Exploring Plain Vision Transformer Backbones for Object Detection

ECCV 2022

Yanghao Li, Hanzi Mao, Ross Girshick†, Kaiming He†
Facebook AI Research

Presenter: Jaeju An
2022-10-04

Exploring Plain Vision Transformer Backbones for Object Detection

→ Plain Vision Transformer를 Object Detection에 적용하기 (Re-designing X)

Modern Object Detectors

- There are bunch of detection design . . .

Ex) Fast RCNN, Faster RCNN, Mask RCNN, …

- Components: Backbone, Neck, Head

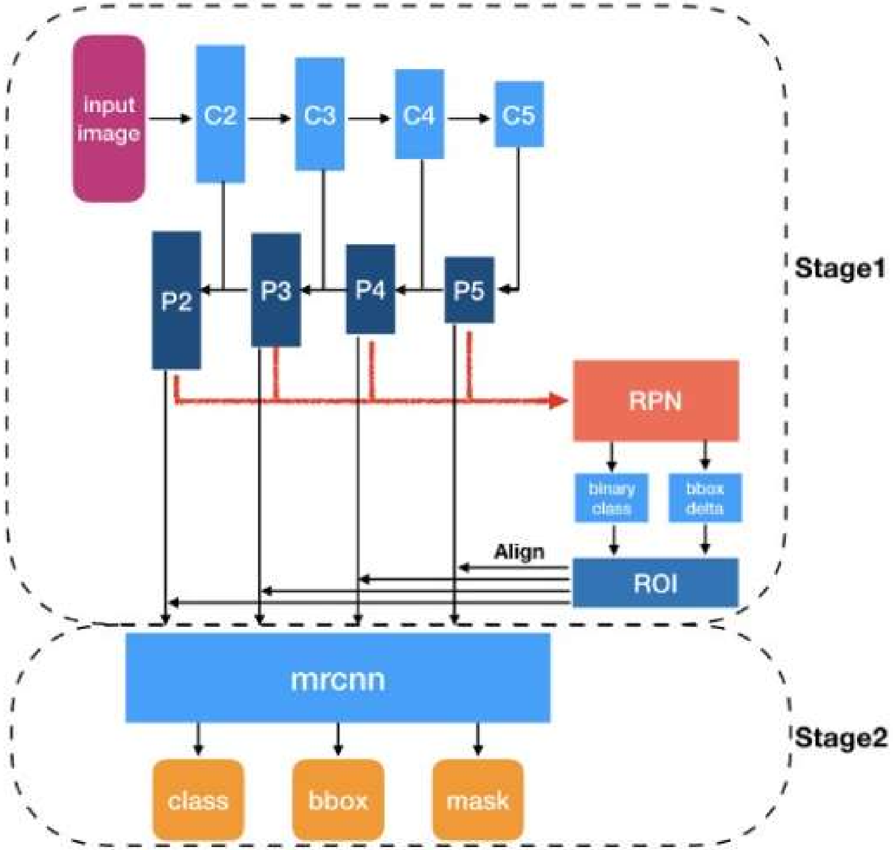


Illustration of Mask RCNN structure

Modern Object Detectors

- There are bunch of detection design . . .

Ex) Fast RCNN, Faster RCNN, Mask RCNN, …

- Components: Backbone, Neck, Head

Backbone. Extracting Features

Ex) ResNet, EfficientNet, etc.,

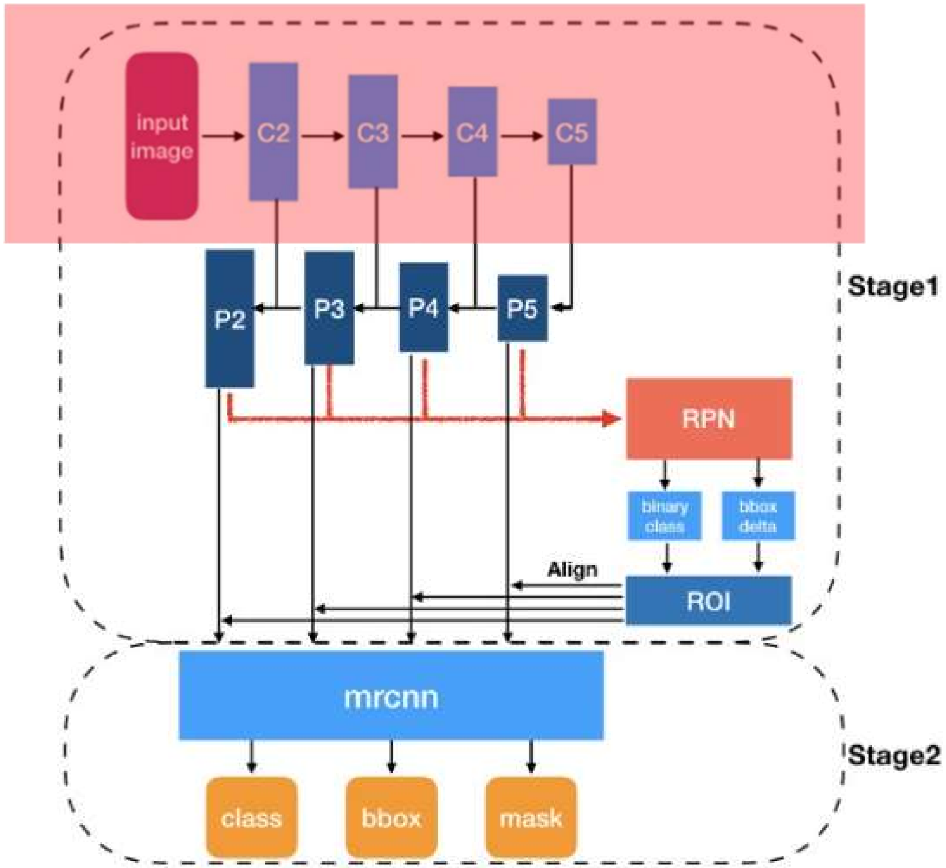


Illustration of Mask RCNN structure

Modern Object Detectors

- There are bunch of detection design . . .

