붙임3

/작품 최종보고서

2022 학년도 제 2 학기

캐시 헌터: 컴퓨터 비전 기반 수익형 광고 어플리케이션

최시열 (2019315530)

2022 년 10 월 24 일

지도교수: 우 홍 욱

■ 요약

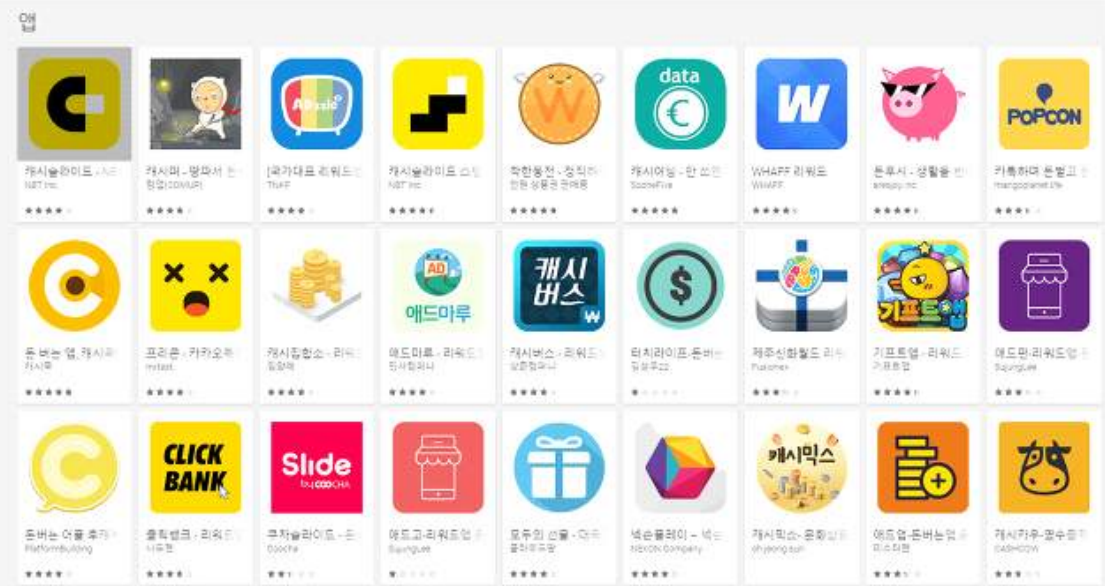
Computer Vision 기반 수익형 광고 어플리케이션 "캐시 헌터"는 사용자가 실생활에서 발견한 광고를 핸드폰으로 찍어 서버에 전송하면, 추후 현금화 가능한 소액 리워드를 제공하여 능동적인 광고 노출을 유도하는 어플리케이션이다. 업로드 한 사진에 인정 가능한 광고가 존재하는지 여부는 openCV의 FLANN기반 Feature Matching 알고리즘을 사용해 판단을 한다. 어플리케이션의 프런트 엔드는 Flutter의 Dart를 기반으로 제작되기에 iOS와 Android, 그리고 Web에서 동시에 build가 가능 하다. 백엔드는 Amazon Web Service (AWS)의 RDS와 Elastic Beanstalk을 사용해 Database 구축 및 Python Flask Module을 관리하고 있다.

■ 서론

가) 제안 배경 및 과제의 필요성

[제안 배경]

코로나 19의 영향으로 사람들의 핸드폰 사용시간이 폭발적으로 증가하며 캐시 슬라이드와 같은 리워드 어플리케이션들의 매출액이 눈에띄게 증가하였다. 실제로 캐시슬라이드를 개발한 모바일 광고 플랫폼 업체 엔비티는 2021 매출액이 776억원으로 전년동기 대비 75.9% 증가하였고, 영업이익 또한 34억원으로 흑자전환되었다. 또한 2022년도 매출액 또한 1015억원과 영업이익 79억원으로 전망되면서 캐시 리워드 어플리케이션에 대한 수요 및 관심이 폭발 적으로 증가하는 추세를 보였다. 그림 1과 같이 구글 플레이스토어, 애플 앱스토어 등에서도 리워드 어플리케이션이 상위권을 차지하면서 점점 많은 사람들이 다운로드를 함을 확인 할 수 있었다.



<그림 1> 구글 플레이스토어 리워드 어플리케이션 상위 목록

[사례 분석 - 캐시슬라이드와 캐시 워크]

리워드 어플리케이션을 통한 광고 방식은 소비자가 스스로 광고를 노출시키고, 그에 해당하는 보상을 받는 방식이다. 실제로 앱 스토어와 플레이스토어에서 상위권을 차지한 캐시 슬라이드 <그림 2>와 캐시 워크 <그림 3>의 사례를 보면, 캐시 슬라이드는 소비자가 광고를 시작화면에 의도적으로 노출시키고, 그 광고를 슬라이드를 통해 없애면 없앤 횟수만큼 소액의 리워드를 제공하는 방식이다. 또한 캐시 워크는 과거 만보기의 기능을 통해 본인이 움직인 횟수만큼 리워드를 제공하는 방식이며 움직이고 몇 보를 걸었는지 확인하는 과정에서 본인 스스로를 광고에 노출시킨다. 이처럼 단순히 광고를 넘기고싶어하며 반감을 사는 기존의 방식과는 다르게 본인이 조그마한 액티비티와 함께 스스로를 노출시키는 방식을 활용하는 방식이 대두됨에 따라 캐시 헌터에서는 카메라를 통해 사진을 찍고, 그 사진을 올리는 과정을 통해 광고에 노출시키는 전략을 택하게 되었다.

<그림 2,3> 캐시 슬라이드(좌)와 캐시 워크(우)

[광고 효과]

기존의 리워드 어플리케이션와 같이 능동적인 활동을 통해 광고를 광고 소비자가 직접 능동적으로 접하게 할 수 있으며, 광고를 기다리게 한다는 점에서 향상된 광고 효과를 기대할 수 있음. 광고 소비자의 입장에선, 본인이 원하지 않는 상황에서 노출되는 광고에 가장 부정적인 느낌을 받게 되는데, 해당 어플리케이션을 통한 광고 노출은 본인이 직접 광고에 노출하려 노력을 하는 기존의 패러다임과는 전혀 다른 접근 방식이다. 추가적으로, 1원,2원등의 소액이지만 본인의 노력에 대한 보상이 주어진다는 것은 소비자로 하여금 부정적인 심리를 갖지 않게 하며 광고 기업 이미지에 긍정적인 영향을 미칠 수 있다.

[데이터 수집]

현대 사회에서 데이터는 금과 같다라는 말이 있을 정도로 유의미한 데이터를 수집하고 그것을 가공할 수 있는 기술은 매우 중요하다. 많은 광고 투자자들은 본인의 광고가 가장 적은 비용으로 최대의 효율을 낼 수 있는 방법을 모색하기 위해 많은 투자를 아끼지 않는데, 해당 어플리케이션을 이용하여 광고 데이터를 분석할 수 있다. 가령, 20대 남성 이용자가 가장 많이 촬영을 해서 업로드한 광고가 무엇인지, 어느 위치에서 찍은 광고가 가장 빈번했는지 등, 다양한 데이터를 축적할 수 있게 된다. 본인의 광고를 해당 어플에 올리기로 한 기업들에게는 Monthly, Weekly feedback을 통해 어느 위치와 어떠한 방식으로 광고를 하는 것이 가장 효과적인지 분석을 해주어 더 효율적인 광고 기법과 투자를 할 수 있게 한다.

