Computer Vision
Prof. Dr. Songül Varlı

DATASETS- CIFAR10 AND MNIST

CONVOLUTION LAYER

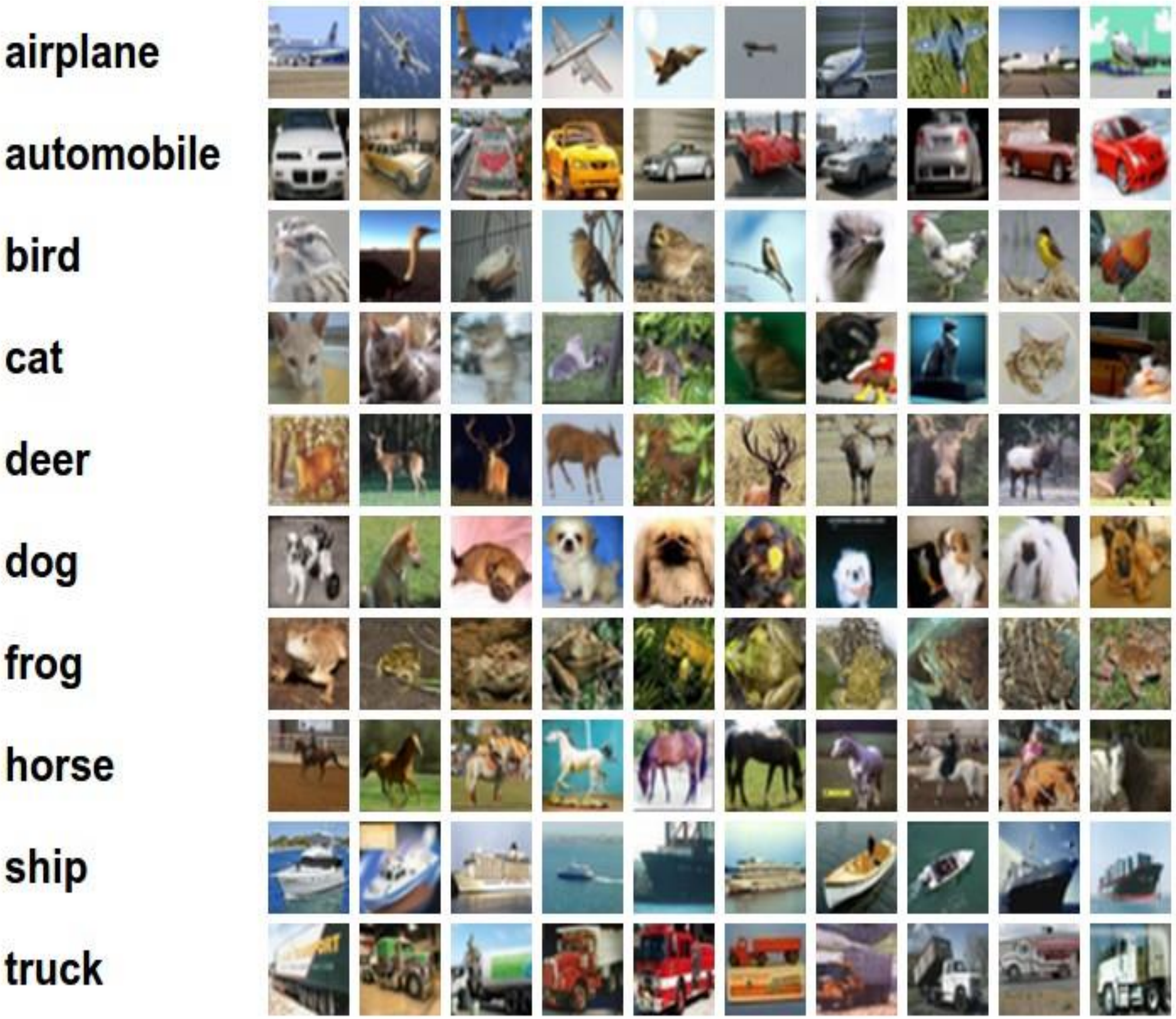
MAXPOOLING

ACTIVATION FUNCTIONS

FULLY CONNECTED LAYER

SOFTMAX OUTPUT

Dataset: CIFAR-10

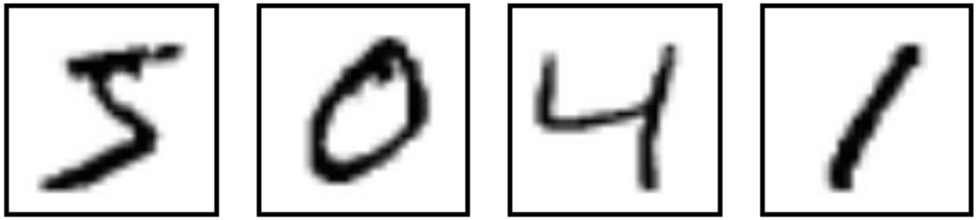


CIFAR-10
10 labels
50,000 training images
each image is **32x32x3**
10,000 test images.

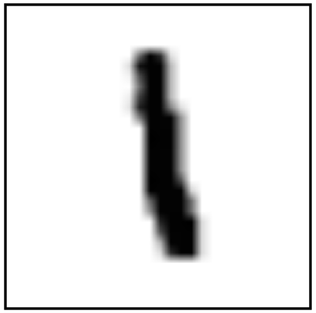
“Programming has Hello World,
machine learning has MNIST”

MNIST Dataset

(Modified National Institute of Standards and Technology database)

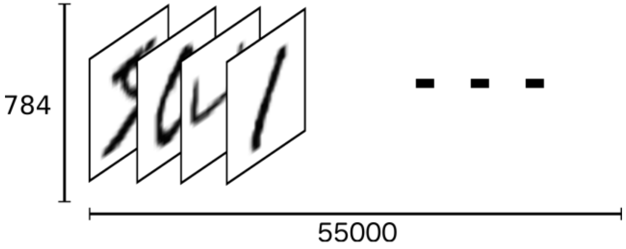


Features: 28× 28 matrix



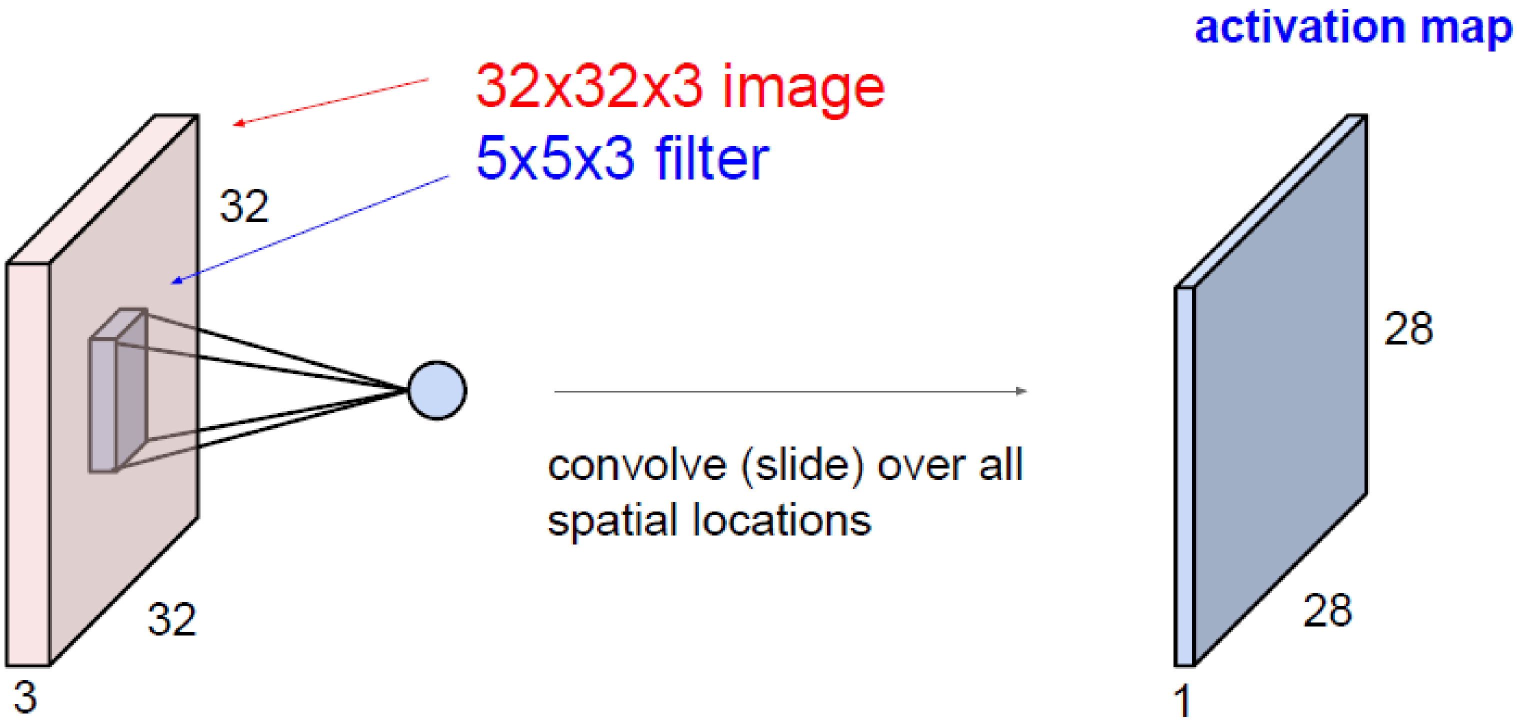
$$\cong \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

784-dim vector$_{\text{mnist.train.xs}}$



Flatten ⟹

Convolution Layer



If we have 6 5x5 filters, we'll get 6 separate activation map. We stack these up to get a new image of size 28x28x6

Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

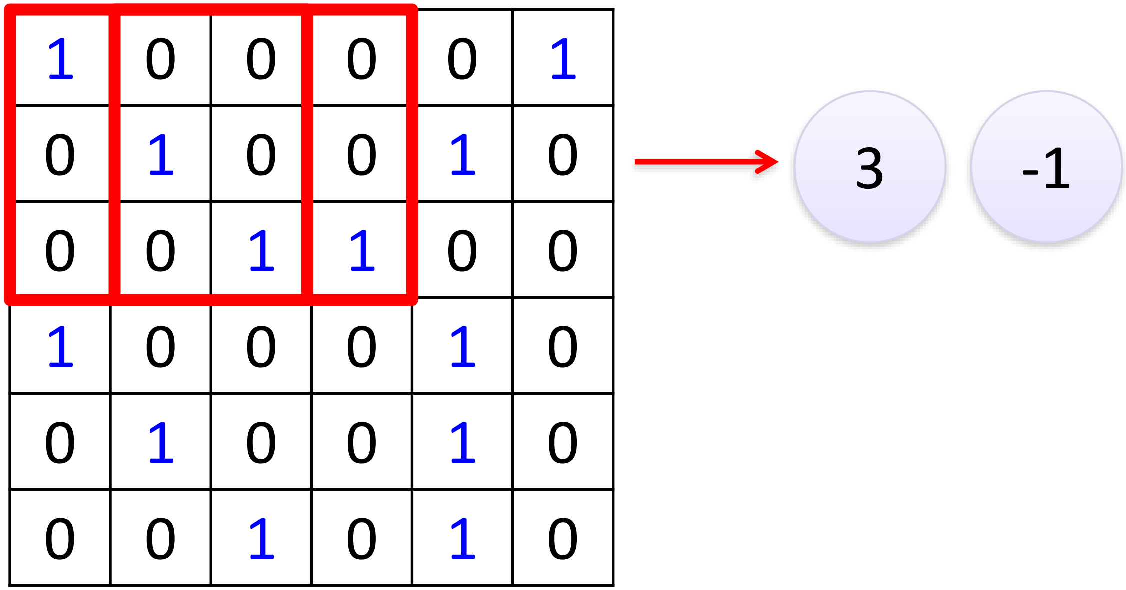
Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