Ex) Fast RCNN, Faster RCNN, Mask RCNN, …

- Components: Backbone, Neck, Head

Backbone. Extracting Features

Ex) ResNet, EfficientNet, etc.,

Neck. Aggregating Features with RoI

Ex) RoI-Align, RoI-Pooling, etc.,

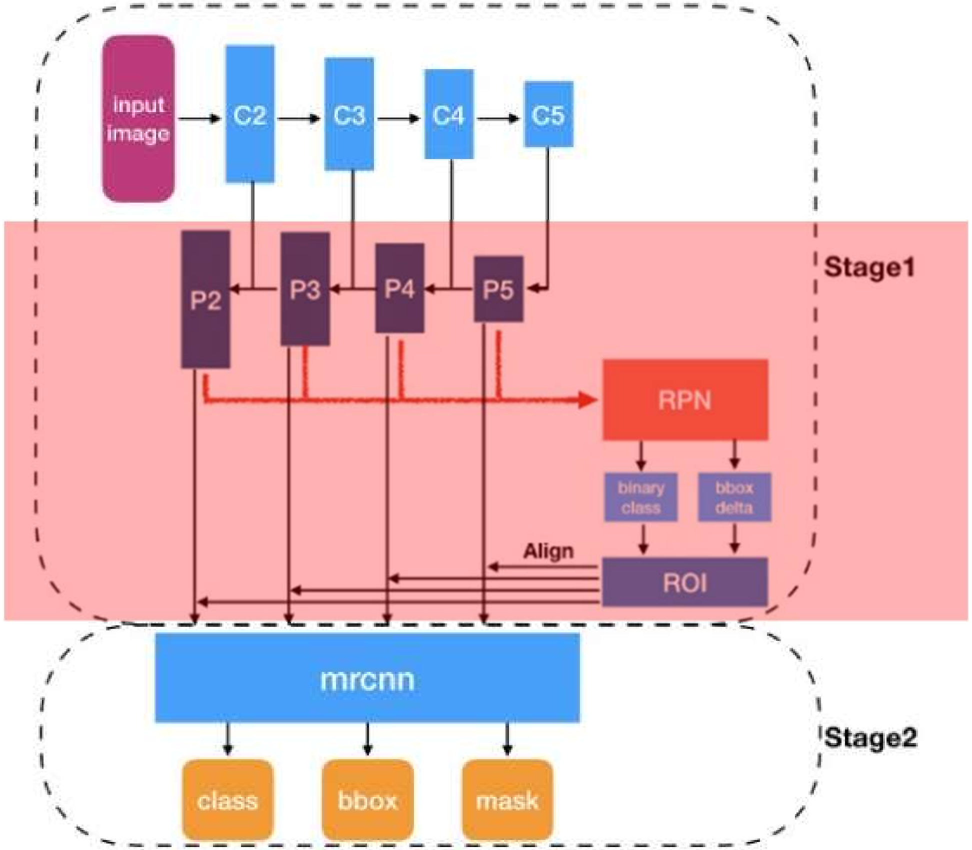


Illustration of Mask RCNN structure

Modern Object Detectors

- There are bunch of detection design . . .

Ex) Fast RCNN, Faster RCNN, Mask RCNN, …

- Components: Backbone, Neck, Head

Backbone. Extracting Features

Ex) ResNet, EfficientNet, etc..

Neck. Aggregating Features with RoI

Ex) RoI-Align, RoI-Pooling, etc..

Head. Predict with Aggregated Features

Ex) Bbox head, Class head, mask head, etc..

- 보통 1 linear layer

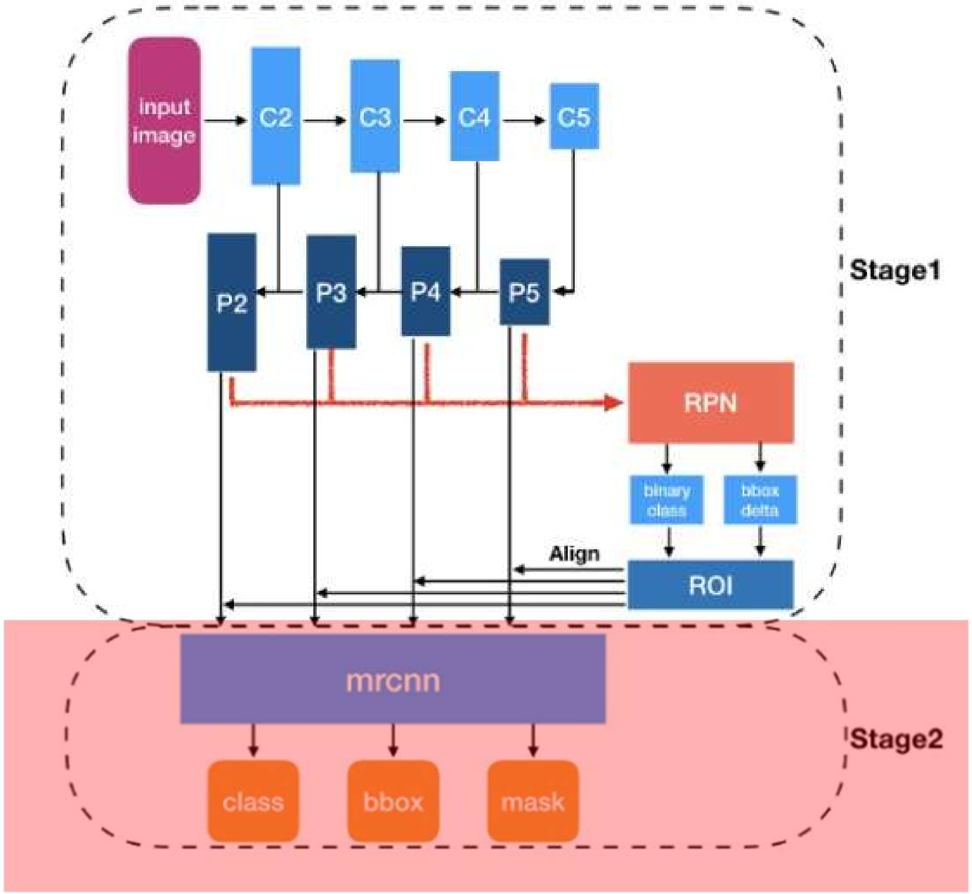
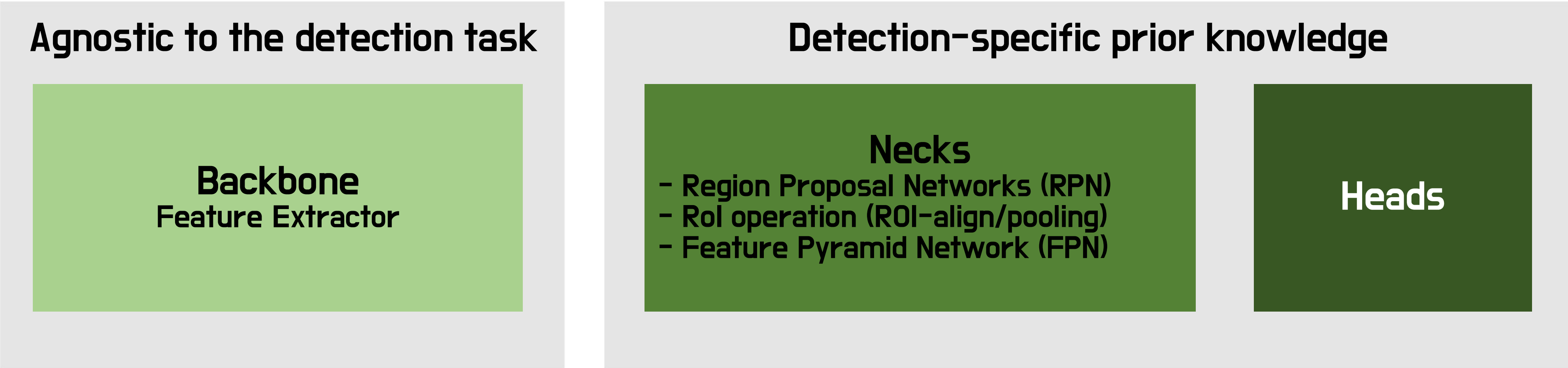