나) 연구논문/작품의 목표

[Hybrid Application]

기존의 유지보수가 어려웠던 iOS, Android를 각각 따로 개발하고 유지/보수를 하며 운영하는 방식에서 벗어나 하나의 언어로 개발하고 두 플랫폼용으로 빌드만 두 번 하는 Hybrid Application을 사용해 개발,유지,보수 과정을 간편화 한다. 실제로 React Native Framework를 개발한 Meta(구 Facebook)사는 본인들의 앱들을 Native에서 React Native로 모두 Migration을 하였다. 물론 Native앱 보다 성능이 일부 떨어지는 가능성이 존재 하지만, 대부분의 Hard한 연산은 Server단에서 진행되기 때문에 사용자 입장에서 불편함을 느낄 정도의 속도차이는 보이지 않을 것으로 예상된다. 해당 개발 방식은 대기업보다는 인력이 제한적인 스타트업들에서 적용이 되어, 향후 해당 기술을 필요로 하는 기업들에게 좋은 Reference가 될 것으로 예상된다.

[DevOps]

DevOps의 관점에서 Computer Vision 관련 기능이 들어간 모바일 어플리케이션을 설계하고 기획 할 때에는 어떠한 요소를 고려해야 하는지 배울 수 있음. 기업들에선 본인들의 기존 product에서 축적된 Data를 통해 본인들의 상품에 맞는 모델을 학습하고, 그 데이터를 바탕으로 더 좋은 제품을 만드는 과정에 있다. 이때, 이론을 상품으로 만드는 과정에든 반드시 Operator가 필요하게 되는데, Development 와

Operation모두가 가능한 Devops가 필요 하게 된다. 그러나 단순한 DevOps가 아닌, Computer Vision 전반에 관한 이해가 있고, 이해를 완료한 API를 실제 상품에 적용에 이르는 상품화 과정을 이해하고, 문제가 될 수 있는 부분들에 대한 경험이 있고 Error handling 경험이 있는 인력이 필요하게 되었다. 이에, 이러한 Full stack Development를 넘어선, Computer Vision 관련 Product까지 제작해보며, 제품 설계에 어떠한 부분을 염두해 두어야 하고, 어떠한 문제들에 봉착하는지 등 다양한 축적하는 것은 향후 DevOps 직무를 이해하는데에 큰도움이 될 것 같아 해당 프로젝트를 기획하게 되었다.

[Real-time Computing]

실시간으로 Computer VIsion기술을 적용해야 하고, 데이터베이스 전체를 탐색하며 각각의 이미지와 input 이미지를 대조해, 일치하는게 있는지 일일이 Prediction하는 과정을 거쳐야 하기 때문에, 모델의 크기가 커지게 되면 곤란한 상황이 발생 함. 모델 경량화에 초점을 두고, 모델의 크기를 줄여 실시간 처리를 용이케 하기 위해서는 어떠한 접근이 필요한 지에 대해 연구를 함을 통해, 향후 Light weighted model for Edge AI 연구에 도움을 줄 수 있다. 현대 IT기기들이 점차 소형화 되고, AP에서 많은 처리를 하기 시작하였기 때문에, 패턴매칭과 같이 기본적이고, 많은 분야에서 사용되는 Computer Vision모델이 경령화가 효과적으로 진행이 된다면, 향후 Edge Device에서 Prediction을 진행하는데에 아무런 문제가 없을 수 있음. 현재로서는, React Native, Flutter등의 플랫폼에서는 Computer VIsion관련 모듈이 존재하지 않고, Python 파일 자체를 어플리케이션에서 실행을 하는 권한이 앱스토어용 어플리케이션 개발자에게는 주어져 있지 않기 때문에, 현재로서는 스스로 만든 인공지능 모델을 활용하기 위해서는 인공지능 모델 API용 Backend 서버를 따로 두고, 그 서버에 처리할 Input값을 넘기고, 결과가 나오면 그것을 돌려받는 System Architecture가 유일 하다. 하지만, 핸드폰들의 사양이 더 좋아지고, 핸드폰과 같은 Edge Device에서 Custom AI model을 Prediction할 수 있는 상태가 된다면, 해당 기술이 각광받을 것 같다.

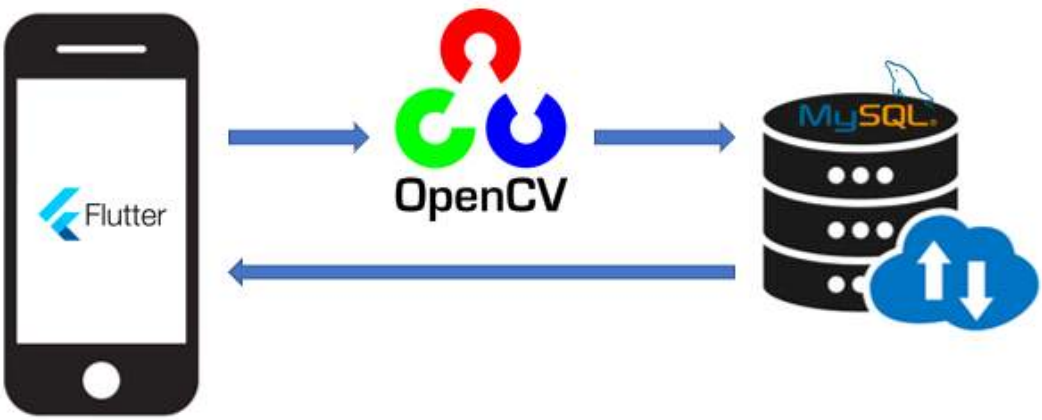
[Microservice Architecture]

기존의 Monolithic Architecture는 하나의 서버에서 모든 것을 처리하기 때문에 개발은 쉬울지 몰라도 유지/보수의 관점에서는 매우 불편하다는 관점이 있다. 그러나 해당 작품에서는 Back-end server를 각각의 서비스 별로 서버를 분리하여 안정성을 확보하고, 하나의 서버가 다운이 되더라도 Disaster Control이 쉽도록 하는 것이

목표이다. 이는 Microservice Architecture라는 것으로, 소프트웨어가 잘 정의된 Application Programming Interface를 통해 통신하는 소규모의 독립적인 서비스로 구성되어 있는 환경에서의 개발을 위한 아키텍쳐 및 조직적 접근 방식이다. 이는 민첩성, 유연한 확장성, 손쉬운 배포, 기술적 자유, Reusable Code, 복원성 등을 가질 수 있기에 Monolithic Architecure에 비해 높은 자유도와 선호도를 갖고 있다.

추가적으로, 이미지 처리를 비롯한 다량의 Request가 한번에 쏟아지기 때문에 적절한 Reverse proxy Engine과 Middleware 선택이 매우 중요하다. 또한 지속적인 유지보수 그리고 배포를 위해 자동화 툴을 적용할텐데 에는 Jenkins, Docker, Kubernetes등의 프레임웍의 도움을 받도록 할 것 이다. 이를 통해 Tech 기업 전반의 서비스가 어떤식으로 운영 되는지에 대한 탐구가 가능하다.

다) 연구논문/작품 전체 overview



<그림 4> 어플리케이션 시스템 아키텍쳐

- 중간 개발 결과, 어플리케이션에서 사진을 찍고, 사진 일치여부를 판단한 후 MySQL에 저장하는 단계까지는 완성함.
- Flutter를 통한 어플리케이션 프런트앤드를 제작해 iOS, Android, Web Browser 모든 환경에서 구동이 가능하도록 제작을 하였다.
- 촬영된 이미지를 openCV 라이브러리를 활용한 Flask 기반 백엔드 연산 서버에서 DB내 광고와 일치 여부를 확인 한다.
- 일치 결과를 MySQL 데이터 베이스에 저장한 뒤 프런트엔드에서 요청이 오면 이를 화면에 보여준다.
- 현재 프런트엔드의 디자인적인 요소는 Figma를 통해 제작이 되었으며 사용자가 보기 편한 인터페이스를 제공하기 위해 User Experience를 고려했다.

