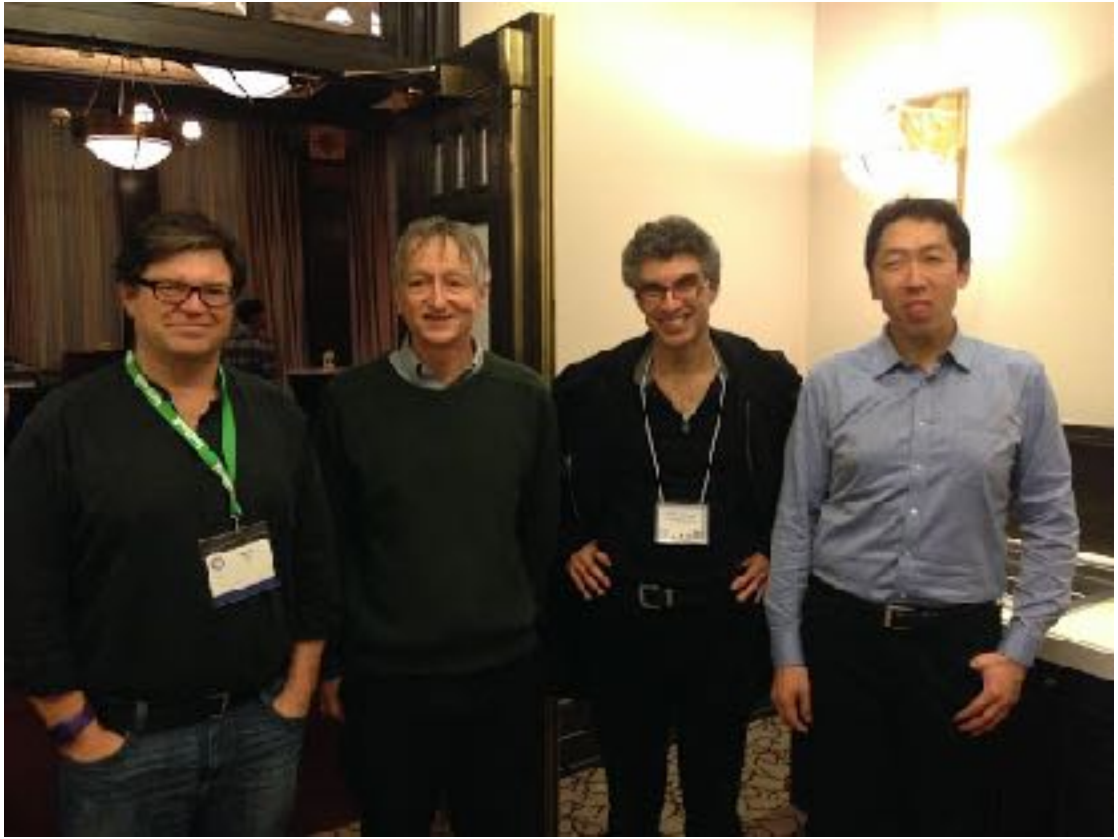
Intro to CNN

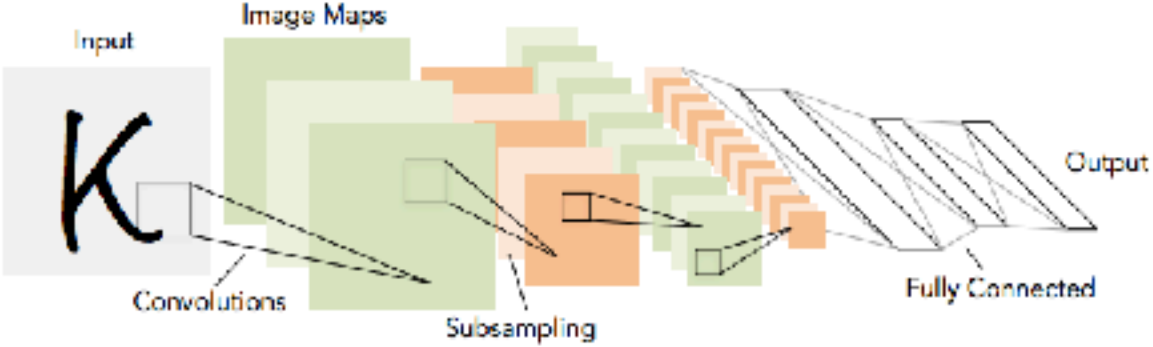
D:plus



Left to right: Yann LeCun, Geoff Hinton, Yoshua Bengio, Andrew Ng at NIPS 2014 (from Andrew Ng's Facebook page).

1998

LeCun et al.



# of transistors

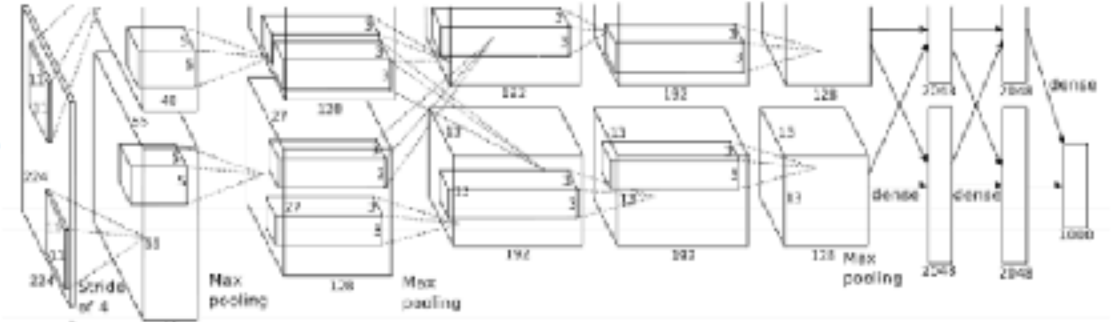
10⁶

# of pixels used in training

10⁷ NIST

2012

Krizhevsky et al.



