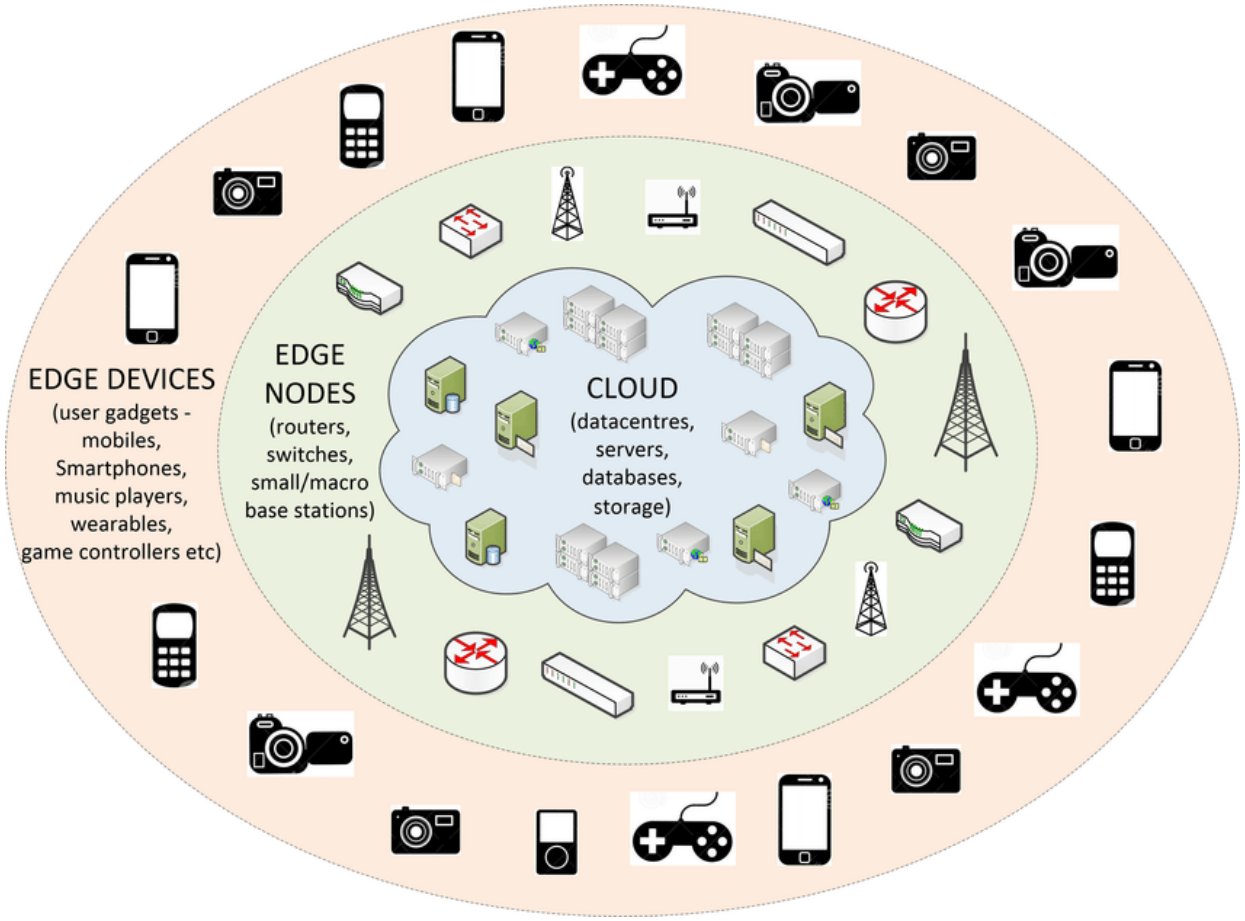
Fast ViT:
A Fast Hybrid VisionTransformer using Structural Reparameterization

Ranjan et al., ICCV2023 Apple (#ofcitation:20)
박지영

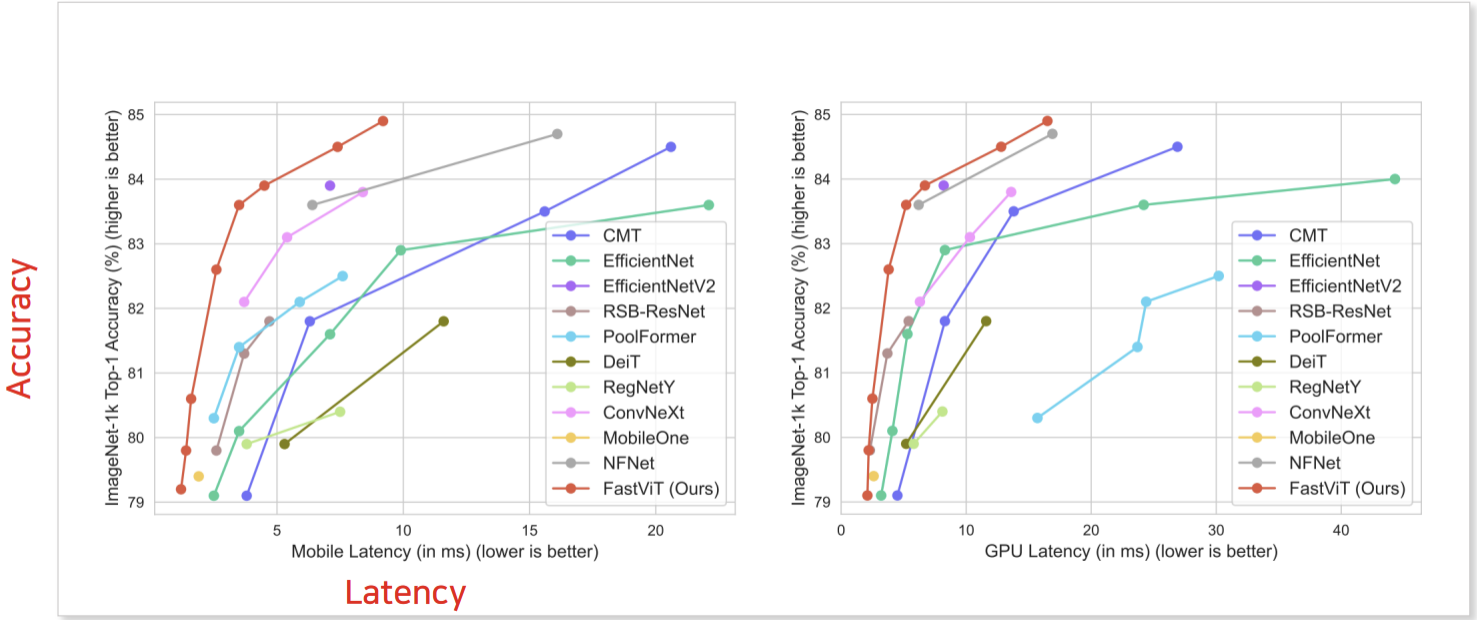
목차

1. Overview



- Vision Transformer들은 image classification및 detection등 다양한 task들에서 SOTA의 성능을 기록함
- 하지만 이런 모델들은 보통 computational cost가 상당히 크기 때문에 실제 업무 환경이나 개인 유저가 활용하기에 어려움이 있음
- Hybrid Vision Transformer를 제안했고 이러한 모델들은 파라미터 수와 연산량이 낮으면서도 경쟁력 있는 성능을 가짐

1. Overview

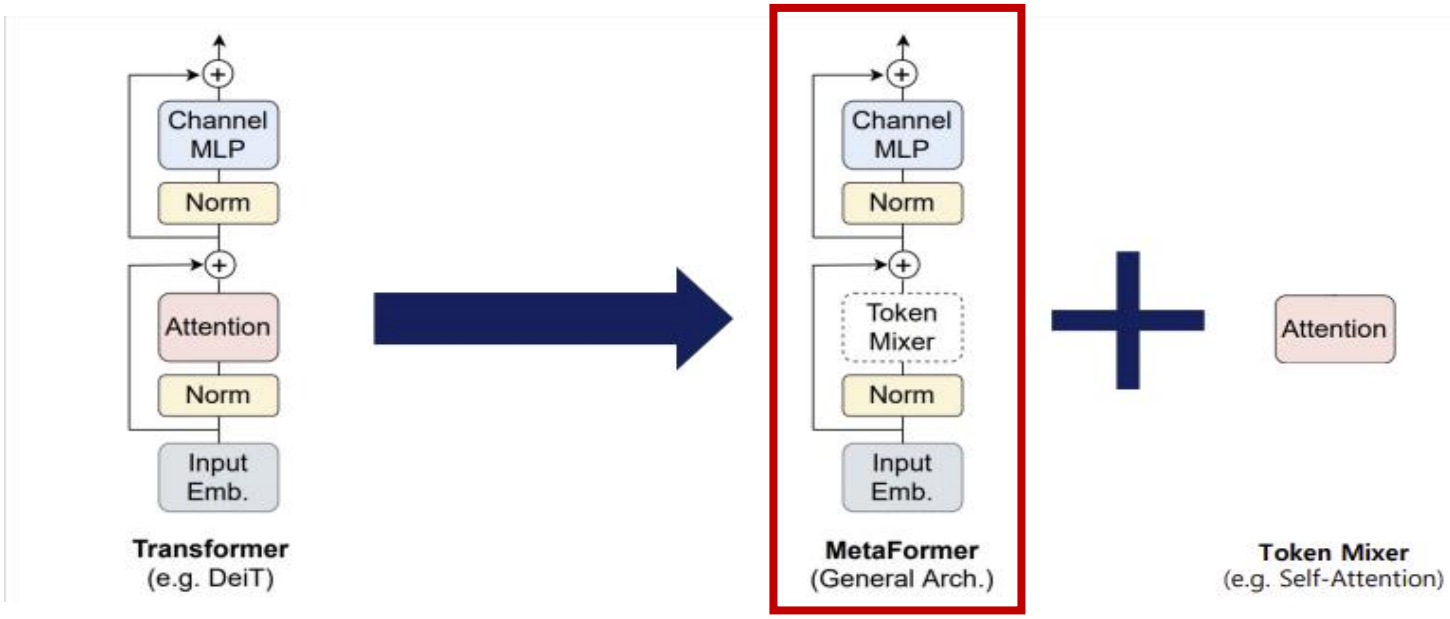


- 논문의 저자는 상대적으로 열악한 edge device환경에선 memory access cost가 latency측면에 악영향을 끼치고 그 주범이 빈번한 메모리 접근을 필요로 하는 skip-connection임을 지적
- 기존에 convolution기반 모델에만 적용되었던 "inference단계에서 skip-connection을 생략하는 방법"인 Reparameterization을 최초로 Hybrid Vision Transformer에 적용하는 FastViT를 제안
- 그 외에도 성능 일부를 포기하고 latency를 크게 감소시키는 방법과 그 보완책을 함께 **제시하여 latency- accuracy 기준 SOTA달성**

2. Background

MetaFormer &TokenMixer

MetaFormer is Actually What You Need for Vision, Yuetal.,CVPR2022,391citation

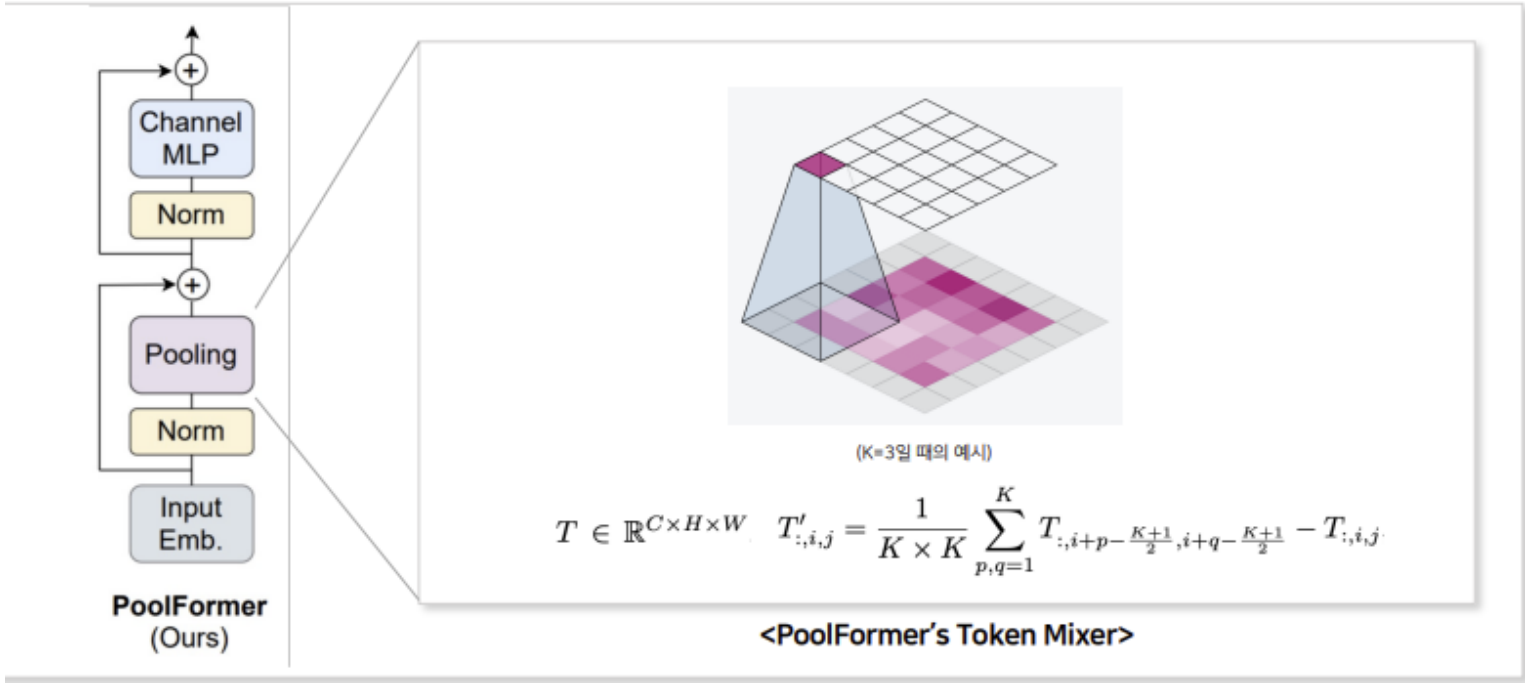


- Vision Transformer의 구조를 좌측 그림과 같이 추상화 했을 때,
- Token Mixer : **patch(token) 간의 정보가 섞이는 부분으로** ViT,DeiT 등의 self-attention이 이에 해당
- Meta Former: **Token Mixer 외의 나머지 구조로** patch embedding, normalization ,skip connection, FFN등이 이에 포함
- 해당 논문 주장의 핵심은 **Vision Transformer의 우수한 성능의 핵심요인은**