- 백엔드 부분의 로드밸런서
- 대해서는 보완이 필요하며, 백엔드 부분의 로드 밸런서 및 다수의 요청을 처리할 수 있는 미들웨어 보완이 필요함

■ 관련연구

[클라우드 서비스의 Disaster Control 및 Load Balancer]

Web기반 서비스의 규모가 확장되어 일반적인 규모의 기업이 감당하기에는 많은 사용자들이 사용을 하게되며 분산 컴퓨팅을 효과적으로 할 수 있는 니즈가 늘어나게 되었다. 이에, IT 기업들은 Cloud Computing 이라는 이름으로 Client들이 만든 프로그램을 호스팅 해주는 분산 클라우드 컴퓨팅 서비스들을 시작하게 되었다. 대표적인 예로는 Amazon 의 Amazon Web Service, Google의 Google Cloud Computing, Microsoft 의 Azure등이 있다.

이러한 클라우드 컴퓨팅 서비스들은 단순히 고성능 컴퓨팅 엔진을 제공을 함에 그치지 않기 때문에 차별점을 갖는다. 우선 서버가 다운되지 않고 하드웨어적으로, 소프트웨어적으로 견고한 서비스를 지속시킬 수 있는 시스템 장애 극복 시나리오를 갖고 있기 때문에, Client들로 하여금 걱정 없이 구독료를 지불만 함으로 서비스를 지속가능하게 유지시킬 수 있다. 추가적으로, 단순히 구동만 계속 시키는 것이 아니라, 효율적으로 그리고 빠르게 서비스를 제공할 수 있어야 하는데 분산컴퓨팅 기술과 모니터링 기술을 통해 이를 가능케 하였다. AWS에서는 부하분산 코디네이션 서비스를 활용해 분산 서버의 계층별 크러스터링과 모니터링 서버 및 오토스케일링을 가능하게 하였다.

기존에는 하나의 하드웨어 서버 머신을 이용하기 때문에 Monolithic Architecture를 많이 사용했었지만 이제는 하드웨어적으로도 각각이 효과적인 머신을 달리 쓸 수 있기 때문에 MicroService Architecture를 선호하는 경향이 늘었다. 예를 들어, 이존에는 하나의 머신에 연산을 위한 고성능 CPU, 저장을 위한 대용량 HDD, 병렬연산을 위한 GPU등을 모두 탑재를 해야 했었다면, 이제는 데이터 저장을 위한 데이터 센터 전용 서버에는 데이터베이스를 저장하고, 순차적인 연산이 필요한 작업에는 고성능 컴퓨팅 엔진이 있지만 용량은 적은 EC2와 같은 CPU를 사용하며 해당 작품에서처럼 이미지 프로세싱등의 병렬작업이 필요한 프로세스의 경우 고성능의 GPU가 탑재된 서버에서 컨테이너를 탑재하는 등의 각각의 의도에 맞는 여러개의 서버를 동시에 이용할 수 있다는 장점이 있다. 이러한 Micro Service Architecture와 같은 경우 여러개의 서버에 물리적으로도 분산이 되어있기 때문에 재난상황에

조금 더 Robust하다는 특징을 보인다.

AWS가 제공하는 Elastic Beanstalk의 최대 장점은 고성능 로드 밸런서가 존재한다는 점이다. 해당 엔진은 Application Load Balancer를 활용하는데 이는 크게 3가지 부분으로 이루어져있다. 먼저 로드밸런서는 클라이언트에 대한 단일 접점 역할을 수행한다. 이는 여러개의 가용영역에서 EC2 인스턴스 같은 여러 대상에 수신 어플리케이션 트래픽을 분산함을 통해 어플리케이션의 가용성을 향상하는 방식을 택하였다. 이때, 하나의 로드 밸런서에 다수의 리스너를 추가할 수 있는데, 리스너란 구성한 프로토콜 및 포트를 사용하여 클라이언트의 연결 요청을 확인 하는 기능을 한다. 리스너에는 우선수위, 작업, 조건등으로 구성된 규칙에 따라 로드 밸런서가 등록된 대상으로 요청을 라우팅하게 된다. 이렇게 리스너를 통해 들어온 리퀘스트는 타겟 그룹이란 그룹단위로 지정된 프로토콜과 포트 번호를 사용해 EC2 인스턴스에 요청을 라우팅 하게 된다. 이러한 모든 과정을 AWS의 모니터링 툴을 통해 서버 컨디션을 체크해주며 이상 상태가 발생 할 경우 알람을 보내 주는 등의 조치를 취해 클라이언트가 즉각의 조치를 취할 수 있도록 한다. [5]

[Cross Platform Mobile Application]

과거에는 Android는 Java/Kotlin 으로, iOS는 Swift로, 각각 Native app을 개발하였으나, React native, Flutter라는 Cross Platform Mobile Application development 언어가 등장함에 따라 한가지의 언어로 개발 및 유지 보수를 해 관리가 용이하게 되었다. React Native는 Meta 사에서 JavaScript를 기반으로 만들어진 언어이며, Flutter는 Google사에서 Dart(Kotlin과 비슷한 문법)이라는 새로운 언어를 통해 만들졌다.

동일환경에서 Flutter와 React Native로 Build된 어플리케이션의 benchmark를 테스트 해봤을 때의 결과는 다음과 같았다.

1) CPU 사용량 측면
Webview를 오픈 할 때에는 FLutter가 9.6%의 CPU를 더 사용하였고,
Listview를 Render할 때에는 React Native가 4.5%의 CPU를 더 사용하였고,
Data Filtering에는 React Native가 14.8%의 CPU를 사용하였다.
2) Memory 사용량 측면
Webview를 오픈 할 때에는 React Native가 20.14MB 메모리를 더 사용하였고,
Listview를 Render할 때에는 Flutter가 2.8MB 메모리를 더 사용하였고,
Data Filtering에는 React Native가 2.06MB 메모리를 사용하였다. [2]

추가적으로, React Native가 Device에서 어떻게 동작을 하는지 알아본 결과, Device의 OEM Widget과 Services들을 접근하는데에는 JavaScript 코드를 Bridge를 거쳐 접근을 해야 했다. 이는, 사용자가 화면에서 스와이프를 한번 진행했을 때 초당 60회 정도 접근을 해야 하는 상황도 오는 상당히 Overhead가 큰 작업이다. 반면 Flutter는 Dart Native Code영역에서 Widget, Randering, Platform Channel등을 모두 지원을 하고, Device 접근시 별도의 Bridge과정 없이 바로 Canvas와 Event, Services등에 접근이 가능하다. [4]

Android 운영체제를 만든 Google사에서 만든 Flutter가 해당앱에서 필수적으로 요하는 기능들에서 더 높은 벤치마크를 보여주었고, 언어의 설계 구조가 안드로이드 기기에서 동작을 할 때에 더 최적화 되어있기 때문에 필자는 이번 프로젝트에서 Flutter를 프론트엔드 개발언어로 채택을 하게되었다. Native 앱이 아니다보니, Hardware Resource를 접근할 때에 접근권한 이슈가 발생을 하는 일부 부분도 있지만, 대부분이 해결이 되어 실제 개발에 들어갈때에는 아무런 문제가 없을 것으로 보인다. 해당 어플리케이션 개발경험을통해 하이브리드앱으로 만들어진 어플리케이션이 각각의 생태계에서 어떤식으로 동작으로 하며, DevOps의 관점에서 Native 앱으로 만들었을때와 Hyrbid 앱으로 만들었을때의 Trade Off를 분석하여 향후 프로젝트에서 결정을 내릴 때 큰 Reference가 되길 희망한다.

