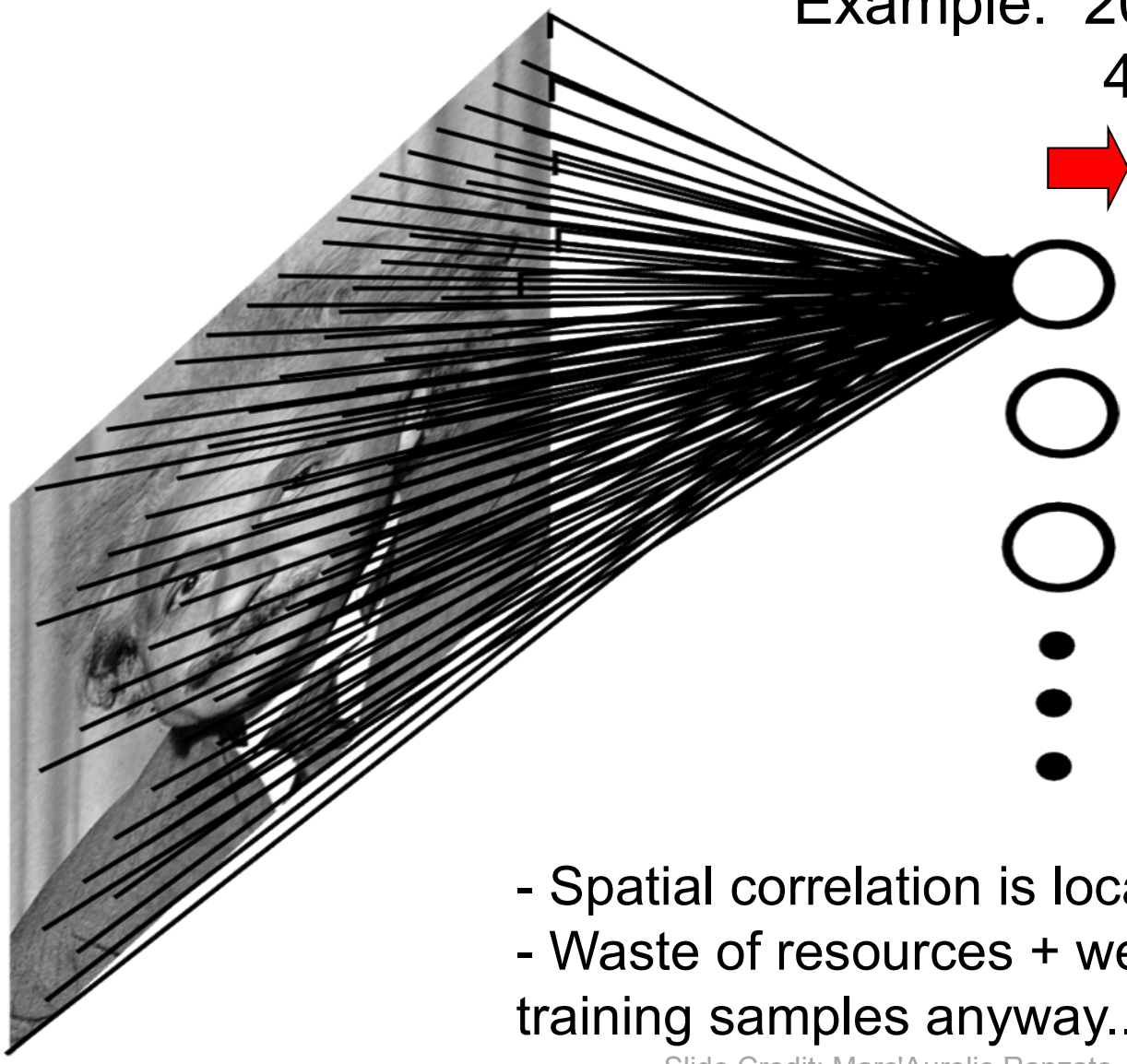
# 고급 소프트웨어 실습(CSE4152)

# Convolutional Neural Network

(CNN)

# Fully Connected Layer

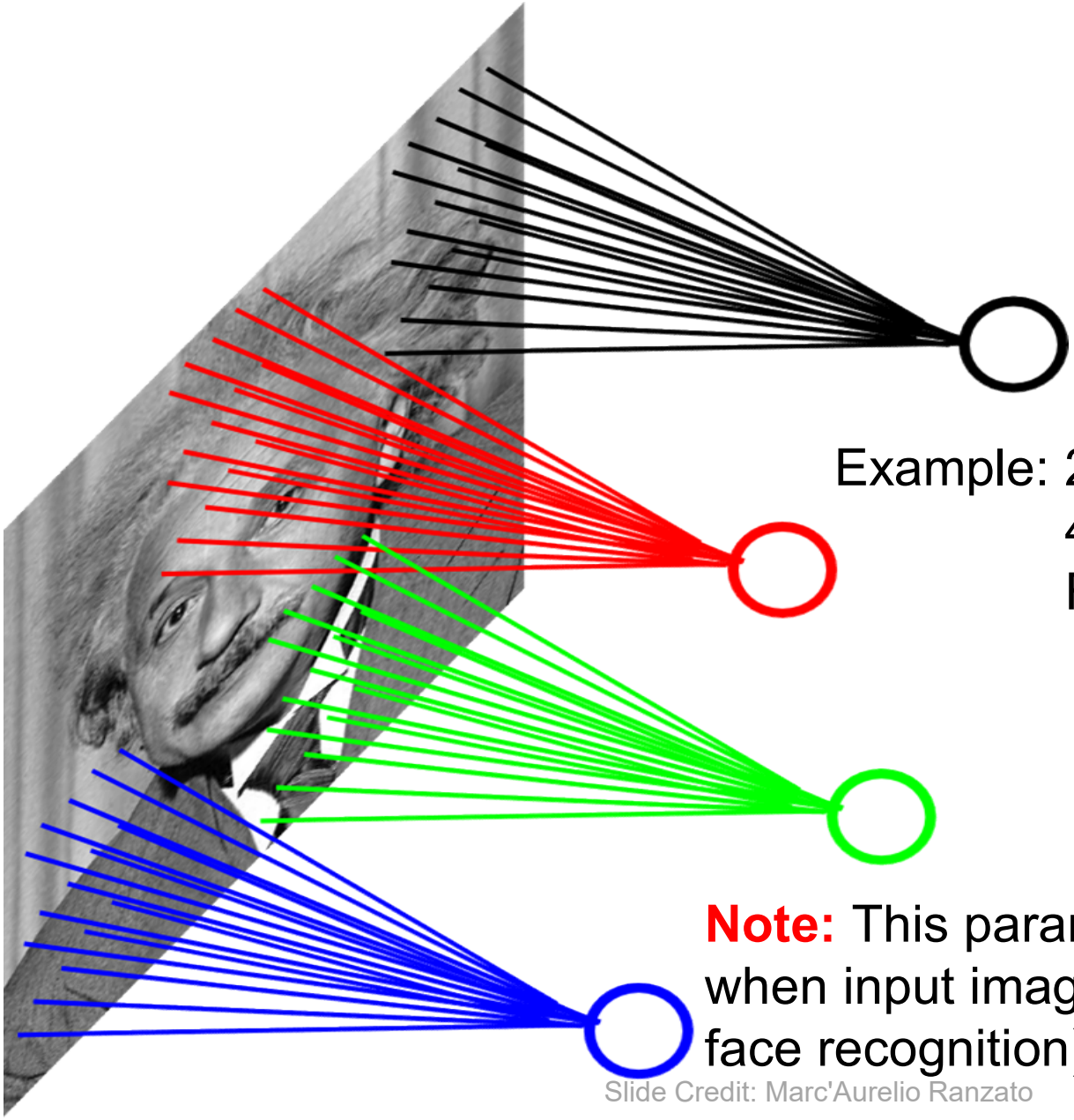


Example: 200x200 image
40K hidden units
**~2B parameters**!!!

40K x 40K = 1600M = 1.6 B

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

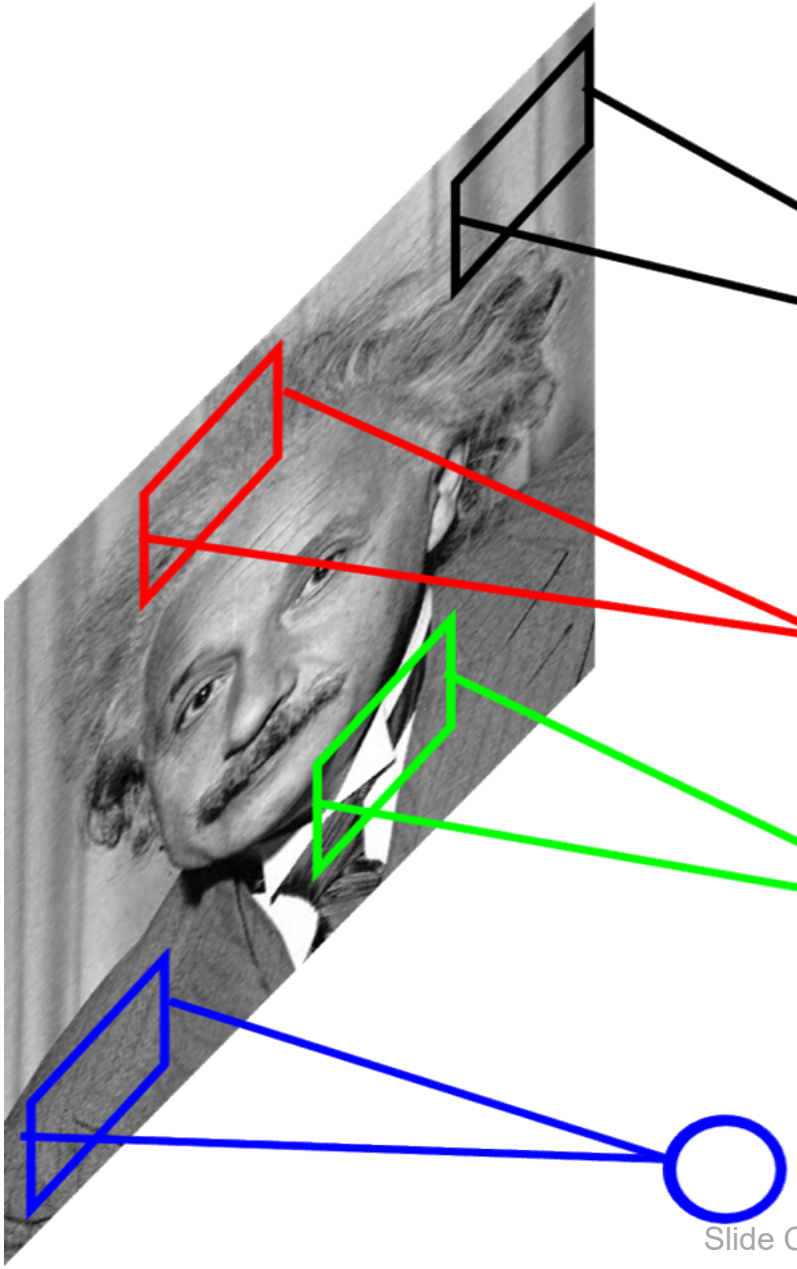
# Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

**Note:** This parameterization is good when input image is registered (e.g., face recognition).

# Locally Connected Layer

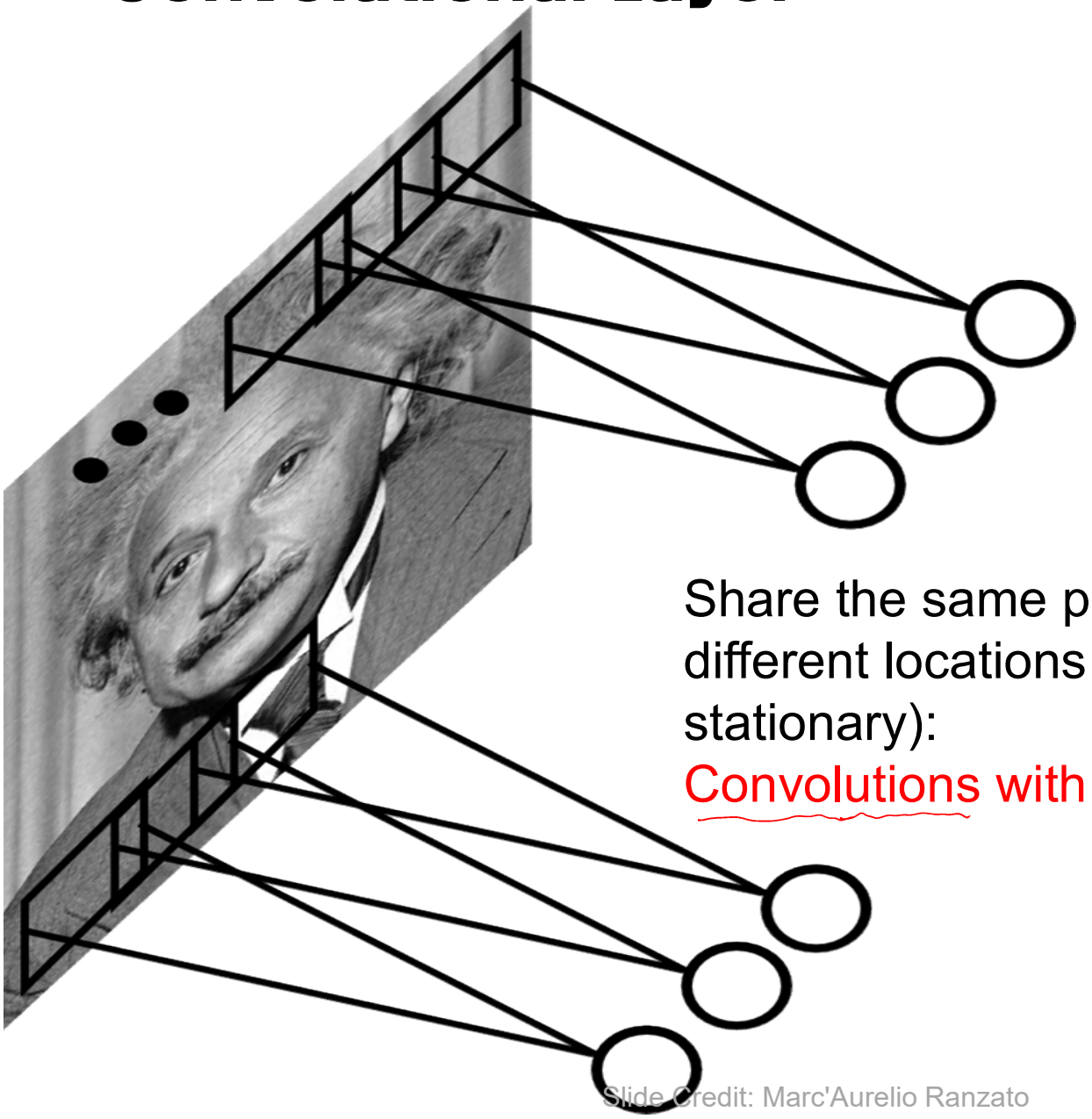


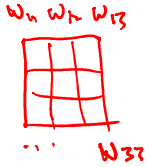
**STATIONARITY?** Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