Self-Attention으로 대표되는 Token Mixer이 기 보다는 **그 나머지 구조인 Meta Former**

2. Background

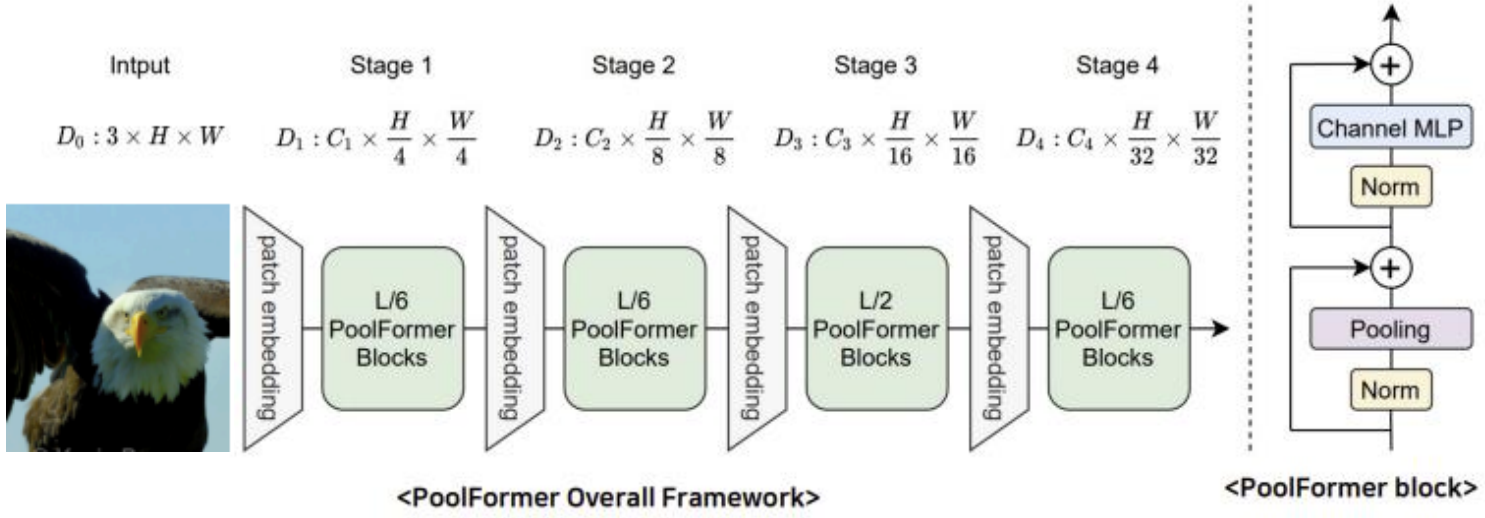
Pool Former block



- 이를 증명하기 위해 ,극단적으로 단순한 Token Mixer를 가진 모듈 "Pool Former block"을 고안
- Self-attention->non-parametric operator, Pooling

2. Background

Pool Former 구조



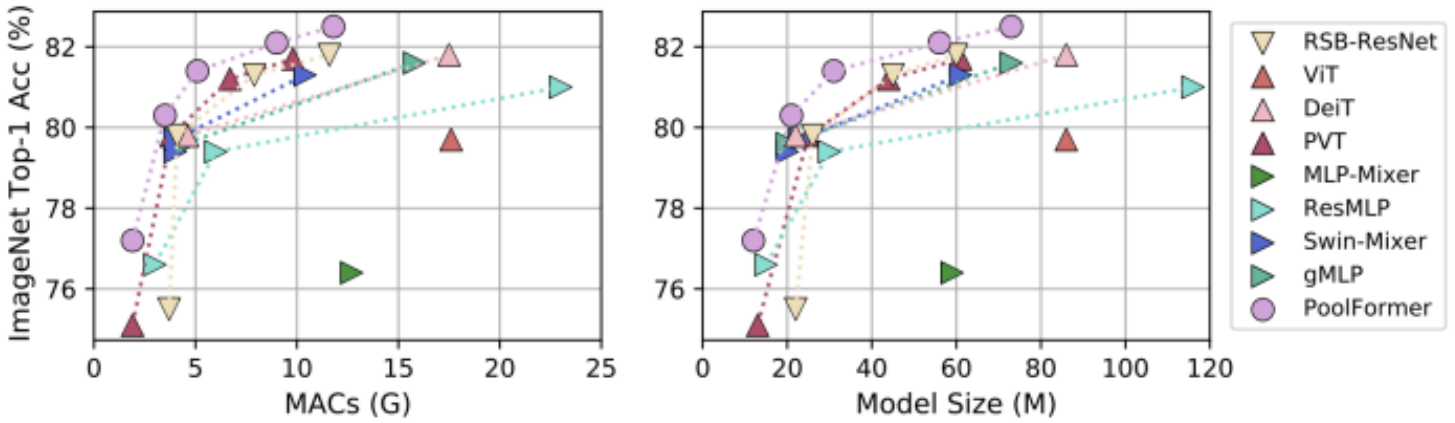
<PoolFormer Overall Framework>

<PoolFormer block>

- PoolFormer는 **네 단계의 MetaFormer로 이루어진 hierarchical 구조**를 채택하여 **MetaFormer의 역할을 극대화함**과 동시에 **가장 단순한 형태의 tokenmixer**를 채택함으로써 **tokenmixer의 역할을 최소화한** 모델
- 각 stage의 patchembedding마다 차원이 축소되고 PoolFormer block에서는 input과 output의 차원이 같은 연산이 수행됨
- 이러한 모델이 ViT,DeiT등 기존의 attention-basedmodel보다 더 좋은 성능을 보인다면 **Token Mixer만큼이나 Meta Former구조의 중요성이 큼을 확인할 수 있음**

2. Background

Pool Former 실험결과 및 결론



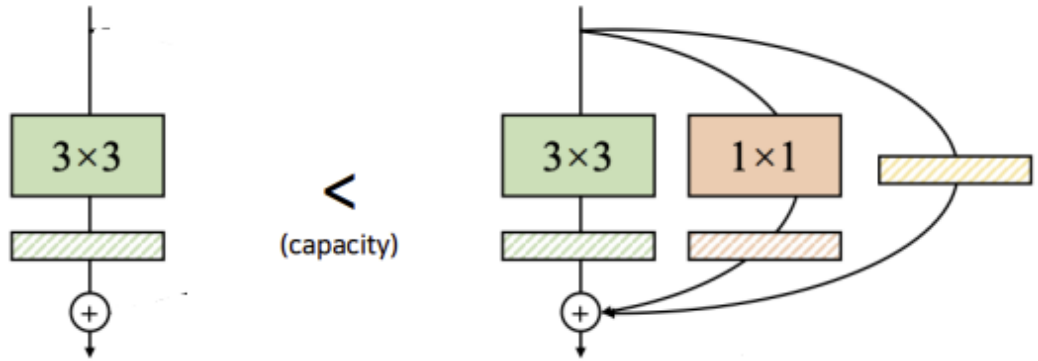
< ImageNet-1K validation accuracy vs. MACs/Model Size >

- 연산량과 모델 크기를 기준으로 봤을 때, **Pool Former는 극단적으로 단순한 Token Mixer**를 가지고도 ViT, DeiT의 성능을 능가했다.
- 이후 많은 논문에서 Token mixer의 개념을 차용했고,
- **FastViT 역시 이러한 개념과 Pool Former 구조의 큰 뼈대를 가져와 사용!**

2. Background

Overparameterization

RepVGG: Making VGG-style ConvNets GreatAgain,Dingetal.,CVPR2021,778citation

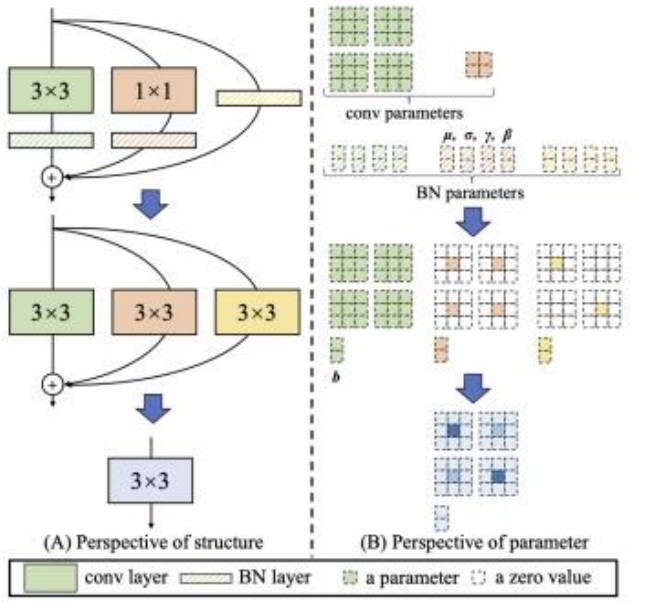


< 3x3 convolution, 1x1 convolution과 skip-connection으로 구성된 Overparameterization >

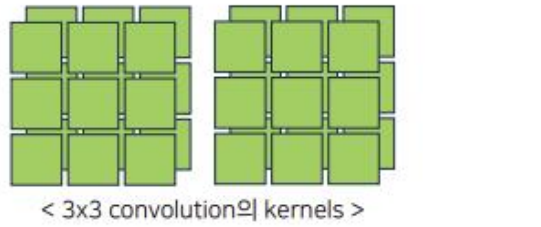
- 다양한 kernel size를 갖는 복수의 convolution연산을 병렬적으로 연결하는 것으로 각각의 연산결과를 summation하여 사용
- 각 convolution연산결과 및 skip-connection의 dimension이 일치해야 한다.
- 이와 같은 방식을 채택함으로써, **layer의 capacity를 향상시킬** 수 있다. +단점 : 학습시간 증가

2. Background

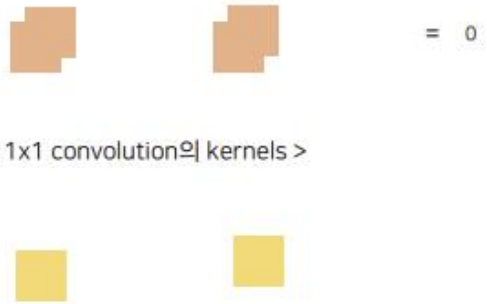
Reparameterization- convolution & skip-connection



< input C=2, output C=2일 때의 Reparameterization 과정 >



< 3x3 convolution의 kernels >



< 1x1 convolution의 kernels >

< 1x1 convolution로 표현한 skip-connection >

- 병렬적으로 연결된 여러 개의 convolution연산들(+BN)과 skip-connection을

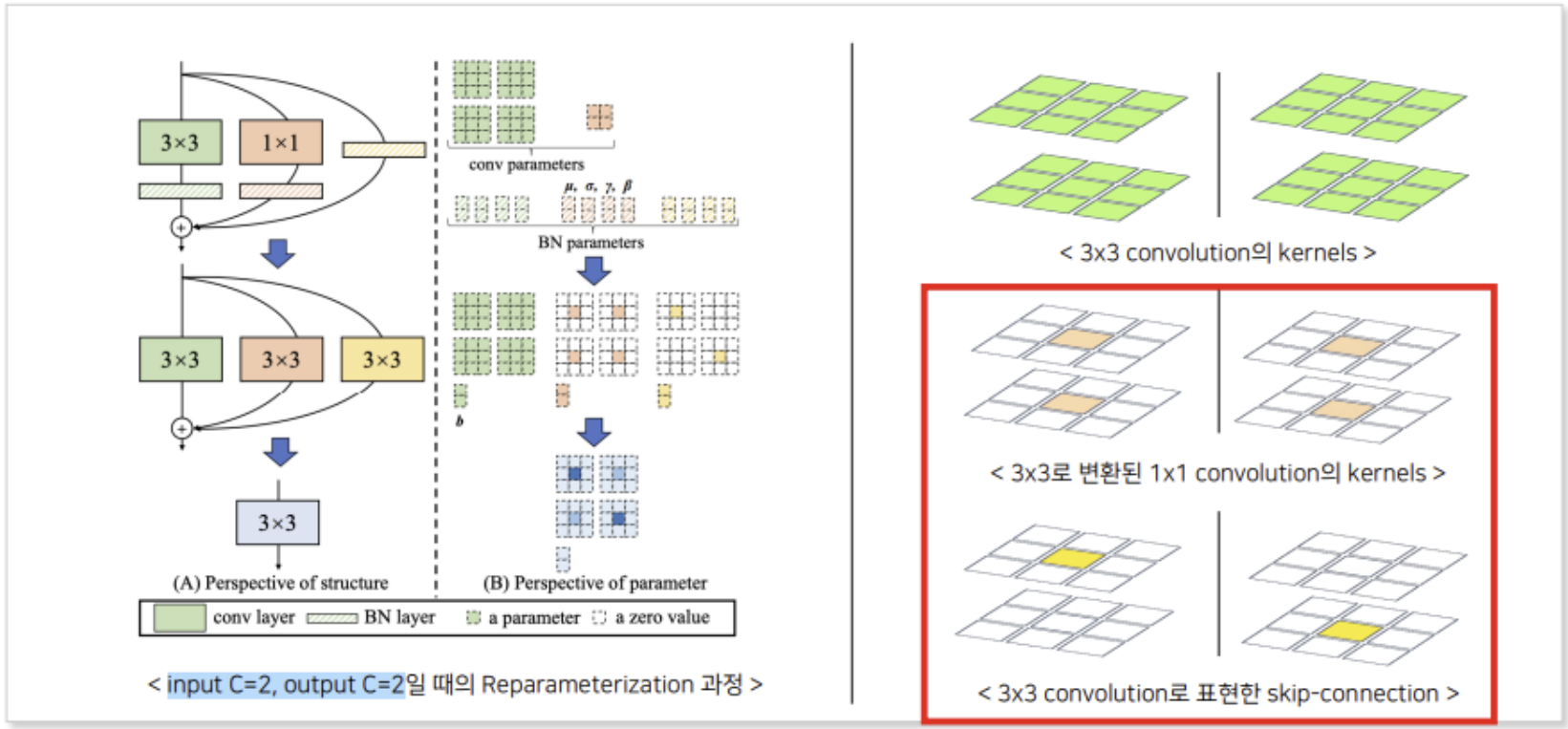
  연산 결과가 같은 하나의 convolution으로 치환하는 방법으로 Inference단계에서 적용 가능하다.

- Reparameterization하면 **원래의 연산과 동일한 결과를 하나의 convolution연산으로** 얻을 수 있다.

