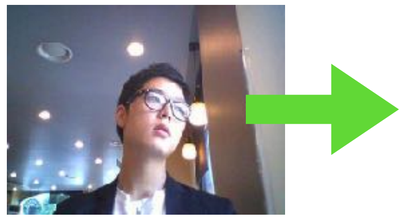
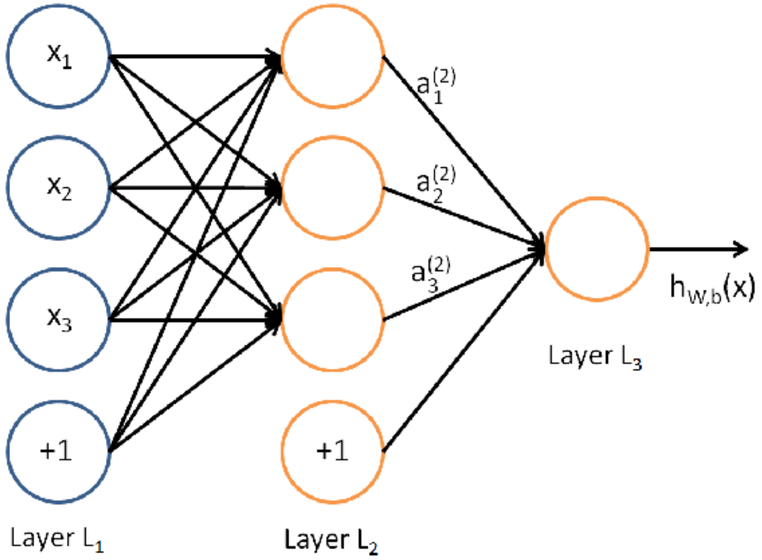
Practical Usage of Convolutional Layer

Overview

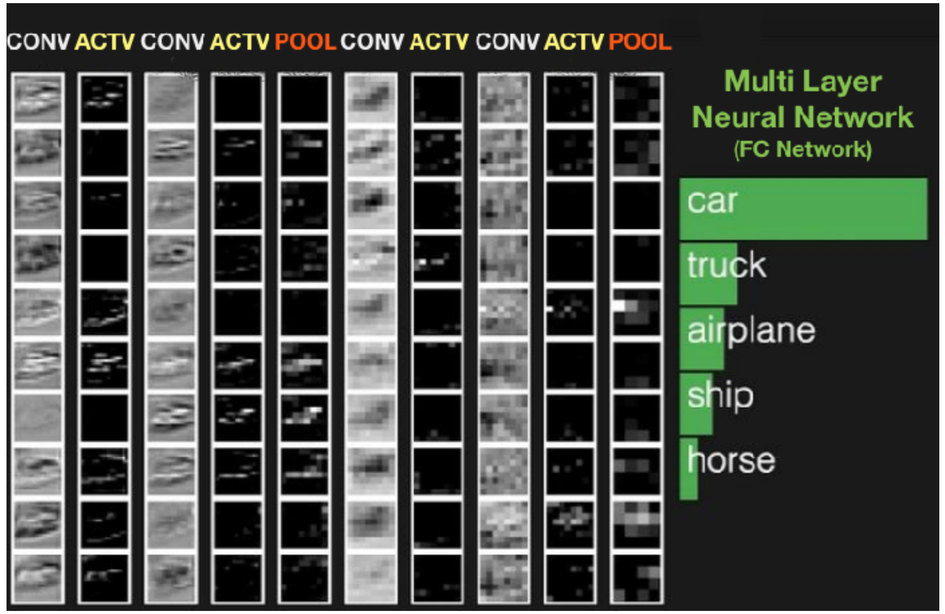
Convolutional Layer

이미지를 convolve하는 Hidden Layer를 만든다.
이렇게 하면 FC Layer보다 더 적은 weight를 갖고도 효율적으로 연산할 수 있다.