# of transistors

GPUs

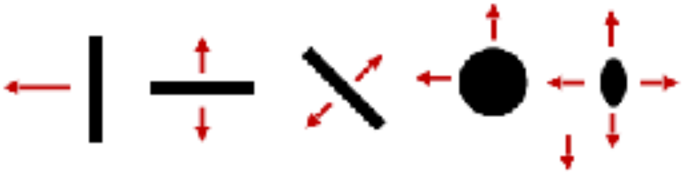
10⁹

# of pixels used in training

10¹⁴ IMAGENET

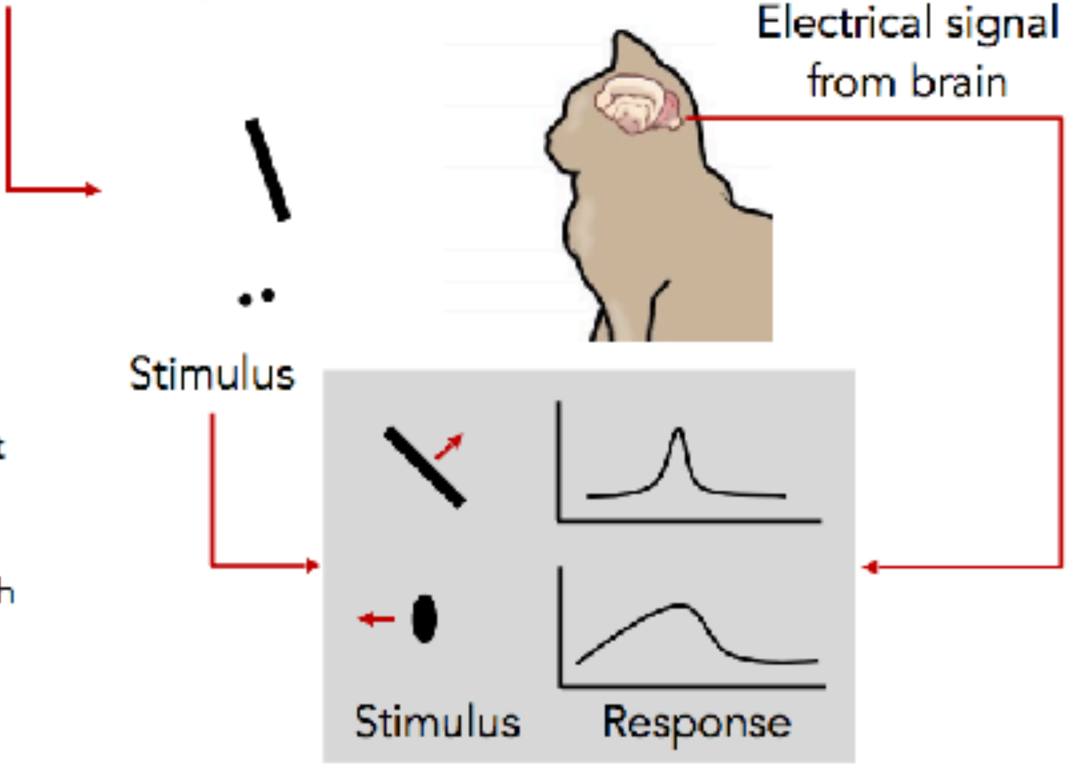
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Hubel & Wiesel, 1959

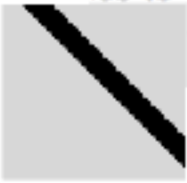


Simple cells:
Response to light orientation

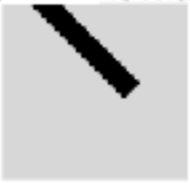
Complex cells:
Response to light orientation and movement



Hypercomplex cells:
response to movement with
nd



No response



Response
(end point)

Cat image by CNX OpenStax is licensed under CC BY 4.0; changes made

1.조각을 본다

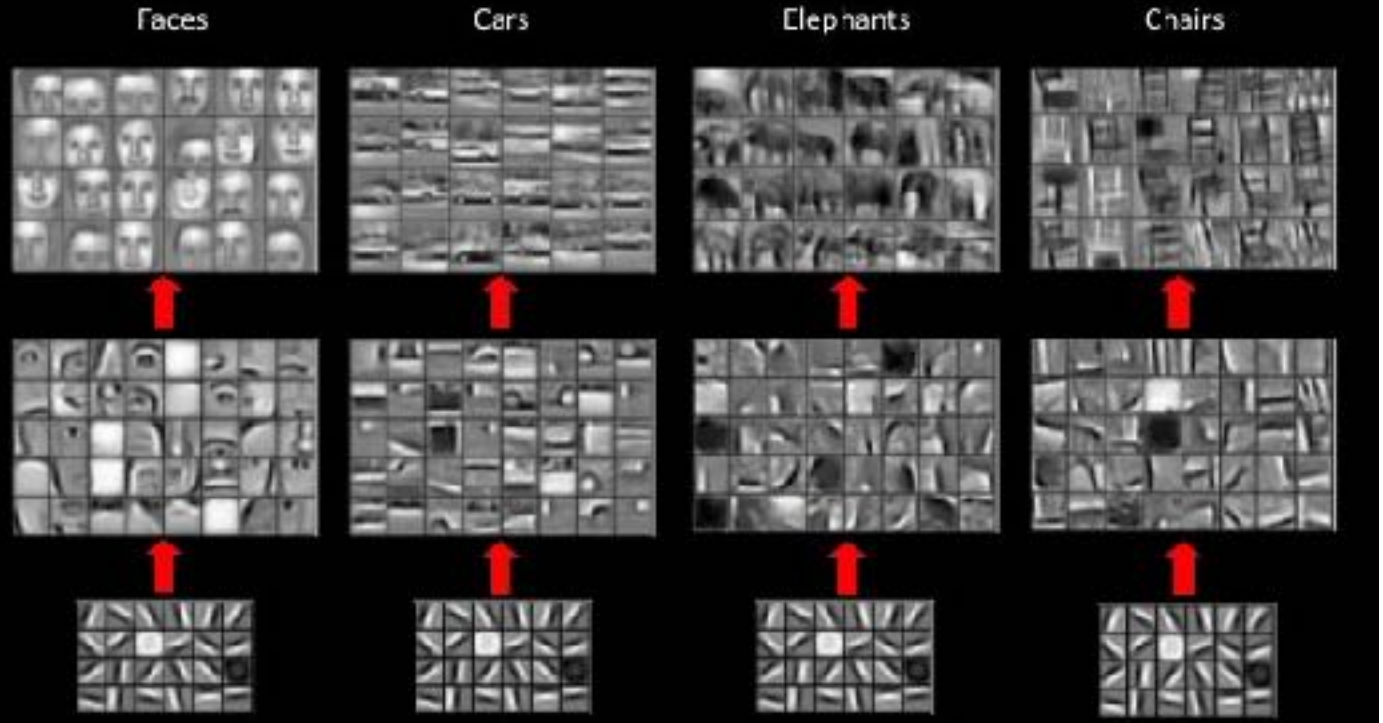
2.각 조각이 조합된 패턴을 본다

3.점점 더 복잡한 조합의 패턴을 본다.

4.반응하는 여러 패턴의 조합을 가지고 이미지를 인식한다.

