BlazeIt: optimizing declarative aggregation and limit queries for neural network-based video analytics

Kang, D., Bailis, P., & Zaharia, M. (2018)

1. Introduction

비디오 분석의 절차

1. 비디오에서 Object detection 수행

2. 그 결과를 데이터베이스에 저장

3. query를 사용하여 저장된 데이터를 검색

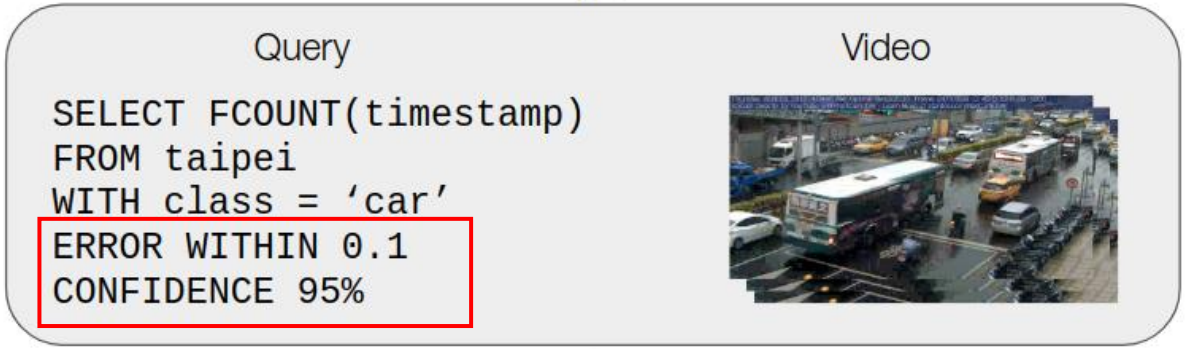
   → 이 작업을 빠르게 하는 것이 목표

비디오 분석 최적화

- FRAMEQL: video분석에 특화된 쿼리 최적화가 가능한 선언적 언어
- aggregation query (집계쿼리)에서는 대조변수 방법 (control variate)을 사용함
- limit query (제한 쿼리)에서는 신경망을 이용한 중요도 샘플링 기법으로 최적화
- Content-based selection에서 필터를 추론하는 방법을 보여줌

[그림 1] SQL syntex

[그림 2] FRAMEQL syntex



2. BLAZEIT system

BLAZEIT

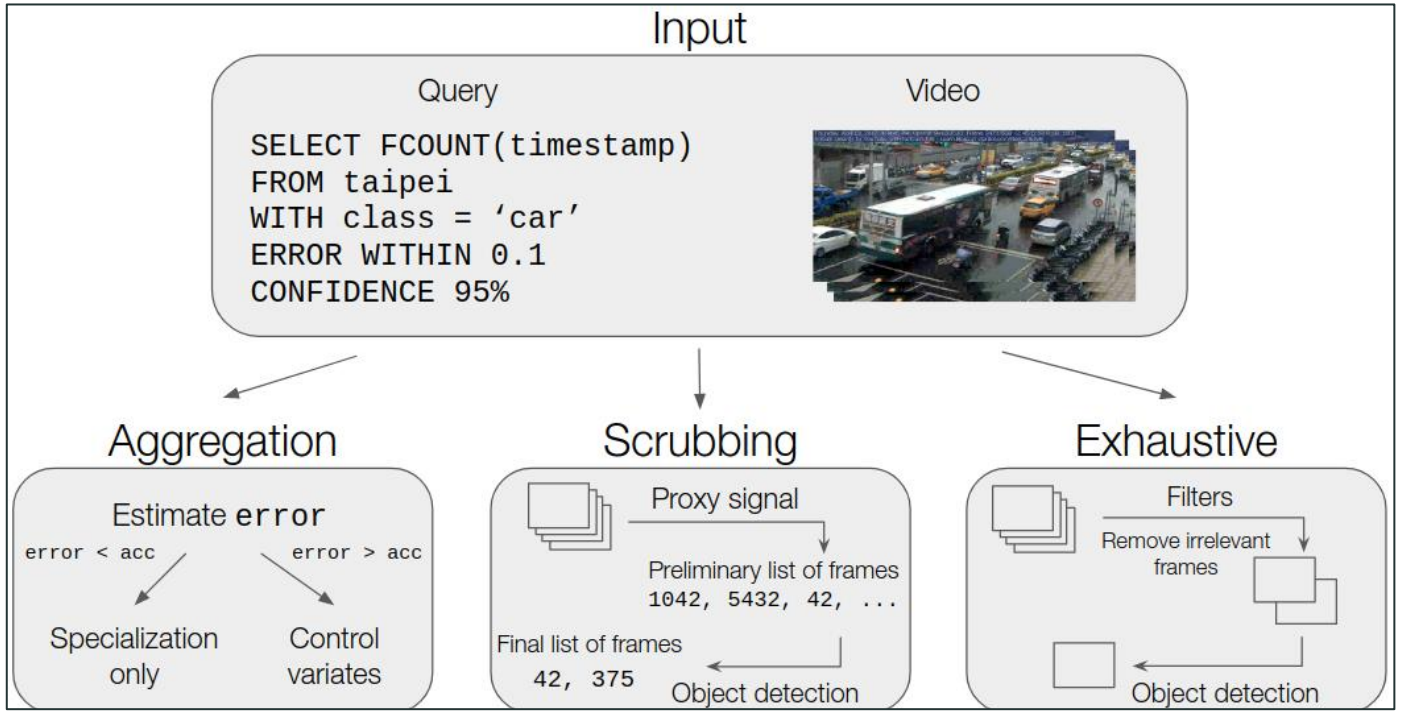
- 비디오 쿼리의 유용성 및 계산문제

  → FRAMEQL query를 가능한 한 빨리 실행하도록 함

- Aggregation and limit query에 새로운 최적화 방법 도입

  → aggregation query: 신경망 방법 도입,

  limit query: 검색법 도입



[그림 3] system overview

2. BLAZEIT system

2.1) 구성

Entity resolution

- **DB에서의 Entity resolution**

  : 데이터베이스에서 서로 다른 레코드들 간에 동일한 개체(entity)를 식별하고 결합하는 과정(중복제거, 일관성 유지 등).

- **video object detection에서의 Entity resolution**

  : 영상에서 object를 식별하고 object의 bounding box를 그리는 과정 (object의 정보 추출).

- 두 개의 인접한 frame과 box를 사용하고 box가 동일한 개체에 해당하는 경우 true.

Specialized neural network

- 특화된 신경망은 단순화된 작업에서 더 큰 신경망(예: Mask R-CNN)을 모방하는 신경망이다.
- 특화된 신경망을 proxy model로 사용하여 aggregation and limit query를 개선할 수 있다.
- BLAZEIT에서는 소형 ResNet을 사용한다.
- Binary detection뿐만 아니라 multi-class detection도 할 수 있도록 확장
- Aggretation and limit query의 정확성을 위해 특화된 신경망의 결과에 통계적 방법을 적용

Proxy model
: 실제 모델 대신 사용되는 작은 모델. 적은 파라미터를 가지며 적은 계산을 필요로 한다.
빠른 추론속도와 낮은 resource를 요구하므로 실시간 응용 프로그램에서 유용하다.

2. BLAZEIT system

Proxy model과 target-model annotated set (TMAS)

- 데이터 수집 시 신경망을 사용하여 video의 일부 프레임 샘플에서 객체를 감지하고 메타데이터 (추가정보)를 TMAS라고 하는 FRAMEQL의 튜플로 저장한다.

  (초기에 한번만 수행. 새로운 데이터가 들어오면 사용함으로 속도 향상)
- BLAZEIT는 query문을 기반으로 proxy 모델 (통계정보 사용) 및 필터 (조건을 만족하는 객체 감지)를 생성

  → proxy 모델 또는 필터의 오류율을 고려해야 한다 (객체감지의 정확도를 위함). 오류율은 hold-out data에서 계산
- Limit query에 대해 false positive (물체가 없는데 있다고 나타남)가 없거나 최소화되도록 보장할 수 있다.

Target-model annotated set
학습을 위해 사용되는 데이터 세트의 주석이 포함되어 있는 파일.

2. BLAZEIT system

2.2) 한계점

Proxy model과 target-model annotated set (TMAS)

- 시스템을 학습시키기 위해 필요한 학습데이터가 요구된다.
- 다른 현대 시스템에도 TMAS가 적용되어야 한다.

