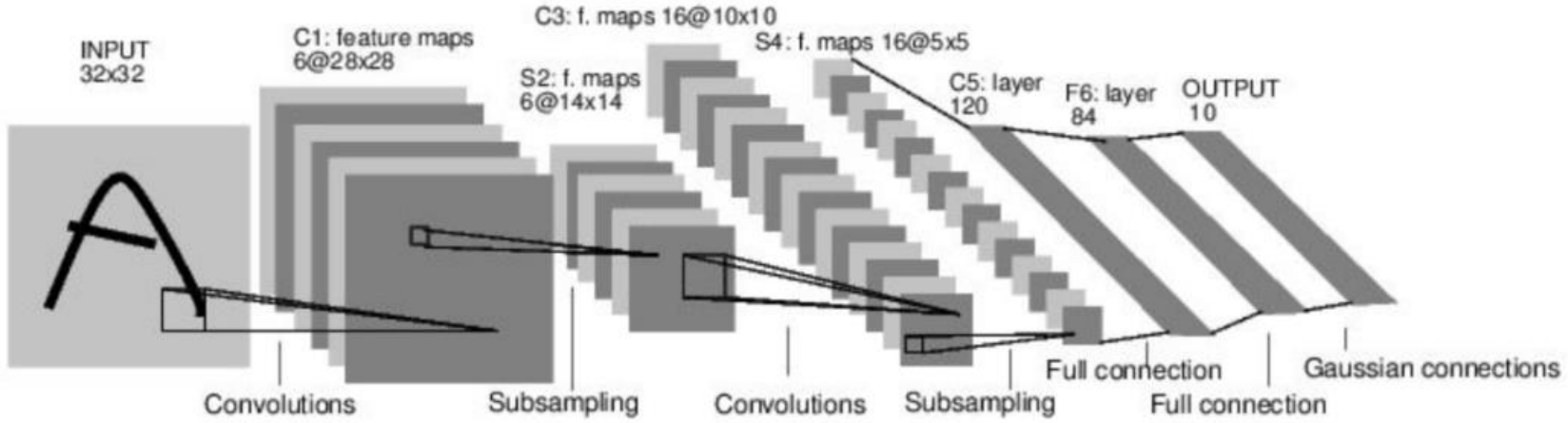
Deep Convolutional Neural Networks



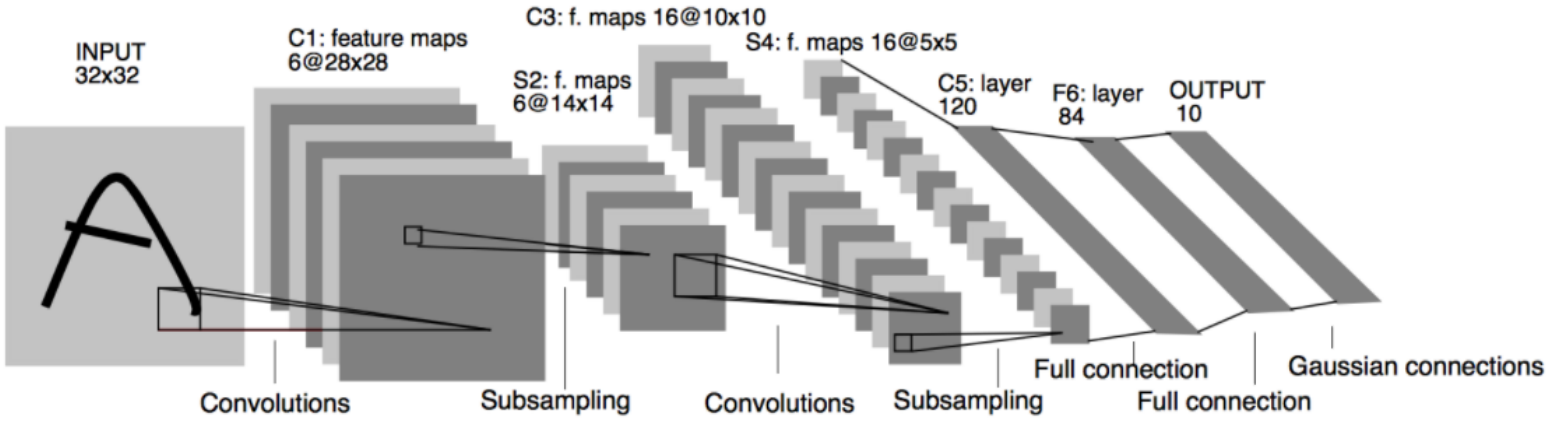
[LeNet-5, LeCun 1980]

Slides from Geoffrey Hinton, Alyosha Efros, Andrej Karpathy

CSC411/2515: Machine Learning and Data Mining, Winter 2018

Michael Guerzhoy and Lisa Zhang

LeNet: a deep neural network

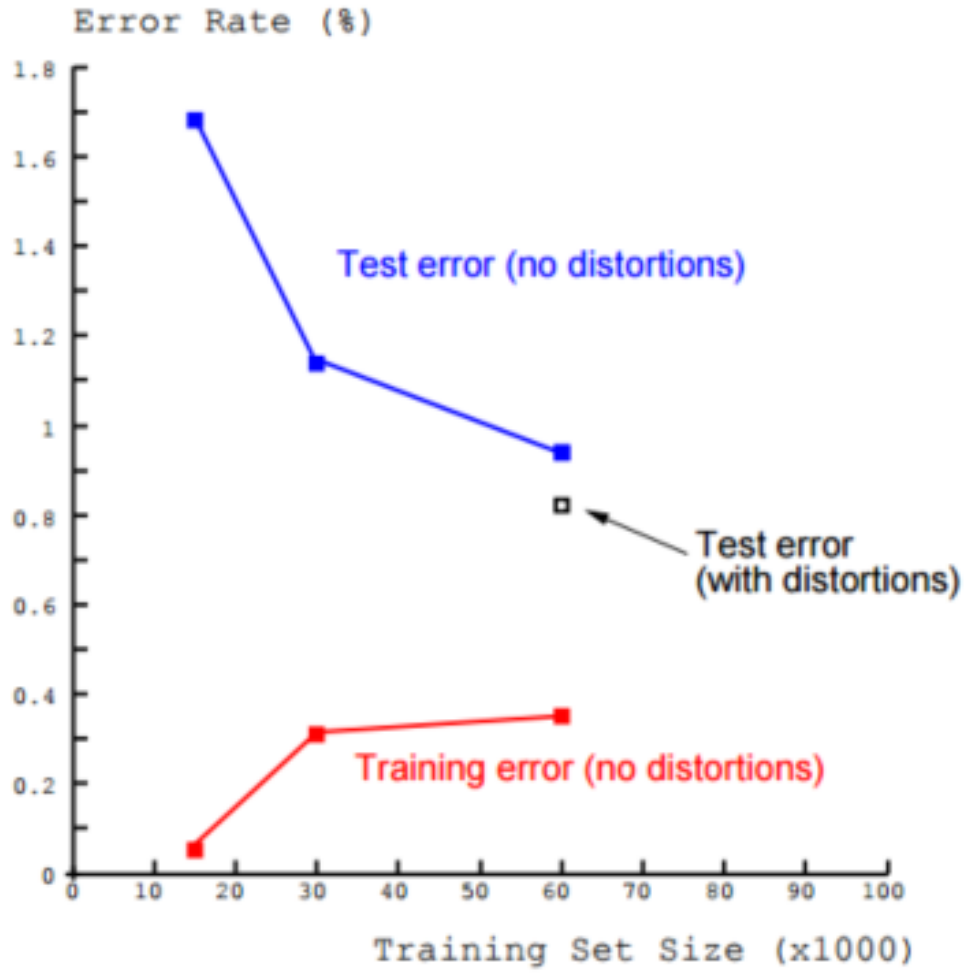


- Built in 80s by Yann LeCun (at one time a post-doc at UofT) at AT&T
- Recognized handwritten characters
- Main ideas:
  - Deep network (many layers)
  - Convolutional layers
  - Subsampling/pooling layers

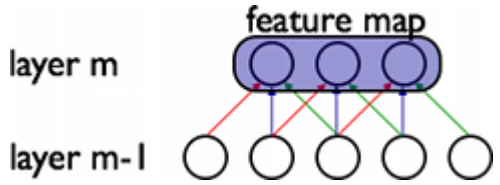
Yann LeCun

LeNet-5 errors

LeNet Performance

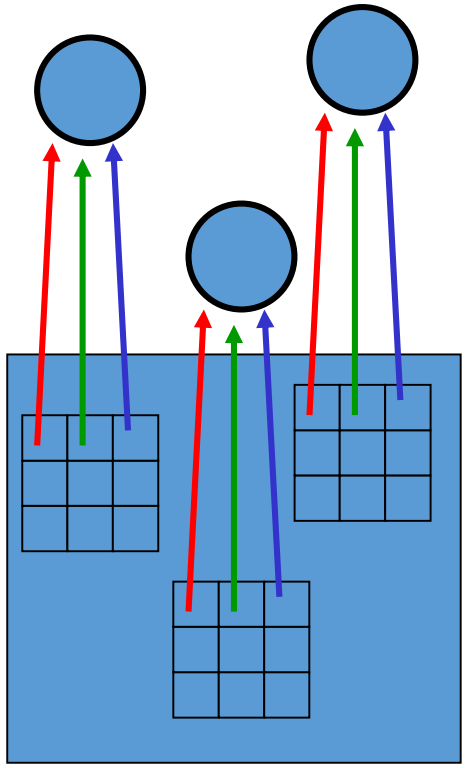


Computing Features



- Idea: each neuron on the higher layer is detecting the same feature, but in different locations on the lower layer
  - Detecting=the output is high if the feature is present
- It's the same feature because the weights are the same
- Note: each neuron is only connected with non-zero weights to a small area in the input

The red connections all have the same weight.



