Deep Learning Course

Introduction to Convolutional Neural Networks

Alejandro Sztrajman

Vienna, Austria, November 2017

Funded by
the European Union

Index

- Linear Image Classifier
- Neural Network Representation
- Deep Neural Networks
- Convolutional Neural Networks
- CNN Example with Keras

Linear Image Classifier

Image Classification

$$f : \text{image} \rightarrow \text{category}$$

Image Classification

$f($  $) \rightarrow \text{category}$

Image Classification

$f($  $) \rightarrow \text{cat}$

Image Classification

$$f(x) = Wx + b$$

stretch pixels into single column

input image

$$\begin{array}{ccccccc} \begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \end{bmatrix} & \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} & + & \begin{bmatrix} 1.1 \\ 3.2 \end{bmatrix} & \rightarrow & \begin{bmatrix} -96.8 \\ 437.9 \end{bmatrix} & \begin{matrix} \text{cat score} \\ \text{dog score} \end{matrix} \\ \text{Weights} & \text{Input Features} & & \text{Bias} & & f(x) & \end{array}$$

Training

Dataset of
images and
annotated
categories

$$f(x_i) = Wx_i + b = y_i$$

Parameters
to optimise

Evaluation

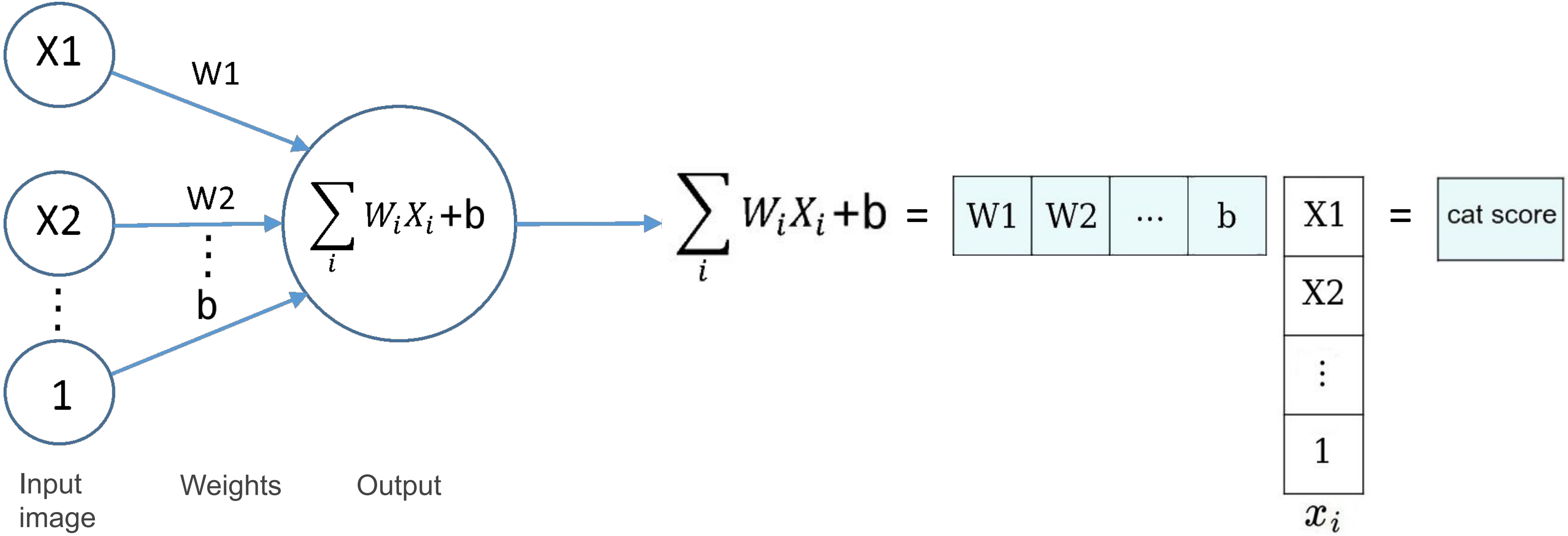
$$f(x_i) = Wx_i + b$$

Known

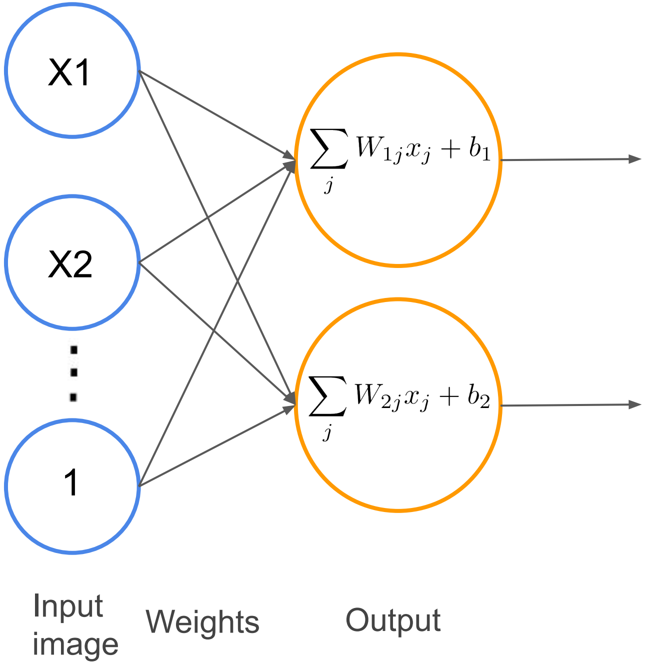
- All the weight of the computation is carried by the training. Evaluation is a simple matrix multiplication.
- Once we compute the parameters W,b we don’t need the training data anymore.

Neural Network Representation

Neuron

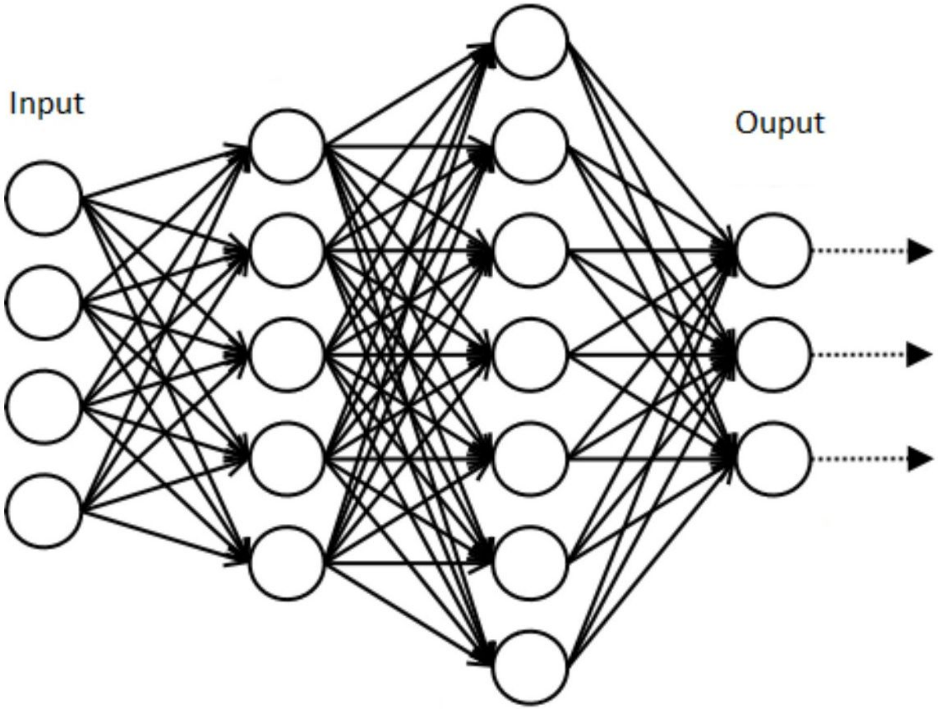

X1
W1
X2
W2
b
1
$\sum_i W_i X_i + b$
$\sum_i W_i X_i + b$ =
W1
W2
...
b
X1
X2
1
x_i
=
cat score
Input image
Weights
Output

Two neurons



$$Wx + b = \begin{bmatrix} W11 & W12 & \cdots & b1 \\ W21 & W22 & \cdots & b2 \end{bmatrix} \underbrace{\begin{bmatrix} X1 \\ X2 \\ \vdots \\ 1 \end{bmatrix}}_{x_i} = \begin{bmatrix} \text{cat score} \\ \text{dog score} \end{bmatrix}$$

