KIAPS
KOREA INSTITUTE OF ATMOSPHERIC PREDICTION SYSTEMS

위성 격자 자료와 합성곱 신경망(CNN)을 통한 공간 정보 학습

차세대수치예보모델개발사업단

전 현 주

2026.2.23

실습자료

Brief Bio

Hyeon-Ju Jeon

Research Scientist at KIAPS

Contact Information

- hjjeon@kiaps.org, higd963@gmail.com
- https://higd963.github.io/

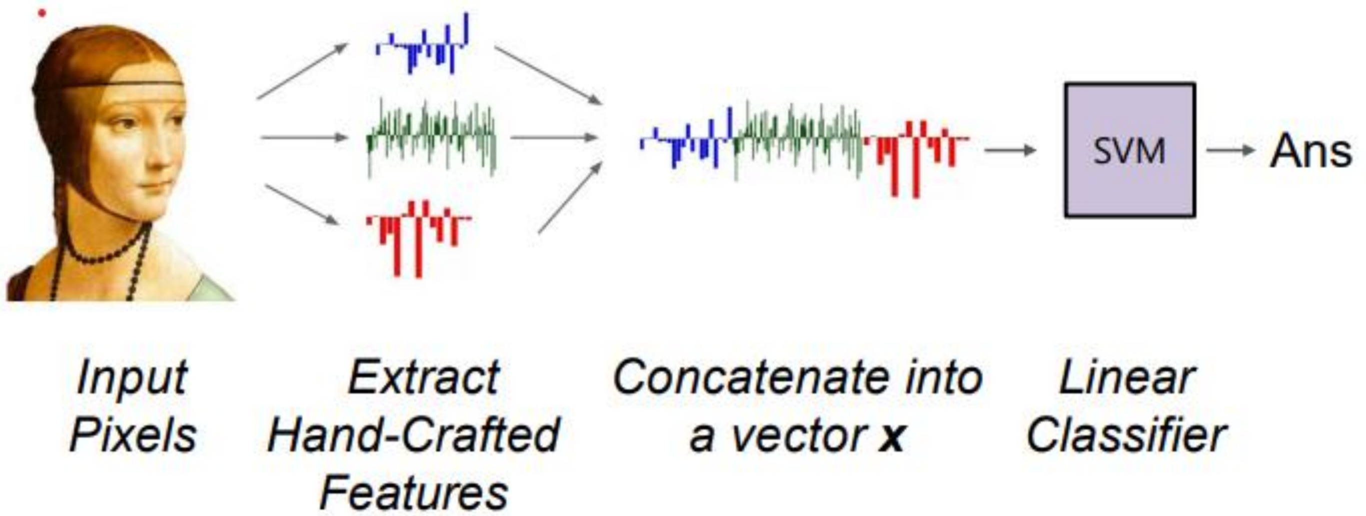
Research Interest

- **Multimodal and Irregular Real-world Data Mining**
- **Applied Machine Learning**
- **Spatiotemporal Graph Neural Networks**
- **Application domains:** Weather prediction, Disease prediction, Biomedical, Bibliographic network analysis, Anomaly detection, etc.

Professional Experience

- Ph.D., Chung-Ang University (Leave of absence)
- **Research Scientist, KIAPS (2021.10 – Present)**
- Lecturer, LG CNS (2021 – 2022)

Life Before Deep Learning



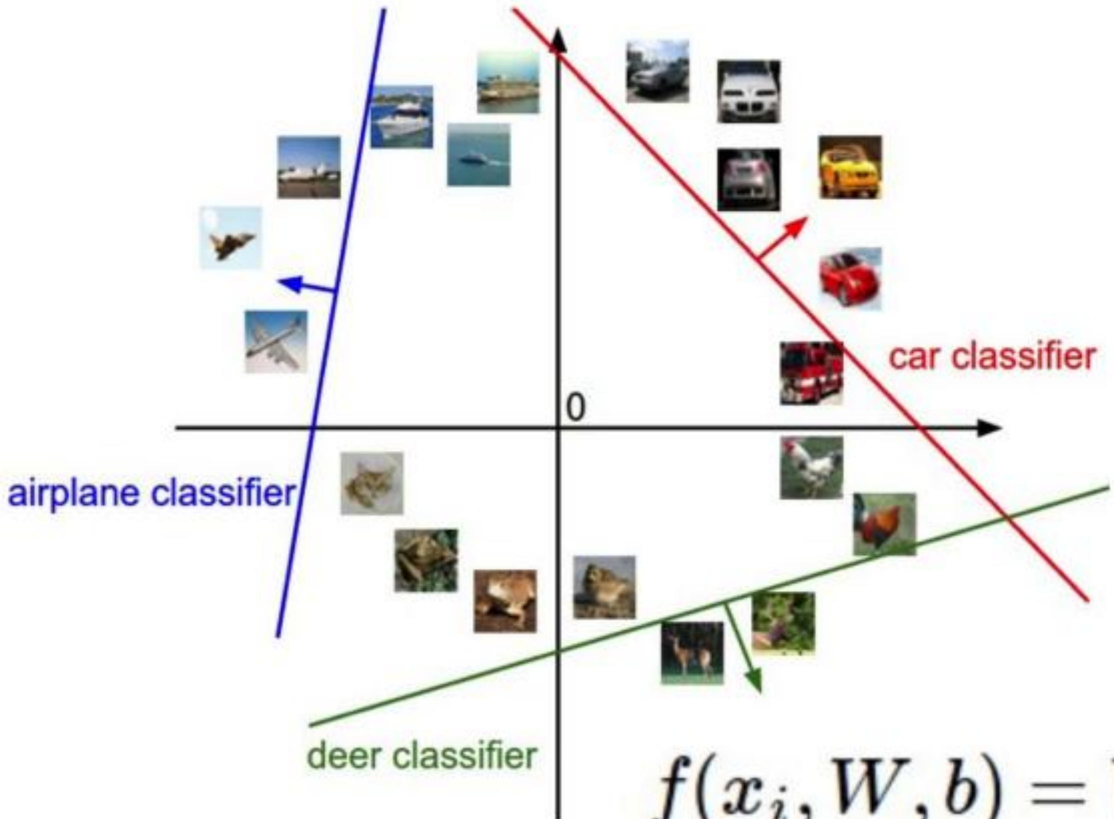
Why use features? Why not pixels?

$$f(x_i, W, b) = Wx_i + b$$

Q: What would be a very hard set of classes for a linear classifier to distinguish?

(assuming x = pixels)

Linearly separable classes



$$f(x_i, W, b) = Wx_i + b$$

Aside: Image Features

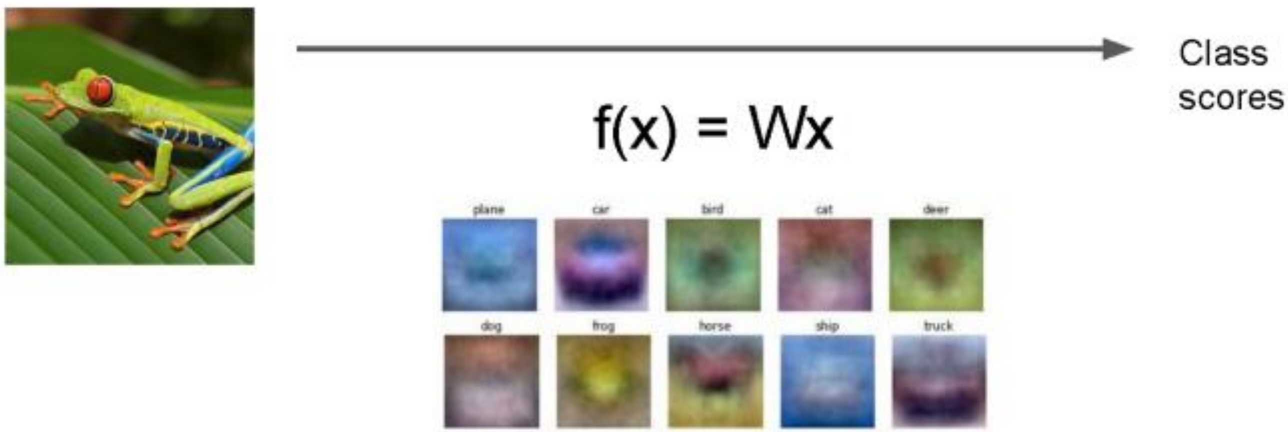
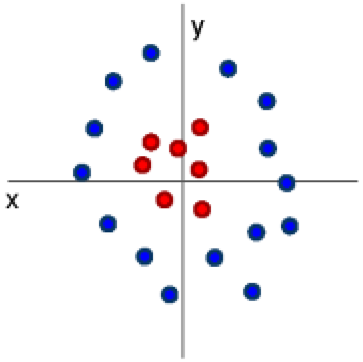
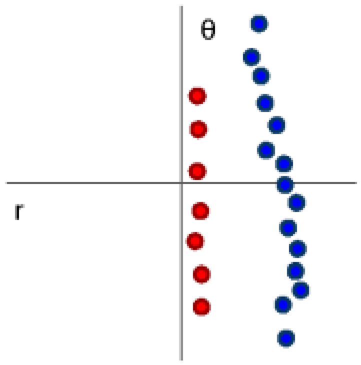


Image Features: Motivation



Cannot separate red and blue points with linear classifier

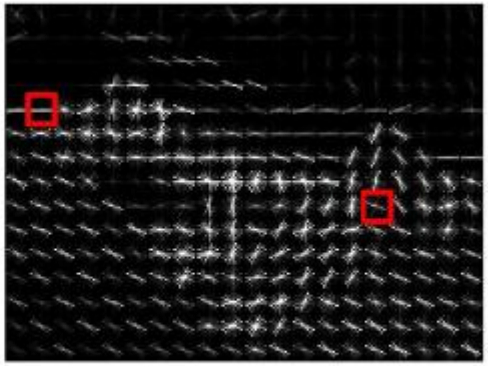
$f(x, y) = (r(x, y), \theta(x, y))$



After applying feature transform, points can be separated by linear classifier

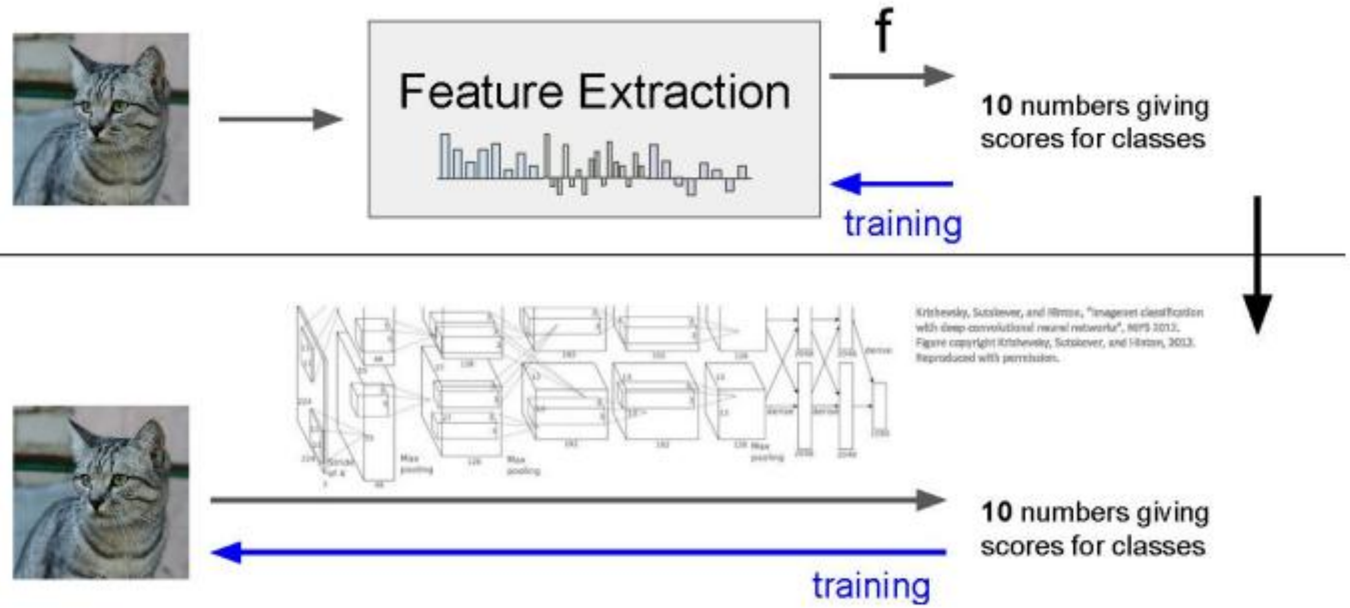
Example: Histogram of Oriented Gradients (HoG)

Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins

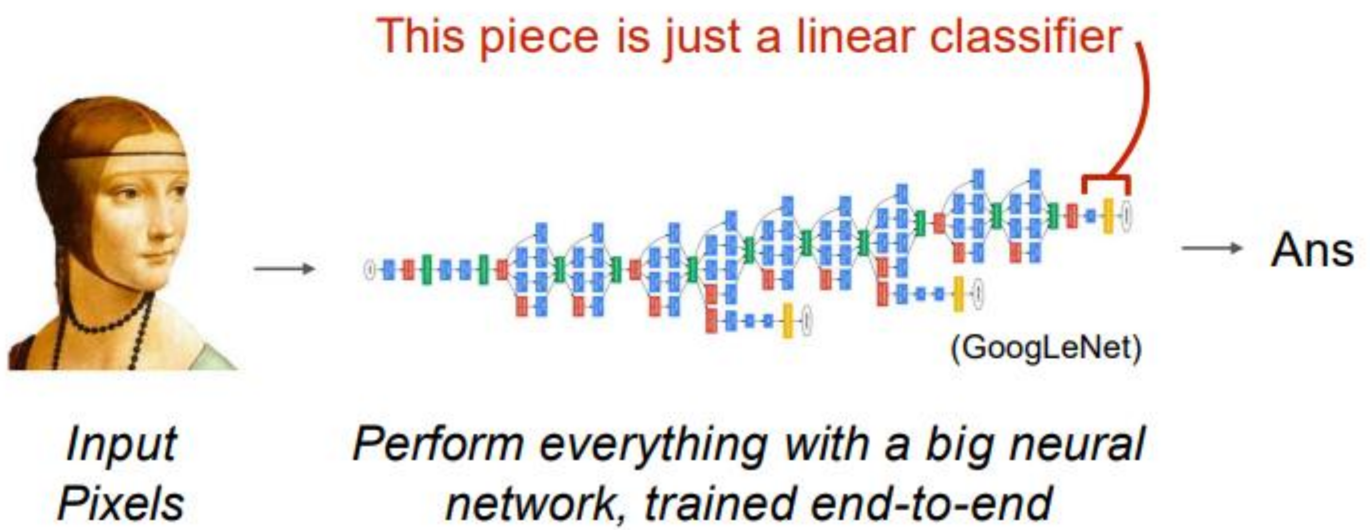


Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

Image features vs ConvNets

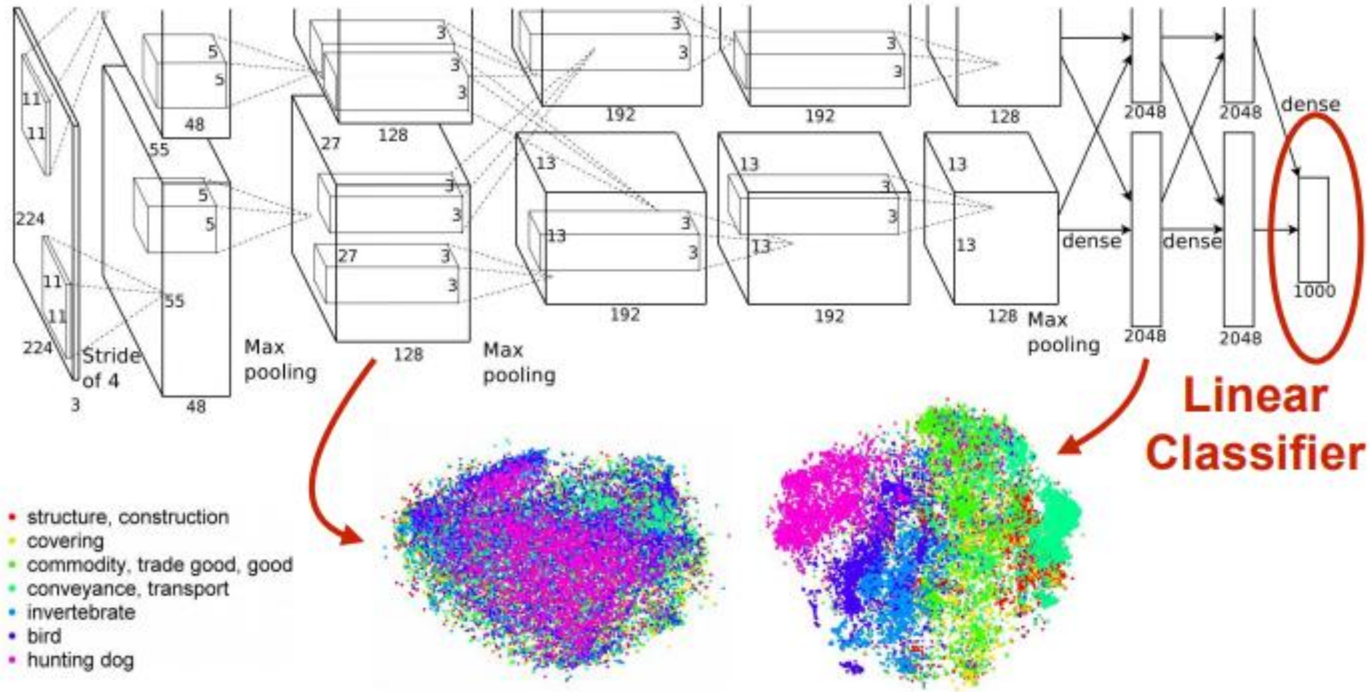


Last layer of most CNNs is a linear classifier



Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable

Visualizing AlexNet in 2D with t-SNE

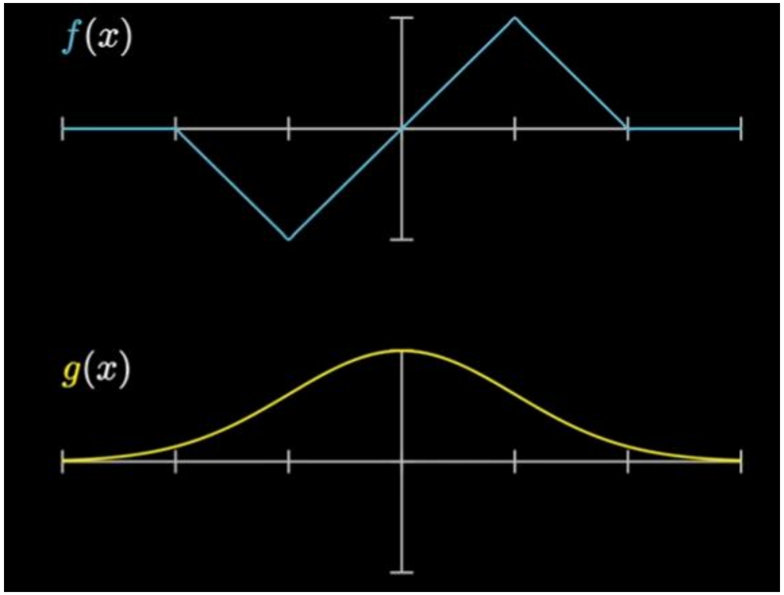


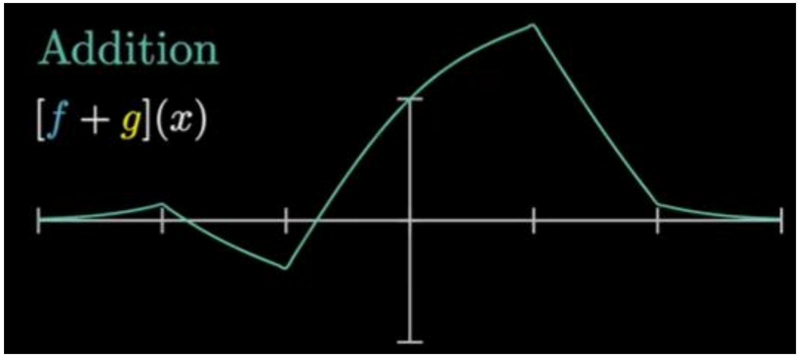
(c) DeCAF_1 (d) DeCAF_6

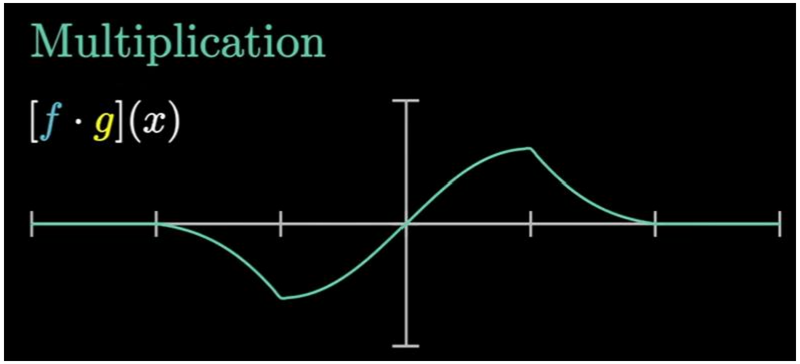
(2D visualization using t-SNE)

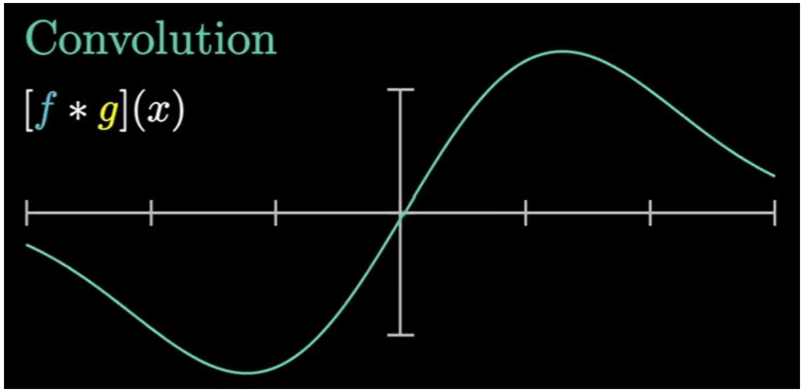
[Donahue, “DeCAF: DeCAF: A Deep Convolutional …”, arXiv 2013]

Convolutional operator



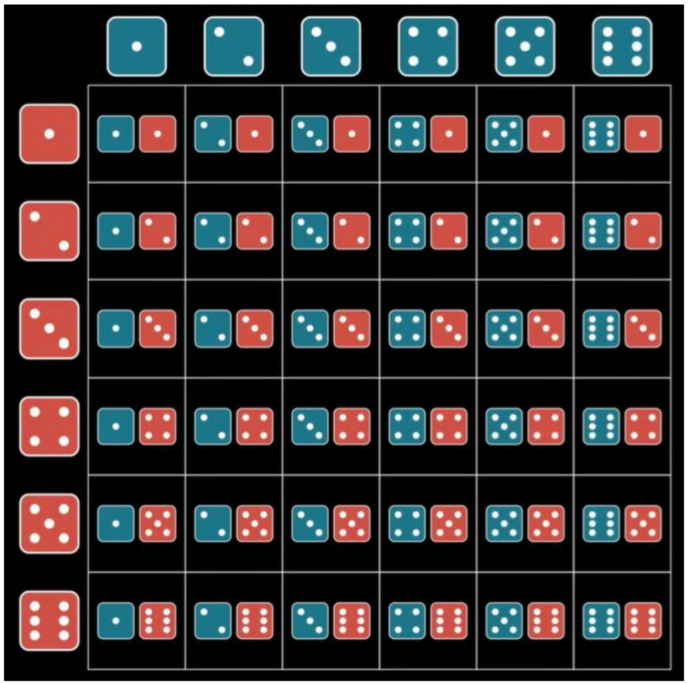




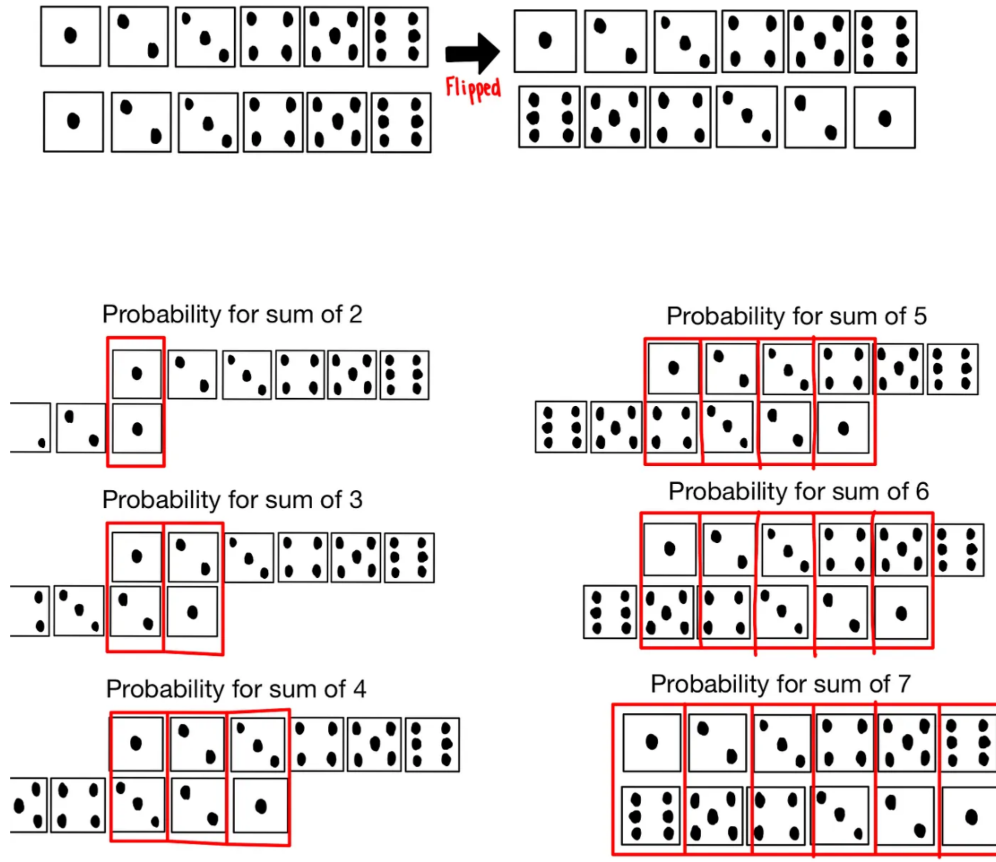


a = [1,2,3,4]
b = [5,6,7,8]
a+b = [6,8,10,12]
a·b = [5,12,21,32]
a*b = [5,16,34,60,61,52,32]

