MobileNetV2

20176359 신수현

MobileNetV2 Key Features

- Depthwise Separable Convolutions

: This dramatically reduce the complexity cost and model size of the network, which is suitable to Mobile devices, or any devices with low computational power

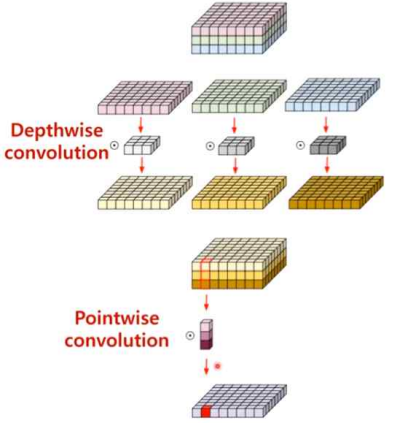
- Linear Bottlenecks

- Inverted Residuals

MobileNet에서는 무엇을 보완하고자 하는가?

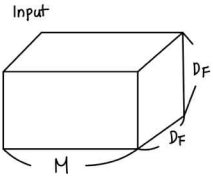
- 메모리 사용량의 감소와 네트워크 자체의 복잡성을 해결하고자 한다.
- Inverted Residuals와 Linear Bottlenecks는 ReLU 함수를 거치게 되면 정보가 손실된다는 것에 영감을 받아 이를 최소화하기 위해 제안되었다.
- ReLU 함수를 사용할 때는 해당 레이어에 많은 채널 수를 사용하고, 해당 레이어에 채널 수가 적다면 선형 함수를 사용한다. 이것이 Inverted Residuals와 Linear Bottleneck의 등장 배경

MobileNetV1의 Recap



- 세로 방향이 채널
- 채널 하나마다 각각 다른 filter로 convolution 후 computation : Depthwise convolution
- 이후 1x1 convolutional filter 적용 : pointwise convolution
- 결과적으로 Standard convolution Filter를 적용한 것과 같다
- 파라미터 개수를 줄이는 것

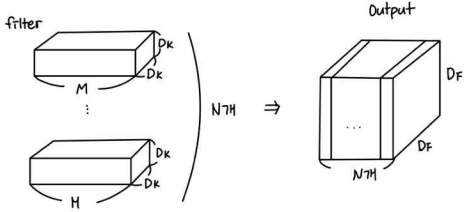
i) 일반 CNN 레이어의 처리



〈 계산량 〉

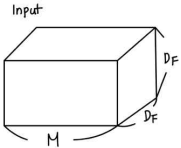
filter 1개당 parameter : $D_K \cdot D_K \cdot M$ 개 · N 개 · $D_F \cdot D_F$ (입력되는 사진의 가로세로)

이전 filter가 총 N개 존재



VS

ii) Depthwise Separable Convolution

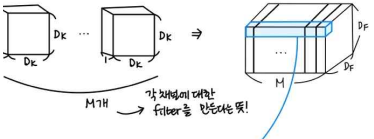


(계산량) 입력 면적

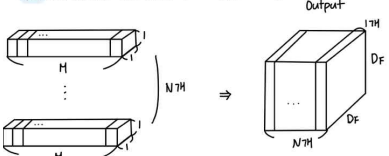
$D_K \cdot D_K \cdot 1 \cdot M \cdot D_F \cdot D_F + 1 \cdot 1 \cdot M \cdot N$개 $\cdot D_F \cdot D_F$

filter parameter · M개 (filter 개수) · 1x1 filter parameter

(출력 채널)



1x1 짜리 convolutional filter (with channel M)



카운트 ($\frac{D \cdot S \cdot C \text{ 계산량}}{NN \text{ 계산량}}$) = $\frac{1}{N} + \frac{1}{D_K^2}$ 만큼 계산량 감소

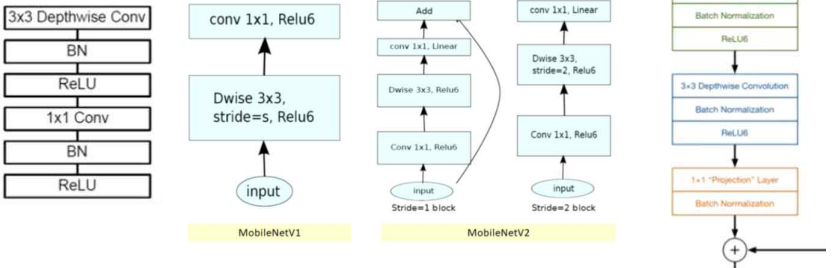
Convolution Block for MobileNetV2

그림은 MobileNetV1과 MobileNetV2에서 사용하는 Convolution block 이다.
stride=2는 down sampling을 위해 사용하는 블락이다.

