 수 있는 기술이 필요하다는 생각에서부터 시작되었다.

위 기술이 필요한 이유로는 불법 유포의 피해자들이 상황을 인지하기 전에 이미 정보가 공공연하게 퍼져있을 가능성이 높으며, 그 정보들이 어디에 있는지 알 수 없다는 점이다. 수 많은 경로가 있고 광범위하기 때문에 개인이 찾기에는 어려움이 많다.

이러한 문제를 해결하기 위해 본 프로젝트는 인공지능을 활용한 얼굴식별 기반의 데이터 크롤링 및 얼굴 검출 서비스를 제공한다. 이 기술은 불법적으로 유포되는 개인의 이미지나 동영상을 신속하게 탐지하고 이를 통해 개인정보의 보호 및 사이버 범죄 예방에 기여할 수 있다.

1.2 기술 개발의 개요, 범위

본 프로젝트의 구현은 디시인사이드와 같은 온라인 커뮤니티에서 이미지와 동영상을 크롤링하는 것으로 시작한다. 이 플랫폼이 선택된 주된 이유는 일반적인 SNS 플랫폼에서 크롤링을 시도할 경우 차단 문제가 자주 발생하며, 짧은 시간 안에 효과적인 크롤링 코드를 개발하기 어렵다는 현실적인 제약 때문이다. 반면, 디시인사이드는 이러한 제약이 상대적으로 적어 크롤링 작업이 더 용이했다. 구현한 크롤링 방식은 기본적으로 다른 플랫폼에도 적용 가능하므로, 추후에 보다 완성도 높은 서비스로 발전시킬 때 이 경험과 과정을 그대로 적용하기만 하면 된다.

크롤링된 데이터는 Google Cloud Storage에 안전하게 저장되며, 각 파일의 메타데이터는 Google BigQuery에 기록된다. 이러한 클라우드 기반 기술의 활용은 대용량 데이터 처리와 관리에 있어서 빠르고 안정적인 처리를 가능하다. 이어서, 로컬로 다운로드 된 이미지와 동영상에서는 얼굴 탐지 기술을 사용하여 Facenet을 추출한다. 이는 고급 얼굴 인식 알고리즘을 통해 이루어지며, 사용자가 제공하는 사진의 Facenet과 크롤링된 데이터의 Facenet을 비교하여 유클리드 거리를 계산함으로써 유사한 얼굴을 식별한다.



<요약>

2. 관련 연구 배경 및 이론

2.1 배경 지식

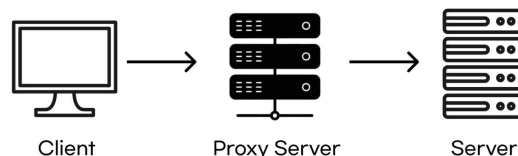
2.1.1 크롤링(Web Crawling)

인터넷 상의 웹 페이지들을 방문하여 데이터를 추출하는 자동화된 방법이다. 이 과정은 보통 웹 크롤러 또는 스파이더라 불리는 스크립트나 프로그램을 사용하여 수행하게 된다. HTML(웹 페이지 구조)를 파싱할 수 있는 라이브러리를 활용하여 크롤링을 진행한다.

(EX BeautifulSoup, Selenium 등)

2.1.2 IP 차단

웹사이트가 자동화된 스크립트의 접근을 막기 위해(서버에 가해지는 부하를 막기 위해) 특정 IP 주소 또는 IP 범위를 차단하는 행위이다.



2.1.3 프록시(Proxy)

IP 차단을 회피하기 위해 IP주소를 변경하거나 회전시키는 방법으로 클라이언트가 자신을 통해서 다른 네트워크 서비스에 간접적으로 접속할 수 있게 해주는 응용프로그램이다.

2.1.4 클라우드(Severless) 컴퓨팅

기업이 직접 데이터센터와 서버와 네트워크 스토리지 등의 인프라를 구매하여 구축하고 운영하는 온프레미스 환경과는 다르게, 하드웨어나 소프트웨어 등을 직접 구축하지 않고 원하는 만큼 자원을 받아 사용한 만큼 비용을 지불하는 서비스이다. 원격 서버 네트워크를 통한 인터넷을 통해 다양한 하드웨어 및 소프트웨어 서비스를 제공한다. 사용자 요구에 맞게 어디서나 클라우드 서비스 이용 가능하다는 유연성이 있고, 유지 보수에 대한 걱정 없이 빠른 배포와 높은 보안성이 장점이다.

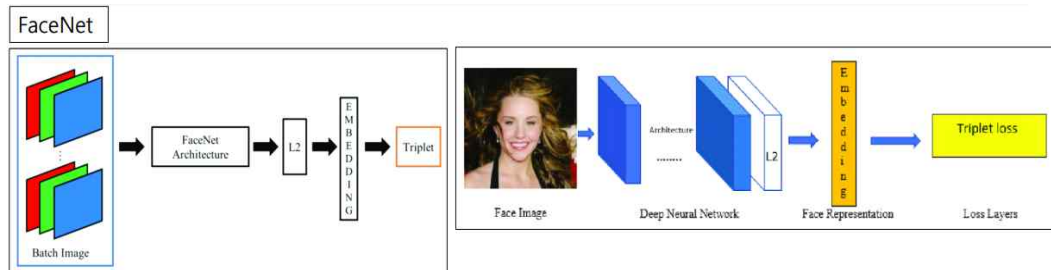


2.1.5 얼굴 탐지 및 인식(Face Detection & Recognition)

얼굴 탐지 및 인식은 컴퓨터 비전의 한 분야로, 디지털 이미지나 비디오에서 인간의 얼굴을 자동으로 식별하는 기술이다. 이 기술은 보안 시스템, 휴대폰 잠금 해제, 소셜 미디어 등 다양한 분야에서 활용되고 있다.

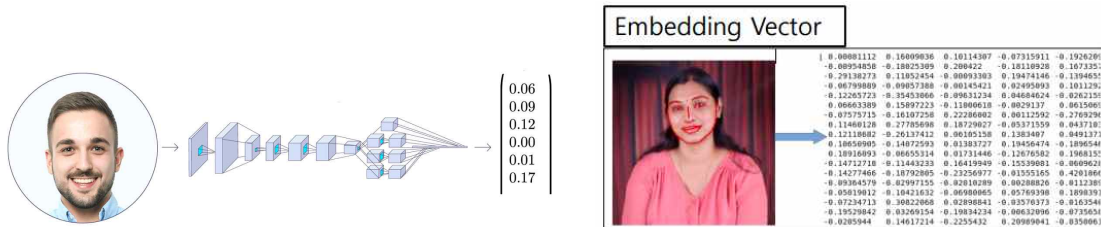
2.1.6 FaceNet

FaceNet은 Google에서 개발한 얼굴 인식 시스템으로, 임베딩 벡터(Embedding Vector)를 직접 학습하는 방식을 사용한다. 이를 통해 모델은 학습 데이터에 없는 새로운 인물에 대해서도 잘 분류할 수 있다. FaceNet은 얼굴 인식에서 높은 정확도를 보여준다.



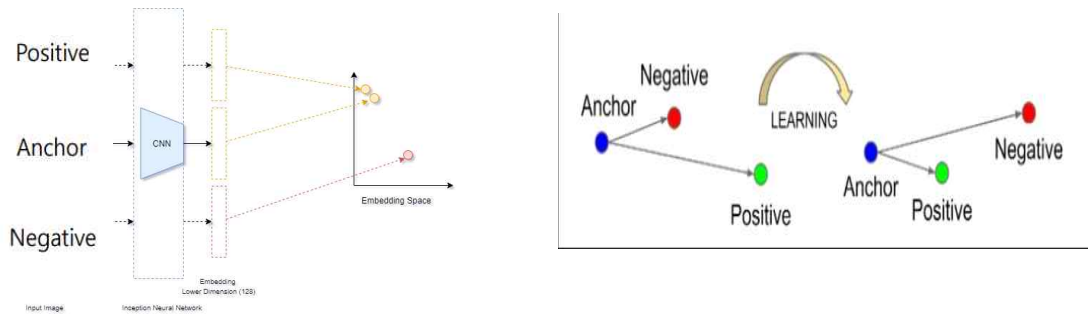
2.1.7 임베딩 벡터(Embedding Vector)

얼굴 인식에서 임베딩 벡터는 얼굴 이미지를 저차원 공간으로 변환하여 특정 인물의 특징을 잘 나타내는 벡터이다. 이 벡터를 통해 특정 인물을 식별하거나 다른 인물과의 유사성을 측정하는 등의 작업을 수행할 수 있다.



2.1.8 Triplet Loss

Triplet Loss는 FaceNet에서 사용하는 학습 방법으로, Anchor(기준), Positive(동일인), Negative(타인) 세 개의 이미지를 사용한다. 학습을 진행할수록 Anchor와 Positive는 가까워지고, Anchor와 Negative는 멀어지게 됩니다. 이는 유클리드 거리를 기반으로 계산되며, 유클리드 거리가 가까울수록 동일인이라고 판단할 수 있다.



2.1.9 유클리드 거리(Euclidean Distance)

유클리드 거리는 두 점 사이의 가장 직선거리를 의미한다. 이는 유클리드 공간에서 가장 기본적인 거리 측정 방법으로, 2차원 또는 3차원 공간에서 두 점 사이의 거리를 계산할 때 주로 사용된다.

2.2 관련 연구

1. 서울시 성범죄 예방 AI 기술

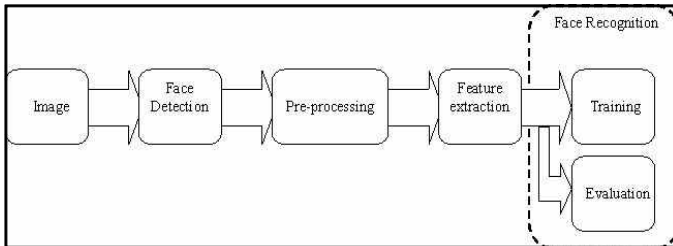
서울시, 디지털성범죄 영상물 자동검색·삭제하는 AI 기술 지자체 최초 개발

서울기술연구원은 디지털 성범죄 영상물의 재유포를 방지하고, 피해자의 고통을 최소화하기 위해 AI 기반의 영상 추적 및 삭제 기술 개발에 착수했다. 이 기술은 성범죄 피해 영상물을 정확하게 식별하고, 삭제 요청까지 원스톱으로 지원하는 방식으로 개발되고 있는 중이다.

