Lecture 16: Convolutional Neural Networks

Mar 15, 2023

CIS 4190/5190

Spring 2023

Convolutional Layer Summary

- Local connectivity
- Weight sharing
- Handling multiple input/output channels
- Retains location associations

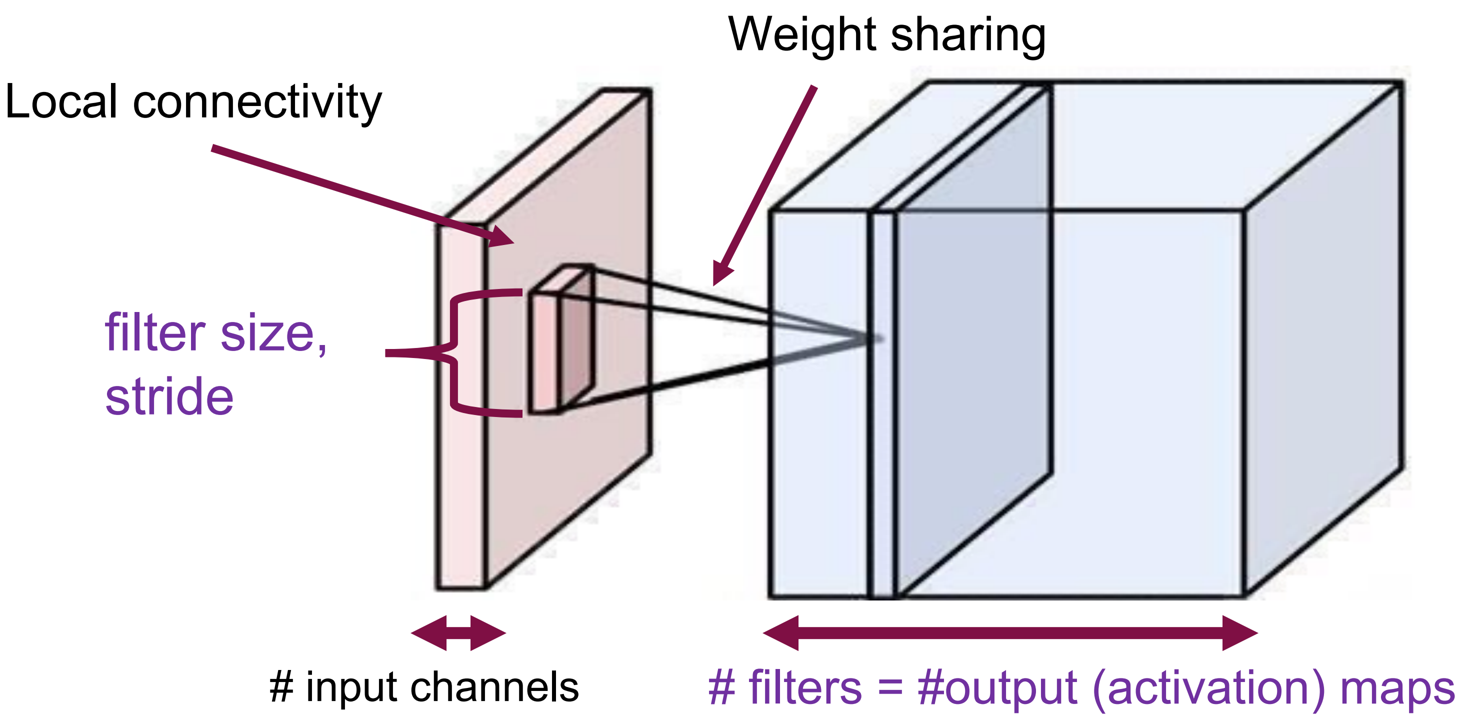
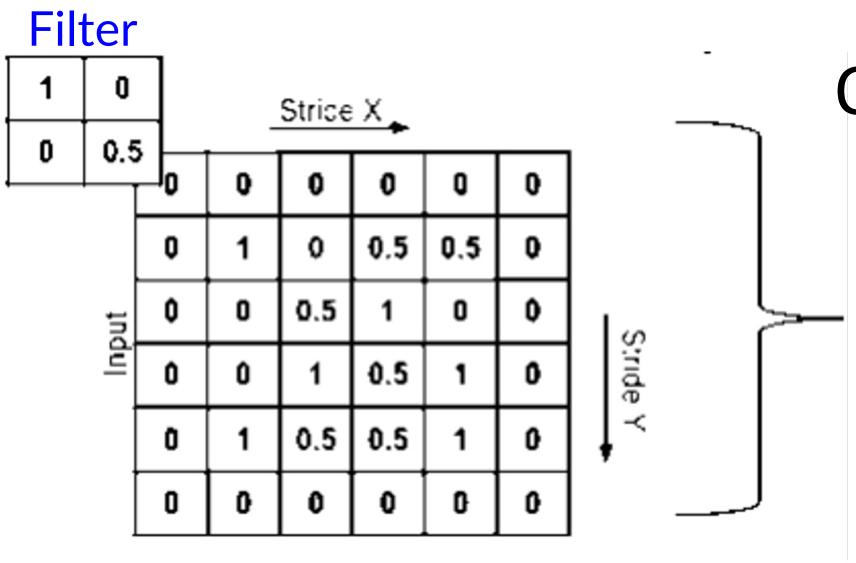


Image credit: A. Karpathy

Slide credit: Jia-Bin Huang

Stride



Output with stride 1

0.5	0	0.25	0.25	0
0	1.25	0.5	0.5	0.5
0	0.5	0.75	1.5	0
0.5	0.25	1.25	1	1
0.	1	0.5	0.5	1

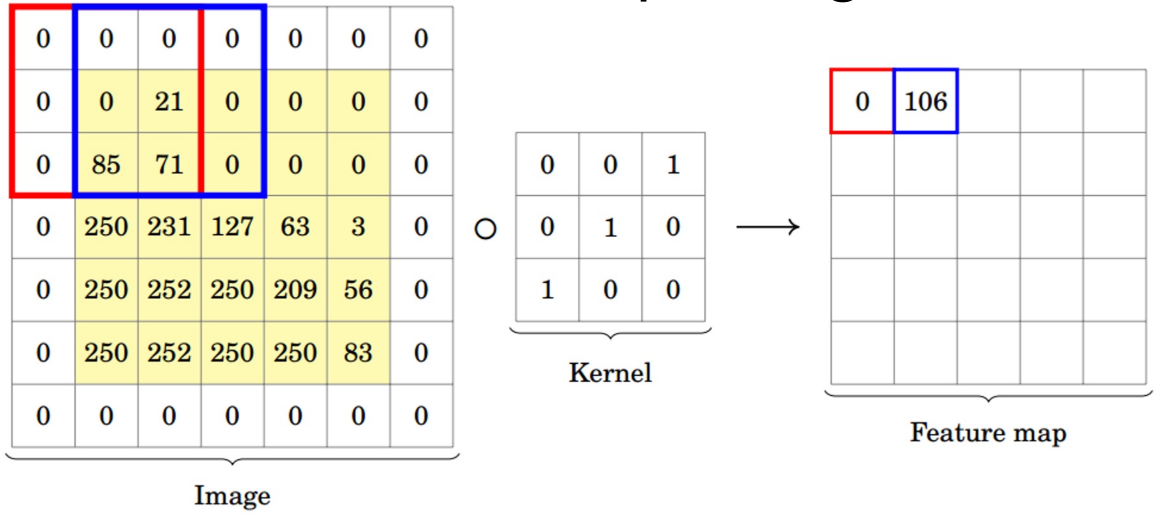
Output with stride 2

0.50	0.25	0.00
0.00	0.75	0.00
0.00	0.50	1.00

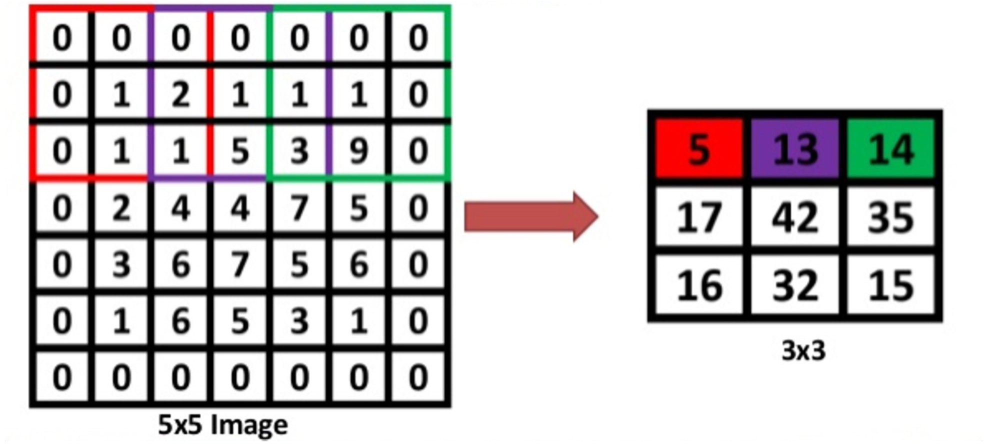
https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

Zero-Padding

stride = 1, zero-padding = 1



stride = 2, zero-padding = 1



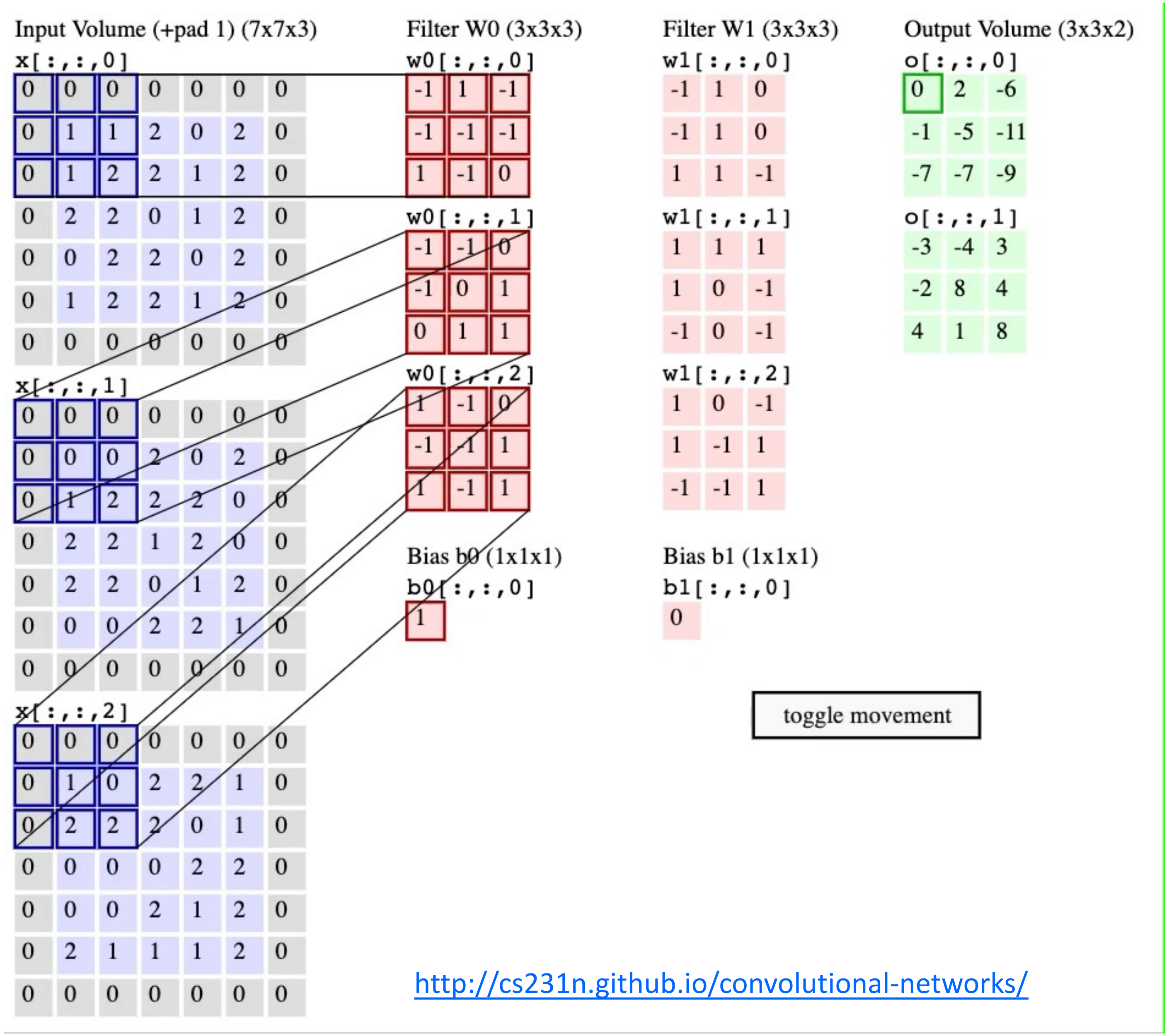
The kernel size, amount of zero-padding, and stride, together determine the output spatial dimensions

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

Convolution Filter Bank Demo

- Notes:
  - Multiple (3) inputs
    - Hence kernels of size 3x3x3
  - Multiple (2) outputs (hence 2 kernels)
  - And one bias parameter for each kernel
  - Stride 2, zero-padding 1

- Net #parameters in the bank:
  - $(3\times3\times3+1)\times2\ =\ 56$



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

Can we back-propagate through a convolution?

