# Announcements

- HW 3 due **Tonight (Wednesday, October 19) at 8pm**
  - HW 4 posted tonight, due **Wednesday, November 2**

- Quiz 6 due **Tomorrow (Thursday, October 20) at 8pm**

# Lecture 14: Computer Vision (Part 2)

CIS 4190/5190

Fall 2022

# Agenda

- **Convolutional & pooling layers**
- **Convolutional neural networks**
- **Feature visualization**
- **Applications**

# Images as 2D Arrays

- Grayscale image is a 2D array of pixel values

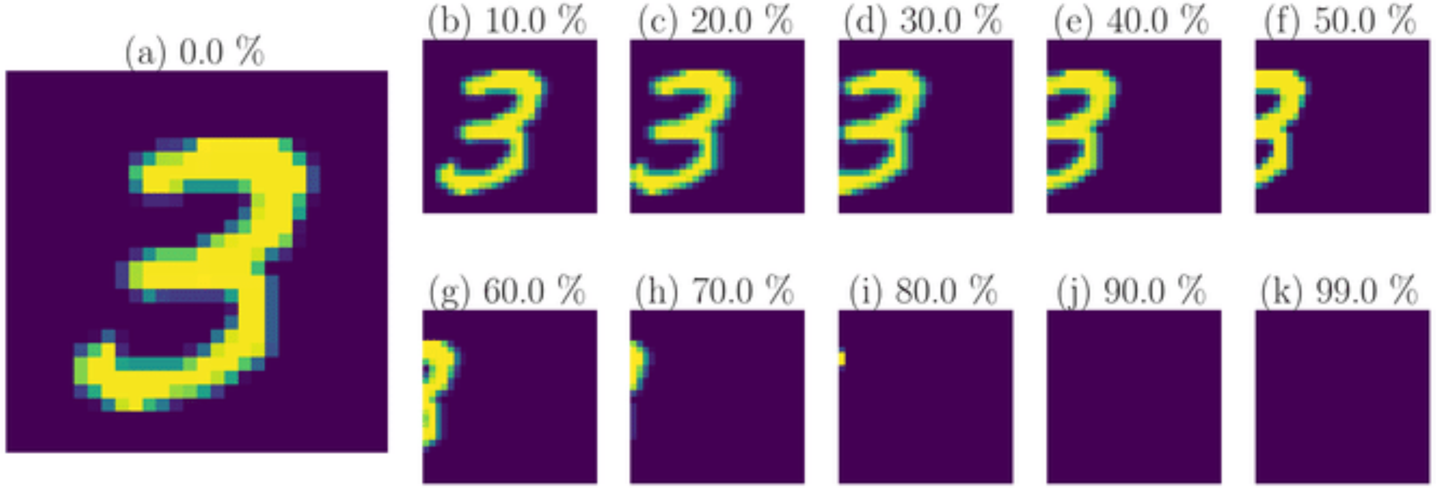
- Color images are 3D array
  - 3rd dimension is color (e.g., RGB)
  - Called "channels"

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 6 | 9 | 8 |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 |
| 5 | 2 | 3 | 0 | 1 | 2 | 3 | 4 | 5 |
| 4 | 3 | 2 | 1 | 0 | 3 | 2 | 5 | 4 |
| 7 | 4 | 5 | 2 | 3 | 0 | 1 | 2 | 3 |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 | 3 | 2 |
| 9 | 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Source: S. Narasimhan, S. Lazebnik

# Structure in Images

- **Translation invariance**
  - Consider image classification (e.g., labels are cat, dog, etc.)
  - **Invariance:** If we translate an image, it does not change the category label



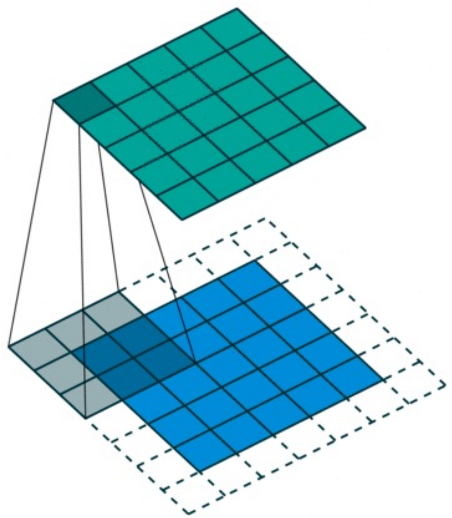

**Source:** Ott et al., Learning in the machine: To share or not to share?

# Structure in Images

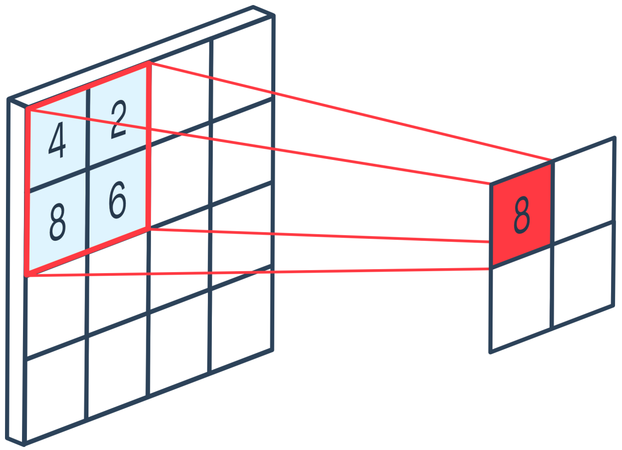
- **Translation equivariance**
  - Consider object detection (e.g., find the position of the cat in an image)
  - **Equivariance:** If we translate an image, the the object is translated similarly

# Structure in Images

- Use layers that capture structure



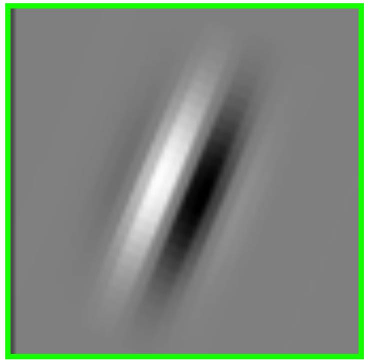
**Convolution layers**
(Capture equivariance)

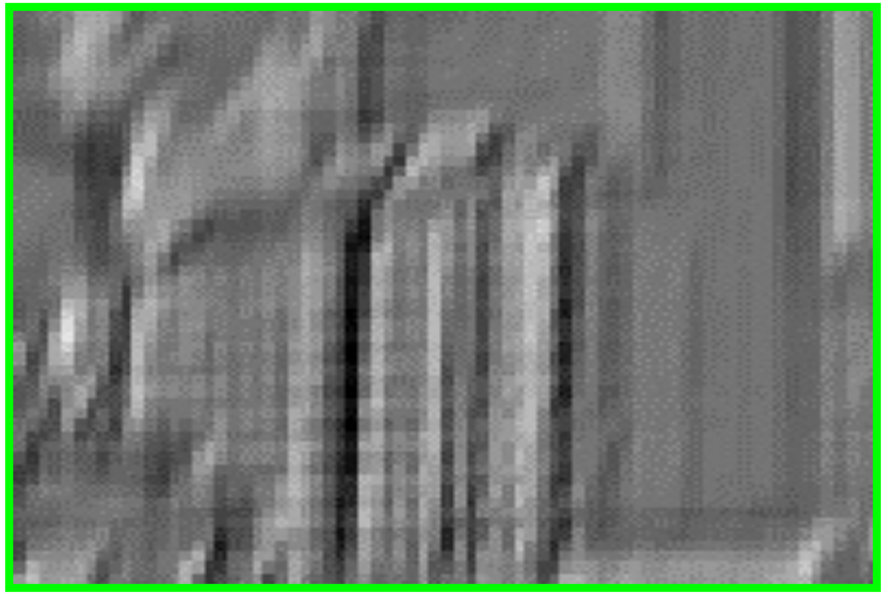



**Pooling layers**
(Capture invariance)

https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
https://peltarion.com/static/2d_max_pooling_pa1.png

# Convolution Filters





$$\text{output}[i,j] = \sum_{\tau=0}^{k-1}\sum_{\gamma=0}^{k-1} \text{filter}[\tau,\gamma]\cdot\text{image}[i+\tau,j+\gamma]$$

graphic credit: S. Lazebnik

# 2D Convolution Filters

- **Given:**
  - A 2D input $x$
  - A 2D $h \times w$ kernel $k$

- The 2D convolution is:

$$y[s,t] = \sum_{\tau=0}^{h-1} \sum_{\gamma=0}^{w-1} k[\tau,\gamma] \cdot x[s+\tau, t+\gamma]$$

# 2D Convolution Filters

| | | |
|---|---|---|
| -1 | -1 | -1 |
| 2 | 2 | 2 |
| -1 | -1 | -1 |

Horizontal lines

| | | |
|---|---|---|
| -1 | 2 | -1 |
| -1 | 2 | -1 |
| -1 | 2 | -1 |

Vertical lines

| | | |
|---|---|---|
| -1 | -1 | 2 |
| -1 | 2 | -1 |
| 2 | -1 | -1 |

45 degree lines

| | | |
|---|---|---|
| 2 | -1 | -1 |
| -1 | 2 | -1 |
| -1 | -1 | 2 |

135 degree lines

Example Edge Detection Kernels

Result of Convolution with Horizontal Kernel

https://aishack.in/tutorials/image-convolution-examples/

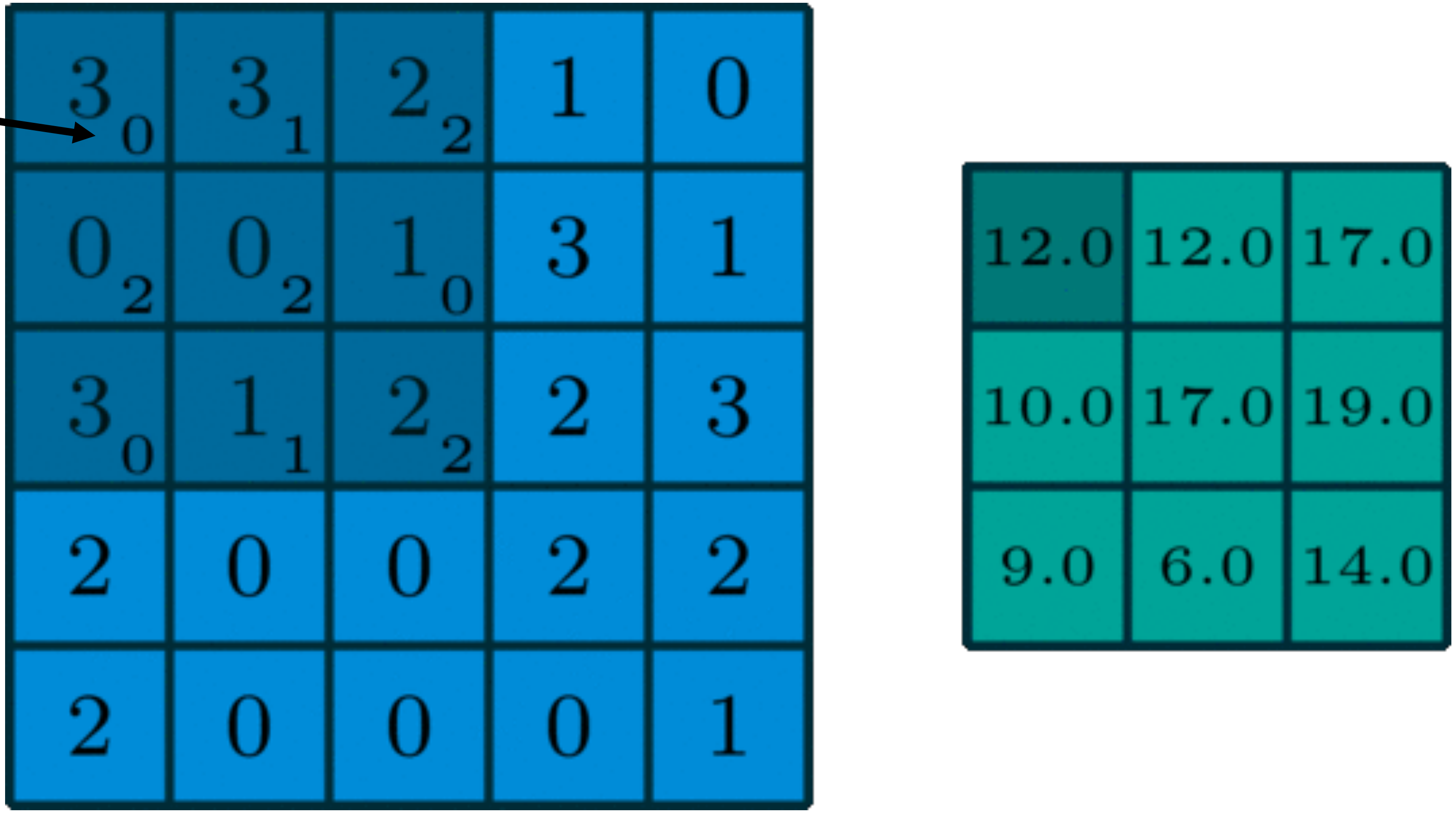
# 2D Convolution Filters

- Historically (until late 1980s), kernel parameters were handcrafted
  - E.g., “edge detectors”

- In convolutional neural networks, they are learned
  - Essentially a linear layer with fewer “connections”
  - Backpropagate as usual!

