PT

1. Introduction
2. Related Work
3. Transformer Tracking
4. Experiments

발표는 Transformer Tracking 논문의 구성대로 진행하겠습니다.

Introduction

- Transformer Tracking
  - 2021년 CVPR에 Accept된 논문
  - Visual Tracking 에 Transformer를 접목한 TransT라는 모델을 제안한 논문
  - **TransT:** Transformer Tracking Model

Transformer Tracking

Xin Chen[1] ; Bin Yan[1] ; Jiawen Zhu[1], Dong Wang[1] † Xiaoyun Yang[3] and Huchuan Lu[1,2]
[1]School of Information and Communication Engineering, Dalian University of Technology, China
[2]Peng Cheng Laboratory [3]Remark AI
{chenxin3131, yan_bin, jiawen}@mail.dlut.edu.cn
wdice@dlut.edu.cn, xyang@remarkholdings.com, lhchuan@dlut.edu.cn

Transformer Tracking은 2021년 CVPR에 Accept된 논문이며, Visual Tracking 분야에 Transformer를 접목해서 TransT라는 모델을 제안하였습니다.

Introduction

- Visual Tracking
  - 비디오 영상에서 시간에 따라 움직이는 어떤 객체를 찾는 과정
  - 첫 번째 frame에서 어떤 객체를 추적할지 정하고 해당 객체만 계속해서 추적한다.

- **Given Arbitary Target**
  어떤 물체를 추적해야 되는지 정해져 있지 않다.
- **Localize Target in Video**
  추적되는 객체는 비디오에서 Localization된다.
- **Class Agnostic**
  클래스와 전혀 관계가 없다.
- **Hard Negative**
  Negative를 Positive라고 잘못 예측하기 쉽다.
- **Object Deformation**
  추적하려는 객체는 항상 같은 모습이 아니다.

Siam Mask Object Segmentation Tracking Demo

DELAB 4

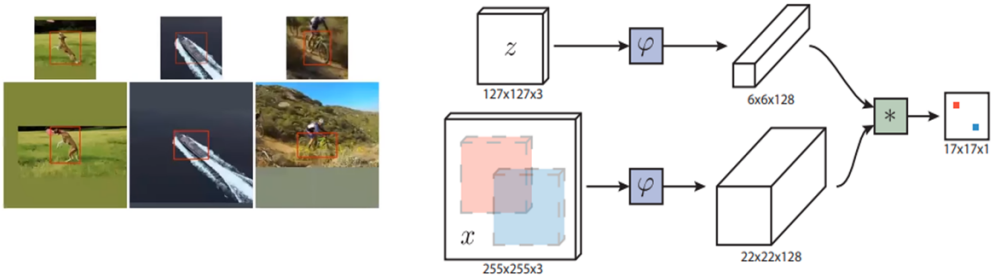
따라서 Visual Tracking부터 소개해드리겠습니다. Visual Tracking은 비디오 영상에서 시간에 따라 움직이는 어떤 객체를 찾는 과정입니다. 영상을 보시면 이해가 쉬울텐데, 이렇게 첫 번째 frame에서 어떤 객체를 추적할지 정하면 해당 객체만 계속 추적하게 됩니다. 따라서 단일 객체 추적 문제라고 생각할 수 있습니다. 이러한 Visual Tracking은 크게 다섯가지 특징이 있습니다.

첫 번째로 어떤 물체를 추적해야 되는지 정해져 있지 않습니다. 사용자가 지정하는 객체가 무엇이든 추적을 해야 됩니다.
두 번째로 추적되는 객체는 비디오에서 계속 바운딩 박스가 쳐진 상태여야 합니다.
세 번째로 클래스와 전혀 무관한 문제입니다. 클래스에 대한 학습 데이터가 필요가 없습니다.
네 번째로 Hard Negative합니다. 영상에서 노란색 옷을 입은 다른 운동 선수는 Negative지만, positive인 사람도 노란 색 옷을 입었으므로 Positive라고 잘못 예측하기 쉽습니다.
마지막으로 객체가 변형되는 경우에도 잘 파악해야 됩니다. 추적하는 대상이 항상 같은 모습이 아니기 때문입니다.

이러한 Tracking 문제를 푸는 방법은 여러 가지가 있습니다. 그 중에서 현재에 가장 많이 쓰이는 기법인 샴 네트워크 방식을 소개해드리겠습니다.

Related Work

- Siamese-based method
  - Siam Network: 두 개의 이미지를 각각 같은 구조의 네트워크에 입력하여 벡터화 시킨 이후, 두 벡터간의 유사도를 반환하는 모델
  - Siam FC: 샴 네트워크로 객체 추적을 시도한 첫 모델
  - z: Template, x: Search Region, *: correlation