Learning of object parts

Examples of learned object parts from object categories



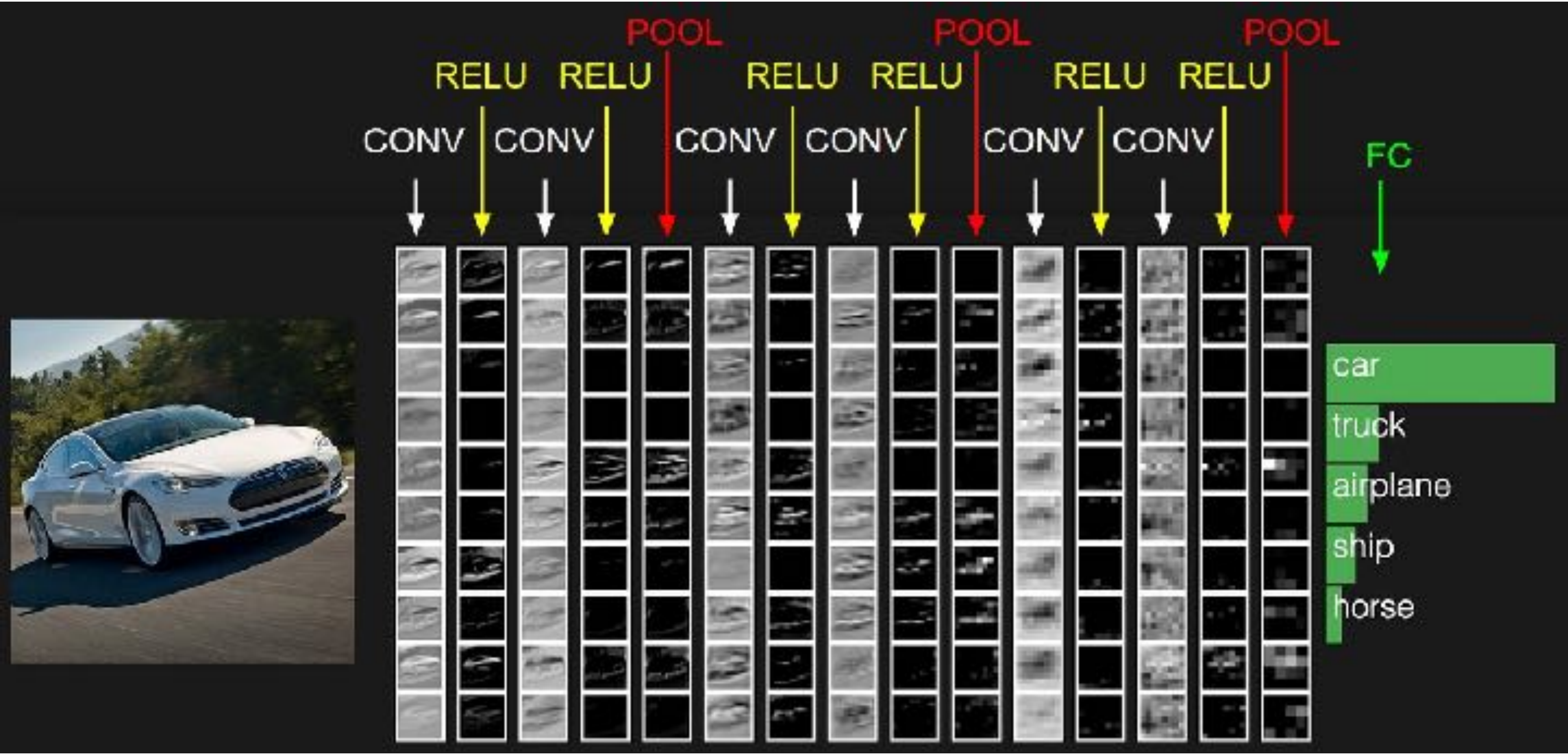
Faces

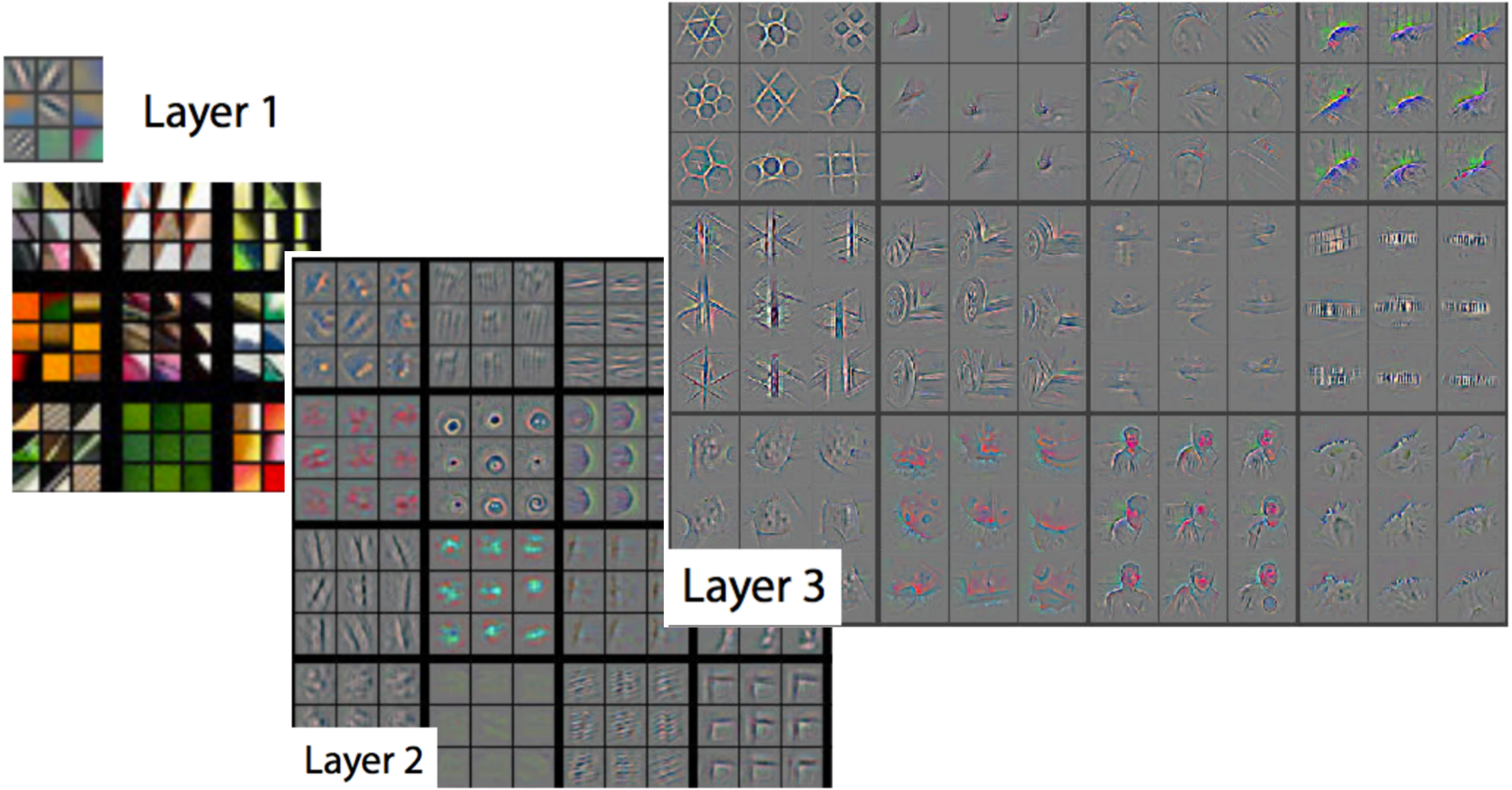
Cars

Elephants

Chairs

“필터를 CNN이 스스로 학습한다.”

