Convolutional Neural Networks

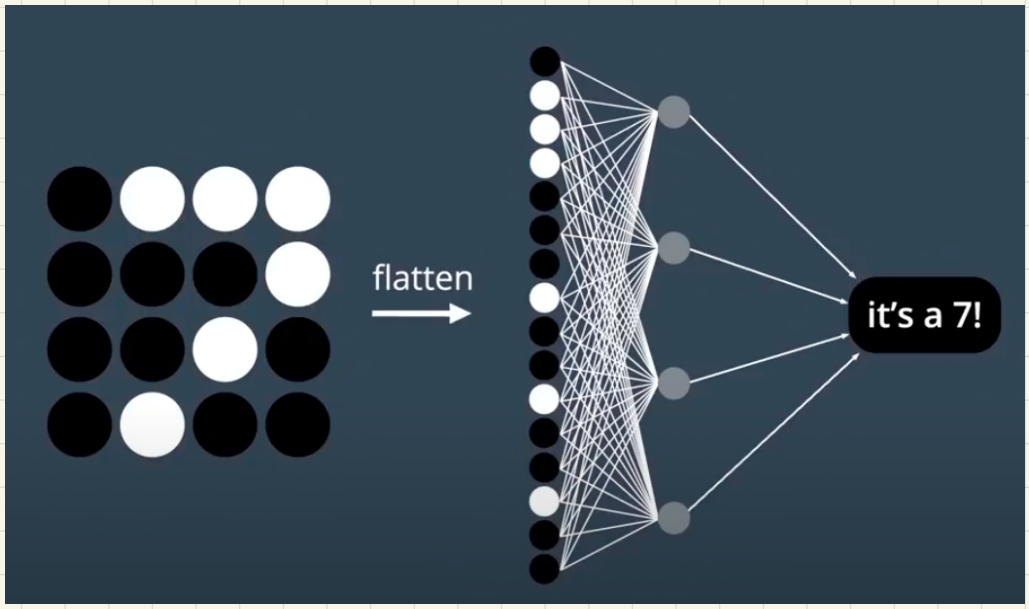
CNN

CNN vs. MLP

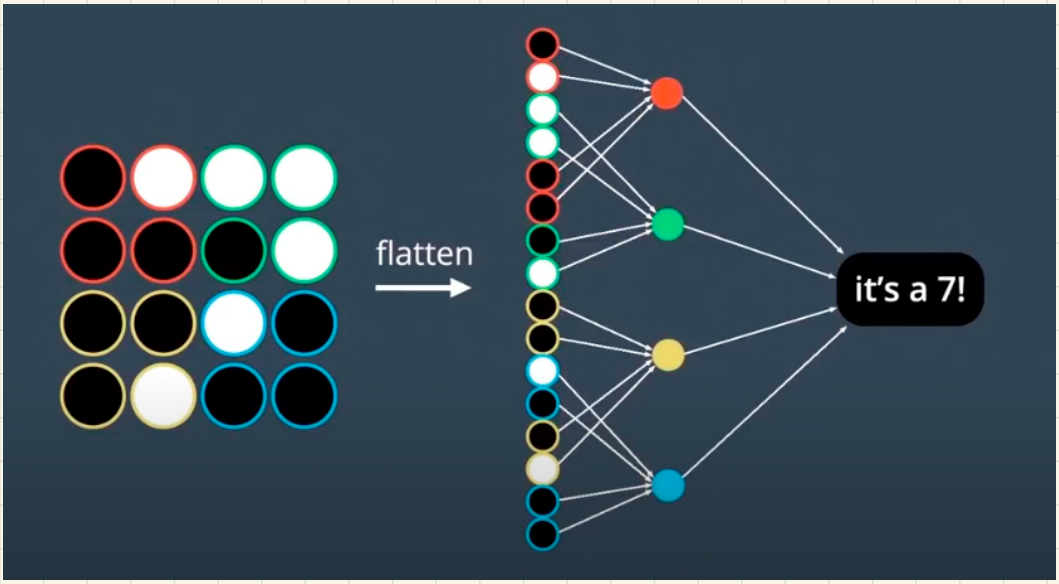
- Only use fully connected layer [Parameters ↑]
- Disgard 2D info when flattening images into vector

Local Connectivity

- CNN uses sparsely connected layer
- Accept matrices as input



In MLP, every hidden node is responsible for gaining the understanding of an entire image all at once.



