

동영상에서 객체 탐지를 이용한 영상 타임스탬프 시스템 개발



201724566 전승윤

201724513 유동운

201724404 강태환

지도교수 이 도 훈

목 차

1. 서론.....	1
1.1. 과제 배경 및 기존 문제점.....	1
1.2. 과제 목표.....	1
2. 과제 배경 지식 및 계획.....	2
2.1. 객체 검출.....	2
2.2. 객체 검출 모델 선정.....	2
2.2.1. YOLO.....	2
2.2.2. SSD.....	3
2.2.3. RetinaNet.....	3
2.2.4. 성능 비교.....	3
3. 과제 내용.....	5
3.1. 과제 계획.....	5
3.2. 학습 datasets 구축.....	6
3.3. YOLOv5 모델 학습.....	7
3.4. Logo Detection 구현.....	12
3.5. Web 프레임워크 구현.....	15
4. 연구 결과 분석 및 평가.....	20
5. 향후 연구 방향.....	27
6. 개발 일정 및 역할 분담.....	28
7. 참고 문헌.....	29

1. 서론

1.1. 과제 배경 및 기존 문제점

간접광고란 제품이나 기업을 직접적인 방법 대신 간접적으로 명시하는 광고를 총칭한다. 현재 한국의 방송법 시행령 제59조의3 간접광고에서는 방송에서의 간접광고의 가이드라인을 제공하고 있다. 또한 최근 방송에서의 간접광고의 빈도는 계속해서 높아지고 있다.

대부분의 간접 광고는 광고주와 방송사 사이에 대행사가 간접 광고를 중재하는 형태이다. 광고주는 대행사 사이에서는 의뢰를 통해 간접 광고를 기획하며, 대행사와 방송사는 직접적인 촬영을 진행한다. 그리고 최종 광고의 결과물에 대해 광고주는 검증을 거쳐 계약에 명시된 조건으로 촬영이 진행되었는지 검사할 필요가 있다.

그러나 대부분의 검증 작업에서는 인력을 사용해 직접 확인하는 경우가 대다수이다. 이를 효율적으로 해결할 수 있는 전용 툴 또한 따로 존재하지 않는다. 그러므로 만약 검사해야 하는 영상의 길이가 2시간이면 인력을 사용해 최소 2시간 동안은 영상을 점검해야 한다.

1.2. 과제 목표

따라서, 본 과제에서는 간접 광고가 있는 영상에서 기업의 로고가 검출되었는지 알 수 있는 AI 모델의 제작을 목표로 두고 있다. 목표 달성을 위해 로고 이미지를 모은 datasets을 구축한 후, Object detection에 좋은 성능을 보이는 모델을 골라 학습에 사용할 예정이다.

또한 서비스 이용자가 직접 영상과 이미지를 입력해 자사 로고의 노출 시간을 확인할 수 있는 웹 프레임워크를 제작한다. 앞에서 사용한 로고 학습 모델을 적용할 것이며 추가적인 기능을 넣어 사용자로부터 더욱 확실한 검증을 도울 생각이다. 이를 개발한다면 광고주와 대행사 사이에서의 빠른 검증이 가능해지므로 시간에 대한 효율성을 확보할 수 있다.

2. 과제 배경 지식 및 계획

2.1. 객체 검출

객체 검출이란 입력된 영상으로부터 모든 또는 지정된 몇 가지의 카테고리의 객체를 영상 내에서 찾은 후, 각 분류 또는 지역화를 진행하는 작업이다. 분류는 검출한 객체가 어떤 카테고리에 속하는지 알아내는 작업이다. 대부분의 분류 작업에서는 객체에 관해 카테고리별 점수를 책정, 가장 높은 점수를 받은 카테고리를 배정한다. 지역화는 그 객체가 어디에 위치하는지 찾는 작업이다. 이를 위해 객체를 나타내는 labeling box를 영상 위에 추가로 그려 검출 작업을 하는 사람들에게 시각적으로 표기를 한다.

객체 검출은 크게 두 가지 종류로 분류할 수 있다. 1-stage detection 또는 2-stage detection으로 분류하며, 이는 앞에서 말한 분류와 지역화를 동시에 하는지 또는 따로 하는지에 따라 달라진다. 1-stage detection은 분류와 지역화 작업을 동시에 진행한다. 그러므로 2-stage에 비해 이른 시간을 보이지만, 대신 그만큼 정확도에서 나쁜 성능을 보인다. 2-stage detection은 지역화 후 따로 분류를 진행한다. 1-stage에 비해 일을 두 번 진행하기 때문에 검출 시간에서 오래 걸리지만 그에 대한 대가로 정확한 성능을 보인다.

2.2. 객체 검출 모델 선정

객체 검출에서 좋은 성능을 보이기 위해, 본 과제에서는 객체 검출의 가장 최신의 State-of-the-art 모델을 사용하기로 했다. 그러므로 과제 착수를 하기 전 후보군을 만든 후 성능과 활용도 측면에 중점을 맞추어 본 과제에서 사용할 모델을 정하기로 했다.

아래의 내용은 후보로 선정된 모델들이며, 각각의 특징과 성능을 조사한 내용을 기술한다.

2.2.1. YOLO

YOLO는 You Only Looks One의 줄임말로 사진에서 하나의 신경망과 한 번의 계산으로 객체를 찾아낼 수 있는 객체 검출 모델이다. 사진에서 $n \times n$ 크기의 grid로 나눈 후, 각 grid에 대해 찾을 객체가 있을 확률을 각각 구하는 regression 방식이다. 현재 YOLO는 버전 7까지 개발되었고, 원래의 개발자 Joseph Redmon은 버전 3까지 담당하였다. 버전 4까지는 리눅스 환경의 Darknet에서 가능하며, 버전 5부터는 딥러닝 라이브러리인

Pytorch로 구현되었다.

2.2.2. SSD

Single Shot Multibox Detector는 기존의 객체 검출 모델의 문제점이 단 하나의 feature map으로 학습하기 때문이라고 생각해 이를 해결한 모델이다. 전의 모델들은 하나의 feature map에서 학습했기 때문에 객체의 크기에 따라 정확도가 달랐다. SSD는 이를 위해 해상도 별로 feature map을 만들어 학습을 진행했고, 이에 따라 좋은 정확도를 가질 수 있었다.

2.2.3. RetinaNet

RetinaNet은 기존의 1-stage detection 모델이 가지고 있던 객체와 배경 클래스의 불균형을 해결하기 위해 전의 loss 함수를 대신하여 Focal Loss라는 새로운 함수를 사용했다. Focal Loss는 기존의 Loss function에서 scale factor를 곱해 예측하기 쉬운 example에는 더욱 낮은 loss 값을, 반대로 예측하기 어려운 example에는 더욱 높은 Loss 값을 가진다. 이렇게 함으로써 Focal Loss는 전체적인 Loss를 낮추어 더 좋은 성능을 보이게 한다.

2.2.4. 성능 비교

각 객체 검출 모델의 성능을 비교하기 위해 모델의 각 논문에서 기술된 평가 목록의 내용을 토대로 비교했다. 논문의 평가에서 사용된 datasets은 객체 검출 성능 평가에 주로 사용되는 COCO datasets을 사용했다. 평가 지표로는 mAP와 AP50이 존재한다. mAP는 AP의 mean 값이며, AP는 기존의 precision-recall 그래프를 알기 쉽게 숫자로 만든 지표이고, AP에 붙는 숫자는 threshold값을 의미한다. AP와 mAP 값이 클수록 그 알고리즘은 객체 검출에서 우수한 성능을 보인다.

Table 9: Comparison of the speed and accuracy of different object detectors on the MS COCO dataset (test-dev 2017). (Real-time detectors with FPS 30 or higher are highlighted here. We compare the results with batch=1 without using tensorRT.)

Method	Backbone	Size	FPS	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	54 (P)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	43 (P)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	33 (P)	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%

그림 1 - COCO dataset에서의 YOLOv4의 평가 지표 [1]

Method	VOC2007 test		VOC2012 test		COCO test-dev2015		
	07+12	07+12+COCO	07++12	07++12+COCO	trainval35k		
	0.5	0.5	0.5	0.5	0.5:0.95	0.5	0.75
SSD300	74.3	79.6	72.4	77.5	23.2	41.2	23.4
SSD512	76.8	81.6	74.9	80.0	26.8	46.5	27.8
SSD300*	77.2	81.2	75.8	79.3	25.1	43.1	25.8
SSD512*	79.8	83.2	78.5	82.2	28.8	48.5	30.3

Table 6: **Results on multiple datasets when we add the image expansion data augmentation trick.** SSD300* and SSD512* are the models that are trained with the new data augmentation.

