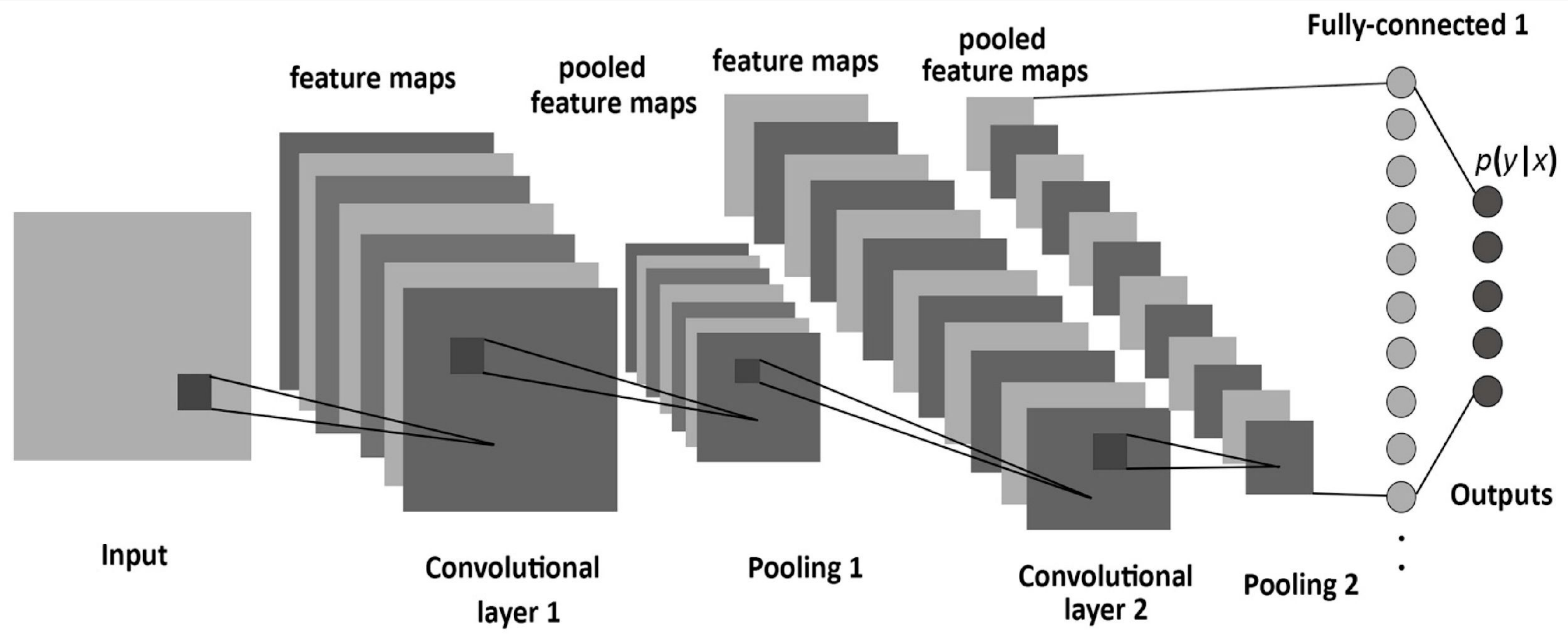
Lecture 10: Convolutional Neural Networks 1

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman and Chris Tanner

A Convolutional Network



+ ReLU

+ ReLU

The code

In [ ]:

```python

mnist_cnn_model = Sequential() # Create sequential model

# Add network layers
mnist_cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
mnist_cnn_model.add(MaxPooling2D((2, 2)))
mnist_cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
mnist_cnn_model.add(MaxPooling2D((2, 2)))
mnist_cnn_model.add(Conv2D(64, (3, 3), activation='relu'))

mnist_cnn_model.add(Flatten())
mnist_cnn_model.add(Dense(64, activation='relu'))

mnist_cnn_model.add(Dense(10, activation='softmax'))

mnist_cnn_model.compile(optimizer=optimizer,
              loss=loss,
              metrics=metrics)

history = mnist_cnn_model.fit(train_images, train_labels,
                              epochs=epochs,
                              batch_size=batch_size,
                              verbose=verbose,
                              validation_split=0.2,
                              # validation_data=(X_val, y_val) # IF you have val data
                              shuffle=True)
```

DONE

Outline

1. Motivation
2. CNN basic ideas
3. Building a CNN

Outline

1. **Motivation**
2. CNN basic ideas
3. Building a CNN

Feedforward Neural Network, Multilayer Perceptron (MLP)

We assume that the response variable, Y, relates to the predictors, X, through some unknown function expressed generally as:

$$Y = f(X) + \varepsilon$$

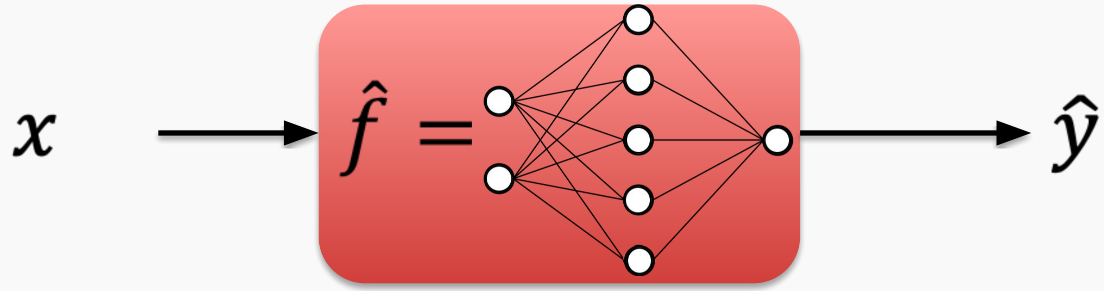
Here, f is the unknown function expressing an underlying rule for relating Y to X, ε is the random amount (unrelated to X) that Y differs from the rule $f(X)$.

A ***statistical model*** is any algorithm that estimates f. We denote the estimated function as $\widehat{f}$.

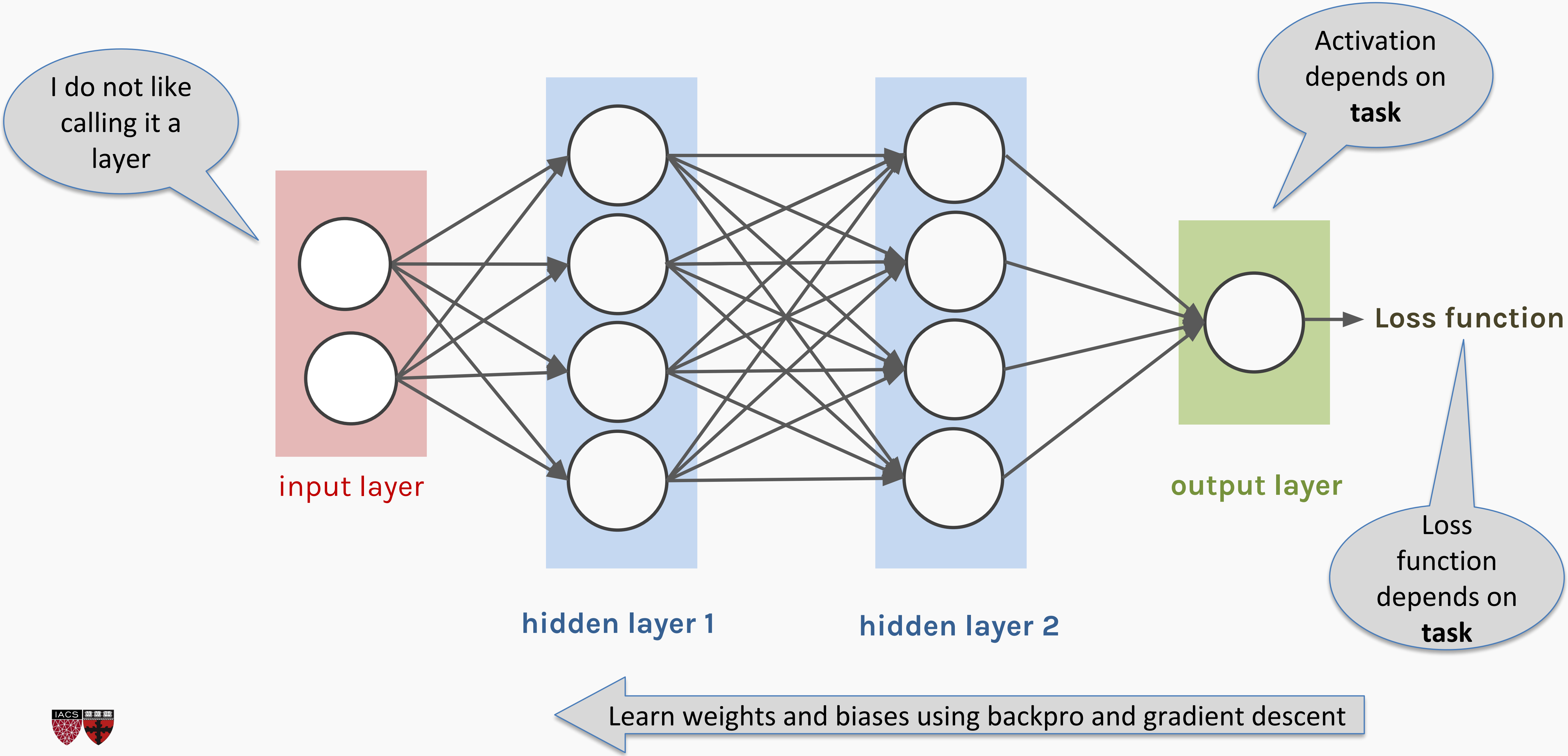
Feedforward Neural Network, Multilayer Perceptron (MLP)

A **function** is a relation that associates each element x of a set X to a single element y of a set Y

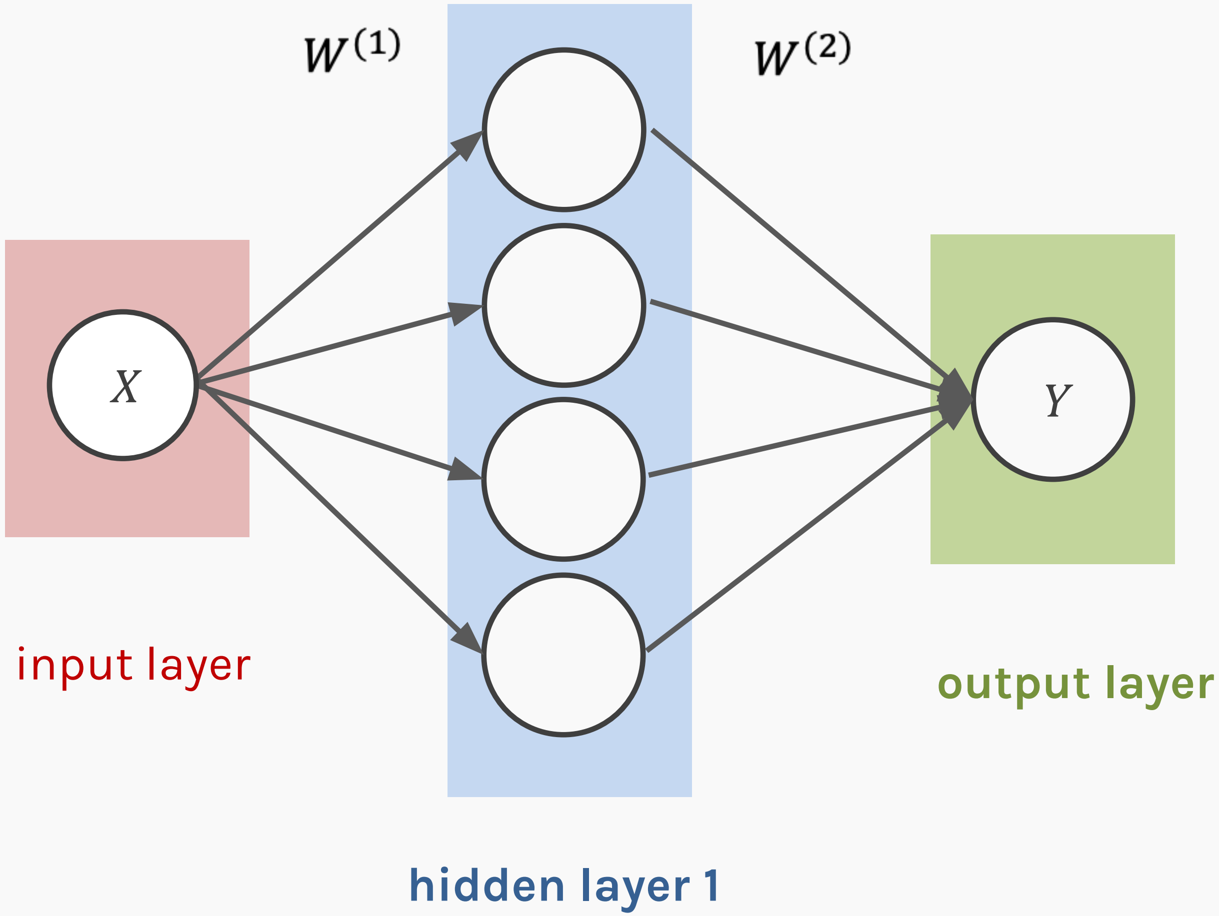
Neural networks can approximate a wide variety of functions



Quick review of MLPs



MLP as an additive model



activation

$$Y = \sum_j W_j^{(2)} \underbrace{f(W_j^{(1)} X + b_j)}_{\text{Basis functions.}}$$

Y is a linear combination of these basis functions.

We learn the coefficients of the basis functions $W_j^{(2)}$ as well as the parameters of the basis functions $(W_j^{(1)}, \beta\_j)$

If activation is ReLU then b_j's are the locations of the *knots*.

