# Convolutional neural network

손경아

아주대학교

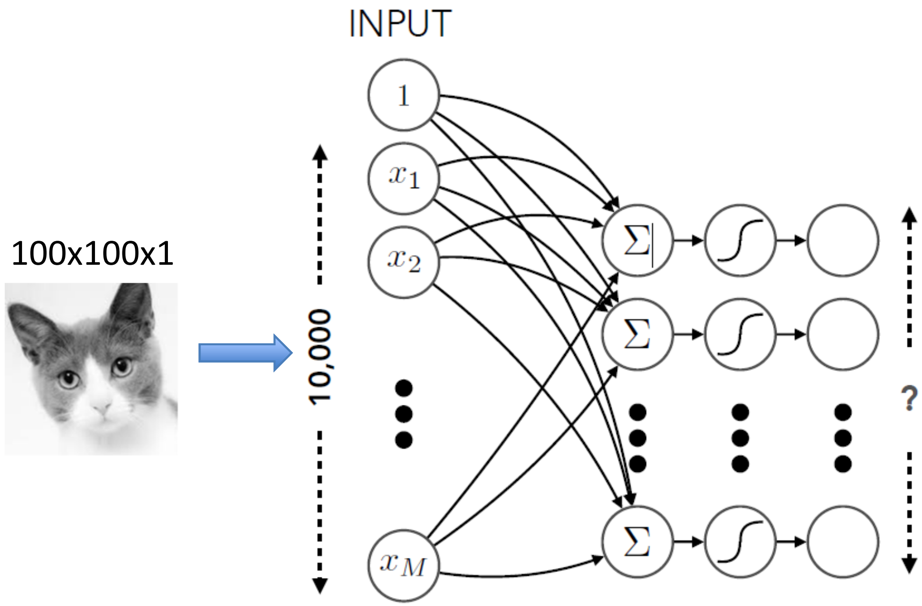
# Inductive bias

- Two components for solving a task
  - Priors (bias): things assumed beforehand
  - Learning (variance): things extracted from data

**strong priors, minimal learning**
- fast/easy to learn and deploy
- may be too rigid, unadaptable

**weak priors, much learning**
- slow/difficult to learn and deploy
- flexible, adaptable

# Neural network with images




**How many parameters?**

#units x 10,000 = # weights

1 -------------------→ 10,000

10 -----------------→ 100,000

100 ---------------→ 1,000,000

…

100,000 ---------→1,000,000,000

# Convolutional networks

- Scale up neural networks to process very large images / video sequences
  - Sparse connections
  - <u>Parameter sharing</u>

- Automatically generalize across spatial translations of inputs

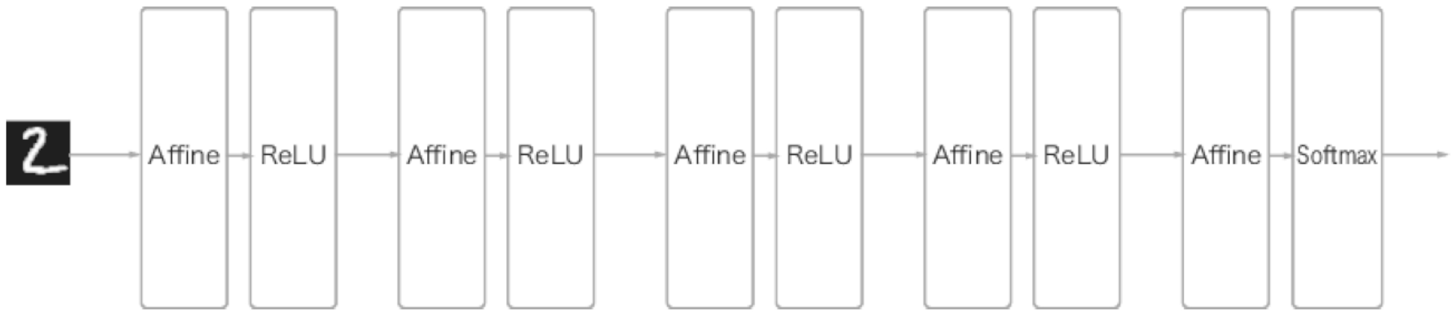
- Applicable to any input that is laid out on a grid (1-D, 2-D, 3-D, …)

# Key idea

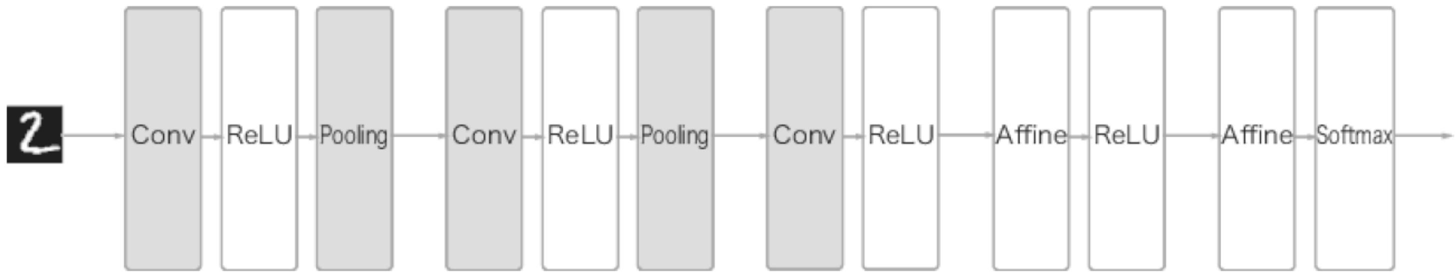
- Replace matrix multiplication in neural nets with convolution
- Everything else stays almost the same

# Convolutional networks

Network with Fully Connected (FC) layers



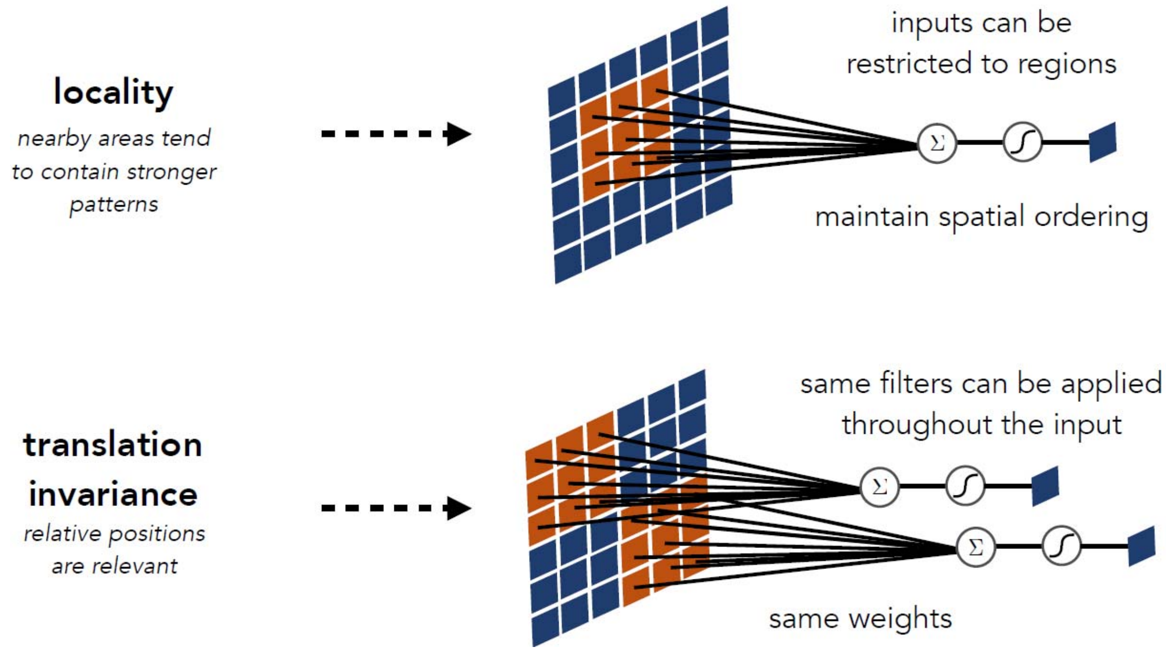

Network with convolutional layers



# Spatial structure in images

- Locality
  - Nearby areas tend to contain stronger patterns
- Translation invariance
  - Relative (rather than absolute) positions are relevant

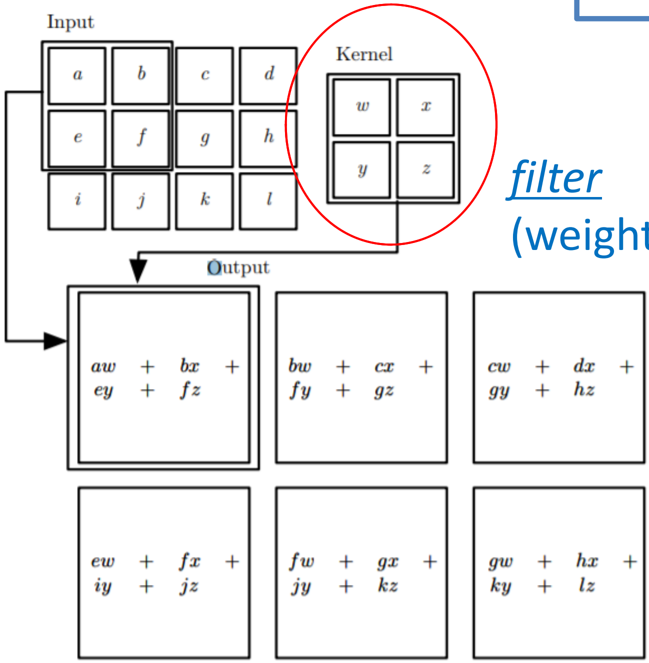
# Exploit spatial structure in images



# Convolution

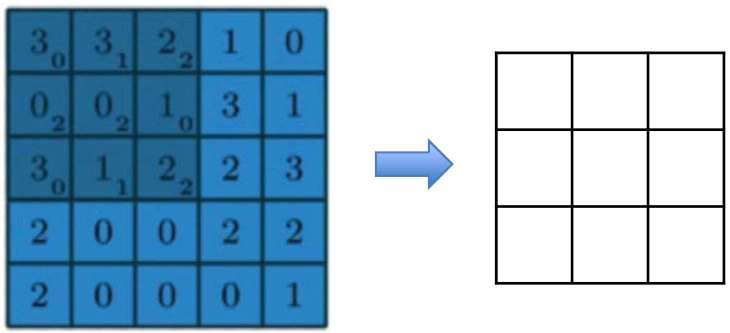
- 2D convolution

take *dot product* of filter and each input location

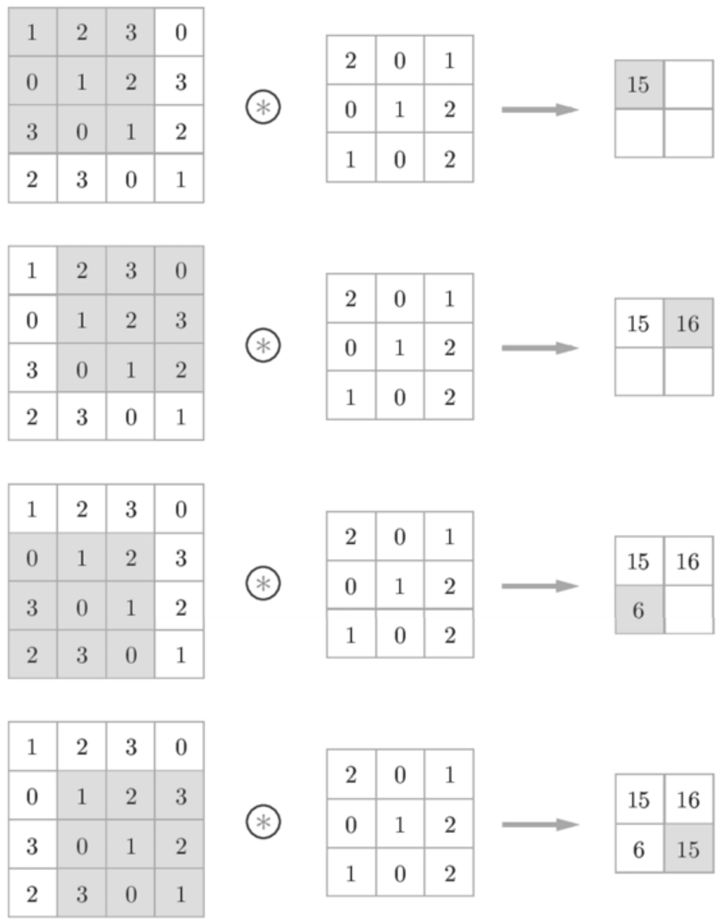



*filter*
(weight matrix)