Object detection

- 미리 정의된 클래스 외에는 인식하지 않는다.

예를 들어, 학습시킨 자동차는 인식할 수 있지만 학습시키지 않은 세단과 SUV는 구별할 수 없다.

Model drift

- 배치분석설정 (일괄적으로 처리되는 데이터셋)에서만 TMAS, sampling이 가능
- 스트리밍 (실시간) 설정에서도 분석이 가능하나 model drift가 발생할 수 있음
  - 쿼리를 더 느리게 실행하지만 but, 정확도에 영향을 미치지 않음
  - Model drift를 완화하기 위해 새로운 데이터의 일부를 레이블링하고 drift를 모니터링하거나 지속적으로 재훈련해야 함

Scale out

- BLAZEIT의 프로토타입이 단일 서버에서 작동하도록 구현됨
- 데이터 처리 과정은 병렬처리가 가능하므로 여러 대의 서버에서 처리하도록 확장이 가능하다.
- 이러한 방식은 대규모데이터를 처리할 때 작업비용이 증가할 수 있다는 단점이 있다.

3. FRAMEQL: Expressing Complex Visual Queries

FRAMEQL

- BLAZEIT에서 적용하는 query 언어로 비디오 분석을 할 수 있도록 만들어진 SQL언어
- Aggregation and limit query를 포함한 새로운 query문
- 각 튜플은 프레임 내 1개 object에 해당한다.
- BLAZEIT는 Aggregation and limit query를 최적화하면서 정확도를 보장하고자 한다. 이때 object detection을 위한 신경망을 가능한 적게 호출한다.

적용 예시: 도시 계획 시나리오

1) 최소 1대의 버스와 최소 5대의 자동차 클립을 찾는다.

2) 빨간색 버스를 찾는다.

```
SELECT * | expression [, ...]
  FROM table_name
  [ WHERE condition  ]
  [ GROUP BY expression [, ...]  ]
  [ HAVING condition [, ...]  ]
  [ LIMIT count  ]
  [ GAP count  ]
  [ ERROR WITHIN tol AT CONFIDENCE conf  ]
```

Figure 4: FRAMEQL syntax.

[그림 4]

[그림 5]

3. FRAMEQL: Expressing Complex Visual Queries

SQL의 확장: 구문 추가

- **GAP, ERROR: 오차 허용을 지정**
- 사용자가 timestamp를 선택할 때 GAP 키워드는 반환된 프레임이 최소한 GAP 프레임 떨어져 있도록 한다.
- Ex) 사건이 포함된 10개의 연속 프레임이 있을 때, GAP = 100이면 10개 프레임 중 하나만 반환된다.
- 사용자는 질의에 신뢰수준과 함께 절대오차, false positive error, false negative error를 지정할 수 있다.

Syntactic element	Description
FCOUNT	Frame-averaged count (equivalent to time-averaged count), i.e., COUNT(*) / MAX(timestamp)
ERROR WITHIN	Absolute error tolerance
FPR WITHIN	Allowed false positive rate
FNR WITHIN	Allowed false negative rate
CONFIDENCE	Confidence level
GAP	Minimum distance between returned frames

Table 2: Additional syntactic elements in FRAMEQL. Some of these were adapted from BlinkDB [5].

- **FCOUNT: 프레임 평균값 계산**

첫 번째 비디오: 10,000개 프레임 중 모든 프레임에서 차(car)가 있음 → FCOUNT=1

두 번째 비디오: 10개 프레임 중 1개의 프레임만 차(car)가 있음 → FCOUNT=0.1

학생

학번	char(18)
이름	char(18)
주소	char(18)
학년	char(18)

Field	Type	Description
timestamp	float	Time stamp
class	string	Object class (e.g., bus, car)
mask	(float, float)*	Polygon containing the object of interest, typically a rectangle
trackid	int	Unique identifier for a continuous time segment when the object is visible
content	float*	Content of pixels in mask
features	float*	The feature vector output by the object detection method.

```
SELECT FCOUNT(*)
FROM taipei
WHERE class = 'car'
ERROR WITHIN 0.1
AT CONFIDENCE 95%
```

프레임 당 평균 차량 수를 계산

```
SELECT timestamp
FROM taipei
GROUP BY timestamp
HAVING SUM(class='bus')>=1
   AND SUM(class='car')>=5
LIMIT 10 GAP 300
```

적어도 하나의 버스와 다섯 대의 차량이 있는 프레임을 선택

4. Query Optimizer

Aggregation query와 limit query의 최적화

- **규칙 기반 옵티마이저**
  - aggregation 키워드 (FCOUNT)가 포함된 경우 aggregation에 대한 최적화
  - limit 키워드가 포함된 경우 limit query에 대한 최적화
  - 그 외의 query의 경우 NOSCOPE와 비슷한 필터를 적용
- **장점**
  - 필터 및 특화된 신경망을 사용하므로 object detection 비용을 줄임
  - 두 최적화 모두 쿼리 실행 속도만 높이고 쿼리 정확도에는 영향을 미치지 않는다. (다음 장에서 계속)
  - 최적화 외에도 특화된 신경망모형의 가중치와 그 결과를 저장하여 작업을 재사용할 수 있다.

평균 정확도 초당프레임

Method	mAP	FPS
YOLOv2 [63]	25.4	80
Mask R-CNN [32]	45.2	3
Specialized NN	N/A	35k
Decoding low-resol video	N/A	62k
Color filter	N/A	100k

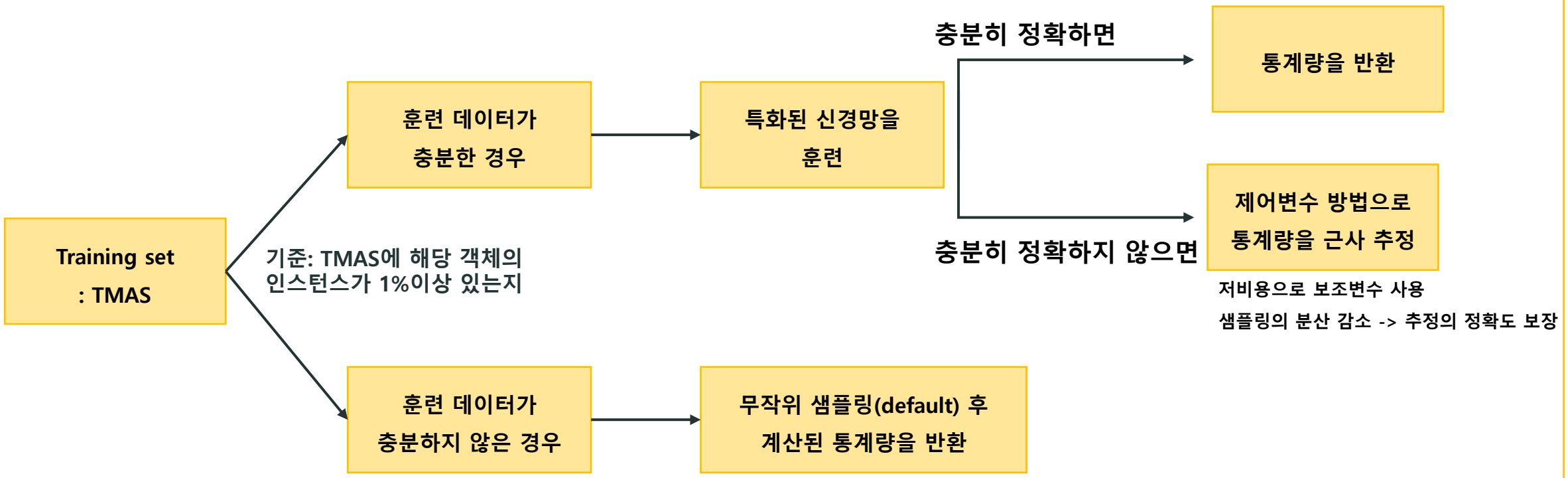
Table 3: A comparison of object detection methods, filters, and speeds. More accurate object detection methods are more expensive. Specialized NNs and simple filters are orders of magnitude more efficient than object detection methods.