그림 2 - COCO dataset에서의 SSD 평가 지표 [2]

	backbone	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
<i>Two-stage methods</i>							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
<i>One-stage methods</i>							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	40.8	61.1	44.1	24.1	44.2	51.2

Table 2. **Object detection single-model** results (bounding box AP), vs. state-of-the-art on COCO test-dev. We show results for our RetinaNet-101-800 model, trained with scale jitter and for $1.5\times$ longer than the same model from Table 1e. Our model achieves top results, outperforming both one-stage and two-stage models. For a detailed breakdown of speed versus accuracy see Table 1e and Figure 2.

그림 3 - COCO dataset에서의 RetinaNet 평가지표 [3]

위의 표 이미지는 각각 YOLOv4, SSD, RetinaNet에 대한 결과이다. YOLOv5는 현재 논문이 없지만 대신 공식 사이트에서 YOLOv4와 평가 지표를 비교한 기록이 있다. [4] 버전 4와 비교했을 경우 큰 정확도의 차이는 없지만, 영상 분석에 있어 높은 속도를 내고 있음을 말하고 있다. 이를 유의해 mAP와 AP50의 점수를 비교한 결과 일반적으로 SSD < RetinaNet < YOLOv5의 순서로 성능을 가짐을 알 수 있었다. 이에 따라 프로젝트에서는 성능이 좋은 YOLOv5의 학습으로 진행하였다.

3. 과제 내용

3.1. 과제 계획

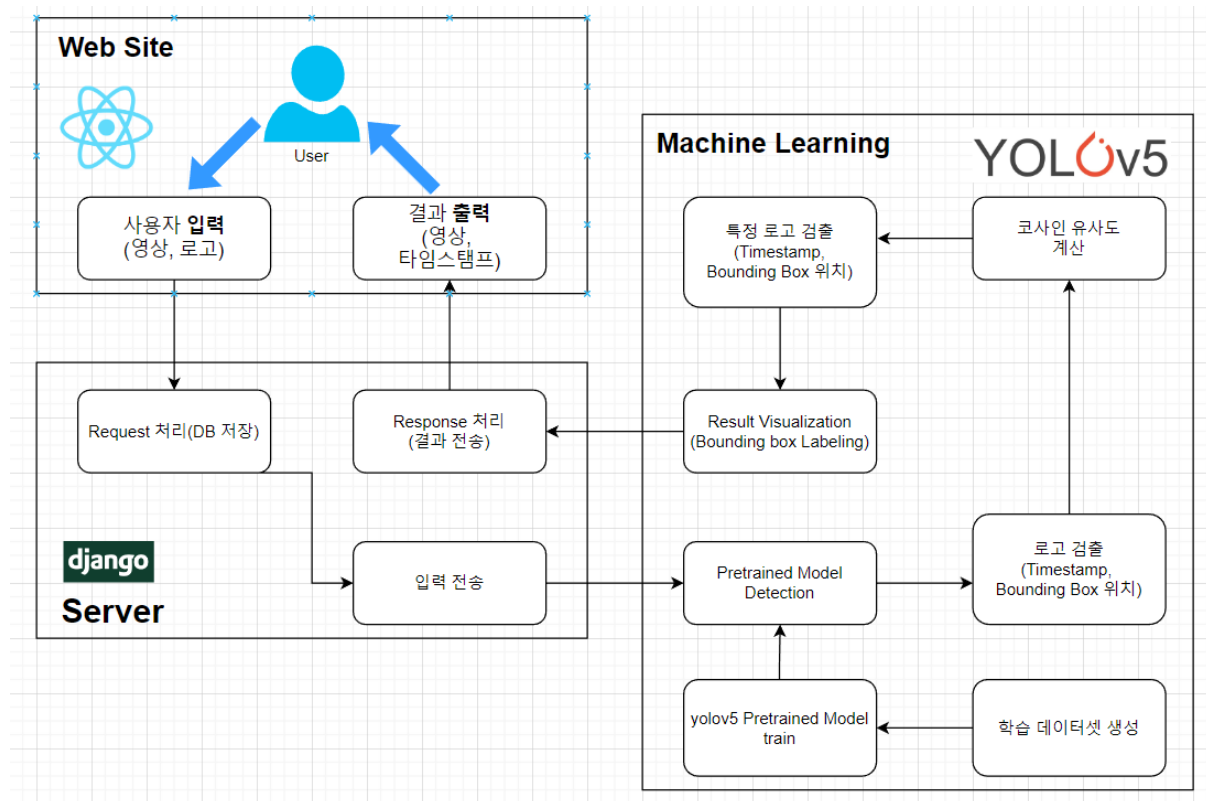


그림 4 – 본 과제의 순서도

본 과제의 목표는 사용자로부터 웹 페이지를 통해 영상과 검출할 로고의 이미지를 입력으로 받은 후, 이를 이용해 검출 작업에 대한 결과물을 보여주는 것이다. 이를 해결하기 위해 위와 같은 순서도로 과제의 순서도를 계획했다.

입력된 영상과 이미지를 Django 서버로 전송한다. 전송받은 Django 서버는 수신한 영상과 이미지로부터 객체 검출 작업을 시작한다. 이때 미리 제작된 pretrained 모델을 사용한다. pretrained 모델은 YOLOv5를 사용해 학습된 모델이며, 모든 로고에 대한 이미지를 datasets으로 삼아 학습한다. 즉 이 pretrained 모델을 사용한 검출의 목적은 영상에 존재하는 모든 로고의 검출이다.

모든 로고의 검출을 완료한 후 입력된 로고 이미지와 그 객체 사이의 유사도 검사를 시작한다. 유사도 검사에서는 코사인 유사도를 이용한다. 이미지별로 feature vector를 추출한 후 vector를 1차원 배열로 resize 한다. 각 이미지로부터 나온 1차원 배열 간의 유사도 값은 0과 1 사이의 값으로 도출된다. 1에 가까울수록 둘 이미지 간의 유사도가 높다

고 판단한다.

유사도 값이 threshold 값 이상일 경우, 찾으려는 브랜드의 로고로 결론을 내린 후 이들의 결과를 취합한다. 이 취합한 결과를 Django의 DB에 기록한 후 이를 웹 프레임워크로 다시 전송한다. 웹 프레임워크는 영상과 이미지 전송에 관한 결과를 받은 후 결과 창으로 리다이렉트 한다.

결과 창에서는 새로 라벨링 한 영상과 타임 스탬프의 형식으로 보여준다. 영상을 사용자에게 전달하여 어떤 객체가 찾으려는 로고인지 알 수 있게 하며, 또한 타임 스탬프를 버튼으로 구현해 각 버튼을 눌러 그 시간대로 영상을 이동시킬 수 있게 구현한다.

3.2. 학습 datasets 구축

본 과제의 datasets은 kaggle에 업로드 되어있는 공유 datasets이다 LogoDet-3K를 사용하였다. LogoDet-3K은 크게 9개의 큰 카테고리, 하위에 3,000개의 카테고리로 되어있는 약 15만 개의 이미지를 가진 datasets이다. LogoDet-3k는 이미지에서 로고 위치와 어떤 브랜드인지 나타내는 메타 데이터가 담긴 xml 파일 또한 같이 제공한다. 이는 YOLOv5 학습에 필요한 정보이기 때문에 학습 datasets을 만드는 데 필요한 시간을 줄여준다.

Root Category	Sub-Category	Images	Objects
Food	932	53,350	64,276
Clothes	604	31,266	37,601
Necessities	432	24,822	30,643
Others	371	15,513	20,016
Electronic	224	9,675	12,139
Transportation	213	10,445	12,791
Leisure	111	5,685	6,573
Sports	66	3,945	5,041
Medical	47	3,945	5,185
Total	3,000	158,652	194,261

그림 5 - LogoDet-3k 내부 이미지 구성

YOLOv5에서 학습할 수 있게 LogoDet-3K의 전처리가 필요하다. 먼저 LogoDet-3k에 존재하는 모든 데이터를 학습에 사용하기에는 너무 많은 양이다. 학습할 수 있는 시간이 한정적이기 때문에 모두 학습에 사용하는 것이 아닌 큰 카테고리 9개 중 일부를 사용하거나 전체 로고 데이터 중 일부를 무작위로 골라 사용하였다.

LogoDet-3K에서는 각 사진과 라벨 정보(객체의 위치)가 같은 이름으로 jpg와 xml 파일로 저장되어 있다. YOLOv5의 라벨 데이터는 지정된 형식의 txt 파일로 받아야 하므로 xml 파일을 읽어 parsing 한 후 지정된 형식으로 변경해주어야 한다.