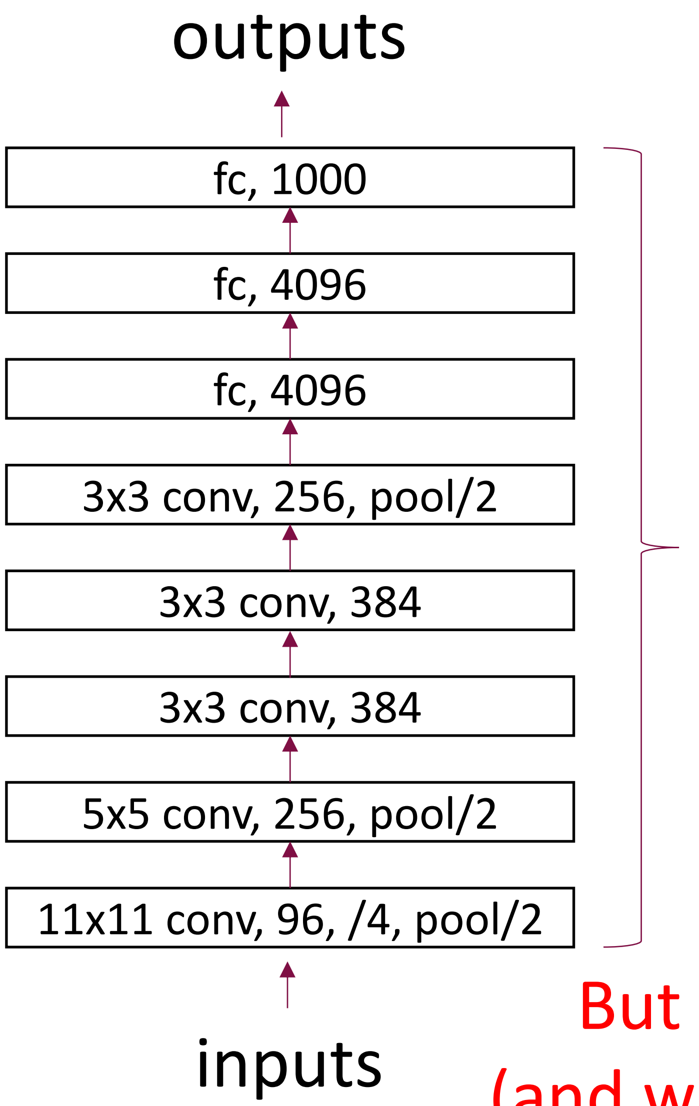
- Yes!
- A convolution is after all a special case of a linear operation $Y = WX$, with local connections and shared weights.
- Differentiable w.r.t. its inputs, as well as w.r.t. its weights.

The paraphernalia around convolutions inside CNNs

Pooling, Normalization, Activation Functions …

Convolutions inside a neural network

Example CNN architecture: The 2012 AlexNet!



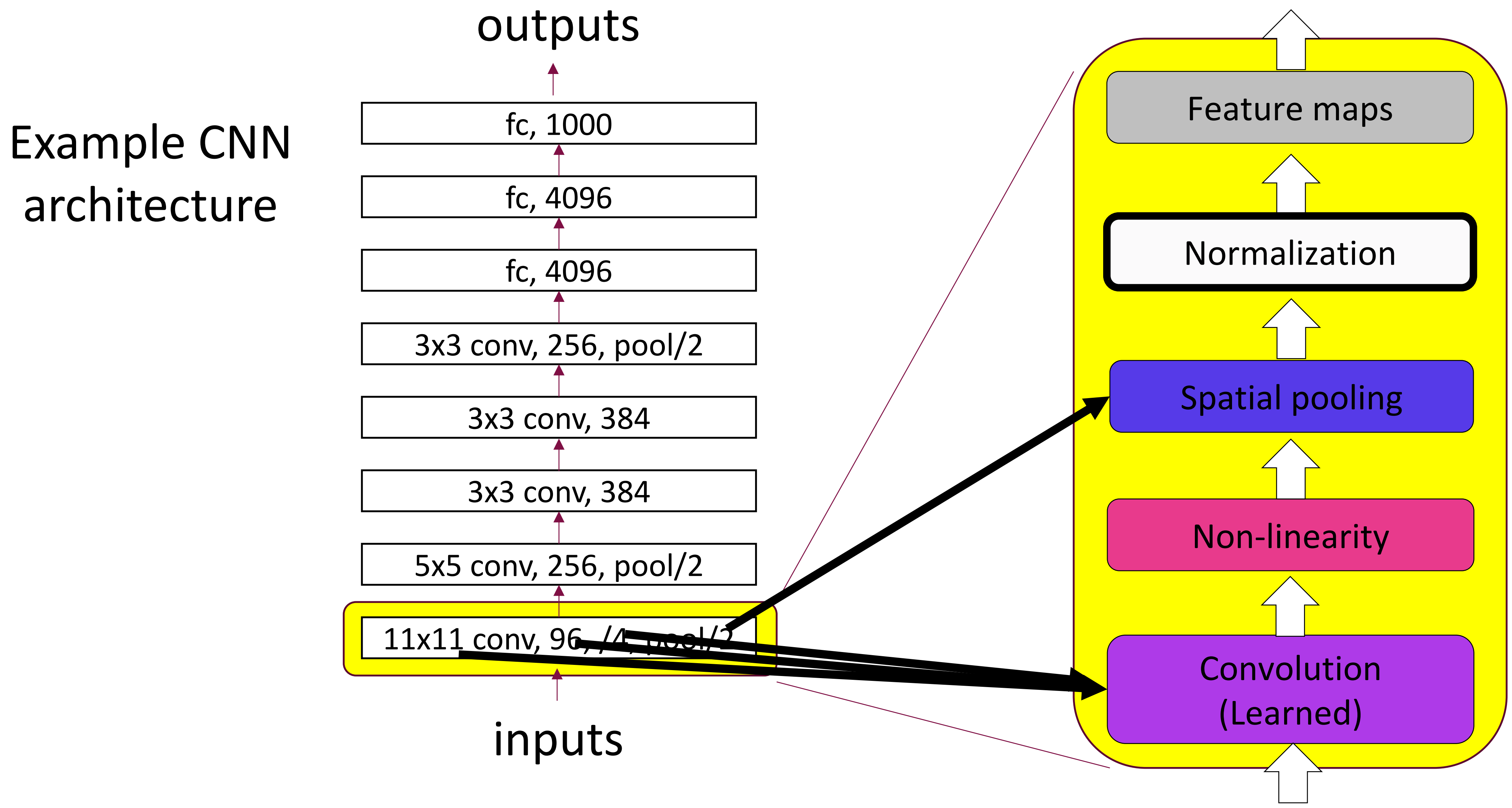
“8 layers”, really “8 layer blocks”

“5 convolution blocks” followed by 3 fully connected layers

More on AlexNet soon!

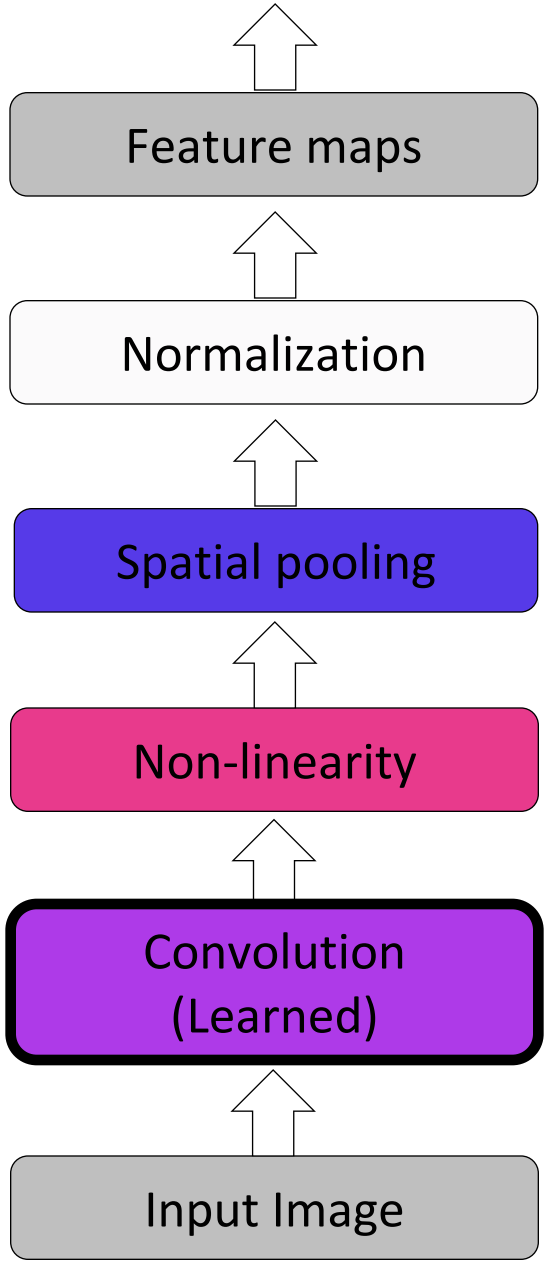
But first, what is a “convolution block”?
(and what are all the numbers in each layer?)

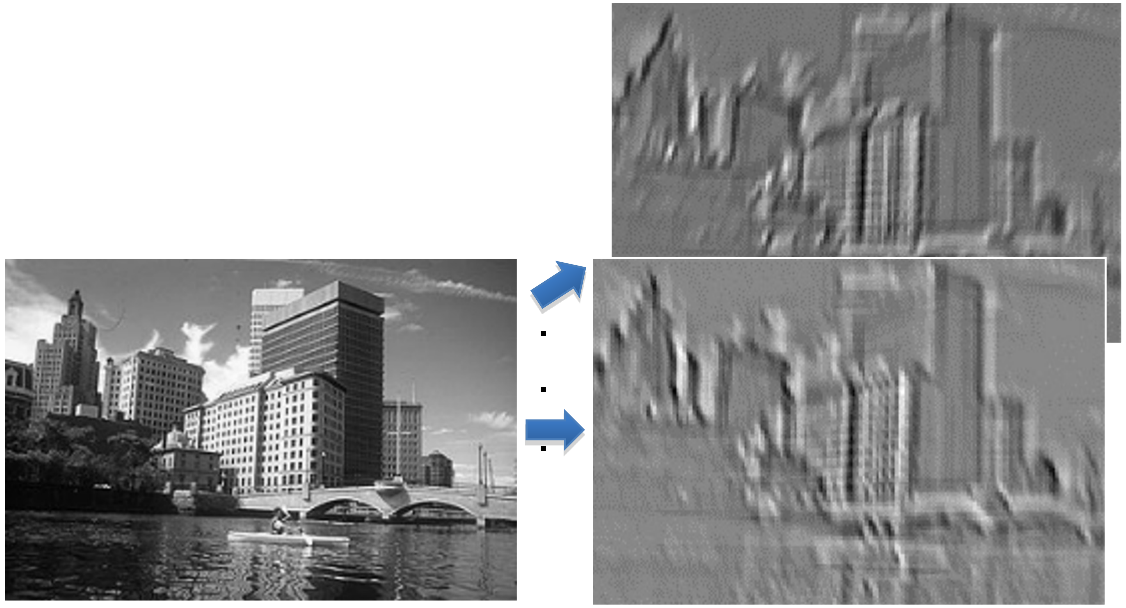
Typical accompaniments to “convolution layers”



