UCSF
University of California
San Francisco

Introduction to Convolutional Neural Network

Valentina Pedoia, PhD
Musculoskeletal Quantitative Imaging Research Group
Dept. of Radiology and Biomedical Imaging UCSF

Agenda

- **General concepts on Neural Networks**
  - Definitions, Architecture types, Backpropagation, Activation Functions
- **Multilayer … Deep is better**
- **Handcraft Feature vs Feature Learning**
- **Convolutional Neural Networks**
  - Main Architectural Elements :
    - Convolutional
    - ReLu
    - Pooling
    - Fully Connected Layer

Neural Network

- A Neural Network is an information processing paradigm that **is inspired by the biological nervous systems**, such as the human brain's information processing mechanism.

- **The key element of this paradigm is the structure of the information processing system.** It is composed of a large number of **highly interconnected processing elements** (neurons) working in unison to solve specific problems.

- NNs, "like people", **learn by example**.

- A NN is configured for a specific application, such as pattern recognition or data classification, **through a learning process**.

- Learning in biological systems involves **adjustments to the synaptic connections** that exist between the neurons. **This is true of NNs as well.**

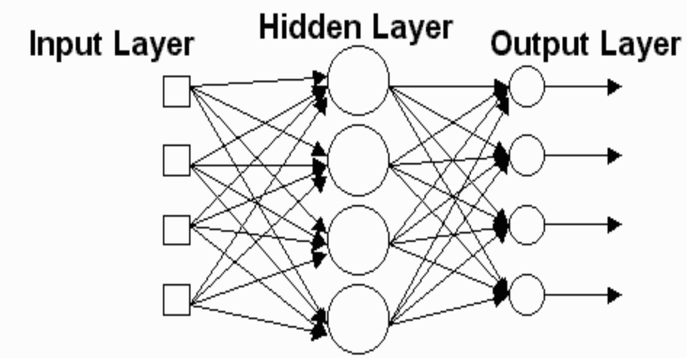


Figure 2 The anatomy of a neural network.

Types of Neural Network Architecture

Feed-forward networks

Feed-forward NNs allow **signals to travel one way only**; from input to **output.**
There is no feedback (loops) i.e. the output of any layer does not affect that same layer.

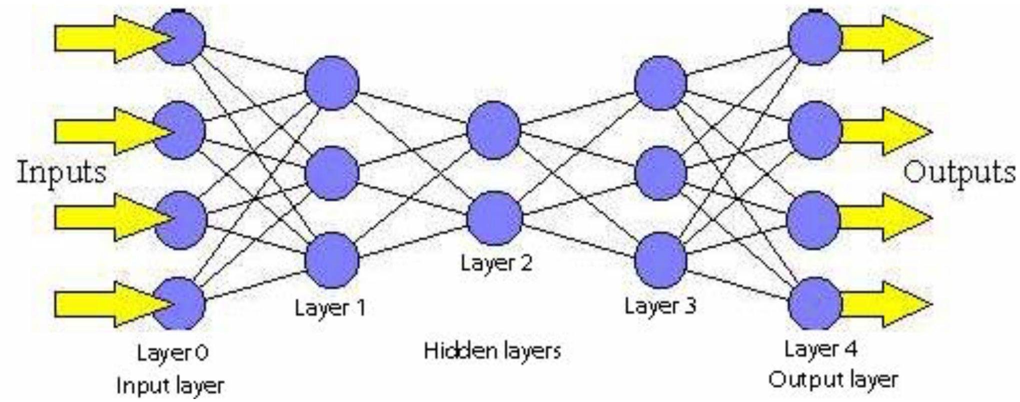
Feedback networks

Feedback networks can have signals traveling in both directions by introducing loops in the network.

Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point.
They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.
Feedback architectures are also referred to as **interactive or recurrent**

