Apply to Machine Learning

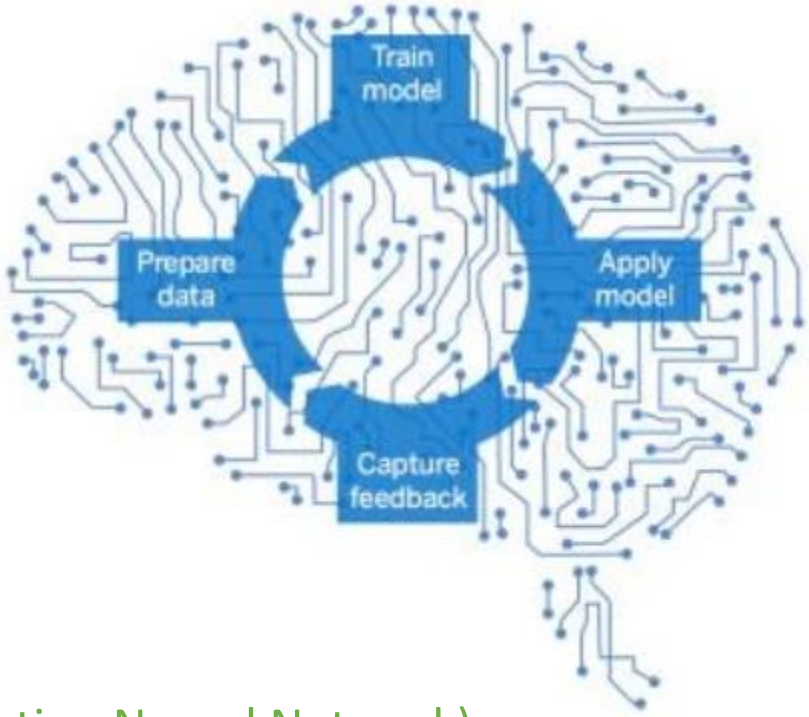
Week 11
Machine Learning

이선우(Seon Woo Lee)

인하대학교

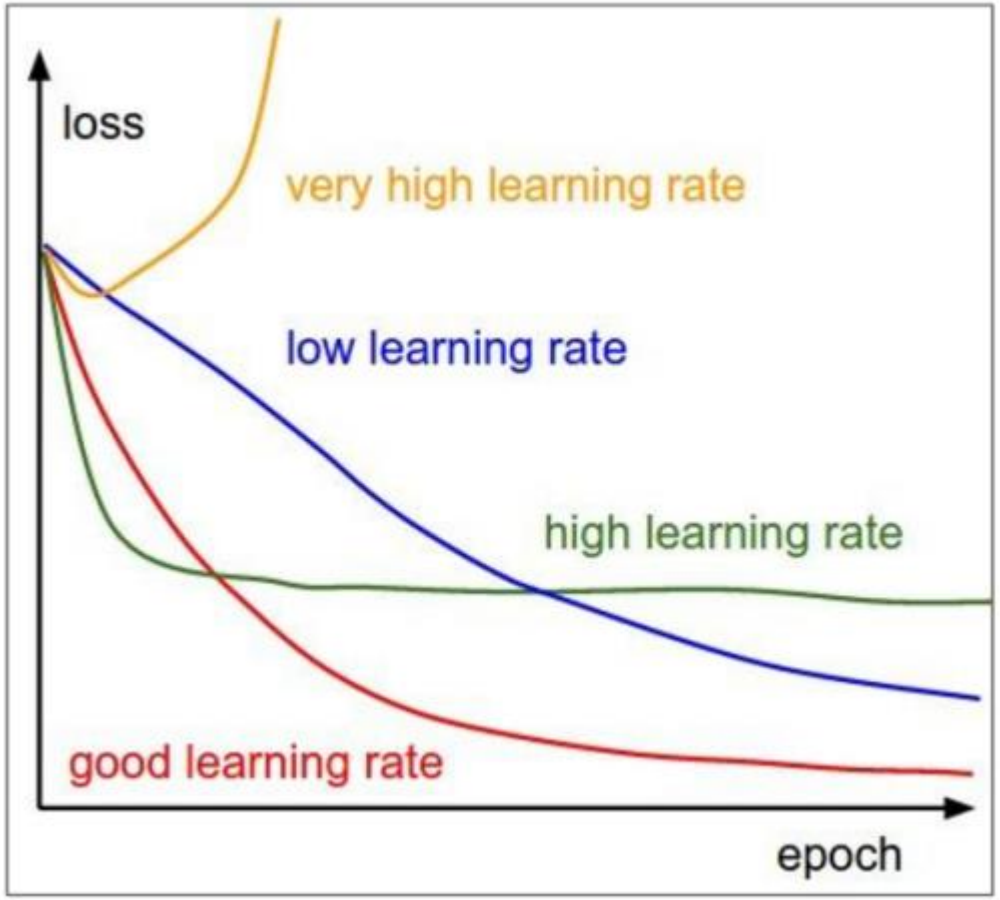
HCI Lab
Inha University
Human-Computer Interaction Lab

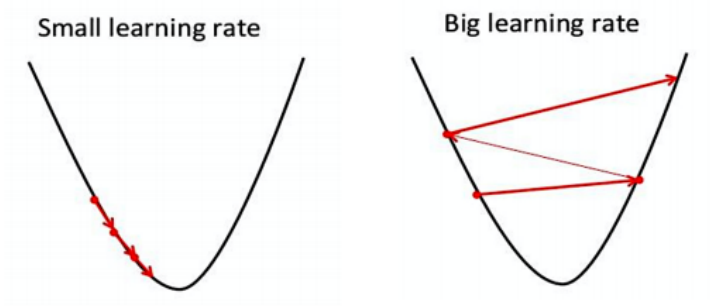
Contents

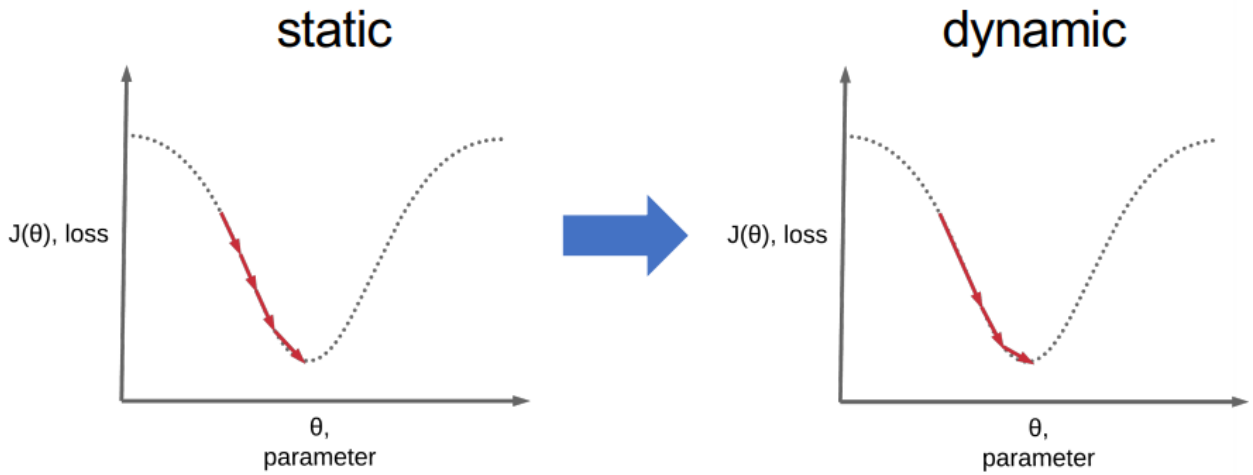


1 Review

학습속도 스케줄링



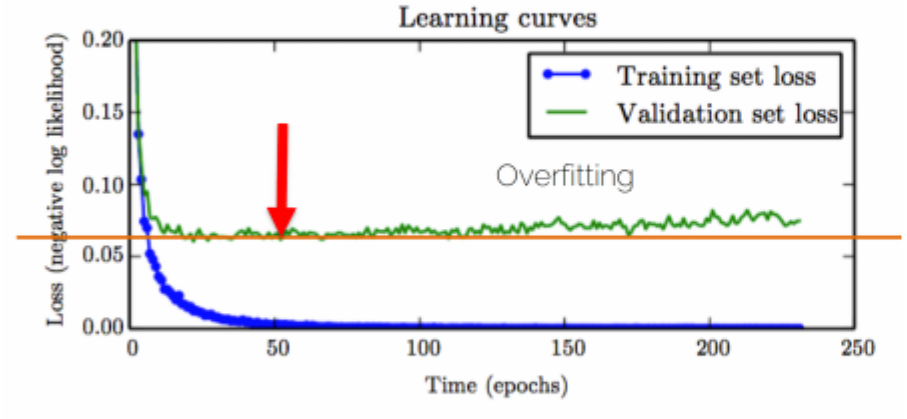




오버피팅 대응책(+)

- 그 밖의 과적합(Overfitting) 피하는 방법

Early Stopping: Always do this




```
# Additional information
EPOCH = 5
PATH = "model.pt"
LOSS = 0.4

torch.save({
            'epoch': EPOCH,
            'model_state_dict': net.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': LOSS,
            }, PATH)
```

checkpont

Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot
that worked best on val

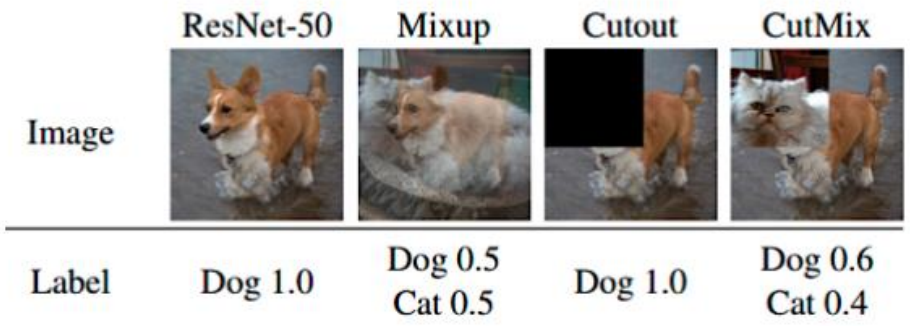
오버피팅 대응책(+)

- 그 밖의 과적합(Overfitting) 피하는 방법(이미지)

Regularization: Cutmix

Training: Train on random blends of images
Testing: Use original images

Examples:
Dropout
Data Augmentation
DropConnect
Cutout / Random Crop
Mixup
Cutmix



Yun, Sangdoo, et al. "Cutmix: Regularization strategy to train strong classifiers with localizable features." *ICCV* 2019

Regularization - In practice

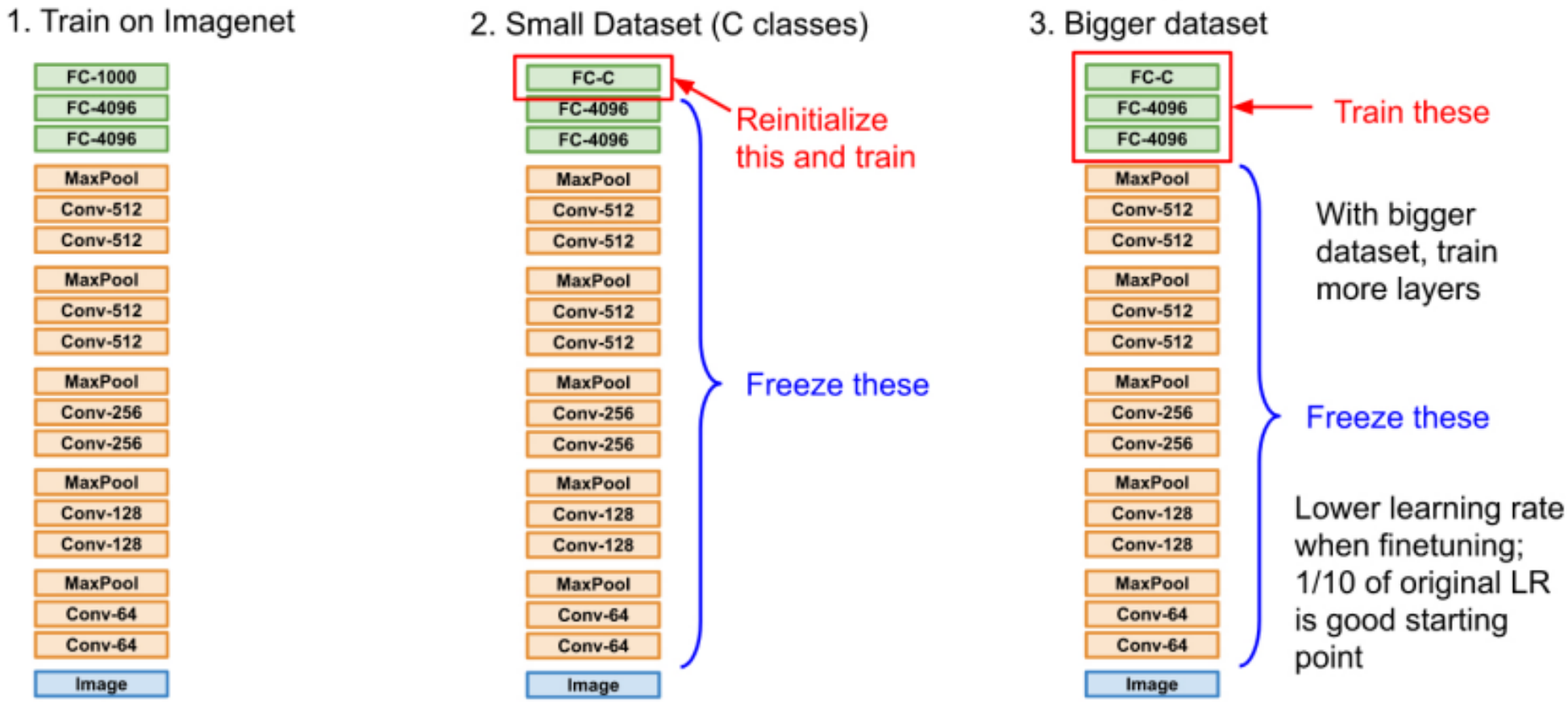
Training: Add random noise
Testing: Marginalize over the noise

Examples:
Dropout
Data Augmentation
DropConnect
Cutout / Random Crop
Mixup
Cutmix

- Consider dropout for large fully-connected layers
- Batch normalization and data augmentation almost always a good idea
- Try cutout and mixup especially for small classification datasets
- Try Cutmix if possible

전이학습(Transfer Learning)

- 사전에 학습된 데이터를 이용하여 문제를 해결하는 방법



2 Practice

Pytorch MLP Tutorials

https://github.com/LEE-SEON-WOO/DL-Tutorials-Py_or_Tf

DL-Tutorials-Py_or_Tf

Star 0 LinkedIn

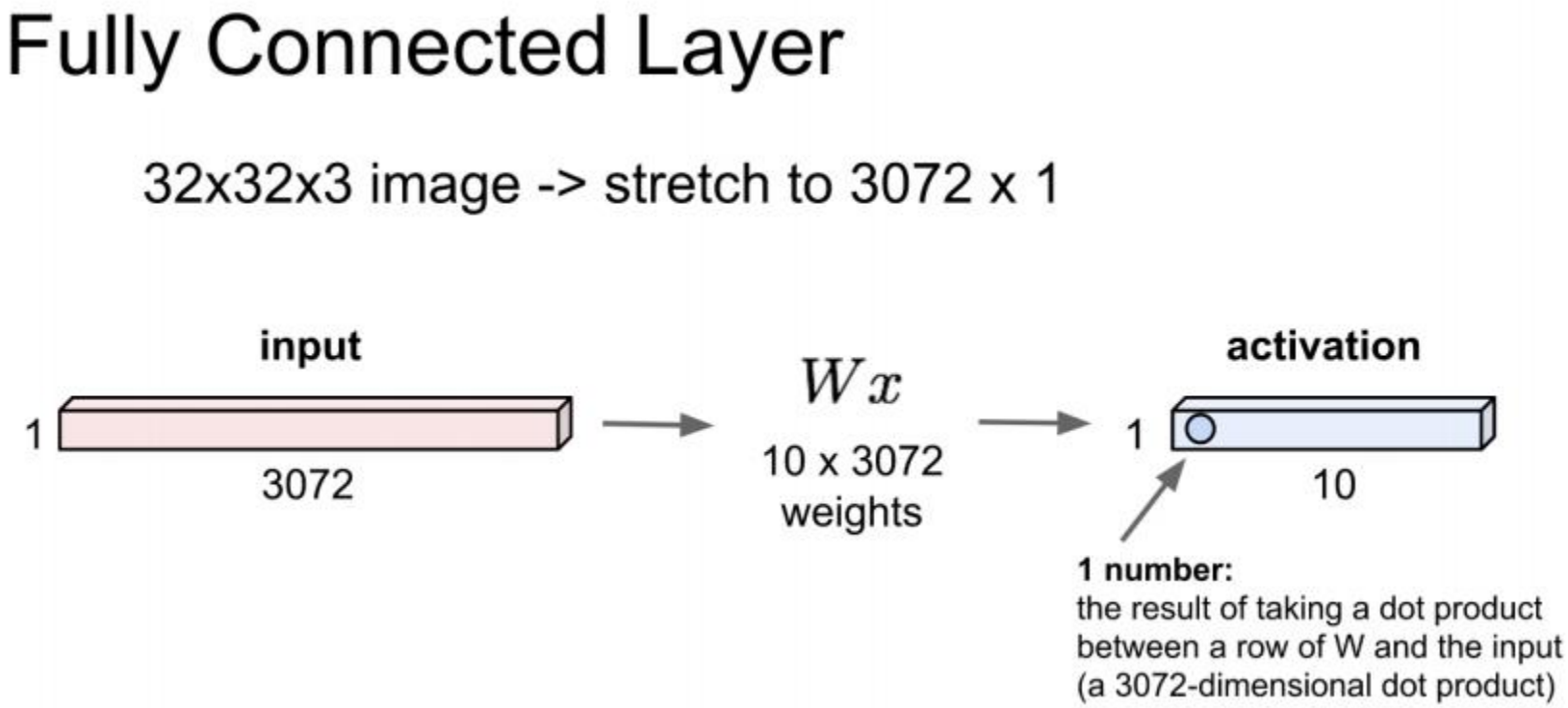
🔥Among the top 10 ML repos on GitHub

- Learn Python DL-Tutorials-Py_or_Tf with notebooks.
- Use data science libraries like NumPy and Pandas.
- Implement basic ML models in TensorFlow 2.0 + Keras or PyTorch.
- Learn best practices with clean code, simple math and visualizations.