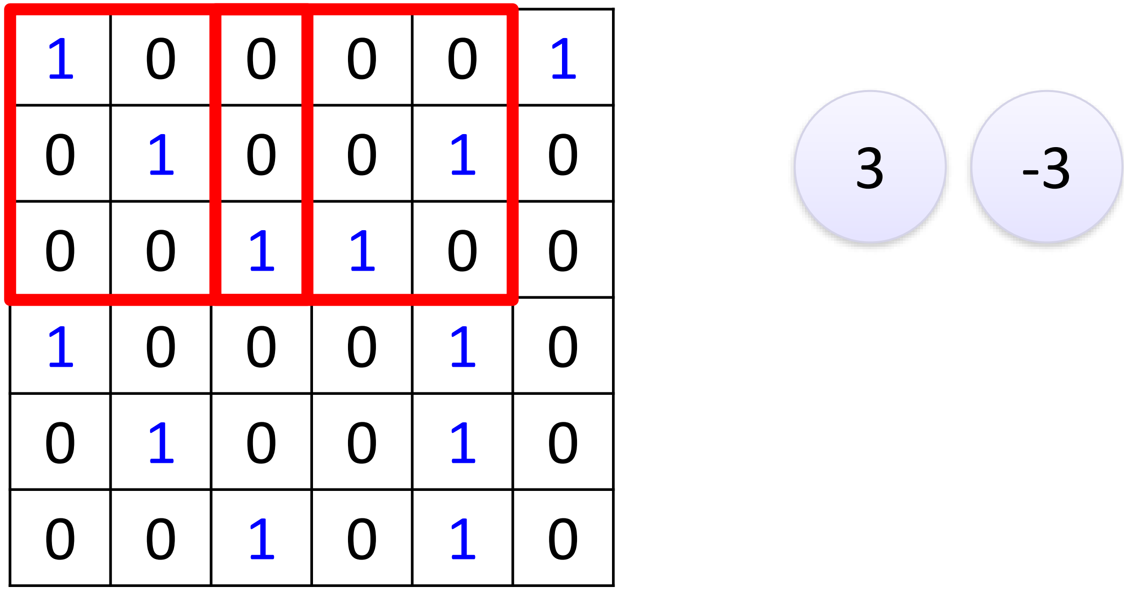
6 x 6 image

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

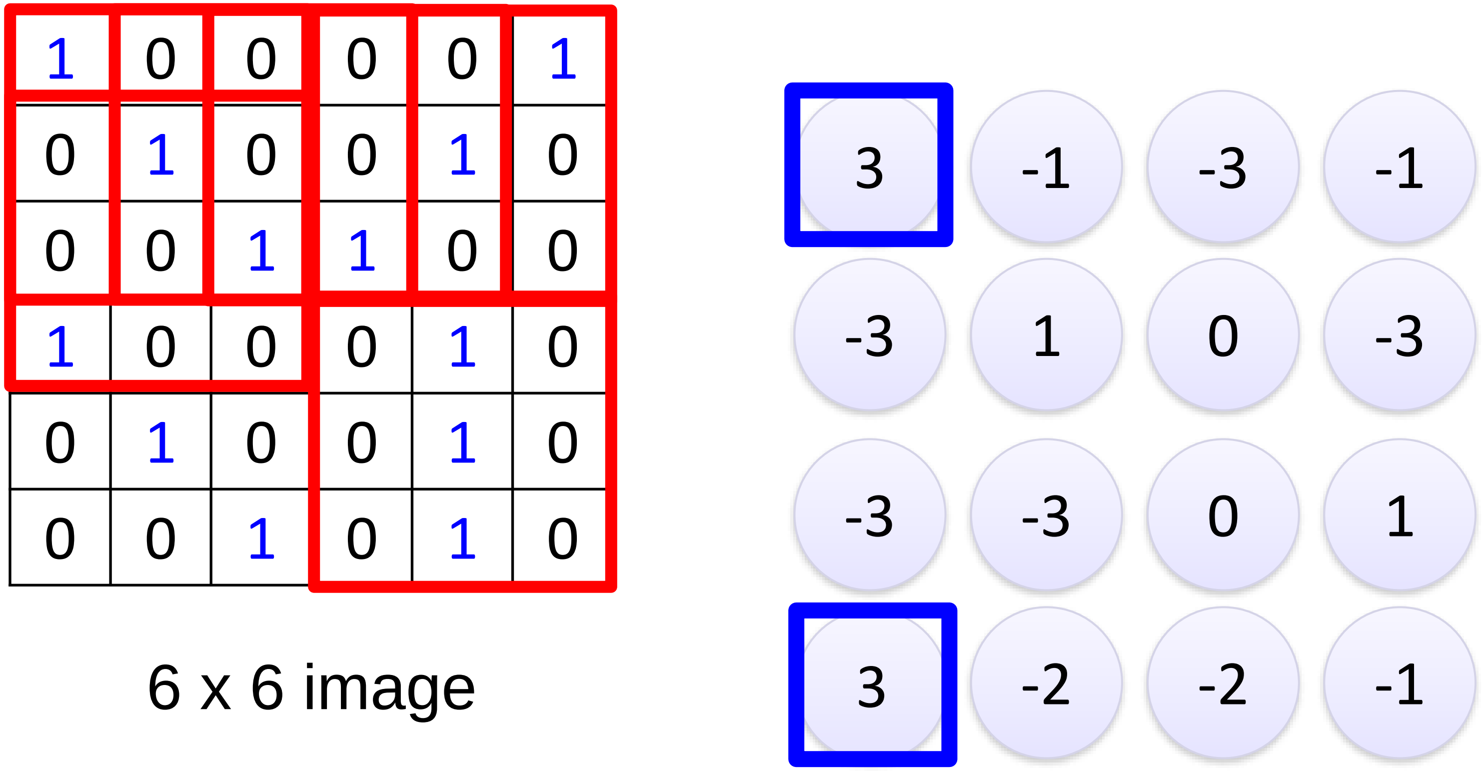
6 x 6 image

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

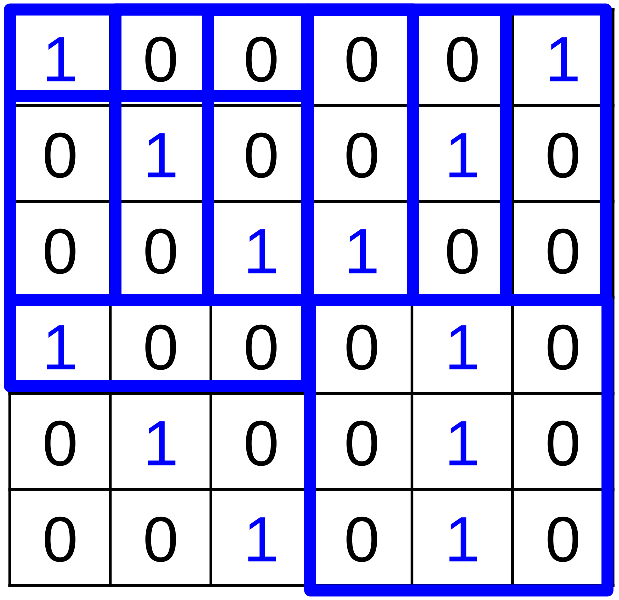
Convolution

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

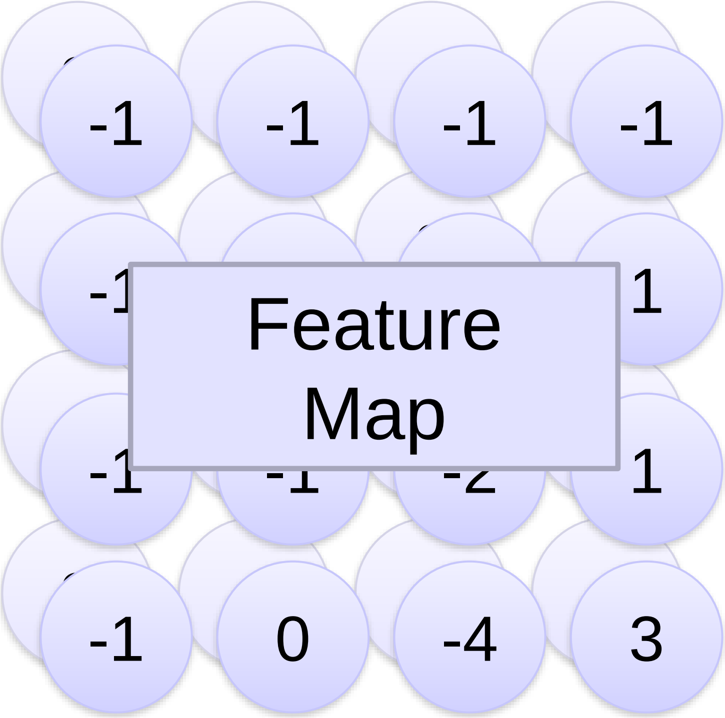
stride=1

Repeat this for each filter



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



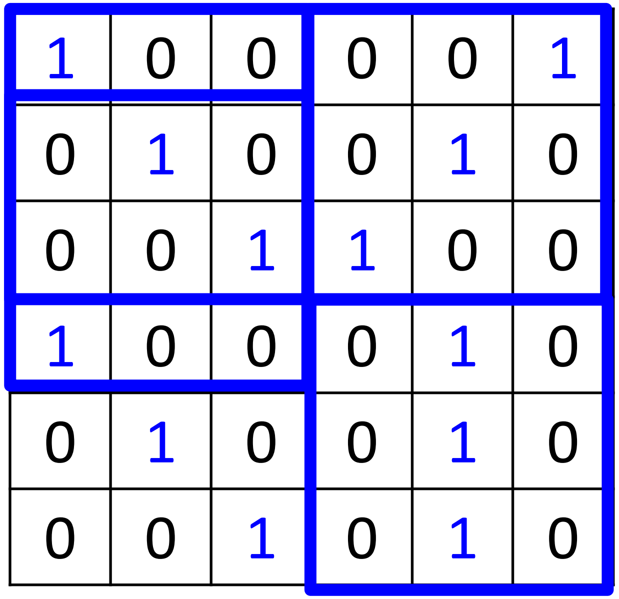
Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

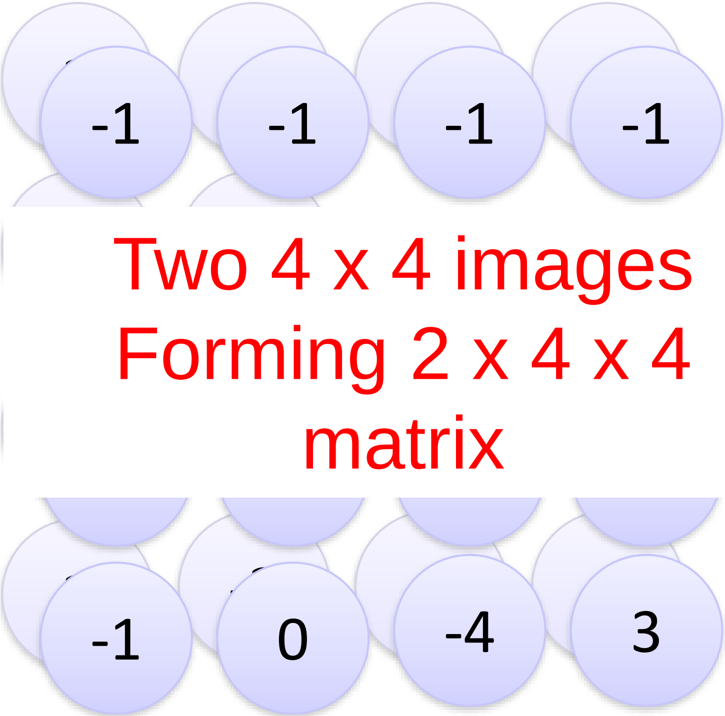
Filter 2

stride=1



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4
matrix

Convolution
Color image: RGB 3 channels

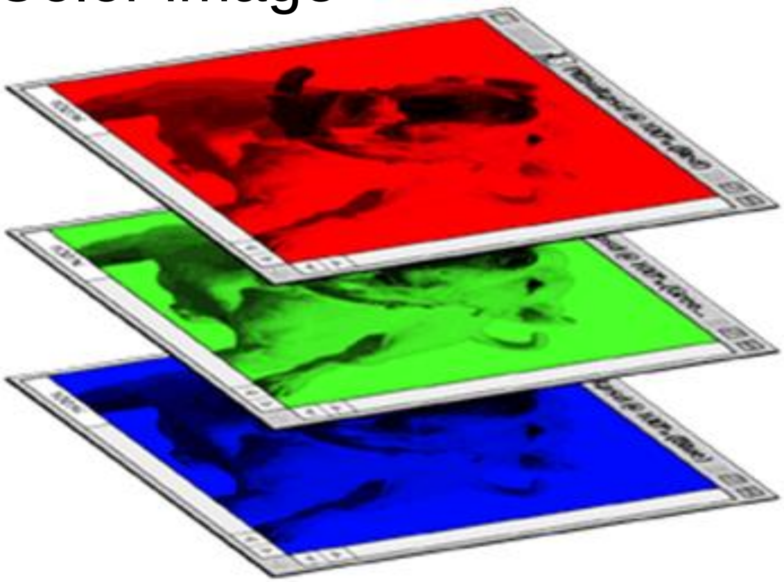
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

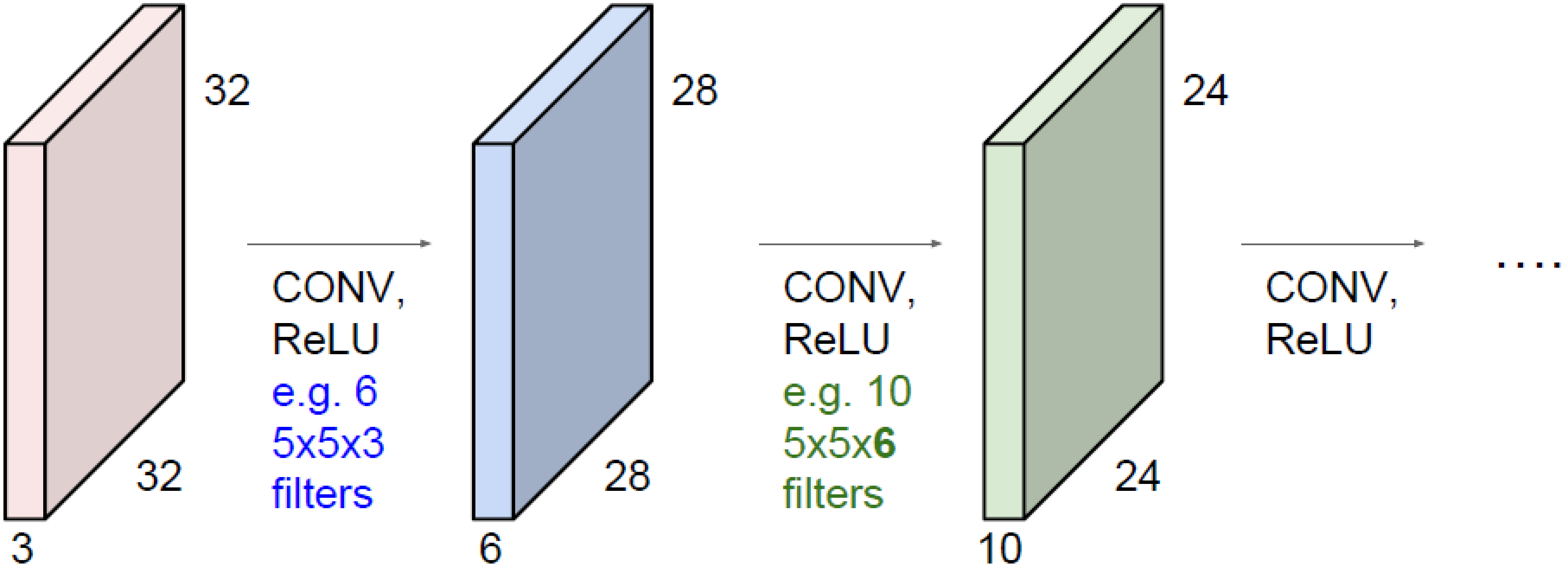
Filter 2

Color image



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



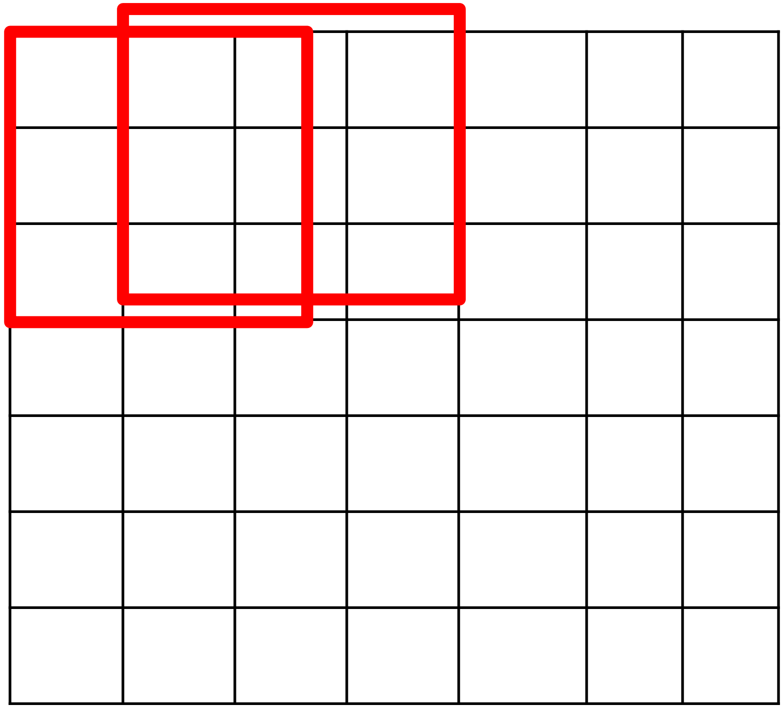
Low level features

Mid level features

High level features

A closer look at convolution operation:

