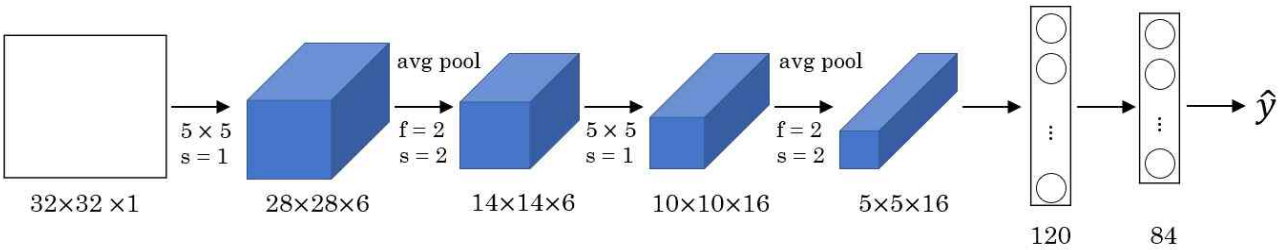
Deep Convolutional Models: Case Studies

LeNet-5

[LeCunet al., 1998.

Gradient-based learning applied to document recognition]

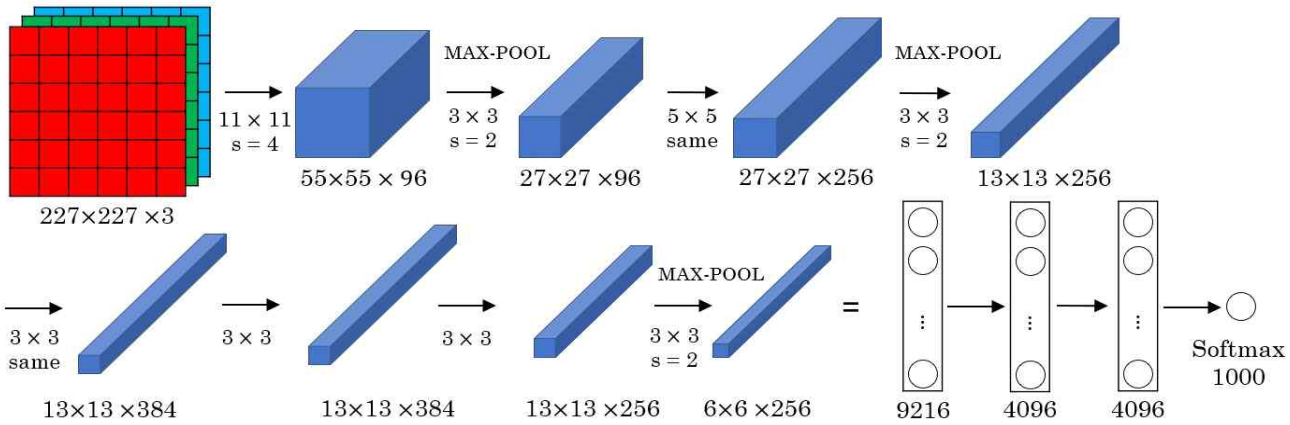


1. CONV-POOL-CONV-POOL-FC-FC 2. n_H, n_W ↓, n_C↑ 3. 60k parameters
이 논문이 나왔을 때는 Max Pooling보다 Avg Pooling을 많이 사용했다. 그리고 Padding이나 유용한 CONV를 사용하지 않았다. CONV layer를 적용할 때마다 장점이 많아졌기 때문이다.
요즘에는 마지막에 softmax를 output layer로 사용하지만, 당시에는 다른 분류기를 사용했다.

AlexNet

[Krizhevskyet al., 2012.

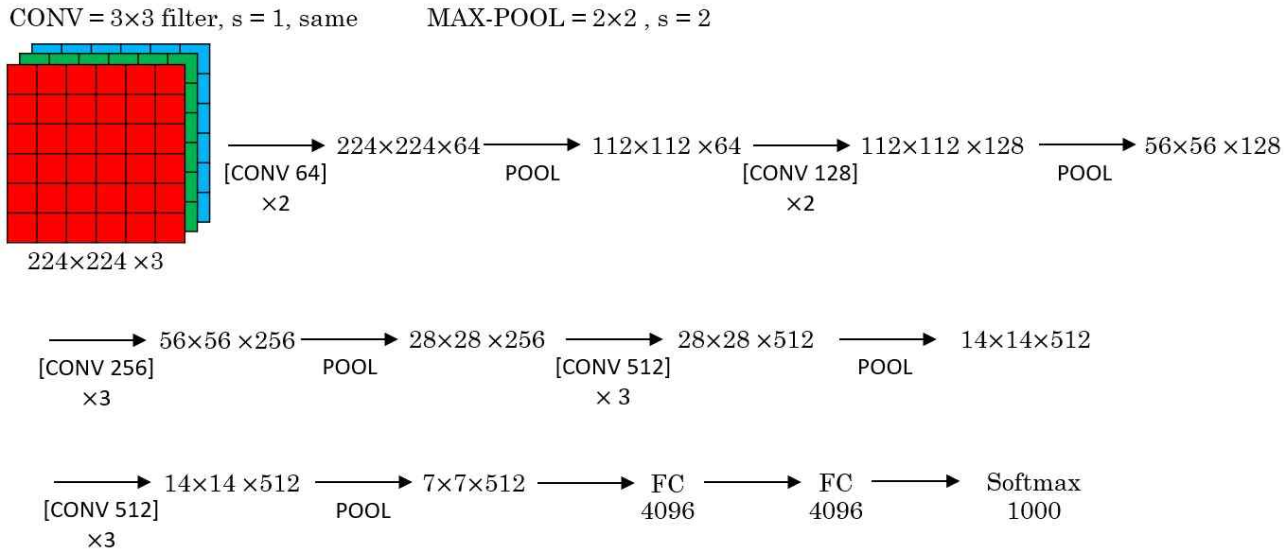
ImageNet classification with deep convolutional neural networks]



1. n_H, n_W ↓, n_C↑ 2. 60m parameters
1. Similary to LeNet, but much bigger 2. ReLU 3. Multiple GPUs 4. Local Response Normalization
논문에서는 224x224x3의 이미지를 사용하지만 실제로는 227x227x3 경우에 더 잘 들어맞는다.