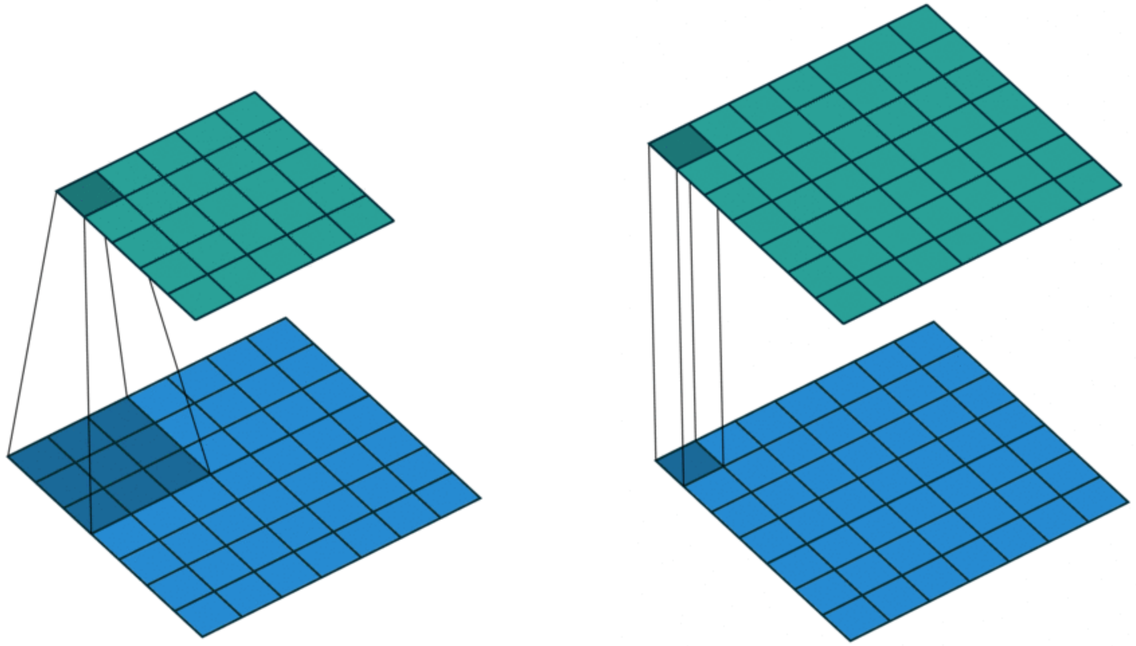
Before
Multi-layer Neural Network
(Fully-connected Neural Network)



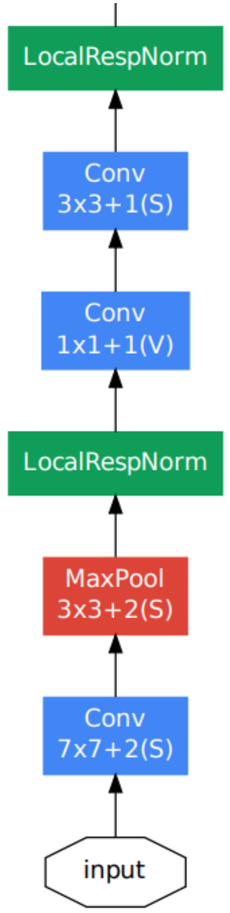
After
Convolutional Neural Network

Convolutional Layer

일단 취지는 좋았다. 하지만 우리에게는 아직 다양한 고민거리가 남아있다.
ex: 가로x세로는 얼마가 가장 적합한 것인가?, 얼마나 깊게 쌓아야 하는가?



쌓아야하는 레이어의 크기(가로x세로)에 따라 그 결과가 달려지며



쌓아야하는 레이어의 순서에 따라서도 결과가 달라진다.

