Week 6: Convolutional Neural Networks

Some slides adapted from Stanford Deep Learning Course(CS231n) presentations and slides.

HELLO!

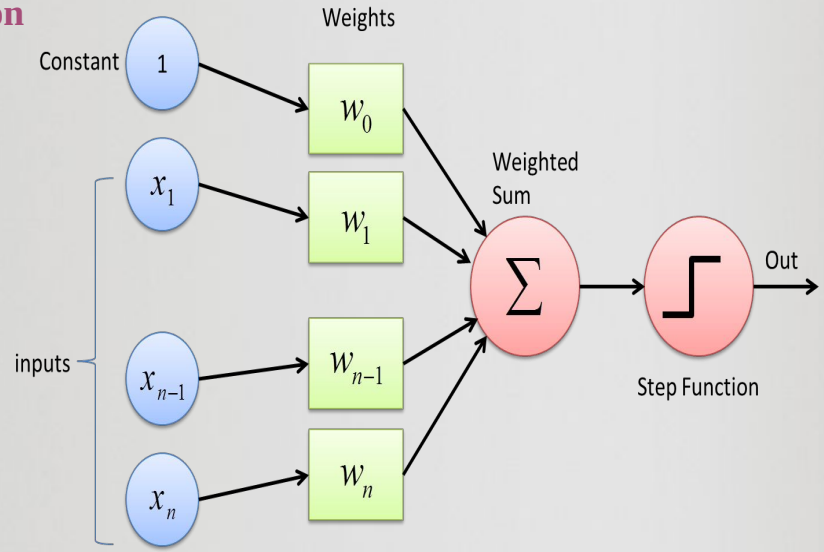
I am Payam Pourashraf

I am here to talk about Convolution Neural Networks.

A bit of history...

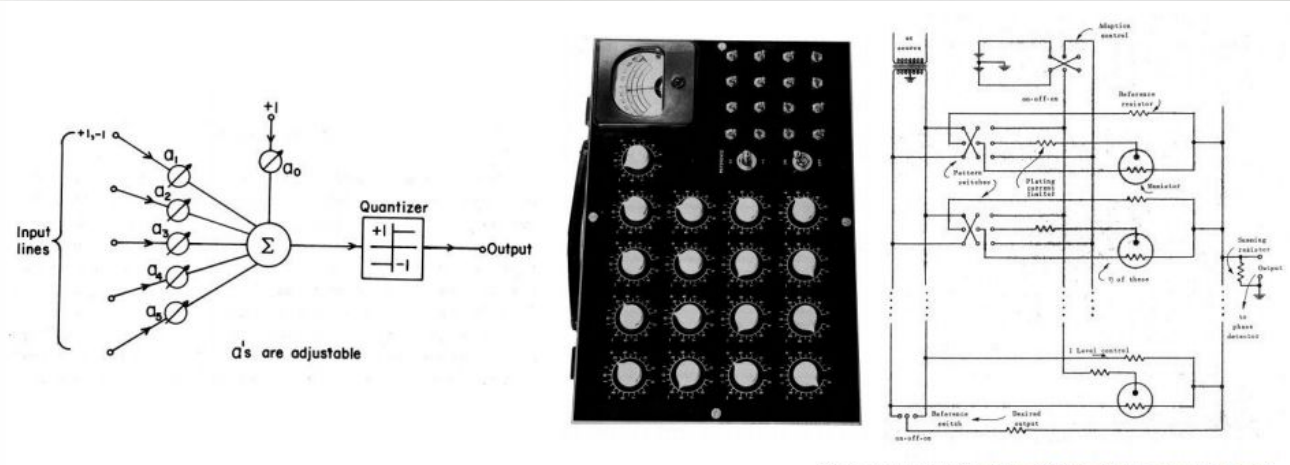
Frank Rosenblatt, ~1957: Perceptron

Recognized letters of the alphabet



A bit of history...

Widrow, 1960: Adeline/Madeline



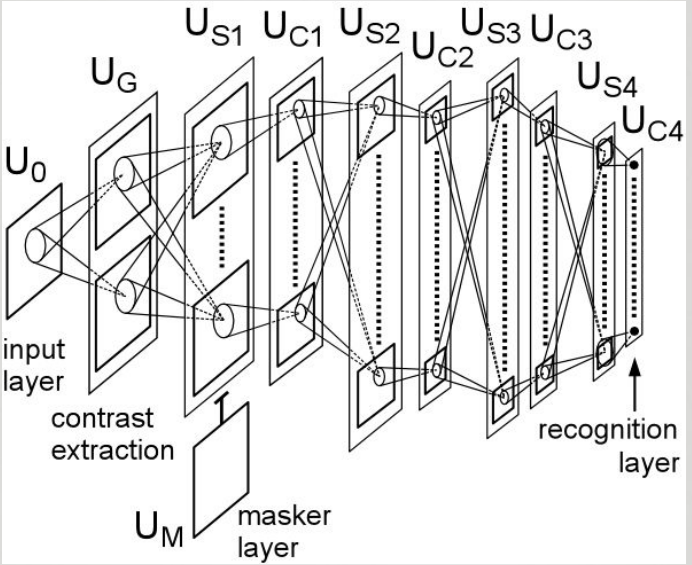
A bit of history…

Fukushima, 1980: Neocognitron

“sandwich” architecture (SCSCSC…)

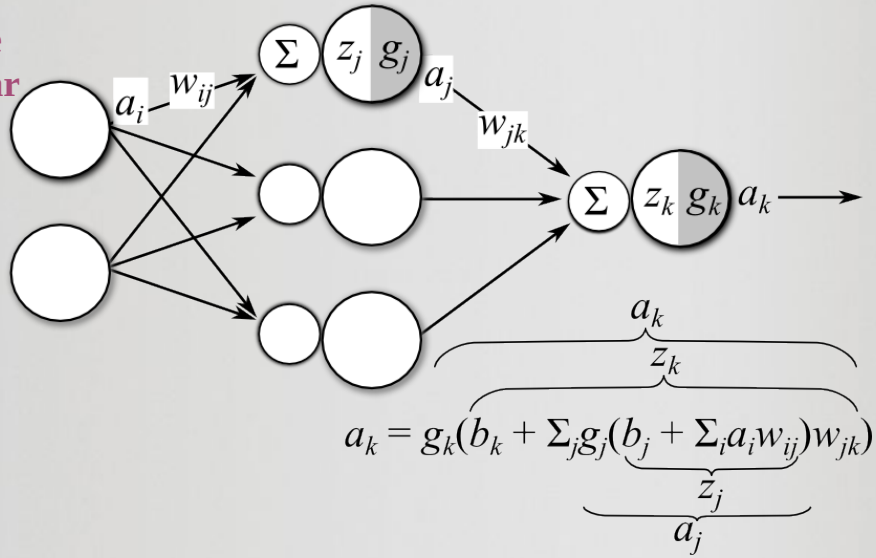
simple cells: modifiable parameters

complex cells: perform pooling



A bit of history...

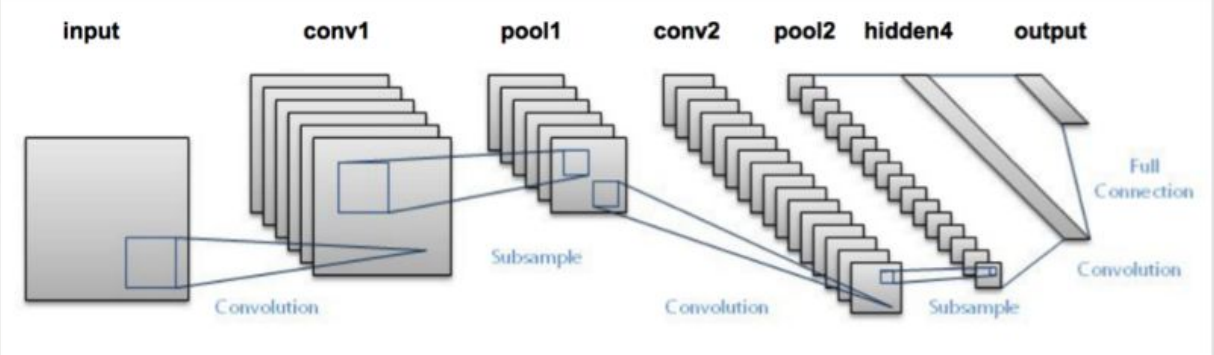
Rumelhart et al., 1986: First time back-propagation became popular



A bit of history...

