Técnicas Avanzadas de Data Mining y Sistemas Inteligentes

Maestría en Informática
Escuela de Posgrado
Pontificia Universidad Católica del Perú

2018-2

Renato Hermoza - renato.hermoza@pucp.edu.pe

Review

```
Dense(200, activation='relu')

Dropout(0.2)

Dense(100, activation='relu')

Dropout(0.2)

Dense(100, activation='relu')

Dropout(0.2)

Dense(10, activation='softmax')

Dropout(0.2)
```

```
Dense(200, activation='relu')

Dropout(0.2)

Dense(100, activation='relu')

Dropout(0.2)

Dense(100, activation='relu')

Dropout(0.2)

Dense(10, activation='softmax')

Dropout(0.2)
```

```
Dense(200, activation='relu', kernel_initializer='glorot_normal’)

Dense(100, activation='relu', kernel_initializer='glorot_normal’)

Dense(100, activation='relu', kernel_initializer='glorot_normal’)

Dense(10, activation='softmax', kernel_initializer='glorot_normal’)
```

```
Dense(200, activation='relu', kernel_initializer='he_normal')

Dense(100, activation='relu', kernel_initializer='he_normal')

Dense(100, activation='relu', kernel_initializer='he_normal')

Dense(10, activation='softmax', kernel_initializer='glorot_normal’)
```

```
Dense(200, activation='relu', kernel_initializer='he_normal')

BatchNormalization()

Dense(100, activation='relu', kernel_initializer='he_normal')

BatchNormalization()

Dense(100, activation='relu', kernel_initializer='he_normal')

BatchNormalization()

Dense(10, activation='softmax')

BatchNormalization()
```

```
Dense(200, activation='relu', kernel_initializer='he_normal', use_bias=False)

BatchNormalization()

Dense(100, activation='relu', kernel_initializer='he_normal', use_bias=False)

BatchNormalization()

Dense(100, activation='relu', kernel_initializer='he_normal', use_bias=False)

BatchNormalization()

Dense(10, activation='softmax')

BatchNormalization()
```

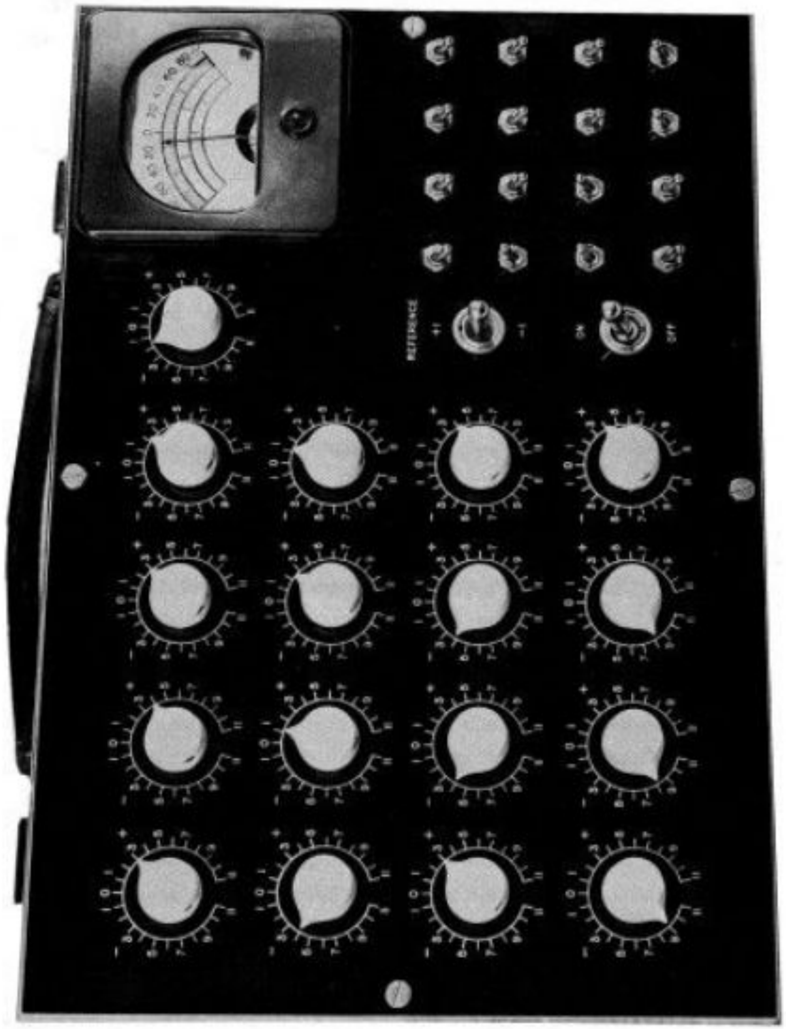
History Review

Mark I Perceptron

Frank Rosenblatt ~1957

Adeline/Madeline

Widrow and Hoff ~1960



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

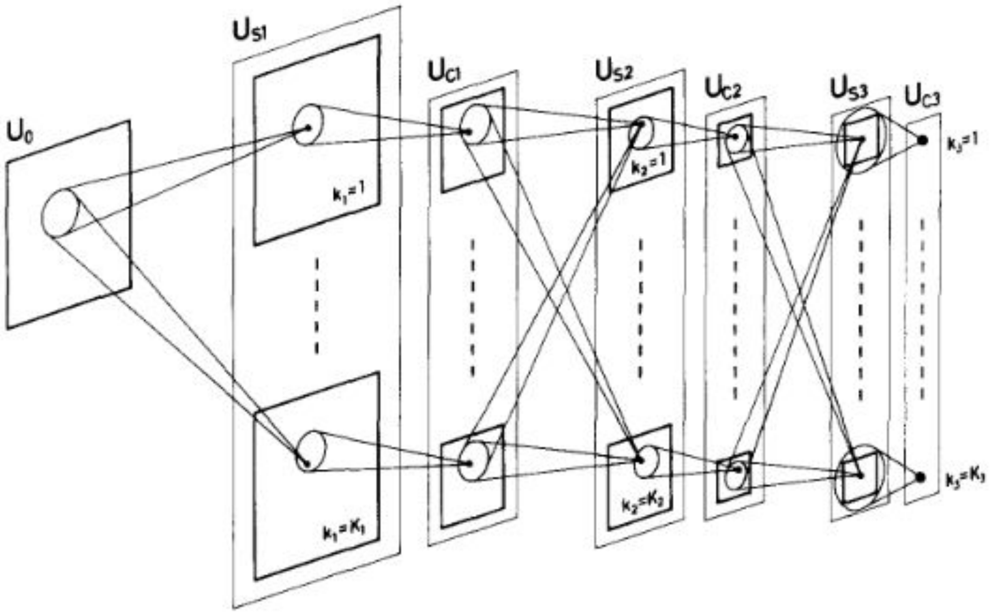


Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

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

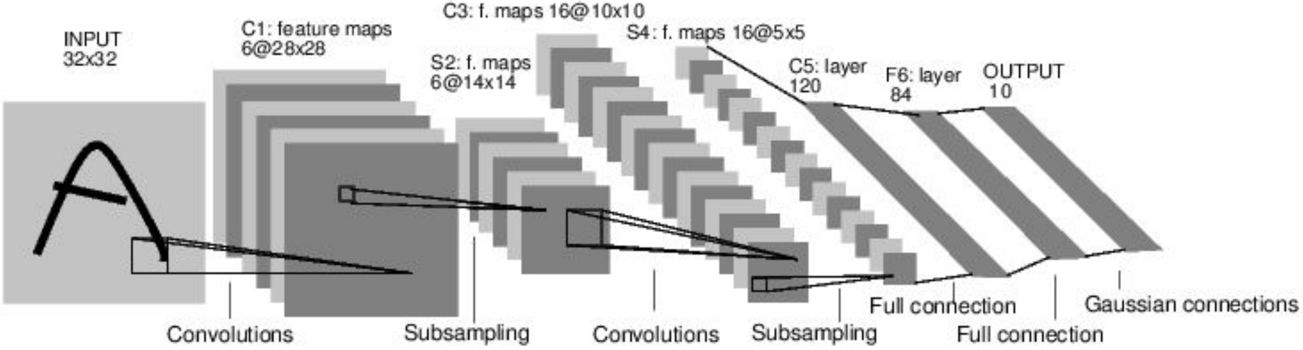
$$\partial E/\partial y_j = y_j - d_j \tag{4}$$

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

$$\partial E/\partial x_j = \partial E/\partial y_j \cdot dy_j/dx_j$$

Learning representations by back-propagating errors

Rumelhart et. al., 1986



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

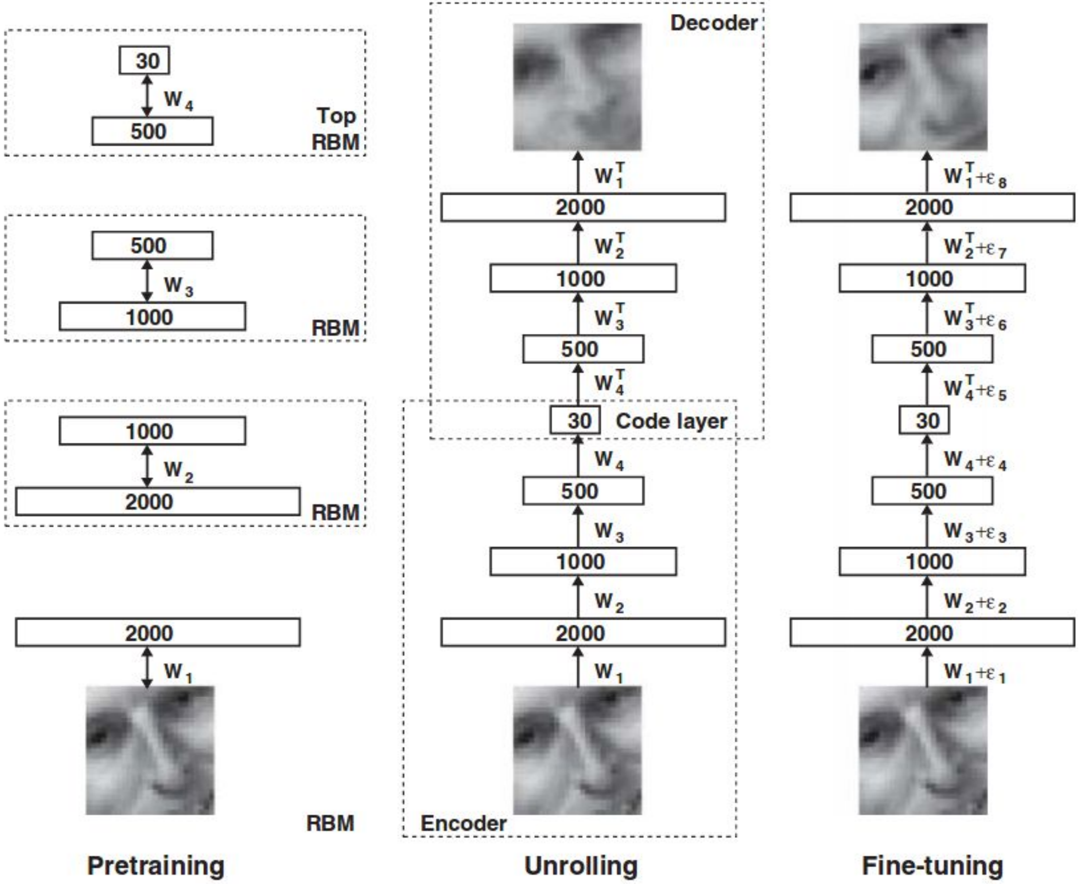
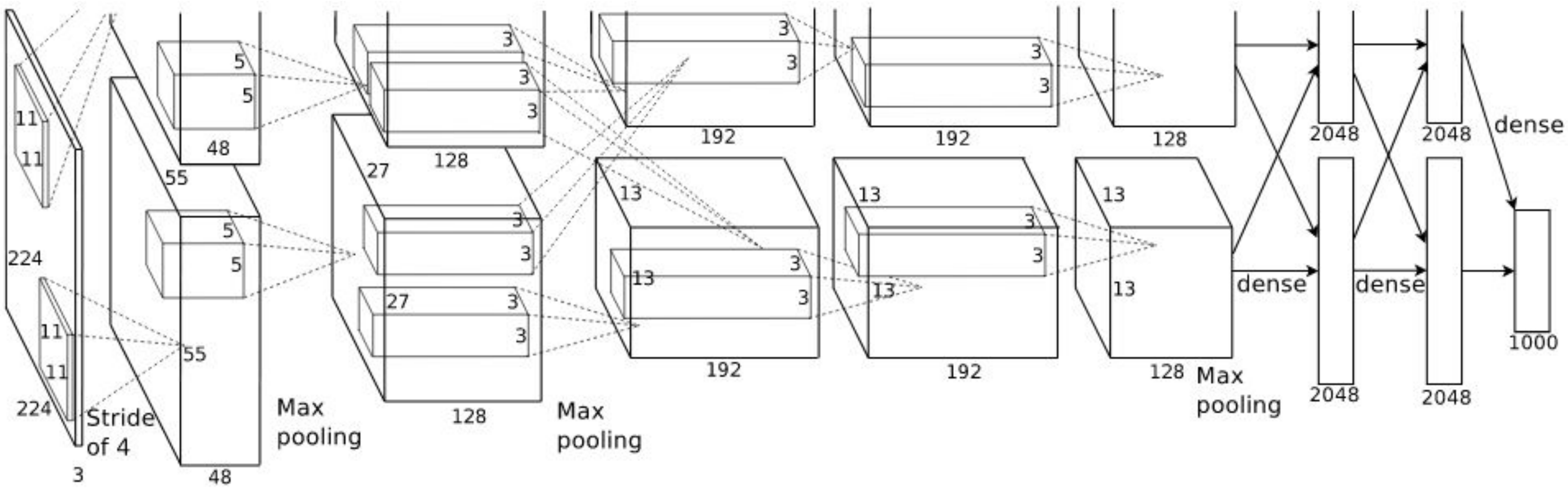