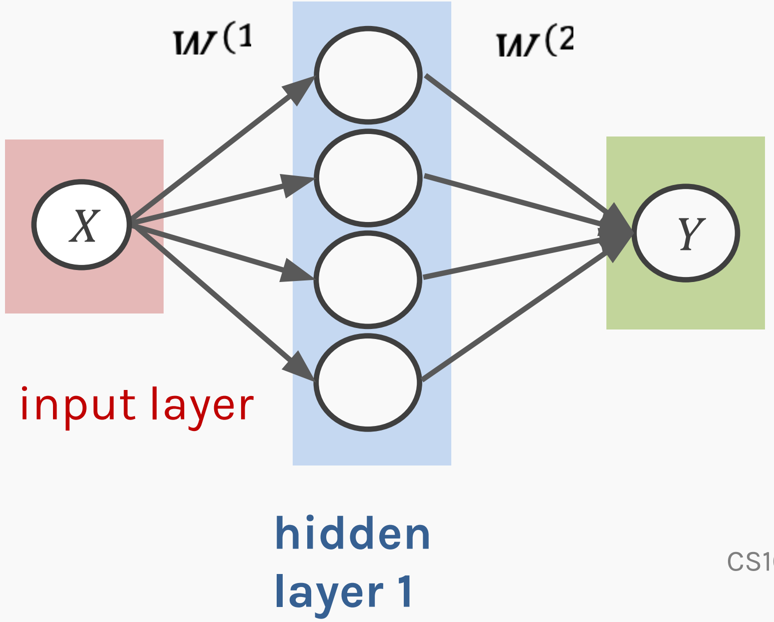
MLP as an additive model (cont)

From lecture 1:

$$E(Y|x) = \alpha_0 + \alpha_1 x + \beta_1 (x - \xi_1)_+ + \beta_2 (x - \xi_2)_+ + \cdots + \beta_k (x - \xi_k)_+$$

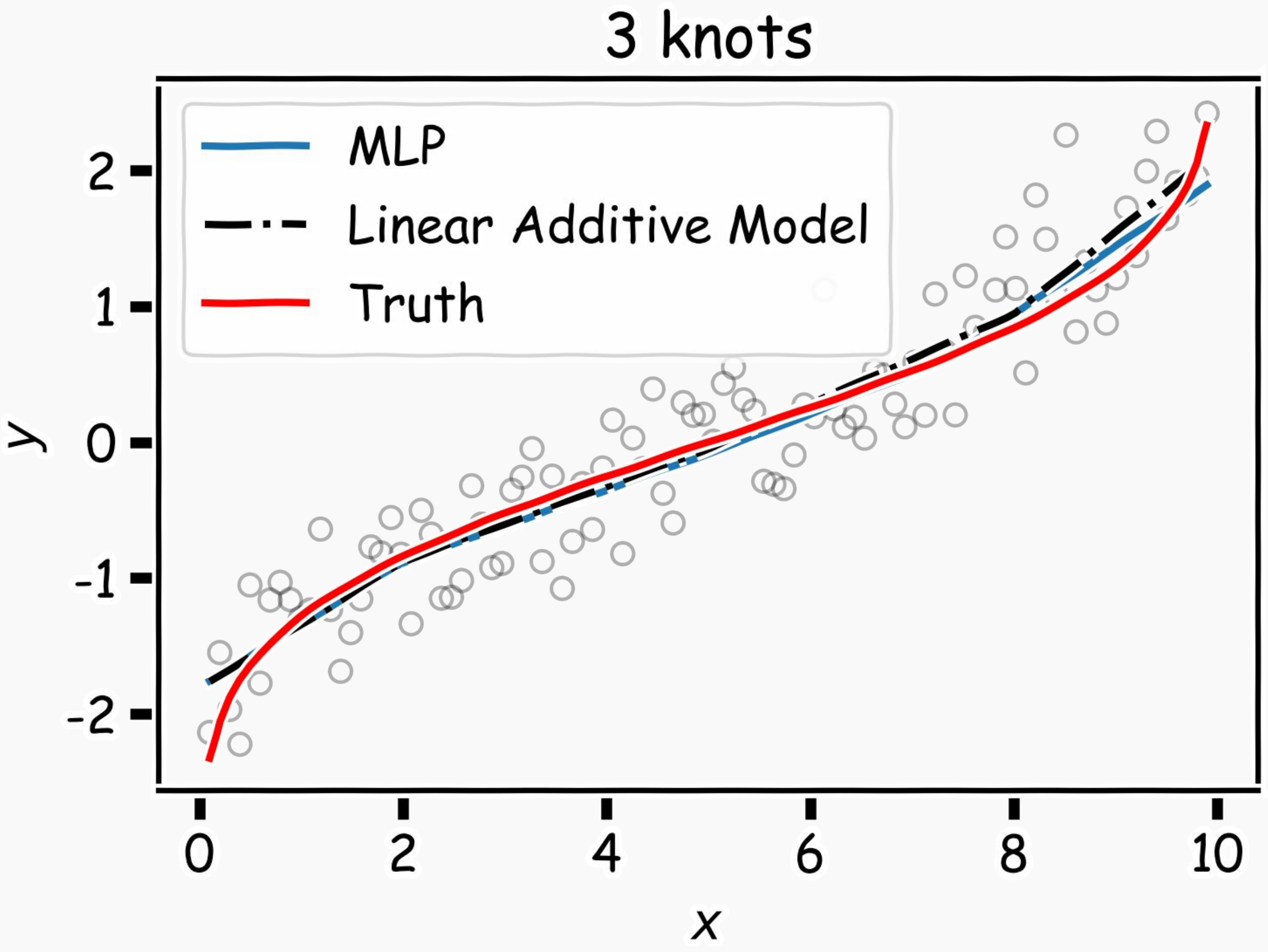
Minor modification:

$$E(Y|x) = \alpha_0 + \beta_0 (x - \infty)_+ + \beta_1 \underbrace{(x - \xi_1)_+}_{\mathrm{ReLU}(Wx + \xi_1) \text{ where } W = 1} + \beta_2 (x - \xi_2)_+ + \cdots + \beta_k (x - \xi_k)_+$$



Location of Knots can be learned as well as the β's and α_0

MLP as an additive model (cont)



MLP:

$$\xi_1 = 1.98248$$
$$\xi_2 = 5.03615$$
$$\xi_3 = 7.91110$$

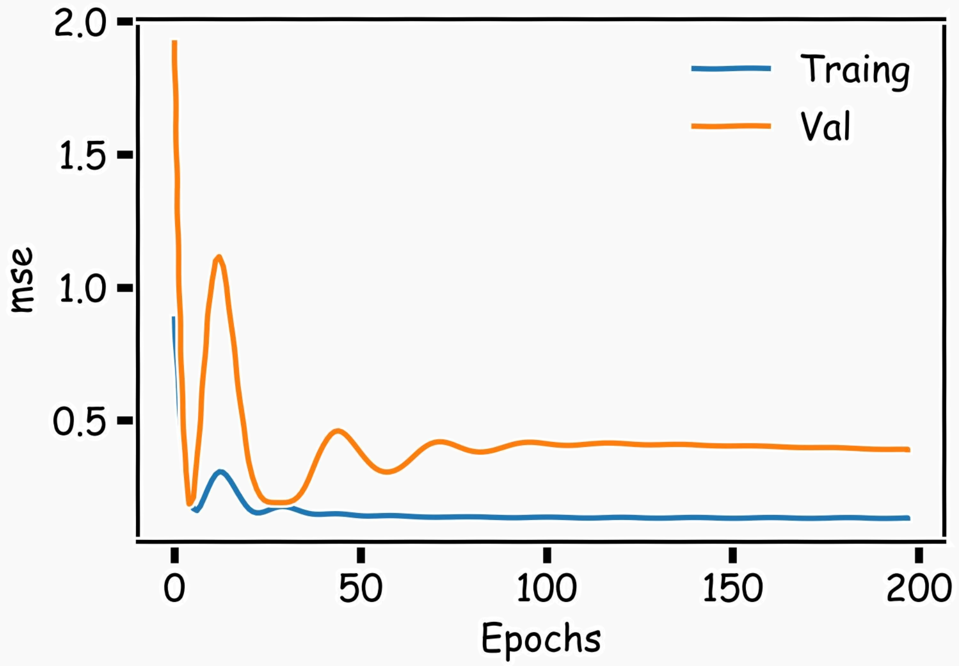
MLP as an additive model (cont)

If we add more neurons, it clearly overfits.

CS109A Lecture 21:

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout



Main **drawbacks** of MLPs

- MLPs use one node for each input (e.g. pixel in an image, multiplied by 3 in RGB case). The amount of weights rapidly becomes unmanageable for large images.
- Training difficulties arise, overfitting can appear.

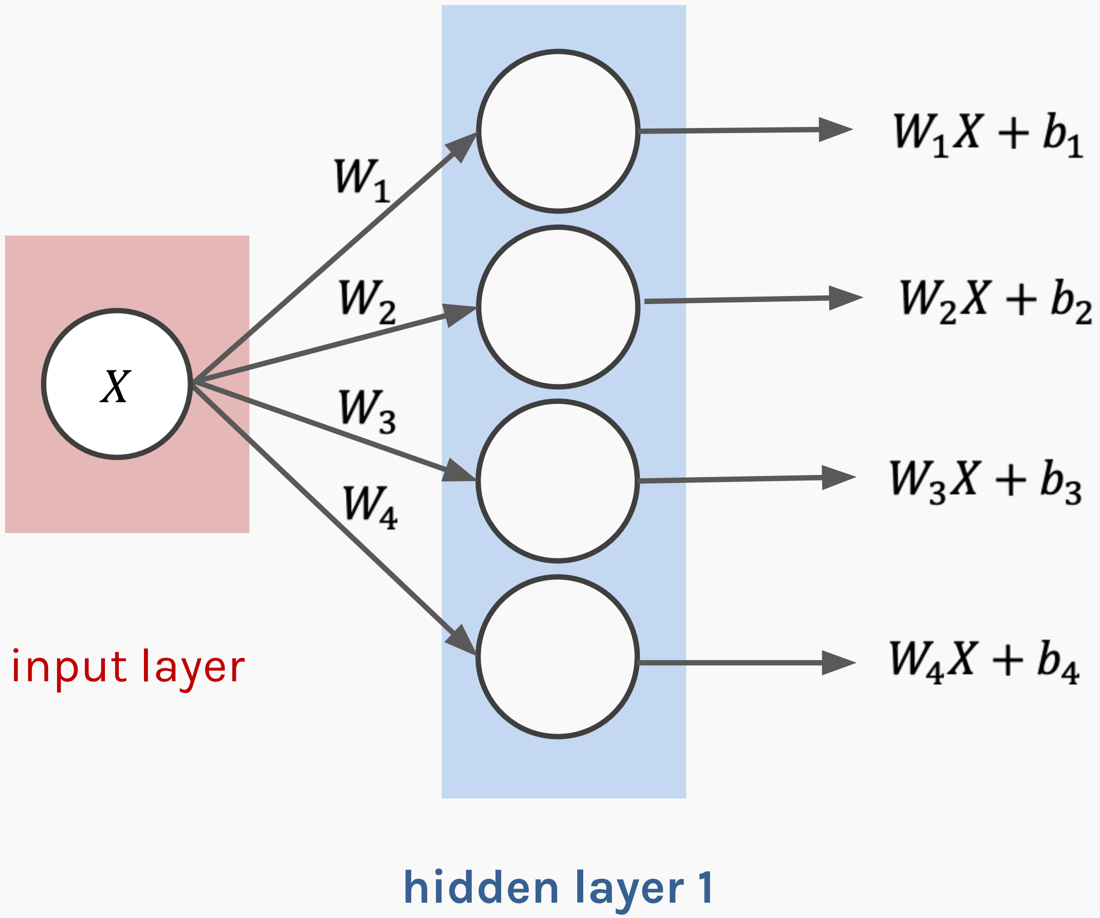
- MLPs react differently to an input (images) and its shifted version – they are not translation invariant.

Main **drawbacks** of MLPs

- **MLPs use one node for each input (e.g. pixel in an image, multiplied by 3 in RGB case). The amount of weights rapidly becomes unmanageable for large images.**
- **Training difficulties arise, overfitting can appear.**

- MLPs react differently to an input (images) and its shifted version – they are not translation invariant.

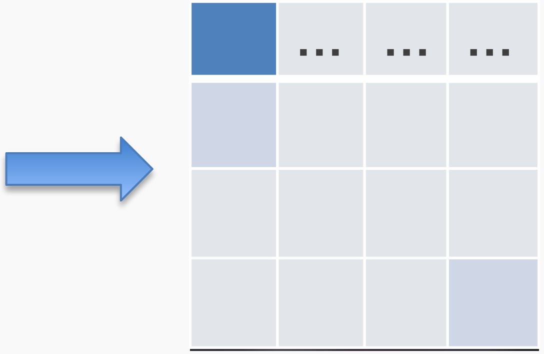
MLP: number of weights



How many weights?

- If $X \in \mathbb{R}$ then $W_n \in \mathbb{R}$
- If $X \in \mathbb{R}^m$ then $W_n \in \mathbb{R}^m$

MLP: number of weights for images



If we consider each pixel as an independent predictor, then $X \in \mathbb{R}^{4x4}$ or 16 predictors, and therefore 16 weights for each node in the fist hidden layer.

From 109A Lecture 7:

A strong motivation for performing model selection is to avoid overfitting, which we saw can happen when:

- **there are too many predictors:**
  - the feature space has high dimensionality
  - the polynomial degree is too high
  - too many cross terms are considered

MLP: number of weights for images

Example: CIFAR10

Simple 32x32 color images (3 channels)

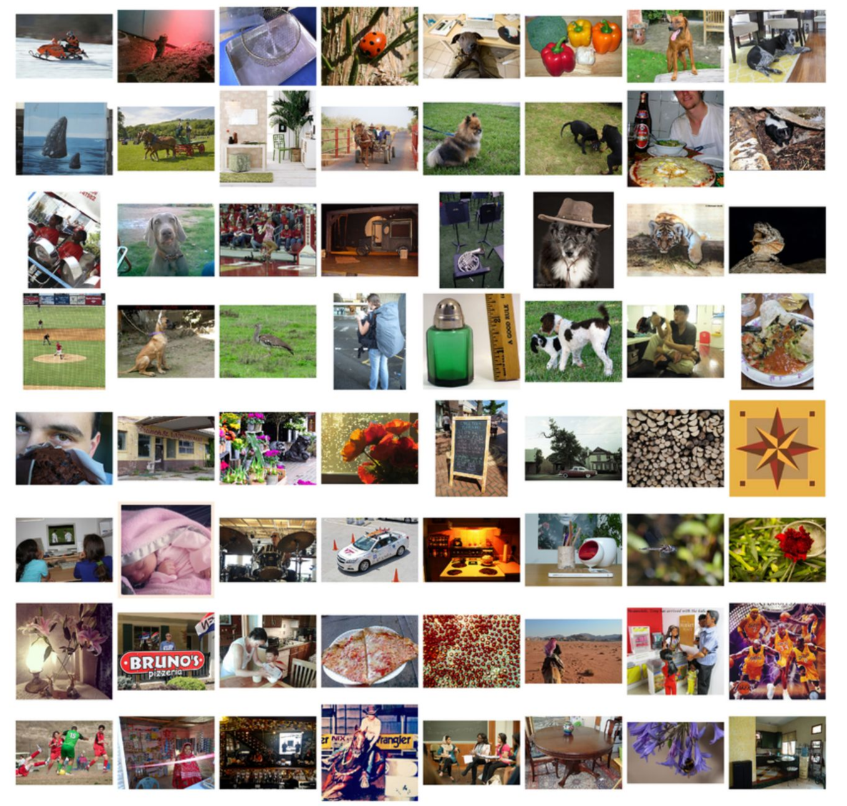
Each pixel is a feature: an MLP would have 32x32x3+1 = 3073 weights per neuron!

