MobileViT Survey

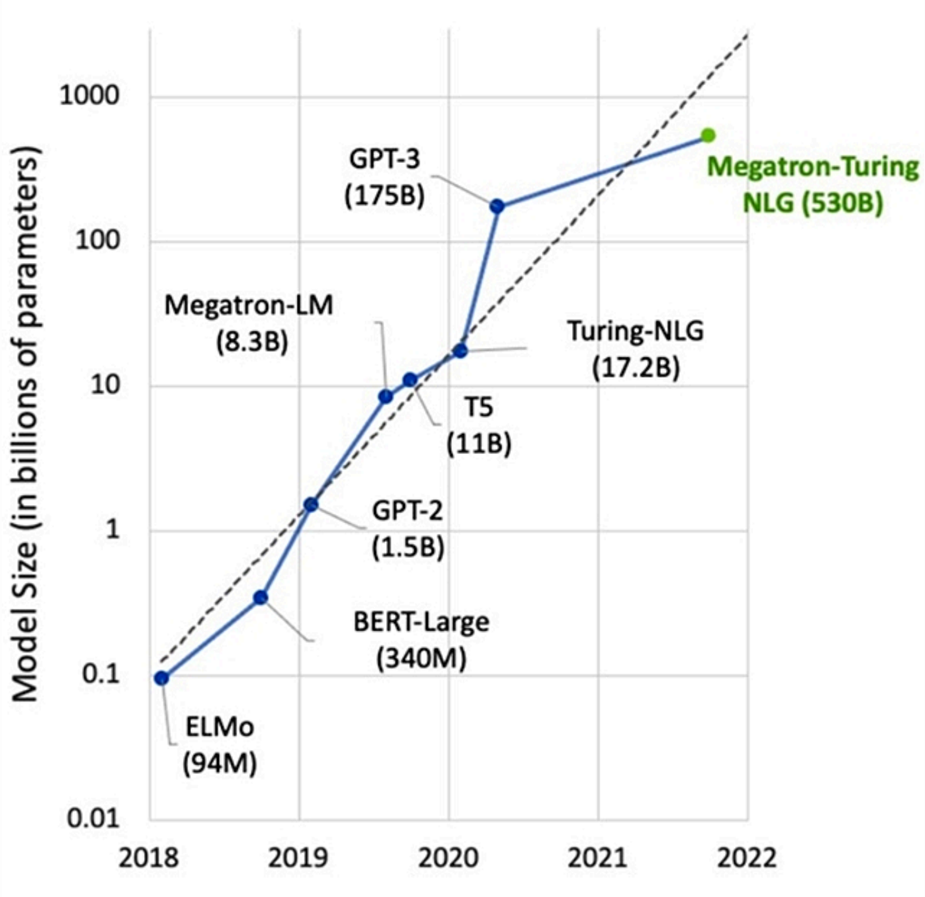
MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer

Written by Kim EunHo

Introduction

Bigger Model, Better Performance

Not available in mobile environment



Increasing number of parameters

https://www.researchgate.net/figure/Trend-of-sizes-of-state-of-the-art-NLP-models-with-time-Source-Using-DeepSpeed-and_fig3_370225792

Written by Kim EunHo

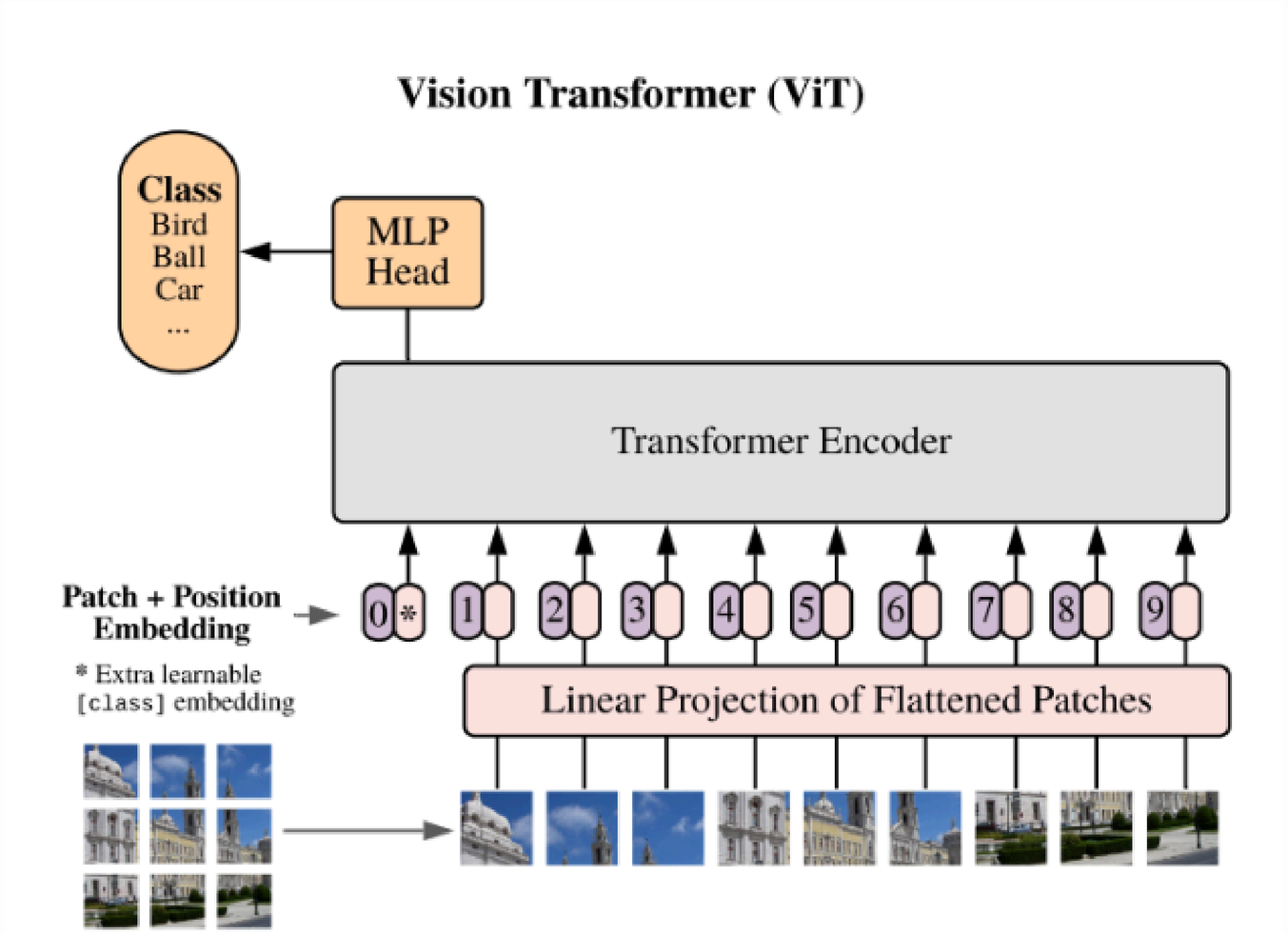
ViT(Vision Transformer)

Advantages

- Input-adaptive weighting
- Global processing

Disadvantages

- High computation
- Requires a lot of data



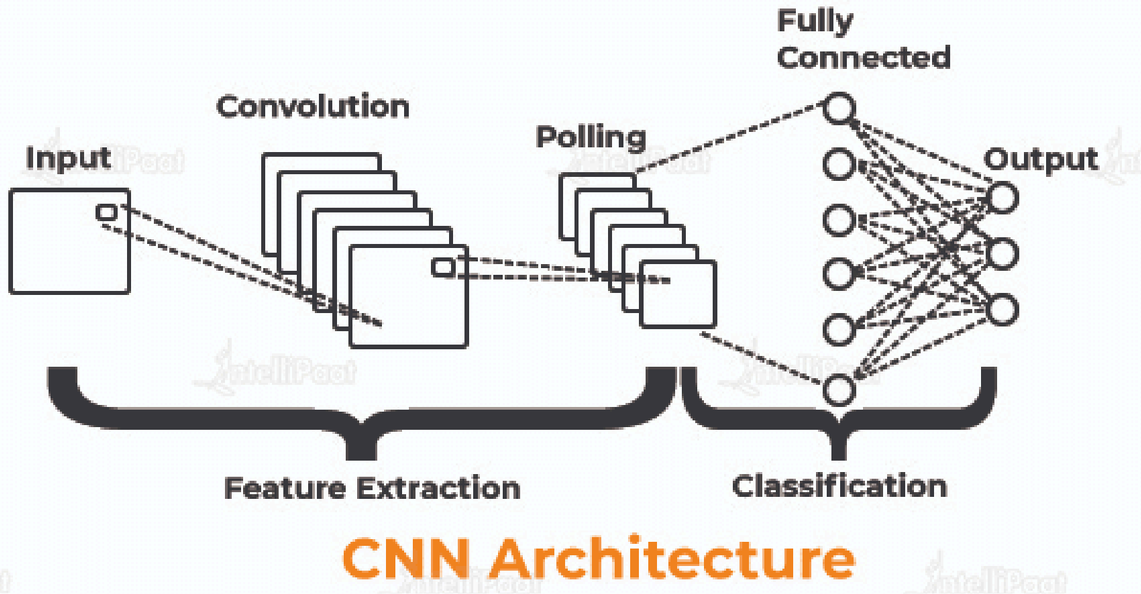
https://paperswithcode.com/method/vision-transformer

