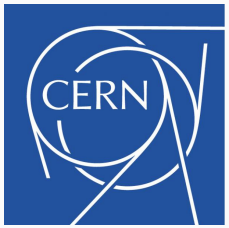
A practical approach to Convolutional Neural Networks

Daniel Hugo Cámpora Pérez

inverted CERN School of Computing, Mar 5th - 7th, 2019

Universidad de Sevilla
CERN



Outline

Introduction

Some key ANN concepts

Convolutional neural networks

Overview of historically important networks

Cifar-10 example

RICH reconstruction example

Bibliography

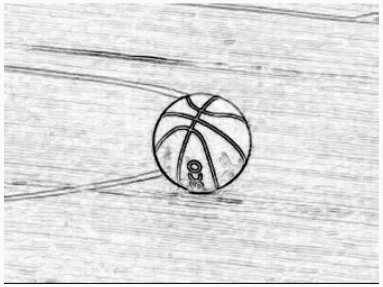
Introduction

Some concepts and history

Stages of Visual Representation, David Marr, 1970s

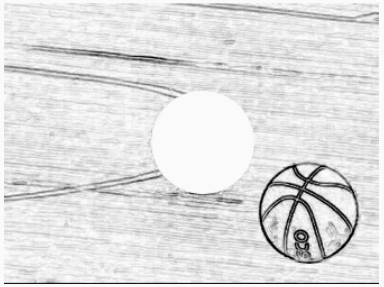
Input image

Perceived intensities



Edge image

Zero crossings, blobs, edges, bars, ends, virtual lines, groups, curves, boundaries



2½D model

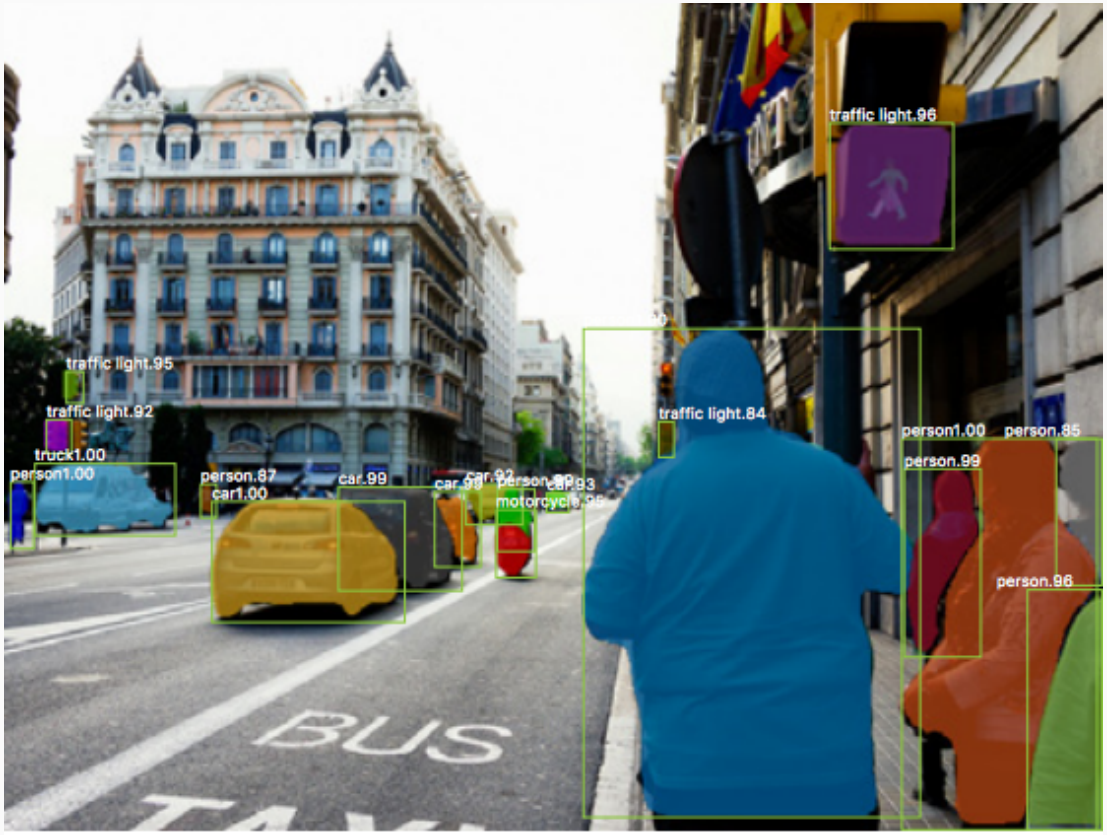
Local surface orientation and discontinuities in depth and in surface orientation

3D model

3D models hierarchically organized in terms of surface and volumetric primitives

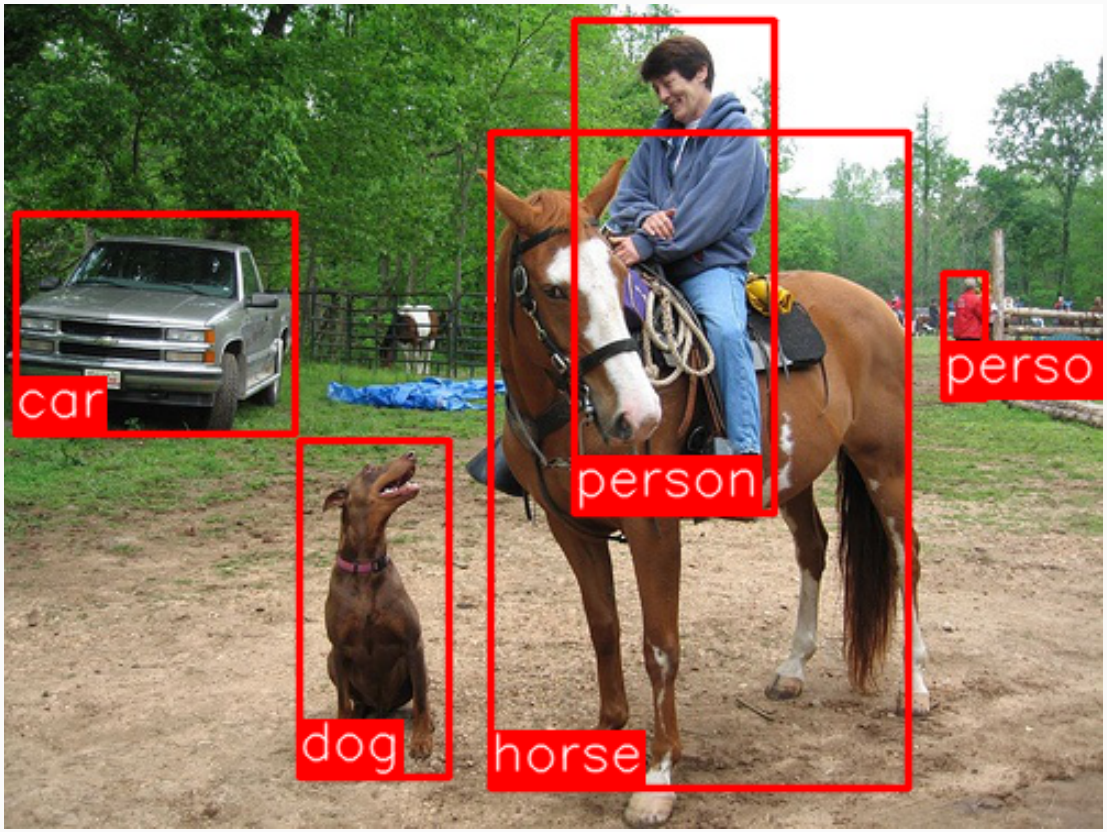
Semantic segmentation

In semantic segmentation, our goal is to classify the different objects in the image, and identify their boundaries.



Object detection

In object detection, the objective is to find and classify the objects in an image.



Face detection

Face Detection, Viola & Jones, 2001

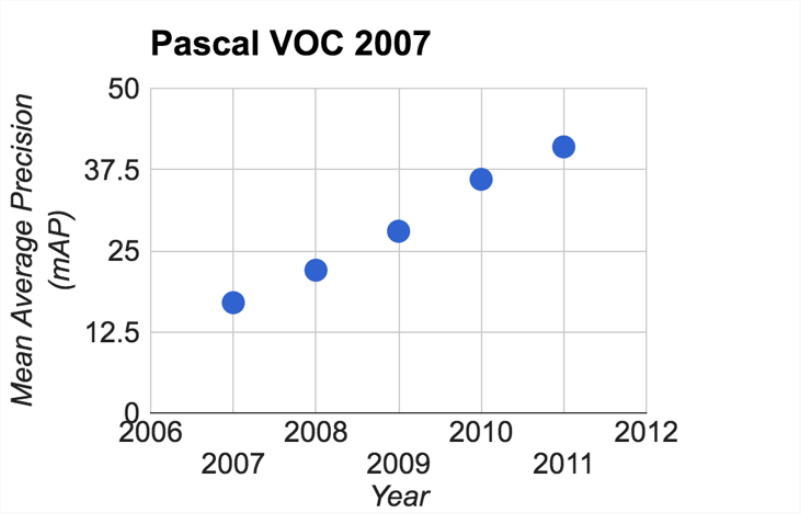
Object recognition

Object Recognition from Local Scale-Invariant Features, David Lowe, 1999

Having 3D representations of objects, said objects are recognized in a variety of scenarios, including scenarios with occlusion.

PASCAL Visual Object Challenge (VOC)

Object from 20 categories. Success is measured by how many were correctly classified.



ImageNet challenge

The ImageNet challenge was created in 2009. Similarly to the VOC, images and categories are presented. However, 1.4M images with 1k object classes are presented. A classification is successful if the sought category is in the first 5 categories outputted by the algorithm.

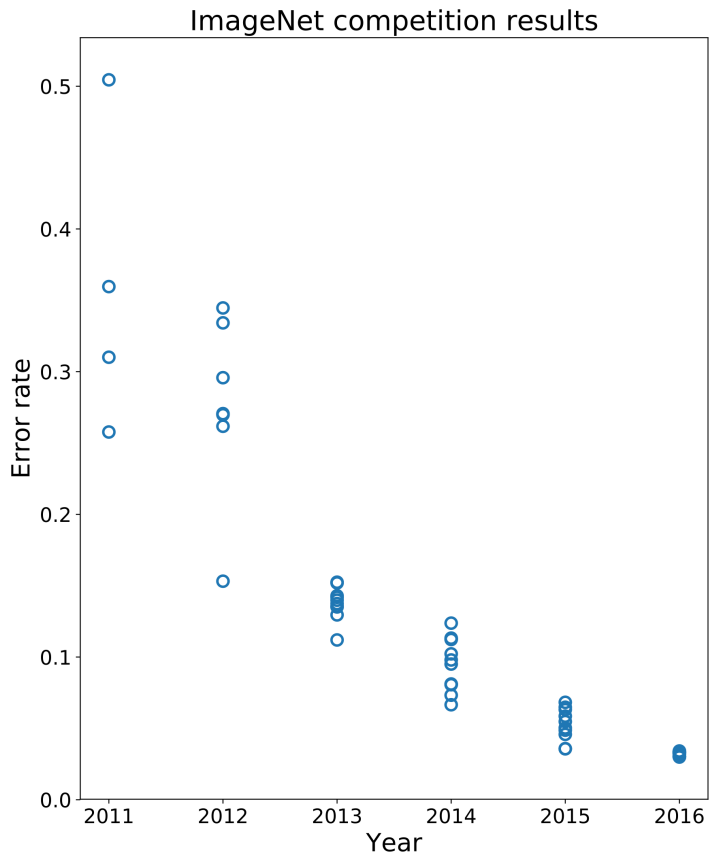
For instance, the following classification would be correct:

- Scale
- T-shirt
- Steel drum
- Drumstick
- Mud turtle

ImageNet challenge (2)

In 2012, the Convolutional Neural Network named AlexNet got 10% less error rate than any of its competitors.