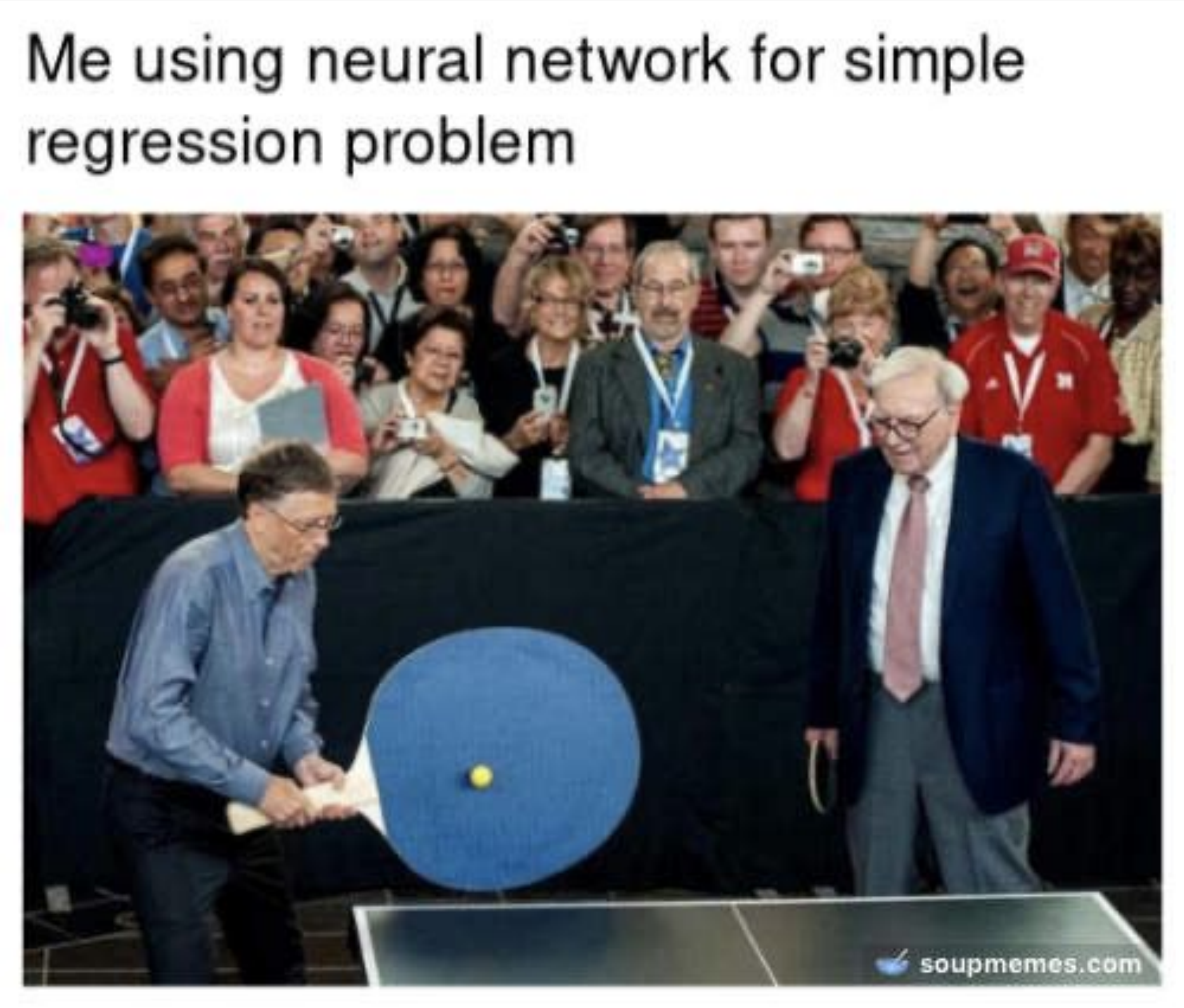
MLP: number of weights for images

Example: ImageNet

Images are usually 224x224x3: an MLP would have 150129 weights per neuron. If the first layer of the MLP is around 128 nodes, which is small, this already becomes very heavy to calculate.

Model complexity is extremely high: overfitting.




Me using neural network for simple regression problem
soupmemes.com

Model Selection and Dimensionality Reduction

Recall from 109A to reduce the number of predictors we can:

- PCA
- Stepwise Variable Selection
- Regularization, in particular L1 will produce sparsity
- Drop predictors that are highly correlated
- **Summarize input (image) with high level features => feature extraction or representation learning**

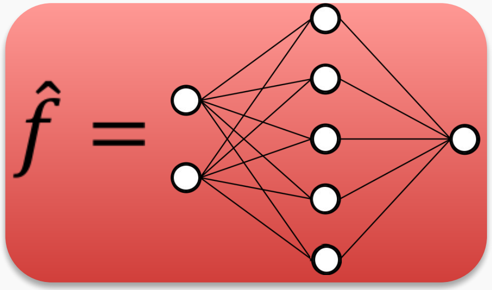
Feature extraction

x

Features:
1. Bald
2. Grey hair
3. Oval shape head
4. Glasses
5. Awesome

Features:
1. Bald
2. Grey hair
3. Oval shape head
4. No Glasses
5. Awesome

$x' \rightarrow$



$\rightarrow \hat{y}$

Model Selection and Dimensionality Reduction

Features:
1. Bald
2. Grey hair
3. Oval shape head
4. No Glasses
5. Awesome ?

Features:
1. Not Bald
2. Dark hair
3. Oval shape head
4. Glasses
5. Extremely Awesome

Image analysis

Imagine that we want to recognize swans in an image:

Cases can be a bit more complex…

Round, elongated head with orange or black beak

Long white neck, square shape

Oval-shaped white body with or without large white symmetric blobs (wings)

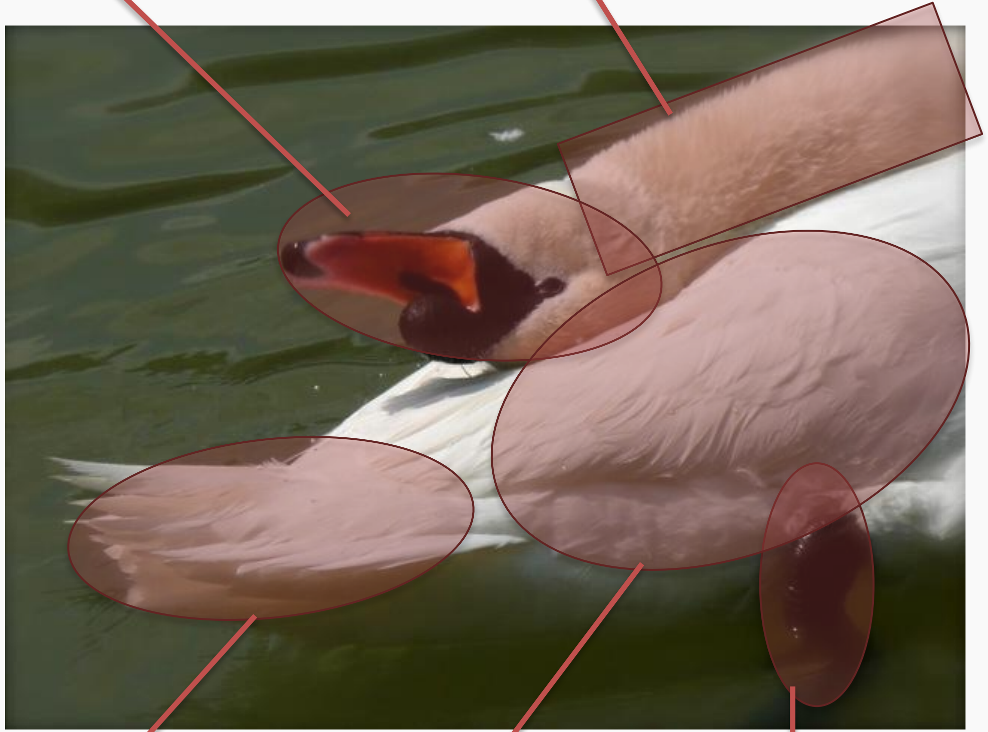
Now what?

Round, elongated head with orange or black beak, can be turned backwards

Long white neck, can bend around, not necessarily straight

Small black circles, can be facing the camera, sometimes can see both

Black triangular shaped form, on the head, can have different sizes



White tail, generally far from the head, looks feathery

White, oval shaped body, with or without wings visible

Black feet, under body, can have different shapes

White elongated piece, can be squared or more triangular, can be obstructed sometimes

Luckily, the color is consistent…

We need to be able to deal with these cases

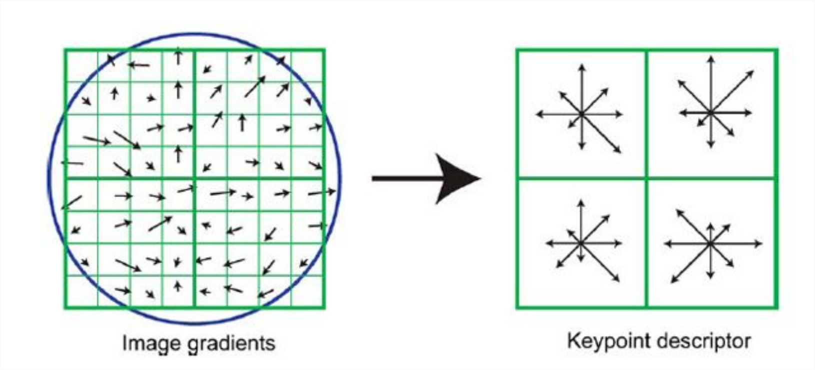
And these

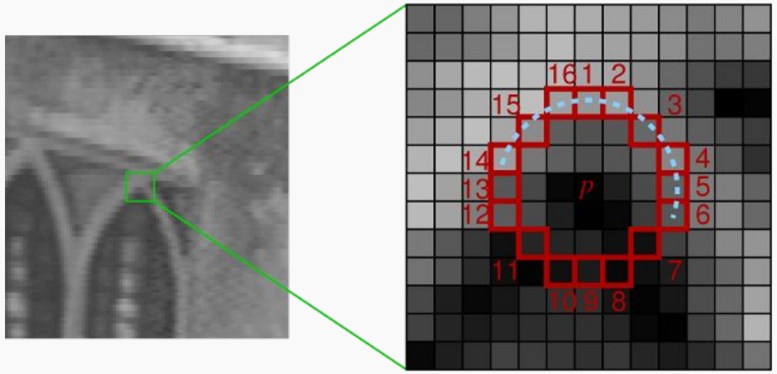
Man in swan tent photographing swans

Image features

- We've been basically talking about detecting features in images, in a very naïve way.
- Researchers built multiple computer vision techniques to deal with these issues: SIFT, FAST, SURF, BRIEF, etc.
- However, similar problems arose: the detectors where either too general or too over-engineered. Humans were designing these feature detectors, and that made them either too simple or hard to generalize.

SIFT feature descriptor





FAST corner detection algorithm

Image features (cont)

- What if we learned the features to detect?
- We need a system that can do Representation Learning (or Feature Learning).

Representation Learning: technique that allows a system to automatically find relevant features for a given task. Replaces manual feature engineering.

Multiple techniques for this:

- Unsupervised (K-means, PCA, ...).
- Supervised (Sup. Dictionary learning, Neural Networks!)

Moreover

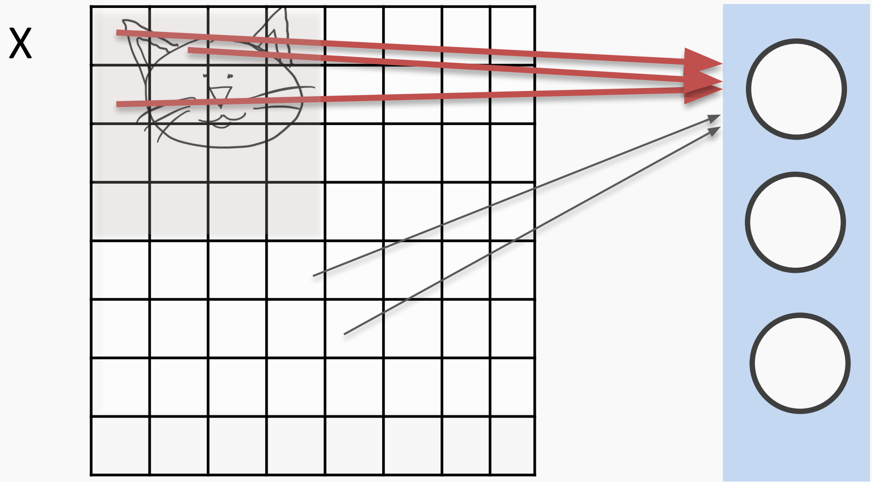
Nearby pixels are more strongly related than distant ones.

Objects are built up out of smaller parts.

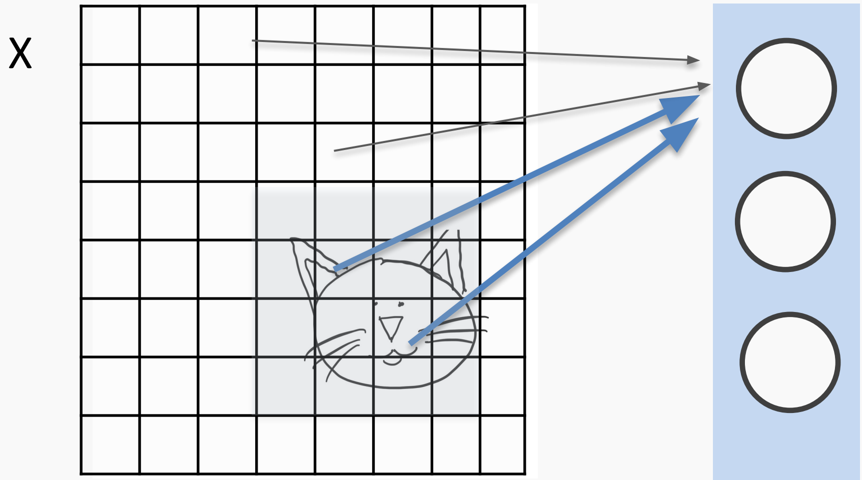
Outline

1. Motivation
2. **CNN basic ideas**
3. Building a CNN

Each neuron from first layer has one weight per pixel. Recall, the importance of the predictors (here pixels) is given by the value of the coefficient (here W).



In this case, the red weights will be modified to better recognize cat.

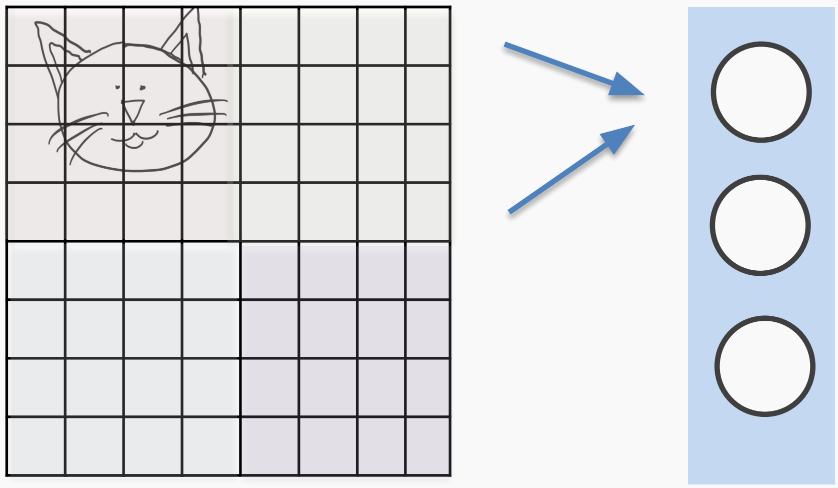


In this case, the blue weights will be modified.