Feature Detection

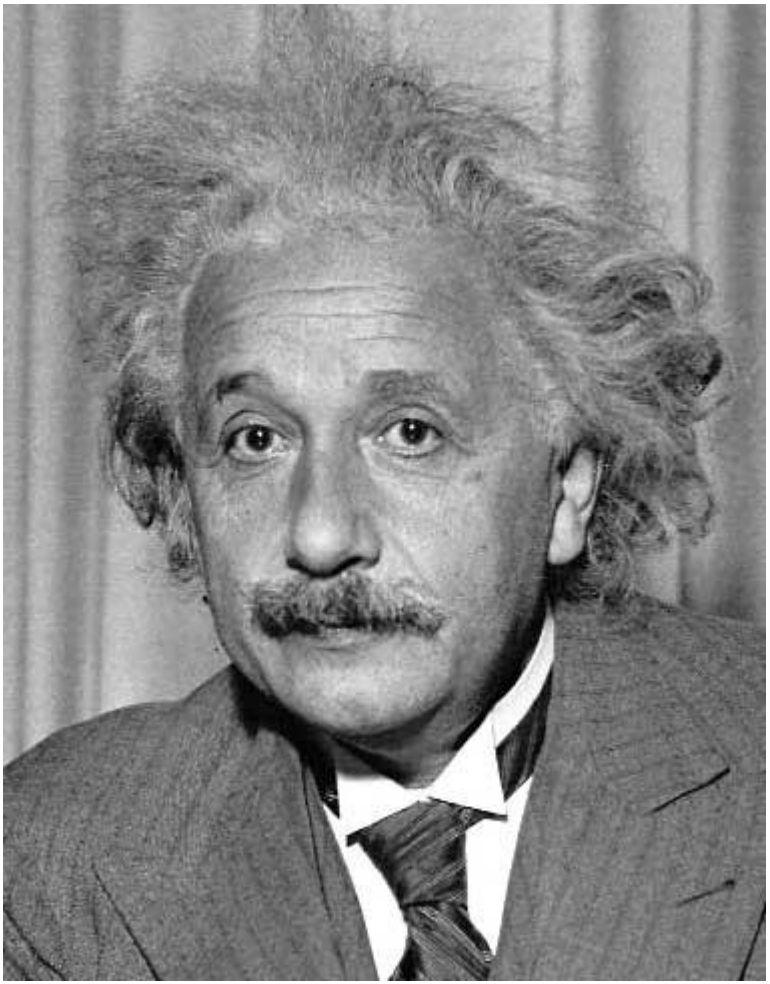
- The weights of each unit in the upper layer can be represented as a 2D array
- To compute the input to each neuron in the upper layer, we are computing the dot product between the 2D array (called *kernel*) and the area of the lower layer to which the neuron is connected (called the *receptive field*)

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

3x3 weights array for a 3x3 area in the input

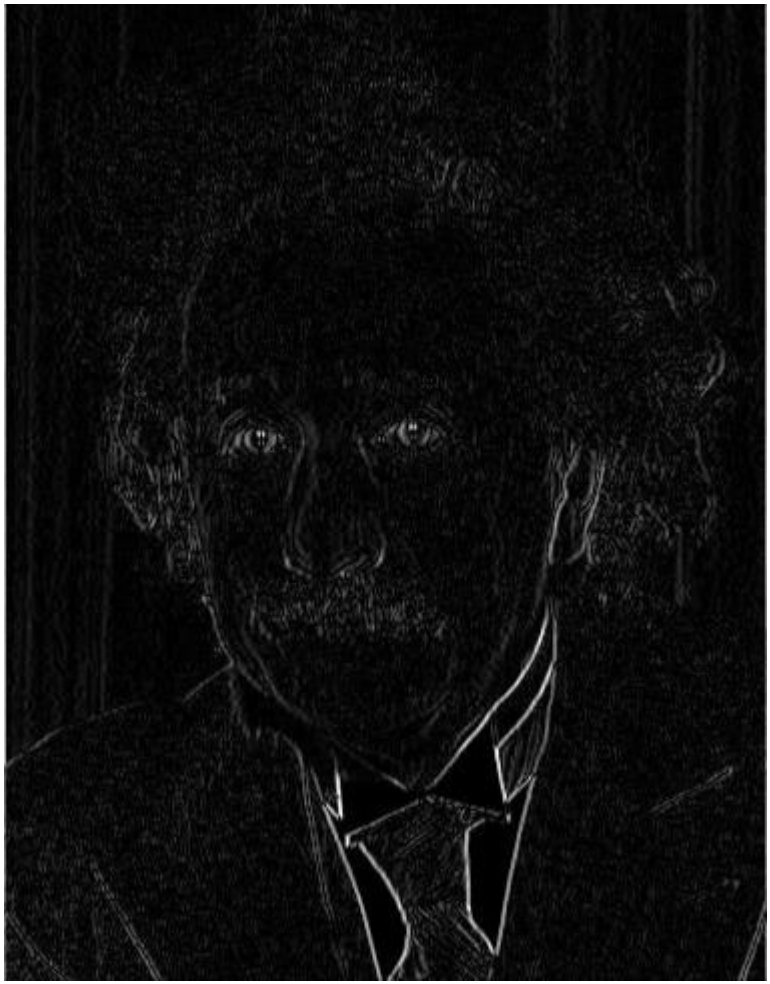
- The operation of computing the feature layer from the lower layer is called *convolution* (technically, "cross-correlation," but the differences between convolution and cross-correlation is unimportant here.)

Convolution Example: Sobel Filter



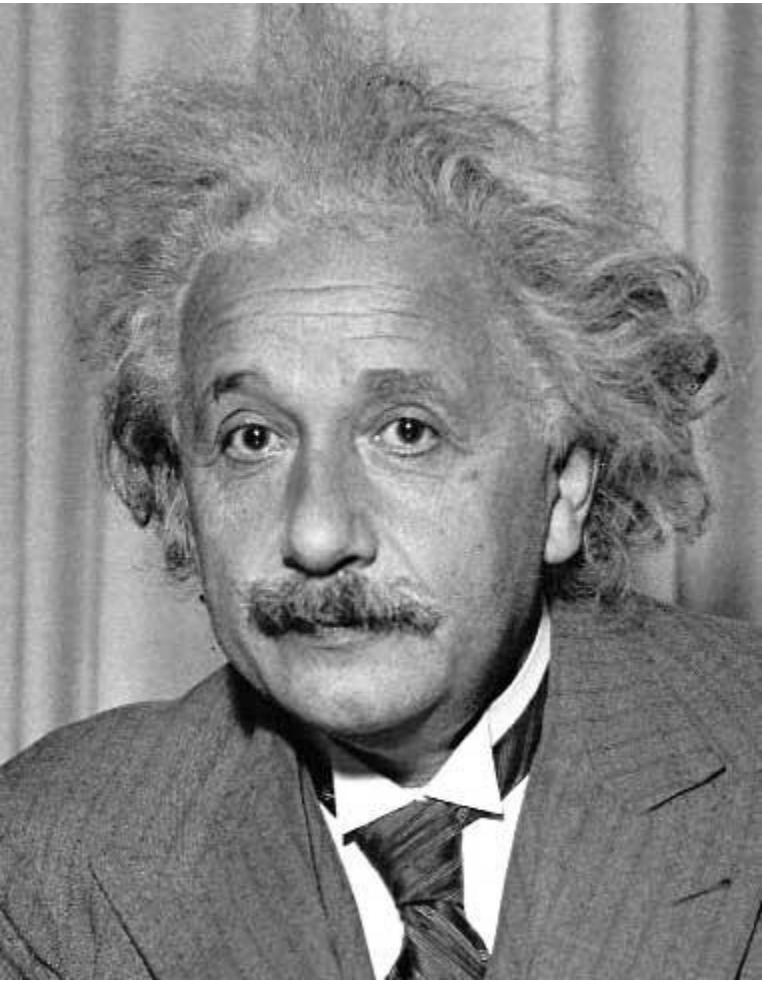
*

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



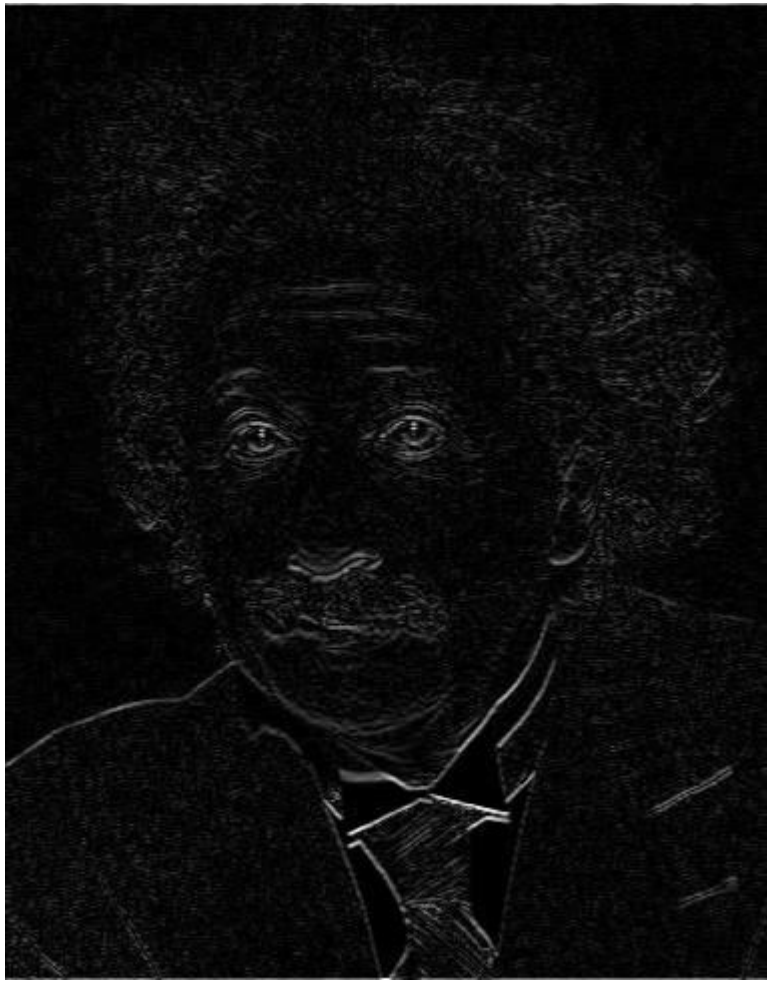
Vertical Edge
(absolute value)

Convolution Example: Sobel Filter



*

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

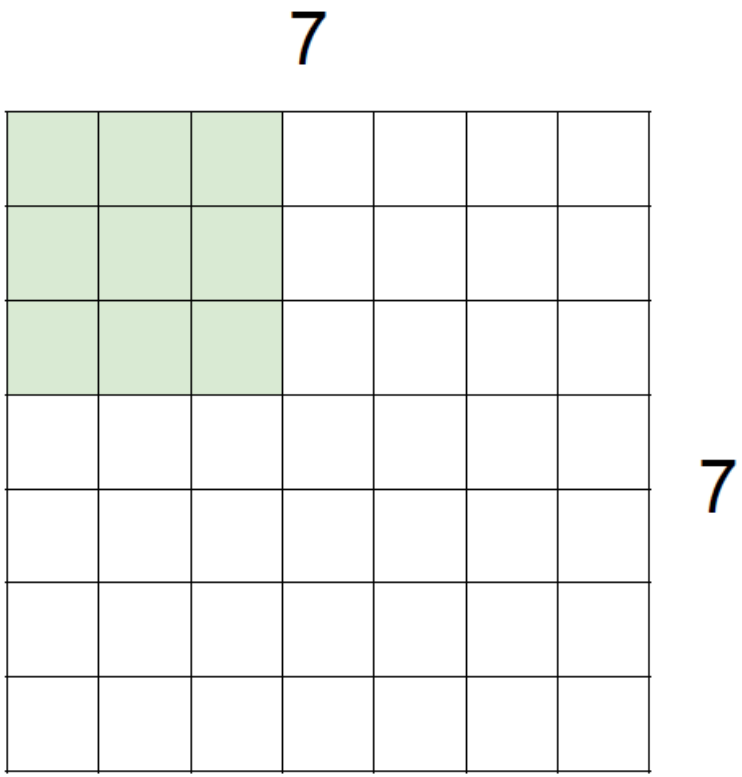


Horizontal Edge
(absolute value)