Multiple neurons and layers



Input layer | Hidden Layer 1 | Hidden Layer 2 | Output Layer

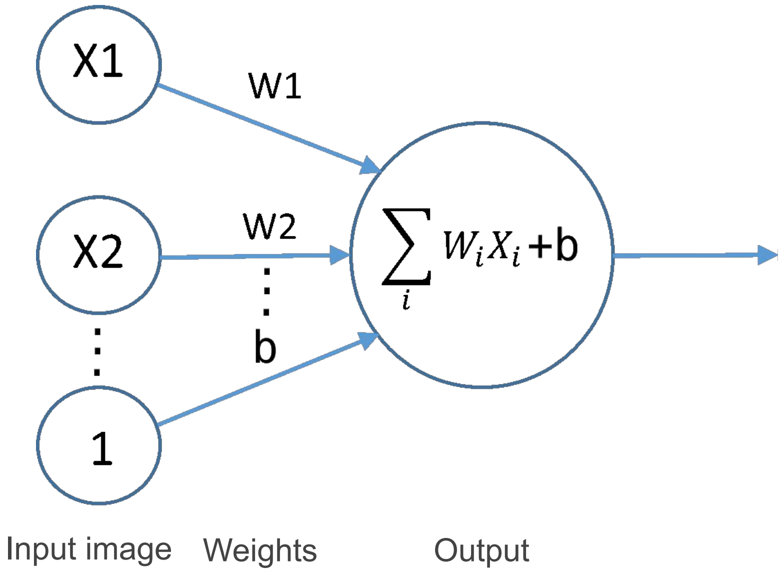
- We can keep adding neurons in each layer and this means adding more classifiers.

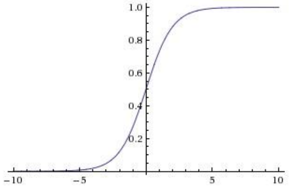
- However adding more layers of neurons does not increase the complexity of the classifier, the model is still linear.

Activation Functions*

* Based on Slides by A. Peñate Sanchez.

Neuron with activation function

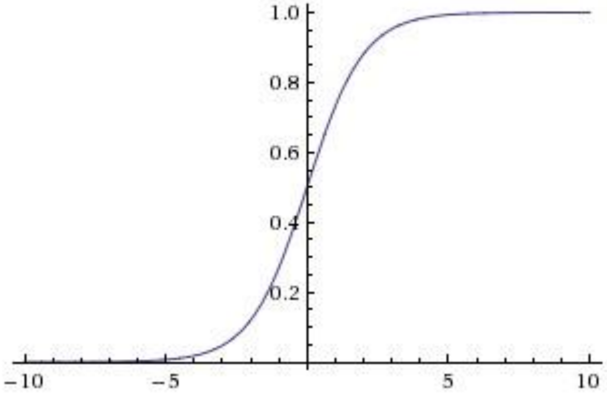




- We add a nonlinear operation applied to the output of the neuron.
- This increases the complexity of the classifier and allows the optimisation to favor certain features over others (automatic feature selection).

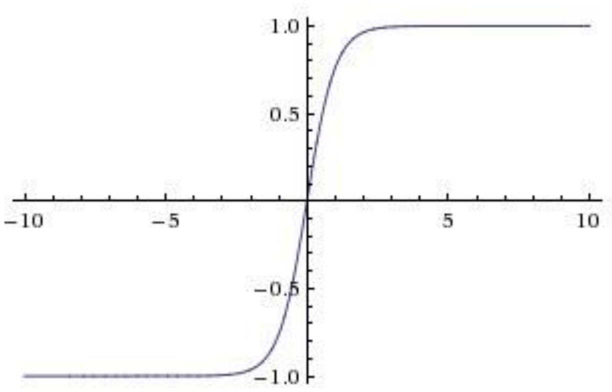
Activation functions

- Sigmoid
  - Inspired by probability theory
  - Single neuron corresponds exactly to the input-output mapping defined by logistic regression.
  - $$f(x) = \frac{1}{1 + exp(-x)}$$