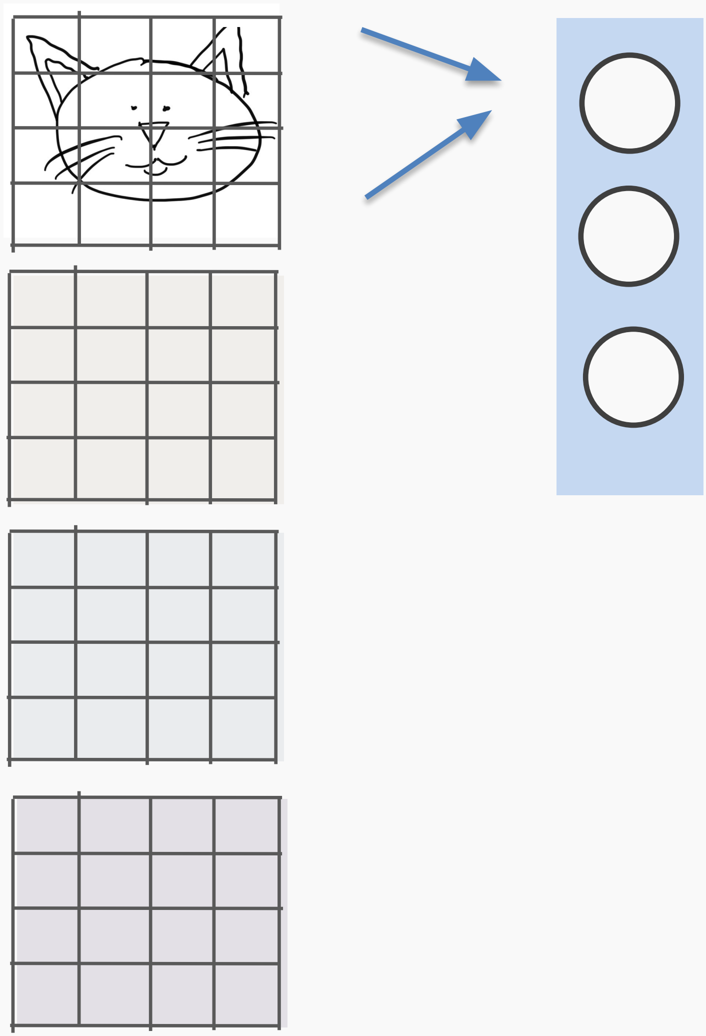
We are learning **redundant** features. Approach is not robust, as cats could appear in yet another position.

Solution: Cut the image to smaller pieces.

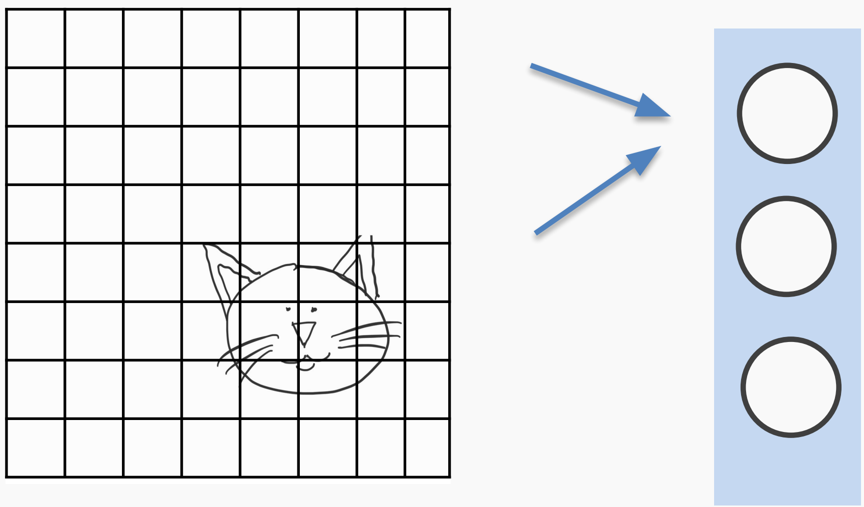
$X: 8\times8$

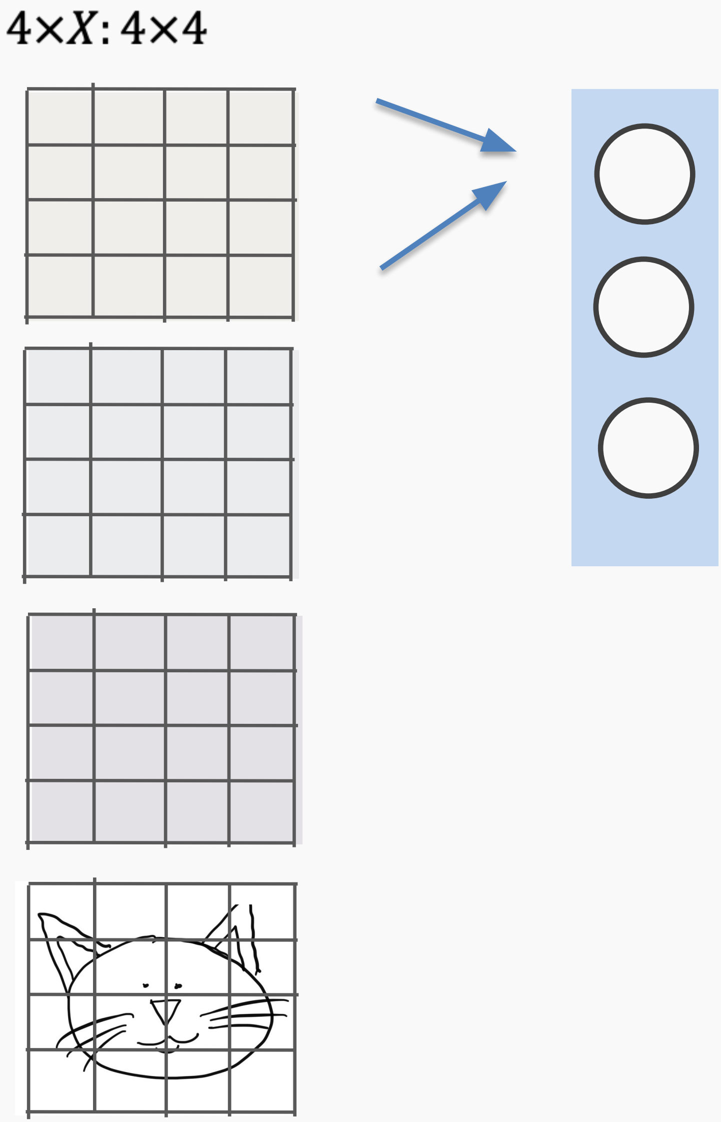


$4\times X: 4\times4$



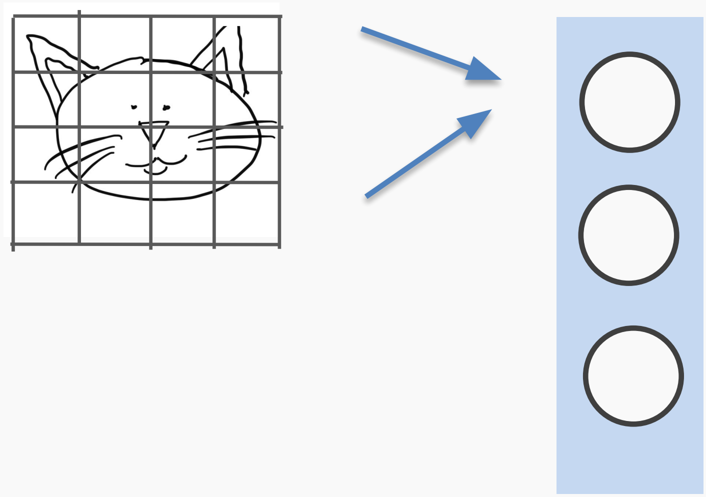
Do the same for all images



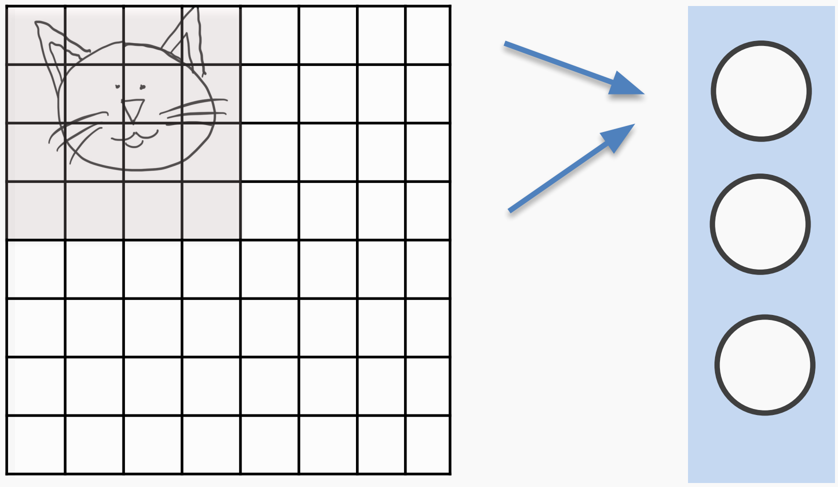


But what if cat is not the box?

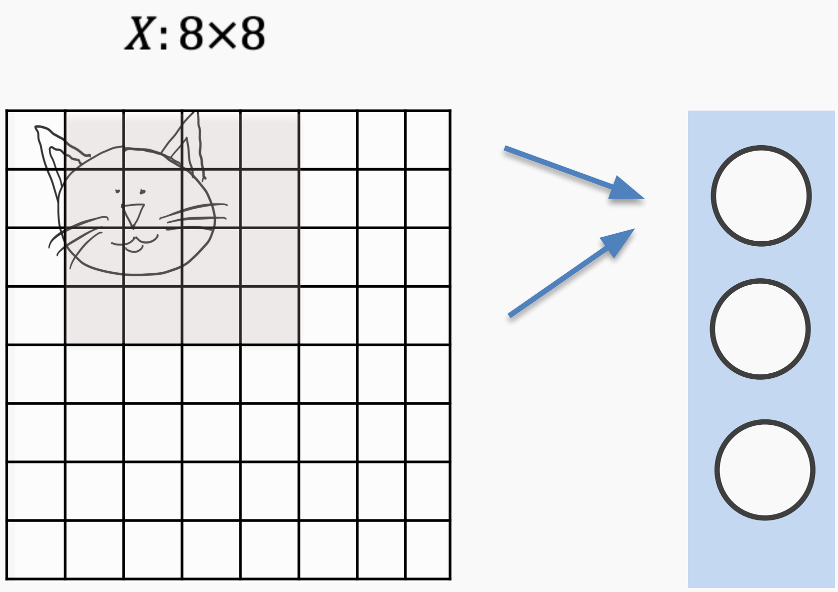
$4 \times X: 4 \times 4$

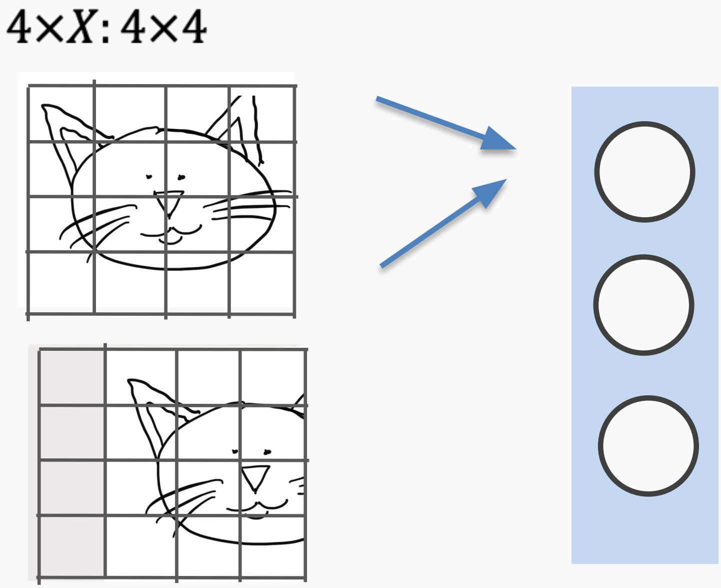


$X: 8 \times 8$



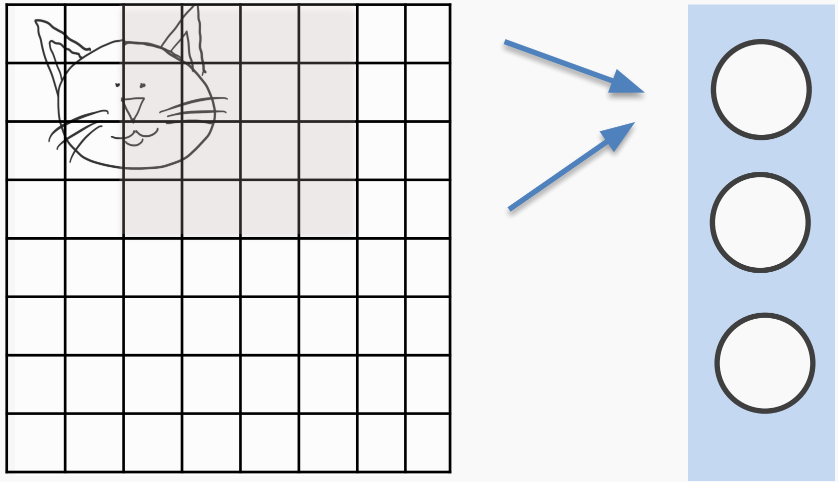
But what if cat is not the box?



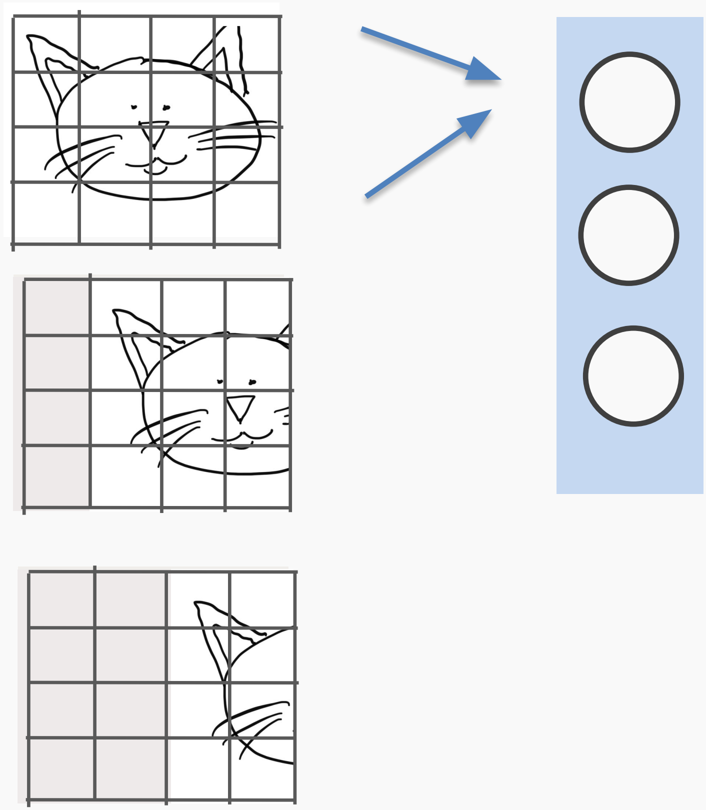


But what if cat is not the box?