기존의 안면매칭 기법과 더불어 움직임 패턴, 오디오 주파수, 대화 내용 등을 포함한 복합적 분석을 통해 정확도를 향상하는 접근 방식을 채택했다. 머신러닝과 딥러닝 기술을 결합하여 비디오, 오디오, 텍스트 데이터를 융합적으로 분석하고, 이를 통해 검색 정확도를 최고 수준으로 끌어올린다. 서울 디지털 성범죄 안심 지원센터에서는 AI 기술을 활용하여 불법 촬영물의 재유포를 차단하고, 신속하게 대응하기 위한 시스템을 구축하고 있다.

현재 이미지와 동영상을 통해 얼굴을 검출해내는 본 연구와는 다르게 비전과 오디오까지 함께 고려하는 아키텍처이기 때문에 굉장히 높은 성능을 낼 것으로 보인다.

2. 기존의 얼굴 인식 알고리즘은 대체로 각 인물을 하나의 카테고리로 삼아 이미지 분류 모델을 학습시키는 방식을 사용하였다. 이러한 방식은 각 인물에 대한 충분한 수의 사진을 수집하여 Classifier를 학습시키는 것이 필요하며, 이는 데이터 수집과 관리에 많은 노력이 필요하다는 단점이 있다.



이에 더해, 이 방식의 또 다른 큰 단점은 학습 데이터셋에 없는 새로운 인물이 등장했을 때, 그 인물을 효과적으로 분류하기 어렵다는 점이다. 새로운 인물을 인식하려면 해당 인물의 데이터를 추가로 수집하고 모델을 다시 학습시켜야 하는 문제가 있다.

반면, FaceNet은 이러한 문제를 극복한 모델이다. FaceNet은 새로운 인물에 대한 추가적인 학습 없이도 효과적으로 얼굴을 인식할 수 있다. 이는 FaceNet이 임베딩 벡터를 직접 학습함으로써 가능하며, 이 임베딩 벡터는 각 얼굴의 고유한 특징을 효과적으로 포착한다. 이렇게 학습된 임베딩 벡터는 유클리드 거리를 이용하여 얼굴 간의 유사성을 측정하므로, 학습 데이터셋에 없는 새로운 인물의 얼굴 이미지도 효과적으로 분류할 수 있다.

FaceNet은 얼굴 이미지를 128차원의 임베딩 벡터로 변환하는 것을 목표로 한다. 이 임베딩 벡터는 이미지 내의 얼굴을 나타내며, 유사한 얼굴은 유사한 벡터를 생성한다. 이 기술은 LFW(Labeled Faces in the Wild) 데이터셋에서 99.63%의 정확도를 보여주었으며, 현재도 많은 얼굴 인식 시스템에서 기반 기술로 사용되고 있다.

하지만, FaceNet은 주로 서양인의 얼굴에 대한 데이터를 학습하였기 때문에, 동양인 얼굴에 대한 인식률이 상대적으로 떨어질 수 있다. 이를 해결하기 위해, 동양인 얼굴에 대한 데이터를 추가로 학습시키는 연구가 필요하다.

3. 시스템 설계 및 구현

3.1 사용자 요구사항

3.1.1 웹사이트(디시인사이드)에서 자동 크롤링 기능

- 원하는 갤러리에서 자동으로 이미지와 동영상을 크롤링 할 수 있는 기능
- 사용자가 디시인사이드의 모든 갤러리에서 크롤링을 수행할 수 있도록 구현

3.1.2 크롤링 된 데이터의 메타데이터 추출 및 저장 기능

- 크롤링 된 각 파일에 대한 메타데이터(파일 이름, 게시글 URL, 날짜 등)를 정확하게 추출
- 추출된 메타데이터는 데이터베이스 또는 적절한 데이터 저장소에 안전하게 저장
- 추출된 데이터를 원하는 조건을 담은 쿼리문으로 원하는 정보만 탐색 및 저장

3.1.3 스케줄링 및 자동화 기능

- 크롤링 작업을 정기적으로 또는 특정 조건에서 자동으로 시작할 수 있는 스케줄링 기능
(EX. 매일 특정한 시간 혹은 특정 간격 시간)

3.1.4 정확한 얼굴 탐지

- 크롤링된 이미지나 비디오 데이터에서 얼굴을 정확하게 탐지하는 기능.
- 얼굴 탐지는 다양한 상황과 조건에서도 얼굴을 정확하게 탐지하는 기능.

3.1.5 신뢰성 있는 얼굴 인식

- 사용자가 제공한 얼굴 사진과 크롤링 된 얼굴 데이터를 비교하여 동일 인물 여부를 판단할 수 있는 기능

3.2 시스템 설계의 목표

●기능 목표.

1. 웹 크롤링 모듈

- 디시인사이드와 같은 특정 웹사이트에서 이미지와 동영상을 자동으로 추출하는 것.
- 다양한 종류의 디시인사이드 갤러리 구조를 인식하고 처리할 수 있는 유연성
- 이미지와 동영상 링크를 정확하게 식별하고 추출하는 기능

2. 데이터 처리 및 저장 모듈

- 크롤링 된 데이터와 메타데이터를 효율적으로 처리하고, 이를 Google Cloud Storage와 BigQuery에 저장
- 크롤링 된 데이터의 형식을 표준화하고 필요한 정보를 추출하는 데이터 파싱
- 메타데이터를 포함한 정보의 구조화 및 BigQuery 데이터베이스에의 삽입 및 조회

3. 고정밀 얼굴 탐지 알고리즘

- 크롤링된 이미지나 비디오 데이터에서 얼굴을 정확하게 탐지할 수 있는 고정밀의 얼굴 탐지 알고리즘 사용
- 다양한 상황에서도 얼굴을 신속하게 탐지할 수 있도록 최신 기술의 라이브러리를 활용

4. 신뢰성 있는 얼굴 인식 기능

- 사용자가 제공한 얼굴 사진과 크롤링 된 얼굴 데이터를 비교하여 동일 인물인지 아닌지를 정확하게 판단할 수 있는 신뢰성 있는 얼굴 인식 기능을 개발
- 고성능 얼굴 인식 알고리즘인 FaceNet을 사용하고, 128차원의 고유 임베딩 벡터를 통해 얼굴 이미지 간의 유사성을 표현

●성능 목표

1. 빠른 크롤링 속도

- requests 라이브러리를 통해 웹 페이지의 HTML 콘텐츠를 효율적으로 로드
- BeautifulSoup을 사용하여 HTML을 파싱하고 가장 필요한 최적화된 선택자를 사용하여 데이터 추출 속도를 향상
- 다양한 최적화 방법을 실행해봄으로써 크롤링 속도 향상

2. 높은 데이터 정확도

- 추출된 데이터의 정확성을 보장하기 위해, URL 형식의 정확성등을 검사는 로직 구현
- 로깅 시스템을 통해 오류를 기록함으로써 오류 감지 및 수정 매커니즘 구현
- 게시글에 있는 모든 데이터 빠짐없이 크롤링

3. 시스템 안정성

- 크롤링 된 정보와 메타데이터가 Google Cloud Storage와 BigQuery에 충돌 없이 안전하게 저장

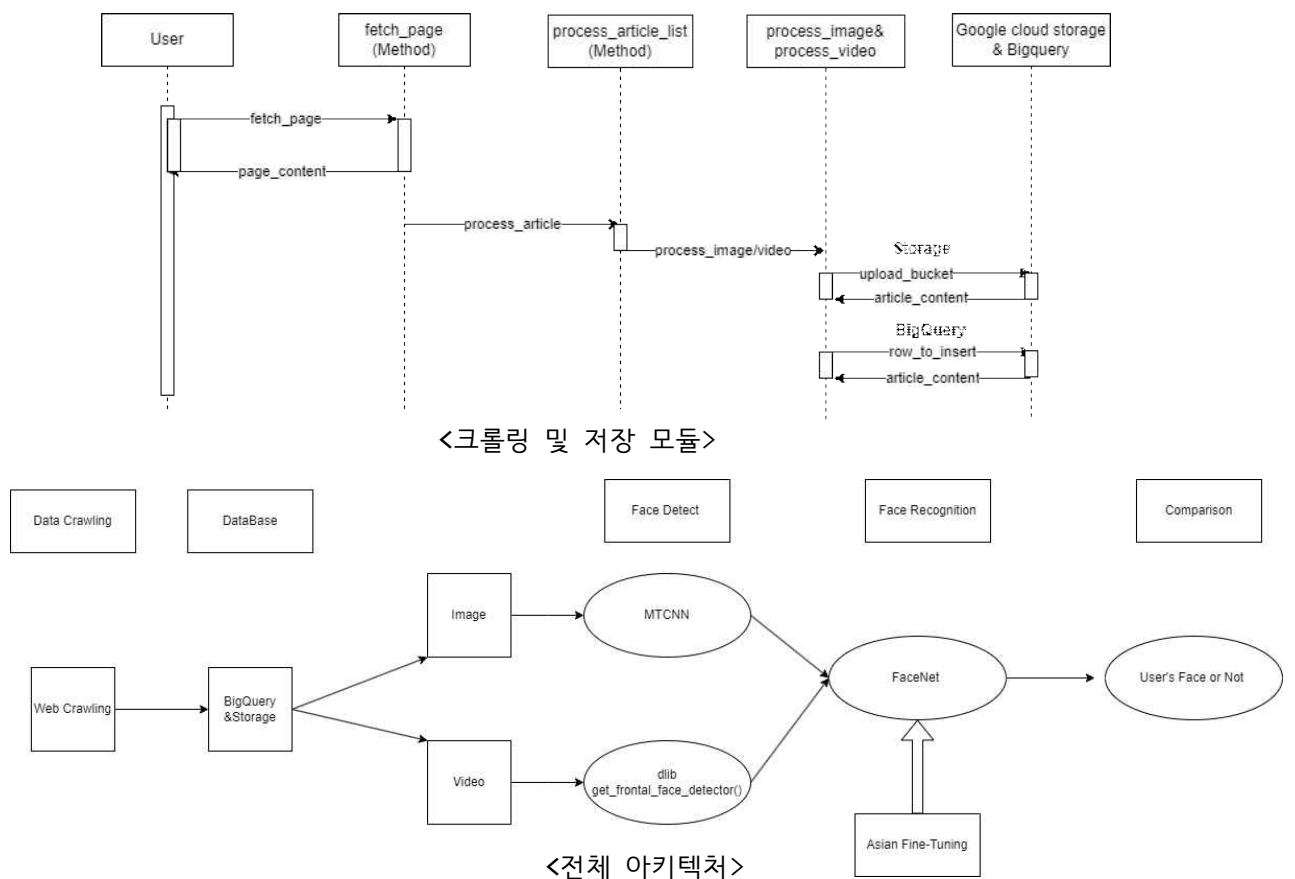
4. 고성능 얼굴 탐지

- 얼굴 탐지의 정확도를 극대화하여 얼굴 탐지율은 약 95% 이상을 목표

5. 얼굴 인식률 향상

- FaceNet은 주로 서양인 얼굴에 대한 데이터를 학습하였기에, 동양인 얼굴에 대한 인식률이 상대적으로 떨어질 수 있다
- 이를 개선하기 위해 동양인 얼굴에 대한 데이터를 추가로 학습시키는 Fine-Tuning 방법을 활용하여 얼굴 인식률을 향상