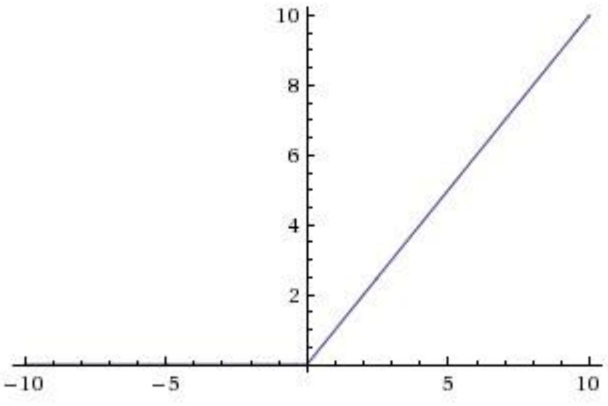
Activation functions

- Hyperbolic tangent
  - rescaled version of the sigmoid



Activation functions

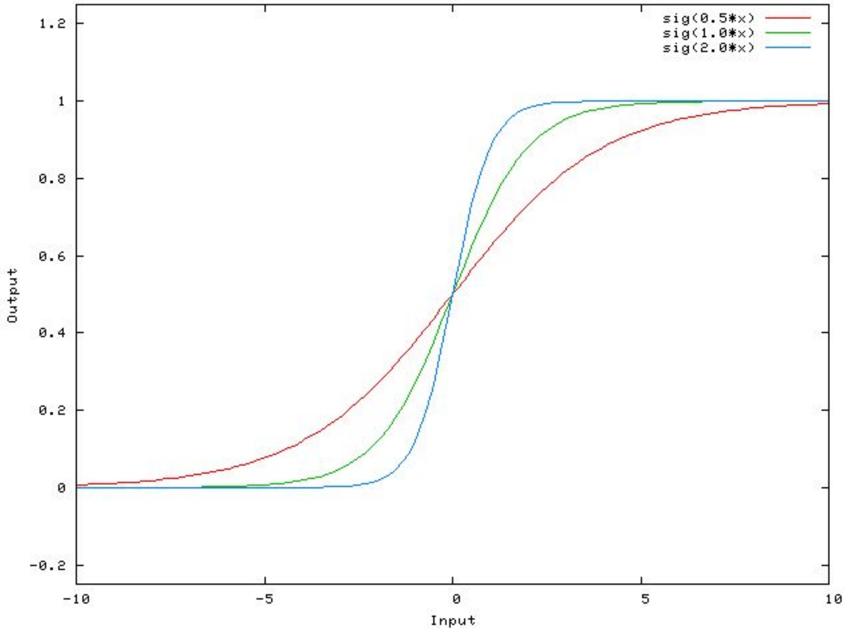
- ReLU (Rectified Linear Unit)
  - $f(x) = \max(0, x)$
    - Out = 0   if (Input of i) < 0
    - Out = linear otherwise
  - **The most popular activation function for deep neural networks**



ReLU typically learns much faster than tanh, [Glorot et al. ICAIS 2011]

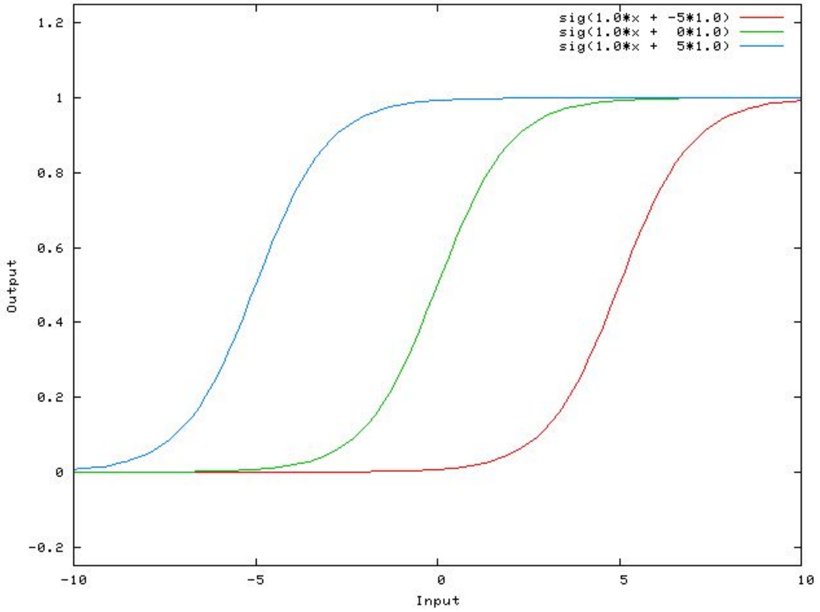
Connection Weights

- Effect of different weights in a neuron connection (sigmoid activation)



