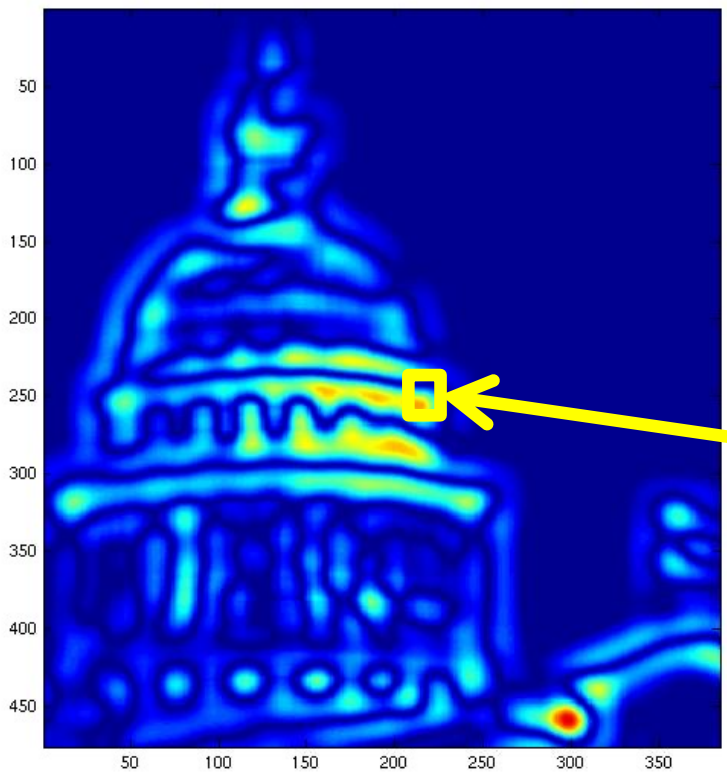
# Feature Learning with Neural Networks

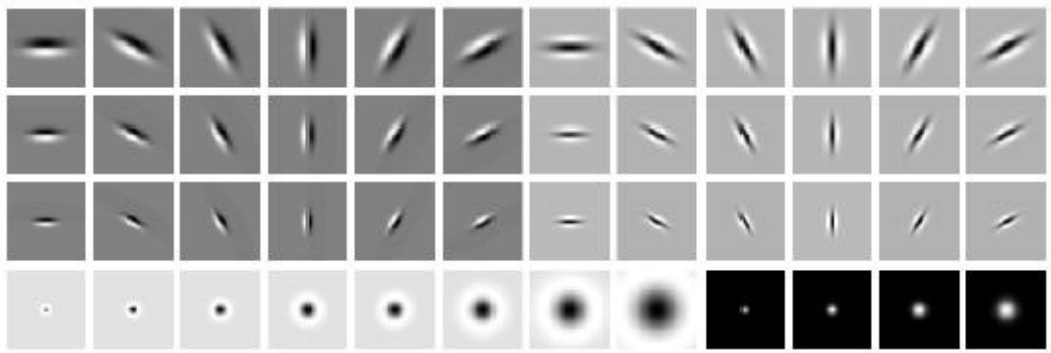
(aka Deep Learning)

CS194: Intro to Computer Vision and Comp. Photo

Angjoo Kanazawa, UC Berkeley, Fall 2021

# Recap: Filter Bank Response



[r1, r2, …, r38]

We can form a feature vector from the list of responses at each pixel.

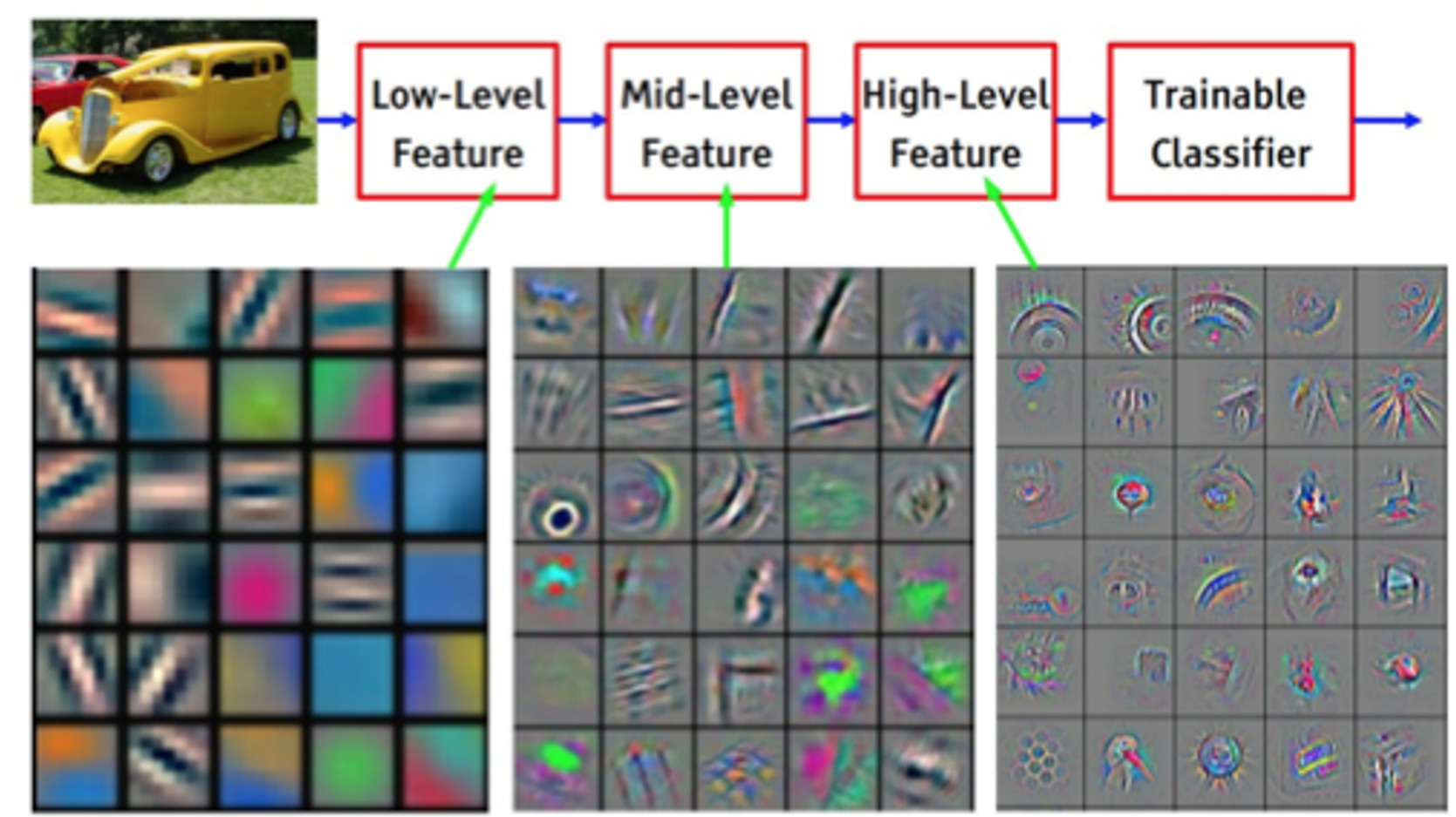
# The story so far

- Feature bank responses are biologically motivated image representations
- This is useful for recognition tasks:

→ Ex: Cluster the features into “visual words” to represent an image to do classification

→ Convolution with a linear kernel followed by simple non-linearities is a good model for early visual cortex (Retina, LGN, V1), but how to go beyond this?

How can we learn these filters??




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

*[ slide courtesy Yann LeCun ]*

# Next Step

- To learn good features, we first need a <u>task</u>!

# Quick Background on Statistical Learning Framework

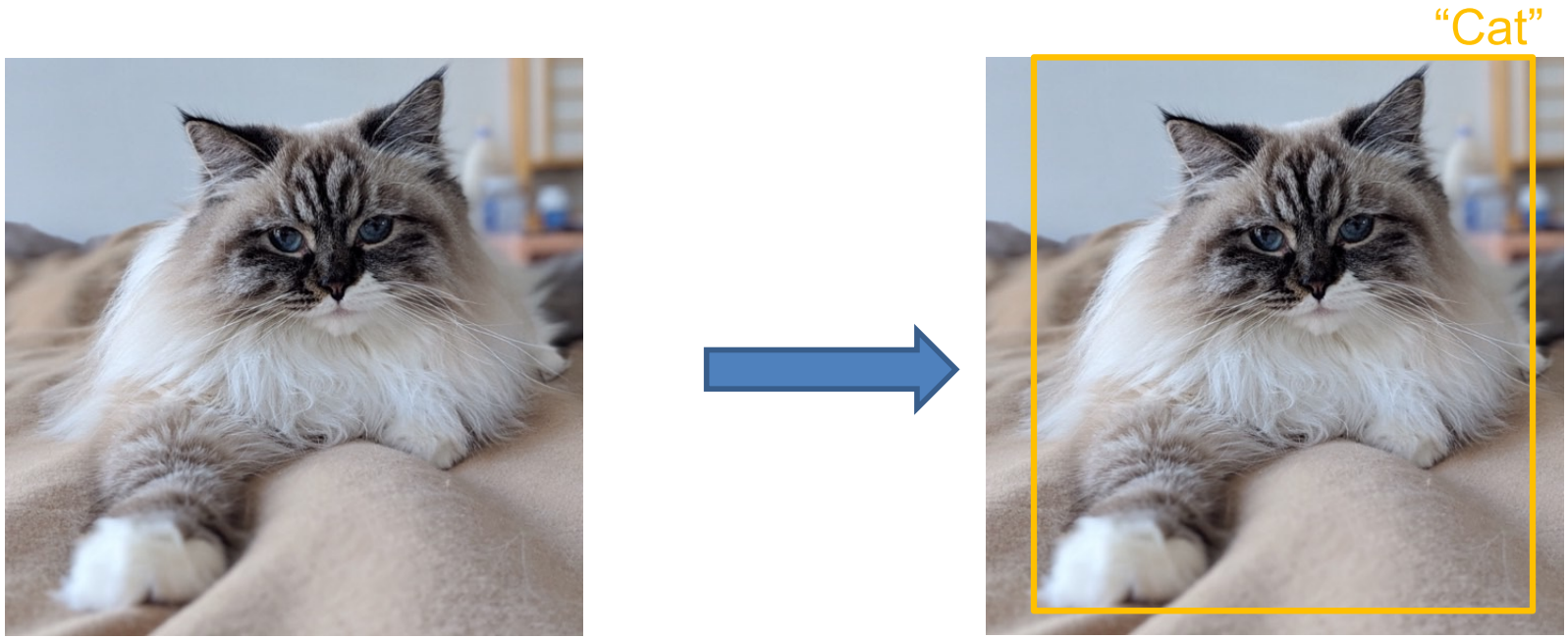
Slides courtesy of Lana Lezebnik

# Common Tasks

”Cat”

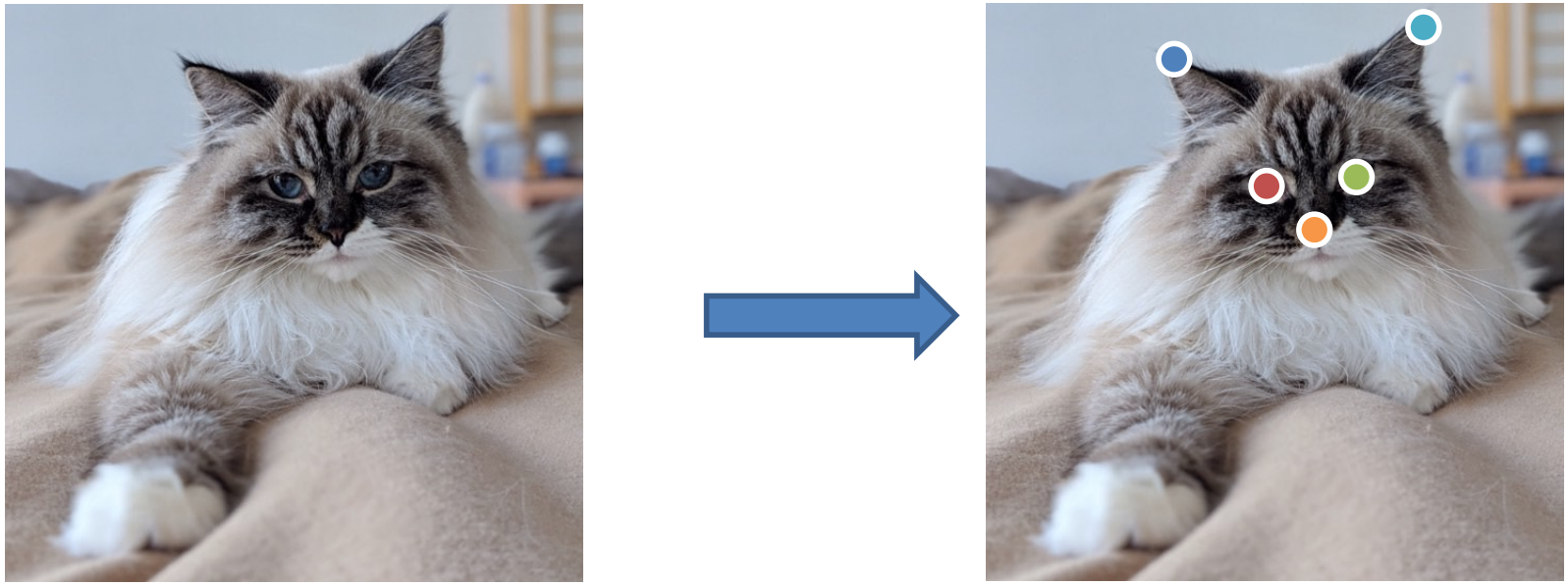
Image Classification

# Common Tasks




Object Detection

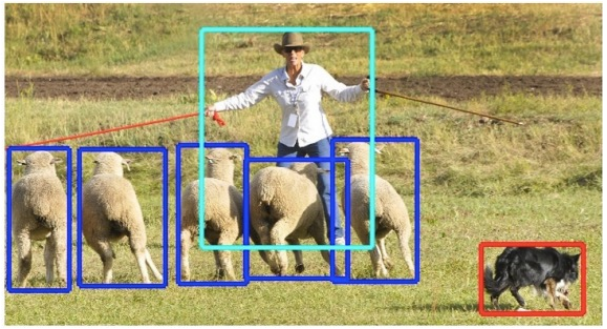
# Common Tasks



Keypoint Detection (project 5!)

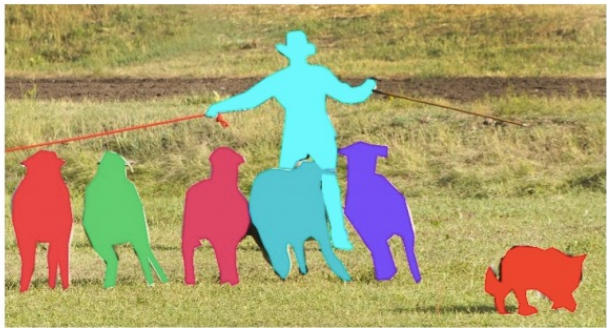
# Detection, semantic segmentation, instance segmentation


image classification



object detection

semantic segmentation



instance segmentation

Slide courtesy of Lana Lezebnik

Image source

# Image classification

Slides courtesy of Lana Lezebnik

# The statistical learning framework

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

f(  ) = “apple”

f(  ) = “tomato”

f(  ) = “cow”

Slide courtesy of Lana Lezebnik

# The statistical learning framework

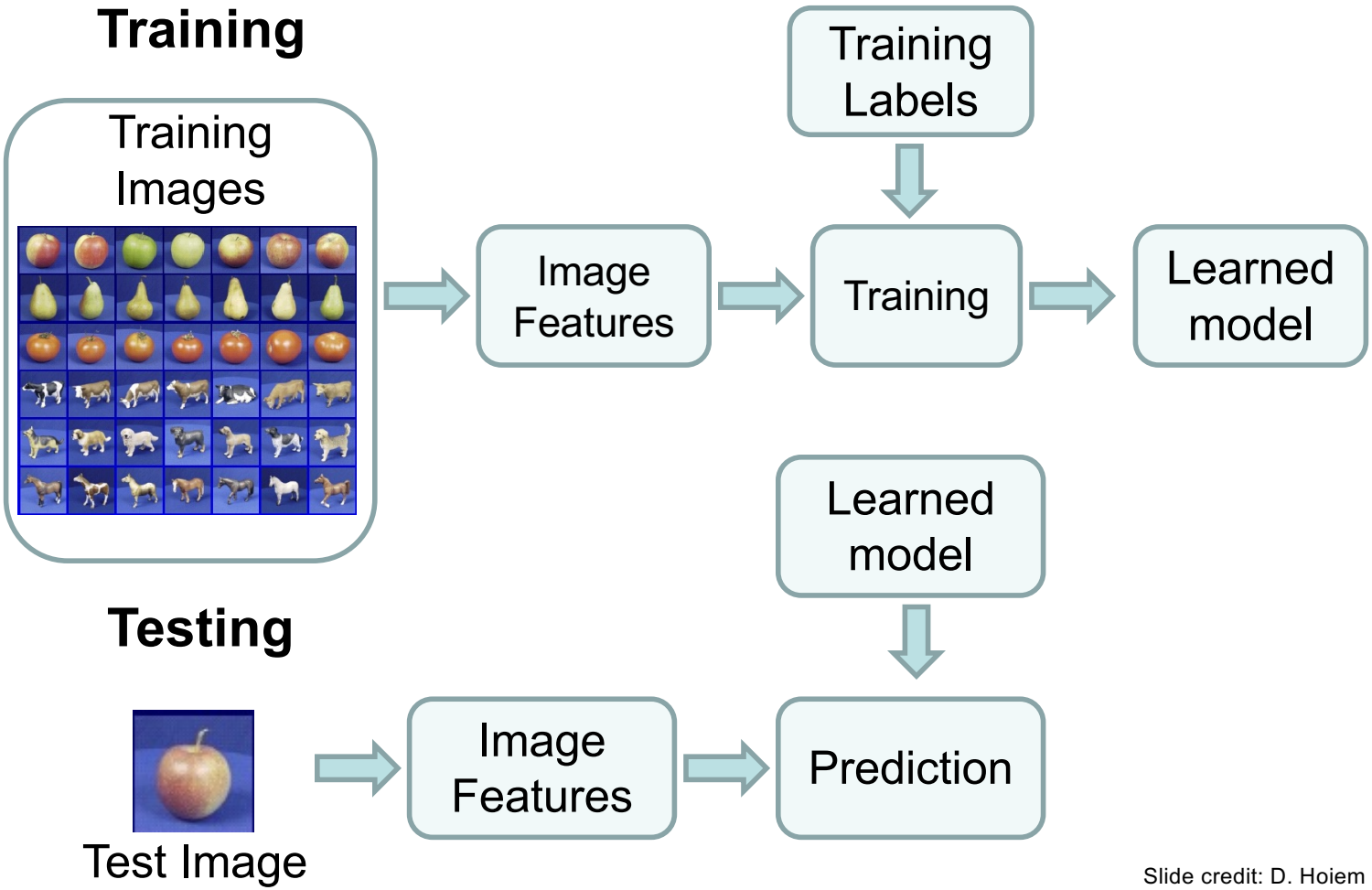
$$y = f(\mathbf{x})$$

output — prediction function — Input feature representation

- **Training set:** 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
- **Test set:** apply $f$ to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$
- **Validation set:** same as test, held-out to tune hyper-parameters. Never use the "test" set for tuning! That's cheating!

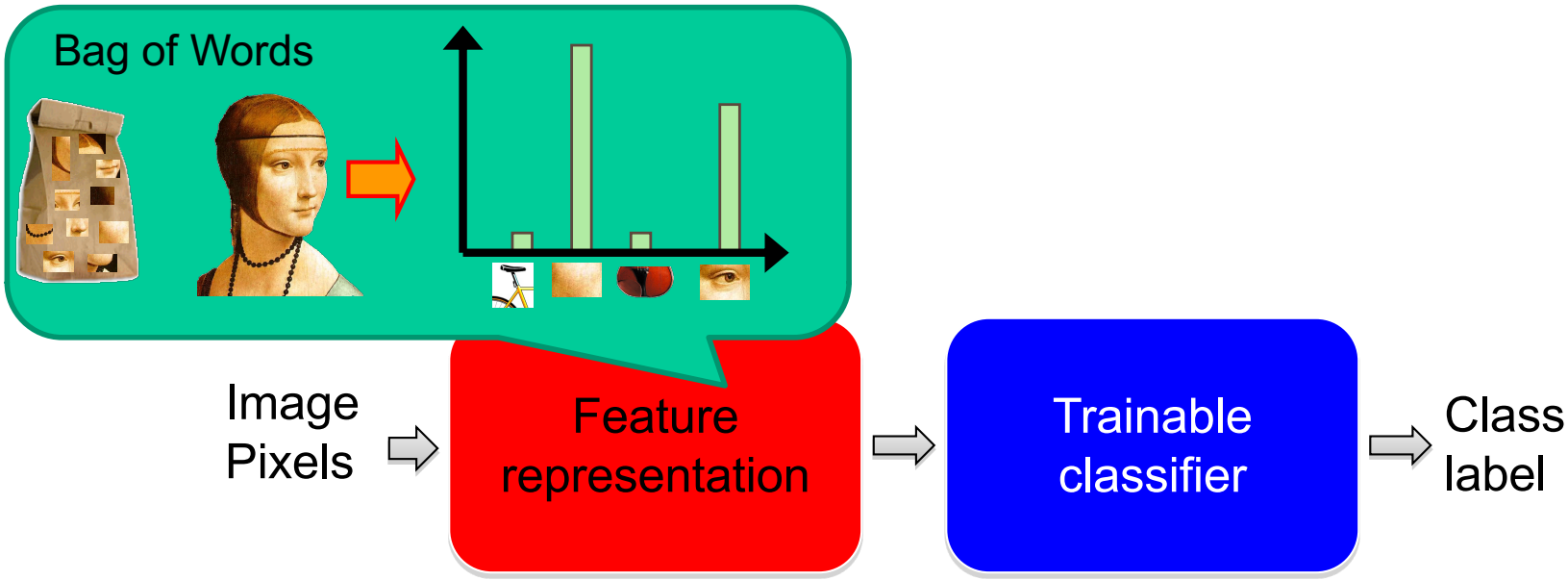
Slide modified from Lana Lezebnik

# Steps




Slide credit: D. Hoiem

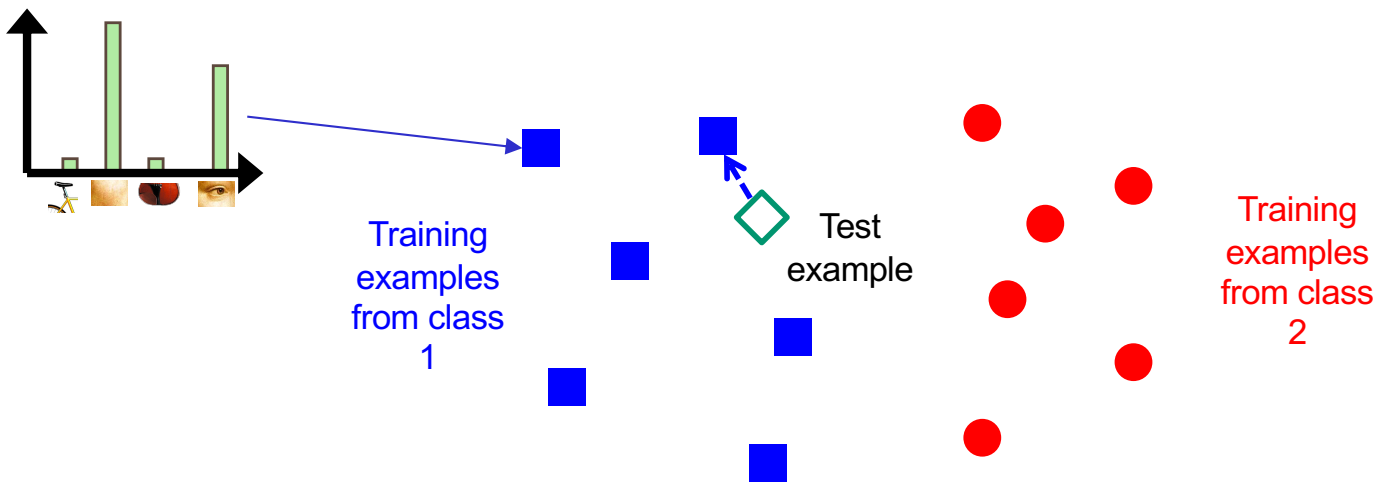
# “Classic” recognition pipeline




- Hand-crafted feature representation
- Off-the-shelf trainable classifier

Slide modified from Lana Lezebnik

# Classifiers: Nearest neighbor



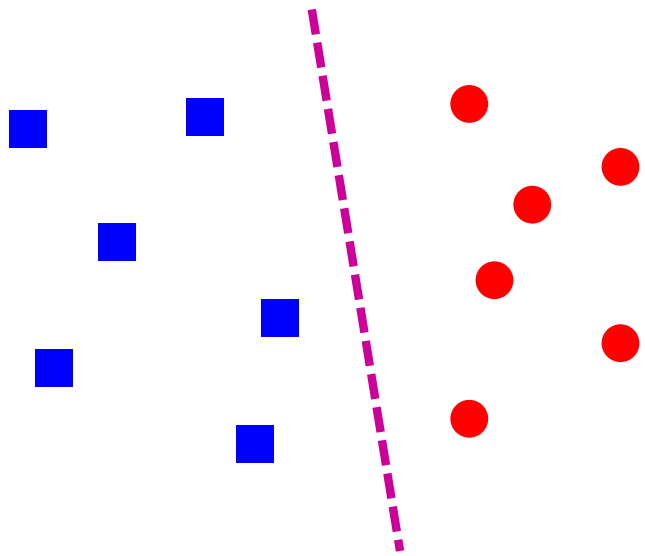

f(**x**) = label of the training example nearest to **x**

All we need is a distance or similarity function for our inputs

No training required!