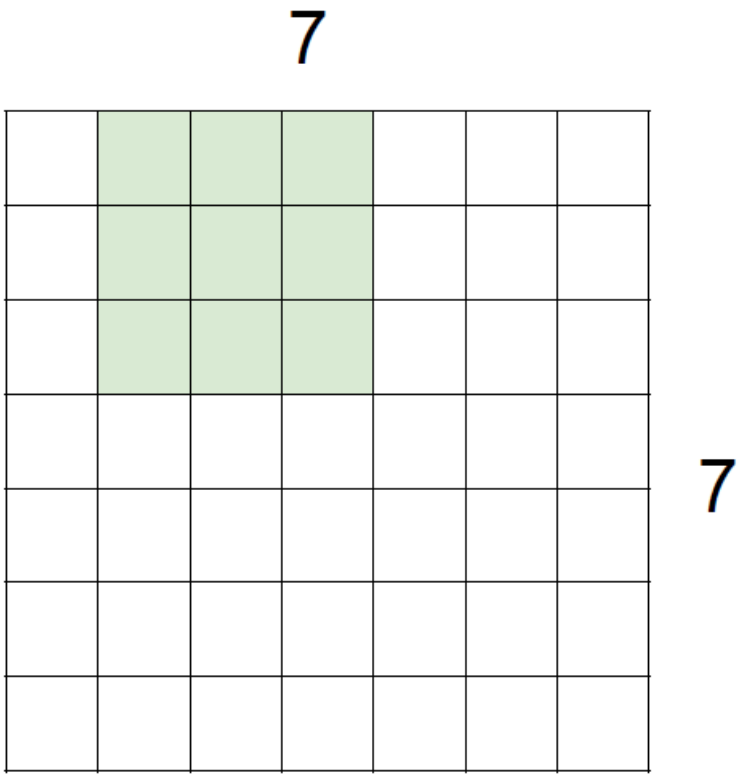
Convolution Example: Blob Detection

$*$

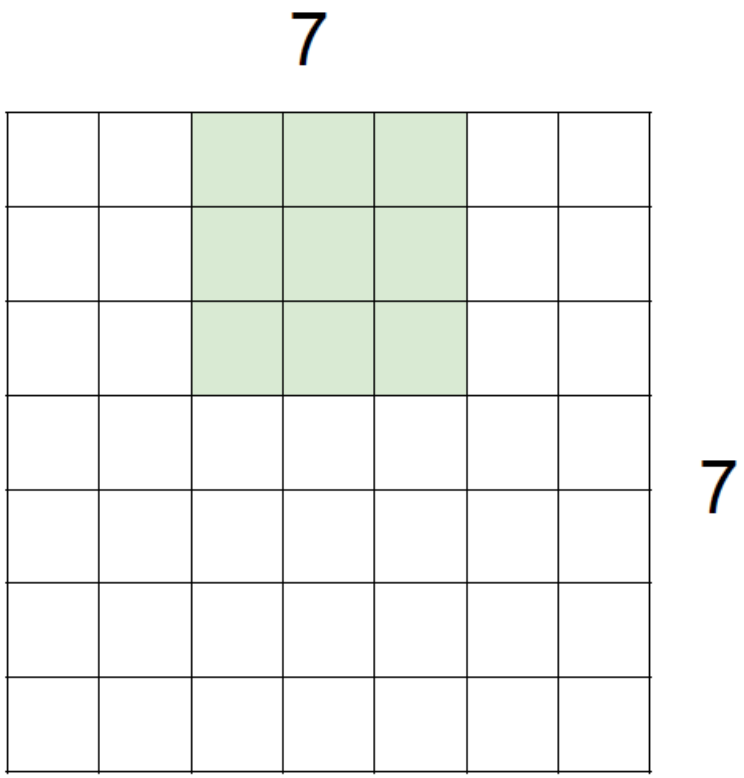
$$\begin{pmatrix} 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \end{pmatrix}$$



