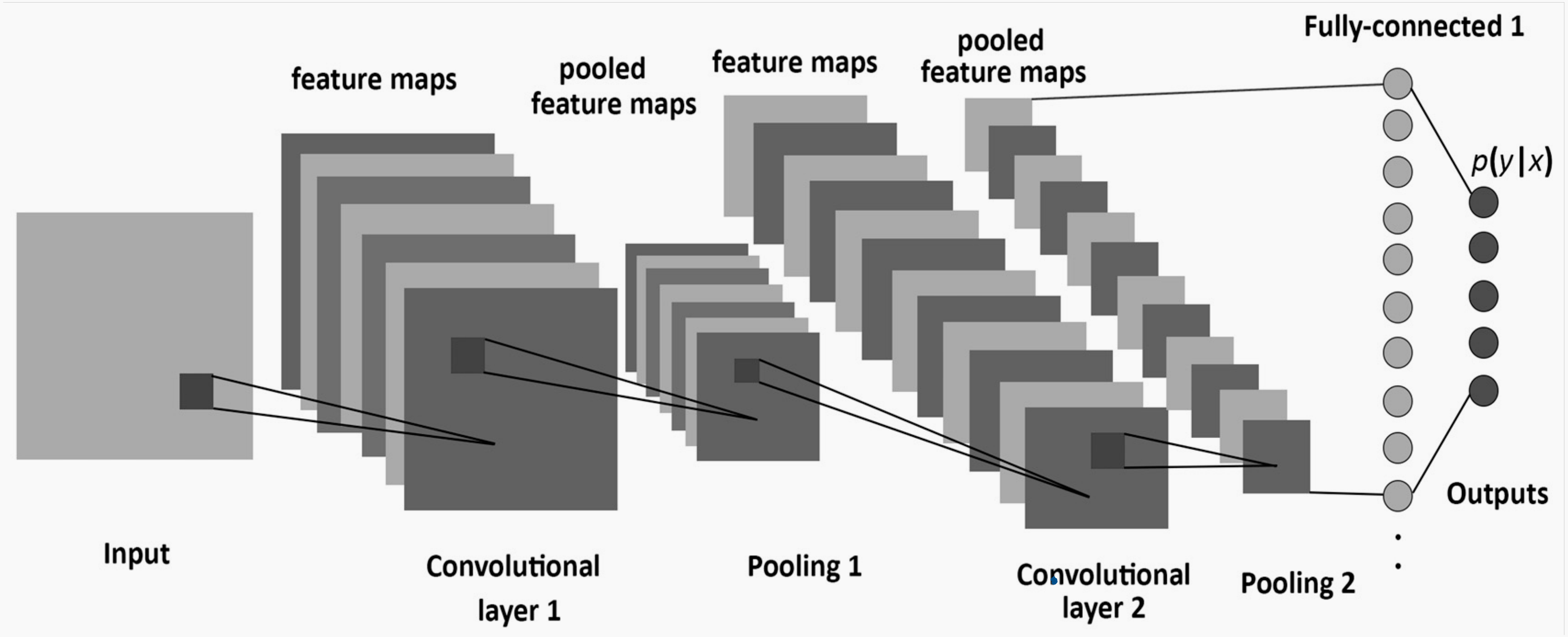
Convolutional Neural Networks I

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman

A Convolutional Network



+ ReLU

+ ReLU

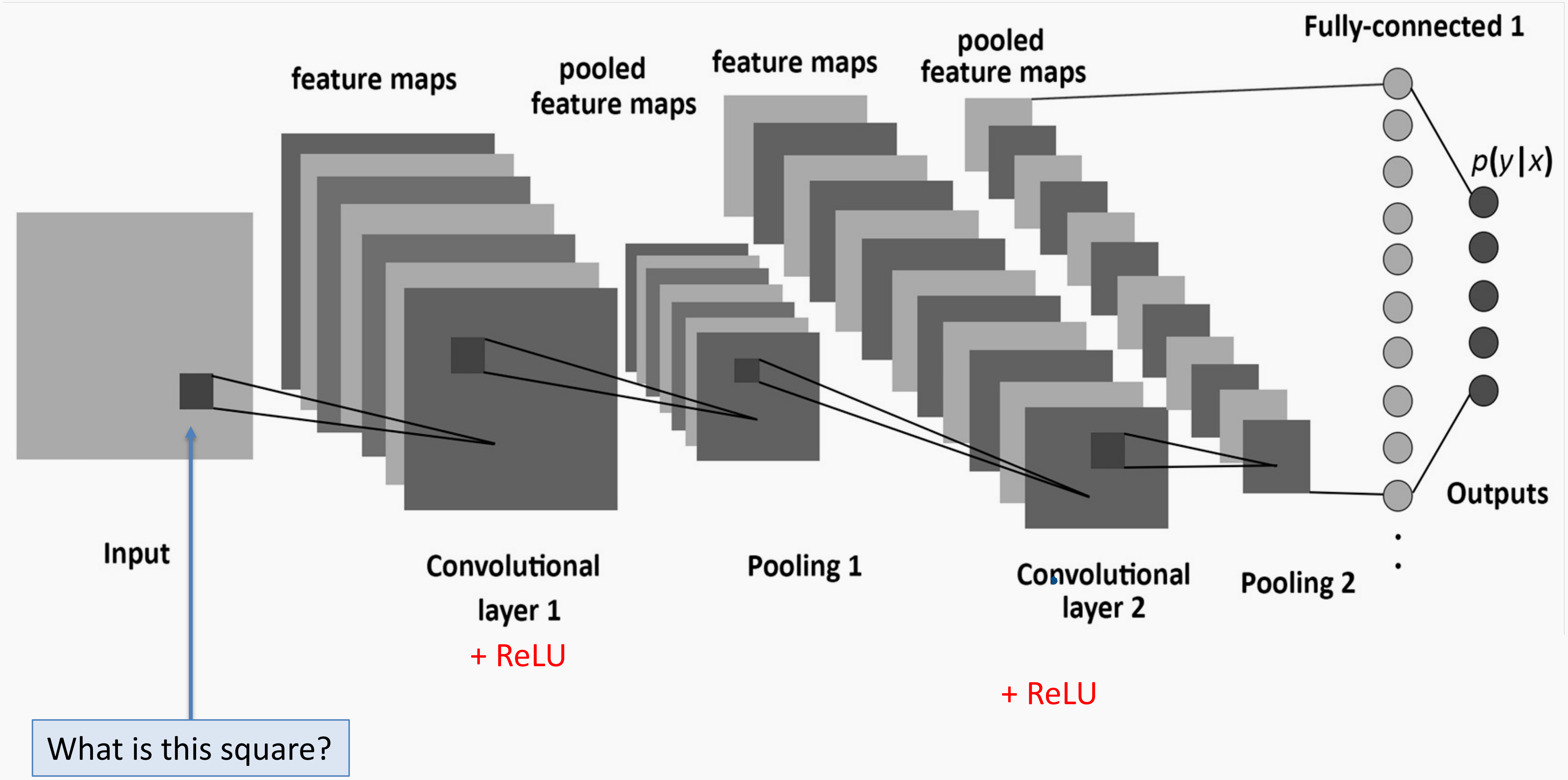
The code

In [ ]:

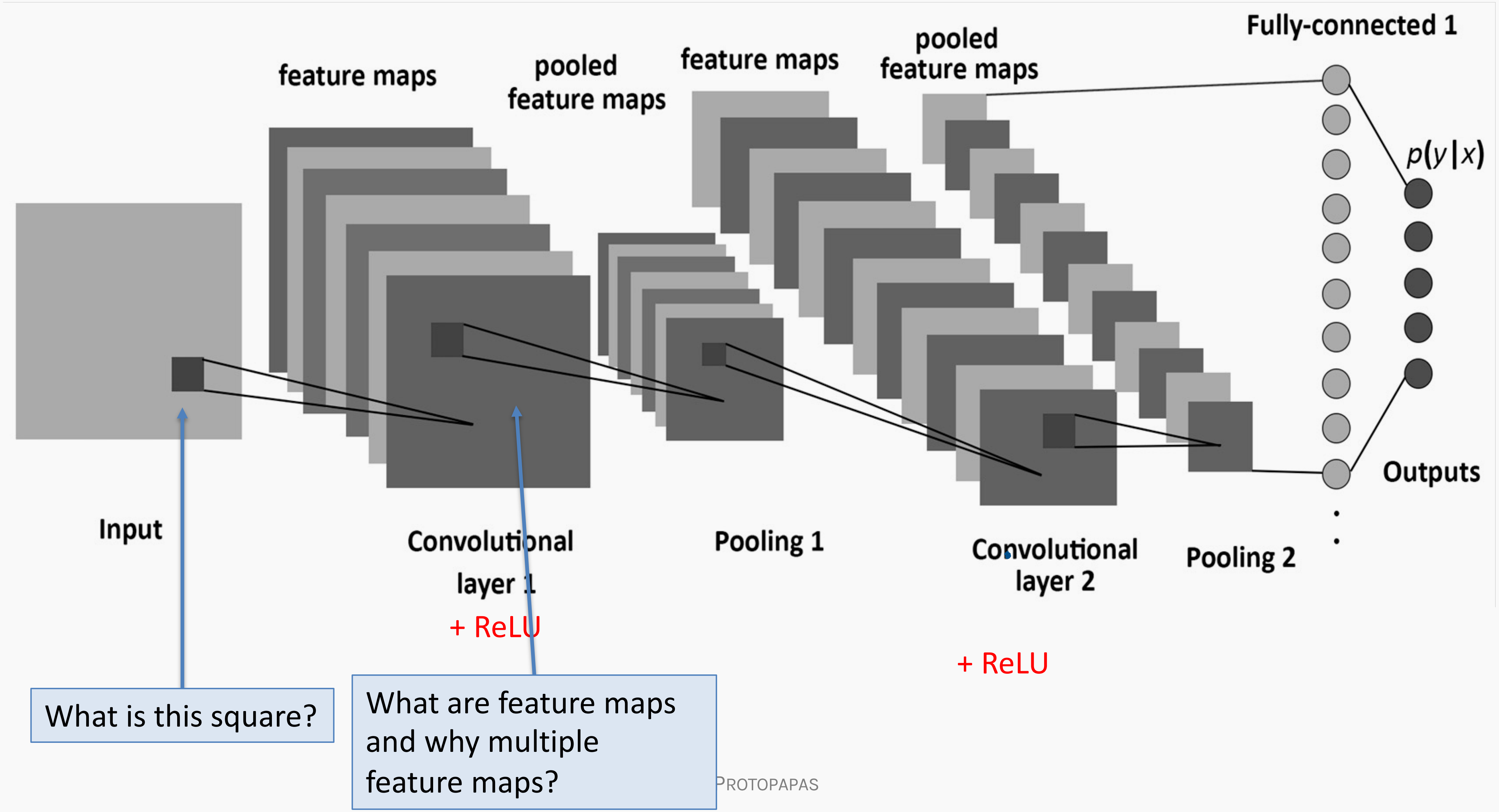
```python

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

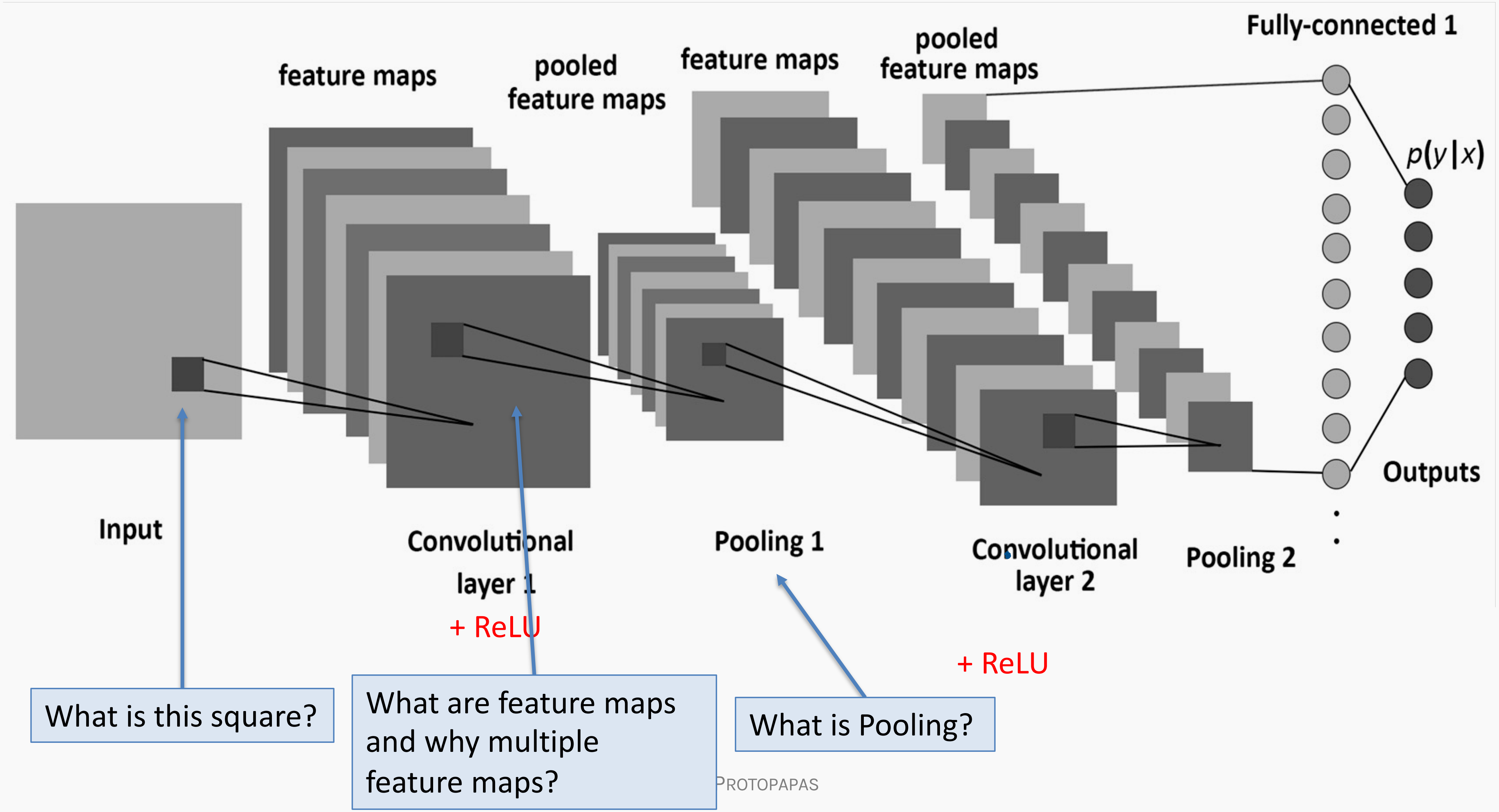
A Convolutional Network



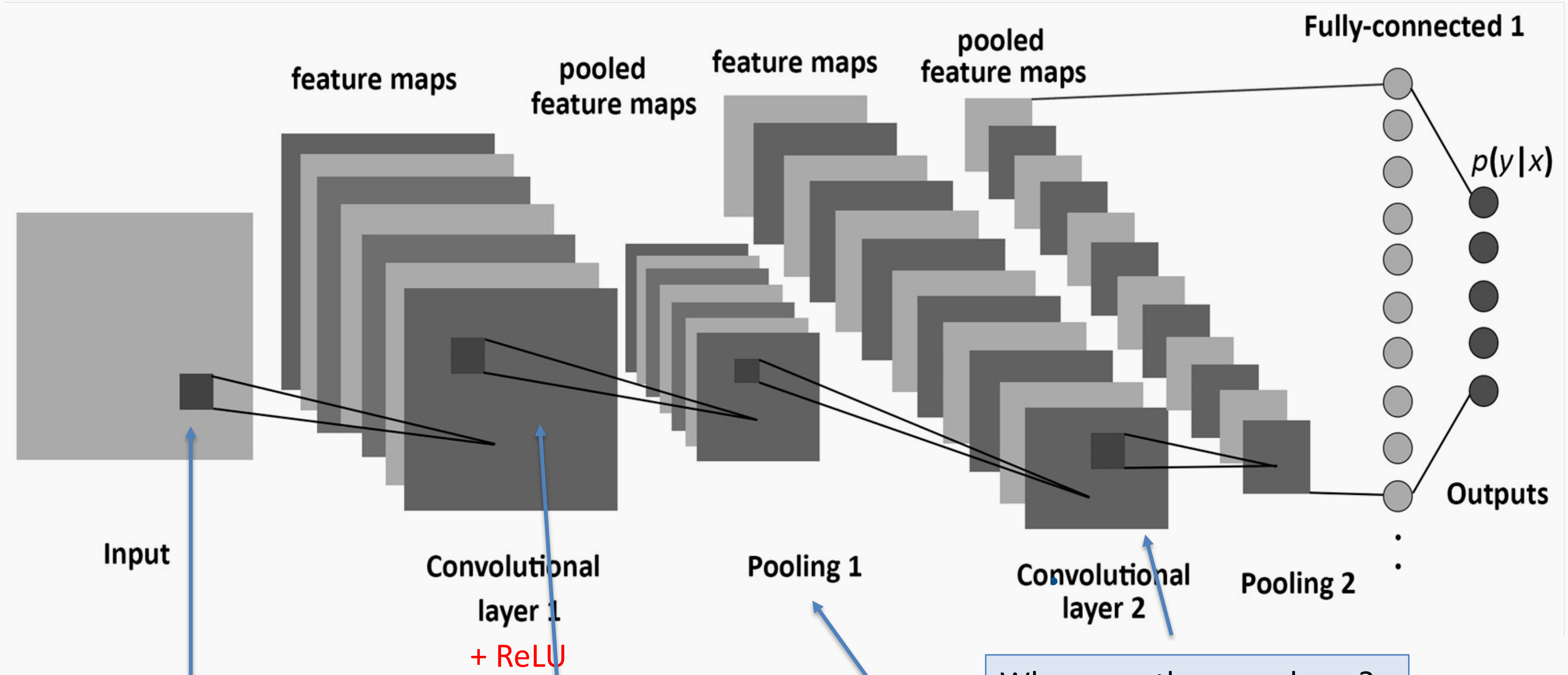
A Convolutional Network



A Convolutional Network



A Convolutional Network



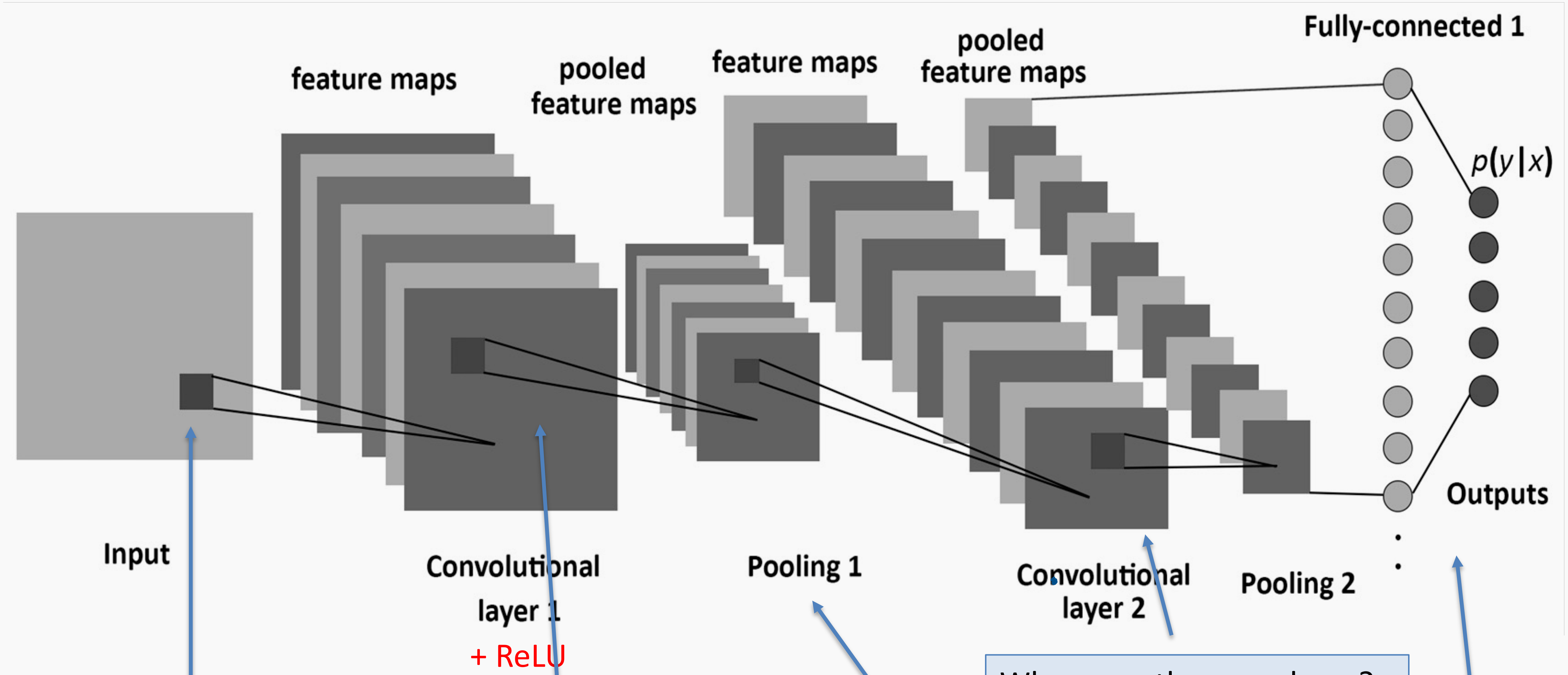
What is this square?

What are feature maps and why multiple feature maps?

What is Pooling?

Why more than one layer?

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?