  연산량 감소(conv가 하나이므로),memoryaccesscost 감소(skip-connection이 없으므로)

2. Background

Reparameterization- convolution & skip-connection

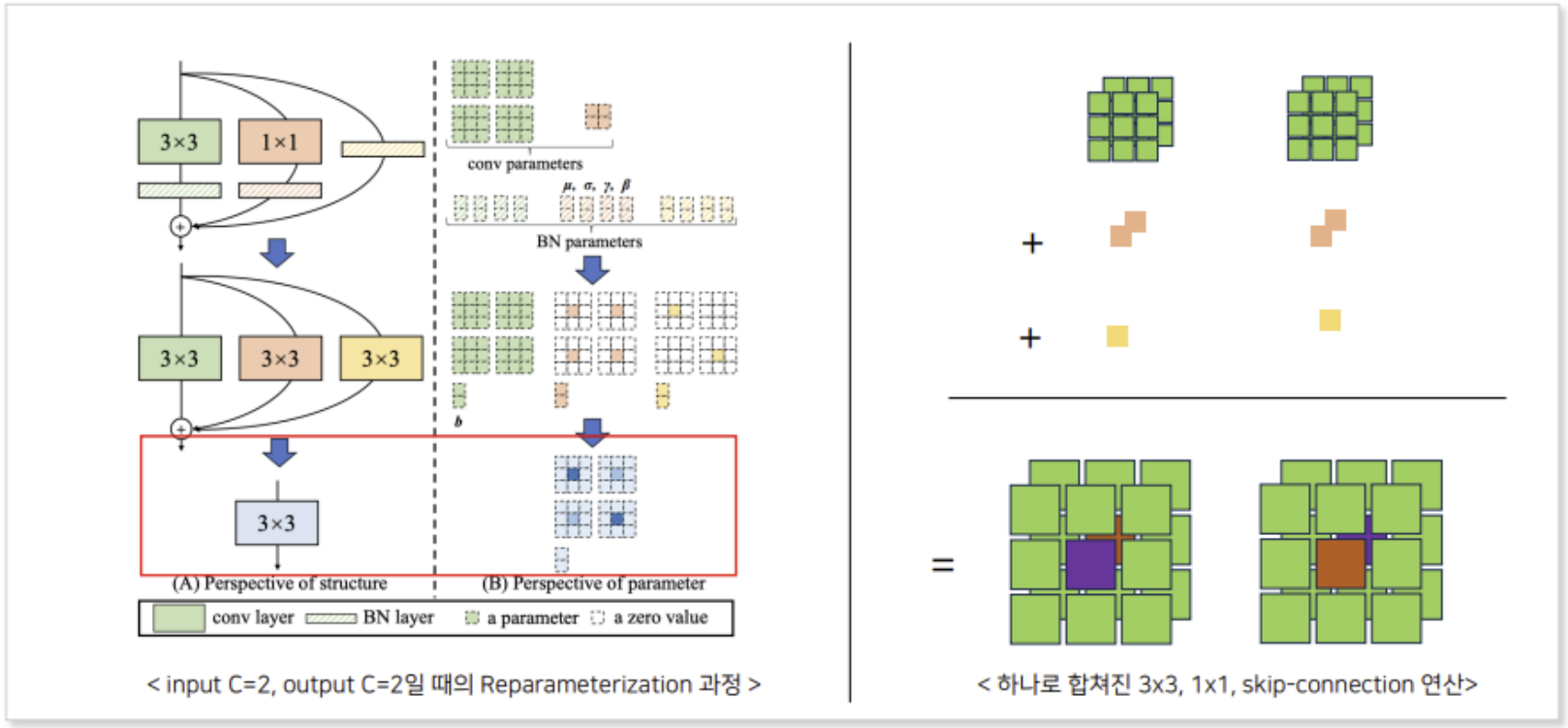


1x1 convolution 연산은 주변부의 weight가 0인 3x3 convolution으로 변환될 수 있다.

skip-connection 역시 원하는 size의 convolution 연산으로 표한할 수 있다.

2. Background

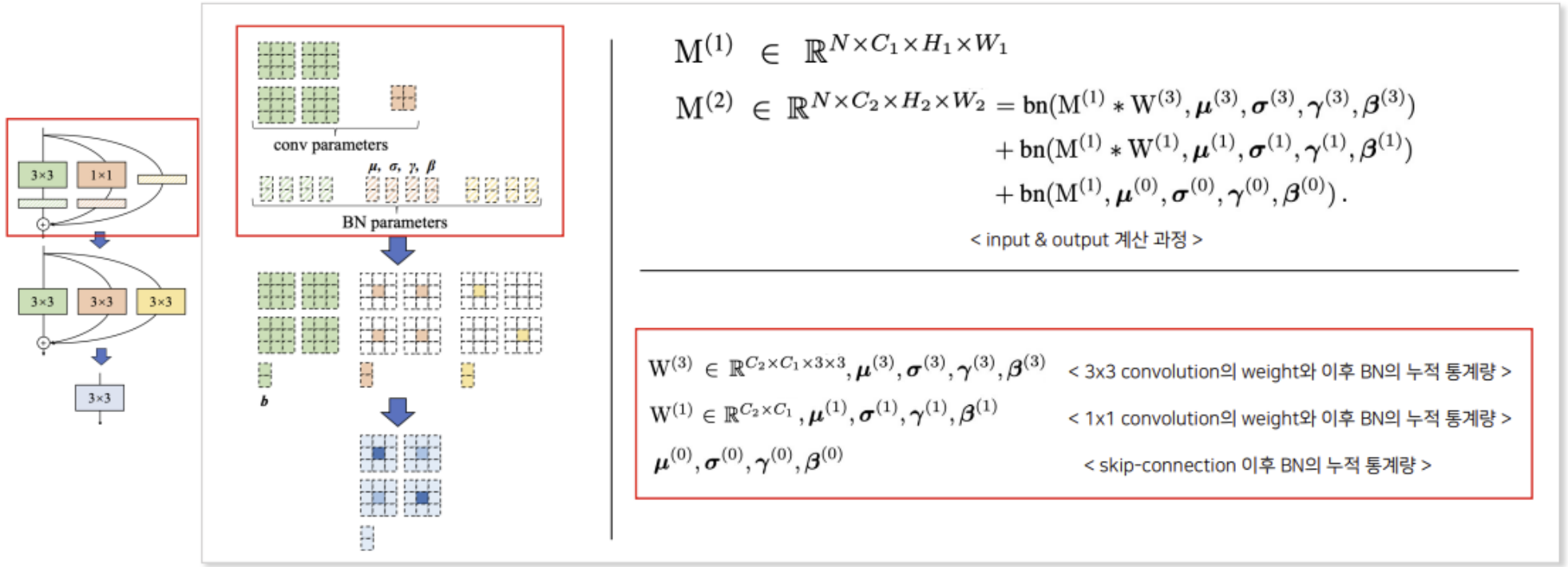
Reparameterization- convolution & skip-connection



- Inference시, **동일한 kernelsize, stride를 갖는 convolution들을 하나의 convolution연산으로 결합하여 사용**

2. Background

Reparameterization batchnormalization



$$M^{(1)} \in \mathbb{R}^{N \times C_1 \times H_1 \times W_1}$$

$$M^{(2)} \in \mathbb{R}^{N \times C_2 \times H_2 \times W_2} = \mathrm{bn}(M^{(1)} * W^{(3)}, \mu^{(3)}, \sigma^{(3)}, \gamma^{(3)}, \beta^{(3)}) + \mathrm{bn}(M^{(1)} * W^{(1)}, \mu^{(1)}, \sigma^{(1)}, \gamma^{(1)}, \beta^{(1)}) + \mathrm{bn}(M^{(1)}, \mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}).$$

< input & output 계산 과정 >

$W^{(3)} \in \mathbb{R}^{C_2 \times C_1 \times 3 \times 3}, \mu^{(3)}, \sigma^{(3)}, \gamma^{(3)}, \beta^{(3)}$ < 3x3 convolution의 weight와 이후 BN의 누적 통계량 >

$W^{(1)} \in \mathbb{R}^{C_2 \times C_1}, \mu^{(1)}, \sigma^{(1)}, \gamma^{(1)}, \beta^{(1)}$ < 1x1 convolution의 weight와 이후 BN의 누적 통계량 >

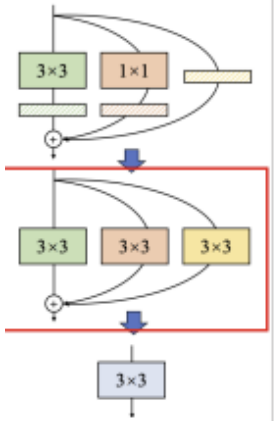
$\mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}$ < skip-connection 이후 BN의 누적 통계량 >

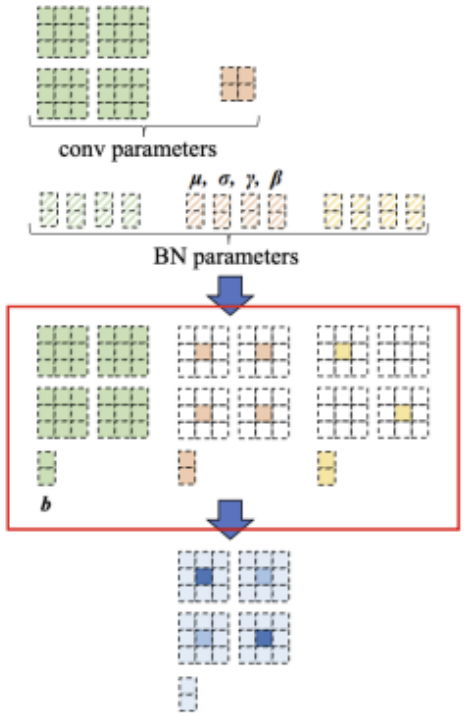
- "Convolution과 그 convolution에 이어지는 BN" 역시 **하나의 convolution으로 reparameterize**할 수 있다.
- 학습 과정 동안 업데이트된 convolution의 weight와 BN의 누적 통계량을 사용한다.
- $\mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}$ are each the accumulated mean, standard deviation and learned scaling factor and bias of the BNl ayer.

-RepVGG 원문 발췌- Tip:pytorch의 BN은 inference시, BatchNormalization의 원래 의미와 다르게 누적된 통계량을 고정적으로 이용한다.

2. Background

Reparameterization batch normalization





$$\mathrm{M}^{(1)} \in \mathbb{R}^{N \times C_1 \times H_1 \times W_1}$$

$$\begin{aligned}\mathrm{M}^{(2)} \in \mathbb{R}^{N \times C_2 \times H_2 \times W_2} = {} & \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(3)}, \boldsymbol{\mu}^{(3)}, \boldsymbol{\sigma}^{(3)}, \boldsymbol{\gamma}^{(3)}, \boldsymbol{\beta}^{(3)}) \\ & + \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(1)}, \boldsymbol{\mu}^{(1)}, \boldsymbol{\sigma}^{(1)}, \boldsymbol{\gamma}^{(1)}, \boldsymbol{\beta}^{(1)}) \\ & + \mathrm{bn}(\mathrm{M}^{(1)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}, \boldsymbol{\gamma}^{(0)}, \boldsymbol{\beta}^{(0)}) .\end{aligned}$$

< input & output 계산 과정 >

$$\mathrm{bn}(\mathrm{M}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta})_{:,i,:,:} = (\mathrm{M}_{:,i,:,:} - \boldsymbol{\mu}_i)\frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i$$

< Batch Normalization의 수식 >

$$\mathrm{W}'_{i,:,:,:} = \frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i}\mathrm{W}_{i,:,:,:}, \quad \mathrm{b}'_i = -\frac{\boldsymbol{\mu}_i \boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i$$

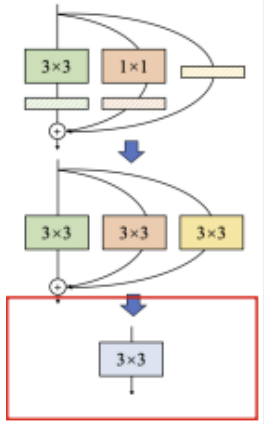
$$\mathrm{bn}(\mathrm{M} * \mathrm{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta})_{:,i,:,:} = (\mathrm{M} * \mathrm{W}')_{:,i,:,:} + \mathrm{b}'_i$$

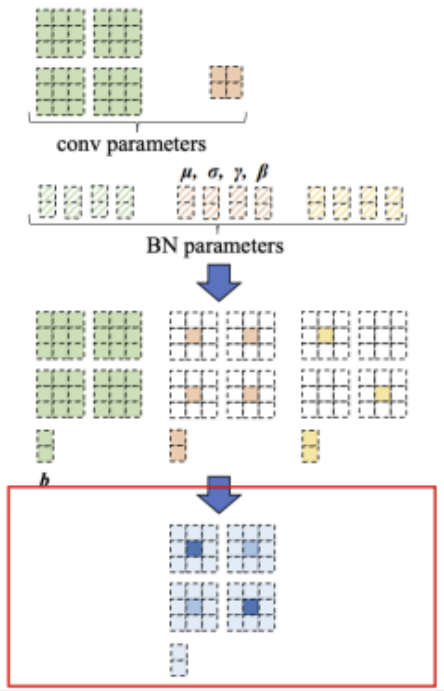
< Convolution + BN -> Convolution with updated weight and bias >

- 네 개의 통계량이 고정된 상황에서, **Conv과 그에 이어지는 BN은 수식 전개를 통해 bias를 가진 conv연산으로 치환**할 수 있다.

