Fundamentals of Deep Learning

Pontificia Universidad Católica del Perú
Summer Camp en IA
2025

Notes adapted from Dr. César Beltrán (PUCP) and Dr. Ivan Serina (UNIBS)

Review

Scalars

- A scalar is a single number
- Integers, real numbers, rational numbers, etc.

Vectors

- A vector is a 1-D array of numbers:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

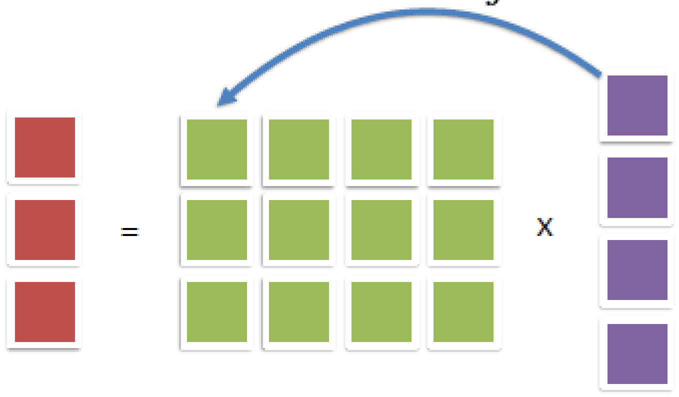
- Can be real, binary, integer, etc.
- Example notation for type and size:

$$\mathbb{R}^n$$

Matrices

- Multiplications (matrix and vector)

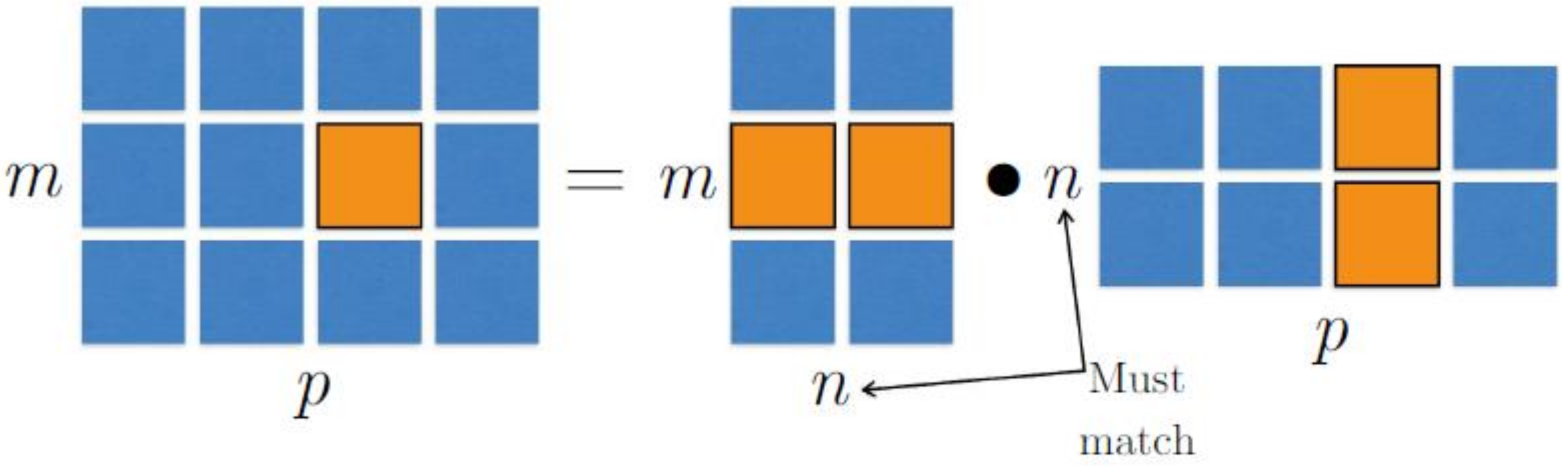
$$c = Ab \text{ where } c_i = \sum_j A_{ij} b_j$$



Matrix (Dot) Product

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}.$$

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}.$$



Tensors

A tensor is an array of numbers, that may have

- zero dimensions, and be a scalar
- one dimension, and be a vector
- Two dimensions, and be a matrix
- or more dimensions.

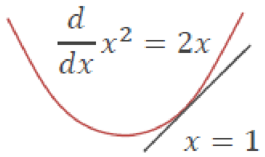
Review Scalar Derivative

y	a	x^n	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	nx^{n-1}	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

a is not a function of x

y	$u+v$	uv	$y = f(u), u = g(x)$
$\frac{dy}{dx}$	$\frac{du}{dx} + \frac{dv}{dx}$	$\frac{du}{dx}v + \frac{dv}{dx}u$	$\frac{dy}{du}\frac{du}{dx}$

Derivative is the slope of the tangent line



Gradients

		Scalar	Vector
		x	$\mathbf{x}$
Scalar	y	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
Vector	$\mathbf{y}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

Chain Rule

Chain rule for scalars:

$$y = f(u),\ u = g(x) \qquad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

Generalize to vectors straightforwardly

$$\underset{(1,n)}{\frac{\partial y}{\partial \mathbf{x}}} = \underset{(1,)}{\frac{\partial y}{\partial u}}\underset{(1,n)}{\frac{\partial u}{\partial \mathbf{x}}} \qquad \underset{(1,n)}{\frac{\partial y}{\partial \mathbf{x}}} = \underset{(1,k)}{\frac{\partial y}{\partial \mathbf{u}}}\underset{(k,n)}{\frac{\partial \mathbf{u}}{\partial \mathbf{x}}} \qquad \underset{(m,n)}{\frac{\partial \mathbf{y}}{\partial \mathbf{x}}} = \underset{(m,k)}{\frac{\partial \mathbf{y}}{\partial \mathbf{u}}}\underset{(k,n)}{\frac{\partial \mathbf{u}}{\partial \mathbf{x}}}$$

Chain Rule

Assume $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, \quad y \in \mathbb{R}$

$$z = \left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right)^2$$

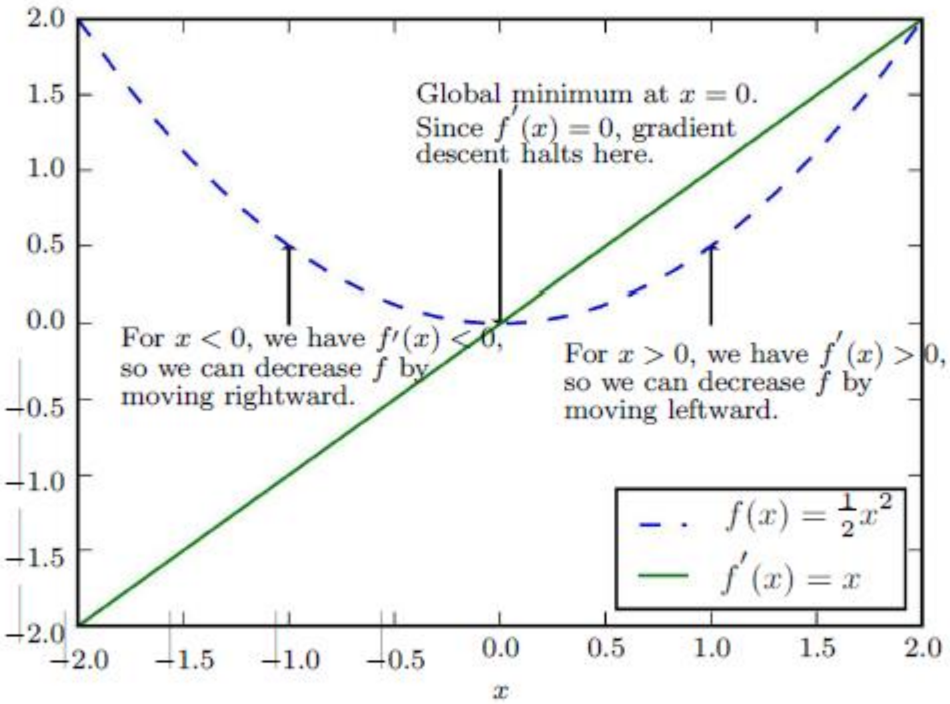
Compute $\dfrac{\partial z}{\partial \mathbf{w}}$

Decompose

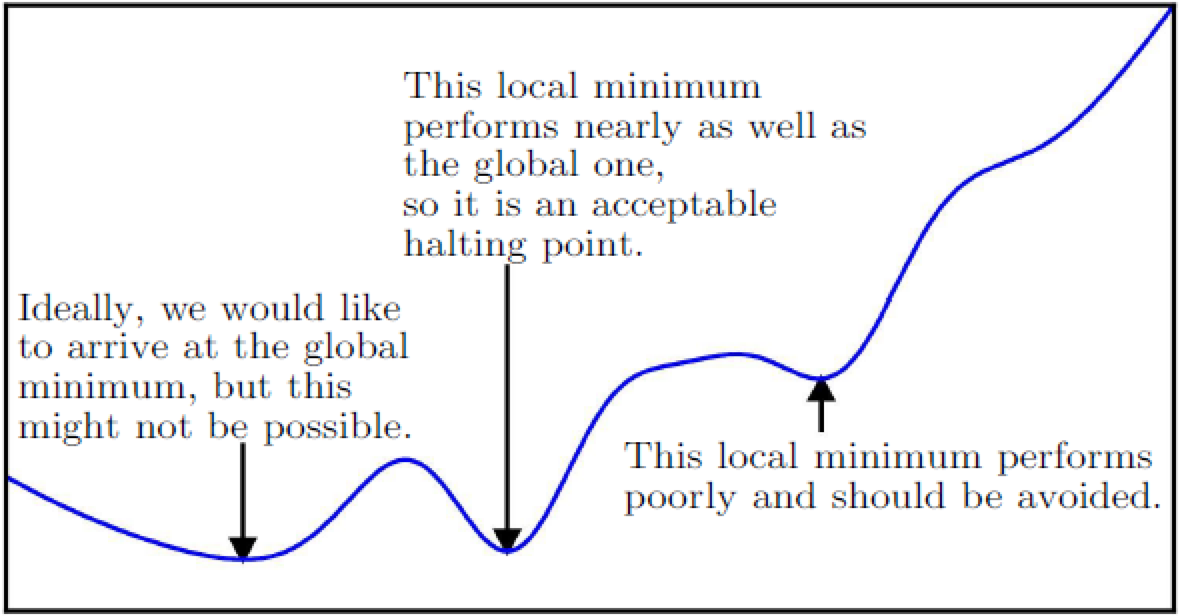
$$a = \langle \mathbf{x}, \mathbf{w} \rangle$$
$$b = a - y$$
$$z = b^2$$

$$\begin{aligned}
\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial \mathbf{w}} \\
&= \frac{\partial b^2}{\partial b}\frac{\partial a - y}{\partial a}\frac{\partial \langle \mathbf{x}, \mathbf{w} \rangle}{\partial \mathbf{w}} \\
&= 2b \cdot 1 \cdot \mathbf{x}^T \\
&= 2\left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right)\mathbf{x}^T
\end{aligned}$$

Gradient Descent



Approximate Optimization



History Review

Mark I Perceptron

Frank Rosenblatt ~1958

Mark I Perceptron

Vol. VI, No. 2, Summer 1958

research trends

CORNELL AERONAUTICAL LABORATORY, INC., BUFFALO 21, NEW YORK

The Design of an Intelligent AUTOMATON

by FRANK ROSENBLATT

Introducing the perceptron — A machine which senses, recognizes, remembers, and responds like the human mind.

STORIES about the creation of machines having human qualities have long been a fascinating province in the realm of science fiction. Yet we are now about to witness the birth of such a machine — a machine capable of perceiving, recognizing, and identifying its surroundings without any human training or control.

Development of that machine has stemmed from a search for an understanding of the physical mechanisms which underlie human experience and intelligence. The question of the nature of these processes is at least as ancient as any other question in western science and philosophy, and, indeed, ranks as one of the greatest scientific challenges of our time.

Our understanding of this problem has gone perhaps as far as had the development of physics before Newton. We have some excellent descriptions of the phenomena to be explained, a number of interesting hypotheses, and a little detailed knowledge about events in the nervous system. But we lack agreement on any integrated set of principles by which the functioning of the nervous system can be understood.

We believe now that this ancient problem is about to yield to our theoretical investigation for three reasons:

First, in recent years our knowledge of the functioning of individual cells in the central nervous system has vastly increased.

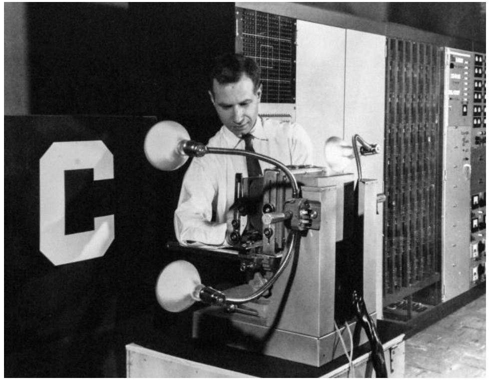
Second, large numbers of engineers and mathematicians are, for the first time, undertaking serious study of the mathematical basis for thinking, perception, and the handling of information by the central nervous system, thus providing the hope that these problems may be within our intellectual grasp.

Third, recent developments in probability theory and in the mathematics of random processes provide new tools for the study of events in the nervous system, where only the gross statistical organization is known and the precise cell-by-cell "wiring diagram" may never be obtained.

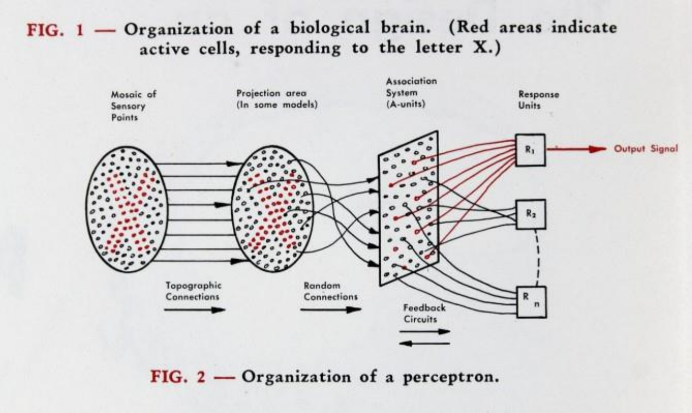
Receives Navy Support

In July, 1957, Project PARA (Perceiving and Recognizing Automaton), an internal research program which had been in progress for over a year at Cornell Aeronautical Laboratory, received the support of the Office of Naval Research. The program had been concerned primarily with the application of probability theory to

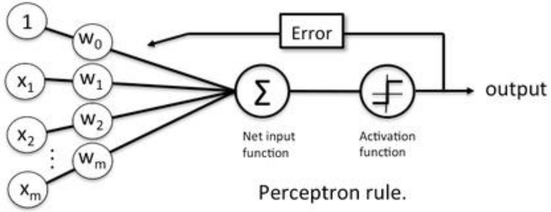
The first page of Rosenblatt's article, “The Design of an Intelligent Automaton,” in Research Trends, a Cornell Aeronautical Laboratory publication, Summer 1958.



Rosenblatt and the perceptron.

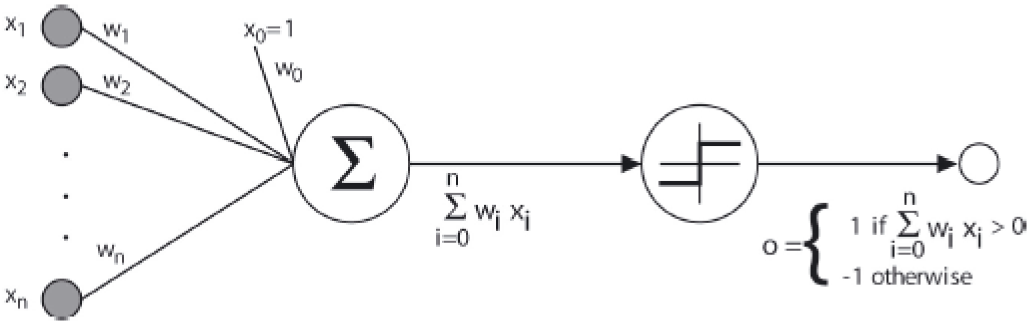


An image of the perceptron from Rosenblatt's “The Design of an Intelligent Automaton,” Summer 1958.