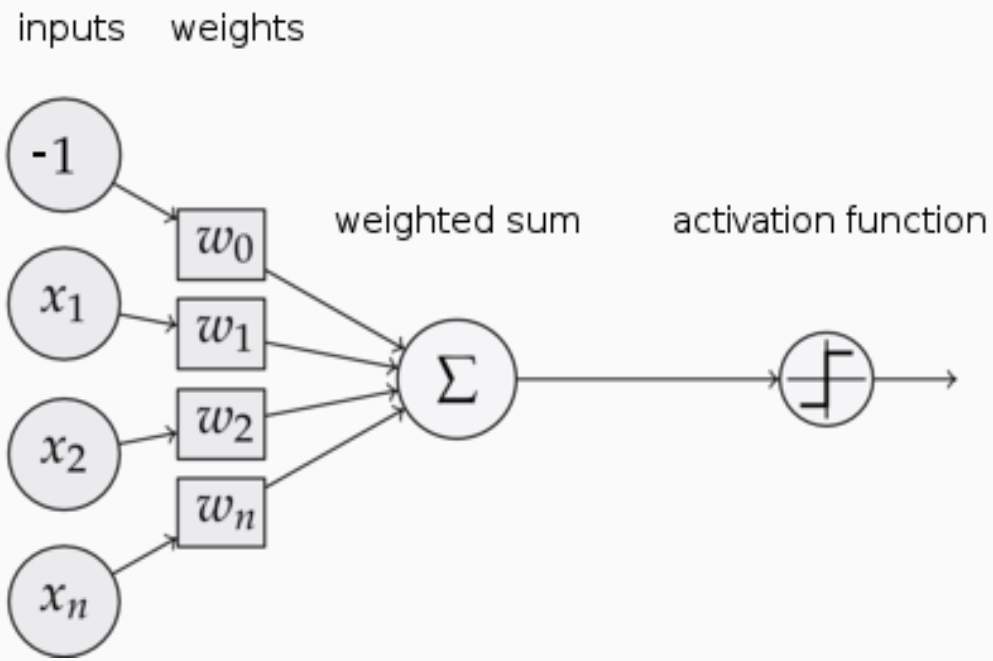
Recently (2015), the ImageNet challenge has been solved with an error lower to that of humans!



Some key ANN concepts

The perceptron

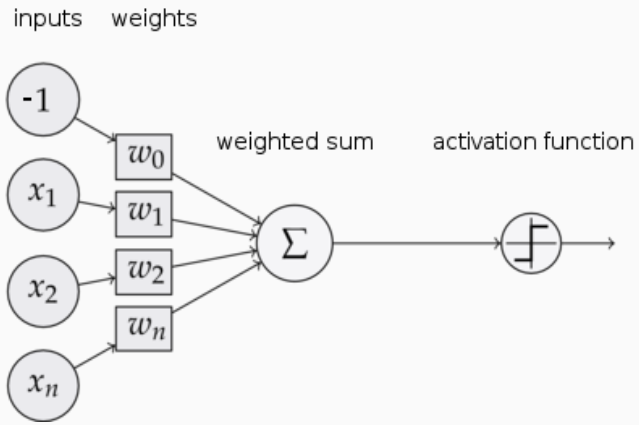
You can think of a perceptron as a mathematical model, *inspired* in a single neuron.



The perceptron (2)

It consists of:

- Inputs $-1, x_1, x_2, \ldots, x_n$
- Weights $w_0, w_1, w_2, \ldots, w_n$
- Activation function $f(x)$



With these ingredients, the output of the perceptron is calculated as follows,

$$y = f(\sum_{j=0}^{n} w_j x_j) \tag{1}$$

Activation functions

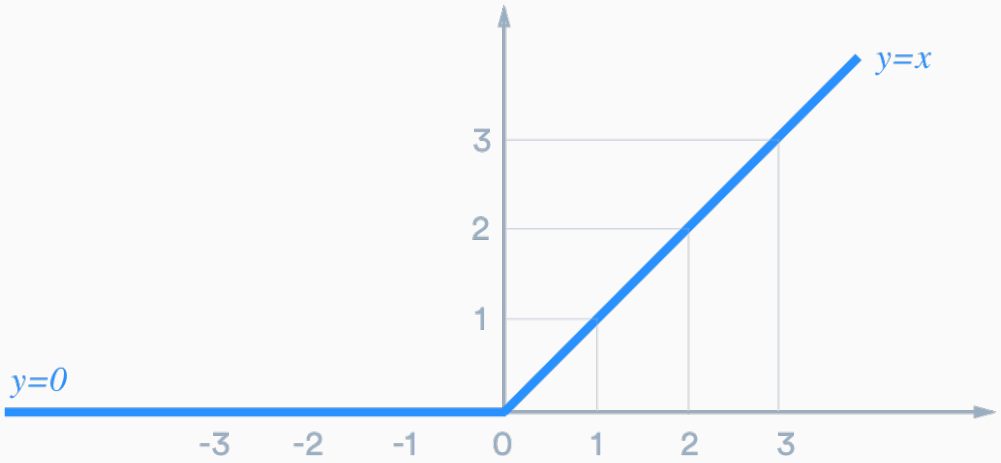
The activation function $f(x)$ defines the relation between the states of the neighbouring neurons and the state of the neuron under study. It must be chosen according to the nature of the state of the neuron.

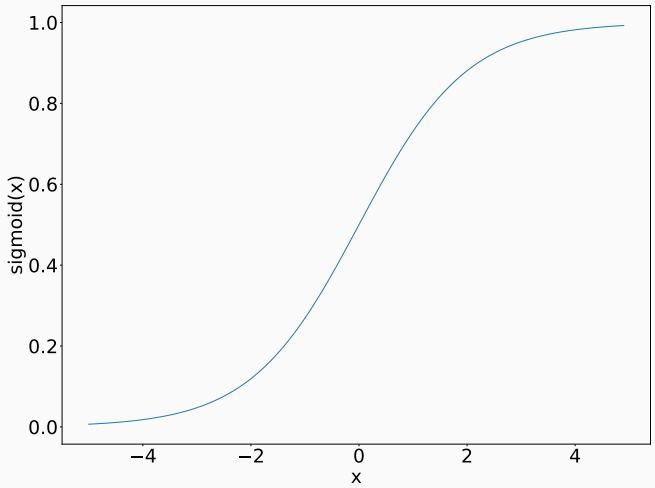
Here are some typical activation functions:

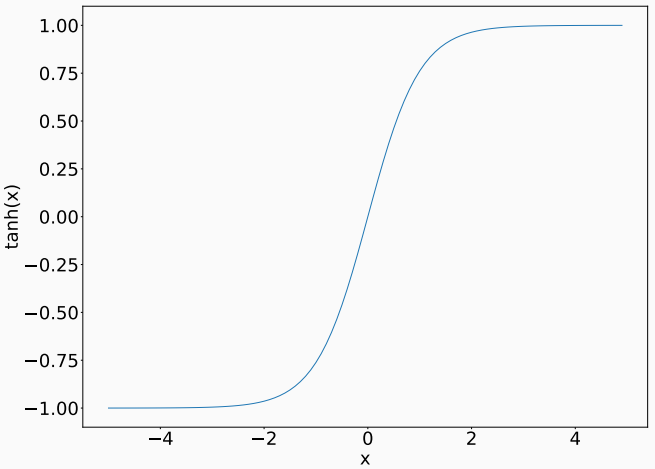
- $R(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$
- $\sigma(x) = \dfrac{1}{1 + \mathrm{e}^{-x}}$
- $tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

Visually

The Rectified Linear Unit function is widely used, as it is very cheap to compute. Sigmoid (σ) and hyperbolic tangent (*tanh*) are also pretty common.







Differentiability

It's useful to have an activation function that is differentiable to let our network learn with the backpropagation algorithm. ReLU (R), the sigmoid (σ) and hyperbolic tangent ($tanh$) are indeed differentiable,

$$R'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{2}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{3}$$

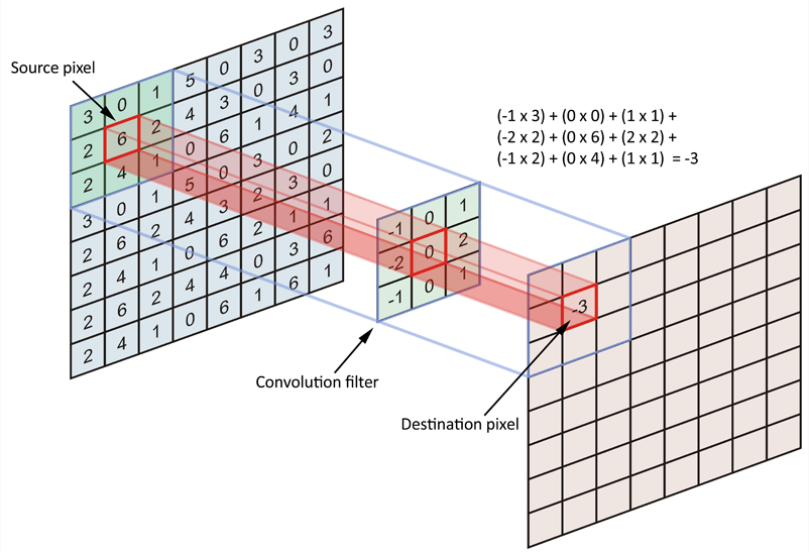
$$tanh'(x) = 1 - tanh^2(x) \tag{4}$$

Convolutional neural networks

What is a convolution?

