

---

# 캡스톤 디자인 프로젝트 최종보고서

---

데이터 크롤링과 얼굴인식을 이용한 개인정보 영상 검색 시스템

이충헌(151779)

김태준(164160)

김병수(181526)



캡스톤 지도교수 서명 유 석 봉 

## [제목/주제]

데이터 크롤링과 얼굴인식을 이용한 개인정보 영상 검출 시스템.

## [배경 및 문제]

- 디지털성범죄 영상물 유통사례 적발 4,584건 (2018.7.31. 방송통신위원회)
- 카메라 등을 이용 불법촬영 범죄 꾸준한 증가 (2019.12.30. 여성가족부)  
(‘12) 2,400건 → (‘13) 4,823건 → (‘14) 6,623건 → (‘15) 7,623건 → (‘16) 5,185건 → (‘17) 6,470건
- 피해를 최소화하기 위해 영상의 유출을 막기위한 빠른 조치의 필요성이 대두됨.
- 피해자가 자신의 영상이 어디에 유출되었는지 직접 찾기 어려움.
- 목적 : 개인정보가 인터넷에 광범위하게 유출되기 전 신속한 대응.

### <필요성 및 기대 효과>

- 1 ) 리벤지 포르노(디지털성범죄)영상 노출 이후에 빠른 후속조치가 이루어 질 수 있도록 커뮤니티, SNS에서 최대한 빠르게 본인의 얼굴 등이 노출되었다고 의심되는 영상, 사진 등을 찾을 수 있음.
- 2 ) 유출에 따른 피해를 최소화하는 시스템을 개발함으로써, 개인정보의 무단 활용을 방지함.
- 3 ) 궁극적으로 데이터 수집을 통한 빠른 후속 대처 및 법적 대응에 용이하게 함.

## [목표]

웹 사이트에서 웹 크롤링을 통한 영상 내 얼굴과 같은 개인정보를 인식함으로써, 개인정보 유출로 인한 피해 방지 기술 연구/개발

## [팀 구성]

<소프트웨어 공학과>

팀명 : T2

팀 리더 : 이충헌(151779)

팀원: 김태준(164160), 김병수(181526)

[요구 사항]

요구사항 ID	명칭	설명
R001	데이터 수집	특정 사이트 크롤링.
R002	데이터 저장	크롤링 된 데이터 저장.
R003	얼굴인식	얼굴인식 딥러닝 학습.
R004	매칭 시스템	사용자의 데이터 매칭 및 비교
R005	웹 서비스	종합적으로 사용할 수 있는 웹 서비스

요구사항 ID		R001	요구사항 명칭	데이터 수집
상세 설명	정의	사진 및 영상 데이터 수집		
	세부	크롤링을 통한 사진 및 영상 데이터 수집		
	내용	크롤링 기법을 이용하여 개인정보 영상이 유출되었을 거라고 예상되는 사이트에서 데이터 수집.		

요구사항 ID		R002	요구사항 명칭	데이터 저장
상세 설명	정의	데이터 저장		
	세부	크롤링 된 데이터를 서버에 저장		
	내용	크롤링 된 데이터의 이미지를 저장하고, URL과 이미지 Hash값을 데이터베이스에 저장.		

요구사항 ID		R003	요구사항 명칭	얼굴인식
상세 설명	정의	얼굴 인식을 위한 딥러닝 학습.		
	세부	얼굴인식 모델 학습		
	내용	동양인 얼굴 데이터셋을 이용하여 모델을 추가 학습시킴. 사용 용도에 맞게 모델 수정.		

요구사항 ID		R004	요구사항 명칭	매칭 시스템 (face verification)
상세 설명	정의	사용자 이미지 매칭.		
	세부	수집된 데이터와 입력 이미지 비교.		
	내용	학습된 얼굴인식 모듈을 이용하여 사용자의 이미지를 수집한 데이터와 비교		

요구사항 ID		R005	요구사항 명칭	웹 클라이언트
상세 설명	정의	웹 서버 및 클라이언트		
	세부	사용자가 쉽게 접근할 수 있는 웹 서비스 개발.		
	내용	웹 서버 및 웹 클라이언트 개발		

## 주요 역할

1. 크롤링 구현 및 검증: 이충헌(151779)
2. 얼굴 인식 인공지능 모델(Arcface) 설계 및 검증: 김병수(181526)
3. 얼굴 인식 인공지능 모델 학습 및 성능 테스트: 이충헌(151779), 김병수(181526)
4. Spring을 통한 웹 서버 구현 및 검증: 김태준(164160)
5. 데이터베이스 구현 및 검증: 김태준(164160)
6. 웹 클라이언트 구현: 김태준(164160), 김병수(181526)

주요 역할 외에도, 이 캡스톤 프로젝트는 기본적으로 2인씩 묶어서 페어 프로그래밍으로 진행됐다.

