<PROJECT 최종보고서>

SNUCSE x INNODEP

CCTV 영상 기반

이상행위 검출 모델 개발

이재필, 파티마, 박소정

신영길 교수님

INNODEP AI연구부문 문동민 리더

Table of Contents

1. Abstract

기존 CCTV 관제센터의 운영에는 많은 문제점과 한계가 존재하며, 관제 방식에 있어 AI학습모델의 도입과 같은 기술적 개선이 필요한 상황이다. 물론 CCTV 인공지능 감시기능을 개발하더라도, 아직 인간의 참여가 완전히 필요없다는 건 아니다. 두 가지 이유가 있는데, 우선 첫 번째는 기술적으로 '비정상행위'라는 것을 정확히 파악하는 모델이 현재까지 없다. 그리고 두 번째는 만약에 그러한 기술이 가능하다고 하더라도, CCTV가 감시하고 판단해야 하는 상황이 인간의 삶에 매우 큰 영향을 미치는 분야로, 인간의 최종적인 판단 없이 오로지 기계로만 최종 판단을 하는 게 부적절한 분야일 수 있기 때문이다.

따라서 본 프로젝트는 전적으로 인간을 도와주는 보조기능으로서, "CCTV 영상 기반 이상행위 감지, 분석 및 검출 학습모델"을 개발하는데 초점을 맞추었다. 이는 앞서 언급한 바와 같이 CCTV가 감시하고 판단하는 영역이 인간의 생명과 삶에 큰 영향을 미치는 분야로서 인간이 아닌 기계의 전적 판단에 의지하는 것이 부적절하며, 아울러 보조적인 역할에 머물도록 함으로써 아직은 완벽한 모델이 개발 되지는 않았지만, 적절하게 지금까지 개발된 모델과 기술을 융합하여 인간에게 이롭고 의미있는 제품을 만들어낼 수 있다고 판단하였기 때문이다.

이를 위해 우리는 비정상행위의 판단과 그에 따른 보조기능으로 얼굴인식을 조합하여 최종 제품을 만들고자 한다. 다시 말해, 기존 관제센터 운영 시스템의 감시체계에 이상행위 감지, 분석 및 검출가능한 AI학습모델을 도입하여, 기존 시스템의 한계를 극복하는 것이 목표이다.

목표를 달성하기 위해, Anomaly Detection과 Face Tracking Module을 개발하는 것이 주된 프로세스이다. 특히, Face Tracking Module 구현을 위해서는, 영상 속 인물의 얼굴 추적을 위해 처음 해당 얼굴영역을 detect하고 여기에서 나타난 image로부터 얻은 feature array와, 또 다른 영상에서의 얼굴영역을 detect하고 여기로부터 feature array를 얻어서 두 array의 유사성을 검사하고자 한다. 실험은 INNODEP에서 제공받은 비디오로 진행하고, Anomaly Detection과 Face Tracking System을 구동시켜보고자 한다.

2. Introduction

기존 CCTV 관제센터에서는 주로 인적인 요소에 많은 부분 의존을 하며 운영되고 있었다. CCTV의 개수도 많지 않고, 아주 특수한 부분에만 한정적으로 사용되었던 예전에는 인적인 요소에만 의존해서 해결하는 것이 경제적이고 합리적인 해결책일 수 있었다.

시간이 갈수록 CCTV의 필요성이 점점 높아졌고, 오늘날에는 CCTV의 사용이 매우 확장되어 그 수가 예전에는 상상할 수 없을 정도로 많아졌으며 범위도 굉장히 넓어졌다. 지금까지는 이에 대한 기술적인 요구가 주로 관리 통제의 부분이 아닌 하드웨어적인 부분과 네트워크, 코덱의 성능향상 등에 대해서 이루어져왔다.

그러나 이제 CCTV가 커버하는 일상의 영역이 커지고 그 숫자가 기하급수적으로 증가함에 따라, 관리 통제의 부분에 있어 전적으로 사람에게 의지하기 힘든 시점에 도달하였다. 즉, 기존 CCTV를 관제하는 센터의 운영 시스템에

여러가지 문제점이 있고, 단순 육안 감시만으로는 한계점이 분명하게 존재하는 상황이다.

기존의 기술 진보를 보여왔던 하드웨어, 코덱, 네트워크 부분과 달리, 운영, 관리에 대한 기술적 요구는 사람의 인지, 판단을 보완 대치할 수 있는 요구 사항을 가지기 때문에 기술적인 요구 수준이 매우 높았으며, 실질적으로 오늘날과 같이 인공지능이 도약을 이루기 전까지는 아주 간단한 룰-베이스의 방법론을 제외하고는 불가능으로 보였던 영역이었음이 사실이다.

하지만, 알파고를 필두로 인공지능의 가능성이 매우 크게 열리고, 그에 따라 여러 인공지능 모델이 개발됨에 따라서, 기존에는 거의 불가능으로 여겨졌던 분야들이 점차 정복되어가고 있다. 본 프로젝트에서는 여러 모델들을 살펴보며 이러한 분야에 CCTV 감시의 영역이 포함될 수 있음을 확신했고, 실제로 이 모델들을 우리가 만든 응용에 접목하여 보이고자 한다.

3. Background Study

A. 관련 접근방법/기술 장단점 분석

[Anomaly Detection에 대한 기존의 접근 방법]

i. 특정 행위에 대한 Anomaly Detection을 수행하는 방법

특정 행위를 판별하기 위한 rule-base algorithm을 개발하는 방법으로, 의미있는 첫번째 시도이기는 했지만, 비정상행위라는 것이 매우 광범위한 행위를 그 내용에 담고 있기 때문에 실제 비정상행위 탐지에 사용되기에는 부적절했다.

ii. sparse coding에 기반하여 Anomaly Detection을 수행하는 방법

실제 비디오에 찍히는 영상의 대부분은 정상행위라는 것에 기반하여, 카메라에 들어온 영상의 일부분을 정상행위 재구성을 위한 sparse coding 을 구하는데 쓰고, 그 이후로 들어오는 영상에 대해서 sparse coding을 이용하여 재구성할 수 있는지 확인하는 방법으로 비정상행위를 파악하는 방법이다.

하지만 실제 우리가 사용하는 CCTV영상이라는 것이 카메라가 고정 되지 않고 수시로 각도를 바꾸는 특징을 가지고 있으며, 매우 긴 시간동안 지속적으로 영상을 획득하기 때문에, 환경 측면에서 sparse coding map을 구성할 때와는 전혀 다른 환경을 가진 정상상황이 반복적으로 나타나기 때문에, false-positive율이 매우 높을 수 밖에 없어 사용하기 곤란한 점이 많다.

[Feature Extractor에 대한 여러가지 접근 방법]

Anomaly Detection을 하기 위해서는 우선적으로 화면에 나타난 여러가지 feature들을 뽑아내는 기능이 필요하며, 우리는 이것을 Feature Extractor라고 하기로 하였다.

하지만, 실질적으로 Feature Extractor 자체를 다루는 논문은 발견하지 못하였다. 대신 영상의 행위 분석을 위한 여러가지 시도가 있어왔으며, 대체적으로 Anomaly Detection은 해당 행위 분석 모델의 feature map을 그 parameter로 하여 개발하는 것이 그 기본이었고, 직관적으로 볼 때에도, 행위를 분석하기 위해 영상의 시공간의 의미를 분석하는 것이 비정상 행위를 판단하는 기본이 됨이 자명하기 때문에, 영상 행위 분석을 하기 위해 개발된 여러 모델들을 개조하여 그로부터 feature map을 뽑아 Anomaly Detection에 활용한다는 것이 우리의 기본 아이디어이며, 따라서 영상의 action recognition의 방법론을 바탕으로 가장 우수한 모델을 이용하여 우리의 Feature Extractor로 사용하고자 하였다.

이러한 action recognition 방법론으로는,

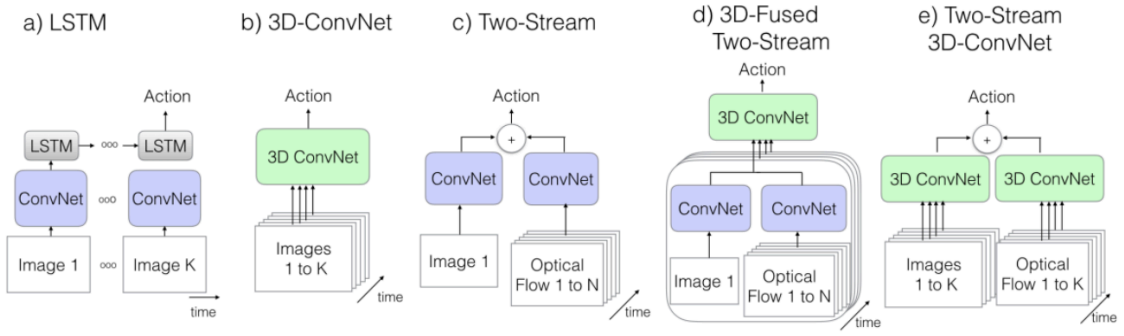
1. 기존의 2D ConvNet에 LSTM을 결합한 방법

2. 하나의 Image에 대한 2D ConvNet과 Optical Flow N frame에 대한 ConvNet을 결합한 Two-Stream기법

3. 2번의 모델의 최상부에 3D ConvNet을 결합한 방법

4. Two-Stream 3D ConvNet

등이 있다.



이러한 모델들을 이용하여 지금까지 여러 벤치마크가 있어왔으며, 그 결과 우리가 제안하고자 하는 Two-Stream I3D의 결과가 가장 우수한 것을 확인할 수 있다.

Architecture	UCF-101			HMDB-51			Kinetics		
	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow
(a) LSTM	81.0	–	–	36.0	–	–	63.3	–	–
(b) 3D-ConvNet	51.6	–	–	24.3	–	–	56.1	–	–
(c) Two-Stream	83.6	85.6	91.2	43.2	56.3	58.3	62.2	52.4	65.6
(d) 3D-Fused	83.2	85.8	89.3	49.2	55.5	56.8	–	–	67.2
(e) Two-Stream I3D	**84.5**	**90.6**	**93.4**	**49.8**	**61.9**	**66.4**	**71.1**	**63.4**	**74.2**

기존에 우리가 제안하였던 방법은 3D ConvNet이었는데, 이는 비디오에 특화된 모델이다. 따라서 직관적으로 보았을 때, video의 Anomaly Detection을 한다고 하였을 때, 가장 적합한 방법으로 생각이 되며, 그러한 이유로 많은 모델들이 이를 Feature Extractor로 사용하였다. 하지만, 이 방법의 경우 parameterization하기에는 너무 dimension이 높고, 이로 인해 학습시키는 것이 어렵다는 것이 첫번째 문제이며, 또한 비디오 전용의 모델이기 때문에, 라벨링된 비디오가 매우 중요한데, 그러한 비디오를 구하는 것은 기존의 Image dataset과는 비교도 되지 않게 힘든 측면이 있어 문제가 되며, 또한 기존에 개발되어 발전되어있는 수많은 2D Dataset과 2D ConvNet의 유산을 전혀 활용할 수 없음으로 인해서 아무것도 없는 상태에서 처음부터 모델을 구성하고 테스트 해야 했기 때문에, 모델 자체의 고도화 등에서 약점을 가질 수 밖에 없다.