[LeCun, Bottou, Bengio, Haffner], 1998: LeNet

Gradient-based learning applied to document recognition.



What is an Image?

Image

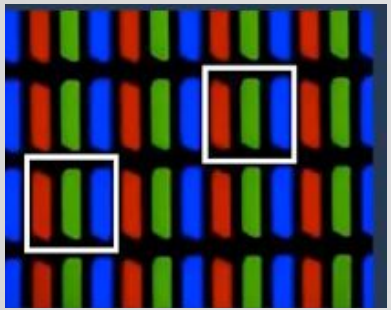
An image is matrix that specifies the color of various pixels in terms of the amount of red, green and blue components.

Every pixel is assigned an RGB value, each component being a value between 0 to 255. These components are mixed together to form a unique color value.

E.g. Blue is rgb(0,0,255) and Black is rgb(0,0,0)

Displaying an Image

Any display of an image consists of an arrangement of red, green, and blue dots. A set of one dot of each color form a pixel



What is a Kernel Filter?

Kernel

A kernel is a square matrix that specifies spatial weights of various pixels in an image.

Different image processing techniques have different kernels.

Example: A kernel for blurring is (1/9)*

1	1	1
1	1	1
1	1	1

Some famous kernels

1	1	1
1	1	1
1	1	1

3*3 Mean Kernel

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

5*5 Gaussian Kernel

2	1	0
1	0	-1
0	-1	-2

3*3 Sobel Kernel

What is Convolution?

BIG CONCEPT

Convolution

Convolution of a matrix involves laying a matrix over another and then calculating the weighted sum of all pixel values.

0	0	0	0	0	0
0	105	102	100	97	96
0	103	99	103	101	102
0	101	98	104	102	100
0	99	101	106	104	99
0	104	104	104	100	98

Image Matrix

Kernel Matrix

0	-1	0
-1	5	-1
0	-1	0

320					

Output Matrix

$$0*0+0*-1+0*0$$
$$+0*-1+105*5+102*-1$$
$$+0*0+103*-1+99*0 = 320$$

Convolution with horizontal and vertical strides = 1

Image Processing

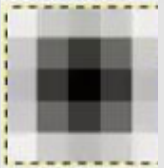
How to process an image?

Input Image → (Kernel / Convolution) → Output Image

100	100	100	100	100
100	0	0	0	100
100	0	0	0	100
100	0	0	0	100
100	100	100	100	100