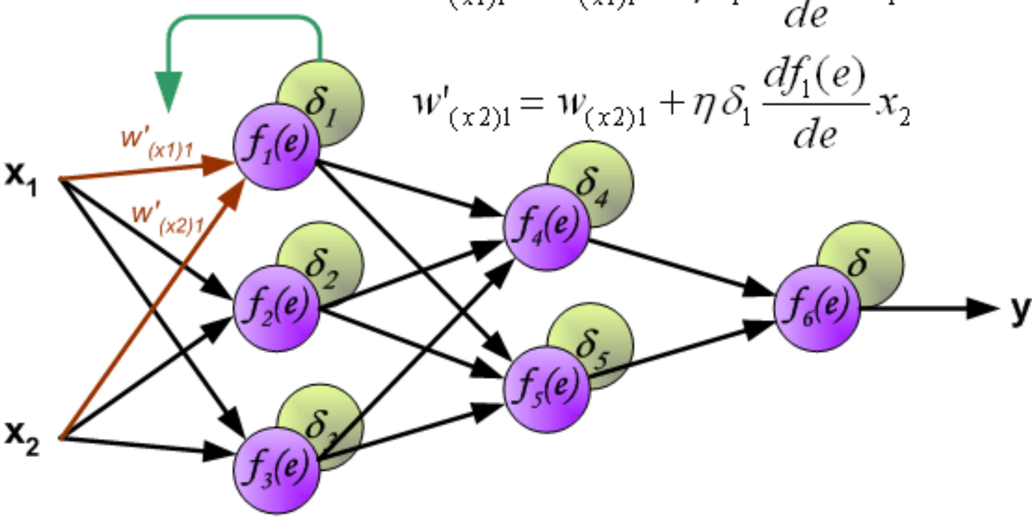
UCSF

Learning Algorithm



$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



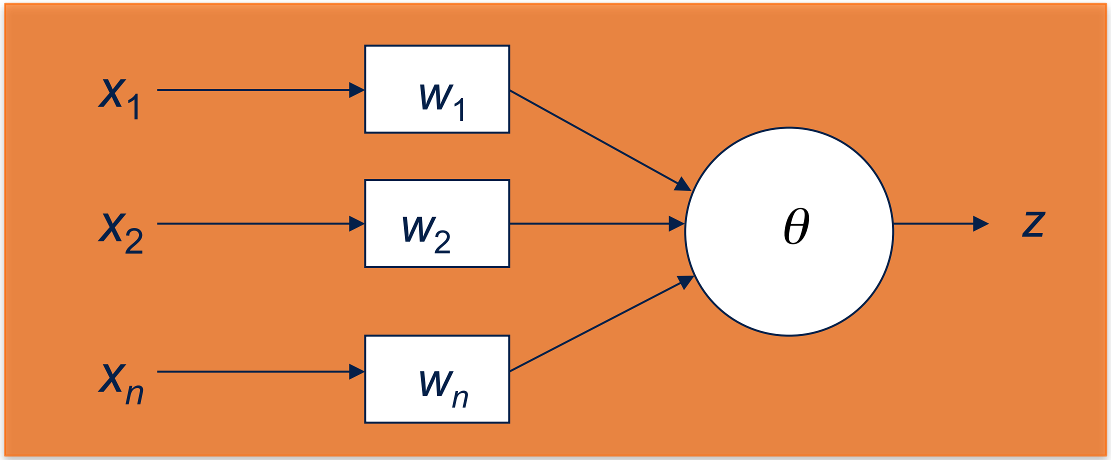
Backpropagation:

Randomly initialize the parameters

Calculate total error at the top

Then calculate contributions to error, at each step going backwards

The most simple neural network: Threshold Logic Unit



$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Training

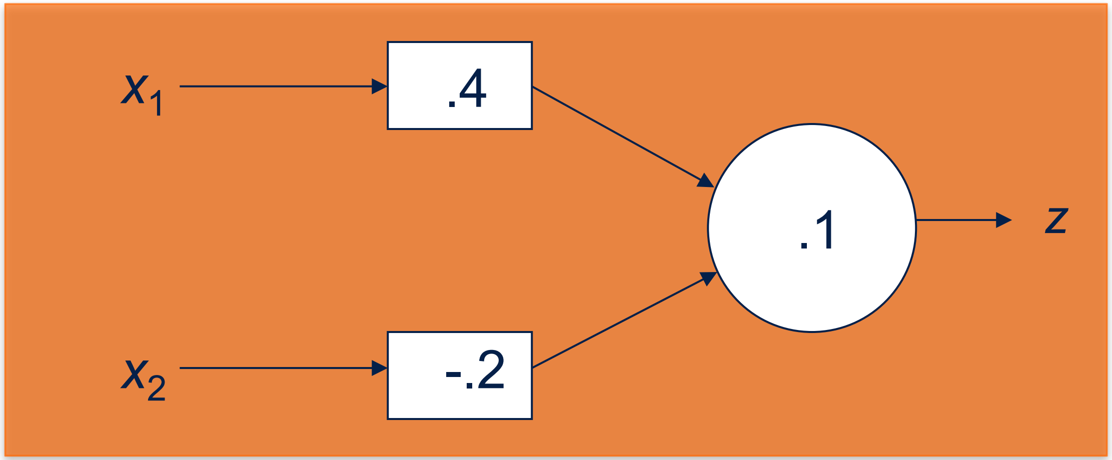
$$\Delta w_i = c(t - z)\, x_i$$

- Where w_i is the weight from input i to perceptron node,
- c is the learning rate (the user decide this parameter),
- t_j is the target for the current instance,
- z is the current output, and
- x_i is i^{th} input