Slide courtesy of Lana Lezebnik

# Linear classifiers: Binary classification

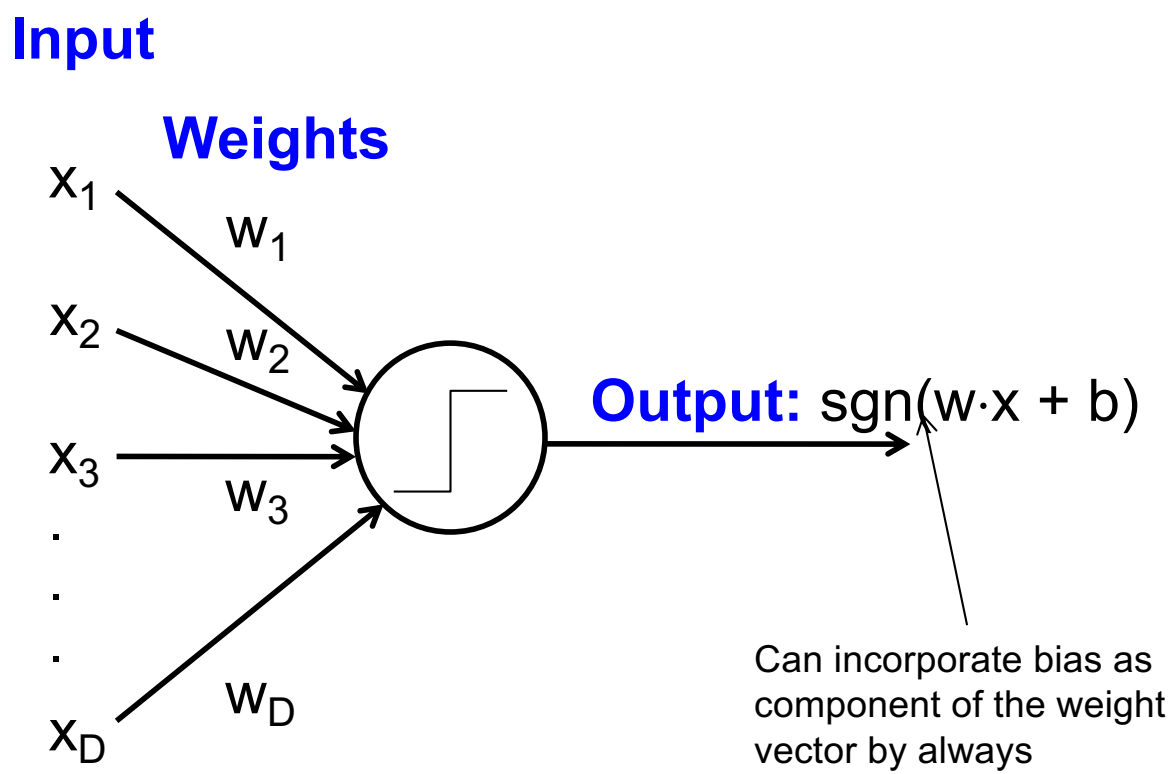


Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

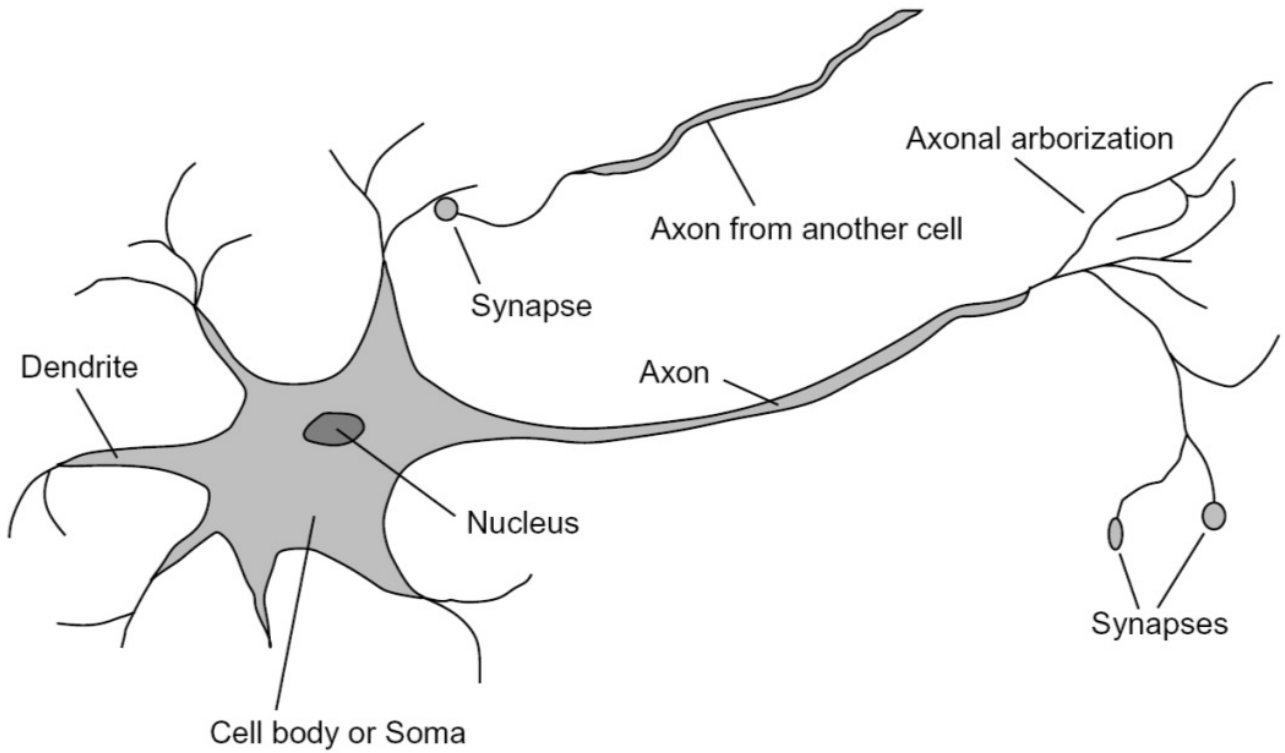
Slide courtesy of Lana Lezebnik

# Linear classifiers: Perceptron

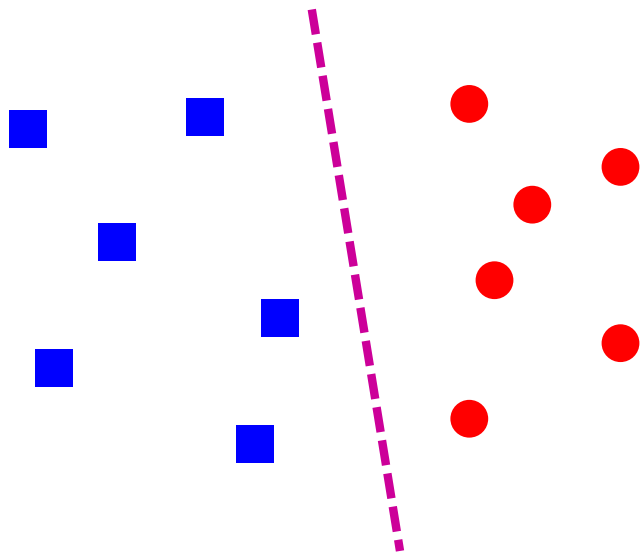



Can incorporate bias as component of the weight vector by always including a feature with value set to 1

# Loose inspiration: Human neurons



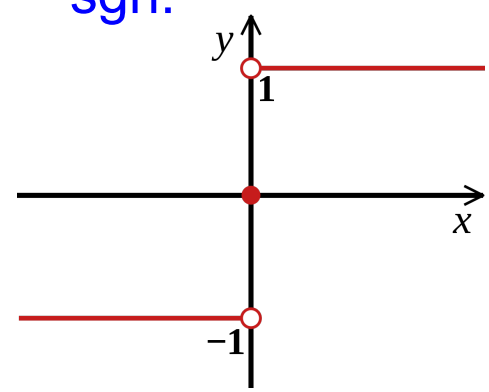
# Back to: Perceptrons



Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

sgn:




Slide courtesy of Lana Lezebnik

## Objective (Loss) functions

- Find weights w to minimize the prediction loss between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$$

- Possible losses (for binary problems):
  - **Quadratic loss:** $l(\mathbf{x}_i, y_i; \mathbf{w}) = (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$
  - Log Likelihood loss: $l(\mathbf{x}_i, y_i; \mathbf{w}) = -\log P_{\mathbf{w}}(y_i \mid \mathbf{x}_i)$

# Perceptron training algorithm

- Initialize weights **w** randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training example **x** with label y:
  - Classify with current weights:

$$y' = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$$

  - If classified incorrectly, update weights:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - y'\right)\mathbf{x}$$

(α is a positive *learning rate* that decays over time)

Slide courtesy of Lana Lezebnik

# Perceptron update rule: Binary Classification

$$y' = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - y'\right)\mathbf{x}$$

- The raw response of the classifier changes to

$$\mathbf{w} \cdot \mathbf{x} + \alpha\left(y - y'\right)\left\|\mathbf{x}\right\|^2$$

- If *y* = 1 and *y'* = -1, the response is initially *negative* and will be *increased*
- If *y* = -1 and *y'* = 1, the response is initially *positive* and will be *decreased*

Slide courtesy of Lana Lezebnik

# Linaer Classifiers: Beyond Binary Classification

Example Setup: 3 classes



Model – one weight per class: $\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{w}_2$

$\boldsymbol{w}_0^T\boldsymbol{x}$ big if cat

$\boldsymbol{w}_1^T\boldsymbol{x}$ big if dog

$\boldsymbol{w}_2^T\boldsymbol{x}$ big if hippo

Stack together: $\boldsymbol{W}_{3xF}$ where **x** is in $R^F$

# 3-Classes

| | | | | | |
|---|---|---|---|---|---|
| Cat weight vector | 0.2 | -0.5 | 0.1 | 2.0 | 1.1 |
| Dog weight vector | 1.5 | 1.3 | 2.1 | 0.0 | 3.2 |
| Hippo weight vector | 0.0 | 0.3 | 0.2 | -0.3 | -1.2 |

$\boldsymbol{W}$

| | |
|---|---|
| | 56 |
| | 231 |
| | 24 |
| | 2 |
| bias | 1 |

$\boldsymbol{x_i}$

| | |
|---|---|
| -96.8 | Cat score |
| 437.9 | Dog score |
| 61.95 | Hippo score |

$\boldsymbol{W x_i}$

Weight matrix a collection of scoring functions, one per class

Prediction is vector where jth component is "score" for jth class.

Diagram by: Karpathy, Fei-Fei

# Training 3-Classes

| | | | | | | |
|---|---|---|---|---|---|---|
| Cat weight vector | 0.2 | -0.5 | 0.1 | 2.0 | 1.1 | 56 |
| Dog weight vector | 1.5 | 1.3 | 2.1 | 0.0 | 3.2 | 231 |
| Hippo weight vector | 0.0 | 0.3 | 0.2 | -0.3 | -1.2 | 24 |
| | | | | | | 2 |
| | | | | | bias | 1 |

$\boldsymbol{W}$

$\boldsymbol{x_i}$

Weight matrix a collection of scoring functions, one per class

| | |
|---|---|
| -96.8 | Cat score |
| 437.9 | Dog score |
| 61.95 | Hippo score |

$$\boldsymbol{s_i} = \boldsymbol{W x_i}$$

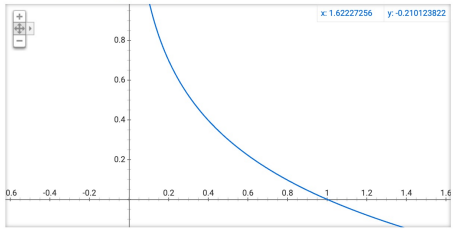
Prediction is vector where jth component is "score" for jth class.

| | |
|---|---|
| 0.00000... | Cat probability |
| .9999 | Dog probability |
| 0.0000..... | Hippo probability |

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Want the correct class to have probability = 1

Graph for -log(x)

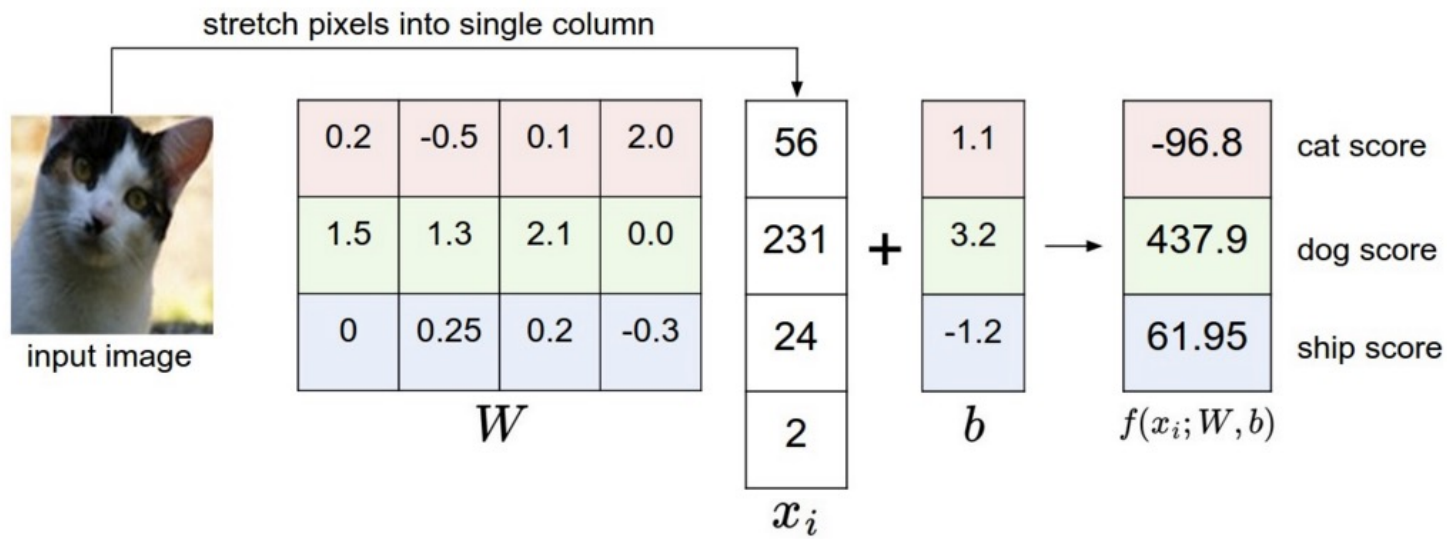


Loss

$$L_i = -\log(p_{y_i})$$

Diagram by: Karpathy, Fei-Fei

# Visualizing linear classifiers





Source: Andrej Karpathy, http://cs231n.github.io/linear-classify/

# Linear Classifiers: Visual Intuition

Decision rule is $\mathbf{w}^T\mathbf{x}$. If $\mathbf{w}_i$ is big, then big values of $x_i$ are indicative of the class.

## Deer or Plane?

# Linear Classifiers: Visual Intuition

Decision rule is $\mathbf{w}^T\mathbf{x}$. If $\mathbf{w}_i$ is big, then big values of $x_i$ are indicative of the class.

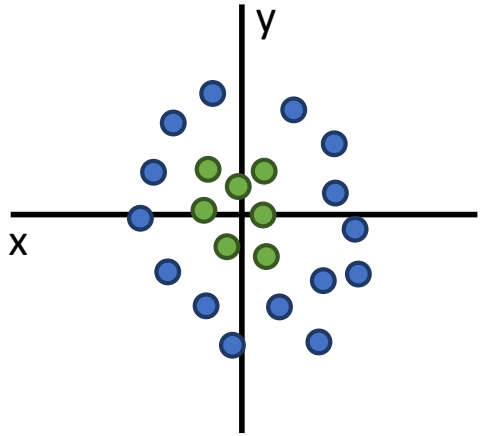
## Ship or Dog?