4. Query Optimizer

4.1) Optimizing Aggregates

```
SELECT FCOUNT(*)
FROM taipei
WHERE class = 'car'
ERROR WITHIN 0.1
AT CONFIDENCE 95%
```

- 데이터의 일부 통계량 (프레임당 차량의 평균 수)와 같은 것들로, 프레임 안의 객체의 수를 빠르게 세는 것이 목적
- 모든 프레임에서 object detection을 하는 것이 가장 정확하지만, but 너무 느리기 때문에 error bound를 지정함으로써 더 빠르게 실행함
  (사용자가 엄격한 오차 허용 한도를 입력한 경우, 신경망의 정확도가 떨어질 수 있음)



4. Query Optimizer

Data: TMAS, unseen video, 훈련데이터 (TMAS)와 unseen video를 준비
$uerr \leftarrow$ user's requested error rate, 사용자가 error rate와 confidence level을 지정함
$conf \leftarrow$ user's confidence level
Result: Estimate of requested quantity 마지막 결과는 통계량 (개수)을 반환
if *training data has instances of object* **then**
  train specialized NN on TMAS;
  $err \leftarrow$ specialized NN error rate;
  $\tau \leftarrow$ average of specialized NN over unseen video;
  if $P(err < uerr) < conf$ **then**
    return τ;
  else
    $\hat{m} \leftarrow$ result of Equation 2 (control variates);
    return $\hat{m}$;
  end
else
  Return result of random sampling.; 훈련 데이터의 인스턴스에 객체가 없으면 무작위샘플링 후 계산된 통계량를 반환
end
Algorithm 1: BLAZEIT's Aggregation Query Procedure

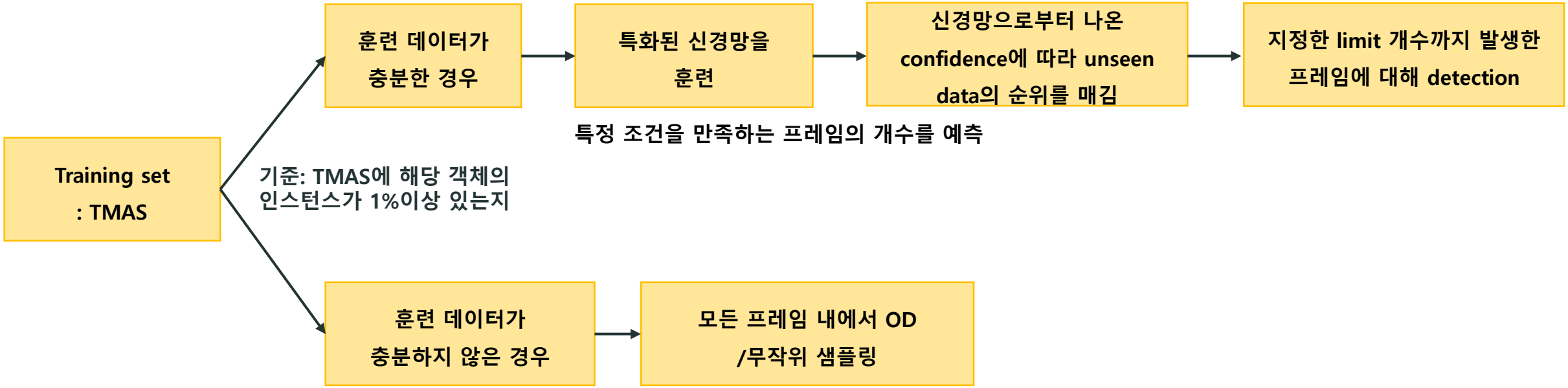
훈련 데이터의 인스턴스에 객체가 있으면 신경망으로 학습시킴
신경망으로부터 unseen data의 통계량을 계산
1) 만약 신경망에서의 성능이 충분히 정확하면
통계량 답을 반환

2) 신경망의 성능이 충분히 정확하지 않으면
대조변수법으로 계산된 통계량을 반환

4. Query Optimizer

4.2) Optimizing limit

- (찾고자 하는 객체가 드물게 발생한다면) 객체를 모든 프레임에서 찾는 것이 아니라 객체가 존재할 가능성이 높은 프레임을 선택
- query에서 GAP 키워드를 사용하는 경우, 한 번 프레임을 찾은 후에는 주변의 GAP 프레임을 무시하여 처리 비용을 더욱 줄일 수 있음



예시)
적어도 1개의 버스와 5개의 자동차가 들어간 프레임 찾기
→ 버스와 자동차의 개수를 각각 예측하는 신경망을 훈련시킴
→ 위 조건을 만족하는 프레임의 확률의 합을 signal로 간주함
→ 가장 confidence가 높은 순부터 지정한 limit 개수의 프레임을 사용함

5. Implementation

Video ingestion

- OpenCV를 사용해 video를 불러온다
- 프레임을 각 신경망에 대해 적절한 크기로 변형한다 (신경망은 65x65, OD에서는 짧은 너비가 600 pixel이 되도록).
- Pixel을 적절하게 정규화한다.

Specialized NN training

- Pytorch v1.0을 사용하여 신경망을 학습한다.
- Video를 업로드하고 65x65로 크기를 변형, ImageNet정규화한다.
- 배치사이즈를 16으로 하여 cross–entropy loss로 훈련시킨다.
- 신경망 학습은 Momentum 0.9로 힌 SGD로 최적화한다. 특화된 신경망은 'tiny ResNet'을 사용한다.

Identifying object across frames

- Trackid 를 계산하는 데 기본값은 motion IOU를 사용한다.
- 연속된 두 프레임에서 각 객체들의 IOU를 계산한다. 두 객체가 겹치는 면적을 두 객체 전체 면적의 합으로 나눈 것으로 0과 1 사이의 값을 가진다.
- 0.7이상의 IOU값을 가진 객체들은 같은 객체로 간주하고 동일한 trackid를 부여한다.

6. Evaluation

- 사전학습된 신경망(pretrained NNs)을 사용하여 추가 데이터 수집 및 학습 과정 없이 object detection을 수행
  - 레이블링: MASK R-CNN, FGFA, YOLOv2 중 각 비디오에 대해 가장 정확한 방법을 수동으로 선택
- 현재 최고 수준의 detection 방법도 작은 객체에 대한 성능이 여전히 낮다. 따라서 프레임 크기에 비해 객체가 큰 영역만 고려하였다.

각 비디오 및 객체 클래스에 대해 객체가 존재하는 것으로 간주할 신뢰도 임계값을 수동으로 선택

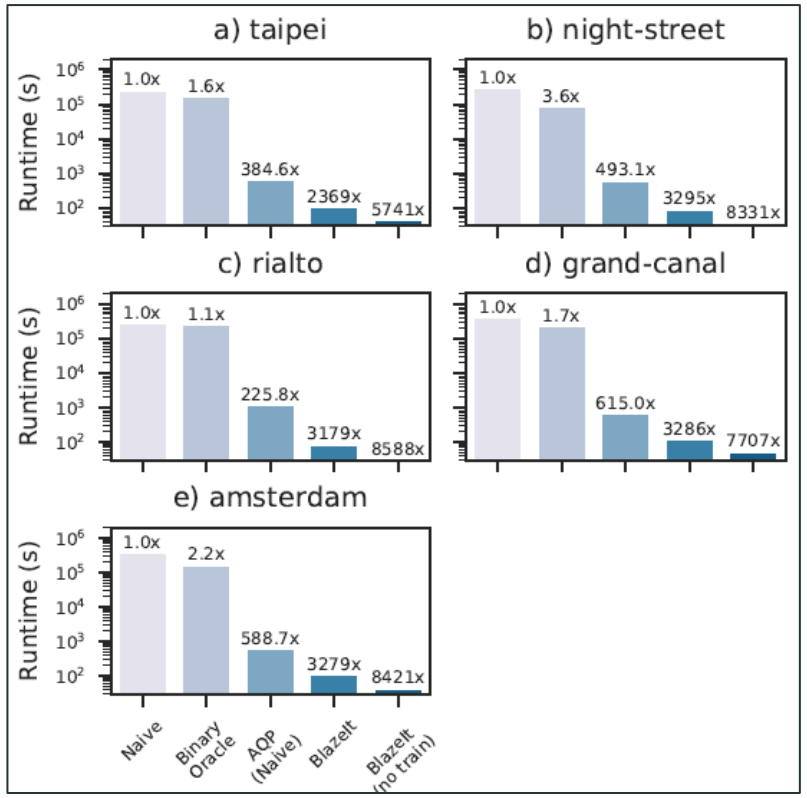
Video Name	Object	Occupancy	Avg. duration of object in scene	Distinct count	Resol.	FPS	# Eval frames	Length (hrs)	Detection method	Thresh
taipei	bus	11.9%	2.82s	1749	720p	30	1188k	33	FGFA	0.2
	car	64.4%	1.43s	32367						
night-street	car	28.1%	3.94s	3191	720p	30	973k	27	Mask	0.8
rialto	boat	89.9%	10.7s	5969	720p	30	866k	24	Mask	0.8
grand-canal	boat	57.7%	9.50s	1849	1080p	60	1300k	18	Mask	0.8
amsterdam	car	44.7%	7.88s	3096	720p	30	1188k	33	Mask	0.8
archie	car	51.8%	0.30s	90088	2160p	30	1188k	33	Mask	0.8

Table 4: Video streams and object labels queried in our evaluation. We show the data from the test set, as the data from the test set will influence the runtime of the baselines and BLAZEIT.