Outline

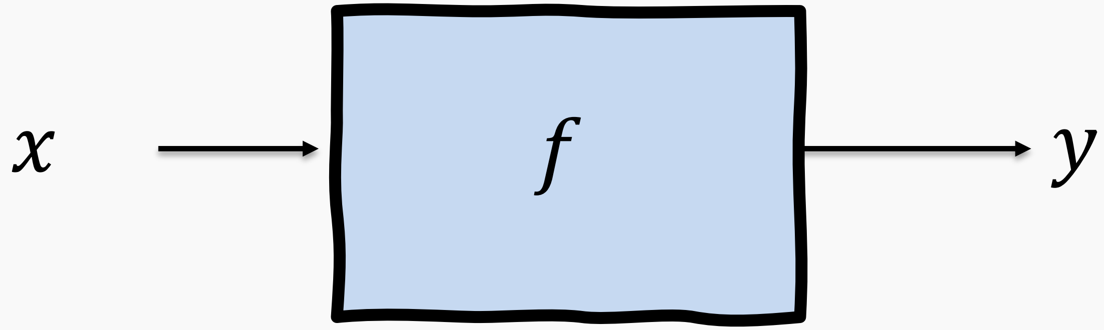
1. Motivation
2. CNN basic ideas
3. Building a CNN

Outline

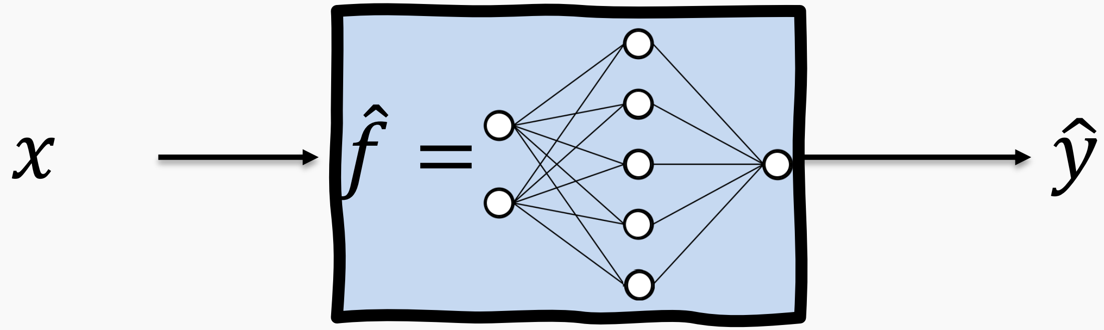
1. **Motivation**
2. CNN basic ideas
3. Building a CNN

Feed forward Neural Network, Multilayer Perceptron (MLP)

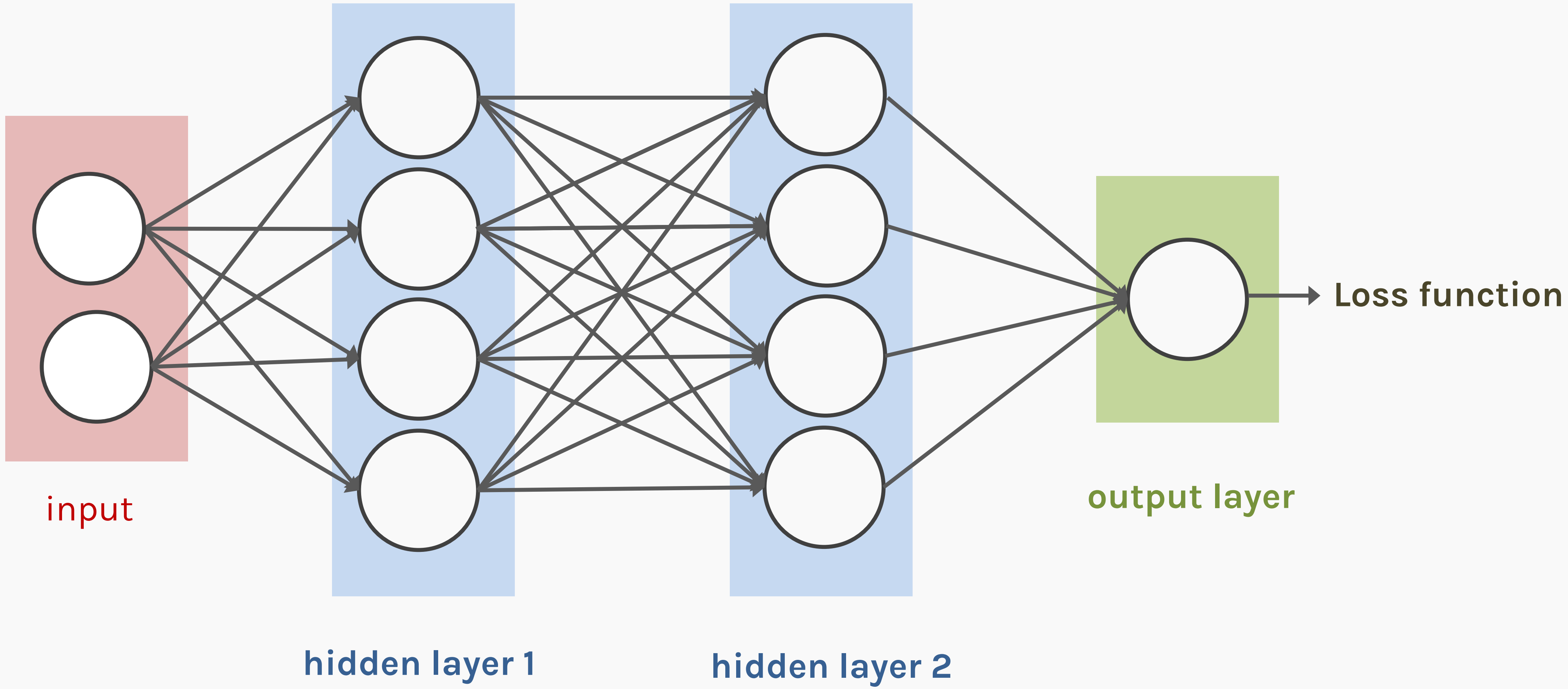
A **function** is a relation that associates each element x of a set X to a single element y of a set Y



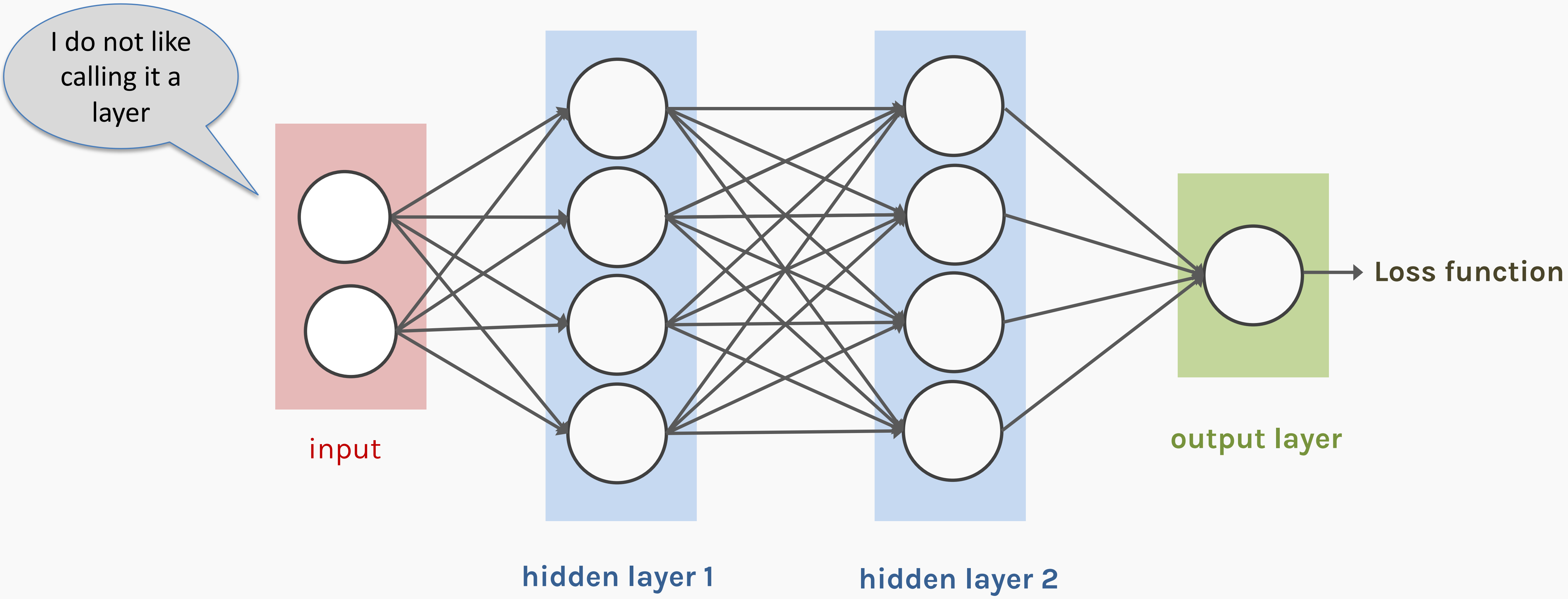
Neural networks can approximate a wide variety of functions



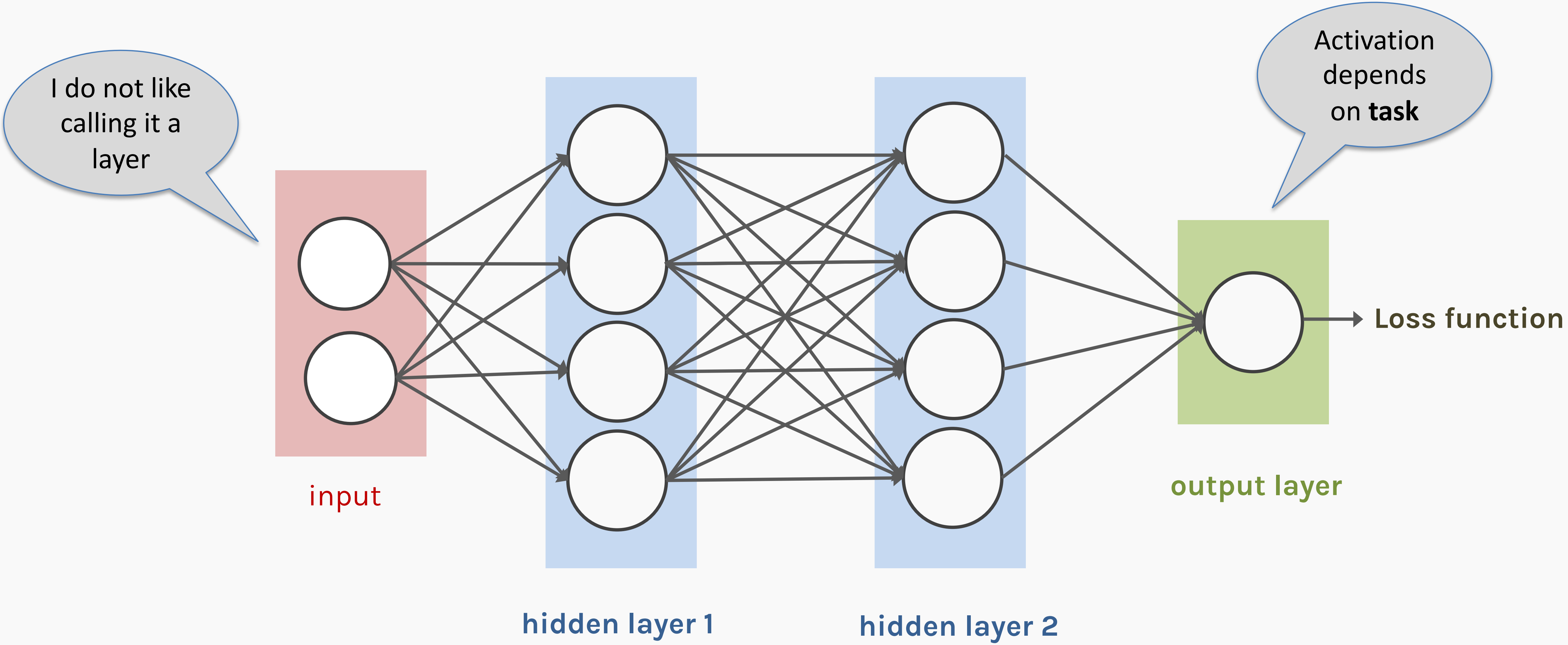
Quick review of MLPs



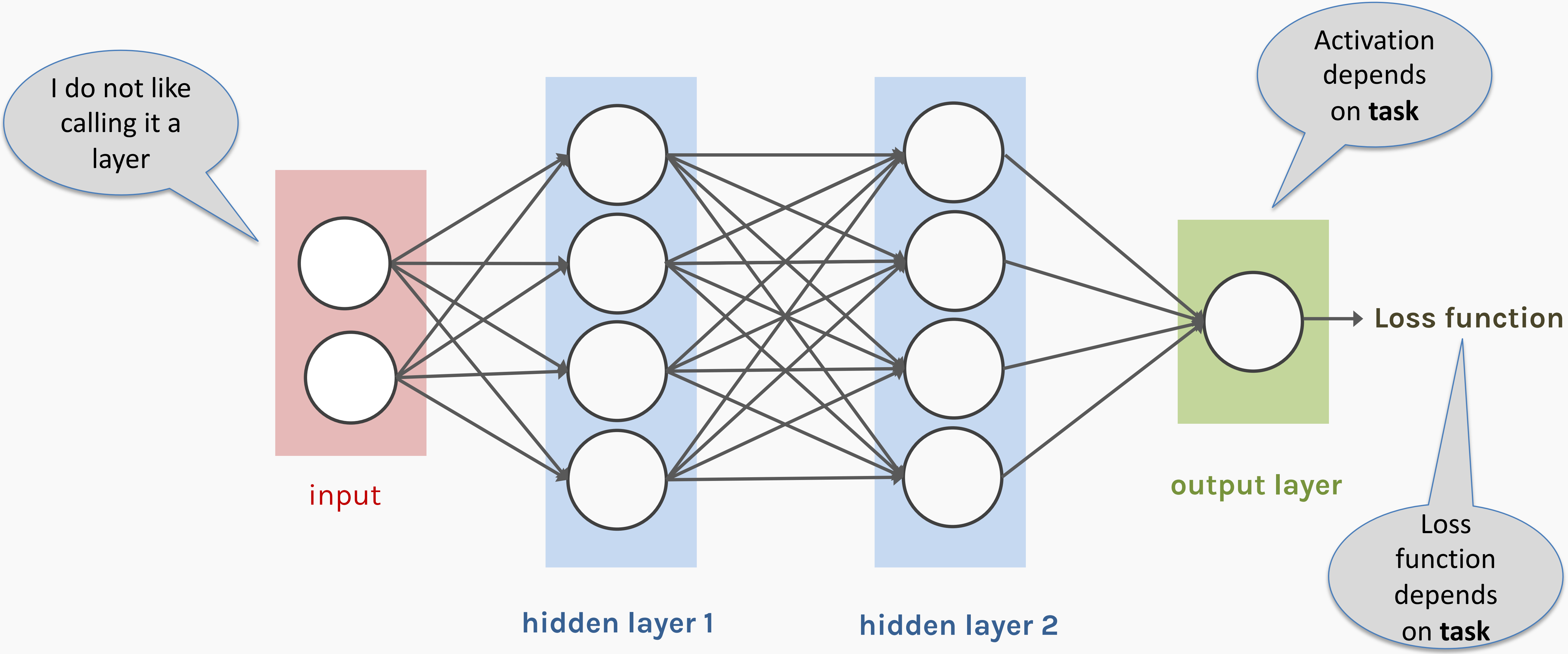
Quick review of MLPs



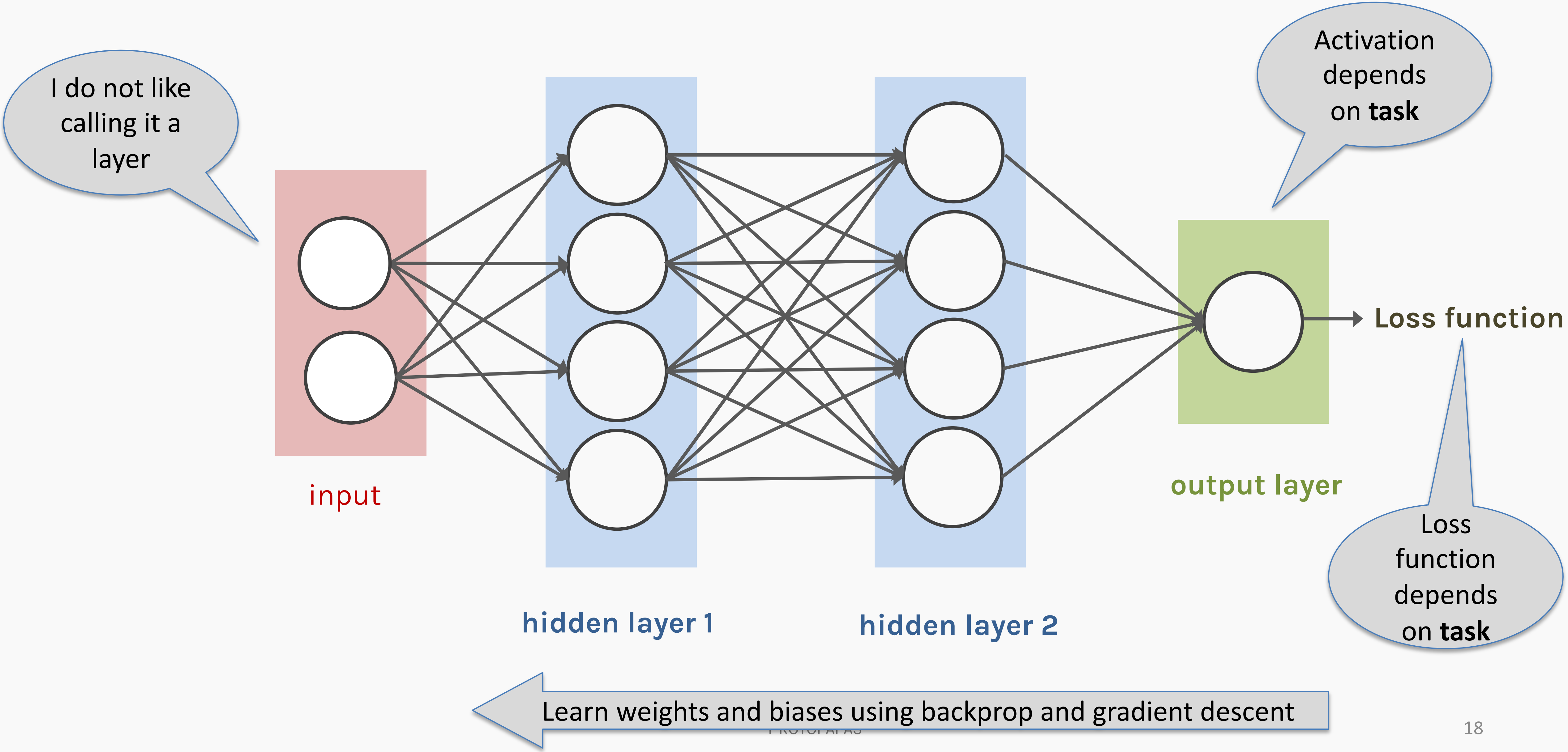
Quick review of MLPs



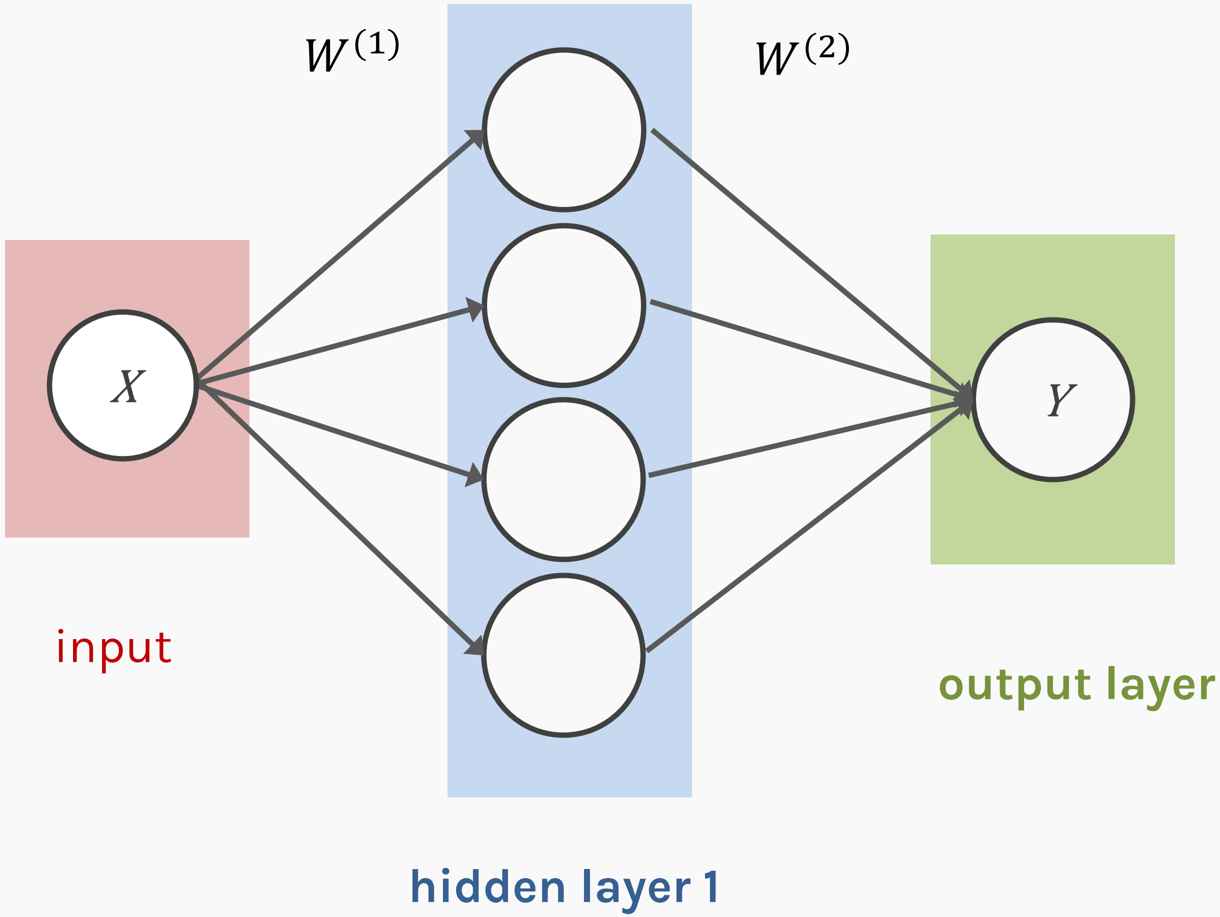
Quick review of MLPs



Quick review of MLPs



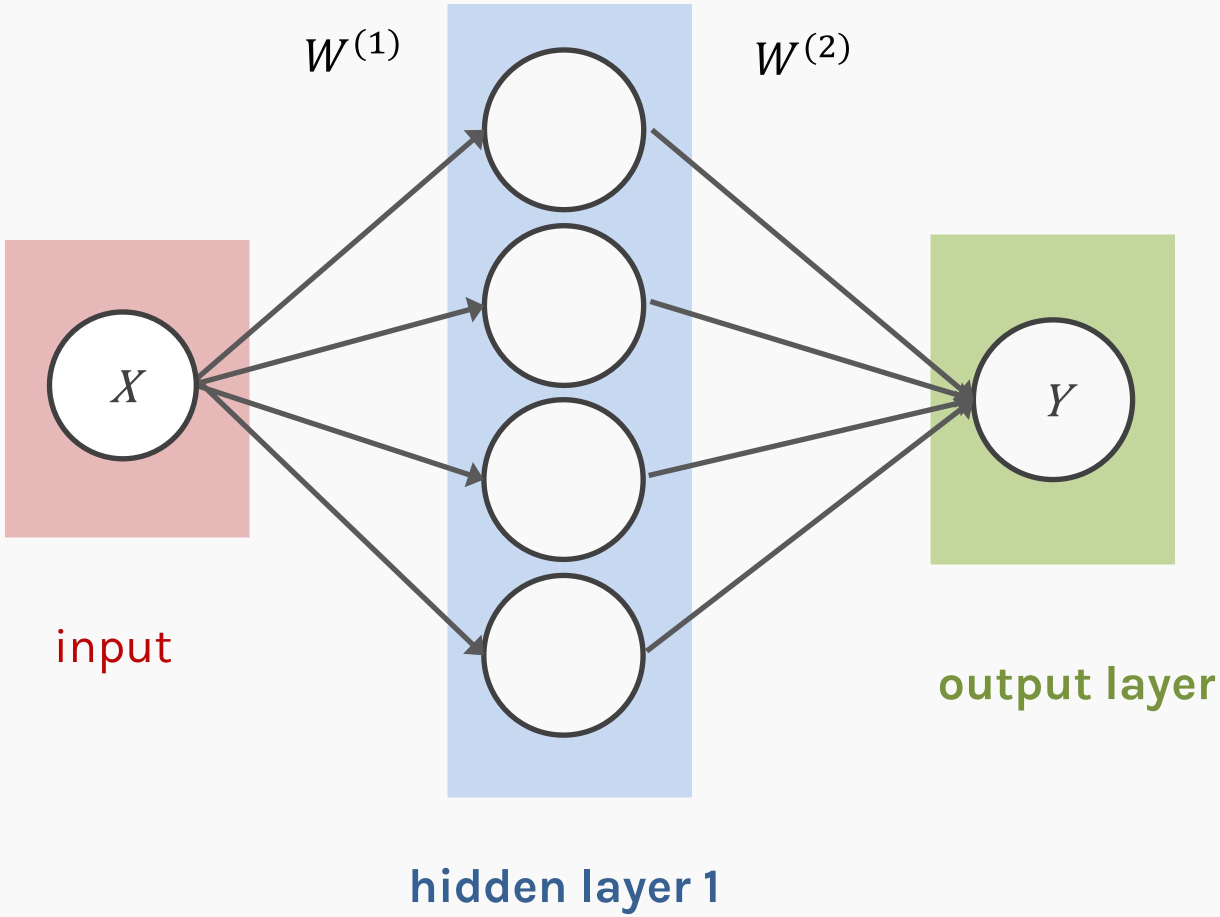
MLP as an additive model



activation

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

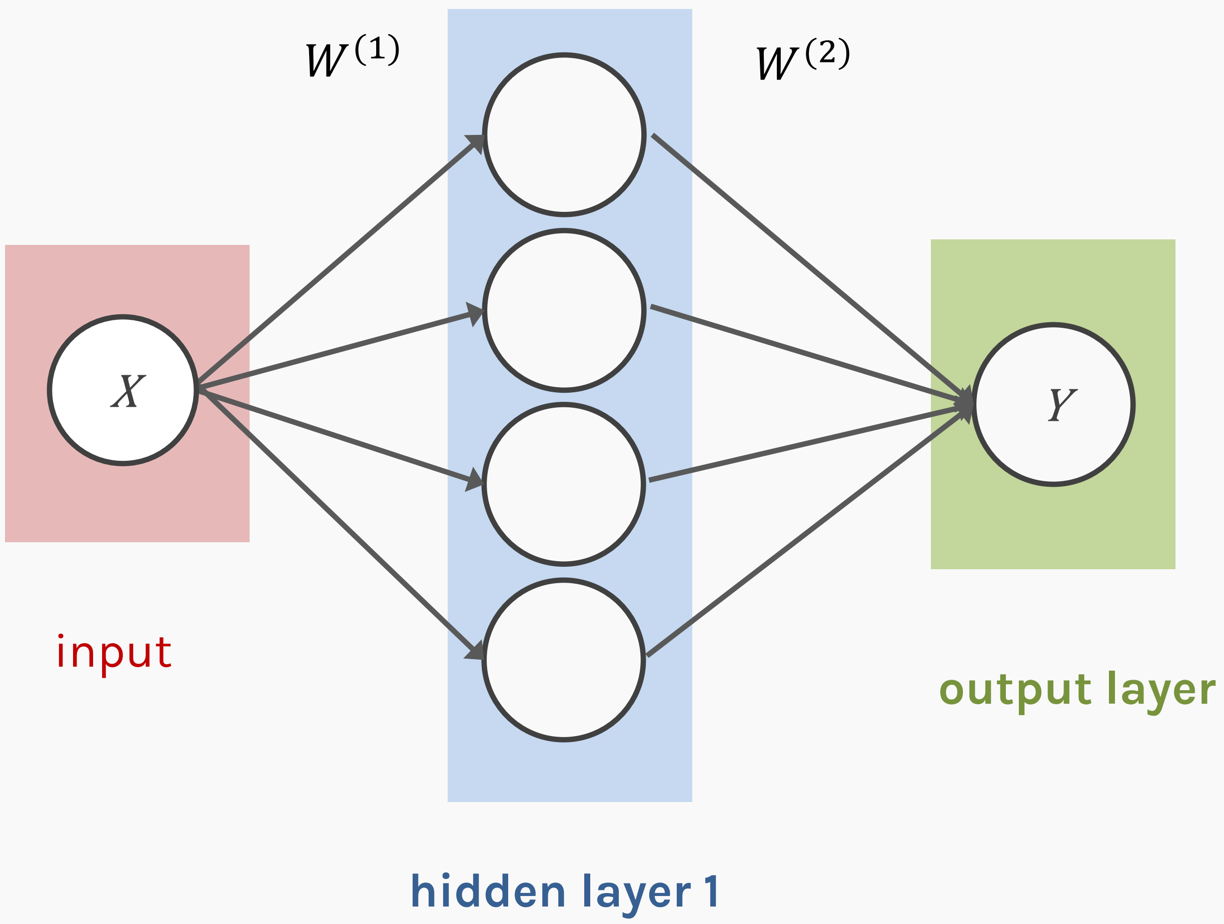
MLP as an additive model



activation

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

MLP as an additive model



activation

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

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

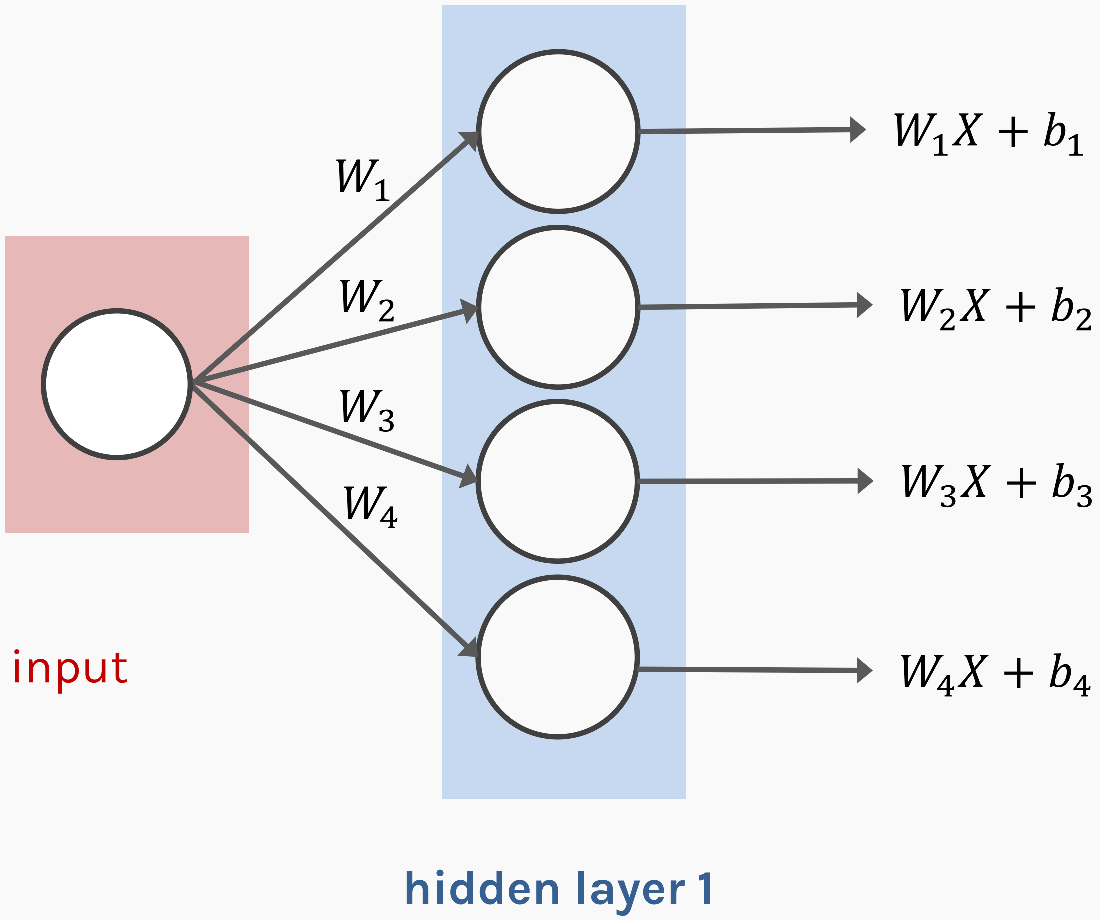
Main **drawbacks** of MLPs

- 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.