2. Background

Reparameterization batch normalization





$$\mathrm{M}^{(1)} \in \mathbb{R}^{N \times C_1 \times H_1 \times W_1}$$

$$\begin{aligned}\mathrm{M}^{(2)} \in \mathbb{R}^{N \times C_2 \times H_2 \times W_2} &= \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(3)}, \boldsymbol{\mu}^{(3)}, \boldsymbol{\sigma}^{(3)}, \boldsymbol{\gamma}^{(3)}, \boldsymbol{\beta}^{(3)}) \\ &+ \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(1)}, \boldsymbol{\mu}^{(1)}, \boldsymbol{\sigma}^{(1)}, \boldsymbol{\gamma}^{(1)}, \boldsymbol{\beta}^{(1)}) \\ &+ \mathrm{bn}(\mathrm{M}^{(1)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}, \boldsymbol{\gamma}^{(0)}, \boldsymbol{\beta}^{(0)}) .\end{aligned}$$

< input & output 계산 과정 >

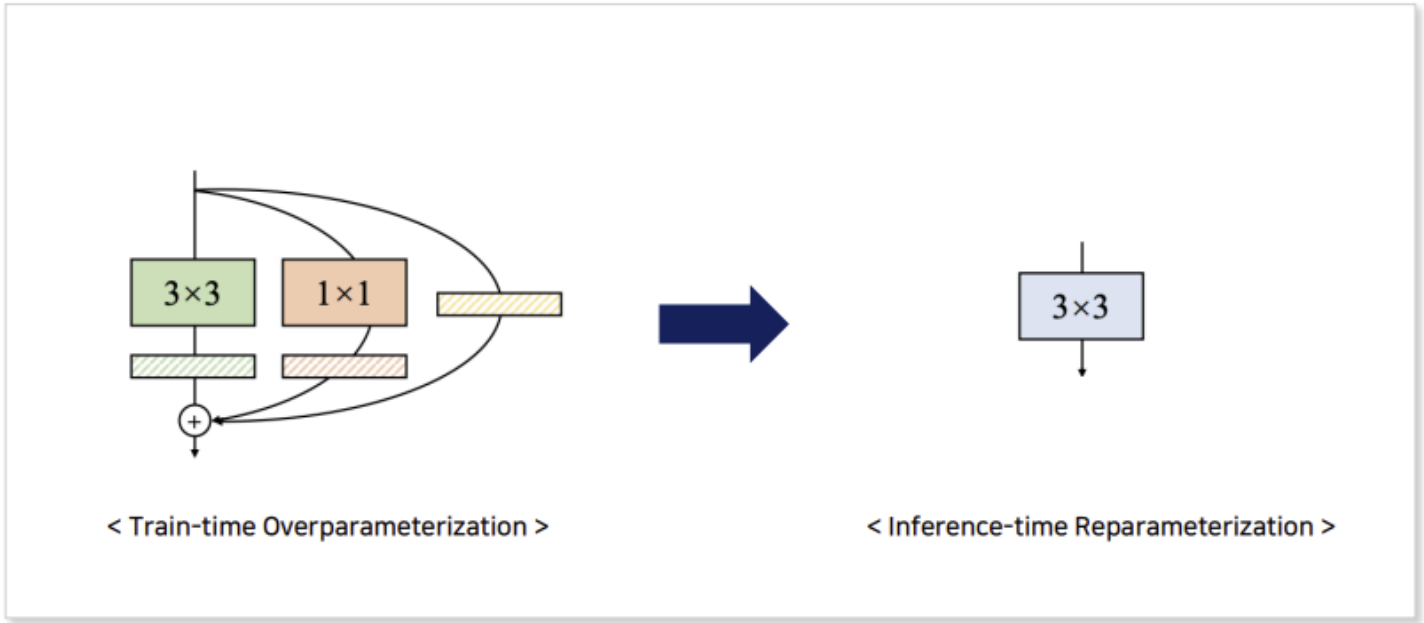
$$\begin{aligned}\mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(3)}, \mu^{(3)}, \sigma^{(3)}, \gamma^{(3)}, \beta^{(3)}) &= \mathrm{M}^{(1)} * \mathrm{W}'^{(3)} + \mathrm{b}'^{(3)} \\ \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(1)}, \mu^{(1)}, \sigma^{(1)}, \gamma^{(1)}, \beta^{(1)}) &= \mathrm{M}^{(1)} * \mathrm{W}'^{(1)} + \mathrm{b}'^{(1)} \\ \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(0)}, \mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}) &= \mathrm{M}^{(1)} * \mathrm{W}'^{(0)} + \mathrm{b}'^{(0)}\end{aligned}$$

$$\begin{aligned}\mathrm{M}^{(2)} &= \mathrm{M}^{(1)} * (\mathrm{W}'^{(3)} + \mathrm{W}'^{(1)} + \mathrm{W}'^{(0)}) + \mathrm{b}'^{(3)} + \mathrm{b}'^{(1)} + \mathrm{b}'^{(0)} \\ \mathrm{M}^{(2)} &= \mathrm{M}^{(1)} * \mathrm{W}^{*} + \mathrm{b}^{*}\end{aligned}$$

- convolution 연산과 skip-connection은 **linear연산이기 때문에 하나의 연산으로 결합**할 수 있다.
- **같은 kernelsize를 갖는 convolution으로 치환된 3x3conv, 1x1conv, skip-connection**을 결합한다.
- 결합은 단순하게 weight끼리,bias끼리 더하는 방식으로 이루어진다.

2. Background

Overparameterization &R eparameterization 정리



- 학습 시에, **Overparameterization을 통해 network의 capacity**를 향상시킨다.
- Inference시에,모든 Convolution연산과 BN, skip-connection을 **하나의 convolution연산으로 Reparameterize**한다.
- **Overparameterization의 이점과 skip-connection의 효과를 유지한 채, 연산량과 memoryaccesscost 감소의 이점을 얻는다.**

3. Proposed method

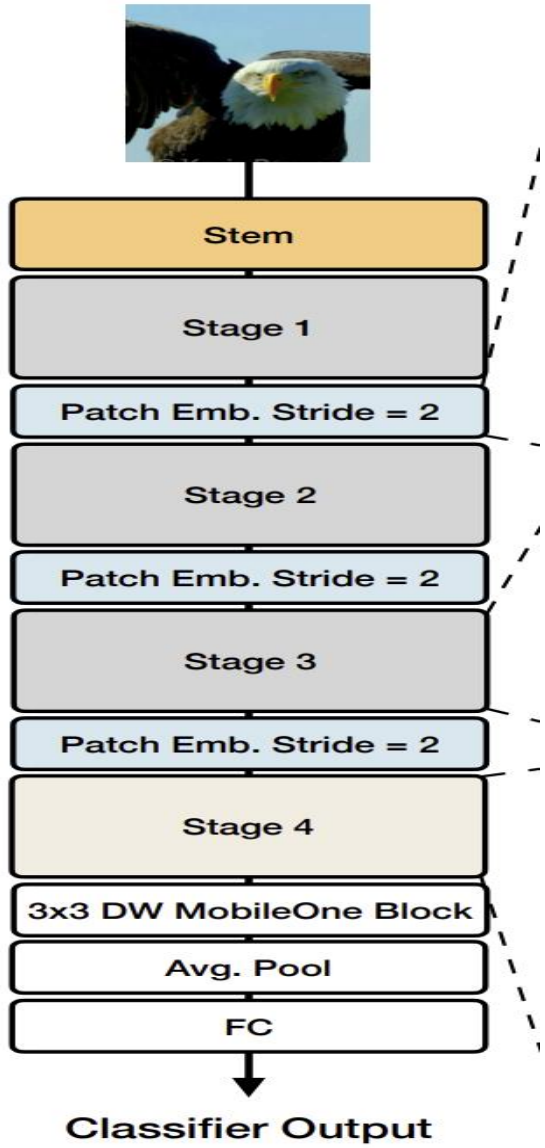
3. Proposed Method

FastViT의 탄생목적

- 실제 **edge device에서의 latency를 최대한 줄이고자** 하는게 논문의 목표
- 저자들은 파라미터 수나 연산량만큼이나 **메모리에 접근할 때 발생하는 memory access cost가 latency를 증가시키는 주 요인**이고

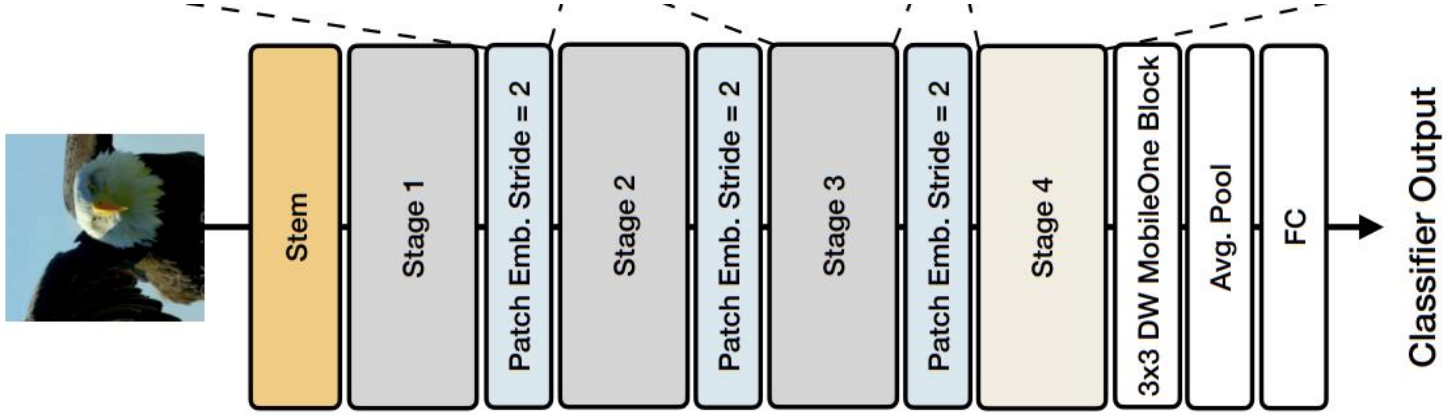
  skip-connection이 memory access cost가

  매우 크다는 점을 지적하며 이를 개선하고자 함
- iPhone이나 desktop grade GPU등 edge device에서 **SOTA latency-accuracy를 갖는 FastViT**를 제안



3. Proposed Method

FastViT의 핵심 Point



Latency감소 목적

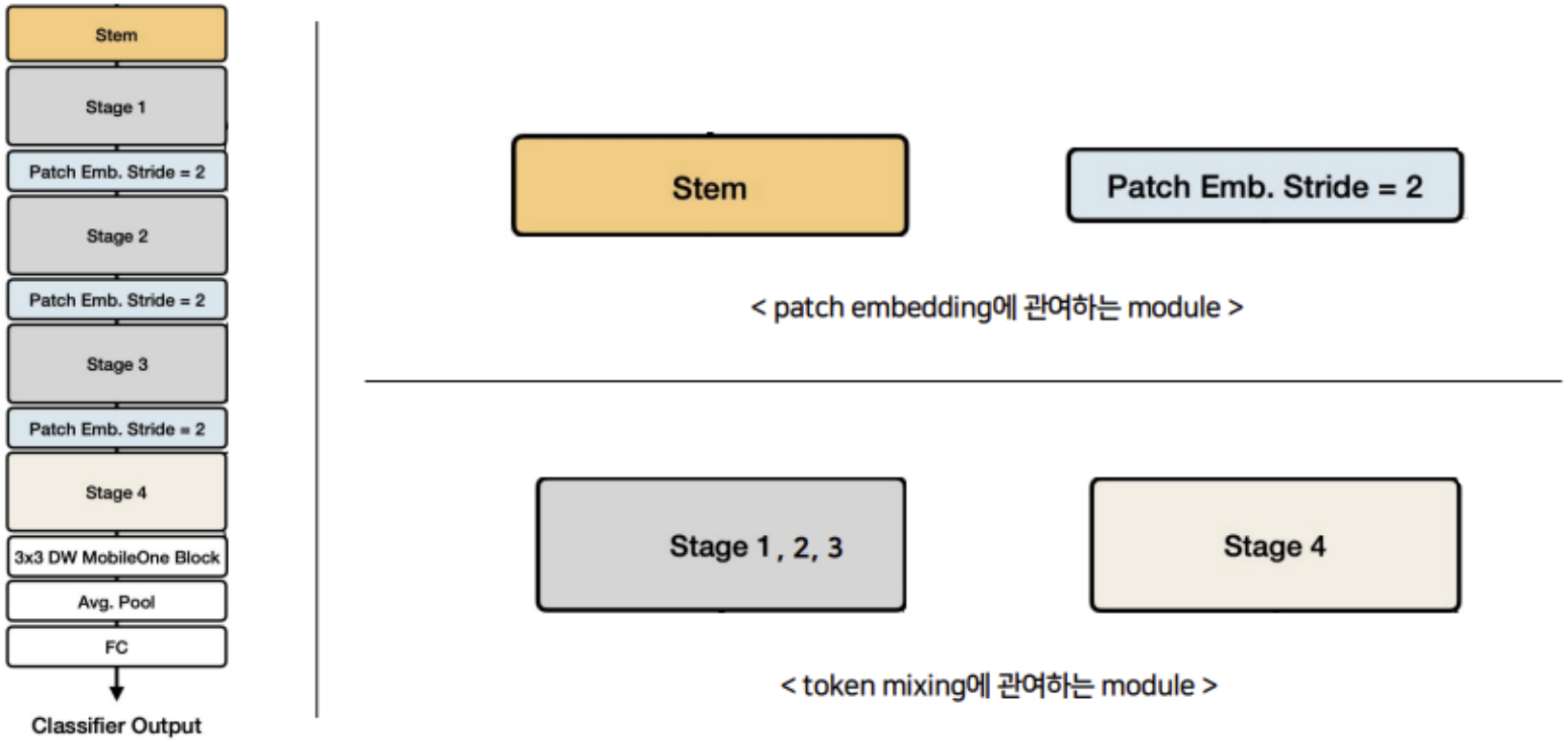
Depthwise Seperable Convolution
RepMixer
Reparameterization

Capacity및 Receptivefield회복 목적 (부작용 보완)

Train-time Overparameterization
Large Kernel Convolution

3. Proposed Method

FastViT 구조



FastViT의 구조는 크게 **patch embedding에 관여하는 부분과 token mixing에 관여하는 부분**,두 개로 나눌 수 있음

3. Proposed Method

FastViT 구조 Patch embedding Part

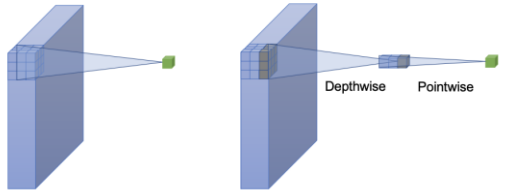
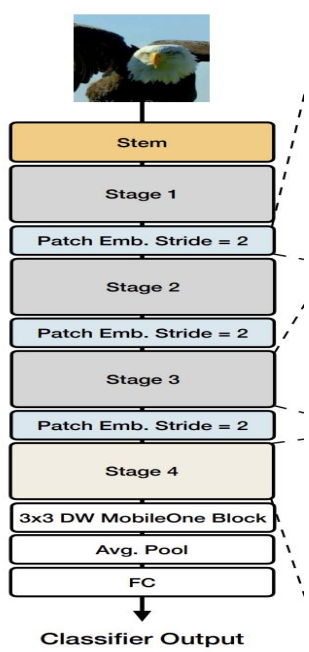
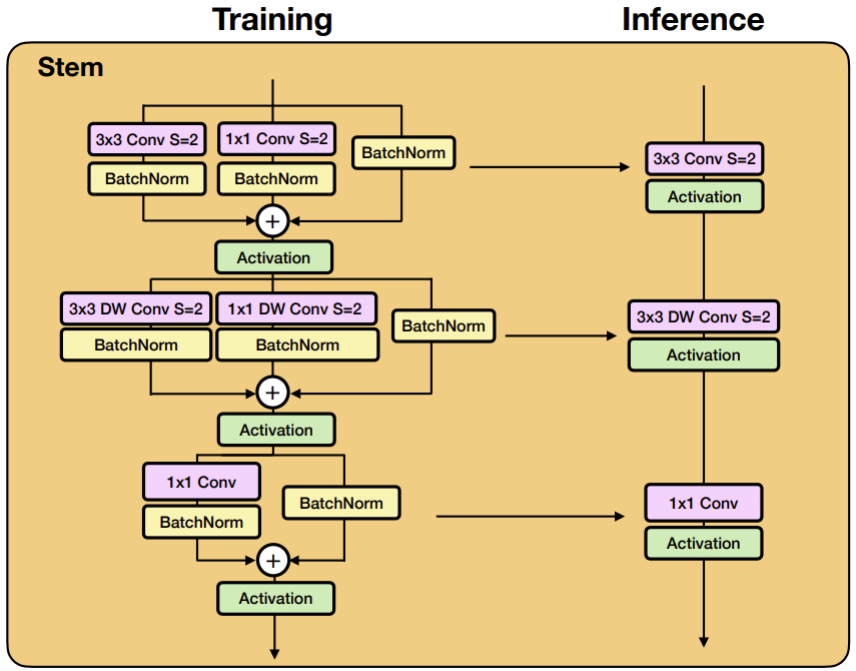
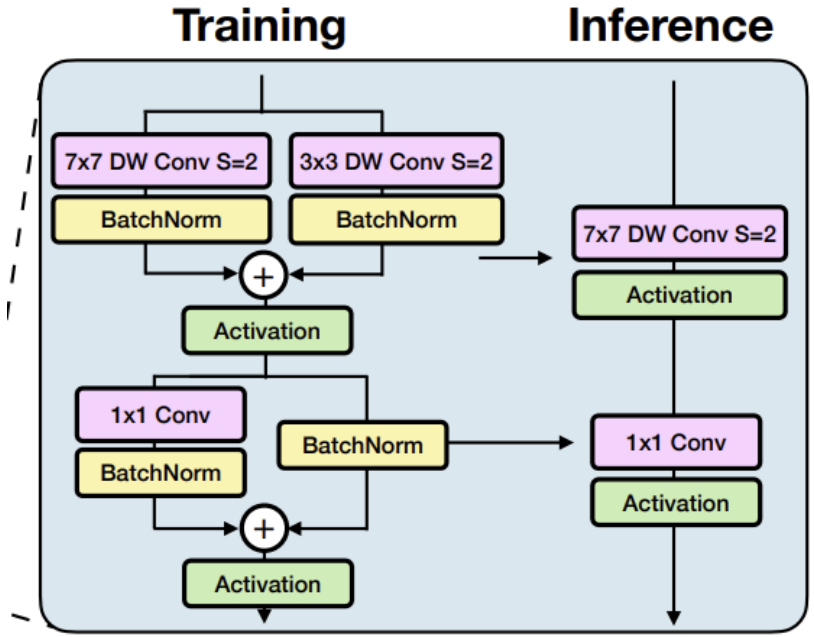


Figure 3: Standard convolution and depthwise separable convolution.