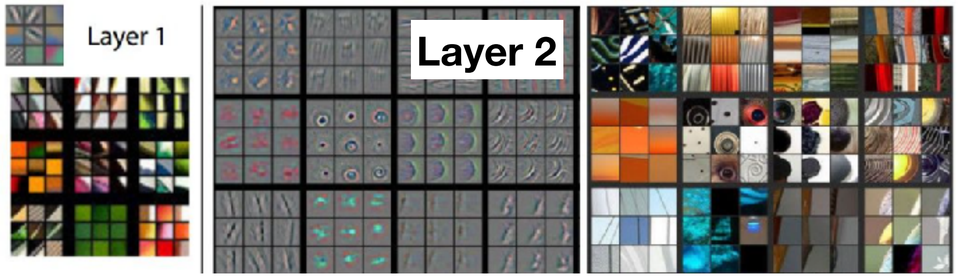
Best Practices for Convolutional Layers

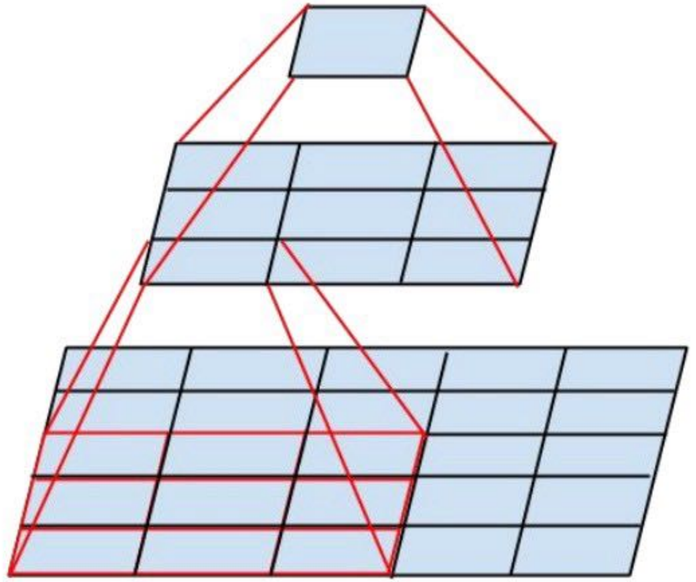
Q1) Convolutional Layer의 가로x세로는 몇 개가 좋은가?

정답: 3x3이 가장 좋다.

- 5x5 1개랑 3x3 2개는 동일한 영역을 커버한다.
- 하지만 깊이(Depth, 이하 D)가 동일할 경우,
  - 5x5은 D x (5 x 5 x D) = **$25D^2$**개의 weight가,
  - 3x3 3개는 2 x (D x (3 x 3 x D)) = **$18D^2$**개의 weight가 필요하다.
- 그러므로 3x3이 더 적은 weight와 연산량으로 동일한 영역을 커버할 수 있다.
- 또한 레이어는 쌓으면 쌓을수록 더 고차원적인 연산을 하기 때문에, 동일한 weight로 레이어를 더 많이 쌓는 3x3이 더 유리하다.

keras.layers.Conv2D(filters=64, **kernel_size=(3, 3)**)





작은 Convolutional layer를 여러개 쌓으면,
결국 큰 이미지에 대한 연관관계를 계산하는 것과 마찬가지라고 볼 수 있다.

Very Deep Convolutional Networks for Large-Scale Image Recognition
Simonyan and Zisserman, 2014. https://goo.gl/mbKWhq