이러한 문제점을 기반으로 나온 것이 Two-Stream기법, 3D-Fused, 그리고 우리가 제안하고자 하는 Two-Stream I3D와 같은 모델들이며, 이들은 기본적으로 기존의 2D ConvNet의 유산을 활용하고 이를 영상구조에 최적화 시키는 방향으로 발전시켜 트레이닝에 있어 기존의 ImageNet Dataset을 활용할 수 있다는 점이 큰 장점으로, 이를 통해 기본적인 영상학습 데이터셋인 kinetics-400과 결합하여 모델 최적화를 이룰 수 있었으며, 그 결과 3D ConvNet기법 대비 큰 향상을 이루었으며, 그 중에서도 Two-Stream I3D Model이 가장 우수한 결과를 보임을 확인할 수 있다.

따라서 우리는 action recognition에서 좋은 결과를 낸 모델과 같은 feature를 활용한다면, Anomaly Detection에도 분명 좋은 영향이 있을 것으로 판단하였고, 따라서 이를 도입하게 되었다.

[Face Detection, Face Recognition에 대한 기존의 접근 방법]

Face Recognition은 사람의 얼굴 특징을 기반으로 패턴을 분석하고 연관시켜 사람의 신원을 확인하거나 식별하는 데 사용되는 기술이다. Face Recognition을 이해하려면 먼저 Face Detection이라는 용어를 알아야 하는데, 이는 이미지 또는 동영상에서 얼굴을 감지하는 데 사용되는 기술로, 단지 Object Detection의 일부일 뿐이다. 그리고 Face Detection은 Face Recognition 프로세스의 첫 번째 단계라고 할 수 있다.

FR 파이프라인에서는, 먼저 Face Detector를 사용하여 얼굴을 localize한다. 이렇게 함으로써 Multitask Cascaded Convolutional Network를 사용한다. 그리고 bound box 알고리즘을 사용하여 한 프레임에 많은 bounding box를 얻는다. non max suppression을 적용한 후에는 input으로서 신경망에 제공될 box가 아주 적게 남고 그것이 얼굴인지 아닌지 여부를 확인한다. 그리고 facial landmark 좌표를 계산하는데, 이것은 이 모델에서 양쪽 눈, 코끝과

입술의 좌표를 계산한다.

이 모델은 3개의 Loss Function으로 구성되는데, 첫 번째는 Face Classification Loss이다. 이 Loss는 표본 i가 얼굴에 해당하는지 여부를 확인하고, 우리의 ground truth label을 기반으로 그것이 얼굴인지 아닌지를 알 수 있다.

Face Classification Loss:

$$L_i^{det} = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i)))$$

where pi is the probability produced by the network that indicates sample xi being a face.

그리고 두 번째 Loss는 Bounding Box Regression Loss로, 우리는 이 box의 좌표를 조정하려고 하며, 이는 예측된 box를 ground-truth와 비교한다.

Bounding Box Regression Loss:

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2$$

마지막으로 세 번째는 Facial Landmark Localization Loss로, 이는 코, 좌우 눈, 입가에 해당한다.

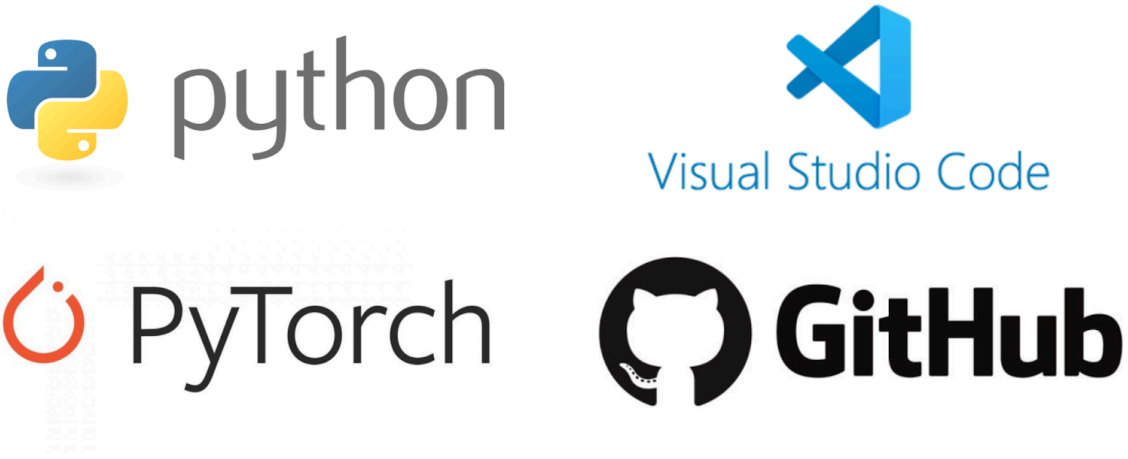
Facial Landmark Localization Loss:

$$L_i^{landmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2$$

For training the data : 0.3보다 적은 Intersection over Union이 있는 모든 영역은 negative한 경우이고, 우리는 이것을 background로 생각한다. 그리고 Intersection over Union이 0.65보다 큰 경우, part faces에 대해 IoU가 0.4에서 0.65 사이라면, positive한 경우이다.

그리고 나서, VGGFace2 및 CASIA-Webface에서 사전훈련된 resnet 모델을 사용했다. VGGFace2 데이터셋에 대해 간단히 설명하자면, 이는 9131개의 object로부터 331만개의 이미지를 포함하는 대규모 face 데이터셋이며, 각 subject에 대해 평균 362.6개의 이미지가 있다. 이미지는 Google 이미지 및 검색에서 다운로드되었고, 포즈, 나이, 조명, 민족 및 직업에서 큰 variation을 가진다. 또한, 각 identity의 이미지에 대해 자동화 및 수동 필터링 단계를 사용하여 높은 정확도를 보장한다. 따라서 이것은 Face Recognition을 위한 가장 인기 있는 데이터셋 중 하나가 되었다.

B. 프로젝트 개발환경



Python을 기본 개발 언어로 하였으며, Pytorch 프레임워크를 기본 프레임워크로 사용하였다. ide의 경우 visual studio code를 주로 사용하였다. 운영체제의 경우 osx, ubuntu linux, windows 10 등의 환경에서 진행하였으며, osx의 경우 gpu지원이 불가하여, 최종적으로는 ubuntu linux와 windows 10에서 수행하였다.

4. Goal/Problem & Requirements

A. Problem

기존에 CCTV를 관제하는 센터의 운영 시스템에서, 공공의 안전을 위해 설치되는 CCTV의 숫자는 기하급수적으로 늘어나고 있는데, 이러한 CCTV가 50% 증가하는 동안 이를 관제하는 모니터 요원은 14%밖에 증가하지 않았다. 다시 말해, 인력 충원 속도보다 CCTV가 늘어나는 속도가 훨씬 빠르다.

그리고 1인당 적정 CCTV 관제 수는 48대씩이지만 지금은 1인당 300~400대 정도를 관제하고 있다고 한다. 즉, 적정관제를 위해서는 대략 7000여명의 관제 요원 충원이 필요하며, 연간 1650억원의 인건비가 추가적으로 확보되어야 가능한 실정이다.

또한, 기존 감시체계는 단순육안감시로써, 관제를 하다보면 집중력이 저하되고 감시 정확도는 시간이 지날수록 눈에 띄게 감소하는 경향이 있었다. 때문에 객체의 움직임에 대한 의미 분석도 어려우며, 확대화면으로 활성화해야 객체의 의도를 조금이라도 더 정확하게 확인할 수 있는 상황이다.

B. Goal & Requirements

우리는 공공의 안전과 직결될 수 있는 연구과제로서 해당 주제가 굉장히 의미 있다고 판단하여 '영상 속 이상행위 상황 혹은 영역을 감지하고 분석 가능한 AI학습모델을 개발'해서 전국 최소 90만대의 CCTV에 해당 기술을 적용할 수 있도록 하자는 목표를 갖게 되었다. 이를 위한 구체적인 프로세스로, 우선 회사 측 요구사항으로는 영상 속에서 이상행위 상황을 감지하는 AI모델을 개발해내는 것이었고, 상호 간 논의 끝에, 구체적인 학습모델로는 Anomaly Detection을 구현하기로 결정했다.

해당 구현을 위해서는, UFC CRIME에 제시된 여러 anomaly한 상황에 대해서 auc 80% 이상의 달성을 목표로 하였다. 이를 위해서 UFC Crime Dataset을 이용하여 학습을 진행하였다. 이를 통해 실제 CCTV 화면에서 동일한 수준의 검출율을 보이는 것이 목표이다.

Face Detection을 구현하기 위해서는 INNODEP회사로부터 제공받은 Head Detector와, MTCNN Face Detector를 비교해보고 적절한 모듈을 선택하였다. Face Detection Model 구현은 둘 중 선택한 모듈(MTCNN Face Detector)을 이용하여 얼굴 영역을 출력해 바로 inception-resnet v1을 vgg-face2로 학습시킨 Face Recognition과 연동하여 사용한다. 그리고 마지막으로 inception-resnet v1을 vgg-face2로 학습시킨 Face Recognition 모델을 구현하여 Face Tracking Module을 완성시키고 Test를 진행한다.

구체적인 목표는 다음과 같다.

- Video or Photo image Training and Test Framework
- Anomaly Detection Model Research
- Anomaly Detection Model 구축
- Integration on the INNODEP system
- Face Recognition Model Research
- Face Detection on the Anomaly Video Frames
- Integration of Face Detection Model
- Face Recognition Module 구현
- Integration of Face Recognition on the system
- Check and Test Whole System Flow
- Optimization and Debugging
- Prepare Demonstration

5. Approach

A. Anomaly Detector Module

우리는 기본적으로 크게 두가지의 Approach가 있다. 하나는 Feature Extractor에 대한 부분이고, 다른 하나는 Anomaly Detection에 대한 부분이다.

우선 Feature Extractor에 대한 approach에서 앞서 설명한 것과 같이, 기존의 C3D 모델을 버리고, Two-Stream I3D 모델을 도입하였다. 그 이유는 앞서 살펴본 바와 같이, 구조자체의 복잡성과 비디오만을 이용하여 트레이닝할 수 있음으로 인해 생기는 문제, 그리고 기존의 뛰어난 2D ConvNet의 장점을 살리지 못하는 점, 그로 인해 모델자체를 처음부터 개발해야 해서 모델의 고도화 등에 있어서 단점이 많아 포기했다.