Written by Kim EunHo

CNN(Convolution Neural Network)

Advantages

- Local inductive bias
- Less sensitivity to data augmentation



Disadvantages

- Difficulty in Learning Long-Range Dependencies

https://intellipaat.com/blog/tutorial/artificial-intelligence-tutorial/convolution-neural-network/

Written by Kim EunHo

MobileViT

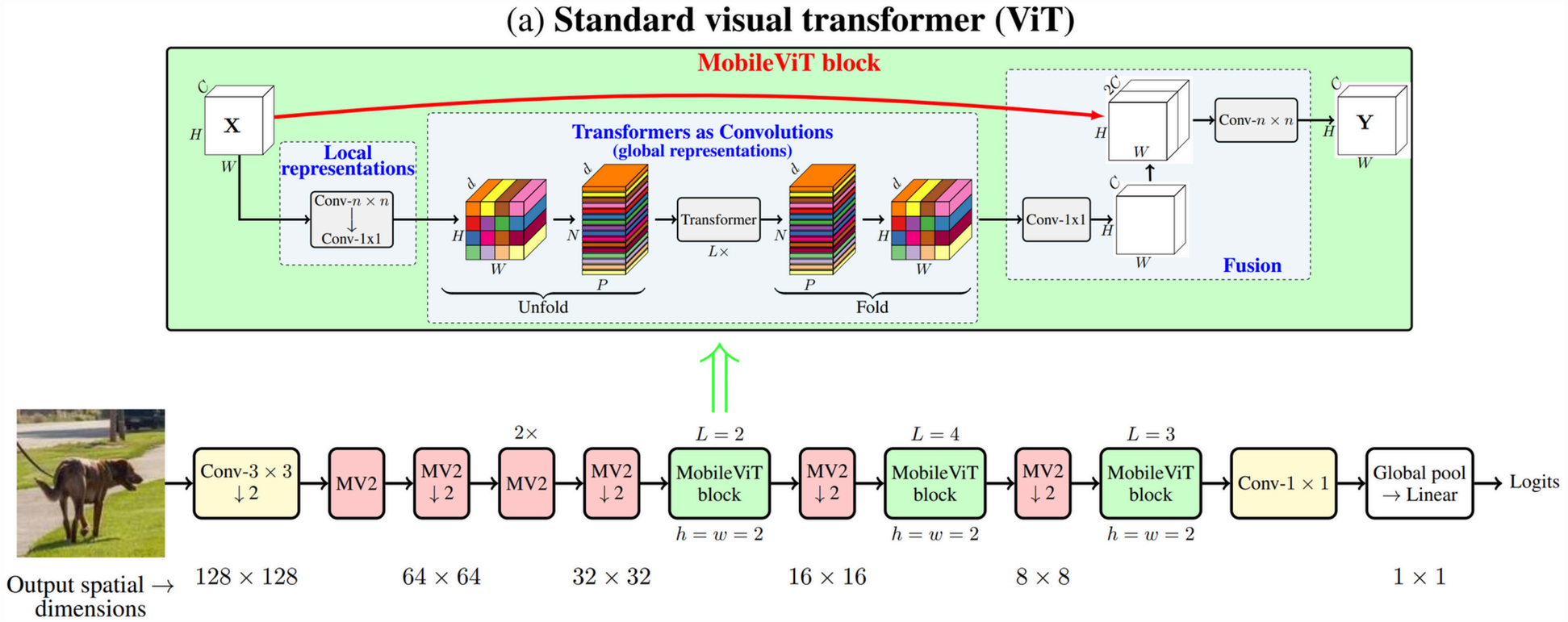
Advantages

- Light-weight
- Low-latency
- Local inductive bias

MobileViT and MobileNetv3 have a similar number of parameters, but shows a performance difference of 10% compared to the imageNet-1k dataset.

Written by Kim EunHo

MobileViT Architecture



MobileNetv2(MV2) Block + MobileViT Block

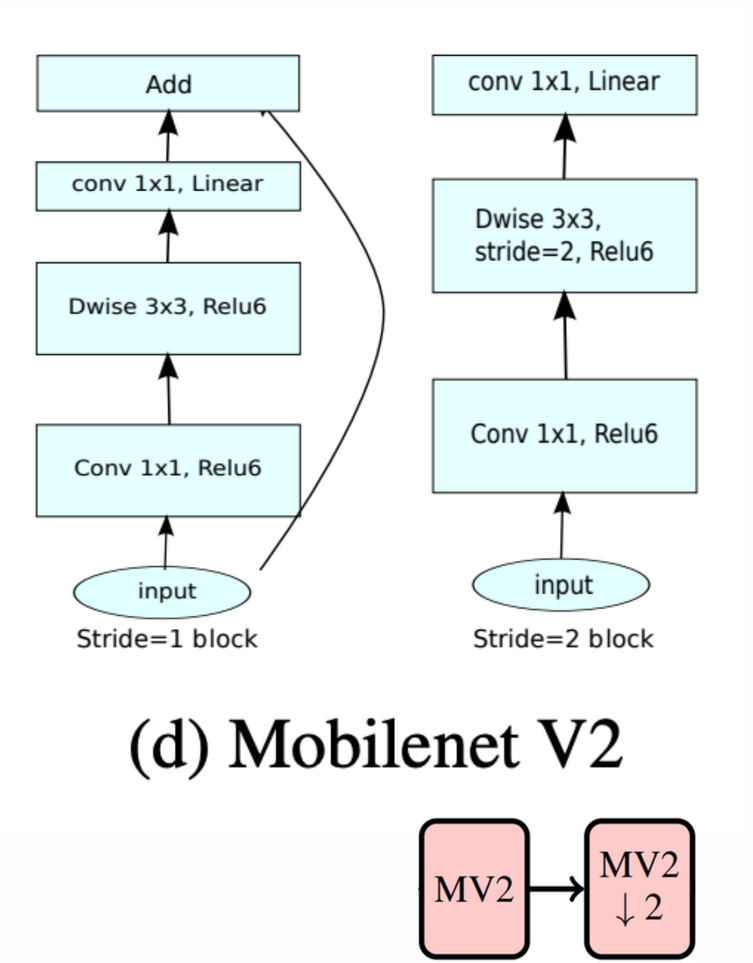
Local representations → Global representations → Fusion

Written by Kim EunHo

MobileNetv2 Architecture

MobileNetv2 Block

- Apply Inverted Residual architecture
- Stride = 1 Block is a residual block that adds input to the final output.
- Stride = 2 Block is a block that performs downsampling.
- There is no activation function in the last layer of both blocks.



