2021. 09. 16.

MobileNetV2

: Inverted Residuals and Linear Bottlenecks

인하대학교 정보통신공학과
12181879 이동건

목차

"모바일에 적합한 네트워크를 만들자!"

Resource ⬇️

Accuracy ⬆️

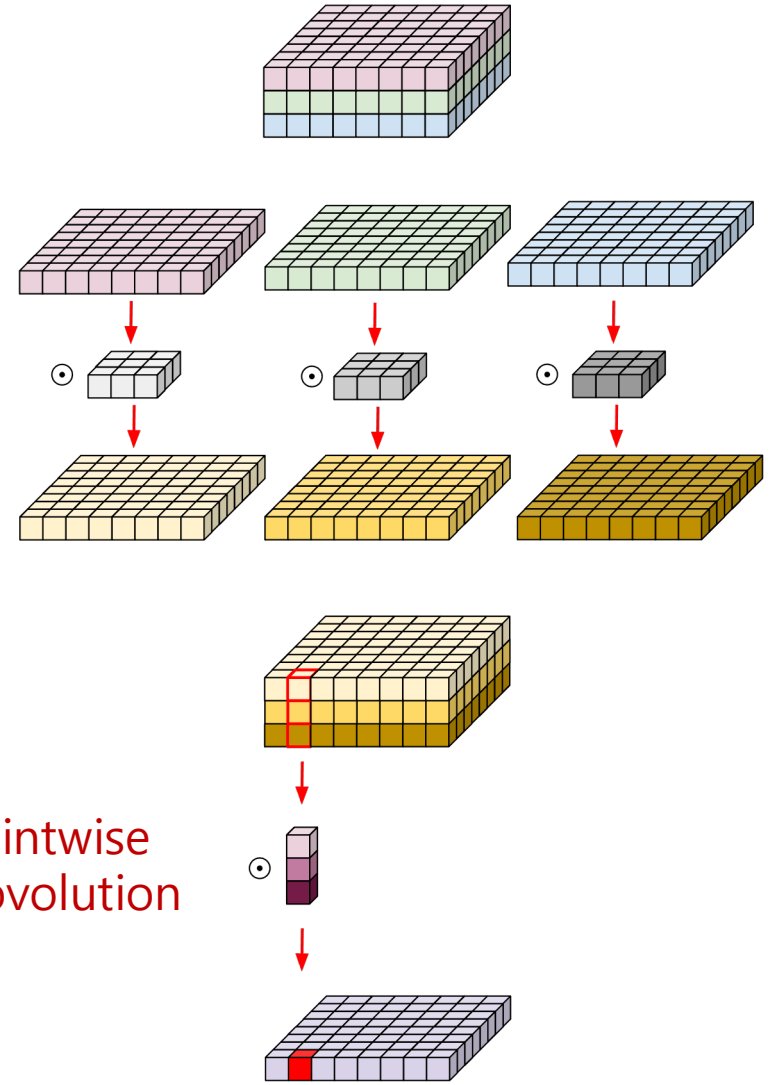
Speed ⬆️

Key Features

- Depthwise Seperable Convolution
- Linear Bottlenecks
- Inverted Residuals

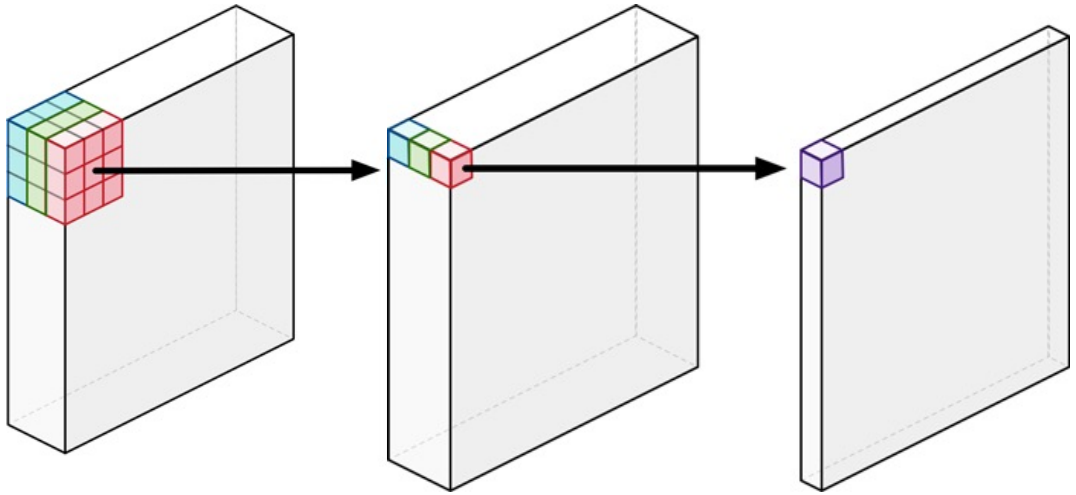
Recap – MobileNetV1; Depthwise Seperable Covolution (Depthwise conv + Pointwise conv)

