Machine Learning

CNN

김선녕(ksycafe@gmail.com)

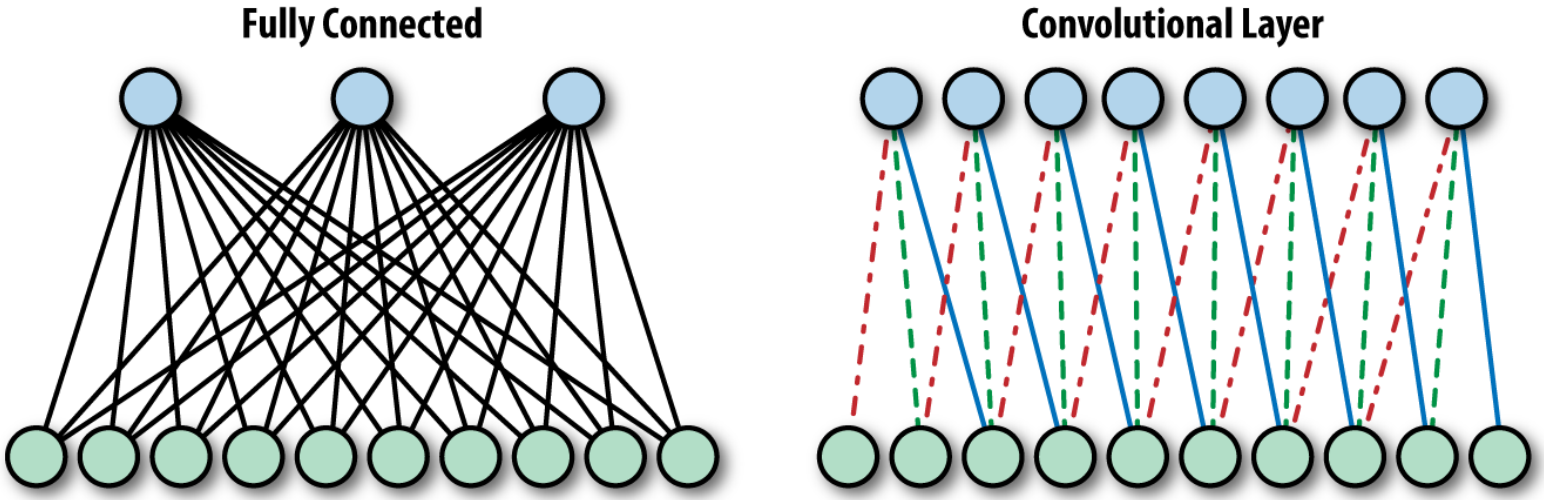
- CNN은 1989년 Yann LeCun이 발표한 논문 "Backpropagation applied to handwritten zip code recognition"에서 처음 소개
- 이미지 인식 분야에서 딥러닝을 활용한 기법은 대부분이 CNN이 기반
  - 희소연결 : 특성 맵에 있는 하나의 원소는 작은 픽셀 패치 하나에만 연결
  - 파라미터 공유 : 동일한 가중치가 입력 이미지의 모든 패치에 사용
- 일반적인 행렬 곱셈 대신 합성곱을 사용하는 신경망
- 이미지 인식과 음성인식등 다양한 곳에서 사용
  - 이미지 검색 서비스
  - 자율주행 자동차
  - 영상 자동 분류 시스템
  - 음성인식이나 자연어 처리 같은 다른 작업에서도 사용

image recognition

https://www.youtube.com/watch?v=w1n60lF40Ao&list=PL-jkQgH7sqGQw1h27uDE3kX2lb5ZKNSDB

Fully Connected vs Convolution Layer

- *Fully Connected Layer*
  - 각 뉴런은 이전 계층의 모든 뉴런에 연결
  - $CIFAR-10$ 데이터 셋은 $32 \times 32 \times 3 = 3072$개의 가중치가 필요
  - $200 \times 200 \times 3$의 크기의 이미지는 $200 \times 200 \times 3 = 120{,}000$개의 가중치가 필요.
  - 뉴런이 다층으로 존재하므로 가중치의 개수는 매우 크게 증가 : 비용증가
- *Convolutional Layer*
  - 각 뉴런은 이전 계층의 일정한 수의 뉴런에 연결
  - 연결의 가중치는 모든 뉴런이 공유



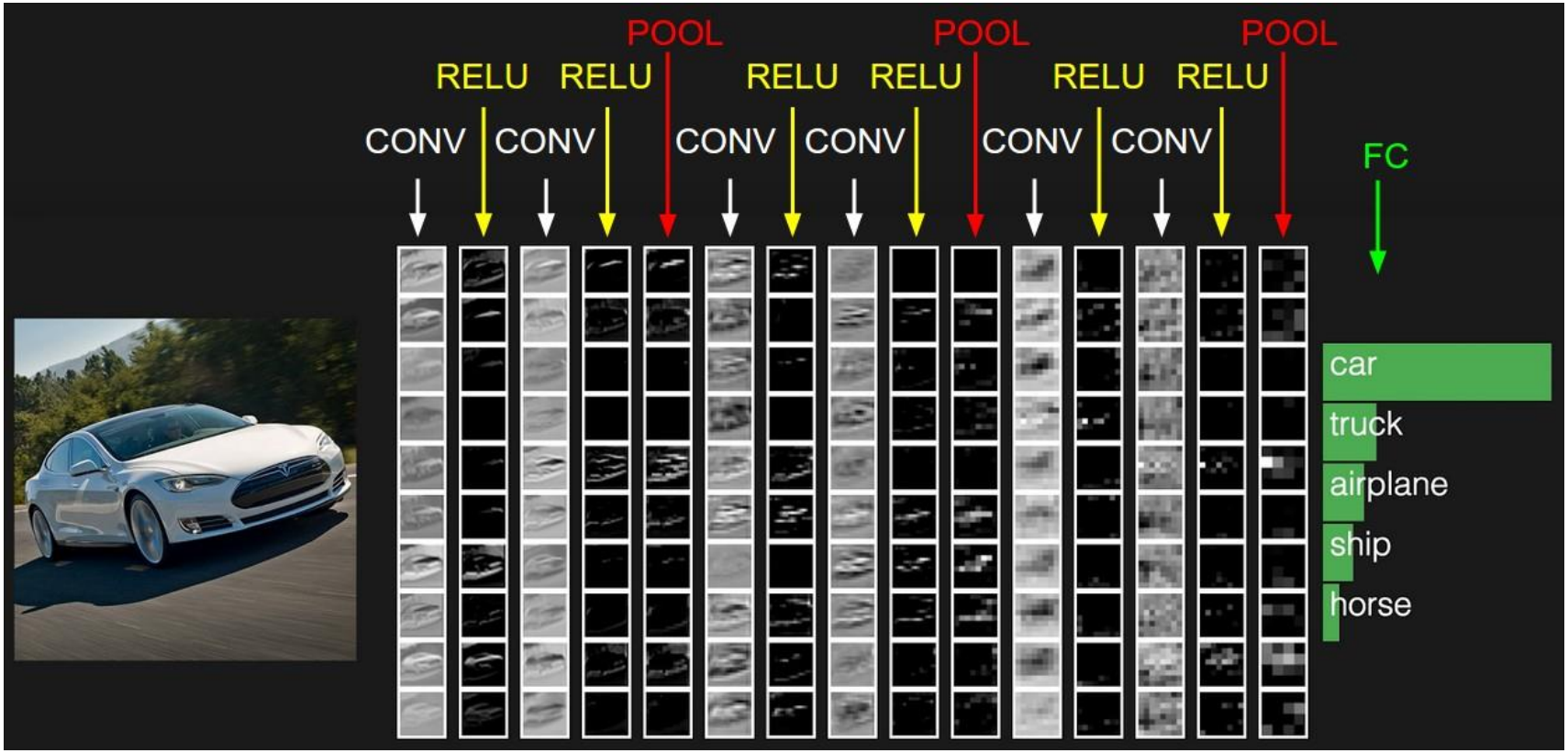
Learning Tensorflow, Oreilly

용어정리

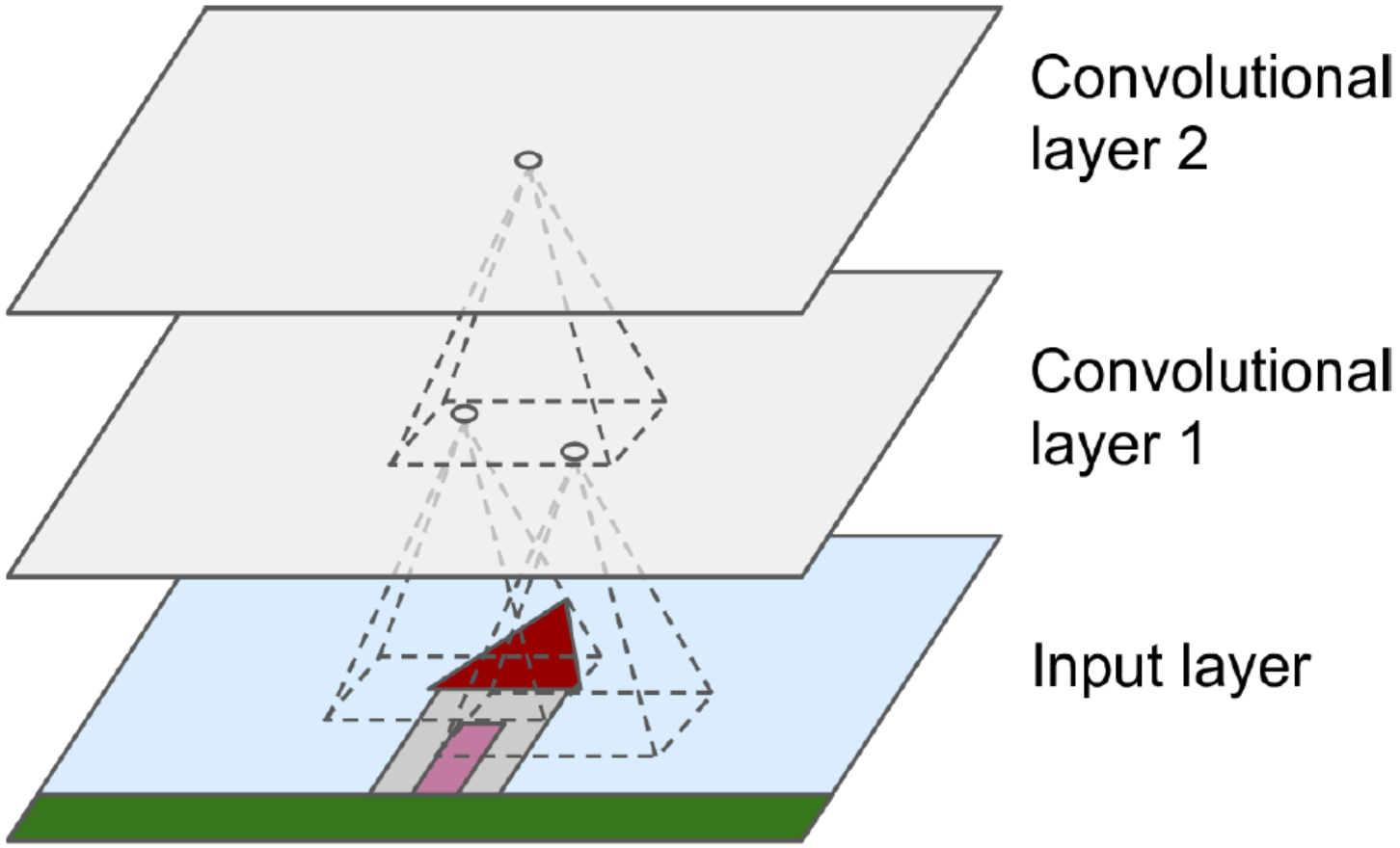
- $Channel(Color, depth)$
  - 이미지 픽셀 하나하나는 실수
  - MNIST 데이터 셋은 1개의 색상으로 2차원 데이터 $28 \times 28 \times 1$ (784개 데이터)
  - RGB는 3개의 실수로 표현한 3차원 데이터 $32 \times 32 \times 3$ (3072개 데이터)
- $Filter(Kernel)$
  - 특징 검출기
  - 이미지의 특징을 찾아내기 위한 공용 파라미터
  - 학습된 가중치인 W로 매개변수화
- $Feature\ map(Activation\ map)$
  - 각 계층의 입출력 데이터
- $Padding$
  - 입력데이터 주변을 특정 값으로 채워 늘리는 것
- $Stride$
- $Convolution\ Layer$
- $Pooling\ Layer$
- $Fully\ Connected\ Layer$