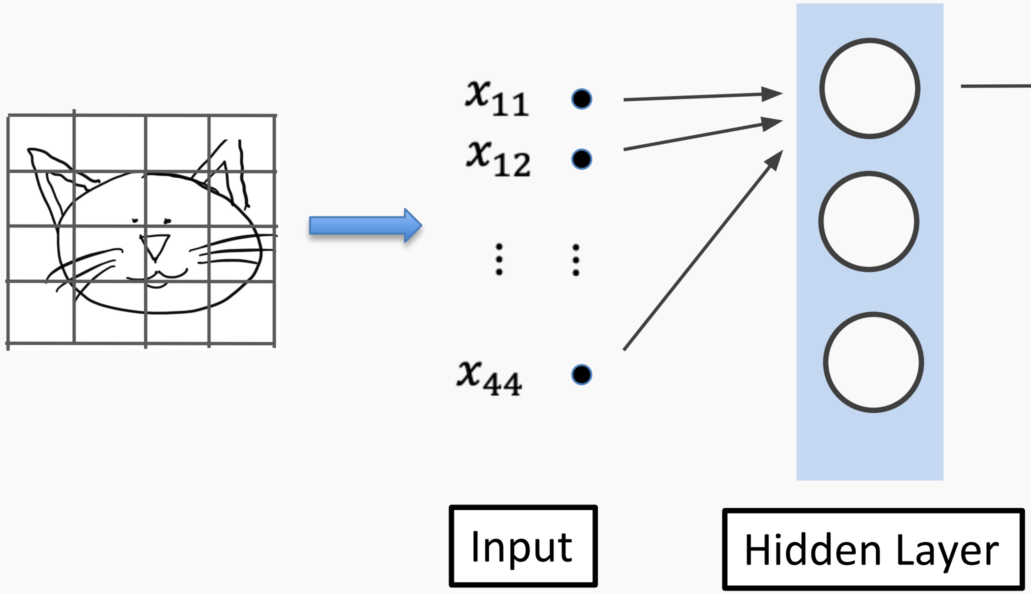
X: 8×8



$4 \times X$: 4×4



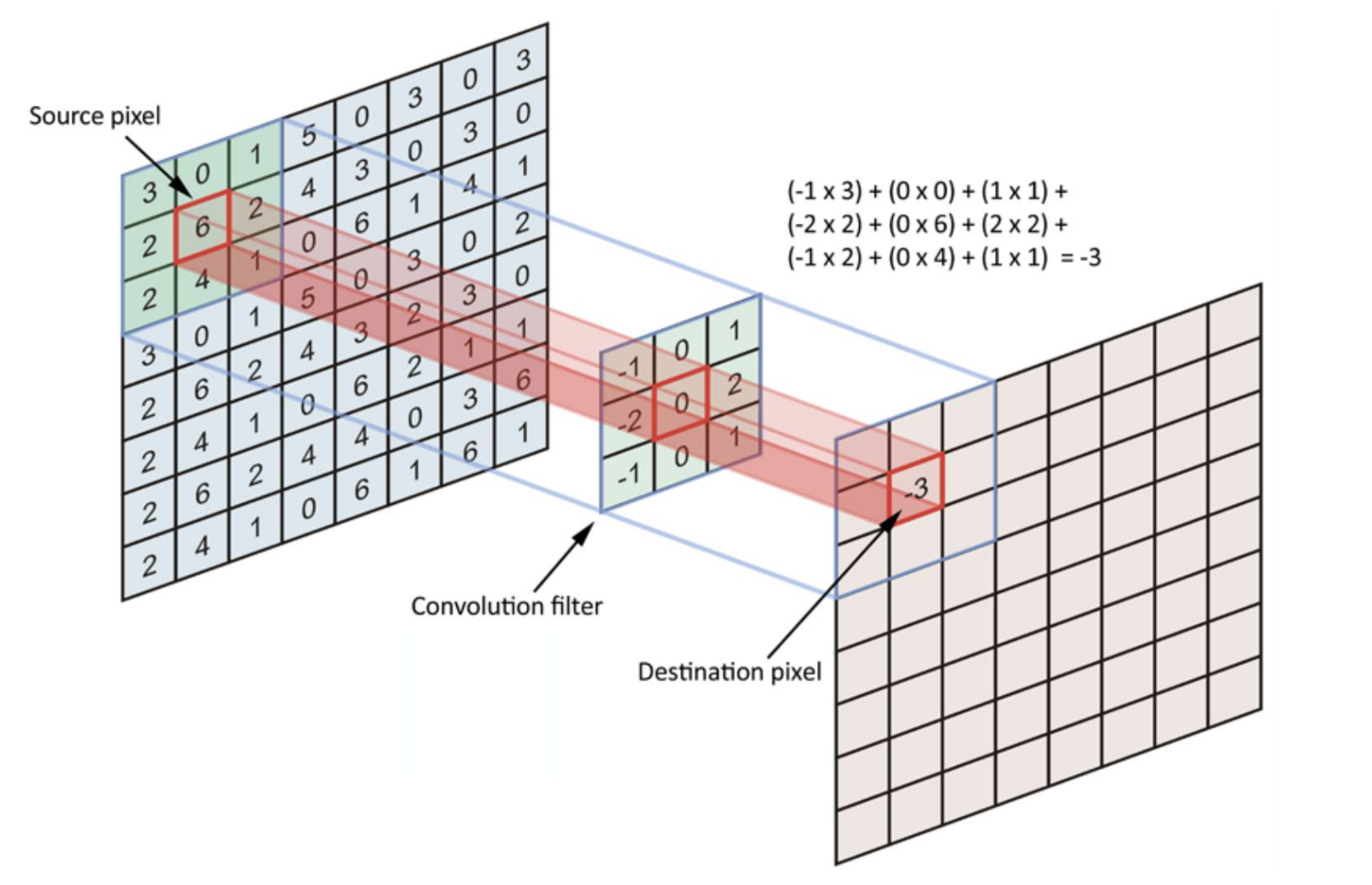
Convolution



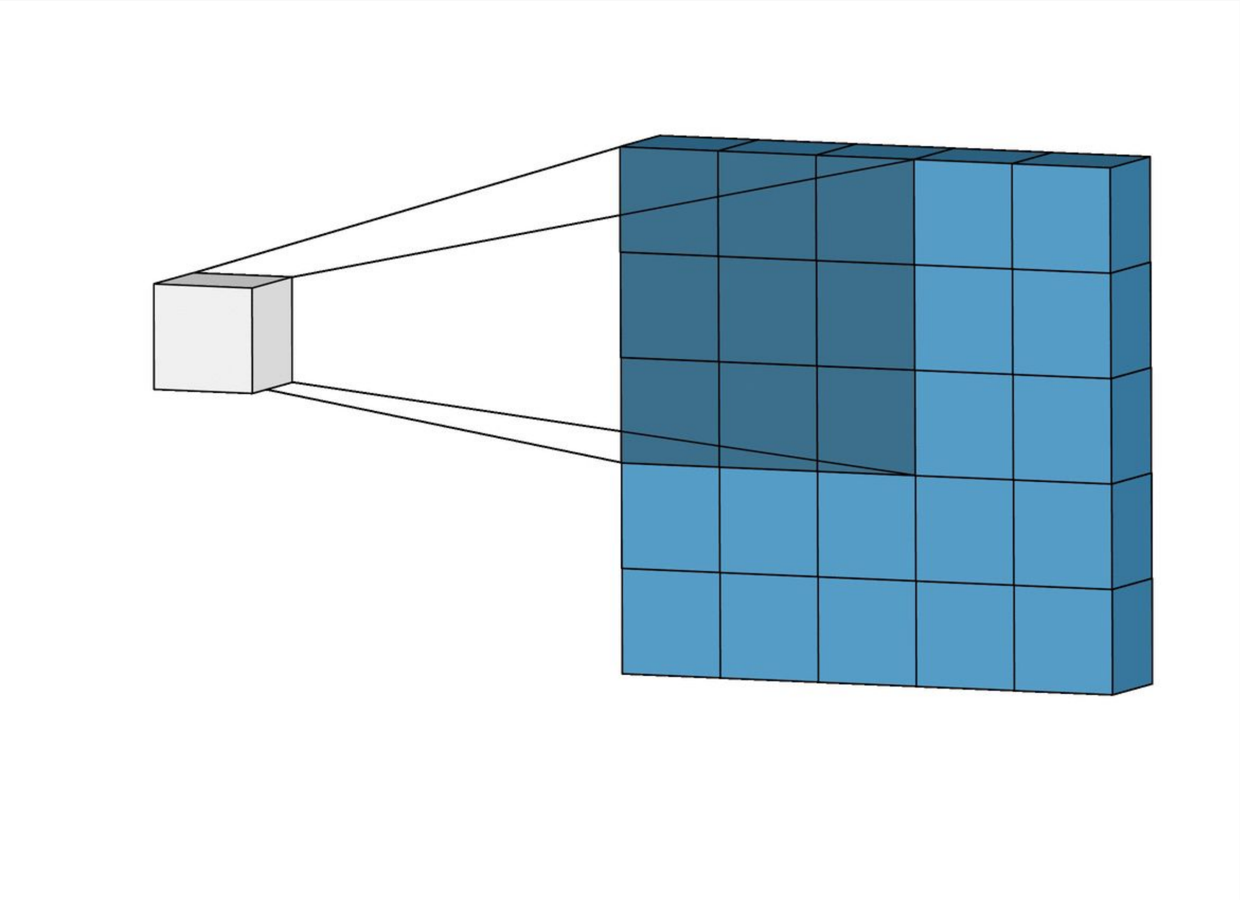
$$h_1 = f(W_{11}x_{11} + W_{12}x_{12} + \cdots + W_{44}x_{44})$$

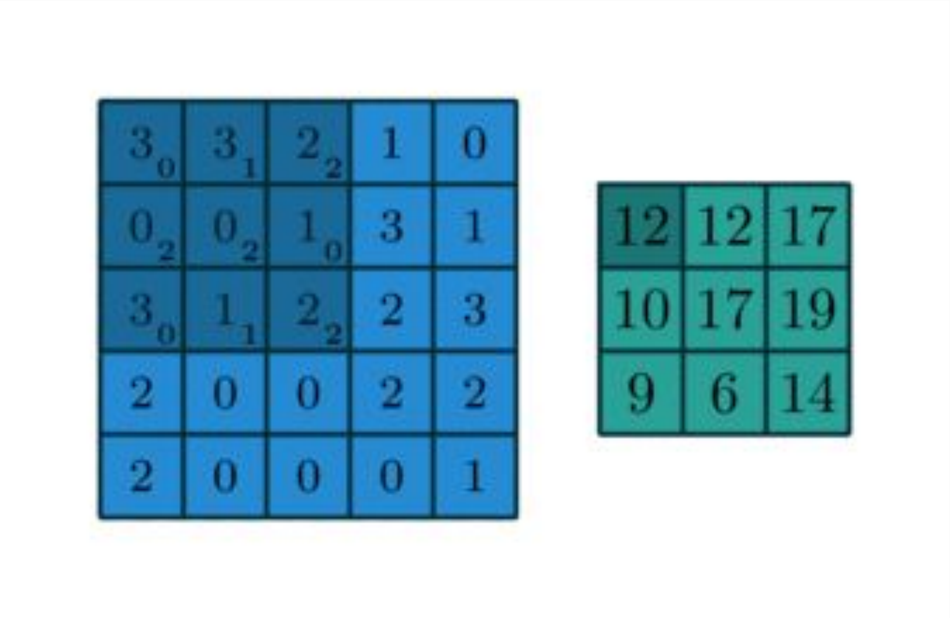
$$\sum \begin{array}{|c|c|c|c|} \hline W_{11} & W_{12} & W_{13} & W_{14} \\ \hline & & & \\ \hline & & & \\ \hline & & & W_{44} \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline x_{11} & x_{12} & x_{13} & x_{14} \\ \hline & & & \\ \hline & & & \\ \hline & & & x_{44} \\ \hline \end{array}$$

“Convolution” Operation

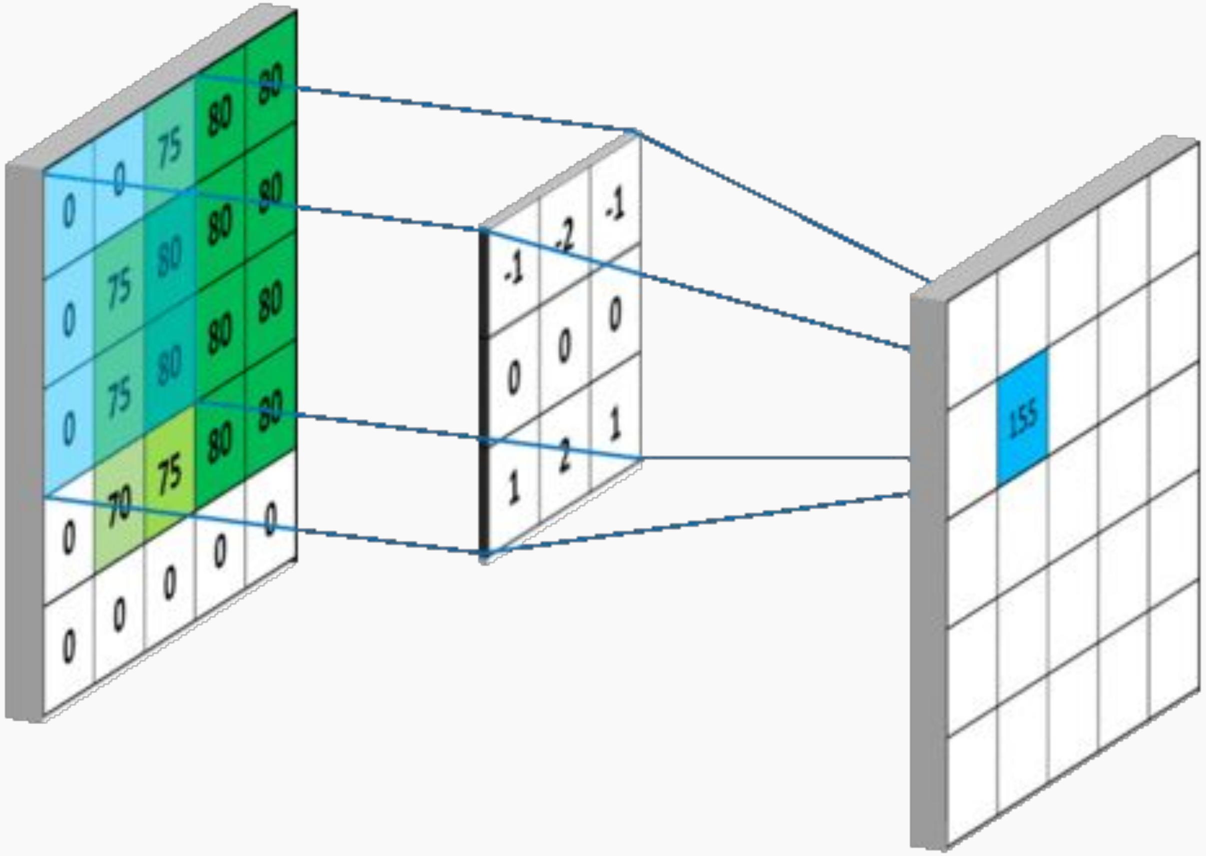


Convolutions – step by step





Convolutions – another example



Convolution and cross-correlation

- A **convolution** of f and g ($f * g$) is defined as the integral of the product, having one of the functions inverted and shifted:

$$(f * g)(t) = \int_a f(a)g(t - a)da$$

Function is inverted and shifted left by t

- Discrete convolution:

