CEFET/RJ

CBPF

Deep Learning
Applications in Astronomy

Clécio R. Bom

clearnightsrthebest.com

debom@cbpf.br

February 11, 2018

Program

Lecture 01

Introdução ao aprendizado de máquina
Algoritmo Backpropagation
Redes Multi-Layer Perceptron
Convolutional Neural Network.

Lecture 02

Training and convergence
Quality checks
AlexNet,VGG
Inception
Resnet

Lecture 03

Auto Encoders.
Generative Adversarial Networks (GAN).

Lecture 04

Region Based Convolutional Neural Networks (R-CNN).
Long-Short Term Memory (LSTM)
Reinforcement Learning

You will need:

-> Python (anaconda3)

-> Keras

-> Tensorflow

-> matplotlib

-> numpy

For the examples you will also need:

-> astropy

Google colabs (recommended) :

General Announcements

We will have several do-yourself examples in google collabs or in python notebooks. The examples and datasets required can be downloaded in

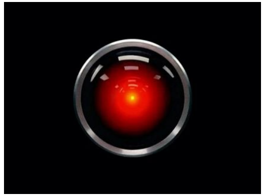
clearnightsrthebest.com

We have two tutors:

- Luciana Olivia Dias, MSC.
- Patrick Schubert.

They will be available from 13h-15h (1 p.m. - 3 p.m.). They can help with the examples.

Why go deep?



There is just too many data to analize (not enough woman/man power)

Automatize (lazyness? highter productivity)

Get intuitions, find patterns never seen before

If you like the idea of world ruled by robots

I want to make (tons of) money (outside academia) and live by the beach

How Deep (and Shallow) Learning works

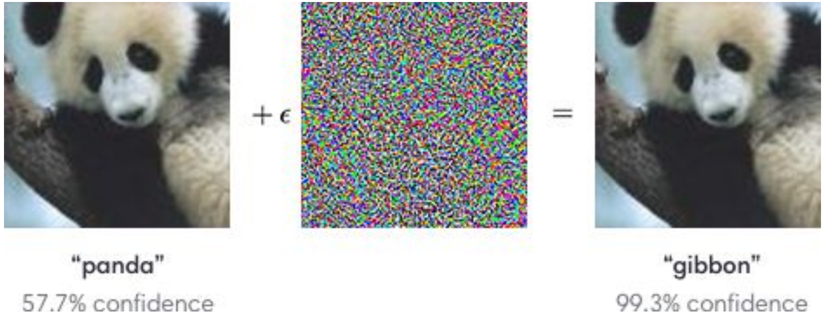
main Strategies : Supervised or Unsupervised

Common applications: Classification, Regression

Semantic segmentation, data simulations, image enhancement, beat humans in games

Caveats: data hungry, not self-explanatory , “unhuman” errors, very specialized.

Would they ever be like humans? Shall I tell my lazy uncle to start a campaign against robots?

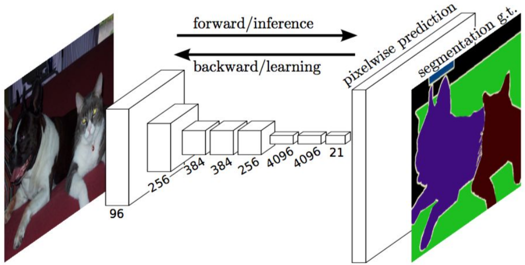


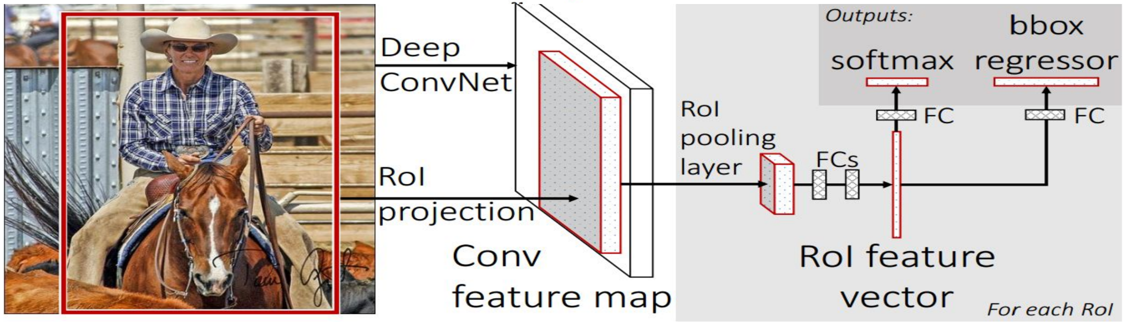
RCNNs and Semantic Segmentation

The Tradicional CNNs are build to run on same size images only.

This is an issue in many real life applications.

So, we need region proposals.



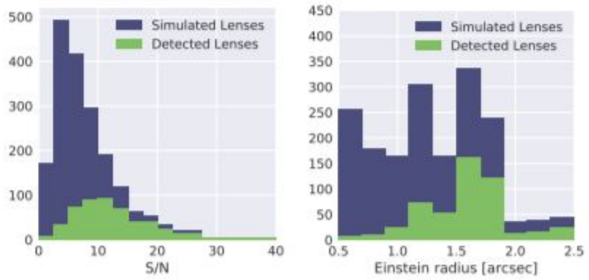


Adapted from: arXiv:1504.08083v2

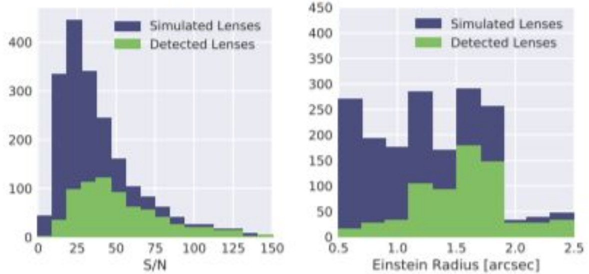
Girshick et al 2015.

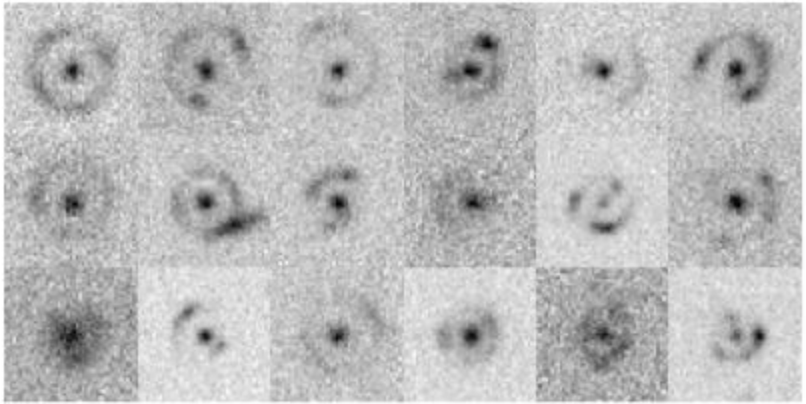
Classification : CNNs for Strong Lensing Detection

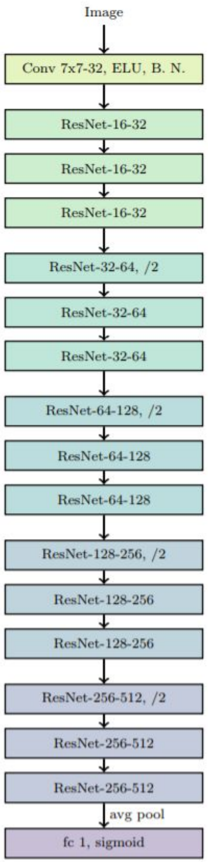
The authors trained and validate the model on a set of 20,000 LSST-like mock observations including a range of lensed systems of various sizes and signal-to-noise ratios (S/N).