- Least perturbation principle
  - Only change weights if there is an error
  - small *c* rather than changing weights sufficient to make current pattern correct
- Iteratively apply a pattern from the training set and apply the perceptron rule
- Each iteration through the training set is an *epoch*
- Continue training until total training set error ceases to improve
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

Let's try a simple example ….

1. Initialize the weights and the threshold

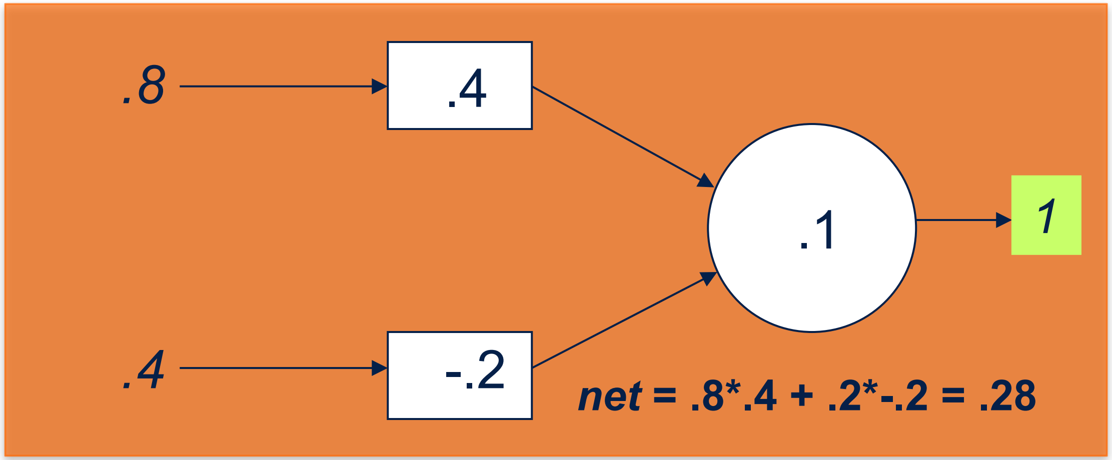


x_1	x_2	t
.8	.2	1
.4	.3	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Let's try a simple example ….

1. Initialize the weights and the threshold

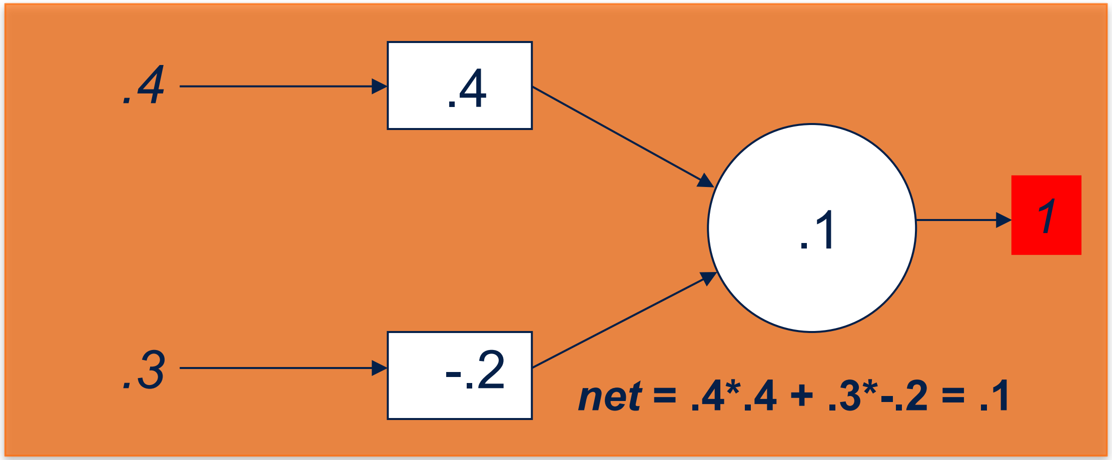


x_1	x_2	t
.8	.2	1
.4	.3	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Let's try a simple example ….