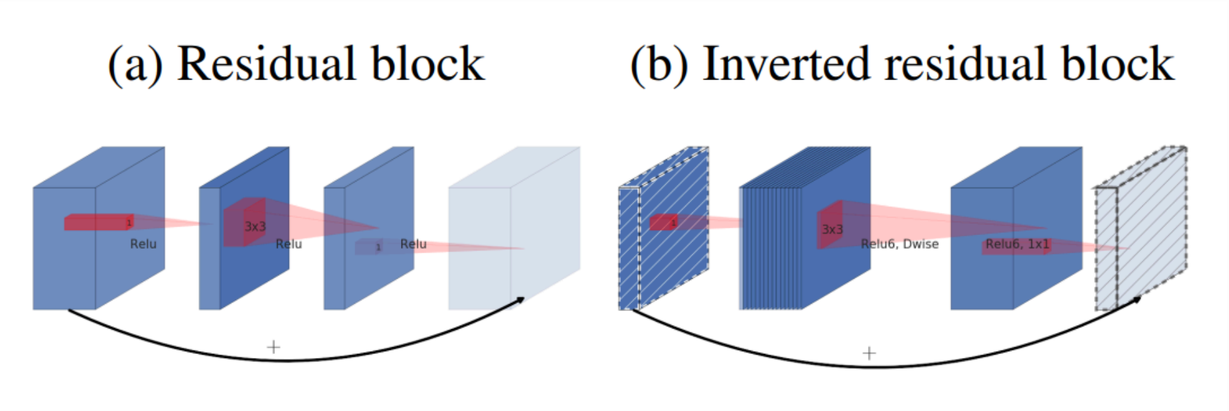
https://paperswithcode.com/method/mobilenetv2, https://arxiv.org/abs/1801.04381

MobileNetv2 Architecture

Residual vs. Inverted Residual

- Residual(MobileNetv1)
  - wide-narrow-wide
  - Prevent loss of learned information

- Inverted Residual(MobileNetv2)
  - narrow-wide-narrow
  - Suppose there is little input data but enough information to specify the target
  - Reduce memory usage



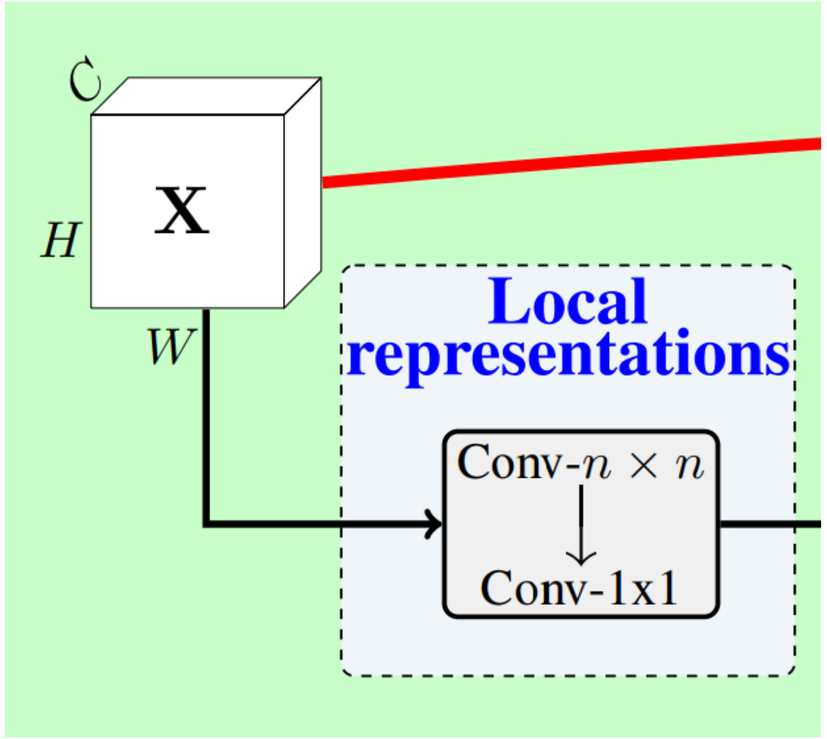
https://medium.com/@dlgkswn3124/summary-mobilenetv2-v3-950575322778

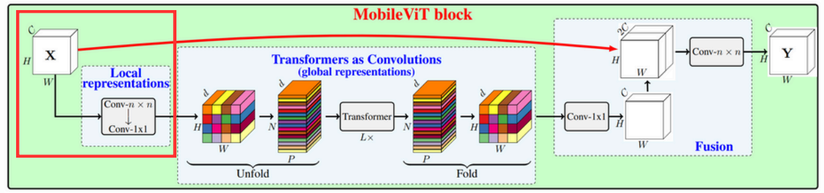
Written by Kim EunHo

MobileVIT Block

Local representation

- Local representation is the process of obtaining local information.
- First, perform N x N convolution operation on the input image.(in this paper use N = 3)
- And then perform point wise convolution operation
- Projection to high dimension through point wise convolution layer





Written by Kim EunHo

MobileVIT Block

Local representations

```
def forward_spatial(self, x: Tensor) -> Tensor:
    res = x

    fm = self.local_rep(x)

    # convert feature map to patches
    patches, info_dict = self.unfolding(fm)

    # learn global representations
    for transformer_layer in self.global_rep:
        patches = transformer_layer(patches)

    # [B x Patch x Patches x C] --> [B x C x Patches x Patch]
    fm = self.folding(patches=patches, info_dict=info_dict)

    fm = self.conv_proj(fm)

    if self.fusion is not None:
        fm = self.fusion(torch.cat((res, fm), dim=1))
    return fm
```

```
self.local_rep = nn.Sequential()
self.local_rep.add_module(name="conv_3x3", module=conv_3x3_in)
self.local_rep.add_module(name="conv_1x1", module=conv_1x1_in)
```

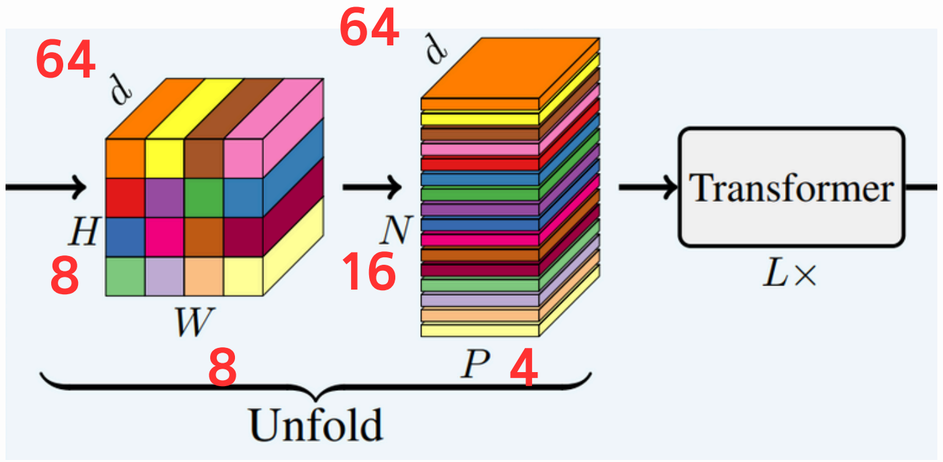
- Save input x to res. This is later used in residual operations.
- Put input x into local_rep
- local_rep consists of 3x3 and 1x1 convolution.

Written by Kim EunHo

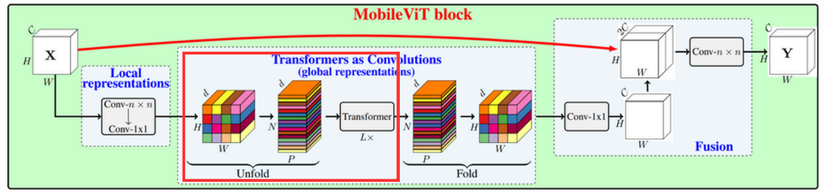
MobileVIT Block

Global representations

- The result of local representation is defined as X_L(X_L = W x H x d).
- Unfold X_L into non-overlapping patches. This is defined as X_U(X_U = P x N x d).
- X_U performs Transformer operation. This is defined as X_G.



$$\mathbf{X}_G(p) = \text{Transformer}(\mathbf{X}_U(p)), 1 \leq p \leq P$$



Written by Kim EunHo

MobileVIT Block

Global representations code

```
def forward_spatial(self, x: Tensor) -> Tensor:
    res = x

    fm = self.local_rep(x)

    # convert feature map to patches
    patches, info_dict = self.unfolding(fm)

    # learn global representations
    for transformer_layer in self.global_rep:
        patches = transformer_layer(patches)

    # [B x Patch x Patches x C] --> [B x C x Patches x Patch]
    fm = self.folding(patches=patches, info_dict=info_dict)

    fm = self.conv_proj(fm)

    if self.fusion is not None:
        fm = self.fusion(torch.cat((res, fm), dim=1))
    return fm
```

- FM has local information and obtains patches by unfolding it.
- Insert the obtained patches into the Transformer layer.