graphic credit: S. Lazebnik

Convolve → activation function → pool → normalize



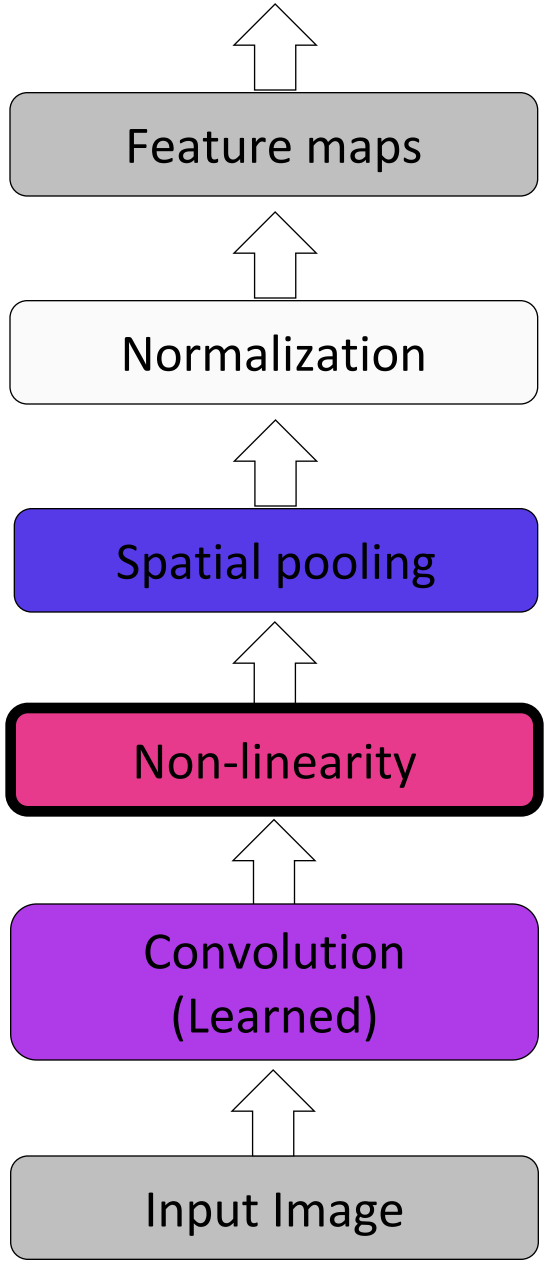


Input

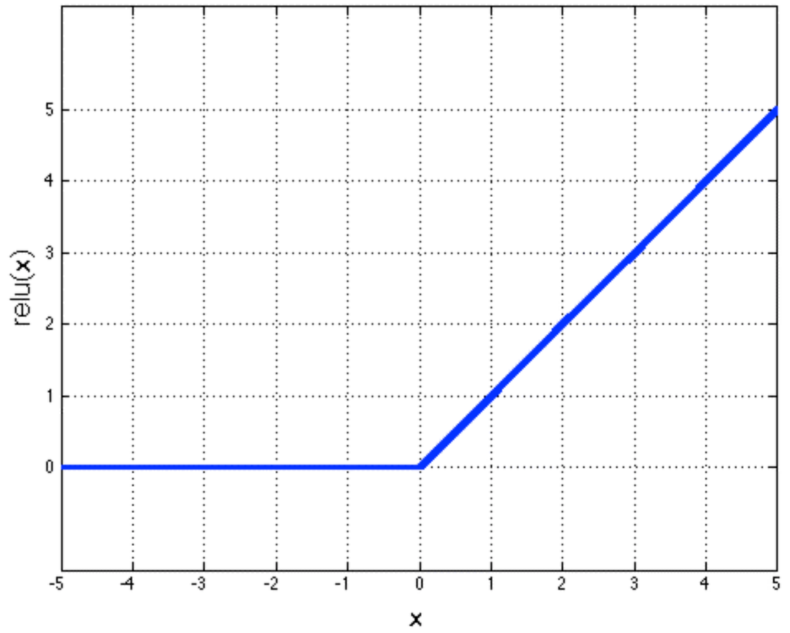
Feature Map

slide credit: S. Lazebnik

Convolve → activation function → pool → normalize

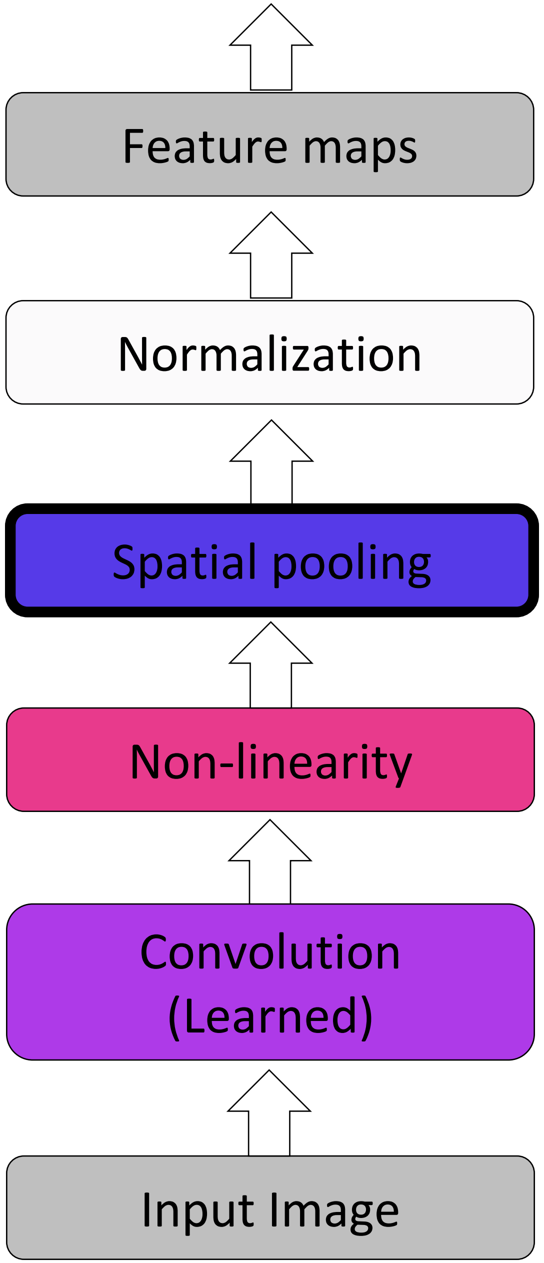


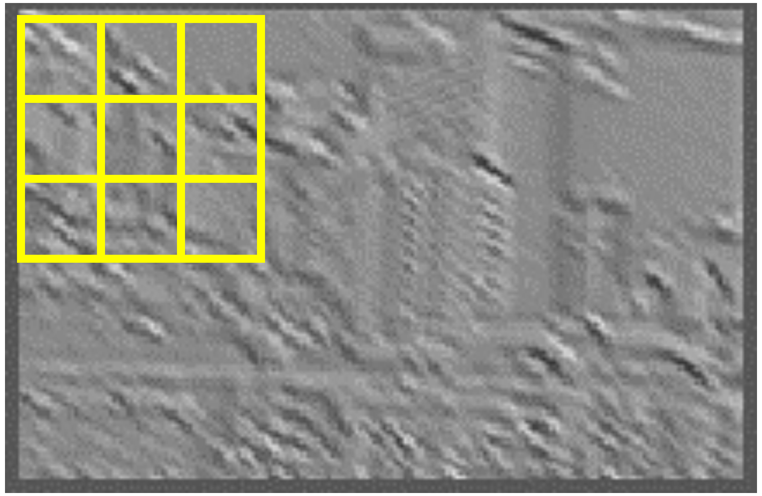
Rectified Linear Unit (ReLU)



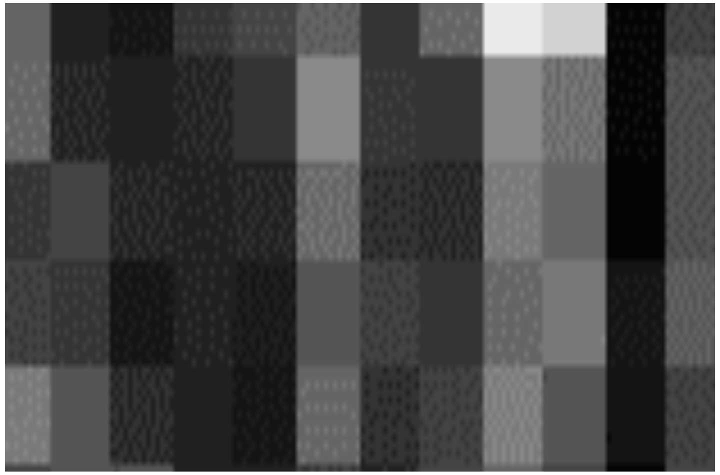
slide credit: S. Lazebnik

Convolve → activation function → pool → normalize





Max pooling

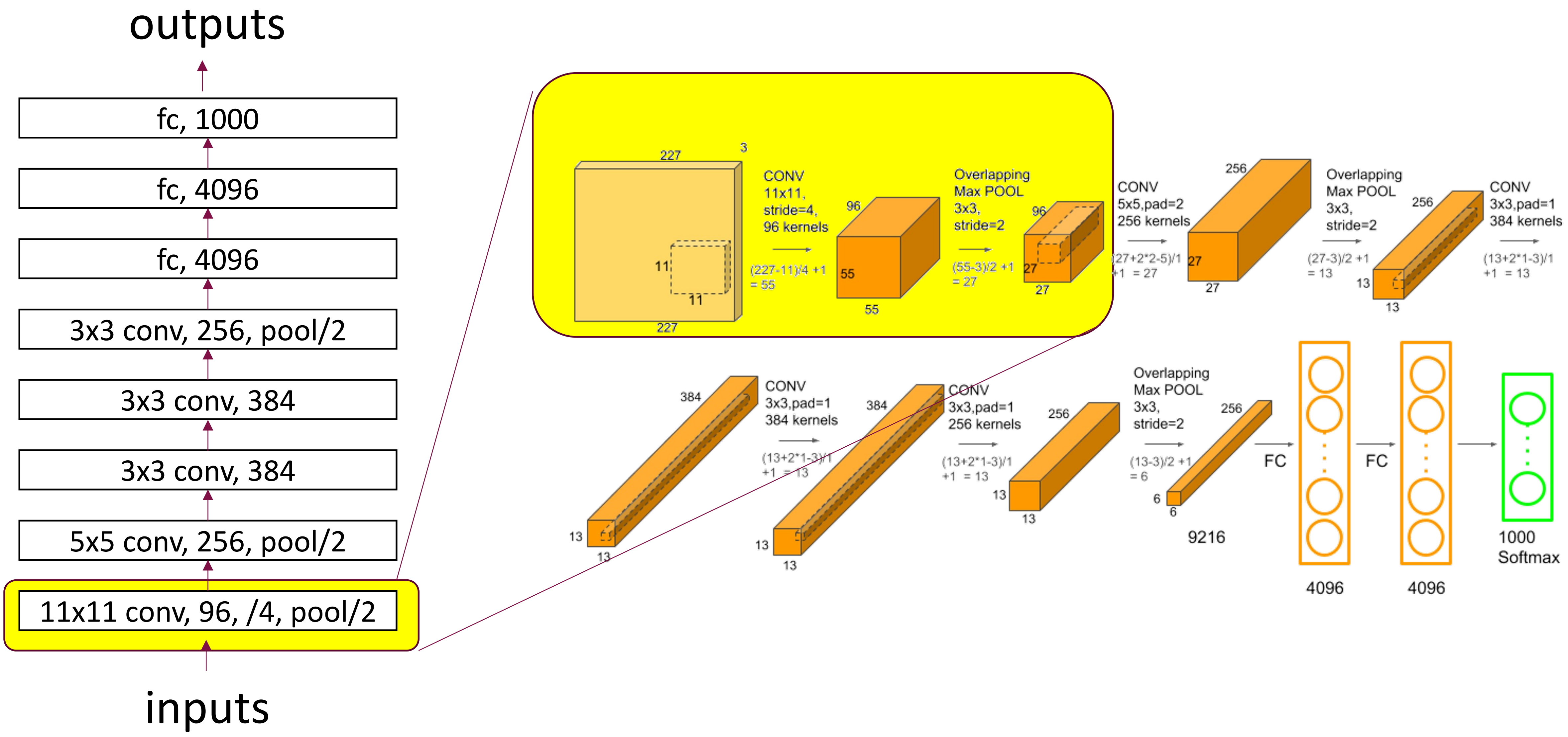


Max-pooling: *translation invariance.*
Often applied with a stride.
No learnable parameters.

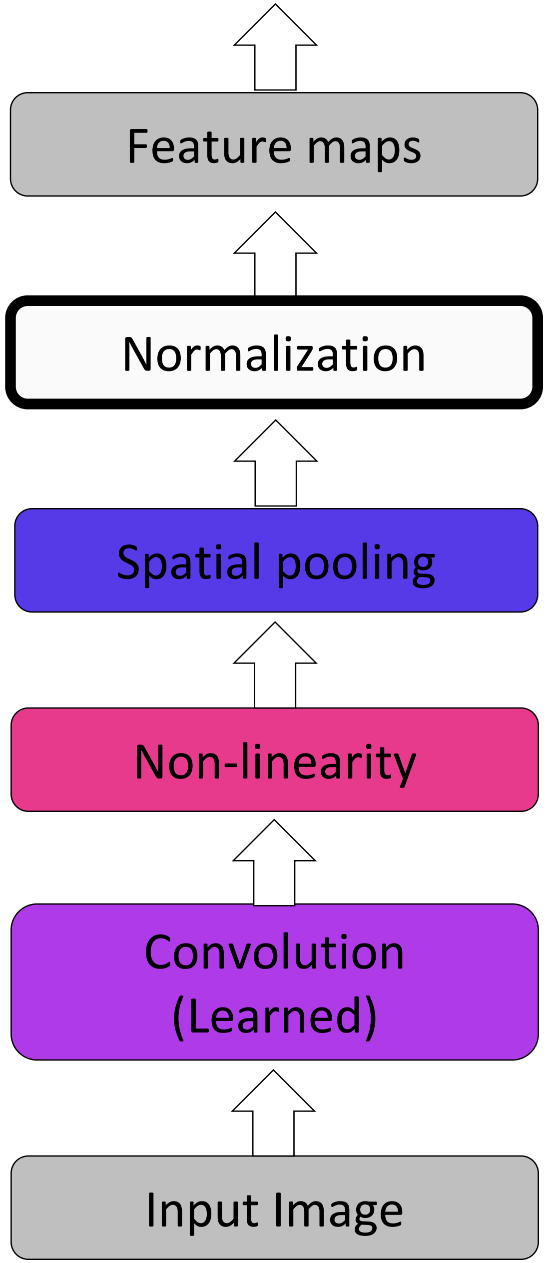
Convolution provides equivariance to shift
Pooling provides invariance to shift