Notebooks	Python	NumPy
Pandas	TensorFlow	PyTorch
Linear Regression	Logistic Regression	Multilayer Perceptrons
Data & Models	Utilities	Preprocessing
Convolutional Neural Networks	Embeddings	Recurrent Neural Networks

2 CNN(Convolution Neural Network)

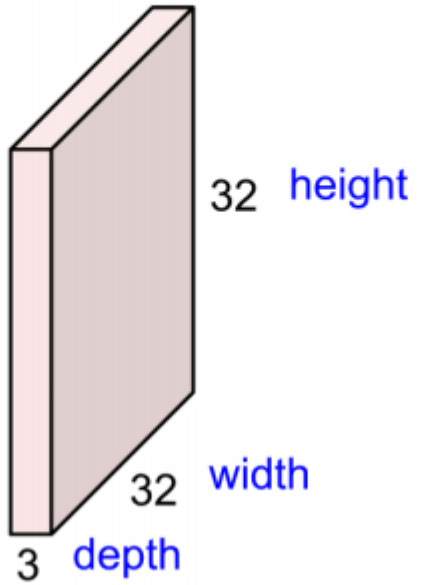
CNN(Convolution Neural Network)



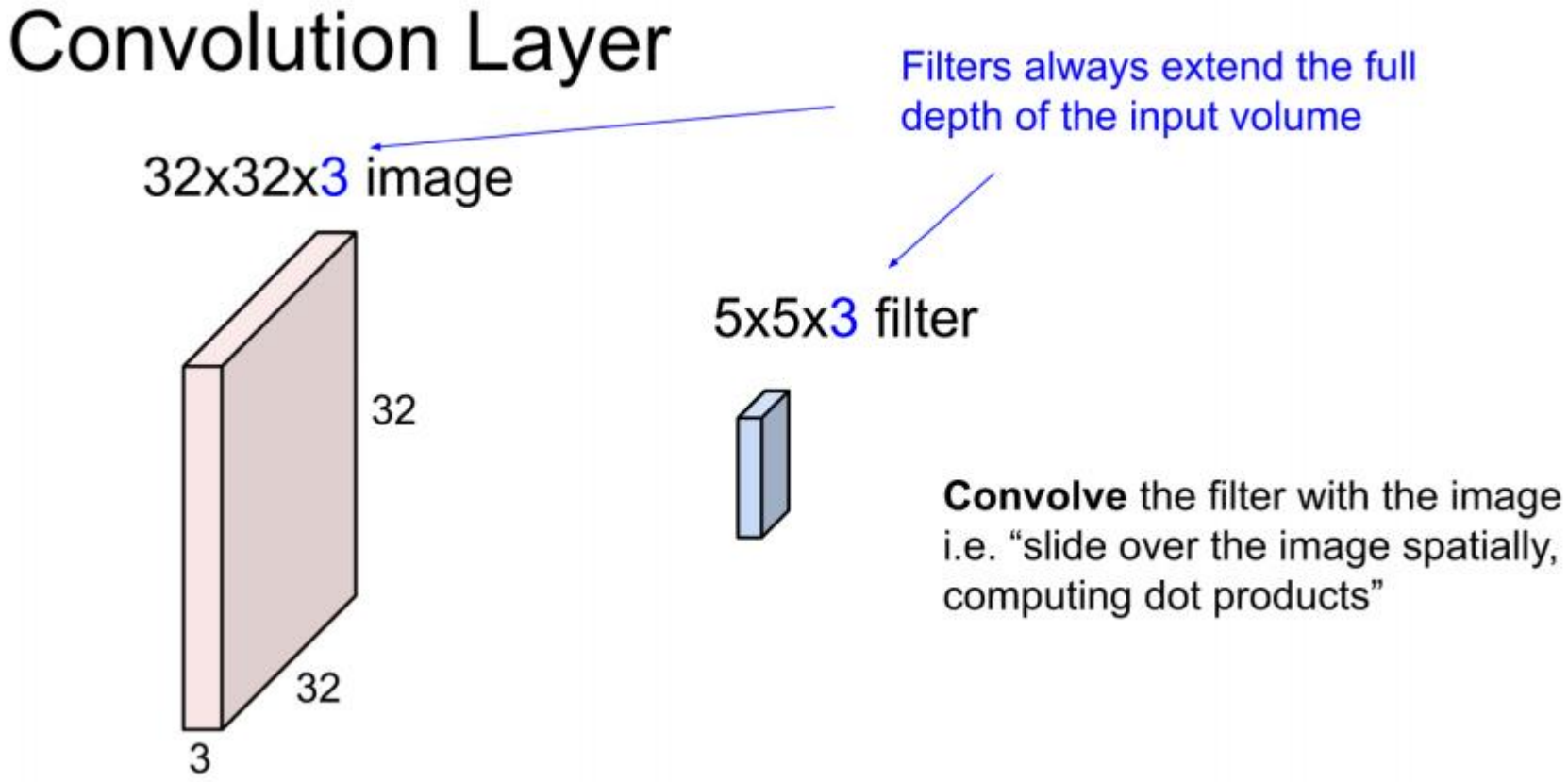
CNN(Convolution Neural Network)

Convolution Layer

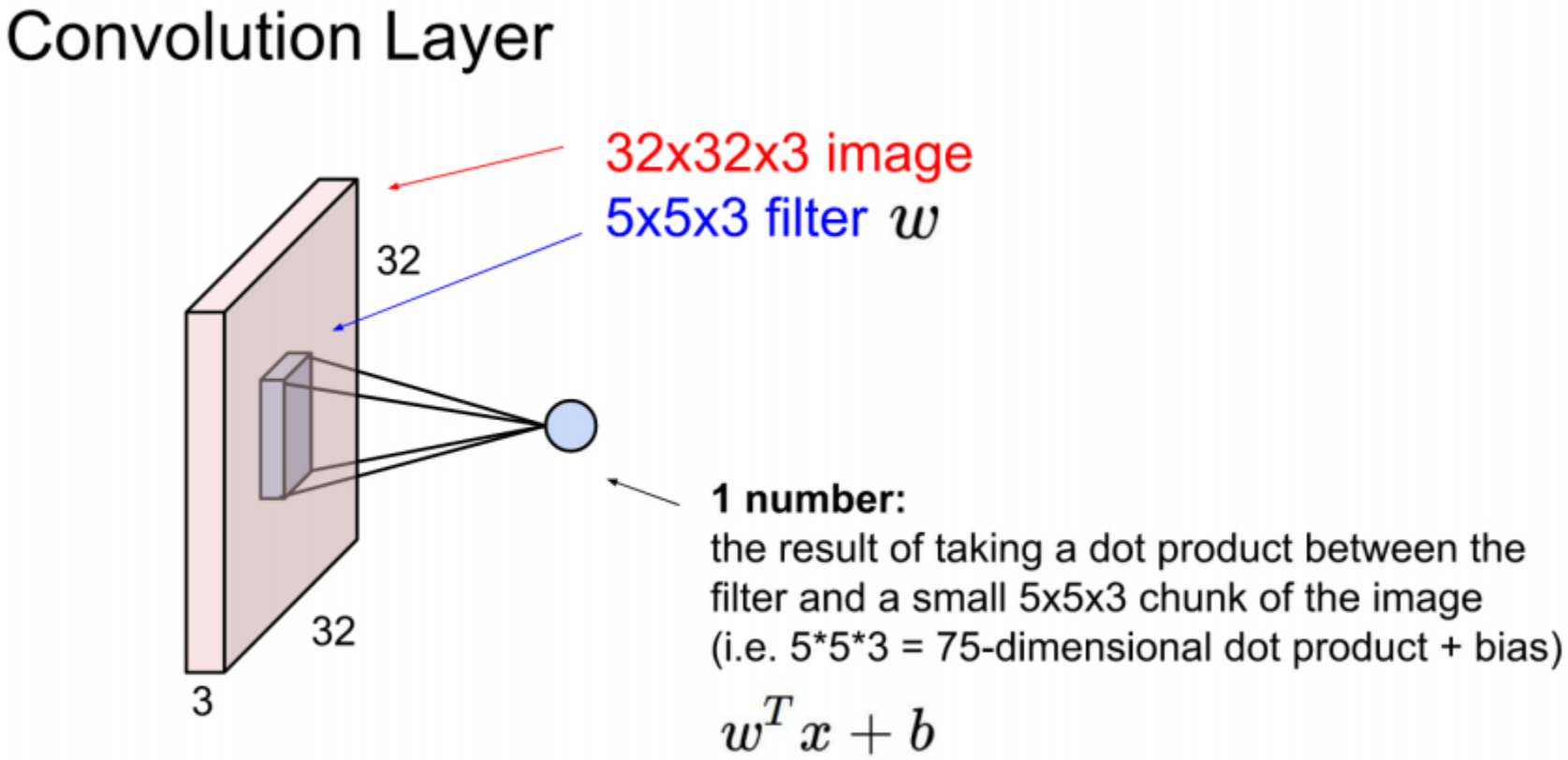
32x32x3 image -> preserve spatial structure



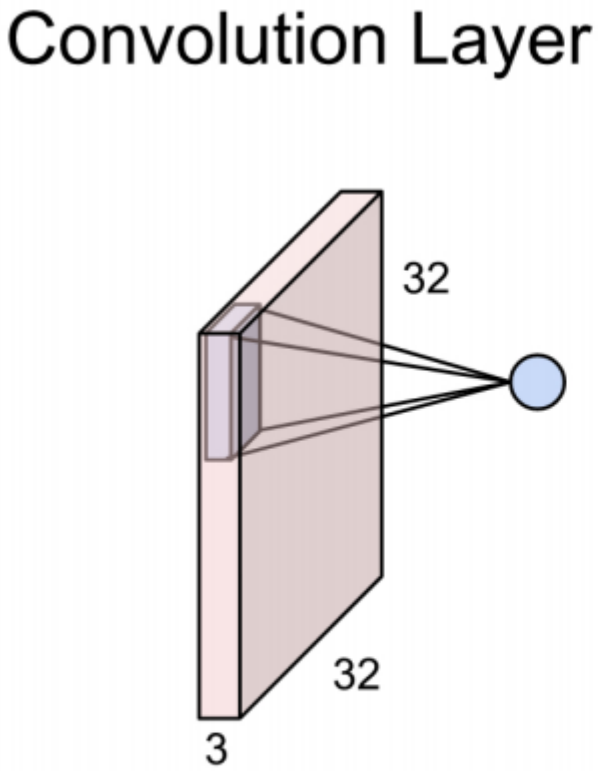
CNN(Convolution Neural Network)



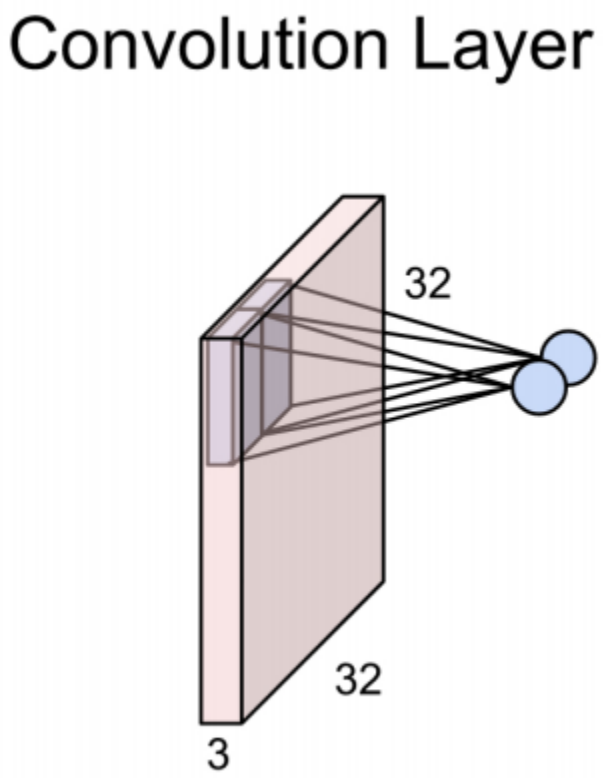
CNN(Convolution Neural Network)



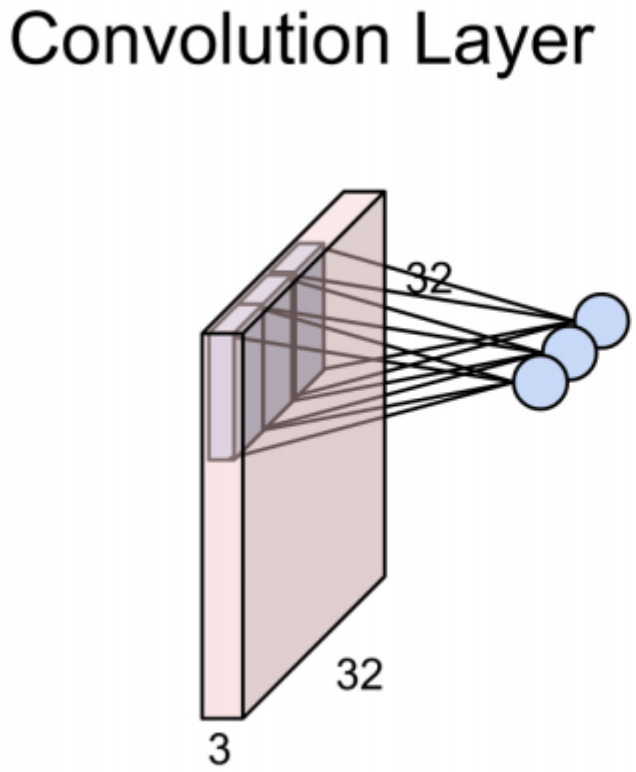
CNN(Convolution Neural Network)



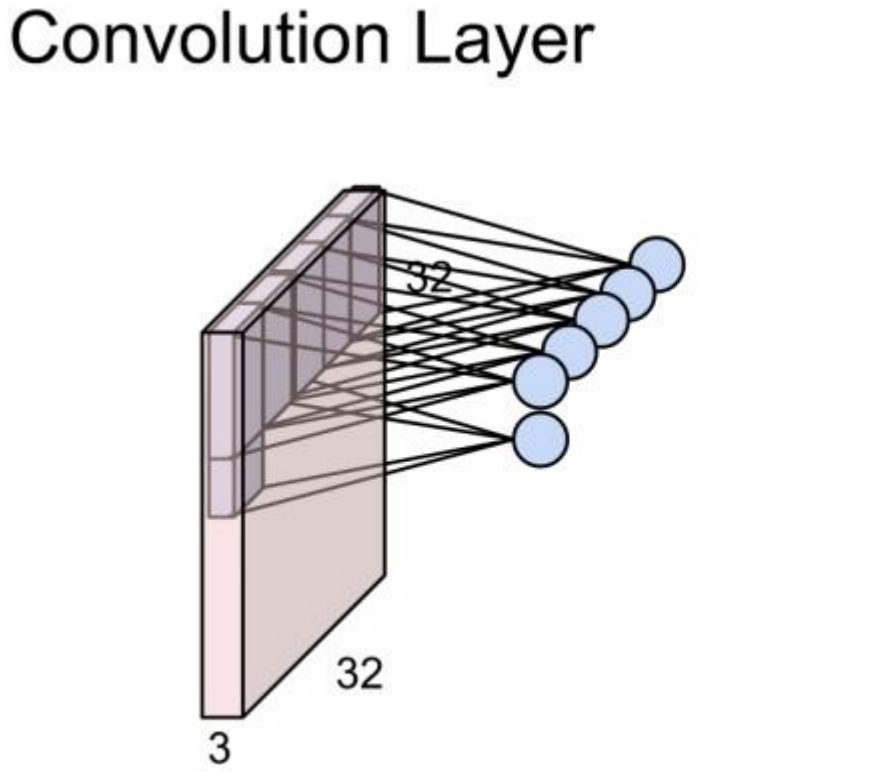
CNN(Convolution Neural Network)



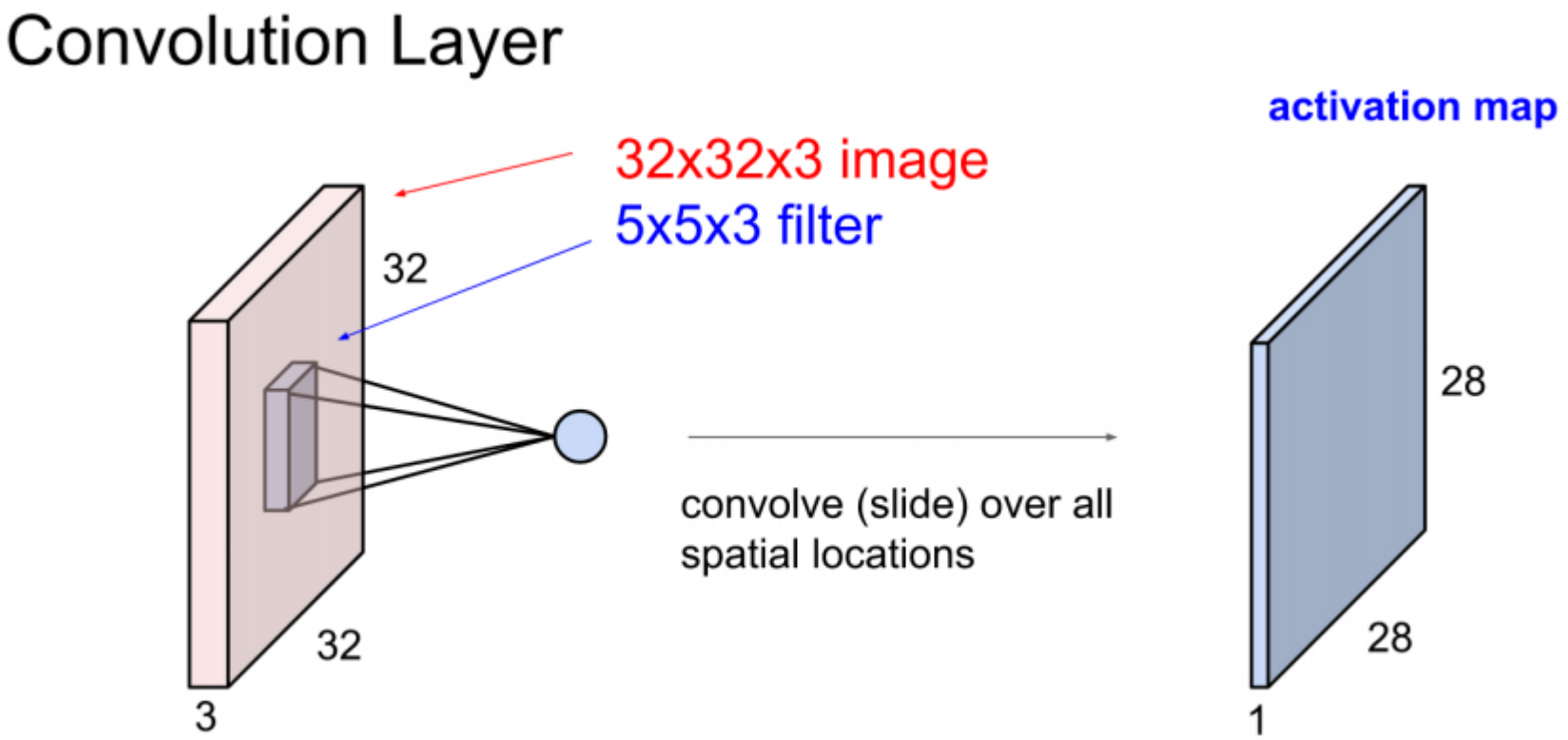
CNN(Convolution Neural Network)