# Are linear classifiers enough??

One template per class:
Can’t recognize different modes of a class

# Why?? Linear Classifiers not enough

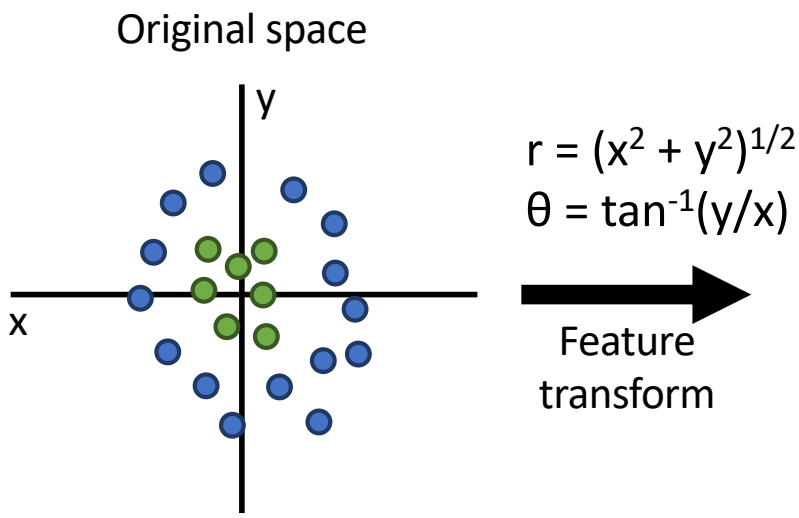
Geometric Viewpoint



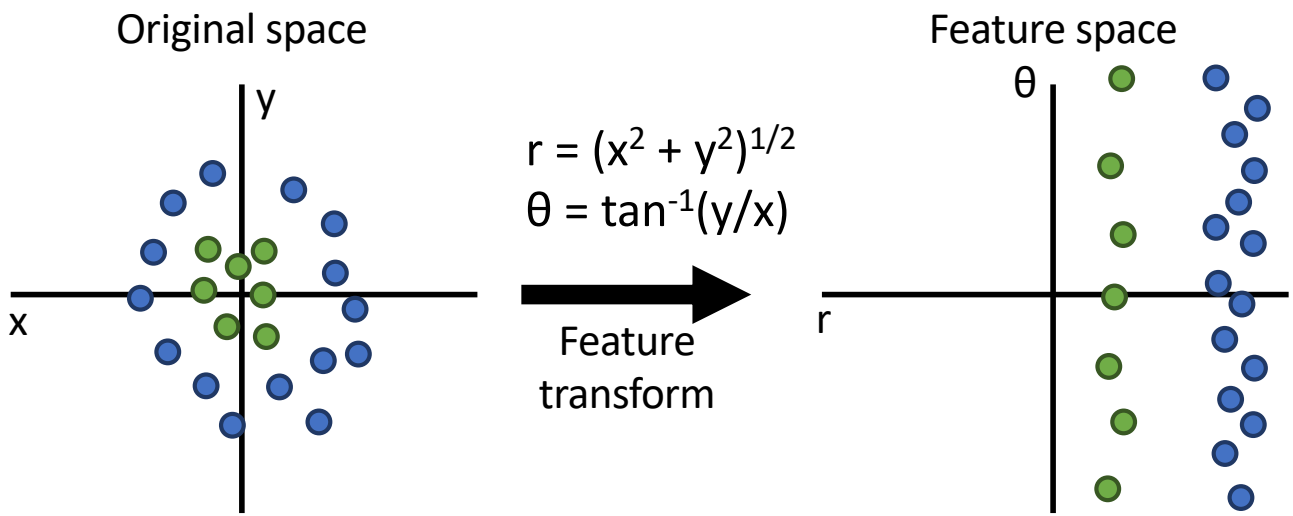

Visual Viewpoint

One template per class:
Can't recognize different
modes of a class

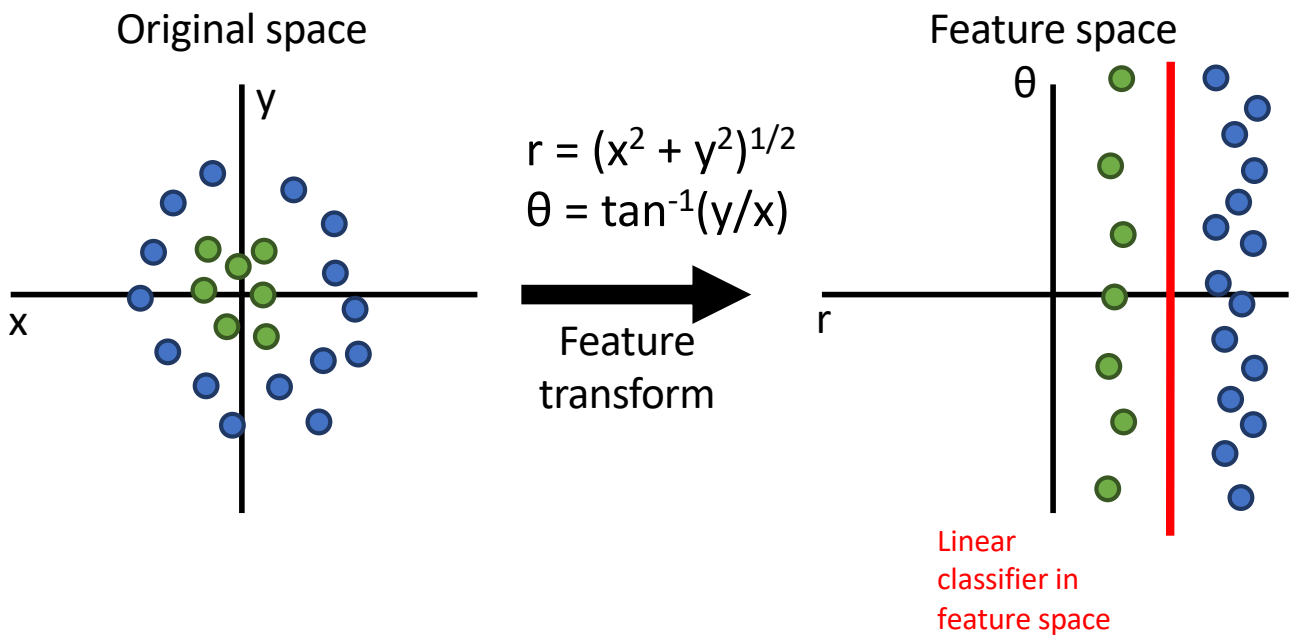
# One solution: **Feature Transforms**



# One solution: **Feature Transforms**

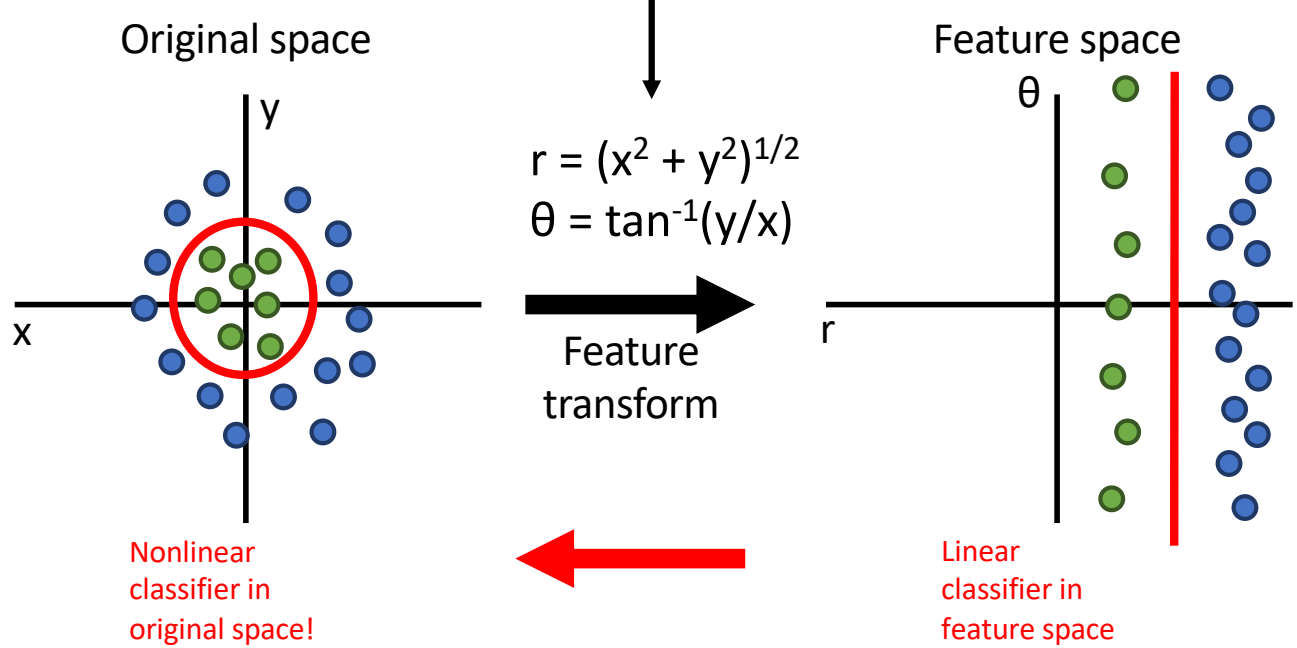


# One solution: **Feature Transforms**

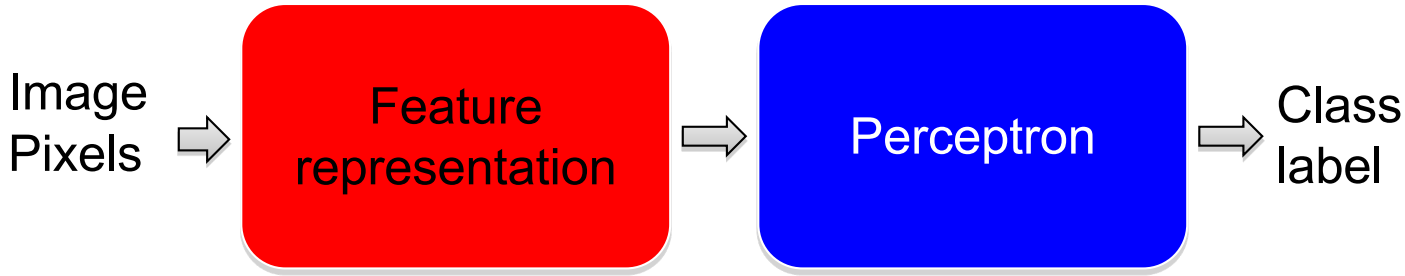


# One solution: **Feature Transforms**

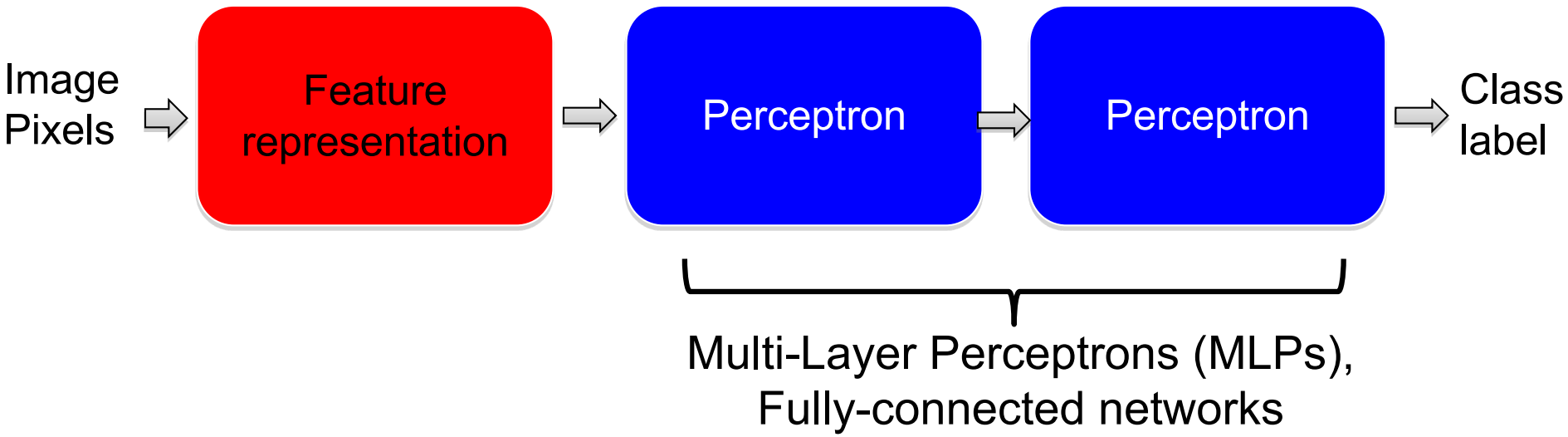
Neural Networks (MLPs) learn the transforms itself from data



# Recognition pipeline: With Perceptrons



# Recognition pipeline: With Multiple Perceptrons



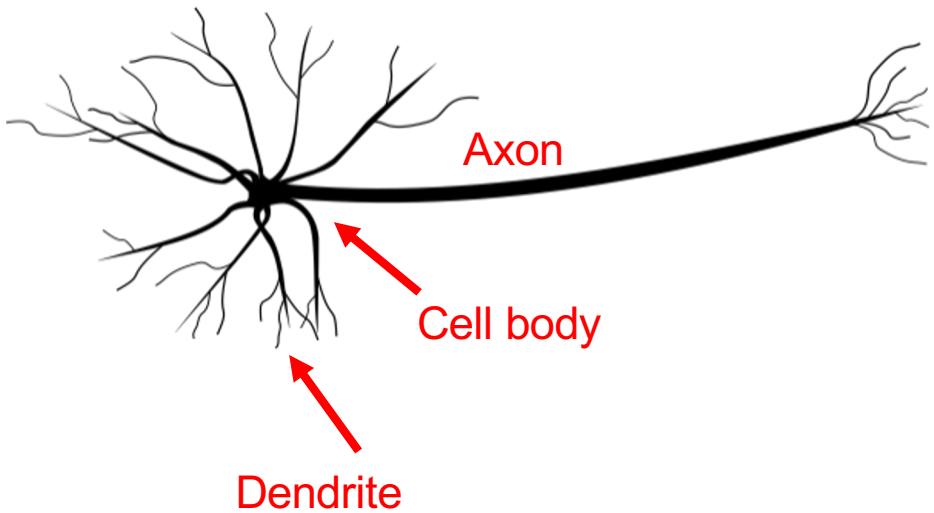

Can replace feature computation with layers of perceptrons!
= Deep Network

# “Deep” recognition pipeline

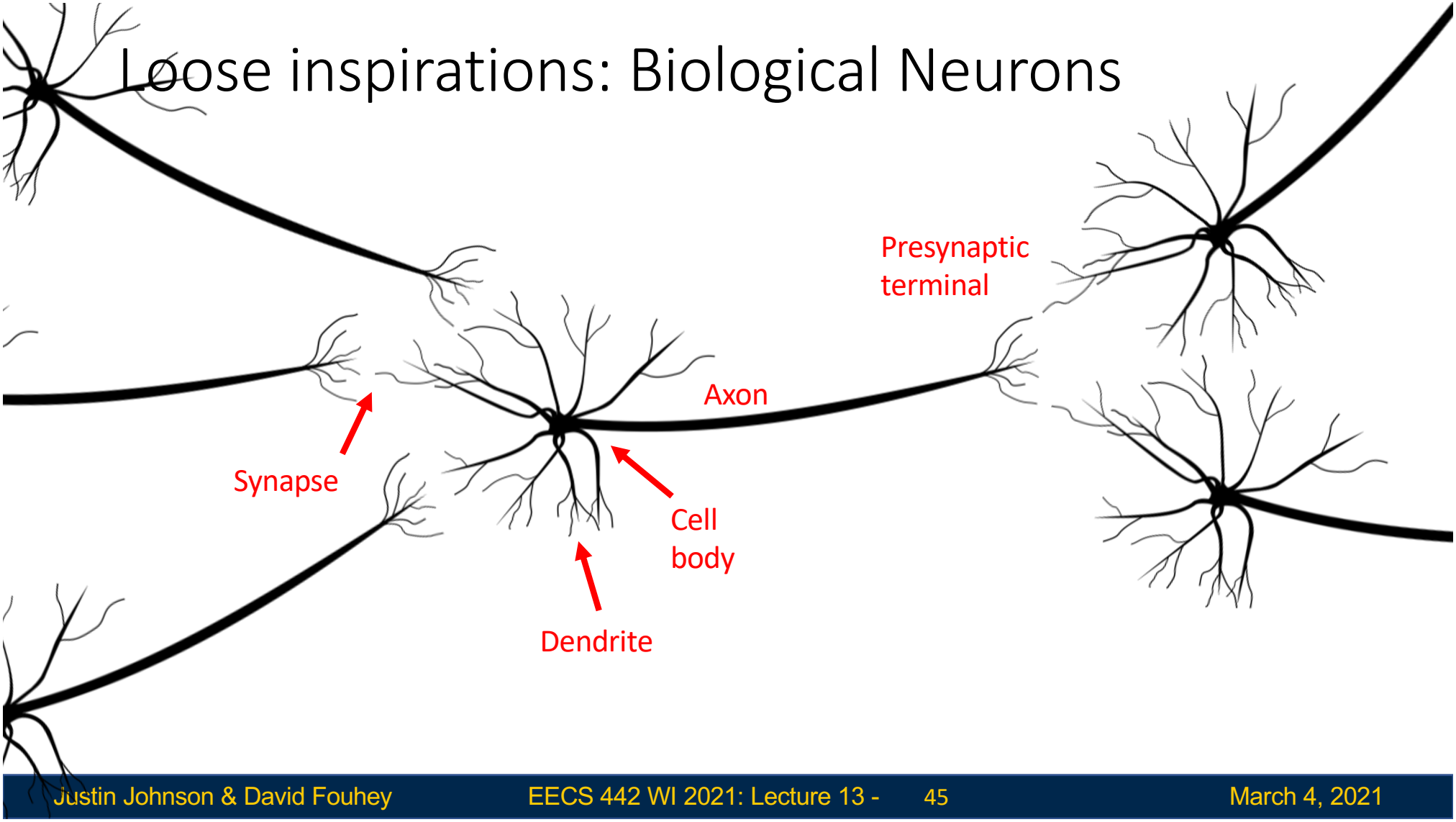

- Learn a *feature hierarchy* from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

Slide courtesy of Lana Lezebnik

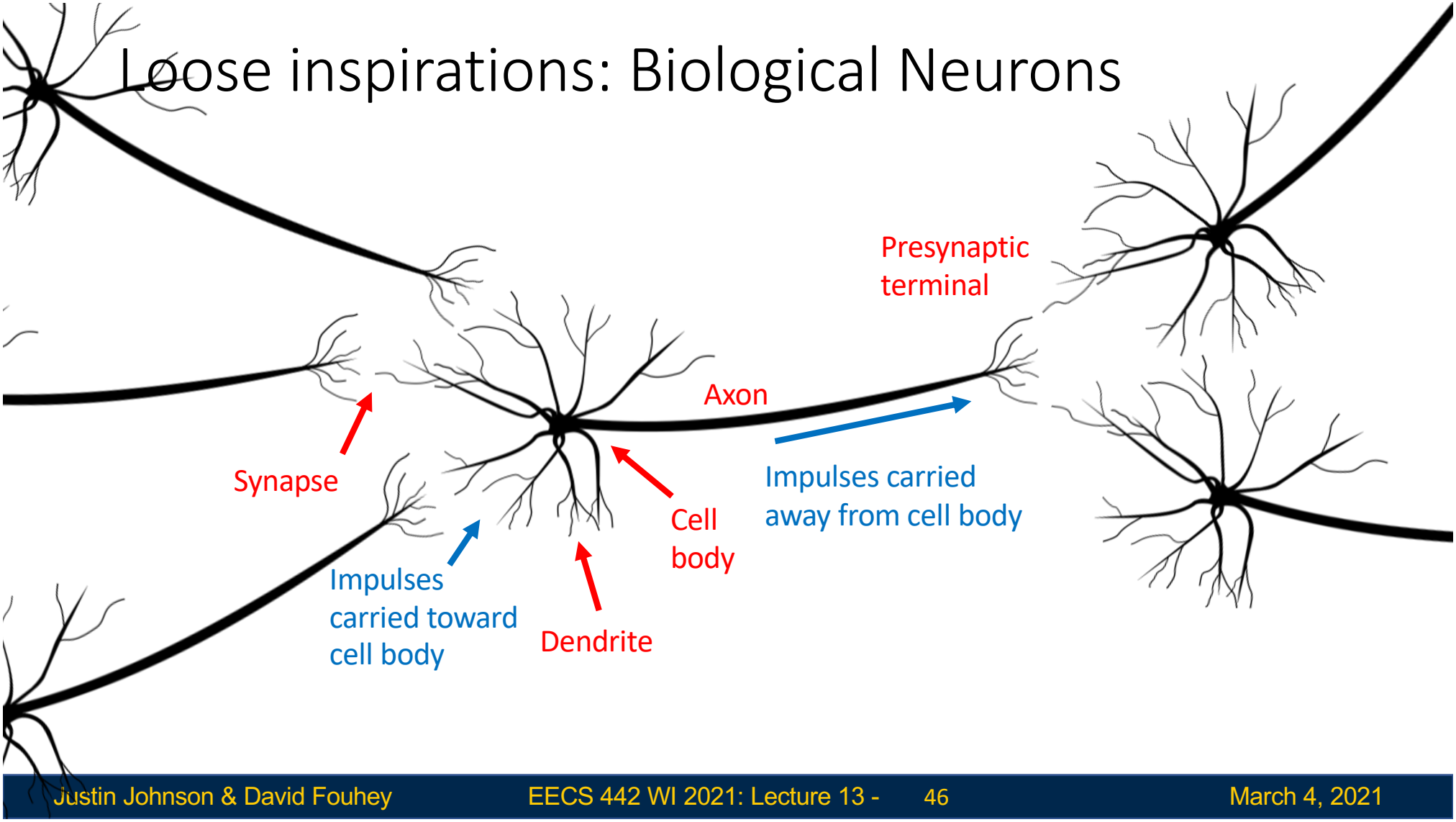
# Loose inspirations: Biological Neurons



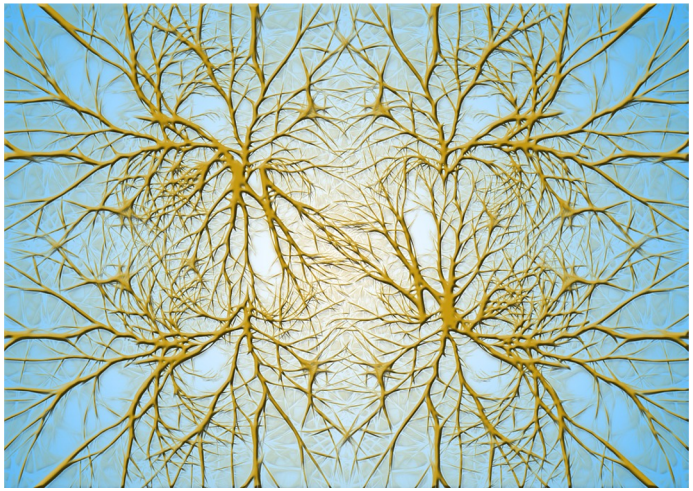
# Loose inspirations: Biological Neurons



# Loose inspirations: Biological Neurons

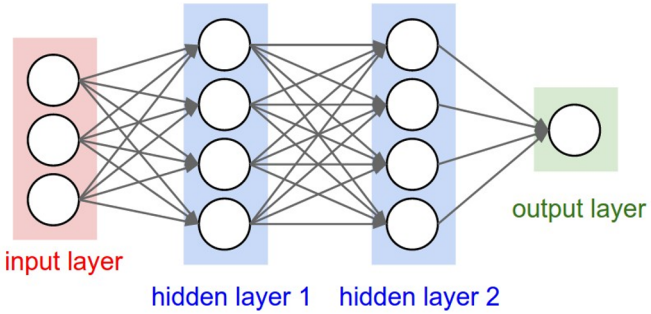


Biological Neurons:
Complex connectivity patterns





Neurons in a neural network:
Organized into regular layers
for computational efficiency



# Be very careful with brain analogies!

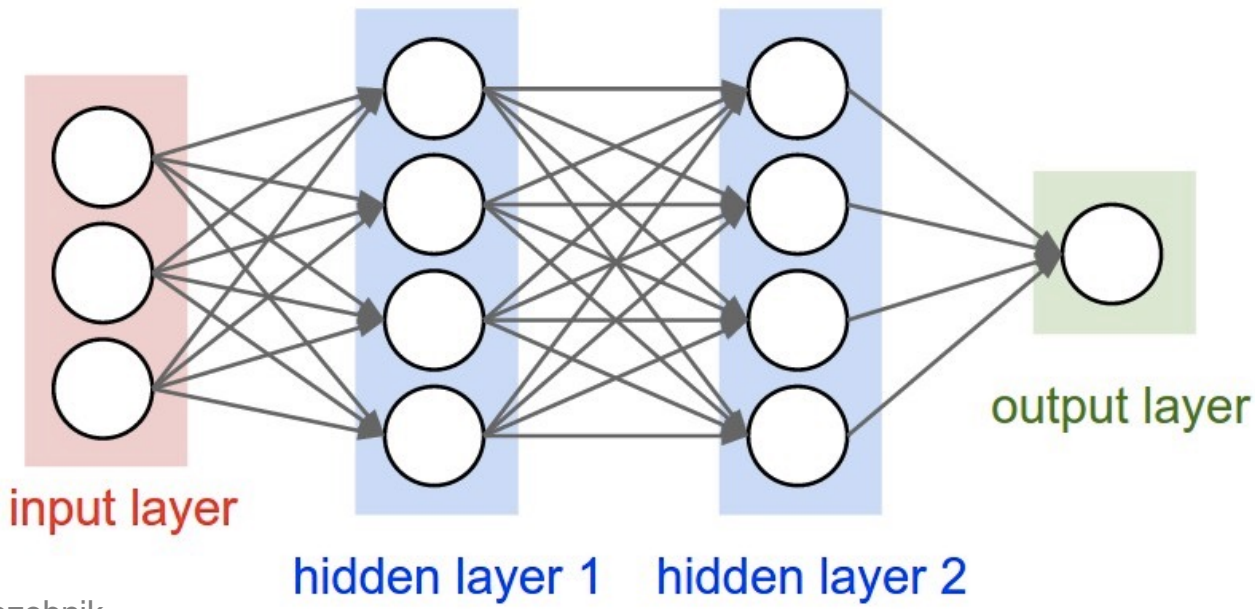
**Biological Neurons:**

- Many different types
- Can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Can have feedback, time-dependent
- Probably don't learn via gradient descent

[Dendritic Computation. London and Hausser]

# Multi-layer perceptrons

- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
  - This requires each perceptron to have a nonlinearity



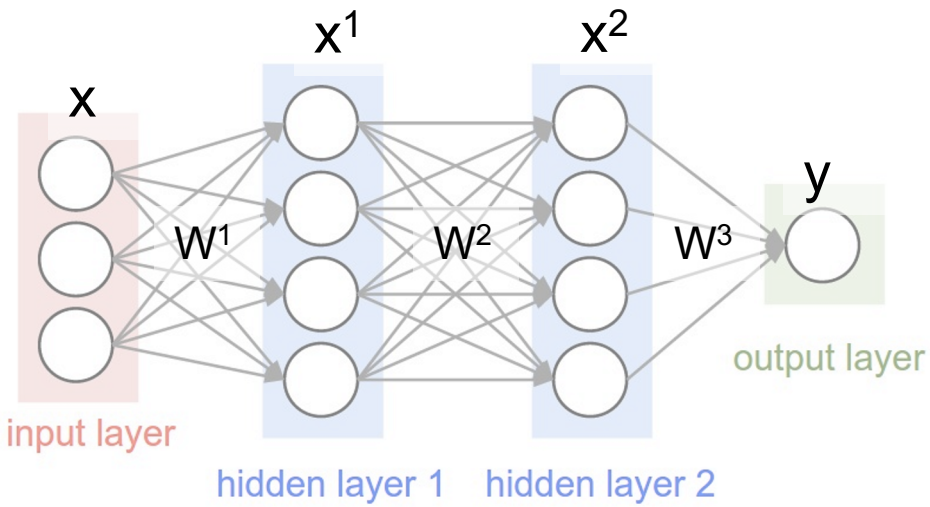

Slide courtesy of Lana Lezebnik

# Remember this?

”Cat”

Image Classification

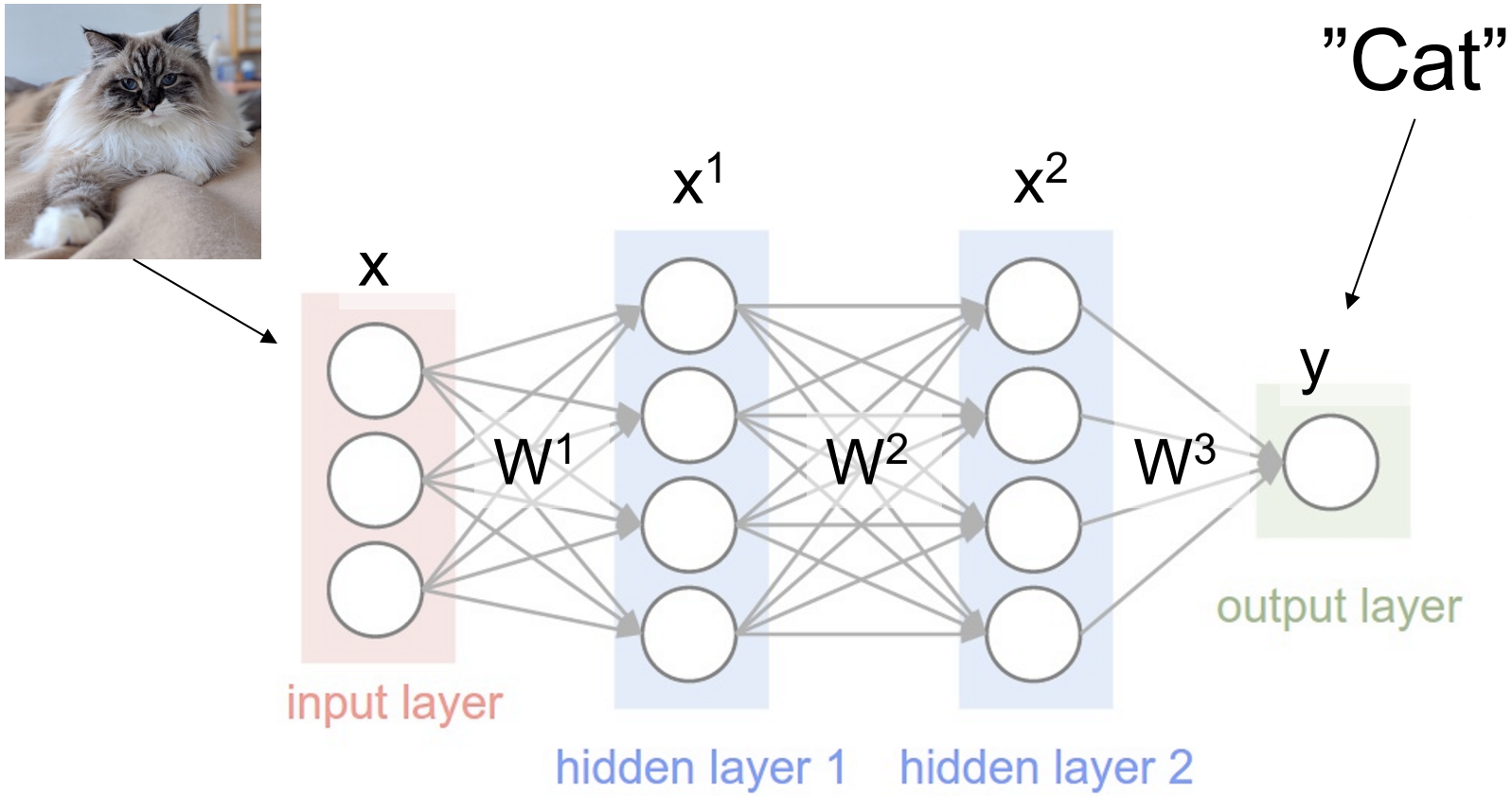
# Remember this?