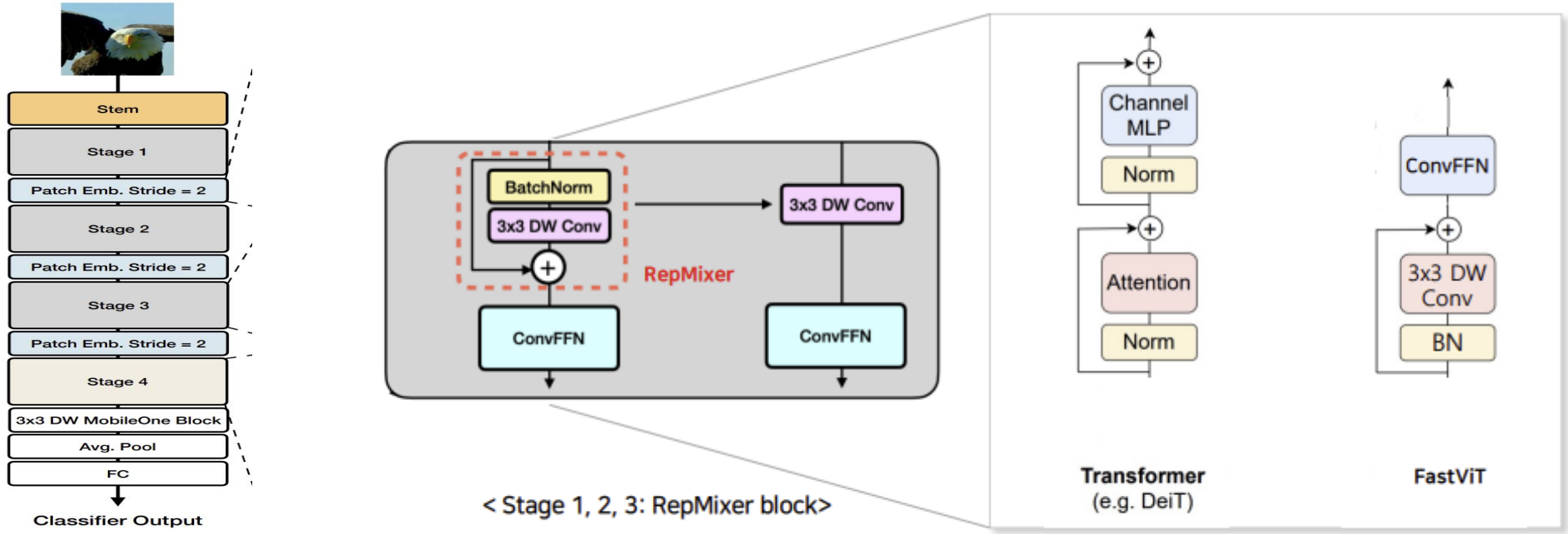
(b)



- FastViT는 Pool Former의 hierarchical구조를 채택했기 때문에 **각 Stage 시작 시 수행되는 patch embedding에서 순차적으로 height와 width가 감소하고 embedding dimension이 증가한다.**
- 첫 번째 patch embedding인 Stem에서는 보다 많은 차원축소가 수행, **핵심정보를 추출하는데 도움 + 동시에 이후 연산량을 줄여준다.**
- 두 경우 **모두에서 연산량을 줄여주는 depthwise seperable convolution을 적극적으로 활용하며 이에 따른 capacity감소를 보상해주는 overparameterization이 적용**,최종적으로 inference시엔 reparameterization이 수행된다.

3. Proposed Method

FastViT 구조 Token mixing Part(Stage1,2,3)



- FastViT는 reparameterization이 가능한 단순한 형태의 tokenmixer인 RepMixer를 제안
- RepMixer는 **단순한 3x3Depthwise Convolution.**
- RepMixer의 BN과 skip-connection역시 **inference단계에서 reparameterize**된다.

3. Proposed Method

RepMixer

- σ : non-linear activation function
- BN : Batch Normalization
- DWConv : depthwise convolution layer

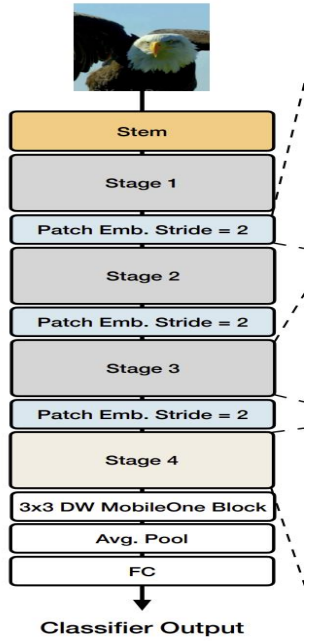
$$Y = \mathrm{BN}(\sigma(\mathrm{DWConv}(X))) + X \tag{1}$$

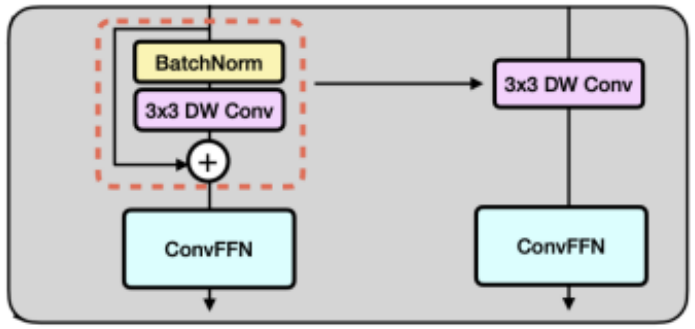
$$Y = \mathrm{DWConv}(\mathrm{BN}(X) + X \tag{2}$$

장점:
추론 시점에 **single depthwise convolution**로 reparameterize할 수 있게 되어 **연산량이 줄어든다**

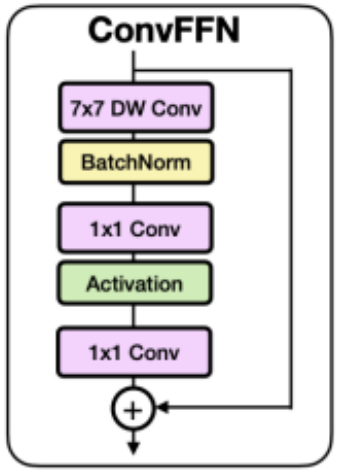
3. Proposed Method

FastViT 구조 Token mixing Part (Stage 1,2,3)





< Stage 1, 2, 3: RepMixer block>

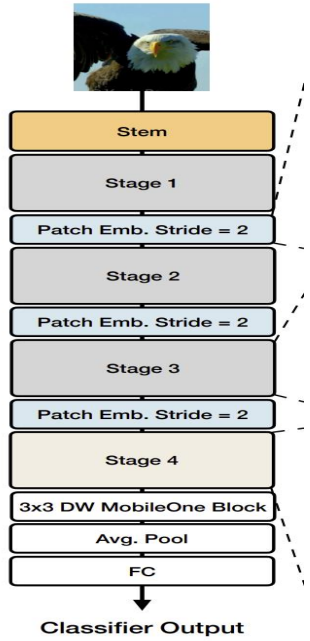


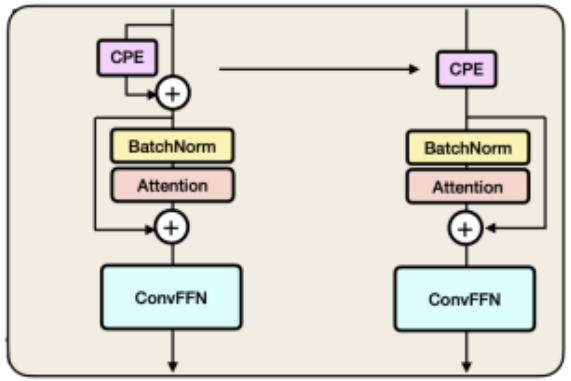
< ConvFFN >

- LocalViT, CeiT, PVT등에서 사용하는 **ConvFFN**을 채택,이는 모델의 **accuracy와 robustness를 향상시킴**
- ConvFFN은 **Transformer의 Feed Forward Network의 역할에 해당하고 DW convolution을 추가한 형태,** 하나의 depthwise conv와 2개의 1x1 pointwise conv로 구성된다.
- ConvFFN 중 7x7DWConv+BN만 reparameterize된다.

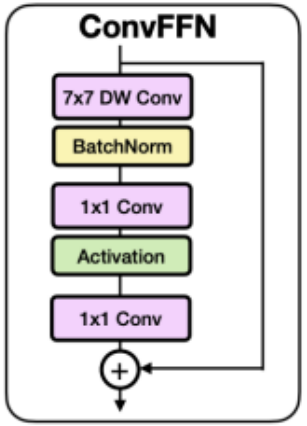
3. Proposed Method

FastViT 구조 TokenmixingPart(Stage4)





< Stage 4: Self-attention block >



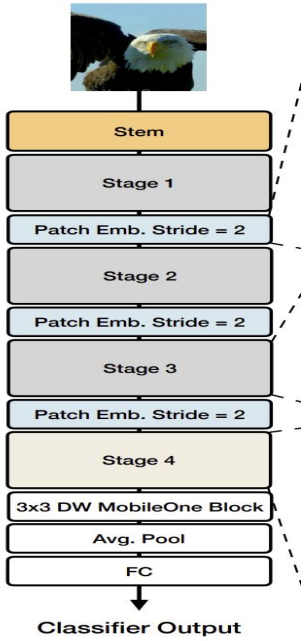
< ConvFFN >

- 저자들은 latency와 accuracy의 trade-off를 고려하여

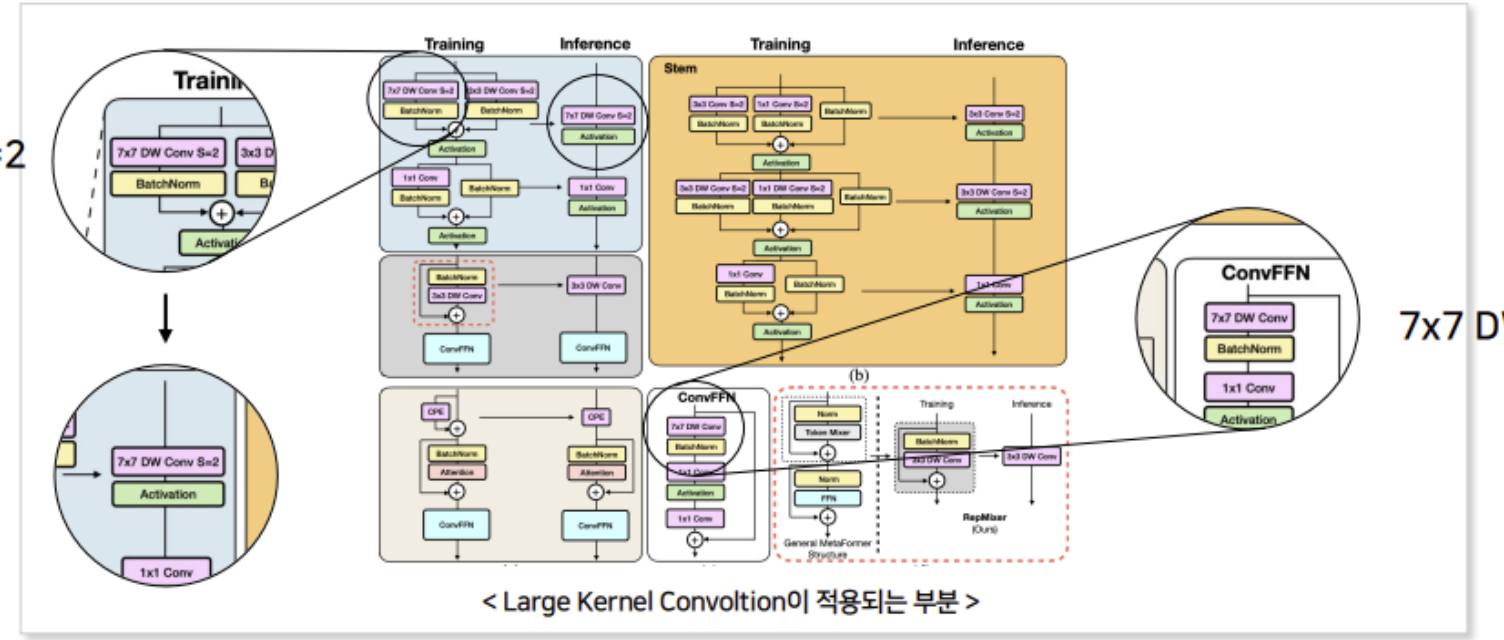
  후반 Stage두 개(3,4)와 Stage하나(4)의 token mixing을 self-attention으로 교체하는 ablation실험들을 실시하였다.
- 실험 결과, **경미한 latency상승 대비 accuracy gain이 컸던 setting (Rep, Rep, Rep, SA)을 최종 모델로 제시**
- Conditional Positional Encoding은 DW convolution을 통해 수행되므로 이 역시 reparameterize 할 수 있다.