CNN(Convolution Neural Network)



CNN(Convolution Neural Network)

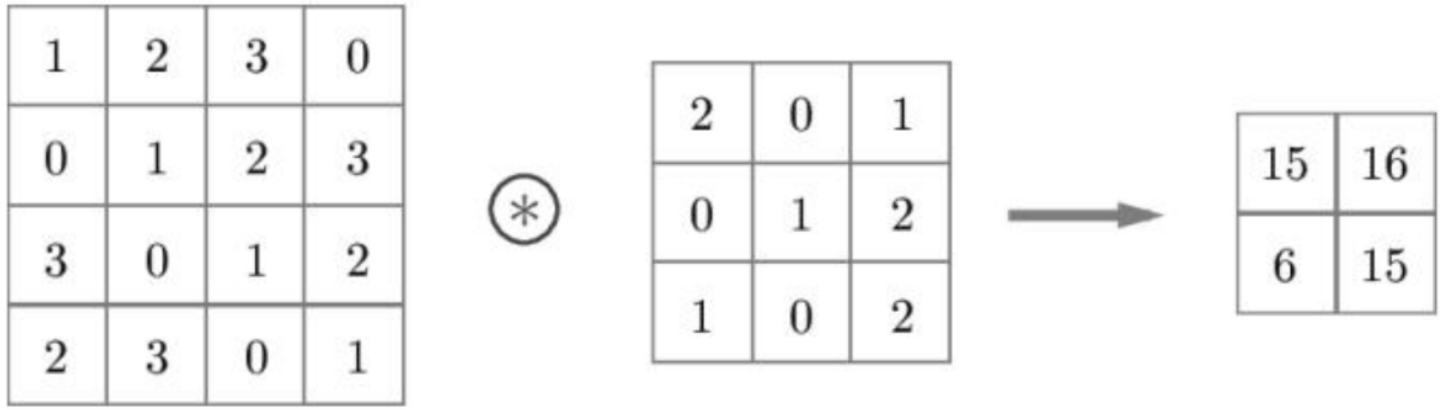


CNN(Convolution Neural Network)

- 완전연결 계층의 문제점: 데이터의 형상 무시
- 합성곱 연산 (필터 연산)

  *필터 = 커널

그림 7-3 합성곱 연산의 예 : 합성곱 연산을 ⊛ 기호로 표기



CNN(Convolution Neural Network)

그림 7-4 합성곱 연산의 계산 순서

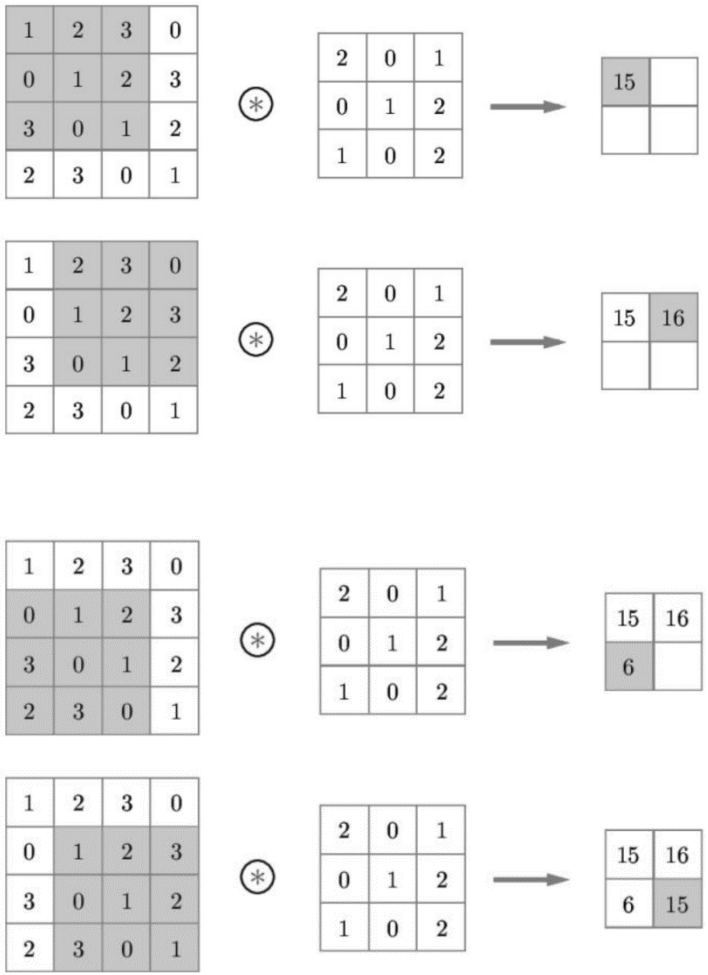
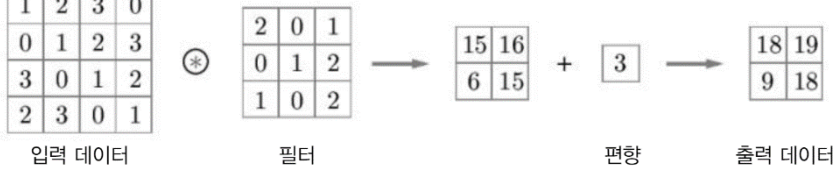
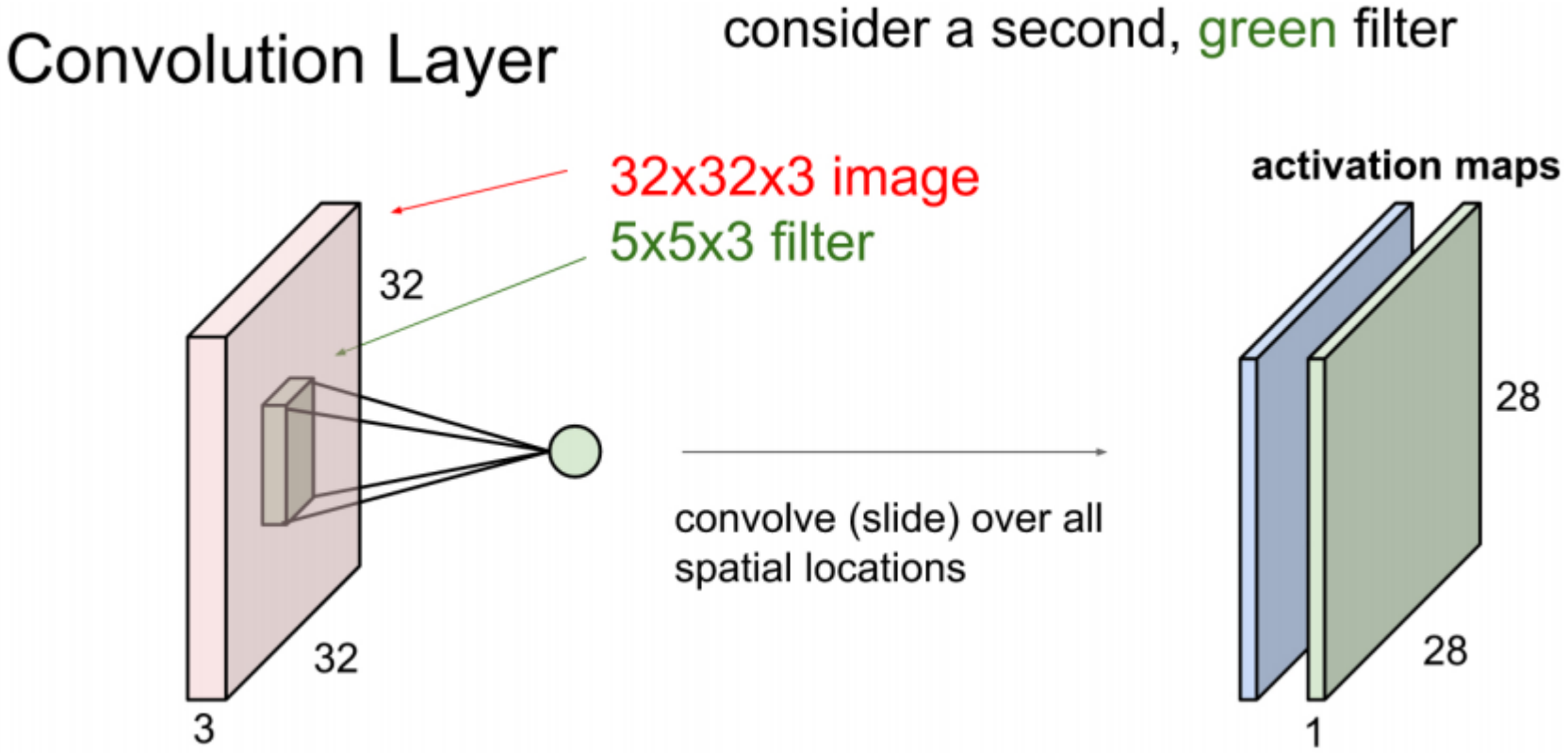


그림 7-5 합성곱 연산의 편향 : 필터를 적용한 원소에 고정값(편향)을 더한다.

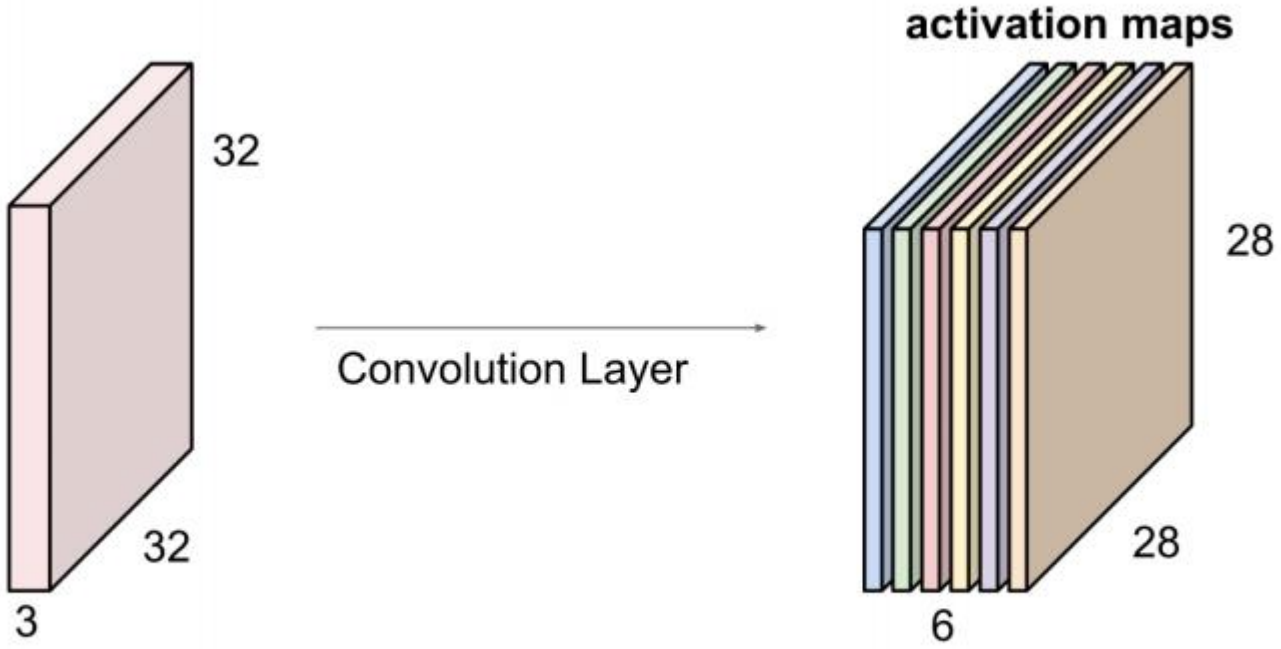


CNN(Convolution Neural Network)



CNN(Convolution Neural Network)

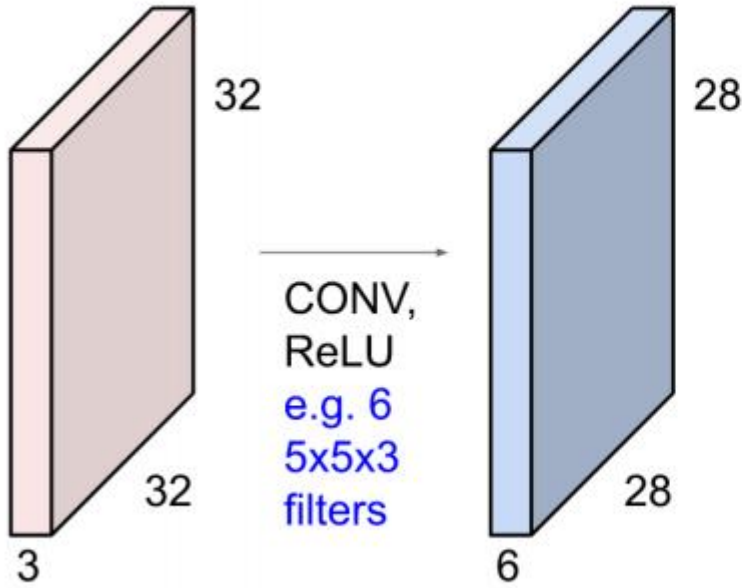
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



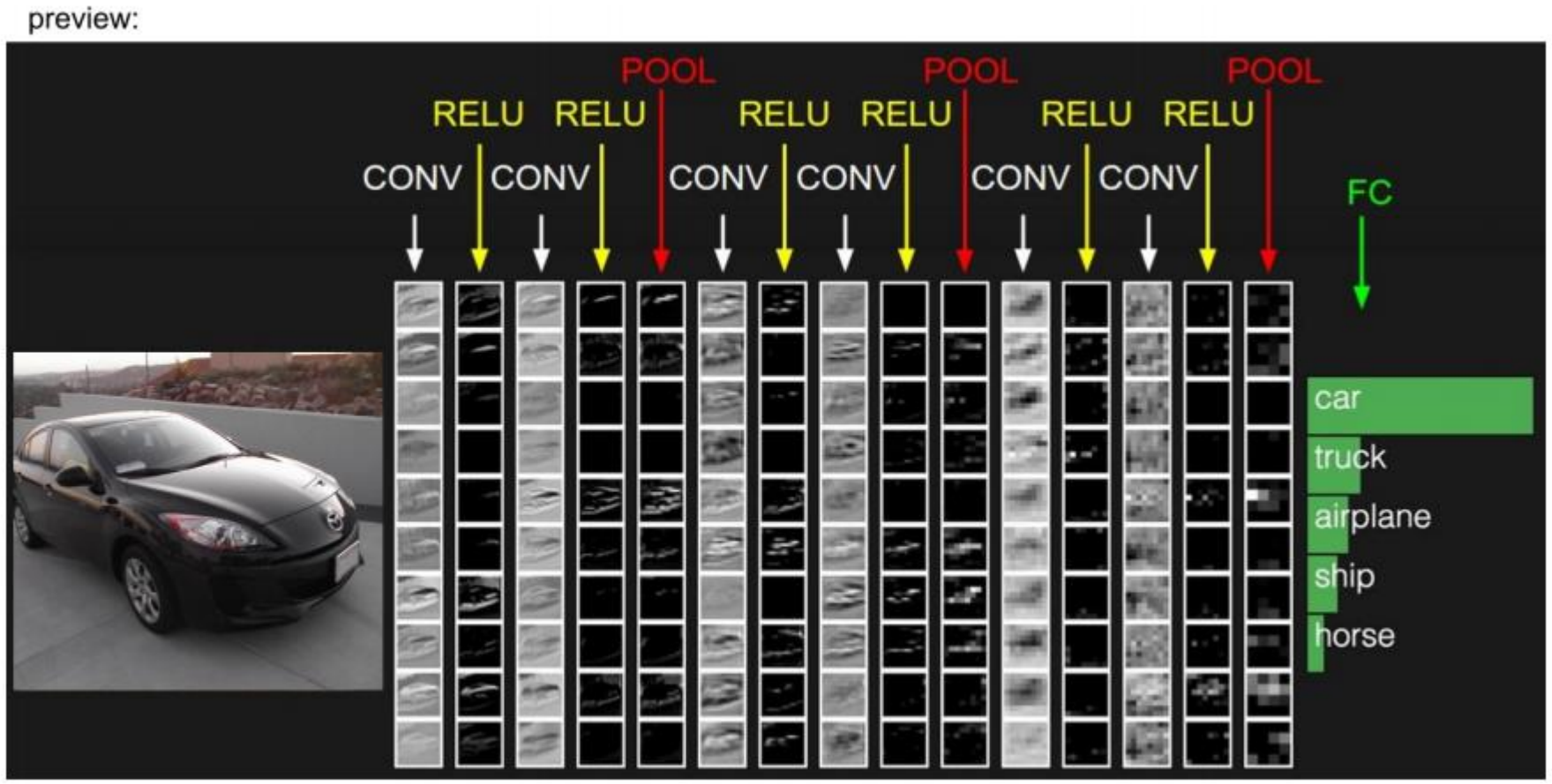
We stack these up to get a "new image" of size 28x28x6!