”Cat”

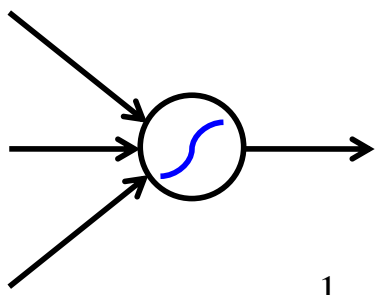
Image Classification

# Multi-layer perceptrons / Fully-Connected Layer

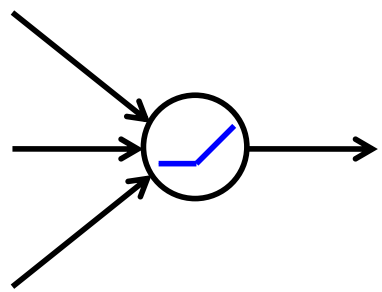


# Multi-layer perceptrons

- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
  - This requires each perceptron to have a nonlinearity
  - To be trainable, the nonlinearity should be *differentiable*



**Sigmoid:** $g(t) = \frac{1}{1+e^{-t}}$



**Rectified linear unit (ReLU):** $g(t) = \max(0,t)$

Most popular. Use subderivative at 0,

Slide courtesy of Lana Lezebnik

# Why do we need non-linearities?

**Input image**: $x \in \mathbb{R}^D$
**Category scores**: $s \in \mathbb{R}^C$

**Linear Classifier**:

$$s = Wx$$
$$W \in \mathbb{R}^{C \times D}$$

**2-layer Neural Net:**

$$s = \mathrm{W}_2 \max(0, W_1 x)$$
$$W_1 \in \mathbb{R}^{H \times D}$$
$$W_2 \in \mathbb{R}^{C \times H}$$

**3-layer Neural Net:**

$$s = \mathrm{W}_3 \max(0, W_2 \max(0, W_1 x))$$

**Q**: with no activation function?

$$s = W_2 W_1 x$$

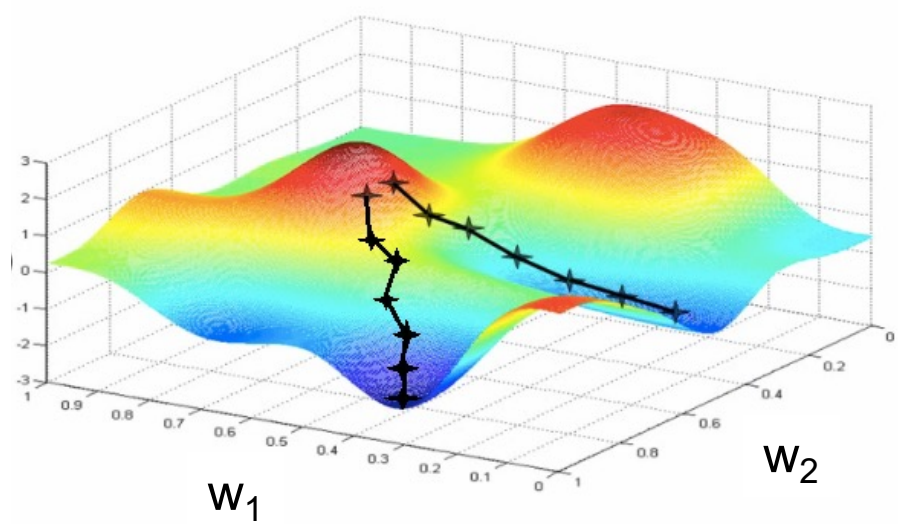
Slide modified from Johnson & Fouhey

# Training of multi-layer networks

- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$




Slide courtesy of Lana Lezebnik

# Training of multi-layer networks

- Find network weights to minimize the prediction loss between true and estimated labels of training examples:

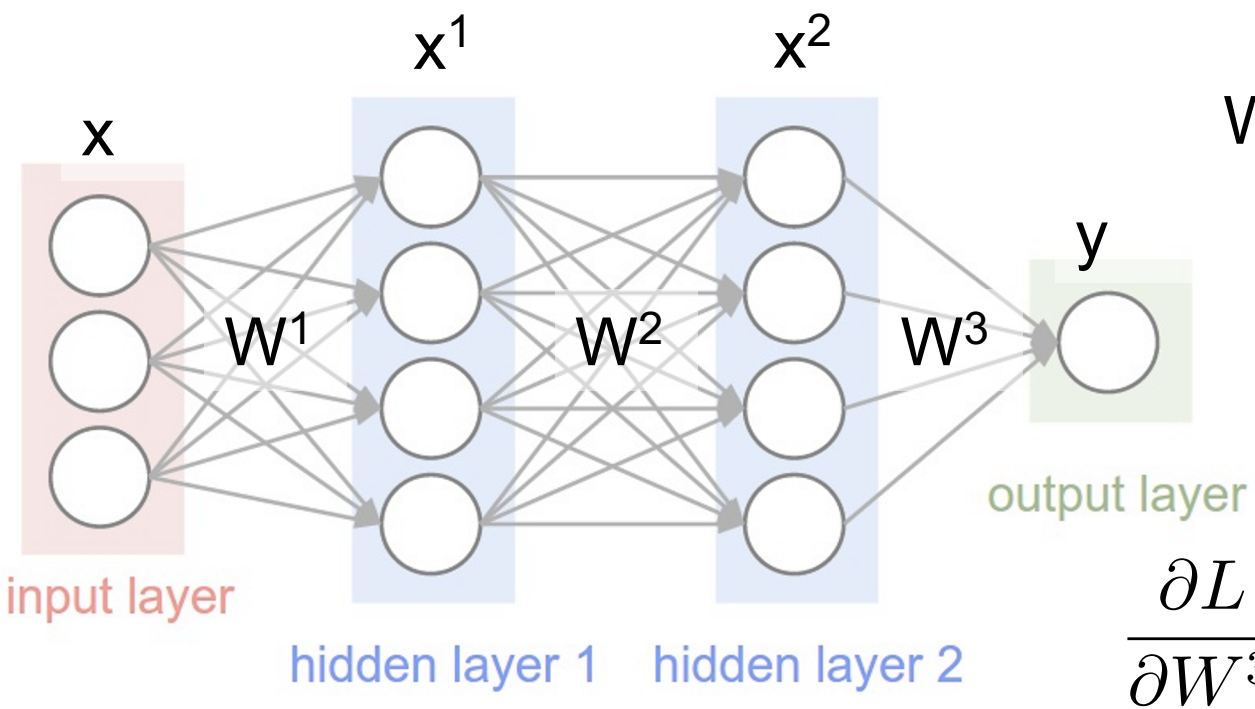
$$E(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Slide courtesy of Lana Lezebnik

# Back prop in 2 slides




Where $x^l = \sigma(W^l x^{l-1})$

What do we need for training Ws?

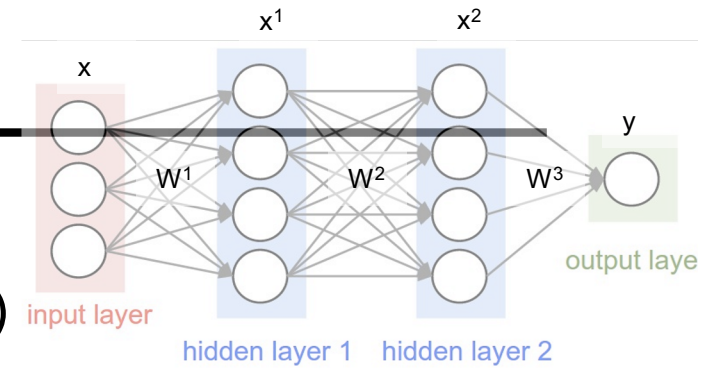
$$\frac{\partial L}{\partial W^3}, \frac{\partial L}{\partial W^2}, \frac{\partial L}{\partial W^1}$$

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial W^3}$$

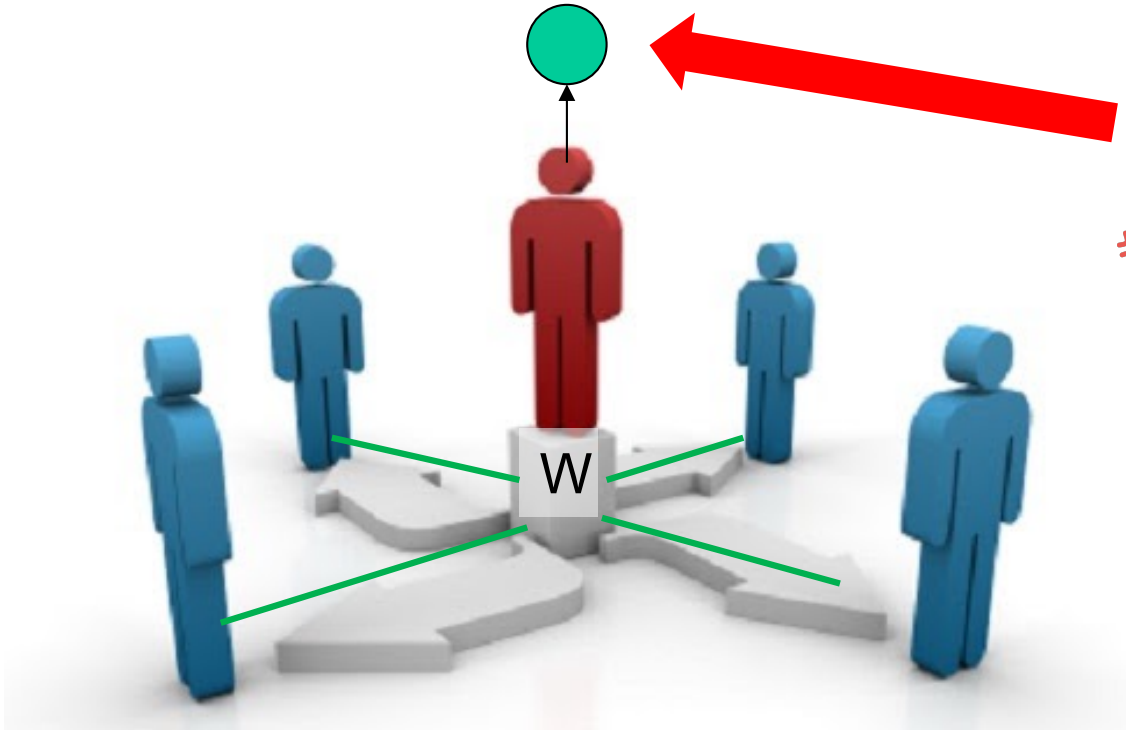
Chain rule!!

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x^2}\frac{\partial x^2}{\partial W^2}$$

# Back prop intuition




1. Every single layer is like a team of neurons (x), that a manager (W) is listening to decide outputs (y)




2. Then you get a feedback from higher ups of what you should've done

3. You adjust your confidence in each of your neurons,
4. Then pass on what they should've said

# The point is

Each layer needs to adjust how it listened to each neuron (W) and compute gradients wrt its input and pass it down.

$$\frac{\partial L}{\partial W^l} = \frac{\partial L}{\partial x^{l+1}} \frac{\partial x^{l+1}}{\partial W^l}$$

Updates: Higher up orders multiplied by how much the manager listened to each neuron

$$\frac{\partial L}{\partial x^l} = \frac{\partial L}{\partial x^{l+1}} \frac{\partial x^{l+1}}{\partial x^l}$$

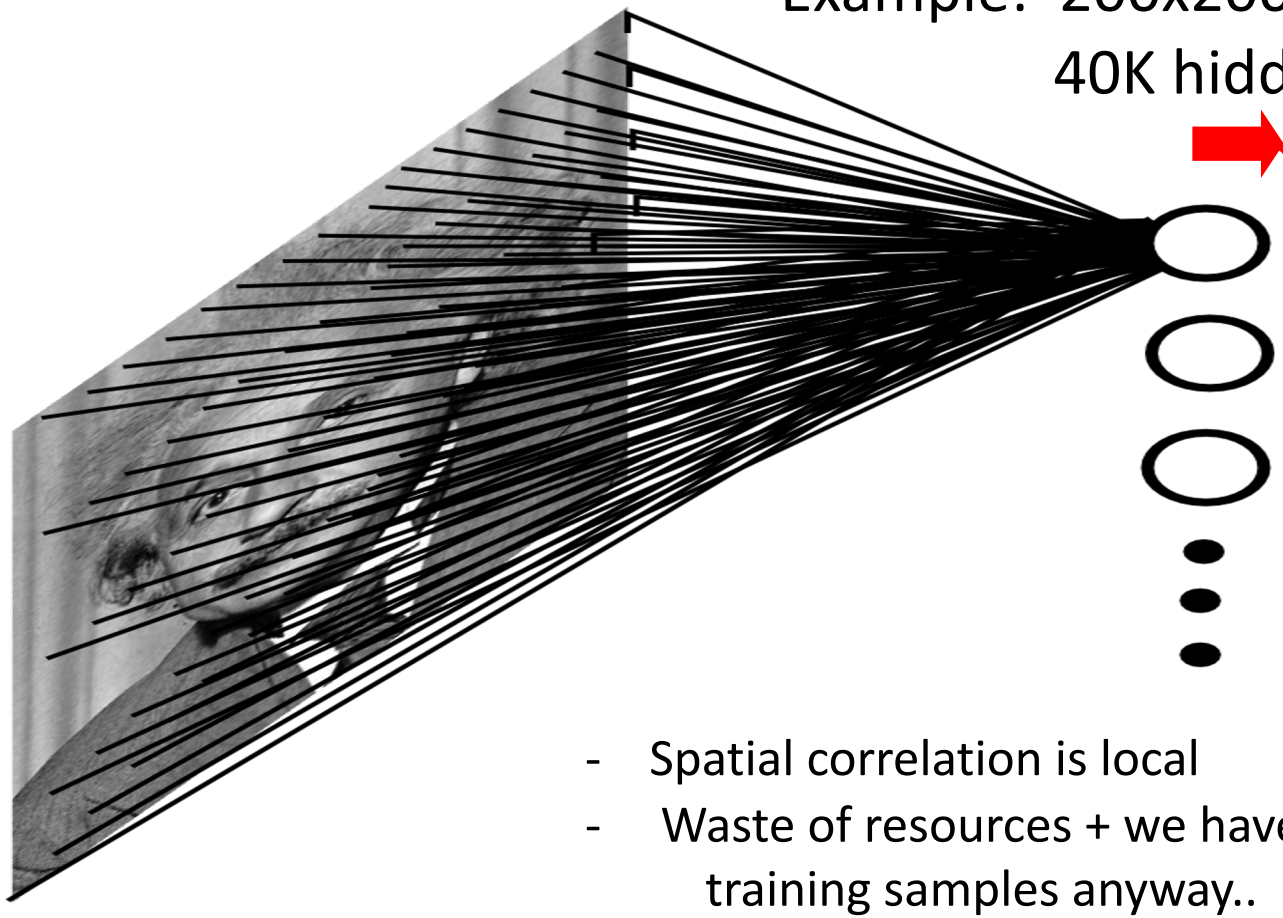
Pass down: Higher up orders, weighted by how much the manager listened to each neuron

# What we have so far: Fully Connected Layers (MLPs)

Example: 200x200 image

40K hidden units

**2B parameters**!!!

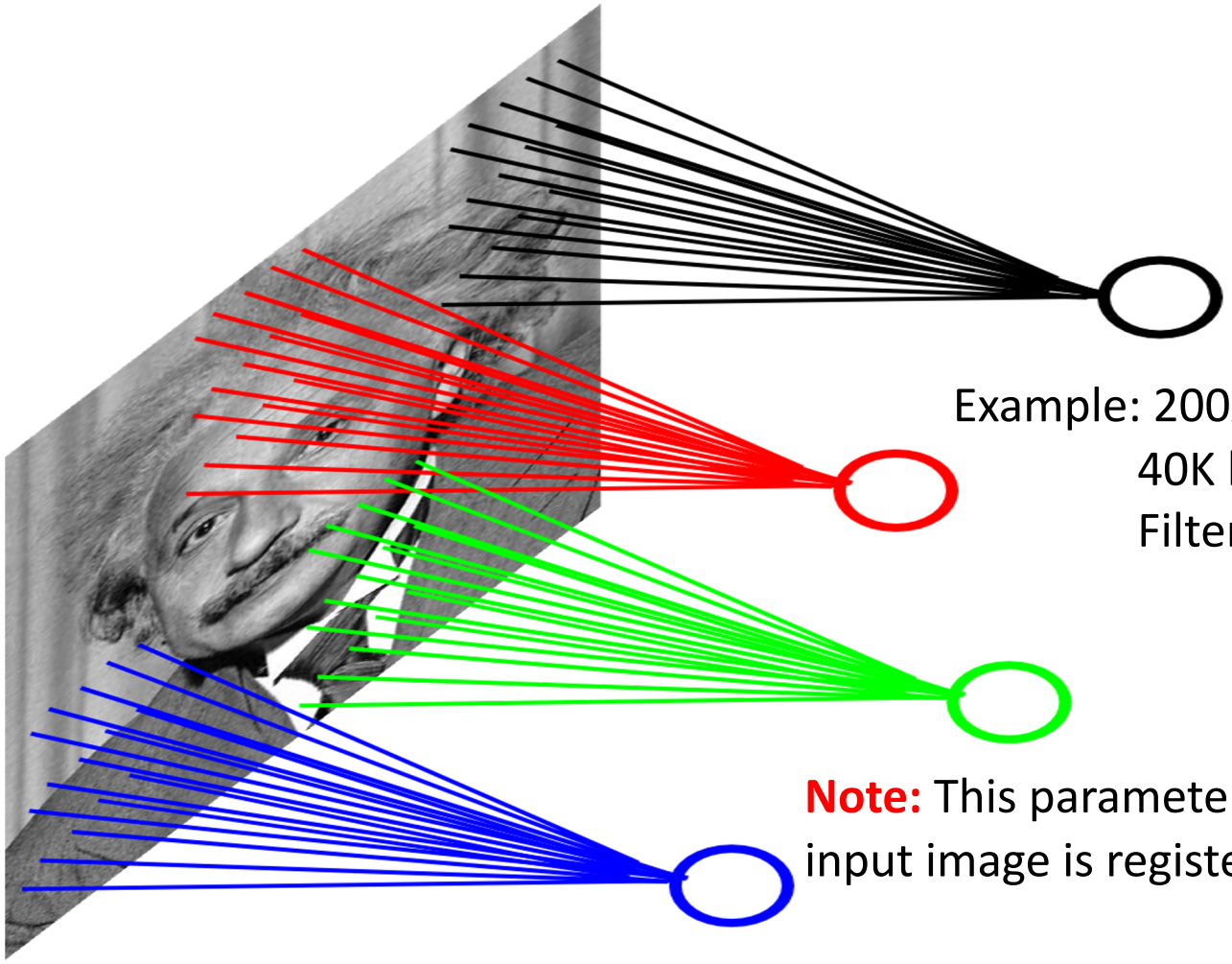


- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..