(a) Single best epoch images



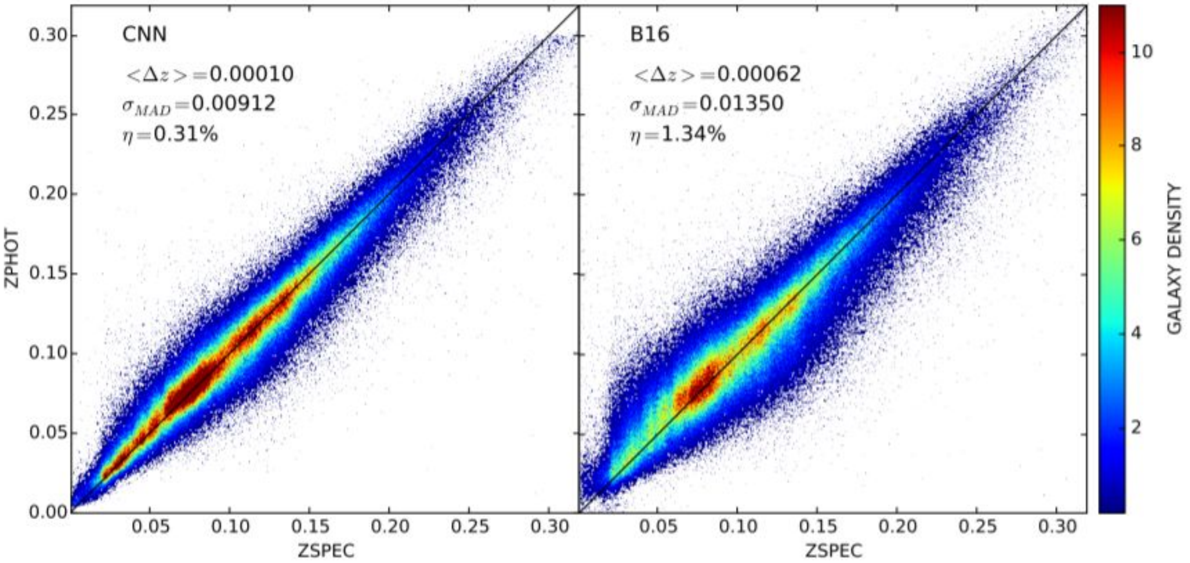




arXiv:1703.02642

Regression: CNNs for Photo-z in SDSS

Competitive photo-zs using cut-outs arXiv:1806.06607
CNN vs K-NN fitting



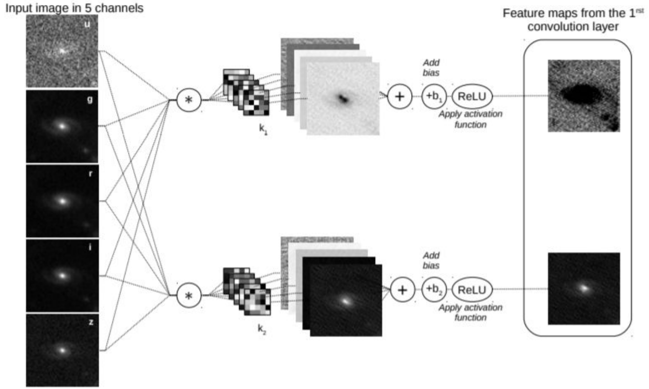
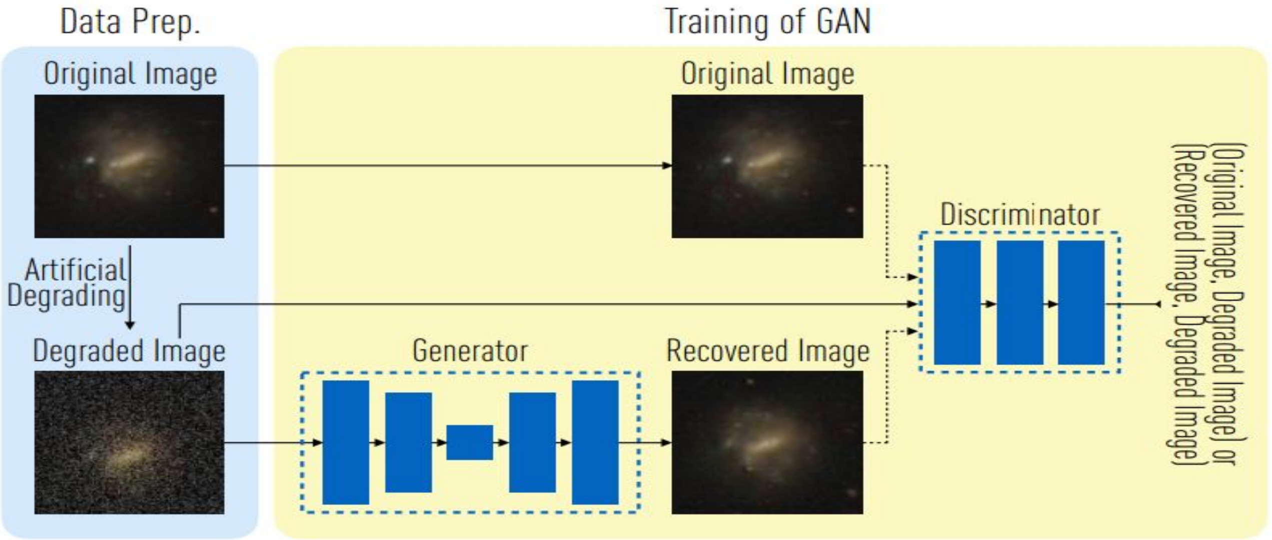


Image Enhancement : Generative Adversarial Neural Networks

First proposed by Goodfellow et al. 2014 arXiv:1406.2661

GAN has been exploited to restore images, simulate images.

Schawinski et al. 2017 used to deconvolve images beyond the deconvolution limit (arXiv:1702.00403v1).



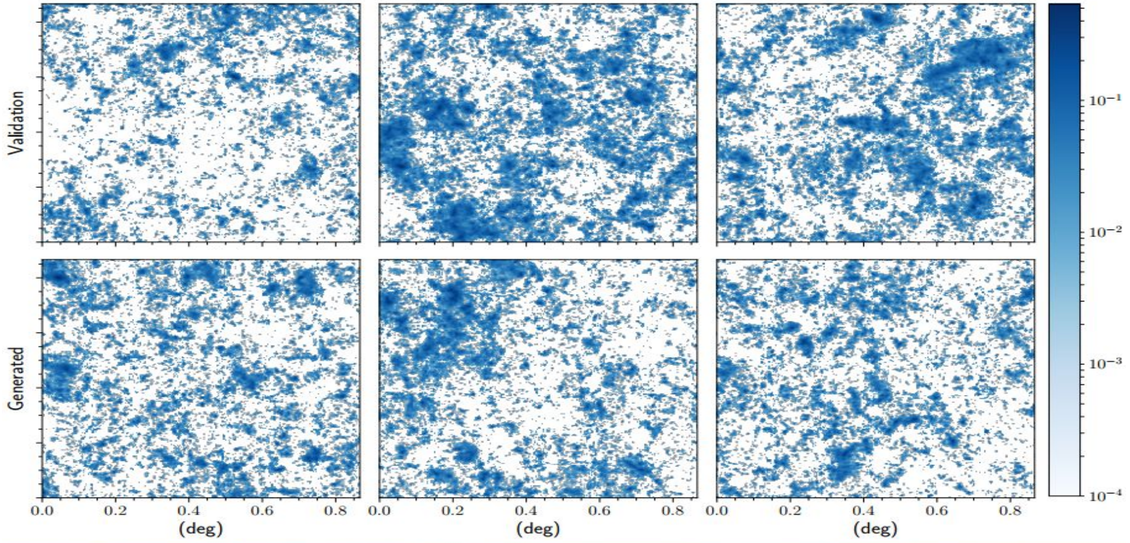
Data Simulations: Generative Adversarial Neural Networks

