MoViNets: Mobile Video Networks for Efficient Video Recognition

2021년 초, 비교적 최근에 나온 논문이고 구글에서 내서 올해 CVPR에도 실린 논문입니다.

제가 이번 방학 동안 Efficient DL 스터디를 하면서 경량화 및 mobile 모델들에 대한 공부를 했었는데 이의 연장선으로, 그리고 지금 면수님이랑 annotation과 함께 진행하는 작은 video recognition project에 잘 맞는 것 같아 흥미로워 가져오게 됐습니다.

[abstract]

현재 비전에서 architecture에 관한 여러가지 연구가 진행되었지만 이제 모바일 적용을 하려면 경량화가 필요했기 때문에 그 기술이 필요한 구글이 이와 관련한 연구를 많이 한 것 같습니다. 우리가 흔히 사용하는 3D CNN은 video recognition에서 뛰어난 성능을 보여주지만 parameter들 때문에 연산속도가 매우 느리고 메모리가 부족한 상황이 많았습니다. 대부분의 현존 모델은 gpu 사용을 하고 있으며 실시간으로 on-device processing이 불가능했다. 하지만 해당 논문에서는 3가지의 방법 제시를 통해 높은 정확도와 연산 효율을 얻을 수 있었다고 합니다. 그 3가지는 {1,2,3} 다음과 같은데 이제부터 설명을 해드리겠습니다.

[Introduction]

3D CNN을 활용한 모델 (e.g. temporal convolution, non-local blocks, etc.)은 한번에 모든 input frames를 연산하기 때문에 mobile devices에 적용하면 메모리 limit으로 인해 정확도가 낮았습니다.  또한, 최근까지 각광받던 X3D networks는 3D CNN의 효율을 높혔지만 large temporal window를 사용하면 high cost, small temporal windows를 사용하면 low accuracy를 보였다.

    - MoViNet search space : Neural Architecture Search (NAS)가 시공간 feature를 효과적으로 trade-off 한다.

    - Stream Buffers : 비디오를 연속적인 subclips으로 처리한다. -> online prediction 가능

    - Temporal Ensembles : 독립적으로 두 MoViNets를 학습시키고 평균값을 취한다.

[movinet search space]

MobileNetV3를 기반으로  search space를 만들었다. MobileNetV3는 AotuoML적인 관점을 이용해 structure를 search하고 tuning한다. inverted bottleneck layers (varying filter widths, bottleneck widths, block depths, and kernel sizes per layer)의 blocks들로 구성되어 있다. X3D와 비슷하게 MobileNetV3의 2D blocks를 3D video를 다룰 수 있도록 확장시켰다.

T frames, S^2 resolution(해상도 HxW), fps(위의 table에서는 stride \tau로 표현)값을 가진 video가 input data로 들어오게 된다. 각각의 block을 위해 base filter width c^base와 block내에서 반복 횟수를 정하는 layers L <= 10을 찾는다. 모든 block마다 {0.75, 1, 1.25}를 feature map channels에 곱하고 8의 배수로 반올림한다. (n=5로 5개의 block이 지정된다.) 마지막 block의 spatial resolution이 7^2가 되도록 하기 위해 4번째 block)을 제외하고 나머지는 각 block의 1번째 layer는 strided spatial downsampling을 실시한다. block은 점진적으로 feature map channels을 증가시킨다 {16, 24, 48, 96, 96, 192}. 최종 convolutoin layer's base filter width는 512이고 classification layer 전에 2048D dense layer가 있다.

새로운 time dimension과 함께, 각 layer의 3D kernel size (k^time x (k^space)^2)를 고른다. {1x3x3, 1x5x5, 1x7x7, 5x1x1, 7x1x1, 3x3x3, 5x3x3} 중에서 선택한다. 이는 네트워크의 receptive field가 이상적인 방향으로 커지도록 하면서 layer가 다른 차원의 특징에 집중할 수 있기 때문이다. 몇몇 kernel size는 input filter 개수를 다르게 가지는 것이 좋을 수 있다. 따라서 bottleneck widths c^expand를 c^base에 관하여 {1.5, 2.0, 2.5, 3.0, 3.5, 4.0} 곱하며 찾는다. 각 layer는 1x1x1 3D conv layer가 전후로 위치하여 c^base와 c^expand 사이에서 expand와 project를 한다.

spatial squeeze-and-excitation (SE)를 적용하는 대신 SE blocks를 사용하여 시공간적 feature를 강조하기 위해 3D average pooling을 모든 bottleneck block에 적용한다. 하지만 SE block을 적용하면 연산량이 늘어날 수 있기 때문에 사용하지 않아도 된다.