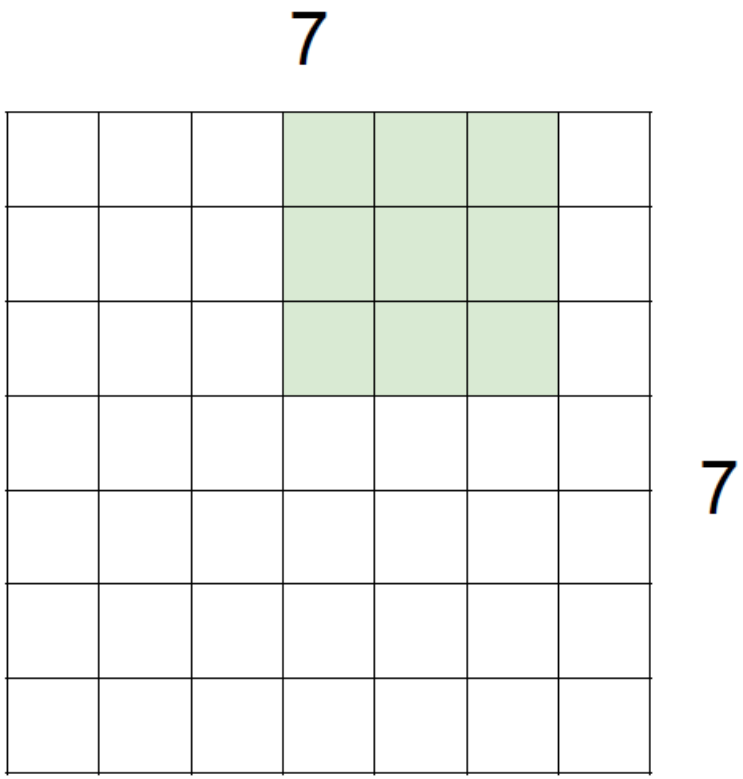
7x7 input (spatially)
assume 3x3 filter



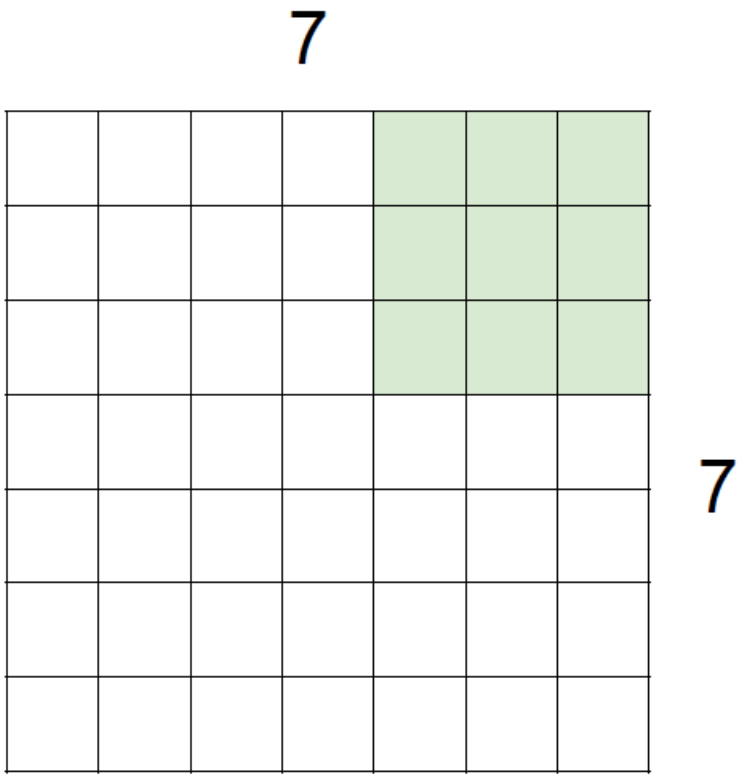
7x7 input (spatially)
assume 3x3 filter



7x7 input (spatially)
assume 3x3 filter

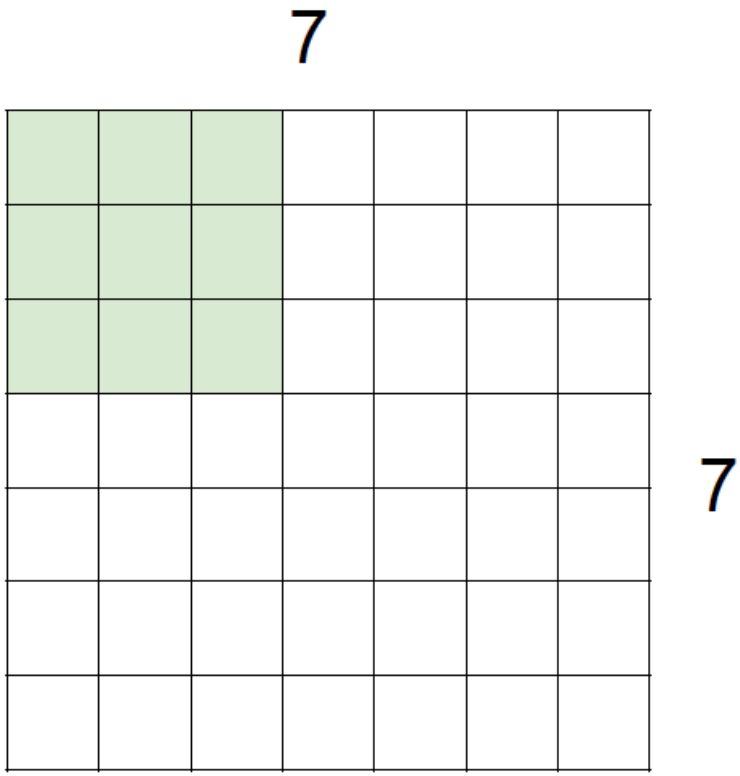


7x7 input (spatially)
assume 3x3 filter

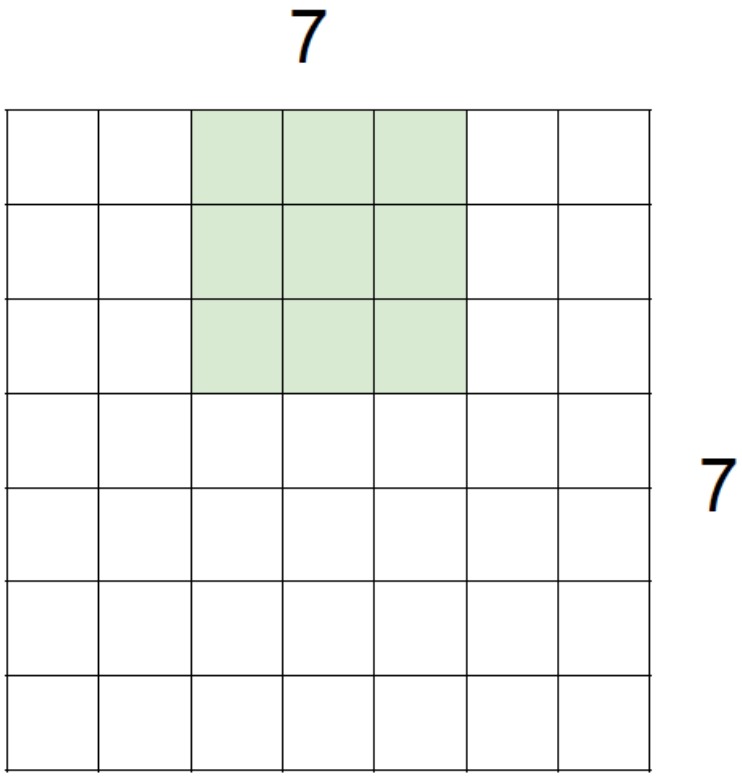


7x7 input (spatially)
assume 3x3 filter

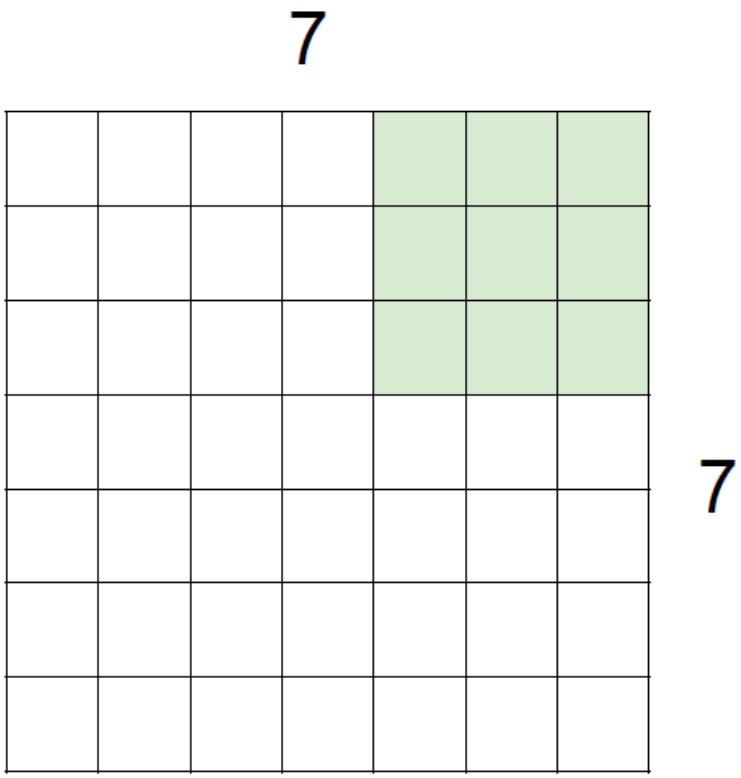
=> 5x5 output



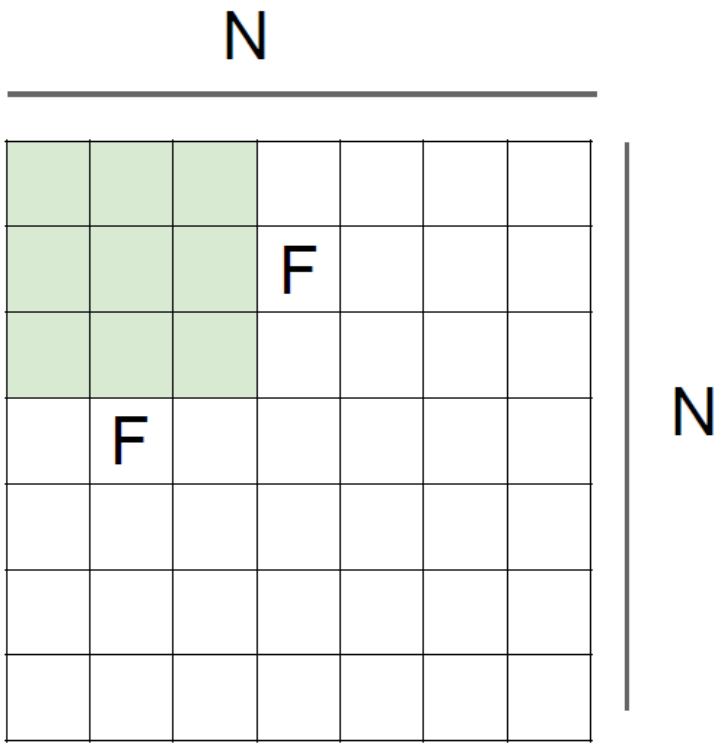
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



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



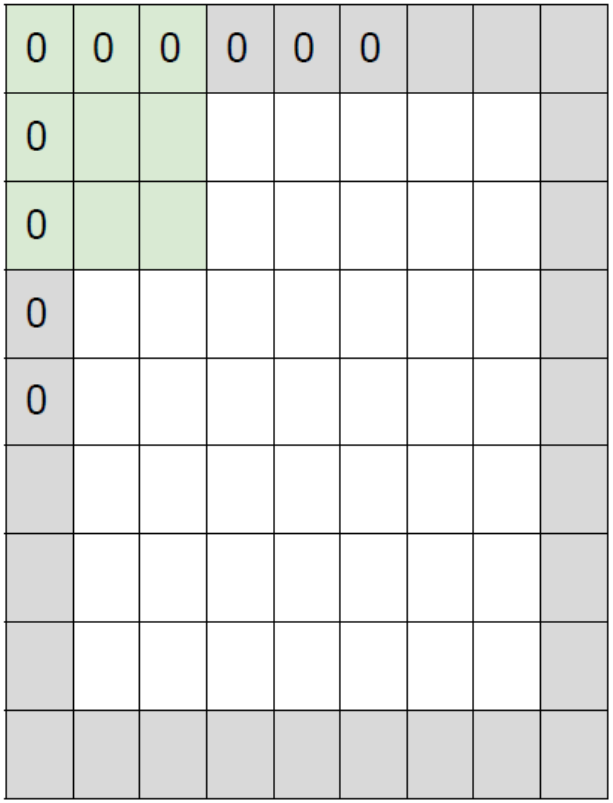
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!



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 :\

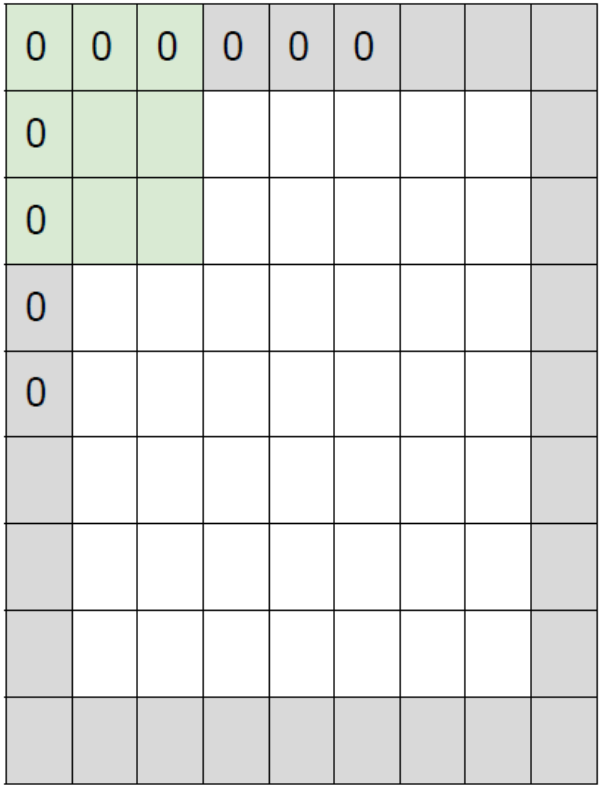
In practice: Common to zero pad the border



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

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

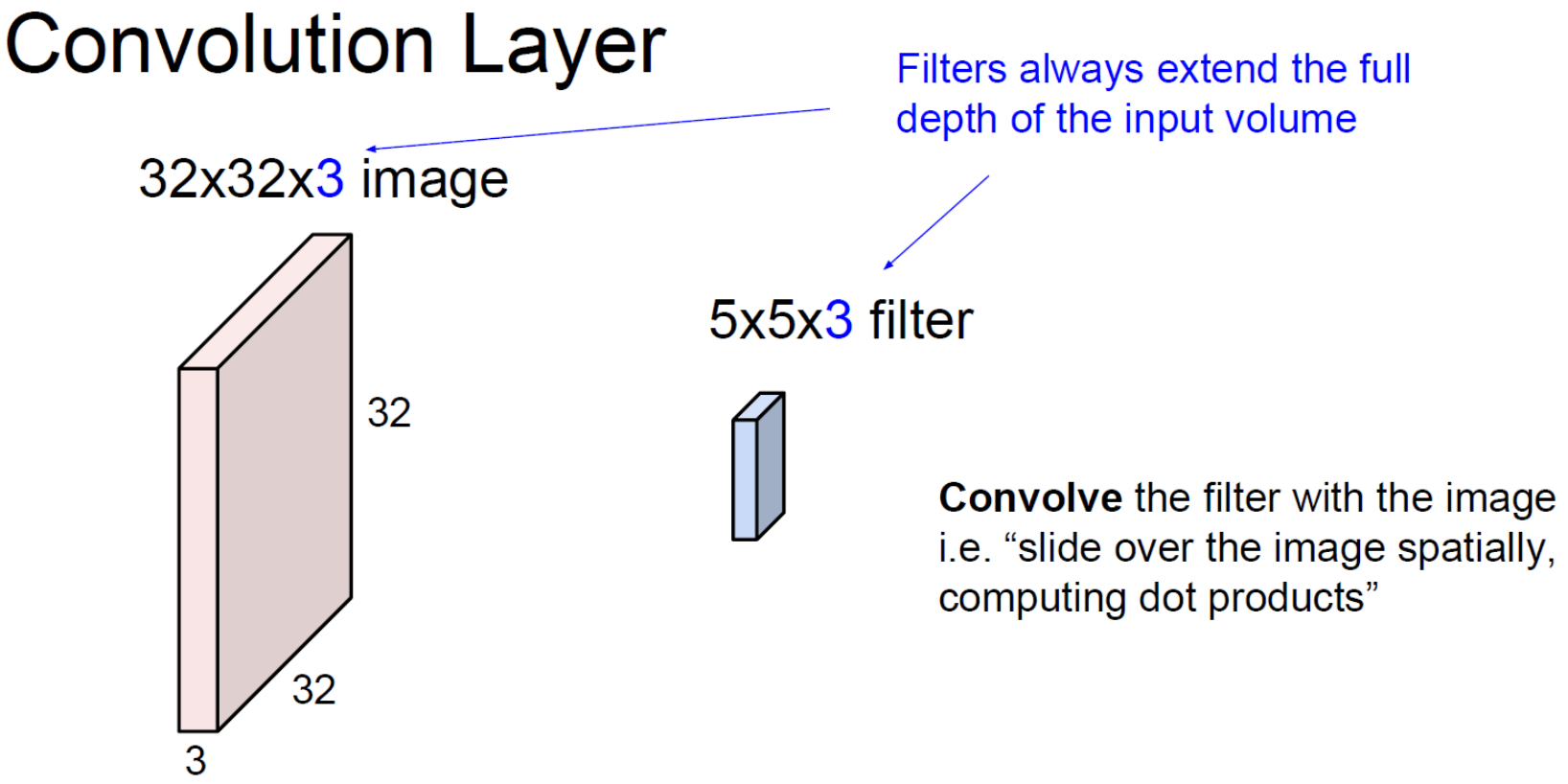
In practice: Common to zero pad the border



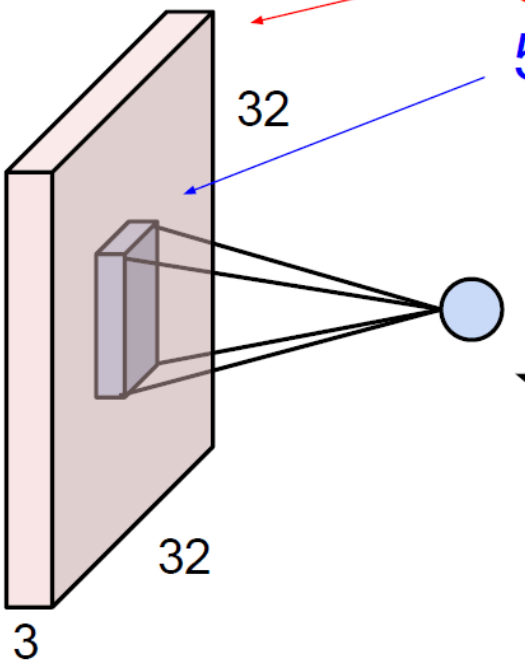
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in 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