Stride=1

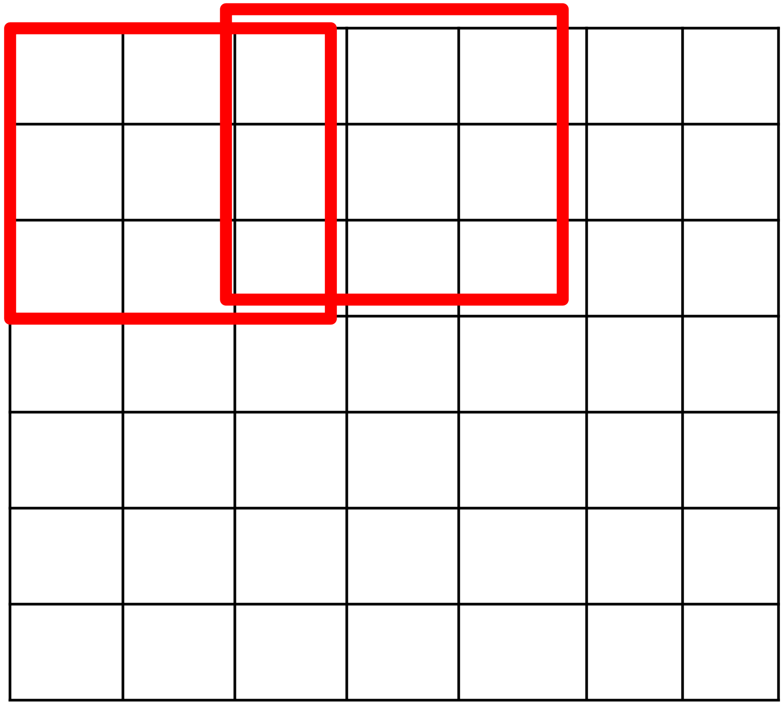


N=input image size
7x7 input image

F=filter size
3x3 filter

Output will be 5x5

Stride=2



N=input image size
7x7 input image

F=Filter Size
3x3 filter

Output will be 3x3

Output Size of convolution operation?

e.g. N=7 and F=3

For Stride=1 , $\text{Output} = \frac{(7-3)}{1} + 1 = 5$

For Stride=2 , $\text{Output} = \frac{(7-3)}{2} + 1 = 3$

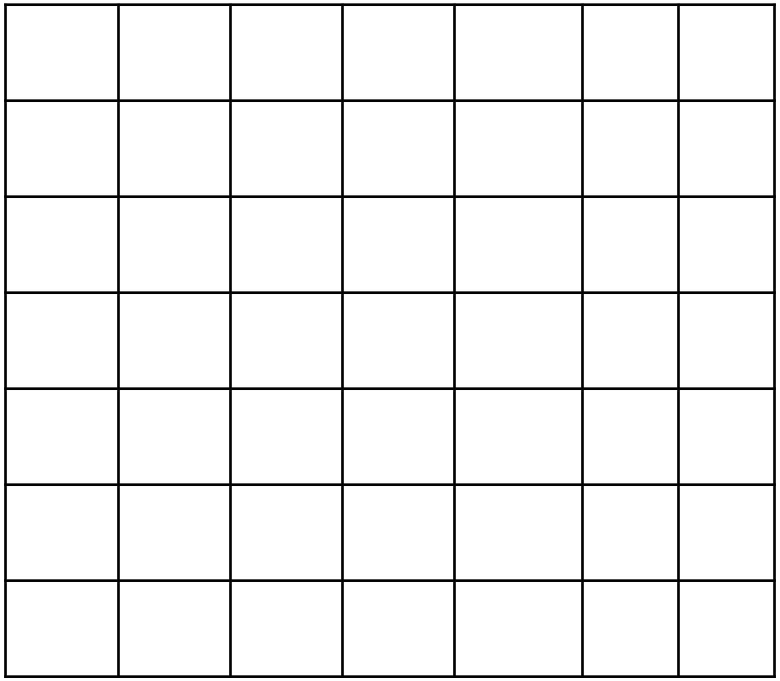
For Stride=3 , $\text{Output} = \frac{(7-3)}{3} + 1 = 2{,}33$ (Does not fit)

$$Output\ Image\ Size = \frac{N - F}{Stride} + 1$$

N: Input Image Size, F:Filter Size

In practice: Common to zero pad the border

P=1 (Zero padding with one)



N=input image size
7x7 input image

After zero padding with one
Input image will be 9x9x

F=Filter size
3x3 filter

Output will be 7x7
(Same size with input image)

Filters of size F x F and zero padding (F-1) / 2 will preserve the output image size

e.g.

F=3 , Zero padding with 1

F=5 , Zero padding with 2

Exercise:

Input image volume= 32x32x3

10 Filters with the size of 5x5x3 with stride 1, pad 2

Output volume size=???

Exercise:

Input image volume= 32x32x3

10 Filters with the size of 5x5x3 with stride 1, pad 2

Output volume size= 32x32x10

Exercise:

Input image volume= 32x32x3

10 Filters with the size of 5x5x3 with stride 1, pad 2

Number of parameters in this layer=???

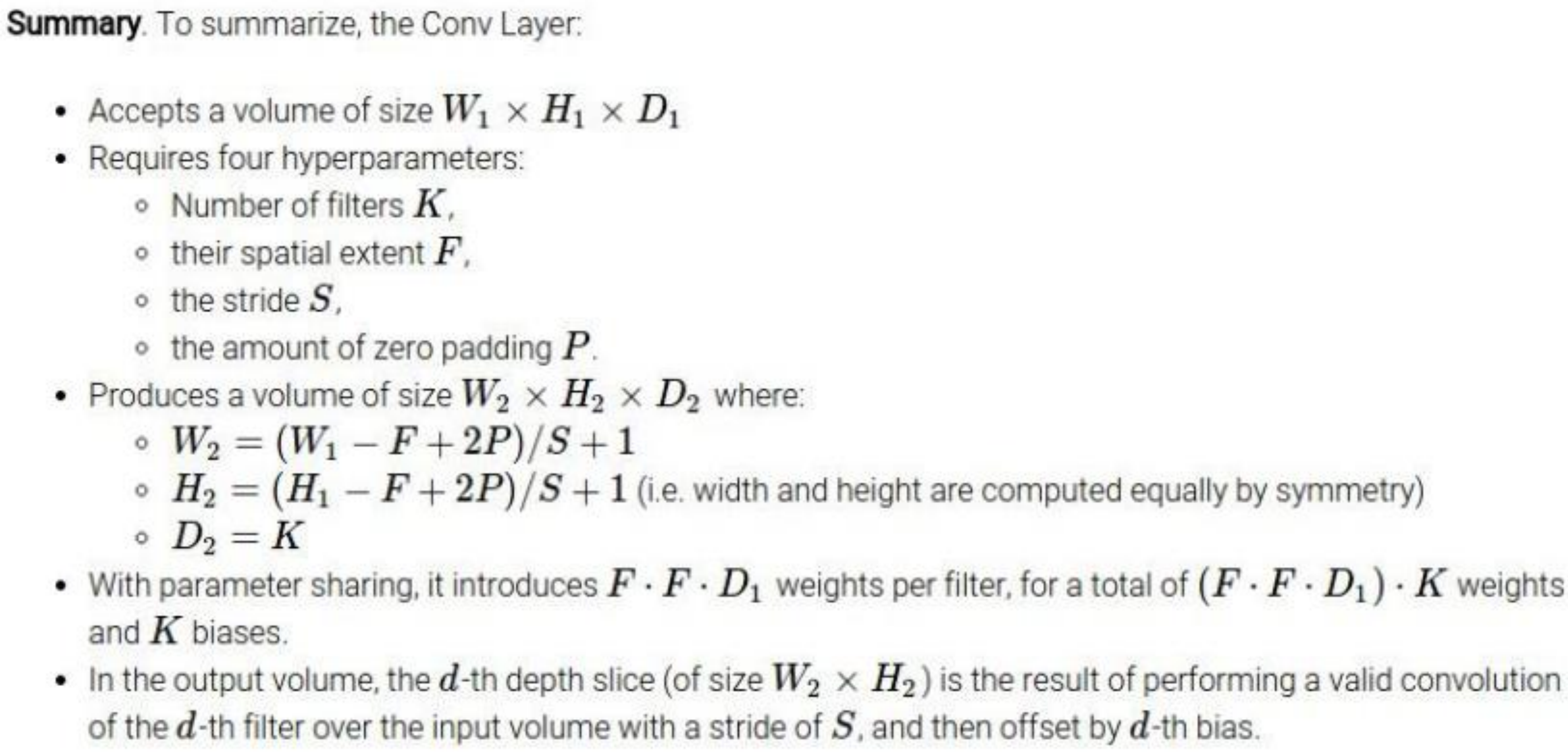
Exercise:

Input image volume= 32x32x3

10 Filters with the size of 5x5x3 with stride 1, pad 2

Each filter has 5x5x3=75 +1 (+1 bias parameter)

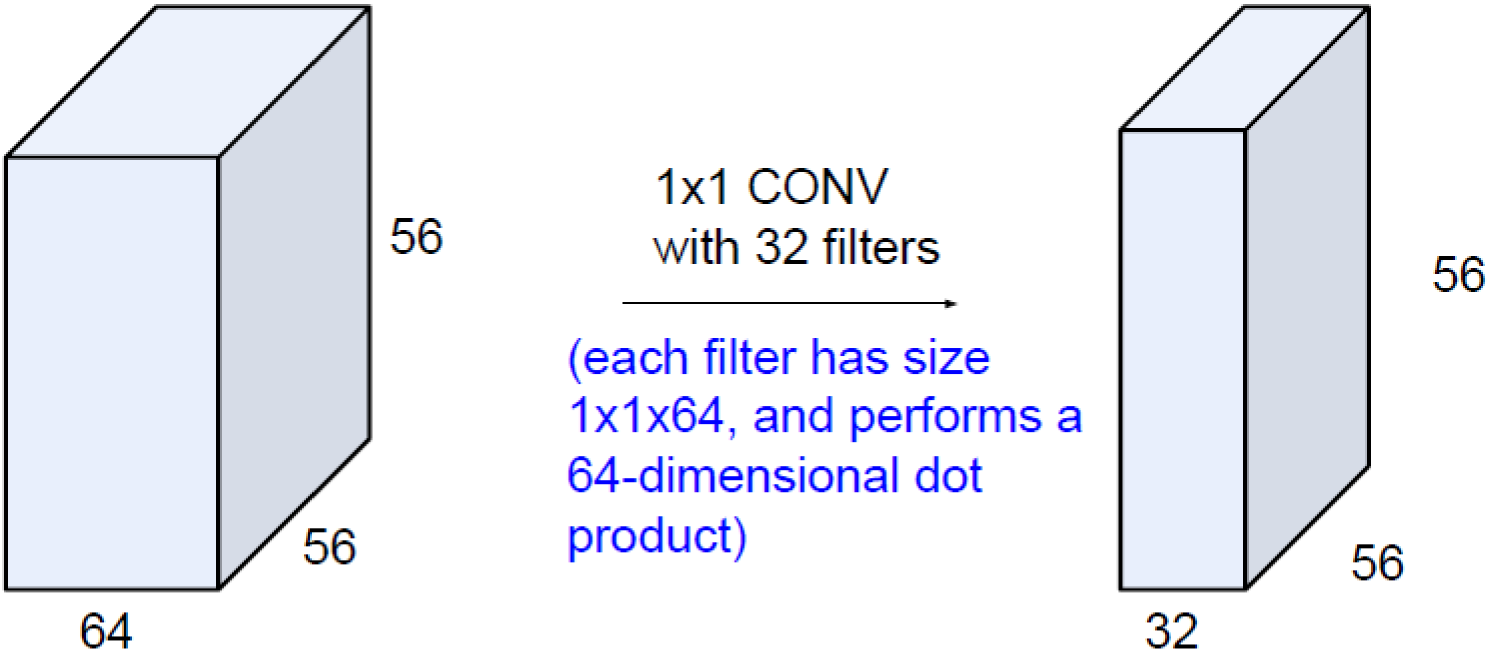
Number of parameters in this layer= 10x76=760 parameters



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.