[scaling]

search space는 MoViNet-A2를 기준이 되고 다른 MoViNet에 대해서는 EfficientNet에서도 사용된 compound scaling을 적용한다. EfficientNet과 다른점은 single model에 대해  scale하지 않고 search space 스스로에 대해서만 scale한다.

Memory grows proportionally to the number of input frames: 그래서 해결을 하기 위해 stream buffer가 나왔다

[stream buffer]

(원래)

memory를 줄이기 위한 기존의 방법은 위의 이미지와 같다. 전체 T frame에서 subclips T^clip을 overlapping하여 각각의 예측값들을 평균내어 최종값을 산출해낸다. T^clip < T frames이기 때문에 계산량은 O(T^clip)이지만 한계는 여전히 존재한다.

    - 각 subclip에 대해 temporal receptive fields에 제한을 주고 long range dependencies를 무시한다.

    - overlap된 부분을 두번 계산하기 때문에 효율성이 다소 감소한다.

(stream buffer는)

비디오를 n개의 non-overlapping subclips로 나눴을 때, 현재의 subclip을 x\_i^clip이라고 둔다 (i < n). b 길이의 buffer B와 subclip을 time dimension에 맞게 concatenate한다. (아래의 f() 함수는 spatiotemporal operation으로 3D convolution 등이 있다.)

Fi=f(Bi⊕xclipi)Fi=f(Bi⊕xiclip)

다음 clip을 계산할 때, buffer를 update한다.

Bi+1=(Bi⊕xclipi)[−b:]Bi+1=(Bi⊕xiclip)[−b:]

([-b:]는 마지막 concatenated input의 마지막 b frame을 의미한다. 따라서 subclips n이 증가하거나 전체 비디오 frame T와 일정한 계산 복잡도를 지닌다.)

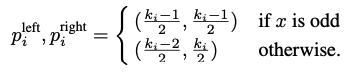
 (F: feature map)

Stream buffer를 사용하면, 16 프레임의 단일 클립을 8 프레임의 클립 2개로 나눌 수 있고 batch 사이즈를 2배로 늘릴 수 있다. 이는 buffer를 사용하지 않았을 때보다 훨씬 큰 efficiency 효과를 지닌다.

[causal operation]

stream buffer의 causality (인과성)을 위해 feature는 future frames으로부터 절대로 계산하지 않는다. 이 방법을 사용하면 subclip을 activation이나 prediction에 영향을 주지 않으면서 single frame으로 줄일 수 있고 3D CNN이 실시간 비디오에 대해 예측이 가능하다.

convolution의 translation equivariant property때문에 모든 temporal convolution을 CausalConvs로 바꾸었다. 위의 사진에서 볼 수 있듯이 기존의 convolution은 점차 중앙으로 몰리게 된다. 따라서 padding trick을 적용한 CausalConvs로 교체한 것이다. normal 3D convolution padding은 (k\_t, k\_h, k\_w), stride s = 1에서 다음과 같을 때,



time axis padding은 아래와 같다.

https://blog.kakaocdn.net/dn/bs2Y2c/btq2Rkck7Rf/SZfwzPW05KtCbGrDvBhqPk/img.png

(또한, stream buffer와 CausalConv 사용 할 때, causal padding을 buffer itself로 바꾸어서 memory를 아낄 수 있다.) 이전 subclip의 마지막의 몇몇 frames을 복사하여 다음 subclip의 padding으로 사용한다. 이를 통해 memory를 아낄 수 있( stream buffers는 오직 다수 frame에 걸쳐진 feature를 뽑아내는 layer 이전에서만 필요한 것을 알 수 있다.)

[cgap]

CGAP를 사용하여 temporal dimension을 포함한 global average pooling을 근사한다.