CNN(Convolution Neural Network)

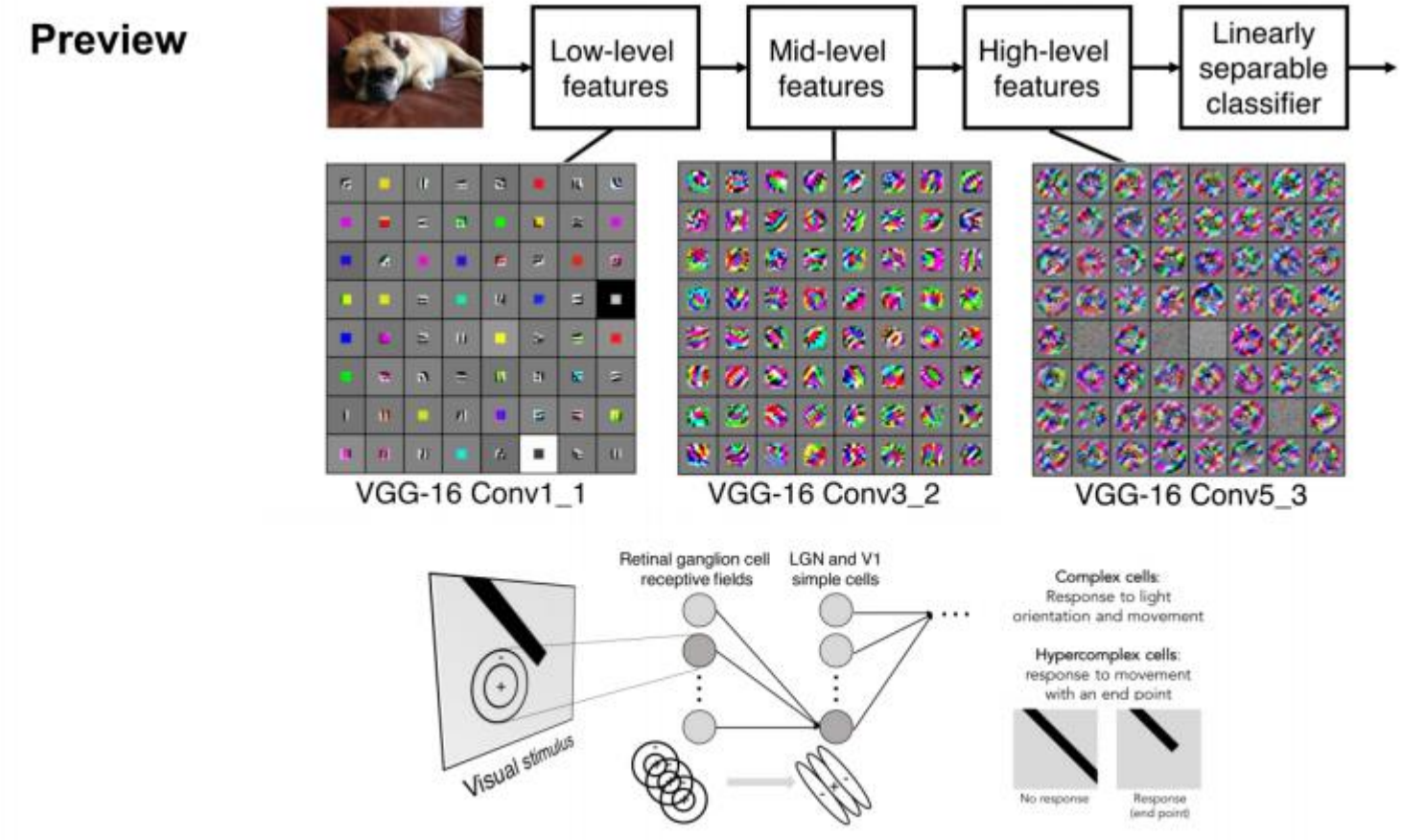
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



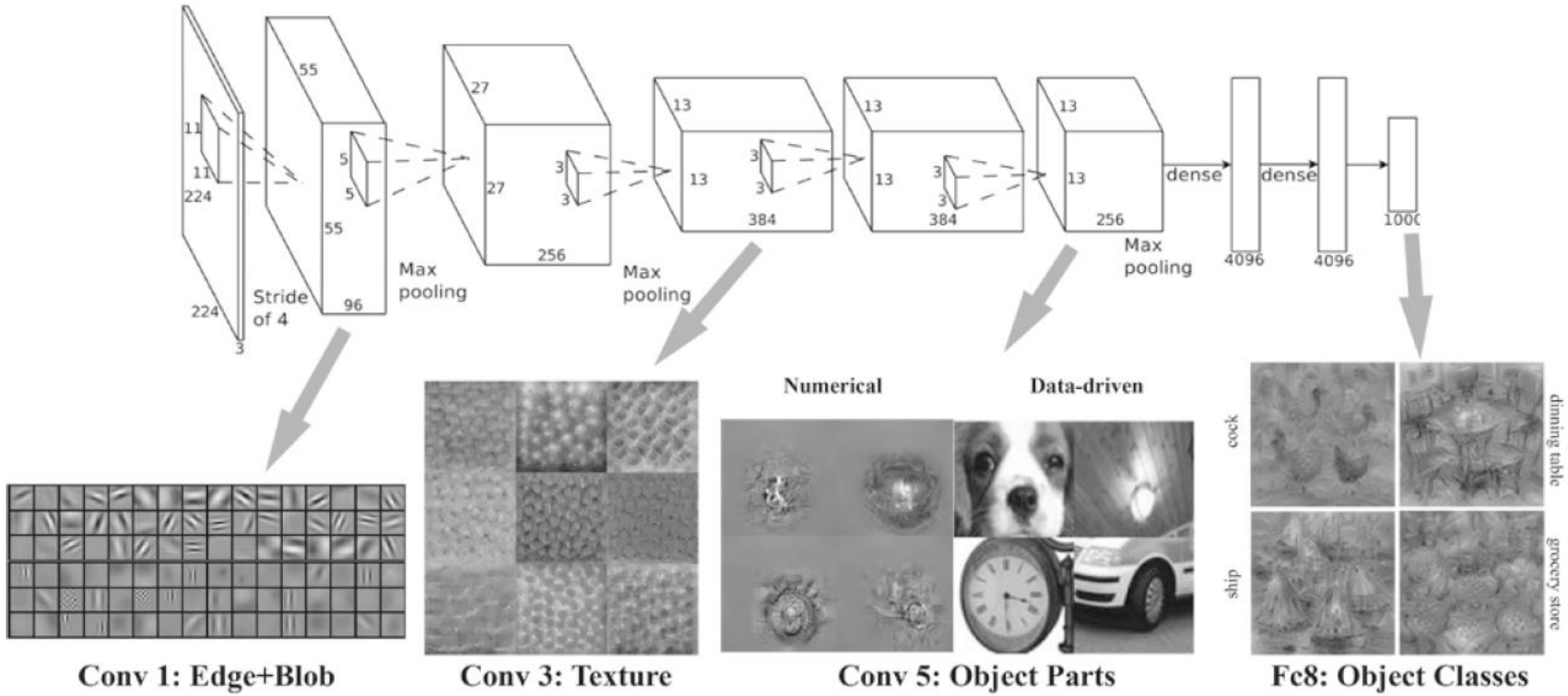
CNN(Convolution Neural Network)



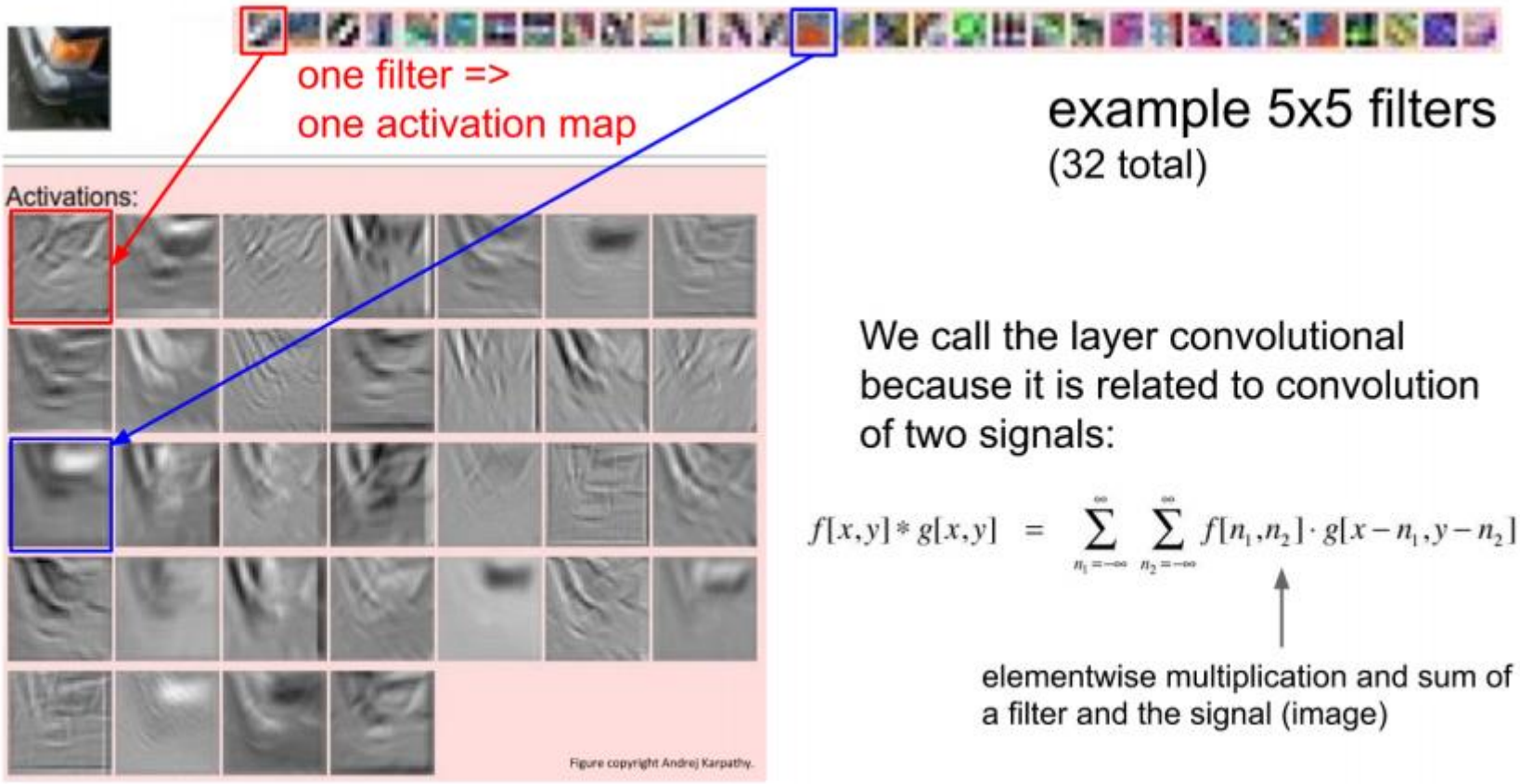
CNN(Convolution Neural Network)



CNN(Convolution Neural Network)



CNN(Convolution Neural Network)



example 5x5 filters
(32 total)

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

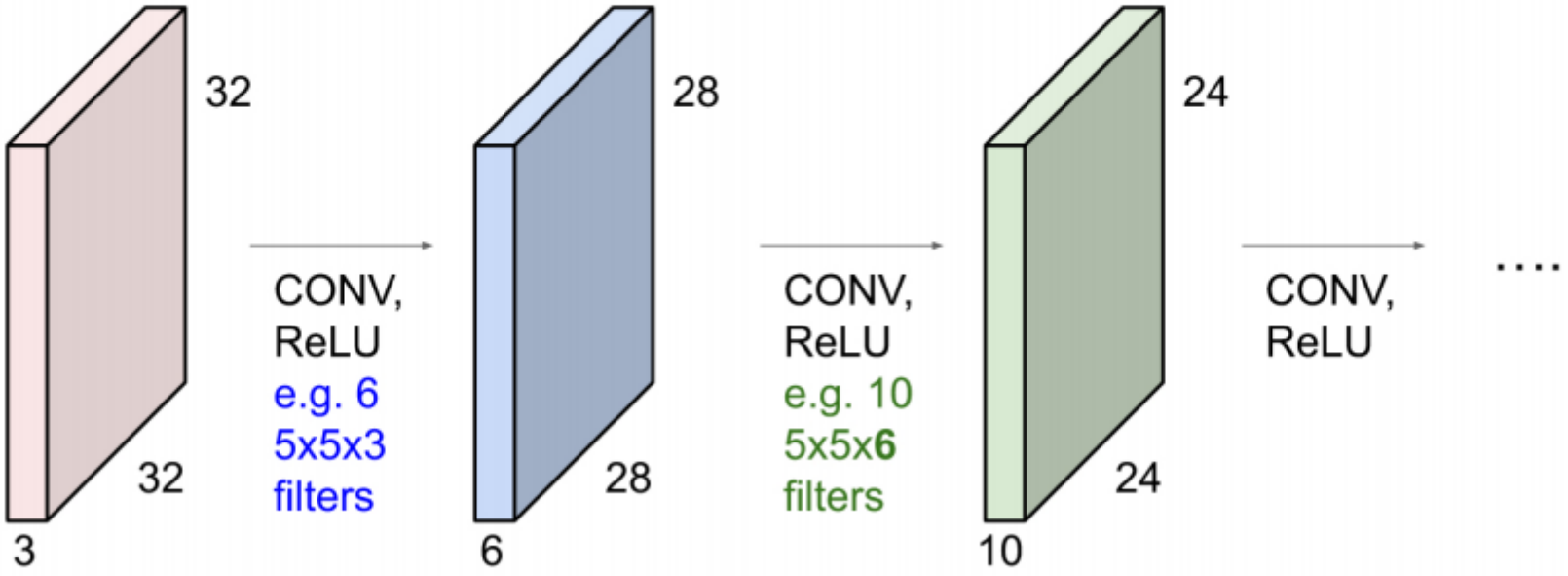
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

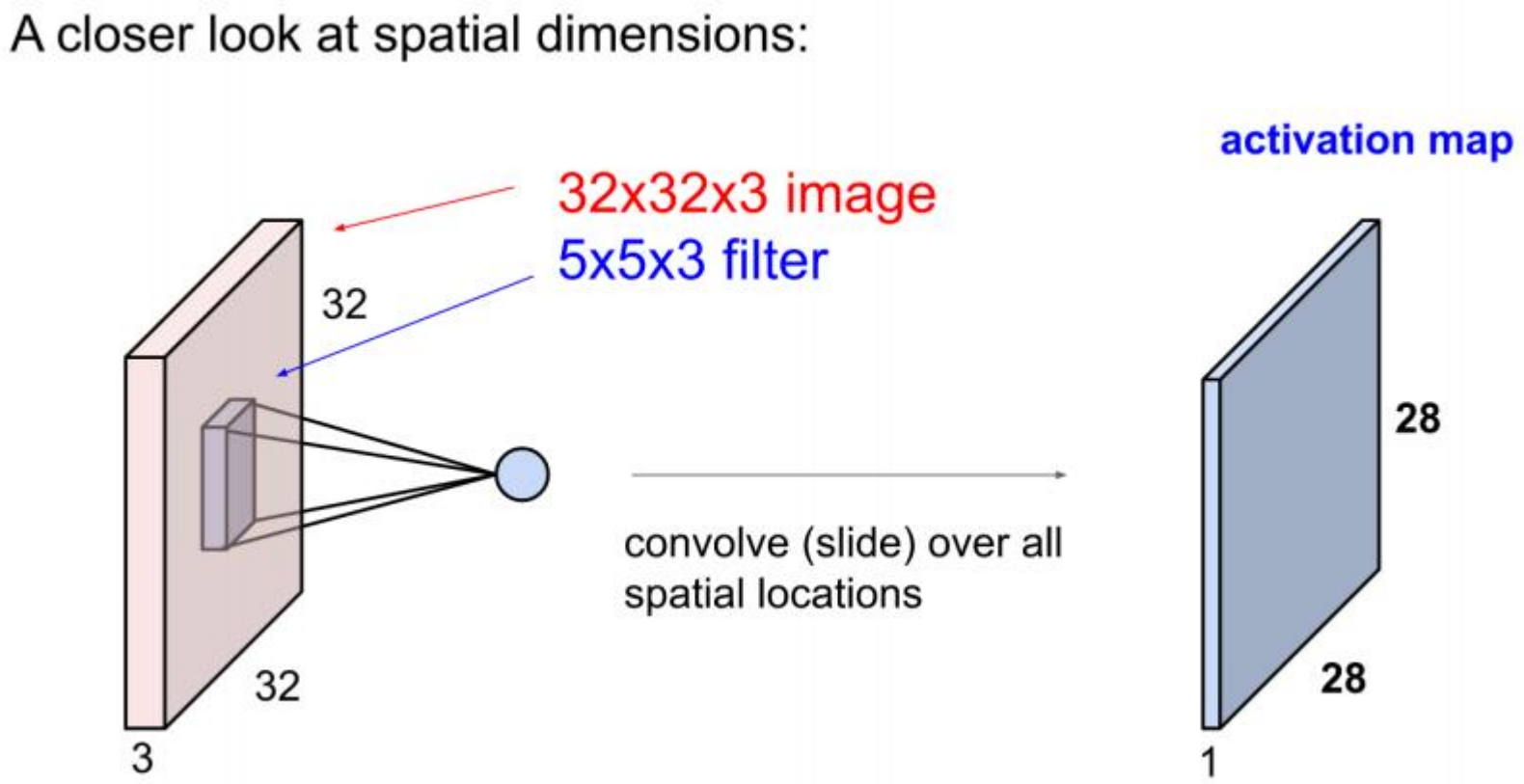
CNN(Convolution Neural Network)

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

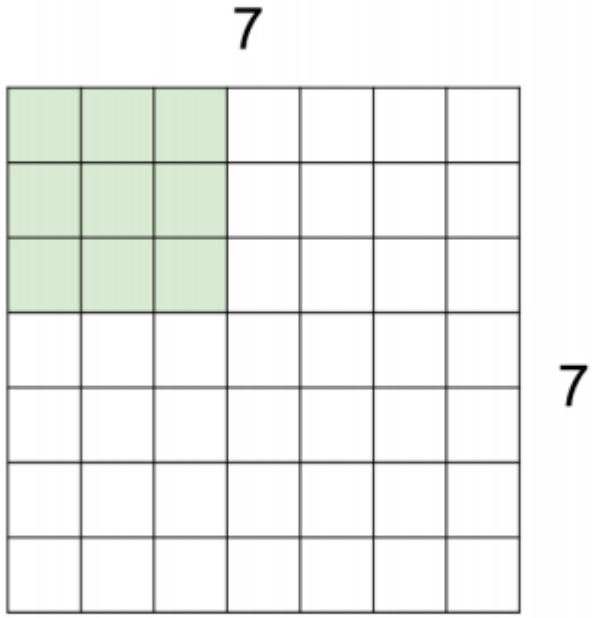


CNN(Convolution Neural Network)