(btw, 1x1 convolution layers make perfect sense)

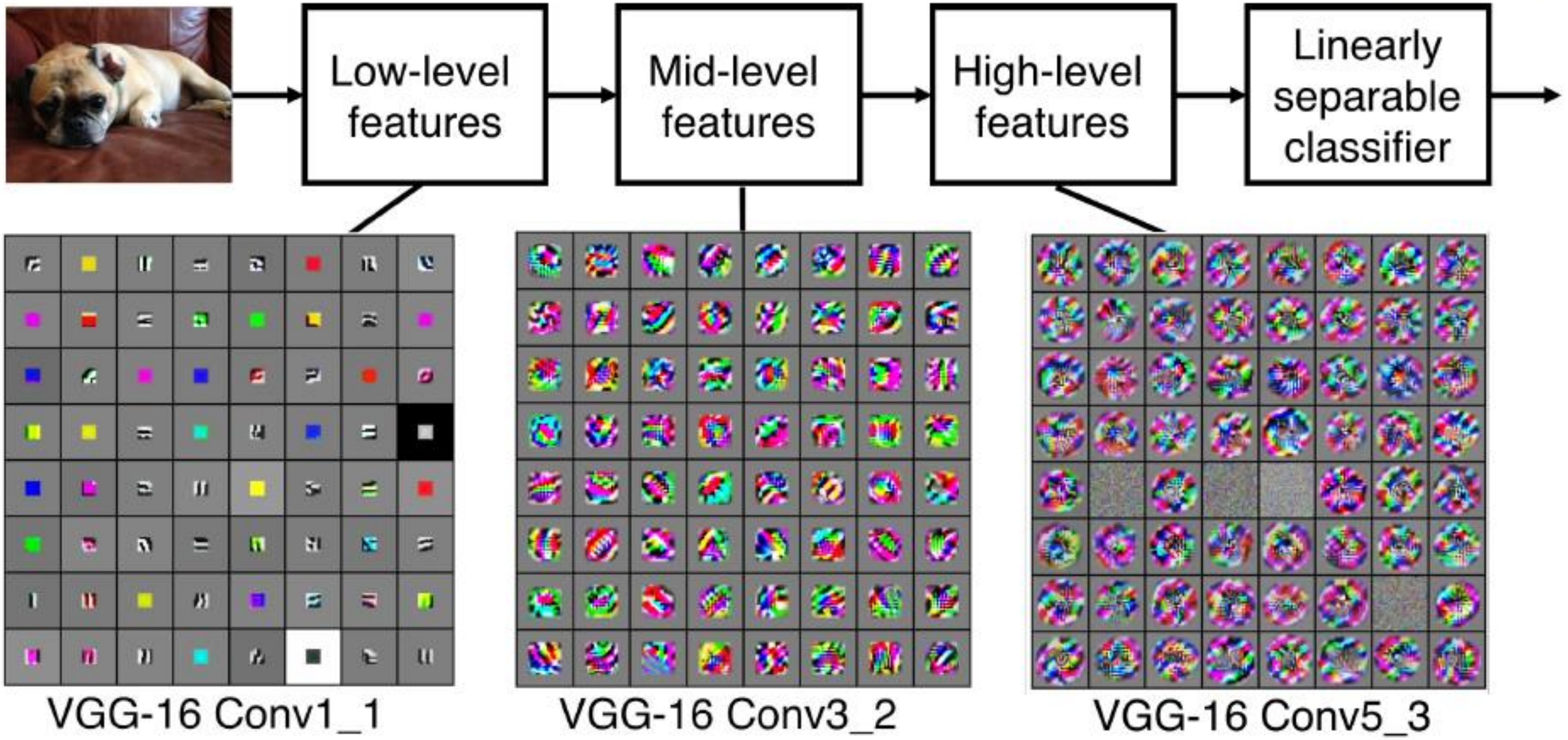


(each filter has size 1x1x64, and performs a 64-dimensional dot product)

Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



Low-level features

Mid-level features

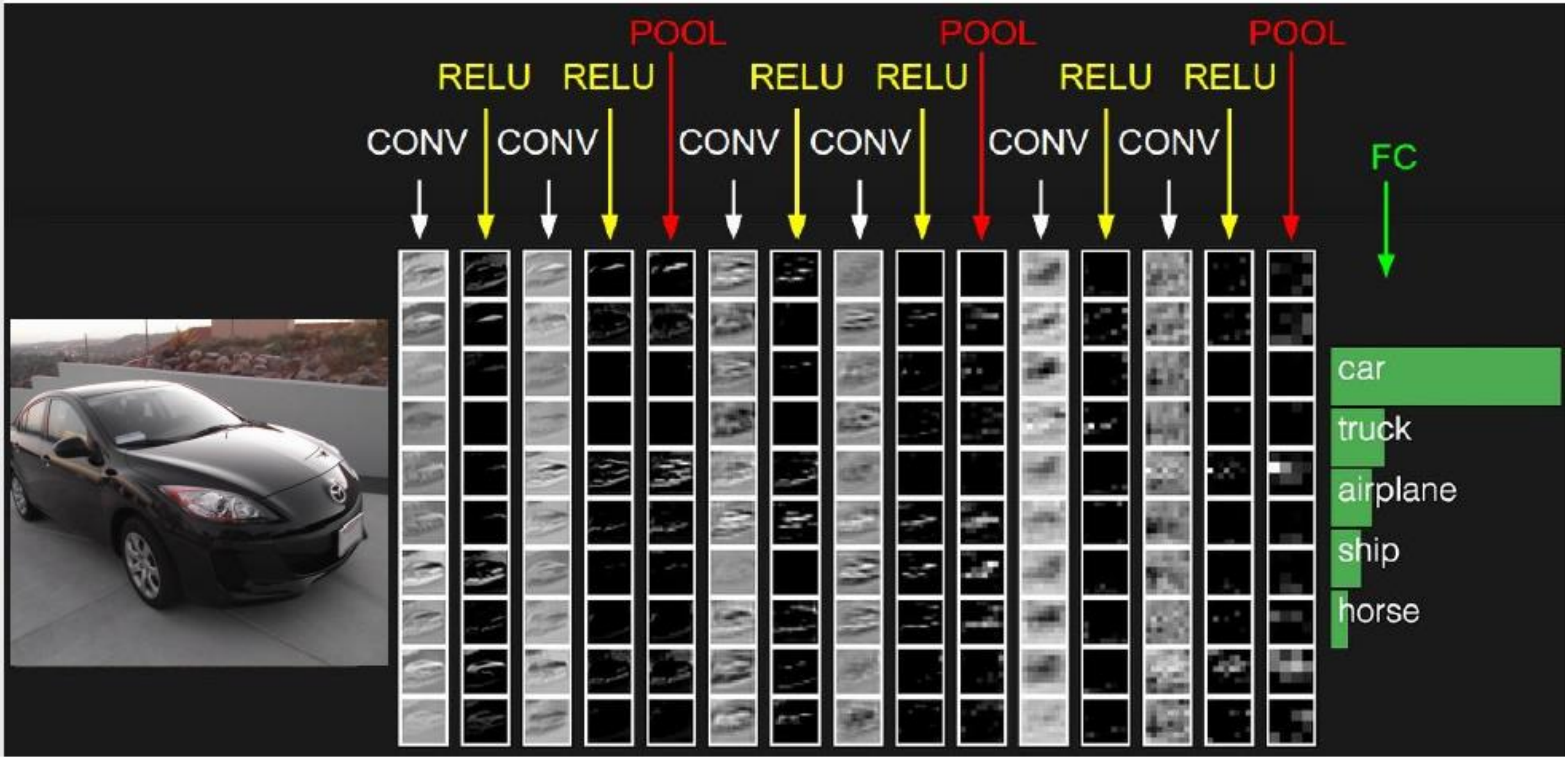
High-level features

Linearly separable classifier

VGG-16 Conv1_1

VGG-16 Conv3_2

VGG-16 Conv5_3

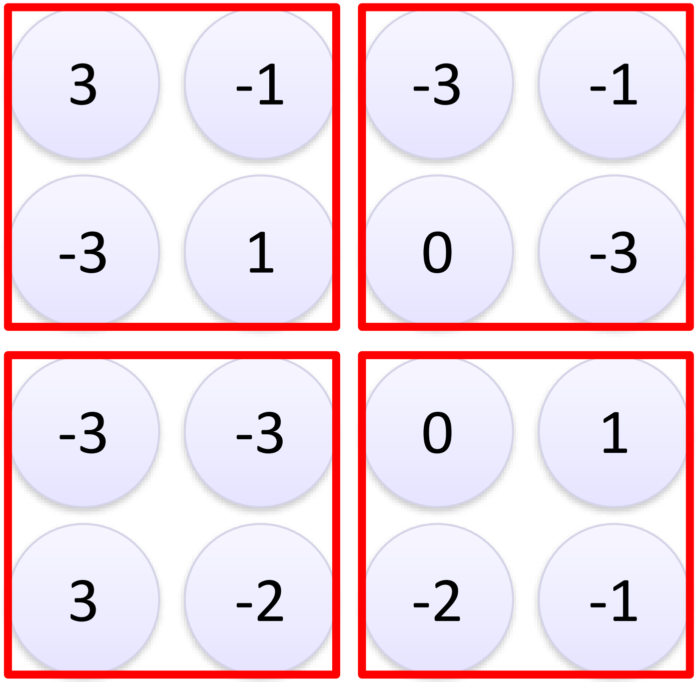

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Max Pooling

- Makes the representation smaller and more manageble
- Operates over each activation map independently

1	-1	-1
-1	1	-1
-1	-1	1

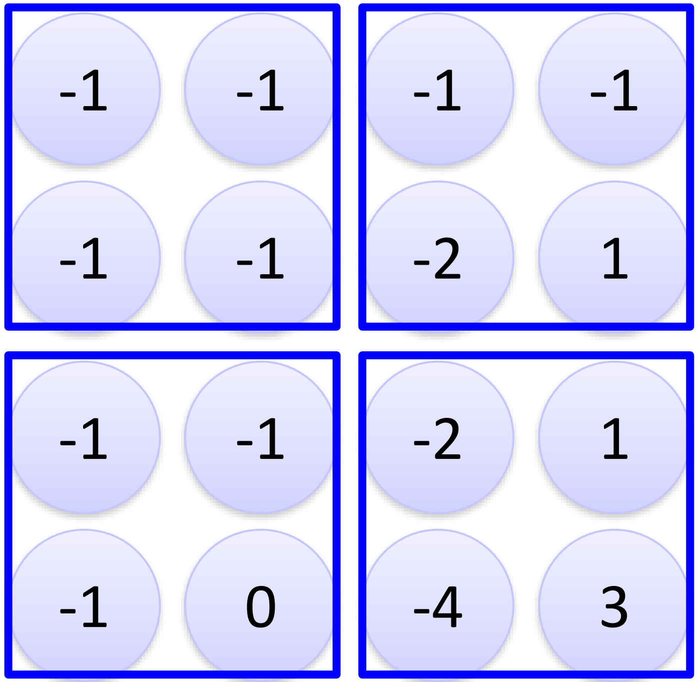
Filter 1



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

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

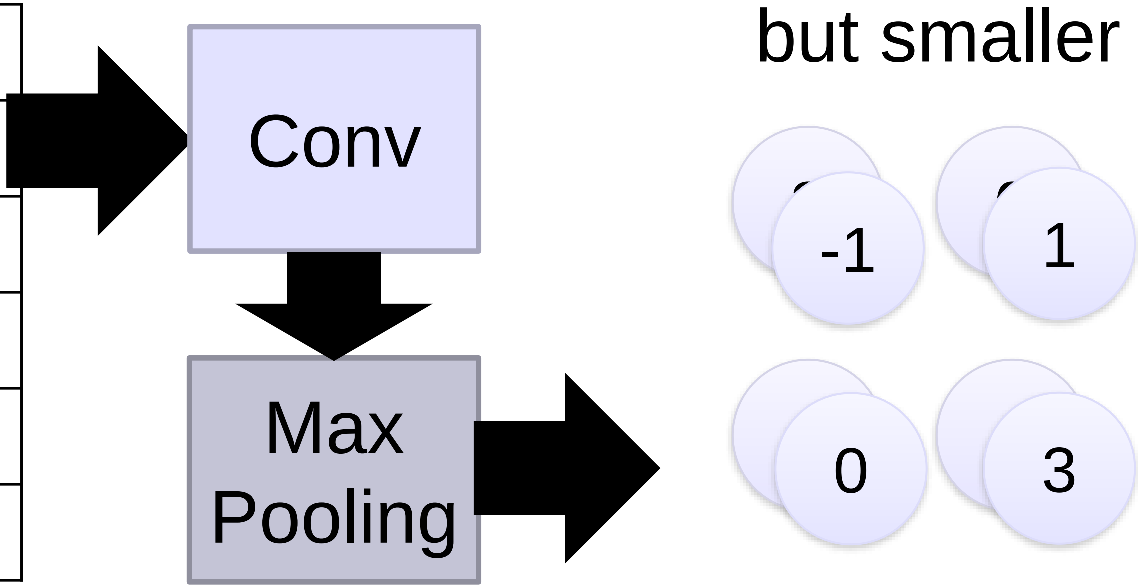
We can subsample the pixels to make image smaller

→ fewer parameters to characterize the image

Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



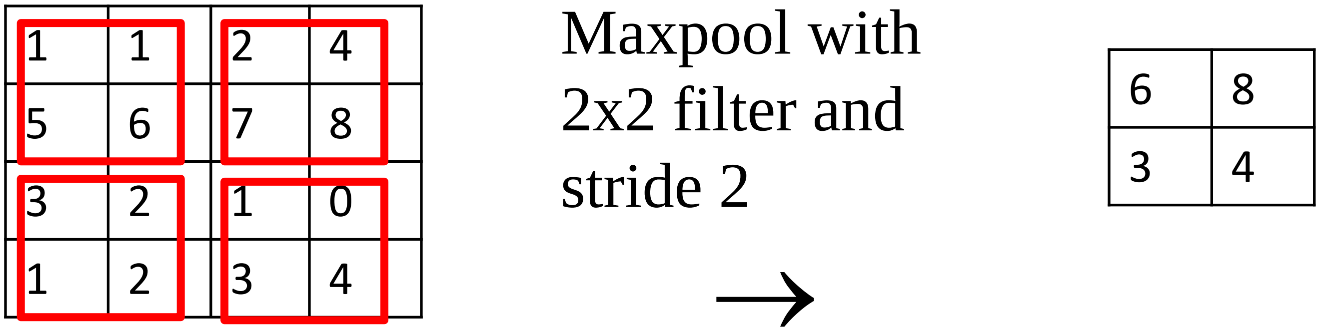
New image but smaller

2 x 2 image

Each filter is a channel

Max Pooling

- Makes the representation smaller and more manageble
- Operates over each activation map independently



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Maxpool with 2x2 filter and stride 2

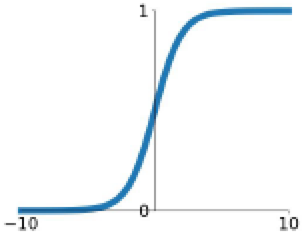
→

6	8
3	4

Activation Functions

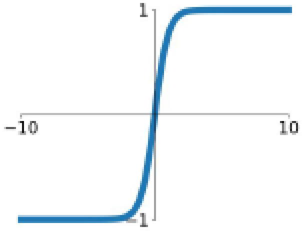
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



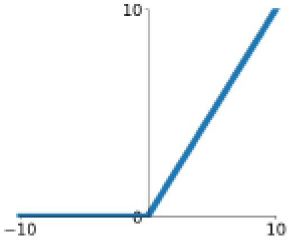
tanh

$\tanh(x)$



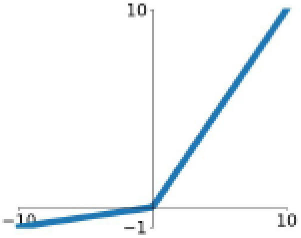
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

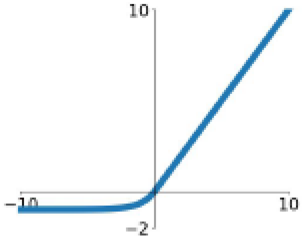


Maxout

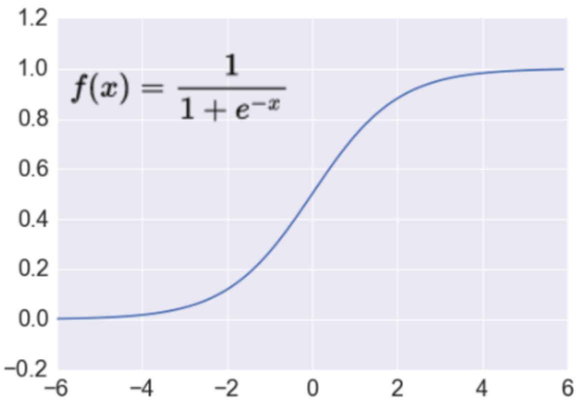
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Function: Sigmoid



Takes a real-valued number and "squashes" it into range between 0 and 1.