또한 LogoDet-3K의 라벨 데이터는 픽셀로 표시하고 있다. 그러나 YOLOv5에서 필요한 라벨 데이터는 절대적인 픽셀값이 아닌 사진 전체의 길이를 1로 보고 표현하는 상대적인 값이므로 이에 알맞게 바꿔줘야 한다. Python의 ElementTree를 이용해 xml 파일을 읽은 후 각각 계산해주어 YOLOv5가 사용할 수 있게 바꿔준다.

3.3. YOLOv5 모델 학습

YOLOv5 이용한 객체 검출 모델을 만들기 위해서 YOLOv5 모델 학습을 진행하였다. 모델 학습은 PyTorch를 사용하며 초기에는 Jupyter Notebook에 코드를 작성하여 Google Colab에서 수행하였고 이후 모델 학습 서버를 지원받아서 서버에서 학습을 수행하였다.

```
!python train.py --img 416 --batch 64 --epochs 20 --data Det-normalizS_M_L_E_T --weights yolov5s.pt
```

학습을 위해 여러 Datasets 집단을 사용하였고, 집단 내에서 학습 성능이 뛰어나고 영상 sample에서 로고 탐지를 잘하는 Datasets을 비교하여 사용했다. 각 Datasets의 내용 및 구성은 아래의 표로 정리한다.

Datasets	Data 수	Train-Validation 비율	Category
A	8000	4:1	Sport, Medical
B	10000		전체
C	12000		
D	14000		
S_M_E_L_T	33703	3:1:1(Test)	Sport, Medical, Electronic, Leisure, Transportation
normalizeS_M_E_L_T	15000		Sport
normalize	27000		전체

표 1 - 사용한 Datasets

또한 각 학습 별 기록 및 비교를 위해 YOLOv5에서 제공하는 학습 결과 시각화 리더보드 툴인 WanDB(Weight & Biases)를 이용했다. 아래의 사진은 WanDB에서 학습의 결과를 비교하는 창이다.

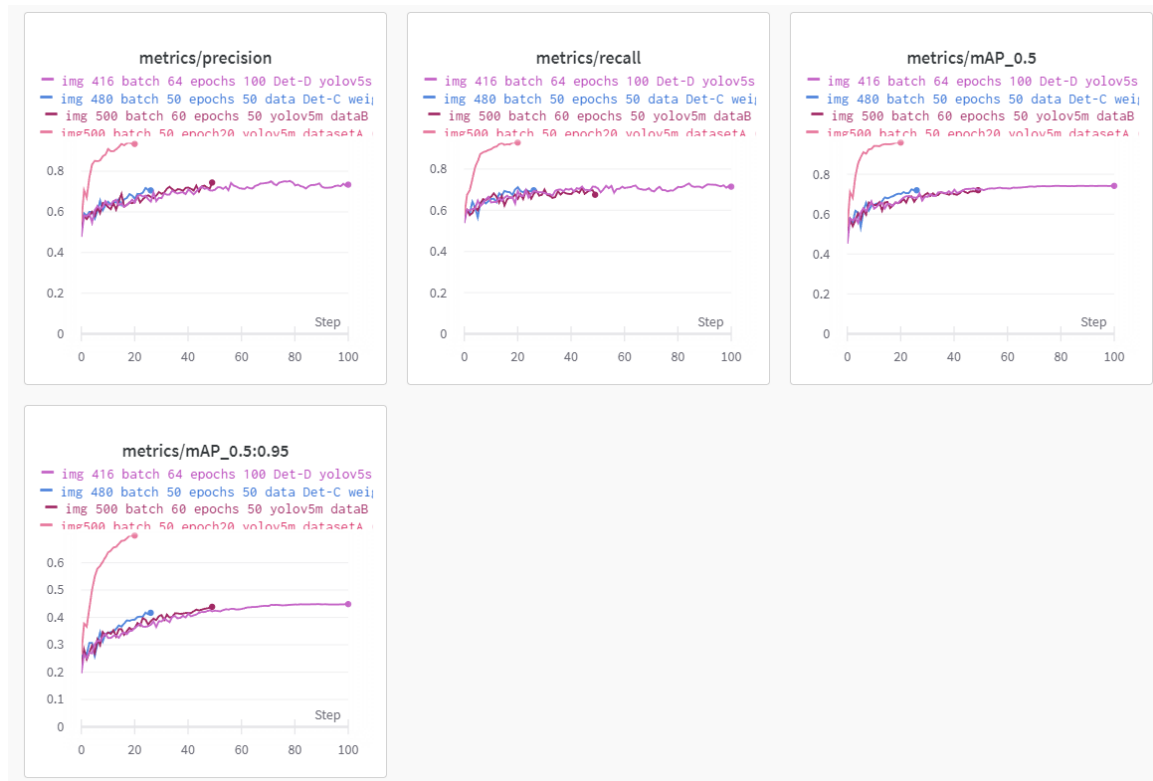


그림 6 - WanDB

마지막으로 학습한 결과가 실제 동영상에서 검출이 되는 것을 확인하기 위해 스포츠 하이라이트 영상을 사용했다. 총 3가지의 영상을 사용했으며 각 영상의 내용은 축구와 야구 중계 화면이다. 영상 내에서는 멀쩡히 로고가 노출되기보다는 로고가 빠르게 지나가거나 흐릿하게 보이거나, 유니폼에 있는 경우 구겨진 상태도 보인다. 실시간으로 방송에 노출되었던 만큼 효과적인 로고 검출 시험 영상이 될 것으로 기대되어 사용하였다.



그림 7 - 사용된 스포츠 영상

처음 학습은 카테고리의 조합과 각 set 별로 최적의 인자 값을 찾기 위해 Datasets A, B, C, D에서만 이루어졌다. 아래는 그 학습의 결과 수치와 검출 결과이다.

Datasets	Data 수	mAP	Precision
A	8000	0.942	0.9221
B	10000	0.66	0.6623
C	12000	0.7013	0.6645
D	14000	0.7139	0.7003

표 2 - Datasets A~D의 결과 지표



그림 8 - Detection Result of Datasets A



그림 9 - Detection Result of Datasets D

지속된 학습의 결과로, 앞으로의 학습에서 사용할 인자는 imgSize 416, batch 64, epochs 20으로 사용하기로 했다. 대체로 이 인자들이 좋은 학습 결과를 보였기 때문이다.

분석한 결과, Datasets A가 가장 좋은 성능 지표를 보이는 것을 확인했고, Datasets B~D는 Datasets A에 비해 모자란 성능 지표를 보이는 것을 알 수 있었으나 영상 sample 분석 결과 Datasets D가 가장 많은 로고를 탐지하는 것을 확인할 수 있었다. 이러한 결과는 Datasets A와 B~D의 data 구성의 차이점에서 온 것으로 추측된다. Datasets A는 LogoDet-3K와 같은 서브 카테고리 내에서 Train-Validation set을 나누는 방식을 취했지만, 이에 반해 Datasets B~D는 LogoDet-3K의 전체 이미지에서 랜덤하게 뽑은 이미지에서 Train-Validation을 나누었기 때문에 A에 비해 Train-Validation set 사이의 유사도가 다소 부족하므로 지표에서 약점을 가질 수밖에 없다.

Datasets D는 로고를 검출하는 성능 면에서 유의미한 결과를 보이지만, 무작위적으로 이미지가 골라 Datasets이 구성됐기 때문에 의도한 모델 학습을 끌어낼 수 없다는 단점이 있다. 이를 해결하기 위해 그다음 학습에서는 LogoDet-3k의 카테고리를 제한함으로써 특정한 분야의 로고에 대한 정확도를 높이하고자 했다.

Datasets S_M_E_L_T는 위에서 말한 의도성을 확보하고자 추가되었다. LogoDet-3K Datasets에서 5개의 카테고리(Sports, Medical, Electronic, Lesuire, Transportation)에서 카테고리별로 모든 이미지를 뽑아서 train, val, test를 3:1:1 비율로 나누어 구성하였다. 이 Datasets의 학습 결과 수치와 검출 결과는 아래와 같다.

Datasets	Data 수	mAP	Precision
S_M_E_L_T	33703	0.9245	0.8848
normalizeS_M_E_L_T	15000	0.1679	0.1859
normalize	27000	0.7498	0.6953

표 3 – Datasets SMELT, normalizeSMELT, normalize의 평가 지표

DataSets S_M_E_L_T에서 학습 결과 평가 지표는 Datasets A만큼 높게 확보됐지만, 영상 sample 확인 결과 Datasets D에서 탐지할 수 있던 로고 객체들을 탐지할 수 없었다. 이는 각 이미지 카테고리별로 이미지 개수에 차이가 있어 특정 분야의 로고에 대해서 편향되어 학습되었기 때문이다. 이를 해결하기 위해 각 카테고리별로 3000개씩 이미지를 뽑아서 정규화한 Datasets normalizeS_M_L_E_T를 만들어서 학습을 진행하였다.



