Feature Learning with Neural Networks

(aka Deep Learning)

CS194: Intro to Computer Vision and Comp. Photo

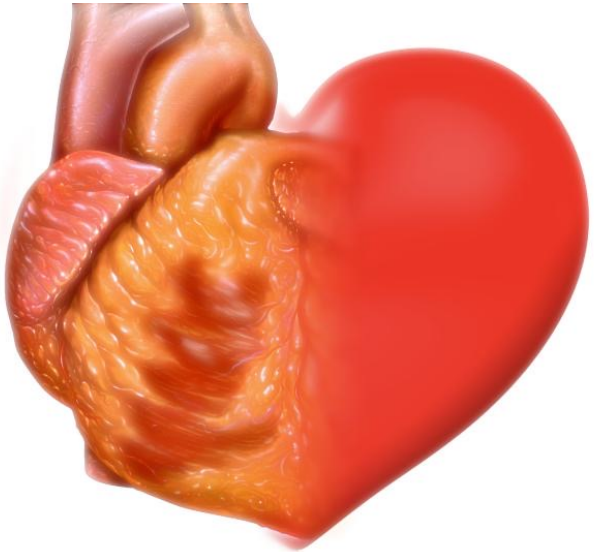
Angjoo Kanazawa & Alexei Efros, UC Berkeley, Fall 2022

project page

Proj 1 class choice award winner!!

Proj 2 highlights: results next class

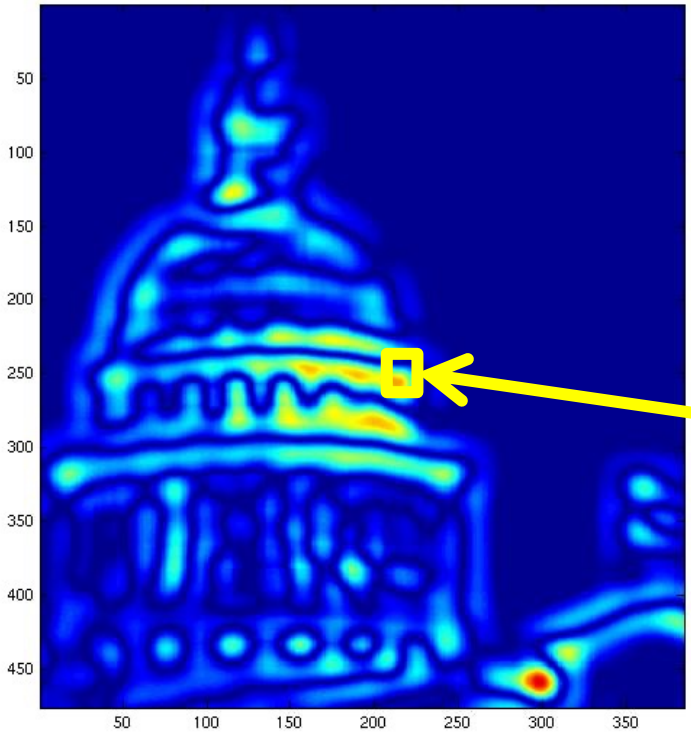
Andrew Zhang

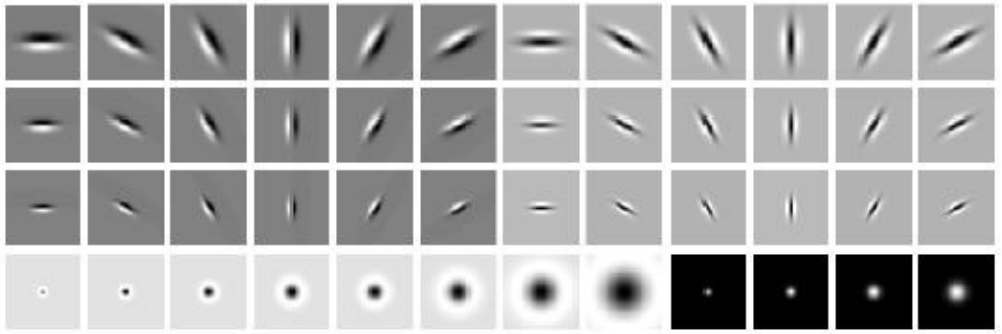


Kayla Kranen

Huibo Yang

Recap: Filter Bank Response





[r1, r2, …, r38]

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

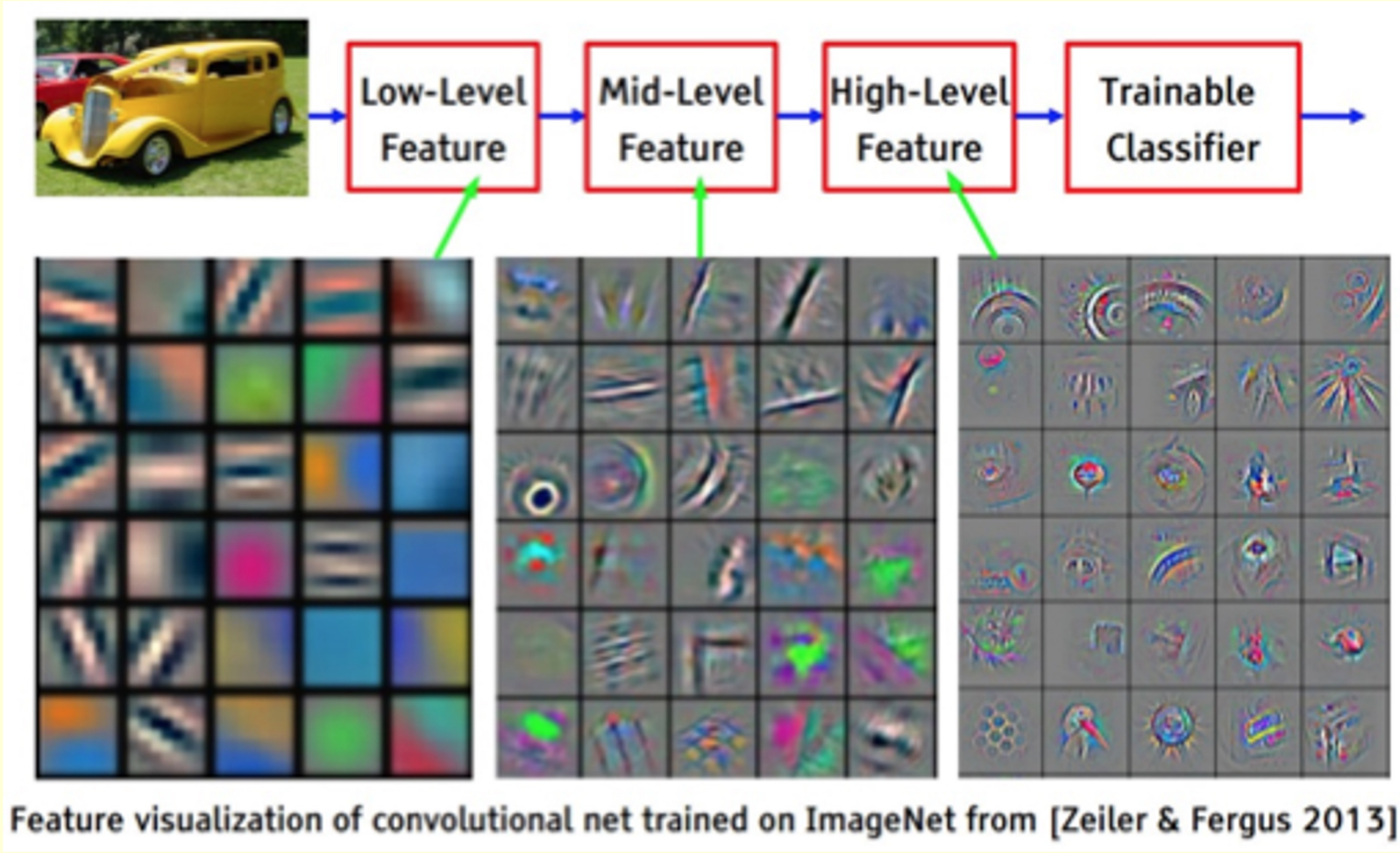
The story so far

- 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]

How to learn?

- To learn useful features, we first need a task!

Quick Background on the Statistical Learning Framework

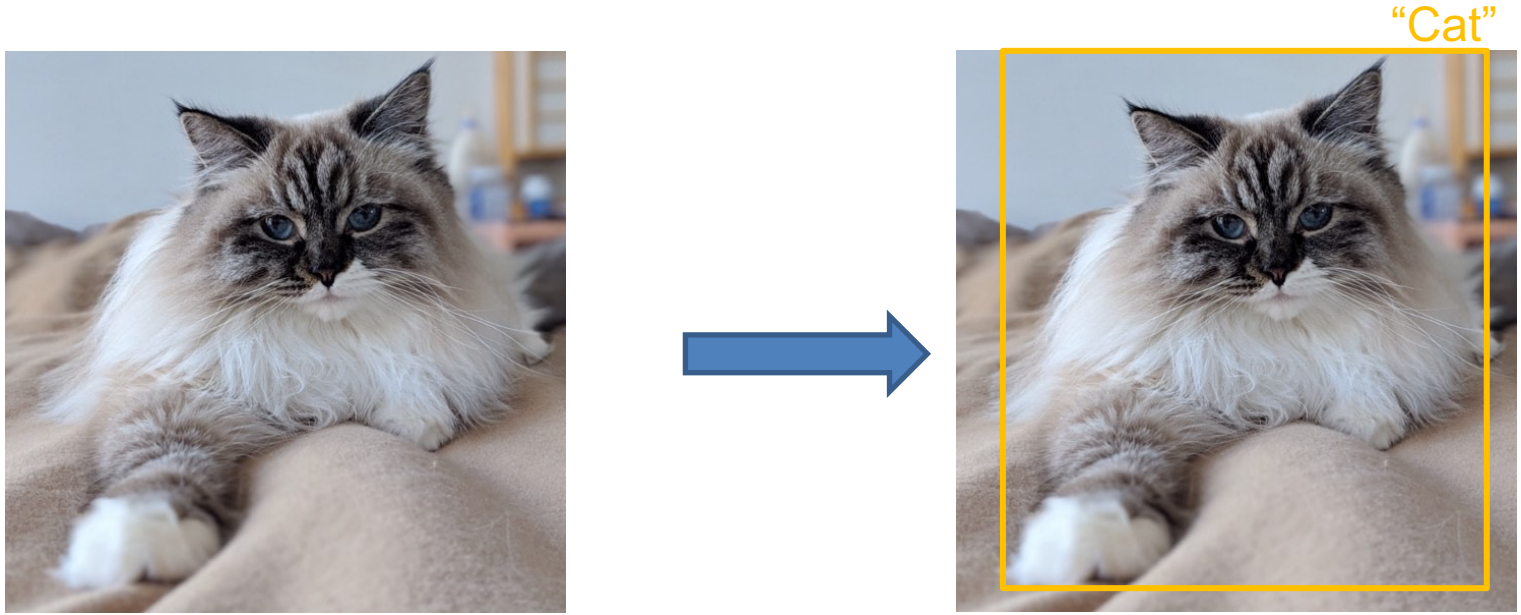
Slides courtesy of Lana Lezebnik

Common Tasks

”Cat”

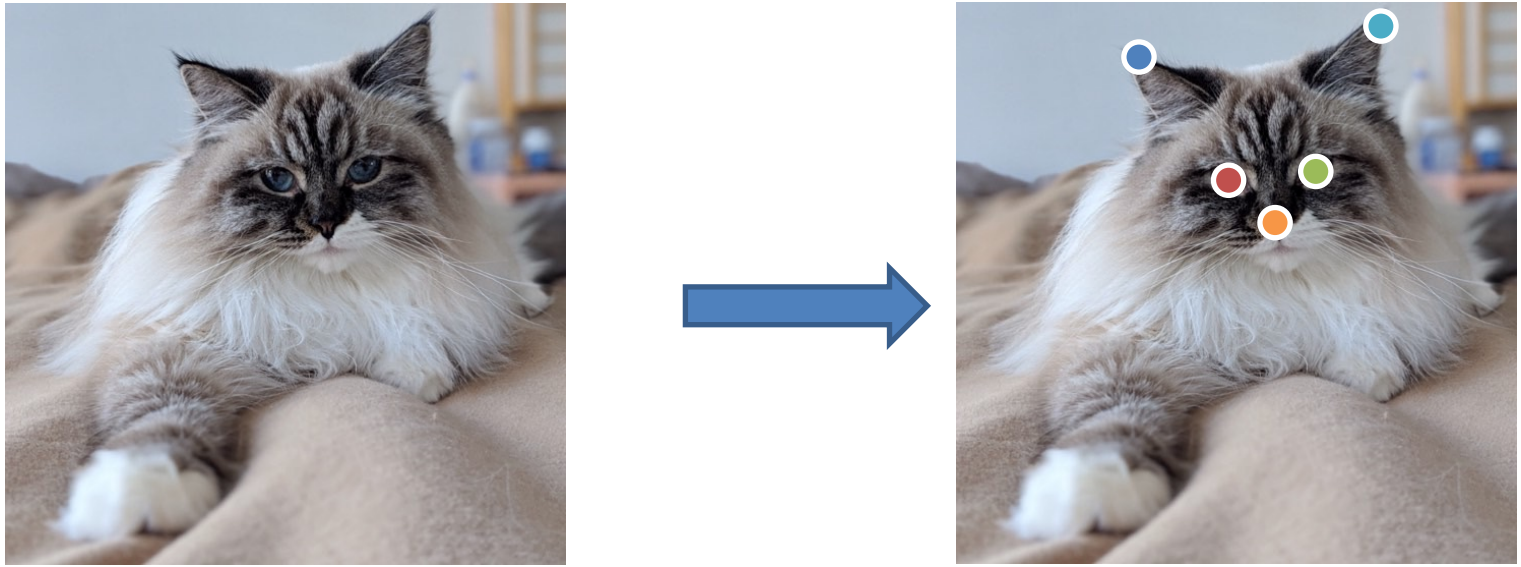
Image Classification

Common Tasks



Object Detection

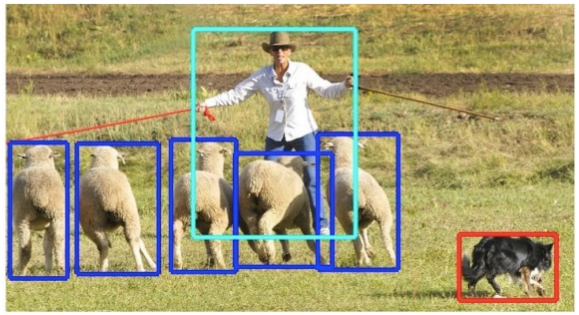
Common Tasks



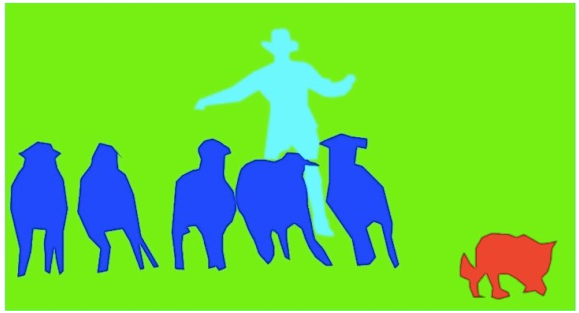
Keypoint Detection (project 5!)

Detection, semantic segmentation, instance segmentation

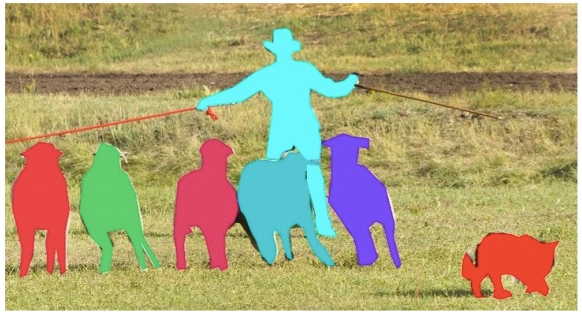
image classification



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:

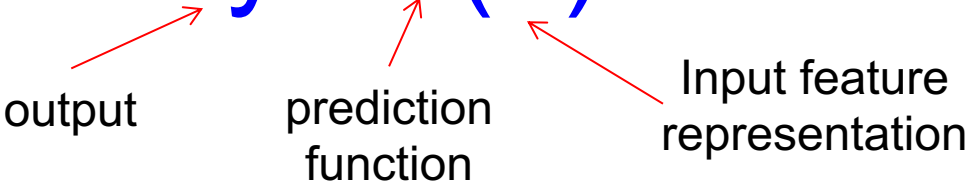
f() = “apple”

f() = “tomato”

f() = “cow”

The statistical learning framework

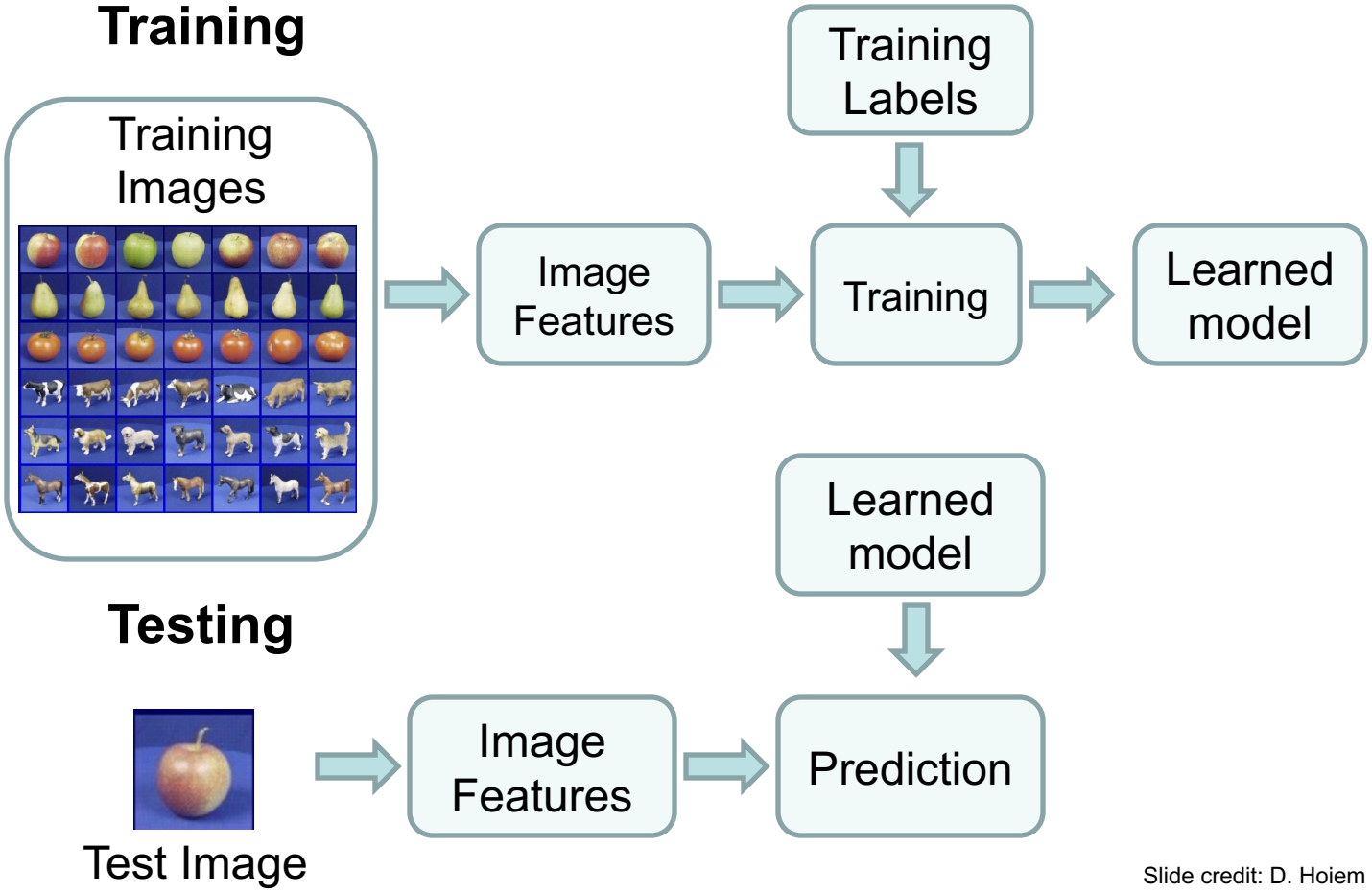
$$y = f(\mathbf{x})$$



- **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!

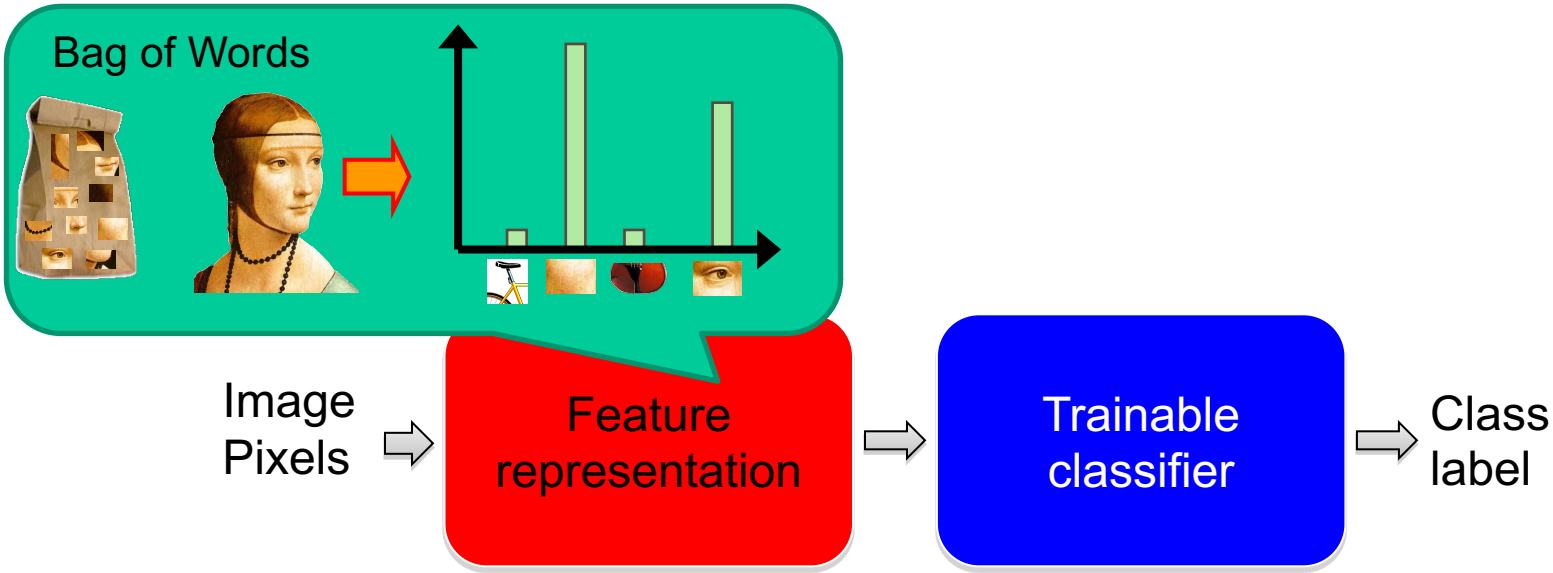
Slide modified from Lana Lezebnik

Steps



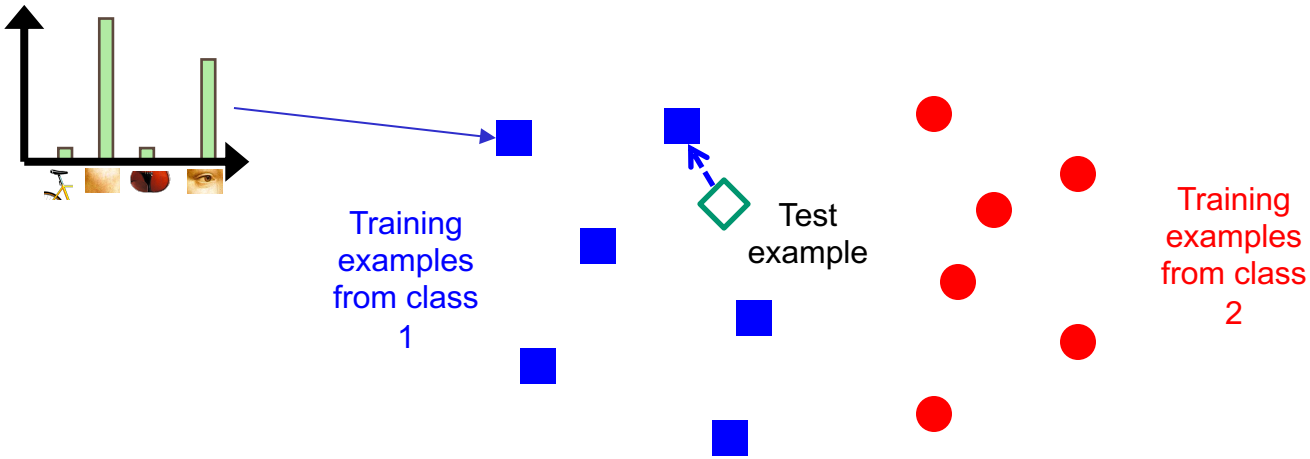
Slide credit: D. Hoiem

“Classic” recognition pipeline



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

Non-Linear Classifiers: Nearest neighbor

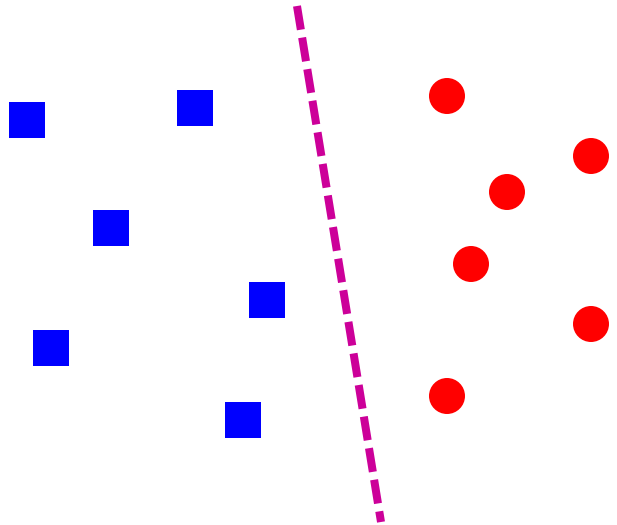


f(**x**) = label of the training example nearest to **x**

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

No training required!

