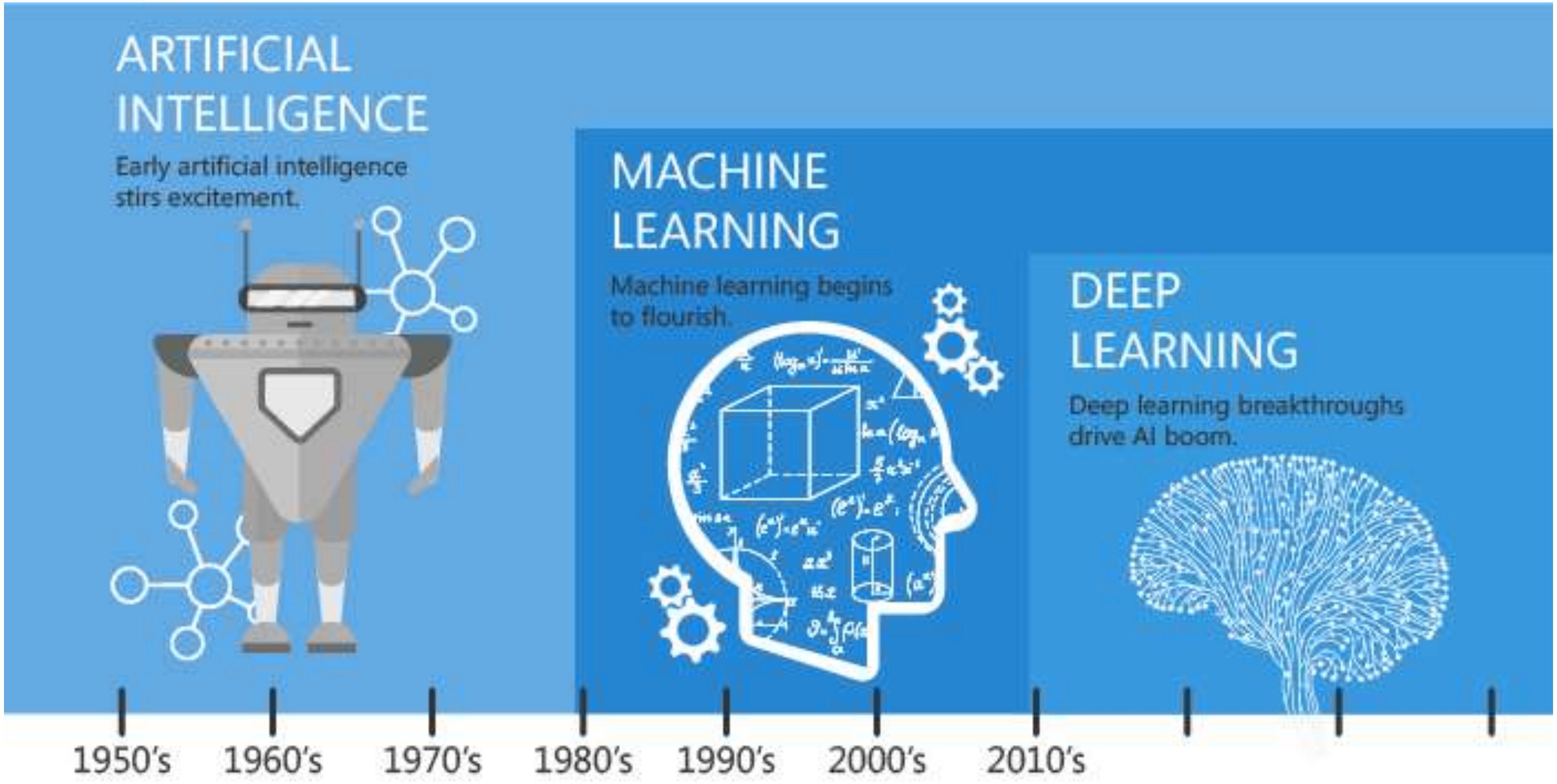
# Deep Learning

Stéphane Gentric, Ph. D.
Senior Expert
Research Unit Manager

Fev 2020

- **Intro**
- **Deep @ Idemia**
- **Data**
- **Learning Frameworks**
- **Neural Network**
- **CNN Architecture**
- **Optimization**
- **Overfitting**
- **More DNN Architectures**
- **Adversaries**
- **Tips**

# ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, AND DEEP LEARNING




Since an early flush of optimism in the 1950's, smaller subsets of artificial intelligence - first machine learning, then deep learning, a subset of machine learning - have created ever larger disruptions.


**In "Nature" 27 January 2016:**

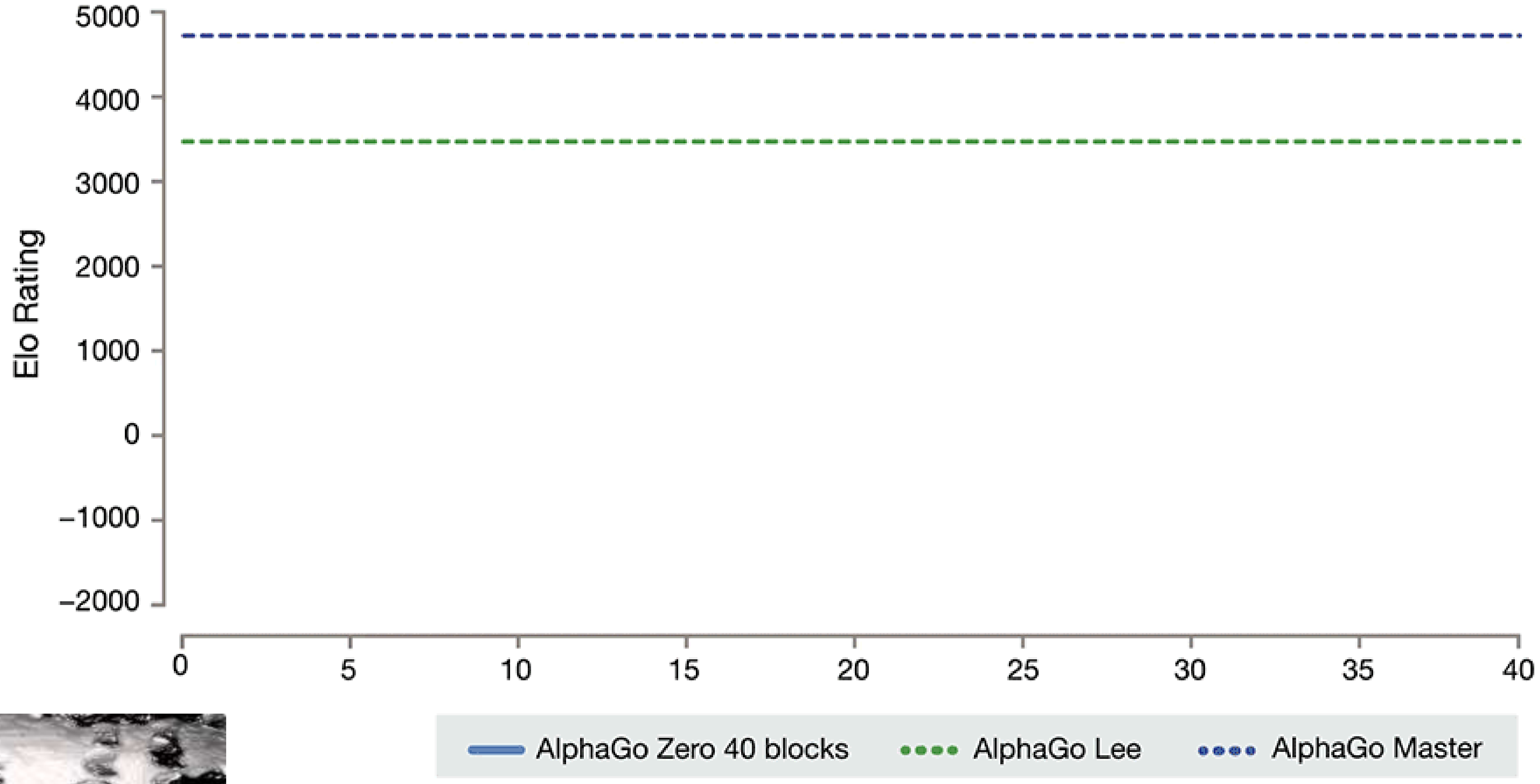
**"DeepMind's program AlphaGo beat Fan Hui, the European Go champion, five times out of five in tournament conditions..."**

**"AlphaGo was not preprogrammed to play Go: rather, it learned using a general-purpose algorithm that allowed it to interpret the game's patterns."**

**"…AlphaGo program applied deep learning in neural networks (convolutional NN) — brain-inspired programs in which connections between layers of simulated neurons are strengthened through examples and experience."**

# ALPHAGO ZERO: LEARNING FROM SCRATCH

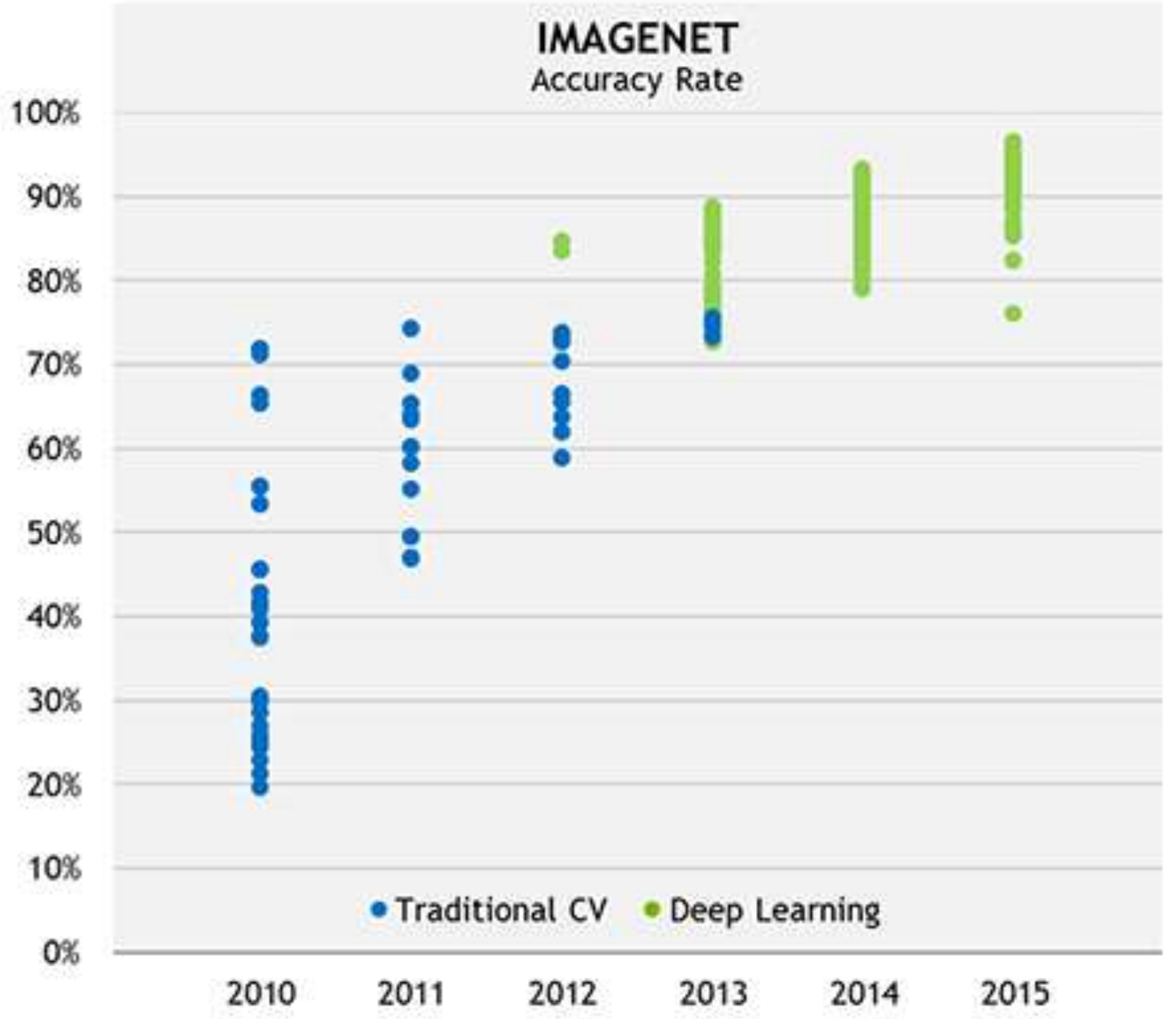





# The ImageNet challenge

- Crucial in demonstrating the effectiveness of deep CNNs
- Problem: recognize object categories in Internet imagery
- The 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) classification task - classify image from Flickr and other search engines into 1 of 1000 possible object categories
- Serves as a standard benchmark for deep learning
- The imagery was hand-labeled based on the presence or absence of an object belonging to these categories
- There are 1.2 million images in the training set with 732-1300 training images available per class
- A random subset of 50,000 images was used as the validation set, and 100,000 images were used for the test set where there are 50 and 100 images per class respectively

# IMAGENET CHALLENGE: RECOGNIZE OBJECT CATEGORIES


- 1000 object categories
- **Hand-labeled** images
- 1.2 million images in the training set

Global R&D / Machine / Deep Learning in R&T

January 2019

# A plateau, then rapid advances

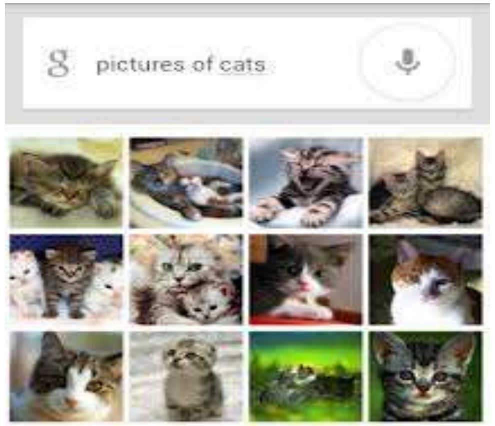
**"Top-5 error" is the % of times that the target label does not appear among the 5 highest-probability predictions**

**Visual recognition methods not based on deep CNNs hit a plateau in performance at 25%**

| Name | Layers | Top-5 Error (%) | References |
|---|---|---|---|
| AlexNet | 8 | 15.3 | Krizhevsky et al. (2012) |
| VGG Net | 19 | 7.3 | Simonyan and Zisserman (2014) |
| ResNet | 152 | 3.6 | He et al. (2016) |

# Deep Learning – from Research to Technology





**Deep Learning - breakthrough in visual and speech recognition**

# Deep @ Idemia

2

# A short history of Biometrics at Idemia

**1924**
Creation of **SAGEM**

**1982**
Creation of **Morpho Systèmes**

**1993**
SAGEM acquires **Morpho Systèmes**

…

**2005**
Sagem and Snecma merge to create
SAFRAN

**2006**
**Orga Kartensysteme** joins Safran

**2007**
Creation of **Sagem Sécurité**

**2008**
**Sdu-Identification** joins Safran

**2009**
**GE Homeland Protection** joins Safran

**Printrak** joins Safran

**2010**
Sagem Sécurité becomes **Morpho**

**2011**
L-1 Identity Solutions joins Safran and will become **MorphoTrust USA**

**2012**
**Cassis International** joins Safran

**2013**
**CPS Technologies SAS** and **ABnote Czech sro** join Safran

**StarChip** joins Safran

**2014**
**Dictao** joins Safran

**2015**
**Airtag** joins Safran

**2016**
Morpho becomes
SAFRAN
Identity & Security

**2017**
Safran I&S joins Oberthur and becomes
IDEMIA

# #1 Leader de l'Identité Augmentée dans un monde de plus en plus digital

Près de
3 Milliards de $
de chiffre d'affaires

+ de 14 000 Collaborateurs
dont + de 2 000 en R&D

Plus de
80 Nationalités

Un ensemble de solutions innovantes et intégrées
200 Millions d'$
en R&D en 2017 & plus de 1400 familles de brevets actives

Production de cartes à grande échelle

+ de 3 Milliards
de documents d'identité délivrés dans le monde

1,2 Milliard
de cartes SIM livrées en 2017

+ de 850 Millions
de cartes de paiement produites en 2017

+ de 250 Millions
d'éléments sécurisés embarqués déployés dans le monde

+ de 20
solutions de gestion d'abonnement déployées dans le monde

**N°1** des systèmes d'identification biométriques

**1 800** institutions financières nous font confiance

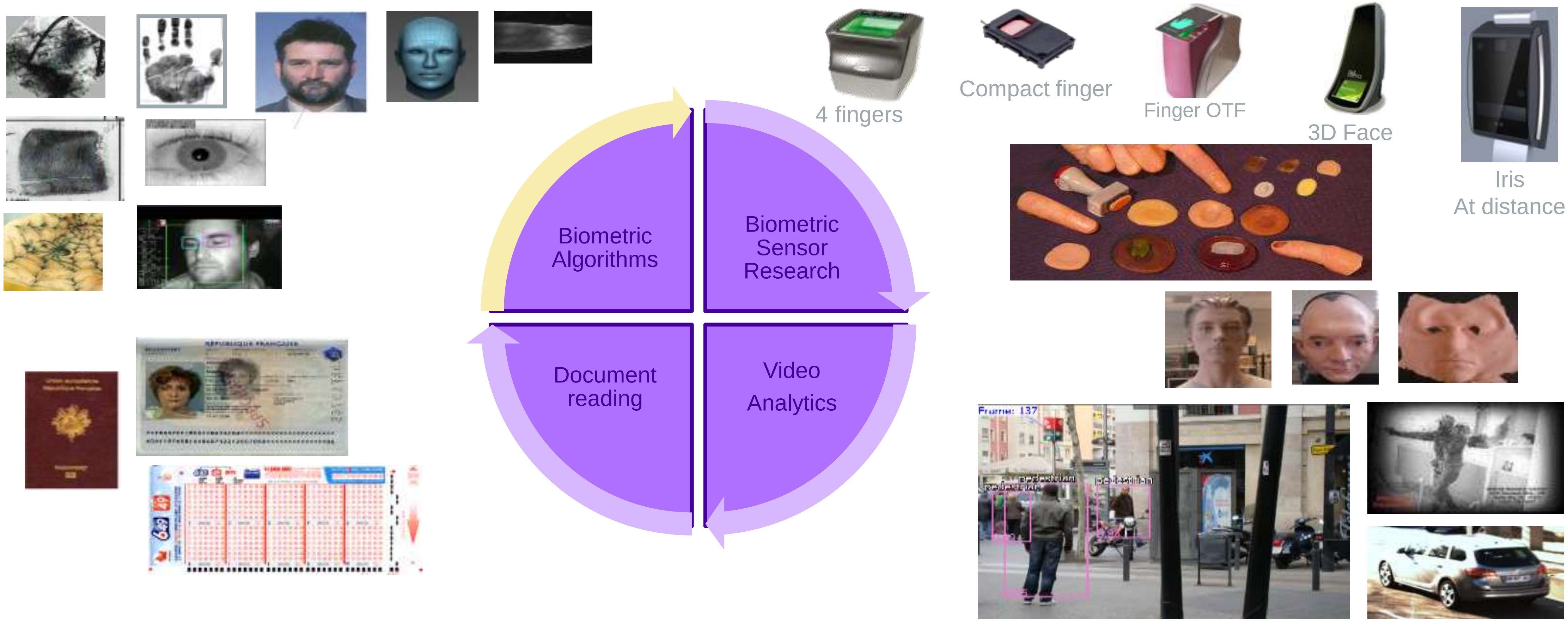
**N°1** des solutions d'identité civile

**+ de 500**
opérateurs mobiles nous font confiance

**Les plus grands OEM industriels**
nous font confiance

**N°1** dans l'émission de permis de conduire aux États-Unis

# Global R&D - Research and Technology Unit



# Face Recognition Markets

- **Identity Solution**
  - Secure Deliverance of a unique right (Vote, ID document)
  - Statistically good Quality images … but very large population

UIDAI
- Unique ID for India
- 10 fingers + 2 irises + 1 face
- 1+ billion citizens

- **Forensic Application**
  - Heterogeneous image quality
  - Police officer may spend time on a single case

FBI
- central US system for latent fingerprints, tenprints and faces

- **Border Control Application**
  - Increase workflow & Maintain Security level
  - Passenger cooperation level.