first layer : 1×1 convolution with ReLU6 (expansion layer)

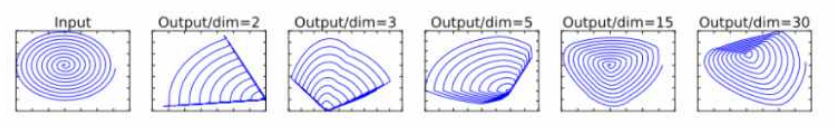
second layer : depthwise convolution

third layer : another 1×1 convolution but without any non-linearity.



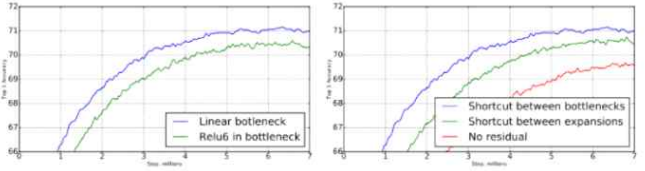
입력값과 정보의 비례관계

- 아래 그림은 입력값에 ReLU 함수를 적용한 것이다. 출력 채널이 작은 경우에 정보가 소실되고, 크면 클수록 정보를 보존한다는 것을 실험적으로 증명



(1) Linear Bottlenecks

Bottleneck 구조는 ResNet에서 연산량 감소를 위해 제안된 구조이다. 이 BottleNeck 구조에 채널 수가 적은 레이어는 linear 함수를 사용한다. 채널 수가 적은 레이어에 비선형 함수 ReLU를 사용하면 정보손실이 발생하기 때문이다. 아래 그림은 비선형 함수와 선형 함수를 사용했을 때의 성능 표이다. 선형 함수의 성능이 더 좋다.



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

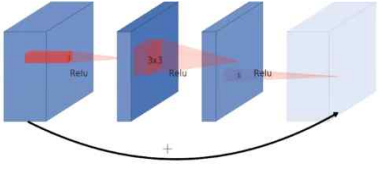
(b) Impact of variations in residual blocks.

(2) Inverted residuals

- 기존의 BottleNeck 구조는 첫 번째 1x1 conv layer에서 채널 수를 감소시키고 3x3 conv로 전달한다.
- 채널 수가 감소된 레이어에서 ReLU 함수를 사용하면 정보 손실이 발생하게 된다.
- 따라서 첫 번째 레이어에서 입력값의 채널 수를 증가시키고 3x3conv layer로 전달한다.

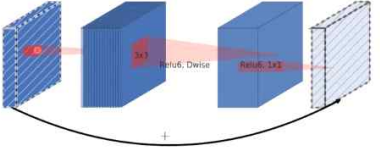
〈기존 Residual Block〉

wide → narrow(bottleneck) → wide approach



〈Inverted Residual Block〉

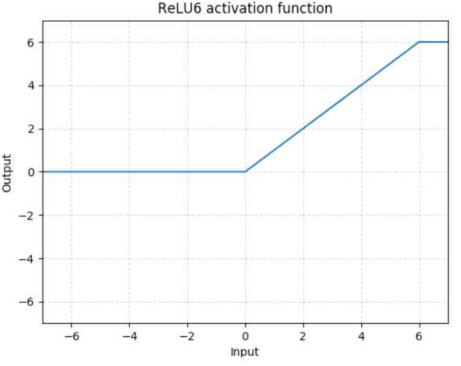
narrow → wide → narrow approach



빗금은 ReLU 없이 Linear 이용했다는 의미

(3) ReLU6

- MobileNetV2와 MobileNetV1에서는 ReLU6 함수를 사용한다. 연산량 감소 효과가 있다.



MobileNetV2 convolution block 연산

- 최종적으로 MobileNetV2 convolution block은 아래와 같은 연산을 한다.