Linear classifiers: Binary classification

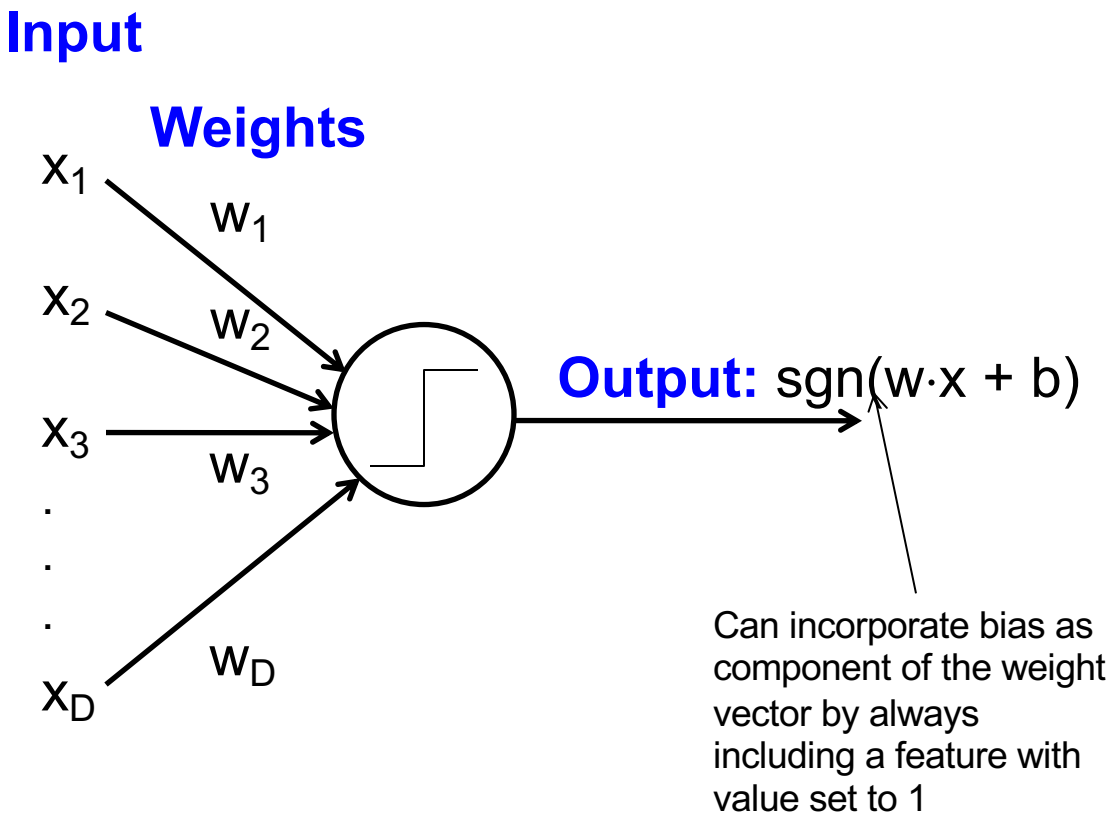


Find a *linear function* to separate the classes:

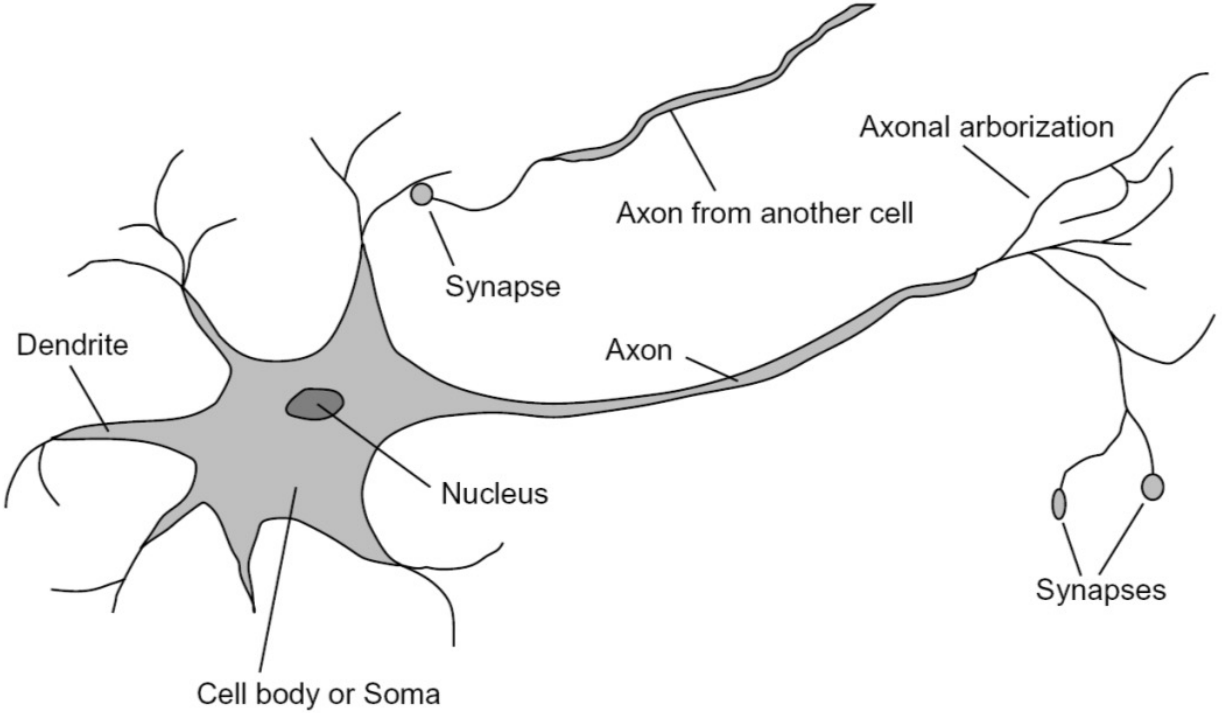
$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Slide courtesy of Lana Lezebnik

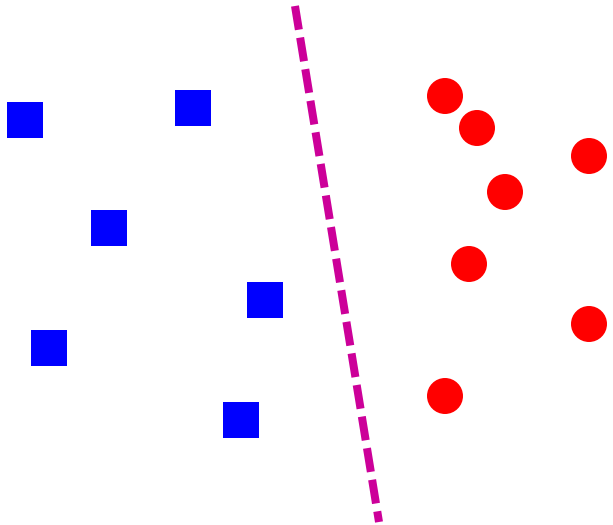
Linear classifiers: Binary Classification aka Perceptron



Loose inspiration: Human neurons



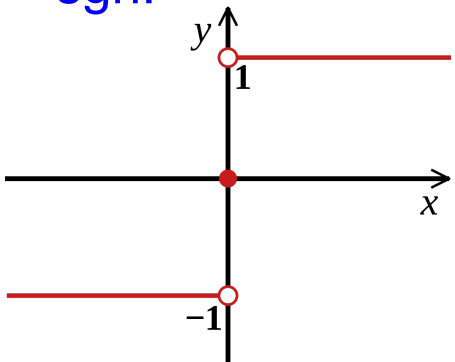
Linear Classifiers: Perceptrons



Find a *linear function* to separate the classes:

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

sgn:



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:

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

  - 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

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

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

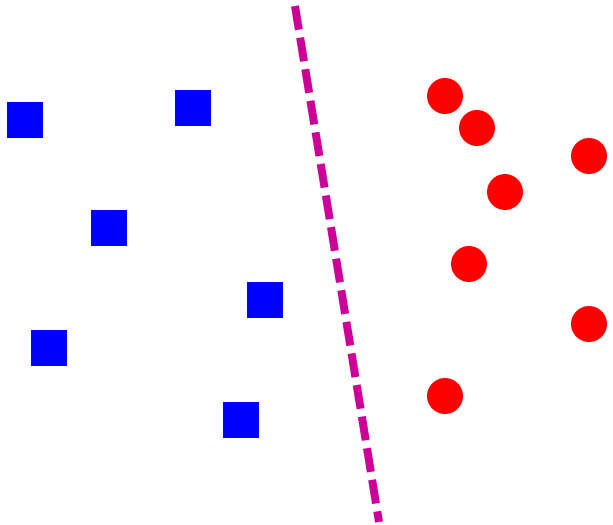
- The raw response of the classifier changes to

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

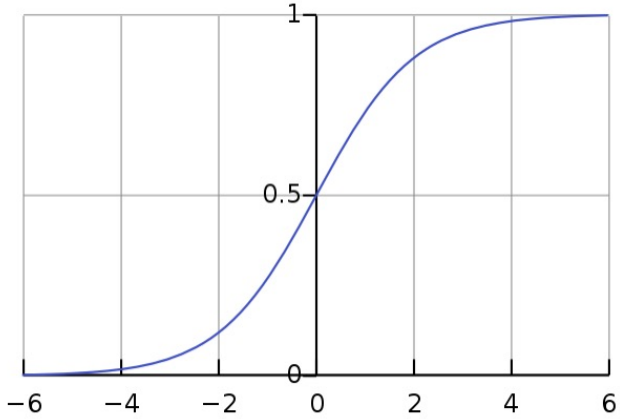
- If *y* = 1 and *y'* = -1, the response is initially *negative* and will be *increased*
- If *y* = -1 and *y'* = 1, the response is initially *positive* and will be *decreased*

Slide courtesy of Lana Lezebnik

Other Linear Classifiers: Logistic Regression



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



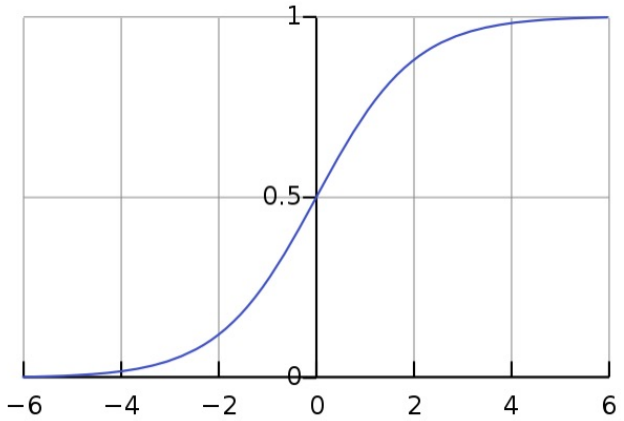
Soft version of sign with the sigmoid fn σ

$$y' = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

To classify, threshold: f(x) = y' > 0.5

Logistic Regression: Probabilistic Interpretation

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(\mathbf{w}\mathbf{x}+b)}}$$



Ground-truth label = {0, 1}

P(Y = 1 | **x**)?

= σ(**w** · **x** + b)

what is P(Y = 0 | **x**) ?

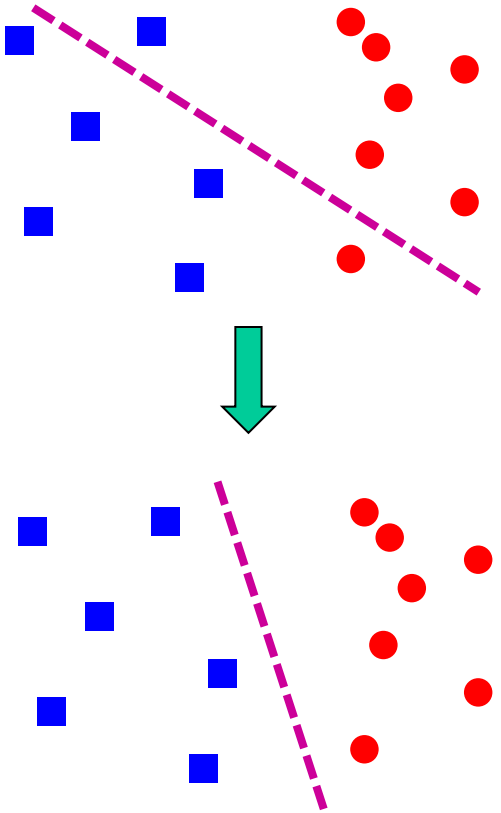
= 1 – P(Y = 1 | **x**)

How do we find, or learn, the W?

We need an objective!!

aka Loss function

“How far am I from the true labels
aka *ground truth*”



Objective (Loss) functions

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

$$\text{minimize}_{\mathbf{w}}\, L(\mathbf{w}) = \sum_i l(\mathbf{x}_i, y_i; \mathbf{w})$$

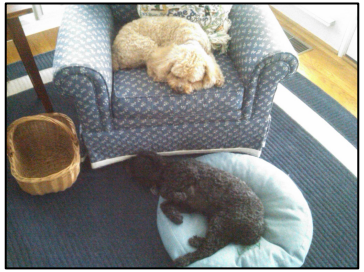
- Possible losses (for binary problems):
  - **Quadratic loss:** $l(\mathbf{x}_i, y_i; \mathbf{w}) = (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$
  - Log Likelihood loss: $l(\mathbf{x}_i, y_i; \mathbf{w}) = -\log P_{\mathbf{w}}(y_i \mid \mathbf{x}_i)$

$$= -\sum_k \mathbb{1}(y_i = \mathrm{k}) \log P_{\mathbf{w}}(\mathrm{k} \mid \mathbf{x}_i)$$

k = {0, 1} for binary case, can be multi-class k={0, … K}

Linear Classifiers: Beyond Binary Classification

Example Setup: 3 classes


Model – one weight per class: w_0, w_1, w_2

$w_0^T x$ big if cat

$w_1^T x$ big if dog

$w_2^T x$ big if hippo

Stack together: W_{3xF} where **x** is in R^F

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

W

x_i

-96.8	Cat score
437.9	Dog score
61.95	Hippo score

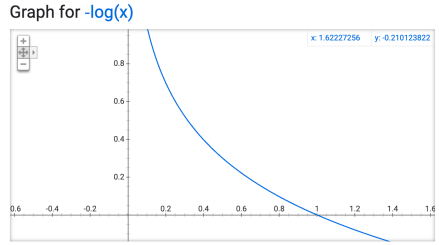
Wx_i

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



Loss

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

Cat weight vector	0.2	-0.5	0.1	2.0	1.1	56	-96.8	Cat score	0.00000… Cat probability
Dog weight vector	1.5	1.3	2.1	0.0	3.2	231	437.9	Dog score	.9999 Dog probability
Hippo weight vector	0.0	0.3	0.2	-0.3	-1.2	24	61.95	Hippo score	0.0000….. Hippo probability
						2			
					bias	1			

$\boldsymbol{W}$

$\boldsymbol{x_i}$

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

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

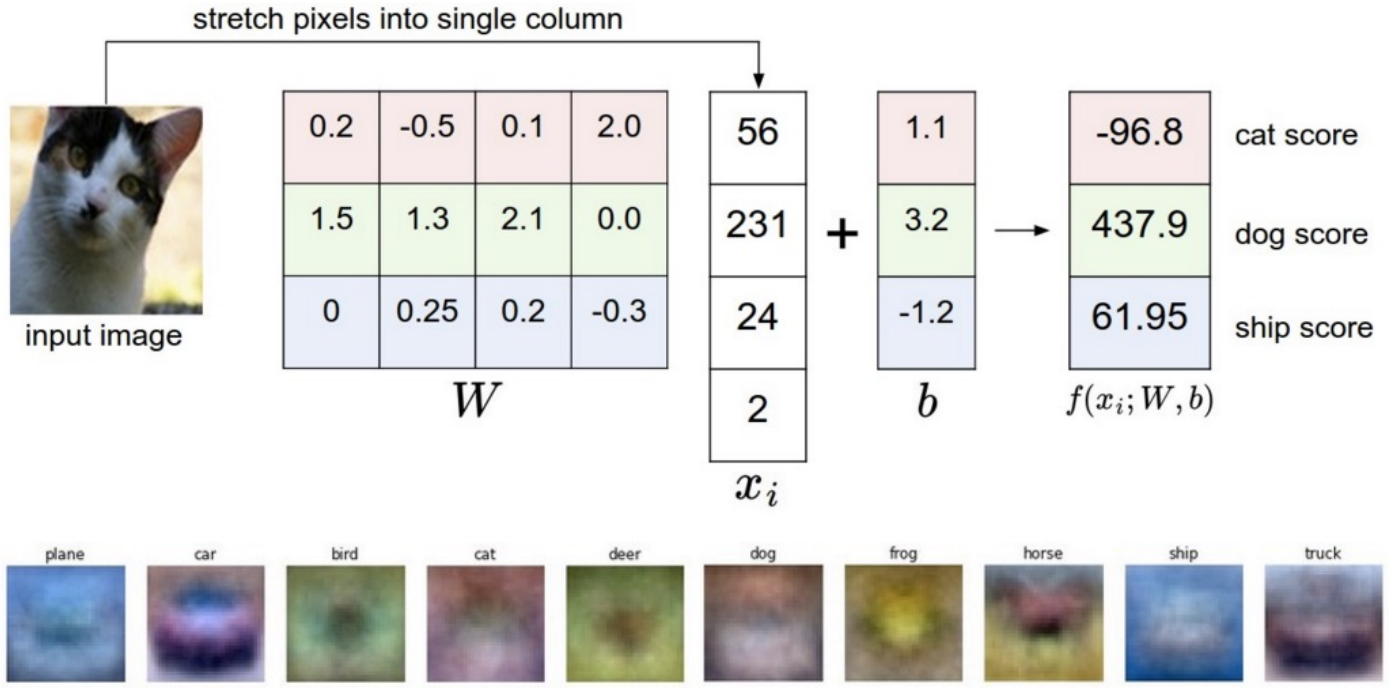
Weight matrix a collection of scoring functions, one per class

Prediction is vector where jth component is “score” for jth class.

Want the correct class to have probability = 1

Diagram by: Karpathy, Fei-Fei

Visualizing linear classifiers

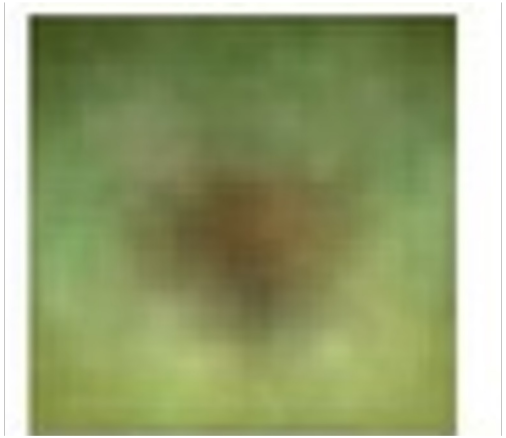


Source: Andrej Karpathy, http://cs231n.github.io/linear-classify/

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.

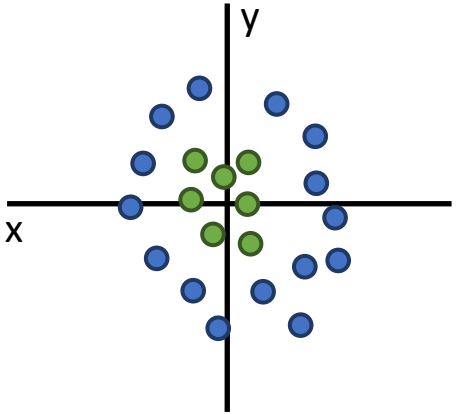
Ship or Dog?

Are linear classifiers enough??

