hochschule mannheim

EMB-Lab, Fak. N

Convolutional Neural Networks

DLM – Deep Learning Methoden

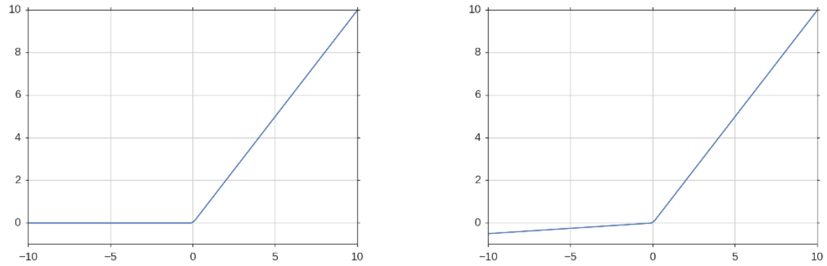
Piotr Bialecki

Carla Gil, Benjamin Kraus, Prof. Dr. Marcus Vetter

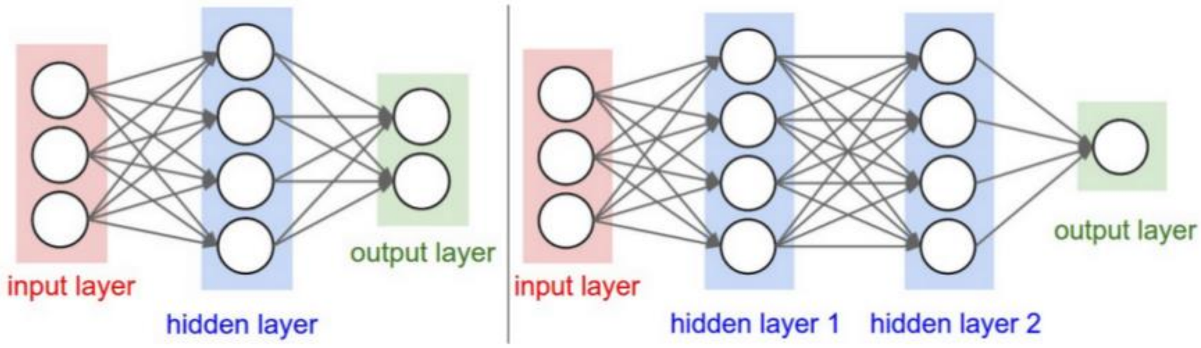
Mannheim, 27.11.2017

Recap

- (Deep) Neural Networks
  - Layer connectivity
  - Nonlinearities
  - Vanishing / Exploding Gradient
- Hyperparamters
  - Learning rate
  - ...



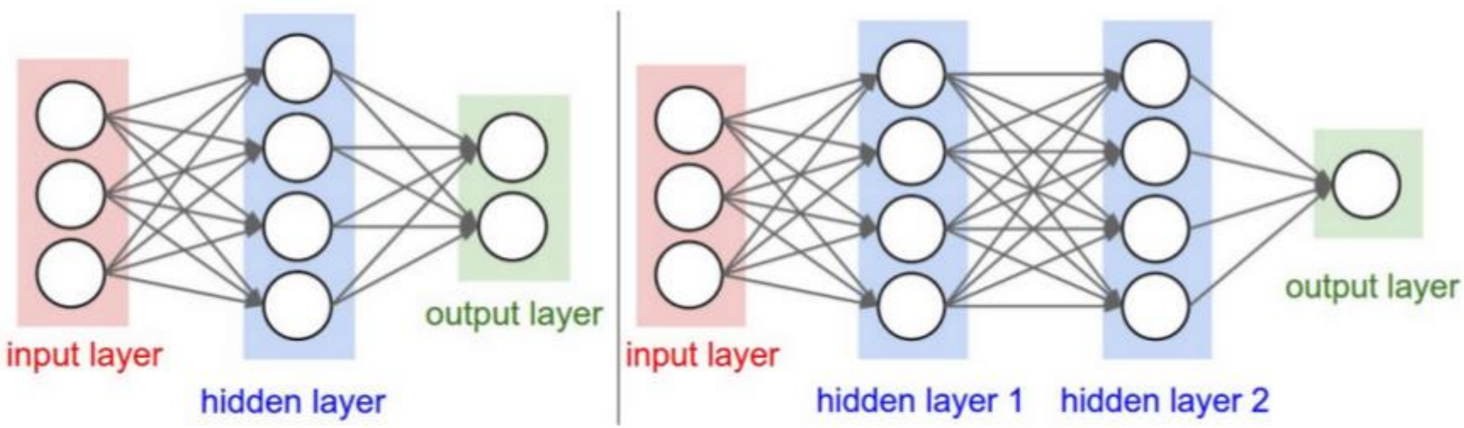
(a) ReLU (b) LeakyReLU



[Karpathy, CS231n]

Mini-batch SGD

- Loop:
  - **Sample** a batch of data
  - **Forward** prop it through the network, get loss
  - **Backprop** to calculate the gradients
  - **Update** the parameters (weights) using the gradient

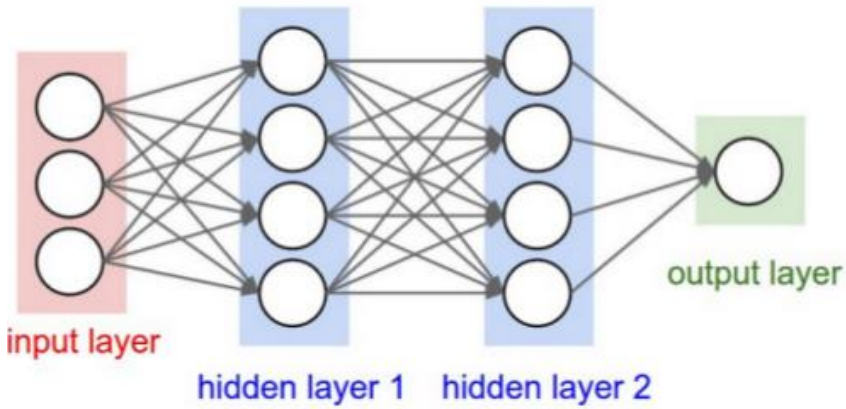


[Karpathy, CS231n]

Image data

Can we use Fully-connected nets for image classification?

Is a simple reshaping of the image working?

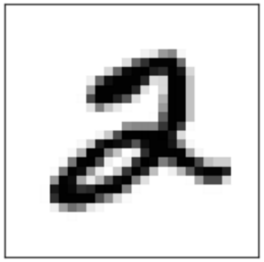


[Karpathy, CS231n]

Why Convnets?

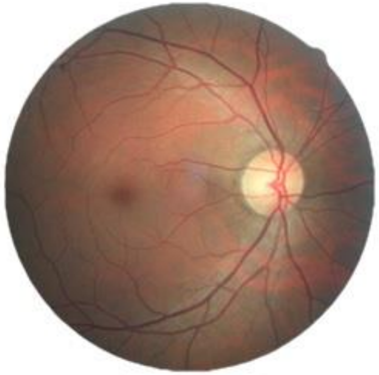
Images are (often) large

- MNIST: 28 x 28 x 1 (width x height x channels)
  = 784 input dimensions



- 200 x 200 x 3
  = 120,000 input dimensions

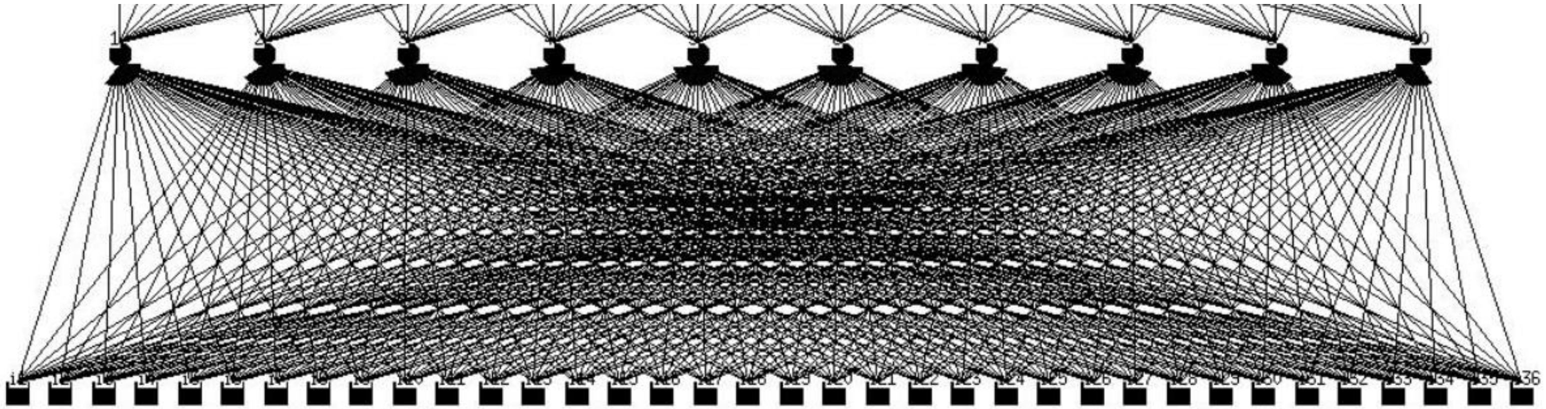
- 512 x 512 x 3
  = 786,432 input dimensions



Why Convnets?

Feeding the reshaped image into a fully connected net yields a huge amount of parameters (weights)

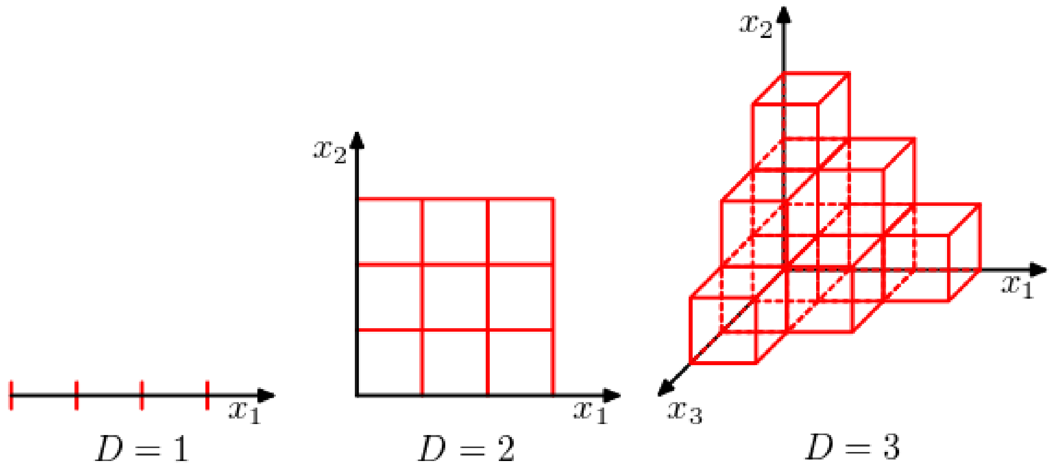
- Weight matrix: $W = w_{ij} \in \mathbb{R}^{m \times n}$
- With *m* pixels (input) and *n* neurons in the 1st hidden layer



[Yoshi Komiri]

Why Convnets?

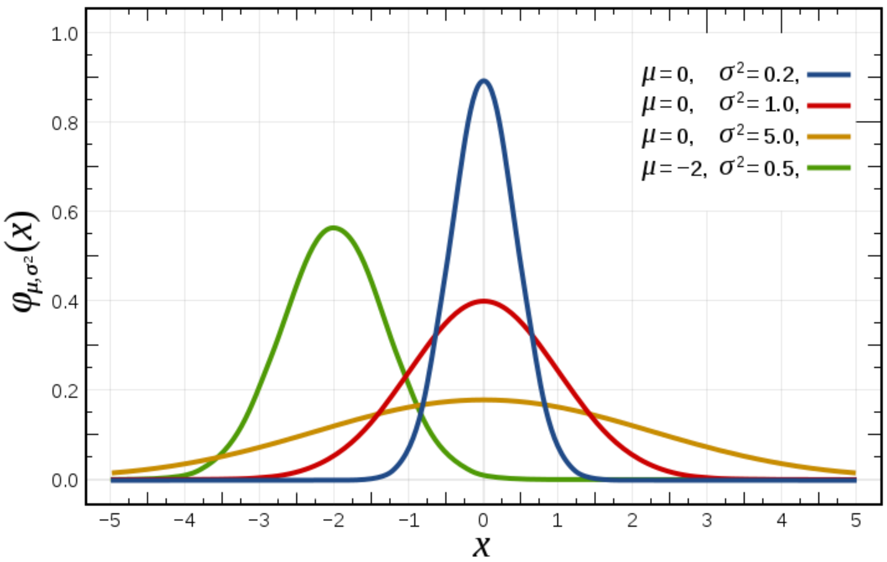
- Huge amount of weights leads to large **network capacity**
- Each image can be understood as a single point in a high-dimensional feature space
- Remember the **curse of dimensionality**
  - The volume of the feature space increases so fast that the available data become sparse
  - Amount of data to "fill" the space grows exponentially



[Bishop, Patern Recognition and Machine Learning]

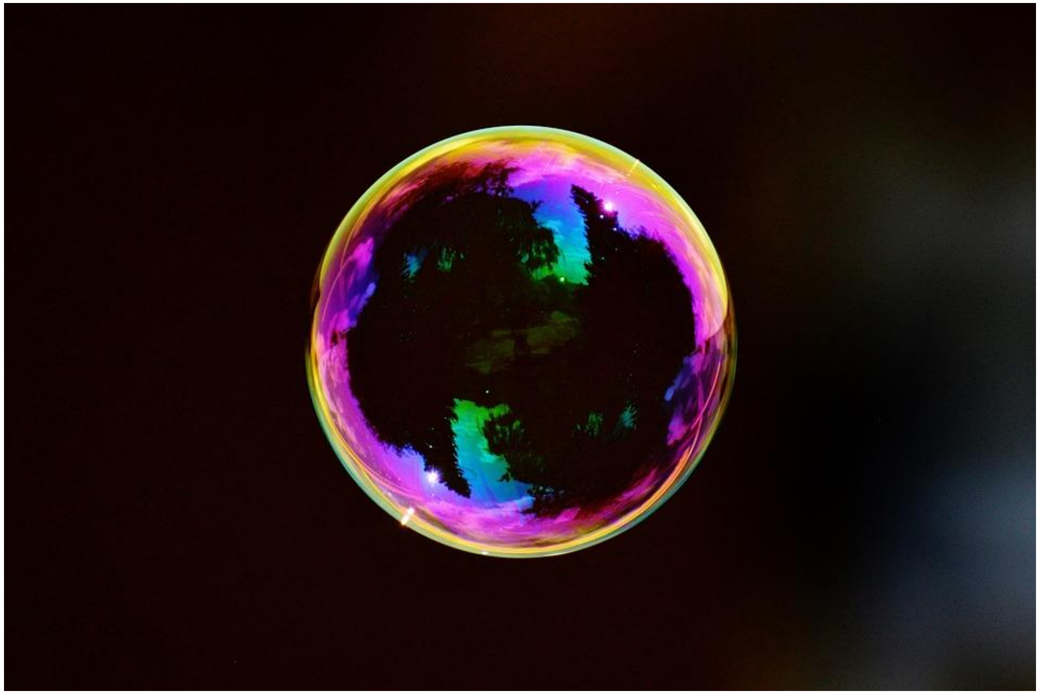
Why Convnets?

- Human intuition is mostly wrong in high dimensions!
- E.g.: Where do you think is the most probability mass of a high dimensional multivariate normal distribution?



Why Convnets?

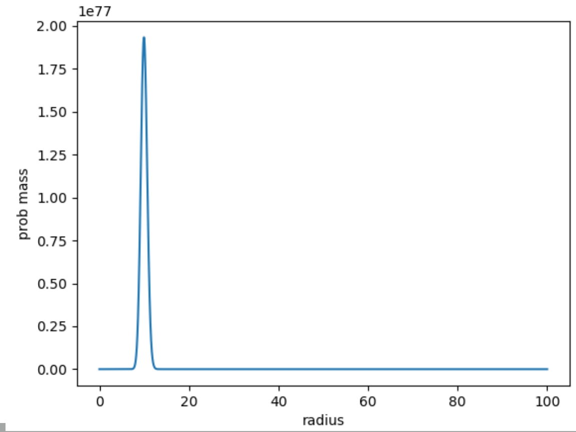
- Human intuition is mostly wrong in high dimensions!
- E.g.: Where do you think is the most probability mass of a high dimensional multivariate normal distribution?
- -> It's concentrated in a thin shell some distance away from the origin!



Why Convnets?

- Human intuition is mostly wrong in high dimensions!
- E.g.: Where do you think is the most probability mass of a high dimensional multivariate normal distribution?
- -> It's concentrated in a thin shell some distance away from the origin!