Convolution with dice



$6^2 = 36$ Combinations



Convolution of a_i and b_i

$$(a * b)_n = \sum_{\substack{i,j \\ i+j=n}} a_i \cdot b_i$$

	a_1	a_2	a_3	a_4	a_5	a_6
b_1	$a_1 \cdot b_1$	$a_2 \cdot b_1$	$a_3 \cdot b_1$	$a_4 \cdot b_1$	$a_5 \cdot b_1$	$a_6 \cdot b_1$
b_2	$a_1 \cdot b_2$	$a_2 \cdot b_2$	$a_3 \cdot b_2$	$a_4 \cdot b_2$	$a_5 \cdot b_2$	$a_6 \cdot b_2$
b_3	$a_1 \cdot b_3$	$a_2 \cdot b_3$	$a_3 \cdot b_3$	$a_4 \cdot b_3$	$a_5 \cdot b_3$	$a_6 \cdot b_3$
b_4	$a_1 \cdot b_4$	$a_2 \cdot b_4$	$a_3 \cdot b_4$	$a_4 \cdot b_4$	$a_5 \cdot b_4$	$a_6 \cdot b_4$
b_5	$a_1 \cdot b_5$	$a_2 \cdot b_5$	$a_3 \cdot b_5$	$a_4 \cdot b_5$	$a_5 \cdot b_5$	$a_6 \cdot b_5$
b_6	$a_1 \cdot b_6$	$a_2 \cdot b_6$	$a_3 \cdot b_6$	$a_4 \cdot b_6$	$a_5 \cdot b_6$	$a_6 \cdot b_6$

$$P(\blacksquare + \blacksquare = 2) = a_1 \cdot b_1$$

$$P(\blacksquare + \blacksquare = 3) = a_1 \cdot b_2 + a_2 \cdot b_1$$

$$P(\blacksquare + \blacksquare = 4) = a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1$$

$$P(\blacksquare + \blacksquare = 5) = a_1 \cdot b_4 + a_2 \cdot b_3 + a_3 \cdot b_2 + a_4 \cdot b_1$$

$$P(\blacksquare + \blacksquare = 6) = a_1 \cdot b_5 + a_2 \cdot b_4 + a_3 \cdot b_3 + a_4 \cdot b_2 + a_5 \cdot b_1$$

$$P(\blacksquare + \blacksquare = 7) = a_1 \cdot b_6 + a_2 \cdot b_5 + a_3 \cdot b_4 + a_4 \cdot b_3 + a_5 \cdot b_2 + a_6 \cdot b_1$$

$$P(\blacksquare + \blacksquare = 8) = a_2 \cdot b_6 + a_3 \cdot b_5 + a_4 \cdot b_4 + a_5 \cdot b_3 + a_6 \cdot b_2$$

$$P(\blacksquare + \blacksquare = 9) = a_3 \cdot b_6 + a_4 \cdot b_5 + a_5 \cdot b_4 + a_6 \cdot b_3$$

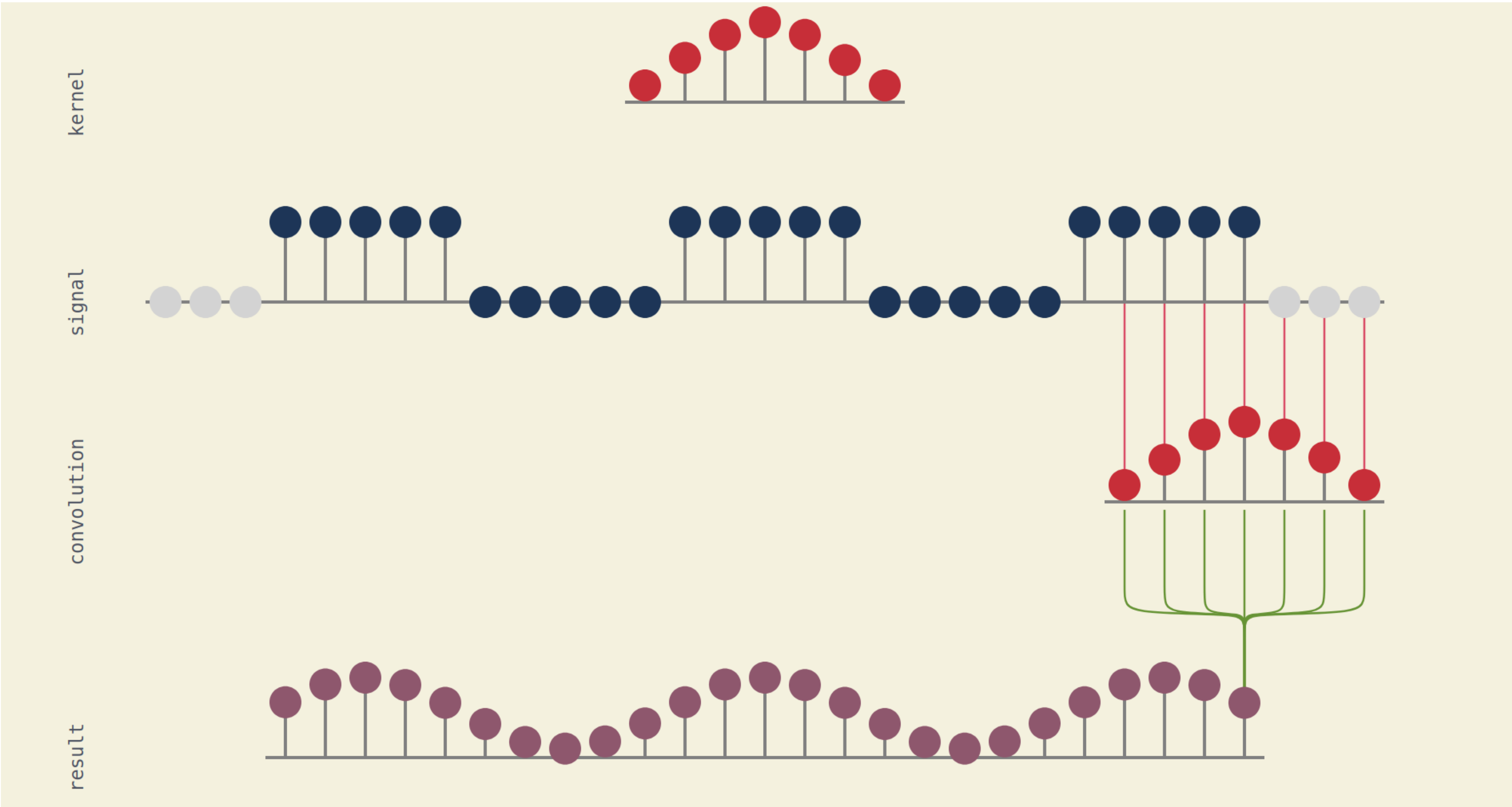
$$P(\blacksquare + \blacksquare = 10) = a_4 \cdot b_6 + a_5 \cdot b_5 + a_6 \cdot b_4$$

$$P(\blacksquare + \blacksquare = 11) = a_5 \cdot b_6 + a_6 \cdot b_5$$

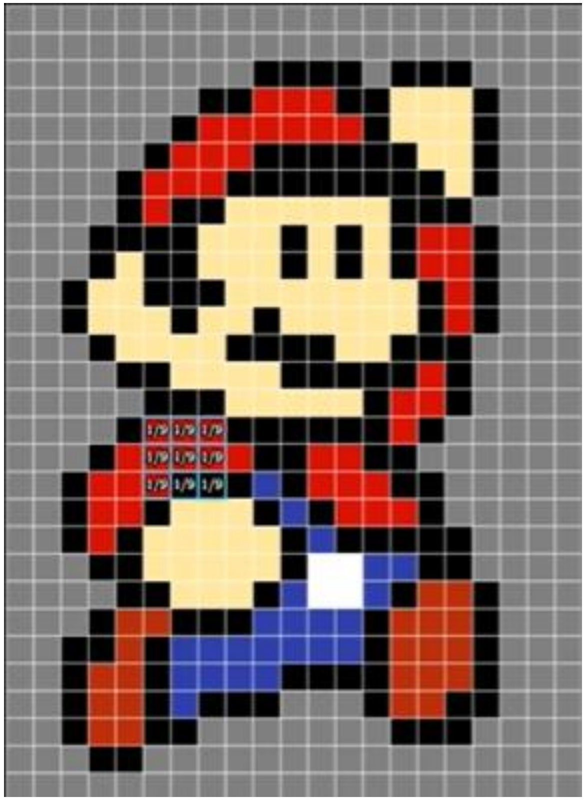
$$P(\blacksquare + \blacksquare = 12) = a_6 \cdot b_6$$

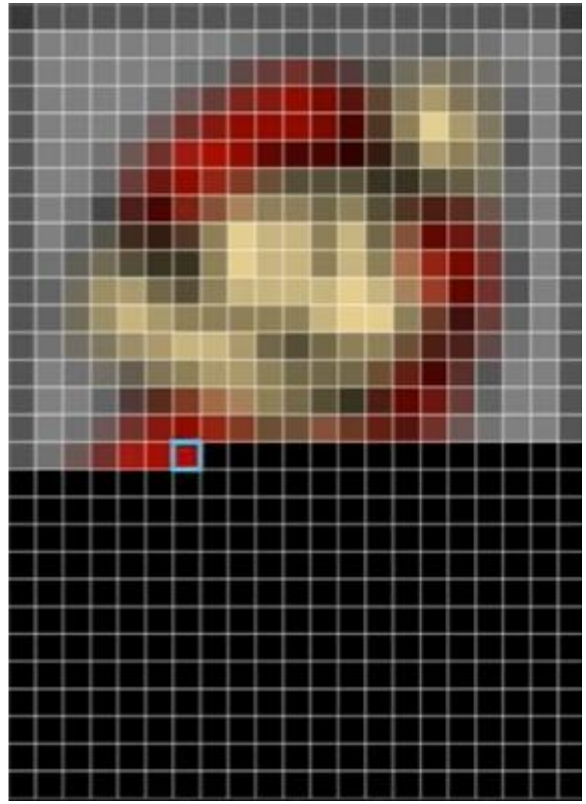
Example: (1,2,3) * (4,5,6)

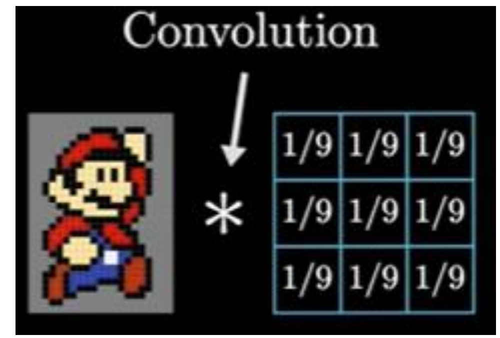
Example: Moving average



Example: 2D



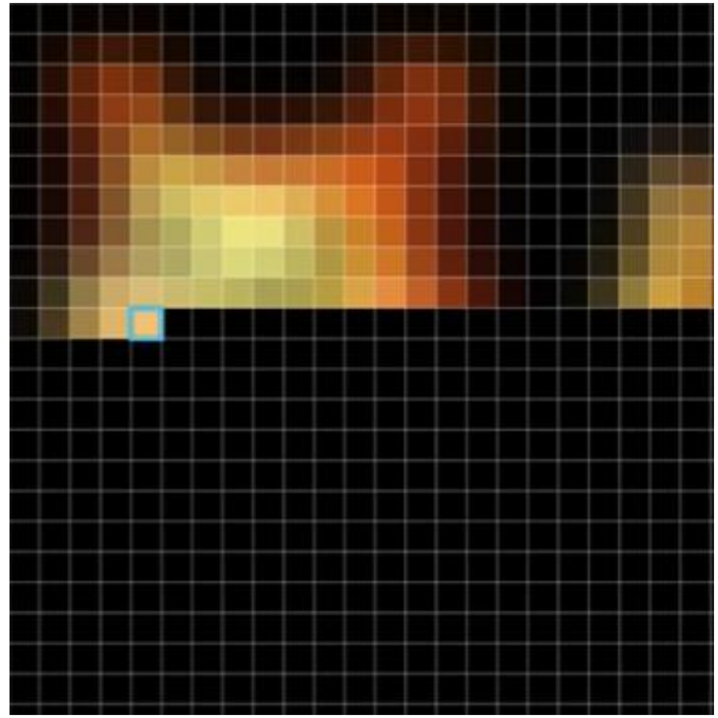




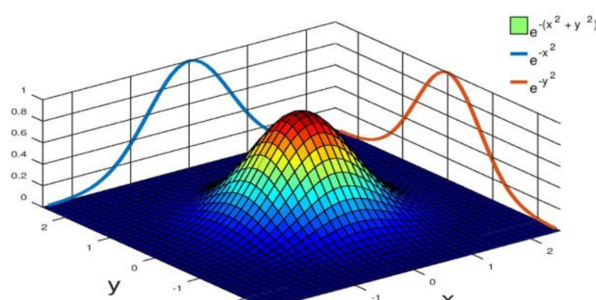
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$$\frac{1}{9}\begin{bmatrix}0.5\\0.5\\0.5\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.8\\0.0\\0.0\end{bmatrix}+\frac{1}{9}\begin{bmatrix}0.0\\0.0\\0.0\end{bmatrix}$$

Example: 2D



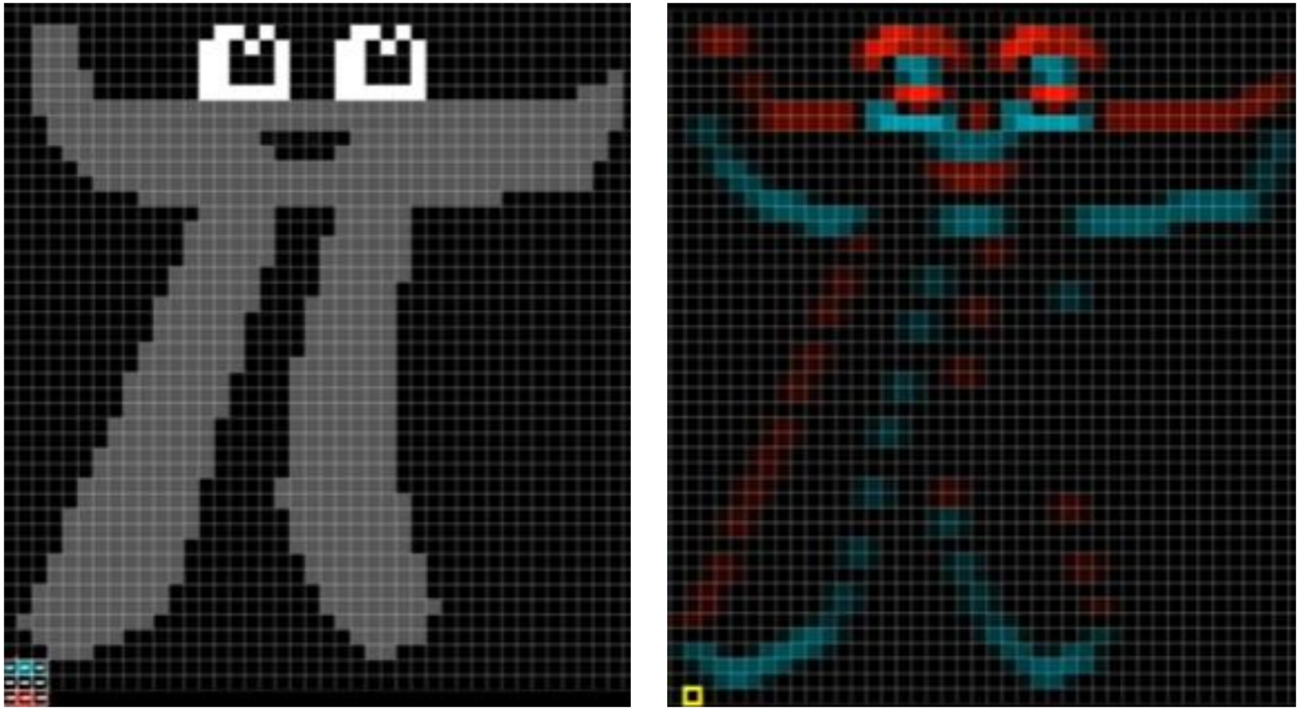
0.003	0.013	0.022	0.013	0.003
0.013	0.060	0.098	0.060	0.013
0.022	0.098	0.162	0.098	0.022
0.013	0.060	0.098	0.060	0.013
0.003	0.013	0.022	0.013	0.003



Example: 2D

kernel

0.25	0.00	−0.25
0.50	0.00	−0.50
0.25	0.00	−0.25



Example: (3,1,4,1,5,9) * (7,7,5)

Designed kernels vs. Learned kernels

- Classical signal processing:
  - mean kernel → spatial smoothing
  - Gaussian kernel → physically meaningful smoothing
  - Zero-sum kernel → edge detection

- However, real-world tasks are different:
  - Typhoon structure detection
  - Atmospheric pattern identification
  - Temperature field prediction

What is the correct kernel for a given task?