6. Evaluation

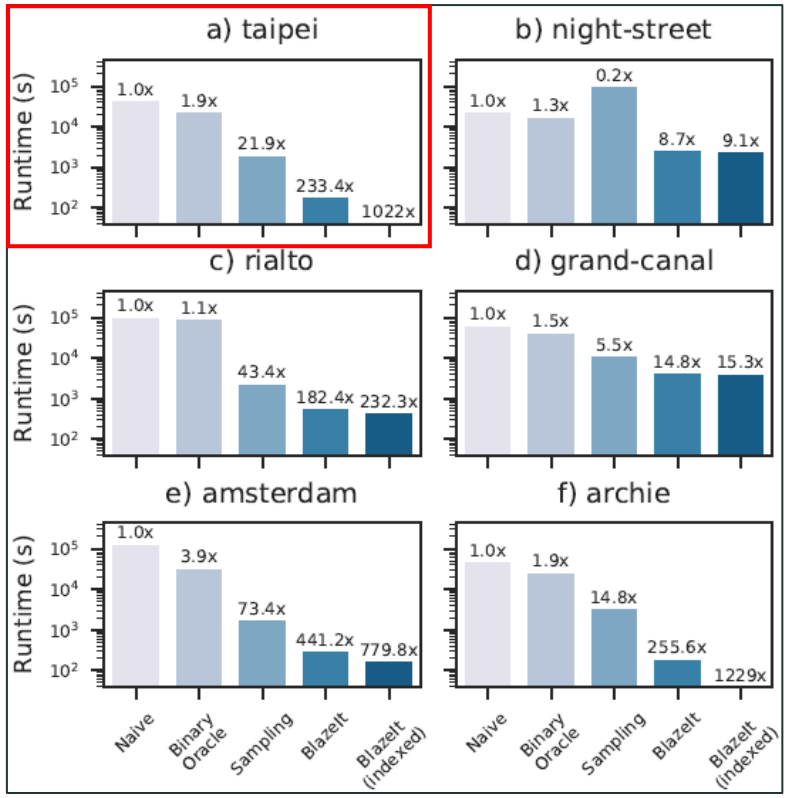
```
SELECT FCOUNT(*)                SELECT timestamp
FROM taipei                     FROM taipei
WHERE class = 'car'             GROUP BY timestamp
ERROR WITHIN 0.1                HAVING SUM(class='bus')>=1
AT CONFIDENCE 95%                  AND SUM(class='car')>=5
                                LIMIT 10 GAP 300
```

Aggregate query



- AQP에 비해 최대 14배 속도향상
- Oracle은 관심개체 수가 많을 때 성능이 좋지 않음

limit query



- Baseline model들에 비해 1000배 이상의 속도향상

데이터셋의 인스턴스 수와 object 갯수

Video name	Object	Number	Instances
taipei	car	6	70
night-street	car	5	29
rialto	boat	7	51
grand-canal	boat	5	23
amsterdam	car	4	86
archie	car	4	102

Table 7: Query details and number of instances. We selected rare events with at least 10 instances.

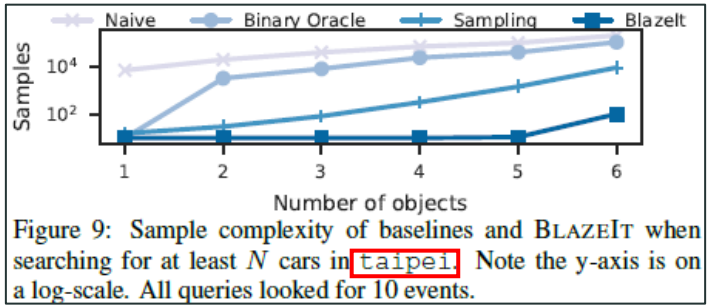


Figure 9: Sample complexity of baselines and BLAZEIT when searching for at least N cars in taipei. Note the y-axis is on a log-scale. All queries looked for 10 events.

Taipei 데이터의 샘플복잡도

Miris: Fast object track queries in video

BASTANI, Favyen, et al. (2020)

1. Introduction

객체 추적 쿼리

- object tracking: 같은 object 인스턴스에 해당하는 object detection의 sequence
- query는 시간적으로 겹쳐진 객체를 포함하는 join과 복잡한 조건문으로 이루어짐
- 전체 비디오에서 OD 및 tracking하는 query를 실행하는 것은 많은 컴퓨팅 시간이 필요하다. 일부 프레임만 사용하는 방식이 있었지만 tranking문제는 모든 프레임을 사용해야 할 것이기 때문에 적용하기 적절하지 않다.

[그림 1] object tracking

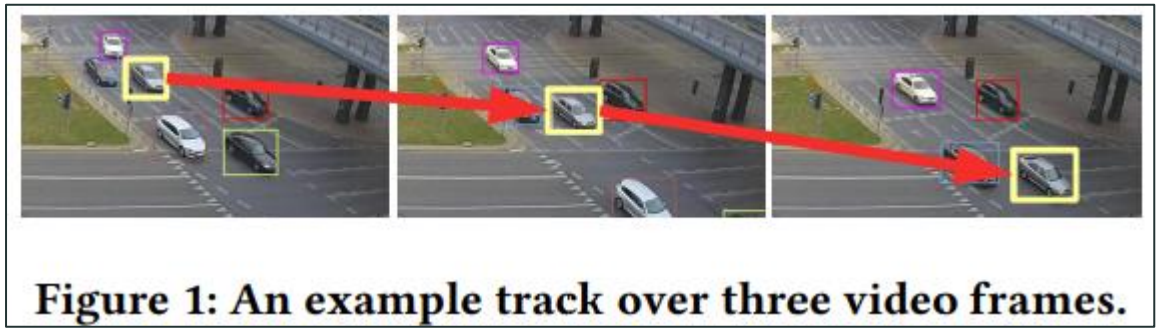


Figure 1: An example track over three video frames.

Query-driven tracking

- Query processing + object detection
- 쿼리 실행 시 객체추적을 줄인 주파수를 적용하여 object detection 작업 부하를 최소화함

[그림 2] object tracking query의 예시
자동차가 자전거 선수를 고속으로 지나치고 가까운 거리에서 지나치는 상황

```
SELECT car, cyclist FROM (PROCESS inputVideo
      PRODUCE car, cyclist USING objDetector)
WHERE Speed(car) > 30 km/h   자동차의 빠르기
AND Angle(car, cyclist) < 10 deg   자동차와 자전거 선수가 같은 방향으로 가는 상태
AND MinDistance(car, cyclist) < 1 m   1미터 내에 있어야 함
```

1. Introduction

1.1) Queries

문제

- 자동차가 자전거 선수를 고속으로 지나치고 가까운 거리에서 지나치는 상황

```
SELECT car, cyclist FROM (PROCESS inputVideo
      PRODUCE car, cyclist USING objDetector)
WHERE Speed(car) > 30 km/h
AND Angle(car, cyclist) < 10 deg
AND MinDistance(car, cyclist) < 1 m
```

쿼리 실행

Predicates P

- 조건1: 자동차의 빠르기
- 조건2: 자동차와 자전거 선수가 같은 방향으로 가는 상태
- 조건3: 1미터 내에 있어야 함

True/False

True/False

True/False

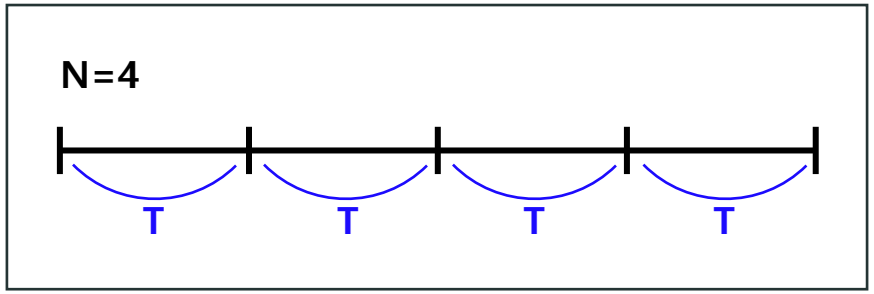
join

T/T/T인 Tuple 선택

1. Introduction

1.2) preprocessing

1. 비디오를 랜덤하게 N개 segment로 나눔
2. 각 segment에 대해 object detection (ex. YOLO), tracking algorithm (ex. IOU) 적용
3. 50:50으로 훈련데이터, 검증데이터로 나눔



- N이 크거나 T가 크면 처리 시간이 오래걸림 & 더 정확한 결과
- N, T간 trade-off를 잘 결정해야 한다. 논문의 경우 N=24, T=5가 효율적이었다고 함.