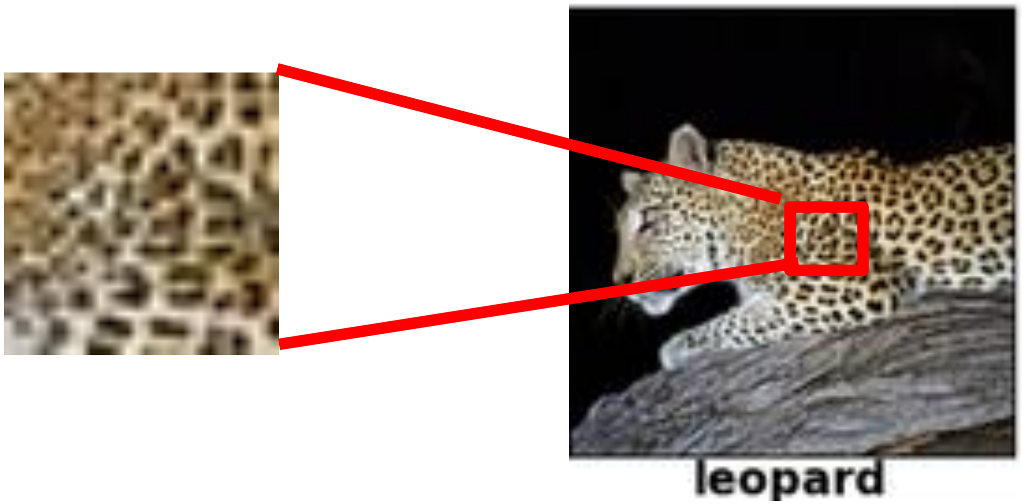
- Volume of a sphere in d dimensions is proportional to r^d.
  - For delta radius: vol_shell = delta_r * d * r^(d-1)
- Probability density: prob_dens = exp(-r^(2) / 2)
- Therefore: Probability mass: vol_shell * prob_dens
- --> Leads to Chi distribution



Why Convnets?

Images have (often) a local structure

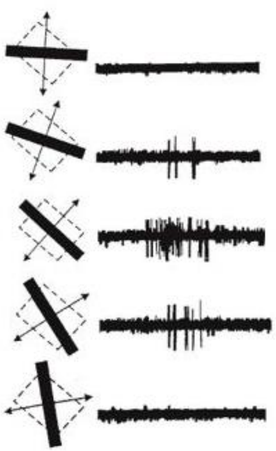
- Not every pixel contains new "information"
  - Neighboring pixels are correlated
  - Using every pixel is wasteful
- Image data can be seen as a composition of semantic elements
  - e.g. we only need to learn the leopard pattern once

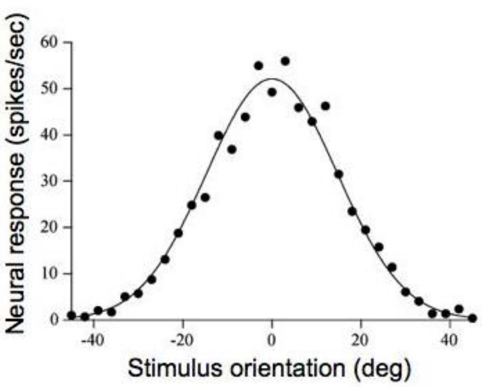


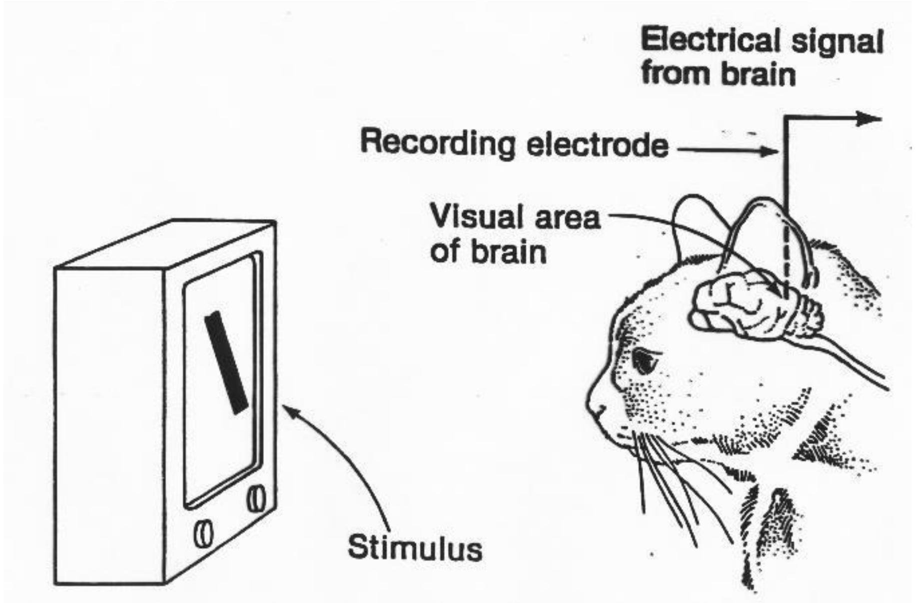
Why Convnets?

The nature as a role model

- Hubel & Wiesel
  - 1959: Receptive Fields of Single Neurones in the Cat's Striate Cortex
  - 1962: Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex
  - 1968: ...



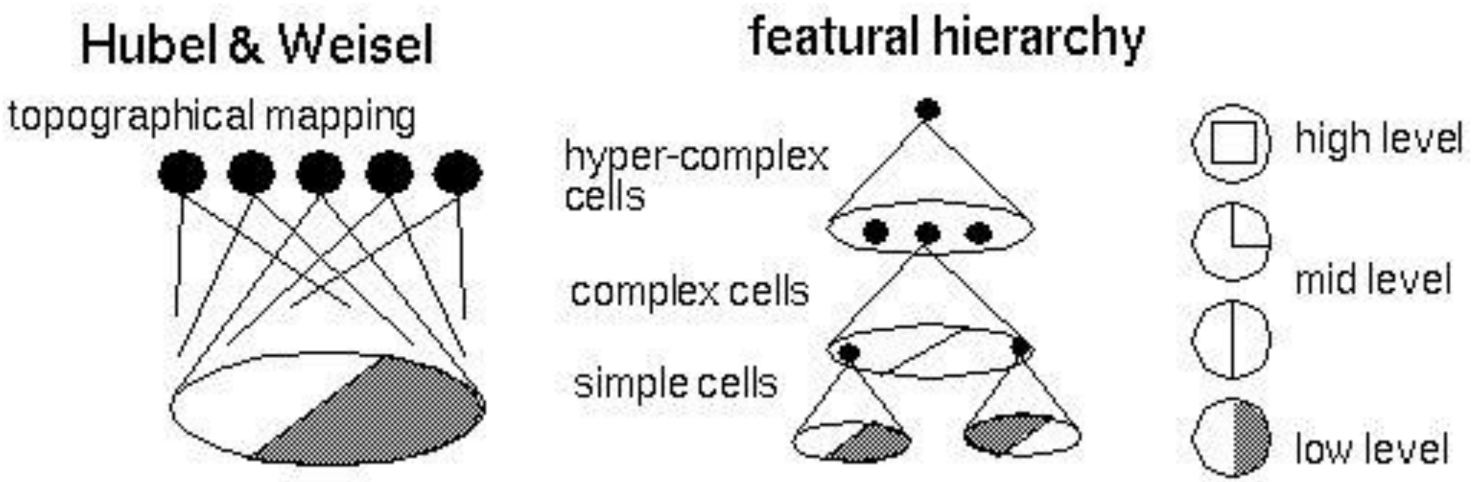




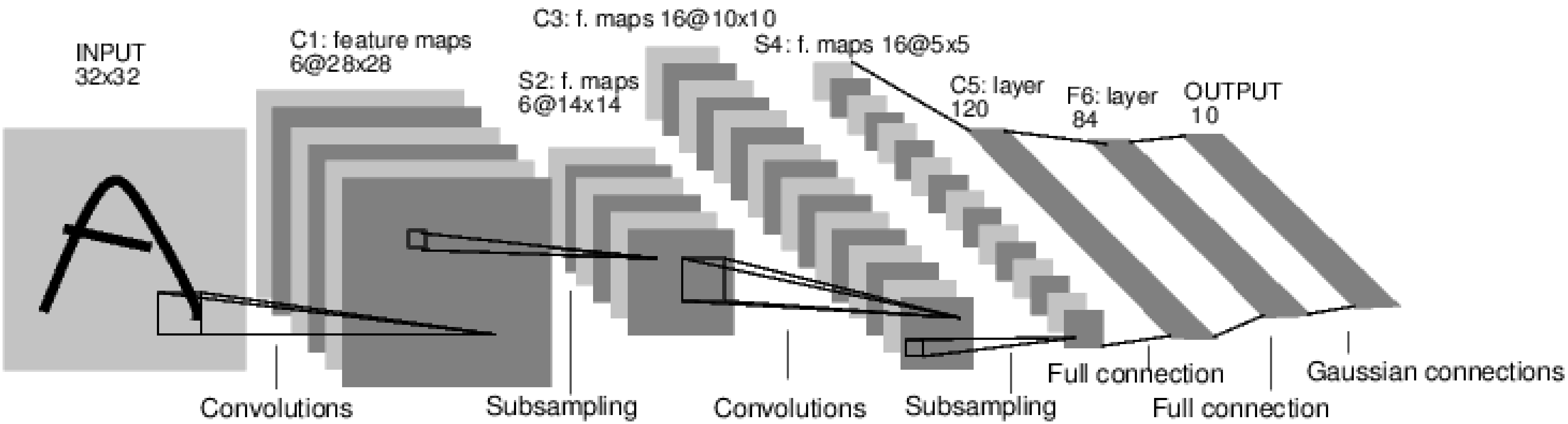
Why Convnets?

The nature as a role model

- Hubel & Wiesel's main findings are that the visual cortex is organized hierarchically
- Low level features (edges, …)
- To high level "concepts"



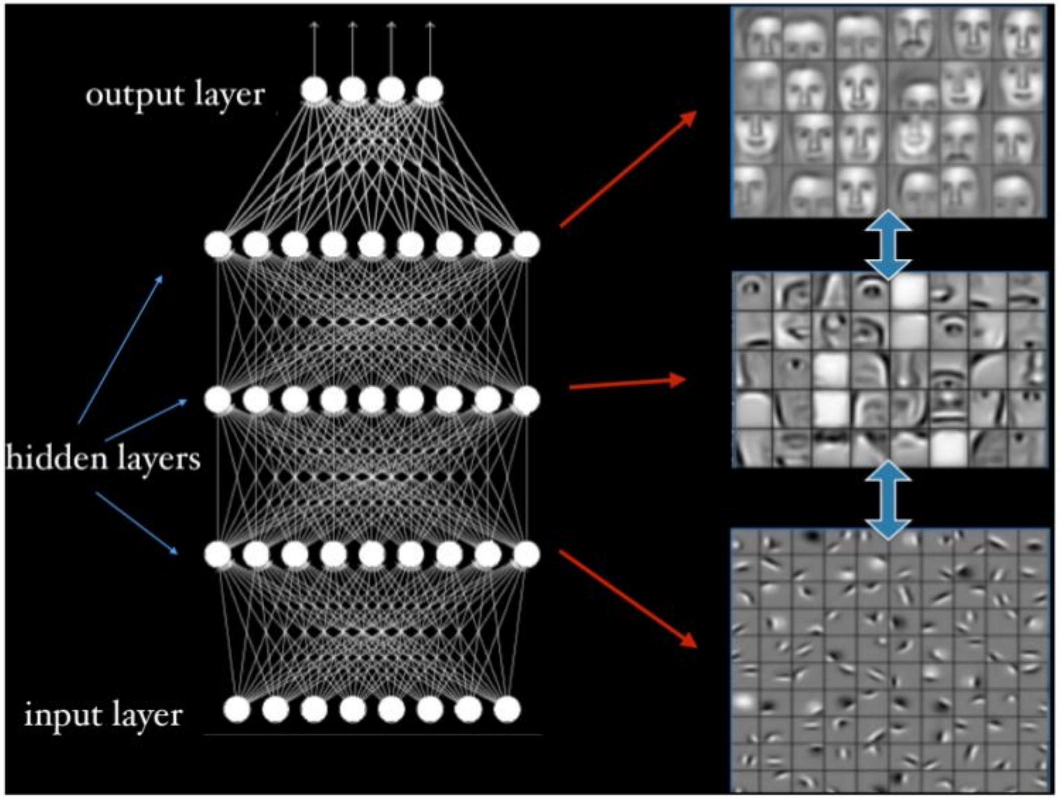
Convolutional Neural Networks



[Yann LeCun 1980, LeNet-5]

ConvNets imitate the "mammalian" vision

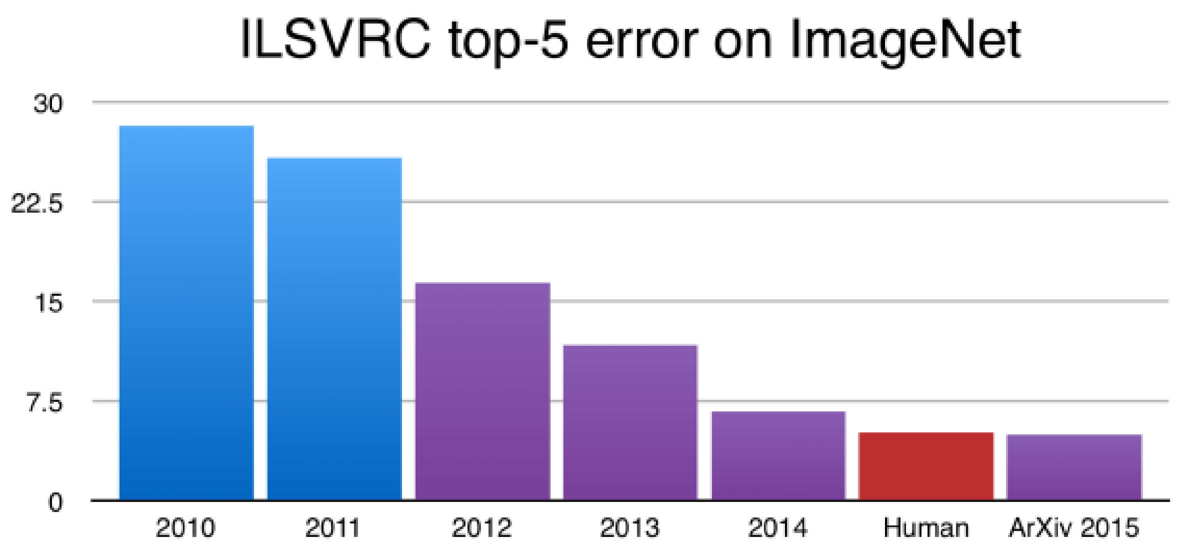
- Early layers learn "simple" features
- Later layers stack these features together and produce more complex features



[Lee et al., Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations]

ConvNets imitate the "mammalian" vision

- And perform very well



[NVIDIA]

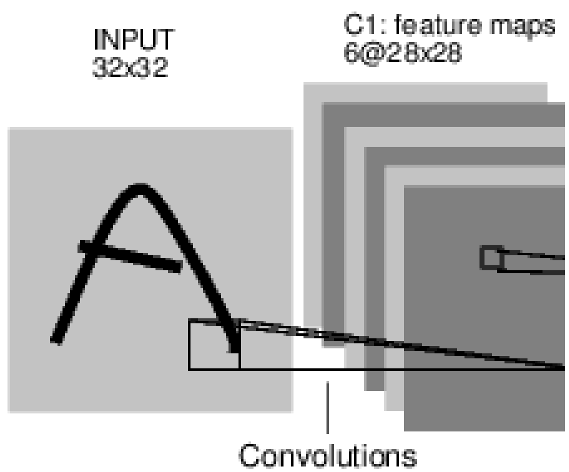
Convolutional Neural Networks

ConvNets consist of

- *Convolution Layers*
- *Subsampling / Pooling Layers*
- Detector Layer: Rectified Linear Units (Nonlinearities)
- Fully-Connected Layers

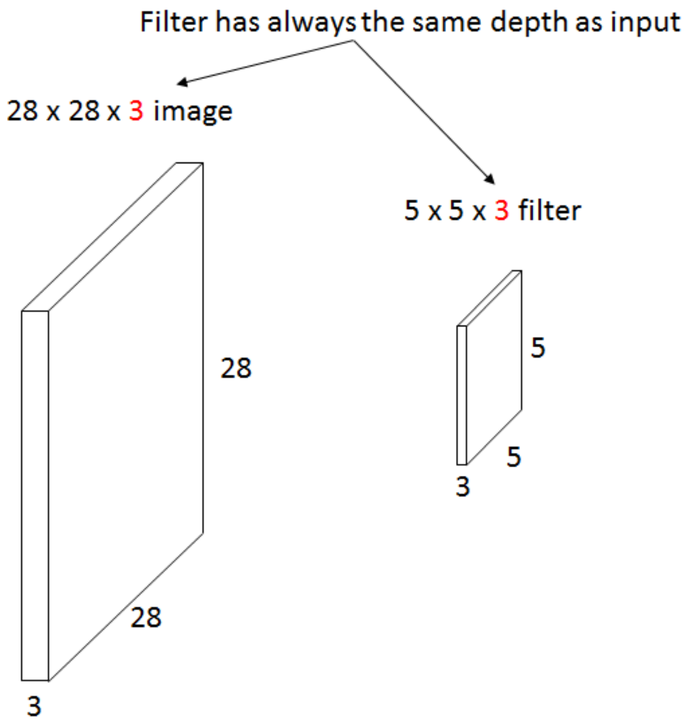
Convolution layers

- Are the core building blocks of ConvNets
- Consist of a set of learnable fiters
- Each filter is small spatially (width and height) but extends through the full depth of the input volume