Illustration of Mask RCNN structure

Modern Object Detectors



Detection-Agnostic Backbone

- Practical Usage: ResNet 101
- 아니면 다른 CNN-based 모델을 차용

Detection-Specific Functions

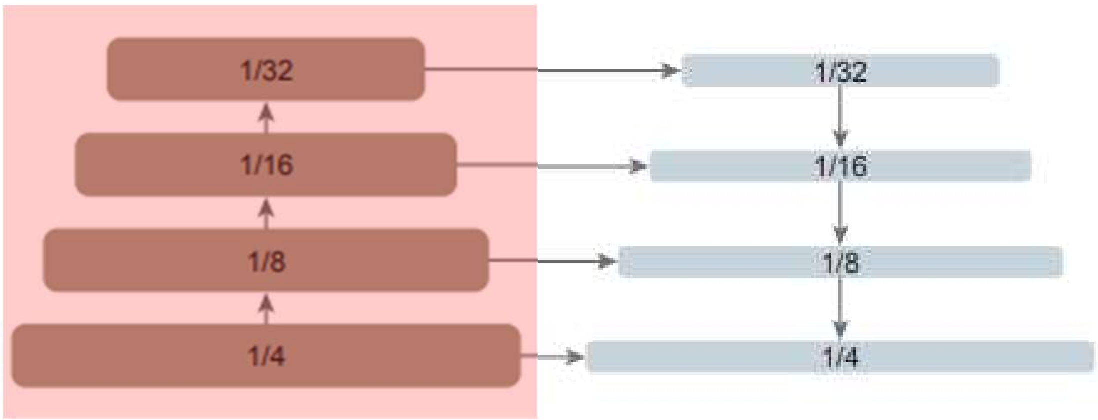
- detection-specific functions은 backbone과는 무관하게 발전되어온 것들임
  - → Backbone과 무관하게 발전할 수 있던 이유: Backbone이 모두 (ConvNet의 디자인의 영향으로) multi-scale과 hierarchical한 구조를 갖고있다.
  - → 모든 detection design이 (Fast RCNN, Mask RCNN, Cacaded RCNN) 상기의 구조를 갖고있음

Modern Object Detectors. Feature Pyramid Network

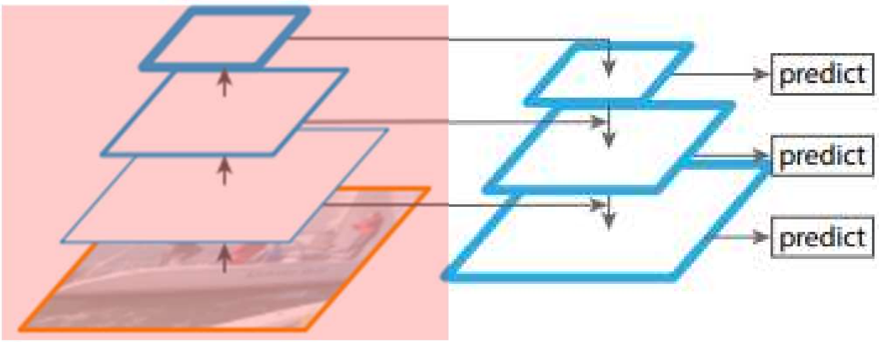
- Object Detection이 무조건 Hierarchical 구조를 갖게 하는 원인: Feature Pyramid Network

Feature Pyramid Networks for Object Detection, CVPR 2017, Facebook AI Research

Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie

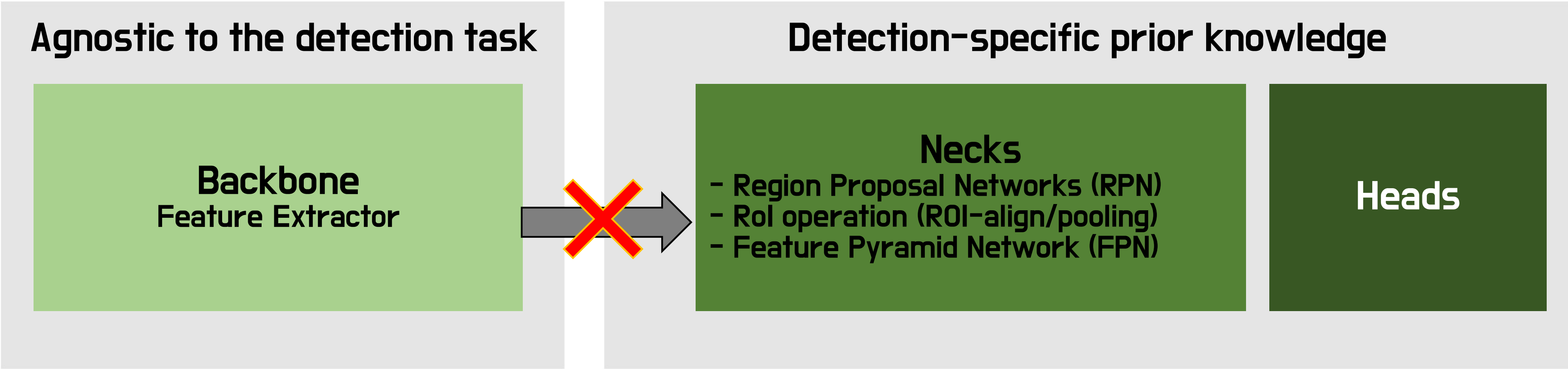


hierarchical backbone, w/ FPN



(d) Feature Pyramid Network

Modern Object Detectors for ViT?