1. Initialize the weights and the threshold



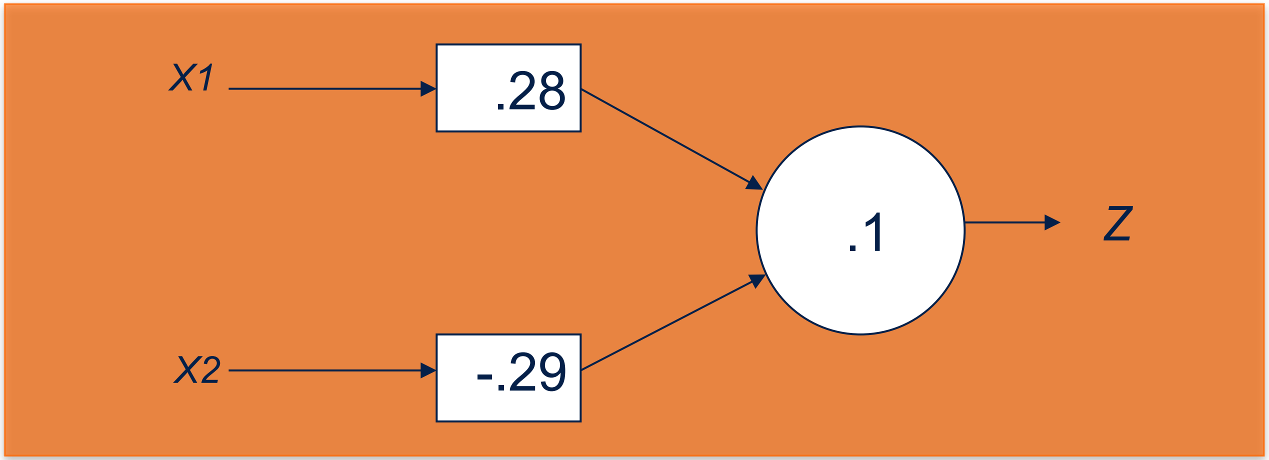
x_1	x_2	t
.8	.2	1
.4	.3	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Let's try a simple example ….

$$\Delta w_i = c(t - z)\ x_i$$

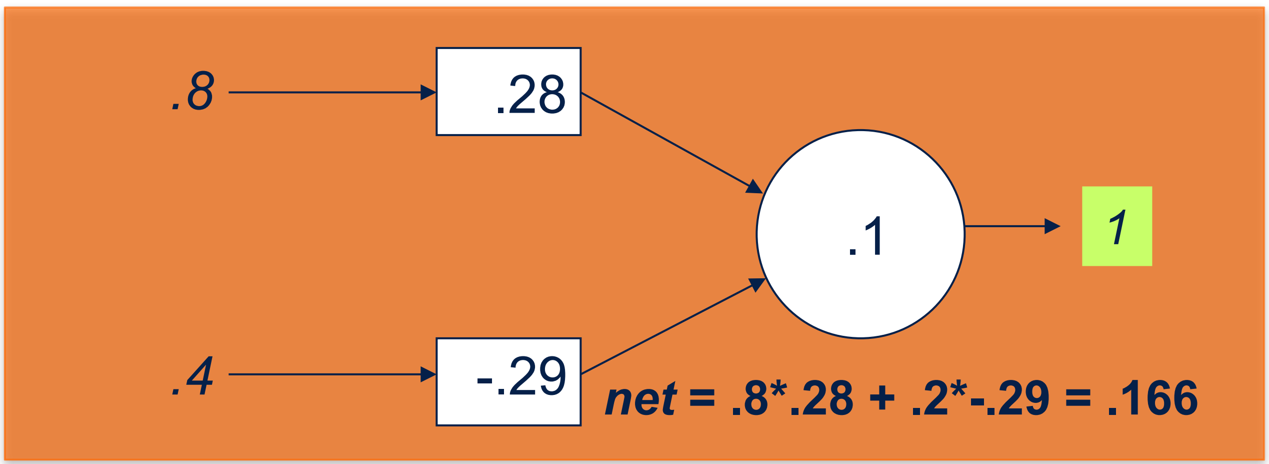
$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$



Let's try a simple example ….