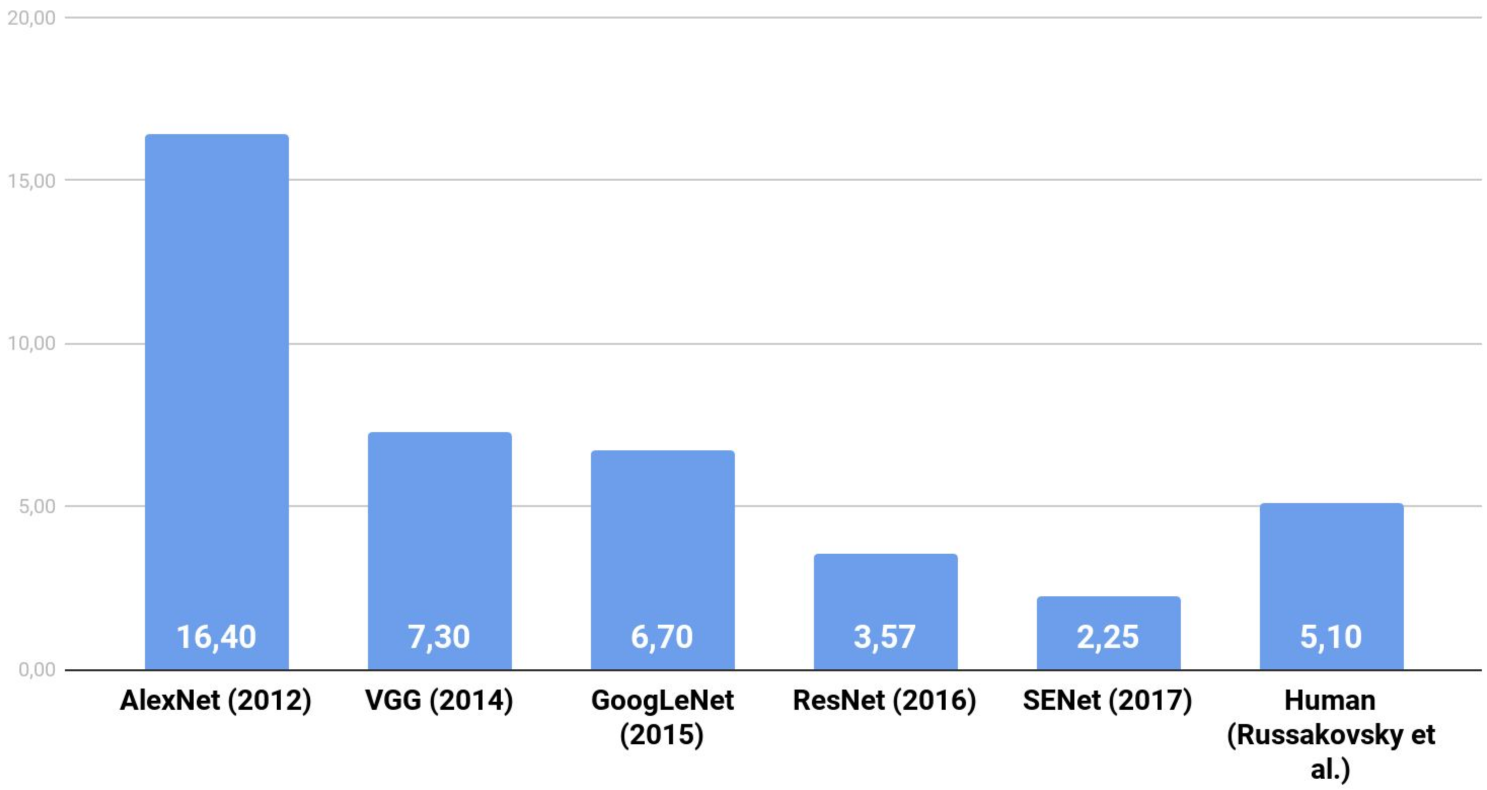
Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create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



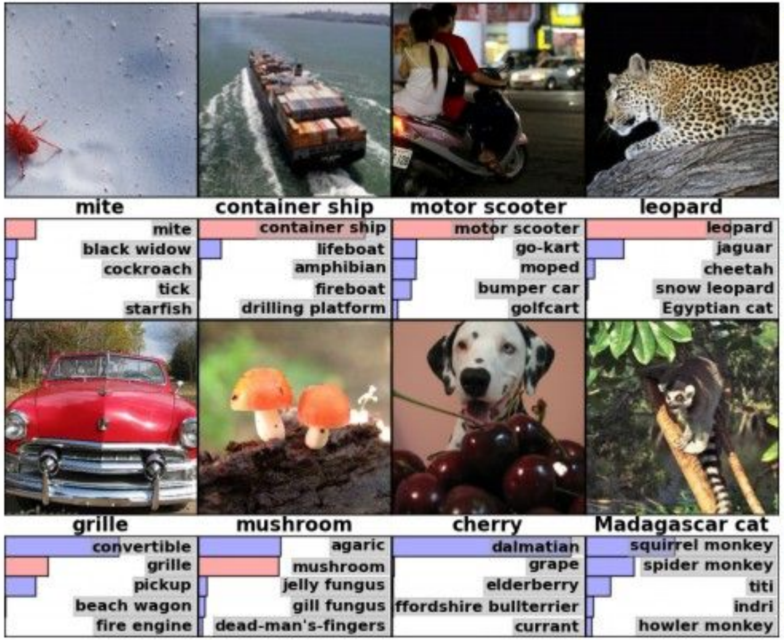
Imagenet classification with deep convolutional neural networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012

ImageNet Top 5 Error Rate



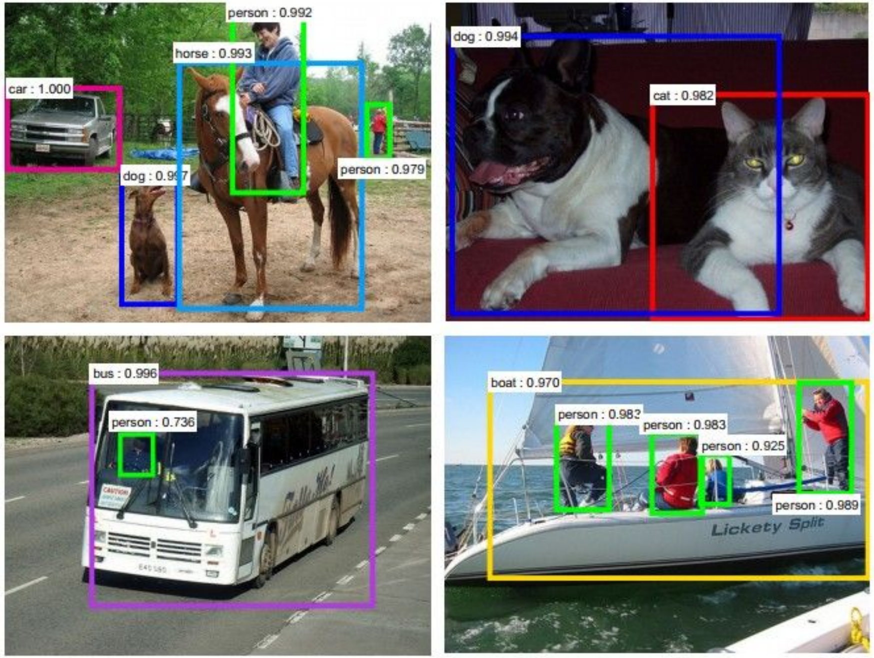
Classification



Retrieval

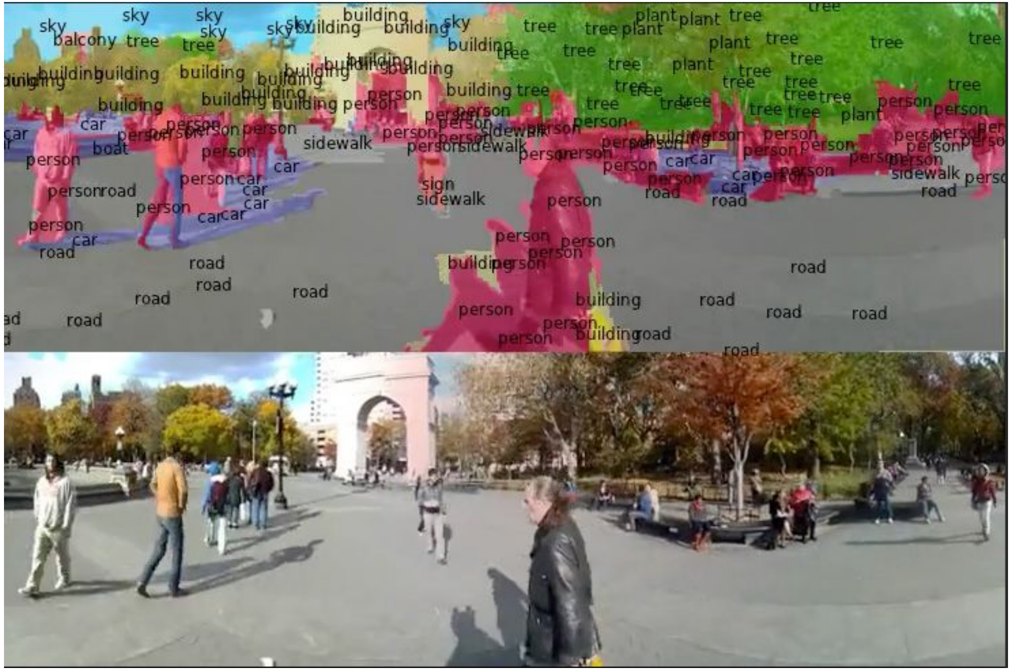
[Krizhevsky 2012]

Detection



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

Segmentation



[Farabet et al., 2012]

Convolutional Neural Networks

Hubel & Wiesel,

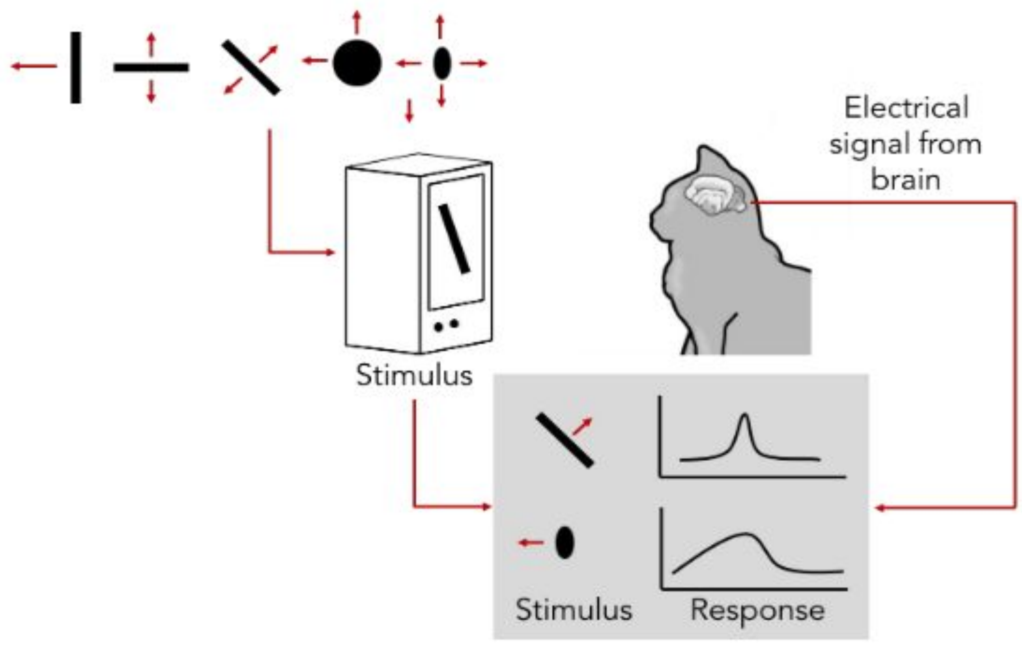
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

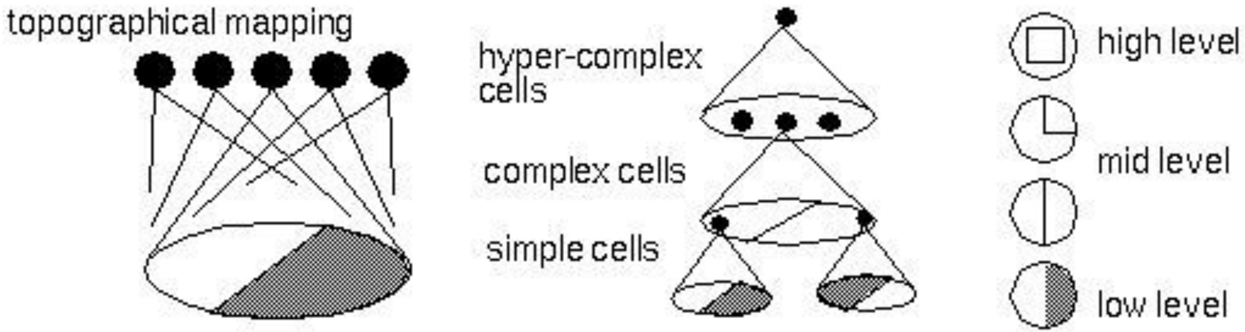
RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...