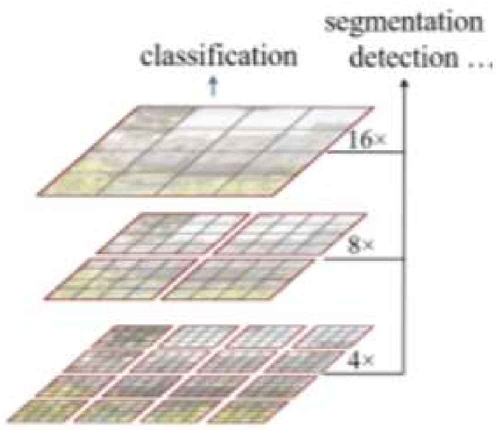
- Powerful Backbone Introduced ViT (Vision Transformer)
  - ViT is a plain, non-hierarchical architecture maintains a single-scale feature map throughout

- ViT를 backbone으로 Detection task에 적용할 때 문제가 발생 !

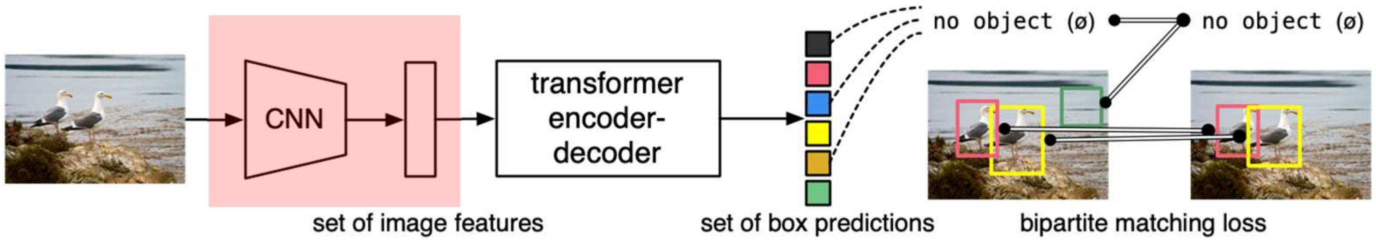
Modern Object Detectors for ViT?

- ViT를 backbone으로 Detection task에 적용할 때 문제가 발생 !

Solution 1. Re-introduce hierarchical aspects into the plain ViT backbone.
-> Swin Transformers and others



Swin Transformer



DETR Detection Framework

Modern Object Detectors for ViT?

- ViT를 backbone으로 Detection task에 적용할 때 문제가 발생 !

Solution 2. Use plain, non-hierarchical backbones with precise adaptation
-> plain: non-hierarchical, single-scale property
-> backbone: pre-training available architecture

RQ. Downstream task로 object detection을 할 때, Plain backbone을 수정없이 어떻게 이용?

Modern Object Detectors for ViT?

해당 연구가 성공적이면 . . .

1. Enable the use of original ViT backbones for object detection

→ This will decouple the pre-training design from the fine-tuning demands, maintaining the independence of upstream vs. downstream tasks, as has been the case for ConvNet-based research.

2. Achieve less inductive biased detection framework than other Transformer-based models

→ As the non-local self-attention computation can learn translation-equivariant features (find-tuning), they may also learn scale-equivariant features from certain forms of supervised or self-supervised learning (pre-training).

Methodology

Goal: Removing the hierarchical constraint on the backbone

→ Apply Minimal modifications to adapt a plain backbone to the object detection task only fine-tuning time

→ It should be available on any detection mechanism, such as, Fast RCNN, Mask RCNN, or Cascaded families

Simple Feature Pyramid & Backbone adaptation

Methodology

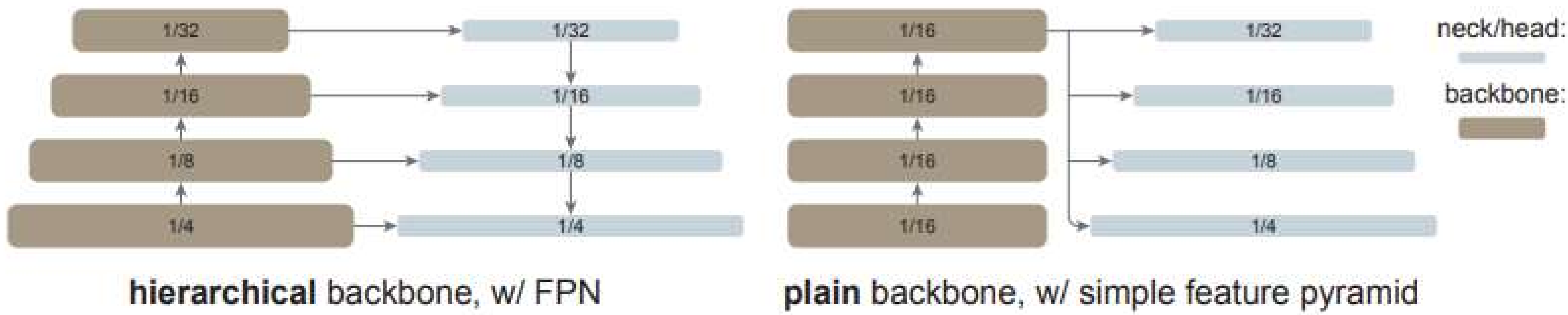
Simple Feature Pyramid & Backbone adaptation

Minimal Adaptation:
→ Abandon Feature Pyramid Design, instead, use only the last feature map of plain ViT backbone
→ It does not require hierarchical backbone

추가적인 문제 발생
+ Object Detection uses 'High Resolution Image' → ViT의 경우 Quadratic Complexity 발생
→ 해결하기위해 non-overlapping window attention 사용 (without 'shifting', Swin이랑 다르다는 얘기)

위 adaptation은 fine-tuning에서만 동작함! 따라서 pre-training을 어떻게 가져가던 상관없음

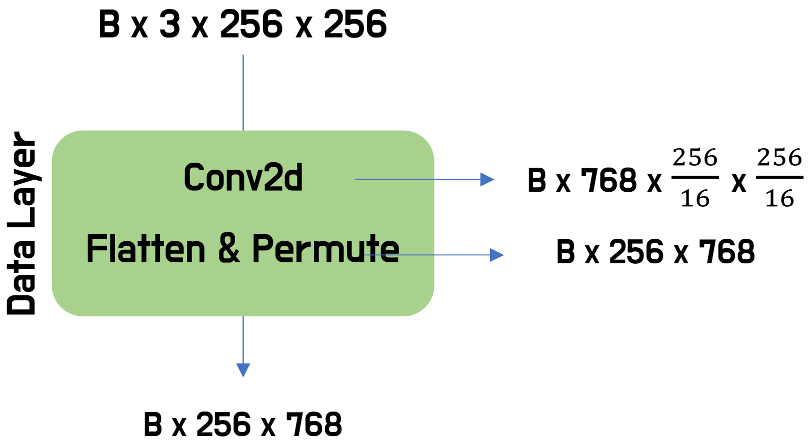
Methodology 1) Simple Feature Pyramid. FPN vs. SPF