Ranzato

# Locally Connected Layer



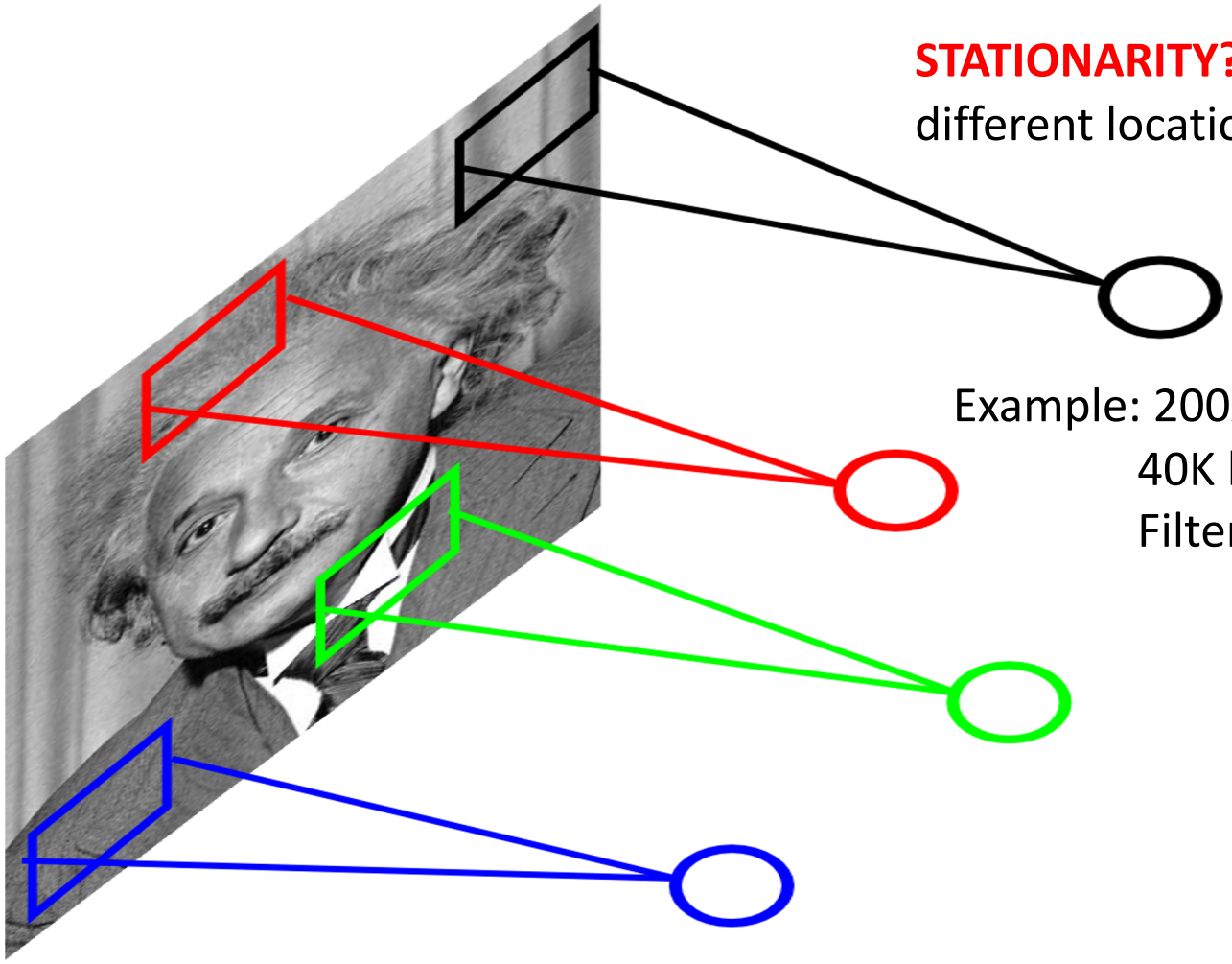
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

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



Ranzato

# Locally Connected Layer



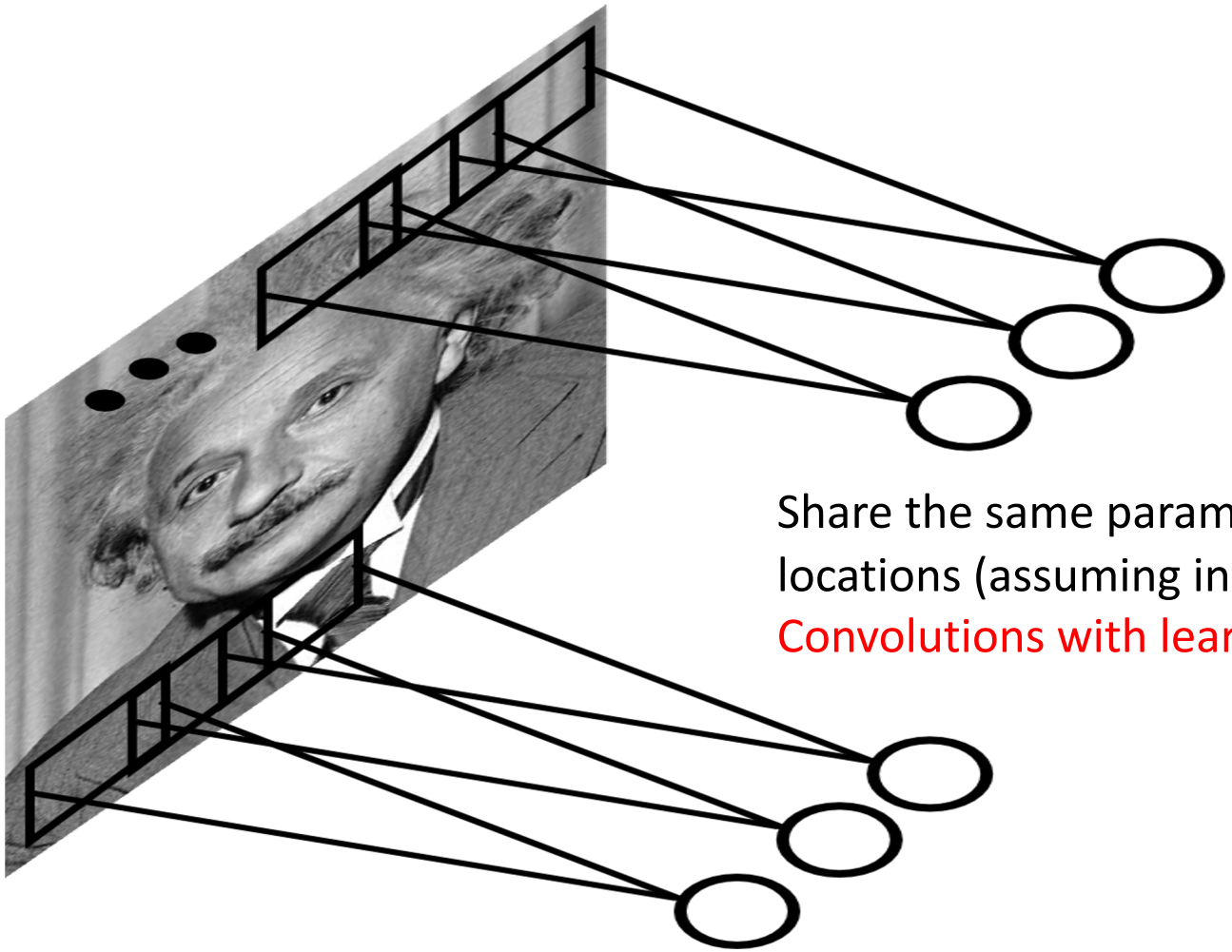
**STATIONARITY?** Statistics is similar at different locations

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



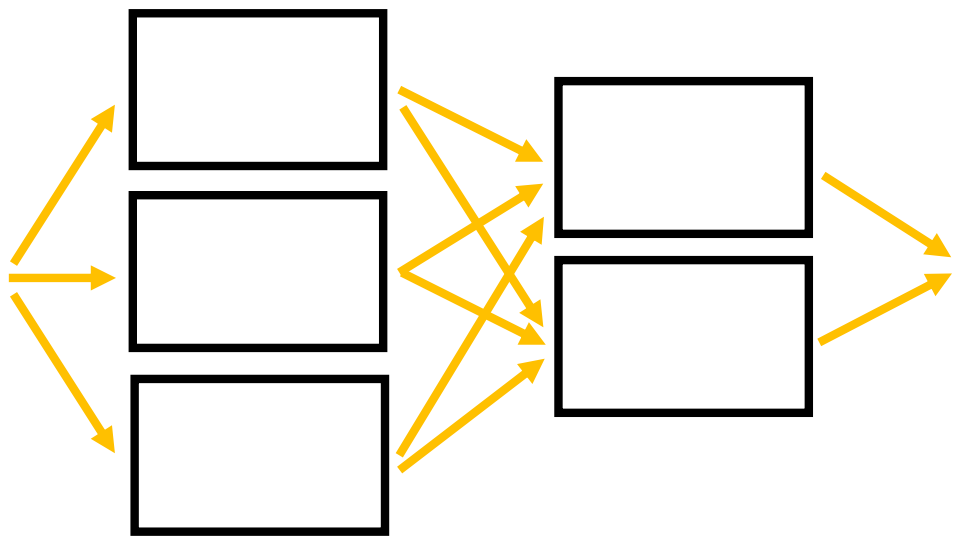
Ranzato

# Convolutional Layer



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

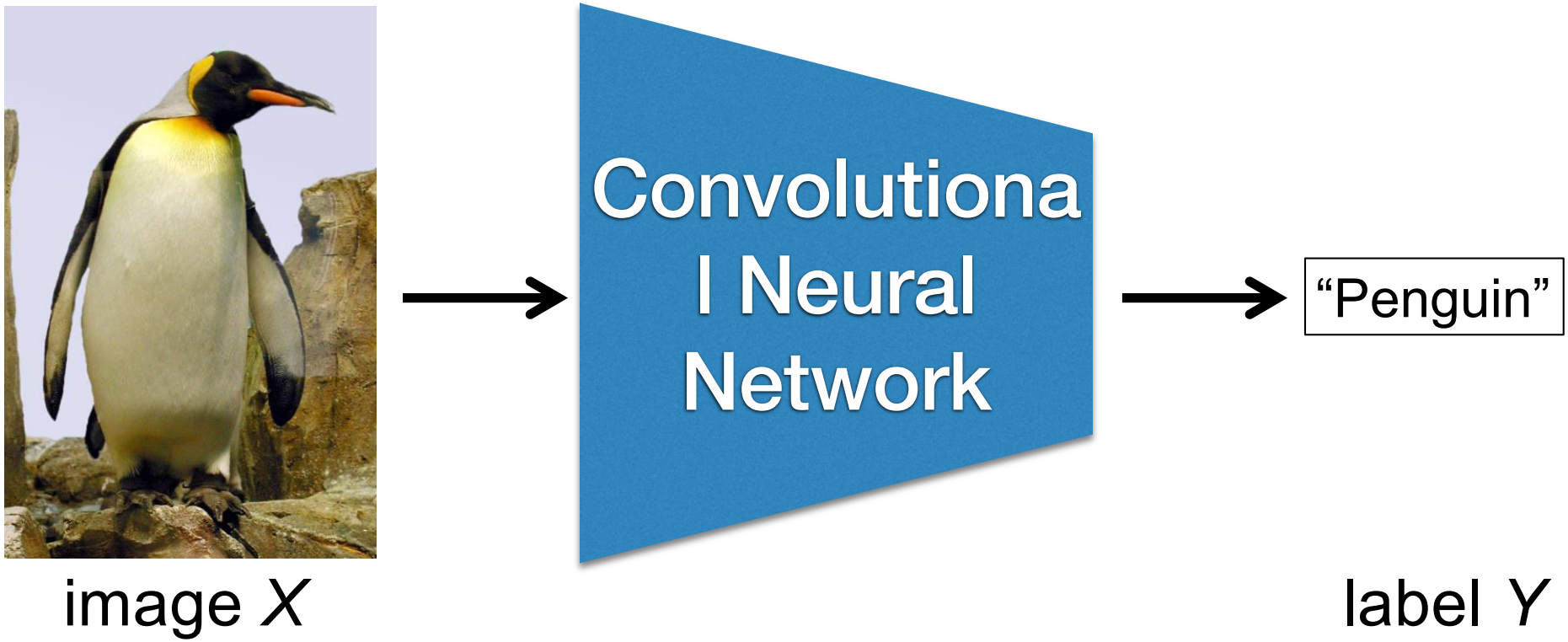
# Convolutional Neural Networks



CS194: Computer Vision and Comp. Photo

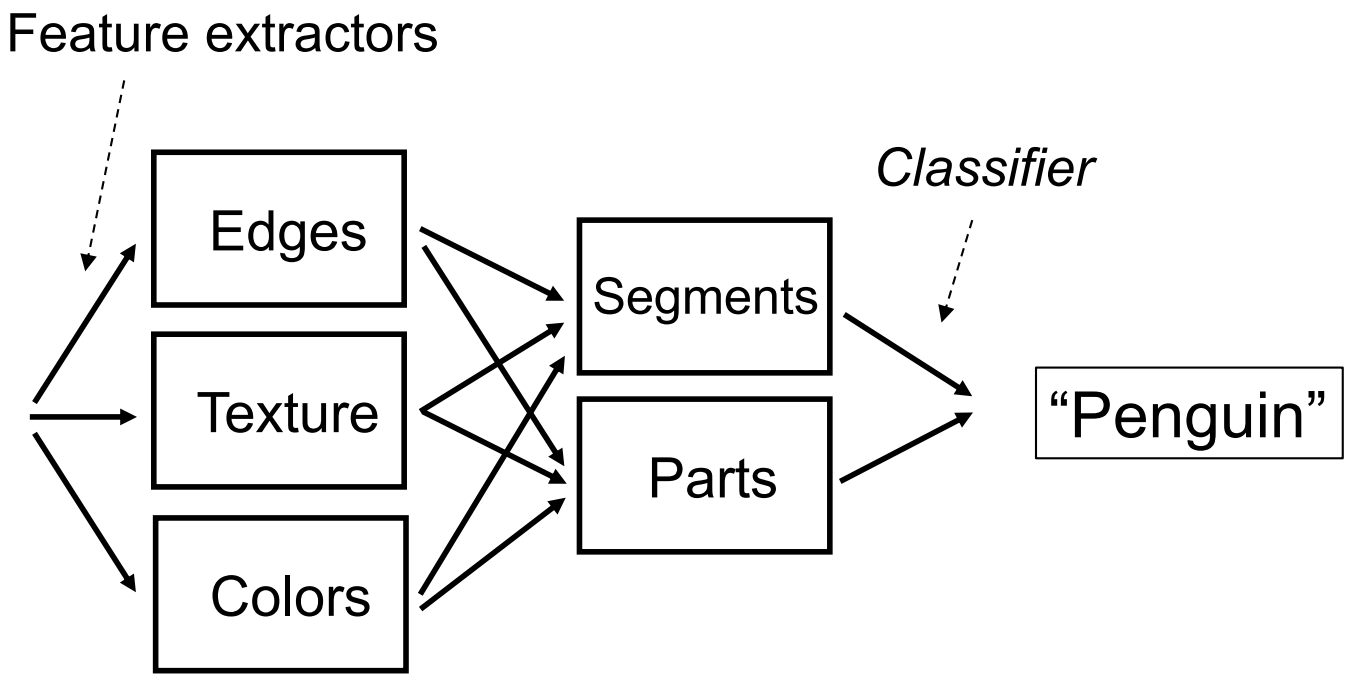
Alexei Efros, UC Berkeley, Fall 2020

# Neural Nets: a particularly useful Black Box



# Classic Object Recognition

image *X*

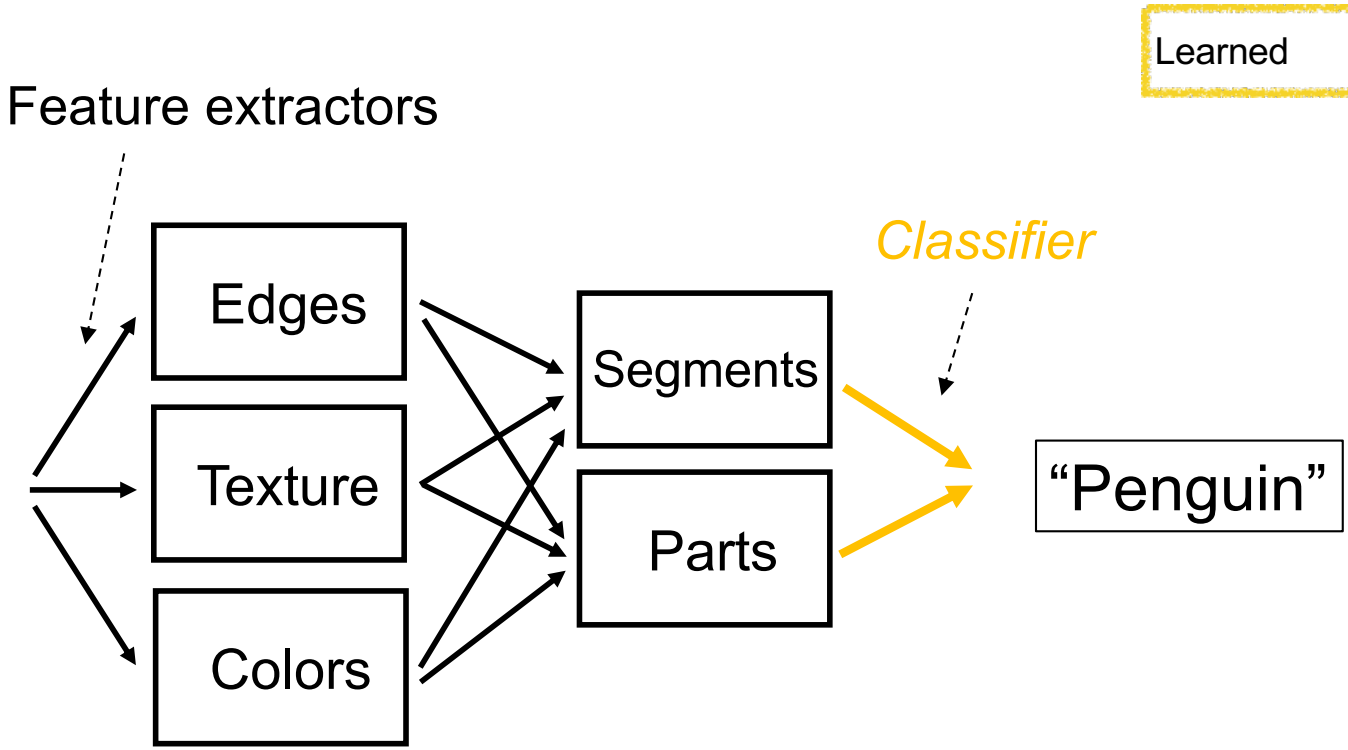



label *Y*

Slide by Philip Isola

# Classic Object Recognition

image *X*




label *Y*

Slide by Philip Isola

# Learning Features

Learned

Feature extractors

image $X$

Edges

Texture

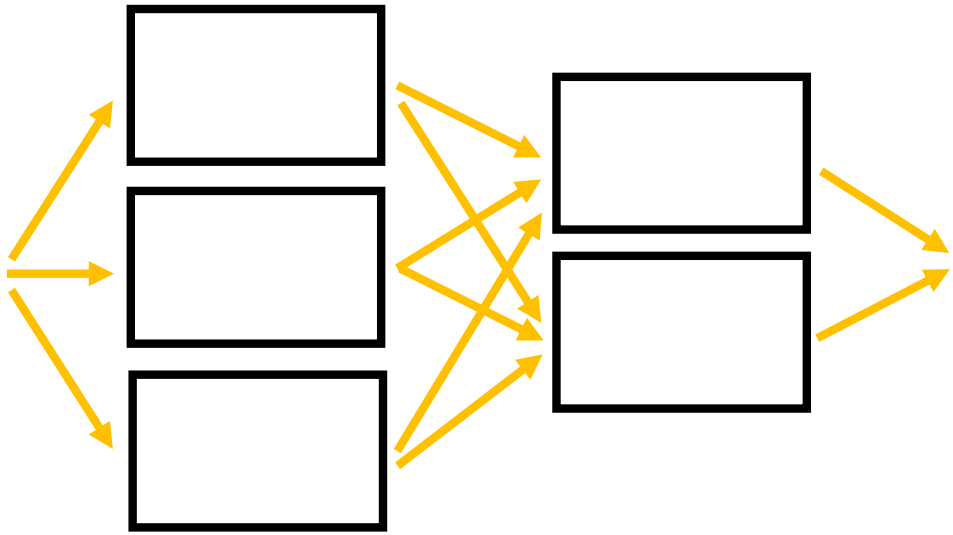
Colors

label $Y$

# Neural Network:
# algorithm + feature + data!

Learned

image *X*

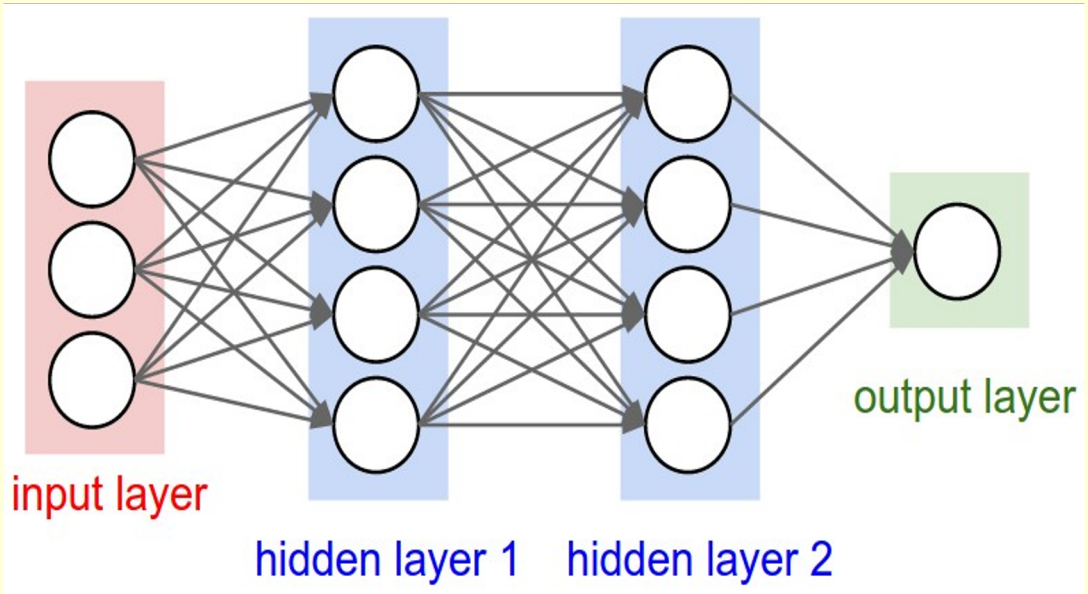


“Penguin”

label *Y*

Slide by Philip Isola

# Vanilla (fully-connected) Neural Networks

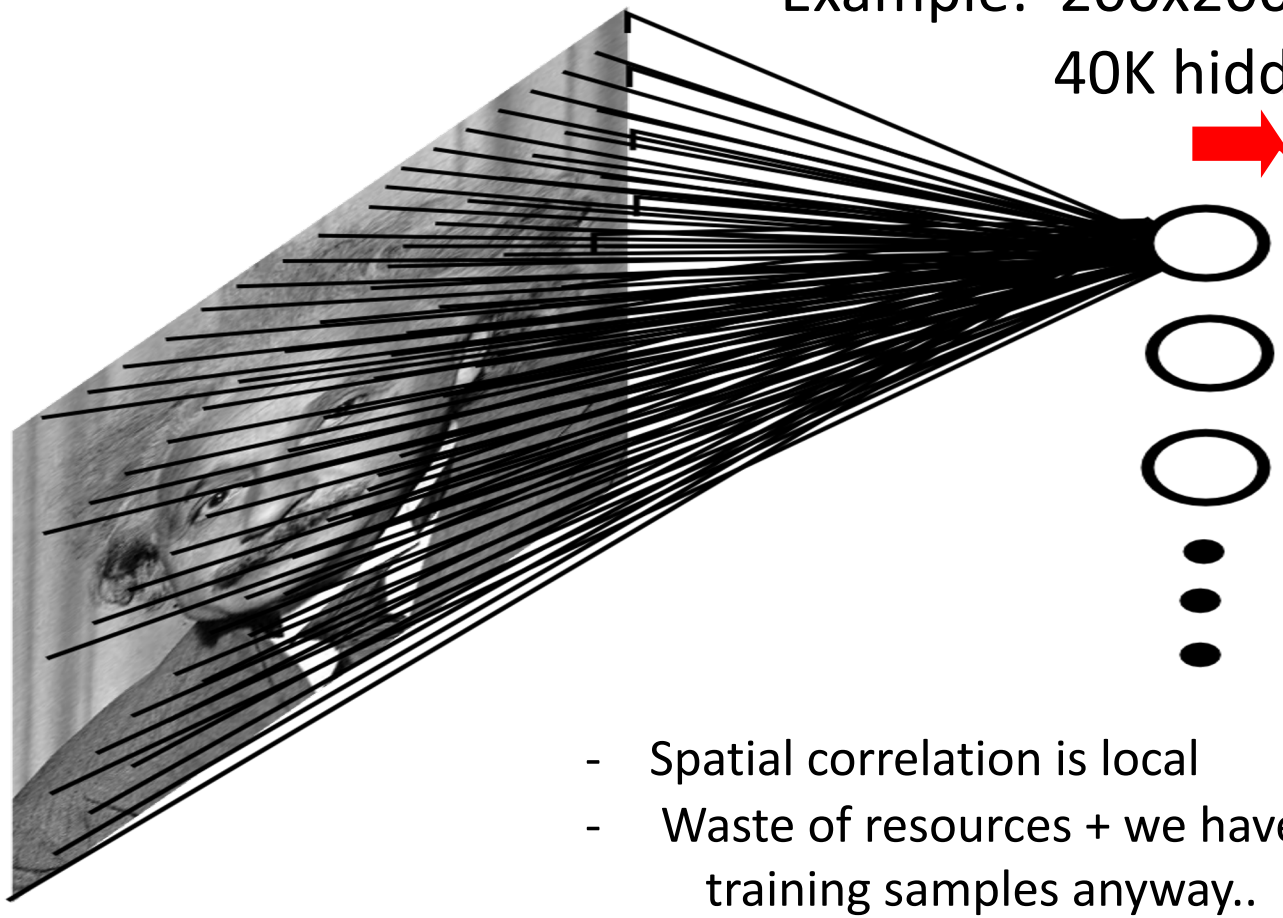


# Fully Connected Layer

Example: 200x200 image

40K hidden units

**2B parameters**!!!

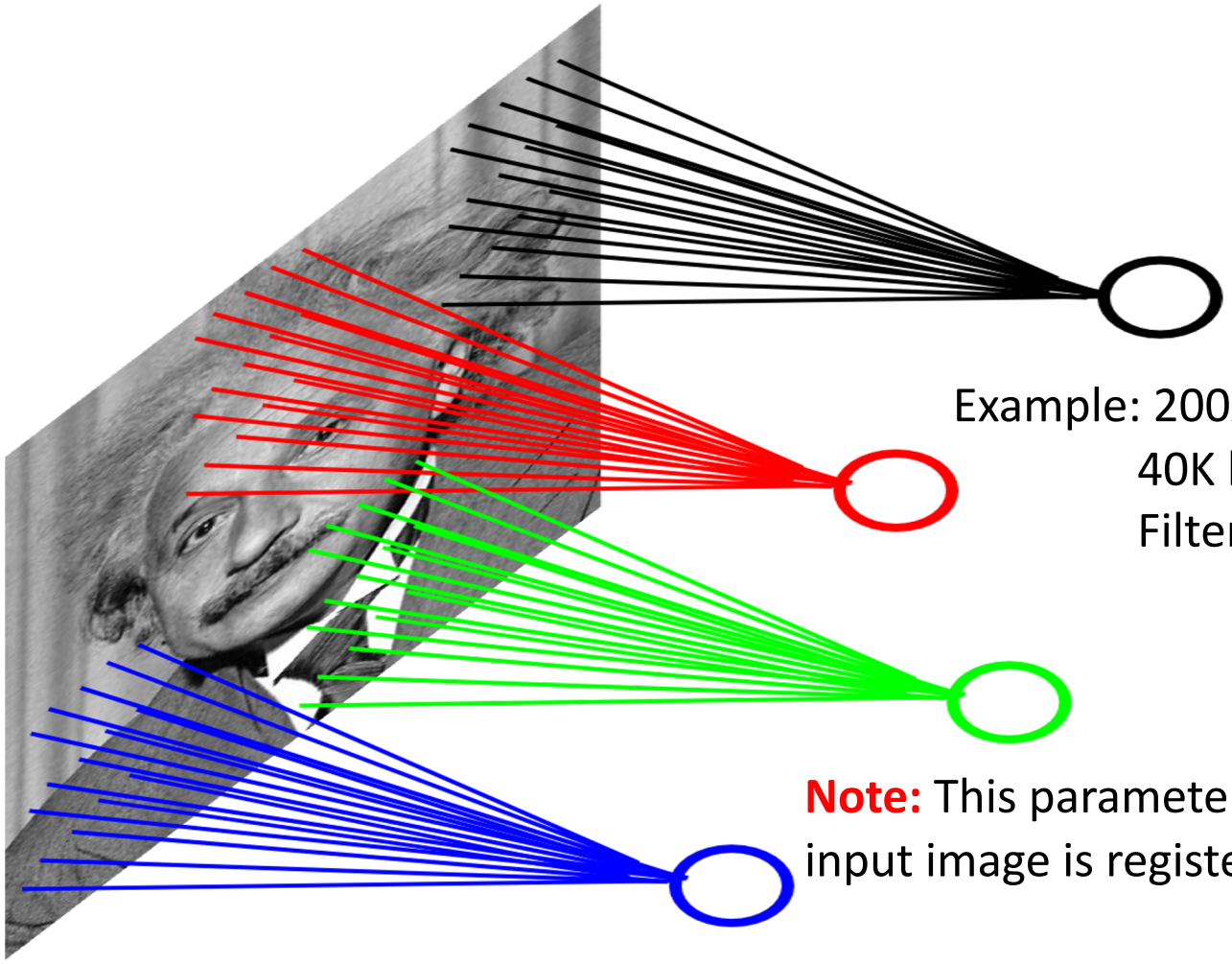


- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..