filter weights: $\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$



- 2D convolution

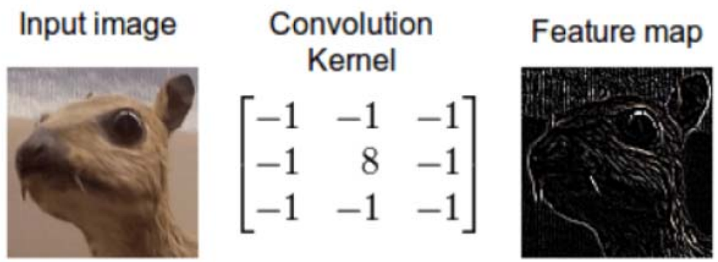


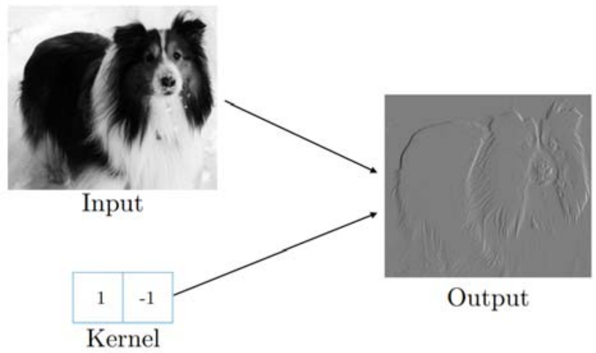
- Convolution with bias

$$\underset{\text{입력 데이터}}{\begin{bmatrix}1&2&3&0\\0&1&2&3\\3&0&1&2\\2&3&0&1\end{bmatrix}} \circledast \underset{\text{필터}}{\begin{bmatrix}2&0&1\\0&1&2\\1&0&2\end{bmatrix}} \rightarrow \begin{bmatrix}15&16\\6&15\end{bmatrix} + \underset{\text{편향}}{\boxed{3}} \rightarrow \underset{\text{출력 데이터}}{\begin{bmatrix}18&19\\9&18\end{bmatrix}}$$

Bias (1x1)

# Convolution example

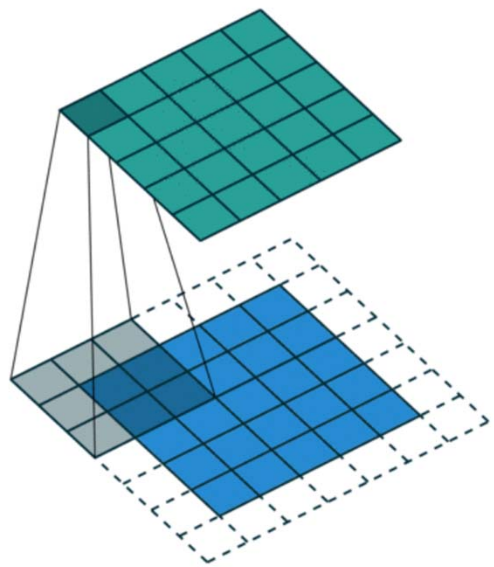



- Acts like a high-pass filter: pixels near <u>edges</u> survive
- Can be performed very efficiently on modern libraries and GPUs

# Padding

- Use padding to preserve spatial size

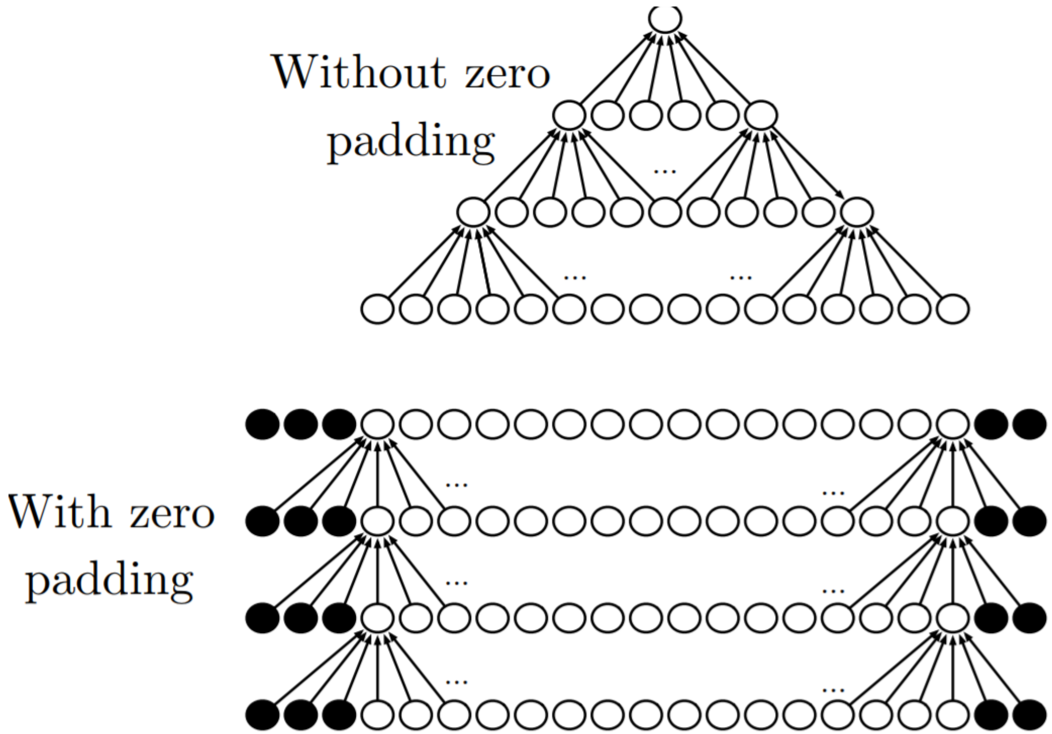


typically add *zeros* around the perimeter

# Zero-padding

$$\underset{(4,\ 4)}{\begin{bmatrix}1&2&3&0\\0&1&2&3\\3&0&1&2\\2&3&0&1\end{bmatrix}} \circledast \underset{(3,\ 3)}{\begin{bmatrix}2&0&1\\0&1&2\\1&0&2\end{bmatrix}} \longrightarrow \underset{(4,\ 4)}{\begin{bmatrix}7&12&10&2\\4&15&16&10\\10&6&15&6\\8&10&4&3\end{bmatrix}}$$