3.3 시스템 설계



3.4 구현

3.4.1 구현 환경

- 운영체제
: Window11, Linux, Debian 11
- 개발 도구
: Visual Studio Code, PyCharm, Jupyter Lab, Google BigQuery, Google Storage, TensorFlow Enterprise 2.11
- 프로그래밍 언어 : Python (3.7.11)

```
import requests
from urllib import request
from bs4 import BeautifulSoup
import time
from datetime import datetime
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager
import os
from google.cloud import storage
from google.cloud import bigquery
from selenium.common.exceptions import NoSuchElementException
from urllib.parse import urlparse, parse_qs
```

- requests: 웹 페이지의 HTML 콘텐츠를 서버로부터 요청하고 가져오는 데 사용.
- BeautifulSoup: HTML 문서를 파싱하고 데이터를 추출하는 데 사용.
- selenium: 웹 브라우저 자동화를 위해 사용. 동적 웹페이지에서 동영상 다운로드 링크를 추출하는 데 사용됨.
- webdriver-manager: Selenium 웹드라이버의 종속성 관리 및 자동 업데이트를 위해 사용.
- Google Cloud Storage: 크롤링된 데이터를 저장하는 클라우드 스토리지 서비스.
- Google BigQuery: 크롤링된 데이터의 메타데이터를 저장하고 관리하는 빅데이터 웨어하우스 서비스.
- logging: 로그 관리 및 오류 추적을 위해 사용.
- urllib: URL 검증에 사용.

3.4.2 구현 결과

1. 크롤링 시 IP 우회를 위한 Proxy

```
# 헤더 설정
headers = [
    {'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36'},
]
proxies={
    'http': 'socks5://127.0.0.1:9050',
    'https': 'socks5://127.0.0.1:9050',
}
```

(설정)

```
response = session.get(url, headers=headers[0], proxies=proxies, timeout=10)
```

(실제 사용 예시)

User-Agent 헤더를 사용하여 요청을 보내는 클라이언트 정보를 설정했다. 이는 웹 서버로부터 오는 응답을 브라우저처럼 처리하기 위한 것으로 Chrome 브라우저의 일부로 설정했다. 또한 IP차단 문제를 해결하기 위해 127.0.0.1의 9050포트를 통해 사전에 설치함 SOCKS5 프록시를 사용하도록 설정했다. 이로써 IP우회와 익명성을 갖추게 되었다. 실제로 사용할 때는 header와 proxies 매개변수를 사용하여 크롤링을 진행한

2. URL 검증 및 세션 관리

```
## URL 형식을 검증하는 함수
def is_valid_url(url):

    parsed = urlparse(url)
    return bool(parsed.netloc) and bool(parsed.scheme)

session = requests.Session()
response = session.get(url, headers=headers[0], proxies=proxies, timeout=10)
```

사용자로부터 입력받은 URL의 유효성 검사 및 HTTP 세션을 통한 안정적이고 빠른 웹페이지 접근을 시도한다. URL 검증을 통해 유효하지 않은 웹 주소에서 발생할 수 있는 오류를 방지하며 크롤링 속도를 향상하기 위해 실험을 통해 가장 빠른 방법인 HTTP세션을 이용한 방법으로 구현하였다.

3. 에러 처리 및 로깅

```
# 로깅 설정
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

로깅 설정을 통한 기본적인 정보를 기록하였다. 로깅 기능을 통해 스크립트의 기본적인 실행 정보를 기록, 문제 발생 시 기본적인 추적이 가능하게 하였다.

4. 크롤링 코드

번호	말머리	제목	글쓴이	작성일	조회	추천
2796	설문	🟢 끝까지 잘 살 것 같았는데 이혼해서 충격 준 스타는?	운영자	23/12/11	-	-
4711232	공지	🔴 최근 일들에 관한 공지 (케인, 파리, 중계) [46]	담쥐	23.11.06	8197	84
4767797	공지	🔴 토트넘 핫스파 마이너 갤러리 달력 (12.04 ver) [8]	AngePostecoglou	23.11.24	3817	31
4225240	공지	🔴 토트넘 핫스파 마이너 갤러리 통합공지 [1]	담쥐	23.06.08	12279	11
4019978	공지	🔴 완장소환기 [8]	ㅇㅇ (221.143)	23.03.24	38241	72
4829925	일반	🟡 클집 애는 미드필더 땡어 올티가 자기 자리 같음 [1]	ㅇㅇ	17:19	14	0
4829924	짤&S	🟢 손흥민 MOM 트로피 들고 인증샷 찍은 로메로.jpg	ㅇㅇ (115.138)	17:18	87	6
4829923	일반	🟢 소신발언) 매디슨 없으면 손흥민 땡이 맞는 듯 [8]	ㅇㅇ (175.193)	17:18	56	0
4829922	일반	🟢 매디슨 박싱데이 전에는 올? [4]	ㅇㅇ	17:16	77	0
4829921	일반	🟡 아라우호랑 로메로 스왈딜 하면 찬성임? [1]	ㅇㅇ (122.36)	17:15	39	0
4829920	일반	🟢 손흥민의 쥘 축구스승이 아버지임? [2]	찰옥수수	17:15	55	0
4829919	일반	🟢 아직 사랑하면 개추	행신포스틱	17:14	33	0
4829918	일반	🟢 토트넘 올타임 베스트 11	ㅇㅇ (112.164)	17:13	96	0
4829917	인터뷰	🟡 [손흥민] "탈장 경험으로 허설에게 빠른 수술 권유" [5]	ㅇㅇ (115.138)	17:13	427	21
4829916	일반	🟡 진짜 클루셉이 중원 개새끼들만 맡아주면 강한팀인데 [3]	ㅇㅇ (121.155)	17:10	97	0
4829915	일반	🟡 투헬은 거울에 아라우호 원하는 중이라고함 ㅋㅋㅋㅋㅋ [3]	ㅇㅇ	17:10	117	0
4829914	일반	🟢 seg에이전트 "텐하프 임기동안 많은 선수들을 맨유로... [1]	ㅇㅇ	17:09	111	6
4829913	일반	🟡 고드프리 게이 요즘 주전도 못 꿰찰 정도로 폼 안출면...	Oliver_Skipp	17:08	36	0
4829912	일반	🟢 일본사는일본에서점가서손흥채우고왔다 [1]	ㅇㅇ (1.235)	17:08	85	3
4829911	일반	🟢 포체티노 "거울에 영입 더 필요할 수도" [4]	ㅇㅇ	17:07	188	0
4829910	일반	🟡 사르는 박스만 가면 축구력 씹하락 하는 게 문제임	ㅇㅇ	17:07	39	0
4829909	일반	🟢 고드프리 유명한 반칙 두 장면.gif [1]	ㅇㅇ	17:07	122	4

위 페이지는 디시 인사이드 커뮤니티의 특정 갤러리 화면이다. 사용자에게 제공받는 크롤링 대상 URL은 위 화면 형태여야만 한다. 우리는 위 페이지의 각 게시글에 대해서 연속적으로 크롤링을 하는 코드를 구현하였다. 각 게시글 목록을 읽어나가면서 게시글 옆에 있는 icon을 통해 이미지인지 동영상인지를 확인한다.



이미지



동영상

```
def process_article_list(article_list):
    """ 게시글 목록을 처리하는 함수 """
    for tr_item in article_list:
        if tr_item.find('em', class_='icon_img icon_pic') is not None:
            process_image_post(tr_item)
        elif tr_item.find('em', class_='icon_img icon_movie') is not None:
            process_video_post(tr_item)
```

tr_item중 class가 icon_img icon_pic를 포함하고 있다면 그 게시글은 이미지를 담고 있기 때문에 이미지를 크롤링하는 코드를 실행시킨다. 그와 반대로 icon_img icon_movie인 클래스를 포함하고 있다면 동영상을 크롤링하는 코드를 실행시킨다.

4.1. 이미지 크롤링

```
def process_image_post(tr_item):
    print('+*30)
    print("이미지 크롤링")
    #print(tr_item)
    date_tag = tr_item.find(class_='gall_date')
    date = date_tag.get('title')
    date = date.split(' ')[0]
    print("게시 날짜: ", date)
    # 제목 추출
    title_tag = tr_item.find('a', href=True)
    title = title_tag.text

    print("제목: ", title)
    print("주소: ", title_tag['href'])

    # 이미지가 있는 게시물에 request
    arurl = ARTICLE_BASE_URL + title_tag['href']
    article_response = requests.get(arurl, headers=headers[0], proxies=proxies)
    print("url: ", article_response.url)
    article_id = (title_tag['href'].split('no=')[1]).split('&')[0]
    print("게시물 ID : ", article_id)

    # URL 검증 로직 추가
    article_url = ARTICLE_BASE_URL + title_tag['href']
    if not is_valid_url(article_url):
        logger.error(f"Invalid URL: {article_url}")
        return # 함수 종료

    article_content = fetch_page(article_url)
    if article_content is None:
        return # 함수 종료

    article_soup = BeautifulSoup(article_response.content, 'html.parser')
    # 게시물 부분의 태그
    article_contents = article_soup.find('div', class_='writing_view_box').find_all('div')
    # 아래 이미지 다운로드 받는 곳에서 시작
    image_download_contents = article_soup.find('div', class_='appending_file_box').find('ul').find_all('li')
```

이미지 크롤링 코드를 실행시키면 맨 위 게시글부터 크롤링을 실행하게 된다. 해당 게시글의 URL이 유효한지 확인 후 게시 날짜, 제목, 주소를 추출한 후 해당 게시 글로 페이지를 이동하게 된다.

원본 첨부파일 1

033A6DA6-E219-400E-8B07-BA8B0B7EBA97.jpg

이동한 페이지에서 BeautifulSoup 라이브러리를 사용하여 HTML 문서를 파싱하고, 위 그림에 해당하는 이미지 파일의 소스URL을 찾아낸다.