Convolutional layers: RGB images

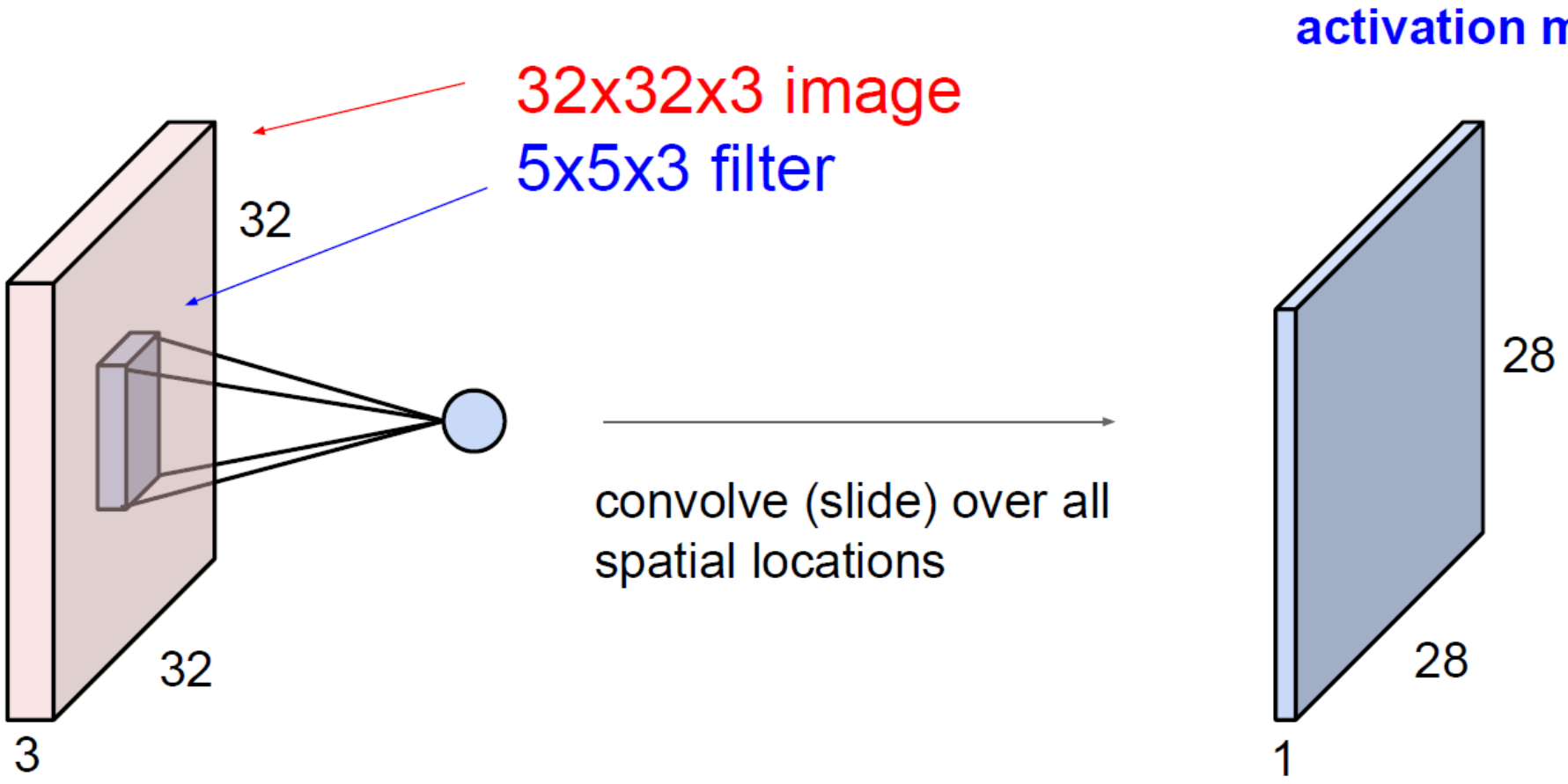


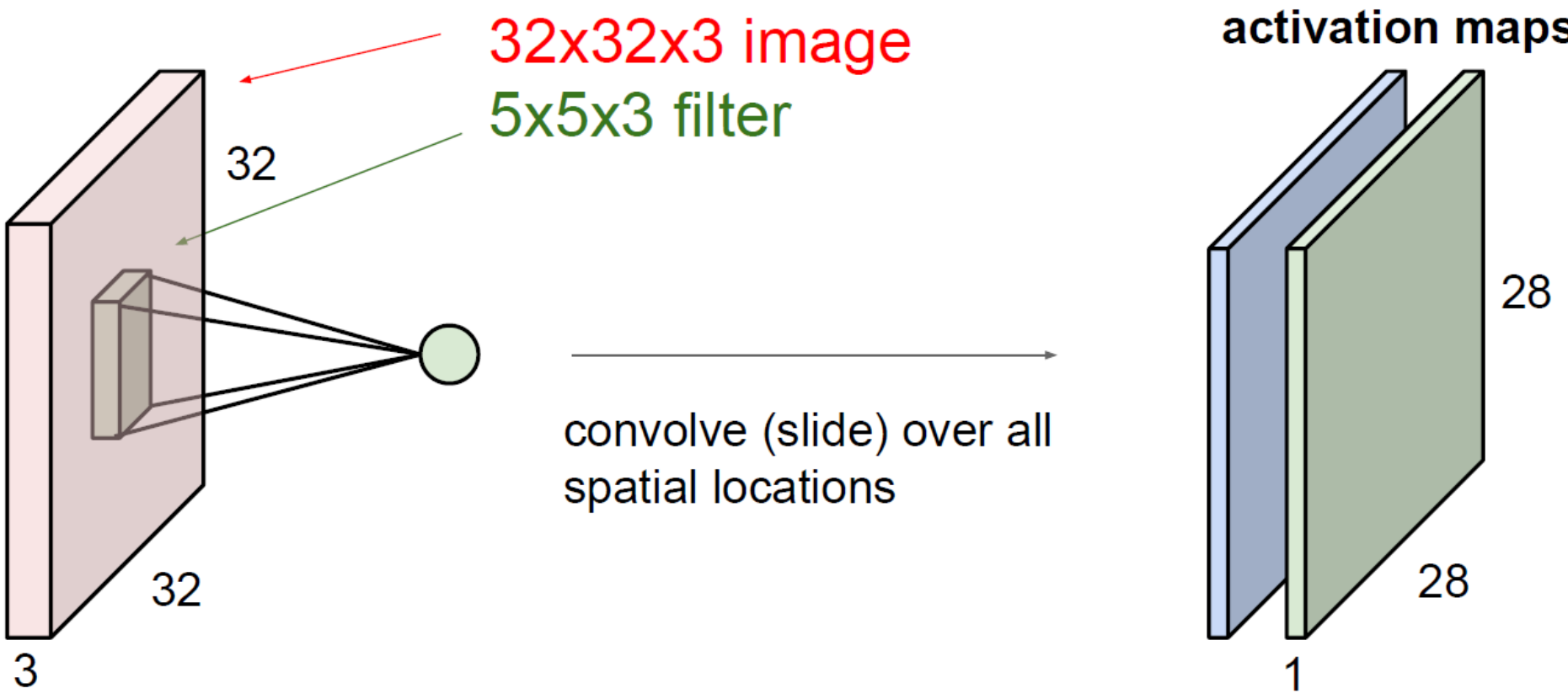
Convolution Layer

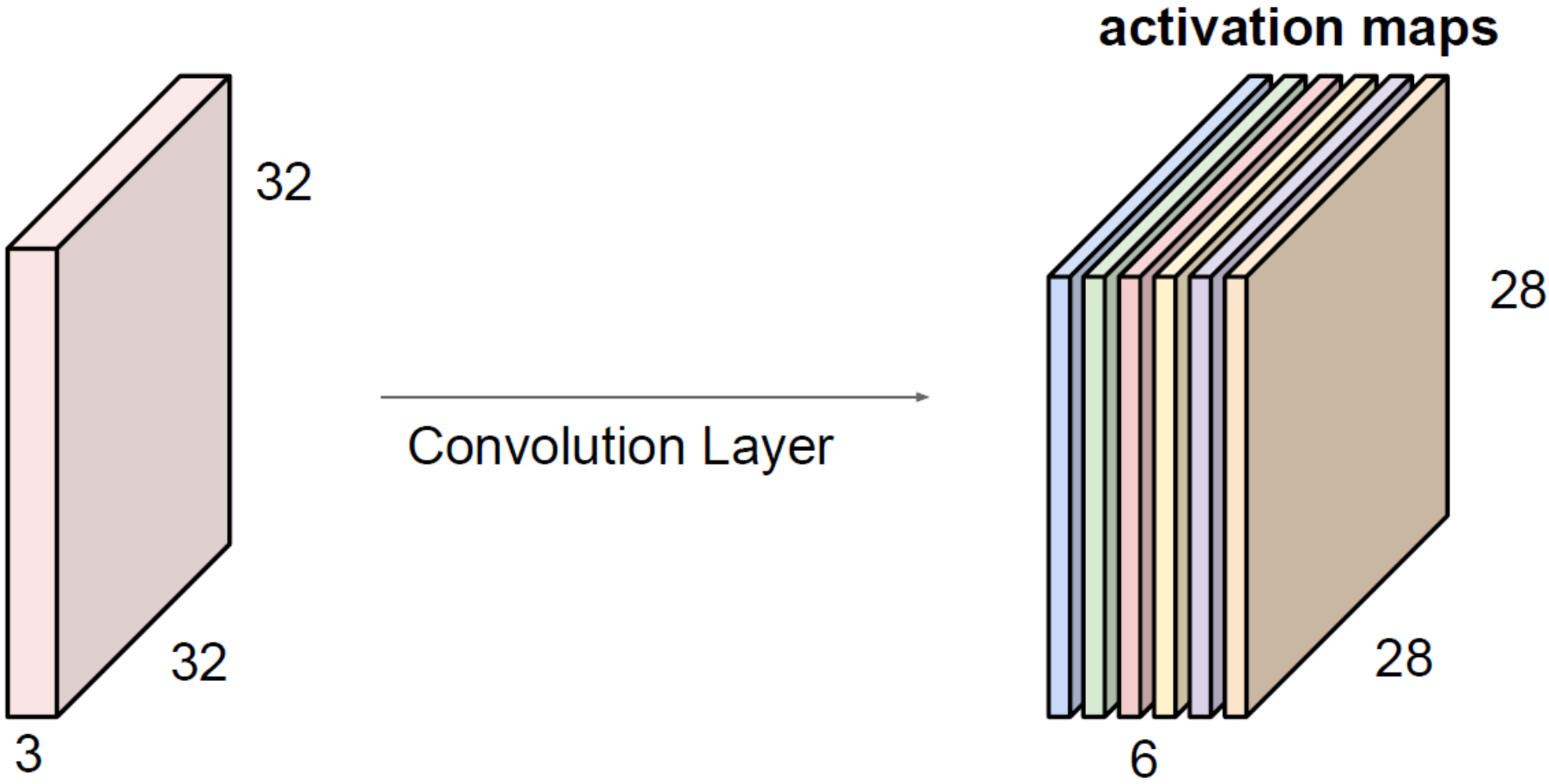


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

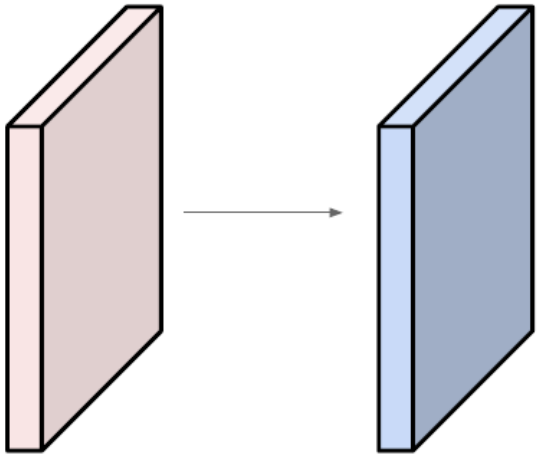

32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation map
28
28
1


32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation maps
28
28
1



We stack these up to get a “new image” of size 28x28x6!