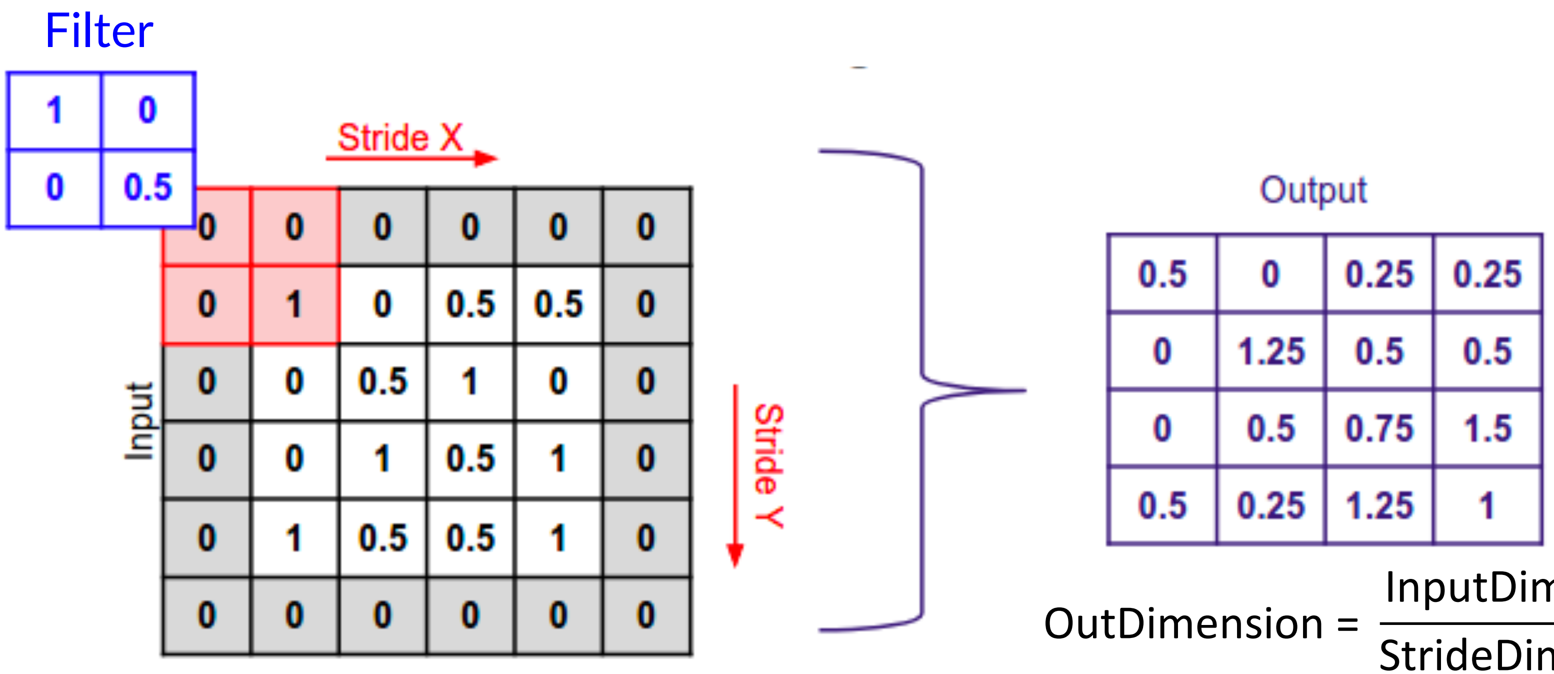
# Convolution Layers

Learnable parameters



# Convolution Layer Parameters

- **Stride:** How many pixels to skip (if any)
  - **Default:** Stride of 1 (no skipping)

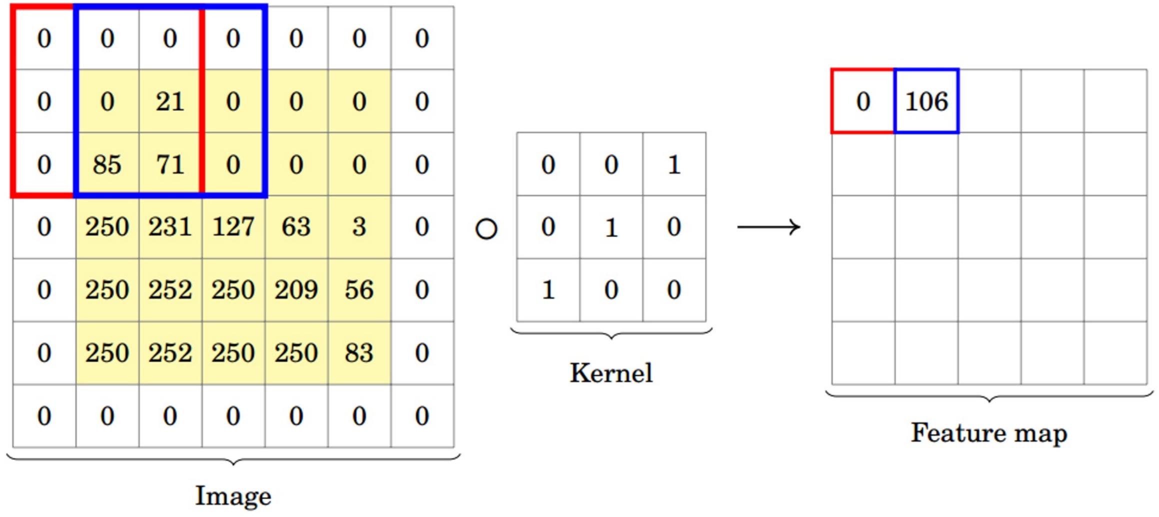



$$\text{OutDimension} = \frac{\text{InputDimension}}{\text{StrideDimension}}$$

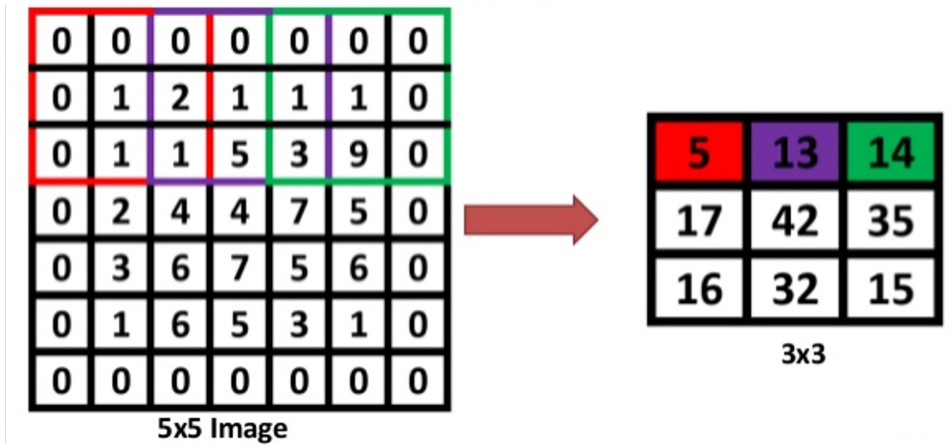
https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

# Convolution Layer Parameters

- **Padding:** Add zeros to edges of image to capture ends
  - **Default:** No padding




stride = 1, zero-padding = 1




stride = 2, zero-padding = 1

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

# Convolution Layer Parameters

- **Summary:** Hyperparameters
  - Kernel size
  - Stride
  - Amount of zero-padding
  - Output channels

- Together, these determine the relationship between the input tensor shape and the output tensor shape