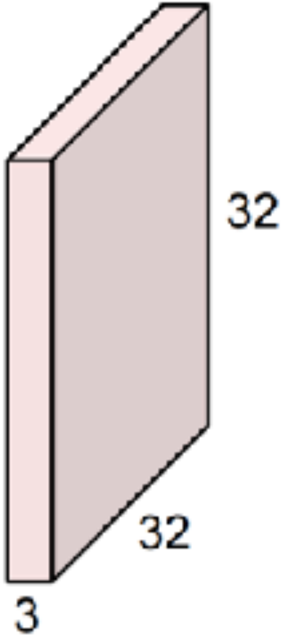

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse



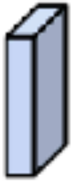
https://arxiv.org/pdf/1311.2901.pdf

Convolution Layer

32x32x3 image



5x5x3 filter

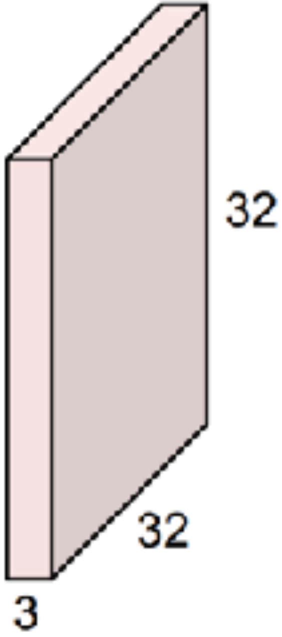


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

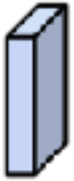
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

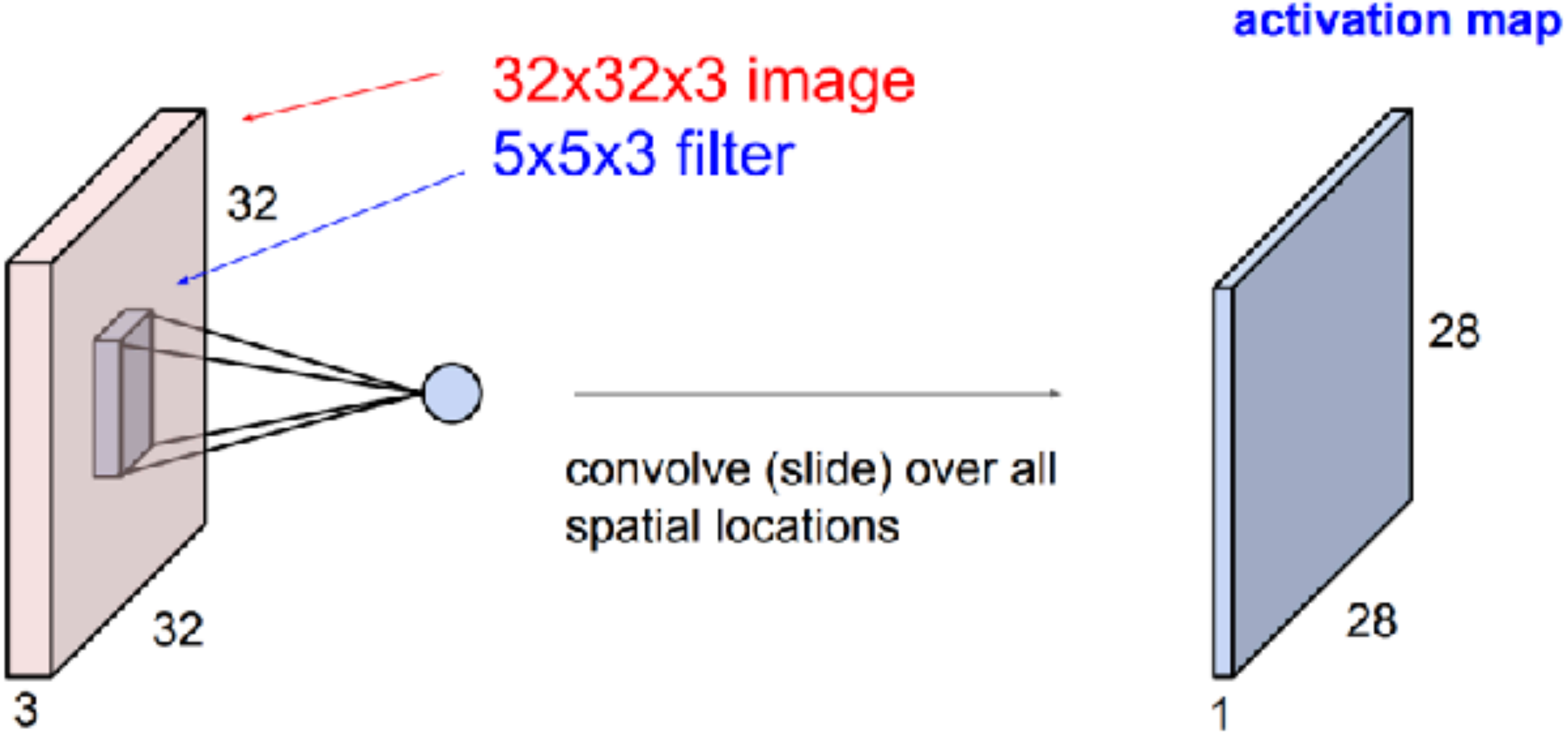


5x5x3 filter



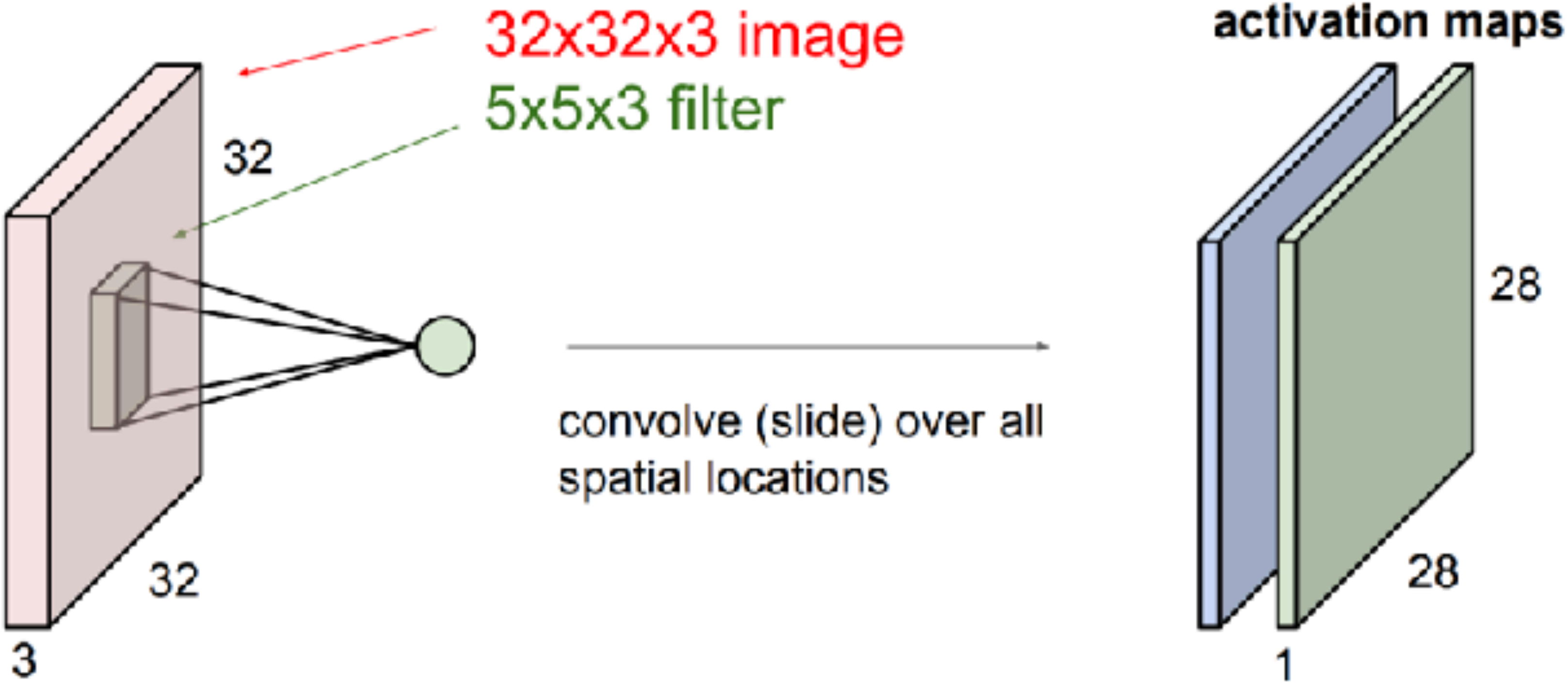
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Convolution Layer