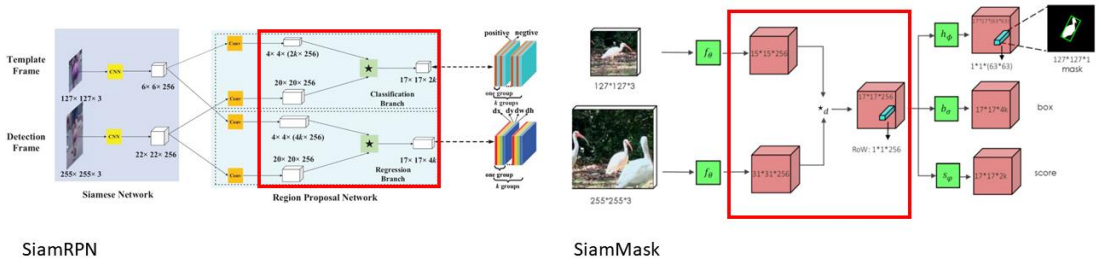
DELAB 5

샴 네트워크는 두 개의 이미지를 각각 같은 구조의 네트워크에 입력하여 벡터화시킨 이후, 두 벡터간의 유사도를 반환하는 모델입니다. 이 때, 샴 네트워크의 weight는 공유됩니다.
그리고 Siam FC는 이 샴 네트워크를 이용해서 객체 추적을 시도한 첫 모델입니다.
모델의 입력으로 사용되는 z는 Template으로 첫 번째 frame에서 추적해야 될 대상이 담긴 이미지입니다. x는 Search Region으로 n번째 frame에서 추적해야 될 대상을 찾으려는 이미지입니다.
두 이미지는 샴 네트워크에 입력돼서 각각의 feature map이 출력됩니다. 이후 correlation연산을 통해 두 이미지의 공통인 특징을 찾게 됩니다. 여기서 correaltion은 z의 feature map이 필터로 사용돼서 x의 feature map과 convolution한다고 생각하면 됩니다.
그 결과는 17x17x1 feature map이 되는데, 만약 공통인 객체가 x이미지의 빨간 영역에 존재하면 최종 feature map에서도 빨간 점 부분이 유사도가 높습니다. 그렇지 않고 파란 영역에 존재하면 파란 점 부분이 유사도가 높습니다.
결국 이 최종 feature map을 upsampling해서 유사도가 높은 점 주위로 바운딩 박스를 찾아내는 개념입니다.

이러한 샴 네트워크 기반 객체 추적은 학습 시간은 오래 걸리지만 온라인 학습(인퍼런스 중 학습)을 하지 않으므로 fps가 다른 기법 대비 굉장히 높습니다.
또한 성능도 준수해서 다른 업그레이드된 모델들이 많이 발표되었습니다.

Related Work

- Siamese-based method
  - correlation 연산은 global context를 충분히 활용하지 못 함 -> Local Optimum
  - correlation 연산 시 semantic information이 손실 됨 -> Target의 boundary에 대한 부정확한 예측
  - High-Accuracy Tracking Alogrithms를 설계한 것의 병목 현상이 됨



그런데 샴 네트워크에서 유사도를 구할 때 사용하는 correlaton은 여러 가지 문제가 있다고 합니다.

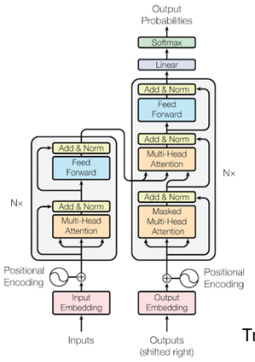
첫 번째로 correlation 연산 자체가 local하게 특징을 파악하는 방식이므로 global context를 충분히 활용하지 못해서 결국 local optimum에 빠진다는 것입니다.

두 번째로 correlation으로 연산을 하면 의미있는 정보가 손실이 돼서 Target의 boundary를 예측할 때 부정확할 수 있다는 것입니다.

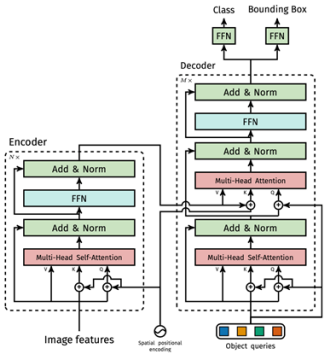
따라서 SiamFC 이후에 SiamRPN이나 SiamMask 등 업그레이드된 모델이 나왔어도 correaltion을 사용한다는 점에서 병목 현상이 발생된다고 논문에서는 말합니다.

Related Work

- Transformer
  - Transformer는 Attention 기반 Encoder와 Decoder로 Machine Translation 문제를 해결한 첫 모델
  - 최근에는 Vision 영역까지 Transformer가 사용됨 -> DEtection TransformeR
  - DETR의 성공과 Detection과 Tracking 사이의 밀접한 관계에 자극을 받아 Transformer를 Tracking Field에 도입



Transformer (translation)



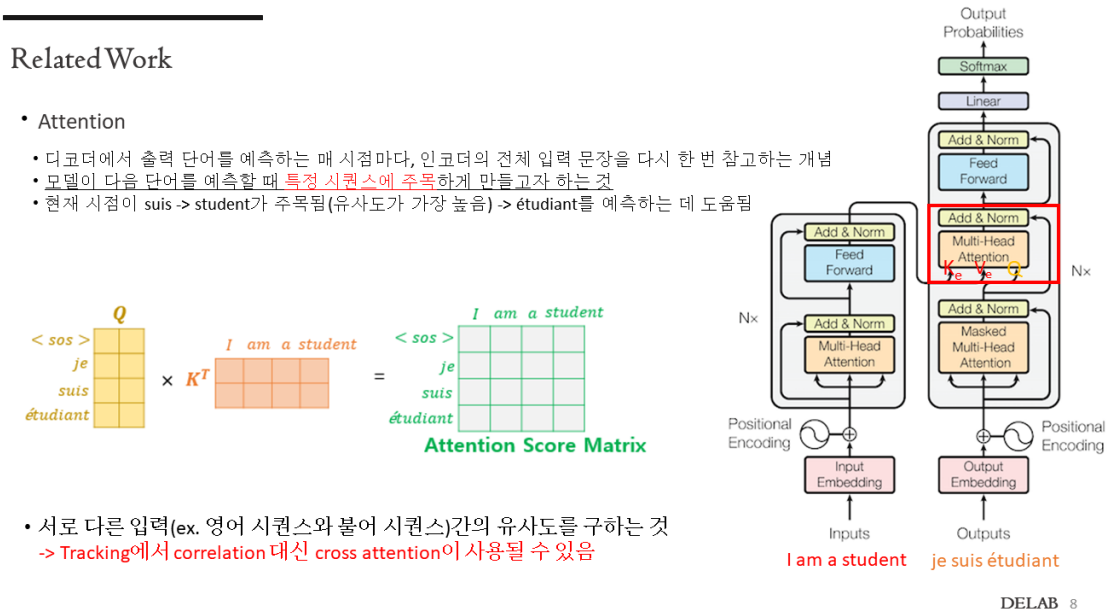
Transformer (Detection)