Images courtesy of Cornell Chronicle (2019)

Perceptron



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

$o(x_1, \ldots, x_n) = -w_0$ is the **threshold**

Simpler vector notation (adding $x_0 = 1$):

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ -1 & \text{otherwise.} \end{cases} = sgn(\mathbf{w} \cdot \mathbf{x})$$

Perceptron training rule

$$o(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Determine weights w_i

$$w_i \leftarrow w_i + \Delta w_i$$

where

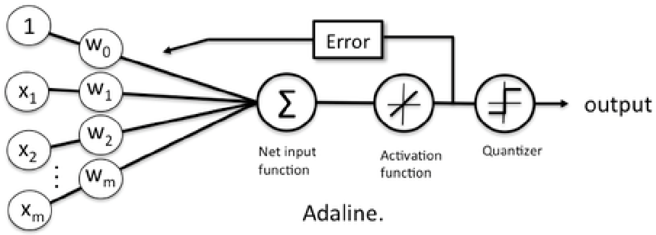
$$\Delta w_i = \eta(t - o)x_i$$

- $t = c(\mathbf{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.05) called *learning rate*

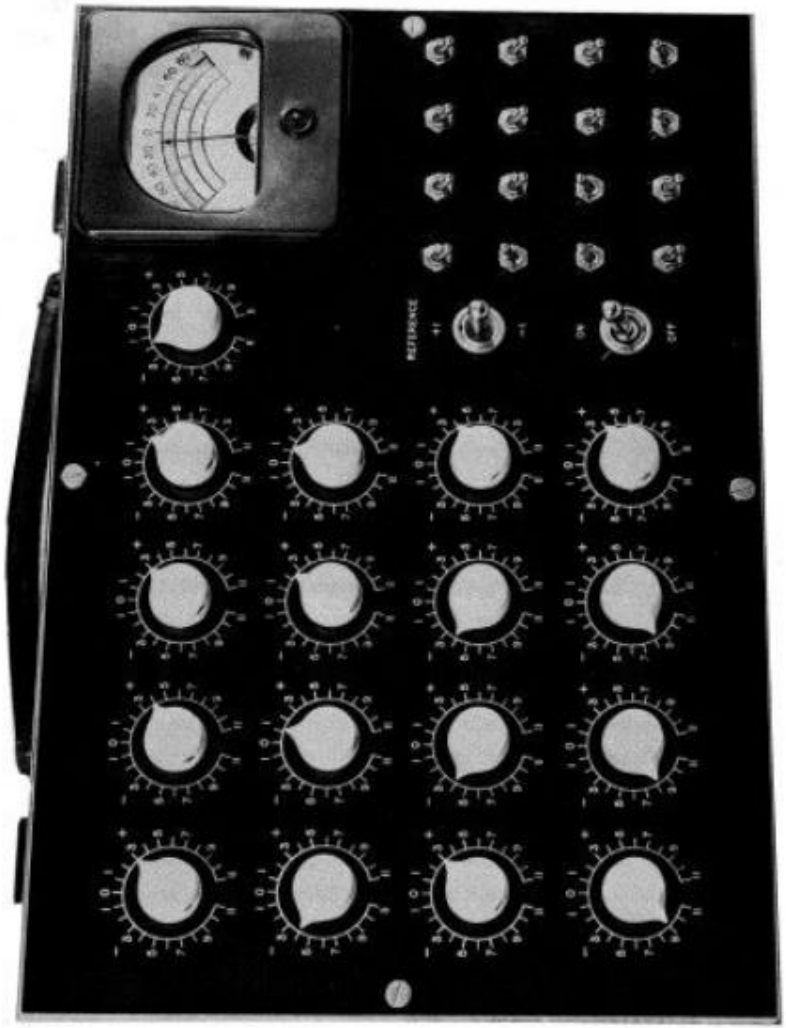
Adeline/Madeline

Widrow and Hoff ~1960

Adaptive Linear Neuron (Adeline)



https://www.youtube.com/watch?v=IEFRtz68m-8



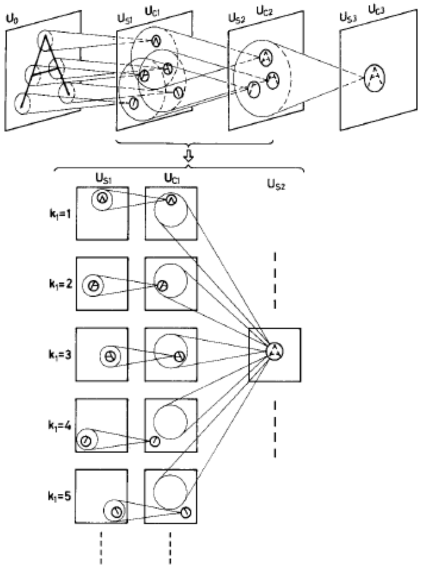


Fig. 5. An example of the interconnections between cells and the response of the cells after completion of self-organization

Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.

Fukushima K. 1980

https://www.youtube.com/watch?v=Qil4kmvm2Sw

The backward pass starts by computing $\partial E/\partial y$ for each of the output units. Differentiating equation (3) for a particular case, c, and suppressing the index c gives

$$\partial E/\partial y_j = y_j - d_j \tag{4}$$

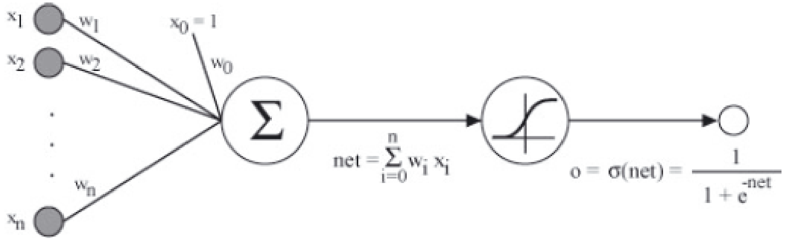
We can then apply the chain rule to compute $\partial E/\partial x_j$

$$\partial E/\partial x_j = \partial E/\partial y_j \cdot \mathrm{d}y_j/\mathrm{d}x_j$$

Learning representations by back-propagating errors

Rumelhart et. al., 1986

Sigmoid unit



$\sigma(x)$ is the sigmoid function $\frac{1}{1+e^{-x}}$ (nonlinear and differentiable)

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient descent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ **Backpropagation**

Cost Function

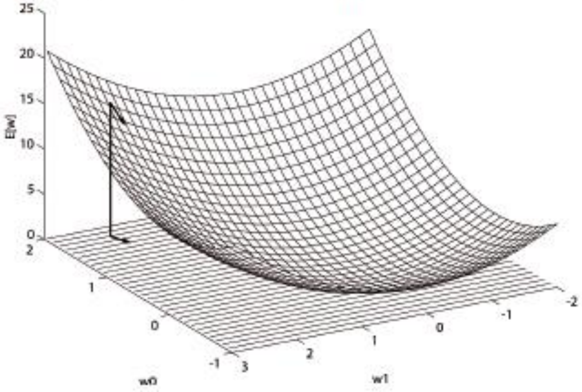
$$o_{\mathbf{x}}(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_n x_n = \mathbf{w} \cdot \mathbf{x}$$

Let's learn w_i's from training examples $D = \{\langle \mathbf{x}^{(k)}, t^{(k)} \rangle\}$ that minimize the sum of the squared errors

$$E[\mathbf{w}] \equiv \frac{1}{2} \sum_{k=1}^{|D|} (t^{(k)} - \mathbf{w} \cdot \mathbf{x}^{(k)})^2$$

Gradient Descent

Gradient: $\nabla E[\mathbf{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$



Training rule:

$$\Delta \mathbf{w} = -\eta \nabla E[\mathbf{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Every weight is modified by a small quantity in the opposite direction (addition or subtraction) that mostly minimizes E

Backpropagation Algorithm

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do (each iteration is an **epoch**)
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k, compute $\delta_k = o_k(1 - o_k)(t_k - o_k)$
  3. For each hidden unit h, compute $\delta_h = o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$
  4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \eta \delta_j x_{ji} \ (= w_{ji} - \eta \frac{\partial E}{\partial w_{ji}})$$

In a 2-layer networks w_{ji} are weights from the input to the hidden units and from the hidden to the output units. For $i = k$, $\frac{\partial E}{\partial w_{ji}}$ defined as for single layer units.

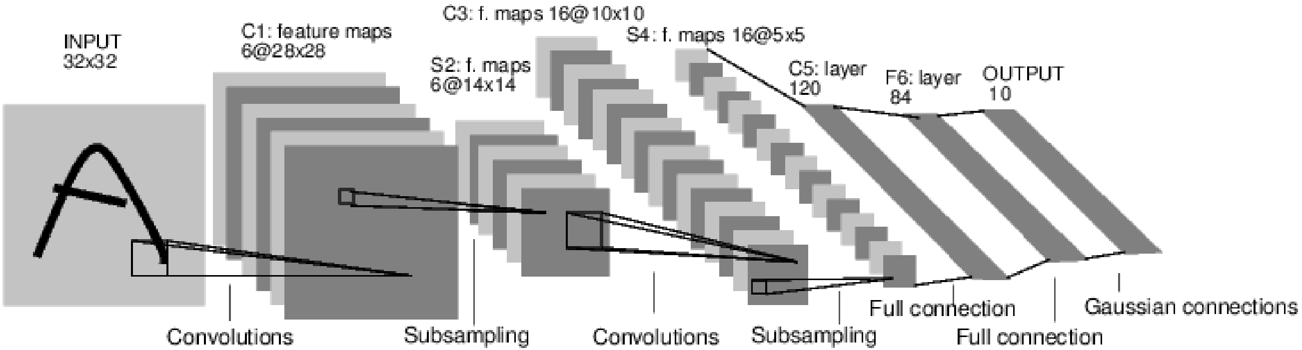


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Gradient-based learning applied to document recognition

Y. Le Cun et. al, 1998

Reducing the Dimensionality of Data with Neural Networks

Hinton and Salakhutdinov 2006

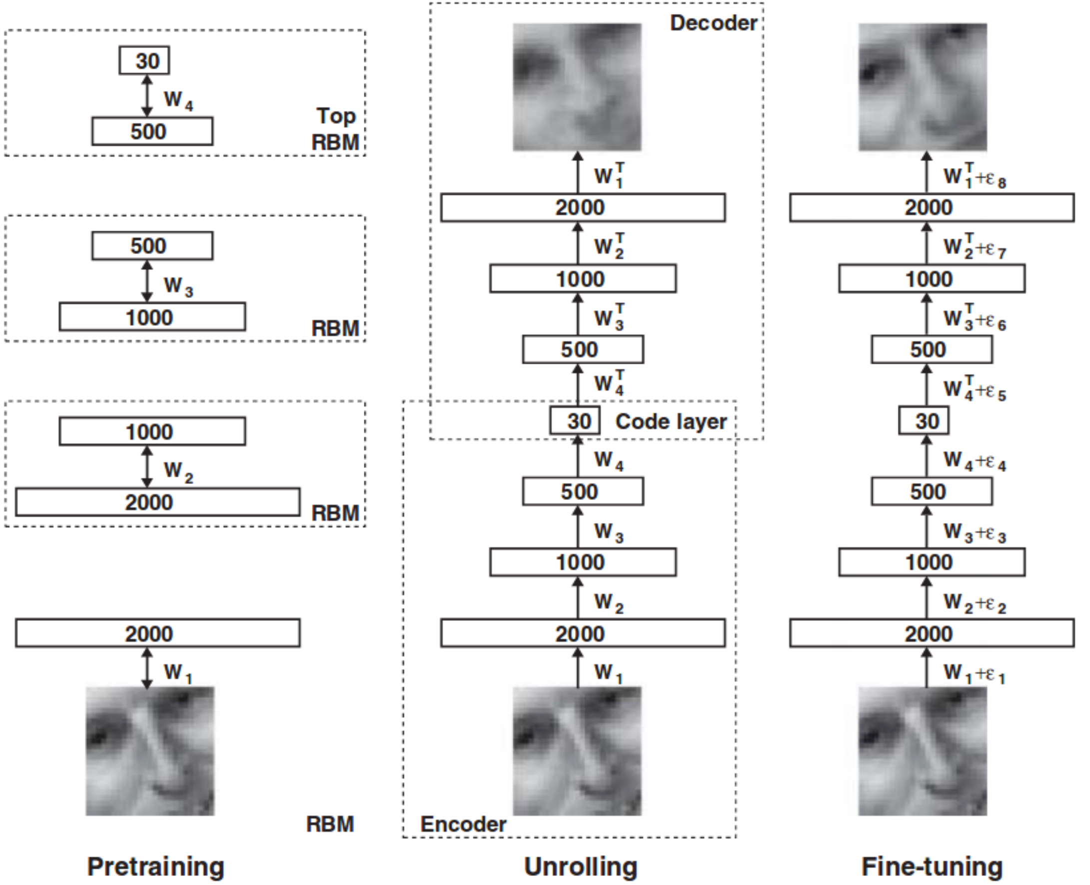
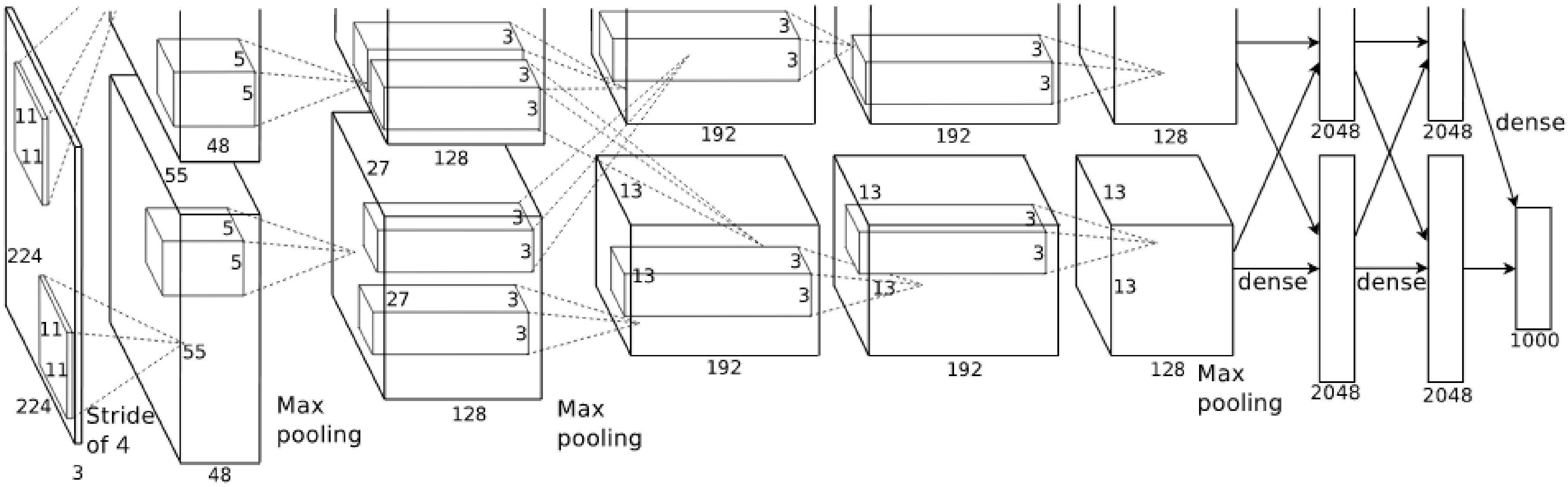


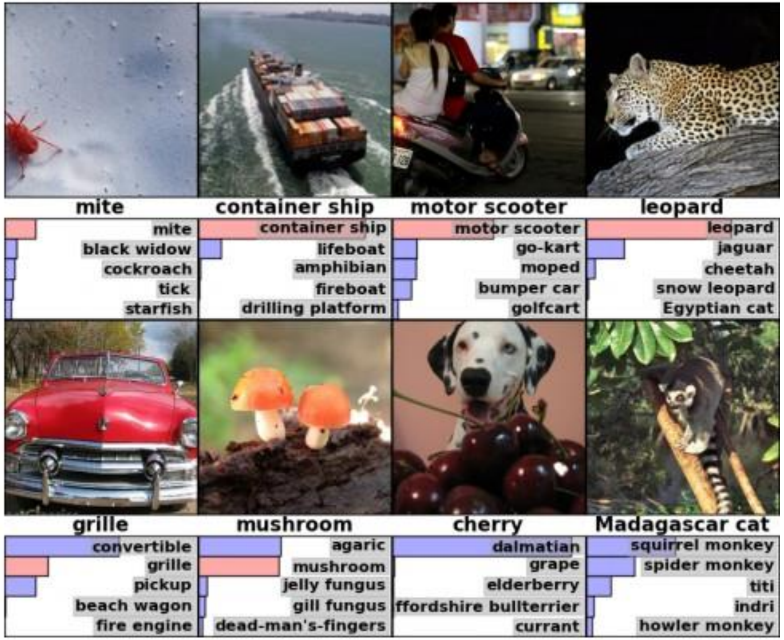
Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.