(다음 페이지의 간트 차트 및 깃허브 저장소, 구현파트 참고)

**[간트 차트]**

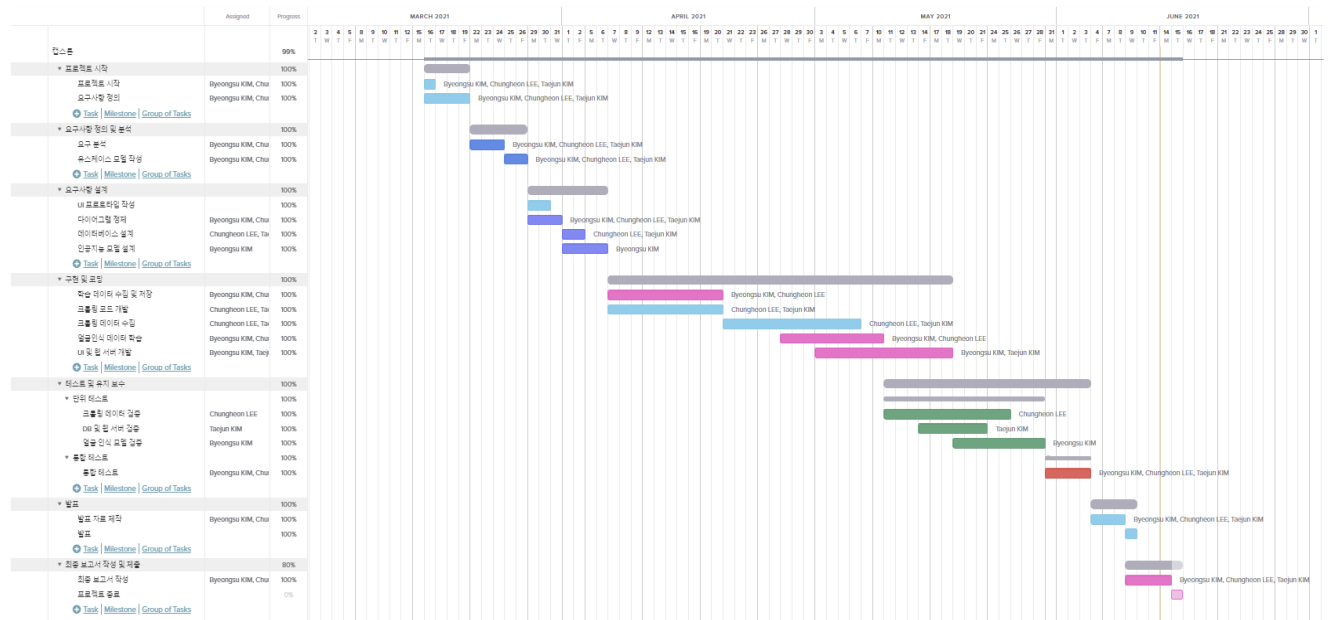


Figure 1. 간트차트.

**[설계]**

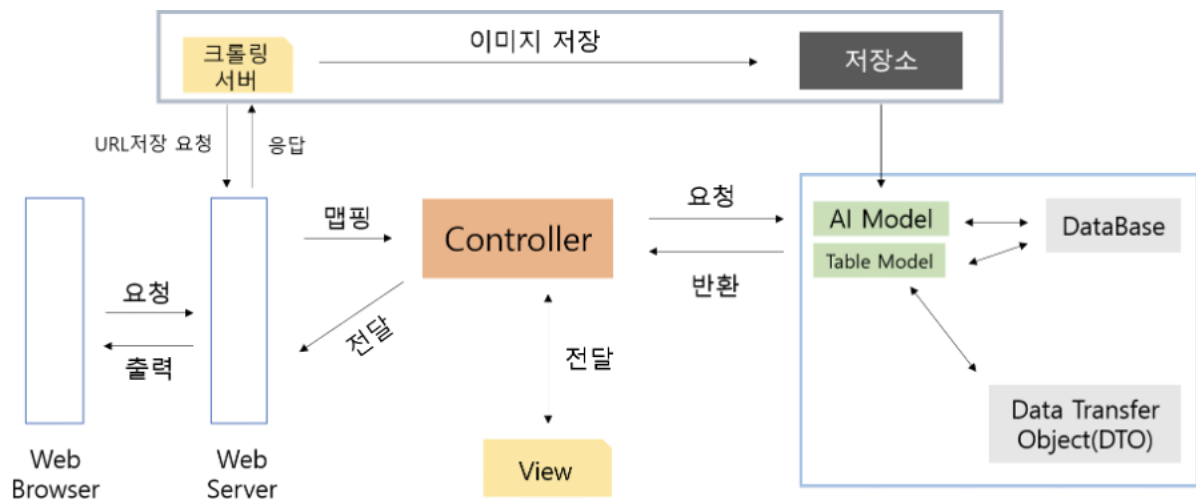


Figure 2. MVC모델.

MVC 모델을 사용했다. 사용자가 웹 서버에 사진 매칭을 요청하면 웹 서버가 사용자의 입력과 함께 컨트롤러에게 사용자 요청을 전달한다. 컨트롤러는 인공지능 모델에게 사용자의 요구 처리를 요청하고 이후 모델이 데이터를 비교하여 그 결과값을 컨트롤러와 웹 서버를 통해

반환한다. 컨트롤러는 이미지와 hash값을 결과로 반환 받고, hash값을 이용해 데이터베이스에서 url을 탐색하고, View로 전달하여 결과를 보여준다.

크롤링 서버는 수집한 이미지를 저장소에 저장하고, 이미지의 hash정보와 이미지가 존재하는 웹 사이트의 URL을 DB에 저장하라고 요청한다. 웹 서버는 컨트롤러를 통해 Table Model에게 사용자의 요구를 처리하도록 요청한다.