SmartGate
- Australian/New Zealand Airports
- 80+ Gates - facial
- 40+ millions crossing

- **Connected Objects & Financial Institutions**
  - Liveness, UI, low CPU/Memory

# Object & Face Recognition Pipeline

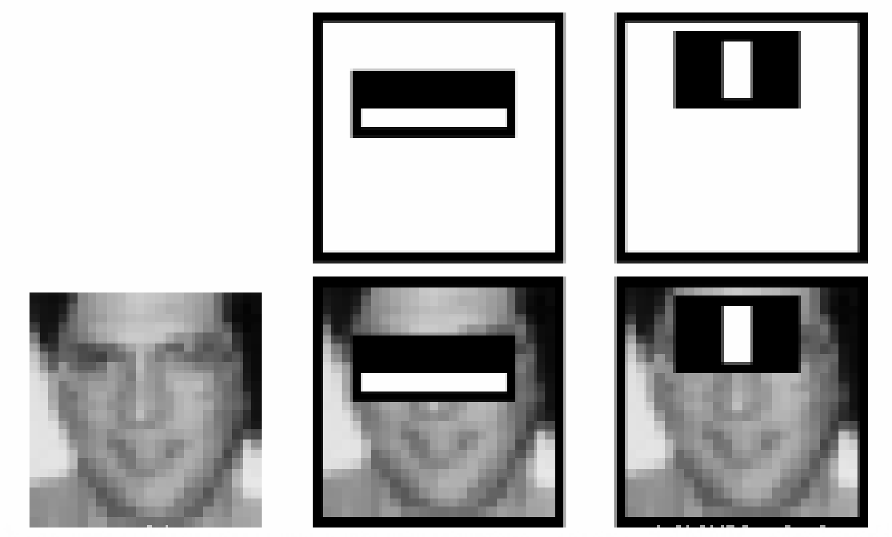
| Static Image | | Video Stream |
|---|---|---|
| Find the Objects/Faces in images. Mugshot, ID Doc, Selfie, in the wild | Detection Tracking | Track objects/persons in video<br>Limit false face detections. |
| Understand and correct : pose, light sources, occlusions | Registration | Track and refine pose estimation to improve robustness. |
| Extract discriminative features and qualities | Features Extraction | Extract discriminative features and qualities<br>Select and fuse features. |
| Optimize speed / accuracy trade-off | Matching | Control False Alarm Rate. |

# Face Detection Algorithm

Classifier

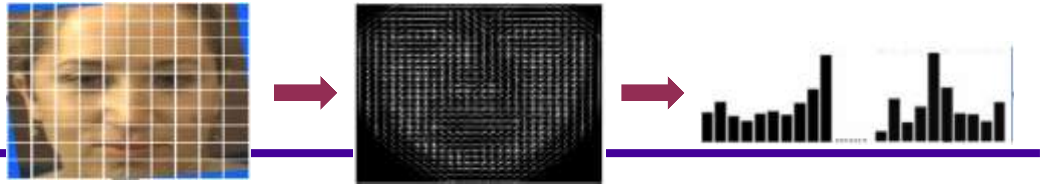
- Robust features, ex: Haar, Filter bank, SIFT, SURF, HOG, color histograms
- Machine learning, ex: SVM, Ada-Float boost, decision tree

The classifiers are learned on a database of faces and non faces. Bootstrap and selection of hard examples.
The detection pipeline is a fusion of those approaches for obtaining the best speed/accuracy tradeoff.



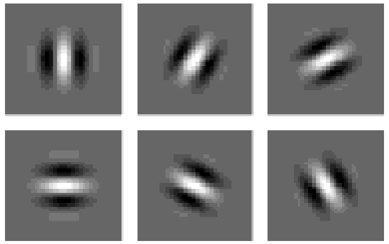
P.Viola et M.Jones, « Robust Real-time Object Detection », IJCV, 2001

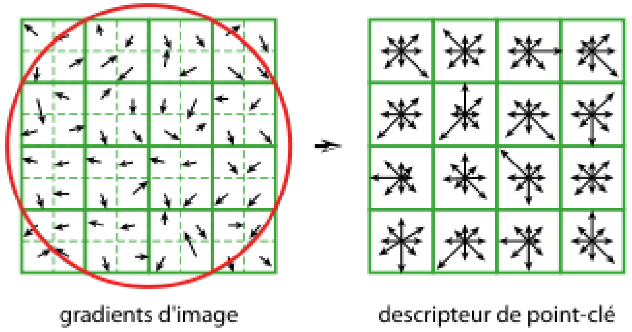
# Traditional Approaches



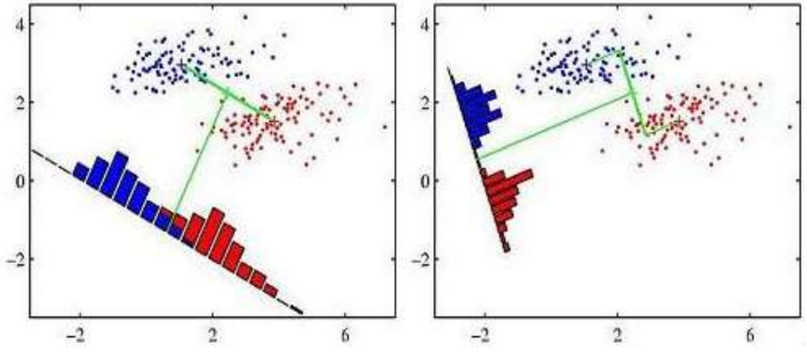
## These methods are composed of two steps

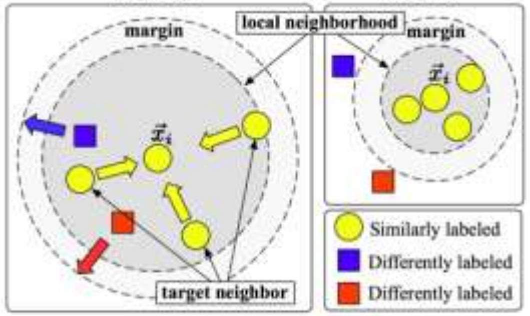
1. Agnostic features are extracted from the images
   - Most of the discriminative information is born by the gradients
   - Various representations of the gradients have been proposed
     - SIFT, HOG, LBP
     - Filter bank responses (Gaussian derivatives, Gabor, etc.)






2. A transformation of the initial feature space is learned
   - It reduces the dimensionality and concentrates the discriminative information (remove noise and redundancy)
   - Most methods simply use a linear transformation
     - LDA, ITML, LMNN, …
   - But more complex methods exist (e.g. local metric)

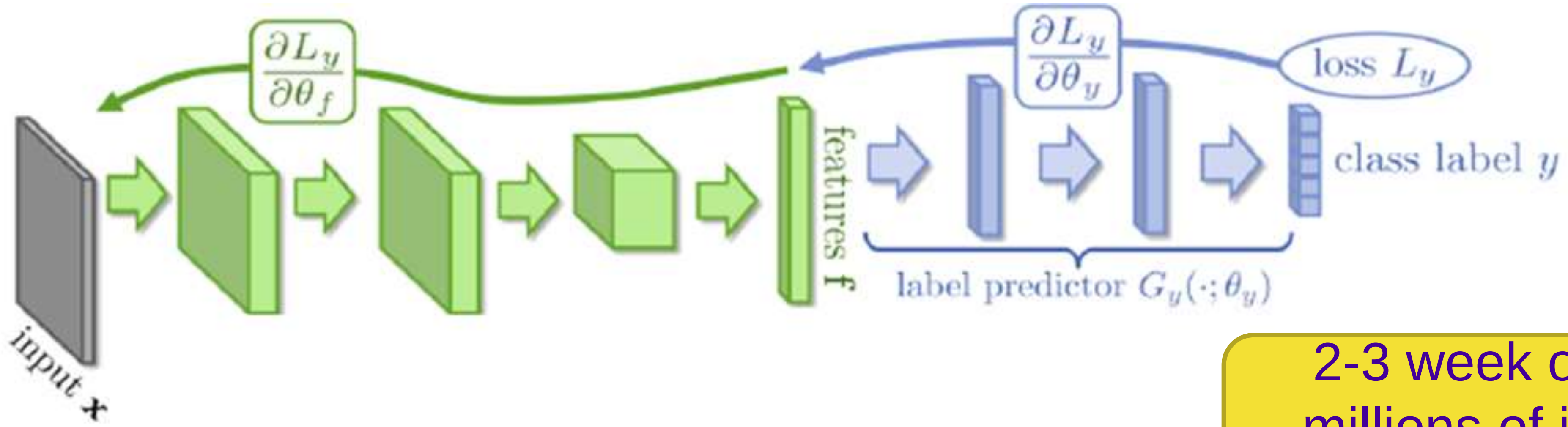





**A well-engineered traditional method gives accurate results on well aligned images but lacks of robustness**

# Deep Networks for Face Recognition

- A Face template is learned directly from the pixels.




2-3 week on GPUs
millions of iterations
millions of images

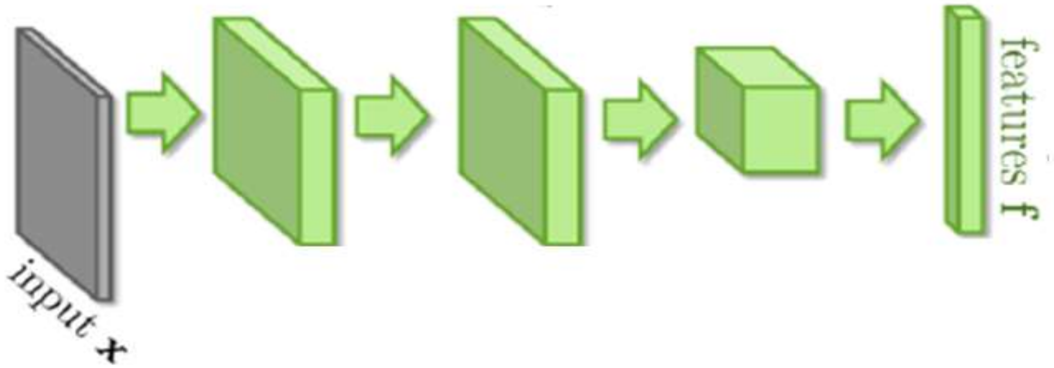
**Biometric Performances comes from :**

- **The size of the image database** ( thousands of identities, hundreds of images per id )
- **The architecture of the Network ( number of layers, maps …)**
- **The loss functions**

# Deep Networks for Face Recognition

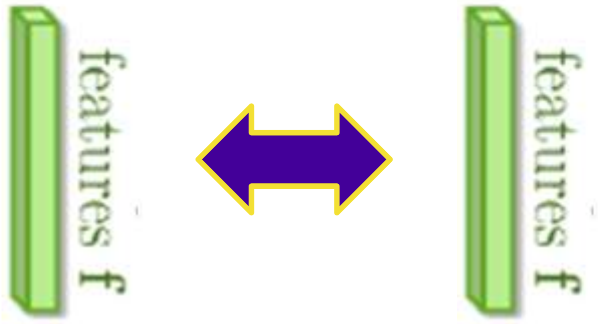
**Coding = build a template from an image**

- A chain of forward operation in the Network
- Coding is done on CPU or GPU. On a PC, a Smartphone or in the Cloud



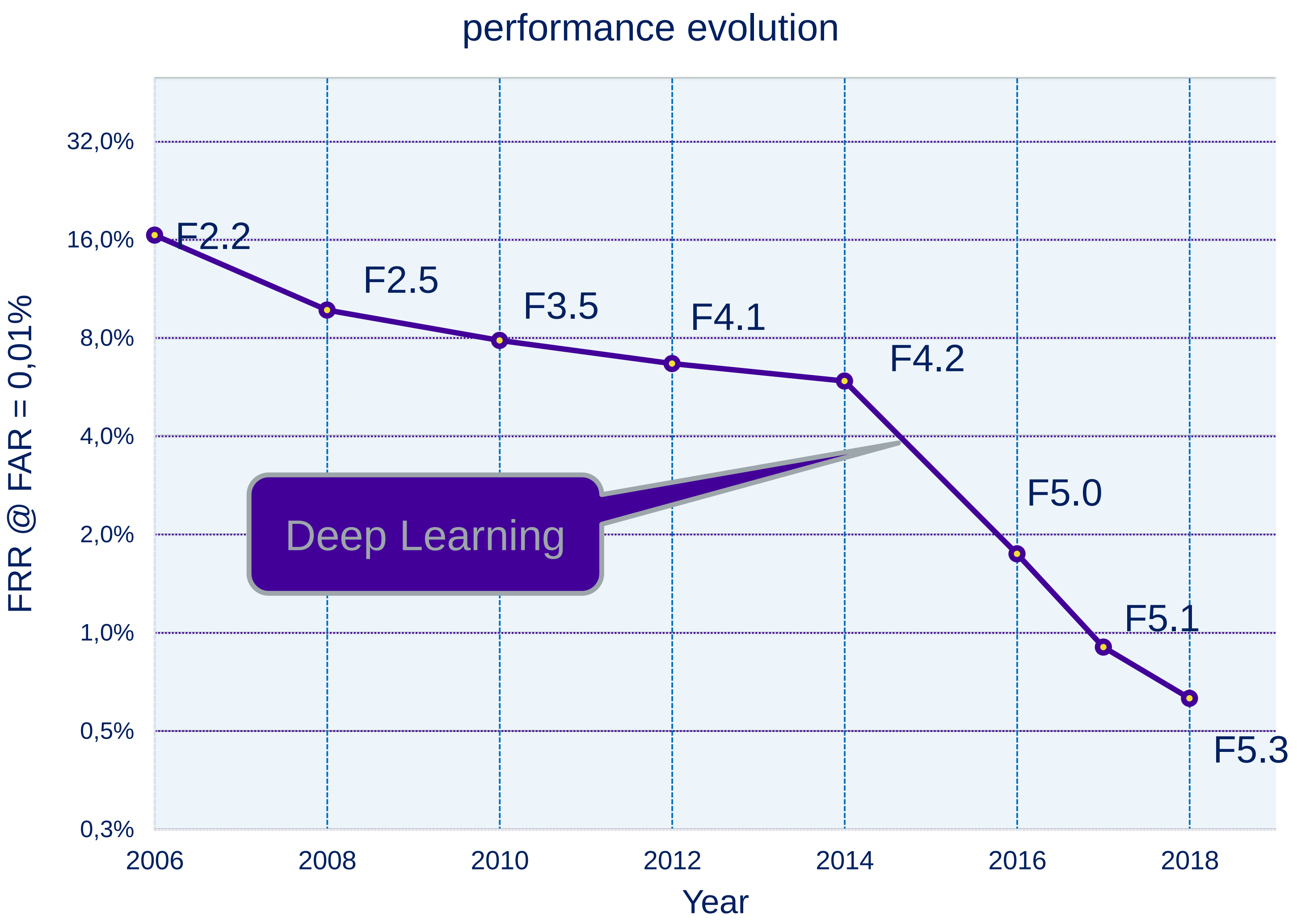

**Matching = compare 2 templates**

- A simple Euclidian distance is used
- Very fast. Millions per second per core



# Face Recognition Performances

- **Median value on 40 different internal databases.**
- **Absolute performance varies with images quality**



# Vendor Benchmark

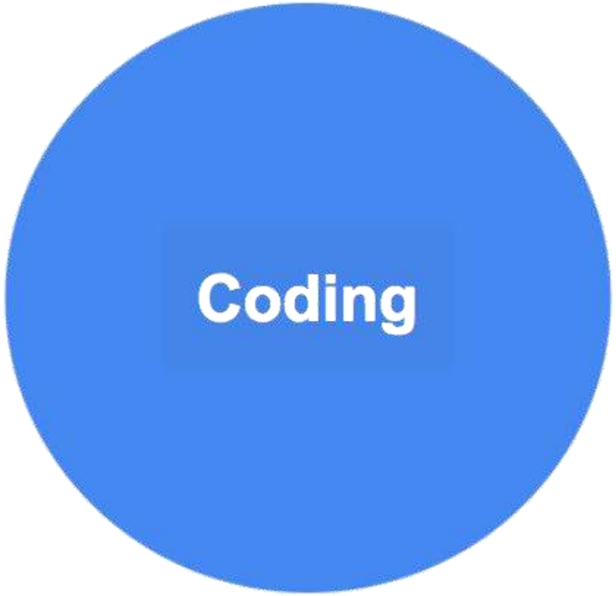


- **Ranked 1st** - Minutiae Interoperability Exchange (MINEX III, Ongoing MINEX)
- **Ranked 1st** - NIST Proprietary Fingerprint Template Test Phase II (PFT II)
- **Ranked 1st** - NIST Evaluation of Latent Fingerprint Technologies: Extended Feature Sets (ELFT-EFS Image Only)
- **Ranked 2nd** - Fingerprint Vendor Technology Evaluation 2012 (FPVTE)
- **Ranked 1st** - Face Recognition Vendor Tests (FRVT), December 2017
- **Ranked 2nd** – Face in Video Evaluation (FIVE)
- **Ranked 1st** – Iris Recognition Vendor Test (IREX IV)
- **Ranked 1st** - Tattoo Recognition Technology – Challenge (Tatt-C 2015)

# DATA

# 3

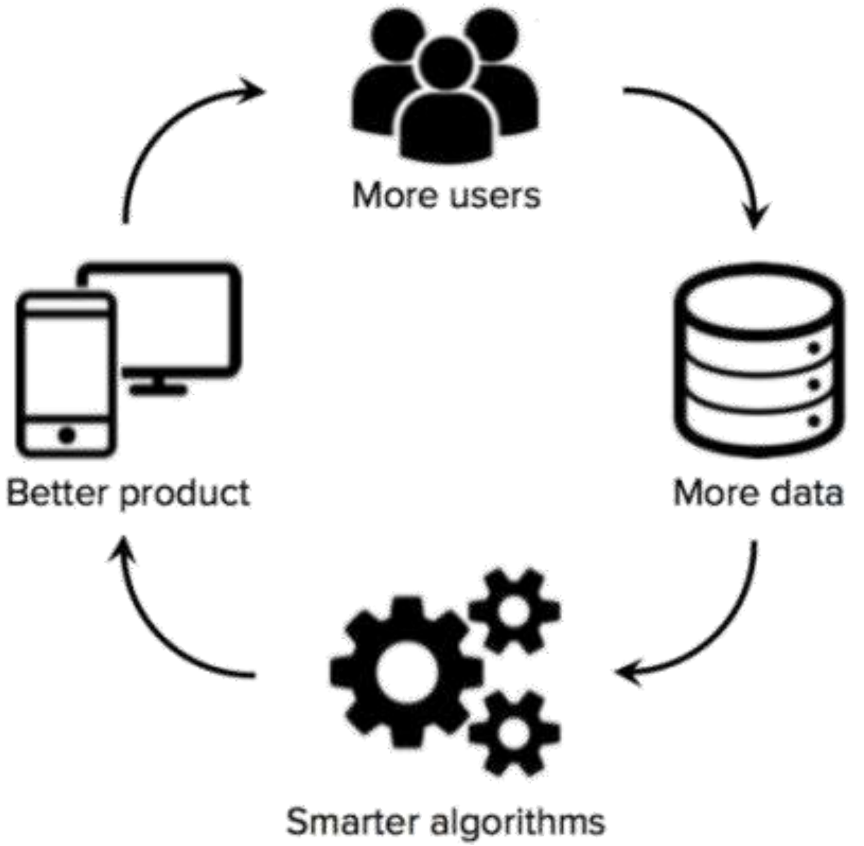
# HOW TO ACHIEVE?