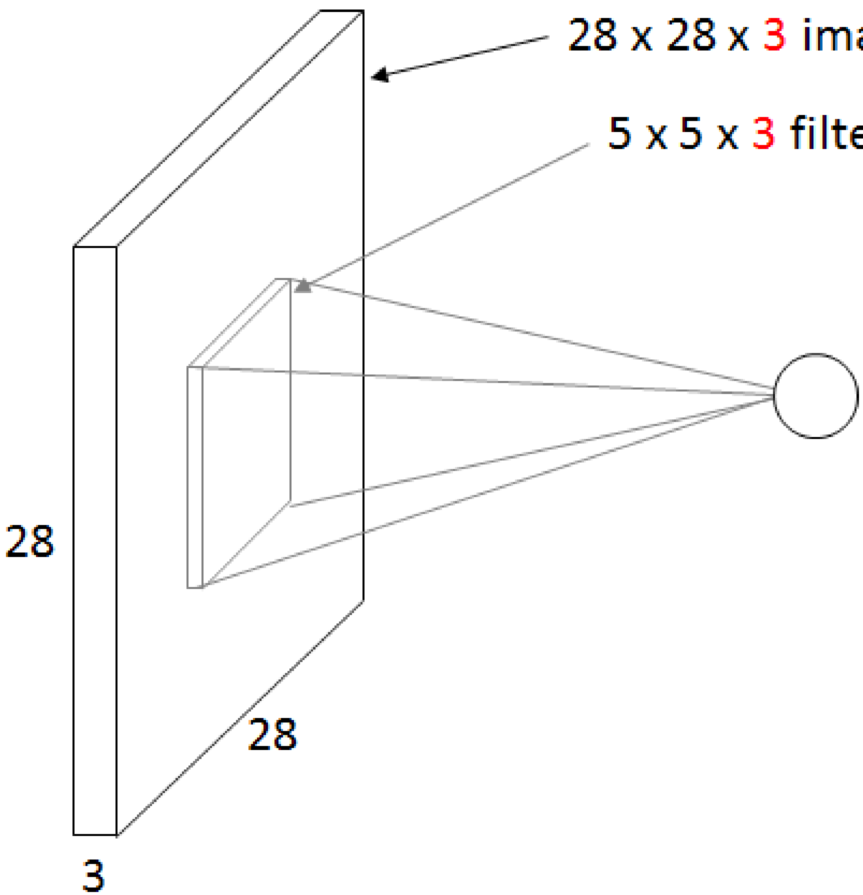


Convolution layers

- Filters slide through the input activation and compute a dot product of its filter weights and the input at any position
- Same as convolution of input and filter

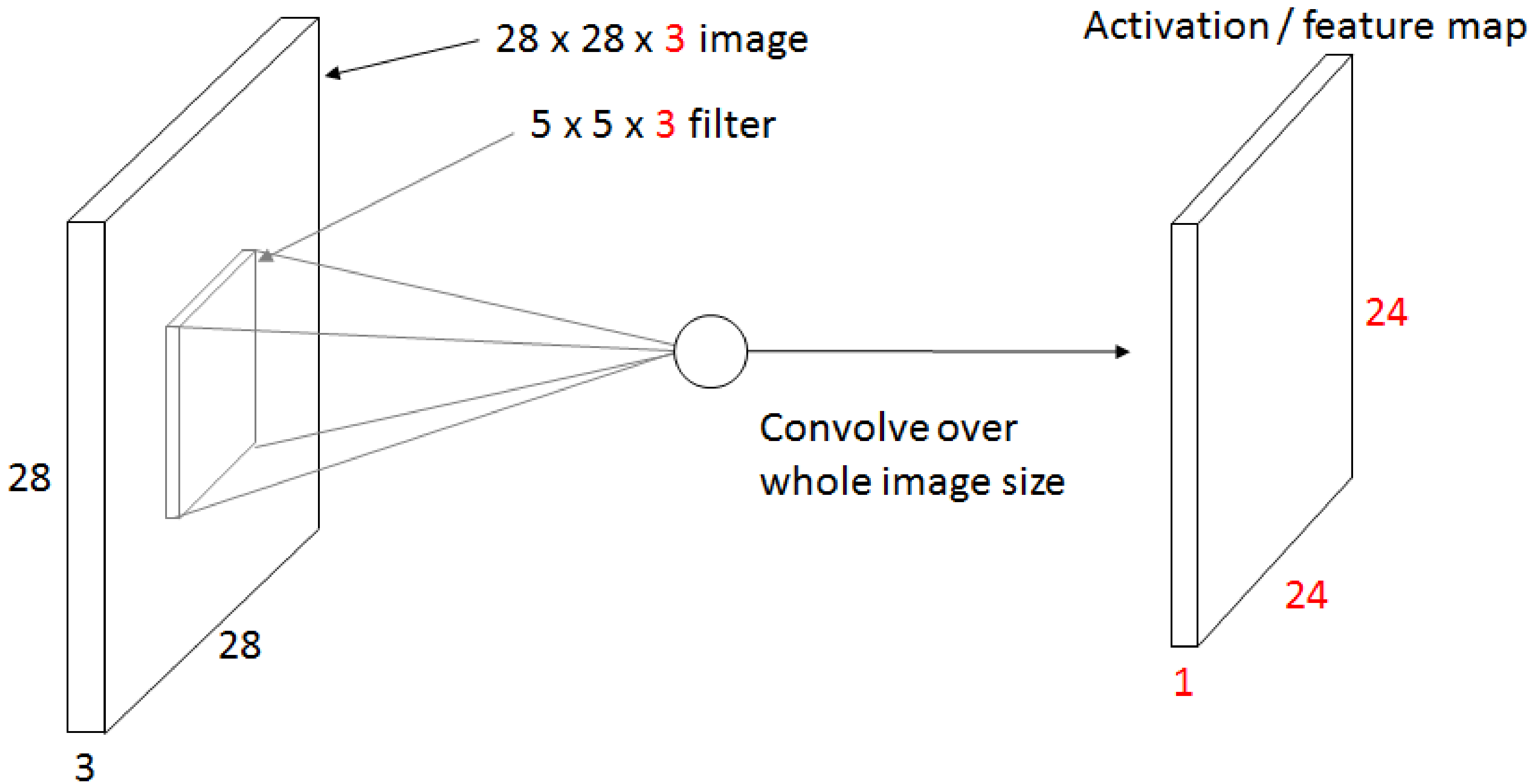


Convolution layers



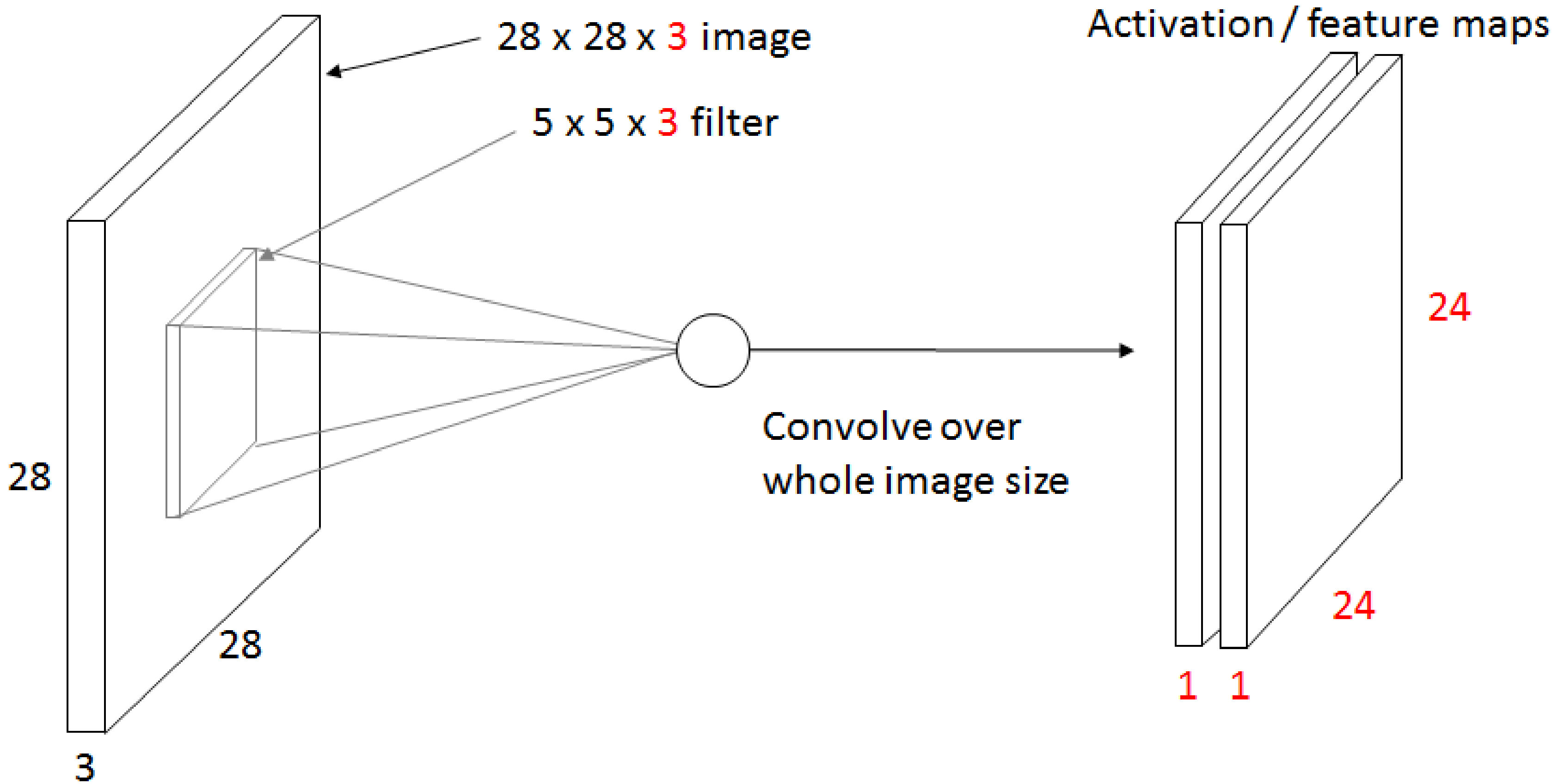
1 result:
75 dimensional dot product + bias between the filter and the 5x5x3 chunk of the image at the current filter position

Convolution layers



Convolution layers

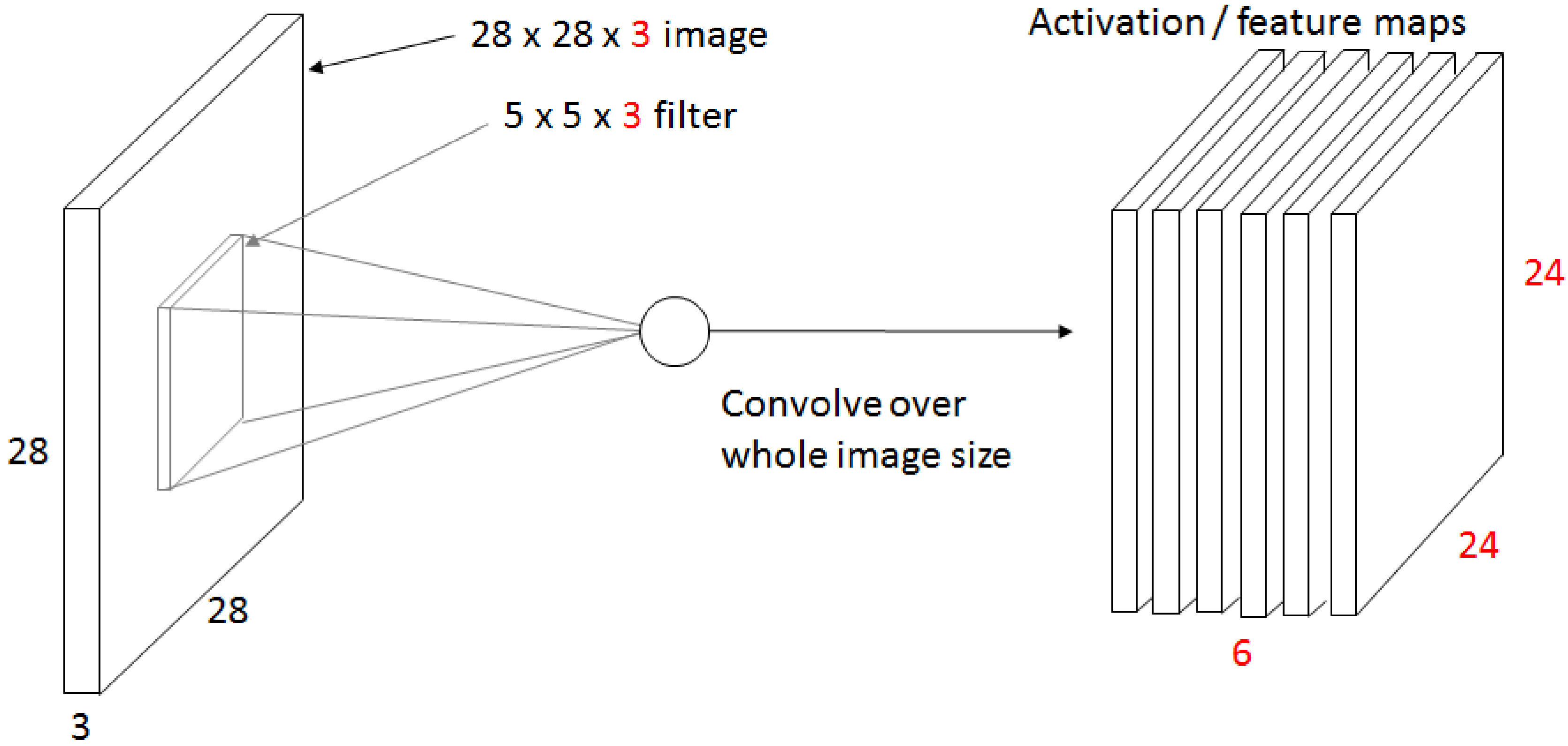
A second filter creates another feature map!



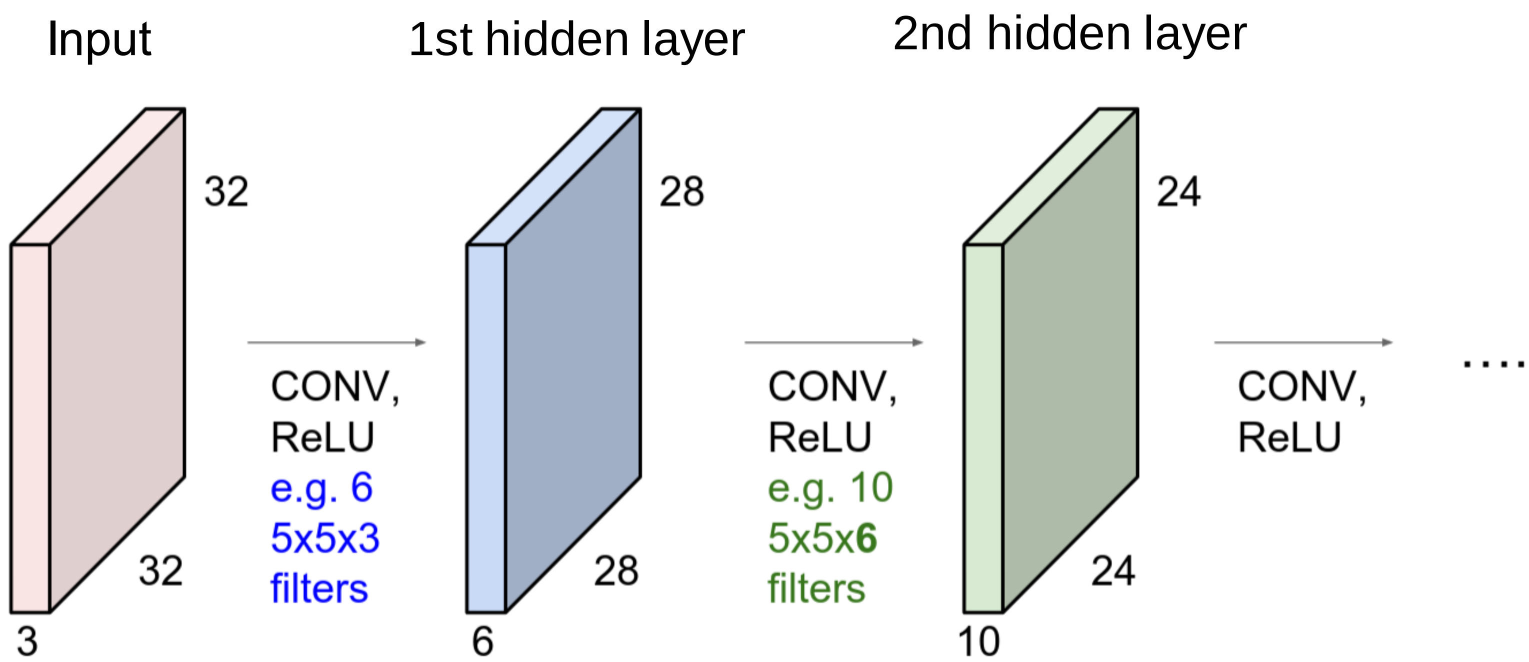
Convolution layers

6 5x5 filters create **6** separate feature maps.

