

CS-UY 4563: Lecture 19

Convolutional Neural Networks

NYU Tandon School of Engineering, Prof. Christopher Musco

COURSE LOGISTICS

- I will be reviewing project proposals over the next few days. If you need feedback sooner, please email separately to set up a meeting, or stop by office hours.
- Written Homework 4 due **next Monday**.
- Work through `demo_convolutions.ipynb`, now on course website.

CONVOLUTIONAL FEATURE EXTRACTION

Last lecture: **Convolution** in 1, 2, and 3D is a powerful generic tool for designing natural **feature transformations** for time-series data, audio data, or images.

$$\begin{array}{l} \mathbf{x} \\ \boxed{1 \quad 4 \quad 3 \quad -1 \quad 2 \quad -4 \quad 1 \quad 0 \quad 2 \quad -1} \\ \mathbf{w} \quad \boxed{1 \quad 2 \quad 1} \end{array} \longrightarrow$$

$$\begin{array}{l} \mathbf{x} \\ \boxed{1 \quad 4 \quad 3 \quad -1 \quad 2 \quad -4 \quad 1 \quad 0 \quad 2 \quad -1} \\ \mathbf{w} \quad \boxed{1 \quad 2 \quad 1} \end{array} \longrightarrow$$

$$\begin{array}{l} \mathbf{x} \\ \boxed{1 \quad 4 \quad 3 \quad -1 \quad 2 \quad -4 \quad 1 \quad 0 \quad 2 \quad -1} \\ \mathbf{w} \quad \boxed{1 \quad 2 \quad 1} \end{array} \longrightarrow$$

$$\boxed{12 \quad 9 \quad 3 \quad -1 \quad -5 \quad -1 \quad 3 \quad 3}$$

$$\mathbf{u} = \mathbf{x} * \mathbf{w}$$

2D CONVOLUTION

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3 ₀	2 ₁	1 ₂	0
0 ₂	1 ₂	3 ₀	1	
3 ₁	2 ₀	2	3	
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2 ₀	1 ₁	0 ₂
0 ₀	1 ₂	3 ₂	1 ₀	
3 ₁	2 ₀	2 ₁	3 ₂	
2 ₀	0 ₁	0 ₂	2 ₂	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	0 ₁	1 ₂	3	1
3 ₂	1 ₂	2 ₀	2	3
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	1 ₁	3 ₂	1	
3 ₁	2 ₂	2 ₀	3	
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	1 ₀	3 ₁	1 ₂	
3 ₁	2 ₂	2 ₂	3 ₀	
2 ₀	0 ₀	2 ₁	2 ₂	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	0 ₁	1 ₃	1	
3 ₂	1 ₁	2 ₂	2	3
2 ₂	0 ₂	0 ₀	2	2
2 ₀	0 ₁	0 ₂	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	1 ₁	2 ₂	3	1
3 ₁	0 ₂	0 ₀	2 ₂	3
2 ₀	0 ₁	0 ₂	2 ₀	2
2	0 ₀	0 ₁	0 ₂	1

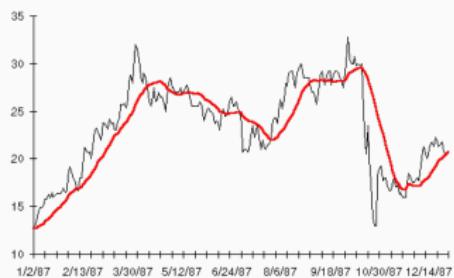
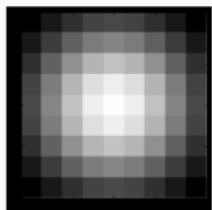
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	1 ₀	3 ₁	1 ₁	
3 ₁	2 ₀	2 ₁	3 ₂	
2 ₀	0 ₂	2 ₂	2 ₀	2
2	0 ₀	0 ₁	0 ₂	1 ₂