$$\Delta w_i = c(t - z)\ x_i$$

$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$

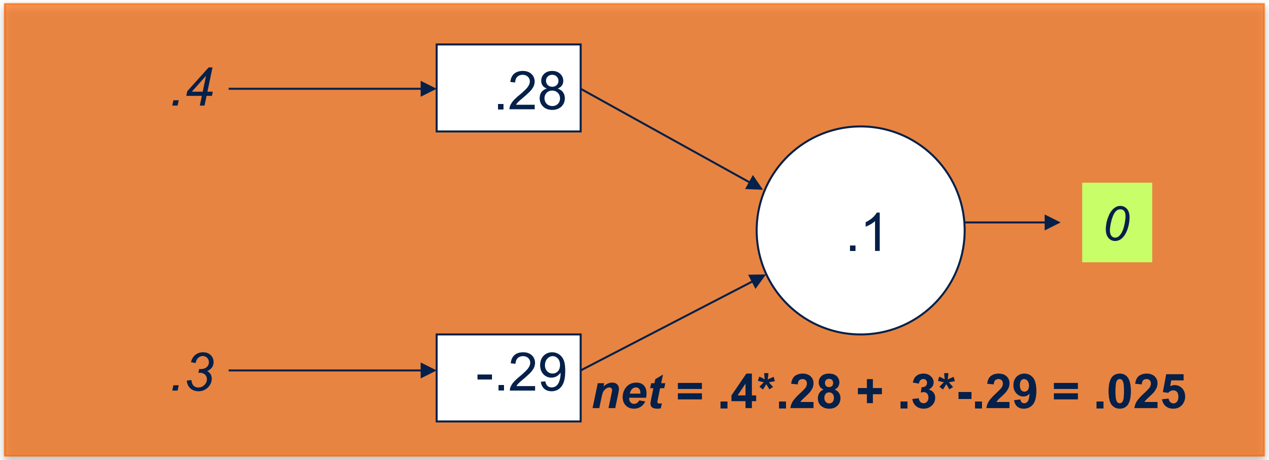


x_1	x_2	t
.8	.2	1
.4	.3	0

Let’s try a simple example ….

$$\Delta w_i = c(t - z)\, x_i$$

$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$

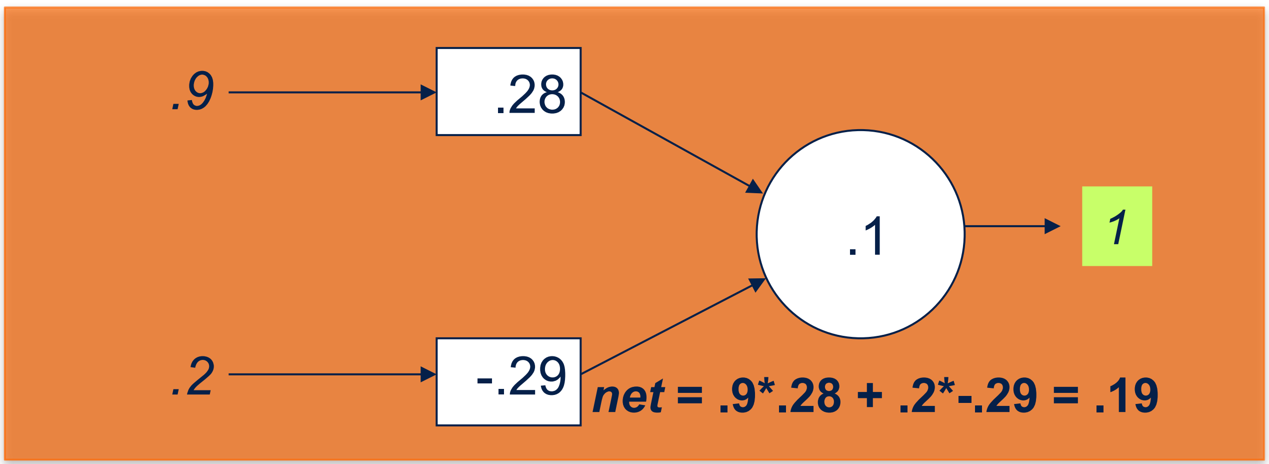


x_1	x_2	t
.8	.2	1
.4	.3	0

Let’s try a simple example ….