These feature maps are stacked into a 24x24x**6** feature volume / image



A ConvNet is a sequence of ConvLayers connected with activation functions

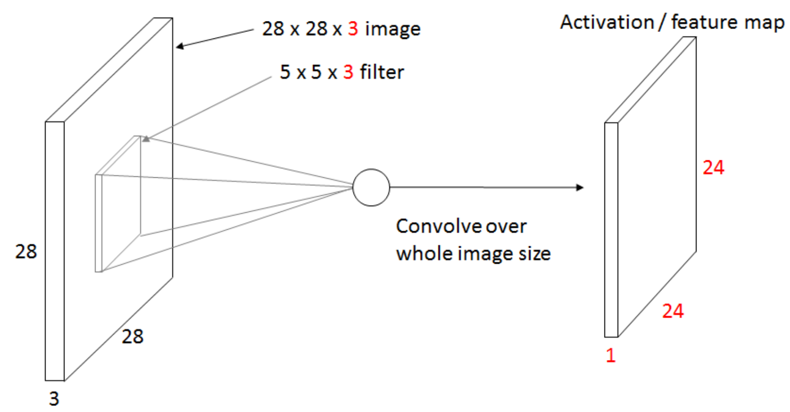


[Karpathy, CS231n]

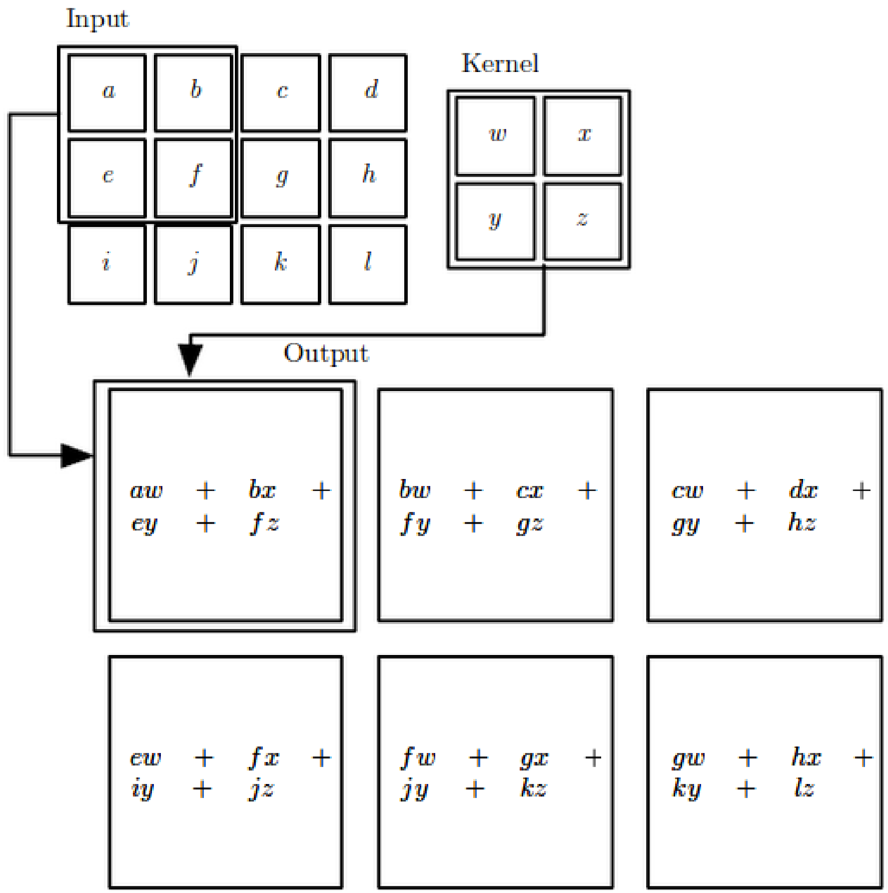
Convolutions affect spatial dimensions

- General formula $$s(t) = \int x(a)w(t-a)da$$
- Often denoted as $$s(t) = (x * w)(t)$$
- Discrete formula $$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

[Goodfellow et al., Deep Learning]

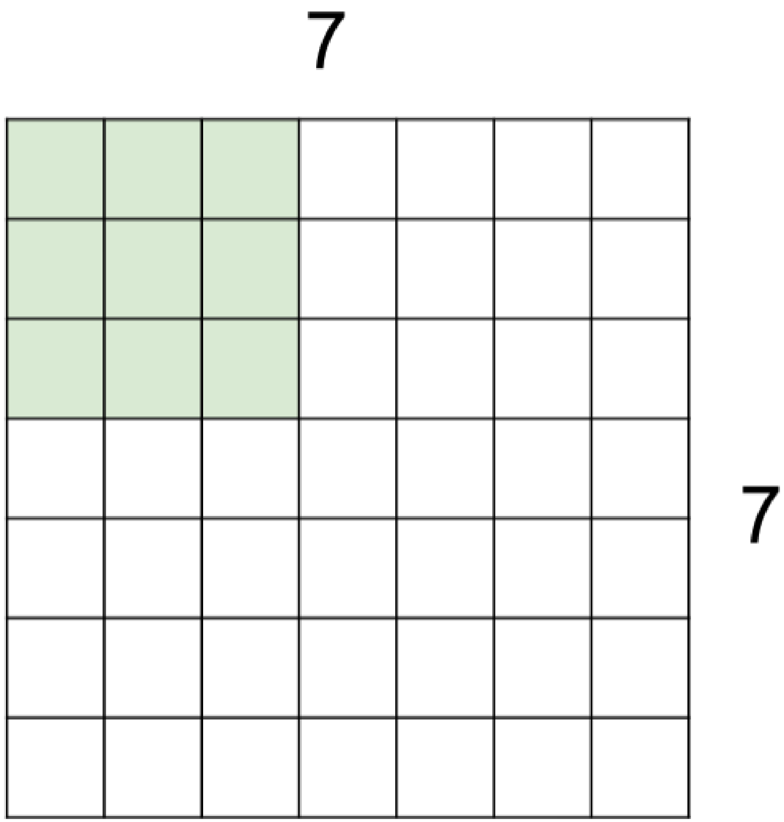


"Valid" 2-D convolution without kernel flipping



[Goodfellow et al., Deep Learning]

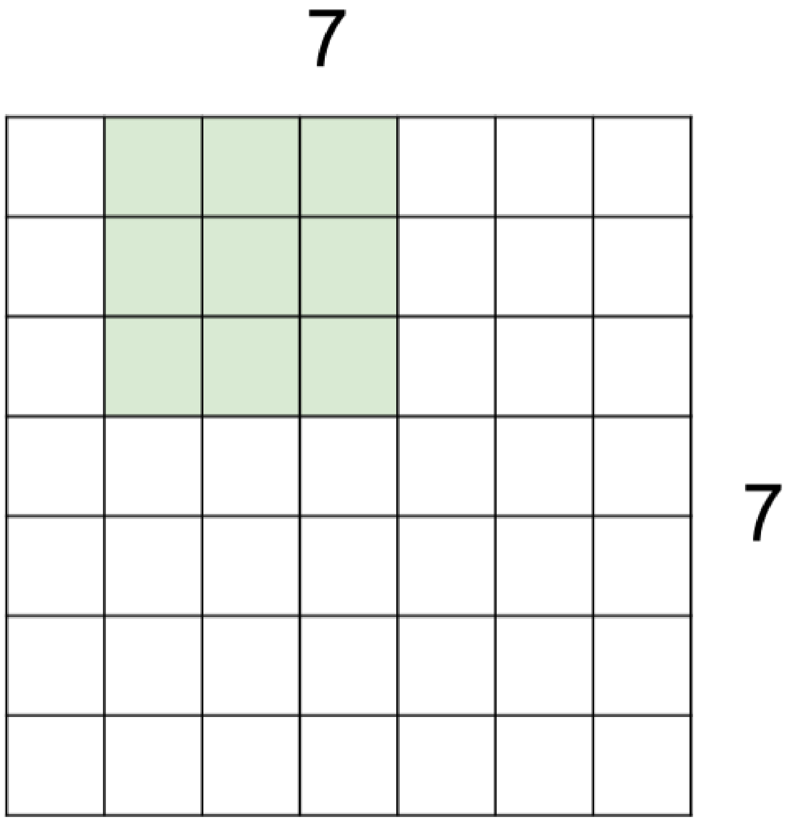
"Valid" 2-D convolution without kernel flipping



7x7 input (spatially)
assume 3x3 filter

[Karpathy, CS231n]

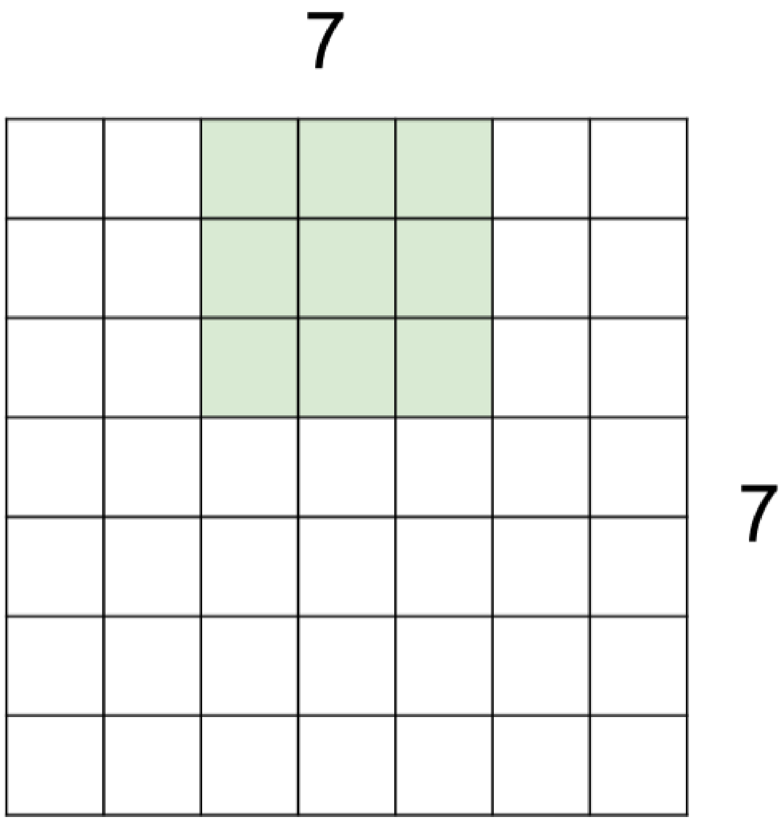
"Valid" 2-D convolution without kernel flipping



7x7 input (spatially)
assume 3x3 filter

[Karpathy, CS231n]

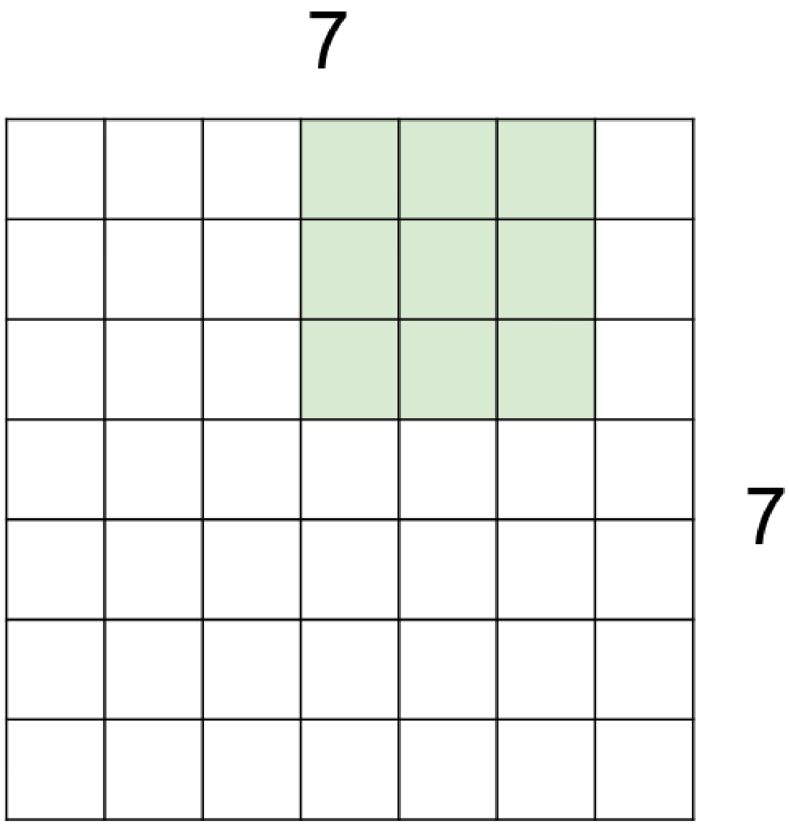
"Valid" 2-D convolution without kernel flipping



7x7 input (spatially)
assume 3x3 filter

[Karpathy, CS231n]

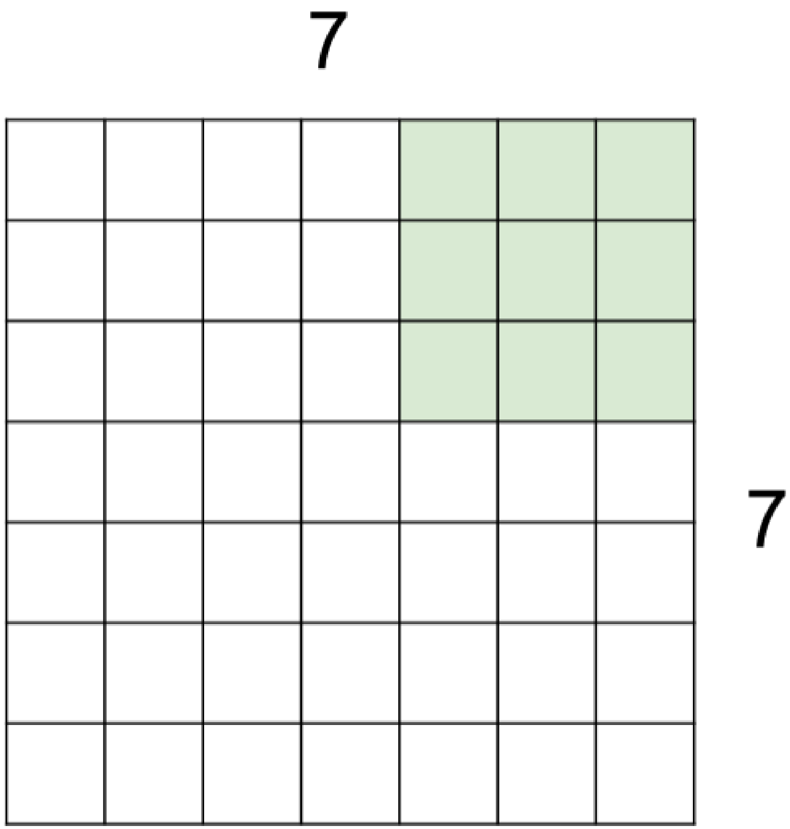
"Valid" 2-D convolution without kernel flipping



7x7 input (spatially)
assume 3x3 filter

[Karpathy, CS231n]

"Valid" 2-D convolution without kernel flipping

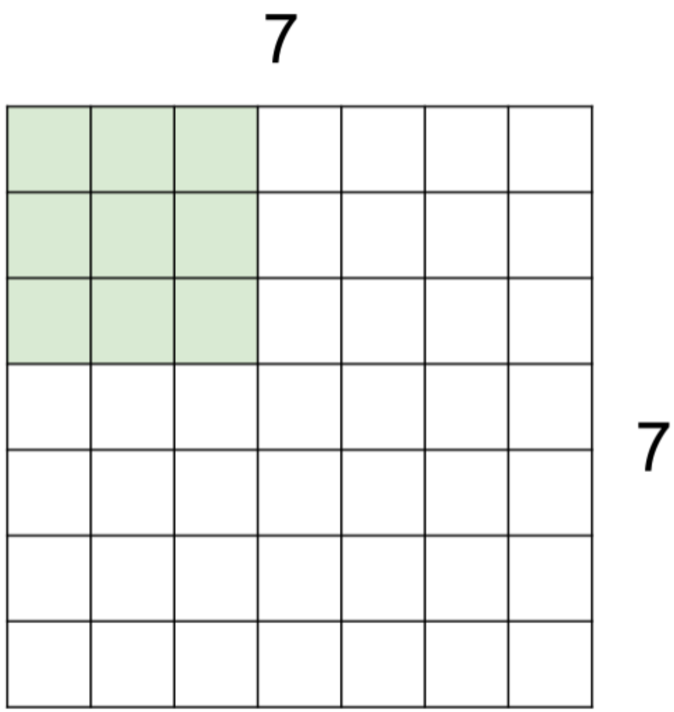


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

[Karpathy, CS231n]