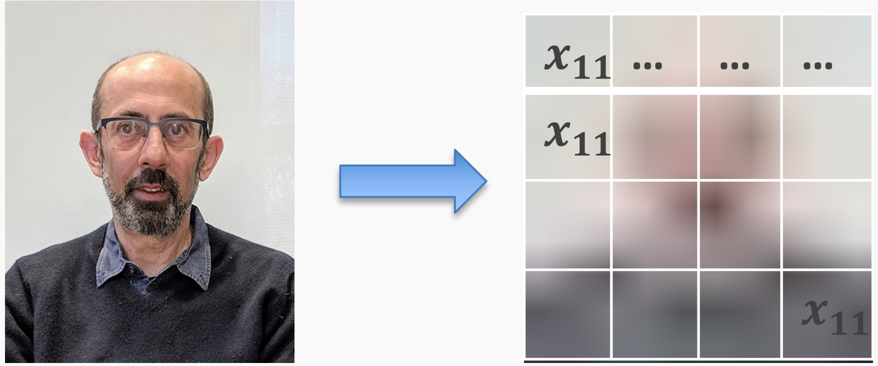
MLP: number of weights



How many weights?

- If $X \in \mathbb{R}$ then $|\,\mathrm{W}_i\,| = 1$
- If $X \in \mathbb{R}^m$ then $|\mathrm{W}_i| = \mathrm{m}$

MLP: number of weights for images



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

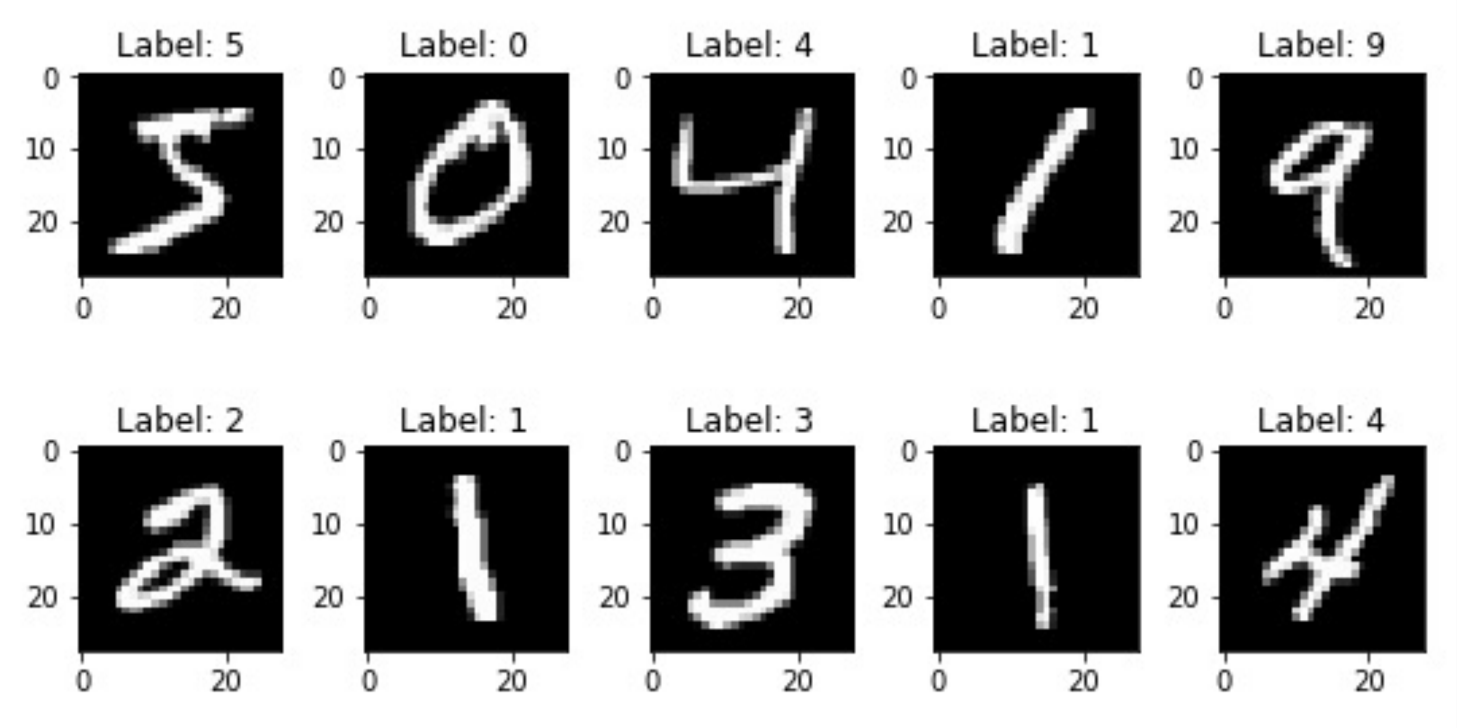
A strong motivation for performing model selection is to avoid overfitting, which we saw can happen when:

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

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

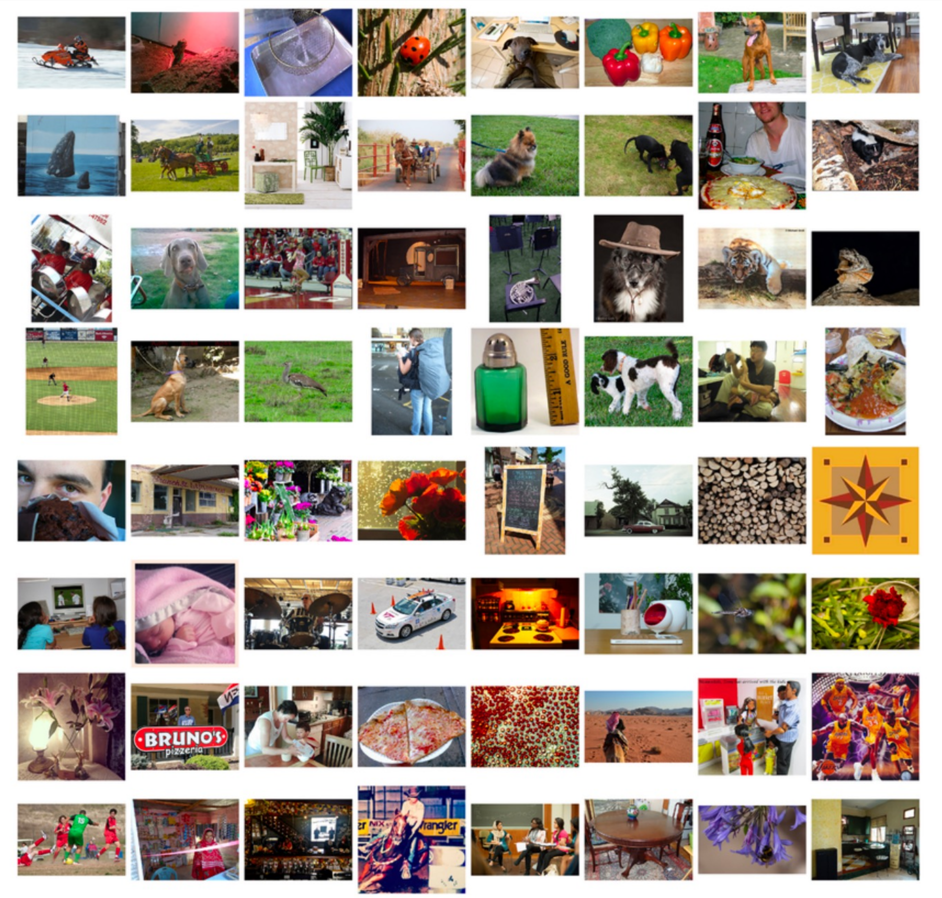
CIFAR10, a dataset of images that are commonly used to train machine learning models. It contains 60,000 32x32 color images in 10 different classes.

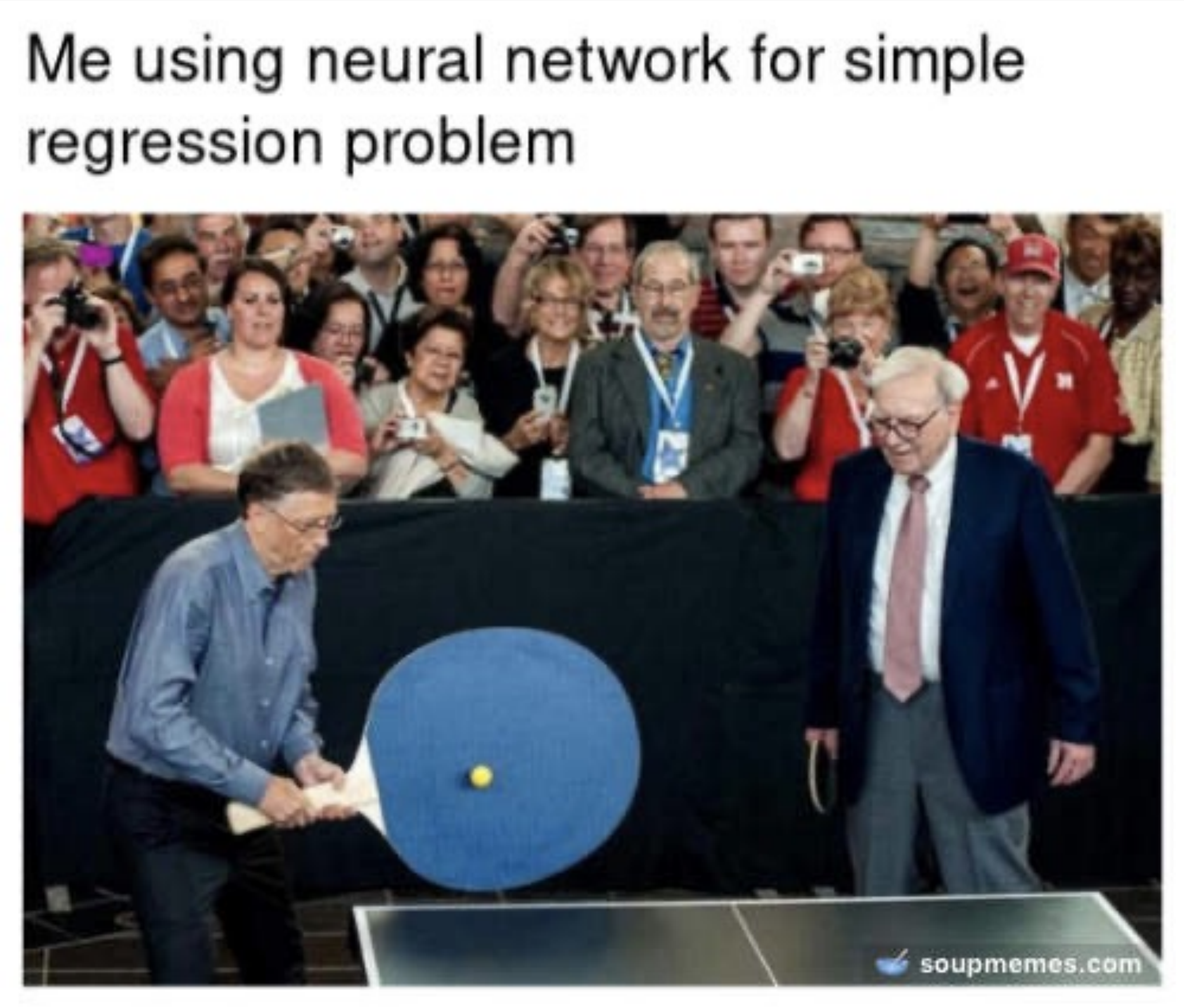
Each pixel is a feature: an MLP would have $32x32x3 + 1 \; = \; 3073$ weights per neuron!