An image convolution is a transformation pixel by pixel, done by applying to an image some transformation defined by a set of weights, also known as a *filter*. Let s be a set of source pixels, and w a set of weights, a pixel y is transformed as follows:

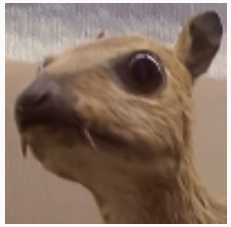
$$y = \sum_{i=0}^{n} s_i w_i \tag{5}$$



Examples of filters

Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

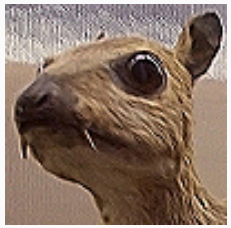


Edge detection

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

Sharpen

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



Blur

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Architecture overview

Regular neural networks don't scale well to full images. To see this, let's have a look at a subset of the CIFAR-10 dataset:

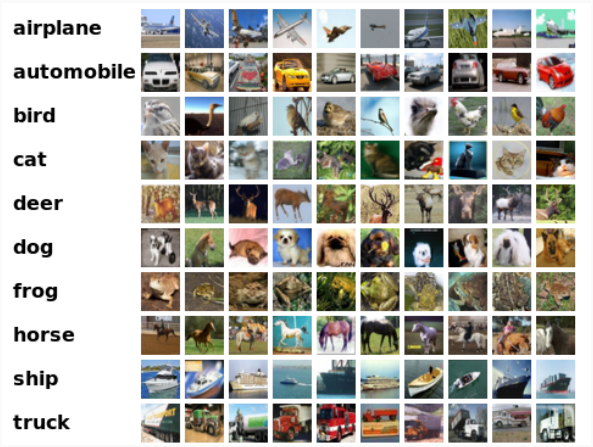
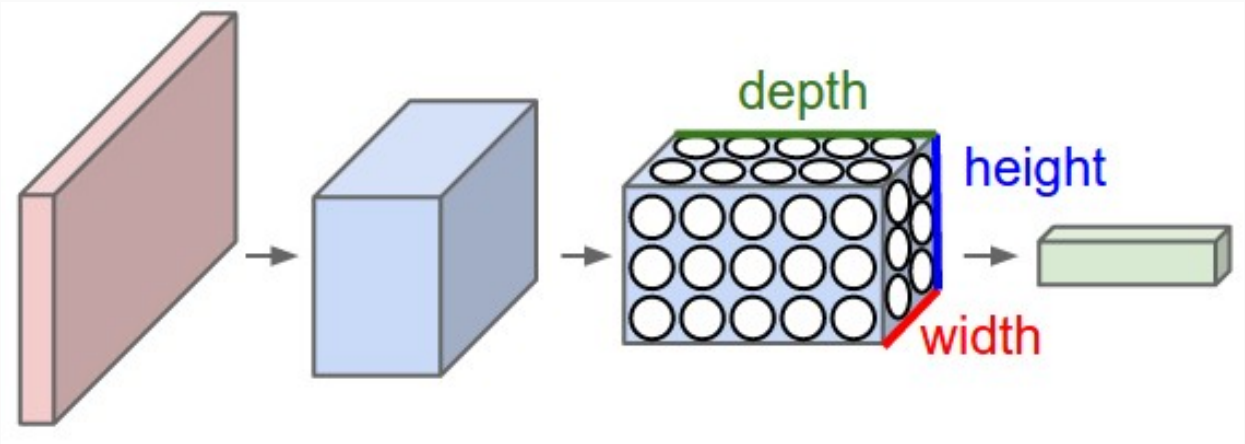


Figure 1: A subset of the CIFAR-10 dataset.
(https://www.cs.toronto.edu/~kriz/cifar.html)

This dataset is composed of images of size 32x32x3 (width x height x color channels), categorized to what they contain. Note even though these are 2D images, due to the color channels, we are dealing with volumes.

Architecture overview (2)

If we have a perceptron connect to all the inputs, this would imply $32 * 32 * 3 = 3072$ weights. This number quickly runs out of hand when scaling to larger images.



Usually, Convolutional Neural Networks deal with this problem by using a feedforward network, and having **local connectivity** between the layers, that is, we will connect each neuron to only a local region of the input volume.

Types of layers

We will look at three basic kinds of layers:

- Convolutional layers
- Pooling layers
- Fully-connected layers

To describe these layers we will refer to their connectivity and three dimensions: width, height and depth.

Convolutional layers

A convolutional layer consists of a set of learnable filters. Every filter is spatially *small* in width and height, but extends through the full depth of the input volume.

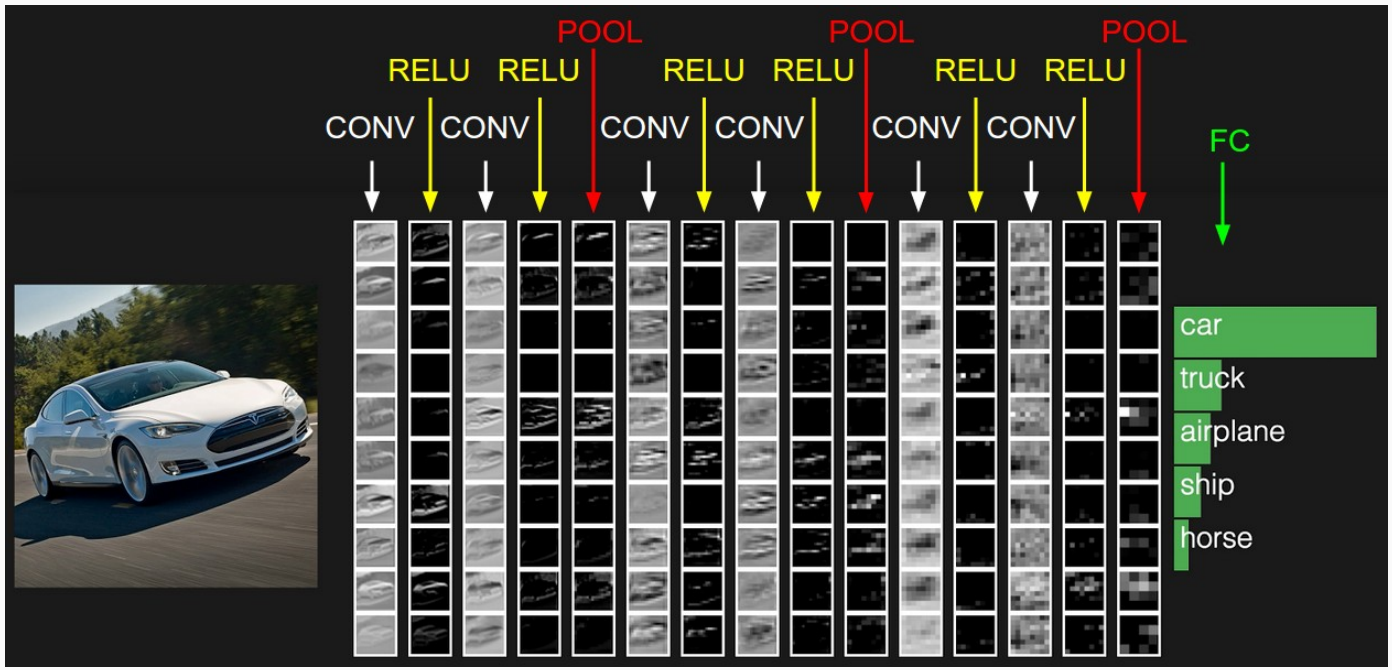
- Learnable filters are applied to **local regions**
- A weight *activation map* determines the impact of the filter
- Weights can be shared across activation functions to reduce the number of learnable parameters
- Depth (or channels) conceptually will learn different *features*
- Translation equivariant: If the input changes, the output changes in the same way

Convolutional layer examples

- Example 1 (http://cs231n.github.io/assets/conv-demo/index.html)
- Example 2 (https://github.com/vdumoulin/conv_arithmetic)
- Example 3 (https://thomaskeck.github.io/talks/DeepLearning.html#/2/6)

Learnt filters

Intuitively, the network will learn filters that activate when they see some type of **visual feature**, like an edge or a blotch of some color on the first layer, or eventually honeycomb or wheel-like patterns on higher layers of the network.



Convolutional layers spatial arrangement

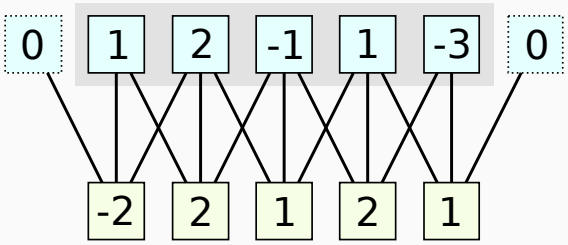
Three parameters control the size of the output volume:

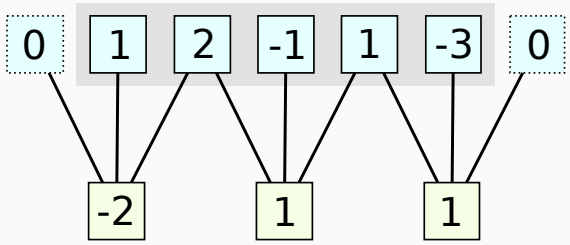
- **Depth** Number of filters we would like to use, each searching for a different characteristic.
- **Stride** Determines how the filter is slid across the input. It is uncommon to slide more than three pixels at a time.
- **Zero-padding** Determines the number of pixels filled with zeroes around the border. Sometimes, this is useful to allow certain strides.

We can compute the spatial size of the output volume as a function of the input volume size W, the filter size F, stride S and padding P as:

$$\frac{W - F + 2P}{S} + 1 \tag{6}$$

Example: Convolutional layers spatial arrangement





- **Left** settings: size $W = 5$, padding $P = 1$, stride $S = 1$, filter size $F = 3$. The expected size of the output volume is therefore,

$$\frac{5 - 3 + 2 \cdot 1}{1} + 1 = 5 \tag{7}$$

- **Right** settings: $W = 5$, $P = 1$, $S = 2$, $F = 3$,

$$\frac{5 - 3 + 2 \cdot 1}{2} + 1 = 3 \tag{8}$$

Convolutional layers: Parameters

For each convolutional layer in our network, there will be associated weights or parameters to each of the neurons in the layer. For a layer $l-1$ connected to layer l, with depths D_{l-1} and D_l respectively, and filter size F, the number of parameters required for a convolutional layer is:

$$\begin{aligned}\texttt{parameters} &= \texttt{connection weights} + \texttt{bias weights} \\ &= F \times F \times D_{l-1} \times D_l + D_l\end{aligned}$$

Convolutional layers parameter sharing

Following the assumption that if one feature is useful to compute at some spatial position (x_0, y_0), then it should also be useful to compute at a different position (x_1, y_1), it is possible to reduce dramatically the number of parameters (weights) in a CNN.