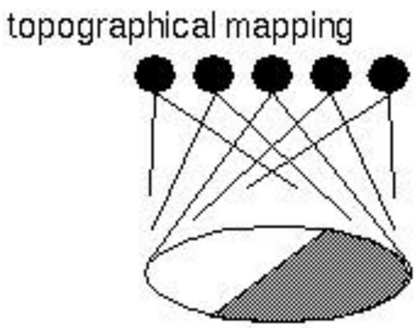


Hubel & Weisel

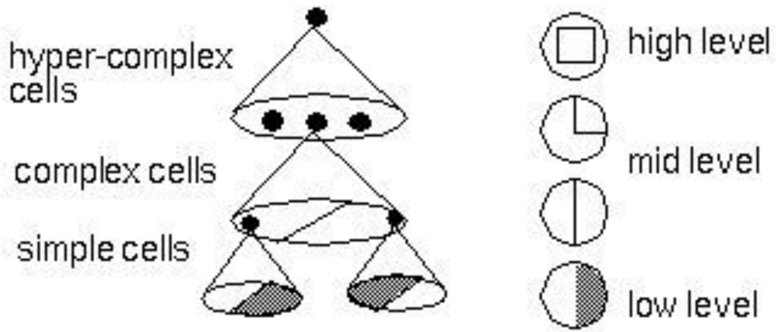
featural hierarchy

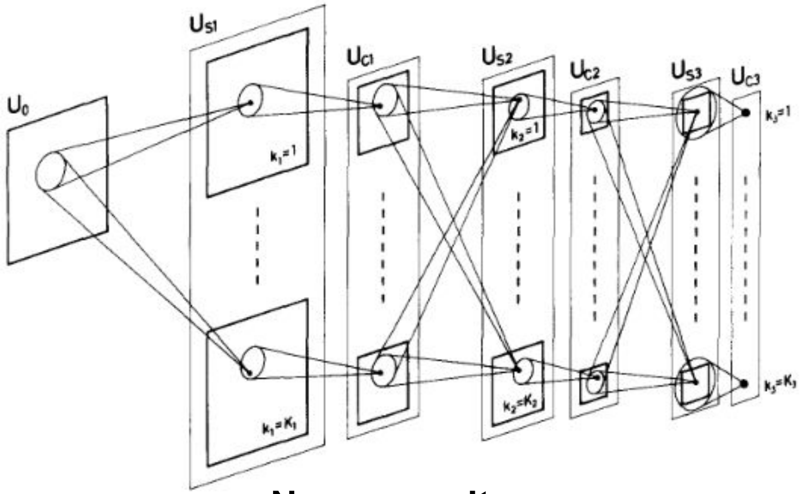


Hubel & Weisel



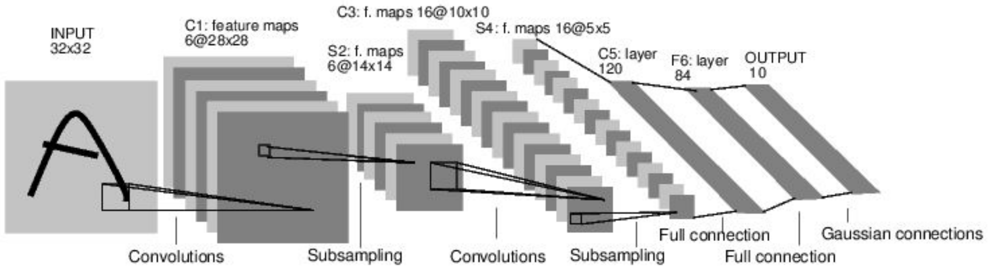
featural hierarchy





Neurocognitron

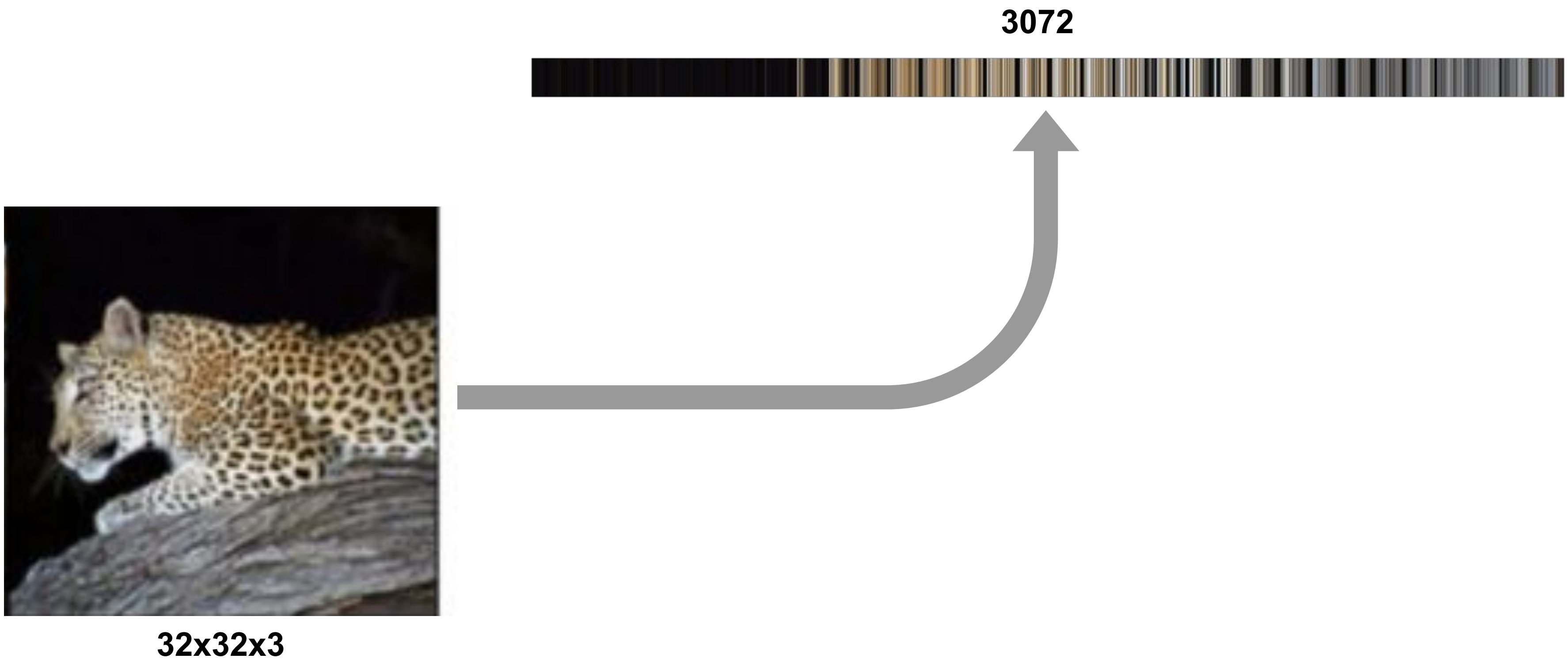
Lenet

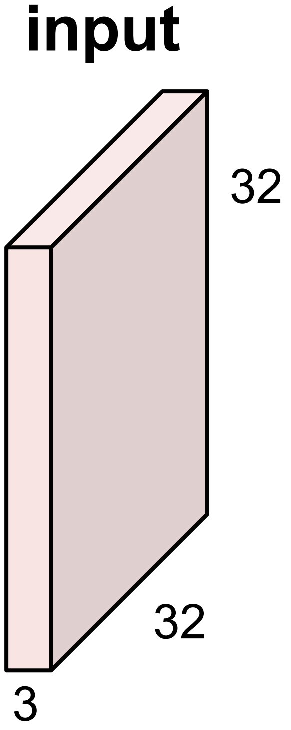


Fully Connected Layer