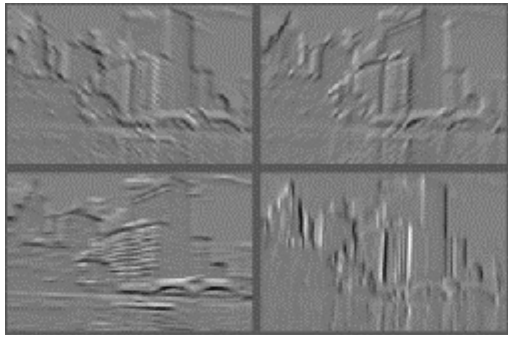
graphic credit: S. Lazebnik

Back to AlexNet

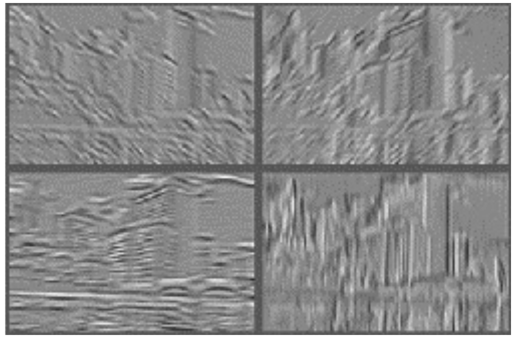


Convolve → activation function → pool → normalize





Feature Maps



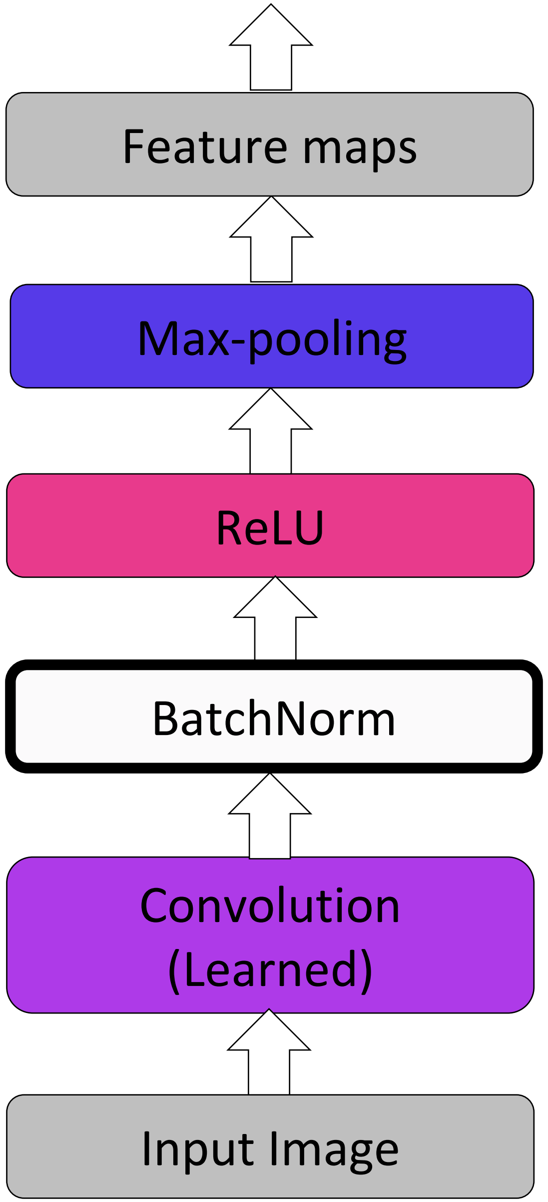
Feature Maps
After Contrast
Normalization

“Contrast normalization” highlights areas where the feature maps change. Used to be a standard component in neural networks. Not used in modern architectures.

graphic credit: S. Lazebnik

Modern variants

- BatchNorm is very commonly used.
- Most common variants of a convolutional block:
  - Conv-BatchNorm-Maxpool-ReLU, or
  - Conv-BatchNorm-ReLU-Maxpool
- Sometimes even no Maxpool, to keep feature map spatial dimensions large. Often in very deep networks.



Often, when people say “convolution layer”, it is implicit that they mean a full convolutional block with various layers following the actual convolutional layer

Back to AlexNet

outputs

fc, 1000

fc, 4096

fc, 4096

3x3 conv, 256, pool/2

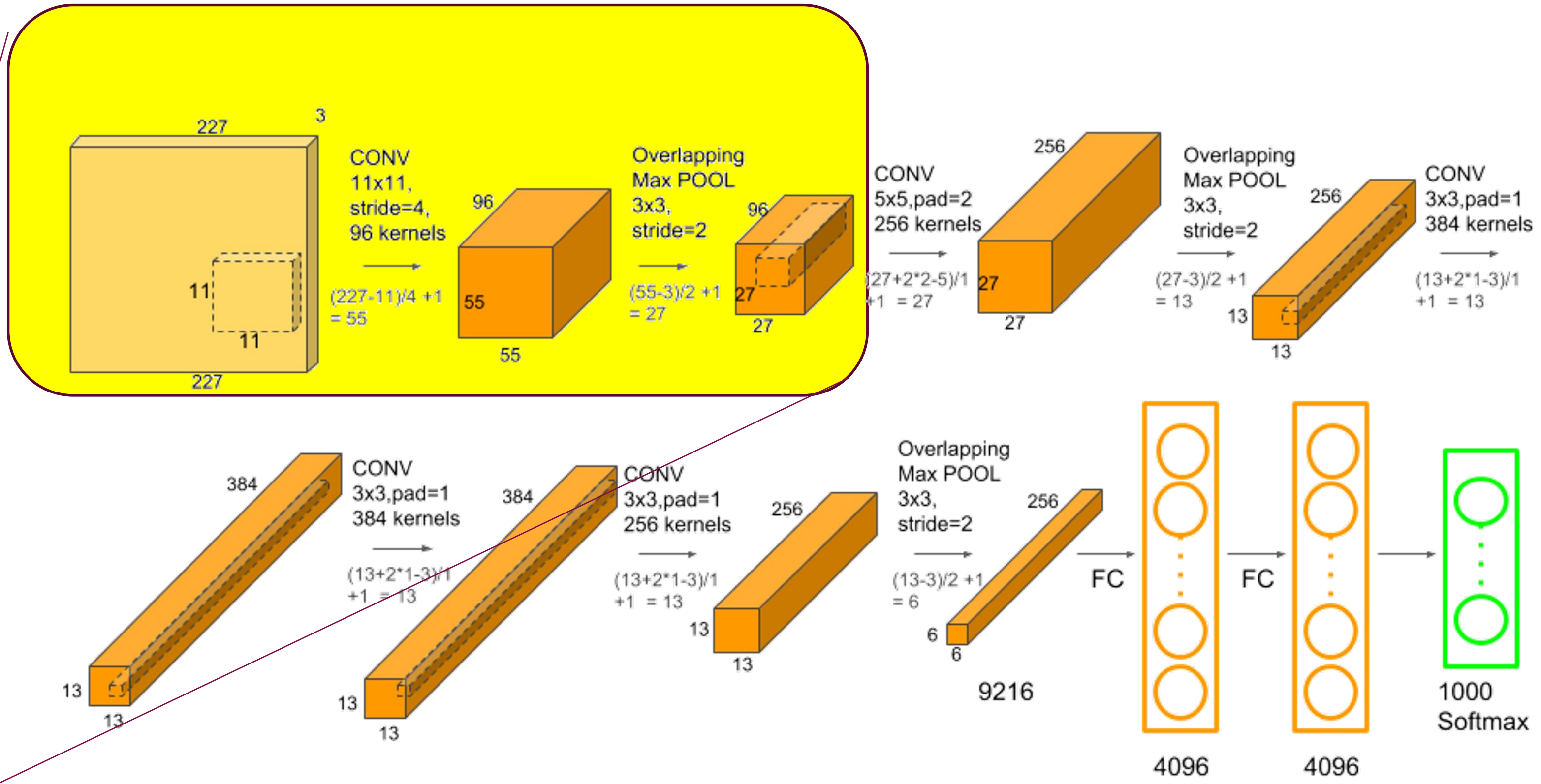
3x3 conv, 384

3x3 conv, 384

5x5 conv, 256, pool/2

11x11 conv, 96, /4, pool/2

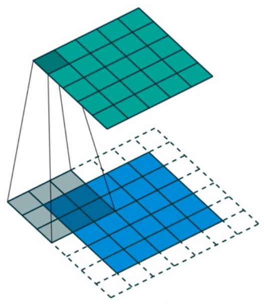
inputs



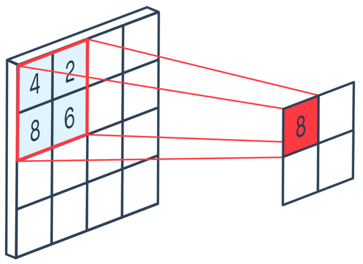
Exercise: work through the rest of the dimensions in this network!

Summary: Image-specific operations in neural nets

- Machinery to convert image matrices into vectors of reasonable dimensions, retaining useful location associations. Two main workhorses:
  - Convolution layers – Location-independent processing. Shift equivariance.
    - Convolutions produce “image”-like feature maps, which retain associations with input pixels.
  - Pooling layers – Binning to make outputs insensitive to translation and reduce dimensionality. Shift invariance.
    - A dog is a dog even if its image is shifted by a few pixels.

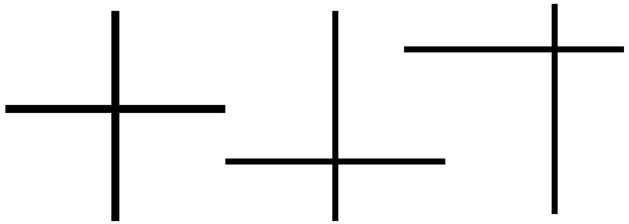


Convolution layers



Pooling layers

A Convolution Exercise



Suppose we want to find out whether the following image depicts Cartesian axes.

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix}$$

As a step towards this, we convolve the image with two filters (no padding, stride of 1).

$$\begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}, \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 1 & 1 & 1 \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

Compute the output by hand.

A Convolution Exercise

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$$

A Convolution Exercise

$$\begin{bmatrix} 0 & 1 & 0 & \\ 0 & 1 & 0 & \\ 1 & 1 & 1 & \\ & & & 2 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & \cdot \\ \cdot & \cdot \end{bmatrix}$$

$$\begin{aligned} &\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right) \\ &\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right) \\ &\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right) = 2 \end{aligned}$$

A Convolution Exercise

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix} \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -2 \\ . & . \end{bmatrix}$$

A Convolution Exercise

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & -2 & 0 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -2 \\ -1 & \cdot \end{bmatrix}$$

A Convolution Exercise

$$\begin{bmatrix} 0 & & & \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix}$$

Convolution Exercise Solution

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix}$$

$$\begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 1 & 1 & 1 \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \longrightarrow \begin{bmatrix} -1 & -\frac{1}{2} \\ \frac{7}{2} & 4 \end{bmatrix}$$

Grayscale image

Filters

Output of filters

Convolutional Exercise Solution

Next, what happens if we run max-pooling on the filter outputs?

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix}$$

Grayscale image

$$\begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \qquad \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 1 & 1 & 1 \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

Filters

$$\begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -\frac{1}{2} \\ \frac{7}{2} & 4 \end{bmatrix}$$

Output of filters

$$\begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Pretty clear vertical line
and
Pretty clear horiz line

Output of max-pool

Example architectures

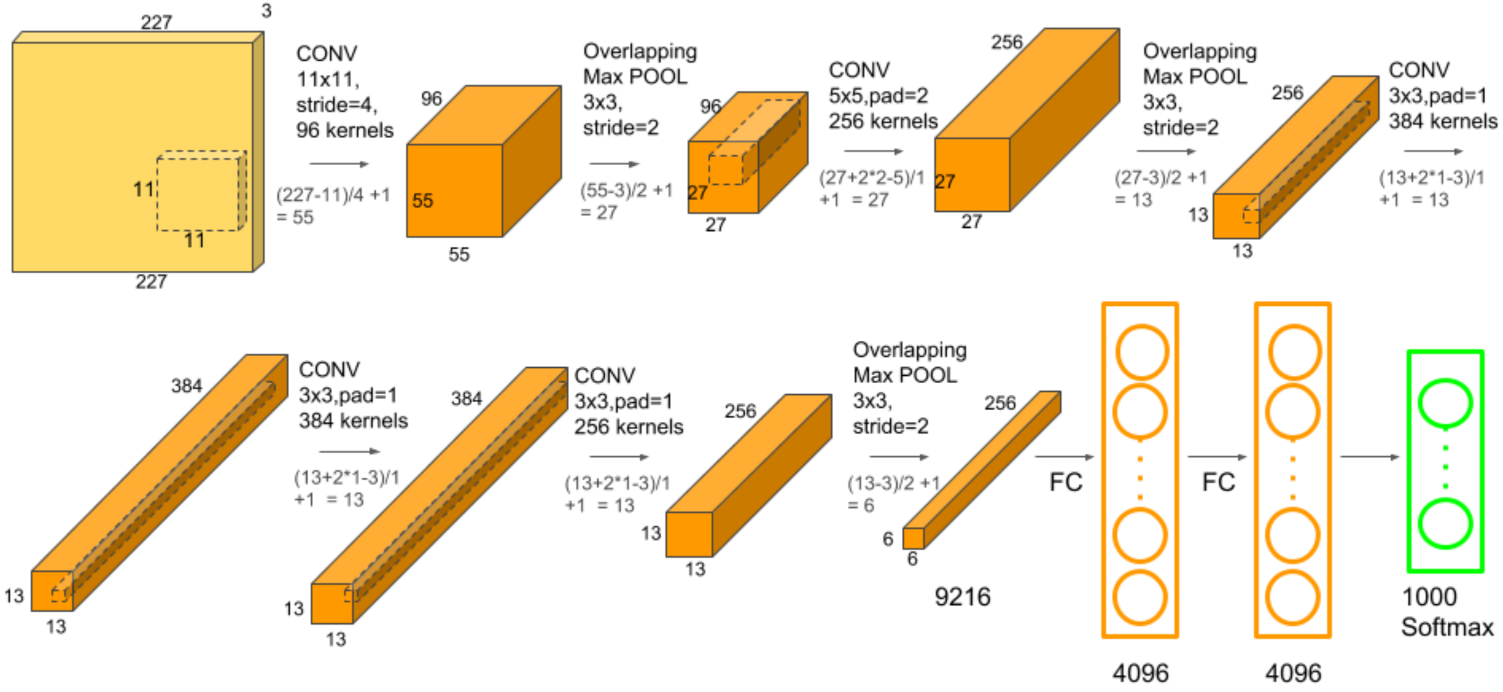
Example architectures

AlexNet, 8 layers
(ILSVRC 2012)