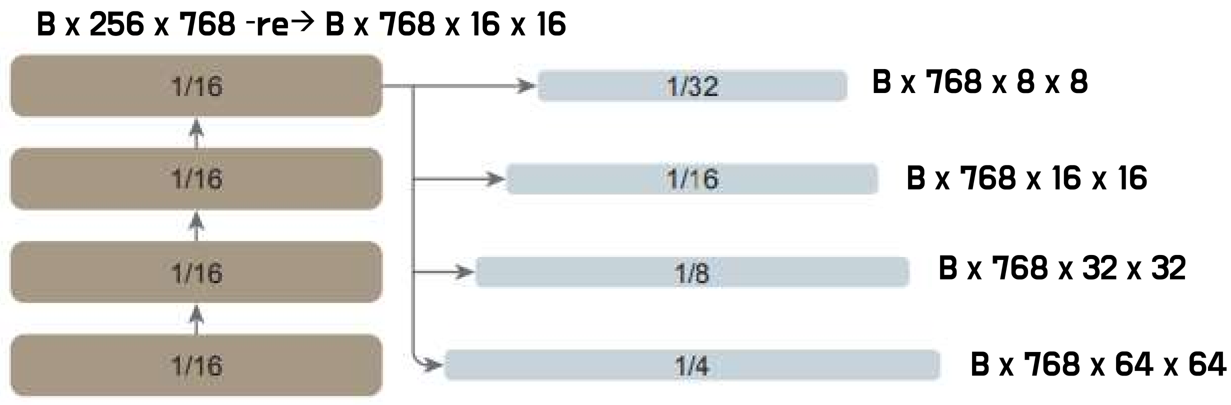
Feature Pyramid Network (FPN): combine the higher-resolution features from earlier stages And the stronger features from later stages

Simple Feature Pyramid (SFP): use only the last feature map from the backbone, which should have the strongest features

Methodology 1) Simple Feature Pyramid. Details & Implementation




```
conv2d = torch.nn.Conv2d(3, 768, kernel_size=16, stride=16)
tensor = torch.randn(1, 3, 256, 256)
res = conv2d(tensor).flatten(2).transpose(1, 2)
res.shape
✓ 0.4s
torch.Size([1, 256, 768])
```



(c) simple feature pyramid

we produce feature maps of scales { 32 , 16 , 8 , 4 } using convolutions of strides {2, 1, $\frac{1}{2}$, $\frac{1}{4}$ },
where a fractional stride indicates a deconvolution.
-> Deconvolution can be implemented many ways

Unofficial Implementation: https://github.com/ViTAE-Transformer/ViTDet/tree/main/mmdet/models/backbones

Methodology 1) Simple Feature Pyramid. Details & Implementation

```python
kernel = 2
embed_dim = 768
conv_level1 = torch.nn.Conv2d(embed_dim, embed_dim, kernel, stride=2)
conv_level2 = torch.nn.Conv2d(embed_dim, embed_dim, kernel, stride=1, padding='same')

# Deconv. 1
conv_level3 = torch.nn.Sequential(torch.nn.ConvTranspose2d(embed_dim, embed_dim, kernel_size=2, stride=2),)
conv_level4 = torch.nn.Sequential(
    torch.nn.ConvTranspose2d(embed_dim, embed_dim, kernel_size=2, stride=2),
    # Norm2d(embed_dim),
    # nn.GELU(),
    torch.nn.ConvTranspose2d(embed_dim, embed_dim, kernel_size=2, stride=2),
)

tensor = torch.randn(32, 256, 768)
tensor = tensor.reshape(32, 768, 16, 16)

feat1  = conv_level1(tensor)
feat2  = conv_level2(tensor)
feat3  = conv_level3(tensor)
feat4  = conv_level4(tensor)

feat1.shape, feat2.shape, feat3.shape, feat4.shape,
```

1.5s

```
(torch.Size([32, 768, 8, 8]),
 torch.Size([32, 768, 16, 16]),
 torch.Size([32, 768, 32, 32]),
 torch.Size([32, 768, 64, 64]))
```

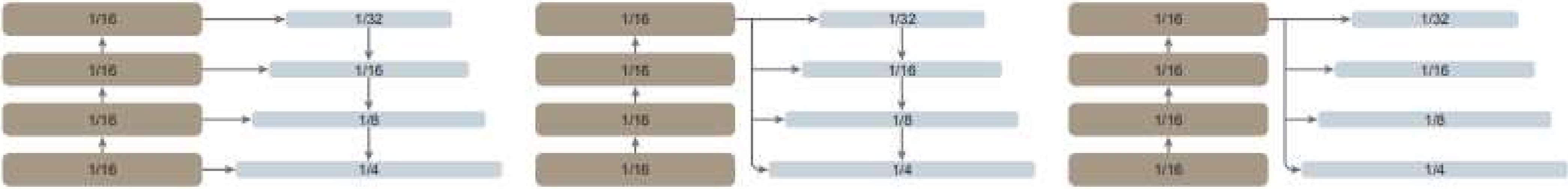
Methodology 1) Simple Feature Pyramid. Details & Implementation

```python
kernel = 2
embed_dim = 768
conv_level1 = torch.nn.Conv2d(embed_dim, embed_dim, kernel, stride=2)
conv_level2 = torch.nn.Conv2d(embed_dim, embed_dim, kernel, stride=1, padding='same')

# Deconv. 2
conv_level3 = torch.nn.Sequential(torch.nn.Upsample(scale_factor=2), torch.nn.Conv2d(embed_dim, embed_dim, kernel, stride=1, padding='same'))
conv_level4 = torch.nn.Sequential(torch.nn.Upsample(scale_factor=4), torch.nn.Conv2d(embed_dim, embed_dim, kernel, stride=1, padding='same'))

tensor = torch.randn(32, 256, 768)
tensor = tensor.reshape(32, 768, 16, 16)

feat1  = conv_level1(tensor)
feat2  = conv_level2(tensor)
feat3  = conv_level3(tensor)
feat4  = conv_level4(tensor)

feat1.shape, feat2.shape, feat3.shape, feat4.shape,
```

4] ✓ 1.5s

```
(torch.Size([32, 768, 8, 8]),
 torch.Size([32, 768, 16, 16]),
 torch.Size([32, 768, 32, 32]),
 torch.Size([32, 768, 64, 64]))
```