Convolution Layer

consider a second, green filter



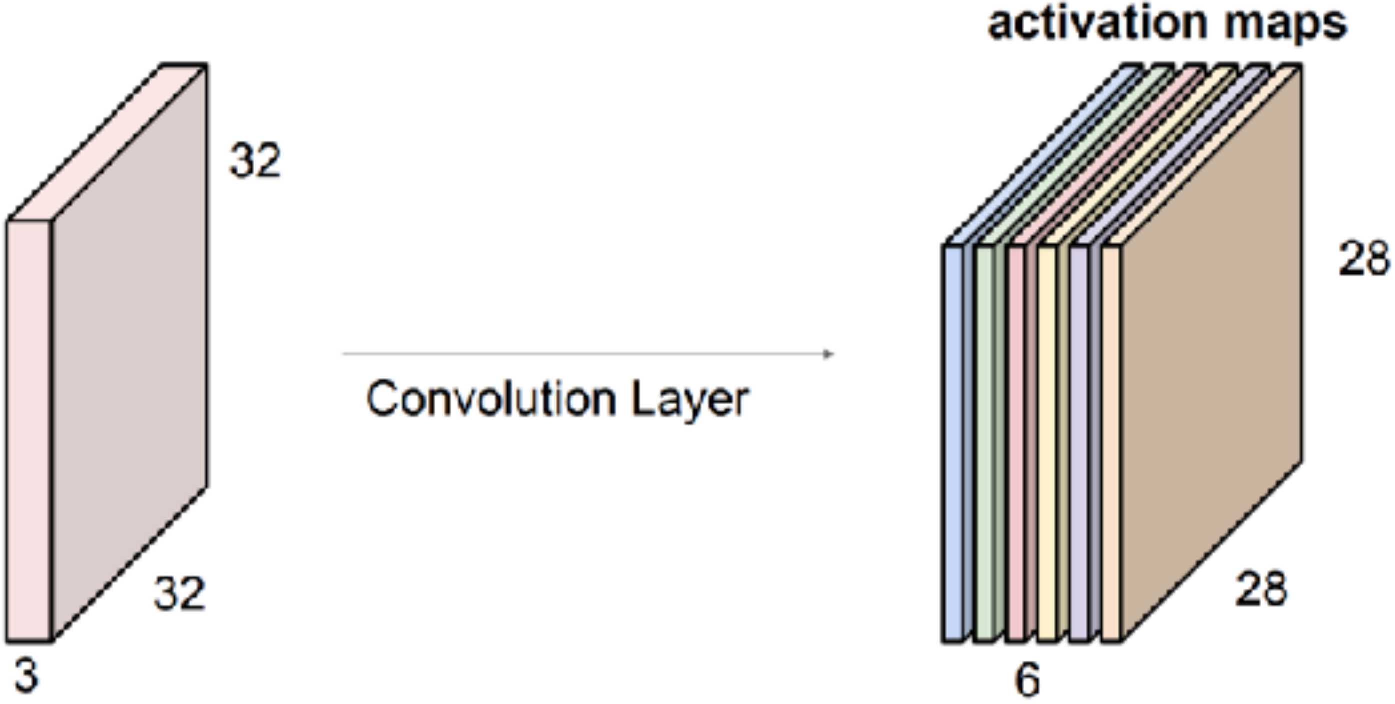
32x32x3 image

5x5x3 filter

convolve (slide) over all spatial locations

activation maps

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



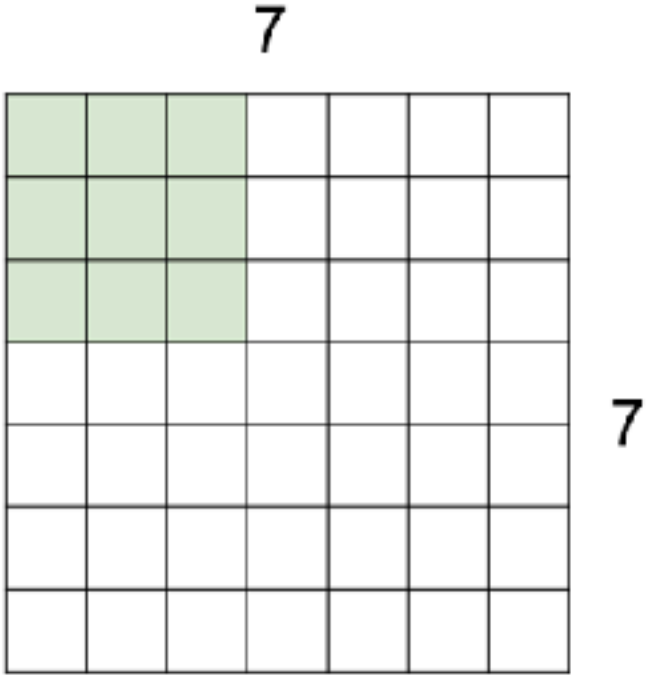
We stack these up to get a "new image" of size 28x28x6!

Convolutional Layer

Filter

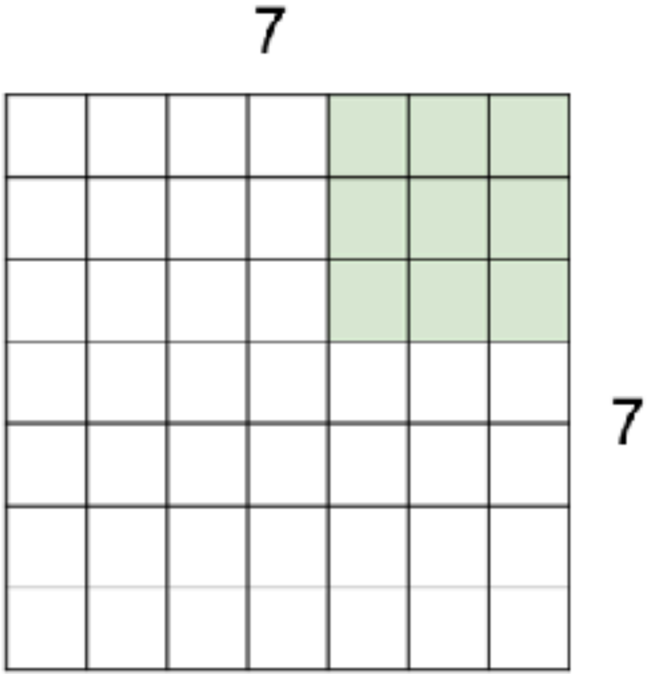
```
# for Conv Layer 01 filter - shape=(5, 5, 3)
W1 = tf.Variable(tf.random_normal([5, 5, 3, 32], stddev=0.01))

# for Conv Layer 02 filter - shape=(5, 5, 32)
W2 = tf.Variable(tf.random_normal([5, 5, 32, 64], stddev=0.01))
```