그렇다면 Two-Stream I3D를 사용한다고 하였을 때, 이를 어떻게 트레이닝하는 것이 가장 좋은 결과를 낼 것인가의 문제가 남아있다. 우리는 기본적으로 Feature Extractor가 인간의 행위 자체에 대한 kernel을 획득하기를 바라기 때문에 기본적으로 Kinetics-400 dataset을 이용할 것이다. 하지만, 앞서 밝힌 바와 같이 기본적으로 우리가 사용하는 베이스 모델이 가장 우수한 2D ConvNet 중의 하나임을 볼 때, ImageNet의 dataset을 미리 학습한 후에 획득된 weight에 Kinetics-400을 적용할 경우, 인간행위에 대한 kernel이 좀더 그 의미가 분명해지는 장점이 있을 수 있다는 예상을 할 수 있으며, 이를 적용하여 테스트한 결과는 다음과 같다.

	Kinetics			ImageNet then Kinetics		
Architecture	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow
(a) LSTM	53.9	–	–	63.3	–	–
(b) 3D-ConvNet	56.1	–	–	–	–	–
(c) Two-Stream	57.9	49.6	62.8	62.2	52.4	65.6
(d) 3D-Fused	–	–	62.7	–	–	67.2
(e) Two-Stream I3D	**68.4** (88.0)	**61.5** (83.4)	**71.6** (90.0)	**71.1** (89.3)	**63.4** (84.9)	**74.2** (91.3)

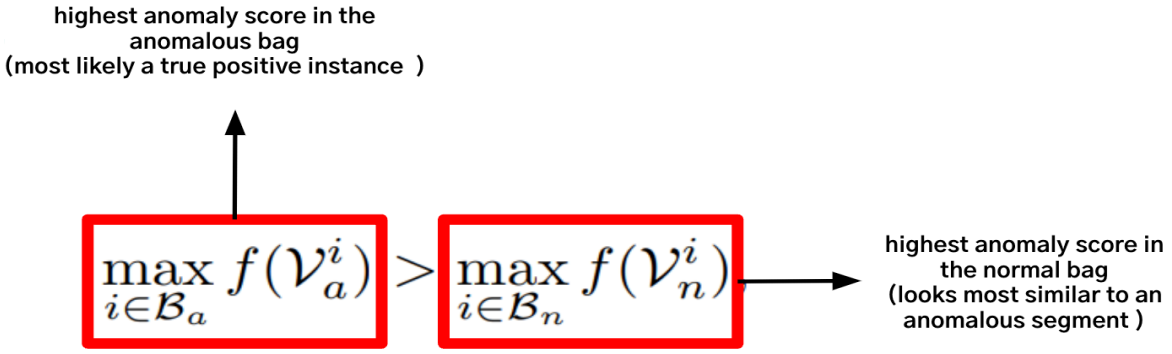
따라서, 우리는 ImageNet을 학습 시킨 이후 Kinetics 400을 학습한 Two-Stream I3D를 활용하여 feature map을 Anomaly Detector에 전달하도록 하는 approach를 사용하였다.

그리고 Anomaly Detection에 대해서는 다음과 같은 접근을 하였다.

우선, 우리가 진행하는 프로젝트의 가장 큰 특징은 실시간 비디오에 대한 분석을 한다는 점이다. 비디오에 대한 분석을 하기 위해서는 비디오를 바탕으로 하는 방대한 dataset이 요구된다. 하지만, image 데이터셋과는 달리 비디오의 경우 그러한 dataset이 매우 부족하고, 있는 자료도 그 수량이 우리의 연구 목표에 도달하기에는 턱없이 부족하다고 판단이 된다. 또한 우리는 일반적인 행위 등에 관한 분석을 시도하는 것이 아닌 비정상 상황이라는 매우 상황 의존적이고 모호할 수 있는 주제에 대해 분석을 하고 있기 때문에 일반적인 행위 비디오의 학습으로는 부족할 수 밖에 없다. 그리고 우리의 일상에서 비정상 상황이라는 것은 0.1%도 되지 않는 일이기 때문에, 비정상행위를 담은 비디오 자체를 구하는 것이 매우 어려운 일인 것이 사실이다.

이에 따라서 우리는 될 수 있으면 많은 비정상 상황에 대한 비디오를 학습하는 것을 목표로 하며 이를 위해서는 아주 정교하게 레이블링 된 비디오를 요구할 것이 아니라, 단지 정상 비정상으로만 구분된 비디오를 확보하여 이를 통해 비정상을 학습할 수 있도록 하는 것을 시도하였다.

이를 위해서 우리는 학습 프레임워크로 MIL을 도입하였으며, 해당 학습 프레임워크에서 우리가 다루는 것이 비디오라는 것에 입각하여 그에 맞는 적절한 Loss 함수를 도입하였다.



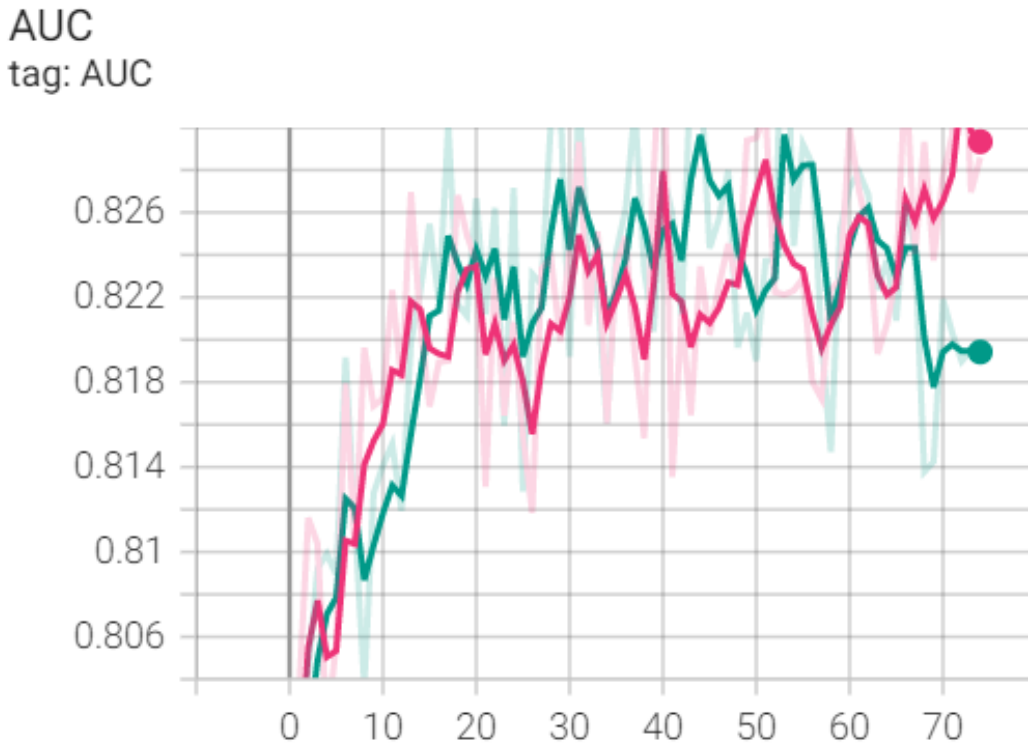
$$l(\mathcal{B}_a, \mathcal{B}_n) = \max(0, 1 - \max_{i \in \mathcal{B}_a} f(\mathcal{V}_a^i) + \max_{i \in \mathcal{B}_n} f(\mathcal{V}_n^i))$$
$$+\lambda_1 \overbrace{\sum_{i}^{(n-1)} (f(\mathcal{V}_a^i) - f(\mathcal{V}_a^{i+1}))^2}^{①} + \lambda_2 \overbrace{\sum_{i}^{n} f(\mathcal{V}_a^i)}^{②},$$

$f(\mathcal{V}_n^i)$ anomaly score (0~1)

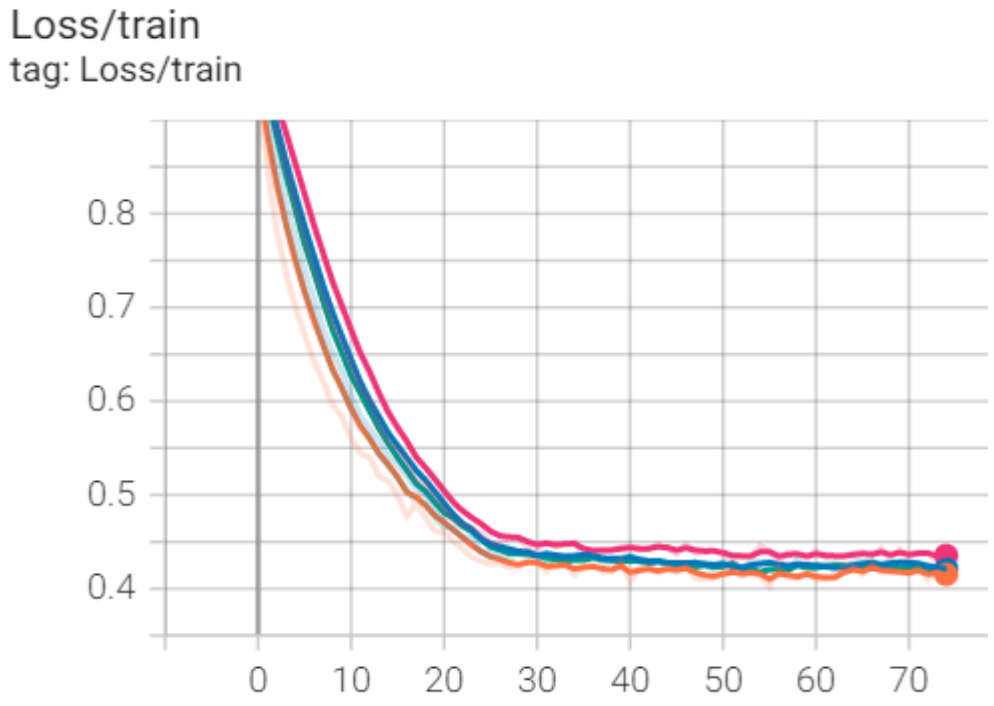
우리가 기존에 참조하던 논문에서는 C3D를 Feature Extractor를 사용하여 75.41%의 AUC를 얻었다.

Method	AUC
Binary classifier	50.0
Hasan *et al.* [18]	50.6
Lu *et al.* [28]	65.51
Proposed w/o constraints	74.44
Proposed w constraints	**75.41**

하지만 Two-stream I3D를 이용한 우리의 모델은 83.2%의 AUC를 달성하였다.



우리는 이를 위해서 75 epoch의 트레이닝을 거쳤고, 0.42의 loss score를 기록했으며, 추가적인 트레이닝을 하여도 loss 결과는 크게 변하지 않는 것을 볼 수 있다.