Depthwise Covolution



Pointwise Covolution

Depthwise Covolution $w \times h \times d$ (1)의 kernel을 이용해서 입력 이미지의 각 Channel별로 Convolution



$w(1) \times h(1) \times d$의 kernel을 이용해서 Convolution (기존의 1x1 conv와 동일)

Pointwise Covolution

https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/

https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Recap – MobileNetV1; Computation

Standard Convolution : $h_i \times w_i \times d_j \times k \times d_i$

Depthwise Seperable Convolution : $h_i \times w_i \times d_j \times (k^2 + d_j)$

Reduction in computations : $\frac{1}{d_j} + \frac{1}{k^2}$

* $h$eight, $w$idth, $d$epth, $k$ernel size

Recap – MobileNetV1; Width Multiplier & Resolution Multiplier

Width Multiplier (α) : Thinner Models

- input과 output의 채널에 곱해지는 상수값 $(1, 0.75, 0.5, 0.25)$

Resolution Multiplier (ρ) : Reduced Representation

- Input image의 height와 width에 곱해지는 상수값

Recap – MobileNetV1; Architecture

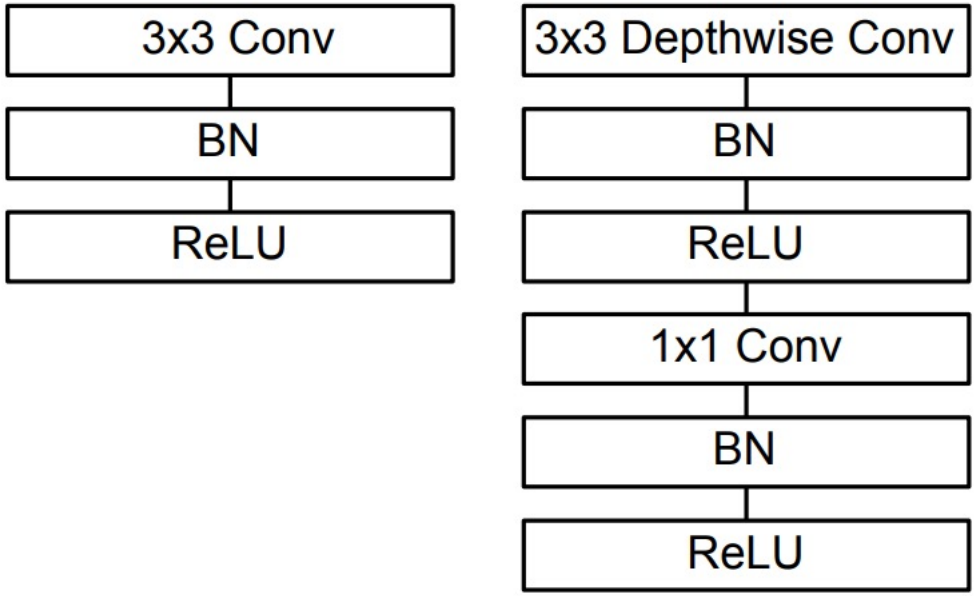
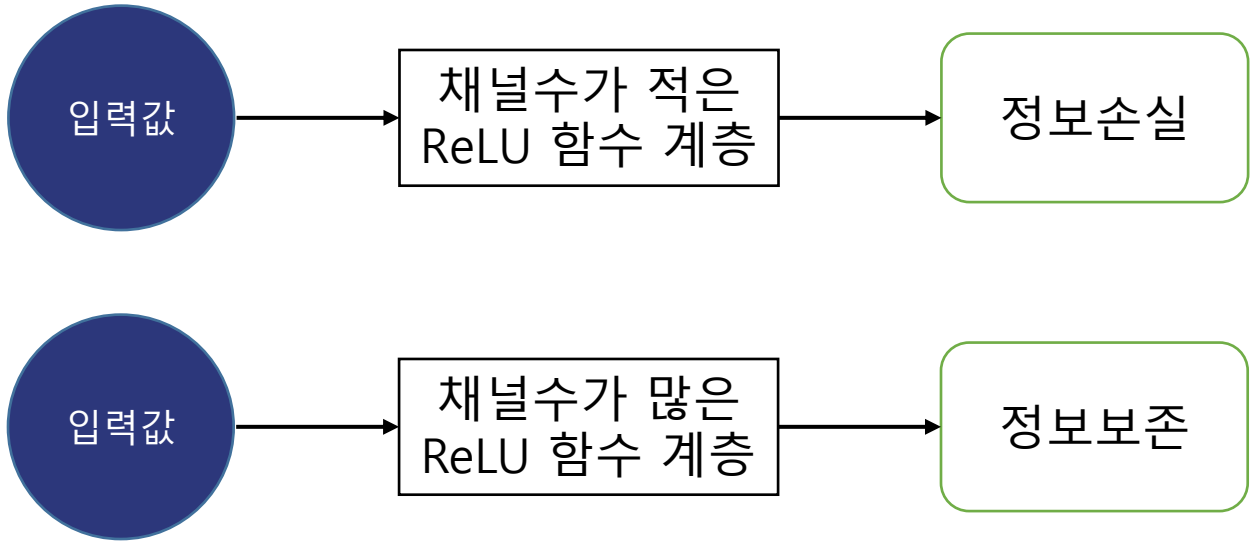