Imagenet classification with deep convolutional neural networks

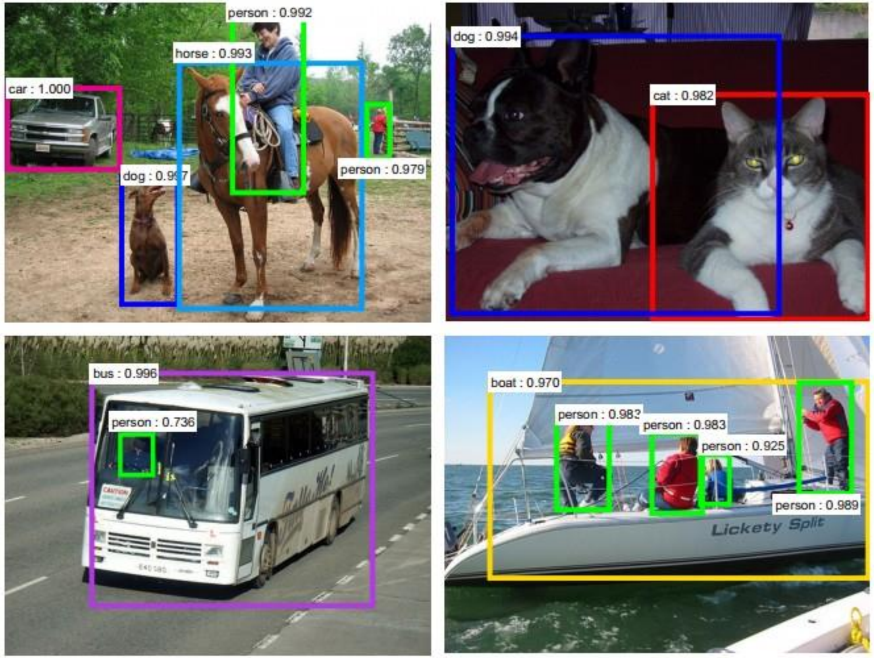
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012

Classification



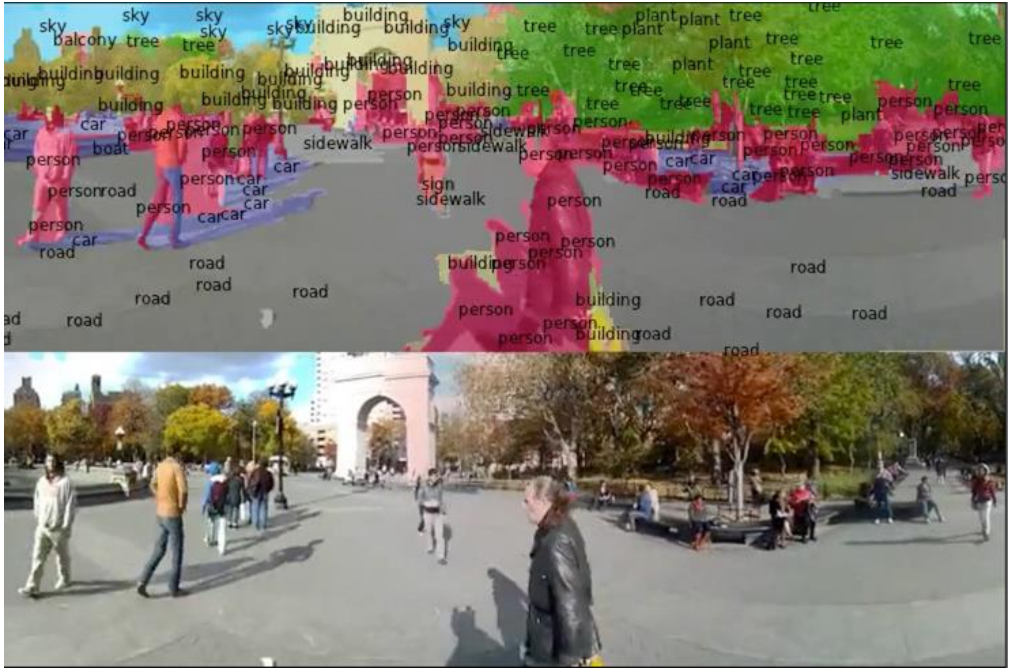
[Krizhevsky 2012]

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



[Farabet et al., 2012]

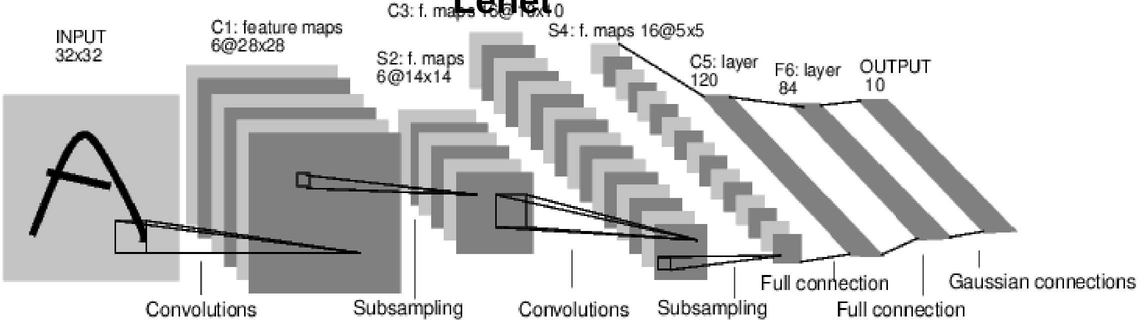
Convolutional Neural Networks

CNN

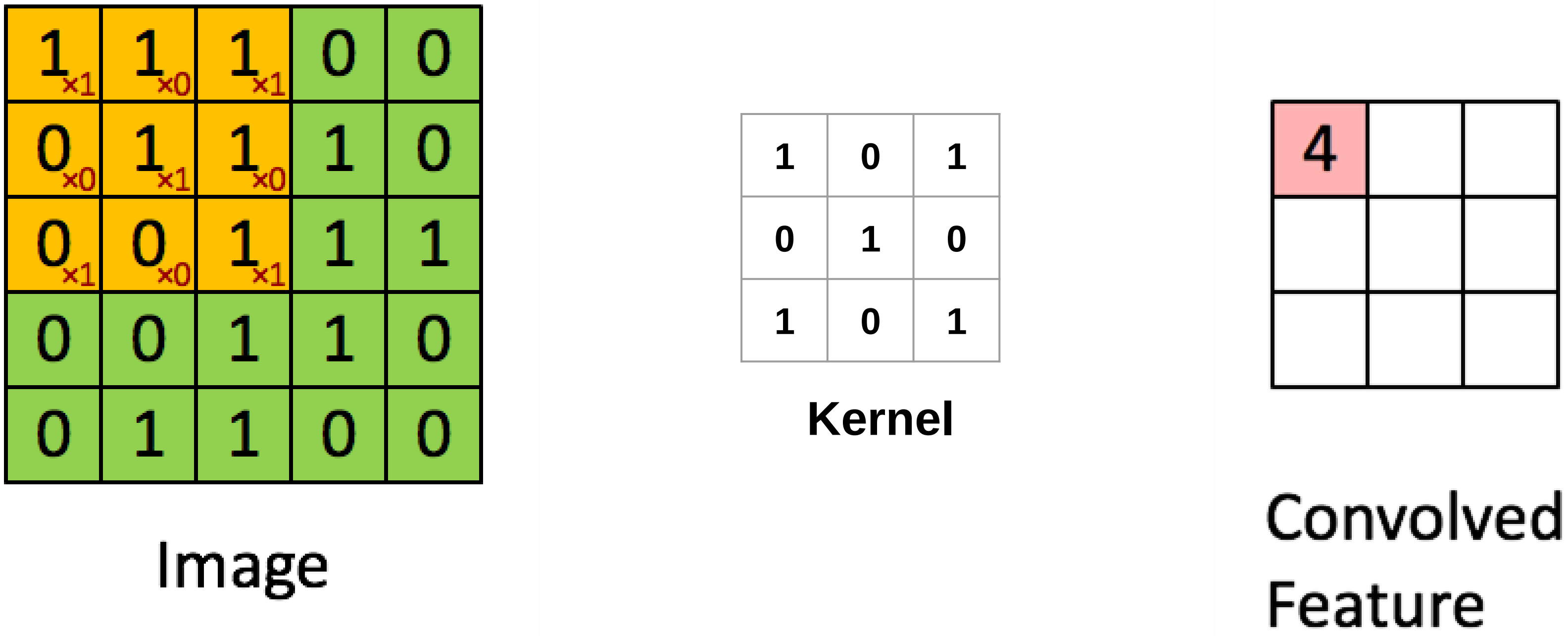
- CNN architecture main task is the feature extraction through 2D or 3D convolutional operations.

- The simple CNN framework involves four layers: convolutional, activation, pooling, and fully connected layer.

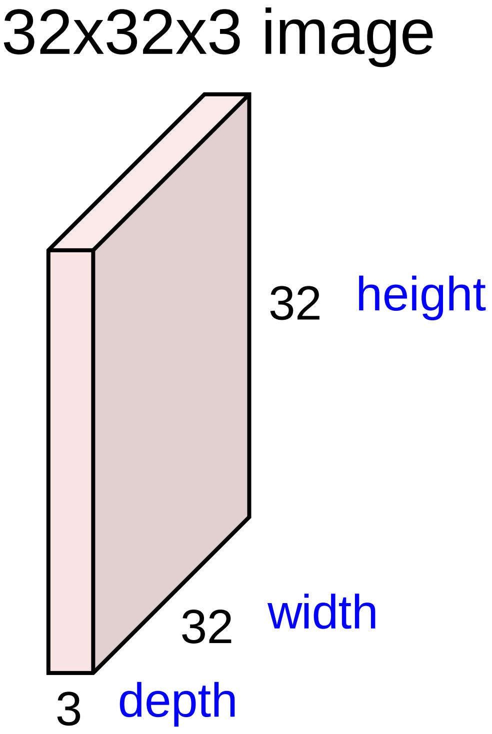
Lenet



¿Qué es una convolución?