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

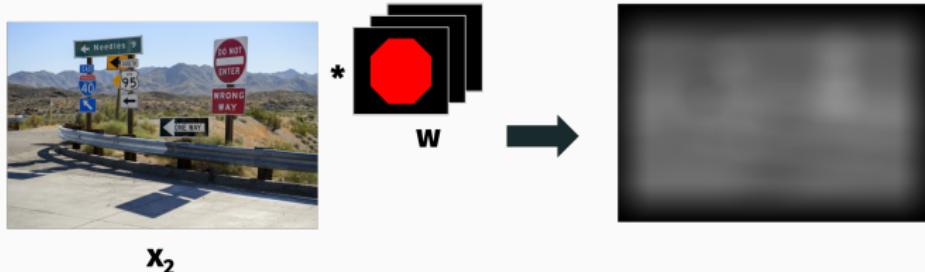
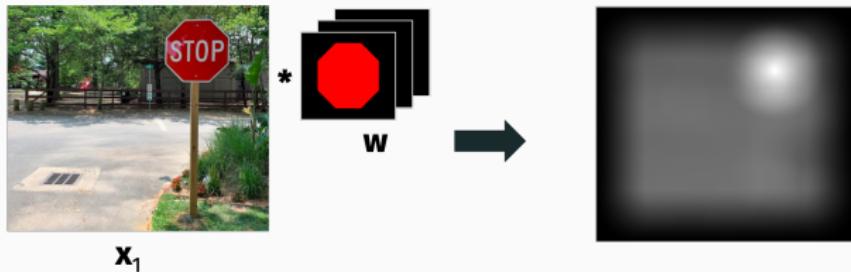
APPLICATION 1: SMOOTHING

A uniform or Gaussian filter can be used to smooth input data:



APPLICATION 2: PATTERN MATCHING

Convolution can be used to find local patterns in images:



SIDE NOTE

Good question from Piazza: Won't this filter yield lots of false positives?



SIDE NOTE

Possible fix that doesn't require image normalization:



APPLICATIONS OF CONVOLUTION

Application 3: Edge detection.

Consider a 2D edge detection filter:

$$W_1 = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$



$x * ?$

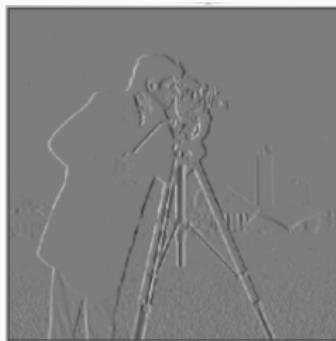


$x * ?$

APPLICATIONS OF CONVOLUTION

Sobel filter is more commonly used:

$$W_1 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad W_2 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



$x * ?$



$x * ?$

DIRECTIONAL EDGE DETECTION

Can define edge detection filters for any orientation.

$$\begin{matrix} \text{X} \\ \text{---} \\ \text{---} \end{matrix} * \begin{matrix} \text{W}_1 & \text{W}_2 & \text{W}_3 & \dots \\ \text{---} \\ \text{---} \end{matrix} = \begin{matrix} \text{U}_1 & \text{U}_2 & \text{U}_3 & \dots \\ \text{---} \\ \text{---} \end{matrix}$$

The diagram illustrates the process of applying directional edge detection filters to an input image. On the left, an input image X is shown as a white circle on a black background. This is followed by a convolution operator (*) and a filter bank consisting of four filters labeled W_1 , W_2 , W_3 , and \dots . Each filter W_i has a specific orientation, represented by diagonal lines within the filter kernel. The result of this convolution is a feature map consisting of four channels, each showing a response to a different orientation. These channels are labeled U_1 , U_2 , U_3 , and \dots . The channels U_1 and U_2 show strong responses along the horizontal and vertical axes respectively, while U_3 shows a diagonal response, indicating the presence of edges at a 45-degree angle.

EDGE DETECTION

How would edge detection as a feature extractor help you classify images of city-scapes vs. images of landscapes?



EDGE DETECTION



I_C

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



E_C



I_L

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



E_L

$$\text{mean}(I_C) = .108 \quad \text{vs.} \quad \text{mean}(I_L) = .123$$

The image with highest vertical edge response isn't the city-scape.