```
for i, li in enumerate(image_download_contents):
    img_tag = li.find('a', href=True)
    img_url = img_tag['href']
    print("url : "+img_url)
    file_ext = img_url.split('.')[-1]
    savename = os.path.join(image_dir, f"image_{article_id}_{i}." + file_ext)
    opener = request.build_opener()
    opener.addheaders = [(('User-agent', 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
    Gecko) Chrome/119.0.0.0 Safari/537.36'), ('Referer', article_response.url))]
    request.install_opener(opener)
    request.urlretrieve(img_url, savename)
```

첨부파일에 해당하는 모든 이미지 파일을 다운받기 위한 반복문을 통해 이미지의 URL을 request라이브러리를 통해 원래 파일명 그대로 로컬로 다운하게 된다.

```
bucket_name = "capstone_image"
upload_to_bucket(f"image_{article_id}_{i}." + file_ext, savename, bucket_name)

# 빅쿼리에 저장
client = bigquery.Client()
table_id = "strange-analog-405708.CrawlingDB.image"
bucket_path = f"gs://{bucket_name}/"
file_path = bucket_path + f"image_{article_id}_{i}.{file_ext}"

rows_to_insert = [
    {"img_file_name": f"image_{article_id}_{i}.{file_ext}", "post_url": article_response.url, "post_date":
    datetime.strptime(date, "%Y-%m-%d").isoformat(), "file_path": file_path},
]
errors = client.insert_rows_json(table_id, rows_to_insert)
if errors == []:
    print("DB에 행추가")
else:
    print("에러발생: {}".format(errors))

print("저장한 URL : "+img_url)

#0.5초 동안 대기
time.sleep(0.5)
```

저장된 이미지는 구글 클라우드 스토리지의 특정 버킷으로 업로드된다. 이 과정은 Google Cloud Storage의 Python SDK를 사용한다. 데이터의 원활한 관리를 위한 DB를 생성하기 위해 각 이미지에 대한 정보(이미지 파일 이름, 원본 게시물 URL, 게시 날짜, 스토리지내 파일 경로)를 Google BigQuery에 저장한다. 마찬가지로 이 과정에서도 Google Cloud BigQuery의 Python SDK를 사용한다.

```
*****
이미지 크롤링
게시 날짜: 2023-12-11
제목: 내가 울프 미담 하나 찾아왔다
주소: /board/view/?id=leagueoflegends5&no=752878&page=1
url: https://gall.dcinside.com/board/view/?id=leagueoflegends5&no=752878&page=1
게시물 ID : 752878
url : https://image.dcinside.com/download.php?no=24b0d769e1d32ca73ce987fa11d02831
+26amp%3B+Pearl+-+92+%5B480p%5D.mkv_20210111_102951.752.jpg
DB에 행추가
저장한 URL : https://image.dcinside.com/download.php?no=24b0d769e1d32ca73ce987fa1
Diamond+26amp%3B+Pearl+-+92+%5B480p%5D.mkv_20210111_102951.752.jpg
```

<크롤링 결과>

이러한 과정을 통해 웹사이트의 이미지 데이터를 크롤링하고 구조화된 형태로 저장하는 것이 가능해진다.

4.2. 동영상 크롤링

```
def process_video_post(tr_item):
    print('*'*12)
    print("동영상 크롤링")
    #print(tr_item)
    date_tag = tr_item.find(class_='gall_date')
    date = date_tag.get('title')
    date = date.split(' ')[0]
    print("게시 날짜: ", date)
    # 제목 추출
    title_tag = tr_item.find('a', href=True)
    title = title_tag.text
    print("제목: ", title)
    print("주소: ", title_tag['href'])
    # 이미지가 있는 게시물에 request
    arurl = ARTICLE_BASE_URL + title_tag['href']

    # URL 검증 로직 추가
    article_url = ARTICLE_BASE_URL + title_tag['href']
    if not is_valid_url(article_url):
        logger.error(f"Invalid URL: {article_url}")
        return # 함수 종료

    article_content = fetch_page(article_url)
    if article_content is None:
        return # 함수 종료
```

동영상 크롤링은 이미지 크롤링과 마찬가지로 해당 게시글의 URL이 유효한지 확인 후 게시글
짜, 제목, 주소를 추출한 후 해당 게시글로 페이지를 이동하게 된다.

하지만 이미지와 같은 방식으로 크롤링할 수 없다. 왜냐하면 HTML 파싱을 통해 동영상 파일
의 URL을 찾을 수 있지만 해당 URL을 이용하여 데이터를 다운받을 수 없도록 디시인사이드
측에서 차단해 놓았기 때문이다. 그렇기에 우리는 다른 방법을 찾아야만 했다.

조회수 342

공유  다운

동영상 파일이 포함된 게시글을 보게 되면 동영상 플레이어 우측 하단에 다운 버튼이 존재한
다. 오직 그 버튼을 이용해서만 동영상 데이터를 다운받을 수 있다. 동영상 데이터를 다운받기
위해서는 자동으로 '다운' 버튼을 누르도록 크롤링 코드를 작성해야만 했다.


```
chrome_options = Options()
chrome_options.add_argument("--headless")
proxy = "socks5://127.0.0.1:9050"
chrome_options.add_argument(f'--proxy-server={proxy}')
chrome_options.add_experimental_option('prefs', {
    "download.default_directory": os.path.abspath(video_dir), # 다운로드 경로 설정
    "download.prompt_for_download": False, # 다운로드 시 확인 대화 상자 끄기
    "download.directory_upgrade": True,
    "plugins.always_open_pdf_externally": True # PDF 파일을 항상 외부에서 열기
})
#service = Service(executable_path=r'C:/Users/tkdwn/chromedriver-win64/chromedriver.exe')
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=chrome_options)
driver.implicitly_wait(7)
# 웹사이트 접속
driver.get(arurl)
# 모든 iframe 웹 요소를 찾을
iframes = driver.find_elements(By.TAG_NAME, 'iframe')
for iframe in iframes:
    iframe_id = iframe.get_attribute('id')
    print(iframe_id)
    if 'movie' in iframe_id:
        # 'movie'가 포함된 id를 가진 iframe으로 전환
        driver.switch_to.frame(iframe)

    try:
        # iframe 내부에서 .btn_down 클래스 이름을 가진 웹 요소를 찾을
        download_button = driver.find_element(By.CLASS_NAME, 'btn_down')
        print(download_button)
        download_button.click()
    except NoSuchElementException:
        print("다운로드 버튼을 찾을 수 없습니다.")
        break

before_download_files = os.listdir(os.path.abspath(video_dir))

# 새로운 파일이 생성될 때까지 기다림
while True:
    time.sleep(1)
    after_download_files = os.listdir(os.path.abspath(video_dir))
    new_files = [f for f in after_download_files if f not in before_download_files and not
        f.endswith('.crdownload')]
    if new_files:
        break
```

‘다운’이라는 버튼을 자동으로 누르게 만드는 크롤링 코드를 작성하기 위해서는 웹 브라우저를 제어하는 라이브러리인 Selenium을 사용해야만 했다. 이를 사용하기 위해 Chrome 브라우저의 옵션을 설정한다. 여기서는 브라우저를 화면에 보이지 않게 하기 위해 headless 옵션을 사용하고, 프록시 서버를 설정하며, 다운로드 경로, 다운로드 확인 창 의 표시 여부 등을 설정한다. 그리고 설정한 Chrome 브라우저를 사용하여 추출한 URL의 웹사이트에 접속한다. 접속한 웹사이트에서 iframe 웹 요소를 찾아 'movie'가 포함된 id를 가진 iframe으로 전환한다. 왜냐하면 HTML 상 첫 번째 iframe요소에 동영상 데이터에 대한 정보가 있기 때문이다. 이 iframe 내에서 다운로드 버튼을 찾아 클릭하여 비디오를 다운한다. 다운로드가 완료될 때까지 기다린다.

```
# 새로 생성된 파일의 경로를 가져옴
file_path = os.path.join(os.path.abspath(video_dir), new_files[0])
file_name = os.path.basename(file_path)

# 파일을 버킷에 업로드
upload_to_bucket(file_name, file_path, 'capstone_video')

# 빅쿼리에 저장
bucket_name = "capstone_video"
client = bigquery.Client()
table_id = "strange-analog-405708.CrawlingDB.video"
bucket_path = f"gs://{bucket_name}/"
file_path = bucket_path + file_name

rows_to_insert = [
    {"videofile_name": file_name, "post_url": driver.current_url, "post_date": datetime.strptime(date,
"%Y-%m-%d").isoformat(), "file_path": file_path},
]

errors = client.insert_rows_json(table_id, rows_to_insert)
if errors == []:
    print("DB에 행추가")
else:
    print("에러발생: {}".format(errors))

break
time.sleep(0.5)
```

이후는 이미지 크롤링 코드와 마찬가지로 동영상 데이터를 Google Cloud Storage에 저장한 후 메타데이터 정보를 Google BigQuery에 저장하여 DB를 생성했다.

```
*****
동영상 크롤링
게시 날짜: 2023-12-11
제목: 12월 11일 위치웹스튜디오 삼승장 고고
주소: /mini/board/view/?id=video&no=3692&page=1
INFO:WDM:===== WebDriver manager =====
INFO:WDM:Get LATEST chromedriver version for google-chrome
INFO:WDM:Get LATEST chromedriver version for google-chrome
INFO:WDM:Driver [/home/jupyter/.wdm/drivers/chromedriver/linux64/120.0.6099.71/chromedriver-linux64/chromedriver] found in cache