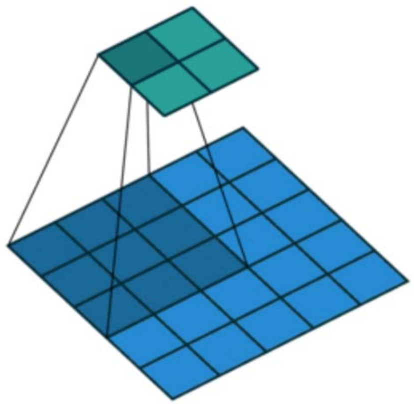
# Zero padding controls size



# Stride

- Use stride to downsample the input

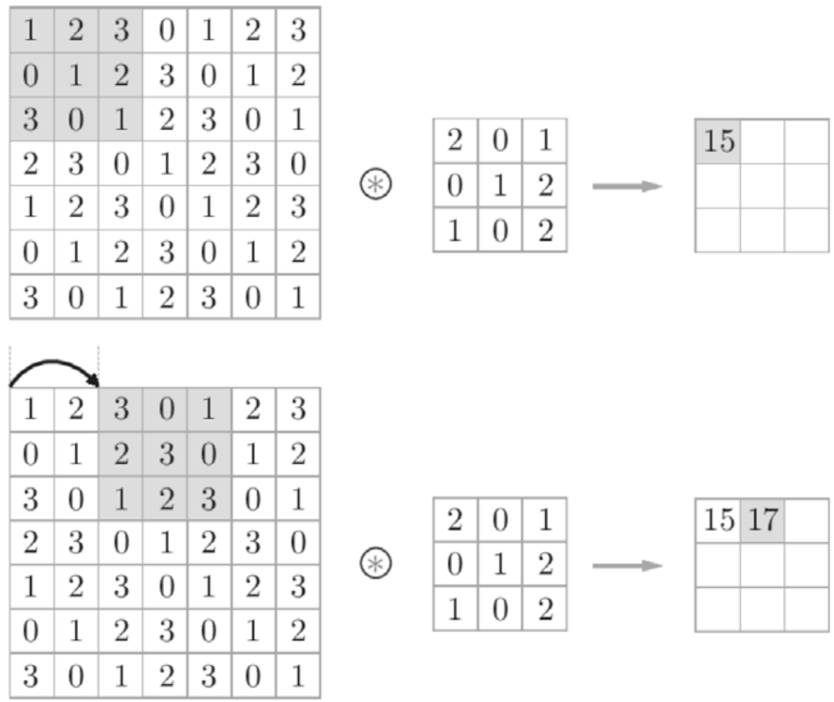
stride = 2



Move *2* pixels at a time

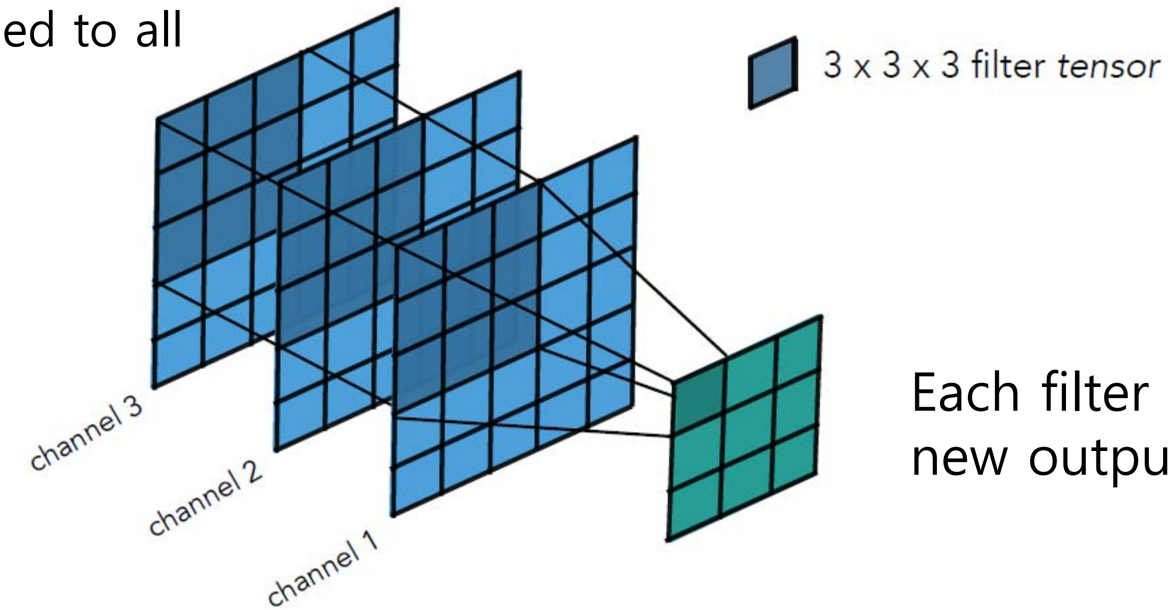
# Stride

stride = 2



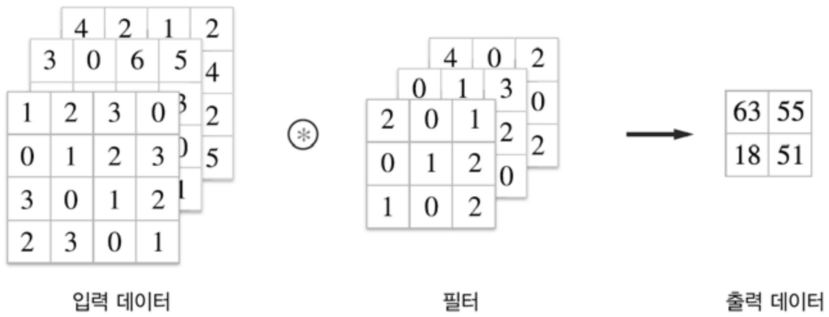
# Multiple channels

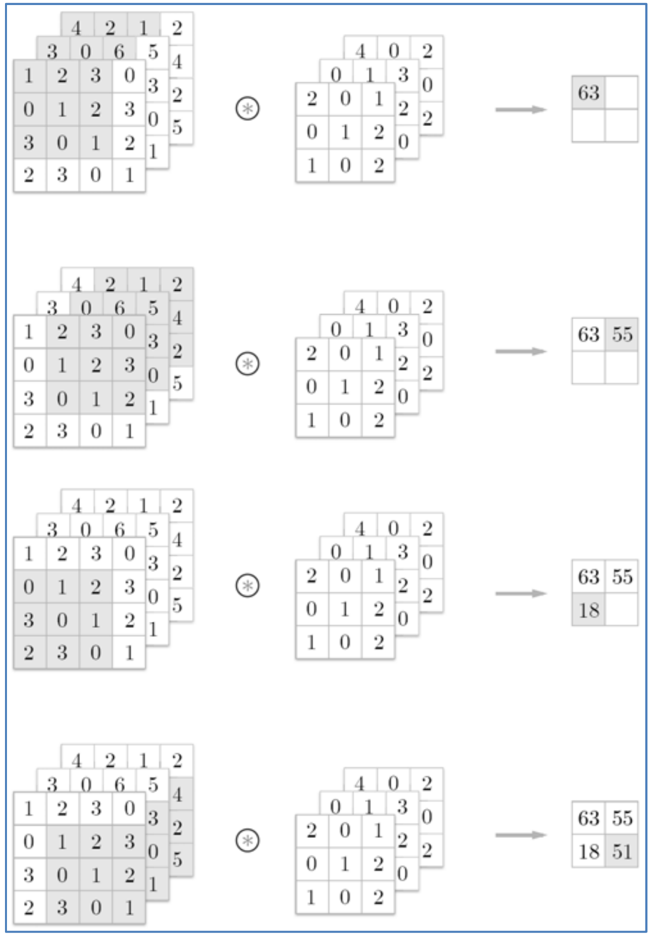
Filters are applied to all input channels