### [요구 사항 구현]

#### - 데이터 수집(R001 - 구현: 이충현(151779)):

특정 사이트에서 개인정보(얼굴 영상)가 포함된 이미지를 수집하기 위해 python의 Face recognition 라이브러리를 이용해서 인터넷 커뮤니티 사이트에서 얼굴 정보 이미지만을 선별하여 저장하거나, 비디오에서 얼굴 키 프레임 추출하여 저장하는 방식을 취한다.

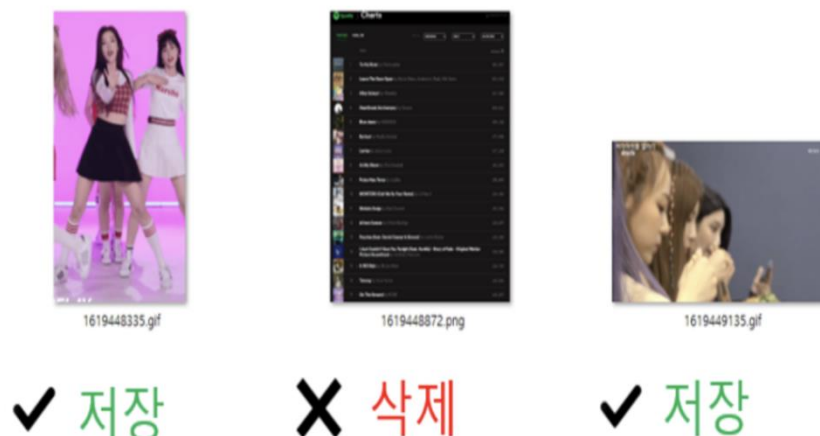


Figure 3. 크롤링 작동 방식 예시.

지정된 유해 사이트에서 게시판들을 크롤링한다. 카테고리를 선택하여 게시물 중에 사진이 포함된 게시물만 선택하여 선택한 모든 게시판에 있는 이미지 및 영상에서 해당 얼굴 정보 이미지가 있는 이미지를 크롤링하고 URL 정보를 가져온다. 이때 사진에 여러 사람이 있다면 각 얼굴만을 크롭하여 각각 저장하게 된다. 예시로 그림 5에서 첫 번째와 세 번째 이미지는 사람의 앞모습 및 옆모습이 식별되어 해당 이미지를 저장하고, 두 번째 이미지는 얼굴에 해당하는 영역을 찾지 못해 삭제 처리한다. 이처럼 개인정보 얼굴 이미지가 유출된 이미지만 선별하여 저장하게 된다. 본 프로젝트에는 법적인 문제로 리벤지 포르노가 유출된 사이트를 크롤링하지 못했고, 대신 예시 사이트로 커뮤니티 사이트(디시인사이드, 유튜브)의 연예인 사진들을 크롤링 했다.

이미지 및 URL 정보의 저장 방식의 자세한 방법은 데이터 저장(R002) 구현 부분에서 설명한다.

- 데이터 저장(R002 - 구현: 이충현(151779), 김태준(164160)):

크롤링을 하여 얼굴 정보가 있는 이미지를 수집하는데 성공했으면 얼굴 이미지들을 얻어 이미지의 고유 해시 정보를 파일의 이름으로 하여 저장소에 저장한다.

이미지의 고유 해시 정보를 쓴 이유는 커뮤니티에서 특정한 사용자가 동일한 이미지를 게시판에 계속 올리는 도배 현상이 관측되었다. 그리하여 동일한 사진이 계속 쌓여 중복된 이미지의 불필요한 용량 차지 및 얼굴 인식을 할때 중복 계산을 하여 시간을 더욱 소모하는 문제가 발생했다. 따라서, 중복된 이미지를 제거하기 위해서 디지털 포렌식에서 사용하는 이미지의 고유 해시(hash) 정보를 이용하여 그림 6의 이미지 예시처럼 고유 해시 정보를 파일 이름으로 해서 사진을 저장소에 저장하고, 이미지의 해시 정보와 URL를 데이터베이스에 저장하는 방식을 사용했다.