Methodology 1) Simple Feature Pyramid. Variants



(a) FPN, 4-stages (b) FPN, last map (c) simple feature pyramid

pyramid design	ViT-B AP^{box}	ViT-B AP^{mask}	ViT-L AP^{box}	ViT-L AP^{mask}
no feature pyramid	47.8	42.5	51.2	45.4
(a) FPN, 4-stage	50.3 (+2.5)	44.9 (+2.4)	54.4 (+3.2)	48.4 (+3.0)
(b) FPN, last-map	50.9 (+3.1)	45.3 (+2.8)	**54.6** (+3.4)	48.5 (+3.1)
(c) simple feature pyramid	**51.2** (+3.4)	**45.5** (+3.0)	**54.6** (+3.4)	**48.6** (+3.2)

Plain ViT 기반 FPN design에 따른 성능 변화 결과

Methodology 2) Backbone Adaptation

- Pre-trained backbone performs global self-attention
- Object Detection uses Higher-resolution inputs during fine-tuning
- Need Global Attention !
  → However, performing Global Attention with all pixels would be memory-exhasutive
- Use Window Attention

Default: Self-attention

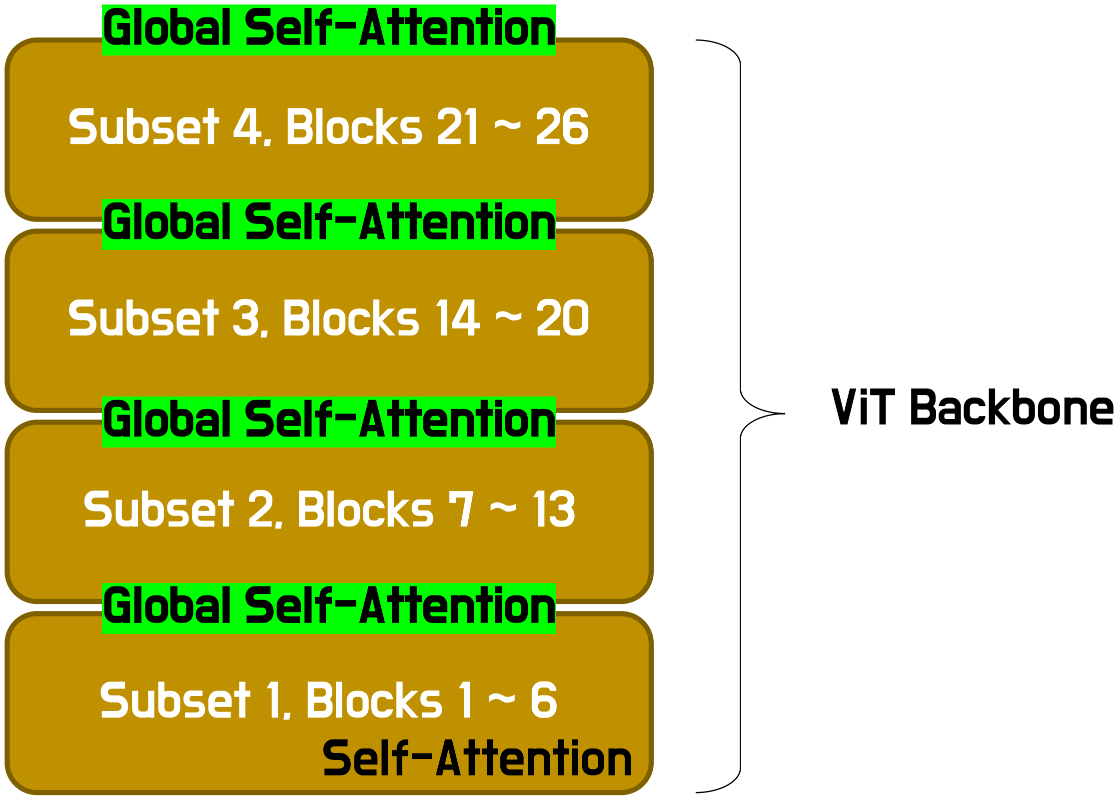
Minimal Adaptation: Global Self-attention with window

→ ViT Backbone의 구조를 4등분하여 subset으로 나누고 각 subset의 마지막에서 attention

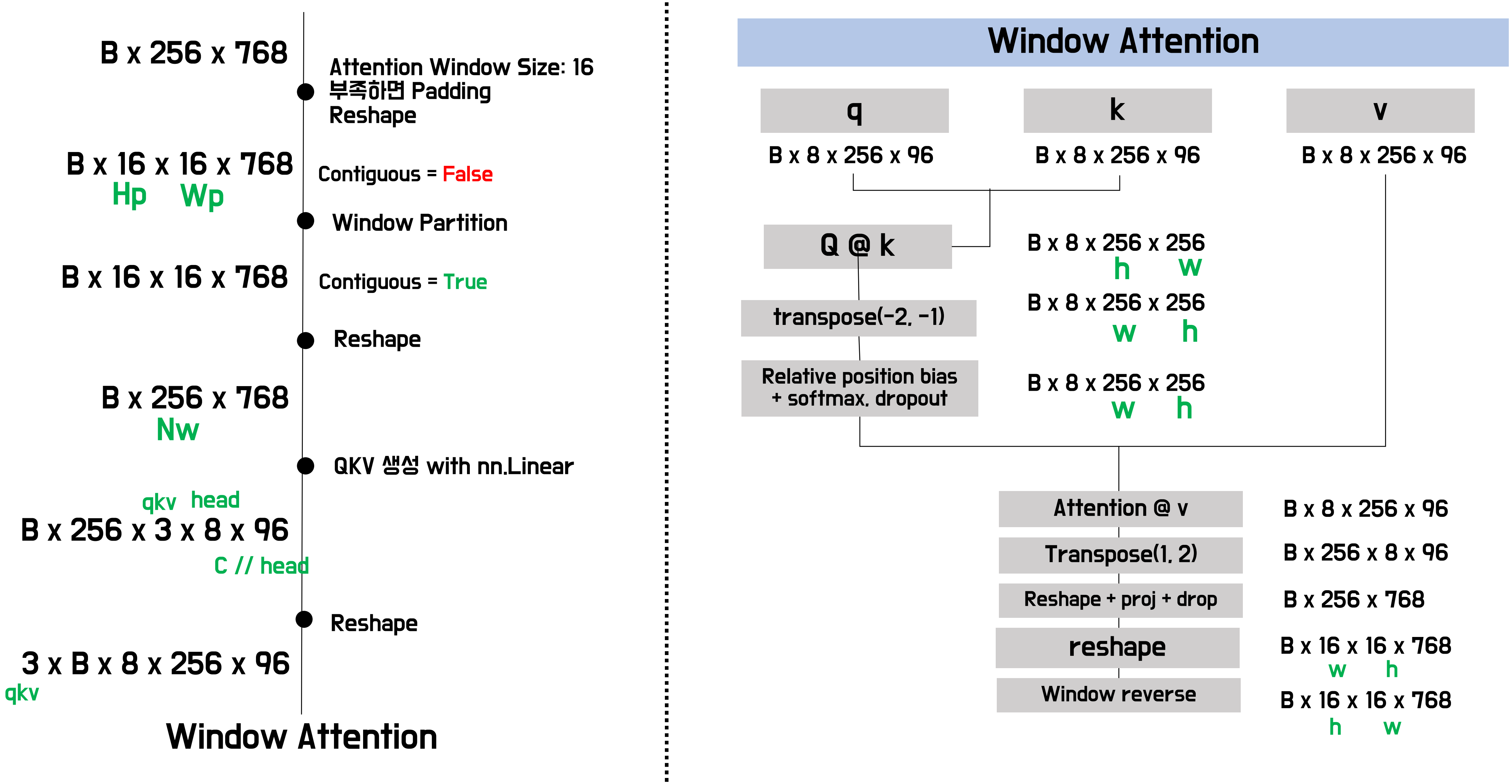
1) Global propagation: Window Attention (hybrid window attention)
2) Convolutional propagation: conv block (with residual connection)