~60M params

"Standard" scheme
[Conv-ReLU-pool?]
[Conv-ReLU-pool?]
[Conv-ReLU-pool?]
…
Fully connected
…
Fully connected

11x11 conv, 96, /4, pool/2
↓
5x5 conv, 256, pool/2
↓
3x3 conv, 384
↓
3x3 conv, 384
↓
3x3 conv, 256, pool/2
↓
fc, 4096
↓
fc, 4096
↓
fc, 1000



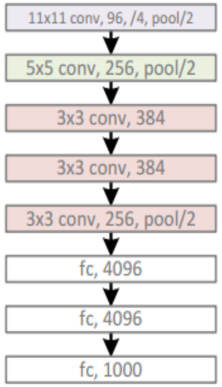
[Source: MSRA slides at ILSVRC15](#)

https://learnopencv.com/understanding-alexnet/

Example architectures

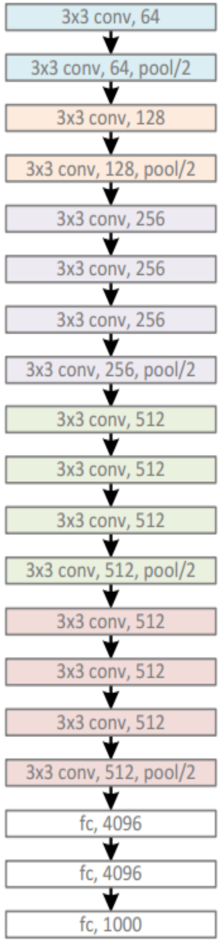
AlexNet, 8 layers
(ILSVRC 2012)

~60M params



VGG, 19 layers
(ILSVRC 2014)

~140M params



“Standard” scheme
[Conv-ReLU-pool?]
[Conv-ReLU-pool?]
[Conv-ReLU-pool?]
…
Fully connected
…
Fully connected

ICCV15

Source: MSRA slides at ILSVRC15

Example architectures

AlexNet, 8 layers
(ILSVRC 2012)
~60M params

VGG, 19 layers
(ILSVRC 2014)
~140M params

ResNet, 152 layers
(ILSVRC 2015)

Less computation in forward pass than VGGNet!

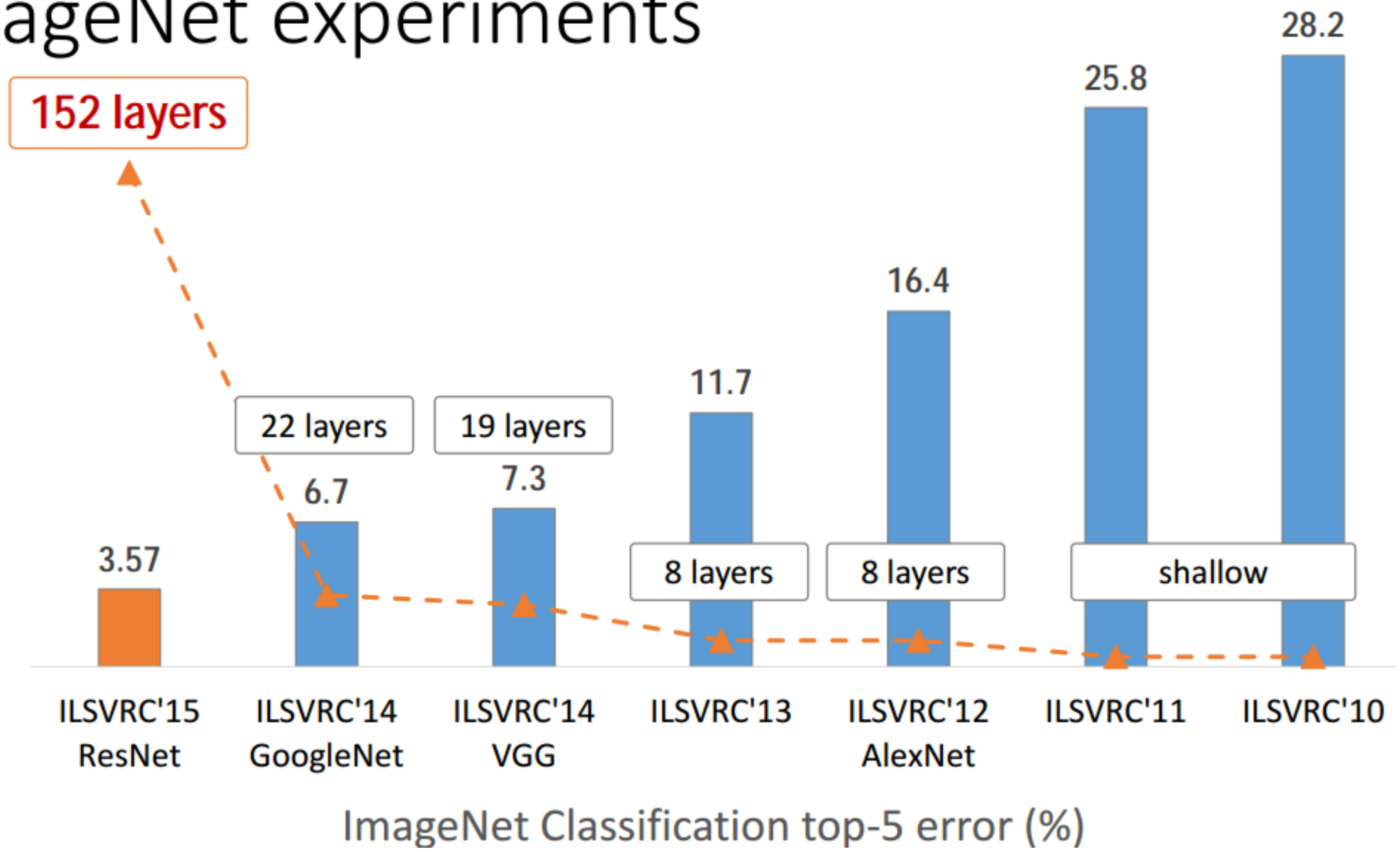
Back to 60M params

GoogleNet, 22 layers
(ILSVRC 2014)
~5M params

ICCV15

[Source: MSRA slides at ILSVRC15]

ImageNet experiments

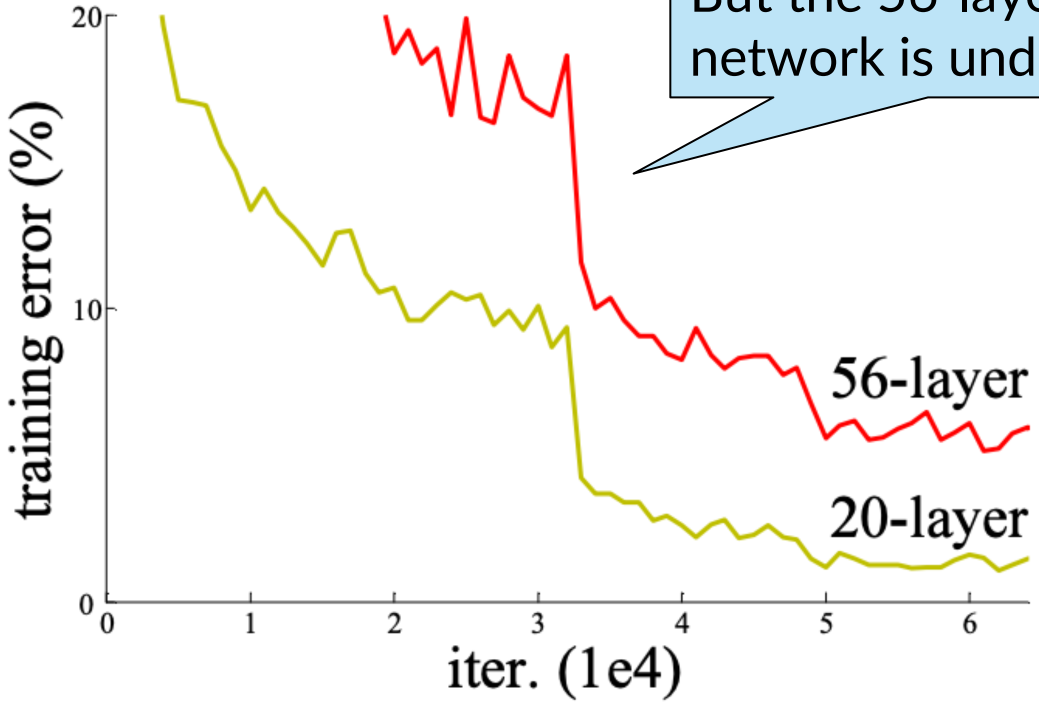


[Source: MSRA slides at ILSVRC15]

Residual Network

- Q: Why are deeper networks not always better?
- Hypothesis 1: Because of overfitting.

But the 56-layer network is underfitting!



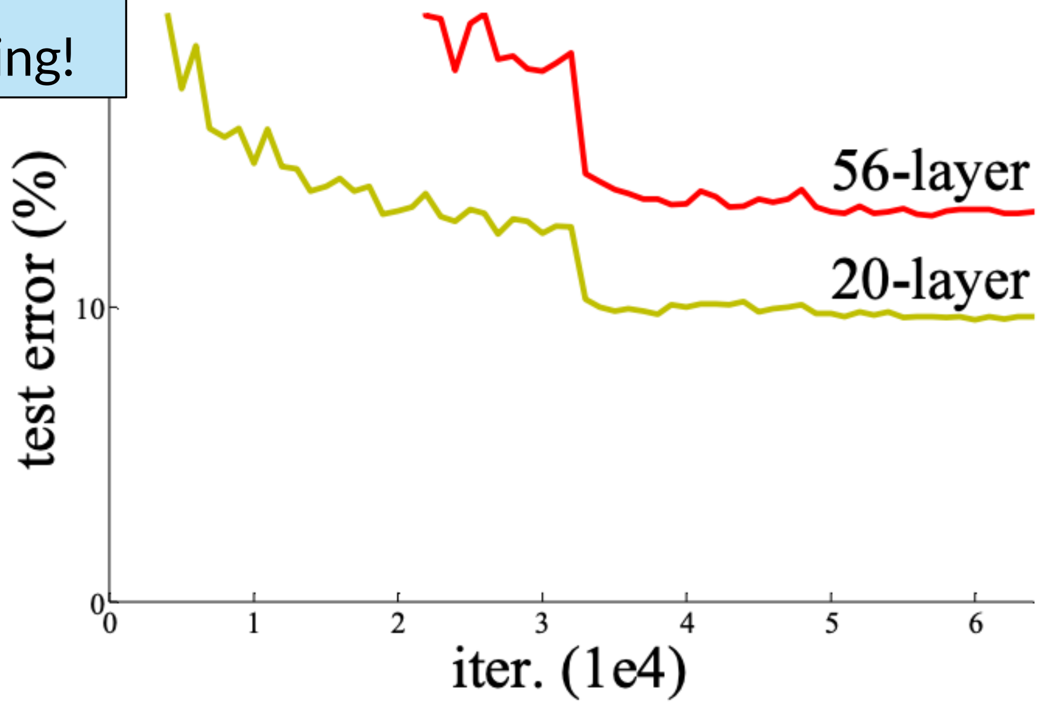


Image credit: He et al, Residual Nets, 2015

Residual Network

- Q: Why are deeper networks not always better?
- Hypothesis 2: Because of optimization issues with deeper networks.

Idea: *Skip connections* that facilitate more direct feedback from the loss to the weights.