CNN: Learning the Convolutional operator

- Any linear and shift-invariant operator can be written as a convolution:

$$T(f) = f * w,$$

  where w is the kernel.

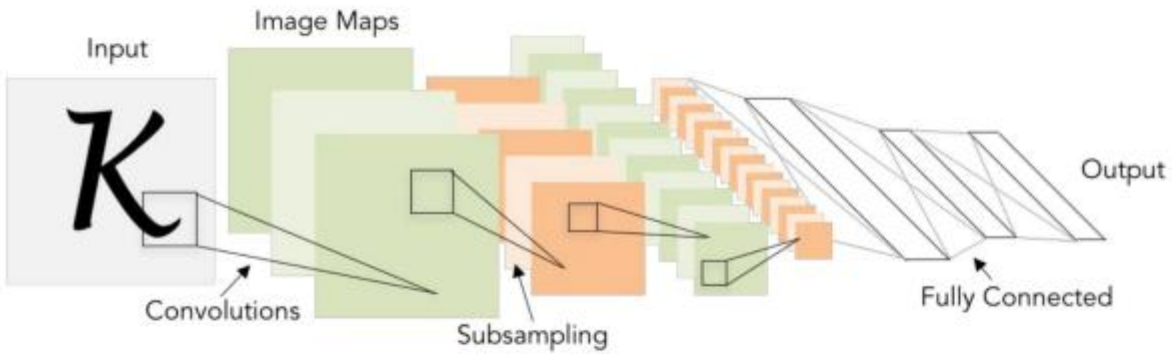
- In a convolutional neural network:
  - A *convolution layer* implements a linear and shift-invariant operator

$$(a * b)[n] = \sum a[k]b[n-k]$$

  - The kernel w is treated as *learnable parameters*
  - Training *optimizes the kernel* for the task

Convolutional neural networks

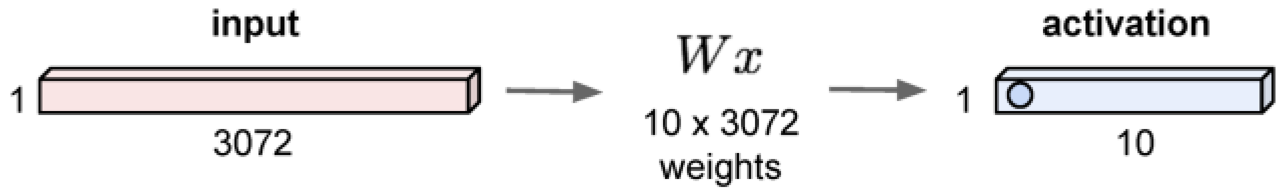
- Layer types:
  - Fully-connected layer
  - Convolutional layer
  - Pooling layer



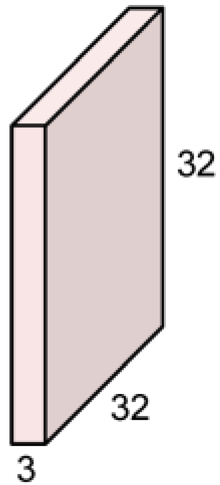
LeNet-5

Fully Connected Layer & Convolution Layer

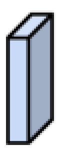
32x32x3 image -> stretch to 3072 x 1



32x32x3 image



5x5x3 filter



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

MLP vs. Convolution for gridded satellite data

KIAPS

- Fully connected (MLP):

$$y = Wx$$

  - Each spatial location has independent weights
  - Nor shift-invariant and ignores locality

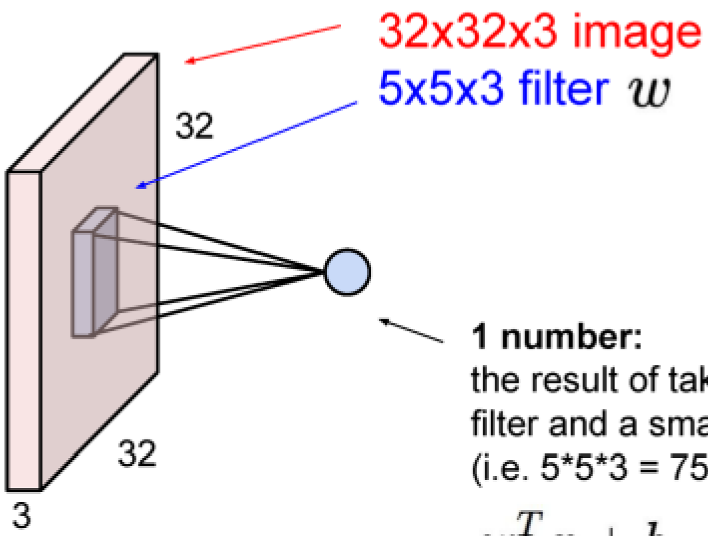
- Convolution:

$$(T * K)(x, y) = \sum_{i,j} T(i, j)K(x - i, y - j)$$

  - Same kernel applied at every location (weight sharing)
  - Each output is a weighted sum of neighboring values

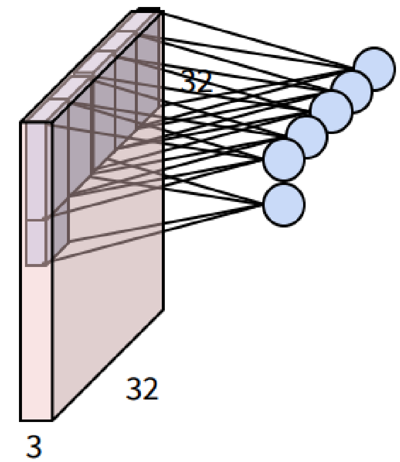
For spatially continuous satellite fields, convolution preserves structures.

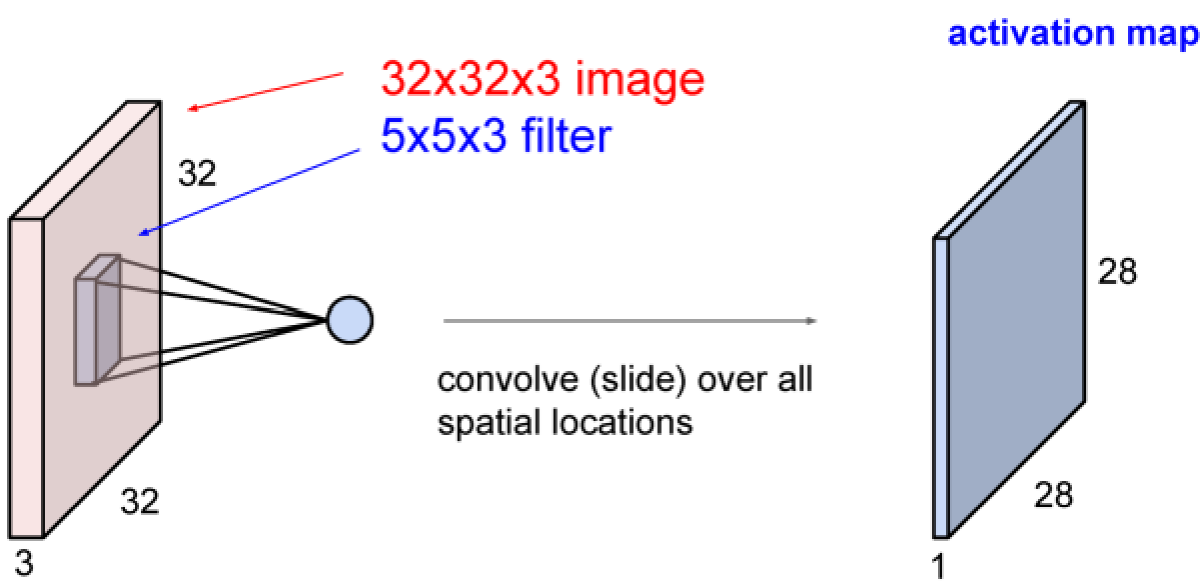
Convolution Layer



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)