What people think AI is about


The reality

# DATA



# Data and label for this session

## Face images coming from the Web :

http://www.cbsr.ia.ac.cn/english/CASIA-WebFace-Database.html

♦ Dong Yi, Zhen Lei, Shengcai Liao and Stan Z. Li, “Learning Face Representation from Scratch”. arXiv preprint arXiv:1411.7923. 2014.

## Pre-processed :

centred on the face, 48*48 pixels, grey levels.

## Labels :

gender automatically deduced from first name

## 4 databases :

training : 1000, 10k and 100k
testing : 10k

## Goal :

finding Gender from Image

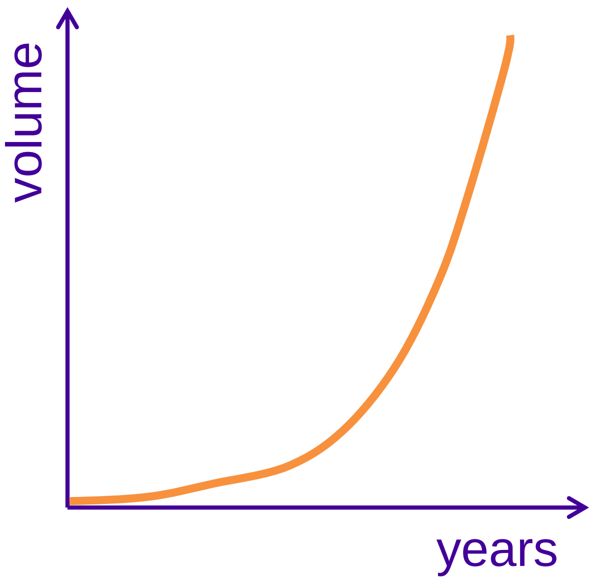
# Learning FRAMEWORKs

# 4

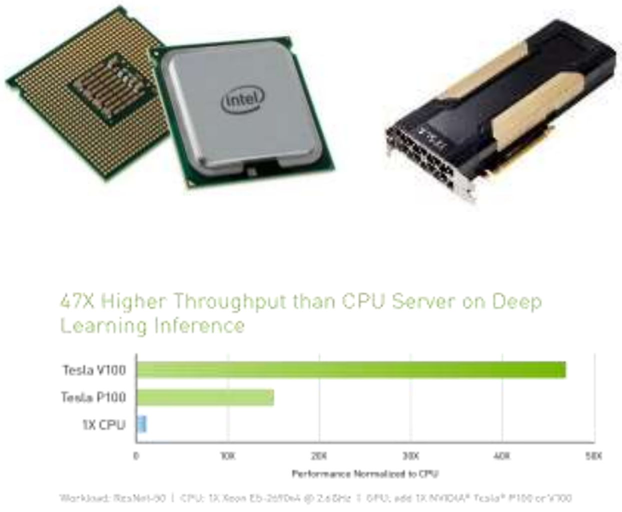
# WHY NOW?

## Three major recent evolutions

### Large Data Set




### Computation Power




### Deep Learning Frameworks available in OpenSource

# Deep Learning Frameworks

Chainer

Microsoft CNTK

dmlc mxnet

Caffe2

K

theano

torch

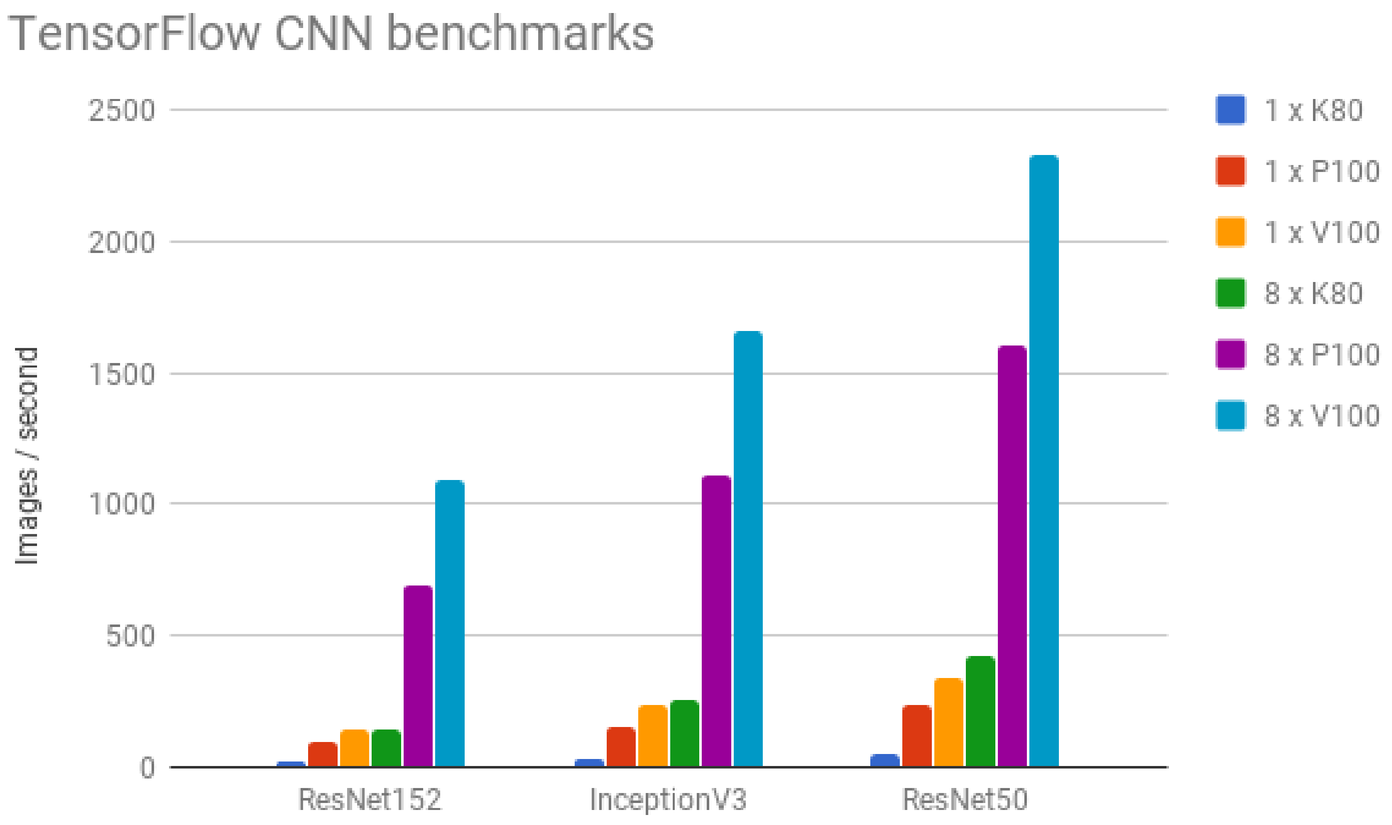
Caffe

TensorFlow

# Deep learning software

**cuDNN is a highly optimized GPU library for NVIDIA units that allows deep learning networks to be trained more quickly**

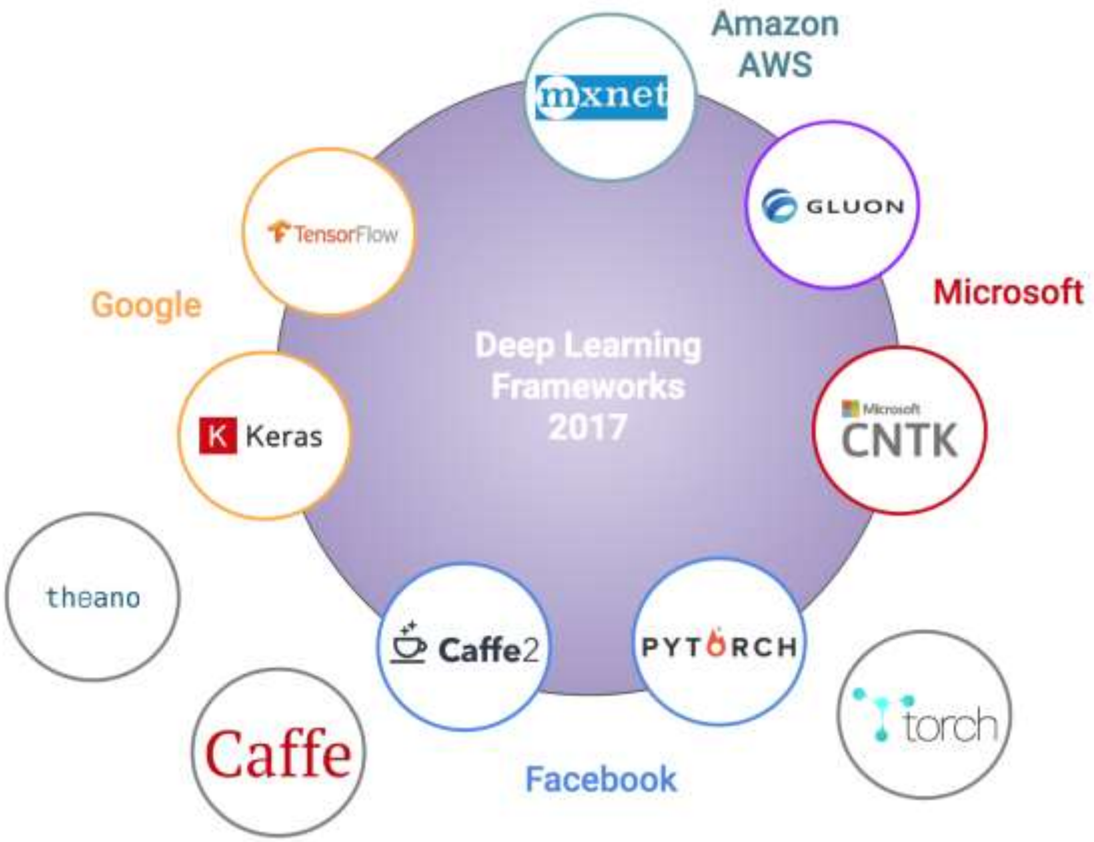
It can dramatically accelerate the performance of a deep network and is often called by the other packages above.



# Deep Learning High Level Frameworks

Keras is a Python library that runs on top of either Theano or TensorFlow that allows one to quickly define a network architecture in terms of layers and also includes functionality for image and text preprocessing

Gluon is a framework built on top of MxNet or CNTK. It allows dynamic computation graph.

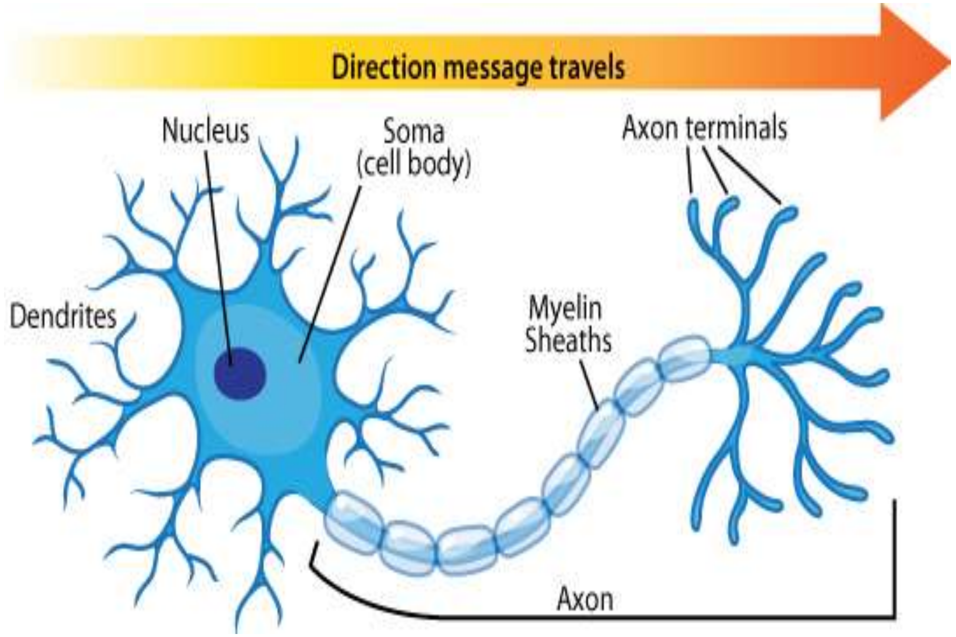


# Neural NETWORK

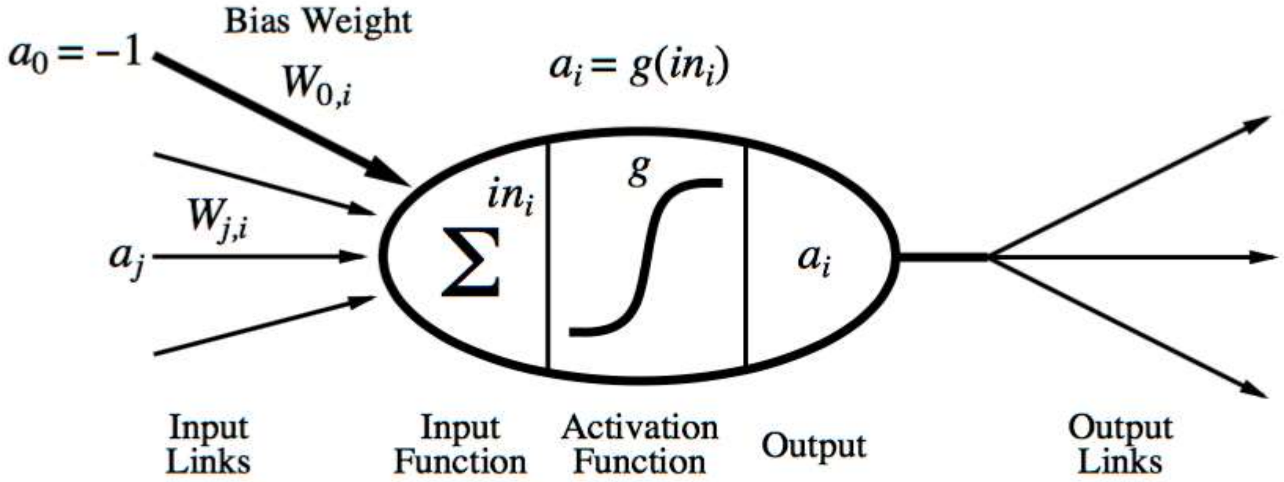
# 5

# Neural Network: A Neuron

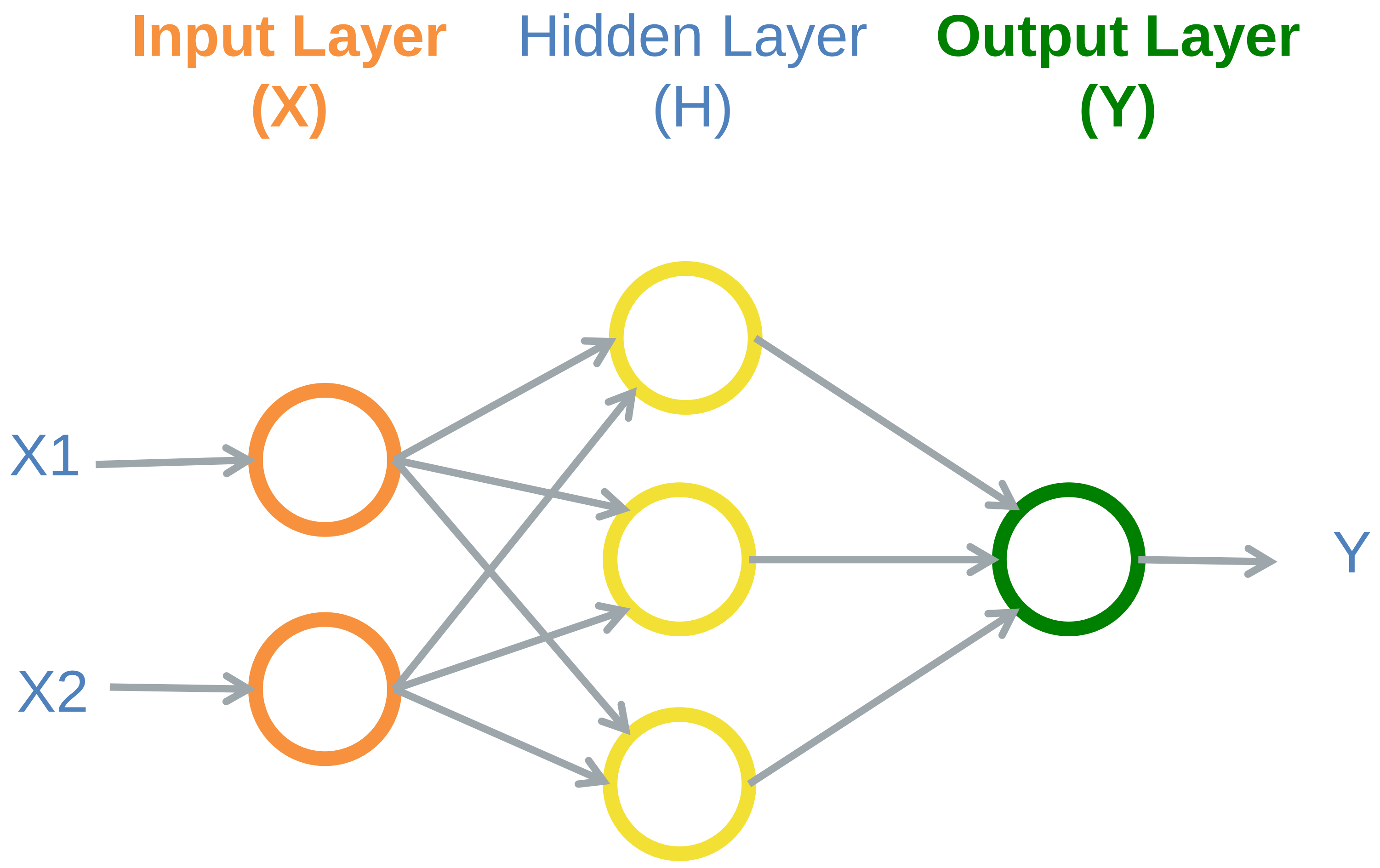
**A neuron is a computational unit in the neural network that exchanges messages with each other.**