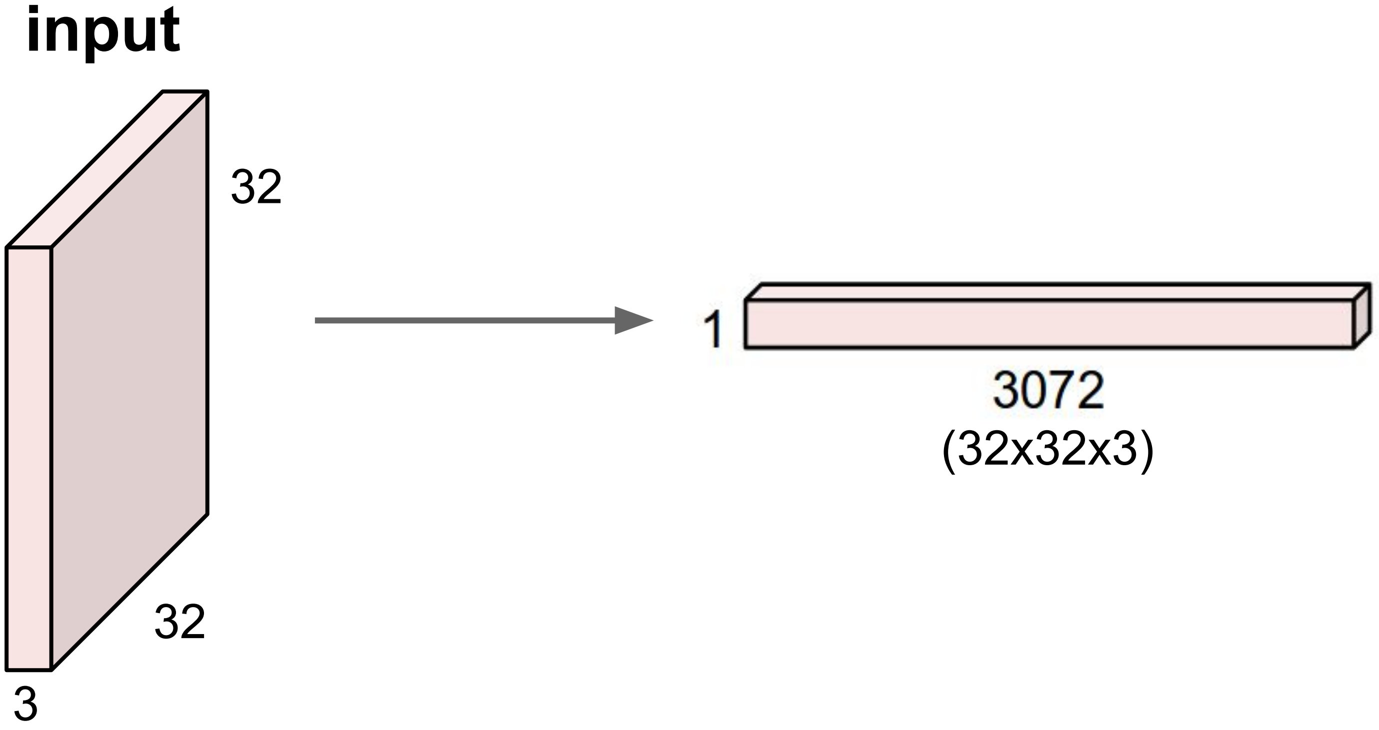
32x32x3

Fully Connected Layer

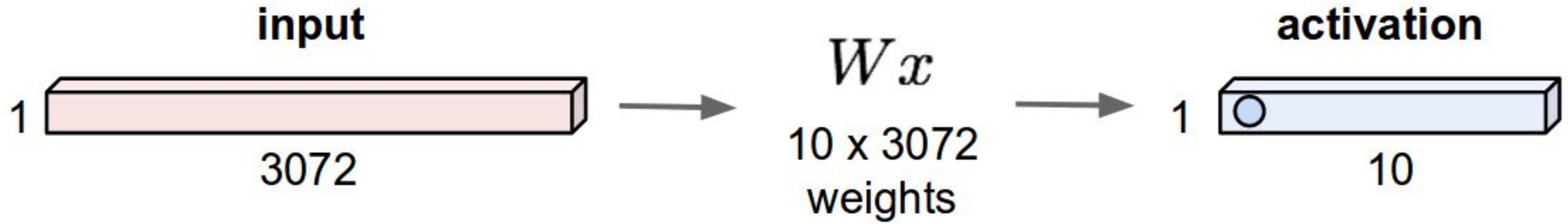



input
32
32
3

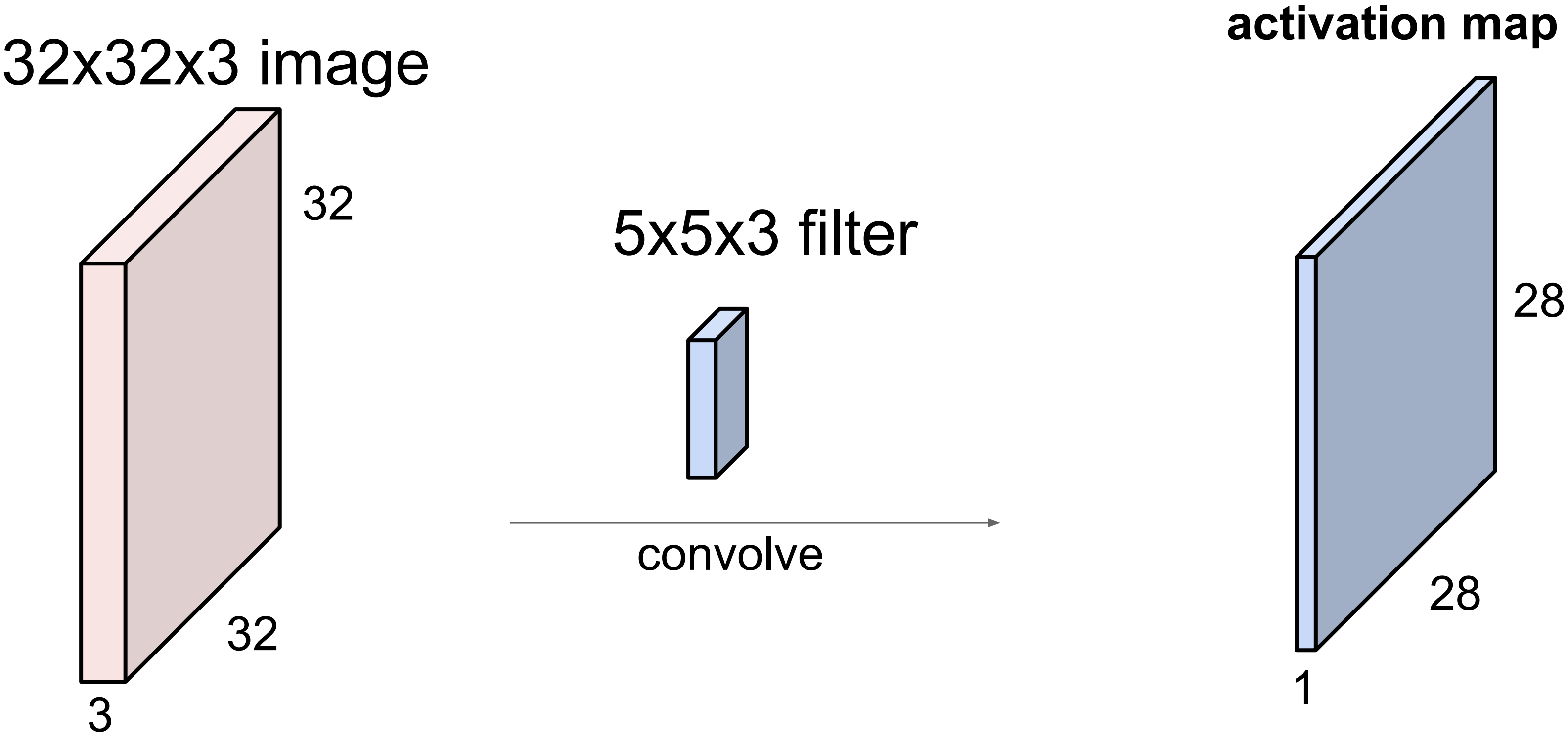
Fully Connected Layer



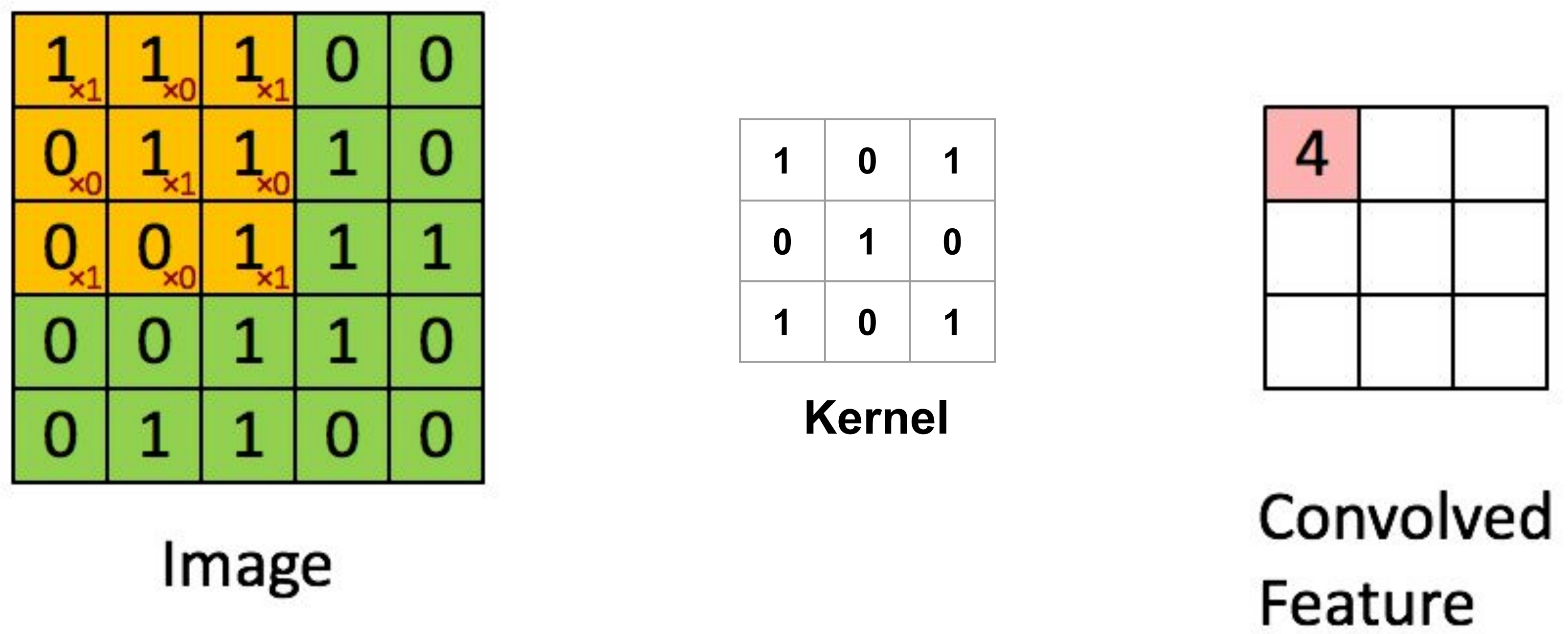
Fully Connected Layer



Convolution Layer

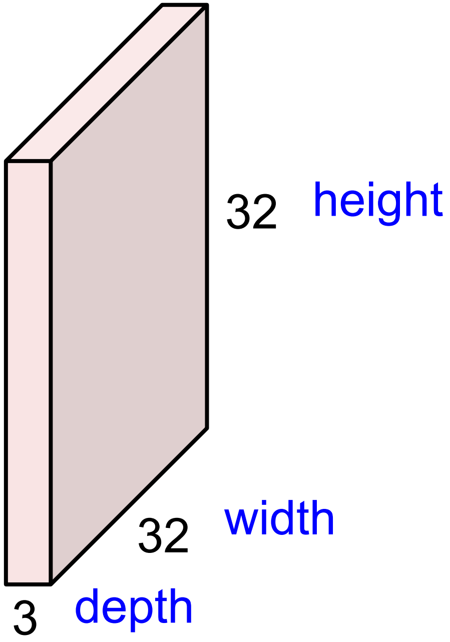


¿Qué es una convolución?