전처리의 효과

- 정확한 track을 수집하여 training/validation set으로 사용하기 위함
- 최적의 query-driven tracking parameter를 얻기 위함

1.3) Query processing

- 사용자가 설정한 정확도를 만족하면서 query의 실행속도 최소화를 위한 실행계획 (뒤에서 다시 언급)

2. Query-driven tracking

Sampling framerate

- 초당 프레임 수

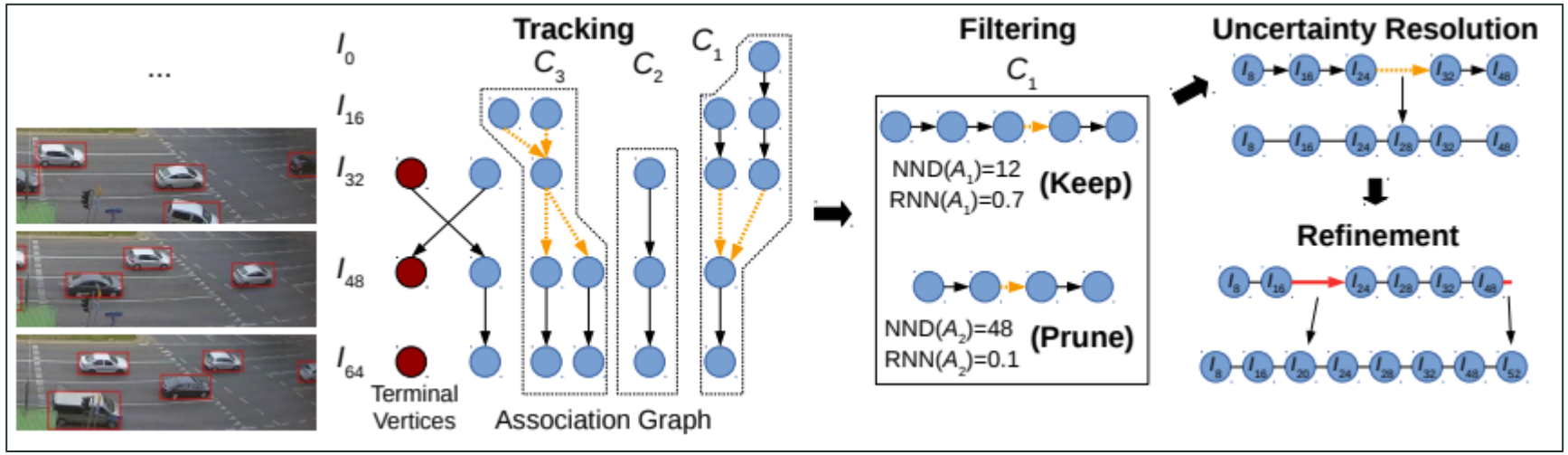
Segment:
tracking의 정확도 good
조건문 판단 good

Other segment

low-framerate

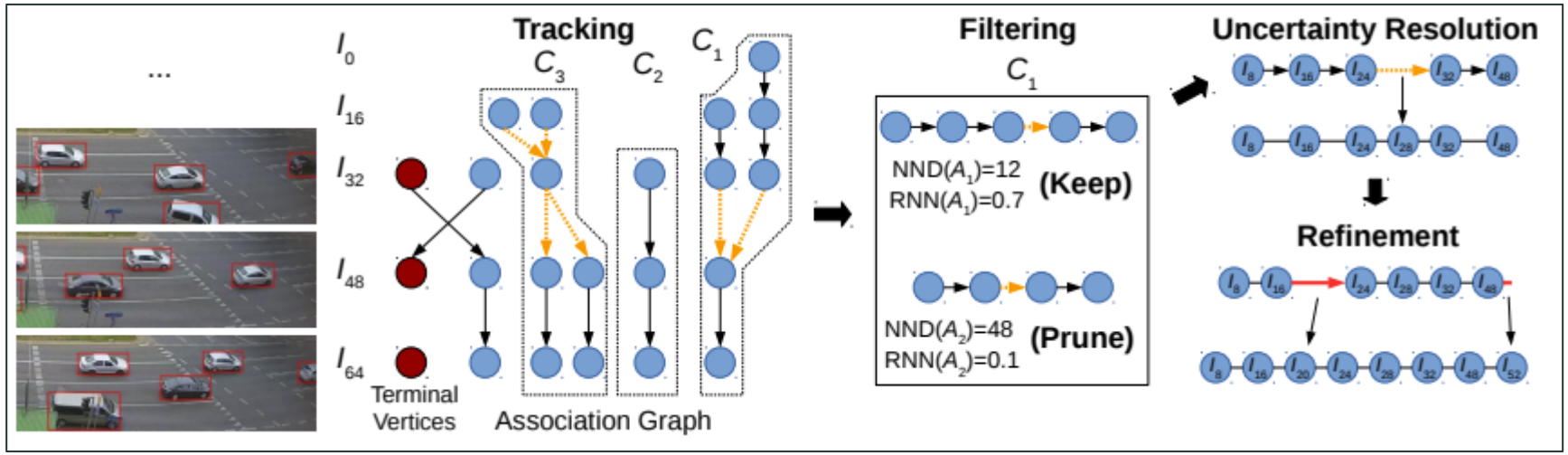
비디오 프레임을 추가하여 처리

[그림 3] four-stage query-driven tracking approach



2. Query-driven tracking

[그림 3] four-stage query-driven tracking approach



사용할 프레임 선택

- Minimum sampling framerate (사용자가 지정한 최소로 필요한 framerate)로 추적

추적모형

- GNN
(감지된 객체들의 시공간 특징을 반영하여 학습)
- Output
감지된 물체가 같은 물체일 확률

매칭 결정이 불확실

- 모든 매칭을 결정 후보로 하는 그룹 C의 트랙 생성
- 그림에서는 C_2만 확실하게 결정된 트랙
- C_1, C_3 은 일부만 확실하게 결정된 트랙
- orange line은 불확실한 엣지

pruning

$Filter(C)$

- 그룹 C에서 조건을 만족하는 트랙만 포함시킴
- 비어있는 경우 tracking으로 돌아감

결정적 트랙 선택

- $Filter(C)$에 남아있는 불확실한 매칭을 해결하기 위한 절차

$resolve(C) \subset Filter(C)$

- $Filter(C)$에서 결정적인 트랙을 Output으로 반환함

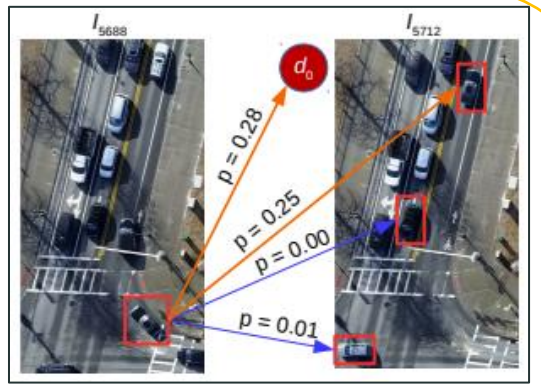
세부 조정

- $resolve(C)$ 내 각 트랙이 조건을 만족하는지 평가
- 프레임 간격이 다소 떨어져 있으므로 중간 프레임으로 추가 탐지하여 $resolve(C)$ 을 확장
- 확장된 트랙에서 조건을 만족하는 트랙을 output으로 반환