$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$

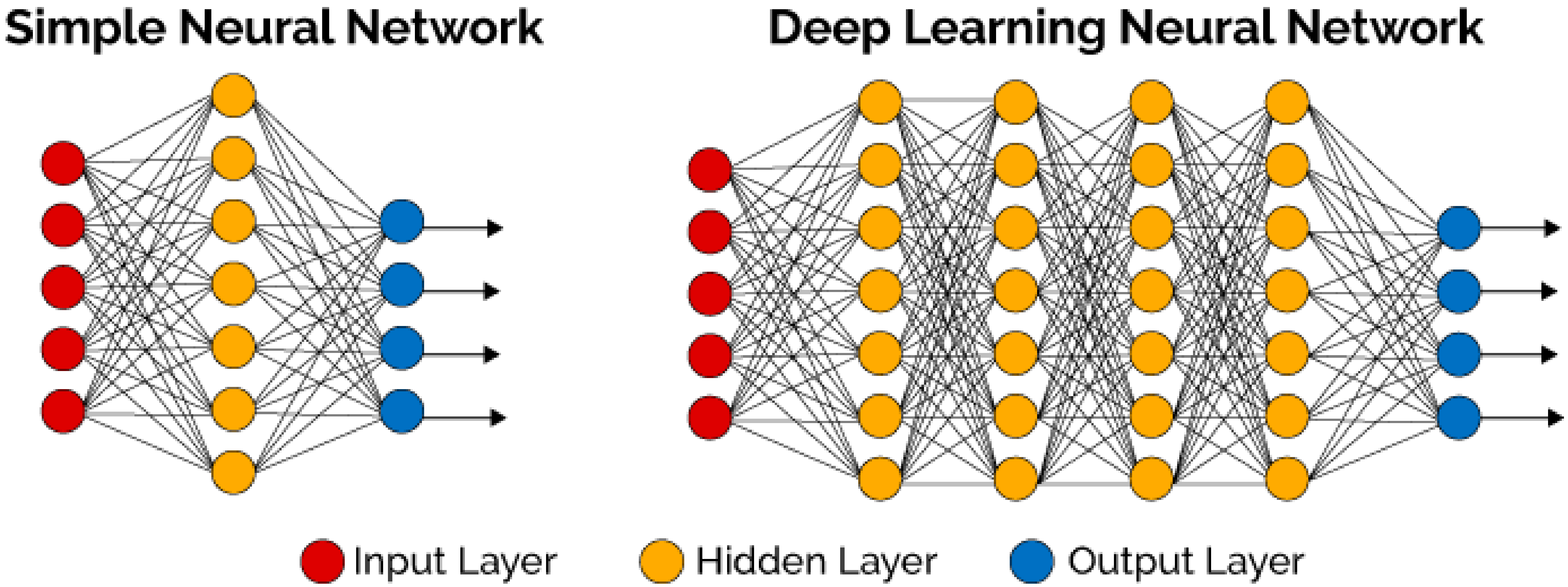


# Neural Networks



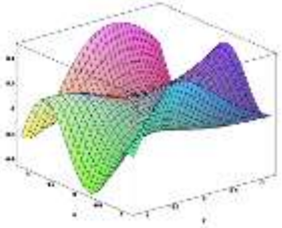

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU

# A deep neural network

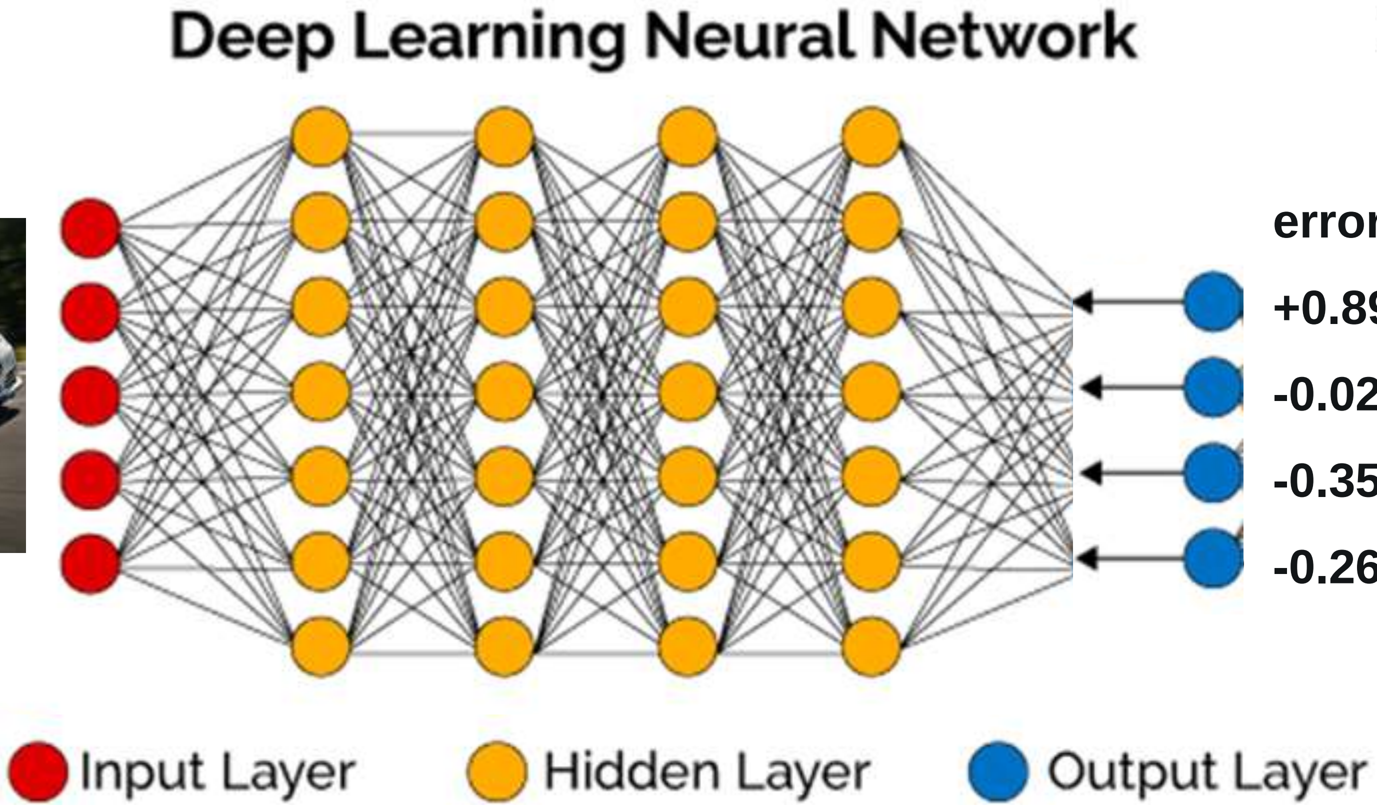


# TRAINING PROCESS

$\boldsymbol{x}$ —— Neural network ——→ $\mathbf{f}(\boldsymbol{x};\boldsymbol{\theta})$ —— Cost: $C_x(\theta)$ ——→ $\hat{\boldsymbol{y}}$



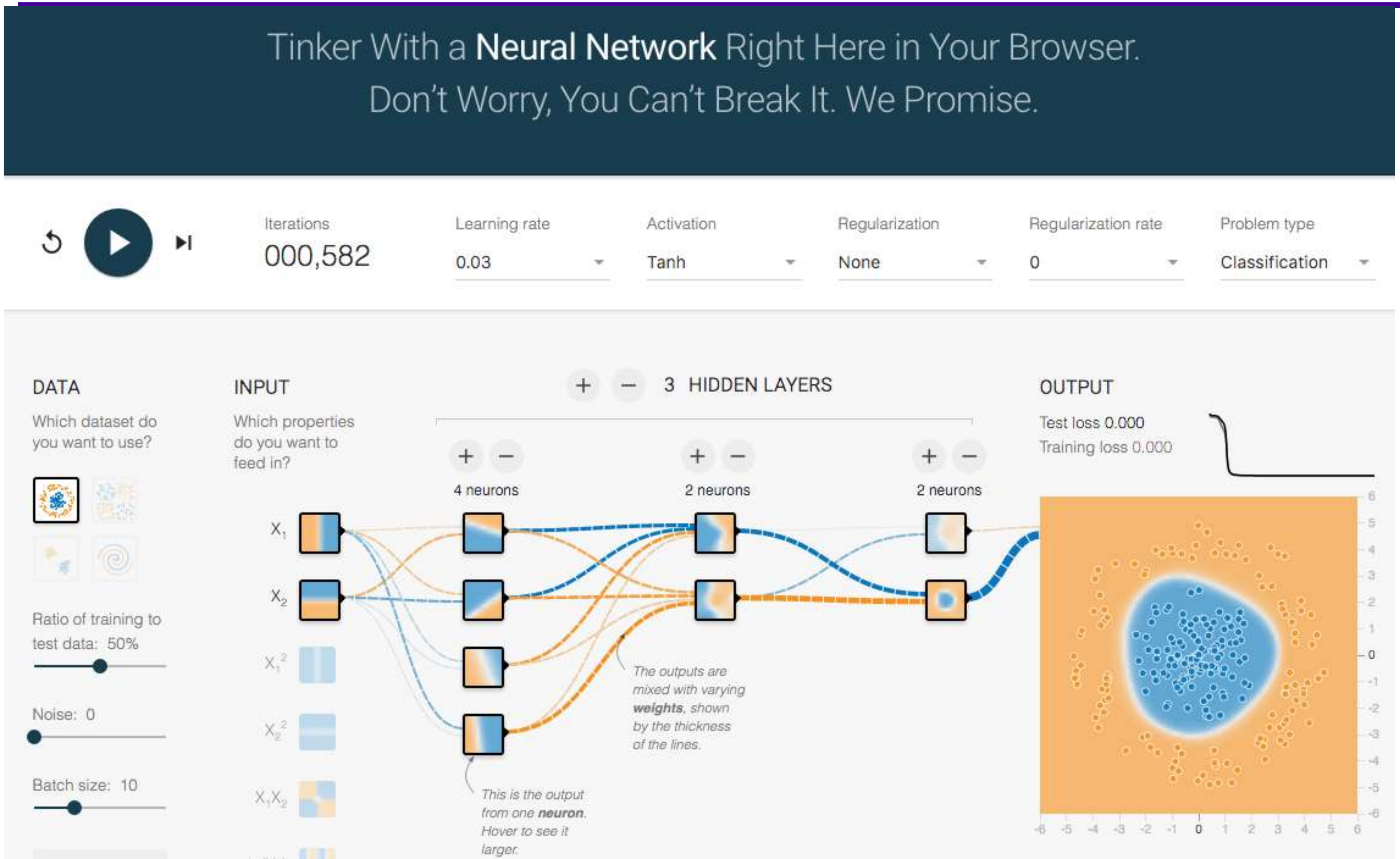
Pixels




| error | Labels |
|---|---|
| +0.89 | 1.0 car |
| -0.02 | 0.0 truck |
| -0.35 | 0.0 airplane |
| -0.26 | 0.0 ship |

# TensorFlow Playground

http://playground.tensorflow.org/



# Tensor Flow

# 5

# Big picture

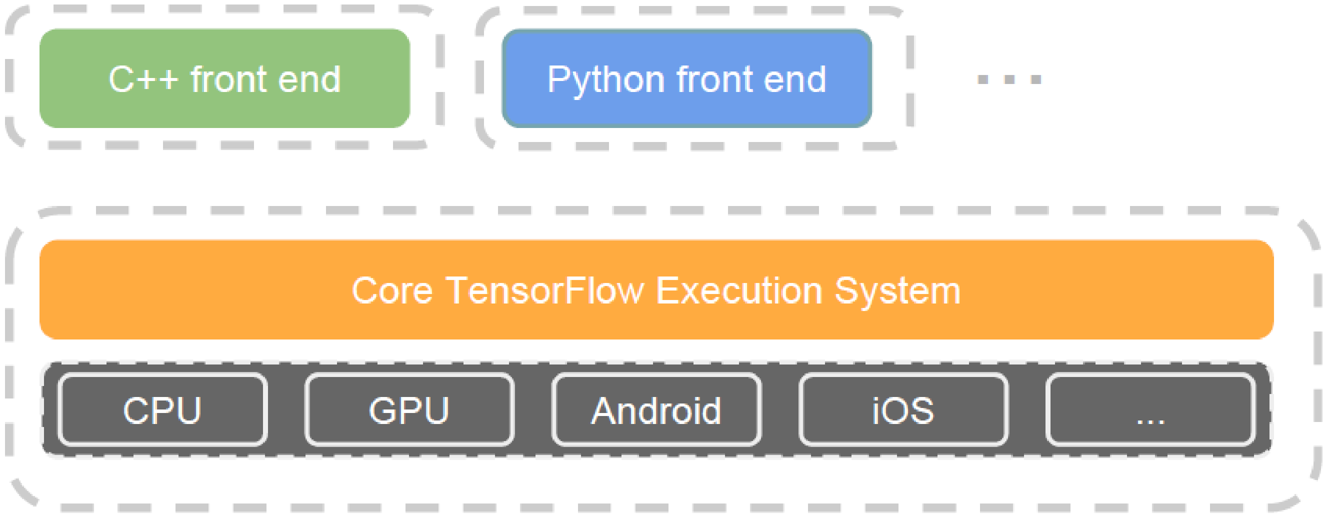
## TensorFlow is a deep learning framework released by Google in November 2015