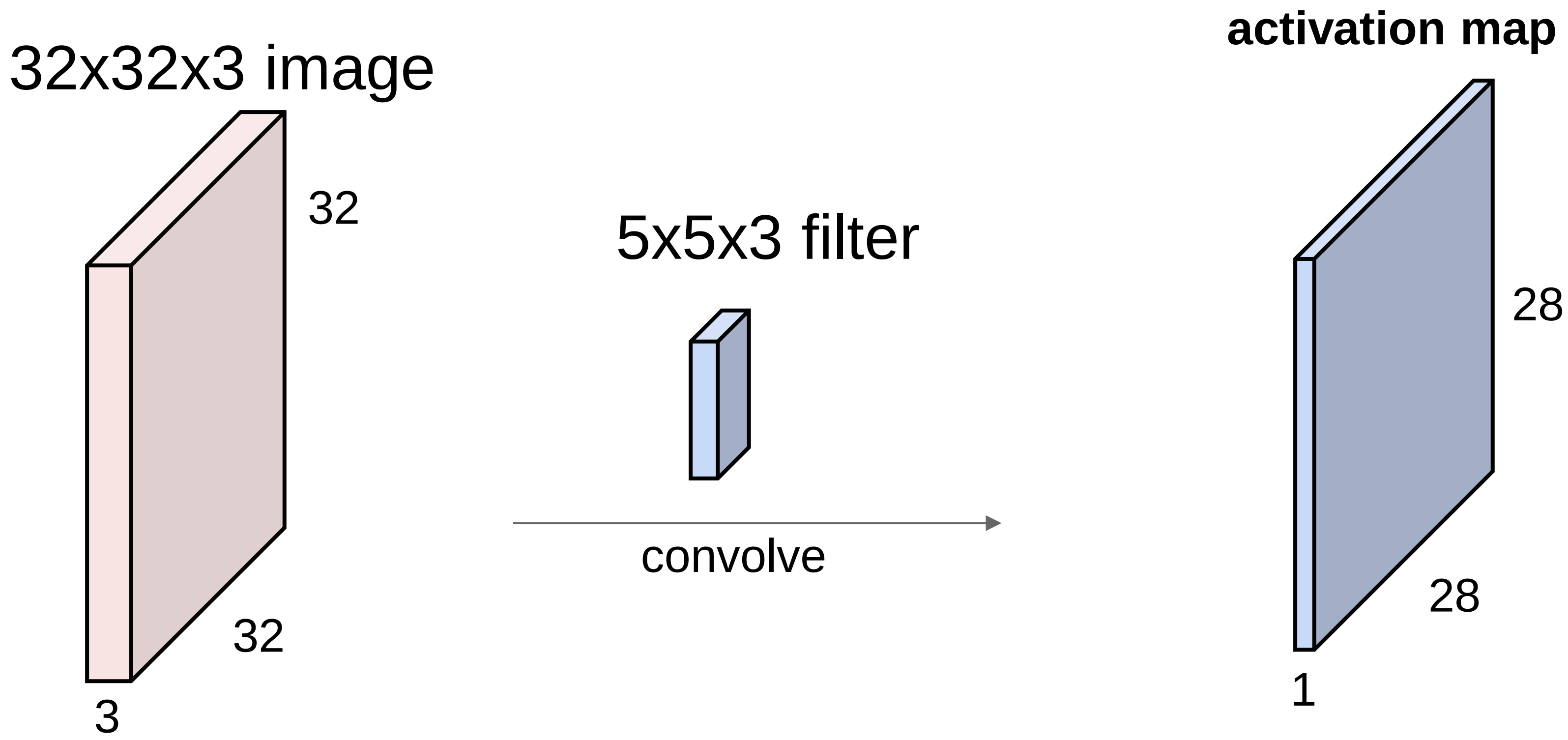


Convolution Layer

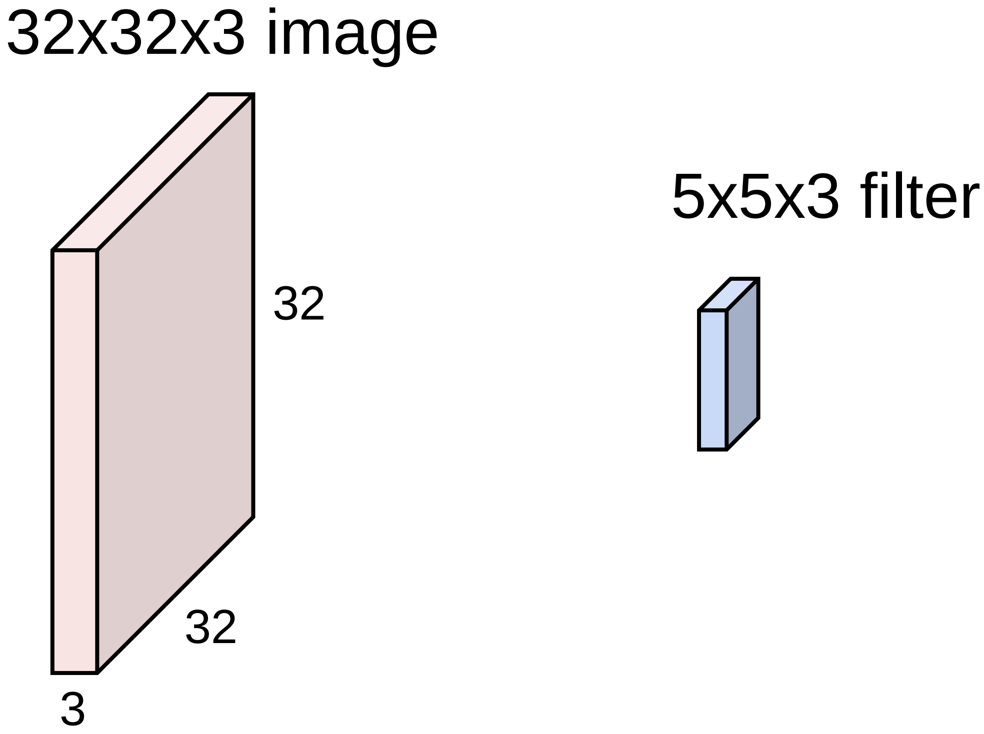


http://setosa.io/ev/image-kernels/

Convolution Layer

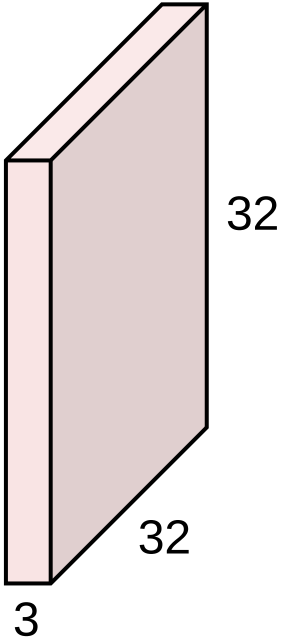


Convolution Layer

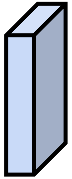


Convolution Layer

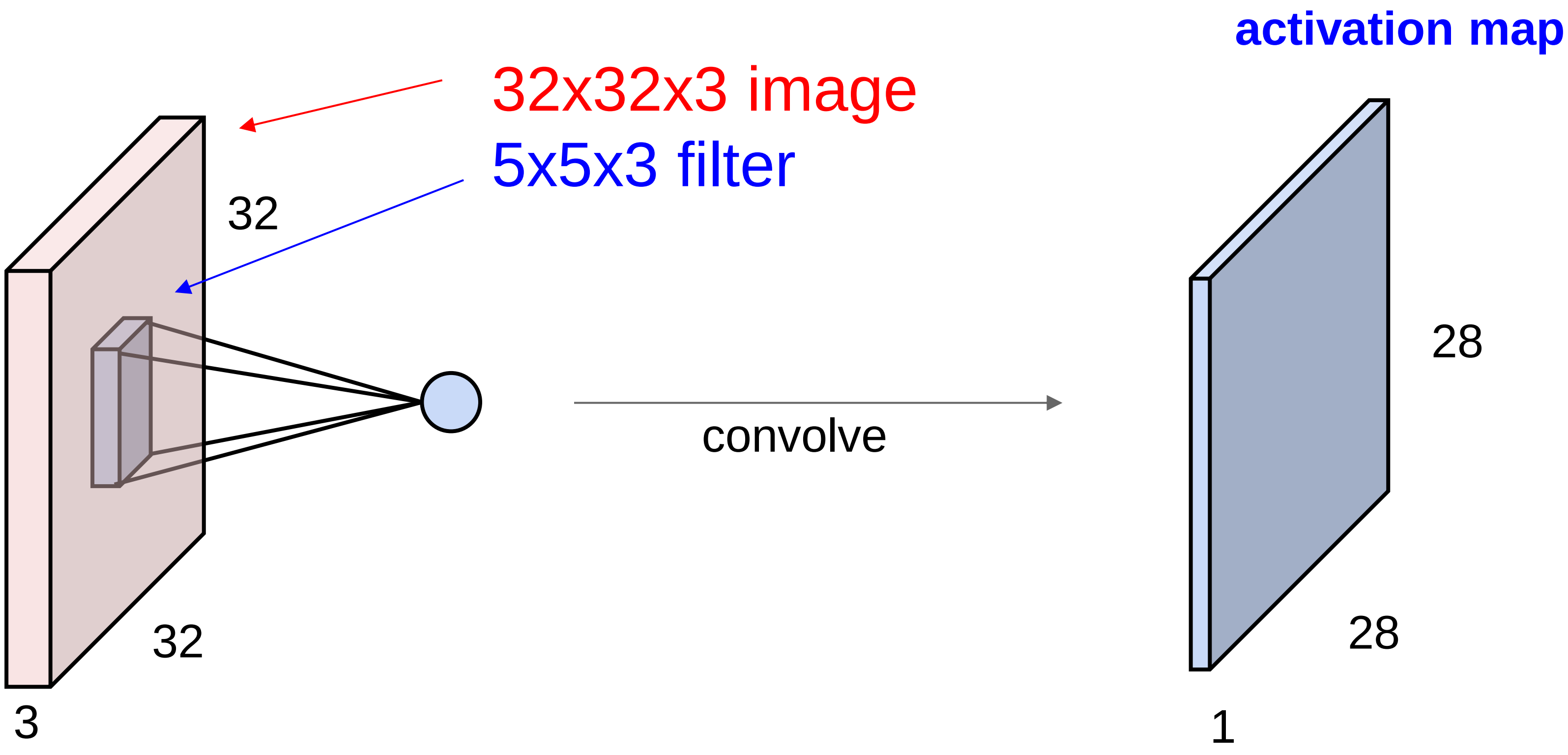
32x32x**3** image



5x5x**3** filter

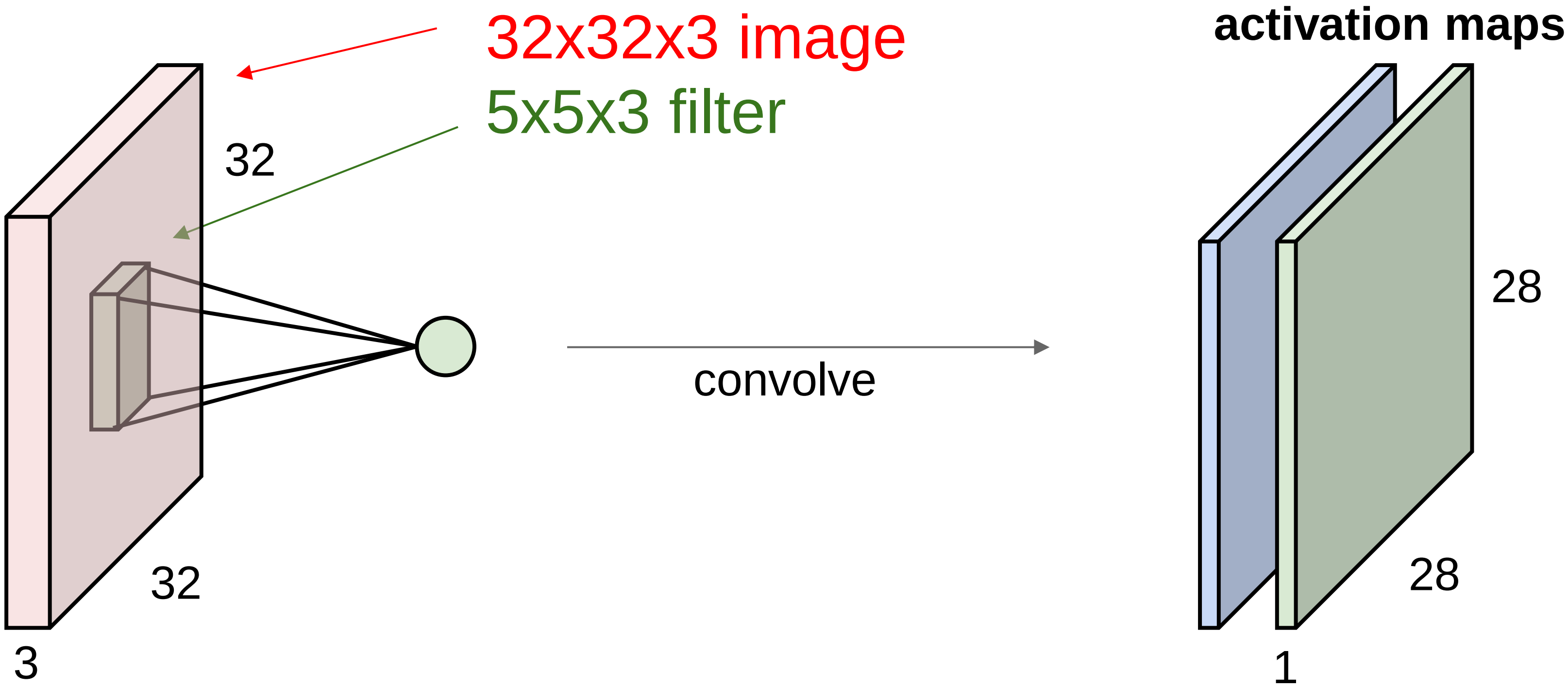


Convolution Layer

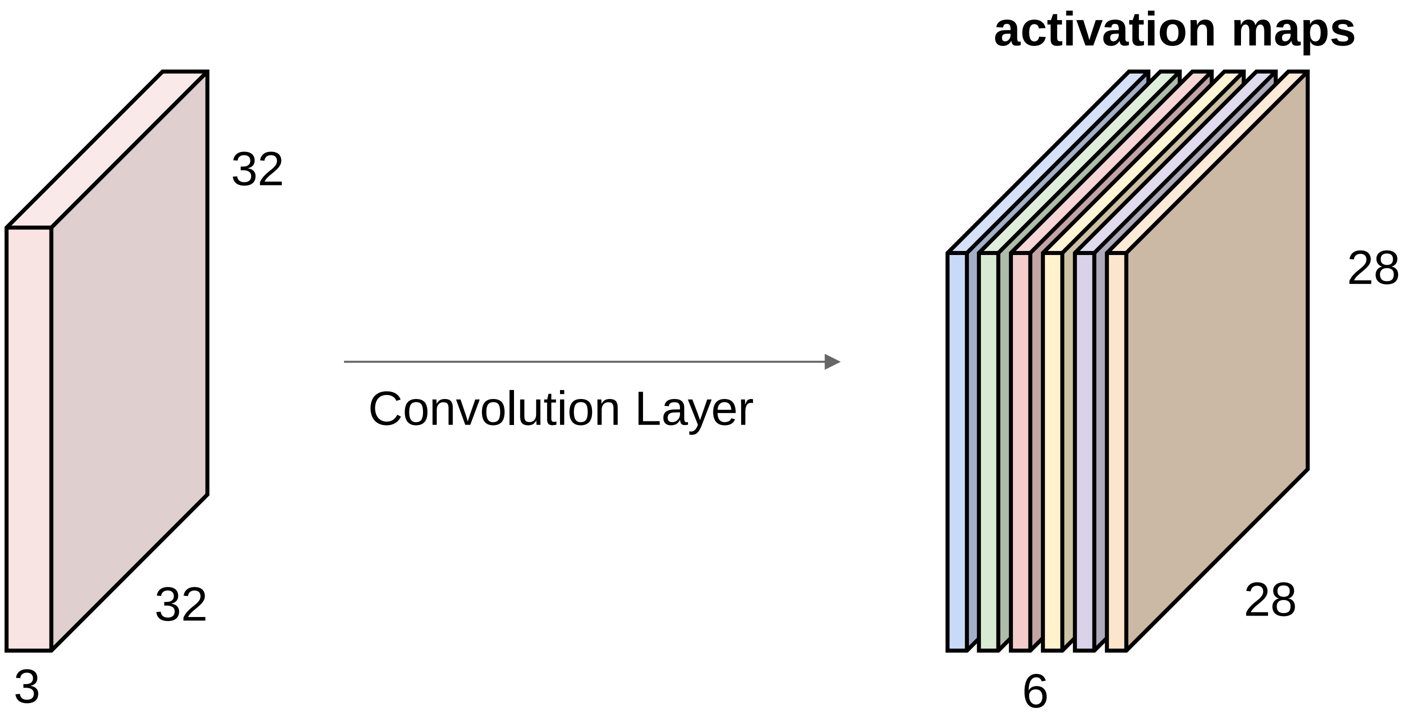


Convolution Layer

Un segundo filtro

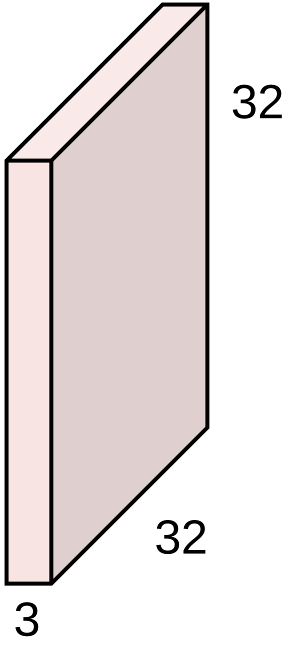


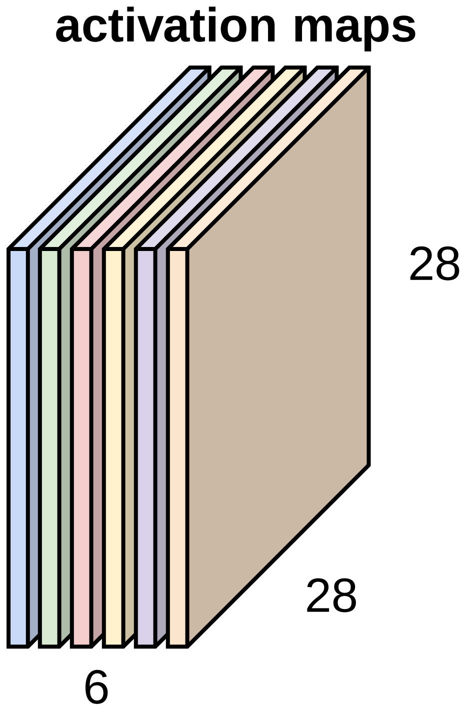
Convolution Layer



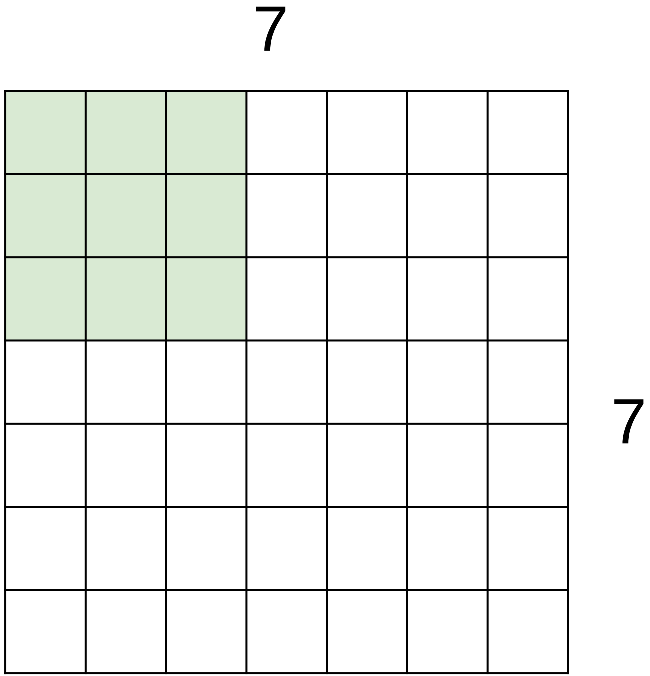
Si tenemos 6 filtros, el resultado tendría la forma: 28x28x6

Convolution Layer

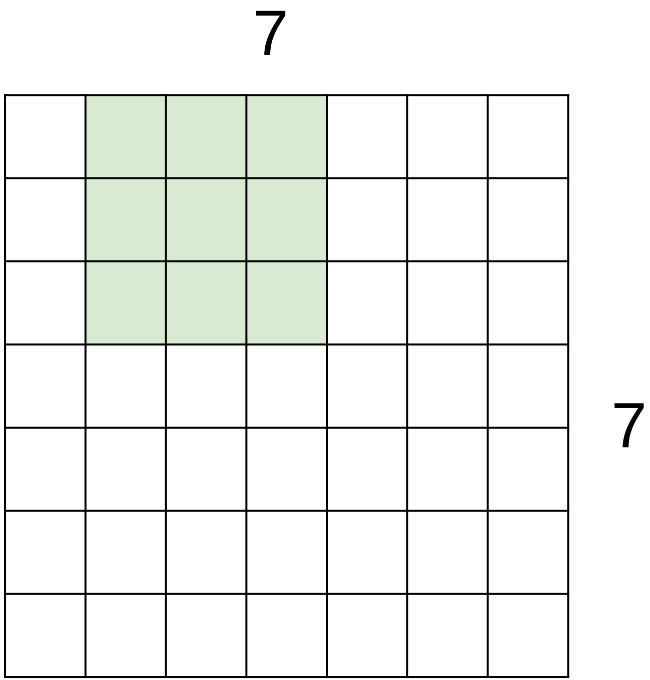




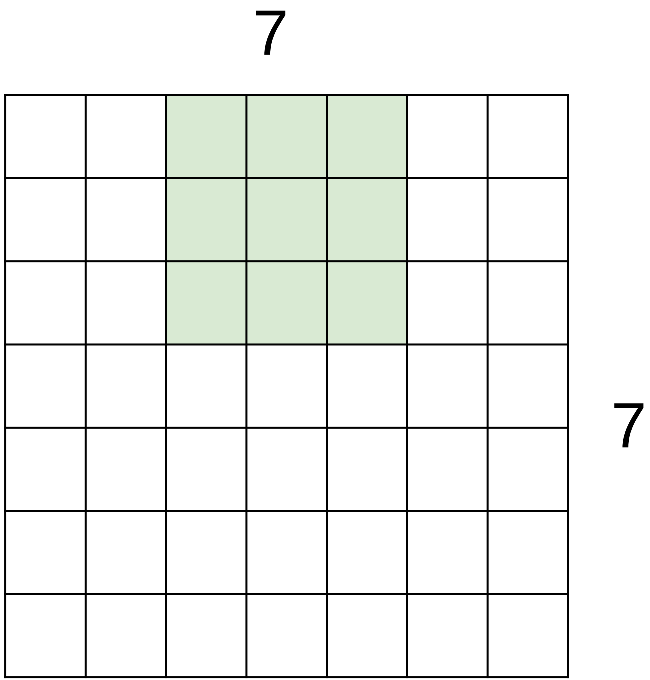
- Kernel size = 5
- # kernels = 6
- padding = 0