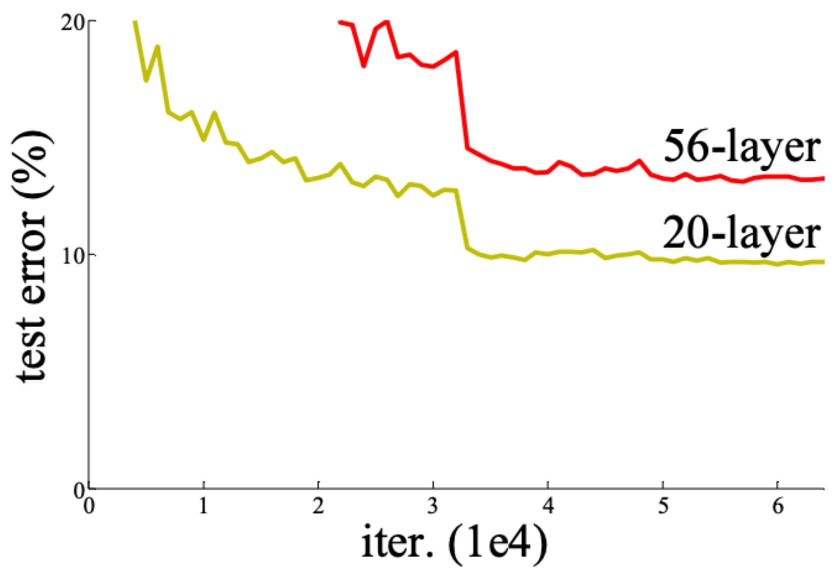


Image credit: He et al, Residual Nets, 2015

Residual Network

Two views of residual connections:

1. Providing shortcuts to gradients on the backward pass.
2. Allowing each "residual block" to fit the residual error function (recall gradient boosting!)

$$F(x) = H(x) - x.$$

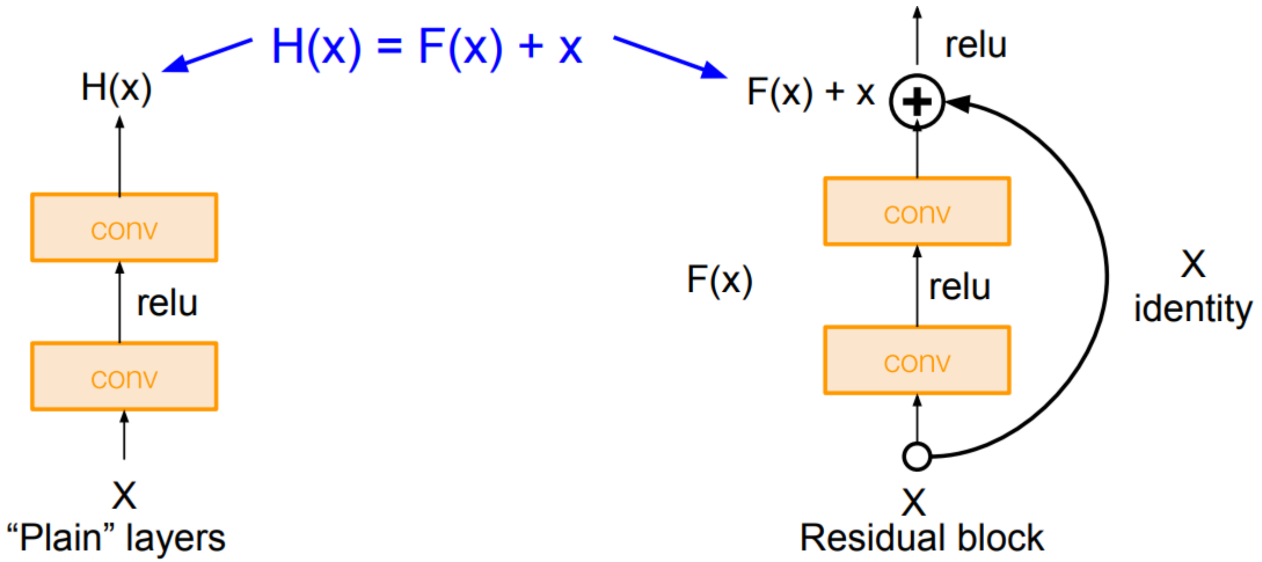


Image credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Residual Network

- Stack lots of residual blocks.
- Zero-padded stride-1 3x3 convolutions + no max-pooling ⇒ maintains feature map size to build very deep nets.
- Reduce dimensions through stride 2 once every K blocks, increase #channels.

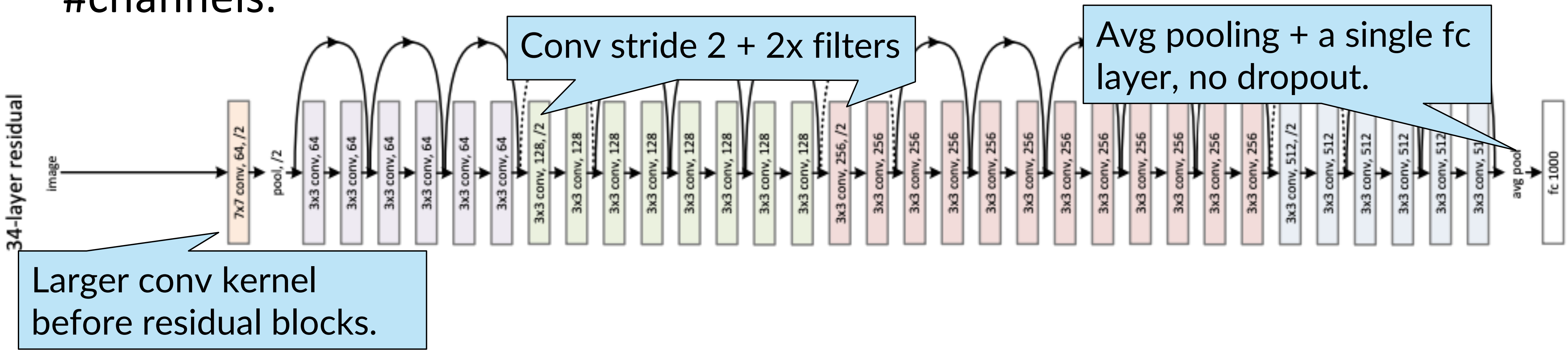


Image credit: He et al, Residual Nets, 2015

Residual block designs

- For deeper networks, improve efficiency through 1x1 convolutions.

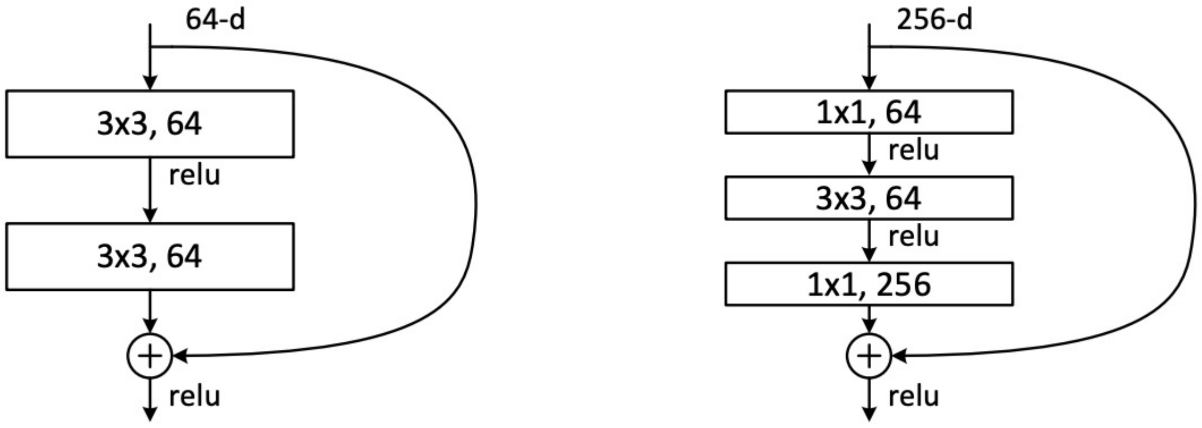


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

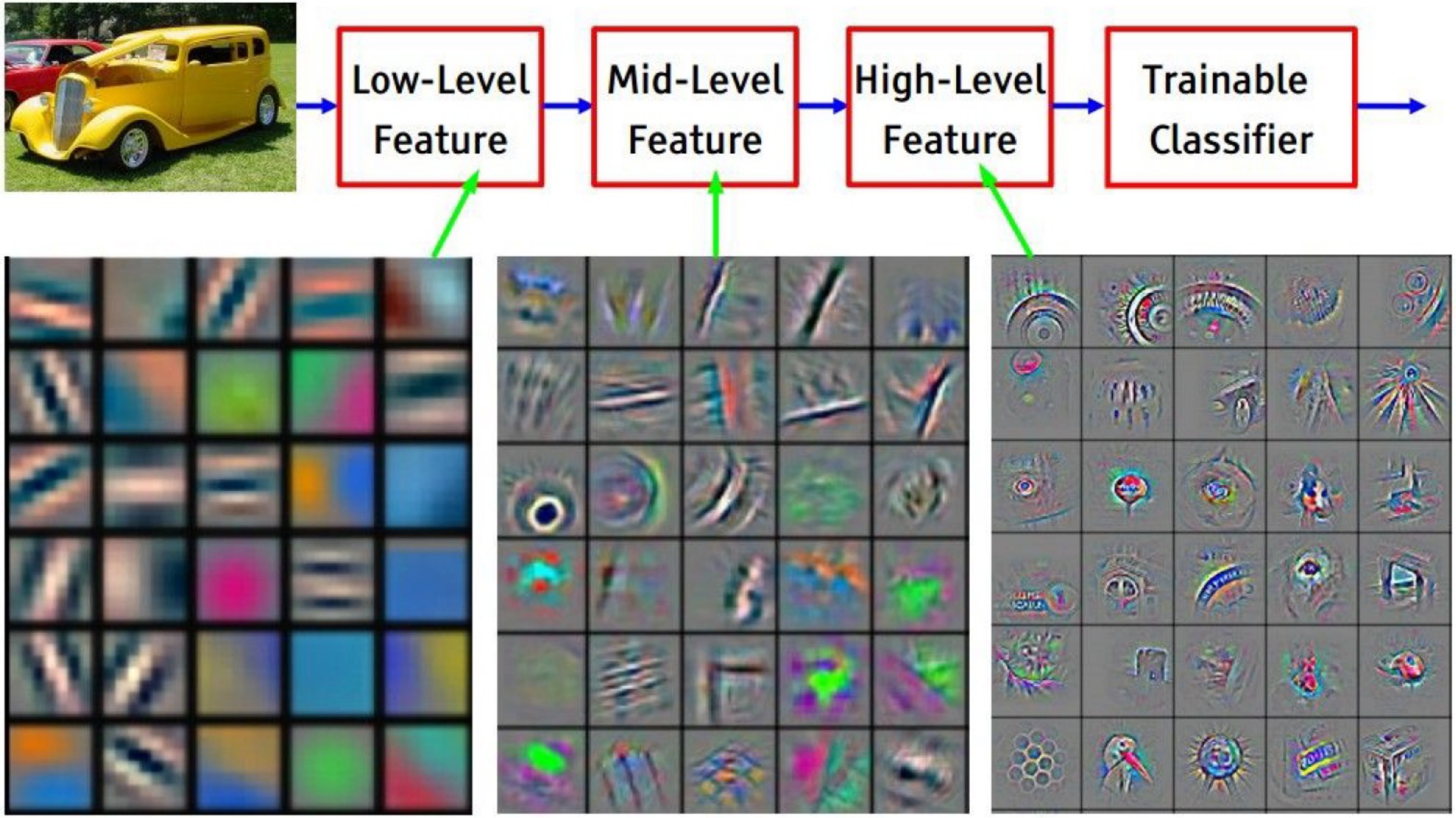
Many other improvements since 2015! E.g. “ResNeXt”, “Identity Mappings”, “ConvNeXt” etc.

Image credit: He et al, Residual Nets, 2015

What do CNNs learn?