Convolution Layer

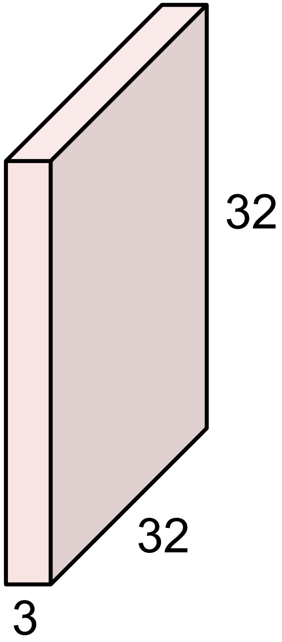
32x32x3 image



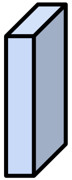
http://setosa.io/ev/image-kernels/

Convolution Layer

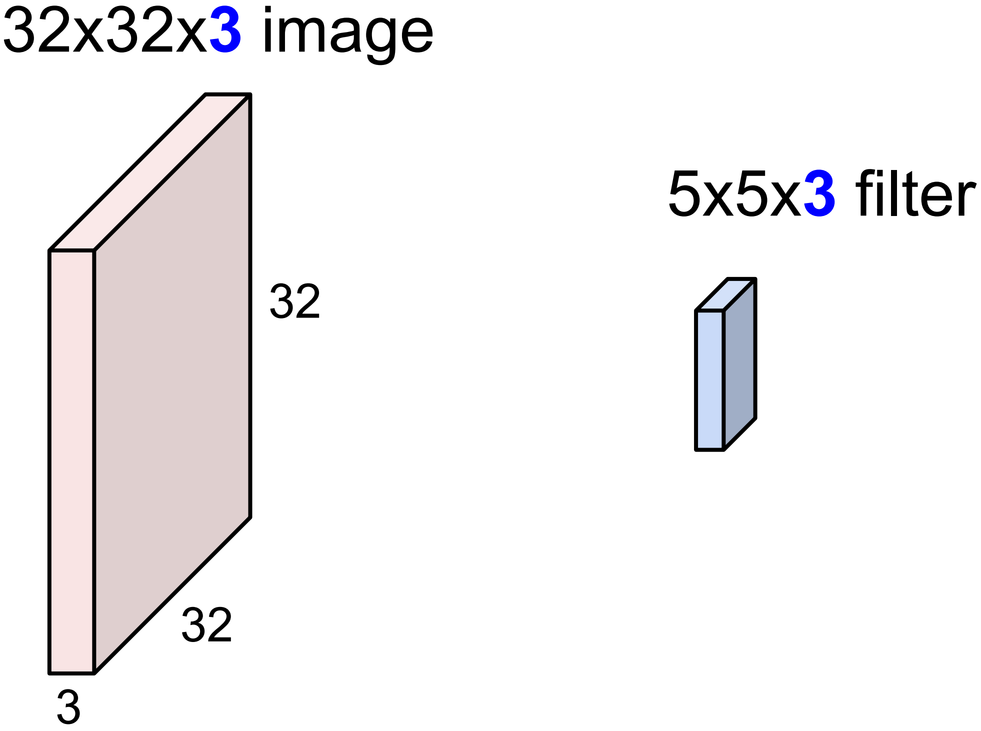
32x32x3 image



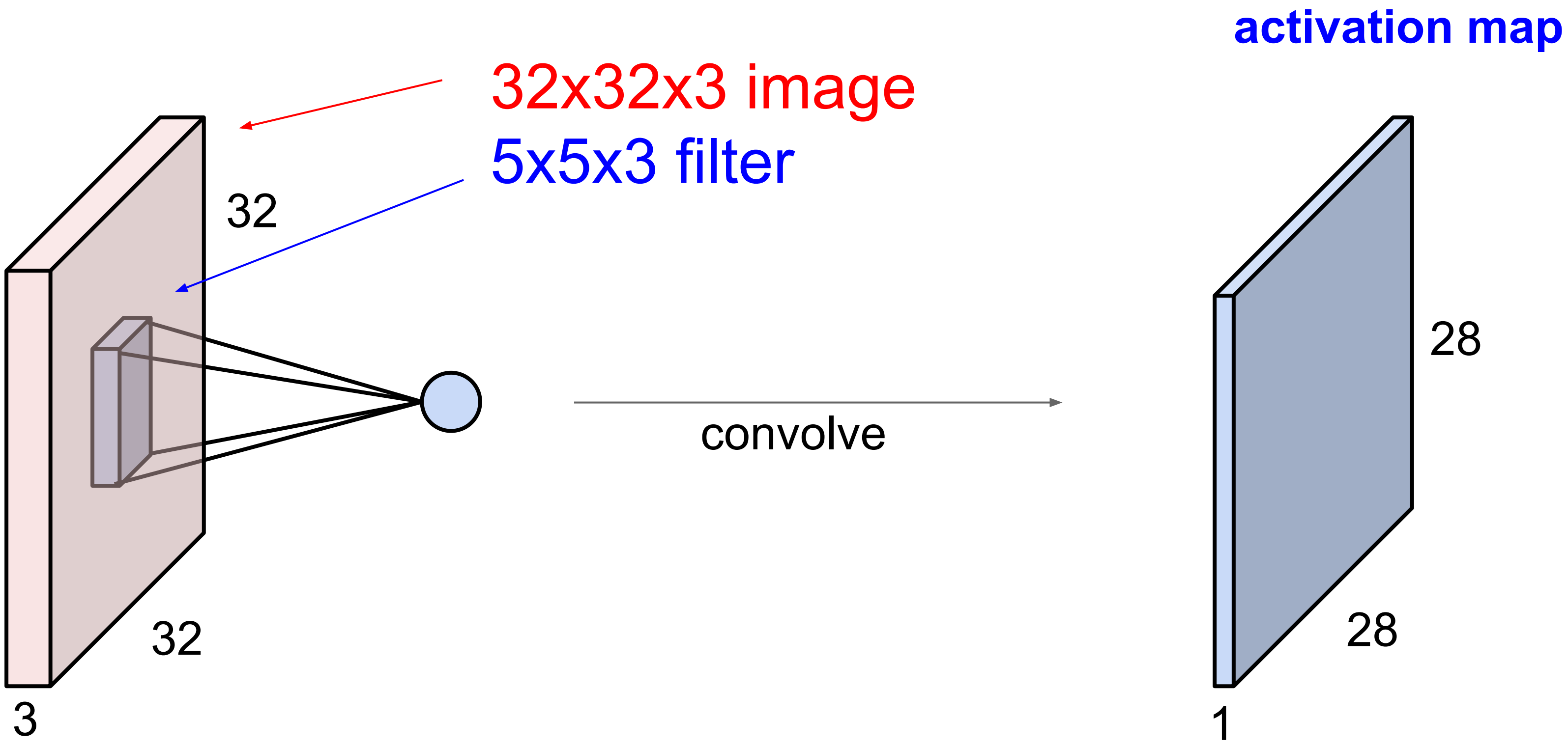
5x5x3 filter



Convolution Layer

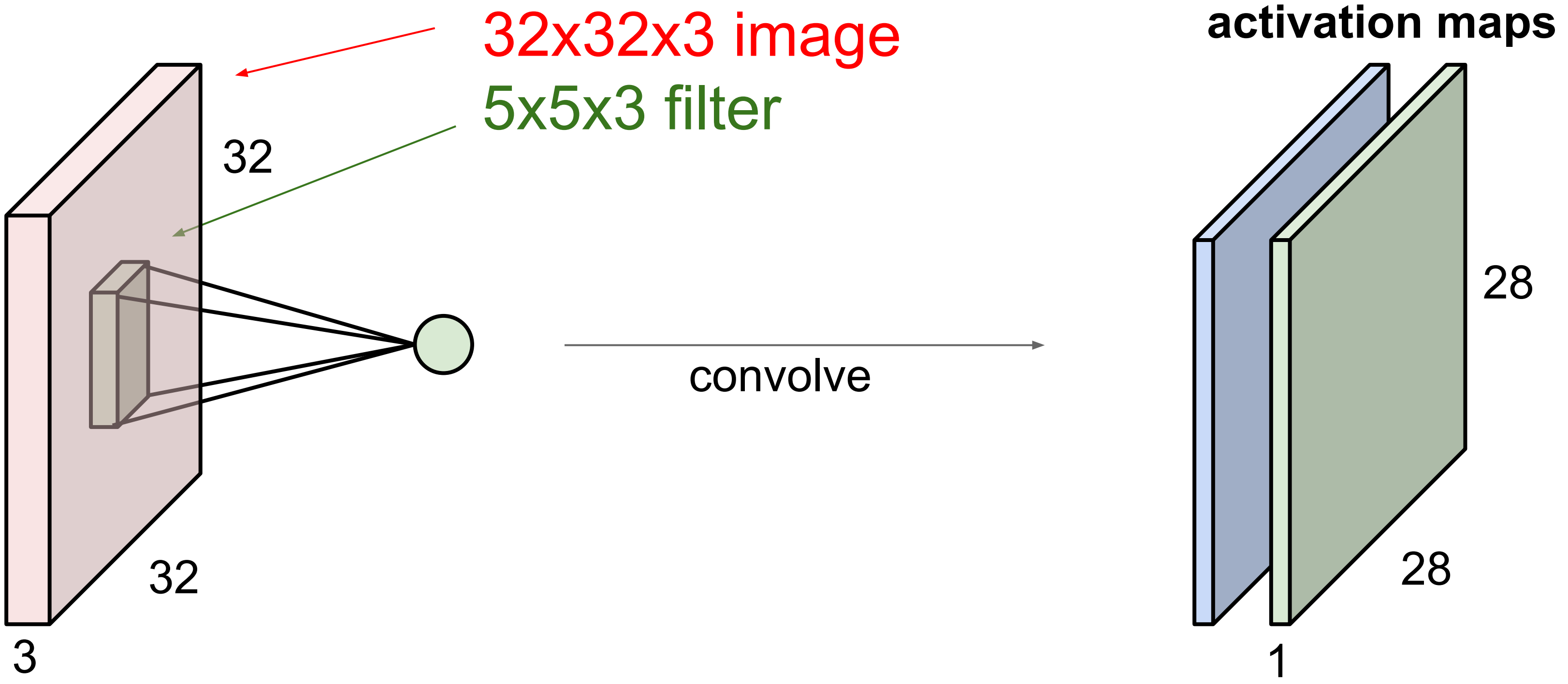


Convolution Layer

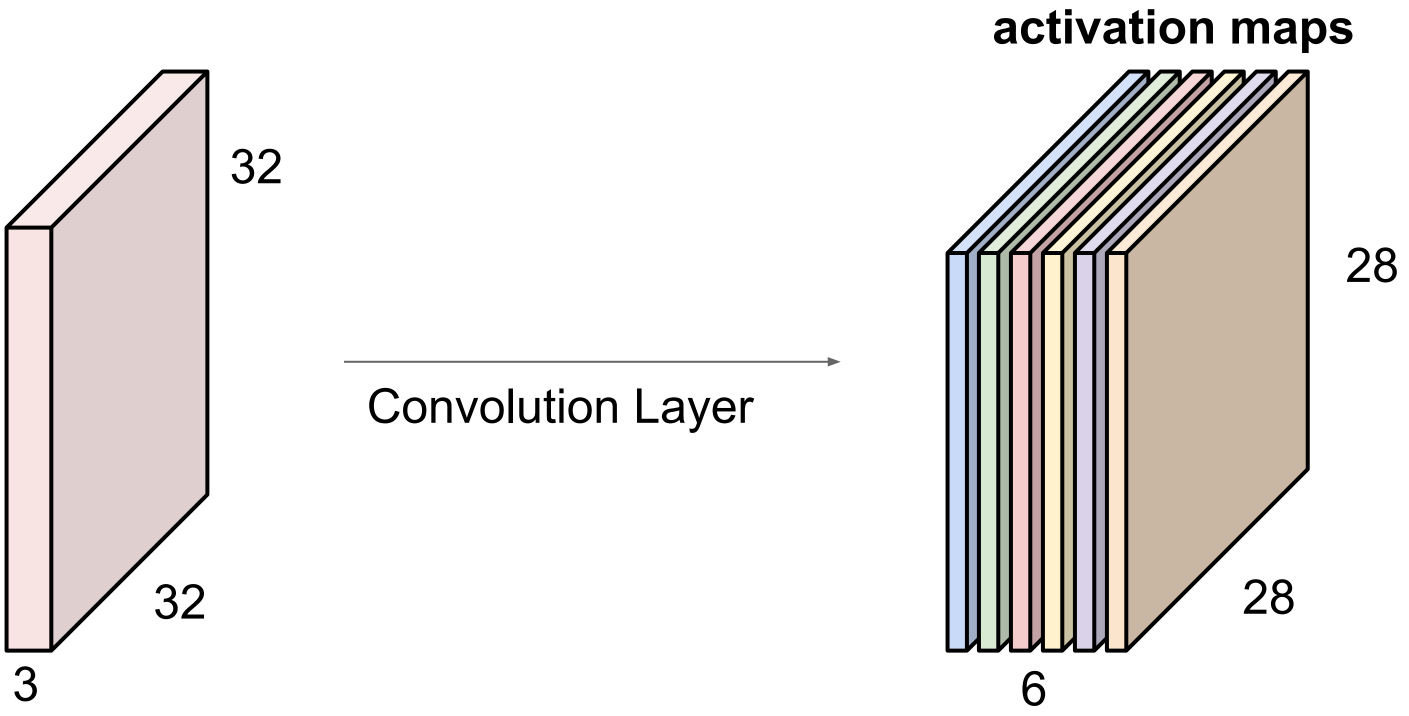


Convolution Layer

Un segundo filtro

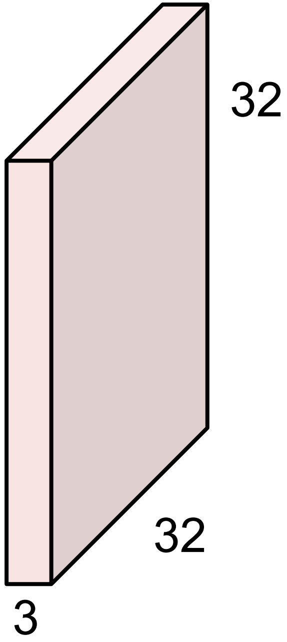


Convolution Layer

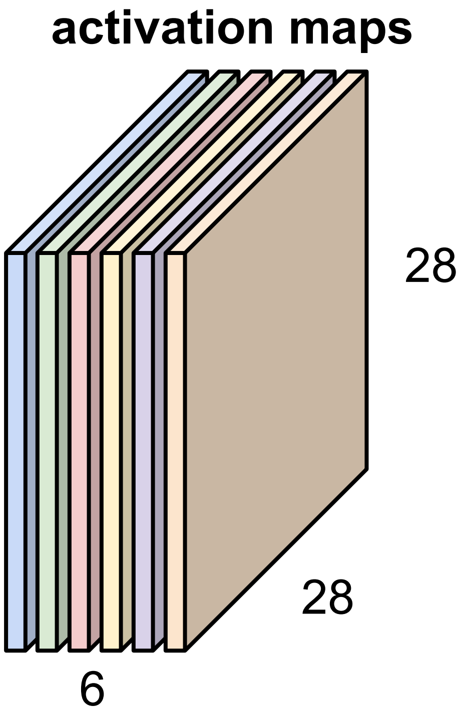


Si tenemos 6 filtros, el resultado tendría la forma: 28x28x6

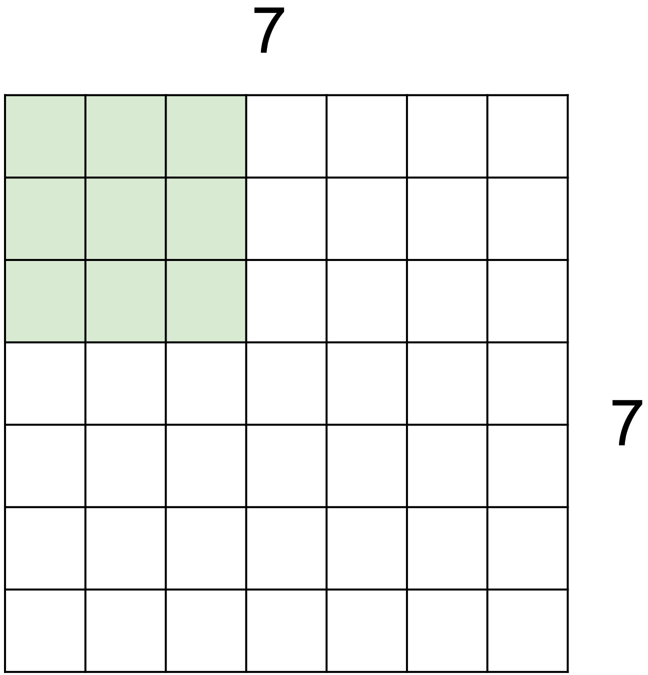
Convolution Layer



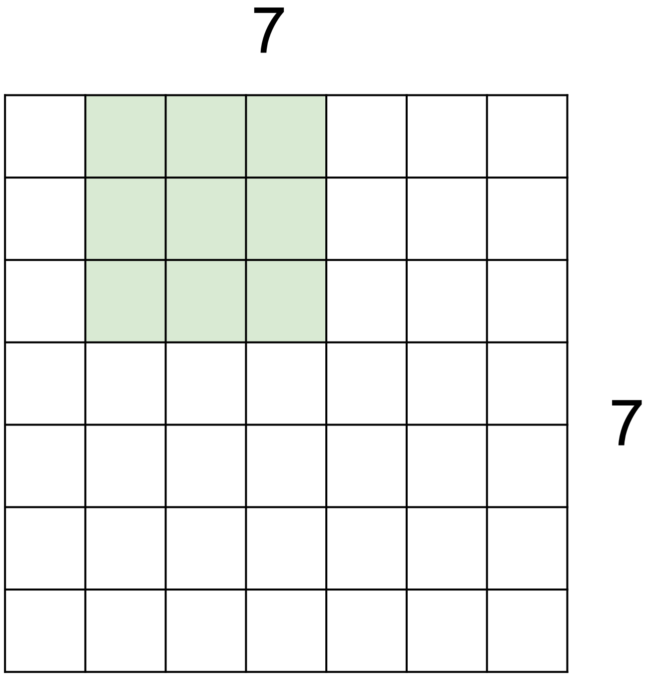
Convolution Layer



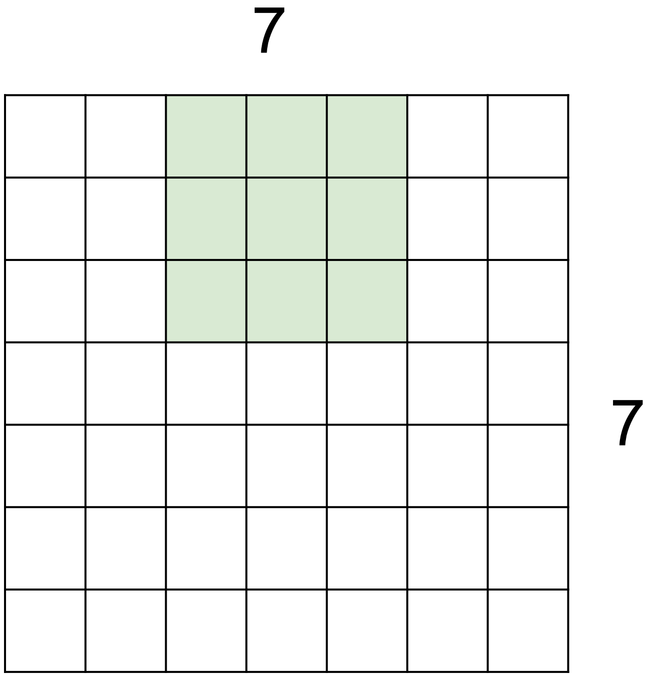
- Kernel size = 5
- # kernels = 6
- padding = 0