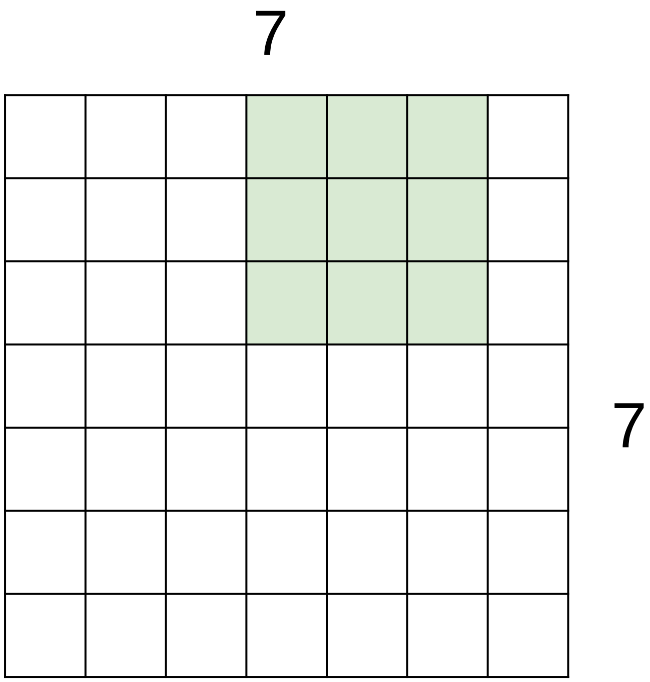
7x7 input
3x3 filter



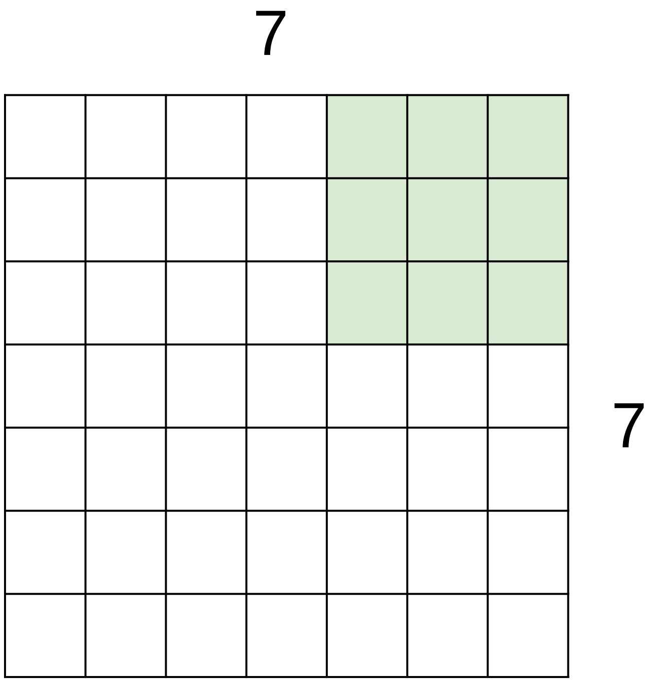
7x7 input
3x3 filter



7x7 input
3x3 filter



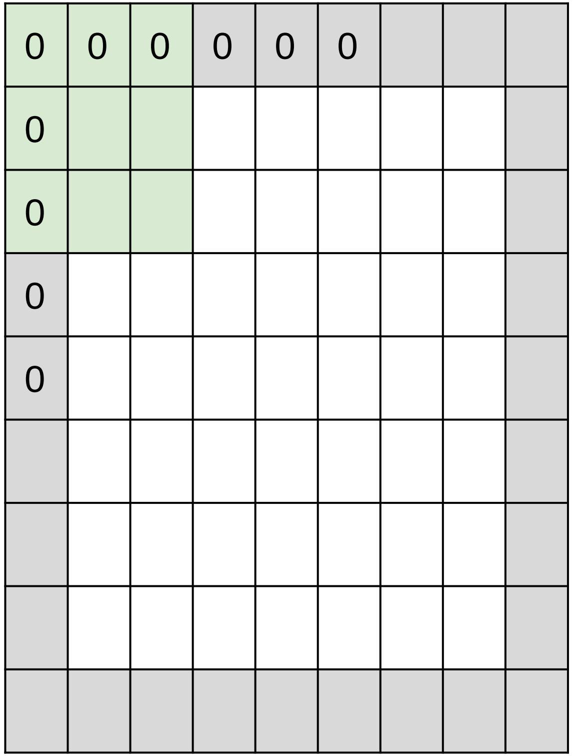
7x7 input
3x3 filter



7x7 input
3x3 filter

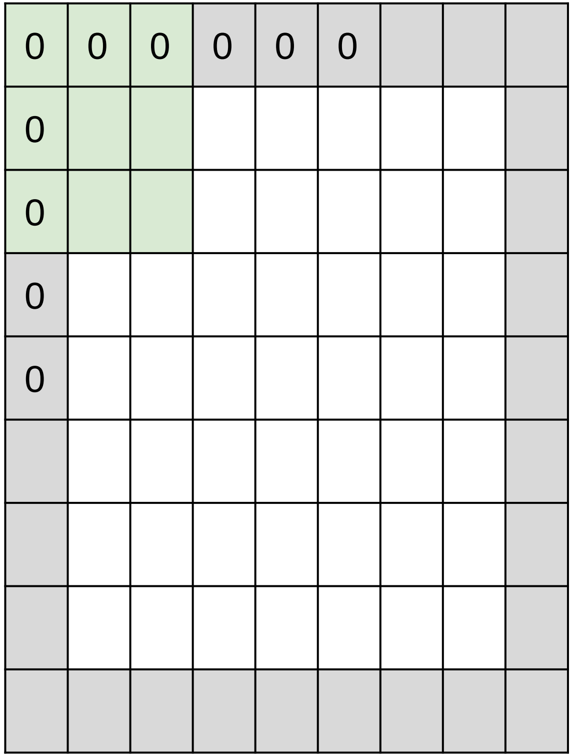
=> 5x5 output

Padding



input 7x7
3x3 filter
padding 1

Padding

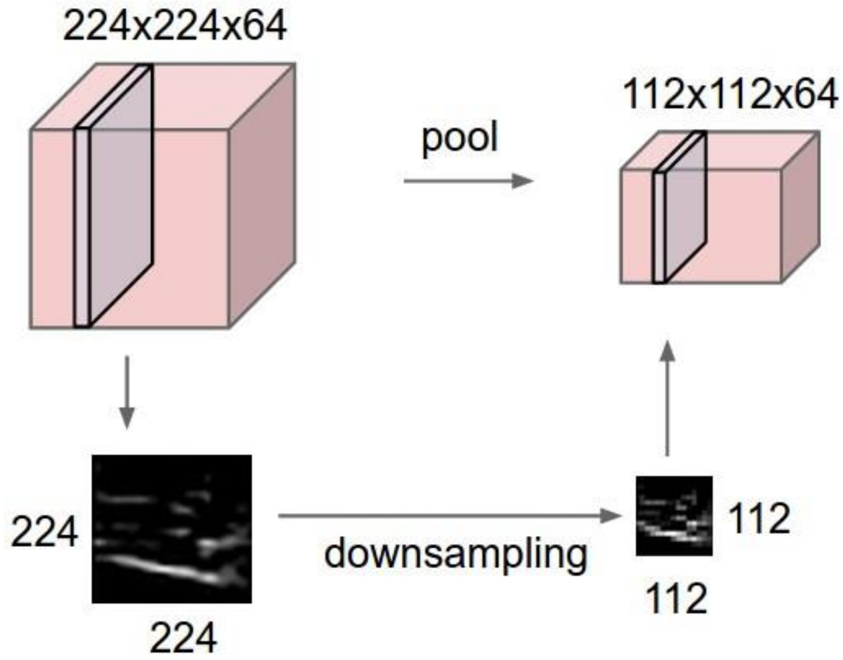


input 7x7
3x3 filter
padding 1

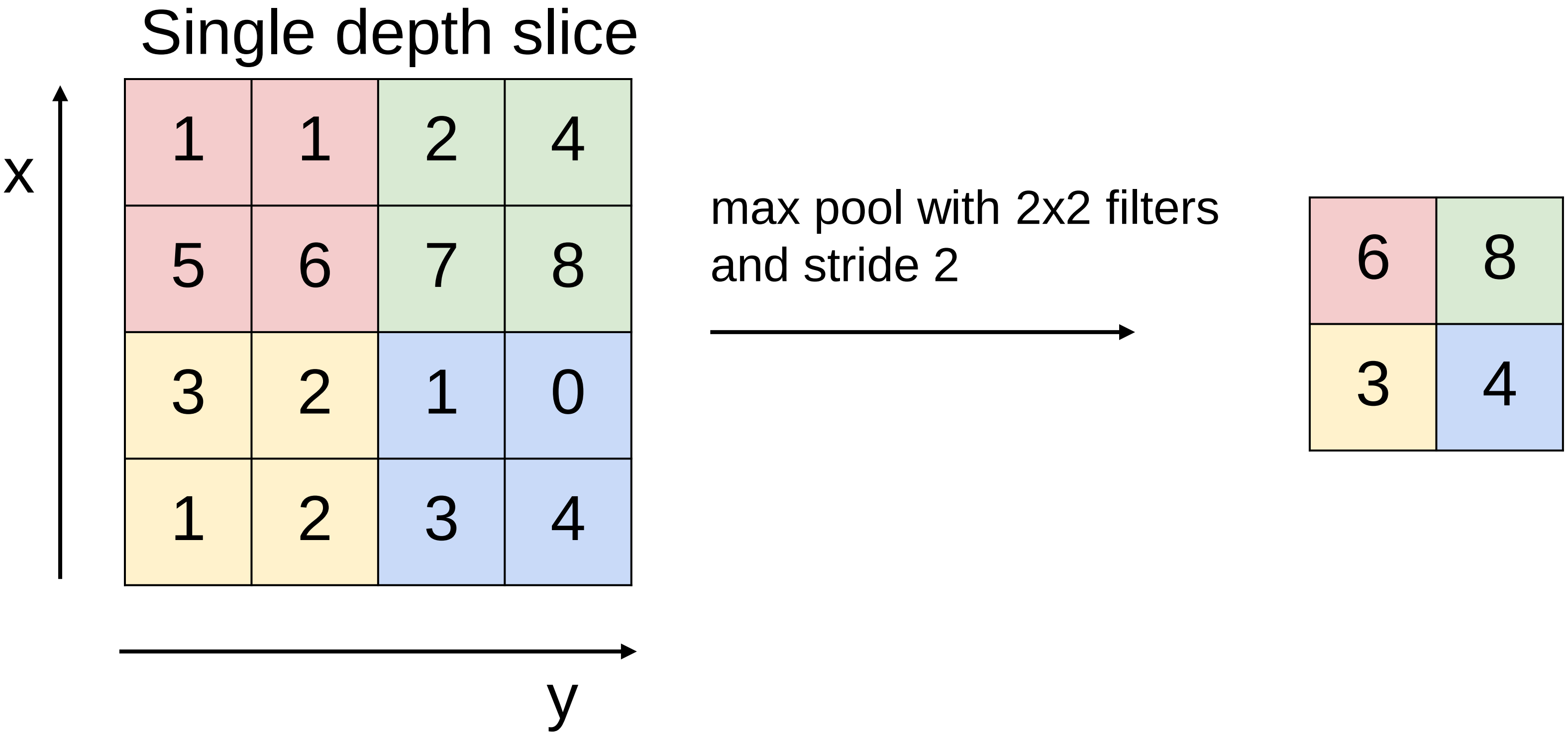
7x7 output!

https://ezyang.github.io/convolution-visualizer/index.html

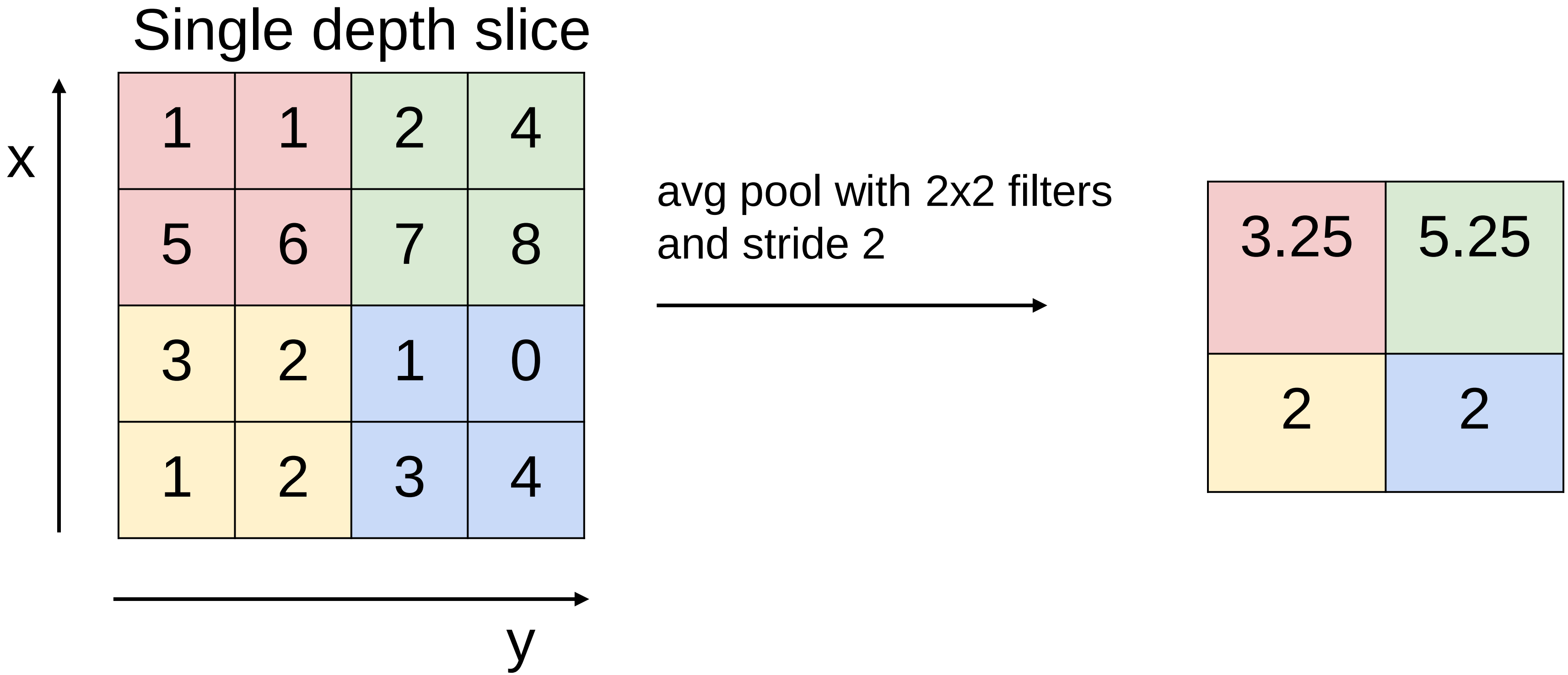
Pooling layer



Max Pooling


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

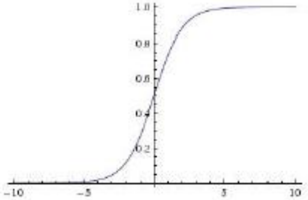
Avg Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
avg pool with 2x2 filters
and stride 2
3.25 5.25
2 2