3. Proposed Method

Large Kernel Convolution



7x7 DW Conv S=2

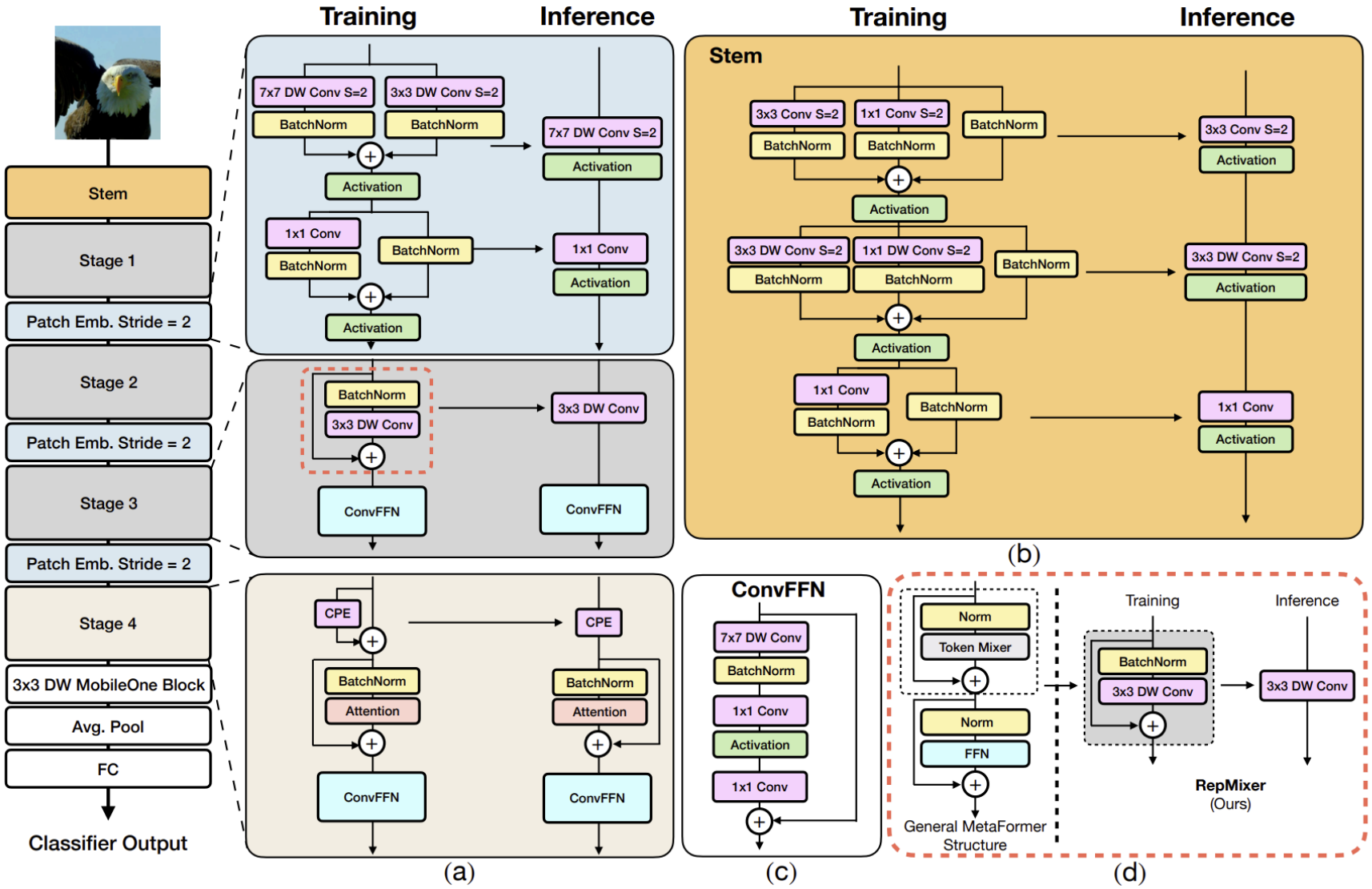


7x7 DW Conv

- RepMixer의 convolution-based token mixing은 연산에 있어서는 효율적이지만
  self-attention 기반에 비하여 상대적으로 local한 receptive field를 갖는다.
- 이를 보완하기 위해
  Patch embedding과 ConvFFN 초반부에 **7x7의 Large Kernel Convolution**을 이용한다.

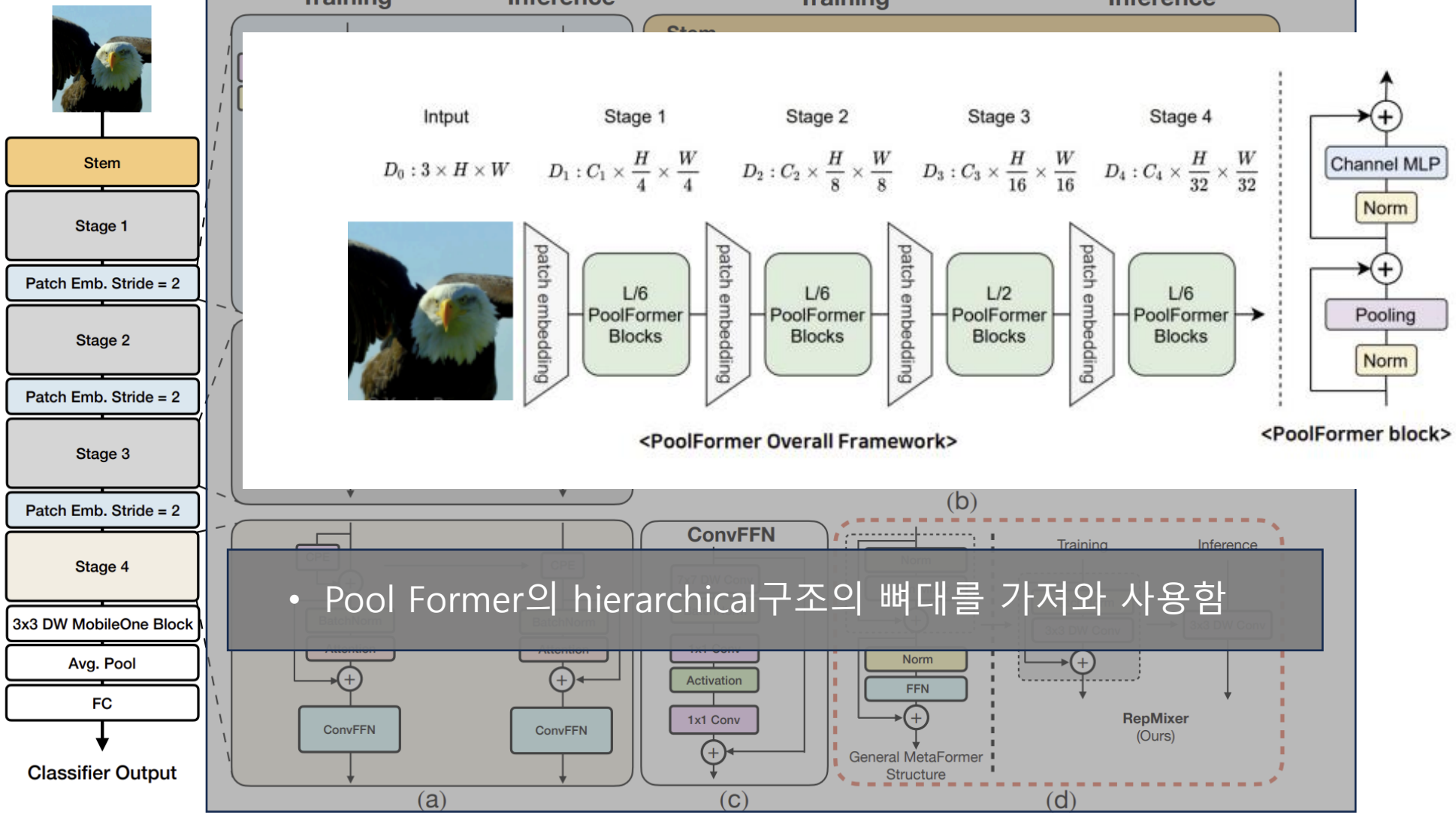
3. Proposed Method

FastViT overall architecture 정리



3. Proposed Method

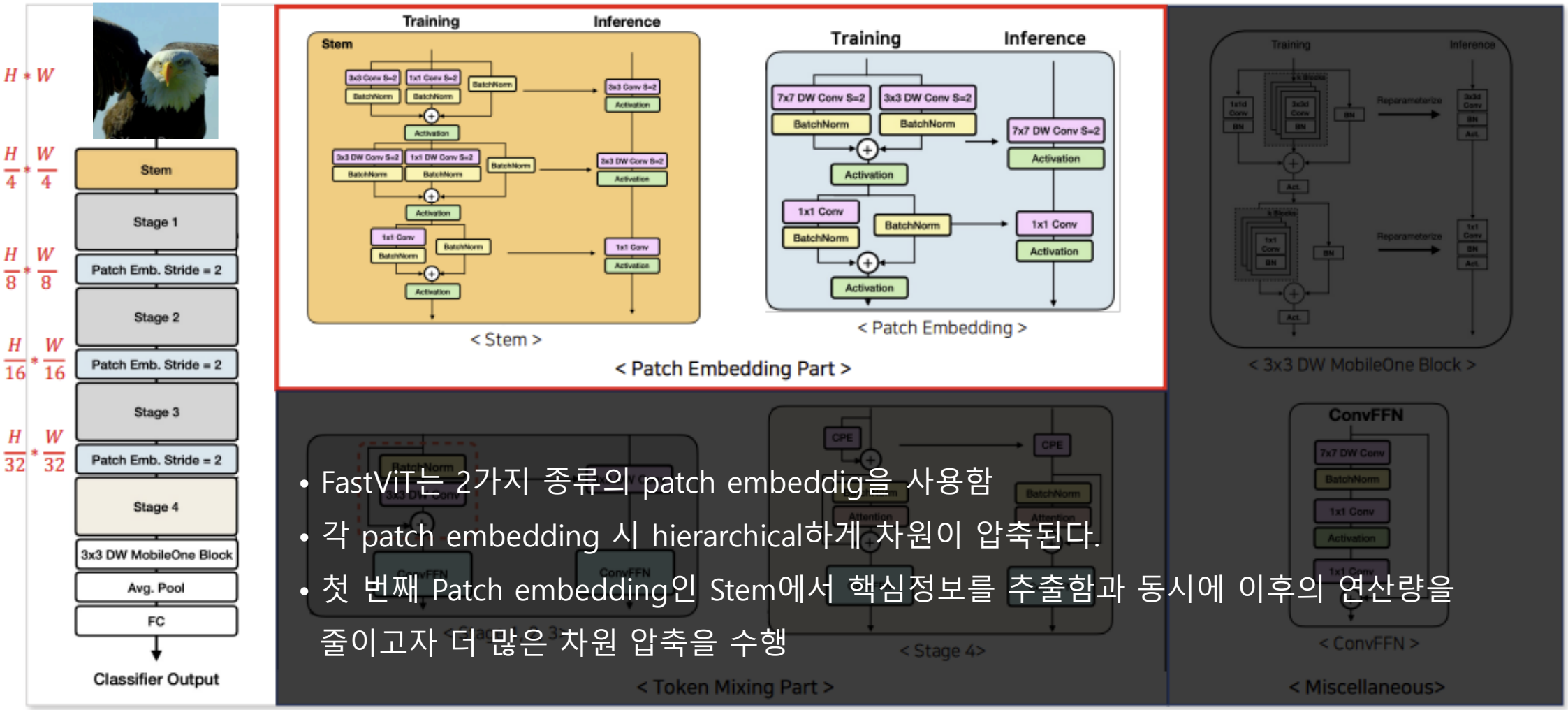
FastViT overall architecture정리 - hierarchical architecture



- Pool Former의 hierarchical구조의 뼈대를 가져와 사용함

3. Proposed Method

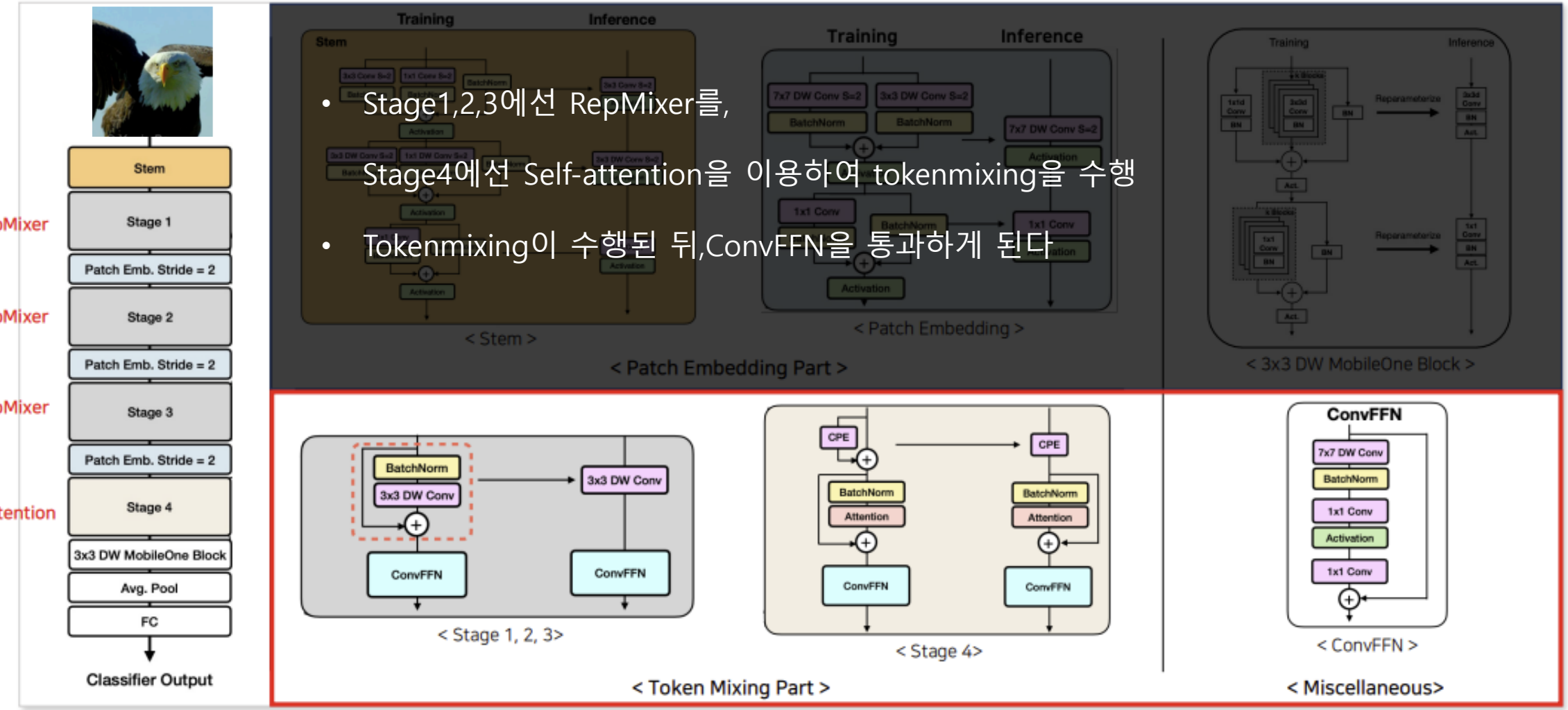
FastViT overall architecture정리 - patch embedding