- Typically, also use a single bias term for each convolution filter

# Convolution Layers

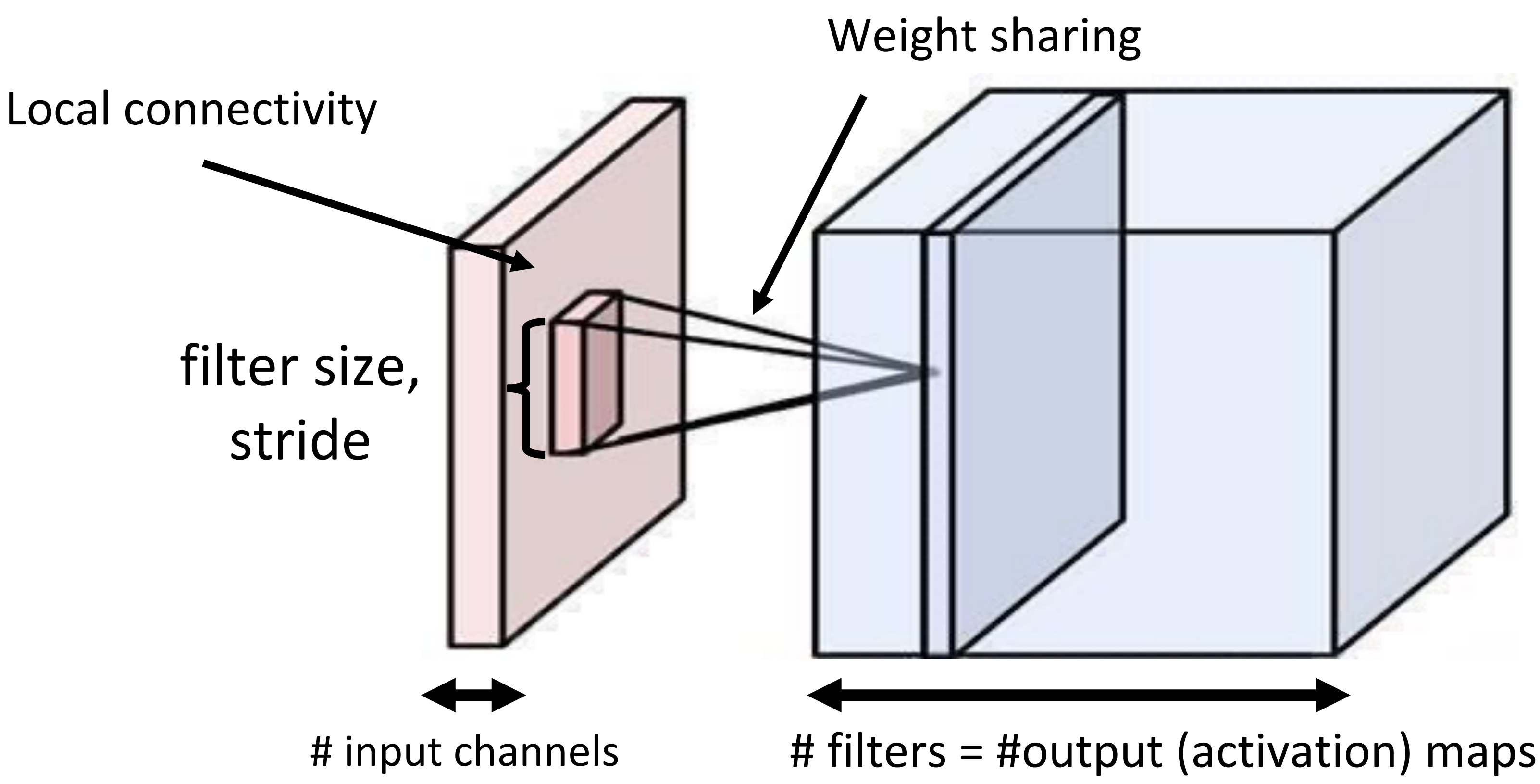
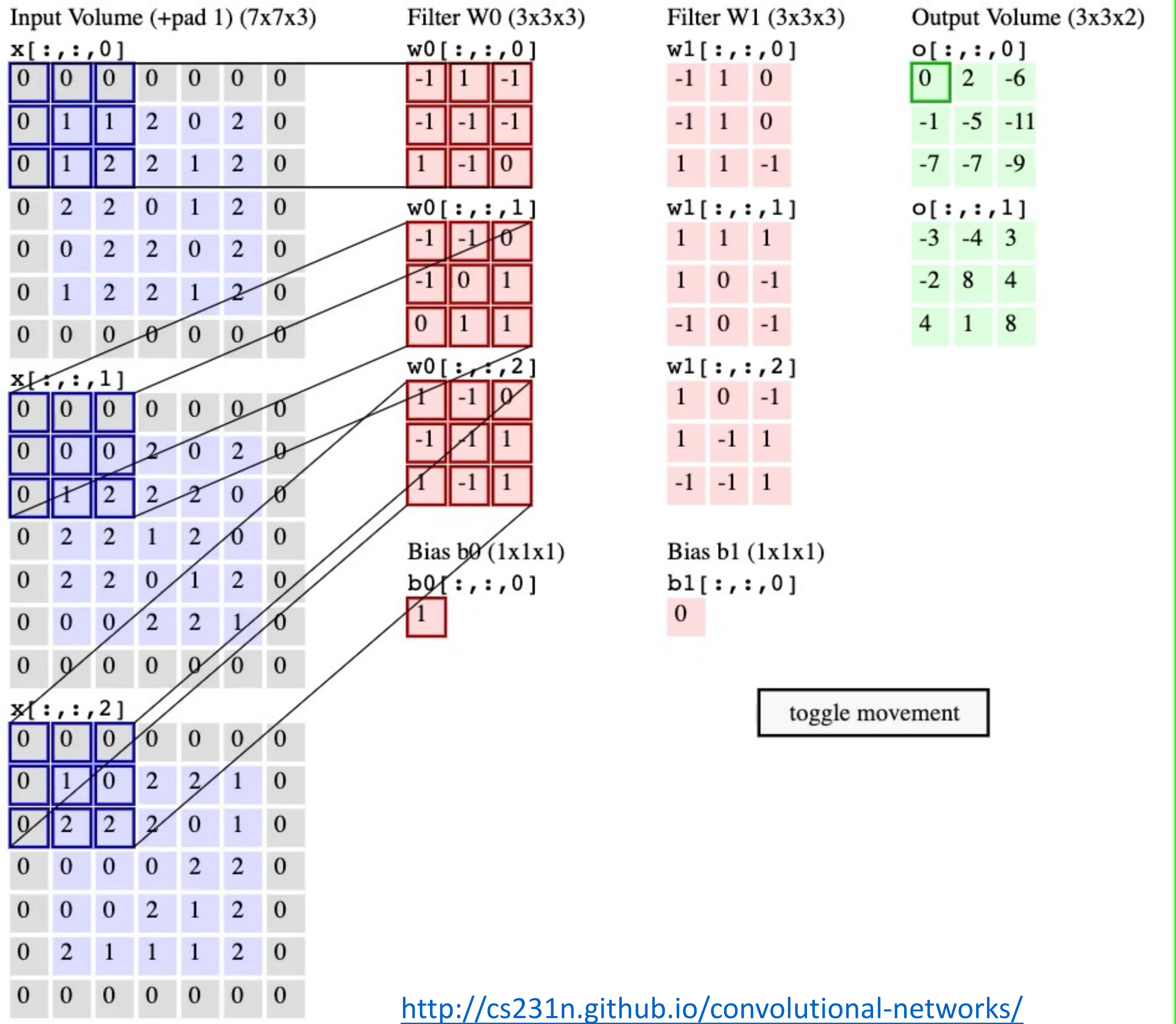



Image credit: A. Karpathy Slide credit: Jia-Bin Huang

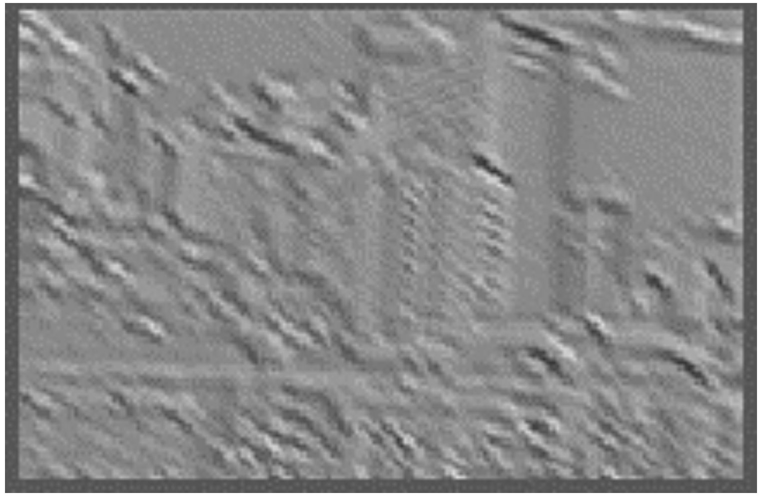
# Example

- Kernel size 3, stride 2, padding 1
- 3 input channels
  - Hence kernel size 3×3×3
- 2 output channels
  - Hence 2 kernels
- Total # of parameters:
  - $(3\times3\times3 + 1)\times2 \;=\; 56$

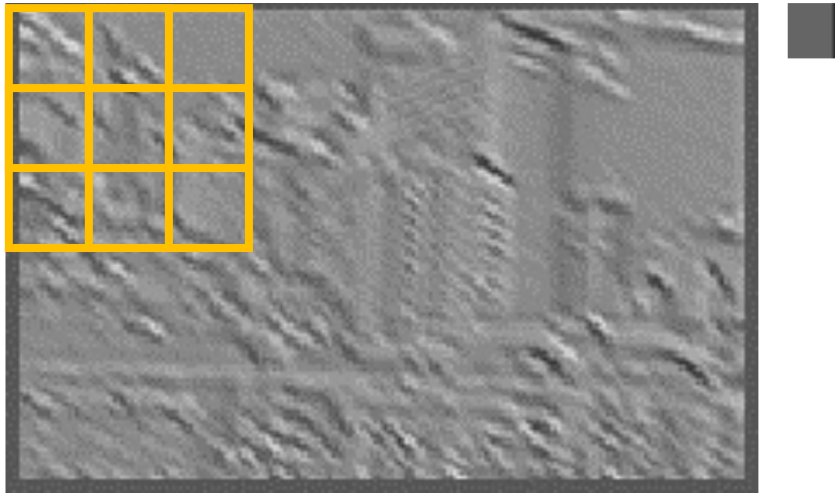


http://cs231n.github.io/convolutional-networks/

# Pooling Layers

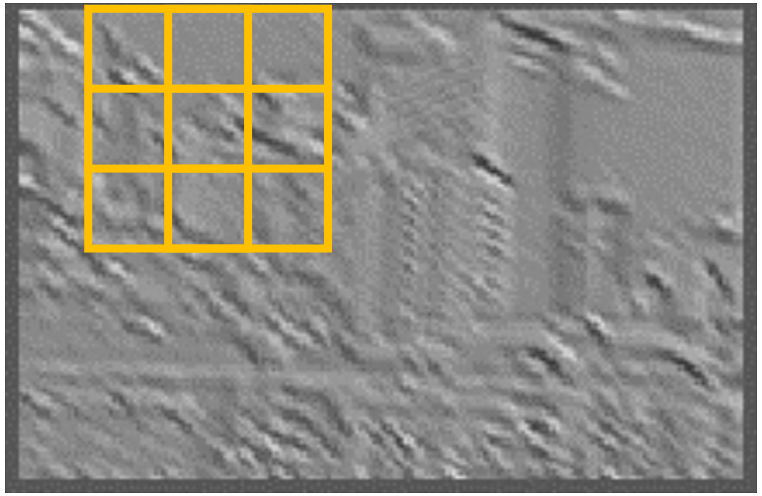


# Pooling Layers



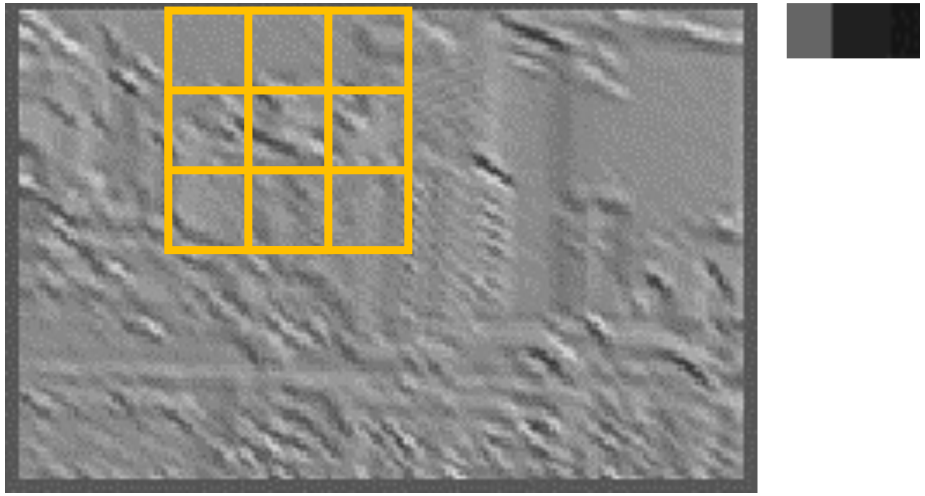
$$\text{output}[0,0] = \max_{0 \le \tau < k} \max_{0 \le \gamma < k} \text{image}[0+\tau, 0+\gamma]$$

# Pooling Layers



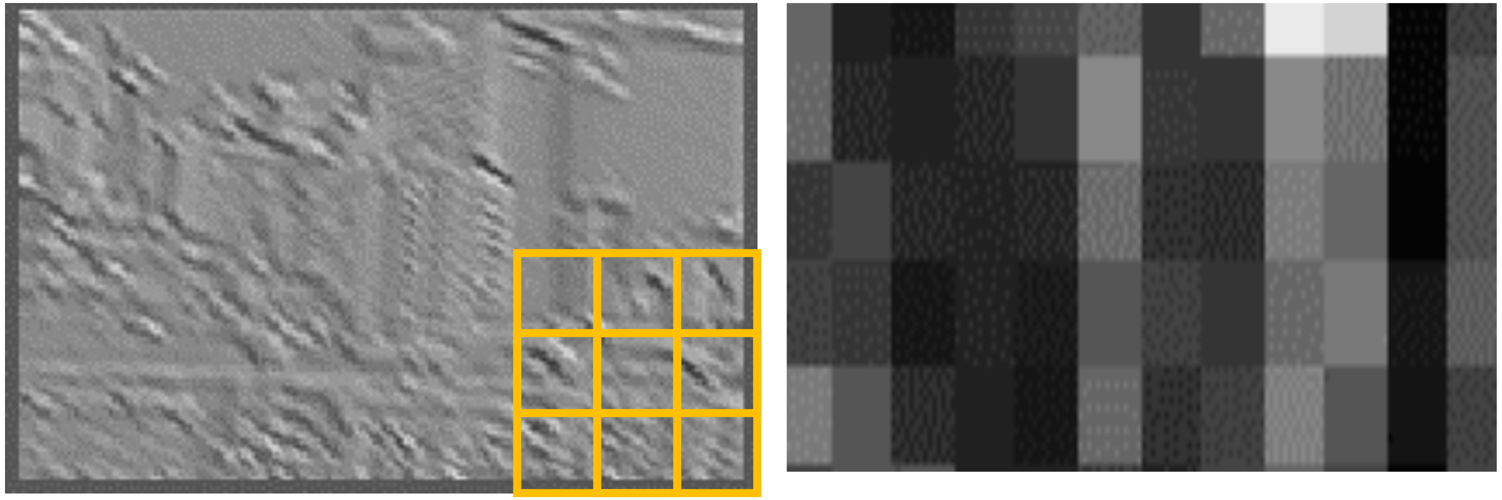
$$\text{output}[0,1] = \max_{0 \le \tau < k} \max_{0 \le \gamma < k} \text{image}[0+\tau, 1+\gamma]$$