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

Why?? Linear Classifiers not enough

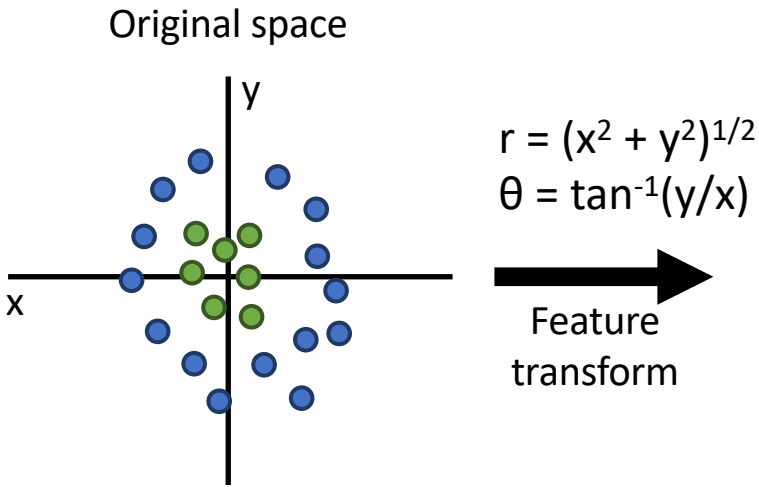
Geometric Viewpoint



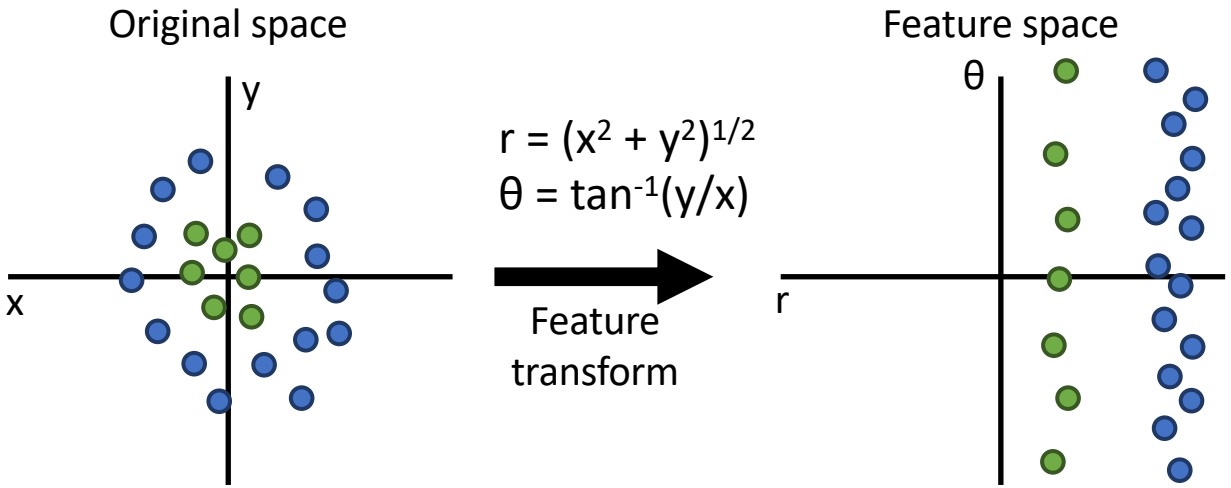
Visual Viewpoint

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

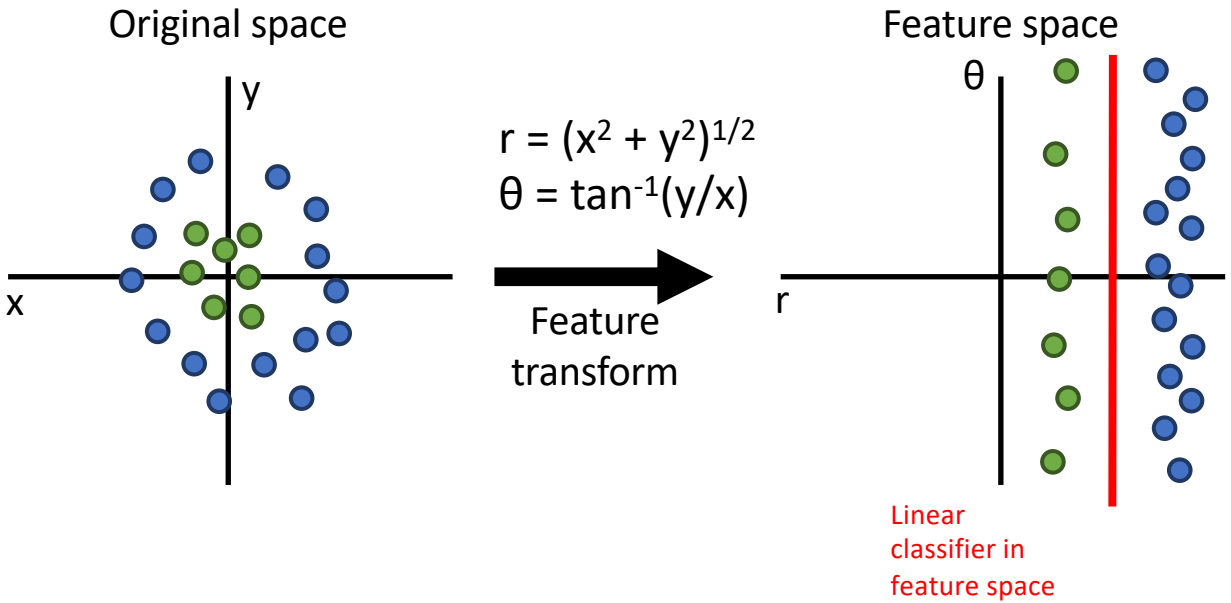
One solution: **Feature Transforms**



One solution: **Feature Transforms**

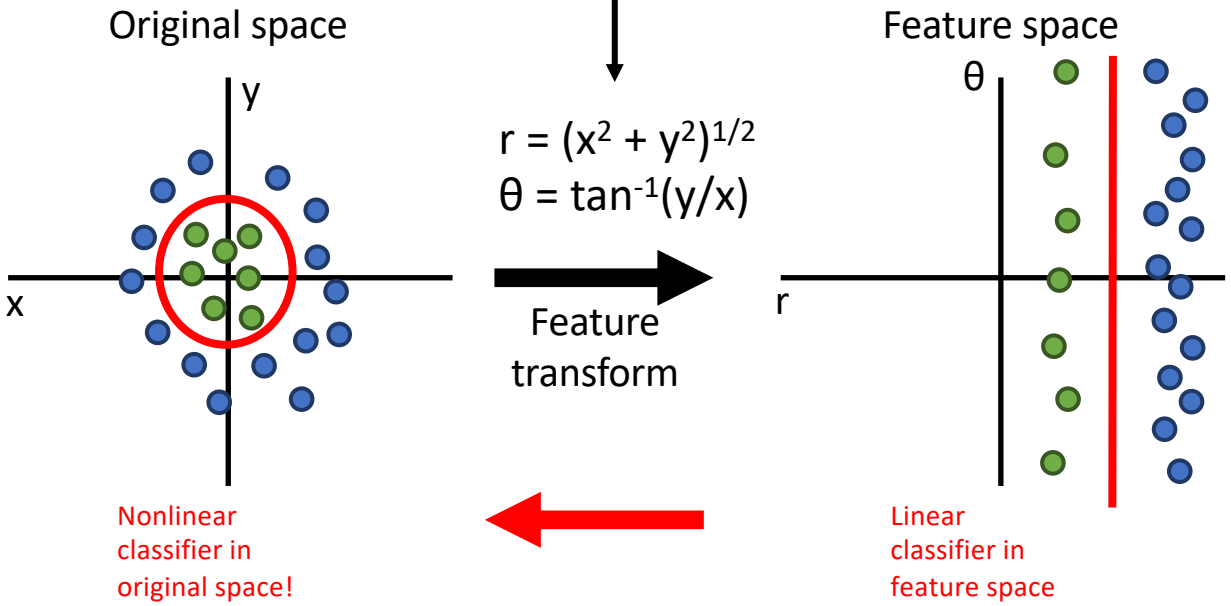


One solution: **Feature Transforms**

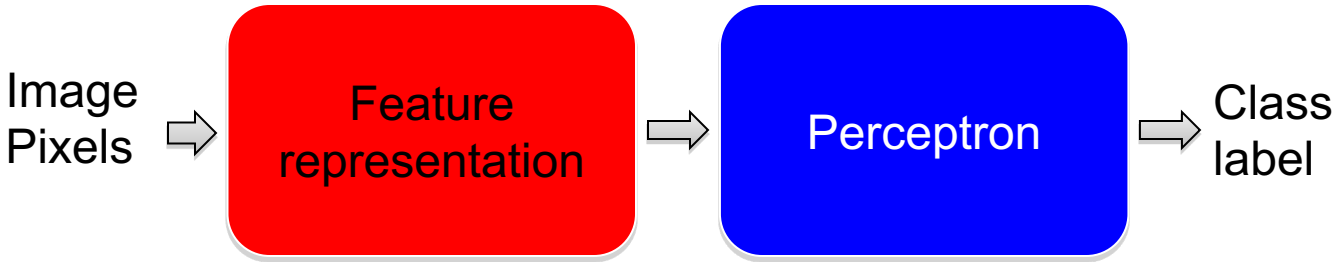


One solution: **Feature Transforms**

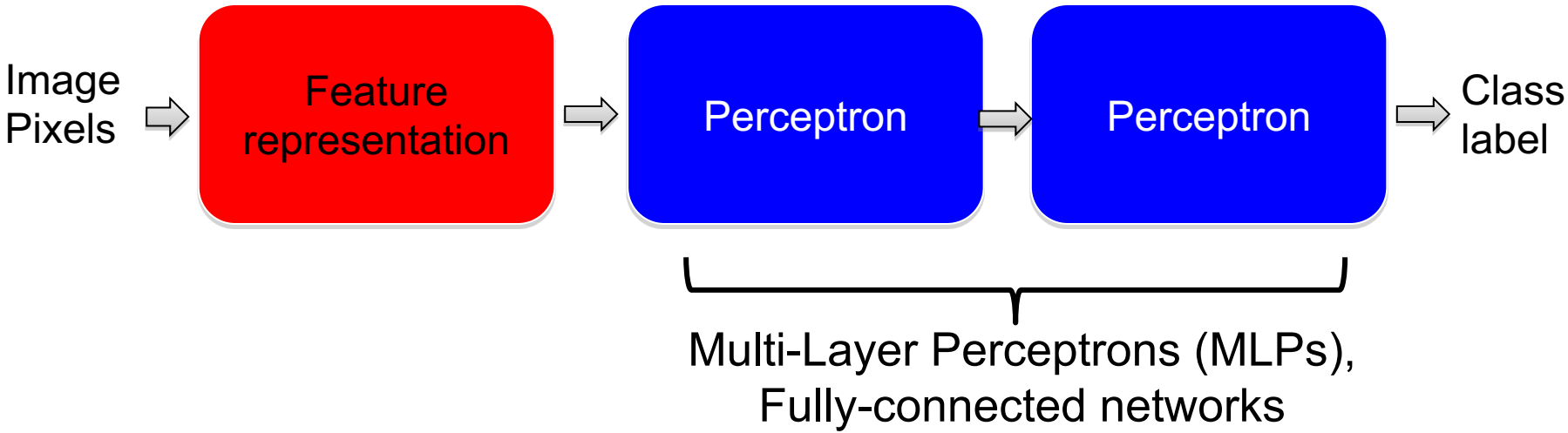
Neural Networks (MLPs) learn the transforms itself from data



Recognition pipeline: With Perceptrons



Recognition pipeline: With Multiple Perceptrons



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

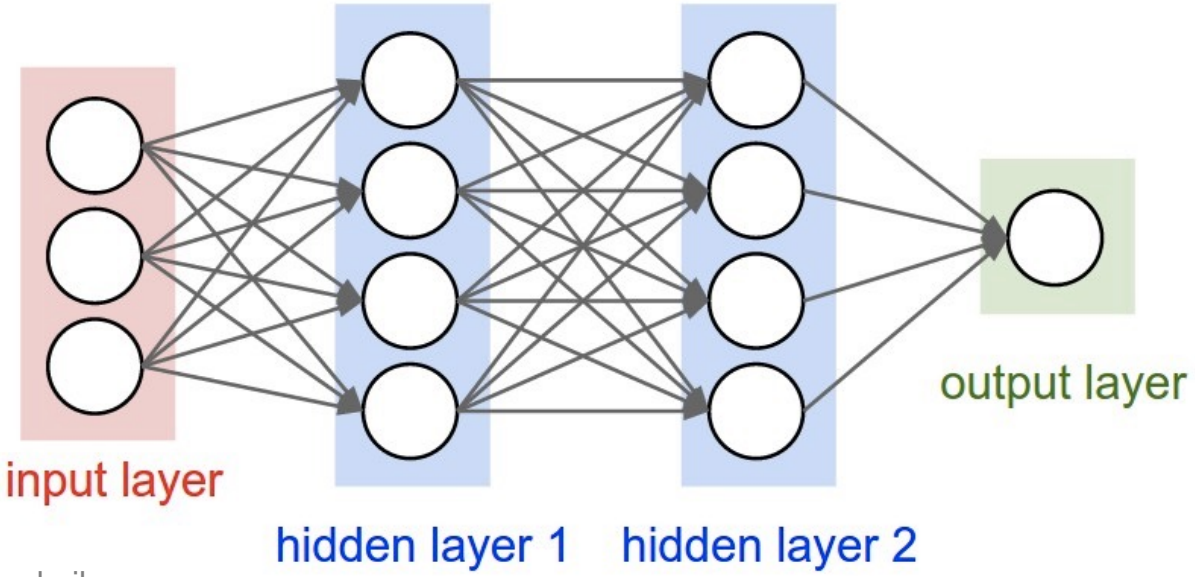
“Deep” recognition pipeline

- 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

Multi-layer perceptrons

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



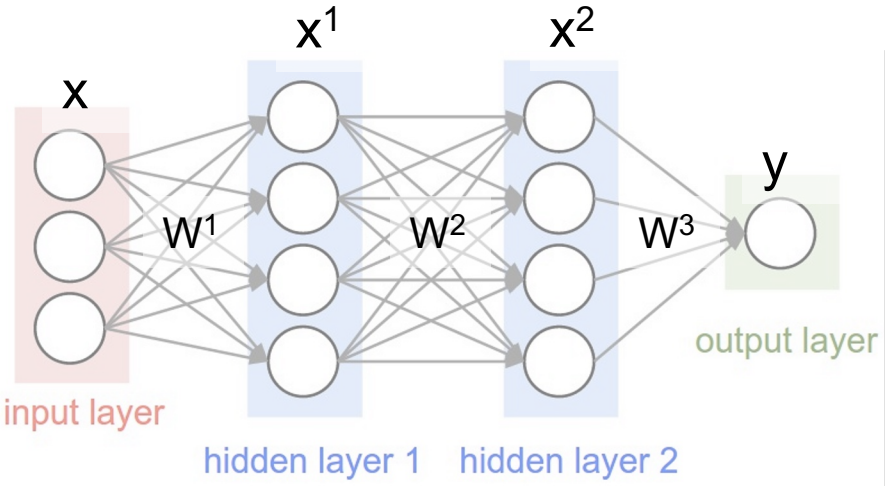
Slide courtesy of Lana Lezebnik

Remember this?

”Cat”

Image Classification

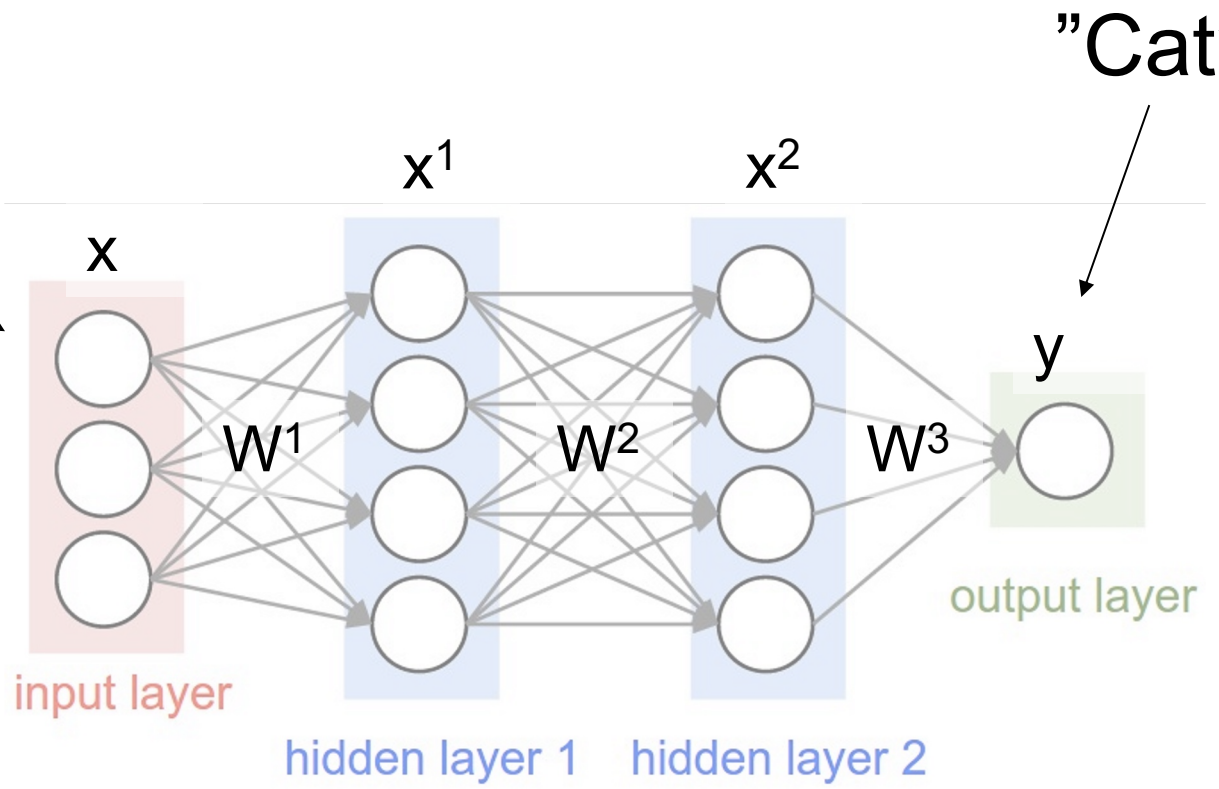
Remember this?



”Cat”

Image Classification

Multi-layer perceptrons / Fully-Connected Layer

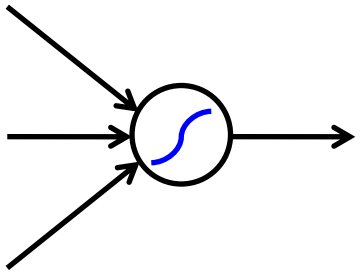


Non-linearity

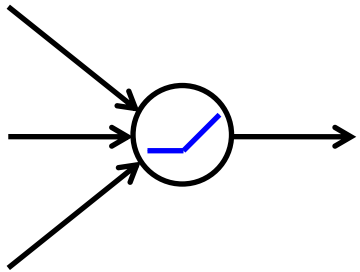
- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
  - This requires each perceptron to have a nonlinearity
  - To be trainable, the nonlinearity should be *differentiable*

One fully-connected layer:

$$z = Wx + b$$
$$y = \sigma(z)$$



Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



Rectified linear unit (ReLU)

$$\sigma(z) = \max(0, z)$$

Most popular. Use subderivatives at 0

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?

$$s = W_2 W_1 x$$

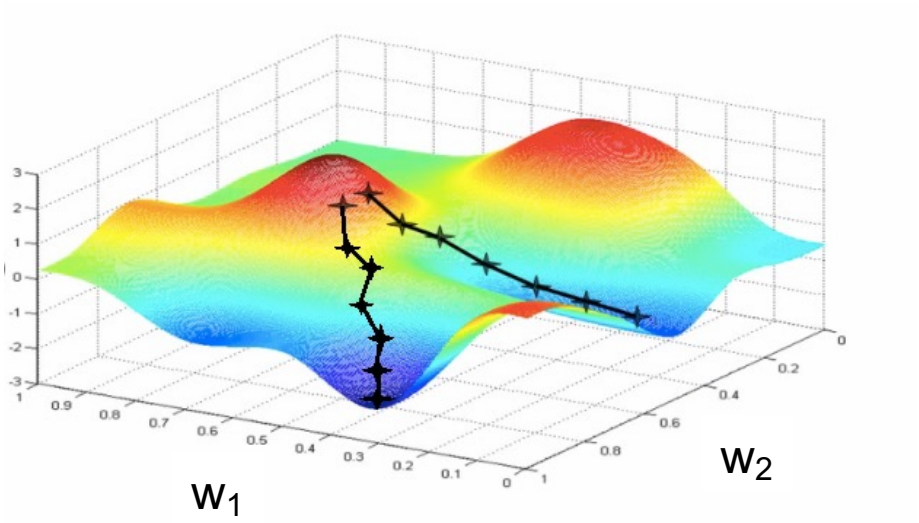
Slide modified from Johnson & Fouhey

Training 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

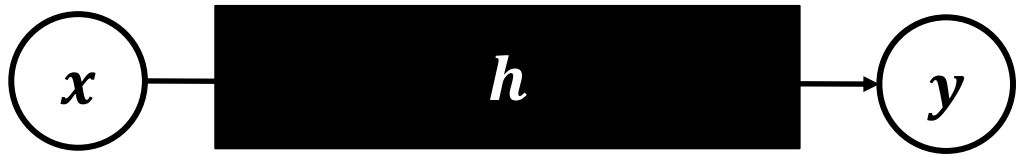
Slide courtesy of Lana Lezebnik

Recall: Chain Rule

$$h(x) = f(g(x))$$

What's the derivative?

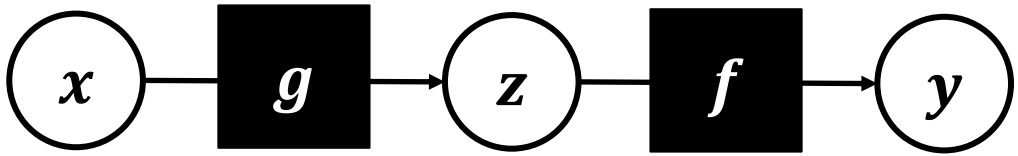
$$h'(x) = f'(g(x))g'(x)$$



Let

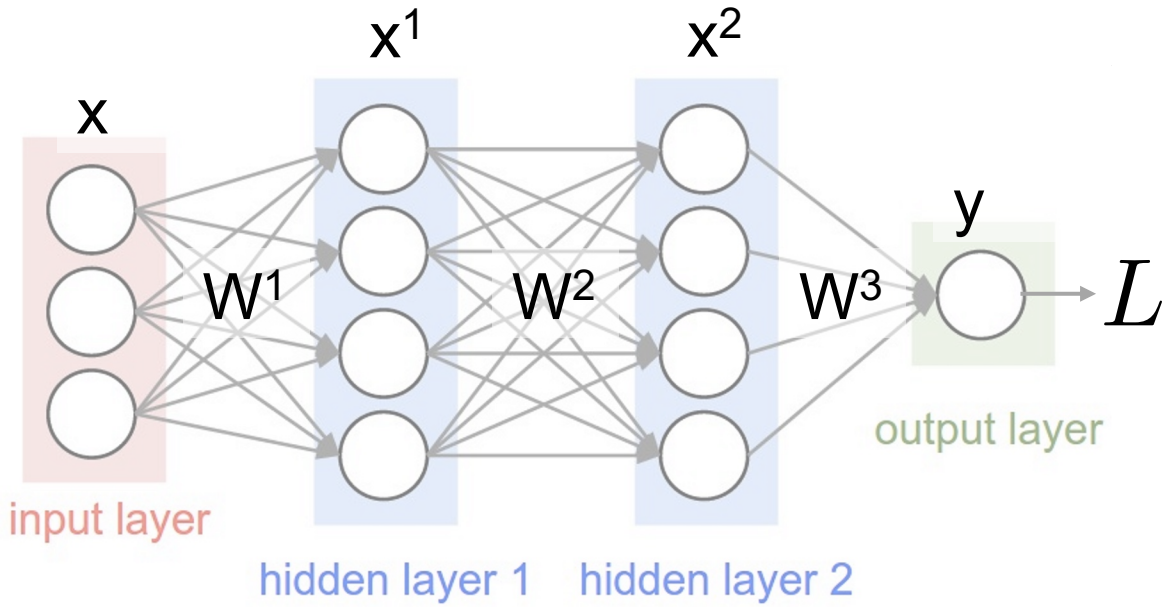
$z = g(x)$

$y = f(x)$



$$\frac{dy}{dx} = \frac{dy}{dz} * \frac{dz}{dx}$$

Back prop in 2 slides



What do we need for training Ws?