Written by Kim EunHo

MobileVIT Block

Global representations code

```
def unfolding(self, feature_map: Tensor) -> Tuple[Tensor, Dict]:
    patch_w, patch_h = self.patch_w, self.patch_h
    patch_area = int(patch_w * patch_h)
    batch_size, in_channels, orig_h, orig_w = feature_map.shape

    new_h = int(math.ceil(orig_h / self.patch_h) * self.patch_h)
    new_w = int(math.ceil(orig_w / self.patch_w) * self.patch_w)

    interpolate = False
    if new_w != orig_w or new_h != orig_h:
        # Note: Padding can be done, but then it needs to be handled in attention function.
        feature_map = F.interpolate(
            feature_map, size=(new_h, new_w), mode="bilinear", align_corners=False
        )
        interpolate = True

    # number of patches along width and height
    num_patch_w = new_w // patch_w  # n_w
    num_patch_h = new_h // patch_h  # n_h
    num_patches = num_patch_h * num_patch_w  # N

    # [B, C, H, W] --> [B * C * n_h, p_h, n_w, p_w]
    reshaped_fm = feature_map.reshape(
        batch_size * in_channels * num_patch_h, patch_h, num_patch_w, patch_w
    )
```

- Get orig_h and orig_w from fm and get new_h and new_w.
- Interpolate is performed when new_w and new_h do not match orig_w and orig_h in size.

Written by Kim EunHo

MobileVIT Block

Global representations code

```
# [B * C * n_h, p_h, n_w, p_w] --> [B * C * n_h, n_w, p_h, p_w]
transposed_fm = reshaped_fm.transpose(1, 2)
# [B * C * n_h, n_w, p_h, p_w] --> [B, C, N, P] where P = p_h * p_w and N = n_h * n_w
reshaped_fm = transposed_fm.reshape(
    batch_size, in_channels, num_patches, patch_area
)
# [B, C, N, P] --> [B, P, N, C]
transposed_fm = reshaped_fm.transpose(1, 3)
# [B, P, N, C] --> [BP, N, C]
patches = transposed_fm.reshape(batch_size * patch_area, num_patches, -1)

info_dict = {
    "orig_size": (orig_h, orig_w),
    "batch_size": batch_size,
    "interpolate": interpolate,
    "total_patches": num_patches,
    "num_patches_w": num_patch_w,
    "num_patches_h": num_patch_h,
}

return patches, info_dict
```

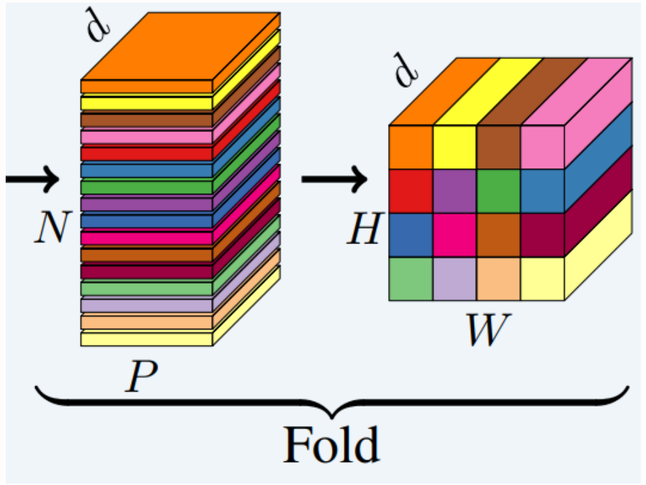
- Afterwards, change [B, C, H, W] to [BP, N, C] through reshape and transform.
- info_dict is used later to help with the fold process.

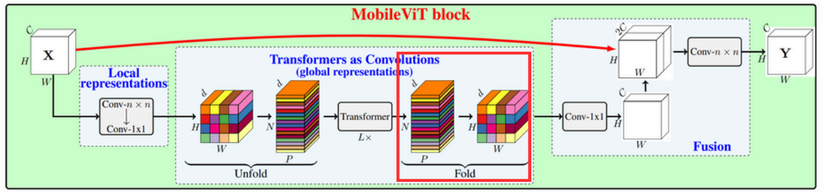
Written by Kim EunHo

MobileVIT Block

Global representations

- The X_G that performed the Transformer operation has both local and global information.
- Fold X_G again to be the same as the input shape.
- When the patch is spread out flat, it spreads in order, so positional embedding is not necessary.





Written by Kim EunHo

MobileVIT Block

Global representations code

```
def forward_spatial(self, x: Tensor) -> Tensor:
    res = x

    fm = self.local_rep(x)

    # convert feature map to patches
    patches, info_dict = self.unfolding(fm)

    # learn global representations
    for transformer_layer in self.global_rep:
        patches = transformer_layer(patches)

    # [B x Patch x Patches x C] --> [B x C x Patches x Patch]
    fm = self.folding(patches=patches, info_dict=info_dict)

    fm = self.conv_proj(fm)

    if self.fusion is not None:
        fm = self.fusion(torch.cat((res, fm), dim=1))
    return fm
```

- Patches obtained through Transformer operation are placed in folding.

Written by Kim EunHo

MobileVIT Block

Global representations code

```
def folding(self, patches: Tensor, info_dict: Dict) -> Tensor:
    n_dim = patches.dim()
    assert n_dim == 3, "Tensor should be of shape BPxNxC. Got: {}".format(
        patches.shape
    )
    # [BP, N, C] --> [B, P, N, C]
    patches = patches.contiguous().view(
        info_dict["batch_size"], self.patch_area, info_dict["total_patches"], -1
    )

    batch_size, pixels, num_patches, channels = patches.size()
    num_patch_h = info_dict["num_patches_h"]
    num_patch_w = info_dict["num_patches_w"]

    # [B, P, N, C] --> [B, C, N, P]
    patches = patches.transpose(1, 3)

    # [B, C, N, P] --> [B*C*n_h, n_w, p_h, p_w]
    feature_map = patches.reshape(
        batch_size * channels * num_patch_h, num_patch_w, self.patch_h, self.patch_w
    )
```

- First, check whether the input dimension is 3D.
- The folding process is carried out in the reverse order of unfolding.

Written by Kim EunHo

MobileVIT Block

Global representations code

```
# [B*C*n_h, n_w, p_h, p_w] --> [B*C*n_h, p_h, n_w, p_w]
feature_map = feature_map.transpose(1, 2)
# [B*C*n_h, p_h, n_w, p_w] --> [B, C, H, W]
feature_map = feature_map.reshape(
    batch_size, channels, num_patch_h * self.patch_h, num_patch_w * self.patch_w
)
if info_dict["interpolate"]:
    feature_map = F.interpolate(
        feature_map,
        size=info_dict["orig_size"],
        mode="bilinear",
        align_corners=False,
    )
return feature_map
```

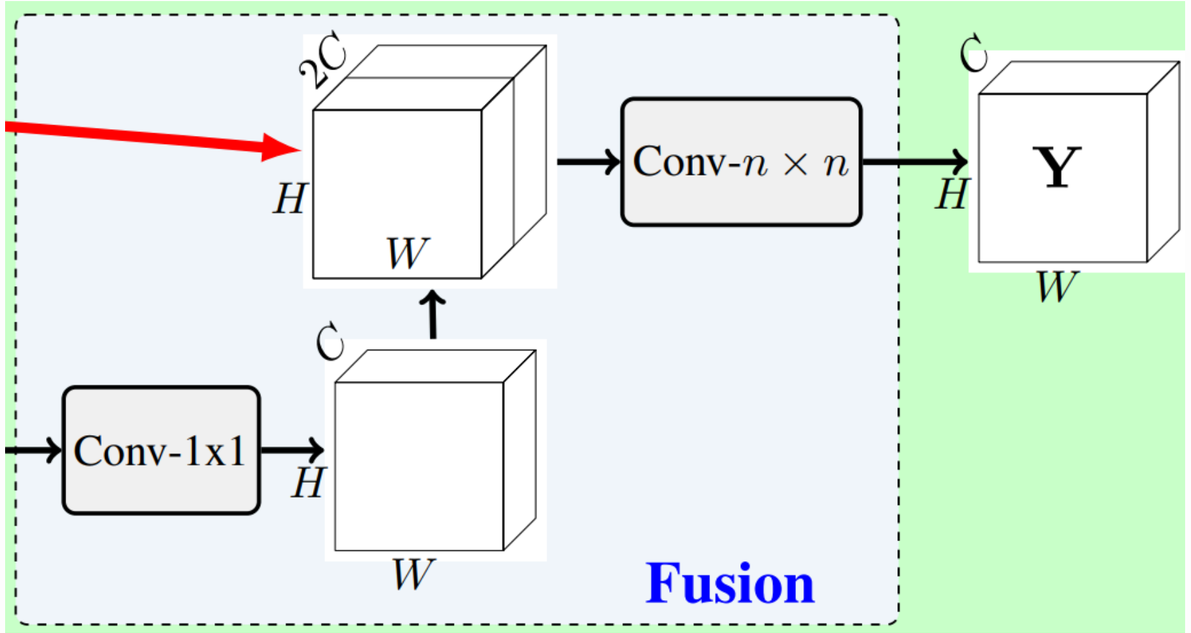
- If interpolation is performed because the image size is different, change it back to the original image.