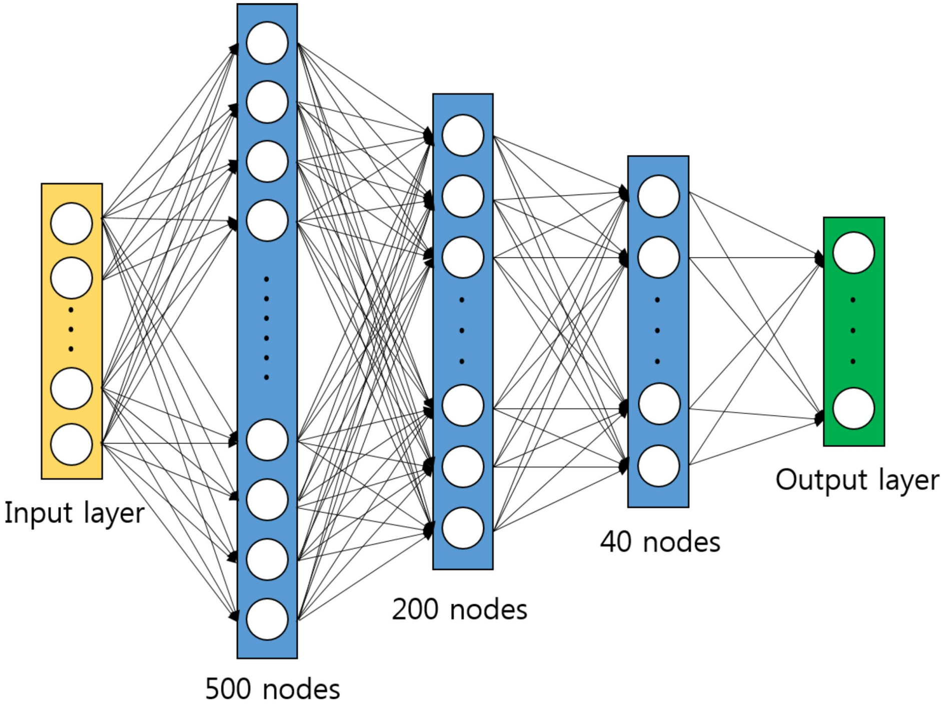
BIAS

- The weights can change the shape of the activation function, the BIAS shifts the function activation response.



- By adjusting the weights and the BIAS a single neuron can represent many different activation functions