두 번째 Related Work는 Transformer에 대한 소개입니다. Transformer는 Attention 기반 Encoder와 Decoder로 기계 번역 문제를 해결한 첫 모델입니다.

현재는 NLP Task 뿐 만 아니라 Vision 영역까지 사용되는 모델이며 대표적으로 Object Detection 의 DETR 모델이 있습니다.

이 논문 저자는 DETR의 성공과 Detection과 Tracking 사이의 밀접한 관계에 자극을 받아 Transformer를 Tracking Field에 도입했다고 합니다.

또한 Transformer Decoder의 Attention에 많은 영감을 받았다고 합니다.

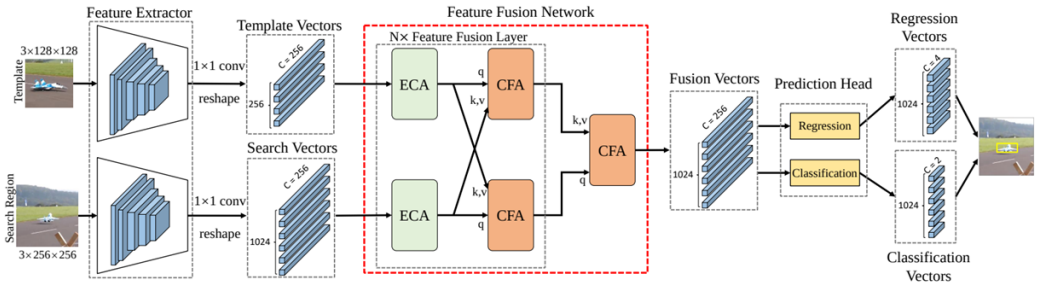


Attention을 소개해드리기 위해 NLP Task로 예시를 보이겠습니다. Attention은 디코더에서 출력 단어를 예측하는 매 시점마다, 인코더의 전체 입력 문장을 다시 한 번 참고하는 개념입니다. 이 때, 단순히 참고만 하고 넘어가는 것이 아니라 입력 문장의 특정 시퀀스에 주목을 하겠다는 것입니다. 예를 들어 'I am a student'를 'je suis etudiant'로 번역을 한다면 Transformer Decoder는 순차적으로 'je', 'suis', 'etudiant'를 예측합니다. 이 때 현재 시점이 'suis'라면 Decoder의 output은 'etudiant'가 나와야 됩니다. 이렇게 다음 단어를 예측하는 순간에 'je suis'에 대한 특징뿐만 아니라 'I am a student'문장에 대한 특징을 이용하고, 이 중에서 특히 'student'시퀀스의 특징을 주목한다면 'etudiant'를 구하는데 도움이 될 것이라는 게 Attention의 주된 아이디어입니다.
좀 더 자세히 말씀드리기 위해 박스친 Attention 모듈에서 계산하는 방식을 가져왔습니다.
Query는 Decoder 입력 문장인 불어에 대한 특징, Key와 Value는 Encoder 입력 문장인 영어에 대한 특징이 담깁니다. 현재 시점이 'suis'이므로 Query와 Attention Score Matrix의 네 번째 행(etudiant에 대한 행)은 무시하겠습니다. 이 때, Attention Score Matrix는 Query와 Key의 내적 계산을 통해 이루어지고 결과적으로 각 시퀀스간의 유사도가 저장됩니다. 이 때 앞 예시에 따르면, Matrix의 3행 4열에 해당하는 student값이 가장 유사도가 높았다는 것입니다. 그리고 이 Attention Score Matrix가 이용되면서 다음 단어인 'etudiant'를 예측하는 데 도움이 되었다는 것입니다.

그런데 물론 NLP의 Attention을 소개해드리려고 이렇게 말씀드린 것은 아닙니다.
Tracking에서 Attention의 핵심은 서로 다른 입력간의 유사도를 구할 수 있다는 것입니다.
NLP Task에서는 서로 다른 시퀀스 간의 유사도를 구하겠지만 비전 영역에서는 서로 다른 이미지 패치간의 유사도를 구하는 것이 될 것입니다.
따라서 correlation대신 이러한 cross attention이 사용될 수 있습니다!

Transformer Tracking

- Overall Architecture
  - **Feature Extractor:** ResNet50을 이용해서 Template과 Search Region의 Feature를 추출
  - **Feature Fusion Network:** Template과 Search Region간의 유사도를 구해 동일한 특징을 파악
  - **Prediction Head Network:** 1024개의 후보 박스에 대해 class score와 coordinates를 결정



이제 본격적으로 모델의 전체적인 구조를 말씀드리겠습니다.

모델은 크게 세 파트로 구분이 됩니다.

첫 번째는 Feature Extractor로, ResNet50을 이용해서 Template과 Search Region의 특징을 각각 추출하게 됩니다.

두 번째는 Feature Fusion Network로, Template과 Feature간의 유사도를 구해 동일한 특징을 파악하는 것입니다. 이 때 '유사도를 구해 동일한 특징을 파악'하는 것을 논문에서는 Fusion한다고 표현하였습니다.