**Note:** This parameterization is good when input image is registered (e.g., face recognition).

# Convolutional Layer

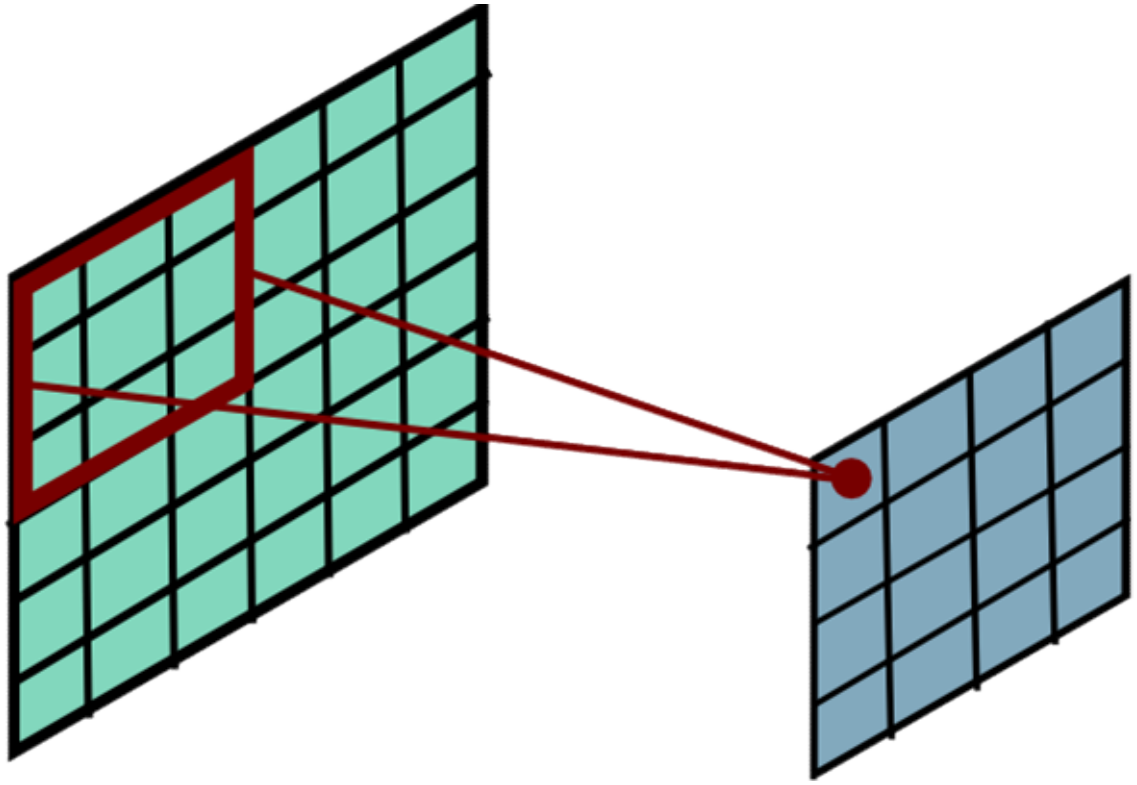




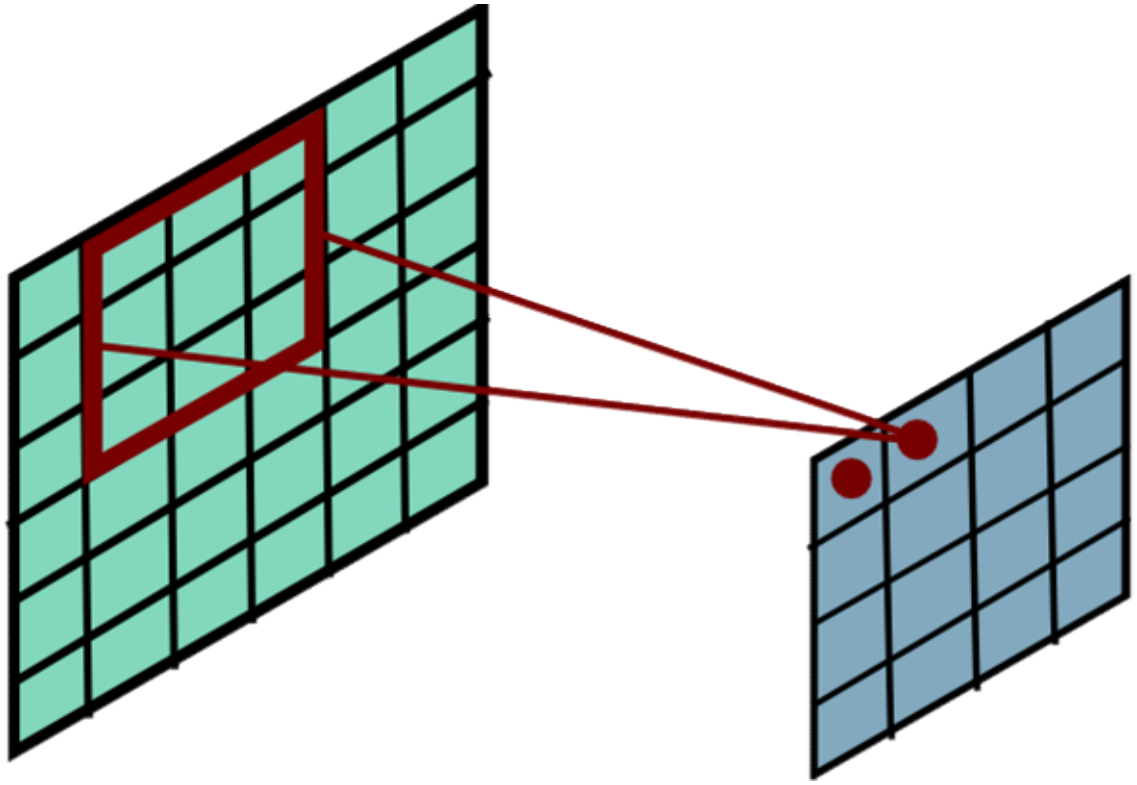
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