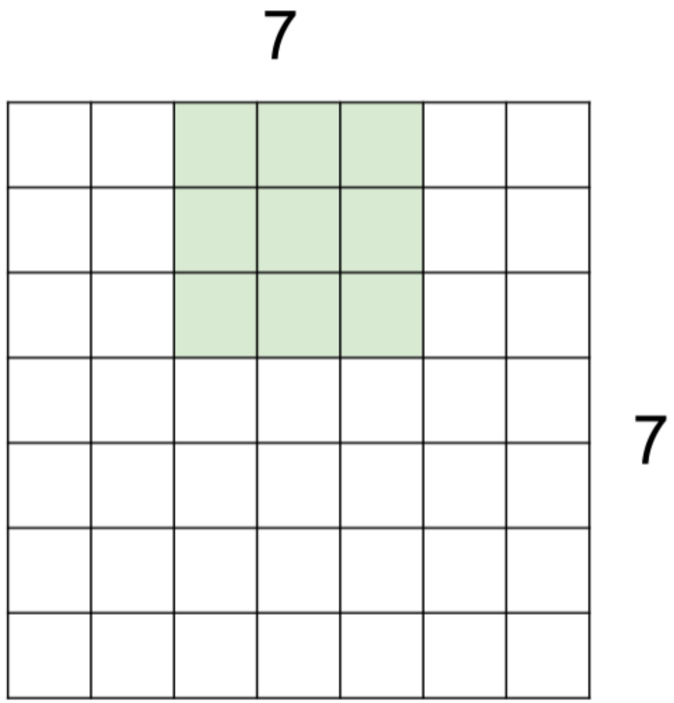
"Valid" 2-D convolution without kernel flipping, stride 2



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

[Karpathy, CS231n]

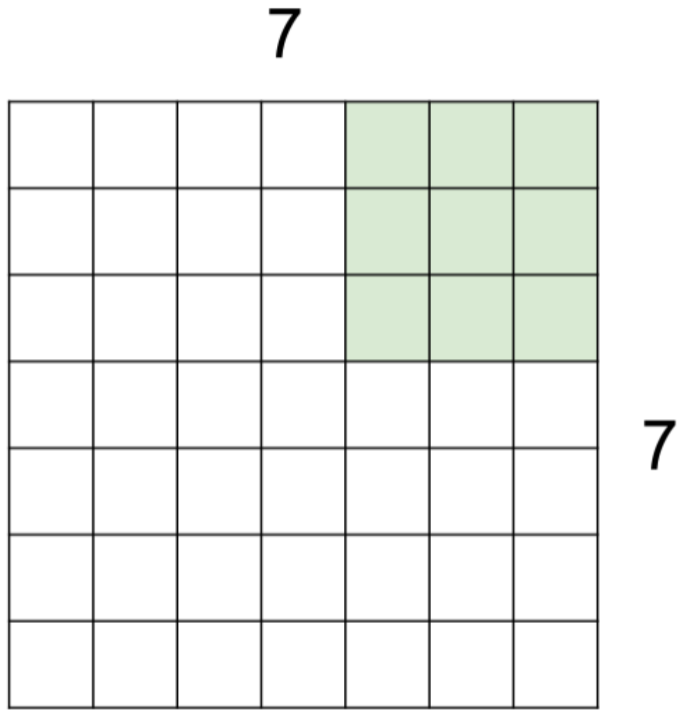
"Valid" 2-D convolution without kernel flipping, stride 2



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

[Karpathy, CS231n]

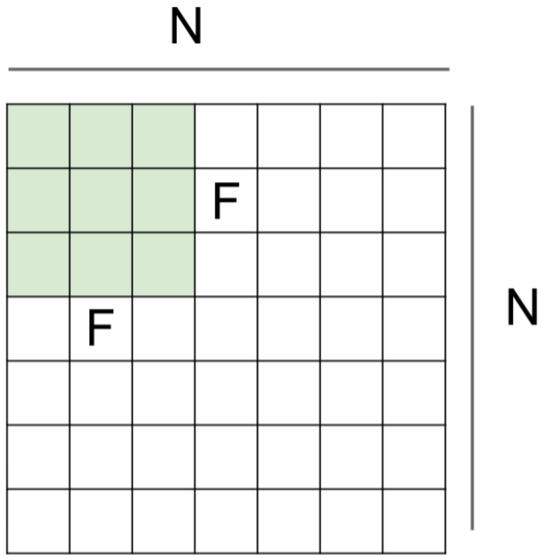
"Valid" 2-D convolution without kernel flipping, stride 2



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

[Karpathy, CS231n]

"Valid" 2-D convolution without kernel flipping

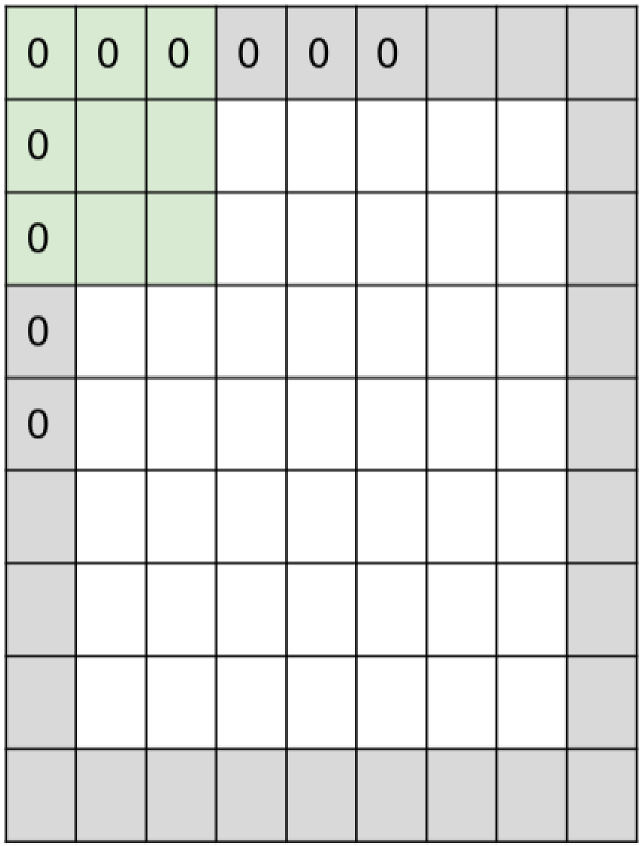


Output size:
(N - F) / stride + 1

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 :\

[Karpathy, CS231n]

Zero pading of border

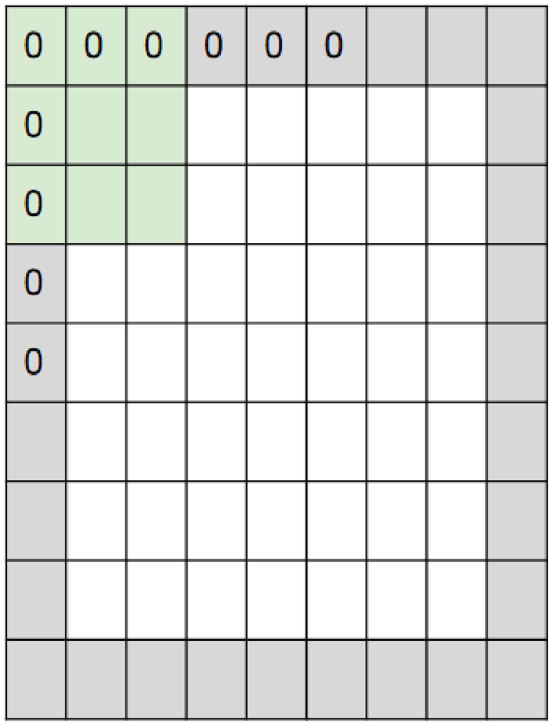


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

[Karpathy, CS231n]

Zero pading of border



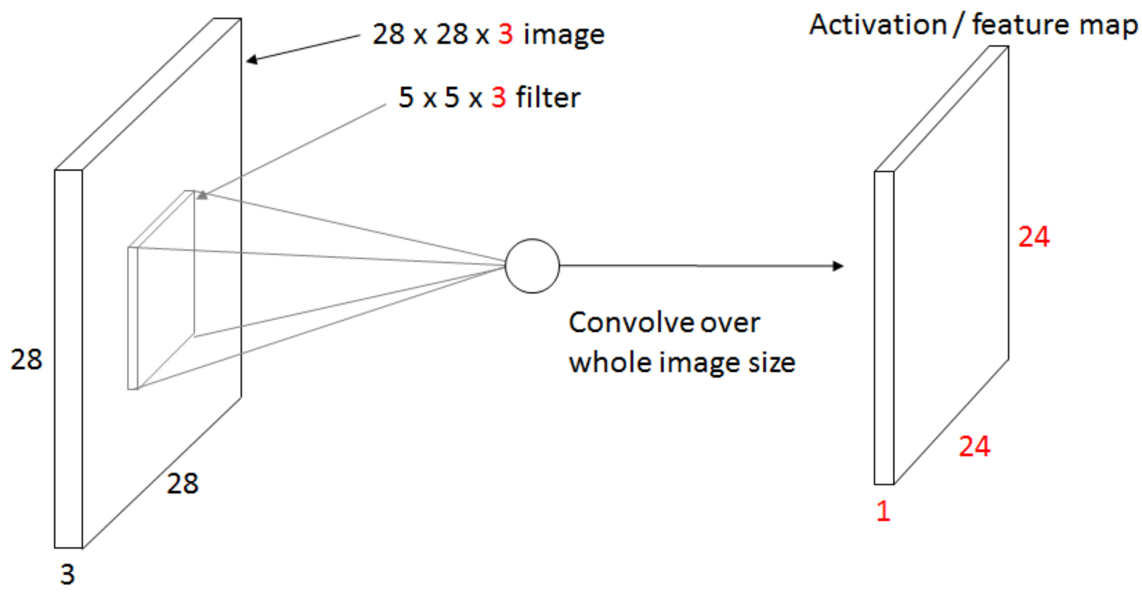
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

[Karpathy, CS231n]

Zero pading of border might preserve the size

Otherwise the volume shrinks spatialy!



Zero pading of border might preserve the size

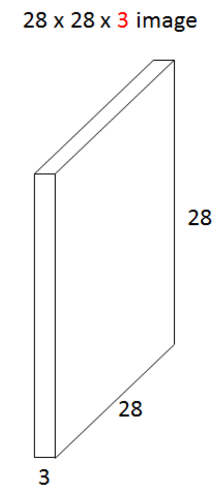
Example:

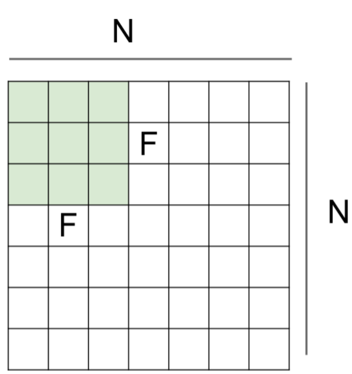
Input image: 28x28x3

10 5x5 filters with stride **1**, pad **2**

Output volume size:

???





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 :\

Zero pading of border might preserve the size

Example:

Input image: 28x28x3

10 5x5 filters with stride **1**, pad **2**

Output volume size:

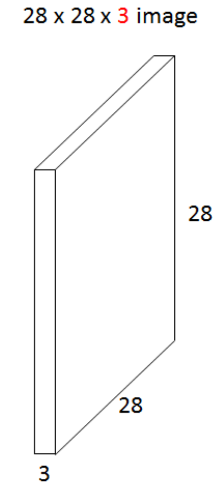
(28 + 2***2 – 5**) / **1** + 1

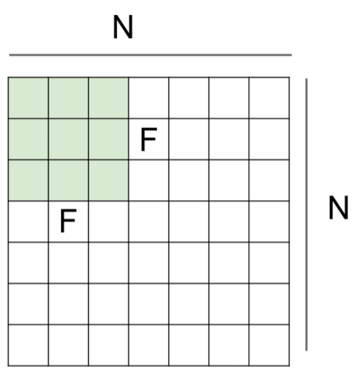
(28 + 4 – 5) / 1 + 1

27 +1 = 28 spatially

Output volume

28x28x**10**





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 :\

Number of parameters

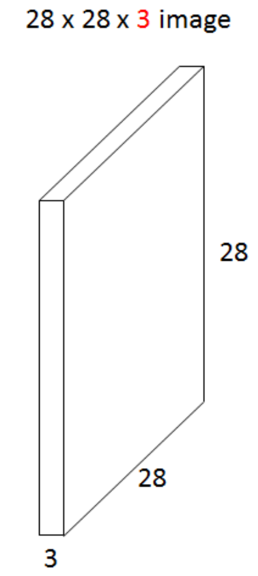
Example:

Input image: 28x28x1

10 5x5 filters with stride 1, pad 2

Number of parameters in layer?

???



Number of parameters

Example:

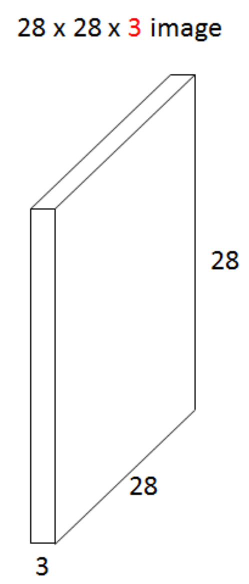
Input image: 28x28x1

10 5x5 filters with stride 1, pad 2

Number of parameters in layer?

Each filter has 5*5***1**+1 = 26 params (+1 for bias)

--> total 26 * 10 = **260**



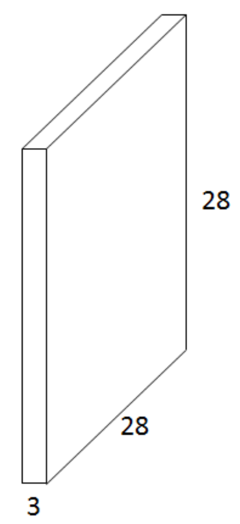
Number of parameters

Example:

Input image: 28x28x3

10 5x5 filters with stride 1, pad 2

Number of parameters in layer?



Number of parameters

Example:

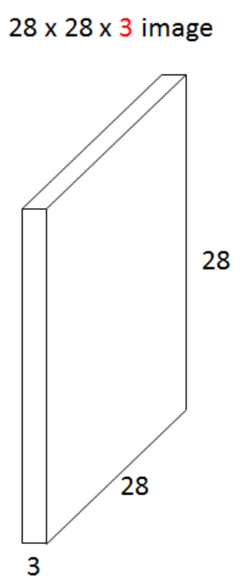
Input image: 28x28x3

10 5x5 filters with stride 1, pad 2

Number of parameters in layer?

Each filter has 5*5***3**+1 = 76 params (+1 for bias)

--> total 76 * 10 = **760**



Convolutional Neural Networks

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.

[Karpathy, CS231n]

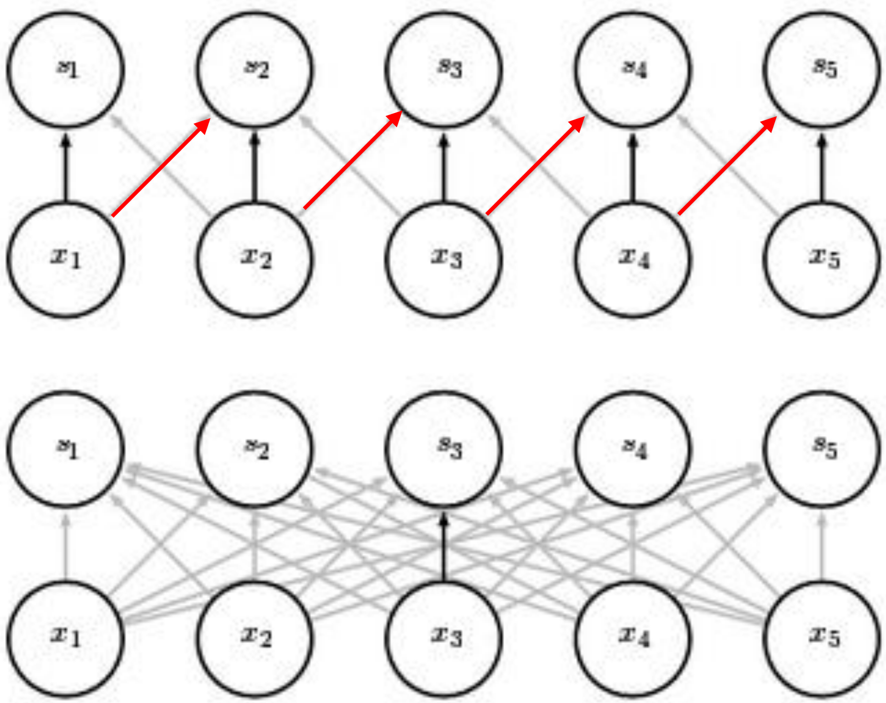
Convolutional Neural Networks

- Keras Conv2D layer

- **filters**: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of `"valid"` or `"same"` (case-insensitive).
- **data_format**: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".
- **dilation_rate**: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any stride value != 1.
- **activation**: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the `kernel` weights matrix (see initializers).
- **bias_initializer**: Initializer for the bias vector (see initializers).
- **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see regularizer).
- **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
- **kernel_constraint**: Constraint function applied to the kernel matrix (see constraints).
- **bias_constraint**: Constraint function applied to the bias vector (see constraints).