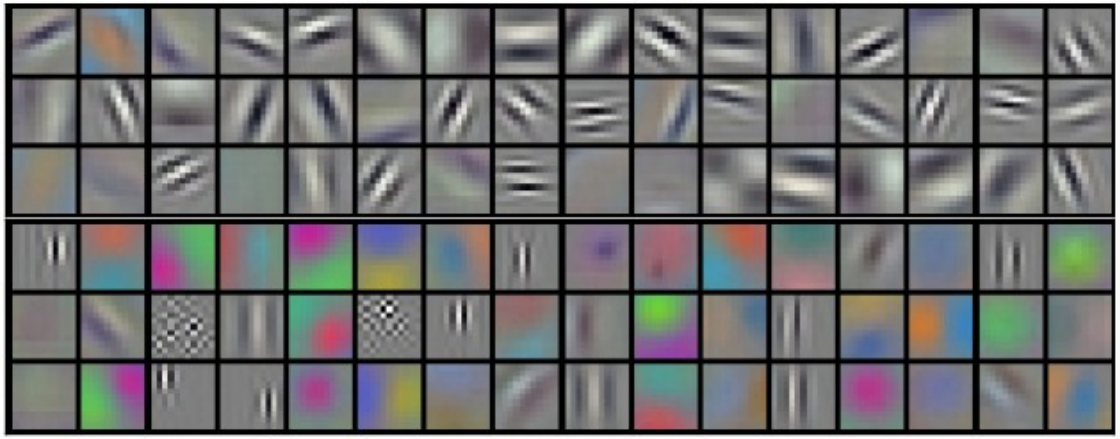


Figure 2: Example filters learnt by Krizhevsky *et al.* [3], who won the ImageNet challenge (`http://www.image-net.org/challenges/LSVRC`) in 2012. The first convolutional layer had a size of $55 * 55 * 96 = 290\,400$ neurons with $11 * 11 * 3 + 1 = 364$ inputs each. This makes for $105\,705\,600$ weights, only feasible due to parameter sharing.

Pooling layers

The function of a pooling layer, also known as a *subsampling* layer, is to progressively reduce the spatial size of the representation, to reduce the amount of parameters and computation in the network. The pooling is typically done using the *average* or *maximum* function, applied to the subset in consideration.

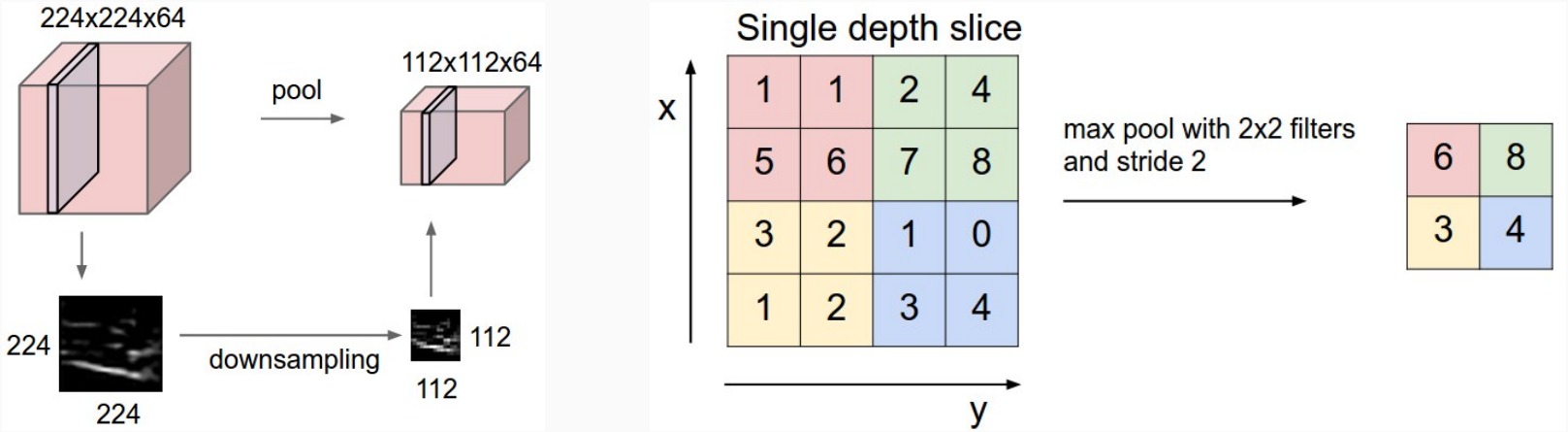


Figure 3: Pooling using the maximum function, with 2x2 filters and stride 2.

Other types of layers

- **Normalization layers** Many types of normalization layers have been proposed, for instance with the intention of implementing inhibition schemes observed in the biological brain. BatchNorm is a popular one.
- **Fully-Connected layers** Neurons in a fully-connected layer have full connections to all activations in the previous layer, as seen in regular neural networks. They are used in the output layers of CNNs.
- **Dropout layers** Dropout is a regularization technique to reduce overfitting. It consists in randomly connecting only a subset of neurons in backpropagation steps. Example (`https://thomaskeck.github.io/talks/DeepLearning.html#/2/10`)

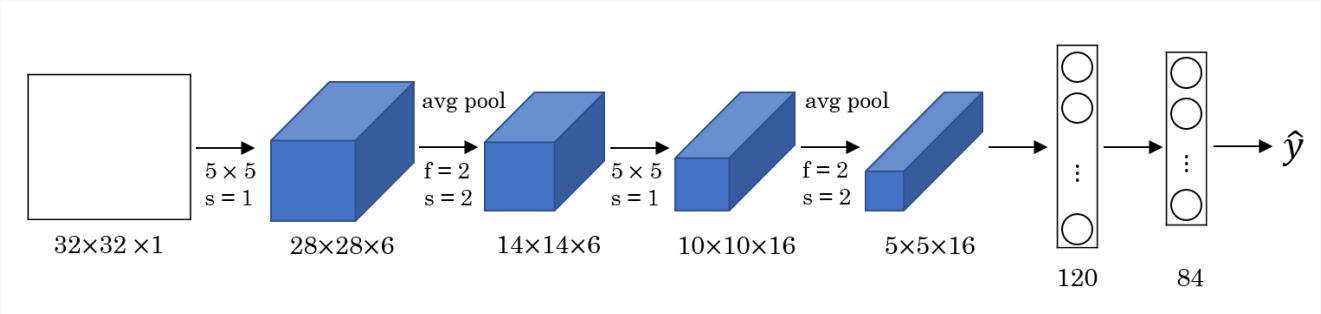
Overview of historically important networks

The MNIST problem

The Modified National Institute of Standards and Technology (MNIST) dataset consists in a training set of 60 000 images of handwritten digits, ranging from 0 to 9, and a separate test set of 10 000 handwritten digits. It is therefore a classification problem, consisting in classifying the input $28 \times 28 \times 1$ images into one of 10 classes.

LeNet - 5

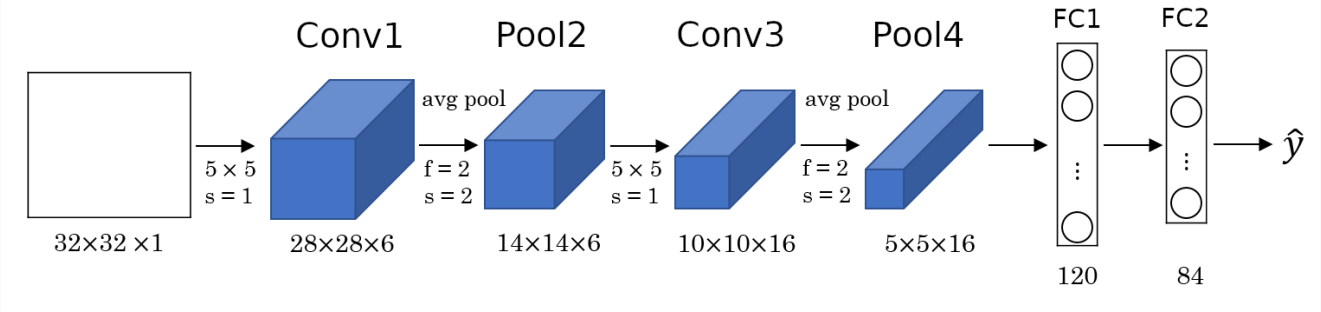
LeCun *et al.* proposed the following CNN to classify the MNIST dataset [2],



Some observations:

- The network consists of two convolutions, two pooling layers, and two fully connected layers
- Both the sigmoid (σ) and hyperbolic tangent (*tanh*) were used as activation functions
- Average pooling is used

LeNet - 5, breakdown of layers (1)

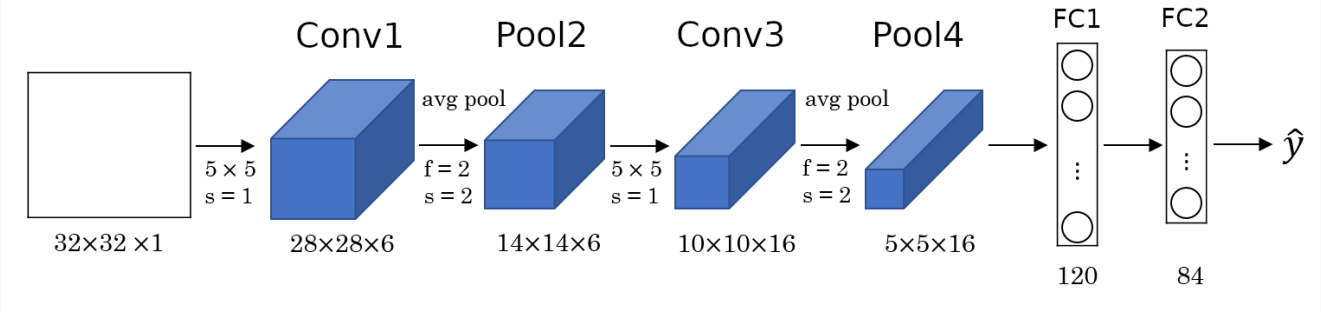


Conv1:

- Input volume size $W = 32$
- Filter size $F = 5$
- Padding $P = 0$
- Stride $S = 1$
- $\frac{W - F + 2P}{S} + 1 = \frac{32 - 5 + 2 \cdot 0}{1} + 1 = 28$
- Depth (number of filters) $D = 6$

Hence, the size of Conv1 is $28 \times 28 \times 6$

LeNet - 5, breakdown of layers (2)

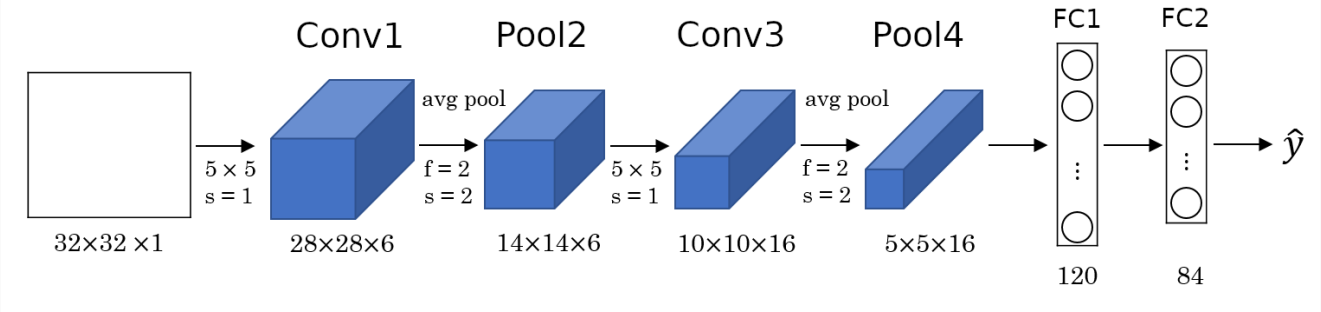


Pool2:

- Input volume size 28
- Filter size 2
- Stride 2
- For every 2×2 pixels, we obtain a single one, effectively dividing the volume size by 2 on each axis