(1/16)*

1	2	1
2	4	2
1	2	1



94	81	75	81	94
81	44	25	44	81
75	25	0	25	75
81	44	25	44	81
94	81	75	81	94

RGB to Grayscale

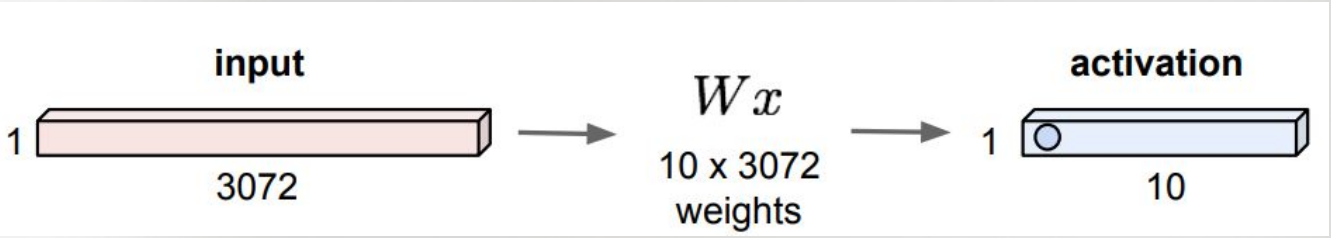
Edge detection

-1	0	1
-2	0	2
-1	0	1

Convolution Neural Network

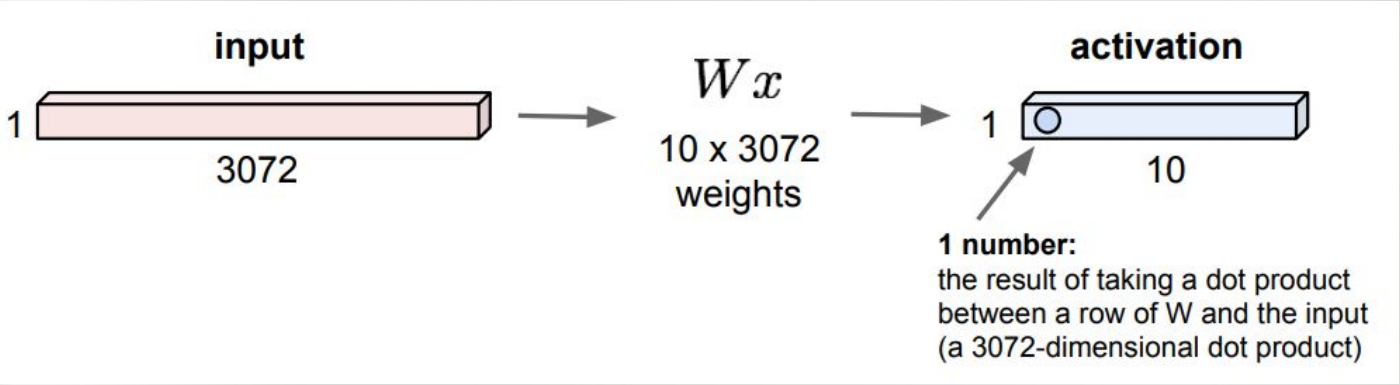
Fully Connected Layer

32*32*3 image -> stretch to 3072*1



Fully Connected Layer

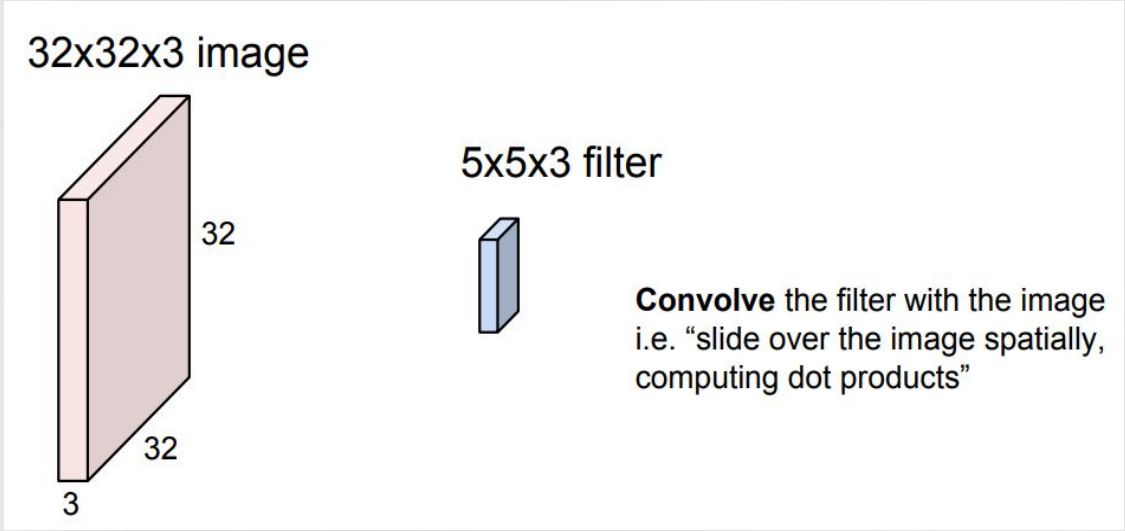
32*32*3 image -> stretch to 3072*1



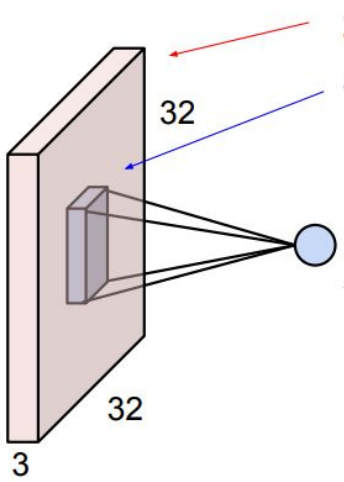
Convolution Layer

32*32*3 image -> Preserve spatial structure

Convolution Layer



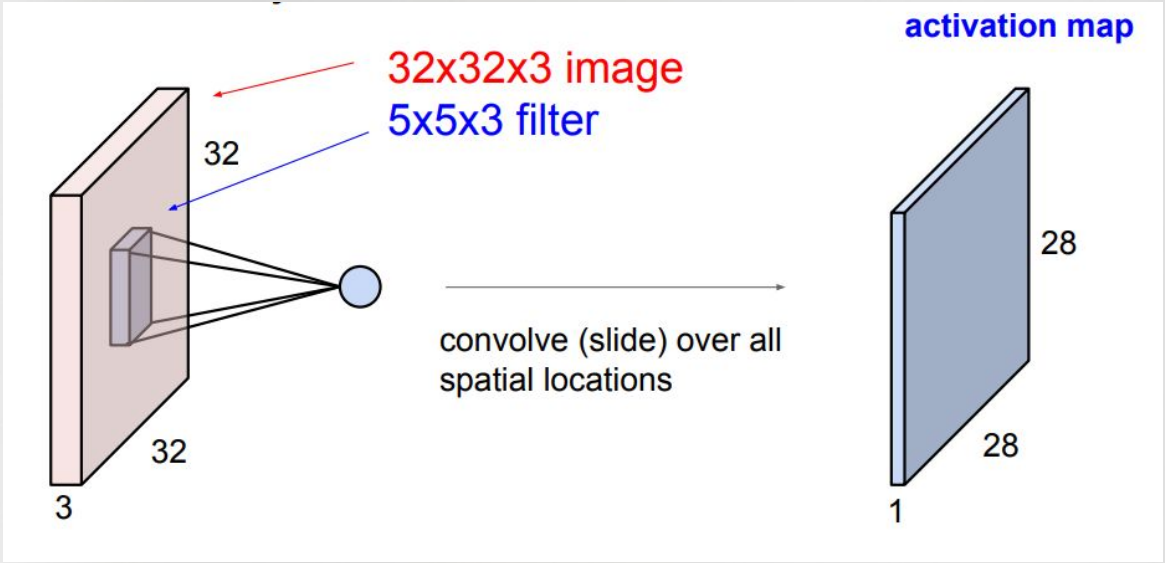
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

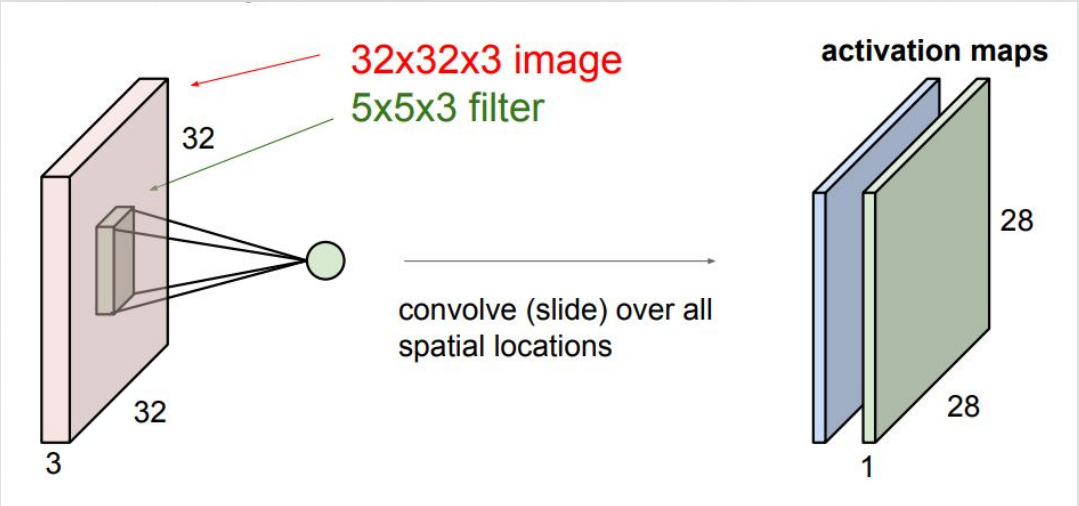
$$w^T x + b$$

Convolution Layer



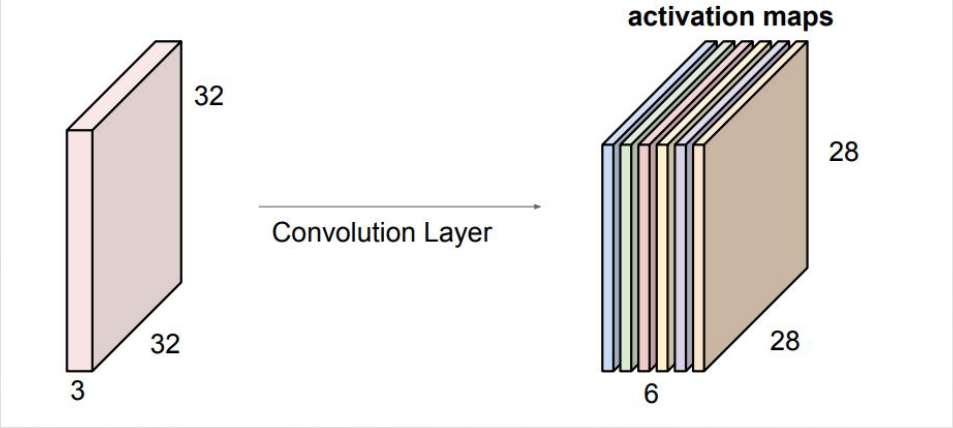
Convolution Layer

Consider a second, green filter



Convolution Layer

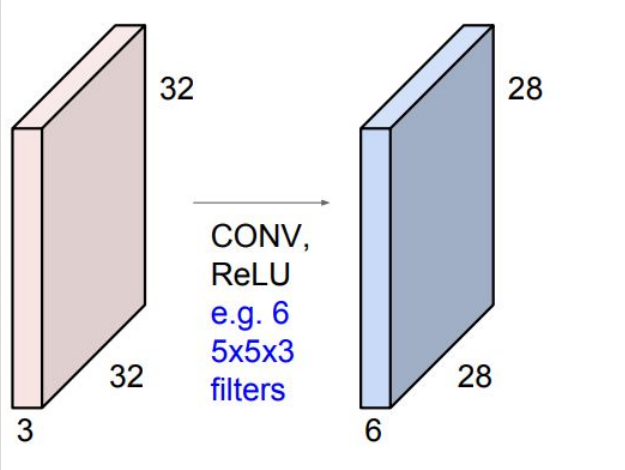
E.g. if we had 6 5*5 filters, we'll got 6 separate activation maps