그림 10 - Detection Result of Datasets normalizeS_M_E_L_T

Datasets normalizeS_M_L_E_T에서 학습 결과, 평가 지표는 모든 Dataset 보다 떨어졌지만, 영상 sample 확인 결과 로고 객체를 잘 확인하는 것을 볼 수 있었다. 추가로 더 좋은 결과를 보기 위해 datasets을 늘려서 학습을 진행하였다. 전체 카테고리에서 각각 3,000장의 이미지를 뽑아서 datasets normalize를 만들어서 학습을 진행하였고, 학습 결과 평가 지표가 이전 Datasets에 비해 떨어지지 않고 영상 sample 결과도 나쁘지 않은 것을 확인하였다. 그러므로 본 과제에서는 datasets normalize를 이용한 학습 모델을 최종 pretrained 모델로 사용하기로 하였다.

단 최종 선정된 학습 모델에서도 경기 중인 선수를 로고로 탐지하거나 일부 로고는 탐지하지 못하는 등 로고 객체 탐지 성능이 좋지 못하다. 이는 YOLOv5 모델이 단색 바탕의 물체를 전부 로고처럼 판단하고 탐지하기 때문이다.

또한 상대적으로 글자로 이루어진 로고의 탐지에는 좋은 성능을 보이지만 그림이나 문양으로 이루어진 로고의 검출에서는 나쁜 성능을 보인다. 글자 로고는 대부분이 영어로 되어 있으며, 이에 따라 로고별 유사성을 찾기 쉽지만, 이에 반해 그림 로고일 경우, 서로 간의 유사성을 찾는 데에 애로 사항이 있어 성능이 떨어진 것으로 판단했다. 또한 LogoDet-3k의 로고 이미지들이 글자 로고로 치우쳐져 있는 편향성이 있기 때문이기도 하다.

이를 해결하기 위해 image에서 일부분을 가리거나, 로고 객체가 없는 이미지를 추가하여 학습을 진행했지만, 오히려 탐지 성능이 나빠지기만 할 뿐 발전이 없어 이 Datasets은 사용하지 않았다. 대신 최대한 많은 로고를 탐지한 후, 이들을 추후 유사도 비교에서 가려낼 수 있도록 학습을 진행하였다.

3.4. Logo Detection 구현

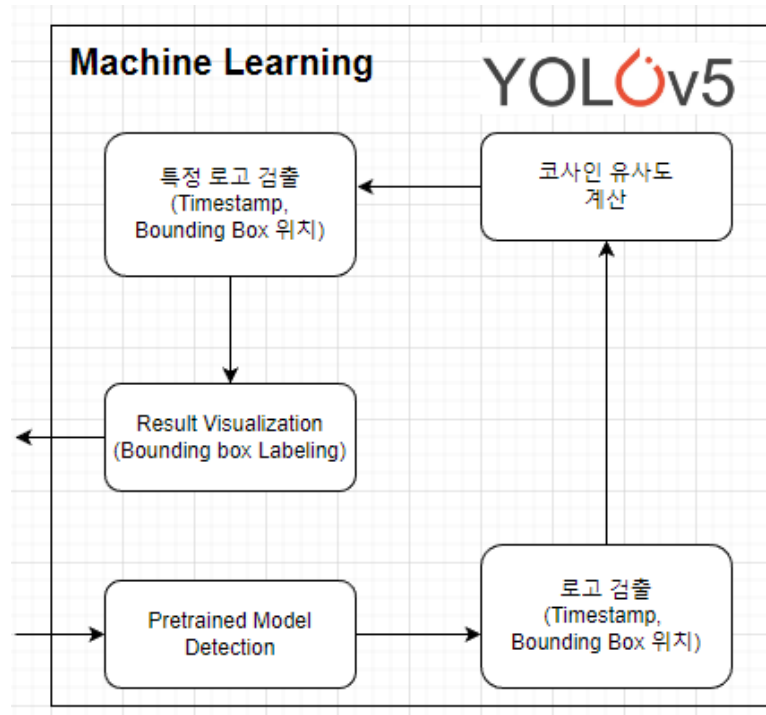


그림 11 – Logo Detection 부분의 순서도

위의 순서도는 Django 프로젝트에서 처리되는 작업을 순서도로 표현한 것이다. 프론트로 부터 전송된 Rest api를 받은 후 입력된 로고 이미지와 영상을 Detection 작업에 사용할 수 있게 threshold 값과 함께 보낸다. 그 후 Detection 작업에서는 먼저 학습된 가중치 파일을 이용해 모든 로고에 관해 검출한다. 이후 검출된 로고 결과를 입력된 이미지와 비교, 입력된 threshold 값보다 클 때 찾으려는 로고라고 결론을 내린다.

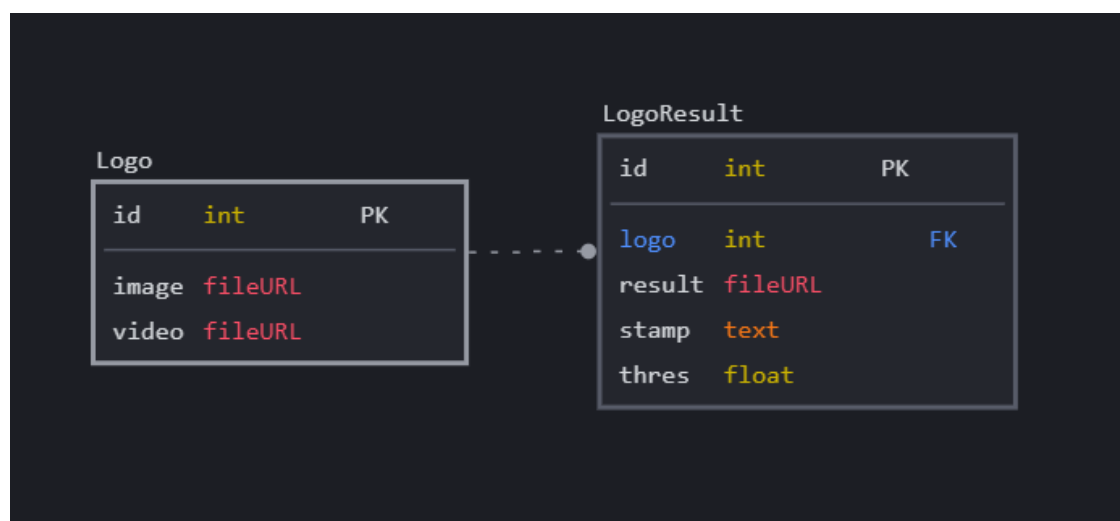


그림 12 – DB 구성

프로젝트에서 사용하는 DB의 구조는 간단하게 구성했다. Logo Table에서는 각 입력 별로 들어오는 로고 이미지와 비디오를 저장하며, 각 Detection의 결과를 나타내는 LogoResult Table과 1대1 관계를 맺고 있다. LogoResult Table은 labeling한 비디오, 결과, 그리고 threshold value를 담고 있다.

Django에서는 이미지나 비디오와 같은 binary 파일을 그대로 저장하는 것이 아니라, 실제 파일을 저장하는 디렉터리를 생성 후, 그 저장된 파일들의 url을 db에 저장한다. LogoResult Table의 result video 또한 디렉터리에 저장한 후에 그 url을 db에 저장한다. 각 파일은 Django root 디렉터리의 files 폴더 내에 저장된다.

학습은 특정한 회사의 로고가 아닌 로고 이미지에 대한 공통점을 찾고자 여러 회사의 로고 이미지 모아 학습되었다. YOLO를 통해 학습했을 때 학습한 클래스가 "Logo" 하나였기 때문에 YOLO를 사용한 Detection 작업은 객체 분류의 작업이 일어나지 않는다.

Algorithm 1: Video-Logo Detection

```

Input: image, video, thres, weight
Output: new_video, prediction
weight = pre-trained weight files;
datasets = LoadImages(video);
/* instance definition */
Model() = DetectMultiBackend(weight);
Classifier() = SecondClassifier(image);
/* loop per frame of video */
foreach frame f, index i of the video video do
    /* prediction jobs */
    prediction = Model(f);
    prediction = NonMaxSuppression(prediction);
    prediction = Classifier(prediction);
    /* labeling jobs */
    new_f = Annotator(f, prediction);
    new_video[i] = VideoWriter(new_f);
end
return new_video, prediction;

```

2-stage Detection을 구현한 알고리즘의 pseudo Code이다. 여기서 사용한 함수 중 SecondClassifier를 제외하고는 YOLOv5에서 제공하는 함수를 사용했다. 먼저 LoadImages를 통해 입력된 Video를 프레임별로 나누어 각 프레임의 배열을 가지는 인스턴스로 변환시켜준다. 그 후 loop를 통해 각 프레임별로 Detection 작업을 시행한다.