First proposed by Goodfellow et al. 2014 arXiv:1406.2661

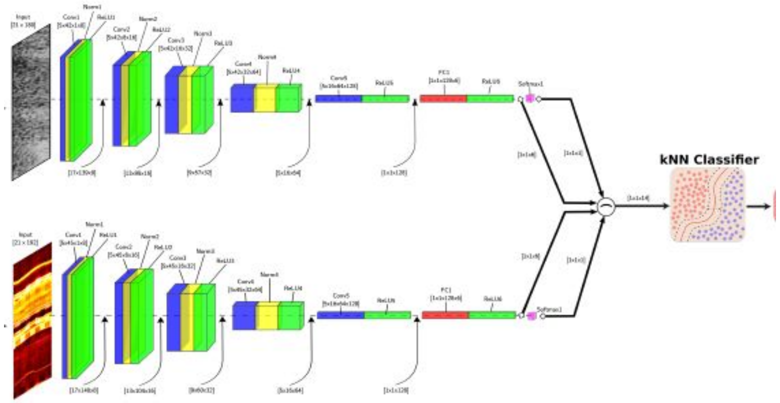
GAN has been exploited to restore images, simulate images.

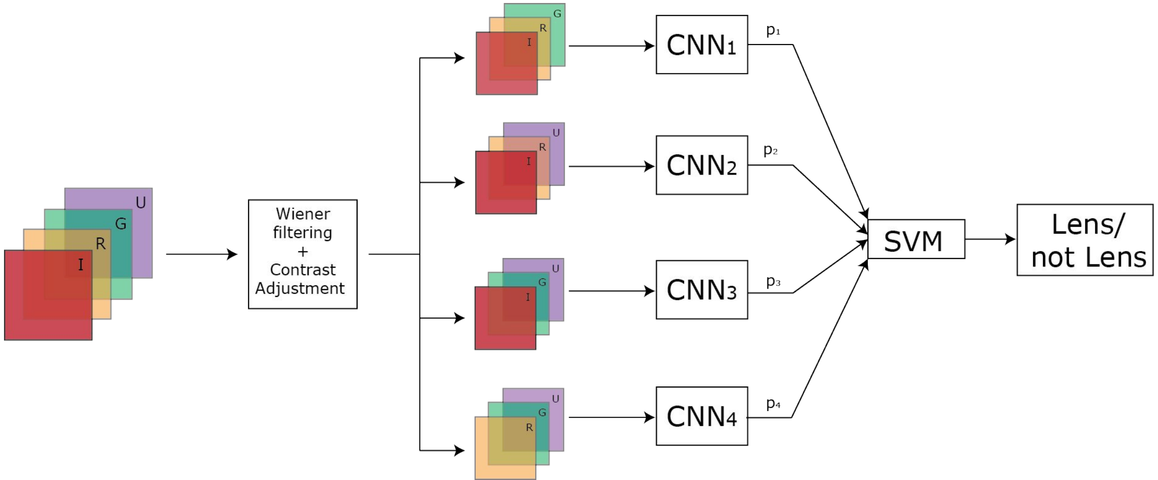
Mustafa et al. 2017 claims that can use GAN to simulate weak lensing convergence maps (arXiv:1706.02390v1).

$$\sigma_8 = 0.798, \; w = -1.0, \; \Omega_m = 0.26 \; \Omega_\Lambda = 0.74.$$



How sophisticated/hard is Deep Learning?





Do I need lots of Math?

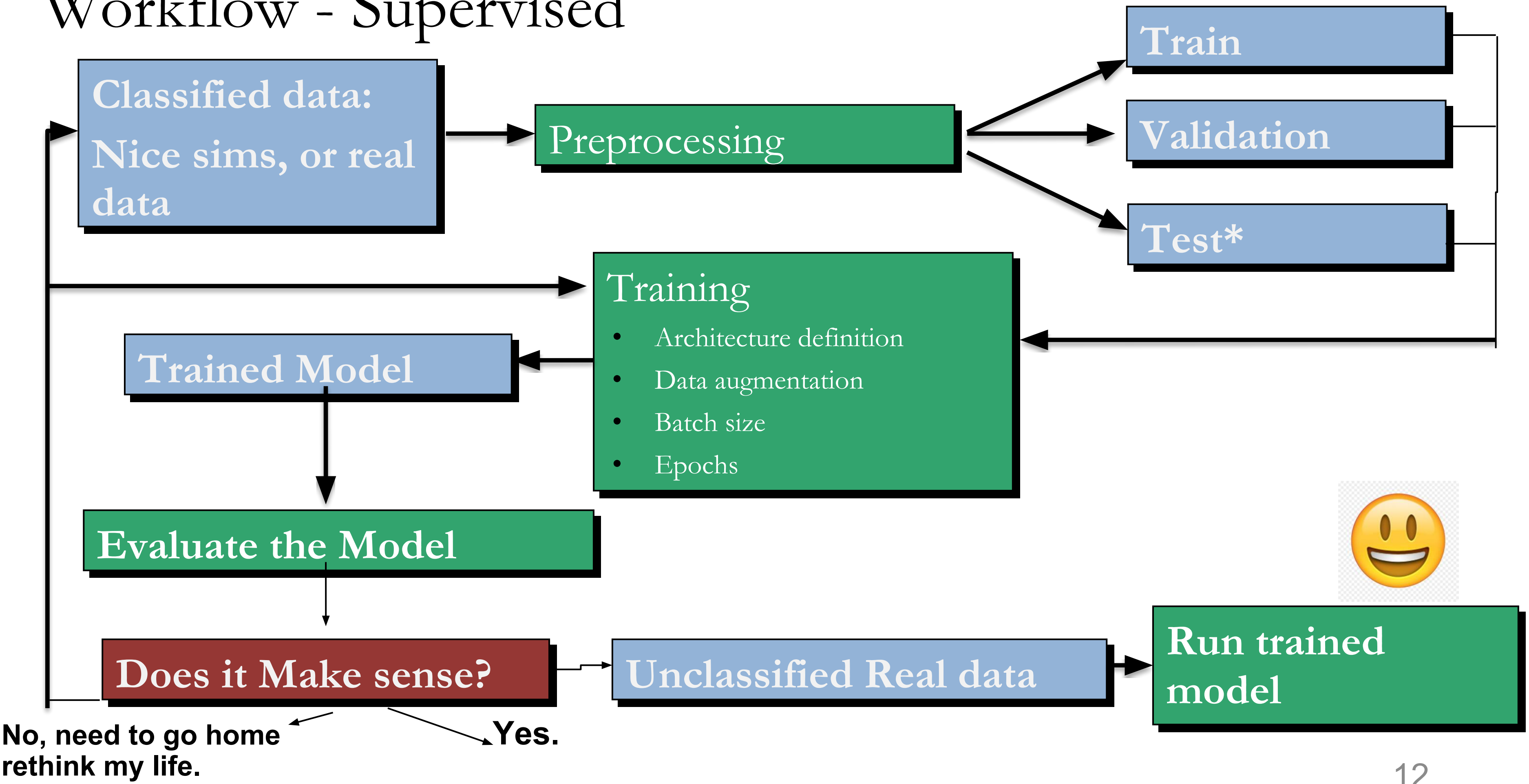
It is Intuitive?

It is a Lego?

It is alchemy?
It is a 'black box'?

Well, all depends on how far you want to go....