Parameter sharing

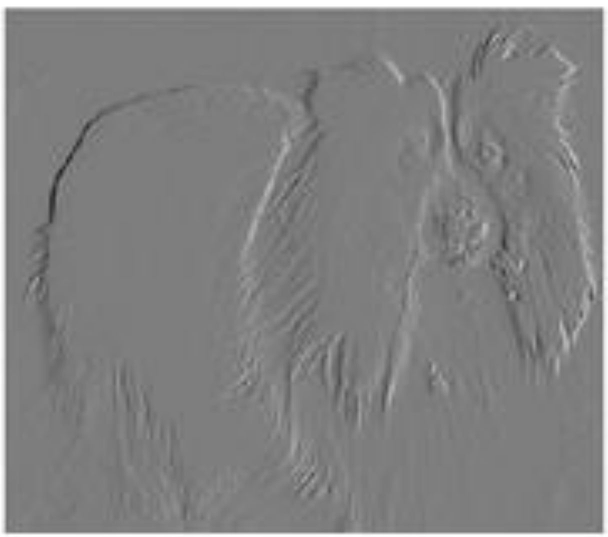
- In fully-connected net, each weight is used once to compute output
- In ConvNet each weight of the filter is used at **every** position of the input
- The weights are **tied!**



[Goodfellow et al., Deep Learning]

Parameter sharing

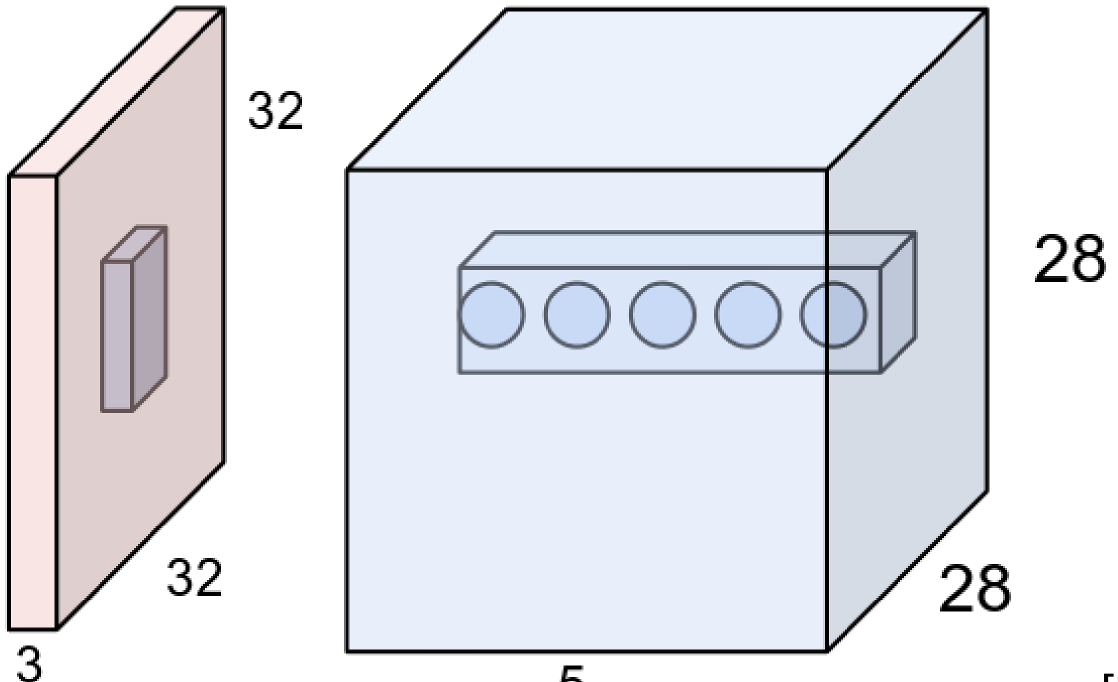
- Instead of learning a weight set for every location, we learn only one set
- Idea: A certain filter (e.g. edge detector) might be useful for every location in the input volume



[Goodfellow et al., Deep Learning]

Each filter learn a different feature

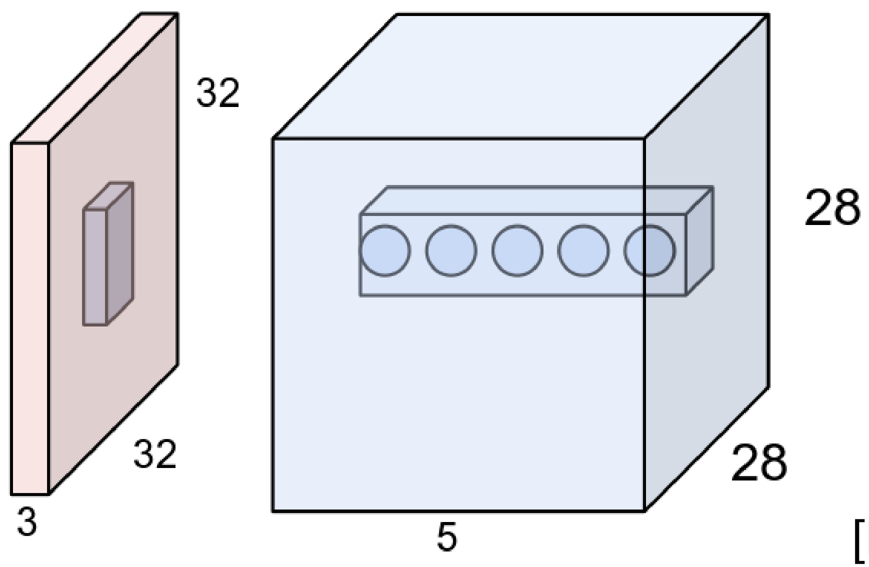
There will be 5 different neurons / filters
looking at the same location in the input volume



[Karpathy, CS231n]

Each filter learn a different feature

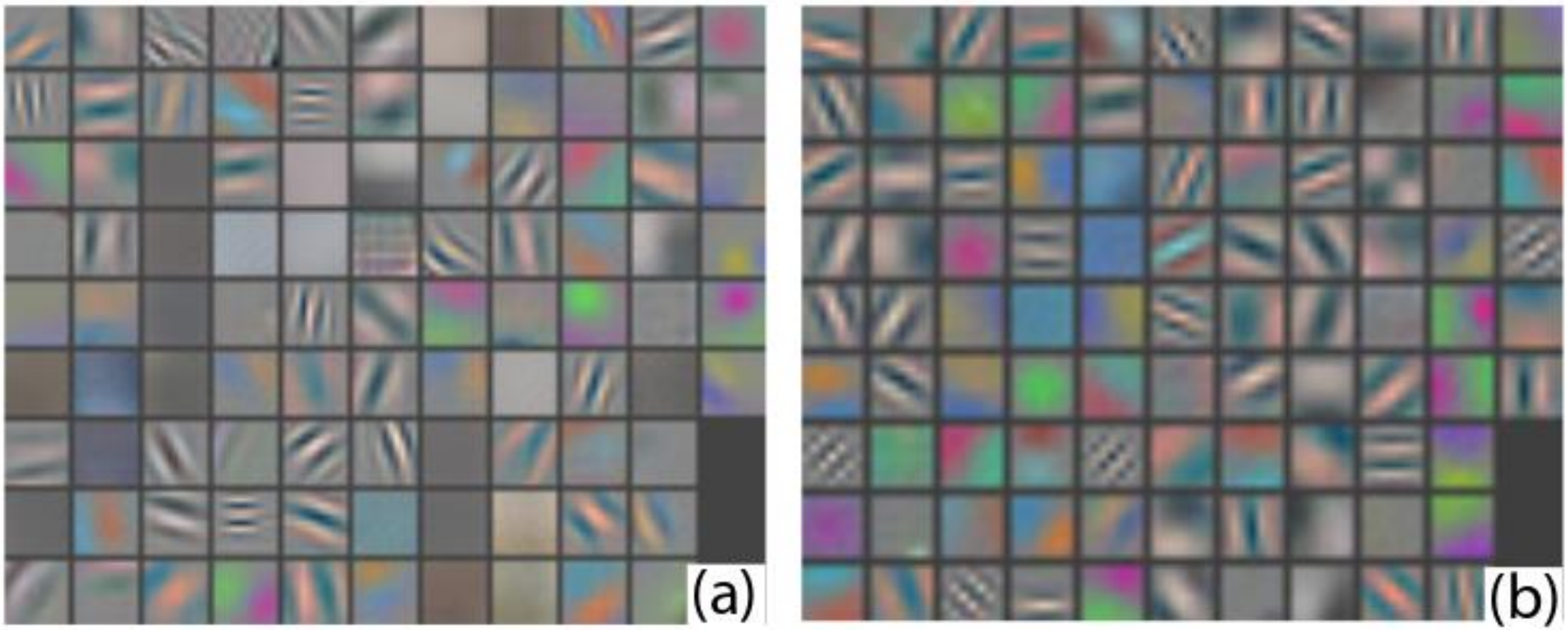
- We get 5 **depth slices** of size 28x28
- During backprop, every neuron in the volume will compute gradient for its weights
- Gradients will be added up across each depth slice and only update a single set of weights per slice



[Karpathy, CS231n]

Each filter learn a different feature

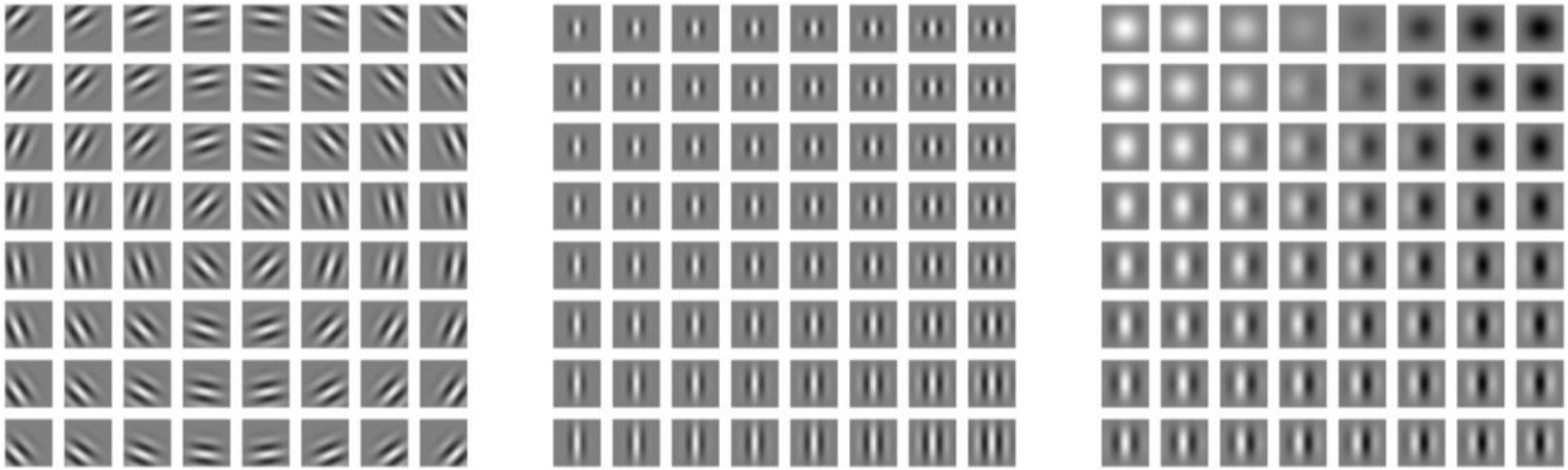
1st layer features of learned ConvNet



[Zeiler and Fergus]

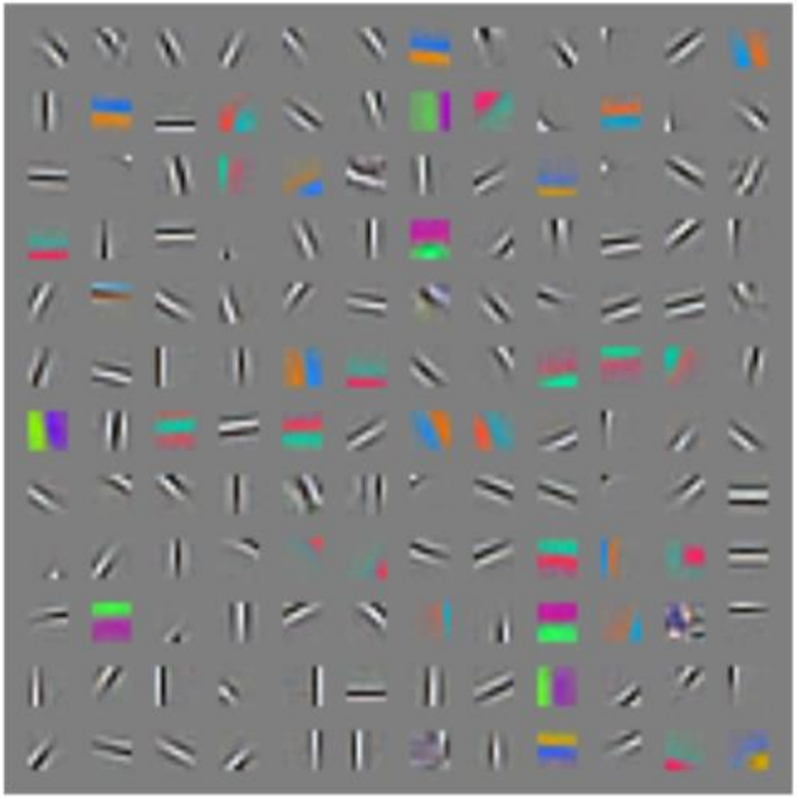
Gabor filters

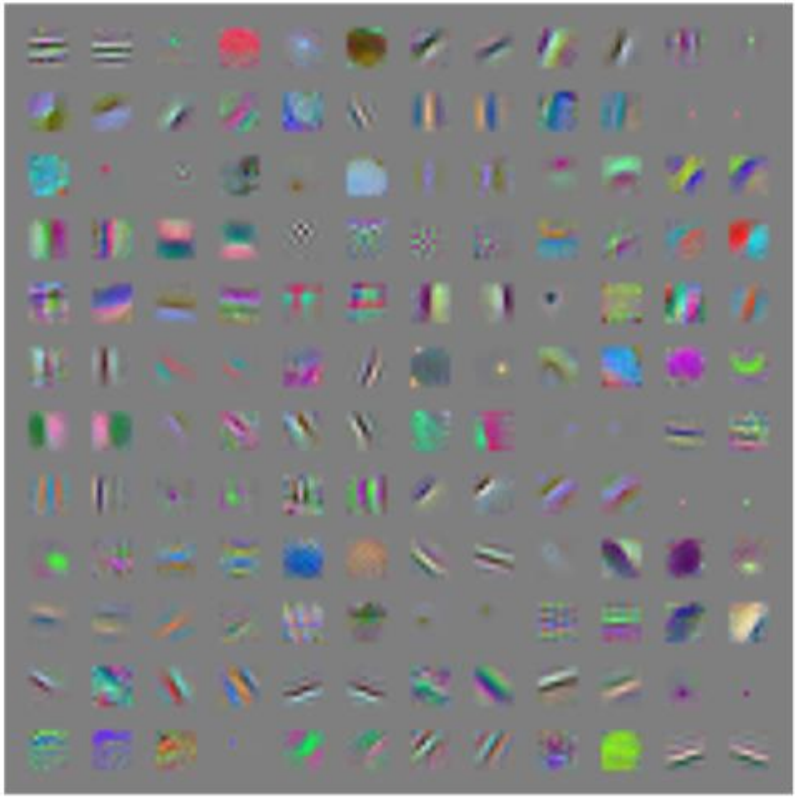
- Linear filters used for edge detection
- Gaussian kernel function modulated by a sinusoidal plane wave
- Simple cells in visual cortex of mammalian brains
  can be modeled by Gabor functions



[Goodfellow et al.,
Deep Learning]

Gabor-like learned Kernels

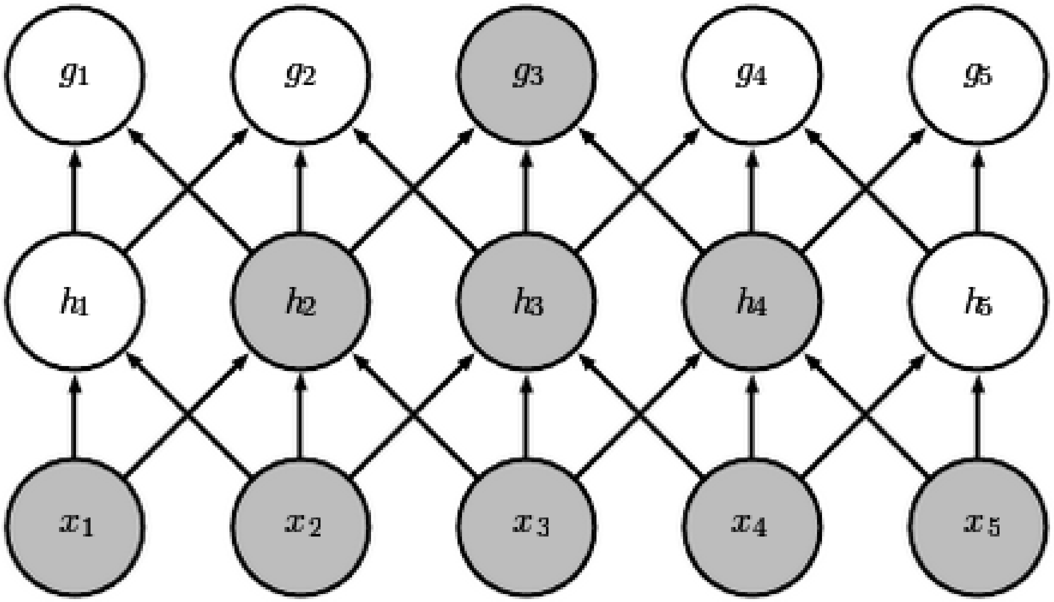




[Goodfellow et al., Deep Learning]