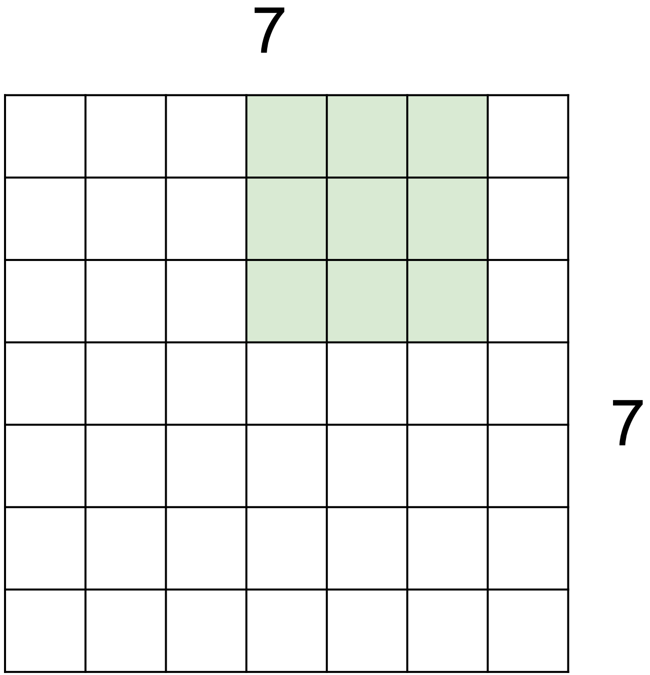
7x7 input
3x3 filter



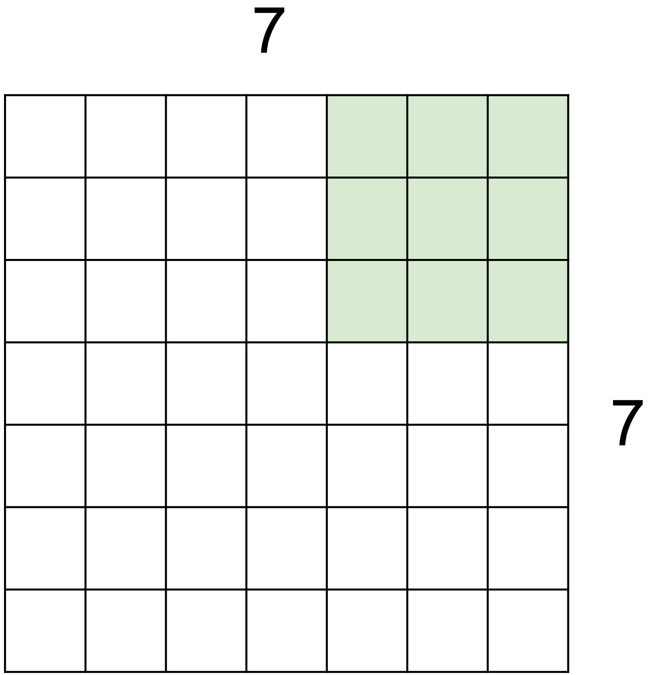
7x7 input
3x3 filter



7x7 input
3x3 filter



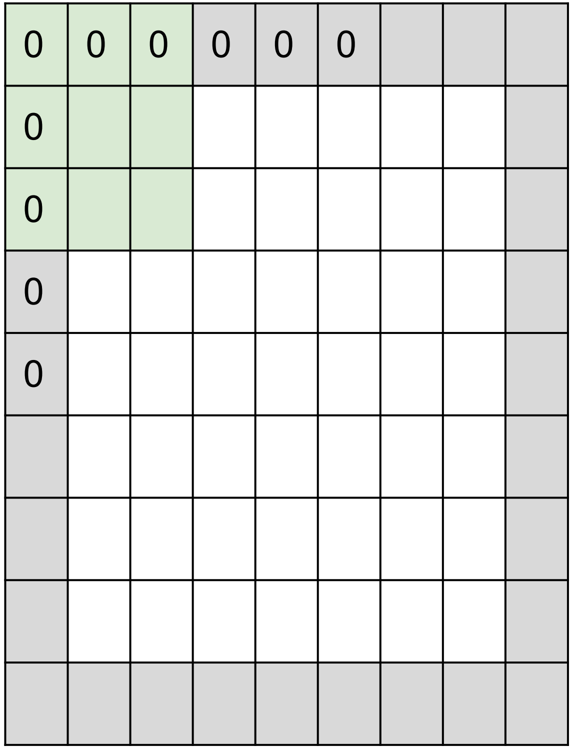
7x7 input
3x3 filter



7x7 input
3x3 filter

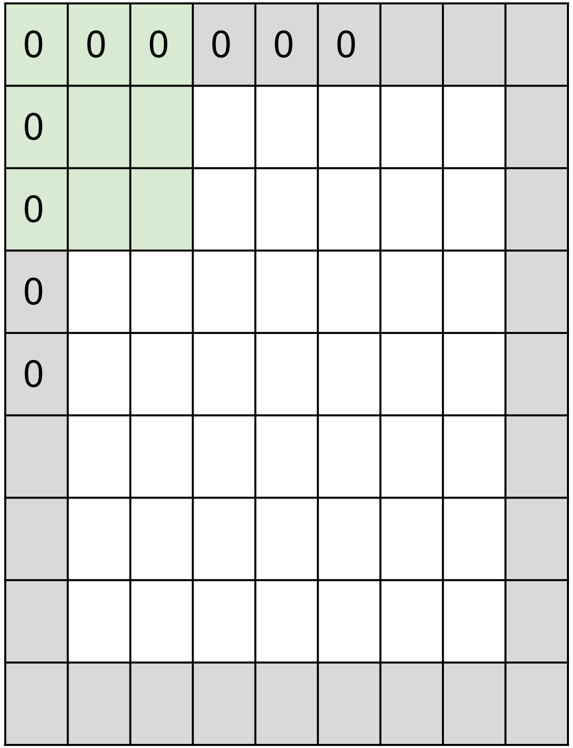
=> 5x5 output

Padding



input 7x7
3x3 filter
padding 1

Padding

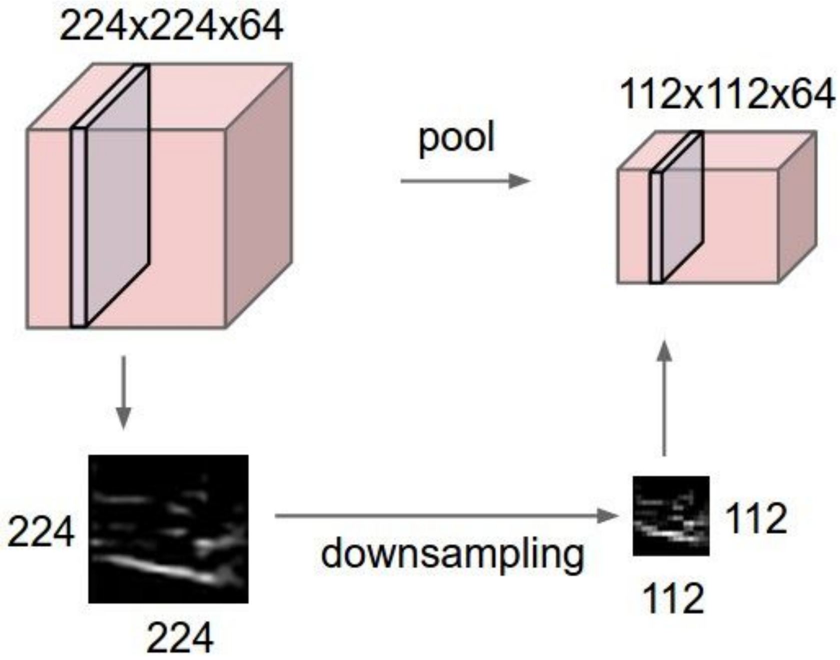


input 7x7
3x3 filter
padding 1

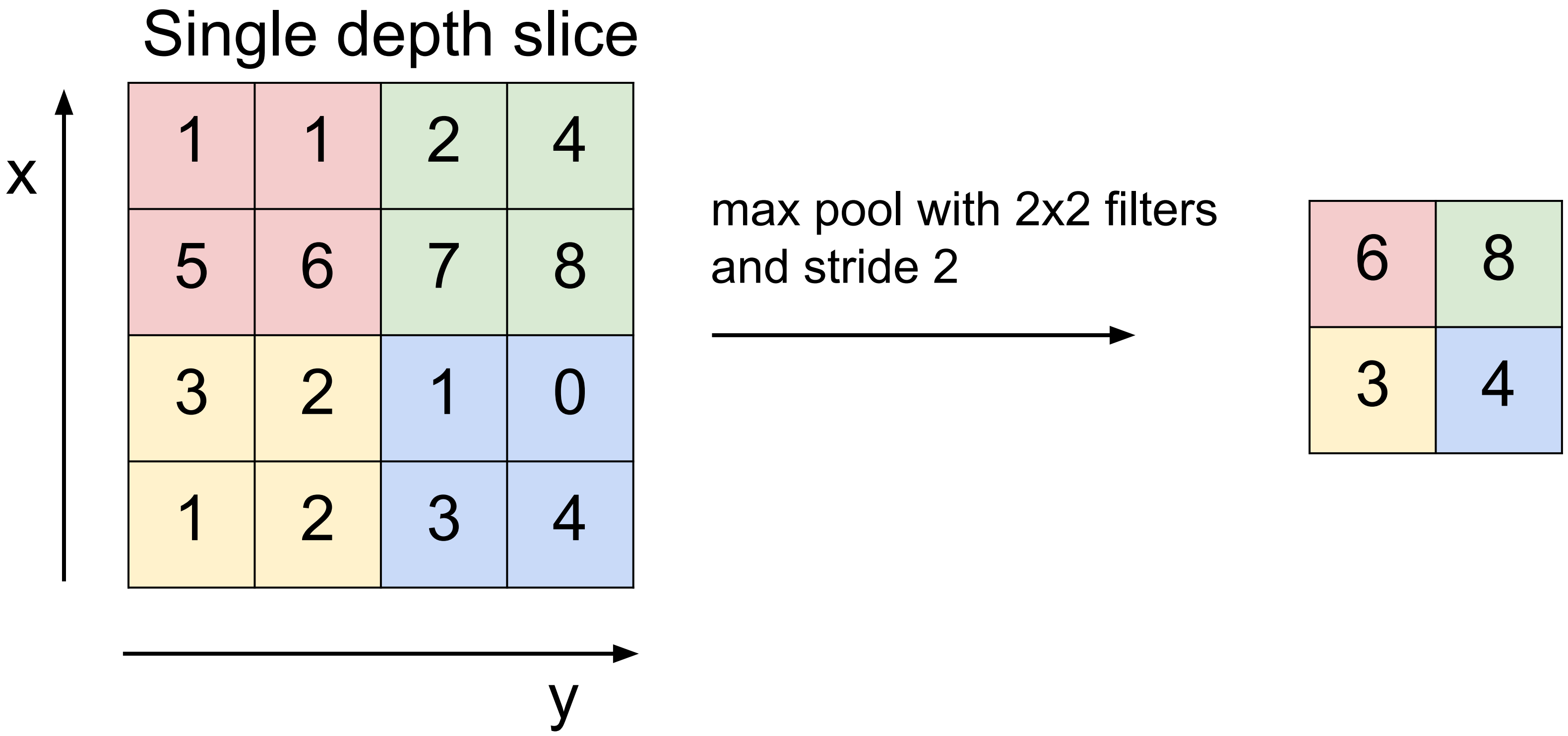
7x7 output!

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

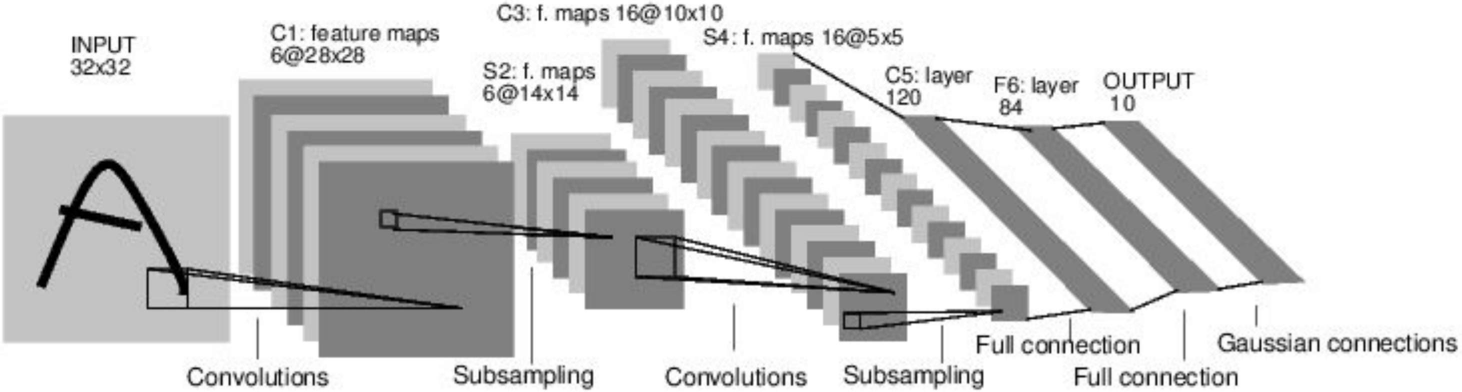
Pooling layer



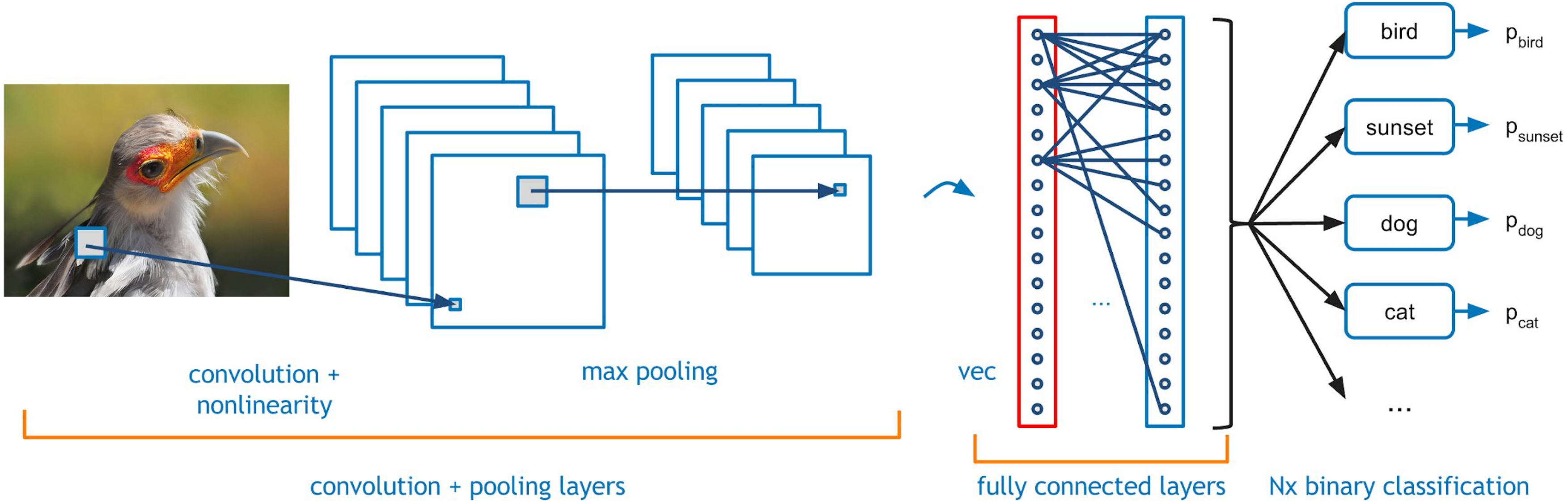
MAX POOLING


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

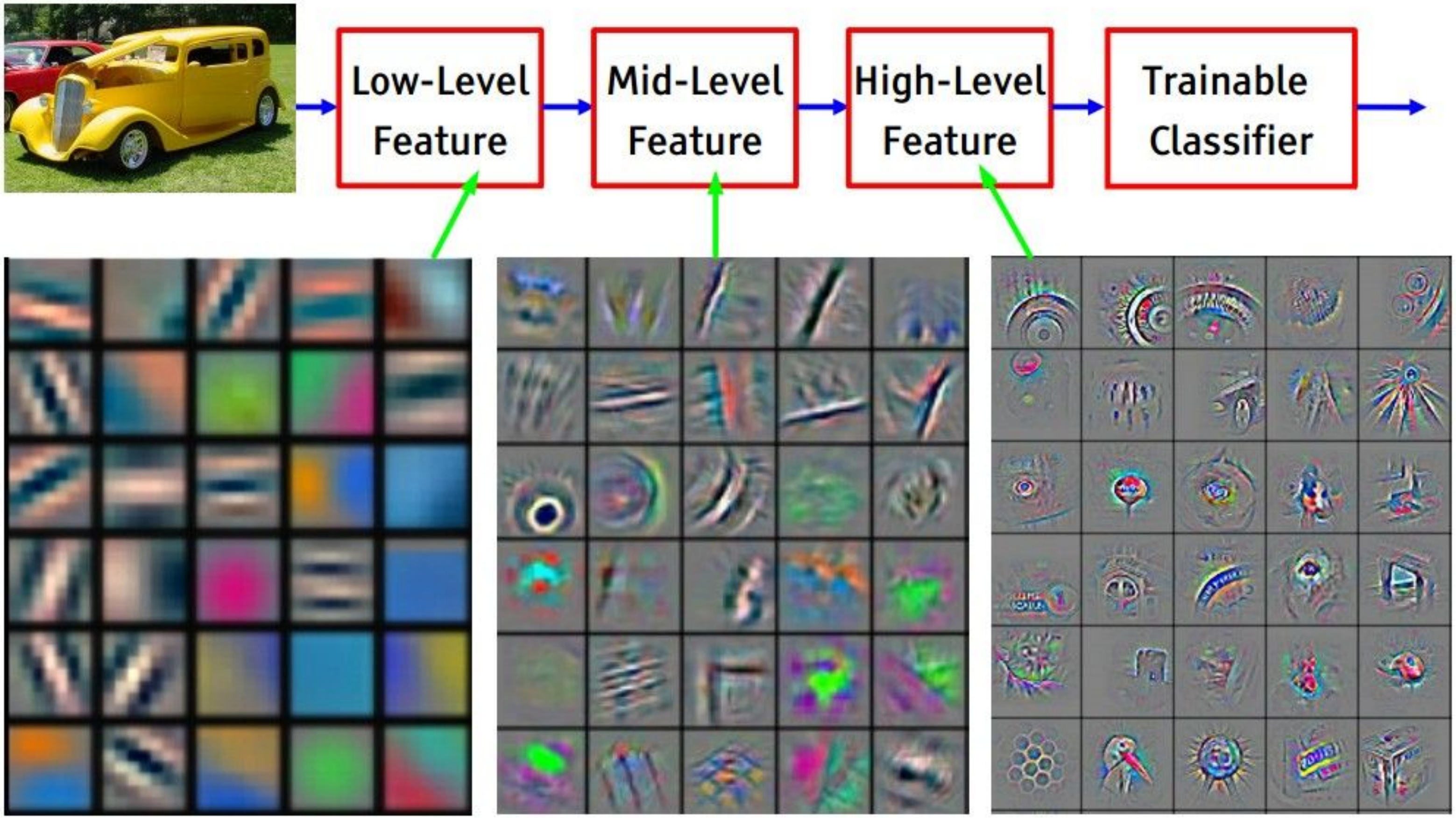
Convolutional Neural Networks



[LeNet-5, LeCun 1980]