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

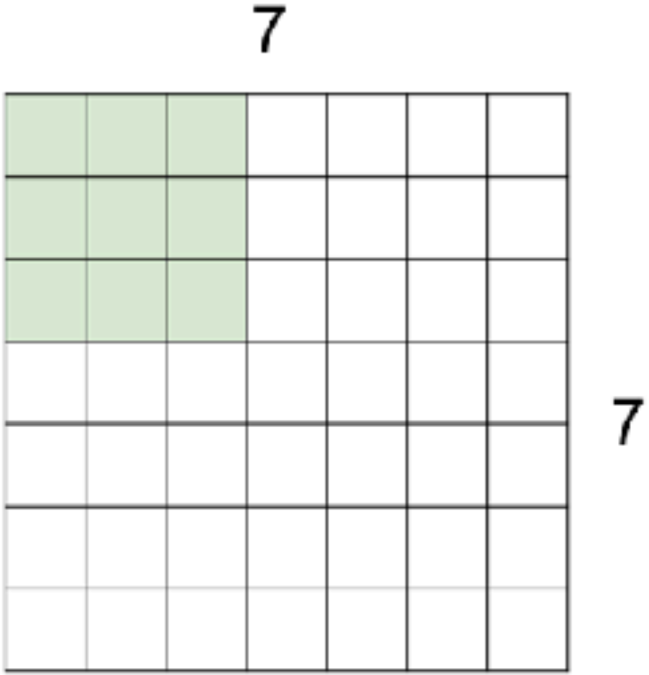
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

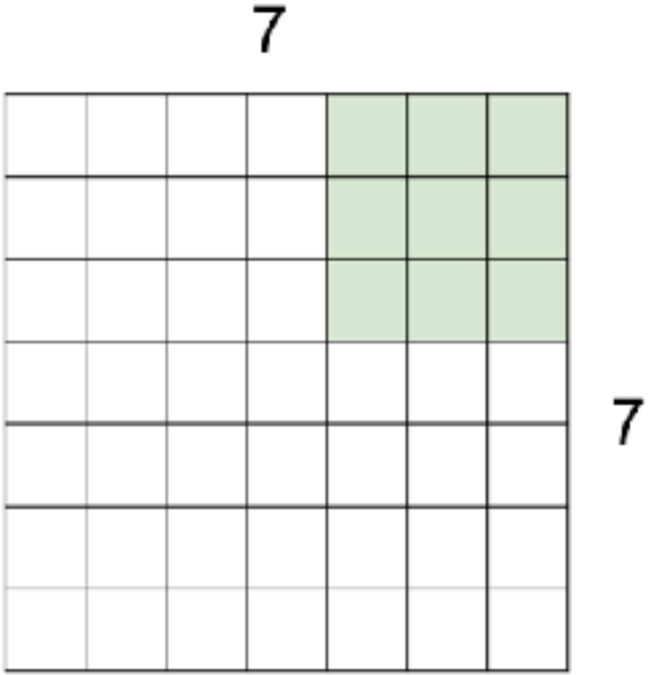
=> 5x5 output

A closer look at spatial dimensions:



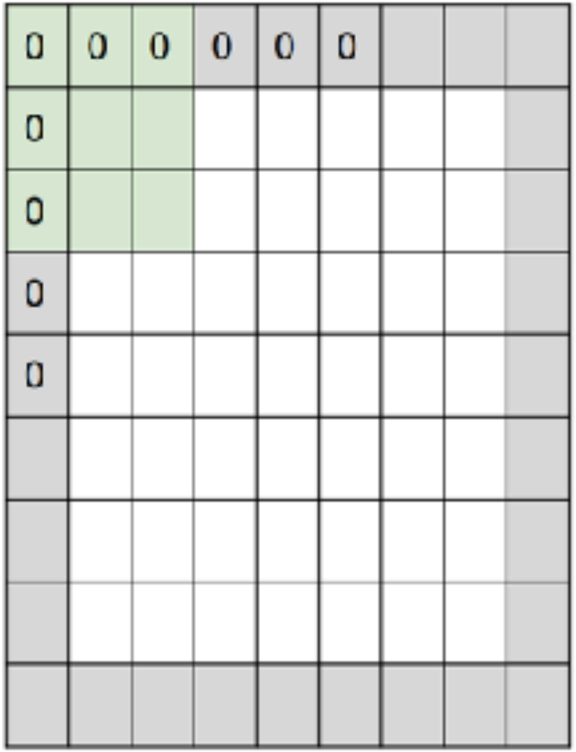
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



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

In practice: Common to zero pad the border



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

7x7 output!