- Less prone to overfitting
- Truly understand the pattern in the image data

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

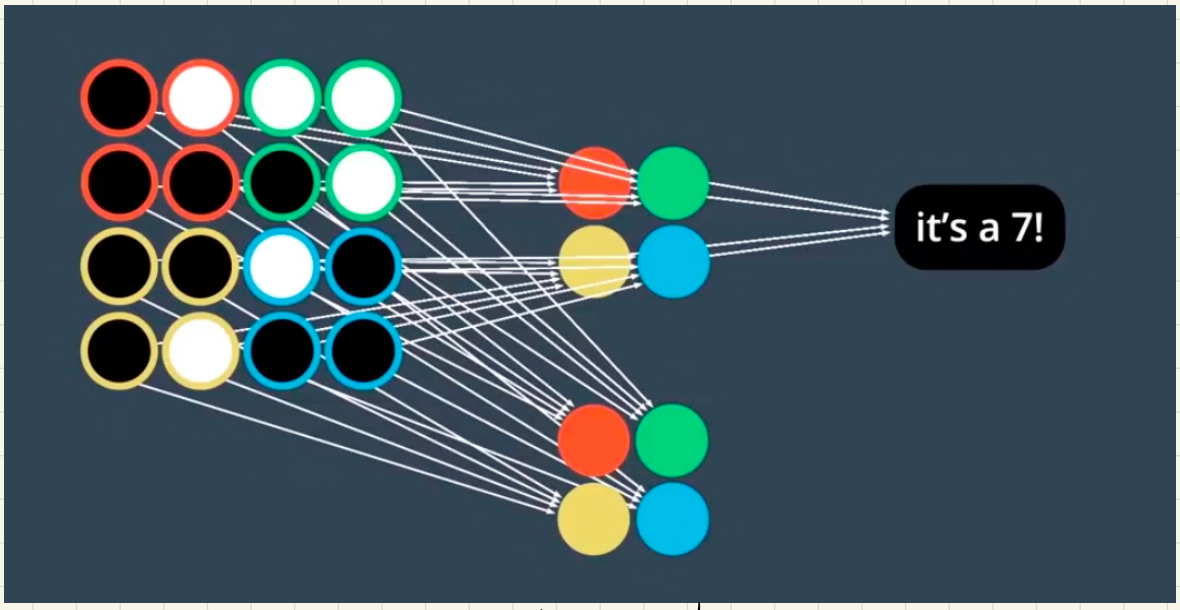
*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.



2 collection of hidden nodes

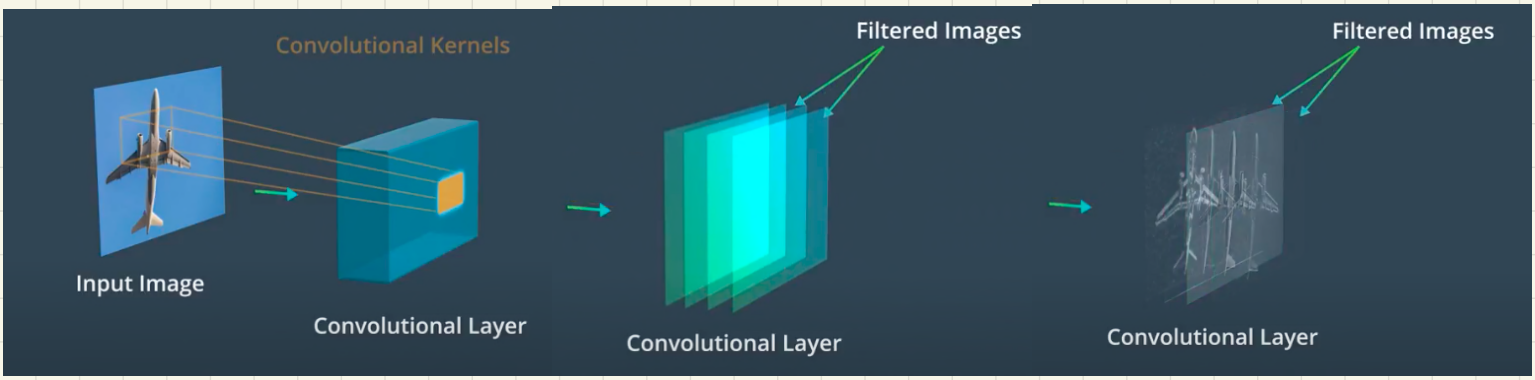
Each collection responsible for a diff. region of img.