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
soupmemes.com

Model Selection and Dimensionality Reduction

Recall from “before” that to reduce the number of predictors we can:

Model Selection and Dimensionality Reduction

Recall from "before" that to reduce the number of predictors we can:

- PCA

Model Selection and Dimensionality Reduction

Recall from “before” that to reduce the number of predictors we can:

- PCA
- Stepwise Variable Selection

Model Selection and Dimensionality Reduction

Recall from "before" that to reduce the number of predictors we can:

- PCA
- Stepwise Variable Selection
- Regularization, in particular L1 will produce sparsity

Model Selection and Dimensionality Reduction

Recall from “before” that 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

Model Selection and Dimensionality Reduction

Recall from “before” that 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

Feature extraction

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

x

WAIT FOR IT

Feature extraction

Features:

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

Feature extraction

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

Feature extraction

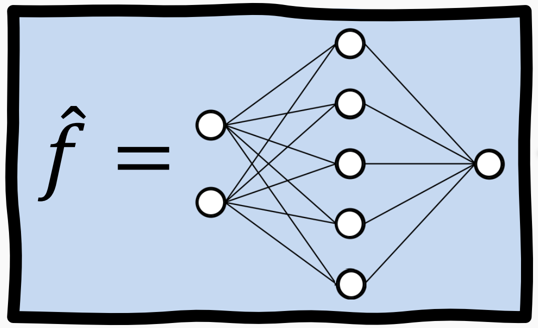
x

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

$\hat{y}$: PAVLOS or NOT PAVLOS

$x' \longrightarrow$



$\longrightarrow \hat{y}$

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…

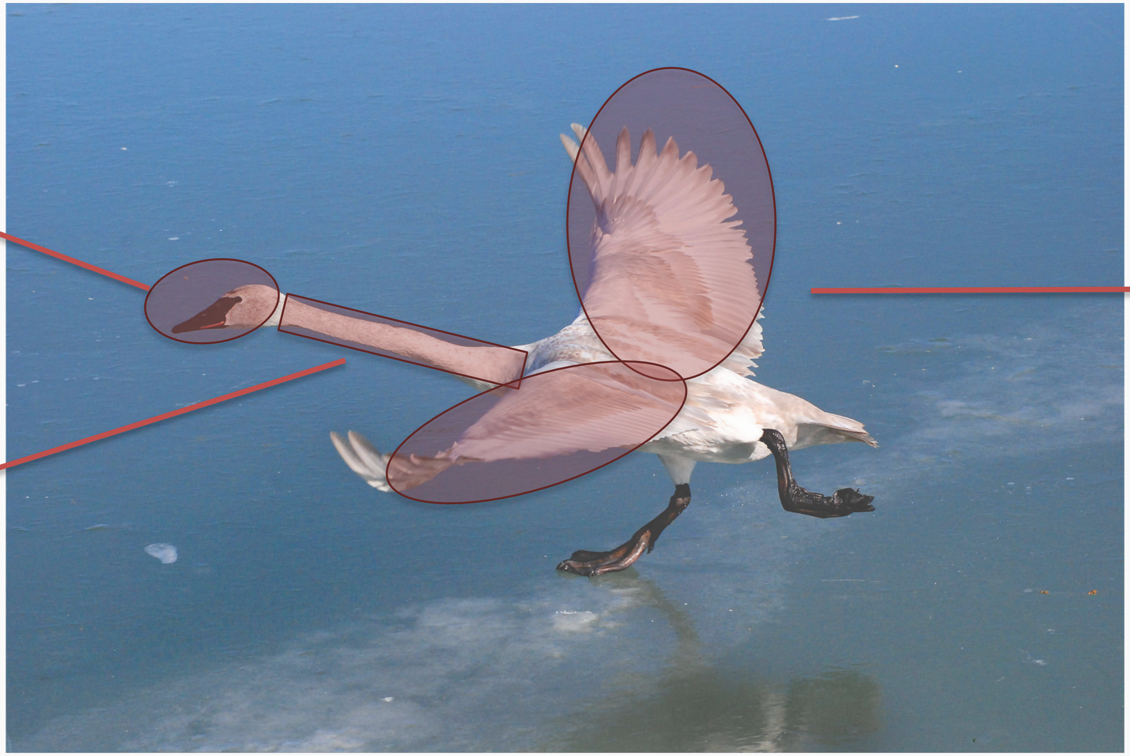
Round, elongated head with orange or black beak

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

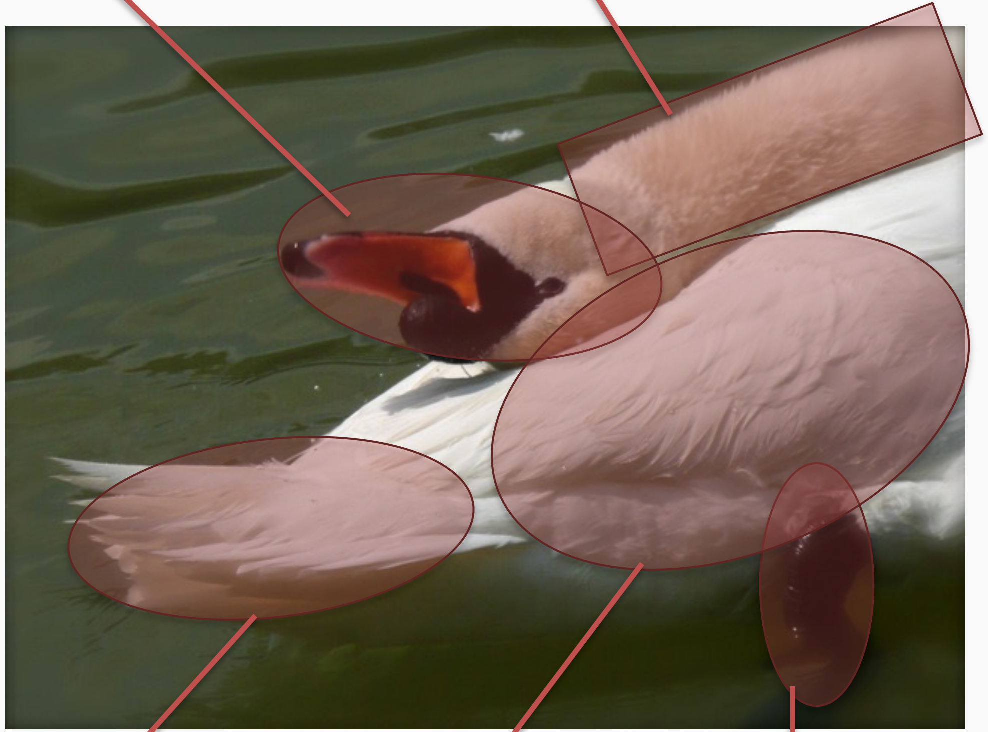


Oval-shaped white body with or without large white symmetric blobs (wings)

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

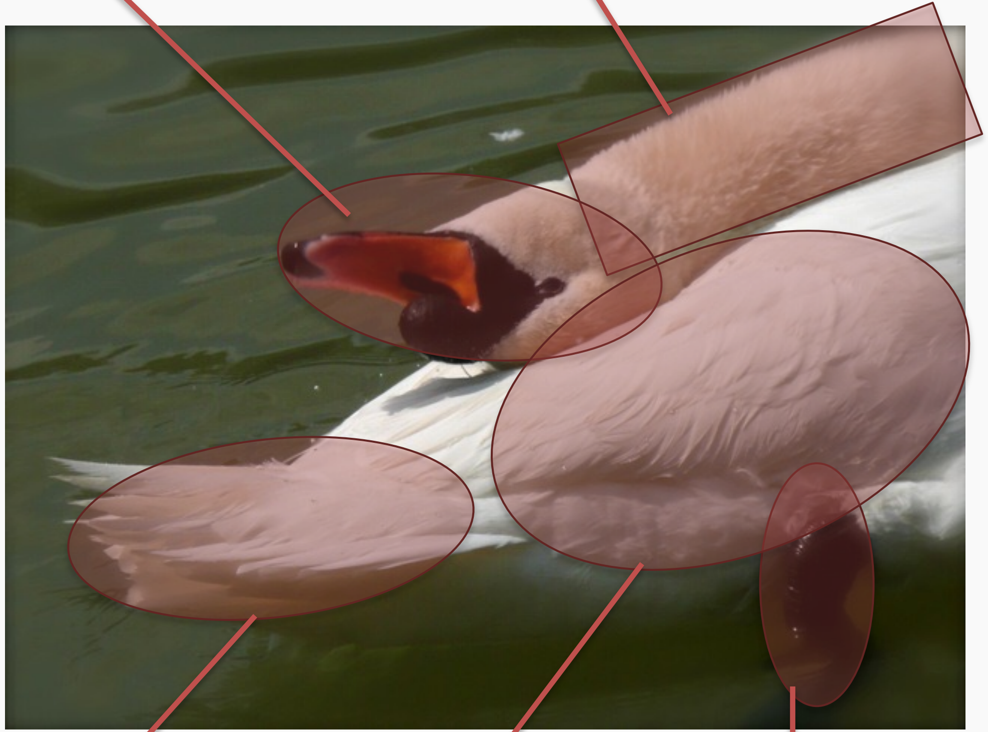
White, oval shaped body, with or without wings visible

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



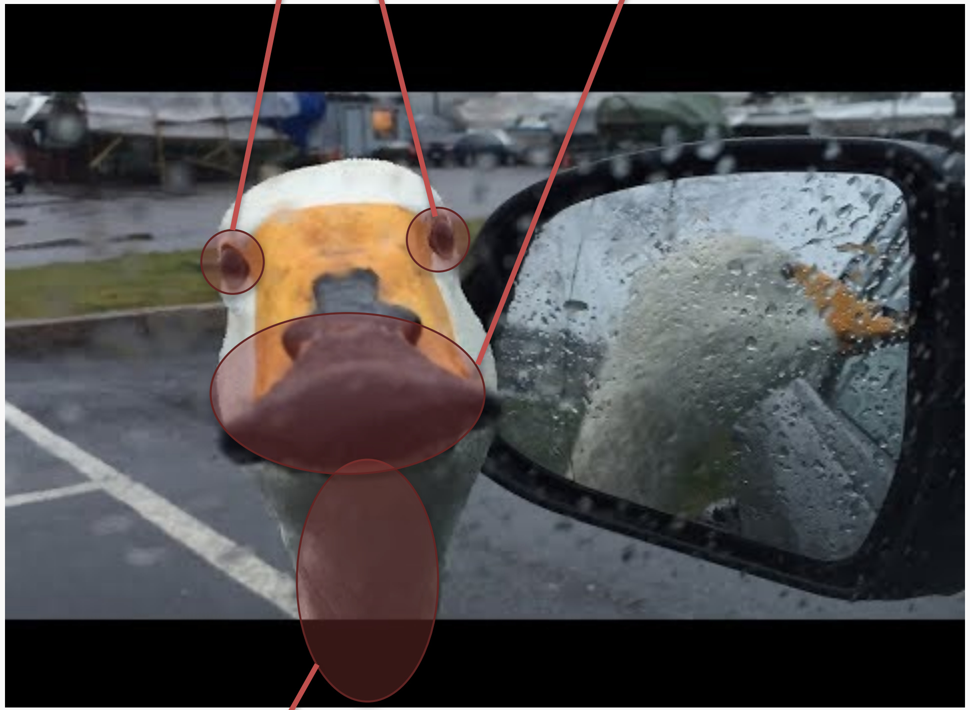
White tail, generally far from the head, looks feathery

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

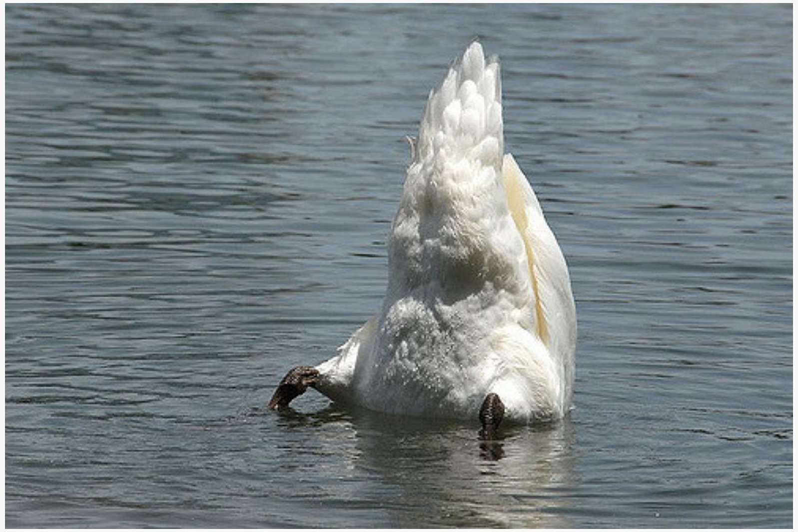


White elongated piece, can be squared or more triangular, can be obstructed sometimes

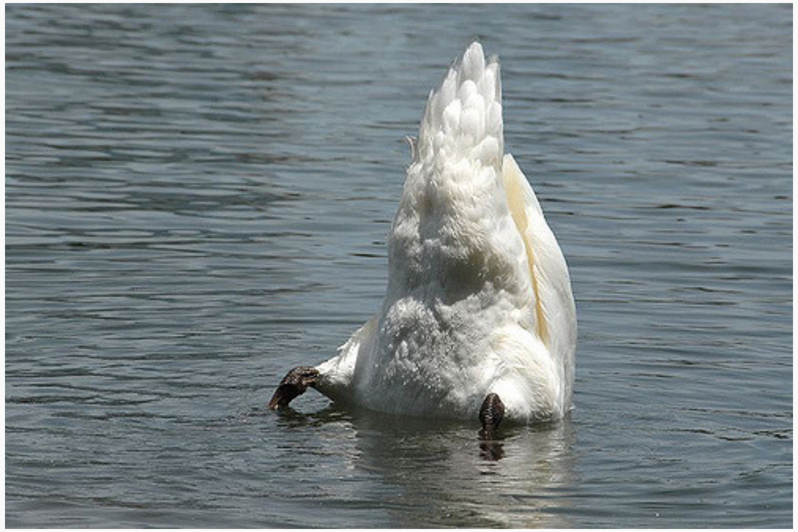
Luckily, the color is consistent…

We need to be able to deal with these cases

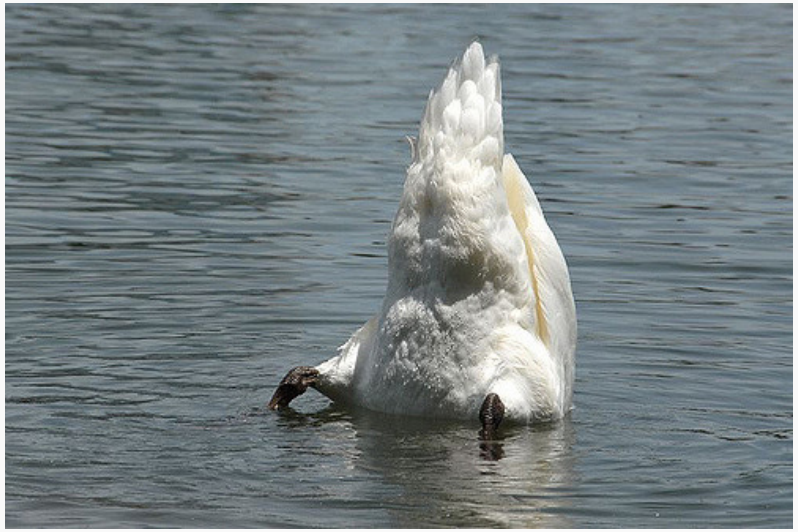
And these



And these



And these



And these

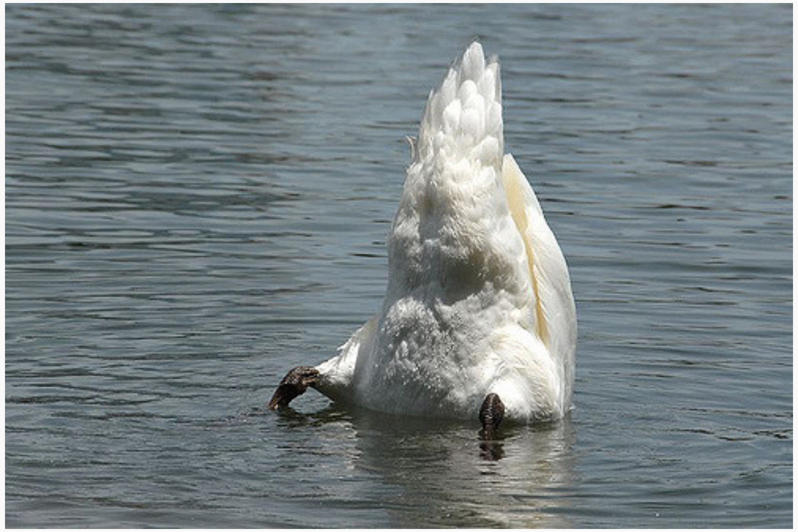
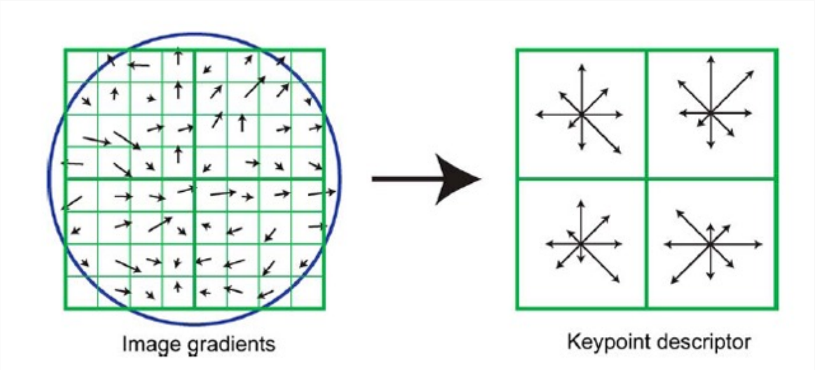
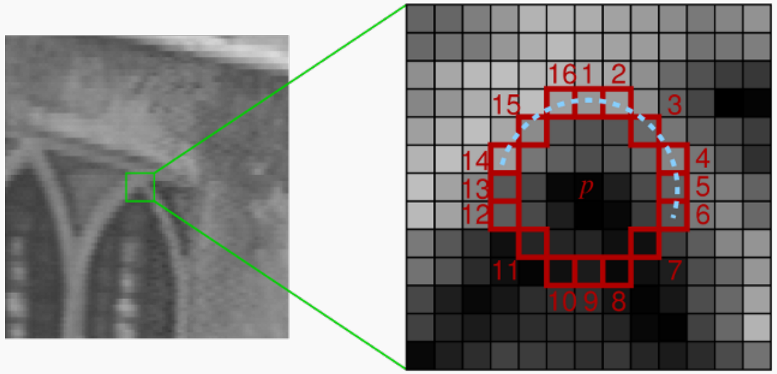


Image features

- 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*.

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.

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

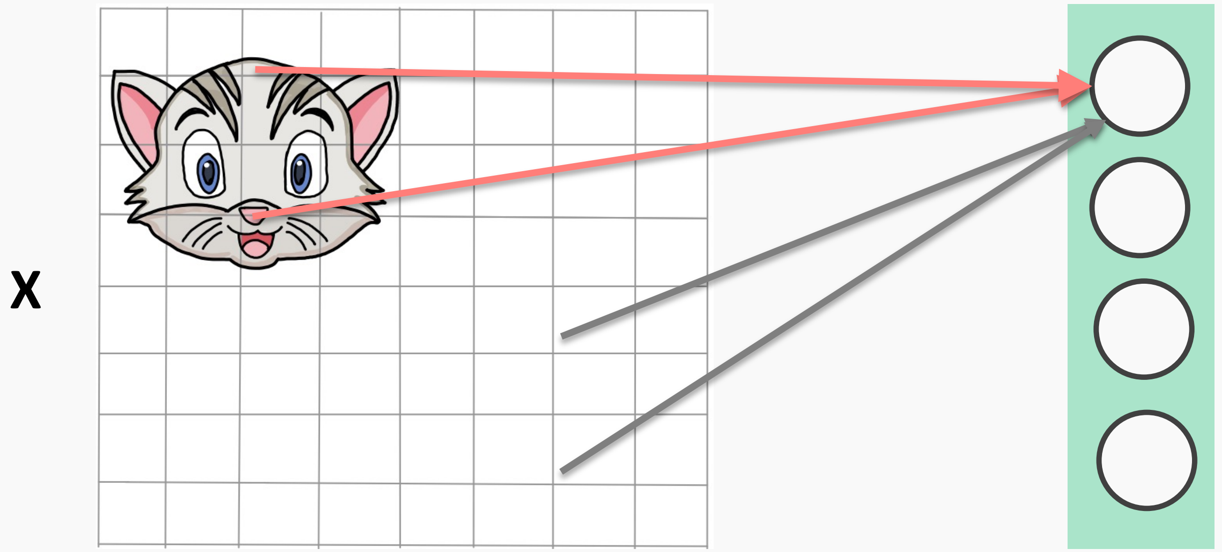
- Nearby Pixels are more strongly related that distant ones
- Objects are built up out of smaller parts
- Images are Local and Hierarchical

And images are invariant

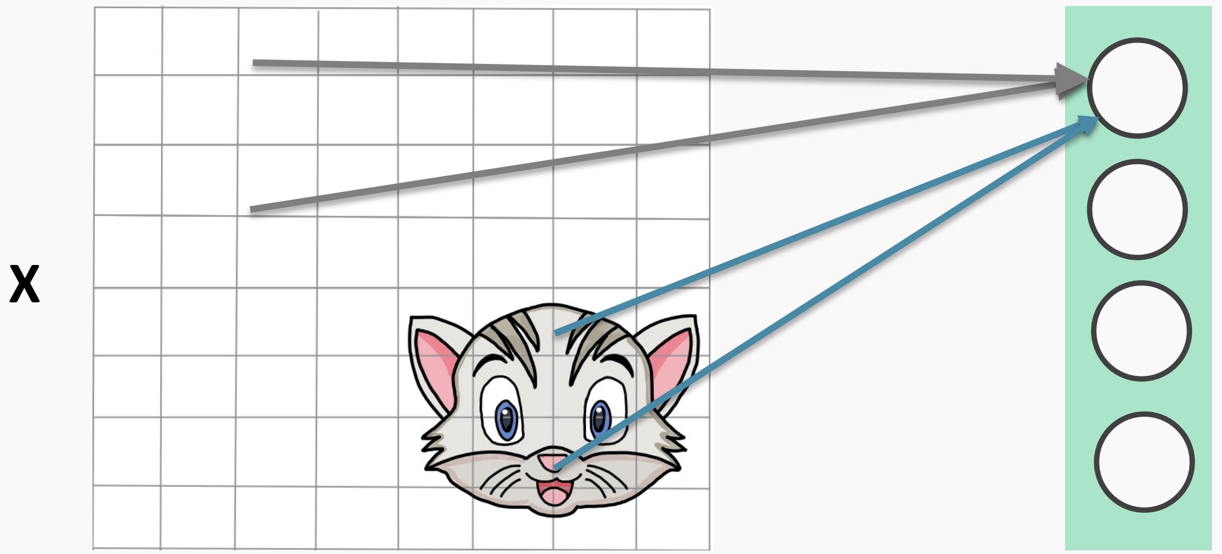
Outline

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 the weight W.

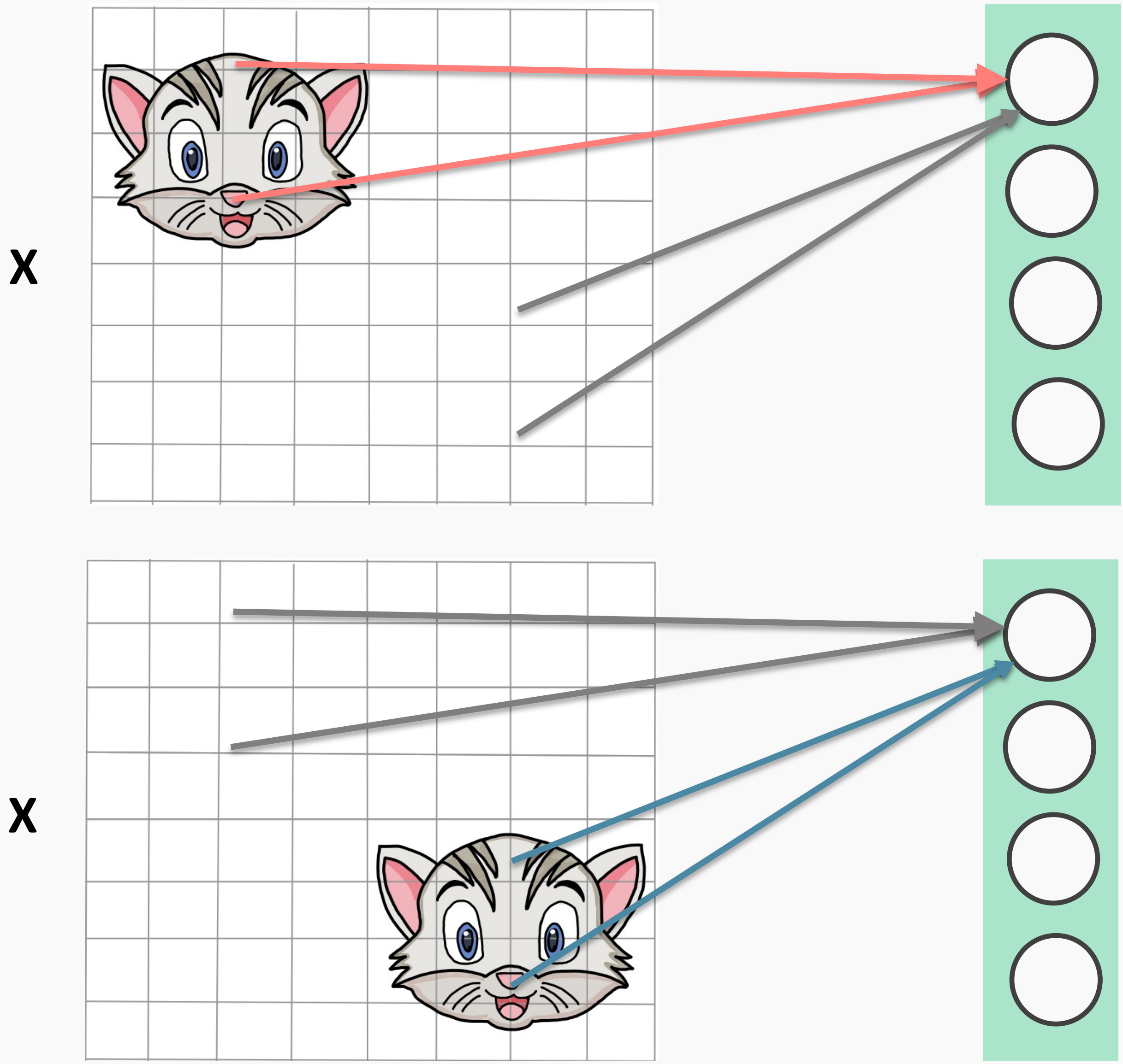


In this case, the red weights will be larger to better recognize cat.



In this case, the blue weights will be larger.

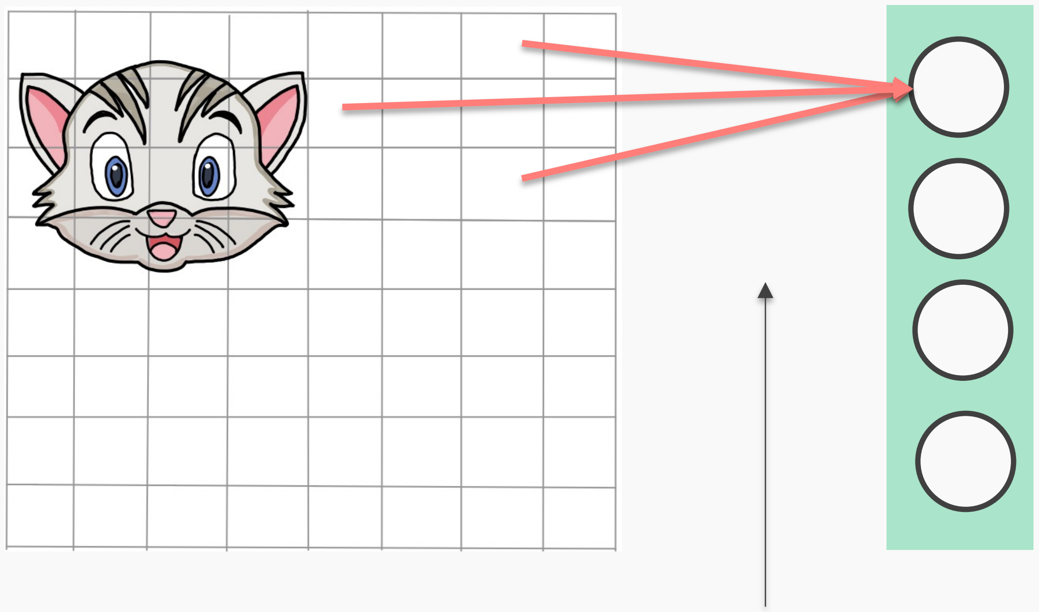
Each neuron from first layer has one weight per pixel. Recall, the importance of the predictors (here pixels) is given by the value of the coefficient, here the weight W.



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

Solution: Cut the image to smaller pieces.

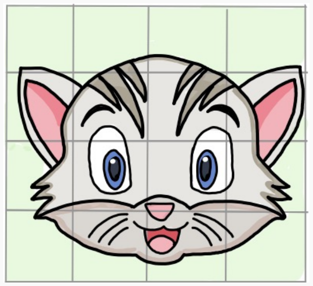
$X: \mathbb{R}^{8\times 8}$



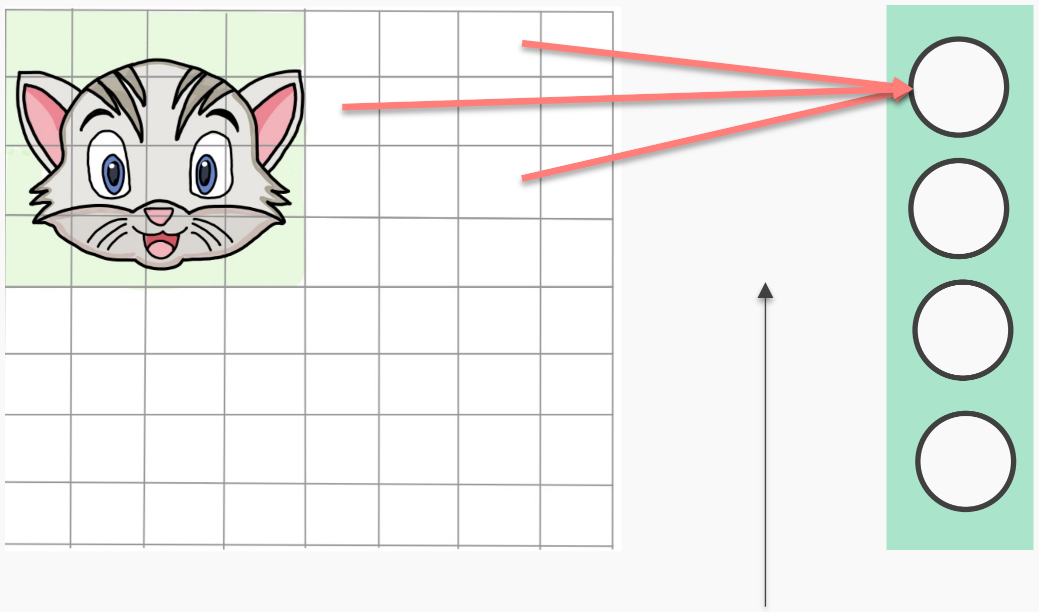
64 weights per neuron

Solution: Cut the image to smaller pieces.