- FastViT는 2가지 종류의 patch embeddig을 사용함
- 각 patch embedding 시 hierarchical하게 차원이 압축된다.
- 첫 번째 Patch embedding인 Stem에서 핵심정보를 추출함과 동시에 이후의 연산량을 줄이고자 더 많은 차원 압축을 수행

3. Proposed Method

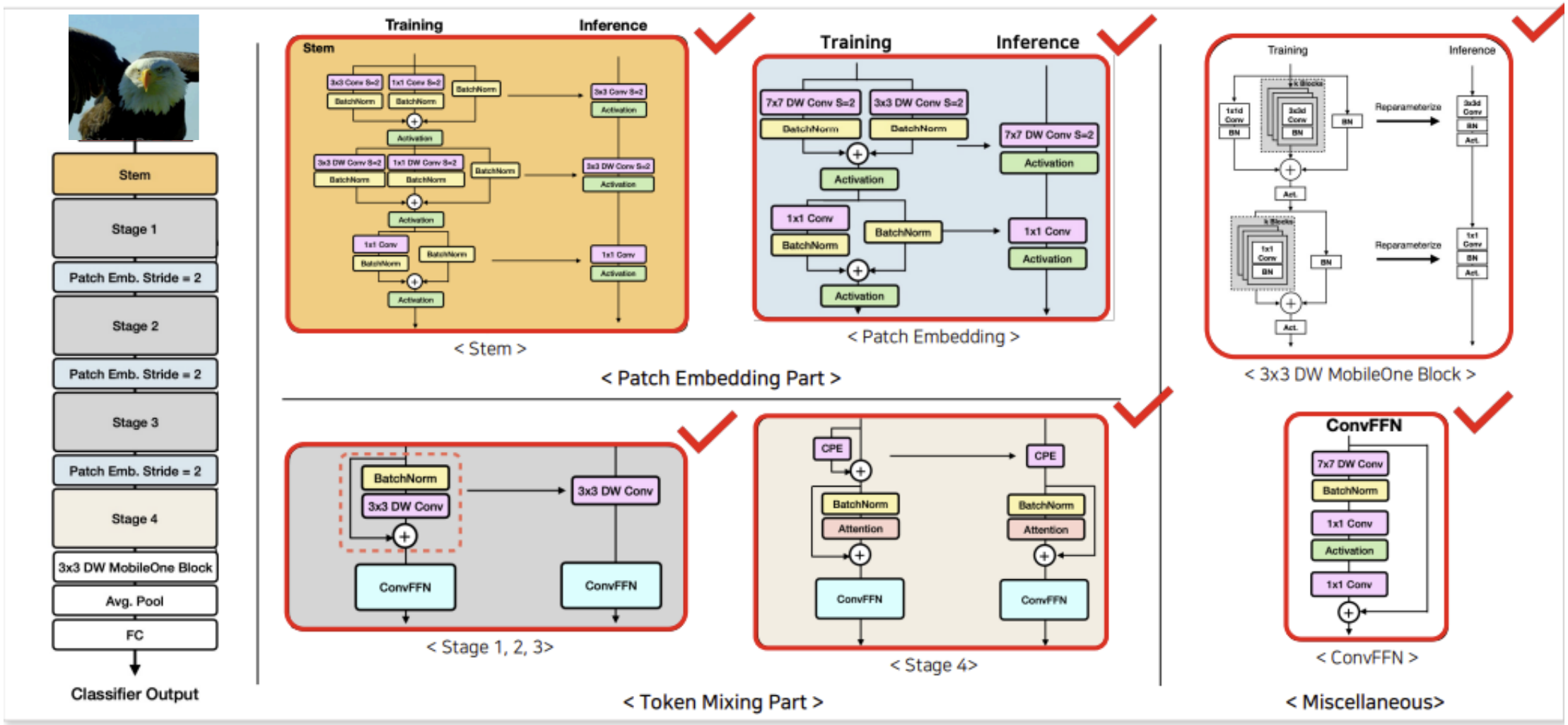
FastViT overall architecture정리 - token mixing



- Stage1,2,3에선 RepMixer를,
  Stage4에선 Self-attention을 이용하여 tokenmixing을 수행
- Tokenmixing이 수행된 뒤,ConvFFN을 통과하게 된다

3. Proposed Method

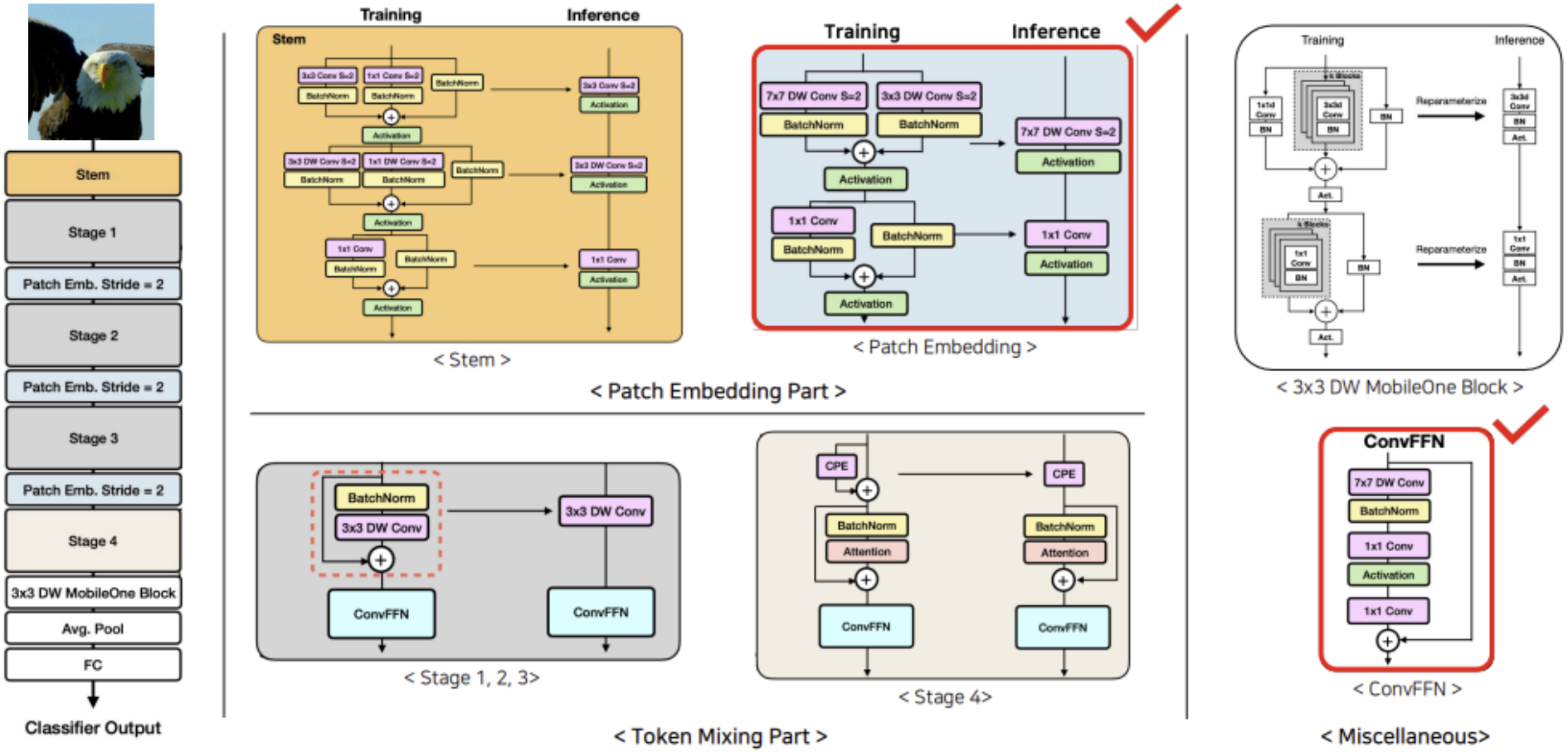
FastViT overall architecture정리 - overparameterization & reparameterization



- Patch embedding과 MobileOne block에서 **Overparameterization을 이용하여 layer의 capacity를 향상시킴**
- 가능한 모든 연산들을 reparameterize한다.

3. Proposed Method

FastViT overall architecture정리 - large kernel convolution



- RepMixer의 상대적으로 **local한 receptivefield를 보완하기** 위해
- Patch embedding과 ConvFFN 초반에 **Large Kernel Convolution을 수행한다.**

4. Experiments

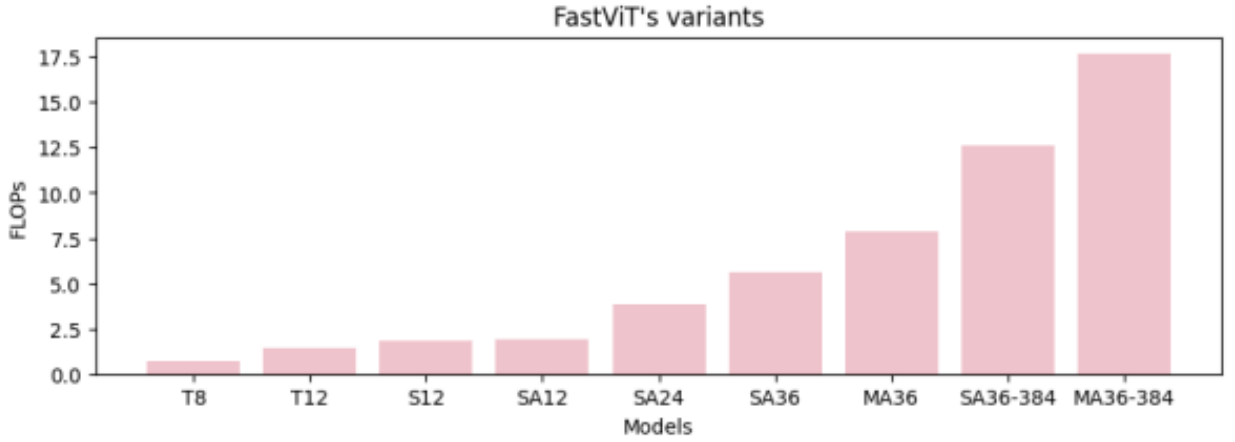
4. Experiments

FastViT's variants

Stage	#Tokens	Layer Spec.		FastViT						
				T8	T12	S12	SA12	SA24	SA36	MA36
Stem	$H \times W$	Conv.		3×3, stride 2						
				3×3 MobileOne Style, stride 2						
				48	64					76
1	$\frac{H}{4} \times \frac{W}{4}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				48	64					76
		FastViT Block	Mixer	RepMixer						
			Exp.	3			4			
			Blocks	2	2	2	2	4	6	6
2	$\frac{H}{8} \times \frac{W}{8}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				96	128					152
		FastViT Block	Mixer	RepMixer						
			Exp.	3			4			
			Blocks	2	2	2	2	4	6	6
3	$\frac{H}{16} \times \frac{W}{16}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				192	256					304
		FastViT Block	Mixer	RepMixer						
			Exp.	3			4			
			Blocks	4	6	6	6	12	18	18
4	$\frac{H}{32} \times \frac{W}{32}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				384	512					608
		FastViT Block	Mixer	RepMixer			Attention			
			Exp.	3			4			
			Blocks	2	2	2	2	4	6	6
Parameters (M)				3.6	6.8	8.8	10.9	20.6	30.4	42.7
FLOPs (G)				0.7	1.4	1.8	1.9	3.8	5.6	7.9

T8, T12, S12, SA12, SA24, SA36, MA36, SA36-384, MA36-384

- 알파벳 뒤의 숫자는 전체 FastViT Block의 수를 의미
- Patch embedding dimension : T ≤ S < M
- T는 ConvFFN의 MLP expansion ratio가 4보다 작은 모델
- A는 Stage 4에서의 self-attention 수행 여부



- 256x256 해상도로 train한 7개의 모델 +384x384 해상도로 fine-tuning한 2개의 모델(SA36,MA36) **총 9개의 모델을 제시**
- FLOPs 기준에서 보면 순차적으로 더 큰 model임을 알 수 있음

4. Experiments

Image Classification Settings

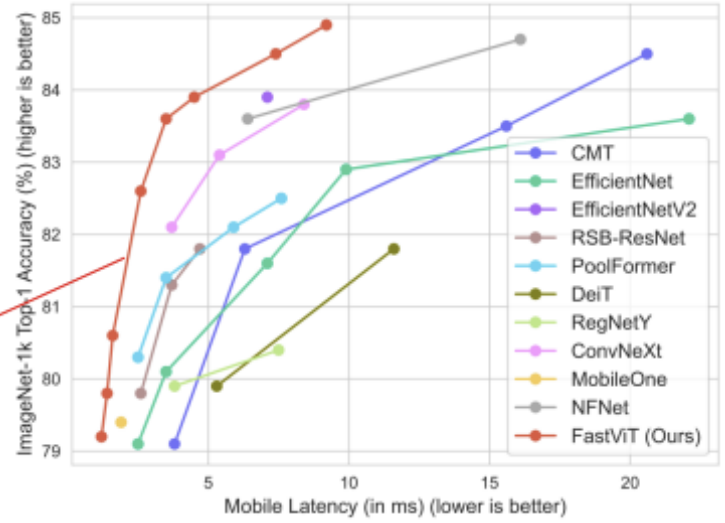
- **Dataset**: ImageNet-1K
  (~1.3M training images, 50K validation images)
- **For iPhone latency measurements**,
  we export the models using Core ML Tools (v6.0) and run it on iPhone 12 Pro Max with iOS 16 and batch size is set to 1 for all the models.
- **For GPU latency measurements**,
  we export the traced model to TensorRT format and run it on NVIDIA RTX-2080Ti with batch size of 8
- We report the **median latency estimate from 100 runs**.