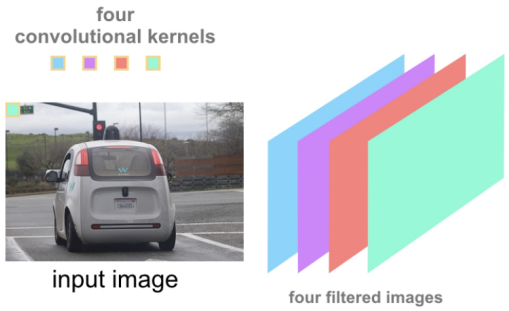
Each collection of diff color share common weights

For instance:

Doesn't matter where the cat is. Parameter Sharing makes it possible to detect patterns at diff. locations.

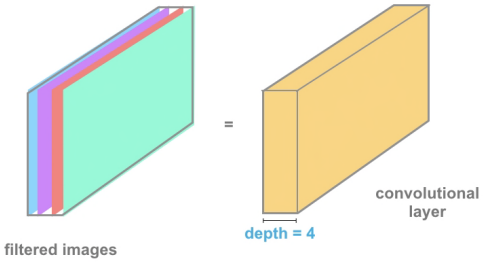
Filters & CNN Layers

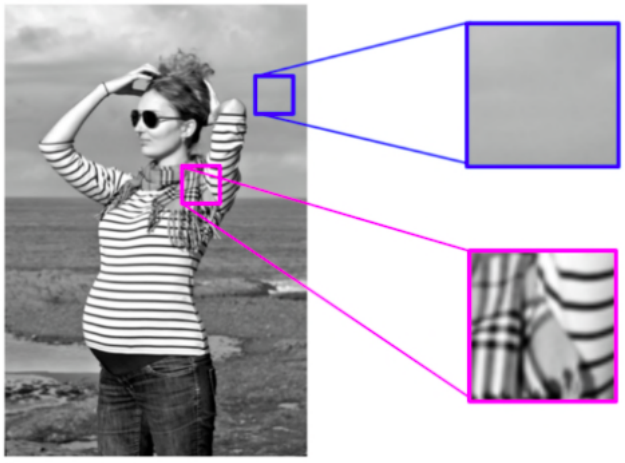




4 kernels = 4 filtered images.

In the example shown, 4 different filters produce 4 differently filtered output images. When we stack these images, we form a complete convolutional layer with a depth of 4!



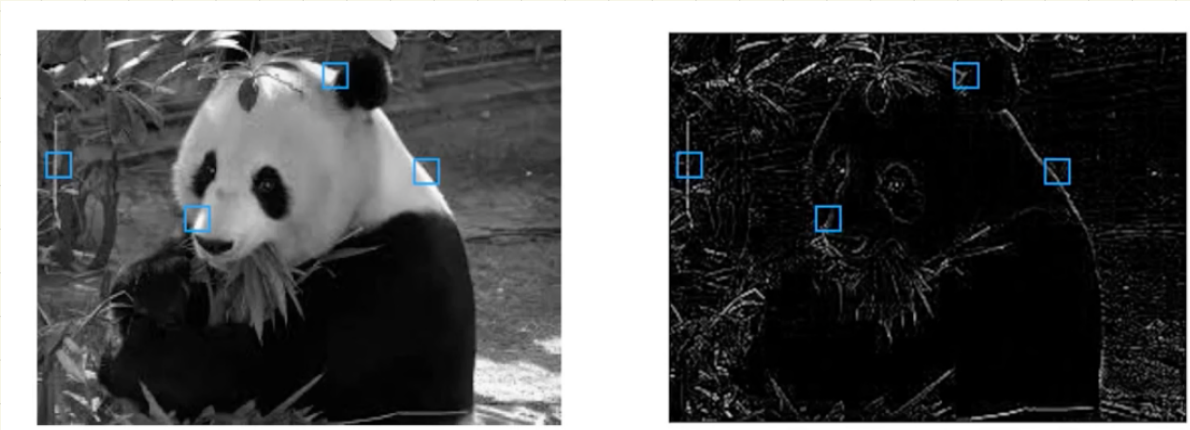


High and low frequency image patterns.

low freq:
not much change to the intensity

High freq:
Changes

High Pass Filters



High Pass Filters:

- Turn low intensity regions into black
- Draw white lines on high intensity region
- Emphasizing edges (area where intensity changes quickly and indicate object boundary)

CONVOLUTION KERNELS

0	-1	0
-1	4	-1
0	-1	0

edge detection filter

0 + -1 + 0 + -1 + 4 + -1 + 0 + -1 + 0 = **0**

Has to add up to 0 or images will be darken or lighten.