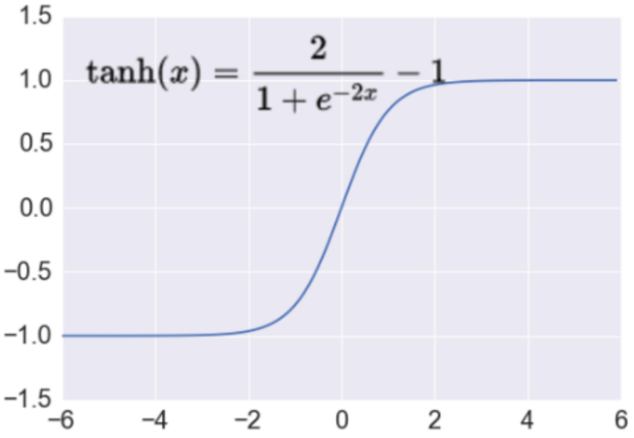
\+ Nice interpretation as the **firing rate** of a neuron

- 0 = not firing at all
- 1 = fully firing

\- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn

- when the neuron's activation are 0 or 1 (saturate)
  - ☹ gradient at these regions almost zero
  - ☹ almost no signal will flow to its weights
  - ☹ if initial weights are too large then most neurons would saturate

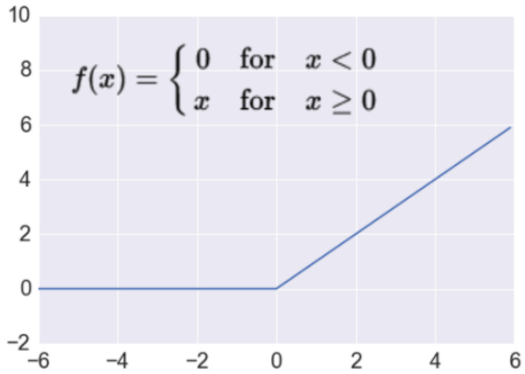
Activation Function: tanh



Takes a real-valued number and "squashes" it into range between -1 and 1.

- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation Function :ReLu
Rectified Linear Unit



$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

Takes a real-valued number and thresholds it at zero

Most Deep Networks use ReLU nowadays

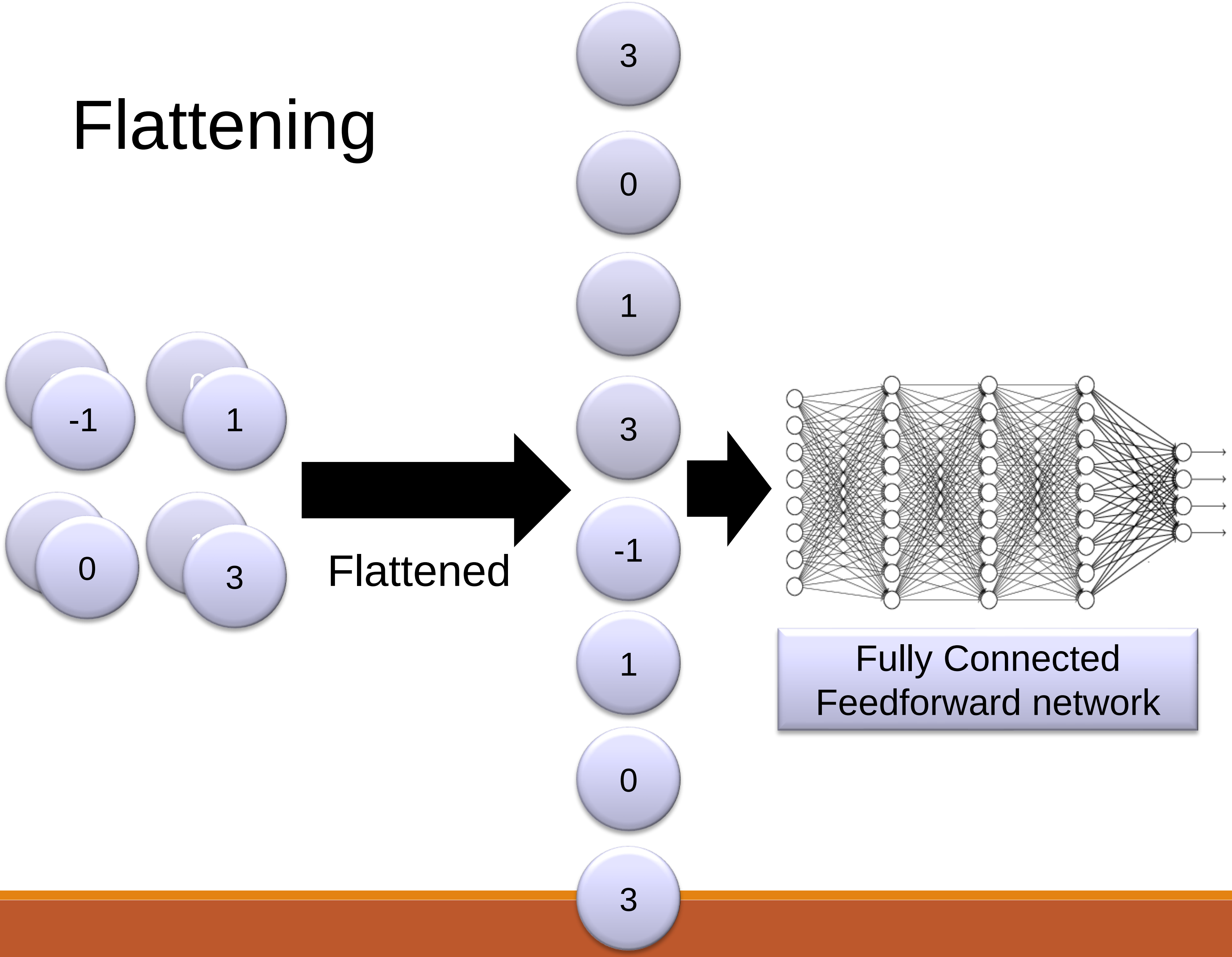
☺Trains much **faster**
- accelerates the convergence of SGD
- due to linear, non-saturating form

☺Less expensive operations
- compared to sigmoid/tanh (exponentials etc.)
- implemented by simply thresholding a matrix at zero

☺More **expressive**

☺Prevents the **gradient vanishing problem**

Flattening

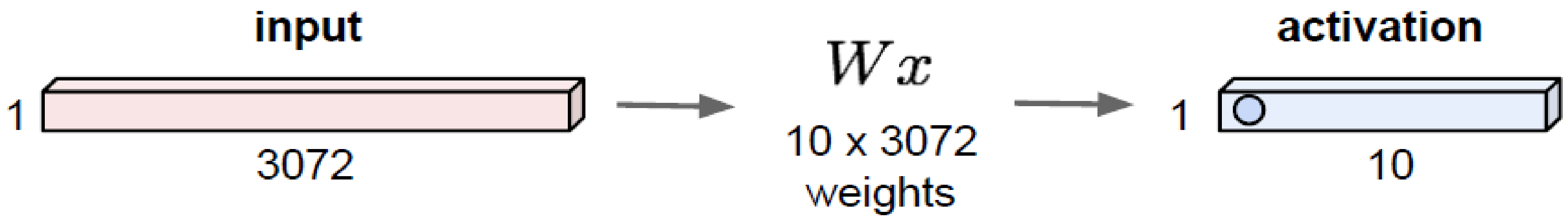


Flattened

Fully Connected Feedforward network

Fully Connected Layer

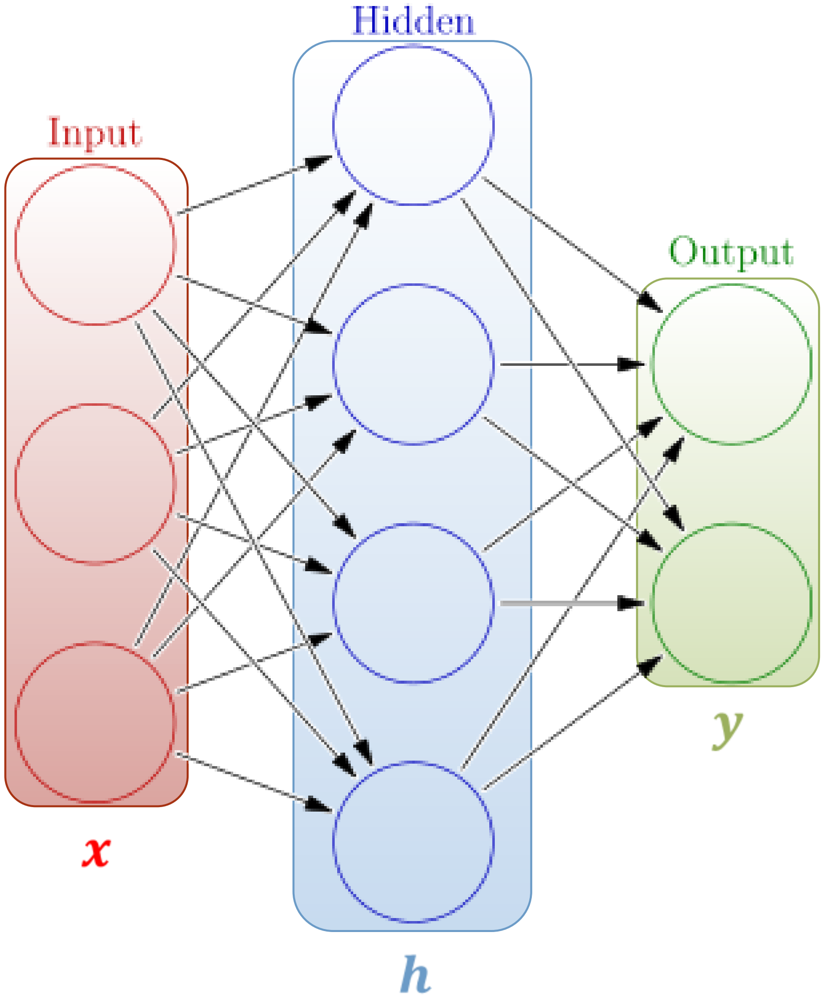
32x32x3 image -> stretch to 3072 x 1



Contains neurons that connect to the entire input volume as in ordinary Neural Networks

$$W_{(10x3072)} * x_{(1x3072)} = Activation_{(1x10)}$$

Neural Network Intro



Weights

$$h = \sigma(W_1 x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

Activation functions

How do we train?

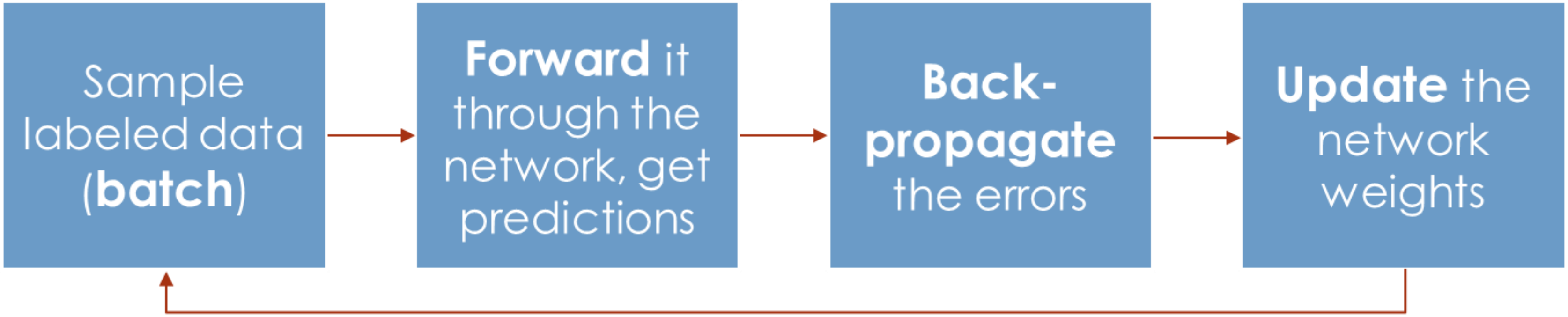
4 + 2 = 6 neurons (not counting inputs)
[3 x 4] + [4 x 2] = 20 weights
4 + 2 = 6 biases