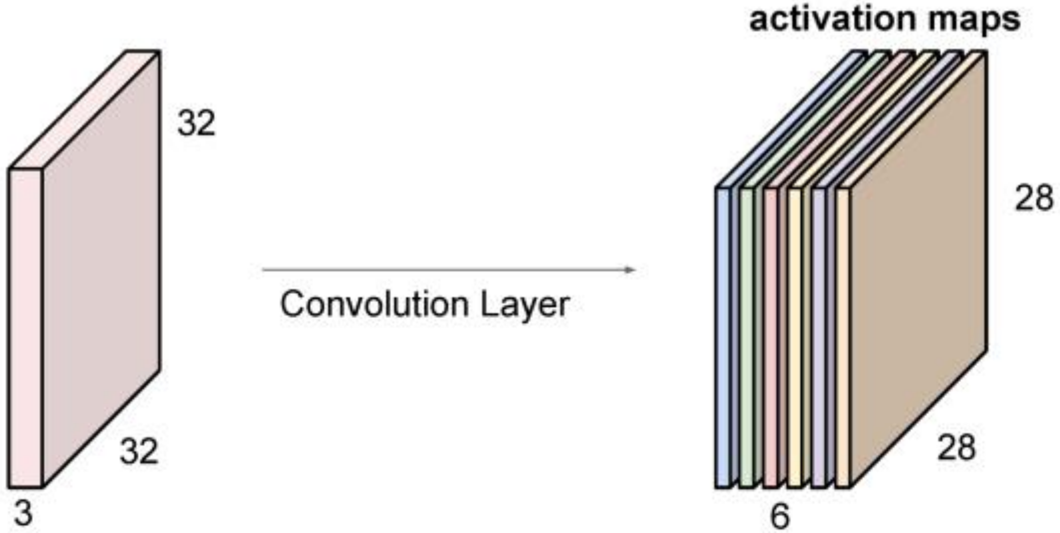
$$w^T x + b$$





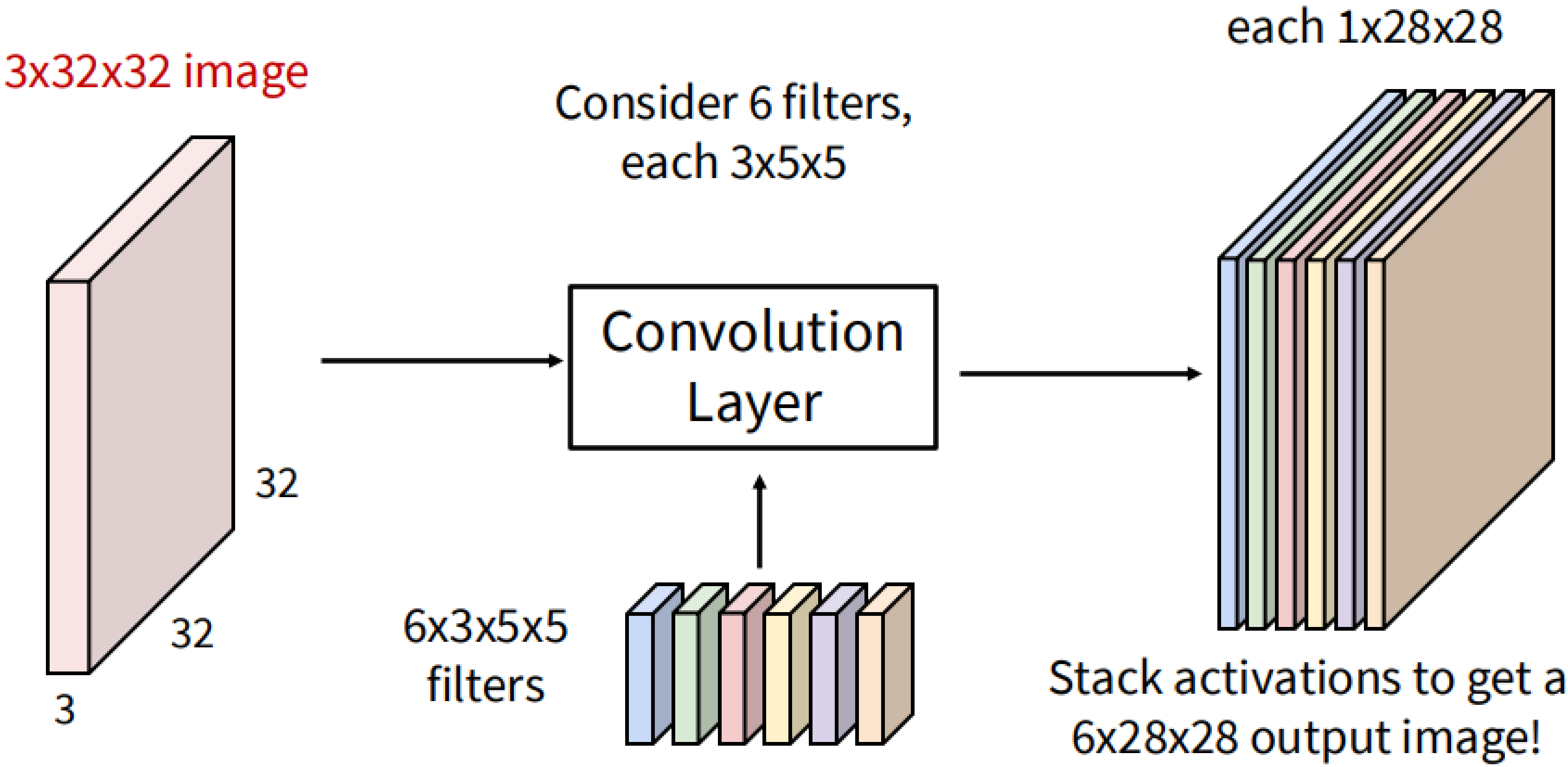
Convolution Layer

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

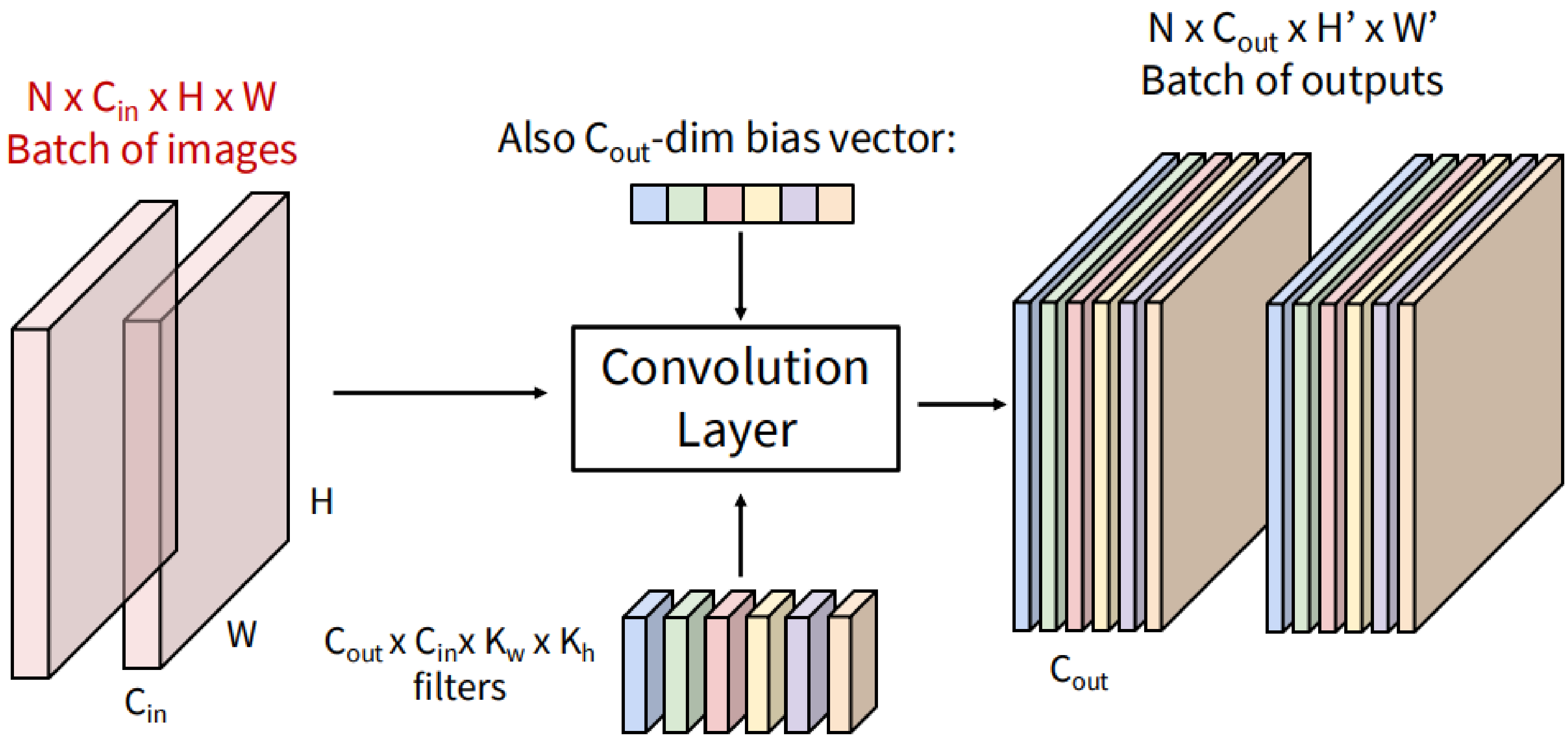


We stack these up to get a "new image" of size 28x28x6!

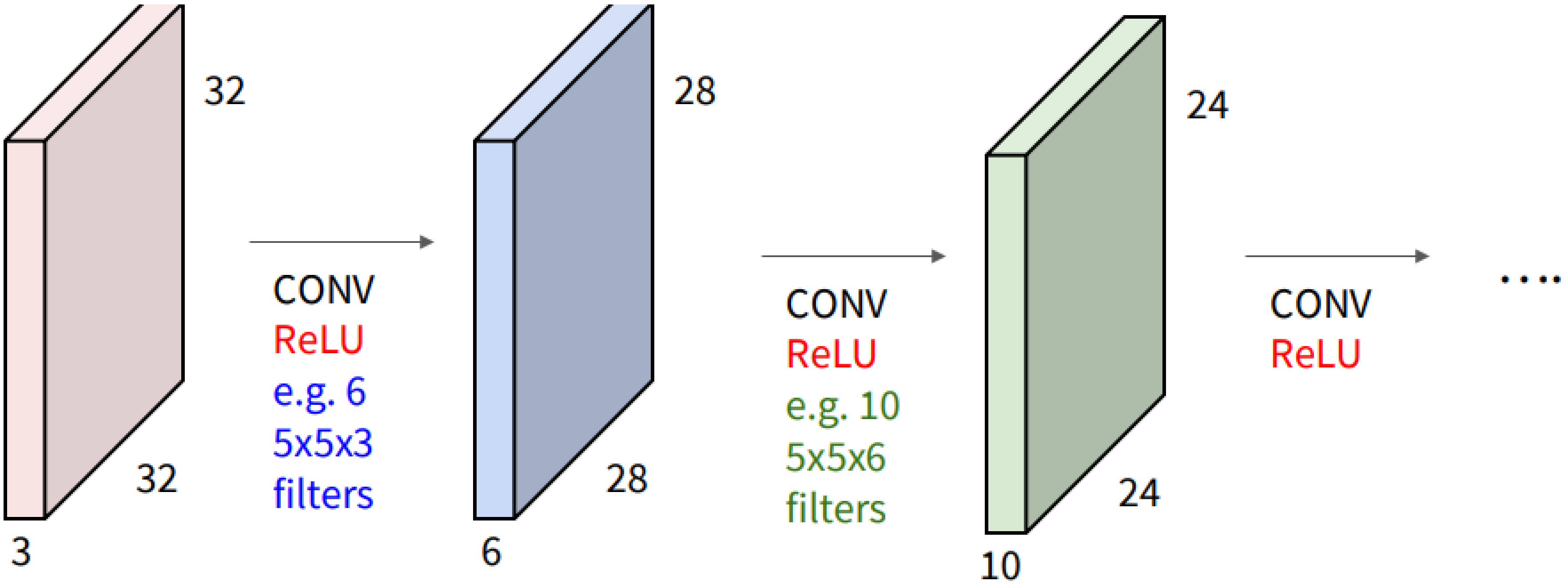
Convolution Layer



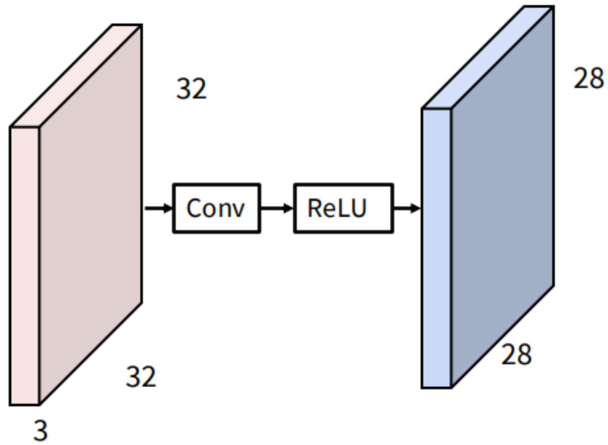
Convolution Layer



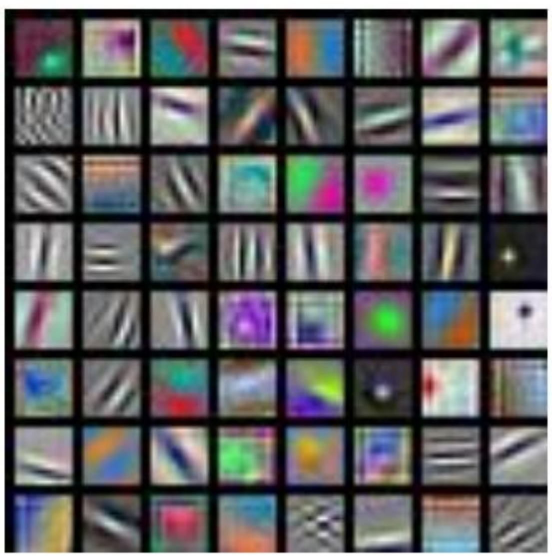
Convolution Layer



What do Conv filters learn?

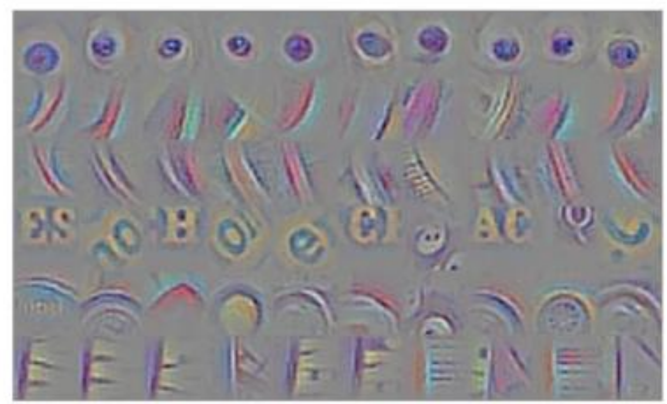


First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11

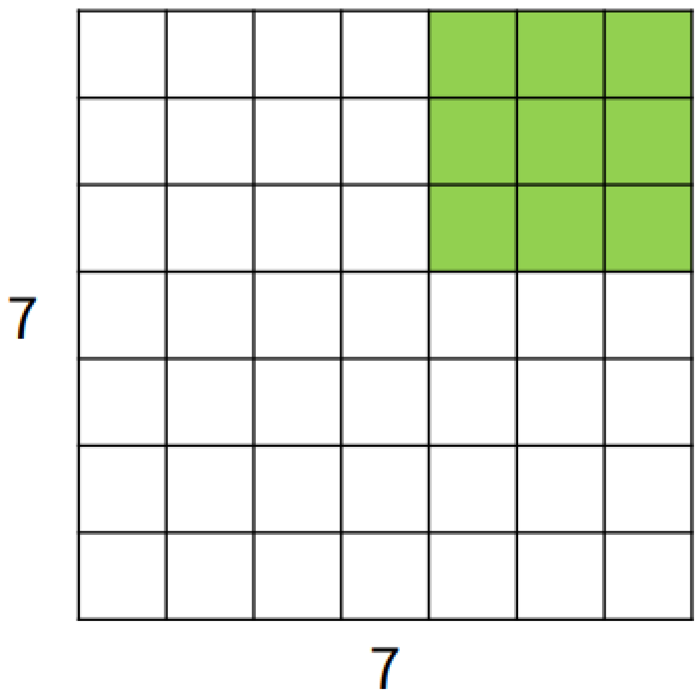
Deeper conv layers: Harder to visualize
Tend to learn larger structures e.g. eyes, letters



6th layer conv layer from an ImageNet model

Visualization from [Springenberg et al, ICLR 2015]

Convolution: Spatial Dimensions



Input: 7x7
Filter: 3x3
Output: 5x5

<u>Problem</u>: Feature maps shrink with each layer!

In general
Input: W
Filter: K
Output: W – K + 1

Convolution: Spatial Dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input: 7x7
Filter: 3x3
Output: 5x5

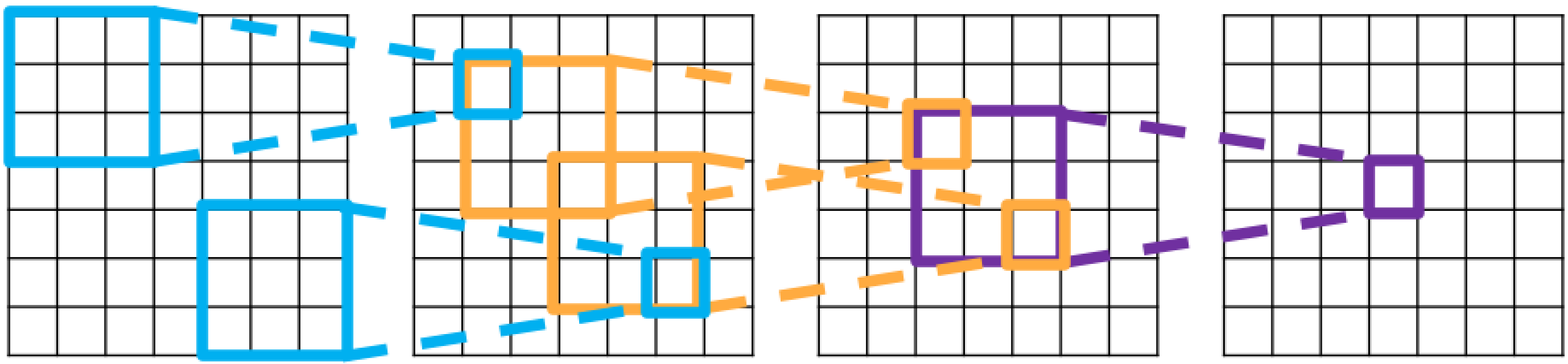
Problem: Feature maps shrink with each layer!