Workflow - Supervised

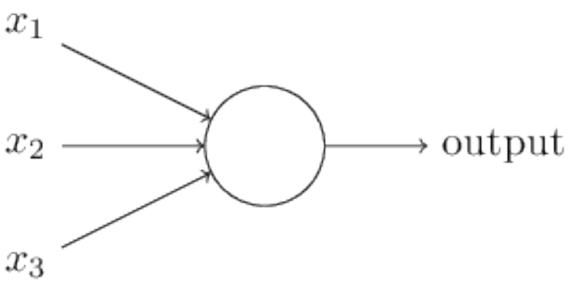


Starting simple...

The most simple start ....

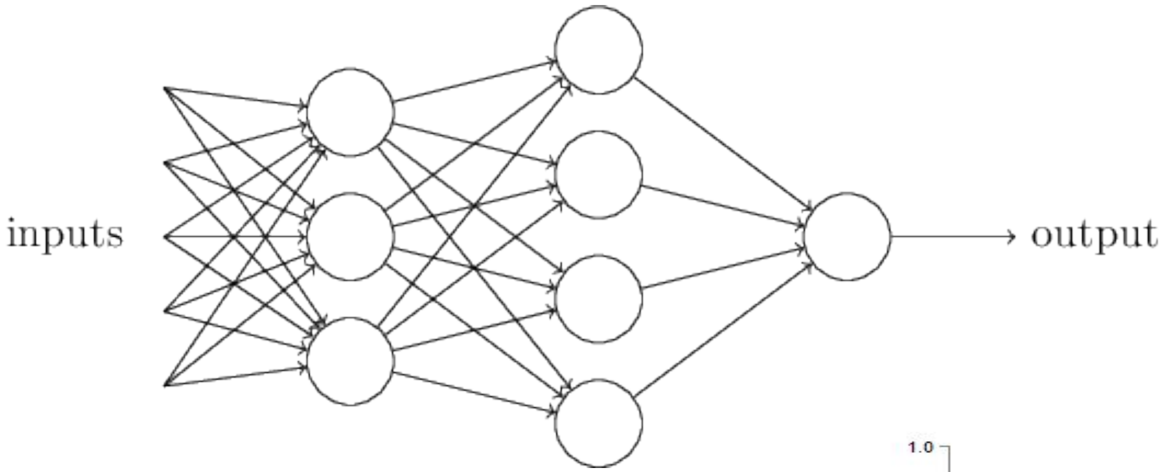
Developed by Frank Rosenblatt in the 1950s and 1960s

Binary output

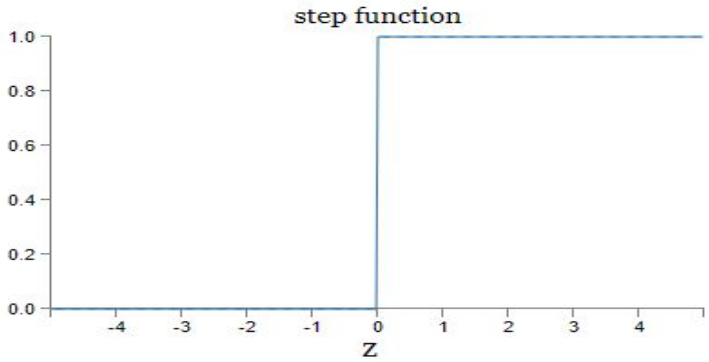


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \qquad (1)$$

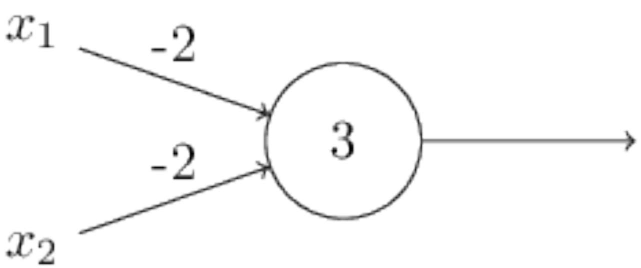
Starting simple...

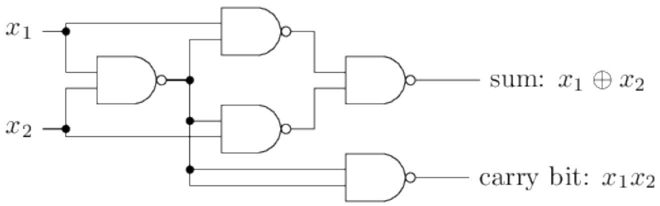


$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



Building a NAND Gate





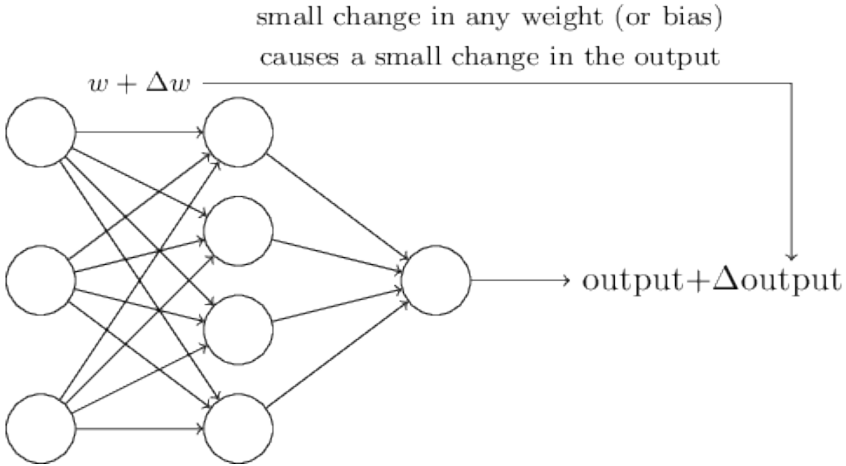
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

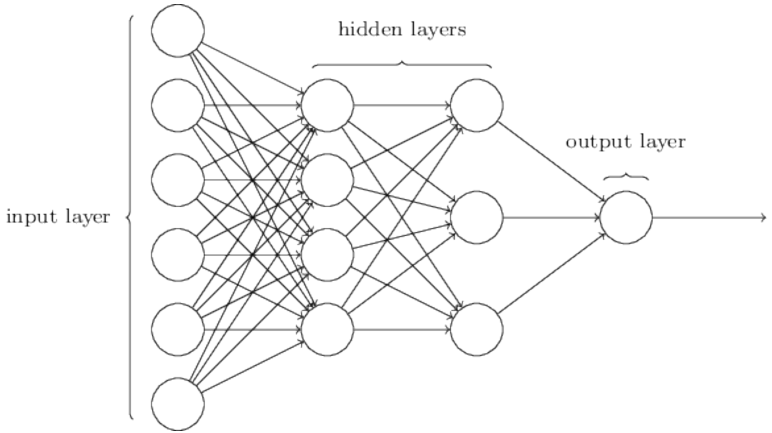
Input		Output
A	B	Y= $\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

Cool, but ...

Easily gets lots of parameters, particularly if everything is connected to everything.

Small variations in the neurons can have strong impact in the output.

