CSC 580 Principles of Machine Learning

14 Convolutional neural networks (CNN)

Chicheng Zhang

Department of Computer Science

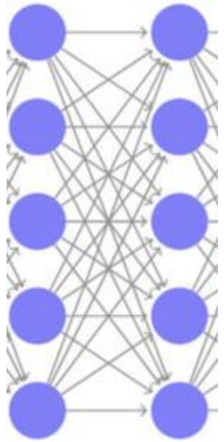
THE UNIVERSITY OF ARIZONA

*slides credit: built upon CSC 580 Fall 2021 lecture slides by Kwang-Sung Jun

NNs for images

- Fully-connected (FC) layers do not scale well to images (width x height x #channels)
  - Need for smaller number of parameters

$a^{(l)}$



$a^{(l+1)}$

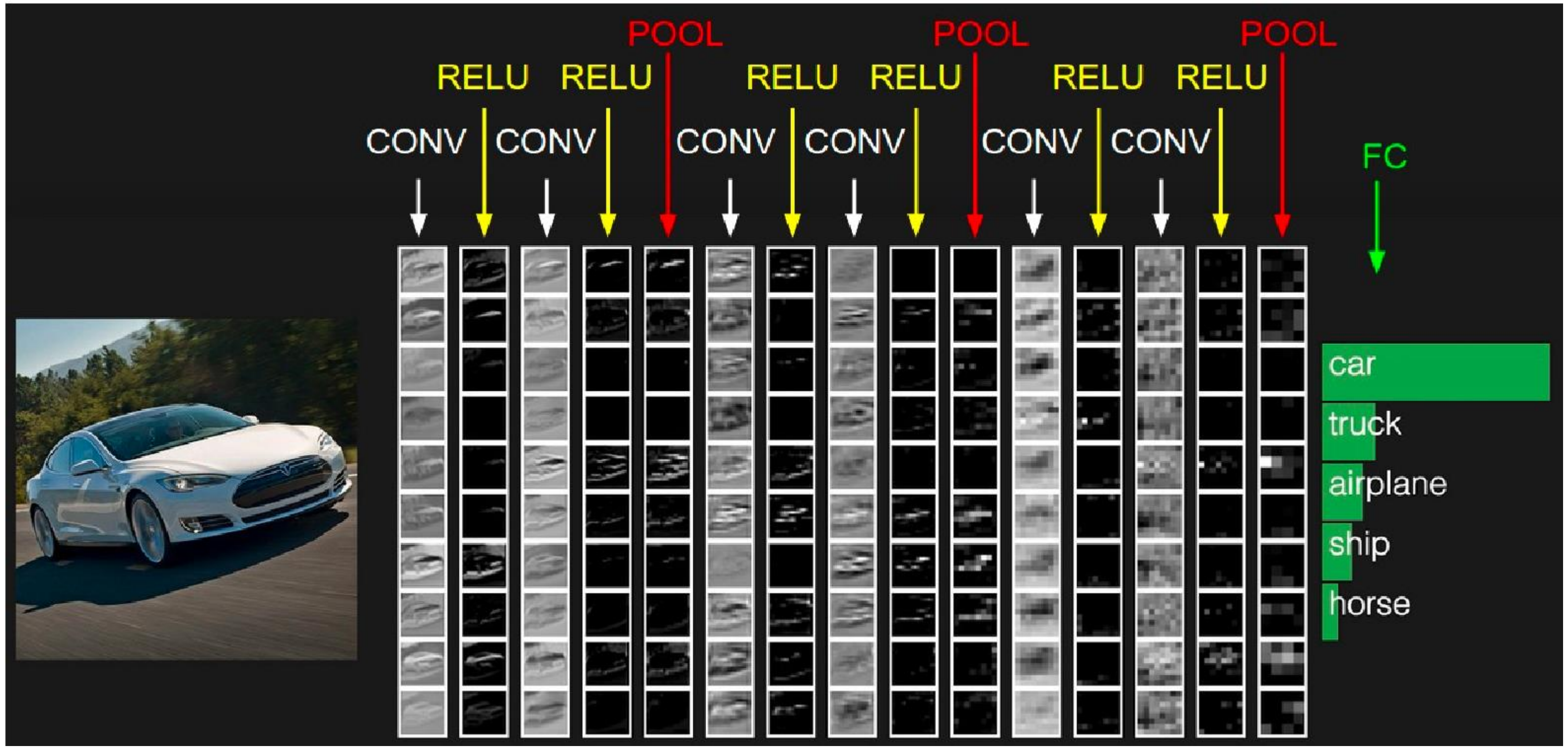
- Note: FCs can learn (pattern, location) combinations in images
  - The learned patterns do not generalize to different spatial locations.

- Can we capture local patterns (e.g. *existence* of a wheel in an image) regardless of the spatial location in the image and leverage them for better classification?
  - low level: edge of some orientation, a patch of some color
  - high level: shape of a wheel
  - i.e. can we learn a group of neurons that detect patterns at all locations?

- Encodes inductive bias

Convolutional neural networks (CNN)

- A.K.A. ConvNet architecture
- A set of neural network architecture that consists of
  - convolutional layers
  - pooling layers
  - fully-connected (FC) layers



(Stanford CS231n)

Convolution: some intuition

- For $f, g: \mathbb{R} \to \mathbb{R}$, define their convolution as:

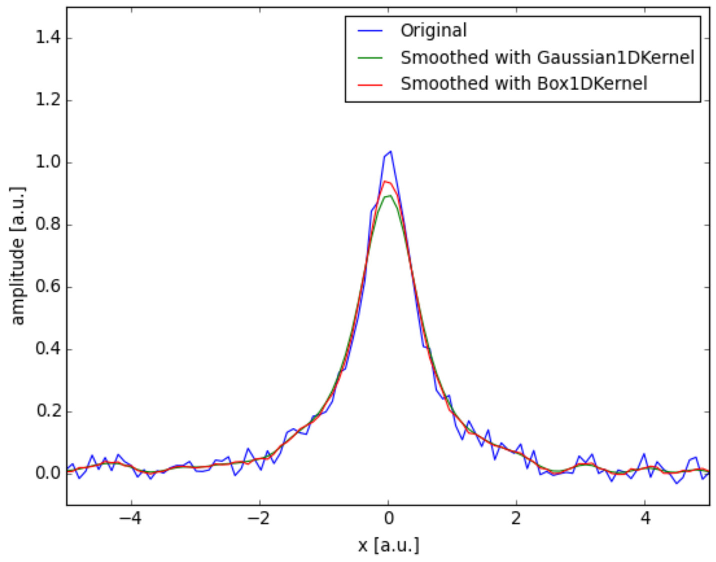
$$(f * g)(x) = \int f(x-y)g(y)\mathrm{d}y$$

- Important special case: g is a function with "narrow support", say $g(y) = 0$ outside $[-1,1]$,

$$\text{Then } (f * g)(x) = \int_{-1}^{-1} f(x-y)g(y)\mathrm{d}y$$

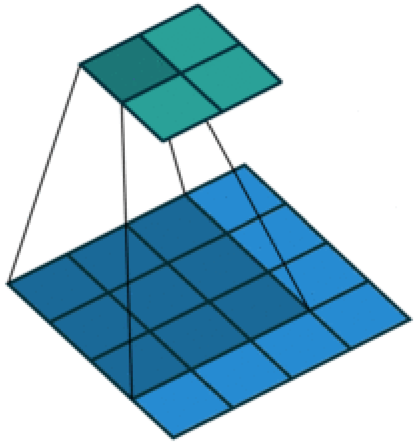
- Informally, for every x, $(f * g)(x)$ is the correlation of
  - $f(z): z \in [x-1, x+1]$
  - $g(z): z \in [-1, +1]$
  - Special case: $g \geq 0$ is a smooth "weighting function"

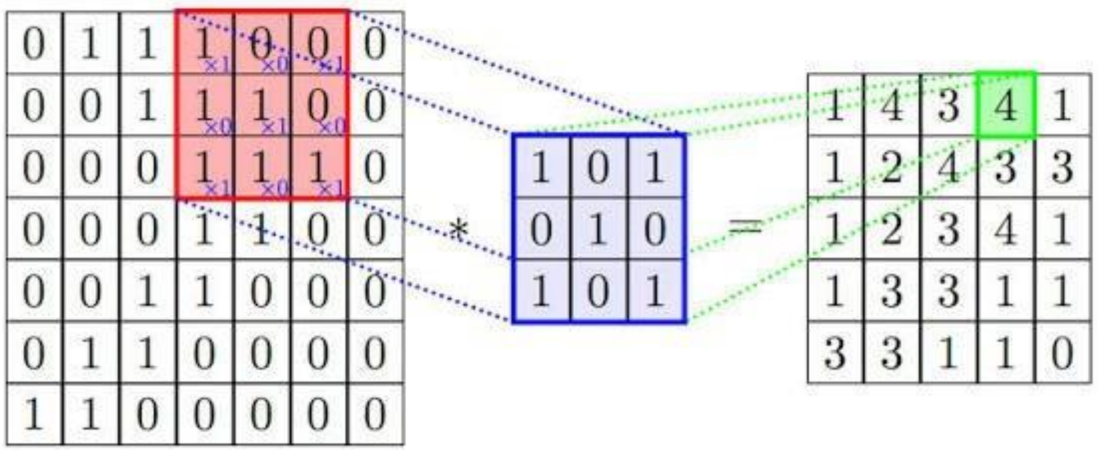
    => $f * g$ is a "smoothing" of f



Convolution for single-channel images

- Consider one filter with weights $\{w_{i,j}\}$ with size F x F
  - For every F x F region of the image, perform inner product (= element wise product, then sum them all)
  - Q: given a w x h image, after convolution with a F x F filter, what is the size of the resulting image?
  - Terminologies: filter size, receptive field size, kernel.





Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional layer for multi-channel images

- Input: w (width) x h (height) x c (#channels)
  - E.g. 32 x 32 x 3
  - 3 channels: R, G, and B

- A convolutional filter on such image is of shape F x F x c
  - Only spatial structure in the first two dimensions
  - Denoted by $\{w_{i,j,k}\}$

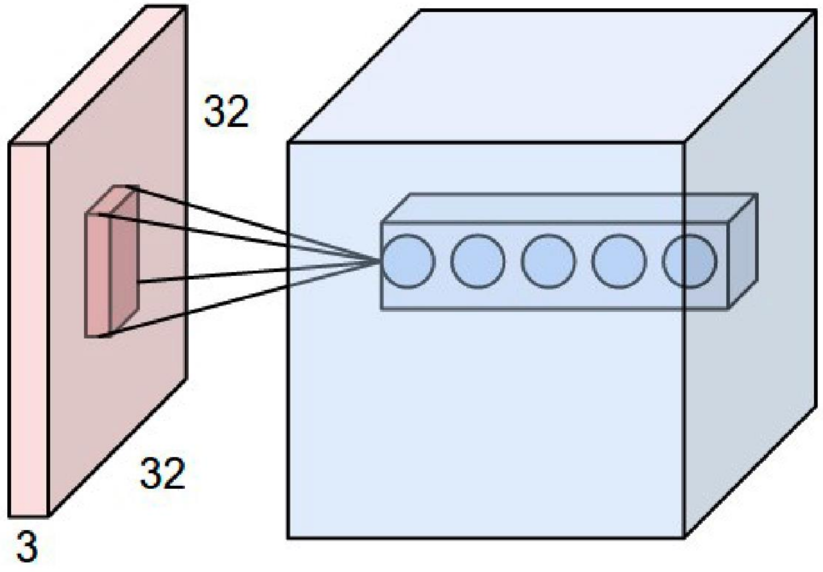
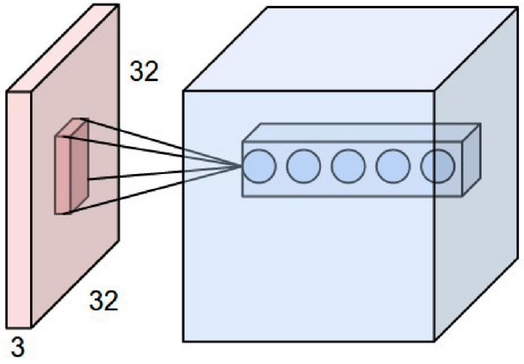
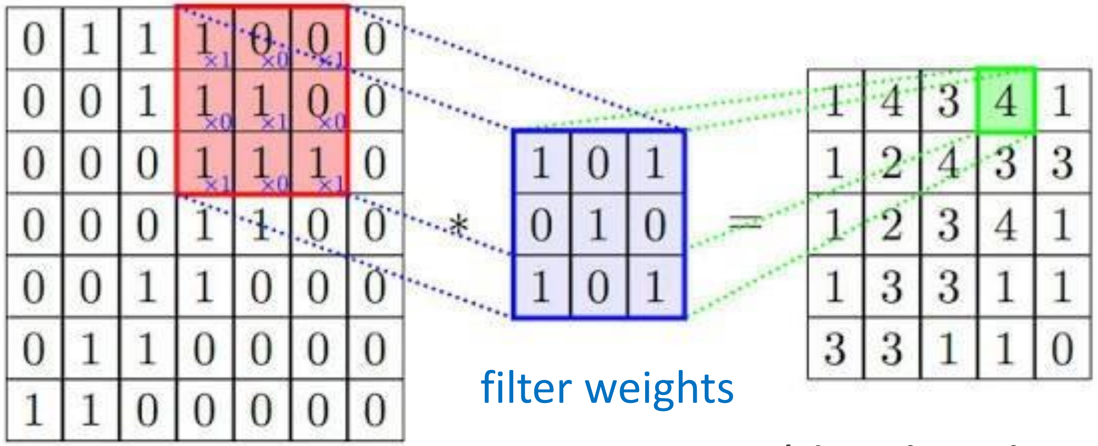


image from Stanford CS231n

Convolutional layer: visual explanation



- Consider one filter with weights $\{w_{i,j,k}\}$ with 5 x 5 x 3
  - Imagine a sliding 3d window.
  - **Convolution:** (detail: usually, add a bias term as well)
    - For every 5 x 5 region of the image, perform inner product (= element wise product, then sum them all)
  - Then apply the activation function (e.g., ReLU)
- Results in 28 x 28 x 1 – called **activation map**.
- Now, we can do K of these filters but with different weights $\{w^{(\ell)}_{i,j,k}\}$ for $\ell \in [K]$ => output is 28 x 28 x K



(depth=1 here)

(image from https://www.quora.com/Why-do-we-use-convolutional-layers)

Convolutional Layer: Why is it useful?

- Why is it useful?
  - The set of weights represent a pattern (i.e., diagonal edge). The activation map represents 'where the pattern has occurred'.

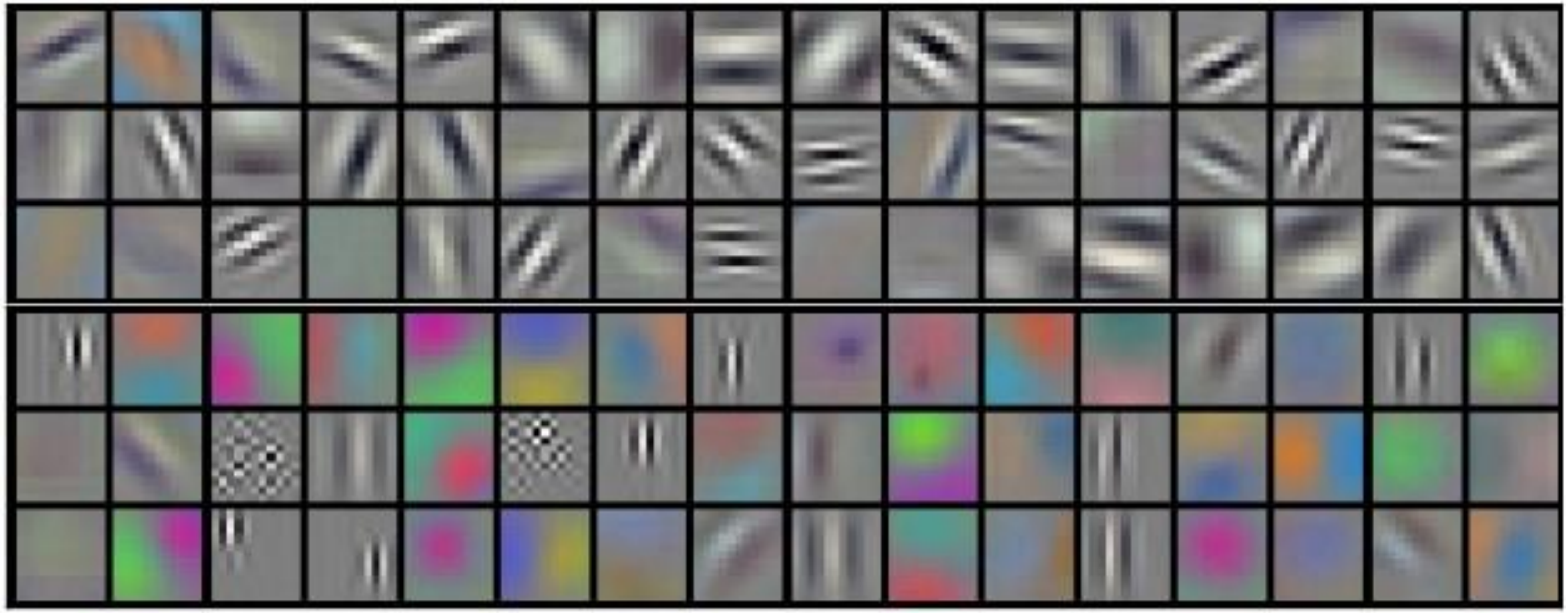
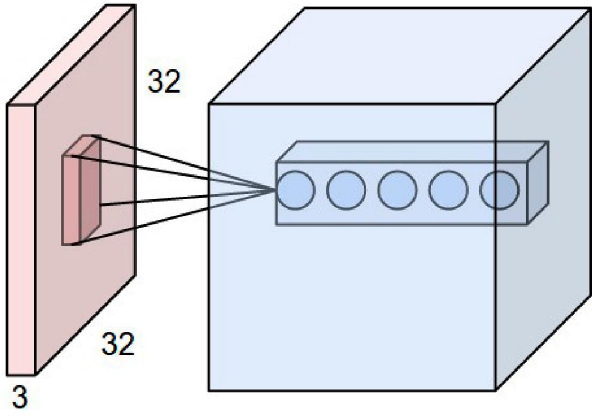


image from Stanford CS231n

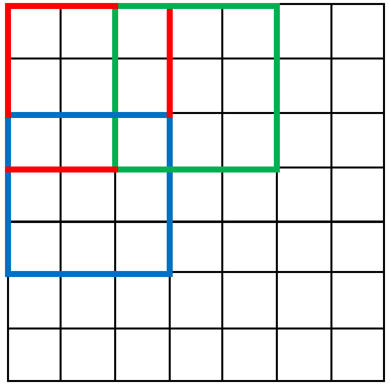
Convolutional layers beyond the first layer

- Generalization: conv layer as the 2nd layer or more
  - Input **volume** (3d object with size w x h x d):
    - the d (called depth) is not necessarily 3
  - Output **volume:** size w’ x h’ x d’, where d’ is the number of filters at the current layer.