Convolve (*): Applying filters

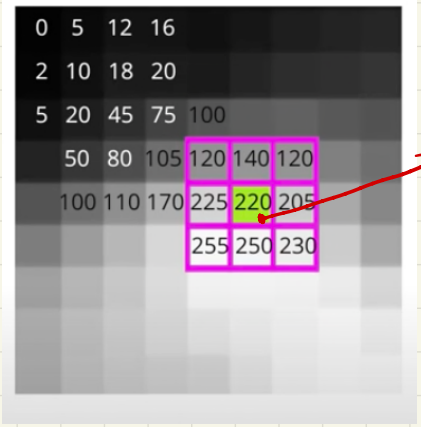
Weights

0	-1	0
-1	4	-1
0	-1	0

*

K F(x,y)

K * F(x,y) = output image



Pixel of interest

Pick a random pixel

0	-140	0
-225	880	-205
0	-250	0

= 60

Not a big value

Doesn't change rapidly

The weighted sum ≡ Pixel value in the output image

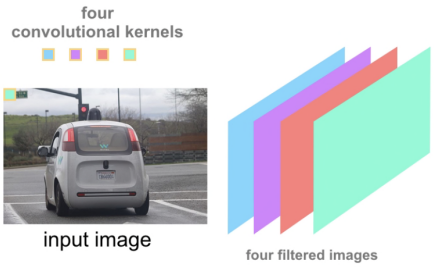
How about corners?

Extend The nearest border pixels are conceptually extended as far as necessary to provide values for the convolution. Corner pixels are extended in 90° wedges. Other edge pixels are extended in lines.

Padding The image is padded with a border of 0's, black pixels.

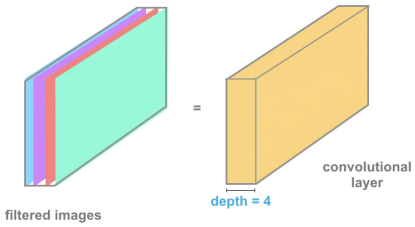
Crop Any pixel in the output image which would require values from beyond the edge is skipped. This method can result in the output image being slightly smaller, with the edges having been cropped.

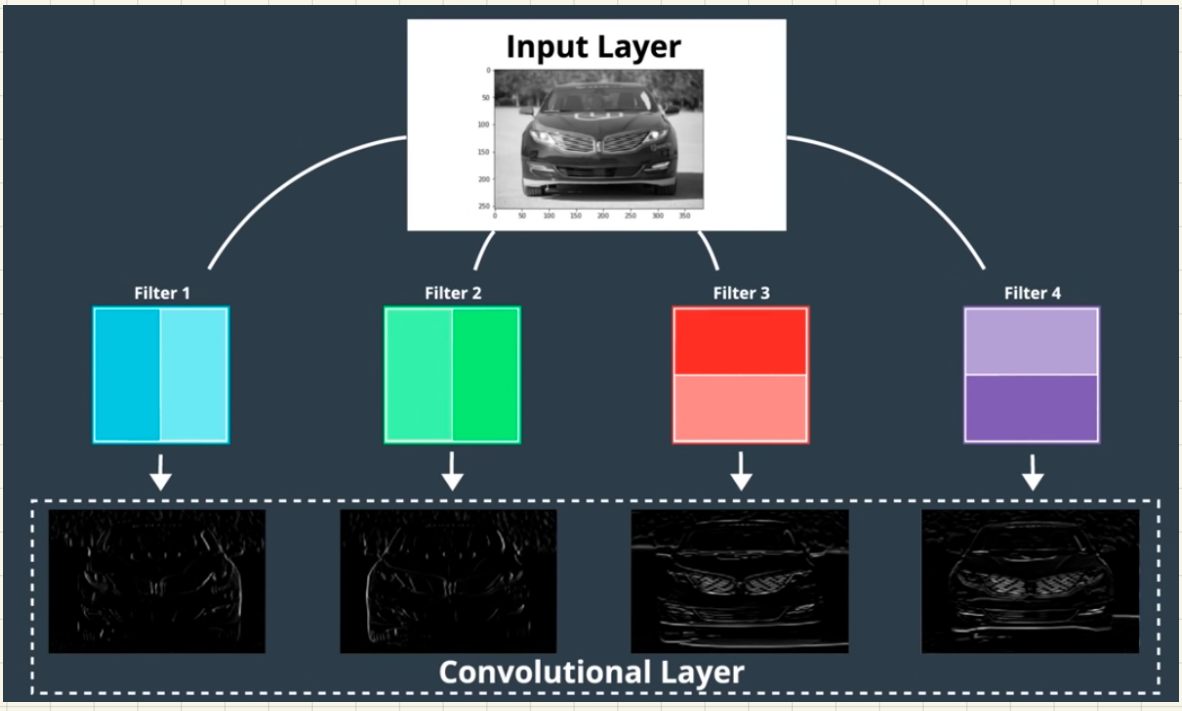
Convolutional Layers



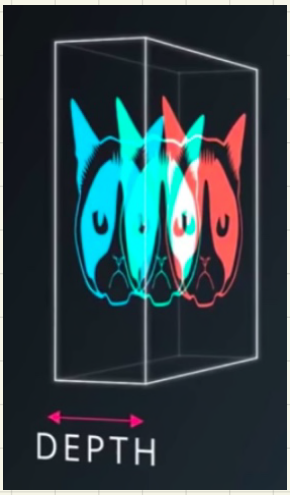
4 kernels = 4 filtered images.