CNN(Convolution Neural Network)

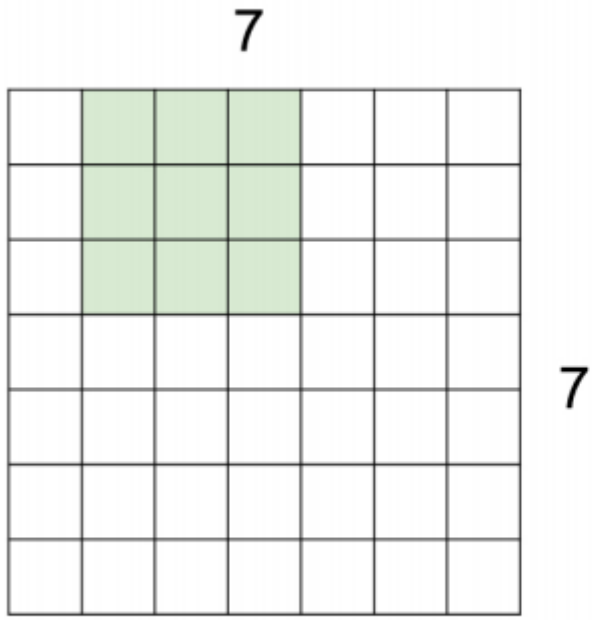
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN(Convolution Neural Network)

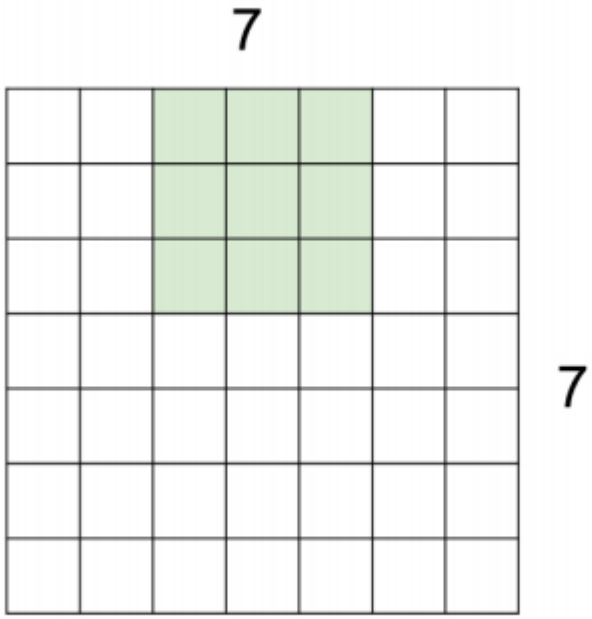
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN(Convolution Neural Network)

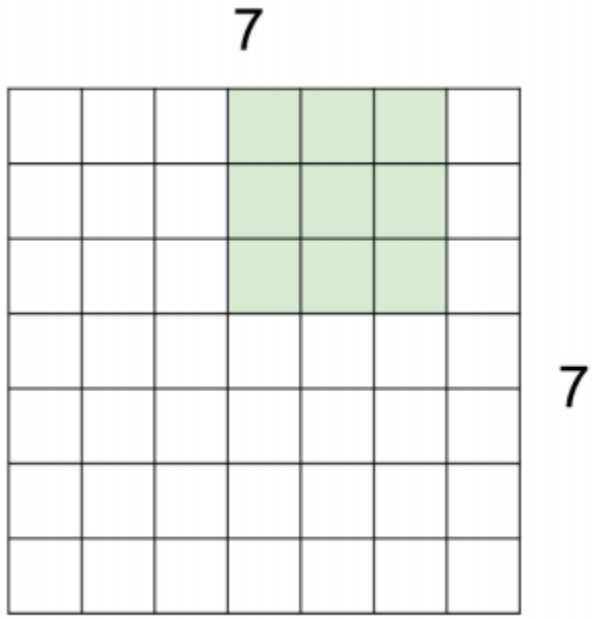
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN(Convolution Neural Network)

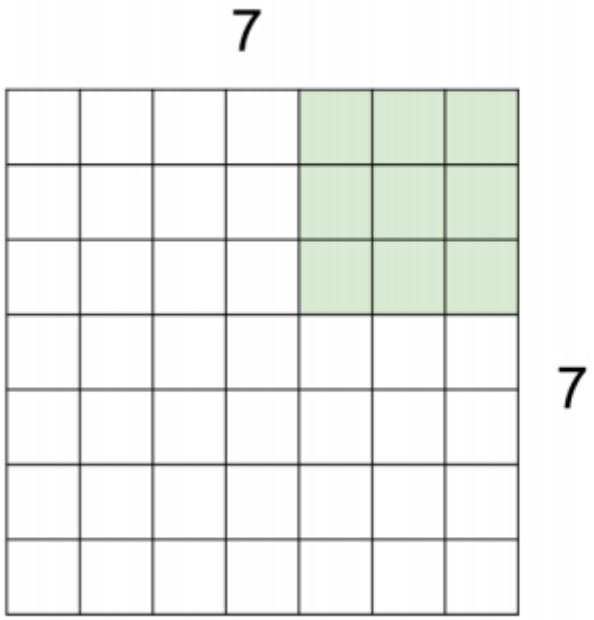
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN(Convolution Neural Network)

A closer look at spatial dimensions:

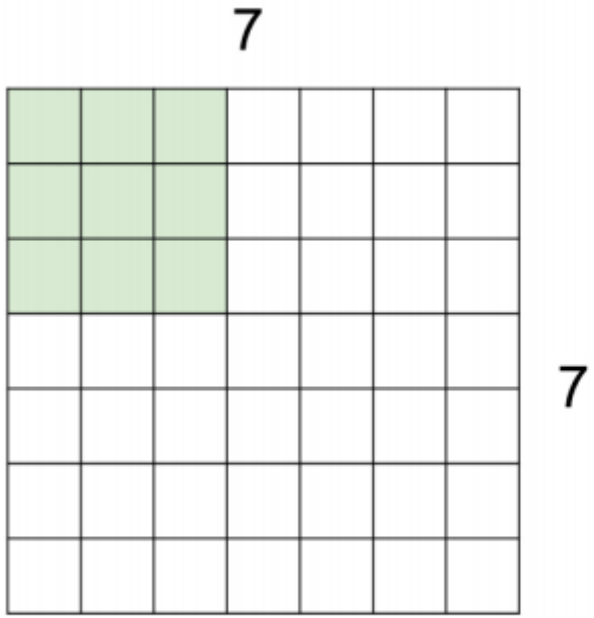


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

CNN(Convolution Neural Network)

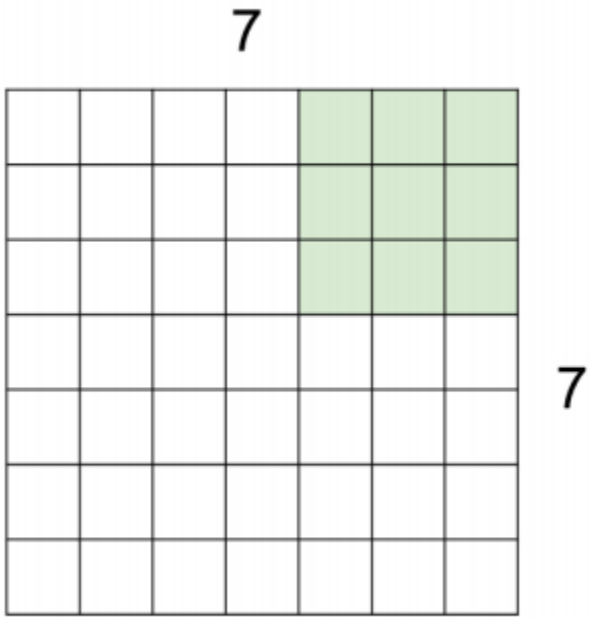
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

CNN(Convolution Neural Network)

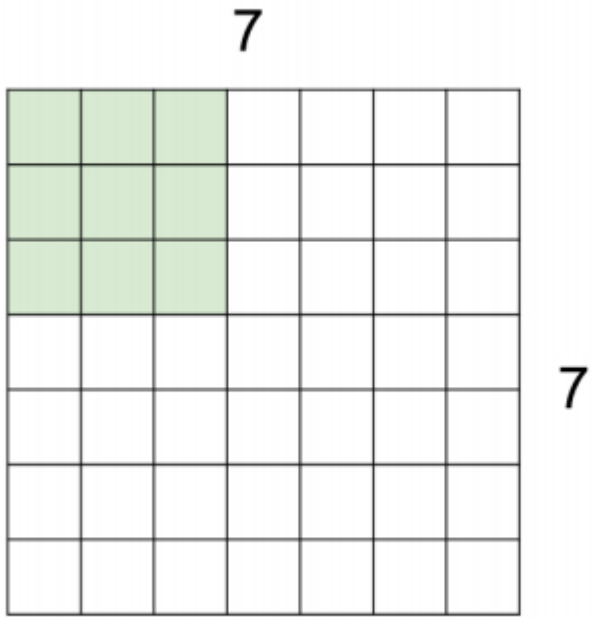
A closer look at spatial dimensions:



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

CNN(Convolution Neural Network)

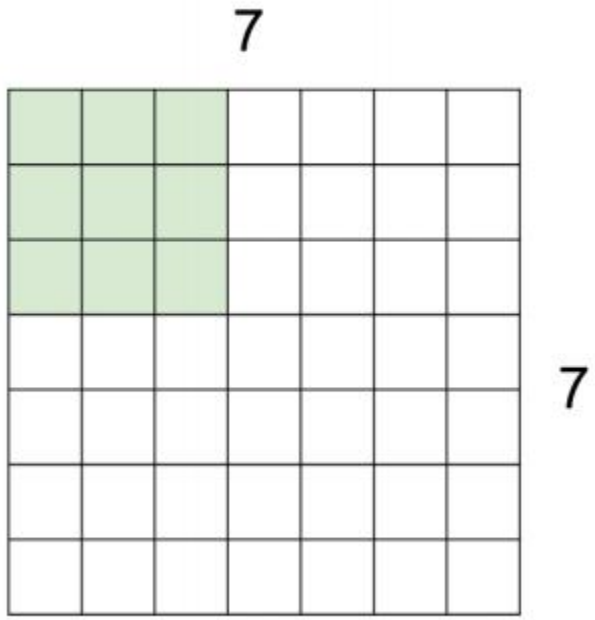
A closer look at spatial dimensions:



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

CNN(Convolution Neural Network)

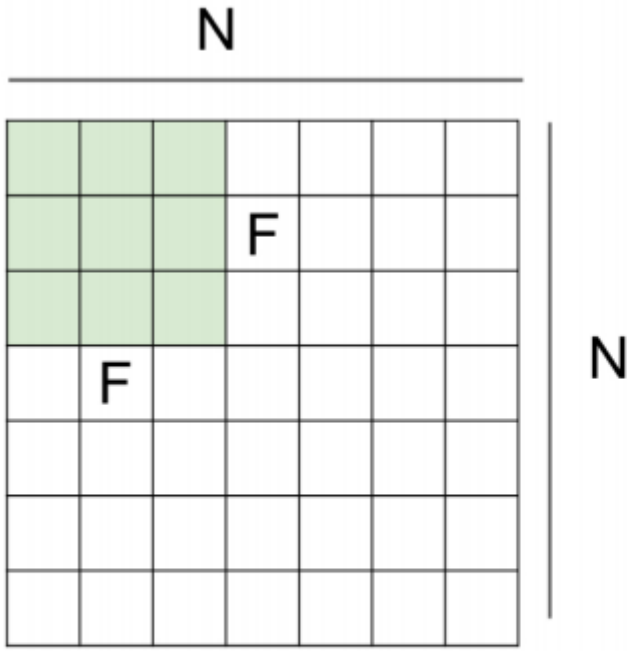
A closer look at spatial dimensions:



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

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

CNN(Convolution Neural Network)



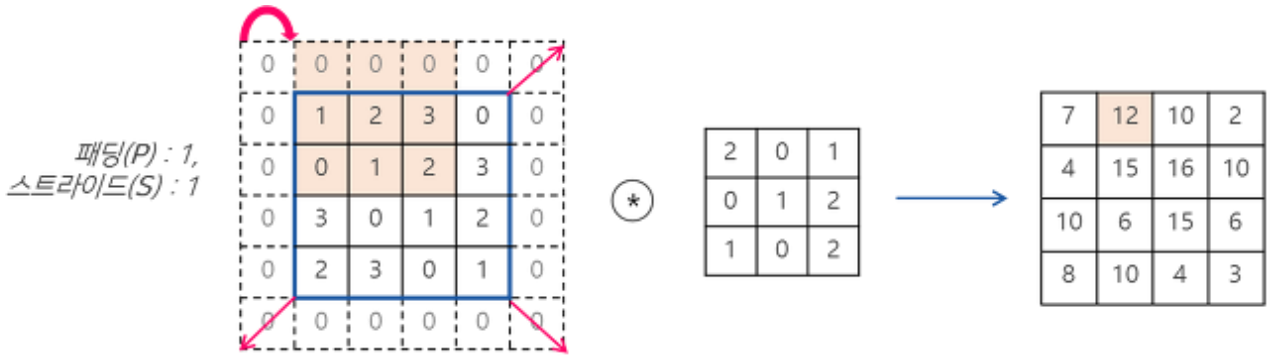
Output size:
(N - F) / stride + 1

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 :\

CNN(Convolution Neural Network)

$$(OH, OW) = \left(\frac{H + 2P - FH}{S} + 1\ , \frac{W + 2P - FW}{S} + 1\right)$$

- $(H,\ W)$: 입력크기
- $(FH,\ FW)$: 필터크기
- $(OH,\ OW)$: 출력크기
- P : 패딩
- S : 스트라이드

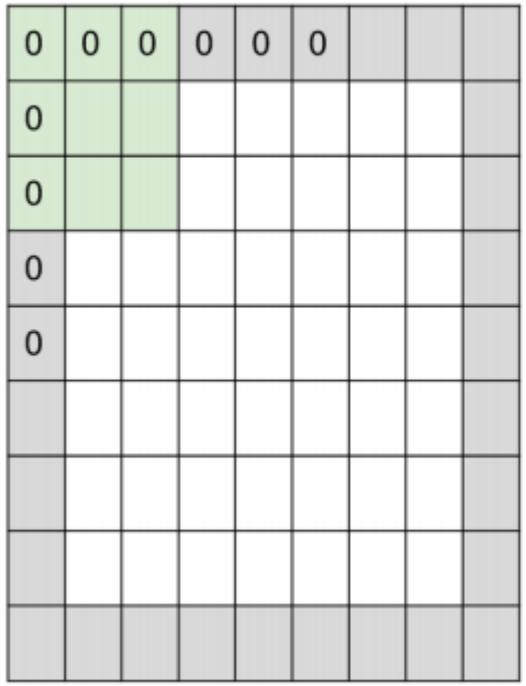


$$(OH, OW) = \left(\frac{4 + 2 * 1 - 3}{1} + 1\ , \frac{4 + 2 * 1 - 3}{1} + 1\right) = (\mathbf{4}, \mathbf{4})$$

HCI Lab
Inha University
Human-Computer Interaction Lab

CNN(Convolution Neural Network)

In practice: Common to zero pad the border

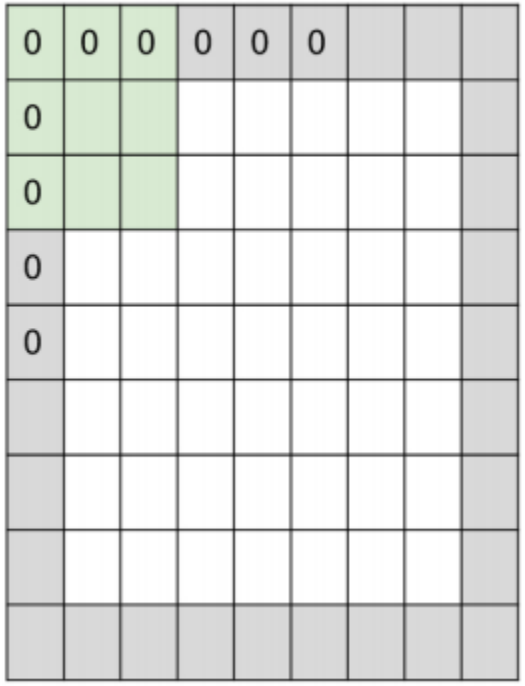


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

CNN(Convolution Neural Network)

In practice: Common to zero pad the border



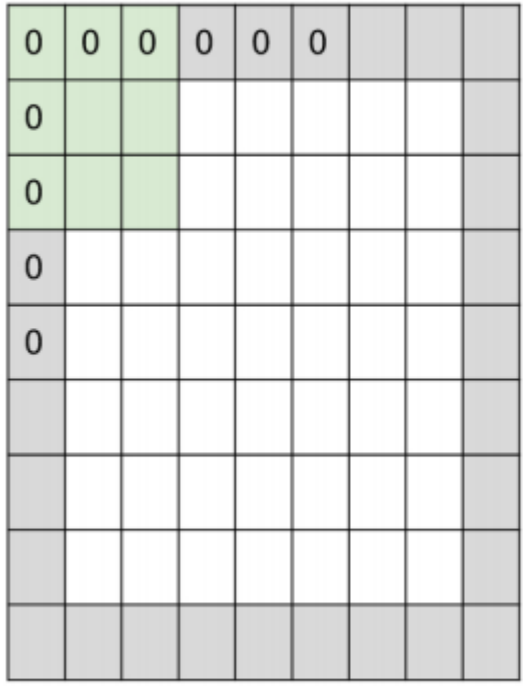
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

(recall:)
(N + 2P - F) / stride + 1

CNN(Convolution Neural Network)