- Interpretation: patterns over the patterns.
    - Each filter now convolves and combines d’ activation maps for each spatial location.
    - e.g., combinations of particular edges and textures

Convolutional layer: More details



- Stride length S
    - Skip input regions; Move the sliding window of a filter not by 1 but by S.
    - E.g., S=2 means skipping every other 5 by 5 region.

- Zero-padding P: add P number of artificial pixels with value 0 around the input image on both sides
    - To ensure the spatial dimension is maintained (otherwise, patterns at the corners are not detected well)
    - If we use P=1, then the activation map will be 30 x 30, not 28 x 28 in our example!

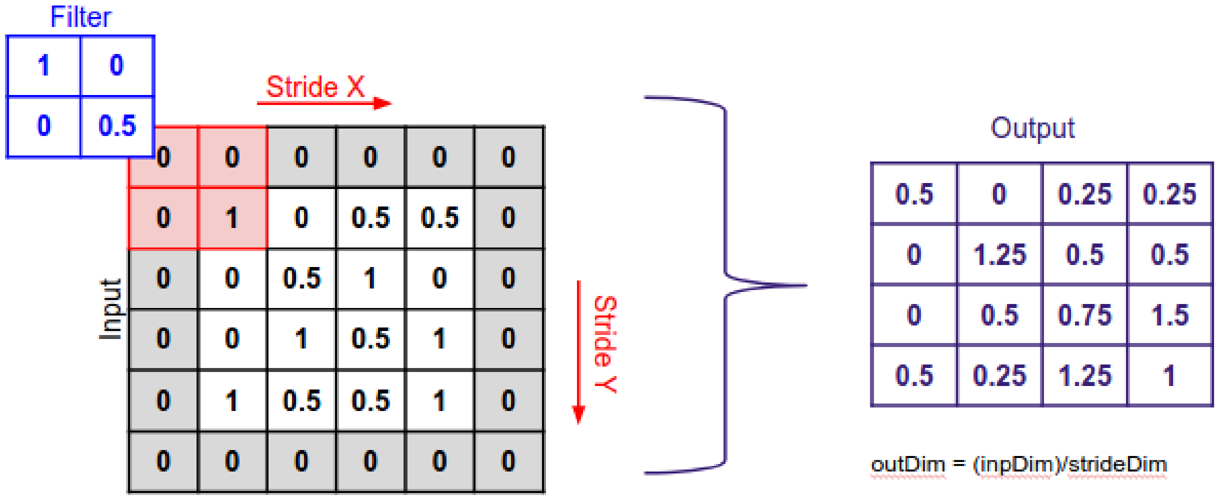
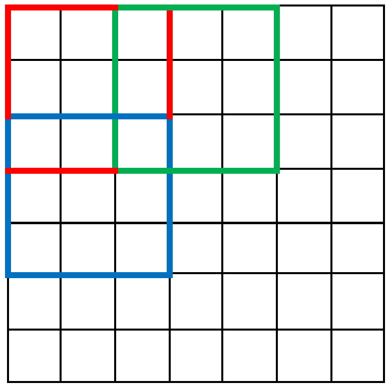


image from https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

Convolutional layer: More details



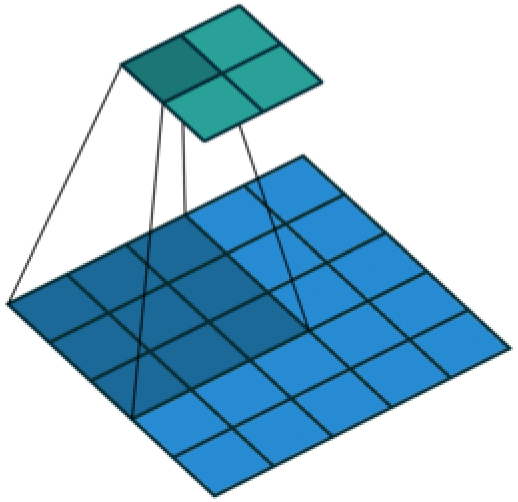
- Stride length S
  - Skip input regions; Move the sliding window of a filter not by 1 but by S.
  - E.g., S=2 means skipping every other 5 by 5 region.

- Zero-padding P: add P number of artificial pixels with value 0 around the input image.
  - To ensure the spatial dimension is maintained (otherwise, patterns at the corners are not detected well)
  - If we use P=2, then the activation map will be 32 by 32 not 28 by 28 in our example!

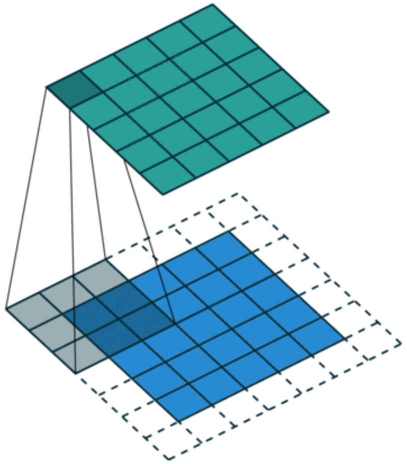
- Rules (same goes for height)
  - W: input volume width, F: filter width (usually, the filter has the same width and height)
  - The output width K = floor((W – F + 2P)/S) + 1
  - E.g., W=32, F=5, P=0, S=1 => K = 28
  - E.g., W=32, F=5, P=2, S=1 => K = 32

Strides and padding: animations

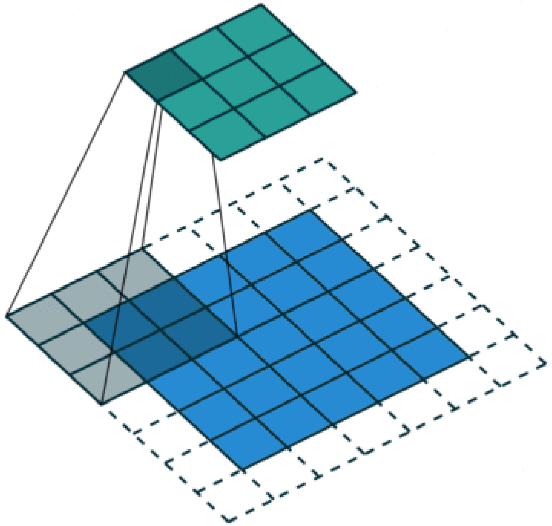
Strides only



Padding only



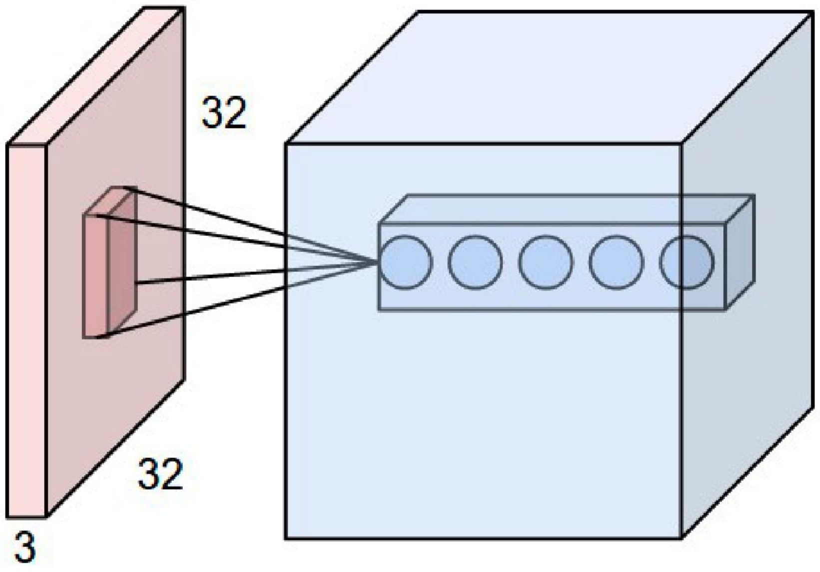
Strides + Padding



Vincent Dumoulin, Francesco Visin - [A guide to convolution arithmetic for deep learning]

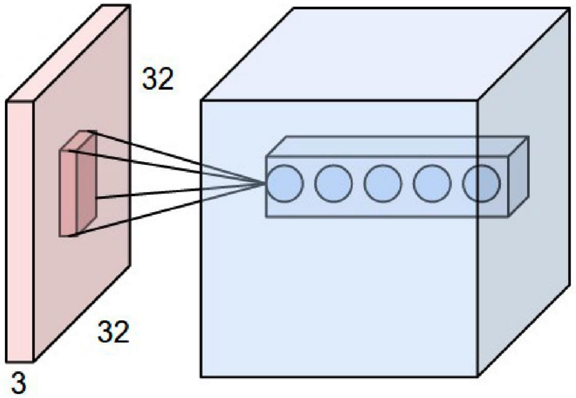
Convolutional layer: Summary

- Input: $W_1 \times H_1 \times D_1$ (width, height, depth)
- Hyperparameters: # of filters K, filter size (=width=height) F, stride S, zero-padding P
- Output: $W_2 \times H_2 \times D_2$
  - $W_2 = \left\lfloor \frac{W_1 - F + 2P}{S} \right\rfloor + 1, \quad H_2 = \left\lfloor \frac{H_1 - F + 2P}{S} \right\rfloor + 1, \quad D_2 = K$
- How many parameters? (# of weights + # of biases)



- Generic recommendation: F=3, S=1, P=1.
- More terminology: <u>depth slice</u> (W by H by 1), <u>depth column</u> (1 by 1 by D)