VGG-16

[Simonyan& Zisserman 2015.
Very deep convolutional networks for large-scale image recognition]

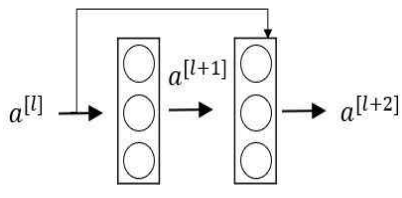


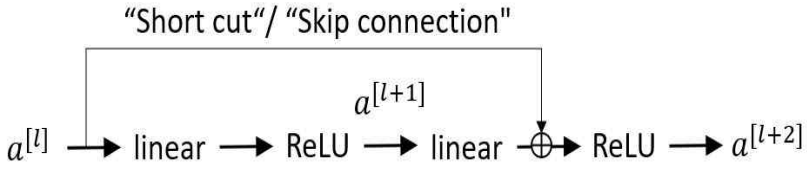
1. n_H, n_W ↓, n_C↑: 높이와 너비는 2배씩 줄어드는 반면 채널의 수는 2배씩 늘어난다. 2. 138m parameters
[CONV 64]x2: 채널 개수가 64개인 CONV 연산 2번, 즉 CONV-CONV-POOL
장점: 신경망 아키텍처를 단순화
VGG-19가 VGG-16보다 큰 버전이긴 한데 실제로는 비슷하여 많은 사람은 16을 사용한다.

ResNet(Residual Network)

[He et al., 2015. Deep residual networks for image recognition]
매우 깊은 심층 신경망은 Vanishing/Exploding gradient 문제 때문에 학습하기 어렵다.
ResNet은 activation뿐만 아니라 input을 훨씬 깊은 layer로도 제공한다.

Residual block & its main path





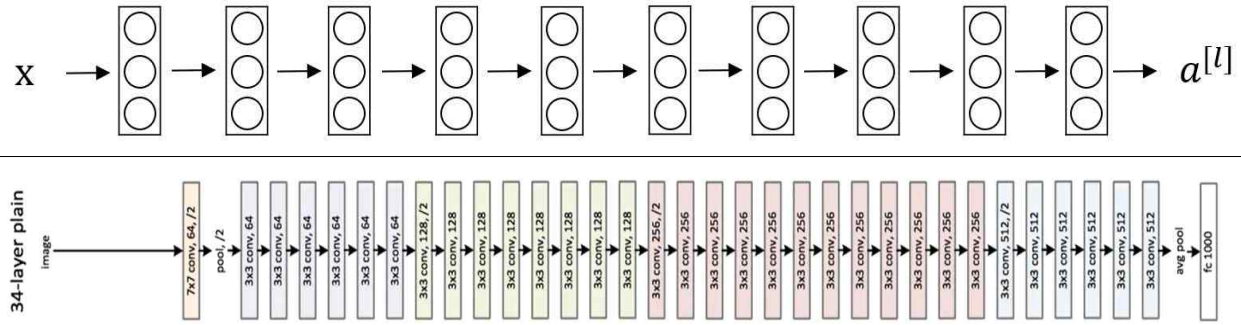
linear l: $z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$

activation l: $a^{[l+1]} = g^{[l+1]}(z^{[l+1]})$

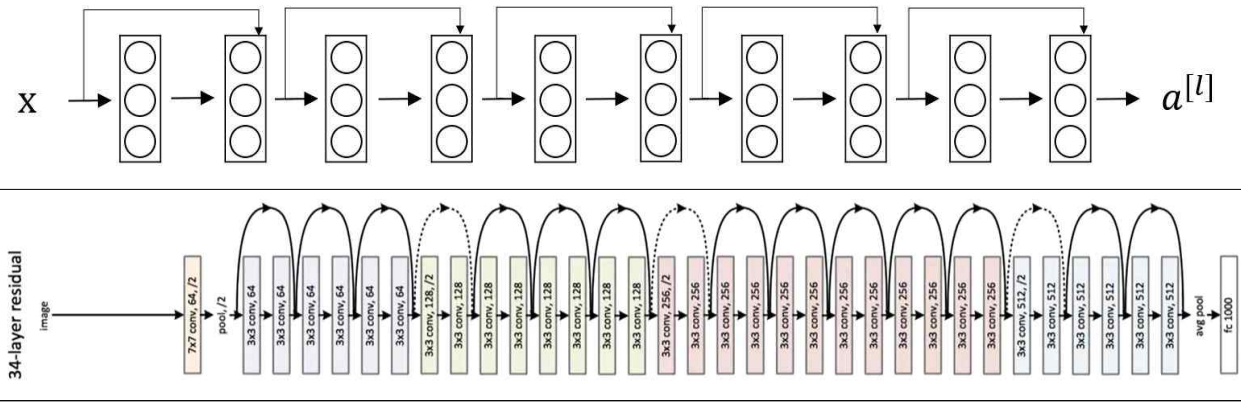
linear $l+1$: $z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$