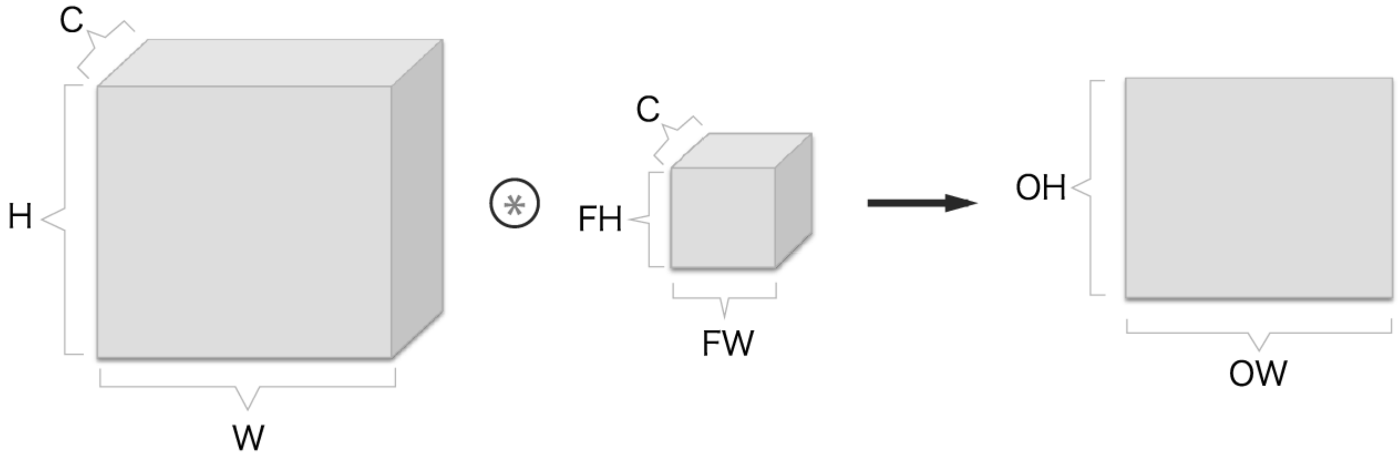

Each filter results in a new output channel

# 3D convolution



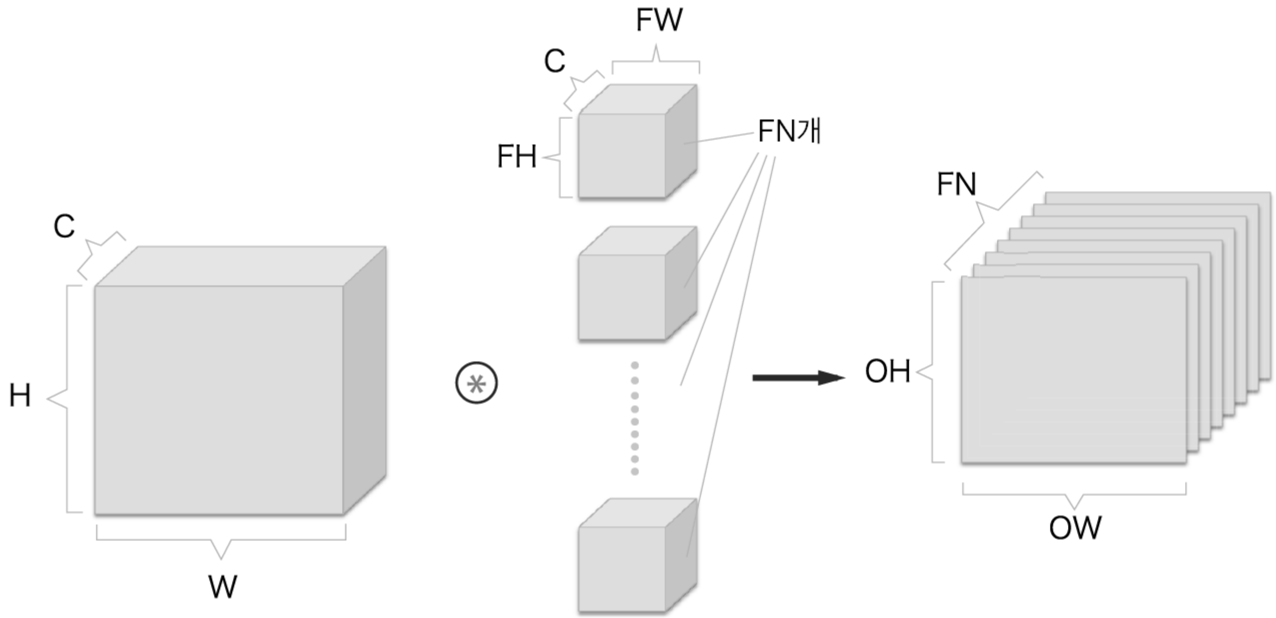



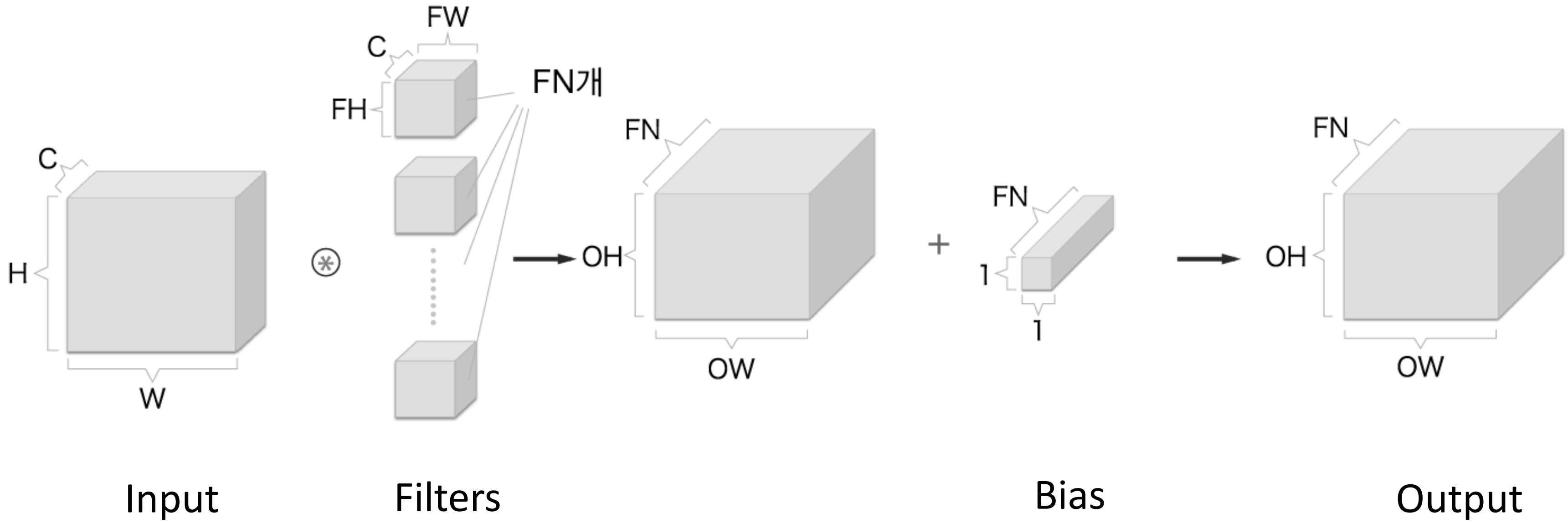
# Convolution



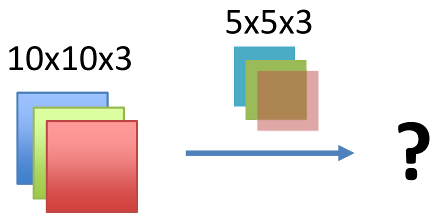

1 filter → 1 output feature map

# Multiple filters




C
H
W
FW
C
FH
FN개
FN
OH
OW
+
FN
1
1
FN
OH
OW
Input
Filters
Bias
Output

# Pop quiz 1



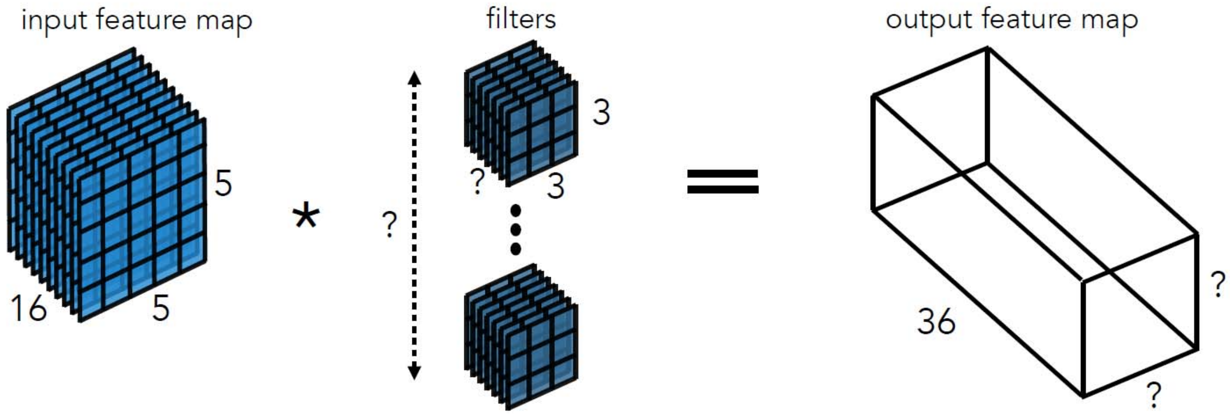

# input channels = 3, each 10 x 10

Kernel (filter) of size 5 x 5 x 3 is applied with stride 1

- What is the dimension of the output channels?

- What if the input maps are zero-padded?

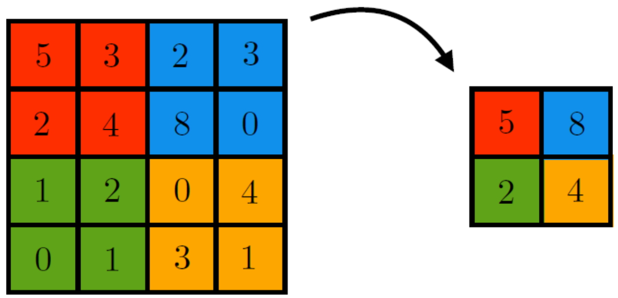
# Pop quiz 2

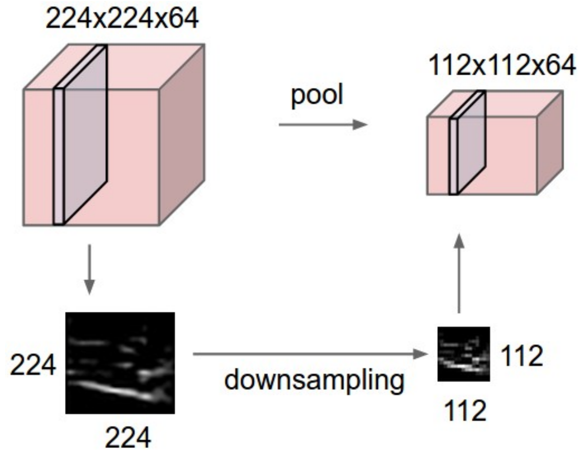



- If we use unit (1) stride and no padding,
  - What is each filter size?
  - How many filters are there?
  - What is the output filter map size?

# Pooling

- Locally aggregates values in each feature map

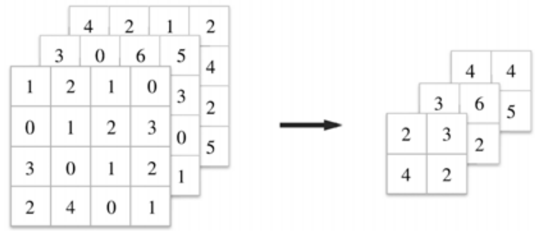


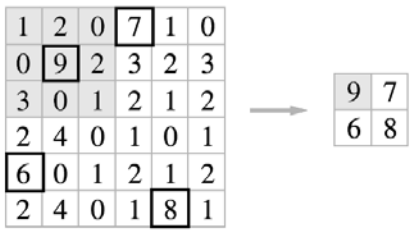

predefined operation: <u>maximum</u>, average, etc.

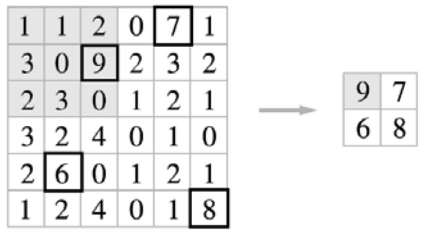
# Pooling layer

- No parameter to learn
- The number of channels does not change

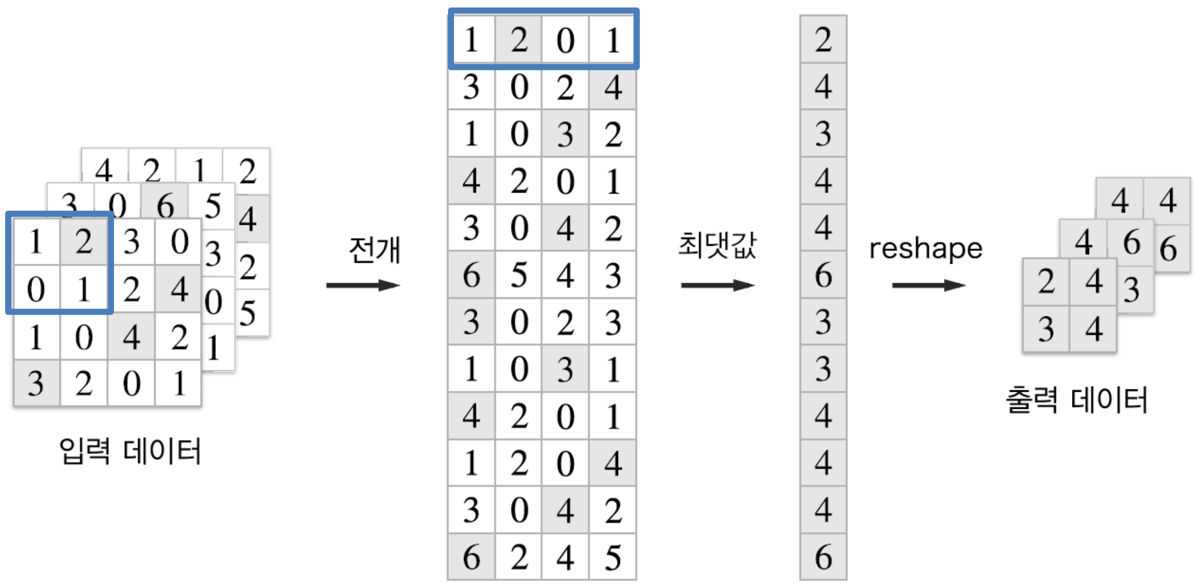


- Robust to input variation



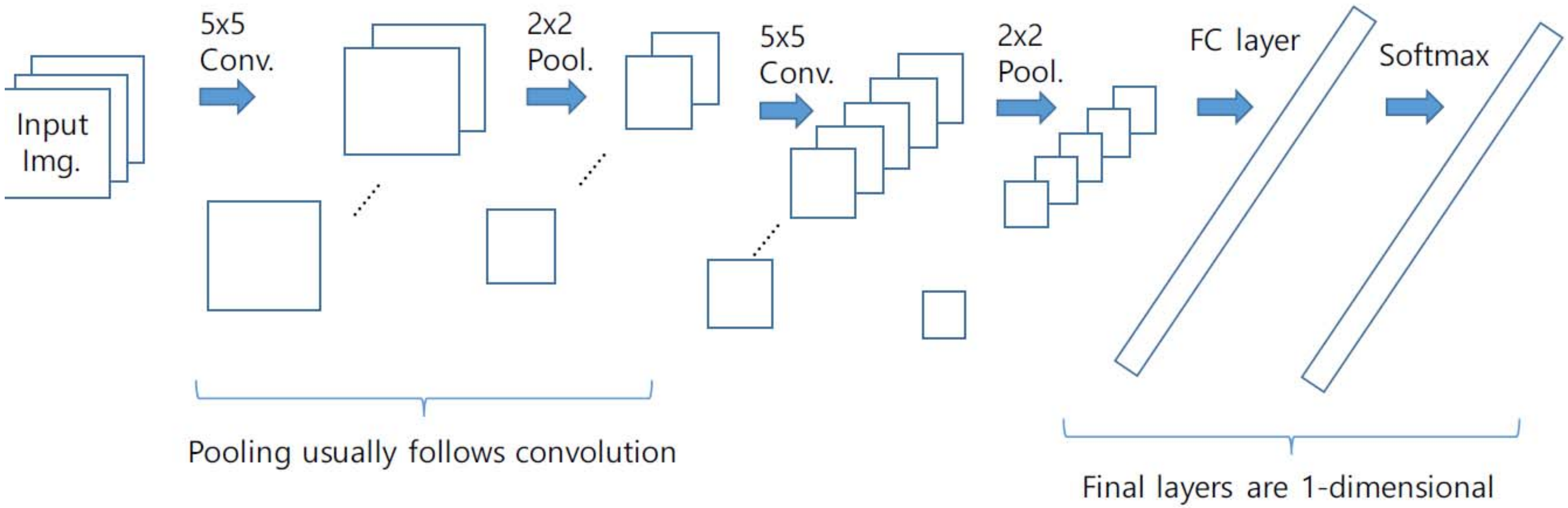


# Implementation: max pooling



# Convolutional Neural network

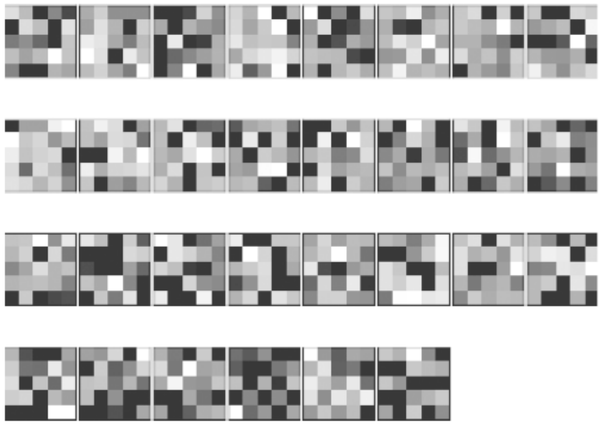
- A deep neural network composed of convolution-pooling layers