Ranzato

# Locally Connected Layer



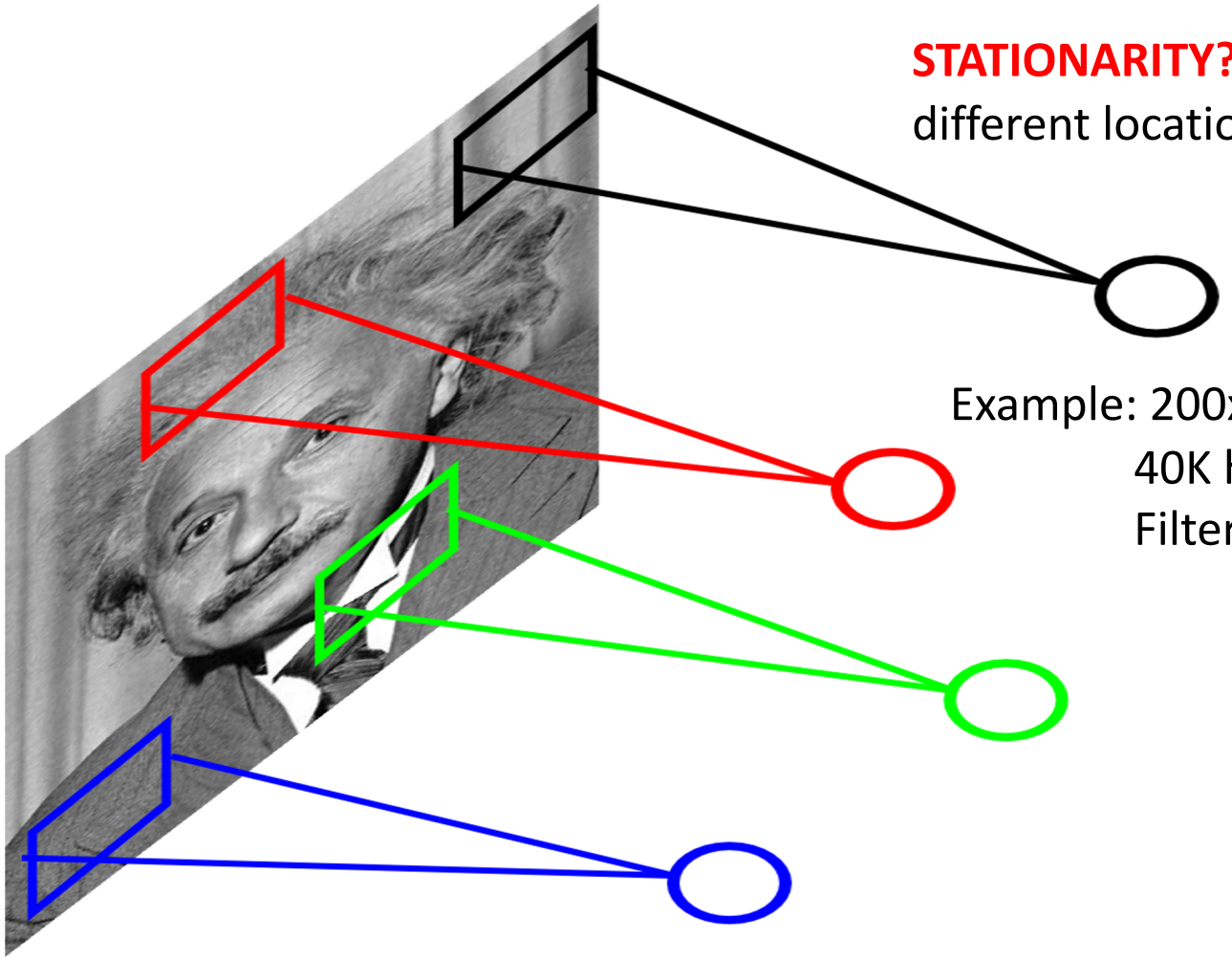
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

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



Ranzato

# Locally Connected Layer



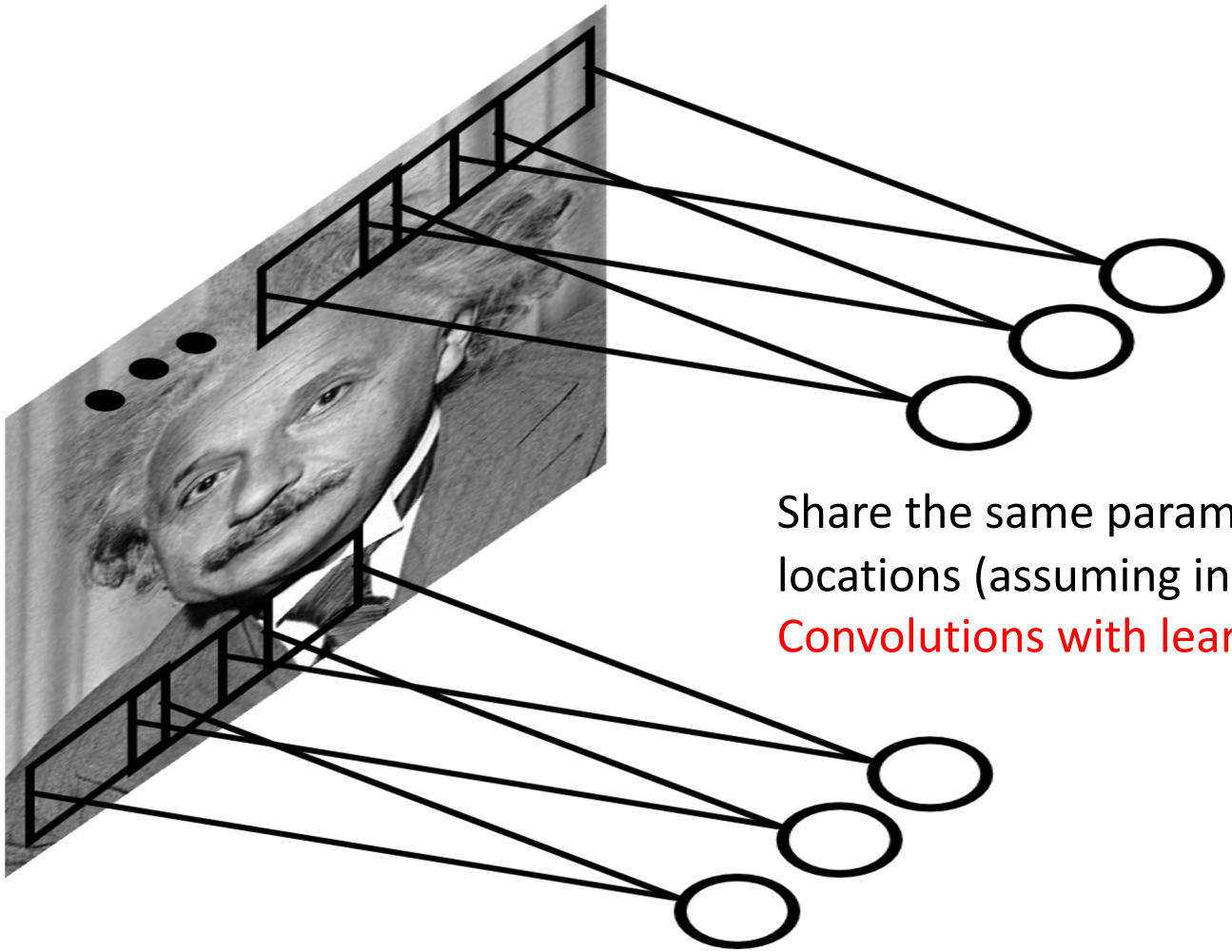
**STATIONARITY?** Statistics is similar at different locations

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



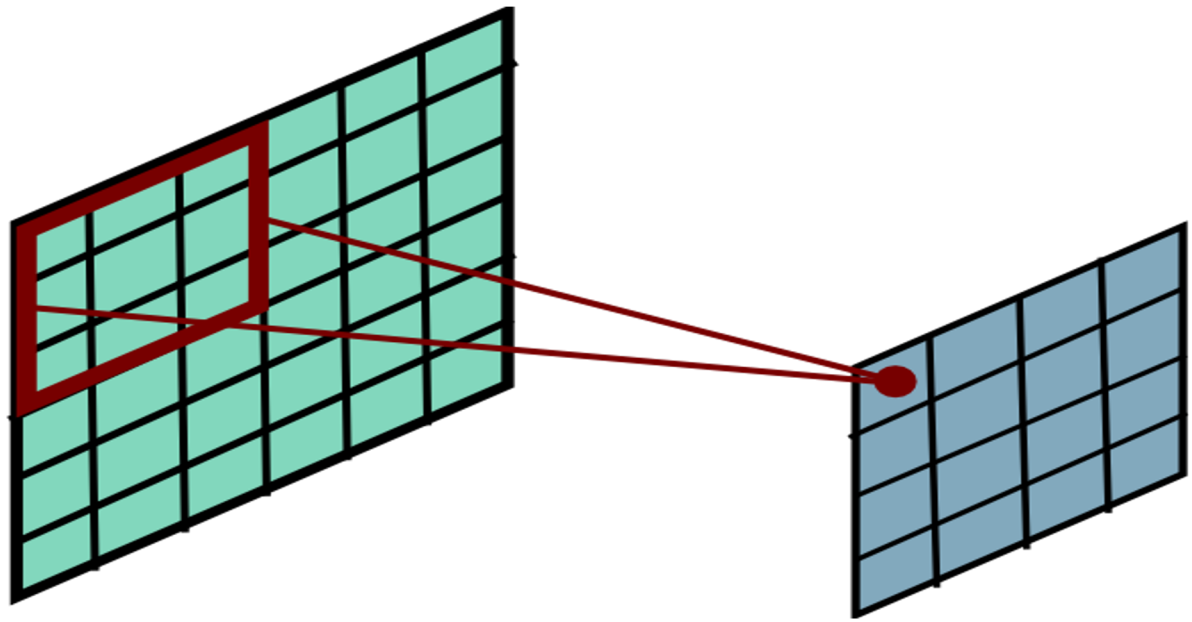
Ranzato

# Convolutional Layer



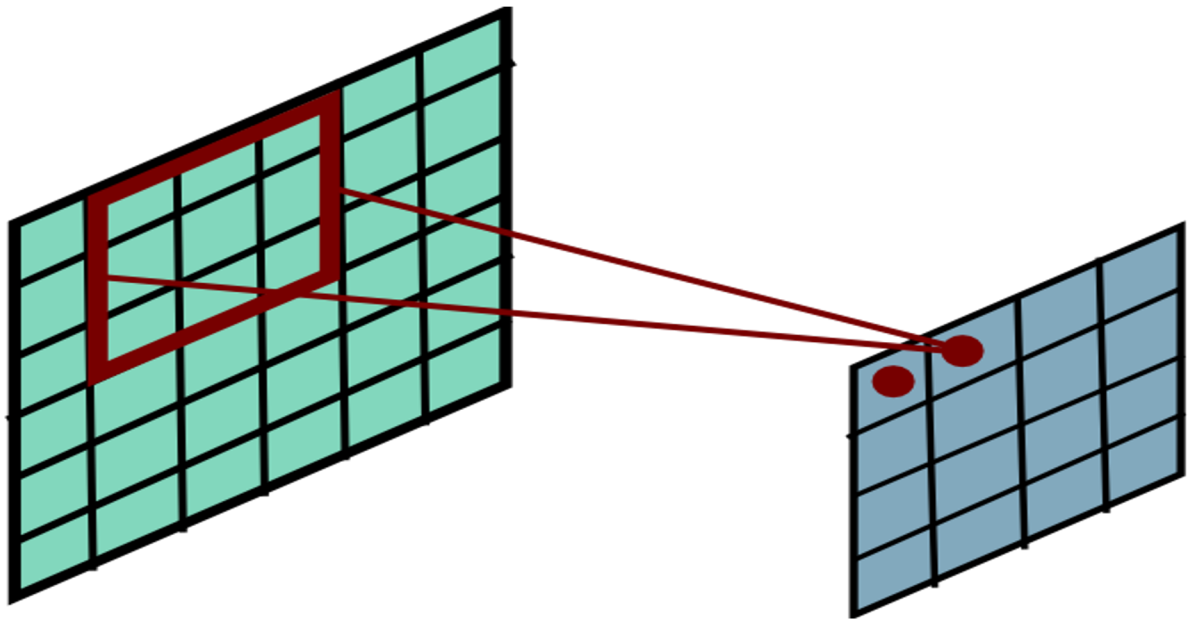
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