각 loop에서는 학습한 weight 파일을 이용해 선언한 DetectMultiBackend 모델을 통해 프레임에서 Detection을 수행, prediction 결과를 받는다. 이때 프레임에서 하나의 객체를

여러 번 검출할 수 있으므로 NonMaxSuppression을 사용해 중복된 검출을 제거한다. 마지막으로 Classifier를 통해 각 검출된 결과와 입력된 로고와 유사한지 아닌지를 판별해 걸러낸다.

loop의 마지막에서는 detection의 최종 결과를 사용해 frame에 labeling을 하여 새로운 frame인 new_f를 만든다. 이 새로 만들어진 frame은 결과를 표현하는 result_video에 사용된다. openCV의 VideoWriter 을 사용해 새로 만들어진 frame 들을 영상에 저장한다.

Algorithm 2: 2nd Classifier

Input: detection, image, threshold
Output: detection
weight = resnet18's pretrained weight, IMAGENET1K_V1;
model = resnet18(weight);
image_vec = GetFeatureVector(model, image);
 /* loop per prediction of detection */
foreach image pred, index i of the detection detection **do**
 pred_vec = GetFeatureVector(model, pred);
 similarity = CosineSimilarity(image_vec, pred_vec);
 if similarity is small then threshold **then**
 delete pred;
 end
end
return detection;

SecondClassifier는 다음과 같은 방법으로 작동된다. 입력된 로고 이미지와 prediction한 결과 간의 코사인 유사도를 계산한다. 이때 유사도의 값은 0과 1 사이의 float 값으로 결과가 나온다. 만약 계산된 유사도의 값이 사용자로부터 받은 threshold 값에 비해 작다면, 그 prediction 결과를 삭제한다.

Algorithm 3: Extract Feature Vector

Input: image, model
Output: feature
feature = array of zeros, size of avgpool of resnet 18, 1X512X1X1;
add layer which copy result to feature after model forward;
model(image)
return feature;

이미지에서 Feature Vector을 추출하는 방법은 다음과 같다. Resnet18의 avgpool layer의 결과를 Feature Vector로 판단, 이를 복사해 코사인 유사도를 계산한다. 이는 Pytorch의 register_forward_hook을 사용해 구현할 수 있다. 이를 사용하면 특정한 layer의 forward 작업 후 hook를 걸어 그다음에 일어날 작업을 지정할 수 있다. 이를 이용해 feature에

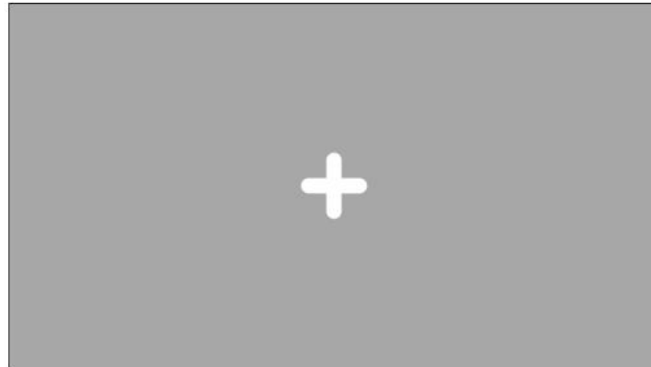
값을 복사하는 것이 가능하다. 복사 후 이를 결과로 출력한다.

3.5. Web 프레임워크 구현

이 프로젝트의 목적은 사용자로부터 로고 이미지와 영상을 입력받아 지정된 로고 이미지를 영상 내에 찾은 후 이를 보여주는 것이다. 그러므로 목적 달성을 위해서는 사용자와의 상호작용을 할 수 있는 웹 프레임워크를 작성해야 한다.

사용자와 직접적인 상호작용을 담당하는 프론트 프로젝트는 React.js 로 작성되었으며, 프론트로부터 전송되는 Rest api를 받아 처리하는 백 서버 프로젝트는 Django로 구현되었다. Rest api 뿐만 아니라, Detection 작업과 영상 작업이 필요하므로 각각 YOLOv5 와 OpenCV를 사용했으며, 이 둘은 Python으로 구현되어 있으므로 Django를 고르게 되었다.

Upload your Video



Accepted files

Rejected files

Upload Logo Image you want to detect



Input Threshold value

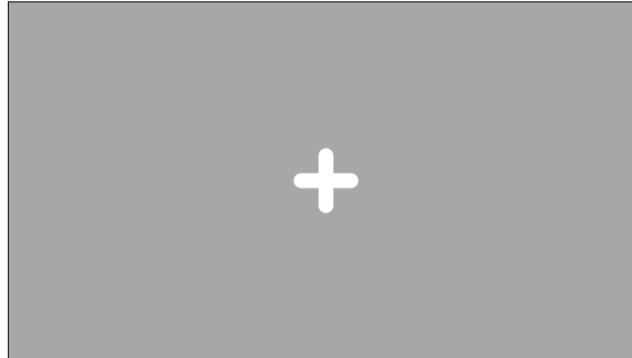
Submit

그림 13 – 기본 웹 사이트

구현된 웹 페이지의 주 페이지다. 이 화면에서 사용자는 영상과 로고 이미지를 입력할 수 있다. 각 입력 버튼을 눌러 파일 탐색기를 통해 넣거나 지정된 파일을 드래그 앤드 드롭 하는 것으로 입력할 수 있다.

LogoFinder

Upload your Video



Accepted files

- sports_test1.mp4 - 74779995 bytes

Rejected files

Upload Logo Image you want to detect



Input Threshold value

Submit

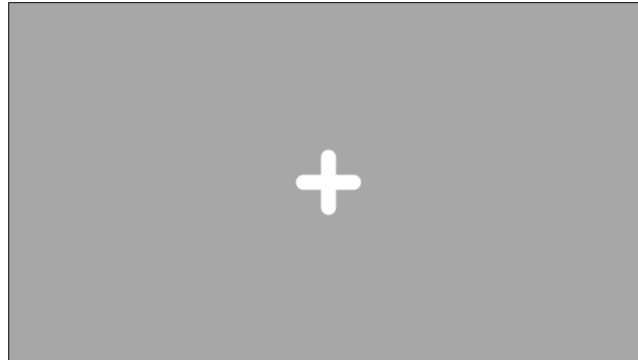
그림 14 – 입력된 영상, 이미지들

위의 화면은 영상과 이미지가 정상적으로 입력되었을 때의 화면이다. 어떤 영상을 넣었는지 파일명을 통해 알 수 있으며, 어떤 로고 이미지가 입력되었는지 알 수 있게 입력 버튼의 background에 반투명하게 출력했다.

마지막으로 사용자로부터 detection에 사용할 threshold 값을 입력받아야 한다. 값의 범위는 0과 1사이여야 하며, 잘못된 값을 입력할 때 실행을 중단, 다시 입력받게 한다. 만약 빈칸으로 제출할 때 threshold의 기본값인 0.95로 계산한다.

LogoFinder

Upload your Video



Accepted files

- sports_test1.mp4 - 74779995 bytes

Rejected files

Upload Logo Image you want to detect



0.98

Submit



그림 15 – response 대기

모든 값을 입력한 후 “Submit” 버튼을 눌러 api 요청을 할 수 있다. 그러나 Detection 작업이 빠른 시간 내에 요청 받기에는 상대적으로 긴 시간을 요구하기 때문에 비동기를 이용해 구현하였다. api 요청 대기 시간은 1분 영상 기준 40초에서 1분 사이로 소요된다. 화면이 멈춰 사용자가 불편함을 느끼게 하는 것을 방지하기 위해 작업의 결과가 올 때까지 Spin animation을 넣어 아직 작업 중임을 보여준다. 프론트에서 성공 response를 받는 순간 결과 창으로 넘어가 Detection 결과를 출력한다.