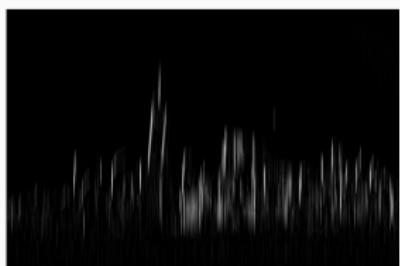
EDGE DETECTION + PATTERN MATCHING

Feed edge detection result into pattern matcher that looks for long vertical lines.

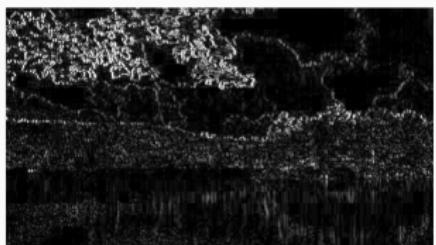


E_C

$$\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} *$$



V_C



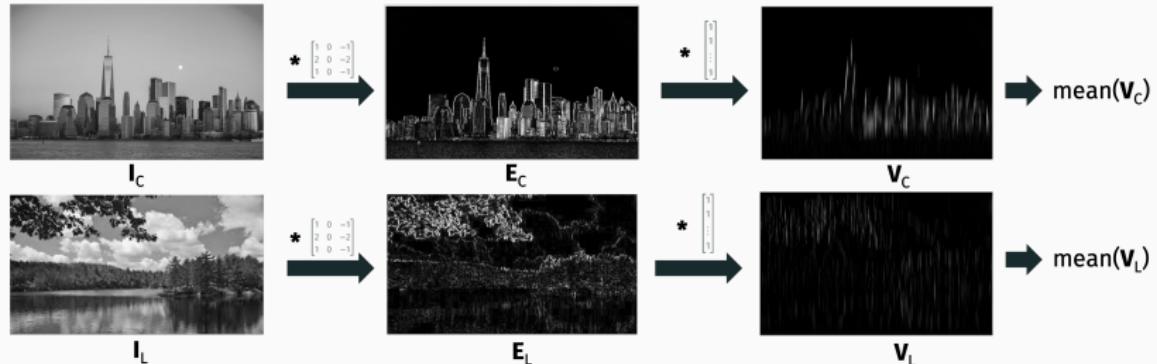
E_L

$$\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} *$$



V_L

HIERARCHICAL CONVOLUTIONAL FEATURES



$$\text{mean}(V_C) = .062 \quad \text{vs. } \text{mean}(V_L) = .054$$

$$\text{mean}(V_C \cdot V_C) = .042 \quad \text{vs. } \text{mean}(V_L \cdot V_L) = .018$$

The image with highest average response to (edge detector) + (vertical pattern) is the city scape.

$\text{mean}(V)$ is an extracted scalar feature which could be used for classifying cityscapes from landscapes using a linear classifier.

HIERARCHICAL CONVOLUTIONAL FEATURES

Hierarchical combinations of simple convolution filters are
very powerful for understanding images.

In particular, edge detection seems like a critical first step.

Lots of evidence from biology.

VISUAL SYSTEM

Light comes into the eye through the lens and is detected by an array of photosensitive cells in the **retina**.

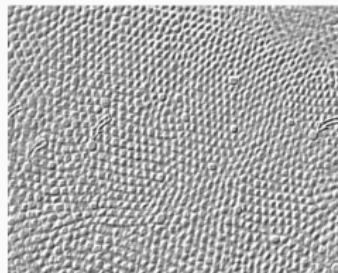
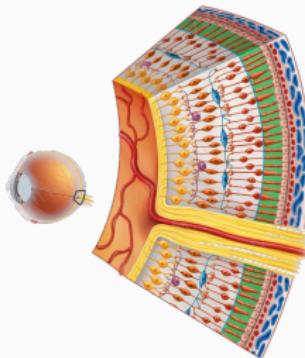
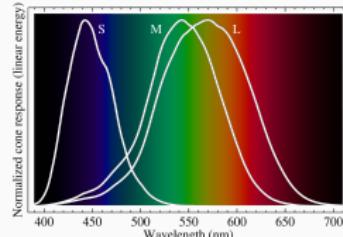