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

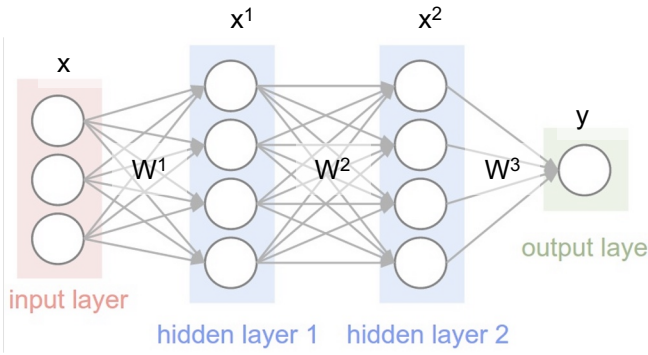
$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial W^3}$$

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

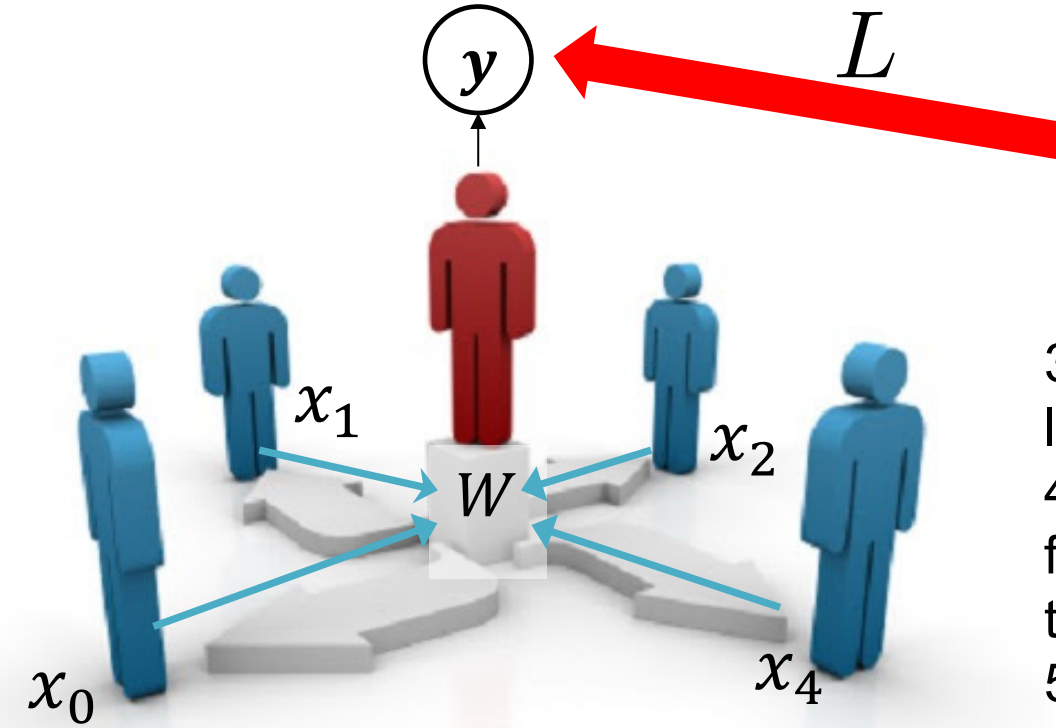
Where $x^l = \sigma(W^l x^{l-1})$

Chain rule!!

Intuitive Backprop Analogy



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



2. Then the boss gives you a feedback on what your output y should've been

3. You adjust how you should've listened to each neurons
4. Each neuron passes on the frustration (L) weighted by how much they were listened to.
5. Repeat down the chain of report.

Backprop as workplace analogy

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 how much to listen to each: Higher up loss multiplied by how loud each neuron was

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

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

Case study with MNIST

Fig. 4. Size-normalized examples from the MNIST database.

Training set: 60k images, with their ground truth label ={0, 1, … 9}

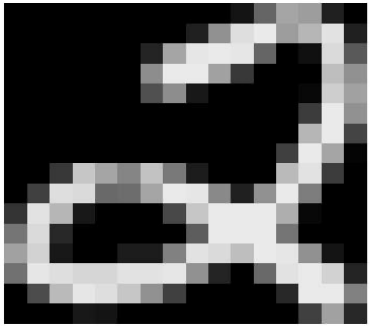
Test set: 10k

Each image is 28x28 black and white.

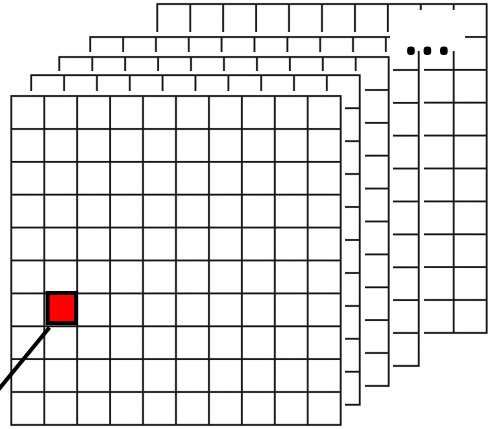
yann.lecun.com/exdb/mnist/
Yann LeCun & Corinna Cortes

One layer NN

input:



$x = 28 * 28 \times 1$



$W = 10 \times 28 * 28$

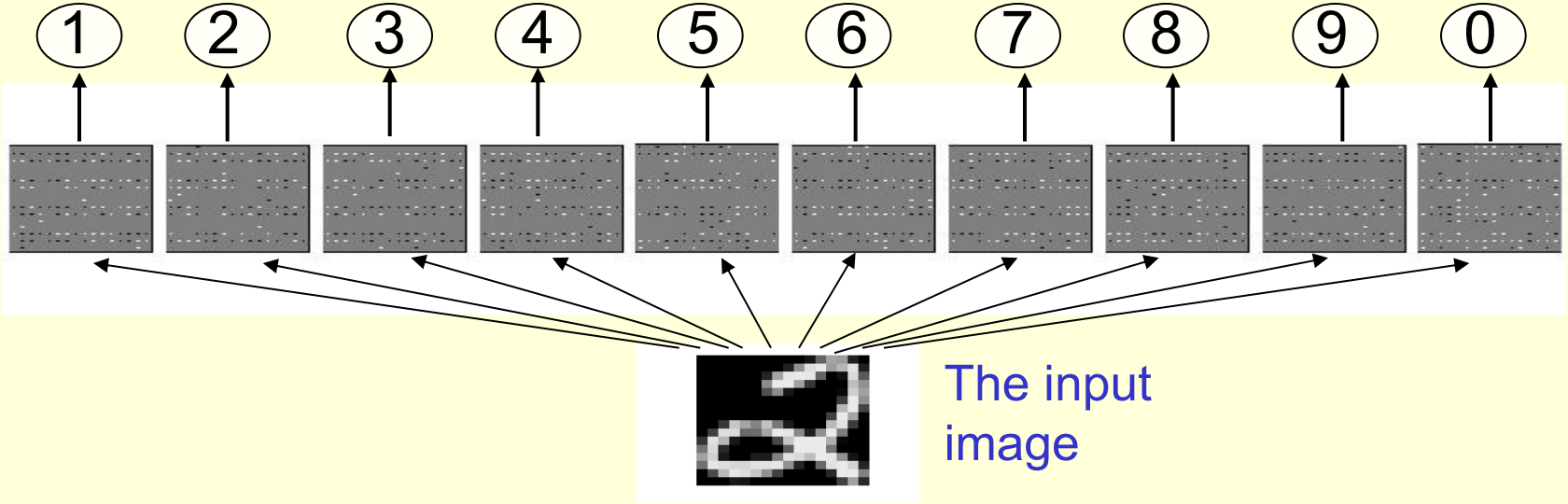
represents how much this weight or “manager” listens to this pixel (i,j)

$=$

$y = 10 \times 1$

take index with the maximum votes as the predicted digit

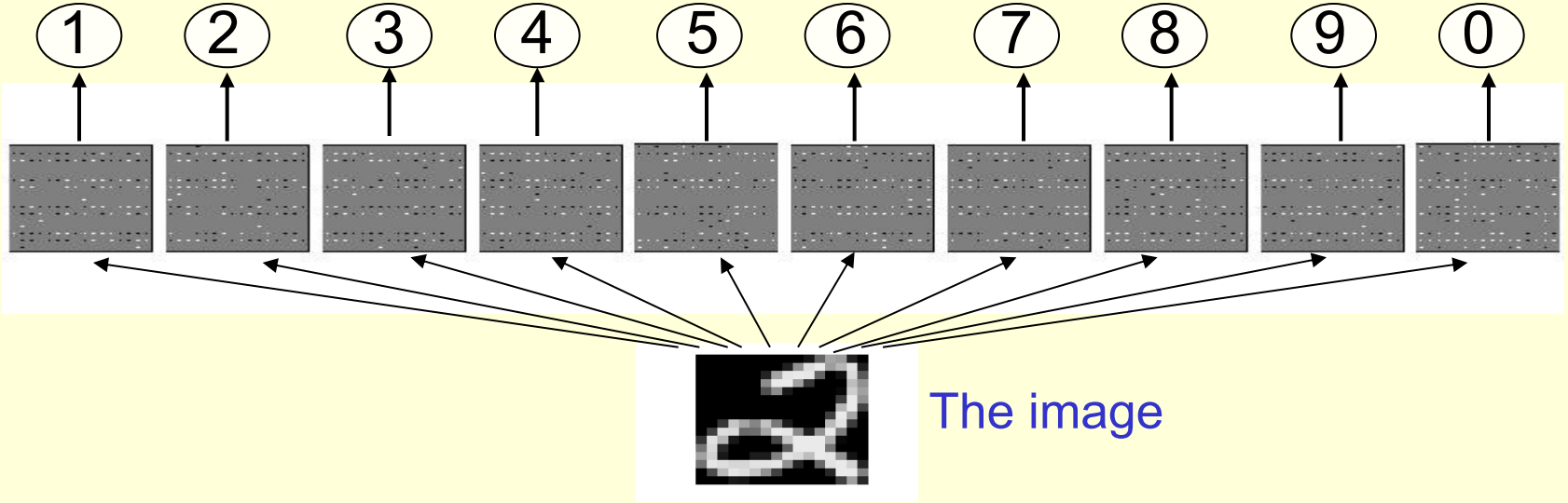
Displaying the Weights



Per-pixel intensity represents the magnitude of the weight and the color (black or white) represents the sign.

Slide by Geoff Hinton

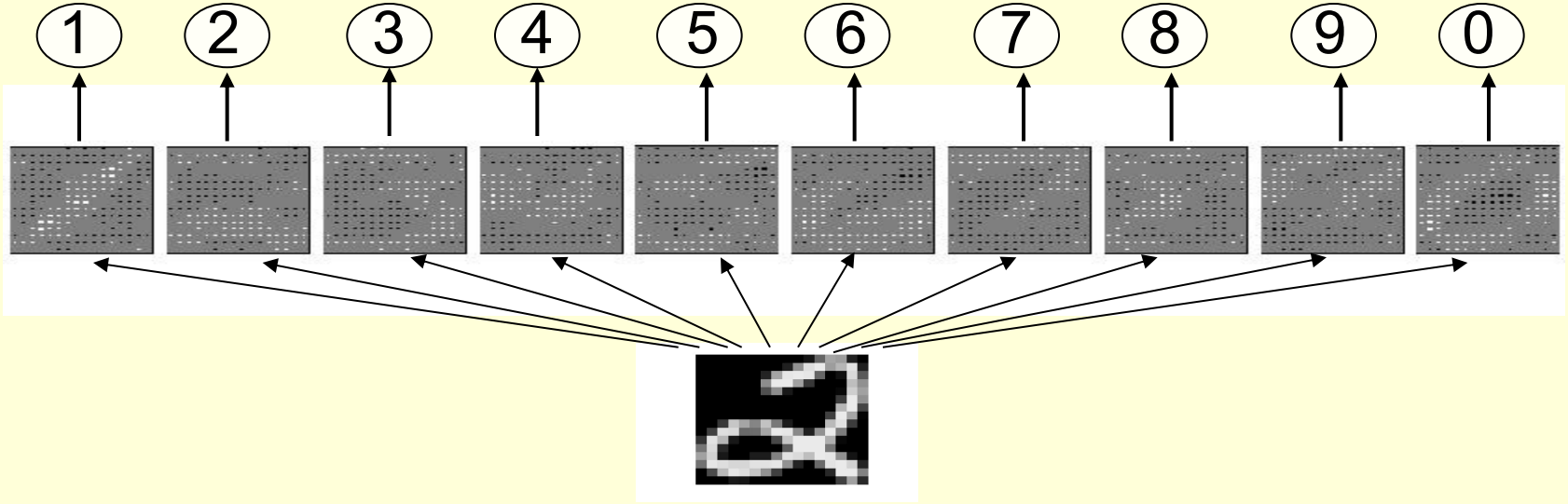
Learn the weights



Show the network an image and increment the weights from active pixels to the correct class.

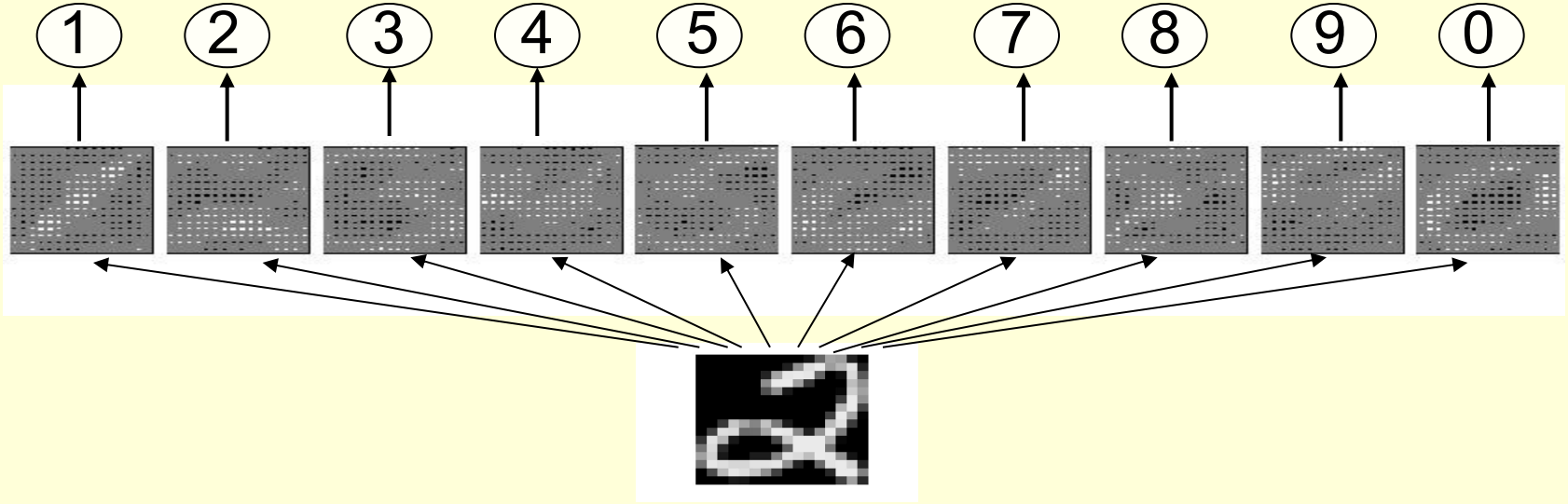
Then decrement the weights from active pixels to whatever class the network guesses.

Slide by Geoff Hinton



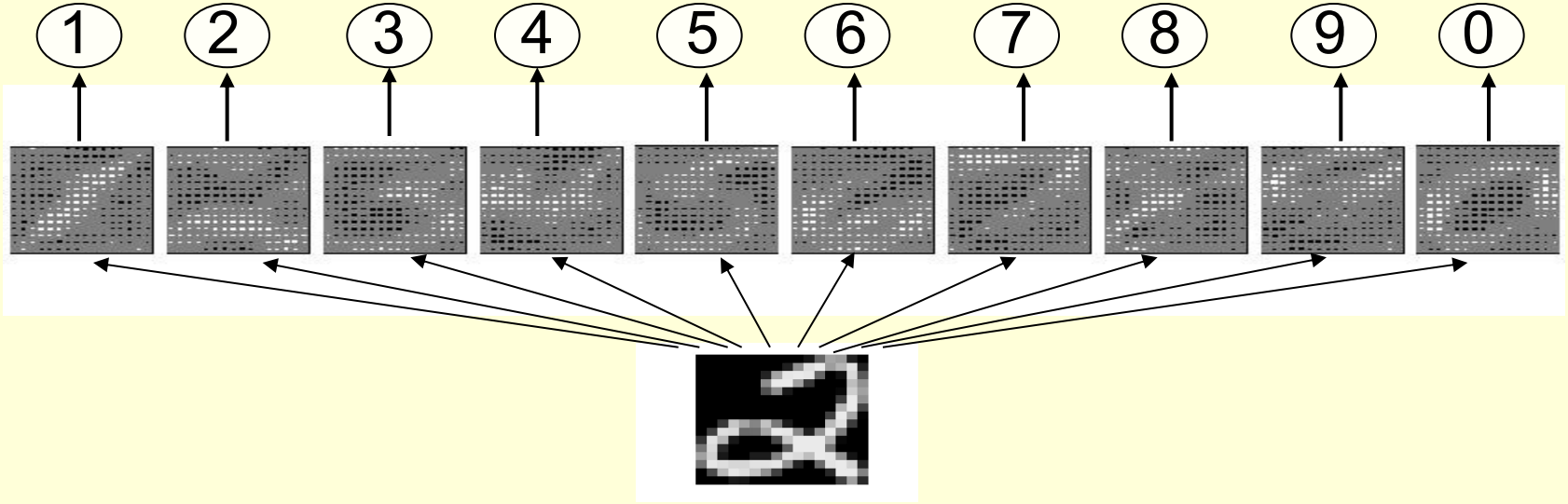
The image

Slide by Geoff Hinton



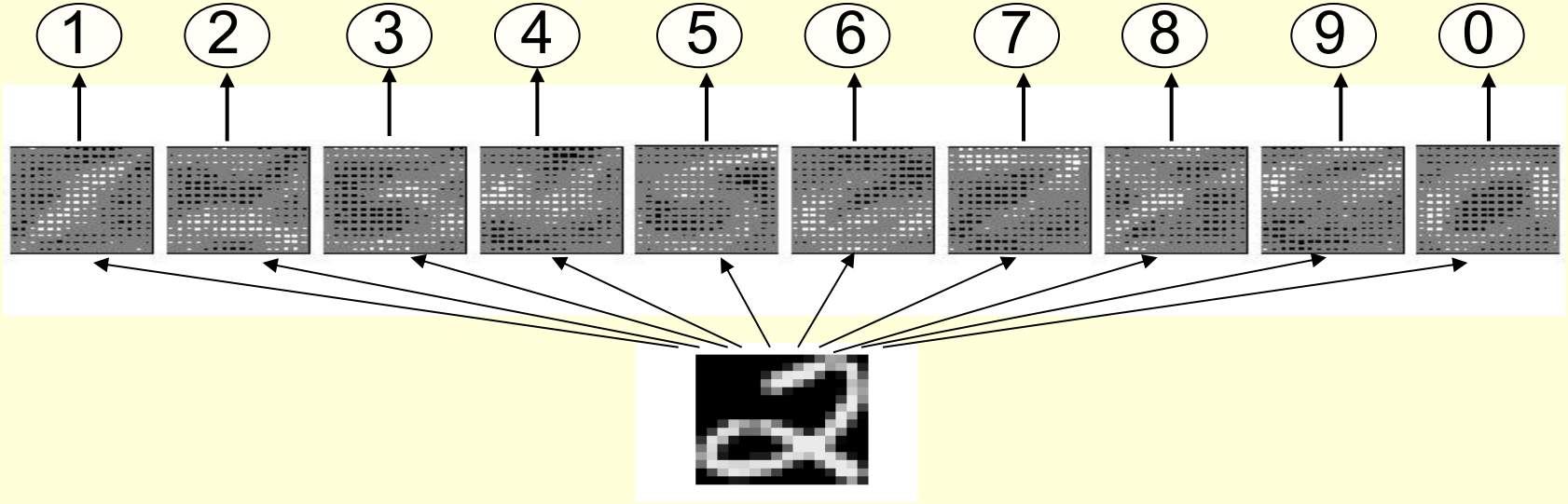
The image

Slide by Geoff Hinton



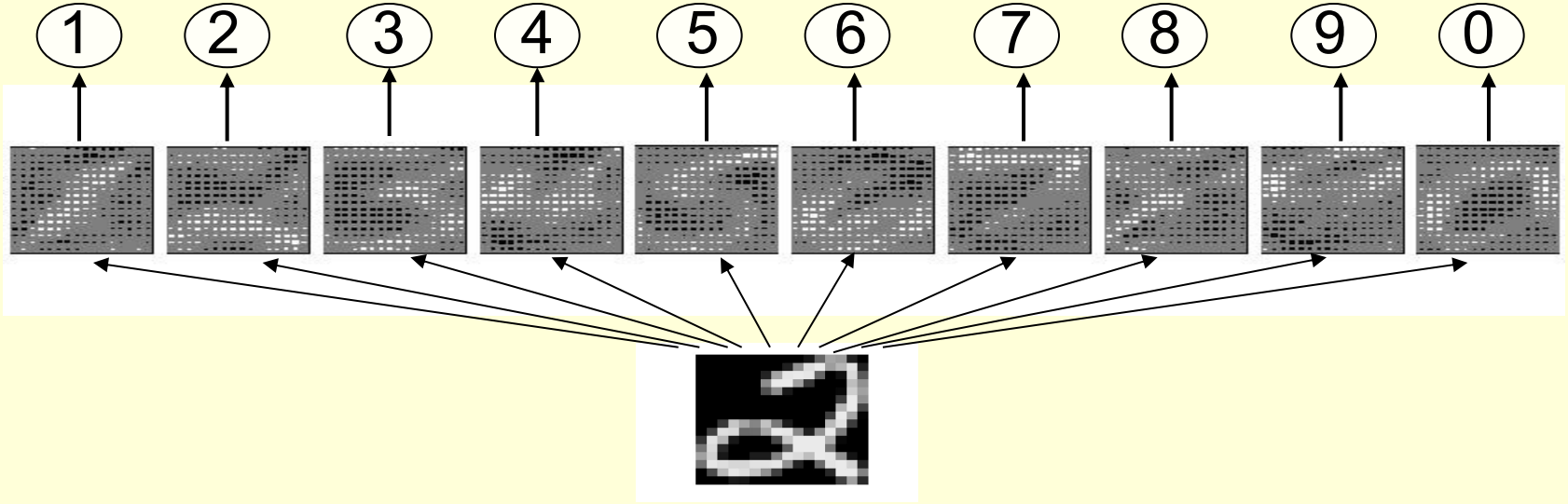
The image

Slide by Geoff Hinton



The image

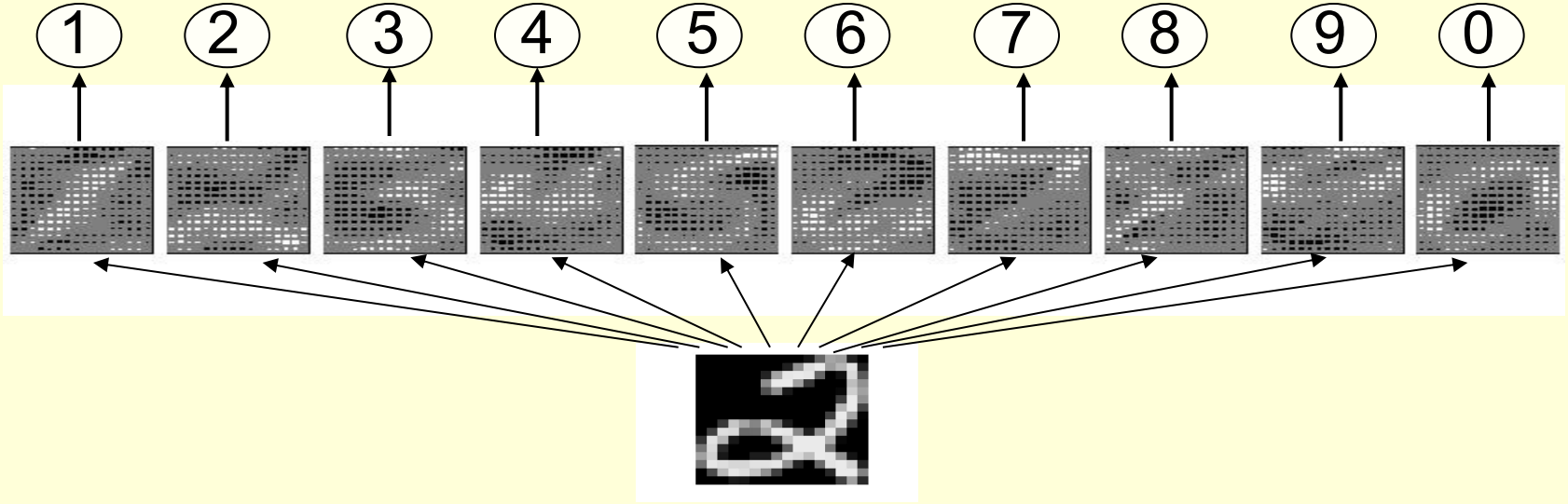
Slide by Geoff Hinton



The image

Slide by Geoff Hinton

The learned weights



The image

Slide by Geoff Hinton

Why the simple learning algorithm is insufficient

- This setup (single learned layer) is equivalent to having a rigid template for each shape.
  - The winner is the template that has the biggest overlap with the ink.
- The ways in which hand-written digits vary are much too complicated to be captured by simple template matches of whole shapes.

