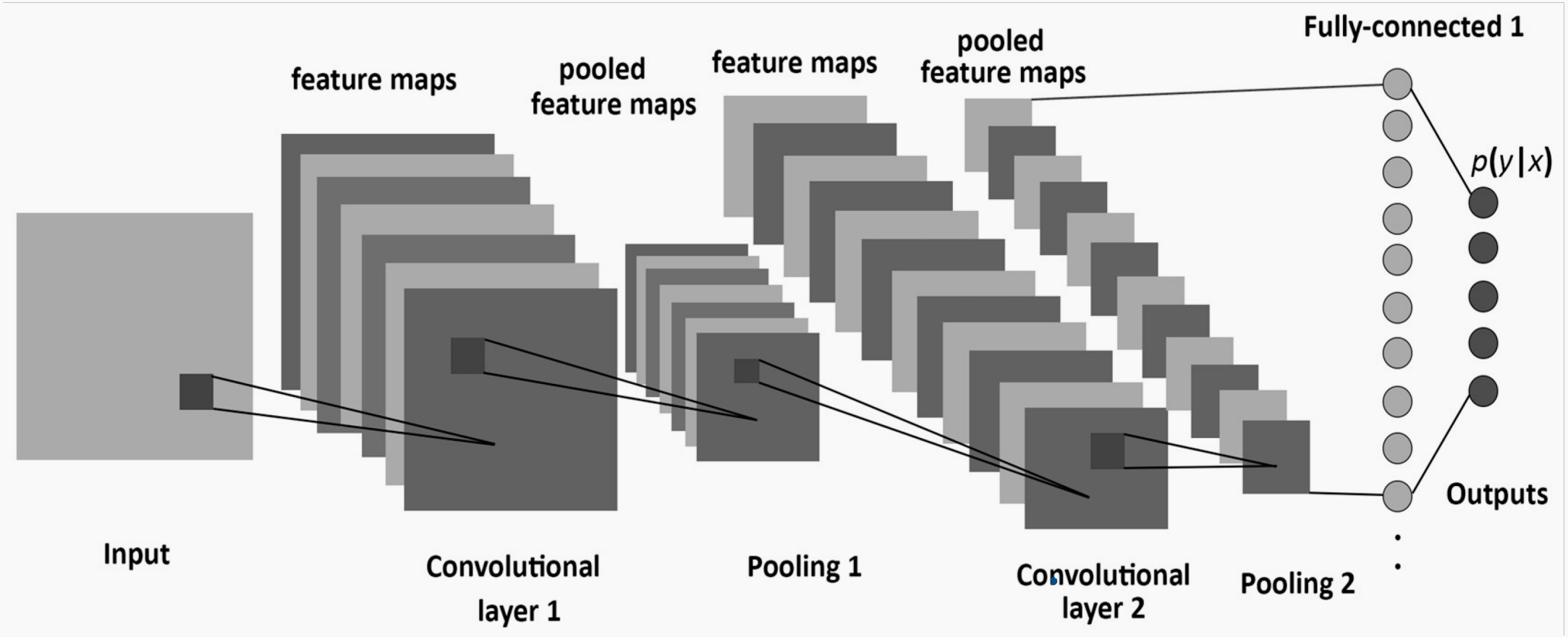
Convolutional Neural Networks

A Convolutional Network



+ ReLU

+ ReLU

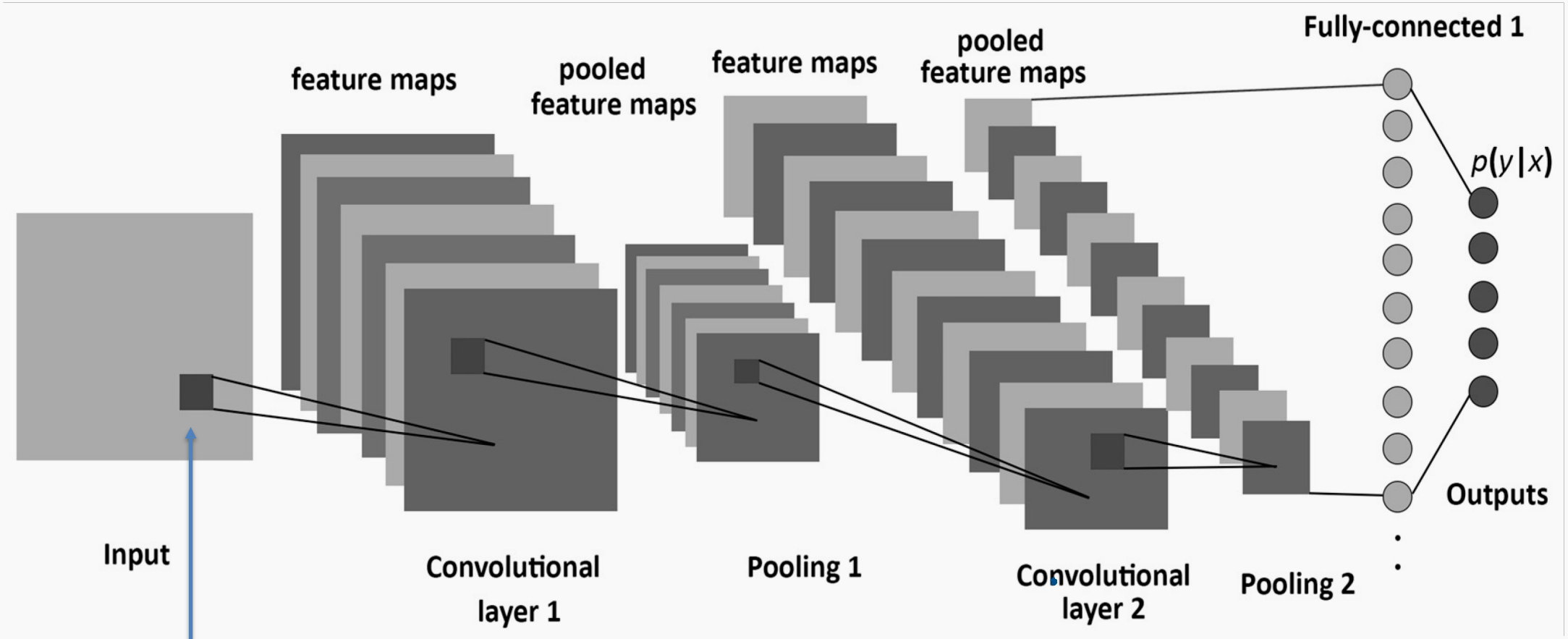
The code

In [ ]:

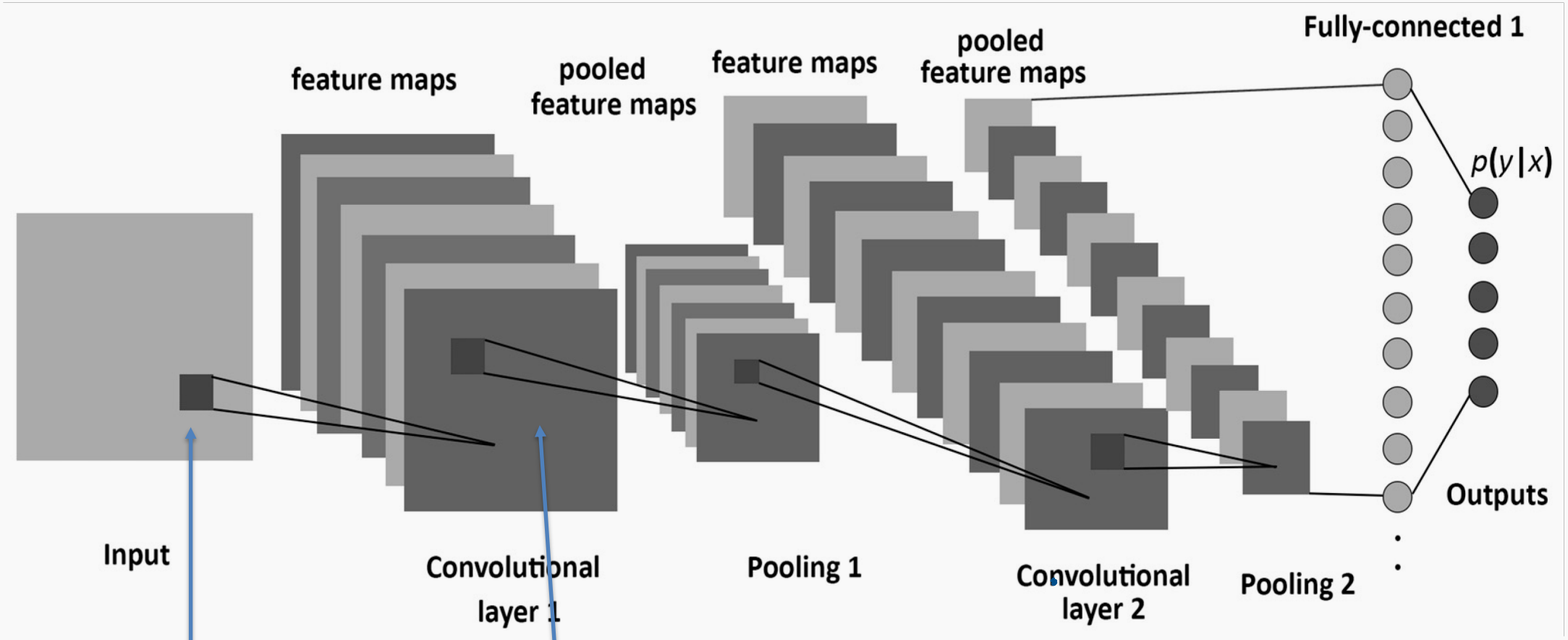
```
mnist_cnn_model = Sequential() # Create sequential model

# 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

A Convolutional Network



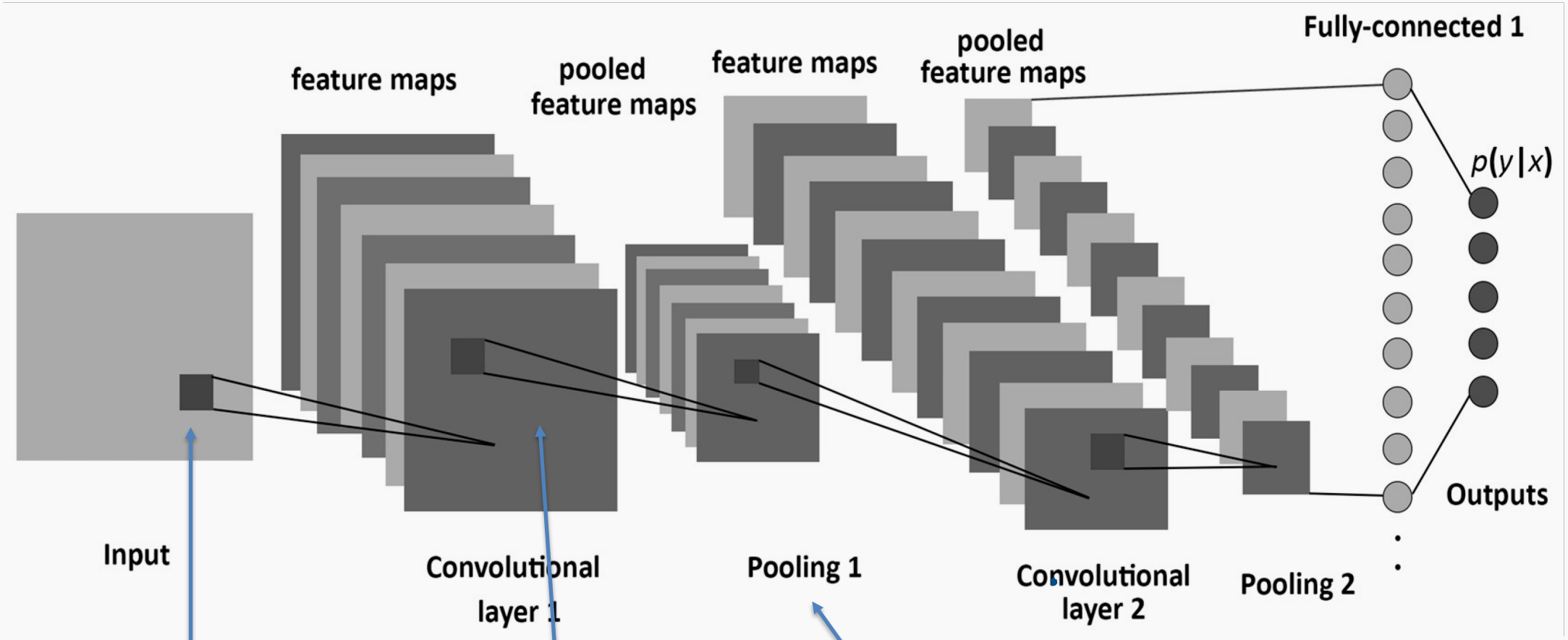
A Convolutional Network



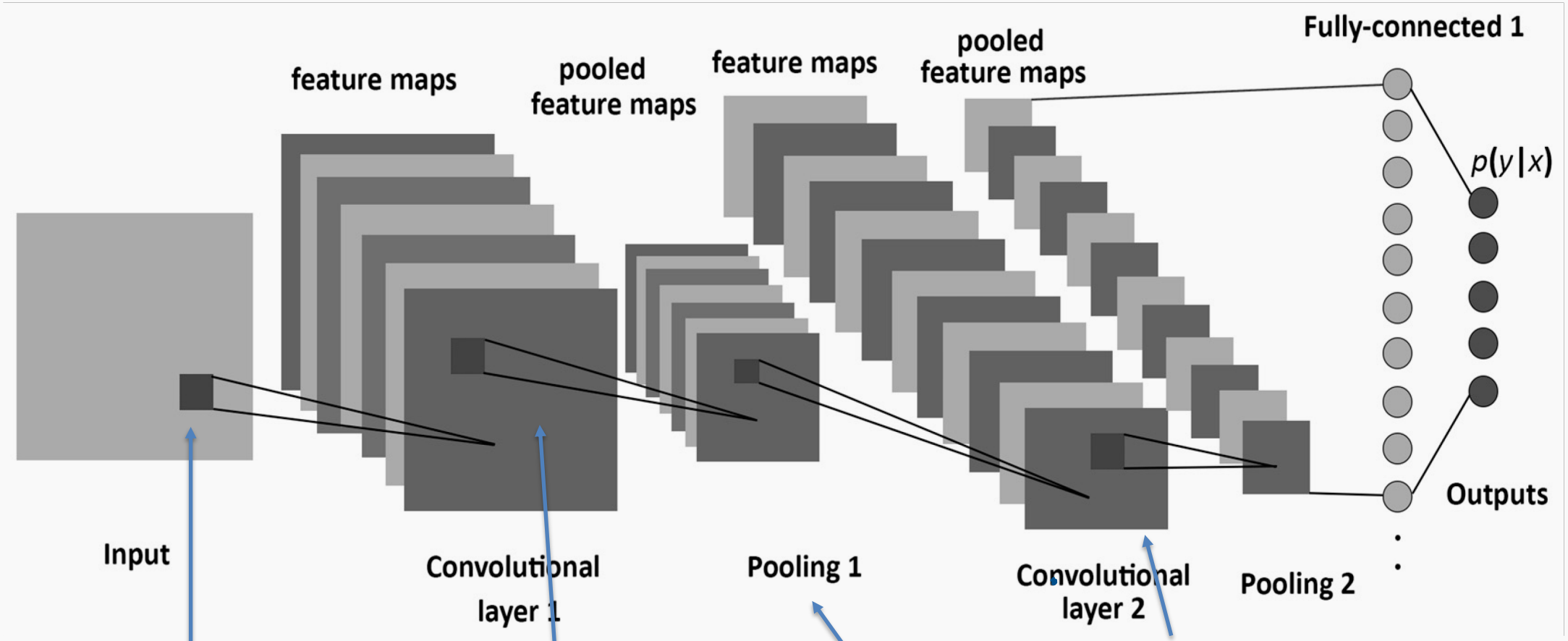
What is this square?

What are feature maps and why multiple feature maps?

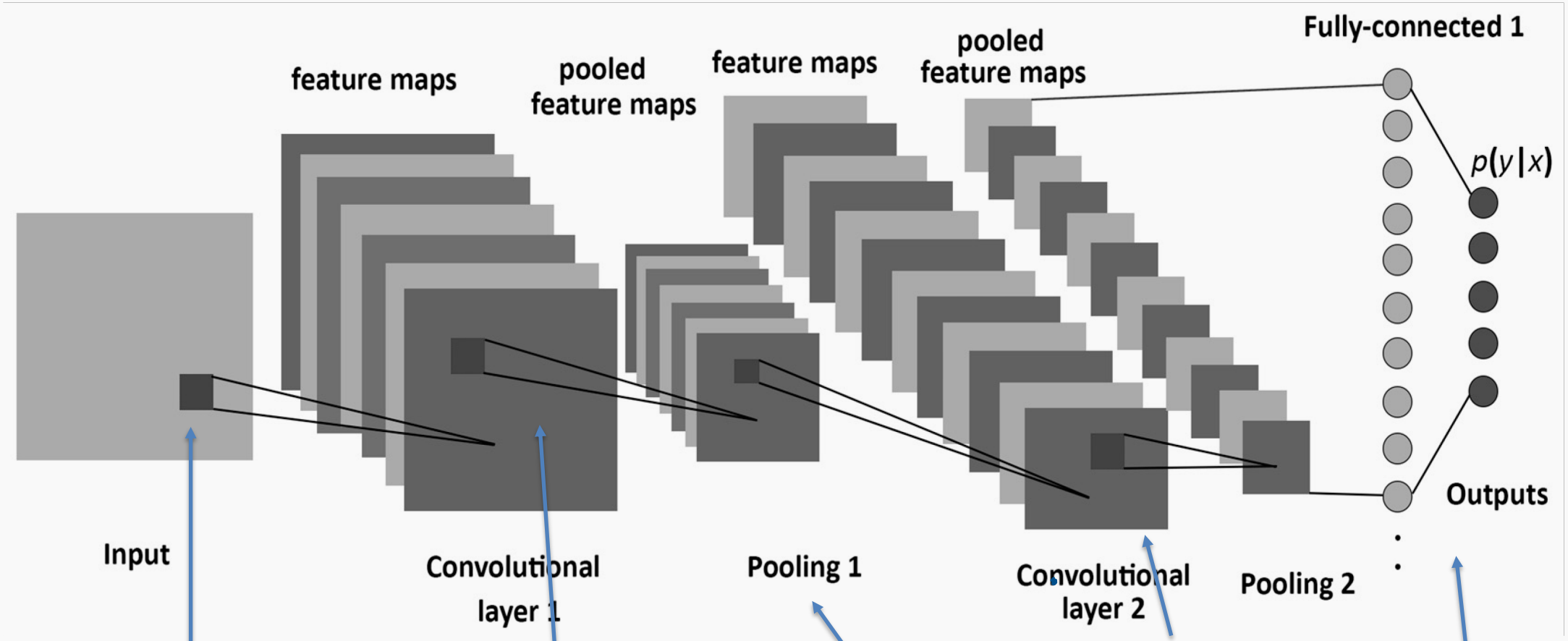
A Convolutional Network



A Convolutional Network



A Convolutional Network



What is this square?

What are feature maps and why multiple feature maps?

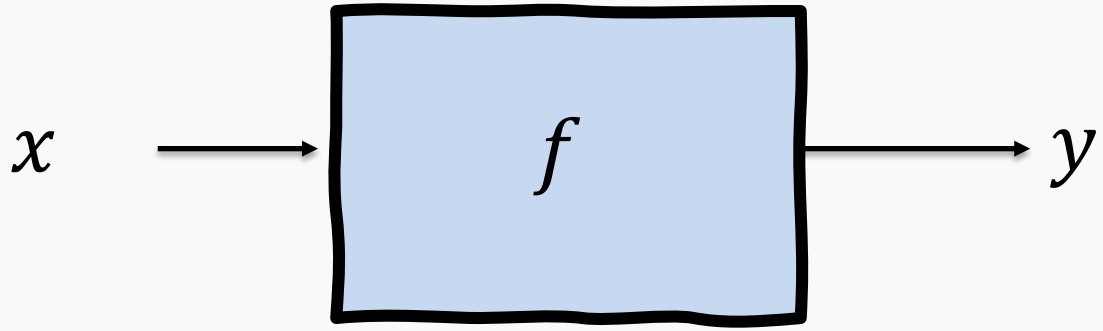
What is Pooling?

Why more than one layer?

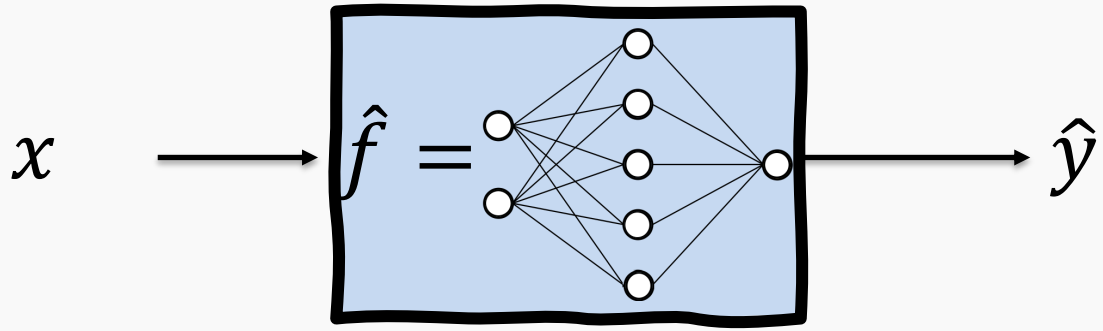
What is going on here?