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

ReLU로 인한 정보 손실



ReLU 함수를 사용할 때는 해당 레이어에 많은 채널 수를 사용하고,
해당 레이어에 채널 수가 적다면 선형 함수를 사용

ReLU로 인한 정보 손실

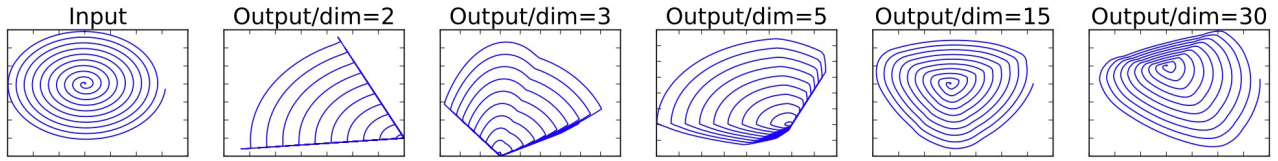
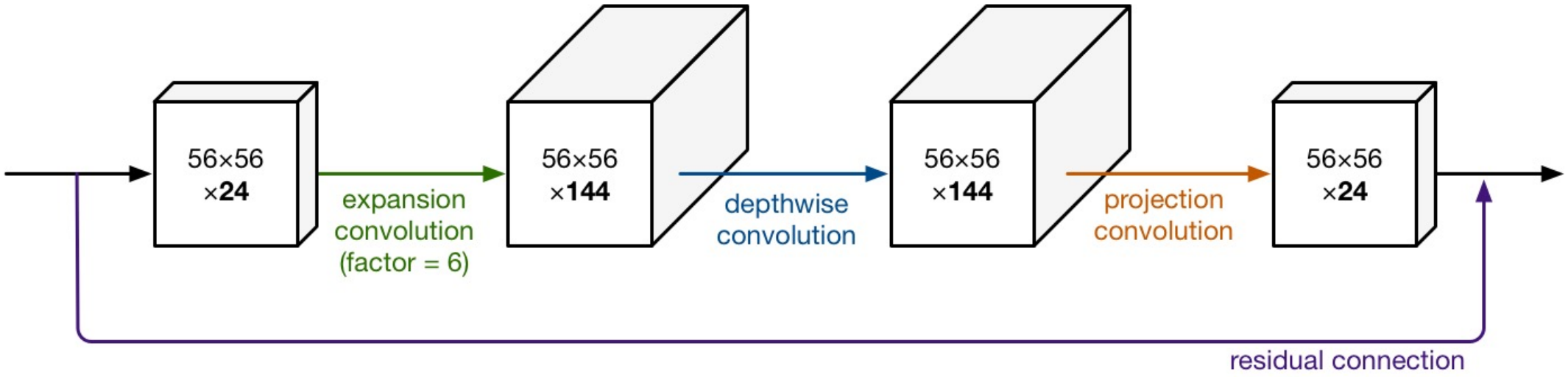


Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an n-dimensional space using random matrix T followed by ReLU, and then projected back to the 2D space using T^{-1}. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

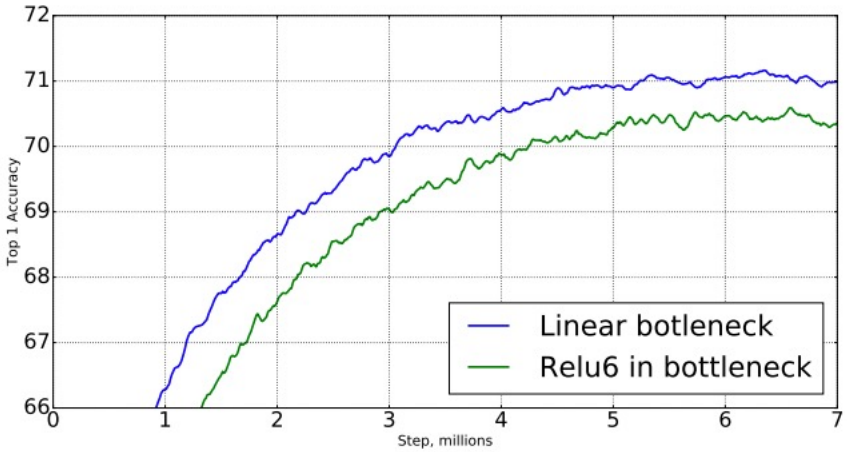
- 입력값에 ReLU 함수를 적용한 그림
- 출력 채널이 작은 경우에 정보가 손실되고,
  크면 클수록 정보를 보존한다는 것을 실험적으로 증명