---

26 learnable **parameters**

Demo

Training

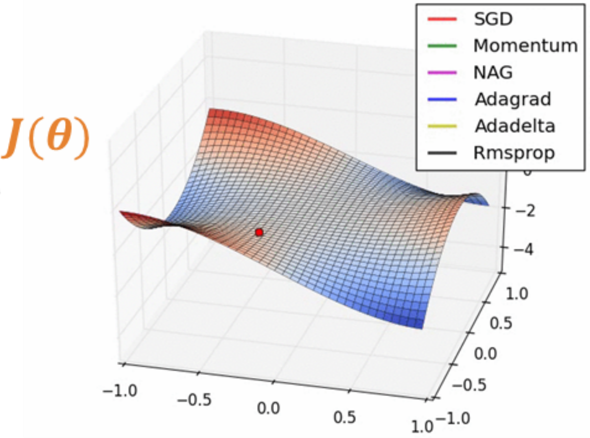


Sample labeled data (**batch**)

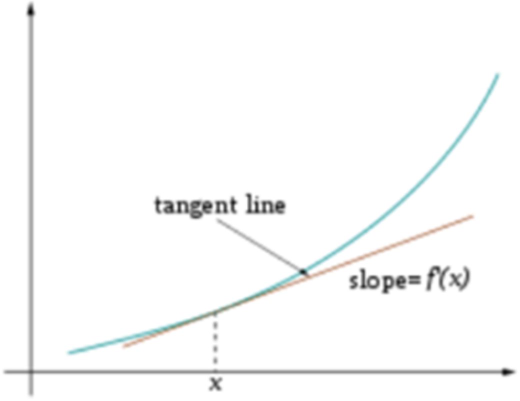
Forward it through the network, get predictions

Back-propagate the errors

Update the network weights



Optimize (min. or max.) **objective/cost function $J(\theta)$**
Generate **error signal** that measures difference between predictions and target values



Use error signal to change the **weights** and get more accurate predictions
Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39

Gradient Descent

objective/cost function $J(\theta)$

Review of backpropagation

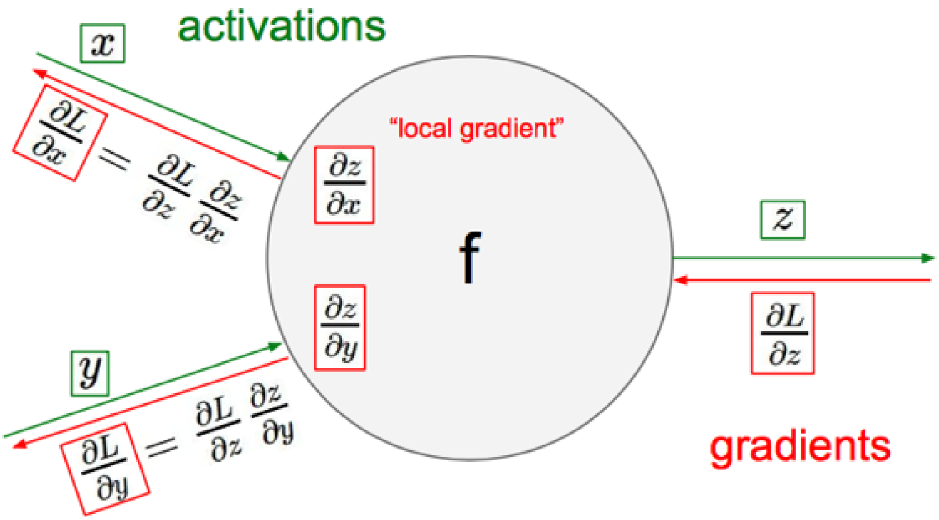
$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{d}{d\theta_j^{old}} J(\theta)$$

Update each element of θ

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

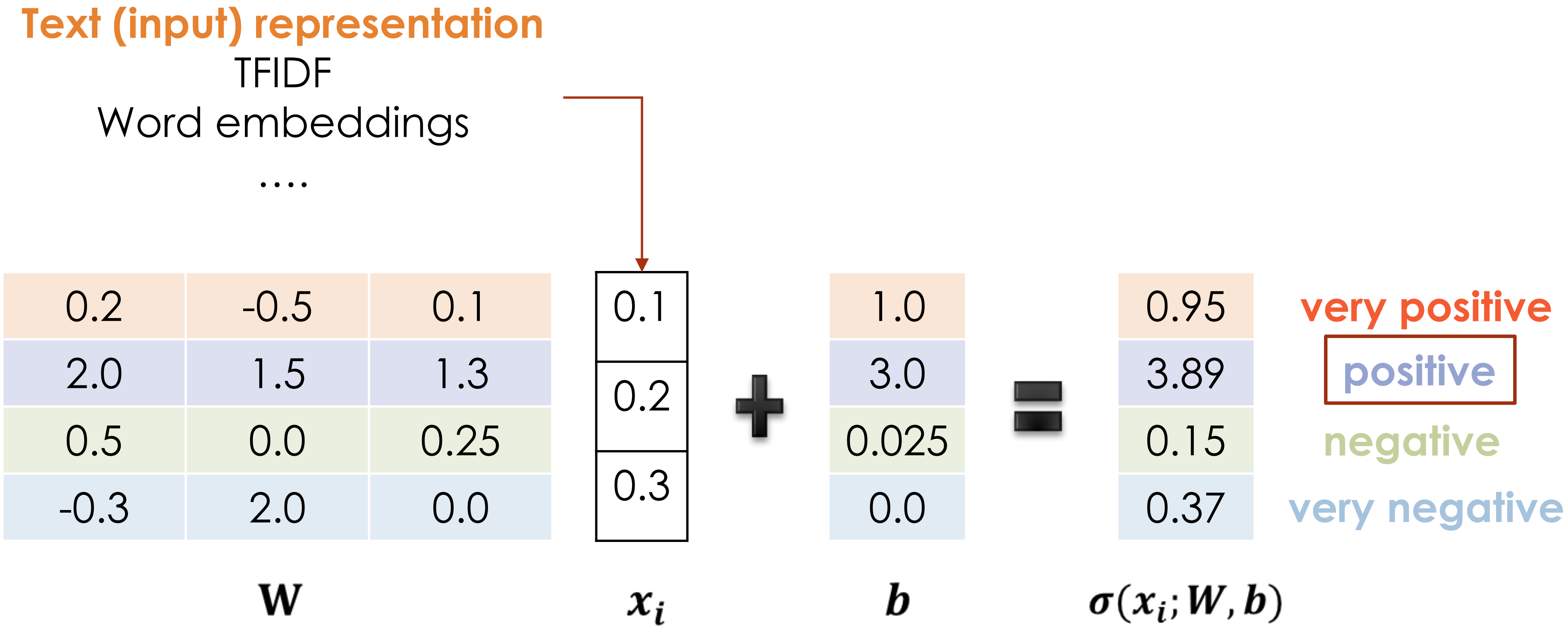
Matrix notation for all parameters

learning rate



Recursively apply **chain rule** though each node

One forward pass



Text (input) representation
TFIDF
Word embeddings
….

$$\begin{bmatrix}0.2 & -0.5 & 0.1\\ 2.0 & 1.5 & 1.3\\ 0.5 & 0.0 & 0.25\\ -0.3 & 2.0 & 0.0\end{bmatrix} \begin{bmatrix}0.1\\ 0.2\\ 0.3\end{bmatrix} + \begin{bmatrix}1.0\\ 3.0\\ 0.025\\ 0.0\end{bmatrix} = \begin{bmatrix}0.95\\ 3.89\\ 0.15\\ 0.37\end{bmatrix}$$

$\mathbf{W}$ $\boldsymbol{x_i}$ $\boldsymbol{b}$ $\boldsymbol{\sigma(x_i; W, b)}$

SoftMax Output

Prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities.

$$P(Y = k|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

$$s = f(x, W)$$

Softmax function

$$\begin{bmatrix} 3.2 \\ 5.1 \\ -1.7 \end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix} 24.5 \\ 164.0 \\ 0.18 \end{bmatrix} \xrightarrow{\text{normalize}} \begin{bmatrix} 0.13 \\ 0.87 \\ 0.001 \end{bmatrix}$$

Today: CNN Architectures

Case Studies

☐ AlexNet
☐ VGG
☐ GoogLeNet
☐ ResNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

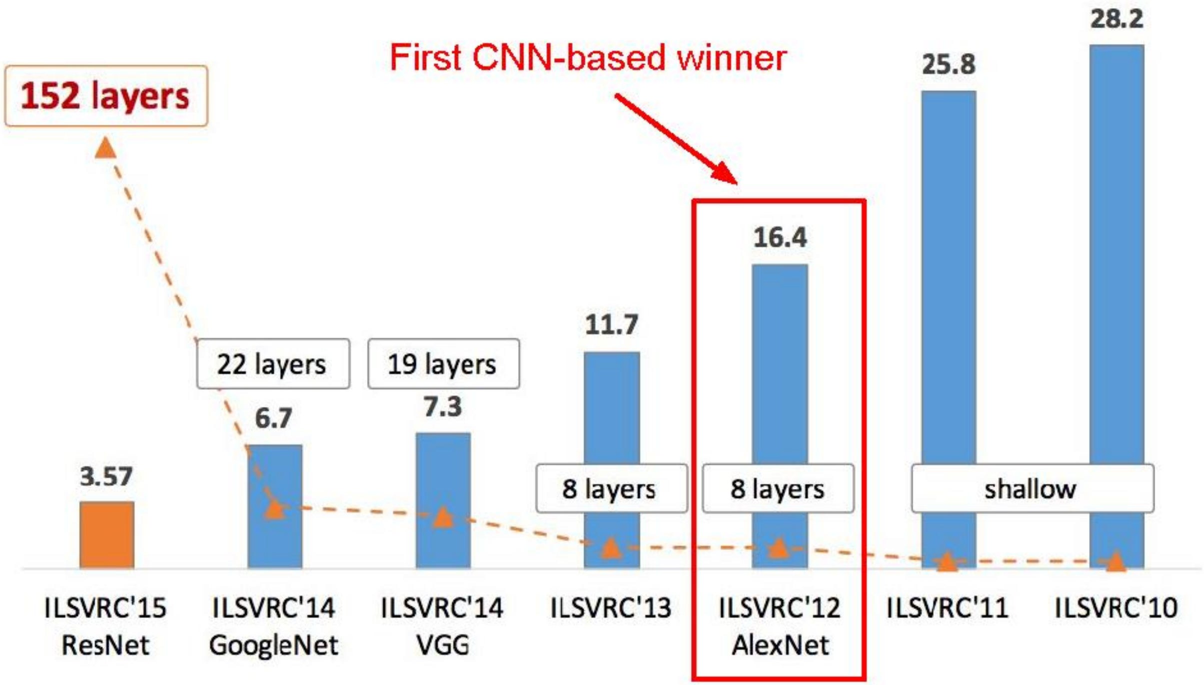
- The annual “Olympics” of computer vision.

- Teams from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more.

- **2012** marked **the first year where a CNN was used** to achieve a top 5 test error rate of 15.3%.

- The next best entry achieved an error of 26.2%.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]
"AlexNet"

Architecture:

- CONV1
- MAX POOL1
- NORM1
- CONV2
- MAX POOL2
- NORM2
- CONV3
- CONV4
- CONV5
- MAX POOL3
- FC6
- FC7
- FC8

Input: 227x227x3 images (224x224 before padding)

First layer: 96 11x11 filters applied at stride 4

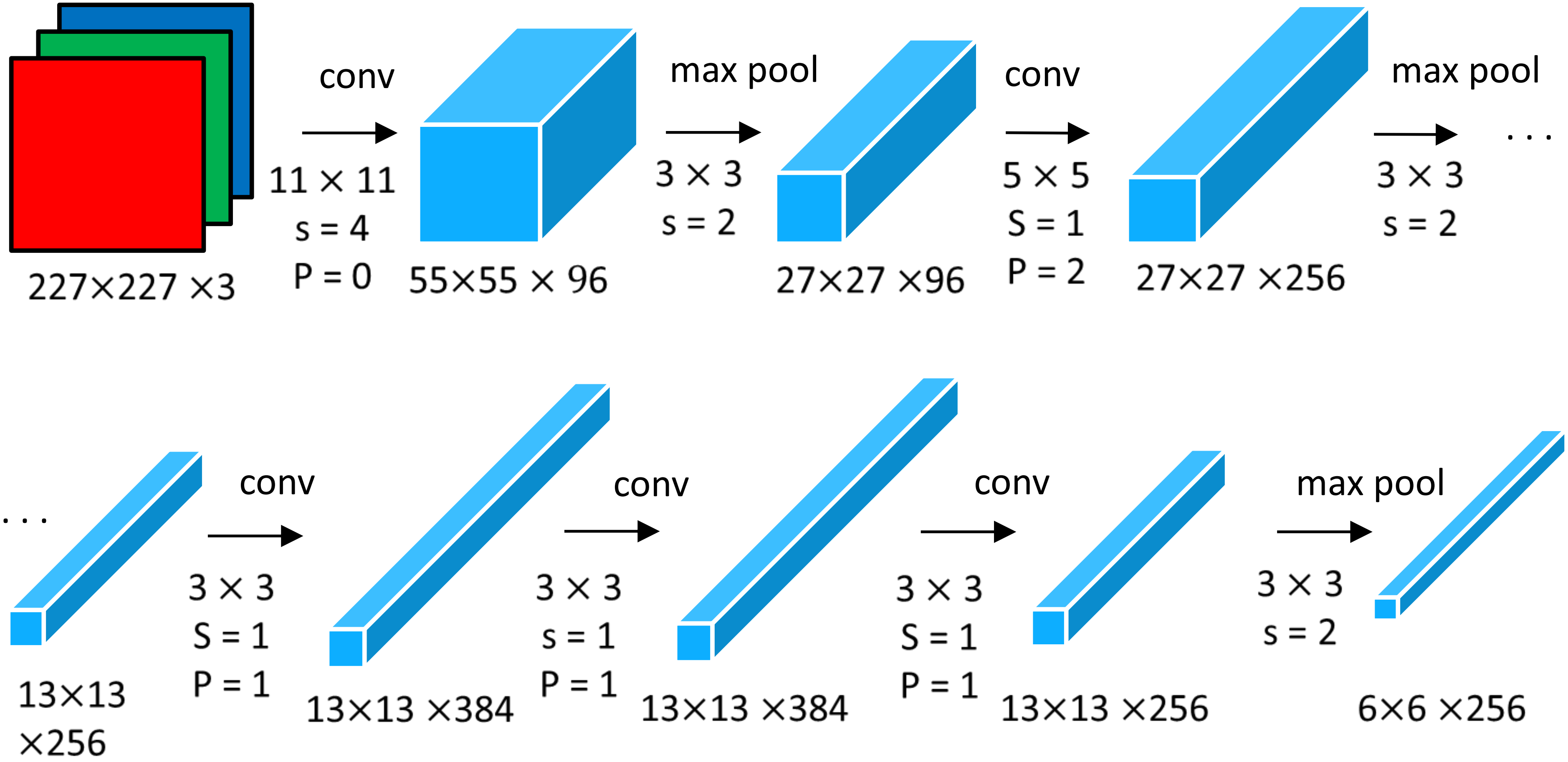
Output volume size?