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



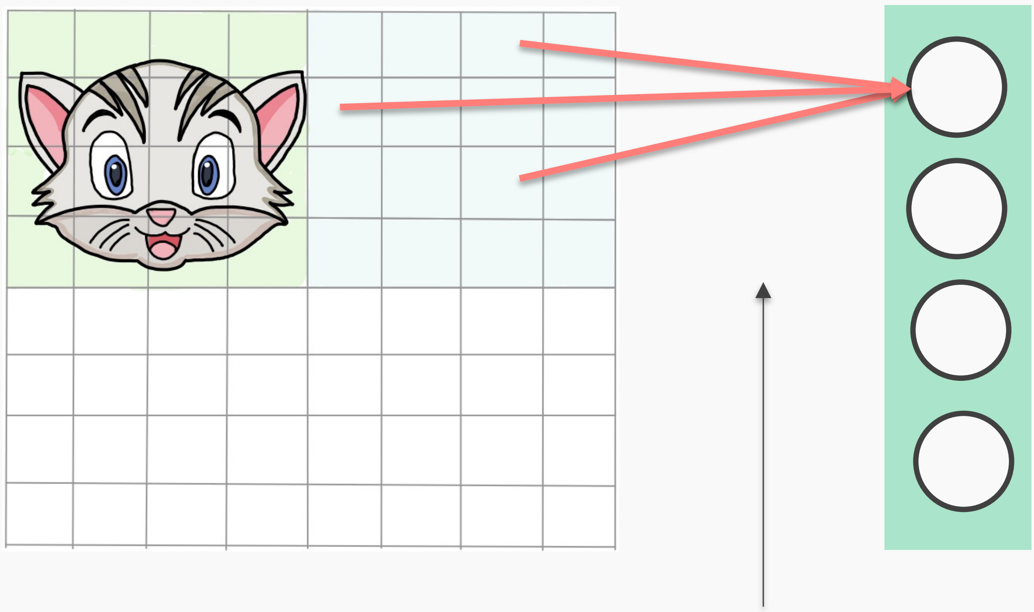
$X: \mathbb{R}^{8\times 8}$



64 weights per neuron

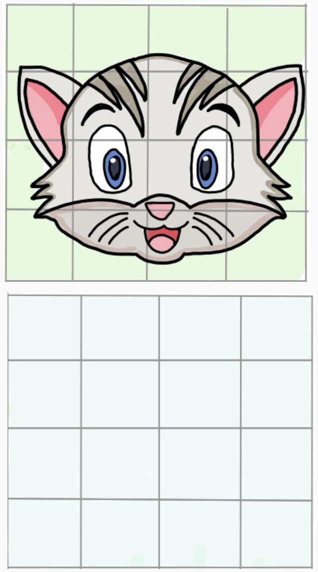
Solution: Cut the image to smaller pieces.

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



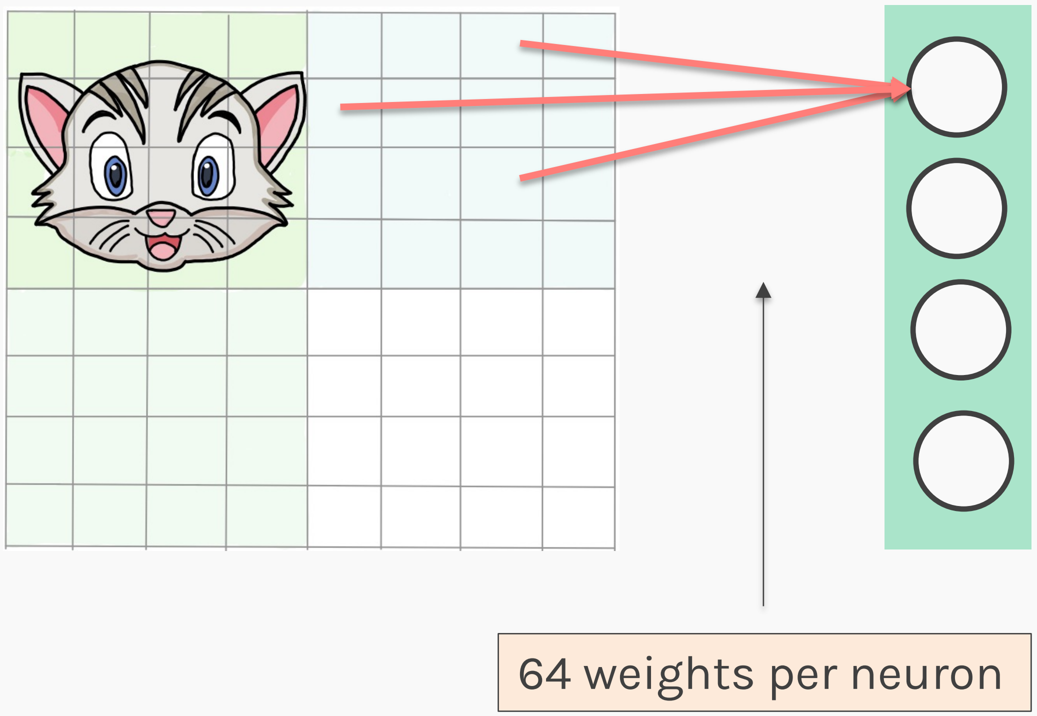
64 weights per neuron

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

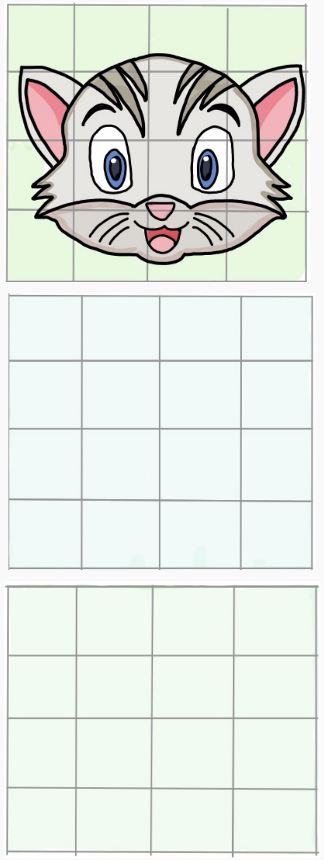


Solution: Cut the image to smaller pieces.

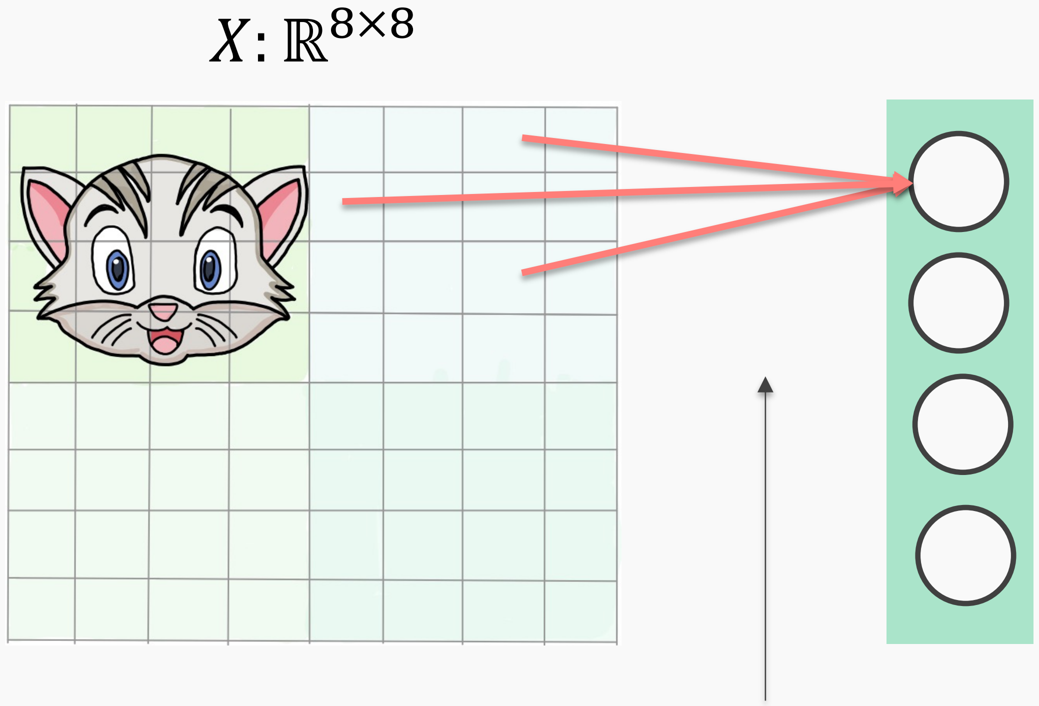
$X: \mathbb{R}^{8\times 8}$



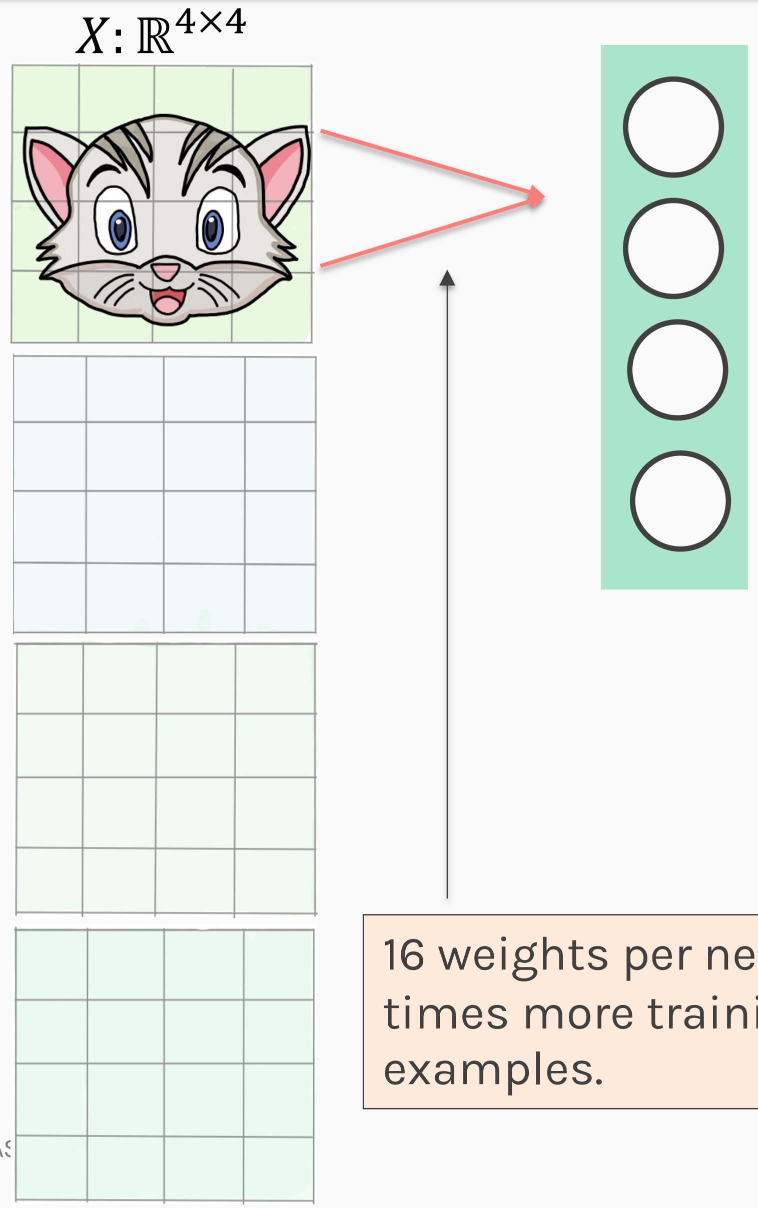
$X: \mathbb{R}^{4\times 4}$



Solution: Cut the image to smaller pieces.

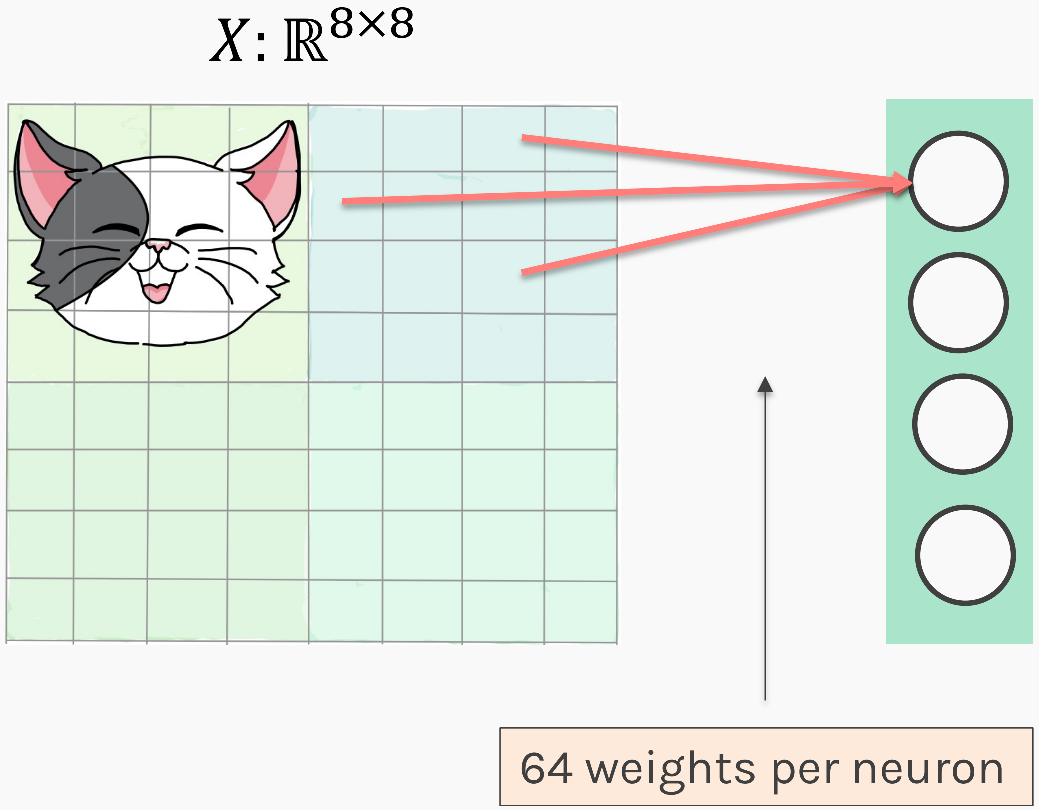


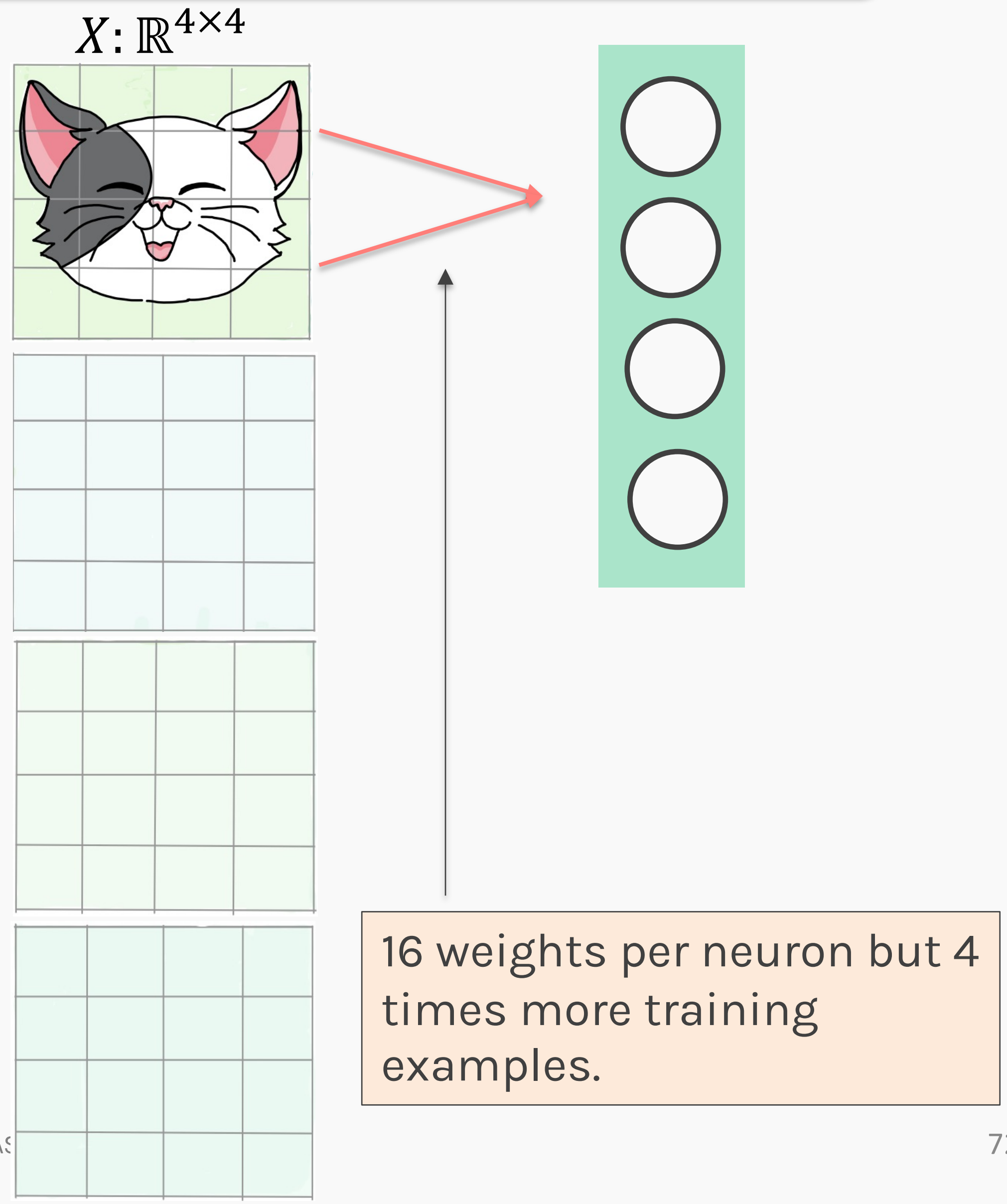
64 weights per neuron



16 weights per neuron but 4 times more training examples.

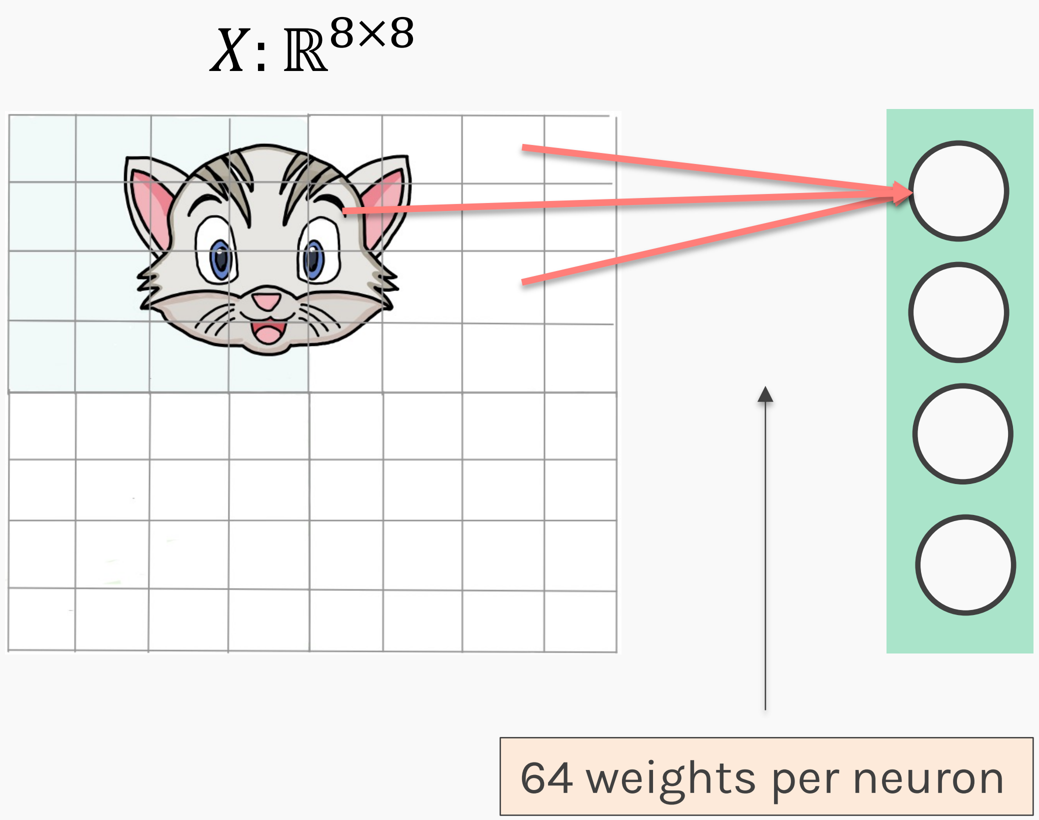
Do the same for all images

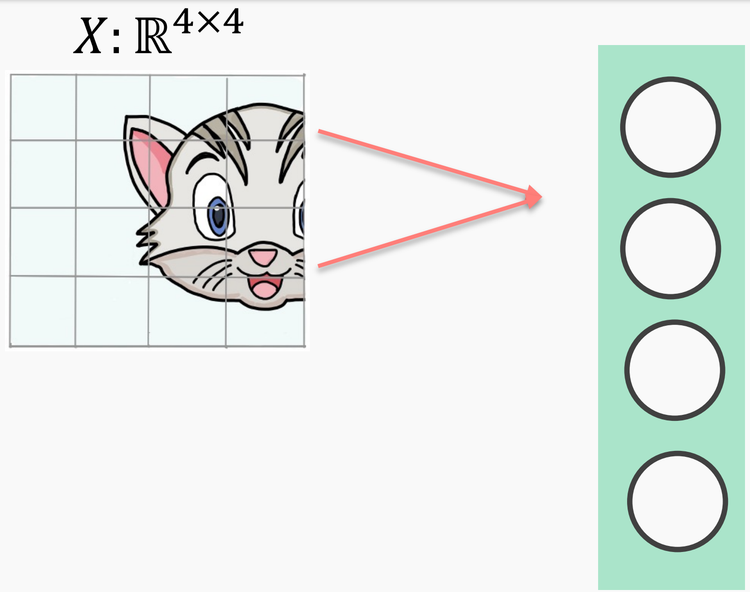




What if the cat is not entirely in one of the 4 boxes?

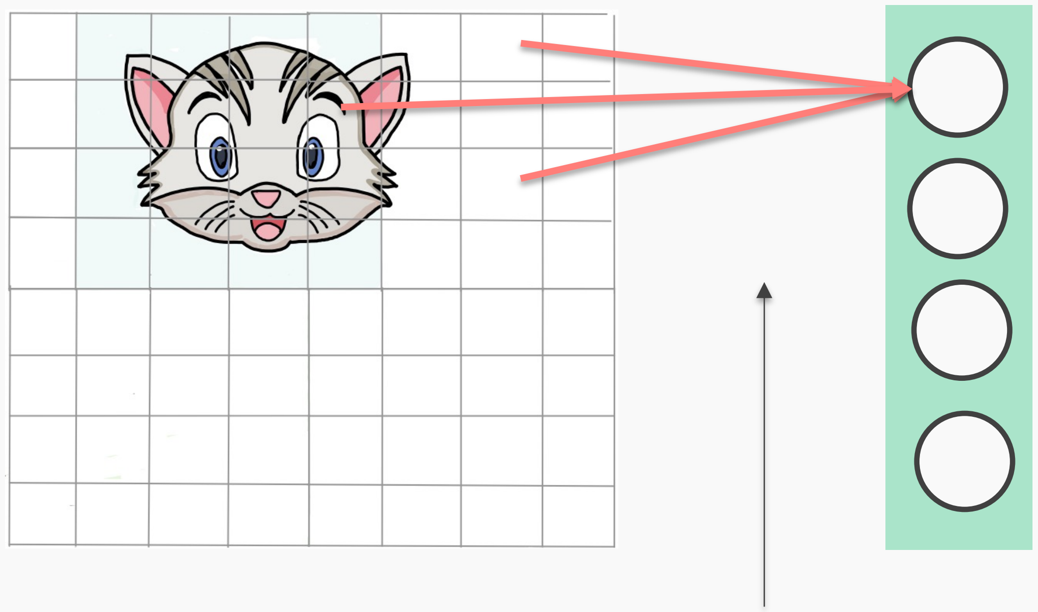
What if the cat is not entirely in one of the 4 boxes?