Fully Connected Deep Neural Networks

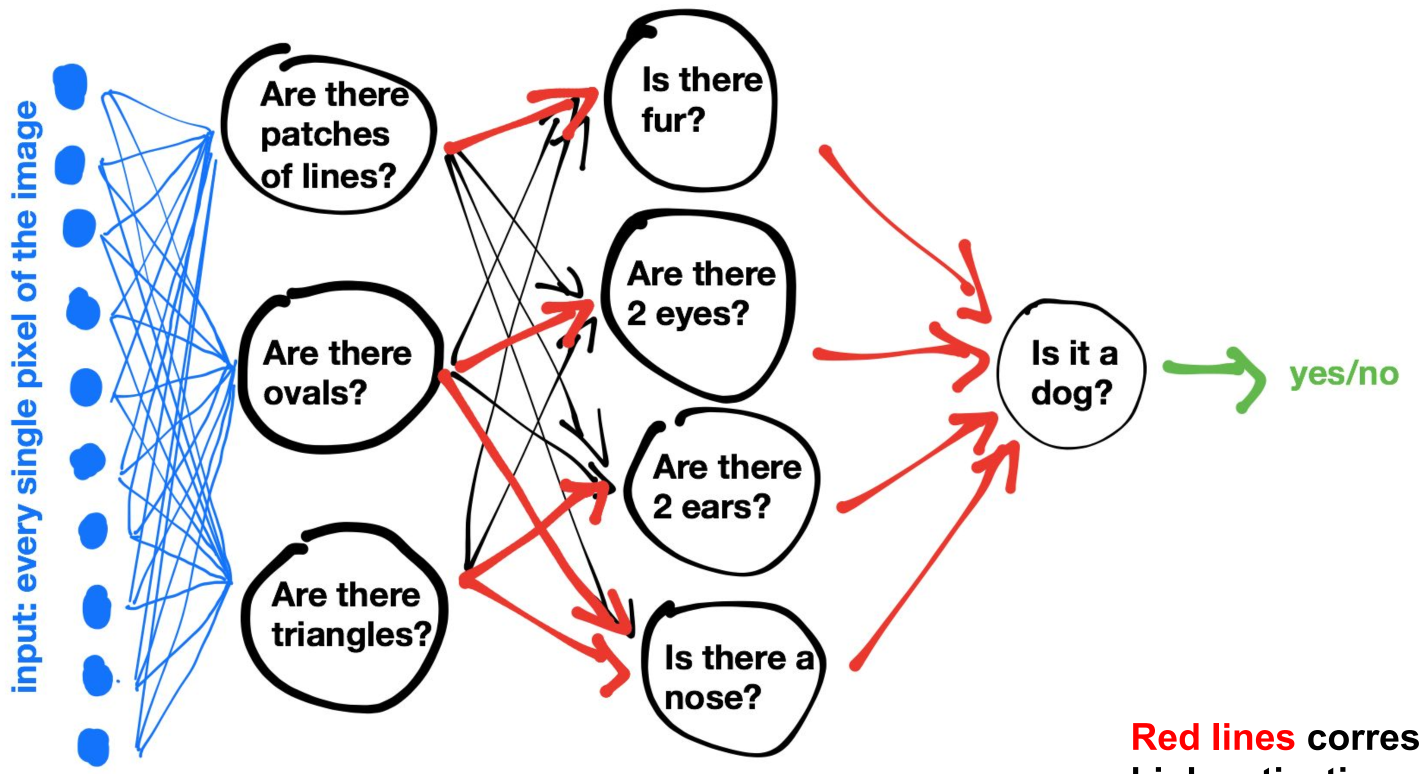


3 Hidden layers

- The final result is a composition of non linear functions applied to the features (pixels).

- The intermediate values that emerge as outputs of one layer of neurons are used as new input features (of increasing complexity) for the next layer.

Fully Connected Deep Neural Networks



Red lines correspond to high activation values.

Convolutional Neural Networks (CNNs)

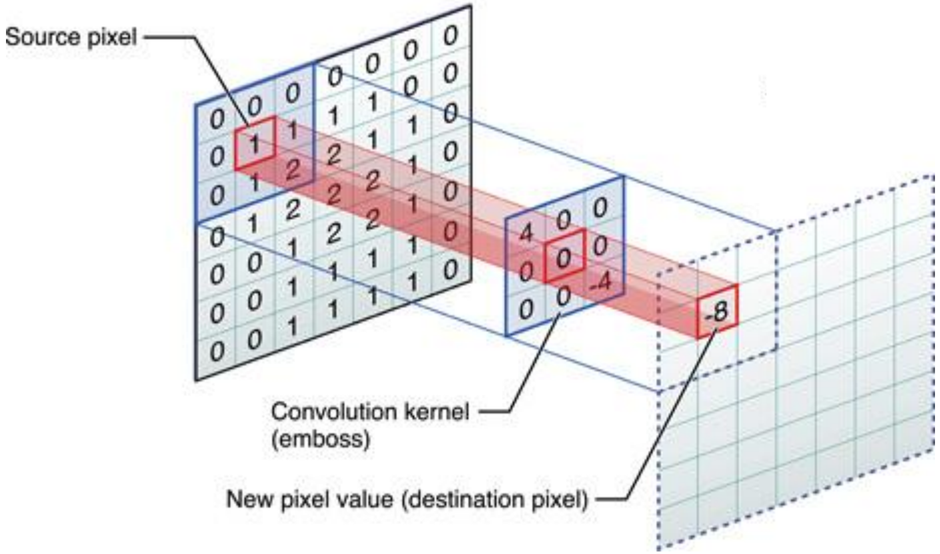
Neural Networks on Images - Problem

So far, the design choice has been to **fully connect** all the hidden units to all the input units.

With images of size 96x96 this start to become *unfeasible*:
Learning features that span the entire image is very computationally expensive

Solution:
<u>Locally Connected Networks</u>
Restrict the connections between the hidden units and the input units

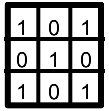
Convolutional Layer



- We leave the input images as 3D arrays (links channels of the same pixel).
- The convolutional layer is composed of a set of kernels which are convolved with the input image.
- Each kernel filters localised features in the input image. The size of the kernels determines the size of the features that can be learned.