Feed forward Neural Network, Multilayer Perceptron (MLP)

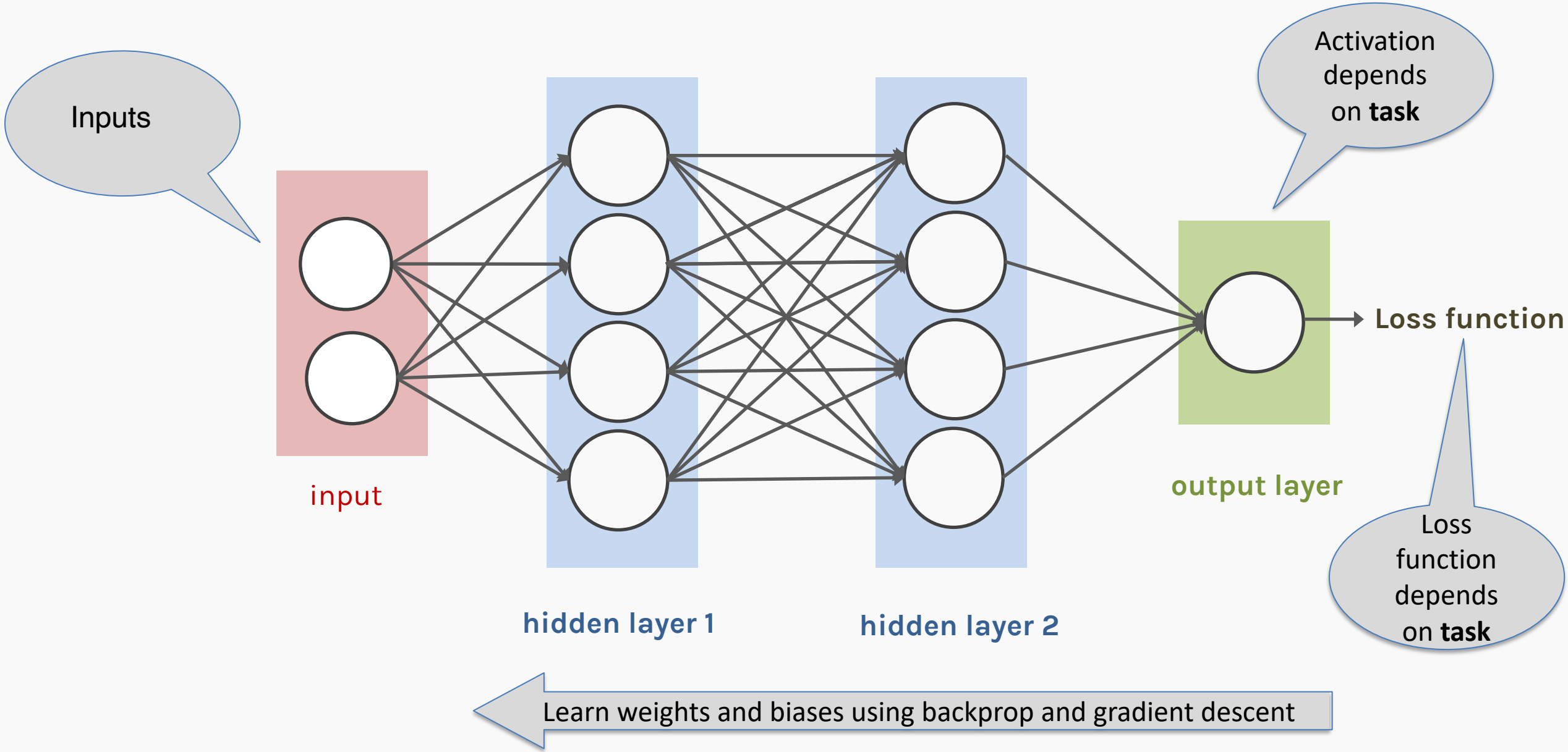
A **function** is a relation that associates each element x of a set X to a single element y of a set Y



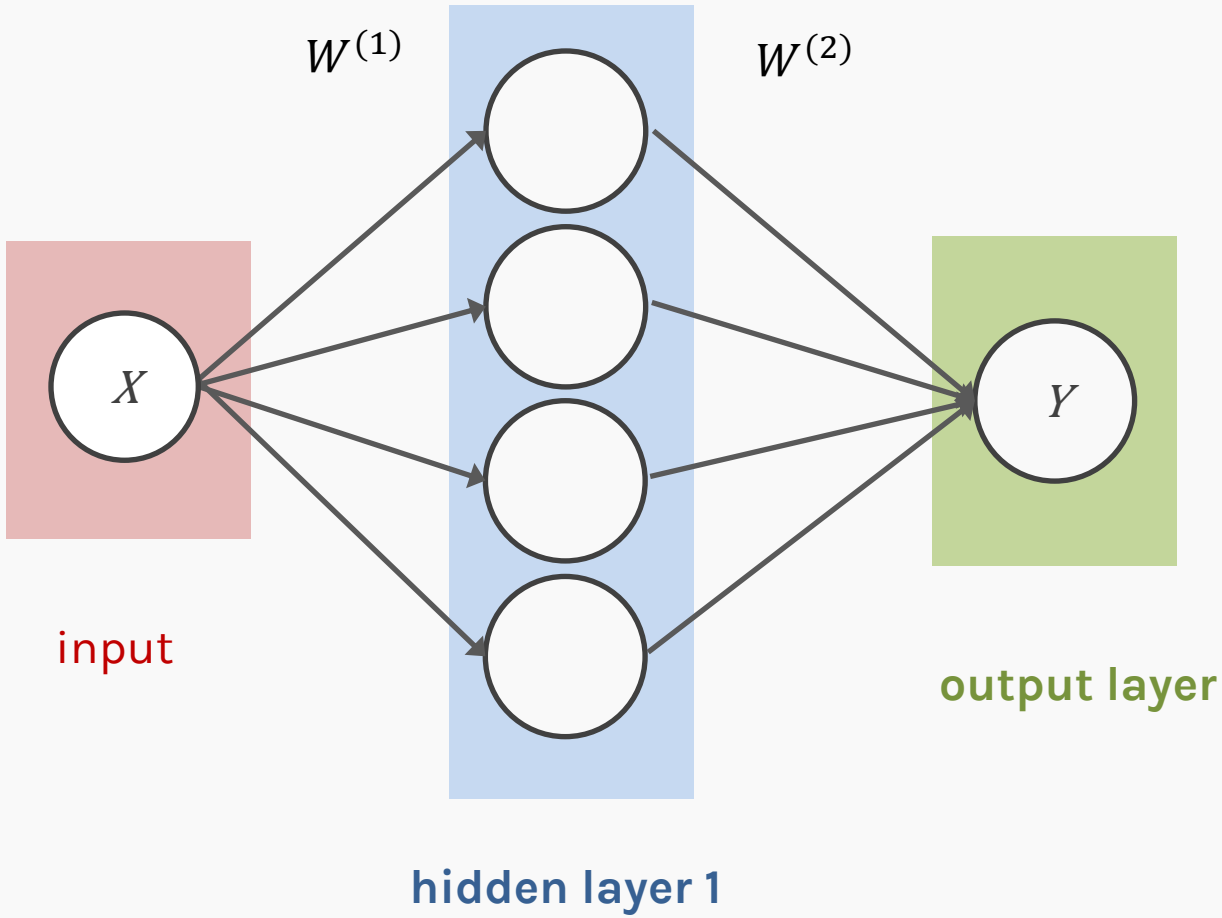
Neural networks can approximate a wide variety of functions



Quick review of MLPs



MLP as an additive model



activation

$$Y = \sum_j W_j^{(2)} f\left(W^{(1)}X + b^{(1)}\right) + b^{(2)}$$

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 $(W_j^{(1)}, \beta_j)$

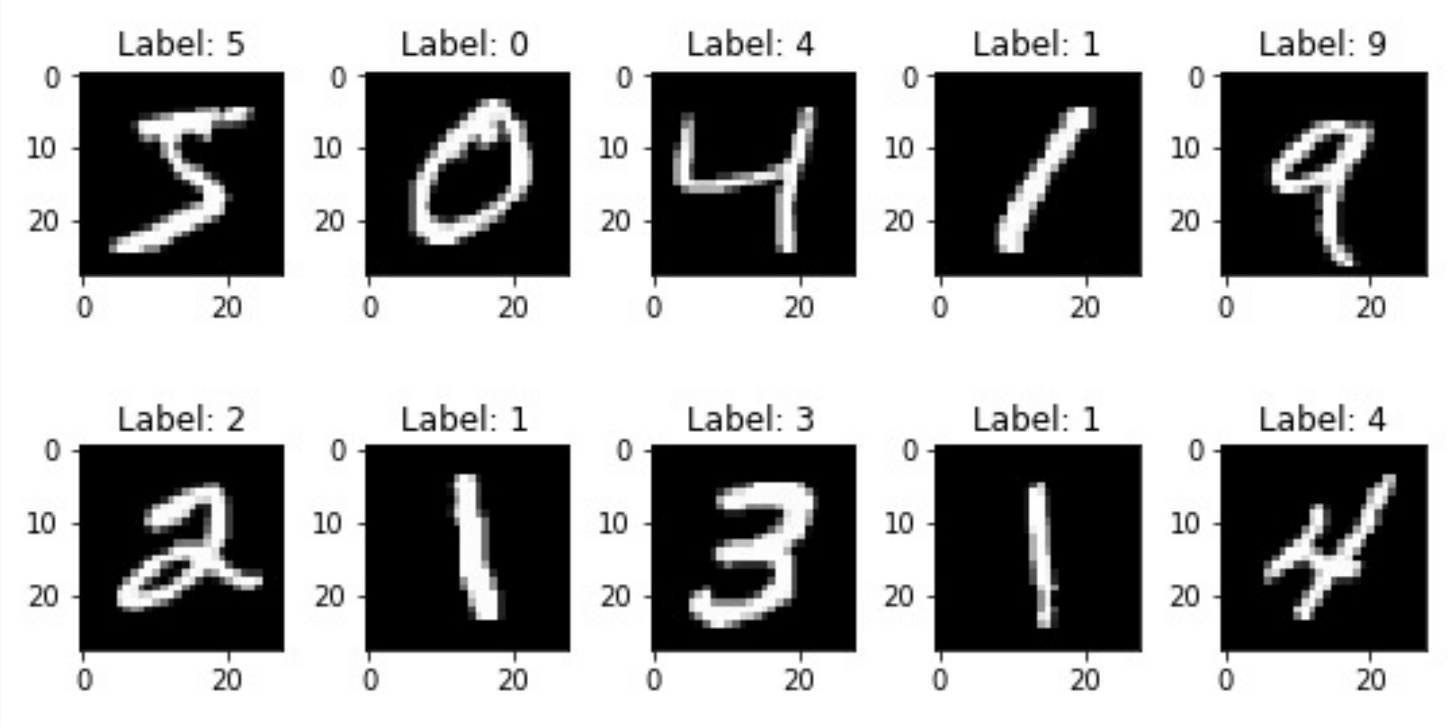
Main **drawbacks** of MLPs

- MLPs use one node for each input (e.g. pixel in an image, or 3 pixel values in RGB case). The number 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.

Common Dataset: MNIST

MNIST, a dataset of handwritten digits that are commonly used to train and test machine learning models. It contains 60,000 28x28 black and white images in 10 different classes for training and another 10,000 for testing.

Each pixel is a feature: an MLP would have $28x28x1 + 1 = 785$ weights per neuron!



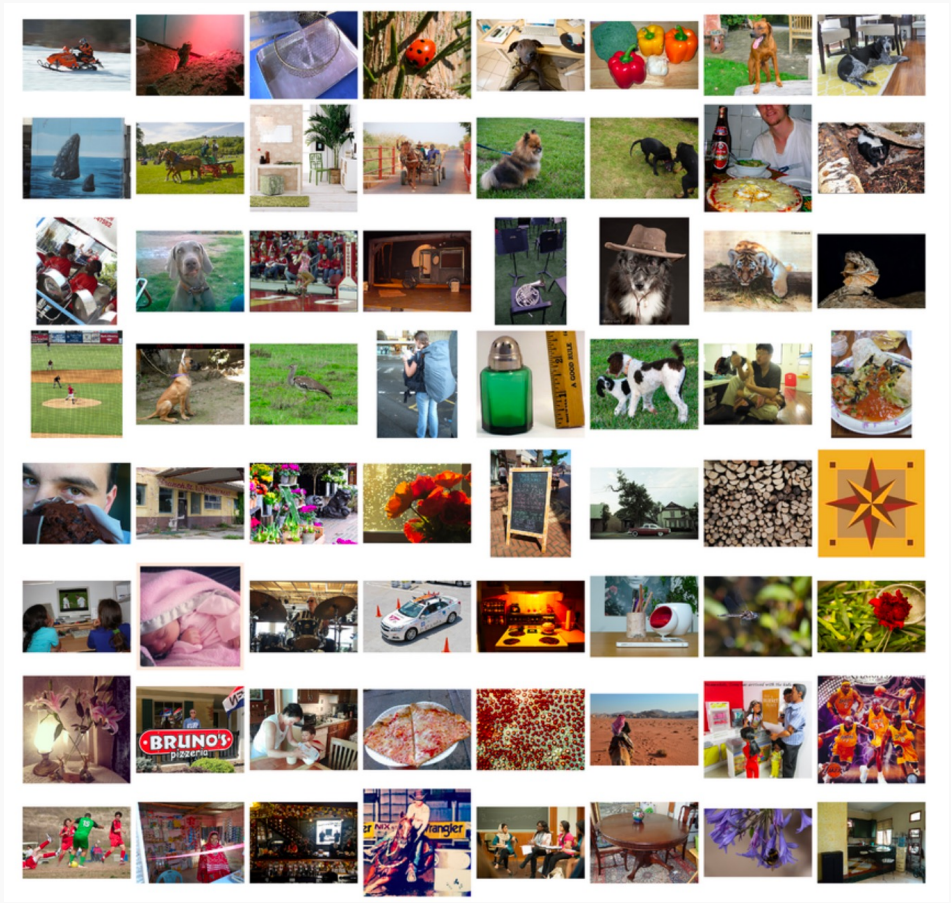
Common Dataset: CIFAR10

CIFAR10, a dataset of images that are commonly used to train machine learning models. It contains 60,000 32x32 color images in 10 different classes.

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

Common Dataset: ImageNet

Example: ImageNet, a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated into 1000 classes by the project to indicate what objects are pictured. In at least one million of the images, bounding boxes are also provided.



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 train.

Me using neural network for simple regression problem

Image analysis

Imagine that we want to recognize swans in an image:

Image analysis

Imagine that we want to recognize swans in an image:

Oval-shaped white blob (body)

Image analysis

Imagine that we want to recognize swans in an image:

Oval-shaped white blob (body)

Round, elongated oval with orange protuberance

Image analysis

Imagine that we want to recognize swans in an image:

Oval-shaped white blob (body)

Round, elongated oval with orange protuberance

Long white rectangular shape (neck)

Cases can be a bit more complex…

Cases can be a bit more complex…

Round, elongated head with orange or black beak

Cases can be a bit more complex…

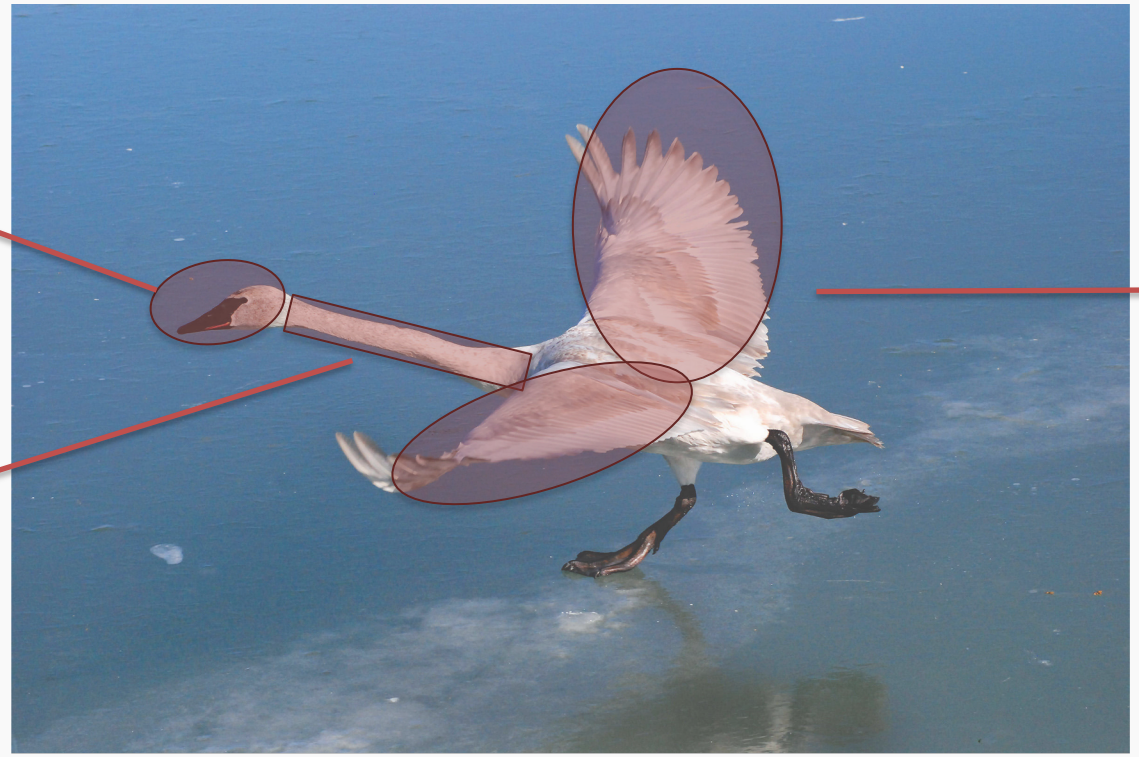
Round, elongated head with orange or black beak

Long white neck, square shape

Cases can be a bit more complex...

Round, elongated head with orange or black beak

Long white neck, square shape

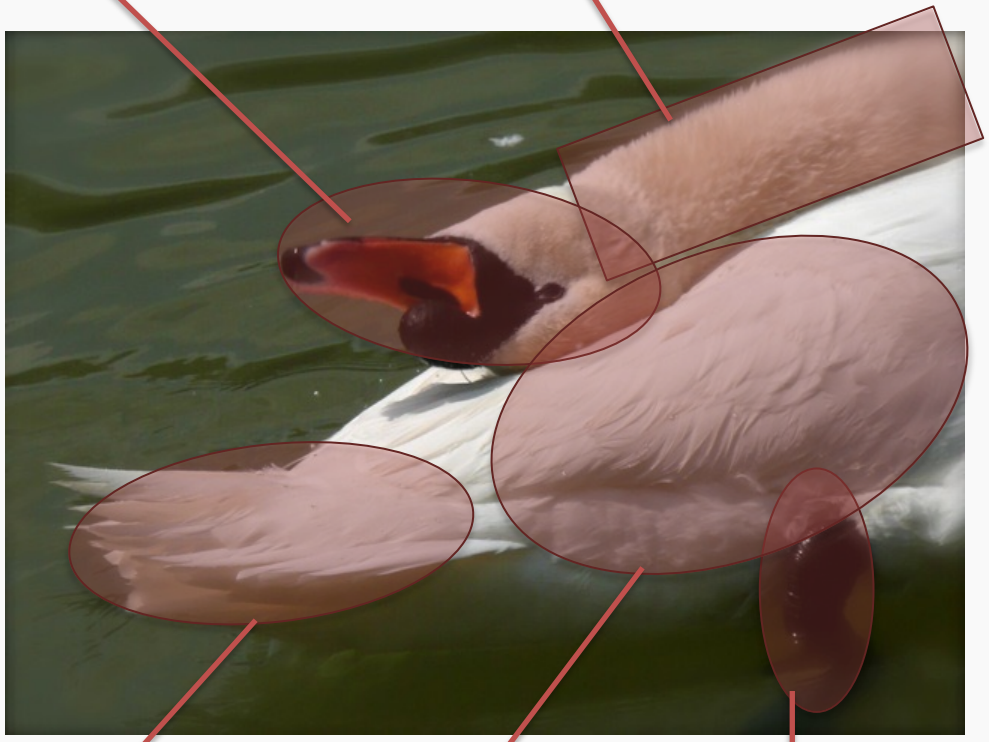


Oval-shaped white body with or without large white symmetric blobs (wings)

Now what?

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

Long white neck, can bend around, not necessarily straight



White tail, generally far from the head, looks feathery

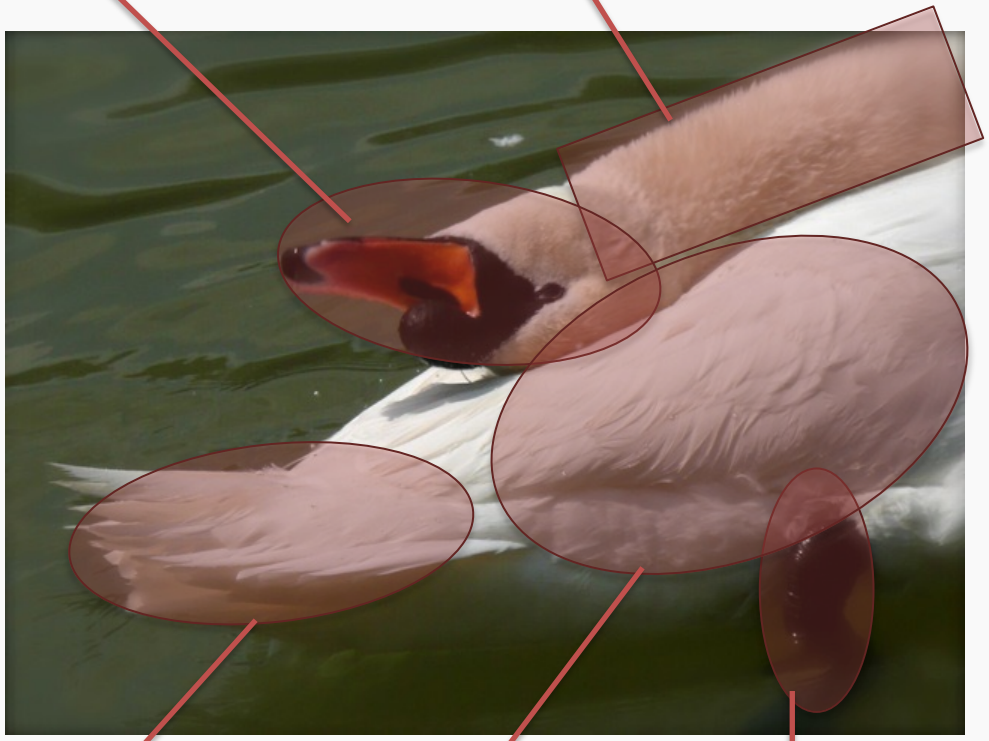
White, oval shaped body, with or without wings visible

Black feet, under body, can have different shapes

Now what?