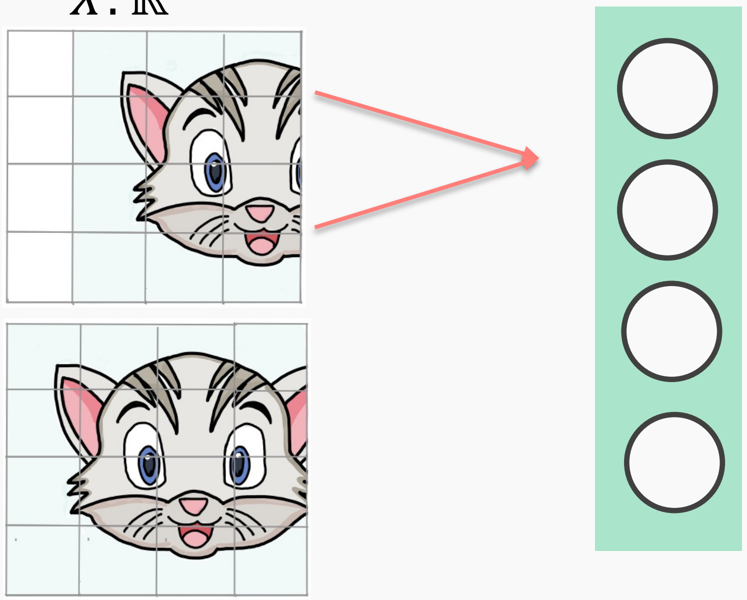
What if the cat is not entirely in one of the 4 boxes?

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



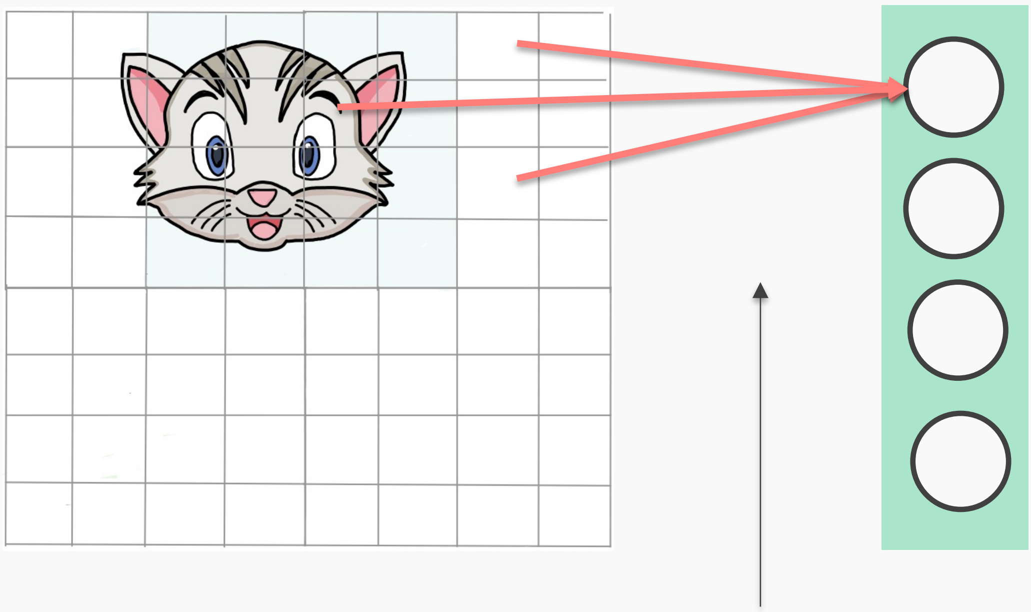
64 weights per neuron

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



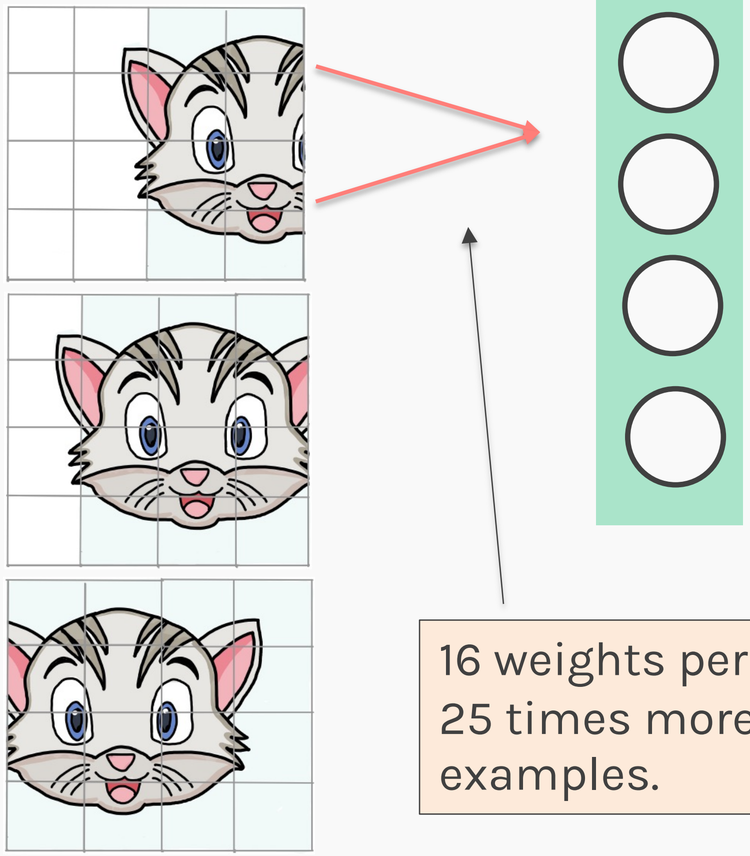
What if the cat is not entirely in one of the 4 boxes?

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



64 weights per neuron

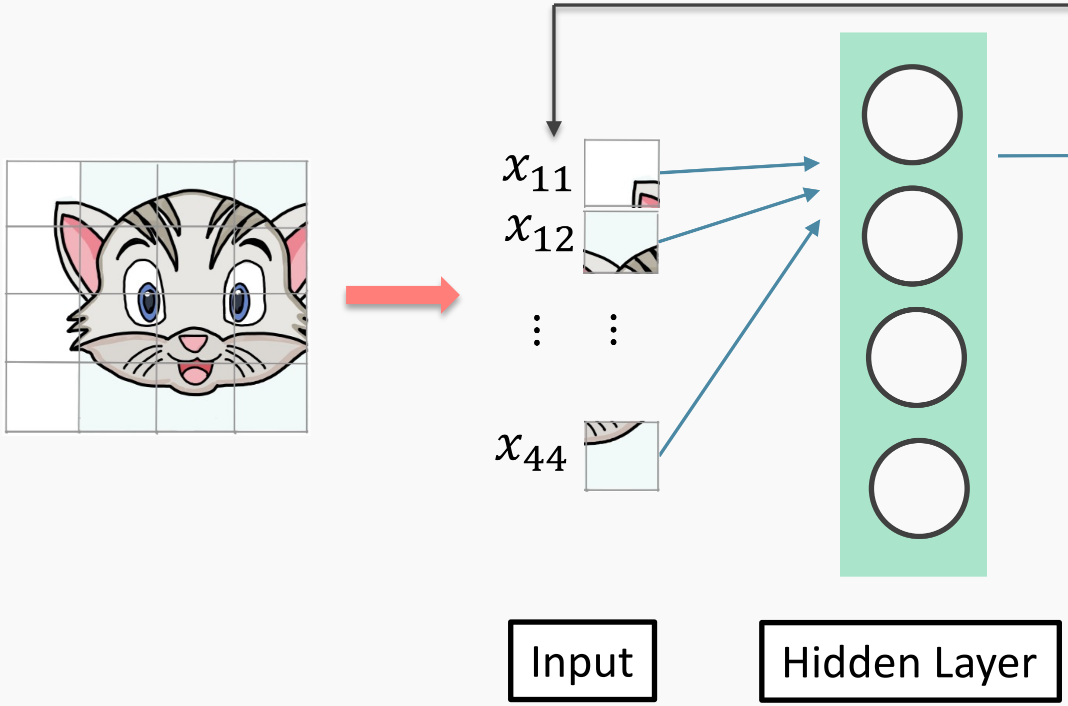
$X: \mathbb{R}^{4\times4}$



16 weights per neuron but 25 times more training examples.

Sliding Window

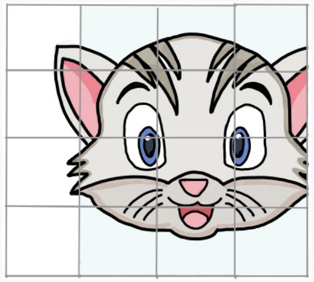
Convolution

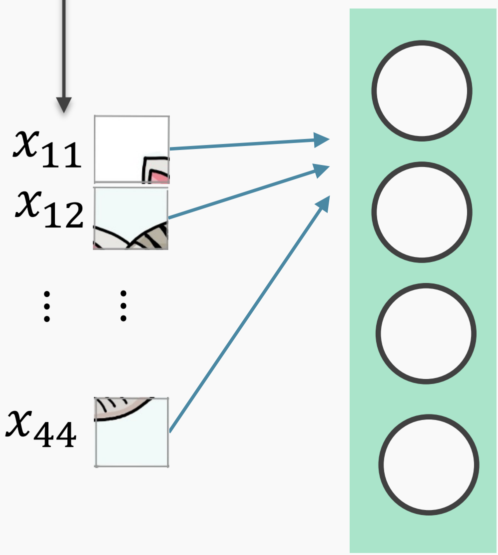


$$h_{11} = f(\underbrace{W_{11}x_{11} + W_{12}x_{12} + \cdots + W_{44}x_{44}}_{\text{16 weights}} + b)$$

Index here represents the cutout

Convolution





Input

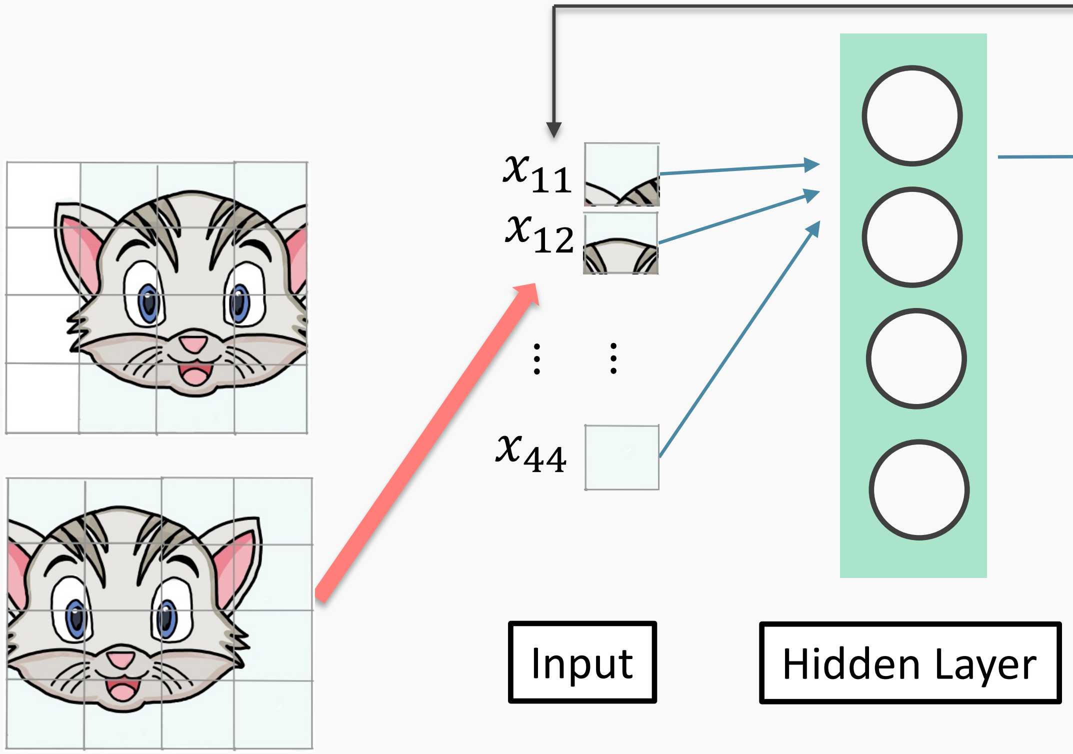
Hidden Layer

$$h_{11} = f(\underbrace{W_{11}x_{11} + W_{12}x_{12} + \cdots + W_{44}x_{44} + b}_{\text{16 weights}})$$

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

Element wise multiplication and addition of all products

Convolution



$$h_{12} = f(\underbrace{W_{11}x_{11} + W_{12}x_{12} + \cdots + W_{44}x_{44} + b}_{\text{16 weights}})$$

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

Element wise multiplication and addition of all products

Convolution

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

Convolution

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

KERNEL, K

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

Convolution

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

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

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

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

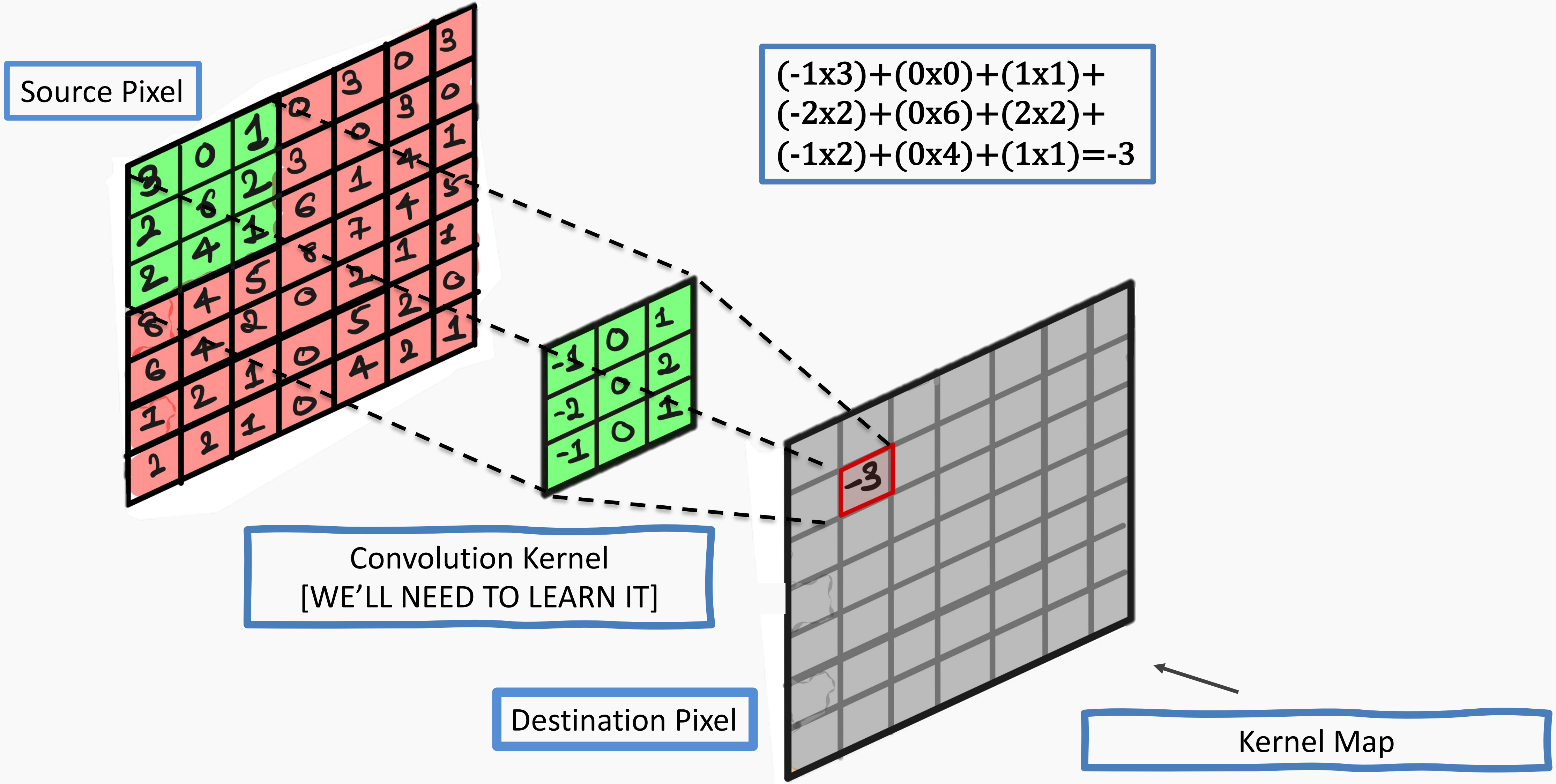
- Discrete convolution:

$$(f * g)(t) = \sum_{a=-\infty}^{\infty} f(a)g(t - a)$$

- Discrete cross-correlation:

$$\boxed{(f \star g)(t) = \sum_{a=-\infty}^{\infty} f(a)g(t + a)}$$

“Convolution” Operation



"Convolution" Operation in action

What does convolving an image with a Kernel do?

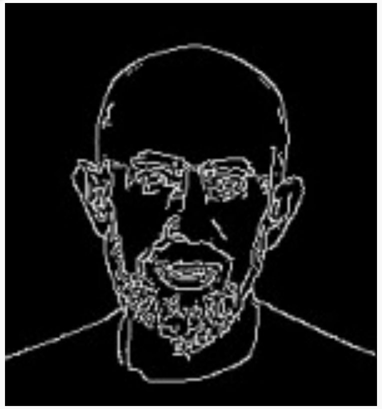
“Convolution” Operation in action

What does convolving an image with a Kernel do?

Kernel

Edge detection

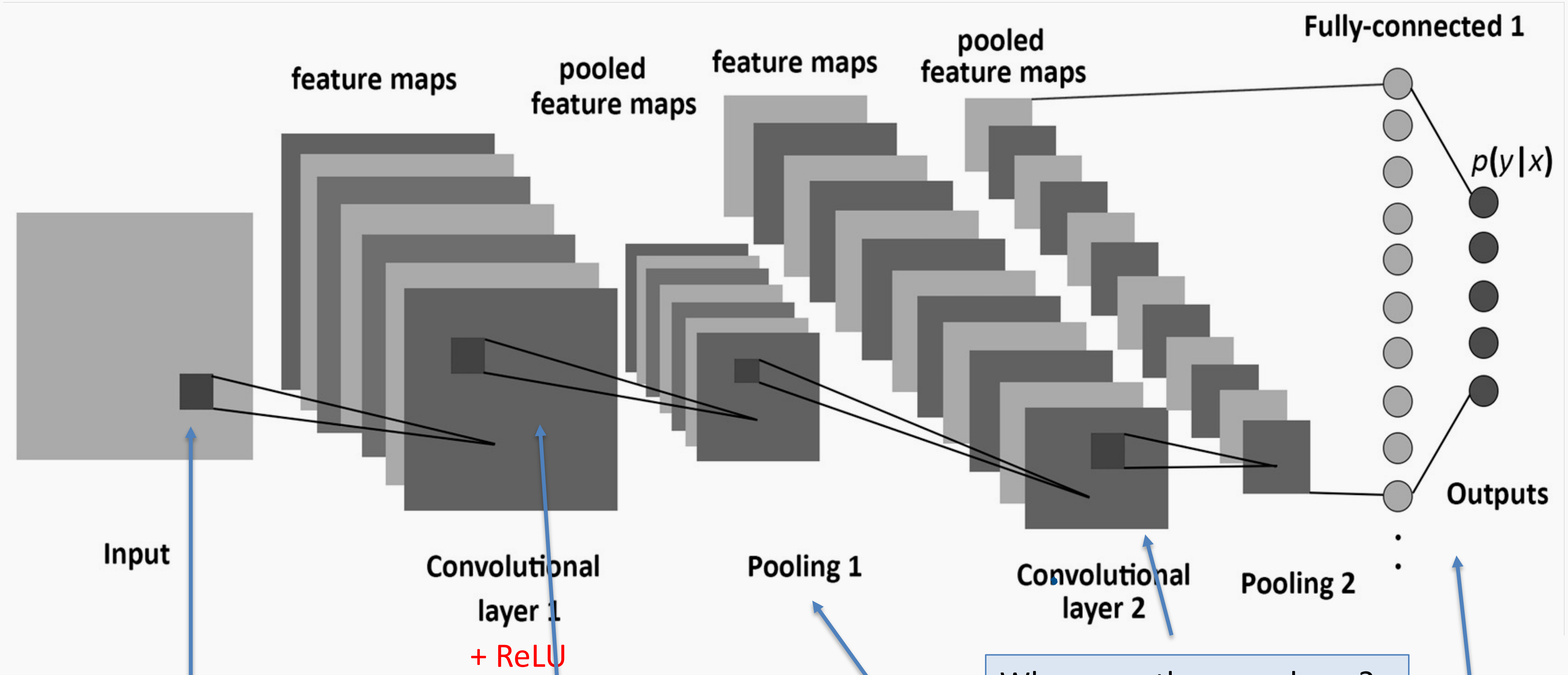
$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$



Sharpen

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

A Convolutional Network



What is this square?

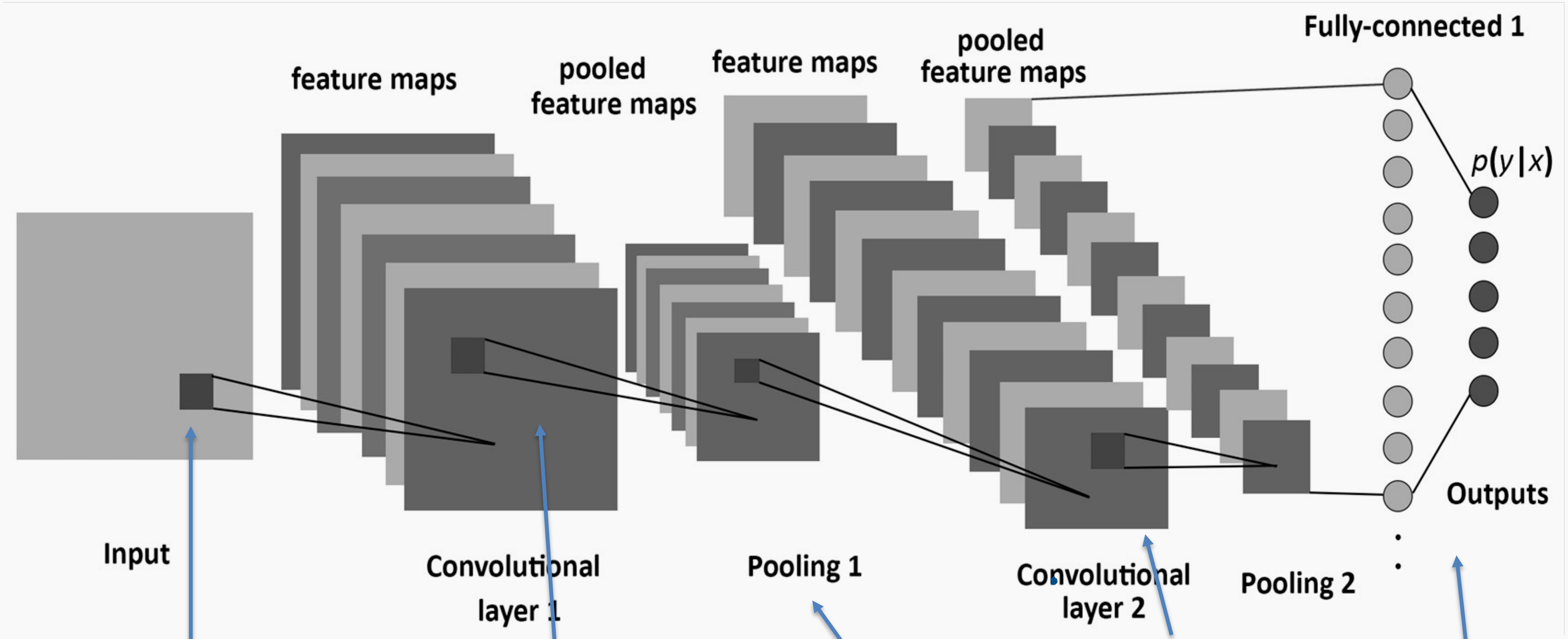
What are feature maps and why multiple feature maps?