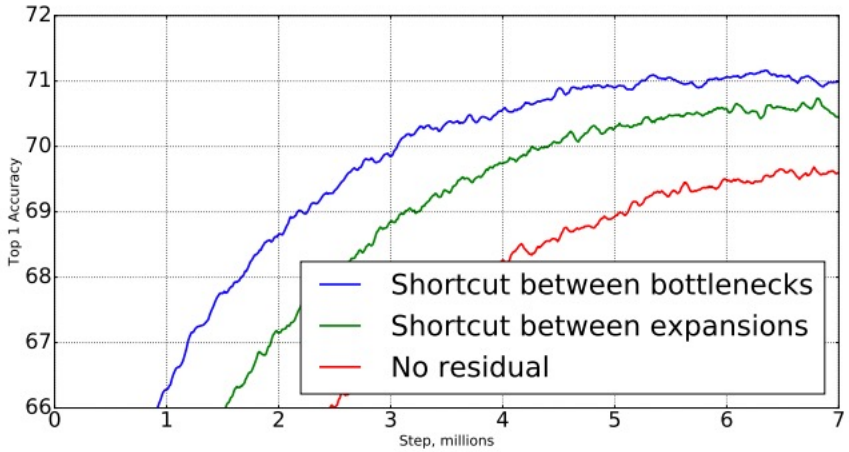
Linear Bottlenecks



https://machinethink.net/blog/mobilenet-v2/

Linear Bottlenecks





(a) Impact of non-linearity in the bottleneck layer.

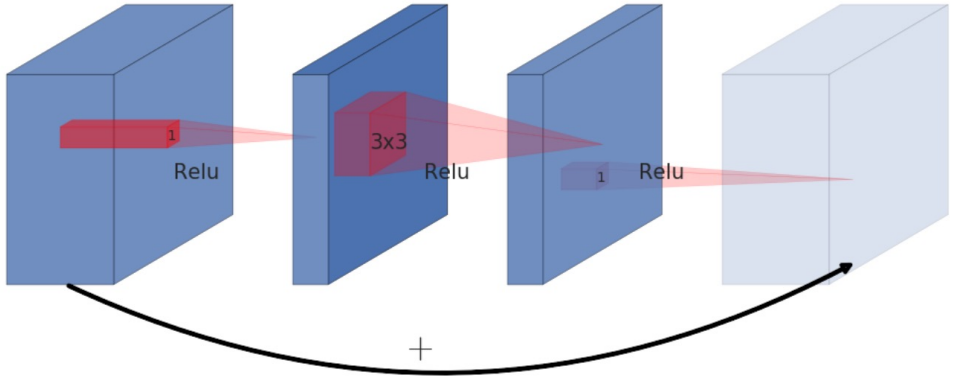
(b) Impact of variations in residual blocks.

Figure 6: The impact of non-linearities and various types of shortcut (residual) connections.

Residual Blocks

일반적인

(a) Residual block



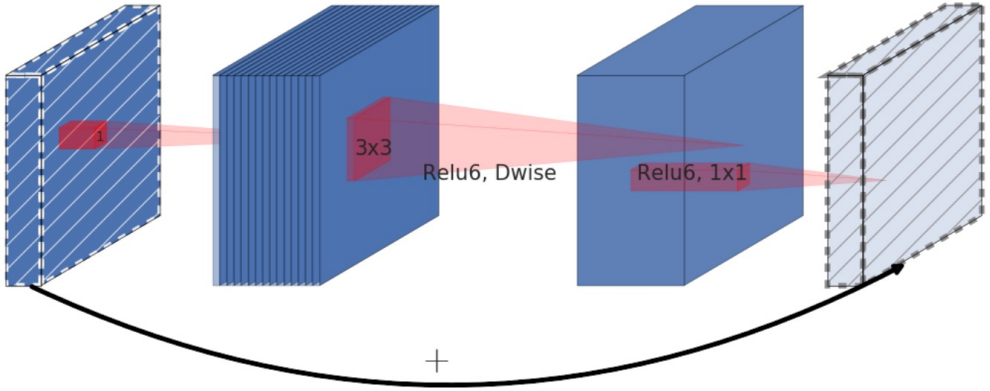
wide → narrow → wide

(bottleneck)