# Convolutional Layer



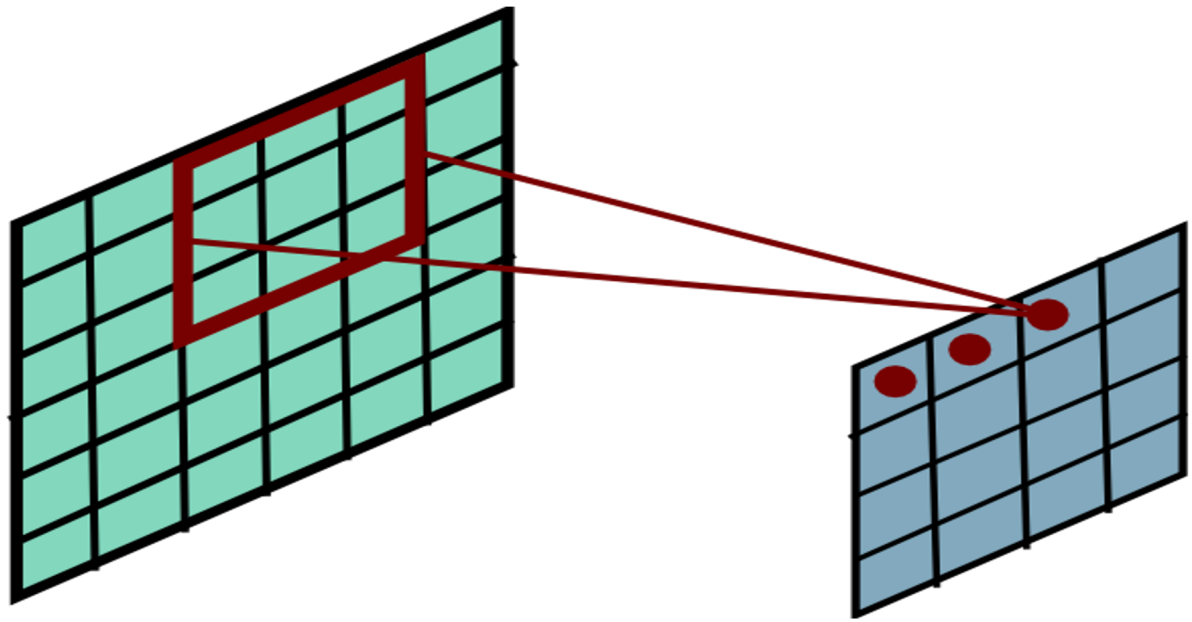
Ranzato

# Convolutional Layer



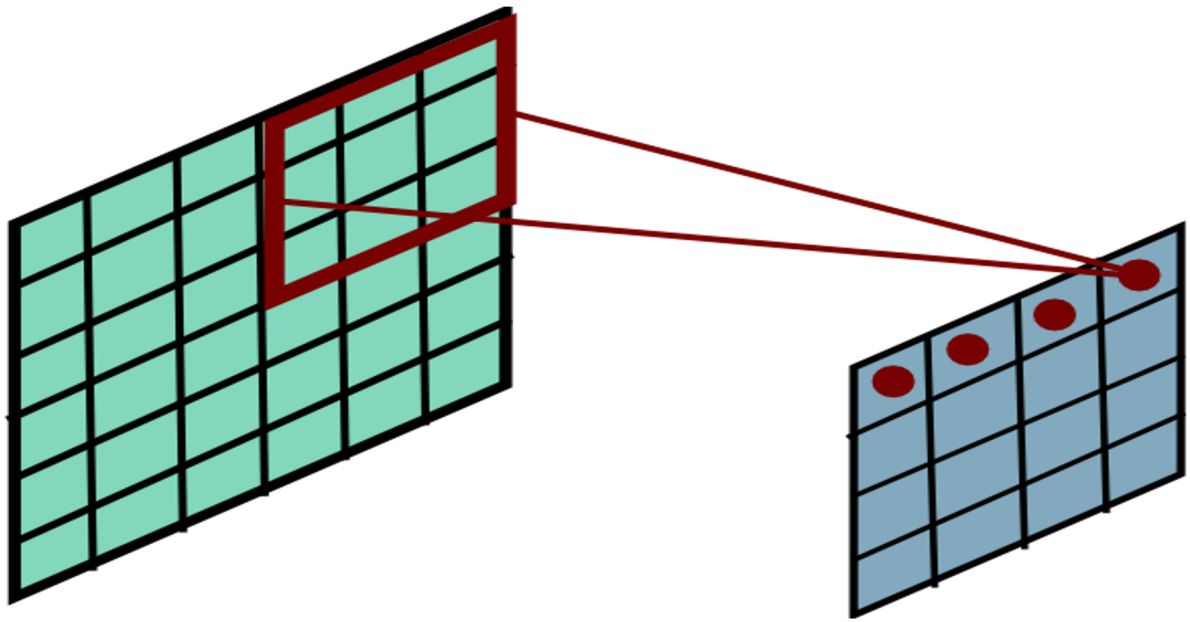
Ranzato

# Convolutional Layer



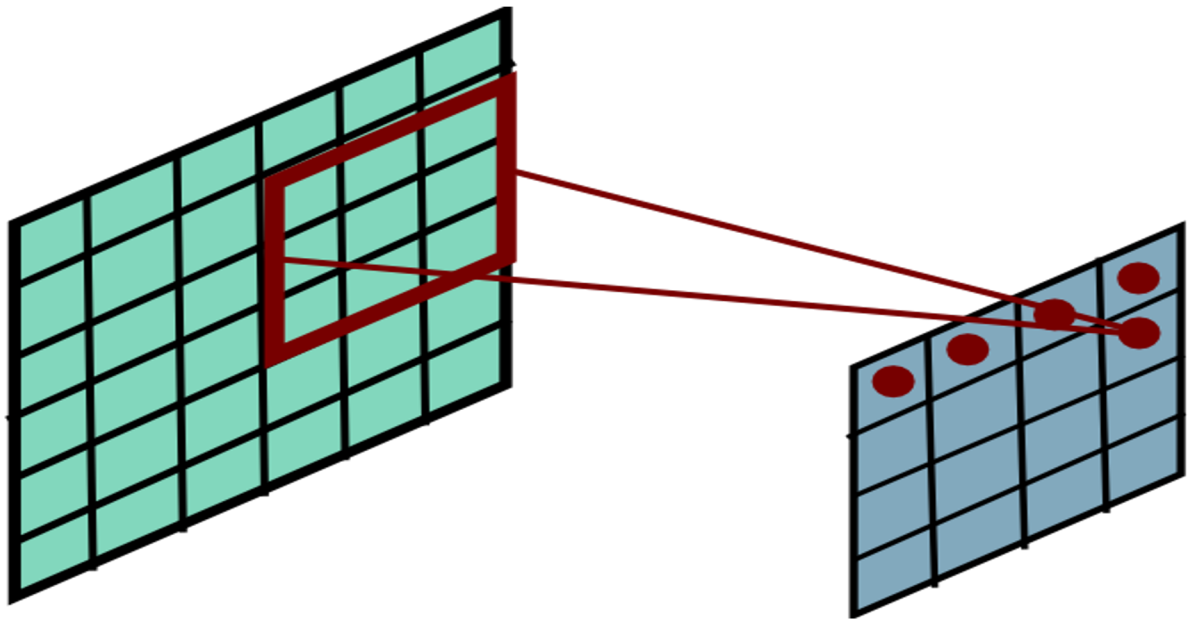
Ranzato

# Convolutional Layer



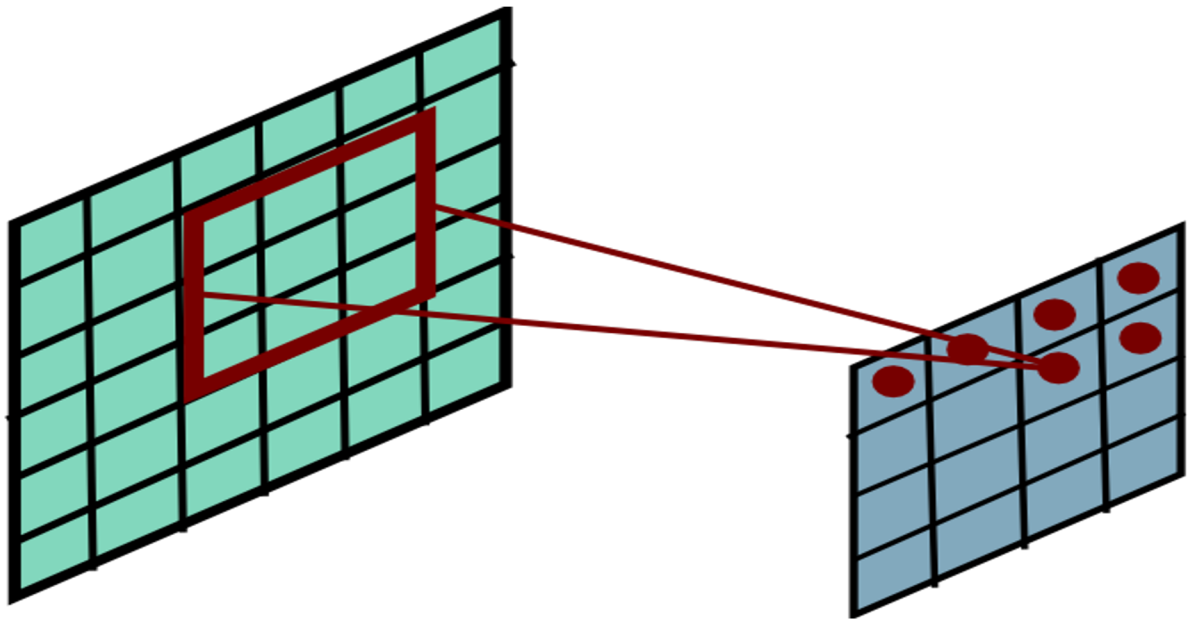
Ranzato

# Convolutional Layer



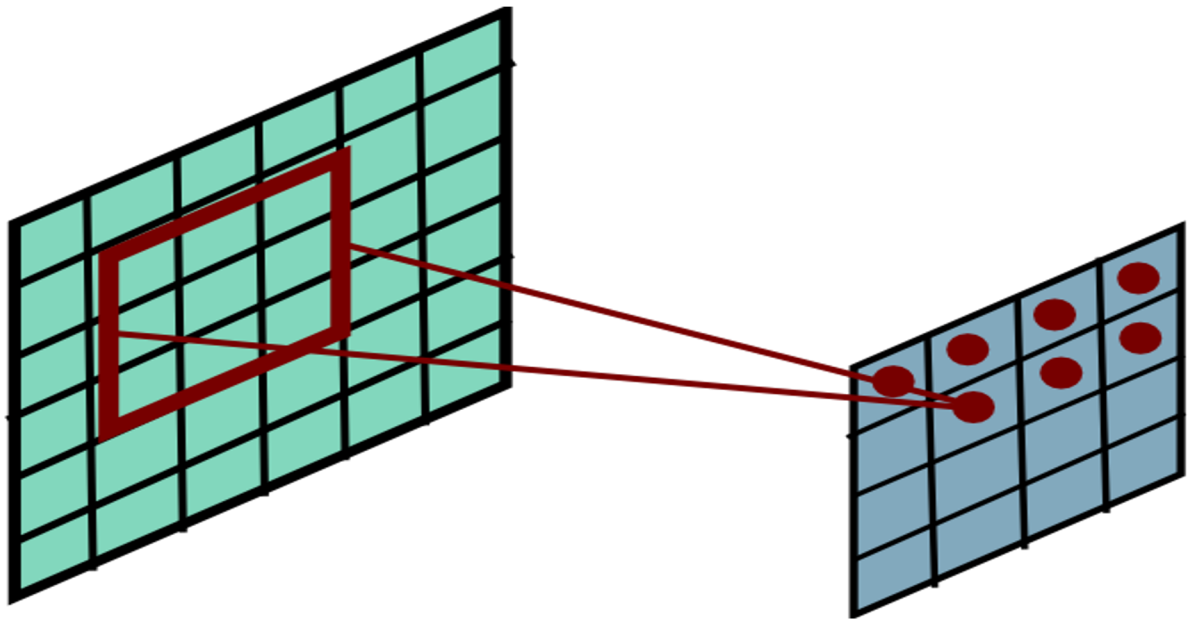
Ranzato

# Convolutional Layer



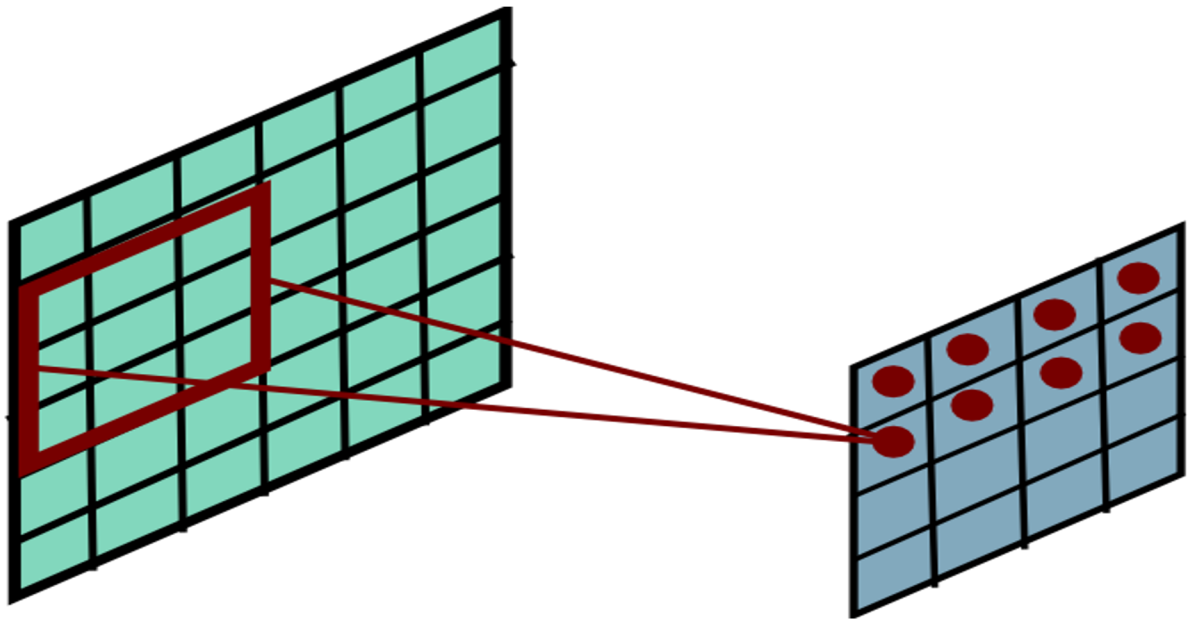
Ranzato

# Convolutional Layer



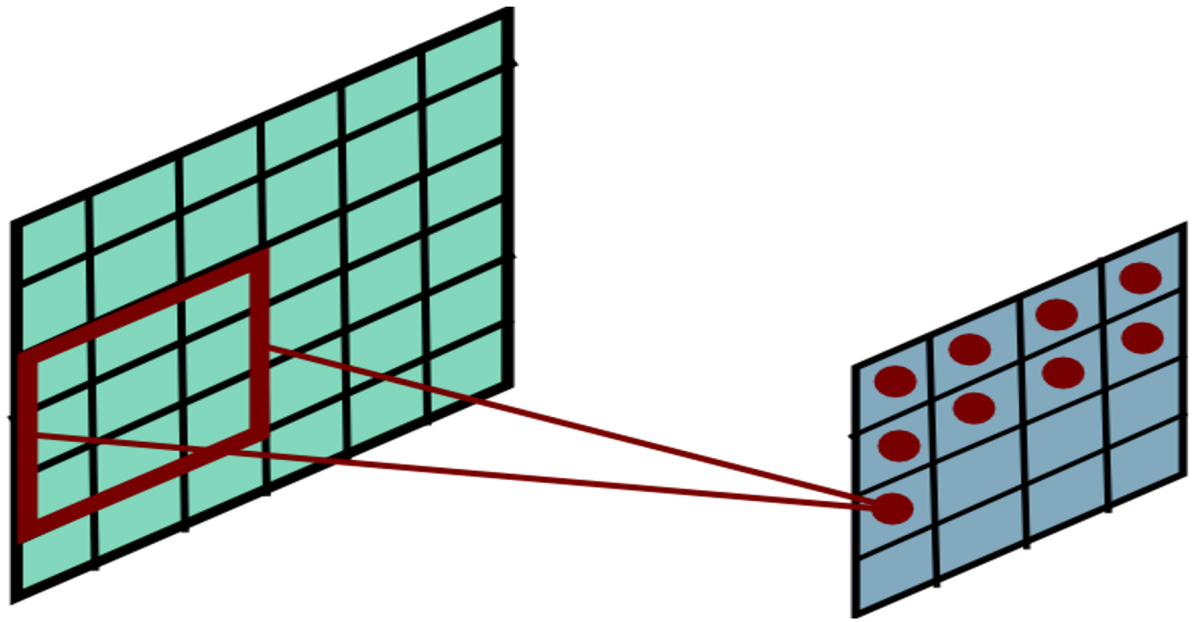
Ranzato

# Convolutional Layer



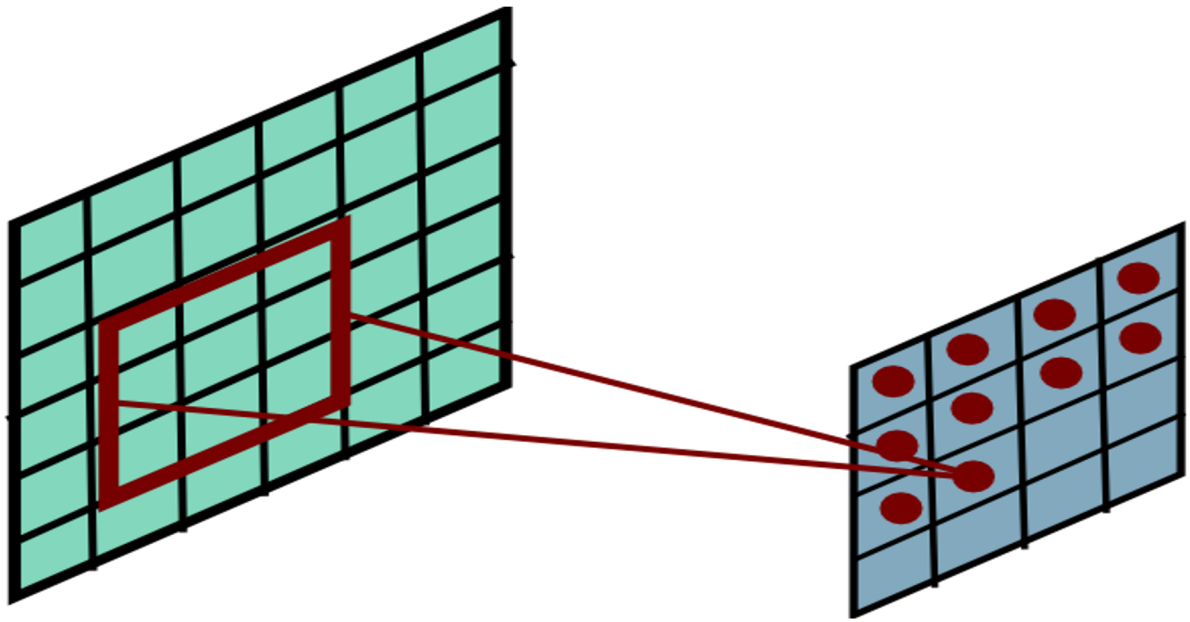
Ranzato

# Convolutional Layer



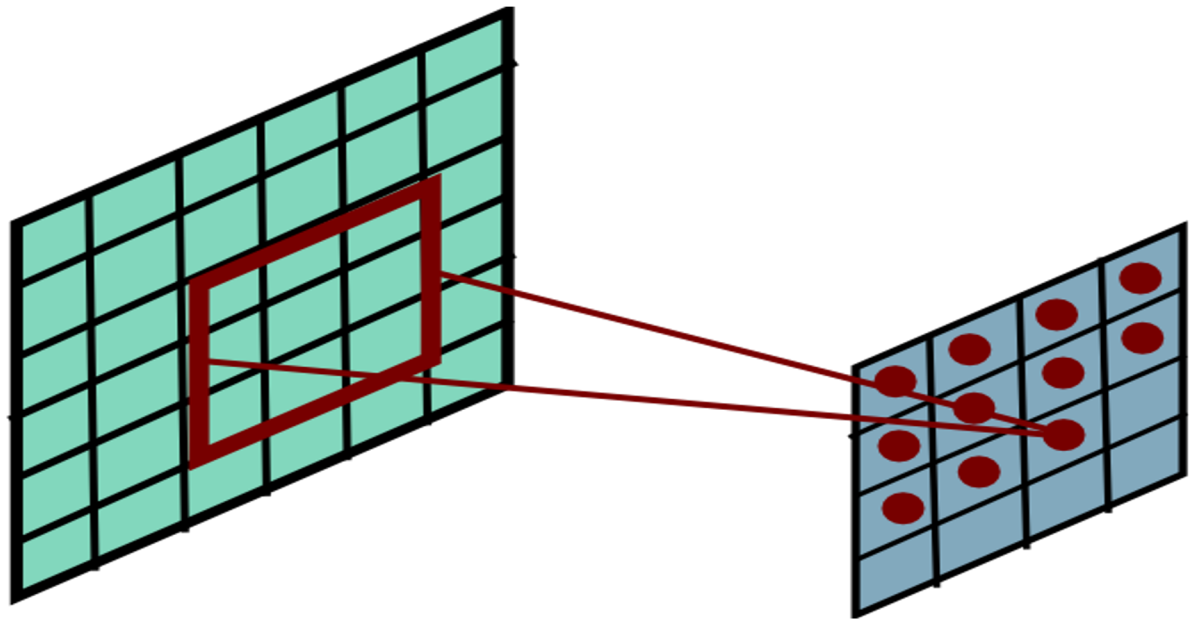
Ranzato

# Convolutional Layer



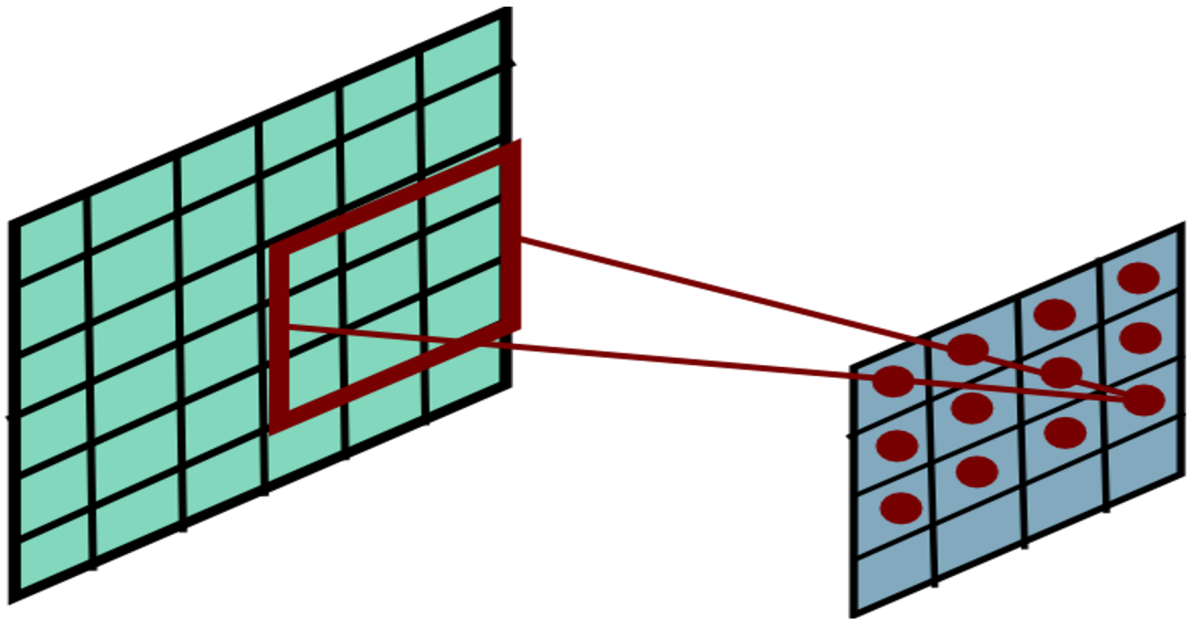
Ranzato

# Convolutional Layer



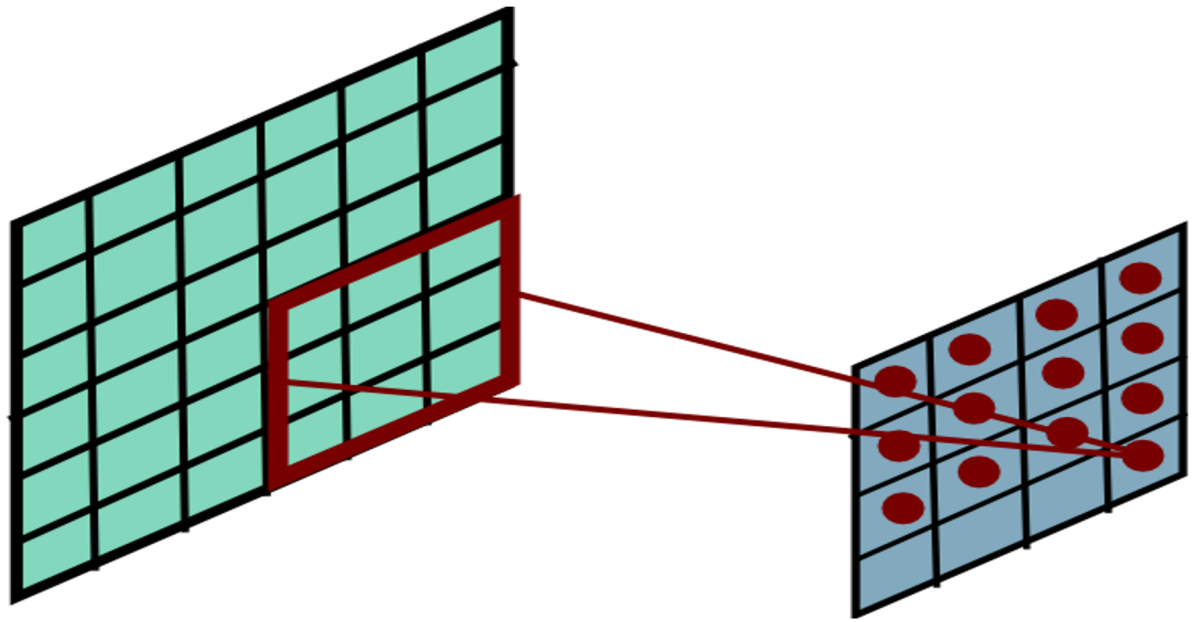
Ranzato

# Convolutional Layer



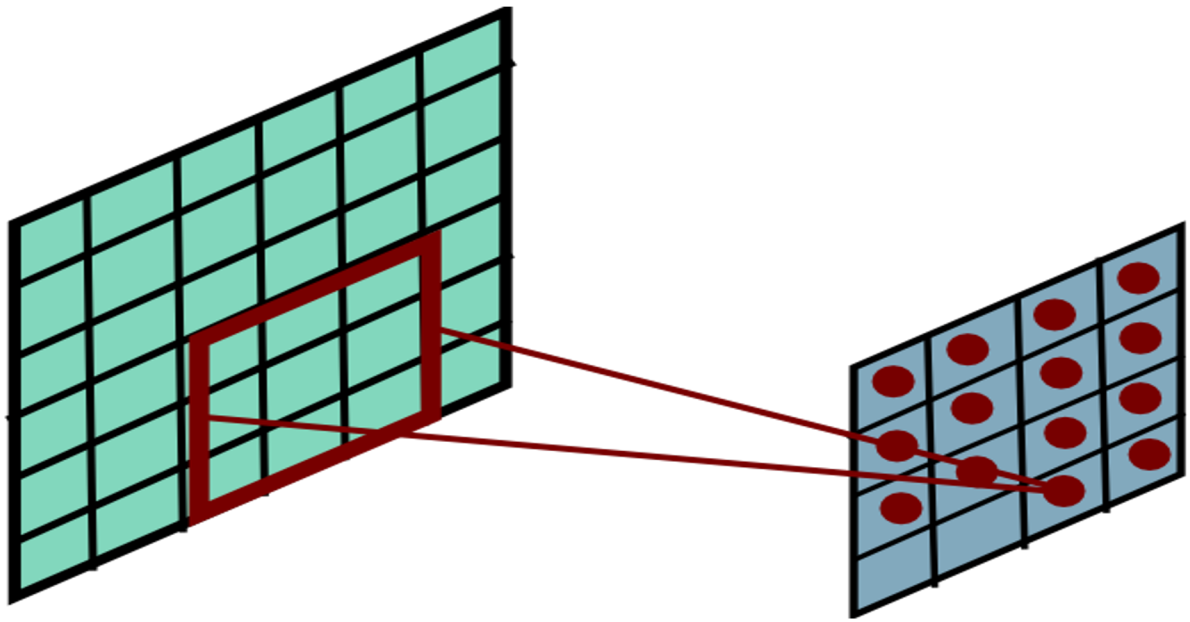
Ranzato

# Convolutional Layer



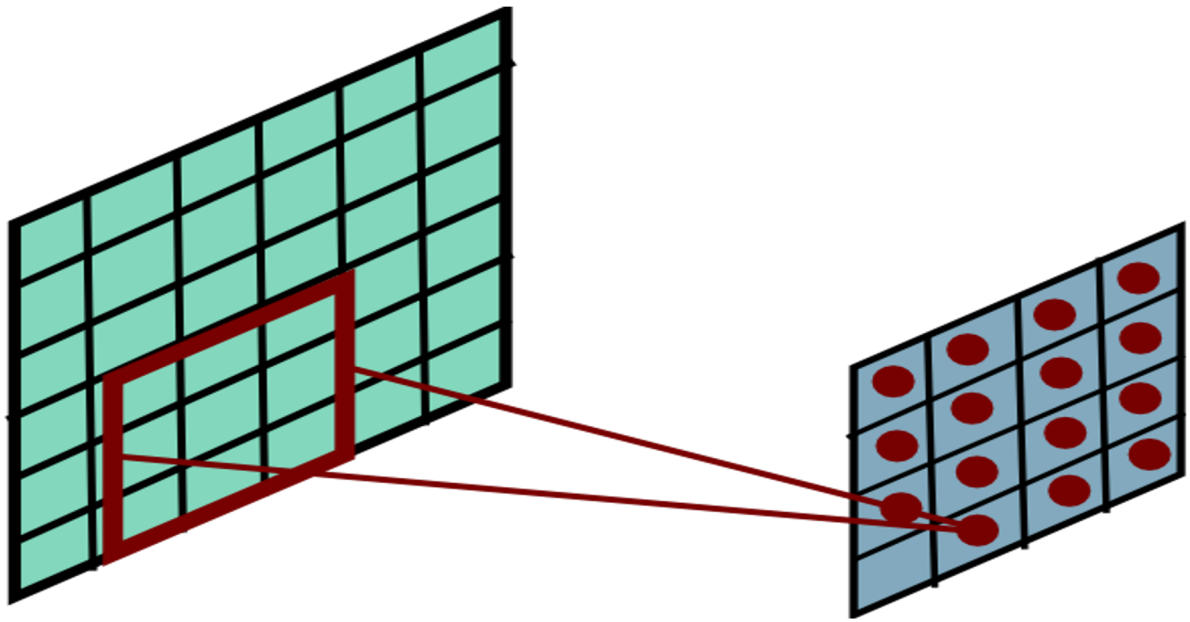
Ranzato

# Convolutional Layer



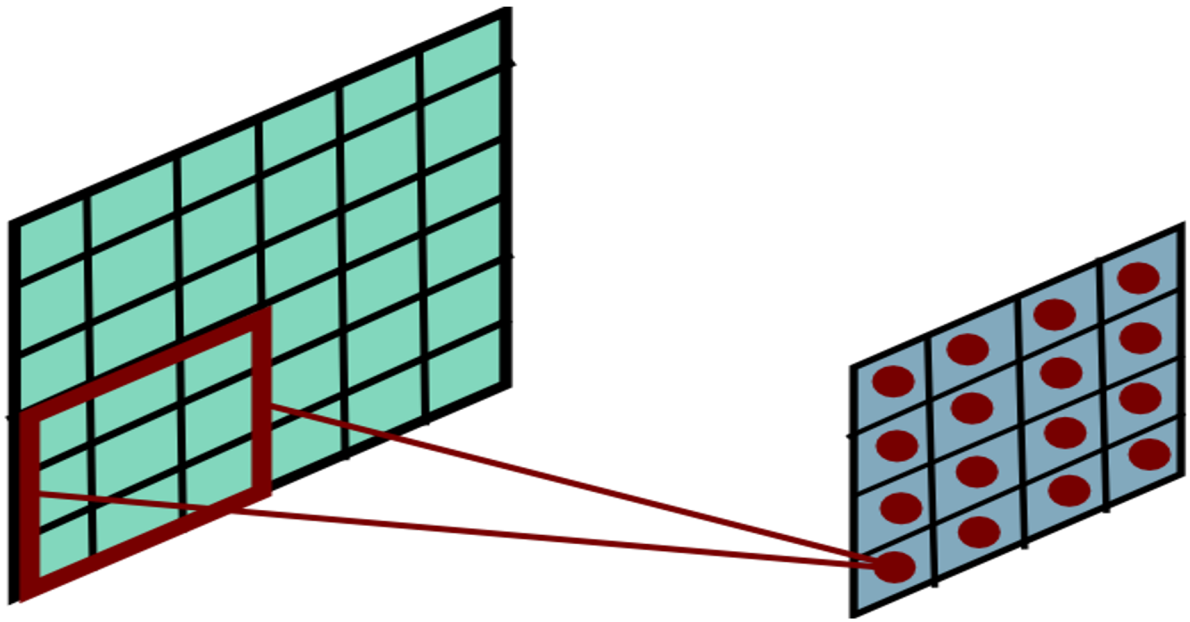
Ranzato

# Convolutional Layer



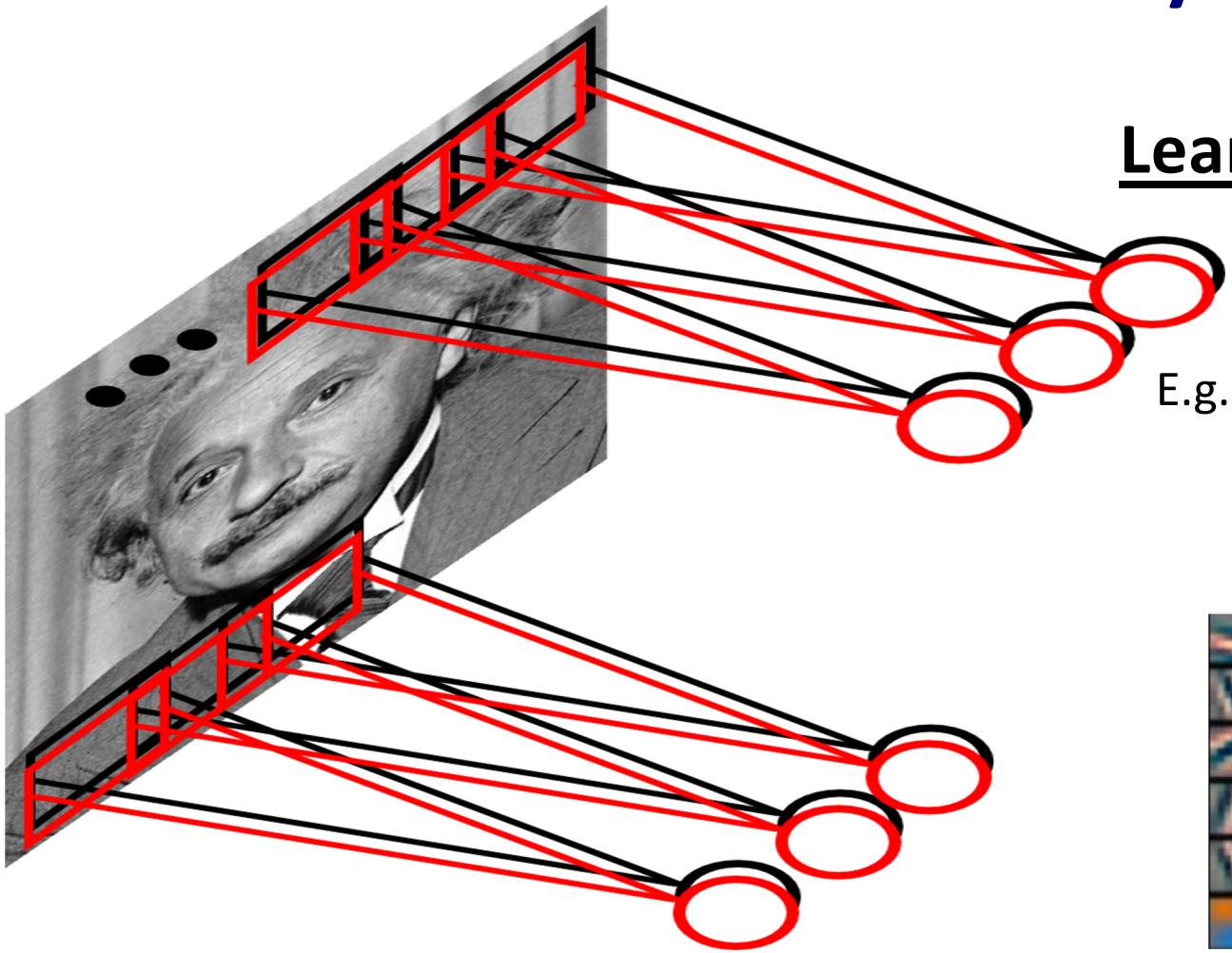
Ranzato

# Convolutional Layer



Ranzato