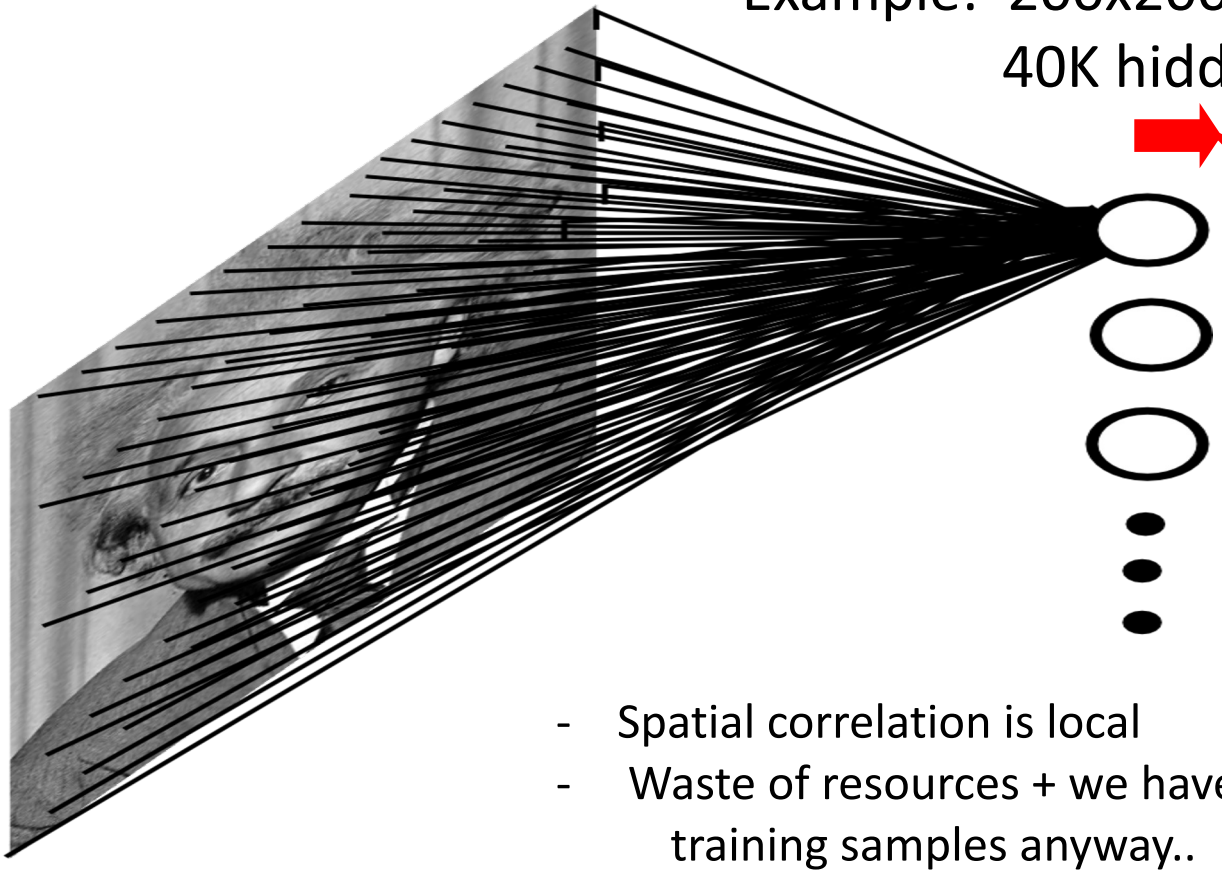
Slide by Geoff Hinton

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

Example: 200x200 image

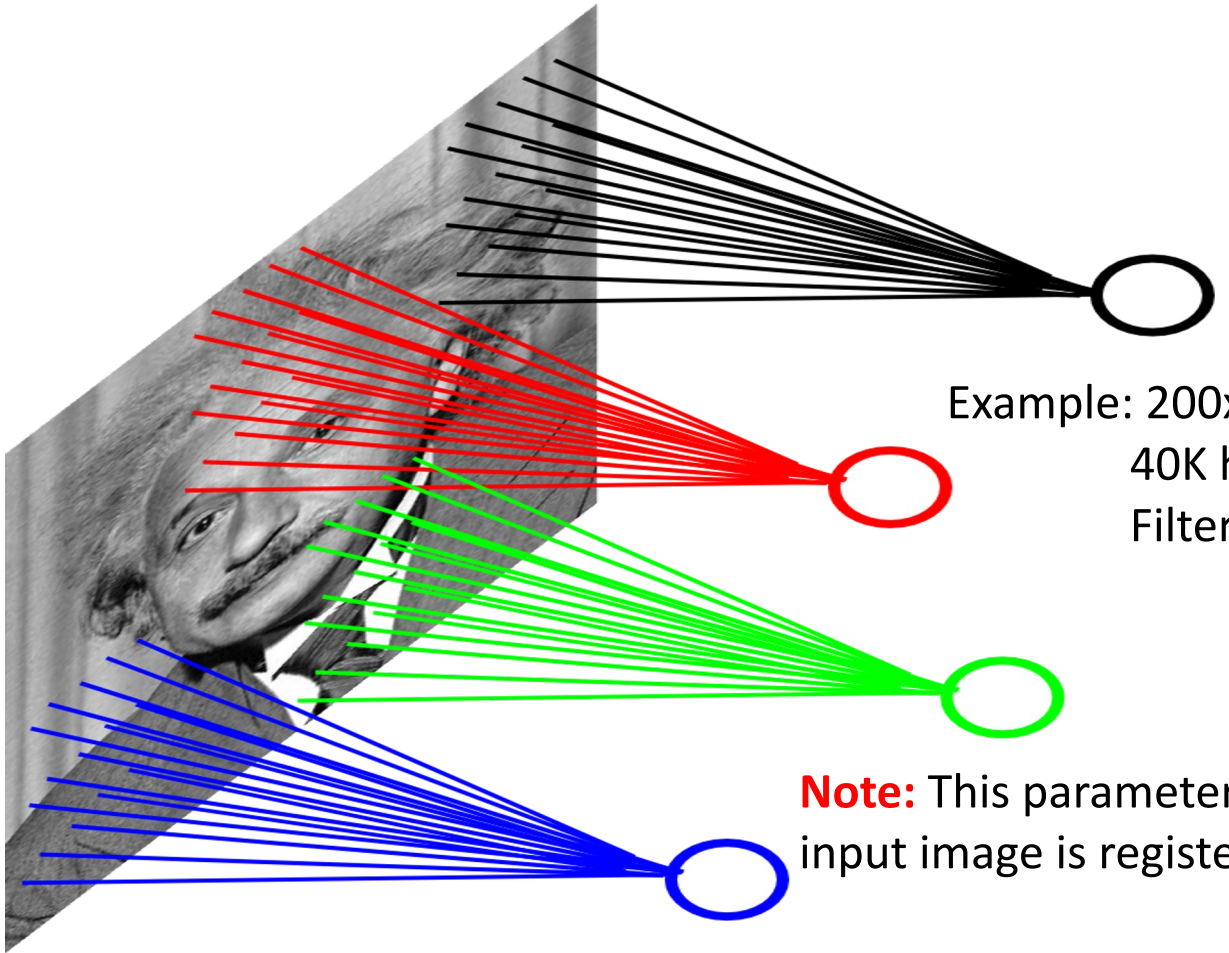
40K hidden units

~2B parameters!!!



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

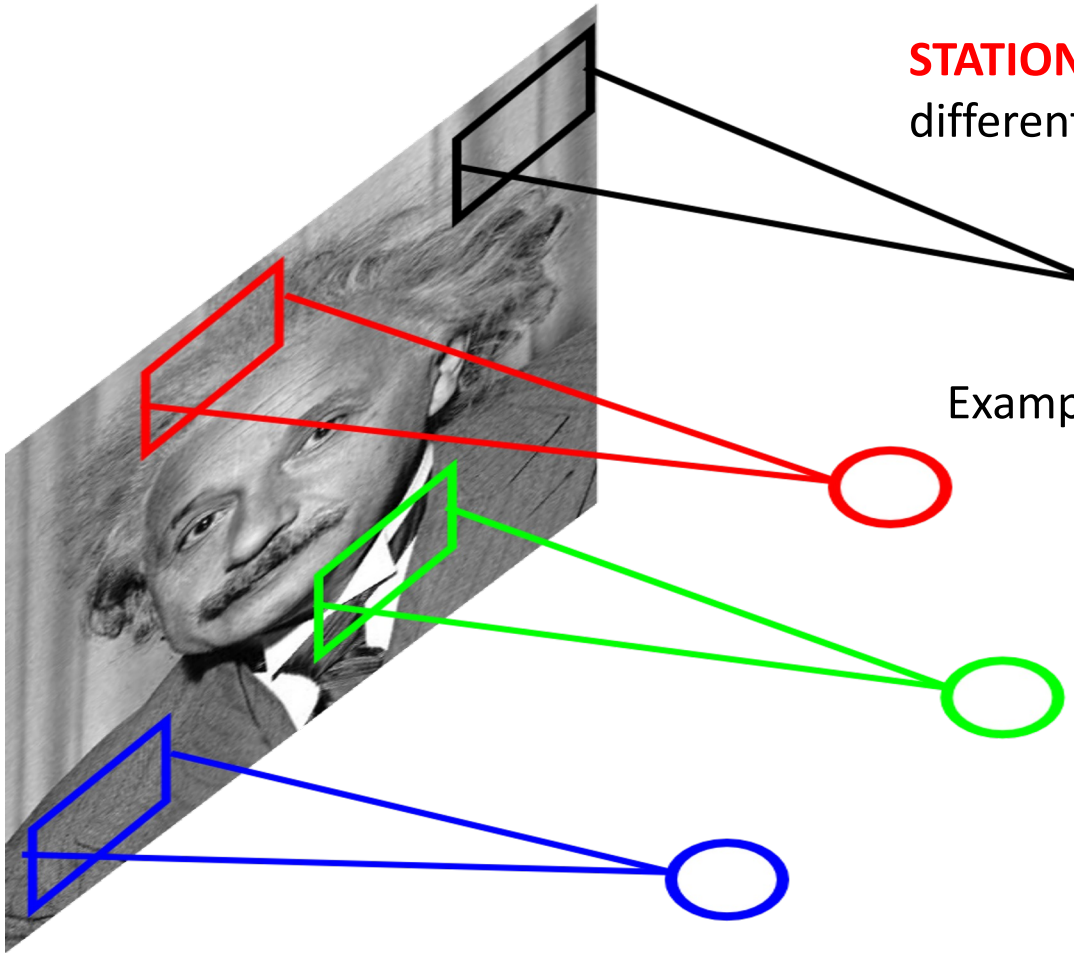
Locally Connected Layer



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

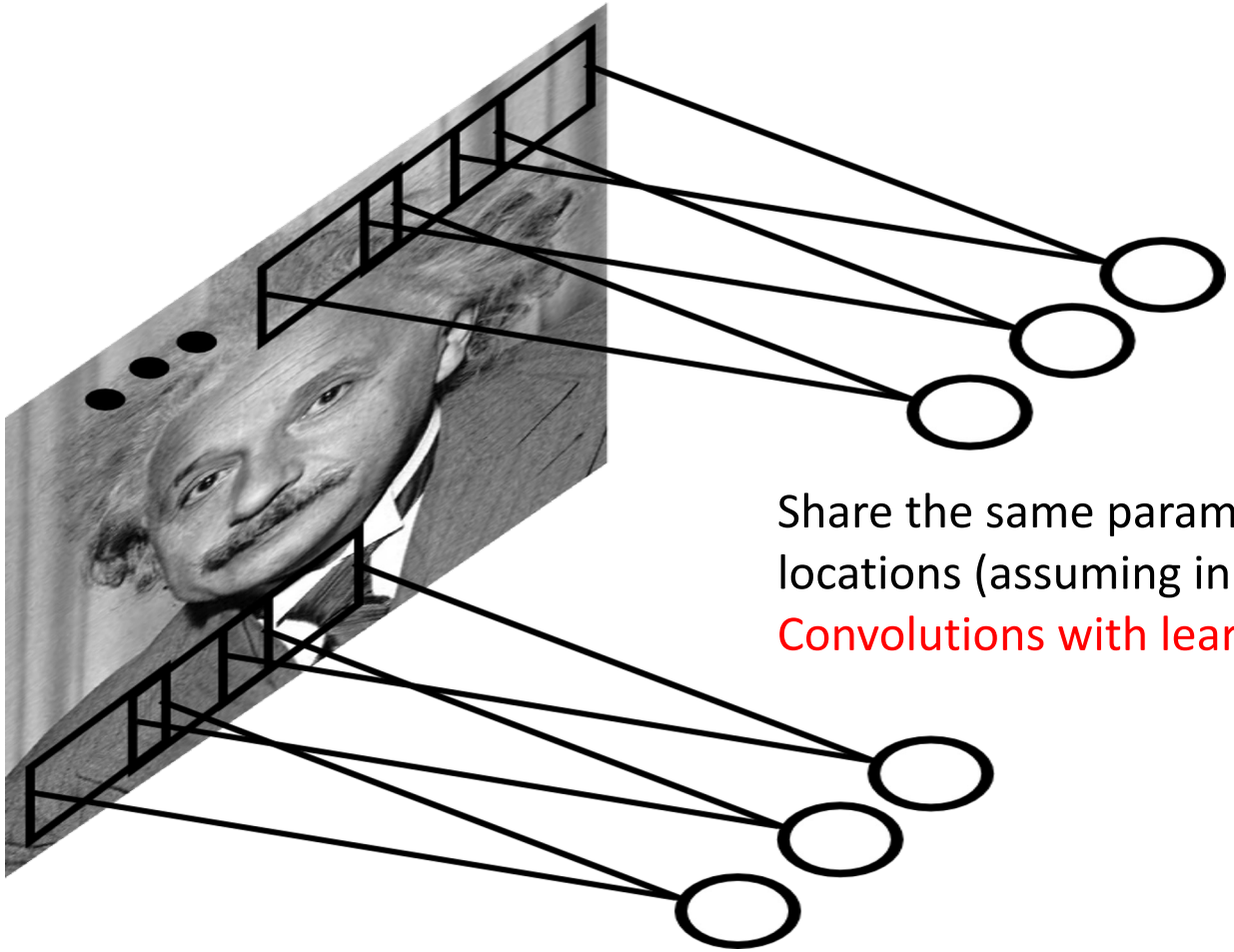
Example: 200x200 image

40K hidden units

Filter size: 10x10

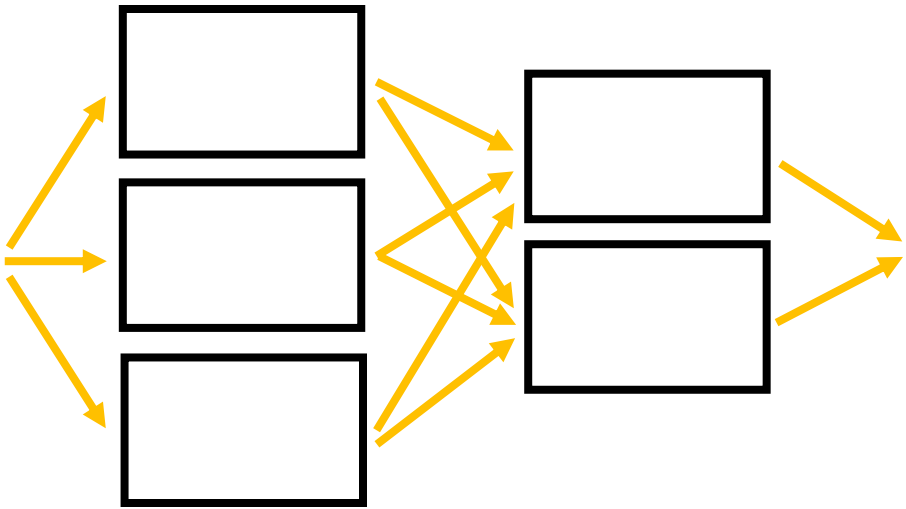
4M parameters

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

Angjoo Kanazawa & Alexei Efros, UC Berkeley, Fall 2022

Neural Nets: a particularly useful Black Box

image *X*

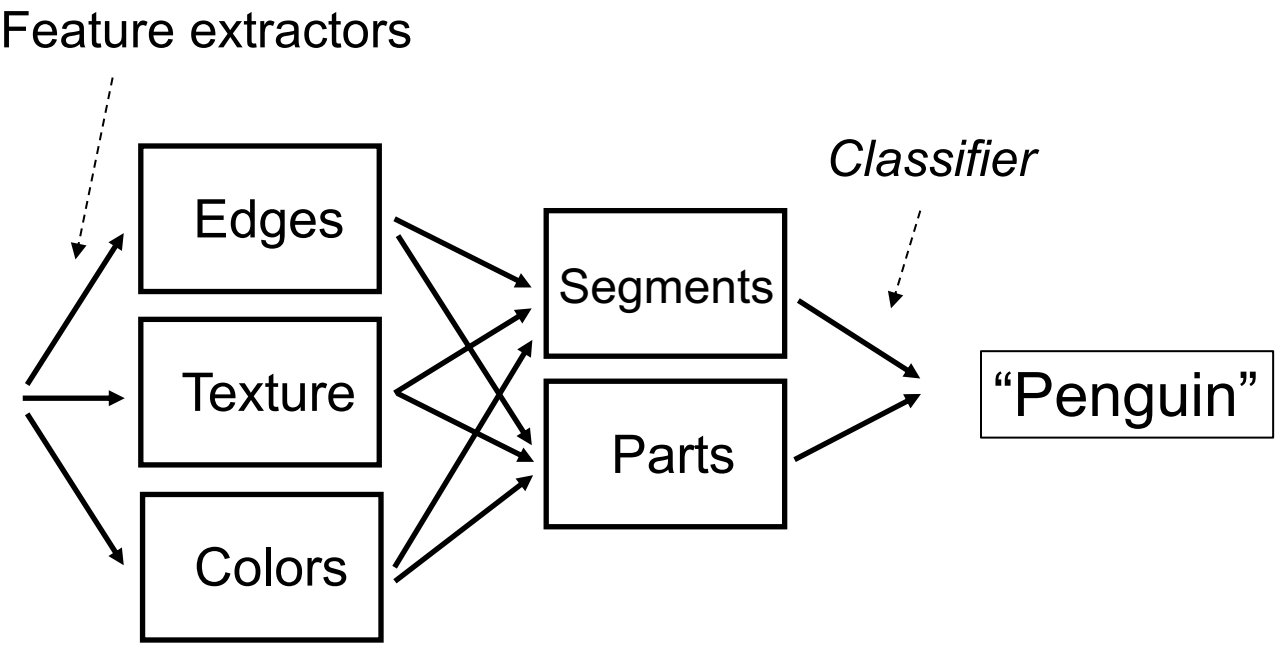
Convolutiona
l Neural
Network

"Penguin"

label *Y*

Classic Object Recognition

image *X*

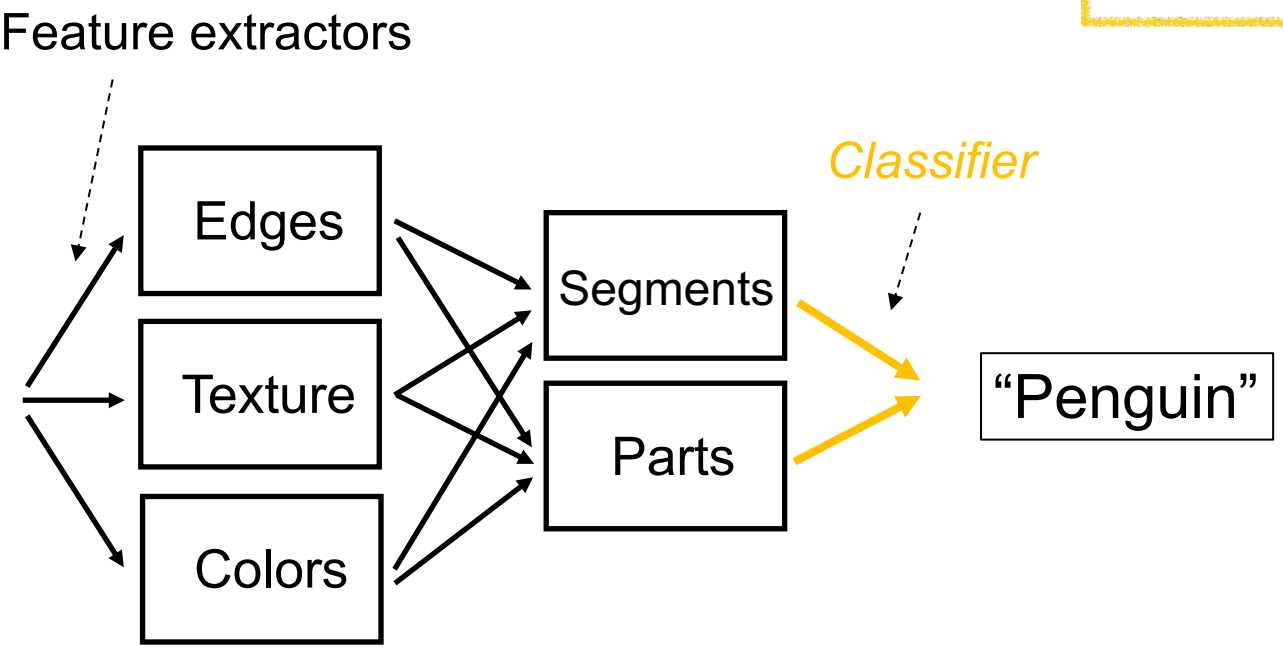


label *Y*

Slide by Philip Isola

Classic Object Recognition

image *X*



label *Y*

Slide by Philip Isola

Learning Features

Learned

Feature extractors

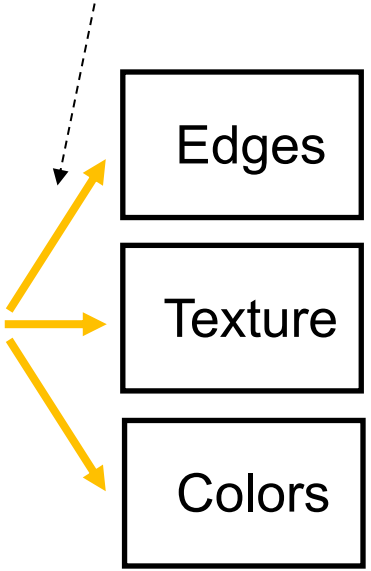
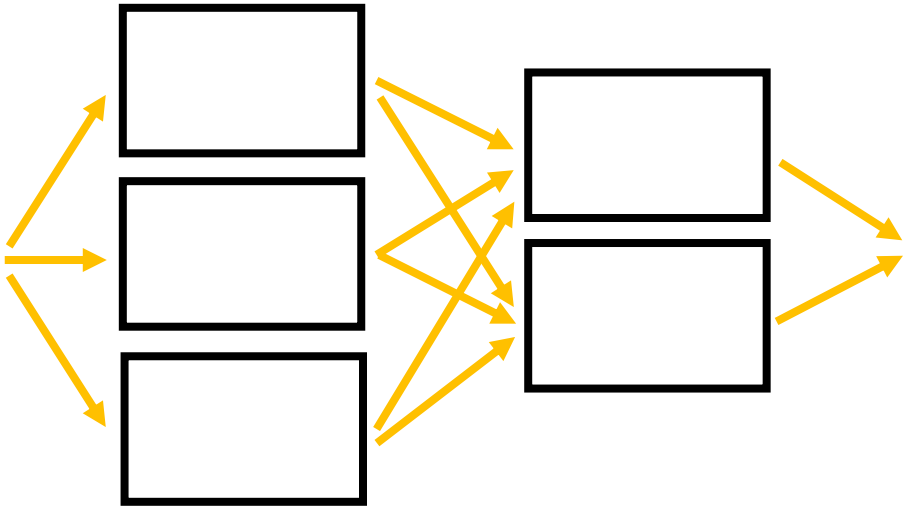


image *X*

label *Y*

Neural Network: algorithm + feature + data!