세 번째는 Object Detection의 RPN 개념처럼 후보 박스를 설정하는 것입니다. 이 때 모든 후보 박스는 객체가 존재 여부가 담긴 class score와 좌표 정보인 coordinates를 가져야 합니다.

지금부터는 각각의 파트에 대해 더 자세하게 말씀드리겠습니다.

Transformer Tracking

- Input
  - Template: 비디오 sequence의 첫 번째 frame이 기준, 대상의 중심 좌표에서 측면 길이의 2배로 확장됨
    -> 정사각형 형태로 바꾼 다음 backbone의 입력으로 사용됨

128x128x3

- Search Region: 이전 frame을 기준, 대상의 중심 좌표에서 측면 길이의 4배로 확장됨
  -> 정사각형 형태로 바꾼 다음 backbone의 입력으로 사용됨

256x256x3

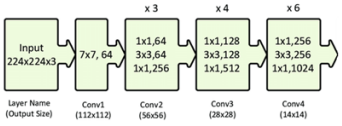
DELAB 10

먼저 모델의 첫 입력으로 사용되는 template과 search region을 조정하는 것부터 시작이 됩니다. template은 비디오 sequence의 첫 번째 frame이 기준이며, 대상의 중심 좌표에서 측면 길이의 2배로 확장시킨 이미지라고 합니다. 이후 정사각형 형태로 바뀌어 결국 128x128 이미지가 입력으로 사용됩니다.

search region은 이전 frame이 기준이며, 대상의 중심 좌표에서 측면 길이의 4배로 확장시킨 이미지라고 합니다 마찬가지로 정사각형 형태로 바뀌며 256x256 크기가 됩니다.