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

Long white neck, can bend around, not necessarily straight



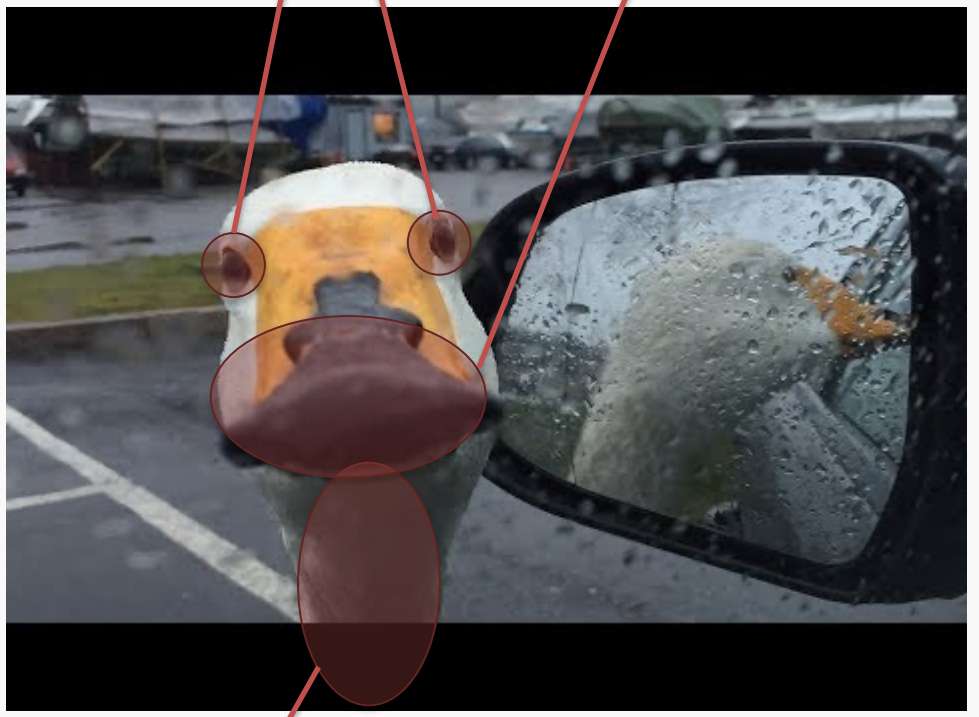
White tail, generally far from the head, looks feathery

White, oval shaped body, with or without wings visible

Black feet, under body, can have different shapes

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

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



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

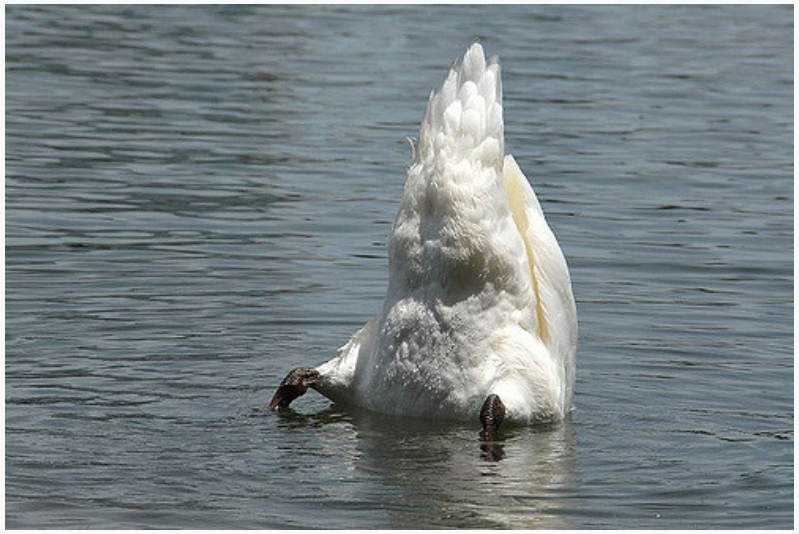
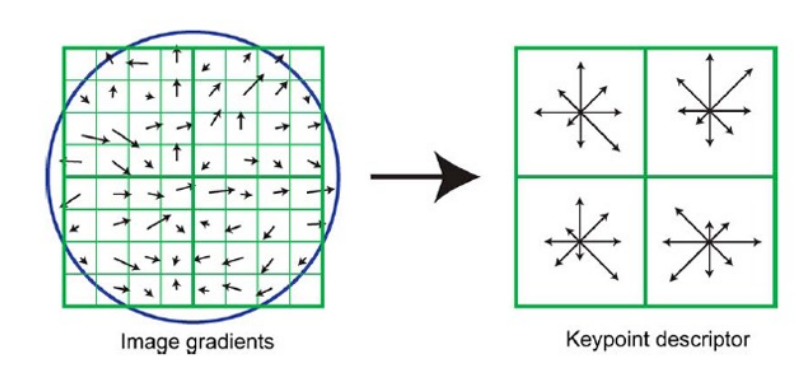
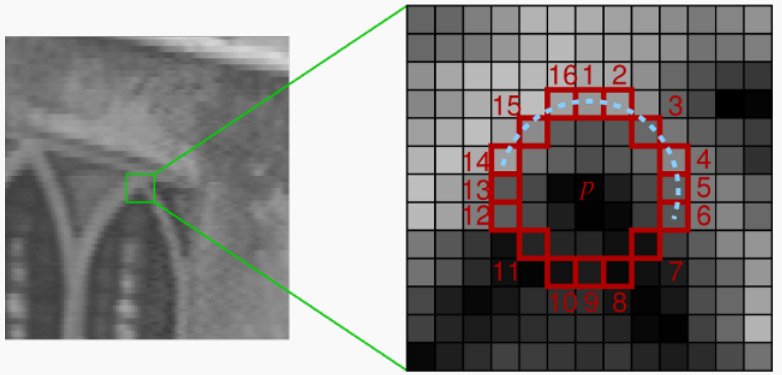


Image features

- 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?
- 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 Dictionary learning
- Neural Networks!

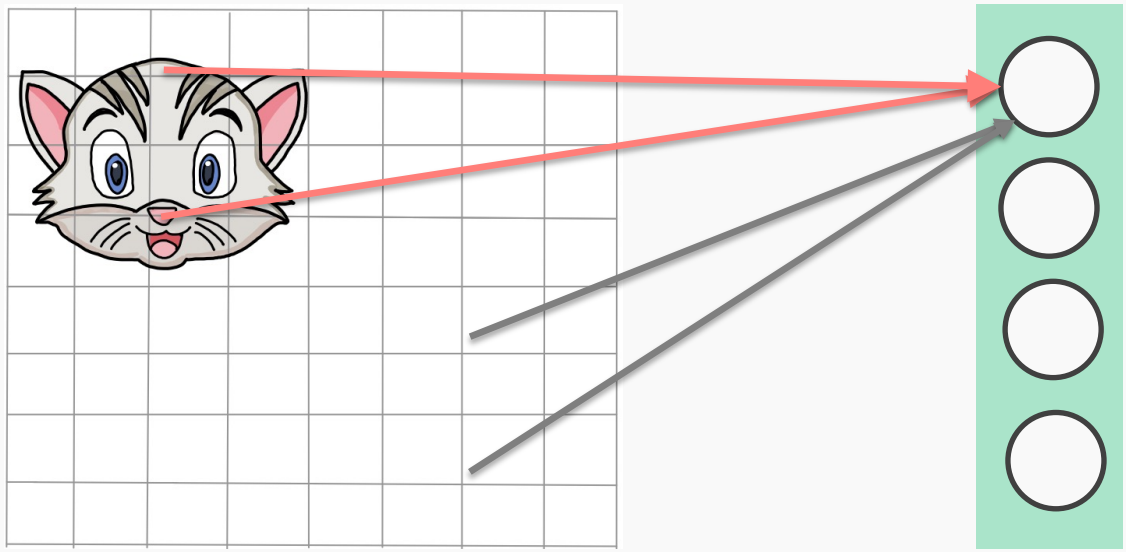
Some extra things to consider

- Nearby Pixels are more strongly related that distant ones
- Objects are built up out of smaller parts
- Images are Local and Hierarchical

And images are invariant

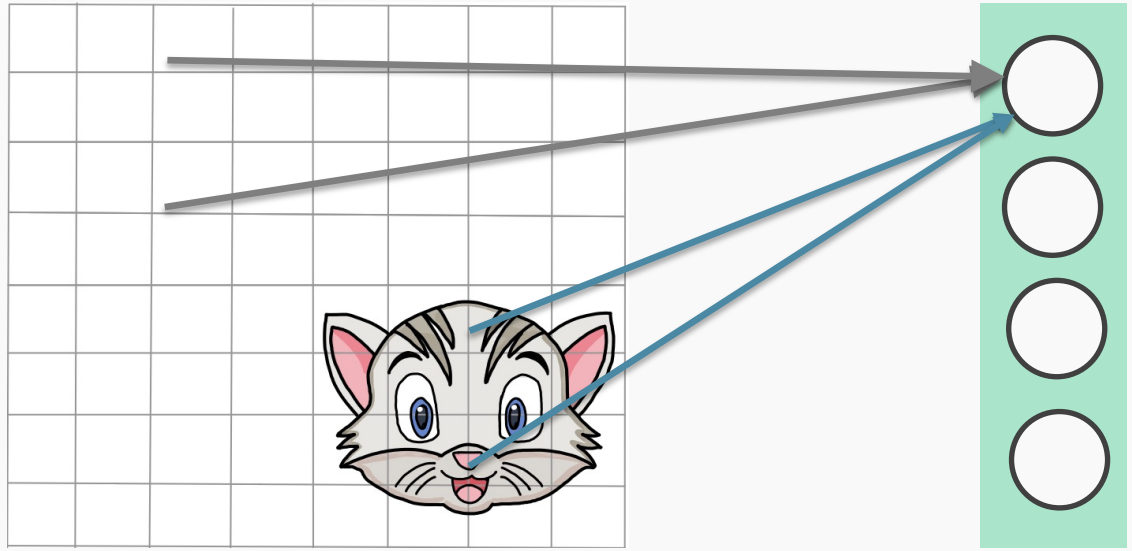
Each neuron from first layer has one weight per pixel. Recall, the importance of the predictors (here pixels) is given by the value of the coefficient, here the weight W.

X



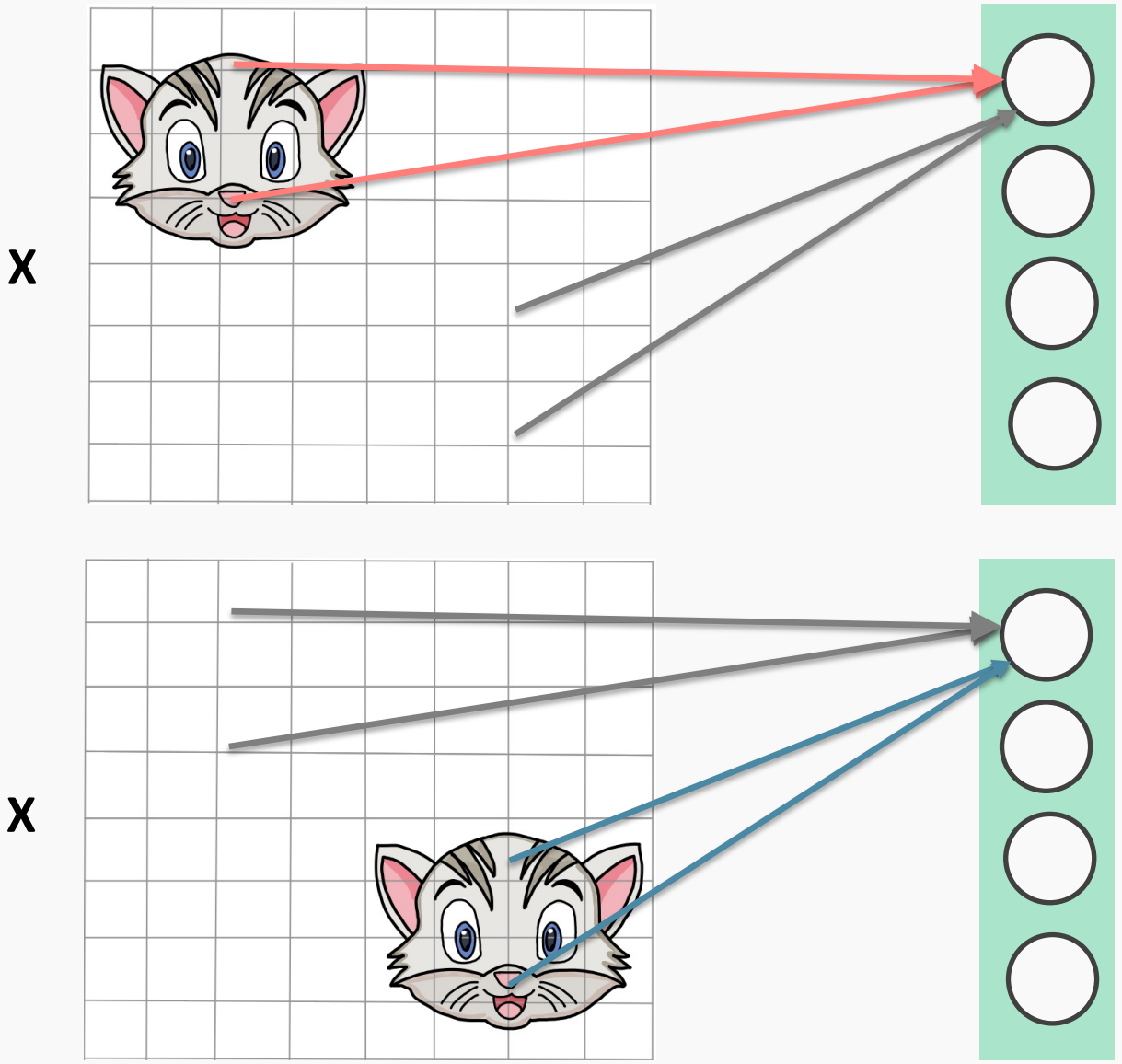
In this case, the red weights will be larger to better recognize cat.

X



In this case, the blue weights will be larger.

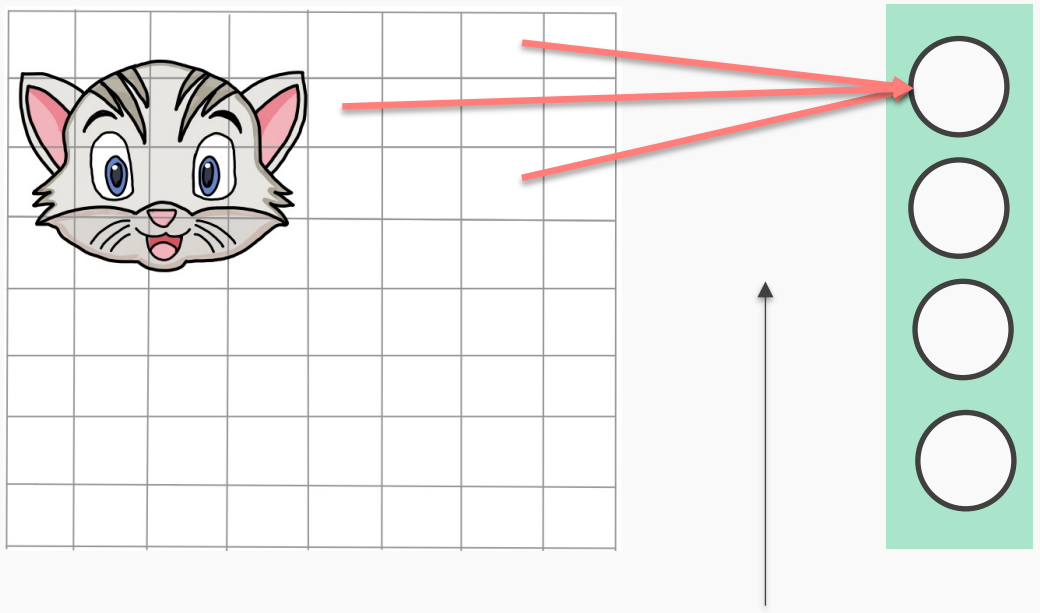
Each neuron from first layer has one weight per pixel. Recall, the importance of the predictors (here pixels) is given by the value of the coefficient, here the weight W.



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

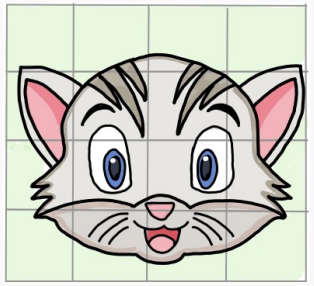
Solution: Cut the image to smaller pieces.

$X: \mathbb{R}^{8\times 8}$

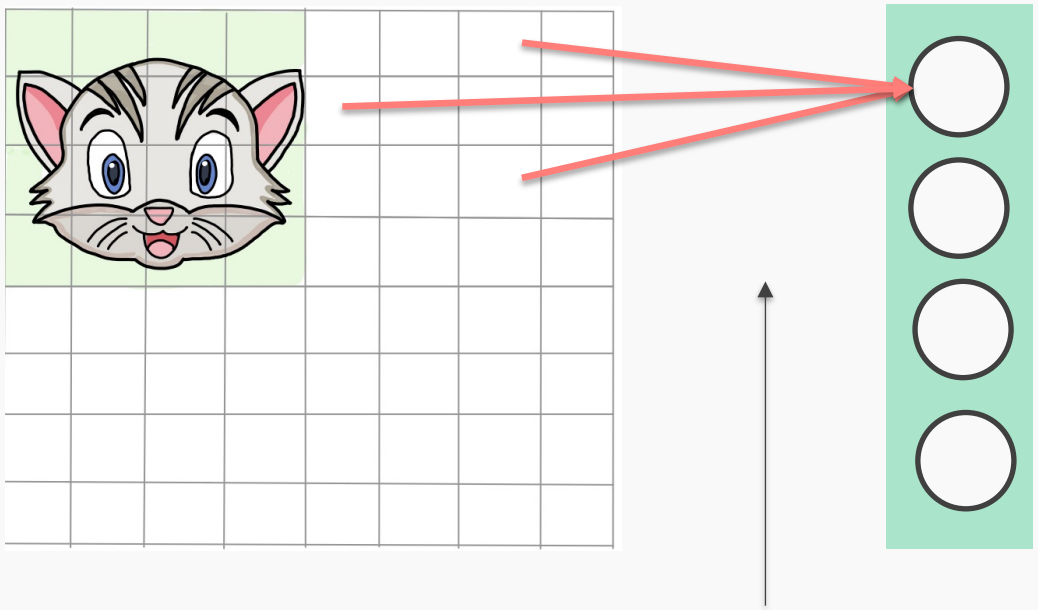


Solution: Cut the image to smaller pieces.

$X: \mathbb{R}^{4\times4}$

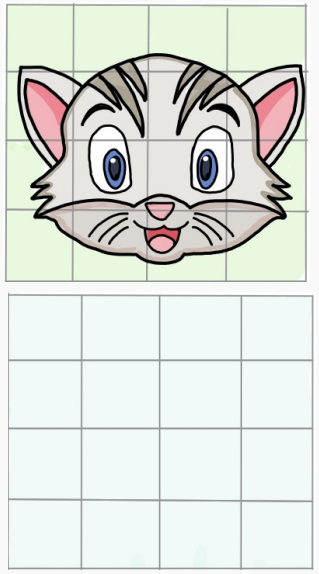


$X: \mathbb{R}^{8\times8}$

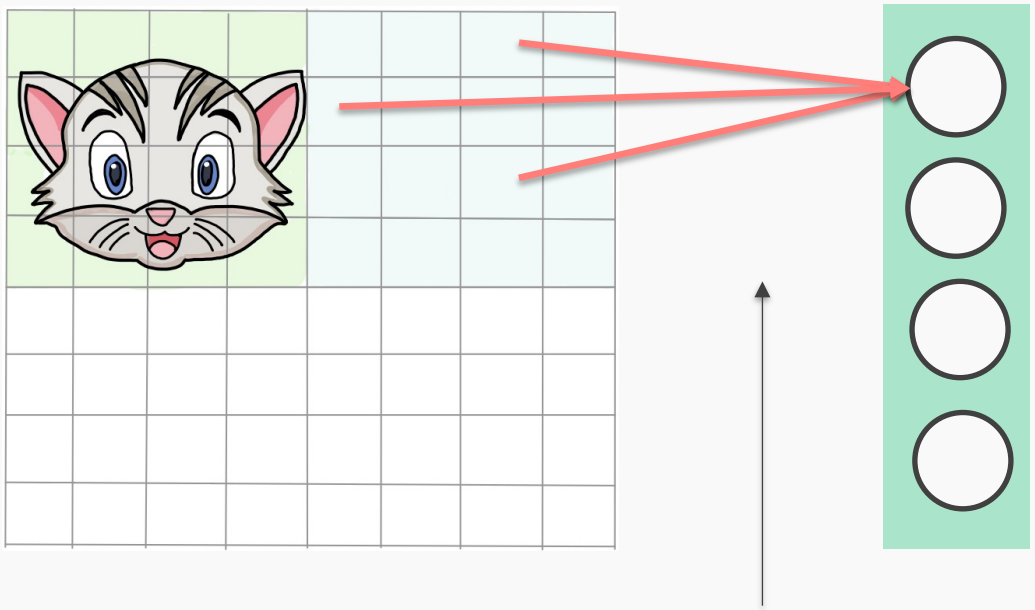


Solution: Cut the image to smaller pieces.