What is Pooling?

Why more than one layer?

What is going on here?

A Convolutional Network



~~What is this square?~~

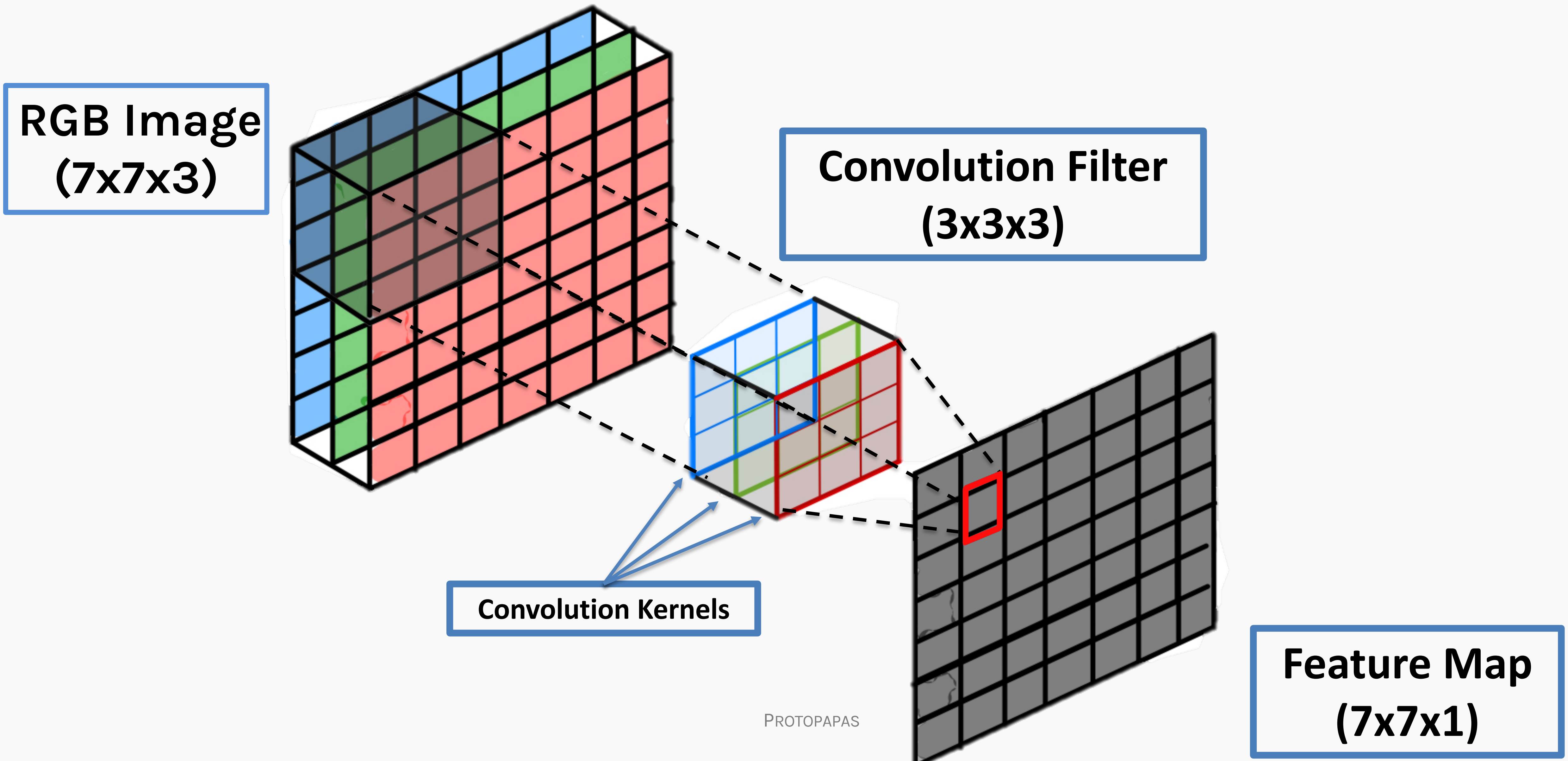
What are feature maps and why multiple feature maps?

What is Pooling?

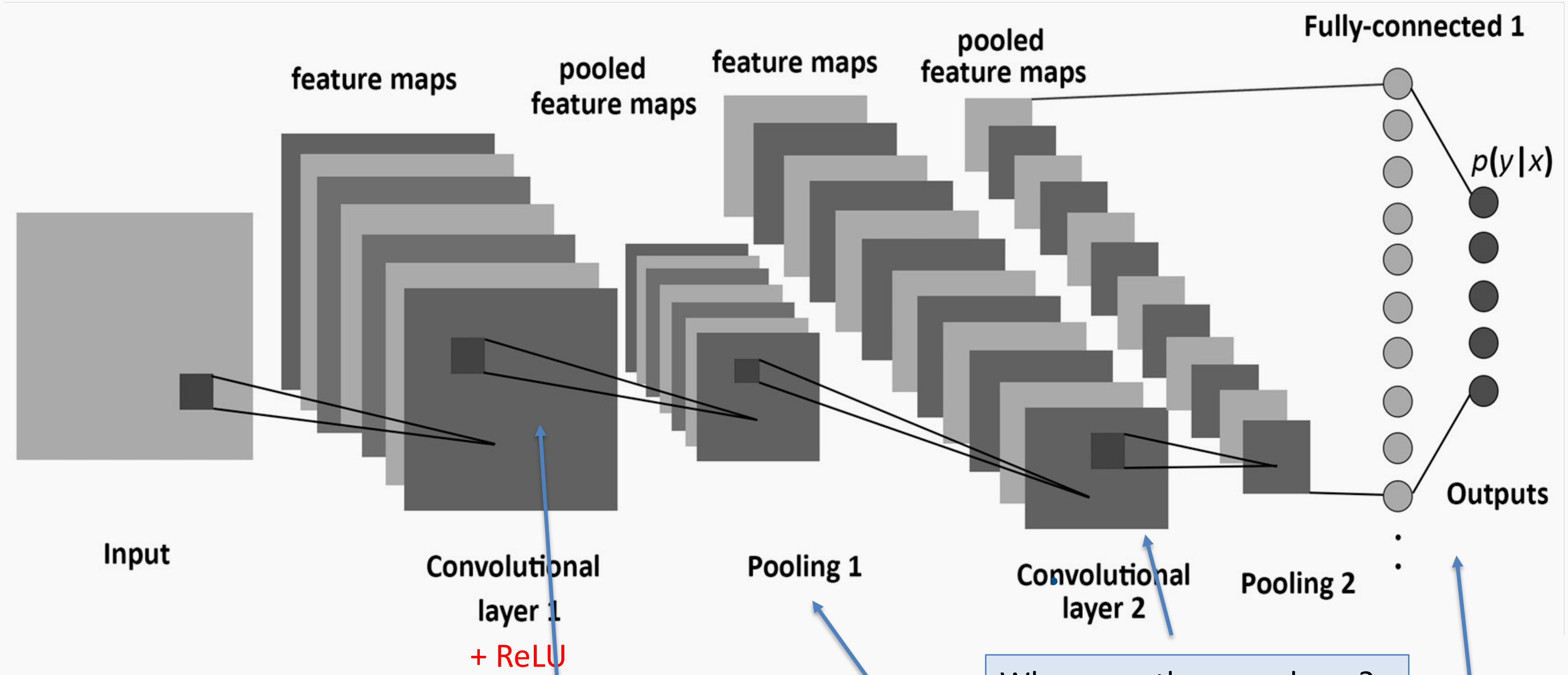
Why more than one layer?

What is going on here?

“Convolution” Operation



A Convolutional Network



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

What is Pooling?

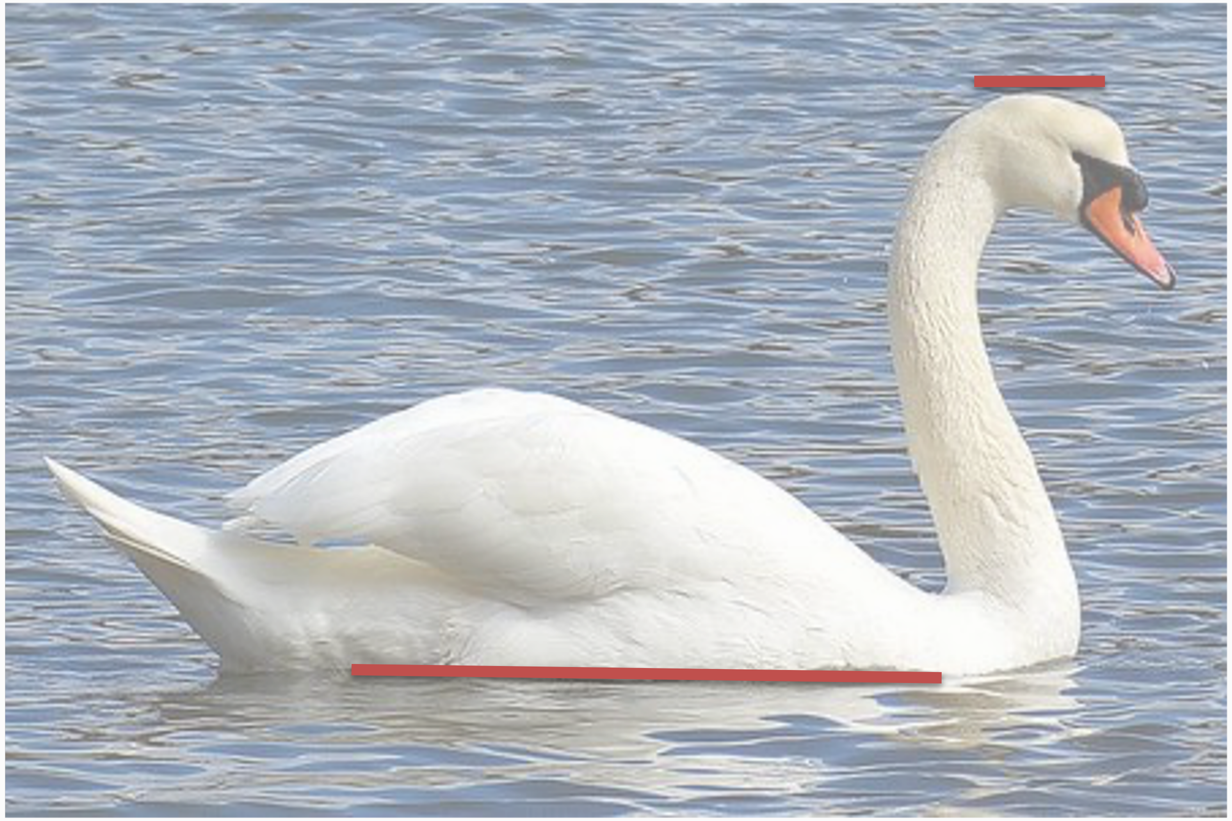
Why more than one layer?

What is going on here?

Why more than one feature map?

LAYER 1:

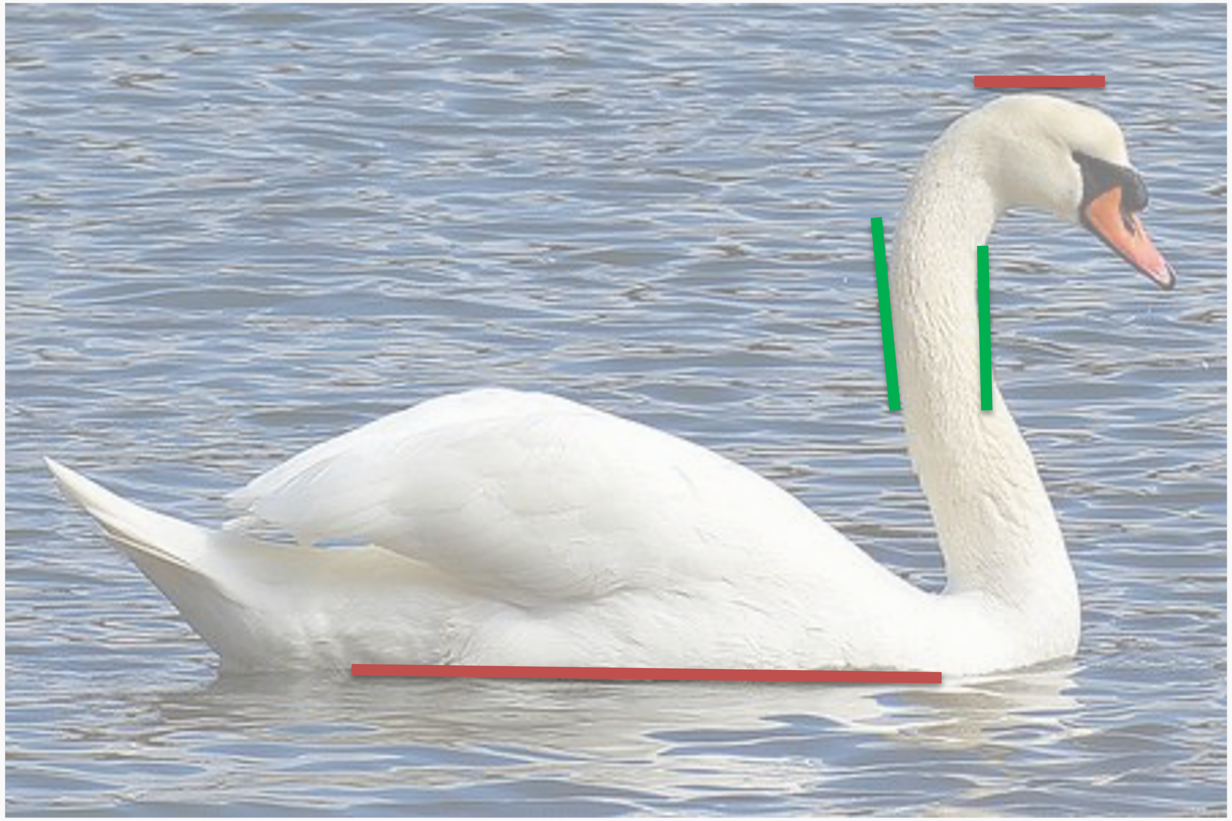
Why more than one feature map?



LAYER 1:

Filter 1: Horizontal Lines

Why more than one feature map?

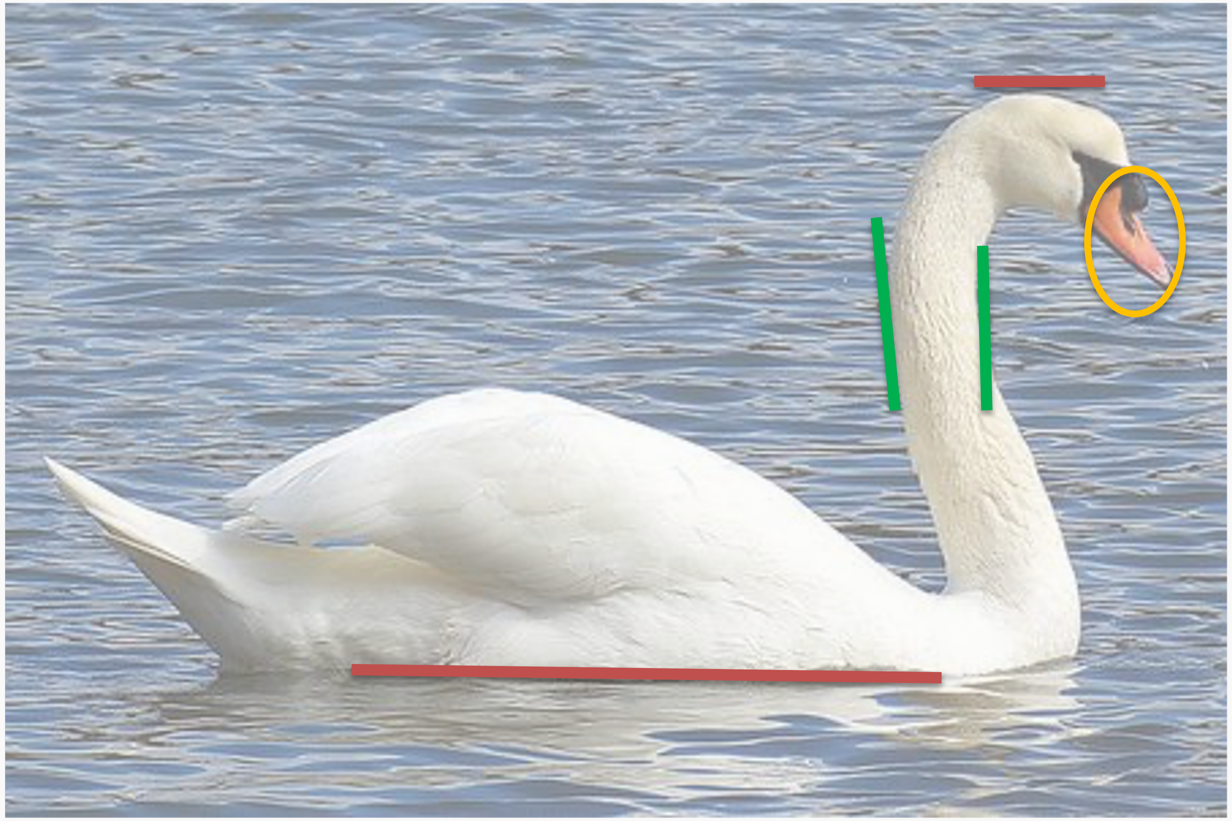


LAYER 1:

Filter 1: Horizontal Lines

Filter 2: Vertical Lines

Why more than one feature map?



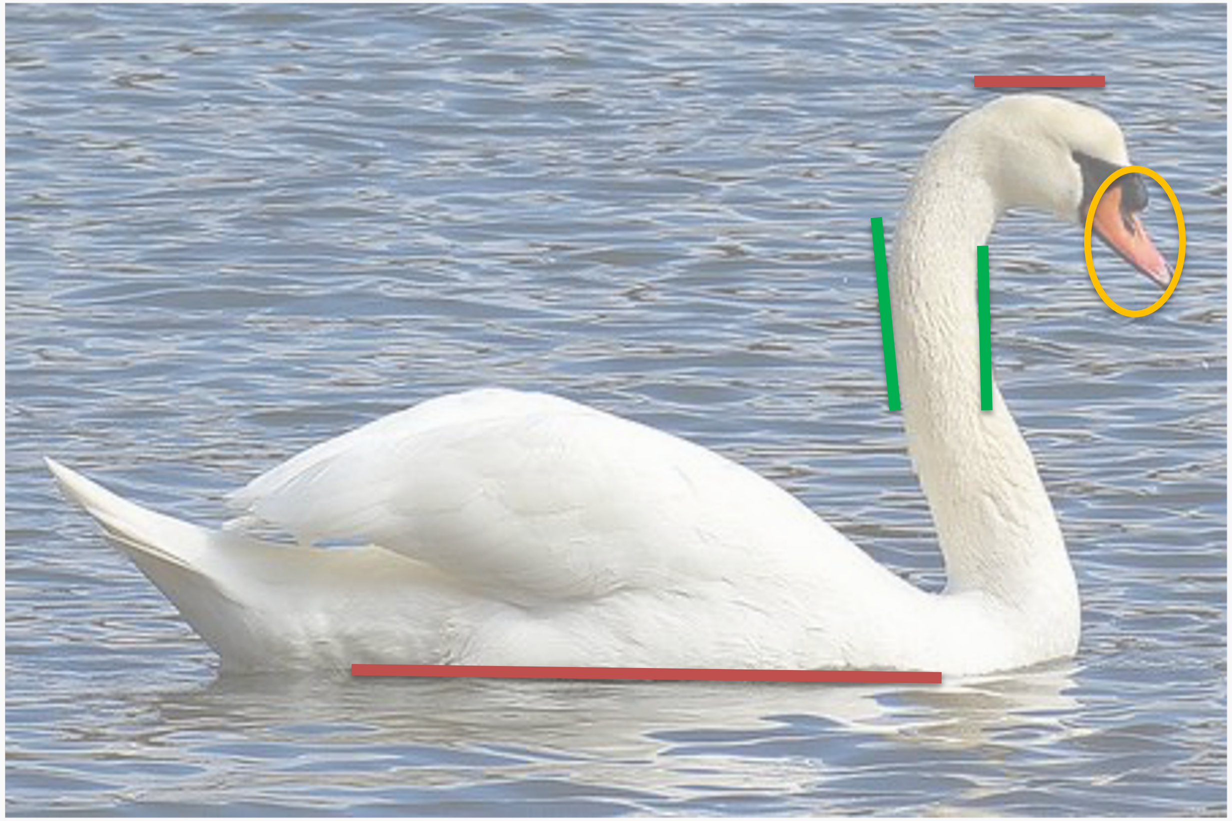
LAYER 1:

Filter 1: Horizontal Lines

Filter 2: Vertical Lines

Filter 3: Orange bulb

Why more than one feature map?