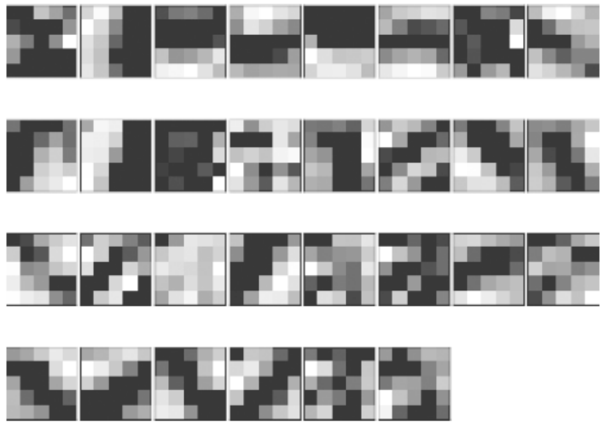
# Filter visualization

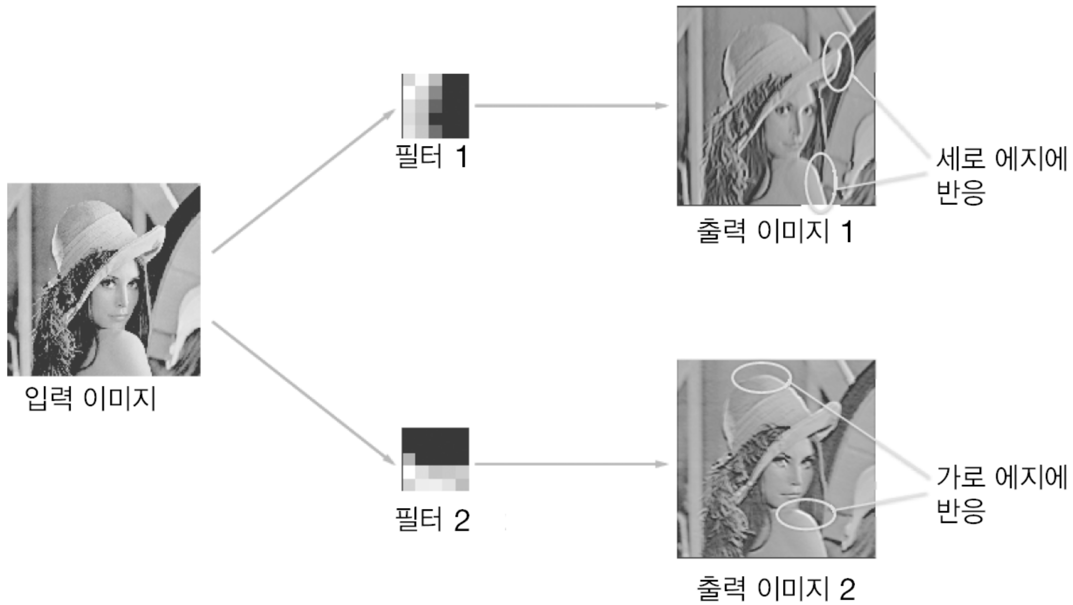
- Filters at 1$^{st}$ layer before and after training

학습 전



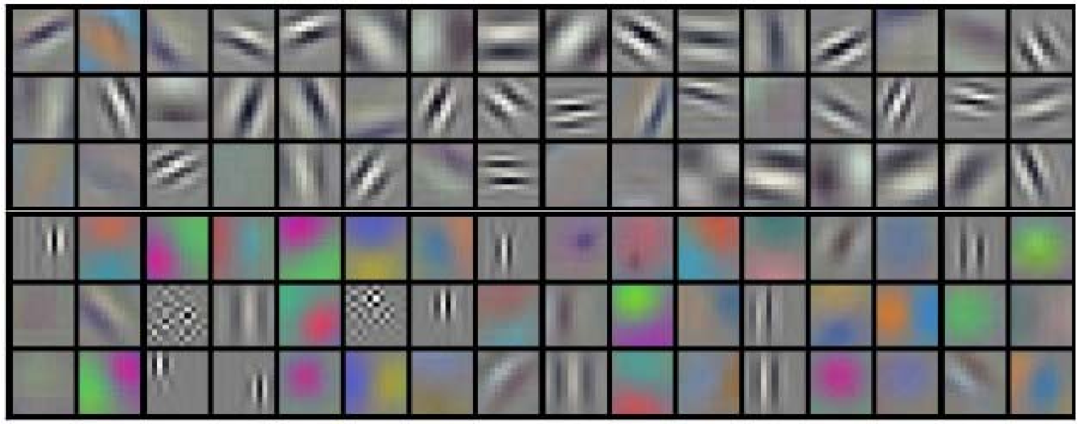
학습 후





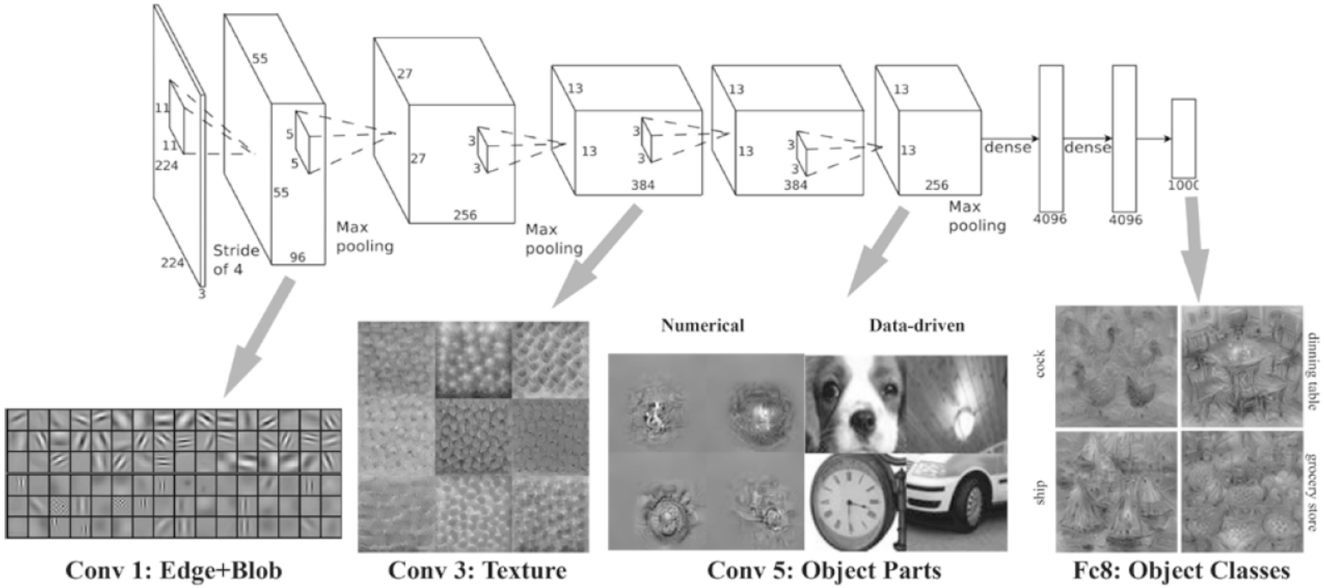
입력 이미지
필터 1
필터 2
출력 이미지 1
출력 이미지 2
세로 에지에
반응
가로 에지에
반응

# Filter visualization



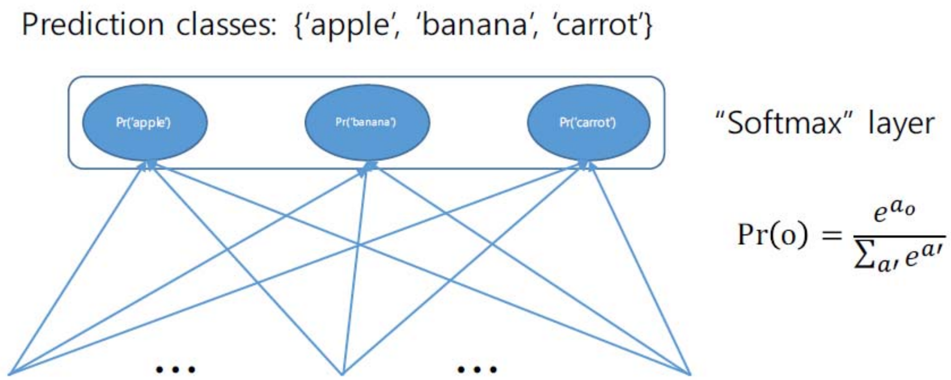
Each of the 96 filters is of size [11x11x3] and shared by the neurons in one depth slice

# Information extracted from each layer



# Classification layer

- Many CNNs are used to perform classification
- The final layer of such CNN is used to represent the class probabilities



# Classification layer

- Since the softmax layer specifies the probability, the cost should be computed differently
- Cross entropy: information-theoretic measure of difference between two distributions

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Prediction:

| |
|---|
| Pr('apple') = 0.2 |
| Pr('banana') = 0.1 |
| Pr('carrot') = 0.7 |

q = (0.2, 0.1, 0.7)

Ground truth: 'carrot'

One-hot representation

p = (0, 0, 1.0)

H(p, q) = -log(0.7) = 0.357

# Training CNNs

- Back-propagation
  - SGD, Adam, AdaDelta, AdaGrad, etc.

- (Trainable) parameters:
  - Kernel patch for each layer
  - bias vectors for each kernel

- (non-trainable) hyperparameters
  - Network depth
  - Number of kernels per layer, kernel size, stride, zero-padding
  - Which activation functions