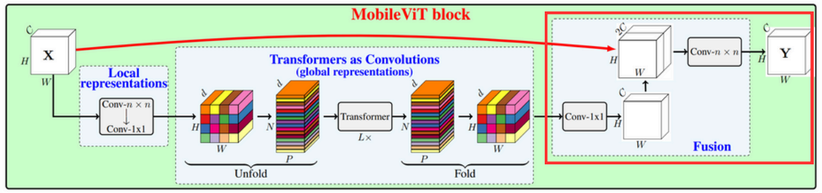
Written by Kim EunHo

MobileVIT Block

Fusion

- Reduces the dimension of data with Point wise convolution.
- Performs concatenation operations with input data and reduced data.
- Finally, nxn convolution is performed to combine the two into one.





Written by Kim EunHo

MobileVIT Block

```
def forward_spatial(self, x: Tensor) -> Tensor:
    res = x

    fm = self.local_rep(x)

    # convert feature map to patches
    patches, info_dict = self.unfolding(fm)

    # learn global representations
    for transformer_layer in self.global_rep:
        patches = transformer_layer(patches)

    # [B x Patch x Patches x C] --> [B x C x Patches x Patch]
    fm = self.folding(patches=patches, info_dict=info_dict)

    fm = self.conv_proj(fm)

    if self.fusion is not None:
        fm = self.fusion(torch.cat((res, fm), dim=1))
    return fm
```

```
self.conv_proj = conv_1x1_out


self.fusion = conv_3x3_out
```

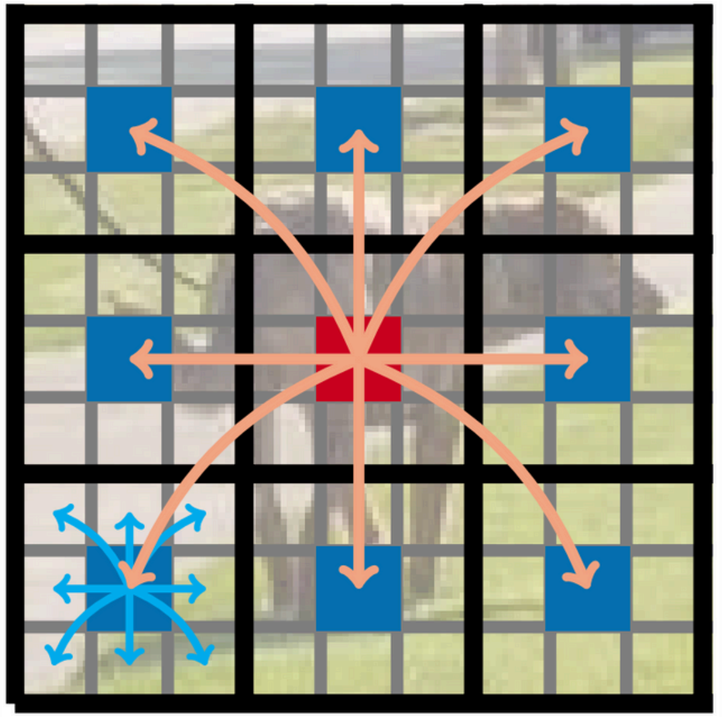
- Finally, the FM obtained through the folding process is passed through conv_proj and fusion to obtain the final output of MobileViT Block.

Written by Kim EunHo

Information Obtain Process

How can MobileViT obtain Local and Global information at the same time?

- Every pixel sees every other pixel in the MobileViT Block.
- **Red pixel** attends to **Blue pixels** using transformer.
- **Blue pixel** have already encoded information about neighboring pixels using convolution.

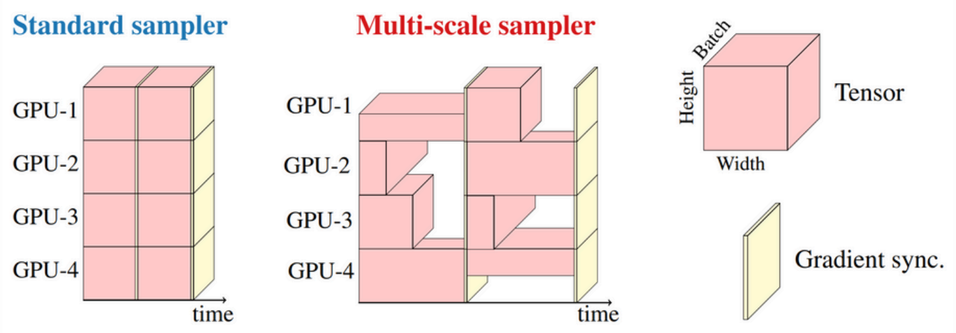


Written by Kim EunHo

Training Method

Multi-scale vs. Standard

- ViT-based model use positional embeddings.
- Learn for each iteration to specific image size.
- Batch size is determined based on the large image
- GPU is used inefficiently when learning a small image.



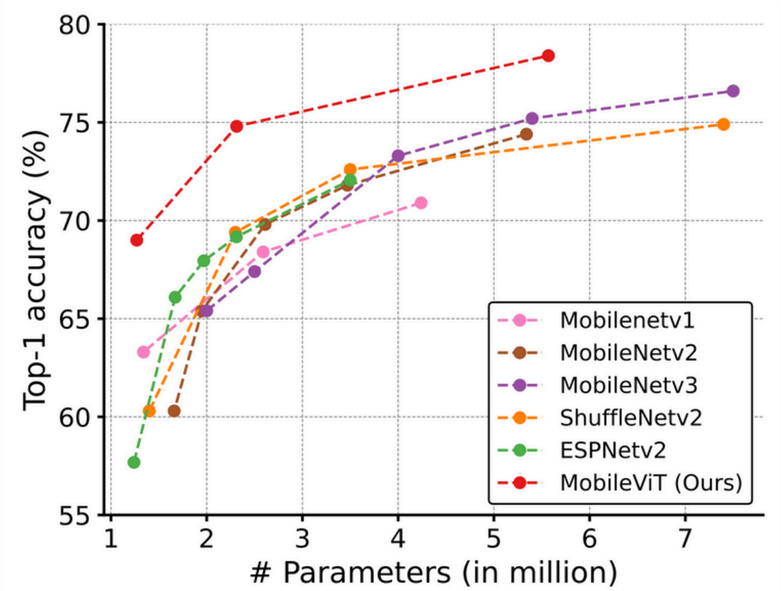
bt = t_{th} iteration's batch size
Ht, Wt = t_{th} iteration's image's height, width
Hn, Wn = biggest image's height, width
b = biggest image's batch size

$$b_t = \frac{H_n W_n b}{H_t W_t}$$

Sampler	# Updates	Epoch time
Standard	375 k	380 sec
Multi-scale (ours)	**232 k**	**270 sec**

Written by Kim EunHo

Performance Comparison



Comparison with light-weight CNNs Graph

Datasets : imagenet-1k validation set

Model	# Params. ⇓	Top-1 ⇑
MobileNetv1	2.6 M	68.4
MobileNetv2	2.6 M	69.8
MobileNetv3	2.5 M	67.4
ShuffleNetv2	2.3 M	69.4
ESPNetv2	2.3 M	69.2
MobileViT-XS (Ours)	2.3 M	**74.8**

Comparison with light-weight CNNs (similar parameters)