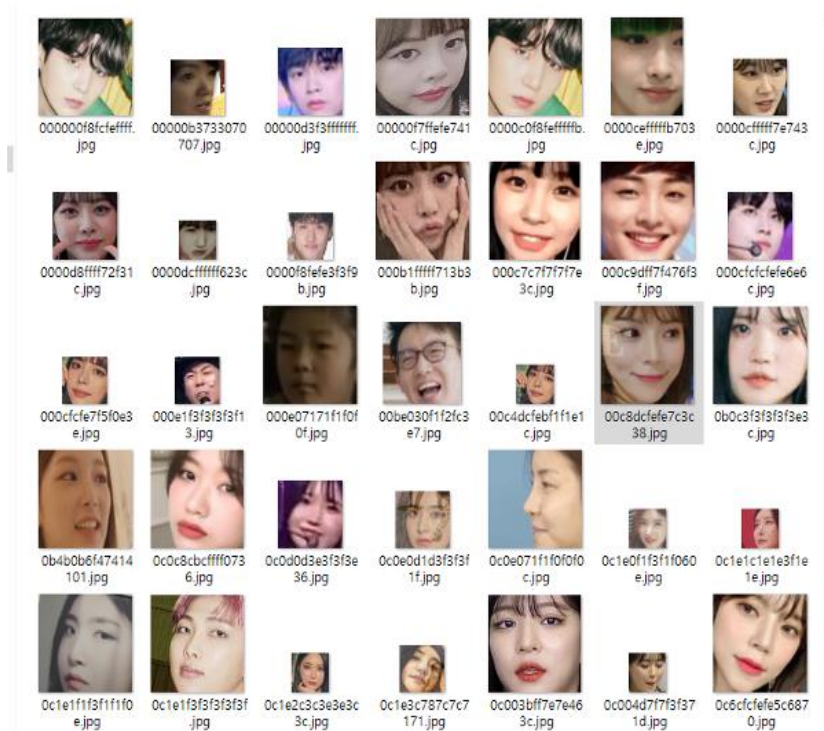


Figure 4. 크롤링 된 이미지들 예시.

해당 방식을 사용하여 크롤링한 6,120개의 이미지 중 중복된 3729개의 이미지를 제거하고 2391개의 크롤링 이미지만을 저장하여 60.93%의 중복된 이미지를 제거하여 저장소 용량 압축 및 모델 성능의 처리시간을 개선하는 효과를 냈다. 다만, 이미지가 약간이라도 변경된다면 해시가 다르기 때문에 다른 이미지라고 인식한다.

DB에 저장된 이미지	6,120
저장소에 저장된 이미지	2,391
중복 이미지 제거 효과	60.93%

Table 1. 해시를 이용한 이미지 압축 효과

다음은 github(<https://github.com/kimtaejun97/CapstoneDesign/tree/master/Crawling>)에서 크롤링 관련 코드 예시이다. 얼굴 이미지의 개수는 크롤링하는 코드의 개수에 비례한다.

- DCImage.py : 디시인사이드(<https://www.dcinside.com/>) 커뮤니티에서 사진을 크롤링해오는 코드이다.
- Getvideo.py : youtube(<https://www.youtube.com/>)에서 동영상을 다운로드 받은 다음 프레임을 나누어서 저장 후 얼굴 이미지만 선별해서 저장소에 저장하는 코드이다.
- Utils.py : 는 DCImage.py와 Getvideo.py 안의 클래스의 부모 클래스로 어디에 저장할 것인지와 데이터베이스에 사진 관련 정보(URL과 사진 이름)를 전송하는 코드이다. 또한, 이미지 hash 추출 관련 함수가 클래스 안에 내재되어 있다.



**Figure 5. 데이터베이스**

데이터베이스는 id(primary key - auto increment), URL, hash 3가지 필드로 구성되어 있다. 수집된 이미지의 정보를 웹 서버에 Post방식으로 전송하면 웹 서버에서는 이를 받아 데이터베이스에 저장한다. 데이터베이스는 AWS RDS의 Mysql을 사용하였다. 저장된 데이터의 사용은 웹 서비스(R005)에서 다룬다.

크롤링 관련 코드 : <https://github.com/kimtaejun97/CapstoneDesign/tree/master/Crawling>

데이터베이스의 연동 : <https://github.com/kimtaejun97/pidetecction-webServer>



- **얼굴인식(R003 - 구현 : 이충현(151779), 김병수(181526)) :**

얼굴인식 모델은 오픈소스로 공개된 ArcFace(<https://github.com/deepinsight/insightface>)를 사용했다. ArcFace(Additive Angular Margin Loss)는 DCNNs를 통해 학습된 feature embeddings의 분별력을 Additive Angular Margin Loss를 추가함으로써 성능을 향상시켰고, 이에 따라 좋은 성능을 자랑한다. 기존 arcface의 정확도는 Ifw 데이터셋을 대상으로 99.82%이다.

그런데 서양인 데이터셋으로 학습된 기존의 모델(pretrained model)은 동양인을 대상으로 할 때 약간 정확도가 떨어져 동양인 데이터셋을 구하였다. 동양인 데이터셋을 이용하여 새로 학습시킬 예정이었으나, 기존 pretrained model의 학습량만큼 학습하기엔 하드웨어 및 시간이 부족하여 처음부터 학습하는 방식을 사용할 수 없었다. 따라서, 기존의 모델(pretrained model)에 동양인 데이터(9만 ID, 23만장)를 이어서 학습하는 방식으로 인공지능 모델을 학습했다. 학습 환경은 GPU RTX 2080 SUPER 1개, ubuntu 18.04에서 학습했다.

오픈소스에서 변경한 사항은 다음과 같다.

1. Arcface는 얼굴 이미지를 1:1 비교해볼때 이미지를 사진에서 .bin 파일로 전부 취합 후 그걸 Arcface의 train 코드나 test 코드에서 load하여 사용하는데 이미지를 바로 load해 사용할 수 있도록 변경하였다. 기존에는 분 단위로 걸리던 프로그램 처리 시간이 초 단위로 줄어드는 처리시간 감소 작업을 했다.
2. Arcface는 얼굴 인식(face verification)을 할때 사진을 가로 112, 세로 112로 고정되고 반드시 들어오는 이미지에는 얼굴이 하나는 있다는 가정하에 얼굴 매칭 작업을 한다. 하지만, 크롤링 데이터나 사용자가 입력하는 이미지는 반드시 한 사람만의 얼굴이 있다고는 보장 못하기 때문에 한사람만의 얼굴을 크롭하는 예외처리 과정을 추가했다. 이때 사진에 여러 사람의 얼굴이 검출된다면 그 중 가장 큰 사람의 얼굴을 크롭하게 된다.