[Computer Vision - Feature Matching]

이번 어플리케이션 제작에서 핵심이 되는 부분은 Client가 카메라를 통해 찍은 사진 안에 인정이 되는 광고가 존재하는지를 판단하는 부분이다. 이에, Computer Vision 기술 중 하나인 Feature Matching 사용이 필수적이다. Feature Matching이란, 서로 다른 두 이미지에서 Keypoint와 Feature descriptor를 비교해 비슷한 객체끼리 짝짓는 것을 말한다. 여기에서 Keypoint란, 이미지에서 특징이 되는부분을 의미하며, Feature Descriptor란 Keypoint 주변 pixel을 일정한 크기의 블록으로 나누어 각 블록에 속한 pixel의 Gradient Histogram을 연산해 낸 것이다.

이때, 주로 사용되는 Feature Descriptor에는 세가지가 존재한다.

1) SIFT(Scale-Invariant Feature Transform)에 기반한 Feature Descripting 알고리즘은 이미지 피라미드를 이용해 기존 Harris Corner Detector가 갖고 있던 Size Invariance를 해결하지 못했다는 문제를 해결한 알고리즘
2) SURF(Speeded Up Robust Features)에 기반한 알고리즘은 SIFT가 속도가 느리

다는 단점을 해결하기 위해 이미지 피라미드를 사용하지않고 필터의 크기를 변화하는 방식으로 성능을 개선한 알고리즘
3) ORB(Oriented and Rotated BRIED)는 Keypoint를 검출하지 않는 Descriptor인 BRIEF에 기반했으며, 속도가 매우 빨라 SIFT와 SURF의 대안으로 사용되는 알고리즘 [1]

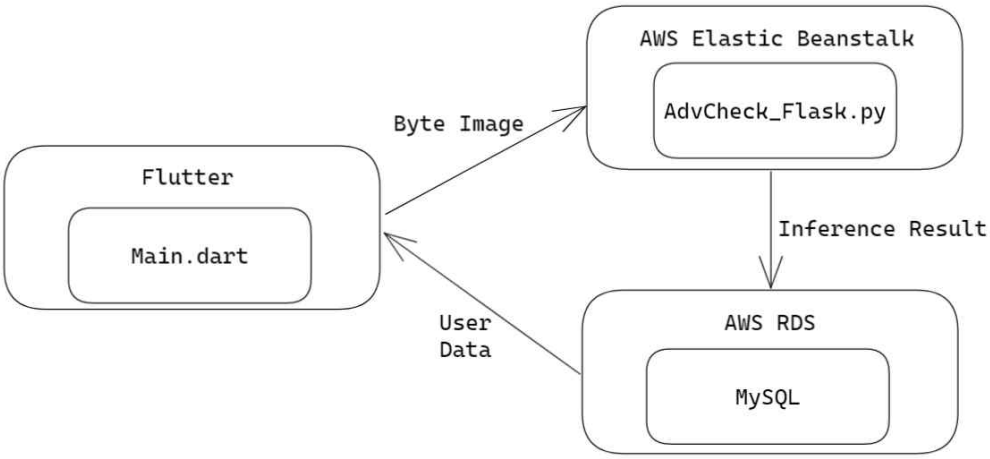
위의 Feature Descriptor를 사용하는 Brute Force기반 Matching 기법은 속도가 매우 느려, FLANN(Fast Library for Approximate Nearest Neighbors Matching)을 사용하는 추세이다. FLANN은 모든 Descriptor를 조사하지않고 이웃하는 Descriptor끼리 비교를 하기 때문에 속도가 빠르며 openCV에서 라이브러리로 지원을 하고 있다.

이처럼 오랜시간 발전을 거친 Feature Matching 알고리즘을 사용해, Input 이미지에서 물체가 이동을 하더라도 그것의 Feature를 Detect 해내고 회전이 되어있더라도 그것을 문제없이 Detect 할 수 있도록, Invariant함을 확보하여 해당 어플리케이션에 적용할 계획이다.

■ 제안 작품 소개

[System Architecture]

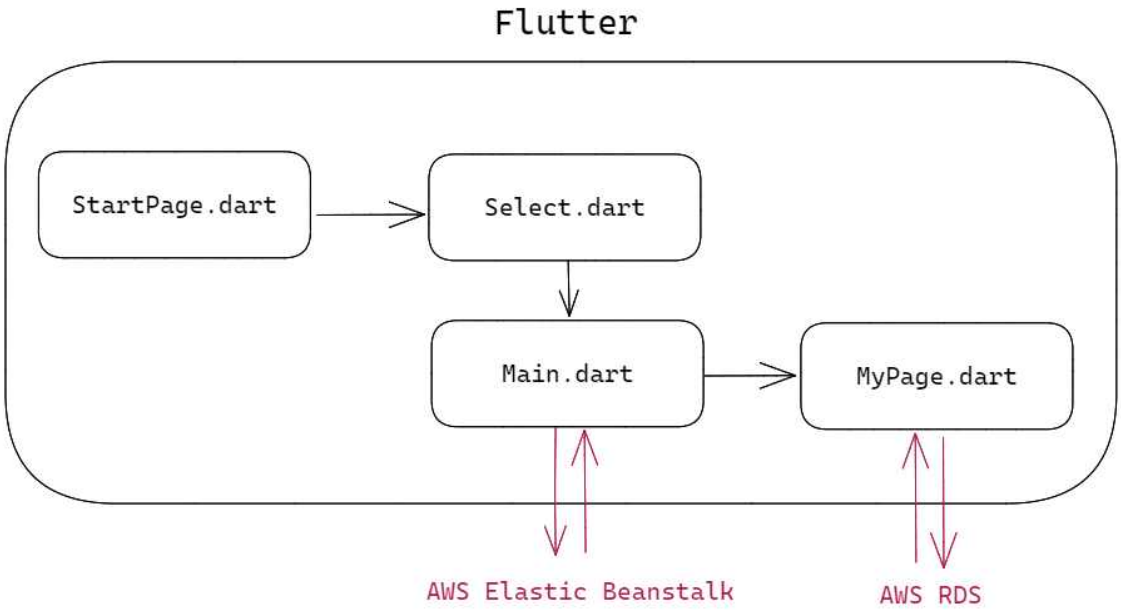
1) 시스템 전반



<모식도 1> High Level Data Flow

- Flutter의 Main.dart에서 Byte로 Encoding된 이미지가 AWS서버로 전송이 된다.
- AWS Elastic Beanstalk에 올려져있는 AdvCheck_Flask.py 파일이 openCV를 통한 Inference를 진행한다.
- Inference 결과는 MySQL기반 AWS RDS 서버에 저장이 된다
- 저장된 결과는 나중에 HTTP 통신을 통해 마이페이지에서 로딩 된다.
- Flutter, Elastic Beanstalk, RDS 내부가 어떤식으로 동작 되는지 아래에 있는 그림 3개를 통해 설명하였다.