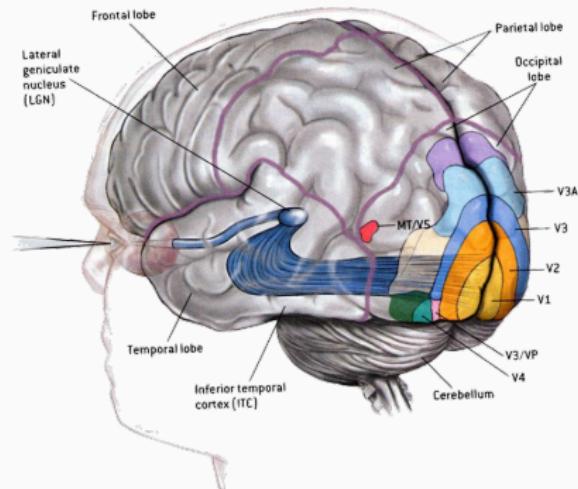
Fig. 13. Tangential section through the human fovea.
Larger cones (arrows) are blue cones. From Ahnelt et al. 1987.

Rod cells are sensitive to all light, larger **cone** cells are sensitive to specific colors. We have three types of cones:



VISUAL SYSTEM

Signal passes from the retina to the primary (V1) visual cortex, which has neurons that connect to higher level parts of the brain.

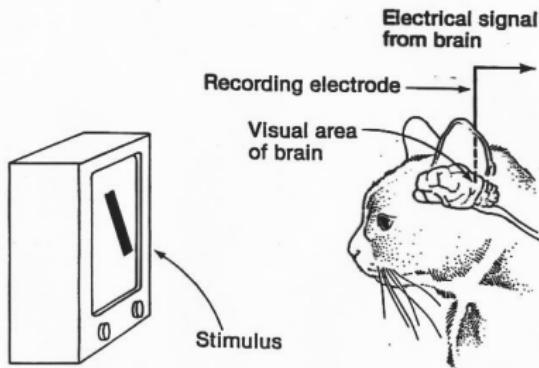


What sort of processing happens in the primary cortex?

Lots of edge detection!

EDGE DETECTORS IN CATS

Huber + Wiesel, 1959: "Receptive fields of single neurones in the cat's striate cortex." Won Nobel prize in 1981.



Different neurons fire when the cat is presented with stimuli at different angles. Cool video at

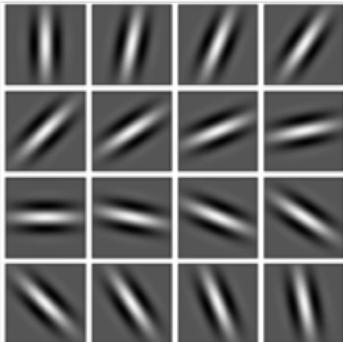
<https://www.youtube.com/watch?v=0GxVfKJqX5E>.

"What the Frog's Eye Tells the Frog's Brain", Lettvin et al. 1959. Found explicit edge detection circuits in a frogs visual cortex.

EXPLICIT FEATURE ENGINEERING

State of the art until ~ 10 years ago:

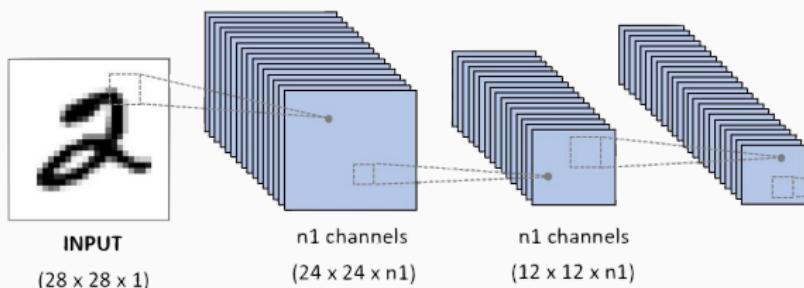
- Convolve image with edge detection filters at many different angles.
- Hand engineer features based on the responses.
- **SIFT** and **HOG** features were especially popular.