## TF is open source

## Software architecture

Core in C++

Front ends in Python and C++

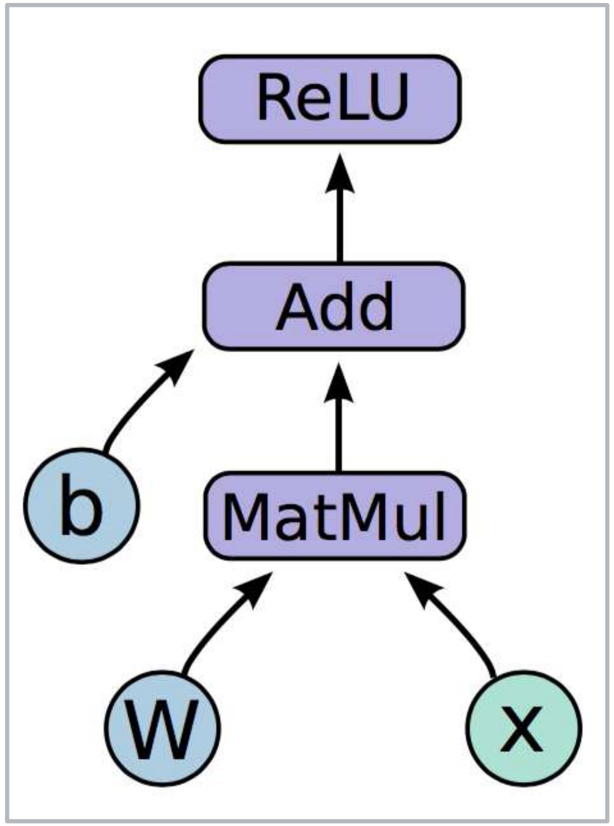


# computational graph

**Graph nodes are operations which have any number of inputs and outputs**

**Graph edges are tensors (n-dimensional arrays) which flow between nodes**

$$h_i = \mathrm{ReLU}(Wx + b)$$



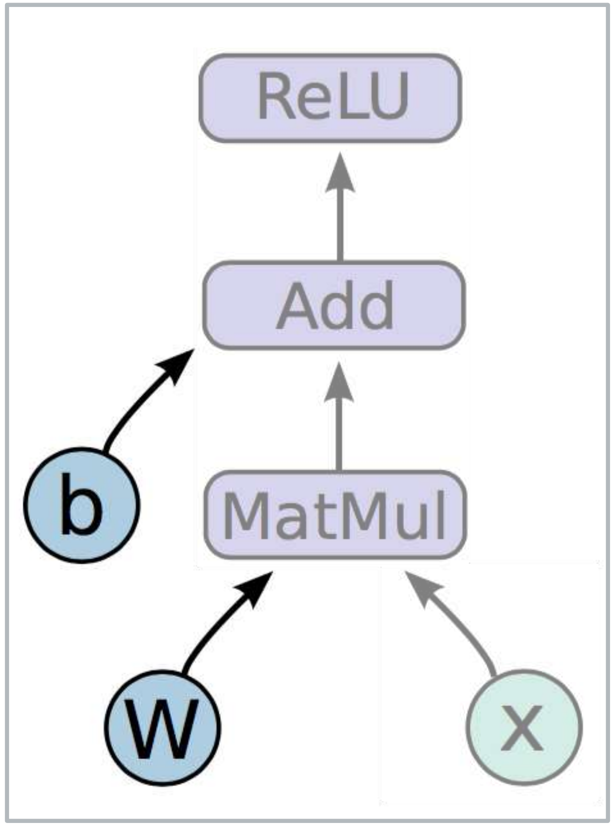
# computational graph

**Graph nodes are operations which have any number of inputs and outputs**

**Graph edges are tensors (n-dimensional arrays) which flow between nodes**

$$h_i = \mathrm{ReLU}(Wx + b)$$

**Variables** are nodes which output their current value. (Parameters to be learned)



# Computational graph

**Graph nodes are operations which have any number of inputs and outputs**

**Graph edges are tensors (n-dimensional arrays) which flow between nodes**

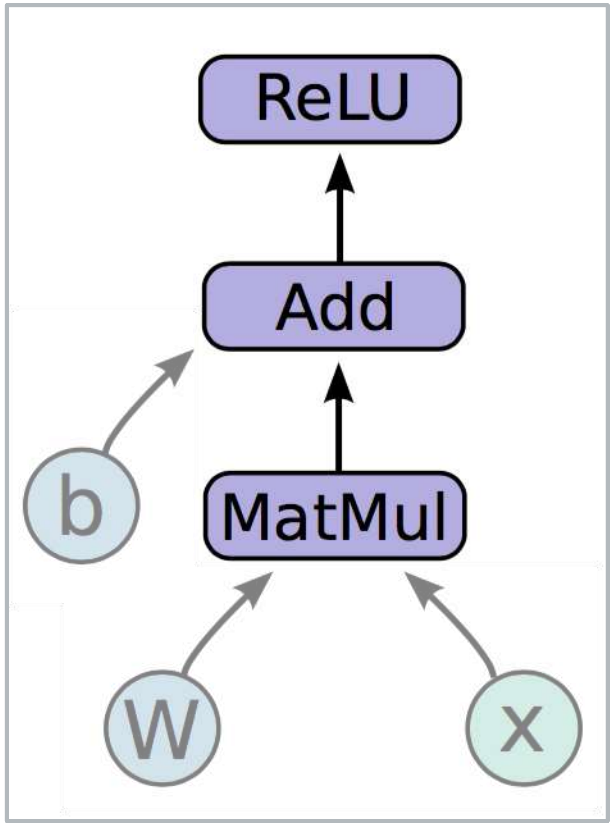
$$h_i = \mathrm{ReLU}(Wx + b)$$

**Mathematical operations:**

**MatMul:** Multiply two matrix values.

**Add** elementwise (with broadcasting).

**ReLU:** Activate with elementwise rectified linear function.



# A large variety of operators is available

**Elementwise math:** cos, exp, log, erf, modulo, abs, sqrt, pow, ceil, floor …

**Linear algebra:** matrix multiplication, inverse, Cholesky, determinant…

**Logical operators:** and, or, xor

**Comparison operators:** >, <, =…

**Reduction operators:** min, max, argmin, argmax, mean, sum, product…

**Slicing/joining:** operators: split, concatenate, tile, pad…

**Image operators:** read/write jpg/png, crop, resize, flip, flop, brightness/contrast/hue adjustment…

**Neural network:** convolution, pooling, ReLU, sigmoid, dropout, batch normalization, local normalization

# Python front end

**Everything is writing python code, no prototype files or anything**

1. **Build python functions describing a graph**

Graph contains parameter specifications

Graph contains model architecture

2. **Define optimization process**

Define values you want to optimize

Attach an optimizer

3. **Fetch and feed data**

…

**You write your optimization loop in python**

You can execute whatever python code want to in it!

# Monitoring

**TensorFlow allows you to monitor pretty much everything in real-time**

Values,

Histograms,

Proportion of non-zero values,

Images.

**You just have to**

Start a web server on the computer where TensorFlow is running

Access to a web page from your own computer (with Chrome)

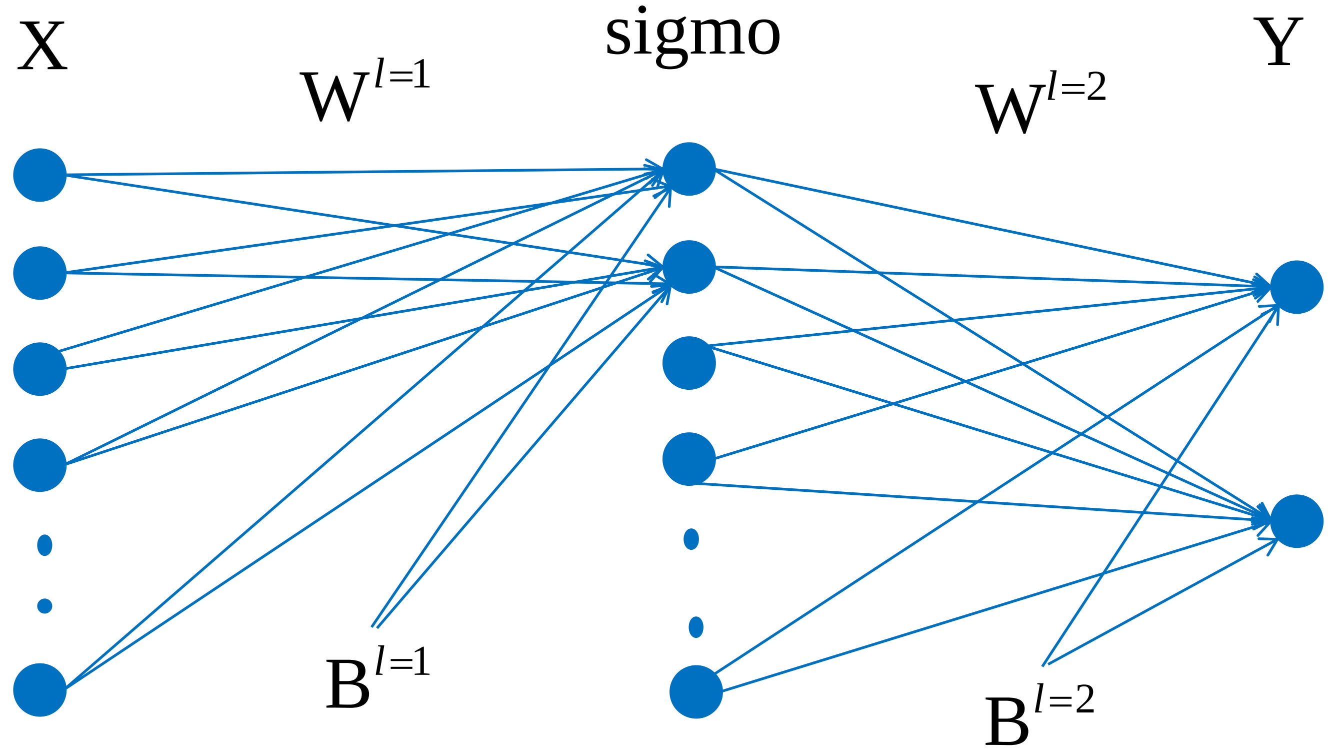
**[TensorBoard]**

# Gender Classification with TensorFlow and Python

- nn1
  - the simpliest script
- nn2
  - logs in tensorboard
- nn3
  - Compute accuracy on a test database
- nn4
  - use convolutional network
- nn5
  - use dropout

- From Session2 to Session3
  - your turn => learn a CNN and fool it !

# nn1 : A simple perceptron




Input layer
2304 neurons

Hidden layers
50 neurons

Output layer
2 neurons

# nn1 : simplest.py

```python
import tensorflow as tf
import numpy as np

# nombre d images
nbdata = 1000
trainDataFile = '../DataBases/data_1k.bin'
LabelFile = '../DataBases/gender_1k.bin'

# taille des images 48*48 pixels en niveau de gris
dim = 2304
f = open(trainDataFile, 'rb')
data = np.empty([nbdata, dim], dtype=np.float32)
for i in range(nbdata):
   data[i, :] = np.fromfile(f, dtype=np.uint8,
count=dim).astype(np.float32)
f.close()

f = open(LabelFile, 'rb')
label = np.empty([nbdata, 2], dtype=np.float32)
for i in range(nbdata):
   label[i, :] = np.fromfile(f, dtype=np.float32, count=2)
f.close()

class fc_layer(tf.Module):
   def __init__(self, input_dim, output_dim):
      w_init = tf.random.truncated_normal([input_dim, output_dim],
                                          stddev=0.1)
      self.w = tf.Variable(w_init)
      print('w      ', self.w.get_shape())
      b_init = tf.constant(0.0, shape=[output_dim])
      self.b = tf.Variable(b_init)
      print('b      ', self.b.get_shape())

   def __call__(self, x):
      return tf.matmul(x, self.w) + self.b

class SimpleNet(tf.Module):
   def __init__(self, input_dim):
      self.fc1 = fc_layer(input_dim,50)
      self.fc2 = fc_layer(50,2)

   def __call__(self, x):
      x = self.fc1(x)
      x = tf.nn.sigmoid(x)
      x = self.fc2(x)
      return x

def train_one_step(model, optimizer, image, label):
   with tf.GradientTape() as tape:
      y = model(image)
      loss = tf.reduce_sum(tf.square(y - label))
      grads = tape.gradient(loss, model.trainable_variables)
      optimizer.apply_gradients(zip(grads, model.trainable_variables))
   return loss

optimizer = tf.optimizers.SGD(1e-5)
simple_model = SimpleNet(dim)

curPos = 0
batchSize = 256

for it in range(5000):
   if curPos + batchSize > nbdata:
      curPos = 0
   loss = train_one_step(simple_model, optimizer,
                data[curPos:curPos + batchSize, :],
                label[curPos:curPos + batchSize, :])

   curPos += batchSize
   if it % 100 == 0 or it < 10:
      print("it= %6d - loss= %f" % (it, loss.numpy()))
```

# nn1 : let's run

urd27@Deep6: ~/Partage/Stephane/td/nn1

```
18:43:07|nn1> python simplest.py
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcublas.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcudnn.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcufft.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
Tensor("truncated_normal:0", shape=(2304, 50), dtype=float32)
<tensorflow.python.ops.variables.Variable object at 0x7f7b7ce0f310>
Tensor("truncated_normal_1:0", shape=(50, 2), dtype=float32)
<tensorflow.python.ops.variables.Variable object at 0x7f7b586d38d0>
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 0 with properties:
name: GeForce GTX 1080
major: 6 minor: 1 memoryClockRate (GHz) 1.7335
pciBusID 0000:02:00.0
Total memory: 7.92GiB
Free memory: 7.81GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0:   Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX 1080, pci bus id: 0000:02:00.0)
it=      0 - loss= 181.141205
it=   1000 - loss= 35.203453
it=   2000 - loss= 39.134914
it=   3000 - loss= 53.792587
it=   4000 - loss= 23.412951
it=   5000 - loss= 30.071821
it=   6000 - loss= 47.140503
it=   7000 - loss= 20.690556
it=   8000 - loss= 26.863918
it=   9000 - loss= 45.297955
it=  10000 - loss= 21.248180
it=  11000 - loss= 26.216721
it=  12000 - loss= 42.632809
it=  13000 - loss= 18.814625
it=  14000 - loss= 23.940264
it=  15000 - loss= 42.456089
```

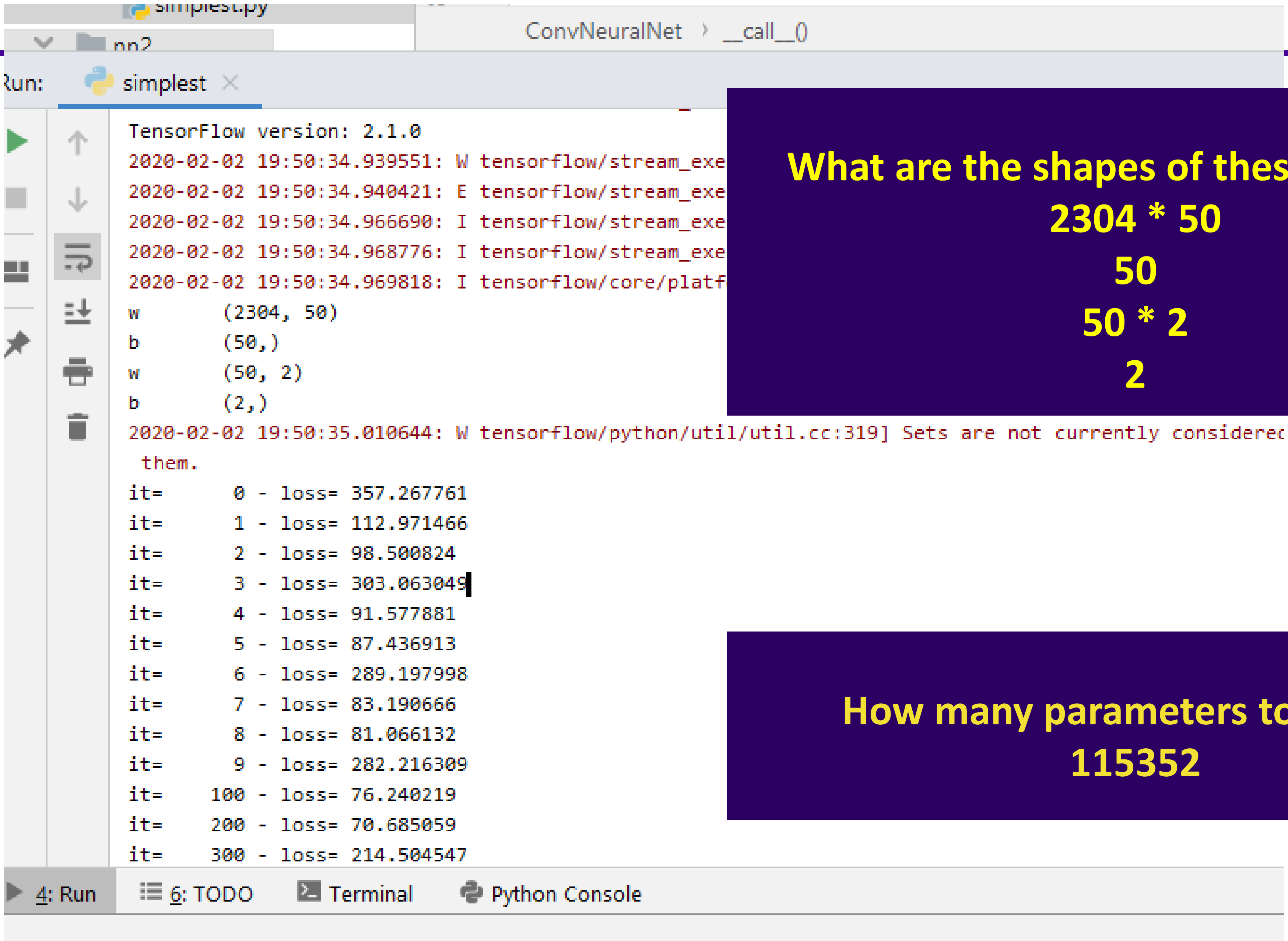
# nn1 : simplest.py

What are the shapes of these tensors ?

```
class fc_layer(tf.Module):
    def __init__(self, input_dim, output_dim):
        w_init = tf.random.truncated_normal([input_dim, output_dim], stddev=0.1)
        self.w = tf.Variable(w_init)
        print('w      ', self.w.get_shape())
        b_init = tf.constant(0.0, shape=[output_dim])
        self.b = tf.Variable(b_init)
        print('b      ', self.b.get_shape())

    def __call__(self, x):
        return tf.matmul(x, self.w) + self.b
```

How many parameters to learn ?

# nn1 : simplest.py




**What are the shapes of these tensors ?**

**2304 * 50**

**50**

**50 * 2**

**2**

**How many parameters to learn ?**

**115352**

# nn2 : simplest_v2.py

***Save and Load the network variables***

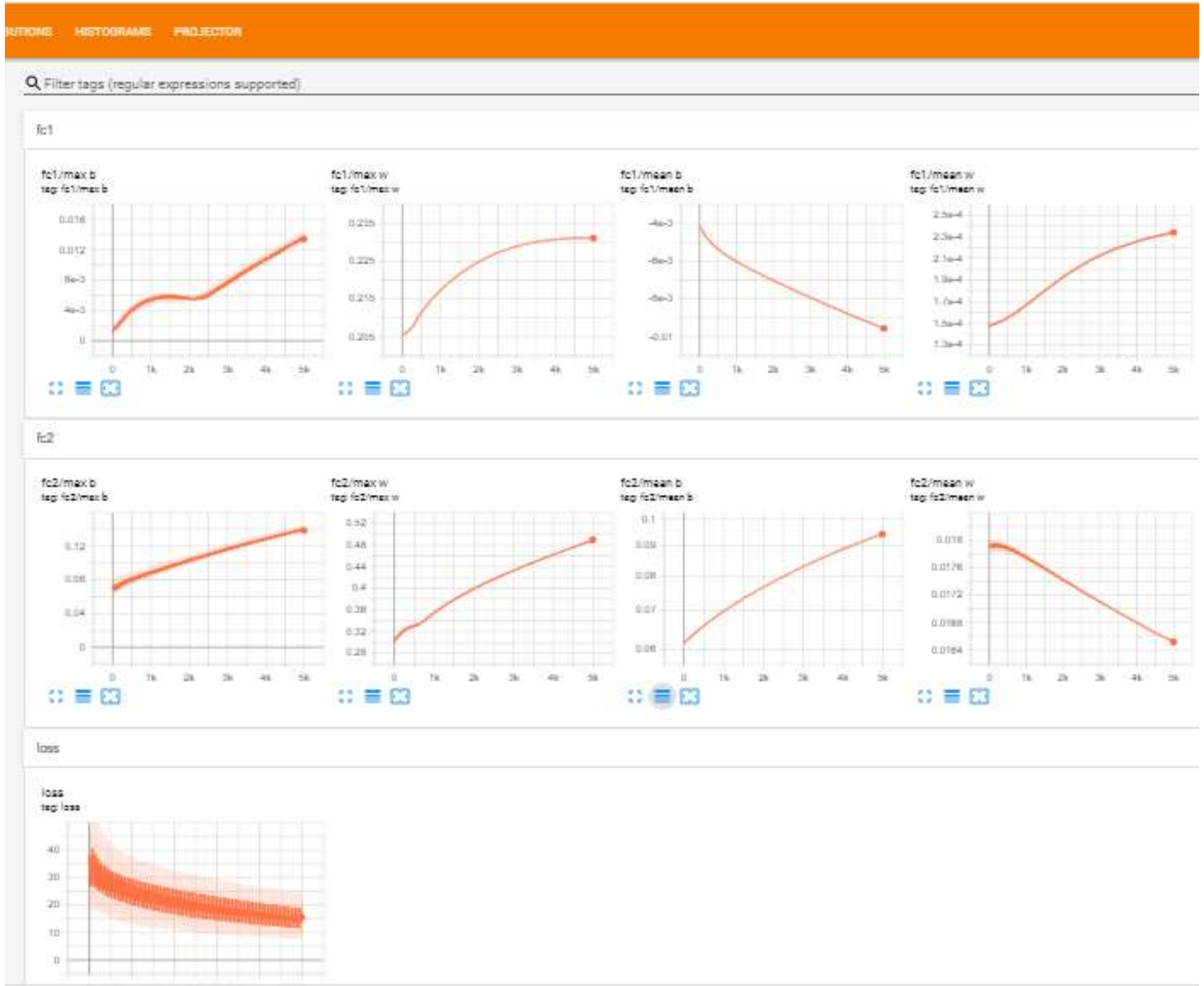
```
ckpt = tf.train.Checkpoint(step,optimizer,net)
ckpt.save('filename')
ckpt.restore('filename-step')
```