CNN Layer

- *Convolutional Layer* : CNN의 핵심 구성층
- *Pooling Layer* : 매개변수와 계산의 양을 줄임으로써 overfitting 방지
- *Fully Connected Layer*
- *ConvNet Demo*
  - https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

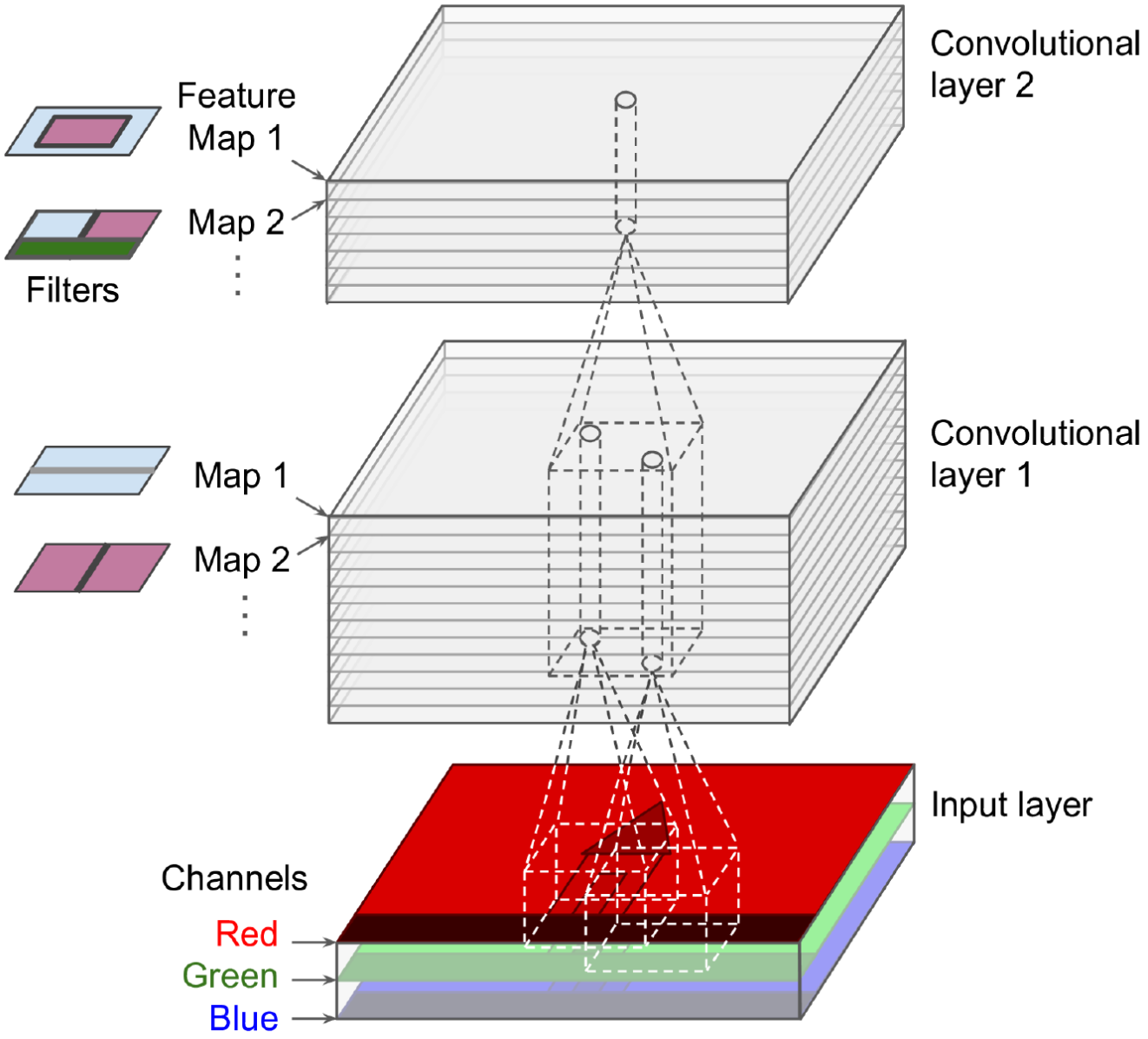


Convolutional Layer



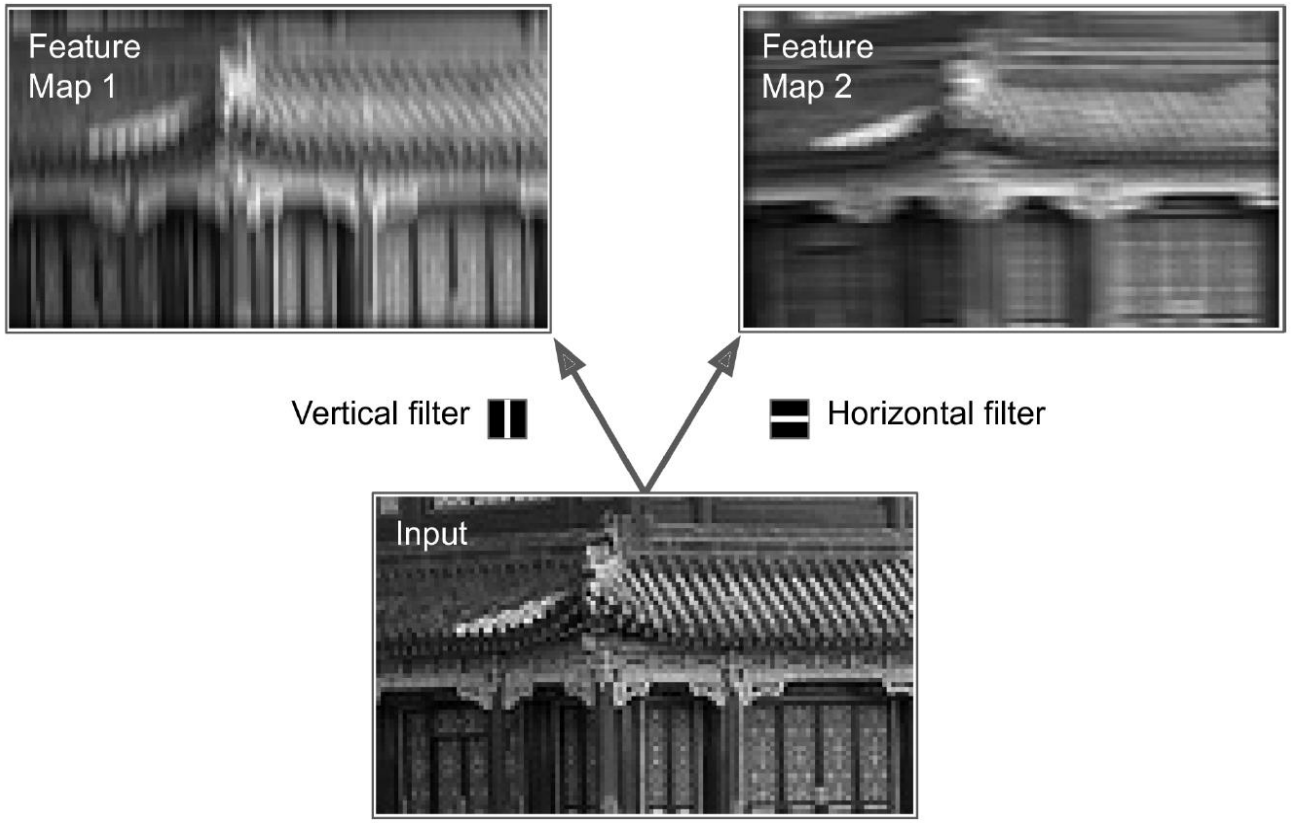
Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

Convolutional Layer



Filter(Kernel)

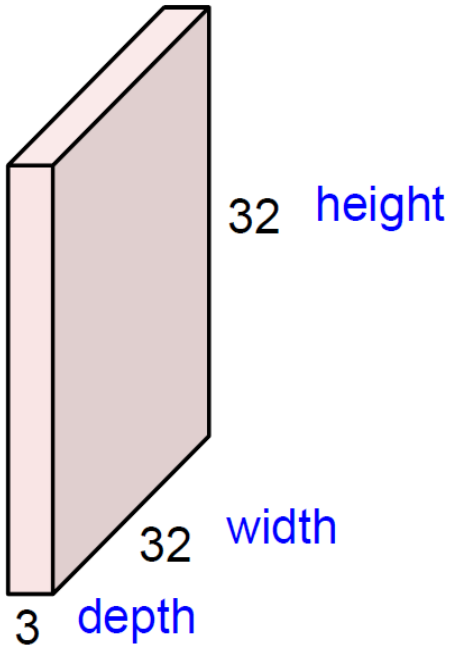
- 뉴런의 가중치는 수용 필드 크기의 작은 이미지로 표현 될 수 있다.
- 흰 수직선이 있는 검은 사각형(7 × 7 행렬) : 가운데 수직선 부분을 제외하고는 수용 필드에 있는 모든 것을 무시. 흰 수직선은 강조되고 나머지는 희미해 짐
- 흰 수평선이 있는 검은 사각형(7 × 7 행렬) : 가운데 수평선 부분을 제외하고는 수용 필드에 있는 모든 것을 무시. 흰 수평선은 강조되고 나머지는 희미해 짐



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

Convolution Layer

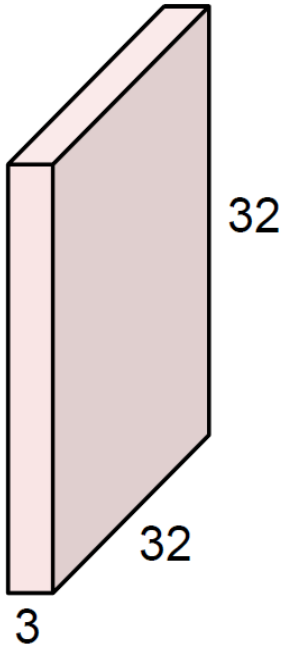
32x32x3 image -> preserve spatial structure



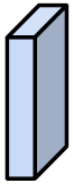
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