B. Face Tracking Module

특정 상황에 등장하는 인물의 얼굴을 추적하기 위해선, 우선 해당 상황에서의 얼굴영역을 감지해야 하고, 감지된 얼굴영역에서 나타난 얼굴이미지로부터 feature array를 획득하여야 한다. 그리고 이렇게 얻어진 feature array를 이용해서 다른 영상의 얼굴영역을 감지하고, 해당 얼굴로부터 역시 feature array를 획득하여, 문제 상황에서 얻어진 feature array와 유사성을 검사하여야 한다.

a. Face Detection

사용할 수 있는 Face Detection 모듈에는 두 개의 후보가 있었다. 첫째, INNODEP에서 제공해주신 INNODEP Head Detector와, 둘째, MTCNN Face Detector이다.

두 모듈은 각각의 장점과 단점이 있는데, 이는 정리하자면 다음 표와 같다.

	장점	단점
INNODEP Head Detector	빠른 속도 - 50ms 낮은 오탐지율	Face Recognition에 바로 적용하기 힘듬
MTCNN Face Detector	Face Recognition에 바로 적용가능	상대적으로 느린 속도 - 100ms 상대적으로 높은 오탐지율 - 패턴에 취약

INNODEP Head Detector의 경우, 속도가 매우 빠르고, 오탐지율이 매우 적고, 아주 작게 보이는 머리도 매우 잘 찾았다. 하지만, Face Detector가 아니기 때문에, 탐지된 Head 영역을 Face 영역으로 재가공할 수 있는 방법이 추가적으로 필요하다.

MTCNN Face Detector의 경우, 속도가 상대적으로 느리고, 얼굴의 패턴을 찾는 구조이기 때문에, 얼굴과 유사한 패턴을 보이는 경우 오탐지되는 경우가 상당히 많았고, 그에 맞추어 추가적인 필터링이 요구되었다. 하지만, Face Detection이 되어, 머리에서도 얼굴 영역만 정확하게 출력해주어 추가적인 작업 없이 Face Recognition과 연동하여 사용이 가능하다. 또 오탐지의 경우도 출력값에 Probability가 같이 출력이 되어 간편하게 필터링이 가능하여 오탐지의 경우가 크게 문제가 되지 않는다. 또한, 해당 모델의 성능이 약간의 최적화만 거치게 된다면 충분히 실시간으로 사용이 가능한 정도의 성능이다.

따라서 우리는 추가적인 작업이 필요한 INNODEP의 Head detector 모듈을 사용하지 않고, MTCNN을 사용하는 것으로 결정하였다.

그림1) MTCNN(좌) INNODEP Head Detector(우)

물론 *그림1)* 에서 보는 바와 같이 비슷한 상황에서 MTCNN의 결과와 INNODEP Head Detector의 결과를 비교해 보았을 때, INNODEP Head Detector를 MTCNN의 첫번째 P-Net 대신 사용하여 구조를 개선한다면, 빠르고 정확한 Face Detection 모델이 구현 가능할 것이다.

MTCNN Face Detection 모듈의 경우, gtx 1080에서 1280x720 이미지에서 100ms/frame의 성능을 보인다. 30fps의 비디오에 실시간으로 적용하기에는 무리가 있는 성능이다. 하지만, 우리가 분석 타겟으로 삼고 있는 비디오의 특성을 살펴보면, Face Detection을 위해서 모든 프레임을 분석할 필요는 없다는 것을 알 수 있다. 비디오의 각 프레임들은 인접 프레임들과 내용이 거의 같은 경우가 대부분이다. 인접 프레임들과 비슷하지 않은 내용이 있는 경우에도, 변화의 크기가 너무 클 경우, bitrate문제로 화면이 깨져서 분석이 불가한 경우가 되거나, 또는 동작이 너무 빨라서 모션블러 현상 등으로 인해서 분석이 불가할 가능성이 높으며, 그렇지 않다고 해도, 해당 이미지 변화를 이끌어낸 요소가 다음 프레임에서 바로 사라지는 경우는 역시 매우 드물기 때문에, 다음 프레임에서 분석해도 충분하다고 할 수 있다. 이러한 특성을 이용하여 홀수 또는 짝수 프레임에서만 Face Detection을 수행한다고 하면, 간단하게 성능을 2배 향상시킬 수 있게 된다.

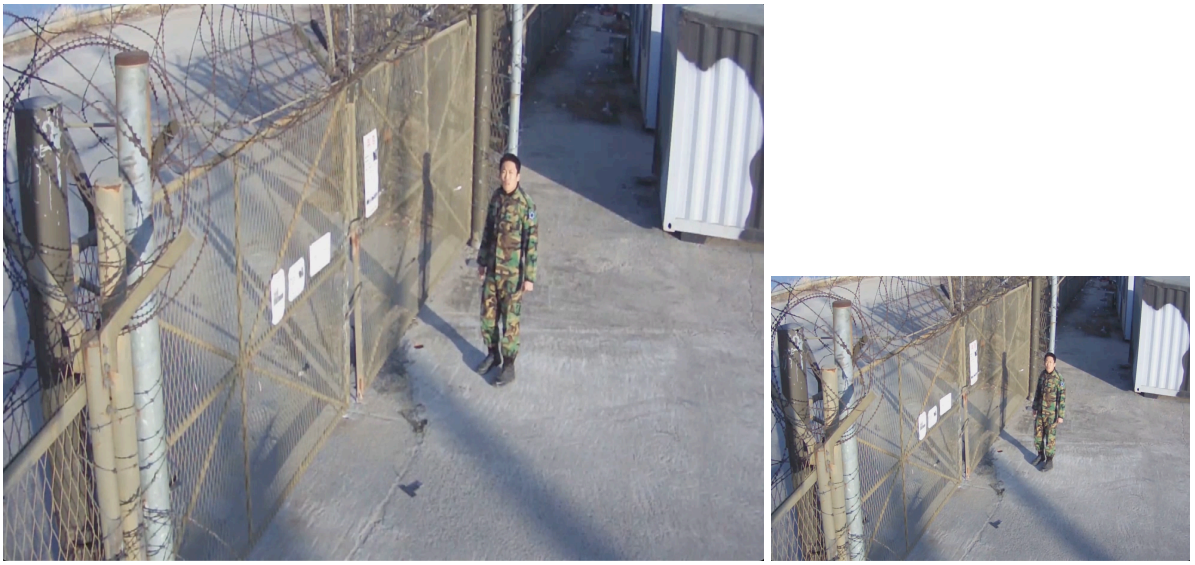


그림2) 해상도 변화에 따른 얼굴 정보 손실

또 한가지 최적화 방법으로는, 소스 비디오의 해상도를 낮추는 방법이 있다. 하지만, *그림2)* 에서 보면 알 수 있듯이,

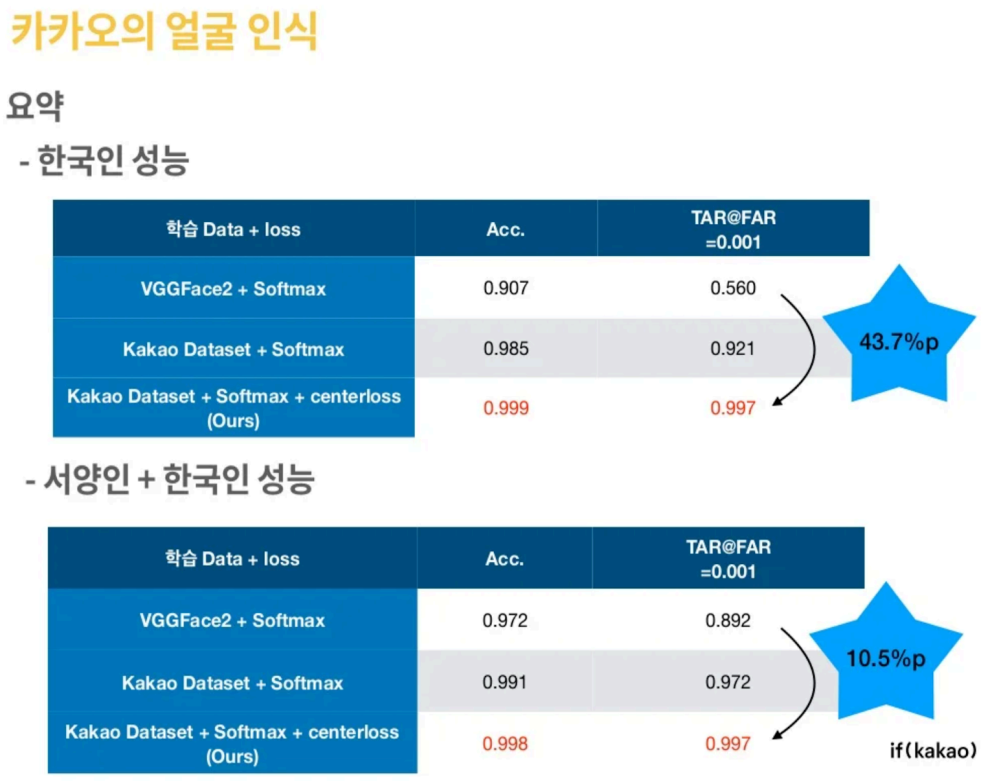
우리가 분석 대상으로 삼고있는 CCTV에서 인물의 얼굴이 화면에서 차지하고 있는 영역이 대부분 충분히 크지 않다. 따라서 해상도를 낮출 경우, Face Detection 자체가 되지 않거나, 해당 이미지로부터 얻을 수 있는 embedding의 내용이 인물의 동일성 여부를 판단하기에는 충분하지 않은 값으로 출력될 가능성이 매우 높다.

따라서, 우리는 10fps의 속도로 분석을 하도록, 소스비디오의 fps를 이용하여 얼마만큼 건너뛸지 결정하도록 하였으며, 이를 통해서 모든 프레임을 분석한 것과 질적으로도 차이가 없으면서 실시간 분석이 가능하도록 하였다.

b. Face Recognition

Face Recognition을 위해서 기본 모델로서는 inception-resnet v1을 vgg-face2로 학습시킨 것을 사용하였다. 기본적으로 매우 좋은 성능을 보이지만, vgg-face2가 주로 서양인의 얼굴 위주로 구성이 되어있어, 이를 한국인의 얼굴 인식을 위해 사용할 경우 문제가 생길 가능성이 예상이 되었다. 이는 중간 발표때, video에서 감정 분석을 하는 프로젝트를 진행하는 팀에서도 언급하였던 문제이기도 했다.

카카오 AI팀에서 Face Recognition 관련 개발을 할 때의 기록을 인터넷에서 찾을 수 있었다. 해당 내용은 vgg-face2를 이용하여 학습시킨 모델을 사용하였을 때 오차가 원래 논문의 결과보다 훨씬 크게 나타나고, 이를 개선하기 위해서 한국인 얼굴 데이터를 6개월 정도 모아서 학습시킨 결과 매우 좋은 결과를 얻었다는 것이었다. 그들이 발표한 성능 향상 자료는 다음과 같다.