movielcon2749984
<selenium.webdriver.remote.webelement.WebElement (session="e4042fcd803ed5afeff370d2dd65c462", element="BEAF5053F8113F6B8002AF62FEB5ACDD_element_37")>
새로운 행이 추가되었습니다.
```

실행 결과











5. Google Cloud Storage

<input type="checkbox"/> 이름 ↑	생성일	위치 유형	위치	기본 스토리지 클래스 ?
<input type="checkbox"/> capstone_image	2023. 11. 25. PM 4:10:51	Multi-region	us	Standard
<input type="checkbox"/> capstone_video	2023. 12. 1. PM 6:22:41	Multi-region	us	Standard

각각의 스토리지 경로에 크롤링 된 이미지/동영상 파일이 저장된다.

<input type="checkbox"/>	이름	크기	유형
<input type="checkbox"/>	 image_14878_0.jpg	26.6KB	image/jpeg
<input type="checkbox"/>	 image_14884_0.jpg	198KB	image/jpeg
<input type="checkbox"/>	 image_14886_0.jpg	140.7KB	image/jpeg
<input type="checkbox"/>	 image_14887_0.jpeg	38.4KB	image/jpeg
<input type="checkbox"/>	 image_14892_0.jpg	586.5KB	image/jpeg
<input type="checkbox"/>	 image_14893_0.jpg	255KB	image/jpeg
<input type="checkbox"/>	 image_14894_0.png	188.4KB	image/png
<input type="checkbox"/>	 image_14899_0.jpg	9.7KB	image/jpeg

<스토리지에 저장된 이미지 파일>

<input type="checkbox"/>	이름	크기	유형
<input type="checkbox"/>	 1.mp4	1.2MB	video/mp4
<input type="checkbox"/>	 17019470352770.mp4	1.5MB	video/mp4
<input type="checkbox"/>	 20231202-165210.mp4	3.3MB	video/mp4
<input type="checkbox"/>	 Desktop+2023.12.06++13.28.42.mp4	328.2KB	video/mp4
<input type="checkbox"/>	 IMG_9752.mov.mp4	2.1MB	video/mp4
<input type="checkbox"/>	 Screen_Recording_20231204_182609_YouTube.mp4	1.2MB	video/mp4
<input type="checkbox"/>	 [E]Desktop+2023.12.08++04.04.21.mp4	2.9MB	video/mp4
<input type="checkbox"/>	 rapidsave.com_ru_pov_alexey_arestovich_do_you_know_what_our...	2.1MB	video/mp4
<input type="checkbox"/>	 rapidsave.com_ru_pov_dear_comrades_today_december_1_the_fo...	1.3MB	video/mp4
<input type="checkbox"/>	 rapidsave.com_ru_pov_in_dnepropetrovsk_an_ukrainian_civillian-6...	1.7MB	video/mp4

<스토리지에 저장된 동영상 파일>

6. Google BigQuery

<input type="checkbox"/>	필드 이름	유형	모드
<input type="checkbox"/>	img_file_name	STRING	NULLABLE
<input type="checkbox"/>	post_url	STRING	NULLABLE
<input type="checkbox"/>	post_date	DATETIME	NULLABLE
<input type="checkbox"/>	file_path	STRING	NULLABLE

<이미지 테이블>

번호	img_file_name	post_url	post_date	file_path
1	image_1632709_0.jpg	https://gall.dcinside.com/mgallery/board/view/?id=arsenalfc&no=1632709&page=1	2023-12-08T00:00:00	gs://capstone_image/image_1...
2	image_1632709_1.jpg	https://gall.dcinside.com/mgallery/board/view/?id=arsenalfc&no=1632709&page=1	2023-12-08T00:00:00	gs://capstone_image/image_1...
3	image_1632709_2.jpg	https://gall.dcinside.com/mgallery/board/view/?id=arsenalfc&no=1632709&page=1	2023-12-08T00:00:00	gs://capstone_image/image_1...

<DB에 저장된 이미지>

<input type="checkbox"/>	필드 이름	유형	모드
<input type="checkbox"/>	videofile_name	STRING	NULLABLE
<input type="checkbox"/>	post_url	STRING	NULLABLE
<input type="checkbox"/>	post_date	DATETIME	NULLABLE
<input type="checkbox"/>	file_path	STRING	NULLABLE

<동영상 테이블>

번호	videofile_name	post_url	post_date	file_path
1	1.mp4	https://gall.dcinside.com/miniboard/view/?id=video&no=3674&page=1	2023-12-08T00:00:00	gs://capstone_video/1.mp4
2	rapidsave.com_ru_pov_in_dnepropetrovsk_an_ukrainian_civilian-62y9tj2fez4c1.mp4	https://gall.dcinside.com/miniboard/view/?id=video&no=3673&page=1	2023-12-08T00:00:00	gs://capstone_video/rapidsave.com_ru_pov_in_dnepropetrovsk_an_ukrainian_civilian-62y9tj2fez4c1.mp4
3	17019470352770.mp4	https://gall.dcinside.com/miniboard/view/?id=video&no=3672&page=1	2023-12-08T00:00:00	gs://capstone_video/17019470...
4	[E]Desktop+2023.12.08+-+04.0...	https://gall.dcinside.com/miniboard/view/?id=video&no=3671&page=1	2023-12-08T00:00:00	gs://capstone_video/[E]Desкто...
5	Desktop+2023.12.01+-+18.08...	https://gall.dcinside.com/miniboard/view/?id=video&no=3649&page=1	2023-12-01T00:00:00	gs://capstone_video/video/Des...
6	20231202-165210.mp4	https://gall.dcinside.com/miniboard/view/?id=video&no=3653&page=1	2023-12-02T00:00:00	gs://capstone_video/video/202...
7	20231202-165210.mp4	https://gall.dcinside.com/miniboard/view/?id=video&no=3653&page=1	2023-12-02T00:00:00	gs://capstone_video/20231202...

<DB에 저장된 동영상>

크롤링된 데이터들의 메타데이터 값들이 각 테이블에 잘 추가되었음을 확인할 수 있다.

■ 빅쿼리 데이터 추출 및 로컬 저장

```
from google.cloud import bigquery
from google.cloud import storage
import os

# 빅쿼리 클라이언트 생성
bq_client = bigquery.Client()

# 쿼리 작성
query = """
    SELECT file_path
    FROM `strange-analog-405708.CrawlingDB.image`
    WHERE file_path LIKE '%.gif'
    """

# 쿼리 실행
query_job = bq_client.query(query)
results = query_job.result()

# 쿼리 결과를 리스트로 변환
results_list = [row for row in results]

# 출력
print(results_list)

# 스토리지 클라이언트 생성
storage_client = storage.Client()
```

얼굴을 검출하고 탐지하기 전에 크롤링한 데이터 중 원하는 조건에 해당하는 데이터만 추출하기 위해 Google BigQuery에서 쿼리하여 로컬에 다운한다.

```
# 모든 결과에 대해 반복
for row in results_list:
    # 파일 경로 가져오기
    file_path_in_bucket = row[0]

    # 버킷 이름 가져오기
    bucket_name = file_path_in_bucket.split('/')[2]

    # 버킷 가져오기
    bucket = storage_client.bucket(bucket_name)

    # 파일 이름 가져오기
    file_name_in_bucket = '/'.join(file_path_in_bucket.split('/')[3:])
    local_file_name = file_name_in_bucket.split('/')[-1]







    # Blob 가져오기
    blob = bucket.blob(file_name_in_bucket)

    # 다운로드할 디렉토리가 있는지 확인하고, 없다면 생성
    if not os.path.exists('train'):
        os.makedirs('train')

    # 로컬에 파일 다운로드
    blob.download_to_filename(f'train/{local_file_name}')
```

위 그림과 같이 원하는 특정 조건을 만족하는 항목들을 쿼리하여 나온 결과(file_path)를 이용하여 스토리지에서 로컬로 다운을 받는 형식이다.


```
(capstone) jupyter@instance-20231207-210721:~$ python download.py
[Row({'gs://capstone_image/image_2618148_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_4813372_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_4813351_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_1626402_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_1626403_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_4800877_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_4800877_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_4800924_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_1616899_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_1641391_0.gif',}), {'file_path': 0}), Row({'gs://capstone_image/image_4829131_0.gif',}), {'file_path': 0})]
```

Name	Last Modified
 image_161...	seconds ago
 image_162...	seconds ago
 image_162...	seconds ago
 image_162...	seconds ago
 image_162...	seconds ago
 image_162...	seconds ago

쿼리한 결과가 train 파일 안에 잘 저장됨을 볼 수 있다. 이후 과정으로는 특정 조건에 대한 데이터들에 대해 얼굴 검출과 탐지과정을 진행하면 된다.

7. Image 얼굴 탐지

```
import cv2
import os
import shutil
from mtcnn import MTCNN
import time

start_time = time.time()

source_folder = "C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceDetect/train"
destination_folder = "C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceDetect/face"

# MTCNN 모델 로드
mtcnn = MTCNN()

total_images = 0
face_detected_images = 0
```

먼저 필요한 라이브러리와 모듈을 import한다. cv2는 OpenCV라고도 불리며, 이미지나 비디오를 처리하는 데 사용된다. os와 shutil는 파일과 디렉토리(폴더)를 관리하는 데 사용되며, mtcnn은 MTCNN 모델을 사용하기 위한 라이브러리이다.

```
for filename in os.listdir(source_folder):
    if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.jfif', '.webp')): # 이미지 파일 형식을 확인
        total_images += 1

        # 이미지 파일의 경로
        image_path = os.path.join(source_folder, filename)

        # 이미지 로드
        image = cv2.imread(image_path)

        # 이미지 전처리 (RGB로 변환)
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # 얼굴 감지
        faces = mtcnn.detect_faces(image_rgb)

        # 얼굴이 하나라도 감지된 경우
        if len(faces) > 0:
            face_detected_images += 1
            # 얼굴이 포함된 이미지를 destination_folder에 저장
            shutil.copy2(image_path, os.path.join(destination_folder, filename))
            print("Face detected in", filename)
        else:
            print("No face detected in", filename)

print("Face Detecting completed")
print(f"Total images: {total_images}")
print(f"Face detected images: {face_detected_images}")

end_time = time.time()
print("Spending time : ", end_time - start_time)
```

다음으로 source_folder에서 모든 이미지 파일을 불러온다. 이미지 파일을 로드하고, 이미지를 RGB 색상 공간으로 변환한 후, MTCNN 모델을 사용하여 이미지에서 얼굴을 감지한다. 얼굴이 감지된 경우, 해당 이미지를 destination_folder로 복사한다. 마지막으로, 전체 이미지 수와 얼굴이 감지된 이미지의 수를 출력한다.

8. 동영상 얼굴 탐지 (키 프레임 추출)

```
import cv2
import time
import os
import dlib

start_time = time.time()

dir_path = 'C:/Users/lmomj/Python/Capstone Design/Capstone Project/FaceDetect_video/Crawled_video'
save_dir = 'C:/Users/lmomj/Python/Capstone Design/Capstone Project/FaceDetect/face'

file_num = 0

# mtcnn = MTCNN()
detector = dlib.get_frontal_face_detector()
# face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

필요한 라이브러리와 모듈을 import 한다.