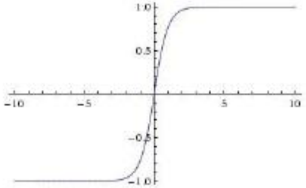
Activation Function

Sigmoid

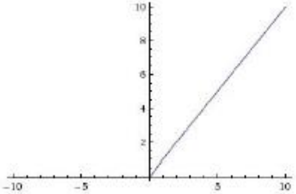
$$\sigma(x) = 1/(1+e^{-x})$$



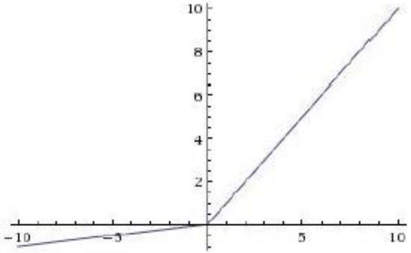
tanh tanh(x)



ReLU max(0,x)

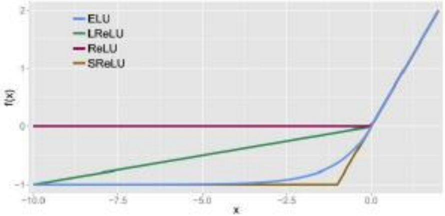


Leaky ReLU
max(0.1x, x)



ELU

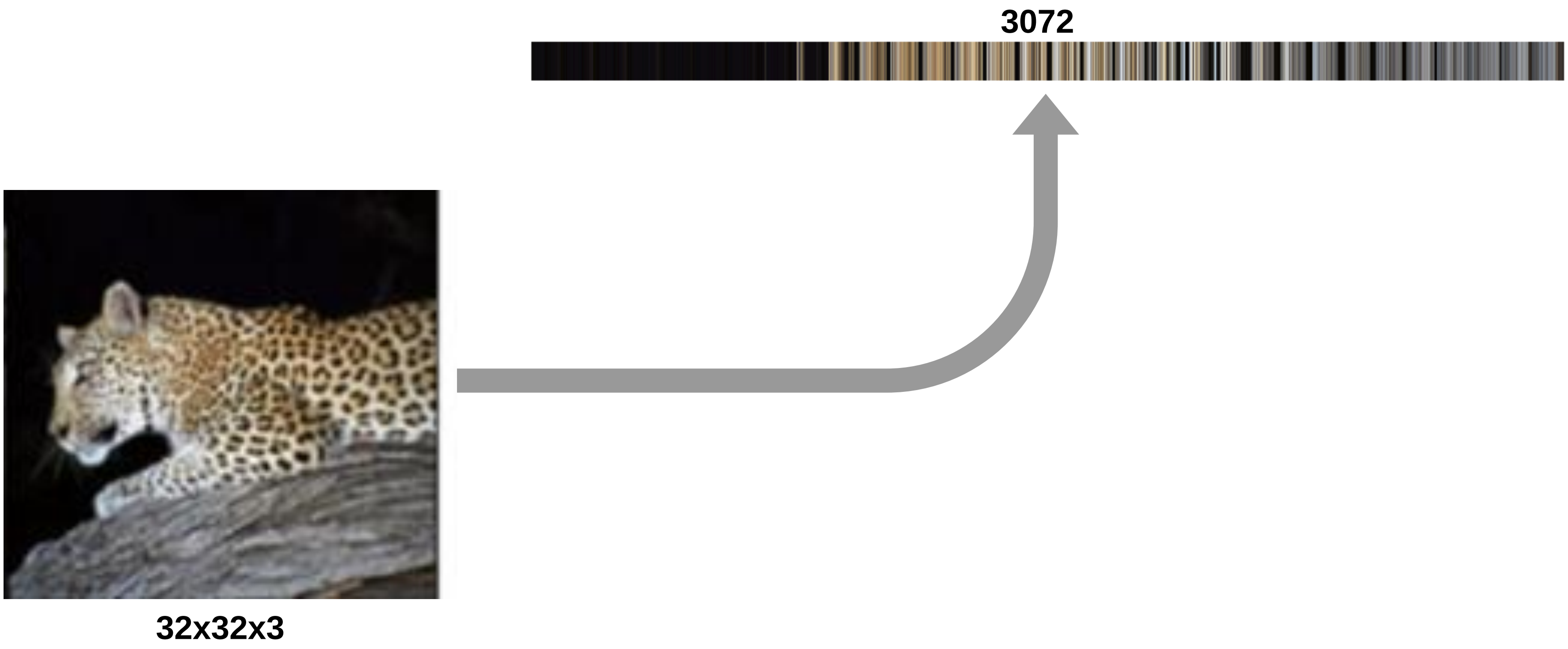
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

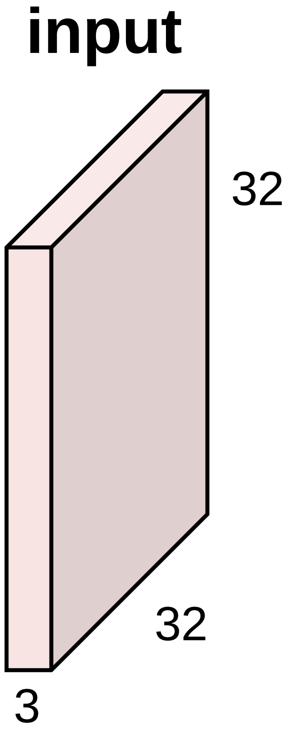


Fully Connected Layer

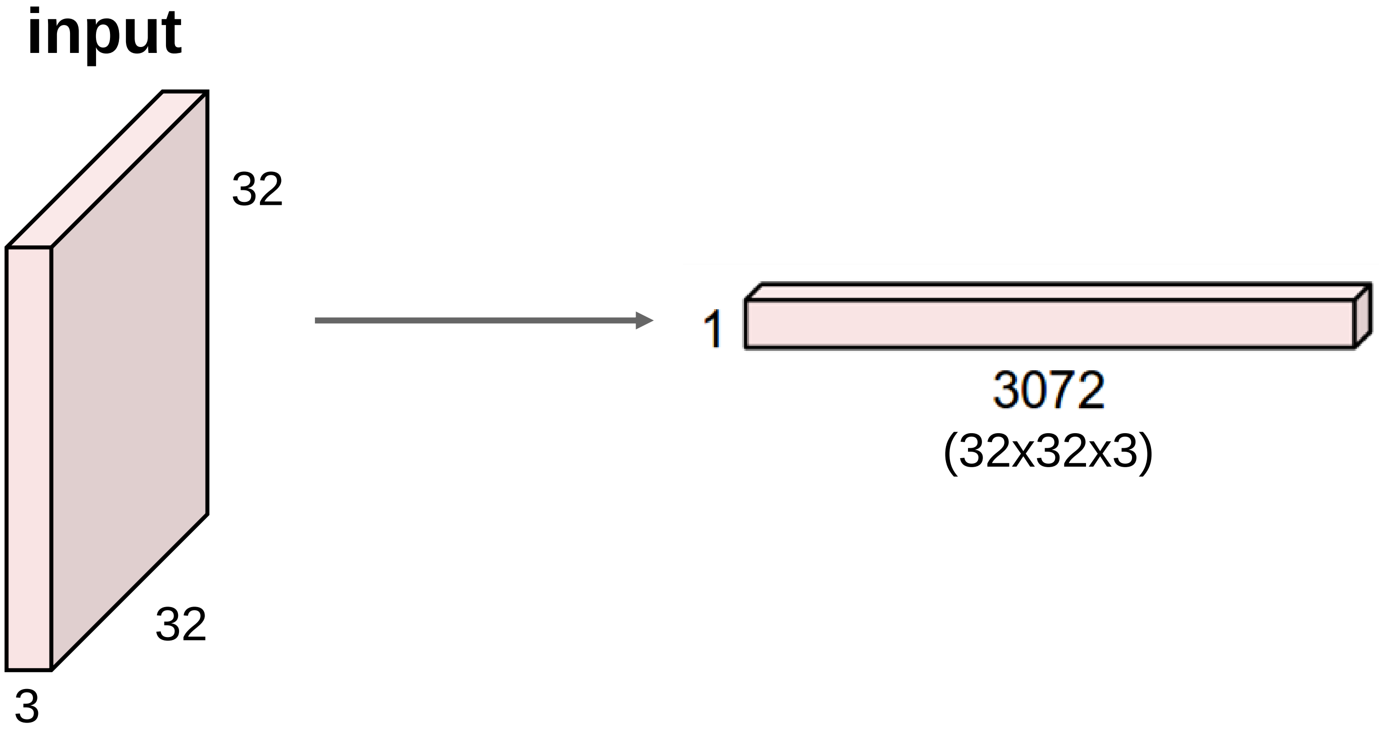
32x32x3

Fully Connected Layer

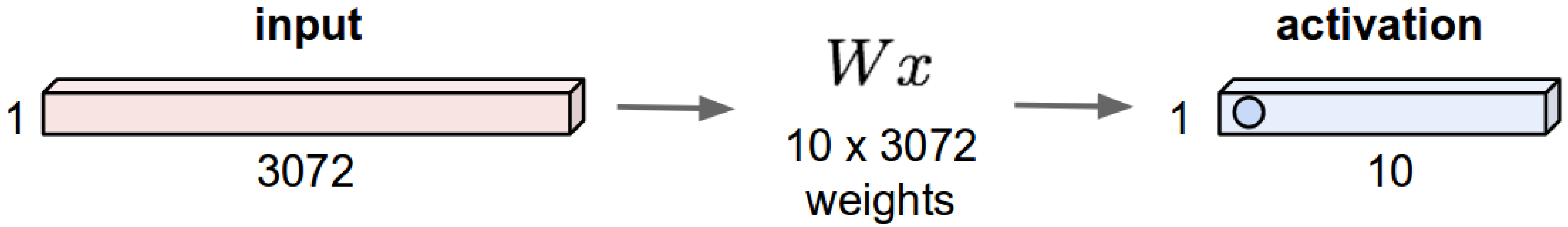


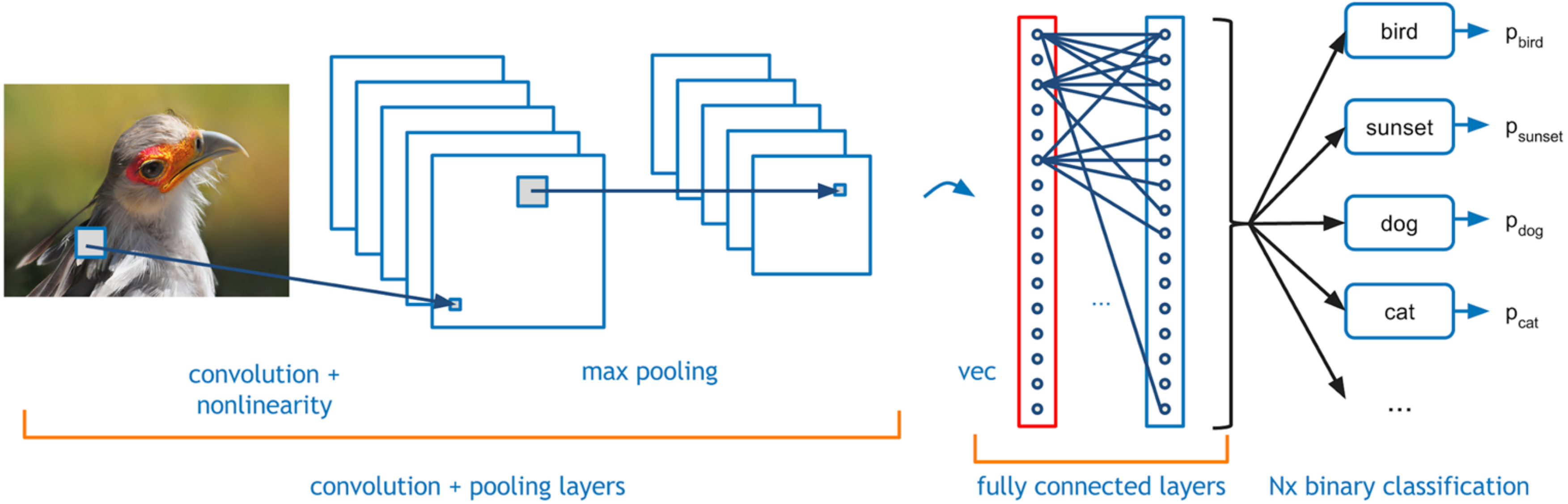

input
32
32
3

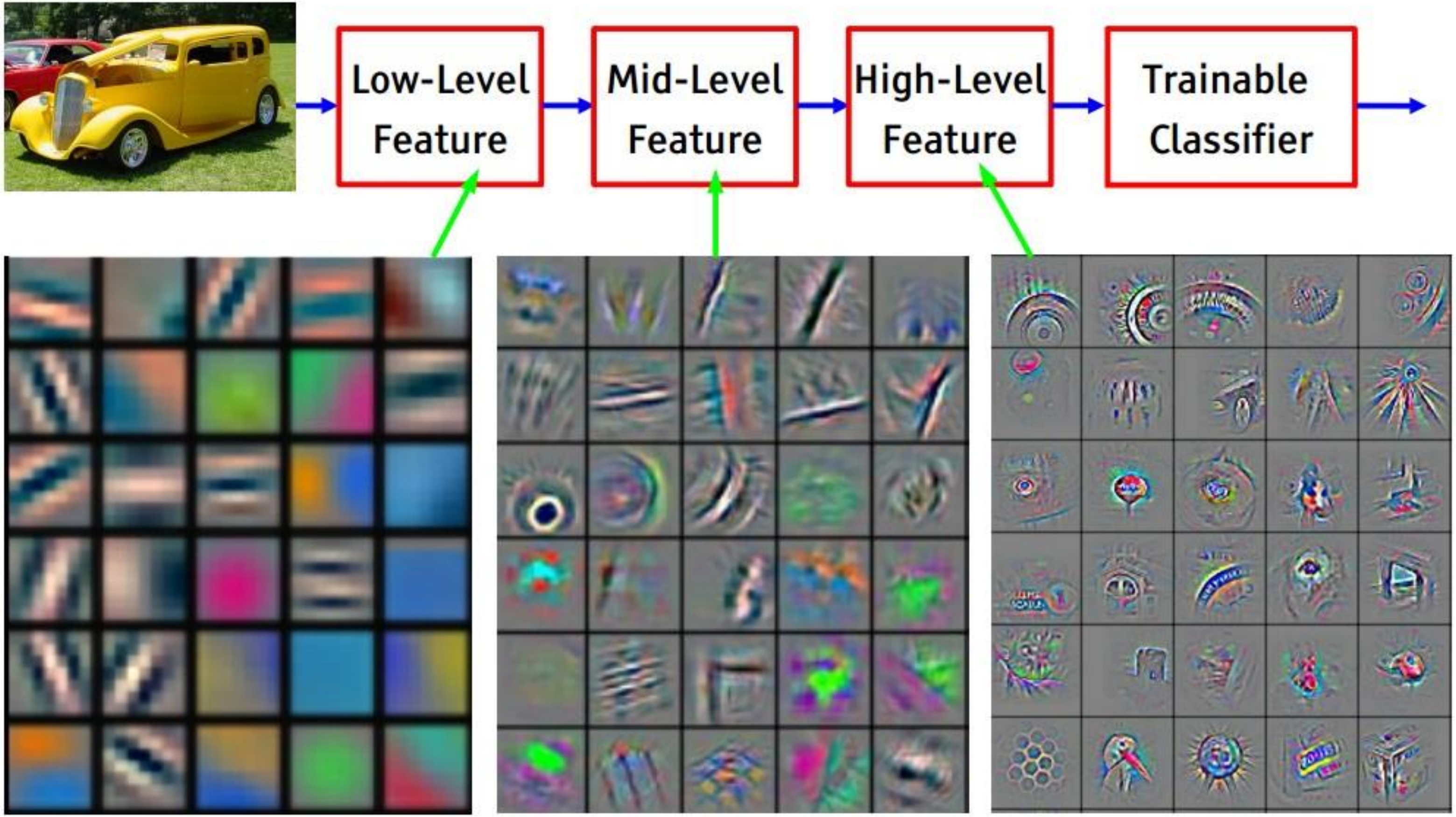
Fully Connected Layer



Fully Connected Layer




convolution +
nonlinearity
max pooling
convolution + pooling layers
vec
...
fully connected layers
bird
sunset
dog
cat
...
p bird
p sunset
p dog
p cat
Nx binary classification