하지만, 우리 여건상 vgg-face2의 40GB가 넘는 데이터를 학습시키는 것도 일정 상 녹록하지 않았으며, 더더욱이 해당 모델의 타겟 환경과 달라 발생하는 문제점을 개선할 정도의 얼굴 데이터를 구하고, notation을 하는 것은

불가능하였다. 하지만, 추후에 한국인 얼굴 데이터를 이용할 경우 좋은 결과를 얻어낼 수 있음을 확인할 수 있었으며, 추가 학습을 통해 우리가 얻은 결과보다 더 좋은 결과를 얻을 수 있을 것임을 알 수 있다.

Anomaly 판단을 위해서 우리가 전달하는 이미지 시퀀스는 64개이다. 100ms마다 하나의 프레임을 분석을 하기 때문에 총 6400ms, 6.4초의 기간안에 나온 얼굴 데이터를 찾아야 한다. 하지만, 해당 시퀀스에서 사람의 얼굴이 Face Detection을 통과하는 경우도 매우 드물었으며, Face Detection을 통과한다고 해도, 얼굴의 포즈가 한쪽으로 치우쳐 있던지, 블러가 심하게 들어가 있는 경우가 많아 Face Recognition에 적합한 데이터가 아닌 경우가 많았다. 또 문제 상황에서 대부분 인물이 하나가 아니라 둘 이상이 서로 상호작용을 주고 받는 경우가 많다. 두 인물이 동시에 Face Detection을 통과하고, 그 데이터가 Face Recognition에 적합한 데이터일 것을 기대하기는 힘들다.

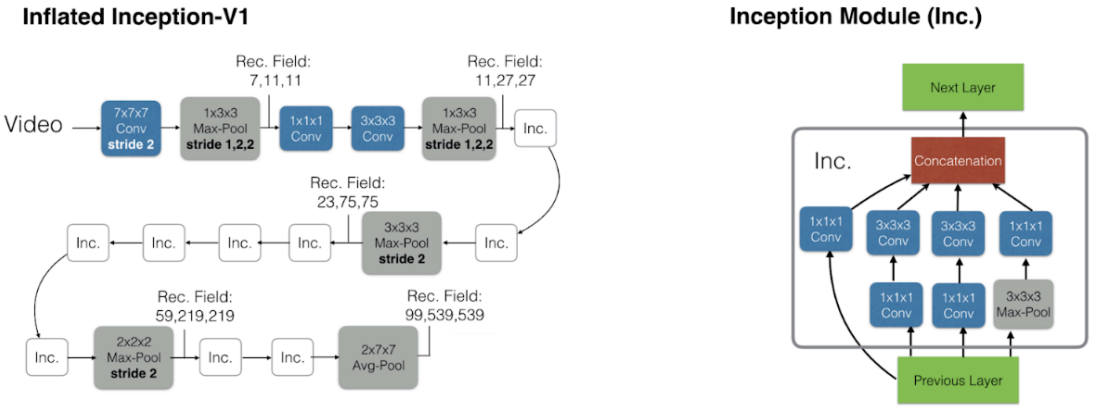
Anomaly 상황은 우리가 분석하는 64 frame의 시퀀스 하나에서 끝나지 않고, 인접한 시퀀스 사이에서 상황이 연속되는 경우가 대부분이다. 하나의 시퀀스에서 얼굴을 찾을 확률이 p라고 하면, n개의 시퀀스에서 얼굴을 찾을 확률은 np라 할 수 있다. 따라서 우리는 Anomaly 상황이 처음 확인된 시퀀스부터, 3개 이상의 시퀀스에서 Anomaly 상황이 검출이 되지 않을 때까지 얼굴 정보를 찾아서 누적시키고, 이를 이용하여 다른 비디오 소스에서 해당 인물과 유사 인물이 있는지 검출하도록 하였다.

이렇게 누적된 Face Embedding정보를 사용함으로써 최초 하나의 Face Embedding만 사용하였을 때는, 0.95미만의 거리수치일 경우 사람이 개입해서 확인해야 하는 정확도를 보였던 것이, 0.7미만으로 정확도를 향상시킬 수 있었고, 이로 인해 인간의 확인이 필요한 경우의 수를 줄일 수 있었다.

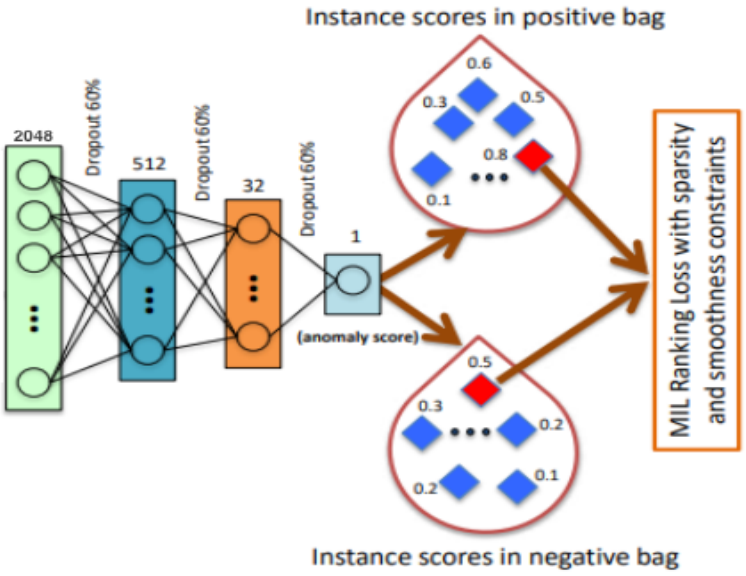
6. Project Architecture

A. Architecture Diagram

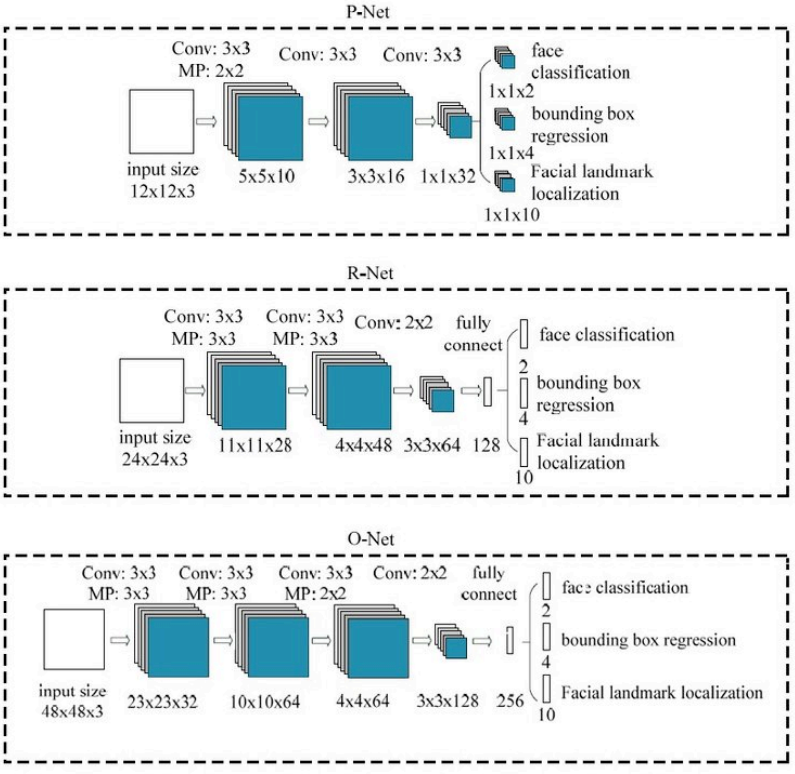
[I3D Model 구조]



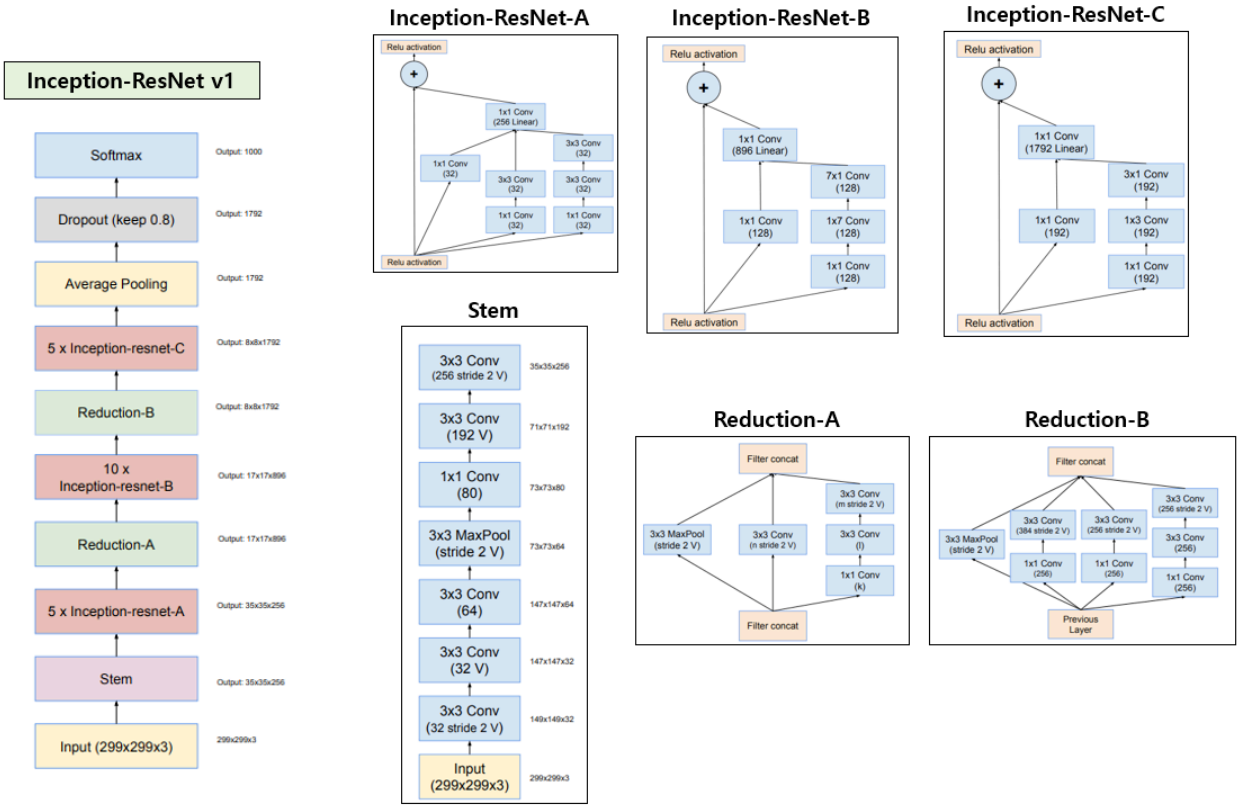
[Learner(Anomaly detection) Model구조]