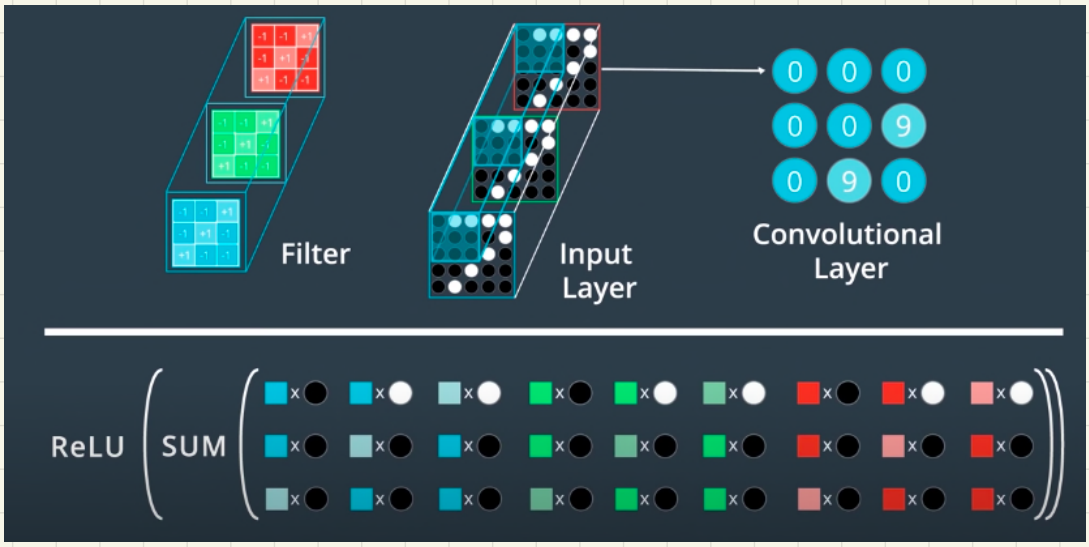
In the example shown, 4 different filters produce 4 differently filtered output images. When we stack these images, we form a complete convolutional layer with a depth of 4!