2. Query-driven tracking

2.1) Tracking

- 두 비디오 프레임 사이의 동일한 물체인지 판단하는 과정
- 가정: 두 프레임에서 감지된 두 객체는 동일한 객체일 수 있다.
- 객체가 카메라 밖에 위치하면 edge로 연결되지 않을 수 있음 → 모형에 terminal node: d_0^k, d_0^{k+f} 를 포함
- edge 선택방법: IOU 대신 CNN+GNN 방법을 사용 (전처리한 데이터를 학습시킴)

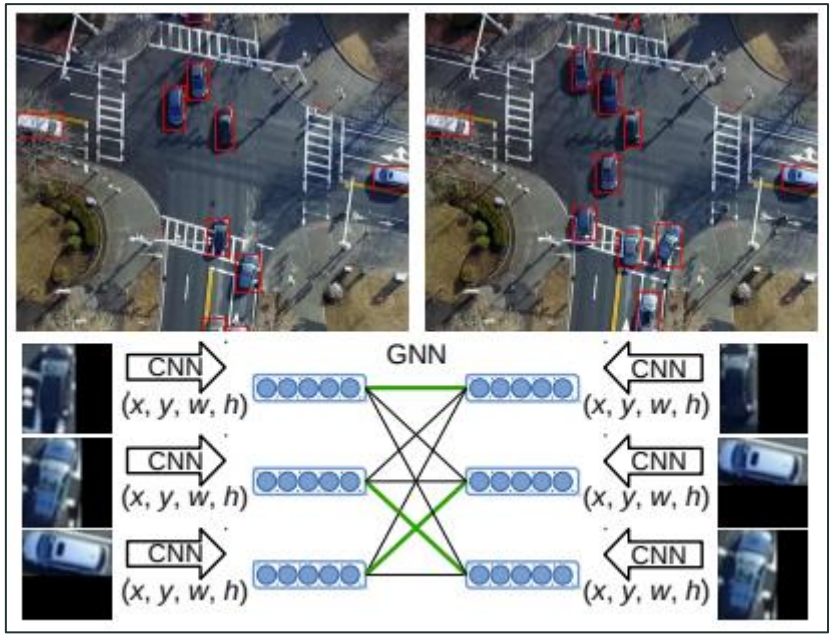


I_k I_{k+f}: I_k에서 f만큼 떨어진 프레임

$D^k = \{d_1^k, \cdots, d_n^k\}$

: sets of detections in frame I_k

each detection is a bounding box $d_i^k = \{x_i^k, y_i^k, w_i^k, h_i^k\}$



$D^{k+f} = \{d_1^{k+f}, \cdots, d_m^{k+f}\}$

: sets of detections in frame I_{k+f}

each detection is a bounding box

$d_i^{k+f} = \{x_i^{k+f}, y_i^{k+f}, w_i^{k+f}, h_i^{k+f}\}$

CNN

- Input: I_k에서 Bbox와 일치하는 영역
- Output: vector of visual feature f_i^k

GNN

- Input: f_i^k and $(x_i^k, y_i^k, w_i^k, h_i^k)$
- Output: $p_{i,j}^k$ (양쪽 edge (d_i^k, d_j^{k+f}) 가 올바르게 매칭될 확률)

[그림 4] low-framerate로 샘플링된 두 연속적 프레임 사이의 매칭 과정

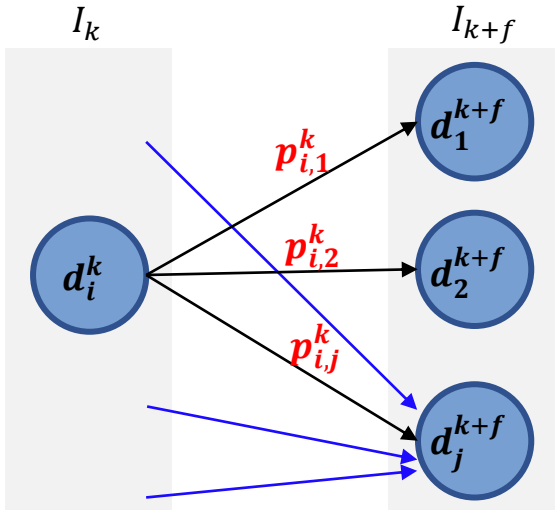
2. Query-driven tracking

2.1) Tracking

object tracking과 query processing의 통합

- 모든 프레임을 사용하지 않고 sampling frame을 사용했기 때문에 추적오류가 발생 (잘못된 매칭 연결)
- 해결: I_k와 I_{k+f} 사이에 중간 프레임 $I_{k+f/2}$을 추가하여 framerate를 높이는 방법을 사용
  → 모형의 신뢰도와 정확도가 높아질 것

- 노드 d_i^k에 대한 최대매칭확률: $MMP(d_i^k) = \min_j p_{i,j}^k$
- 주어진 임계값 Q_f에 대해 $p_{i,j}^k > Q_f MMP(d_i^k)$ 이면 (d_i^k, d_j^{k+f}) 를 연결
  - GNN을 통해 계산된 가장 높은 확률을 가진 edge
    + 가장 높은 확률에 Q_f배 이내의 출력 확률을 가진 모든 엣지가 연결됨
  - Nondeterministic (불확실한 매칭)



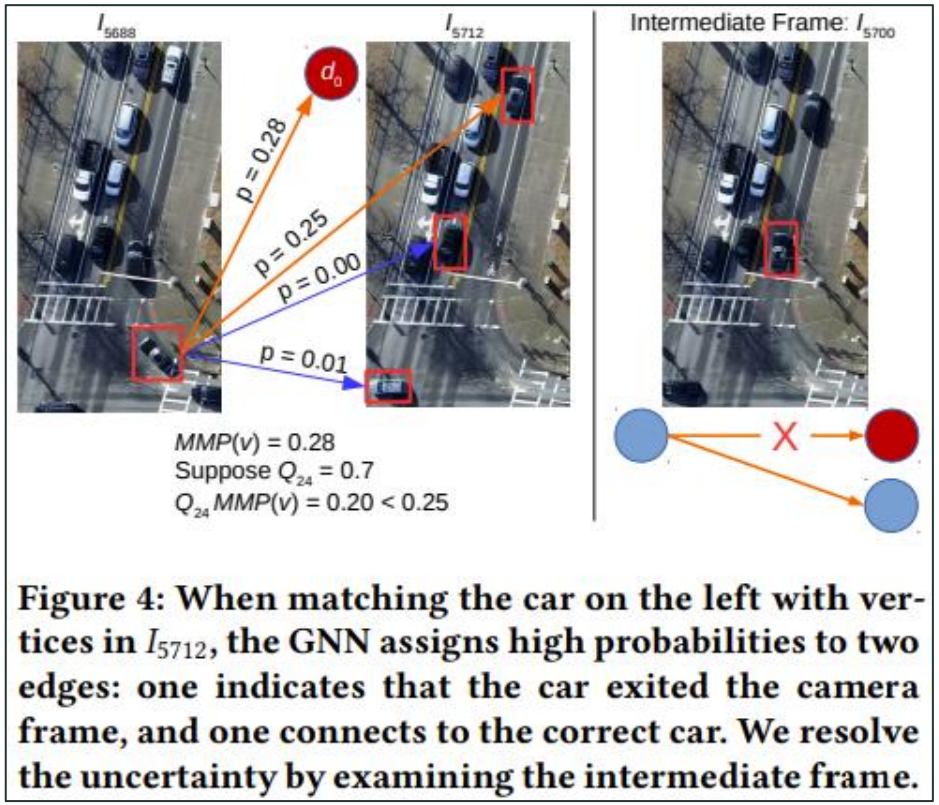


Figure 4: When matching the car on the left with vertices in I_{5712}, the GNN assigns high probabilities to two edges: one indicates that the car exited the camera frame, and one connects to the correct car. We resolve the uncertainty by examining the intermediate frame.