# Pooling Layers



$$\text{output}[0,2] = \max_{0 \le \tau < k} \max_{0 \le \gamma < k} \text{image}[0 + \tau, 2 + \gamma]$$

# Pooling Layers



$$\text{output}[i,j] = \max_{0\le\tau<k} \max_{0\le\gamma<k} \text{image}[i+\tau, j+\gamma]$$

# Pooling Layers

- **Summary:** Hyperparameters
  - Kernel size
  - Stride (usually >1)
  - Amount of zero-padding
  - Pooling function (almost always “max”)

- Together, these determine the relationship between the input tensor shape and the output tensor shape

- **Note:** Unlike convolution, pooling operates on channels separately
  - Thus, $n$ input channels → $n$ output channels

# Summary: Convolution vs. Pooling

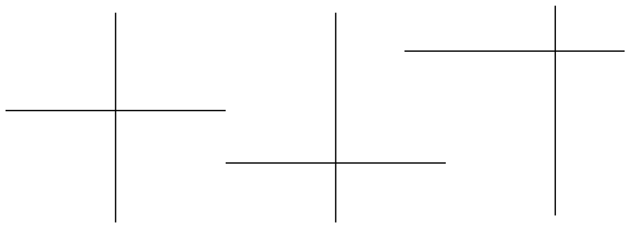
- **Convolution layers:** Translation equivariant
  - If object is translated, convolution output is translated by same amount
  - Produce “image-shaped” features that retain associations with input pixels
- **Pooling layers:** Translation invariant
  - Binning to make outputs insensitive to translation
  - Also reduces dimensionality
- Combined in modern architectures
  - Convolution to construct equivariant features
  - Pooling to enable invariance

# Example

- Suppose we want to predict whether an image depicts Cartesian axes

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

input image



?

target (binary) label

# Example

- **Step 1:** Convolve the image with two filters
  - No padding, stride 1

- **Step 2:** Run max pooling

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

convolution filters

# Example

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

# Example

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

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

# Example

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

# Example

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

# Example

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

# Example

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

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

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

Input image | Convolution | Features

# Example

$$
\begin{tikzcd}
 & \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \arrow[r] & \begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix} \arrow[dr] & \\
\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \arrow[ur] \arrow[dr] & & & \begin{bmatrix} 2 \\ 4 \end{bmatrix} \\
 & \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 1 & 1 & 1 \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \arrow[r] & \begin{bmatrix} -1 & -\frac{1}{2} \\ \frac{7}{2} & 4 \end{bmatrix} \arrow[ur] &
\end{tikzcd}
$$

Input image Convolution Features Pooling

# Example

$$
\begin{tikzcd}
 & \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \arrow[r] & \begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix} \arrow[dr] & & \text{Clear vertical line} \arrow[dl] \\
\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \arrow[ur] \arrow[dr] & & & \begin{bmatrix} 2 \\ 4 \end{bmatrix} & \\
 & \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 1 & 1 & 1 \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \arrow[r] & \begin{bmatrix} -1 & -\frac{1}{2} \\ \frac{7}{2} & 4 \end{bmatrix} \arrow[ur] & & \text{Clear horizontal line} \arrow[ul]
\end{tikzcd}
$$

Input image | Convolution | Features | Pooling

# Agenda

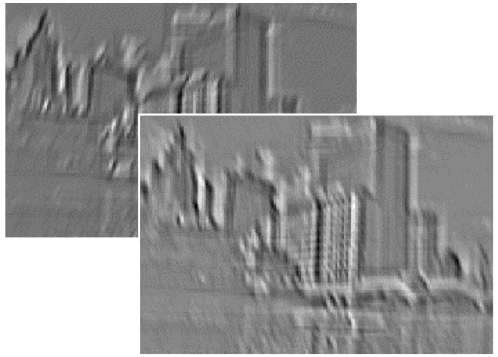
- **Convolutional & pooling layers**
- **Convolutional neural networks**
- **Feature visualization**
- **Applications**

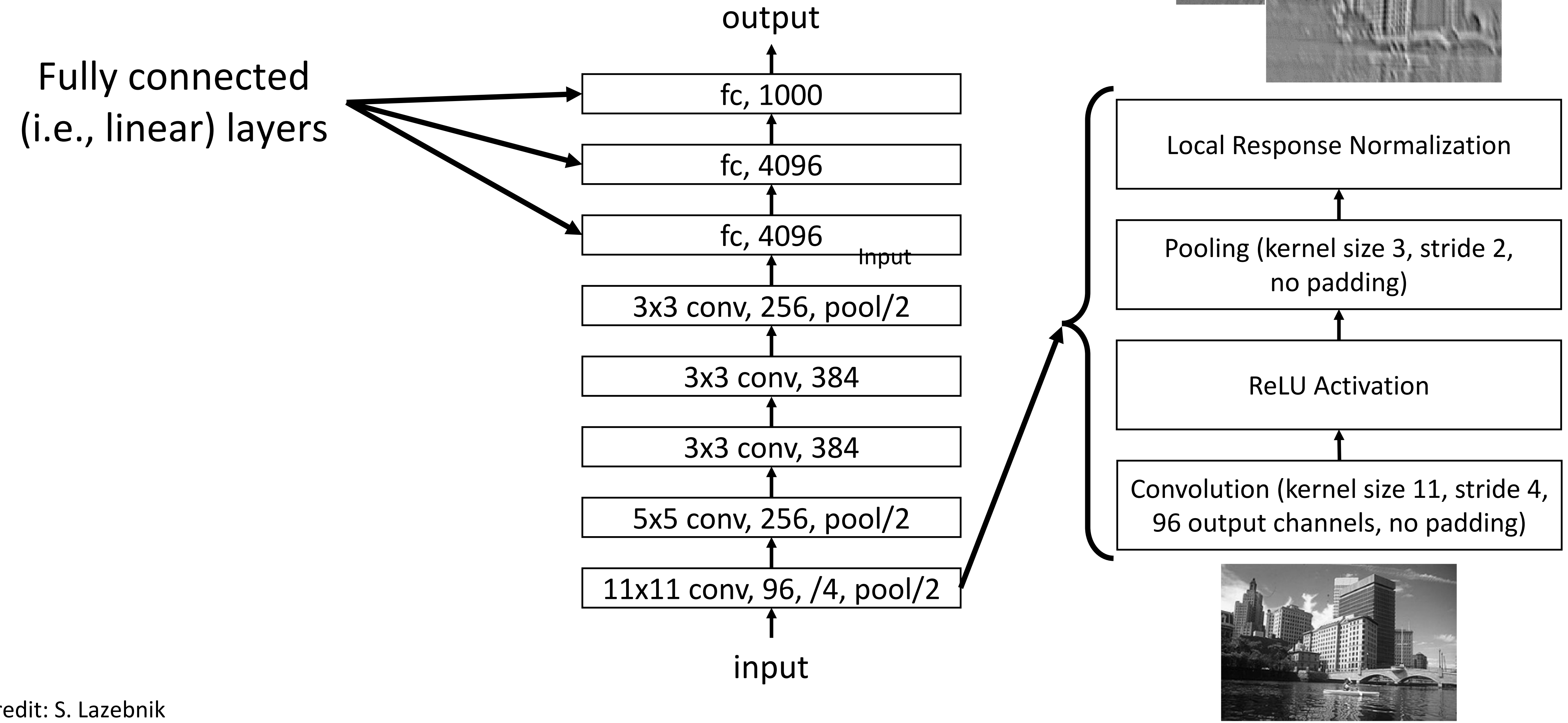
# Example Architecture: AlexNet

- **ImageNet dataset**
  - 1000 class image classification problem (e.g., grey fox, tabby cat, barber chair)
  - >1M image-label pairs gathered from internet and crowdsourced labels

- **AlexNet Architecture (Krizhevsky 2012)**
  - Historically important architecture
  - Image classification network (~60M parameters)
  - Trained using GPUs on ImageNet dataset
  - Huge improvement in performance compared to prior state-of-the-art