∞

“Convolution” Operation in action

Kernel

Edge detection

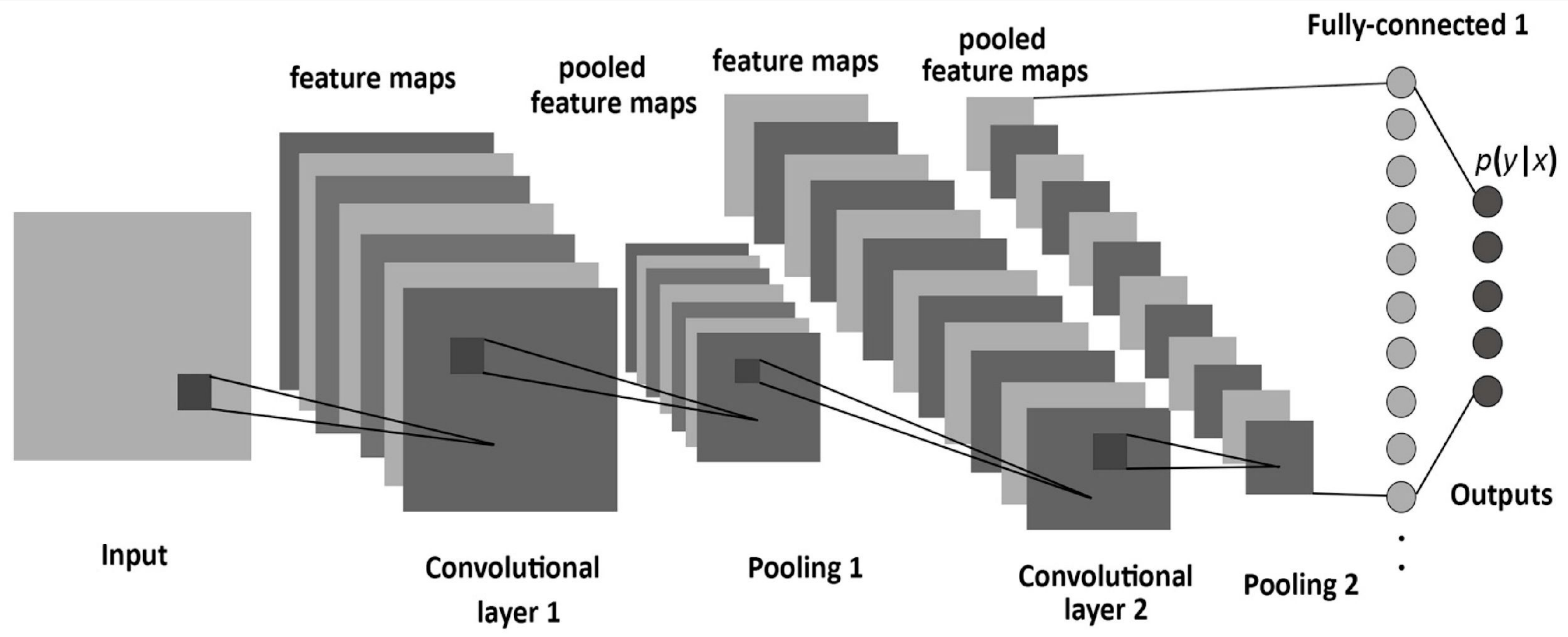
$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

Sharpen

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

wikipedia.org

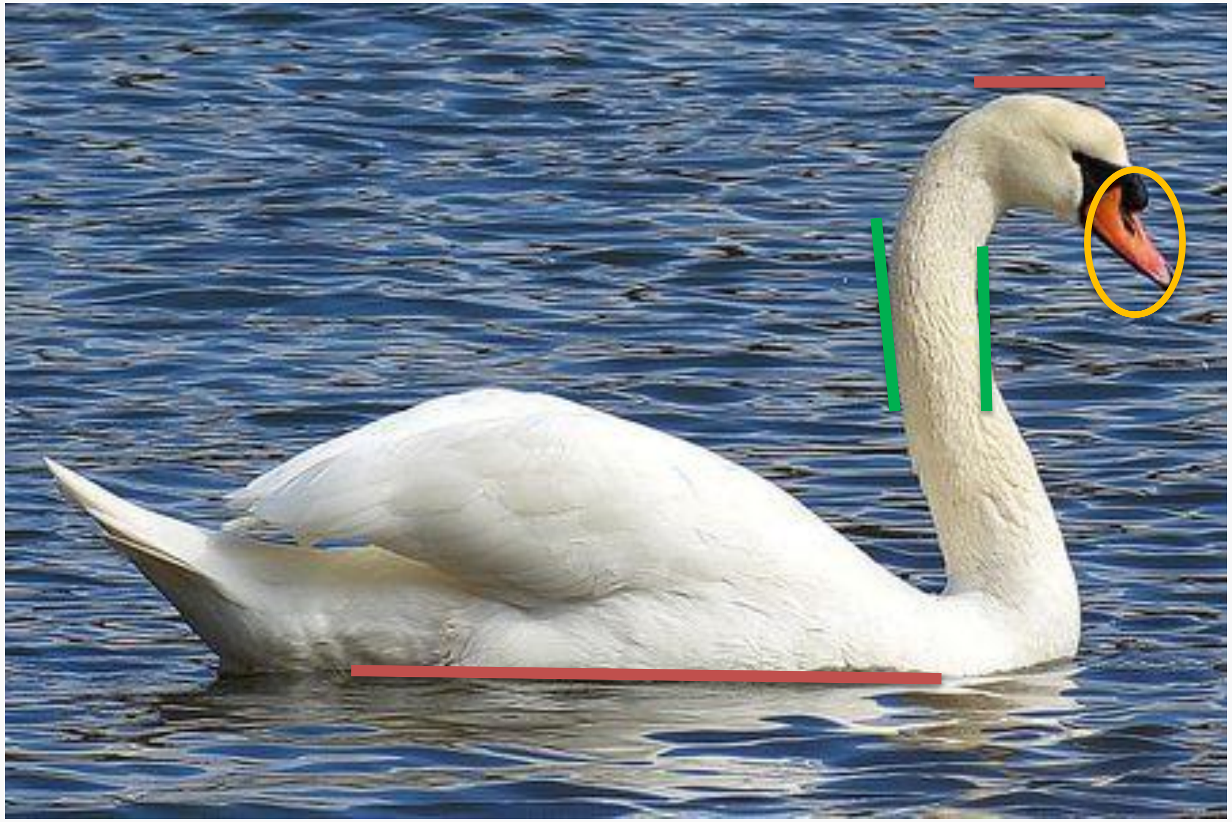
A Convolutional Network



+ ReLU

+ ReLU

Why more than one feature map?



Feature 1: Horizontal Lines

Feature 2: Vertical Lines

Feature 3: Orange bulb

Why more than one layer?

Layer 2, Feature 1: Combine horizontal and vertical lines from Layer 1 produce diagonal lines.

Layer 3, Feature 1: Combine diagonal to identify shapes

So far

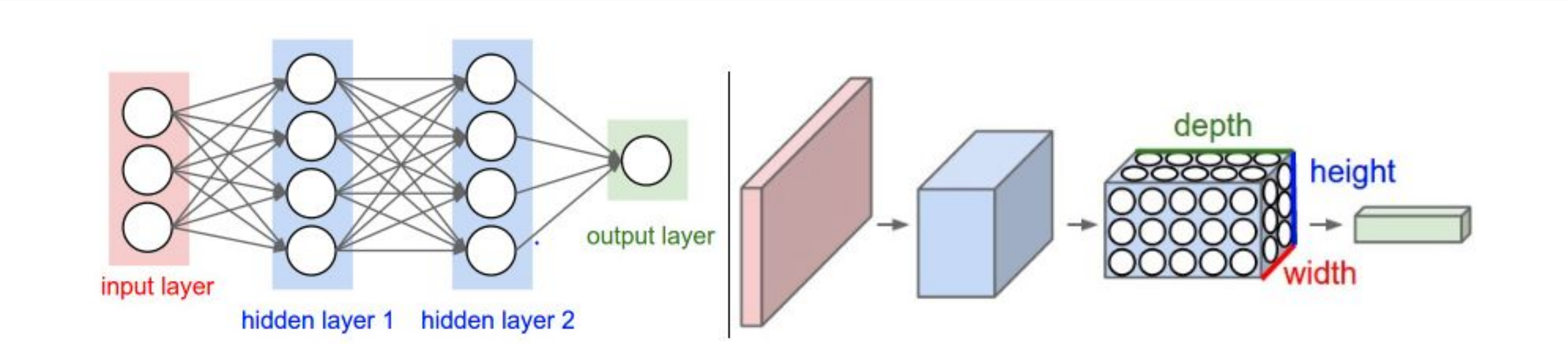
We know that MLPs:

- Do not scale well for images
- Ignore the information brought by pixel position and correlation with neighbors
- Cannot handle translations

The general idea of CNNs is to intelligently adapt to properties of images:

- Pixel position and neighborhood have semantic meanings.
- Elements of interest can appear anywhere in the image.

Basics of CNNs



MLP CNN

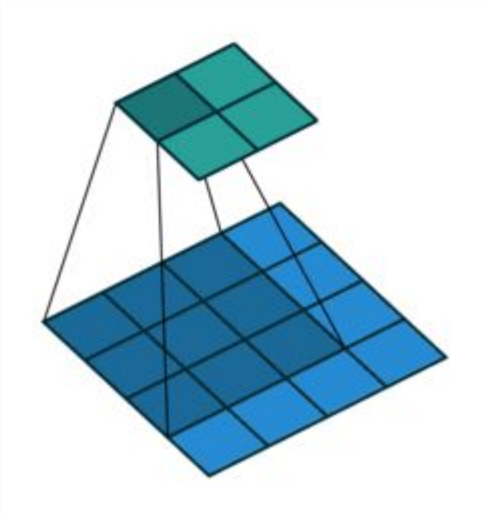
CNNs are also composed of layers, but those layers are not fully connected: they have filters, sets of cube-shaped weights that are applied throughout the image. Each 2D slice of the filters are called kernels.

These filters introduce translation invariance and parameter sharing.

How are they applied? Convolutions!

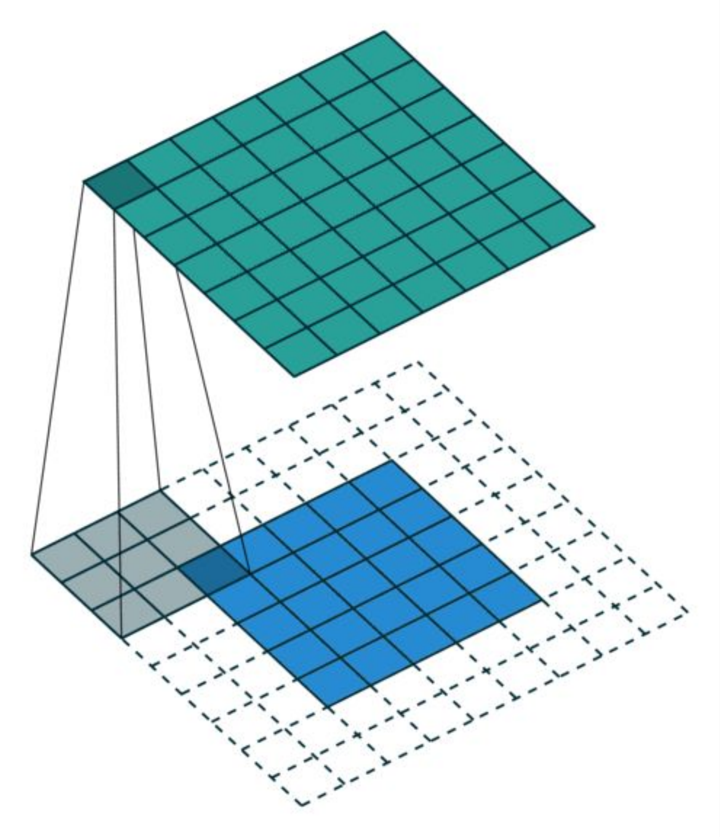
Convolutions – what happens at the edges?

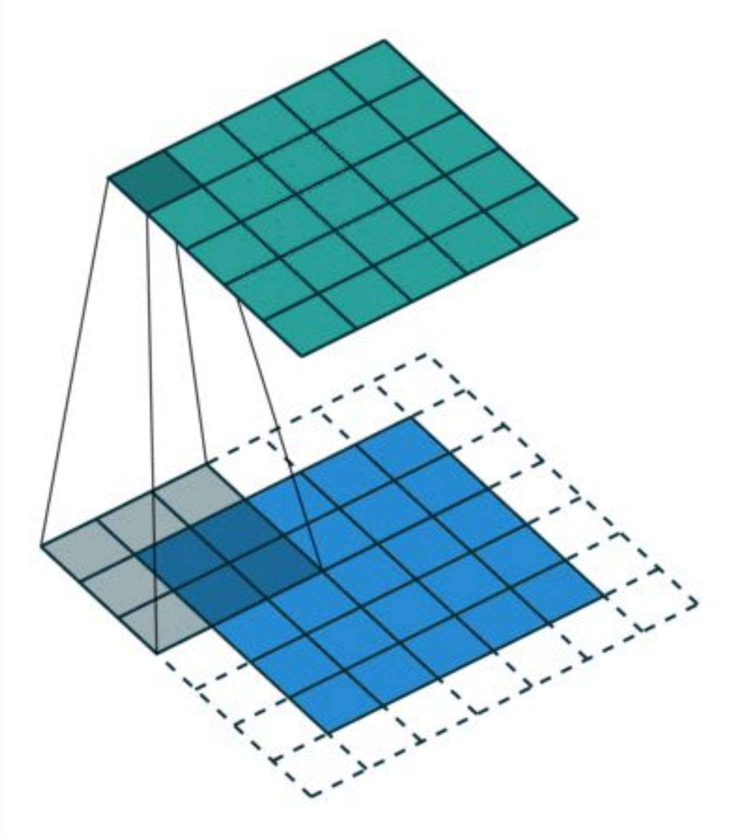
If we apply convolutions on a normal image, the result will be down-sampled by an amount depending on the size of the filter.



We can avoid this by padding the edges in different ways.

Padding





Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.

Same padding. Ensures that the output has the same size as the input.

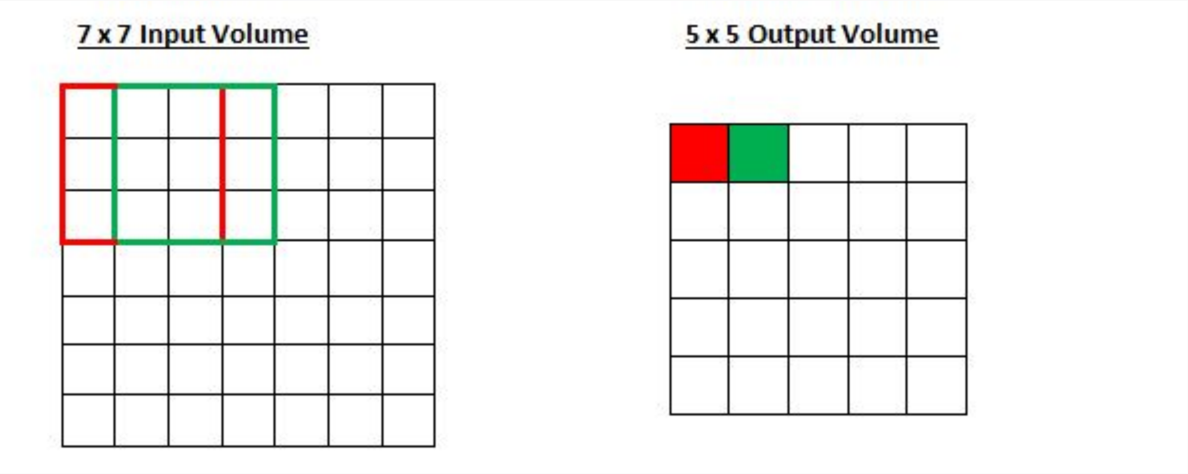
Stride

Stride controls how the filter convolves around the input volume.