Methodology 2) Backbone Adaptation . Variants

Global Propagation

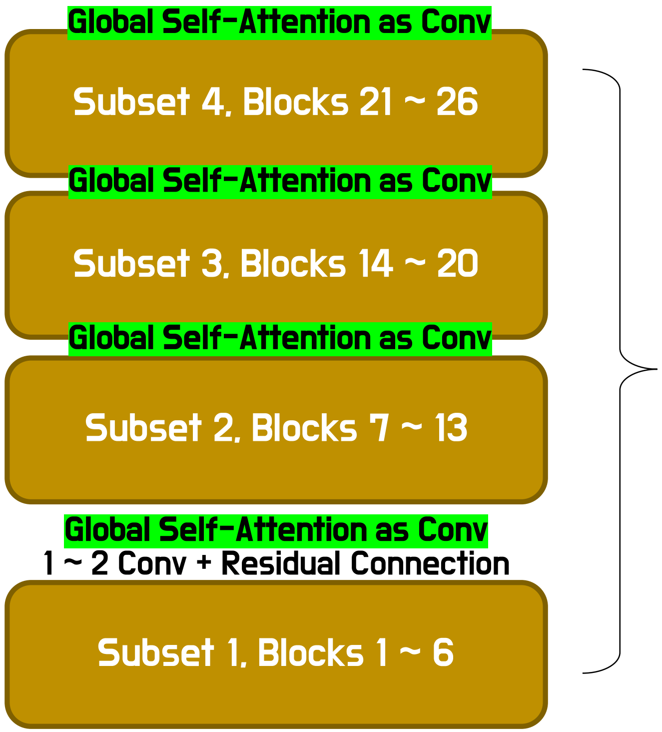


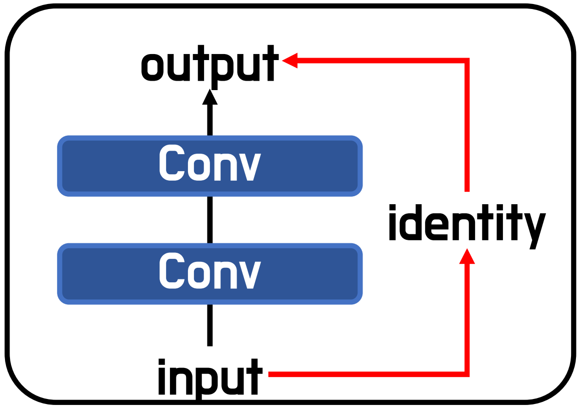
Global Propagation (Global Self-attention, Window Attention without shift)



Methodology 2) Backbone Adaptation . Variants

Convolution Propagation





Global Self-Attention as Conv

Naïve: 3x3
Basic: 3x3 → 3x3
Botleneck: 1x1 → 3x3 → 1x1

prop. conv	AP^{box}	AP^{mask}
none	52.9	47.2
naïve	54.3 (+1.4)	48.3 (+1.1)
basic	**54.8** (+1.9)	**48.8** (+1.6)
bottleneck	54.6 (+1.7)	48.6 (+1.4)

(b) Convolutional propagation with different residual block types (4 blocks).

Methodology 2) Backbone Adaptation . Global vs. Convolution Propagation

prop. strategy	AP^{box}	# params	train mem	test time
none	52.9	1.00× (331M)	1.00× (14.6G)	1.00× (88ms)
4 conv (bottleneck)	54.6 (+1.7)	1.04×	1.05×	1.04×
4 global	54.6 (+1.7)	1.00×	1.39×	1.16×
24 global	55.1 (+2.2)	1.00×	3.34×†	1.86×

매 block마다 global propagation

Plain ViT 기반 Backbone Adaptation에 따른 성능 변화 결과

Train Mem: Batch size 1 기준 Single GPU 소요 메모리

Test Time: A100 GPU 기준 Forward에 소요되는 시간

1. FPN design is not necessary in the case of a plain ViT backbone (Decoupling)

pyramid design	ViT-B AP^box	ViT-B AP^mask	ViT-L AP^box	ViT-L AP^mask
no feature pyramid	47.8	42.5	51.2	45.4
(a) FPN, 4-stage	50.3 (+2.5)	44.9 (+2.4)	54.4 (+3.2)	48.4 (+3.0)
(b) FPN, last-map	50.9 (+3.1)	45.3 (+2.8)	**54.6** (+3.4)	48.5 (+3.1)
(c) simple feature pyramid	**51.2** (+3.4)	**45.5** (+3.0)	**54.6** (+3.4)	**48.6** (+3.2)

Table 1: **Ablation on feature pyramid design** with plain ViT backbones, using Mask R-CNN evaluated on COCO. The backbone is ViT-B (left) and ViT-L (right). The entries (a-c) correspond to Figure 2 (a-c), compared to a baseline without any pyramid. Both FPN and our simple pyramid are substantially better than the baseline, while our simple pyramid is sufficient.

2. Window attention is sufficient as long as information is well propagated across windows in a small number of layers (Large-Scale Training Enable)

prop. strategy	AP^{box}	AP^{mask}
none	52.9	47.2
4 global blocks	54.6 (+1.7)	48.6 (+1.4)
4 conv blocks	**54.8** (+1.9)	**48.8** (+1.6)
shifted win.	54.0 (+1.1)	47.9 (+0.7)

(a) Window attention with various cross-window propagation strategies.

prop. conv	AP^{box}	AP^{mask}
none	52.9	47.2
naïve	54.3 (+1.4)	48.3 (+1.1)
basic	**54.8** (+1.9)	**48.8** (+1.6)
bottleneck	54.6 (+1.7)	48.6 (+1.4)

(b) Convolutional propagation with different residual block types (4 blocks).

prop. locations	AP^{box}	AP^{mask}
none	52.9	47.2
first 4 blocks	52.9 (+0.0)	47.1 (–0.1)
last 4 blocks	54.3 (+1.4)	48.3 (+1.1)
evenly 4 blocks	**54.6** (+1.7)	**48.6** (+1.4)

(c) Locations of cross-window global propagation blocks.

prop. blks	AP^{box}	AP^{mask}
none	52.9	47.2
2	54.4 (+1.5)	48.5 (+1.3)
4	54.6 (+1.7)	48.6 (+1.4)
$24^{\dagger}$	**55.1** (+2.2)	**48.9** (+1.7)

(d) Number of global propagation blocks. †: Memory optimization required.