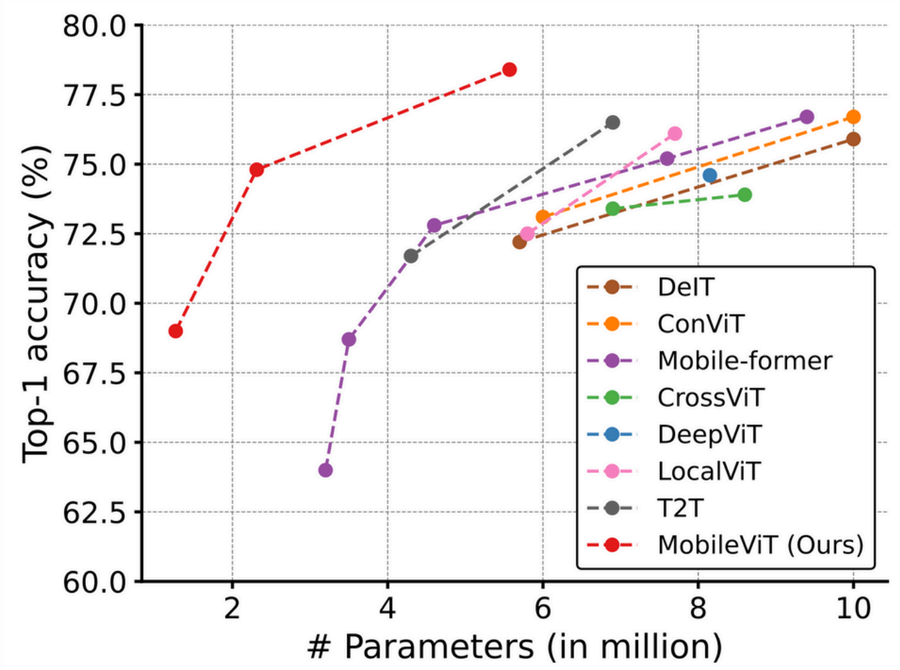
Model	# Params. ⇓	Top-1 ⇑
DenseNet-169	14 M	76.2
EfficientNet-B0	5.3 M	76.3
ResNet-101	44.5 M	77.4
ResNet-101-SE	49.3 M	77.6
MobileViT-S (Ours)	5.6 M	**78.4**

Comparison with Heavy-weight CNNs

Written by Kim EunHo

Performance Comparison

MobileViT vs. ViTs on ImageNet-1k validation set



Row #	Model	Augmentation	# Params. ⇓	Top-1 ⇑
R1	DeIT	Basic	5.7 M	68.7
R2	T2T	Advanced	4.3 M	71.7
R3	DeIT	Advanced	5.7 M	72.2
R4	PiT	Basic	10.6 M	72.4
R5	Mobile-former	Advanced	4.6 M	72.8
R6	PiT	Advanced	4.9 M	73.0
R7	CrossViT	Advanced	6.9 M	73.4
R8	MobileViT-XS (Ours)	Basic	2.3 M	**74.8**
R9	CeiT	Advanced	6.4 M	76.4
R10	DeIT	Advanced	10 M	75.9
R11	T2T	Advanced	6.9 M	76.5
R12	ViL	Advanced	6.7 M	76.7
R13	LocalVit	Advanced	7.7 M	76.1
R14	Mobile-former	Advanced	9.4 M	76.7
R15	PVT	Advanced	13.2 M	75.1
R16	ConViT	Advanced	10 M	76.7
R17	PiT	Advanced	10.6 M	78.1
R18	BoTNet	Basic	20.8 M	77.0
R19	BoTNet	Advanced	20.8 M	78.3
R20	MobileViT-S (Ours)	Basic	5.6 M	**78.4**

Performance Comparison

MobileViT as a general-purpose backbone

Feature backbone	# Params. ⇓	mAP ⇑
MobileNetv3	4.9 M	22.0
MobileNetv2	4.3 M	22.1
MobileNetv1	5.1 M	22.2
MixNet	4.5 M	22.3
MNASNet	4.9 M	23.0
MobileViT-XS (Ours)	**2.7 M**	24.8
MobileViT-S (Ours)	5.7 M	**27.7**

Comparison w/ light-weight CNNs

Feature backbone	# Params. ⇓	mAP ⇑
VGG	35.6 M	25.1
ResNet50	22.9 M	25.2
MobileViT-S (Ours)	**5.7 M**	**27.7**

Comparison with Heavy-weight CNNs

Performance Comparison

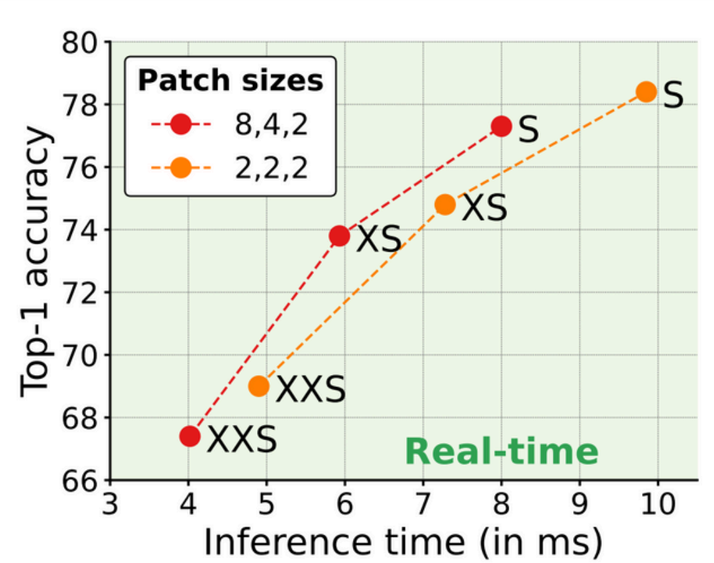
MobileViT as a general-purpose backbone

Feature backbone	# Params. ⇓	mIOU ⇑
MobileNetv1	11.2 M	75.3
MobileNetv2	4.5 M	75.7
MobileViT-XXS (Ours)	1.9 M	73.6
MobileViT-XS (Ours)	2.9 M	**77.1**
ResNet-101	58.2 M	**80.5**
MobileViT-S (Ours)	6.4 M	79.1

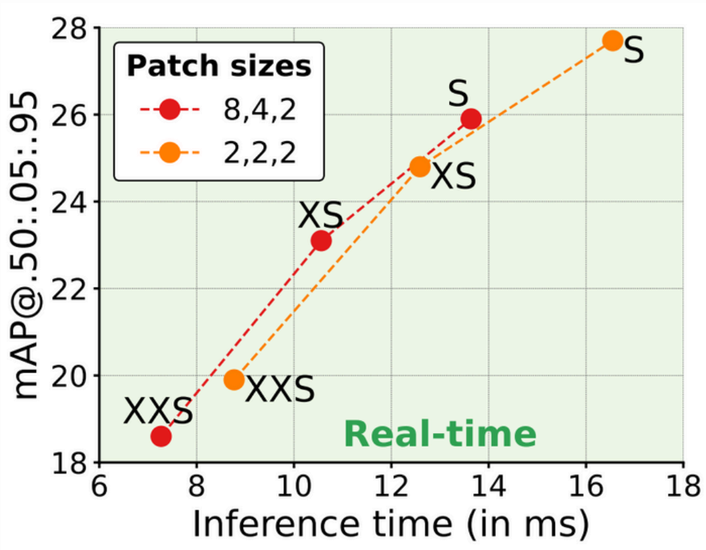
Segmentation w/ DeepLabv3

Performance Comparison

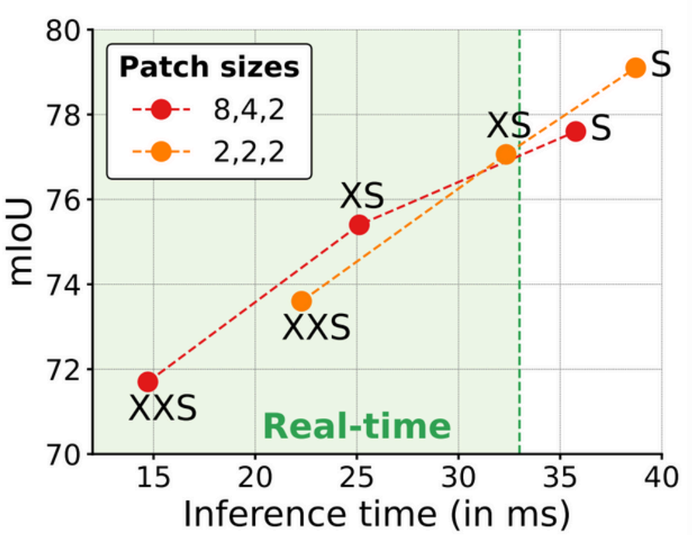
Inference time of MobileViT models on different tasks



Classification @256x256



Detection @320x320



Segmentation @512x512

Written by Kim EunHo

Performance Comparison

MobileVit inference time on-Device

Model	# Params ⇓	FLOPs ⇓	Top-1 ⇑	Inference time ⇓		
				iPhone12 - CPU	iPhone12 - Neural Engine	NVIDIA V100 GPU
MobileNetv2	3.5 M	**0.3 G**	73.3	**7.50 ms**	**0.92 ms**	**0.31 ms**
DeIT	5.7 M	1.3 G	72.2	28.15 ms	10.99 ms	0.43 ms
PiT	4.9 M	0.7 G	73.0	24.03 ms	10.56 ms	0.46 ms
MobileViT (Ours)	**2.3 M**	0.7 G	**74.8**	17.86 ms	7.28 ms	0.62 ms/0.47 ms†

Comparison of each model's Inference time on different devices.