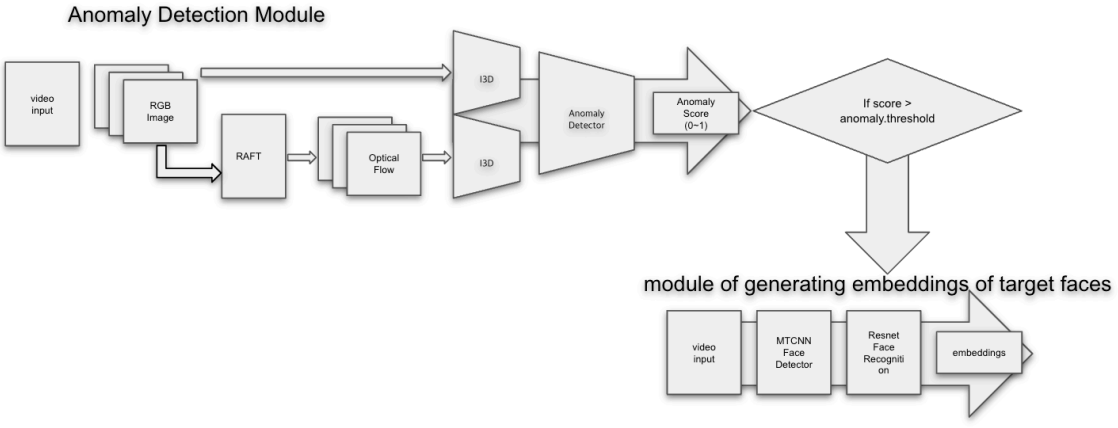
[MTCNN에 사용된 P-Net, R-Net, O-Net의 구조]

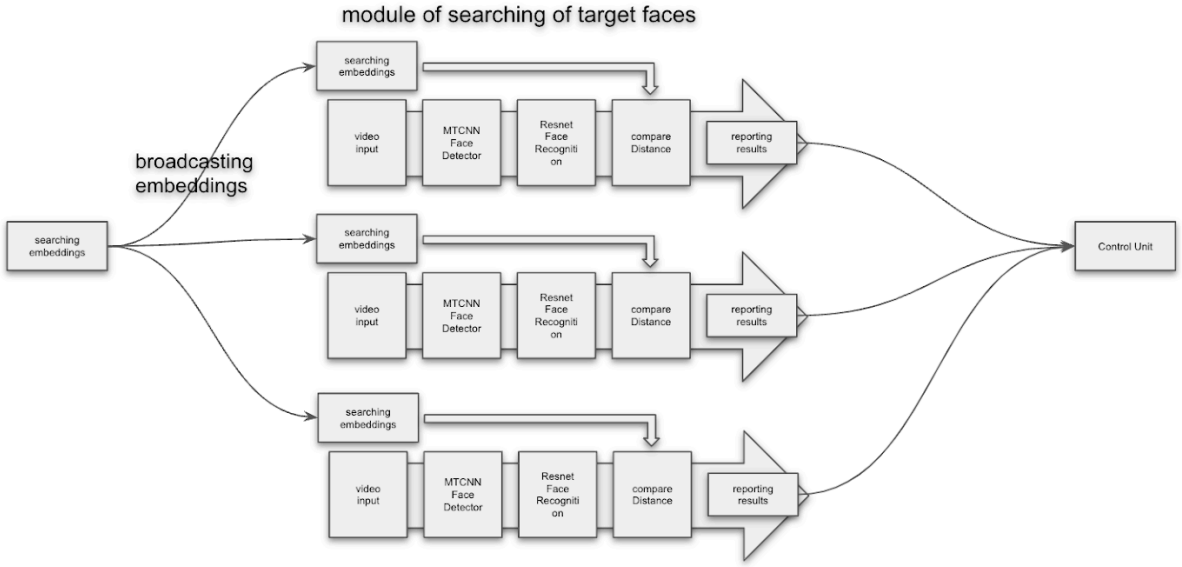


[Face Recognition에 사용된 inception-resnet v1의 구조]



[Anomaly Detection System 전체 구조 다이어그램]





B. Architecture Description

비디오 인풋을 받게 되면, 이를 ExtractI3D Module 내부에서 rgb frame으로 지정된 frame만큼 추출해 낸다. 이렇게 추출된 rgb frame을 이용하여 I3D Module에서 feature map을 1024개를 추출해 내고, raft module에 rgb frame을 parameter로 넘기고 forward를 호출하게 되면, optical flow가 추출이 되고, 이렇게 추출된 optical flow를 I3D Module에 넘겨서 forward를 호출하게 되면 다시 1024개의 feature map이 추출된다. 이렇게 추출된 feature map을 합쳐서 2048개의 parameter로 learner module(anomaly detector)에 넘기고 forward를 호출하게 되면, 최종적으로 anomaly score를 0~1사이의 값으로 출력하게 된다.

여기에서 쓰인 I3D Module의 경우 기존의 2D ConvNet인 Inception-V1을 dimension을 추가하여 시간축에 대한 분석을 추가한 모델로, 우리는 이를 inflated inception-v1이라고 부른다.

Learner module의 경우 이렇게 입력한 2048개의 입력을 3번의 drop-out layer를 통해서 anomaly score를 추출해 낸다.

Anomaly score가 0.2이상인 경우 해당 상황이 문제가 있는 것으로 판단하여 generateEmbeddings module로 rgb frame sequence를 전달하게 된다. 해당 모듈에서는 Face Detection을 수행하여 n개의 얼굴이 찾아지면 해당 정보를 Face Recognition 모듈로 전달하여 embedding정보를 추출하게 된다. 해당 embedding 정보는 list에 누적이 되고, anomaly score가 0.2 이상인 상황이 지속되는 동안 계속해서 최대 20개의 제한을 두고 누적하게 된다. anomaly 상황이 종료로 판단되는 경우나 face embedding list의 entry가 20개가 되는 경우 해당 embedding list를 broadcasting하여 다른 비디오 소스에서 checkEmbedding을 수행하도록 한다. checkEmbedding은 자신의 비디오 소스에서 얼굴 정보를 추출하여 전달된 face embedding list와 compareDistance함수를 통해 유사도 측정을 하여 해당 수치가 0.7미만인 경우 결과 수치와 해당 상황의 이미지를 리포팅하게 된다.

7. Implementation Spec

A. Input/Output Interface

a. input

1) SurveillanceManager Module

- mp4 영상의 url

2) generateEmbeddings Module

- rgb frame list

3) CheckEmbedding Module

- rgb frame
- embedding list

b. output

1) SurveillanceManager Module

- anomaly score (float)

2) generateEmbeddings Module

- embedding list

3) CheckEmbedding Module

- image, score(float)

B. Inter Module Communication Interface

torch.nn.Module을 기본 interface로 하고 있다. pytorch의 기본 인공지능 모듈 인터페이스이다.

C. Modules

a. I3D Module

기본적인 I3D module의 feature extract 기능을 담당하는 Module

b. Raft Module

Optical flow를 extract 하는 Module

c. Learner Module

Anomaly Detection Module

d. MTCNN Module

Face Detection Module

e. InceptionResnetV1 Module

Face Recognition Module

f. ExtractI3D Module

위의 3개의 모듈을 이용하여 video의 frame을 읽어와서 rgb프레임을 생성하여 이를 I3D Module에 입력하여 1024개의 feature를 뽑고, raft 모듈에 rgb프레임을 입력하여 optical flow를 생성하고, 생성된 optical flow를 I3D Module에 입력하여 1024개의 feature를 뽑는다. 이를 learner모듈에 입력하여 anomaly score를 추출하는 역할을 한다.

g. SurveillanceManager Module

video stream으로부터 영상을 rgb로 추출하여 extract i3d module에 feed하고, 그 결과 수치가 0.2 이상이 나올 경우, 해당 rgb stream을 generateEmbeddings에 입력하여 embedding list를 획득한다. 이상 상황이 발생한 이후 anomaly score가 연속 3회 0.2미만의 수치가 나오게되면 anomaly 상황 종료가 되고, 이 때 획득한 embedding list의 길이가 0보다 클 경우 embedding list를 브로드캐스팅하게 된다.

h. CheckEmbedding Module

각각의 video stream 처리 장치에서 embedding list를 전달 받게 되면, 일정 주기마다 rgb frame을 CheckEmbedding Module에 입력하여서 유사성 점수와 해당 상황이 존재할 경우 box 표시된 이미지를 전달 받게 되고, 이를 리포팅하게 된다.

8. Solution

A. Implementation Details

주요 클래스와 각 클래스의 주요 method들은 다음과 같다.

1) Optical Flow 생성을 위한 RAFT class

```
class RAFT(nn.Module):

        def forward(self, image1, image2, iters=20, flow_init=None, upsample=True, test_mode=True)
```

2) Feature Extraction을 위한 I3D class

```
class I3D(torch.nn.Module):

        def forward(self, inp, features=None):

            # Preprocessing

            out = self.conv3d_1a_7x7(inp)

            out = self.maxPool3d_2a_3x3(out)

            out = self.conv3d_2b_1x1(out)

            out = self.conv3d_2c_3x3(out)

            out = self.maxPool3d_3a_3x3(out)

            out = self.mixed_3b(out)

            out = self.mixed_3c(out)

            out = self.maxPool3d_4a_3x3(out)

            out = self.mixed_4b(out)

            out = self.mixed_4c(out)

            out = self.mixed_4d(out)

            out = self.mixed_4e(out)

            out = self.mixed_4f(out)

            out = self.maxPool3d_5a_2x2(out)

            out = self.mixed_5b(out)

            out = self.mixed_5c(out)
```

```
                out = self.avg_pool(out)

                out = out.squeeze(3)

                out = out.squeeze(3)

                out = out.mean(2)
```

3) Anomaly Detection을 위한 Learner class

```
class Learner(nn.Module):

    def forward(self, x, vars=None):

        if vars is None:

            vars = self.vars

        x = F.linear(x, vars[0], vars[1])

        x = F.relu(x)

        x = F.dropout(x, self.drop_p, training=self.training)

        x = F.linear(x, vars[2], vars[3])

        x = F.dropout(x, self.drop_p, training=self.training)

        x = F.linear(x, vars[4], vars[5])

        return torch.sigmoid(x)
```

4) Face Detection을 위한 MTCNN class

```
class MTCNN(nn.Module):

  def forward(self, img, save_path=None, return_prob=False)

  def detect(self, img, landmarks=False)

  def extract(self, img, batch_boxes, save_path)
```