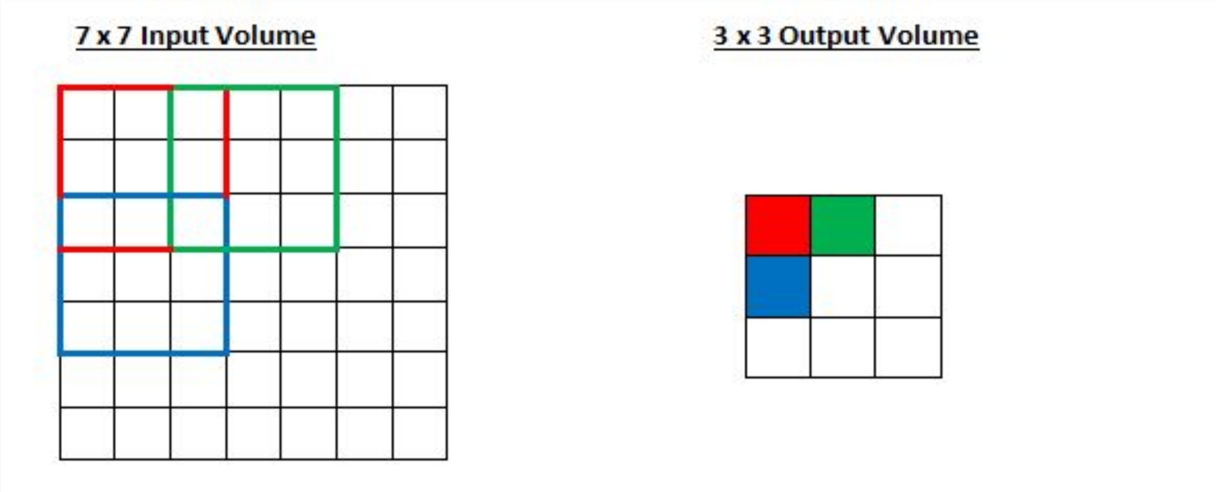
The formula for calculating the output size is:

$$O = \frac{W - K + 2P}{S} + 1$$

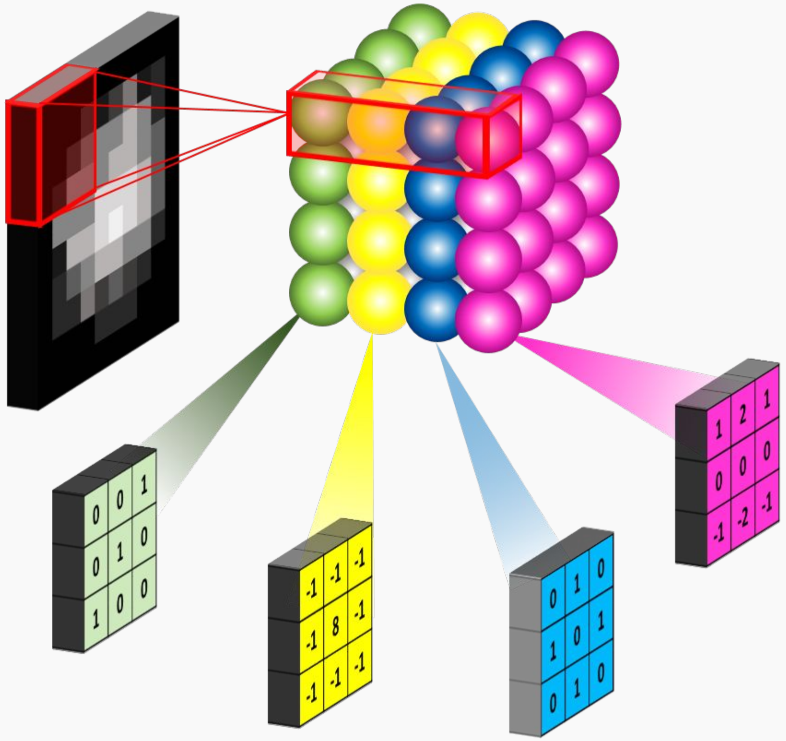
Stride = 1



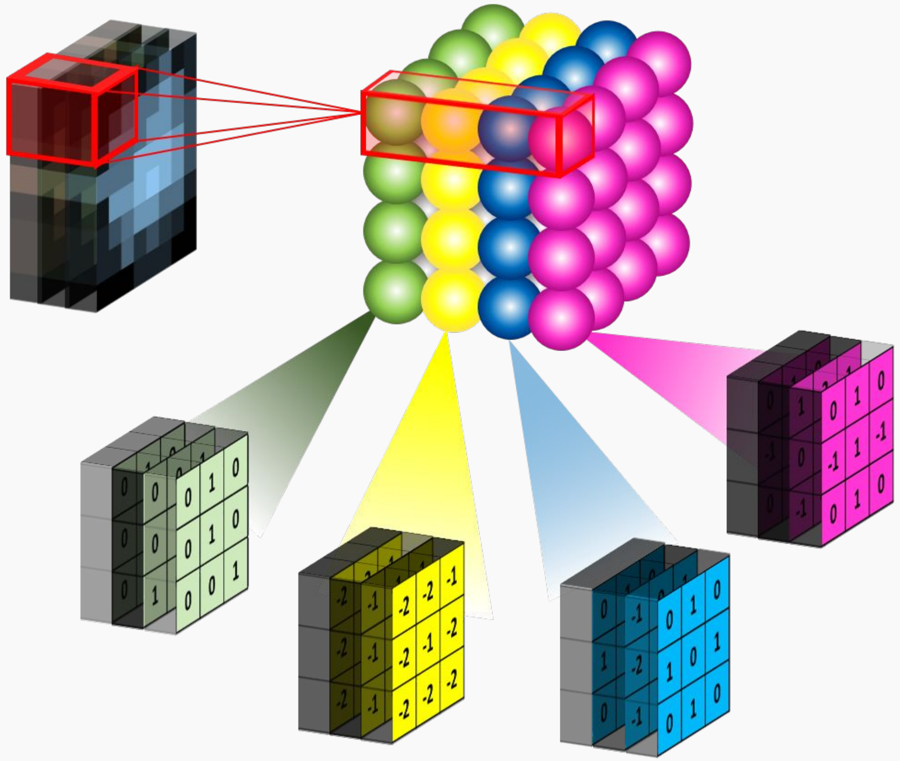
Stride = 2



Convolutional layers



Convolutional layer with four 3x3 filters on a black and white image (just one channel)



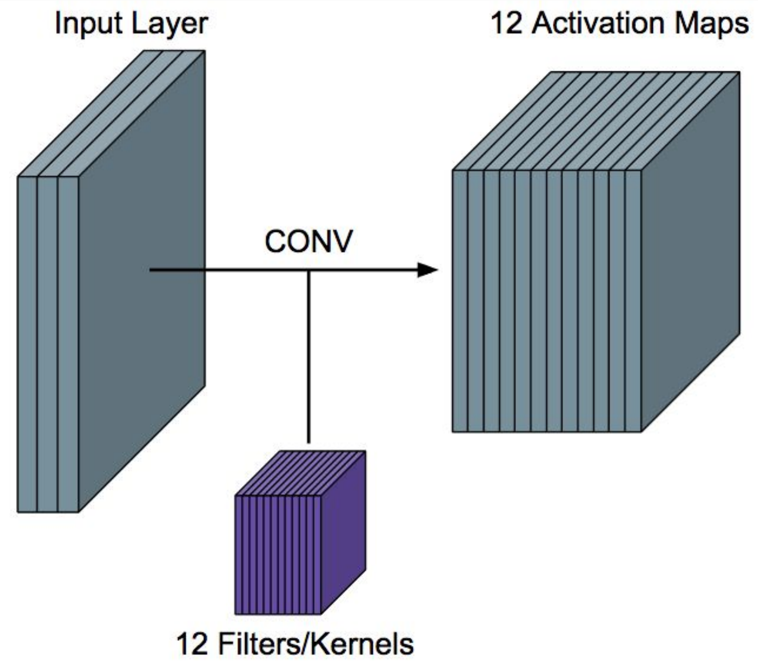
Convolutional layer with four 3x3 filters on an RGB image. As you can see, the filters are now cubes, and they are applied on the full depth of the image..

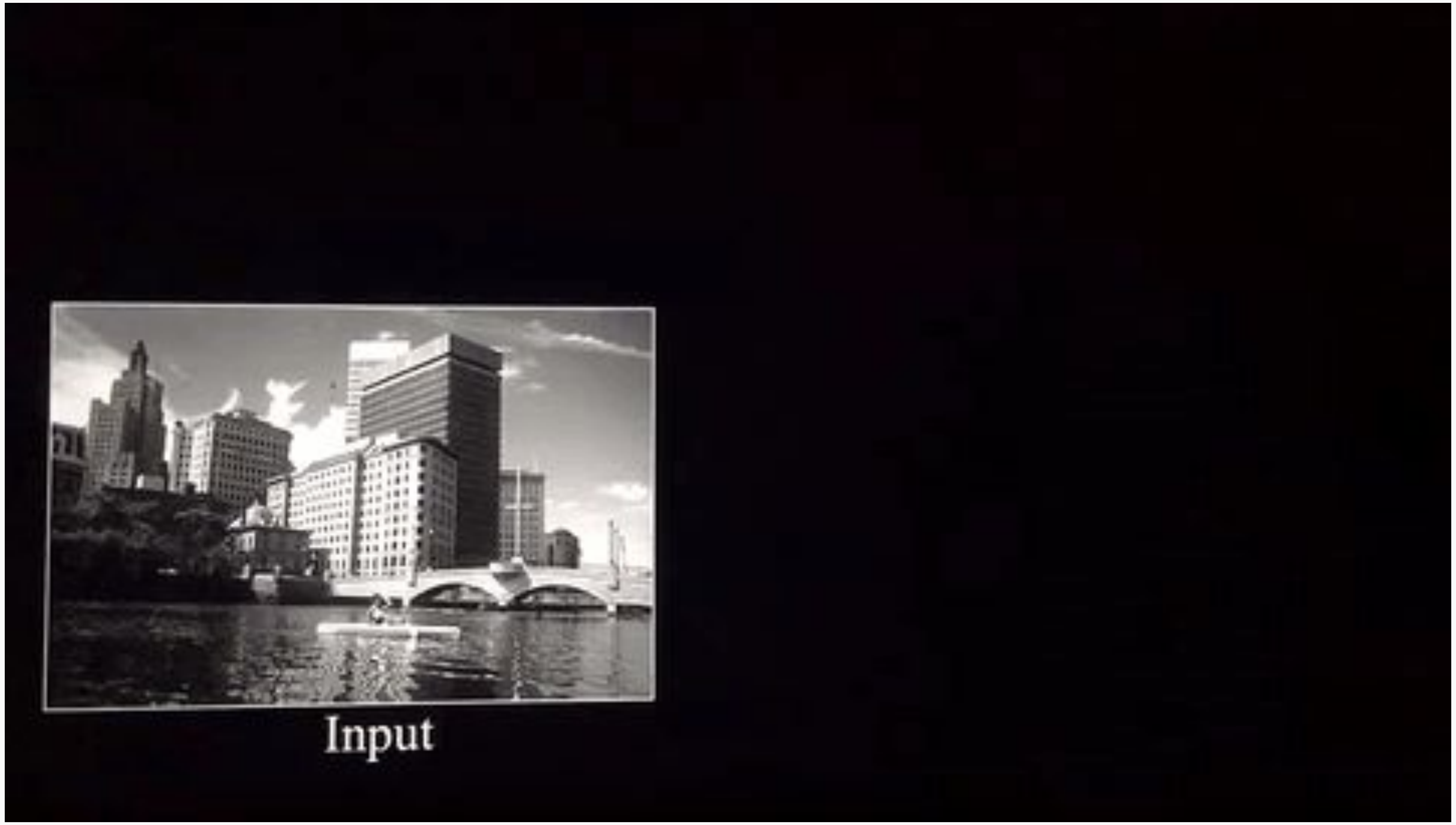
Convolutional layers (cont)

- To be clear: each filter is convolved with the entirety of the 3D input cube, but generates a 2D feature map.

- Because we have multiple filters, we end up with a 3D output: one 2D feature map per filter.

- The feature map dimension can change drastically from one conv layer to the next: we can enter a layer with a 32x32x16 input and exit with a 32x32x128 output if that layer has 128 filters.




Input

Learning CNN

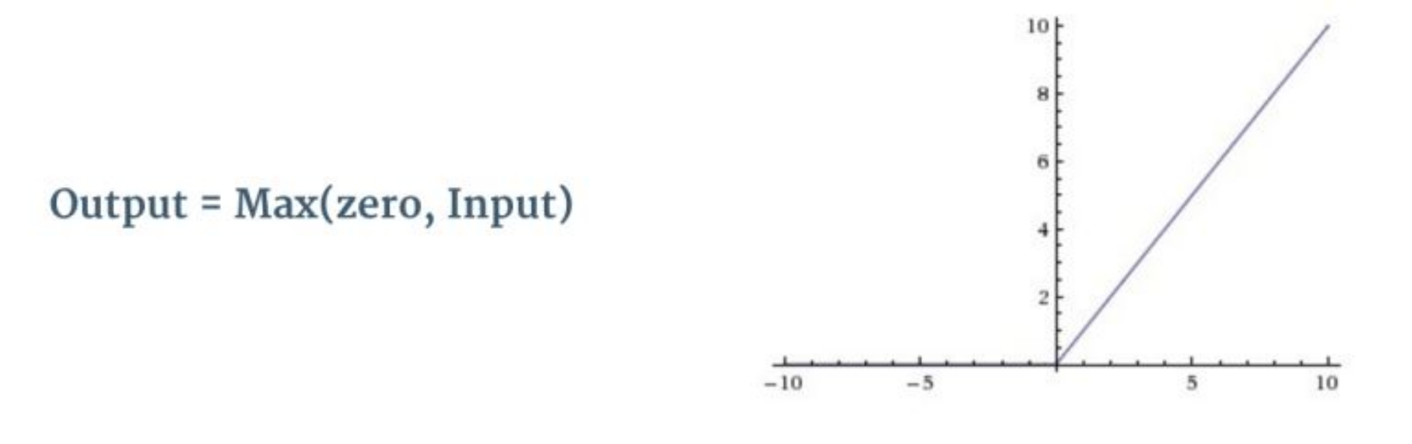
In a convolutional layer, we are basically applying multiple filters at over the image to extract different features.

But most importantly, we are learning those filters!

One thing we're missing: non-linearity.