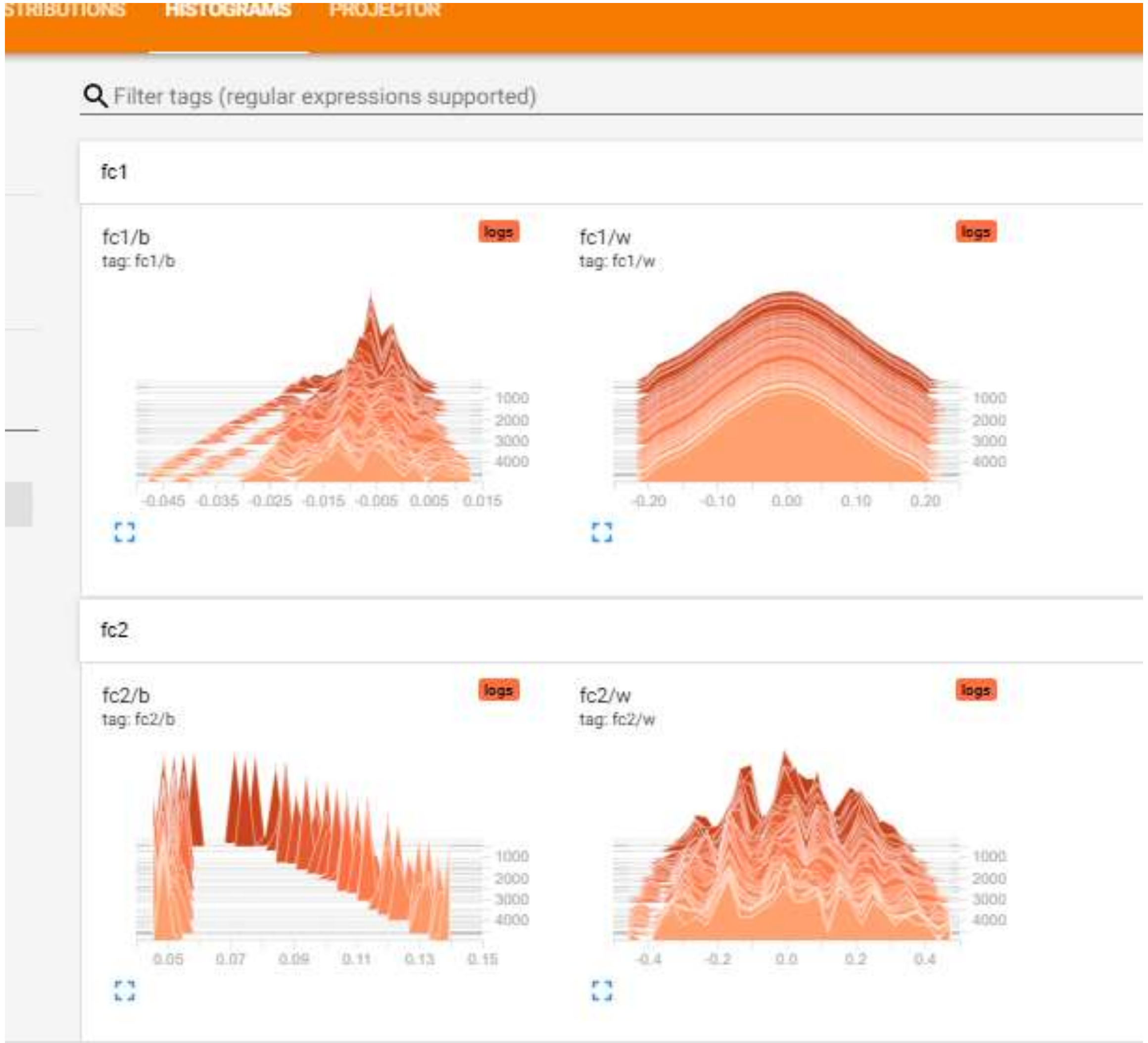
***Following the learning process :***
***Summary***

```
t = tf.summary.create_file_writer('logs')

with t.as_default():

    tf.summary.scalar(name, tensor)
    tf.summary.histogram(name, tensor)
```

# nn2 : simplest_v2.py

```
tensorboard --logdir nn2 --port 8888
```



# nn3 : perceptron for gender classification … does it work ?

## *3 datasets for learning and 1 for testing*

```
train = ds.DataSet('../DataBases/data_1k.bin','../DataBases/gender_1k.bin',1000)
train = ds.DataSet('../DataBases/data_10k.bin','../DataBases/gender_10k.bin',10000)
train = ds.DataSet('../DataBases/data_100k.bin','../DataBases/gender_100k.bin',100000)
test = ds.DataSet('../DataBases/data_test10k.bin','../DataBases/gender_test10k.bin',10000)
```

## *Compute Accuracy*

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(labels, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

## *On a complete dataset*

```
def mean_accuracy(self, model):
    acc = 0
    for i in range(0, self.nbdata, self.batchSize):
        curBatchSize = min(self.batchSize, self.nbdata - i)
        images = self.data[i:i+curBatchSize,:]
        labels = self.label[i:i+curBatchSize,:]
        y = model(images, False)
        correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(labels, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        acc += accuracy * curBatchSize
    acc /= self.nbdata
    tf.summary.scalar('Accuracy %s'%self.name, acc)
    return acc
```

# CNN Architecture

6

- Fully Connected layer
- Activation layer
- Convolutional Layer
- Pooling Layer
- Loss Layer

# Fully Connected layer

# 6.1

# Fully Connected Layer

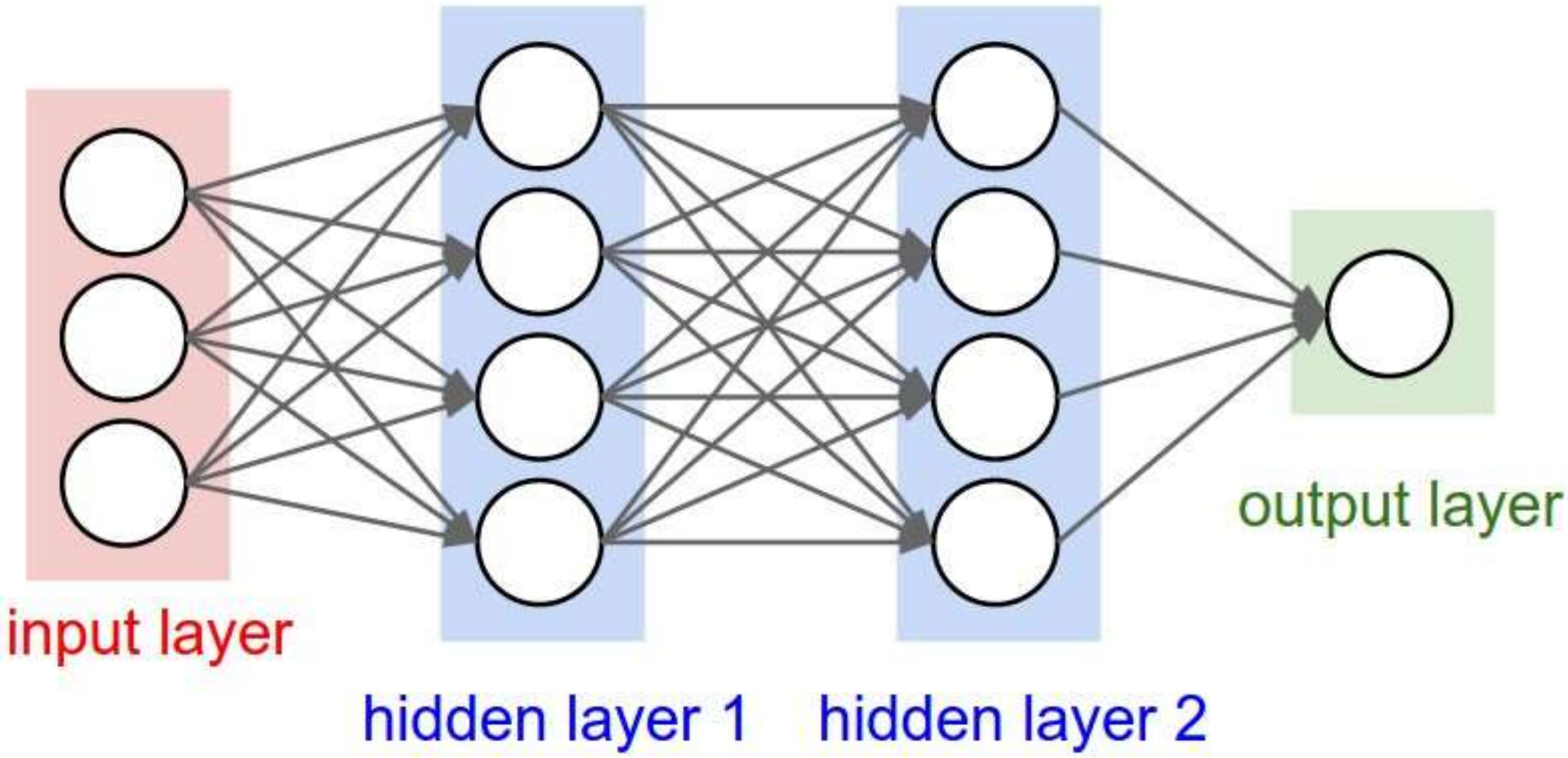
All the neurons of a layer are connected to the next layer.

If $X$ is the input vector of values, $Y$ the output vector and $W$ the parameter matrix.

The computation of $Y$ is a simple scalar product. $Y = W.X$

FC layers (Fully connected) are often named IP layers (Inner Product)

# A regular 3-layer Fully Connected Neural Network




http://cs231n.github.io/convolutional-networks/

# Activation Functions

# 6.2

# Sigmoid function

http://mathworld.wolfram.com/SigmoidFunction.html

$$f(u) = \frac{1}{1+e^{-\beta u}} = \frac{1}{1+e^{-u}}, \text{ for simplicity set } \beta = 1$$

$$\frac{df(u)}{du} = f'(u)$$

*Hence*

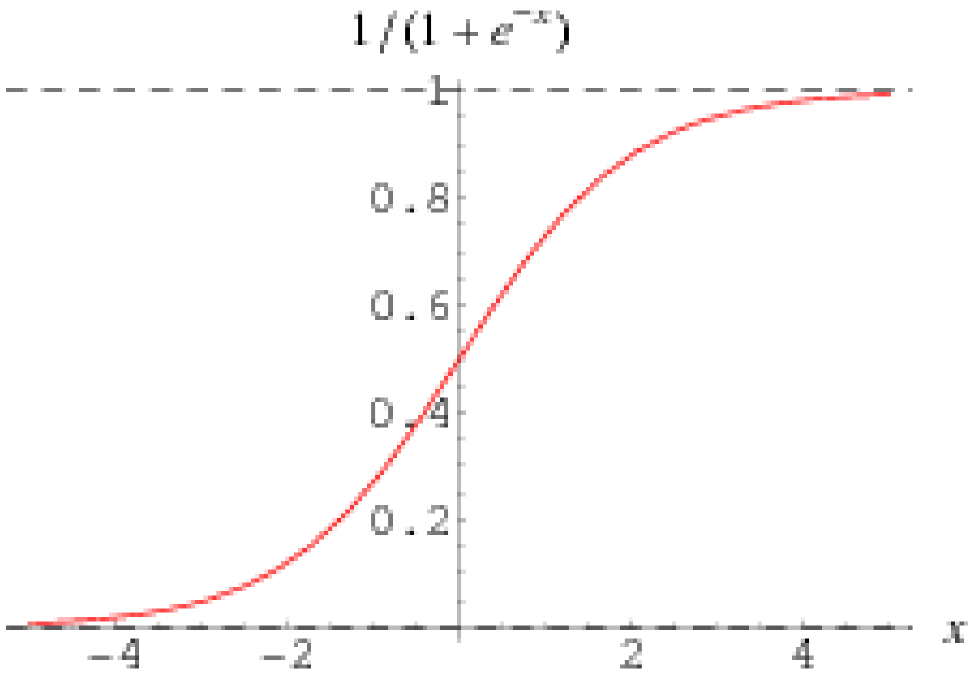
$$f'(u) = \frac{df(u)}{du} = \frac{d\left(\frac{1}{1+e^{-u}}\right)}{d(1+e^{-u})}\frac{d(1+e^{-u})}{du}, \text{(using chain rule)}$$

$$f'(u) = \frac{-1}{(1+e^{-u})^2}(-e^{-u}) = \frac{1}{(1+e^{-u})^2}e^{-u}$$

$$= \frac{1}{(1+e^{-u})}\frac{e^{-u}}{(1+e^{-u})} = \frac{1}{(1+e^{-u})}\frac{[(1+e^{-u})-(1+e^{-u})]+e^{-u}}{(1+e^{-u})}$$

$$= \frac{1}{(1+e^{-u})}\left(1-\frac{1}{(1+e^{-u})}\right) = f(u)(1-f(u))$$

$$\boxed{\text{Thus, } \frac{df(u)}{du} = f'(u) = f(u)(1-f(u))}$$






http://link.springer.com/chapter/10.1007%2F3-540-59497-3_175#page-1

# Rectified Linear Units

**Relu ( x ) = MAX ( 0 , x )**

**More efficient gradient propagation, derivative is 0 or constant, just fold into learning rate**

**More efficient computation: Only comparison, addition and multiplication.**

A variation : Leaky ReLU $f(x) = x$ if > 0 else $ax$ where $0 \le a <= 1$, so that derivate is not 0 and can do some learning for that case.

Lots of other variations

**Sparse activation: For example, in a randomly initialized networks, only about 50% of hidden units are activated (having a non-zero output)**

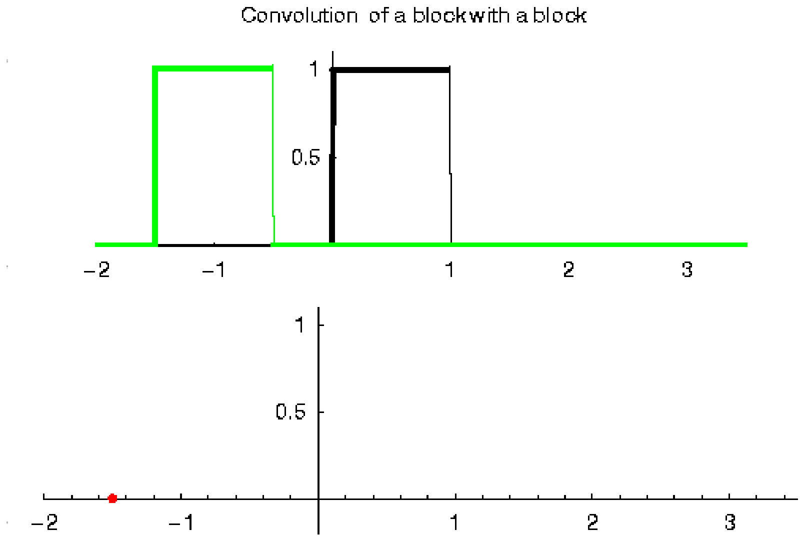
# Convolutional Layer

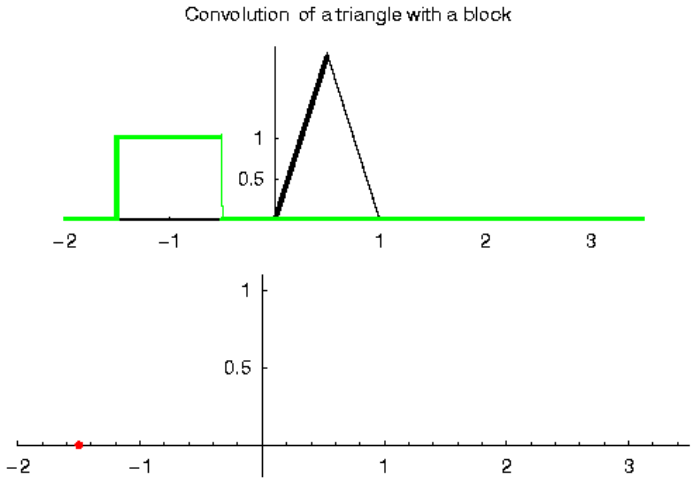
# 6.3

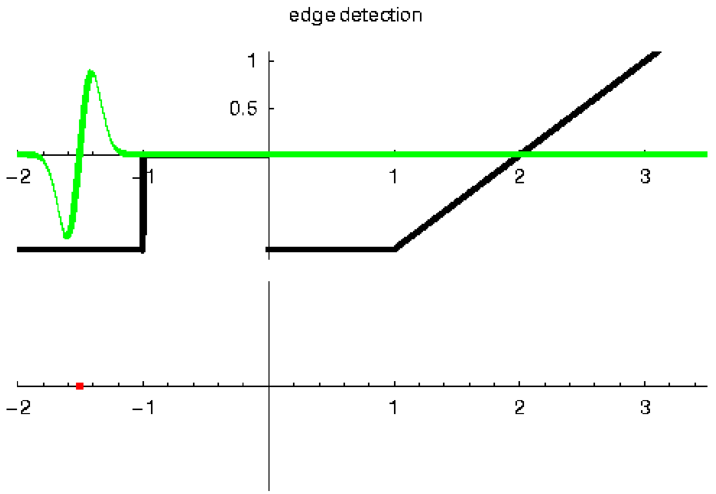
# Background

Convolution is an operation on two functions f and g, which produces a third function that can be interpreted as a modified ("filtered") version of f. In this interpretation we call g the *filter (or Kernel)*

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau) \cdot g(x - \tau)\, d\tau$$



# Background

For functions of a discrete variable *x*, i.e. arrays of numbers, the definition is:

$$f[x] * g[x] = \sum_{k=-\infty}^{\infty} f[k] \cdot g[x-k]$$

Finally, for functions of two variables *x* and *y* (for example images), these definitions become:

$$f(x,y) * g(x,y) = \int_{\tau_1=-\infty}^{\infty} \int_{\tau_2=-\infty}^{\infty} f(\tau_1,\tau_2) \cdot g(x-\tau_1, y-\tau_2)\, d\tau_1\, d\tau_2$$