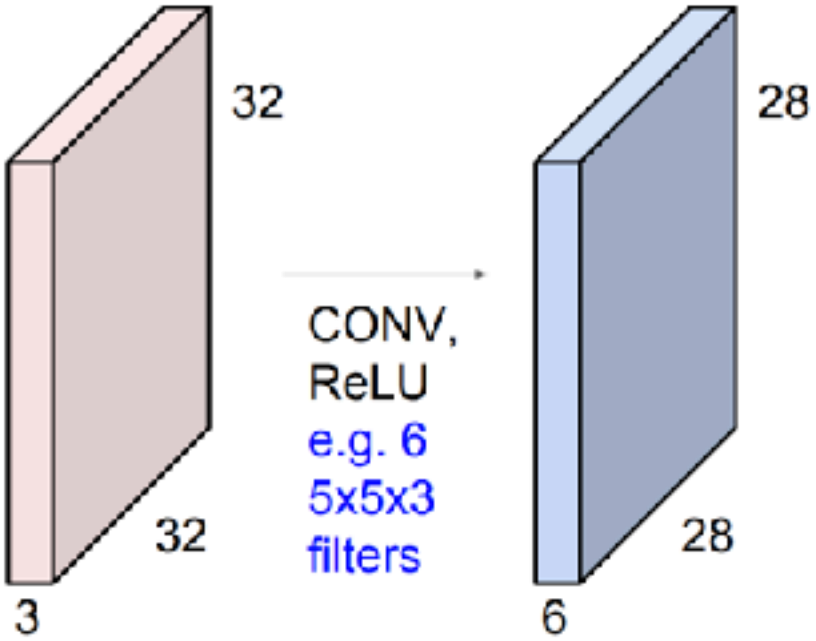
Convolutional Layer

Padding

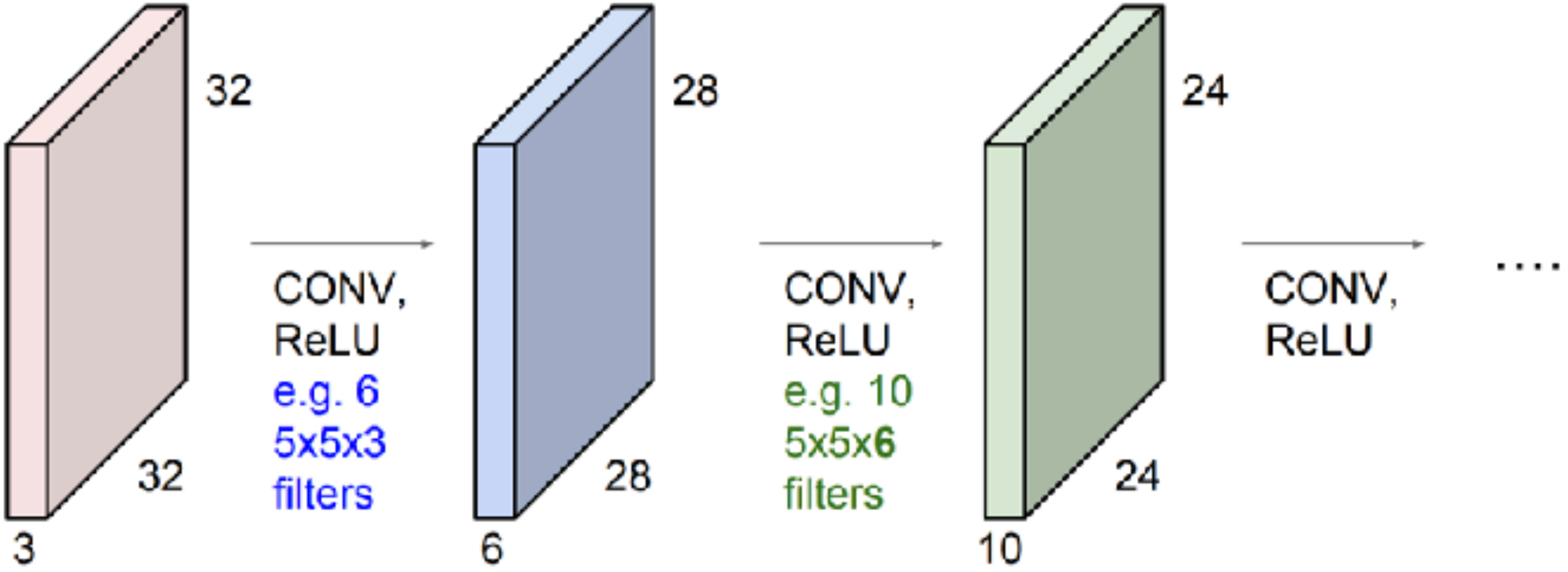
```
# Convolution Layer 01 -> (?, 28, 28, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)

# Pooling Layer 01 -> (?, 14, 14, 32)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding=‘SAME’)
```

참고: input과 같은 size 28x28의 결과를 얻기 위해서
padding 1이 내부적으로 적용 되었음.

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



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



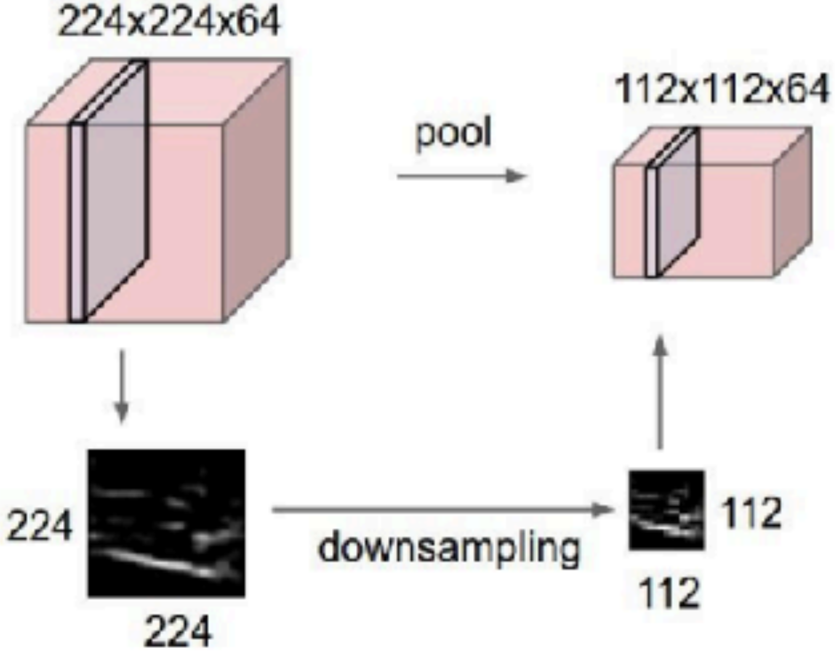
Convolutional Layer

Activation function

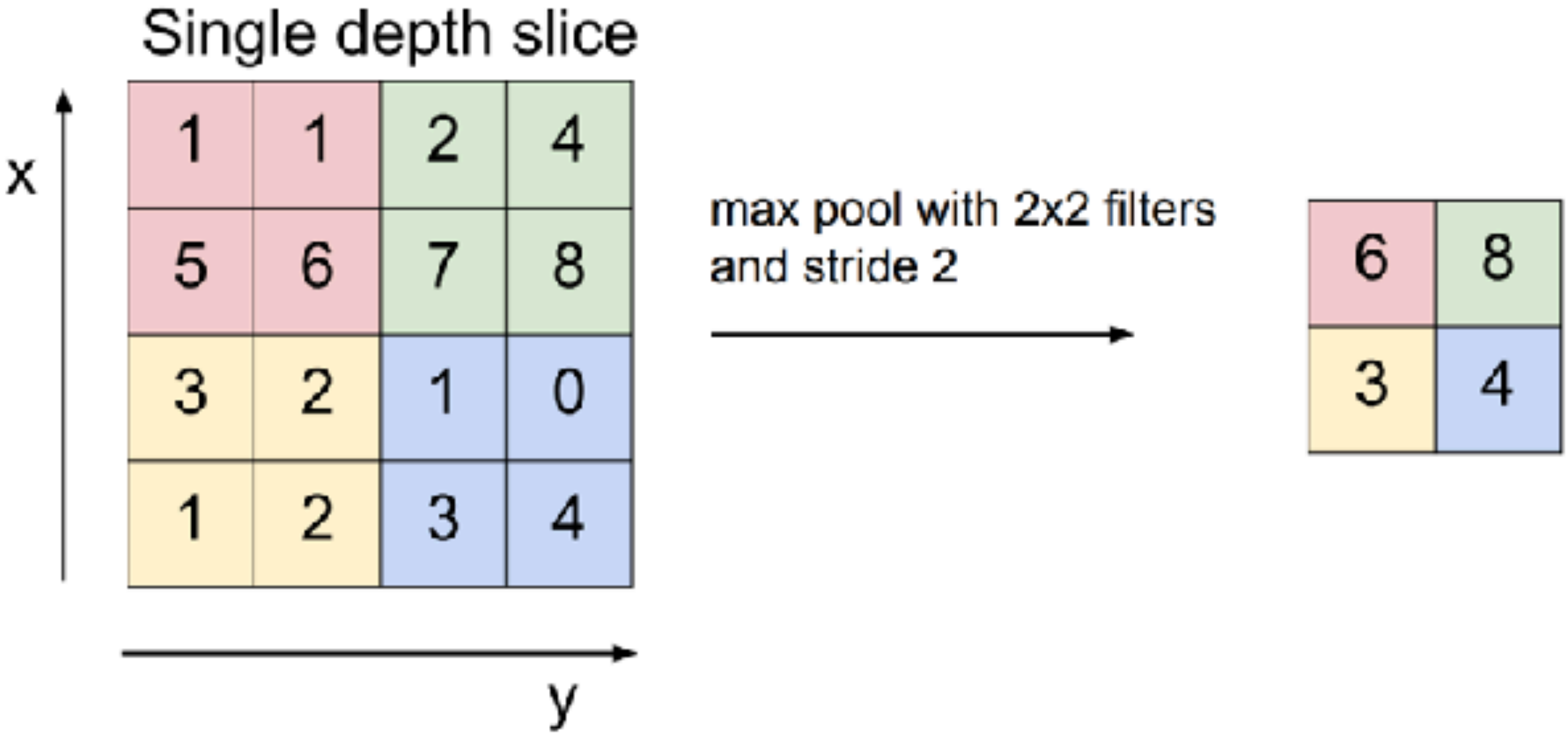
```
# Convolution Layer 01 -> (?, 28, 28, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)

# Pooling Layer 01 -> (?, 14, 14, 32)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding=‘SAME’)
```