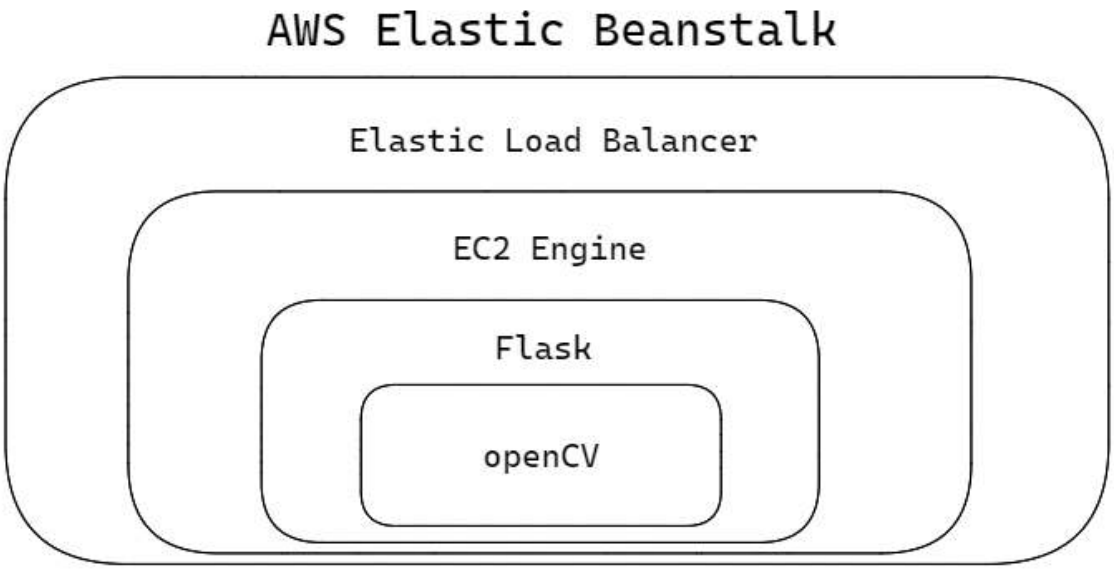
2) 프런트엔드 구성



<모식도 2> Front-End Page Flow

- 첫 화면인 StartPage에서는 앱의 이름인 Cash Hunter라는 글자를 보여주며 스플래시 화면의 기능을 수행한다.
- 다음 화면인 광고 선택 화면에서는 DB에 존재하는 광고 리스트를 Scrollable로 보여주며 유저로 하여금 선택을 할 수 있도록 한다.
- 이후 메인 화면인 사진 촬영 및 업로드 화면에서는 카메라 제어를 통해 사진을 찍은 뒤 해당 사진을 실시간으로 서버에 전송을 하게 된다.
- 해당 결과는 Pop Up 기능을 통해 화면에 표시되며 자동으로 마이페이지로 이동하게 된다.
- 마이 페이지에서는 지난 광고 헌트의 결과, 현재 누적 포인트, 그리고 환전 페이지를 보여주며 다시금 Select 화면으로 돌아갈 수 있도록 한다.

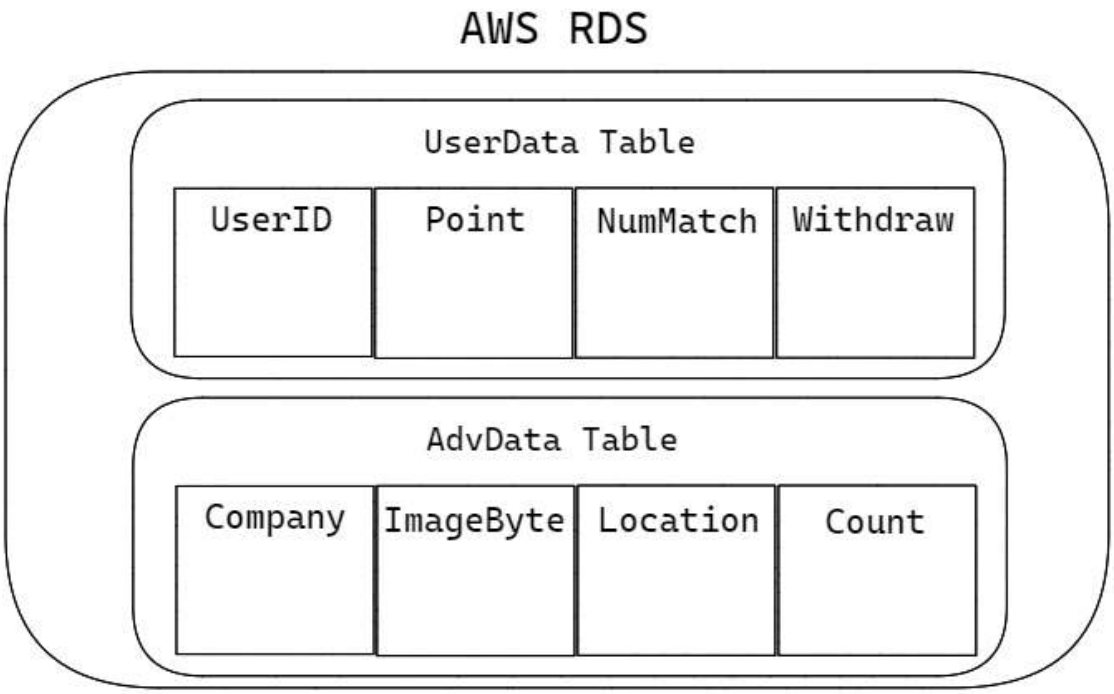
3) 백엔드 구성



<모식도 3> Back End Hierarchy

- openCV FLANN 라이브러리를 불러오는 Python파일을 배포하기위해 AWS Elastic Beanstalk을 활용한다.
- openCV를 구동시키는 AdvCheck_Flask.py는 Flask를 통해 5000번 포트의 웹서버로 구동이 되며 '/' 랜딩 페이지에서 HTTPrequest를 기다린다.
- 해당 파일을 로컬에서 구동하기보다는 24시간 관리가 되는 AWS엔진에서 구동을 하기 위해 해당 코드를 EC2엔진에 올려 구동시킨다.
- Flask의 Multithread기능을 통해 병렬처리를 가능하게 하더라도 openCV inference는 로드가 큰 작업이기에 다수의 동시다발적인 요청이 오게 된다면 심각한 latency를 초래할 수 있다.
- 이에 AWS Elastic Beanstalk에서 제공되는 Elastic Load Balancer를 활용해 동시다발적인 리퀘스트를 비동기 처리가 가능하게 되었다.

4) 데이터베이스 구성



<모식도 4> Database Architecture

- AWS의 관계형 데이터 관리를 위한 기능인 AWS RDS를 활용해 데이터베이스를 관리한다.
- 두 개의 Table이 존재하며 하나는 사용자의 데이터를 기록하기 위한 테이블이며, 다른 하나는 광고 데이터를 활용하기 위한 테이블이다.
- UserData 테이블은 유저들의 데이터를 기록하고 관리하는 곳으로 각 유저가 현재까지 몇 번의 시도를 통해 얼마의 포인트를 축적했는지 등을 관리하는 곳으로 UserID, Point, NumMatch, Withdraw등을 개별 Column으로 관리한다.
- UserData 테이블은 민감한 프라이버시 데이터가 저장이 되기 때문에 암호화 처리를 추가할 계획이다.
- AdvData 테이블은 유저가 선택한 광고를 검색해 해당 이미지를 불러오는데에 활용하게 되며 추후 데이터 통계를 위해 쿼리를 한 유저들의 통계를 기록으로 남겨 광고 데이터 분석에 활용할 수 있도록한다.
- AdvData에는 4가지의 Column이 존재하며 각각 Company, ImageByte, Location, Count등을 기록하여 관리한다.

[핵심 알고리즘]

1) FLANN

먼저 OpenCV 라이브러리로부터 제공되는 SIFT detect 알고리즘을 불러와 img1과 img2에 적용을 해준다. 이후, FLANN_INDEX_KDTREE 알고리즘을 활용해 몇 개의 Match가 존재하는지 파악을 한다. 이때 Threshold를 정해주어야 하는데, 각 match별 distance의 크기가 30% 이내로 차이 날 경우 good으로 간주한다. 이때 good의 비율과 total의 비율을 비교하여 good/total이 10%가 넘어갈 경우 Match로 간주하였다.

```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
good = []
total = []
for m,n in matches:
   total.append(m)
   if m.distance <0.7*n.distance:
     good.append(m)
if len(good)/len(total)*100>MIN_MATCH_COUNT:
   return True
```

<소스 코드 1> FLANN 기반 Feature 매칭 알고리즘

2) HTTP 통신

Flutter와 Flask간의 Http 통신을 해 촬영된 파일 이미지를 전송하는 function을 async를 활용하여 비동기 처리가 가능하게 작성하였다. JSON안에 file을 byte 형식으로 전송하며 Dio라는 패키지를 활용하여 Http 통신을 API를 활용해 쉽게 진행하였다.

```
void _uploadFile(filePath) async {
  String fileName = basename(filePath.path);
  try {
   FormData data = FormData.fromMap({
```

```
    "file": await MultipartFile.fromFile(
     filePath.path,
     filename: fileName,
    ),
   });

   Dio dio = new Dio();
   await dio.post("http://#######:5000/", data: data)
     .then((response) => print(response))
     .catchError((error) => print(error));
  } catch (e) {
   print("Exception Caught: $e");
  }
}
```

<소스 코드 2> Dio를 활용한 Front-Back End간 통신 알고리즘

3) Database 업데이트

Flask 백엔드 서버와 MySQL 데이트베이스 서버를 연결하기위해 PyMySQL 라이브러리를 활용해 해당 연결을 시켜주었다. FrontEnd로부터 넘어온 새로운 유저의 활동 데이터 삽입과 기존 데이터 업데이트를 위해 SQL query의 Insert와 Update를 진행해주었다.