Comparison: FC vs Conv



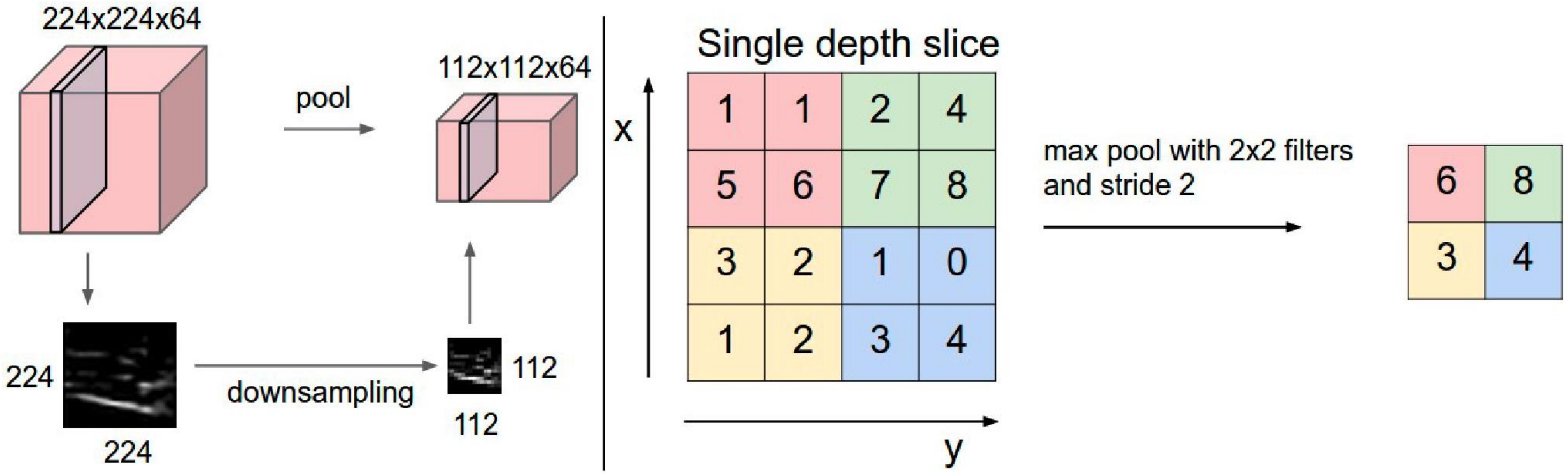
- Conv layer allows *parsimonious* representations:
  - Inter-layer connections are **local**
  - **parameter is shared** across spatial locations.
- In AlexNet, input is 227 by 227 by 3, and the first conv layer output is 55 by 55 by 96 (96 filters)
  - Each filter has 11*11*3 weights with 1 bias => 364 parameters
  - 364*96 = <u>34,944</u> total parameters are used to compute the output 55*55*96 = <u>290,400</u>

- What if we didn't do **parameter sharing**? I.e., for each region of image, use independent filter parameter w.
  - roughly, 290,400 * 364 = 105,705,600
- What if we use FC to compute the same number of outputs? (the parsimony of **local connections**)
  - 230,187 * 290,400 = 66,846,304,800 parameters

- Conv layer can be seen as imposing **inductive bias** specialized for images
- This also prevents overfitting: idiosyncratic pattern that appear in few images are not picked up while training! => useless filters are 'squeezed out' or 'crowded out' by useful filters.

Pooling layer

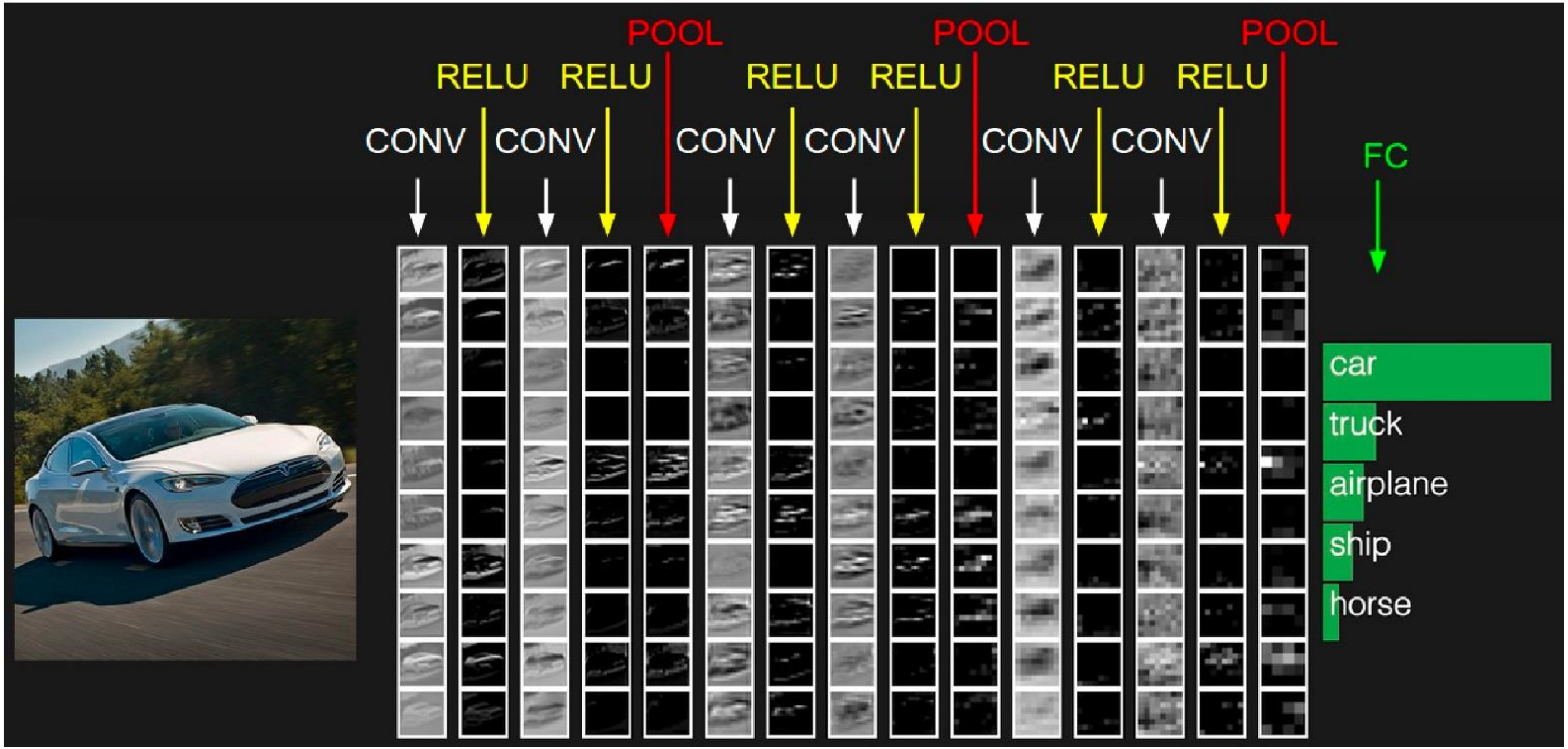
- The role: Summarize the input and scale down the spatial size.
  - has the effect of **routing** the region with the most activation.
- Recall depth slice: take the matrix at a particular depth.
- Max pooling: run a particular filter that computes maximum, for each depth slice.



- Variation: average pooling (but not popular).
- Recommended: Filter size F=2, stride length S=2. (F=3, S=2 is also commonly use – overlapping pooling).
- Note: There are **no parameters** for this layer!