Visualizing and Understanding CNNs

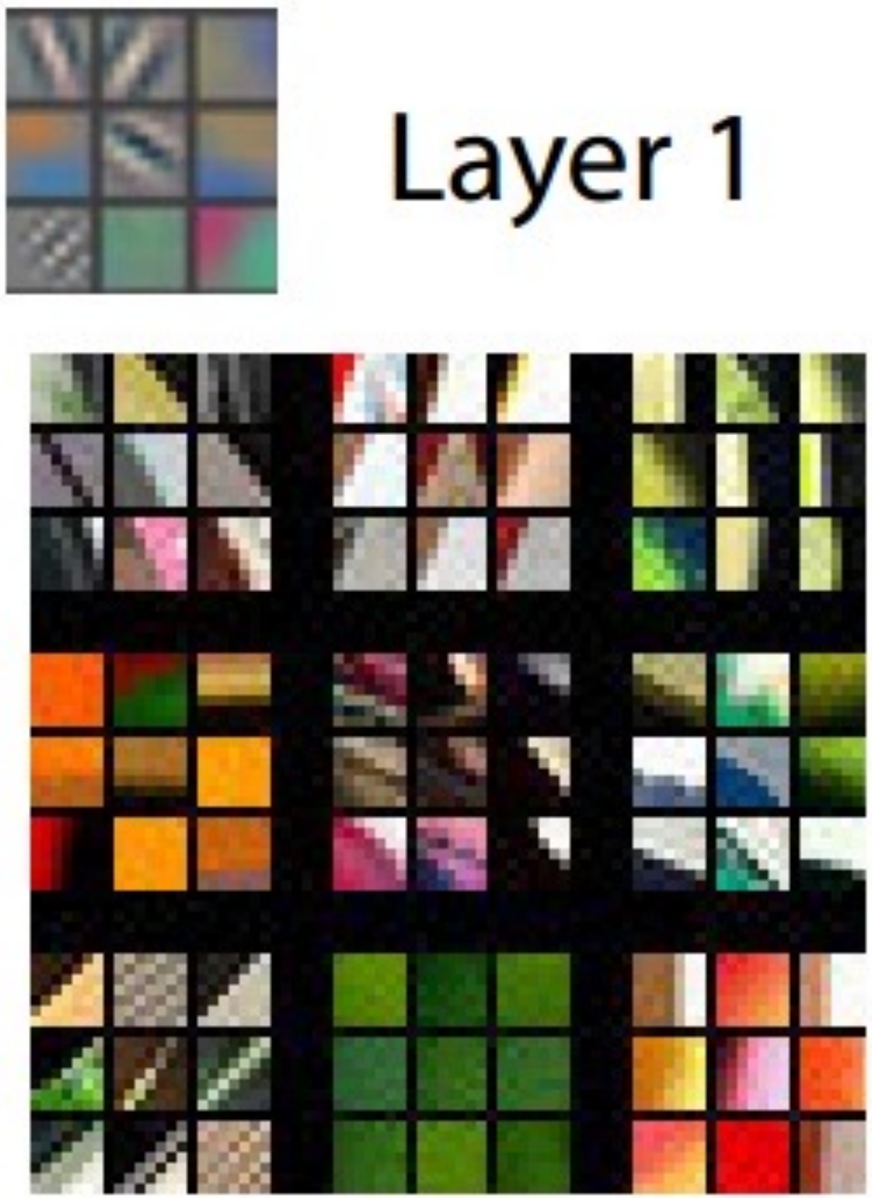
Feature visualization



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Yann LeCun

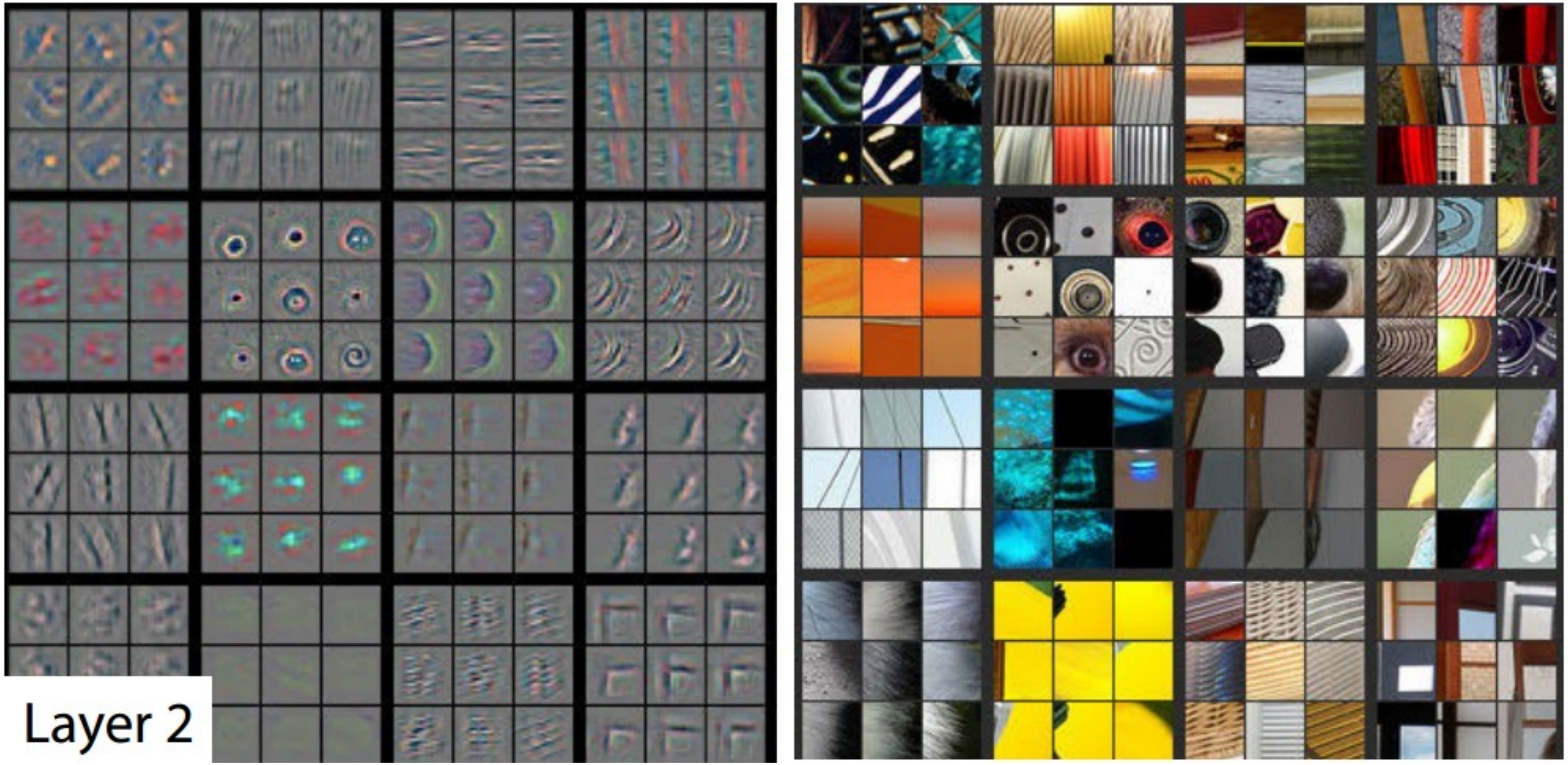
Layer 1



Slide credit: Jia-Bin Huang

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

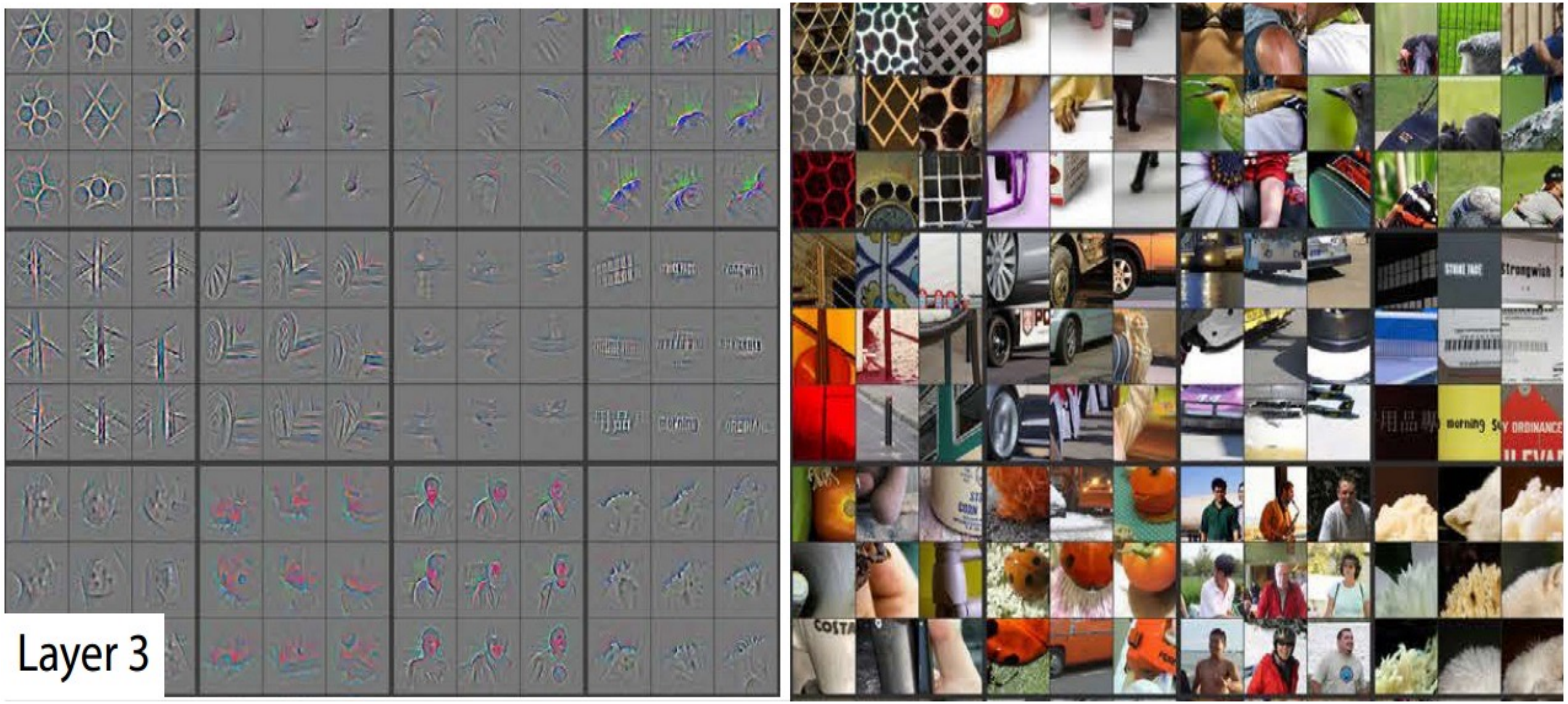
Layer 2



Slide credit: Jia-Bin Huang

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

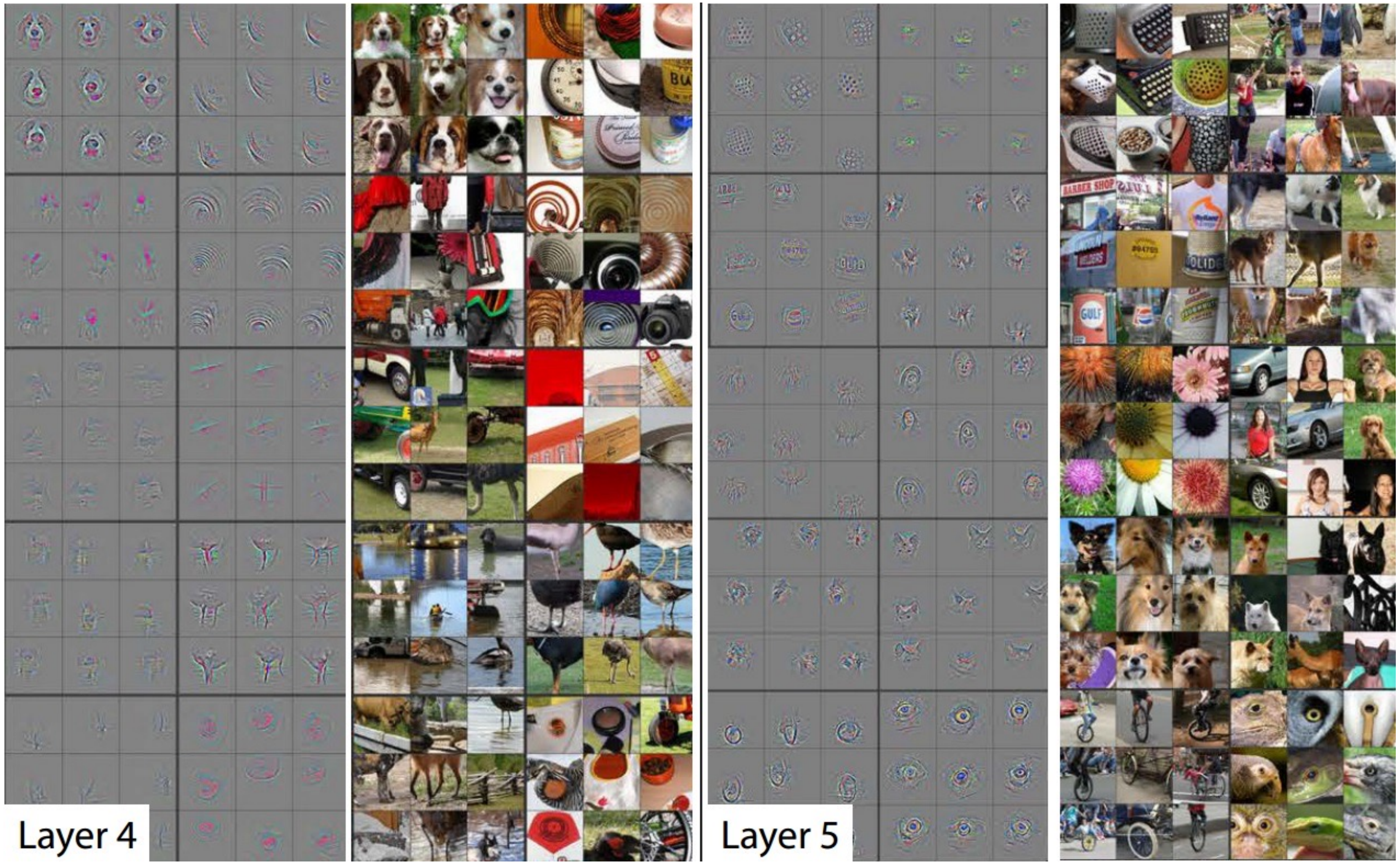
Layer 3



Slide credit: Jia-Bin Huang

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

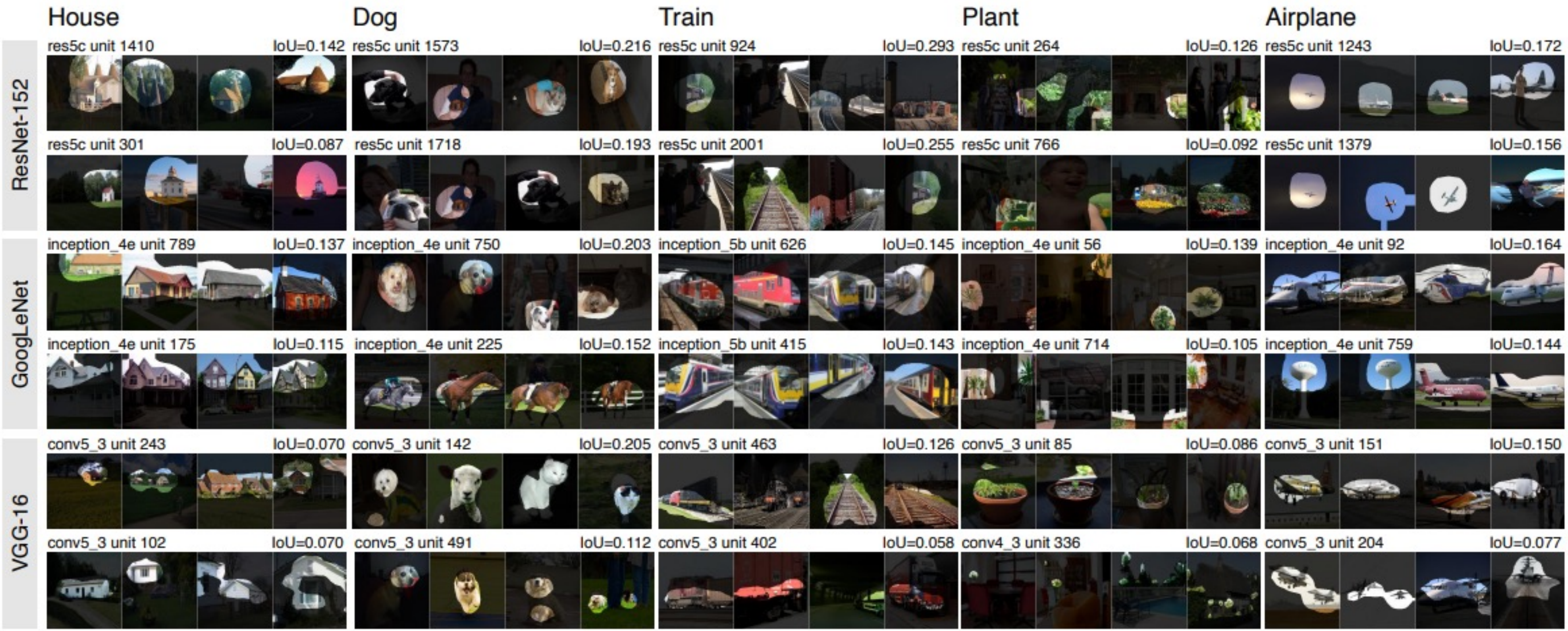
Layer 4 and 5



Slide credit: Jia-Bin Huang

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Network dissection



http://netdissect.csail.mit.edu/

CNNs with small datasets

Can we reuse trained concepts?

Since CNN’s trained for ImageNet object category classification appear to learn many apparently general features, why not reuse these models in some way to perform new tasks?

Transfer learning with CNNs

What if your task doesn't have Imagenet-sized data?

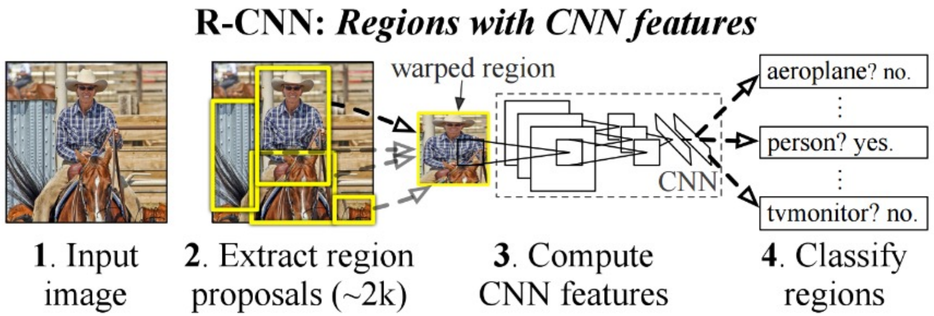
For tasks close to original task, can make do with small datasets + feature extraction or shallow finetuning.

For tasks far from original task, you will need to use moderate-sized datasets + deeper finetuning

Slide credit: Fei-Fei Li and Andrej Karpathy

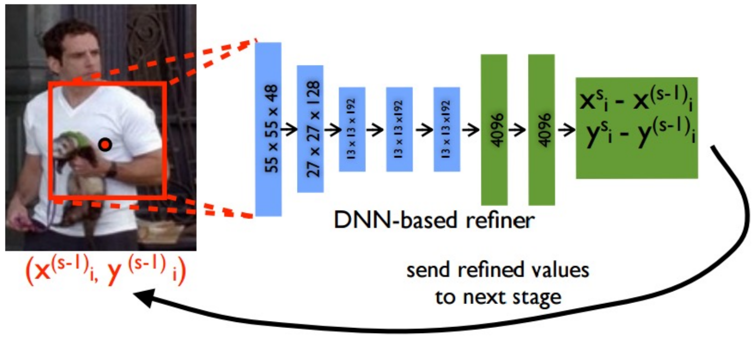
Some sample applications

[Girshick et al. CVPR14]



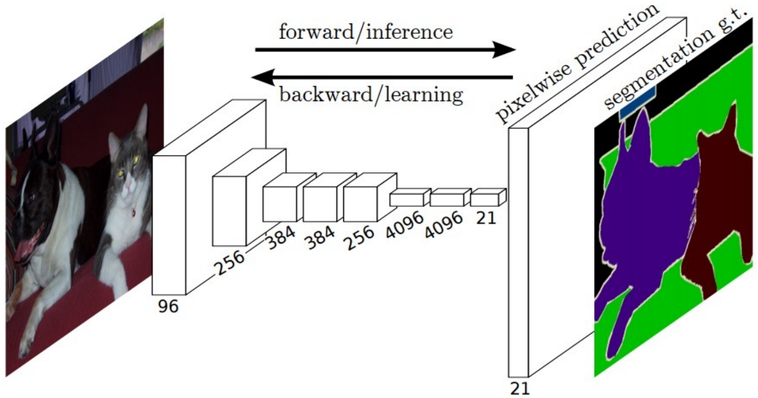
Object detection

[Toshev et al. CVPR14]



Pose detection (regression)

[Long et al. CVPR15]

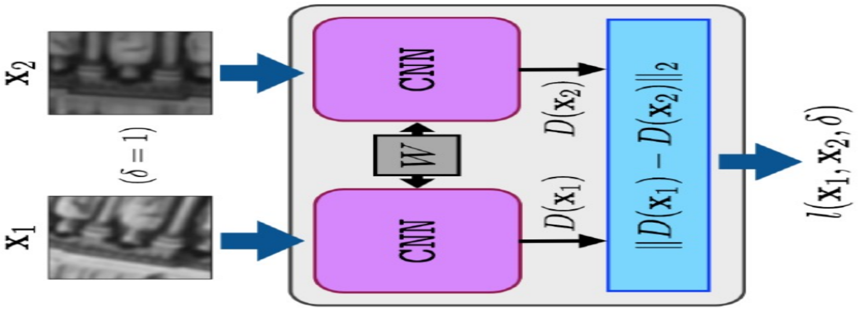


Semantic segmentation

Examples courtesy Jia-Bin Huang

Some sample applications

[Chopra et al. CVPR05]



Picture source

Similarity metric learning

[Dosovitskiy et al. CVPR15]