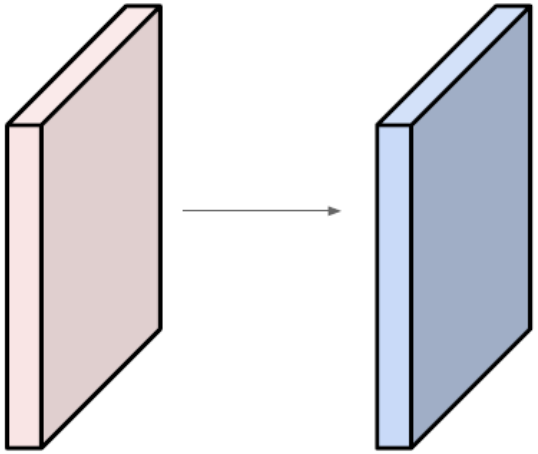
Examples time:



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**

Convolutional Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Convolutional Layers Summary Again

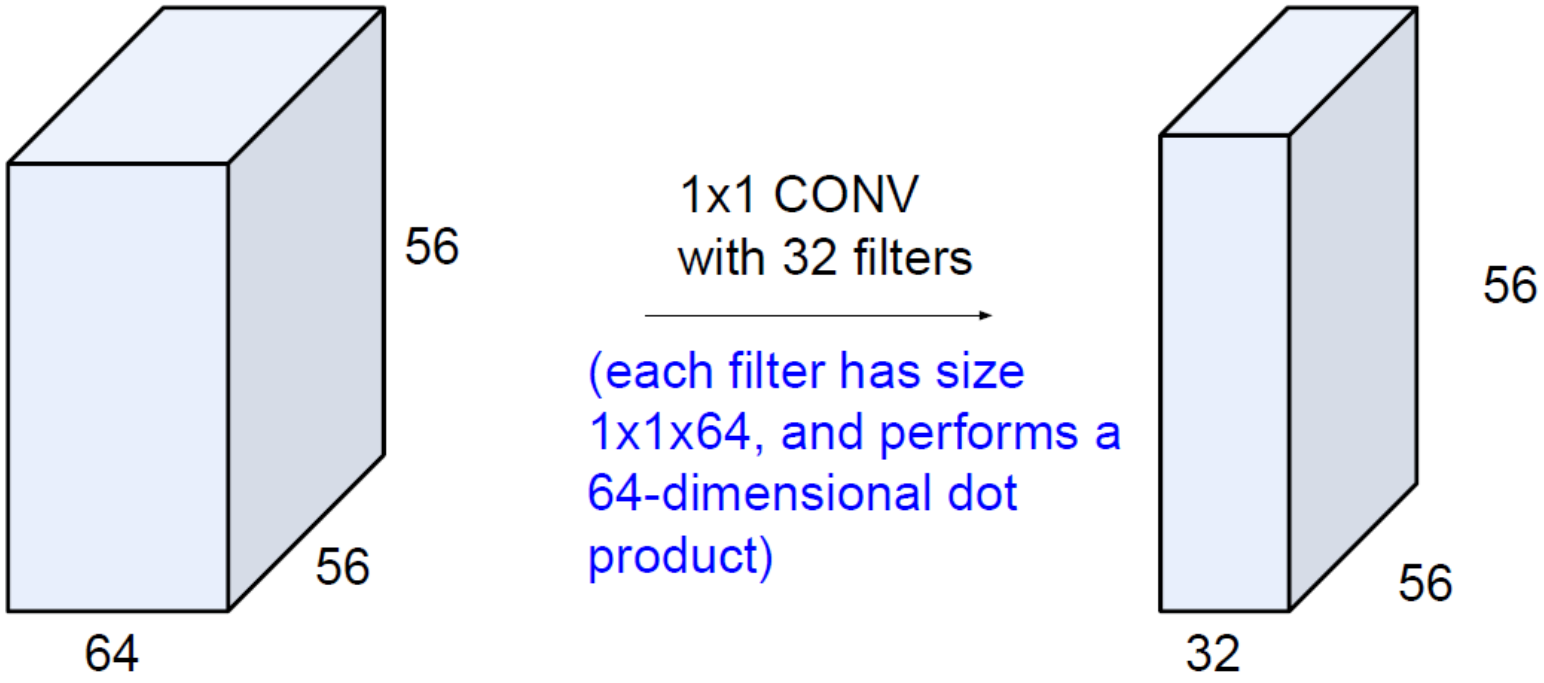
Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Summary. To summarize, the Conv Layer:

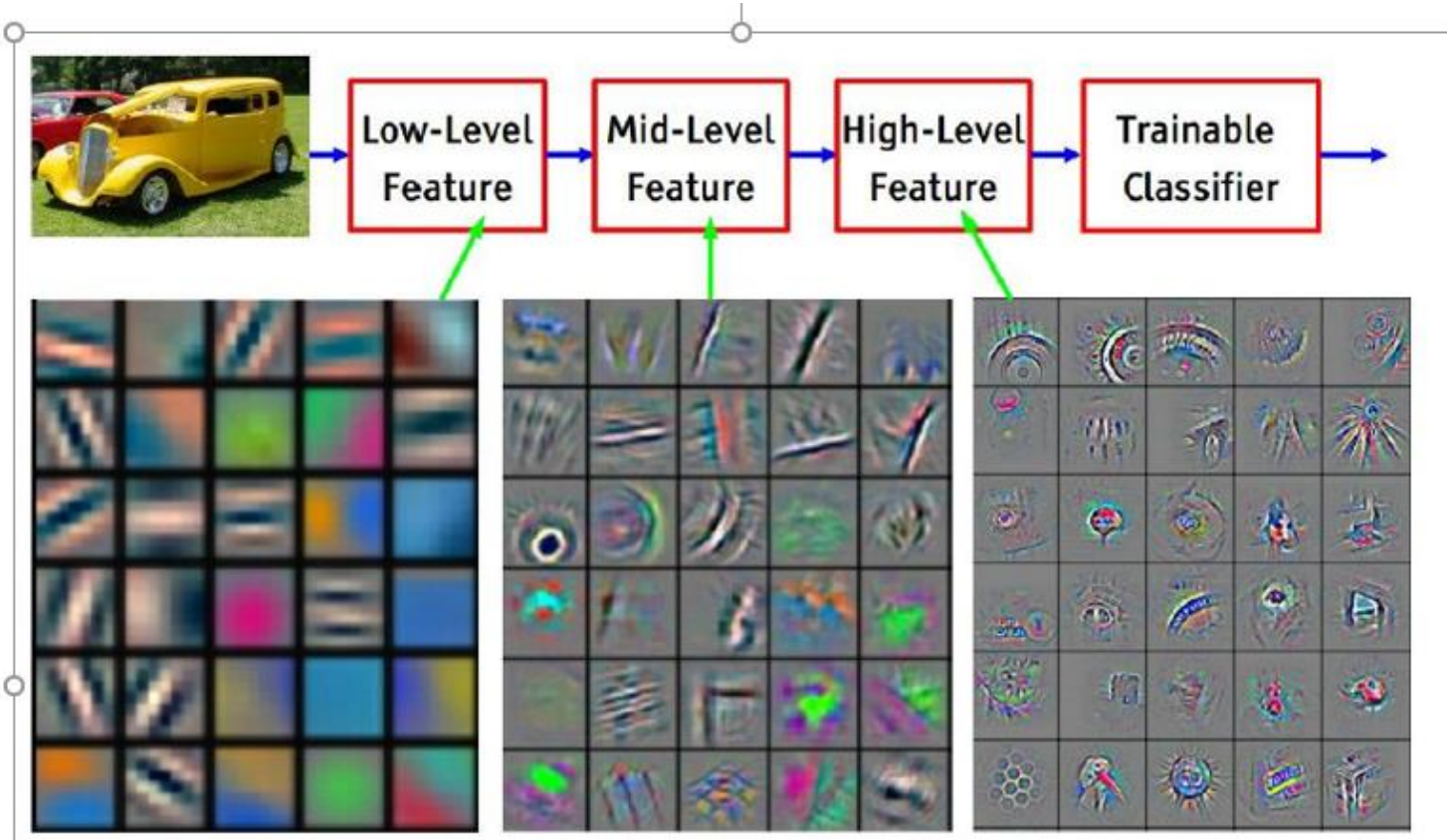
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

(1x1 convolutions?)



- Useful because they *summarize data.*
- Example: a 1x1 filter that computes the grayscale image from and RGB input

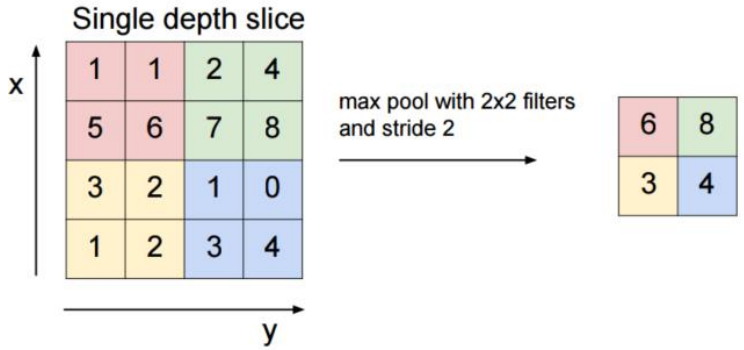
Preview: what sort of filters do convolutional layers learn?