- k채널 입력값은 1x1 conv를 거쳐서 tk 채널로 확장되고 3x3conv에 전달된다. 그리고 linear 1x1 conv를 거쳐서 k'로 채널 수가 감소한다.

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	$\frac{h}{s} \times \frac{w}{s} \times k'$

MobileNetV2 Architecture

- 첫 번째 레이어는 일반적인 conv를 사용하고, 그 이후에 18개의 convolution block을 쌓는다.

t: expansion factor
c: number of output channels
n: repeating number
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	-	

MobileNetV2 Trade-off Hyper Parameters

- Input Resolution 96x96 to 224x224 (Input 이미지 비율 조정)
- Width Multiplier 0.35 to 1.4 (layer의 채널 수 조정)

Memory Efficient?

- The amount of memory is simply the maximum total size of combined inputs and outputs across all operations
- If we treat a bottlenect residual block as a single operation(and treat inner convolution as a disposable tensor), the total amount of memory would be dominated by the size of bottleneck tensors, rather than the size of tensors that are internal to bottleneck(and much larger)

Hyperparameters

- weight decay 0.00004
- learning rate 0.045 (initial), learning rate decay rate of 0.98 per epoch
- batch size 96
- batch normalization for every layer
- momentum 0.9

학습 이미지

- 이처럼 학습 데이터를 다양한 크기로 변환하고 그 중 일부분을 샘플링해 사용함으로써 몇 가지 효과를 얻을 수 있다.

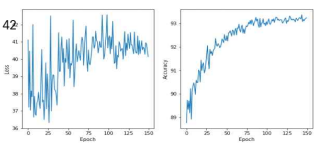
- 한정적인 데이터의 수를 늘릴 수 있다. — Data augmentation 하나의 오브젝트에 대한 다양한 측면을 학습 시 반영시킬 수 있다. 변환된 이미지가 작을수록 개체의 전체적인 측면을 학습할 수 있고, 변환된 이미지가 클수록 개체의 특정한 부분을 학습에 반영할 수 있다. 두 가지 모두 Overfitting을 방지하는 데 도움이 된다.

```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])
```

trainset에 대해서 mean과 std 이용하여 Normalize

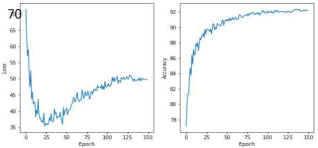
MobileNetV2 : test loss/accuracy

epoch 150, lr decay till 150
starting test accuracy : 80s, ReLU



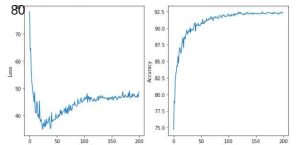
Best Test accuarcy: 93.37

epoch 150, lr decay till 150
starting test accuracy : 70s, ReLU



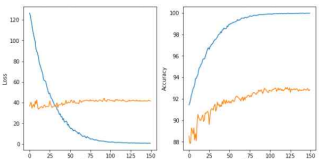
Best Test accuarcy: 92.35

epoch 200, lr decay till 150
starting test accuracy : 70s, ReLU6



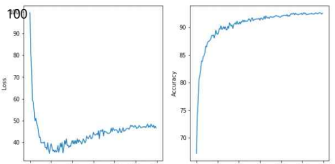
Best Test accuarcy: 92.47

epoch 150, lr decay till 150
starting test accuracy : 80s, ReLU6



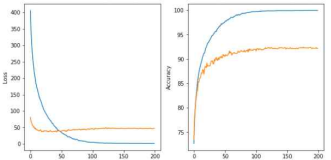
Best Test accuarcy: 93.04

epoch 150, lr decay till 150
starting test accuracy : 70s, ReLU6



Best Test accuarcy: 92.67

epoch 200, lr decay till 200
starting test accuracy : 70s, ReLU6



Best Test accuarcy: 92.49

참고 자료

- Jay S, “[AI 논문 해설] 모바일넷 MobileNet 알아보기”, https://www.youtube.com/watch?v=vi-_o22_NKA (2021.03.25)
- kuangliu github, “mobilenetv2.py, main.py 코드 참고”, https://github.com/kuangliu/pytorch-cifar, (2020.09.18)
- Adnan Siddiqi, “MobileNetV2” , https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c, (2019.5.19)