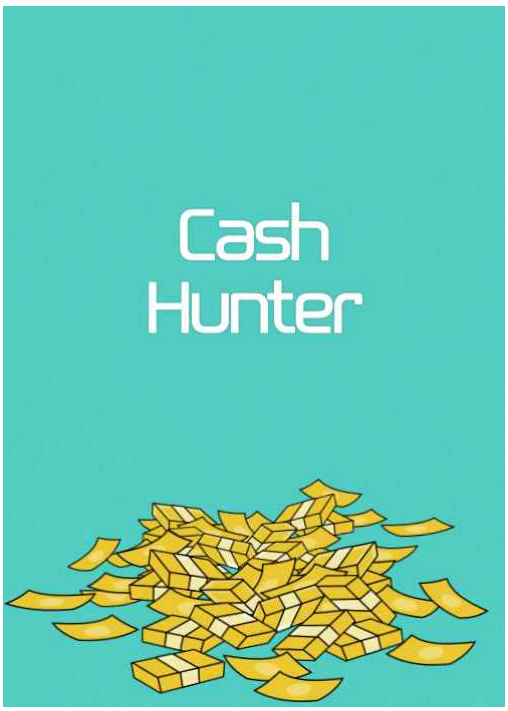
```
db = pymysql.connect(host="*.*.*.*", user="root", passwd="1234", db="free_board", charset="utf8")
cursor = db.cursor()
req = request.get_json()
uid = req["uid"]
time = str(datetime.datetime.now())

db=db.connection()
query_insert = '''
INSERT INTO user_log (uid, time, point) VALUES(%d, '%s', %d);
''' % (uid, time, 2)

query_update = '''
UPDATE user_point SET point=point+1 WHERE uid='%s';
```

```
''' % (uid)

cursor.execute(query_insert)
cursor.execute(query_update)

db.commit()
```

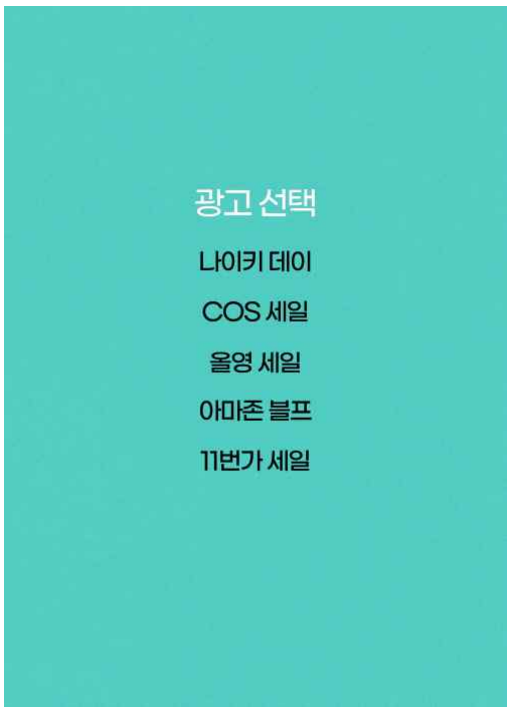
<소스 코드 3> PyMySQL 기반 데이터 처리 알고리즘

■ 구현 및 결과분석

[어플리케이션 화면]



<화면 1> 스플래시 스크린
StartPage.dart



<화면 2> 광고 선택화면
Select.dart

<화면 3> 사진 업로드
Main.dart

<화면 4> 마이페이지
MyPage.dart

[Object detection 모듈 성능]

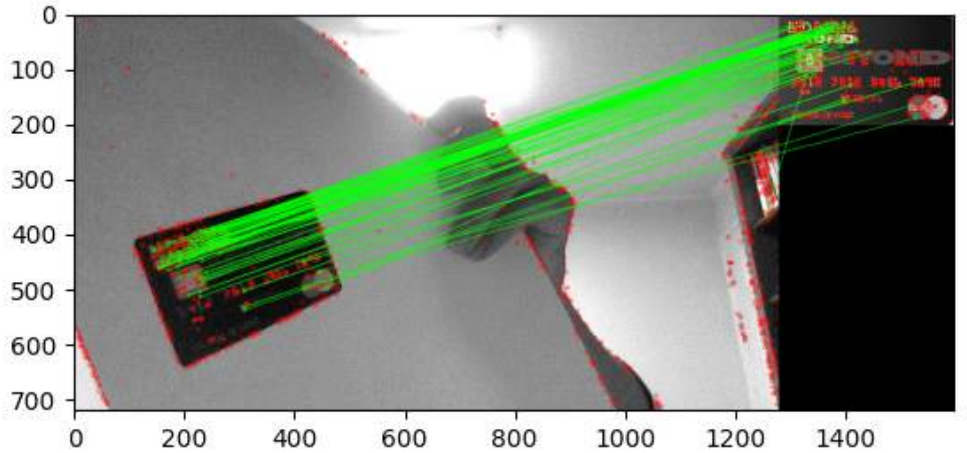
openCV에 구현되어 있는 FLANN기반 알고리즘을 통한 Feature Matching의 성능을 테스트 해 본 결과, 적절한 threshold를 설정할 경우 일치/불일치의 판단을 배포자가 원하는 수준으로 설정이 가능 하였다. 따라서 일치율을 하나의 external parameter로 설정해 두고, 코드의 재배포를 쉽게 할 수 있도록 이 상수 값 하나만 바꾸면 될 수 있도록 설정하였다. 현재 threshold는 distance 는 0.7, MIN_MATCH_COUNT는 10개로 설정이 되어있으며, 측면에서 찍힌 사진, 뒤집힌 사진 등 일반 사용자가 생각해낼 수 있는 edge case들은 모두 일치로 판단히 가능한 기준선이다. 만약 일치율이 너무 높다는 생각이 든다면 distance를 줄이고, MIN_MATCH_COUNT를 올려 더 정확한, 더 정면의, 더 완벽한 사진을 찍을 수 있도록 유도할 수 있다.

또한, FLANN과 같이 RULE BASED 모델을 사용하지 않고, 실제로 CNN모델에 광고 데이터를 학습시키고, 이를 tilt/rotate/flip등의 변형을 시킨 데이터를 학습 시키는

방식을 고안해보았지만, 이는 새로운 데이터가 들어올 때 마다 모델을 새로 학습시켜야 하는 단점이 존재하는등 조금 더 general purpose의 코드를 만들기에는 적절하지 못하다 생각이 되어 배제하였다. 또한, 해당 모델을 CNN모델에 학습을 시킬 때, 매번 overfitting/underfitting 그리고 Inference time을 고려하기 위한 parameter관리 등 너무 많은 작업이 요구 될 것으로 사료되어 FLANN 알고리즘을 채택하였다.

아래 그림 5는 카드사 광고로 자주 쓰이는 신용카드 광고 예시를 종이로 프린트하여 카메라로 찍은 사진과 카드 광고 자체를 FLANN으로 비교 한 결과로 뒤집어졌을 때, 틀어졌을 때, 휘어졌을 때 모두 "일치"라는 결과를 도출 하였다.