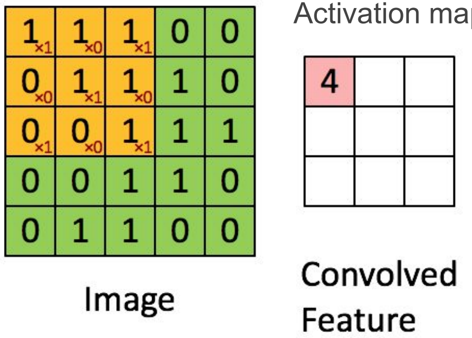
Convolutional Layer

Operation: convolution



Convolutional Kernel

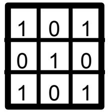
Activation map



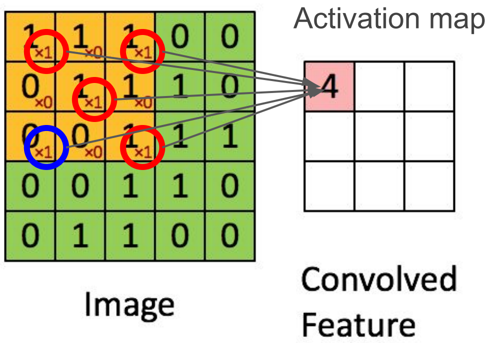
- The weights of the kernels are learned as part of the optimisation (feature selection).
- The output of the conv layer is a stack of 2D activation maps.

Convolutional Layer

Operation: convolution



Convolutional Kernel

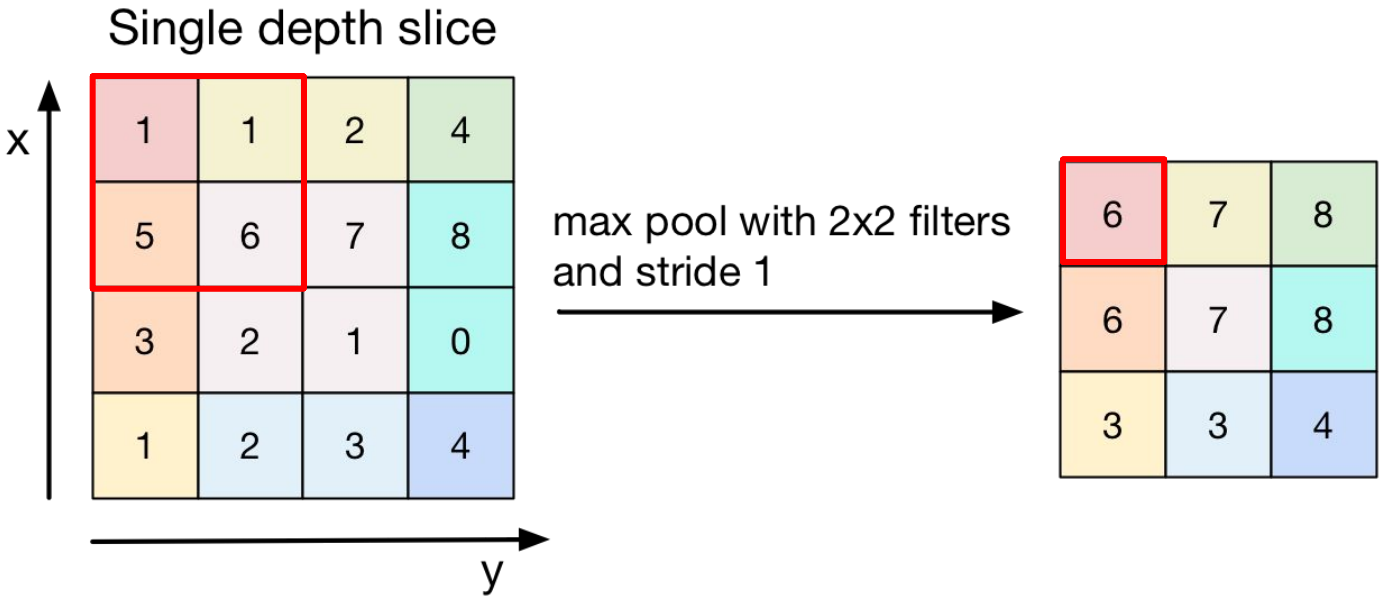


- The weights of the kernels are learned as part of the optimisation (feature selection).
- The output of the conv layer is a stack of 2D activation maps.