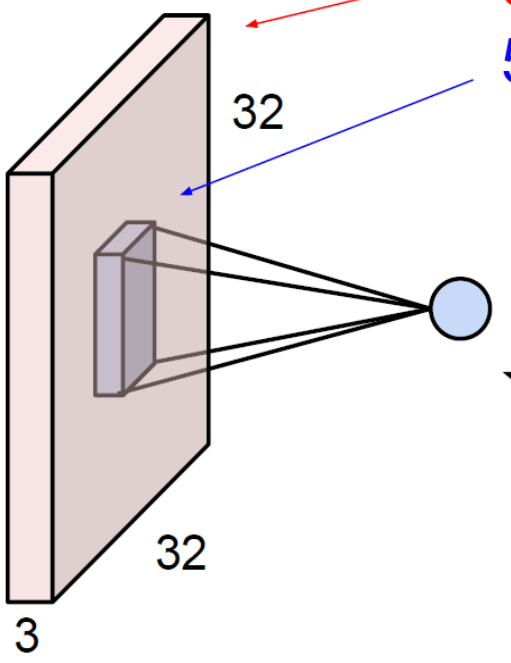


5x5x3 filter



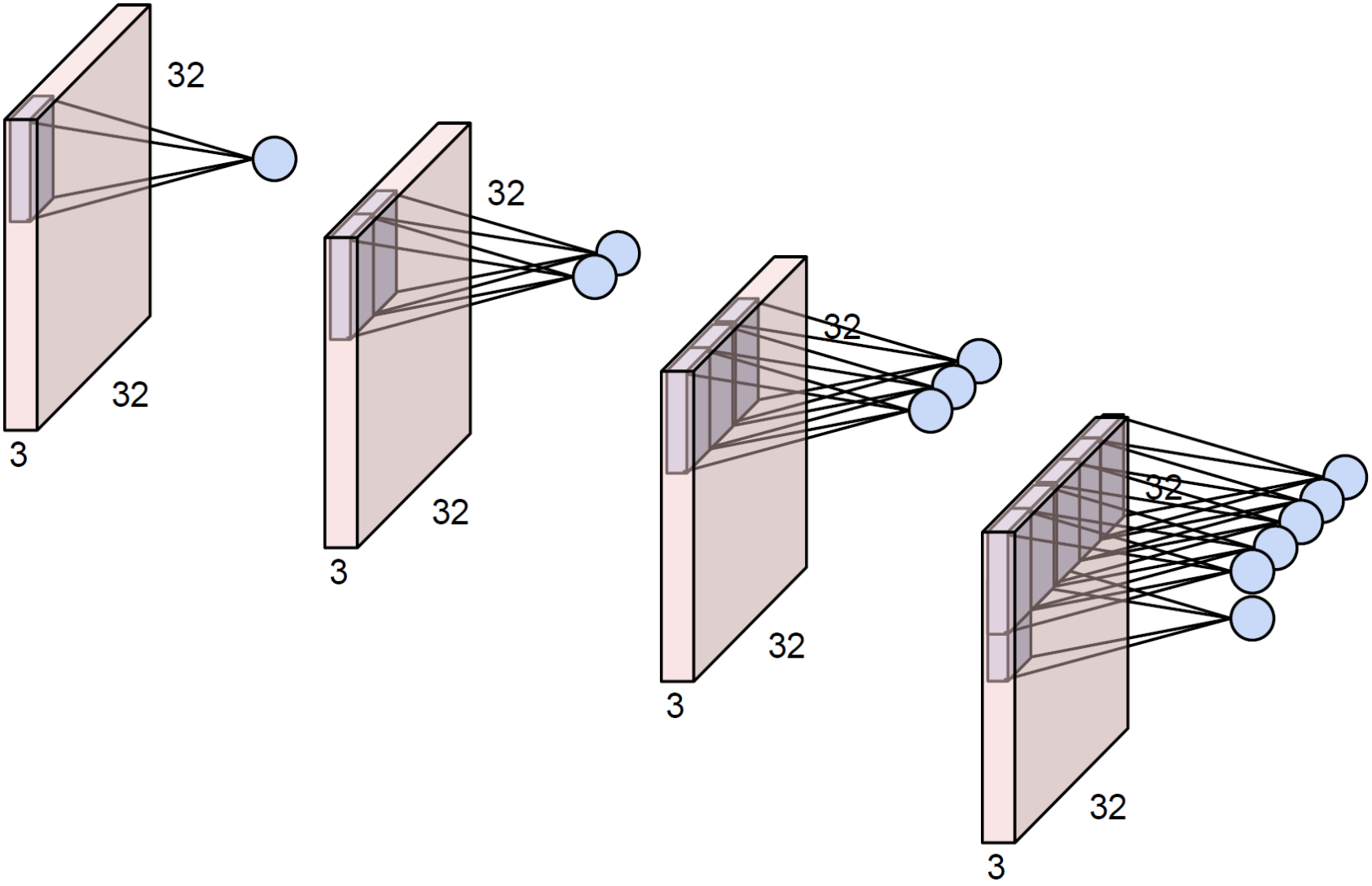
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