Convolutional Neural Networks

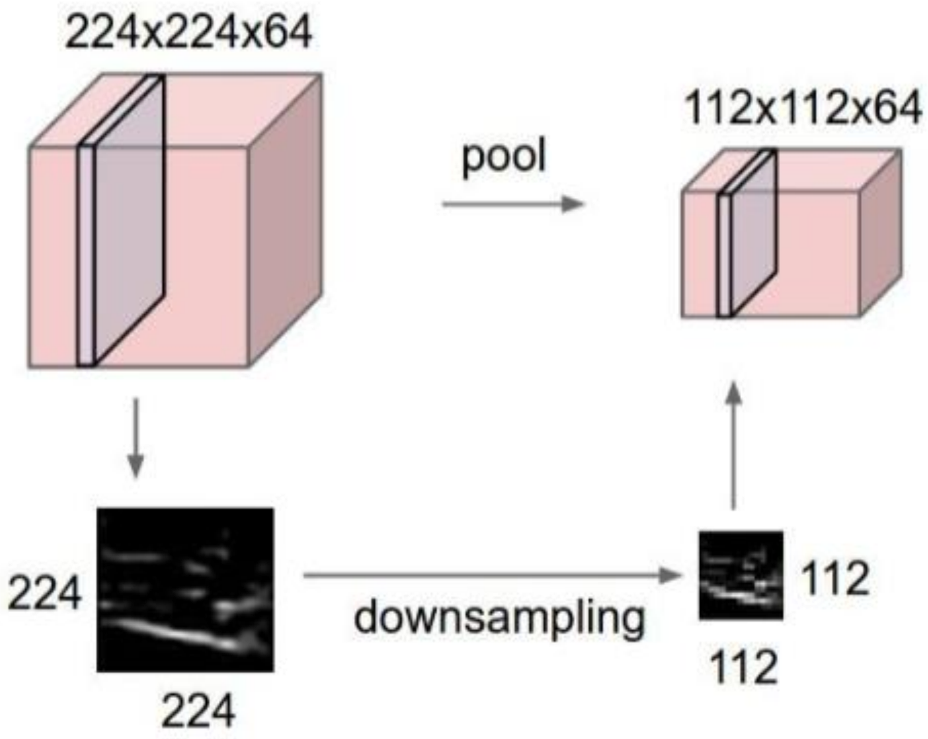
- The receptive field of the units in the deeper layers of a ConvNet gets larger
- This effect can be increased using strided convolutions or *pooling*
- Thus units in deeper layers can be connected to all or most of the input image



[Goodfellow et al., Deep Learning]

Pooling layers

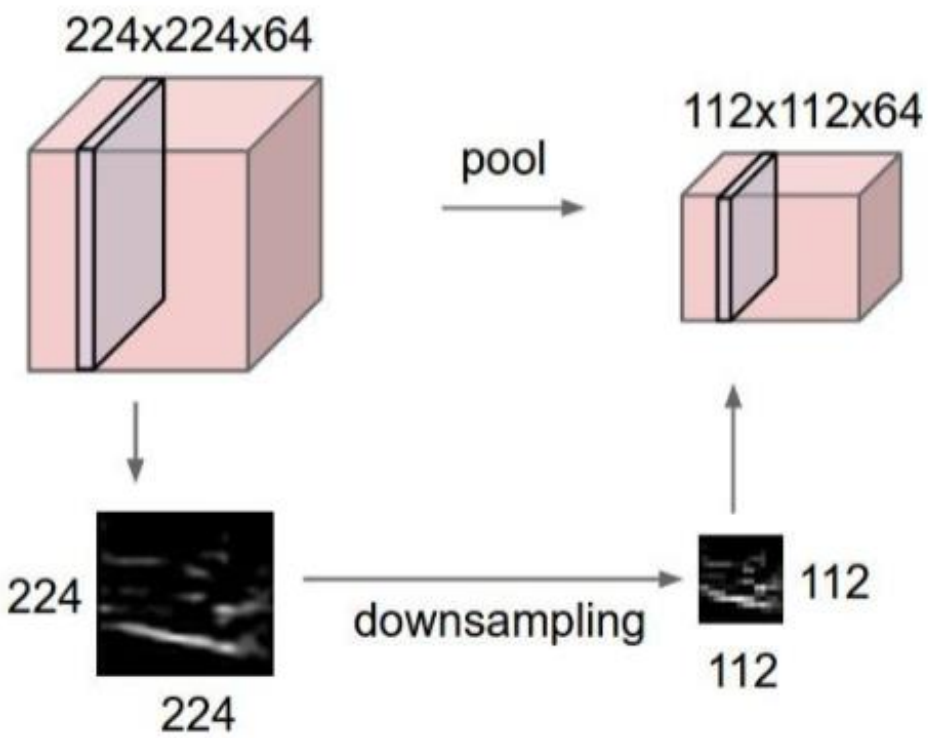
- Makes the feature maps smaller and more manageable
- Operates over each feature map independently



[Karpathy, CS231n]

Pooling layers

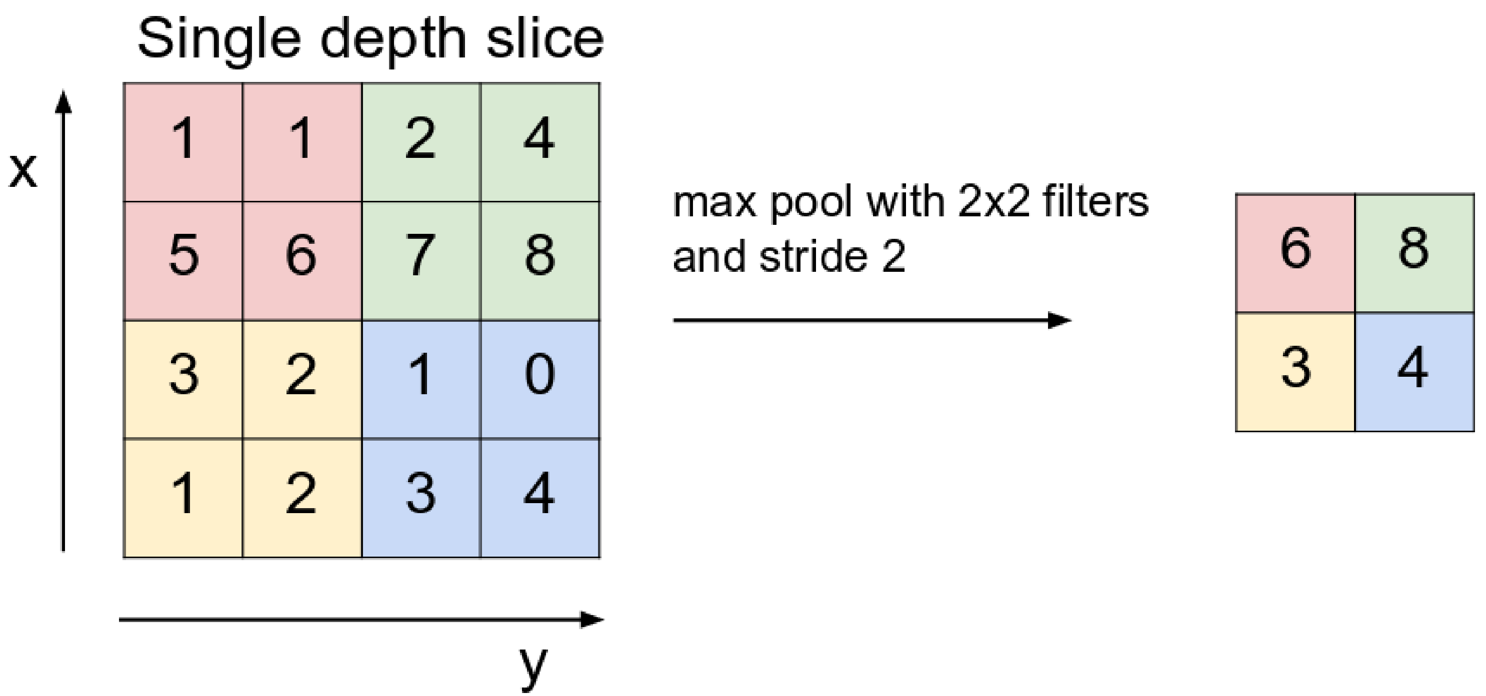
- Replaces the ConvLayer -> ReLU output with a summary statistic of the nearby outputs



[Karpathy, CS231n]

Pooling layers

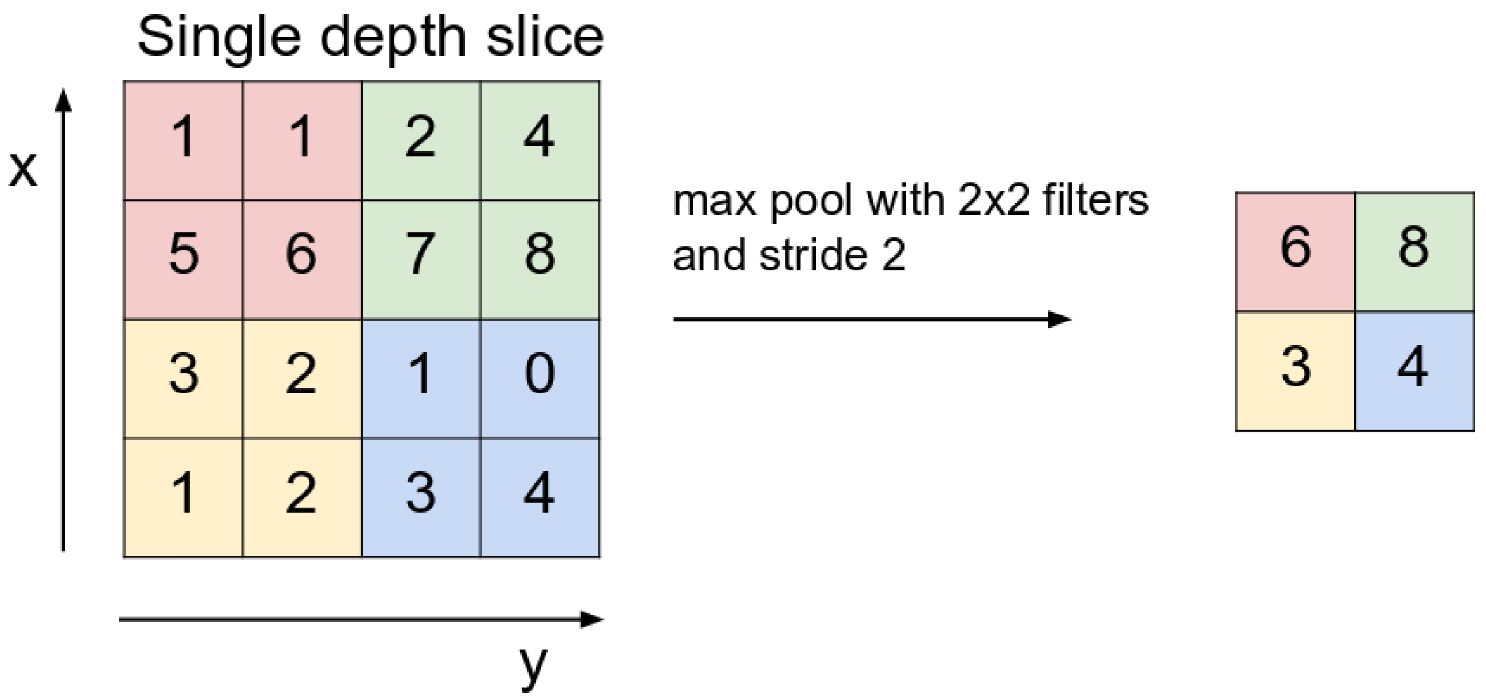
- Max Pooling



[Karpathy, CS231n]

Pooling layers

- Max Pooling
- Eliminates 75% of the feature map



[Karpathy, CS231n]

Pooling layers

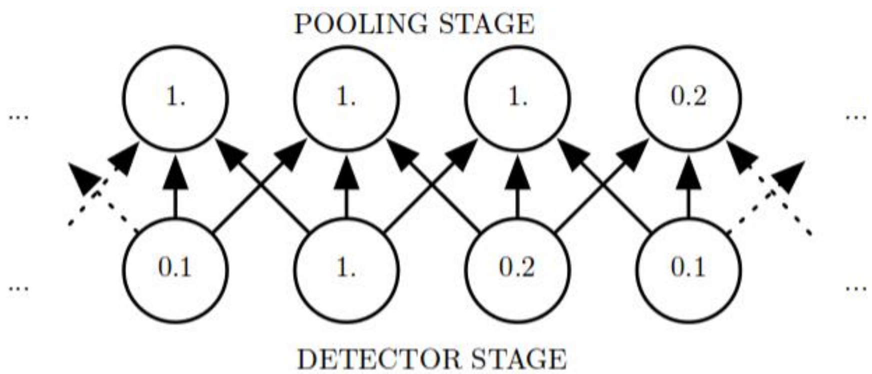
- Max Pooling

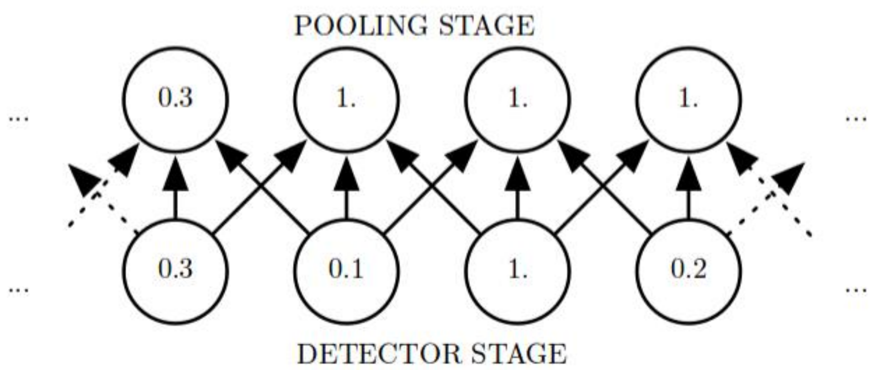
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since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

[Karpathy, CS231n]

Pooling layers

- Pooling helps to make features become approx. **invariant** to small translations of the input





[Goodfellow et al., Deep Learning]

Pooling layers

- Pooling helps to make features become approx. **invariant** to small translations of the input

Why is this interesting?

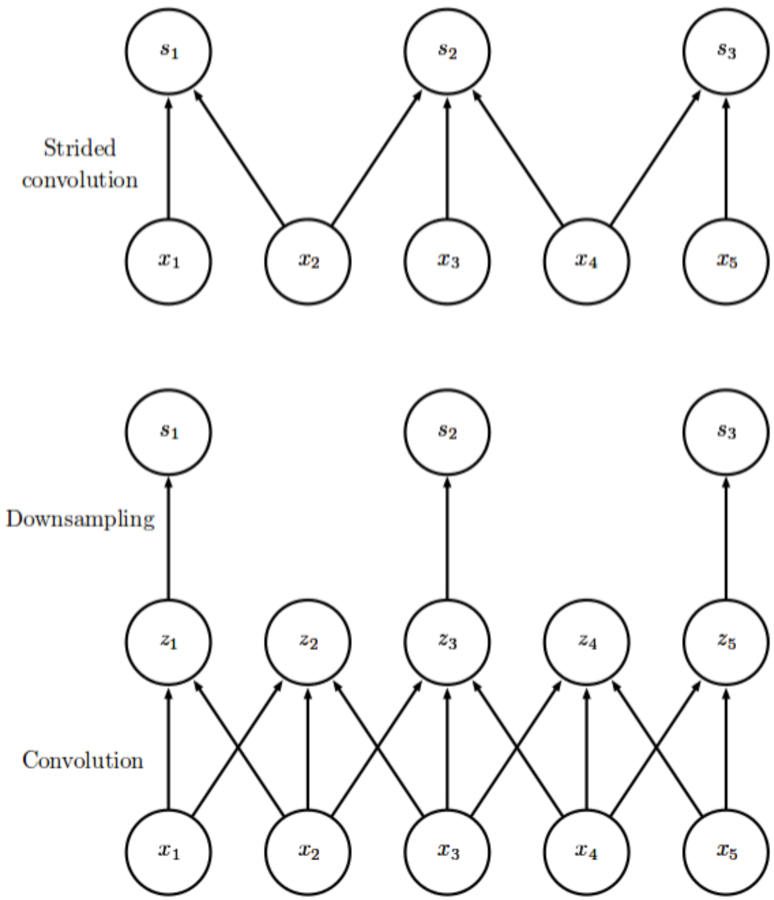
- Can be useful when exact location of a feature is not important
- E.g. in Face detection
- Not necessary to know pixel-perfect location of eyes, nose, mouth, …
- Better: One eye on left side of face, another on right side

Pooling layers

- **Max Pooling**
- Average Pooling
- L2-norm Pooling

- --> However, pooling can also be achieved using strided convolutions!
  - You could and should tese both approaches!