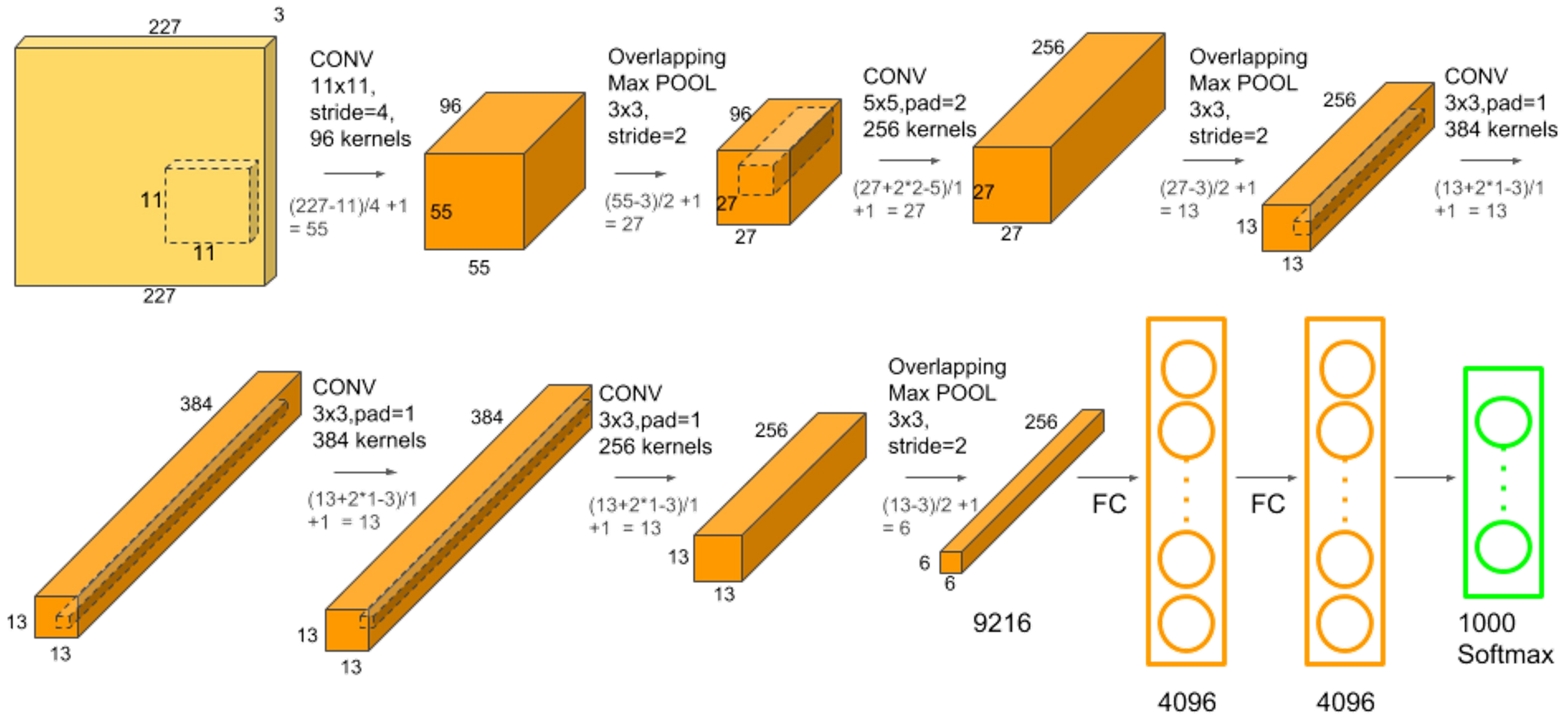
# Example Architecture: AlexNet





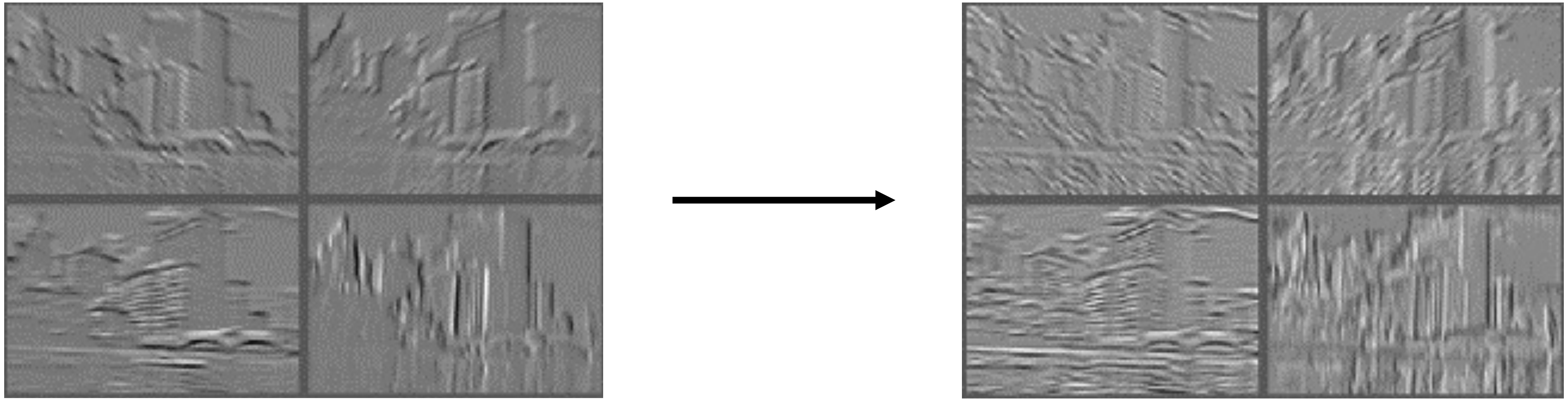

slide credit: S. Lazebnik

# Example Architecture: AlexNet



# Aside: Local Response Normalization

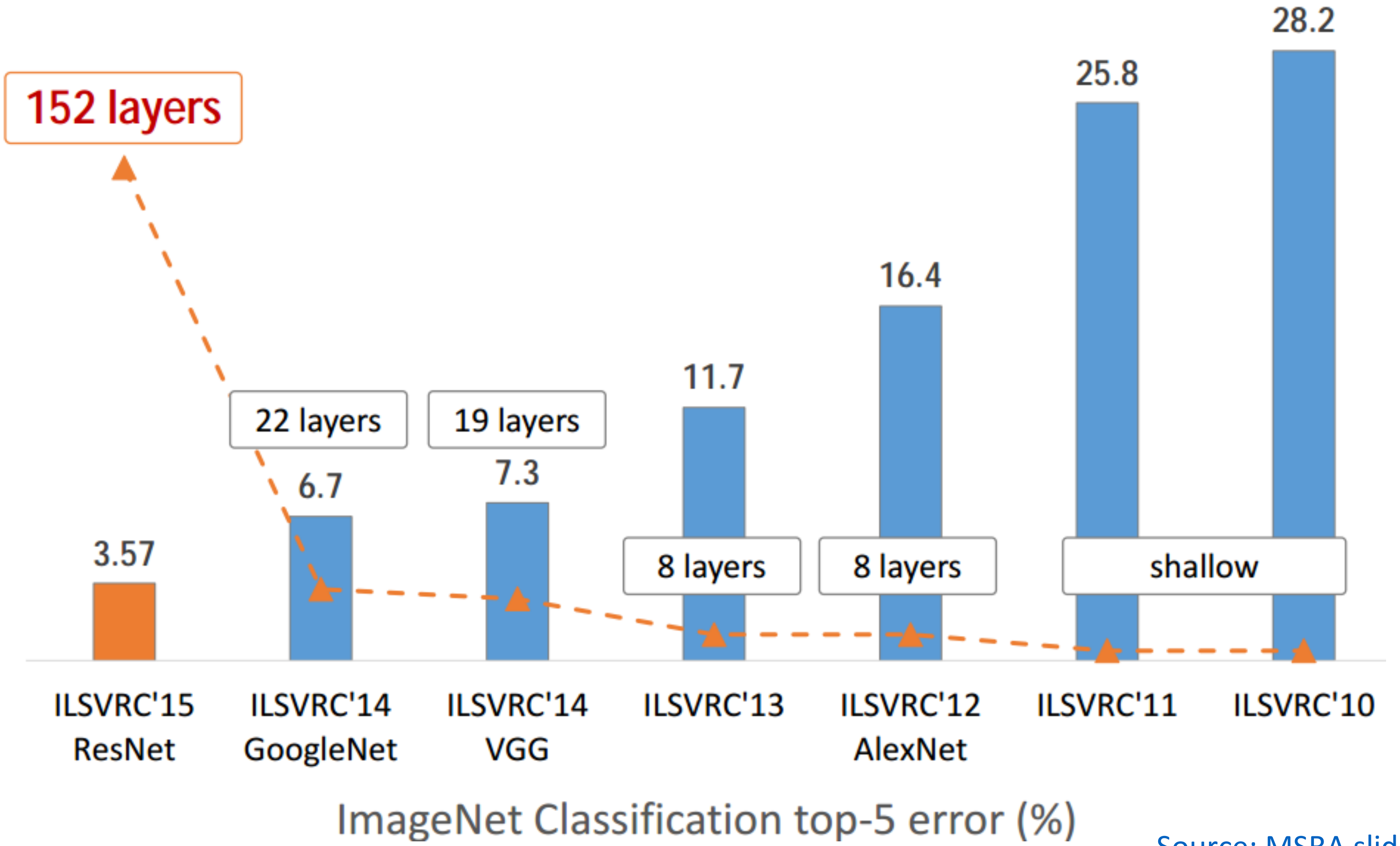
- Highlights areas where the feature maps change
- Historically a standard layer, but no longer used
- Also called “contrastive normalization”



# Convolutional Neural Networks

- "Convolutional layer" often refers to sequence of layers
- **Modern sequence of layers**
  - Convolution → Batch Normalization → Pooling → ReLU
  - Convolution → Batch Normalization → ReLU → Pooling
- Can also omit pooling (especially for very deep neural networks)

# Evolution of Neural Networks




Source: MSRA slides at ILSVRC15

# Evolution of Neural Networks

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

Source: MSRA slides at ILSVRC15

# Residual Connections

- **Challenges with deeper networks**
  - Overfitting?
  - No, 56 layer network underfits!

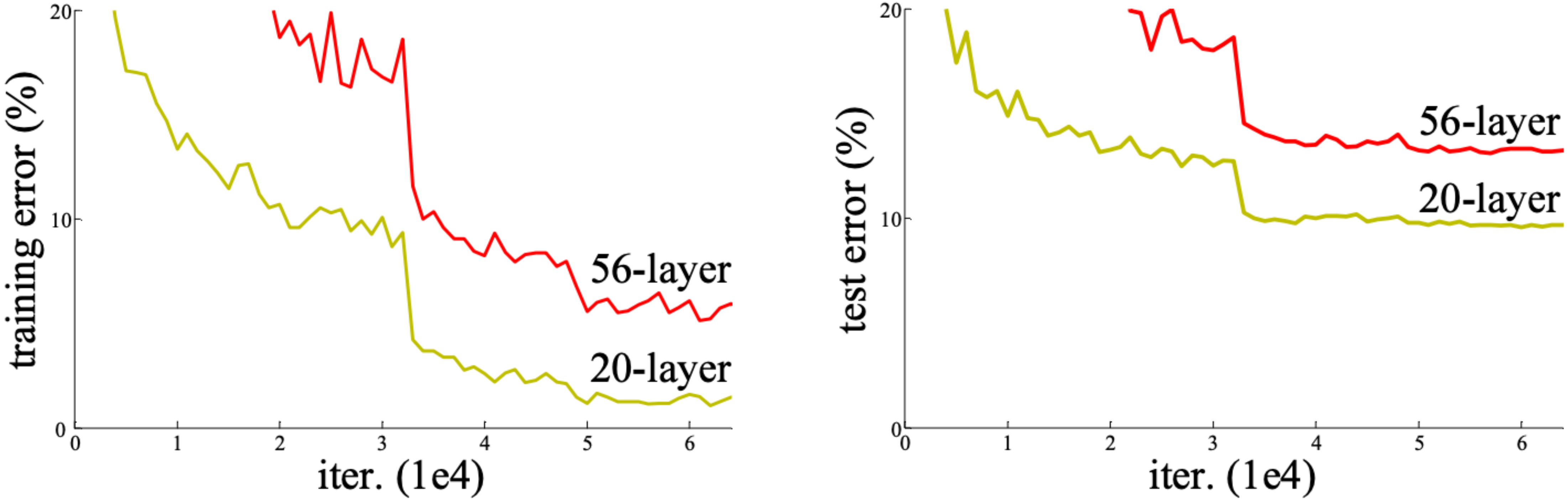



Image credit: He et al, Residual Nets, 2015

# Residual Connections

- **Challenges with deep networks**
  - Overfitting?
  - No, 56 layer network underfits!
- **Optimization/representation**
  - Difficulty representing the identity function!
- **Solution:** "Skip" connections
  - Facilitate direct feedback from loss
  - Easy to represent identity function

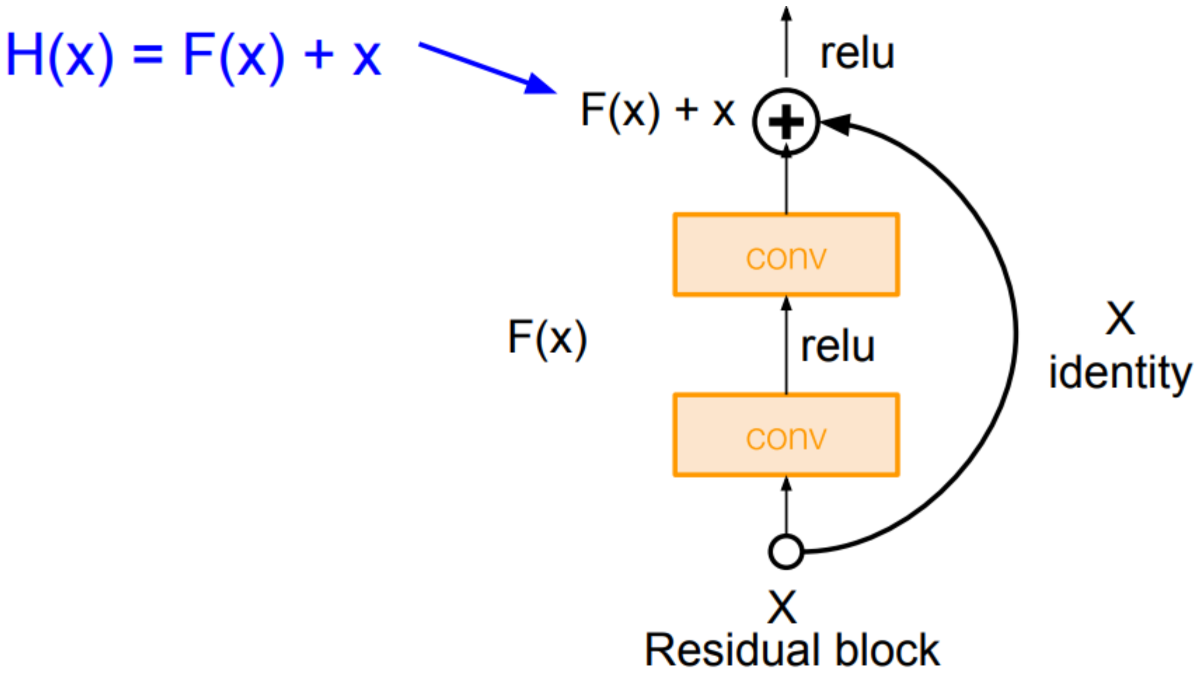



Image credit: Fei-Fei Li, Justin Johnson, Serena Yeung

# Residual Connections

- **Residual layers:** Given any convolutional layer $F(x)$, use

$$H(x) = F(x) + x$$

- Two views of residual connections:
  - **View 1:** Providing shortcuts to gradients on the backward pass
  - **View 2:** Allow each “residual block” to fit the residual error (boosting!)