5) Face Recognition을 위한 InceptionResnetV1 class

```
class InceptionResnetV1(nn.Module):

    def forward(self, x):

        x = self.conv2d_1a(x)

        x = self.conv2d_2a(x)

        x = self.conv2d_2b(x)

        x = self.maxpool_3a(x)
```

```
        x = self.conv2d_3b(x)
        x = self.conv2d_4a(x)
        x = self.conv2d_4b(x)
        x = self.repeat_1(x)
        x = self.mixed_6a(x)
        x = self.repeat_2(x)
        x = self.mixed_7a(x)
        x = self.repeat_3(x)
        x = self.block8(x)
        x = self.avgpool_1a(x)
        x = self.dropout(x)
        x = self.last_linear(x.view(x.shape[0], -1))
        x = self.last_bn(x)
        if self.classify:
            x = self.logits(x)
        else:
            x = F.normalize(x, p=2, dim=1)
        return x
```

6) Anomaly Score를 추출하기 위한 ExtractI3D class

```
class ExtractI3D(torch.nn.Module):
  def extract(self,
              device: torch.device,
              flow_xtr_model: torch.nn.Module,
              anomalyDetector: torch.nn.Module,
              mtcnn: torch.nn.Module,
              resnet: torch.nn.Module,
              models: Dict[str, torch.nn.Module]
              )
```

7) video stream을 관리하고, extractI3D에 비디오 시퀀스를 제공하는 기능을 하는 *SurveillanceManager* class

```
class SurveillanceManager:

  def do(self, video_path: str)
```

8) Broadcasting으로 들어온 embedding list와 현재 처리중 비디오에 나타난 인물의 유사도를 검사하는 CheckEmbedding class

```
class CheckEmbedding:

    def check(check_embeddings, frame):

        frame = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))

        allboxes, probs = mtcnn.detect(frame)

        boxes = []

        bestFound = False

        if allboxes is not None:

            for i in range(0, len(allboxes)):

                if probs[i] > 0.99:

                    boxes.append(allboxes[i])


        if len(boxes) > 0:

            boxes=np.array(boxes)

            faces = mtcnn.extract(frame, boxes, save_path=None)

            papas = faces.chunk(len(boxes), dim=0)

            aligned = []

            for e in papas:

                aligned.append(e.view([-1,160,160]))

            aligned = torch.stack(aligned).to(device)

            embeddings = resnet(aligned).detach().cpu()

            idx = 0

            scores = [100]*len(boxes)

            for embedding in check_embeddings:
```

```
            for c1 in embedding:
                for e1 in embeddings:
                    dist = compareDistance(c1 - e1)
                    if dist < scores[idx]:
                        scores[idx] = dist
                        if dist < self.bestScore:
                            self.bestScore = dist
                            bestFound = True
                    idx+=1


        if bestFound and self.bestScore < 0.7:
            # Draw faces
            frame_draw = frame.copy()
            draw = ImageDraw.Draw(frame_draw)
            idx = 0
            for box in boxes:
                if scores[idx] < 0.7:
                    draw.rectangle(box.tolist(), outline=(255, 0, 0), width=6)
                else:
                    draw.rectangle(box.tolist(), outline=(0, 0, 255), width=6)
                idx+=1


            # Add to frame list
            numpy_image = np.array(frame_draw)
            cv_image = cv2.cvtColor(numpy_image, cv2.COLOR_RGB2BGR)
            return cv_image
    return None
```

B. Implementation Issues

Anomaly Detection & Face Tracking 시스템을 구축하는데 있어서 약간이나마 문제가 되었던 내용들은 다음과 같다.

1) performance issue

개발 초기 예상에는 Anomaly Detection Model과 Face Recognition Model의 속도가 문제가 될 것으로 예상했다. 하지만 실제 개발을 하고 테스트를 해보니, MTCNN Face Detector와 RAFT Optical flow extractor가 속도와 관련된 성능의 대부분의 문제를 일으키는 것을 확인할 수 있었다.

이에 대한 해결 방법으로 ROI를 지정함으로써 해결하는 방법을 생각해 볼 수 있었으나, 될 수 있으면 사람의 반복적인 노력과 개입이 들어가는 방향은 지향하는 것이 추후 활용성을 높일 수 있다고 판단하여 해당 최적화는 시도하지 않았다.

대신, 모든 프레임을 체크하는 것이 아니라, 100ms에 하나의 프레임을 처리하는 것으로 방향을 전환했다. 이를 위해 stream을 지정할 때 해당 stream의 fps를 명기하도록 하였으며, fps를 이용하여 몇번의 frame을 건너뛸지를 계산하여 n번 프레임에서 1번 처리의 형태로 구성하였다. 내부적으로 이를 stride라고 표현했으며, 간단하게

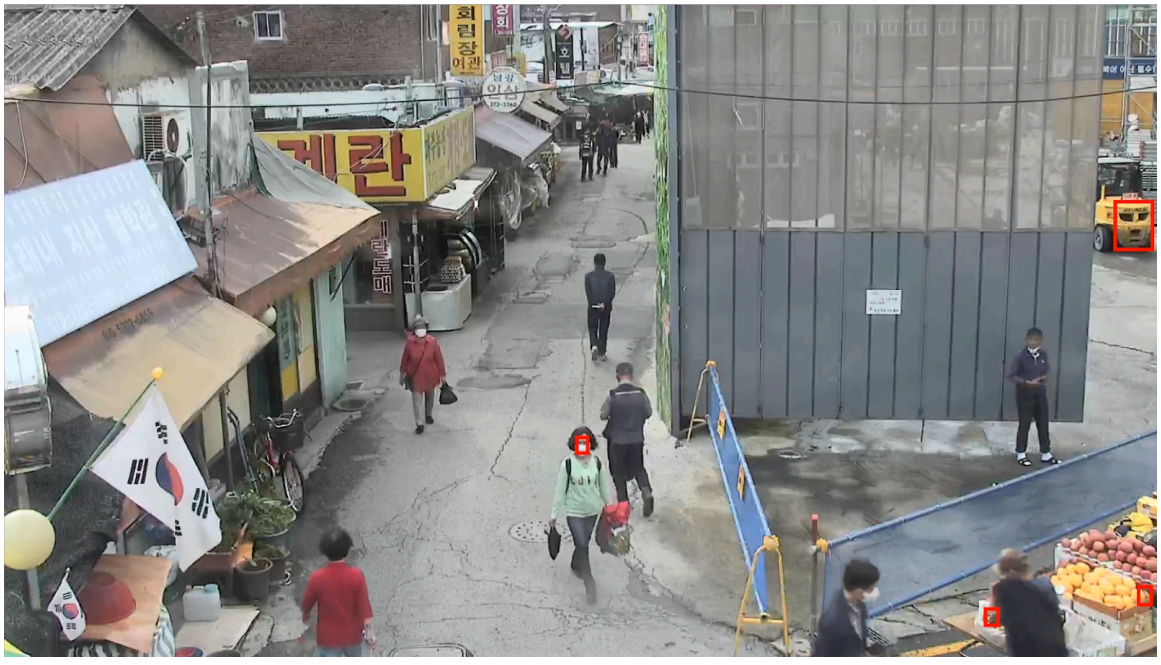
$$self.stride = math.ceil(fps/10)$$

으로 계산하여, 건너뛰도록 했고 이를 통해서 시스템 전체적으로 gtx1080기준으로,

$$(\text{비디오 소스 길이})/(\text{처리 시간}) \approx 1$$

을 달성하였다.

2) Face Detection error



MTCNN의 경우 위의 캡쳐화면에서 볼 수 있듯이, 이목구비의 형태소를 가진 부분을 얼굴로 인식하는 문제가 있다. 따라서 이것을 필터링 해주어야 하였다.

```
# Detect faces
frame = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
allboxes, probs = mtcnn.detect(frame)
boxes = []
if allboxes is not None:
    for i in range(0, len(allboxes)):
        if probs[i] > 0.99:
            boxes.append(allboxes[i])
```

위와 같이 일단 출력하되 box들을 순회하면서 probability 수치가 0.99 이상인 경우만 Face Recognition에 입력할 box로 따로 정리하도록 하였다.

3) multi Face Recognition on one frame

```
if len(boxes) > 0:
  boxes=np.array(boxes)
  faces = mtcnn.extract(frame, boxes, save_path=None)
  papas = faces.chunk(len(boxes), dim=0)
  aligned = []
  for e in papas:
      aligned.append(e.view([-1,160,160]))
  aligned = torch.stack(aligned).to(device)
  embeddings = resnet(aligned).detach().cpu()
```

MTCNN에서 face detect 영역을 extract메소드에 입력할 경우 aligned된 face들을 출력해준다. 문제는 해당 face list들이 모두 합쳐져 있는 형태로 나오기 때문에, 이를 Face Recognition에 적합하도록 다시 분리해서 입력해주고 이를 통해서 embeddings를 얻어오게 된다.

4) Broadcasting embedding lists

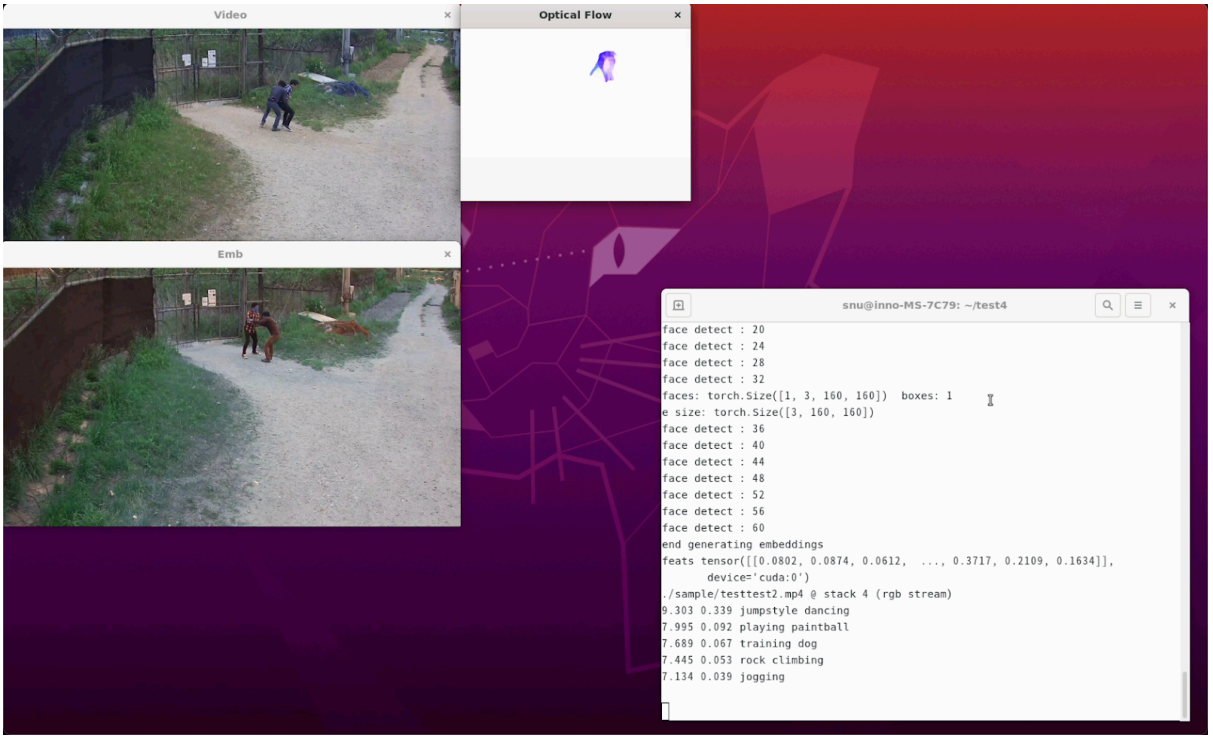
```
if len(stack) - 1 == self.stack_size:
        result, embeddings = anomalyDetector(feats_dict, stack, device, stack_counter, rgb_frames, padder)
        if result > 0.2:
                peaceCnt = 0
        self.list_of_embeddings.extend(embeddings)
        if len(self.list_of_embeddings) >= 20:
                        Broadcasting(self.list_of_embeddings)
                self.list_of_embeddings = []
        else:
                peaceCnt += 1
                if peaceCnt > 3 and len(self.list_of_embeddings) > 0:
                    Broadcasting(self.list_of_embeddings)
                    self.list_of_embeddings = []
                    peaceCnt = 0
```

Broadcasting의 경우 누적시킨 embedding의 entry가 20개 이상이거나, peaceCnt가 3회보다 커서 실질적으로 상황이 종료되었고, embedding list의 entry가 존재하는 경우 broadcasting을 하게 된다.