Transformer Tracking

- Modified Version of ResNet50
  - 마지막 스테이지(layer4) 제거
  - layer3의 Conv2D를 Dilation Convolution(2)으로 변경 -> Receptive Field를 증가
  - layer3의 Conv2D를 stride를 2에서 1로 변경 -> Feature Resolution을 증가




```
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

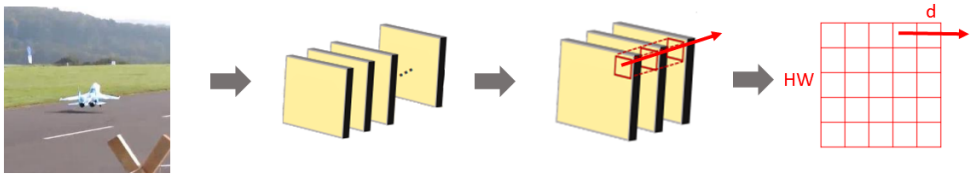
DELAB 11

이후 Feature를 추출하는 Backbone은 ResNet50을 그대로 이용하지 않고 살짝 변형을 시켰다고 합니다. ResNet50은 layer0부터 layer4까지 총 5개의 Residual Block으로 이루어져 있지만, 마지막

스테이지인 layer4는 사용하지 않았다고 합니다. 또한 layer3의 convolution 연산을 dilation convolution으로 변경해서 Receptive Field의 크기를 증가시켰고, downsampling 파트에서 conv2d를 stride 2에서 1로 변경함으로써 Feature Resolution을 증가시켰습니다.

Transformer Tracking

- Feature Extractor
  - Modified ResNet을 통해 Feature Map을 얻음
  - 1x1 Conv를 통해 미리 설정한 임베딩 차원(d=256)으로 축소
  - Flatten하여 HWxd 크기의 행렬을 얻음 -> Transformer의 입력으로 사용



Feature Extraction
(H_0xW_0xC_0) -> (HxWxC)
[H = H_0 / 8, W = W_0 / 8, C = 1024]

1x1 Conv
(HxWxC) -> (HxWxd)
[d = 256]

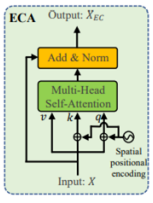
Flatten
(HxWxd) -> HWxd
[Template: 256x256]
[Search Region: 1024x256]

DELAB 12

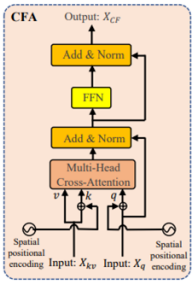
이렇게 수정된 Backbone을 이용하면 Template와 Search Region의 크기는 모두 1/8배가 됩니다. 이후 1x1 conv를 해서 차원 축소를 해서 채널 수를 256으로 만들고, Flatten해서 HWxd 크기의 2차원 벡터로 변형시킵니다. 이 행렬이 트랜스포머의 입력으로 사용됩니다.

Transformer Tracking

- Feature Fusion Network
  - ECA는 Transformer Encoder를, CFA는 Transformer Decoder를 변형시킨 구조
  - **ECA(eco-context argument):** feature representation을 강화하기 위해 multi-head self-attention 사용
  - **CFA(cross-feature argument):** cross-attention을 통해 두 Feature map을 Fusion



$$\mathbf{X}_{EC} = \mathbf{X} + \mathrm{MultiHead}(\mathbf{X} + \mathbf{P}_x, \mathbf{X} + \mathbf{P}_x, \mathbf{X}),$$



$$\mathbf{X}_{CF} = \widetilde{\mathbf{X}}_{CF} + \mathrm{FFN}\left(\widetilde{\mathbf{X}}_{CF}\right),$$
$$\widetilde{\mathbf{X}}_{CF} = \mathbf{X}_q + \mathrm{MultiHead}\left(\mathbf{X}_q + \mathbf{P}_q, \mathbf{X}_{kv} + \mathbf{P}_{kv}, \mathbf{X}_{kv}\right),$$

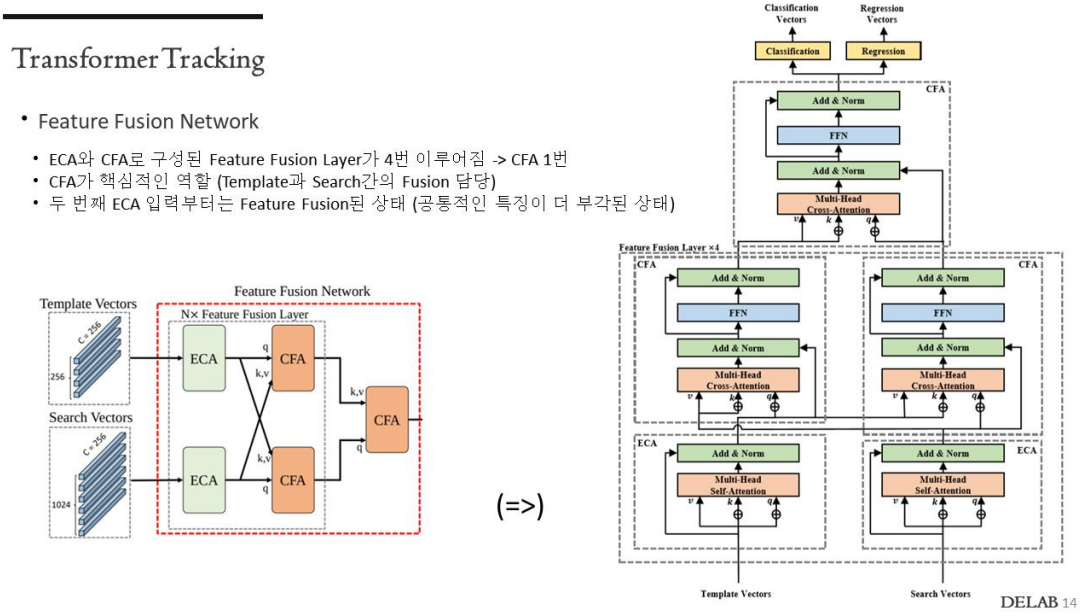
DELAB 13

다음은 Feature Fusion Network에 대한 설명입니다. Feature Fusion부터는 트랜스포머 구조가 사용

이 되지만 원본 그대로를 사용하지는 않습니다. ECA는 Transformer Encoder를 변형시킨 구조로, 기존 Encoder에서 FFN이 사라졌습니다. CGA는 Transformer Decoder를 변형시킨 구조로, 사이드 입력이 사라지고 메인 입력이 두 개로 늘어났습니다. 또한 번역처럼 다음 시퀀스 정보를 감출 필요가 없으므로 Masked Attention이 사라졌습니다. 그리고 식을 보시면 DETR 때처럼 Query와 Key 이미지에 대해서만 Positional Encoding을 하는 것을 확인할 수 있습니다.

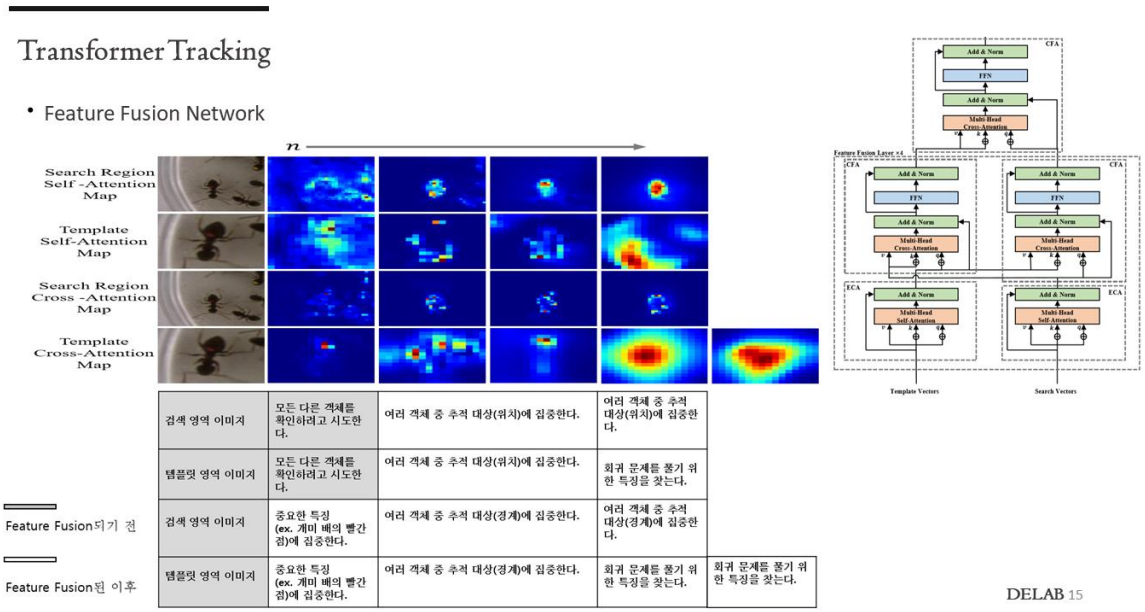
참고로 Query와 Key에만 Positional Encoding을 해도 상관없는 이유는, Attention 연산 시 Query가 요청하는 정보가 Key에 위치한지 파악을 하는 과정에서 위치 정보가 필요하지만, Value는 Key와 대응되는 개념이므로 굳이 위치 정보를 줄 필요가 없기 때문입니다.

그리고 ECA의 역할은 이미지의 특징 표현을 더 강화하는 역할을 하고, CFA의 역할은 Template와 Search Region의 특징을 fusion하는 역할, 즉 공통적인 특징을 찾는 역할을 합니다.