Hence, the size of Pool2 is $14 \times 14 \times 6$

LeNet - 5, breakdown of layers (3)

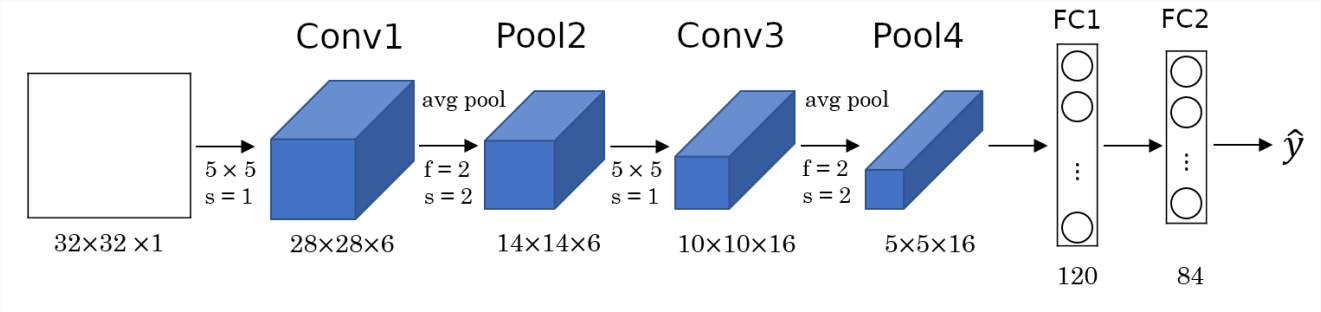


Conv3:

- Input volume size $W = 14$
- Filter size $F = 5$
- Padding $P = 0$
- Stride $S = 1$
- $\frac{W - F + 2P}{S} + 1 = \frac{14 - 5 + 2 \cdot 0}{1} + 1 = 10$
- Depth (number of filters) $D = 16$

The size of Conv3 is $10 \times 10 \times 16$

LeNet - 5, breakdown of layers (4)

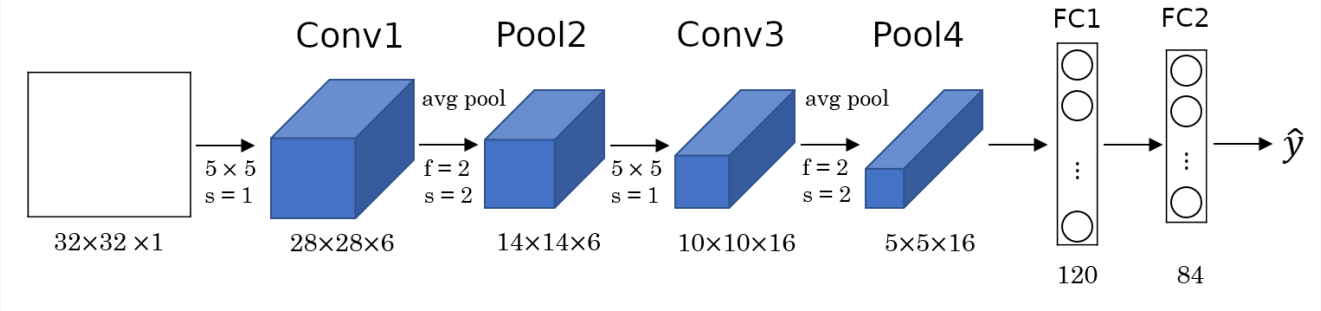


Pool4:

- Input volume size 10
- Filter size 2
- Stride 2
- For every 2×2 pixels, we obtain a single one, effectively dividing the volume size by 2 on each axis

The size of Pool4 is $5 \times 5 \times 16$

LeNet - 5, breakdown of layers (5)

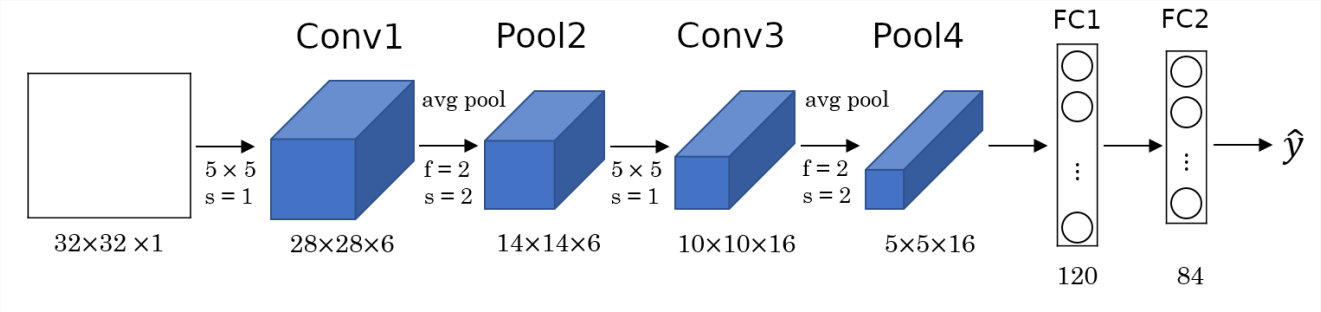


FC1:

- Input size is $5 \times 5 \times 16 = 400$
- Output size is 120

Therefore, the fully connected layer connects all 400 inputs to 120 outputs.

LeNet - 5, breakdown of layers (6)

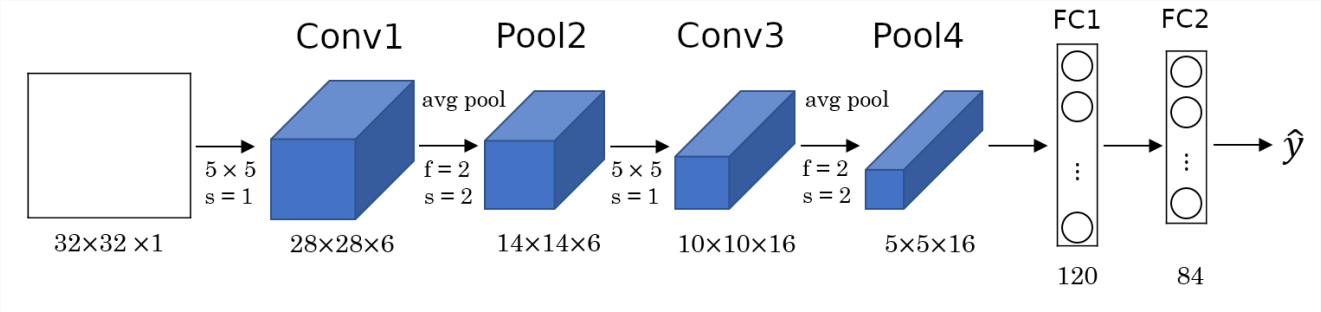


FC2:

- Input size is 120
- Output size is 84

The fully connected layer connects all 120 inputs to 84 outputs.

LeNet - 5, breakdown of layers (7)

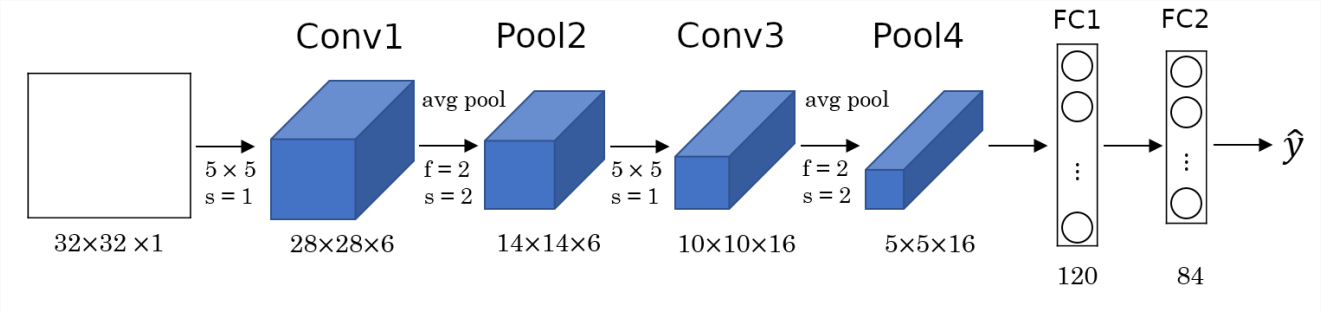


Output layer:

- Input size is 84
- Output size is one element per possible classification value, therefore 10

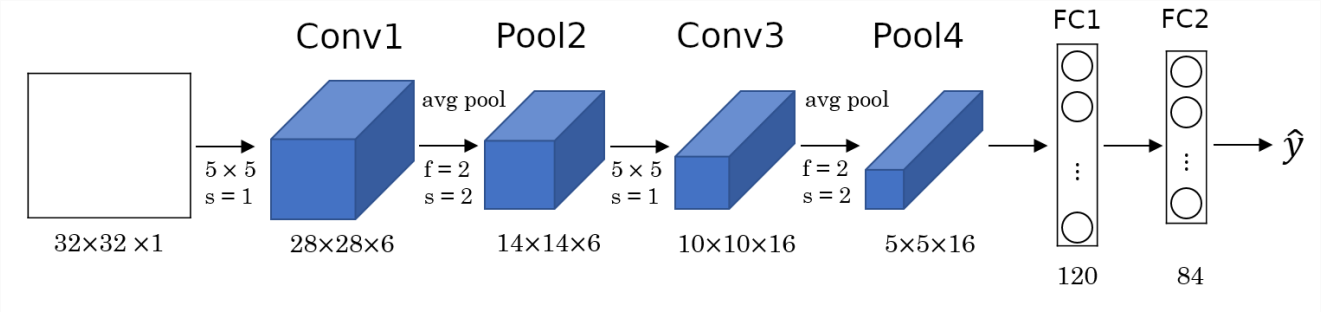
Finally, the output layer is effectively another FC layer.

LeNet - 5, learnable weights



- How many weights do we require?