In practice: Common to zero pad the border

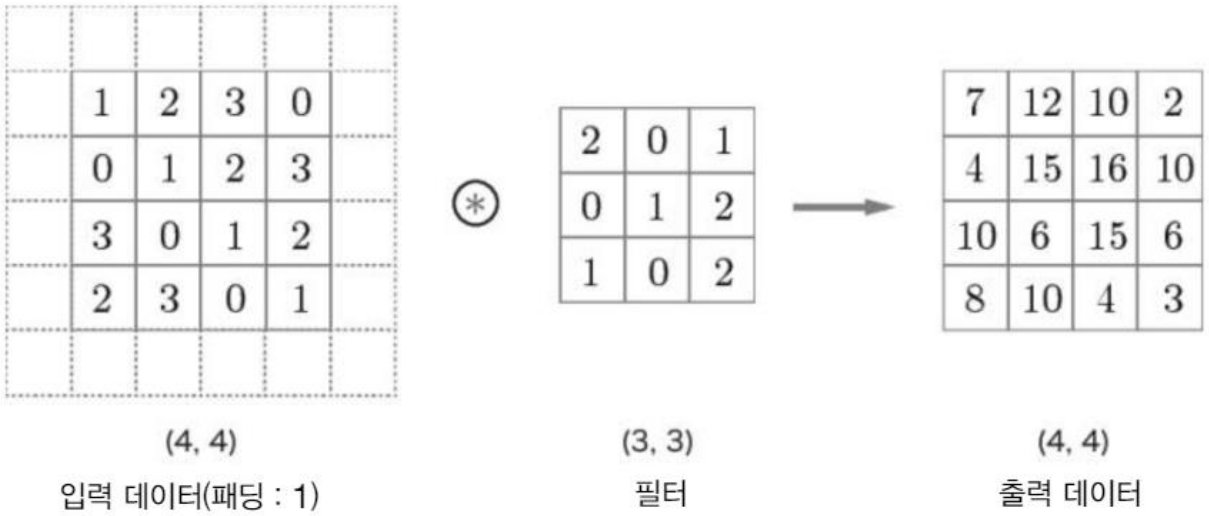


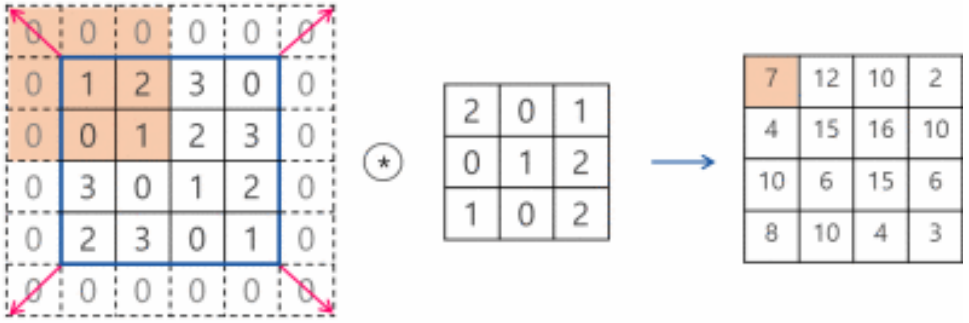
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

CNN(Convolution Neural Network)

그림 7-6 합성곱 연산의 패딩 처리 : 입력 데이터 주위에 0을 채운다(패딩은 점선으로 표시했으며 그 안의 값 '0'은 생략했다).



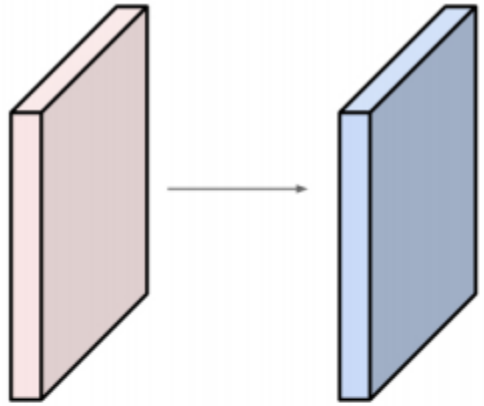


CNN(Convolution Neural Network)

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

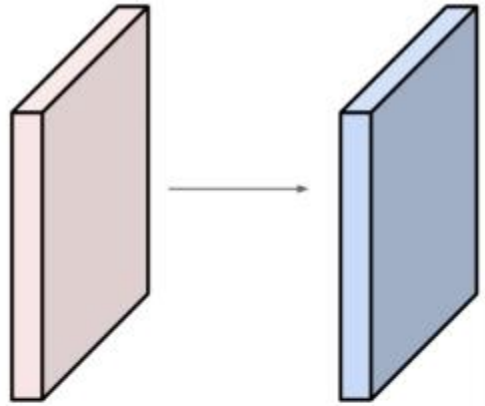


CNN(Convolution Neural Network)

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

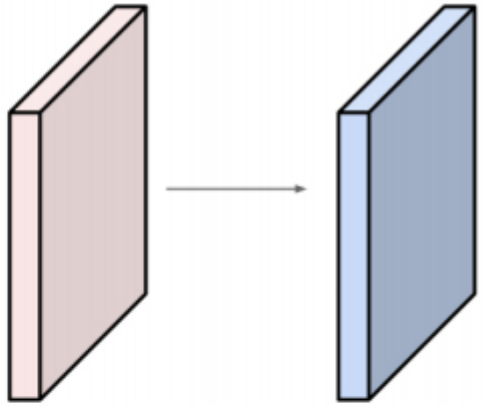
Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



CNN(Convolution Neural Network)

Examples time:

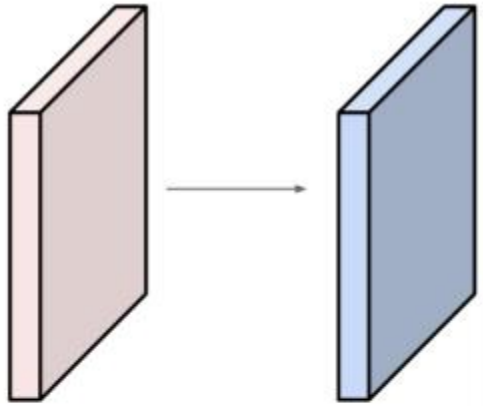
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

CNN(Convolution Neural Network)

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

CNN(Convolution Neural Network)

Convolution layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of W_2 x H_2 x K
where:

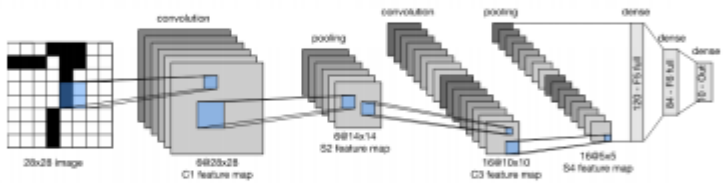
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

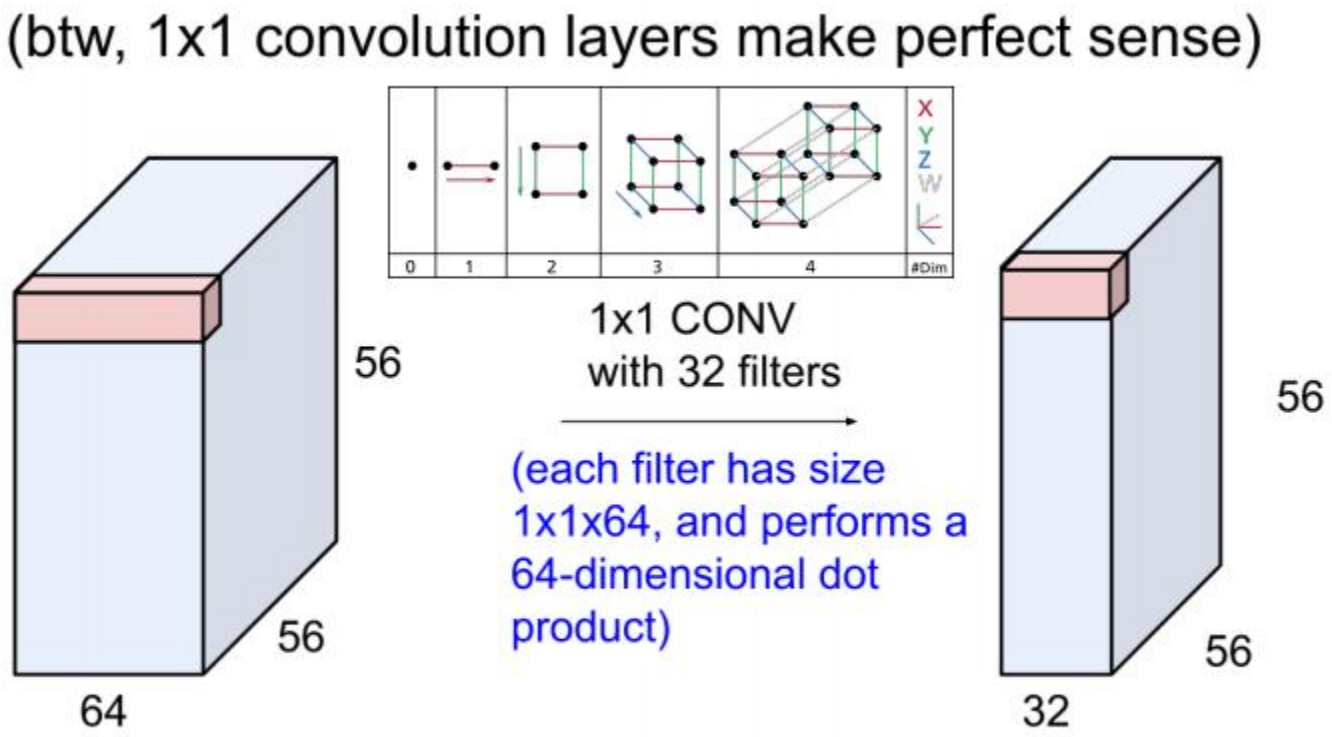
Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 1, S = 1, P = 0 ?



CNN(Convolution Neural Network)



CNN(Convolution Neural Network)

Example: CONV layer in PyTorch

Conv2d

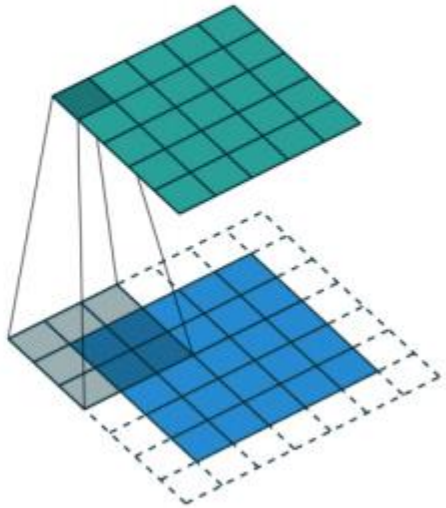
CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

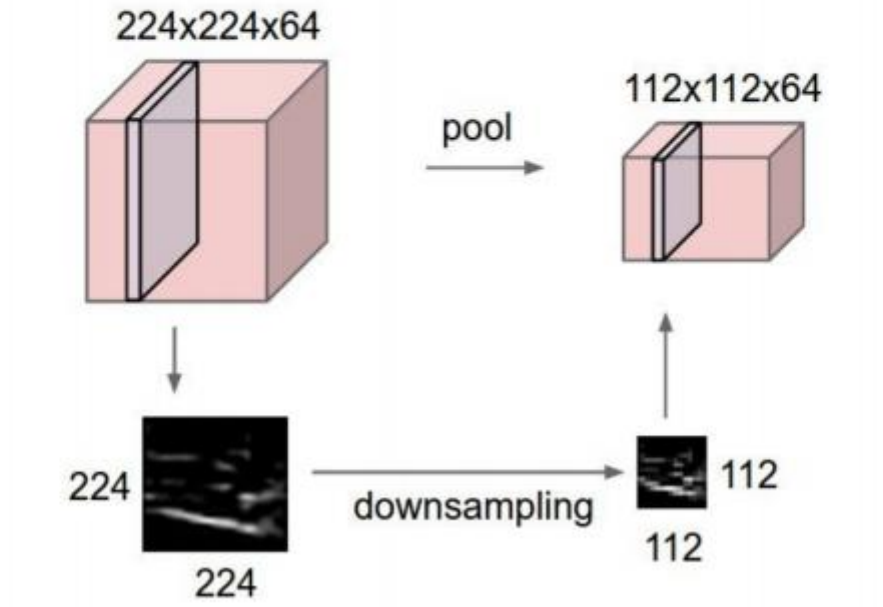
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.



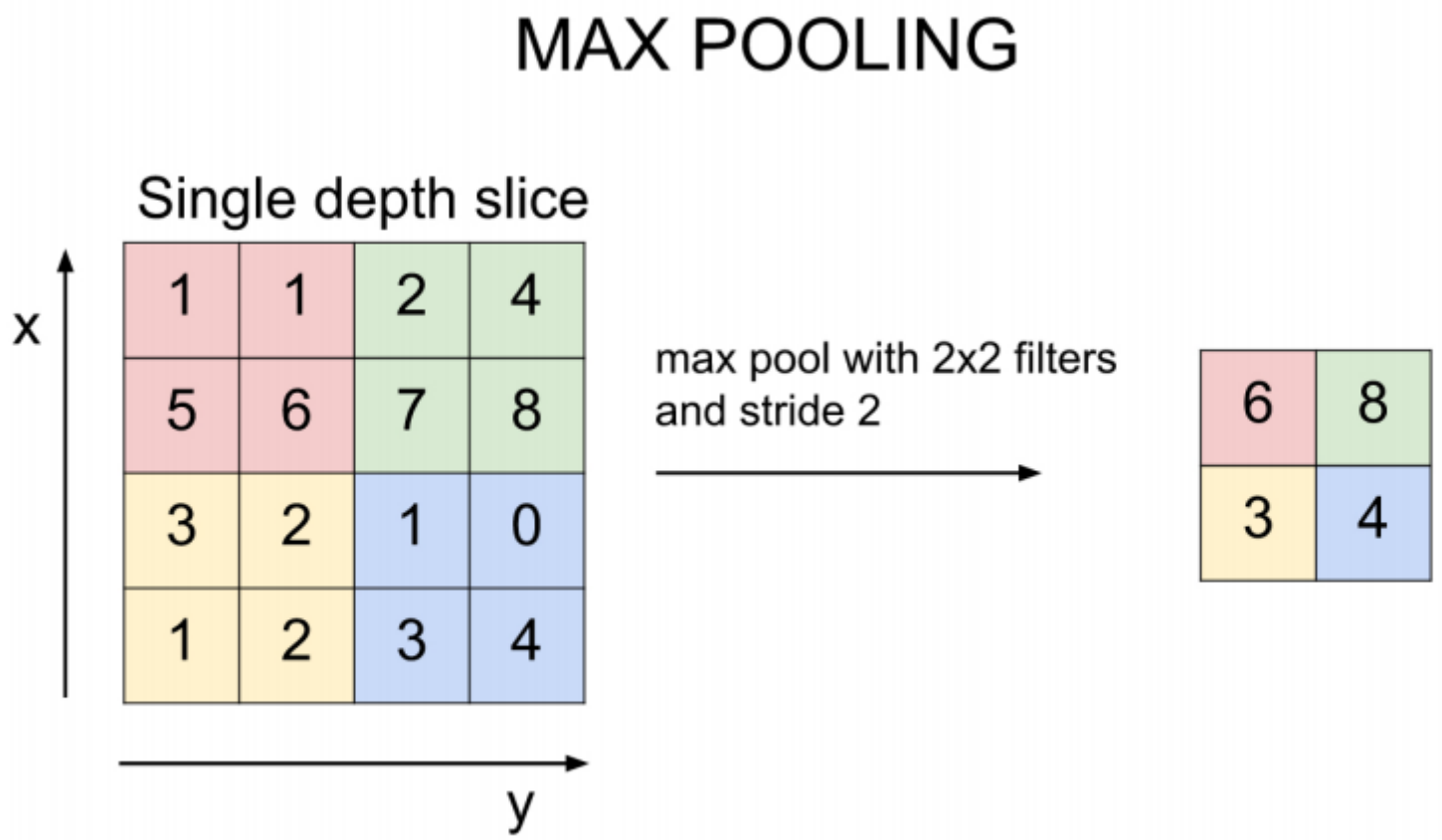
CNN(Convolution Neural Network)

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:
- makes the representation become approximately invariant to small translations of the input



CNN(Convolution Neural Network)



CNN(Convolution Neural Network)

Pooling layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

This will produce an output of W_2 x H_2 x C where:

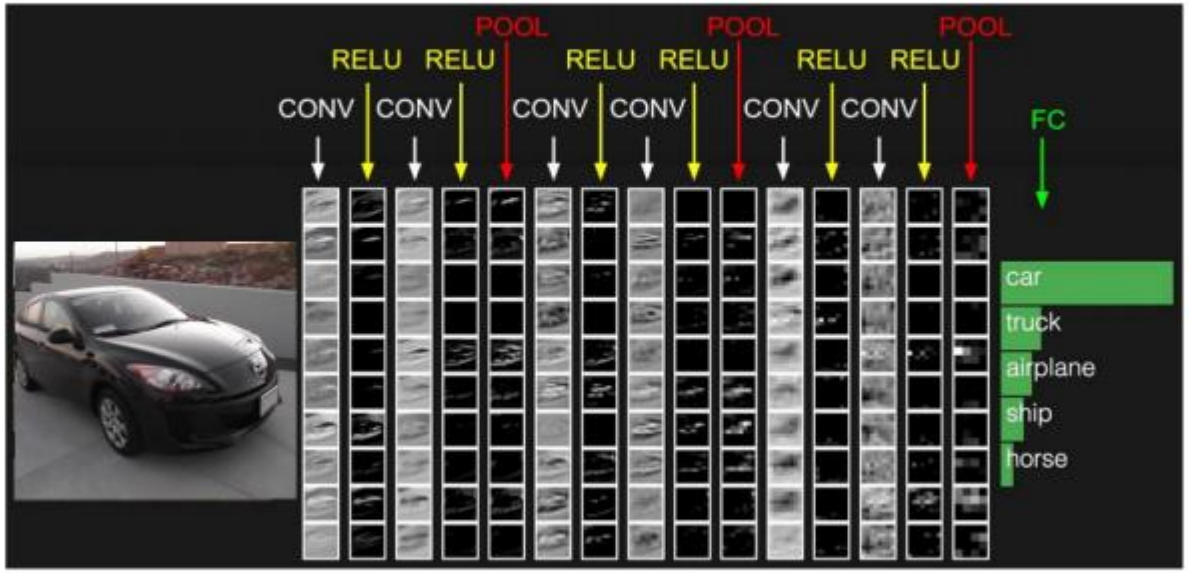
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0