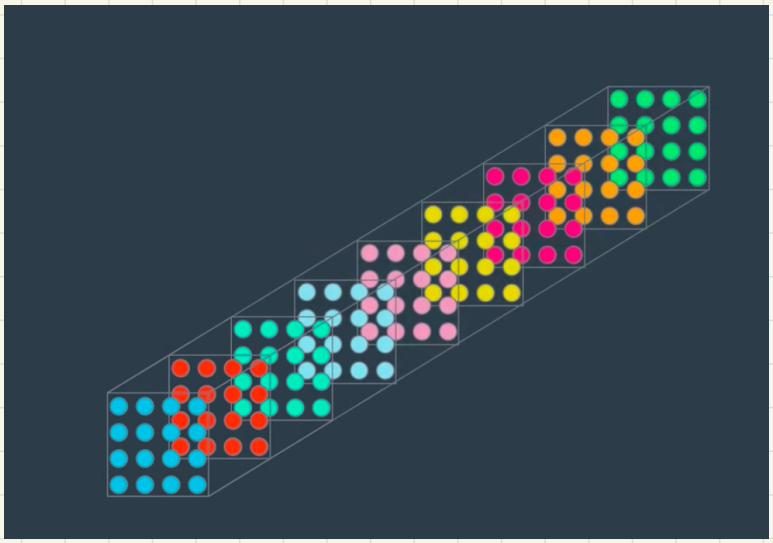


Color Images

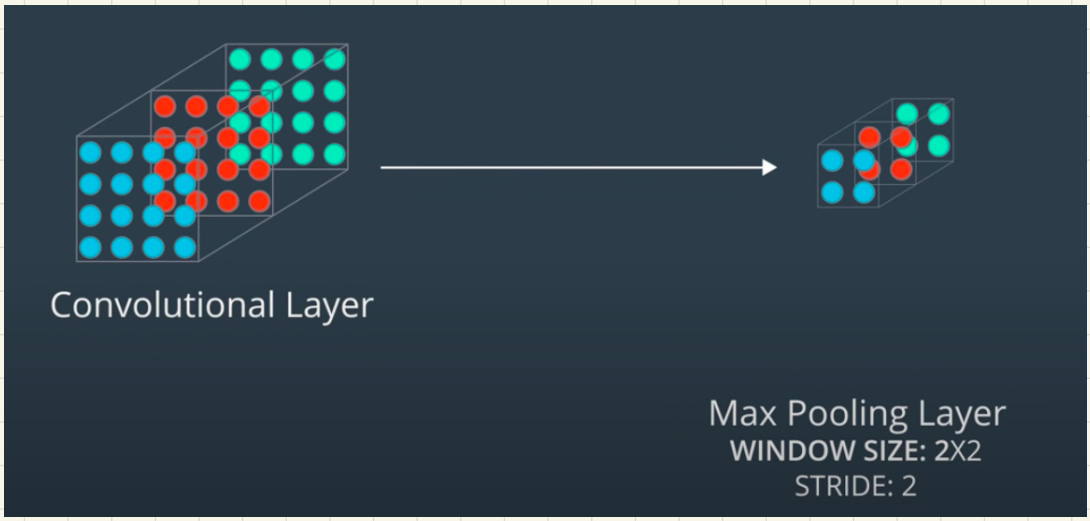




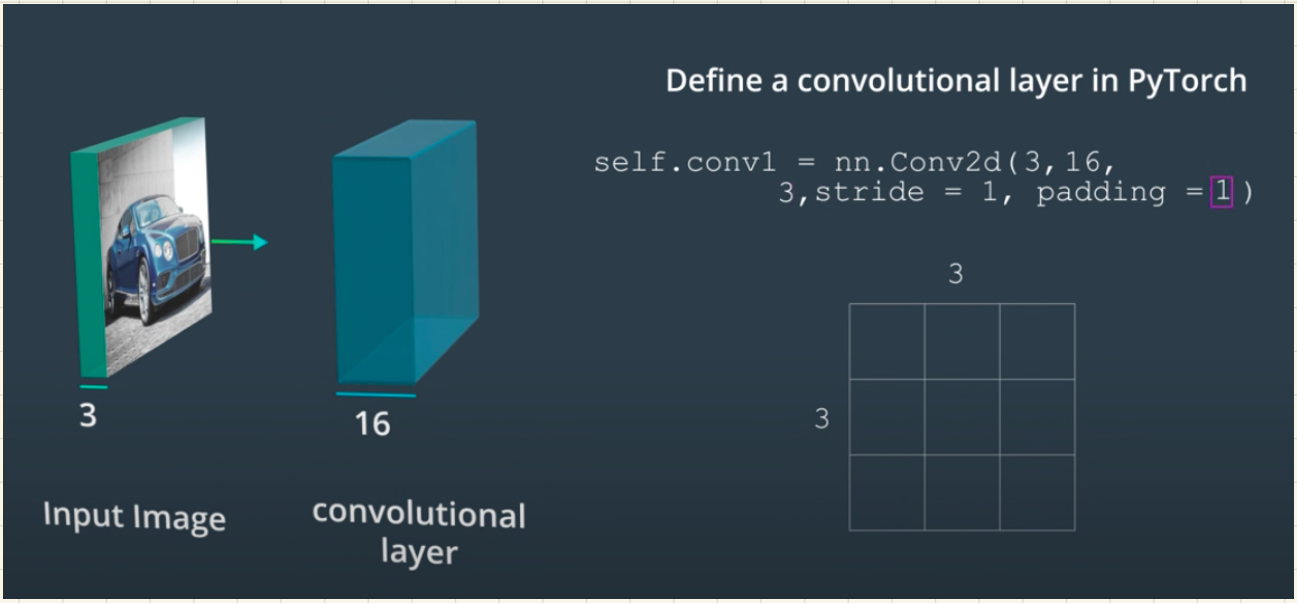
Pooling layer

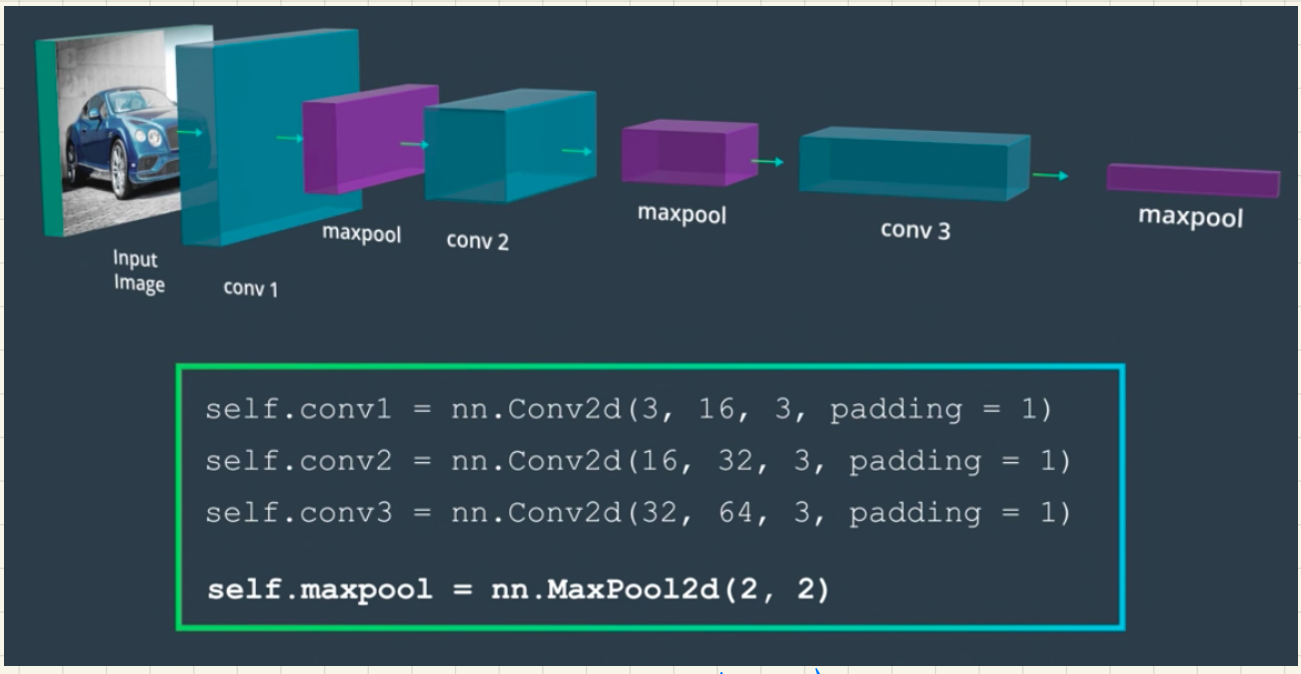


Complicated datasets w/ more objects
=> more filters
=> Dimensions ↑
=> Overfitting



Building a Network





Kernel Size Stride Size → Determines Scaledown factors

Conv layers make the array deep, while maxpooling decrease the height and width.