CGAP를 causally하게 계산하기 위해, cumulative sum을 T'까지 저장하는 single-frame stream buffer를 저장한다.

[positional encoding]

CausalSE는 CGAP를 SE로 적용한 것으로 frame t의 spatial feature map을 CGAP(x, t)로 계산된 SE와 곱하는 것이다. 하지만 결과를 보았을 때, CausalSE가 불안정한 경향이 있었다. 따라서 Transformer의 positional encoding 방법에서 착안하여 sine-based fixed positional encoding를 적용했다. position에 따라 직접적으로 frame index를 사용하고 SE projection 전에 CGAP output vector와 더했다.

Training and Inference

1) Training

memory (사용을 줄이기 위해 batch of examples를 n subclips로 나누고 각 subclip에 대해 prediction을 했다). 하지만 이전 buffer에 backprop를 하지 않음으로써 이전 subclips에 할당된 메모리를 제거할 수 있었다. 대신에 loss를 계산하고 subclips 간에 gradients를 구하고 축적했다. 그 결과 네트워크가 모든 T = nT^clip frames를 고려하며 gradient 적용 전 n forward passes를 수행할 수 있다. 더 긴 temporal에서도 충분히 학습 가능하다.

T^clip을 어떤 값으로 설정하더라도 accuracy에는 영향을 끼치지 않는다. 다만, 논문에서는 빠른 훈련을 위해 GPU 사용했고 T^clip 값으로 {8, 16, 32}를 사용했다.

2) Online Inference

training과 유사하게 subclips 간의 activations를 cache하기 위해 stream buffer를 사용했다. 하지만 T^clip = 1 (single frame)으로 subclip length를 설정하여 memory를 아꼈다. CausalConv와 CausalSE를 사용함으로써 3D video CNN이 실시간으로 가능한 것이 가장 큰 이점이다. (기존의 convolutional RNN과 다르게 동일한 output을 만들기 위해 step 당 frame의 수를 가변적으로 설정할 수 있다. streaming architectures with CausalConv에서는 CGAP를 사용하여 frame-by-frame output features를 pooling하여 label을 예측한다.)

3) Temporal Ensembles

stream buffers를 사용하며 memory 사용량을 줄일 수 있었지만 1% 정도의 정확도가 떨어졌다. 따라서 ensembling strategy를 사용했다. 동일한 architecture의 MoViNets 두 개를 독립적으로 훈련시켰다. 하지만 frame-rate를 절반으로 나누고 temporal duration은 동일하게 했다. 비디오를 두 개의 네트워크에 입력하는데 softmax를 적용하기 전에 하나의 네트워크가 frame을 한 frame으로 offset하고 가중치가 없는 logits에 산술 평균을 적용한다. 이 방법은 frame-rate를 절반으로 줄이기 전에 단일 모델과 동일한 FLOP을 가진 두개의 모델 앙상블을 생성하여 풍부한 예측이 가능하다.

1) Single-Clip vs. Multi-Clip Evaluation

fixed temporal stride로 input video를 single-clip으로 샘플링하여 모델을 평가하였다. multi-clip와 비교하여 FLOPs는 비슷했지만 single-clip이 더 높은 accuracy를 보여주었다. ? multi clip?

Stream buffer가 없을 때, mobilenetv3에 비해 gflop은 감소한 반면 accuracy는 10% 증가. movinet-a0은 X3d보다 gflop과 accuracy 모두 향상 됐고 -a1은 accuracy가 2% 증가

With stream buffers

왼쪽 위: X3d에 비해 accuracy, memory efficient(이건 multi clip 사용)

아래: streaming movinets은 gflops 측며에서 efficient 하다

Stream buffer를 적용 했을 때도 모든 모델에서 1% accuracy가 하락하지만 ensembling까지 apply하면 inference 추가 비용 없이 accuracy를 다시 끌어올릴 수 있다

Stream buffer를 통해 메모리를 줄이지만 accuracy에 추가 cost없이 balance를 유지할 수 있다

[5가지]

5가지 dataset에 적용을 했다

전체적으로 flop은 감소, accuracy는 증가

Se를 적용해야지 high accuracy를 achieve할 수 있다