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

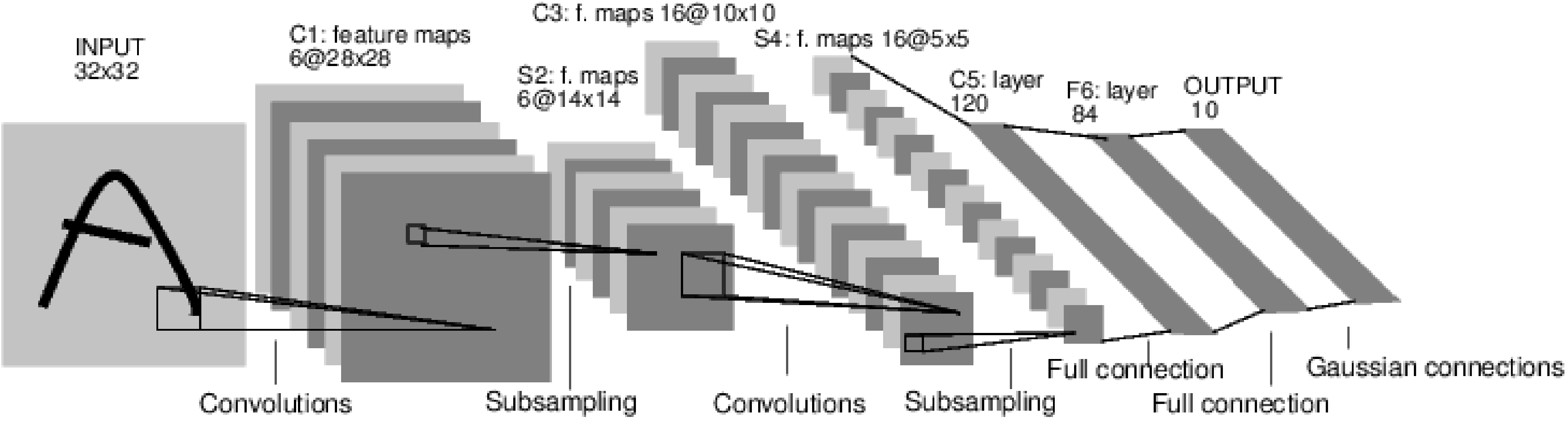
Keras code

```
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten

model = Sequential([
    Conv2D(16, 3, activation='relu', input_shape=(28,28,1)),
        MaxPool2D(),
        Conv2D(32, 3, activation='relu'),
        MaxPool2D(),
        Flatten(),
        Dense(10, activation='softmax')
])
```

Arquitecturas conocidas

LeNet-5

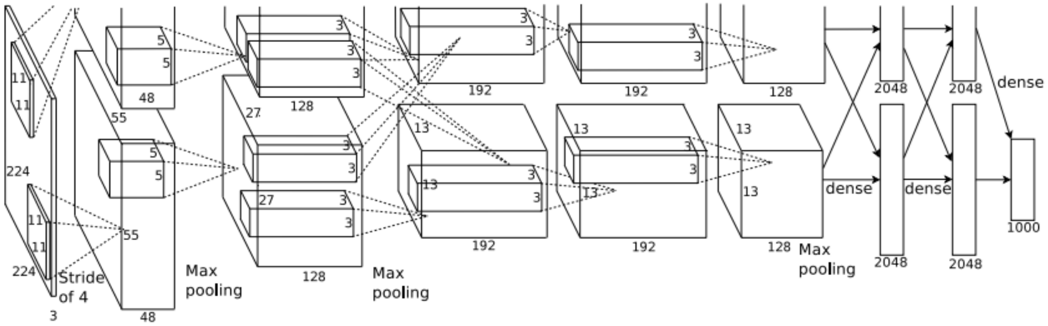
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

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%**

VGGNet

[Simonyan and Zisserman, 2014]

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

best model

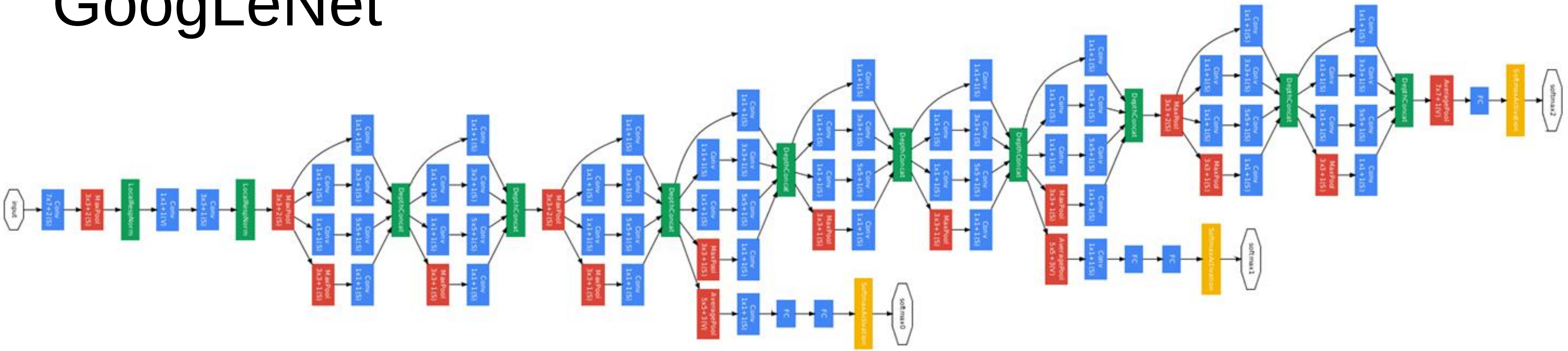
7.3% top 5 error

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					

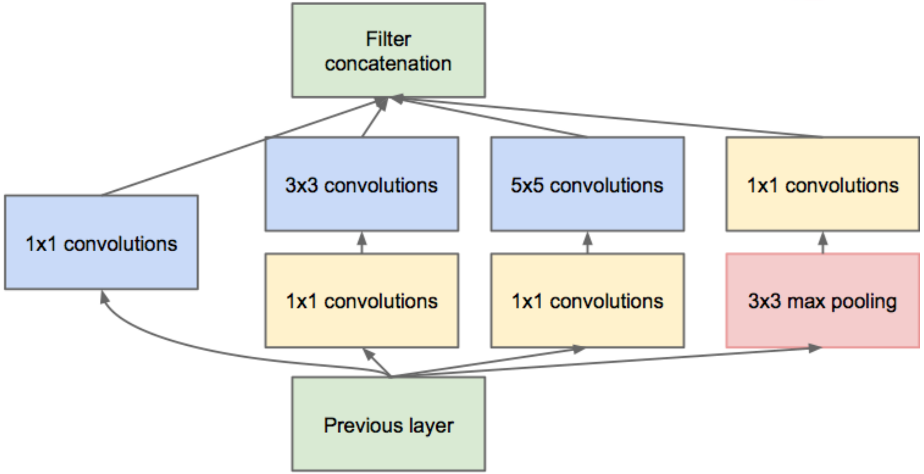
Table 2: **Number of parameters** (in millions).

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

GoogLeNet



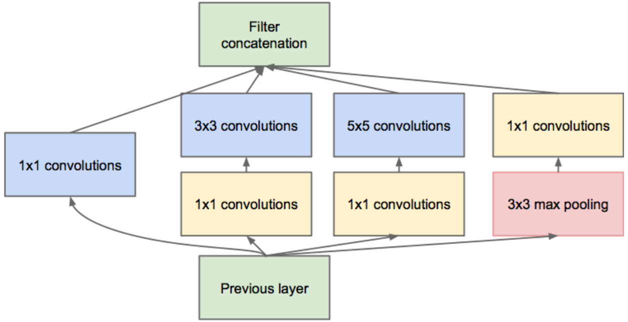
[Szegedy et al., 2014]



Inception module

ILSVRC 2014 winner (6.7% top 5 error)

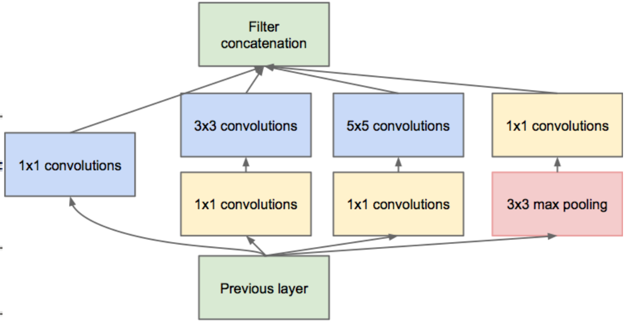
Inception module (Keras code)




```
from tensorflow.keras.layers import Conv2D, MaxPool2D, concatenate

tower_1 = Conv2D(64, 1, padding='same', activation='relu')(input_img)

tower_2 = Conv2D(64, 1, padding='same', activation='relu')(input_img)
tower_2 = Conv2D(64, 3, padding='same', activation='relu')(tower_1)

tower_3 = Conv2D(64, 1, padding='same', activation='relu')(input_img)
tower_3 = Conv2D(64, 5, padding='same', activation='relu')(tower_2)

tower_4 = MaxPool2D(3, strides=(1,1), padding='same')(input_img)
tower_4 = Conv2D(64, 1, padding='same', activation='relu')(tower_3)

output = concatenate([tower_1, tower_2, tower_3, tower_4], axis = 3)
```

Inception module (Keras code)

```
Layer (type)                     Output Shape          Param #     Connected to
==================================================================================================
input (InputLayer)               (None, 112, 112, 3)   0

tower_2_1 (Conv2D)               (None, 112, 112, 64)  256         input[0][0]

tower_3_1 (Conv2D)               (None, 112, 112, 64)  256         input[0][0]

tower_4_1 (MaxPooling2D)         (None, 112, 112, 3)   0           input[0][0]

tower_1_1 (Conv2D)               (None, 112, 112, 64)  256         input[0][0]

tower_2_2 (Conv2D)               (None, 112, 112, 64)  36928       tower_2_1[0][0]

tower_3_2 (Conv2D)               (None, 112, 112, 64)  102464      tower_3_1[0][0]

tower_4_2 (Conv2D)               (None, 112, 112, 64)  256         tower_4_1[0][0]

concatenate_12 (Concatenate)     (None, 112, 112, 256  0           tower_1_1[0][0]
                                                                   tower_2_2[0][0]
                                                                   tower_3_2[0][0]
                                                                   tower_4_2[0][0]
==================================================================================================
```

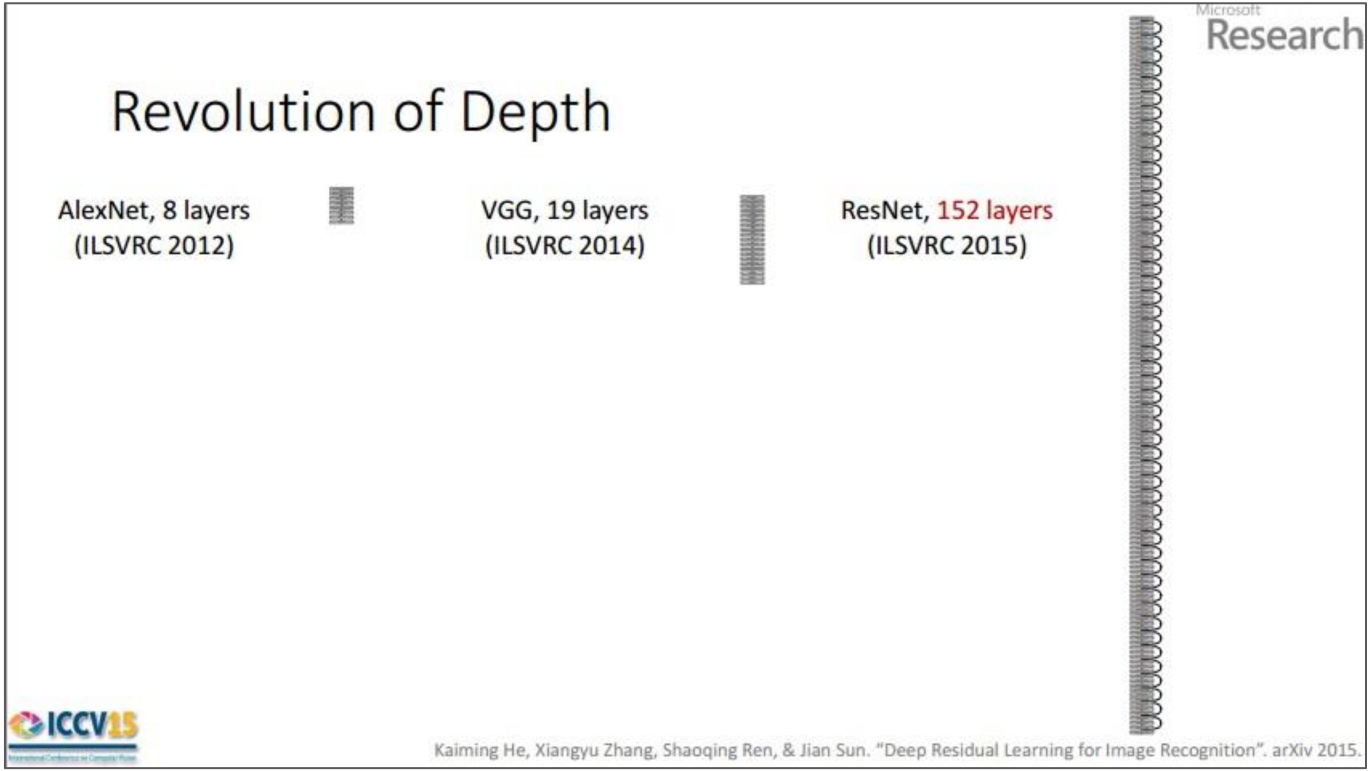


GoogLeNet

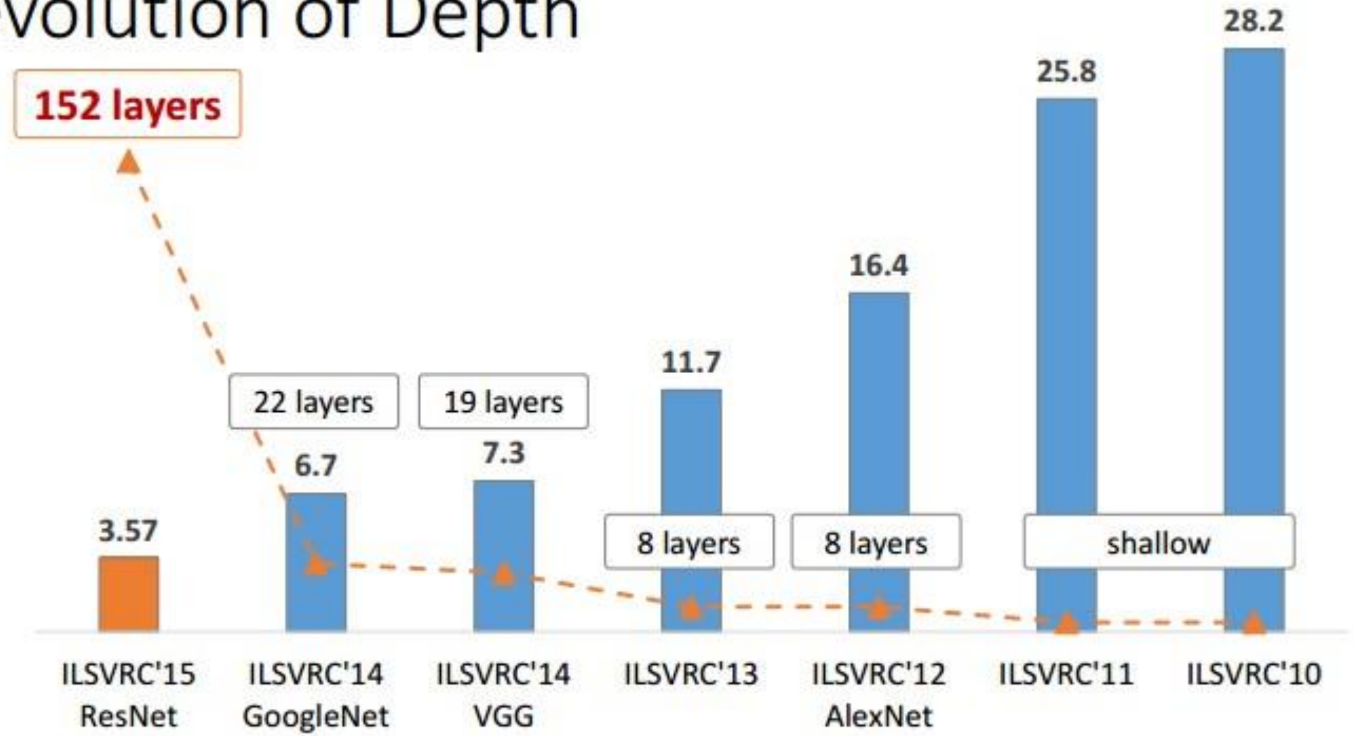
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)