(N-F)/s+1 = (227-11)/4+1 = 55 -> [55x55x96]

Number of parameters in this layer?

(11*11*3)*96 = 35K

AlexNet



227×227 ×3

conv
11 × 11
s = 4
P = 0

55×55 × 96

max pool
3 × 3
s = 2

27×27 ×96

conv
5 × 5
S = 1
P = 2

27×27 ×256

max pool
3 × 3
s = 2

. . .

. . .

13×13
×256

conv
3 × 3
S = 1
P = 1

13×13 ×384

conv
3 × 3
s = 1
P = 1

13×13 ×384

conv
3 × 3
S = 1
P = 1

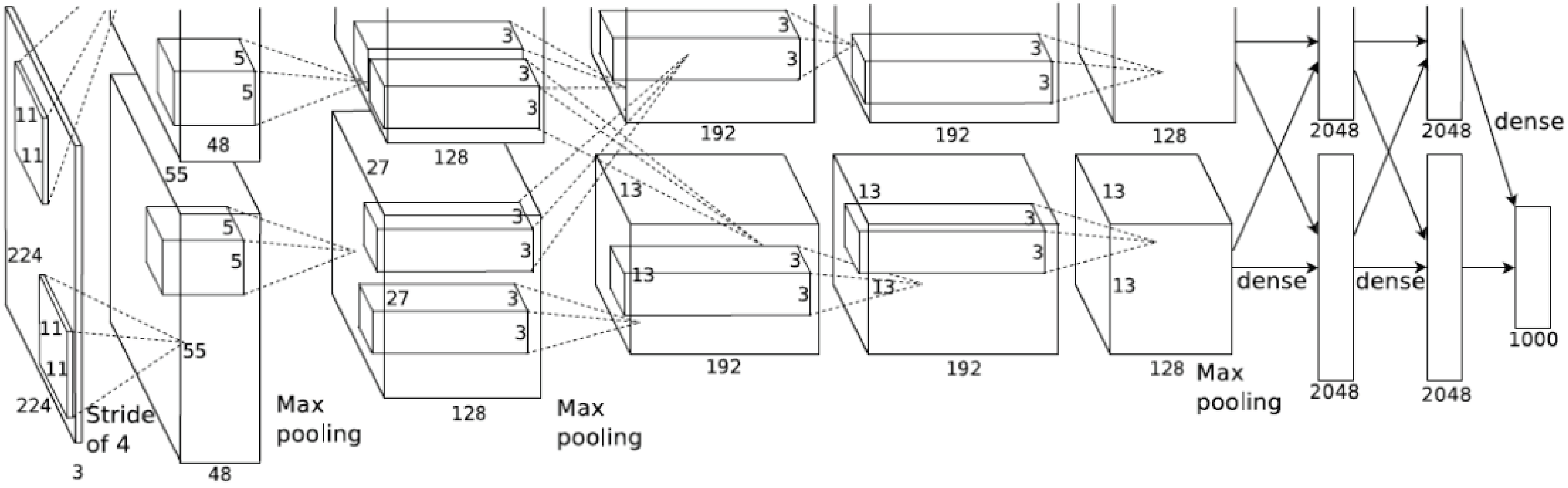
13×13 ×256

max pool
3 × 3
s = 2

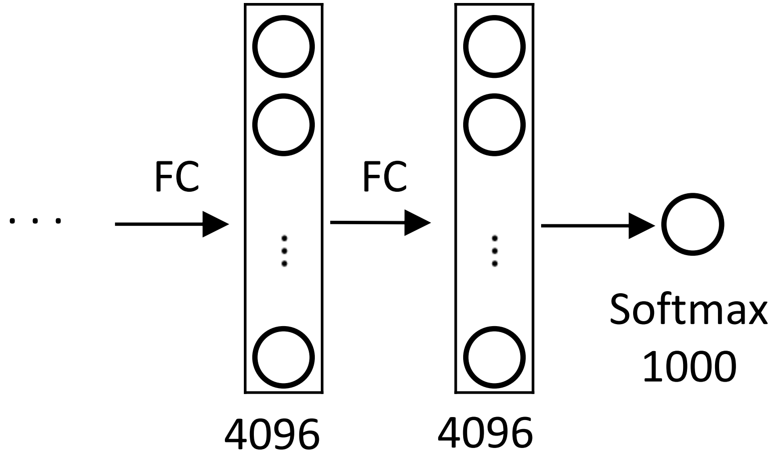
6×6 ×256

AlexNet

- Deep CNN architecture proposed by **Krizhevsky** [*Krizhevsky NIPS 2012*].
  - 5 convolutional layers (with pooling and ReLU)
  - 3 fully-connected layers
  - won ImageNet Large Scale Visual recognition Challenge 2012
  - top-1 validation error rate of 40.7%



AlexNet



VGGNet

Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015

Input
3x3 conv, 64
3x3 conv, 64
Pool 1/2
3x3 conv, 128
3x3 conv, 128
Pool 1/2
3x3 conv, 256
3x3 conv, 256
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
FC 4096
FC 4096
FC 1000
Softmax

- The runner-up at the ILSVRC 2014 competition
- 140 million parameters
- **Smaller filters**

  Only 3x3 CONV filters, stride 1, pad 1
  and 2x2 MAX POOL , stride 2
- **Deeper network**

  AlexNet: 8 layers
  VGGNet: 16 - 19 layers
- VGGNet: 7.3% top 5 error in ILSVRC’14

VGGNet

Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

What is the effective receptive field of three 3x3 conv (stride 1) layers?

7x7

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

Input	memory: 224*224*3=150K	params: 0
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864
Pool	memory: 112*112*64=800K	params: 0
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456
Pool	memory: 56*56*128=400K	params: 0
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
Pool	memory: 28*28*256=200K	params: 0
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 14*14*512=100K	params: 0
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 7*7*512=25K	params: 0
FC 4096	memory: 4096	params: 7*7*512*4096 = 102,760,448
FC 4096	memory: 4096	params: 4096*4096 = 16,777,216
FC 1000	memory: 1000	params: 4096*1000 = 4,096,000

VGG16:

TOTAL memory: 24M * 4 bytes ~= 96MB / image

TOTAL params: 138M parameters

VGGNet

Details/Retrospectives :

ILSVRC’14 2nd in classification, 1st in localization

Similar training procedure as AlexNet

No Local Response Normalisation (LRN)

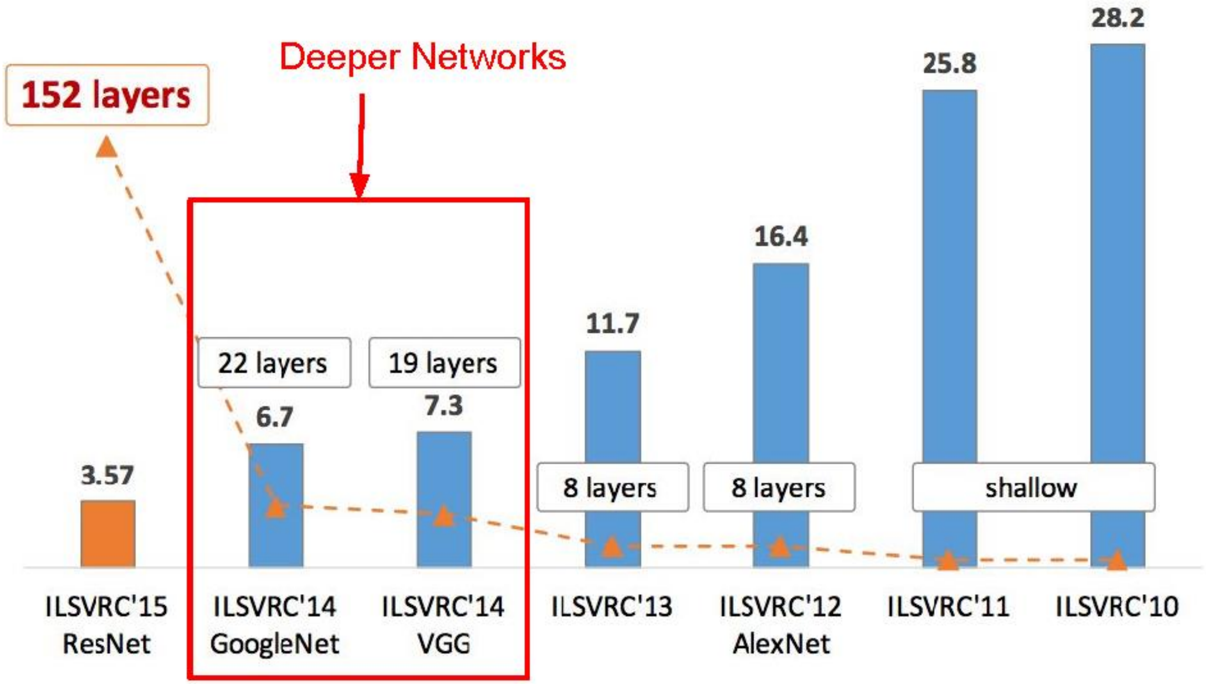
Use VGG16 or VGG19 (VGG19 only slightly better, more memory)

Use ensembles for best results

FC7 features generalize well to other tasks

Trained on 4 Nvidia Titan Black GPUs for **two to three weeks**.

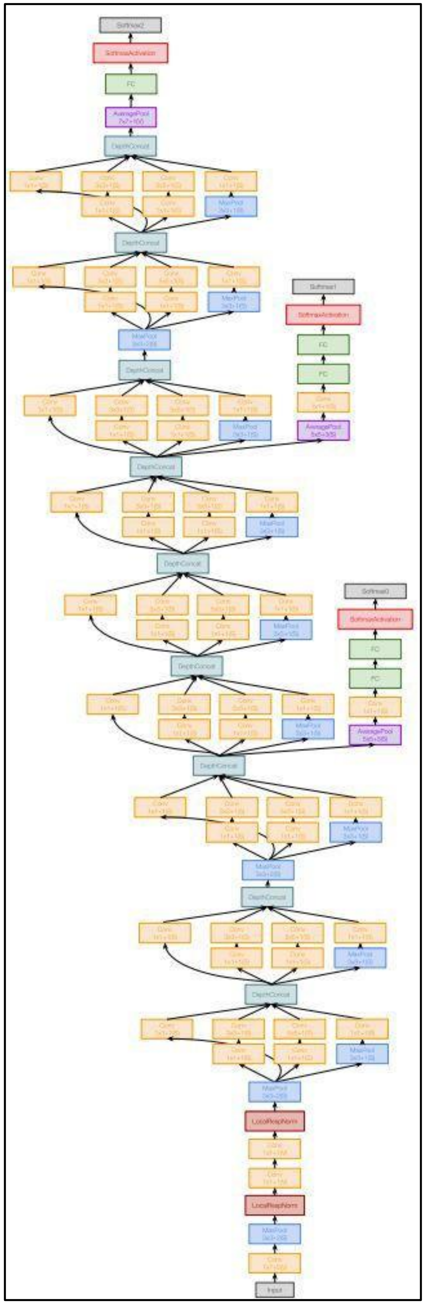
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