cv2는 OpenCV, time은 코드의 실행시간을 측정하기 위해 사용되며, os는 파일과 디렉토리(폴더)를 관리하는 데 사용된다. dlib은 얼굴 감지 및 인식 등의 기능을 제공하는 라이브러리이다. 이미지 얼굴 탐지와 달리 동영상 얼굴 탐지는 검출률보다 속도에 중점을 두었기 때문에 dlib의 get_frontal_face_detector 라이브러리를 사용하였다.

```
#디렉토리 내의 모든 파일을 순회
for filename in os.listdir(dir_path):
    # 파일 확장자가 .mp4 또는 .gif인 경우에만 처리
    if filename.endswith('.mp4') or filename.endswith('.gif'):
        file_path = os.path.join(dir_path, filename)
        cap = cv2.VideoCapture(file_path)

        # 비디오의 초당 프레임 수(FPS)를 구함
        fps = cap.get(cv2.CAP_PROP_FPS)
        # 2초에 한 번씩 프레임을 추출하기 위해 프레임 간격을 설정
        frame_interval = int(fps)

        frame_num = 0
        frame_count = 0
        capture_count = 0 # 캡처 횟수를 세는 변수
```

디렉토리 내의 모든 파일을 순회하며, 파일 확장자가 .mp4 또는 .gif인 경우에만 처리를 진행한다. 비디오 파일을 읽어온 후, 비디오의 초당 프레임 수를 구하고, 이를 기반으로 1초마다 한 번씩 프레임을 추출하도록 프레임 간격을 설정한다.

```
# 얼굴 탐지
faces = detector(img)
if len(faces) == 0:
    continue

# 5초마다 한 번씩 프레임을 추출
if frame_count % frame_interval == 0:
    # 이미지 저장
    cv2.imwrite(f'{save_dir}/file{file_num+1}_captured_frame_{frame_num+1}.jpg', img)
    frame_num += 1
    capture_count += 1 # 캡처 횟수를 세는 변수

    frame_count += 1
    if capture_count == 5: # 캡처 횟수를 세는 변수 == 20:
        break

file_num += 1

end_time = time.time()

print("{} second".format(end_time-start_time))
```

각 프레임에서 얼굴을 감지하고, 얼굴이 감지되었으면 해당 프레임을 이미지 파일로 저장한다.

9. FaceNet을 이용한 얼굴 인식 (동일 인물 판별)

```
from random import choice
from keras.models import load_model
from mtcnn.mtcnn import MTCNN
from PIL import Image
from matplotlib import pyplot as plt

# import numpy as np
from numpy import asarray
from numpy import savez_compressed
from numpy import load, expand_dims, linalg, argmin

from os import listdir
from os.path import join
import time
import shutil
import os

start_time = time.time()
```


필요한 라이브러리와 모듈을 import한다. random, keras, mtcnn, PIL, matplotlib, numpy, os 등의 라이브러리를 사용한다.

```
# 주어진 사진에서 하나의 얼굴 추출
def extract_face(filename, required_size=(160, 160)):
    # 파일에서 이미지 불러오기
    image = Image.open(filename)
    # RGB로 변환, 필요시
    image = image.convert('RGB')
    # 배열로 변환
    pixels = asarray(image)
    # 감지기 생성, 기본 가중치 이용
    detector = MTCNN()
    # 이미지에서 얼굴 감지
    results = detector.detect_faces(pixels)
    if len(results) == 0:
        print("No face detected in the image", filename)
        return None
    # 첫 번째 얼굴에서 경계 상자 추출
    x1, y1, width, height = results[0]['box']
    # 버그 수정
    x1, y1 = abs(x1), abs(y1)
    x2, y2 = x1 + width, y1 + height
    # 얼굴 추출
    face = pixels[y1:y2, x1:x2]
    # 모델 사이즈로 픽셀 재조정
    image = Image.fromarray(face)
    image = image.resize(required_size)
    face_array = asarray(image)
    return face_array
```

extract_face라는 함수를 정의한다. 이 함수는 주어진 이미지 파일에서 얼굴을 검출하고 추출하여 필요한 크기로 재조정하는 작업을 수행한다. 이미지를 불러오고 RGB로 변환한 뒤 배열로 변환하고, MTCNN을 사용하여 얼굴을 검출한다. 검출된 얼굴의 경계 상자를 추출하고, 이를 기반으로 얼굴을 추출한다. 추출된 얼굴을 모델에 필요한 크기로 재조정한다.

```
# 디렉토리 안의 모든 이미지를 불러오고 이미지에서 얼굴 추출
def load_faces(directory):
    faces = list()
    # 파일 열기
    for filename in listdir(directory):
        # 경로
        path = join(directory, filename)
        # 얼굴 추출
        face = extract_face(path)
        if face is not None: # 이 부분을 추가합니다.
            # 저장
            faces.append(face)
    return faces
```

load_faces라는 함수를 정의한다. 이 함수는 디렉토리 내의 모든 이미지 파일에서 얼굴을 추출하여 리스트로 반환한다.

```
# 이미지를 포함하는 각 클래스에 대해 하나의 하위 디렉토리가 포함된 데이터셋을 불러오기
def load_dataset(directory):
    X, y = list(), list()
    for filename in listdir(directory):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.jfif', '.webp')):
            path = join(directory, filename)
            face = extract_face(path)
            if face is not None:
                X.append(face)
                y.append(filename) # 파일 이름을 레이블로 사용
    return asarray(X), asarray(y) #Numpy 배열로 변환
```

load_dataset이라는 함수를 정의한다. 이 함수는 디렉토리 내의 모든 이미지 파일을 불러와서 각 이미지에서 얼굴을 추출하고, 이를 이미지 처리를 위해 배열로 변환한다.

```
# Crawled Dataset 불러오기
trainX, trainy = load_dataset('C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceDetect/face')
print(trainX.shape, trainy.shape)
# User Image 불러오기
testX, testy = load_dataset('C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceNet/val')
print(testX.shape, testy.shape)

# 배열을 단일 압축 포맷 파일로 저장
savez_compressed('faces-dataset.npz', trainX, trainy, testX, testy)

# 얼굴 데이터셋 불러오기
data = load('C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceNet/faces-dataset.npz')
trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
print('불러오기: ', trainX.shape, trainy.shape, testX.shape, testy.shape)

# Pre-trained model 불러오기
model = load_model('C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceNet/model/facenet_keras.h5')
```

그리고 크롤링된 데이터셋과 사용자의 이미지를 불러온다. 불러온 데이터셋을 'faces-dataset.npz' 파일로 저장하고, 저장한 데이터셋을 불러온 후, 각 데이터의 차원을 다시 확인한다. 그 후 pre-trained된 FaceNet 모델을 불러온다. 배열을 단일 압축 포맷 파일로 저장하는 이유는 후에 데이터를 임베딩 벡터로 변환할 때 유용하게 쓰이기 때문이다.

```
def get_embedding(model, face_pixels):
    # 픽셀 값의 척도
    face_pixels = face_pixels.astype('int32')
    # 채널 간 픽셀값 표준화
    mean, std = face_pixels.mean(), face_pixels.std()
    face_pixels = (face_pixels - mean) / std
    # 얼굴을 하나의 샘플로 변환
    samples = expand_dims(face_pixels, axis=0)
    # 임베딩을 갖기 위한 예측 생성
    yhat = model.predict(samples)
    return yhat[0]
```

get_embedding이라는 함수를 정의한다. 이 함수는 주어진 얼굴 이미지를 FaceNet 모델에 입력하여 얼굴 임베딩을 얻는 역할을 한다.

```
# 훈련 셋에서 각 얼굴을 임베딩으로 변환하기
newTrainX = list()
for face_pixels in trainX:
    embedding = get_embedding(model, face_pixels)
    newTrainX.append(embedding)
newTrainX = asarray(newTrainX)
print(newTrainX.shape)

# 테스트 셋에서 각 얼굴을 임베딩으로 변환하기
newTestX = list()
for face_pixels in testX:
    embedding = get_embedding(model, face_pixels)
    newTestX.append(embedding)
newTestX = asarray(newTestX)
print(newTestX.shape)
```

압축한 데이터셋과 사용자의 이미지에 대해 각각 얼굴 임베딩 벡터를 계산하고, 이를 'faces-embeddings.npz' 파일로 저장한다. 저장한 임베딩 벡터 데이터를 불러와서 사용자의 이미지와 크롤링된 각 이미지 사이의 유클리드 거리를 계산한다. 마지막으로, 각 이미지와 사용자 이미지 사이의 유클리드 거리를 출력하여 확인한다. 유클리드 거리가 임계치인 9 이하의 이미지를 'candidate_face' 디렉토리에 따로 저장함으로써 사용자와 동일 의심 얼굴 이미지를 별도로 관리할 수 있다.

```
# 배열을 하나의 압축 포맷 파일로 저장
savez_compressed('faces-embeddings.npz', newTrainX, trainy, newTestX, testy)

# User 이미지의 임베딩 얻기
test_embedding = get_embedding(model, testX[0])

# 각 Crawled 이미지와의 거리 계산
distances = []
for face_emb in newTrainX:
    distance = linalg.norm(face_emb - newTestX) #test_embedding
    distances.append(distance)

# 유클리드 거리 계산값 보여주기
print("")
for i, distance in enumerate(distances):
    print(f"Image {trainy[i]}: Euclidean Distance = {distance:.4f}\n")

# 유클리드 거리가 9 이하인 경우, 해당 이미지를 candidate_face 디렉토리로 이동
dst_dir = 'C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceNet/candidate_face'
for i, distance in enumerate(distances):
    # 유클리드 거리가 9 이하인 경우
    if distance <= 9:
        # 원본 이미지 파일 경로
        src_path = f'C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceDetect/face/{trainy[i]}'

        # 이미지 파일을 이동할 경로
        dst_path = os.path.join(dst_dir, trainy[i])

        # 파일 이동
        shutil.copy(src_path, dst_path)

end_time = time.time()

print("Spending time : ",end_time - start_time)

# 가장 가까운 얼굴의 인덱스 찾기
closest_face_index = argmin(distances)

closest_face_filename = trainy[closest_face_index]

# 가장 가까운 얼굴과의 유클리드 거리 계산
closest_distance = distances[closest_face_index]

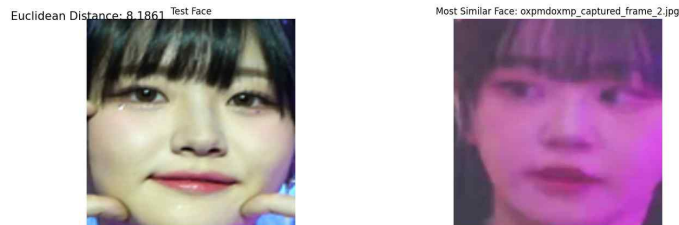
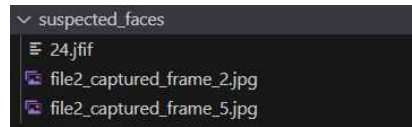
# 유클리드 거리가 Threshold보다 클 경우 "No Matching" 출력
if closest_distance > 9 :
    # "No Matching" 출력
    plt.figure(figsize=(6, 6))
    plt.title('')
    plt.text(0.4, 0.3, 'No Matching', fontsize=20, ha='center')
    plt.axis('off')
    plt.show()
else:
    # Using matplotlib to display the test face and the most similar face side-by-side
    plt.figure(figsize=(16, 5))

    # 사용자 Face 보여주기
    plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st subplot
    plt.imshow(testX[0])
    plt.title('Test Face')
    plt.text(0.5, 0.5, f'Euclidean Distance: {closest_distance:.4f}', fontsize=15, ha='center')
    plt.axis('off')

    # 가장 닮은 얼굴 보여주기
    plt.subplot(1, 2, 2) # 1 row, 2 columns, 2nd subplot
    plt.imshow(trainX[closest_face_index])
    plt.title('Most Similar Face: ' + trainy[closest_face_index])
    plt.axis('off')

plt.show()
```

유클리드 거리가 가장 작은 이미지의 인덱스를 찾는다. 이 인덱스를 통해 가장 유사한 이미지와 그 이미지의 유클리드 거리를 구할 수 있다. 유클리드 거리가 9보다 크면 동일 의심 인물이 없다고 판별하여 'No Matching'을 출력하고, 그렇지 않으면 사용자의 얼굴 이미지와 가장 유사한 이미지를 출력한다.