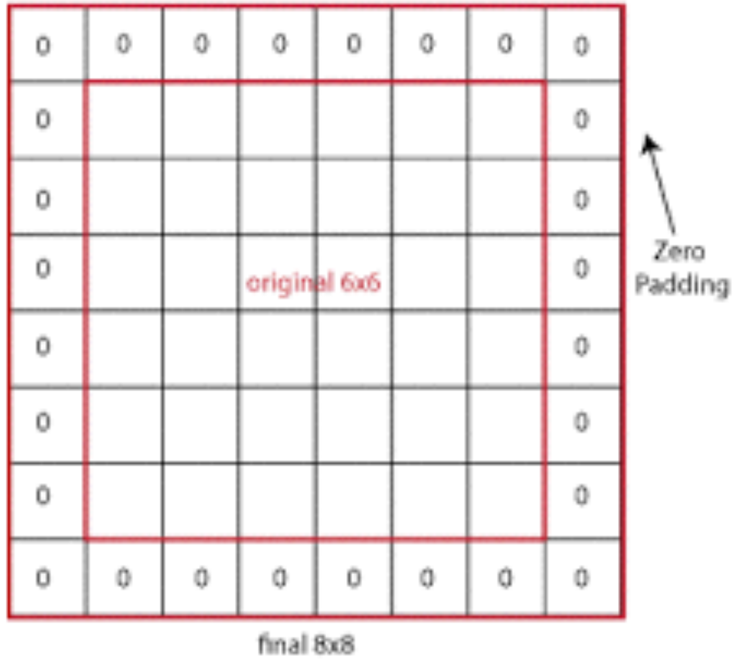
Best Practices for Convolutional Layers

Q2) convolve 연산을 할 수록 Output의 사이즈는 Input보다 작아진다. 이를 어떻게 해결하는가?

정답: zero-padding을 이용한다.

- Convolutional Layer의 크기에 맞게 zero-padding을 넣어주면 Output의 사이즈가 줄어드는 것을 방지할 수 있다.
  - ex: 3x3 convolutional layer라면 zero-padding을 1만 넣어주면 된다.
- 대부분의 딥러닝 프레임워크에서 zero-padding을 지원하며, **SAME**과 **VALID** 옵션 중에 하나를 골라서 사용하면 된다.
  - SAME: zero-padding을 사용. Input과 Output이 동일해지도록 알아서 zero-padding을 넣어줌
  - VALID: zero-padding을 넣지 않음. convolve 연산을 할수록 output 사이즈가 작아짐.
- 또한 zero-padding의 장점은 외곽에 있는 Input에도 상대적으로 더 많은 연산을 할 수 있도록 유도한다는 장점이 있다.

keras.layers.Conv2D(filters=64, kernel_size=(3, 3), **padding='same'**)



Input의 주변에 0으로 감싸주는 것 만으로 Output 사이즈를 동일하게 만들 수 있으며, 또한 외각에 대한 Feature Extraction도 강화된다.