# Natural image datasets

***Caltech-101***

101 classes,
9,146 images

***Caltech-256***

256 classes,
30,607 images

***CIFAR-10***

10 classes,
60,000 images

***CIFAR-100***

100 classes,
60,000 images

***ImageNet***

***Competition***

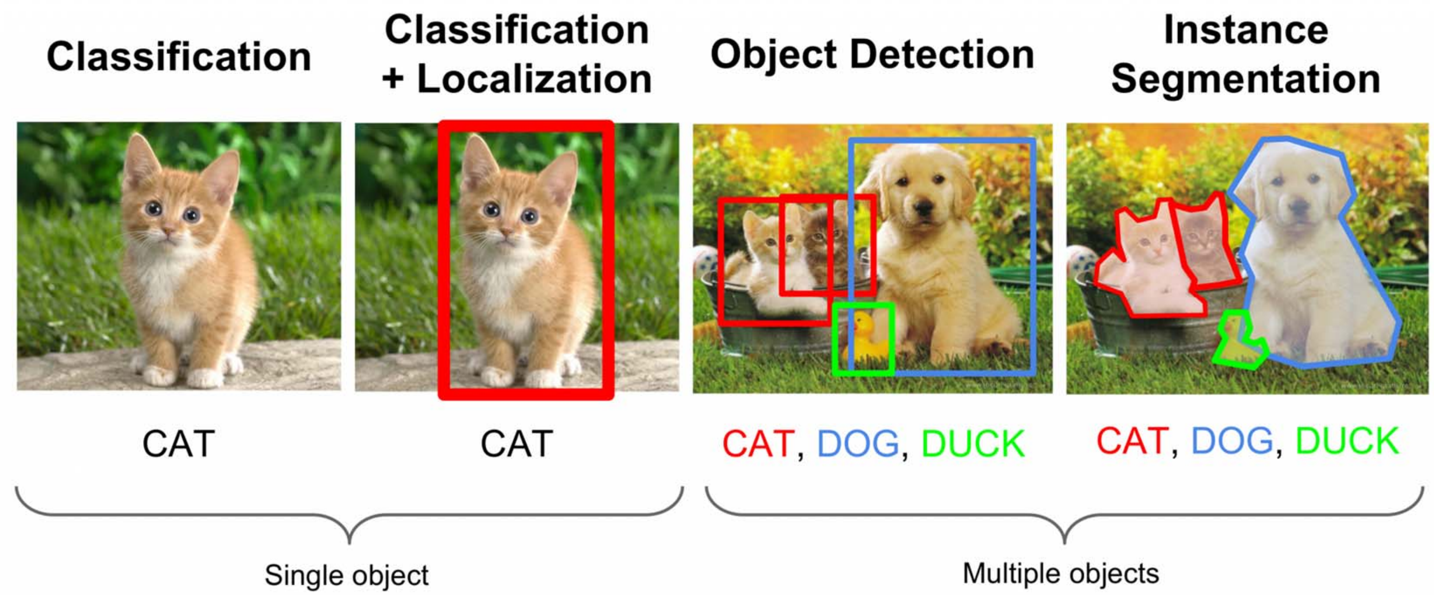
1,000 classes,
1.2 million images

***Full***

21,841 classes,
14 million images

•••

# Computer vision tasks



# Image classification

This image is CC0 public domain

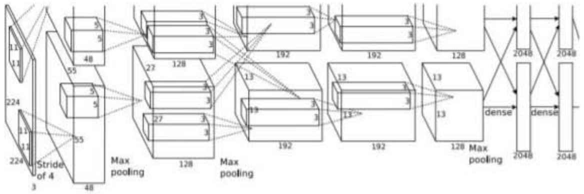


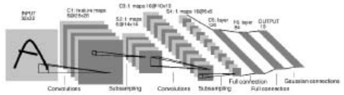
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**Vector:**
4096

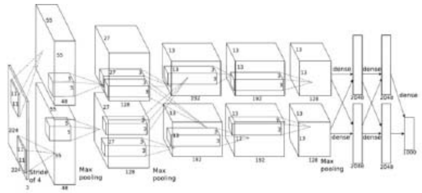
**Fully-Connected:**
4096 to 1000

**Class Scores**
Cat: 0.9
Dog: 0.05
Car: 0.01
...

# CNNs for classification

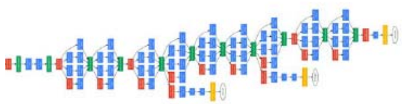


*LeNet*



*AlexNet*

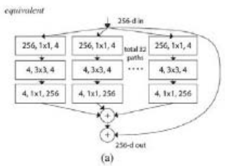
*VGG*



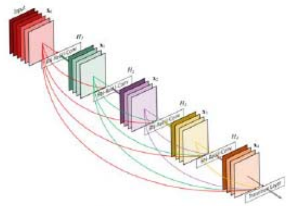
*GoogLeNet*

*ResNet*

*Inception v4*



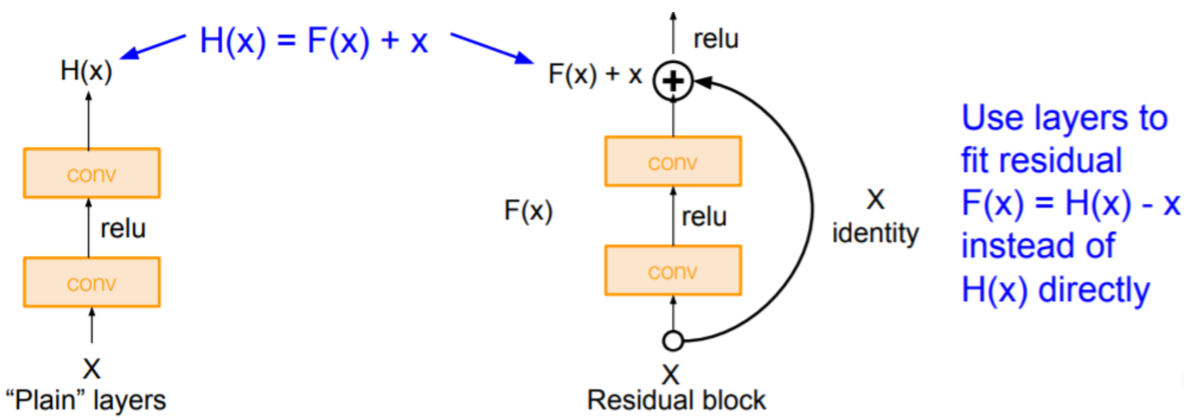
*ResNeXt*



*DenseNet*

# ResNet [He et al., 2015]

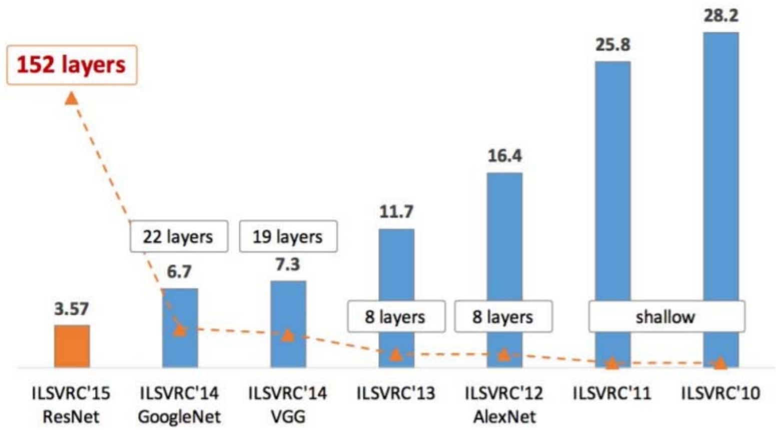
- Very deep networks using residual connections



# ResNet [He et al., 2015]

- Experiment
  - Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
  - Deeper networks now achieve lower training error as expected
  - Swept 1st place in all ILSVRC and COCO 2015 competitions al results

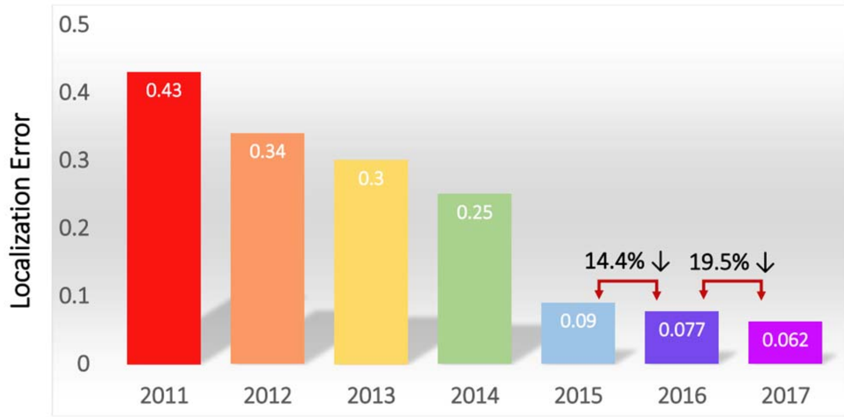
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# ILSVRC (2010-2017)

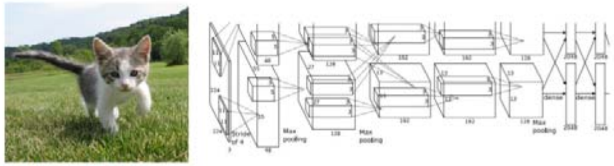
Classification Results


Localization Results

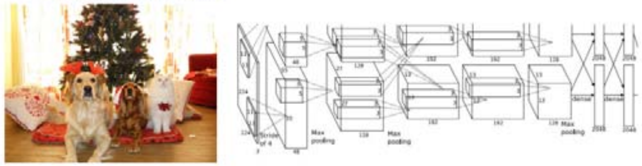


# Object detection

- As regression?

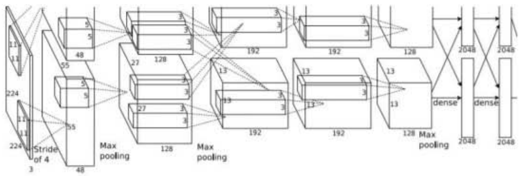


CAT: (x, y, w, h) 4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)
16 numbers

- As classification:
  Sliding window?

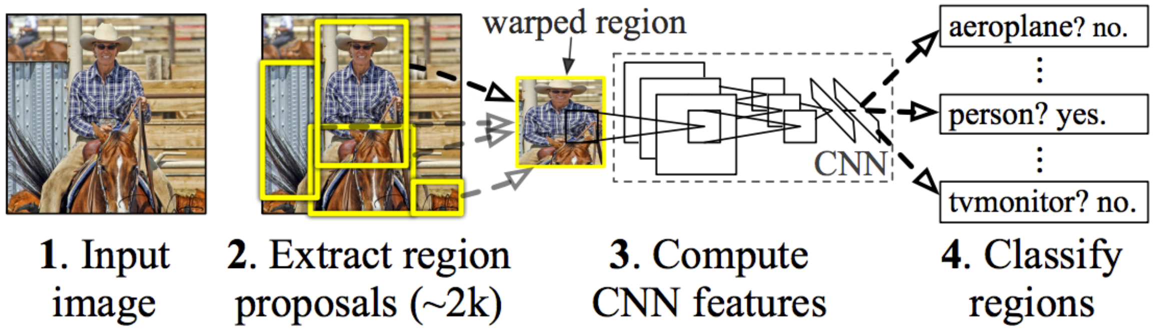


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations and scales, very computationally expensive!

# R-CNN (2014): an early application of CNNs to Object Detection

- Inputs: Image
- Outputs: Bounding boxes + labels for each object in the image.

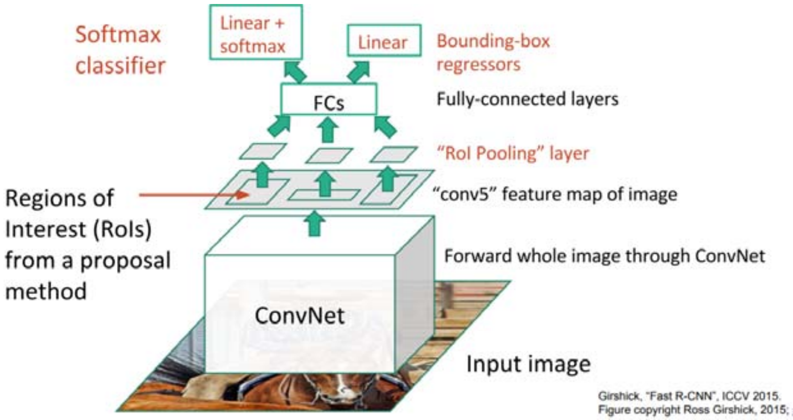



Source: https://arxiv.org/abs/1311.2524.

1. Generate a set of proposals for bounding boxes.
2. Run the images in the bounding boxes through a pre-trained AlexNet and finally an SVM to see what object the image in the box is.
3. Run the box through a linear regression model to output tighter coordinates for the box once the object has been classified.