Solution: Add **padding** around the input before sliding the filter

In general
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

Receptive fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Input

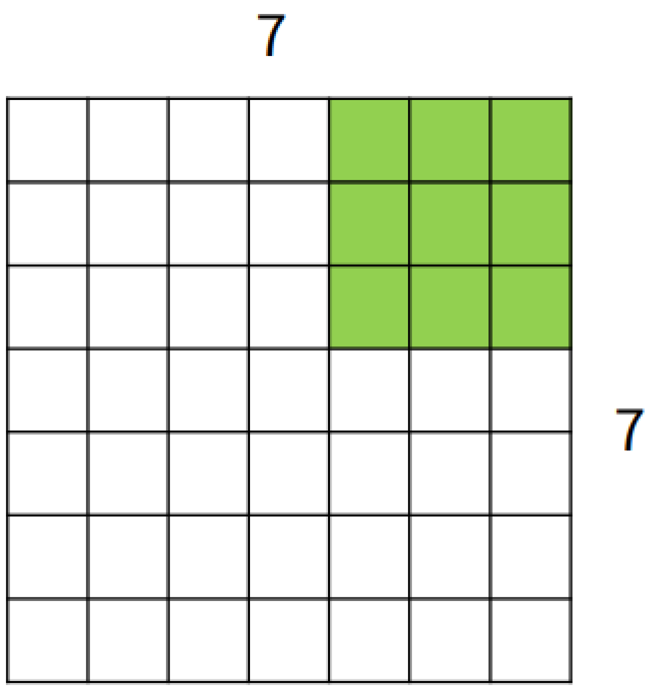
Output

Problem: For large images we need many layers for each output to "see" the whole image image

Solution: Downsample inside the network

Slide inspiration: Justin Johnson

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2
Output: 3x3

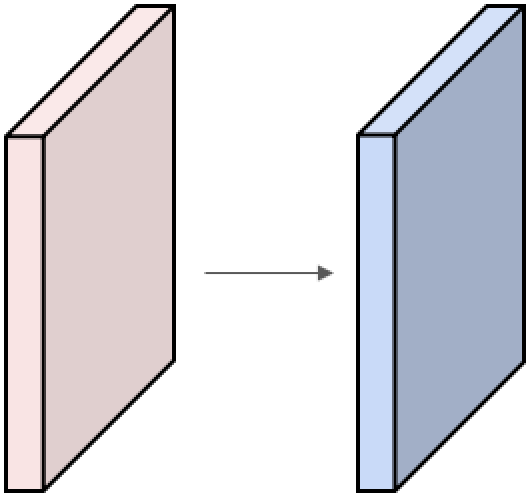
In general:
Input: W
Filter: K
Padding: P
Stride: S

→ Output:
$(W - K + 2P) / S + 1$

Convolution example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: ?



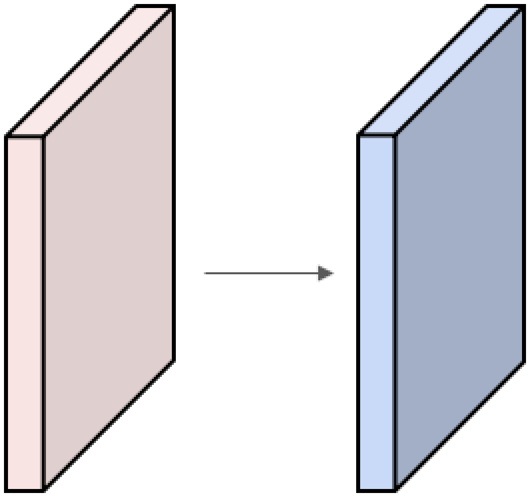
Convolution example

Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32

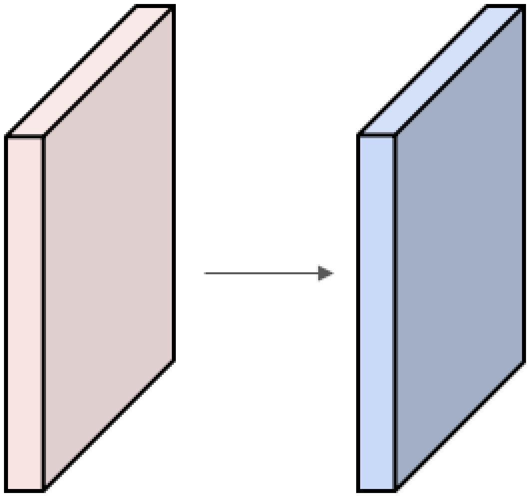
32 = (32+2*2-5)/1+1



Convolution example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: ?

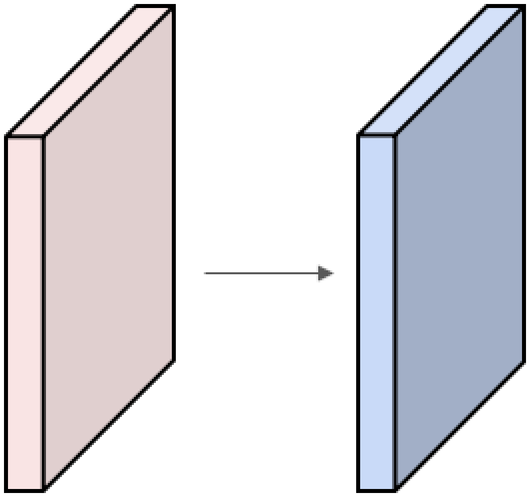


Number of learnable parameters: 760
Parameters per filter: 3*5*5 + 1 (for bias) = 76
10 filters, so total is 10 * 76 = **760**

Convolution example

Input volume: **3** x 32 x 32
10 **5x5** filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations?



Number of learnable parameters: 760
Number of multiply-add operations: **768,000**
10*32*32 = 10,240 outputs
Each output is the inner product of two **3**x**5x5** tensors (75 elems)
Total = 75*10240 = **768K**

PyTorch Convolution Layer

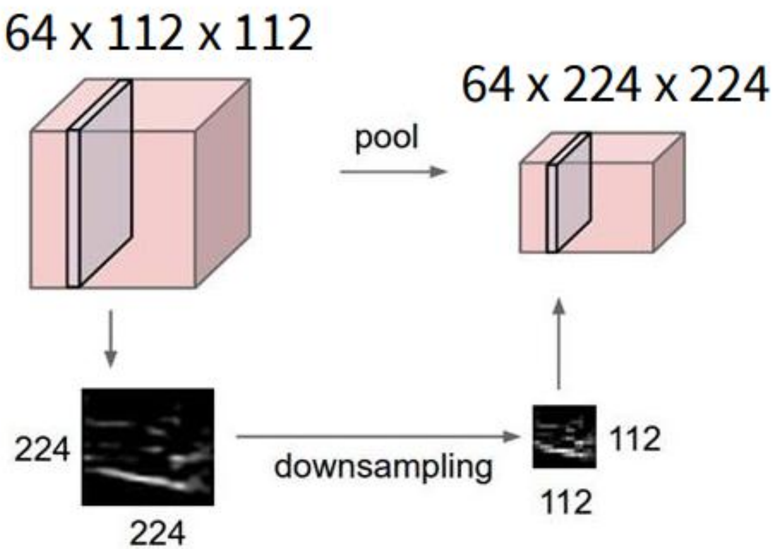
CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

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:

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

Pooling layer



Given an input C x H x W,
downsample each 1 x H x W plane

Hyperparameters:
Kernel Size
Stride
Pooling function

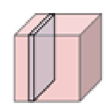
Single depth slice

1	1	2	4
5	**6**	7	**8**
3	2	1	0
1	2	3	**4**

x

y

64 x 224 x 224

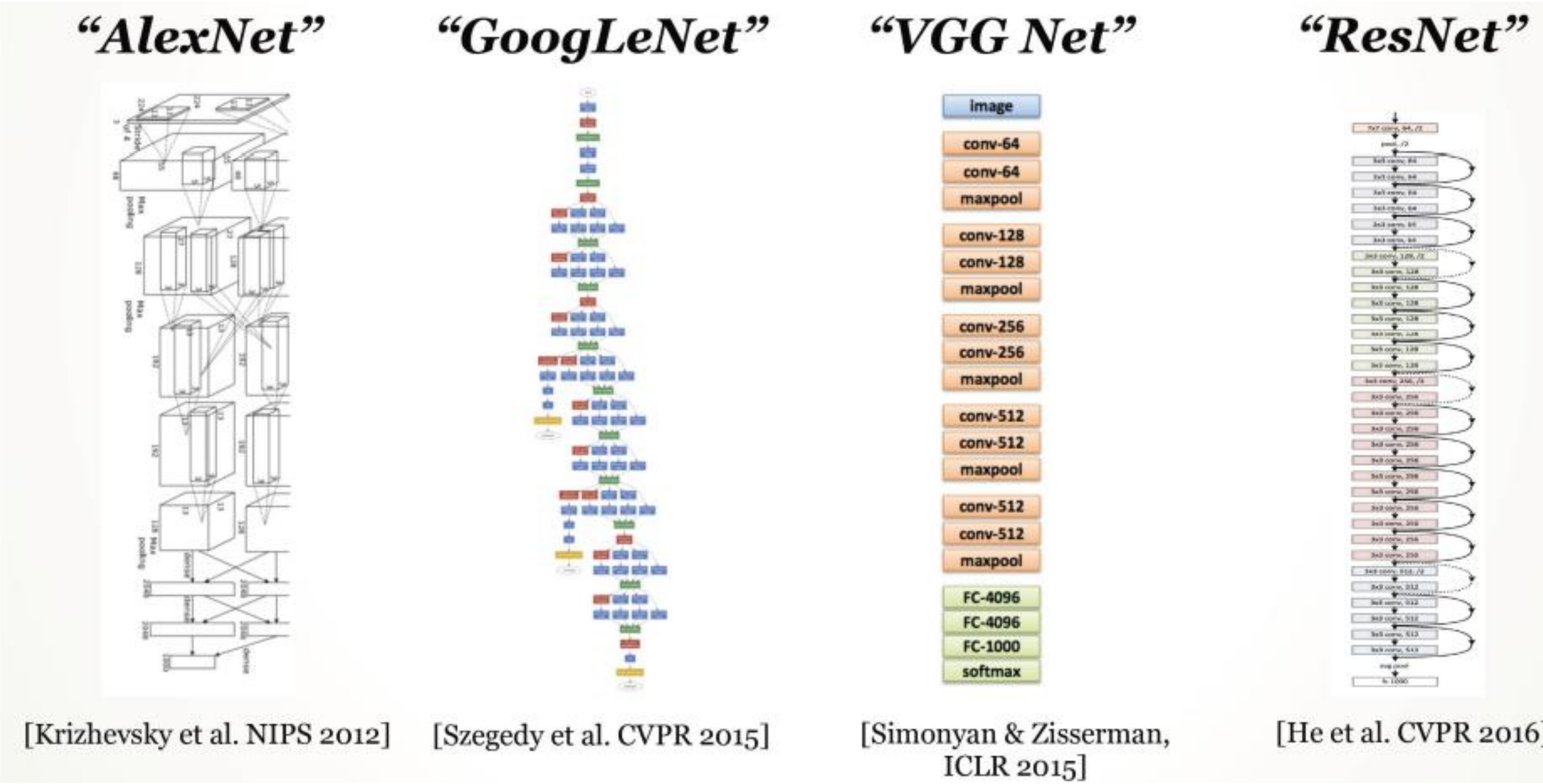


Max pooling with 2x2
kernel size and stride 2

6	8
3	4

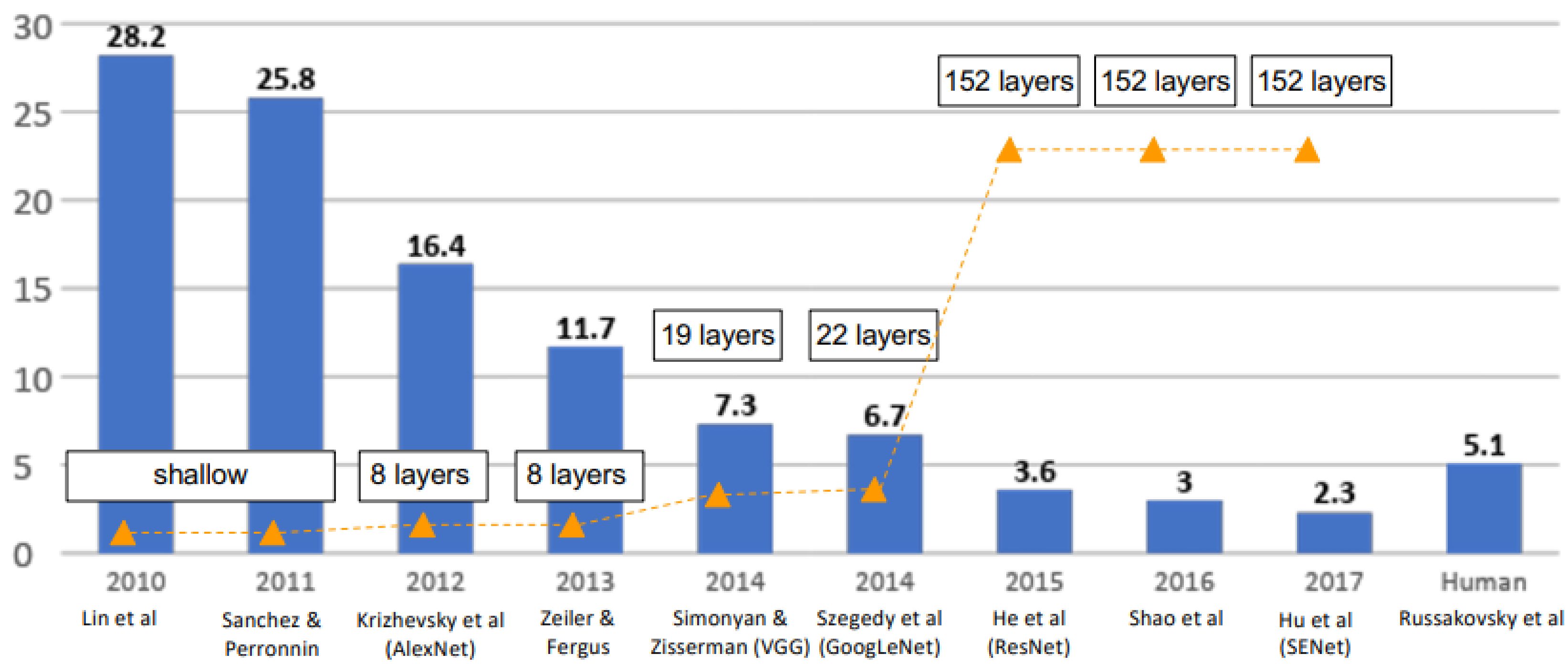
Gives **invariance** to small spatial
shifts. No learnable parameters.