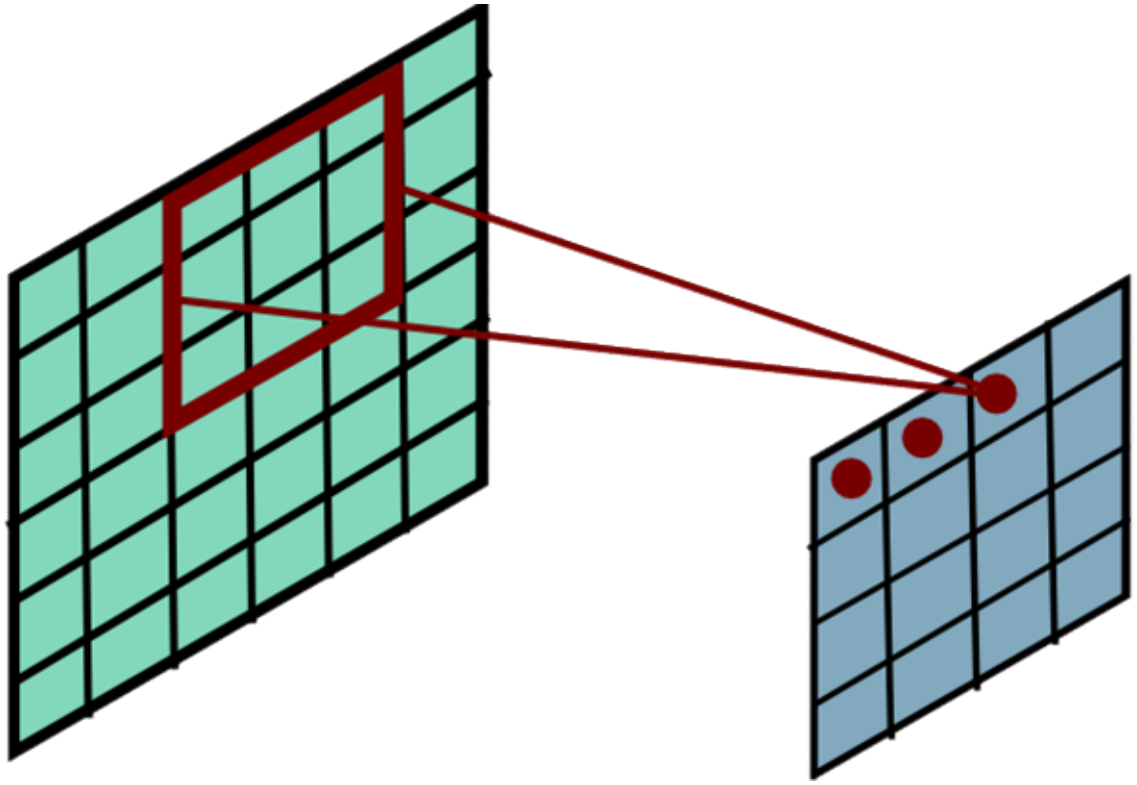
# Convolutional Layer



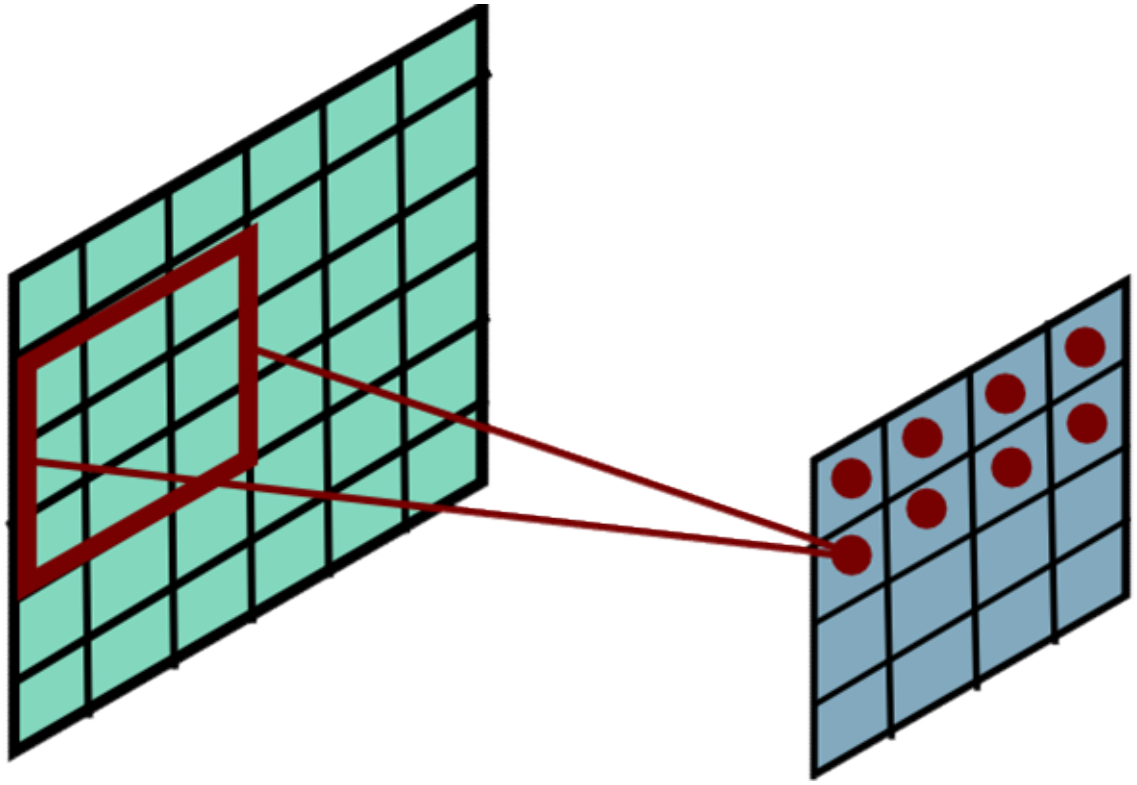
# Convolutional Layer



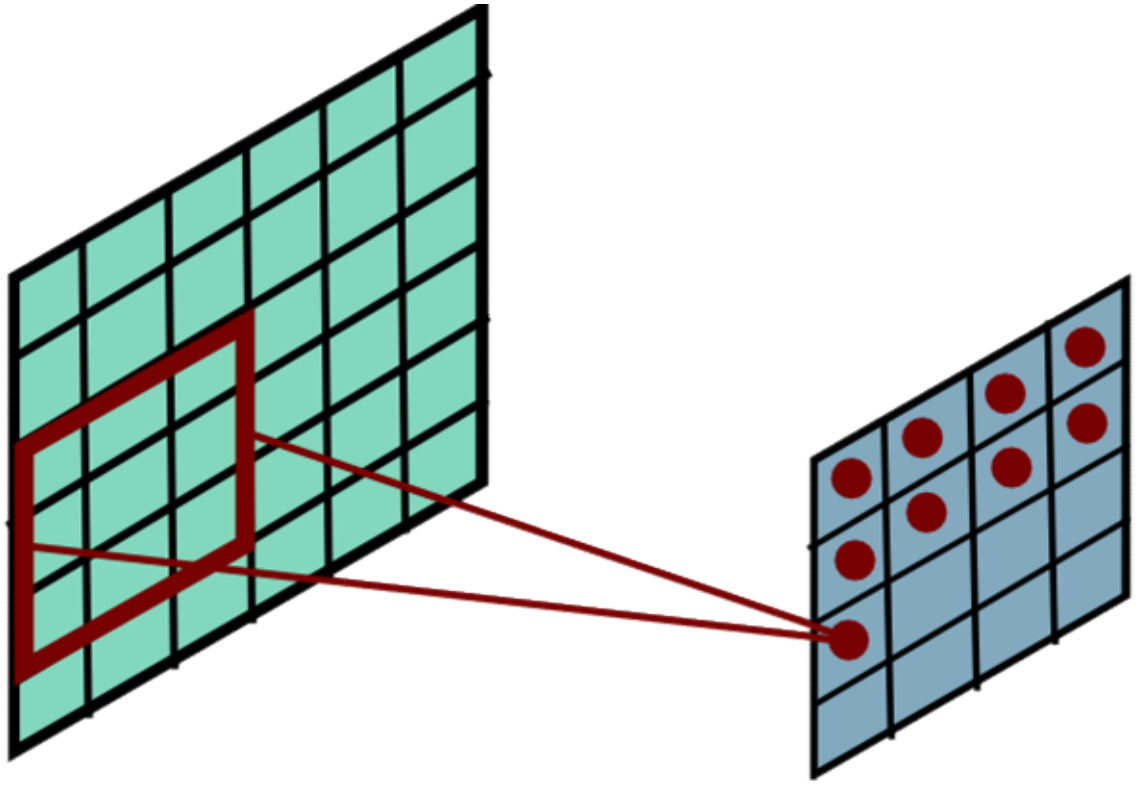
# Convolutional Layer



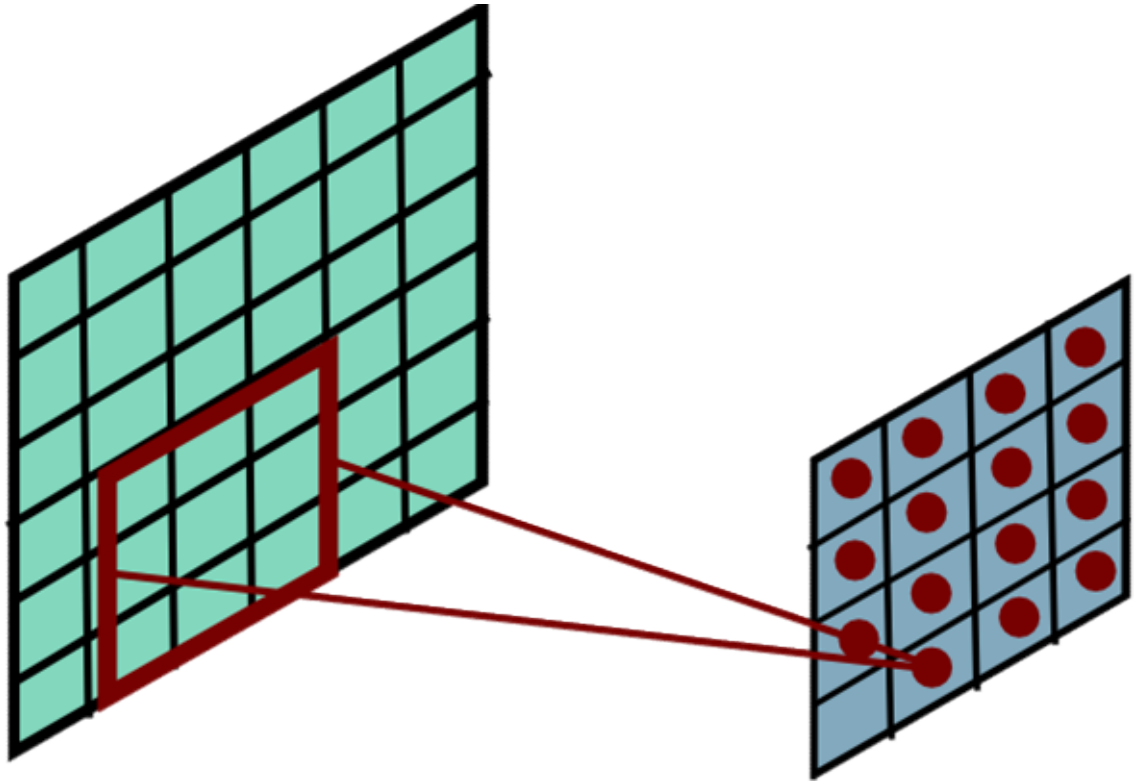
# Convolutional Layer



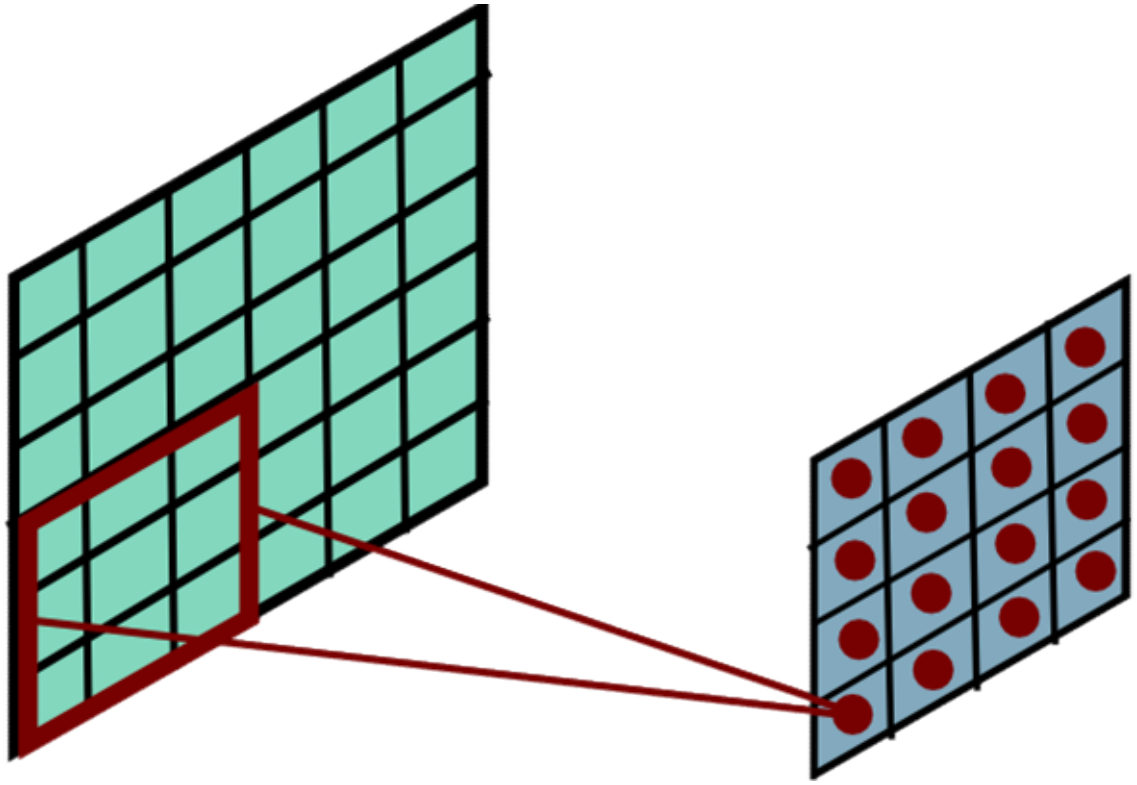
# Convolutional Layer



# Convolutional Layer

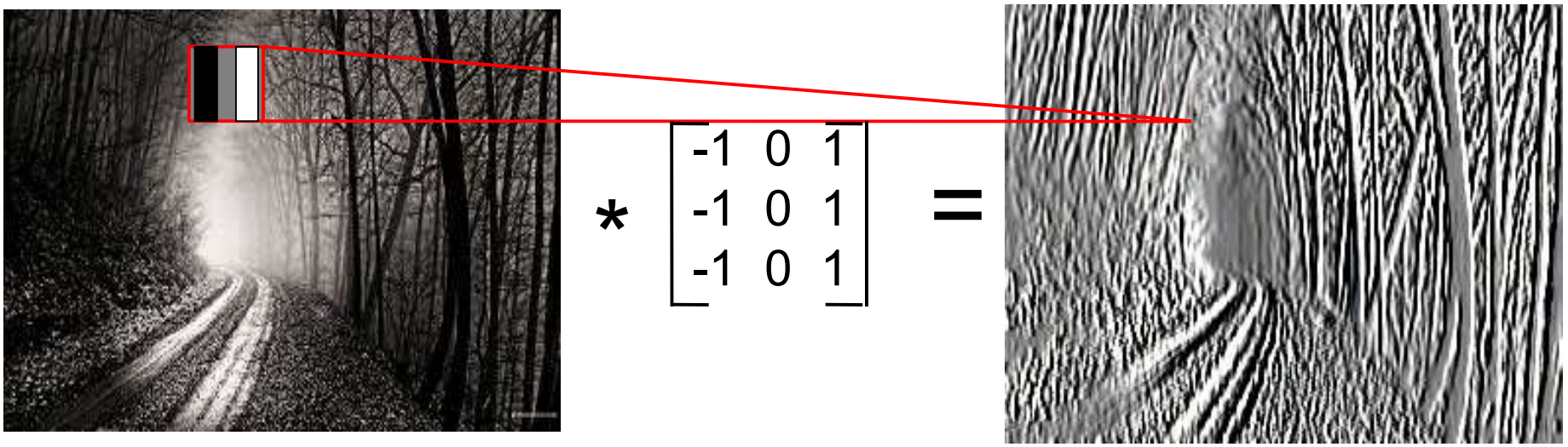


# Convolutional Layer

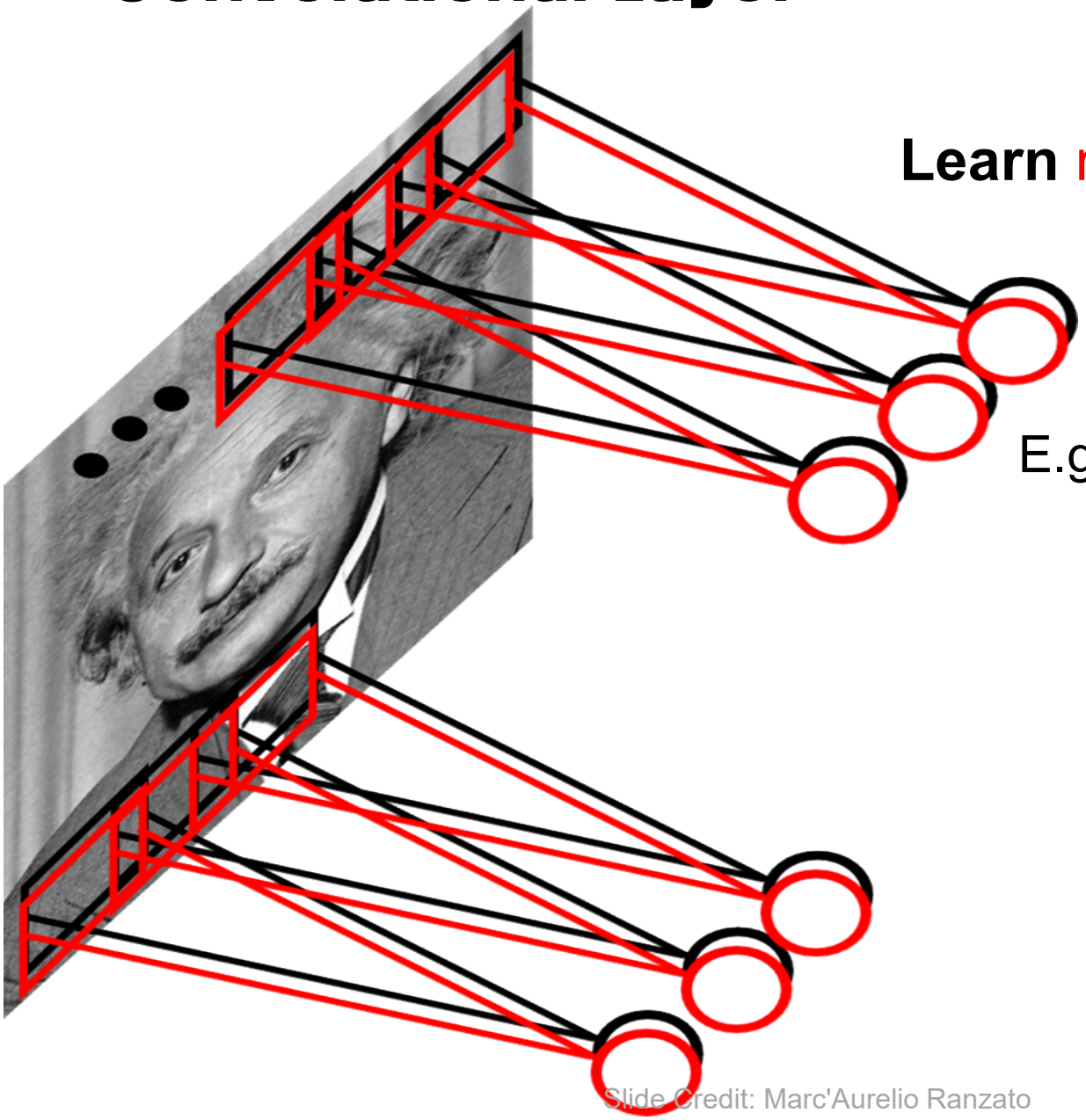


Mathieu et al. “Fast training of CNNs through FFTs” ICLR 2014

# Convolutional Layer



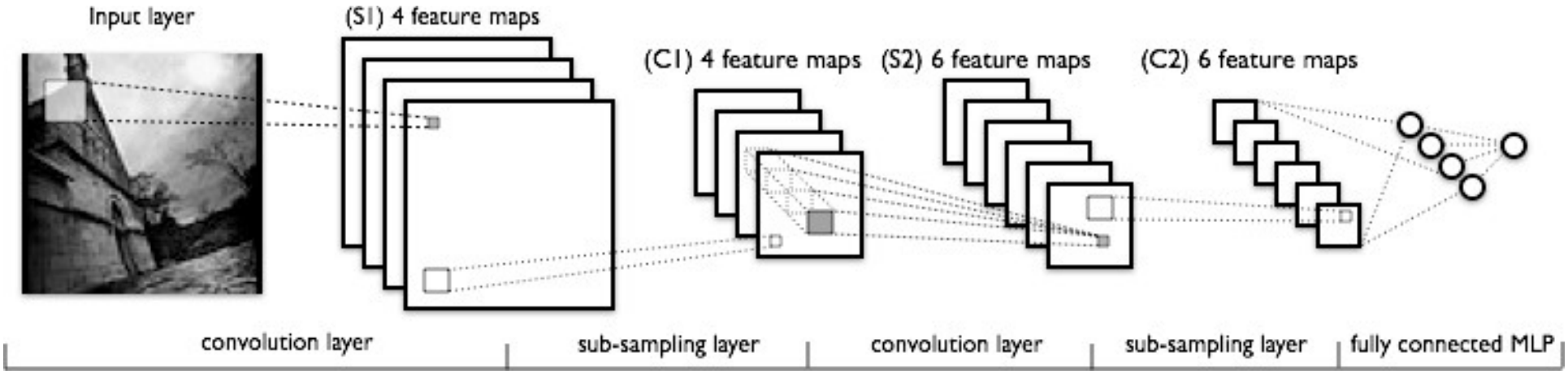