3. Well-pretrained, fine-tuned plain ViT can outperform the hierarchical-backbone (Swin, Mvit)

4. The trend can be seen in different object detection frameworks, including Mask R-CNN, Cascade Mask R-CNN, and others.

backbone	pre-train	Mask R-CNN AP^{box}	Mask R-CNN AP^{mask}	Cascade Mask R-CNN AP^{box}	Cascade Mask R-CNN AP^{mask}
hierarchical-backbone detectors:					
Swin-B	21K, sup	51.4	45.4	54.0	46.5
Swin-L	21K, sup	52.4	46.2	54.8	47.3
MViTv2-B	21K, sup	53.1	47.4	55.6	48.1
MViTv2-L	21K, sup	53.6	47.5	55.7	48.3
MViTv2-H	21K, sup	54.1	47.7	55.8	48.3
our plain-backbone detectors:					
ViT-B	1K, MAE	51.6	45.9	54.0	46.7
ViT-L	1K, MAE	55.6	49.2	57.6	49.8
ViT-H	1K, MAE	**56.7**	**50.1**	**58.7**	**50.9**

COCO 데이터셋 기준 Detection Performance
Hierarchical vs. Plain Backbone

method	framework	pre-train	single-scale test AP^{box}	single-scale test AP^{mask}	multi-scale test AP^{box}	multi-scale test AP^{mask}
hierarchical-backbone detectors:						
Swin-L [42]	HTC++	21K, sup	57.1	49.5	58.0	50.4
MViTv2-L [34]	Cascade	21K, sup	56.9	48.6	58.7	50.5
MViTv2-H [34]	Cascade	21K, sup	57.1	48.8	58.4	50.1
CBNetV2 [36]†	HTC	21K, sup	59.1	51.0	59.6	51.8
SwinV2-L [41]	HTC++	21K, sup	58.9	51.2	60.2	52.1
plain-backbone detectors:						
UViT-S [9]	Cascade	1K, sup	51.9	44.5	-	-
UViT-B [9]	Cascade	1K, sup	52.5	44.8	-	-
ViTDet, ViT-B	Cascade	1K, MAE	56.0	48.0	57.3	49.4
ViTDet, ViT-L	Cascade	1K, MAE	59.6	51.1	60.4	52.2
ViTDet, ViT-H	Cascade	1K, MAE	**60.4**	**52.0**	61.3	53.1

COCO 데이터셋 기준 Detection Performance
State-of-The-Art Models

+ The trend can be seen in the other dataset

- A Dataset for Large Vocabulary Instance Segmentation (LVIS)

LVIS is a dataset for long tail instance segmentation. It has annotations for over 1000 object categories in 164k images.

method	pre-train	AP^{mask}	AP^{mask}_{rare}	AP^{box}
hierarchical-backbone detectors:				
Copy-Paste [19], Eff-B7 FPN	none (random init)	36.0	29.7	39.2
Detic [58], Swin-B	21K, sup; CLIP	41.7	41.7	-
competition winner 2021 [18] baseline, †	21K, sup	43.1	34.3	-
competition winner 2021 [18] full, †	21K, sup	**49.2**	**45.4**	-
plain-backbone detectors:				
ViTDet, ViT-L	1K, MAE	46.0	34.3	51.2
ViTDet, ViT-H	1K, MAE	48.1	36.9	53.4

Table 7: **System-level comparisons with the leading results on LVIS** (v1 val) reported by the original papers. All results are without test-time augmentation. Detic [58] uses pre-trained CLIP [44] text embeddings. †: these entries use CBNetV2 [36] that combines two Swin-L backbones.

Summary of Findings

1. FPN design is not necessary in the case of a plain ViT backbone (Decoupling)

2. Window attention is sufficient as long as information is well propagated across windows in a small number of layers (Large-Scale Training Enable)

3. Well-pretrained, fine-tuned plain ViT can outperform the hierarchical-backbone (Swin, Mvit)

4. The trend can be seen in different object detection frameworks, including Mask R-CNN, Cascade Mask R-CNN, and others.

+ The trend can be seen in the other dataset

Discussion & Conclusion

- Using Plain ViT as Object Detection's Backbone has benefit from
  1) The detector less has 'inductive bias', as it does not use hierarchical features
  2) The model is Translation equivariance by using plain, non-hierarchical ViT
  3) The model is Scale equivariance by using plain ViT with single-scale feature pyramid

- (Subjective) The detection framework is easy-to-implement, simple, and promising, as it can be combined with various powerful pre-training & fine-tuning design

Background of Paper

다음 논문 주제 예측: 비디오 MAE 기반의 Fine-tuning for 비디오

	TITLE	CITED BY	YEAR
[방법론] SSL for ViT: MAE, 비디오	Masked Autoencoders As Spatiotemporal Learners C Feichtenhofer, H Fan, Y Li, K He arXiv preprint arXiv:2205.09113	14	2022
[방법론] Fine-tuning ViT: 이미지 MAE 기반의 Fine-tuning	Exploring Plain Vision Transformer Backbones for Object Detection Y Li, H Mao, R Girshick, K He European Conference on Computer Vision (ECCV), 2022	29	2022
[방법론] SSL for ViT: MAE, 이미지	Masked Autoencoders Are Scalable Vision Learners K He, X Chen, S Xie, Y Li, P Dollár, R Girshick Computer Vision and Pattern Recognition (CVPR), 2022	529	2022
	Benchmarking Detection Transfer Learning with Vision Transformers Y Li, S Xie, X Chen, P Dollar, K He, R Girshick arXiv preprint arXiv:2111.11429	29	2021
[비교논문] SSL for ViT: 비디오	A Large-Scale Study on Unsupervised Spatiotemporal Representation Learning C Feichtenhofer, H Fan, B Xiong, R Girshick, K He Computer Vision and Pattern Recognition (CVPR), 2021 C[x] CODE	92	2021
[비교논문] SSL for ViT: 이미지	An Empirical Study of Training Self-Supervised Vision Transformers X Chen, S Xie, K He International Conference on Computer Vision (ICCV), 2021	354	2021