CONVOLUTIONAL NEURAL NETWORKS

Neural network approach: Learn the parameters of the convolution filters based on training data.

Convolutional Layer



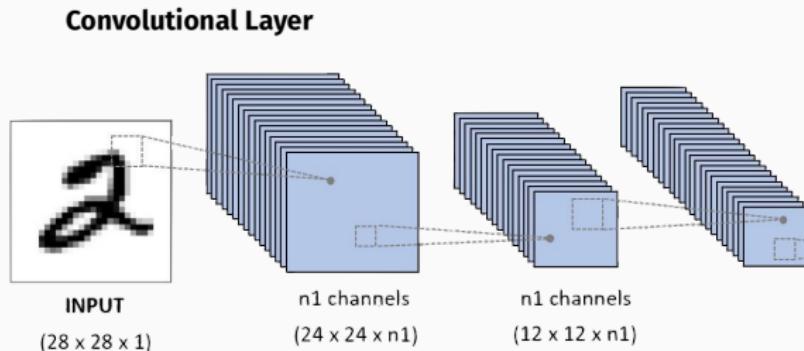
First convolutional layer involves n_1 convolution filters W_1, \dots, W_{n_1} . Each is small, e.g. 5×5 . Every entry in W_i is a free parameter:
 $\sim 25 \cdot n_1$ parameters to learn.

Produces n_1 matrices of hidden variables: i.e. a tensor with depth n_1 .

CONVOLUTIONAL NEURAL NETWORKS

A fully connected layer that extracts the same feature would require $(28 \cdot 28 \cdot 24 \cdot 24) \cdot n1 = 451,584 \cdot n1$ parameters.

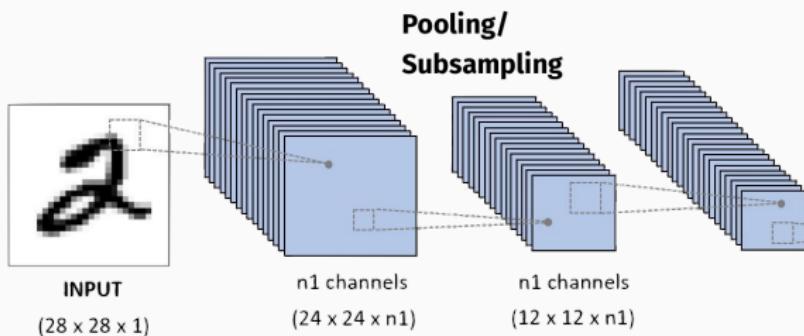
By “baking in” knowledge about what type of features matter, we greatly simplify the network.



Each of the $n1$ outputs is typically processed with a **non-linearity**. Most commonly a Rectified Linear Unity (ReLU): $x = \max(\bar{x}, 0)$.

POOLING AND DOWNSAMPLING

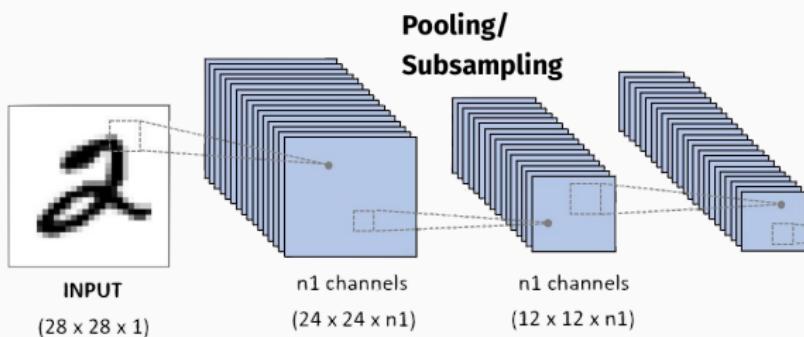
Convolution + non-linearity are typically followed by a layer which performs **pooling** + down-sampling.



Most common approach is **max-pooling**.

POOLING AND DOWNSAMPLING

Convolution + non-linearity are typically followed by a layer which performs **pooling** + down-sampling.



Most common approach is **max-pooling**.