LeNet - 5, learnable weights

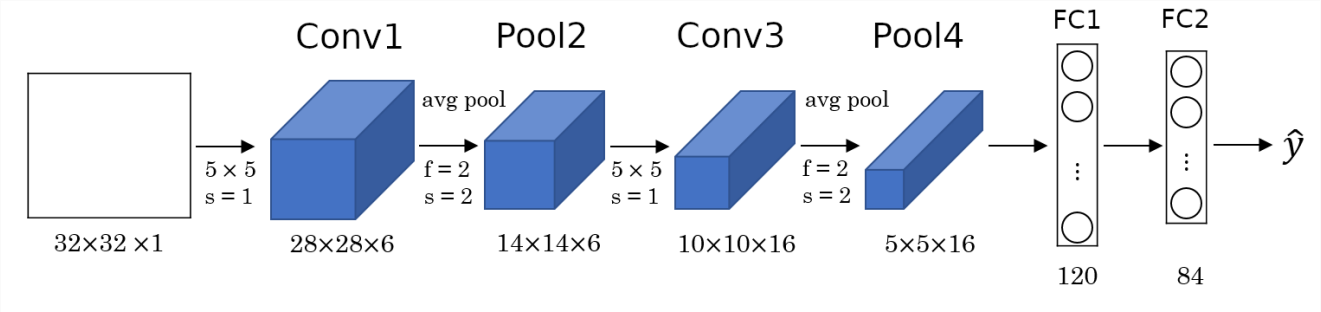


Conv1:

$$\begin{aligned}\texttt{parameters} &= \texttt{connection weights} + \texttt{bias weights} \\ &= F \times F \times D_{l-1} \times D_l + D_l \\ &= 5 \times 5 \times 1 \times 6 + 6 = 156\end{aligned}$$

Given we have 28×28 output volumes, the number of learnable parameters of Conv1 is $28 \times 28 \times 156 = 122\,304$

LeNet - 5, learnable weights



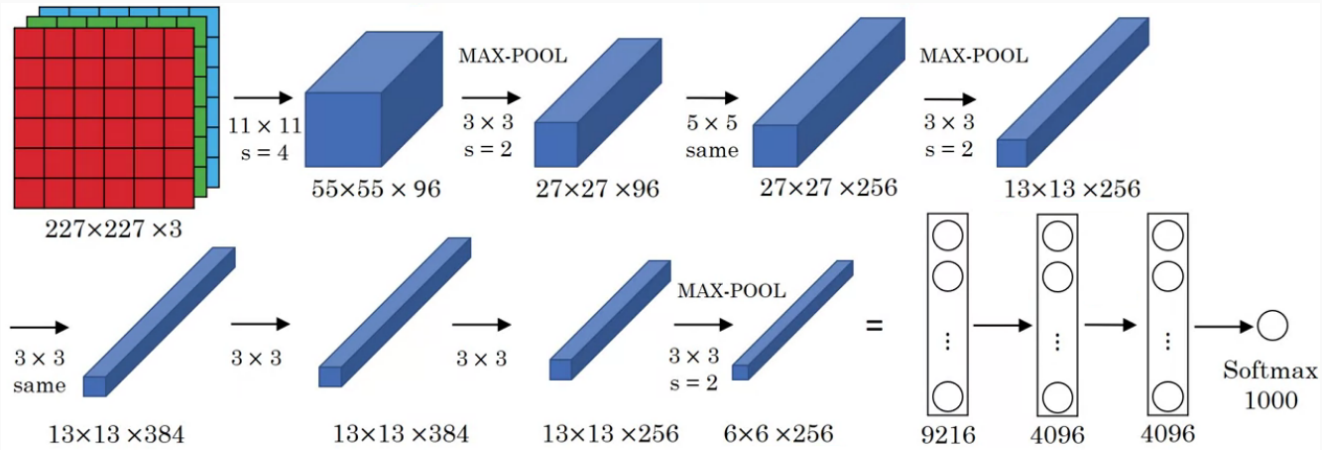
- In total, we have about 340 000 parameters in LeNet-5. Not a lot for today's standards, but definitely impressive for 1998.

The ImageNet challenge

As we introduced before, the ImageNet challenge presents about $1.4 \cdot 10^6$ images of 1000 classes. An answer is the five most probable classes from the output of the algorithm, and if the correct one is contained within those five, the answer is correct.

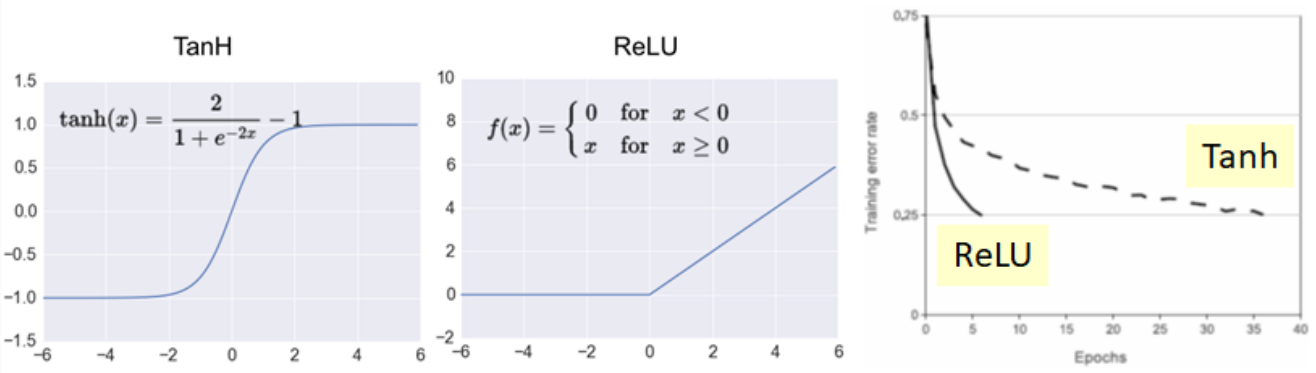
AlexNet

AlexNet is a CNN introduced in 2012 by Krizhevsky *et al.* [3] that won the ImageNet challenge by a margin of more than 10%. The network topology of Alexnet is as follows:



AlexNet - Remarks

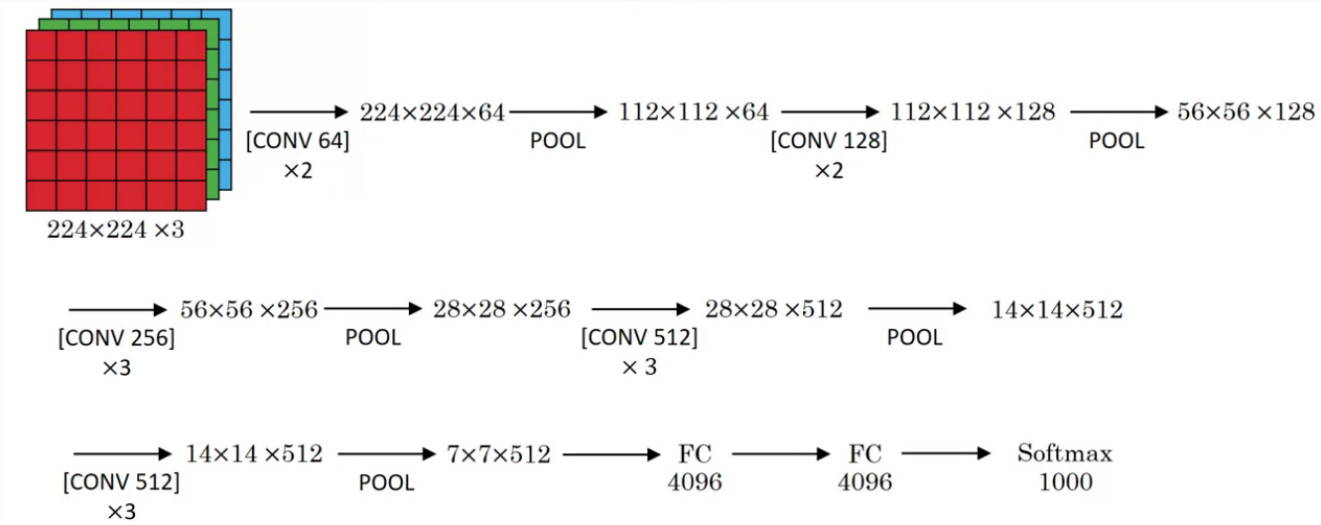
- The Rectified Linear Unit function is introduced. It is claimed to converge up to 6$\times$ faster than the sigmoid or hyperbolic tangent:



- Similar topology to LeNet, but bigger
- Max pooling is used
- It uses padding to keep same size
- About 60*M* parameters
- Multiple GPUs are used in learning, and so layers are separated in two at every step

VGG-16

Recently, many versions of deep neural networks have made an appearance in literature. One such example is VGG, from Simonyan and Zisserman [4]. Various versions of this neural network exist, the one below is VGG-16:

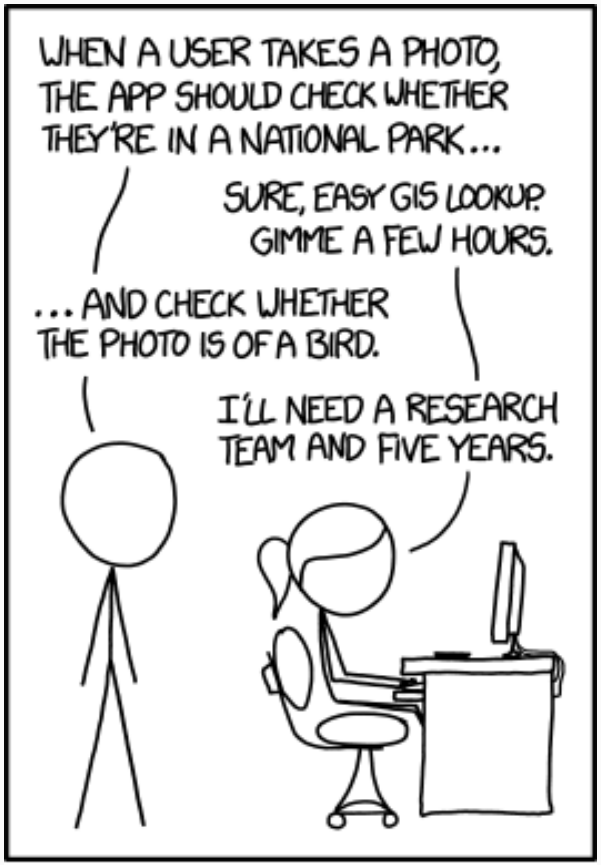


VGG-16 - Remarks

- Note how complex designs are referred to as modular blocks:
  - `[CONV 64]` $\times 2$ refers to two convolutional nets with 64 channels (depth 64)
  - `POOL` - Max pooling is assumed, if not specified otherwise
  - `ReLU` is assumed, if not specified otherwise
- `CONV` is always a 3×3 filter, with stride $S = 1$, and padding $P = 1$ (*same*)
- `POOL` is always a 2×2 MAX-POOL, with stride 2
- About $138M$ parameters
- The design follows a pattern: Every successive `CONV` layer doubles in number of channels (*depth*), and halves in width and height