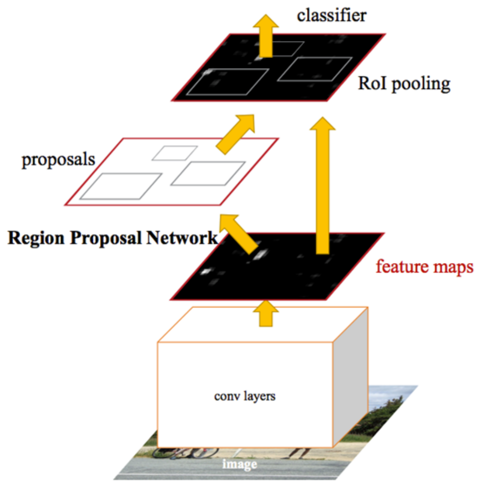
# Fast R-CNN (2015): Speeding up and Simplifying R-CNN

- Fast R-CNN combined the CNN, classifier, and bounding box regressor into one, single network



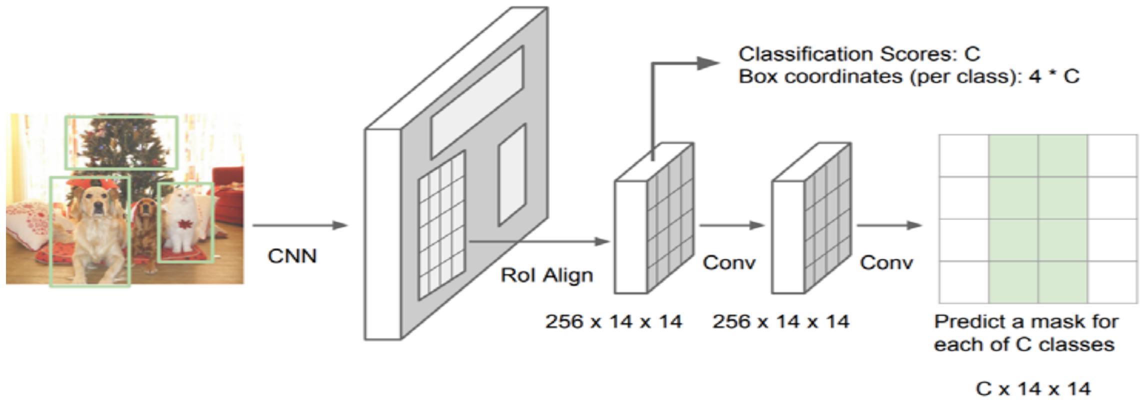

# Faster R-CNN (2016): Speeding up Region proposal

- In Faster R-CNN, a single CNN is used for region proposals, and classifications



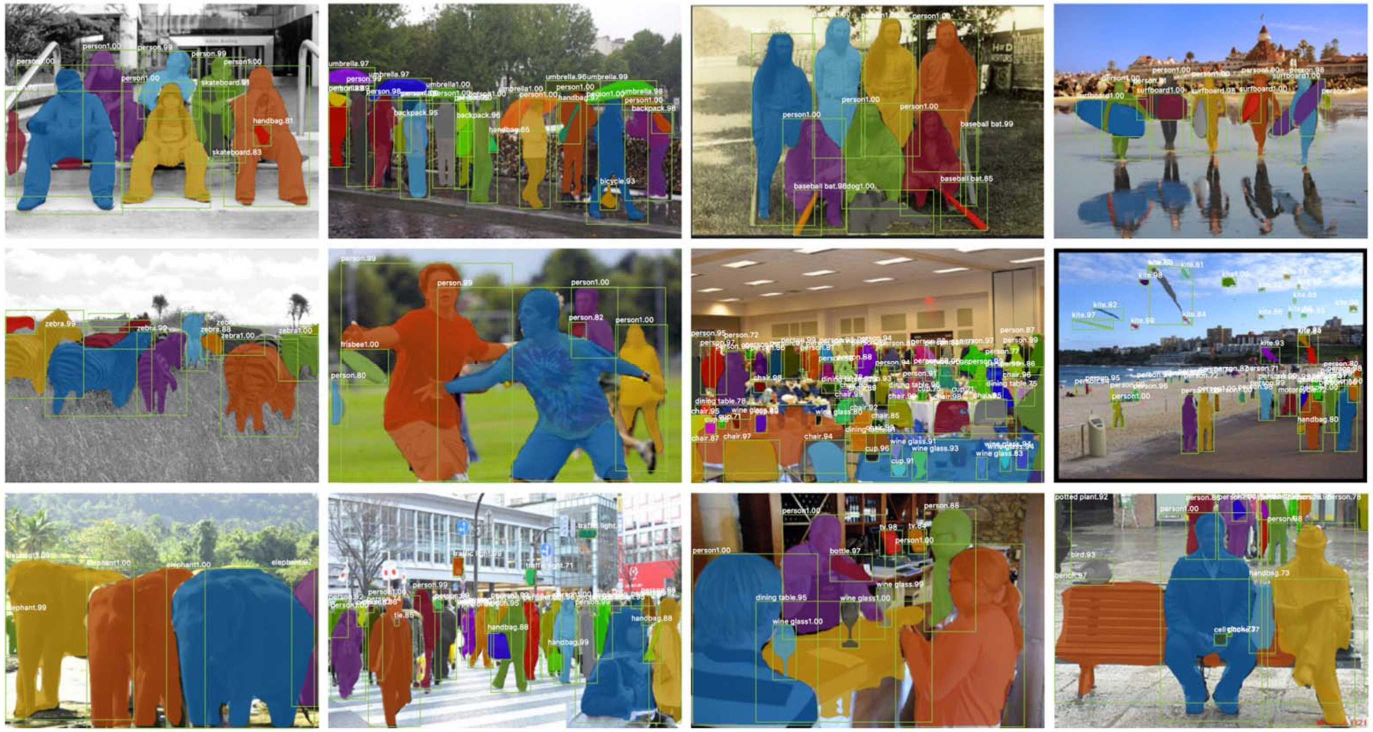
# Mask R-CNN (2017): Extending Faster R-CNN for pixel level segmentation

- In Mask R-CNN, a Fully Convolutional Network (FCN) is added on top of the CNN features of Faster R-CNN to generate a **mask** (segmentation output)



# Mask R-CNN

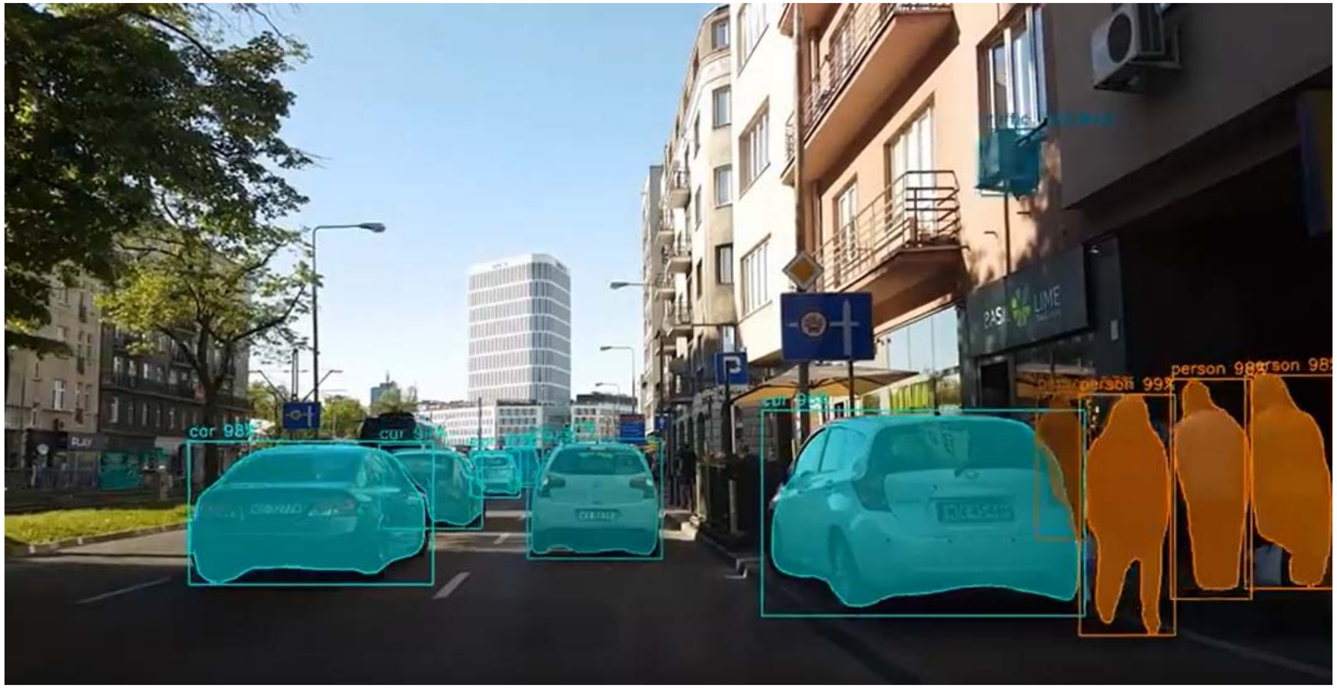
Mask R-CNN is able to segment as well as classify the objects in an image.



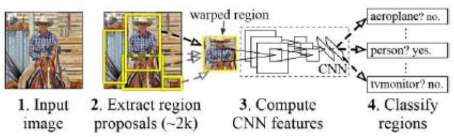
Source: https://arxiv.org/abs/1703.06870.

# Mask R-CNN COCO Object detection and segmentation

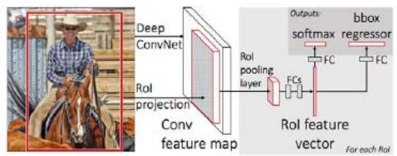
https://www.youtube.com/watch?v=OOT3UIXZztE



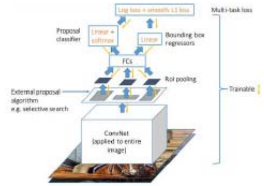
# CNNs for detection, segmentation, etc.