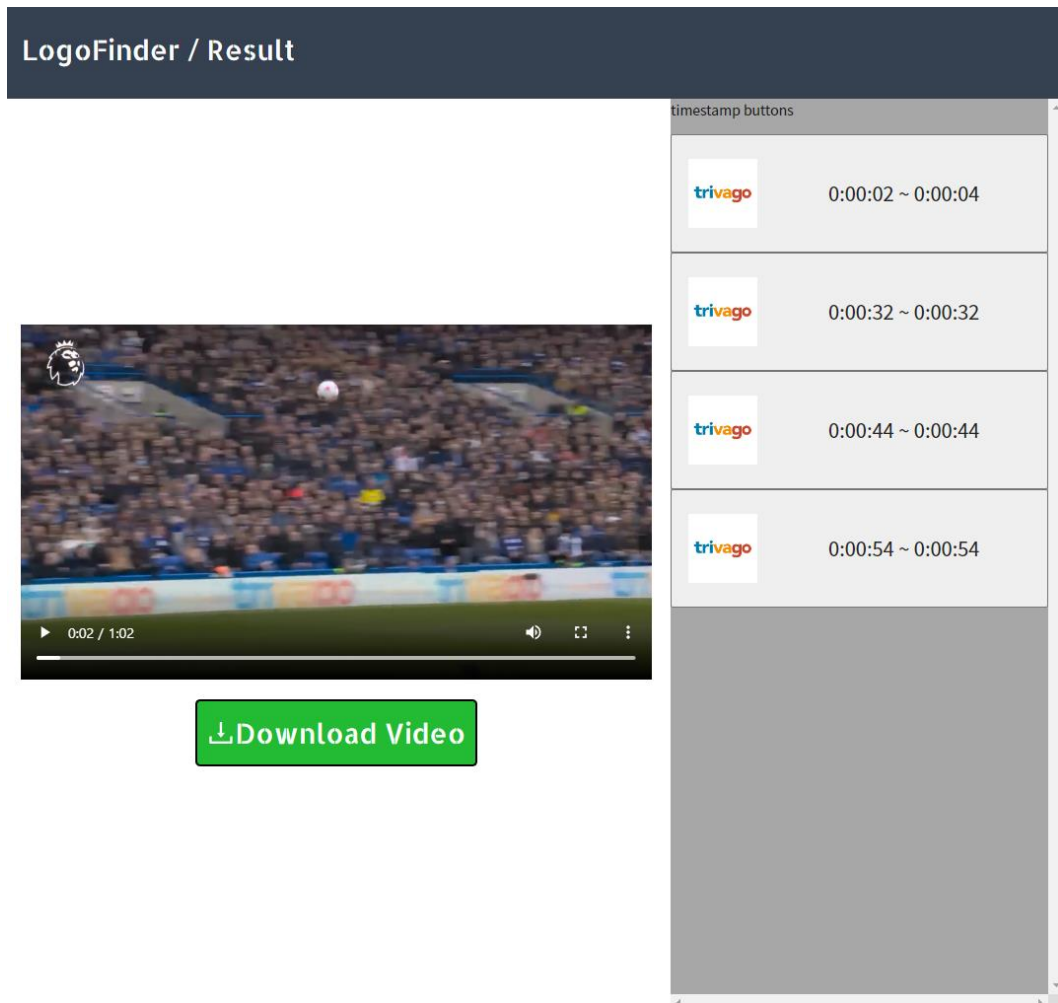


그림 16 – result page

위의 사진은 결과를 출력하는 Result 페이지이다. 왼쪽은 입력한 영상과 Detection 결과를 Bounding Box를 이용해 Labeling한 결과 영상을 다운로드를 받을 수 있는 다운로드 버튼이 있으며, 오른쪽 회색 영역에서는 언제 로고가 검출되었는지 알 수 있는 Timestamp 버튼들이 있다. 각 버튼을 누르면 시작 기간으로 영상을 넘길 수 있는 상호작용을 추가하였다.

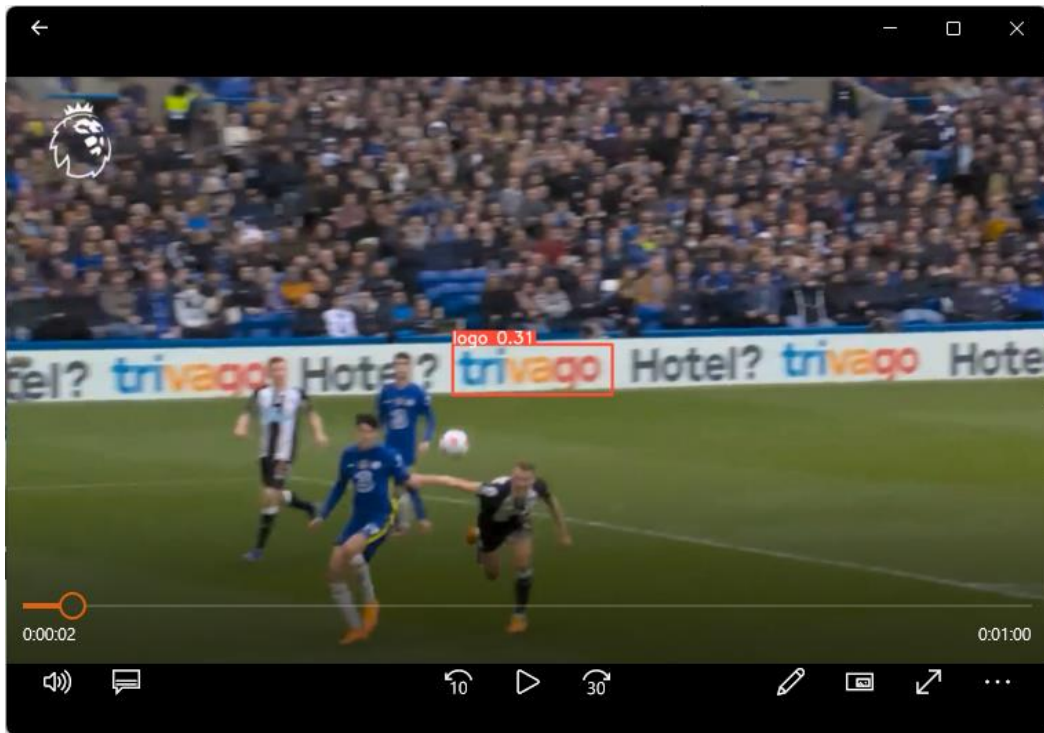


그림 17 – Labeling Video

위의 화면과 같이 Bounding Box로 Labeling된 결과 영상을 다운로드 받아 확인할 수 있다.

4. 연구 결과 분석 및 평가

LogoDet-3K에서 약 33,000개의 로고 객체 이미지를 사용하여 모델학습을 완료하였다. 학습된 모델을 이용하여 3개의 스포츠 경기 하이라이트 영상에서 로고 검출 검증을 진행하였다. 검증에서 사용한 threshold value는 0.85이다. 검출한 로고는 각 영상에서 존재하는 로고이며, 실험한 로고는 아래와 같다.

		
1번 영상 - trivago	2번 영상 - Budweiser	3번 영상 - TD Bank

표 4 – 평가에 사용된 로고

영상 1에서는 trivago 영어 문구 로고를 모두 검출하였으나 약 3분의 2에 대해 잘못된 판별을 했다. 영상에서 trivago 로고는 영상 중 2~5초까지 약 4초가 존재하였는데, 이때 검출 구간이 0~10초, 12초~19초, 21~23초, 25초~52초, 54초-1분으로 총 52초가 'trivago' 로고가 노출되었다고 판별되었다.


영상 2에서는 Budweiser 로고를 모두 검출하였으나, 영상의 약 2분의 1에 대해 로고가 있다고 잘못 판별하였다. 영상에서 Budweiser 로고는 영상 중 8-12초로 약 5초가 존재했다. 임계값을 0.85일 때, 검출구간으로 0~22초로 총 23초가 검출되었다. 이 중에서 5초는 로고를 정확히 검출했지만, 18초는 잘못 판별하였다.

영상 3에서는 TD Bank 로고를 75%를 검출했으나, 영상의 약 2분의 1을 로고가 있다고 잘못 판별하였다. 1분 3초 영상을 탐지하는 데 약 45초가 소요되었다. 영상에서 TD Bank 로고는 영상 중 38~46초, 50~52초로 약 12초가 존재했다. 임계값 0.85로 설정하였는데, 검출구간으로 0~3초, 5초, 7~28초, 30초, 32~46초로 총 43초가 검출되었다. 이 중에서 9초는 로고를 정확히 검출되었고, 34초는 잘못 판별하였다. 이때 검출되지 못한 3초의 경우 1 stage 로고 검출 과정에서 검출이 되지 않았다.

	1번 영상 - trivago	2번 영상 - Budweiser	3번 영상 - TD Bank
영상 길이	1분 2초	30초	1분 3초
영상 검출 시간	1분 15초	20초	45초
로고 노출 시간	4초	5초	12초
임계치 0.85시 검출된 시간	52초	23초	43초
검출 성공 시간	4초	5초	9초
잘못 판별된 시간	48초	18초	34초

표 5 - threshold 0.85 일 때의 결과 지표

LogoFinder / Result



0:08 / 1:02

Download Video

timestamp buttons







	0:00:00 ~ 0:00:10
	0:00:12 ~ 0:00:19
	0:00:21 ~ 0:00:23
	0:00:25 ~ 0:00:52
	0:00:54 ~ 0:01:00

그림 18 – threshold 0.85일 때의 영상 1의 결과

LogoFinder / Result



0:18 / 0:30

Download Video

timestamp buttons


	0:00:00 ~ 0:00:22
--	-------------------

그림 19 - threshold 0.85일 때의 영상 2의 결과

LogoFinder / Result

timestamp buttons

TD	0:00:00 ~ 0:00:03
TD	0:00:05 ~ 0:00:05
TD	0:00:07 ~ 0:00:28
TD	0:00:30 ~ 0:00:30
TD	0:00:32 ~ 0:00:46

Download Video

그림 20 - threshold 0.85일 때의 영상 3의 결과

그러나 이런 결과가 나온 이유는 threshold value 값을 낮게 주었기 때문이다. 그러므로 다음 검사에서는 객체를 검출할 수 있는 최소한의 객체의 수만 보기 위해 threshold value를 최대로 입력하였다.