activation $l+1$: $a^{[l+2]} = g^{[l+2]}(z^{[l+2]} + a^{[l]})$

Plain Network



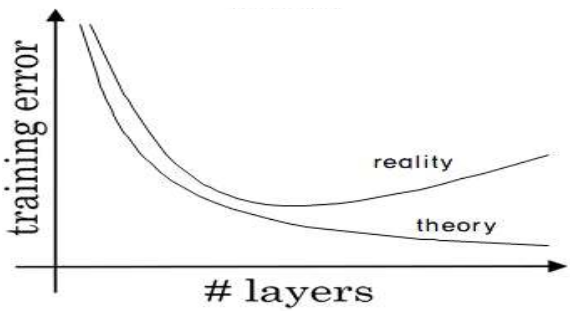
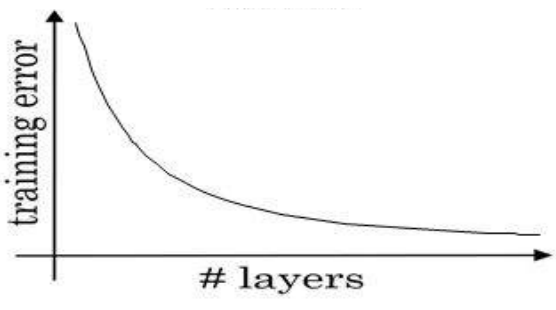
Residual Network



ResNet의 특징

1. 만약 activation $l+1$: $a^{[l+2]} = g^{[l+2]}(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$에서 $W^{[l+2]} = 0$, $b^{[l+2]} = 0$이라면 $a^{[l+2]} = g^{[l+2]}(a^{[l]}) = a^{[l]}$이고 항등함수는 residual block에서 학습하기 쉬우므로 residual block에서 두 layer가 없는 단순한 네트워크이든 두 layer를 더하든 신경망의 능력을 저하하지 않는다.

2. $z^{[l+2]}$와 $a^{[l]}$이 같은 차원을 가질 것이라고 가정한다. 사이즈 유지를 위해 same convolution을 사용한다. 만약 다른 차원이라면 $a^{[l+2]} = g^{[l+2]}(z^{[l+2]} + W_s a^{[l]})$로 처리하여 차원을 맞춰준다. W_s는 파라미터일 수도 있고 고정된 매트릭스일 수도 있다.

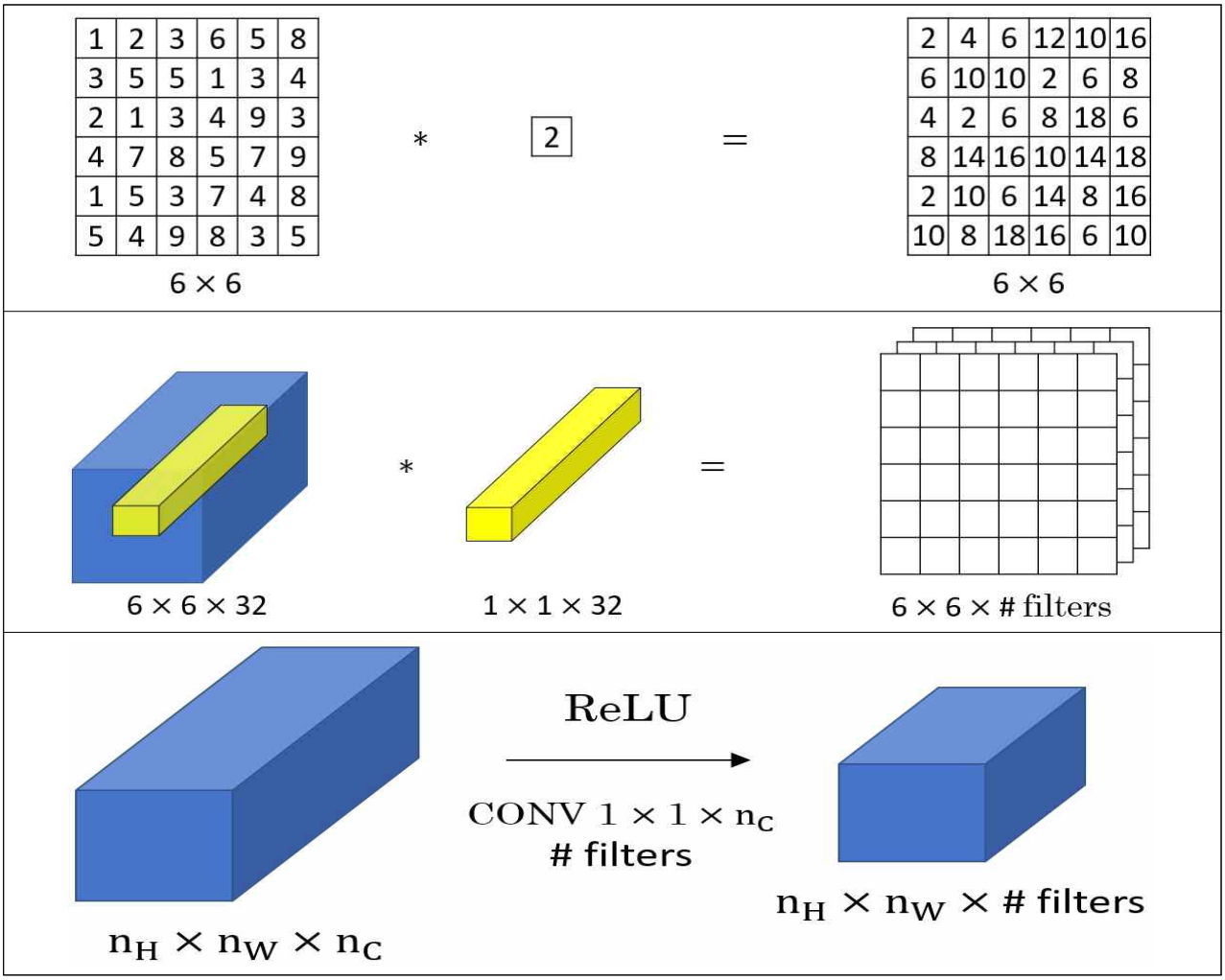
Plain Network	Residual Network
	

중간에 직접 activation value가 들어가서 gradient vanishing/exploding와 같이 학습 능력에 대한 손상을 해결했다. 수행 시 눈에 띄는 손실 없이 신경망을 훨씬 깊게 훈련하도록 해준다.