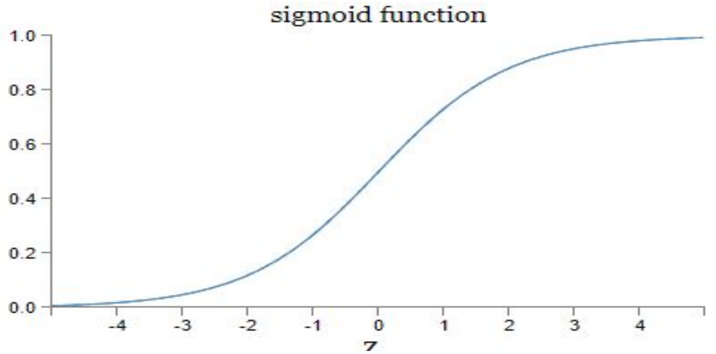


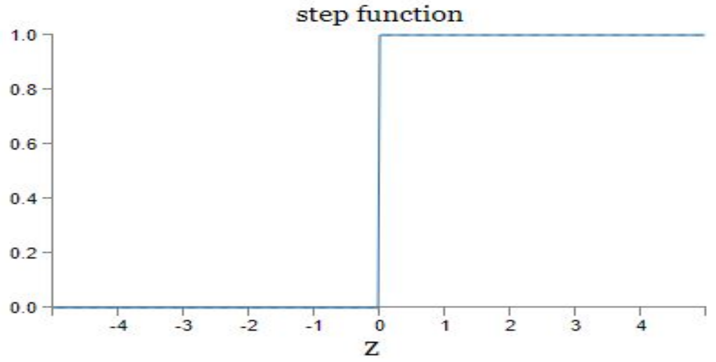


Sigmoid Activation

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$





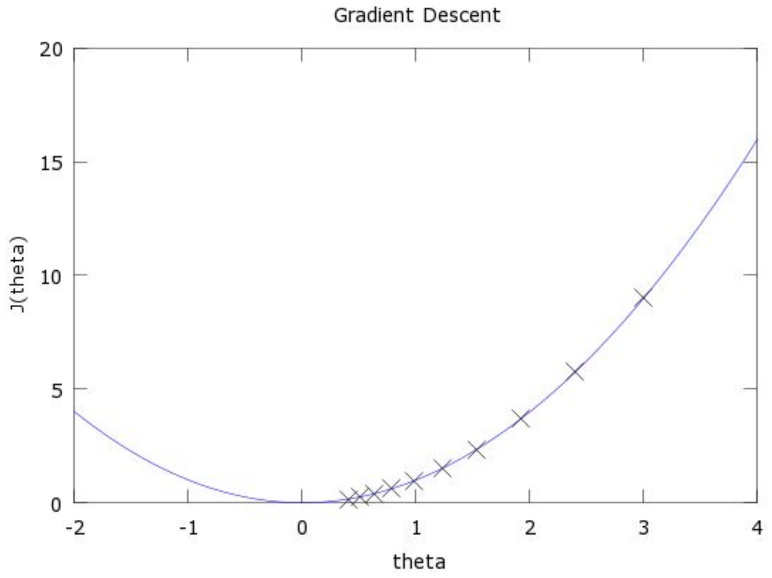
Gradient Descendent / How to optimize this!

Problem : find Θ that minimizes the cost

Let the Cost function $J(\Theta) = \Theta^2$
let's start with $\Theta = 3$
After each iteration we use the update rule:

$$\theta := \theta - \alpha \frac{d}{d\theta} J(\theta)$$

$$\alpha = 0.1, \qquad \frac{d}{d\theta} J(\theta) = 2\theta$$

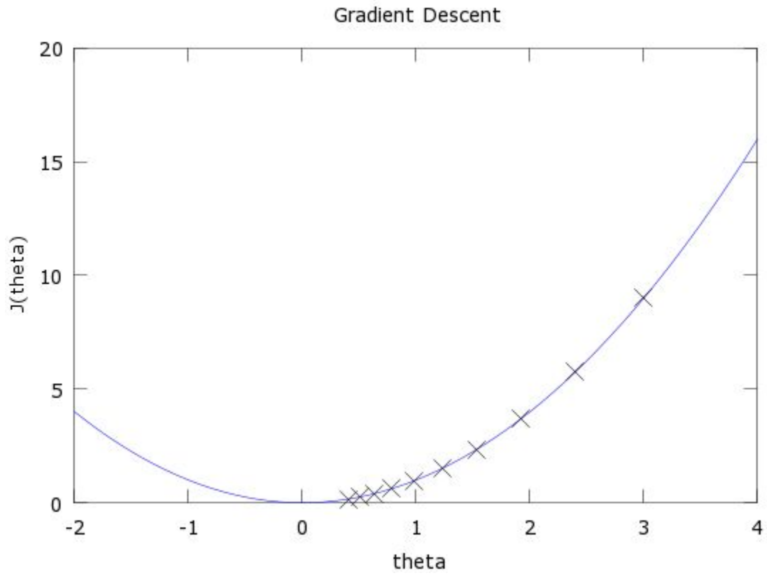


Gradient Descendent / How to optimize this!

Problem : find Θ that minimizes the cost

$$\theta := \theta - \alpha \frac{d}{d\theta} J(\theta) \qquad \alpha = 0.1, \qquad \frac{d}{d\theta} J(\theta) = 2\theta$$

Itera-tion	θ	$\alpha \frac{d}{d\theta} J(\theta)$
1	3	0.6
2	2.4	0.48
3	1.92	0.384
4	1.536	0.307
5	1.229	0.246
6	0.983	0.197
7	0.786	0.157
8	0.629	0.126
9	0.503	0.101
10	0.403	0.081



Adapted from: mccormickml.com/2014/03/04/gradient-descent-derivation/

Gradient Descendent / How to optimize this!

Linear model: $h(x) = \theta_0 + \theta_1 * x$.

Cost Function – "One Half Mean Squared Error":

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Objective:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Update rules:

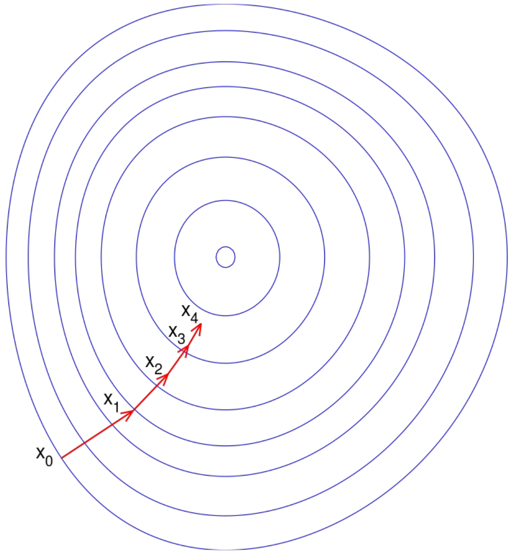
$$\theta_0 := \theta_0 - \alpha \frac{d}{d\theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\frac{d}{d\theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x^{(i)}$$

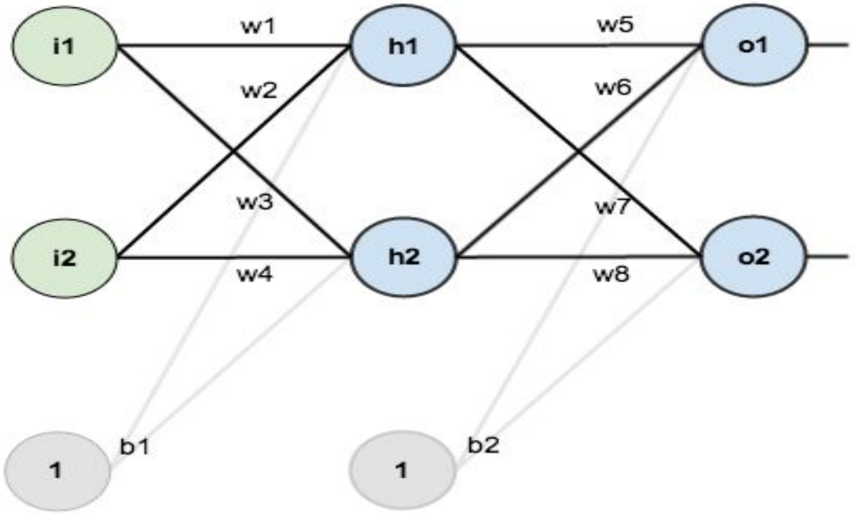


How Neural Nets can perform better than linear regression?