# Convolutional Layer



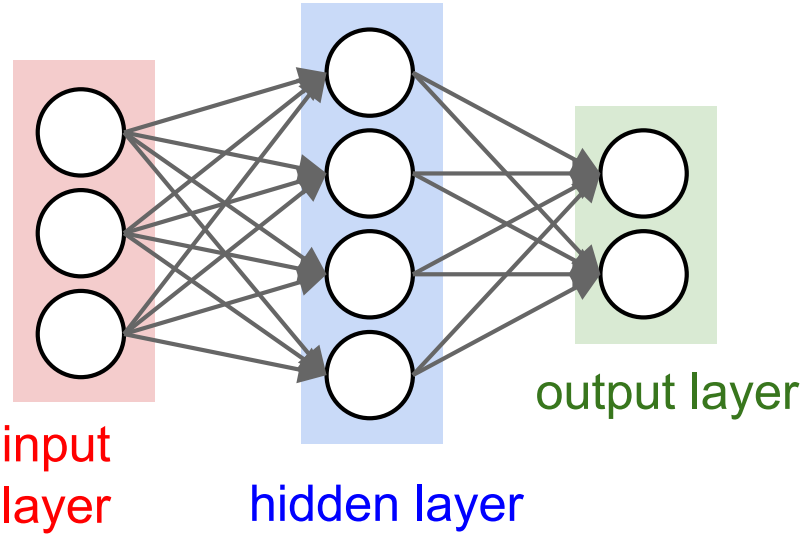
**Learn** multiple filters.

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

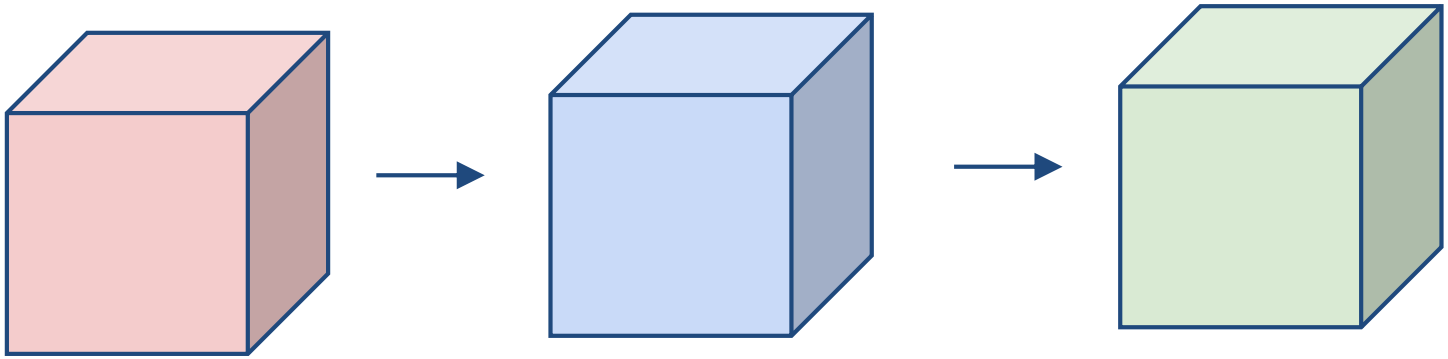


Ranzato

before:

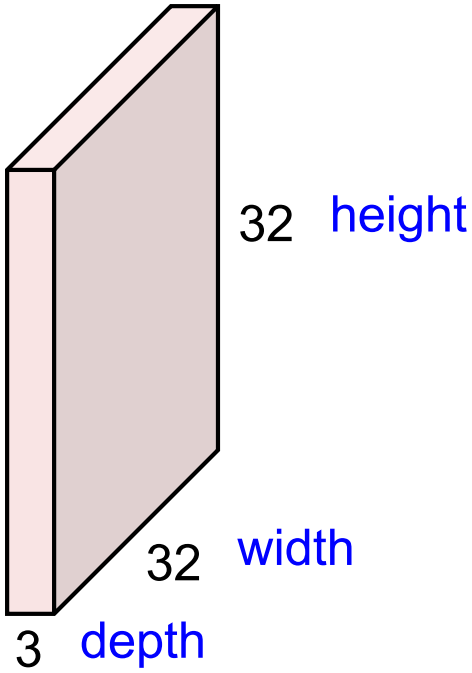



now:



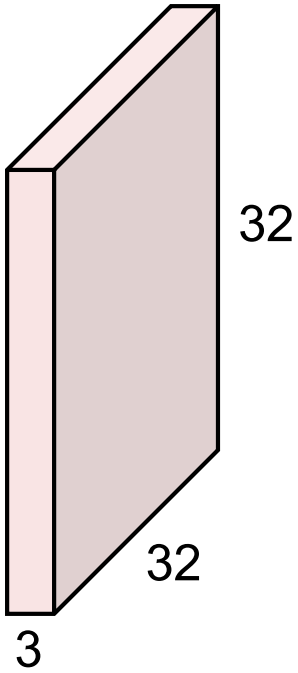
# Convolution Layer

32x32x3 image

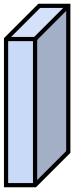


# Convolution Layer

32x32x3 image




5x5x3 filter

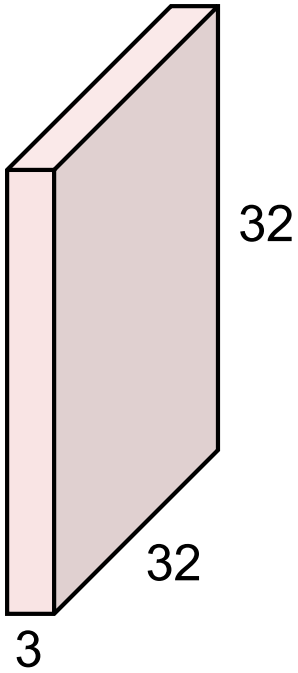


**Convolve** the filter with the image
i.e. “slide over the image spatially, computing dot products”

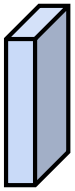
# Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

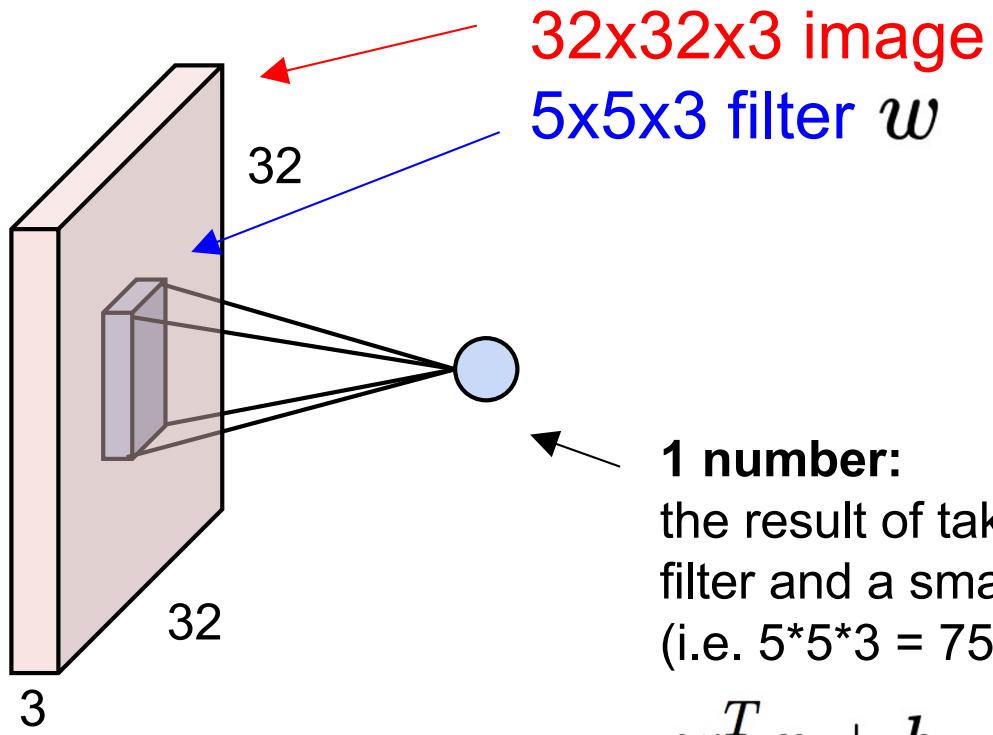



5x5x3 filter



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

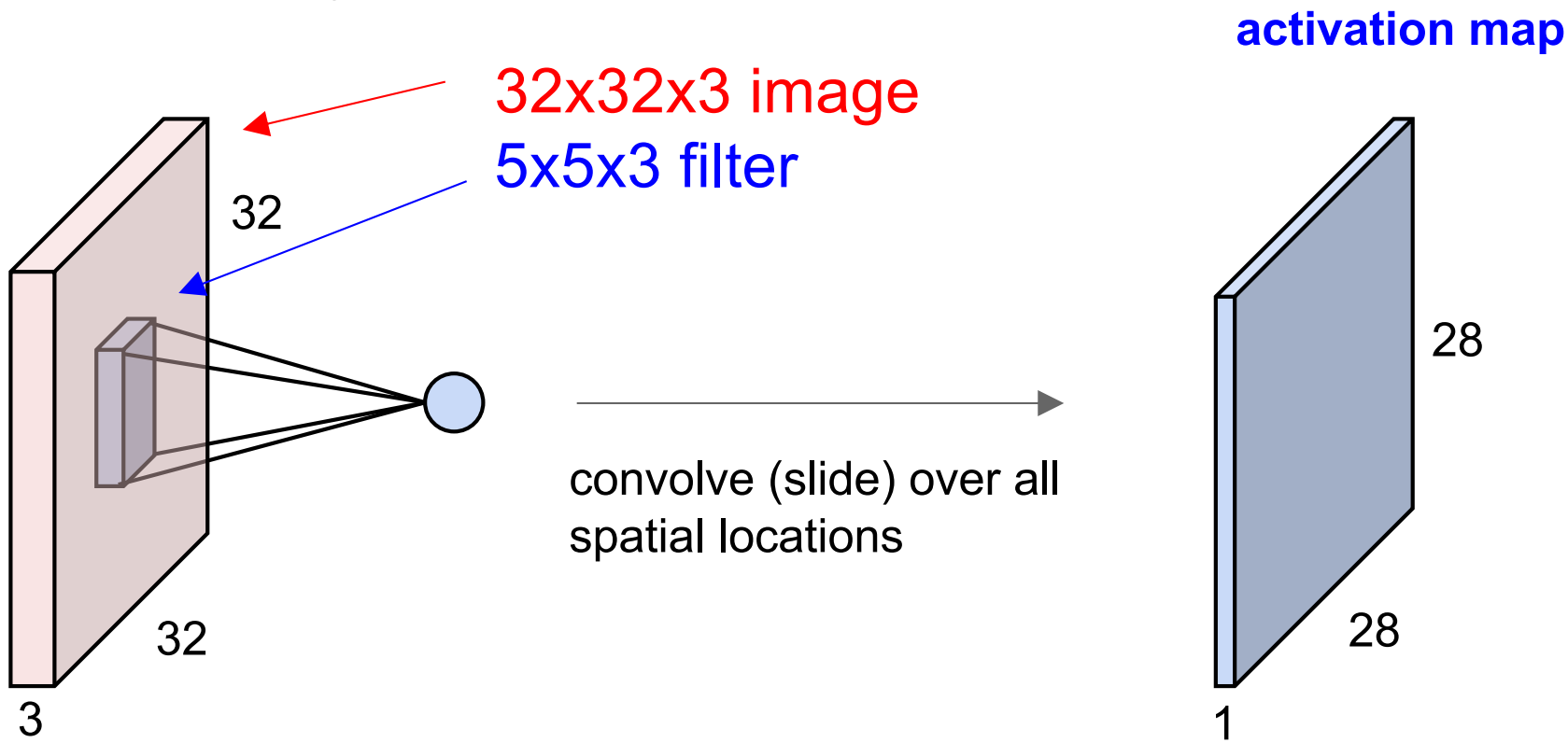
# Convolution Layer




**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)

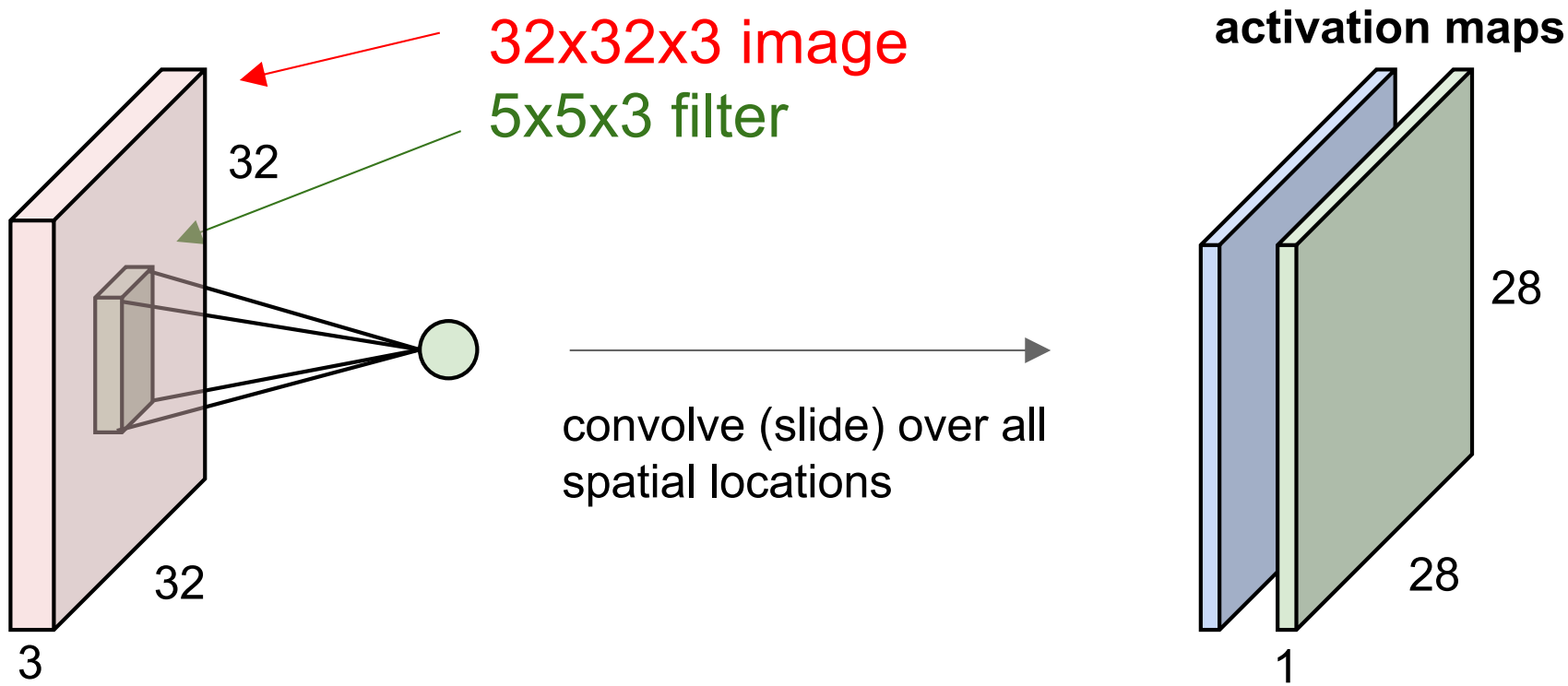
$$w^T x + b$$

# Convolution Layer

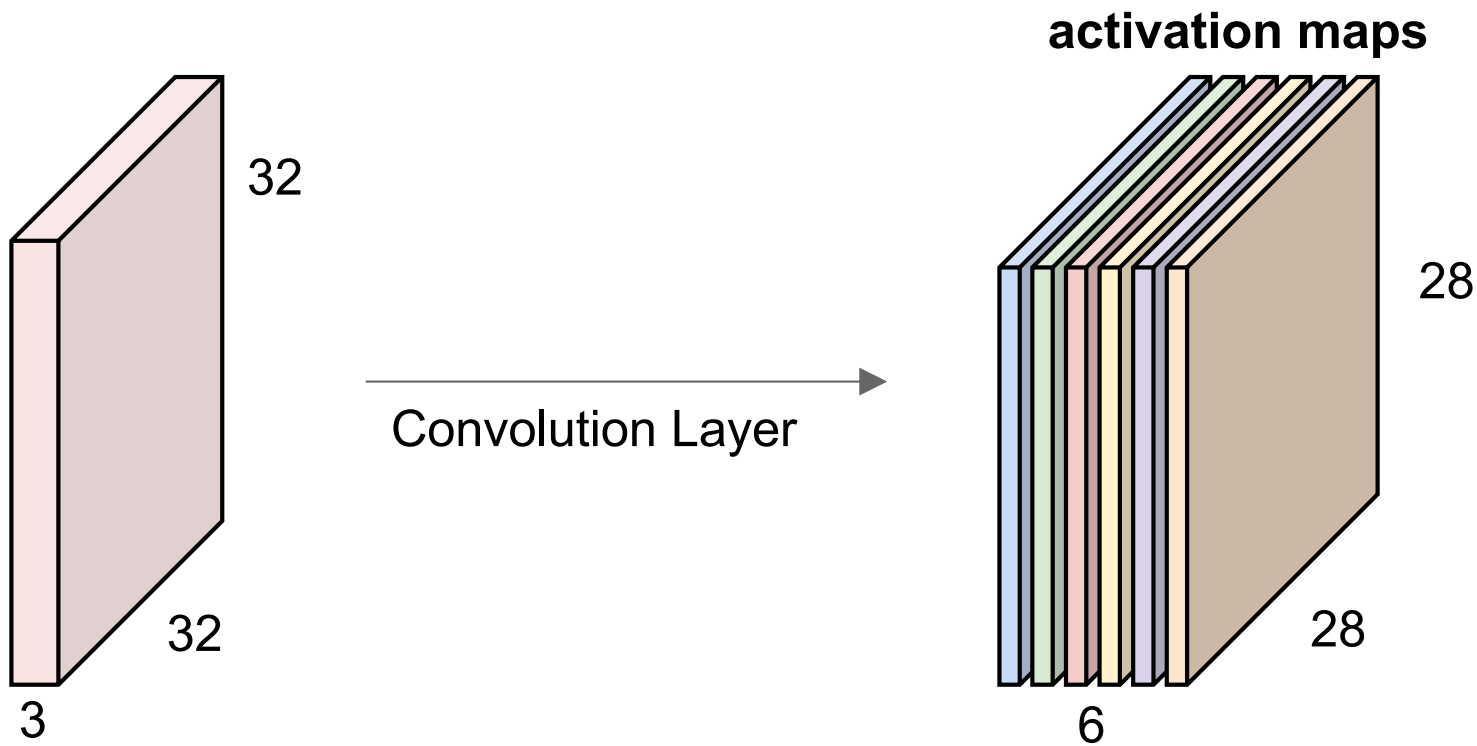


# Convolution Layer

consider a second, green filter



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!