# Convolutional Layer



**Learn** multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

# Convolutional Neural Networks



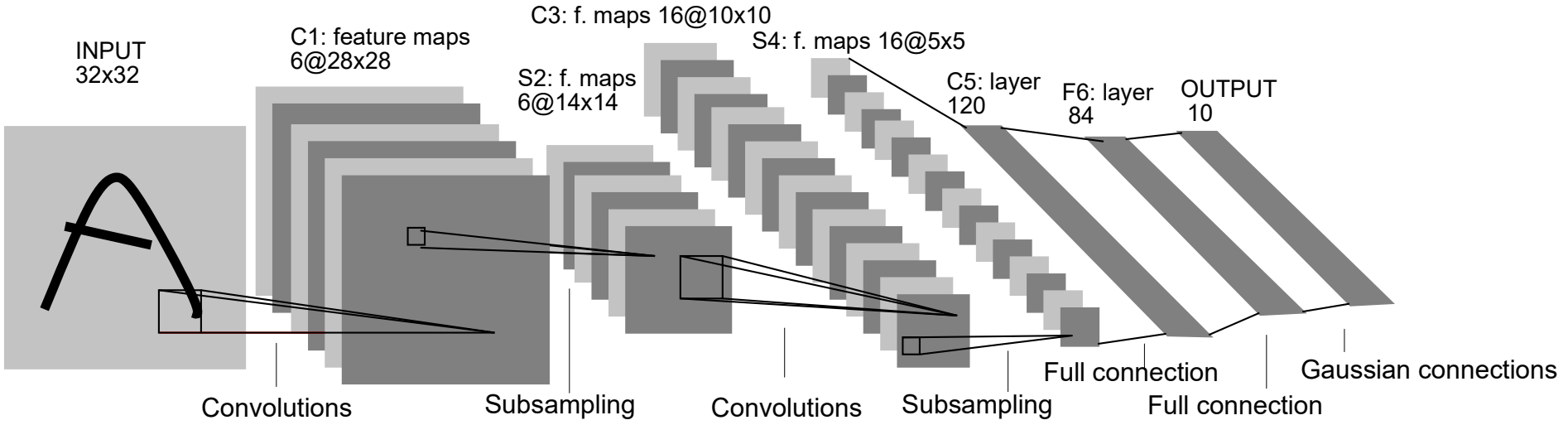
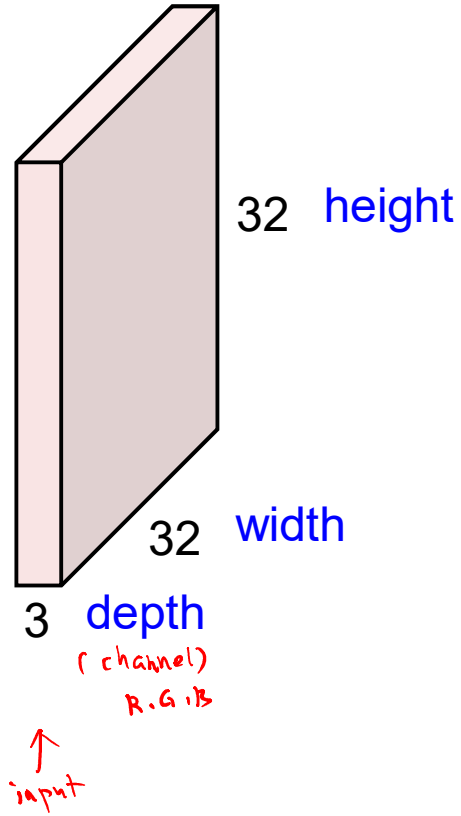




Image Credit: Yann LeCun, Kevin Murphy

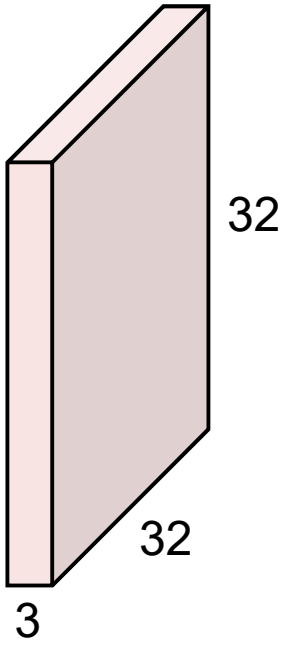
# Convolution Layer

32x32x3 image -> preserve spatial structure

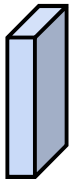


# Convolution Layer

32x32x3 image



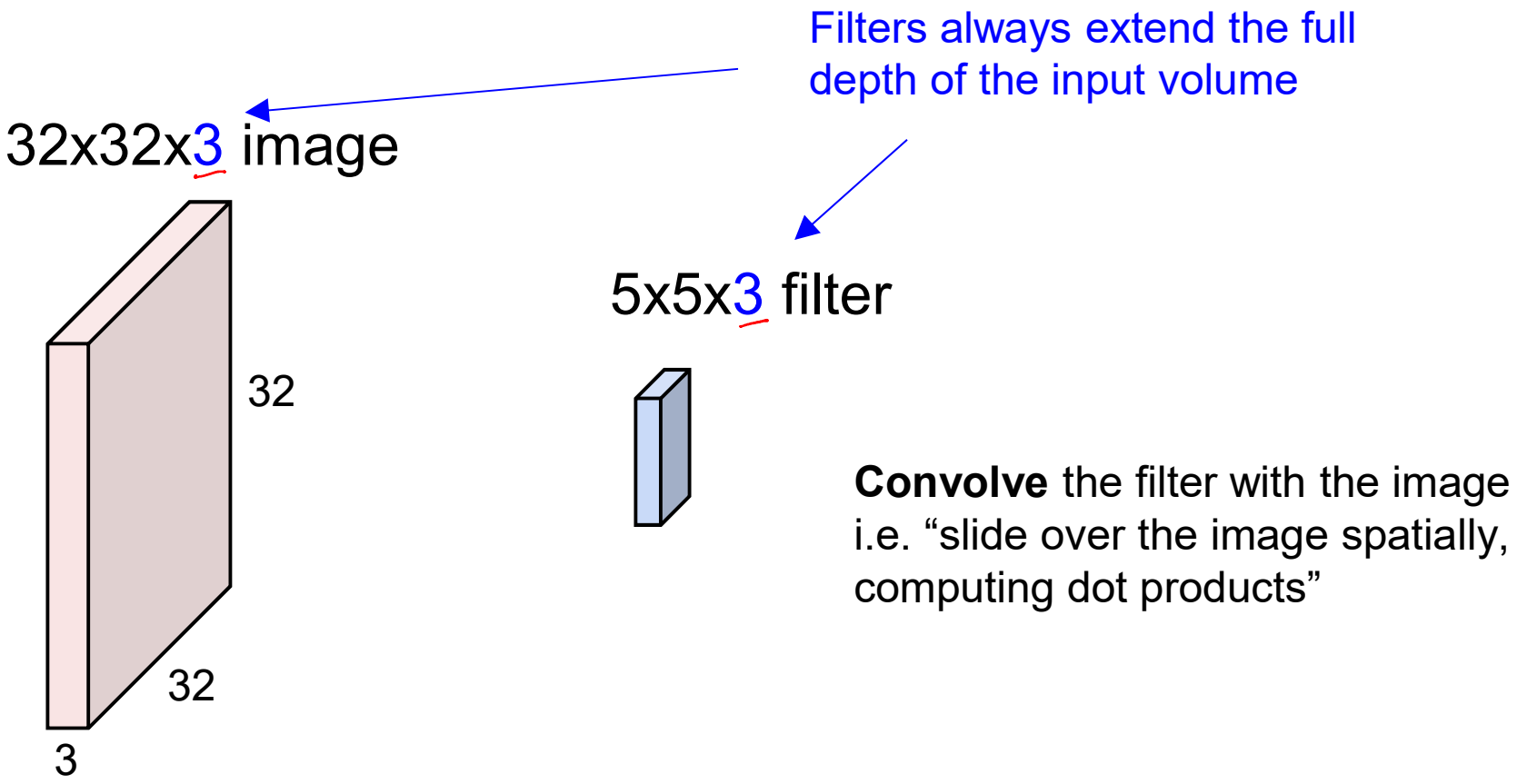

5x5x3 filter



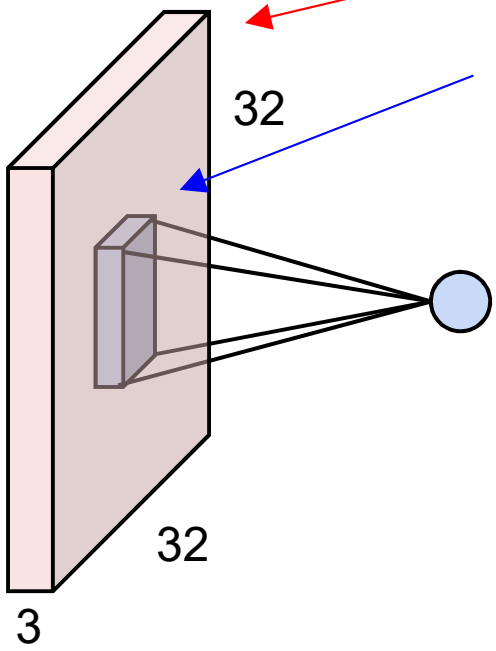
**Convolve** the filter with the image
i.e. “slide over the image spatially, computing dot products”

2D convolution
3D convolution ~~ 3D convolution in NN
Conv2D( )

# Convolution Layer



# Convolution Layer



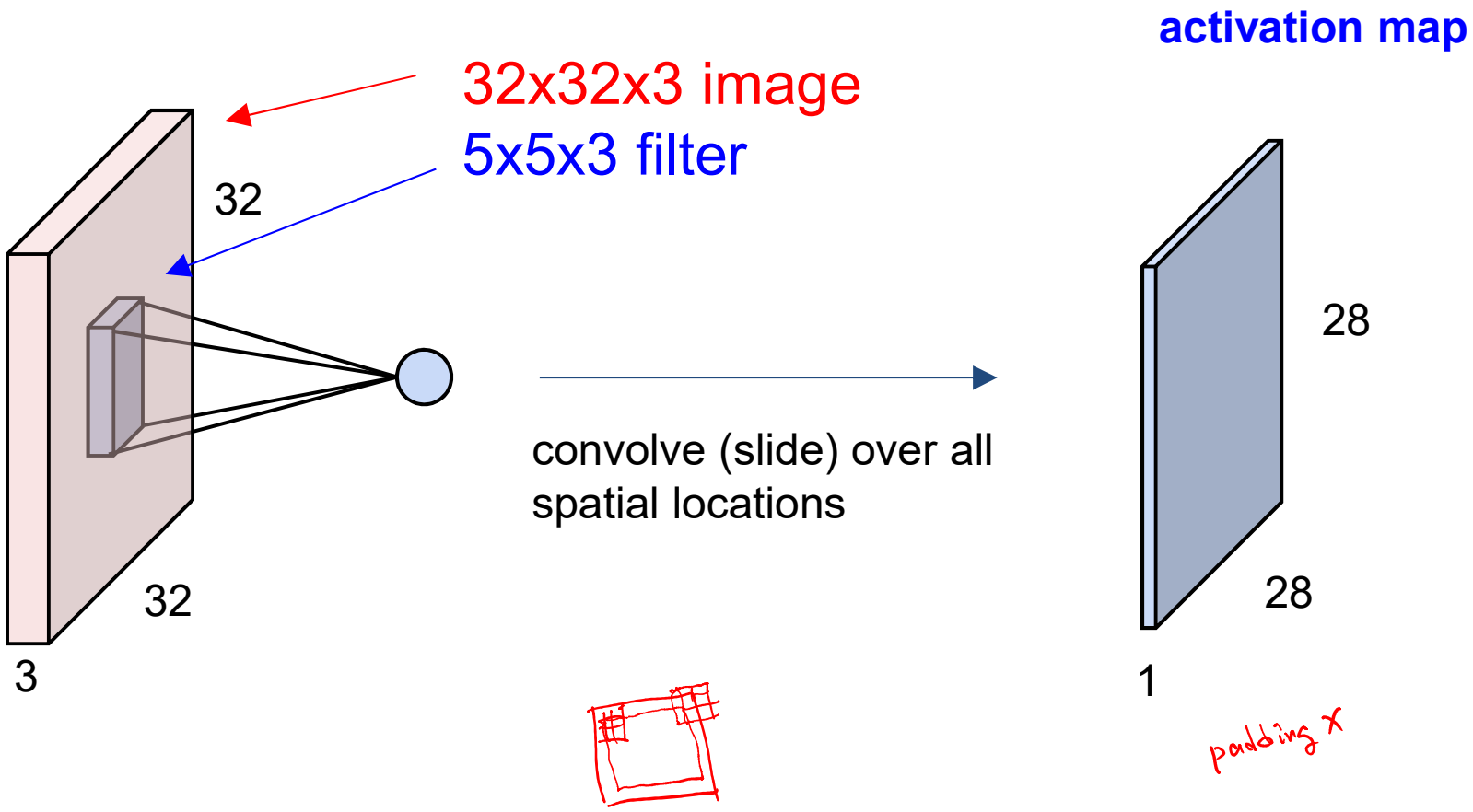

32x32x3 image

5x5x3 filter $w$

**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

# Convolution Layer

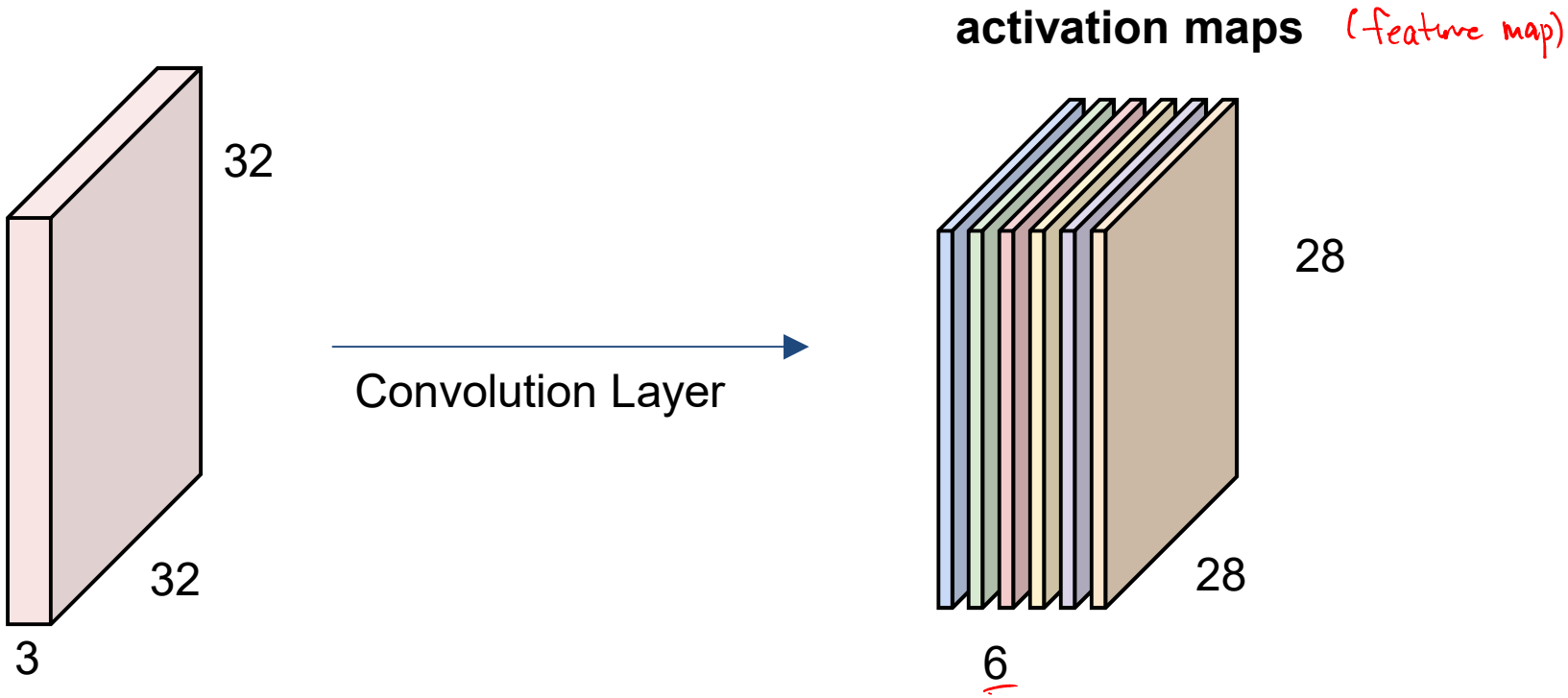


# Convolution Layer

consider a second, green filter

# Convolution Layer

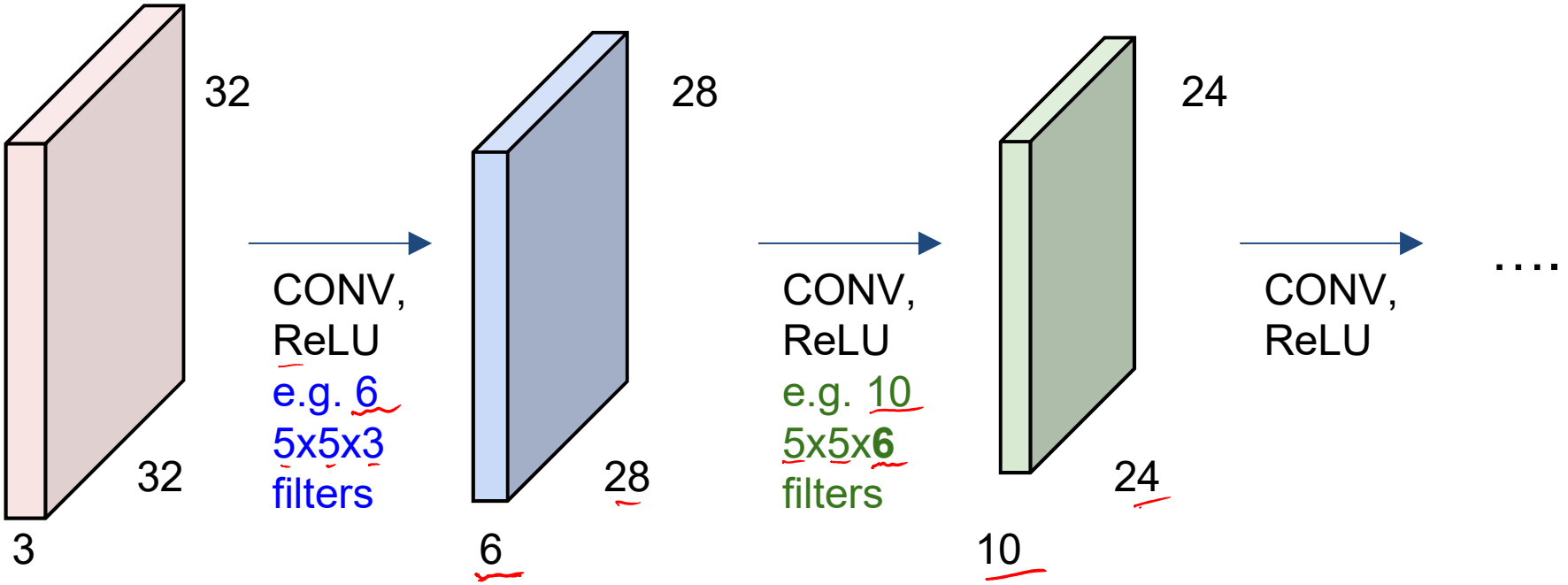
For example, if we had 6 5x5x3 filters, we'll get 6 separate activation maps:

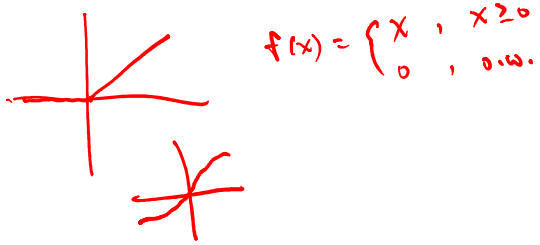



We stack these up to get a "new image" of size 28x28x6!

# Convolution Layer

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



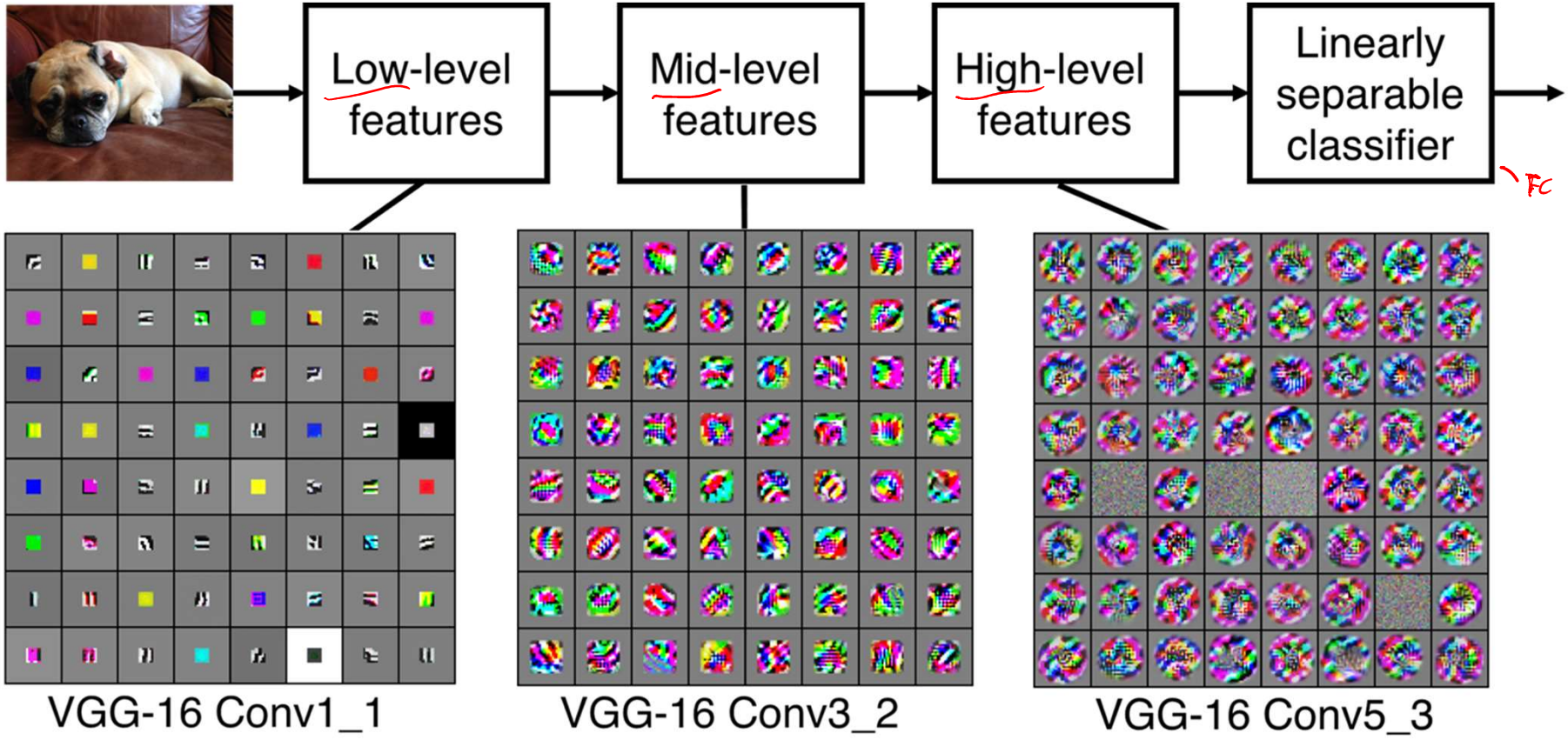

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

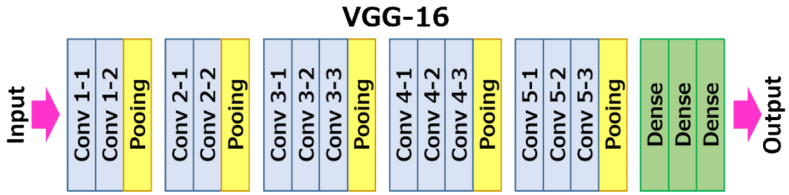
# Convolutional Layer

**Preview**

*[Zeiler and Fergus 2013]*

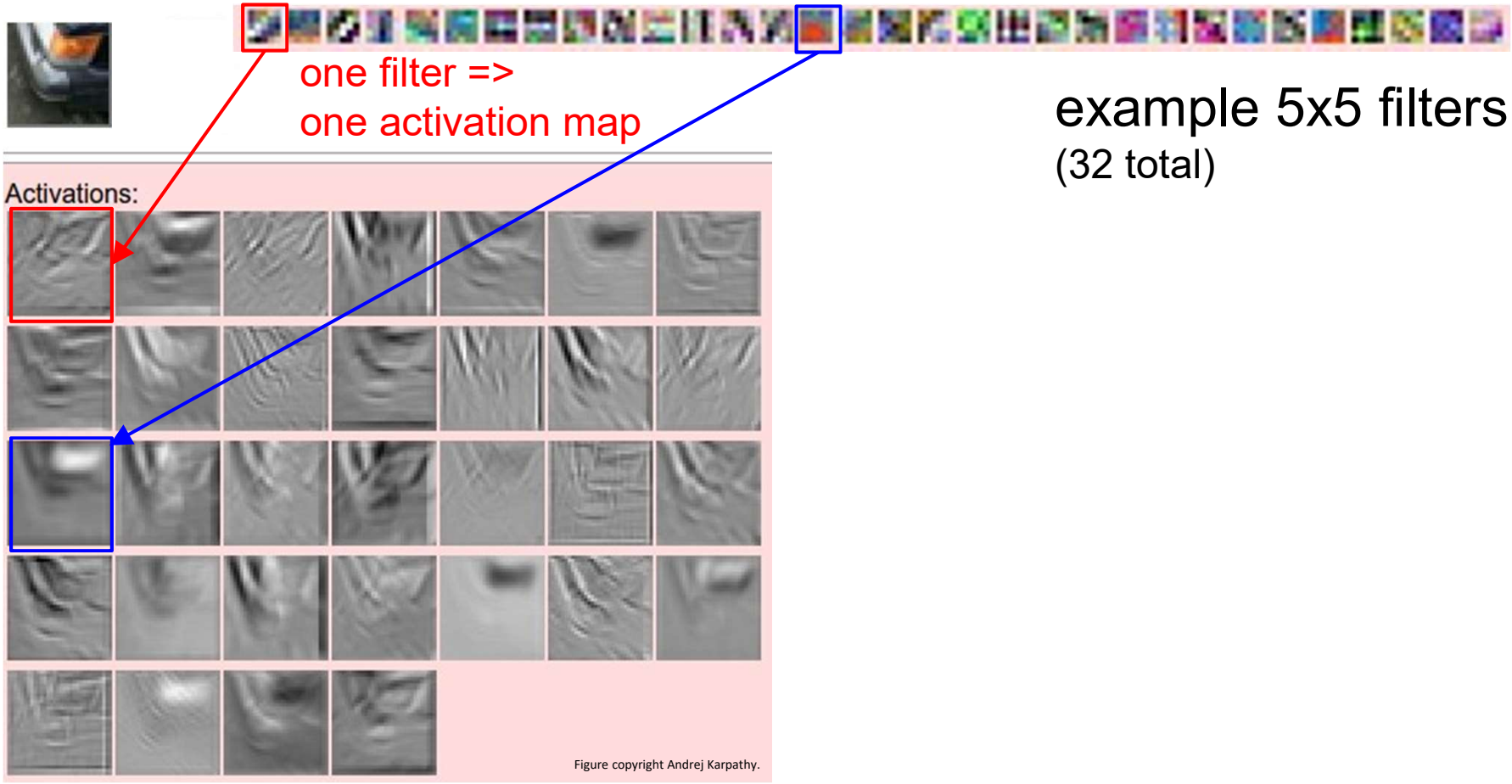
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

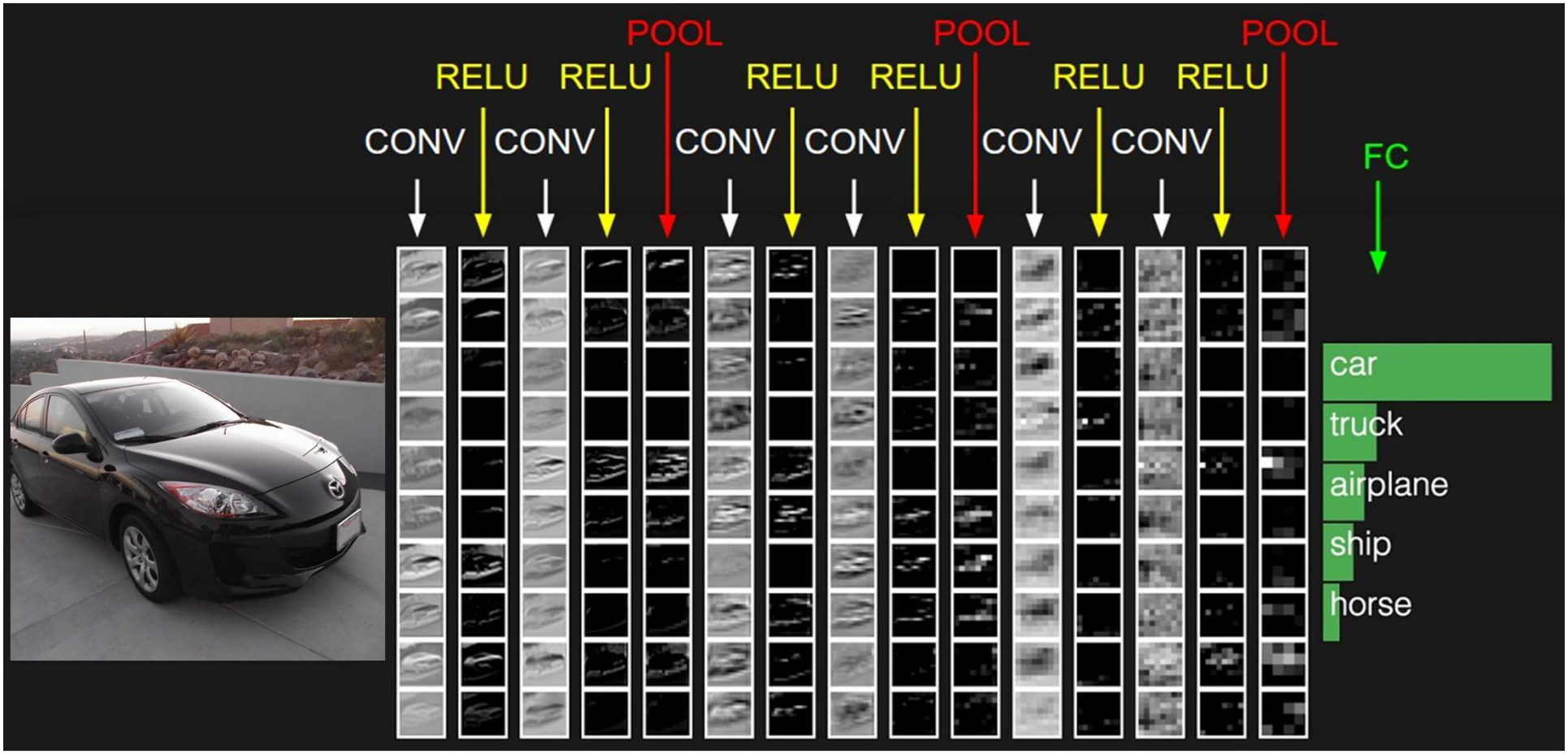
# Convolutional Layer




example 5x5 filters
(32 total)

# Convolutional Layer

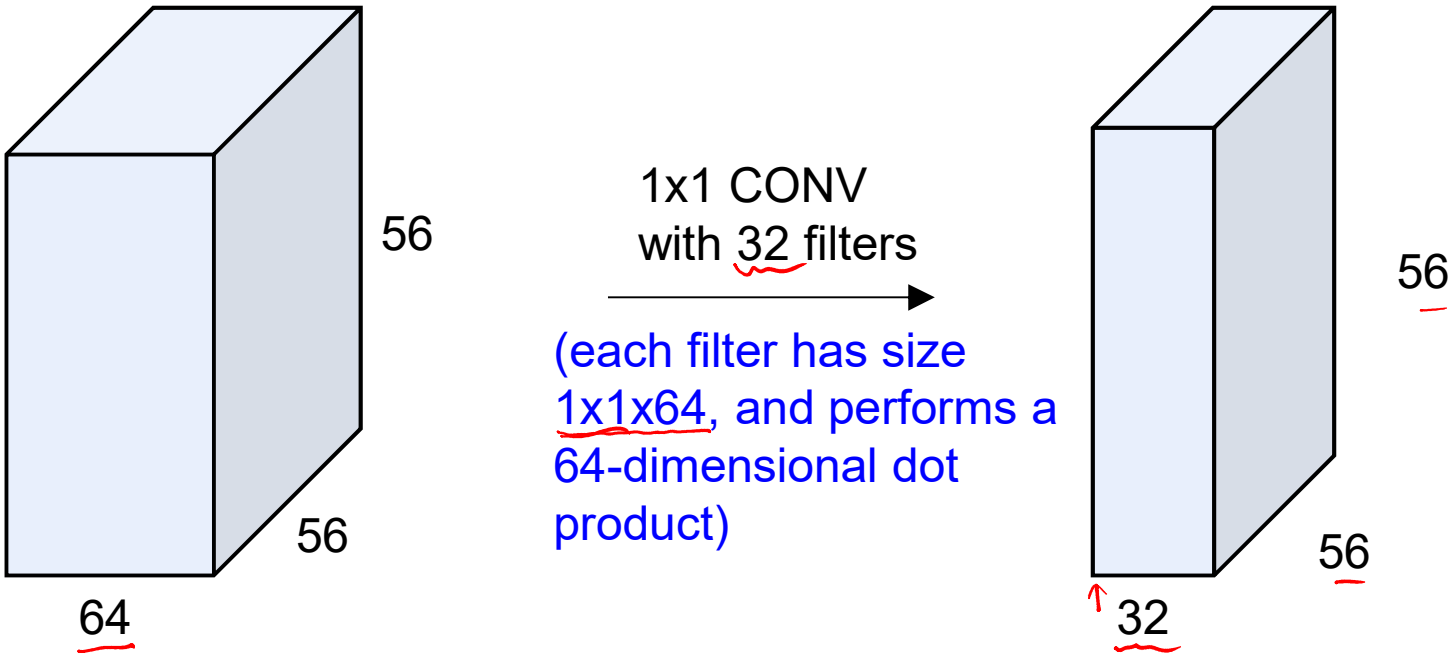
preview:




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

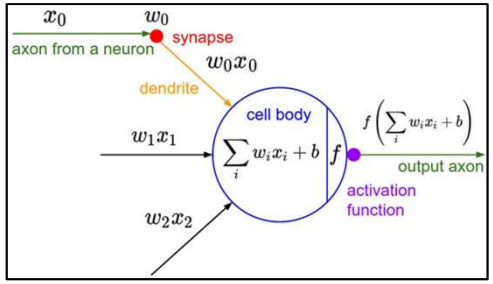
# Convolutional Layer

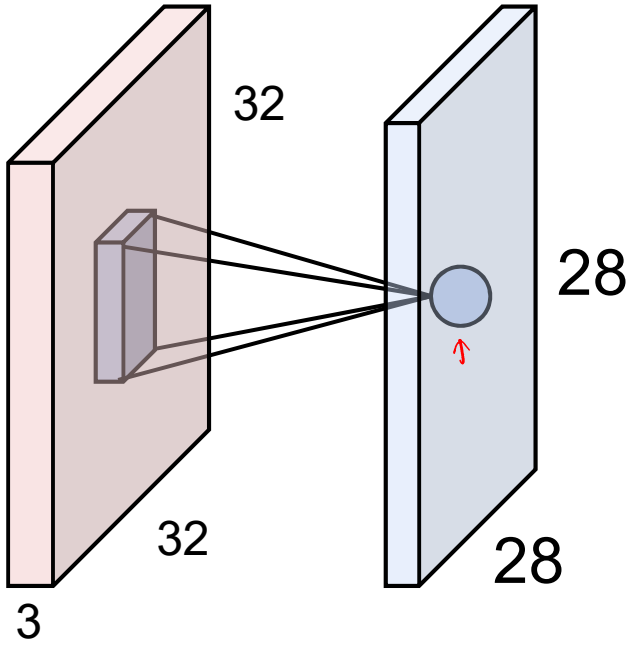
(btw, 1x1 convolution layers make perfect sense)



# Convolutional Layer

The brain/neuron view of CONV Layer



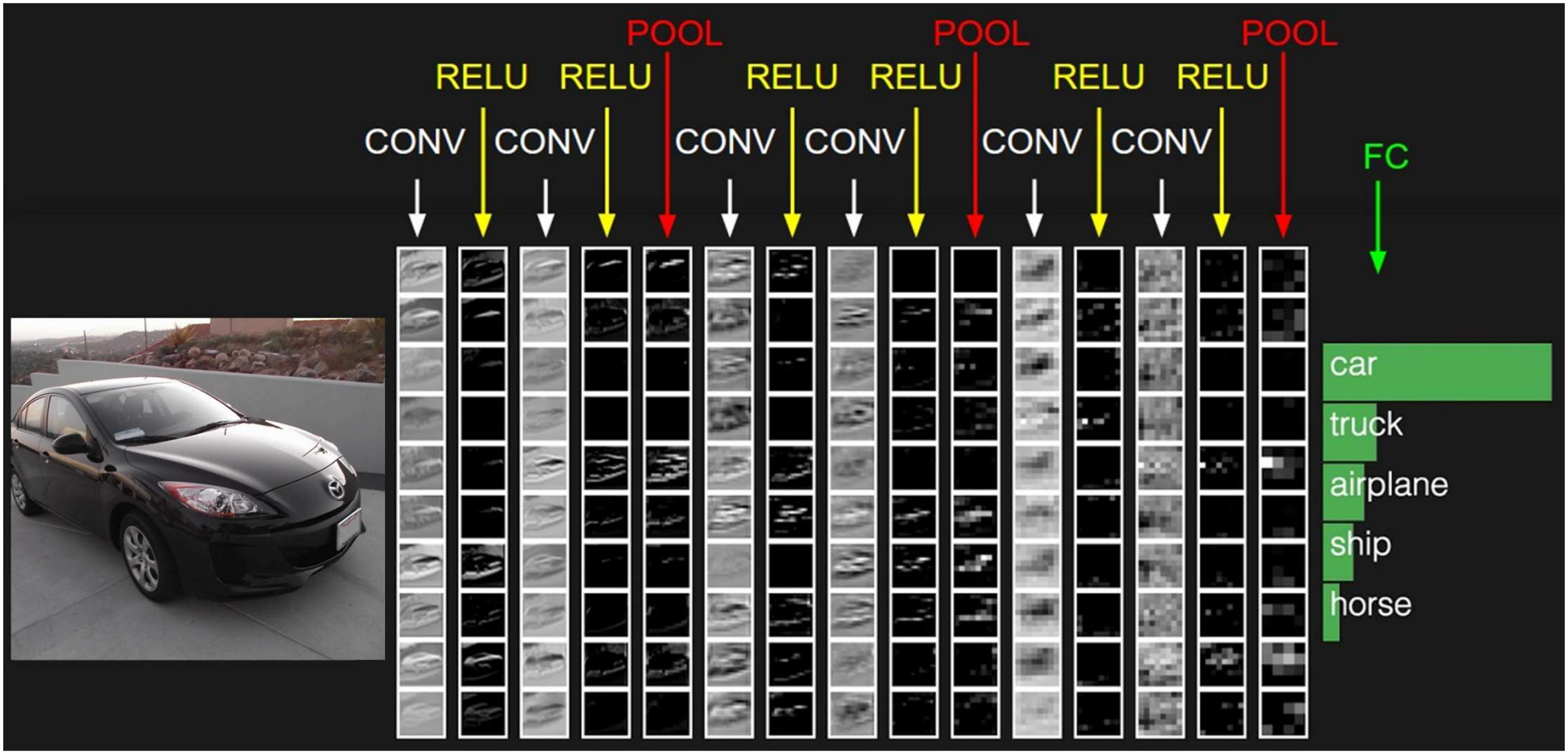




An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

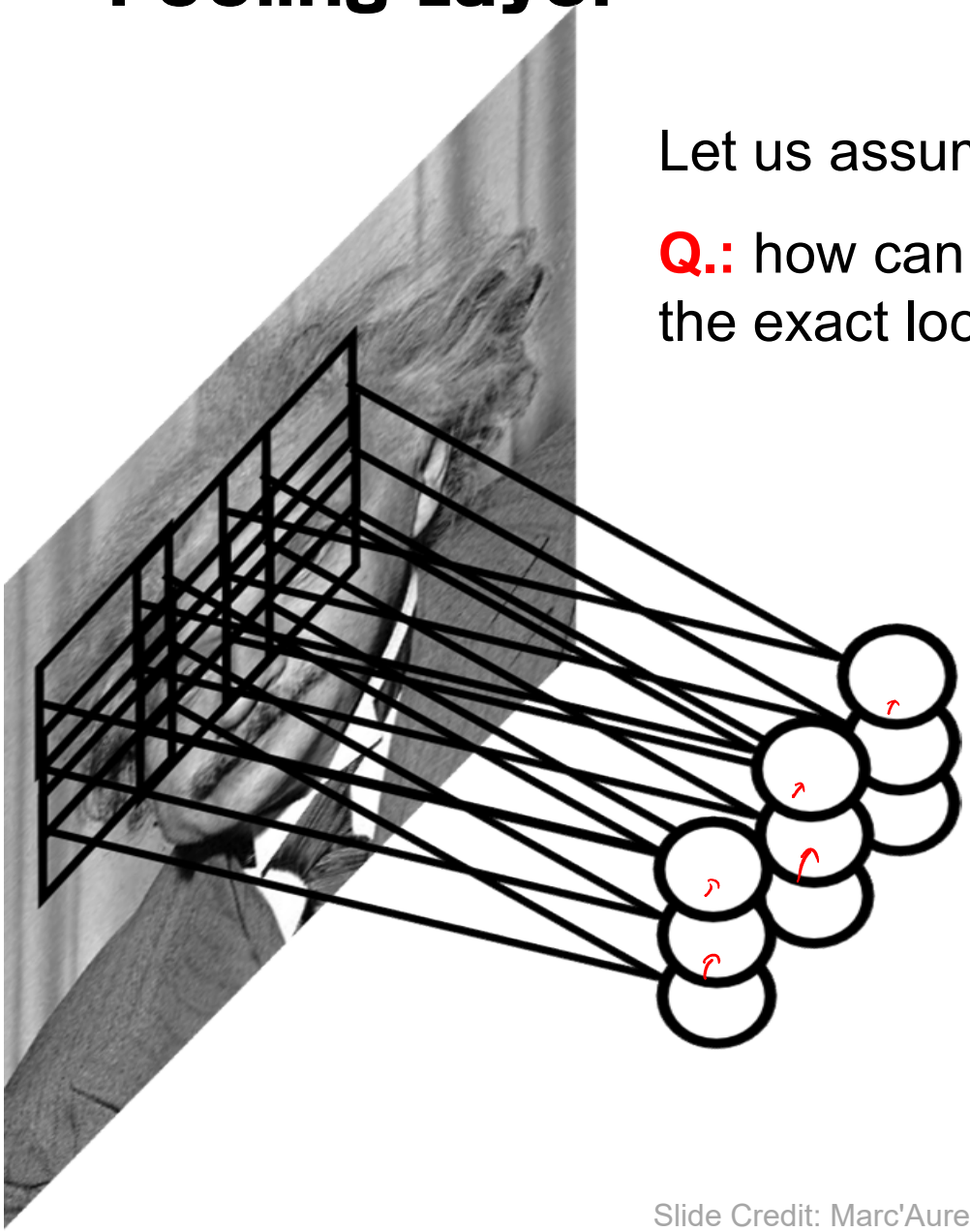
"5x5 filter" -> "5x5 receptive field for each neuron"

# Two more layers to go: POOL/FC




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

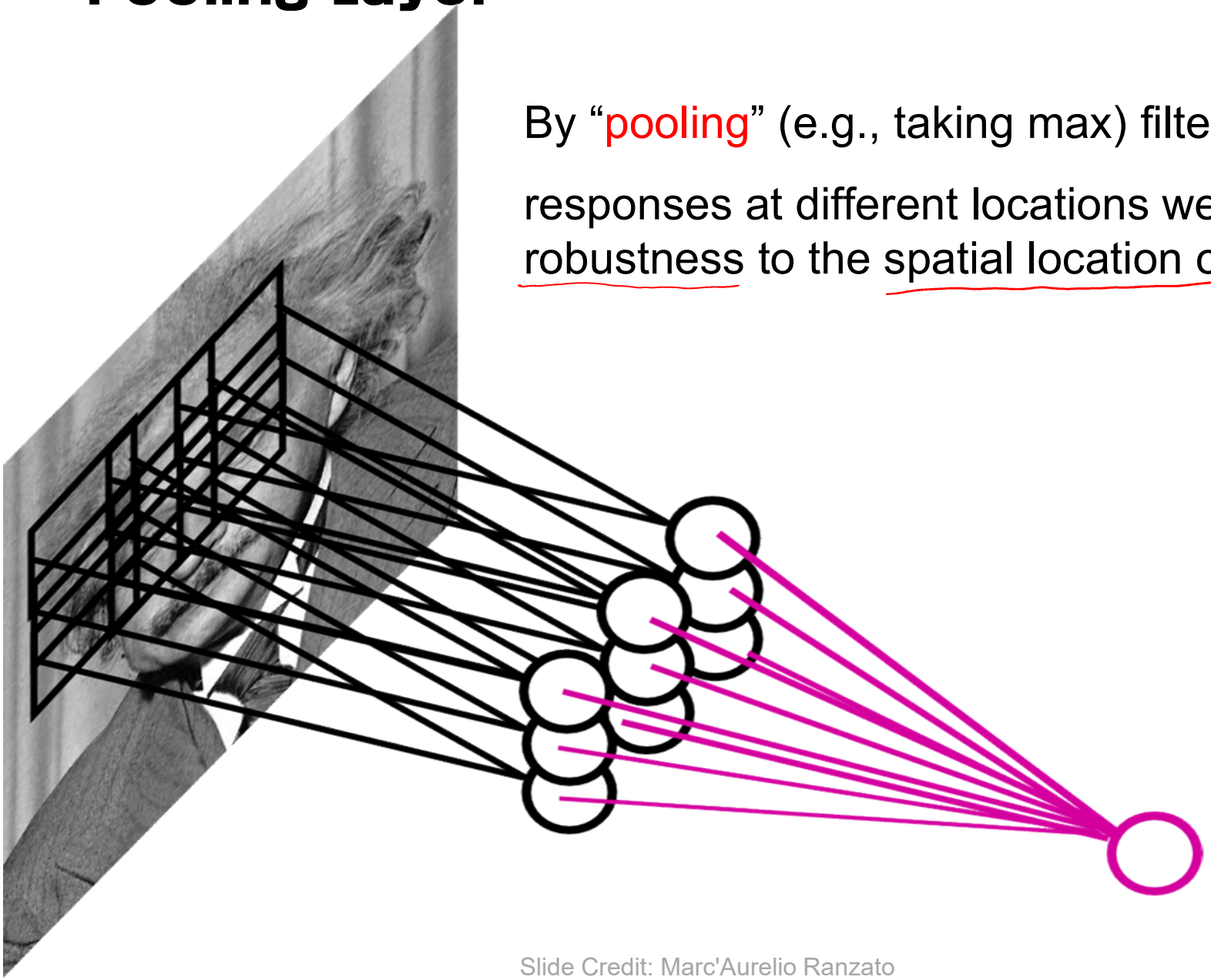
# Pooling Layer



Let us assume filter is an “eye” detector.

**Q.:** how can we make the detection robust to the exact location of the eye?

# Pooling Layer

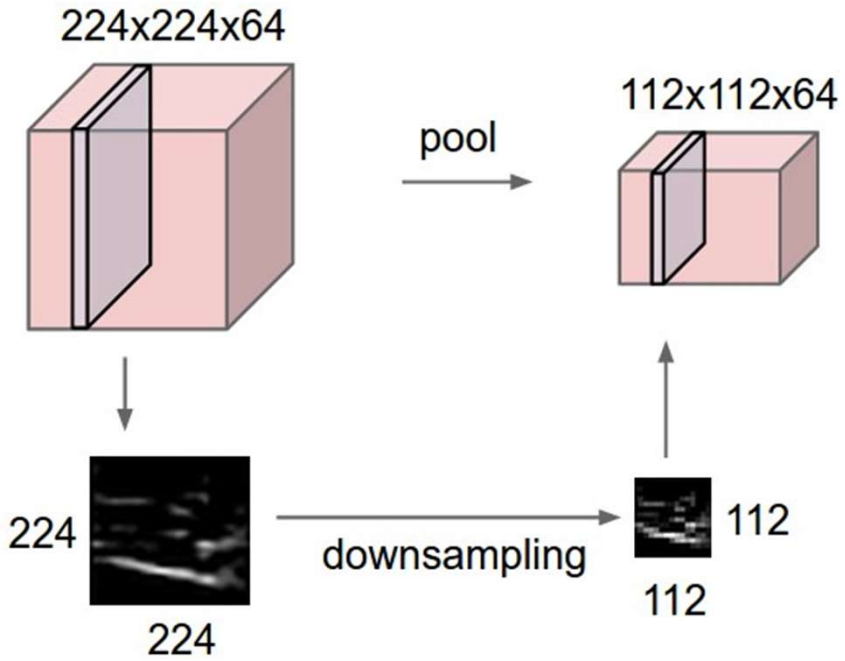


By “pooling” (e.g., taking max) filter

responses at different locations we gain robustness to the spatial location of features.

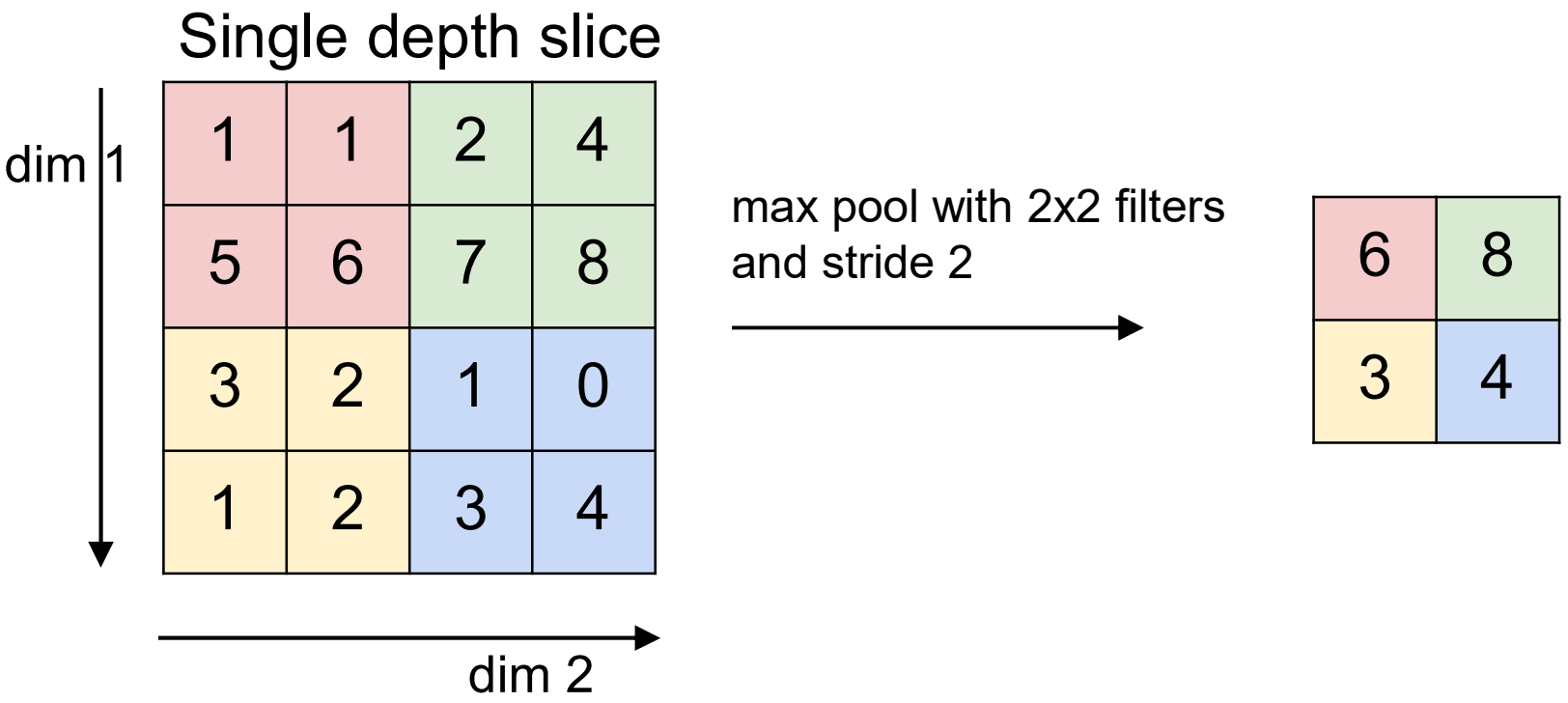
# Pooling layer

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




max pooling ←
avg ..
min ..

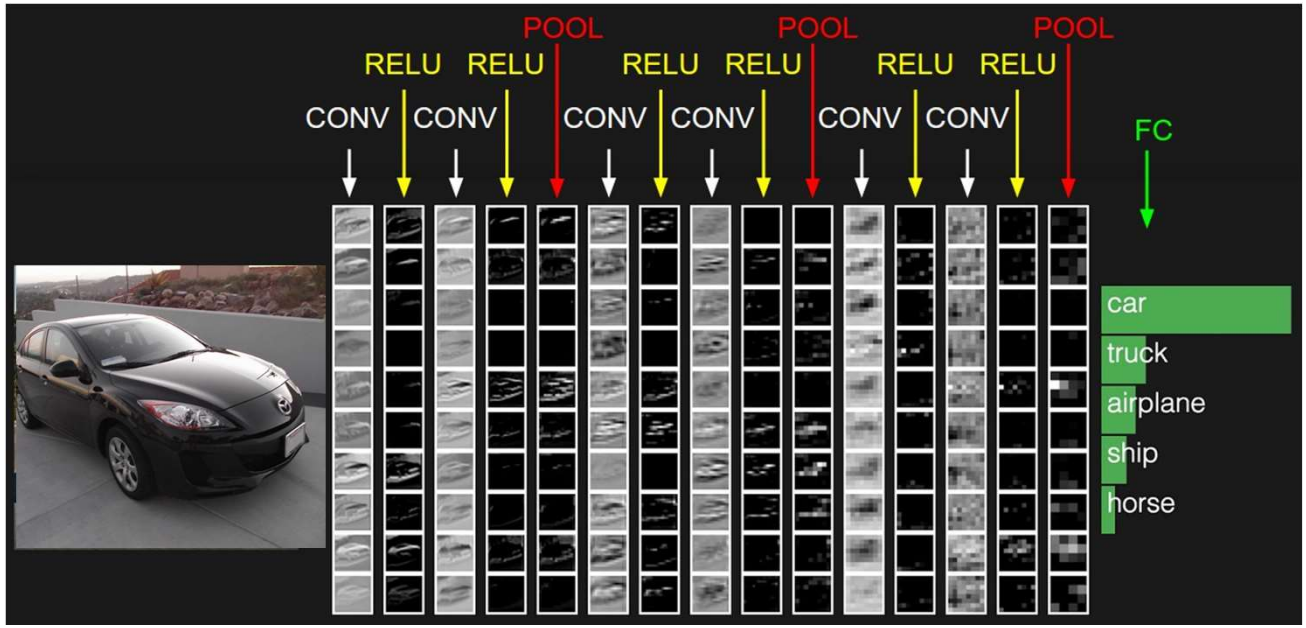
# MAX POOLING




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Fully Connected Layer (FC layer)

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

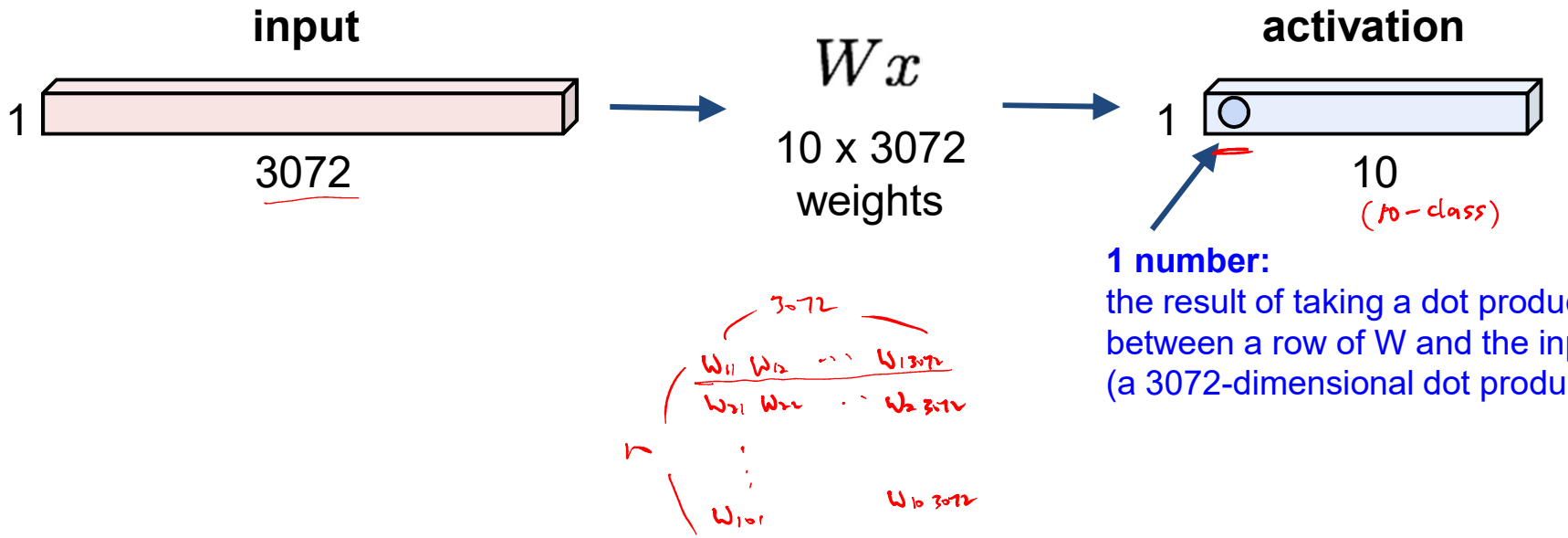



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Fully Connected Layer (FC layer)

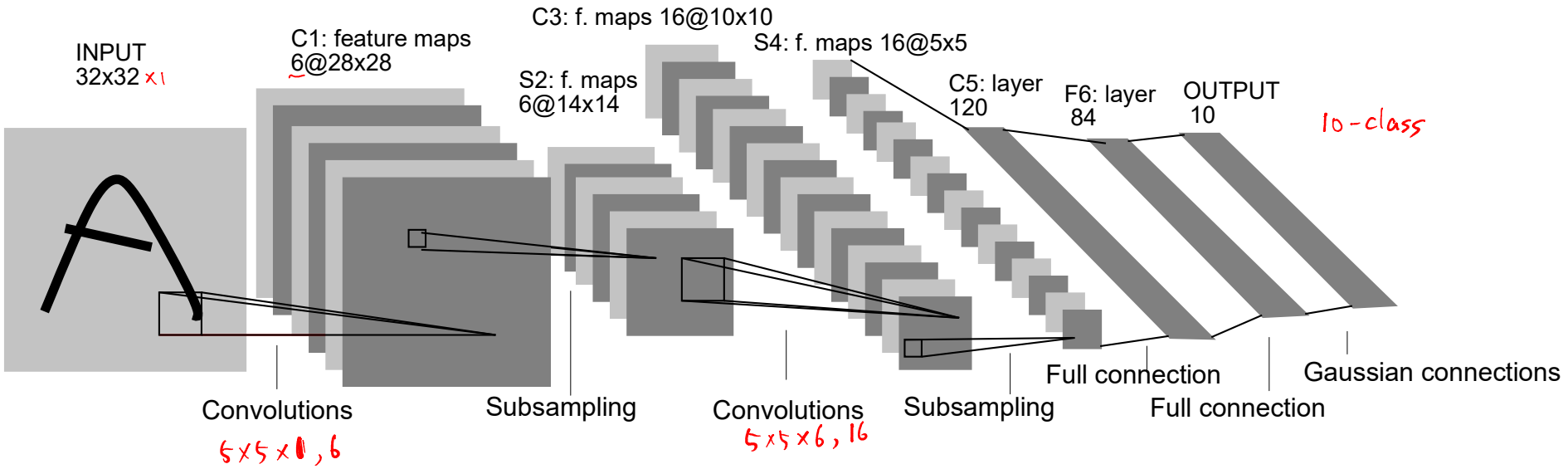
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



# Convolutional Nets

- Example:
  - http://yann.lecun.com/exdb/lenet/index.html



# Backpropagation Algorithm

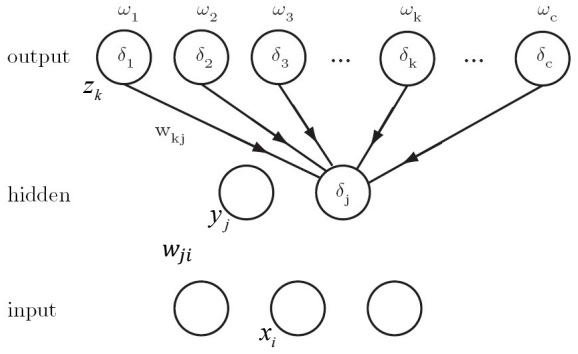
$$w(m+1) = w(m) + \Delta w(m) \qquad \Delta w = -\eta \frac{\partial J}{\partial w} \qquad J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$

$$net_k = w_k^t y \qquad net_j = w_j^t x \qquad z_k = f(net_k) \qquad y_j = f(net_j) \qquad \frac{\partial net_k}{\partial w_{kj}} = y_j$$

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = -(t_k - z_k)f'(net_k)y$$

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k)f'(net_k)$$

$$\Delta w_{kj} = -\eta\frac{\partial J}{\partial w_{kj}} = -\eta\frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \eta(t_k - z_k)f'(net_k)y_j = \eta\delta_k y_j$$




$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$
$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj}$$

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$
$$= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)\,w_{kj}f'(net_j)x_i$$

$$\delta_j = -\frac{\partial J}{\partial net_j} = -\frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}$$
$$= \sum_{k=1}^{c}(t_k - z_k)f'(net_k)\,w_{kj}f'(net_j) = f'(net_j)\sum_{k=1}^{c}w_{kj}\delta_k$$

$$\Delta w_{ji} = \eta\delta_j x_i = \eta\left[\sum_{k=1}^{c}w_{kj}\delta_k\right]f'(net_j)x_i$$

# Backpropagation in CNN

- Can be considered as a special case of MLP
  - Local connection – non connecting edges are disregarded
  - Weight sharing – kernels (filters) are updated
- Convolution based forward passing makes the backward passing also like convolution operations
- Pooling
  - E.g., max pooling – only the max element is updated

For details of backpropagation in CNN : https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c