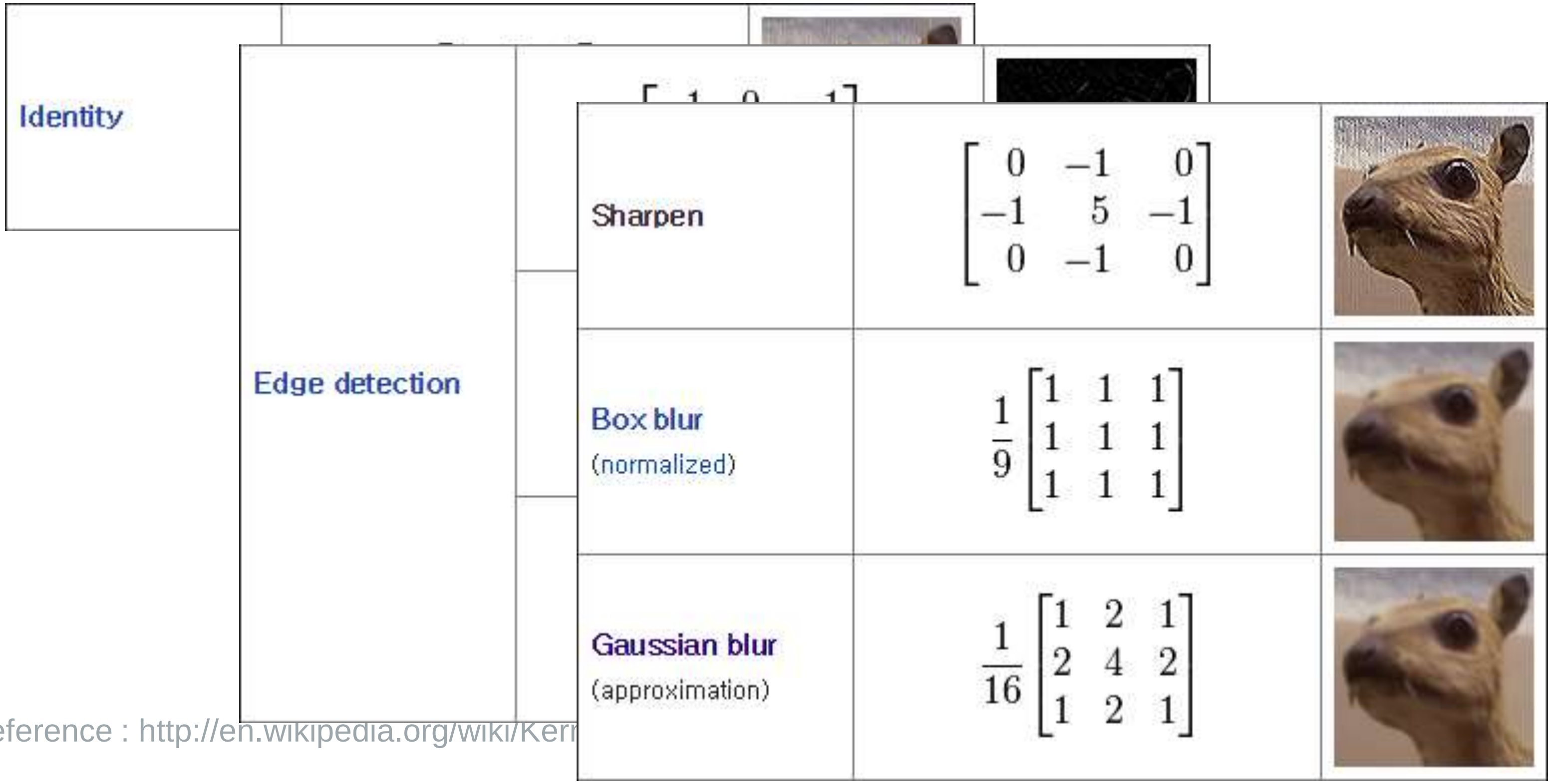
And :

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

## Convolutional Neural Networks

Variation of multi-layer neural networks

## Kernel (Convolution Matrix)

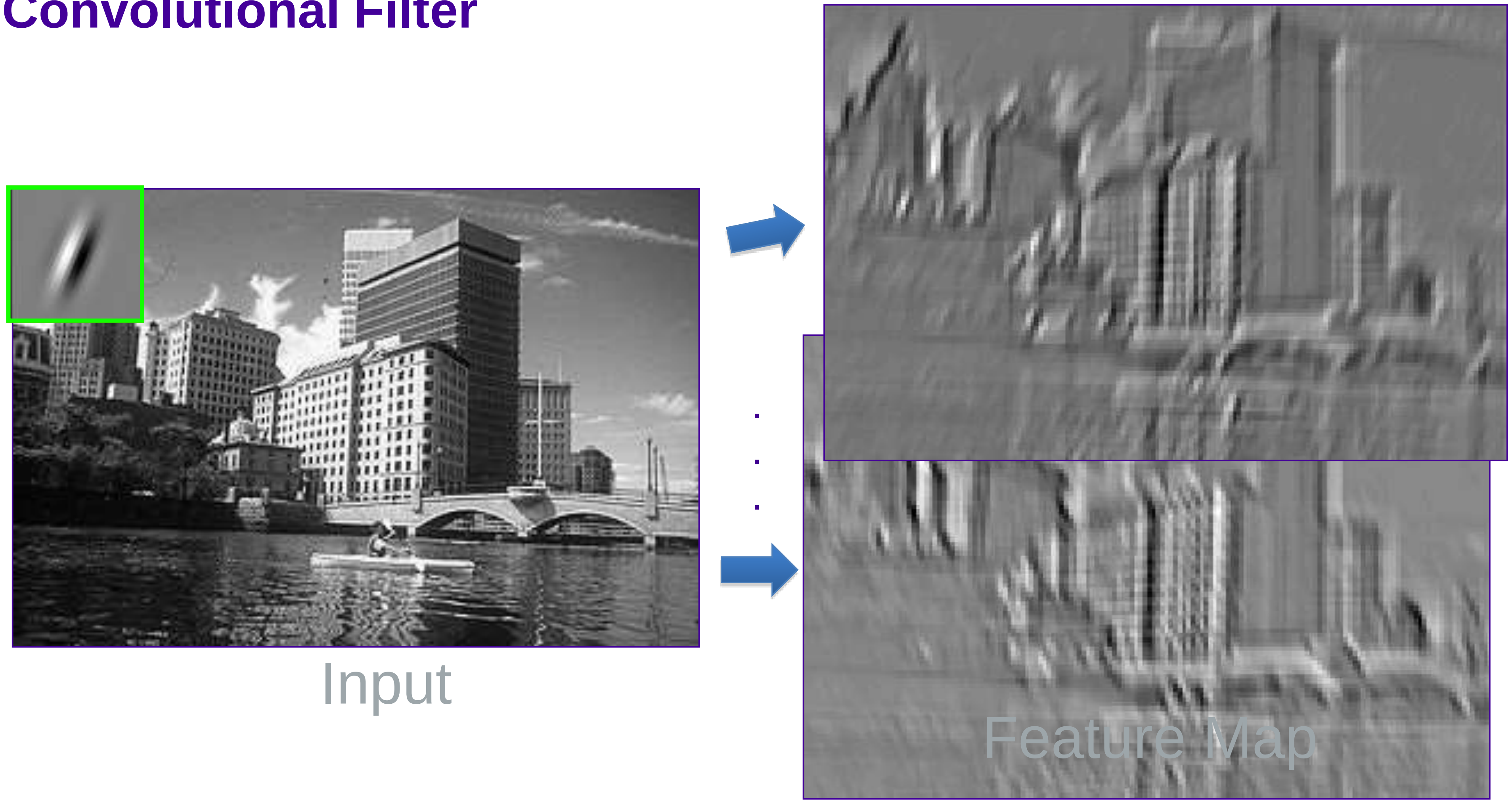


| | | |
|---|---|---|
| Identity | | |
| Edge detection | | |
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ | |
| Box blur (normalized) | $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ | |
| Gaussian blur (approximation) | $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ | |

reference : http://en.wikipedia.org/wiki/Kern

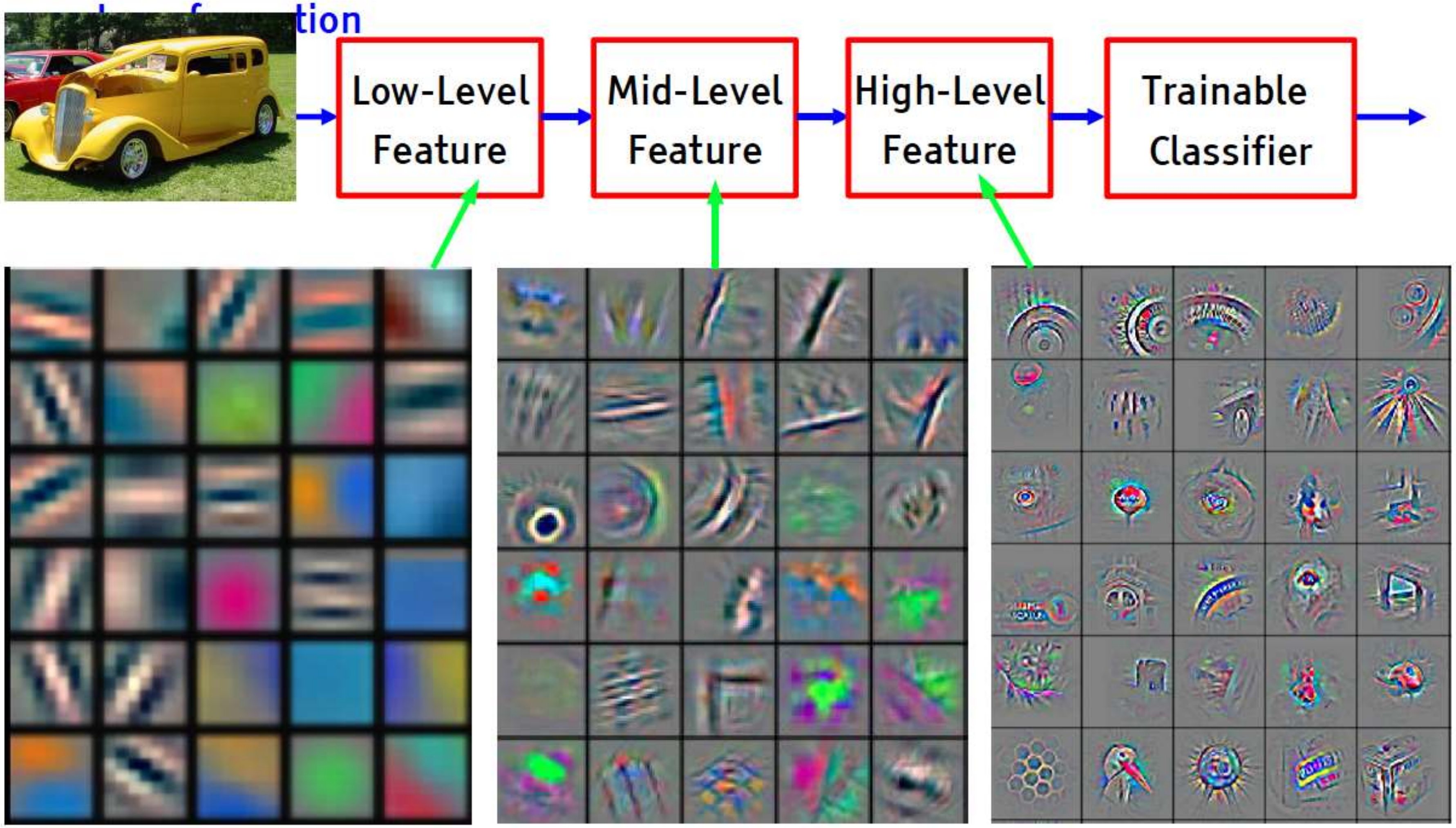
# Background

## Convolutional Filter



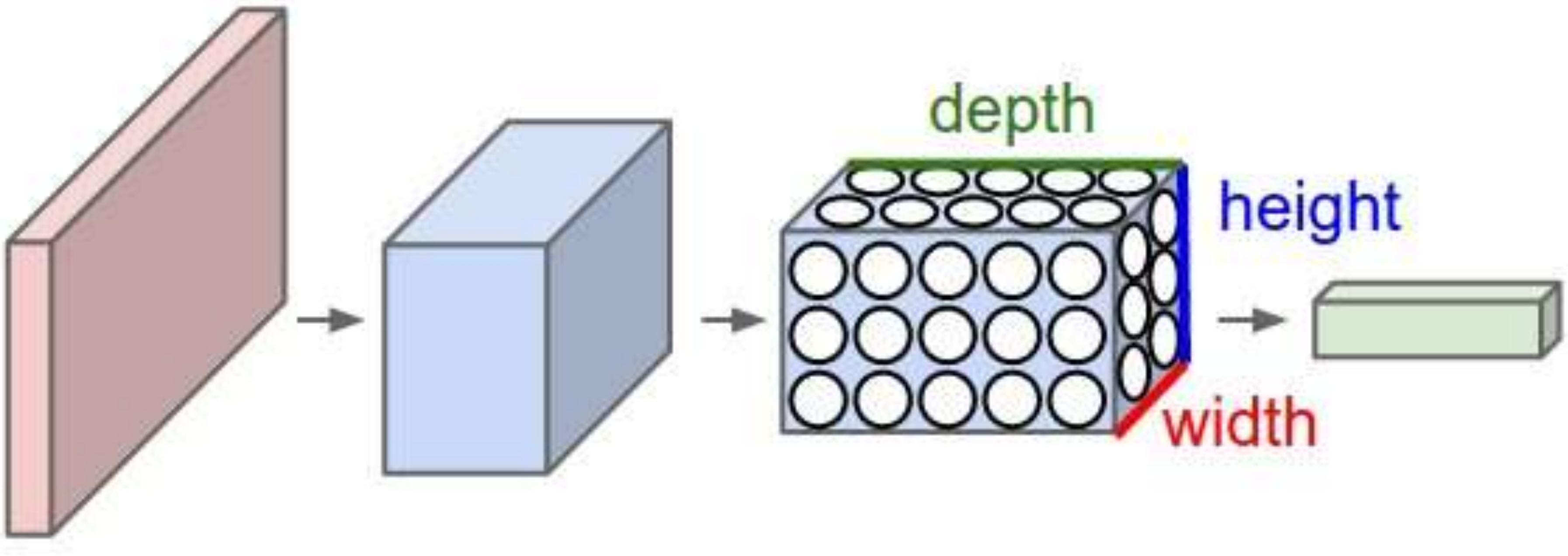

reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

# Deep Learning = Learning Hierarchical Representations



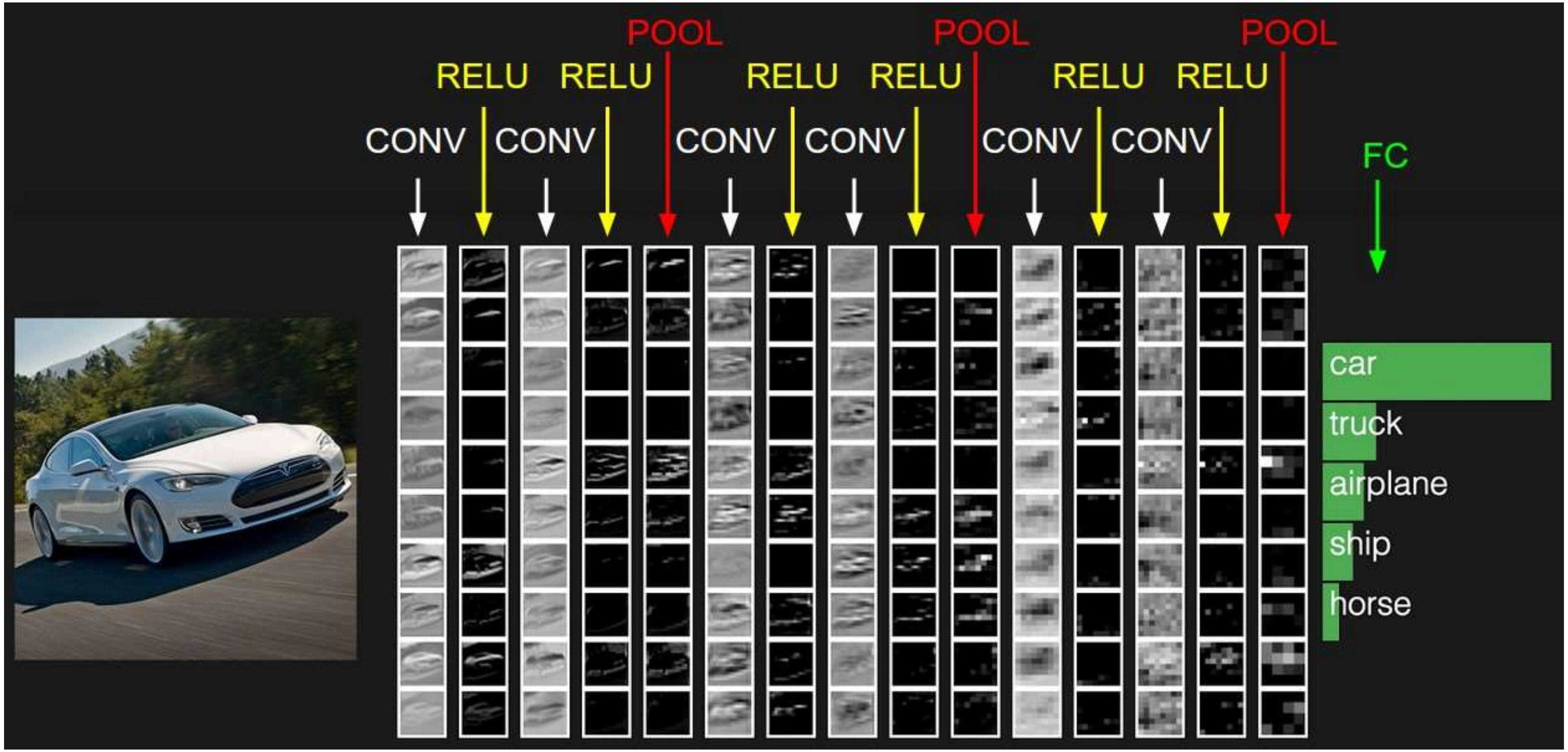

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# A ConvNet looks at data in three dimensions (width, height, depth)