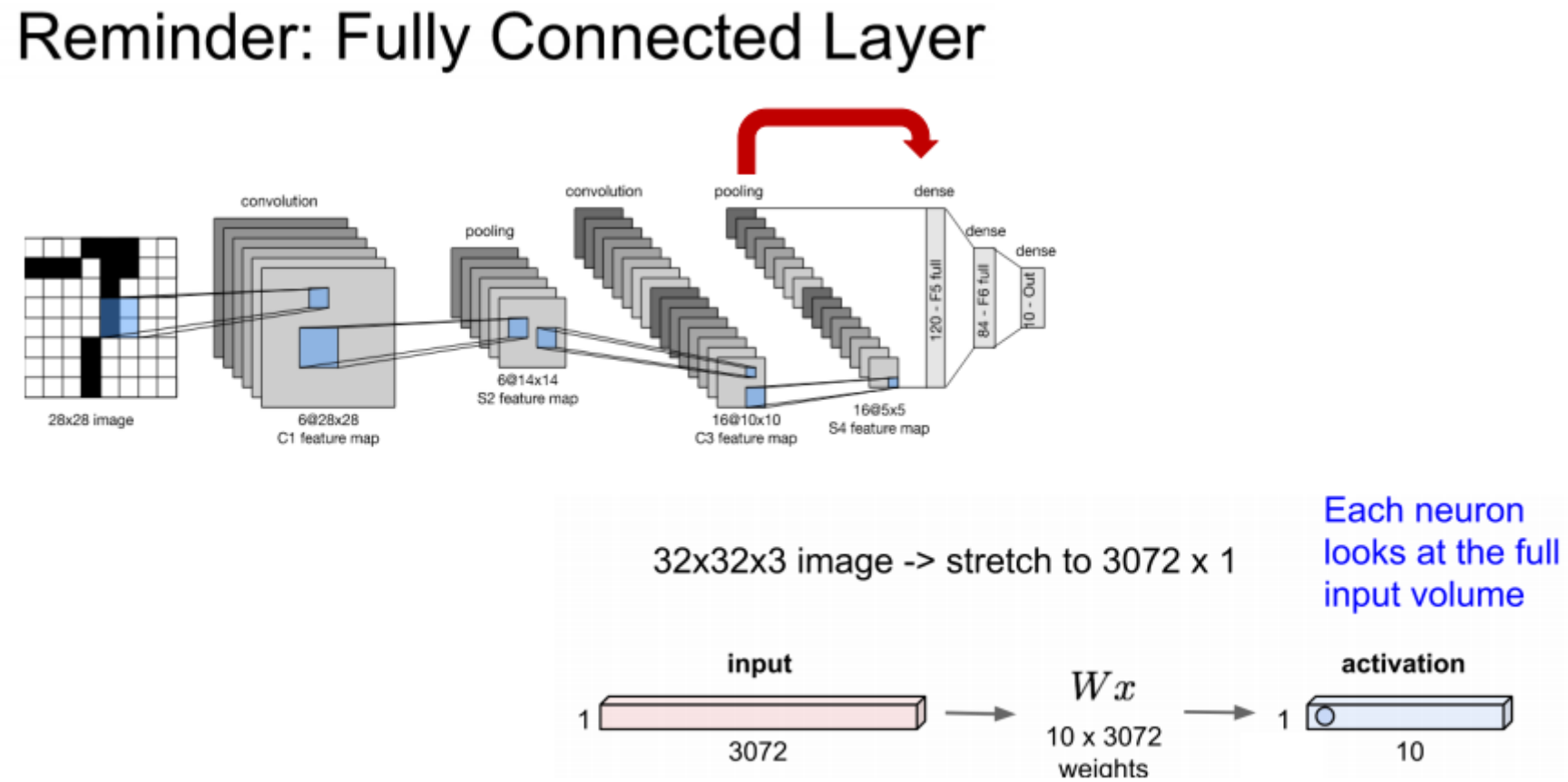
CNN(Convolution Neural Network)

Fully Connected Layer (FC layer)

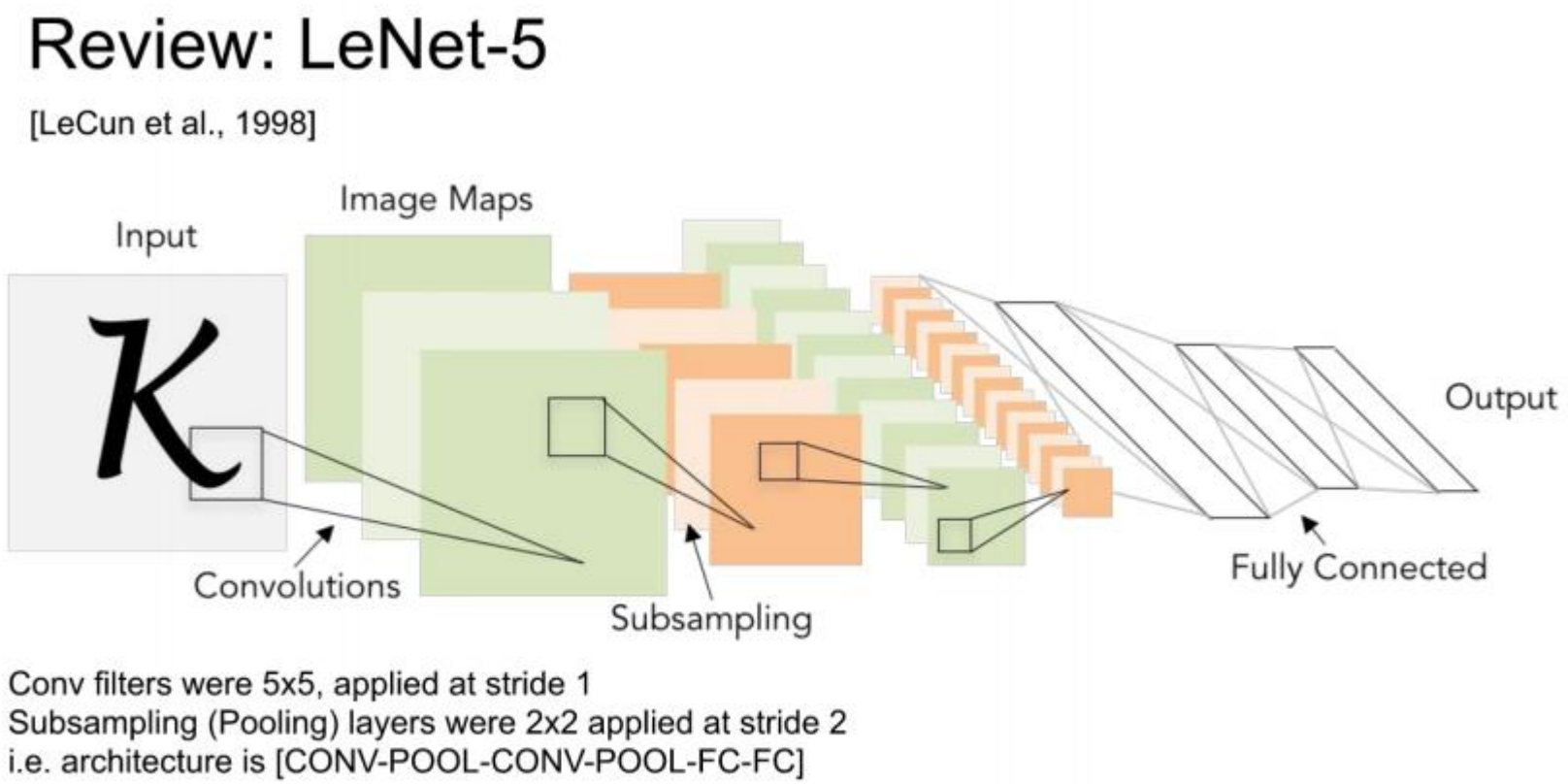
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



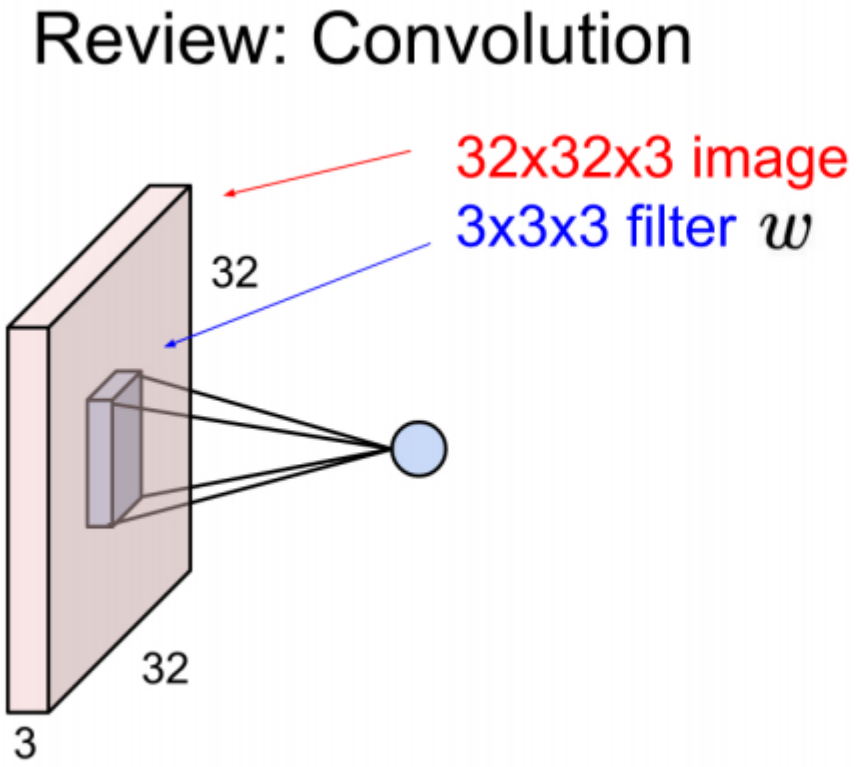
CNN(Convolution Neural Network)

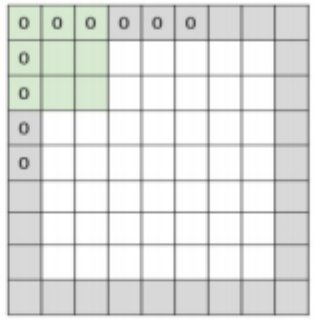


CNN(Convolution Neural Network)

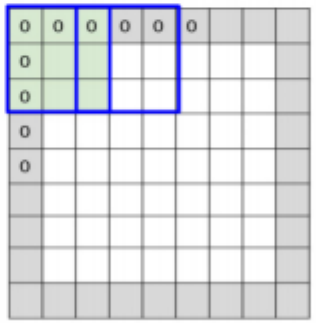


CNN(Convolution Neural Network)



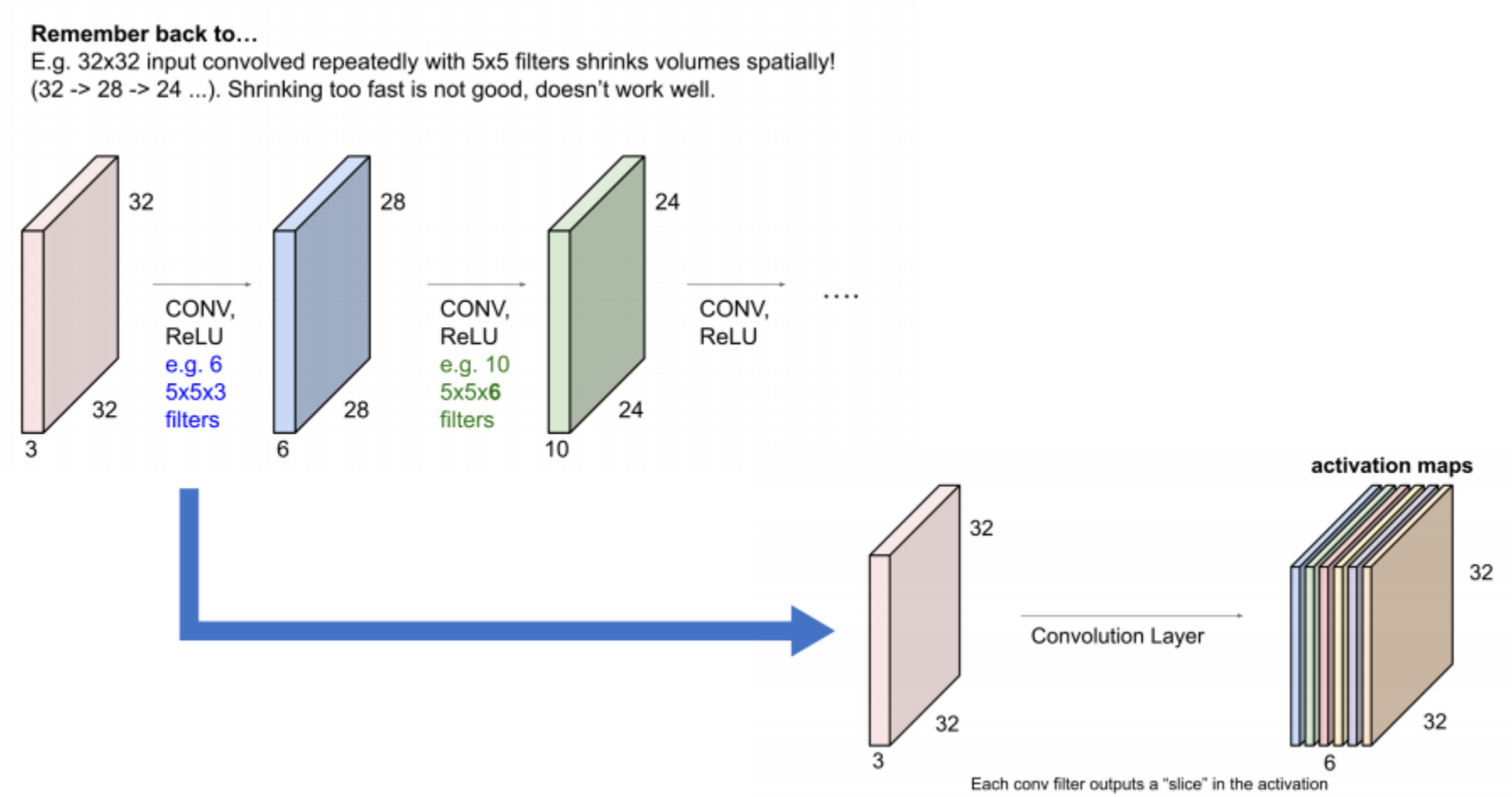


Padding:
Preserve
input spatial
dimensions in
output activations

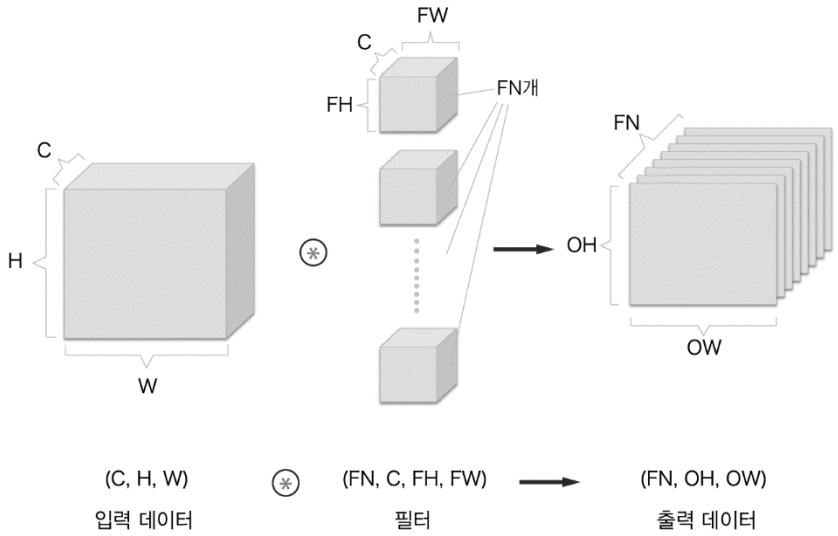


Stride:
Downsample
output activations

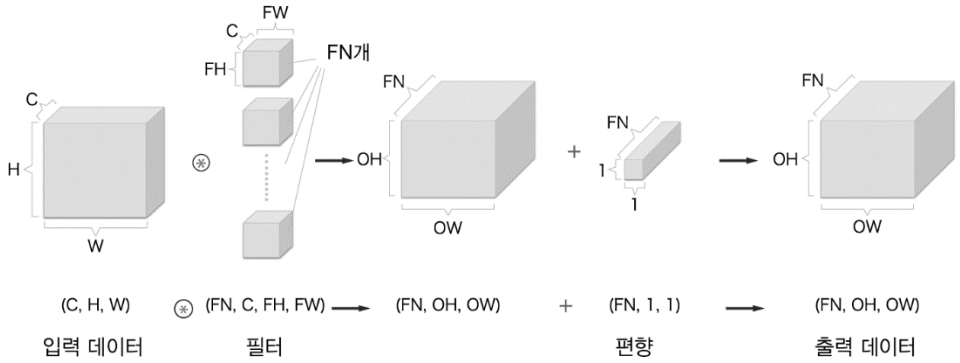
CNN(Convolution Neural Network)



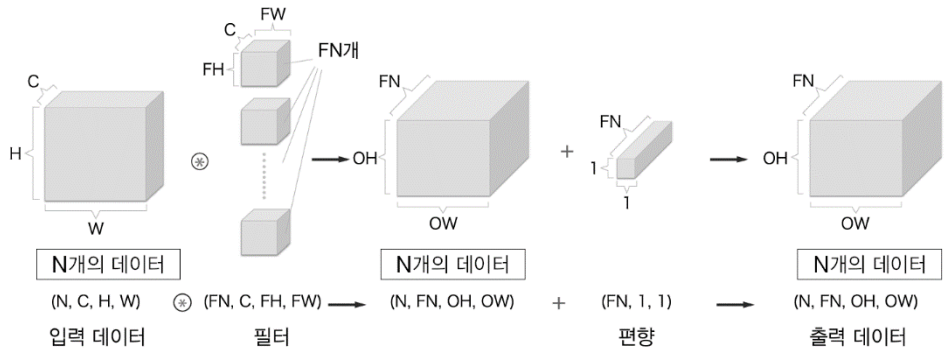
CNN(Convolution Neural Network)