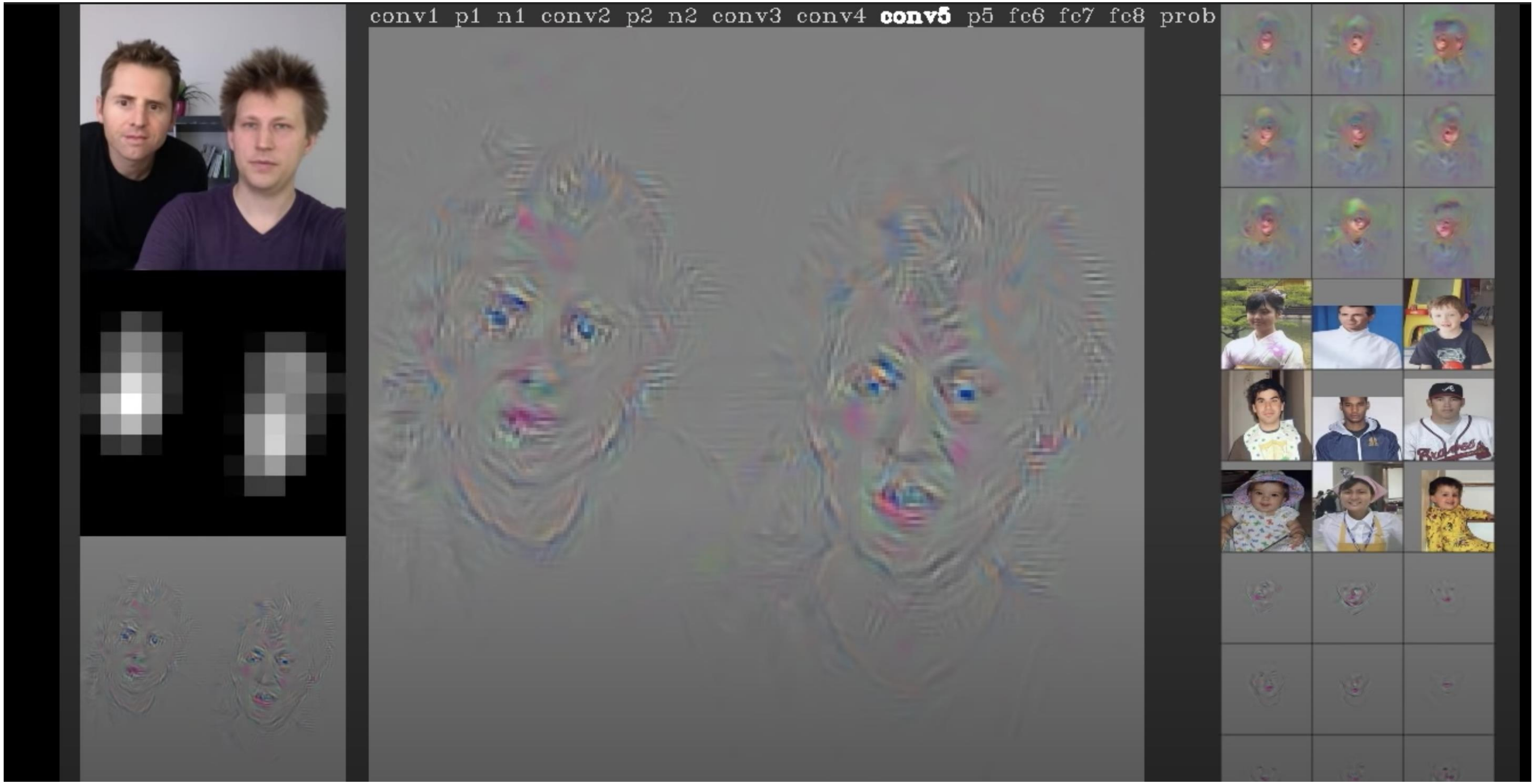
figure from Stanford CS231n

Typical architectural patterns in CNN



Seeing what happens in CNN

- https://yosinski.com/deepvis#toolbox



CNN examples

LeNet-5

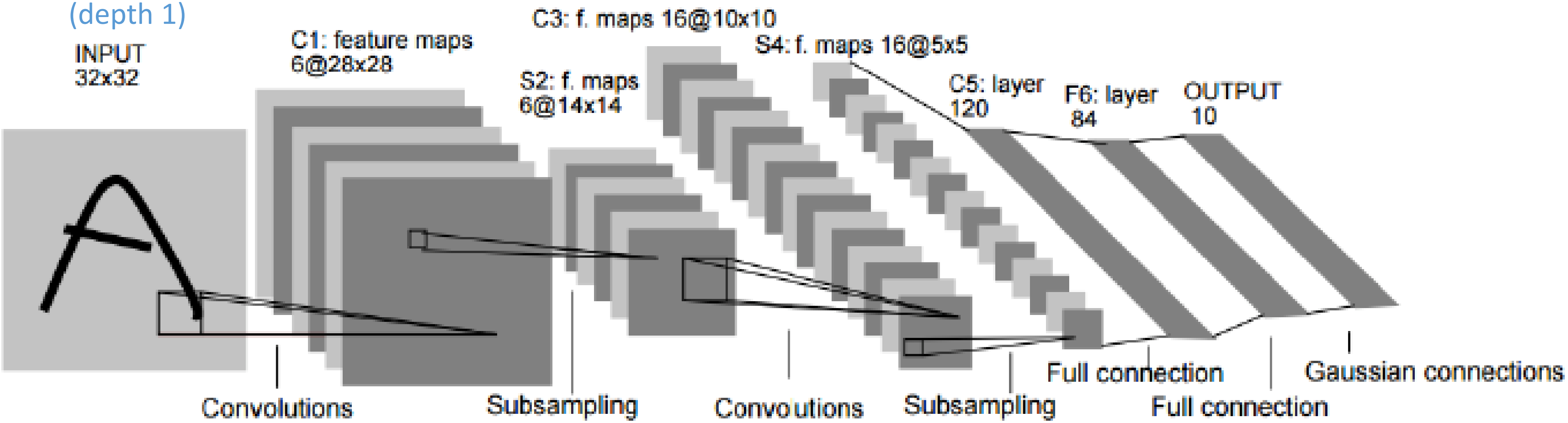
- Proposed in "Gradient-based learning applied to document recognition" *, by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, in Proceedings of the IEEE,* ***1998***

- Apply convolution on 2D images (MNIST) and use backpropagation

- Structure: 2 convolutional layers (with pooling) + 3 fully connected layers
  - Input size: 32x32x1
  - Convolution kernel size: 5x5
  - Pooling: 2x2

LeNet-5

(depth 1)



5 by 5 filters
K=6
stride 1

2x2 pooling
stride 2

5 by 5 by 6 filters
K=16
stride 1

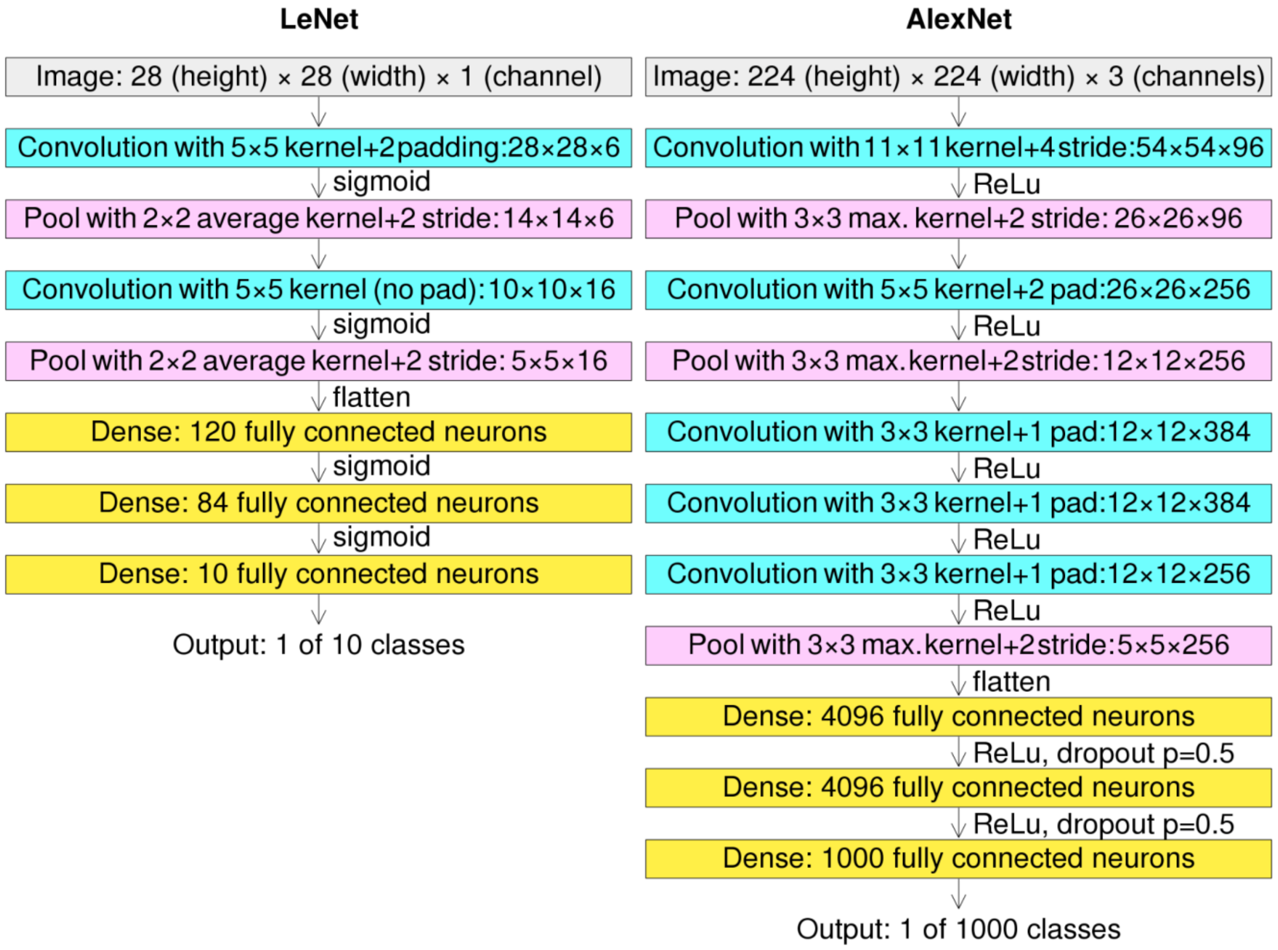
2x2 pooling
stride 2

"Gradient-based learning applied to document recognition" , *by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, in Proceedings of the IEEE, 1998*

AlexNet (2012)

(1000 classes)

- Won the ImageNet competition with <u>top-5 test error rate</u> of 16.4% (second place was 26.2%).
- Almost just an extension of LeNet-5. But, uses ReLU for the first time.



https://en.wikipedia.org/wiki/AlexNet

Krizhevsky, Sutskever, and Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012.