이 네트워크를 좀 더 자세히 살펴보면 다음과 같이 구성되었습니다.

ECA와 CFA로 구성된 Feature Fusion Layer가 총 4번 이루어지고 마지막은 CFA 1번으로 마무리됩니다. 이 때 CFA에 Cross Attention 연산이 있기 때문에 CFA가 핵심이라고 말할 수 있고, CFA를 거친 다음 다시 ECA 입력으로 들어오는 구조이기 때문에, 두 번째 ECA부터는 Feature Fusion된 상태가 입력으로 옵니다. 따라서 ECA의 Self Attention을 거치면 공통적인 특징이 더 부각된 상태가 될 것입니다.



다음은 각 모듈의 Attention Map을 시각화하고 설명을 작성한 표입니다. Feature Fusion되기 전은 회색, 나머지는 흰 색 배경으로 표현을 했는데, 이 때 회색에 해당하는 파트는, 초기 입력과 첫 번째 ECA 모듈의 Attention Map에 해당이 됩니다.

초기에 Self Attention을 할 때는 Template과 Search Region 상관없이 모든 다른 객체를 확인하려고 시도합니다.

이후 처음으로 Cross Attention을 할 때는 동일한 객체(추적 대상)에서 중요한 특징의 역할을 하는 '개미 배의 빨간 점' 따위에 집중하는 모습을 보였다고 합니다.

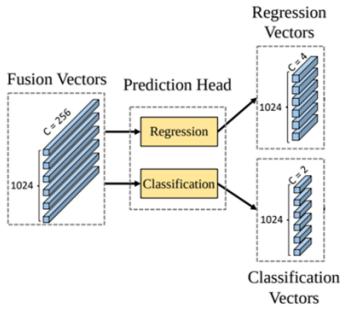
이후 n이 2이상일 때부터는 공통적인 특징의 전체적인 모습에 더욱 집중하는 모습을 보입니다.

이 때는 Self Attention과 Cross Attention이 추구하고자 하는 게 달라 보이는데, Self Attention은 추적 대상(동일 객체)의 위치에 집중하고, Cross Attention은 경계에 집중하는 것처럼 보입니다.

그리고 N이 4일 때, Template Attention Map을 살펴보면 추적 대상에 대한 특징을 찾은 것과는 좀 별개인 느낌을 받습니다. 이 때는, 이미 Target에 대한 위치나 경계 정보를 어느 정도 확보했기 때문에 그 다음 Task에 해당하는 Box Regression 문제를 풀기 위한 특징을 찾는다고 해석할 수 있습니다.

Transformer Tracking

- Prediction Head Network
  - 최종 CFA 모델을 거치면 HWxd 크기의 Feature Map이 나옴 (논문에서는 1024x256)
  - HW개의 vector마다 foreground/background 분류를 함
  - HW개의 vector마다 Normalize된 coordinates를 구함



$$\mathcal{L}_{cls} = -\sum_{j}[y_j \log(p_j) + (1 - y_j)\log(1 - p_j)],$$

L_{cls} : Binary Cross Entropy Loss를 사용해서 Classification
(negative sample에 대한 가중치는 1/16배)

$$\mathcal{L}_{reg} = \sum_{j} \mathbb{1}_{\{y_j=1\}}[\lambda_G \mathcal{L}_{GIoU}(b_j, \hat{b}) + \lambda_1 \mathcal{L}_1(b_j, \hat{b})],$$

L_{reg} : Object가 존재할 시, L1 Loss와 GIoU Loss를 사용해서 Regression
(lambda_G는 2, lambda_1은 5로 설정)

마지막으로 Prediction Head Network입니다.

Transformer의 가장 큰 특징 중 하나가 입력과 출력의 크기가 같다는 부분입니다.

마지막 CFA 모듈의 Query 입력이 Search Region에 대한 Feature Map이므로 출력도 해당 Feature Map의 크기와 같은 1024x256입니다.

이 행렬은 1024개의 벡터라고 볼 수 있는데, 이 네트워크에서는 모든 벡터를 후보박스를 생성하는데 사용합니다. 즉 1024개의 class score정보, 박스 좌표 정보를 예측하는 것입니다.

이 때 사용하는 목적함수는 L_{cls}같은 경우, BCE를 사용하고 Tracking Task 특성 상 Negative Sample이 훨씬 많으므로 Positive를 16배 많이 학습시킵니다.

L_{reg}같은 경우, Object가 존재하는 경우에만 고려를 하는데, 이 때는 L1 Loss랑 GIoU Loss를 같이 사용하므로 최대한 coordinates정보를 잘 예측하는 방향으로 학습시킬 수 있습니다.

이 때 규제에 필요한 lambda는 각각 2랑 5로 설정하였을 때 가장 학습이 잘 됐다고 합니다.

Experiments

- Offline Training
  - COCO, TrackingNet, LaSOT, GOT-10k 데이터로 훈련함
  - Training Samples을 제작하기 위해 각 비디오마다 image pairs를 sampling함
  - COCO Dataset같은 경우, image pairs를 만들기 위해 original image에 transformation을 함

Training Method	
optimizer	AdamW
Learning rate	1e-5 (backbone), 1e-4 (others)
Weight decay	1e-4
Batch Size	38
Epoch	1000 epochs with 1000 Iterations

Dataset	
COCO	330k images, 200k labeled
TrackingNet	30k videos (Diverse Object Class)