◀여러필터를 사용한 합성곱 연산의 예



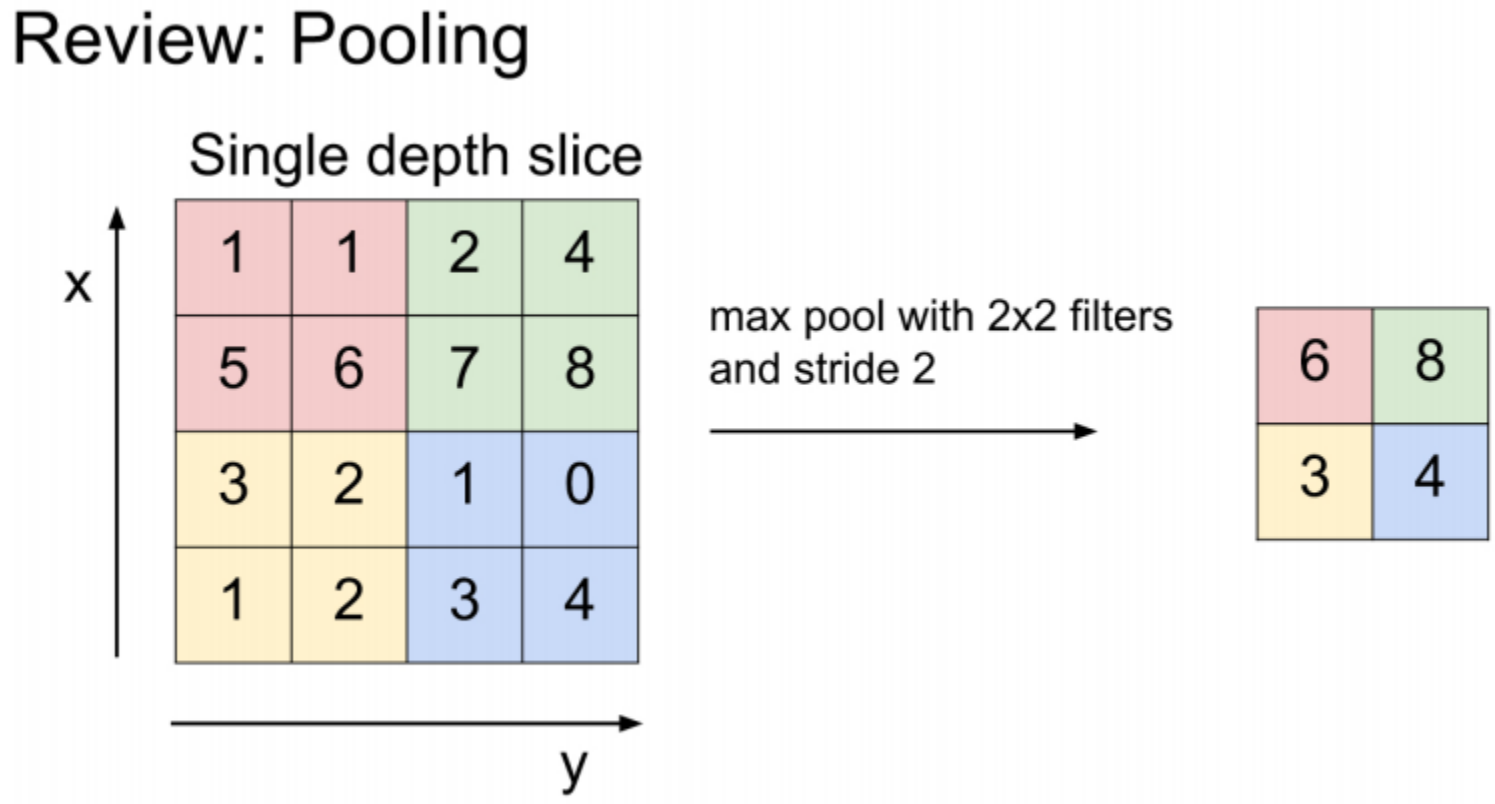
▲ 합성곱 연산의 처리 흐름(편향 추가)



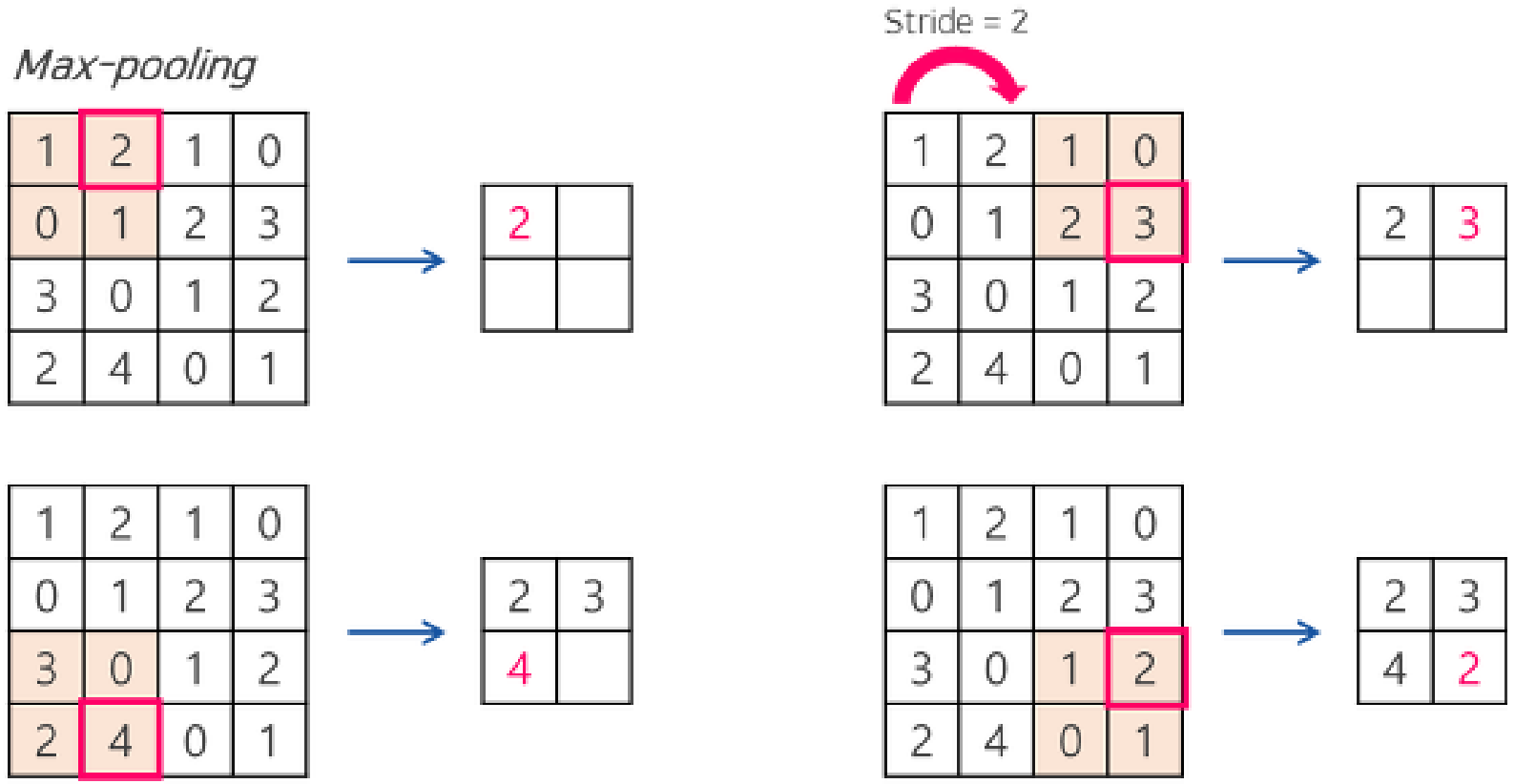
▲ 합성곱 연산의 처리 흐름(배치 처리)

HCI Lab
Inha University
Human-Computer Interaction Lab

CNN(Convolution Neural Network)

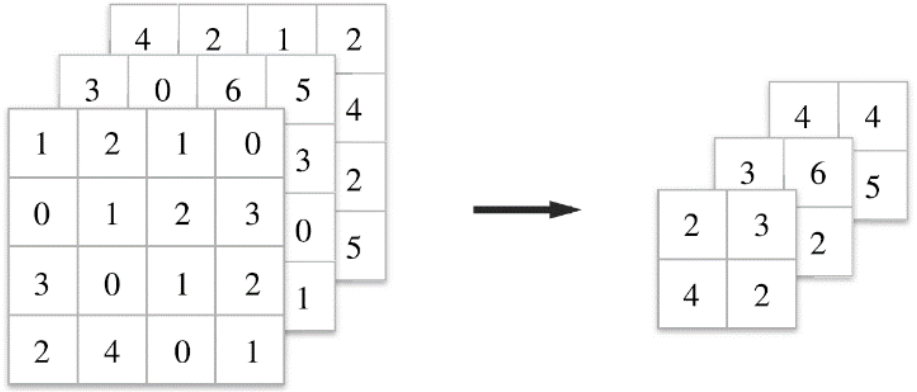


CNN(Convolution Neural Network)

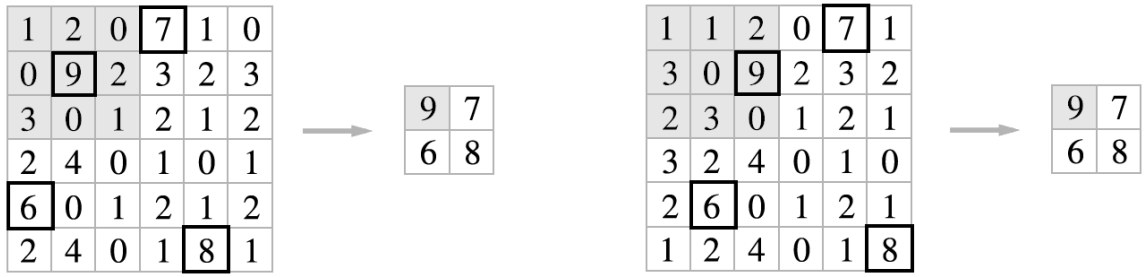


CNN(Convolution Neural Network)

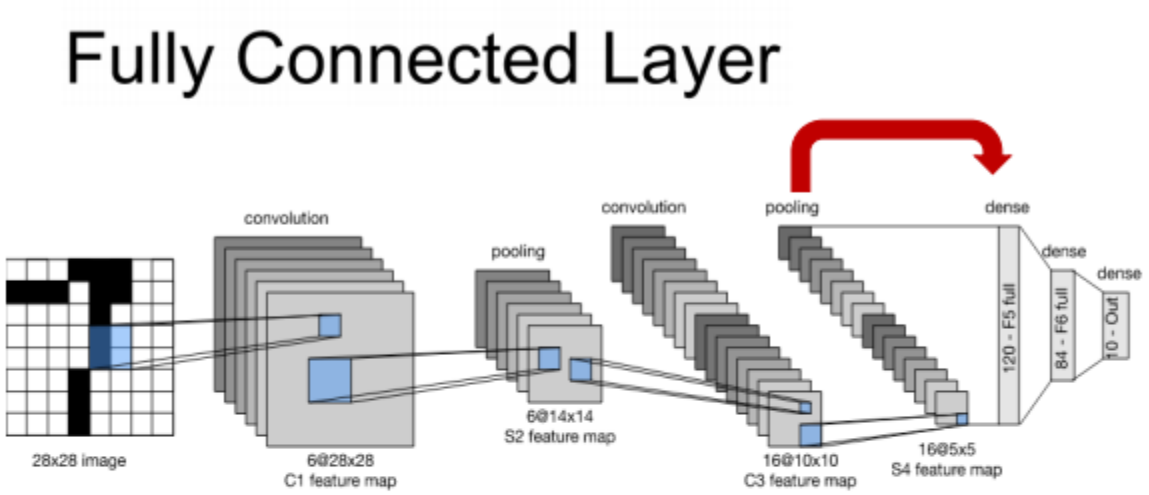
- 풀링 계층은 합성곱 계층과 달리 학습해야 할 매개변수가 없다.
- 풀링 연산은 입력 데이터의 채널 수 그대로 출력 데이터로 보낸다.



- 풀링 계층은 입력의 변화에 영향을 적게 받는다.

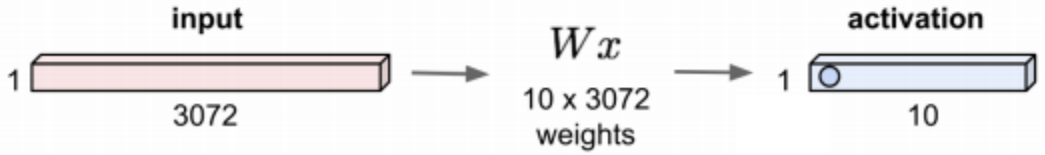


CNN(Convolution Neural Network)



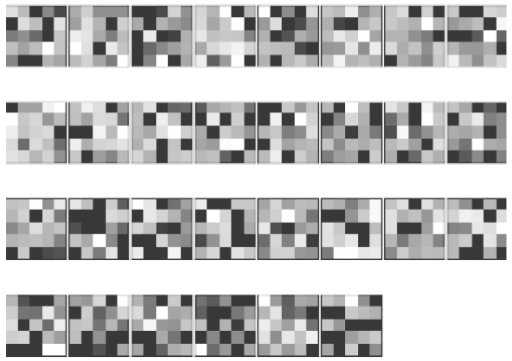
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

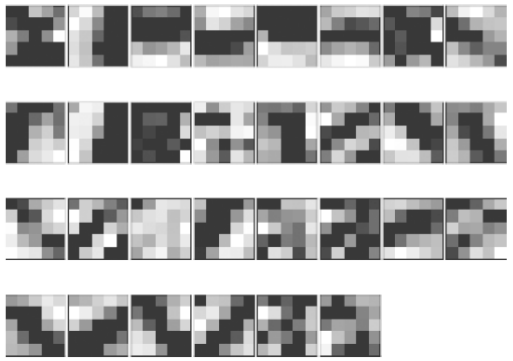


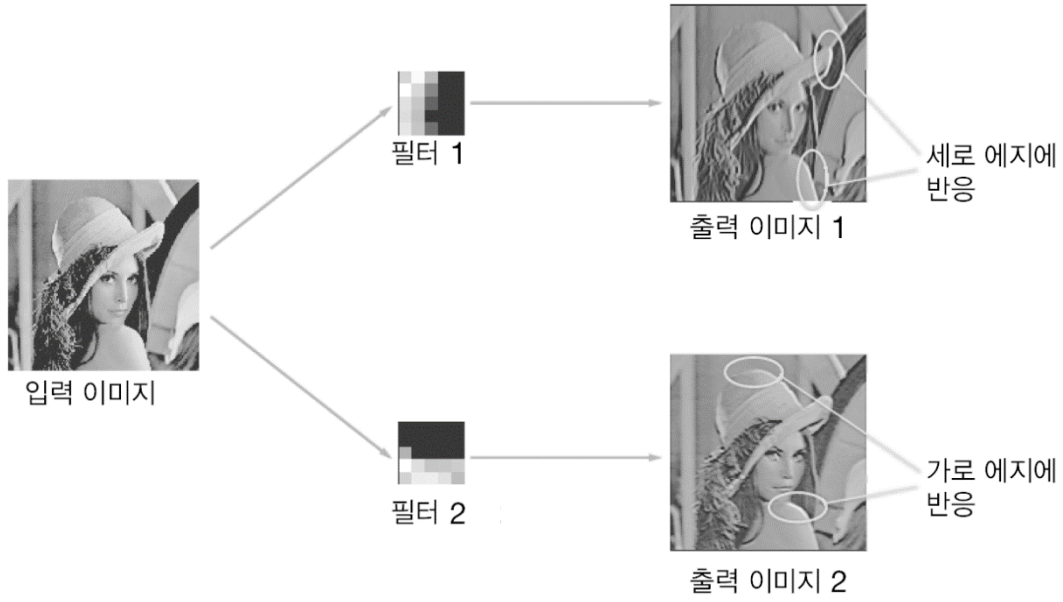
CNN(Convolution Neural Network)

학습 전



학습 후





3 Practice

CNN Tutorials

https://github.com/LEE-SEON-WOO/DL-Tutorials-Py_or_Tf
https://poloclub.github.io/cnn-explainer/

DL-Tutorials-Py_or_Tf

Star 0 LinkedIn

Among the top 10 ML repos on GitHub

- Learn Python DL-Tutorials-Py_or_Tf with notebooks.
- Use data science libraries like NumPy and Pandas.
- Implement basic ML models in TensorFlow 2.0 + Keras or PyTorch.
- Learn best practices with clean code, simple math and visualizations.

Notebooks	Python	NumPy
Pandas	TensorFlow	PyTorch
Linear Regression	Logistic Regression	Multilayer Perceptrons
Data & Models	Utilities	Preprocessing
Convolutional Neural Networks	Embeddings	Recurrent Neural Networks

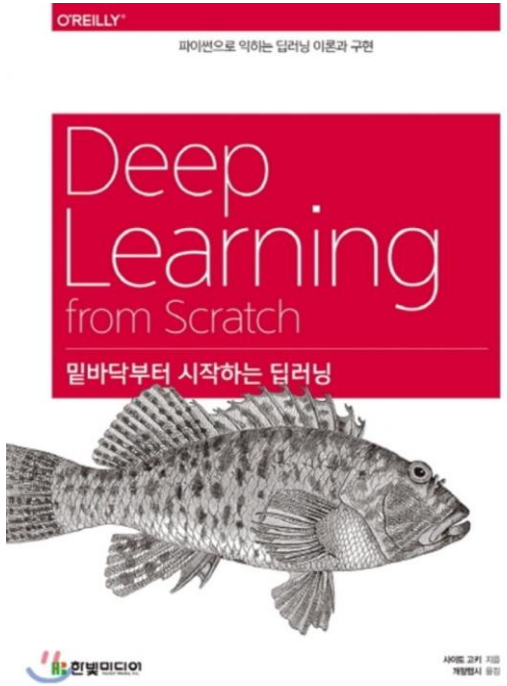
HCI Lab
Inha University
Human-Computer Interaction Lab

Book

주교재

밑바닥부터 시작하는 딥러닝 1, 사이토 고키 지음, 개앞맨시 옮김 .

(Deep Learning from Scratch의 번역서입니다 .)



부교재

Pytorch로 시작하는 딥러닝, 비슈누 수브라마니안 지음 김태완 옮김 .

(Deep Learning with PYTORCH 의 번역서입니다 .)