이 때, match와 total의 비율을 비교했을 때, 완벽하게 모든 글자가 일치하고 edge가 다수 등장할 경우 good/total의 비율이 약 0.2 내외의 결과를 보였다. 이에 match가 발생하지 않을 상황들을 상정하고 테스트를 해 본 결과 0.01 내외의 결과를 보이며 0.1 미만의 값을 항상 보임을 파악하였다.

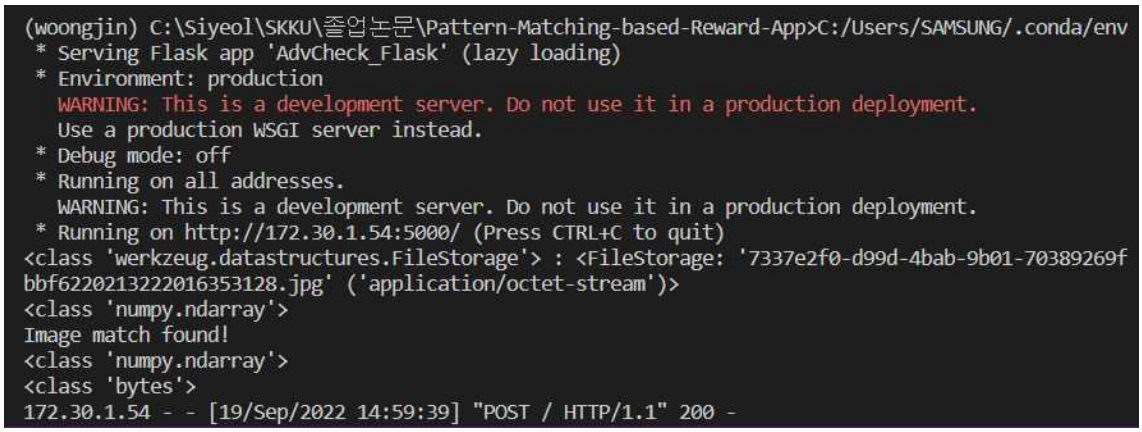


<그림 5> openCV FLANN match 결과 스크린샷

[Flask Backend Module 실행 결과]

현재 openCV모델은 python으로 적용된 라이브러리가 가장 간편하고 성능도 좋기에 python을 백엔드 언어로 채택하였고, 이 중 개발경험이 더 많은 flask를 사용하였다. 그러나, 비동기처리가 가능 한 Fast API라는 백엔드 파이썬 라이브러리가 최근 등장하여, 벤치마크를 비교하였을 때 flask의 multi_threaded = True 기능을 사용했을 때 보다 ~10배 더 빠르고 concurrent한 프로세스의 처리가 가능함을 확인하

여, 백엔드 코드 구성을 Fast API로 최종 변경 할 계획이었으나, Elastic Load Balancer에서 비동기 처리를 지원함을 확인 해 Flask 코드를 활용하더라도 Load balancer를 통한 연산처리를 할 경우 큰 문제가 없을 것으로 보여 Elastic Beanstalk을 채택하였다.



<그림 6> 로컬에서 Flask 구동 한 결과 스크린샷

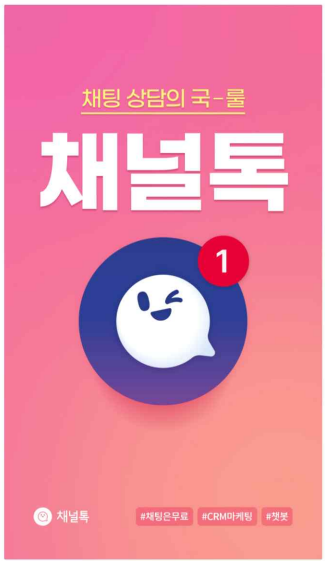
[AWS Elastic Beanstalk 배포]

AWS에는 Flask/Fast API/Django 등 단일 파이썬 파일의 load balancer를 제공해주며 해당 process의 상태를 체크 해주는 elastic beanstalk이라는 서비스가 존재하여 이를 적극 활용 하였다. 직접 GUNICORN, UWSGI등의 middleware를 사용하여 구현하는 것도 가능하였지만, 이는 실제 extreme situation등을 가정 해 다수의 트래픽을 발생시켜보는 테스트등을 통해서 parameter를 튜닝하는 과정을 거쳐야 하기에, AWS에서 존재하는 내장 load balancer instance를 사용하기로 하였다.

또한 AWS RDS라는 관계형 데이터 베이스 시스템 관리 툴을 이용하여, 3306 포트에서 DB를 열어 일치/불일치의 결과를 저장하여 훗날 데이터 분석이 가능하도록 하였다.

[Field Test]

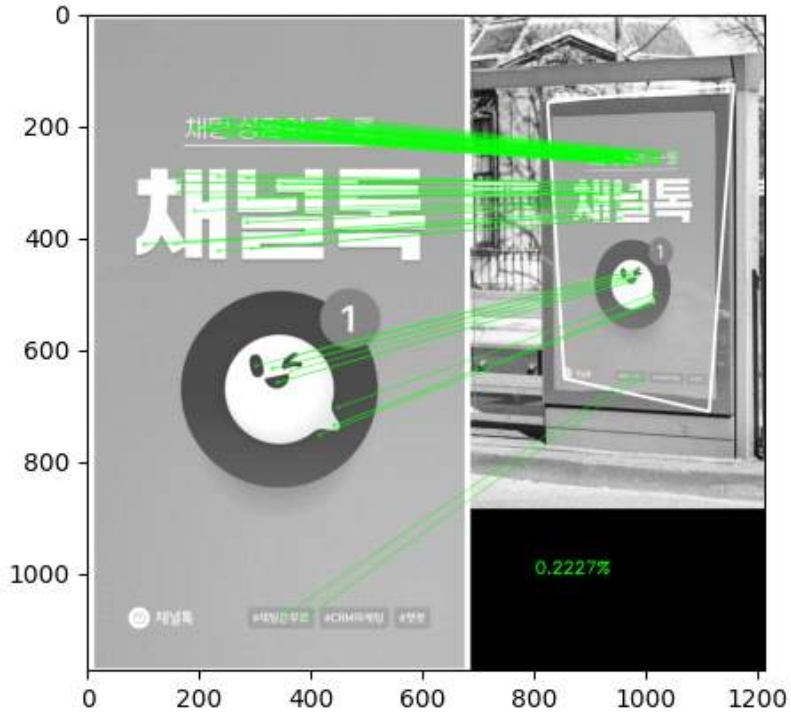
연출 된 사진이 아닌, 밖에서 실제로 찾을 수 있는 옥외 광고에 테스틑를 해보기 위해 Field Test를 진행해 보았다. 실제 사람들이 많이 접할 만한 옥외 광고판은 주로 버스 대기소 광고판이었기 때문에 해당 광고를 적극적으로 활용한 '채널톡'이라는 광고에 테스트를 해보았다.