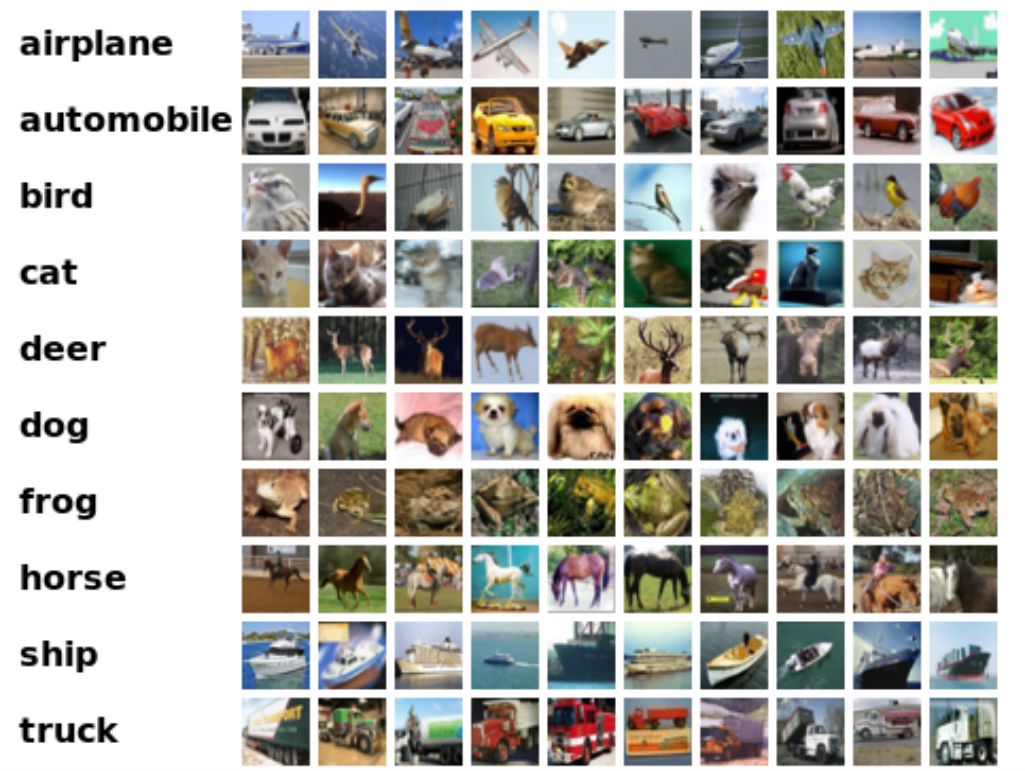
Cifar-10 example

Example: Classifying images



Well, let's actually do it

Our input will be the CIFAR-10 data set, which contains a training set of 50 000 images of 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck). Additionally, it has 10 000 extra images, which can be used for the validation and test sets.



What to expect in these images

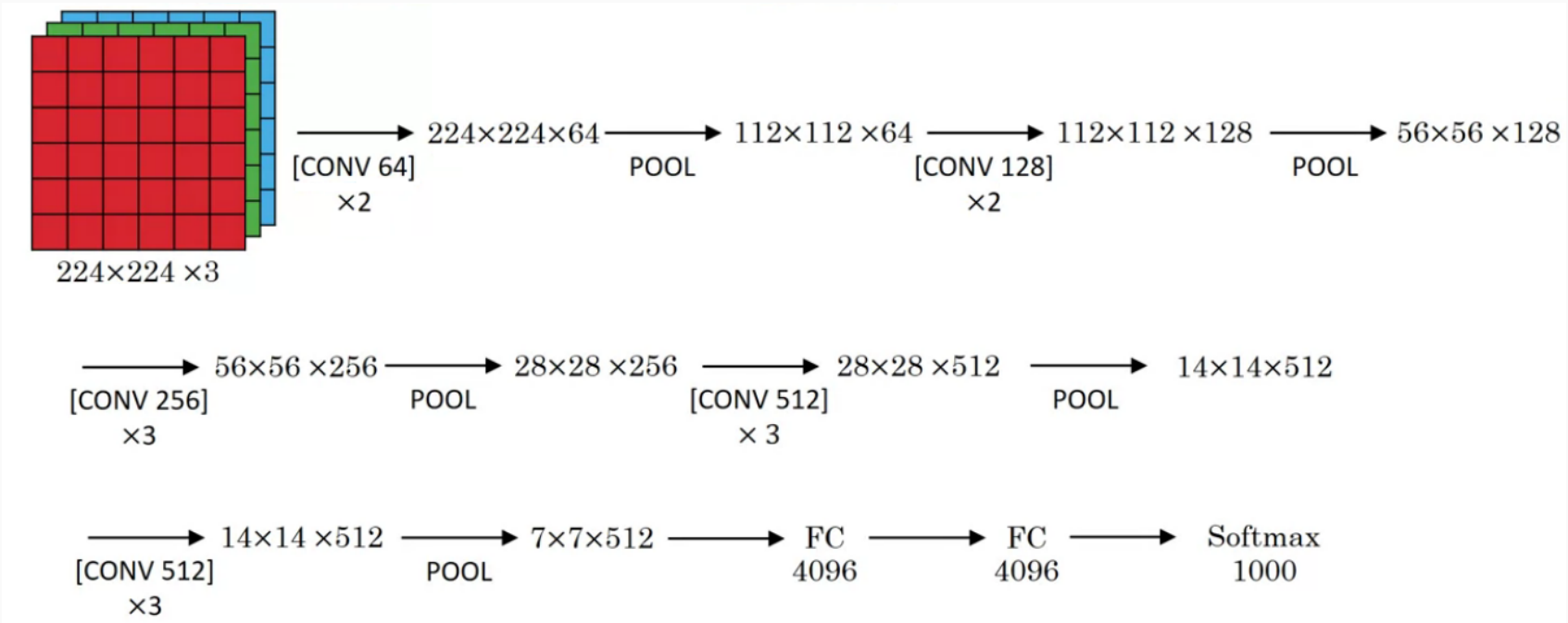
There is no *standard of photography* in these images. Some may be taken up close, others relatively far from the object. The object we want to identify may be rotated, deformed or cropped. Light conditions may vary...

Given all the variations in conditions, instead of just taking the images we will use a *generator* to simulate them, taking as a basis the elements in our training set.

Taking inspiration

We can draw inspiration in an existing topology configuration, like that one from VGG-16.

- No need to reinvent the wheel
- As we will see later, it is even possible to reuse learnt models