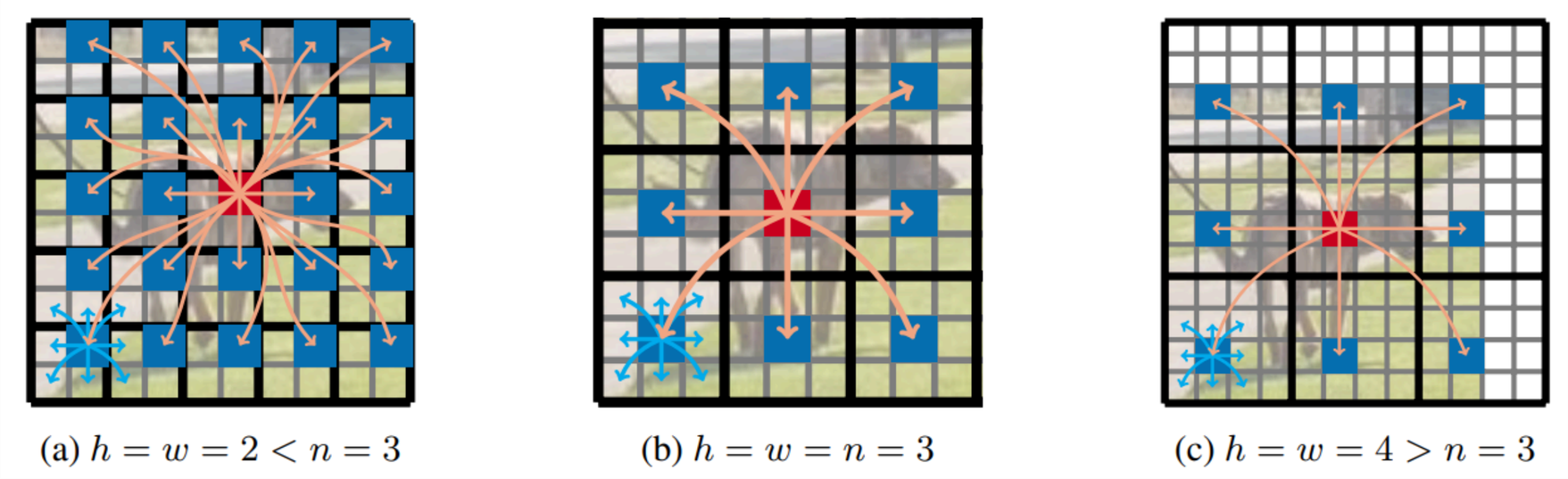
Written by Kim EunHo

Impact of patch sizes

Patch sizes	# Params.	Time ⇓	Top-1 ⇑
2,2,2	5.7 M	9.85 ms	78.4
3,3,3†	5.7 M	14.69 ms	**78.5**
4,4,4	5.7 M	8.23 ms	77.6
8,4,2	5.7 M	8.20 ms	77.3

Comparison of MobileViT-S models by patch size

Impact of patch sizes

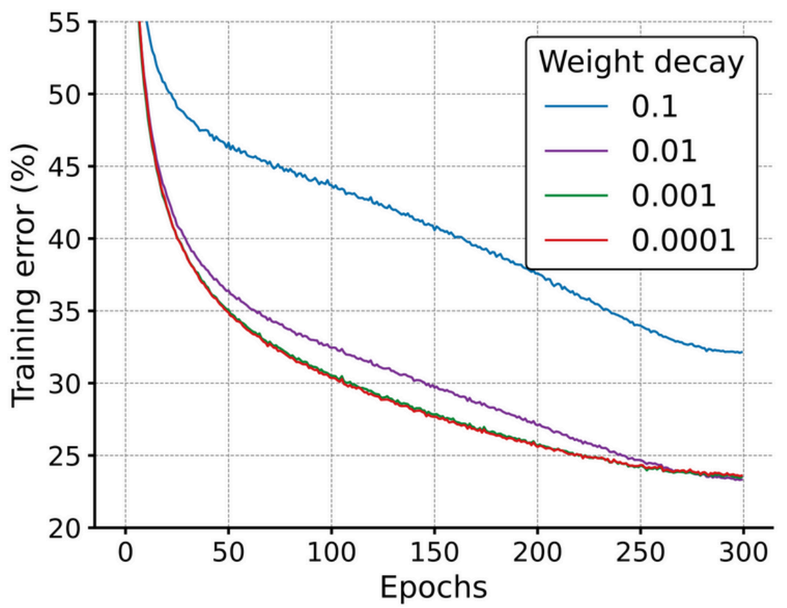


(a) $h = w = 2 < n = 3$ (b) $h = w = n = 3$ (c) $h = w = 4 > n = 3$

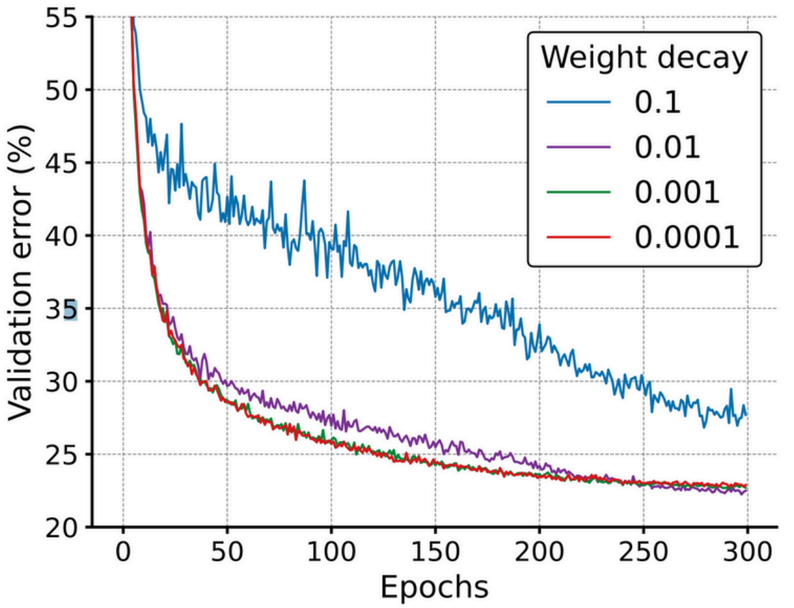
Attention calculation for each patch size

- If the patch size is larger than the kernel size, there may be pixels that cannot be encoded.