Learned



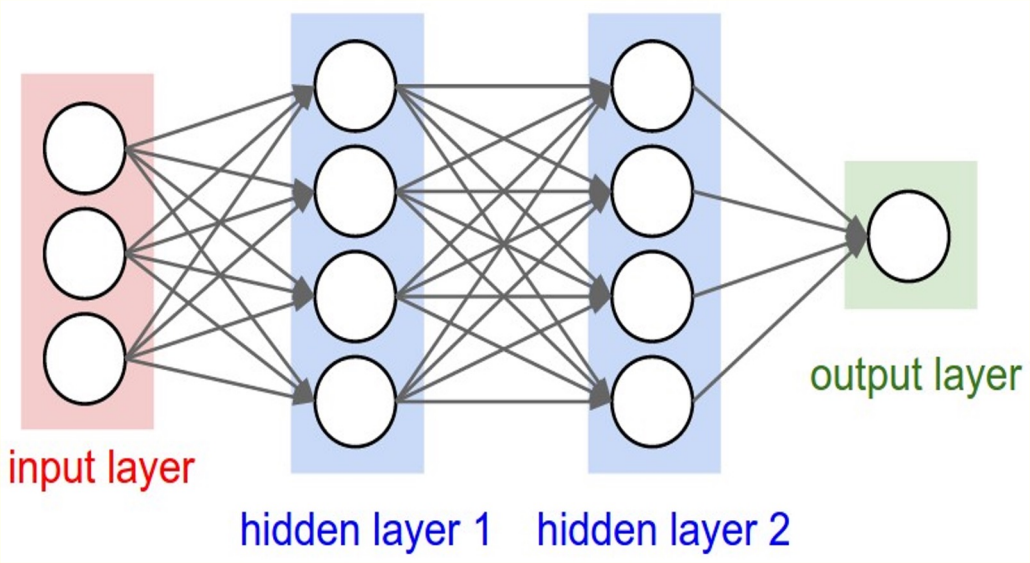
"Penguin"

image *X*

label *Y*

Slide by Philip Isola

Vanilla (fully-connected) Neural Networks

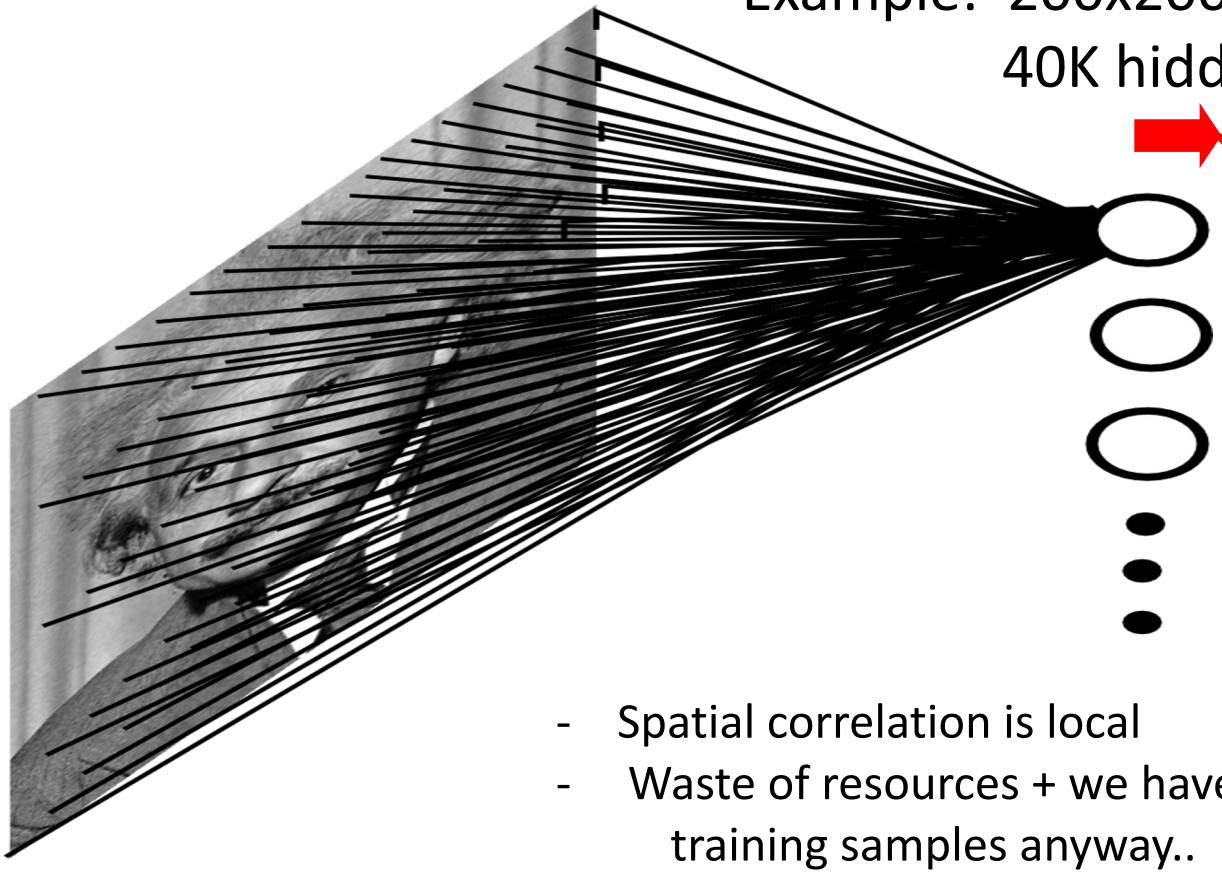


Fully Connected Layer

Example: 200x200 image

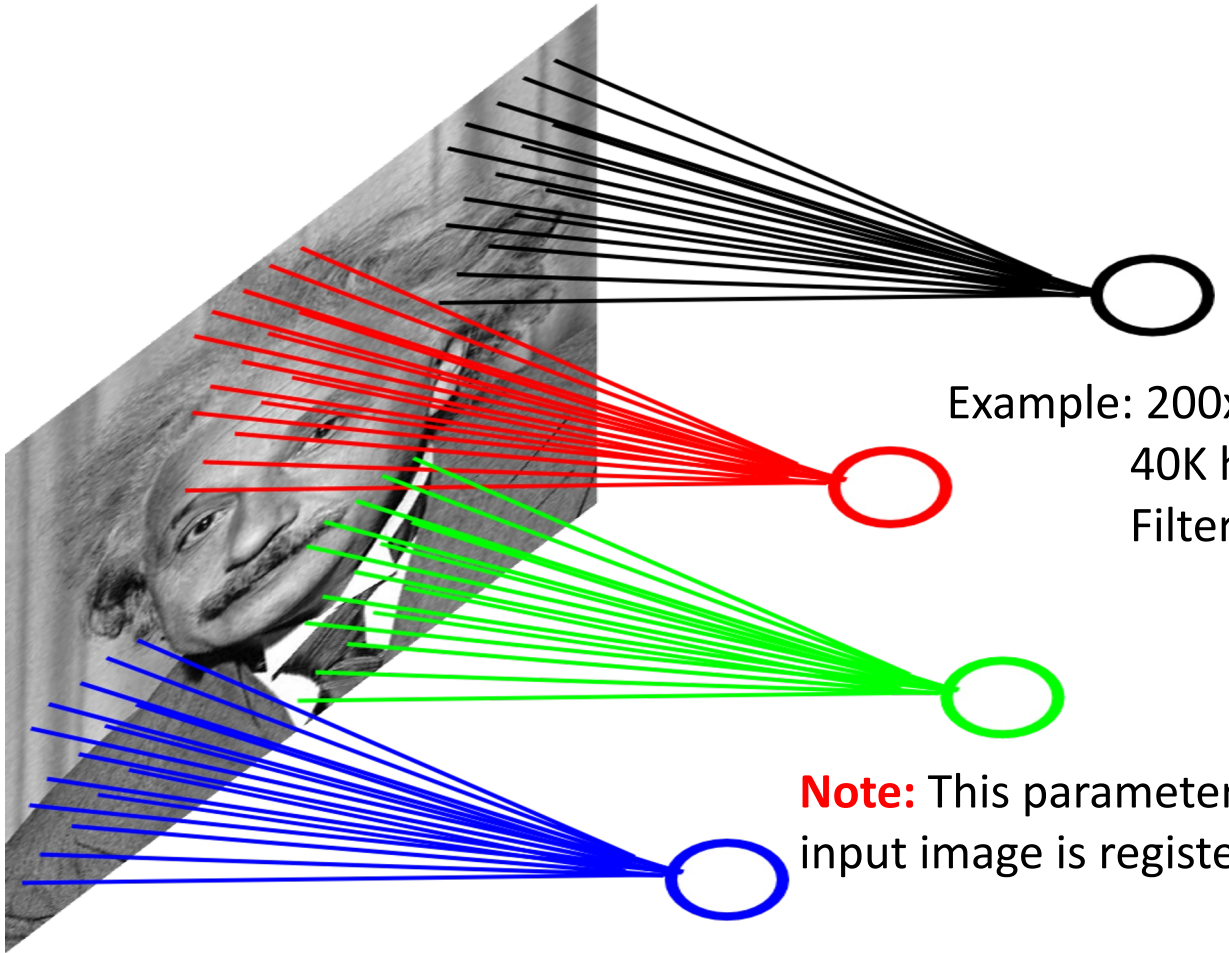
40K hidden units

~2B parameters!!!