또한 사진을 가로 112, 세로 112로 리사이징 하는 코드를 추가했다.

3. GPU의 부족으로 인해 배치 사이즈를 32로 줄이고 learning rate를 0.1에서 0.001로 줄여서 학습했다.
4. 얼굴 매칭(face verification)에선 임계값을 정하는데 분류성능평가지표로 Precision(정밀도)와 Recall(재현율), Accuracy(정확도) 를 사용하여 반복문을 돌려 이미지 테스트에서 가장 관찮게 나온 임계값을 고른다. 그리고 매칭 시켜 볼 두 사진을 모델에 넣은 결과의(임베딩) 차이를 구하여 임계값보다 작으면 두 사진이 같다고 간주한다. 이때 정해진 고정된 임계값이 없고 이미지의 라벨링값(ground truth)를 이용하여 테스트

데이터셋마다 최적의 임계값을 고르는 형태인데 크롤링 이미지의 경우 ground truth가 없으니 임계값을 동일한 방식으로 정할 수 없는 애로사항을 겪었다. 따라서 이 방식을 통계적 방식으로 해결하기 위하여 얼굴 인식때 사용하는 데이터셋인 lfw dataset의 임계값인 1.16을 참고하여 약간 높은 1.2를 임계값으로 정했다.

관련 코드는

<https://github.com/kimtaejun97/CapstoneDesign/tree/master/recognition/ArcFace> 에서 확인할 수 있다.

- 매칭 시스템(R004 - 구현: 김태준(164160), 김병수(181526)) :

인공지능 서버에서는 웹 서버에서 보내는 매칭 요청을 수신한다. 수신된 요청은 대기열에 추가되게 되고, 리소스가 모두 이용되고 있을 경우에는 추가적인 대기시간이 필요하다. 이러한 대기열 방식은 각 요청이 순차적으로 처리됨을 보장한다. 각 클라이언트의 독립성 보장은 R005에서 다룬다.

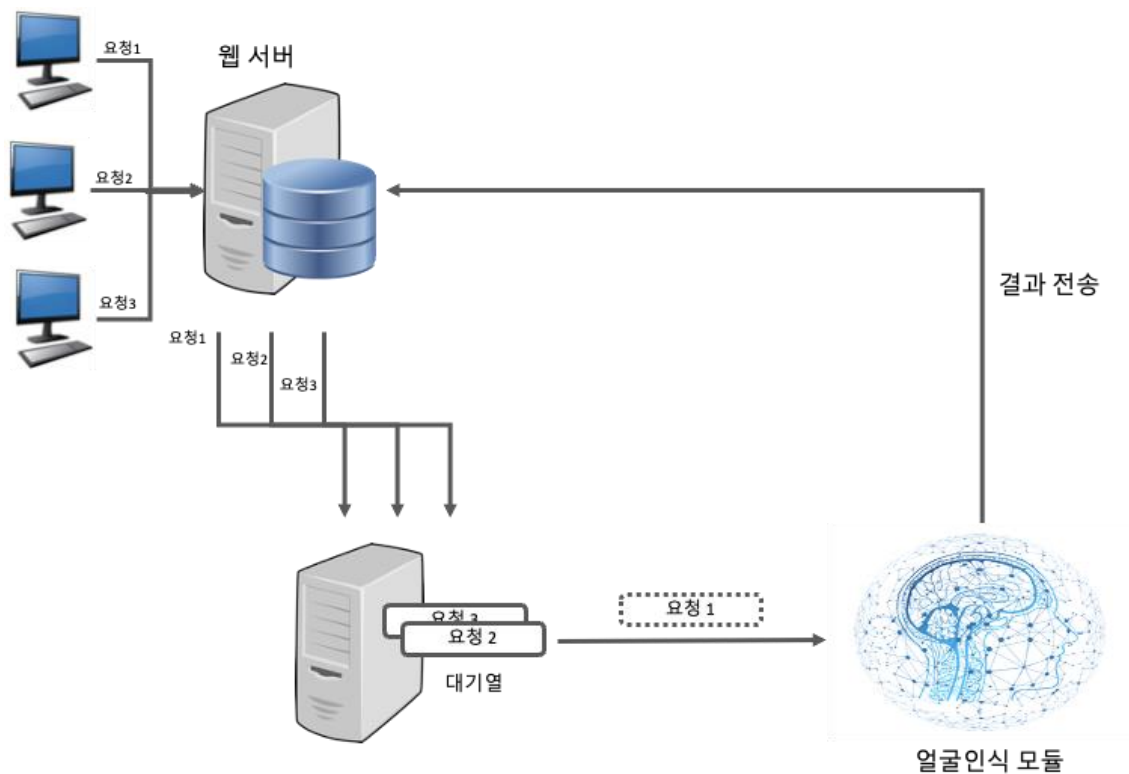


Figure 6. 다중 요청 순차 처리.

멀티쓰레드에서의 안전성을 위해 synchronized 블럭을 사용하였고, Queue 방식을 사용하여 대기열을 구성하였다. 자신의 차례가 되면 remove 메소드를 이용하여 자신의 정보를 반환하고 대기열에서 삭제한다.