Inverted Residuals

narrow → wide → narrow

(b) Inverted residual block

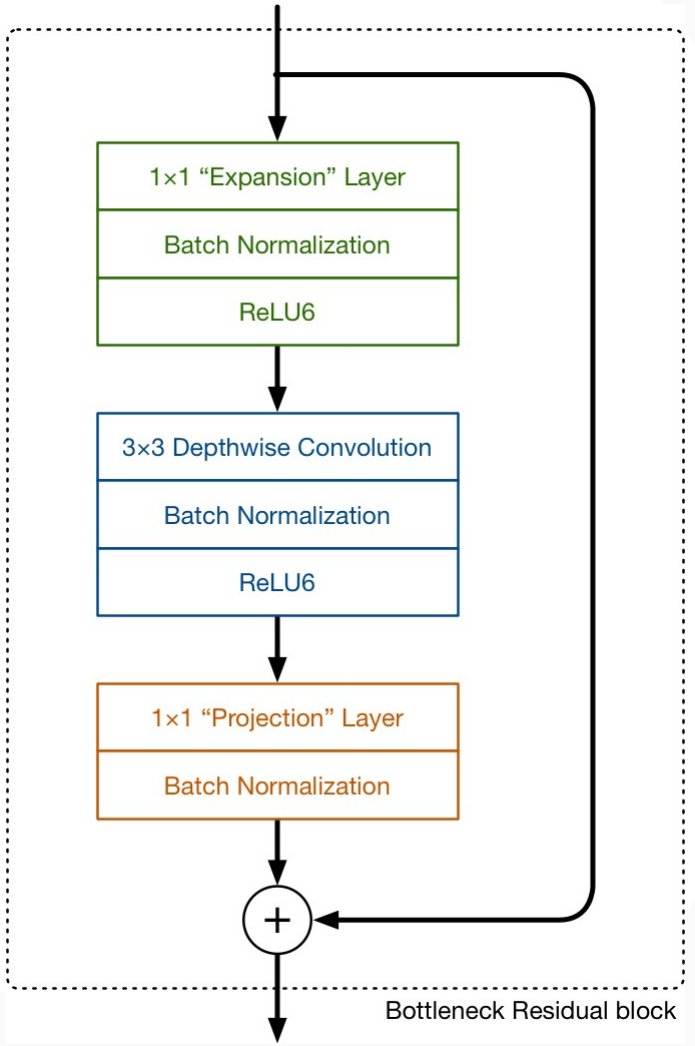


Bottleneck Residual Blocks

The basic building block
: a bottleneck depth-seperable convolution with residuals

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.



https://machinethink.net/blog/mobilenet-v2/

Operations

- $h \times w$: input의 크기
- t : expansion factor
- k : kernel size
- d' : input의 channel 수
- d'' : output의 channel 수

$$h \times w \times t \times d' \times d' + h \times w \times t \times d' \times k \times k + h \times w \times d'' \times t \times d'$$
$$= h \cdot w \cdot d' \cdot t(d' + k^2 + d'')$$

https://machinethink.net/blog/mobilenet-v2/

Model Architecture

- t : expansion factor
- c : output channel의 수
- n : 반복 횟수
- s : stride

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

The Max Number of Channels/Memory (in Kb)

Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	64/1600	16/400	32/800
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	1600K	**400K**	600K

Table 3: The max number of channels/memory (in Kb) that needs to be materialized at each spatial resolution for different architectures. We assume 16-bit floats for activations. For ShuffleNet, we use $2x, g = 3$ that matches the performance of MobileNetV1 and MobileNetV2. For the first layer of MobileNetV2 and ShuffleNet we can employ the trick described in Section 5 to reduce memory requirement. Even though ShuffleNet employs bottlenecks elsewhere, the non-bottleneck tensors still need to be materialized due to the presence of shortcuts between the non-bottleneck tensors.