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

# Residual Networks

- Stack lots of residual blocks!
  - Kernel size 3, no padding, stride 1, no pooling
  - Reduce feature dimensions by using stride 2 once every $K$ blocks
  - Maintains feature size to build very deep nets

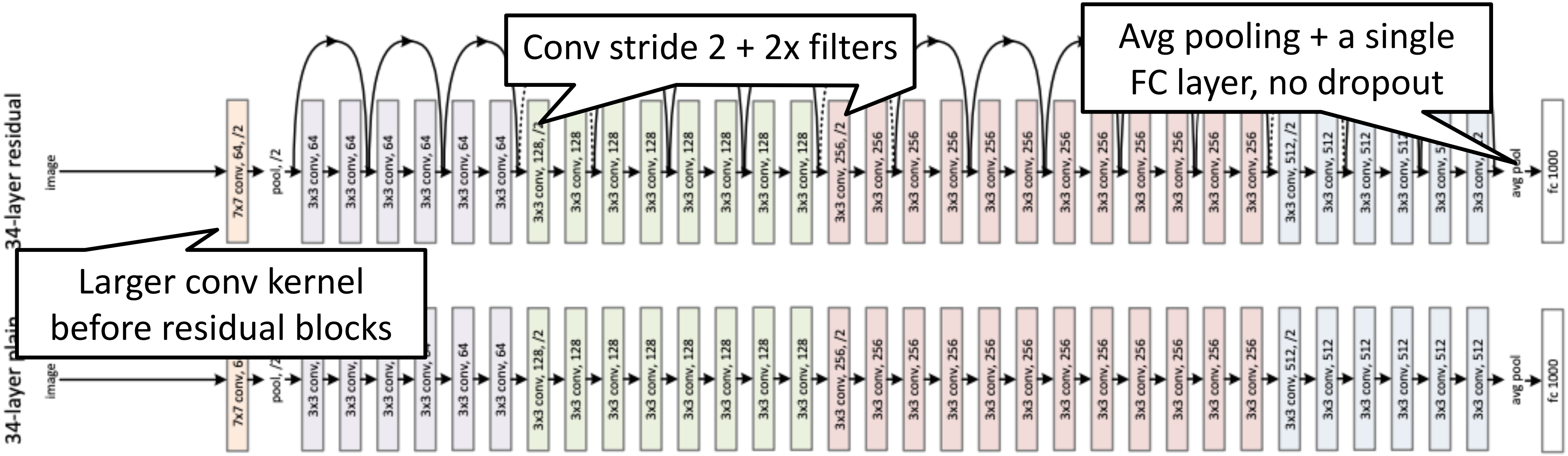



Image credit: He et al, Residual Nets, 2015

# Residual Networks

- For deeper networks, improve efficiency through 1x1 convolutions
- Many other improvements since 2015!
  - E.g., “ResNeXt”, “Identity Mappings”, “ConvNeXt” etc.

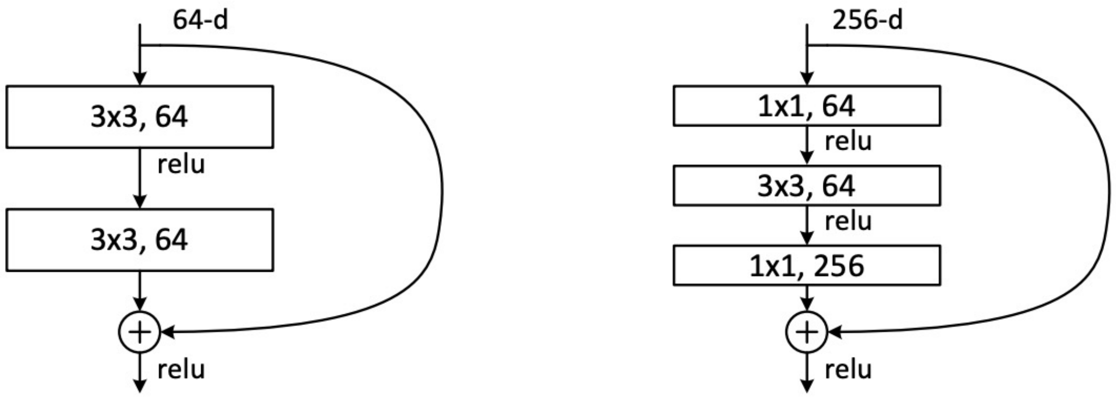


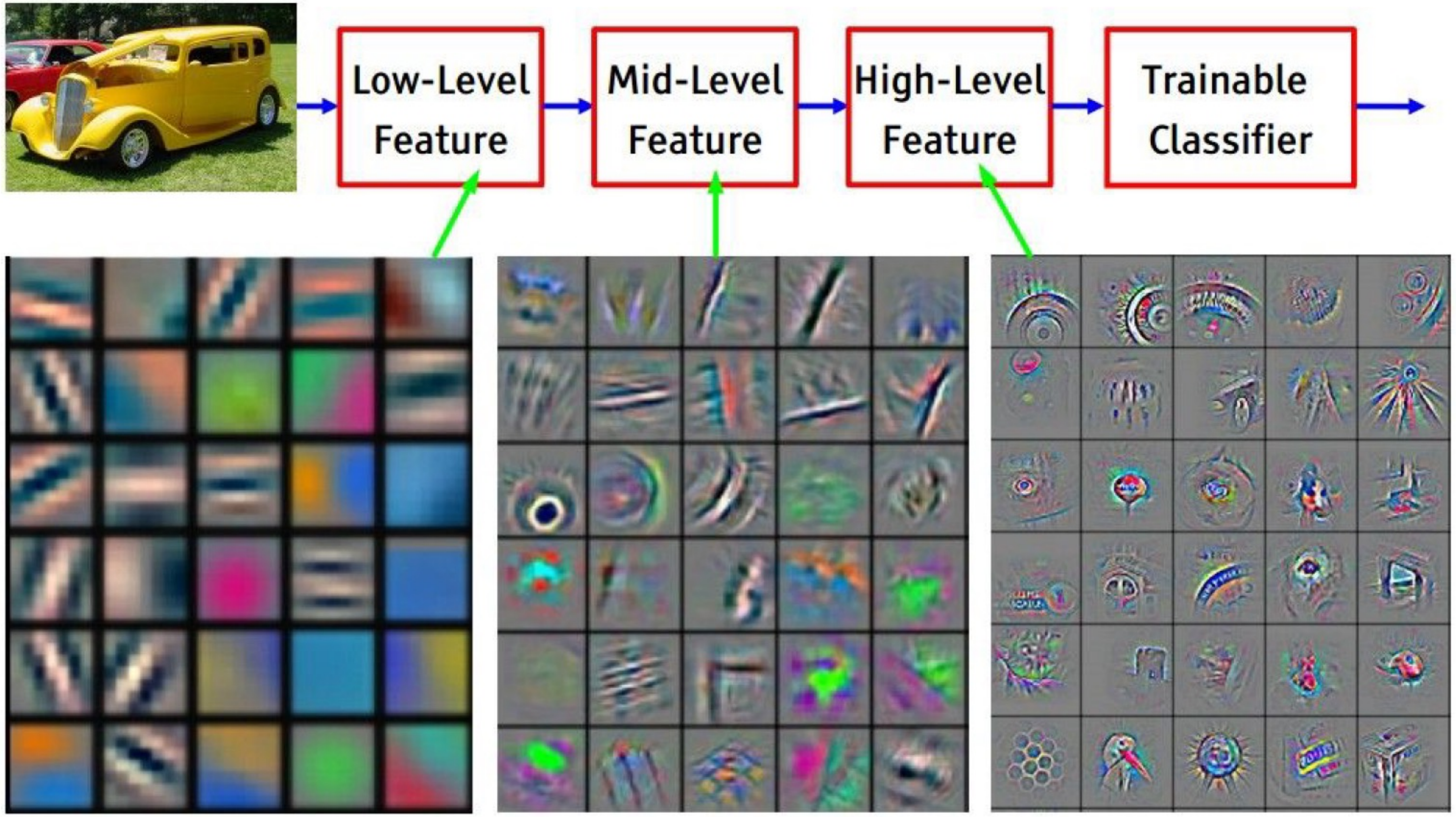

Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

Image credit: He et al, Residual Nets, 2015

# Agenda

- **Convolutional & pooling layers**
- **Convolutional neural networks**
- **Feature visualization**
- **Applications**