먼저 영상 1에서는 threshold value 값을 0.99를 주어 검사하였다. 기존의 threshold 값이었던 0.85에 비해 높은 값이다. 0.85와 같게 영상 내에 존재하는 모든 trivago logo의 검출에 성공했으며 검출 기간 7초 중 성공 기간이 3초나 되는 비율을 보인다.

영상 2에서는 threshold value 값을 0.9를 주어 검사하였다. 1번 영상과 비슷한 검출 기간을 보여주었지만, 검출 성공 기간도 마찬가지로 줄어버린 것을 볼 수 있었다.

영상 3에서는 영상 1과 2에 비해 threshold value를 많이 올리지 못했다. 잘못 판별한 기간은 35초에서 29초까지 줄일 수 있었지만, threshold value 값을 더 높일 때 성공 기간까지 줄어버렸기 때문에 올릴 수는 없었다.

	1번 영상 - trivago	2번 영상 - Budweiser	3번 영상 - TD Bank
입력된 threshold 값	0.99	0.90	0.88
로고 노출 시간	4초	5초	12초
검출 기간	7초	7초	37초
검출 성공 시간	3초	1초	8초
잘못 판별된 시간	4초	6초	29초

표 6 – threshold 값이 최대일 경우 결과 지표

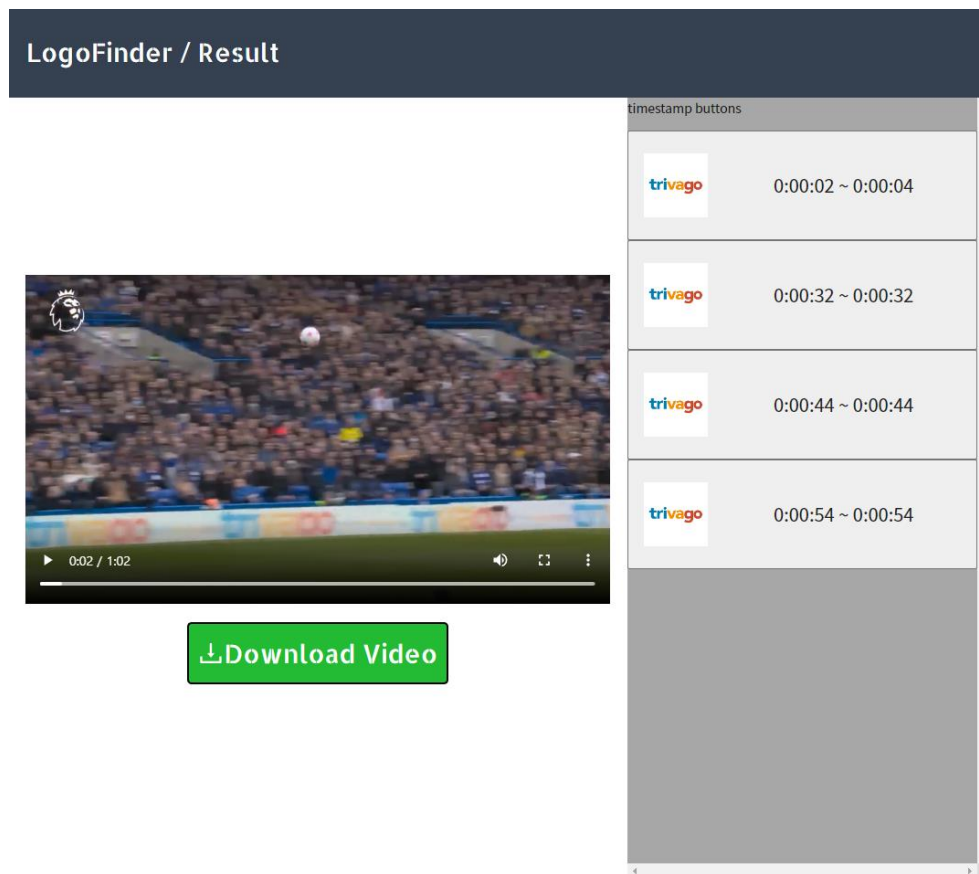


그림 21 – threshold 0.99일 때의 영상 1의 결과

LogoFinder / Result



[Download Video](#)

timestamp buttons



0:00:00 ~ 0:00:01



0:00:10 ~ 0:00:10



0:00:15 ~ 0:00:15



0:00:20 ~ 0:00:22

그림 22 - threshold 0.90일 때의 영상 2의 결과

LogoFinder / Result



[Download Video](#)



0:00:07 ~ 0:00:13



0:00:15 ~ 0:00:23



0:00:25 ~ 0:00:25



0:00:28 ~ 0:00:28



0:00:30 ~ 0:00:30



0:00:32 ~ 0:00:37



0:00:39 ~ 0:00:46

그림 23 - threshold 0.88일 때의 영상 3의 결과

결과에서 보듯이 입력된 영상과 이미지에 따라 입력되어야 하는 threshold value 값이 각각 다른 것을 볼 수 있다. 영상 별 정답 검출 기간의 유사도가 큰 값을 보이는 것을 보면 코사인 유사도의 계산 값이 이미지 유사성을 충분히 표현할 수 있다고 본다. 그러나 이 유사도 값의 분포도가 각 시도별로 다른 것이 문제이다. 이를 표현하기 위해 유사도 계산의 결과를 도식화 하였다.