Performance

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

ImageNet Classification Results

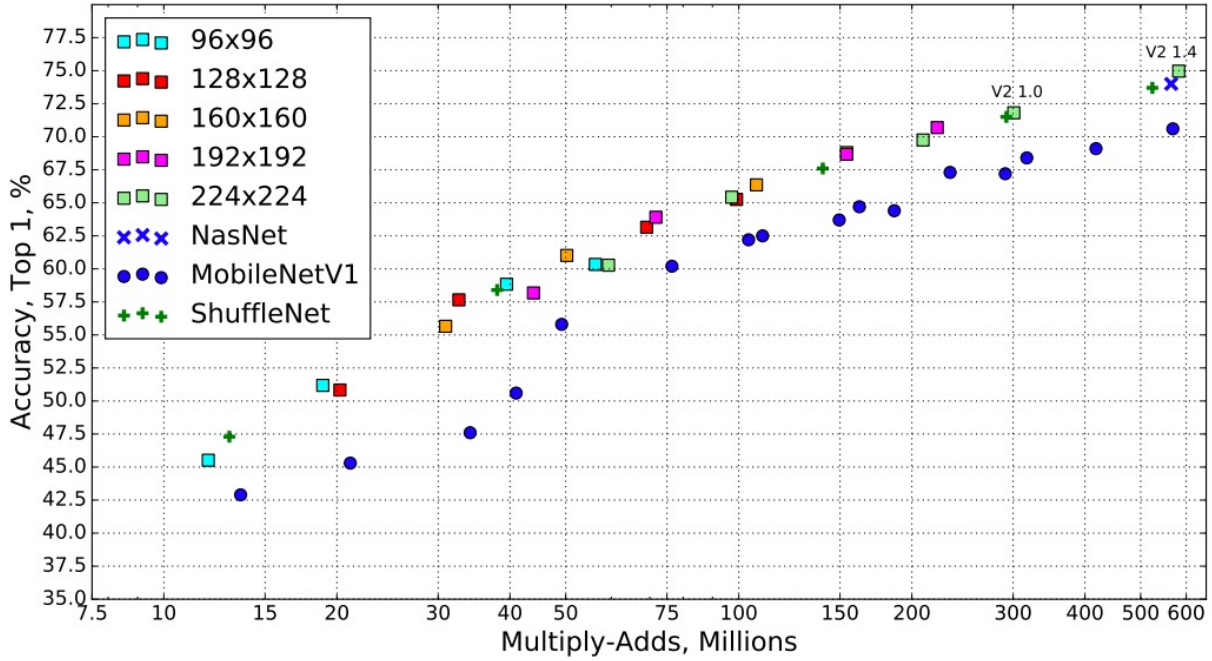


Figure 5: Performance curve of MobileNetV2 vs MobileNetV1, ShuffleNet, NAS. For our networks we use multipliers 0.35, 0.5, 0.75, 1.0 for all resolutions, and additional 1.4 for for 224. Best viewed in color.

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

Performance

Version	MACs (millions)	Parameters (millions)
MobileNet V1	569	4.24
MobileNet V2	300	3.47

Version	iPhone 7	iPhone X	iPad Pro 10.5
MobileNet V1	118	162	204
MobileNet V2	145	233	220

Version	Top-1 Accuracy	Top-5 Accuracy
MobileNet V1	70.9	89.9
MobileNet V2	71.8	91.0

https://machinethink.net/blog/mobilenet-v2/

Reference

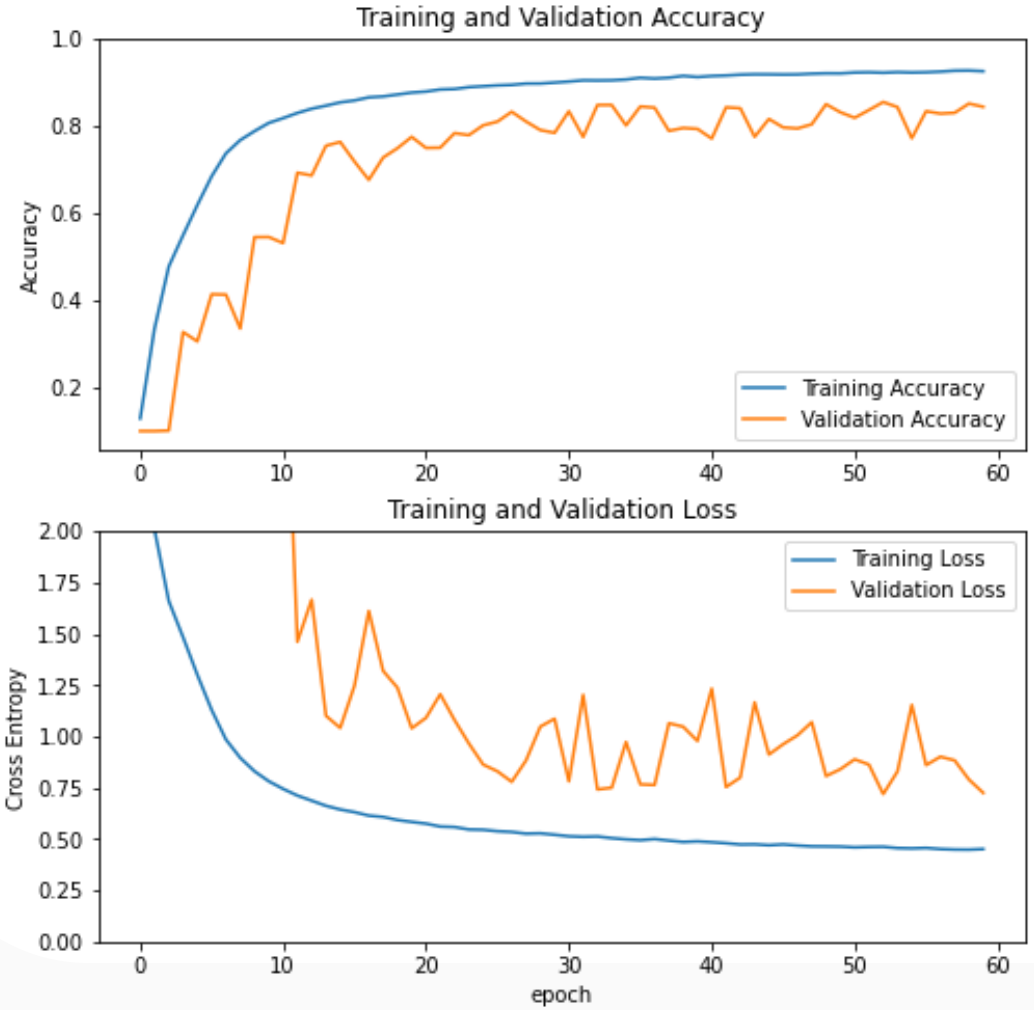
많은 도움
https://www.youtube.com/watch?v=mT5Y-Zumbbw
https://gaussian37.github.io/dl-concept-mobilenet_v2/