CNN Architectures



CNN Architectures

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Layer Normalization

High-level Idea: Learn parameters that let us scale / shift the input data

1. Normalize input data
2. Scale / shift using learned parameters

$$x: N \times D$$

Normalize

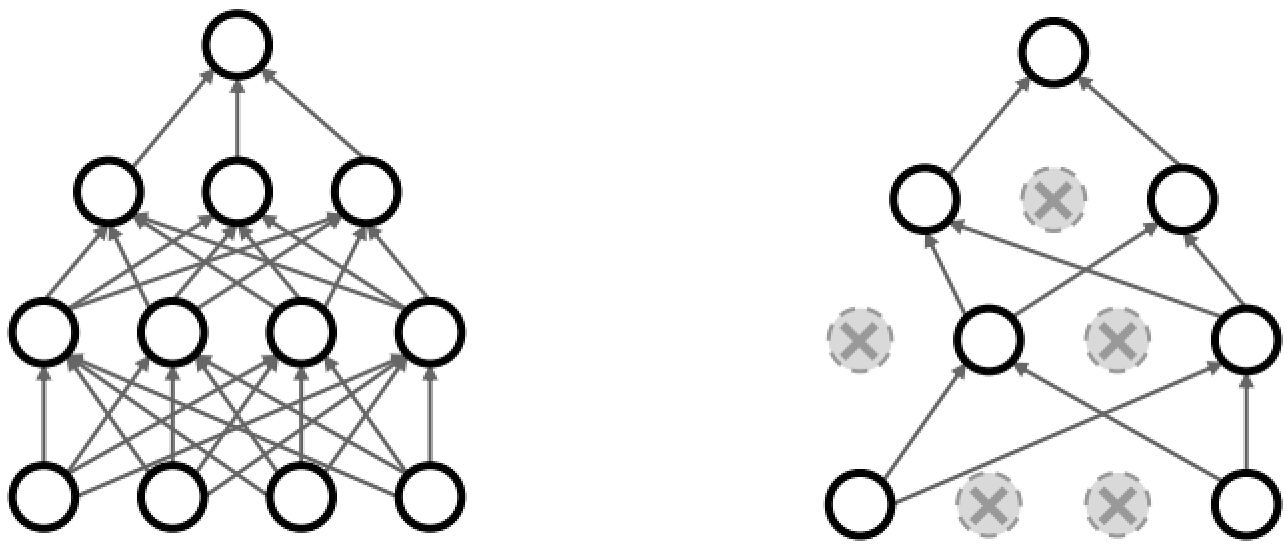
Statistics calculated per batch → $\mu, \sigma: N \times 1$

Learned parameters applied to each sample → $\gamma, \beta: 1 \times D$

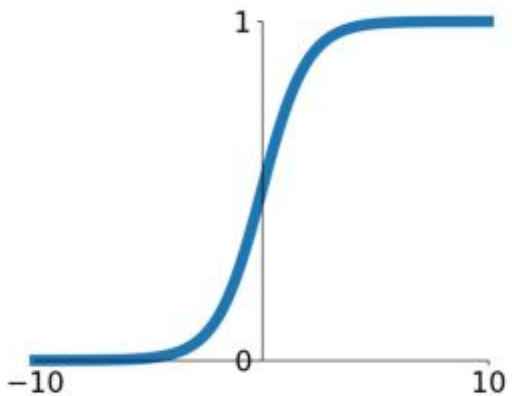
$$y = \gamma(x-\mu)/\sigma+\beta$$

Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Activation Functions



Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

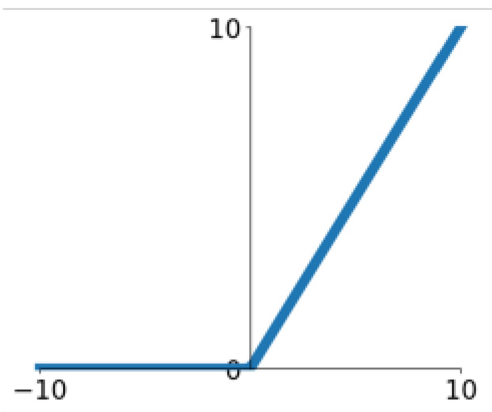
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Key problem:

Many layers of sigmoids → smaller and smaller gradients.

Q: In which regions does sigmoid have a small gradient?

Activation Functions

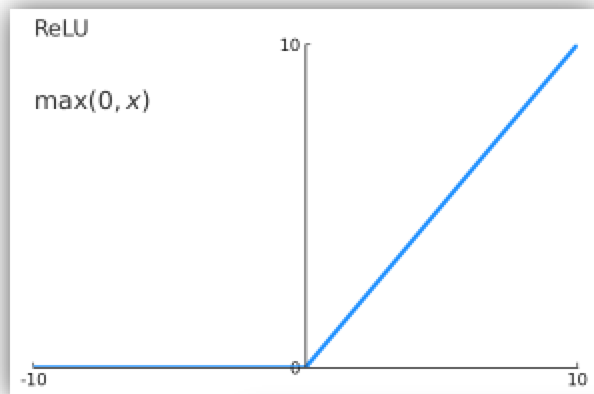


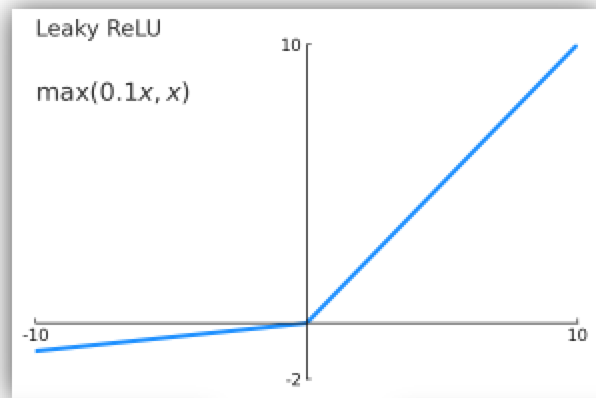
ReLU
(Rectified Linear Unit)

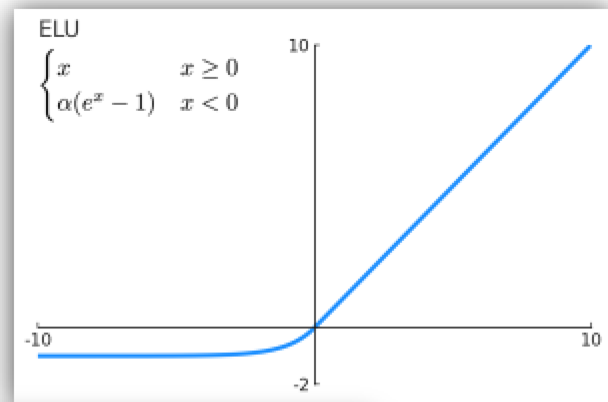
- Computes **f(x) = max(0,x)**

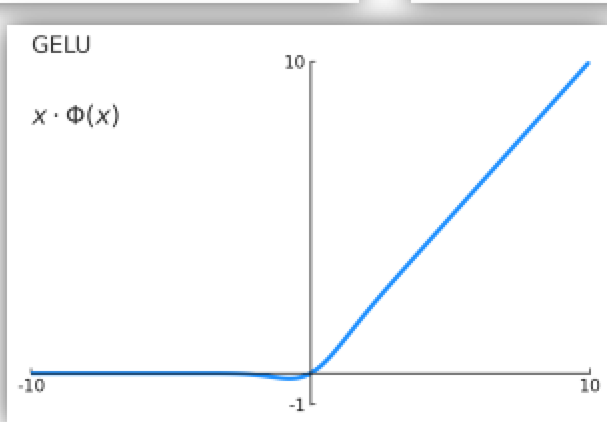
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice (e.g. 6x)

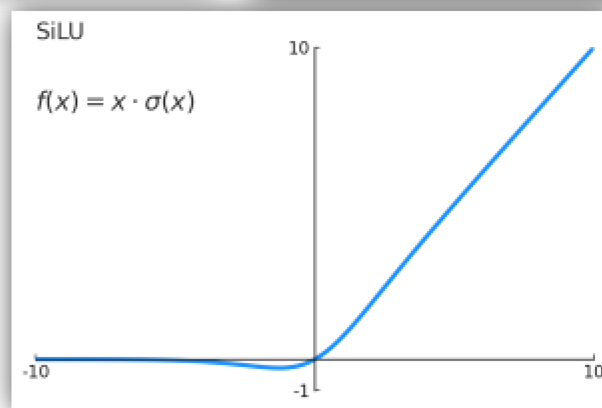
Activation Functions











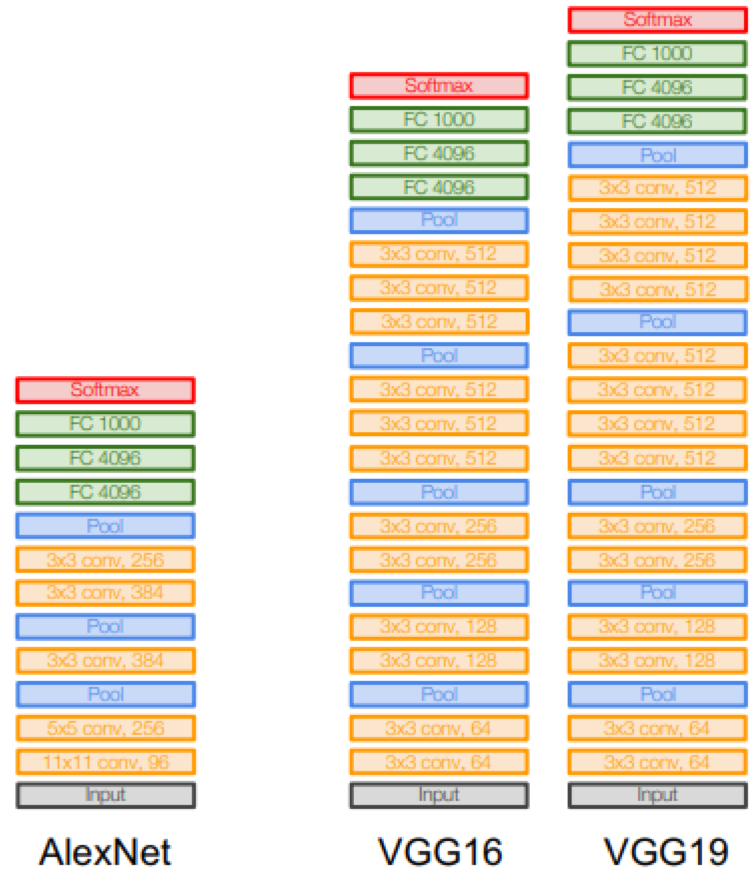
Case study: VGGNet

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

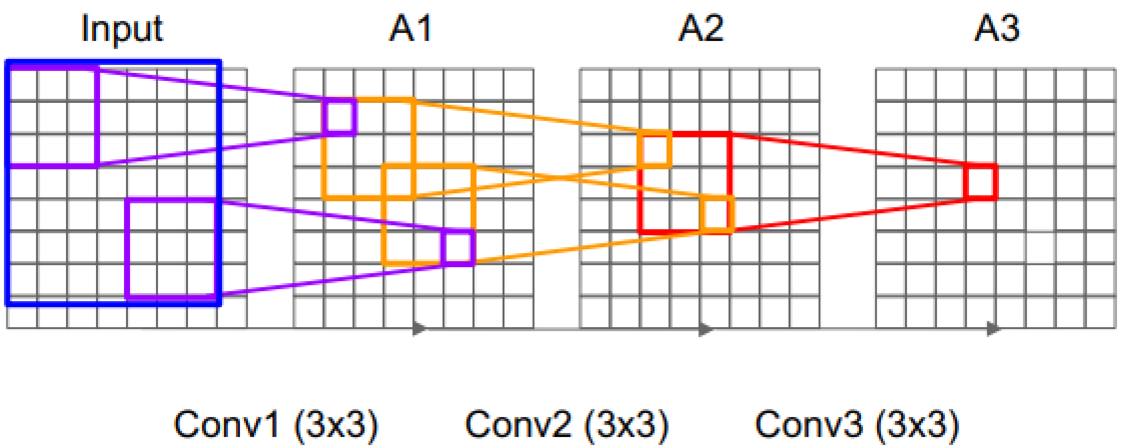
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