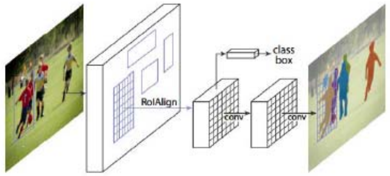

R-CNN



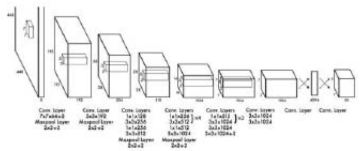

Fast R-CNN



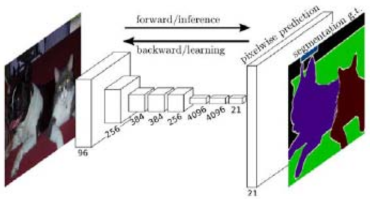
Faster R-CNN



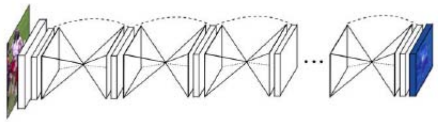

Mask R-CNN



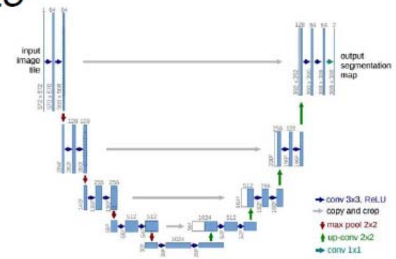
YOLO




FCN



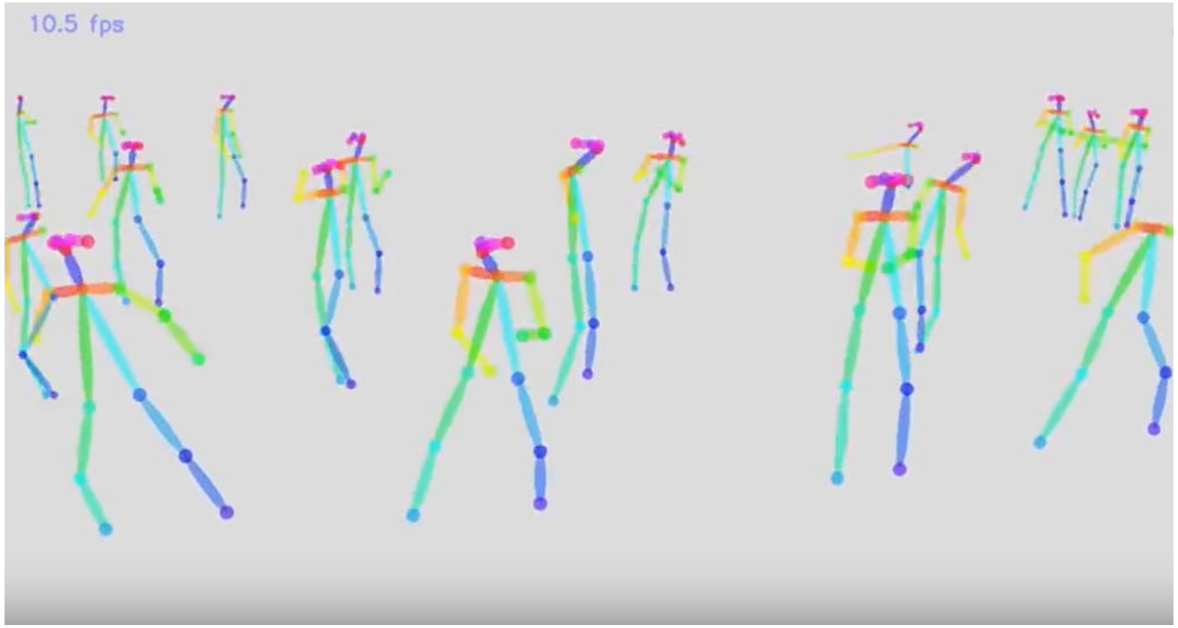
Hourglass




U-Net

# Real-time 2D human pose estimation

https://www.youtube.com/watch?v=pW6nZXeWlGM



# Neural style transfer




Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016

# Neural style transfer

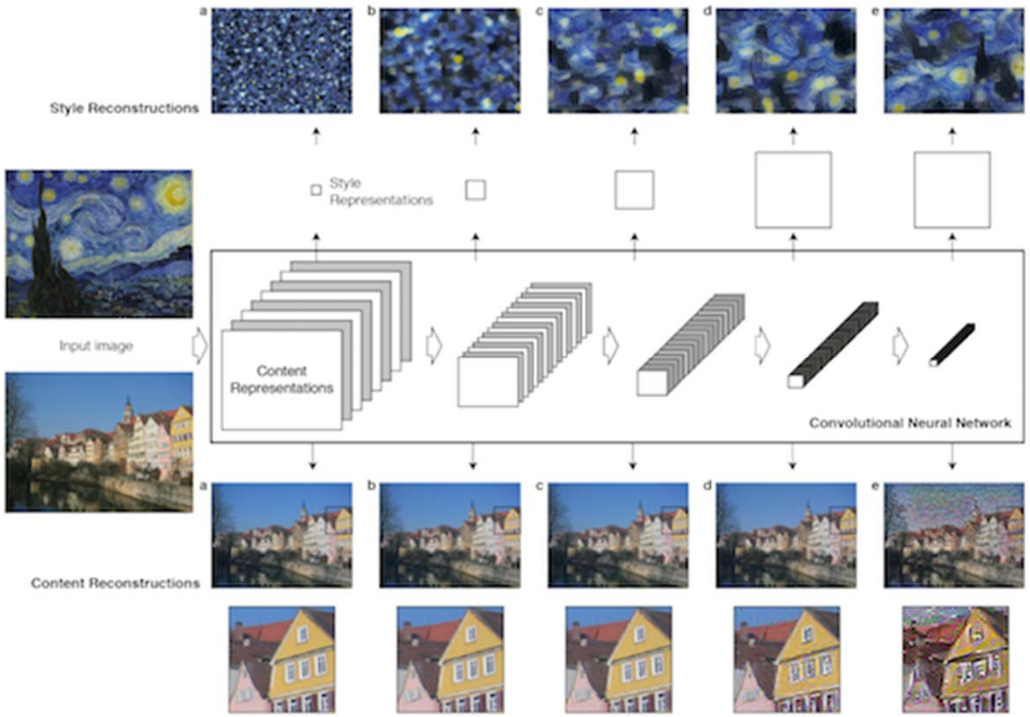
- Loss function: content loss + style loss
  - <u>Content loss</u> = Euclidean distance between the feature map F from the <u>content image</u> and the one P from the <u>generated output image</u>
  - <u>Style loss</u>: take the so called "*Gram matrix*" of each feature map (from <u>*style image, and the output image*</u>), and compute their distances across layers

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$

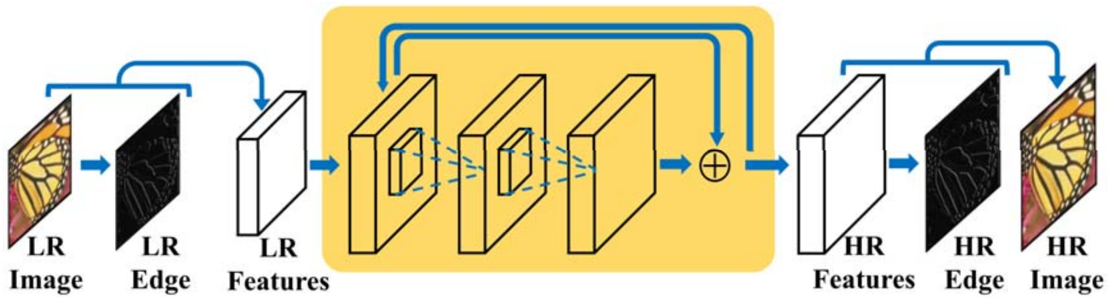
$$\mathcal{L}_{content} = \frac{1}{2} \sum_{i,j} (F_{ij}^{l} - P_{ij}^{l})^2$$

$$\mathcal{L}_{style} = \frac{1}{2} \sum_{l=0}^{L} (G_{ij}^{l} - A_{ij}^{l})^2$$

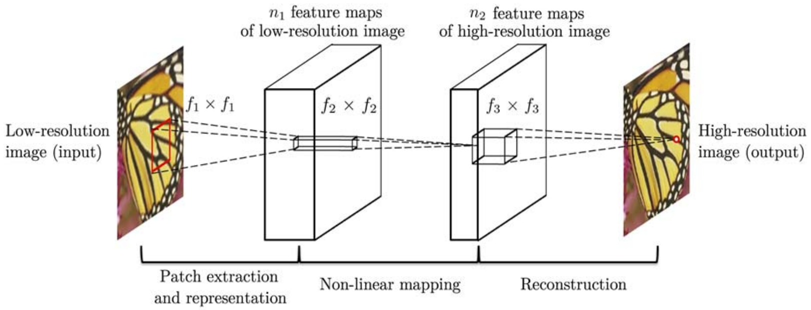
# Neural style transfer



# Image super-resolution

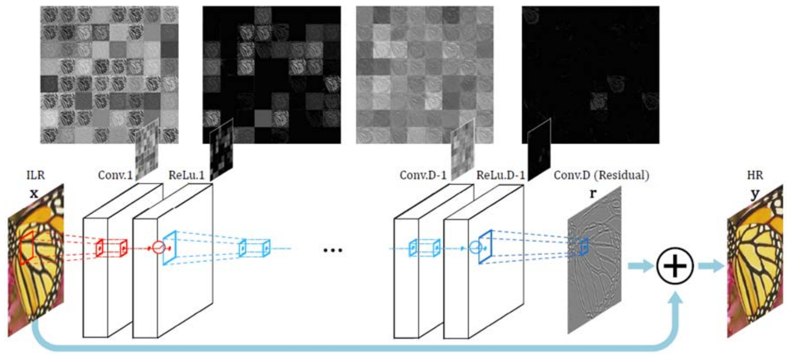



**SRCNN (ECCV 2014):** Image Super-Resolution Using Deep Convolutional Networks




- Simple 3-layer CNN
    - End-to-end learning (LR - HR)
    - Convolution and ReLU

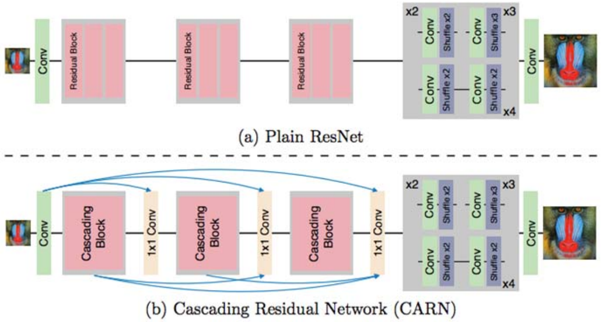
**VDSR (CVPR 2016):** Accurate Image Super-Resolution Using Very Deep Convolutional Networks



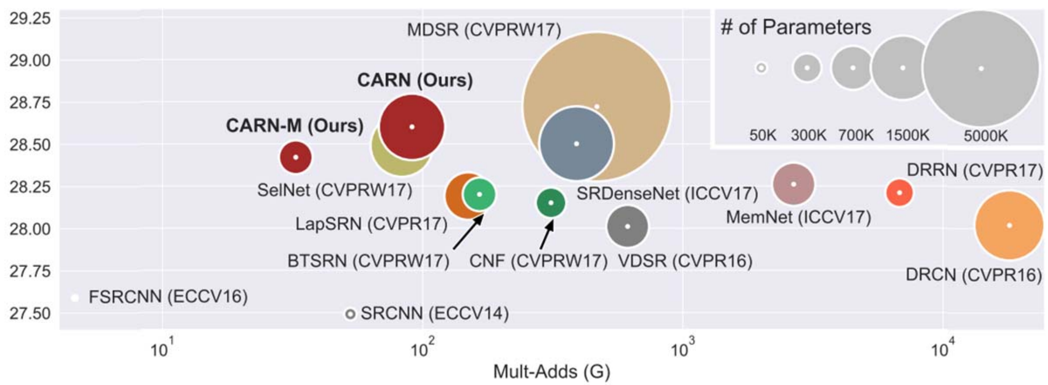

- Very deep CNN (20 layers)
    - Skip connection
    - No dimension reduction such as pooling

# CARN (Ahn et al. ECCV 2018): Fast, Accurate, and Lightweight Super-Resolution with Cascading Residual Network

- Deep learning-based super-resolution
  - (+) Accurate
  - (-) Slow, (-) Heavy
- To effectively perform super-resolution, we add many short-cut connections




See how the PSNR (y-axis) varies depending on:
- computational cost (x-axis) and
- model size (circle area)



# CNN baseline

- Download a pre-trained network
- Or copy-paste an architecture from a related task
- Or:
  - Deep residual network
  - Batch normalization
  - Adam

# Reference

- Books
  - Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville
  - 밑바닥부터 시작하는 딥러닝, 사이토 고키

- Lecture notes
  - Stanford cs231n
  - Caltech CS/CNS/EE155