We stack up these up to get a "new image" of size 28*28*6

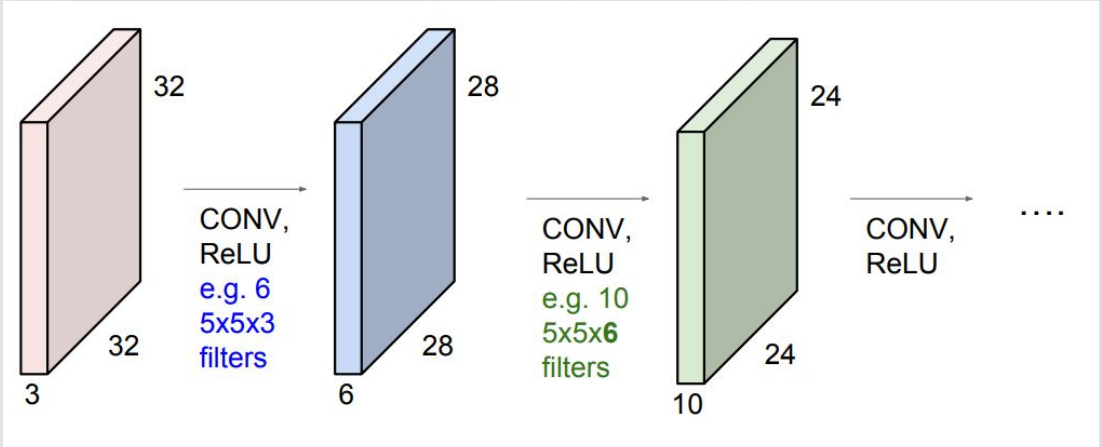
Convolution Layer

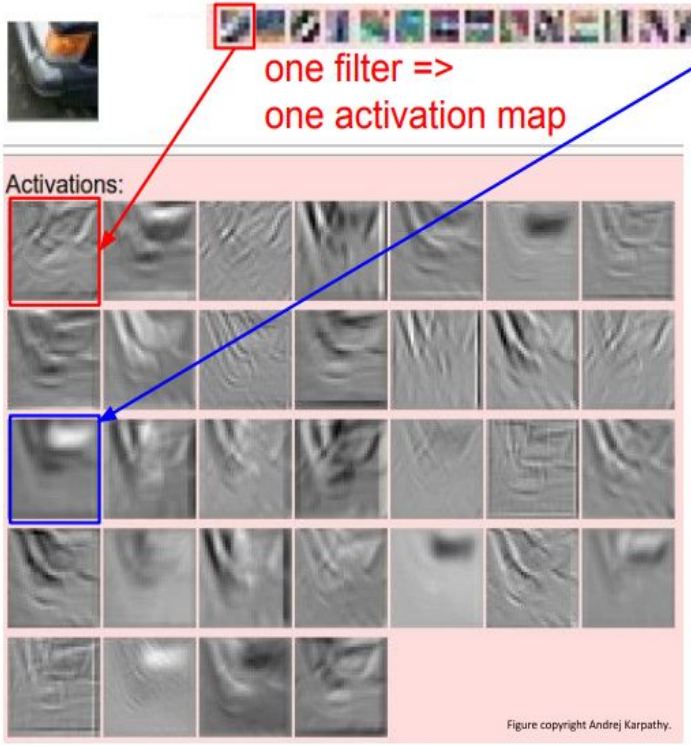
ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolution Layer

ConvNet is a sequence of Convolution Layers, interspersed with activation functions





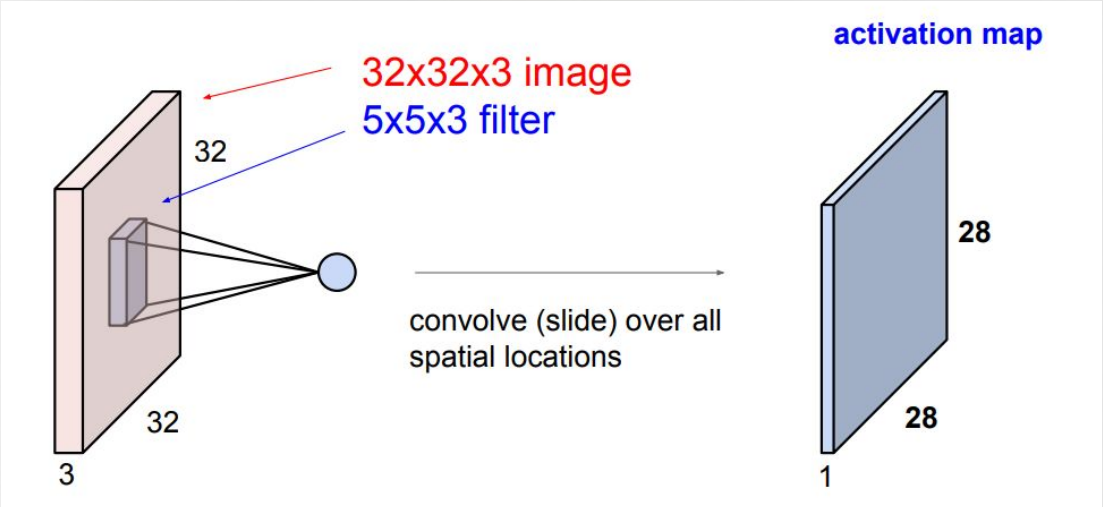
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

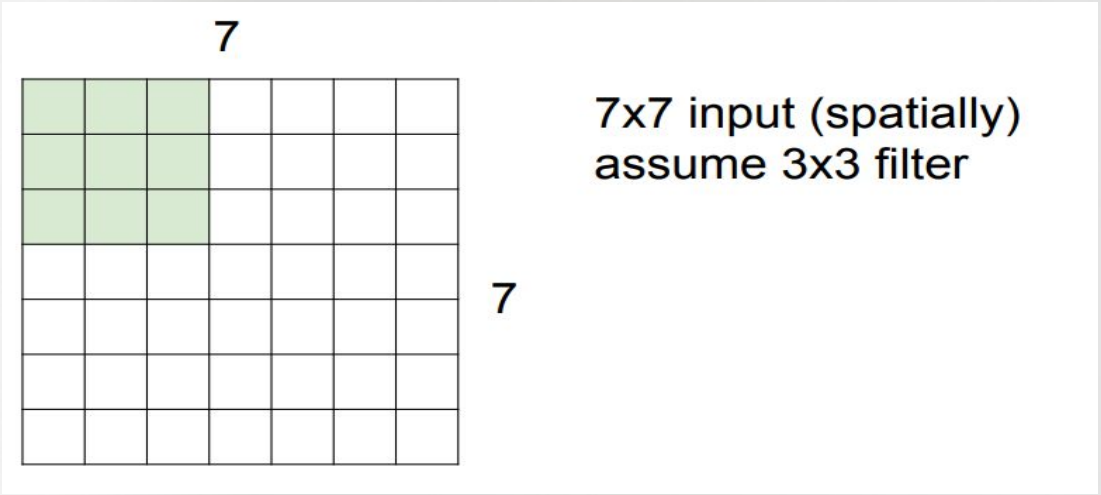
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