POOLING AND DOWNSAMPLING

Max Pooling			
29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

Average Pooling			
31	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

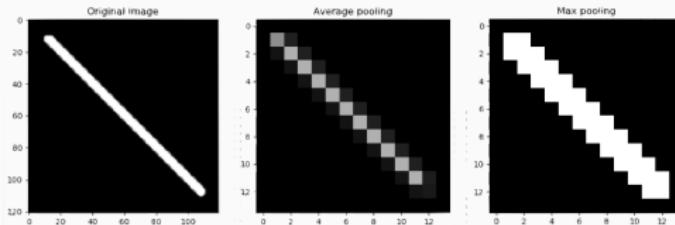
↓
2 x 2
pool size

100	184
12	45

↓
2 x 2
pool size

36	80
12	15

- Reduces number of variables, helps prevent over-fitting and speed up training.
- Helps “smooth” result of convolutional filters.
- Improves shift-invariance.

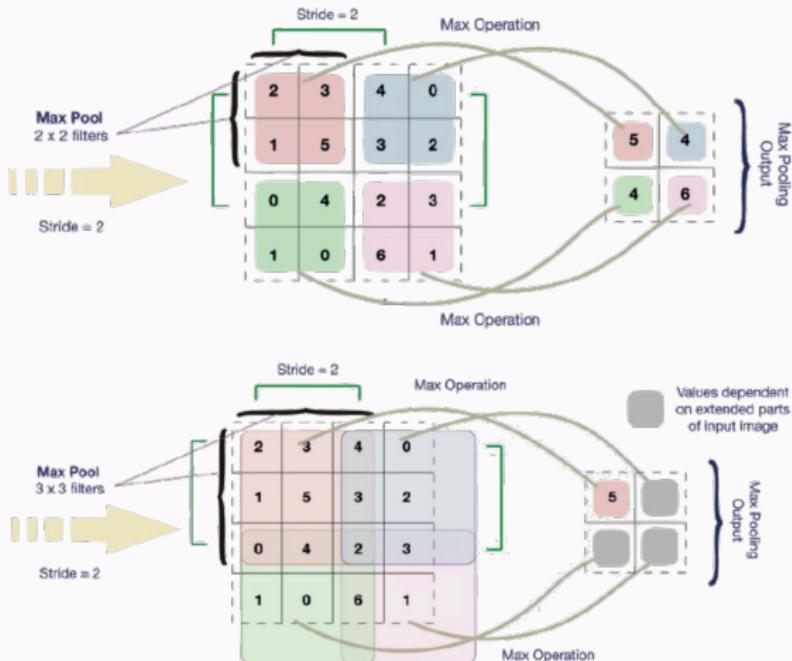


POOLING AND DOWNSAMPLING

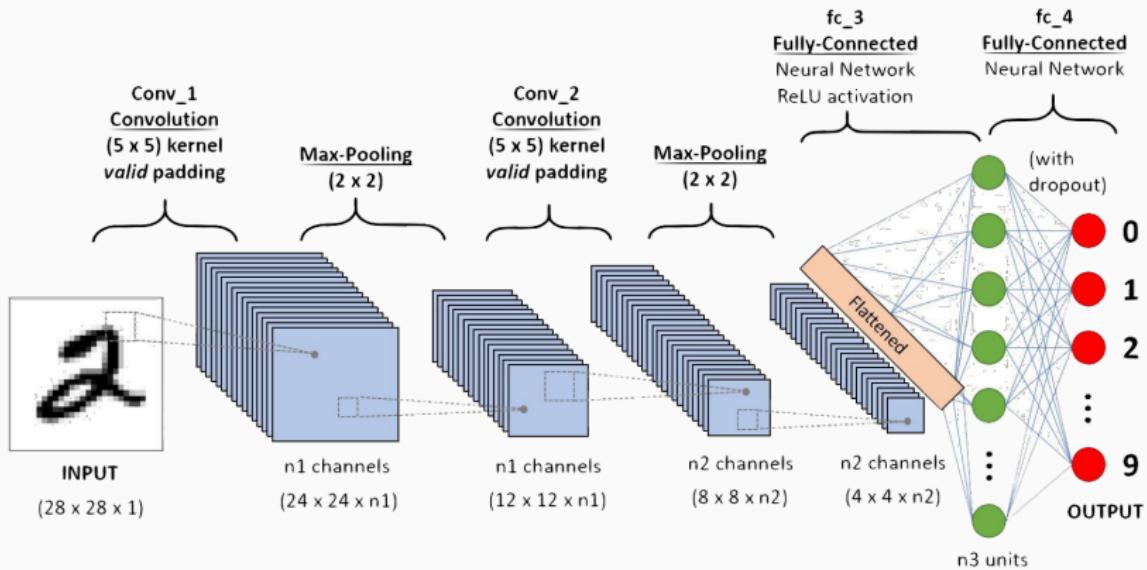
Many possible variations on standard 2x2 max-pooling.