Network in Network: 1x1 convolutions

[Lin et al., 2013. Network in network]

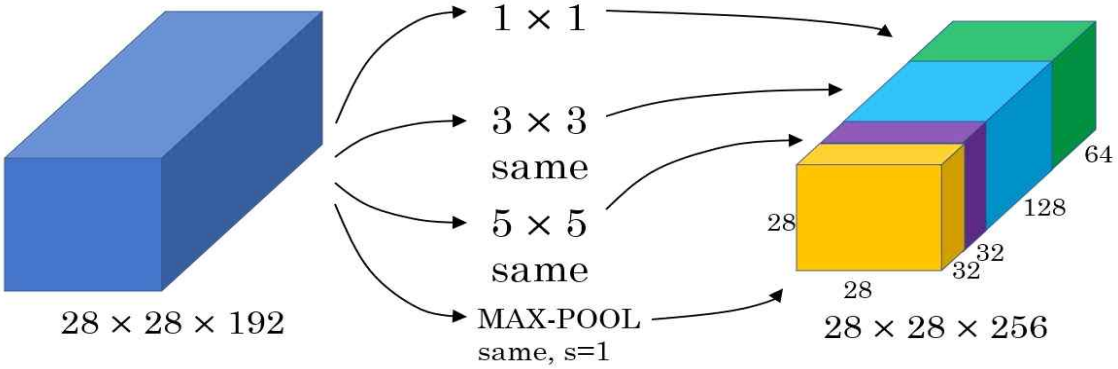
: FC layer를 $n_H \times n_W$ 개의 위치에 각각 적용해서 $n_H \times n_W$ 개의 숫자를 입력값으로 받고 필터의 수 만큼 출력한다.



1x1 convolution의 특징
1. 네트워크의 비선형성 더하기 2. 채널의 개수 조절
채널 개수를 유지해도 좋다. 1x1 Convolution의 효과는 비선형성을 더해주고 하나의 층을 더해줘서 더 복잡한 함수를 학습할 수 있다는 것이다.
pooling을 사용하면 height와 width를 줄일 수 있다. 1x1 필터를 사용하면 채널의 개수를 줄일 수 있다.

Inception network

motivation: 무엇을 할지 고민하는 대신 원하는 모든 filter와 pooling을 하고 모든 아웃풋을 연결한다. 원하는 파라미터가 무엇이든, 원하는 필터 사이즈가 무엇이든 다 학습한다.



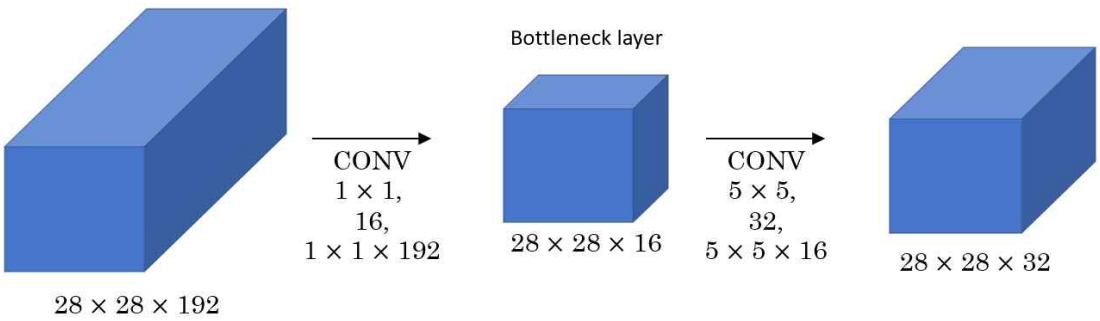
문제는 계산 비용이다.

보라색 아웃풋을 보면 32 filters 5x5x192이므로 computation cost가 28x28x32 x 5x5x192 = 120m이다.

그런데 여기에서 1x1 convolution을 사용하면 computation cost를 약 10배 감축할 수 있다. 약 1억 2천만에서 약 1/10배.

Using 1x1 convolution

bottleneck layer: inception network에서 가장 작은 부분으로
크기를 다시 늘리기 전에 representation을 작게 만든다.



첫 번째 CONV의 computation cost: 28x28x16 x 1x1x192 = 2.4m

두 번째 CONV의 computation cost: 28x28x32 x 5x5x16 = 10.0m

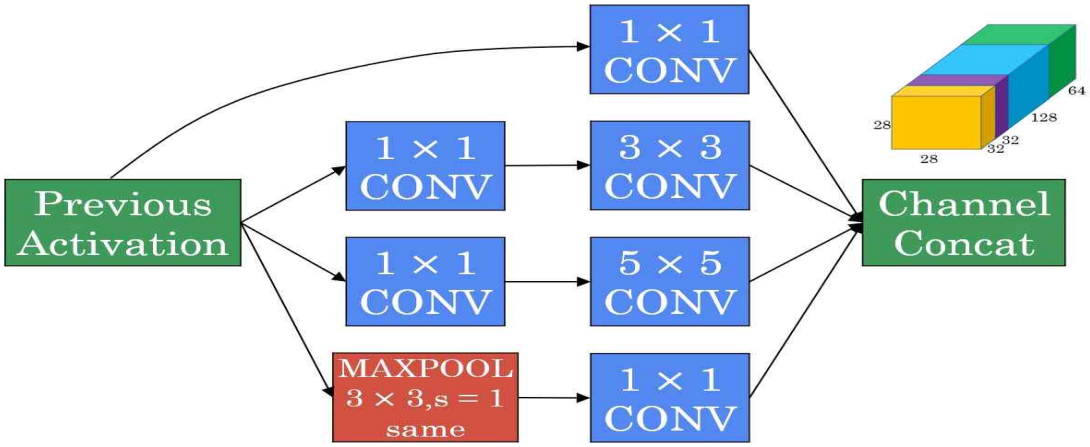
inception network using 1x1 CONV의 computation cost: 12.4m

필요한 덧셈의 횟수는 수행해야 하는 곱셈의 수와 매우 유사하므로 곱셈의 횟수를 셌다.