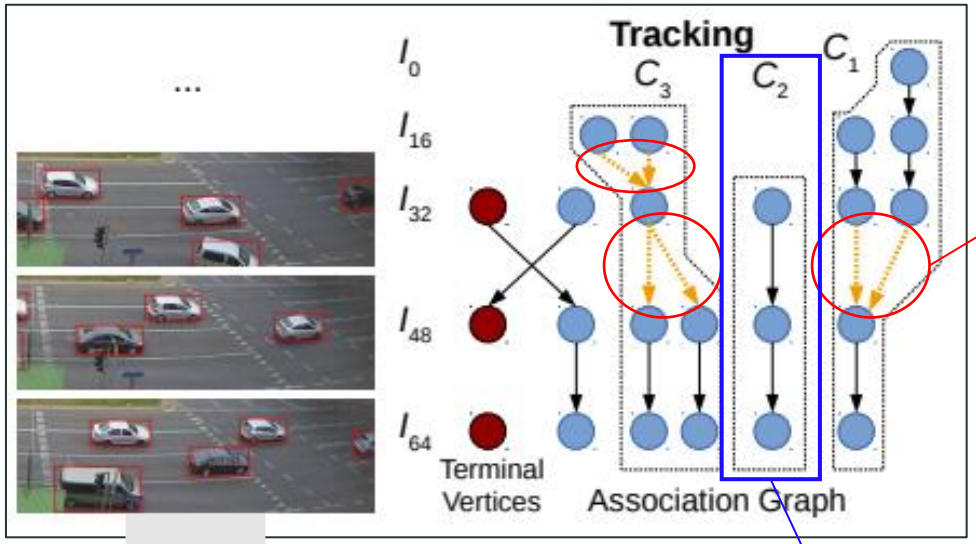
[그림 5] 중간프레임을 추가해 두 노드를 높은 확률로 정확하게 연결함

2. Query-driven tracking

2.1) Tracking

후보트랙 $C_1, C_2, C_3, \ldots$ 에 대하여

- 노드 d_1^k, d_0^k(terminal node)와 edge (d_i^k, d_j^{k+f}) 를 가짐
- 중간 비디오 프레임을 처리하면 해당 트랙이 올바른지 알 수 있음
- 다음 단계: 조건문을 충족시키지 못할 후보 트랙을 제거



후보트랙

완성된 연결이 있는 C (결정된 트랙)는 다음 단계로 넘어가지 않음

프레임을 추가할수록 늘어남

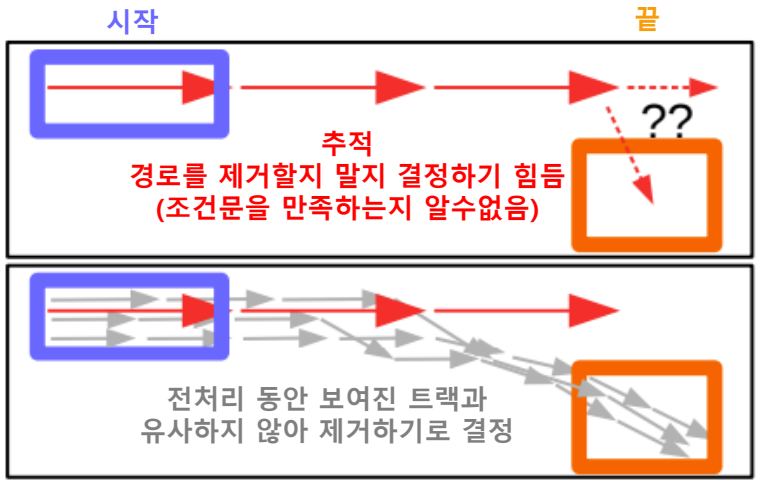
2. Query-driven tracking

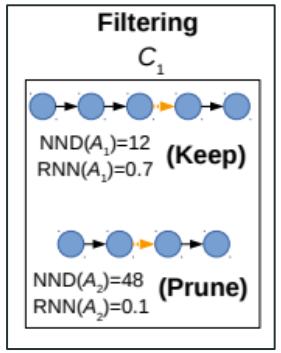
2.2) Filtering

- 전처리된 트랙의 정보를 이용해 추적의 제거여부를 결정

1) 추적 A 를 가진 후보트랙 C
2) Filtering method (거리): $\{F_1, \dots, F_n\}$
3) threshold: $\{T_1, \dots, T_n\}$ } Best 선택
4) $F_i(A) < T_i$이면 후보트랙을 제거

결과: $filter(C)$, $(F_1(A) > T_1) \wedge \cdots \wedge (F_n(A) > T_n)$를 만족하는 트랙을 원소로 가짐





[그림 6] Filtering이 어려운 이유

- 후보트랙에서 filtering 할 트랙 $A =< a_1, \dots, a_n >$와 (예측할 트랙)
- 전처리 과정에서 생성된 트랙 $B =< b_1, \dots, b_n >$
  (모든 프레임에서 감지한 트랙 정보를 담고있음)

Nearest Neighbor Distance Filter

1) a_1과 가장 가까운 detection b_{i1} 찾은 후 유클리디안 거리 d_1 계산
2) a_2과 가장 가까운 detection $b_{i2}, i2 > i1$ 찾은 후 유클리디안 거리 d_2 계산
…
3) $d_1, \dots, d_n$ 의 평균 $D(A,B)$ 를 계산

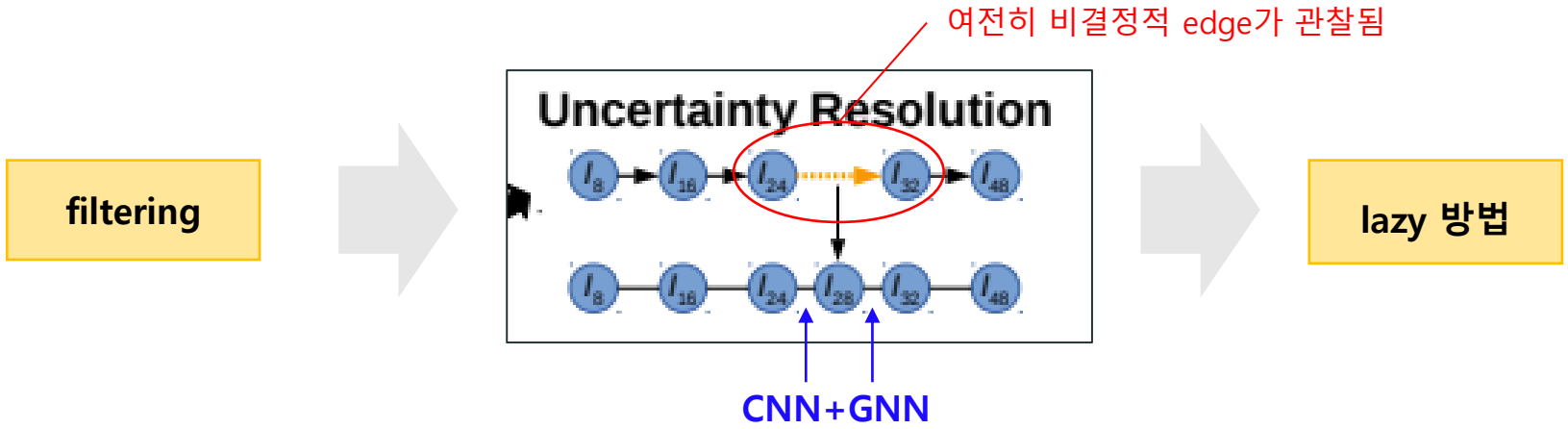
결과: $\mathrm{NND}(A) = \min_{B \in \mathrm{P}(\mathrm{S}_{\mathrm{train}})} D(A,B)$ 이 작을수록 B 는 조건을 만족하는 집합

RNN filter

1. 추적정보를 가진 B로 모형을 학습시킴:
   input: $coarsify(B)$, $B \in \mathrm{S}_{\mathrm{train}}$에서 무작위로 pruning된 detection들, label : $\mathrm{P}(B)$
2. 학습된 모형으로 후보트랙 A이 조건을 만족할 확률 예측

2. Query-driven tracking

2.3) Uncertainty Resolution



- 여전히 비결정적 edge가 존재할 수 있음

  ex) 노드 v에 대해 edge가 연결될 확률 $Q_f MMP(v)$, $Q_f < 1$ 을 초과하는 노드들이 여러 개 일 때
- 중간프레임을 추가 → GNN + CNN → $Q_{f/2} = 1$로 두어 해결할 수 있음

- 앞서 필터링 조건을 만족하는 후보 추적그룹 C를 대상으로 하여 (비용 절감) $Filter(C)$를 $resolve(C)$로 처리하는 과정 (정확한 매칭 결정)

2. Query-driven tracking

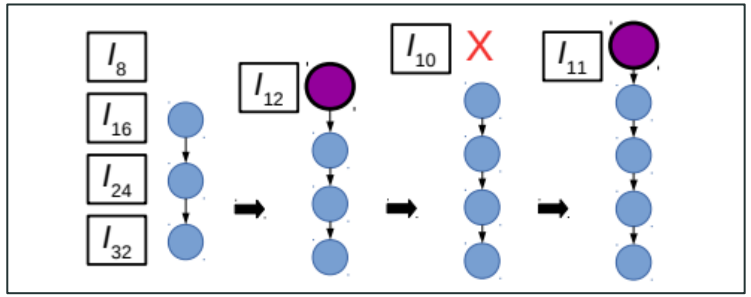
2.4) Refinement

초기 추적경로에 대해 추가적인 감지를 찾아내야 하는 경우

- 초기 추적결과에서 첫 번째 감지 이전이나 마지막 감지 이후에 누락된 부분이 있어 경로가 불완전한 경우

추가 detection

- coarse track이전과 이후의 두 방향의 비디오 프레임으로 추적을 연장한다.
- 최소 샘플링 빈도(f_{min}) parameter를 쿼리 계획 단계에서 설정하여 이진 검색을 중단하는 수준을 제한한다.