LAYER 1:

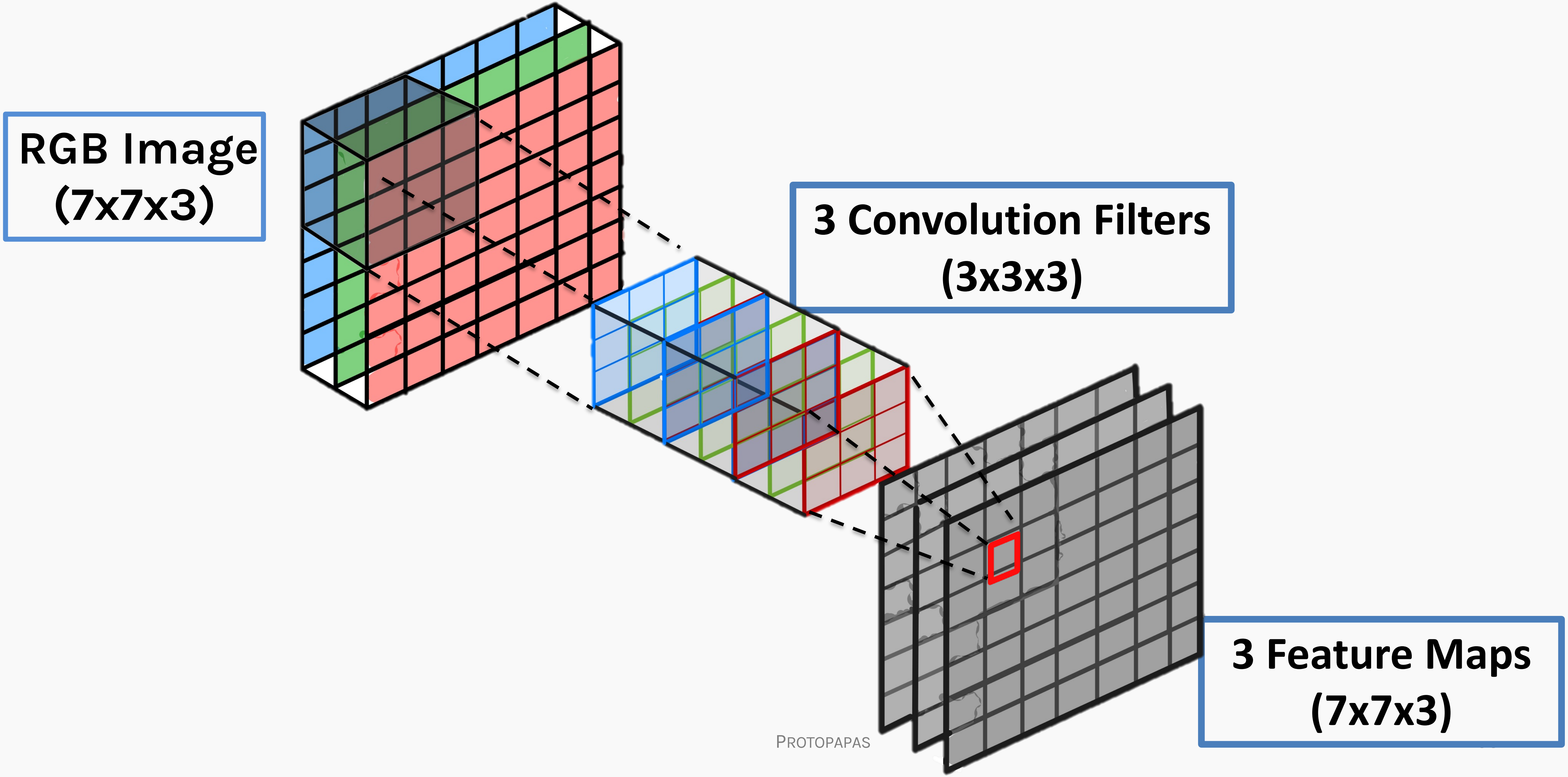
Filter 1: Horizontal Lines

Filter 2: Vertical Lines

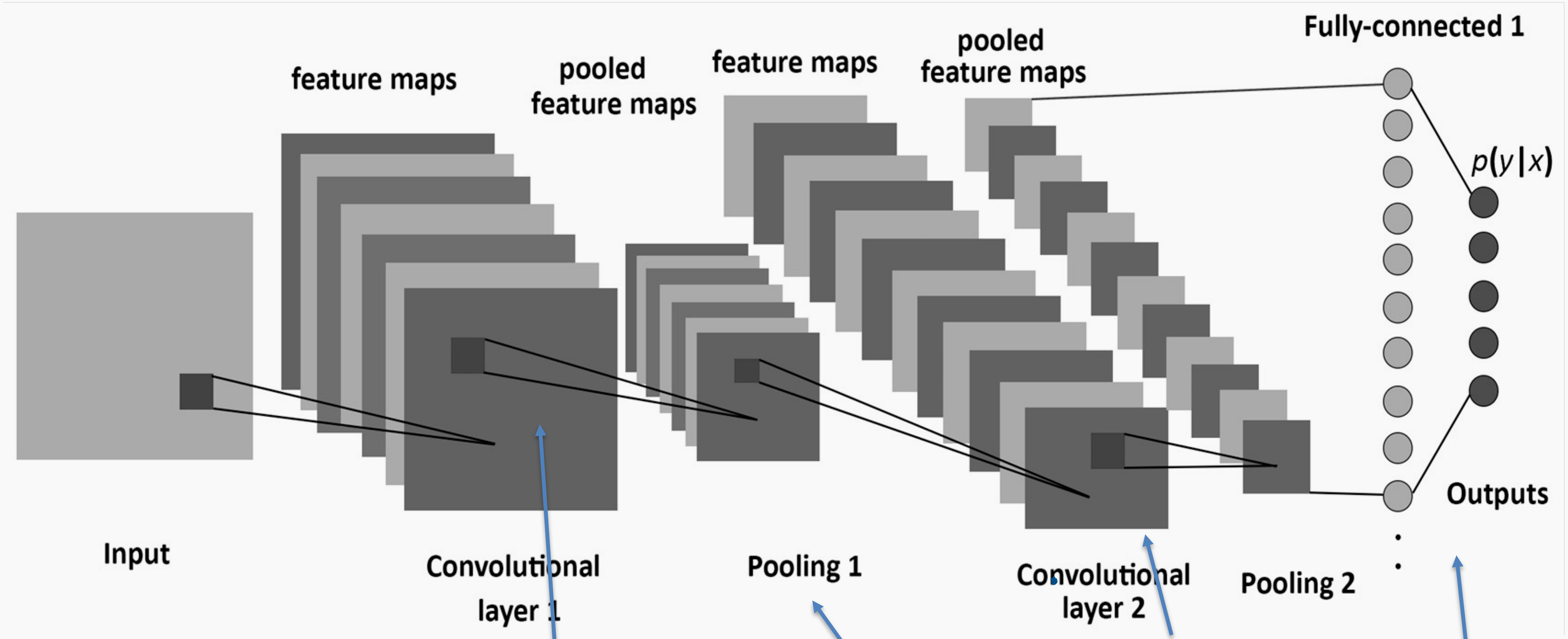
Filter 3: Orange bulb

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?

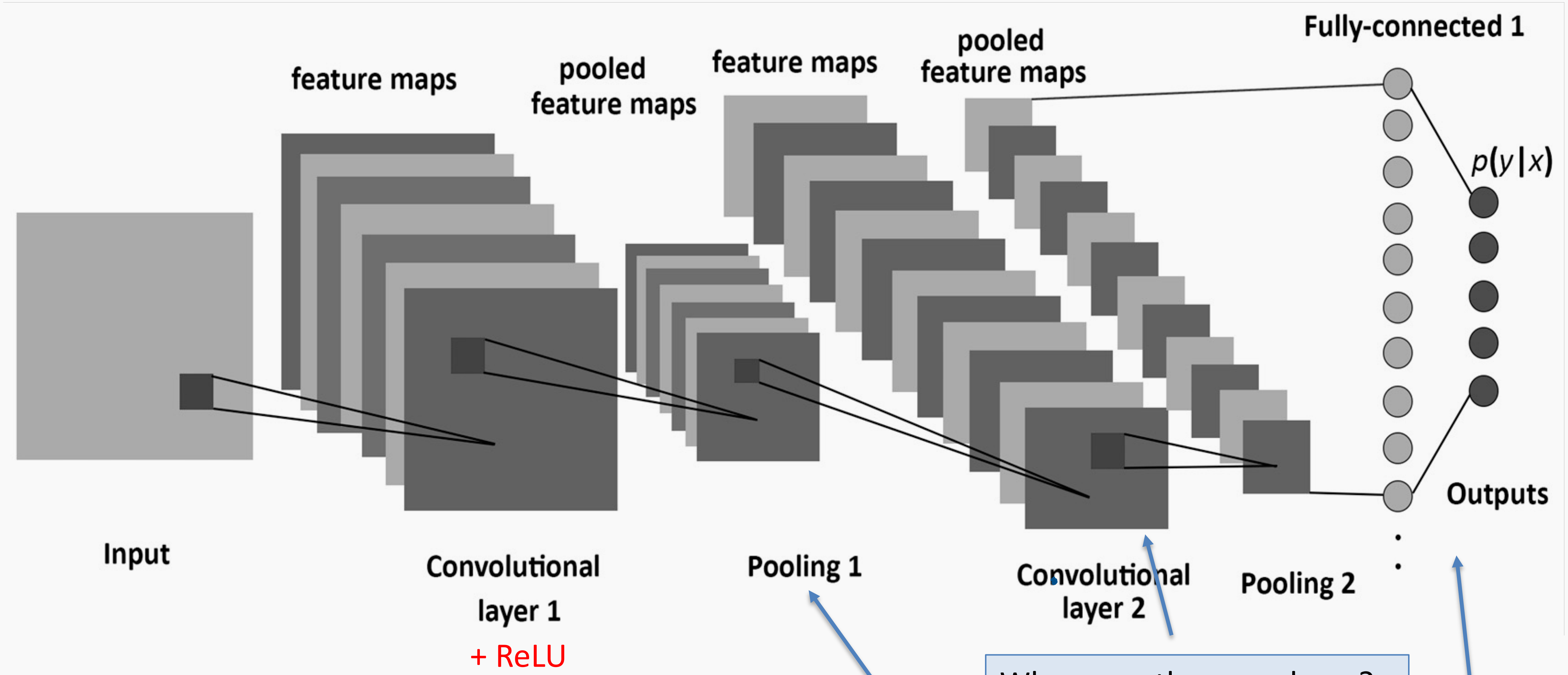
Layer 2, Filter 1: Combines <u>horizontal</u> and <u>vertical</u> lines from Layer 1 produce diagonal lines.

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



What is Pooling?

~~Why more than one layer?~~

What is going on here?

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

So far

We know that MLPs:

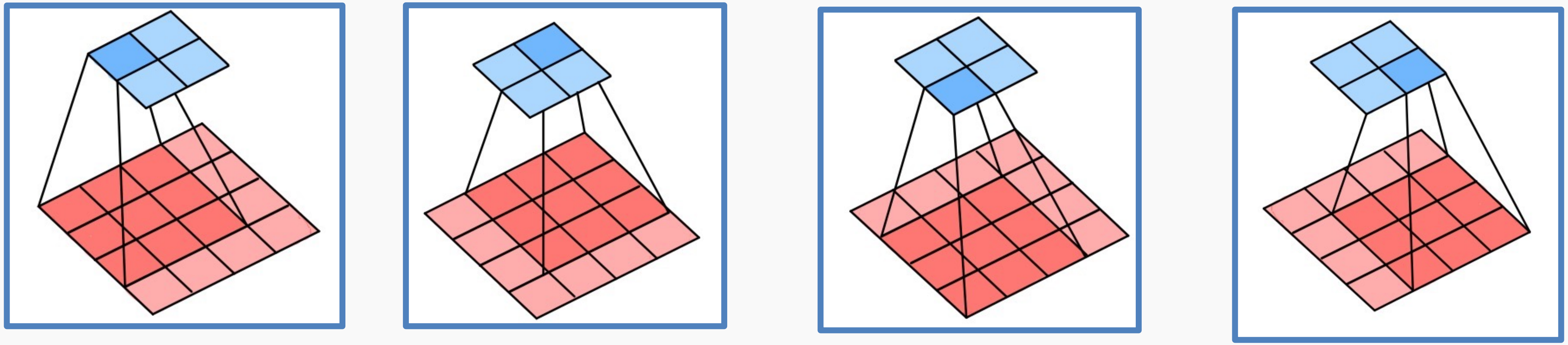
- Do not scale well for images
- Ignore the information brought by pixel position and correlation with neighbors
- Cannot handle translations

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.

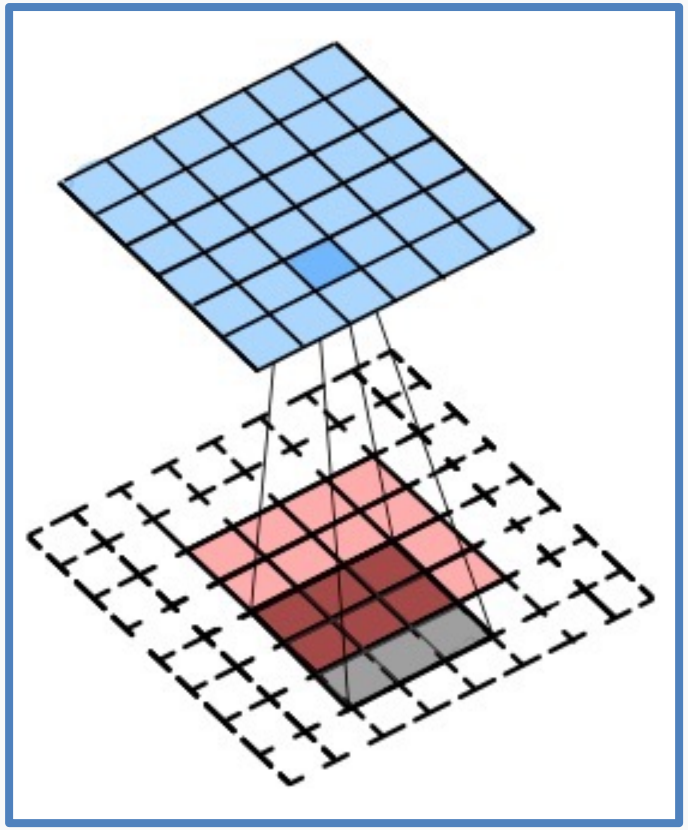
Convolutions – what happens at the edges?

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

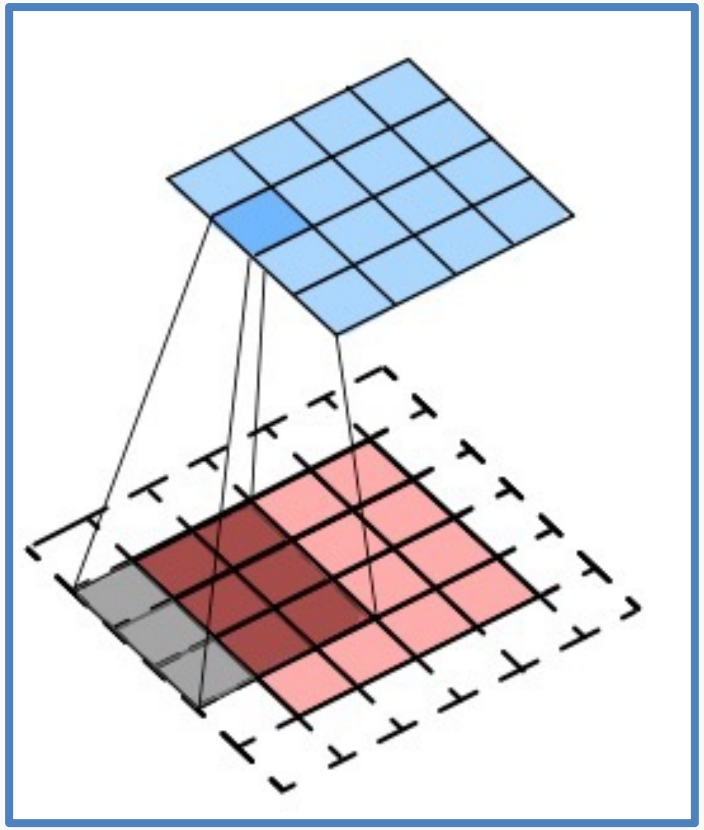


We can avoid this by padding the edges in different ways.

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

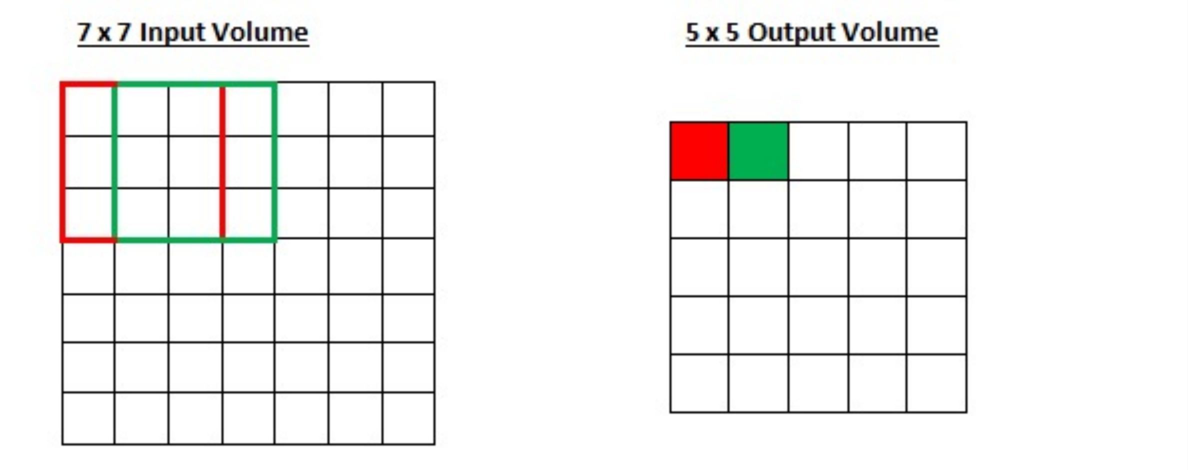
Stride controls how the filter convolves around the input volume.

The formula for calculating the output size is:

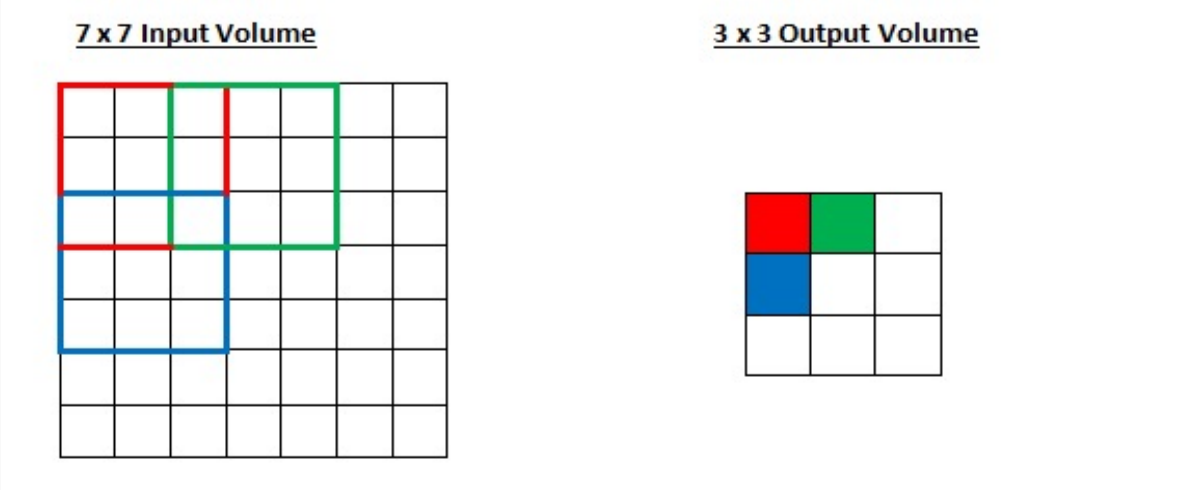
$$\boxed{O = \frac{W - K + 2P}{S} + 1}$$

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

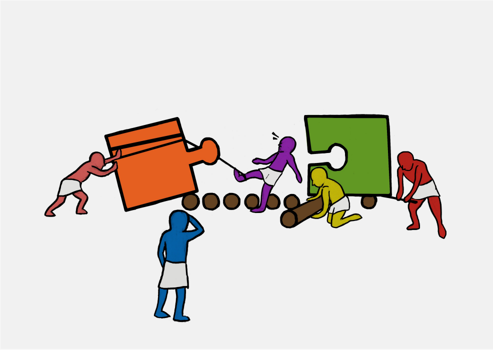
Stride = 1



Stride = 2

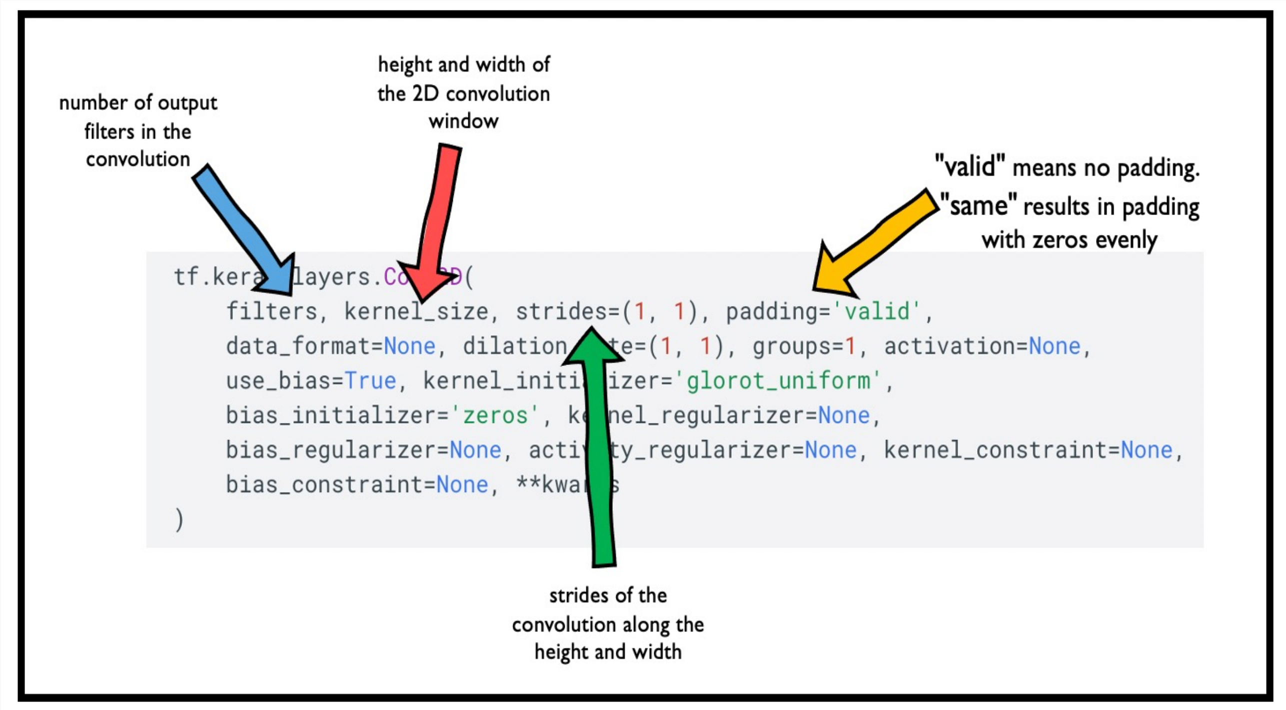


Exercise: Pavlos vs Not Pavlos

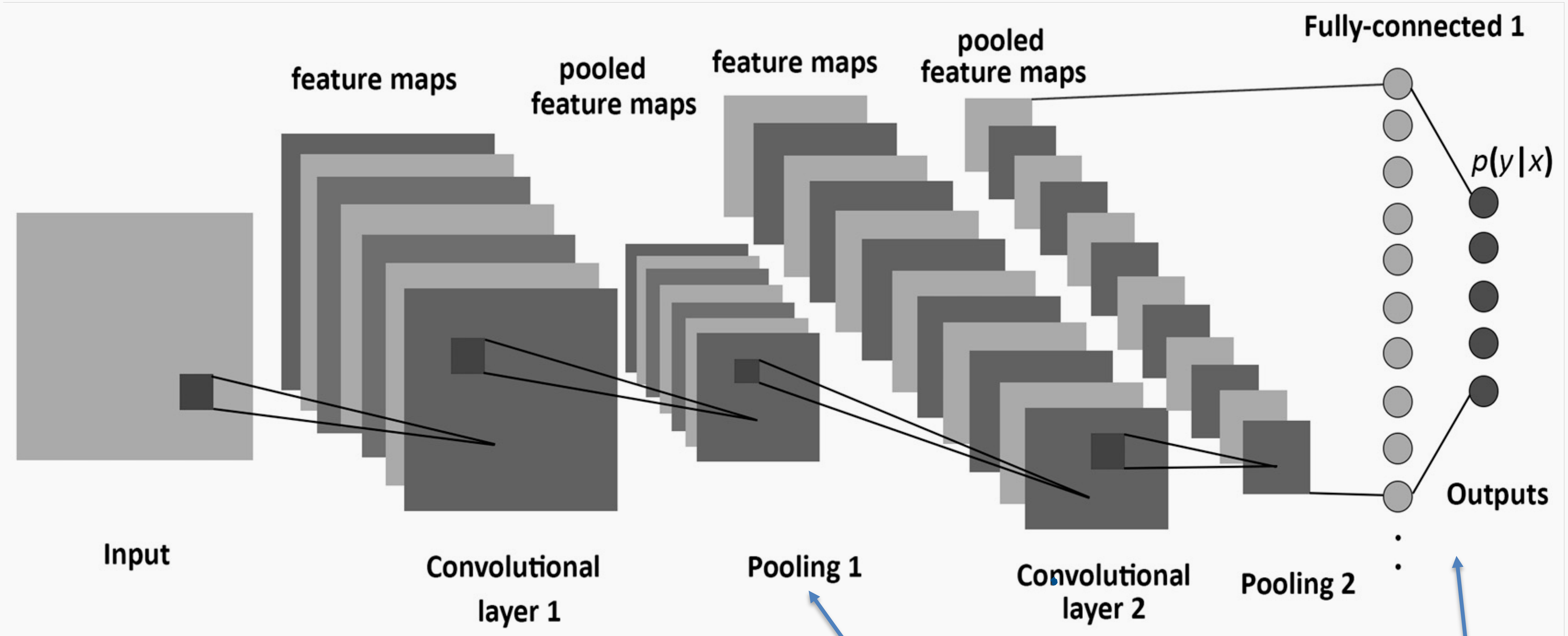


The aim of this exercise is to train a dense neural network and a CNN to compare the parameters between them

- Augment the dataset since we only have one image of Pavlos and the eagle
- Build a simple feed-forward network and train it
- Use the convolution layer to build a simple CNN and train it like the network before
- Compare performance and parameters



A Convolutional Network



+ ReLU

+ ReLU

What is Pooling?

What is going on here?

Common Question after this class

1. It is said that people often choose 3 or 5 as the size of the kernel, are there any advantages this kernel size could have?
2. Why do we need pooling?
3. Would it be possible that CNN without pooling could also have good performance? Are there any alternative?
4. How to calculate the number of parameters in a CNN?
5. Does a CNN with more convolutional layers necessarily have a better performance than a CNN with less layers?
6. Why we need a dense layer after the whole convolved layers?
7. Do CNNs have any drawbacks such as vanishing or exploding gradients like simple MLPs?