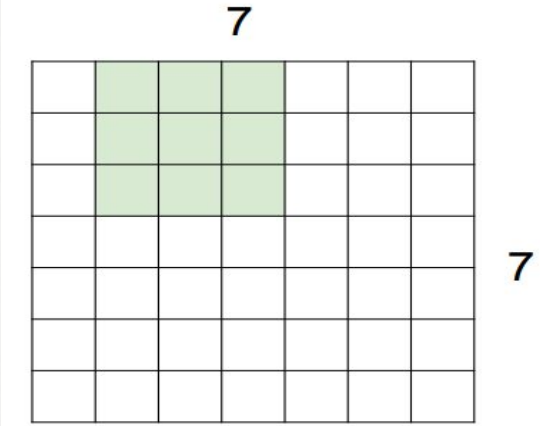
A closer look at spatial dimension



A closer look at spatial dimension

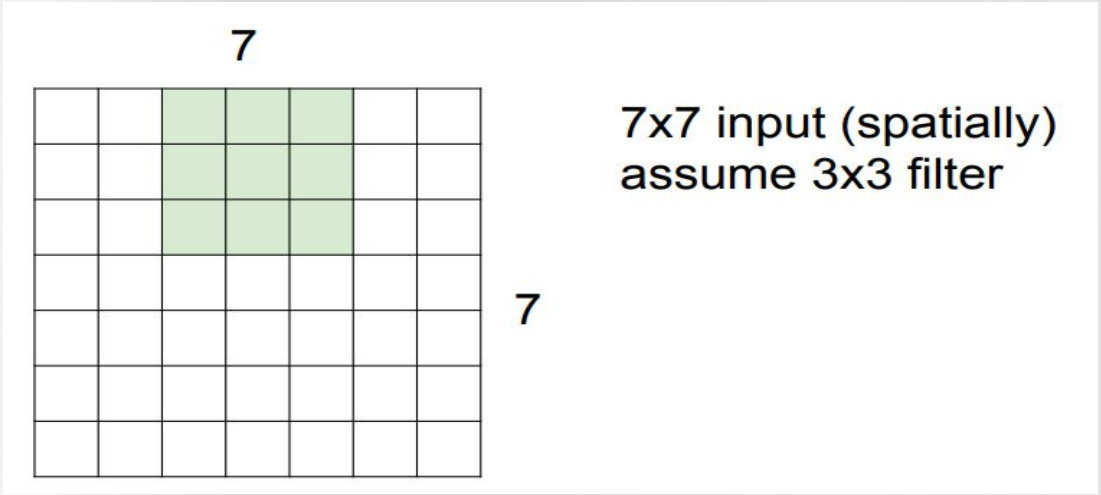


A closer look at spatial dimension

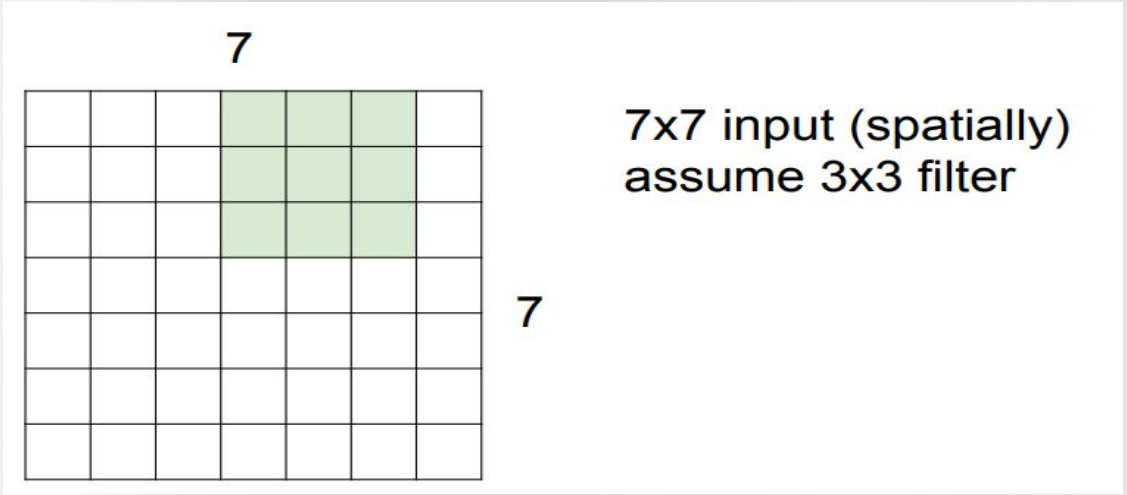


7x7 input (spatially)
assume 3x3 filter

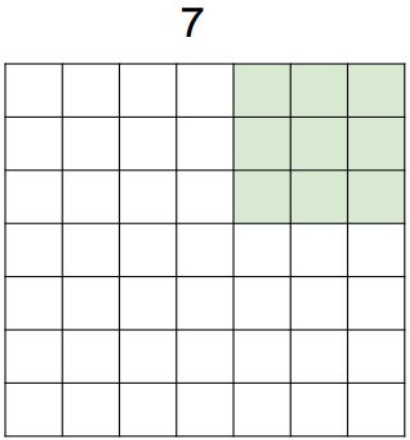
A closer look at spatial dimension



A closer look at spatial dimension



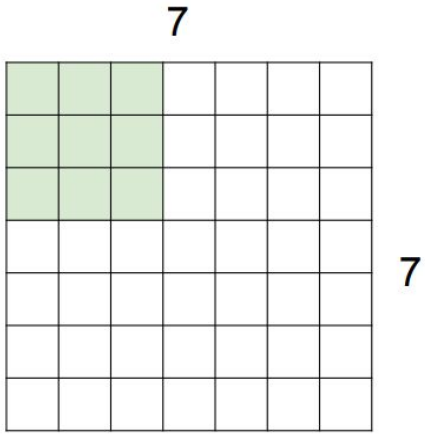
A closer look at spatial dimension



7x7 input (spatially)
assume 3x3 filter

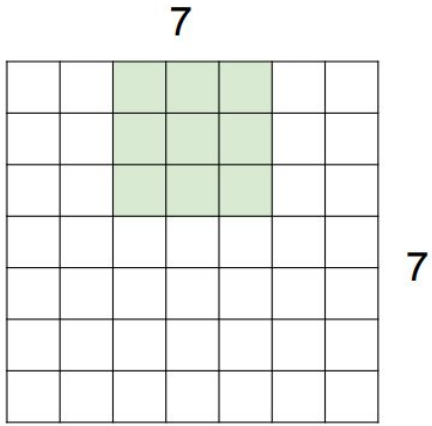
=> 5x5 output

A closer look at spatial dimension



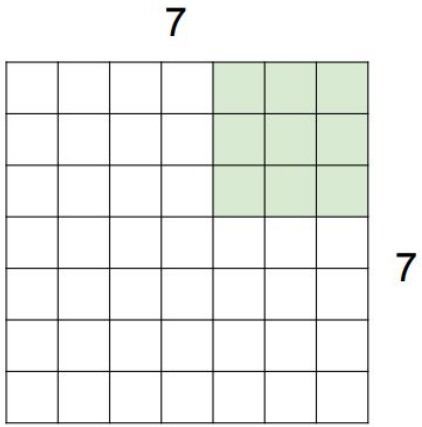
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimension



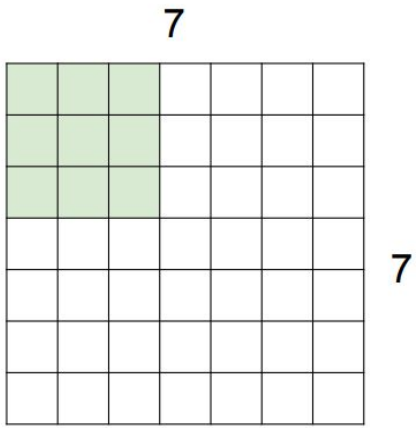
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimension



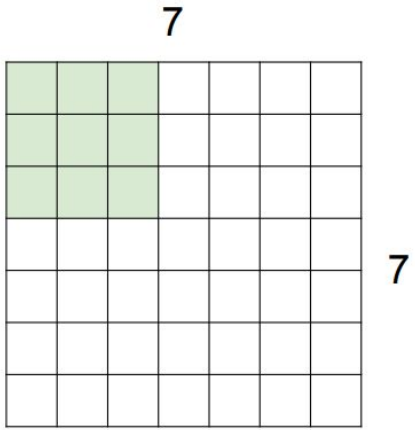
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimension



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

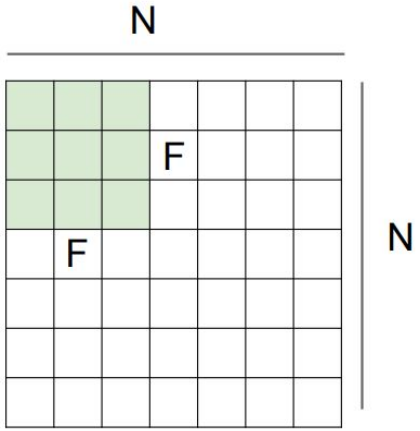
A closer look at spatial dimension



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

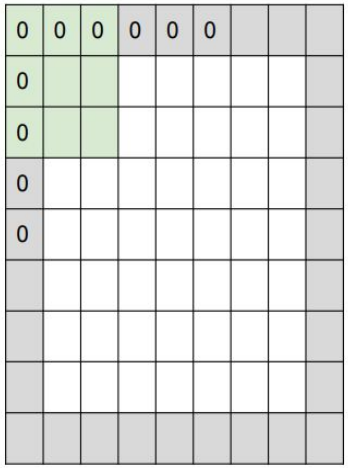
A closer look at spatial dimension



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

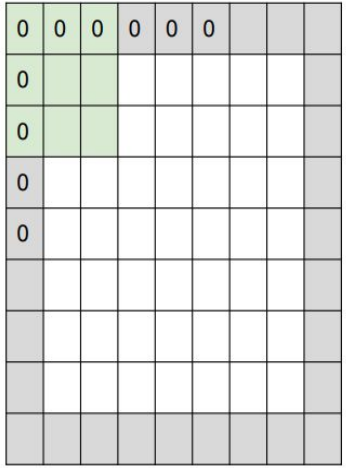
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

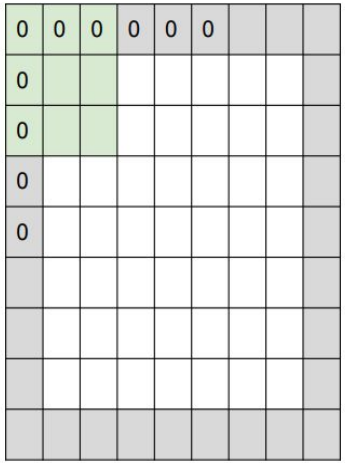
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border