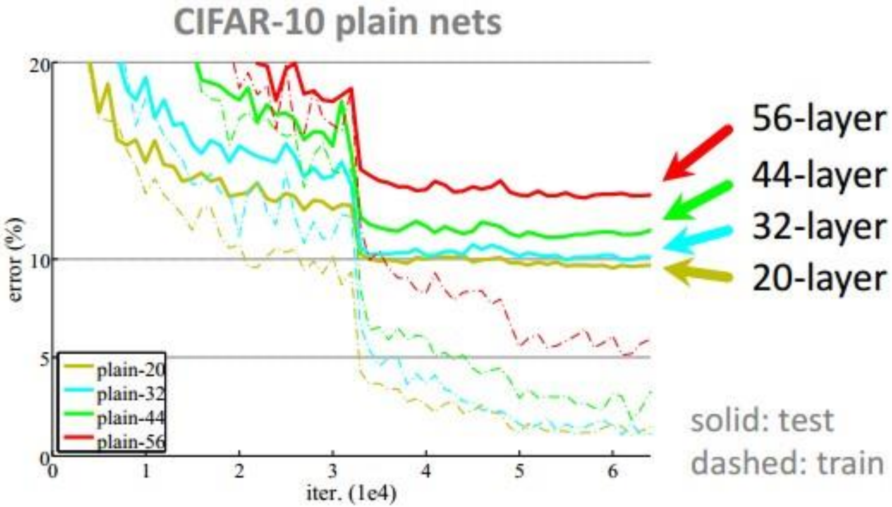
Revolution of Depth

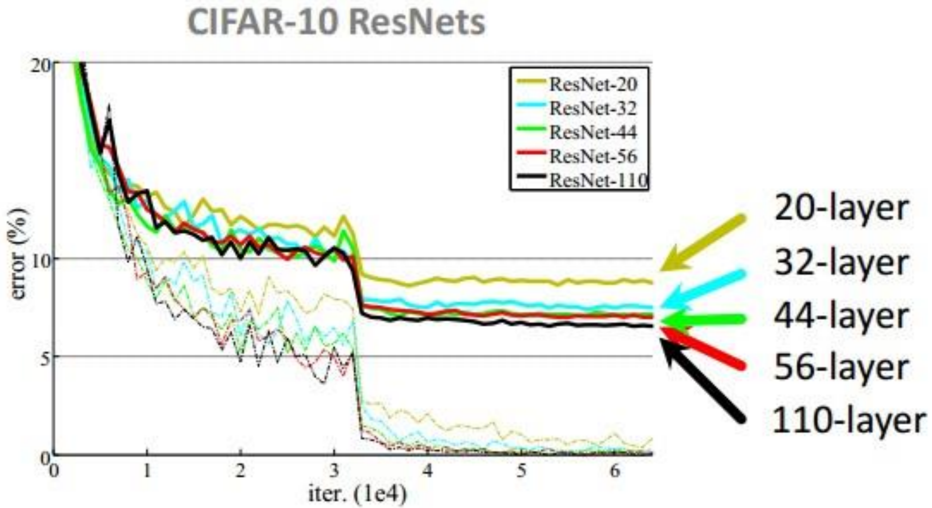


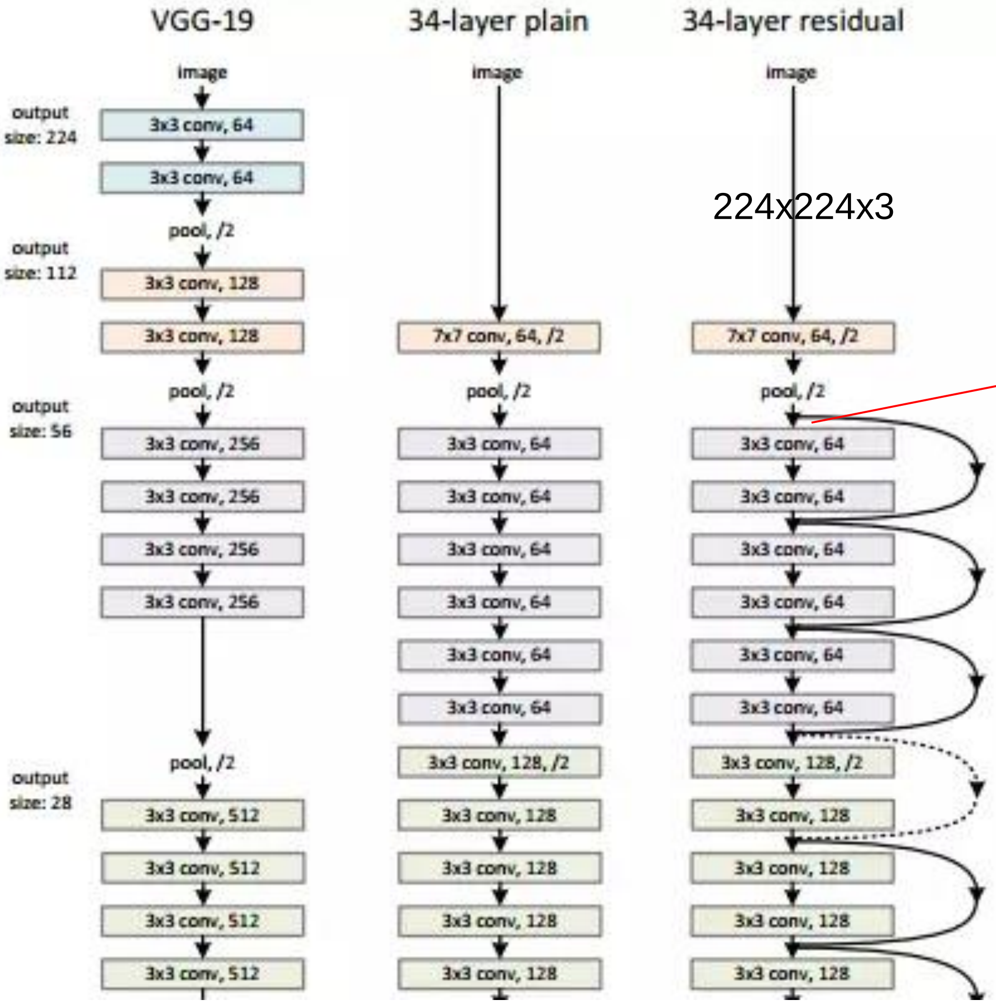
ImageNet Classification top-5 error (%)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

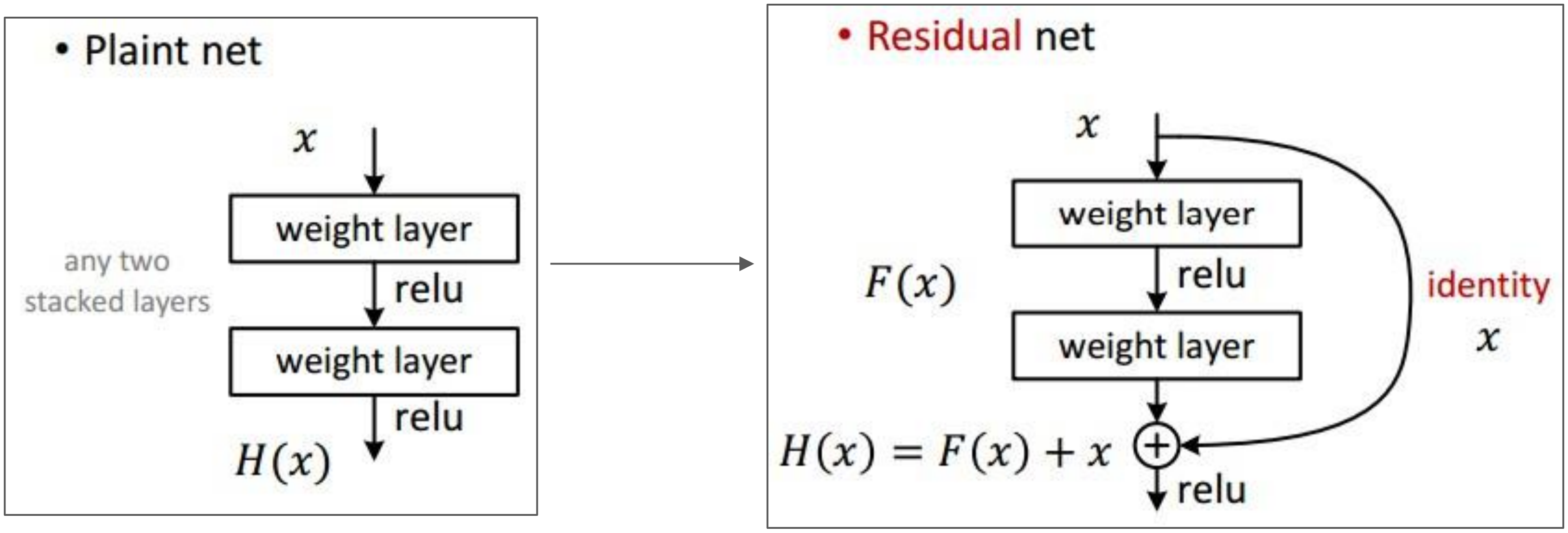
CIFAR-10 experiments



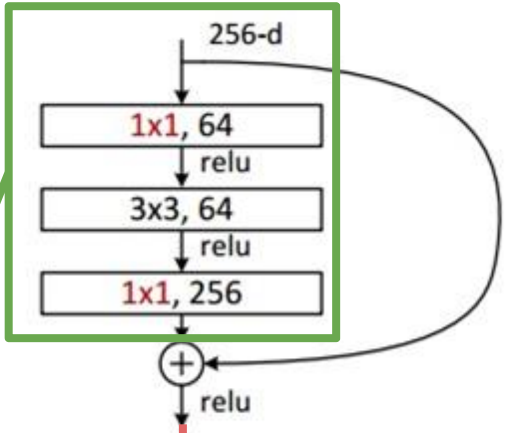




spatial dimension
only 56x56!


• Plaint net
x
weight layer
any two
stacked layers
relu
weight layer
relu
$H(x)$
• Residual net
x
weight layer
$F(x)$
relu
identity
weight layer
x
$H(x) = F(x) + x$
relu

```python
def identity_block(input_tensor, kernel_size, filters, stage, block):
    """The identity block is the block that has no conv layer at shortcut.

    # Arguments
        input_tensor: input tensor
        kernel_size: default 3, the kernel size of middle conv layer at main path
        filters: list of integers, the filterss of 3 conv layer at main path
        stage: integer, current stage label, used for generating layer names
        block: 'a','b'..., current block label, used for generating layer names

    # Returns
        Output tensor for the block.
    """
    filters1, filters2, filters3 = filters
    if K.image_data_format() == 'channels_last':
        bn_axis = 3
    else:
        bn_axis = 1
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = Conv2D(filters1, (1, 1), name=conv_name_base + '2a')(input_tensor)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
    x = Activation('relu')(x)

    x = Conv2D(filters2, kernel_size,
               padding='same', name=conv_name_base + '2b')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2b')(x)
    x = Activation('relu')(x)

    x = Conv2D(filters3, (1, 1), name=conv_name_base + '2c')(x)
    x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2c')(x)

    x = layers.add([x, input_tensor])
    x = Activation('relu')(x)
    return x
```



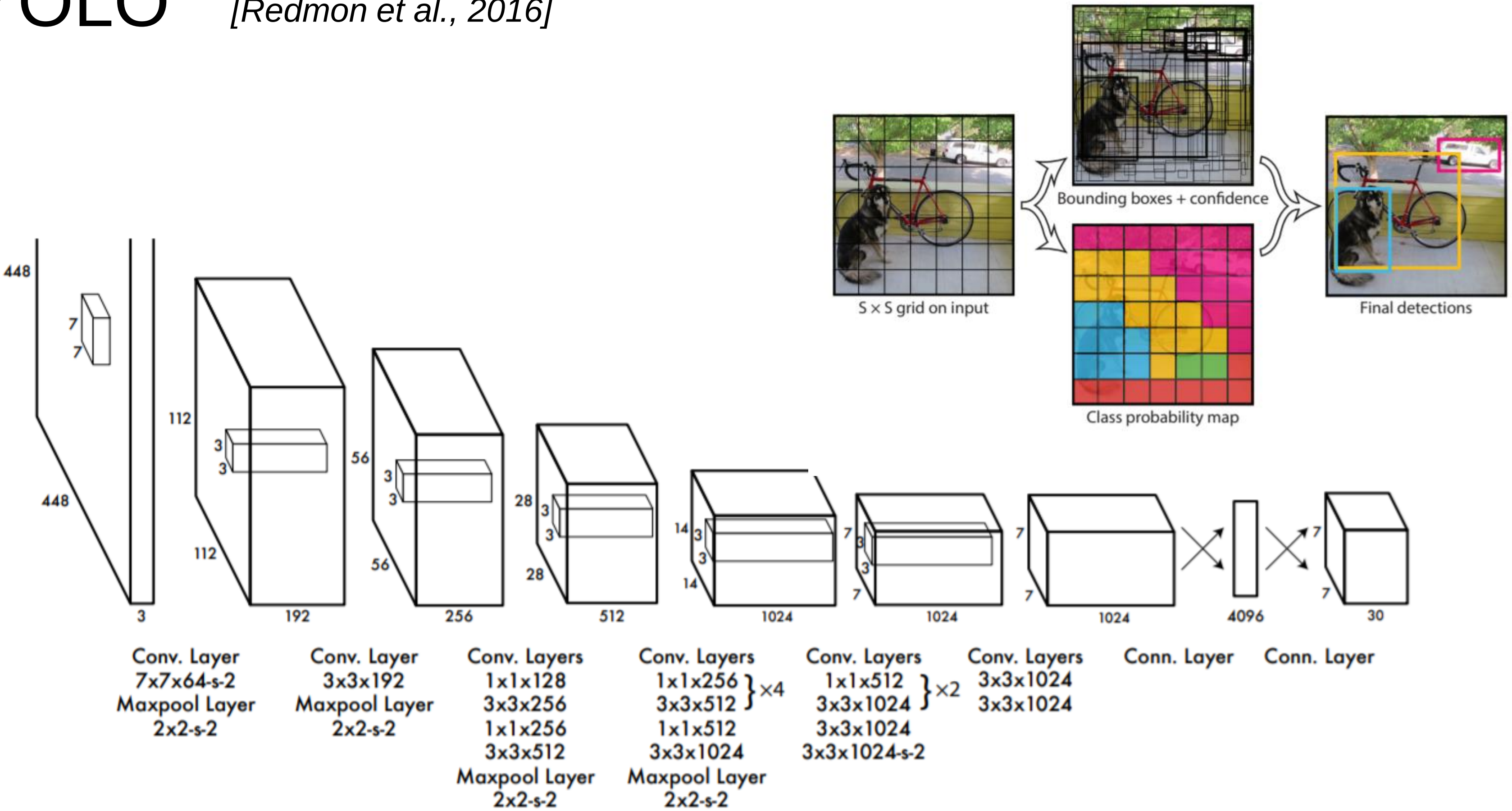
ResNet *[He et al., 2015]*

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

ResNet *[He et al., 2015]*

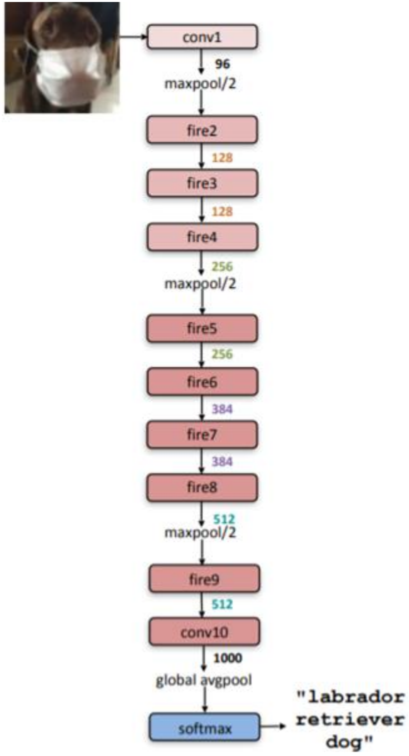
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

YOLO *[Redmon et al., 2016]*



SqueezeNet

[Iandola et al., 2017]



CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations			parameters			compression info				1,248,424 (total)	**421,098** (total)

Thank You !

Susan Palacios Salcedo
PhD Candidate,
PUCP
spalacios@pucp.edu.pe