인공지능 모델에서 타겟 이미지와 비교 이미지의 임베딩 차를 임계값과 비교함으로써 동일 인물인지 평가한다. 만약 임계값 이하라면 매칭된 것으로 판단하여 비교 이미지, Hash 등 결과 데이터를 웹 서버로 전송한다. 임베딩 차의 구간에 따라 매칭률은 높음, 중간, 낮음으로 표현된다.

매칭 요청 수신 서버 : <https://github.com/kimtaejun97/CapstoneDesign/tree/master/AIServer>

결과 전송 : <https://github.com/kimtaejun97/CapstoneDesign/tree/master/recognition/ArcFace>

- 웹 서비스(요구사항 R005 - 구현: 김태준(164160), 김병수(181526)):



Figure 7. 웹 서비스 구현.

해당 요구사항은 서비스 웹 서버와 웹 클라이언트를 뜻한다. Spring framework를 사용하여 구현하였으며 View Template로는 thymeleaf를 사용하였다. 또한 AWS의 EC2서버에 해당 웹 서비스를 배포하였다.

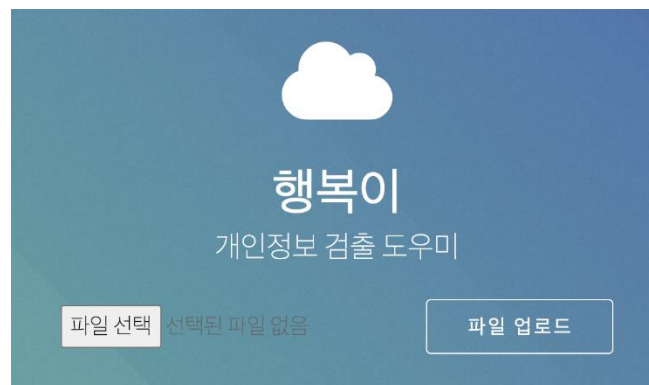
사용자가 개인정보 검출 서비스를 쉽게 사용할 수 있도록 웹 클라이언트를 만들었다. 개인정보 유출의 걱정 때문에 불안해 하는 사용자들을 생각하며 클라이언트 이름은 행복이로 정하였다. 아래 사진은 메인 화면이며, 행복이 서비스에 대한 간단한 설명을 볼 수 있다.

각 요청url은 Controller를 통해 매핑되어 있다. 크게 홈, 이미지 입력, 결과로 나뉘어 진다.



**Figure 8. 홈 View**

홈의 해당 URL로 GET방식으로 요청을 보내면 매핑된 컨트롤러에서 응답으로 index 페이지를 반환하게 되고, 클라이언트는 매칭 시스템에 대한 간략한 설명이 정리되어 있는 홈 뷰를 볼 수 있다. 시작하기 버튼을 클릭하면 이미지를 입력할 수 있는 페이지로 이동한다.



**Figure 9. 입력 화면.**

이미지 입력 페이지에서는 매칭의 대상이 될 이미지를 추가하고, 업로드 할 수 있다. 업로드된 이미지는 매칭 요청 수신 서버(인공지능 서버)로 전달되게 되는데 이 때, HTTP통신을 이용하고, POST방식으로 multipart/form-data 형식으로 전달된다. 이 때 클라이언트의 JSESSIONID도 같이 넘겨주게 된다.

이 페이지에서 사용자가 사진을 업로드 하면 매칭 중 이라는 글씨가 나타난다. 얼굴을 인식할 수 없는 불안정한 사진이라면, 다른 사진을 입력하라는 경고 메시지를 띄워준다.

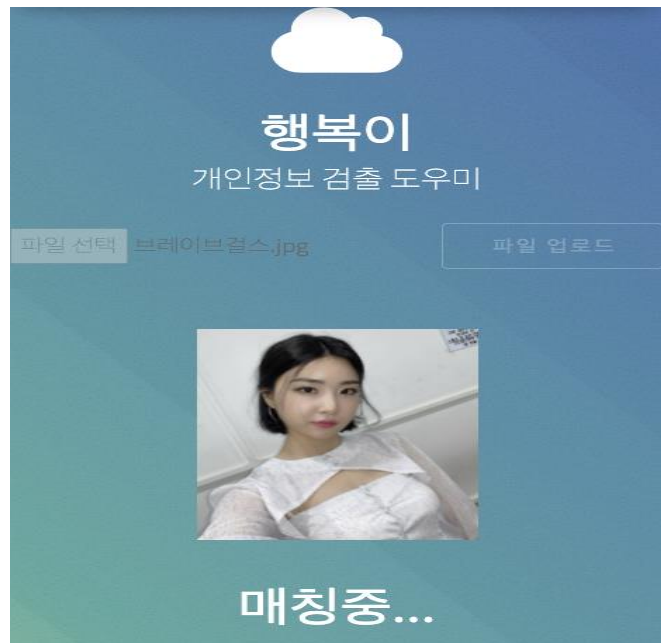



Figure 10. 매칭 화면

매칭이 끝나면 결과 페이지로 이동한다. 결과 페이지에선 사용자가 입력한 사진과 매칭 결과를 보여준다. 만약 사용자의 사진이 특정 사이트에서 발견이 되었다면 발견된 사진과 사이트 URL, 그리고 매칭률을 보여준다. 매칭률은 높음, 중간, 낮음으로 나타난다.