[그림 7] prefix-suffix 방법의 이진검색과정

- 추적경로와 연관된 첫 번째와 마지막 감지값을 식별한다.
- 검색 범위를 절반씩 줄여나가면서 목표값을 찾음

초기 추적경로가 더욱 세밀한 감지를 필요로 하는 경우

- 초기 추적경로 감지에 선형보간 방법이 사용되어 보간오차 발생 우려
- 보간 오차를 줄이기 위해 추가적인 비디오 프레임을 조사해야 한다.

오차 줄이기

- 연속된 세 개의 detection (d_{i-1}, d_i, d_{i+1})마다 가속도 $accel(i) = (d_{i+1} - d_i) - (d_i - d_{i-1})$을 계산한다.
- 가속도가 임계값 T_{accel}을 초과하는 경우 d_{i-1}와 d_i, d_i와 d_{i+1} 사이에 추가 object를 검출하기 위해 추가적인 비디오 프레임을 조사한다.
- 그러면 연속된 detection 쌍 중 최대 가속도가 T_{accel}보다 작아진다.

3. Planning

Query Planner

query

- 객체 추적을 위한 minimum sampling framerate
- Filtering method의 조합
- 불확실한 matching output graph를 언제 고려해야 하는지
- Q_f 값의 설정
- Refinement 방법의 조합

- 목적: 사용자 지정 정확도를 보장하면서 object detection에 사용할 frame수를 최소화

4. datasets

- **the Berkeley DeepDrive dataset**

자동차 카메라로 찍은 1100시간 비디오

- **Tokyo, Warsaw traffic dataset**

고정된 카메라로 찍은 각 60시간 비디오

- **UAV hovering dataset**

헬리콥터로 찍은 도로분기점. 2시간 비디오

- **Resort hotel dataset**

리조트에서 고정된 카메라로 찍은 보행자도로 비디오. 60시간

- BDD를 제외한 데이터에서는 object 의 bounding box를 수동으로 지정, YOLOYv3을 여러 개 학습하였음
- UAV, Tokyo and Warsaw에서는 자동차를, BDD와 Resort에서는 보행자를 30%의 임계값으로 detection

5. Results

비디오 초당 GPU 초 당 처리량 (높을수록 빠름)

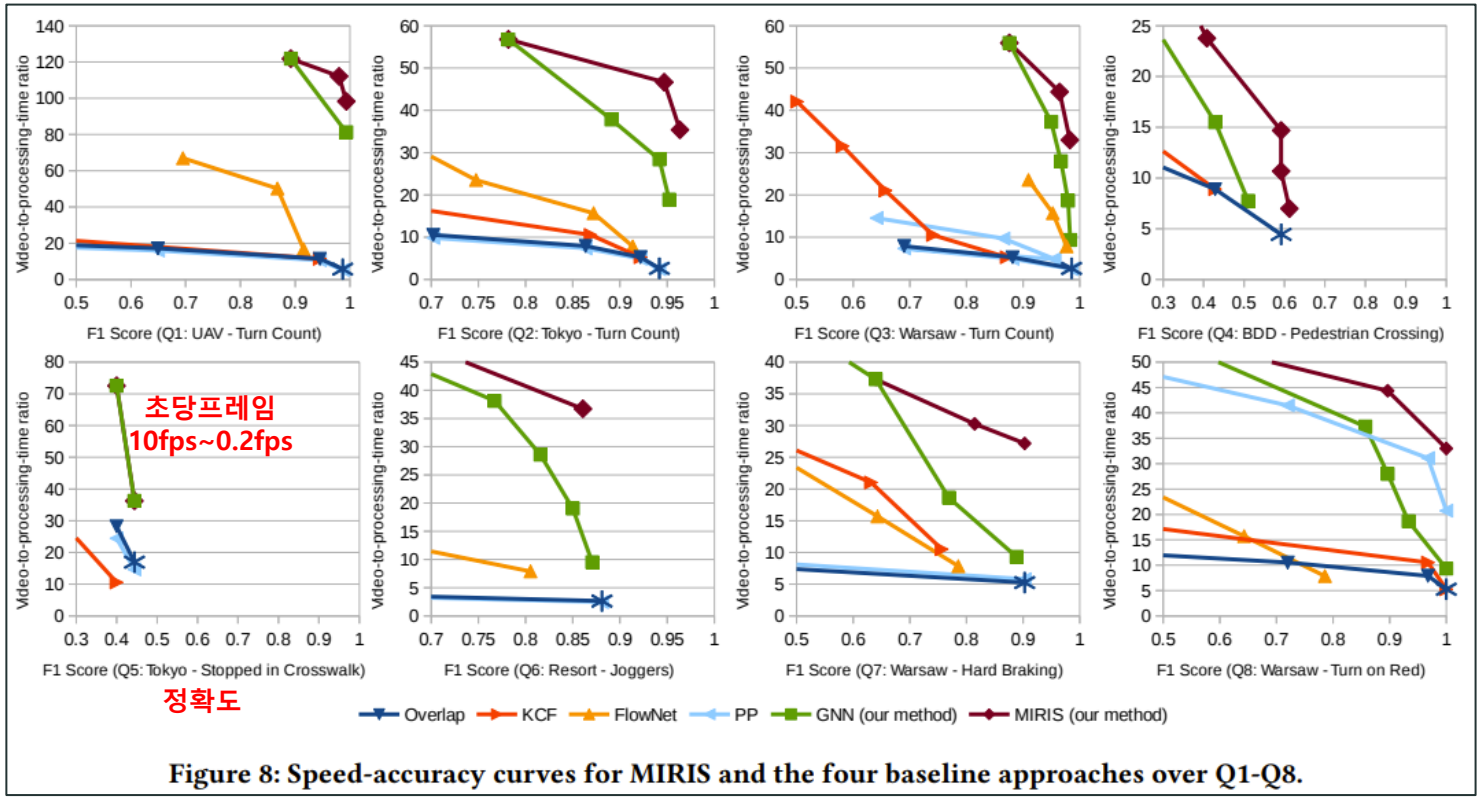


Figure 8: Speed-accuracy curves for MIRIS and the four baseline approaches over Q1-Q8.

- Q1: UAV data에서 교차로에서 특정한 방향으로 움직이는 차량의 수를 식별하는 쿼리
- Q2: Tokyo data에서 교차로에서 특정한 방향으로 움직이는 차량의 수를 식별하는 쿼리
- Q3: Warsaw data에서 교차로에서 특정한 방향으로 움직이는 차량의 수를 식별하는 쿼리
- Q4: BDD data에서 카메라가 장착된 차량 앞에서 한쪽에서 다른 쪽으로 직접 건너는 보행자들의 트랙을 식별하는 쿼리
- Q5: Tokyo data에서 교차보도 중앙에서 차량이 정차하는 경우를 파악하는 쿼리 (15초 이상 정차한 차량을 선택)
- Q6: Resort data에서 조깅하는 사람들을 식별하는 쿼리 (평균속도가 15pixel/sec 이상인 보행자)
- Q7: Warsaw data에서 교차로에서 급정거하는 경우를 식별하는 쿼리
- Q8: Warsaw에서 빨간 신호등에서 우회전을 하는 차량을 파악하는 쿼리