VGGNet (2014): 7.3% error on ImageNet

- Mimic large convolutional filters with multiple small (3x3) convolutional filters
- Every time it halves the spatial size, double the # of filters

(not counting biases)

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

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

[Simonyan and Zisserman, 2014]

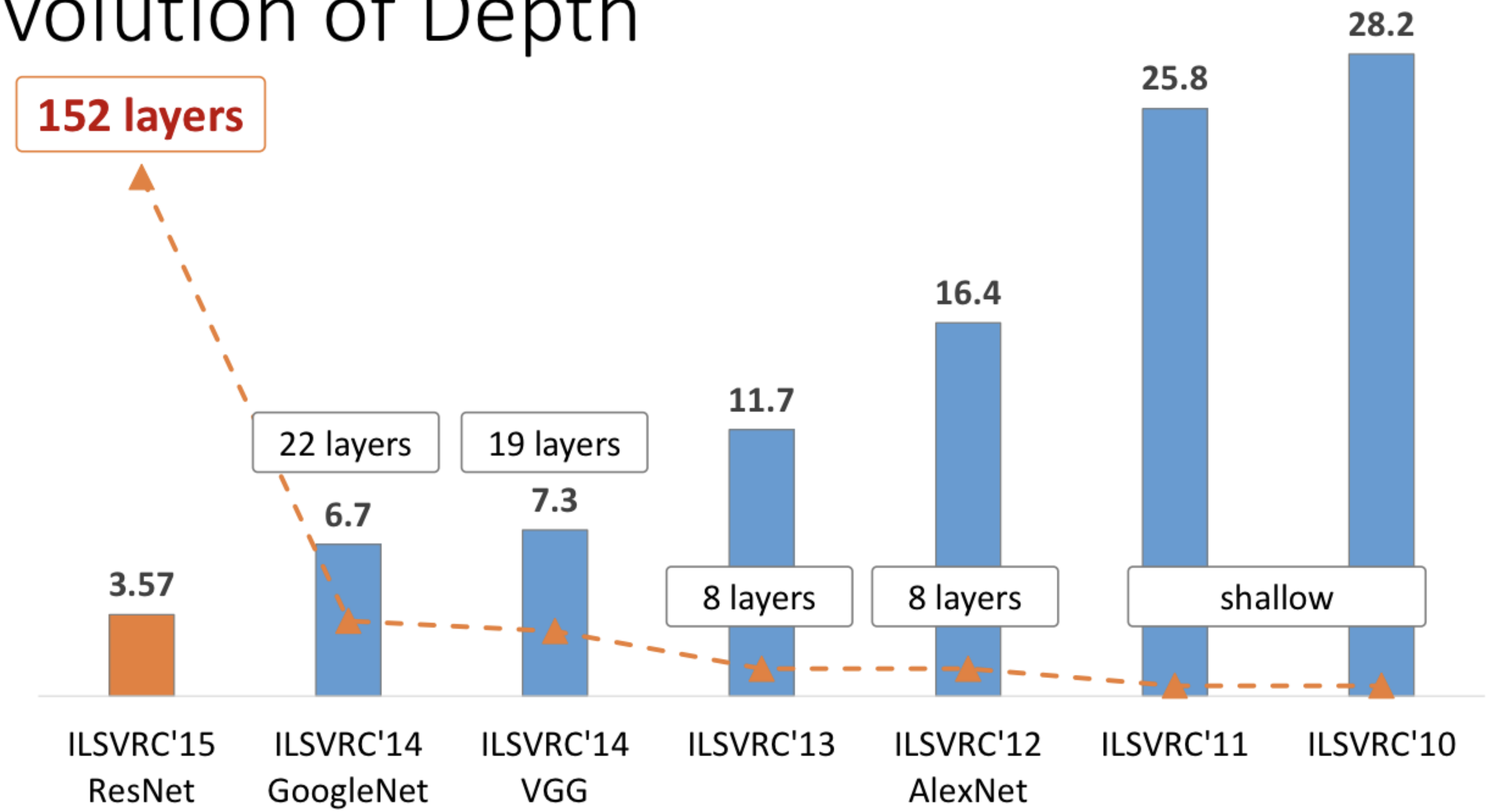
slide from Stanford CS231n

ResNet (2016): 3.5% error on ImageNet

- Proposed in “Deep residual learning for image recognition” by *He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun*. In *Proceedings of the IEEE conference on computer vision and pattern recognition*,. 2016.
- Apply very deep networks with repeated **<u>residual blocks</u>**.
- Structure: simply stacking residual blocks, but the network is very deep.

- Let’s see the motivation.

Revolution of Depth



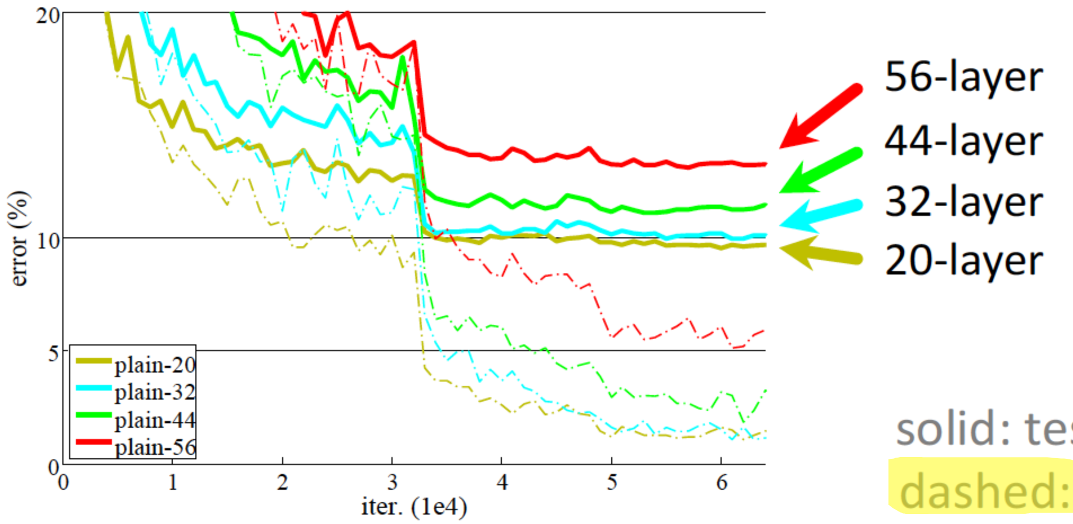
ImageNet Classification top-5 error (%)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf

Deep nets seem to suffer

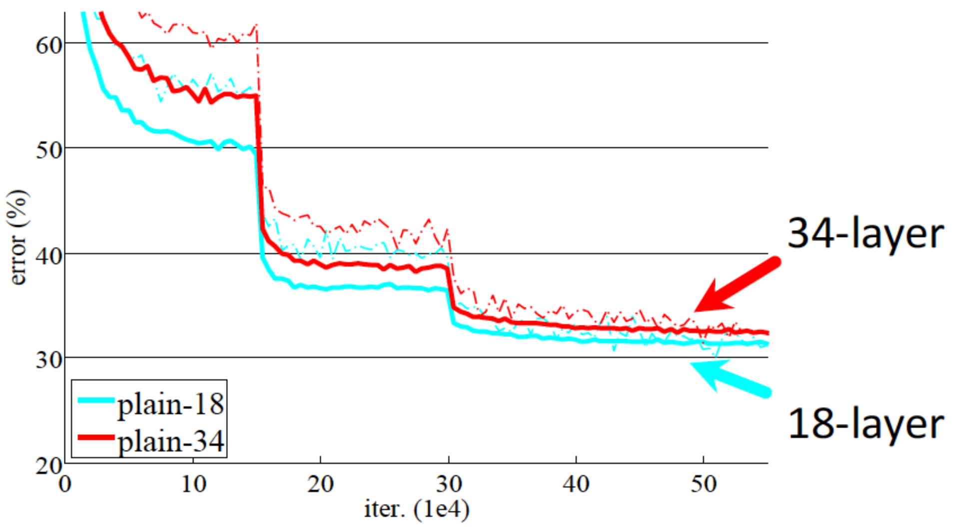
CIFAR-10



solid: test/val

dashed: train

ImageNet-1000



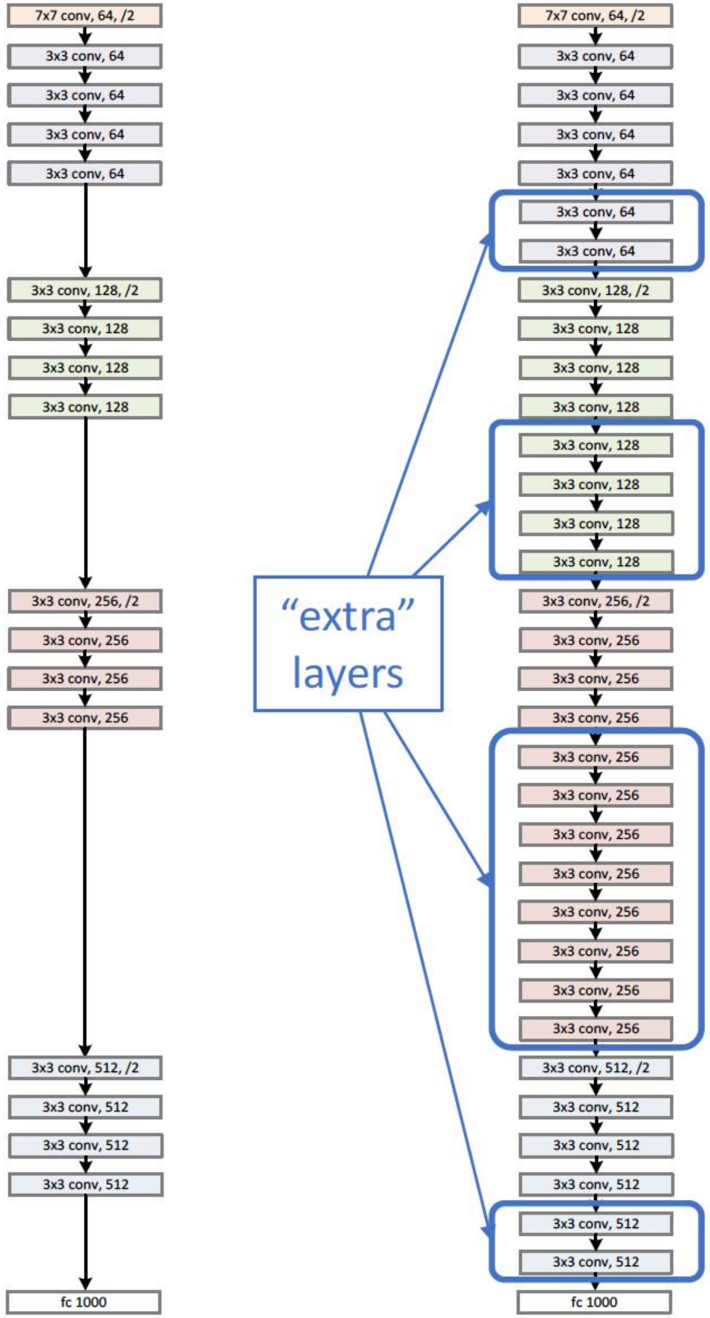
- “Overly deep” plain nets have **higher training error**
- A general phenomenon, observed in many datasets

http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf

Microsoft Research

(slides from Kaiming He)

a shallower model (18 layers)



a deeper counterpart (34 layers)

- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…

http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf

ICCV15 International Conference on Computer Vision

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

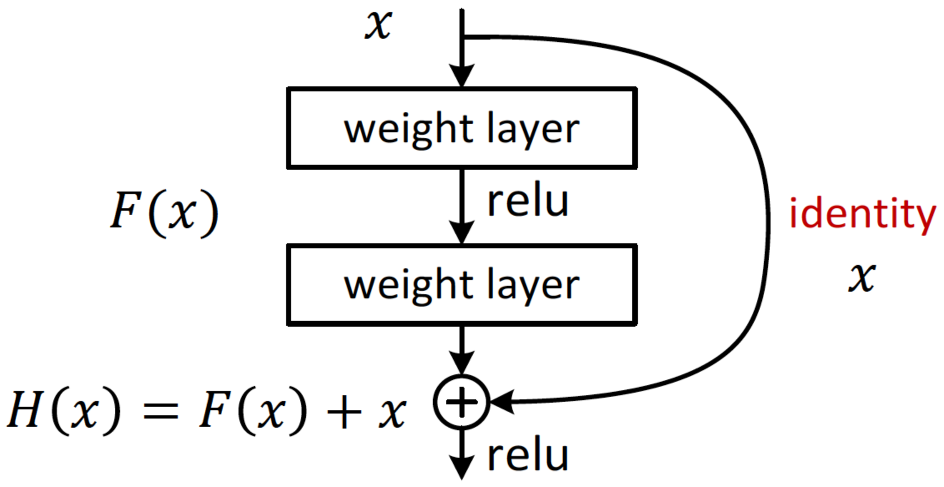
Skip connections for better optimization

- **Skip connections**

- $F(x)$ encodes residual representations, which has previously been explored in early works

- When backprop'ing, by the chain rule, gradients will 'flow' directly to the previous layer.
    - Recall: when the computation graph splits, the gradient is a summation of the gradients of the branches.
    - In contrast, plain CNNs suffer from vanishing gradient problem

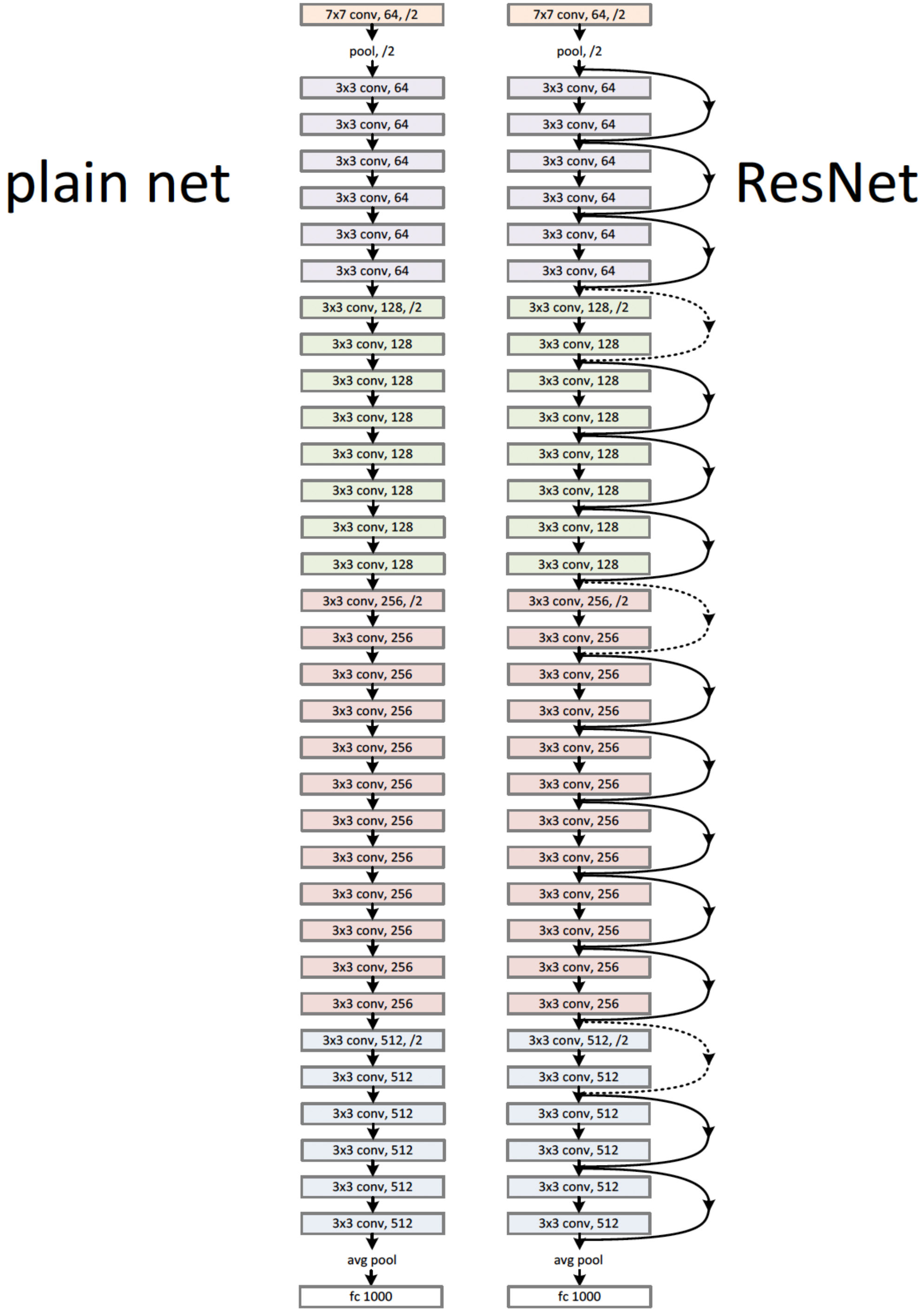
- Residual net



http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf

ResNet

- VGG-style scheme: halve the special size, double the # filters
- Max pool appears only once.
- Use conv layer with stride 2 occasionally to reduce the dimension => called "**bottleneck**" blocks.