Pooling layer

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



MAX POOLING



Single depth slice

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

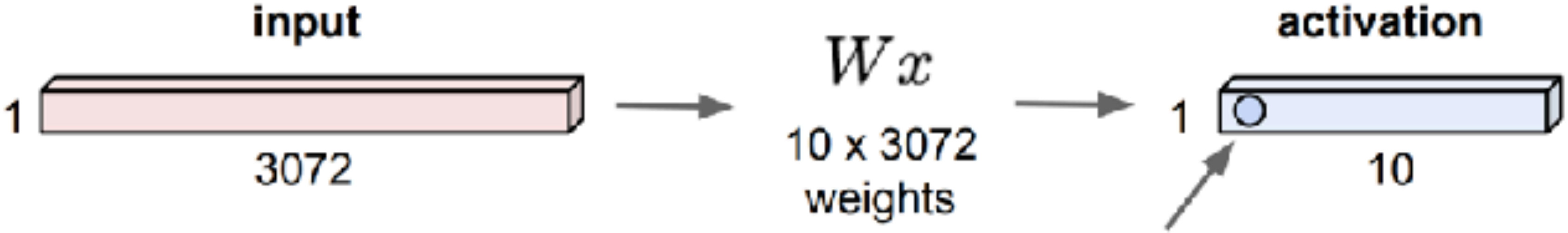
Pooling Layer

Sub sampling - Max Pooling

```
# Convolution Layer 01 -> (?, 28, 28, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)

# Pooling Layer 01 -> (?, 14, 14, 32)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding=‘SAME’)
```

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

Fully Connected Layer

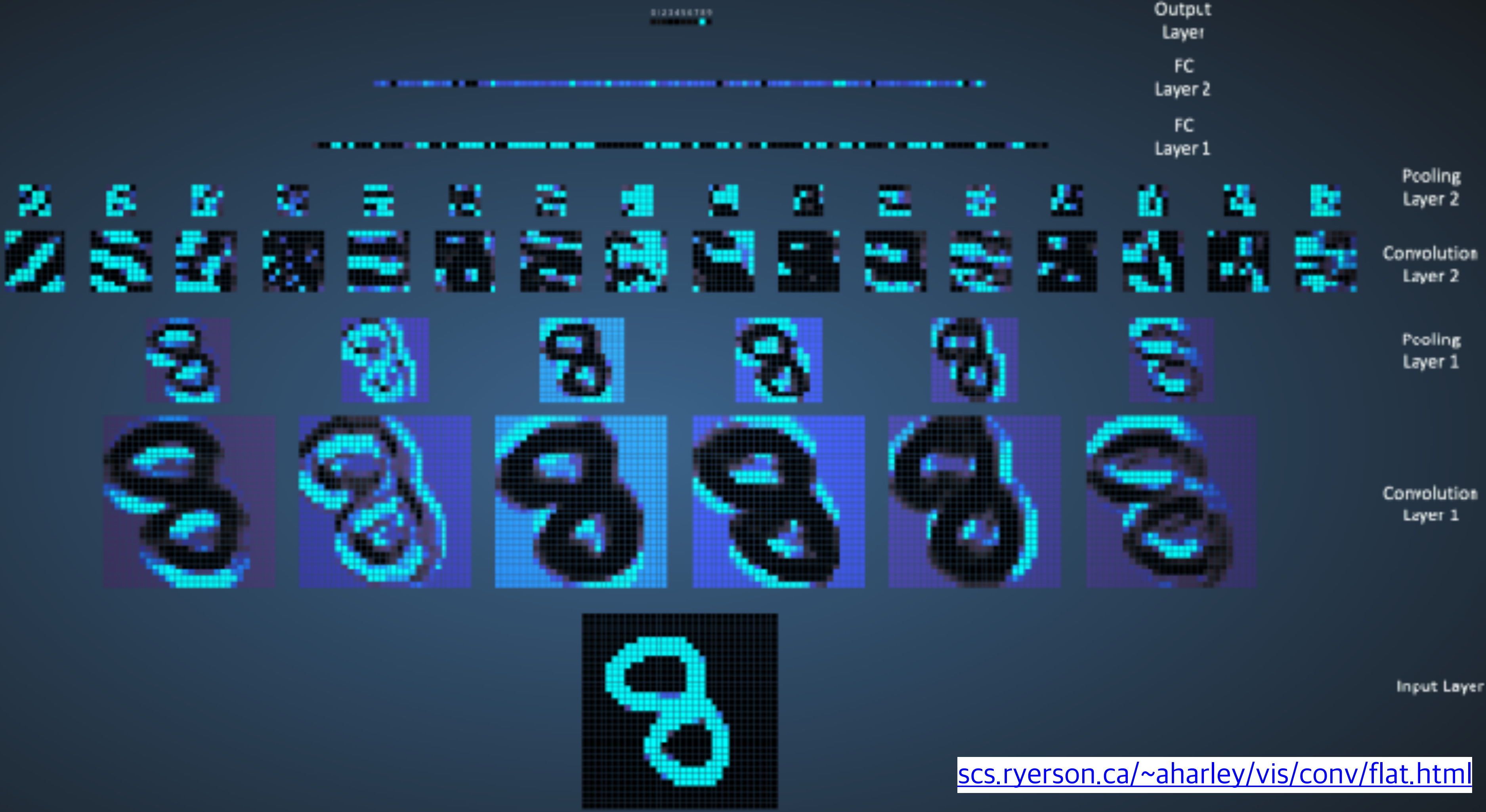
Flatten

```
# flatten
L_FC = tf.reshape(L2_pool, [-1, 7 * 7 * 64], name=“L_flat")

# logits = W_fc * L_FC + bias
logits = tf.add(tf.matmul(L_FC, W_fc), b, name='logits')
```

Cost & Optimizer

Softmax

```
# Cost(loss) function & Optimizer
cost = tf.reduce_mean(
               tf.nn.softmax_cross_entropy_with_logits(
                   logits=logits, labels=Y))
optimizer = tf.train
                       .AdamOptimizer(learning_rate=learning_rate)
                       .minimize(cost)
```



Output Layer

FC Layer 2

FC Layer 1

Pooling Layer 2

Convolution Layer 2

Pooling Layer 1

Convolution Layer 1

Input Layer

scs.ryerson.ca/~aharley/vis/conv/flat.html