GoogLeNet

Going Deeper with Convolutions - Christian Szegedy et al.; 2015

- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

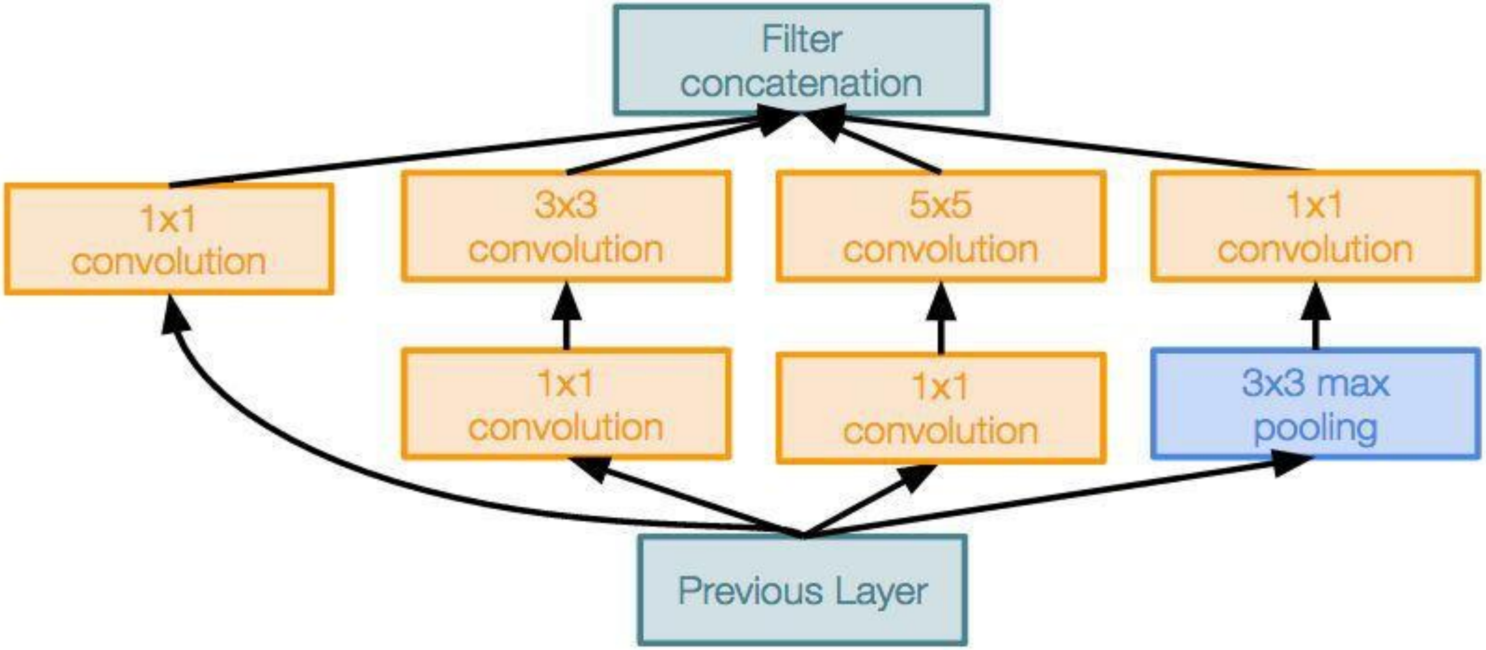


GoogleNet

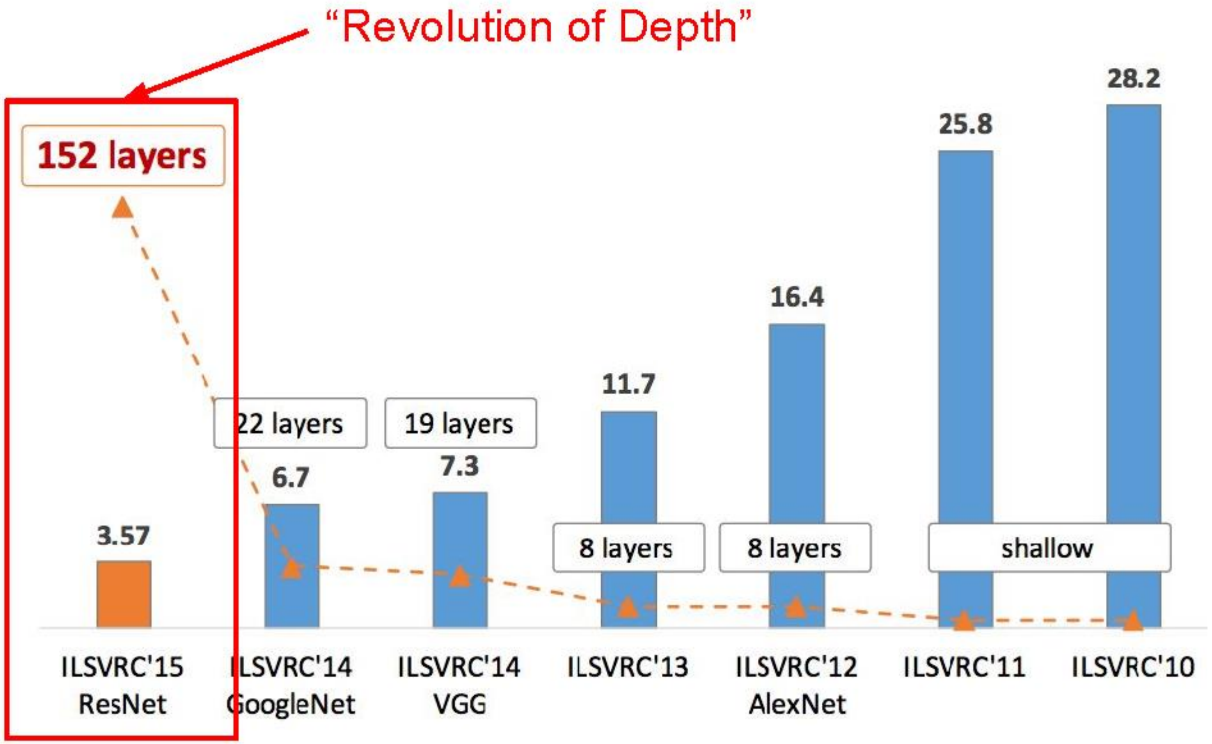
- 22 layers
- Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)

GoogLeNet

“**Inception module”:** design a good local network topology (network within a network) and then stack these modules on top



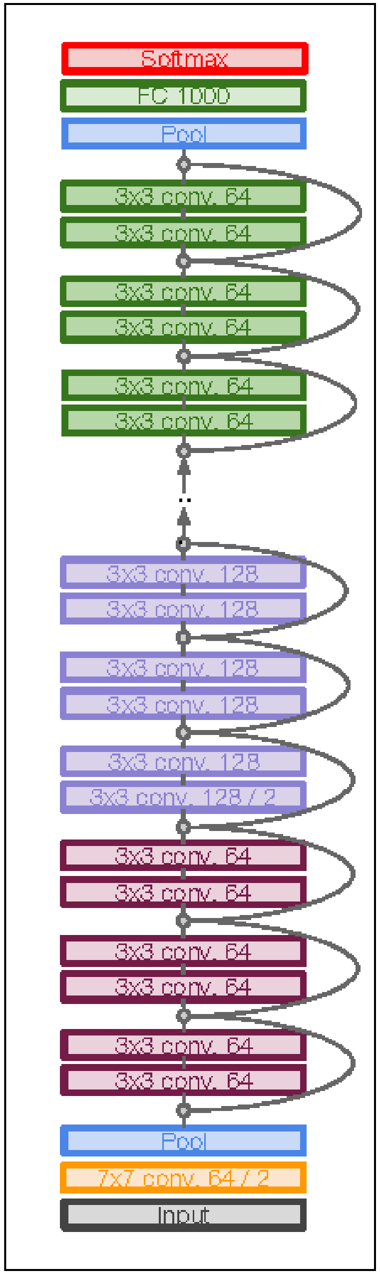
ResNet


"Revolution of Depth"
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

ResNet

Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015

- ☐ Extremely deep network – 152 layers
- ☐ Deeper neural networks are more difficult to train.
- ☐ Deep networks suffer from vanishing and exploding gradients.
- ☐ Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.



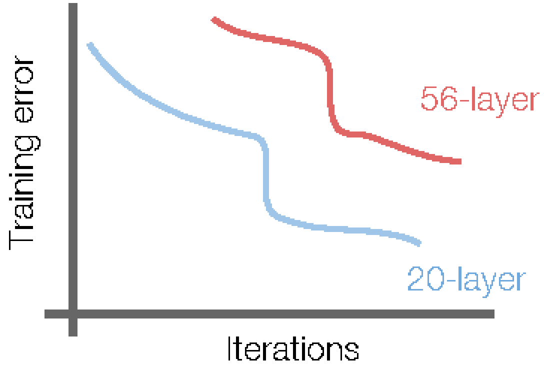
ResNet

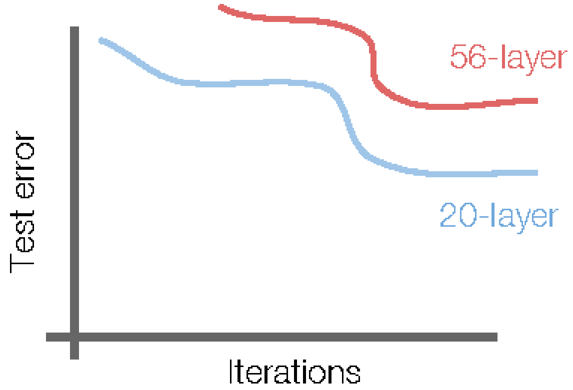
- ILSVRC’15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)

- Swept all classification and detection competitions in ILSVRC’15 and COCO’15!

ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?





- 56-layer model performs worse on both training and test error

-> The deeper model performs worse (not caused by overfitting)!

ResNet

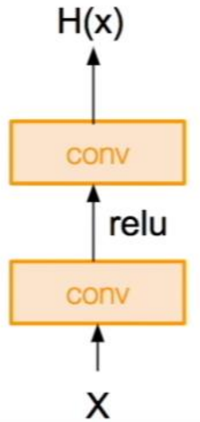
- **Hypothesis**: The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution**: Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.

- We will use **skip connections** allowing us to take the activation from one layer and feed it into another layer, much deeper into the network.
- Use layers to fit residual F(x) = H(x) – x
  instead of H(x) directly

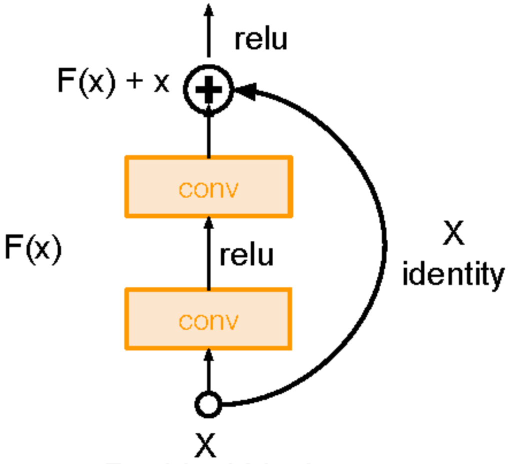
ResNet

Residual Block
Input x goes through conv-relu-conv series and gives us F(x). That result is then added to the original input x.

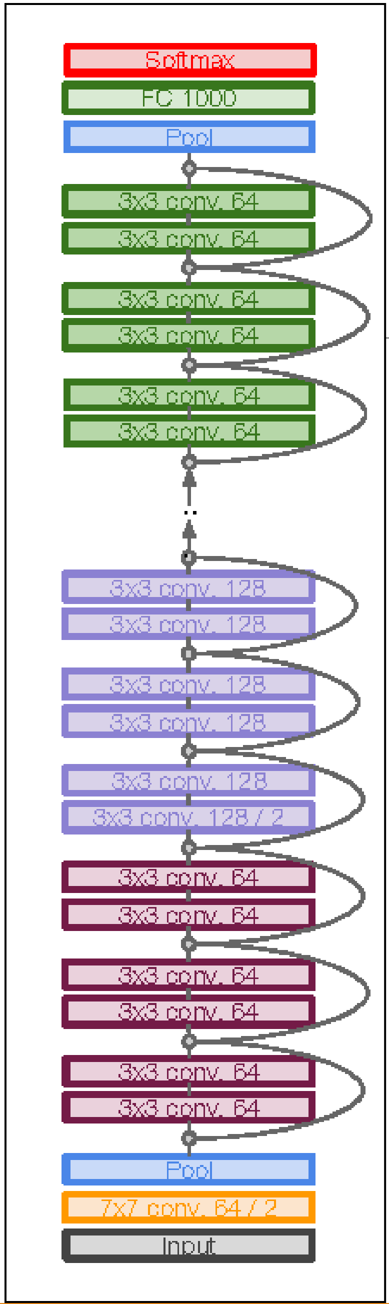
Let’s call that H(x) = F(x) + x.
In traditional CNNs, H(x) would just be equal to F(x). So, instead of just computing that transformation (straight from x to F(x)), we’re computing the term that we have to *add*, F(x), to the input, x.



“Plain” layers



Residual block

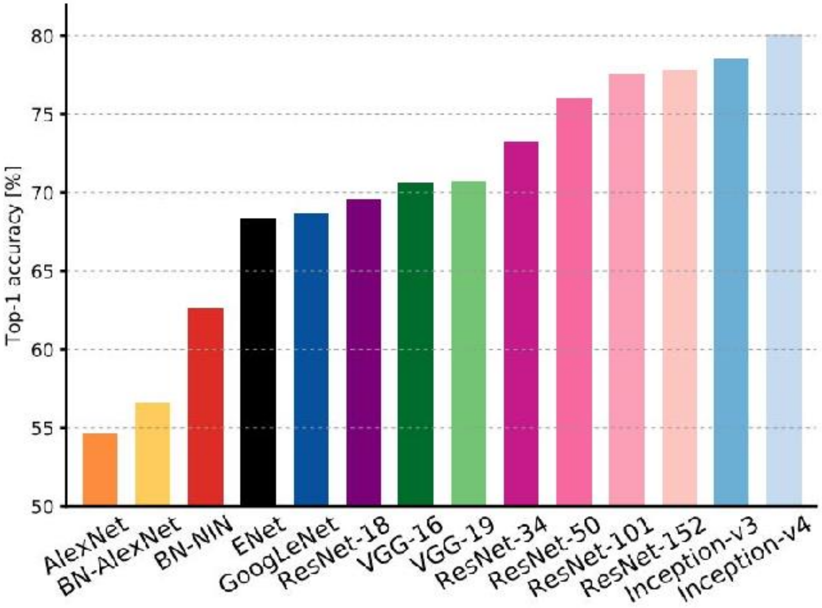


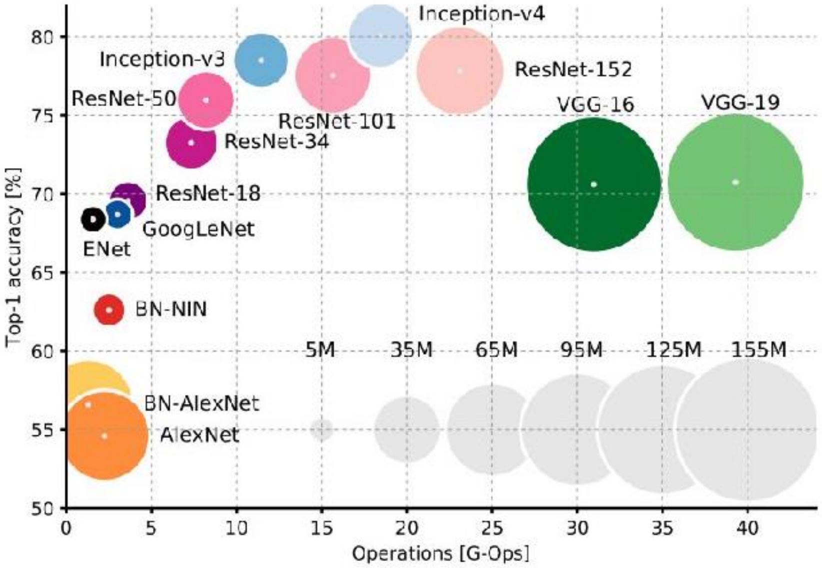
ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

Accuracy comparison





Keras

Keras: The Python Deep Learning library

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

https://keras.io/why-use-keras/

TensorFlow

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition.

https://keras.io/backend/

https://www.tensorflow.org/about

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

print('X_train:', x_train.shape)
print('y_train:', y_train.shape)
print('X_test:', x_test.shape)
print('y_test:', y_test.shape)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

References

- *Gradient-based learning applied to document recognition; ann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner; 1998*
- *ImageNet Classification with Deep Convolutional Neural Networks - Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; 2012*
- *Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015*
- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- *Stanford CS231- Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9*
- *Coursera, Machine Learning course by Andrew Ng.*

References

- *The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3) by Adit Deshpande* *https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html*
- *CNNs Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more … By Siddharth Das* *https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5*
- *Slide taken from Forward And Backpropagation in Convolutional Neural Network. – Medium , By Sujit Rai* *https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e*