Neural networks can in principle model nonlinearities automatically, which you would need to explicitly model using transformations (for instance splines etc.) in linear regression.

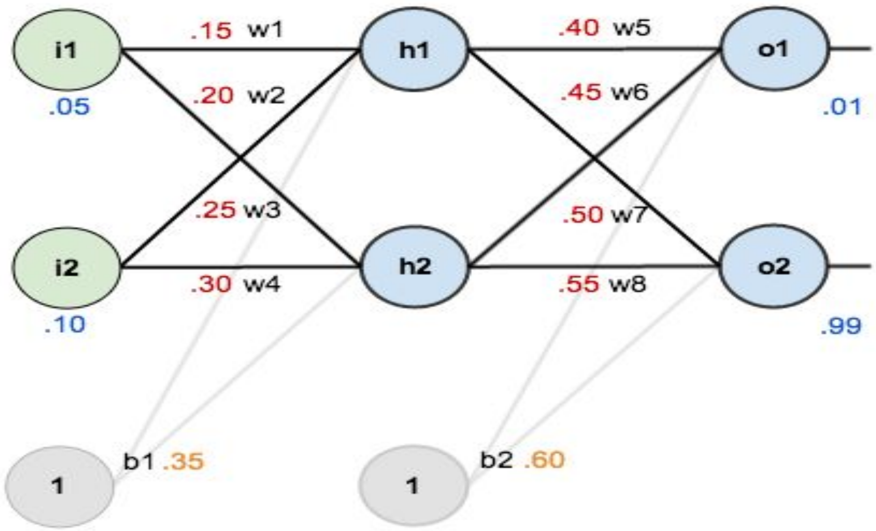
The universal approximation theorem states that a feed-forward dense network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R_n, with a few assumptions on the activation. Thus simple neural networks can represent a wide variety of functions when given appropriate parameters. However, **it does not say a word about learnability of those parameters**.

Let´s Backpropagate



Example adapted from:
mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Let´s Backpropagate



Moving forward ….

Here's how we calculate the total net input for h_1:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

We then squash it using the logistic function to get the output of h_1:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

Carrying out the same process for h_2 we get:

$$out_{h2} = 0.596884378$$

Moving forward ….

Here's the output for o_1:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

And carrying out the same process for o_2 we get:

$$out_{o2} = 0.772928465$$

Calculating the Total Error

We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

The total error

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Repeating this process for o_2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

Considering the chain rule ...

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

Considering the chain rule …

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

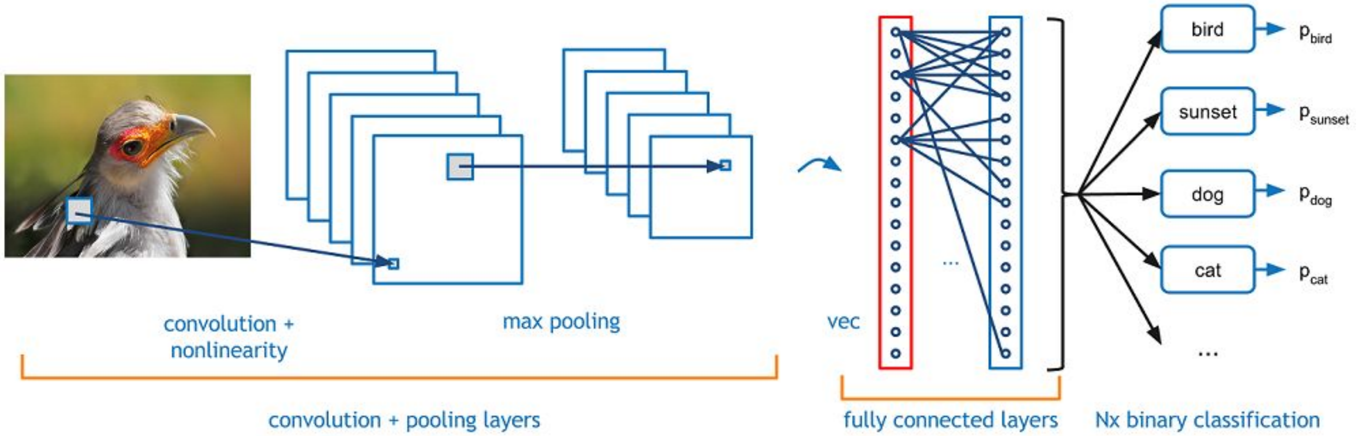
Moving backwards !

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Updating the weights

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

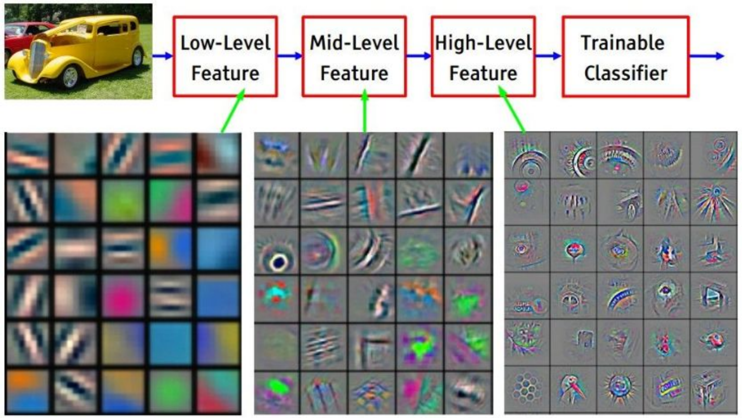
Let's play lego!



Convolutional Neural Networks

What makes CNNs so special?

- Based on mammal visual cortex
- Extract **surrounding-depending** high-order features.
- Specially useful for
    - Images
    - Time-dependent parameters (Speech recognition, Signal analysis)



Adapted from [11]

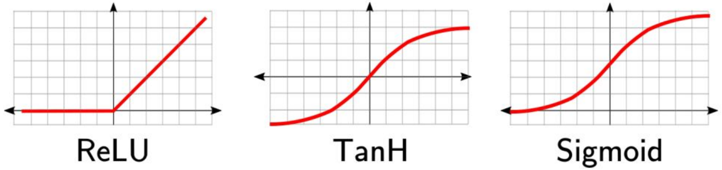
Types of Layers on CNNs

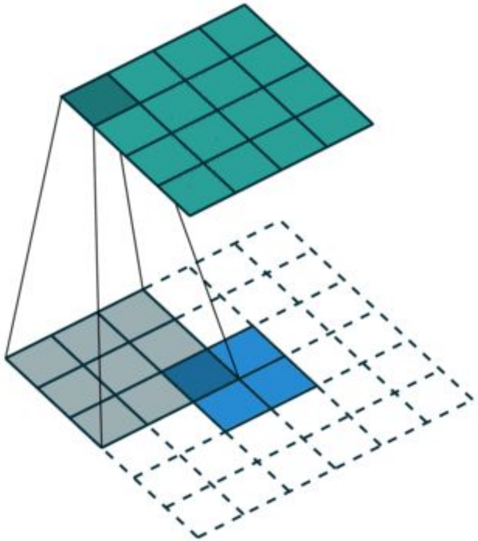
ConvLayers

- Based on Convolution
- Linear Operators
- Automatic (Visual) Feature Extractors.
- Change size of input data.

Activation Layers

- Add **non-linearity**
- Commonly follow each ConvLayer.
- Easy to implement & FAST to execute.