큰 도움
https://seing.tistory.com/58
https://deep-learning-study.tistory.com/541

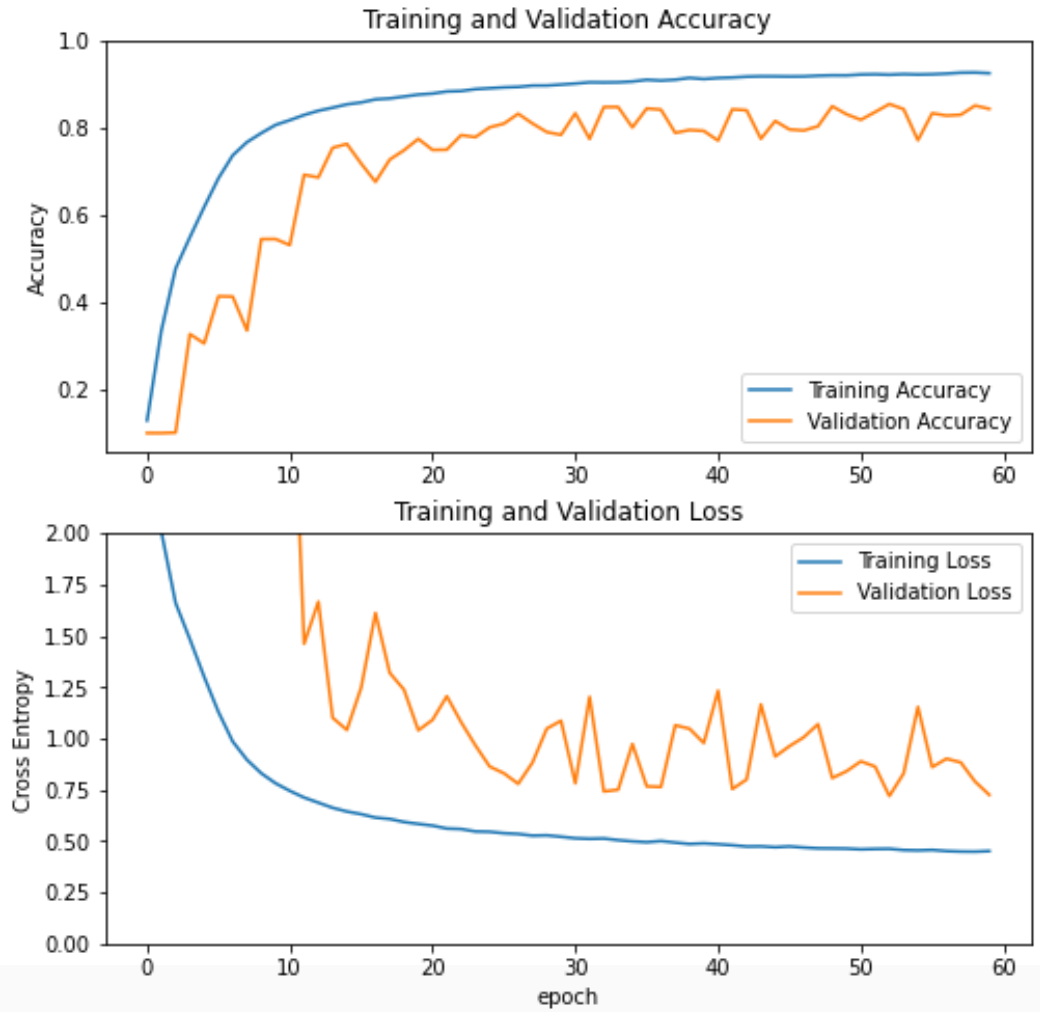
도움
https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/
https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/
https://machinethink.net/blog/mobilenet-v2/