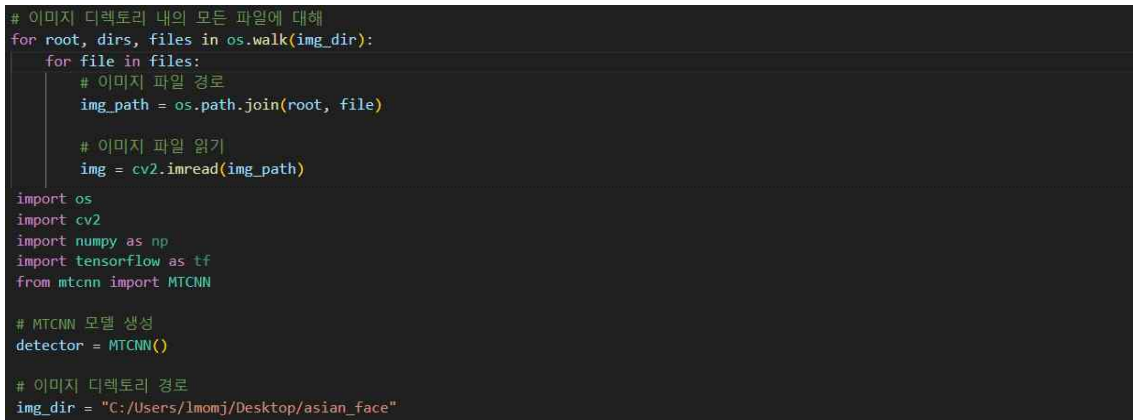


사용자가 의뢰한 사진과의 유클리드 거리가 가장 작은 이미지를 출력하고 Threshold 이내의 이미지들은 사용자와 동일 의심 인물로 간주하여 'candidate_face'파일에 별도로 저장하도록 하였다.

10. FaceNet Fine-Tuning (Data Pre-Processing)



pre-trained model인 FaceNet에서 제공하는 코드를 기반으로 Fine Tuning을 하기 위해. 데이터셋의 구조와 형식을 FaceNet에서 이용한 데이터셋과 일치시켜야 한다. 상위 폴더의 이름을 사용하면서 숫자로 인덱싱을 한 모양이다.



```
# 이미지가 없거나 문제가 있으면 건너뛰기
if img is None:
    continue

# 이미지에서 얼굴 탐지
result = detector.detect_faces(img)

# 얼굴이 탐지되지 않으면 건너뛰기
if not result:
    continue

# 얼굴 영역 추출 (첫 번째 얼굴만 사용)
bounding_box = result[0]['box']
x, y, w, h = bounding_box
cropped = img[y:y+h, x:x+w]

# 이미지 크기 조정
resized = cv2.resize(cropped, (160, 160))

# 정렬된 이미지 저장
aligned_img_path = img_path.replace('asian_face', 'aligned_asian_face')
os.makedirs(os.path.dirname(aligned_img_path), exist_ok=True)
cv2.imwrite(aligned_img_path, resized)
```

해당 작업을 통해 수집한 데이터셋을 FaceNet에 적합하게 구조를 변경하였다.



다만, 원래 FaceNet은 LFW(Labeled Faces in the Wild Home)와 같은 대규모 데이터셋으로 학습되었지만, GPU 사용이 제한되고 시간적 여유가 부족한 현 상황을 고려하여, 약 2,500개의 이미지를 포함하는 작은 데이터셋을 활용해 전이 학습을 진행하게 되었다.

11. FaceNet Fine-Tuning & Model Extraction

TensorFlow를 사용하여 GPU를 확인하지만, GPU를 사용할 수 없는 상황이라 CPU로 학습을 수행한다. 데이터셋의 위치와 사전 훈련된 모델의 경로를 지정하고, 사전 훈련된 FaceNet 모델을 불러온다. 학습에 필요한 파라미터(학습률, 에포크 수, 배치 크기, 검증 데이터셋 비율)를 설정한다.


```
import tensorflow as tf
import matplotlib.pyplot as plt

physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    print("GPU Okay")
else:
    print("GPU Unable")

data_dir = 'C:/Users/lmomj/Desktop/aligned_asian_face'
pretrained_model = 'C:/Users/lmomj/Python/Capstone Design/Capstone_Project/FaceNet/model/facenet_keras.h5'

model = tf.keras.models.load_model(pretrained_model)

learning_rate = 0.001
epochs = 100
batch_size = 32
validation_split = 0.2 # 검증 데이터셋의 비율 설정

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
                                                                validation_split=validation_split) # validation_split 추가
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(160, 160),
    batch_size=batch_size,
    class_mode='binary',
    subset='training') # 학습 데이터셋

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(160, 160),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation') # 검증 데이터셋

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_generator,
                    steps_per_epoch=len(train_generator),
                    validation_data=validation_generator, # 검증 데이터셋 추가
                    validation_steps=len(validation_generator),
                    epochs=epochs)
```

ImageDataGenerator를 이용해 이미지 데이터를 0~1 사이로 스케일링하고, 학습 데이터셋과 검증 데이터셋을 생성한다. 모델을 컴파일하는데, 여기서는 Adam, binary_crossentropy, Accuracy를 사용한다. fit 메서드를 이용해 모델을 학습시킨다. 여기서는 학습 데이터셋과 검증 데이터셋 모두를 사용한다.

```
# 학습이 끝난 후 모델 저장
model.save('trained_model.h5')
```

학습이 끝난 후 새로운 'trained_model.h5'를 추출하여 유클리드 거리에 변화가 있는지 살펴 수 있다.

4. 실험 및 분석

4.1 실험 환경

4.1.1 크롤링 속도 개선

- 크롤링 속도에 영향을 주는 요소 : 네트워크 대역폭, 서버 응답시간, 처리할 데이터의 양

(사용할 크롤링 최적화 전략)

- 네트워크 대역폭: 실험은 서로 다른 네트워크 대역폭을 제공하는 두 가지 유형의 인스턴스에서 수행된다. 인스턴스 1은 상대적으로 낮은 대역폭을 가지며, 인스턴스 2는 높은 대역폭을 제공한다.

Efficient Instance: vCPU 16개, 16GB RAM	Compute Optimized: vCPU 30개, 120GB RAM
<인스턴스 1>	<인스턴스 2>

- 캐싱 메커니즘 사용: 동일한 요청의 결과를 캐시 하여 재사용함으로써 네트워크 호출을 줄이고 크롤링 속도를 향상할 방법

- HTTP 세션 재사용: 동일한 호스트에 대한 연속적인 HTTP 요청에서 TCP 연결을 재사용하여 연결 설정 시간을 절약하는 방법

4.1.3 이미지 얼굴 탐지

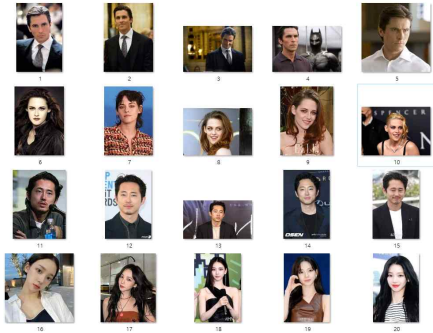
- 정확한 얼굴 탐지를 위해 다양한 라이브러리를 비교해 보았다.
- 속도보다 정확도에 중점을 두었다. 이를 위해 100개의 이미지 샘플 중 얼굴이 포함된 50장의 이미지를 사용하였다.

4.1.4 동영상 얼굴 탐지 (키 프레임 추출)

- 동영상 얼굴 탐지에서는 한 영상에 대해 총 5번의 키 프레임 추출을 수행하므로 소요 시간을 고려하여 어떤 라이브러리가 가장 효율적인지 비교하였다.
- 약 30초짜리 비디오를 각각 다른 라이브러리를 사용하여 검출하는데 소요되는 시간과 검출된 이미지의 개수를 비교하였다. (Video 얼굴 탐지 참고)

4.1.5 유클리드 거리 임계값 설정

- FaceNet을 통해 얼굴에 대한 임베딩 벡터값으로 유클리드 거리를 계산하는데, 해당 얼굴 이미지가 동일 인물인지 아닌지를 설정하는 임계값을 설정해야 한다.
- 서양인 남자 1명(크리스찬 베일), 서양인 여자 1명(크리스틴 스투어트), 동양인 남자 1명(스티븐 연), 동양인 여자 1명(카리나)씩 총 4명을 대상으로 하여 총 20장의 사진에 동일 인물 사진을 5장 포함하여 유클리드 거리를 출력하여 기준치를 설정한다.



4.1.6 Fine-Tuning model 성능 비교

- 기존에 사용한 FaceNet의 모델과 아시아인 얼굴을 추가로 학습한 모델을 비교하여 동양인에 경우 동일 사진에 대해서 같은 작업을 수행했을 때, 유클리드 거리 값이 더 줄어드는지 비교한다.
- FaceNet은 공식 깃허브 레포지토리에 두 개의 pre-trained model을 제공한다.