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



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

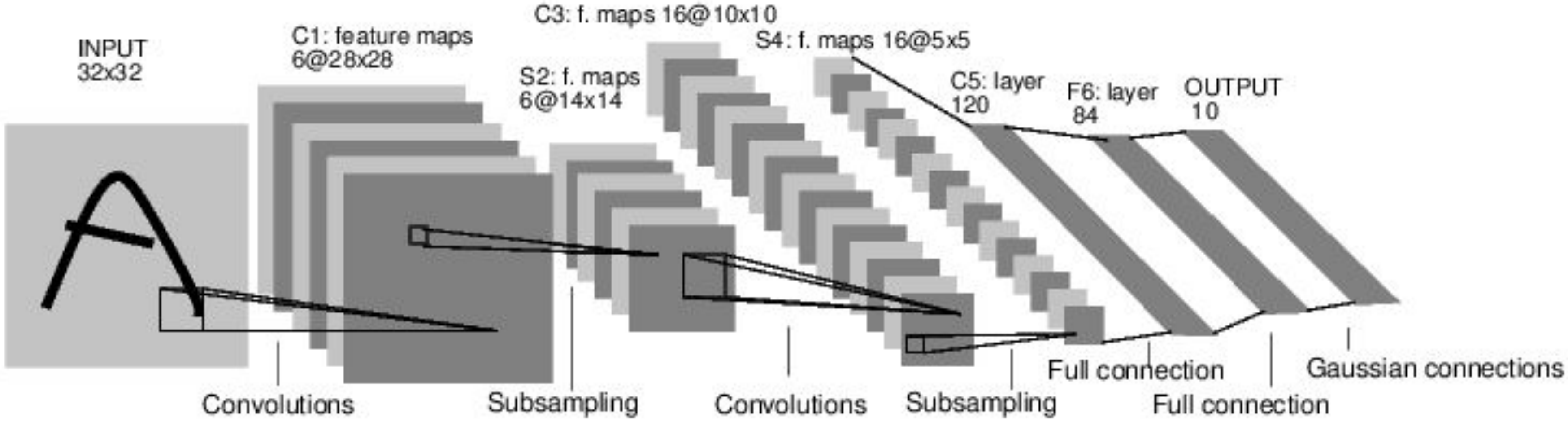
Keras code

```
from 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

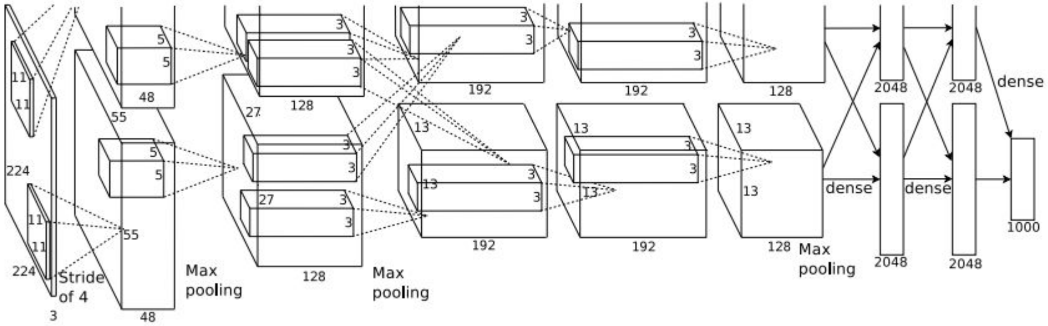
[LeCun et al., 1998]



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

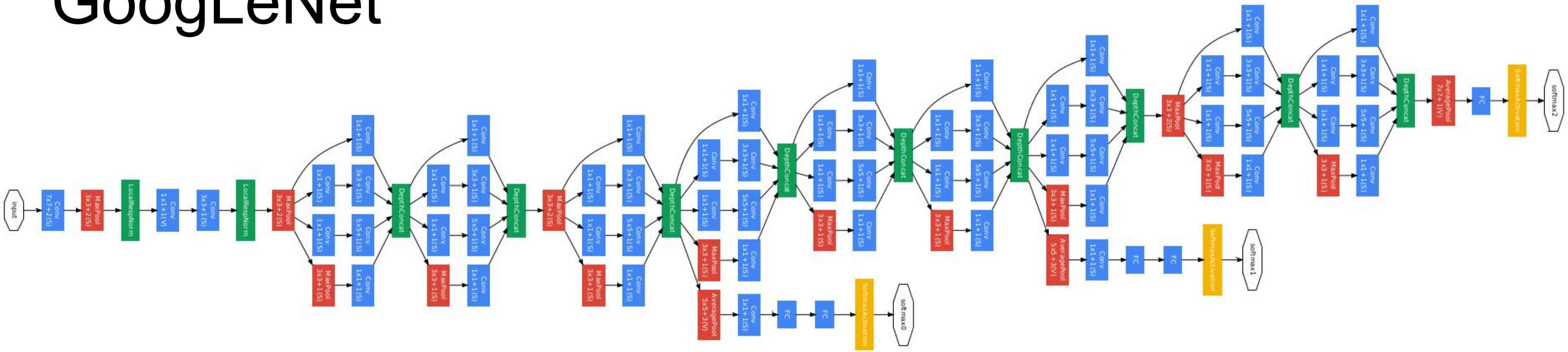
7.3% top 5 error

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					

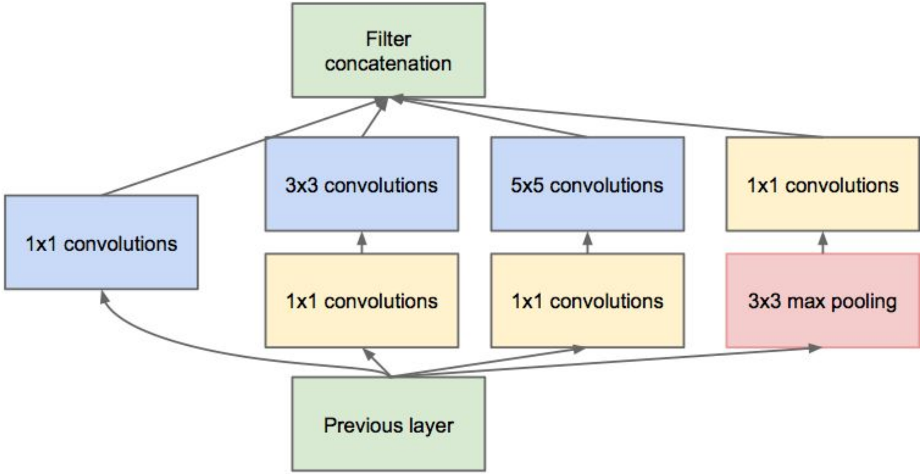
Table 2: **Number of parameters** (in millions).

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

GoogLeNet



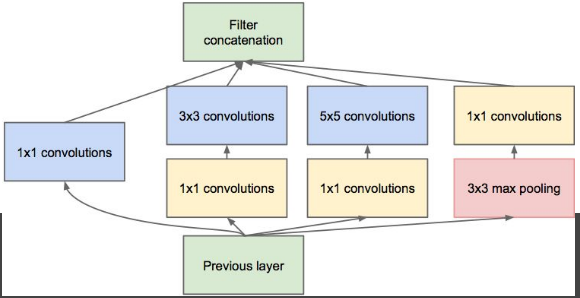
[Szegedy et al., 2014]



Inception module

ILSVRC 2014 winner (6.7% top 5 error)

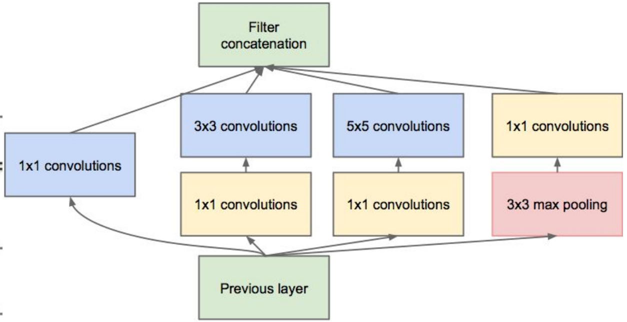
Inception module (Keras code)




```
from 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]

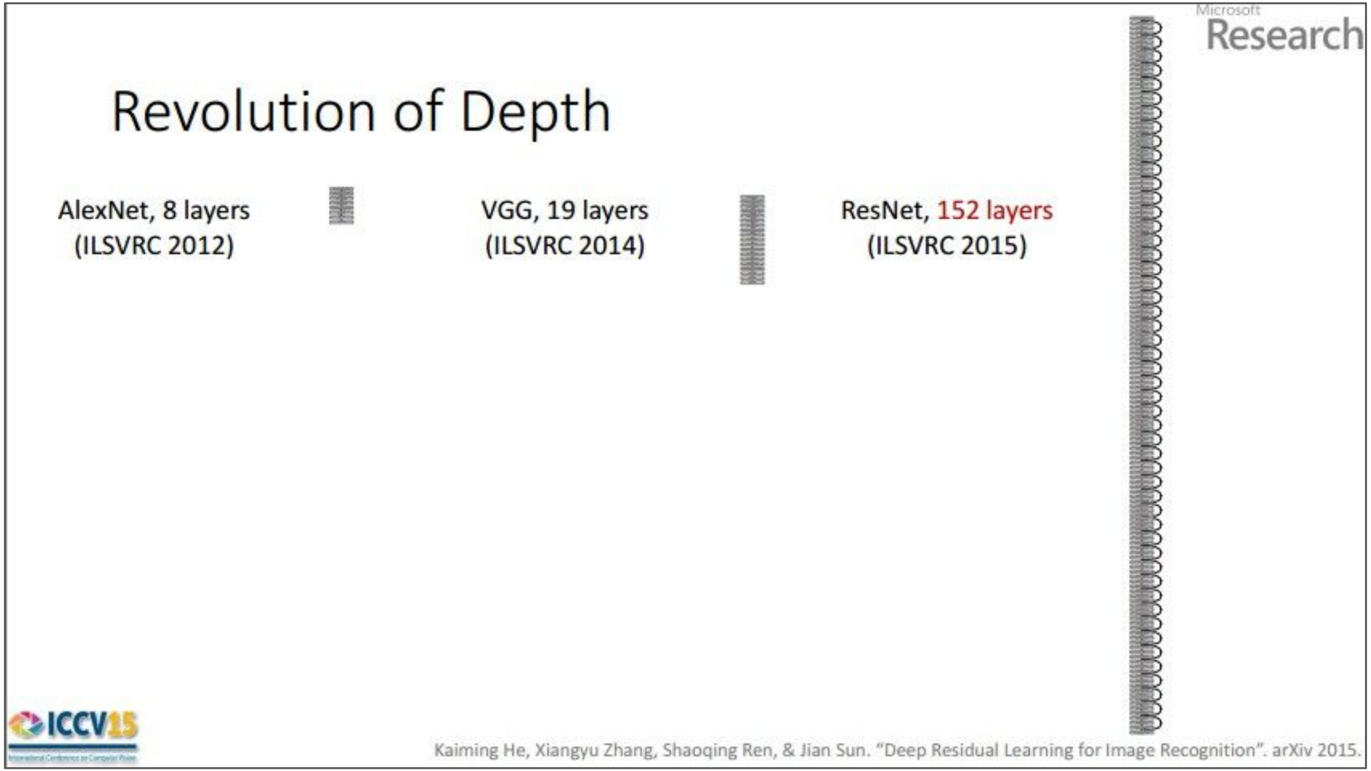


GoogLeNet

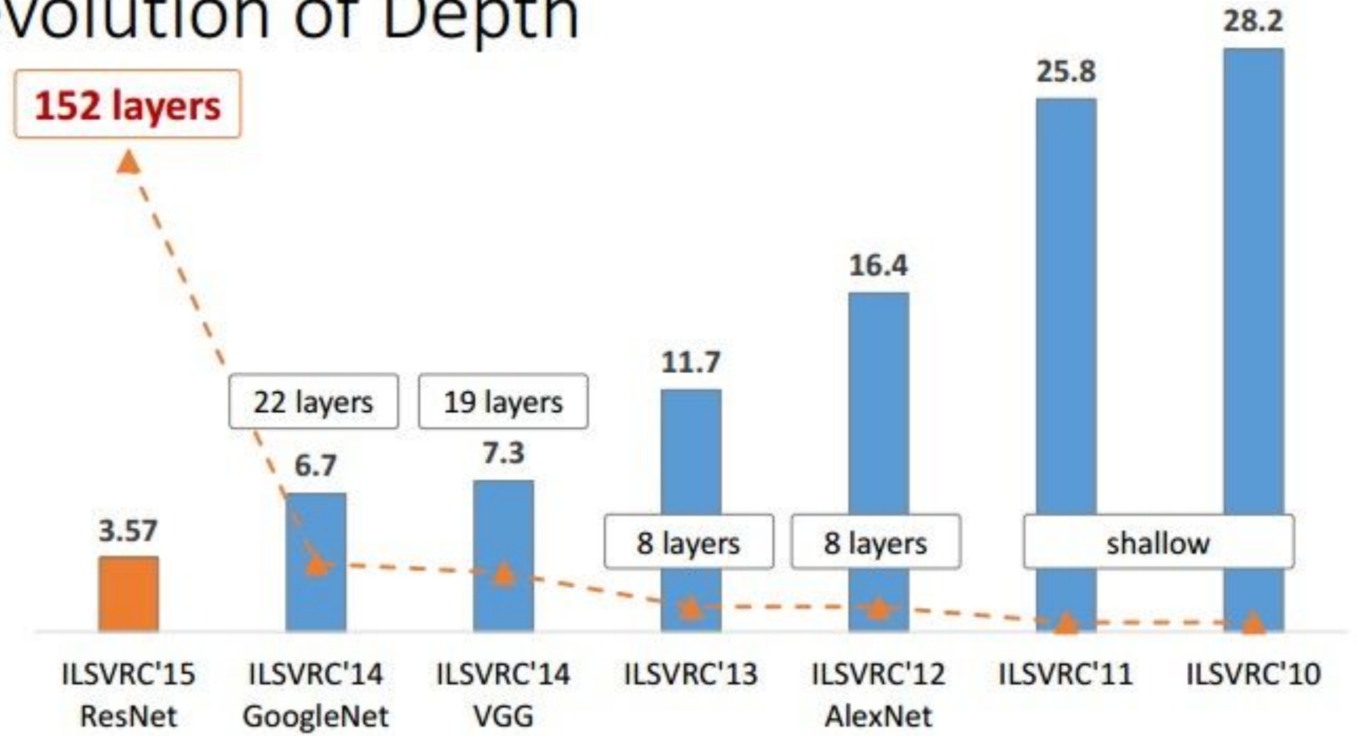
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	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)