Our convolutional neural network

Here is a proposed simple design,

- Input is 32x32x3
- `CONV 32`
- `POOL`
- `CONV 64`
- `POOL`
- `FC 256`
- Output

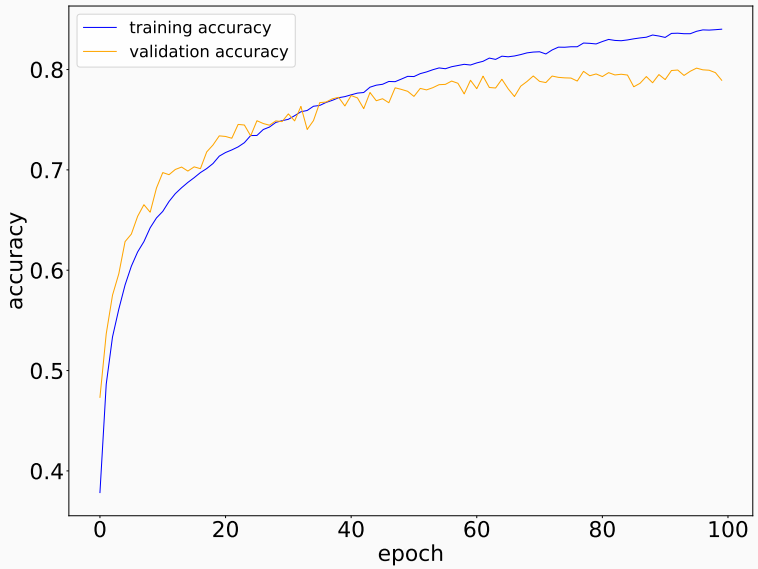
Printing the topology of the network

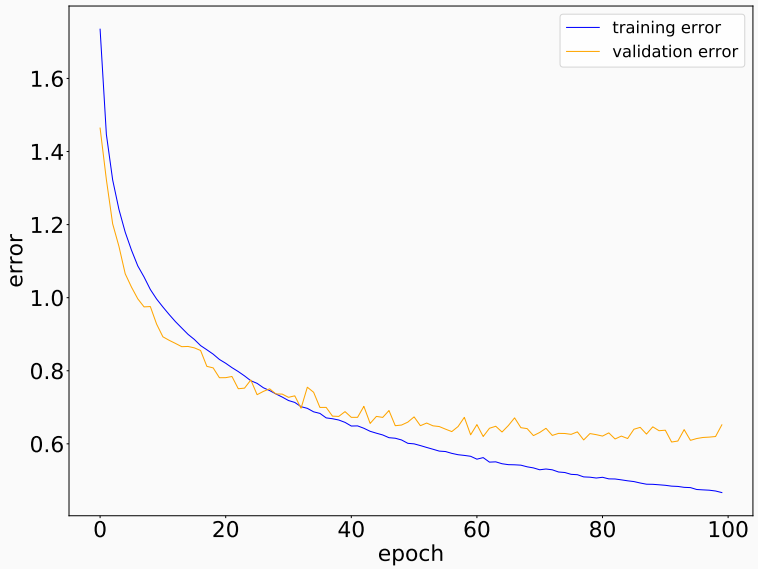
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
activation_1 (Activation)    (None, 32, 32, 32)        0
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
activation_2 (Activation)    (None, 16, 16, 64)        0
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
flatten_1 (Flatten)          (None, 4096)              0
dense_1 (Dense)              (None, 256)               1048832
activation_3 (Activation)    (None, 256)               0
dense_2 (Dense)              (None, 10)                2570
activation_4 (Activation)    (None, 10)                0
=================================================================
Total params: 1,070,794
Trainable params: 1,070,794
Non-trainable params: 0
```

Checking the efficiency of our network

Finally, let's check how we did. Depending on how we separated our data set into a training set and a validation set, results may vary.

- Epochs: 100
- Test loss: 0.6520
- Test accuracy: 0.7893

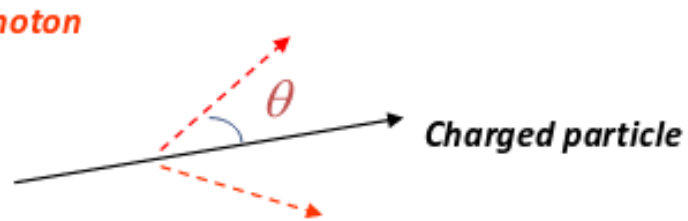




RICH reconstruction example

Problem formulation

Cherenkov radiation principle



$\cos(\theta) = 1/(n\,\beta)$ where n = Refractive Index = $c/c_M = n(E_{ph})$

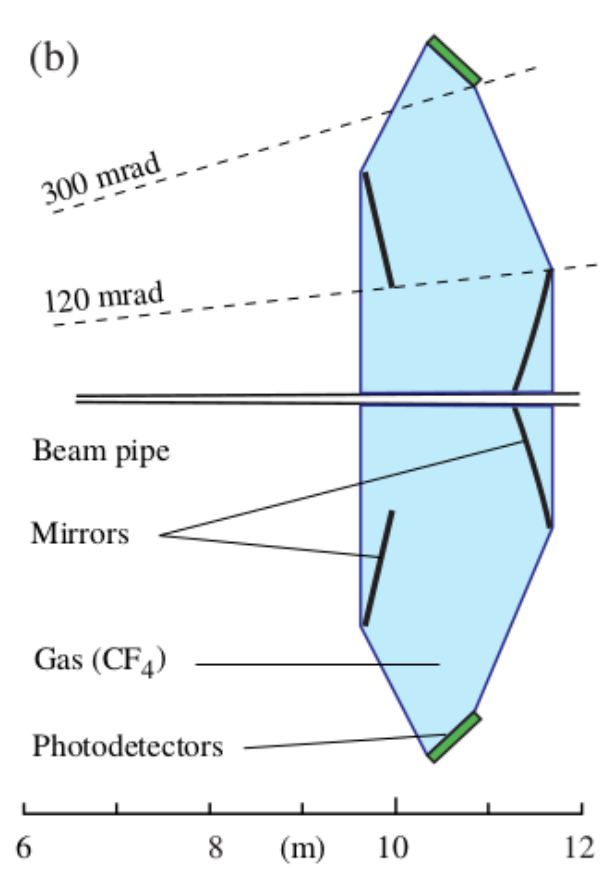
$$\beta = v/c = p/E = p/(p^2 + m^2)^{0.5} = 1/(1+(m/p)^2)^{0.5}$$

β= velocity of the charged particle in units of speed of light (c) vacuum
p, E ,m = momentum, Energy, mass of the charged particle.
C_M = Speed of light in the Medium (Phase velocity) ,
E_{ph} = Photon Energy, λ=Photon Wavelngth.

- Theory of Cherenkov Radiation: Classical Electrodynamics by J.D.Jackson (Section 13.5)

RICH detectors in LHCb

In LHCb, we have two RICH detectors. Below is a schematic XZ view of RICH1:



Analytical solution

The analytical solution consists in *creating photons*, ie. associations of detected pixels in the HPDs / MaPMTs with their originating track segments through the Rich detector.

Once this association is found, a likelihood minimisation algorithm is run in order to find the most likely candidate for each particle.

- Photon creation, heavily involving ray tracing
- Likelihood minimisation

Analytical solution - Ray tracing

The creation of the photons involves a ray tracing algorithm from each segment to the candidate pixels. In turn, this means solving the quartic equation:

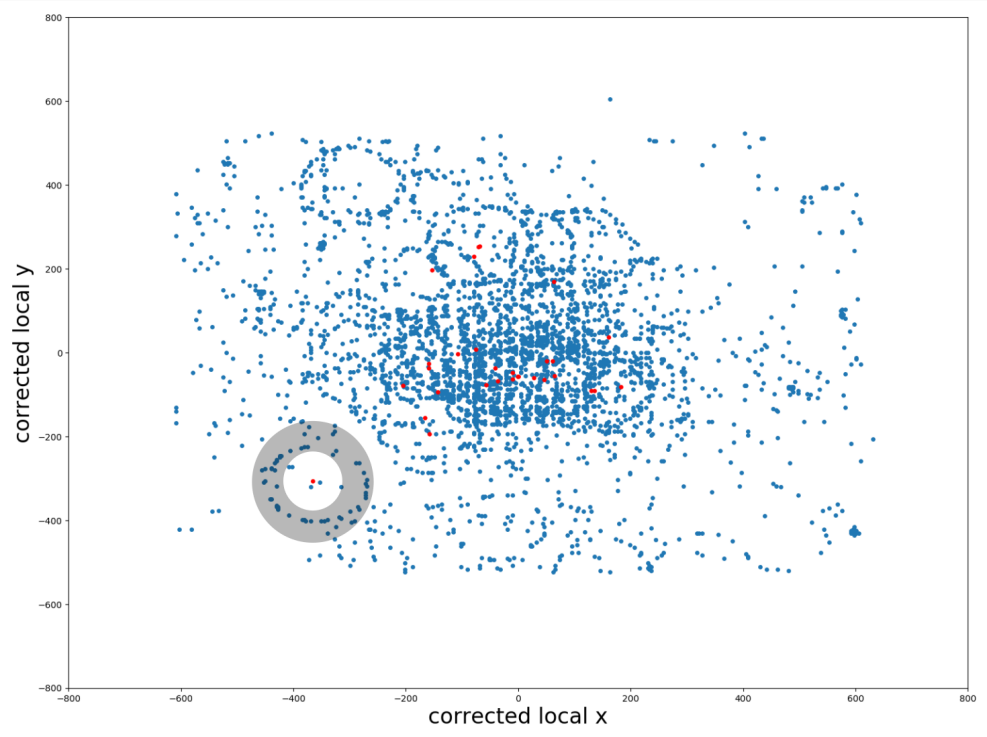
$$4e^2d^2\sin^4\beta - 4e^2d_yR\sin^3\beta + \left(d_y^2R^2 + (e+d_x)^2R^2 - 4e^2d^2\right)\sin^2\beta + 2ed_y(e-d_x)R\sin\beta + (e^2-R^2)d_y^2 = 0\,. \quad (3)$$

This can be solved, using for example a routine in the CERN library [6], and gives four solutions for $\sin\beta$, two complex and two real. Of the real solutions one is the "backward" reflection (that would exist if the mirror were a complete sphere, shown as M′ in Fig. 3); the other is the desired solution, and can be selected from knowledge of the RICH detector geometry. The value of $\cos\beta$ can then be extracted using Eq. 2, and the coordinates of the reflection point M determined.

Bringing ML into the fray

Alternatively, we are studying whether it is possible to transform the problem into a classification problem from an image into a particle ID (one of *pion*, *muon*, *electron*, *kaon*, *proton*, *deuteron*):

- Each track is extrapolated onto the detector plane (red dots)
- A *corona* shape is fed onto a Convolutional Neural Network to identify the particle



Motivation

If we observe the surrounding area of a single track extrapolation and convert it to polars, we end up with the figures below:

Figure 4: Left: Pion. Right: Electron.

Exploring possibilities

Some CNNs are more amenable to solving the problem at hand. In fact, the problem looks very similar to the classical MNIST problem. One such CNN is:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 30, 30, 32)        320
conv2d_29 (Conv2D)           (None, 28, 28, 64)        18496
max_pooling2d_13 (MaxPooling (None, 14, 14, 64)        0
dropout_23 (Dropout)         (None, 14, 14, 64)        0
flatten_13 (Flatten)         (None, 12544)             0
dense_24 (Dense)             (None, 128)               1605760
dropout_24 (Dropout)         (None, 128)               0
dense_25 (Dense)             (None, 6)                 774
=================================================================
Total params: 1,625,350
```

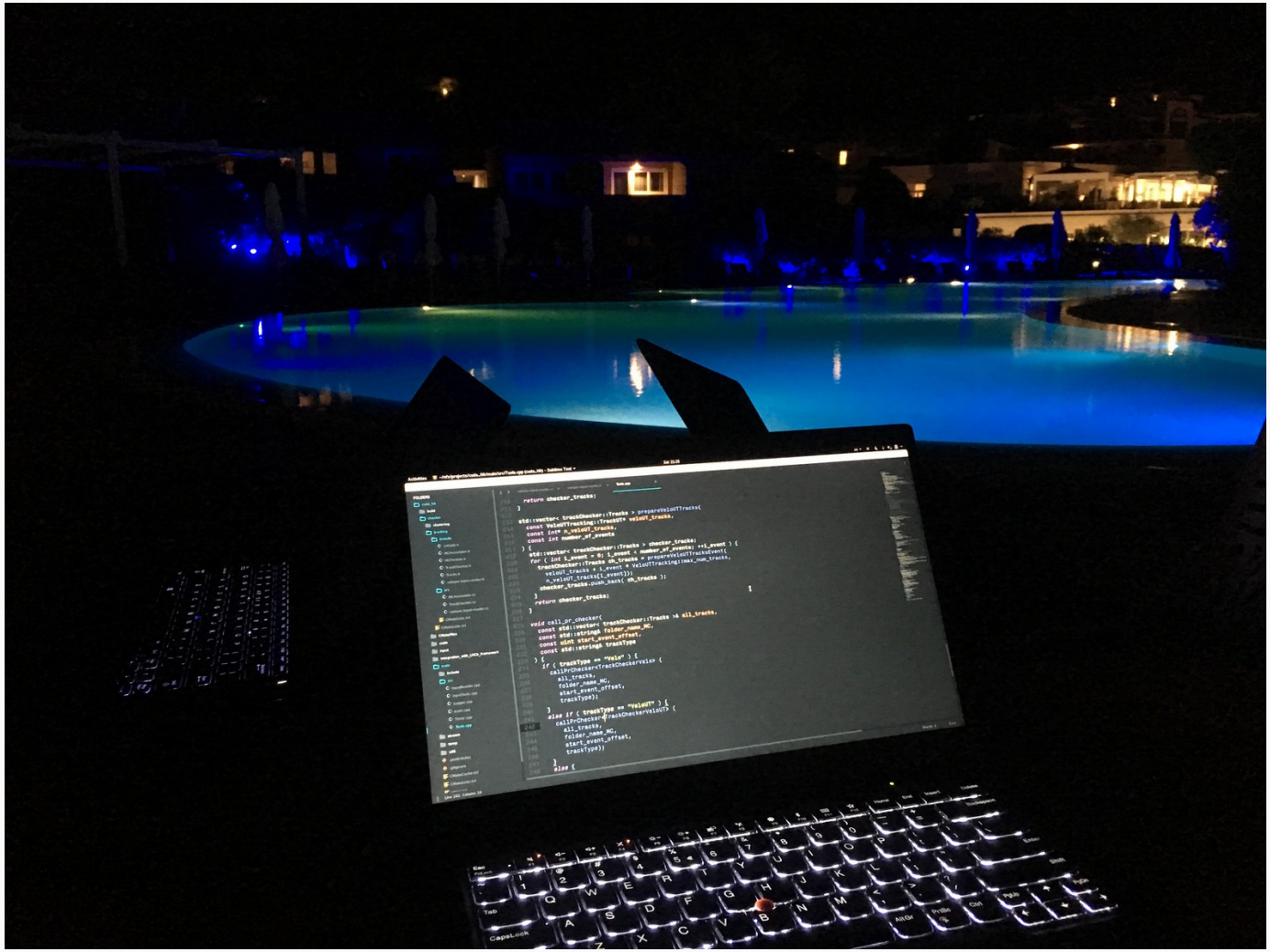
Expected results

Particle type	# particles	Accuracy	Loss
electron	76	0.013158	5.977299
kaon	540	0.557407	1.147596
muon	12	0.000000	12.524119
pion	2058	0.965015	0.140477
proton	299	0.304348	2.457402

Predictions (%tot)	electron	kaon	muon	pion	proton	Purity (%)
electron	0.034	0.034	0.000	2.412	0.067	1.316
kaon	0.000	10.084	0.000	5.829	2.178	55.741
muon	0.000	0.000	0.000	0.369	0.034	0.000
pion	0.034	1.374	0.000	66.533	1.005	96.501
proton	0.000	2.647	0.000	4.322	3.049	30.435
Efficiency (%)	50.000	71.327	0.000	83.727	48.148	

ID eff (%)	K->K,Pr,D : 90.047	pi->e,m,pi : 87.226
MisID eff (%)	K->e,m,pi : 9.953	pi->K,Pr,D : 12.774

See you in the exercise section!



Bibliography

Resources i

[1] CS231n: Convolutional Neural Networks for Visual Recognition. `http://cs231n.stanford.edu/`

[2] LeCun et al., 1998. Gradient-based learning applied to document recognition.

[3] Krizhevsky A., Sutskever I. and Hinton G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Part of: Advances in Neural Information Processing Systems 25 (NIPS).

[4] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition.