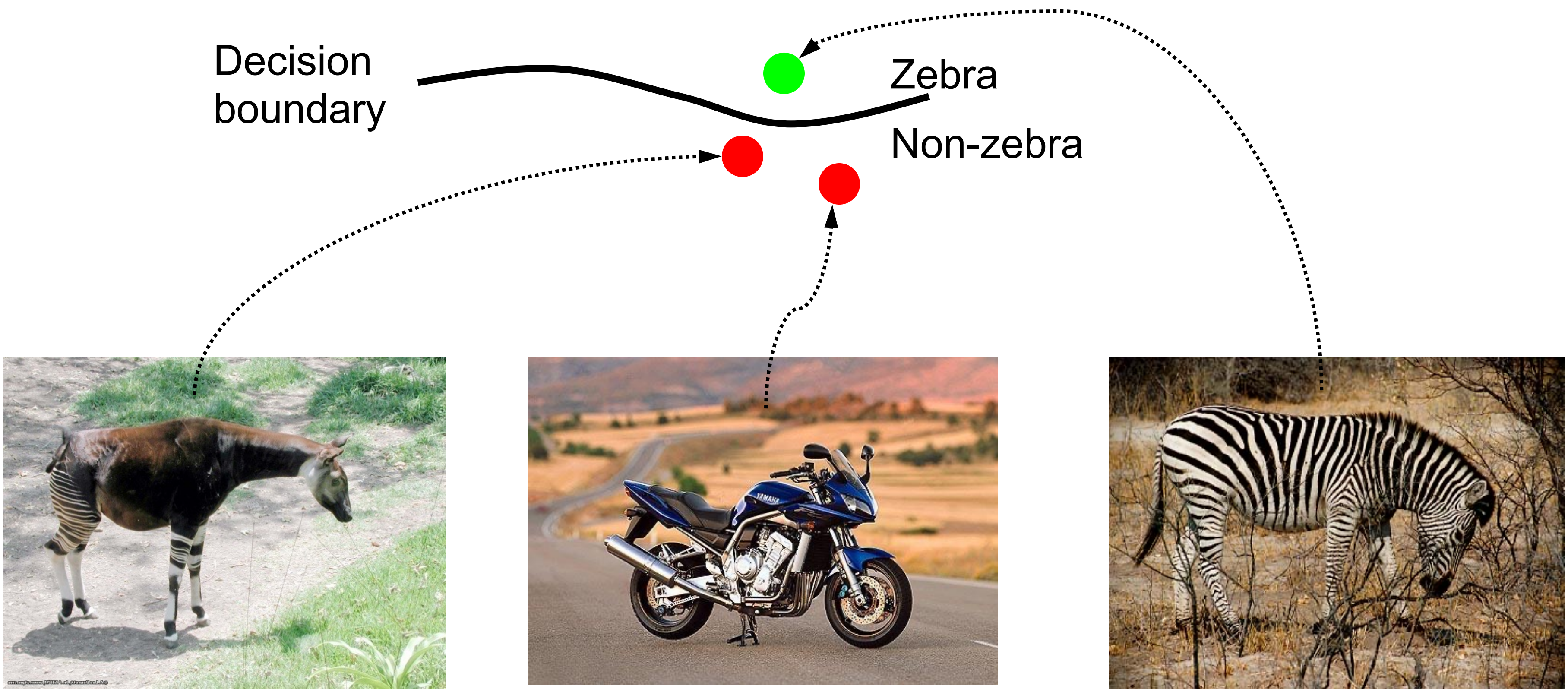
Convolutional Neural Network

Which is the dog?

Classification

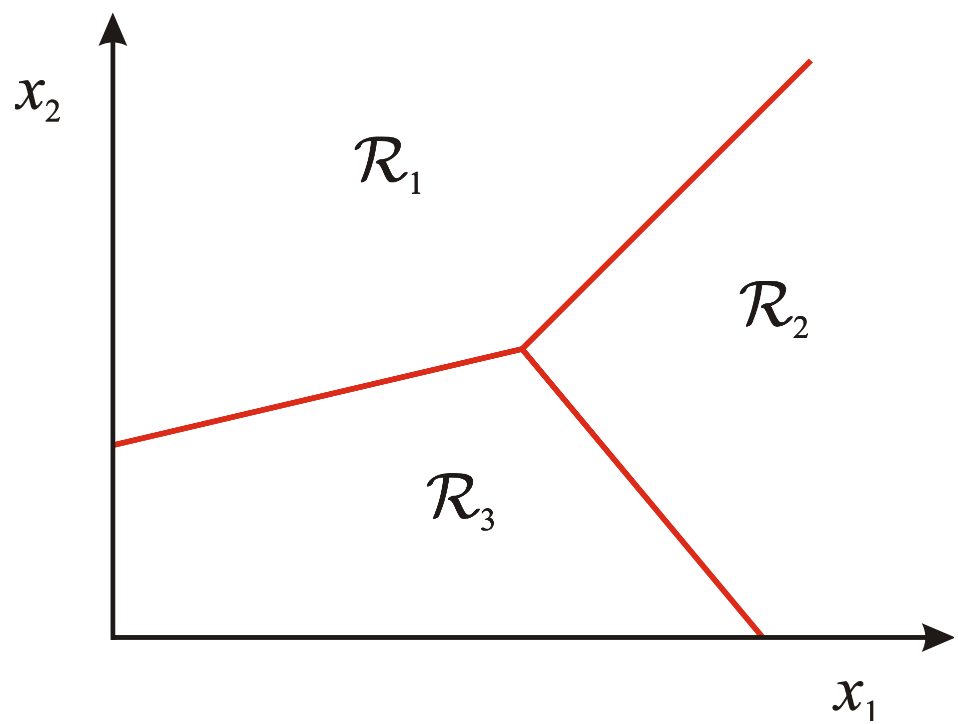
- Given a feature representation for images, how do we learn a model for distinguishing features from different classes?



Slide credit: L. Lazebnik

Classification

- Assign input vector to one of two or more classes
- Input space divided into *decision regions* separated by *decision boundaries*



Slide credit: L. Lazebnik

The machine learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = "apple"

f() = "tomato"

f() = "cow"

The machine learning framework

$$y = f(\mathbf{x})$$

output

prediction function

image / image feature

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

The old-school way

Training

Training Images

Image Features → Training → Learned model

Training Labels → Training

Testing

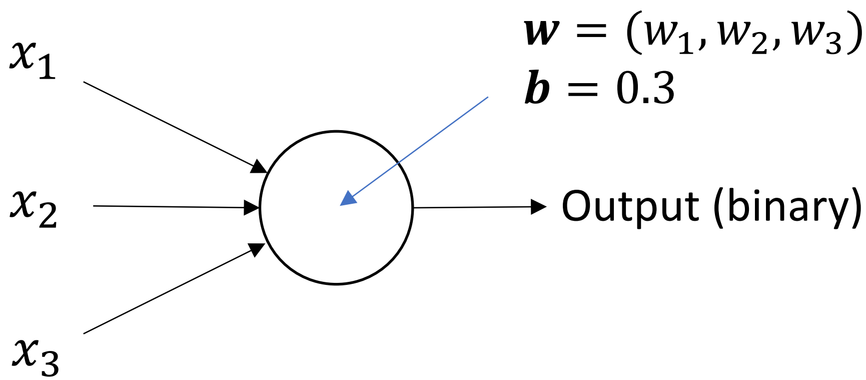
Test Image

Image Features → Learned model → Prediction

Slide credit: D. Hoiem and L. Lazebnik

Neural Networks

- Basic building block for composition is a *perceptron* (Rosenblatt c.1960)
- Linear classifier – vector of weights *w* and a 'bias' *b*



activation functions

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j,$$

Binary classifying an image

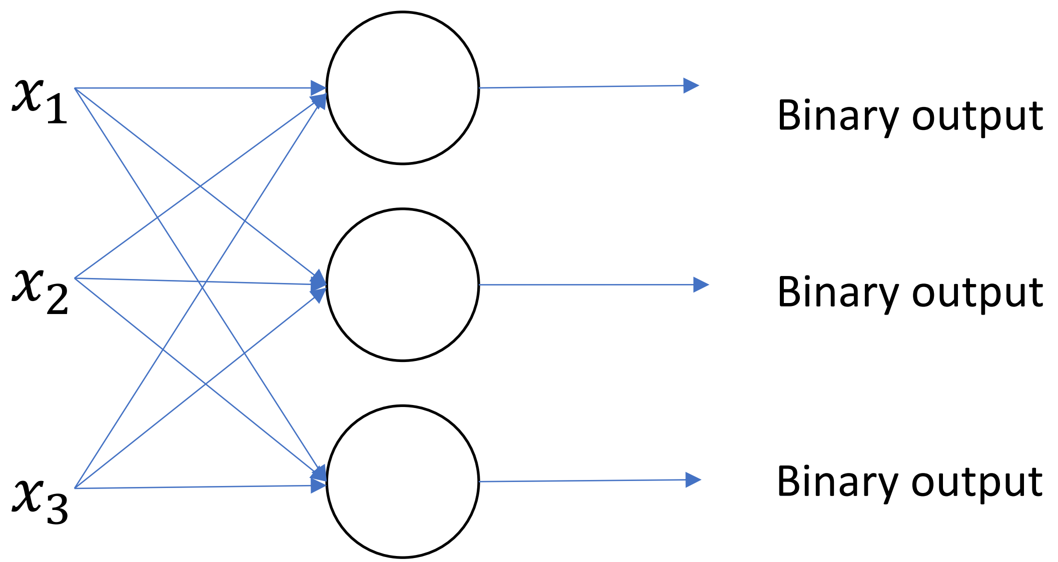
- Each pixel of the image would be an input.
- So, for a 28 x 28 image, we vectorize.
- **x** = 1 x 784

- **w** is a vector of weights for each pixel, 784 x 1
- b is a scalar bias per perceptron

- result = **xw** + b   -> (1x784) x (784x1) + b = (1x1)+b

Neural Networks - multiclass

- Add more perceptrons



Multi-class classifying an image

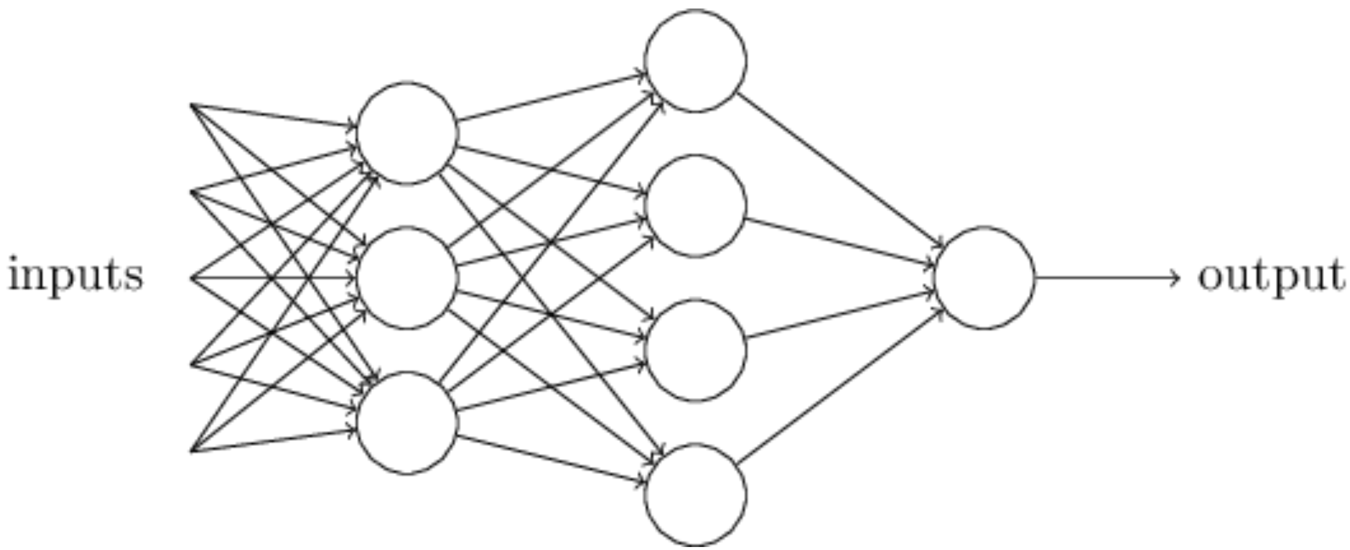
- Each pixel of the image would be an input.
- So, for a 28 x 28 image, we vectorize.
- **x** = 1 x 784

- **W** is a matrix of weights for each pixel/each perceptron
  - **W** = 784 x 10 (10-class classification)
- **b** is a bias *per perceptron* (vector of biases); (1 x 10)

- result = **xW** + **b** -> (1x784) x (784 x 10) + **b**

  -> (1 x 10) + (1 x 10) = output vector

Composition

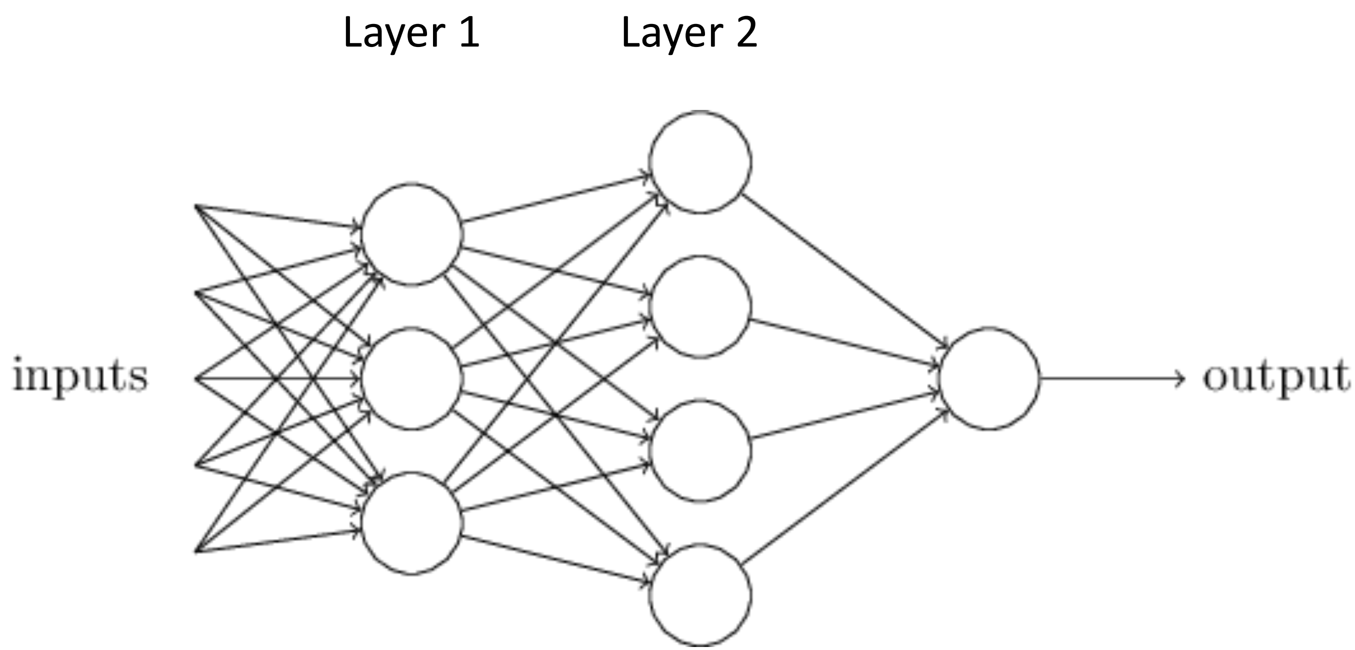


Attempt to represent complex functions as compositions of smaller functions.

Outputs from one perceptron are fed into inputs of another perceptron.

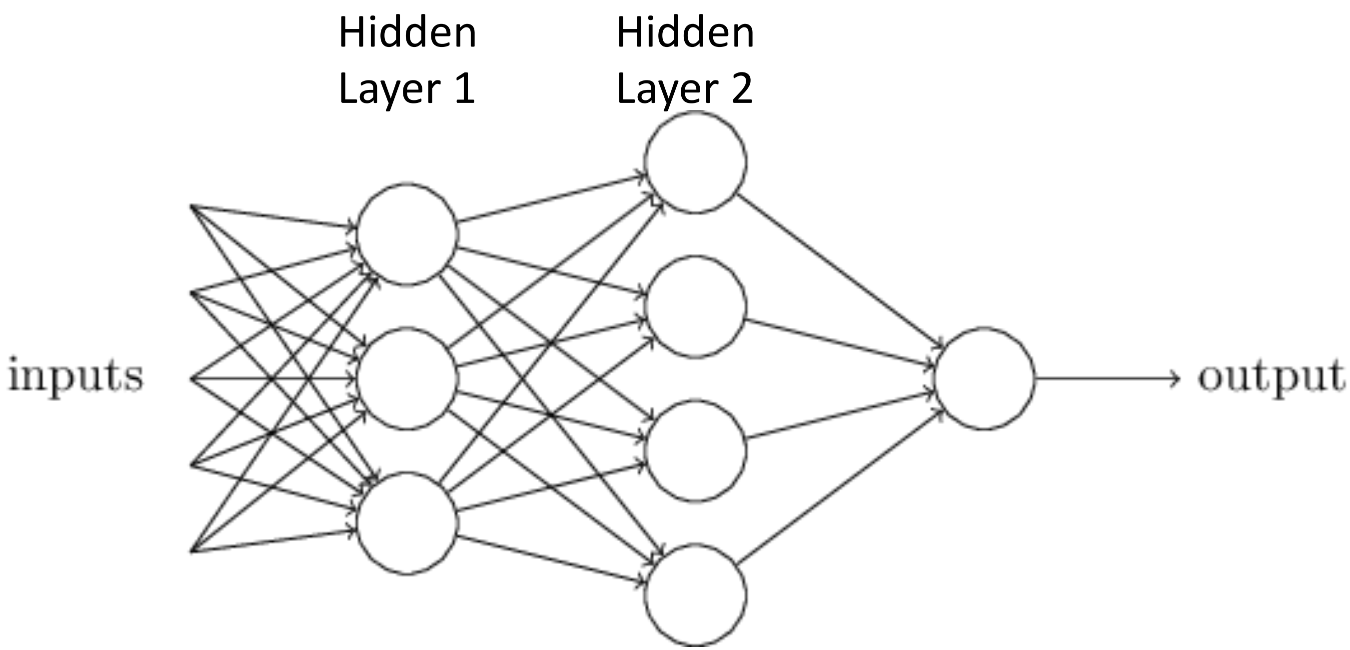
Nielsen

Composition



Sets of layers and the connections (weights) between them define the *network architecture.*

Composition



Layers that are in between the input and the output are called *hidden layers*, because we are going to *learn* their weights via an optimization process.

Nielsen

Linear functions

- We have formed chains of linear functions.
- We know that linear functions can be combined
  - g = f(h(x))

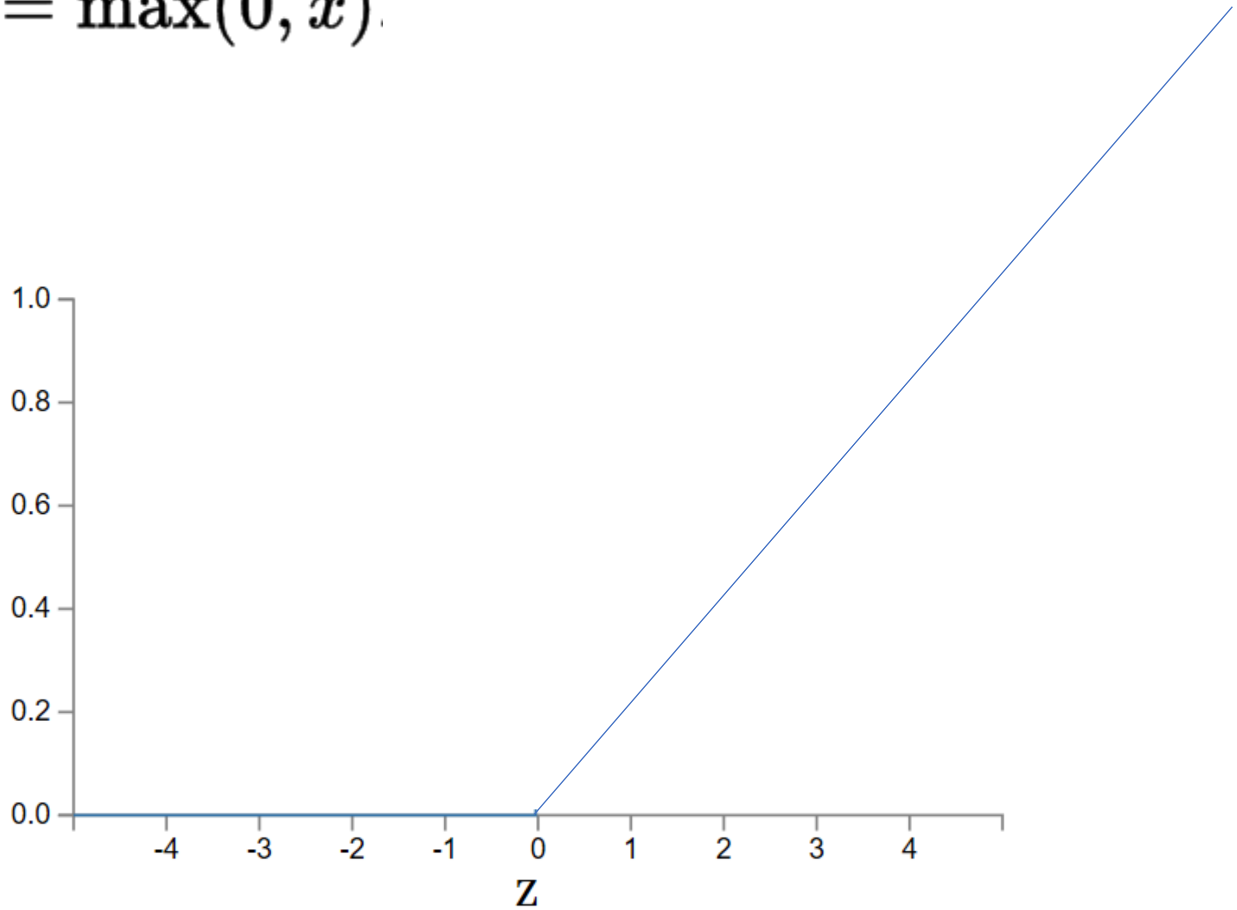
Our composition of functions is really
just a single function

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

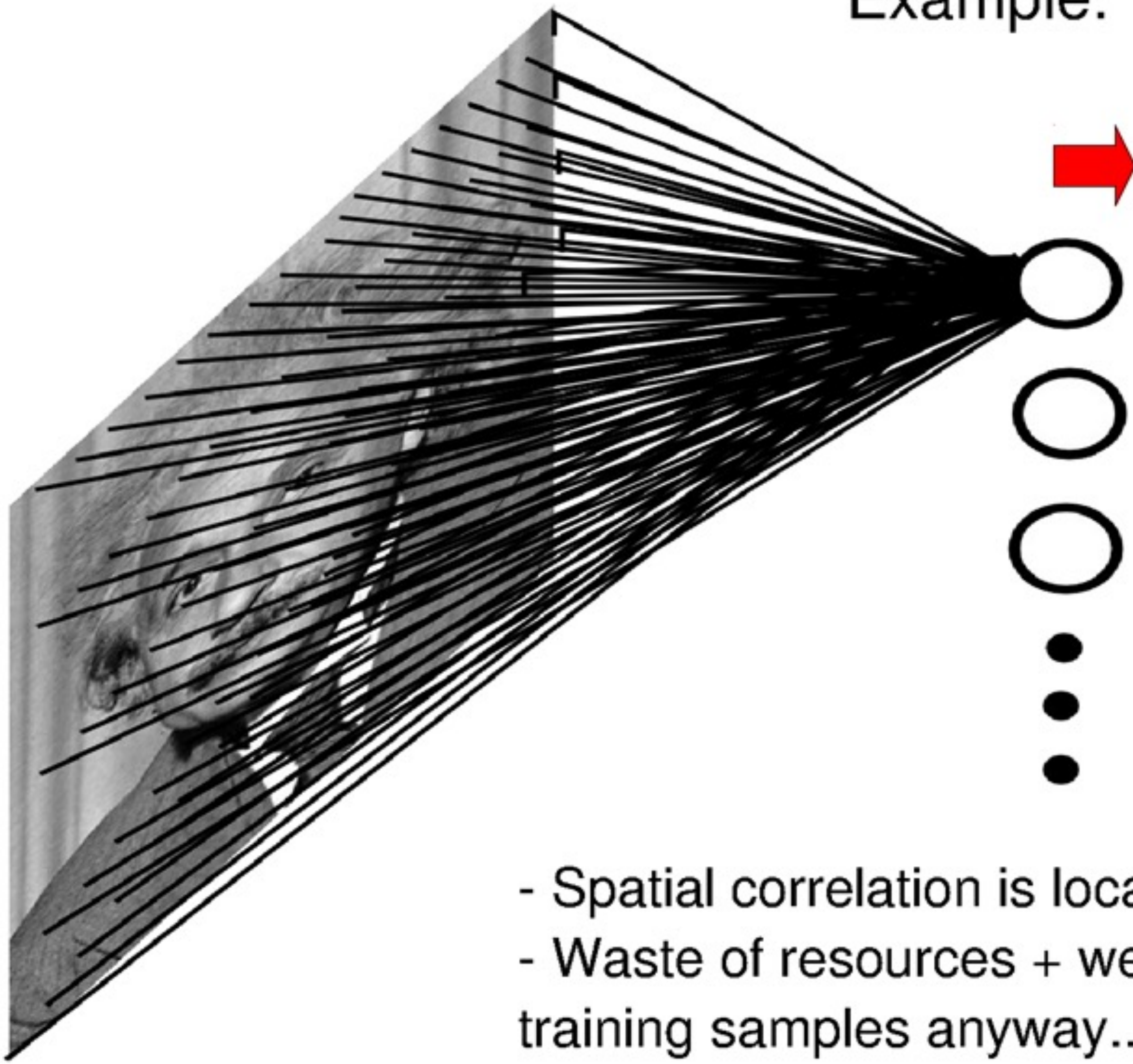
nonlinear

Rectified Linear Unit

- ReLU $f(x) = \max(0, x)$



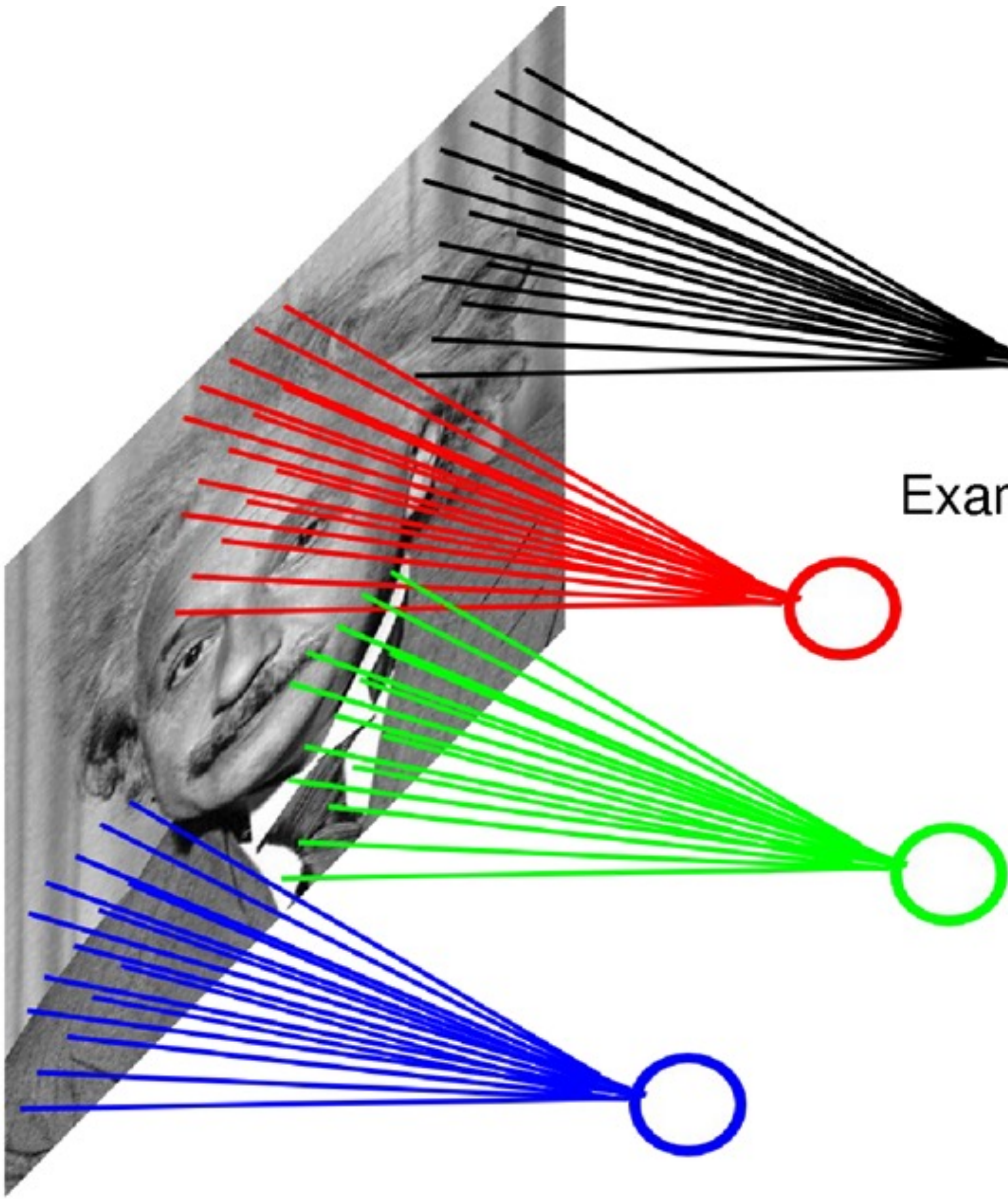
Images as input to neural networks



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

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

Ranzato



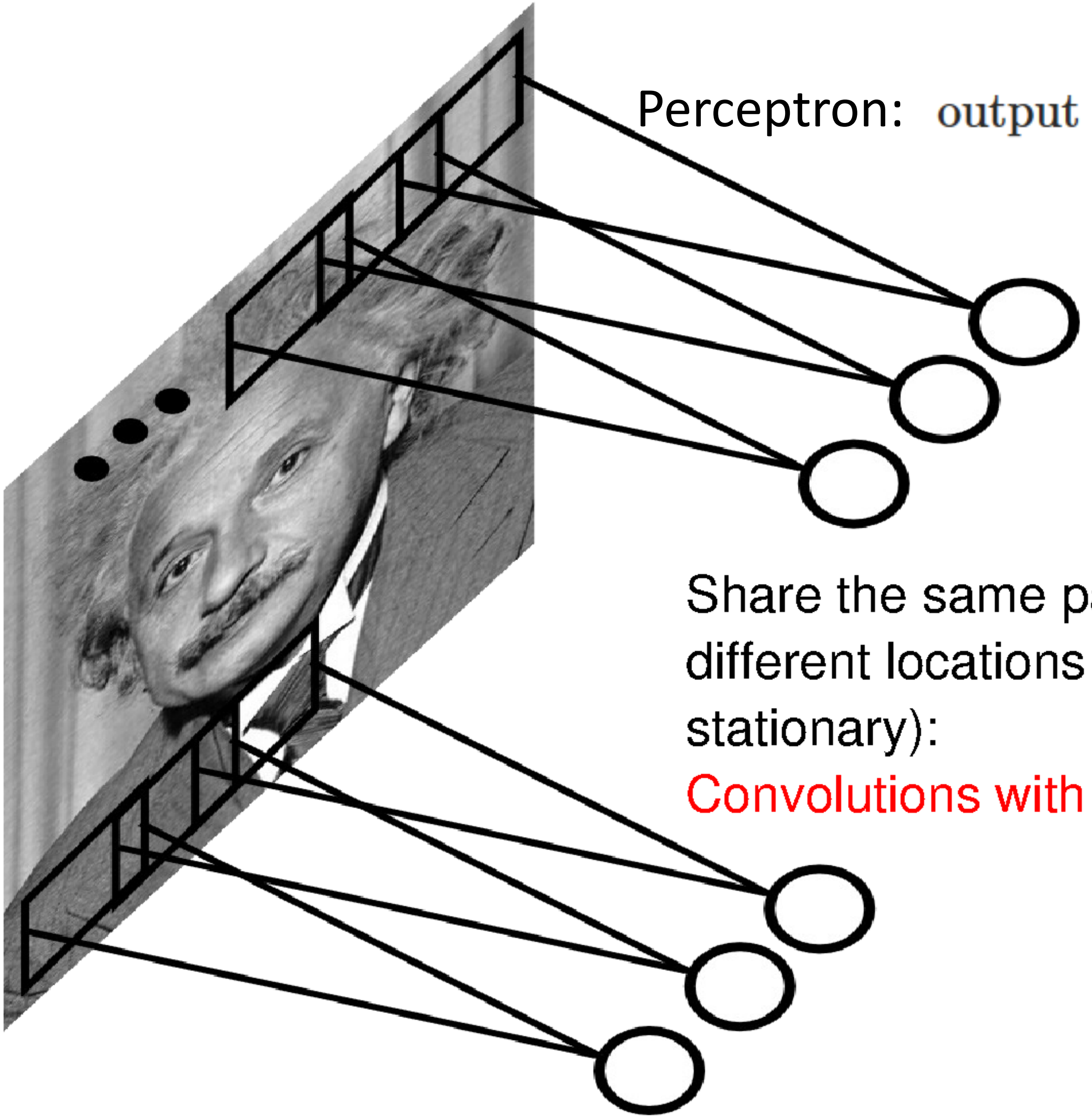
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Ranzato

Convolutional Layer

Perceptron: $\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$

$w \cdot x \equiv \sum_j w_j x_j,$

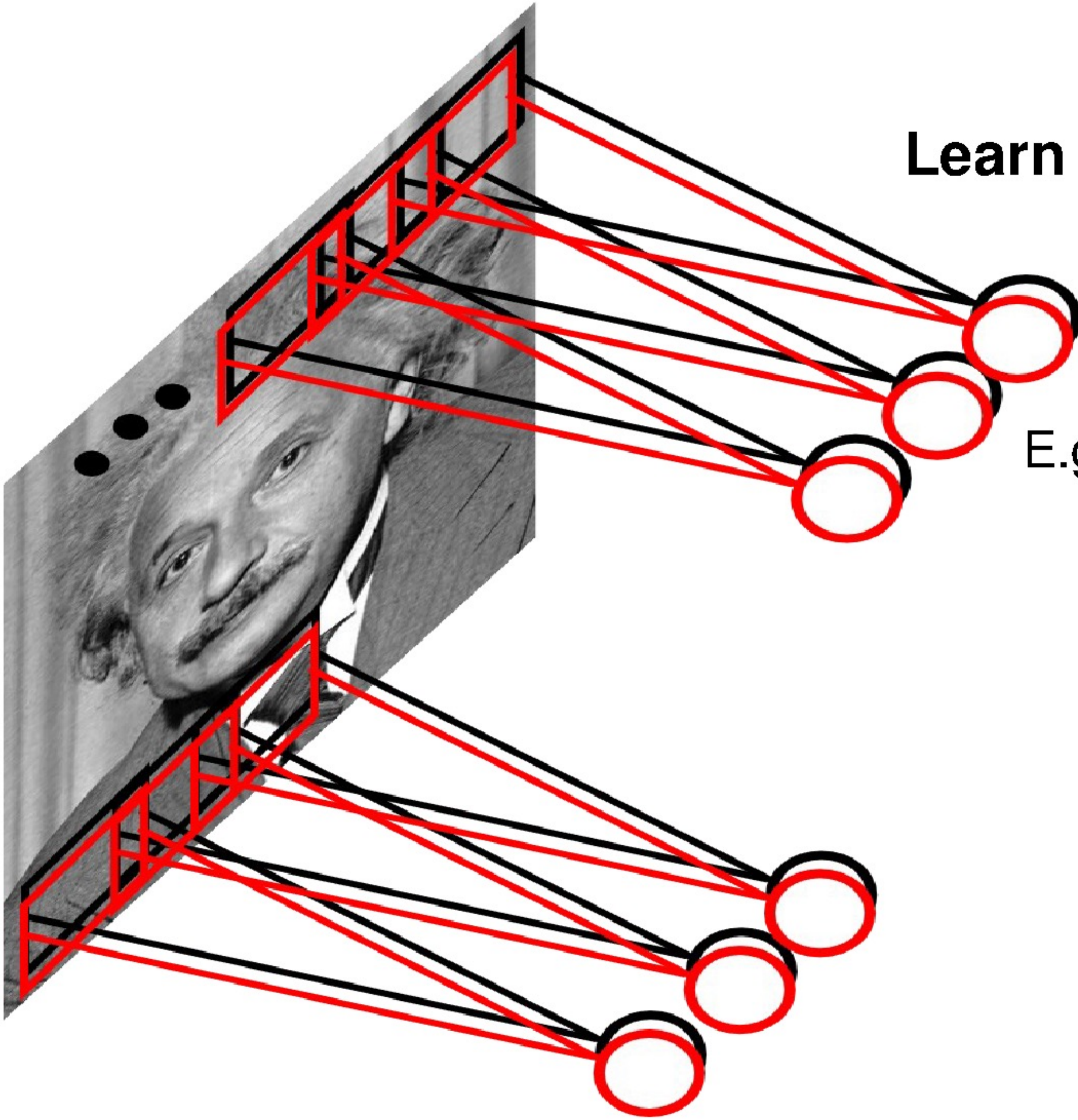


Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

Ranzato

Convolutional Layer



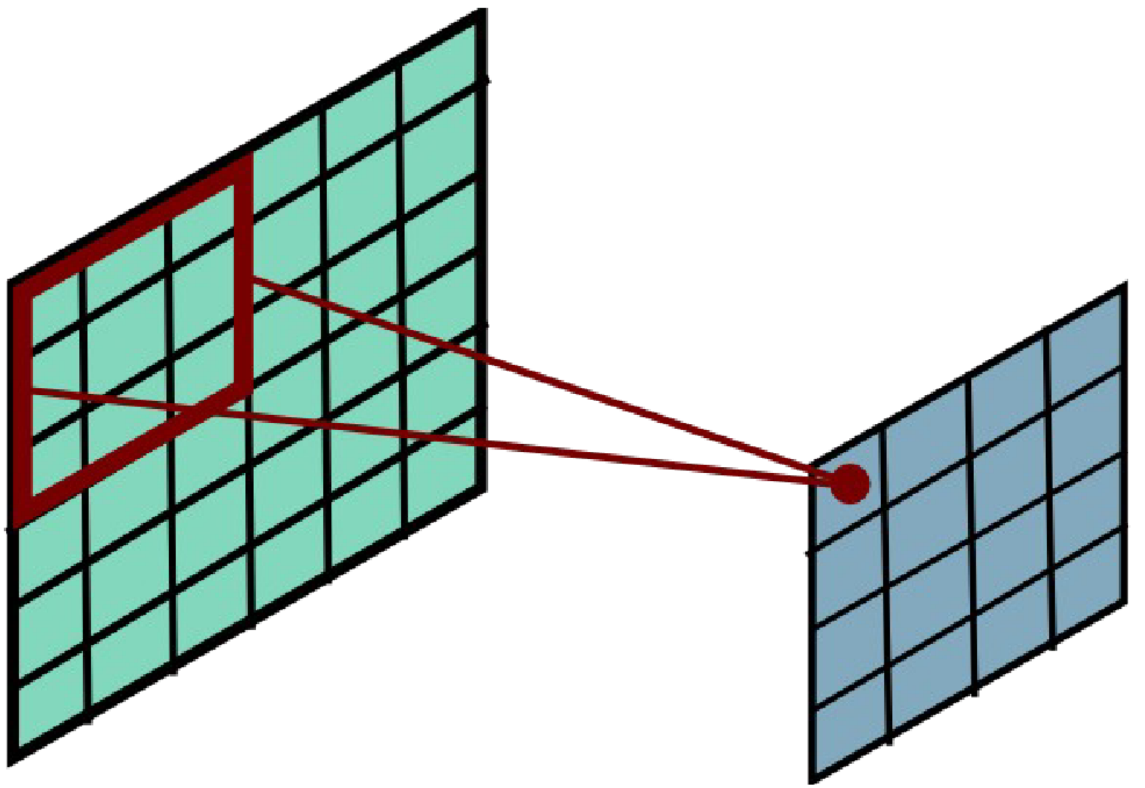
Learn multiple filters.

Filter = 'local' perceptron.
Also called *kernel.*

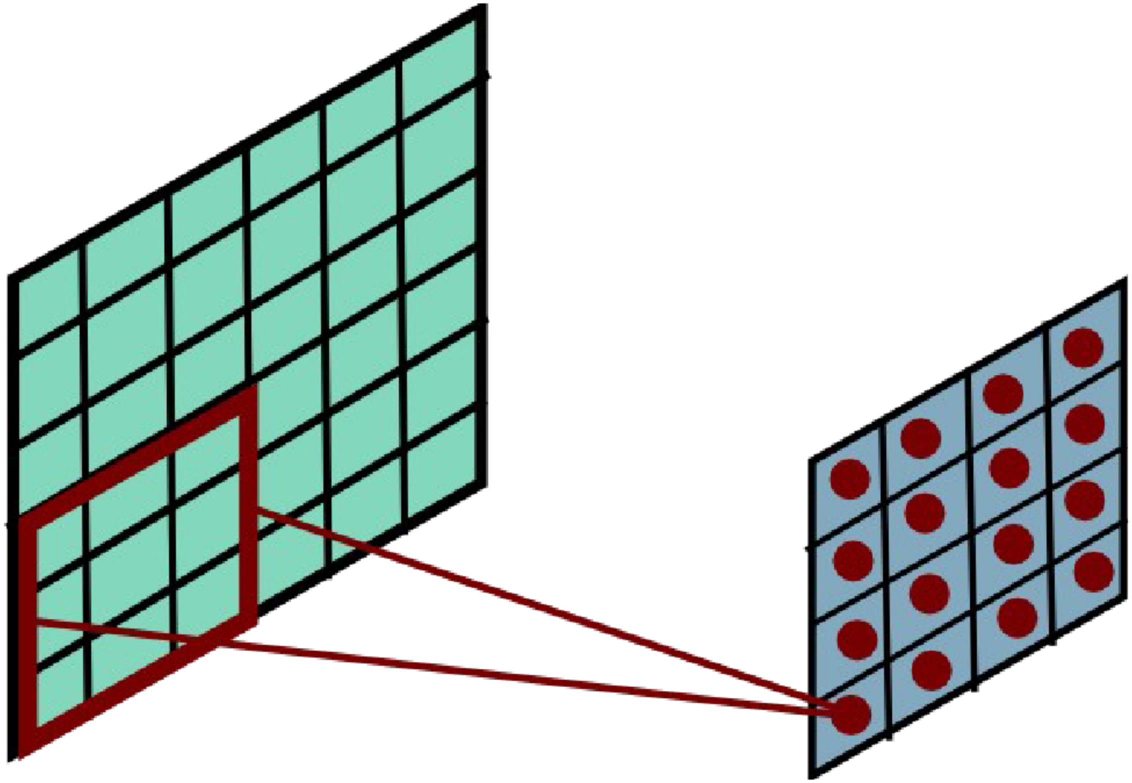
E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

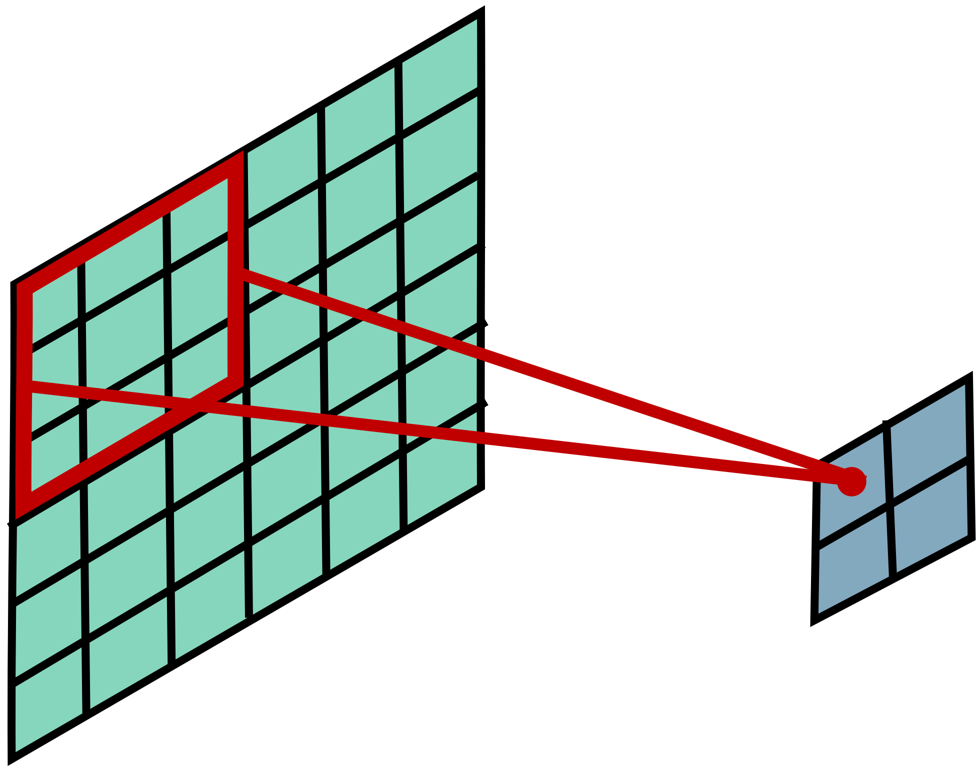
Stride = 1



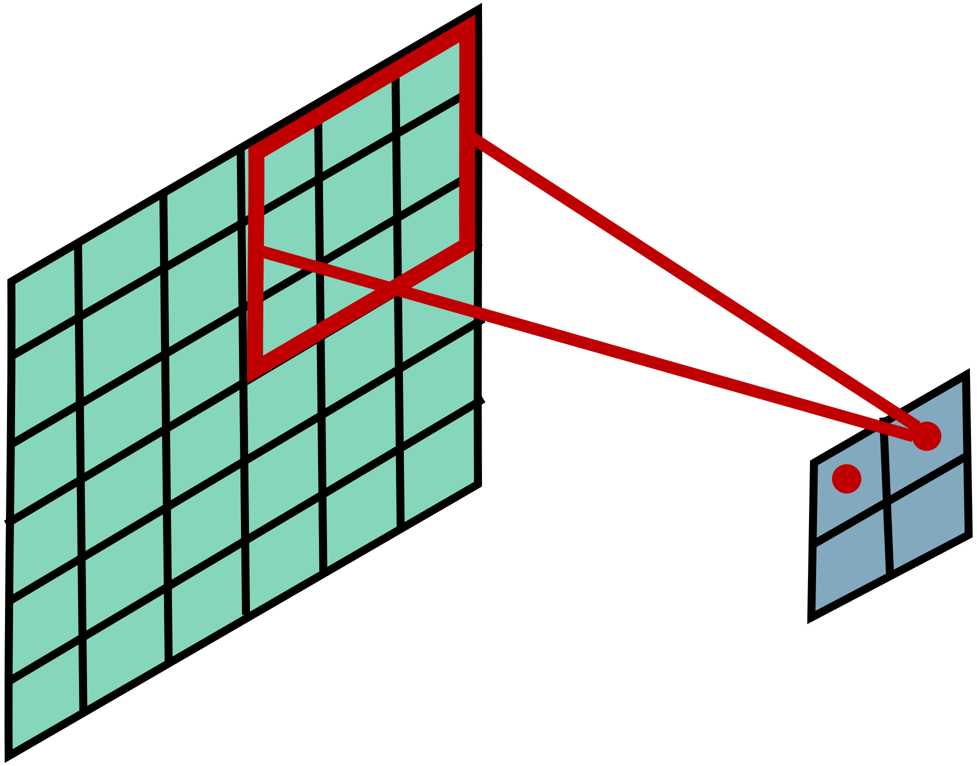
Stride = 1



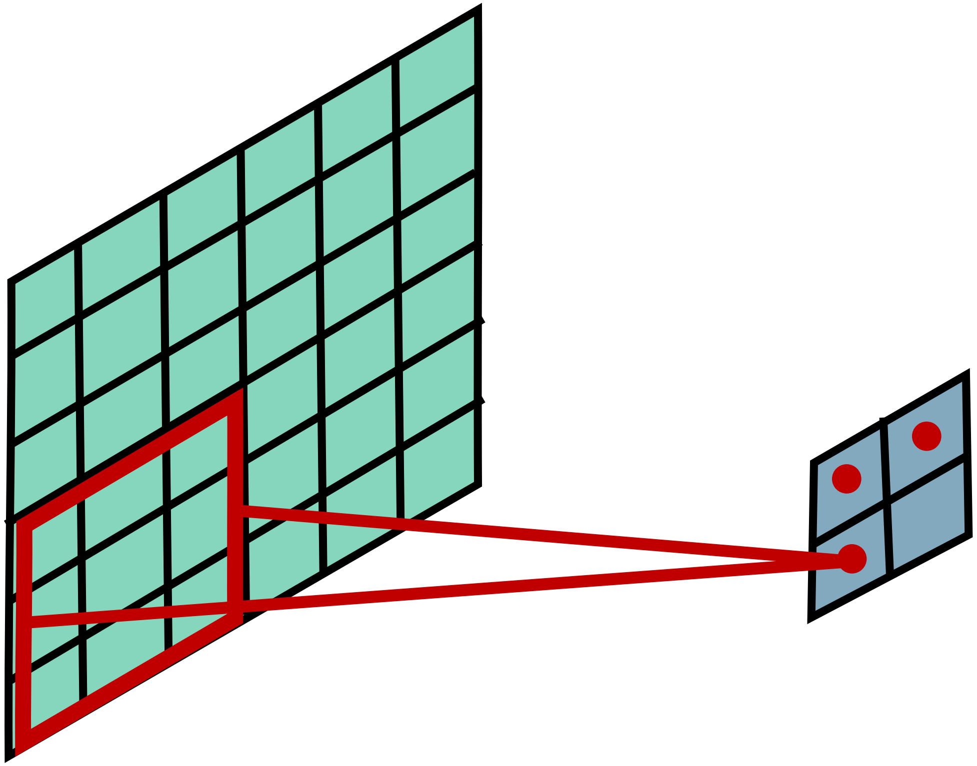
Stride = 3



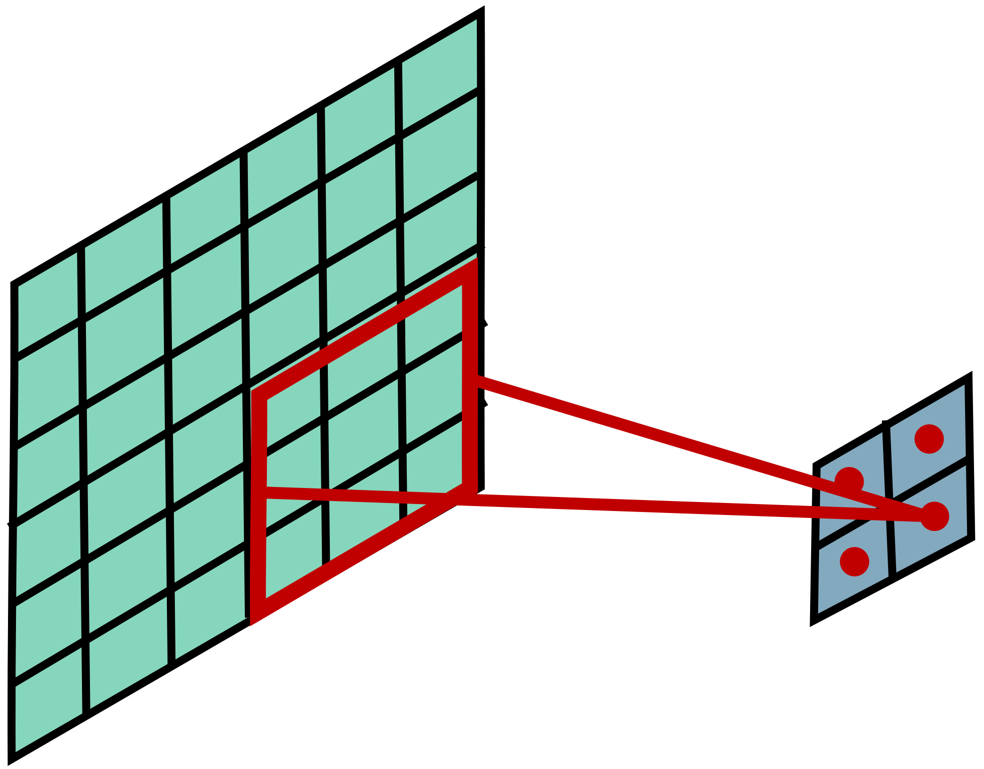
Stride = 3



Stride = 3

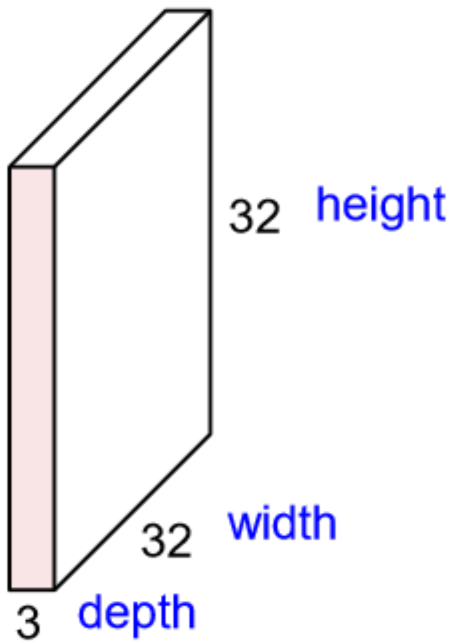


Stride = 3

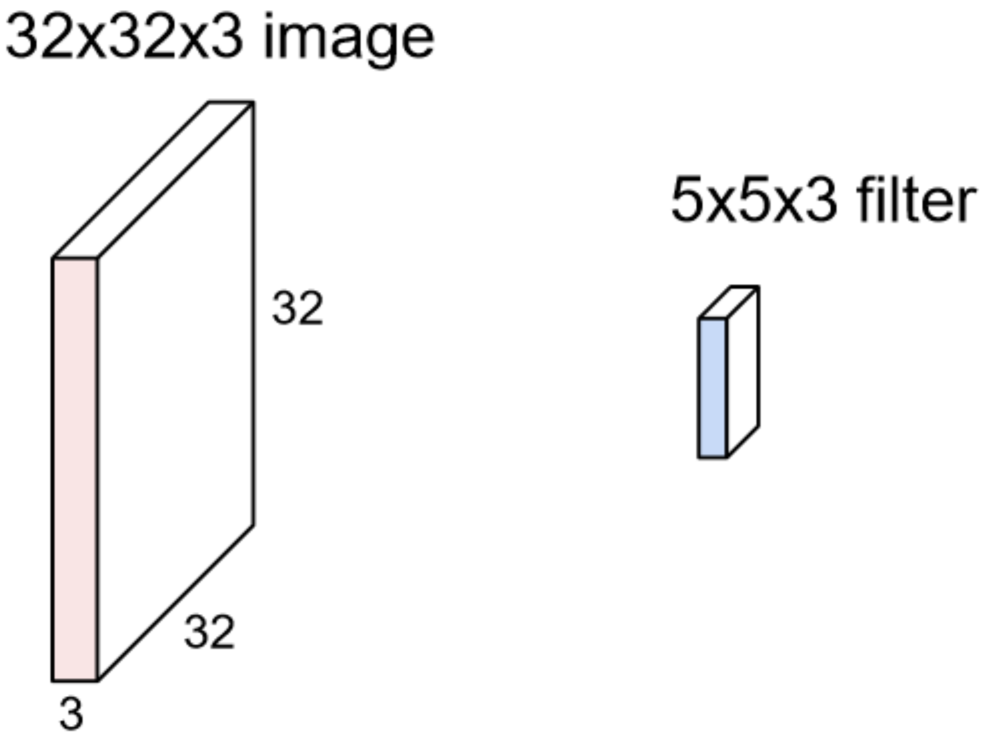


Convolutions: More detail

32x32x3 image



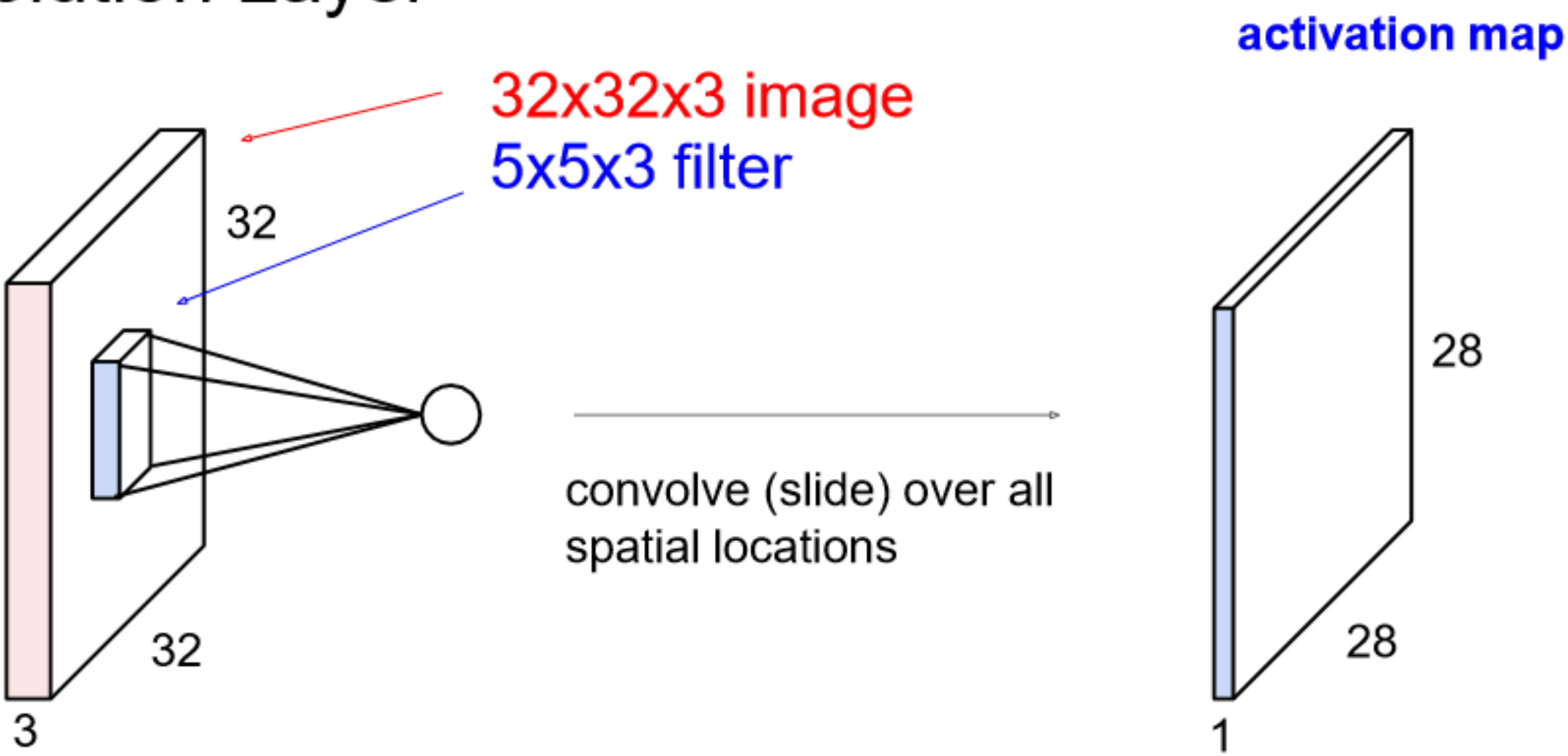
Convolutions: More detail



Andrej Karpathy

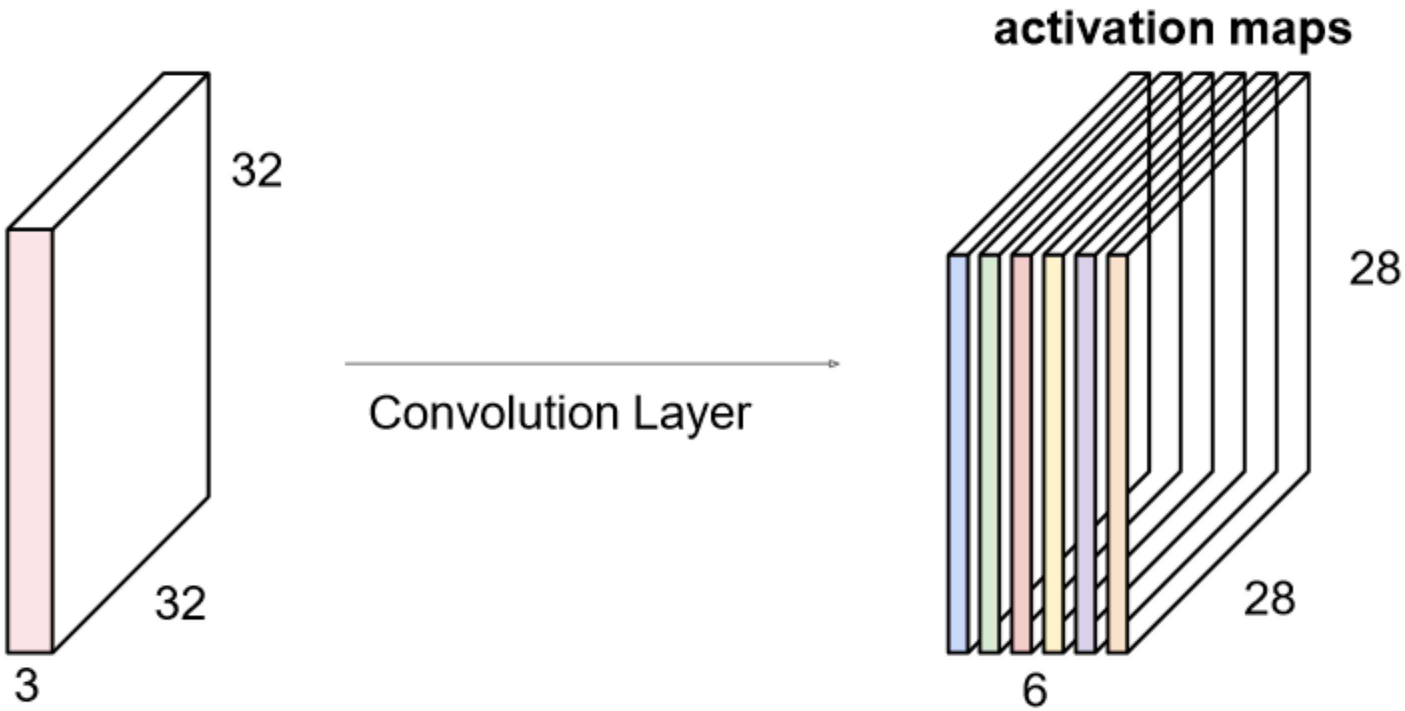
Convolutions: More detail

Convolution Layer



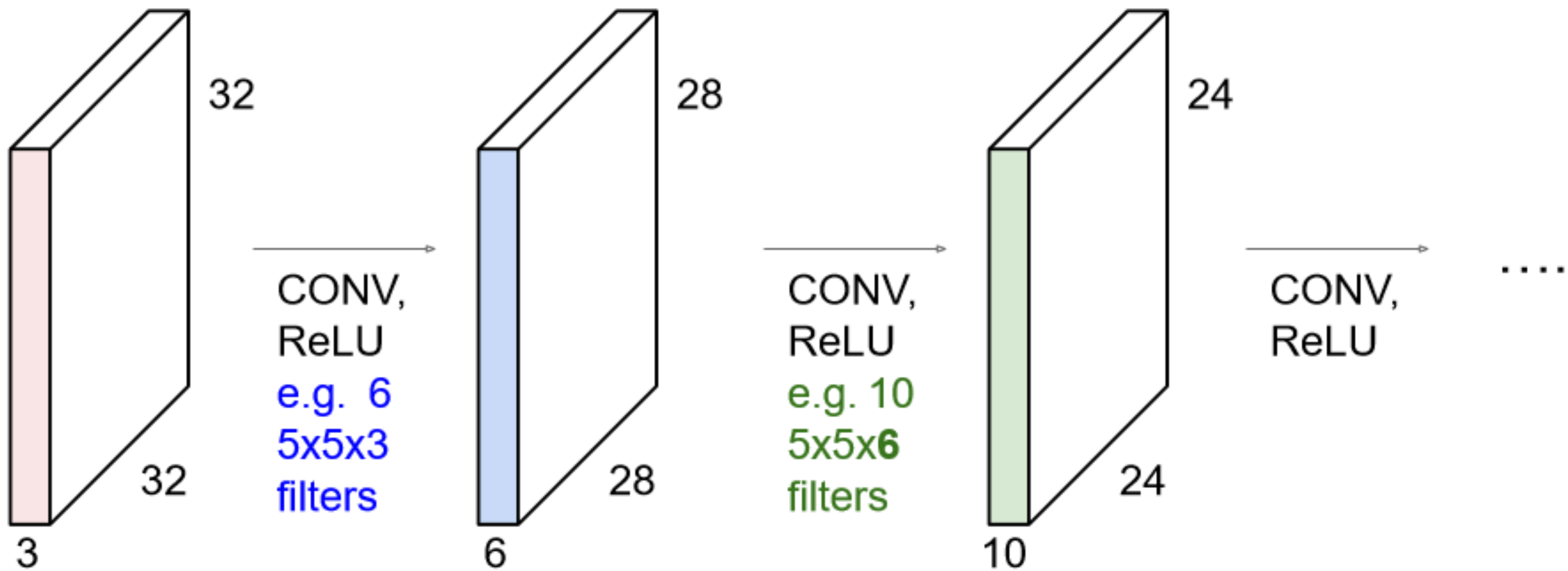
Convolutions: More detail