Impact of weight decay



Training error

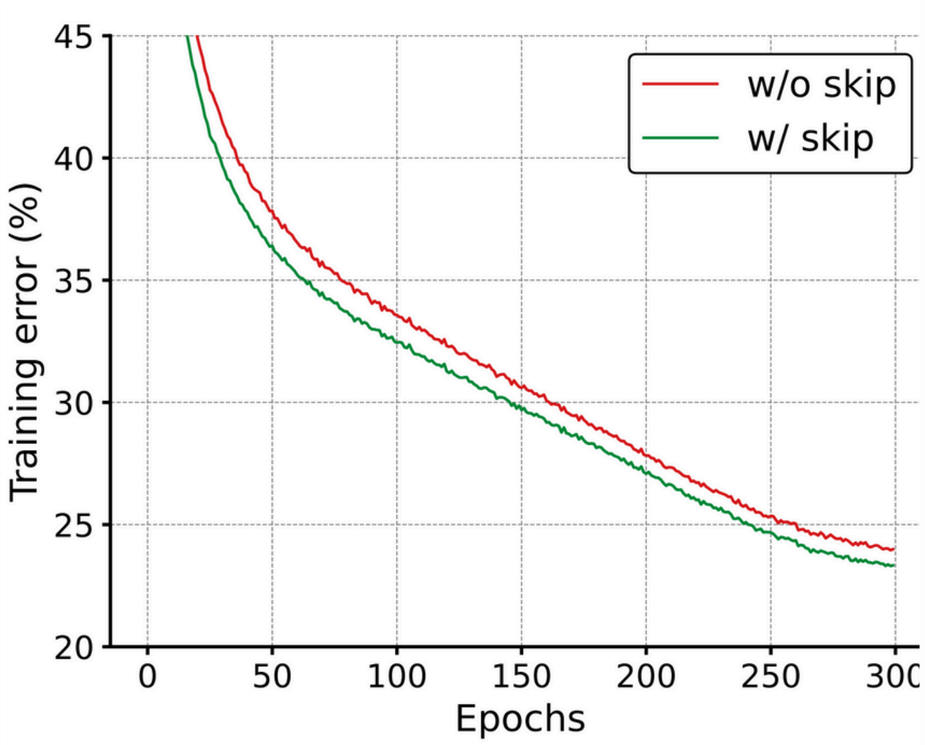


Validation error

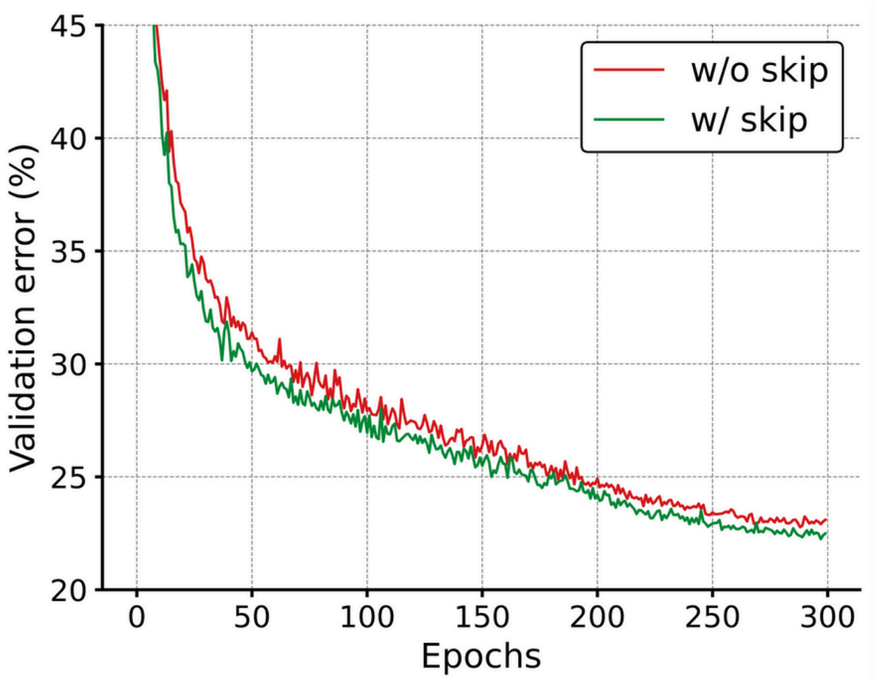
Weight decay	Top-1 ⇑
0.1	76.5
0.01	**78.4**
0.001	77.6
0.0001	77.4

Validation accuracy

Impact of skip connection



Training error



Validation error

Skip-connection	Top-1 ⇑
✗	77.9
✓	**78.4**

Validation accuracy

Porting MobileViT to on-device

Raspberry Pi 5

Samsung Galaxy Note 10+

https://www.researchgate.net/figure/Trend-of-sizes-of-state-of-the-art-NLP-models-with-time-Source-Using-DeepSpeed-and_fig3_37022579
https://bestphonemall.com/NOTE/?idx=190

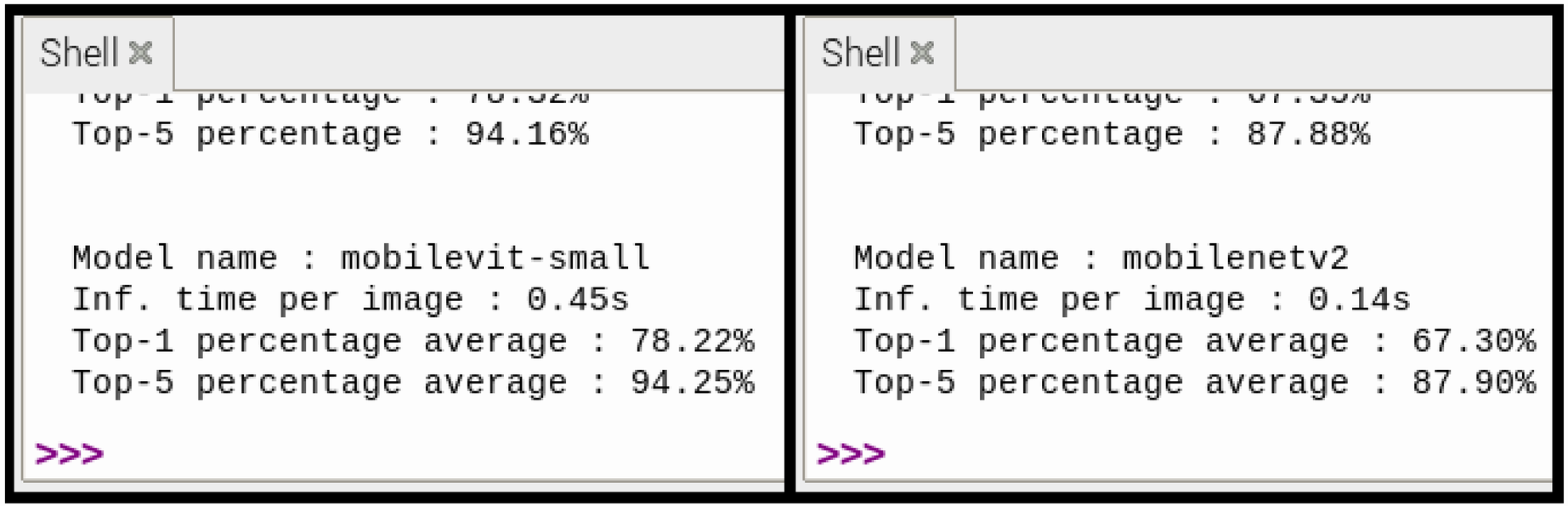
Implementation Environment

Raspberry Pi 5

항목		내용
H/W	CPU	BCM2712 (2.4GHz)
	GPU	VideoCore VII (800MHz)
	MEMORY	SDRAM 4267
	SD card	micro 카드 슬롯, SDR104 고속 모드 지원
S/W	O/S	Debian GNU/Linux 12
	Library	Pytorch=2.2.2 timm=0.9.16

Implementation Result

Raspberry Pi 5



(a)MobileViT (b)MobileNetv2

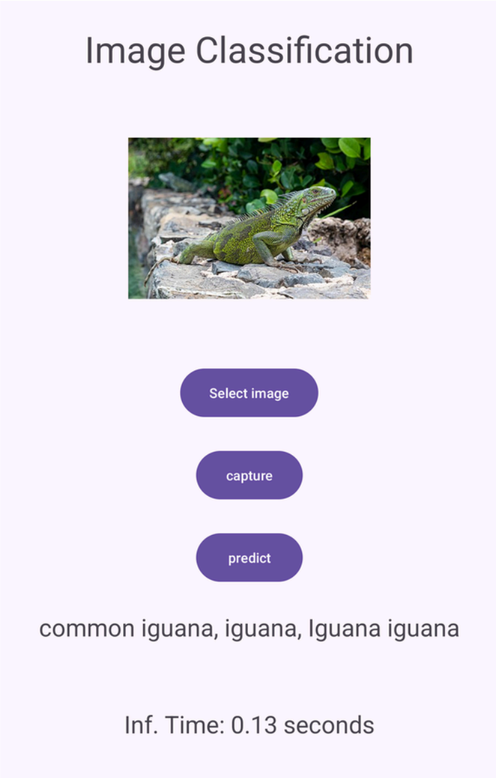
Implementation Environment

Galaxy Note 10+

항목		내용
H/W	CPU	Exynos 9825 (2.7GHz)
	GPU	Mali-G76 MP12 (754MHz)
	MEMORY	SDRAM
S/W	O/S	Android 12
	Program	Android Studio IDE(Android Studio 2023.2.1) SDK 24 이상(Android 7.0 이상)

Implementation Result

Mobile(Galaxy Note 10+)



- Select image: Select the image to inference
- capture: Capture images to use for inference.
- predict: Run inference with the selected image.
- As a result of inference, the class name and inf. time appears.

Written by Kim EunHo

Q/A