Pooling Layer

Goal: progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

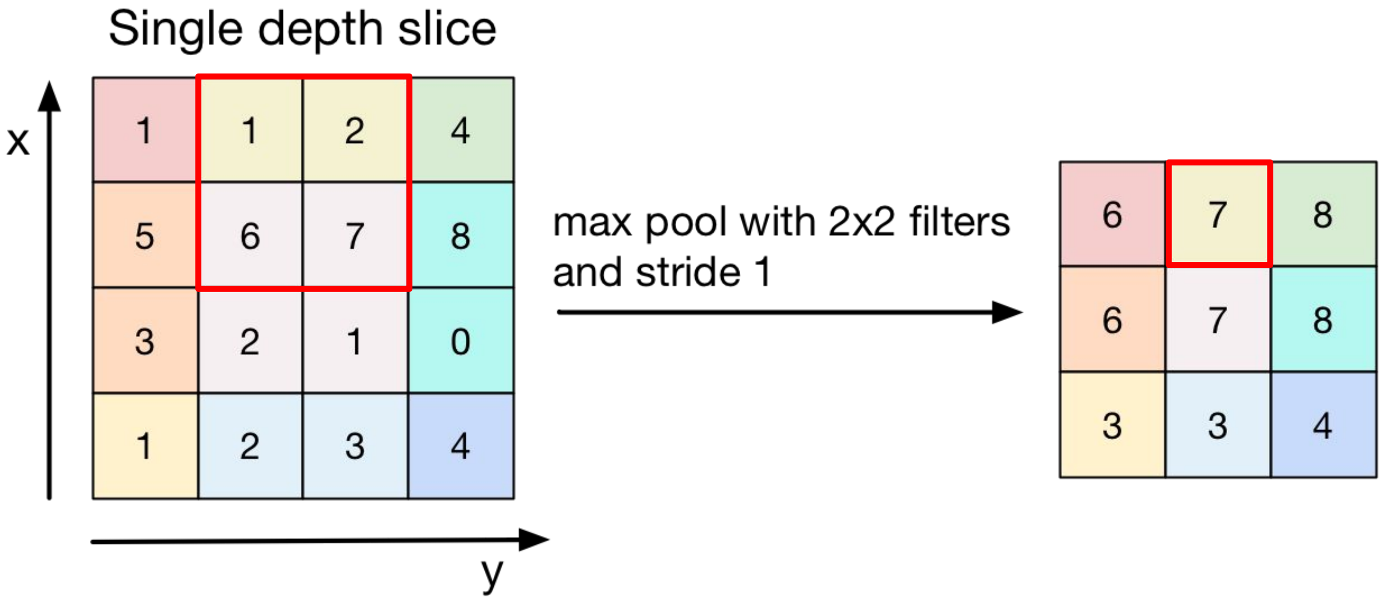
- Max pooling
- Min pooling
- Average pooling



Pooling Layer

Goal: progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

- Max pooling
- Min pooling
- Average pooling



Pooling Layer

If the pooled regions are contiguous areas in the input image, then the pooling units will be **translation invariant**, something necessary:

we want the classifier to still accurately classify objects regardless of their relative position in the image frame.

CONV
RELU
CONV
RELU
POOL
CONV
RELU
CONV
RELU
POOL
CONV
RELU
CONV
RELU
POOL
FC
car
truck
airplane
ship
horse

Dropout Layer

Technique used only in the training phase, addressing the **overfitting problem**.

Idea:

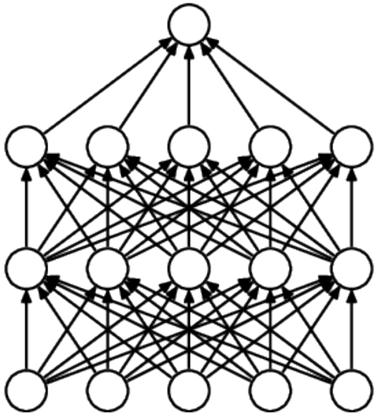
- Randomly drop units during training

Drop out means **temporarily** removing the unit from the network along with all its incoming and outgoing connections

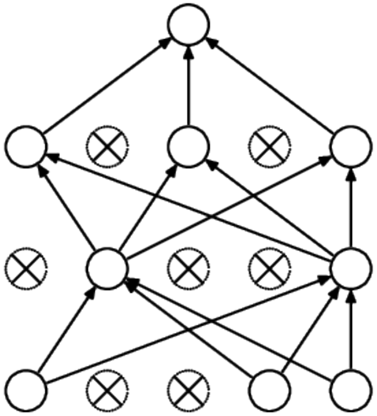
Dropout Layer

Example on fully connected layers

Each unit is retained with a fixed probability p **independent** of other units



(a) Standard Neural Net



(b) After applying dropout.

In addition to prevent overfitting, it also provides a way of approximately **combining** exponentially many **different neural network architectures** efficiently.

A bit of code:
simple example with Keras