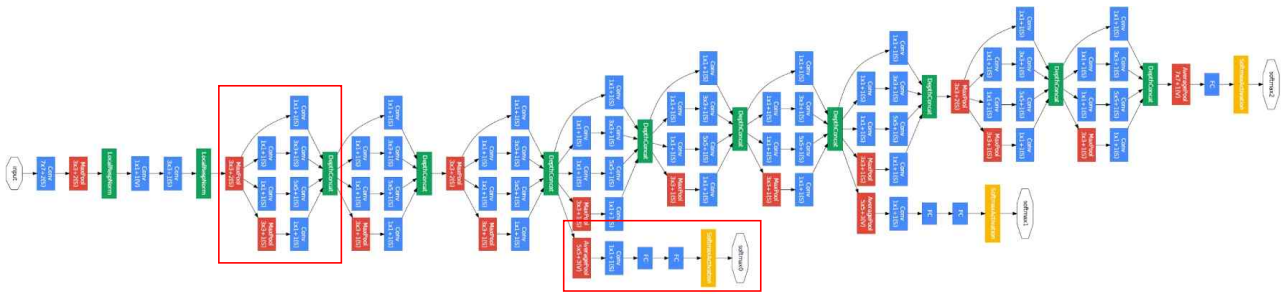
120m(1x1 CONV X) vs. 12.4m(1x1 CONV O)

1x1 CONV와 bottleneck layer로 계산 비용을 줄였다.

Inception module



Inception Network: GoogleNet



inception netwokr의 특징

side branch: 네트워크 중간에 Activation–CONV–FC–FC–Softmax로 ouput label을 예측하려고 시도한다. hidden unit이나 중간 layer에서 feature들에 대한 이미지의 아웃풋 예측을 도와준다. 그리고 inception network에 규칙적인 영향을 미치고 네트워크가 overfitting되는 것을 방지한다.

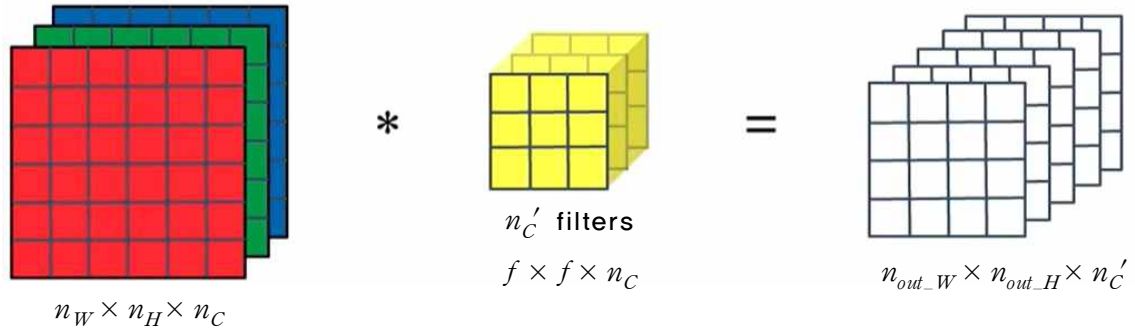
MobileNet

[Howard et al. 2017, MobileNets:
Efficient Convolutional Neural Networks for Mobile Vision Application]

motivation

1. Low computational cost at deployment.
2. Useful for mobile and embedded vision applications.
3. Key idea: Normal vs. depthwise separable comvolutions

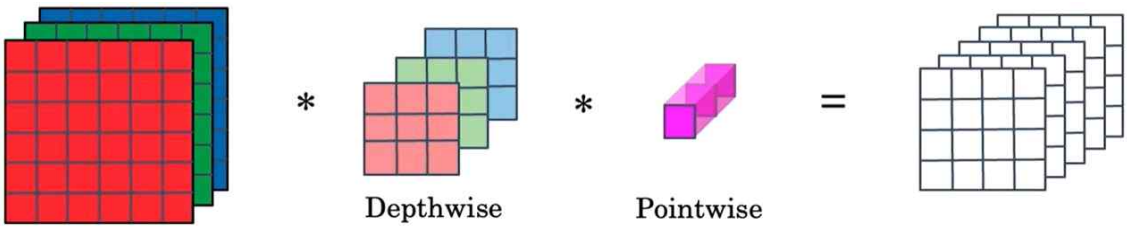
Normal convolution



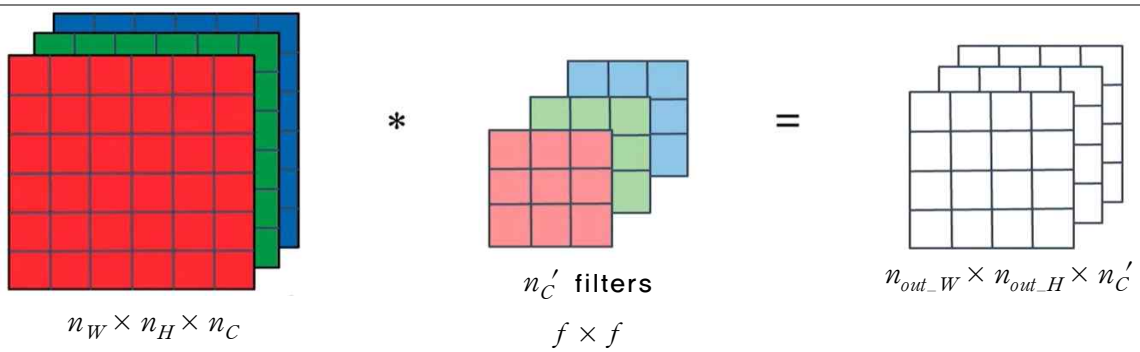
computational cost = # filter params x # filter positions X # filters