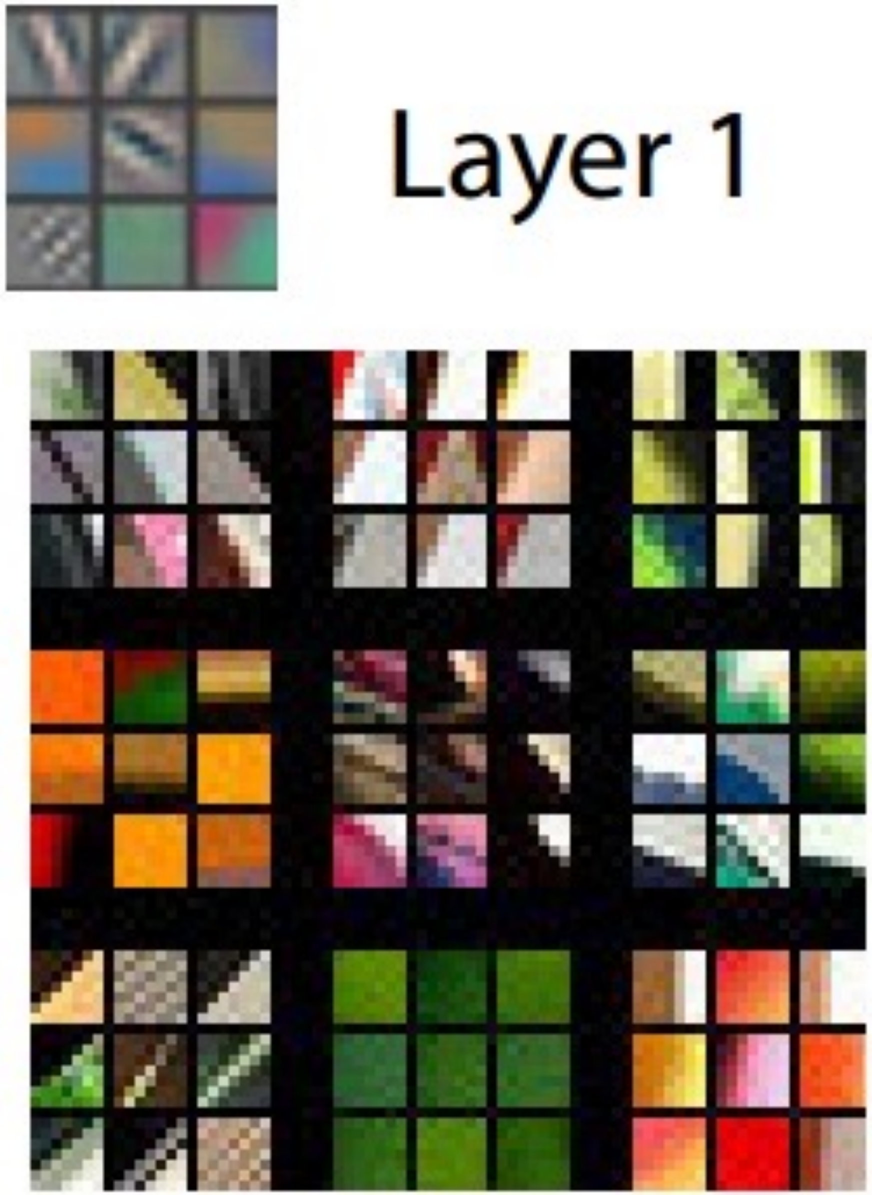
# Feature Visualization




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

Slide credit: Yann LeCun

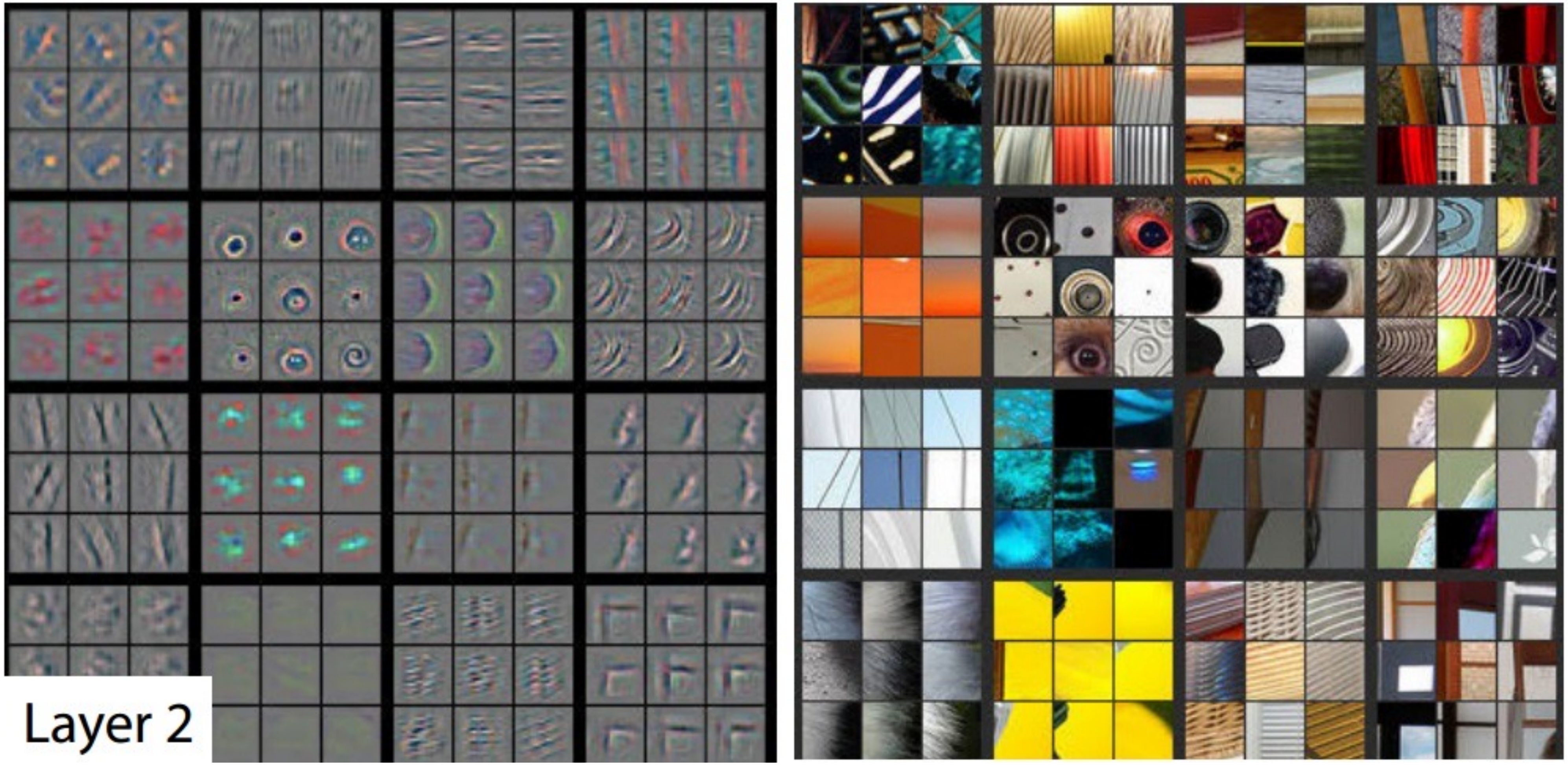
# Layer 1




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

Slide credit: Jia-Bin Huang

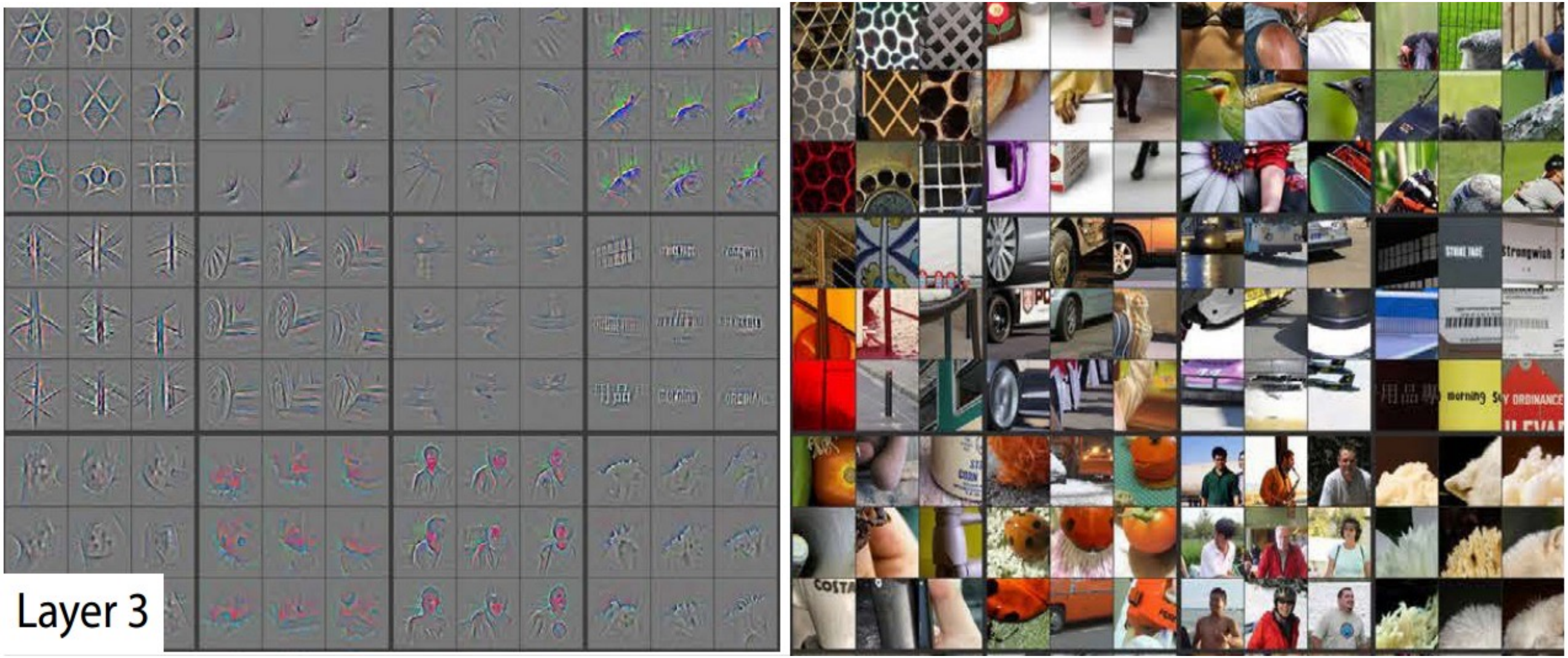
# Layer 2




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

Slide credit: Jia-Bin Huang

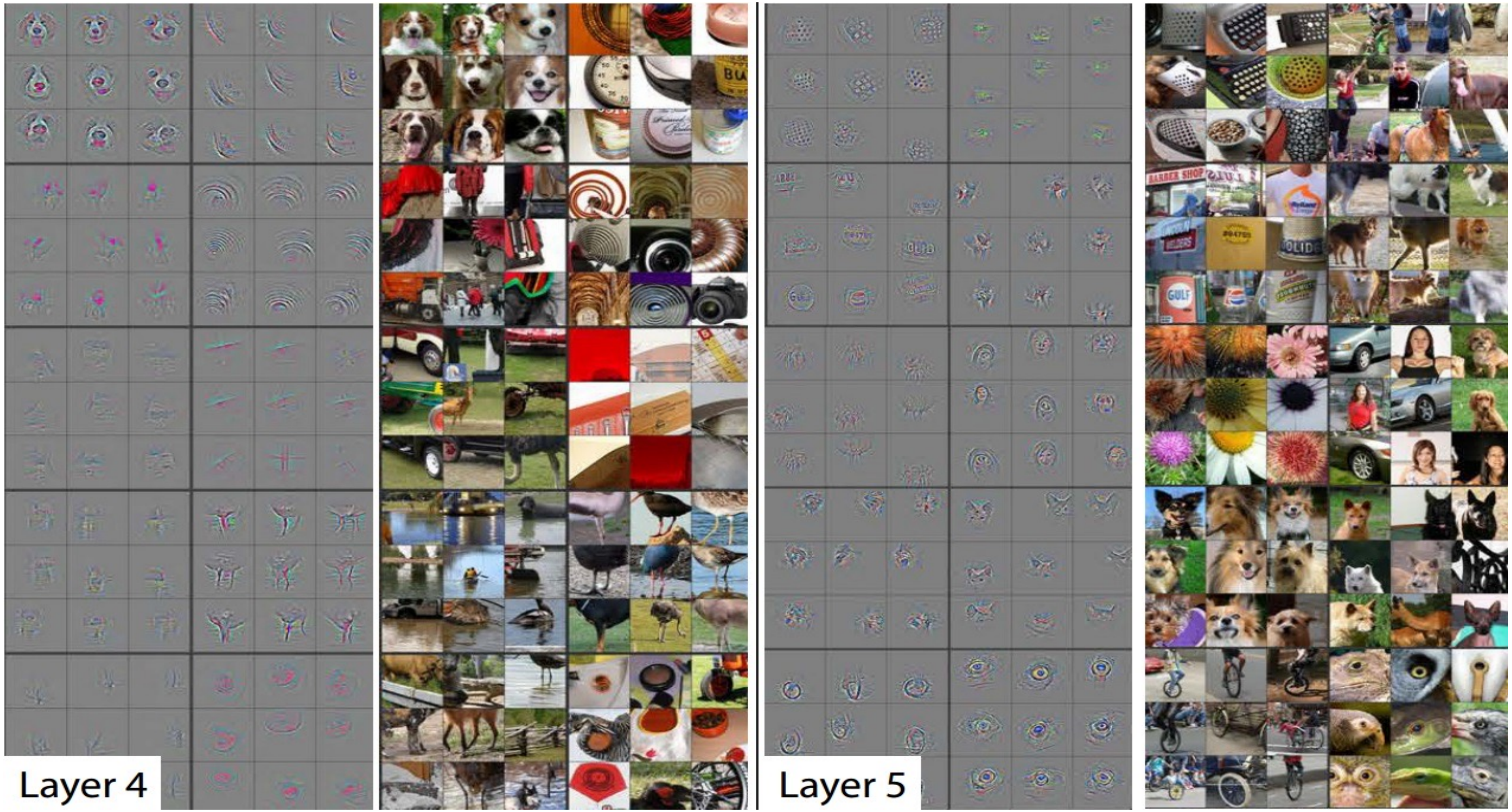
# Layer 3




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

Slide credit: Jia-Bin Huang

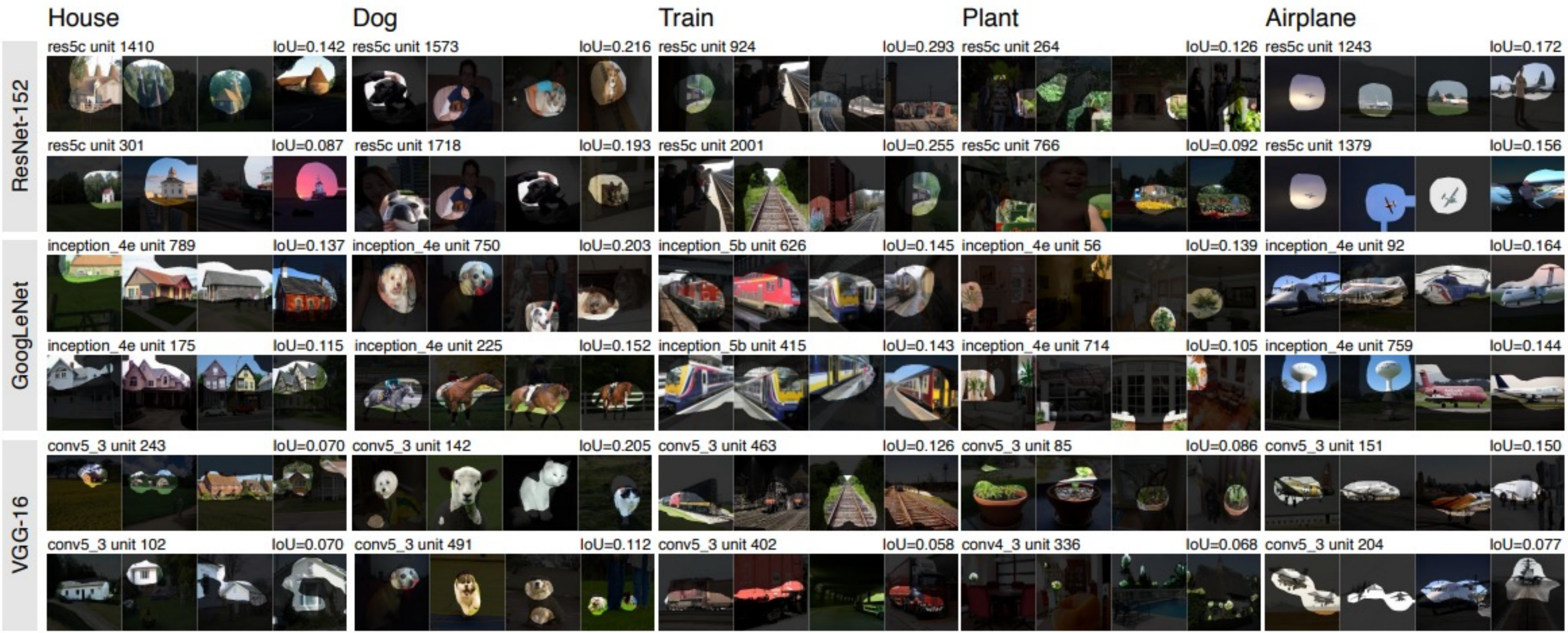
# Layer 3




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

Slide credit: Jia-Bin Huang

# Neural Network Dissection




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

# What About Small Datasets?

- **Transfer learning:** We can reuse trained concepts!
  - Since CNNs trained on ImageNet appear to learn general features
  - We can reuse these models in some way to perform new tasks

- **Strategy 1:** Feature extraction
  - Remove final (softmax) layer and replace with a new one
  - Train only the new layer

- **Strategy 2:** Finetuning
  - Do the same thing but train end-to-end

# What About Small Datasets?

- **New dataset is similar to the original dataset**
  - Can use very small datasets
  - Both strategies work