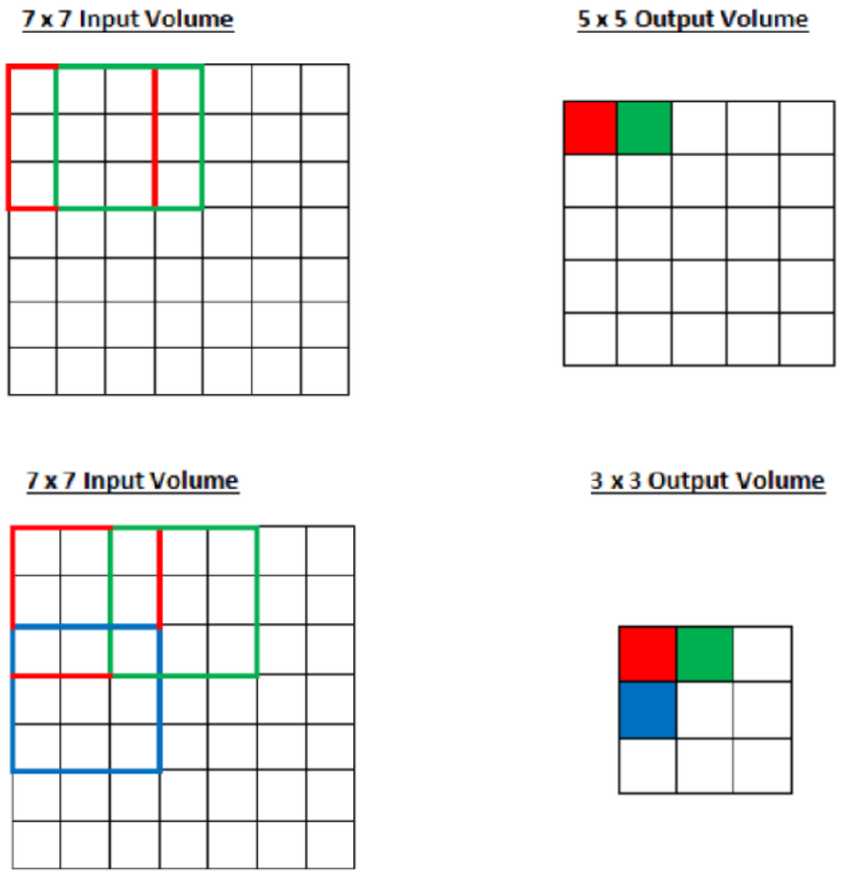
Best Practices for Convolutional Layers

Q3) sliding-windows를 할 때 stride는 얼마나 가장 적합한가?

정답: 1이 가장 기본적인 옵션이다.

- stride를 작게 정하면 그만큼 input을 촘촘하게 convolve하여 feature map을 뽑아낸다. 반면 stride가 크면 그렇지 못하다.

keras.layers.Conv2D(filters=64, kernel_size=(3, 3),
padding='same', **strides=(1, 1)**)



같은 사이즈의 input과 convolutional layer라고 하더라도 stride를 다르게 주면 output size가 달라진다

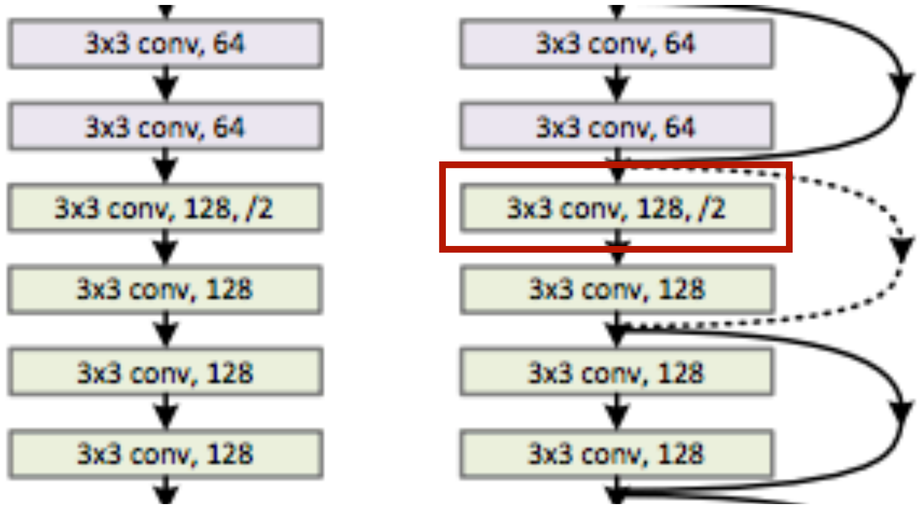
Best Practices for Convolutional Layers

Q3) sliding-windows를 할 때 stride는 얼마나 가장 적합한가?

예외적으로 max-pooling을 대체하고 싶으면 stride를 늘린 Convolutional Layer를 사용한다.

- ResNet에서는 max-pooling 대신 stride를 2를 준 3x3 Convolutional Layer를 사용한다.
- DCGANs(GANs의 업그레이드 버전)에서도 max-pooling 대신 stride를 크게 준 Convolutional Layer를 권장하고 있다.

keras.layers.Conv2D(filters=64, kernel_size=(3, 3),
padding='same', **strides=(2, 2)**)



ResNet에서는 max-pooling 대신
stride를 2로 준 Convolutional Layer를 사용하고 있다.