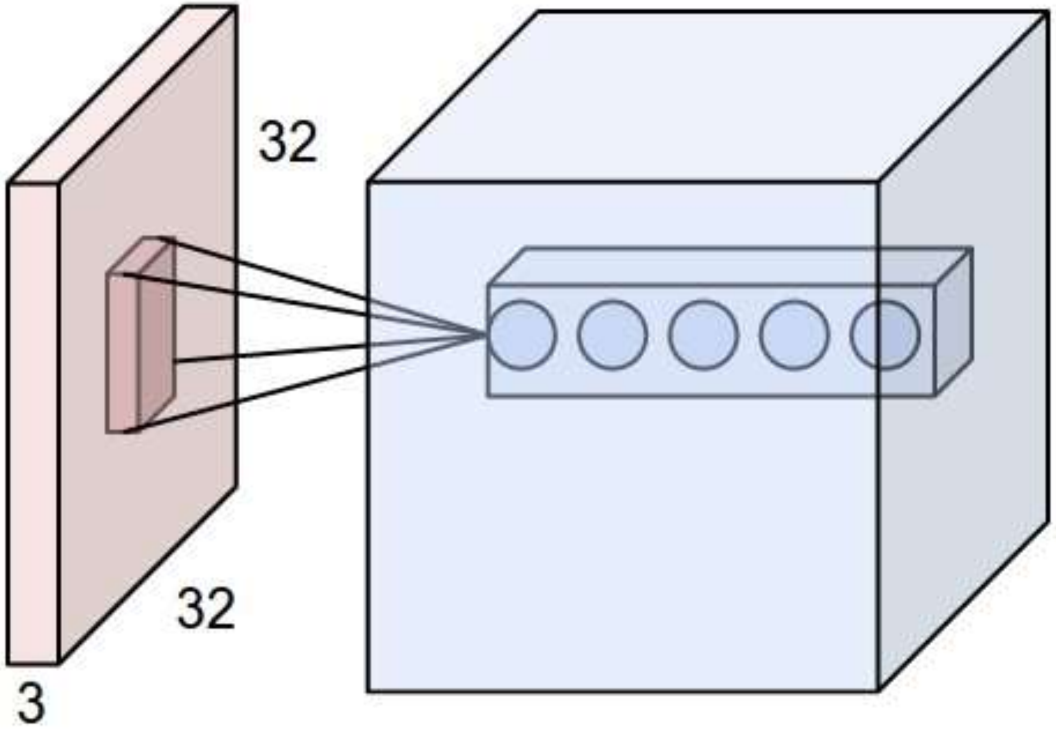

http://cs231n.github.io/convolutional-networks/

# The activations of an example ConvNet architecture.




http://cs231n.github.io/convolutional-networks/

# ConvNets




**Input : "InDim" images with size H,W**

**A Convolution Layer : "outDim" filters with kernel size InDim x KsizeH x KsizeW.**

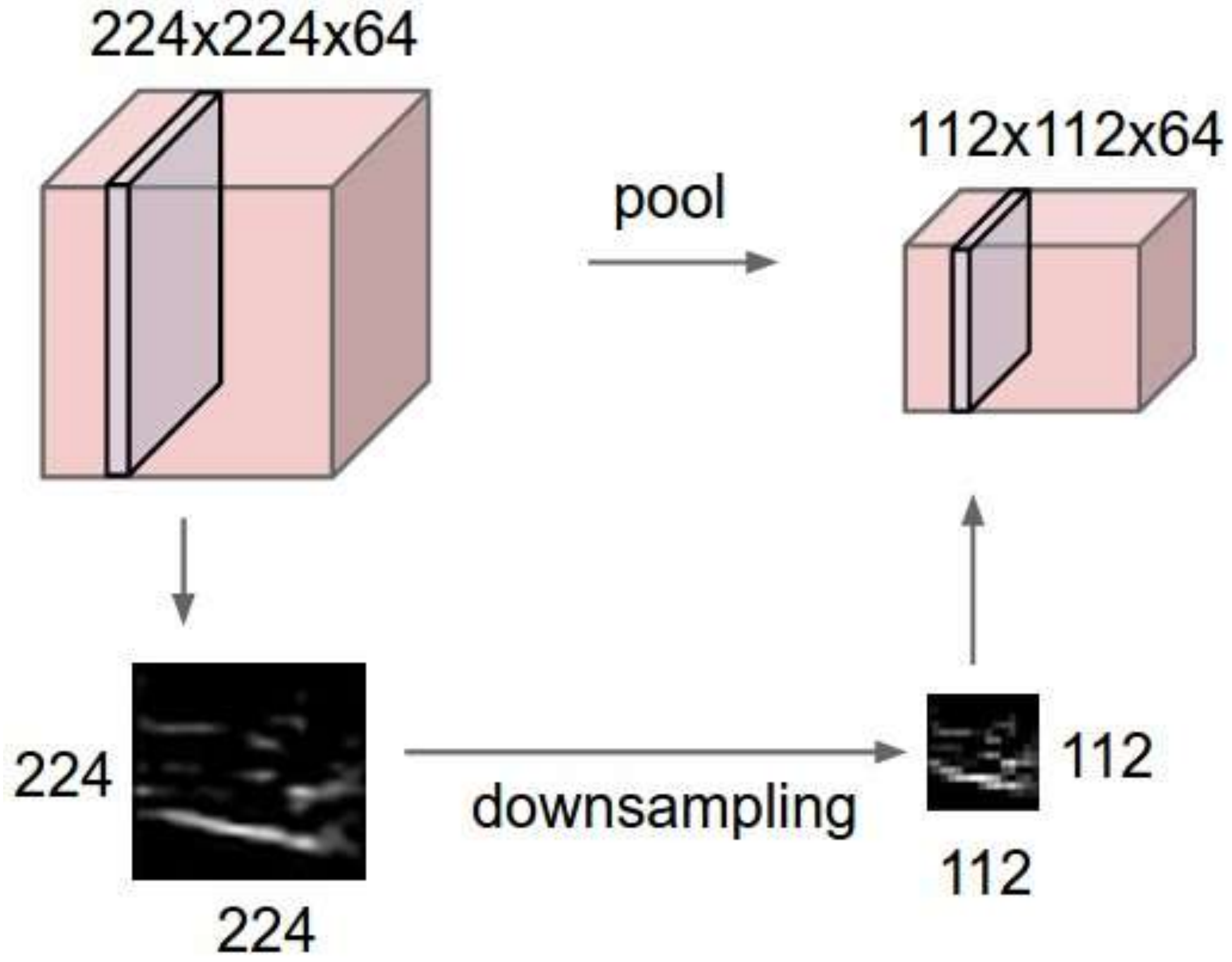
**Output : "outDim" images with size H,W ( if stride == 1)**

http://cs231n.github.io/convolutional-networks/

# *pooling Layer*

# 6.4

# Pooling



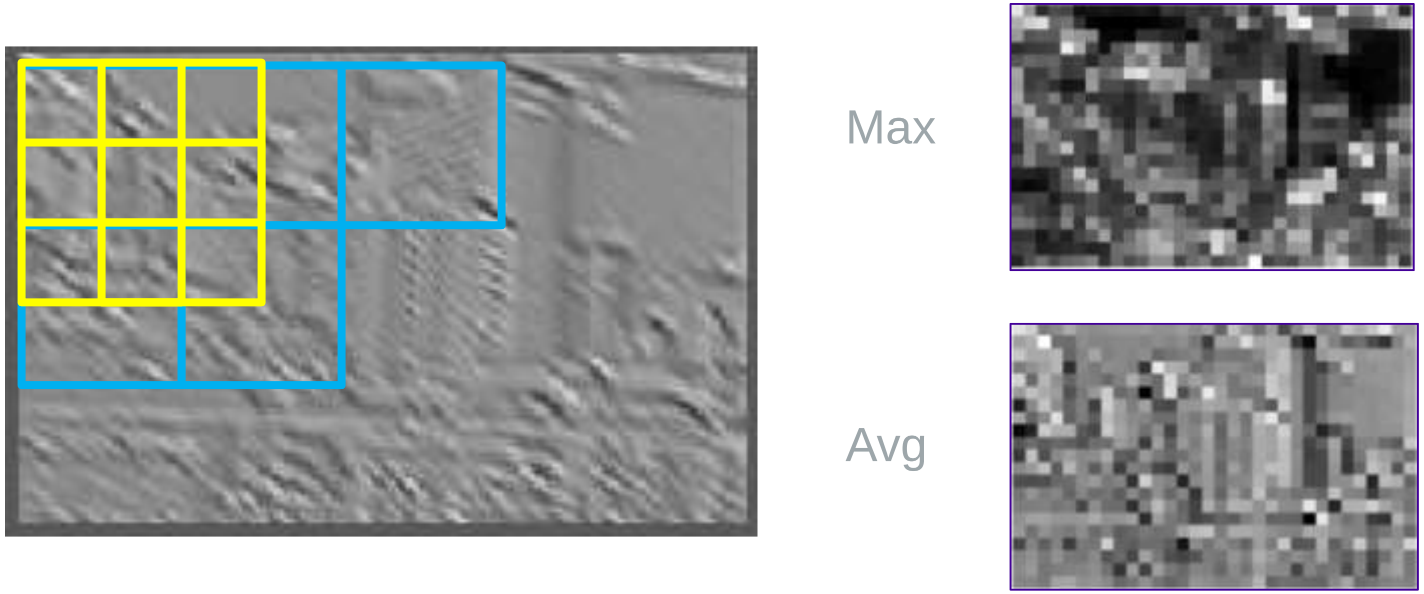

http://cs231n.github.io/convolutional-networks/

# Pooling

## Spatial Pooling

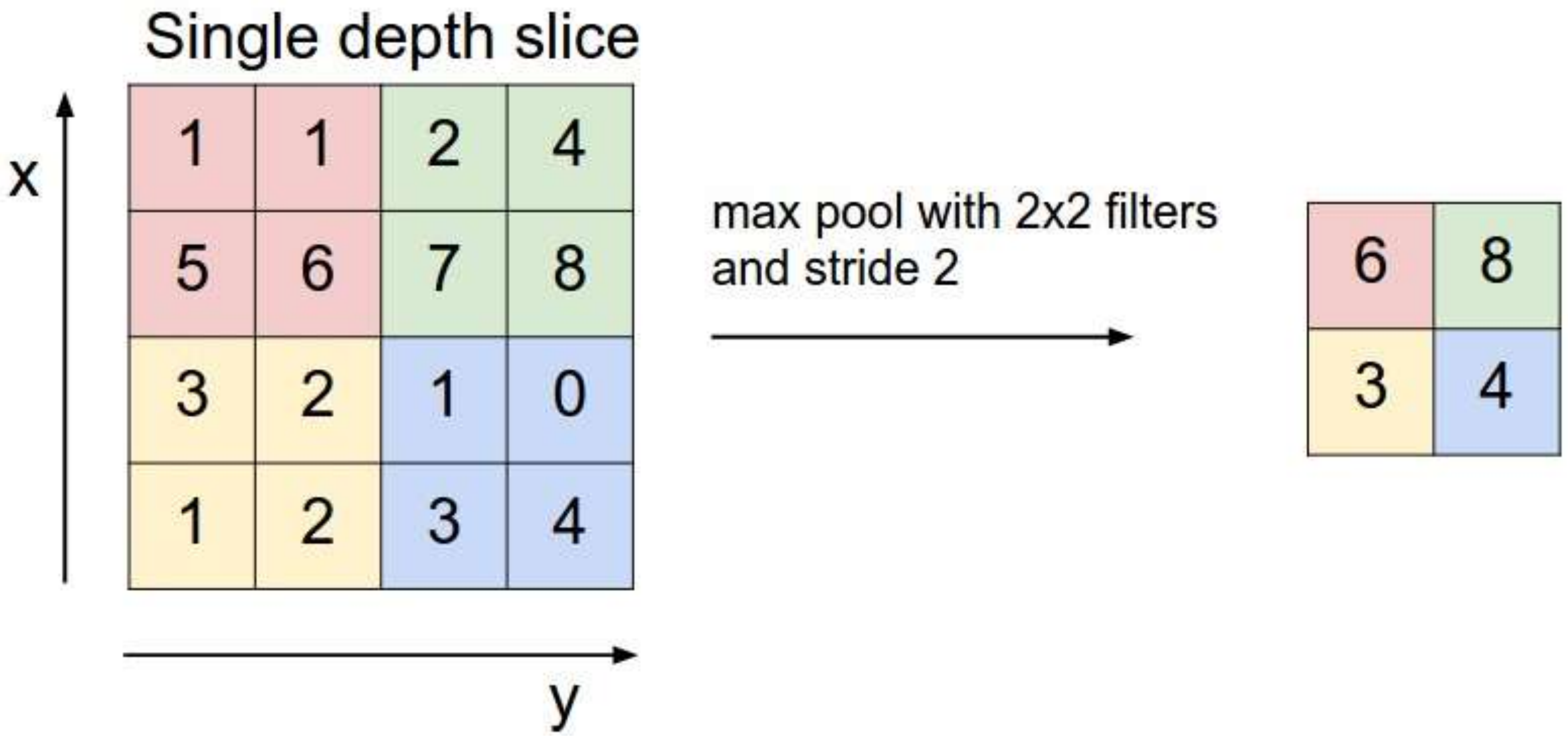
Non-overlapping / overlapping regions

Average or Max




reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

# max pooling




http://cs231n.github.io/convolutional-networks/

# *Loss Layer*

# 6.5

# Minimum Mean Squared Error

**Works generally well for learning tasks**

$$\mathcal{L} = \sum_n \left(y^{(n)} - t^{(n)}\right)^2$$

$y^{(n)}$ **- output of the classifier**
$t^{(n)}$ **- labels**

$$y^{(n)} = F\left(\boldsymbol{x}^{(n)}\right)$$
$$F \in \mathcal{F}$$

$\mathcal{F}$ **- the set of all functions representable by the neural network architecture.**

$$\min_{F \in \mathcal{F}} \sum_n \left(F\left(\boldsymbol{x}^{(n)}\right) - t^{(n)}\right)^2$$

# Softmax function

**Definition**

$$\boldsymbol{y}^{(n)}(i) = \frac{\exp\left(-\boldsymbol{a}^{(n)}(i)\right)}{\sum_j \exp\left(-\boldsymbol{a}^{(n)}(j)\right)}$$

**It can be easily verify that**

$$\sum_i \boldsymbol{y}^{(n)}(i) = 1$$

# Cross Entropy

**Definition:** $p_X(i), q_X(i)$

$$H(p_X|q_X) = -\sum_i p_X(i) \log q_X(i)$$

The cross entropy loss is closely related to the Kullback-Leibler divergence between the empirical distribution and the predicted distribution.

**Cross Entropy as a Loss Function:**

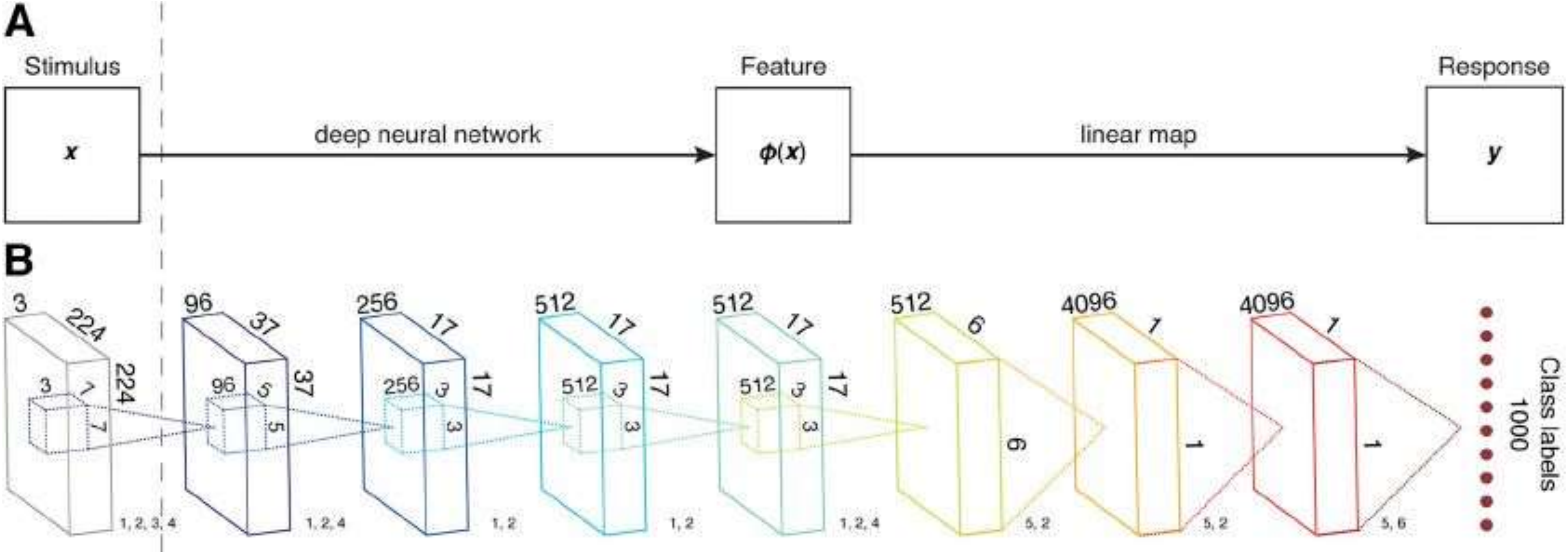
$$\ell^{(n)} = H\left(t^{(n)}|y^{(n)}\right) = -\sum_i t^{(n)}(i) \log y^{(n)}(i)$$

$$\mathcal{L} = \sum_n \ell^{(n)} = -\sum_n \sum_i t^{(n)}(i) \log y^{(n)}(i)$$

# Another CNN visualization

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

# Example of deep convolutional neural network for image classification




**ImageNet : 12 M images**

**Networks : up to 60 M parameters**

## CNN Building Blocks

- **Fully Connected layer**

  tensor = tf.matmul(tensor, W) + B

- **Activation layer**

  tensor = tf.nn.relu(tensor)

- **Convolutional Layers**

  tensor = tf.nn.conv2d(tensor, W, strides, padding) + B

- **Pooling Layers**

  tensor = tf.nn.max_pool(tensor, ksize, strides,padding)

- **Loss Layer**

  tensor = tf.nn.log_softmax(tensor)

# nn4 : a CNN

## *Definition of the network :*

```
class ConvNeuralNet(tf.Module):
   def __init__(self):
      self.unflat = Layers.unflat('unflat',48, 48, 1)
      self.cv1 = Layers.conv('conv_1', output_dim=3, filterSize=3, stride=1)
      self.mp = Layers.maxpool('pool', 2)
      self.cv2 = Layers.conv('conv_2', output_dim=6, filterSize=3, stride=1)
      self.cv3 = Layers.conv('conv_3', output_dim=12, filterSize=3, stride=1)
      self.flat = Layers.flat()
      self.fc = Layers.fc('fc', 2)

   def __call__(self, x, log_summary):
      x = self.unflat(x, log_summary)
      x = self.cv1(x, log_summary)
      x = self.mp(x)
      x = self.cv2(x, log_summary)
      x = self.mp(x)
      x = self.cv3(x, log_summary)
      x = self.mp(x)
      x = self.flat(x)
      x = self.fc(x, log_summary)
      return x
```

# nn4 : number of parameters and shapes …

### *Total number of parameters learned in nn4 ?*

***Conv 1 => (3*3*1+1)*3*** *=30*
***Conv 3 => (3*3*3+1)*6*** *=168*
***Conv 5 => (3*3*6+1)*12*** *=660*
***FC 7 => (6*6*12+1)*2*** *=866*
***Total = 1724***

### *Tensor shape as input for Conv_3 ?*

***In Dataset.py, default batchsize = 128***

***Shape = (128,24,24,3)***