$X: \mathbb{R}^{4\times4}$



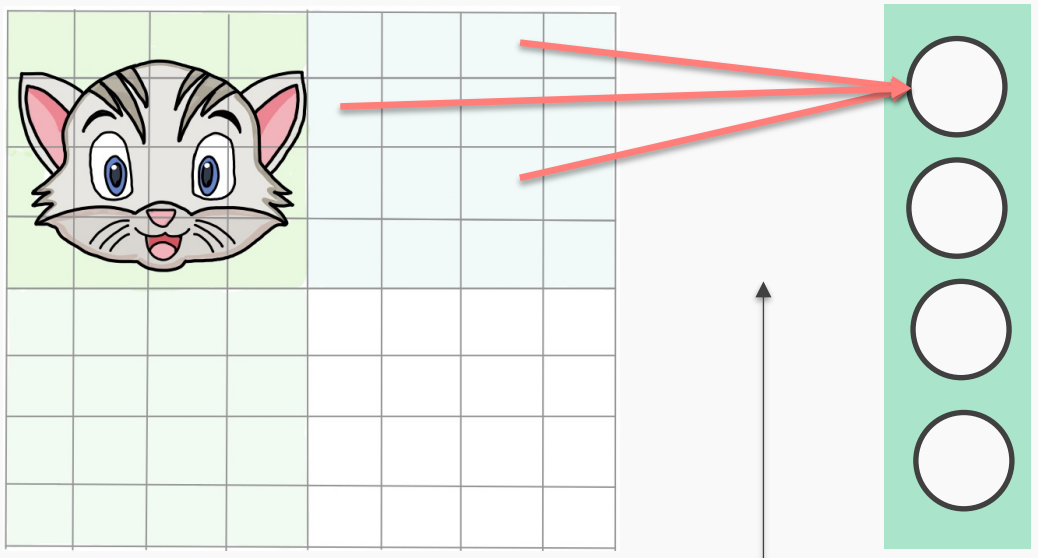
$X: \mathbb{R}^{8\times8}$



64 weights per neuron

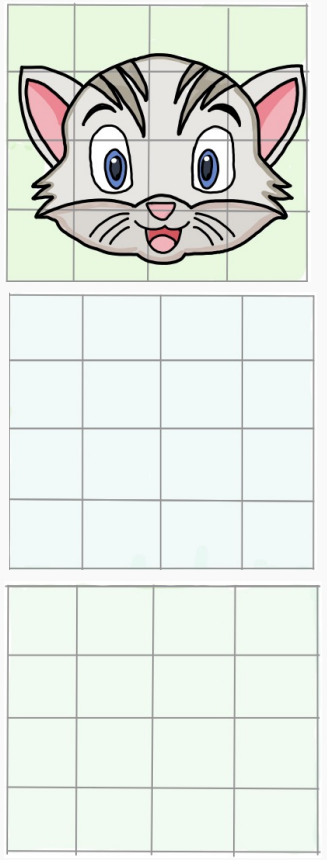
Solution: Cut the image to smaller pieces.

$X: \mathbb{R}^{8\times 8}$



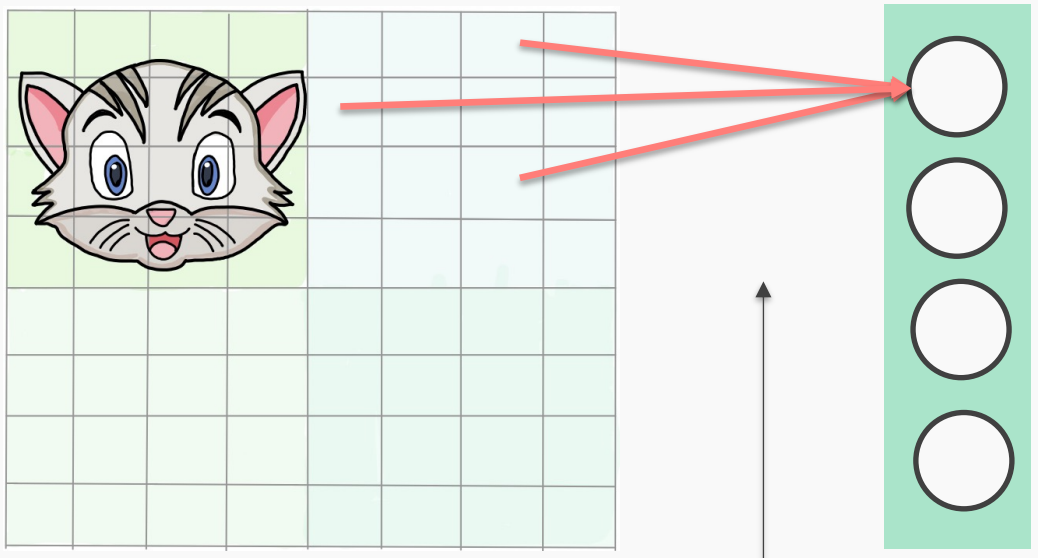
64 weights per neuron

$X: \mathbb{R}^{4\times 4}$



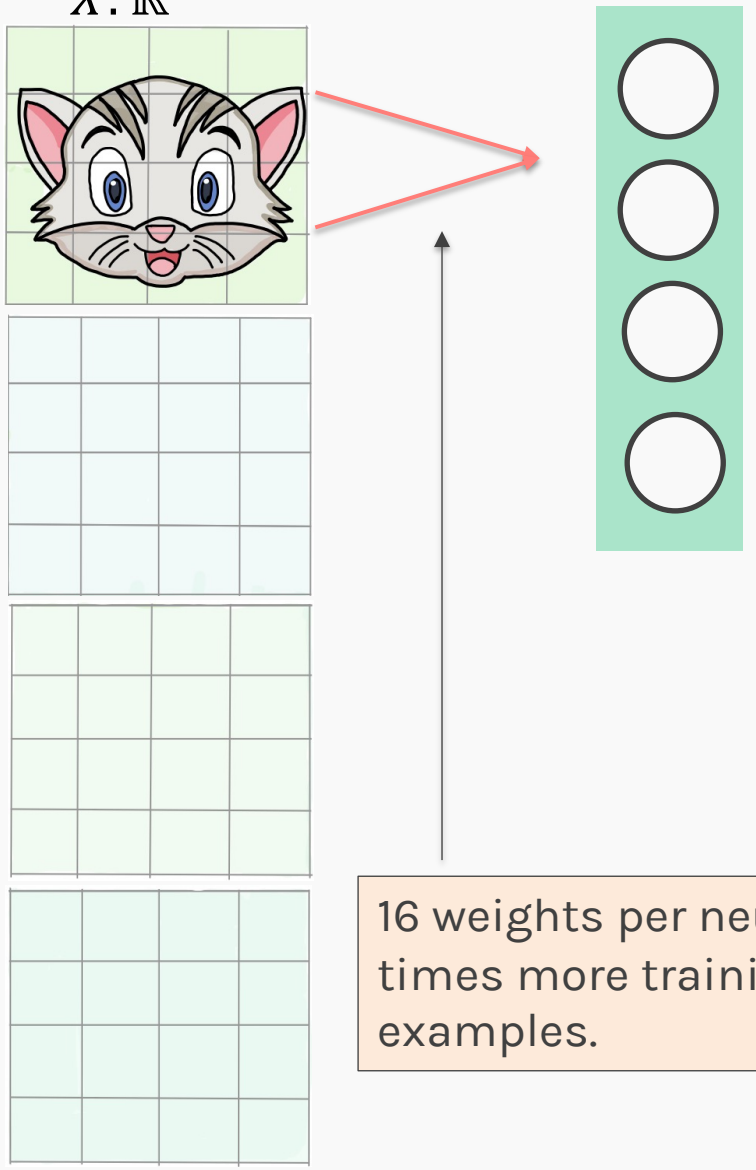
Solution: Cut the image to smaller pieces.

$X: \mathbb{R}^{8\times 8}$



64 weights per neuron

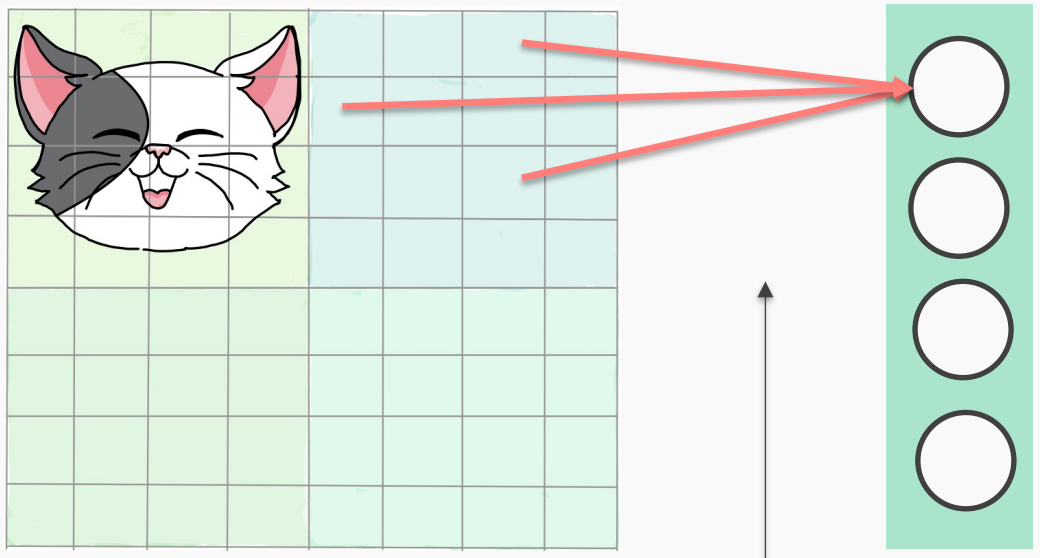
$X: \mathbb{R}^{4\times 4}$



16 weights per neuron but 4 times more training examples.

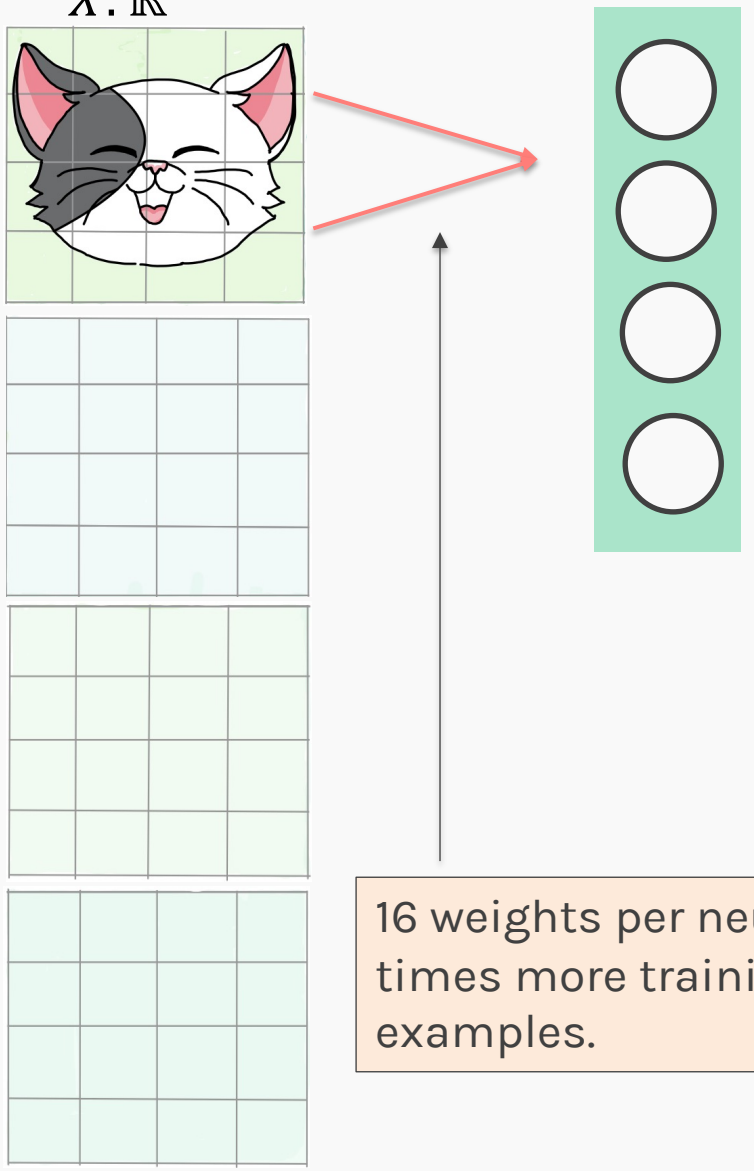
Do the same for all images

$X: \mathbb{R}^{8\times8}$



64 weights per neuron

$X: \mathbb{R}^{4\times4}$

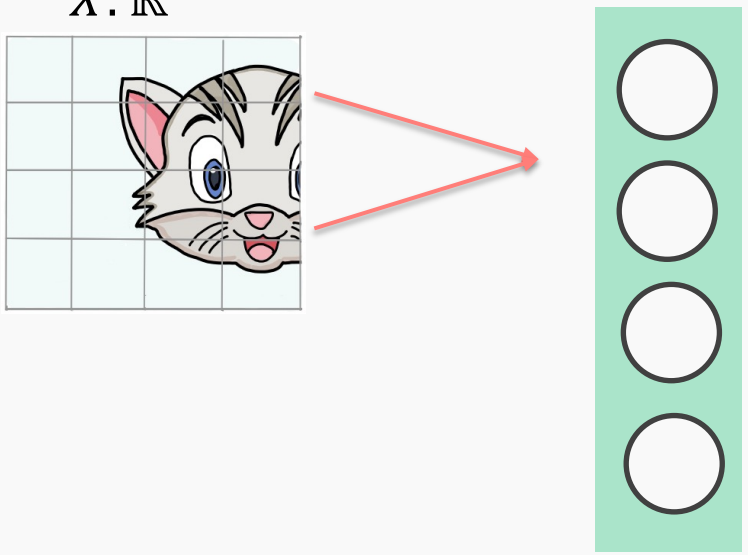


16 weights per neuron but 4 times more training examples.

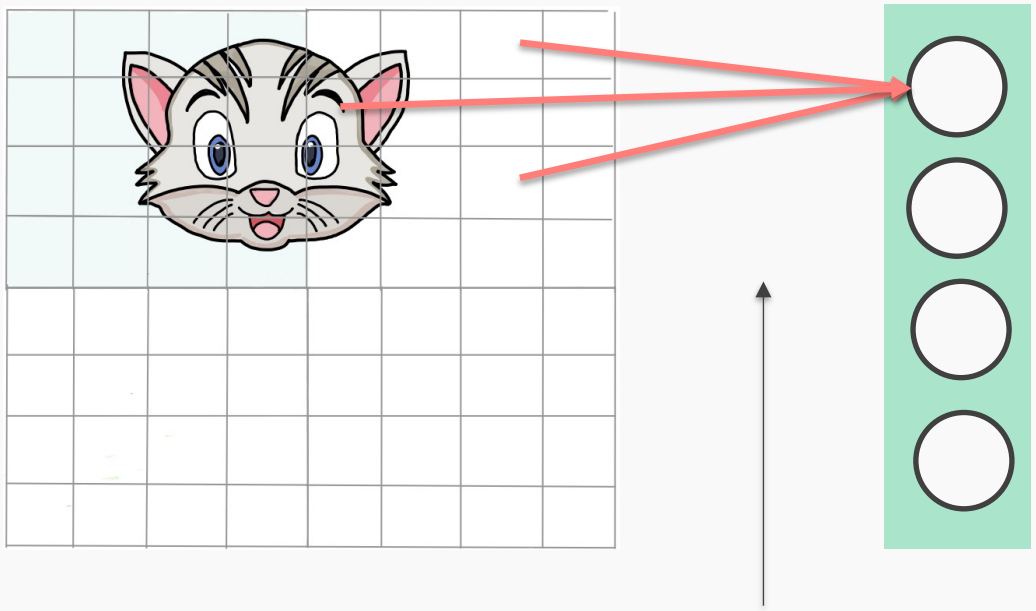
What if the cat is not entirely in one of the 4 boxes?

What if the cat is not entirely in one of the 4 boxes?

$X: \mathbb{R}^{4\times4}$



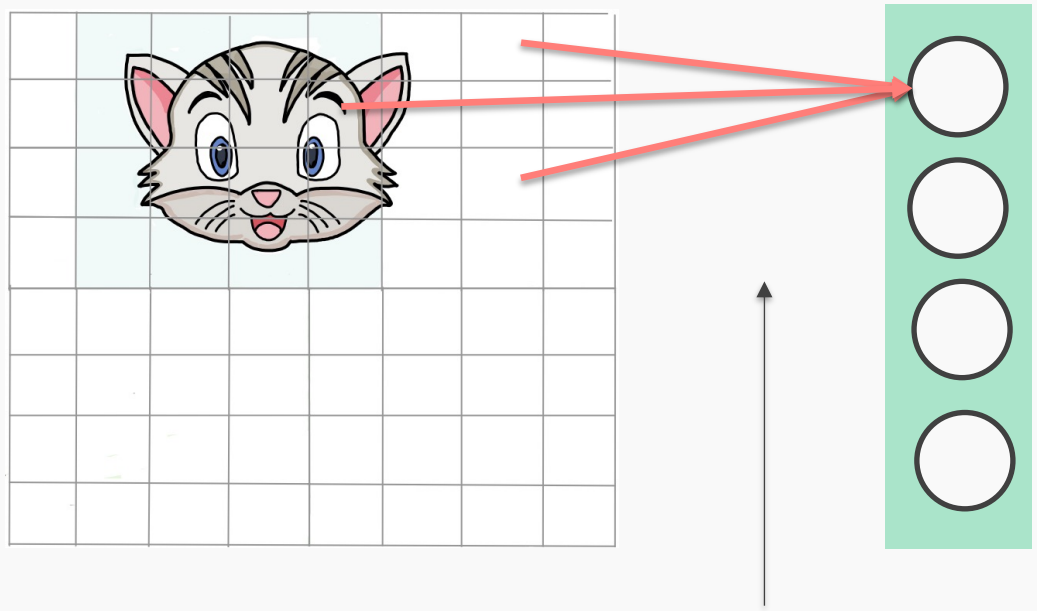
$X: \mathbb{R}^{8\times8}$



64 weights per neuron

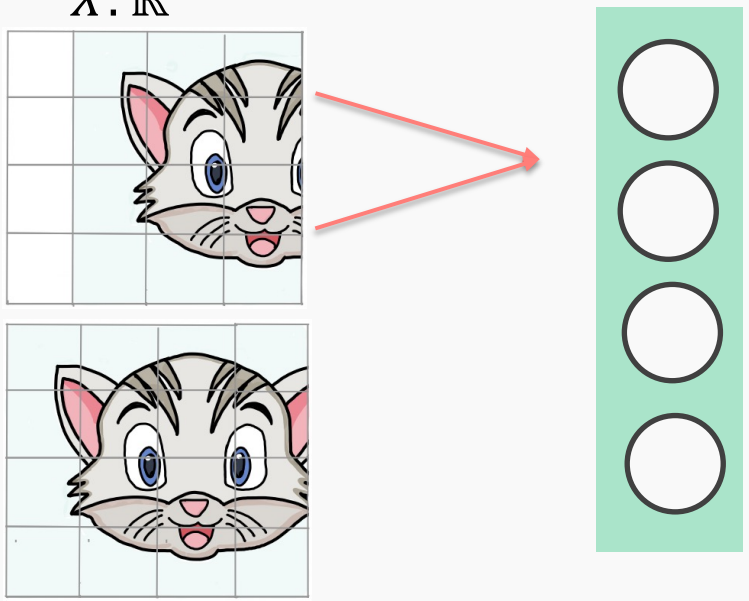
What if the cat is not entirely in one of the 4 boxes?