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

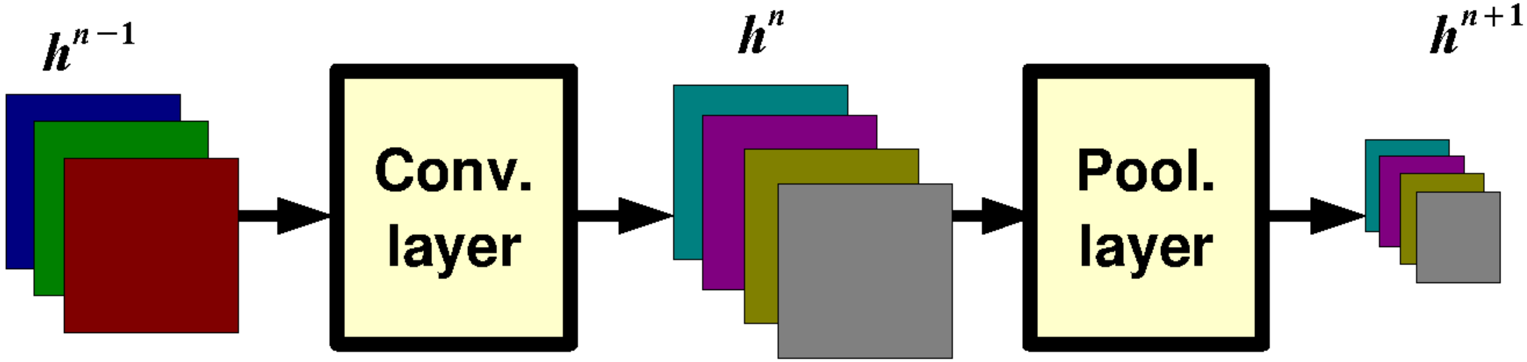


We stack these up to get a "new image" of size 28x28x6!

Convolutions: More detail



Pooling Layer: Receptive Field Size



Pooling Layer: Examples

Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Max Pooling

Filter 1 Filter 2