Convolution with Stride

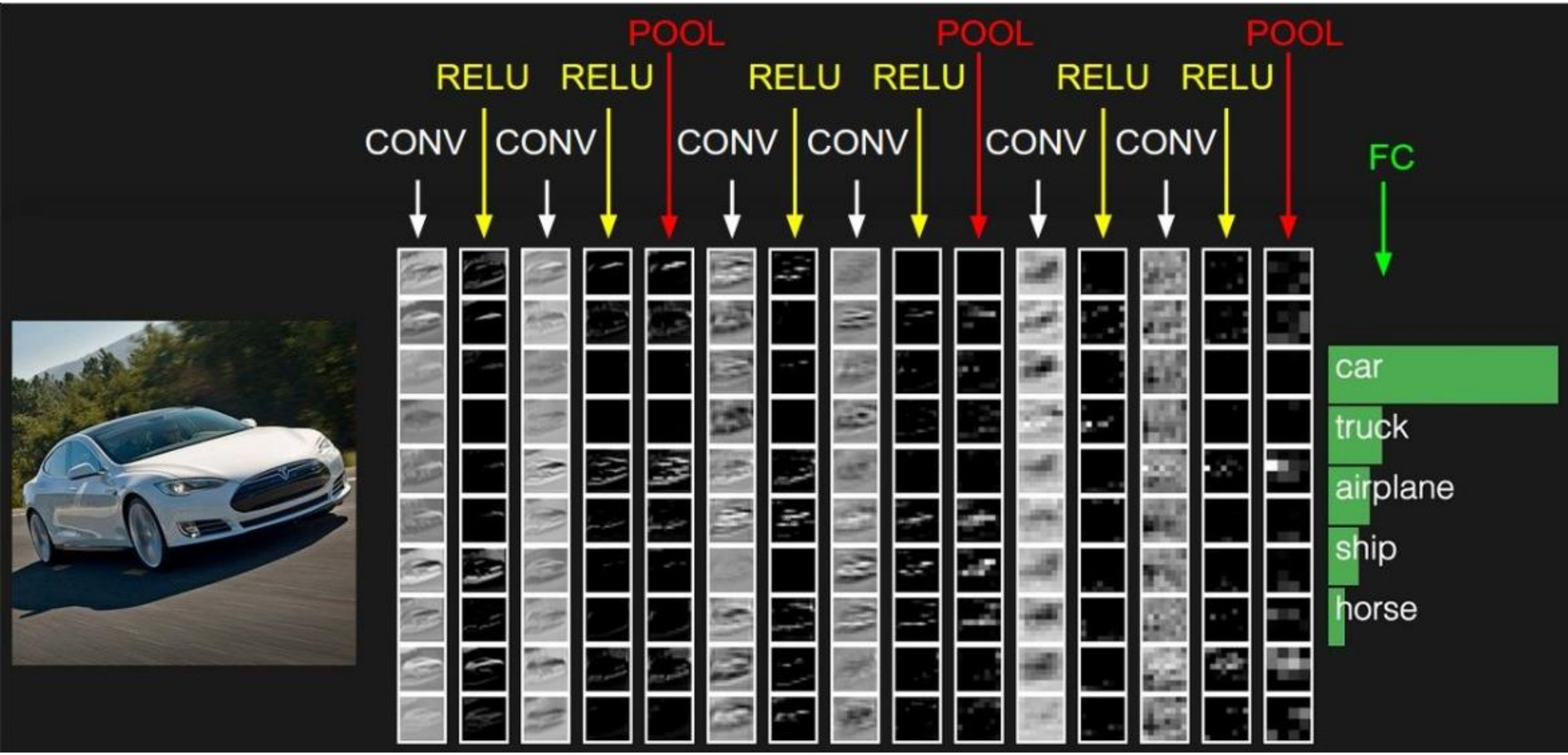


[Goodfellow et al., Deep Learning]

Common architectures

- Stacking of some Conv-ReLU layers
- Followed by a Pooling layer
- Repeat until Feature map has small size
- Add Fully-Connected layer
- Last Fully-connected layer holds outputs (scores)

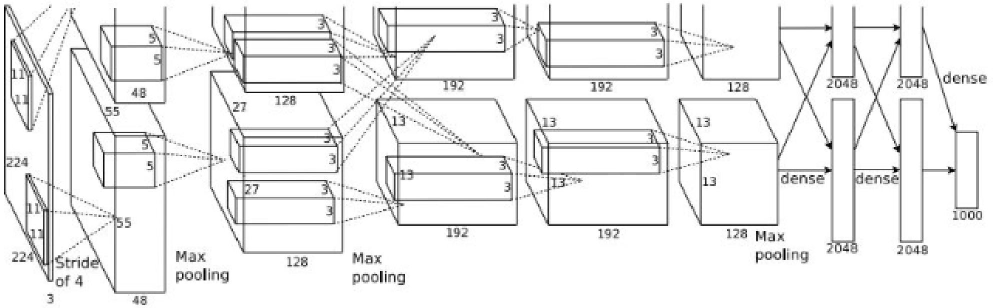
Convolutional Neural Networks



[Karpathy, CS231n]

Case Study: AlexNet

[Krizhevsky et al. 2012]



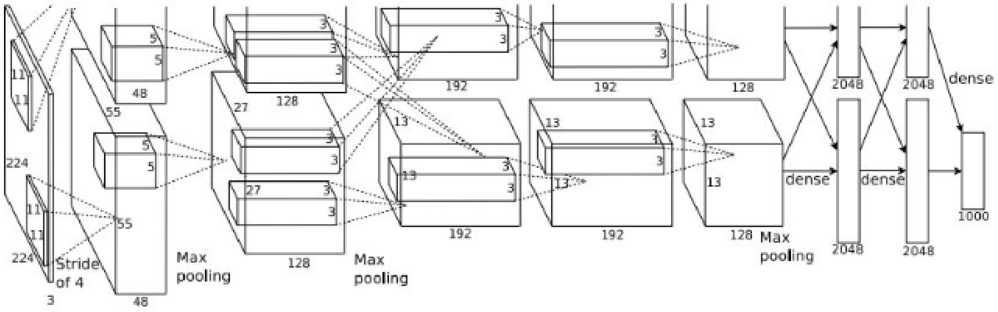
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

[Karpathy, CS231n]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

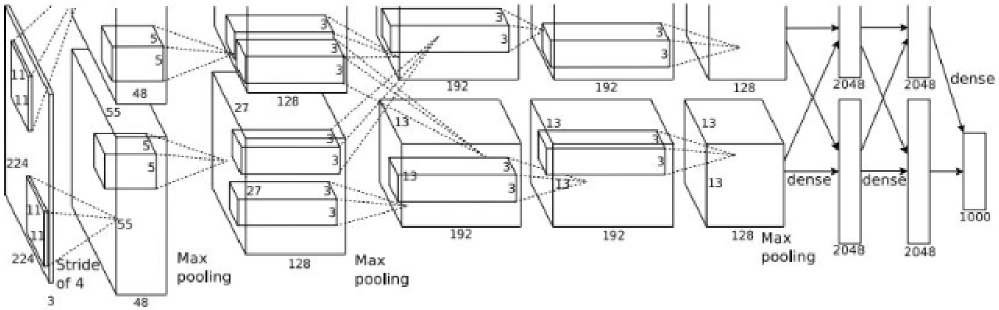
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

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

[Karpathy, CS231n]

Case Study: AlexNet

[Krizhevsky et al. 2012]



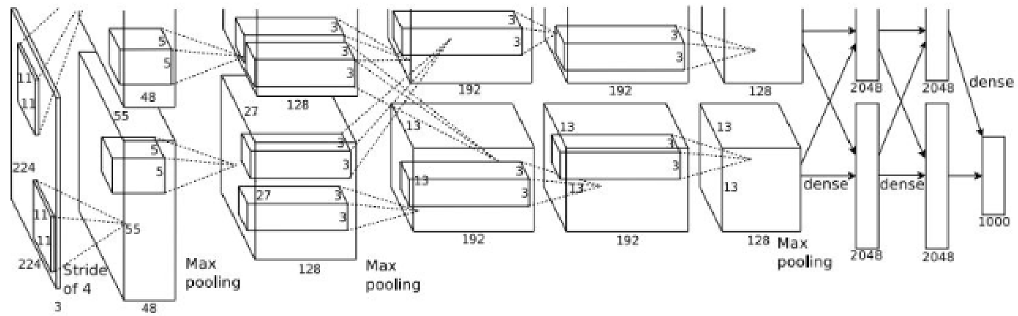
Input: 227x227x3 images

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

[Karpathy, CS231n]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

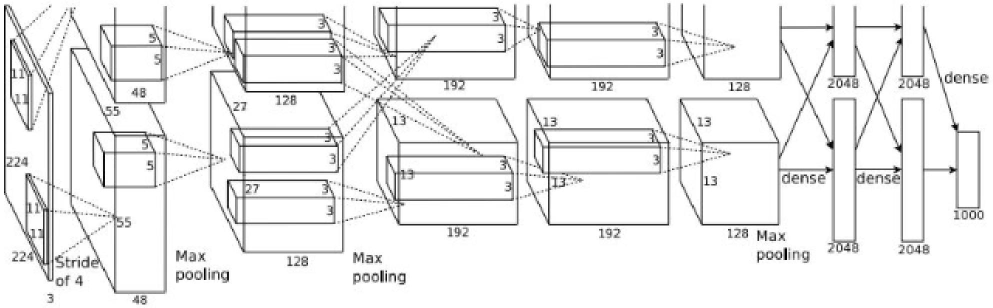
Second layer (POOL1): 3x3 filters applied at stride 2

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

[Karpathy, CS231n]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

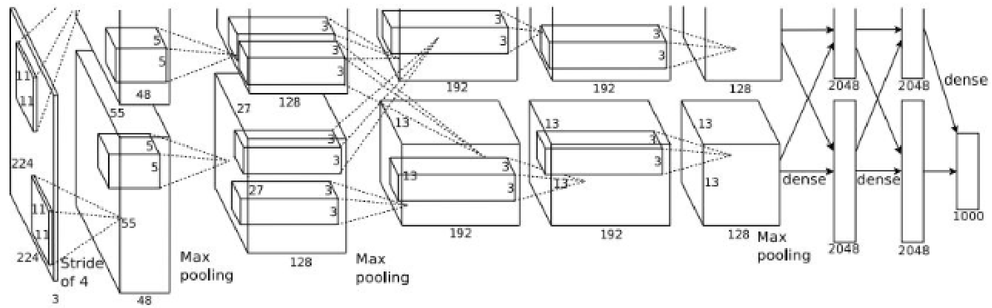
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

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

[Karpathy, CS231n]

Case Study: AlexNet

[Krizhevsky et al. 2012]



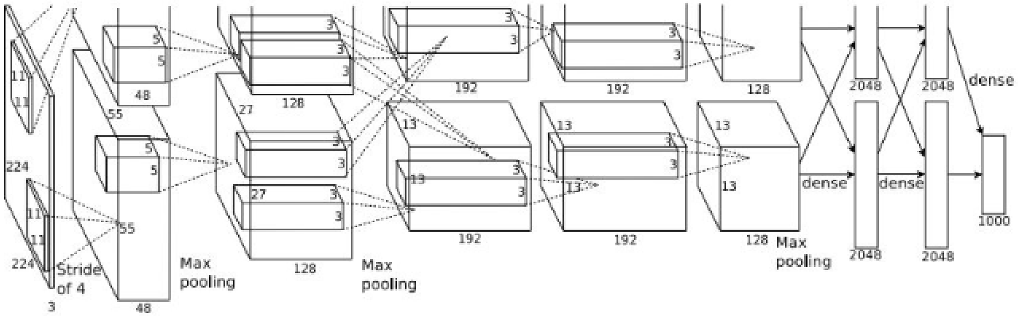
Input: 227x227x3 images
After CONV1: 55x55x96

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

[Karpathy, CS231n]

Case Study: 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)

[Karpathy, CS231n]

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

[Karpathy, CS231n]

Case Study

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

[Karpathy, CS231n]

Case Study

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	co co
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

[Karpathy, CS231n]

Case Study

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

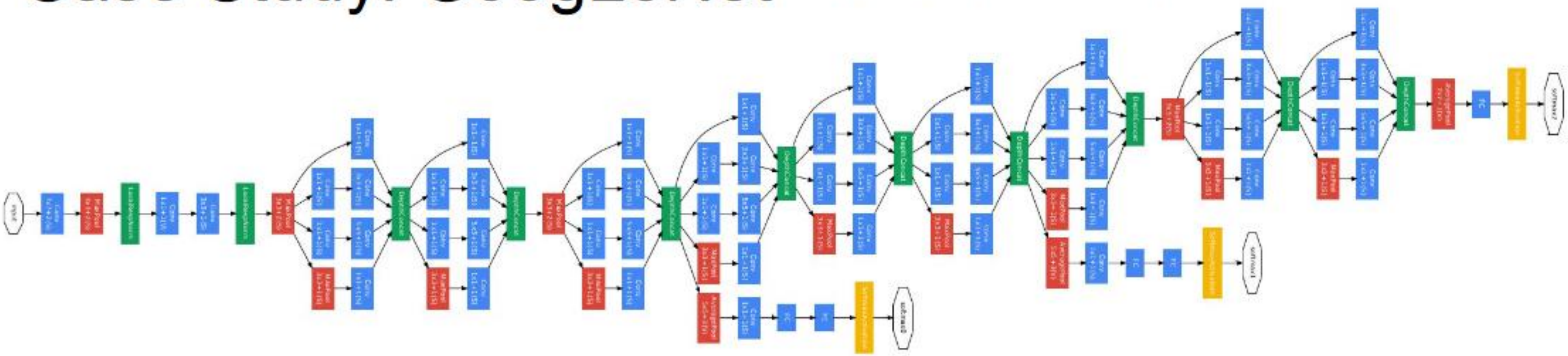
Most params are in late FC

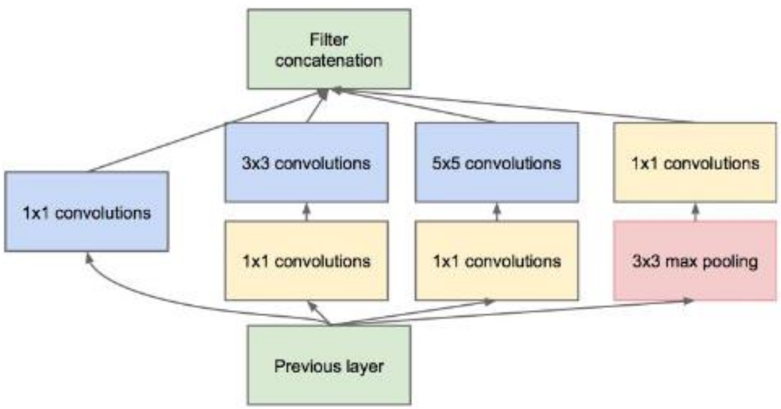
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

[Karpathy, CS231n]

Case Study: GoogLeNet

[Szegedy et al., 2014]





Inception module

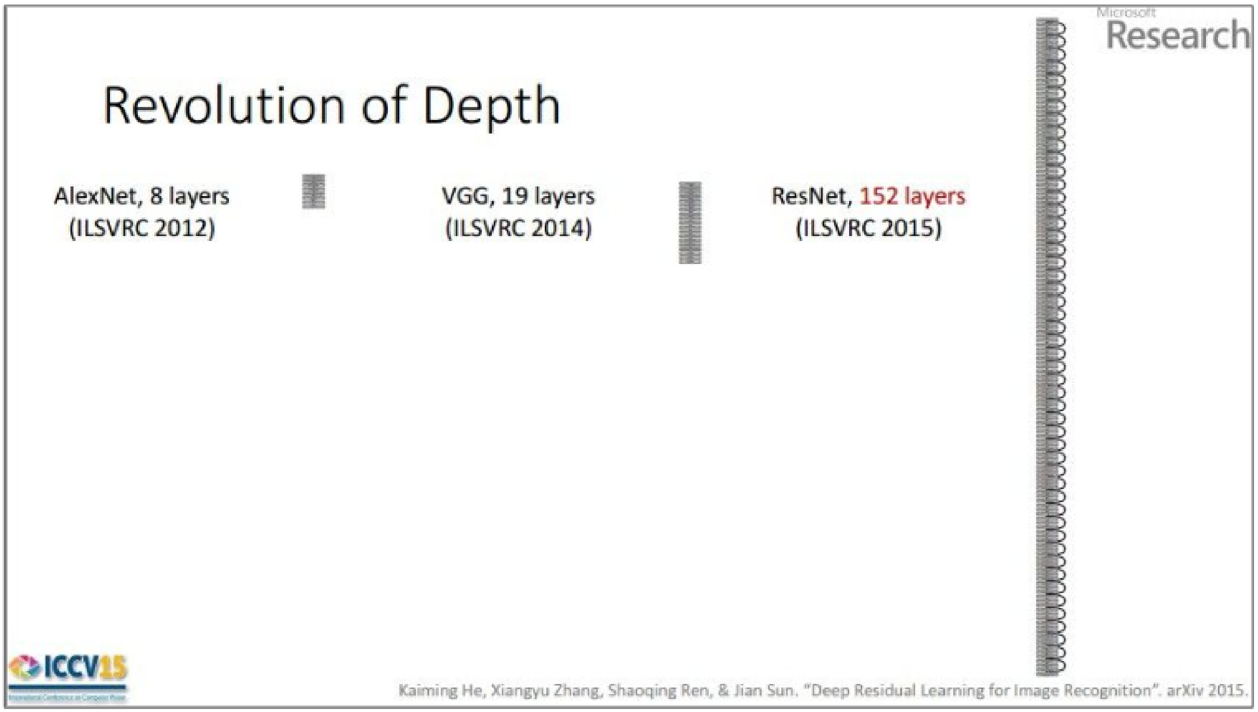
ILSVRC 2014 winner (6.7% top 5 error)

[Karpathy, CS231n]

Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



(slide from Kaiming He's recent presentation)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

[Karpathy, CS231n]

Thank you!

Any questions?