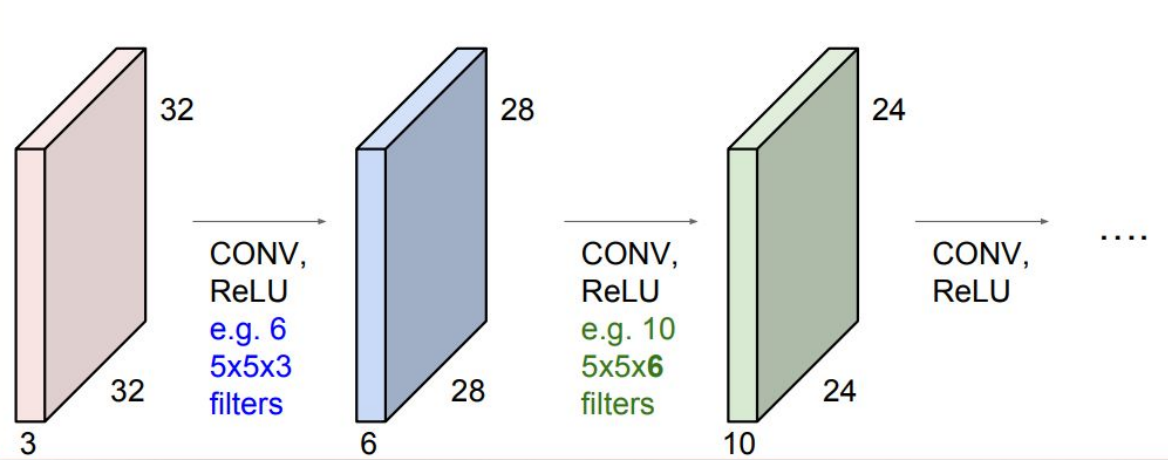


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

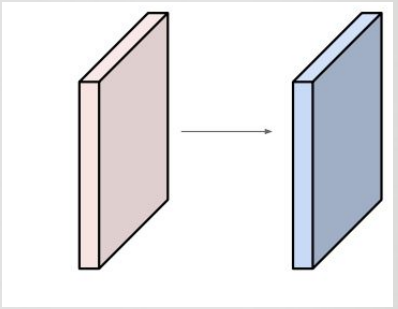
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to:

E.g. 32*32 input convolved repeatedly with 5*5 filters shrinks volumes spatially! Shrinking too fast is not good.



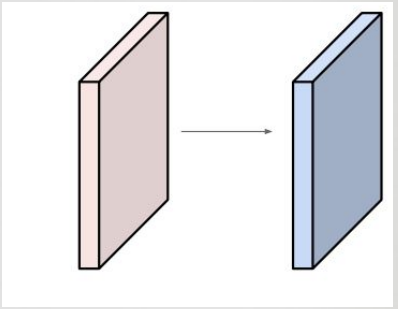
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Examples time:



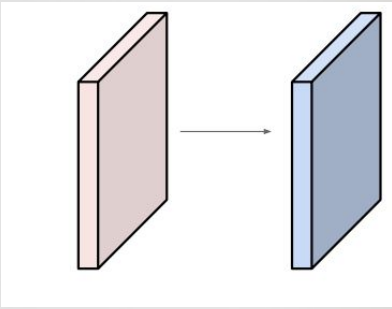
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Examples time:

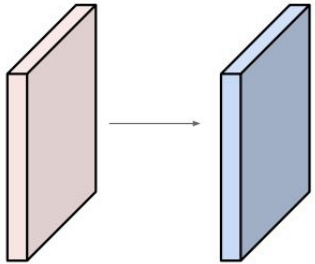
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?



Examples time:

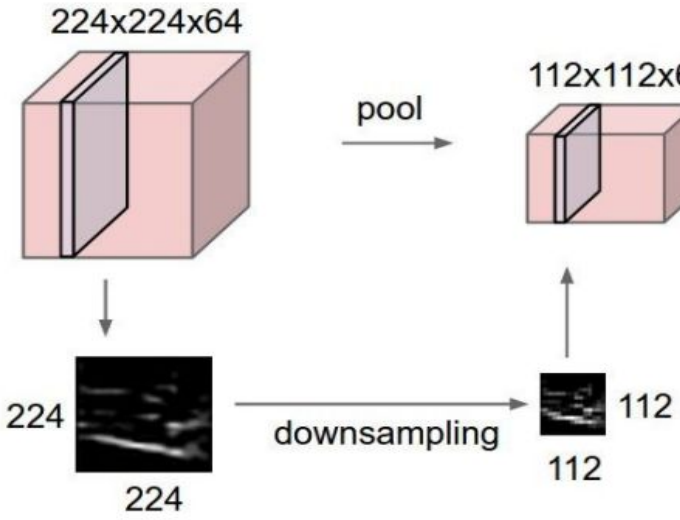
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



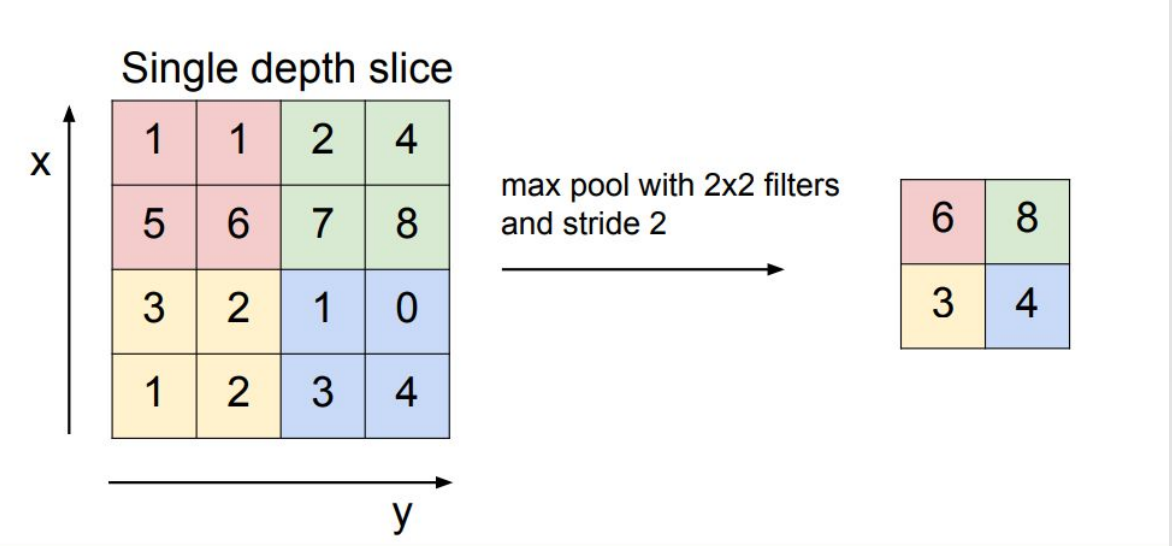
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Pooling Layer