$X: \mathbb{R}^{8\times 8}$



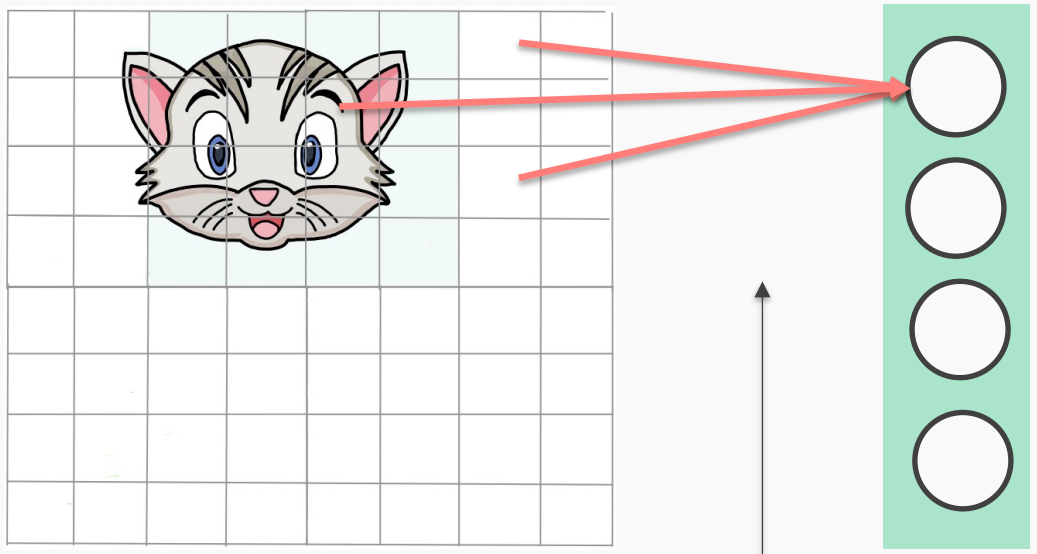
64 weights per neuron

$X: \mathbb{R}^{4\times 4}$



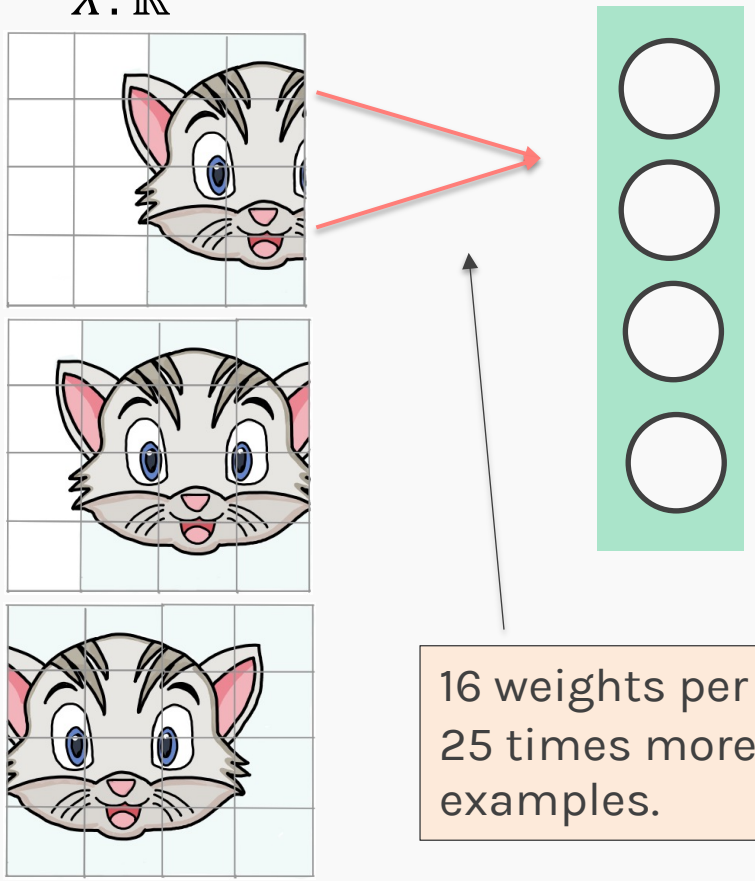
What if the cat is not entirely in one of the 4 boxes?

$X: \mathbb{R}^{8\times 8}$



64 weights per neuron

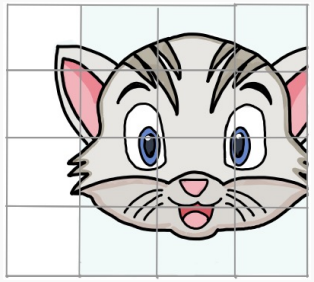
$X: \mathbb{R}^{4\times 4}$

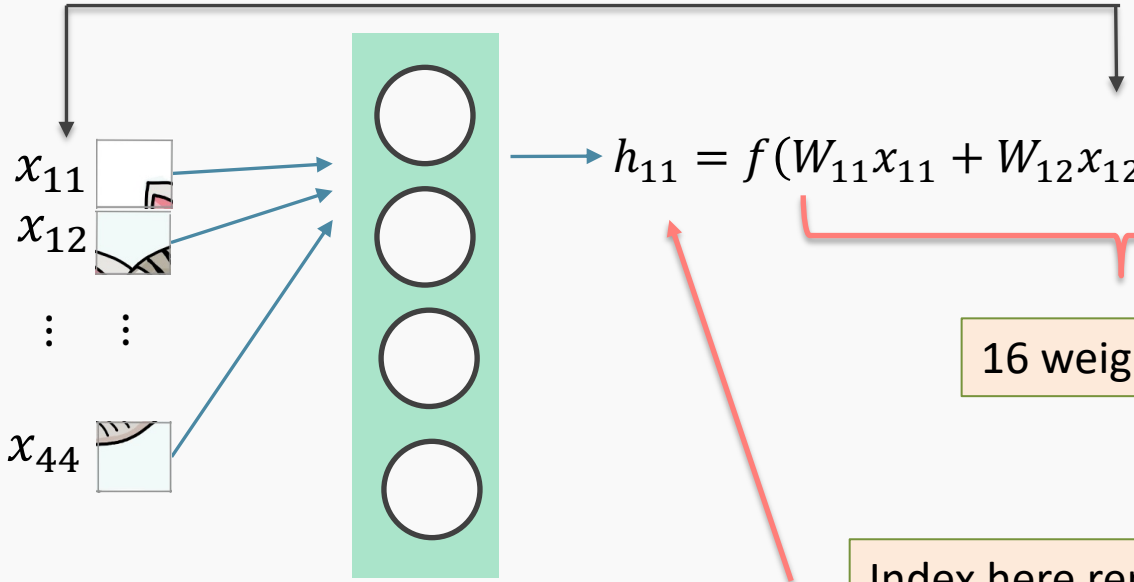


16 weights per neuron but 25 times more training examples.

Sliding Window

Convolution





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

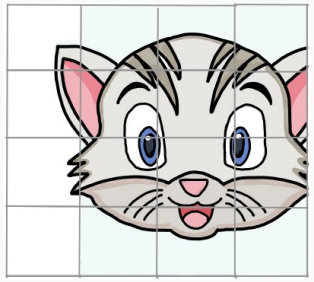
16 weights

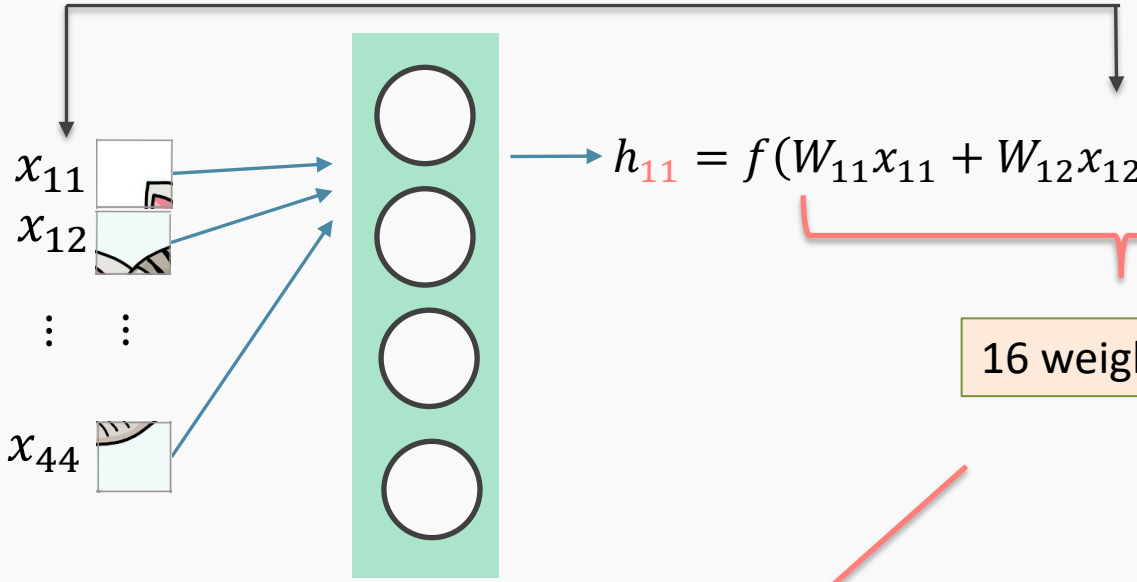
Index here represents the cutout

Input

Hidden Layer

Convolution

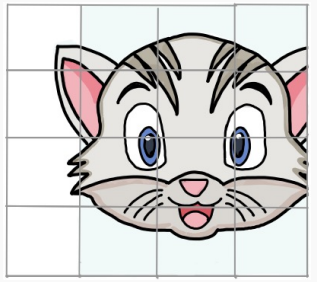


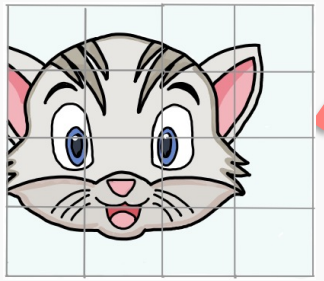


$$h_{11} = \sum \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ & & & \\ & & & \\ & & & W_{44} \end{bmatrix} \star \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ & & & \\ & & & \\ & & & x_{44} \end{bmatrix}$$

Element wise multiplication and addition of all products

Convolution





x_{11}
x_{12}
$\vdots \quad \vdots$
x_{44}

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

16 weights

Input

Hidden Layer

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

Element wise multiplication and addition of all products

Convolution

$$h_{ij} = \sum$$

W_{11}	W_{12}	W_{13}	W_{14}
			W_{44}

$\star$

x_{11}	x_{12}	x_{13}	x_{14}
			x_{44}

Convolution

$$h_{ij} = \sum$$

W_{11}	W_{12}	W_{13}	W_{14}
			W_{44}

$\star$

x_{11}	x_{12}	x_{13}	x_{14}
			x_{44}

KERNEL, K

X is the cutout of image center at $\{i, j\}$

Convolution

$$h_{ij} = \sum$$

W_{11}	W_{12}	W_{13}	W_{14}
			W_{44}

$\star$

x_{11}	x_{12}	x_{13}	x_{14}
			x_{44}

Index here represents the output from this operation

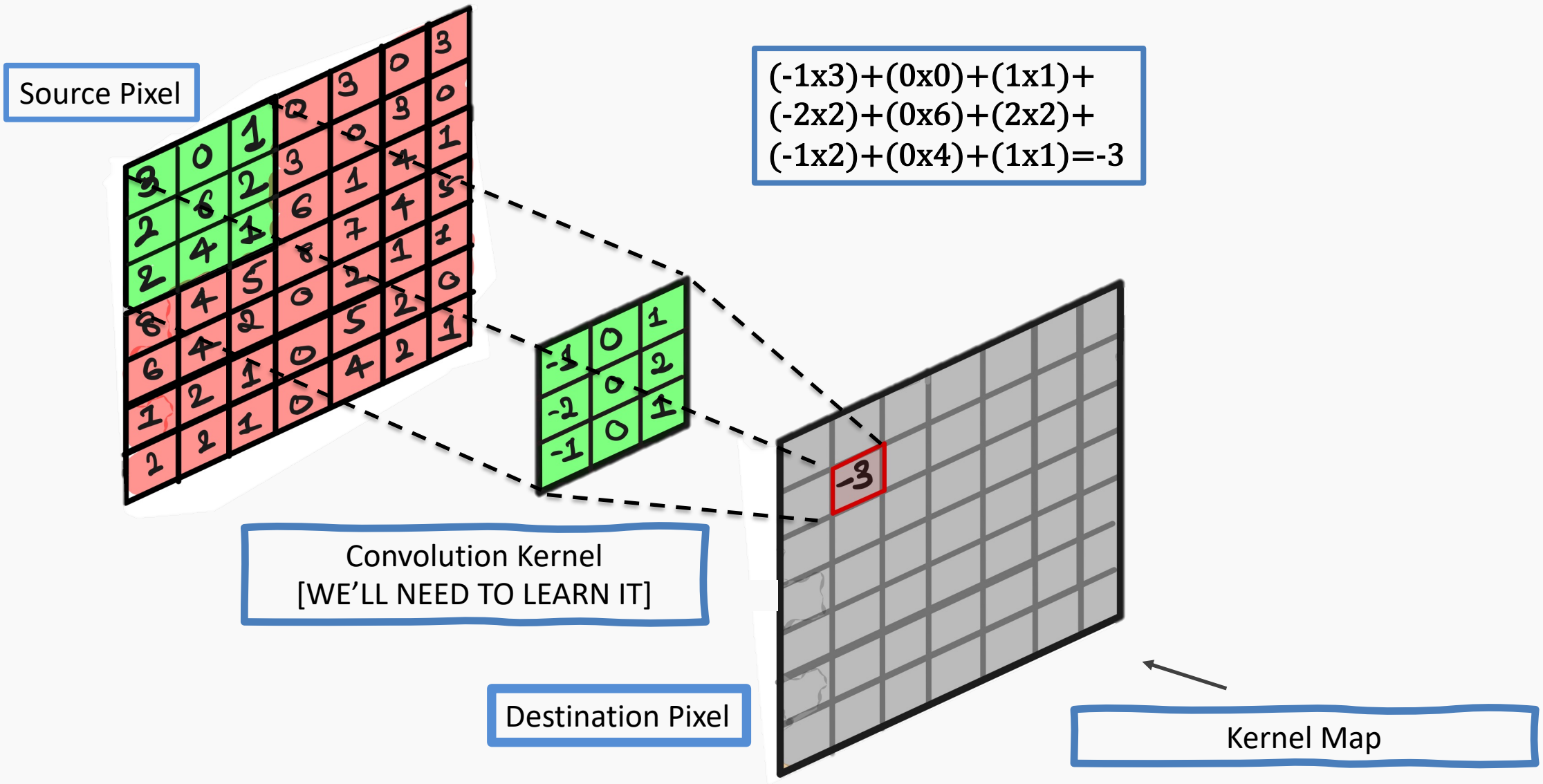
KERNEL, K

X is the cutout of image center at $\{i, j\}$

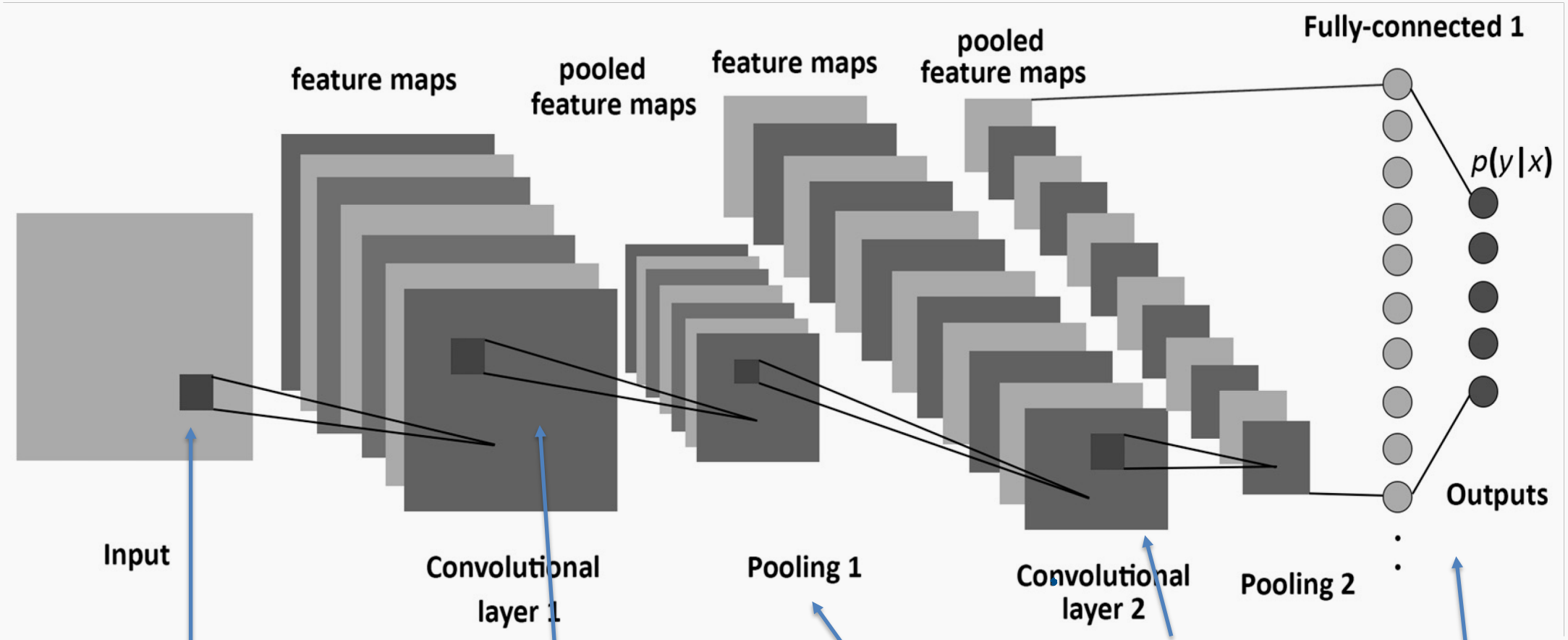
Element wise multiplication and addition of all products = CONVOLUTION

$$H = K \star \mathrm{X}$$

“Convolution” Operation



A Convolutional Network



~~What is this square?~~

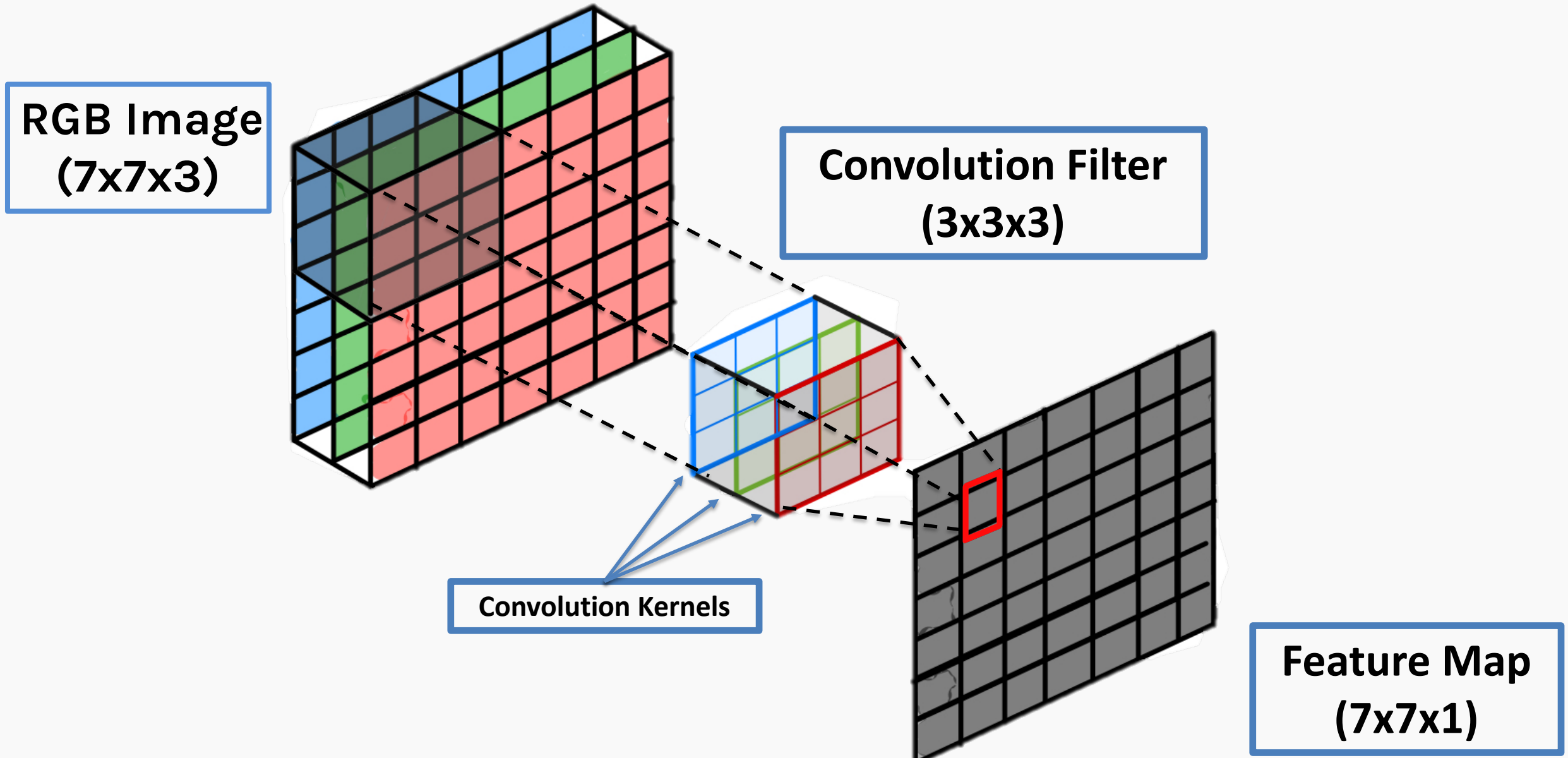
What are feature maps and why multiple feature maps?

What is Pooling?