Deep: more complex, able to detect more features

↓ size: Disgard useless spatial components.

A bit of the notations:

self.conv = nn.Conv2d(16, 32, 3, p=1, s=1)

- 16: In channel or depth
- 32: out channel
- 3: kernel size (f)

130 x 130 x 3

- 130 → W_in
- 3 → in channel

Spatial dims: $\frac{(W\_in - f + 2p)}{s} + 1$

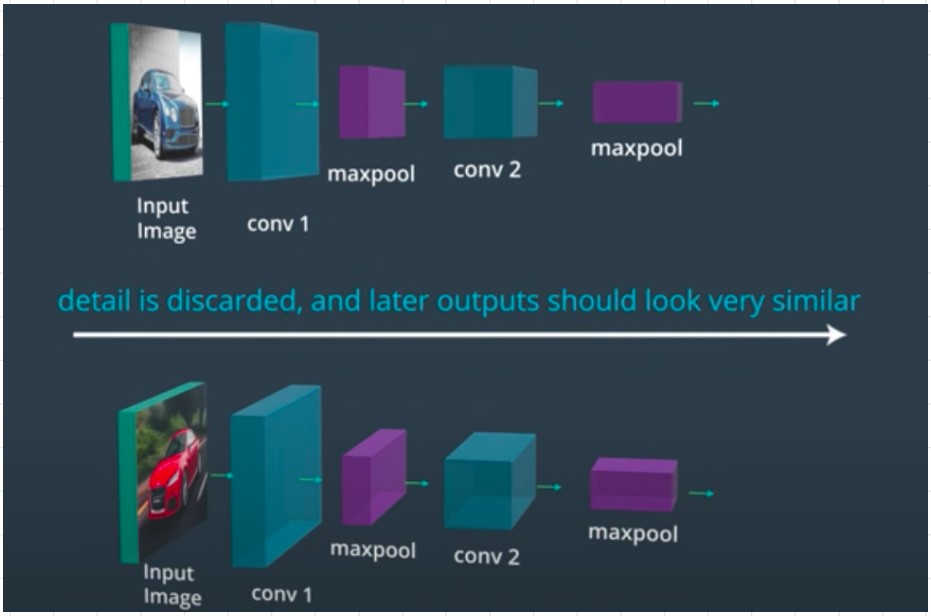
Take in a 3 channel image & produce a 16-filtered images