LaSOT	1400 videos (High Quality Annotation)
GOT-10k	10k videos, 180 for testing

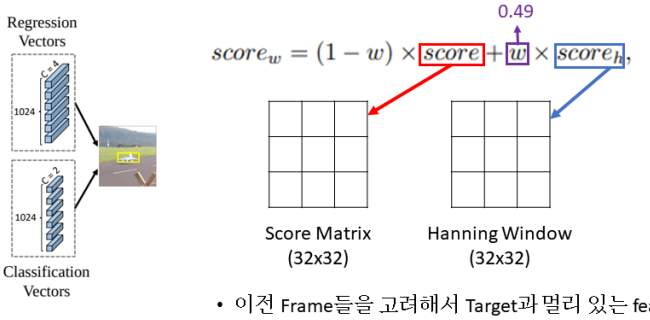
다음은 실험 내용과 결과에 대해 말씀드리겠습니다.

학습 데이터는 COCO, TrackingNet, LaSOT, GOT-10k로 구성이 되었고 이 때 COCO Dataset같은 경우, Detection용 데이터이기는 하나, Tracking도 frame마다 Detection을 하는 작업이기 때문에 같이 사용한 것처럼 보입니다.
데이터셋을 구성할 때는 Template과 Search를 구분하기 위해 Image Pair를 많이 sampling했다고 하고, 이 때 COCO같은 경우 프레임 개념이 없다 보니, (1frame,2frame),(1frame,3frame)따위로 sampling을 할 수가 없습니다. 따라서 transformation을 통해 강제로 Image Pair를 만들었다고 합니다. 예를 들어 (original, blur)인 이미지 쌍으로 추정됩니다.

훈련 방법은 표에 적혀있는 부분을 참고하면 될 것 같고,
훈련 데이터에 대한 소개를 더 드리려고 합니다.
먼저 TrackingNet은 무려 3만개의 비디오를 가지고 있는 대용량 데이터셋입니다. 학습 데이터가 많기 때문에 더 정확도가 높게 나오는 경향이 있습니다.
LaSOT은 1400개의 비디오만 가지고 있지만 굉장히 High Quality한 Annoation을 가지고 있습니다. 뒷장에도 나오지만 비디오에 여러 가지 효과를 준 Annotation들이 굉장히 많습니다.
GOT-10k는 만 개의 비디오를 가지고 있는 평범한 Tracking Dataset이라고 생각하면 됩니다.

Experiments

- Online Tracking
  - Online Tracking은 Inference를 의미함
  - 1024개의 후보 박스에 대해 confidence score를 계산함 (Probability x IoU)
  - Window Penalty라는 Post Processing을 진행함 -> 가장 큰 $score_w$에 해당하는 BBox를 선정



- 이전 Frame들을 고려해서 Target과 멀리 있는 feature들은 값이 낮아짐

인퍼런스에 해당하는 Online Tracking같은 경우, 1024개의 후보 박스 중 한 박스를 고를 필요성이 있습니다. 이 때는 먼저 모든 후보 박스에 대해서 Confidence Score를 계산하는데, 그 과정은 정답일 확률에 IoU를 곱하는 연산입니다.

이후 Window Penalty라는 Post Processing을 진행합니다.

1024개의 Confidence Score를 32x32 행렬로 나타냈을 때 이 행렬과 Hanning Window라는 같은 크기의 필터와 연산을 하는 것입니다. 이 때 연산을 하는 식은 써져 있는 바와 같고 이러한 연산 결과 중 가장 큰 값으로 최종 bbox로 선정하는 것입니다.

참고로 이 Window Penalty라는 방법을 쓰면 이전 Frame들을 고려해서 Target과 멀리 있는 feature들은 값이 낮아지게 된다고 합니다.

자세하게는 모르겠으나 Object가 중간쯤에 위치한다면 외곽 부분의 Hanning Window 필터 값($score_h$)은 낮아지면서 해당 feature의 점수($score_w$)가 낮아지는 것이 아닌가 생각이 됩니다.

Experiments

- Results
  - 기존 Correlation Model들을 압도하는 퍼포먼스를 보임 (SiamR-CNN 제외)

Table 1. State-of-the-art comparison on TrackingNet, LaSOT, and GOT-10k. The best two results are shown in **red** and **blue** fonts.

Method	Source	LaSOT [14]			TrackingNet [30]			GOT-10k [19]		
		AUC	P_{Norm}	P	AUC	P_{Norm}	P	AO	$SR_{0.5}$	$SR_{0.75}$
TransT	Ours	**64.9**	**73.8**	**69.0**	**81.4**	**86.7**	**80.3**	**72.3**	**82.4**	**68.2**
TransT-GOT	Ours	-	-	-	-	-	-	**67.1**	**76.8**	**60.9**
SiamR-CNN [39]	CVPR2020	**64.8**	**72.2**	-	**81.2**	**85.4**	**80.0**	64.9	72.8	59.7
Ocean [48]	ECCV2020	56.0	65.1	56.6	-	-	-	61.1	72.1	47.3
KYS [3]	ECCV2020	55.4	63.3	-	74.0	80.0	68.8	63.6	75.1	51.5
DCFST [49]	ECCV2020	-	-	-	75.2	80.9	70.0	63.8	75.3	49.8
SiamFC++ [44]	AAAI2020	54.4	62.3	54.7	75.4	80.0	70.5	59.5	69.5	47.9
PrDiMP [10]	CVPR2020	59.8	68.8	60.8	75.8	81.6	70.4	63.4	73.8	54.3
CGACD [13]	CVPR2020	51.8	62.6	-	71.1	80.0	69.3	-	-	-
SiamAttn [46]	CVPR2020	56.0	64.8	-	75.2	81.7	-	-	-	-