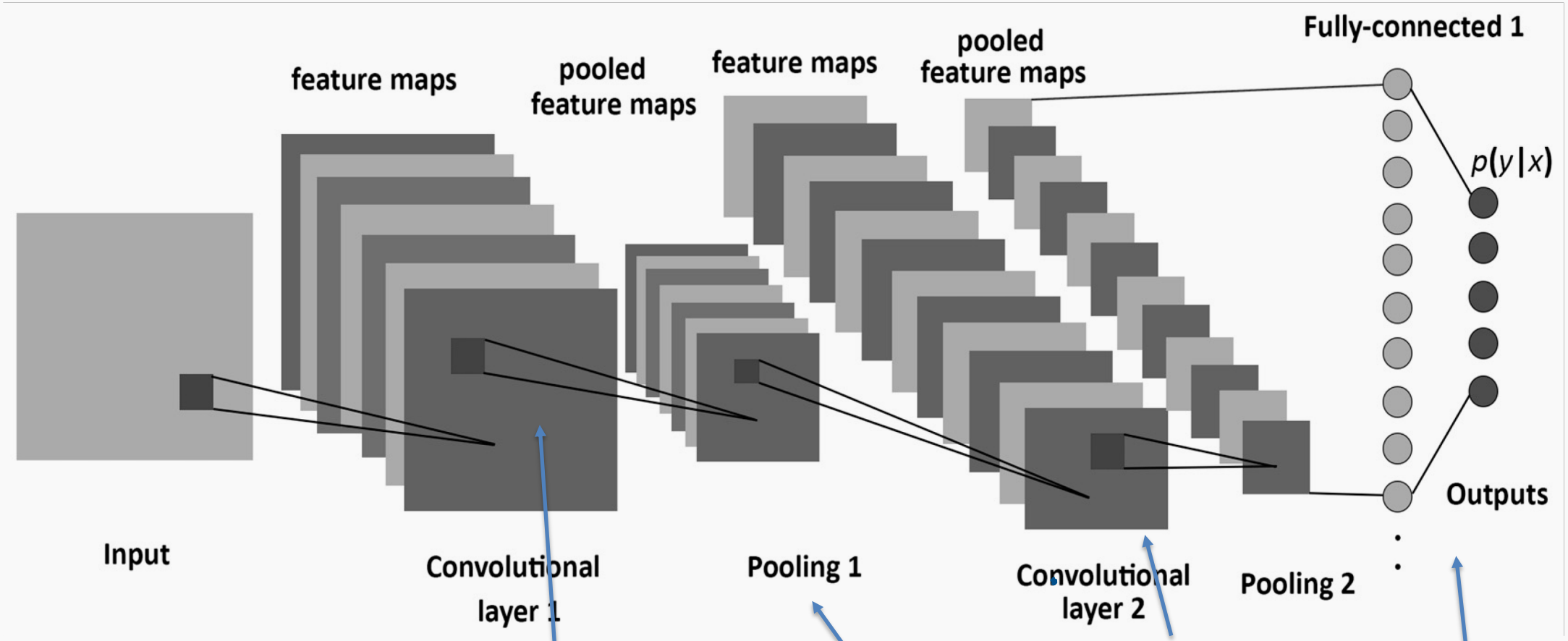
Why more than one layer?

What is going on here?

“Convolution” Operation



A Convolutional Network



~~What are feature maps~~ and why multiple feature maps?

What is Pooling?

Why more than one layer?

What is going on here?

Why more than one feature map?

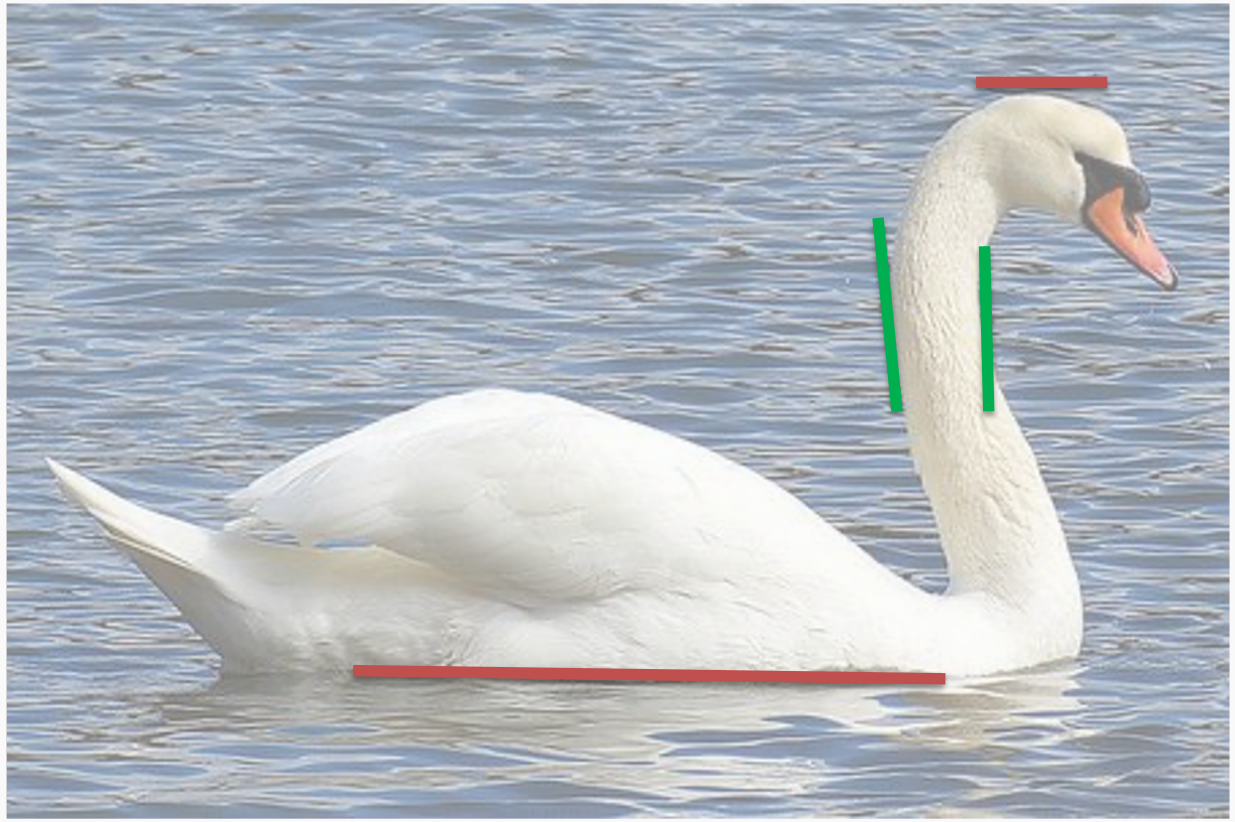
LAYER 1:

Why more than one feature map?

LAYER 1:

Filter 1: Horizontal Lines

Why more than one feature map?

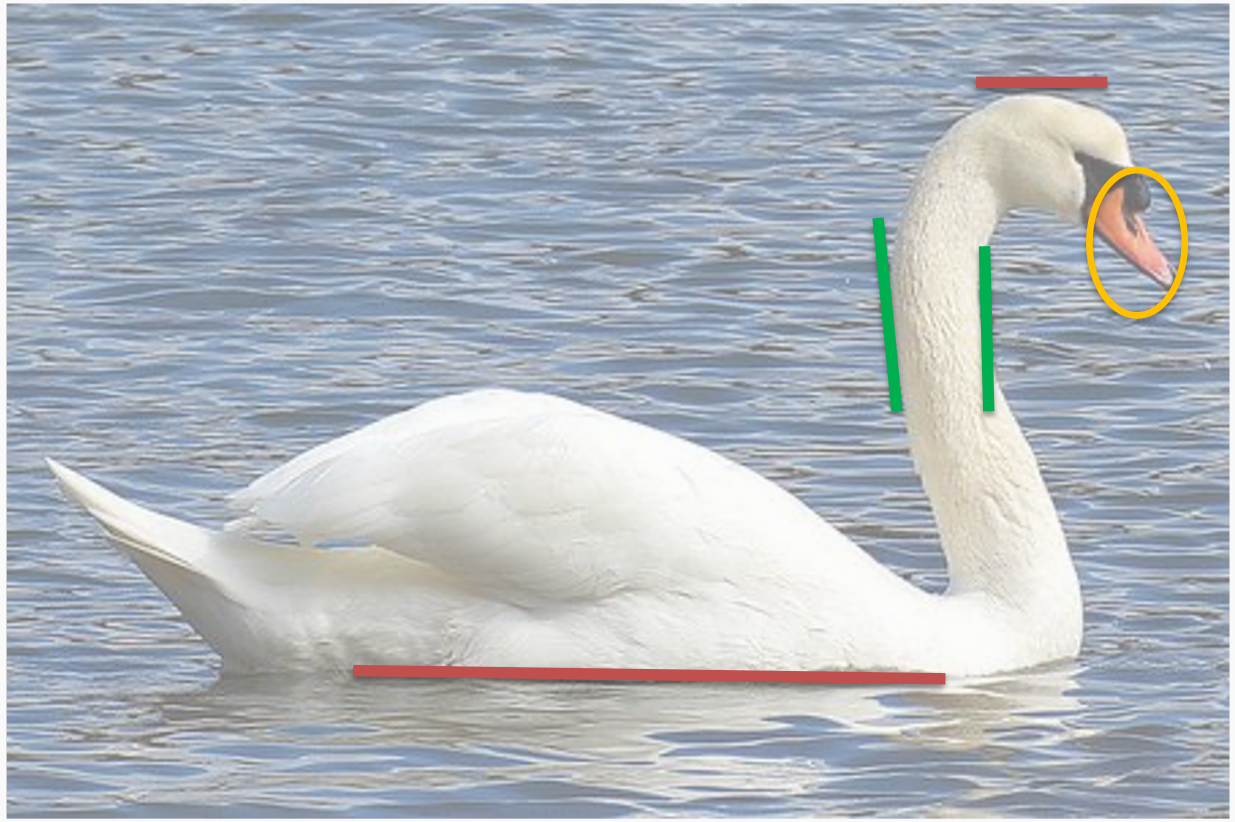


LAYER 1:

Filter 1: Horizontal Lines

Filter 2: Vertical Lines

Why more than one feature map?



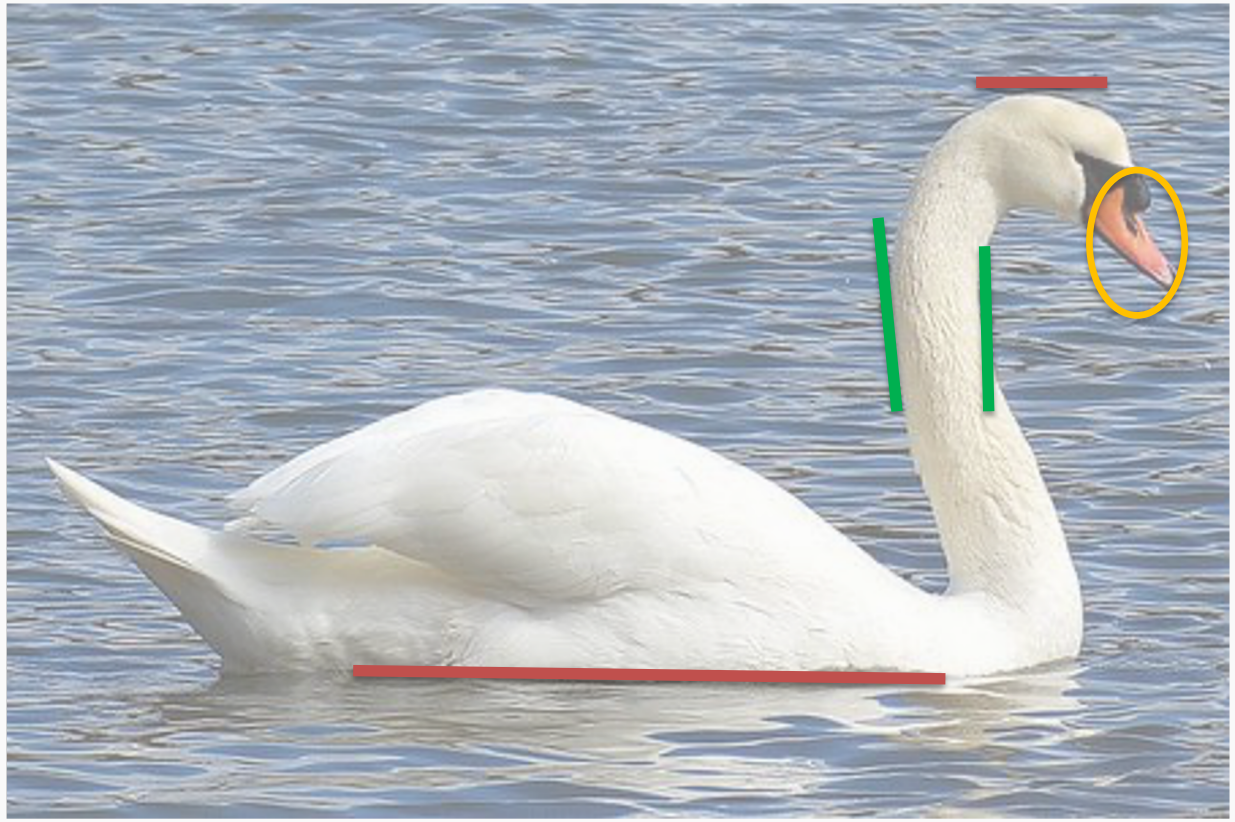
LAYER 1:

Filter 1: Horizontal Lines

Filter 2: Vertical Lines

Filter 3: Orange bulb

Why more than one feature map?



LAYER 1:

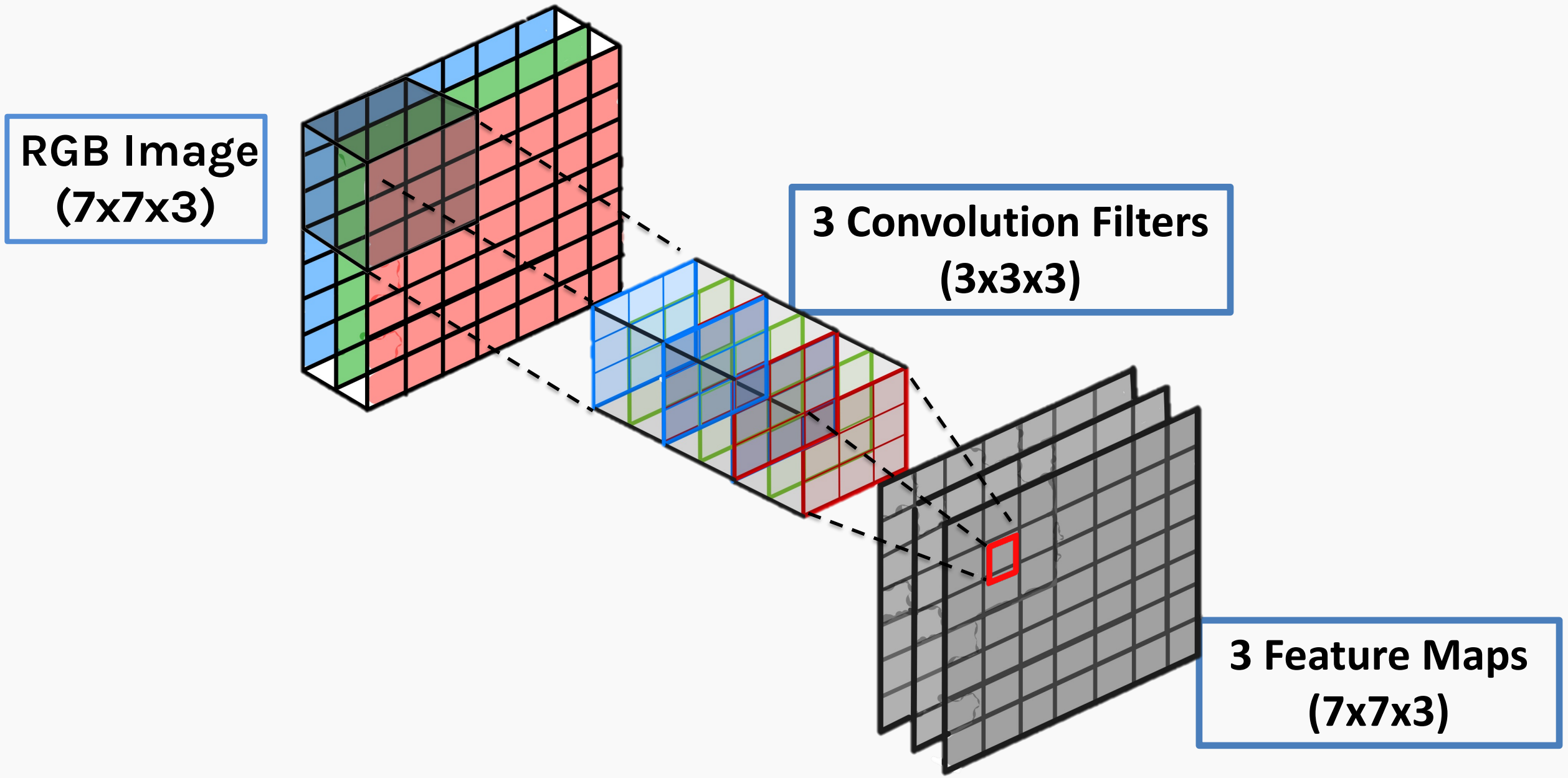
Filter 1: Horizontal Lines

Filter 2: Vertical Lines

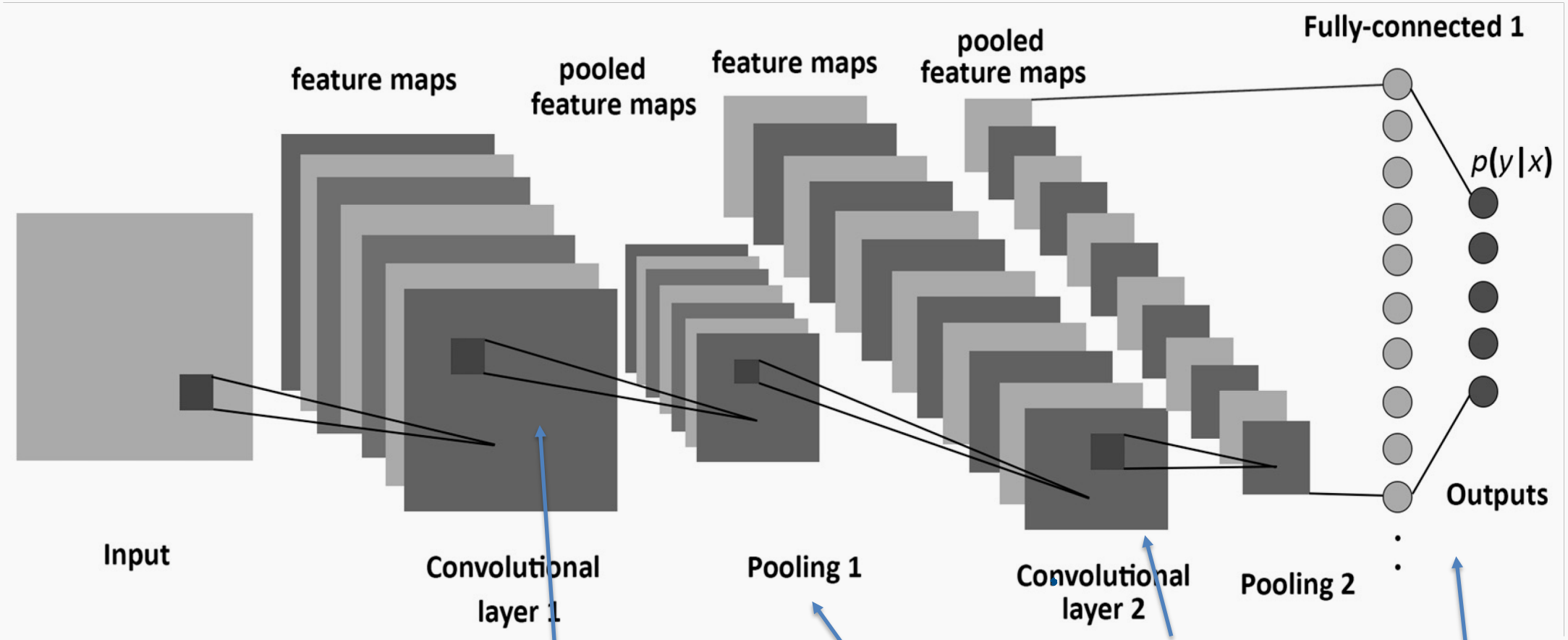
Filter 3: Orange bulb

Different filters identify different features.

“Convolution” Operation



A Convolutional Network



~~What are feature maps and why multiple feature maps?~~

What is Pooling?

Why more than one layer?

What is going on here?

Why more than one layer?

Why more than one layer?