Deep Residual Learning for Image Recognition
He et al, 2015. https://goo.gl/ZCZVgB

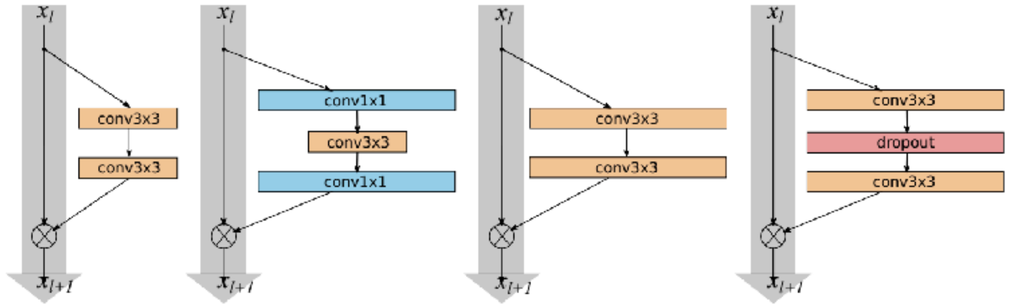
Best Practices for Convolutional Layers

Q4) 레이어 하나의 filters를 늘리는게 좋은가, filters를 줄이고 레이어를 더 많이 쌓는게 좋은가?

정답: filters를 늘리고 레이어를 적게 쌓는게 좋다.

- filter를 크게 준 50레이어의 Wide ResNet이 filter를 작게 준 152레이어의 ResNet보다 더 좋은 성능을 보인다

keras.layers.Conv2D(**filters=64**, kernel_size=(3, 3),
padding='same', strides=(1, 1))



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)

Wide Residual Networks
Zagoruyko and Komodakis, 2016. https://goo.gl/er6HRn

TL;DR

Convolutional Layer는

- **가로x세로 3x3**
- **stride는 1 칸 (max-pooling을 대체하고 싶으면 이를 늘린다)**
- **zero-padding은 1 칸**

의 레이어를, 깊이(depth)보다는 너비(width) 위주로 쌓는 게 좋다.

keras.layers.Conv2D(**filters=?, kernel_size=(3, 3)**,
padding='same', **strides=(1, 1)**)

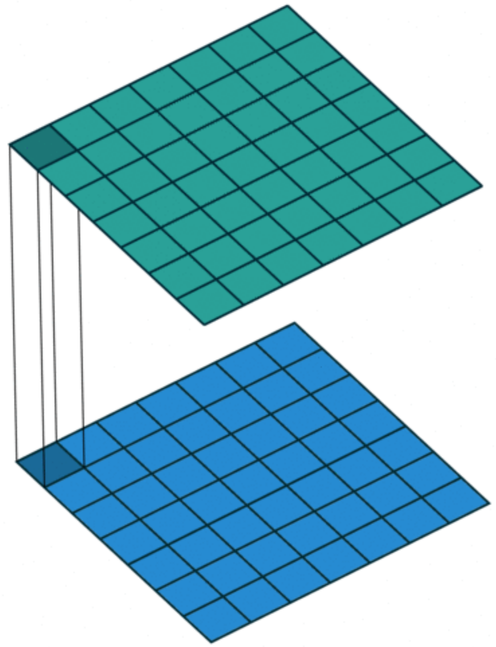
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

1x1 convolutional layer

1x1 convolutional layer

이게 왜 필요한가?

얼핏 아무런 의미가 없어 보인다.
하지만 1x1 convolutional layer는 굉장히 혁신적인 개념이며,
다양한 용도로 CNN에 활용할 수 있다.



픽셀의 연관관계를 계산하지 않고, 한 픽셀씩 연산한다.