개인 정보 유출 결과





매칭 이미지	URL	일치
	<a href="https://gall.dcinside.com/mgallery/board/view/?id=bravegirls0409&amp;no=494743&amp;page=13">https://gall.dcinside.com/mgallery/board/view/?id=bravegirls0409&amp;no=494743&amp;page=13</a>	높음
	<a href="https://gall.dcinside.com/mgallery/board/view/?id=bravegirls0409&amp;no=495195&amp;page=5">https://gall.dcinside.com/mgallery/board/view/?id=bravegirls0409&amp;no=495195&amp;page=5</a>	높음
	<a href="https://gall.dcinside.com/mgallery/board/view/?id=fromis&amp;no=946850&amp;page=5">https://gall.dcinside.com/mgallery/board/view/?id=fromis&amp;no=946850&amp;page=5</a>	높음
	<a href="https://gall.dcinside.com/mgallery/board/view/?id=bravegirls0409&amp;no=494728&amp;page=14">https://gall.dcinside.com/mgallery/board/view/?id=bravegirls0409&amp;no=494728&amp;page=14</a>	높음

Figure 11. 결과 화면.

결과 수신은 해당 컨트롤러와 매핑된 URL에 요청이 수신되면 발생한다. 수신되는 데이터는 매칭된 이미지, 이미지의 Hash값, 매칭 정도이며, Hash값을 이용하여 데이터베이스에서 URL을 탐색한다. 즉 결과 데이터는 매칭 이미지, 이미지의 발견 URL, 매칭 정도이다. 이미지는 multipart/form-data, 나머지 데이터는 Json형식으로 전송된다. 매칭 결과 수신이 완료되고, 이전에 클라이언트가 보냈던 매칭 요청에 대한 응답이 반환되면 결과페이지로 이동하여 수신된 결과데이터를 보여준다.

Name	Value
JSESSIONID	F048CDFFDA63CB29E0...

**Figure 12. 세션 아이디**

매칭과 결과의 수신과정에서 클라이언트간의 구분은 세션아이디를 이용한다. 먼저 클라이언트가 입력 페이지에 접속하게 되면 웹 서버에서는 세션아이디가 없다면 세션아이디를 자동으로 생성하여 Cookie에 넣어준다. 이미지 업로드 버튼을 클릭하는 이벤트가 발생하였을 때 해당 클라이언트의 웹 서버 세션아이디를 가져오게 되고, 이를 입력 이미지와 함께 매칭 요청 수신 서버로 전달하게 된다. 수신 서버에서는 결과를 웹 서버에 보낼 때 헤더를 수정하여 클라이언트의 세션아이디로 요청을 보낸다.

웹 서버에서는 결과를 수신 받고, 이전에 클라이언트가 웹 서버에 접속할 때 생성하였던 세션에 결과데이터를 저장한다.

모든 결과가 수신되고, 클라이언트가 결과페이지로 리다이렉트 될 때 세션아이디에 해당되는 결과데이터를 View에 넘겨주게 된다. 세션에 저장된 데이터는 데이터를 View에 넘겨준 후 바로 제거된다. 각 클라이언트는 다른 세션아이디를 가지기 때문에 클라이언트는 각각 독립된 페이지를 보는 것이 보장된다.