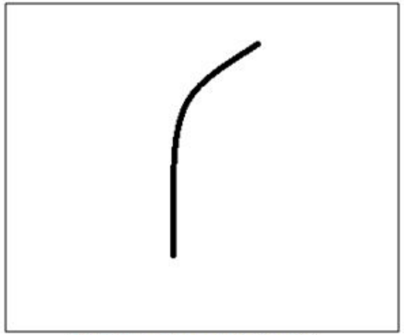


Adapted from [12]

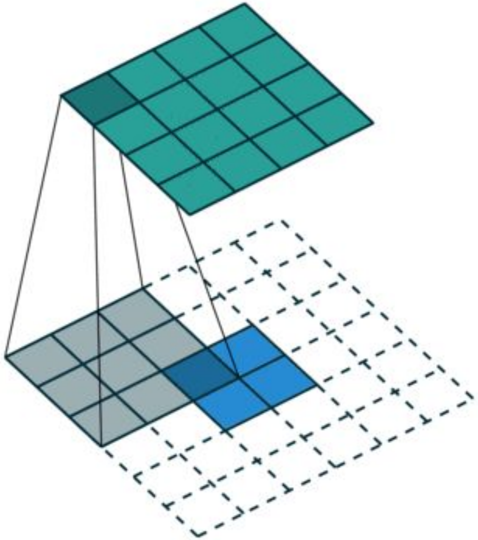
Conv Layers

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

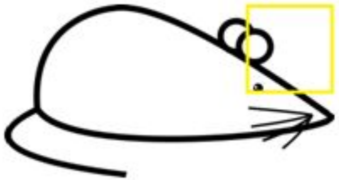
Pixel representation of filter



Visualization of a curve detector filter



Conv Layers



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

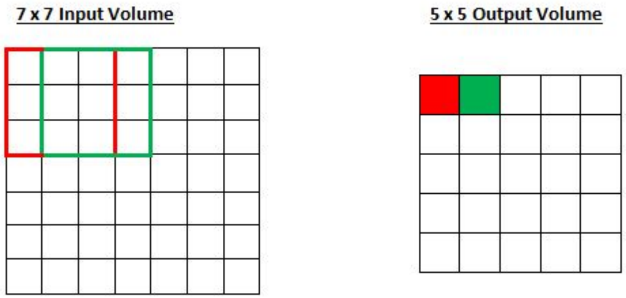
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

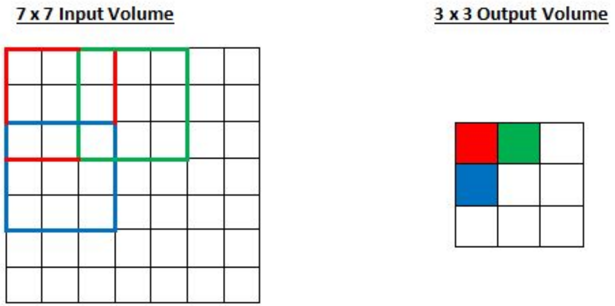
Pixel representation of filter

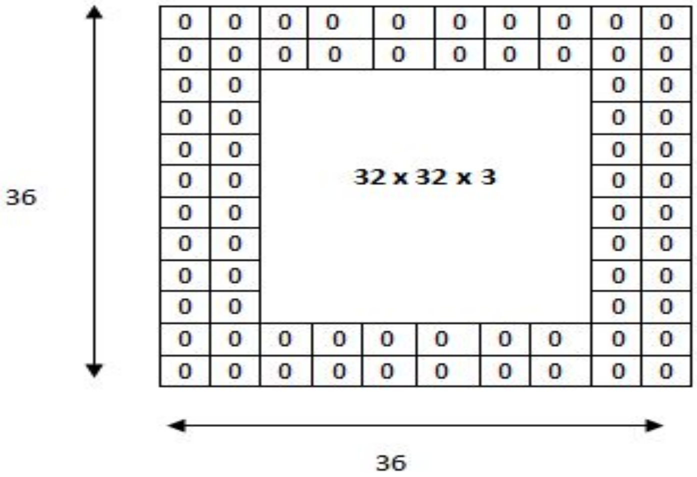
Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Going Back to Convolution….

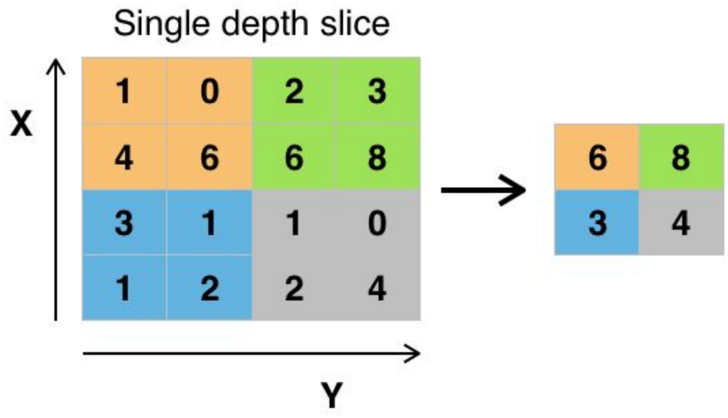
Stride and Padding



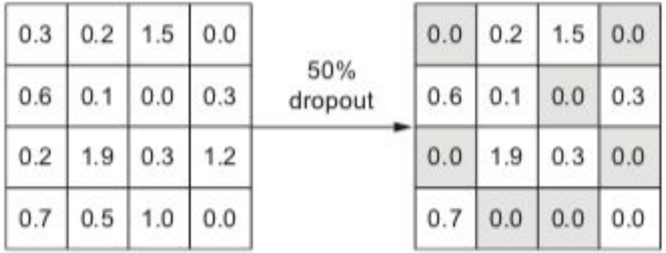




Dropout and Pooling



Example of Maxpool with a 2x2 filter and a stride of 2



Adapted from : https://ml-cheatsheet.readthedocs.io/en/latest/layers.html

The simplest example I know

```
from keras.datasets import mnist
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.models import Sequential

model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Adapted from :

The simplest example I know

```
batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_x, img_y = 28, 28

# load the MNIST data set, which already splits into train and test sets
for us
(x_train, y_train), (x_test, y_test) = mnist.load_data()