$= f \times f \times n_C$ x $n_{out\_W} \times n_{out\_H}$ x n_C'

Depthwise Separable convolution



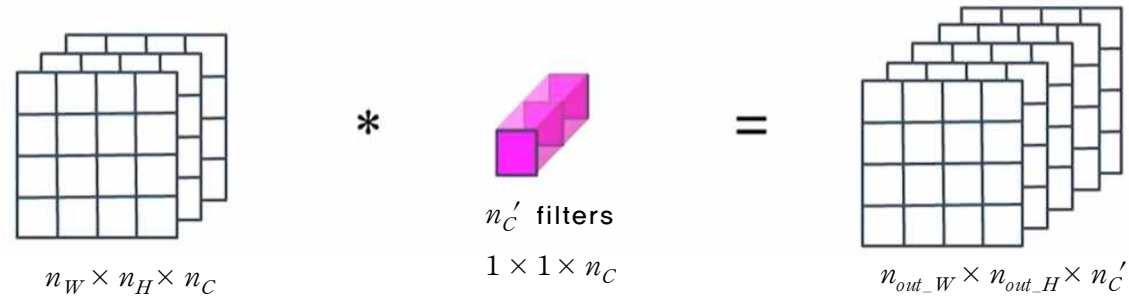
Depthwise convolution



computational cost = # filter params x # filter positions X # filters

$= f \times f$ x $n_{out\_W} \times n_{out\_H}$ x n_C'

Pointwise convolution



computational cost = # filter params x # filter positions X # filters

$= 1 \times 1 \times n_C$ x $n_{out\_W} \times n_{out\_H}$ x n_C'

cost of normal convolution

$f \times f \times n_C$ x $n_{out\_W} \times n_{out\_H}$ x n_C'

cost of depthwise separable convolution

depthwise + pointwise

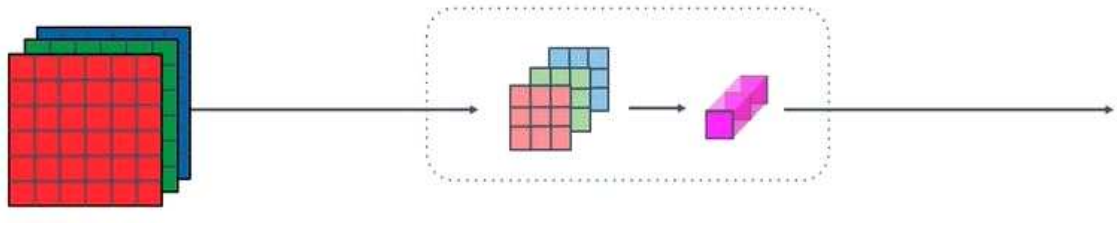
$= f \times f$ x $n_{out\_W} \times n_{out\_H}$ x n_C' + $1 \times 1 \times n_C$ x $n_{out\_W} \times n_{out\_H}$ x n_C'

cost of normal convolution과 cost of depthwise separable convolution의

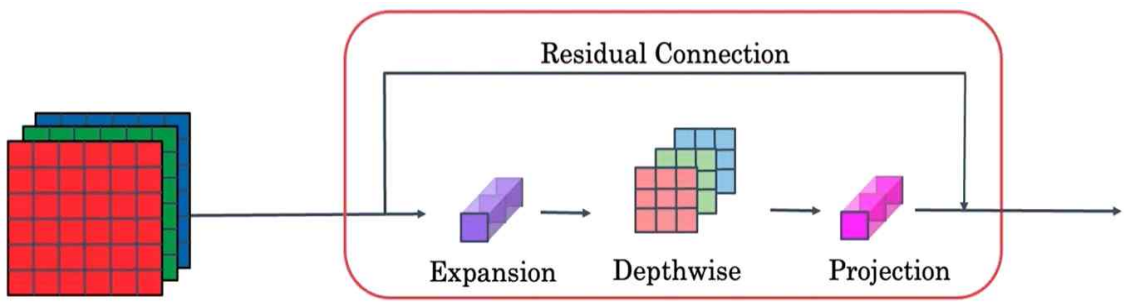
비율 계산 = $\frac{1}{n_C'} + \frac{1}{f^2}$