<그림 7> 채널 톡 그림 원본

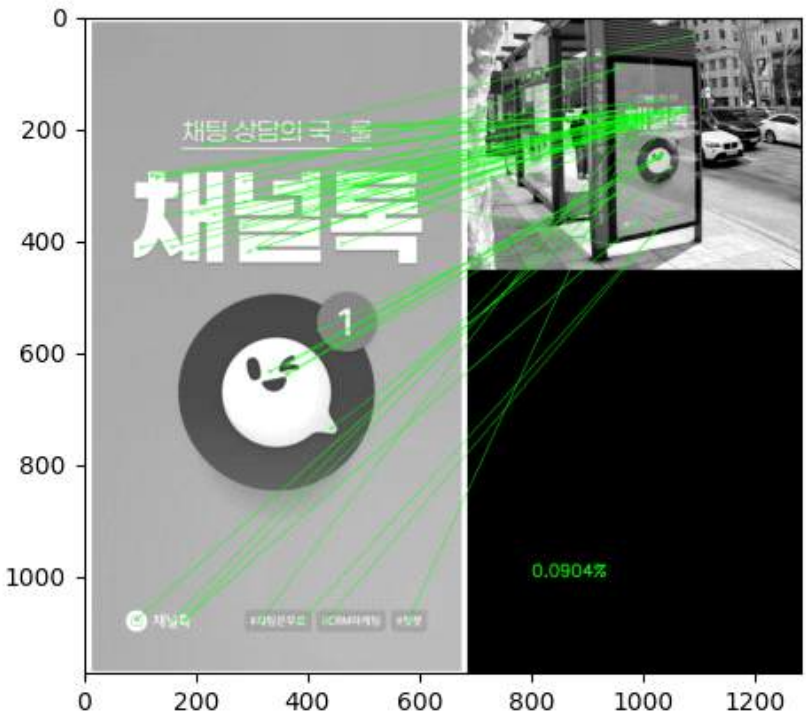
<그림 8, 9> 버스 대기소에 배치 된 채널톡 광고 사진

<그림 8>의 광고를 원본 그림인 <그림 7>과 매치 여부를 진행하였을 때에는 good/total의 비율이 약 22%가 나오게 되었다. 이 경우는 광고 상단에 존재한 "채팅 상담의 국-룰" 이라는 글자가 모두 일치하는 결과이므로 정말 완벽하게 일치하는 경우임을 알 수 있다. 이는 아래 그림 <그림 10>에서 확인 가능하다.



<그림 10> <그림 8>과 원본을 비교한 결과

반면 <그림 9>의 광고를 원본 그림과 비교 했을 때에는 약 9%의 good/total을 보여주었는데 이는 10%보다 낮은 수치이며 광고 상단의 "채팅 상담의 국-룰" 이라는 글자가 "근본있는 채팅상담" 이라는 글자로 바뀐 것을 확인할 수 있다. 이에 필자는 해당 프로그램에서 good/total의 수치를 10%의 Threshold로 설정 하였을 경우 가장 desirable한 결과를 얻어낼 수 있다는 결론을 지어 해당 수치를 선정하게 되었다.



<그림 11> <그림 9>와 원본을 비교한 결과

이러한 결과들을 종합해 good/total의 비율이 10%가 넘으면 Match Success로 간주하였다.

■ 결론 및 소감

서비스 기획, 디자인, 프런트 엔드, 백엔드, DevOps, DBA등 어플리케이션 출시의 모든 개발 과정을 직접 처음부터 끝까지 모두 혼자서 개발한 좋은 경험이었습니다. 기존에는 백엔드, 데이터 베이스 구축등의 한정적인 작업만을 특화한다는 핑계하에 다른 영역들을 등한시 하고 관심을 갖지 않았지만 이번 졸업 작품을 통해서 다른 영역들의 고충은 어떠한지, 전체 서비스의 관점에서 병목 현상이 생기기 좋은 부분은 어디인지 등 정말 소중한 경험을 할 수 있었습니다.

학부 기간동안 7개의 실습을 포함한 65학점의 전공을 들으며 이게 어디에 쓰일지, 언제 또 쓸지 몰랐던 과목들의 지식이 하나하나 도움이 되는 것을 느낄 수 있었던 진귀한 경험이었습니다. 실제로 모바일 앱 프로그래밍 실습 시간에 배운 Flutter를 배울 당시에는 이게 언제 또 쓰일지에 대한 의문을 크게 품었었는데 이런식으로 적용을 해보니 감회가 새로웠습니다. 배운 요구사항 명세, 시스템 아키텍쳐 설계, 디자인 명세서 작성 등 다양한 방식으로의 접근과 계획을 할 수 있어서 좋았습니다. 또한 인간컴퓨터상호작용 (Human Computer Interaction) 과목의 지식도 사용자 경험을 고려한 프런트엔드 제작에 큰 도움이 되었습니다. 심지어는 시스템 프로그래밍, 운영체제, 네트워크 개론 등의 컴퓨터 이론 과목들의 지식들도 HTTP 및 Rest API를 활용한 통신, Load Balancing, Multi Process 환경 등 다양한 용어들을 이해하고 이를 적용하는데에 큰 도움이 되었습니다. 추가적으로, 데이터베이스 개론과 데이터베이스 프로젝트를 통해 다져진 관계형 데이터베이스, 또 비관계형 데이터베이스에 대한 깊은 이해와 활용 경험이 정말 크게 다가와 데이터베이스를 구축하고 Database Admin으로 작업하는 데에 큰 힘이 되었습니다. 이처럼 다양한 전공과목들의 지식들은 한번 씩 Overview 할 수 있는 정말 진귀한 경험이었습니다.

두 번의 방학중 졸업을 위해, 또 커리어를 위해 진행한 두 번의 인턴십도 졸업 작품 프로젝트를 진행하는데에 큰 도움이 되었습니다. 2021년 여름 산학협력 프로젝트를 진행하였고, 2022년 여름 Meta Production Engineer Intern으로 근무를 하였습니다. 이 경험들을 활용해 프로젝트에 적용을 하니 더 완성도 있는 프로젝트를 만들어낼 수 있었던 것 같습니다. 다수의 Transaciton이 발생하는 상황을 가정한 개발이 가능했고, 그런 상황을 어떻게 해결해 나가야 하는지에 트러블슈팅이 원활히 가능 했던 것 같습니다. 또한 두 번의 제대로 된 프로젝트 및 인턴십 경험을 통해 완성도 있는 프로젝트란 무엇인가에 대한 평가 기준과 역치가 높아져 좀 더 완성도 있는 프로젝트를 이끌어갈 수 있는 능력이 생긴 것 같습니다. 단순히 수업 기말 프로젝트 수준의 과제와 프로그램이 아닌 하나의 정식 서비스로 배포가능한 결과물을 얻어내기 위해선 무엇을 해야하는지, 얼만큼의 노력이 필요한지, 인력을 구성할땐 어떤 식으로 팀을 꾸리고 관리를 해야하는지를 알게 되었습니다.

이처럼 학부과정에서 배워왔던 모든 지식과 경험을 총망라 하여 제가 해보았고, 해낼 수 있는 종합적인 능력을 다시 한번 상기시키고 다듬을 수 있었던 정말 "캡스톤"의 역할을 했던 종합 설계 프로젝트였던 것 같습니다. 해당 경험을 통해 앞으로 Industry에 나가 어떻게 일을 해야하고, 무엇을 놓치지 말아야 하는지를 알 수 있었습니다.

■ 참고문헌

[1] Relja Arandjelović and Andrew Zisserman, "Three things everyone should know to improve object retrieval", IEEE, 2012

[2] Hugo Brito; Anabela Gomes; Álvaro Santos; Jorge Bernardino, "JavaScript in mobile applications: React native vs ionic vs NativeScript vs native development", IEEE, June 2018

[3] Sasu Mäkinen, "Designing an open-source cloud-native MLOps pipeline", March 2021

[4] Simon Stender and Hampus Akesson, "Cross-platform Framework Comparison : Flutter & React Native), May 2020.

[5] Elastic Load Balancer Design Document, https://docs.aws.amazon.com/ko_kr/elasticloadbalancing/latest/application/introduction.html

[7] 이용환, 박제호, 김영섭, "SIFT와 SURF 알고리즘의 성능적 비교분석", 반도체디스플레이기술학회지, September 2013.

[8] 정우진, 오장훈, 유동원, "하이브리드 모바일 앱 프레임워크 설계 및 구현", 한국정보통신학회논문지, 2012

[9] 염경훈, 김기형, 김강석, "AWS 클라우드 기반 장애극복 부하분산 메커니즘 및 가용성 평가", 정보과학회, 2018