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test),
          callbacks=[history])
score = model.evaluate(x_test, y_test, verbose=0)
```

Classification Example: Strong Lensing Challenge

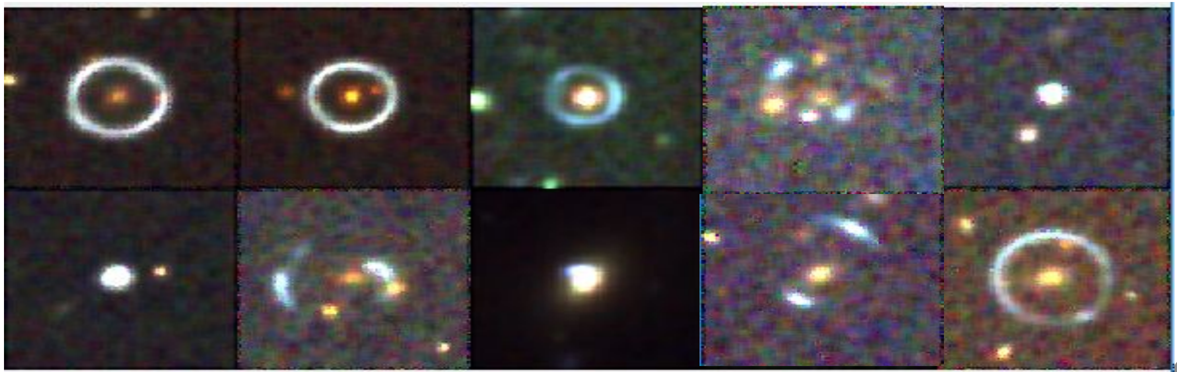
The Challenge:

Classify 200k images, between real data and simulations, in 48 hours for each of the two types (multiband or single band).

To test the algorithm we have 20k simulated images which contains all sorts of problems in the imaging system.

Each team developed different algorithms, mostly based in CNNs.

Metcalf et al.



Sorted by TPR_0 - The True Positive Rate with 0 mistakes

##	Team_name_submit	type	AUROC	TPR0	TPR10	description_short	author.1
## 7	resnet_5d0aad0	Space-Based	0.9225303	2.206807e-01	0.2904204271	CNN	Francois Lanusse
## 14	GAMOCLASS	Space-Based	0.9210117	7.416406e-02	0.3570444584	DL / CNN	Marc Huertas-Company
## 20	CAST-SB	Space-Based	0.8128851	6.909326e-02	0.1186942145	CNN	Clecio Roque De Bom
## 25	All-now	Space-Based	0.7346352	4.900040e-02	0.0659031545	edges/gradiants and Logistic Reg.	Camille Avestruz
## 16	Philippa Hartley	Space-Based	0.8012731	2.934848e-02	0.0717323859	SVM / Gabor	Philippa Hartley
## 17	Philippa Hartley2	Space-Based	0.8092423	2.859788e-02	0.0812650120	SVM / Gabor	Philippa Hartley
## 4	Manchester1	Space-Based	0.8101726	7.354597e-03	0.1739837398	Human Inspection	Neal Jackson
## 15	LASTRO EPFL (13b)	Space-Based	0.9325338	4.773626e-03	0.0779692201	CNN	Mario Geiger
## 2	GAHEC IRAP 1	Space-Based	0.6580909	1.127113e-03	0.0090920476	arc finder	R Cabanac
## 1	space	Space-Based	0.9143197	6.755404e-04	0.0127852282	CNN	Emmanuel Bertin
## 31	CNN_kapteyn	Space-Based	0.8179482	1.000625e-04	0.0002001251	CNN	Enrico Petrillo
## 18	res_bottleneck_87b7e8a	Space-Based	0.9068996	7.506005e-05	0.0038030424	CNN	Eric Ma
## 5	CMU-DeepLens-Resnet-Voting	Space-Based	0.9145407	0.000000e+00	0.0082046692	CNN	Quanbin Ma
## 11	Attempt2	Space-Based	0.7626792	0.000000e+00	0.0008265498	CNN / wavelets	Andrew Davies
## 10	YattaLensLite	Space-Based	0.7622929	0.000000e+00	0.0003502802	Arcs / SExtractor	Alessandro Sonnenfeld

##	Team_name_submit	type	AUROC	TPR0	TPR10	description_short	author.1
## 8	Philippa Hartley2	Ground-Based	0.9310191	2.237273e-01	0.3453159911	SVM / Gabor	Philippa Hartley
## 6	Philippa Hartley	Ground-Based	0.9293543	2.123763e-01	0.3316908714	SVM / Gabor	Philippa Hartley
## 13	resnet_ground_7bf8089	Ground-Based	0.9814321	8.993713e-02	0.4534297041	CNN	Francois Lanusse
## 19	LASTRO EPFL (11i)	Ground-Based	0.9749255	7.493794e-02	0.1131977256	CNN	Mario Geiger
## 9	CMU-DeepLens-Resnet-Voting	Ground-Based	0.9804913	2.445130e-02	0.1027314963	CNN	Quanbin Ma
## 3	All-star	Ground-Based	0.8365358	7.181615e-03	0.0186123524	edges/gradiants and Logistic Reg.	Camille Avestruz
## 26	Manchester-NA2	Ground-Based	0.8913778	2.803645e-04	0.0075297887	Human Inspection	Neal Jackson
## 27	Manchester-NA2	Ground-Based	0.8913778	2.803645e-04	0.0075297887	Human Inspection	Neal Jackson
## 28	Manchester-NA2-Submission2	Ground-Based	0.8913778	2.803645e-04	0.0075297887	Human Inspection	Neal Jackson
## 29	Manchester-NA2-Submission2	Ground-Based	0.8913778	2.803645e-04	0.0075297887	Human Inspection	Neal Jackson
## 30	YattaLensLite	Ground-Based	0.8191702	2.194382e-04	0.0021145867	SExtractor	Alessandro Sonnenfeld
## 12	CAST-GB	Ground-Based	0.8347916	2.005535e-05	0.0003810517	CNN / SVM	Clecio Roque De Bom
## 21	Ground	Ground-Based	0.9557059	0.000000e+00	0.0071018193	CNN	Emmanuel Bertin
## 22	Ground	Ground-Based	0.9557059	0.000000e+00	0.0071018193	CNN	Emmanuel Bertin
## 23	Ground_fixed	Ground-Based	0.9557059	0.000000e+00	0.0071018193		1
## 24	Ground_fixed	Ground-Based	0.9557059	0.000000e+00	0.0071018193		1

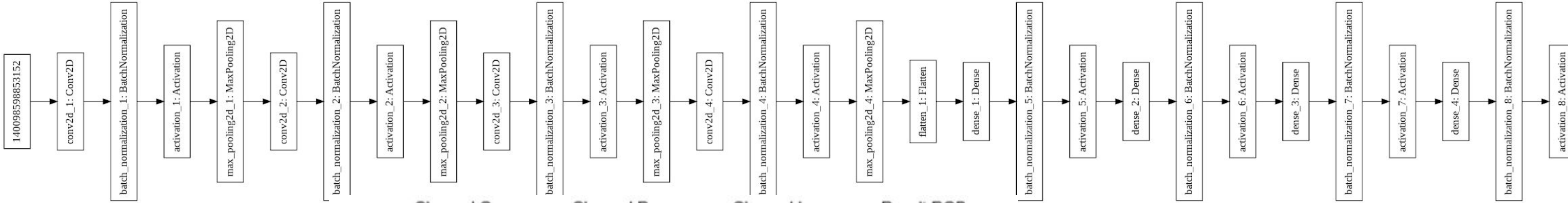
Metcalf et al.
2018
arXiv:1802.03609

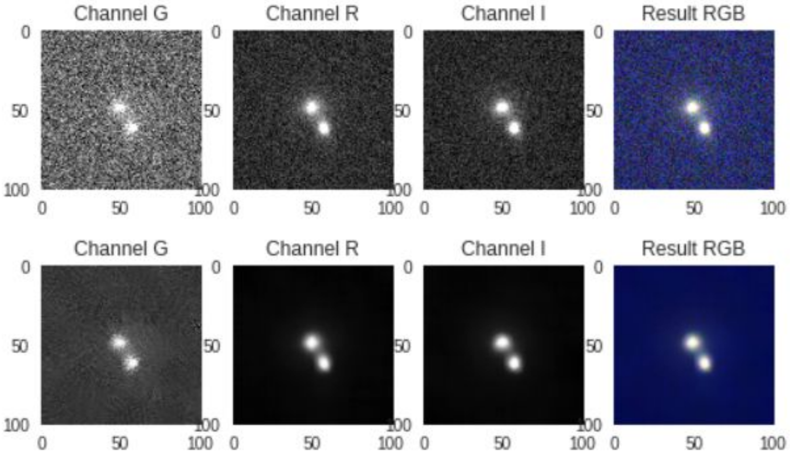
The Bronze
medal

Example 1 - Lens Detect

Project Colab LensDetectNet

Convolutional Neural Network to Detect Lens in Image .fits





Example 1 - Lens Detect

Project Colab LensDetectNet

Convolutional Neural Network to Detect Lens in Image .fits


```
Total params: 49,714,696
Trainable params: 49,712,008
Non-trainable params: 2,688
```

Deep Learning
Applications in Astronomy

Clécio R. Bom

clearnightsrthebest.com

debom@cbpf.br

February 11, 2018