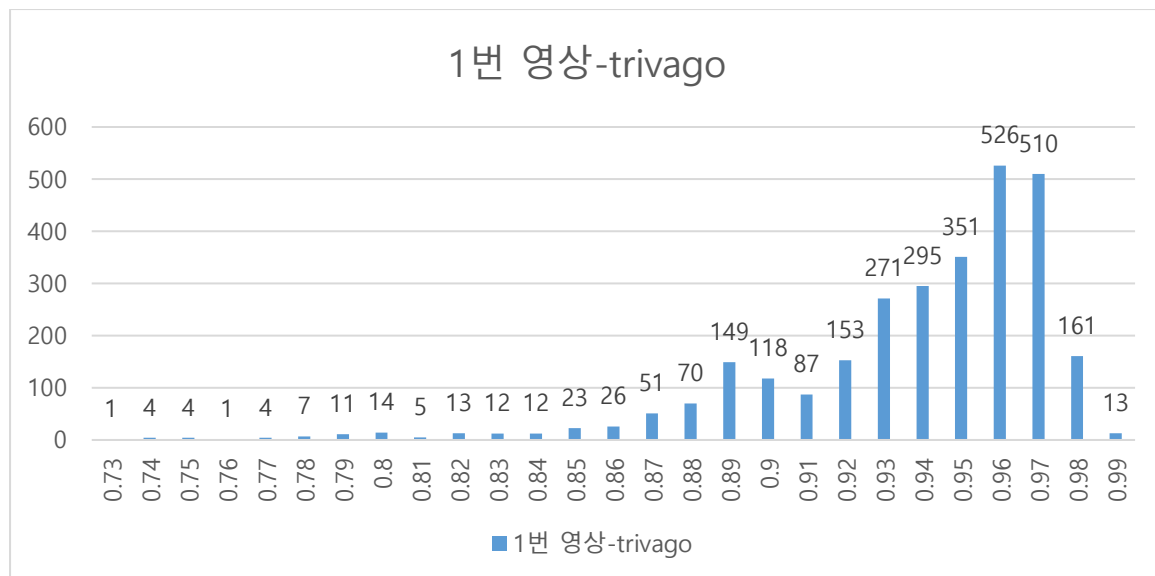


그림 24 - 1번 영상의 객체 유사도 분포

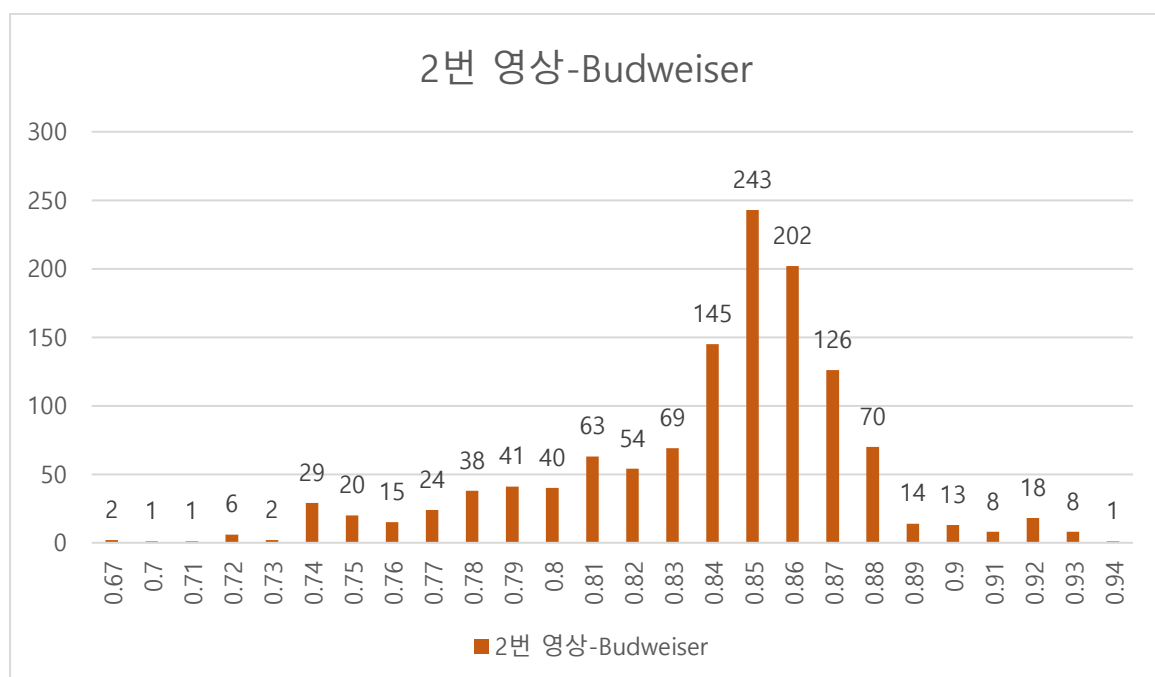


그림 25 - 2번 영상의 객체 유사도 분포

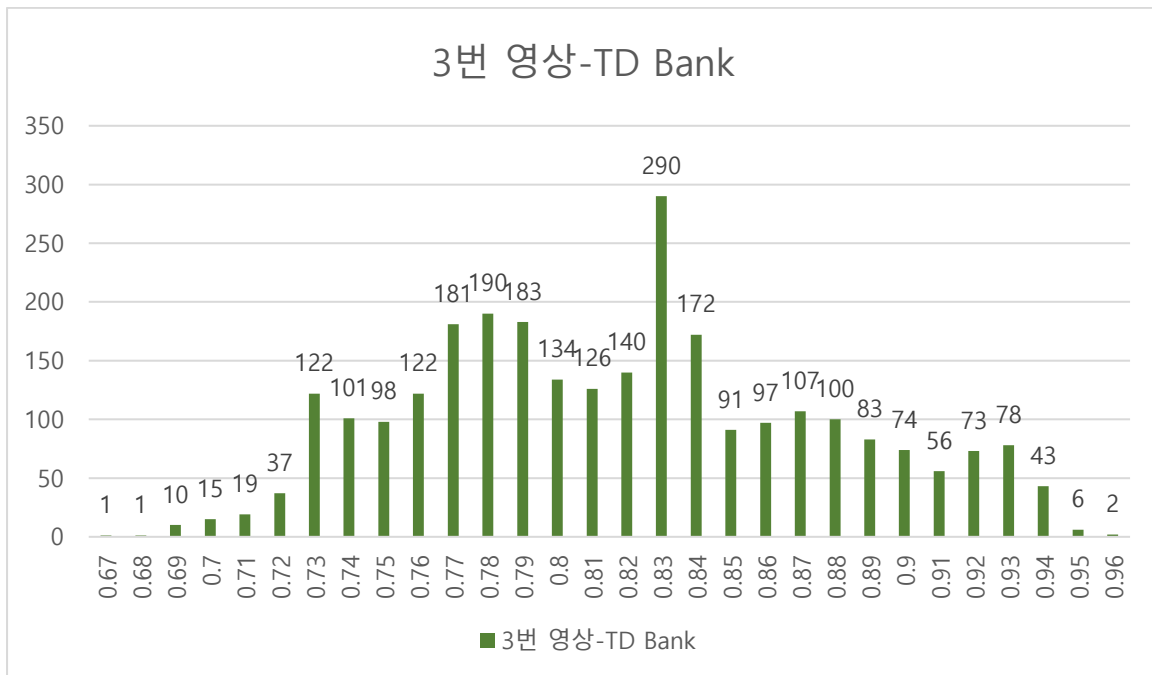


그림 26 - 3번 영상의 객체 유사도 분포

위의 도식은 Second Classifier에서 유사도 계산을 한 결과를 모아 시각화한 도식들이다. 도식에 따르면 1번 영상-trivago > 2번 영상-Budweiser > 3번 영상-TD Bank 순으로 분포의 평균값이 큰 것을 볼 수 있다. 이렇게 입력된 파일에 따라 다른 분포를 보이는 문제 때문에 기존 과제의 웹 프레임워크 방식에 문제가 발생하였다.

기존 과제의 웹 프레임워크에서는 threshold 값을 따로 입력 받지 않고 진행하려 했다. 그러나 위와 같이 입력된 비디오와 로고 이미지에 따라 다른 유사도 분포를 보이는 문제 때문에 부득이하게 사용자로부터 threshold value 값을 추가로 받는 방식을 취하였다.

5. 향후 연구 방향

본 과제에서는 이미지 지역화를 위한 학습에서 각각의 카테고리별로 무작위로 골라진 이미지로부터 이루어졌다. 그러므로 특정한 부분에서만 이루어지는 학습에 비해 많은 사례를 다룰 수 있게 되었지만, 그 대가로 정확도 측면에서 큰 손실이 있었다.

이를 해결하기 위해서는 datasets를 이루는 이미지에서 서로 간의 유사성이 어느 정도 보장되어야 한다. 이를 위해 여러 카테고리가 아닌 하나의 카테고리로부터 학습이 이루어져야 한다. 여기서 행해진 학습으로부터 나온 가중치 파일은 그 카테고리의 로고를 검출하는데 좋은 성능을 보일 것이다.

또한 웹 프레임워크에서 사용자로부터 찾으려는 로고의 카테고리를 입력 받는 과정을 추가한다. 즉, 선택할 수 있는 카테고리 별로 가중치 파일을 준비한 후에, 입력된 카테고리에 따라 각각 다른 가중치 파일을 사용해 검출을 진행한다. 위와 같이 진행하면 로고 객체 검출에서 좀 더 정확한 성능을 보일 수 있을 것이다.

마지막으로 유사도 계산에서도 정확도를 올릴 수 있을 것으로 보이는 개선 방안이 존재한다. 기존의 방식에서는 YOLOv5가 아닌 다른 학습 모델인 resnet18을 사용해 feature vector를 추출했다. resnet18 대신 본 과제에서 검출에서 사용했던 pretrained 모델을 사용해 feature vector 추출 후 유사도를 계산하는 방법도 있다. 로고에 대해 학습된 모델이기 때문에 로고에 관한 feature vector만을 추출할 수 있을 것이다.

6. 개발 일정 및 역할 분담

6월			7월				8월					9월			
13	20	27	4	11	18	25	1	8	15	22	29	5	12	19	26
Datasets 모집						유사도 계산 알고리즘 개발				최종 테스트					
웹 플랫폼 개발							검출 결과 출력 및 시각화								
			검출 모델 개발 및 테스트					웹 연동 및 배포 관리							
					중간 보고							최종 보고서 작성 및 발표 준비			

표 7 - 개발 일정

이름	작업
전승윤	웹 프레임워크 개발 Github 버전 관리 Docker 컨테이너 개발 및 배포 관리 Detection 알고리즘 개발
유동운	모델 학습 및 검증 리팩토링 및 테스트 담당
강태환	학습 데이터 생성 및 전처리 평가 지표 계산 및 평가
공통	필요한 개념 학습 보고서 작성 및 발표 준비

표 8 - 역할 분담

7. 참고 문헌

- [1] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao "YOLOv4: Optimal Speed and Accuracy of Object Detection"
- [2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg "SSD: Single Shot MultiBox Detector"
- [3] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár "Focal Loss for Dense Object Detection"
- [4] Jacob Solawetz - YOLOv5 New Version - Improvements And Evaluation. Available: <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>