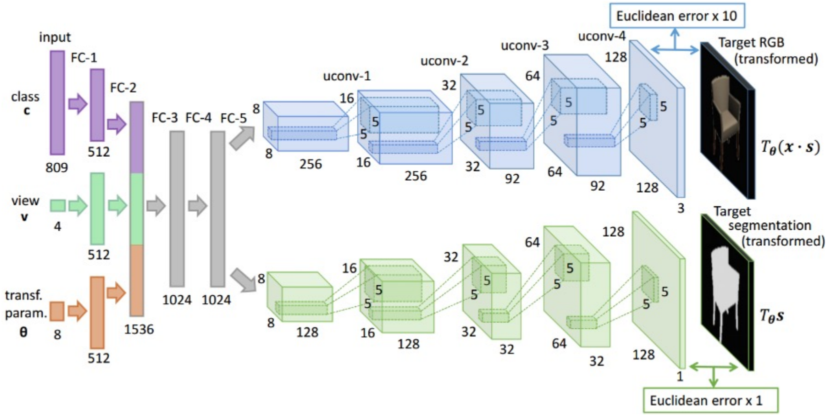
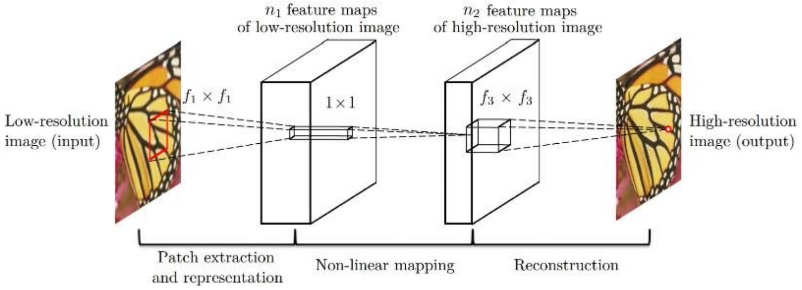


Image generation

[Dong et al. ECCV 2014]



Low-level image processing: (superresolution, deblurring, image quality etc.)

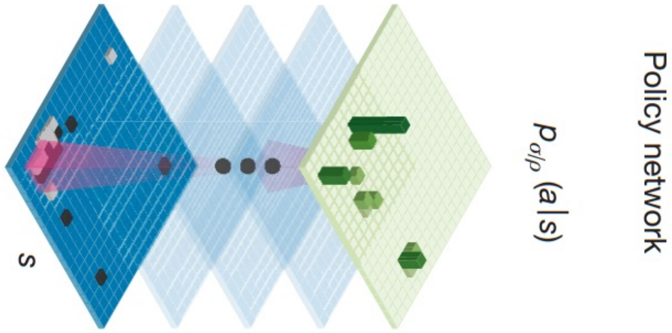
Examples courtesy Jia-Bin Huang

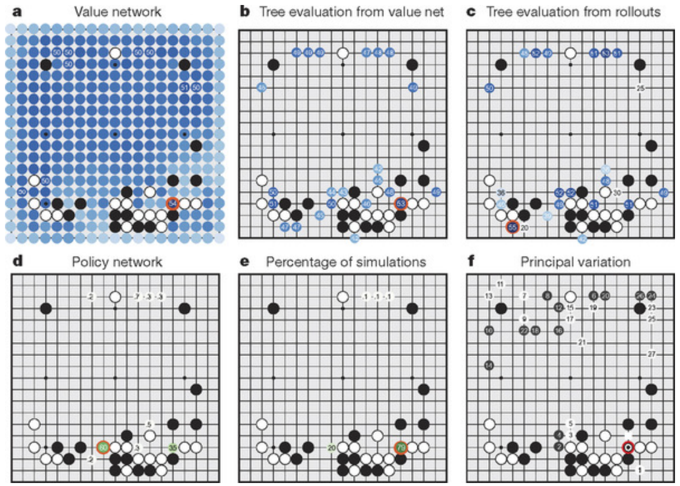
Game playing!

CNN + Reinforcement learning

[Mnih et al, Nature' 15]

[Silver et al, Nature '16]





ConvNet Art!

See if you can tell artists' originals from machine style imitations at: http://turing.deepart.io/

Paper: Gatys et al, "Neural ... Style", arXiv '15
Code (torch): https://github.com/jcjohnson/neural-style

Pytorch Training Loop

Pytorch Training Loop

```python
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

Looping over mini-batches

Zero out all old gradients

Runs forward pass model.forward(data)

Loss computation

Backpropagation

Gradient step

Pytorch Training Loop

```python
def main():
    torch.manual_seed(1)
    device = torch.device("cuda")
    train_loader = torch.utils.data.DataLoader(
        datasets.MNIST('../data', train=True, download=True,
                       transform=transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize((0.1307,), (0.3081,))
                       ])),
        batch_size=64, shuffle=True)

    model = Net().to(device)
    optimizer = optim.Adam(model.parameters(), lr=1e-4)
    sc[illegible]e=1, gamma=0.9)
    for epoch in range(1, 15):
        train(model, device, train_loader, optimizer, epoch)
        scheduler.step()
```

Load dataset

Loop over epochs (full passes over data)

Minibatch SGD for one epoch

Update base learning rate

Pytorch Model

- To use your model (once it has been trained):

```
model.eval()      # puts model in evaluation mode
label = model(input)    # forward pass to compute
outputs
```