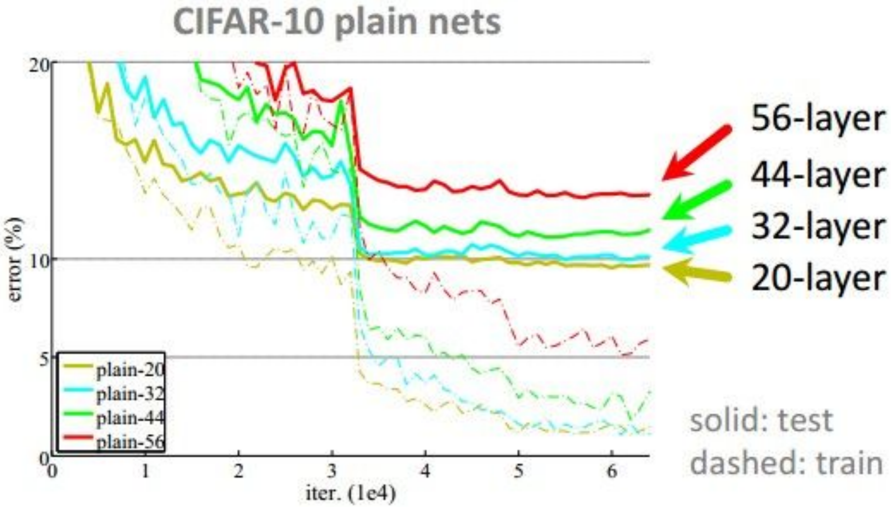
Revolution of Depth

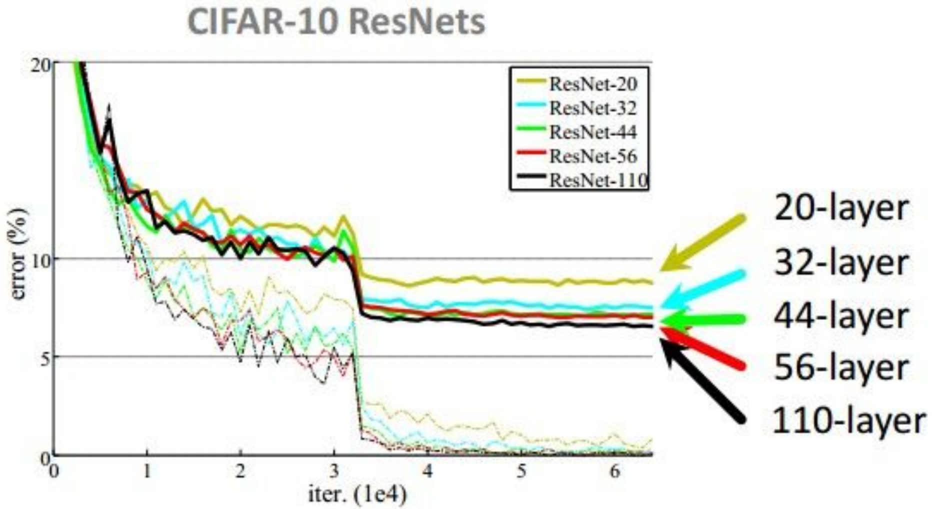


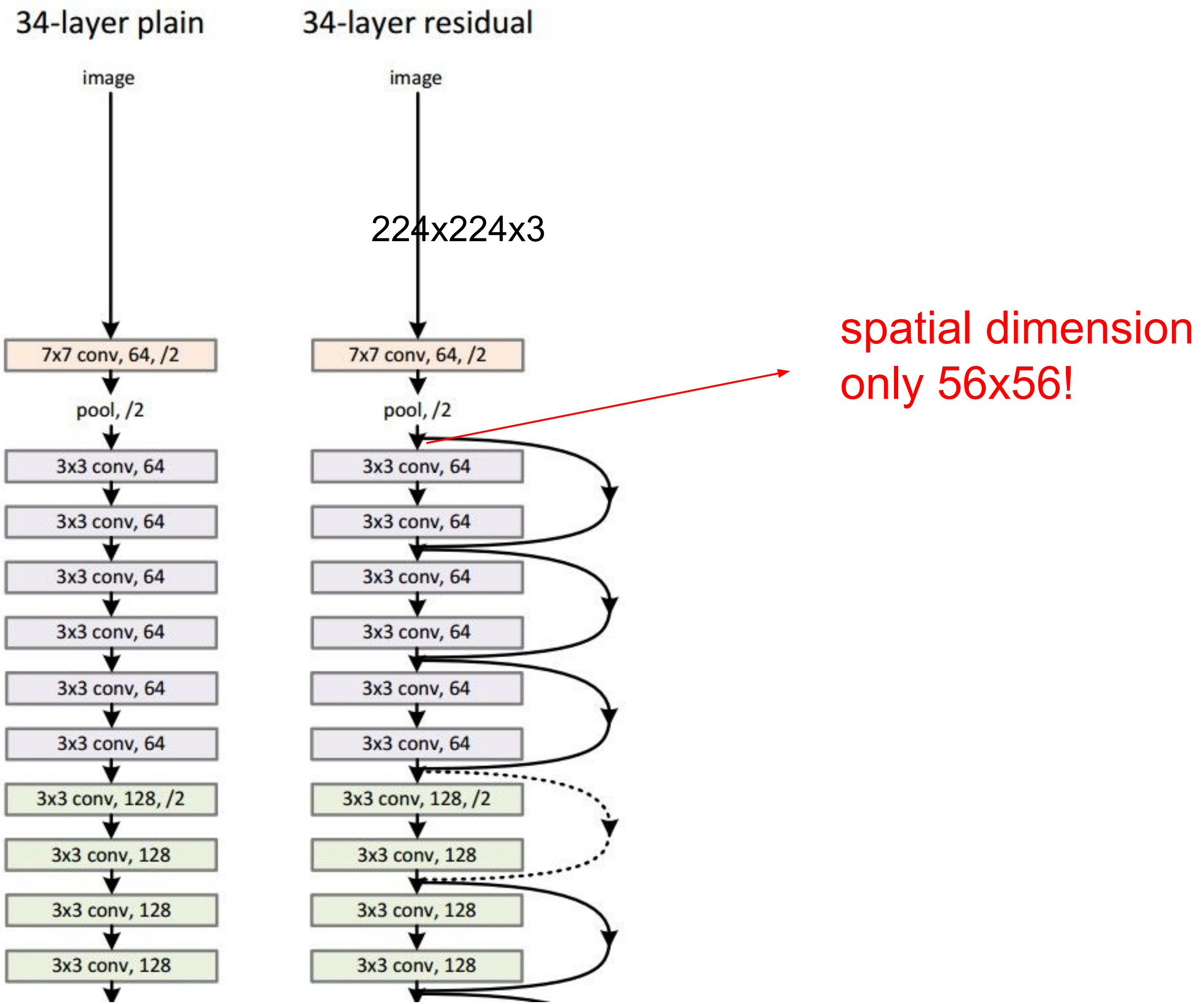
ImageNet Classification top-5 error (%)

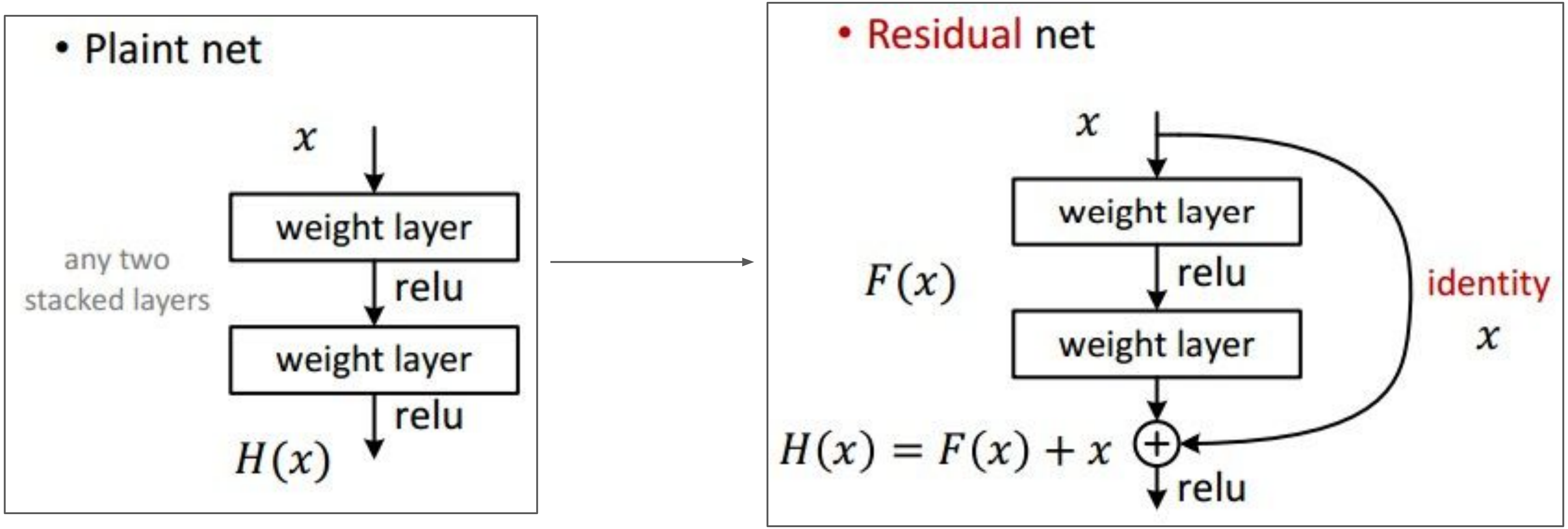
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

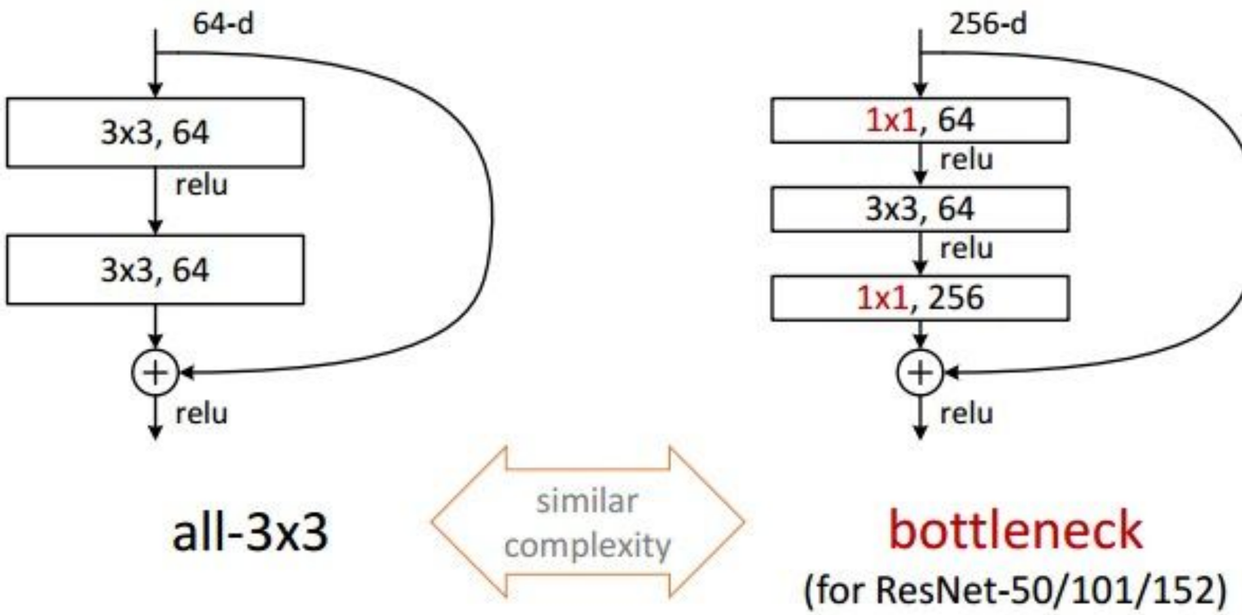
CIFAR-10 experiments





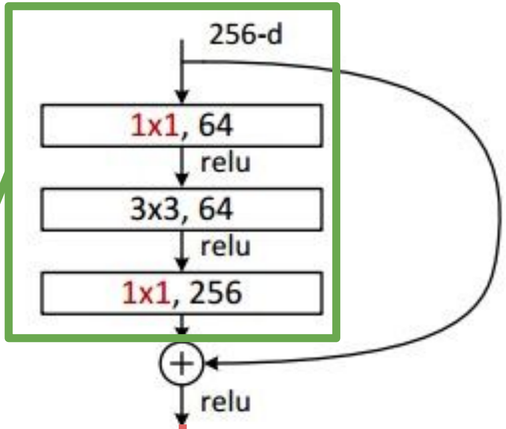

34-layer plain
34-layer residual
image
image
224x224x3
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
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
x
weight layer
$H(x) = F(x) + x$
relu



64-d
3x3, 64
relu
3x3, 64
relu
all-3x3
similar
complexity
256-d
1x1, 64
relu
3x3, 64
relu
1x1, 256
relu
bottleneck
(for ResNet-50/101/152)

```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

Code time

https://colab.research.google.com/