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

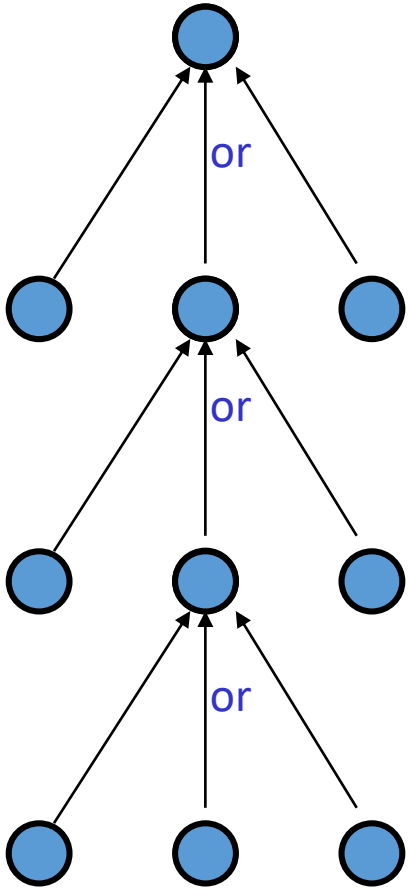
Pooling Features (“subsampling”)

- The job of complex cells in the visual cortex
- Max Pooling
  - Is there a diagonal edge somewhere in an area of the image?
  - Take the maximum over the responses to the feature detector in the area
- Average Pooling
  - Is there a blobs pattern in an area of the image?
  - Take the average over the responses to the feature detectors in the area
- Max Pooling generally works better



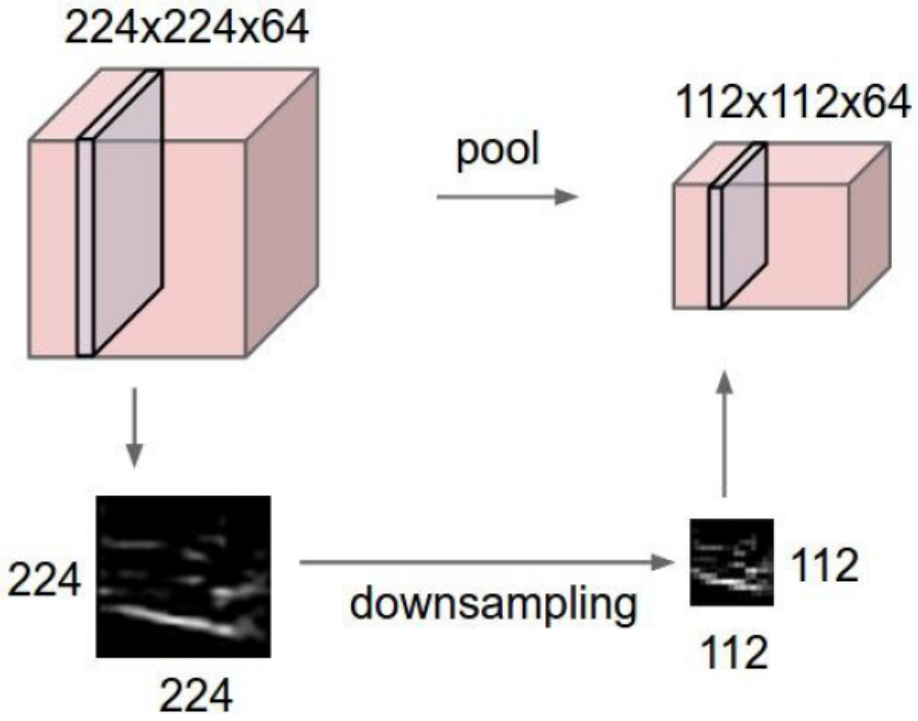
Max Pooling as Hierarchical Invariance

- At each level of the hierarchy, we use an “or” to get features that are invariant across a bigger range of transformations.
- (Average Pooling is a little bit like an “AND”)
- At the top of the network, we can have a neuron that lights up if a certain shape appears *anywhere* in the image



Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



Pooling Layer

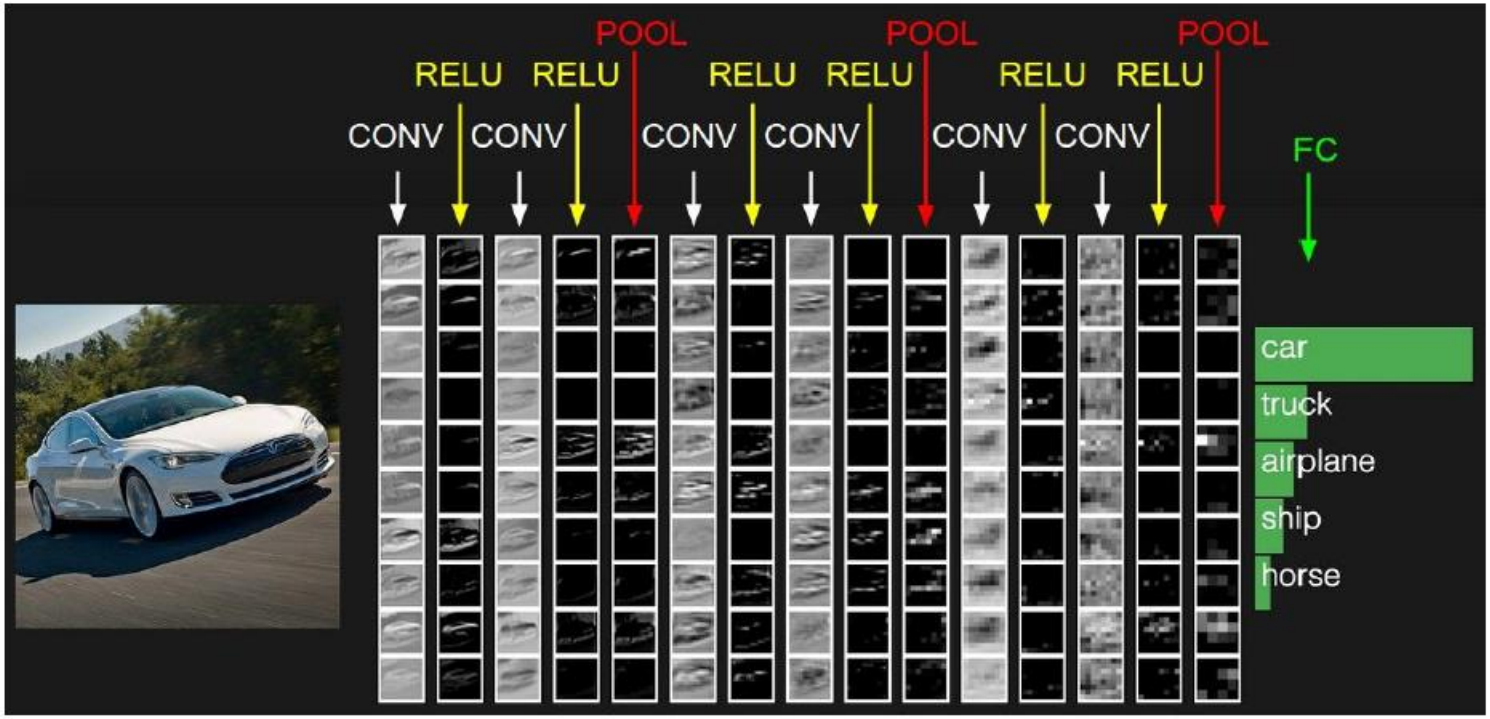
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

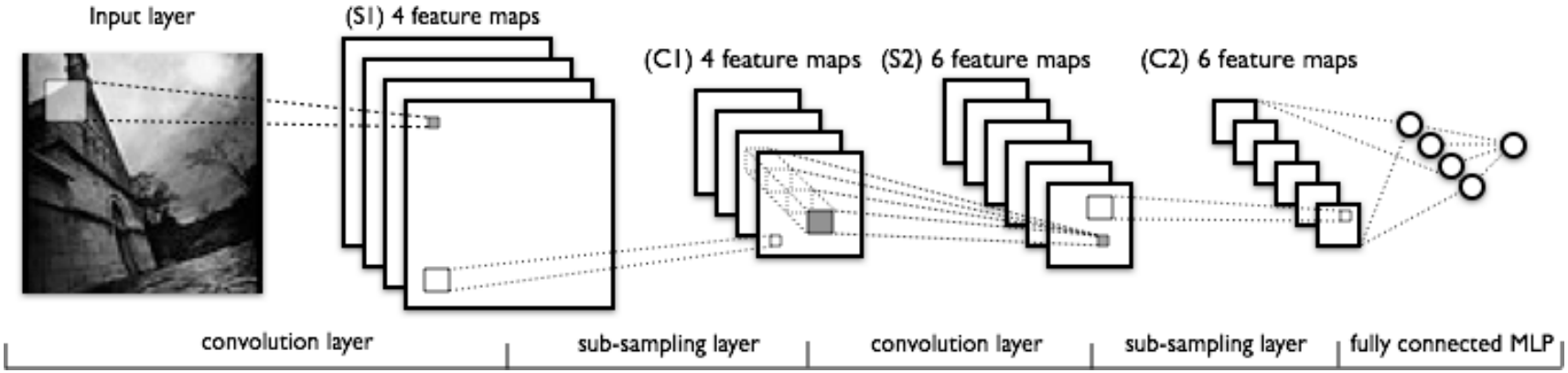
F = 2, S = 2
F = 3, S = 2

Fully-Connected Layer

- Contains neurons that connect to the entire lower year, as in ordinary neural networks



Putting it All Together



- Different types of layers: convolution and subsampling.
- Convolution layers compute features maps: the response to multiple feature detectors on a grid in the lower layer
- Subsampling layers pool the features from a lower layer into a smaller feature map

Why Convolutional Nets

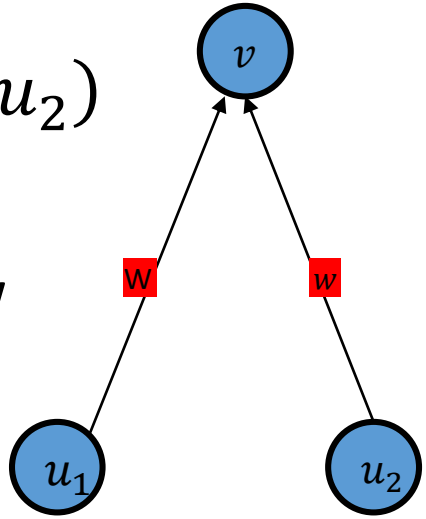
- It's possible to compute the same outputs in a fully connected neural network, but
  - The network is much harder to learn
  - There is more danger of overfitting if we try it with a really big network
    - A convolutional network has fewer parameters due to weight sharing*
- It makes sense to detect features and then combine them
  - That's what the brain seems to be doing

* Small fully connected networks can work very well, but are hard to train

Learning Convolutional Nets: Replicated Weights

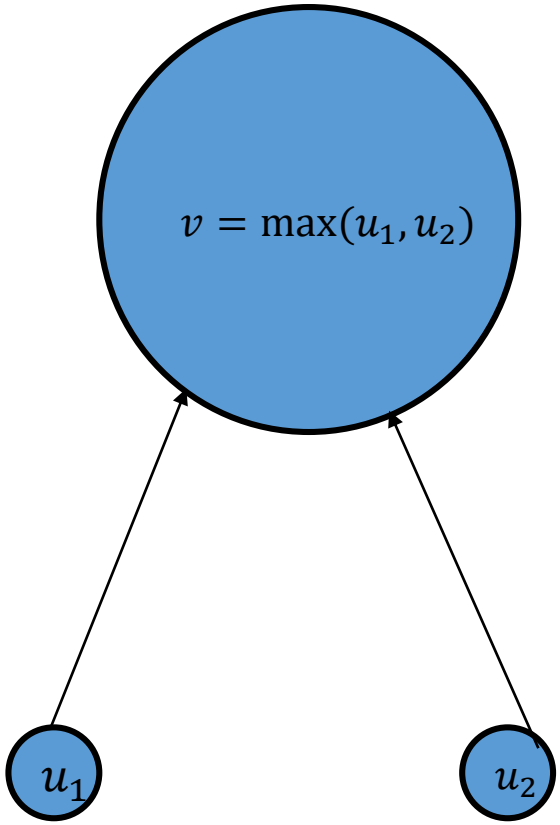
- $v = g(Wu_1 + Wu_2)$
- $\frac{\partial v}{\partial W} = (u_1 + u_2)g'(Wu_1 + W\,u_2)$
  $= u_1 g'(Wu_1 + Wu_2) + u_2 g'(Wu_1 + Wu_2)$