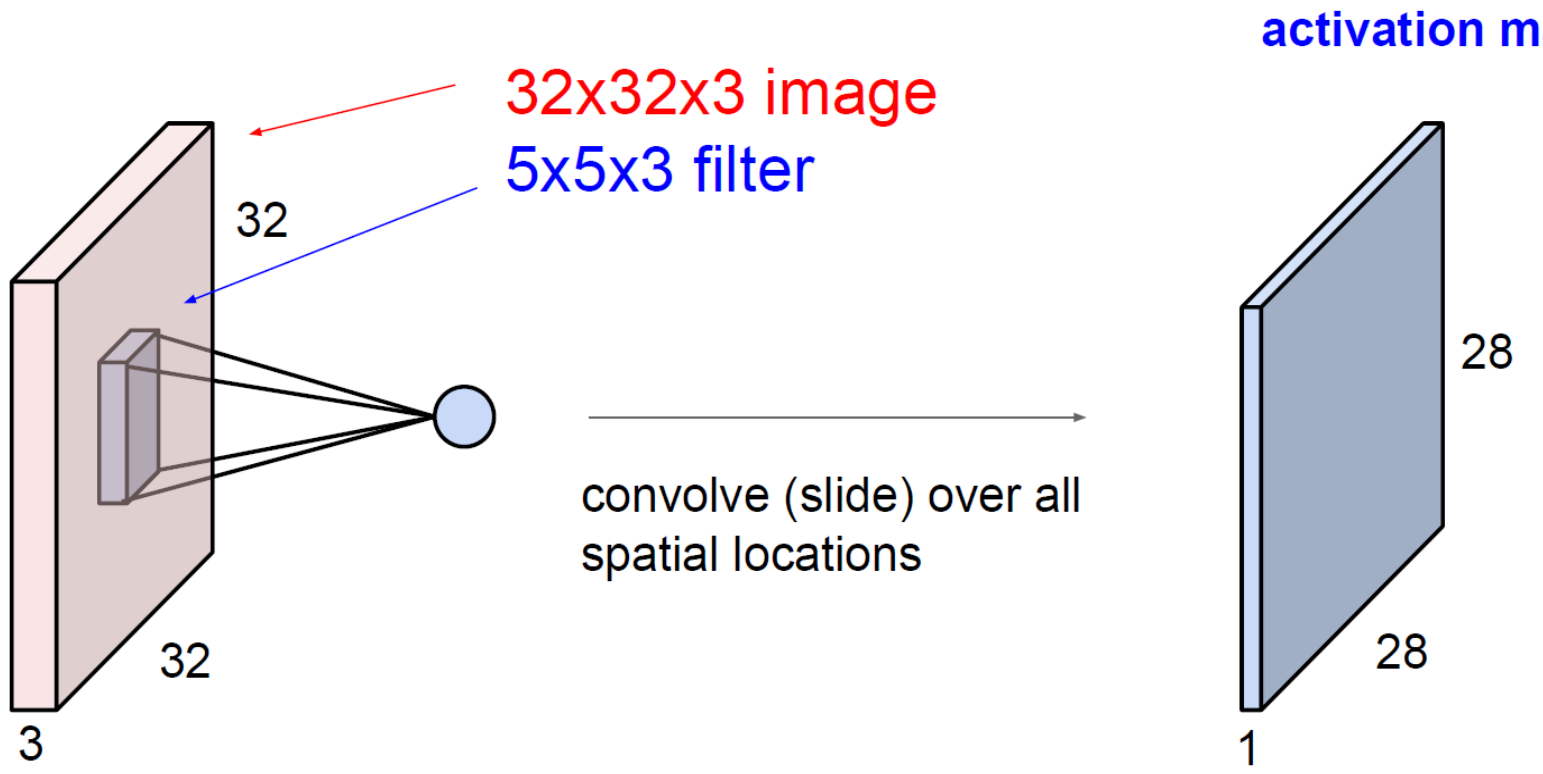


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

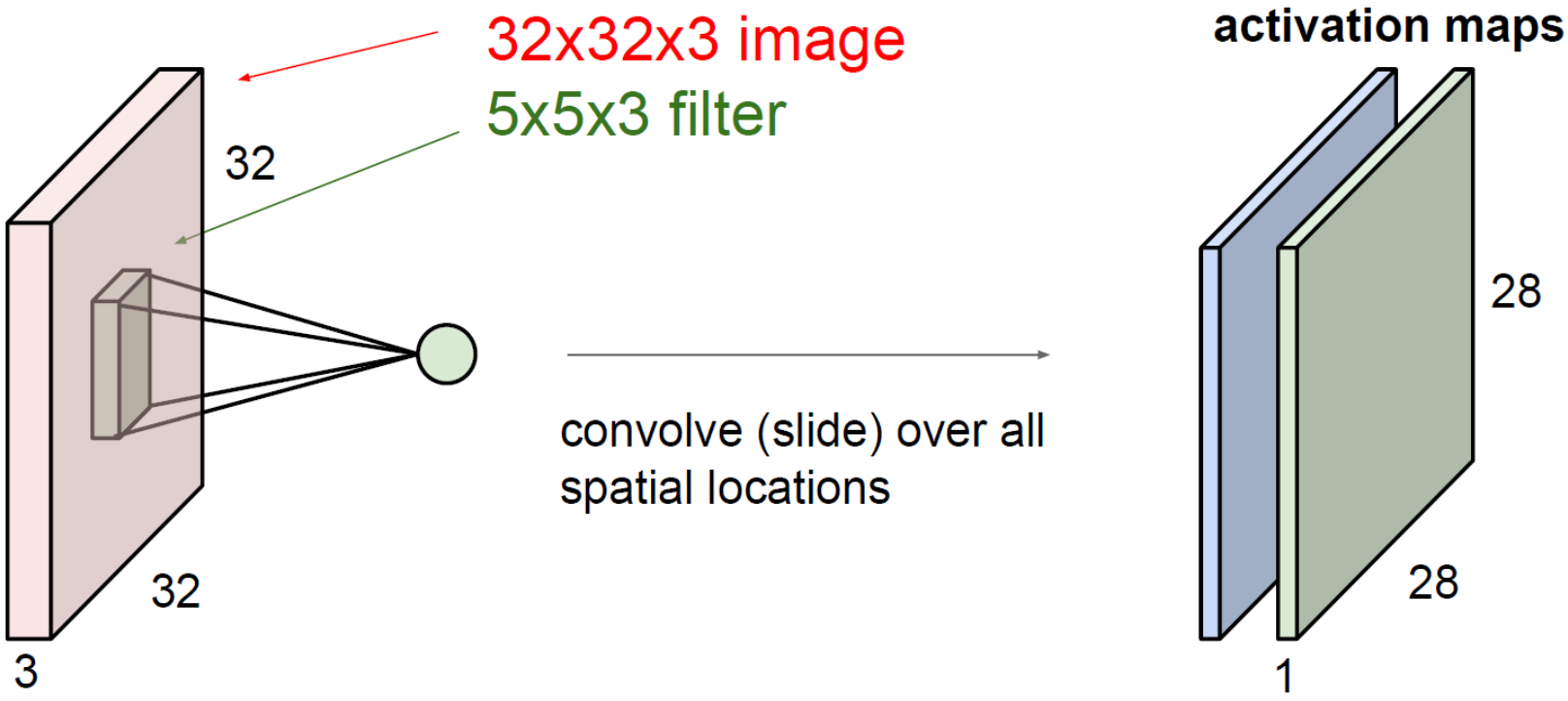

32
32
3
32
32
3
32
32
3
32
32
3

Convolution Layer

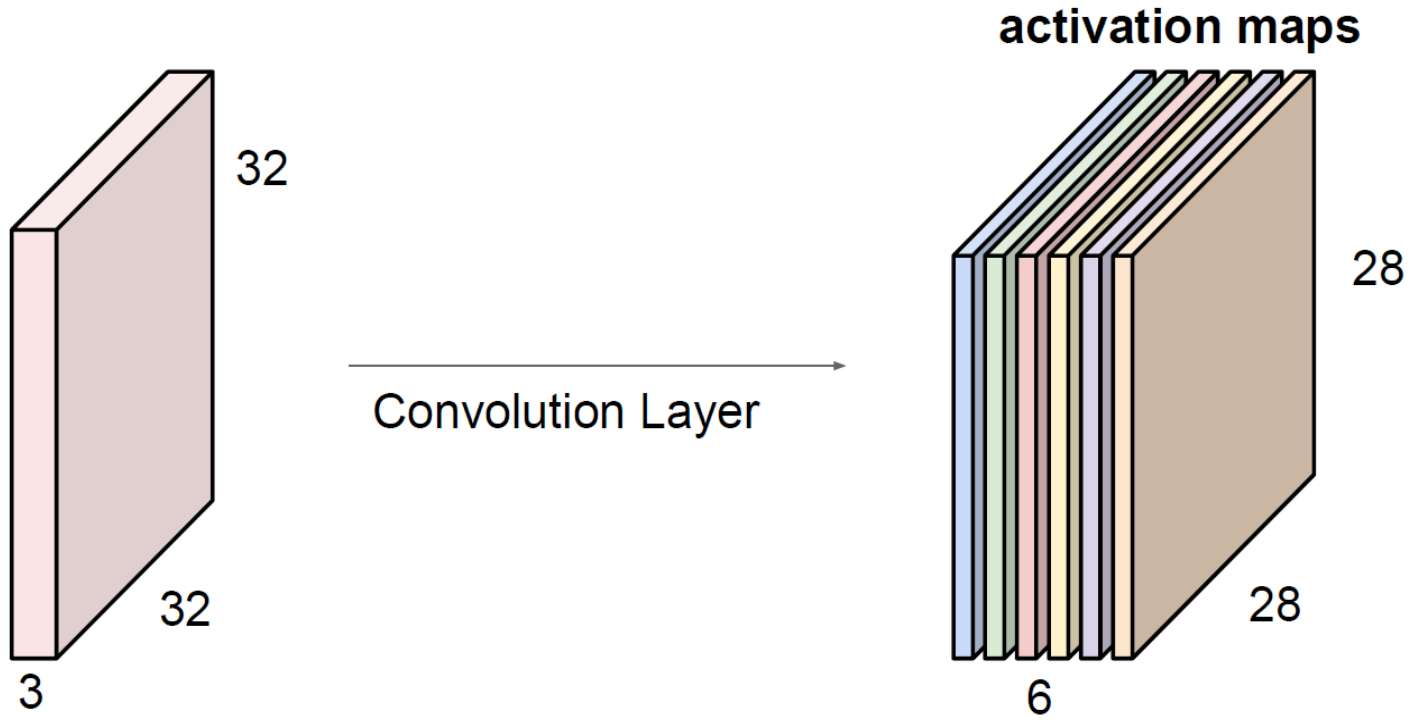


Convolution Layer

consider a second, green filter

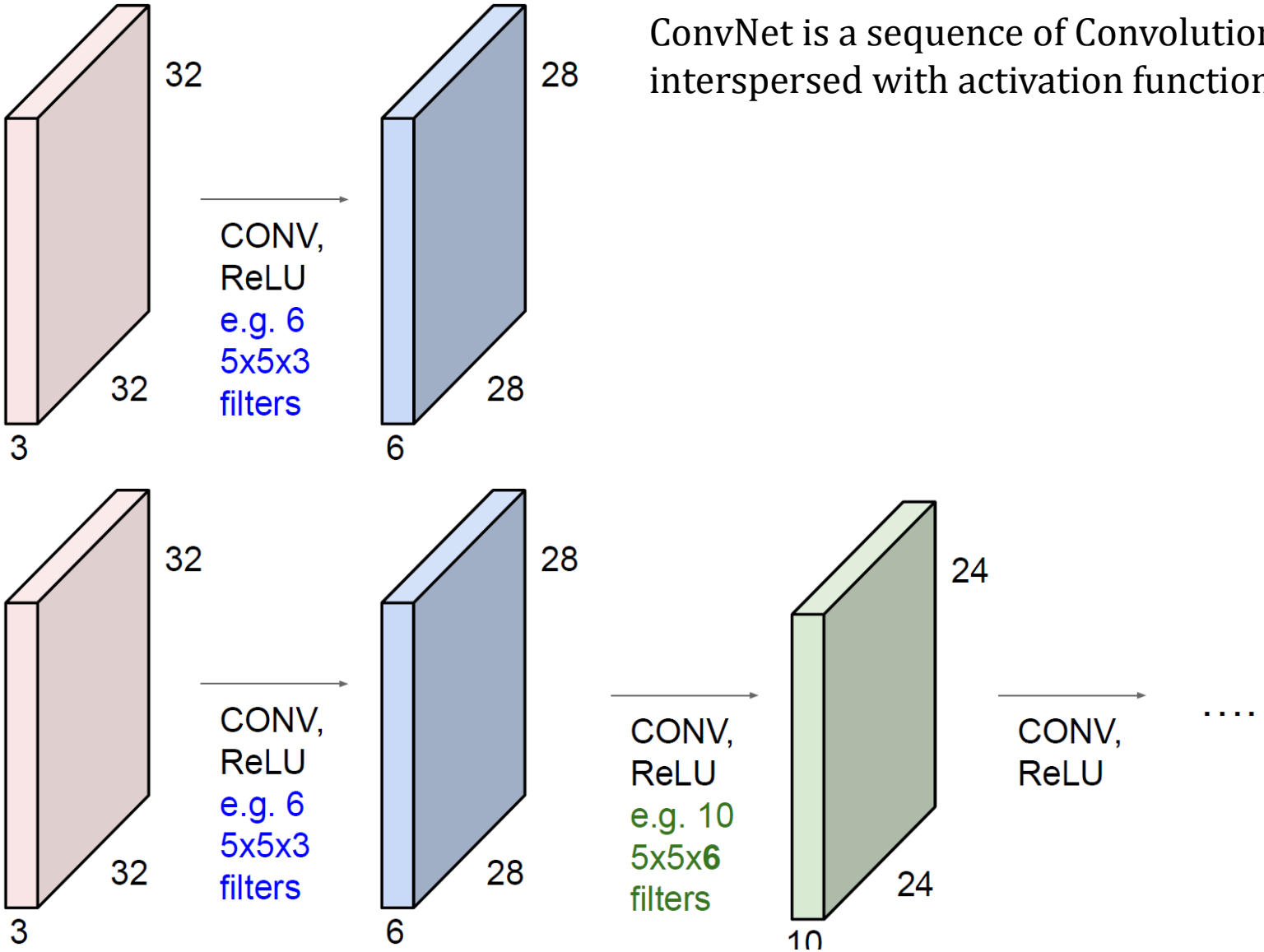


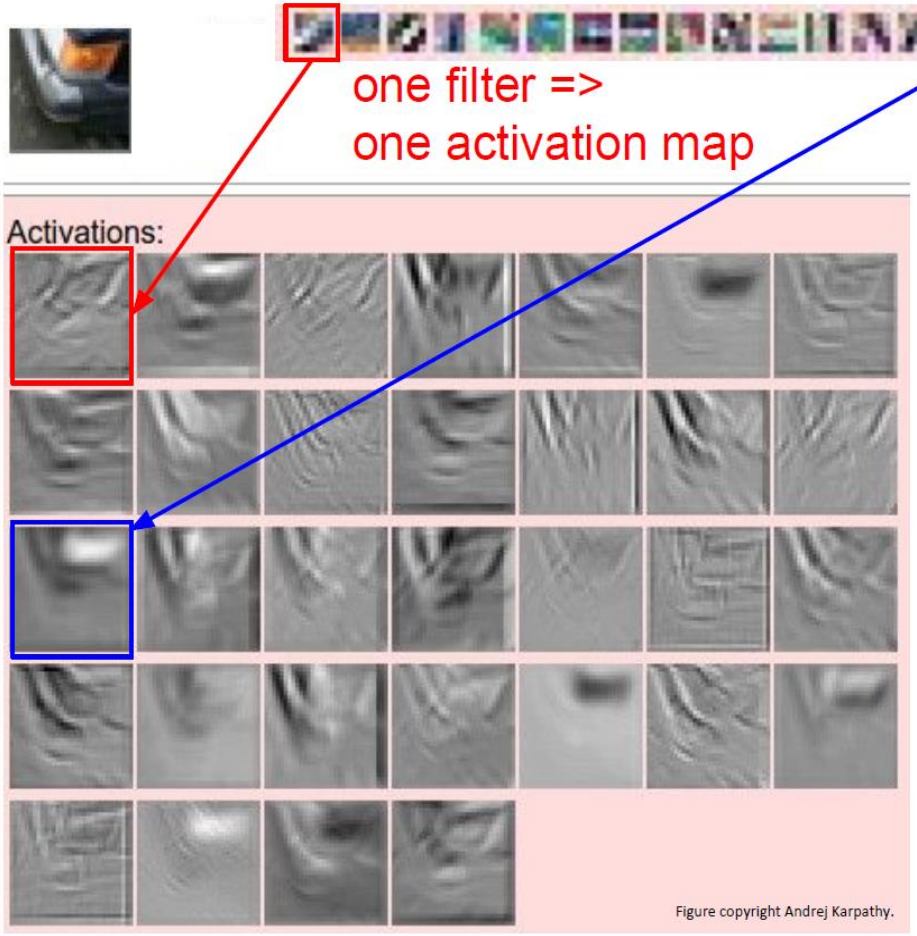
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

ConvNet is a sequence of Convolution Layers, interspersed with activation functions



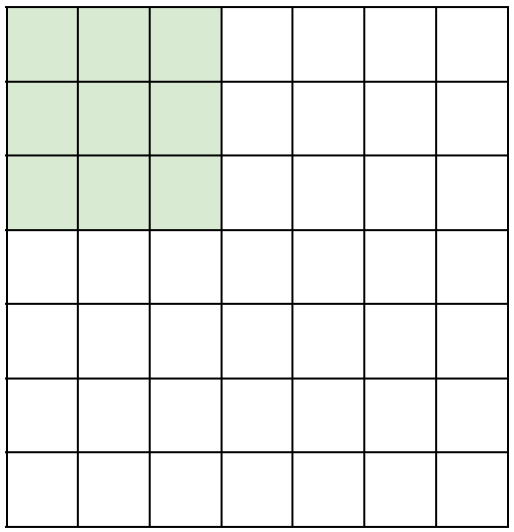


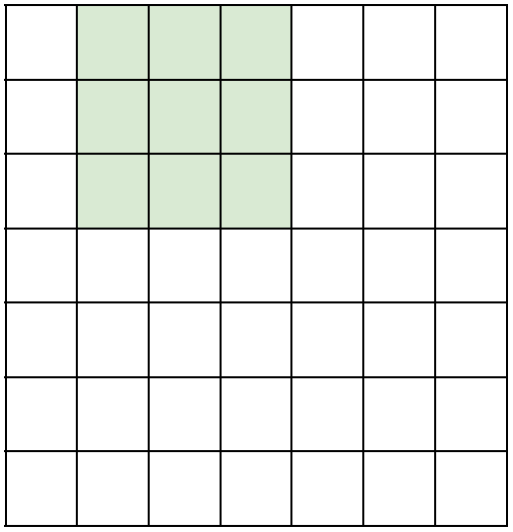
example 5x5 filters
(32 total)

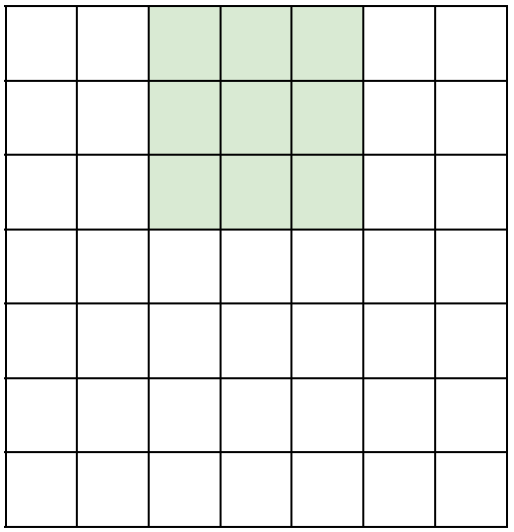
We call the layer convolutional because it is related to convolution of two signals:

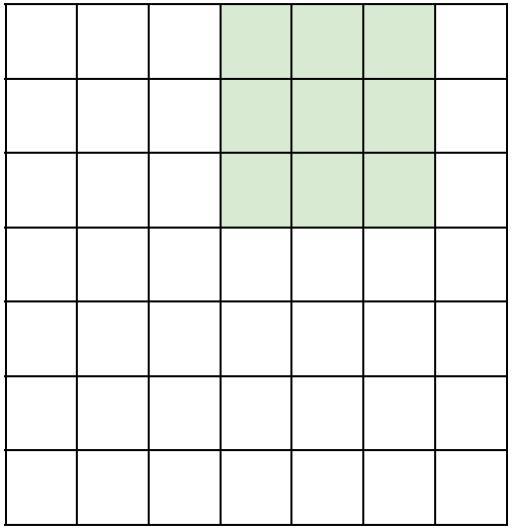
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

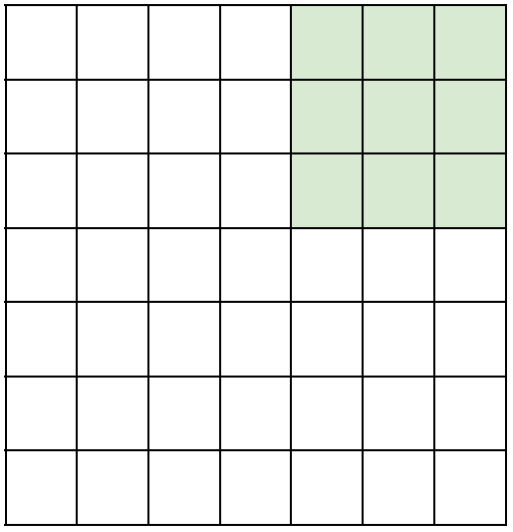
elementwise multiplication and sum of a filter and the signal (image)