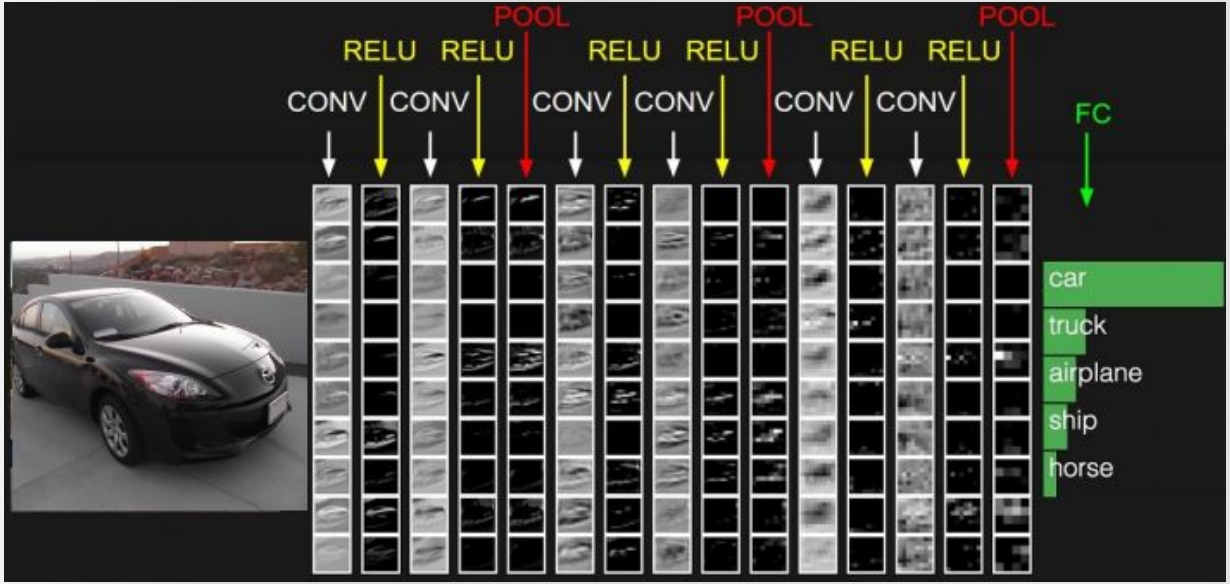
Makes the representations smaller and more manageable



Max Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Fully Connected Layer



CNN Architectures

AlexNet

[Krizhevsky et al. 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

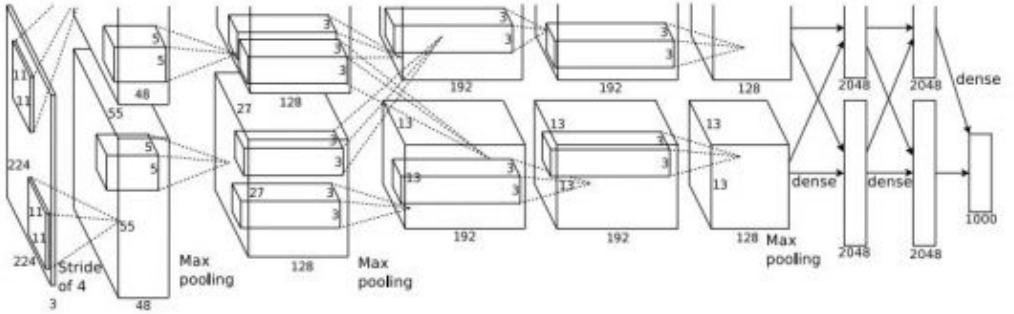
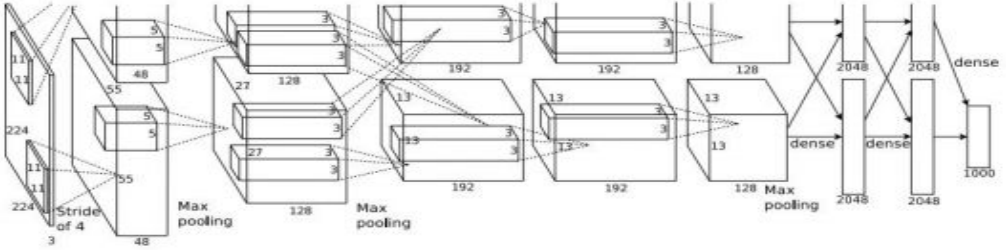