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

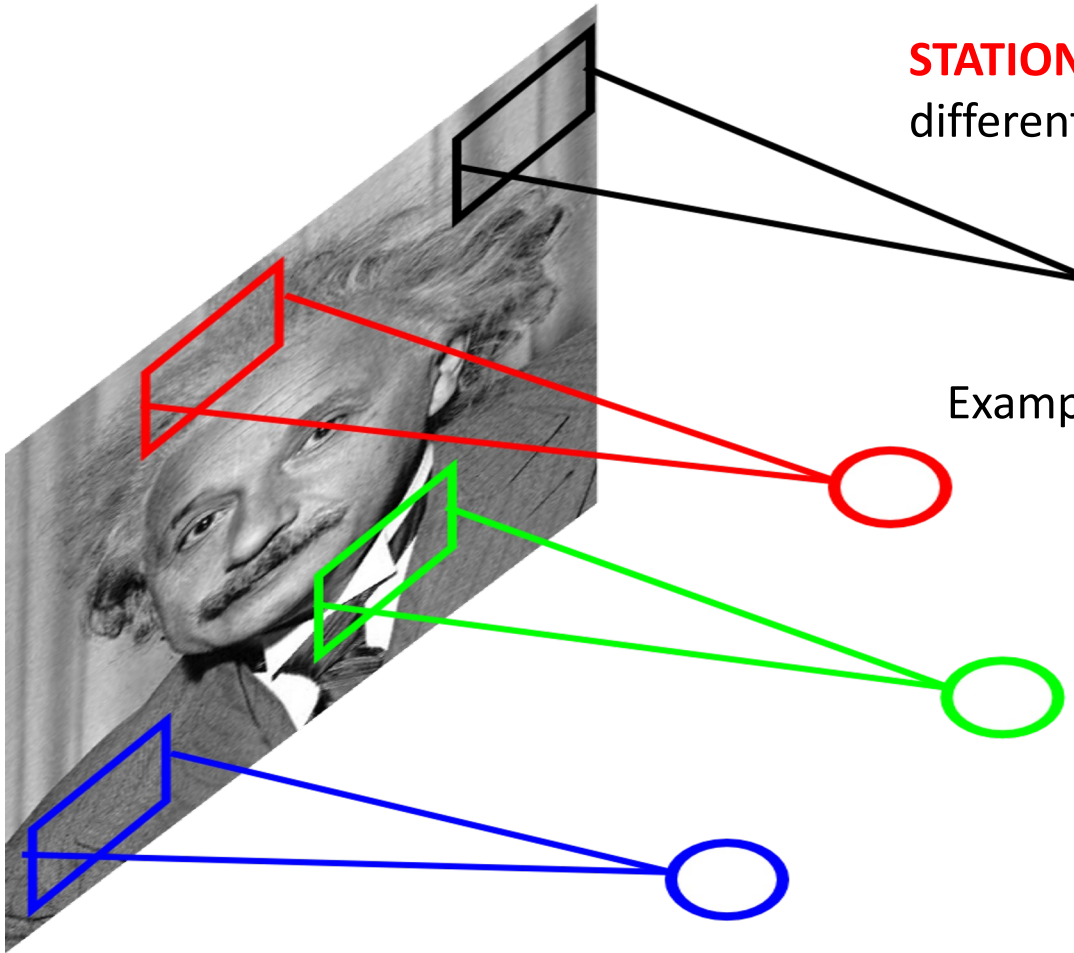
Locally Connected Layer



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

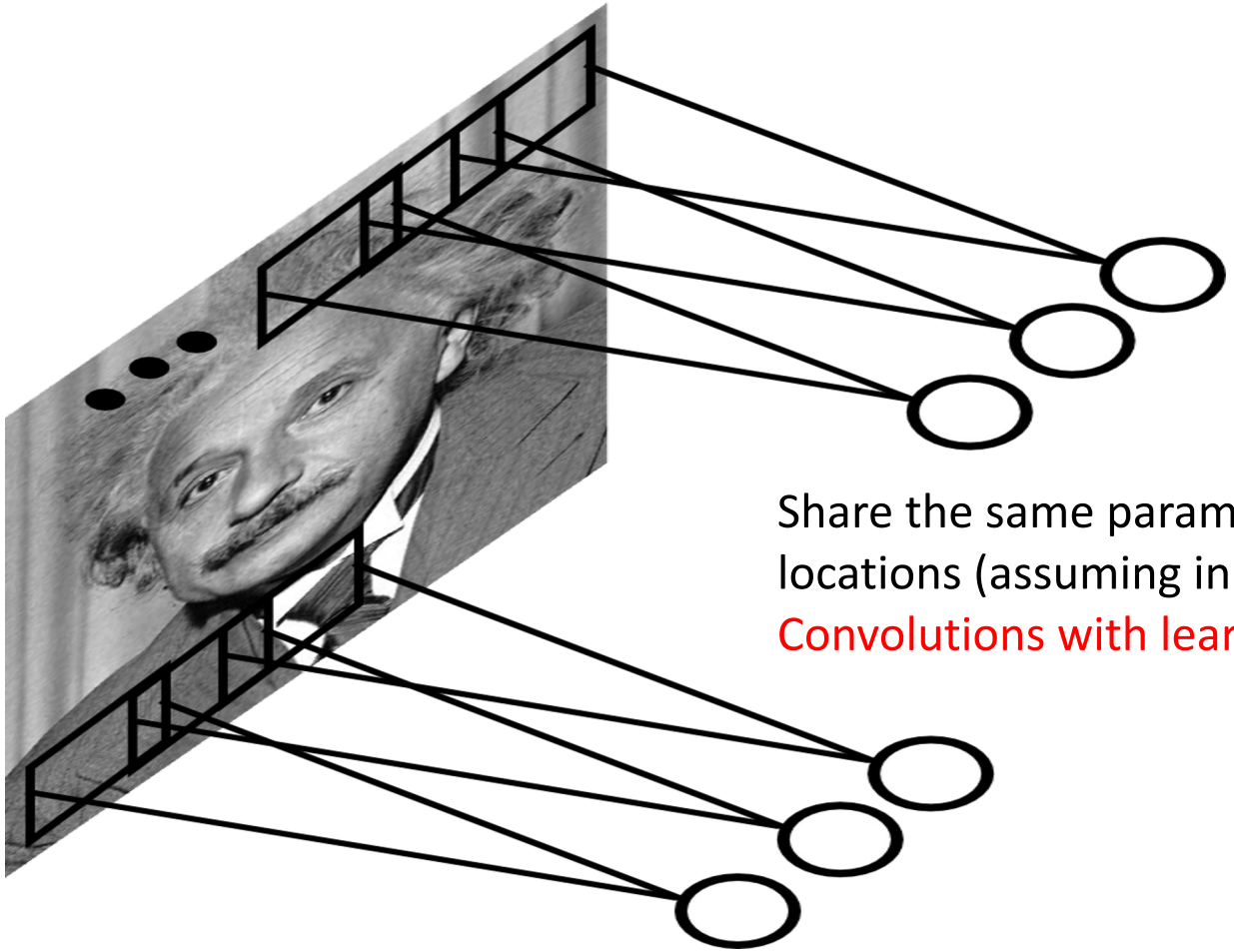
Example: 200x200 image

40K hidden units

Filter size: 10x10

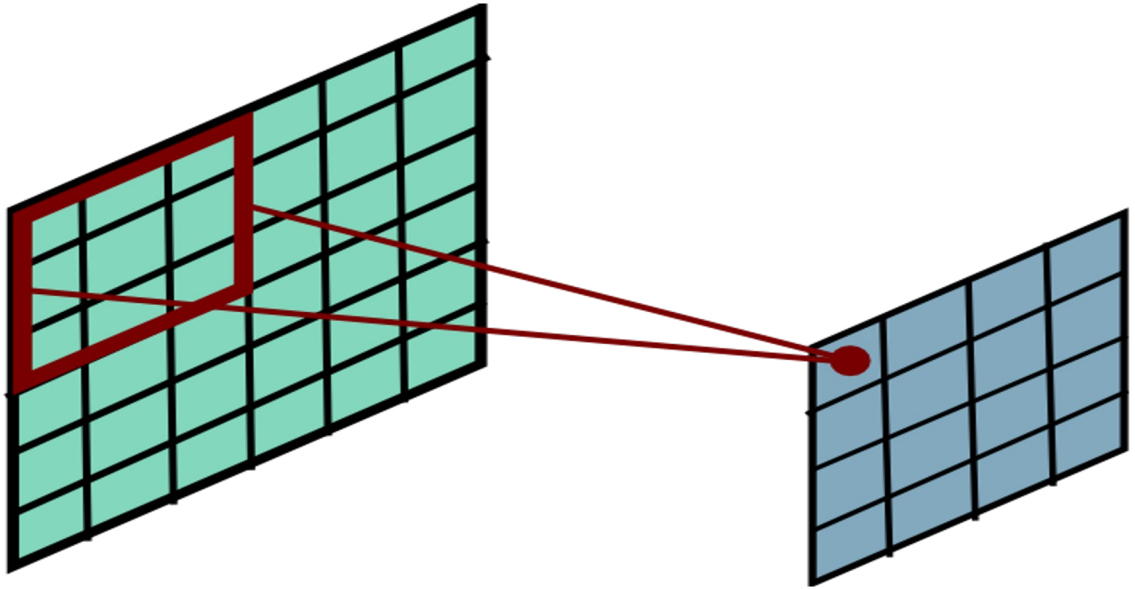
4M parameters

Convolutional Layer

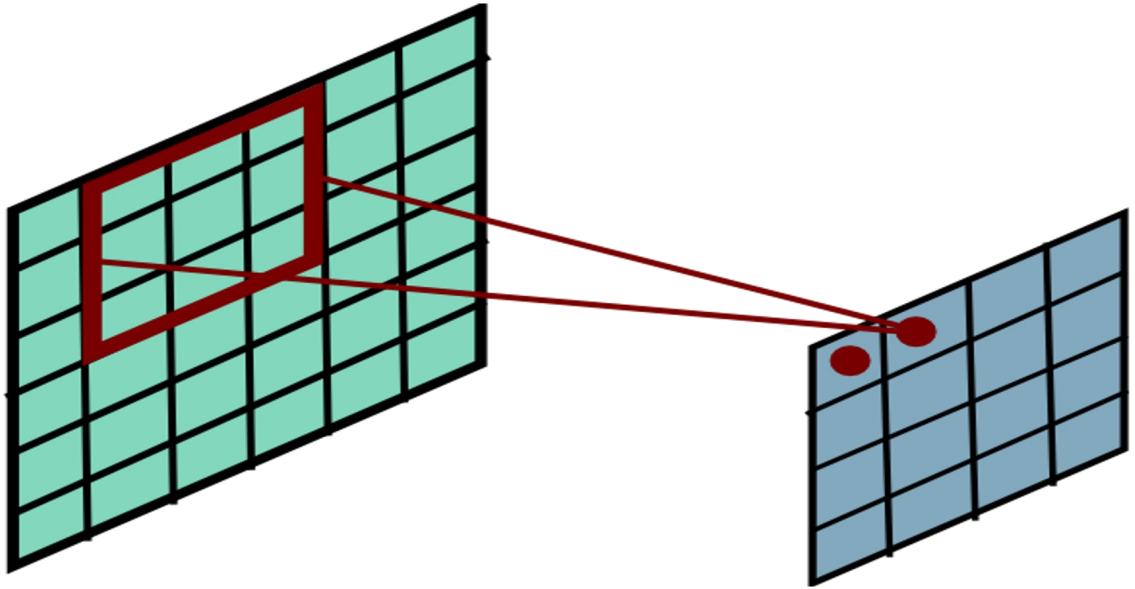


Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

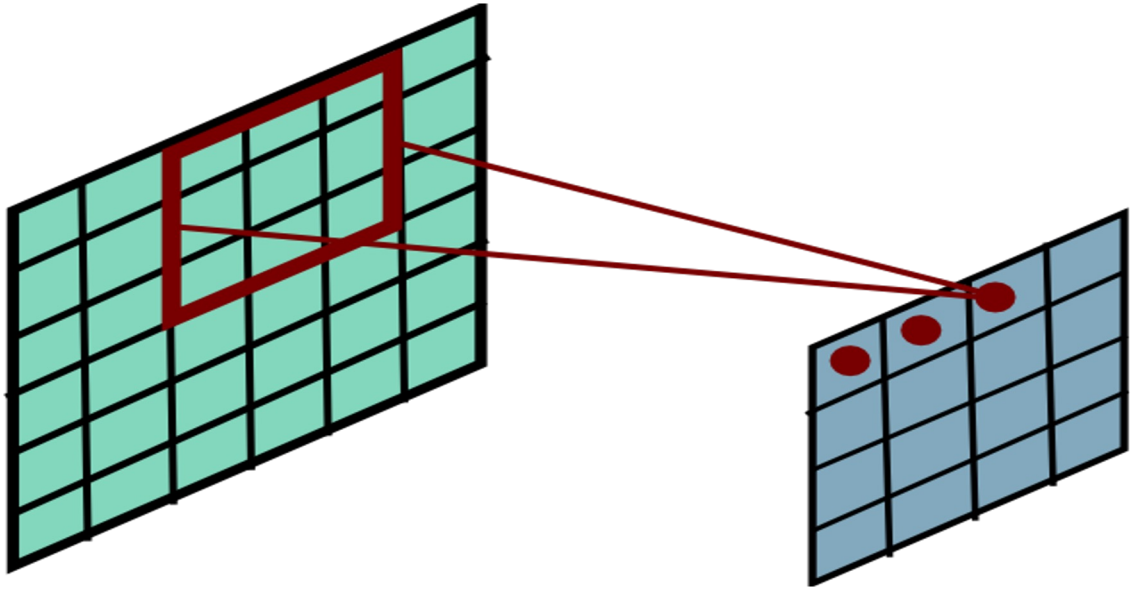
Convolutional Layer



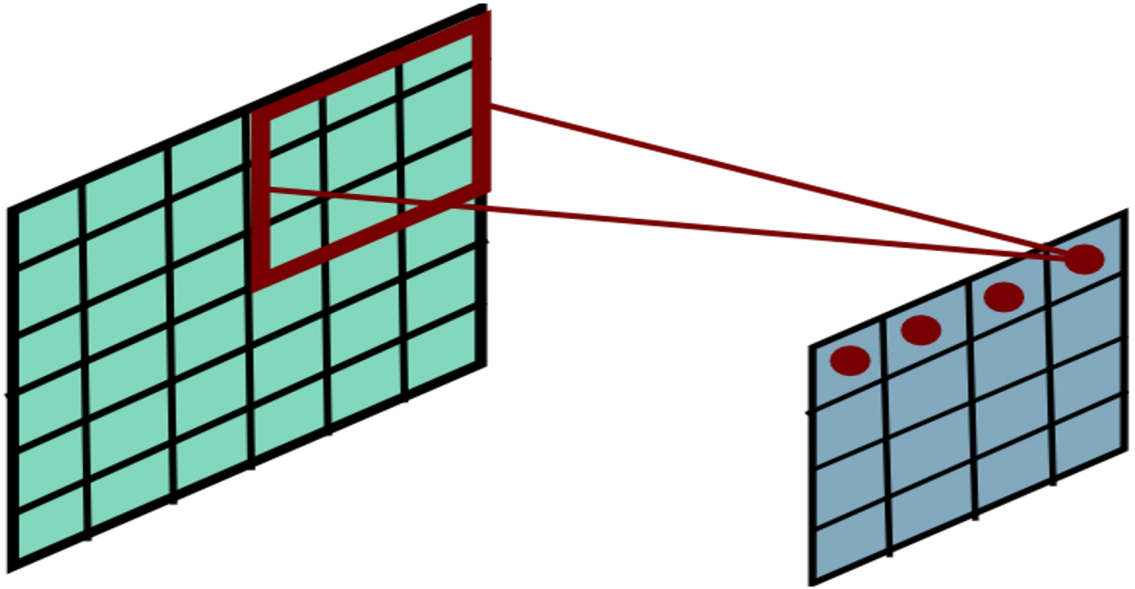
Convolutional Layer



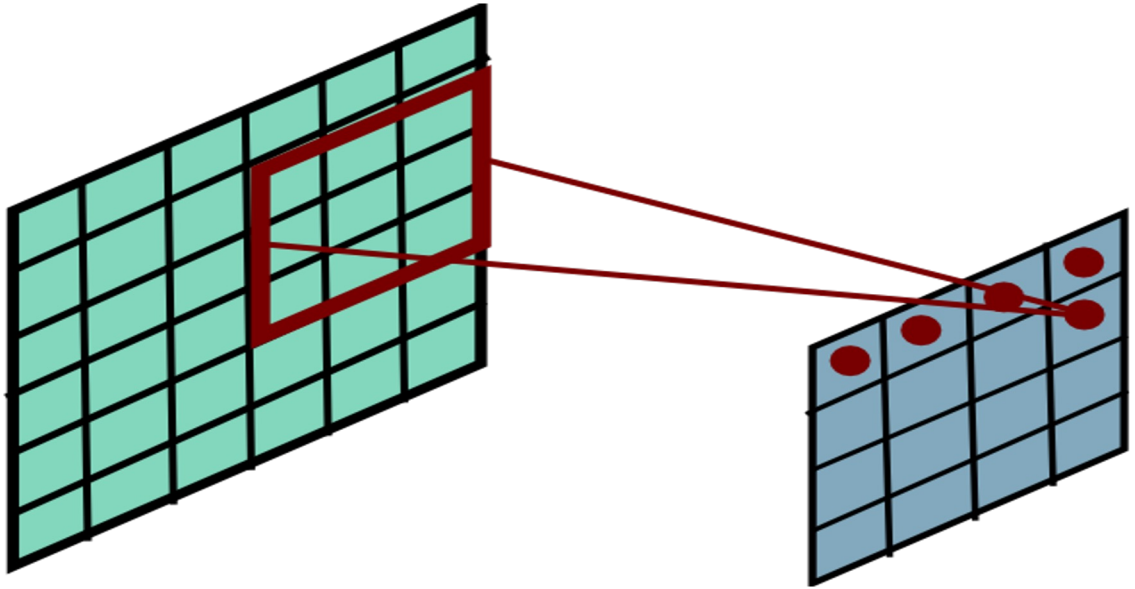
Convolutional Layer



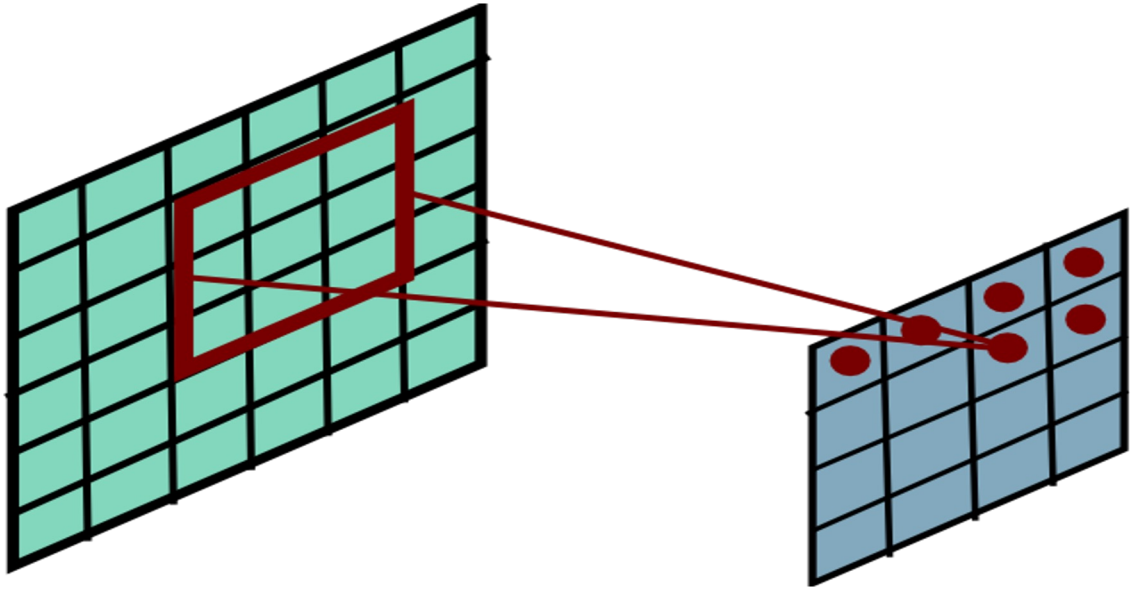
Convolutional Layer



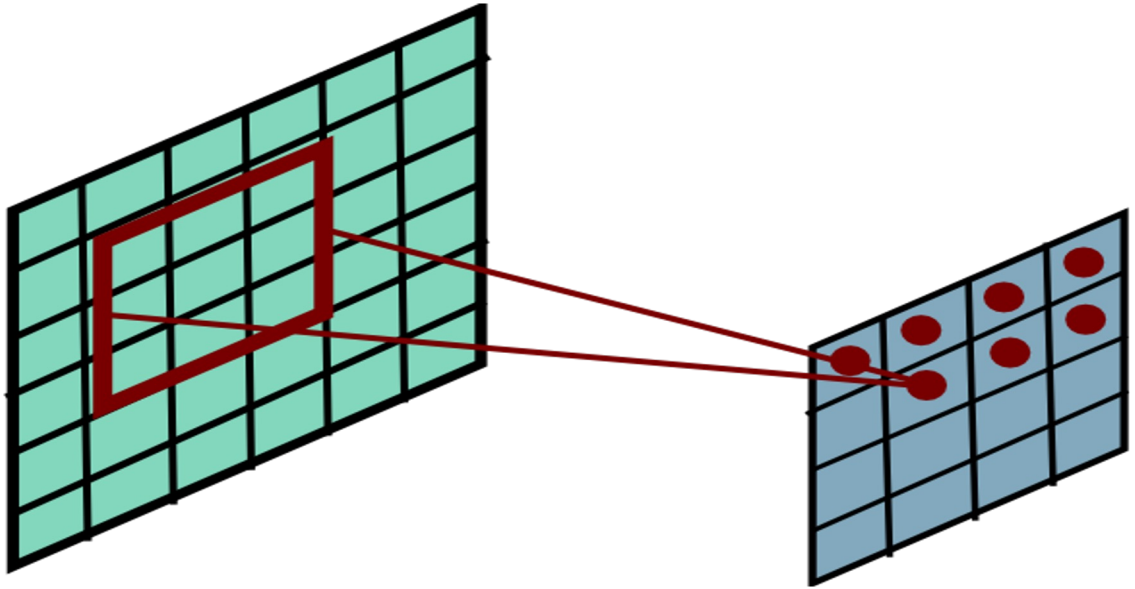
Convolutional Layer



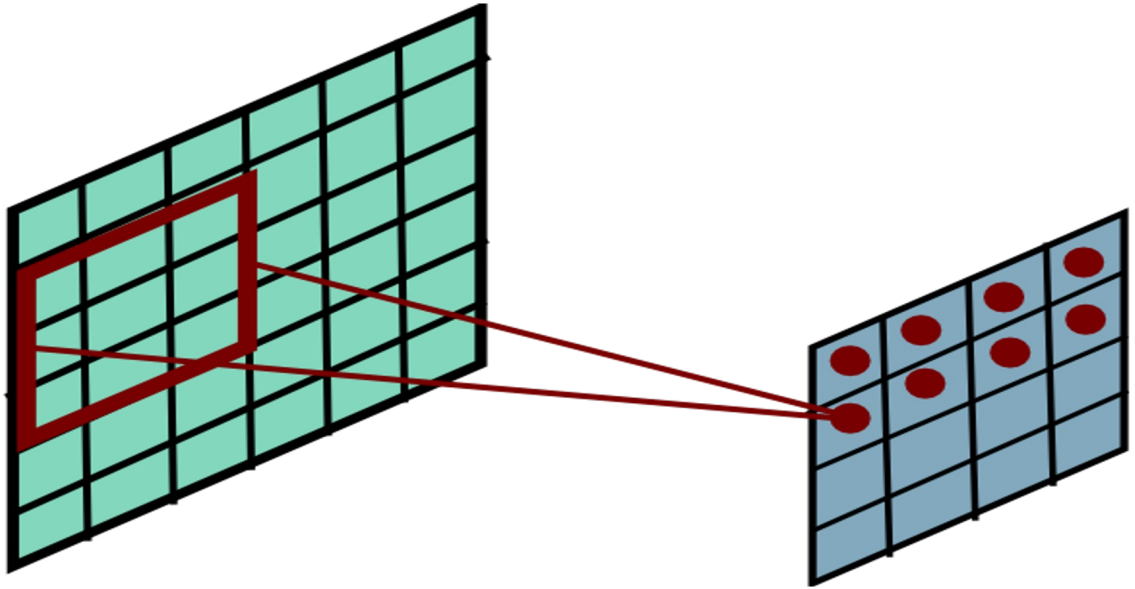
Convolutional Layer



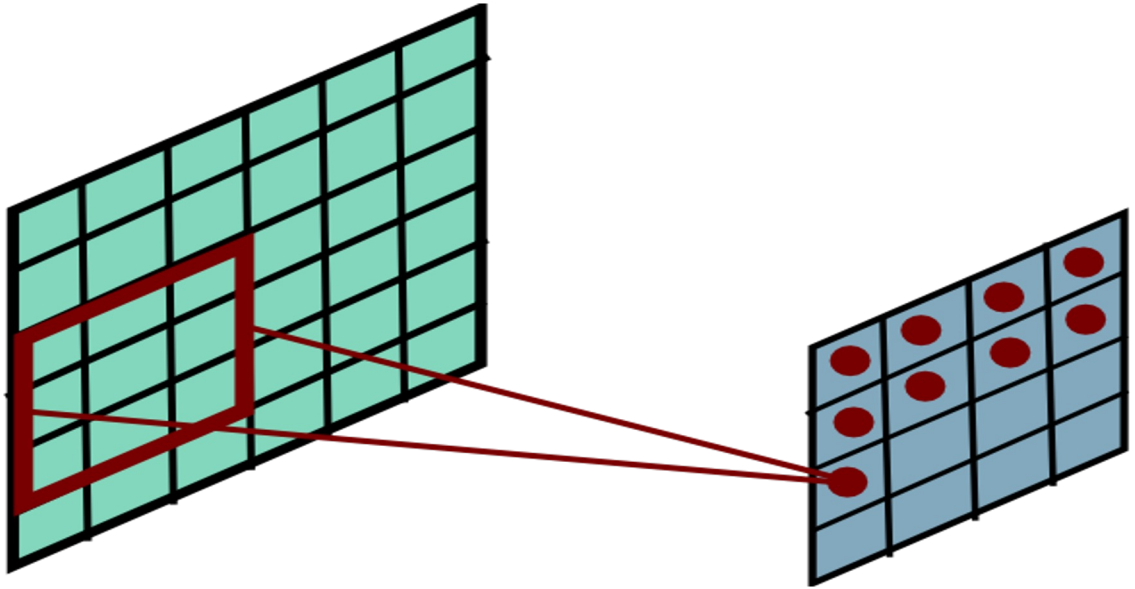
Convolutional Layer



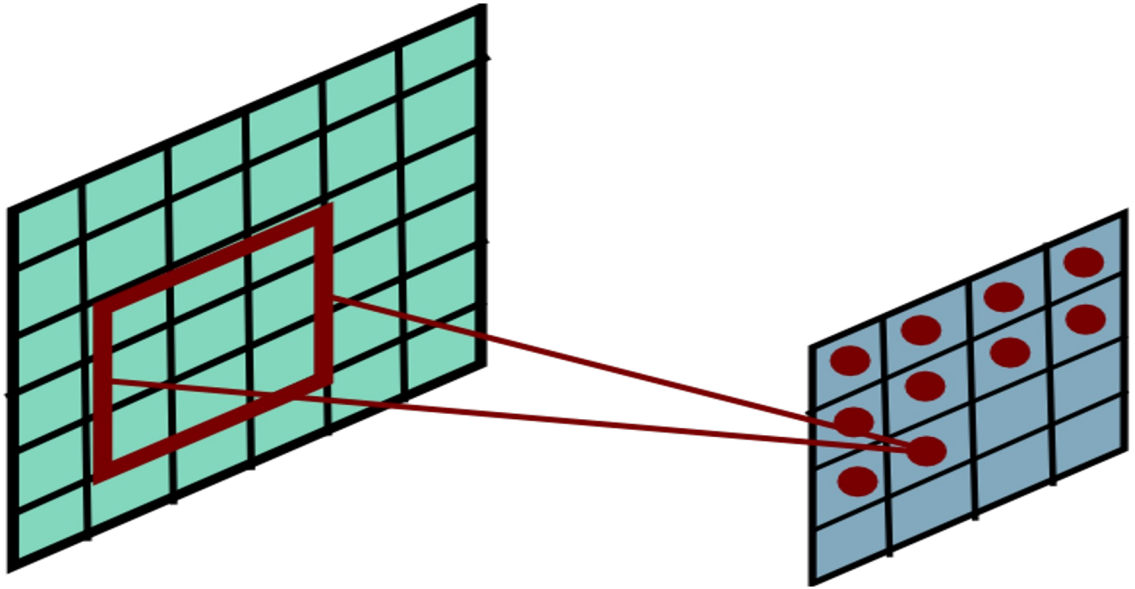
Convolutional Layer



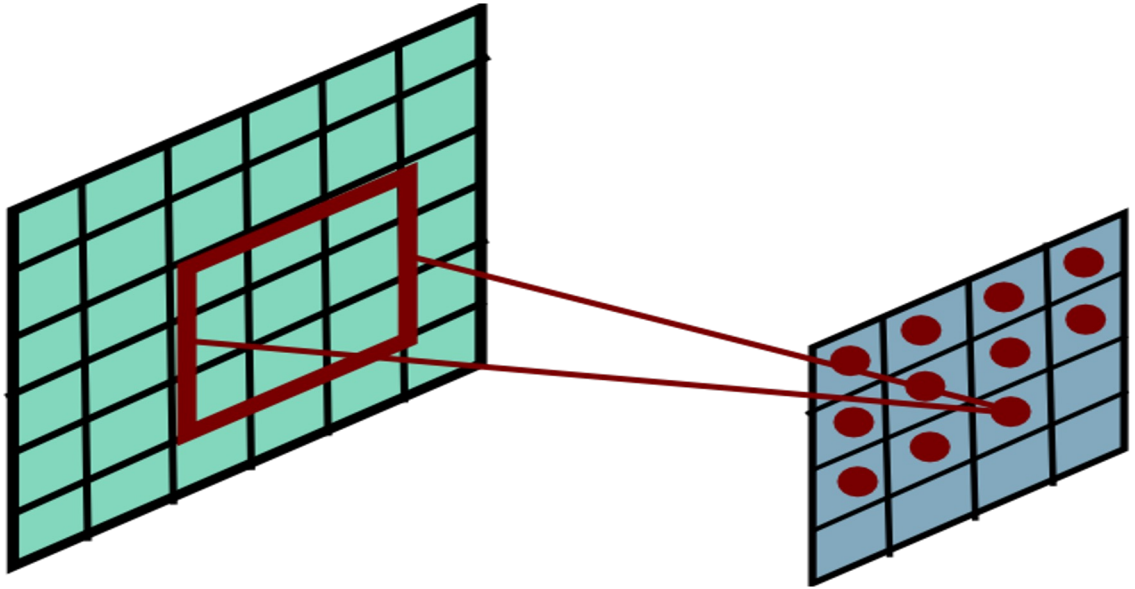
Convolutional Layer



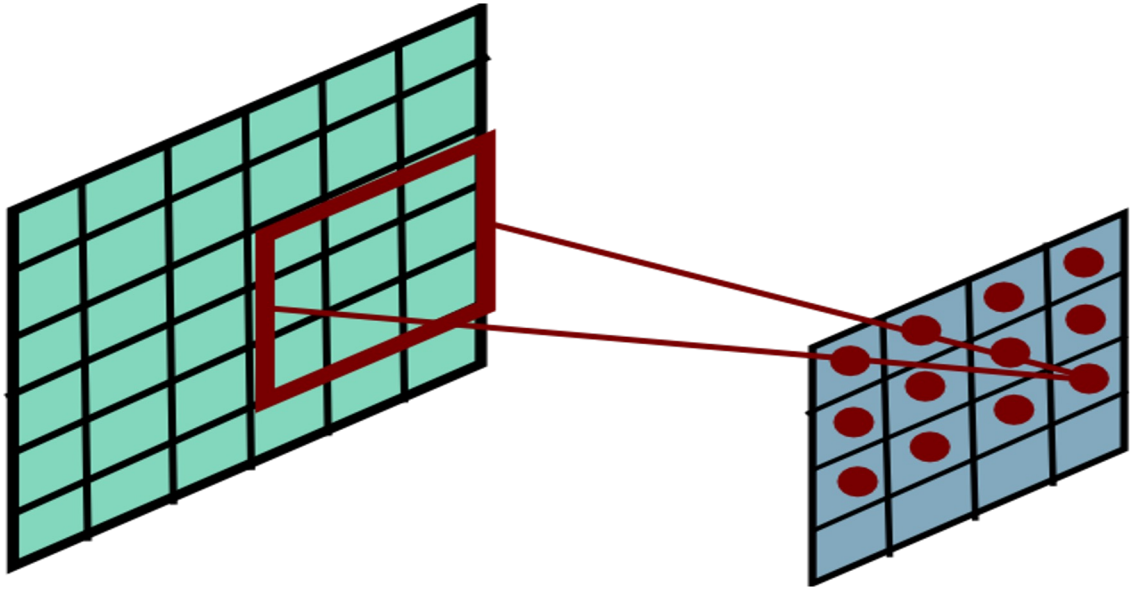
Convolutional Layer



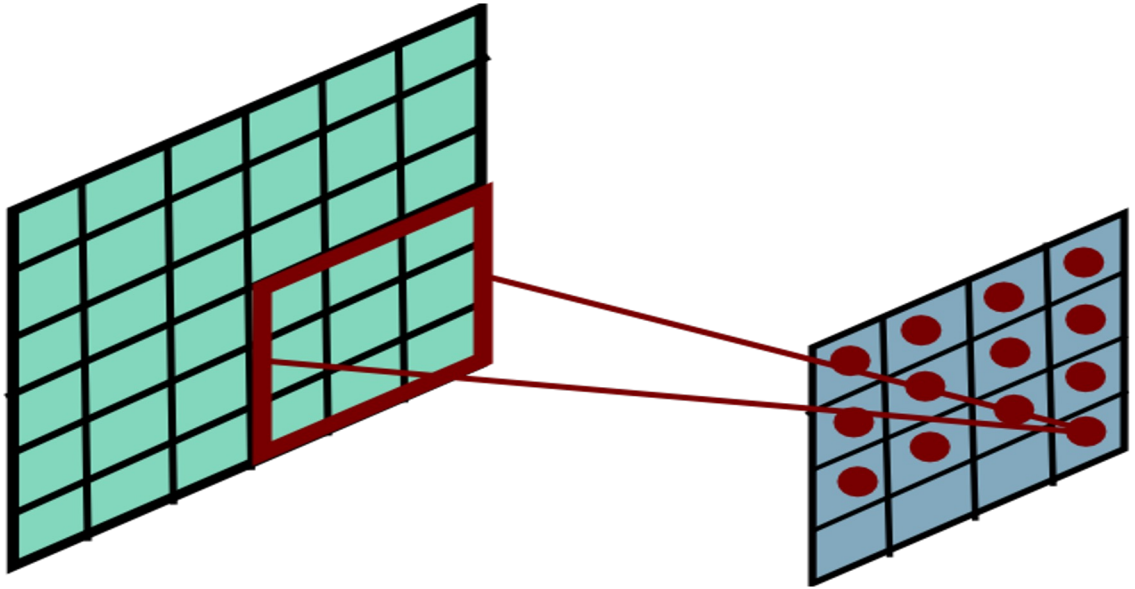
Convolutional Layer



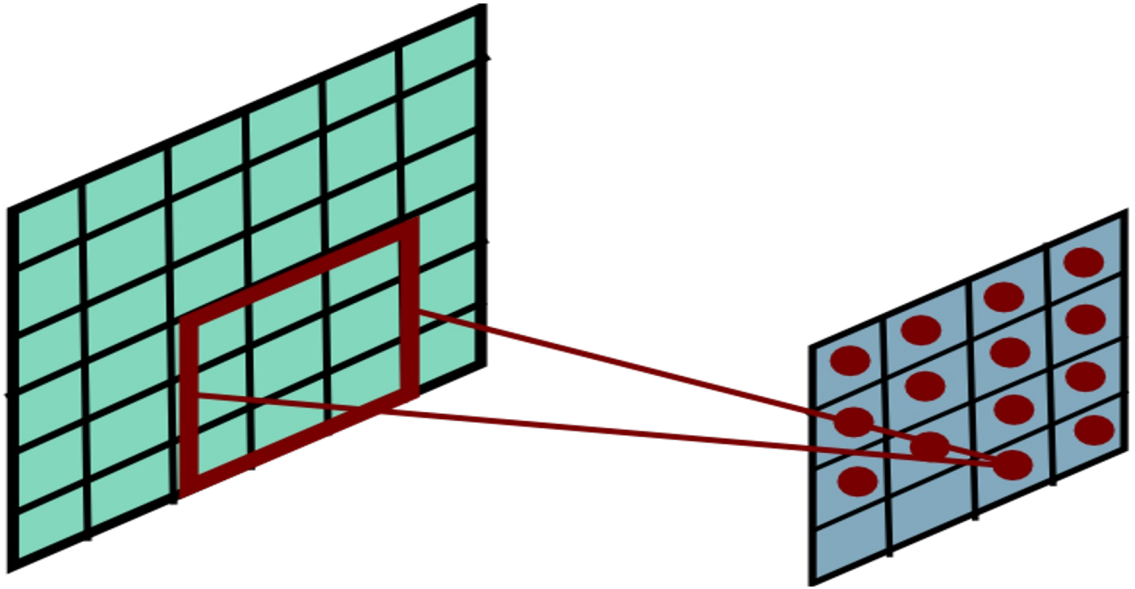
Convolutional Layer



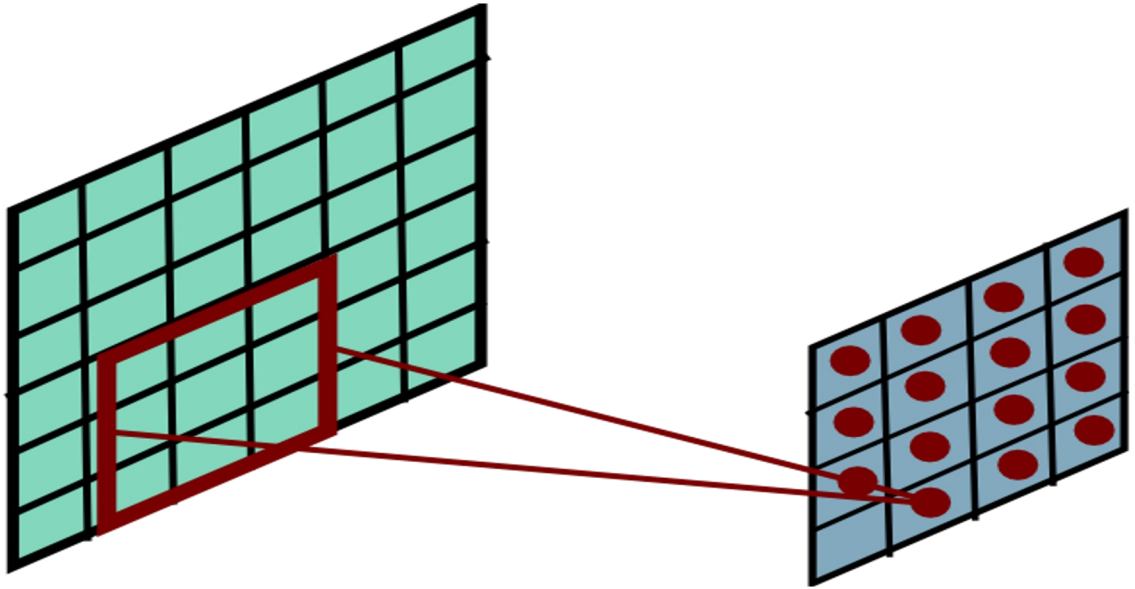