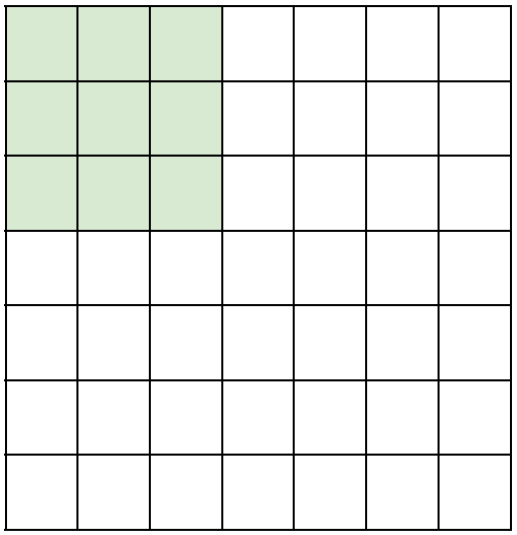


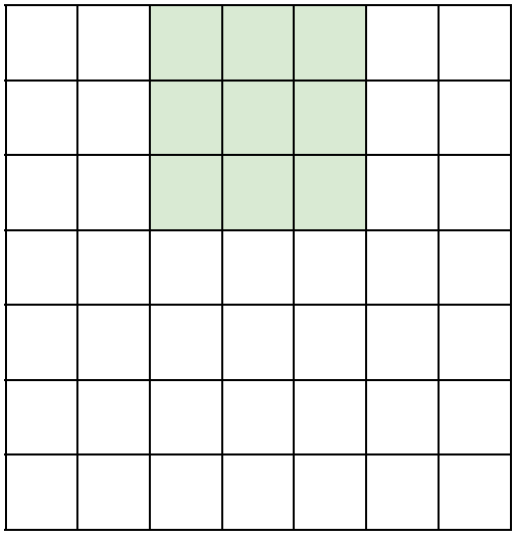


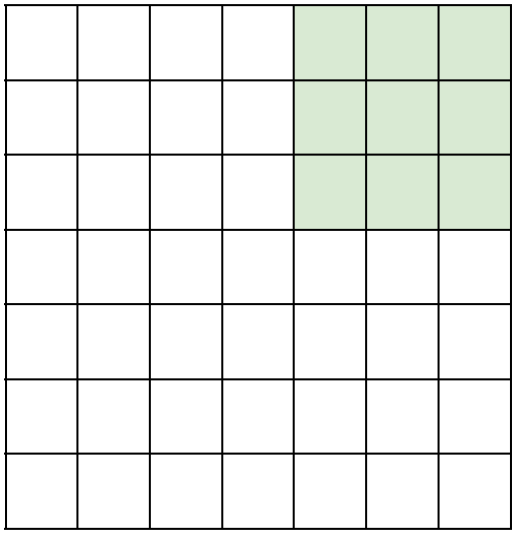
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Stride

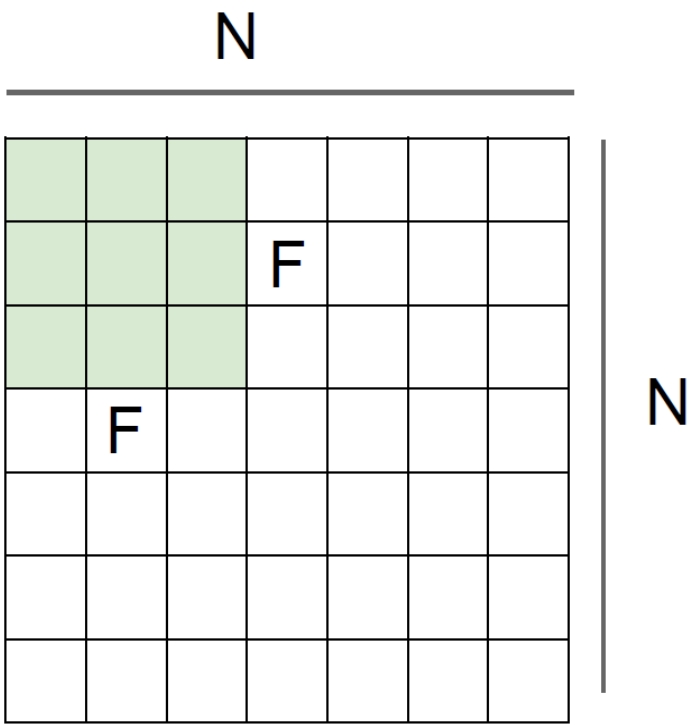






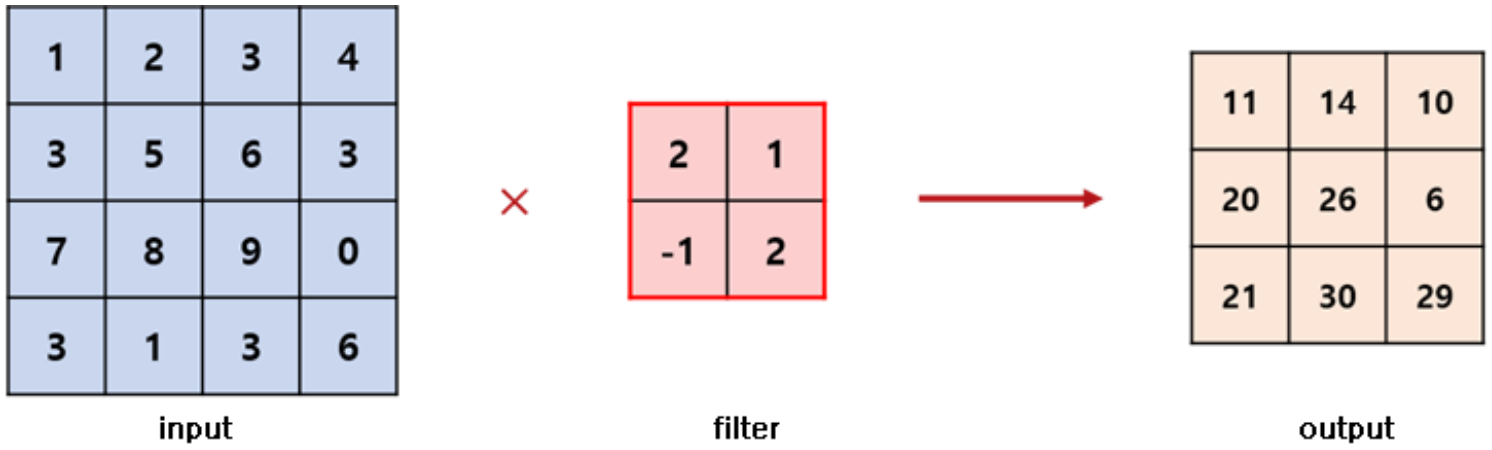
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

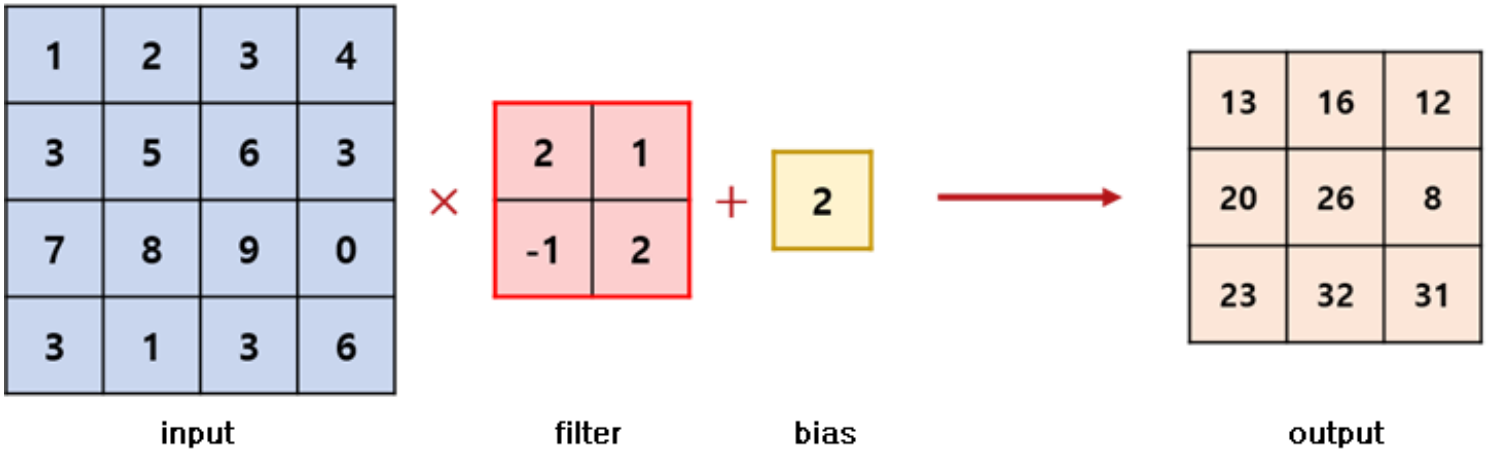
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**



Output size:
(N - F) / stride + 1

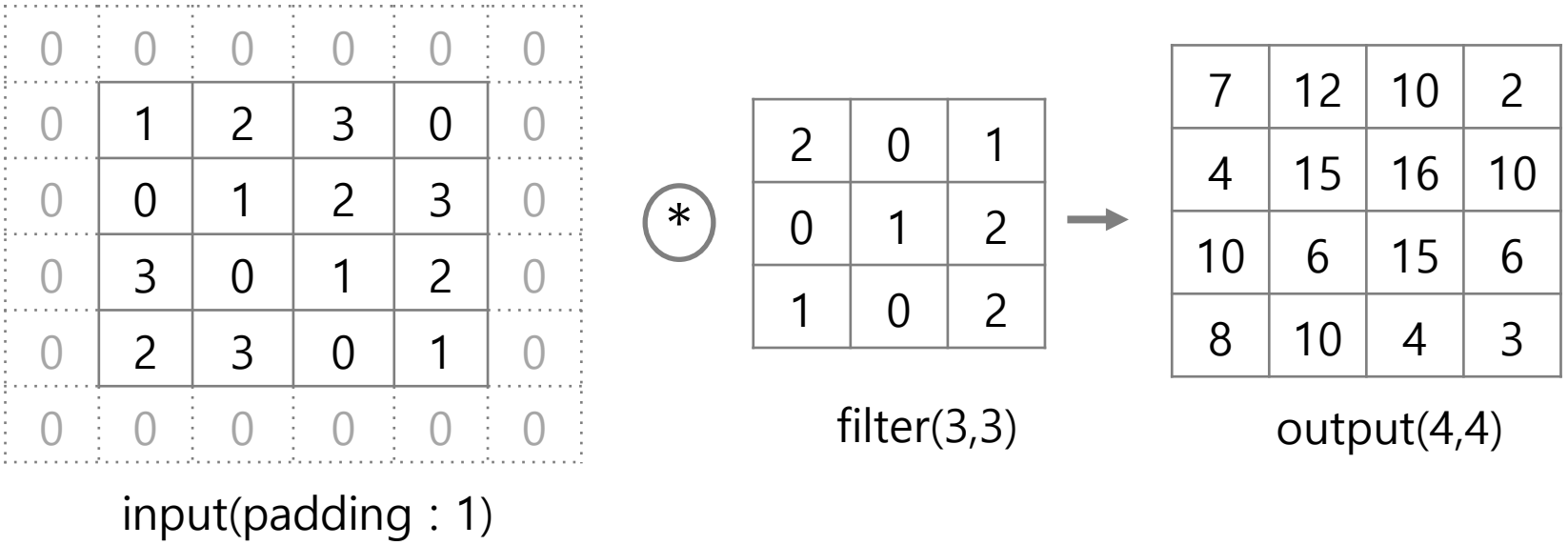
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\


1 2 3 4
3 5 6 3
7 8 9 0
3 1 3 6
input
×
2 1
-1 2
filter
11 14 10
20 26 6
21 30 29
output

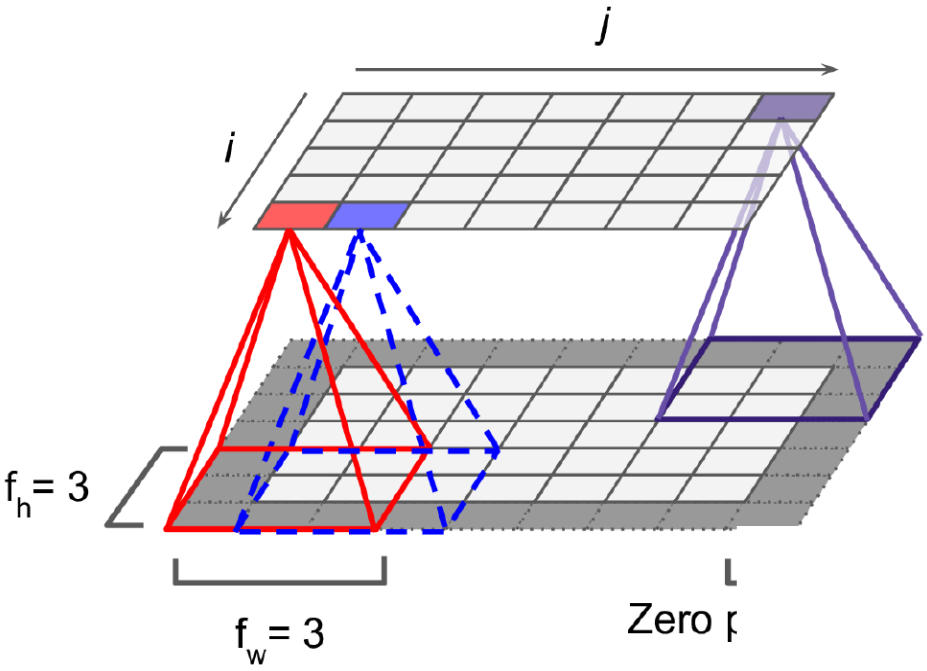

1 2 3 4
3 5 6 3
7 8 9 0
3 1 3 6
input
×
2 1
-1 2
filter
+
2
bias
13 16 12
20 26 8
23 32 31
output