각각의 video stream 처리 모듈에서 broadcasting받은 embedding list를 CheckEmbedding 모듈에 입력하여, 현재 rgb frame에서 나오는 인물정보와 대조를 하여 0.7미만의 차이점을 보이는 인물이 나타날 경우 보고를 위한 image를 구성하여 리포트한다.

초기에는 이런식으로 하지 않고 무조건 인물을 하나 이상 찾기만 하면 바로 broadcasting을 하도록 하였는데, 이렇게 되니 찾아진 인물의 pose등이 분석하기 어려운 pose이거나, 블러링등으로 0.95이하만 되면 리포팅 하도록 하였다. 그렇지 않으면 실질적으로 리포팅이 거의 이루어지지 않았고, 그럴 경우 더 문제가 될 수 있다고 판단해서였다. 그러나 이럴 경우 false positive rate가 너무 올라가게 되어 시스템적으로 의미가 약하다고 판단하여 위와같이 최대한 인물정보를 추출하고, 그를 바탕으로 진짜 비슷하게 생긴 경우만 리포팅할 수 있도록 0.7로 기준값을 바꾸어 어느정도 만족할 수 있는 결과를 갖게 되었다.

9. Results



실험장면1) RGB Stream, Optical 스트림을 이용하여 Anomaly Detection을 하고, anomaly 상황에서 embedding을 뽑아내는 것을 보여주기 위한 Emb창이 떠있는 모습

실험장면2) 최종 리포트 화면 - 동일 인물로 판단되는 인물의 얼굴에 붉은색 박스가 그려져있고, 총 4개중 3개의 비디오에서 해당 인물을 발견한 모습

A. Experiments (실험 진행)

1) 실험 소스

INNODEP 제공 검증 비디오 16개

- Anomaly 상황이 담긴 비디오의 개수 7개 - 싸움, 유기, 침입 등
- Anomaly 상황이 담기지 않은 비디오의 개수 9개 - 복잡한 동네 뒷골목, 복잡한 패턴이 그려진 공원, 사람들이 무리지어 다니는 도로, 구내 식당 입구, 배식대, 테이블석 등

Face Recognition용 비디오 4개

2) 실험의 진행

(1) SurveillanceManager 모듈에서 INNODEP 검증 비디오를 하나씩 읽어서 Anomaly Detection & Face Tracking System을 구동 시킨다

(2) 로그와 비디오의 상황을 비교하여 현재 제대로 상황 인식이 되고 있는지 체크한다.

(3) Anomaly 상황에서 Face Detection 여부를 확인하고, broadcasting이 이루어졌을 때, 다른 소스 비디오에서 해당 인물을 찾는지 리포트 이미지를 통해서 확인한다.

B. Result Analysis and Discussion (실험 결과)

1) 실험 결과 요약

전체 test video - 16개

Anomaly 상황을 담은 비디오 클립의 개수 - 7개

Face Recognition 체크를 위한 비디오 클립의 개수 - 4개

Anomaly 상황을 Anomaly로 판단한 경우 - 6개

Anomaly 상황이 아닌데 Anomaly로 판단한 경우 - 1개

Anomaly 상황에서 Face Detection에 성공한 경우 - 3개

Face Detection에 성공한 경우, Face Recognition에 성공한 경우 - 8개

2) 실험 결과 토의

INNODEP에서 제공해 주신 16개의 테스트 비디오와, Face Recognition을 찾는데 사용한 비디오 4개를 이용하여 테스트를 진행하였다. 16개의 비디오 가운데 Anomaly 상황을 담은 비디오의 개수는 7개였다. 이를 이용하여 Anomaly Detection 체크를 하였으며, Anomaly상황을 Anomaly로 리포트한 경우가 6개, Anomaly 상황이 아닌데

Anomaly로 리포트한 경우가 1개 발생하였다. 따라서 Anomaly 판단에 실패한 경우는 2가지 경우로 볼 수 있고, 따라서 판단 성공률은 87.5%를 보였다고 할 수 있으며, 이는 우리가 개발하면서 예상한 수치와 어느정도 맞는 결과라고 할 수 있다.

Anomaly 상황에서 Face Detection에 성공한 경우가 총 3회였으며, 이 중에서 각 4번씩 Face Recognition 검증 비디오를 통해 체크한 결과 8번의 경우 제대로 판단이 되었으나, 4번의 경우는 잘못된 판단을 내렸다.

그림3) 얼굴이 완전히 뭉게져있어 Face Detection이 불가능했던 경우

그림4) Face Detection이 성공하기는 했는데 얼굴 정보의 부족으로 Face Recognition에서 실패한 케이스

여기서 Anomaly 상황이 총 7번 일어났고, 각각 Face Detection을 수행했으나, Face Detection 자체가 되지 않은 경우가 무려 4번이나 되며 이는 무언가 심각한 문제가 있는 것으로 판단할 수 있다. 하지만, 해당 비디오의 경우 사람이 보아도 얼굴을 판단할 수 없을 정도로 이미지가 뭉개져 있는 경우가 2가지, 뒤통수만 보여주는 경우가

2가지로, 실제 해당 경우는 인간이 개입한다고 해도 얼굴을 확인할 수는 없는 경우로 판단할 때, 실제 약간이라도 얼굴이 드러날 경우 100%로 Face Detection을 할 수 있으나, CCTV 등의 경우에는 그렇게 얼굴이 60% 정도도 드러나지 않는 경우들이 있다는 것을 알 수 있고, 이러한 경우 등을 대비하기 위해서는 지속적으로 Face Tracking을 진행하여 얼굴을 선제적으로 확보해 두는 방법을 고려해 보아야 할 것이다.

그리고 Face Recognition에 실패한 4가지 경우에는, Anomaly Clip에서의 인물이 리포팅된 인물과 닮은 것은 확실한데, 동일 인물인지는 사람이 봐도 애매한 경우라 판단이 된다. 얼굴이 아닌 전체적인 몸의 형상 등으로 판단할 때는 좀 더 확실하게 말할 수 있지만, 얼굴만으로는 판단이 어려운 경우로 판단이 되며, 얼굴 뿐만이 아니라 여러가지 신체적인 특징 등을 이용할 수 있다면 유용성을 확실하게 더 높일 수 있을 것이라 판단이 된다.

10. Division & Assignment of Work

항목	담당자
논문 분석	이재필, 파티마, 박소정
Anomaly Detection 트레이닝	파티마
mtcnn, resnet 리서치	파티마
Feature Extractor 구현	이재필, 박소정
Face Detection, Face Recognition Module 개발	이재필, 박소정
전체 pipeline 구성	이재필
테스트 시스템 구성	이재필, 박소정

11. Conclusion

우리가 처음 이 프로젝트를 진행할 때, 과연 Anomaly라는 추상적인 상황이 AI에 의해서 인지될 수 있는 것인가 하는 의구심이 적지 않았다. 논문의 내용을 읽고, 해당 내용을 구현해보고, 실제 테스트를 하게 되면서 갖게 된 생각은, 100% 의지할 수 있는 수준의 AI를 당장 구현하는 것은 상당히 어려운 일이라고 판단이 되지만, 80% 이상의 정도에서 인간의 노력을 경감시켜주는 목적 정도로는 충분히 구현이 가능한 일이라고 판단이 되었다.

실질적으로 CCTV 관리 인력을 도와주는 시스템이 되기 위해서 문제가 발생한 상황의 인지 뿐 아니라, 인물에 대한 트래킹까지 된다면 정말 많은 도움이 될 것이라고 생각했기 때문에 조금은 무리지만, Anomaly Detection과 Face Recognition을 하나의 프로젝트에서 리서치, 구현하기로 했다.

비록 우리가 최종적으로 구현한 시스템이 상품으로서 작동할 정도의 완성도를 가지고 있다고는 감히 말하기 어렵다. 하지만, 이러한 시스템의 가능성에 대한 프로토타이핑으로써는 충분히 의미를 둘 수 있는 작업이었다고 감히 말할 수 있다고 생각한다.

비록 여러 AI Model을 도입하는 바람에 하나 하나의 모델에 대한 깊이에 있어서는 조금 부족한 부분이 있음은 사실이다. 하지만, 다른 측면으로 보면, 하나 하나를 보았을 때는 산업적인 의미가 약한 AI들을 하나의 완결된 시나리오에서 융합하여 실제 산업현장이나 우리 실생활에 의미가 있는 상품이 될 수 있음을 보여준 부분에서는 또 의미가 있지 않았나 생각한다.

한학기는 생각보다 길지 않았고, 각자에게도 할 일이 적지 않아 아쉽게도 우리가 계획, 개발, 프로토타이핑 했던 프로젝트가 실제 real world에서 사용되는 것까지 확인하지는 못했지만 앞으로 해당 분야의 뉴스에 조금은 더 귀를 기울이며 INNODEP에서 해당 시스템과 유사한 시스템을 출시했다는 소식을 기다리게 될 것 같다.

◆ [Appendix] User Manual

우리가 제공하는 Anomaly Detection을 사용하기 위해선 가장 간단한 방법은 SurveillanceManager class를 사용하는 것이다.

사용 방법은
detector=SurveillanceManager() #객체를 생성하고
detector.do(videoPathStr) #분석하고자 하는 비디오의 url을 입력해 주면 된다.

현재 우리는 Broadcasting관련 네트워크 함수를 작성한 것은 아니고, Broadcasting함수에서 CheckEmbedding을 바로 호출하였다.

해당 함수를 개조하여 네트워크 패킷등을 전송하는 것으로 개조하면 분산시스템 구성이 가능하다.