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet

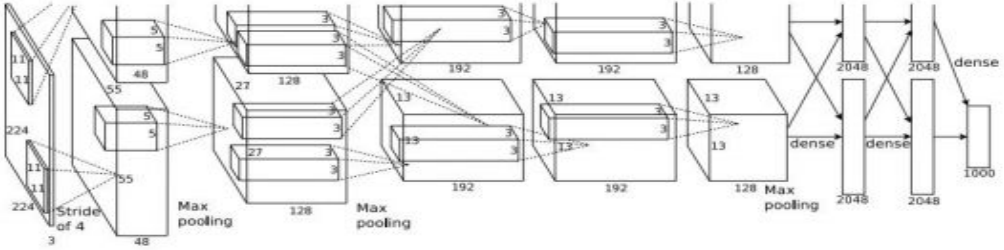


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

AlexNet

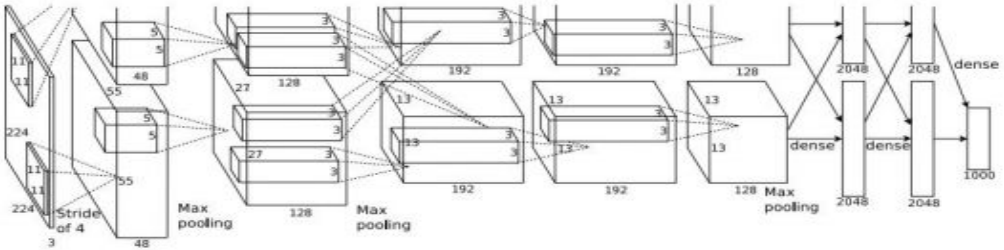


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

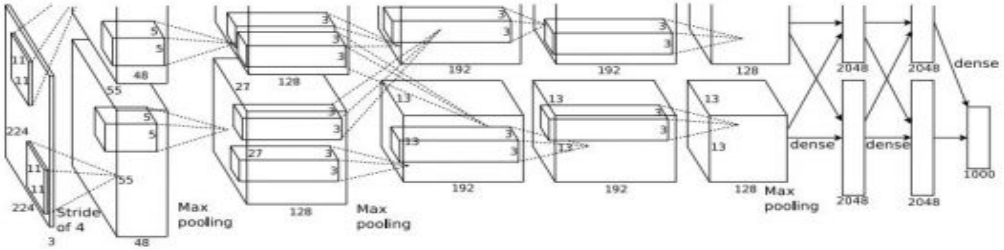
AlexNet



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

AlexNet

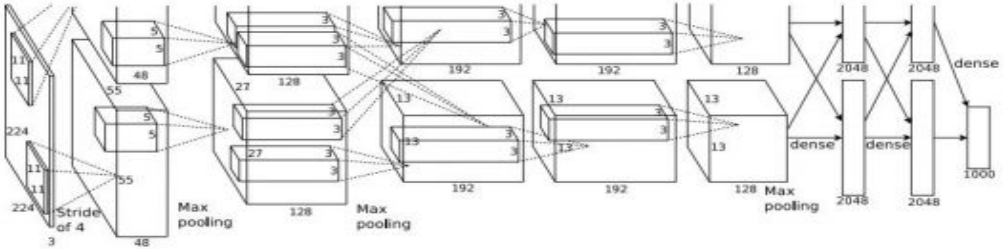


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

AlexNet

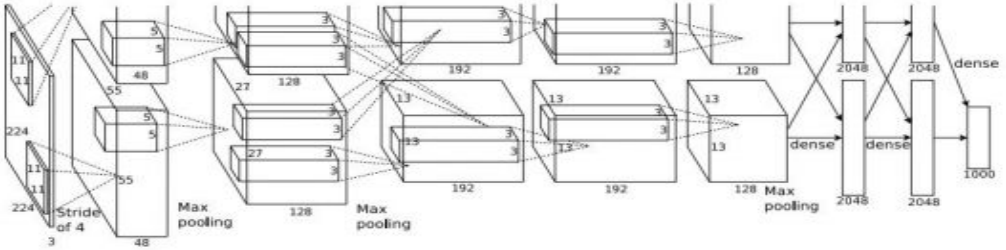


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

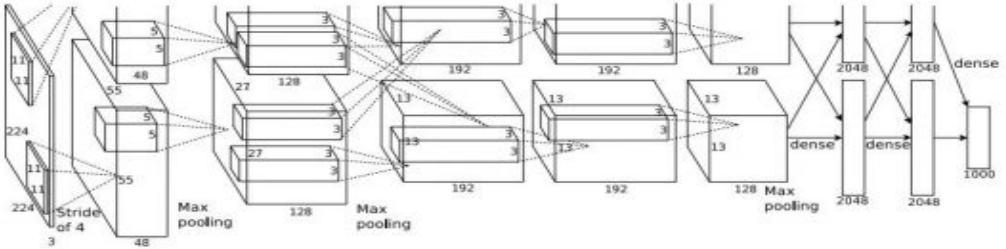
AlexNet



Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

AlexNet

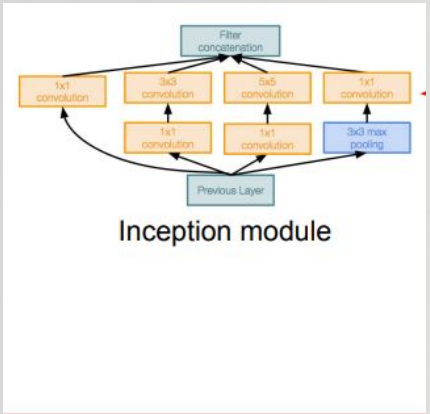


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

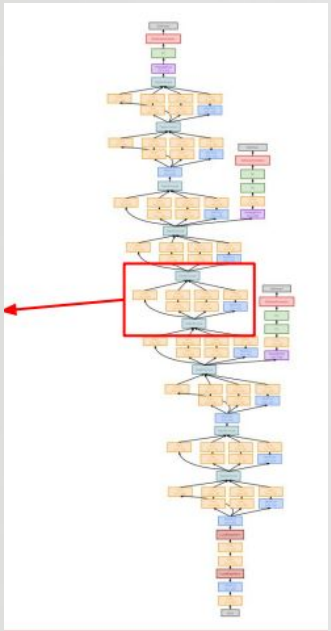
VGGNet


Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 256
3x3 conv, 384
Pool
3x3 conv, 384
Pool
5x5 conv, 256
11x11 conv, 96
Input
AlexNet
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG16
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG19

GoogleNet



Inception module



Deep Learning Hardware

My Computer

CPU

GPU

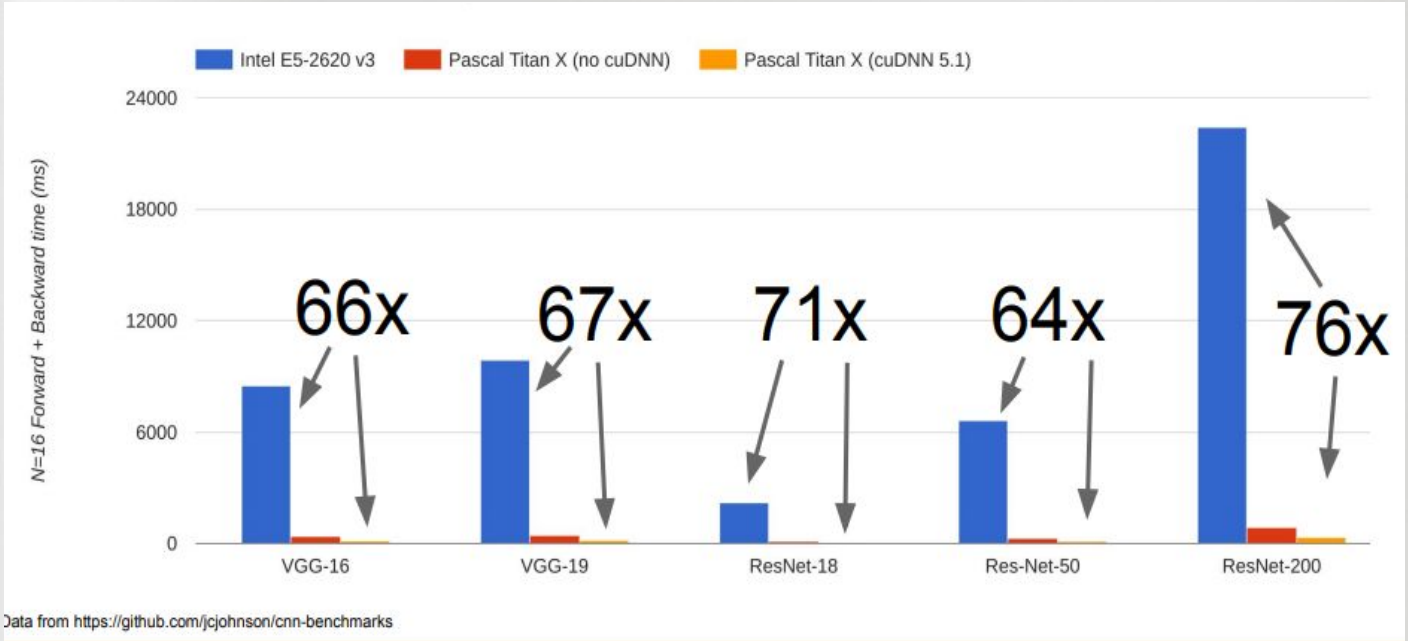
CPU vs GPU

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$339	~540 GFLOPs FP32
GPU (NVIDIA GTX 1080 Ti)	3584	1.6 GHz	11 GB GDDR5 X	$699	~11.4 TFLOPs FP32

CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

CPU vs GPU


Intel E5-2620 v3
Pascal Titan X (no cuDNN)
Pascal Titan X (cuDNN 5.1)
N=16 Forward + Backward time (ms)
24000
18000
12000
6000
0
66x
67x
71x
64x
76x
VGG-16
VGG-19
ResNet-18
Res-Net-50
ResNet-200
Data from https://github.com/jcjohnson/cnn-benchmarks

CPU vs GPU

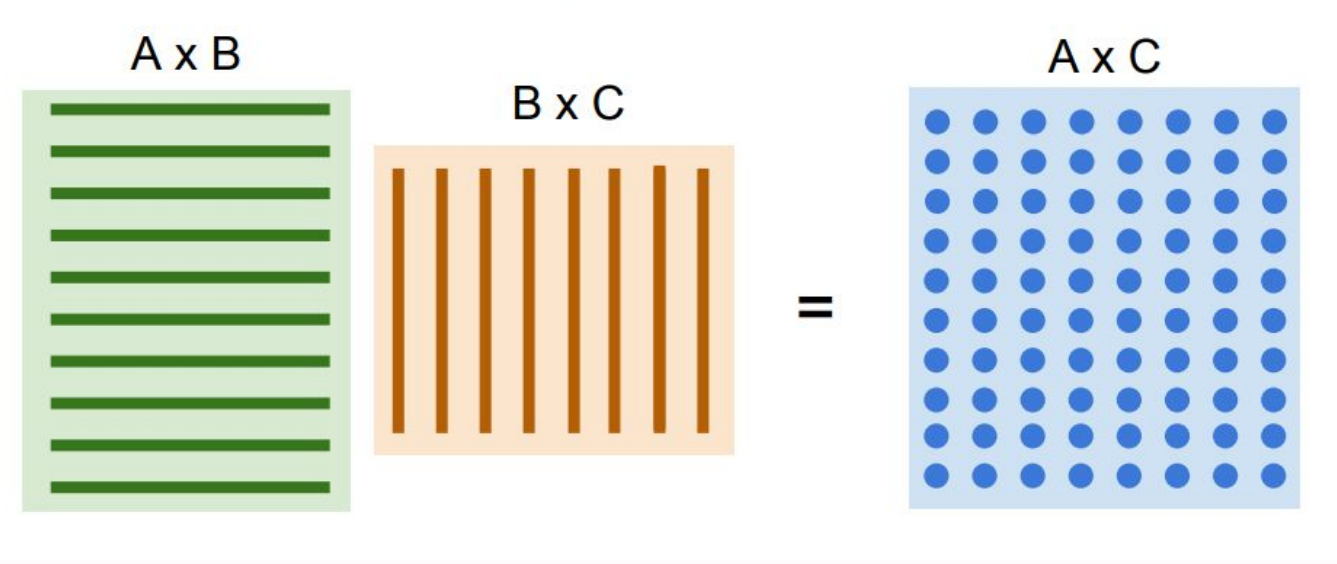
	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$339	~540 GFLOPs FP32
GPU (NVIDIA GTX 1080 Ti)	3584	1.6 GHz	11 GB GDDR5 X	$699	~11.4 TFLOPs FP32
TPU NVIDIA TITAN V	5120 CUDA, 640 Tensor	1.5 GHz	12GB HBM2	$2999	~14 TFLOPs FP32 ~112 TFLOP FP16
TPU Google Cloud TPU	?	?	64 GB HBM	$6.50 per hour	~180 TFLOP

CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

TPU: Specialized hardware for deep learning

Example: Matrix Multiplication



THANKS!

Any questions?