ReLU

The most successful non-linearity for CNNs is the Rectified Non-Linear unit (ReLU):

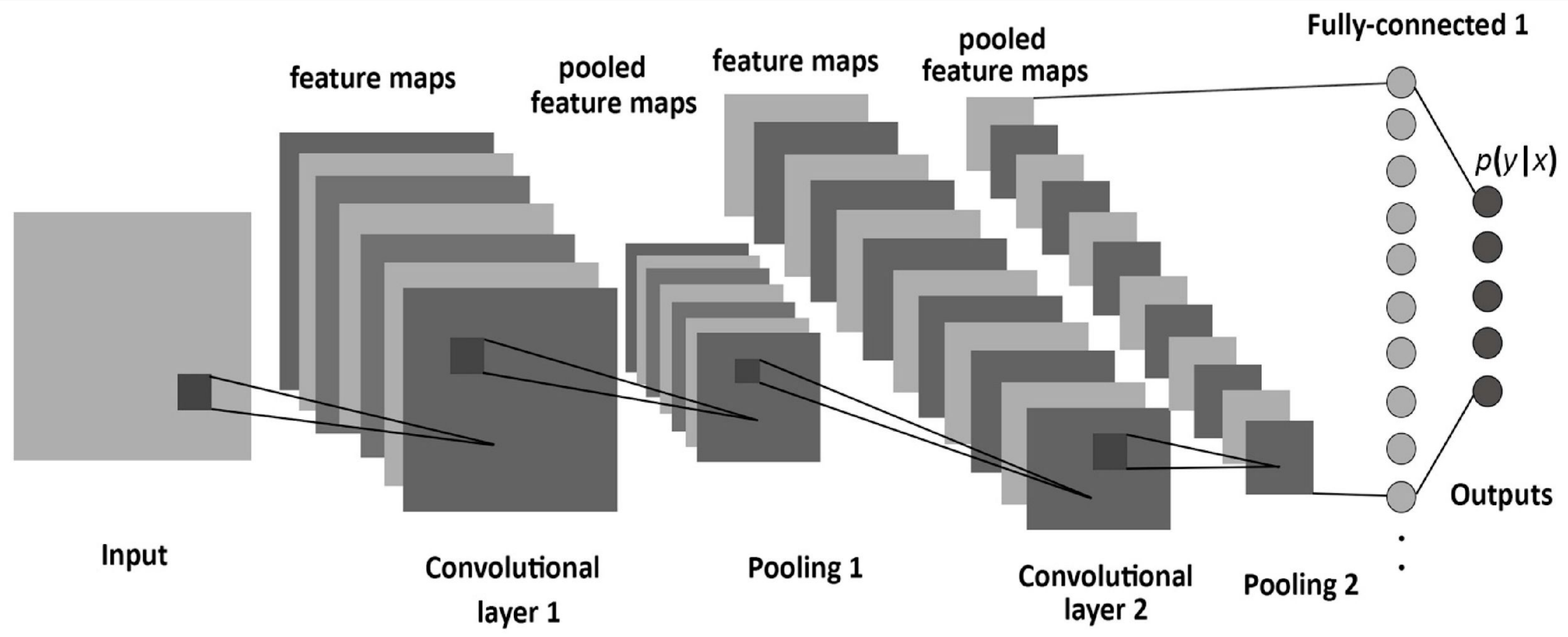


Combats the vanishing gradient problem occurring in sigmoids, is easier to compute, generates sparsity (not always beneficial)

Convolutional layers so far

- A convolutional layer convolves each of its filters with the input.
- Input: a 3D tensor, where the dimensions are Width, Height and Channels (or Feature Maps)
- Output: a 3D tensor, with dimensions Width, Height and Feature Maps (one for each filter)
- Applies non-linear activation function (usually ReLU) over each value of the output.
- Multiple parameters to define: number of filters, size of filters, stride, padding, activation function to use, regularization.

A Convolutional Network



+ ReLU

+ ReLU

Pooling

Invariant to small, "local transitions"

- Face detection: enough to check the presence of eyes, not their precise location

Reduces input size to final fully connected layers

No learnable parameters

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window and stride 1

7	7	8

Pooling (cont)

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window and stride 2

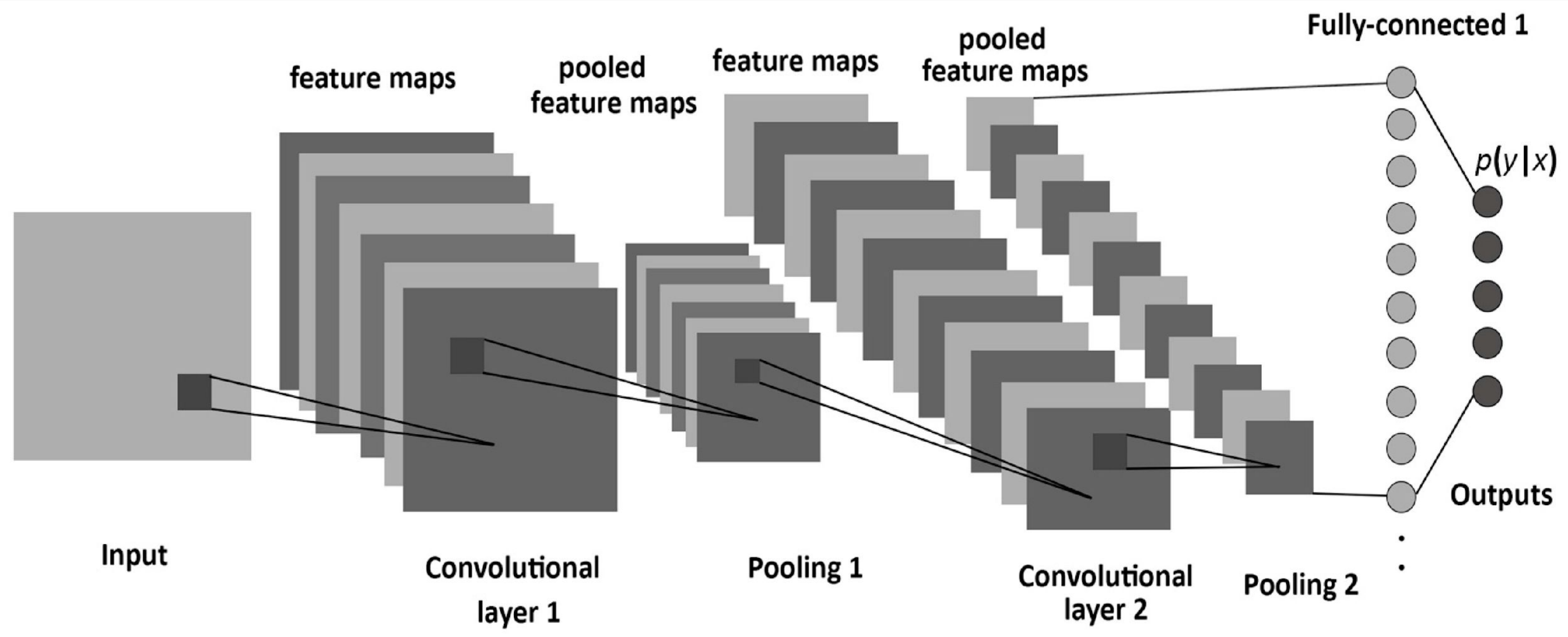
7	8
3	4

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

mean pool with 2x2 window and stride 2

7	11
3.5	4

A Convolutional Network



+ ReLU

+ ReLU

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

- Convolutional Layers
- Pooling Layers
- Fully connected Layers

Building a CNN

A c
ty

Convolutional Layers

Action

- Apply filters to extract features
- Filters are composed of small kernels, learned.
- One bias per filter.
- Apply activation function on every value of feature map

Parameters

- Number of kernels
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding
- Regularization type and value

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

- Convolutional Layers
- Pooling Layers
- Fully connected Layers

Building a CNN

A c
ty

Pooling Layers

Action

- Reduce dimensionality
- Extract maximum of average of a region
- Sliding window approach

Parameters

- Stride
- Size of window

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filte, reduced spatial dimensions

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional Layers

Pooling Layers

Fully connected Layers

Building a CNN

A c
typ

Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification

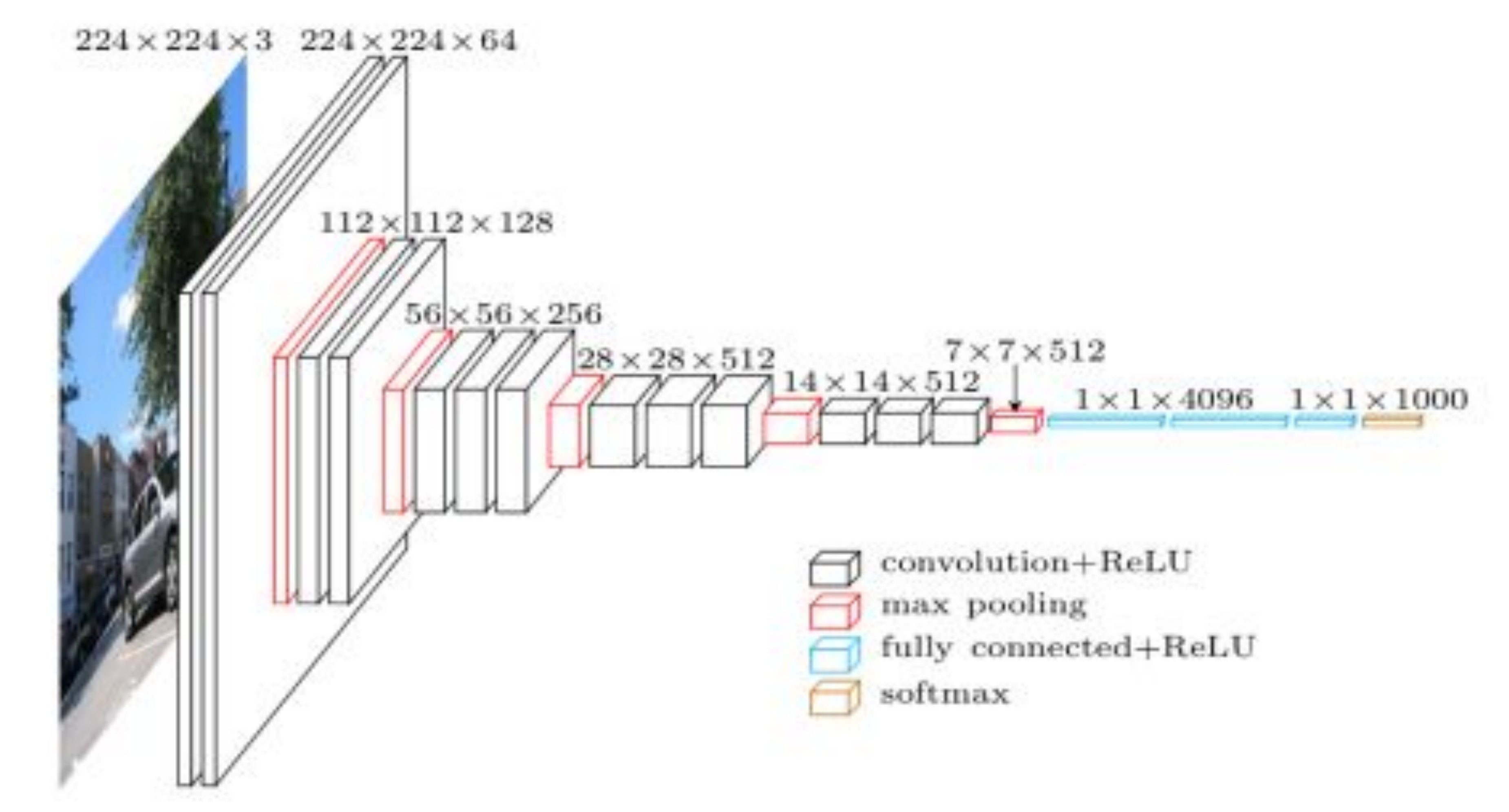
Parameters

- Number of nodes
- Activation function: usually changes depending on role of layer. If aggregating info, use ReLU. If producing final classification, use Softmax.

I/O

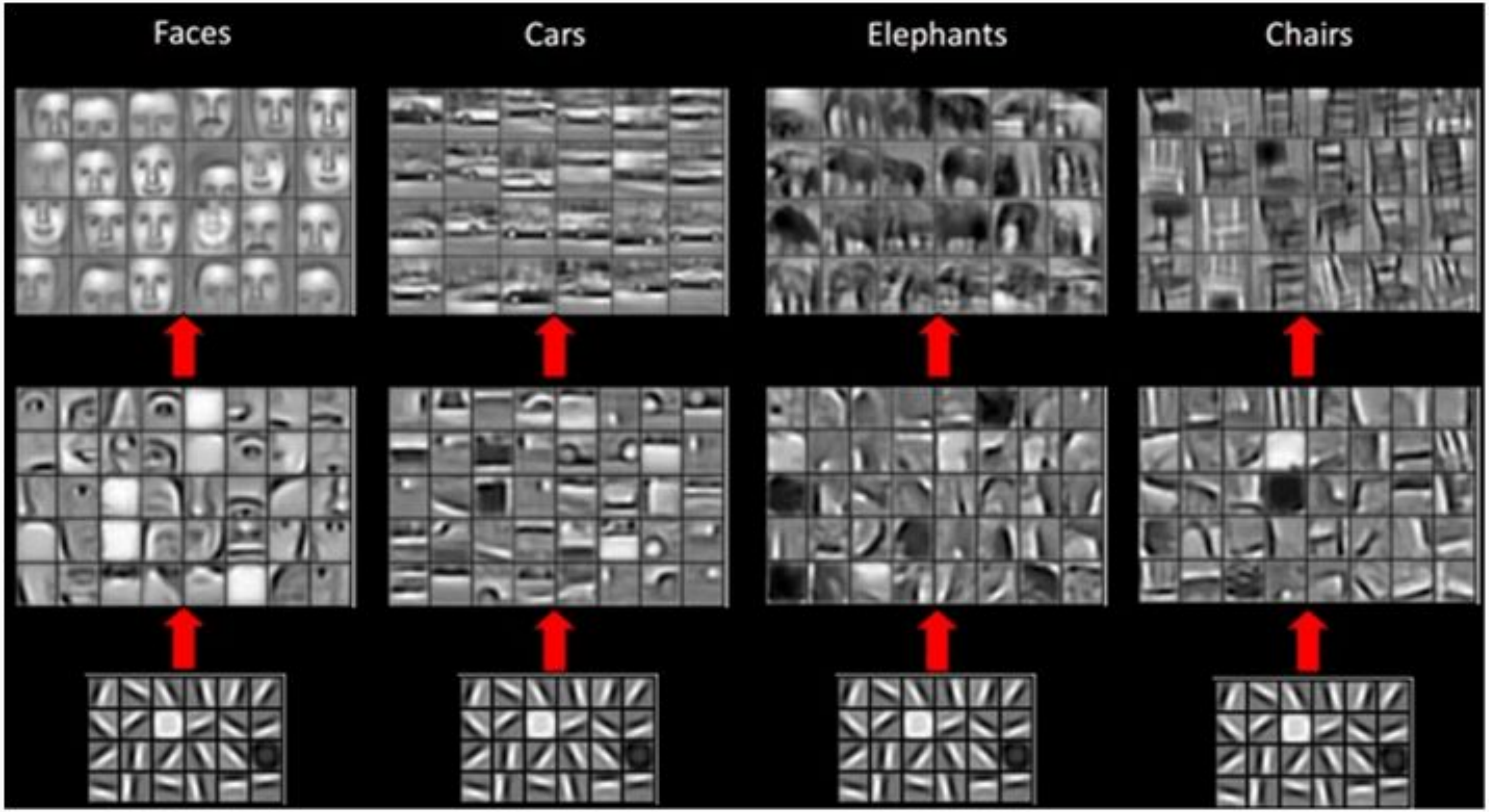
- Input: FLATTENED 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Fully built CNN (VGG)



What do CNN layers learn?

- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters: edges, corners, etc.
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc.
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.


Faces
Cars
Elephants
Chairs

Examples

- I have a convolutional layer with 12 4x4 filters that takes an RGB image as input.
    - What else can we define about this layer?
        - Activation function
        - Stride
        - Padding type
    - How many parameters does the layer have?

$$\underbrace{12}_{\text{Number of filters}} \times \underbrace{4 \times 4}_{\text{Size of Filters}} \times \underbrace{3}_{\text{Number of channels of prev layer}} + \underbrace{12}_{\text{Biases (one per filter)}} = 588$$

Examples

- Let C be a CNN with the following disposition:
  - Input: 32x32x3 images
  - Conv1: 8 3x3 filters, stride 1, padding=same
  - Conv2: 16 5x5 filters, stride 2, padding=same
  - Flatten layer
  - Dense1: 512 nodes
  - Dense2: 4 nodes
- How many parameters does this network have?

$$\underbrace{(8 \times 3 \times 3 \times 3 + 8)}_{\text{Conv1}} + \underbrace{(16 \times 5 \times 5 \times 8 + 16)}_{\text{Conv2}} + \underbrace{(16 \times 16 \times 16 \times 512 + 512)}_{\text{Dense1}} + \underbrace{(512 \times 4 + 4)}_{\text{Dense2}}$$

3D visualization of networks in action

http://scs.ryerson.ca/~aharley/vis/conv/

https://www.youtube.com/watch?v=3JQ3hYko51Y