Case study: VGGNet

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

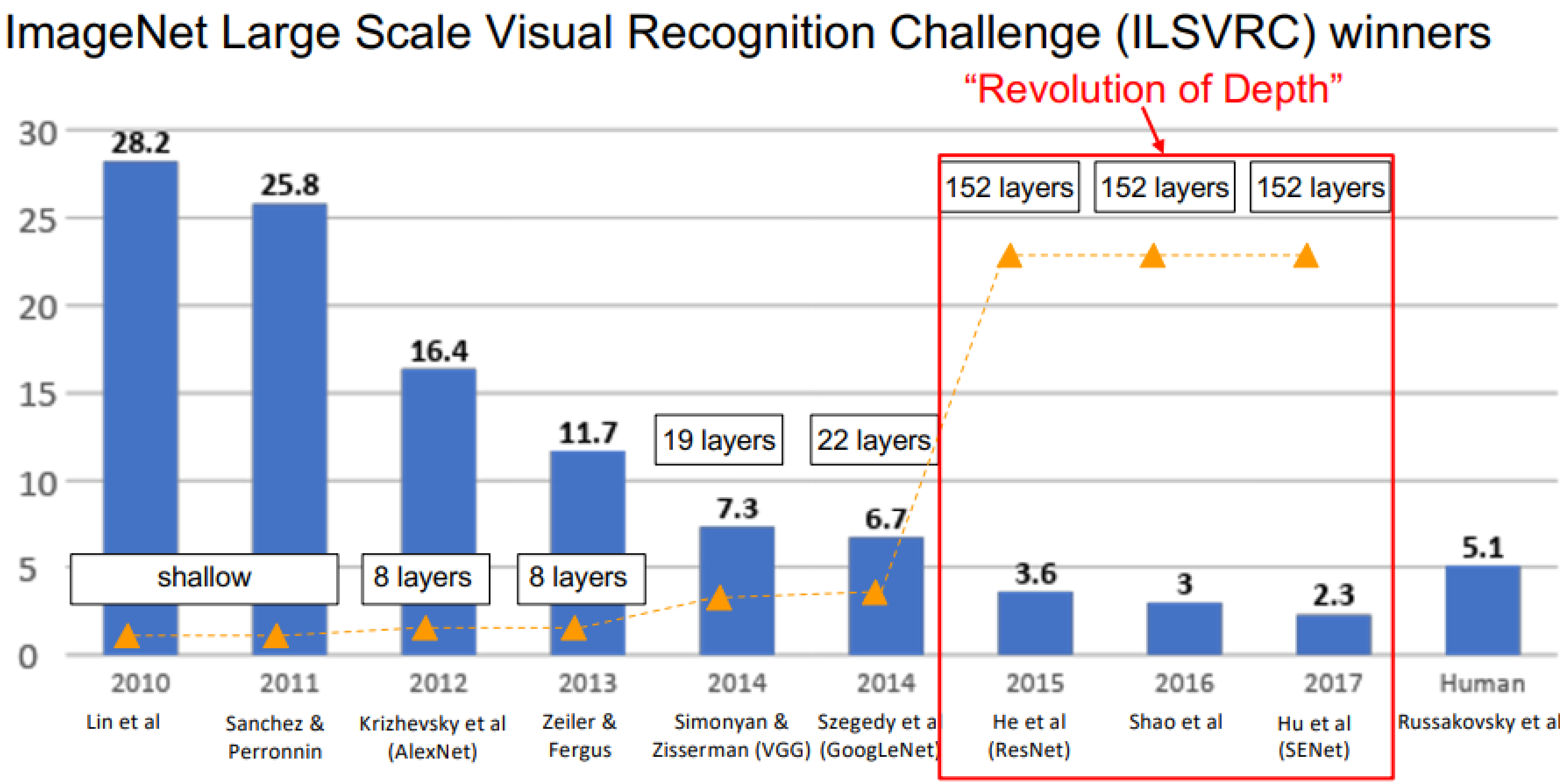


Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

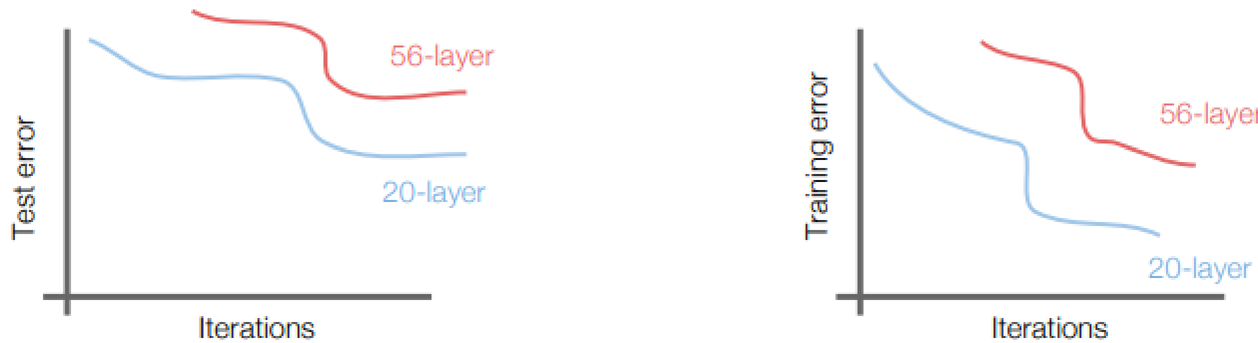
And fewer parameters: 3 * (3^2C^2) vs. 7^2C^2 for C channels per layer

Case Study: ResNet



Case Study: ResNet

What happens when we continue stacking deeper layers on a "plain" convolutional neural network?



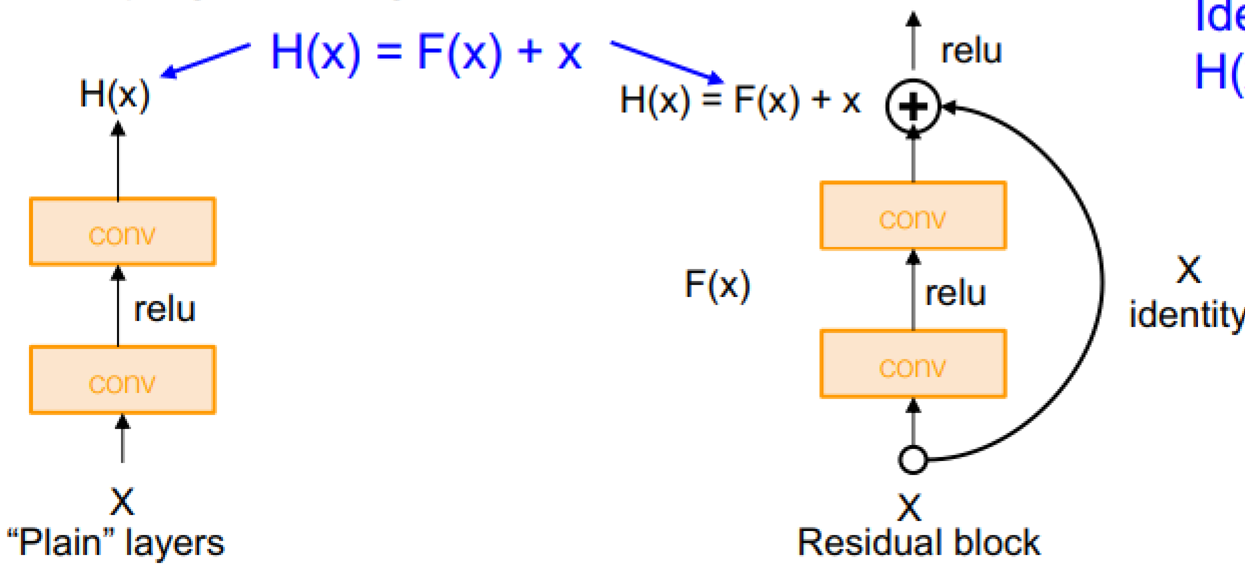
56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it's not caused by overfitting!

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

Case Study: ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



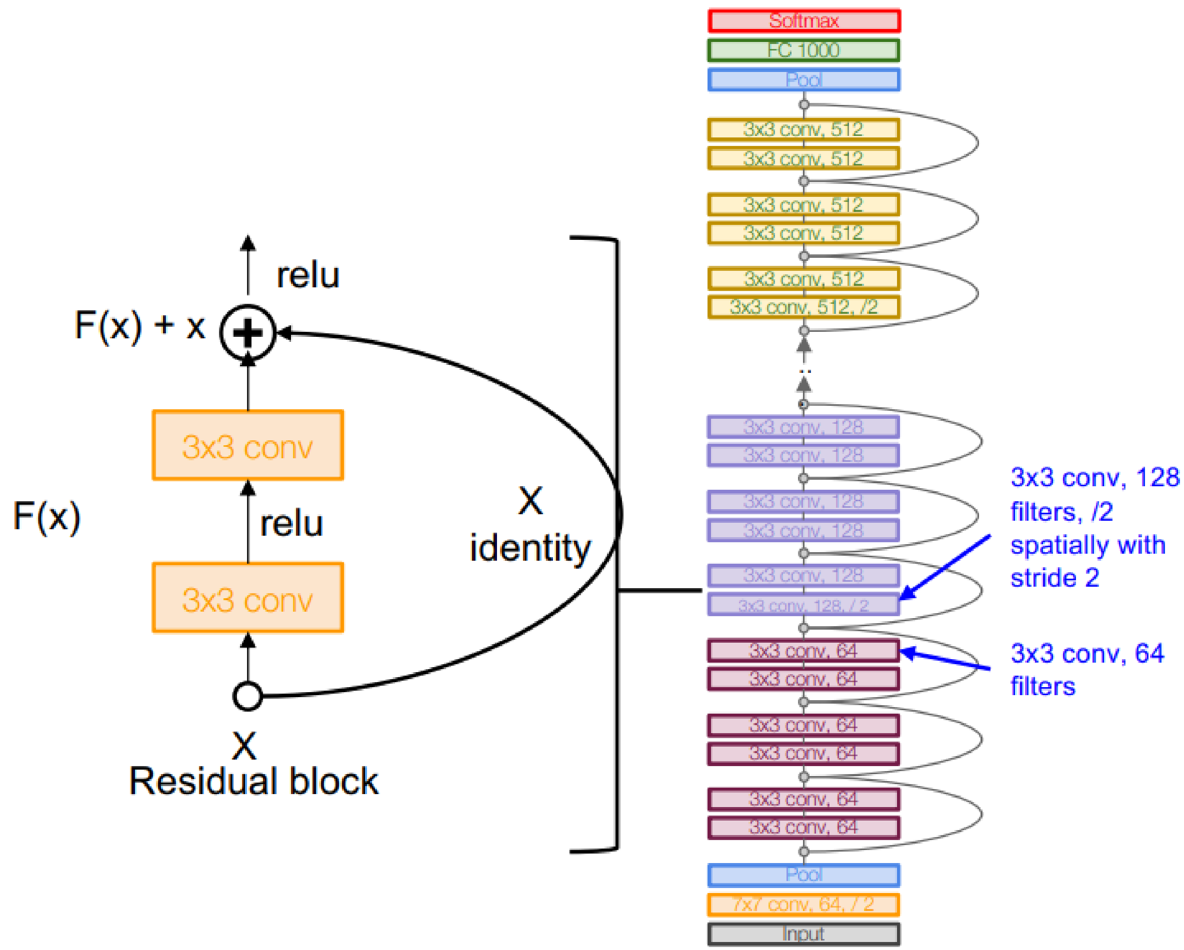
Identity mapping:
$H(x) = x$ if $F(x) = 0$

Use layers to fit **residual** $F(x) = H(x) - x$ instead of $H(x)$ directly

Case Study: ResNet

Full ResNet architecture:

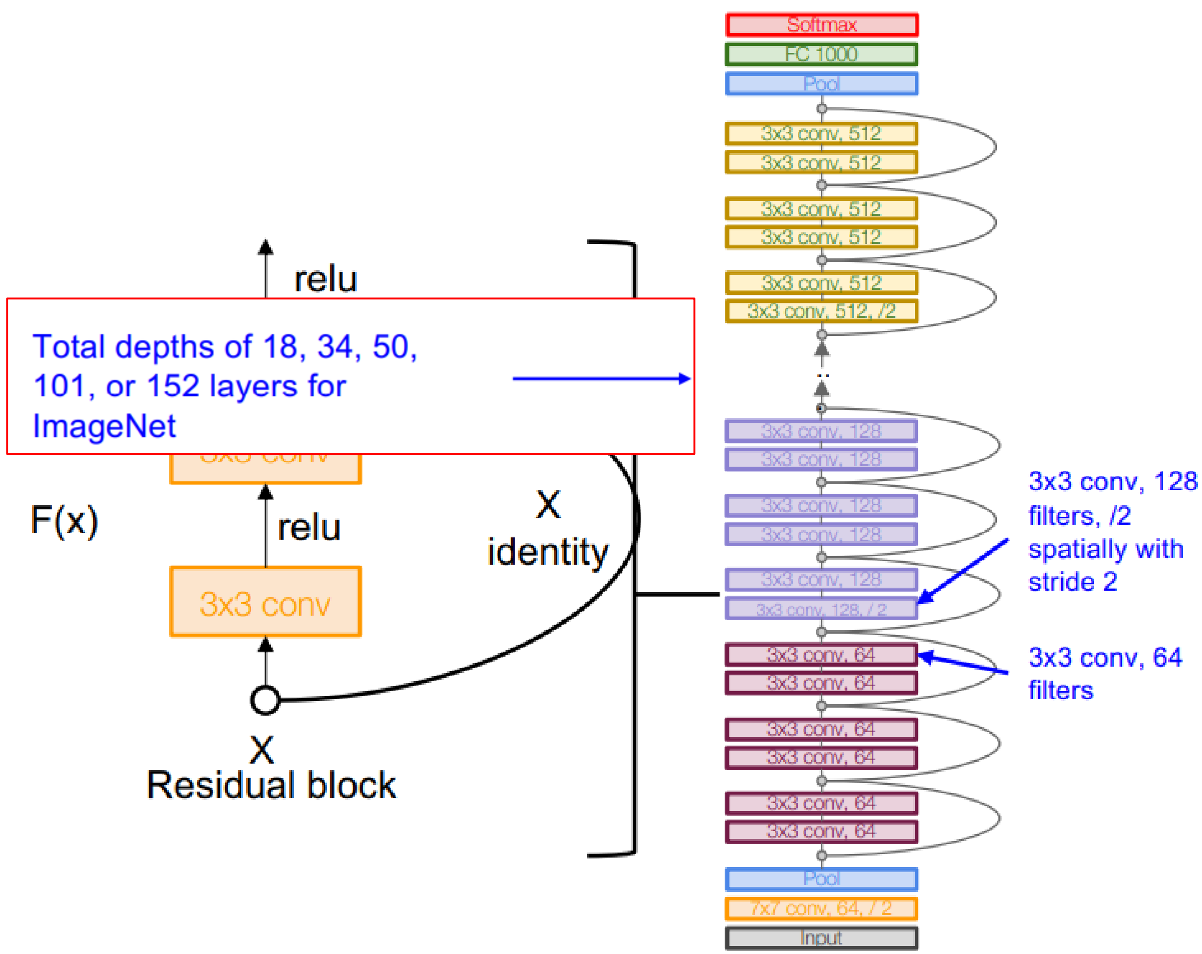
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension) Reduce the activation volume by half.



Case Study: ResNet

KIAPS

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension) Reduce the activation volume by half.



Q&A

Thank you for listening ☺