MAML [40]	CVPR2020	52.3	-	-	75.7	82.2	72.5	-	-	-
D3S [26]	CVPR2020	-	-	-	72.8	76.8	66.4	59.7	67.6	46.2
SiamCAR [16]	CVPR2020	50.7	60.0	51.0	-	-	-	56.9	67.0	41.5
SiamBAN [5]	CVPR2020	51.4	59.8	52.1	-	-	-	-	-	-
DiMP [2]	ICCV2019	56.9	65.0	56.7	74.0	80.1	68.7	61.1	71.7	49.2
SiamPRN++ [21]	CVPR2019	49.6	56.9	49.1	73.3	80.0	69.4	51.7	61.6	32.5
ATOM [9]	CVPR2019	51.5	57.6	50.5	70.3	77.1	64.8	55.6	63.4	40.2
ECO [8]	ICCV2017	32.4	33.8	30.1	55.4	61.8	49.2	31.6	30.9	11.1
MDNet [31]	CVPR2016	39.7	46.0	37.3	60.6	70.5	56.5	29.9	30.3	9.9
SiamFC [1]	ECCVW2016	33.6	42.0	33.9	57.1	66.3	53.3	34.8	35.3	9.8

그래서 실험 결과를 살펴보면 이전에 Correaltion 방식을 사용했던 대부분의 모델들을 압도하는 퍼포먼스를 보입니다. 물론 SiamR-CNN 모델 같은 경우 예외적인 부분이기는 하나 이전에 높게 평가됐던 Ocean 모델보다도 약 8% 가량 높은 AUC를 보입니다.

Experiments

- Results
  - 왼쪽 그림은 LaSOT dataset에 대한 AUC 비교
  - 오른쪽 표는 Post Processiong 여부에 따른 성능 비교

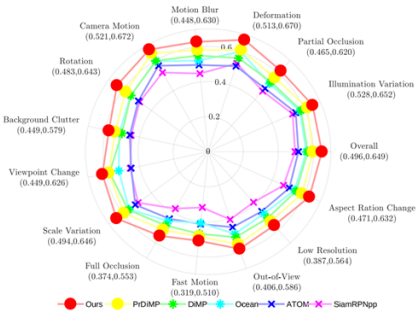


Figure 5. AUC scores of different attributes on the LaSOT dataset.

Method	LaSOT [14]			TrackingNet [30]			GOT-10k [19]		
	AUC	P_{Norm}	P	AUC	P_{Norm}	P	AO	$SR_{0.5}$	$SR_{0.75}$
TransT	**64.9**	**73.8**	**69.0**	**81.4**	**86.7**	**80.3**	**72.3**	**82.4**	**68.2**
TransT-np	62.9	71.5	66.9	81.1	86.4	80.0	71.5	81.5	67.5
TransT(ori)	62.3	71.1	66.2	81.3	86.1	78.9	70.3	80.2	65.8
TransT(ori)-np	60.9	69.4	64.8	80.9	85.6	78.4	68.6	78.2	65.1

그리고 LaSOT Dataset으로 AUC를 비교할 때도 다양한 비디오 효과에 무관하게 항상 최고의 성능을 보였다고 말하고 있고,
오른쪽 표 같은 경우, Post Processing을 하면서 ECA와 CFA모듈을 사용했을 때 가장 성능이 좋았다고 말하고 있습니다.

Experiments

- Comparison with correlation
  - Correlation 대신 CFA 모듈을 사용 시 성능이 약 **15%** 상승
  - ECA 모듈과 같이 사용 시 성능이 **2%** 추가 상승

Method	ECA	CFA	Correlation	LaSOT [14]			TrackingNet [30]			GOT-10k [19]		
				AUC	P_{Norm}	P	AUC	P_{Norm}	P	AO	$SR_{0.5}$	$SR_{0.75}$
TransT	√	√		**64.9**	**73.8**	**69.0**	**81.4**	**86.7**	**80.3**	**72.3**	**82.4**	**68.2**
TransT		√		62.9	71.9	66.2	81.1	86.2	79.1	70.6	81.2	65.7
TransT	√		√	57.7	65.4	59.5	77.5	82.2	74.0	62.8	72.2	54.8
TransT			√	47.7	48.6	41.7	68.8	71.4	60.9	50.9	58.0	33.3
TransT-np	√	√		62.9	71.5	66.9	81.1	86.4	80.0	71.5	81.5	67.5
TransT-np		√		61.0	69.6	64.5	80.0	85.0	77.9	68.1	78.3	64.0
TransT-np	√		√	57.3	65.2	58.8	76.2	80.8	72.8	61.4	70.7	53.7
TransT-np			√	35.3	17.9	20.1	46.5	40.3	27.4	38.2	36.8	7.0

Correlation based Feature Fusion		Cross Attention based Feature Fusion
semantic information이 부족함	⟷	semantic information이 풍부함
Global context를 활용하지 못함		Global context를 잘 활용함

마지막으로 가장 중요한 correlation mathod와의 비교입니다.
ECA를 사용하지 않은 경우 CFA를 사용한 것만으로 성능이 약 15% 상승이 됩니다.
여기에 ECA를 사용하면 Feature Fusion된 결과를 더 부각시키므로 성능이 2% 추가 상승됩니다.

이것이 가능한 이유는 correlation은 의미있는 정보도 부족하고 global context를 활용하지도 못하는 반면, cross attention은 의미있는 정보도 충분하고 global context를 잘 활용하기 때문이라고 합니다.