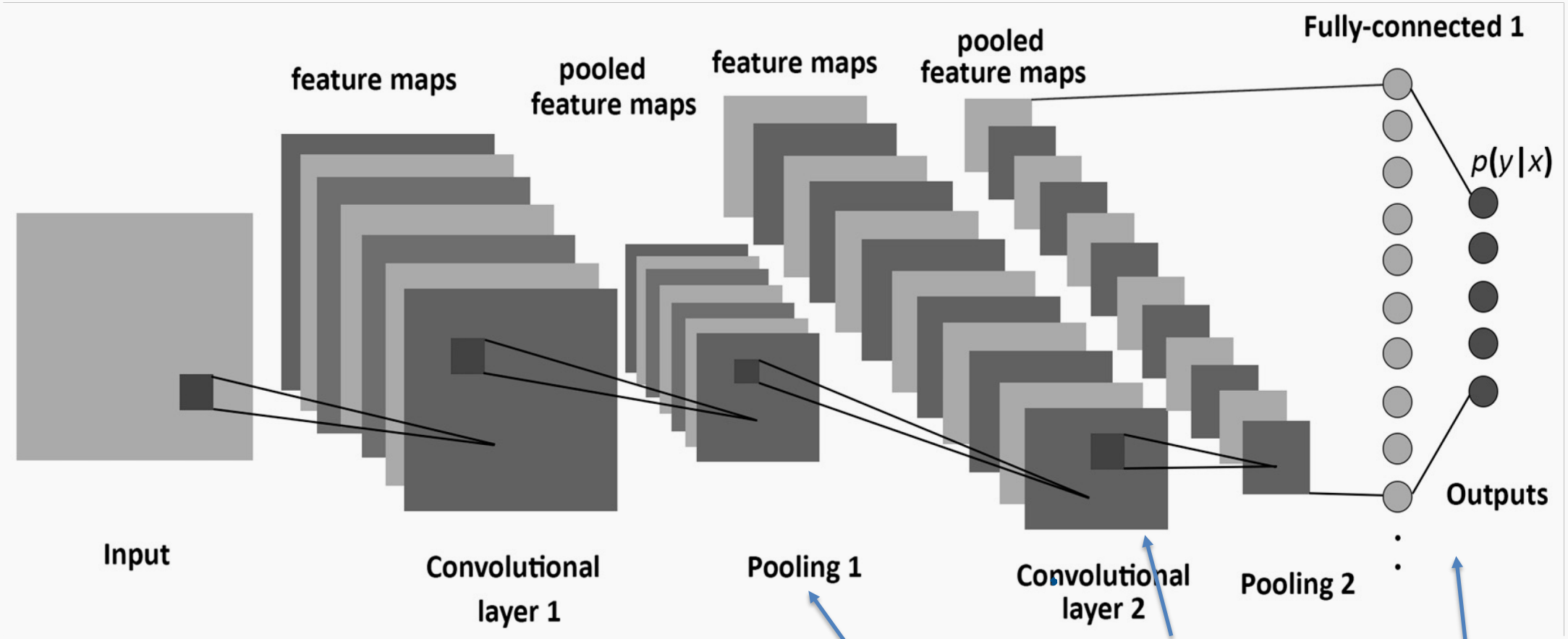
Layer 2, Filter 1: Combines horizontal and vertical lines from Layer 1 produce diagonal lines.

Why more than one layer?

Layer 2, Filter 1: Combines horizontal and vertical lines from Layer 1 produce diagonal lines.

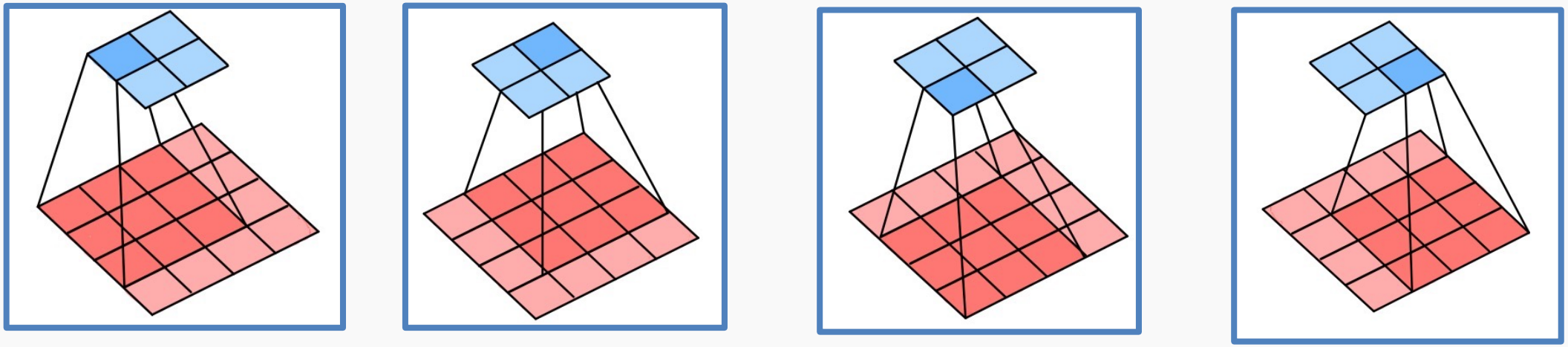
Layer 3, Filter 1: Combines diagonal lines to identify shapes

A Convolutional Network



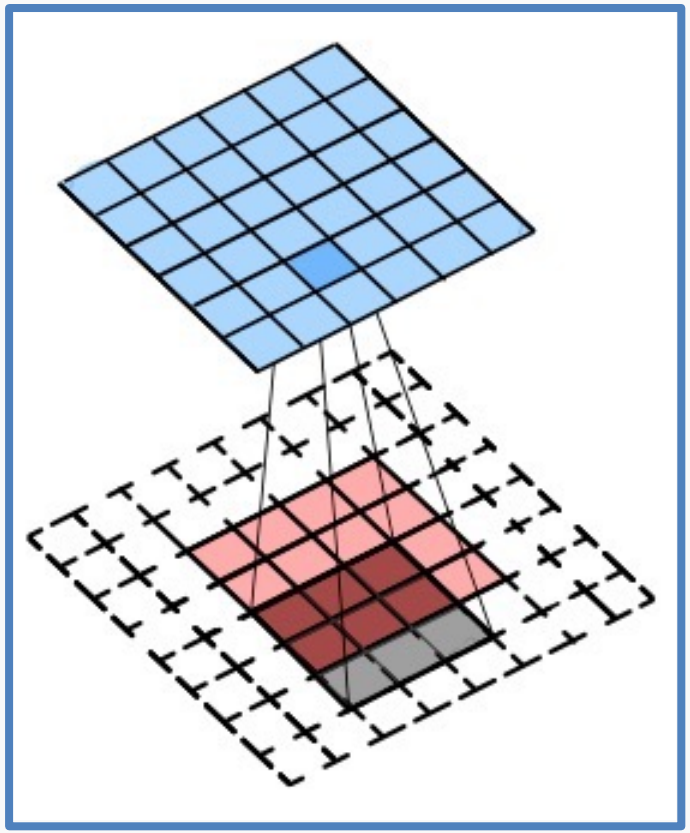
Convolutions – what happens at the edges?

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

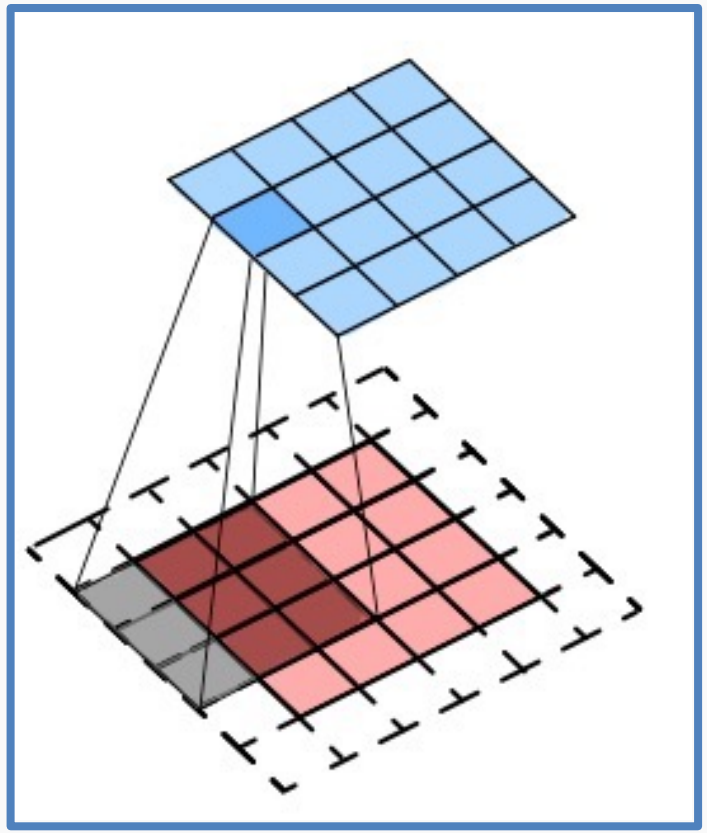


We can avoid this by padding the edges in different ways.

Padding



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



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

Stride

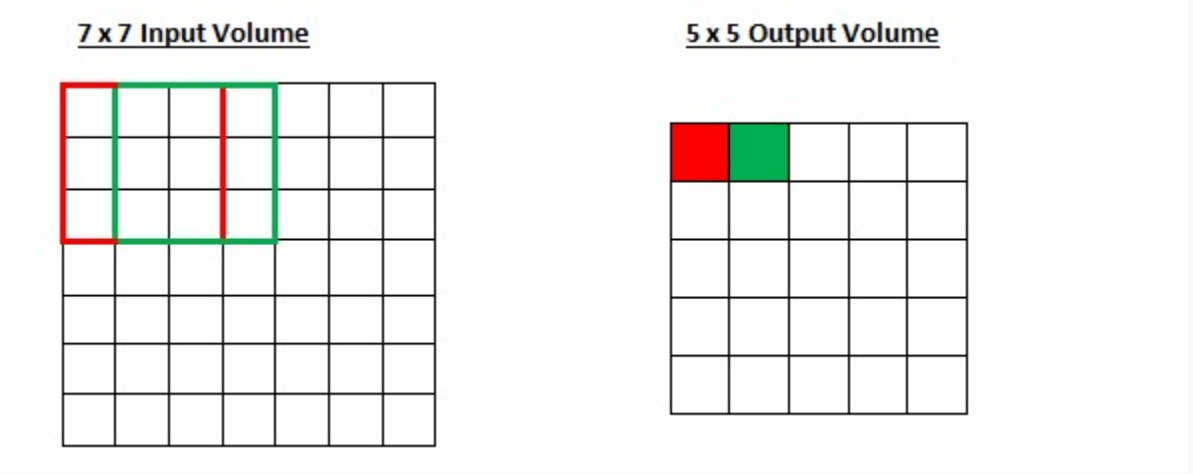
Stride controls how the filter convolves around the input volume.

The formula for calculating the output size is:

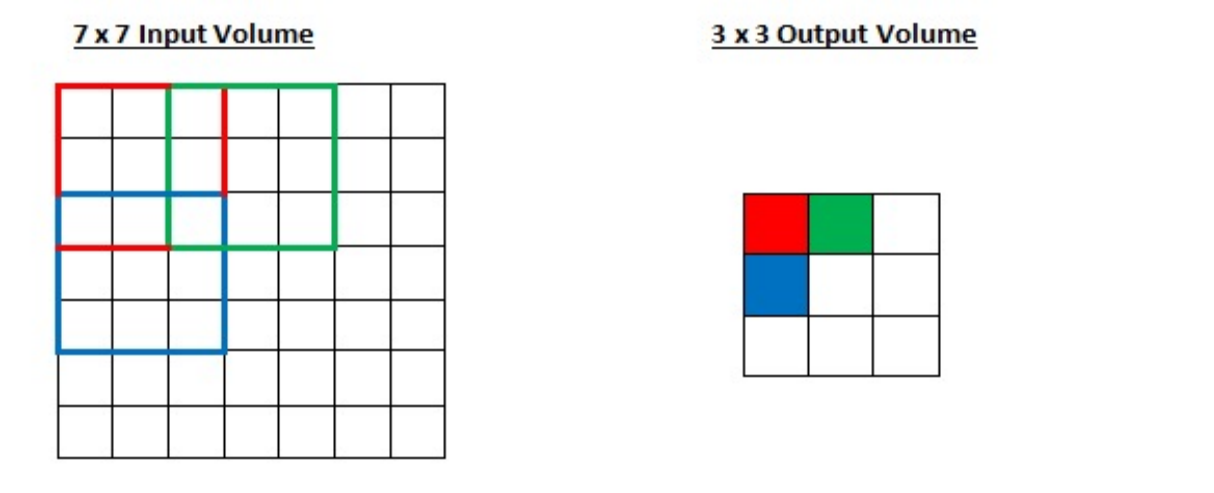
$$O = \frac{W - K + 2P}{S} + 1$$

Where O is output dim, W is the input dim, K is the filter size, P is padding and S the stride

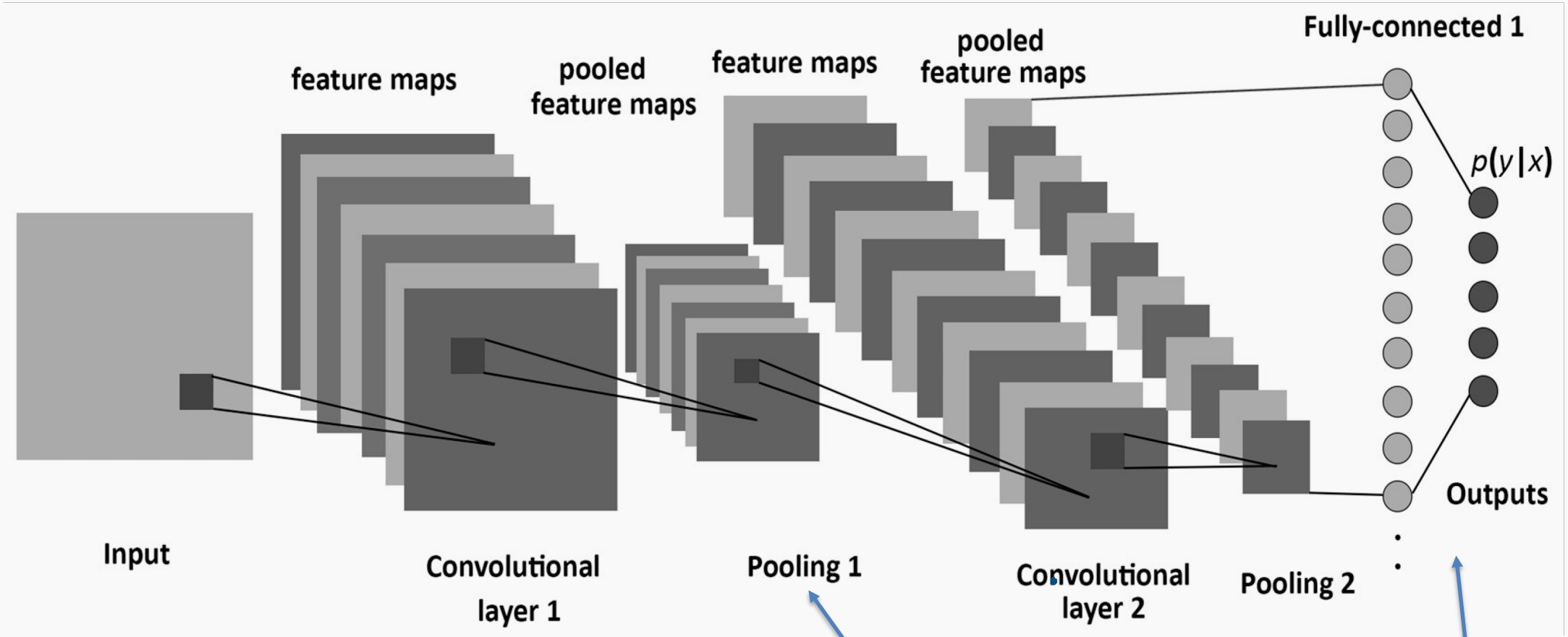
Stride = 1



Stride = 2



A Convolutional Network



Training CNN

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

But most importantly, we are learning those filters!

One thing we're missing: non-linearity.

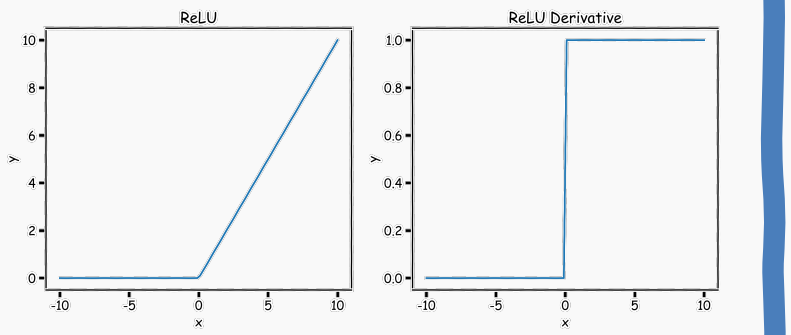
HOW? We use BackProp and SGD as we did with FCNN

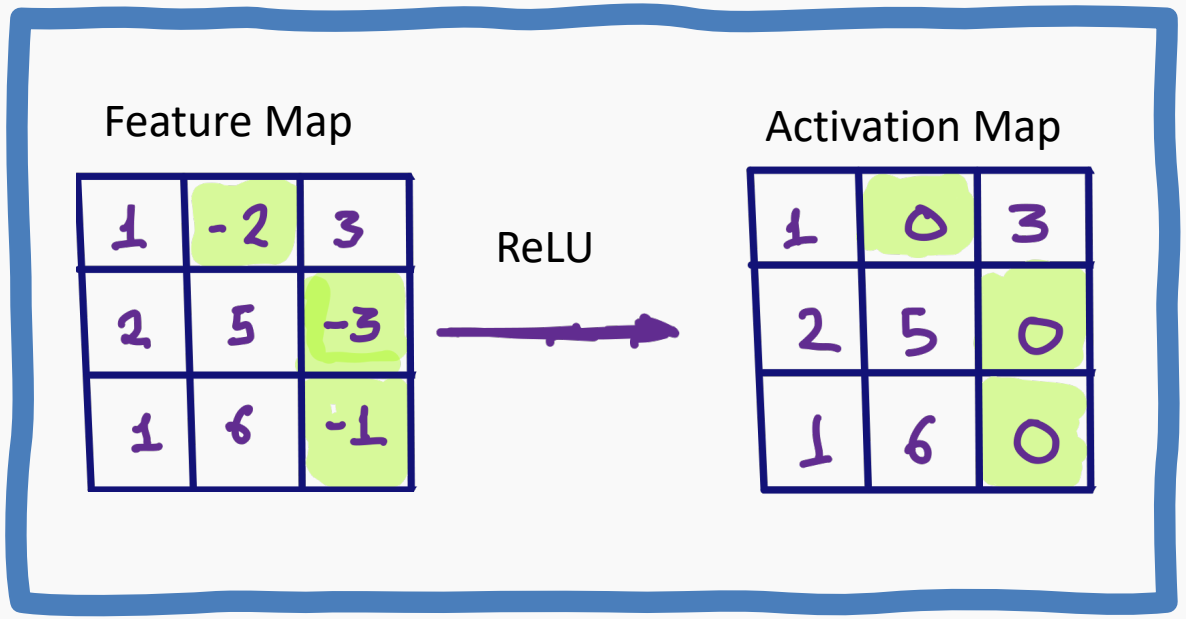
ReLU

We apply non-linear activation **after** convolution as we did for FCNN.

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

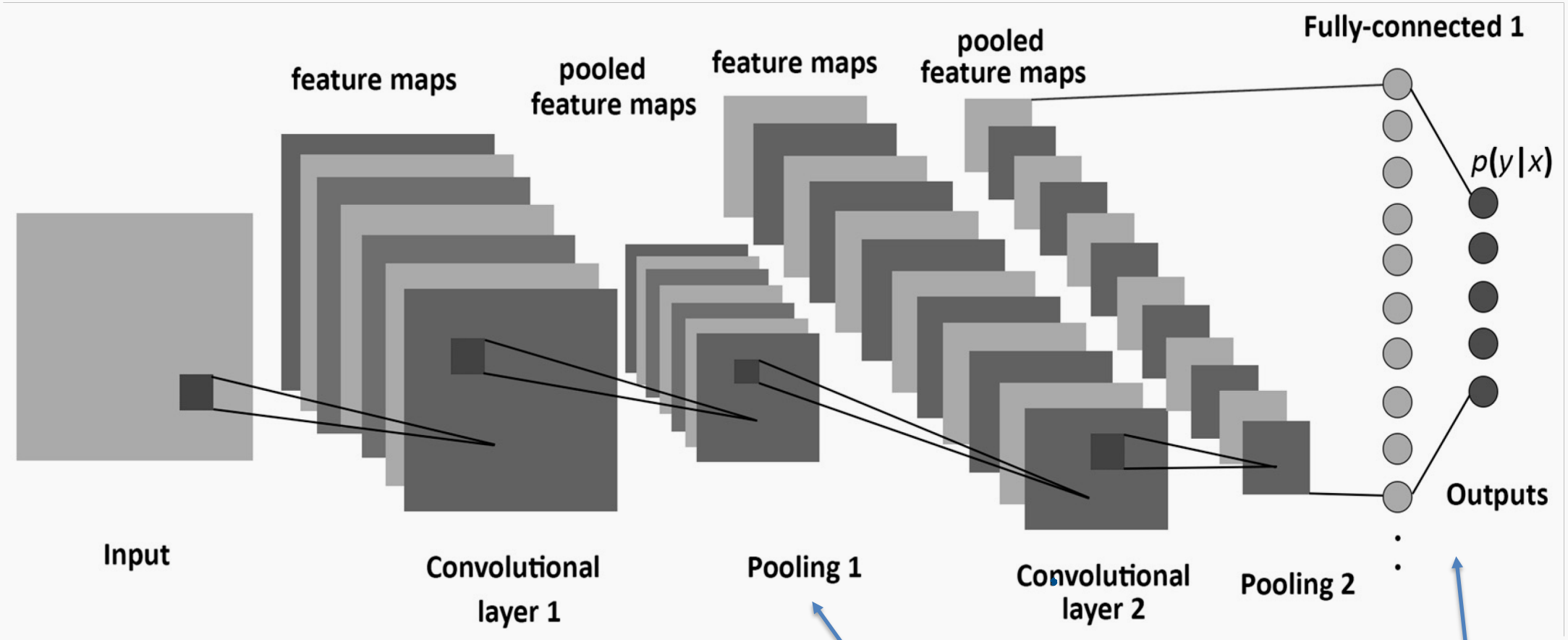
$$y = \max(0, x)$$





This combats the vanishing gradient problem occurring in sigmoid, it is easier to compute, and generates sparsity.