Model name	LFW accuracy	Training dataset	Architecture
20180408-102900	0.9905	CASIA-WebFace	Inception ResNet v1
20180402-114759	0.9965	VGGFace2	Inception ResNet v1

- LFW accuracy가 조금 더 높은 VGGFace2 dataset을 사용한 architecture 모델을 사용했다.

4.2 실험 결과 및 분석

4.2.1. 크롤링 최적화 및 속도 개선

	인스턴스 1 (대역폭 1 ~ 16Gbps)	인스턴스 2 (대역폭 20 ~ 100Gbps)
최적화 하지 않은 경우	109.071 초	103.387 초
HTTP 세션 재사용	106.332 초	101.431 초
캐싱 메커니즘 사용	108.842 초	106.389 초

- 네트워크 대역폭의 영향: 더 높은 대역폭을 가진 인스턴스 2가 모든 경우에서 약간 더 빠른 크롤링 속도를 보여주었다. 이는 높은 대역폭이 데이터 전송 속도를 향상하고, 따라서 전체 크롤링 시간을 줄일 수 있음을 나타낸다.

- HTTP 세션 재사용의 효과: HTTP 세션 재사용은 모든 경우에서 크롤링 속도를 약간 개선했다. 이는 TCP 연결의 재설정 시간을 줄여 네트워크 지연을 최소화하는 효과가 있음을 말한다.

- 캐싱 메커니즘의 효과: 캐싱 메커니즘을 사용한 경우, 전반적인 크롤링 속도에 큰 차이가 나타나지 않았다. 이는 구현한 크롤링 코드는 동일한 요청이 반복되는 경우가 극히 드물기 때문에 캐싱 메커니즘의 효과는 거의 없다고 보인다.

- 따라서, 높은 네트워크 대역폭을 가진 인스턴스로 HTTP 세션 재사용하는 방법으로 크롤링 코드를 구현하여 크롤링 과정을 실행시킴으로써 크롤링 속도를 향상했다.

4.2.2 이미지 얼굴 탐지

1.	dlib.cnn_face_detection_model_v1	이미지 당 4 min 이상 걸림
2.	dlib get_frontal_face_detector	37/50(74%), 1장 / 약 0.8s
3.	face_recognition()	42/50(84%), 1장 / 약 5s
4.	opencv cascadeclassifier	32/50(64%), 1장 / 약 0.5s
5.	MTCNN	49/50(98%), 1장 / 약 0.5s

- 실험 결과 여러 라이브러리 중 MTCNN 라이브러리가 약 98%의 얼굴 탐지율을 보였다.

- 이미지 1장당 약 0.5s의 소요 시간이 요구되어, 정확한 검출률을 중점으로 둔 이미지 얼굴 탐지에서는 MTCNN 라이브러리가 최적의 알고리즘이라고 판단할 수 있다.

4.2.3 동영상 얼굴 탐지 (키 프레임 추출)

1.	dlib.cnn_face_detection_model_v1	10 min 초과하여 중지
2.	dlib get_frontal_face_detector	약 58s 소요, 27장 이미지 검출
3.	face_recognition()	약 92s 소요, 30장 이미지 검출
4.	opencv cascadeclassifier	약 134s 소요, 17장 이미지 검출
5.	MTCNN	약 702s 소요, 35장 이미지 검출

- 실험 결과 30초짜리 동영상을 처리하는데, dlib의 get_frontal_face_detector가 약 60초의 처리 시간을 요구하며 총 27장의 얼굴 이미지를 추출하였다.

- 비록 검출된 이미지의 수가 가장 많은 라이브러리는 아니지만, 처리 속도를 고려하였을 때, 이 알고리즘이 가장 효율적인 것으로 판단하였다.

4.2.4 유클리드 거리 임계값 설정

	6	7	8	9	10	11
크리스찬 베일	20%(0)	20%(0)	60%(0)	80%(1)	80%(3)	100%(4)
크리스틴 스튜어트	40%(0)	40%(0)	40%(0)	60%(1)	80%(2)	100%(3)
스티븐연	20%(0)	40%(0)	60%(0)	80%(0)	100%(1)	100%(2)
카리나	20%(0)	20%(0)	60%(0)	80%(1)	80%(2)	100%(4)

- 표에서는 동일 인물 사진 검출률은 각 임계값에 대해 동일 인물로 판정된 사진의 비율을 나타내며, 괄호 안의 숫자는 잘못 판정된 사진의 수를 나타낸다.

- 총 4명을 대상으로 하여 총 20장의 사진에 동일 인물 사진을 5장 포함하여 기준치를 바꿔가며 유클리드 거리를 측정한 결과 다음과 같은 동일 인물 검출률과 타인의 얼굴 이미지가 포함된 개수가 도출되었다.

- '크리스찬 베일'의 경우 임계치를 6으로 했을 때, 5장의 본인 사진 중 1장이 검출되었고, 임계치를 9로 설정했을 때 본인 사진 4장이 검출됨과 동시에 본인 사진이 아닌 '스티븐 연'의 사진이 1장 검출되었다.

- 이와 같은 결과로 유클리드 거리의 임계값을 9로 설정하는 것이 동일 인물 사진 검출률과 타인 사진 검출률을 고려하였을 때 최적의 값이라고 선정하였다.

4.2.5 Fine-Tuning model 성능 비교



(images.jpeg)

기존 모델 : Image images.jpeg Euclidean Distance = 12.9605

새로운 모델 : Image images.jpeg Euclidean Distance = 12.6610

-기존 pre-trained model과 동양인 데이터로 추가 학습한 새로운 모델을 비교한 결과, 새로운 모델이 조금 더 우수한 성능을 보였다.

- 같은 'images' 이미지에 대한 유클리드 거리 계산 결과, 기존 모델에서는 12.9605였던 것이 새 모델에서는 12.6610으로 줄어들었다.

- 0.3이라는 미미한 차이이지만, 동양인 데이터에 대한 추가 학습을 통해 성능이 개선될 수 있음을 시사한다.

5. 결론 및 향후 과제

5.1 연구 과제 기여

5.1.1 사회적 기여

이 프로젝트는 개인의 프라이버시 보호와 사이버 공간에서의 안전을 강화하는 데 중요한 역할을 한다. 불법적으로 유포된 개인정보를 신속하게 탐지하고 대응함으로써, 사이버 범죄를 예방하고 피해자를 보호한다. 이는 사람들이 온라인 공간에서 자유롭게 소통하고 정보를 공유할 수 있는 환경을 조성할 수 있게한다.

5.1.2 기술적 기여

얼굴 인식/검출 기술과 클라우드 기반 데이터 관리 시스템을 통합한 본 연구의 접근 방식은 새로운 차원의 개인 정보 보호 기술을 제시한다. 얼굴 인식/검출 기술은 대규모 데이터 분석과 실시간 처리가 가능하도록 정확도와 속도측면에 집중해서 개발하였으며, 클라우드 기반의 데이터 저장과 처리 시스템은 대용량 데이터를 효과적으로 관리하며 보안성까지 갖췄다.

5.2 향후 연구 및 개발 과제

- 플랫폼 확장 및 크롤링 기술의 다양화: 현재는 디시인사이드에 초점을 맞추고 있지만, 이와 유사한 다른 온라인 커뮤니티나 SNS 플랫폼으로 크롤링 범위를 확장하는 것이 필요하다. 다양한 플랫폼에 적합한 크롤링 전략과 기술을 개발하여, 보다 광범위한 데이터 수집 및 분석이 가능하게 해야 한다.
- 크롤링 성능과 효율성 향상: 현재 구현된 크롤링 코드는 기본적인 기능을 제공하지만, 대규모 데이터 수집과 처리를 위해서는 성능과 효율성을 개선할 필요가 있다. 이를 위해 멀티 스레딩, 비동기 처리, 그리고 클라우드 기반의 분산 처리 기술 등을 적용해야 한다.
- 보안 및 개인정보 보호 강화: 크롤링 과정에서 수집된 데이터의 보안과 사용자의 개인정보 보호를 강화하는 것이 중요하다. 이를 위해 데이터 암호화, 접근 제어, 그리고 개인정보 비식별화 기술등을 적용한다.
- 자동화된 대응 메커니즘 개발: 크롤링을 통해 식별된 불법적인 콘텐츠에 대한 자동화된 대응 메커니즘을 개발하는 것이 필요하다. 이는 불법 콘텐츠의 신고, 차단 및 삭제 요청 등을 포함한 매커니즘을 구현한다.
- 사용자 인터페이스 및 경험 개선: 최종 사용자가 서비스를 더 쉽게 이용할 수 있도록 사용

자 인터페이스(UI) 및 사용자 경험(UX)를 개선해야 한다.

- 현재의 얼굴 탐지 및 인식 실험은 이미지와 비디오 파일을 기반으로 진행되었지만, 실시간 카메라 영상에서의 얼굴 탐지와 인식 성능을 실험하여 실제 상용화할 수 있도록 한다.

- FaceNet 외에도 다양한 얼굴 인식 알고리즘이 존재하는데, 이런 다른 알고리즘과의 성능 비교를 통해 더 나은 서비스를 구현할 수 있다.

6. 참고문헌

6.1 웹 크롤링 및 데이터 수집 기술

<https://zladnrms.tistory.com/164> 크롤링 코드 참고

<https://requests.readthedocs.io/en/latest/user/advanced/#proxies> 프록시 우회 레퍼런스

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/> bs4 레퍼런스

https://www.selenium.dev/documentation/webdriver/getting_started/install_library/
selenium 레퍼런스

6.2 클라우드 저장 및 데이터베이스 관리

<https://cloud.google.com/bigquery/docs?hl=ko> 구글 빅쿼리 공식문서

<https://cloud.google.com/storage/docs?hl=ko> 구글 스토리지 공식문서

<https://cloud.google.com/compute/docs/network-bandwidth?hl=ko> 구글 클라우드 인스턴스

6.3 FaceNet Fine-Tuning

<https://hyunah-home.tistory.com/entry/Facenet-%EC%96%BC%EA%B5%B4-%EC%9D%B8%EC%8B%9D-%EB%AA%A8%EB%8D%B8-Fine-tuning-%ED%95%98%EA%B8%B0?category=1019729>

<https://github.com/davidsandberg/facenet>

<https://www.kaggle.com/datasets/scienseenthusiast/asian-face?select=face>

6.4 얼굴 탐지 및 인식 알고리즘

<https://medium.com/@saranshrajput/face-detection-using-mtcnn-f3948e5d1acb>

<https://ukayzm.github.io/python-face-recognition/>

<https://thecodingnote.tistory.com/8>

<https://eehoeskrap.tistory.com/355>