$\frac{28-3+2}{1} + 1 = 28$

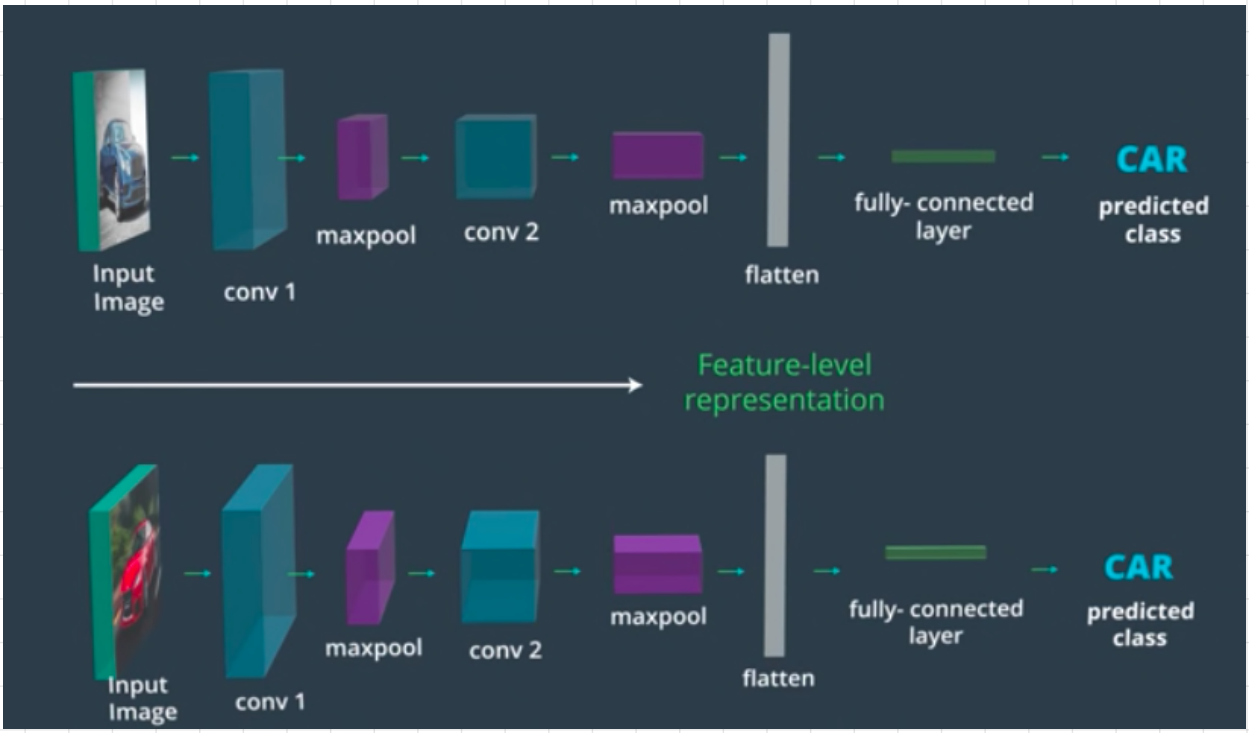
28 x 28 x 1 —conv1→ 28x28x32 —pool→ 14x14x32

14x14x64 —conv2→ 7x7x64 —pool2→ 3.5x3.5x4 —pool3 + conv3

Feature Vector



- Details about images are discarded (about what the car looks like)
- Just care about it is a car!
- Questions about general shape & unique patterns



After details have been distilled, all is fed into a fc.

Grand Breaking Architectures.

- AlexNet
  - Dropout & ReLU
- VGG
  - VGG16/19. long sequence of 3x3 conv, sep. by 2x2 pooling layers, 3 fc layers
- ResNet
  - ↑ layers → ↓ performance, ∵ vanishing grad
  - Skipping layers

Visualizing CNN

The CNN learns to do this on its own. There is no special instruction for the CNN to focus on more complex objects in deeper layers. That's just how it normally works out when you feed training data into a CNN.

Summary:

- Passes an image through convolutional blocks
  - conv
  - pooling
- Result: a set of feature map w/ reduction in size from the input image
- Through training process, complex features had been distilled → only care about general shape & unique characteristics
- Flatten feature map → vector → fc.
  fc → Prob. distribution of class scores
- Extracts predicted class label