Code Review

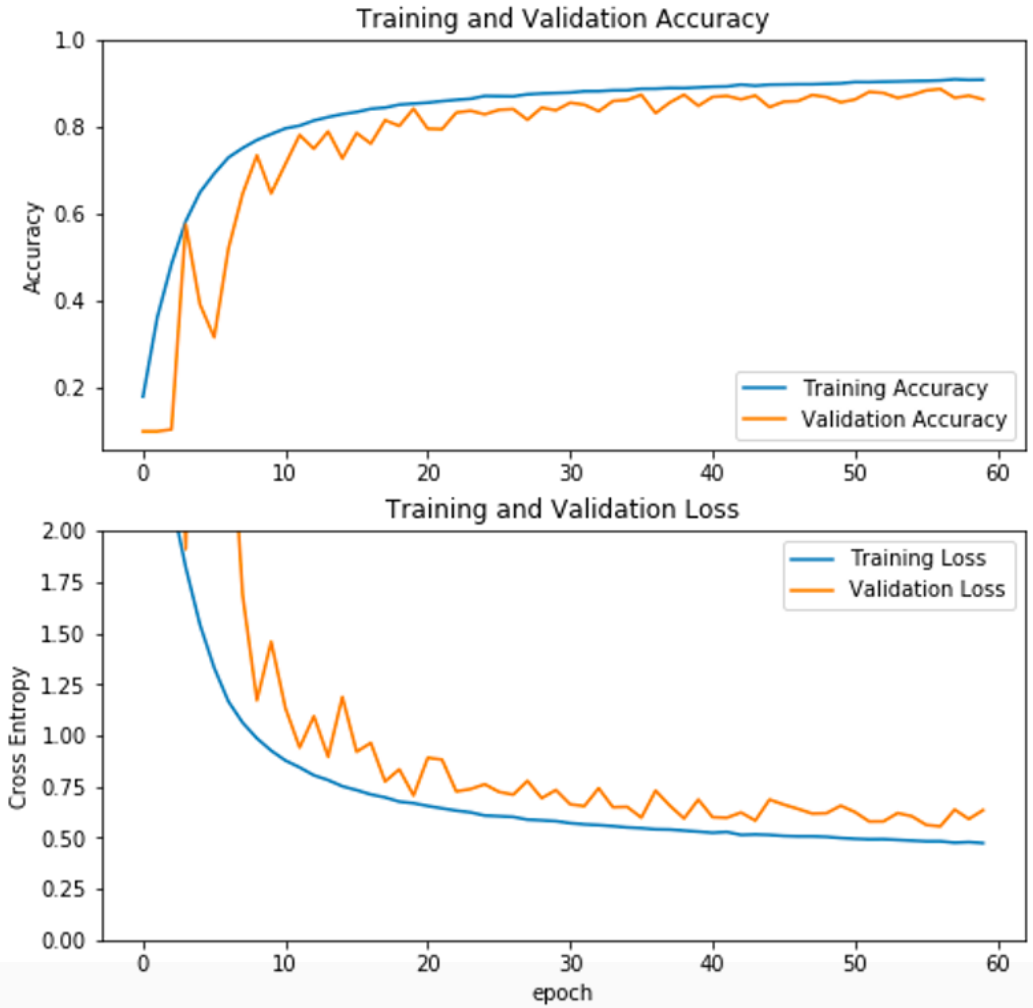
MobileNetV1 vs MobileNetV2



MobileNetV1



MobileNetV2



Reference

- MobileNetV1
  1. https://github.com/ruchi15/CNN-MobileNet-CIFAR10
     - https://colab.research.google.com/github/ruchi15/CNN-MobileNet-CIFAR10/blob/master/CNN_Project_1_MobileNet_Design.ipynb
  2. https://deep-learning-study.tistory.com/549
     - https://colab.research.google.com/github/Seonghoon-Yu/Paper_Review_and_Implementation_in_PyTorch/blob/master/Classification/MobileNetV1(2017).ipynb

- MobileNetV2
  1. https://github.com/ruchi15/CNN-MobileNetV2-Cifar10
     - https://github.com/ruchi15/CNN-MobileNetV2-Cifar10/blob/master/CNN_Assignment_5_Design.ipynb
  2. https://www.youtube.com/watch?v=VO58tq3M1Jc
     - https://github.com/bigdatachobo/Study/blob/master/Deep_Learning/source/Parking_Space_Finder/Vacant_Parking_Space_Finder.ipynb