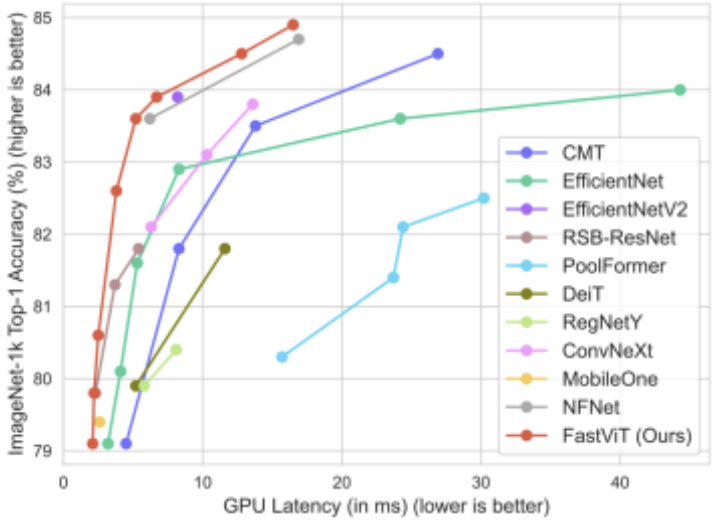
Hyperparameter	Training T8, T12, S12, SA12, SA24, SA36, MA36	Fine-tuning SA36, MA36
Stochastic depth rate	[0.0, 0.0, 0.0, 0.1, 0.1, 0.2, 0.35]	[0.2, 0.4]
Input resolution	256×256	384×384
Data augmentation	RandAugment	RandAugment
Mixup α	0.8	0.8
CutMix α	1.0	1.0
Random erase prob.	0.25	0.25
Label smoothing	0.1	0.1
Train epochs	300	30
Warmup epochs	5	None
Batch size	1024	1024
Optimizer	AdamW	AdamW
Peak learning rate	1e-3	5e-6
LR. decay schedule	cosine	None
Weight decay rate	0.05	1e-8
Gradient clipping	None	None
EMA decay rate	0.9995	0.9995

4. Experiments

Image Classification- Results

FastViT





Model	Eval Image Size	Param (M)	FLOPs (G)	GPU Latency (ms)	Mobile Latency (ms)	Top-1 Acc. (%)
EfficientNet-B6 [49]	528	43.0	19.0	44.3	51.7	84.0
EfficientNet-B7 [49]	600	66.0	37.0	72.6	85.5	84.3
CMT-B* [16]	256	45.7	9.3	26.9	20.6	**84.5**
Swin-B [34]	384	88.0	47.0	-	57.2	**84.5**
FastViT-SA36	384	30.4	12.6	**12.8**	**7.4**	**84.5**

< FastViT와 EffNet 및 Swin transformer 간 비교>

- Latency-accuracy trade-off에 있어 mobile device(iPhone12Pro)와 desktop-grade GPU(RTX2080Ti)상에서 **SOTA 성능을** 보였다.
- 경량화 모델이 아닌 EfficientNet과 Swin Transformer, CMT와의 비교를 통해 **Top-1 accuracy performance 대비 latency가 현저하게 적음을 확인**할 수 있음

4. Experiments

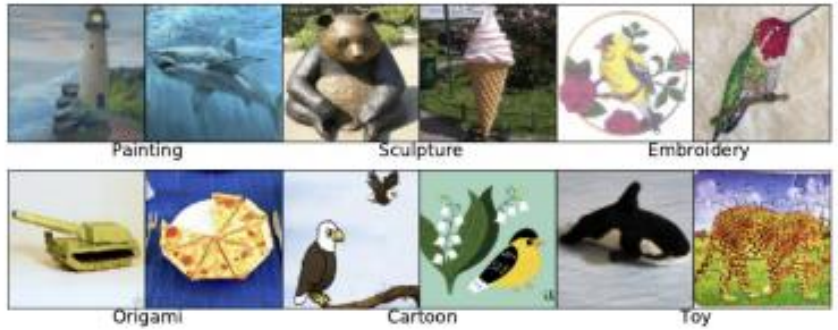
Robust Evaluation- Settings

- **Dataset**: Trained on ImageNet-1K,

  Evaluated on

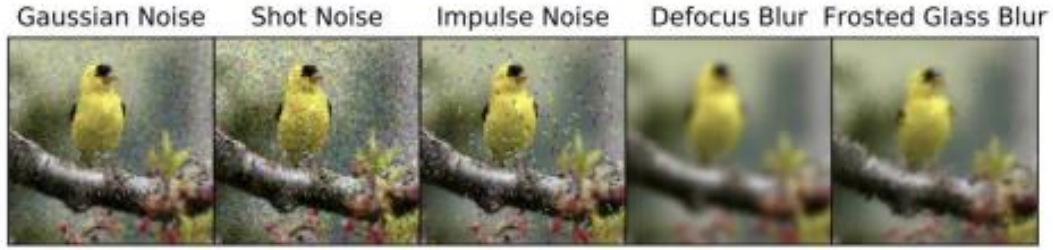
- **ImageNet-A(IN-A)**: ResNet-50에 의해 오분류된 example들

- **ImageNet-R(IN-R)**: ImageNet에 존재하는 class들의 rendition version



- **ImageNet-Sketch(IN-SK)**: ImageNet에 존재하는 class의 sketch version

- **ImageNet-C(IN-C)**: ImageNet test-set에 blur, noise등의 corruption을 가한 데이터셋



- ImageNet-1K에 대해 학습된 모델들을 아래의 데이터셋들에 대해 evaluation한다.
- Common Corruption Robustness: IN-C on mCE (mean Corruption Error:다양한 corruption을 적용한 결과의 error평균)
- Out-of-distribution Robustness: IN-A, IN-R, IN-SK on top@ 1 accuracy

4. Experiments

Robustness Evaluation- Results

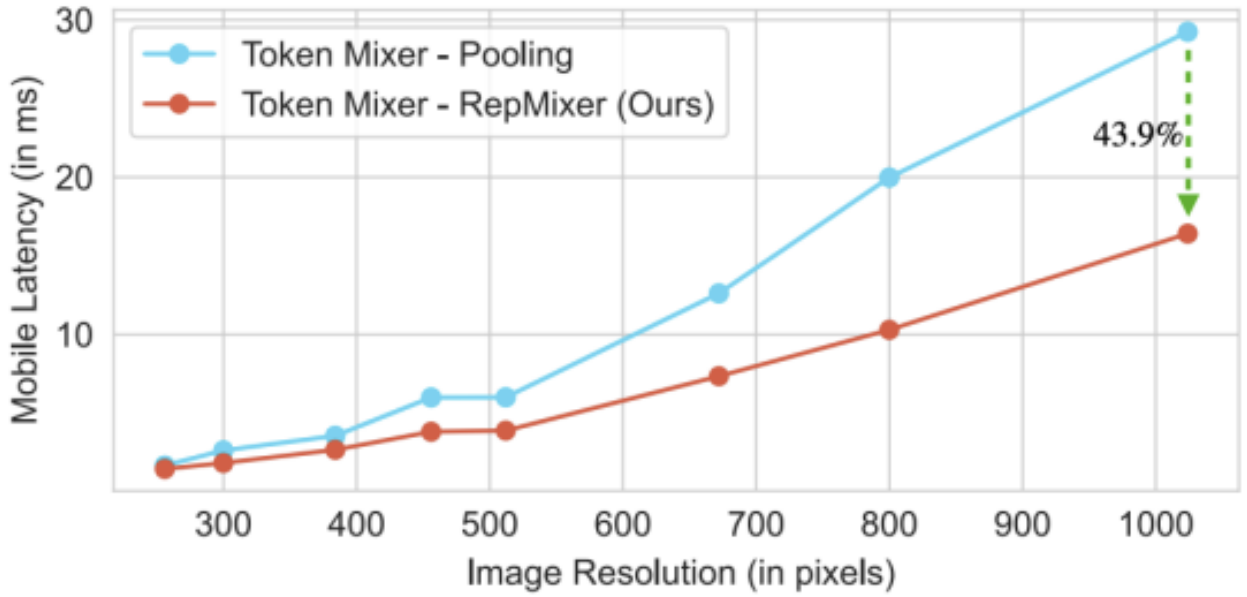
lower is better

Model	#Params	#FLOPs	Clean	IN-C (↓)	IN-A	IN-R	IN-SK
PVT-Tiny	13.2	1.9	75.0	79.6	7.9	33.9	21.5
RVT-Ti	8.6	1.3	78.4	**58.2**	13.3	**43.7**	**30.0**
FastViT-SA12	10.9	1.9	**80.6**	62.2	**17.2**	42.6	29.7
PVT-Small	24.5	3.8	79.9	66.9	18.0	40.1	27.2
ResNet-50*	25.6	4.1	79.0	65.5	5.9	42.5	31.5
ResNeXt50-32x4d	25.0	4.3	79.8	64.7	10.7	41.5	29.3
FastViT-SA24	20.6	3.8	**82.6**	**55.3**	**26.0**	**46.5**	**34.0**
Swin-T	28.3	4.5	81.2	62.0	21.6	41.3	29.1
ConViT-S	27.8	5.4	81.5	**49.8**	24.5	45.4	33.1
RVT-S	22.1	4.7	81.7	50.1	24.1	46.9	35.0
ConvNeXt-T	28.6	4.5	82.1	53.2	24.2	47.2	33.8
EfficientNet-B4	19.3	4.2	83.0	71.1	26.3	47.1	34.1
FastViT-SA36	30.4	5.6	**83.6**	51.8	**32.3**	**48.1**	**35.8**
ConvNeXt-S	50.0	8.7	83.1	51.2	31.2	**49.5**	**37.1**
FastViT-MA36	42.7	7.9	**83.9**	**49.9**	**34.6**	**49.5**	36.6

- FLOPs기준으로 묶은 모델들을 비교했을 때, 최근의 vision transformer보다 robust함과 동시에 pure-CNN기반의 모델보다 더 빠른 속도를 보임

4. Experiments

Skip-connection이 정말로 latency에 있어서 bottleneck으로 작용했는가?



- MetaFormer-S12구조의 **token mixer를 RepMixer로 변경하여 실험을 수행**
- Meta Former와 Meta Former-RepMixer는 약 1.8GFLOPs의 동일한 연산량을 갖는다.
- 두 모델에 대하여 input image의 해상도를 키워가며 inference시의 latency를 측정한 결과,

  skip-connection을 제거한 RepMixer의 경우에서 해상도가 커질수록 latency 차이가 커짐을 확인할 수 있다.

4. Experiments

Ablation: Effect of Overparameterization

Model	Ablation	Top-1 (%)	Train Time (hrs)
FastViT-SA12	Without Train-Time Overparam.	80.0	31.3
	With Train-Time Overparam.	**80.6**	33.4
FastViT-SA36	Without Train-Time Overparam.	83.3	73.5
	With Train-Time Overparam.	**83.6**	76.7

< Ablation on Train-time Overparameterization>

- Train-time overparameterization을 적용하면 추가된 branch로 인해 computational overhead가 증가한다.
- 그러나 제안한 모델처럼 Patch embedding과 MobileOne block에만 Overparameterization을 적용하면 학습 시간 상승이 그리 크지 않다.
- 결과적으로, **acceptable한 학습시간 상승에 Top-1accuracy향상**

4. Experiments

Ablation: Effect of Stage 4's Self-attention

Variant	Stages				Params	Top-1	Mobile
	1	2	3	4	(M)	(%)	Latency (ms)
Standard Setting							
V1	RM	RM	RM	RM	8.7	78.9	1.3
V2	RM	RM	RM	SA	10.8	79.9	1.5
V3	RM	RM	SA	SA	12.4	81.0	3.7
Large Kernel Convolutions (7×7)							
V4	RM	RM	RM	RM	8.8	79.8	1.4
V5	RM	RM	RM	SA	10.9	80.6	1.6

< Ablation on Stage 4's Self-attention & Large Kernel Convolution>

- Large Kernel Convolution을 적용하지 않은 경우와 적용한 경우 모두

  마지막 Stage에만 SA를 적용하면, **경미한 latency상승에 약 1%pt의 Top-1acc 향상을** 보였다.

- 하지만 Stage3과 4에 SA를 적용하면 급격한 latency상승이 일어났다.

Stage4에 SA를 적용하는 것은 latency-accuracy trade-off 관점에서 reasonable

4. Experiments

Ablation: Effect of Large Kernel Convolution

Variant	Stages				Params	Top-1	Mobile
	1	2	3	4	(M)	(%)	Latency (ms)
Standard Setting							
V1	RM	RM	RM	RM	8.7	78.9	1.3
V2	RM	RM	RM	SA	10.8	79.9	1.5
V3	RM	RM	SA	SA	12.4	81.0	3.7
Large Kernel Convolutions (7×7)							
V4	RM	RM	RM	RM	8.8	79.8	1.4
V5	RM	RM	RM	SA	10.9	80.6	1.6

< Ablation on Stage 4's Self-attention & Large Kernel Convolution>

- 동일 Token Mixe rsetting상의 모델들을 비교할 경우 ([V1,V4] ,[V2,V5]),

  두 경우 모두, **1ms의 경미한 latency 상승에 1%pt의 Top-1acc 향상을** 보였다.

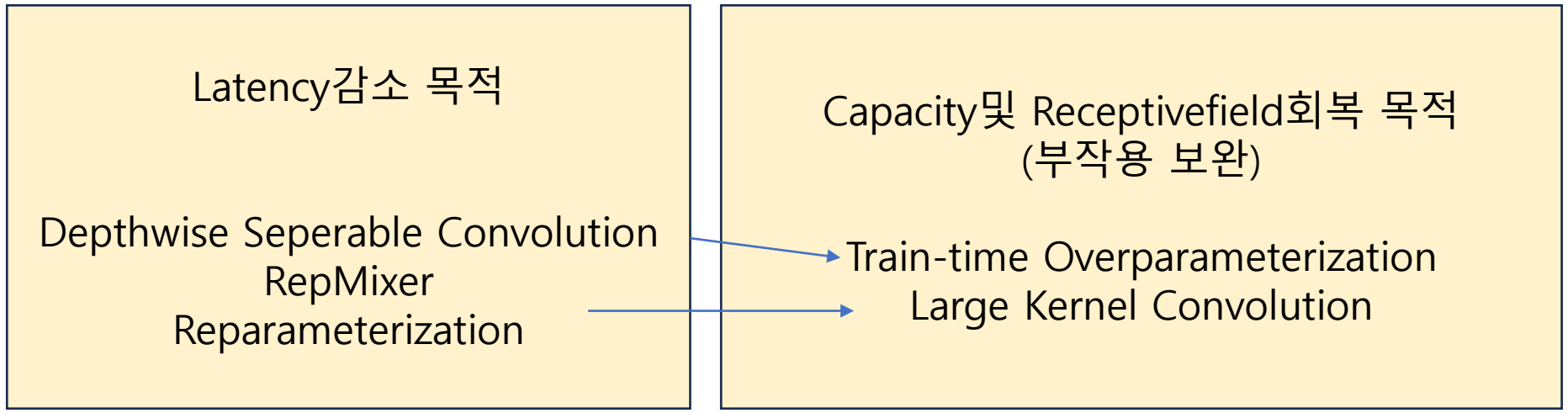
Large Kernel Convolution을 적용하는 것은 latency-accuracy trade-off 관점에서 reasonable!

5. Conclusion

Conclusion

- Edge device상에서 latency를 최소화 할 수 있는 hybrid vision transformer, FastViT를 제안

- **Latency를 감소시키기 위한 방법과** 그에 따라 나타나는 부작용을 완화할 수 있는 보완책을 함께 제안



- Image Classification task에서 **SOTA latency-accuracy trade-off**달성,
  이외에도 Robust Evaluation task에서 비슷한 연산량의 모델들과 비교 시 가장 robust한 모습을 보여줌

감사합니다.

Q & A