An example Image Portion
for Max Pooling
Numbers represent
the pixel values

2	3	4	0
1	5	3	2
0	4	2	3
1	0	6	1



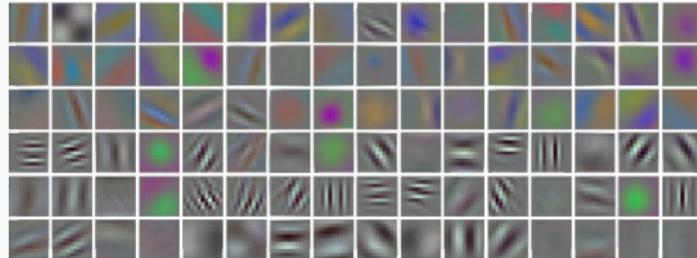
OVERALL NETWORK ARCHITECTURE



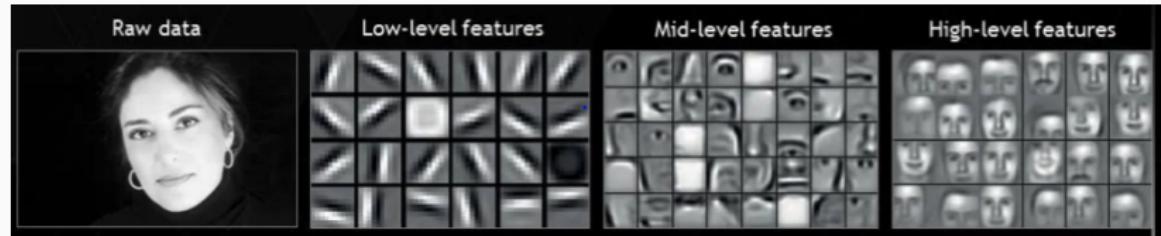
Each layer contains a 3D tensor of variables. Last few layers are standard fully connected layers.

UNDERSTANDING LAYERS

What type of convolutional filters do we learn from gradient descent? **Lots of edge detectors in the first layer!**

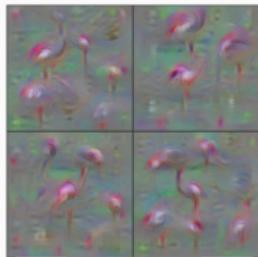


Other layers are harder to understand... but the hypothesis is that hidden variables later in the network encode for “higher level features”:

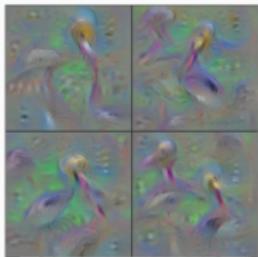


UNDERSTANDING LAYERS

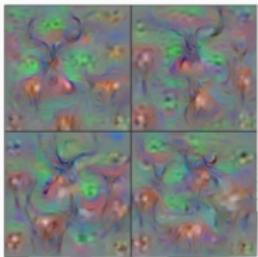
Technique to probe later neurons: Use optimization to find images that most strongly “activate” a given neuron deep in the network.