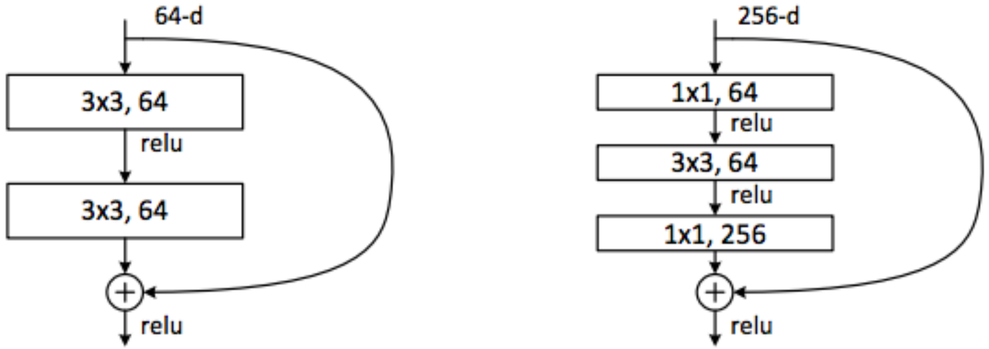
The usage of 1x1 convolutional layers

차원 축소(dimension reductionality)를 위해 사용할 수 있다

가령 우리가 동일한 width x height의 input이 있다면

- 좌측은 **depth가 64**밖에 안 되지만 (64 x (3 x 3 x 64)) x 2 = **73,728개**의 weight가 필요하다.
- 후자는 **depth가 256**이나 되지만 (256 x 1 x 1 x 64) + (64 x 3 x 3 x 64) + (64 x 1 x 1 x 256) = **69,632개**의 weight가 필요하다.

depth는 후자가 더 높지만(256 > 64) 그럼에도 weight와 그 연산은 1x1 convolutional layer를 도입한 후자가 더 효율적이다!



레이어에 앞뒤에 1x1 layer를 끼워넣으면
같은 3x3 layer라도 더 효율적으로 연산할 수 있다.
이를 병목(bottleneck) 디자인이라고 한다.

Deep Residual Learning for Image Recognition
He et al, 2015. https://goo.gl/ZCZVgB

The usage of 1x1 convolutional layers

Fully-connected layer를 대체할 수 있다

가령 우리의 Input이 25,000개라고 했을 때,

- **4,096개의 뉴런을 가진 hidden layer를 만들면**: weight는 25,000 x 4,096 = 102,400,000 개나 된다!
- **반면 4,096개의 depth를 가진 1x1 convolutional layer 변환하면**: weight는 1 x 1 x 4,096 = 4,096개로 획기적으로 줄어든다! (0.004% 감소)

그러므로 FC-layer는 1x1 convolutional layers로 대체 가능하며 이 방식이 더 메모리 효율적이다.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

“In Convolutional Nets, there is no such thing as "fully-connected layers”. There are only convolution layers with 1x1 convolution kernels and a full connection table.”
Yann LeCun. https://www.facebook.com/yann.lecun/posts/10152820758292143

The usage of 1x1 convolutional layers

딥러닝 모델에서는 대부분의 Fully-connected layer가 대부분의 weight를 차지한다.
그러므로 이를 1x1 convolutional layer로 교체하면 비약할만한 메모리 향상을 노릴 수 있다.

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

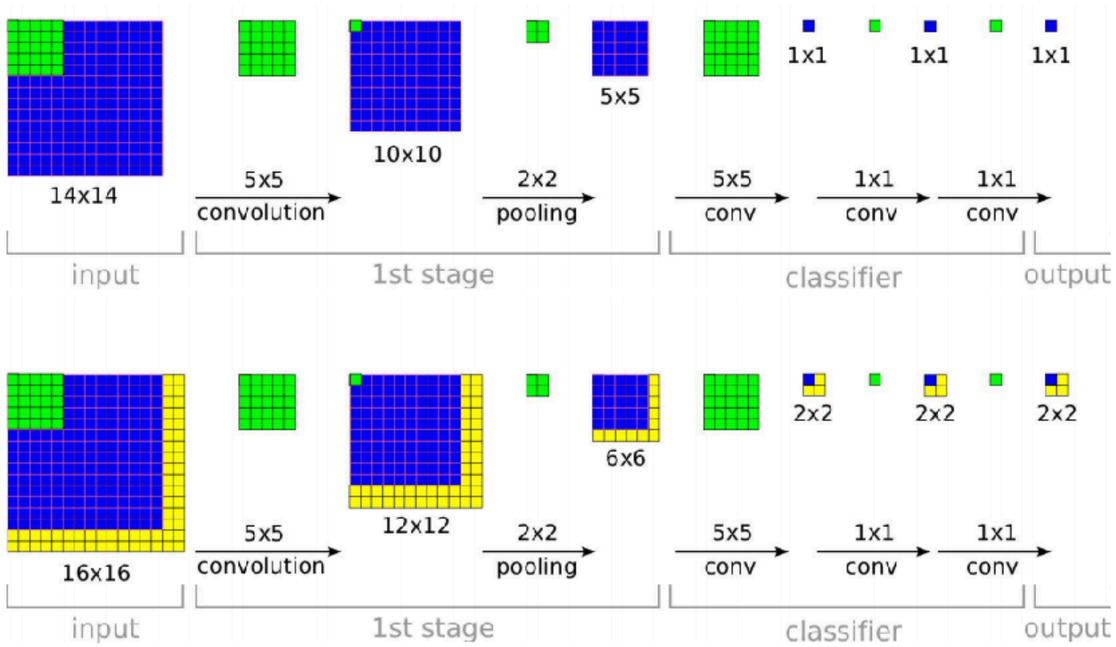
Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