- Note: if u_1 is positive but u_2 is negative, W will be "pulled" in different directions by the two



Learning Convolutional Nets: Max Pooling

- $\frac{\partial v}{\partial u_i} = \begin{cases} 1, u_i > u_j, \forall j \neq i \\ 0, otherwise \end{cases}$
- The u's are real, so let's not worry about them being equal
- The gradient only flows to the unit that's responsible for the value of *v*
  - Makes sense! The other ones aren't likely detecting any patterns



IMAGENET Large Scale Visual Recognition Challenge

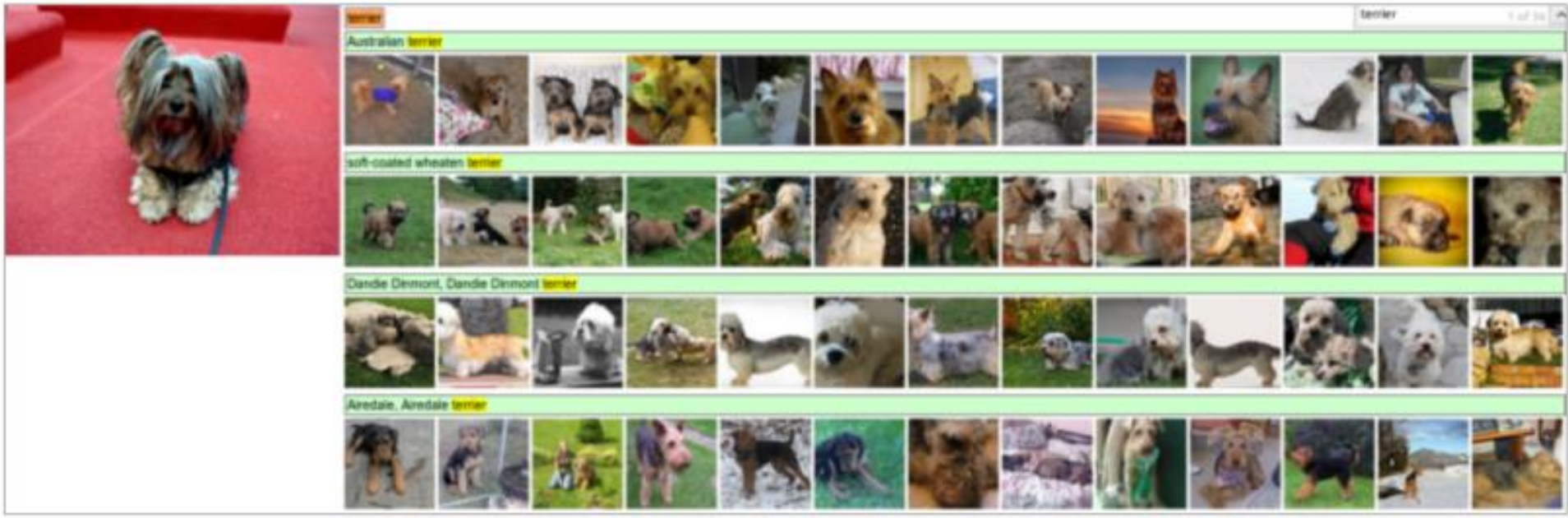
- About one million images, 1000 object categories in the training set
- Task: what is the object in the image
  - I.e., classify the image into one of 1000 categories
- Evaluation: is one of the best 5 guesses correct?


mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
mushroom
cherry
Madagascar cat
convertible
grille
pickup
beach wagon
fire engine
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
dalmatian
grape
elderberry
ffordshire bullterrier
currant
squirrel monkey
spider monkey
titi
indri
howler monkey

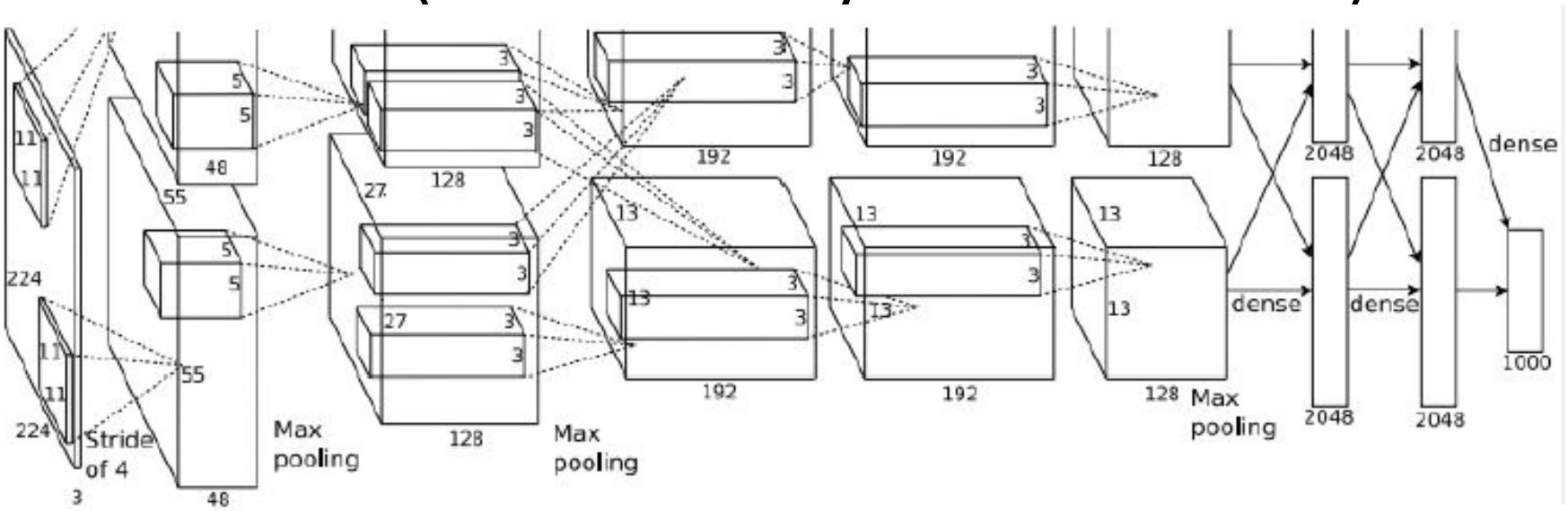
Human Performance on ImageNet

- http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/
- 5.1% error (i.e., none of the 5 guesses the person makes are correct)
- Try it yourself!
  - http://cs.stanford.edu/people/karpathy/ilsvrc/

Dogs Are Hard to Classify!



AlexNet (Krizhevsky et al. 2012)



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Aside: ConvNets vs. Monkeys

- Extract the features (neuron activities) from the Inferior Temporal Cortex of Rhesus Macaques when the monkeys are looking at images
- Extract features from the top layers of ConvNets when the ConvNets are looking at images
- Use both sets of features to classify images

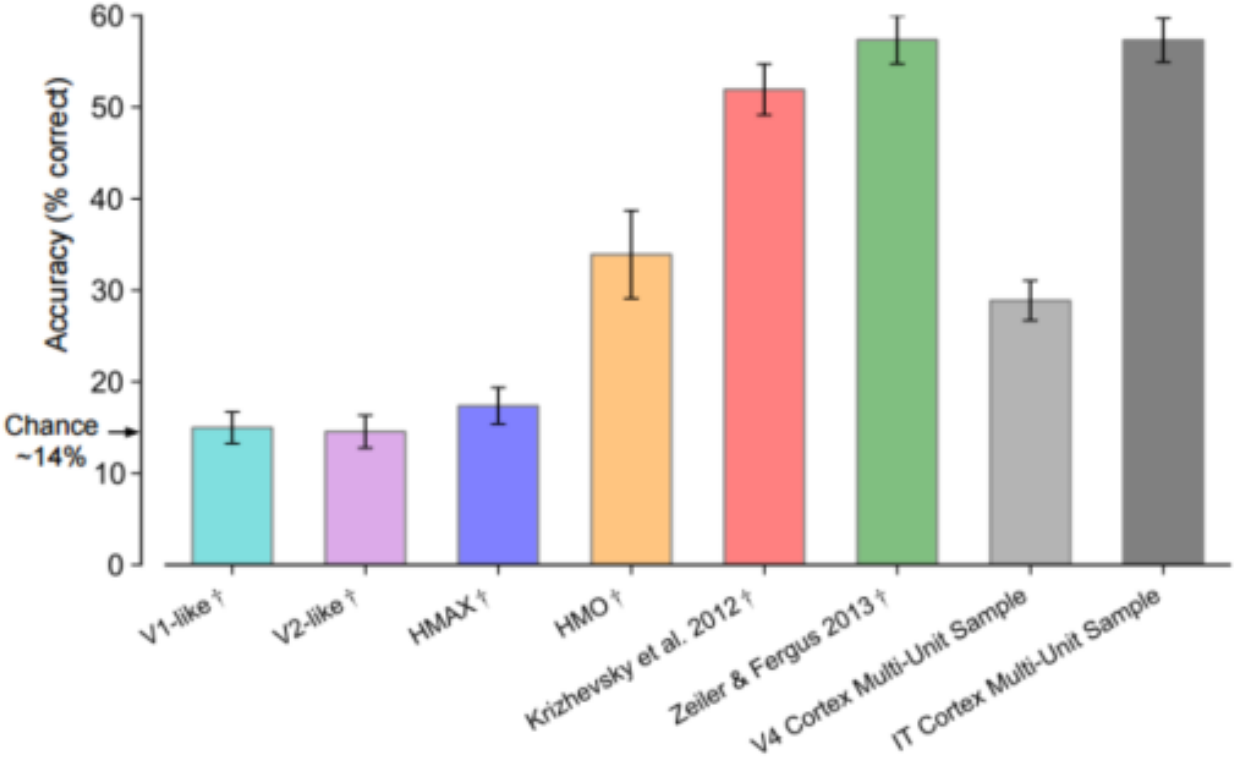
Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition

Charles F. Cadieu[1,*], Ha Hong[1,2], Daniel L. K. Yamins[1], Nicolas Pinto[1], Diego Ardila[1], Ethan A. Solomon[1], Najib J. Majaj[1], James J. DiCarlo[1]

1 Department of Brain and Cognitive Sciences and McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA 02139

2 Harvard–MIT Division of Health Sciences and Technology, Institute for Medical Engineering and Science, Massachusetts Institute of Technology, Cambridge, MA 02139

∗ E-mail: Corresponding cadieu@mit.edu



A Brute Force Approach

- Convolutional Networks architectures use knowledge about invariances to design the network architecture/weight constraints
- But it's much simpler to incorporate knowledge of invariances by just creating extra training data:
  - for each training image, produce new training data by applying all of the transformations we want to be insensitive to (Le Net can benefit from this too)
  - Then train a large, dumb net on a fast computer.
  - This works surprisingly well if the transformations are not too big

Deep Neural Networks as a Model of Computation

- Most people's first instinct when they think about building an image
- Classifier r is to write a complicated computer program
- A deep neural network *is* a computer program:

```
h1 = f1(x)
h2 = f2(h1)
h3 = f3(h2)
…
h9 = f9(h8)
```

- Can think of every layer of a neural network as one step of a parallel computation
- Features/templates are the functions that are applied to the previous layers
- Learning features ⇔ Learning what function to apply at step t of the algorithm

Deep Learning

Differentiable Programming

https://www.facebook.com/convolutionalmemes/photos/a.287833355004505.1073741828.287764591678048/424941947960311/?type=3&theater