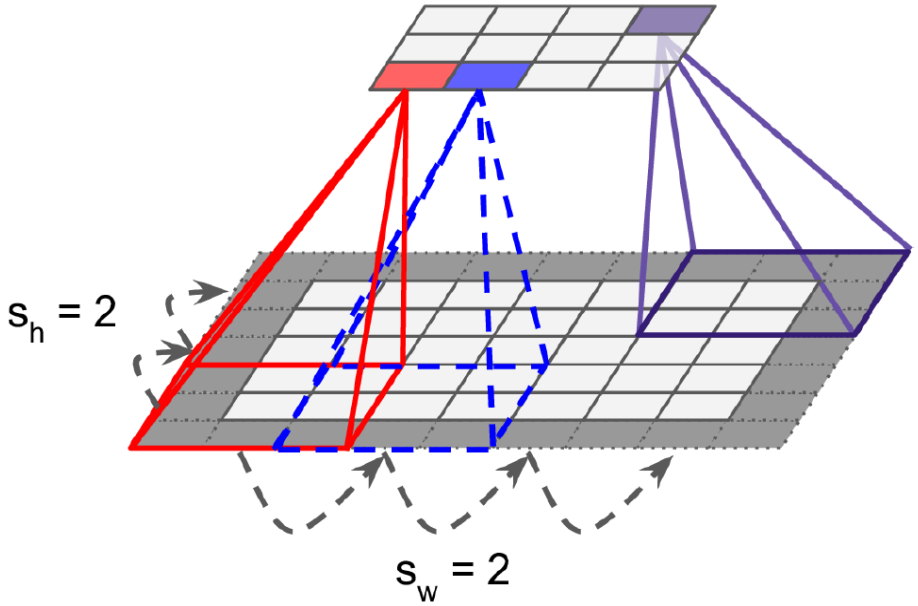
패딩(padding)

- 패딩은 출력 크기를 조정할 목적으로 사용
  - 합성곱 연산을 거칠 때마다 크기가 작아지면 출력 크기가 1이 되는 것을 방지
- 입력 데이터의 공간적 크기를 고정한 채로 다음 계층에 전달 가능