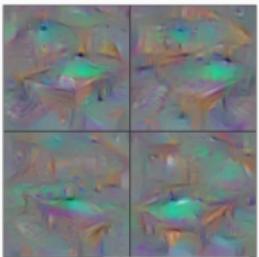
Flamingo



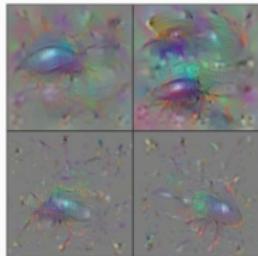
Pelican



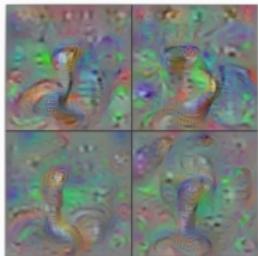
Hartebeest



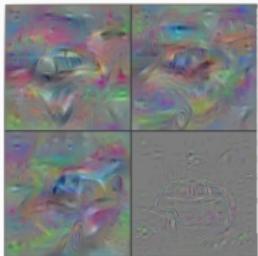
Billiard Table



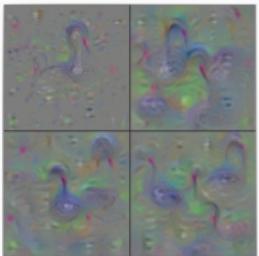
Ground Beetle



Indian Cobra



Station Wagon



Black Swan

TRICKS OF THE TRADE

Beyond techniques discussed for general neural nets (back-prop, batch gradient descent, adaptive learning rates) training convolutional networks requires a lot of “tricks”.

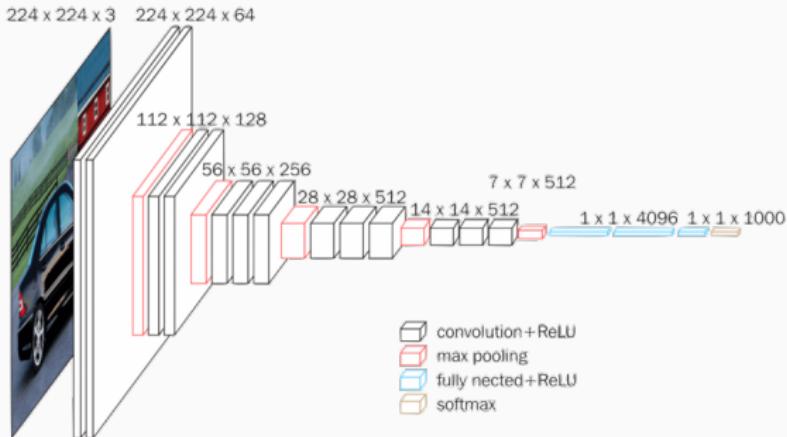
- Batch normalization (accelerate training).
- Dropout (prevent over-fitting)
- Residual connections (accelerate training, allow for more depth – 100s of layers).

And convolutional networks require **lots of training data**.

TRANSFER LEARNING

What if you want to apply deep convolutional networks to a problem where you don't have a lot of data?

Idea behind transfer learning: features transformations learned when training a classifier on e.g. Imagenet are often useful in other problems, even with different inputs, classes, etc.



TRANSFER LEARNING

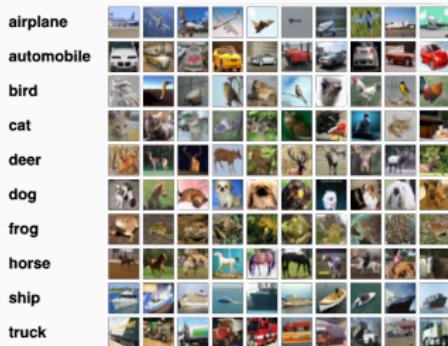
- Download state of the art pre-trained network (Alexnet, VGG, Inception, etc.)
- Chop off classification layer.
- Use first part of network as feature extractor.
- Solve classification problem using more scalable methods: kernel SVM, logistic regression, shallow fully connected net, etc.

Very easy to do in Tensorflow/Keras. Many pre-trained networks are made available.

DEMOS

Two demos to be released shortly:

- Classification of CIFAR-10 dataset using 2 layer neural nets in Keras. You will likely want to use Google Collab to access a GPU.



- Transfer learning in Keras.