Convolutional Layer



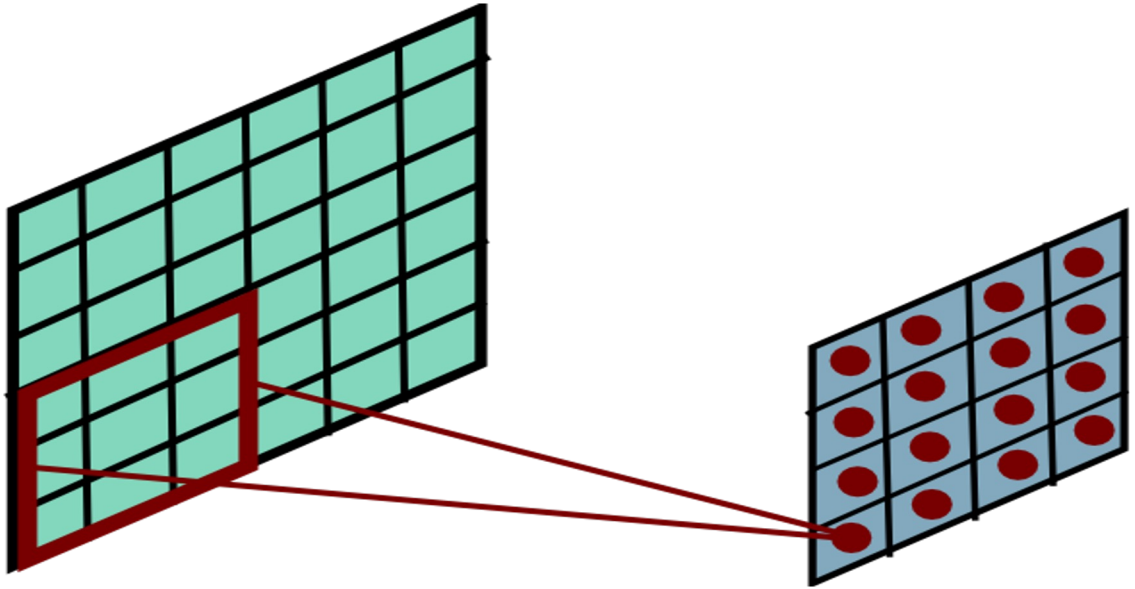
Convolutional Layer



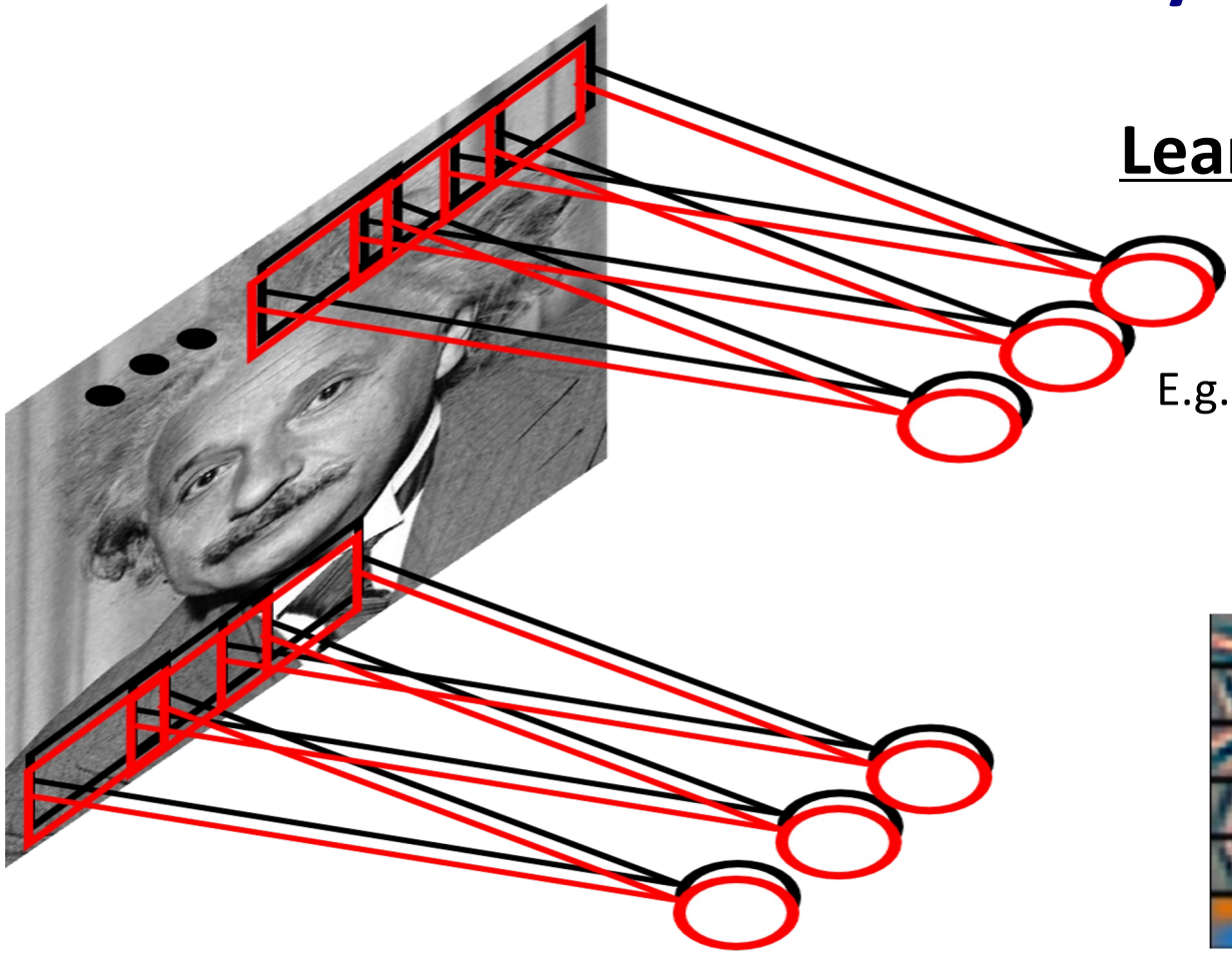
Convolutional Layer



Convolutional Layer



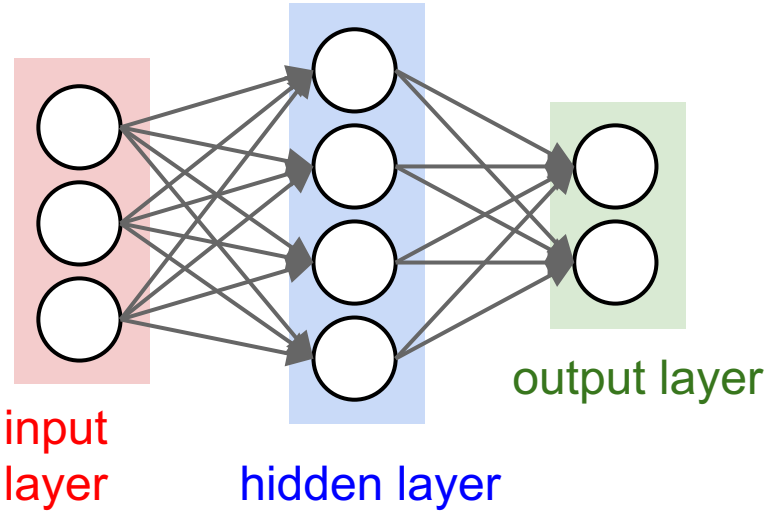
Convolutional Layer



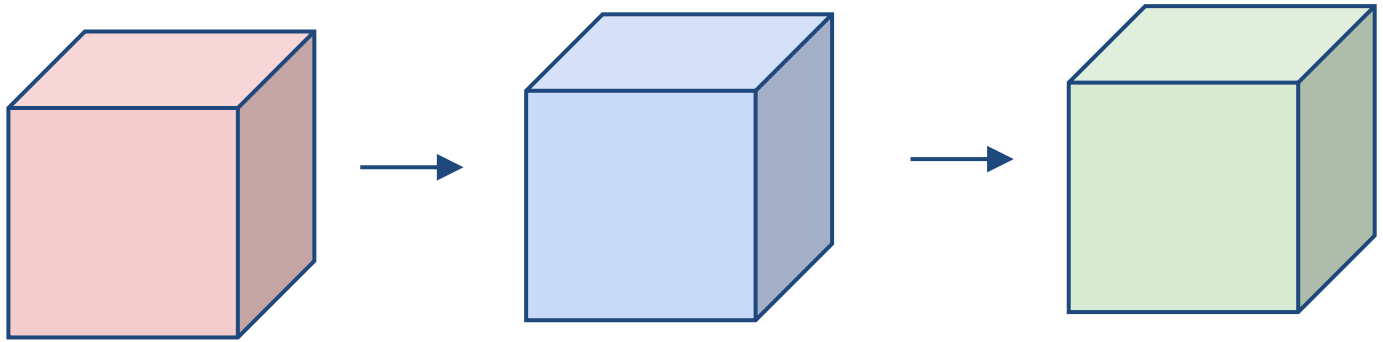
Learn multiple filters.

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

before:

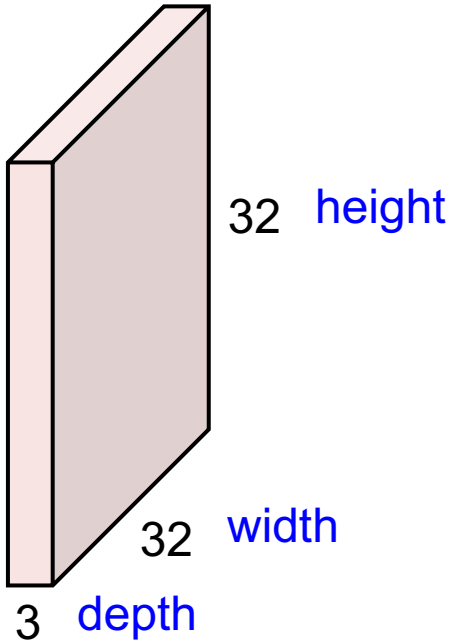


now:



Convolution Layer

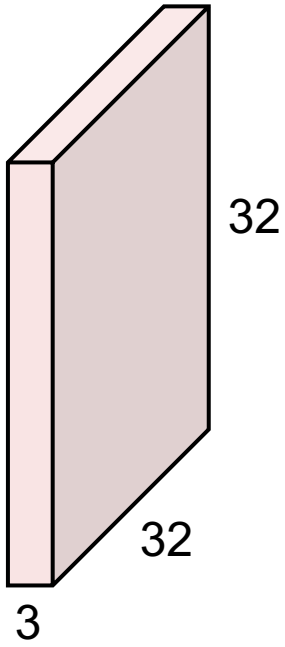
32x32x3 image



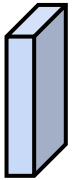
1

Convolution Layer

32x32x3 image



5x5x3 filter



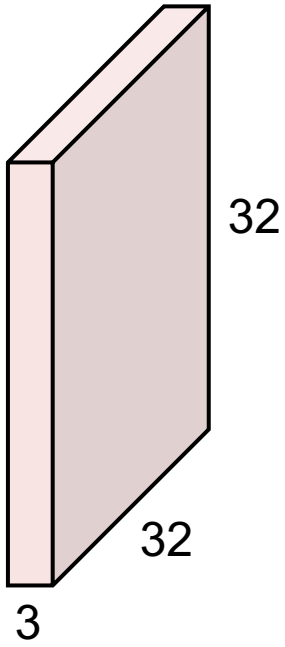
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

1

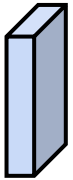
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



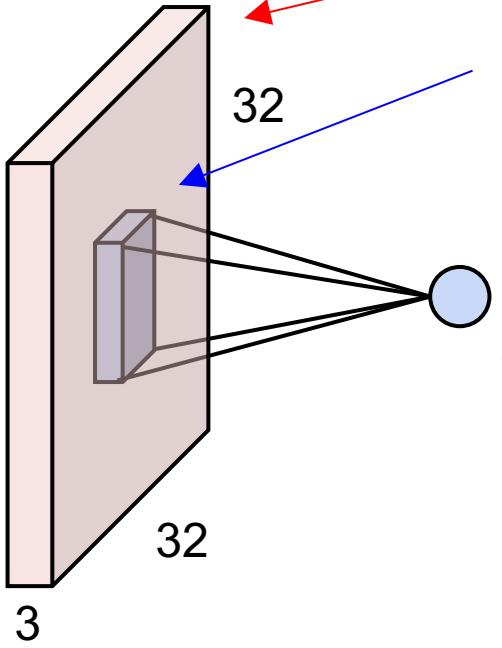
5x5x3 filter



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

1

Convolution Layer



32x32x3 image

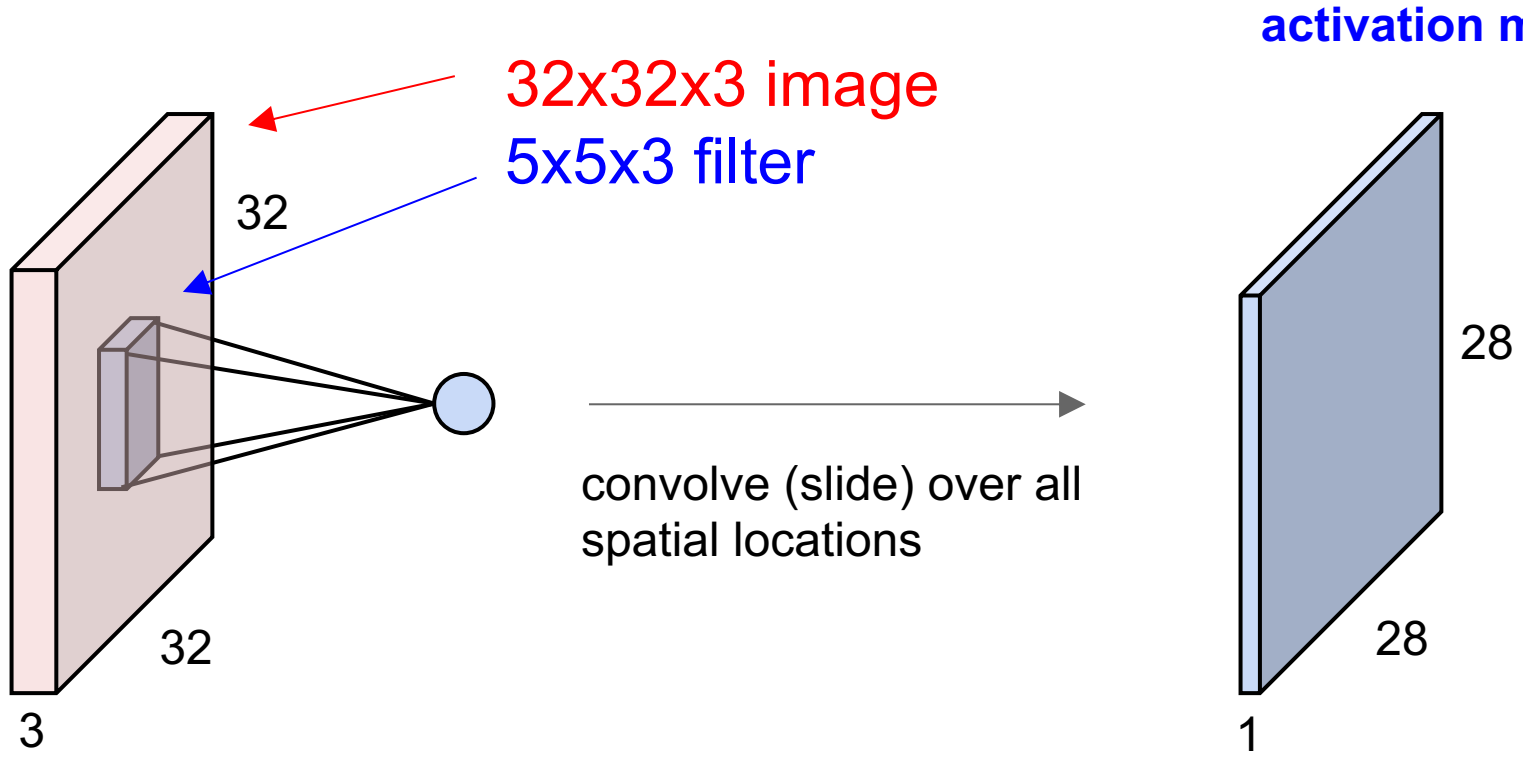
5x5x3 filter w

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)

$$w^T x + b$$

1

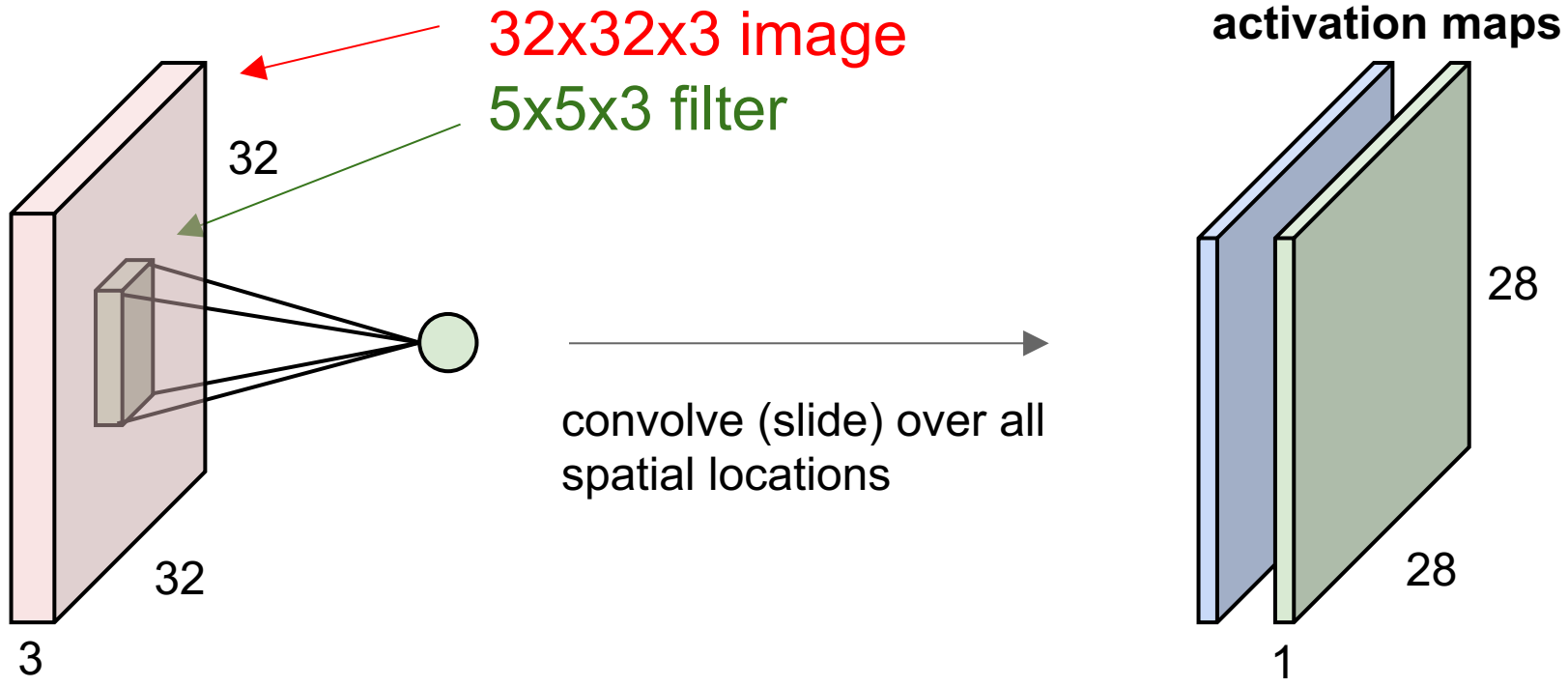
Convolution Layer



1

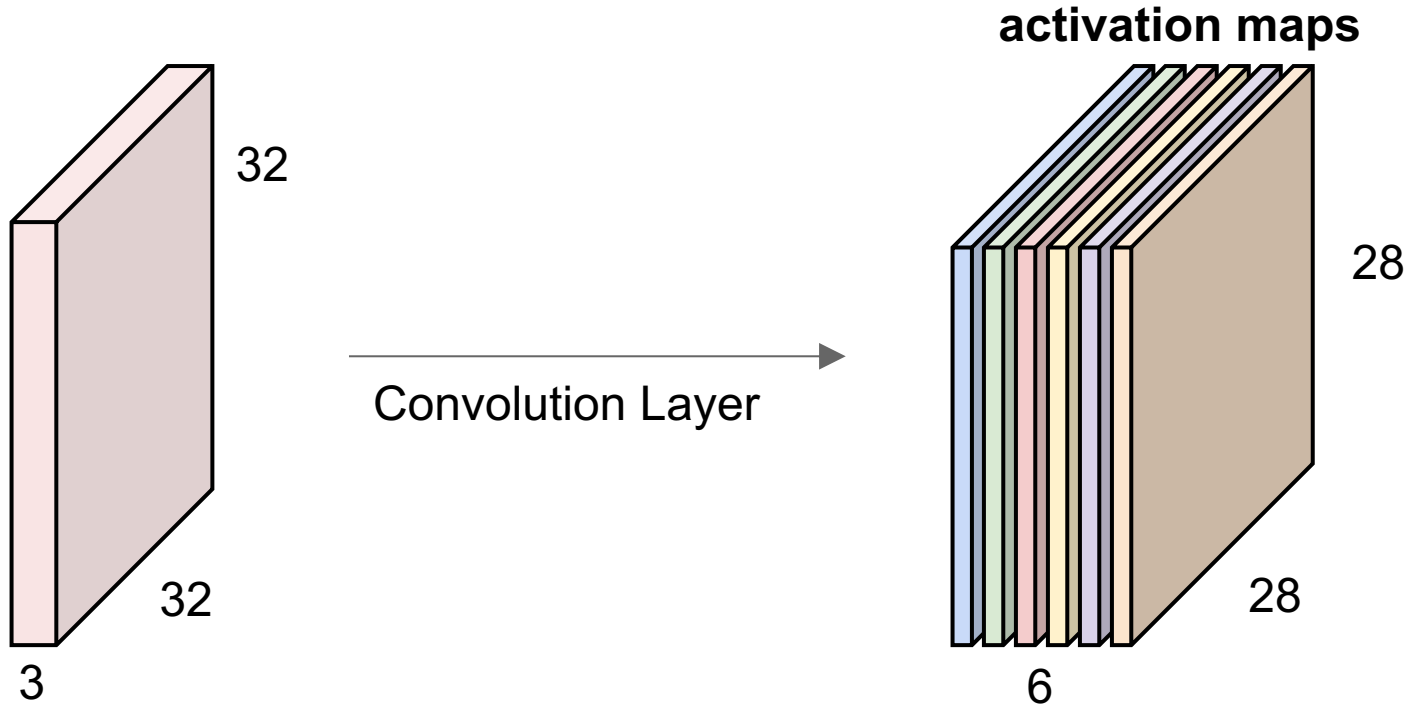
Convolution Layer

consider a second, green filter



1

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!