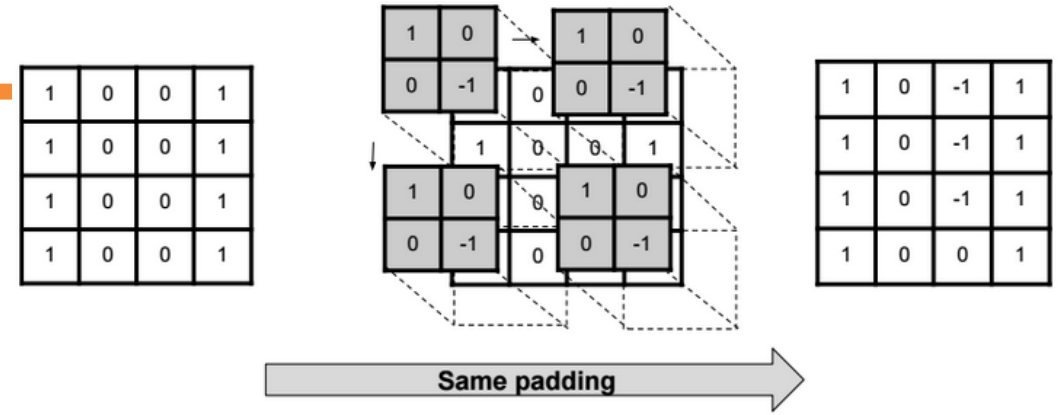


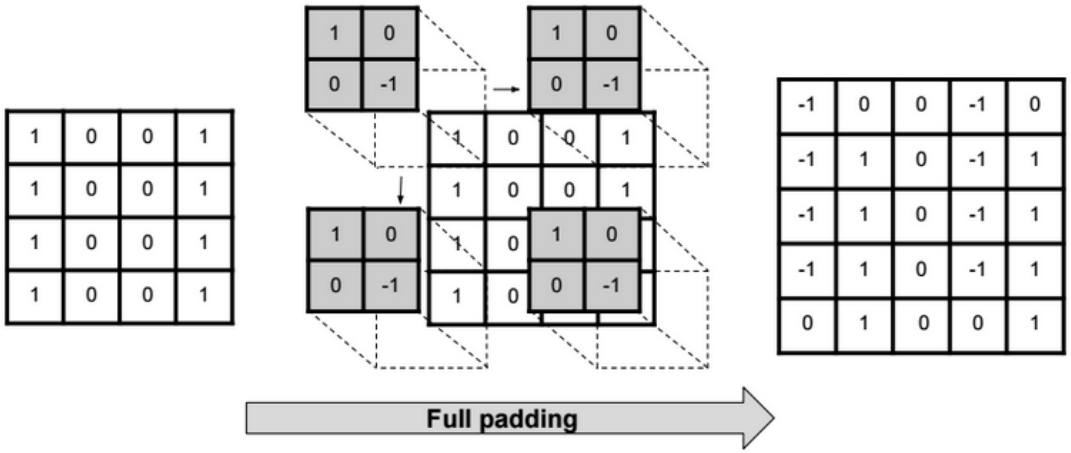
Zero padding & Stride

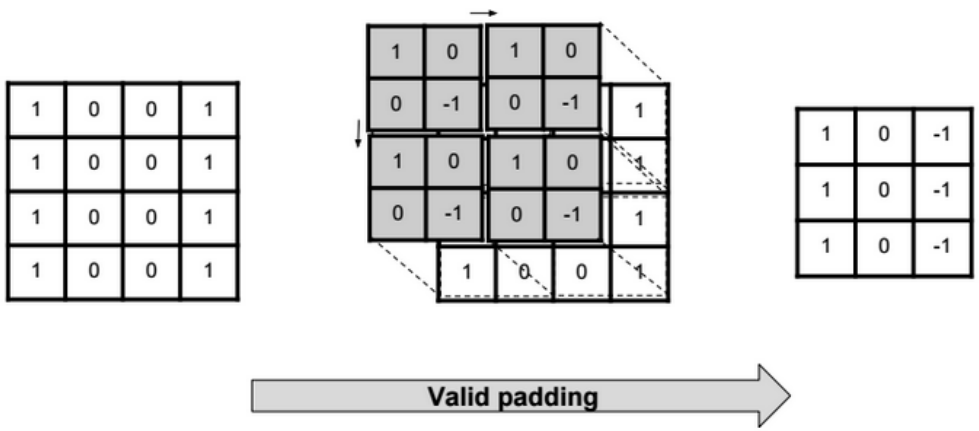




Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

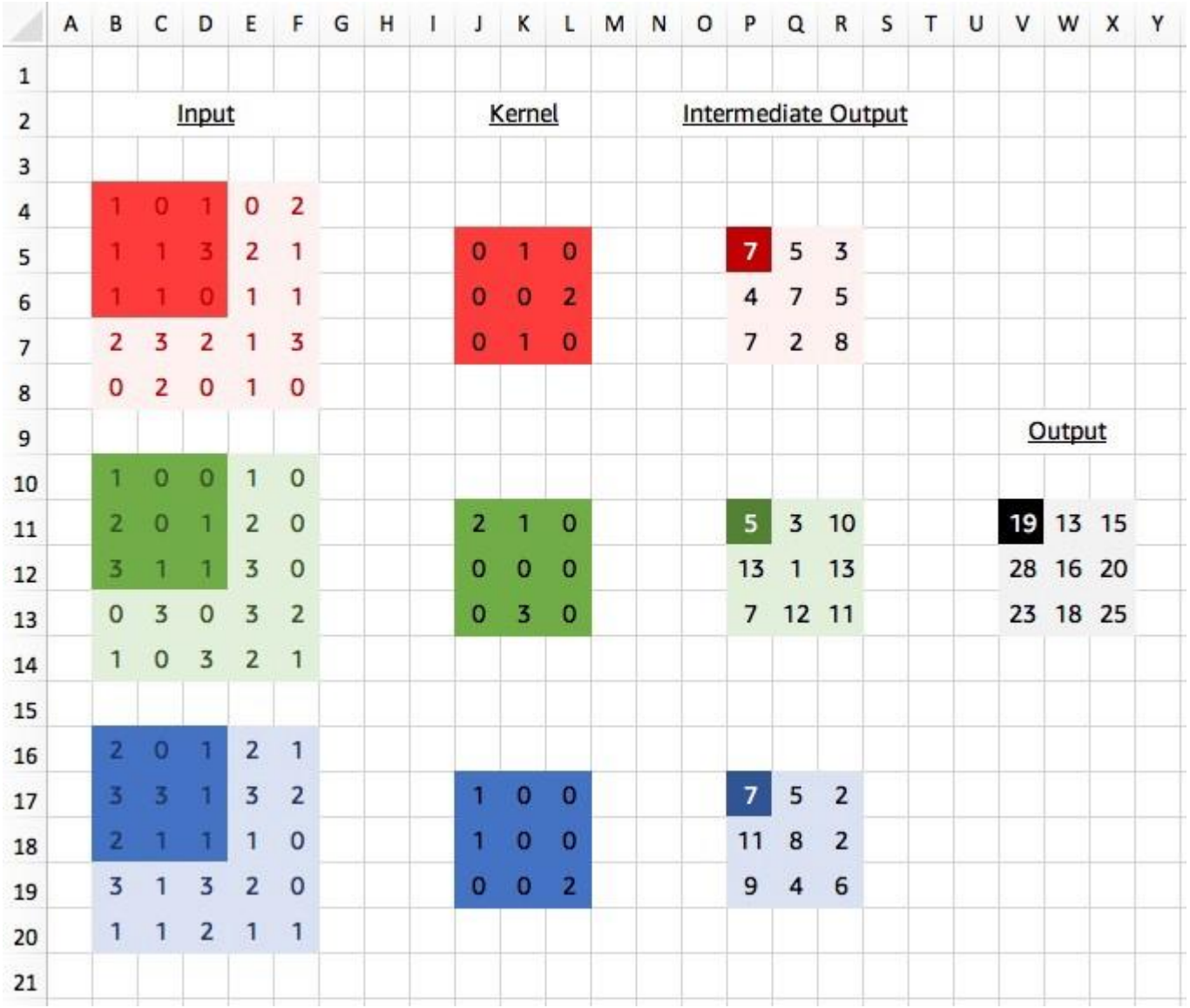






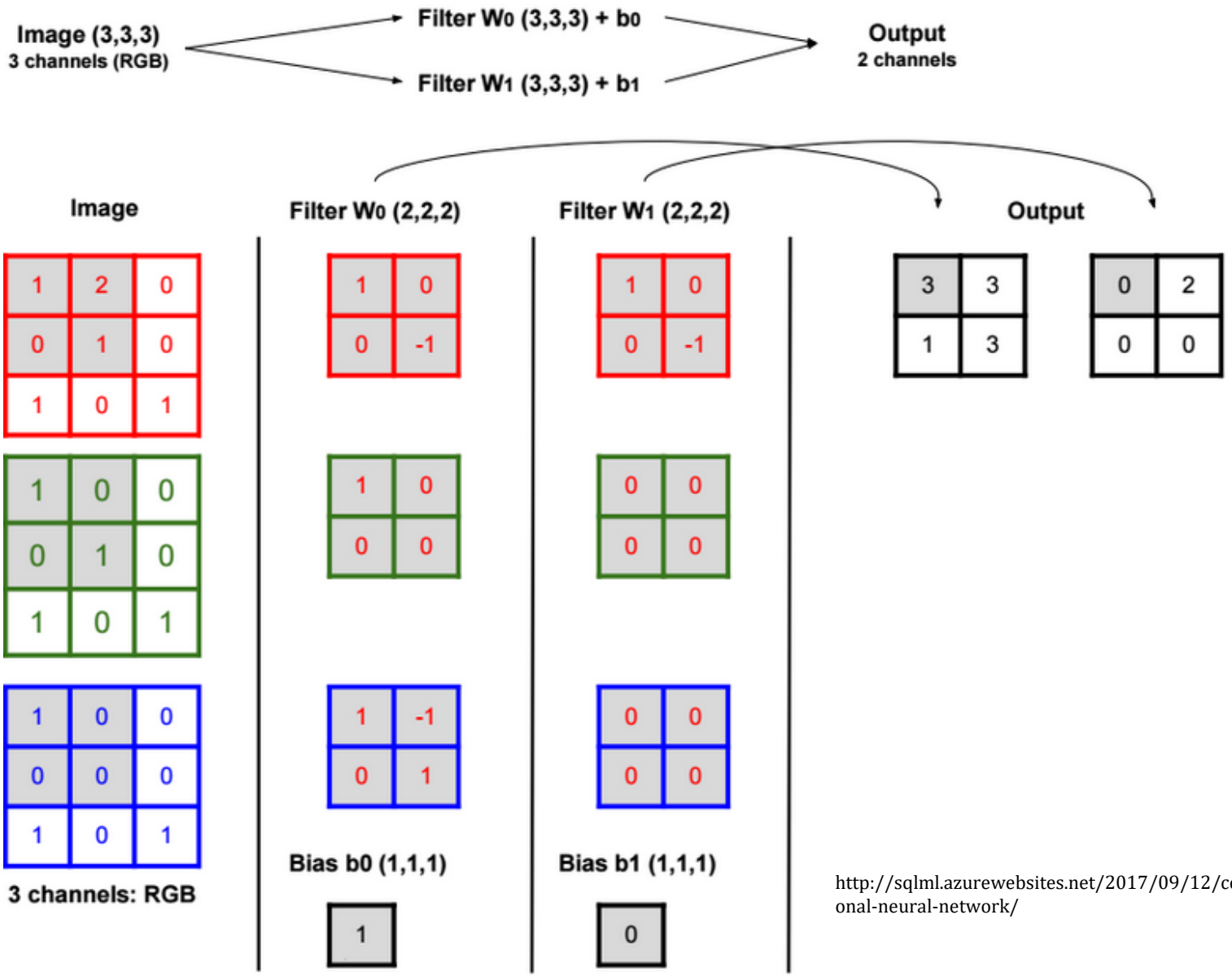
http://sqlml.azurewebsites.net/2017/09/12/convolutional-neural-network/

Filter 1개



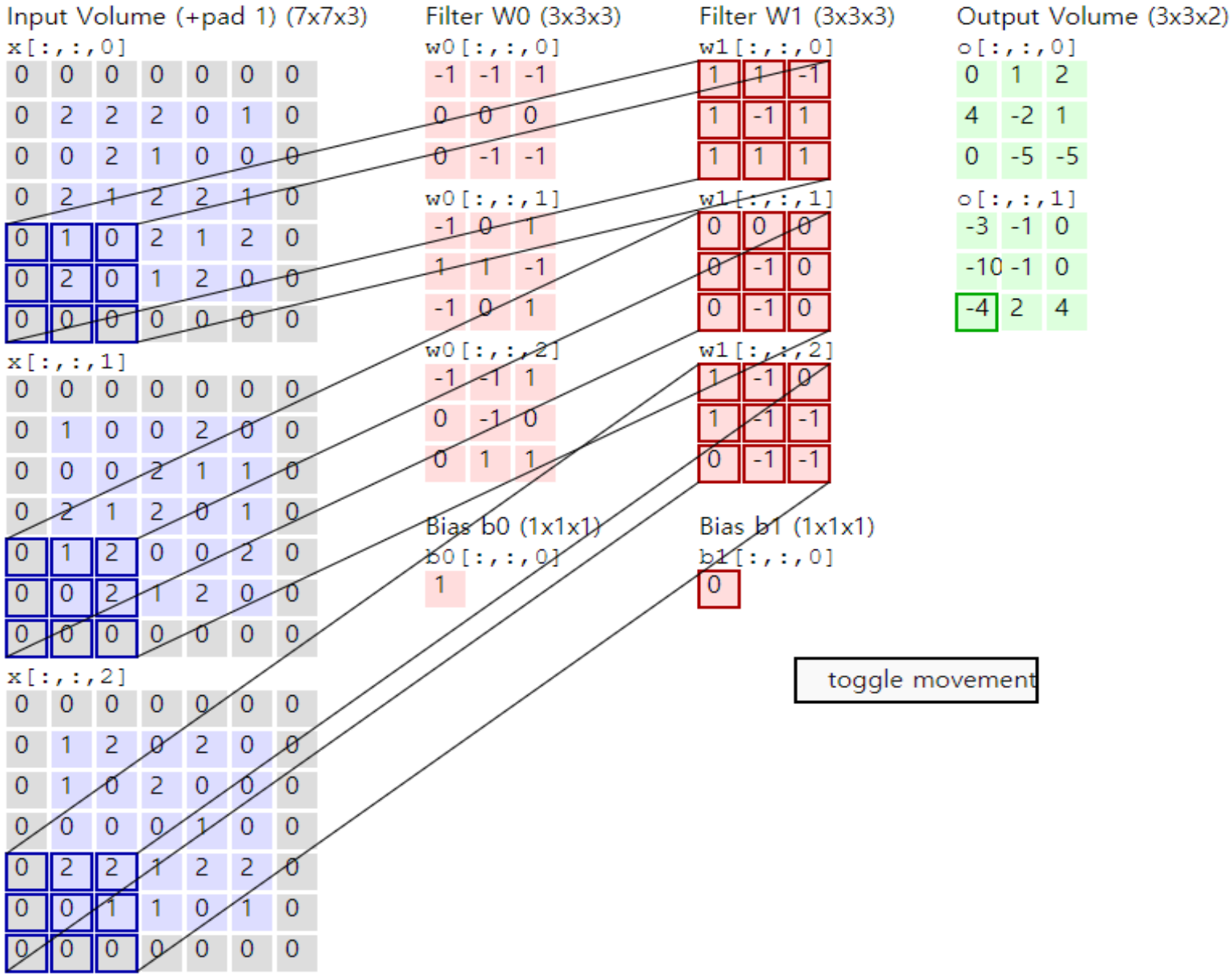
https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108

Filter 2개



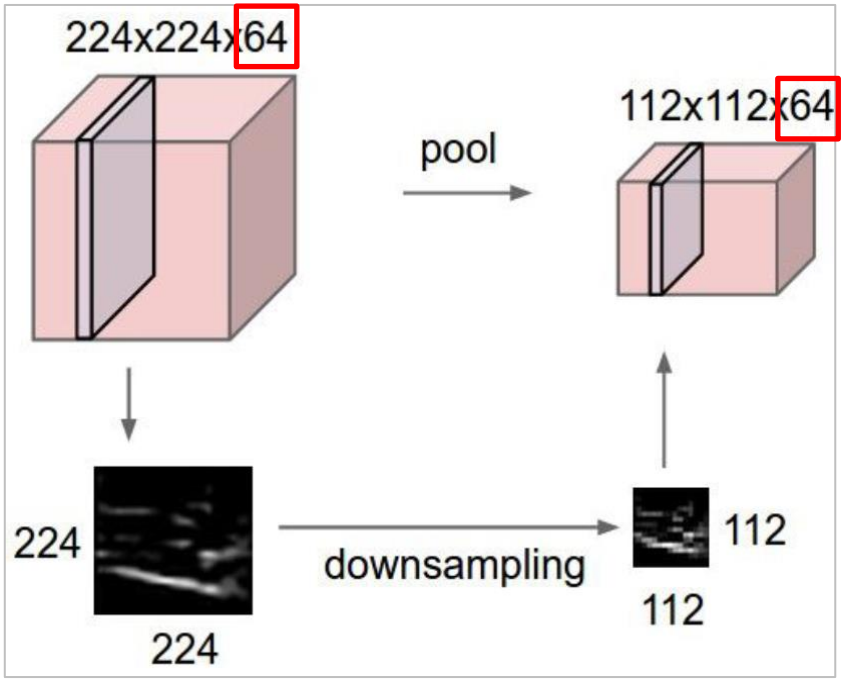
http://sqlml.azurewebsites.net/2017/09/12/convolutional-neural-network/

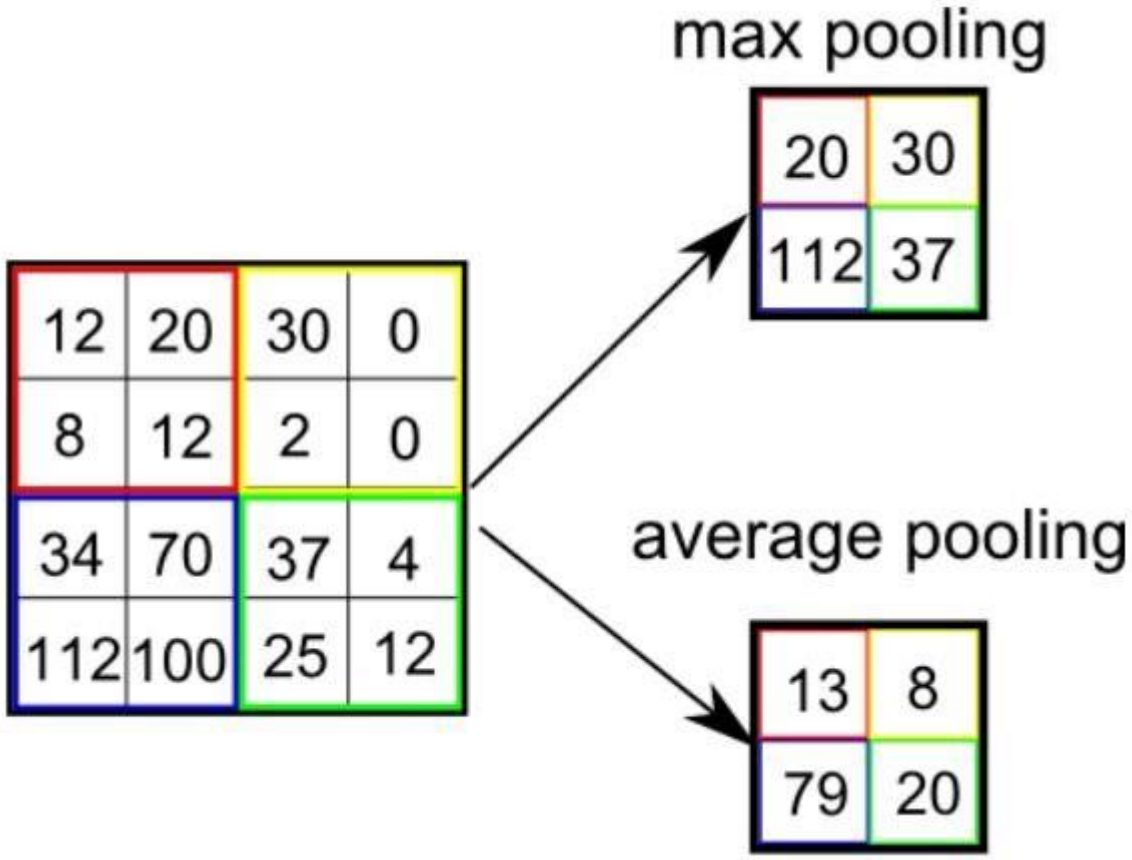
http://cs231n.github.io/convolutional-networks/