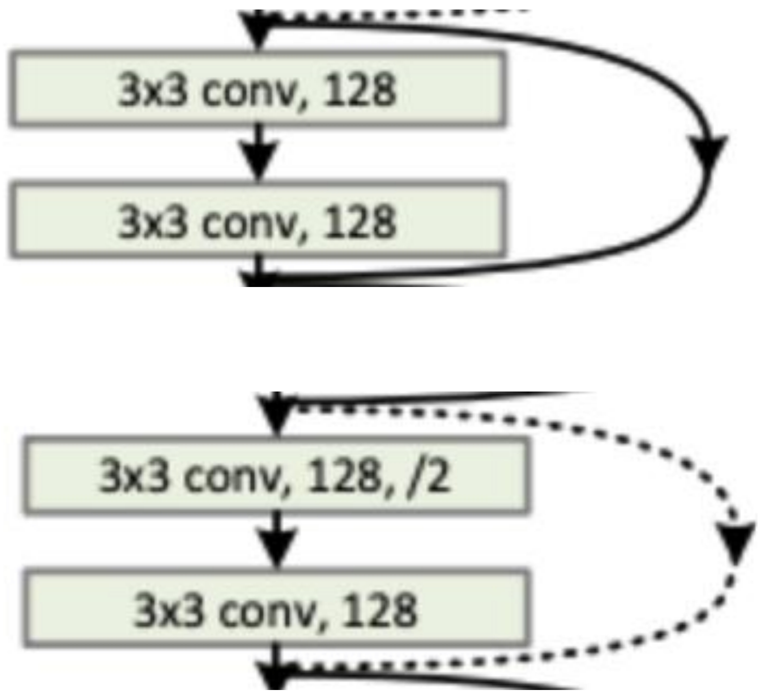


http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf

ResNet in PyTorch

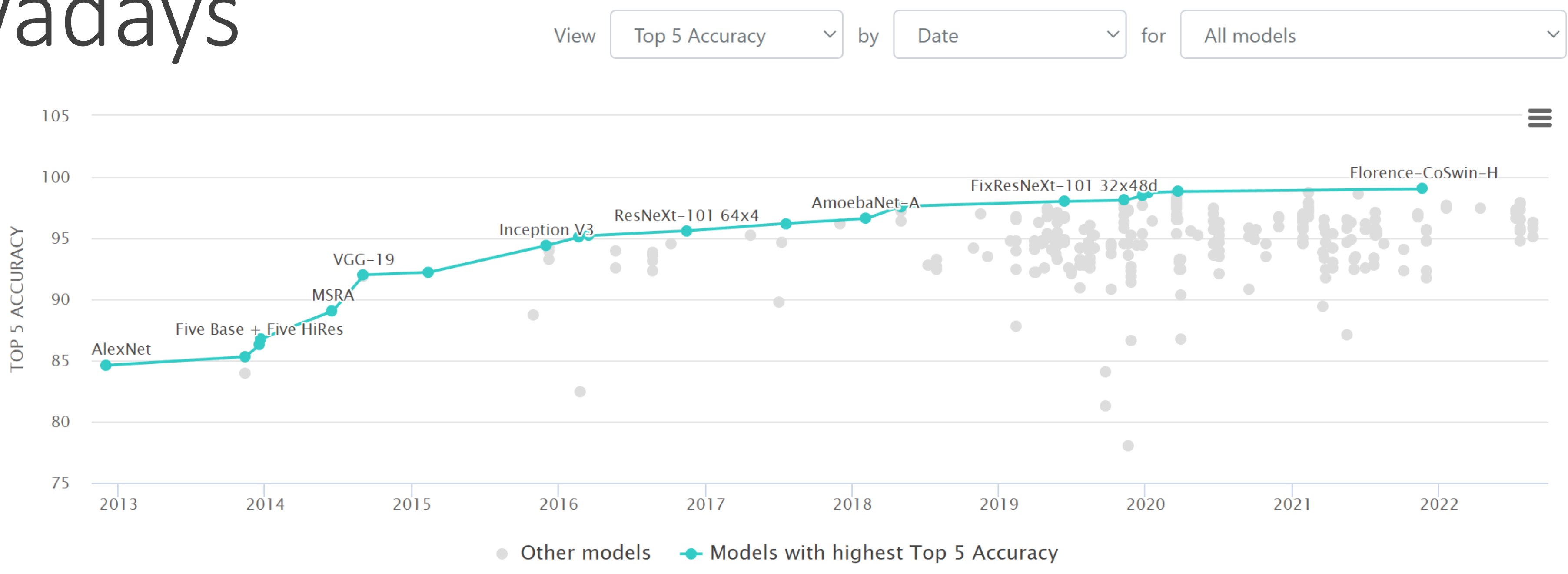
- Torchvision implementation: https://pytorch.org/vision/0.8/_modules/torchvision/models/resnet.html

```
class Bottleneck(nn.Module):
        def forward(self, x):
            identity = x

            out = self.conv1(x)
            out = self.bn1(out)
            out = self.relu(out)

            out = self.conv2(out)
            out = self.bn2(out)
            out = self.relu(out)

            out = self.conv3(out)
            out = self.bn3(out)

            if self.downsample is not None:
                identity = self.downsample(x)

            out += identity
            out = self.relu(out)

            return out
```

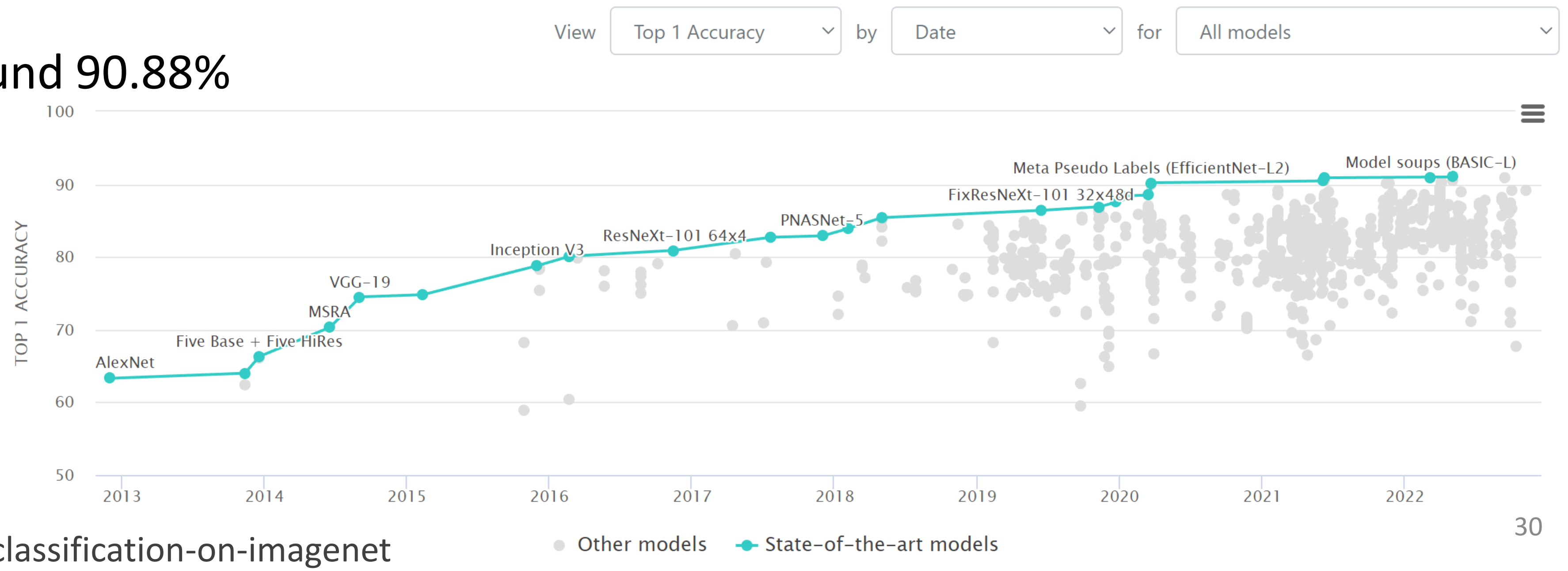


ImageNet nowadays

- Top-5 accuracy is boring



- SoTA top-1 accuracy is around 90.88%



https://paperswithcode.com/sota/image-classification-on-imagenet

Summary

- Convolutional neural networks (CNNs): convolution layers, pooling layers

- Some representative CNN architectures

AlexNet (2012)

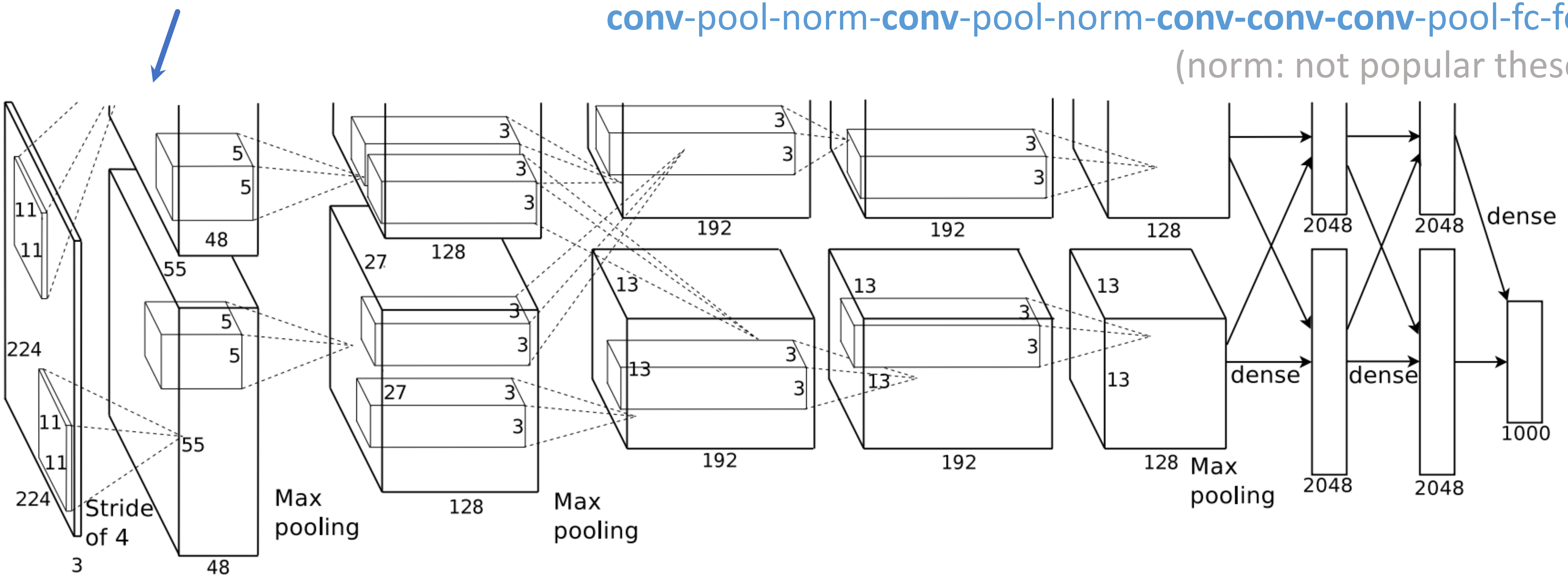
(1000 classes)

- Won the ImageNet competition with <u>top-5 test error rate</u> of 15.3% (second place was 26.2%).
- Almost just an extension of LeNet-5. But, uses ReLU for the first time.

96 filters of 11x11, stride 4

conv-pool-norm-**conv**-pool-norm-**conv-conv-conv**-pool-fc-fc-fc

(norm: not popular these days)



Krizhevsky, Sutskever, and Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012.

- Consider one filter with weights $\{w_{i,j,k}\}$ with 5 by 5 by 3
  - For every 5 by 5 region of the image, perform inner product (= element wise product, then sum them all)
    - This is called **convolution.**
  - then, apply the activation function (e.g., ReLU)
- Results in 28 by 28 matrix – called **activation map**.
- Now, we can do K of these filters but with a different weight. $\{w_{i,j,k}^{(\ell)}\}$ for $\ell \in [K]$. => output is 28 x 28 x K
- Terminologies: filter size, receptive field size, kernel.

Comparison: FC vs Conv

- A unique feature of conv layer: **parameter is shared** across spatial locations.
- In AlexNet, input is 227 by 227 by 3, and the first conv layer output is 55 by 55 by 96 (96 filters)
  - Each filter has 11*11*3 weights with 1 bias   =>   364 parameters
  - 364*96 = <u>34,944</u> total parameters are used to compute the output 55*55*96 = <u>290,400</u>

- What if we didn't do parameter sharing? I.e., for each region of image, use independent filter parameter w.
  - roughly, 290,400 * 364 = 105,705,600
- What if we use FC to compute the same number of outputs?
  - 230,187 * 290,400 = 66,846,304,800 parameters

- Conv layer can be seen as imposing **inductive bias** specialized for images
- This also prevents overfitting: idiosyncratic pattern that appear in few images are not picked up while training! => useless filters are 'squeezed out' or 'crowded out' by useful filters.