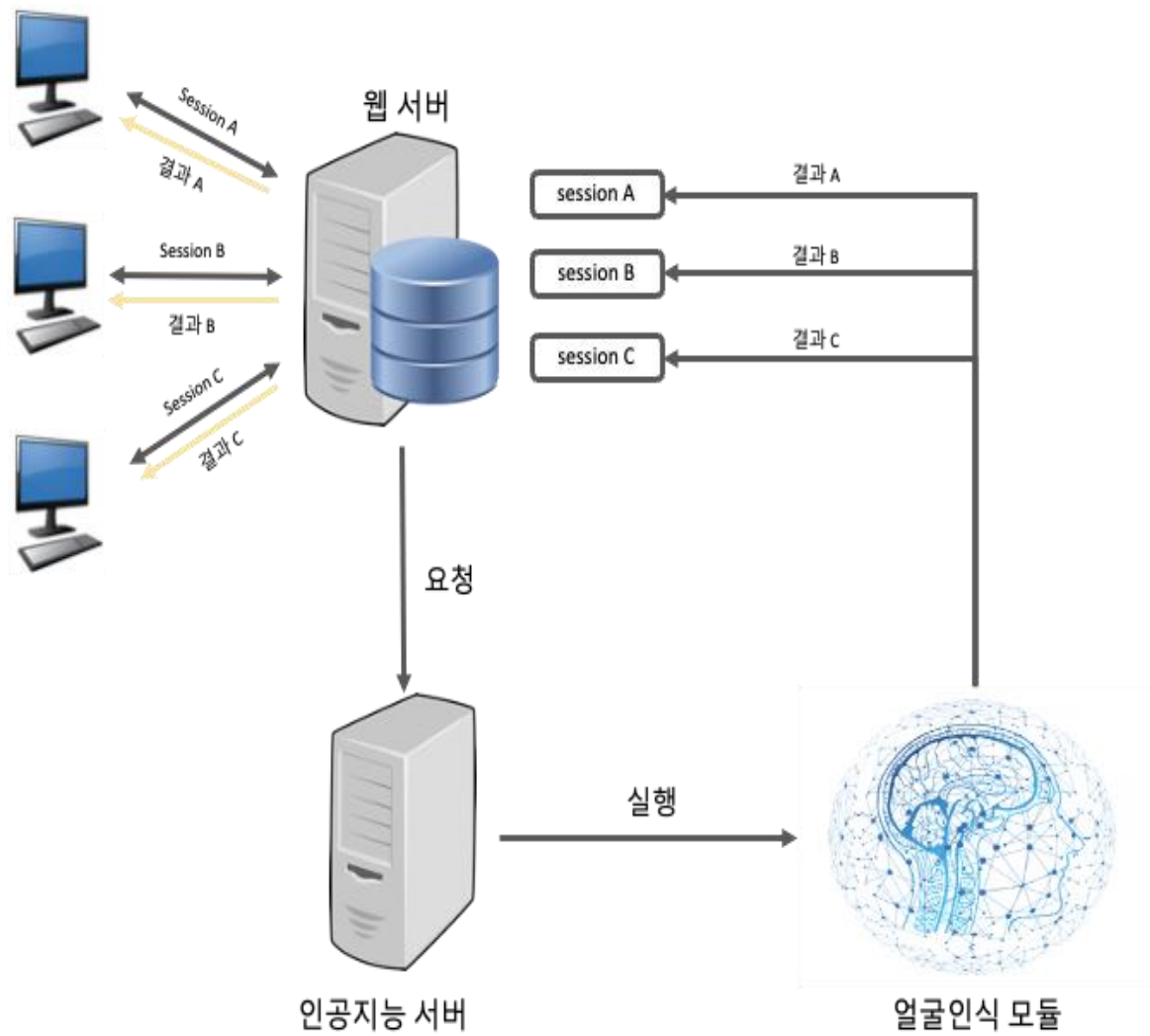


Figure 13. 클라이언트 구분.

웹 서버 및 클라이언트 : <https://github.com/kimtaejun97/pidetction-webServer>

AWS EC2에 웹 서버를 올렸고, 김태준 학생의 개인 컴퓨터의 port를 열어 인공지능 서버 환경을 구성했다. 학내 배포로 주변 지인들의 반응 및 피드백을 얻었다.

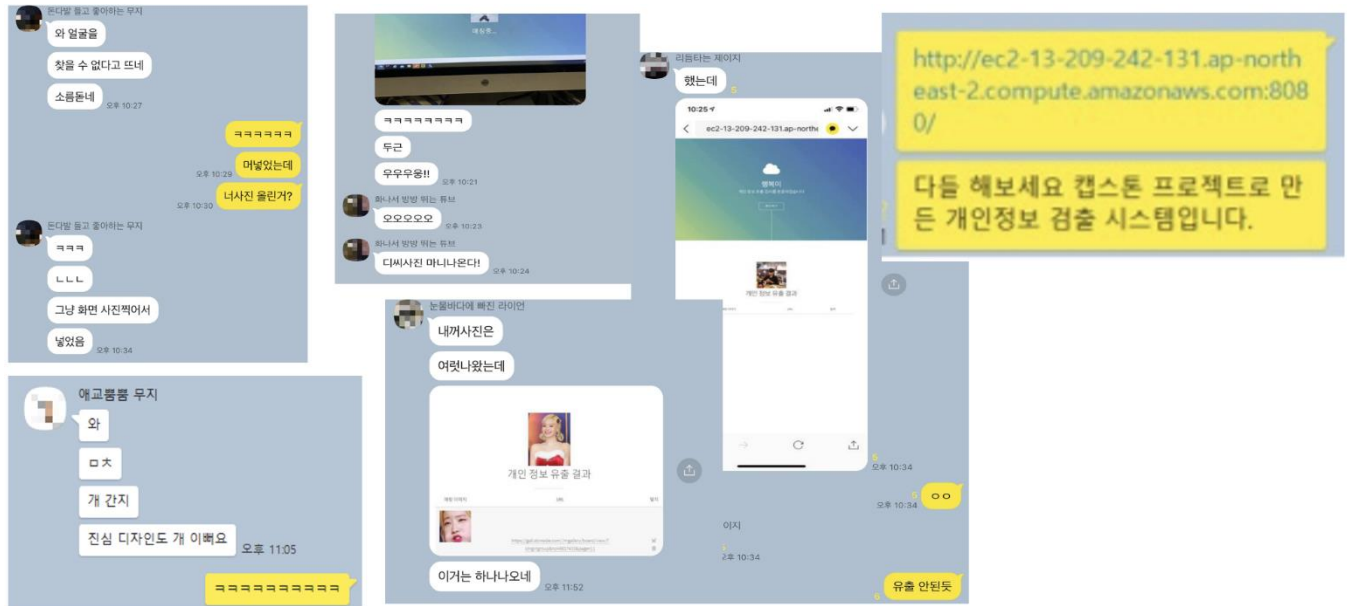


Figure 14. 배포.

### <기업연계 프로젝트>



Figure 15. 기업 멘토님과의 미팅.

기업 (주) 카라멜라의 요구로 기업 연계 프로젝트를 수행했다.

기업 멘토님과 주기적인 미팅 및 연락을 통하여 프로젝트를 진행했고 프로젝트에 대한 피드백 및 크롤링에 관한 조언을 받았다.