cost of depthwise separable convolution가 10배 정도 싸다

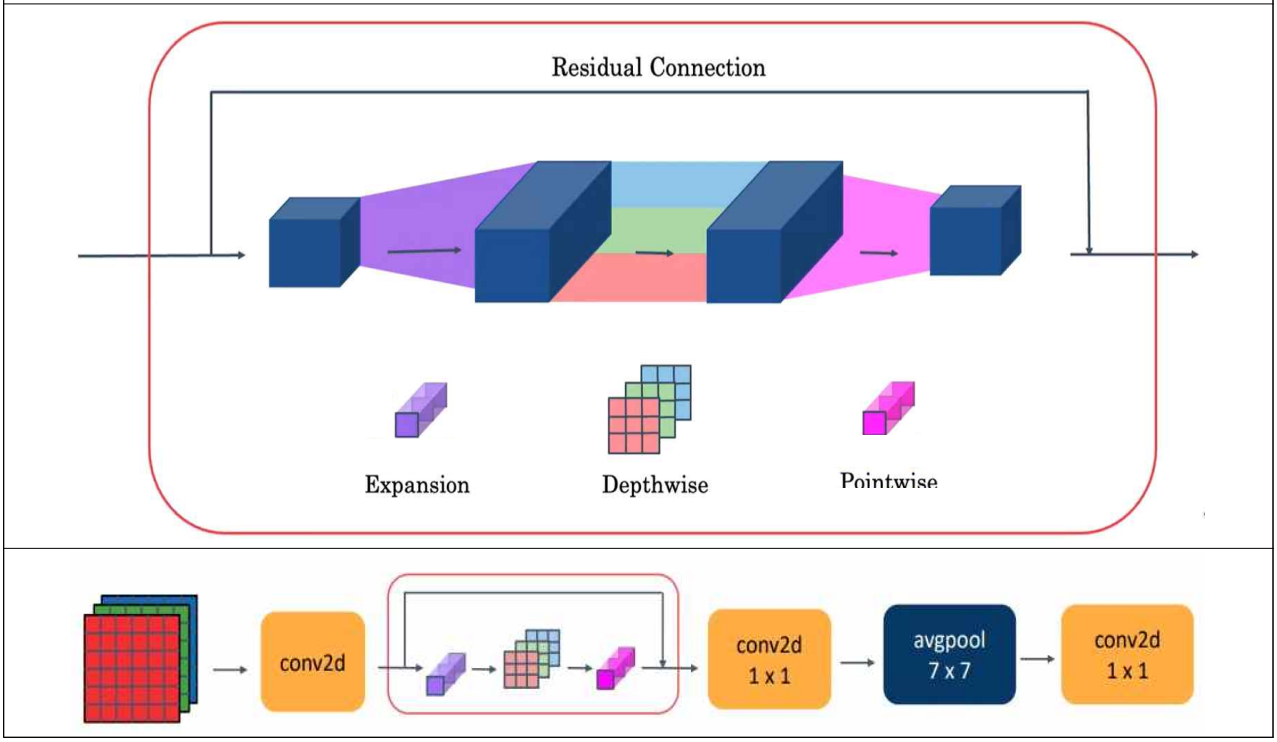
MobileNet v1



MobileNet v2



[Sandler et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks]

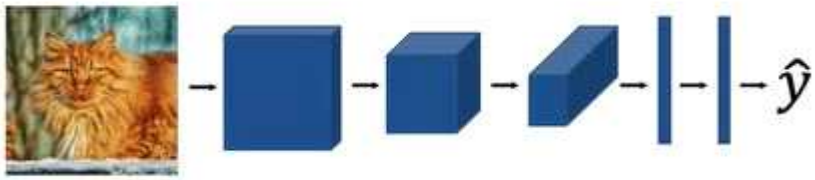


EfficientNet

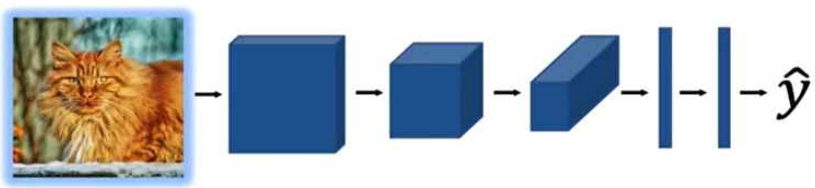
[Tan and Le, 2019, EffientNet:
Rethinking Model Scaling for Convolutional Neural Networks]

Baseline 기준으로 resolution과 depth, wide를 유동적으로 적용한다.