A Convolutional Network

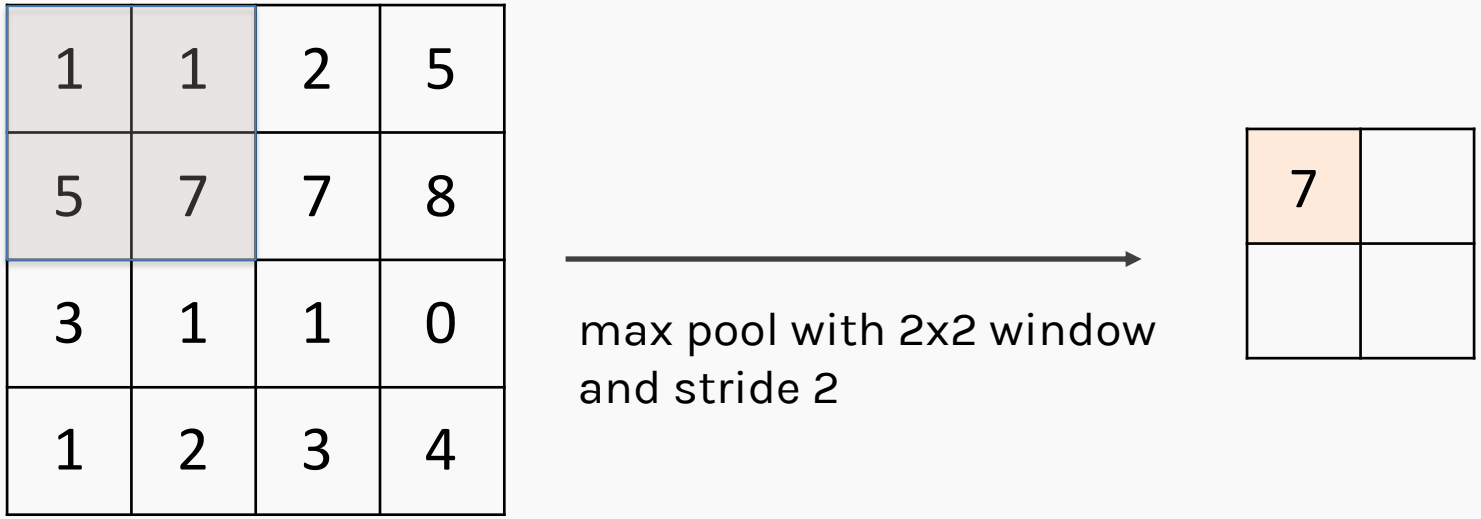


Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps*.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:



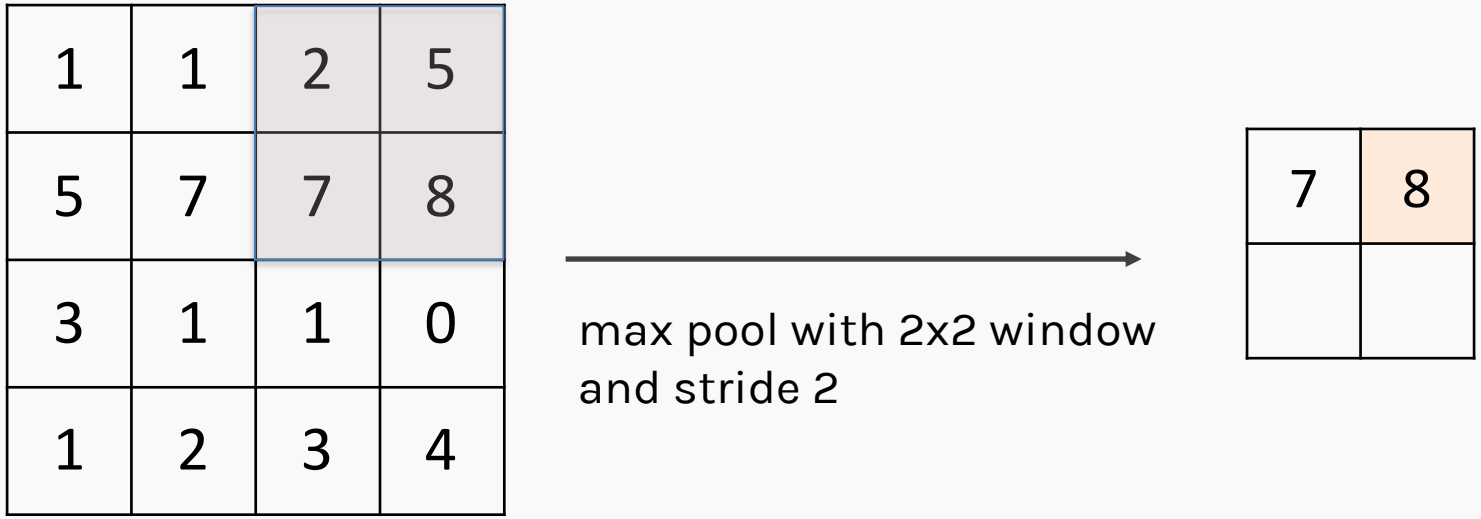
* Maxpooling could be applied before ReLU.

Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:

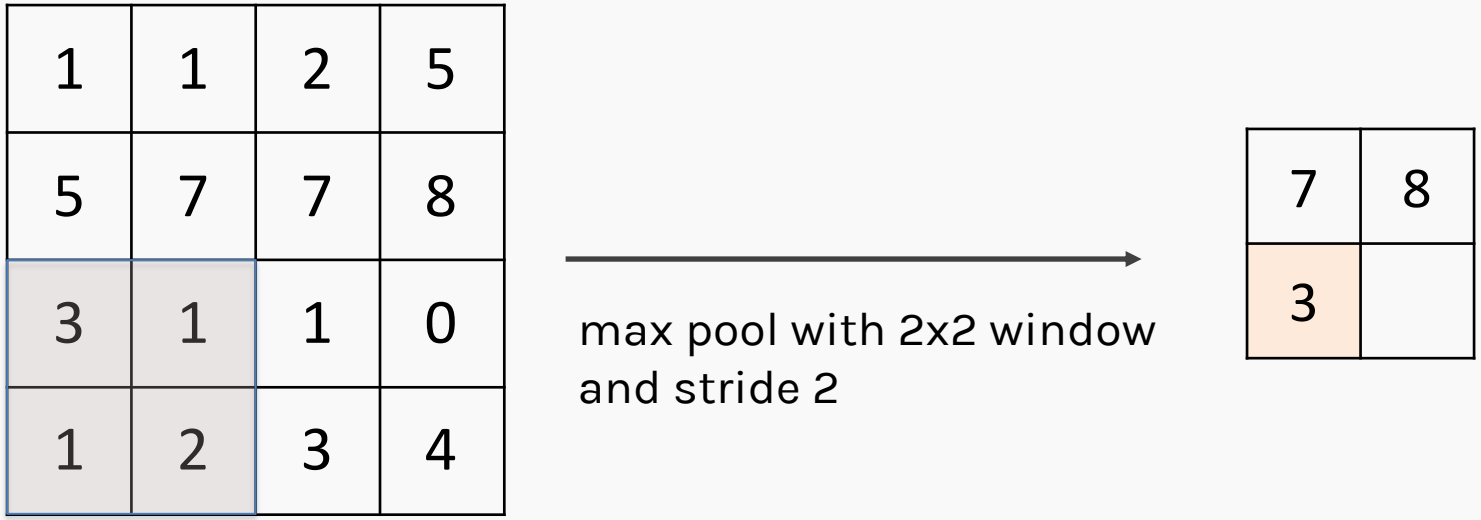


Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:

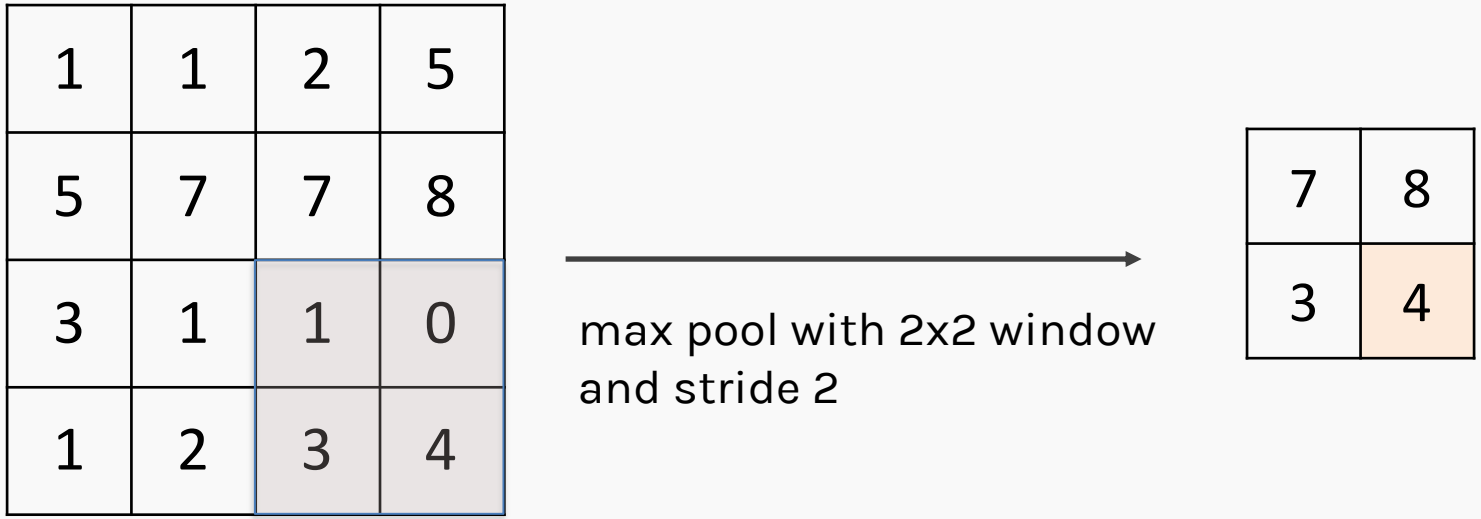


Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:



Pooling

A **pooling** layer is a **new layer** added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps*.

The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.

Pooling involves selecting:

- A pooling operation, much like a filter, to be applied to feature maps: e.g. max, mean, median.
- The size of the pooling operation.
- The stride.

* Maxpooling could be applied before ReLU.

Pooling

Pooling involves selecting:

- A pooling operation, much like a filter, to be applied to feature maps: e.g. max, mean, median.
- The size of the pooling operation.
- The stride.

The size of the pooling operator must be smaller than the size of the feature map; specifically, it is almost always 2×2 applied with a stride of 2 using **max** pooling.

Pooling

Pooling involves selecting:

- A pooling operation, much like a filter, to be applied to feature maps: e.g. max, mean, median.
- The size of the pooling operation.
- The stride.

The size of the pooling operator must be smaller than the size of the feature map; specifically, it is almost always 2×2 applied with a stride of 2 using **max** pooling.

Invariant to small, "local transitions"

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

Reduces input size of the final fully connected layers (more later)

No learnable parameters

Pooling: more examples with stride 2x2

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

max pool with 2x2 window and stride 2x2

→

7	8
3	4

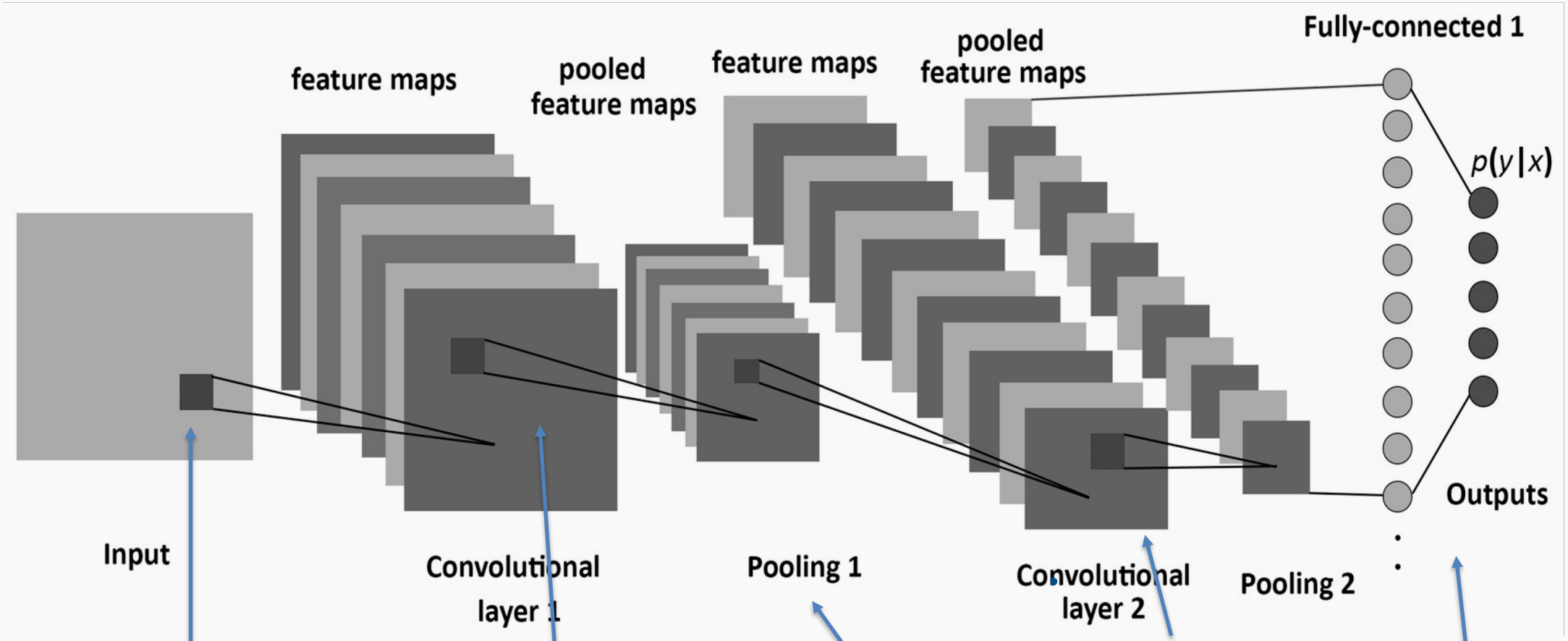
1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

mean pool with 2x2 window and stride 2x2

→

3.5	5.5
1.75	2

A Convolutional Network



What is this square?

What are feature maps and why are there multiple feature maps?

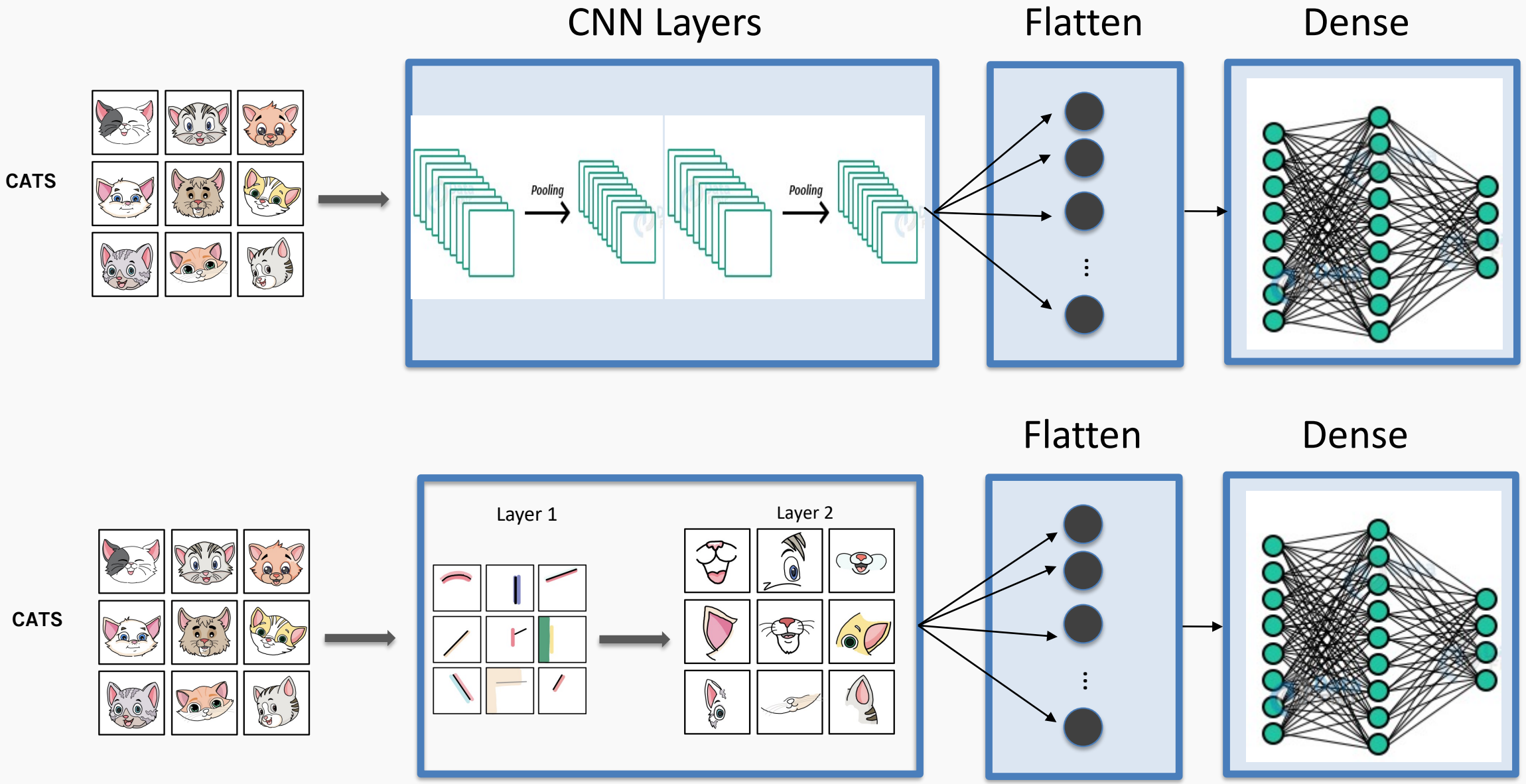
What is Pooling?

Why is there more than one layer?

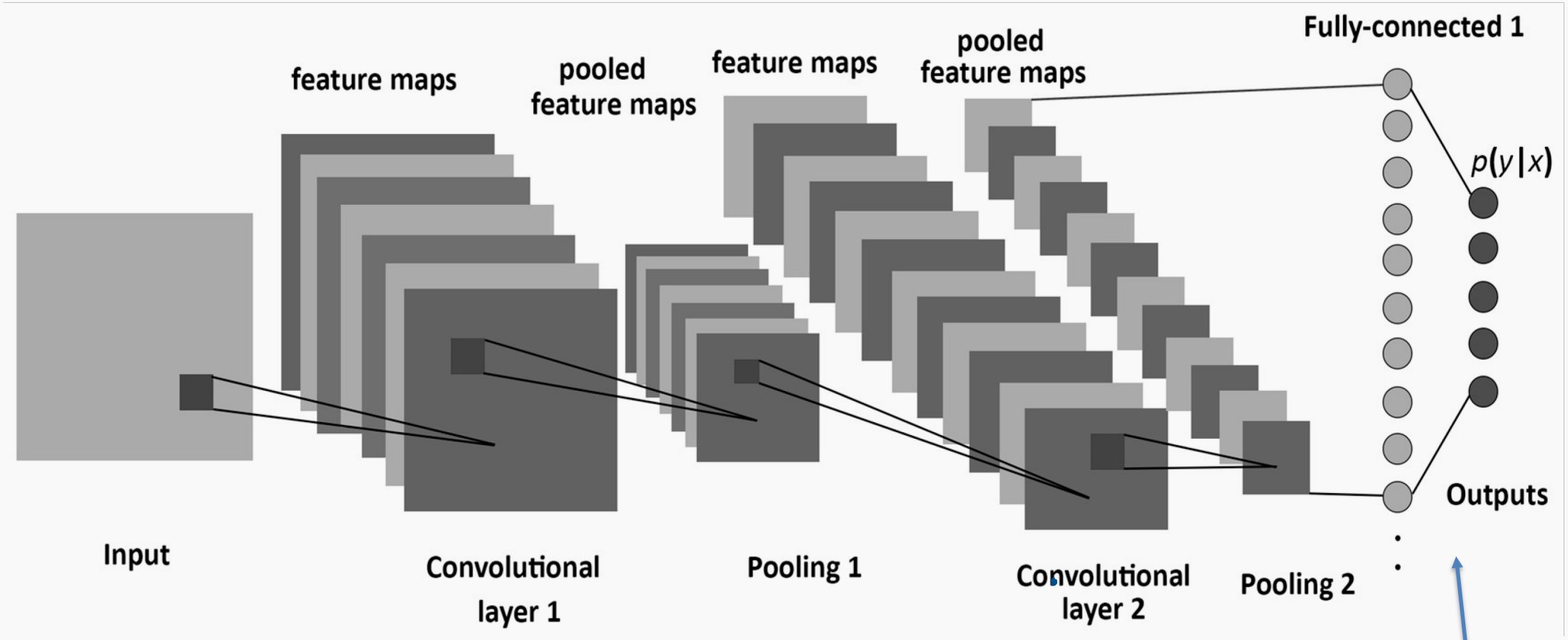
What is going on here?

What do CNN layers learn?

- Each CNN layer learns features 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.


CNN Layers
Flatten
Dense
CATS
Pooling
Pooling
Flatten
Dense
CATS
Layer 1
Layer 2

A Convolutional Network



~~What is going on here?~~

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

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

I/O

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

Building a CNN

Pooling Layers

Action

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

Parameters

- Stride
- Size of window

I/O

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

Building a CNN

Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification, regression, segmentation, etc

Parameters

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

I/O

- Input: **FLATTENED** previous set of feature maps
- Output: Probabilities for each class or simply prediction for regression $\hat{y}$

A Convolutional Network