Pooling Layer

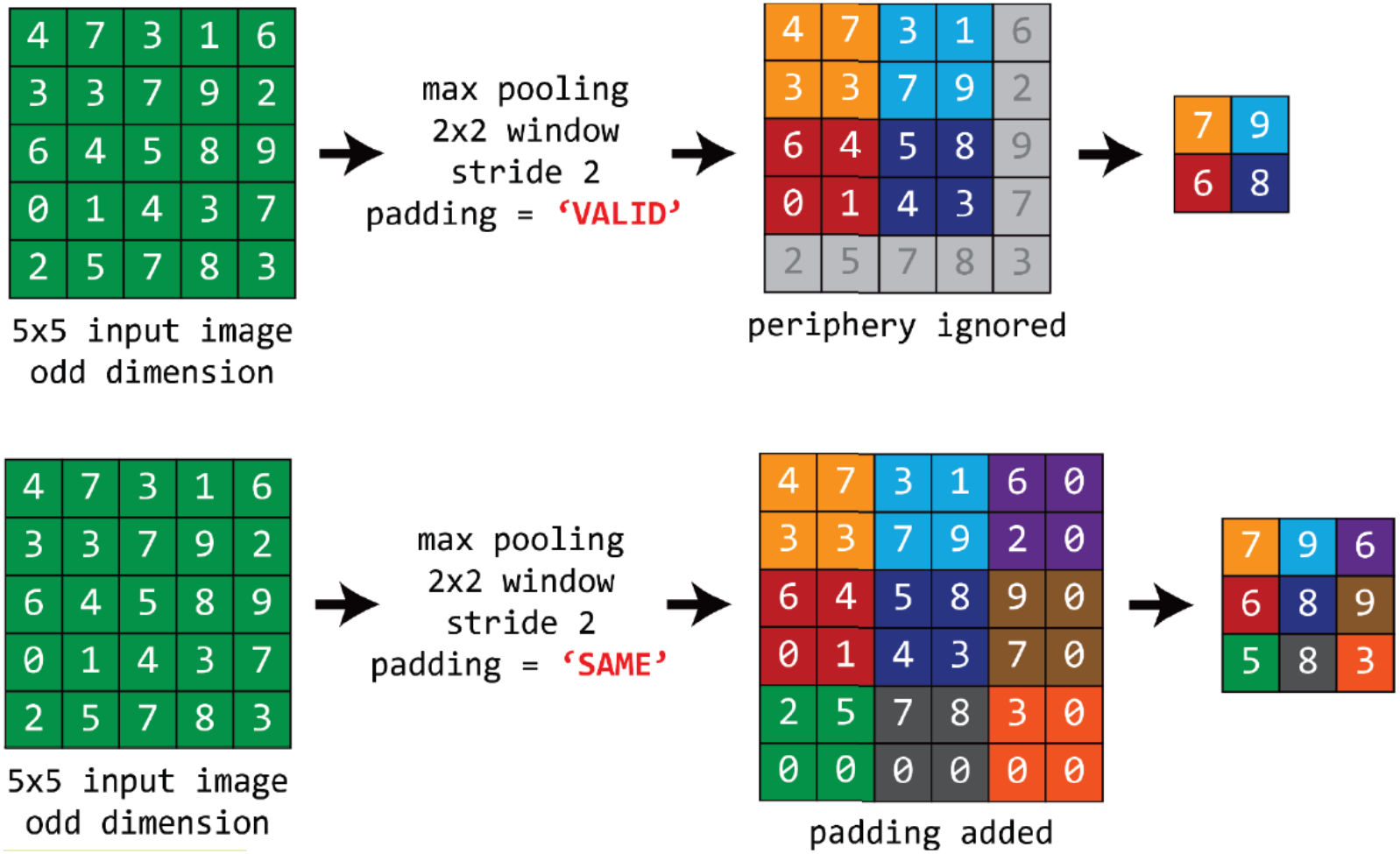
- Pooling
  - 네트워크의 파라미터의 개수나 연산량을 줄이기 위해 사이즈를 줄이는 것
  - 과적합(over fitting)을 조절하는 효과도 있다.
  - 사이즈 2 × 2와 stride 2를 가장 많이 사용
  - Max Pooling, Average Pooing
- 특징
  - 입력에 대해 항상 같은 연산을 하므로 파라미터는 따로 존재하지 않는다
  - 채널 수가 변하지 않는다. 입력데이터의 채널 수 그대로 출력 데이터로 내보낸다.
  - 입력데이터가 조금 변해도 풀링의 결과는 잘 변하지 않는다
    - 입력의 변화에 영향을 적게 받는다.



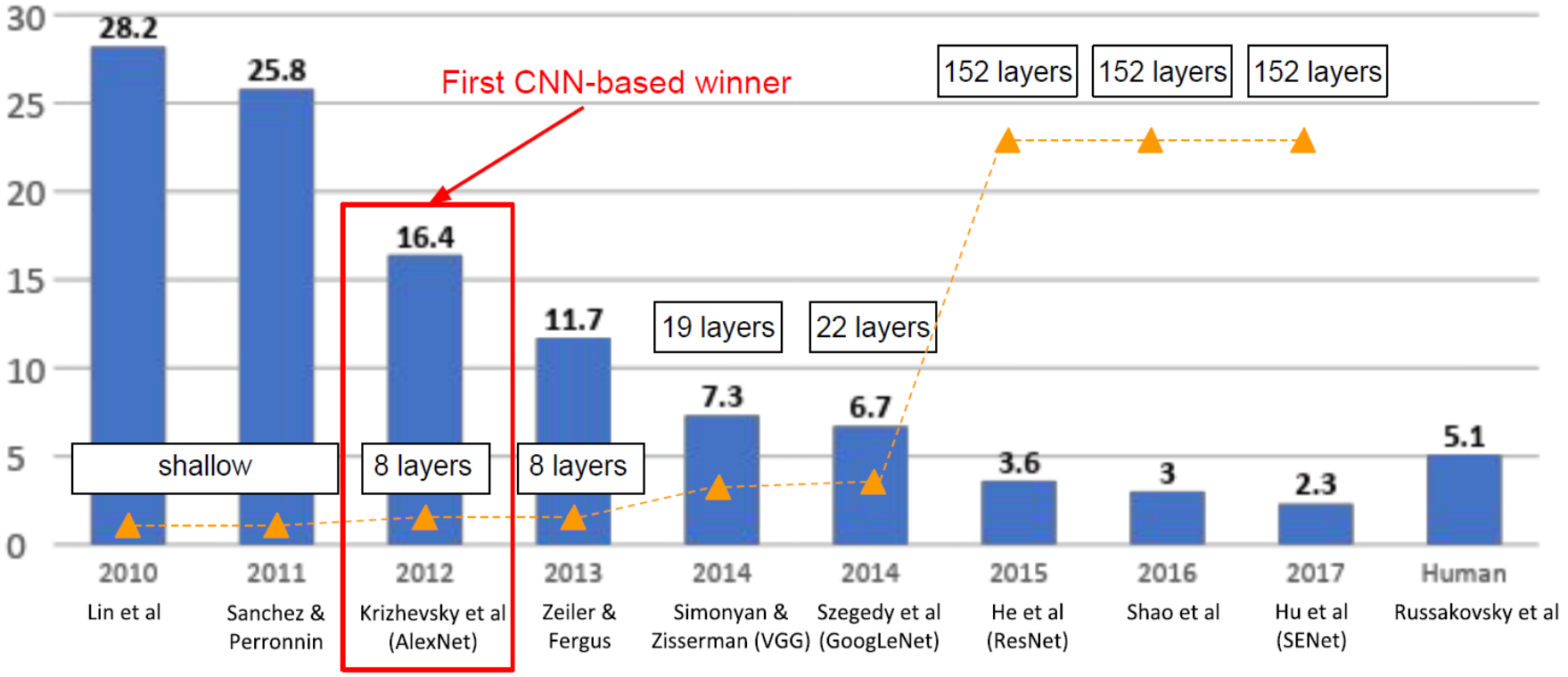

max pooling
20 30
112 37
12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
average pooling
13 8
79 20

Pooling with Padding

- Better handle odd dimesions

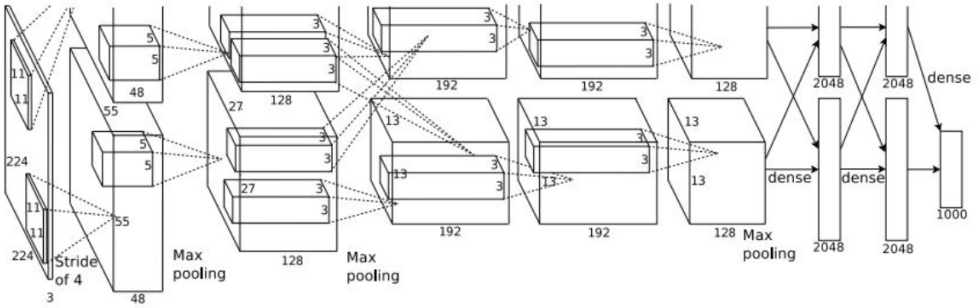


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: AlexNet

[Krizhevsky et al. 2012]

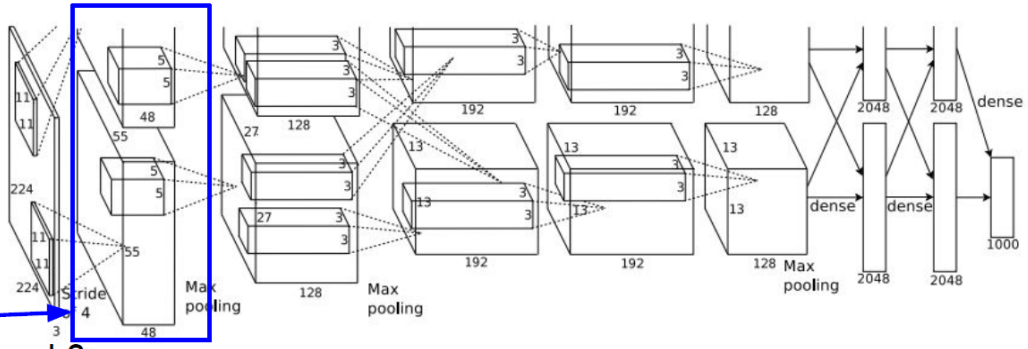


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: AlexNet

[Krizhevsky et al. 2012]



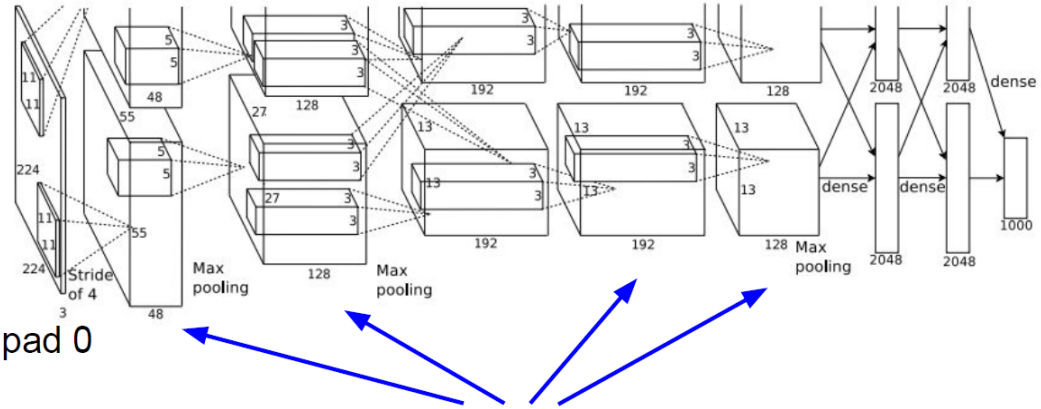
[55x55x48] x 2

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

Case Study: AlexNet

[Krizhevsky et al. 2012]

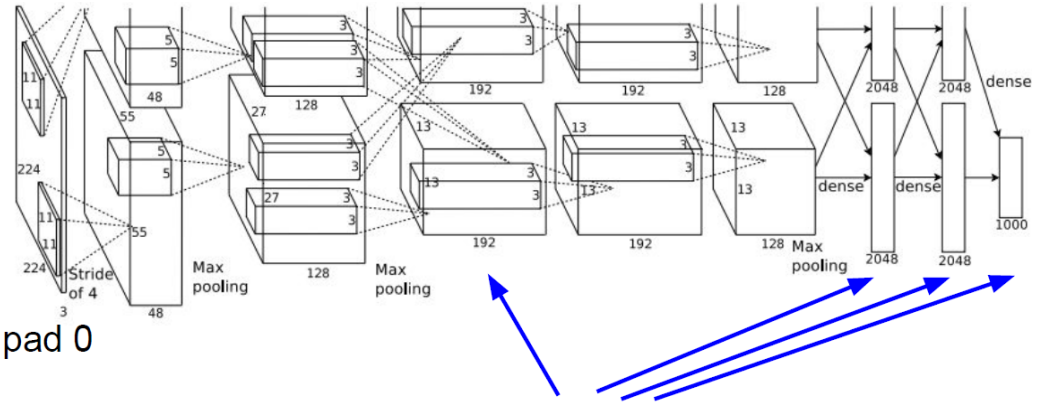


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

CONV1, CONV2, CONV4, CONV5:
Connections only with feature maps on same GPU

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

CONV3, FC6, FC7, FC8:
Connections with all feature maps in preceding layer, communication across GPUs