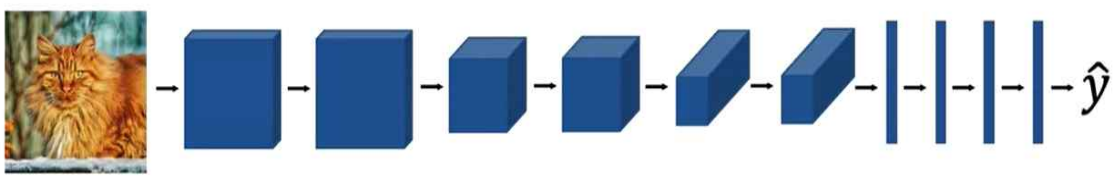
Baseline

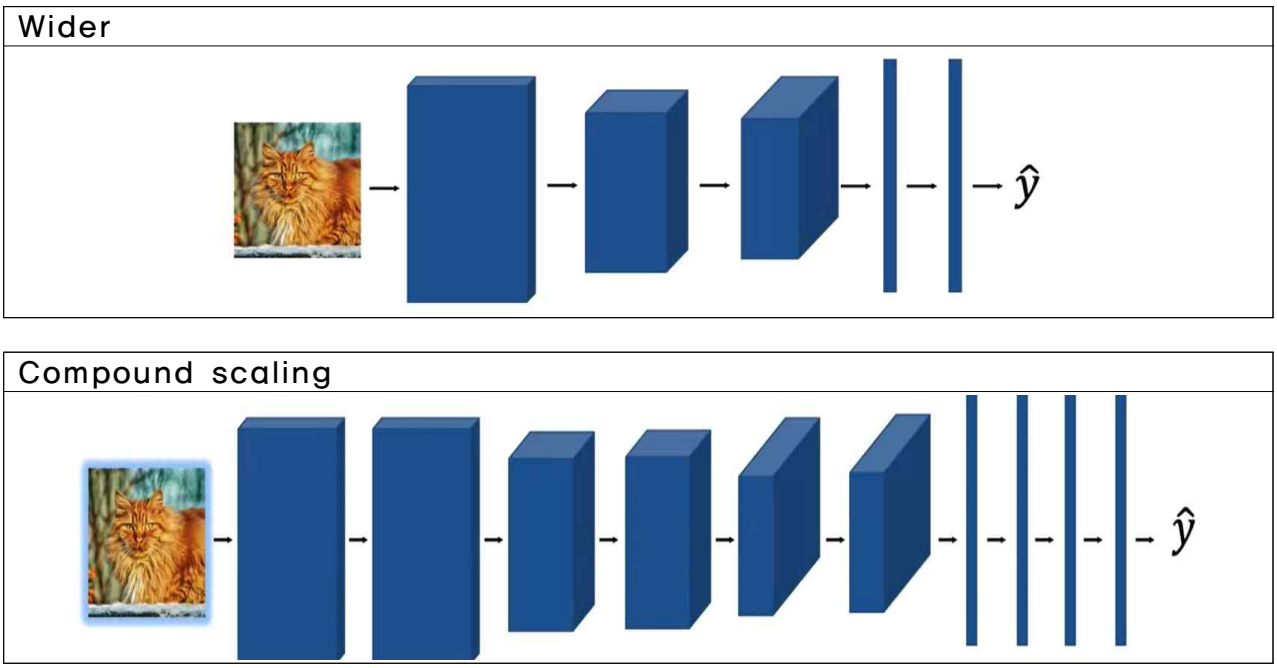


Higher Resolution

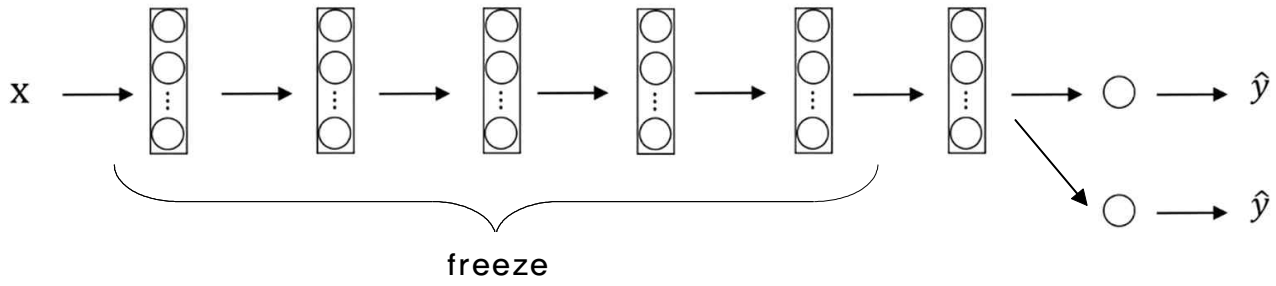


Deeper





Transfer learning



이미 학습되어있는 모델의 코드뿐만 아니라 가중치도 그대로 적용한다.

When: 새로 학습시킬 만큼 충분히 많은 데이터나 시간이 없다.

safety disk or the pre-compute method의 장점:

activation을 업데이트할 필요가 없다.

규칙: 새로 보려는 데이터와 레이블이 많을수록 더 적은 레이어를 고정(freeze)한다.

데이터가 충분하지 않으면 Softmax와 관련된 파라미터만 조정하고

나머지 layers의 파라미터는 고정하는 것을 권장한다.

데이터가 충분히 많으면 전체 layers를 다시 학습시키는 것도 좋다.

Data augmentation

Original image

1. Mirroring

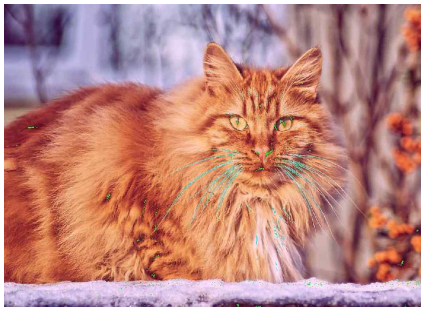
2. Random Cropping



3. Rotation
4. Shearing
5. Local warping

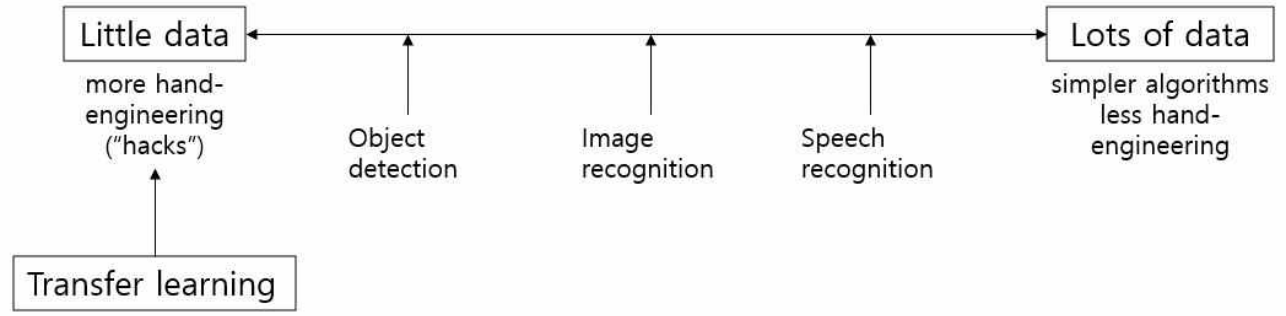
Color shifting

색상 왜곡이나 색상 이동을 하면 학습 알고리즘이 이미지 색상 변화에 강해진다.



Practical advice for using ConvNet

Data vs. Hand-engineering



Two sources of knowledge

1. Labeled data
2. Hand engineered features/network architecture/other components

Tips for doing well on benchmarks/winning competitions

BUT 시간이 너무 오래 걸린다.

1. Ensembling

   Train several networks independently and average their ouputs.

2. Multi-crop at test time

   Run classifier on multiple versions of test images and average results.

Use open source code

1. Use architectures of networks published in the literature.
2. Use open source implementations if possible.
3. Use pretrained models and fine-tune on your dataset.