- **New dataset is different from original dataset**
  - Transfer learning still works!
  - Moderate-sized datasets
  - Finetune end-to-end
  - **Examples:** Medical images, audio spectrograms, etc.

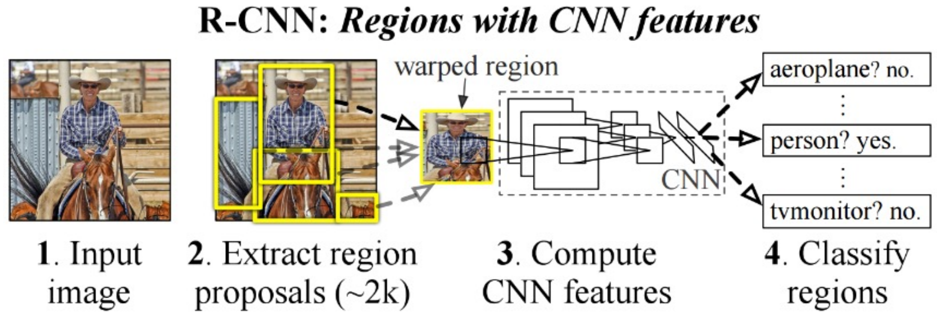
# Agenda

- **Convolutional & pooling layers**
- **Convolutional neural networks**
- **Feature visualization**
- **Applications**

# Applications

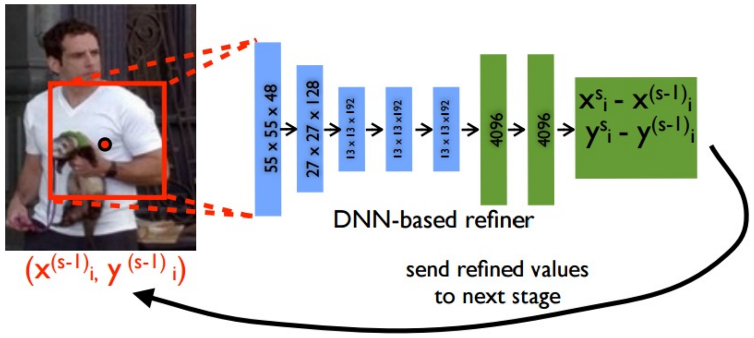
**Object detection**

[Girshick et al. CVPR14]



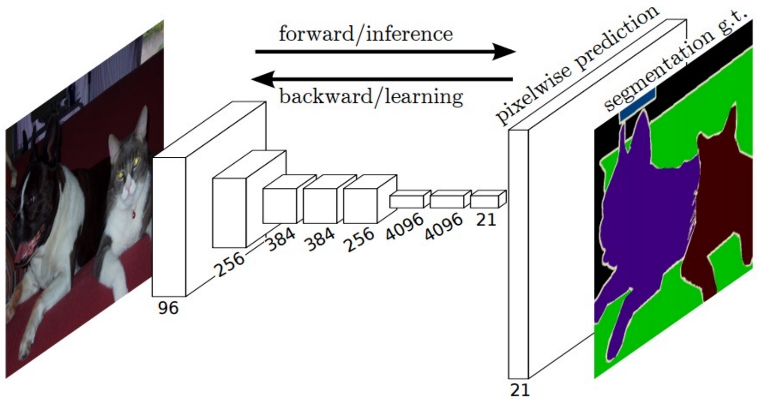

**Pose detection (regression)**

[Toshev et al. CVPR14]




**Semantic segmentation**

[Long et al. CVPR15]

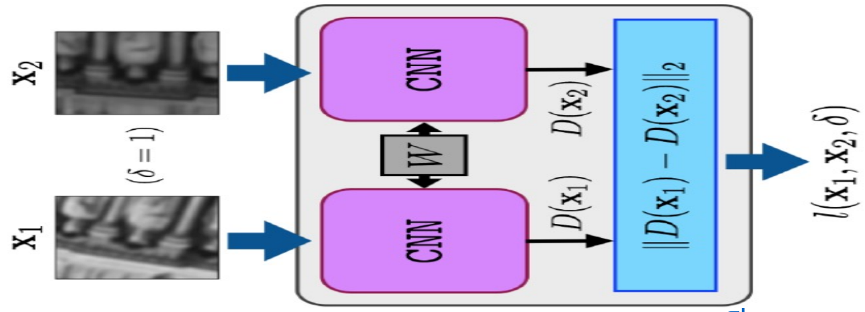



Examples courtesy Jia-Bin Huang

# Applications

**Similarity metric learning**

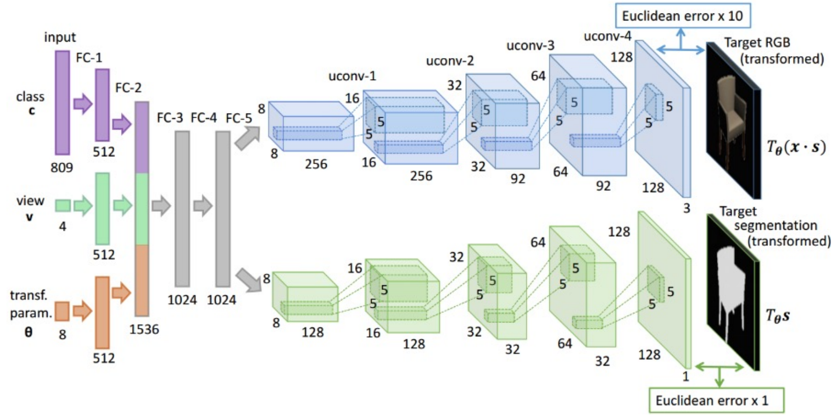
[Chopra et al. CVPR05]




Picture source [Girsh]

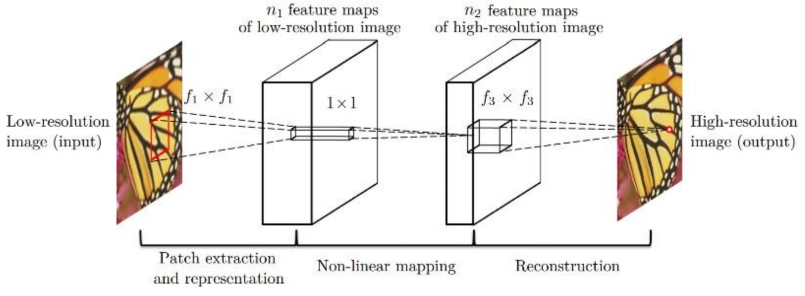
**Image generation**

[Dosovitskiy et al. CVPR15]




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

[Dong et al. ECCV 2014]



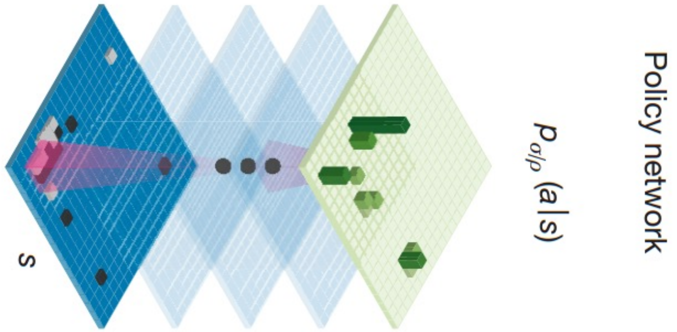

Examples courtesy Jia-Bin Huang

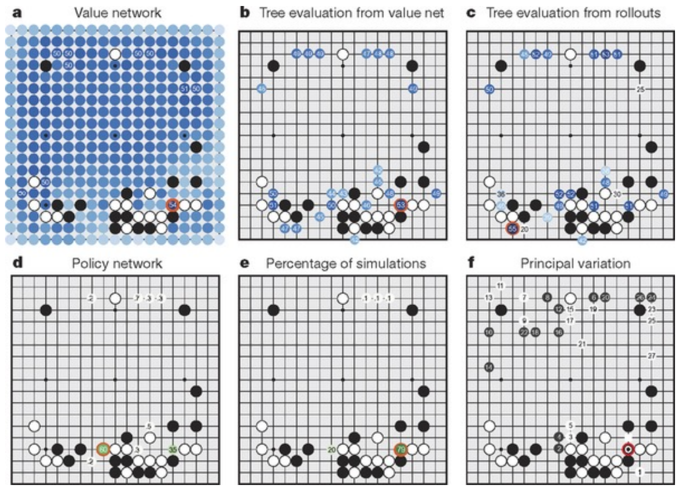
# Applications: Game Playing

CNN + Reinforcement learning

[Mnih et al, Nature' 15]


[Silver et al, Nature '16]



# Applications: Art Generation

See if you can tell artist 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