The usage of 1x1 convolutional layers

Fully-connected layer를 없앤다면
input size에 구애받지 않고 자유롭게 이미지를 넣을 수 있다

- Fully-connected layer는 Input의 사이즈에 민감하기 때문에, 조금이라도 사이즈가 다르면 이미지가 들어가지 않는다.
- 반면 convolutional Layer는 사이즈에 영향을 받지 않기 때문에, 어떠한 크기의 이미지라도 집어넣을 수 있다.
- 이를 활용하면 test time에 train time에 집어넣은 Input 사이즈 보다 더 큰 값을 집어넣으면, 자연스럽게 연산이 이루어 지면서 **더 큰 사이즈의 이미지를 계산할 수 있다.**



OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks
Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & Lecun, Y https://goo.gl/KfSPbf

Others

일반적인 convolutional layer 이외에도 다양한 레이어가 존재한다.
하지만 자주 쓰이지 않는 개념이기 때문에, 필요할 때마다 그때그때 찾아서 쓰면 된다.

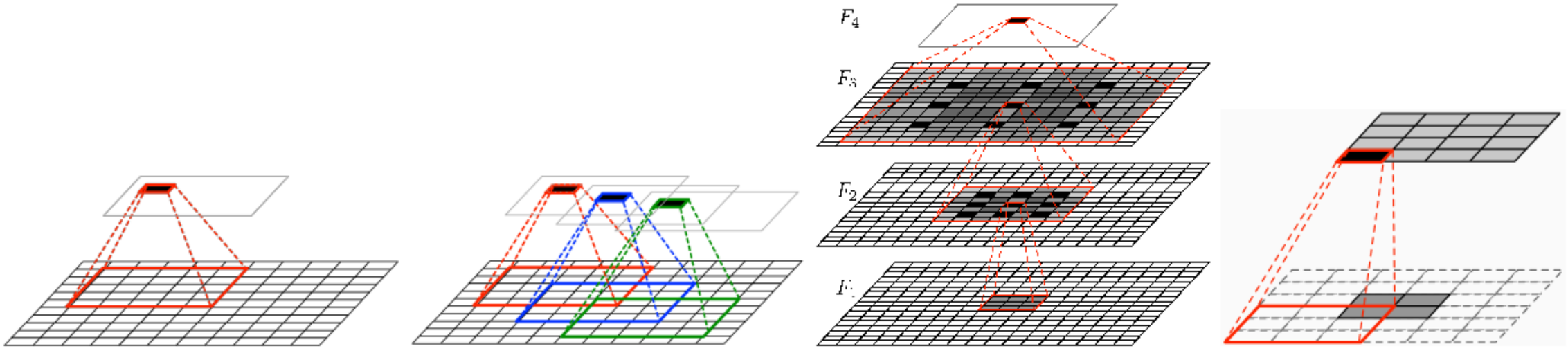


Figure 3: Illustration of (a) Convolution, (b) Tiled Convolution, (c) Dilated Convolution, and (d) Deconvolution

Recent Advances in Convolutional Neural Networks
https://arxiv.org/abs/1512.07108

Q & A