$$\Delta w_i = c(t - z)\, x_i$$

$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$



x_1	x_2	t
.8	.2	1
.4	.3	0

Training Set

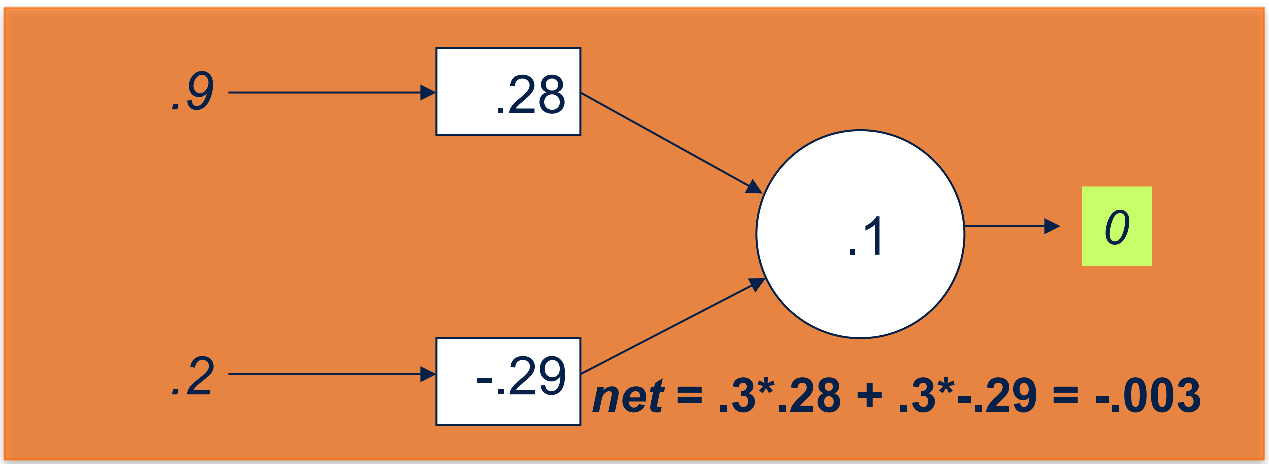
x_1	x_2	t
.9	.2	1
.3	.3	0

Test Set

Let's try a simple example ….

$$\Delta w_i = c(t - z)\, x_i$$

$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$



x_1	x_2	t
.8	.2	1
.4	.3	0

Training Set

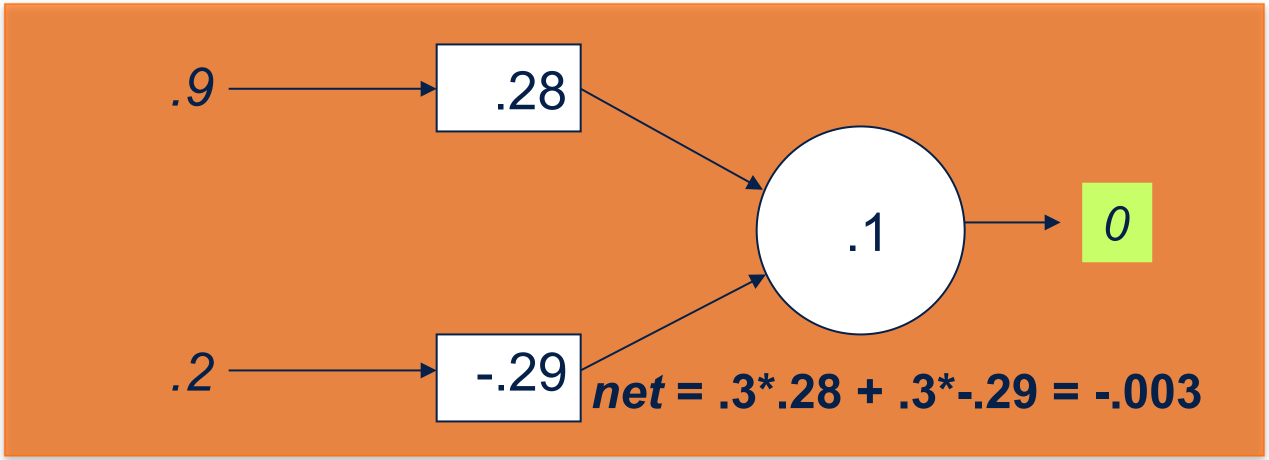
x_1	x_2	t
.9	.2	1
.3	.3	0

Test Set

Which pattern did we learn ?

$$\Delta w_i = c(t - z)\, x_i$$

$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$



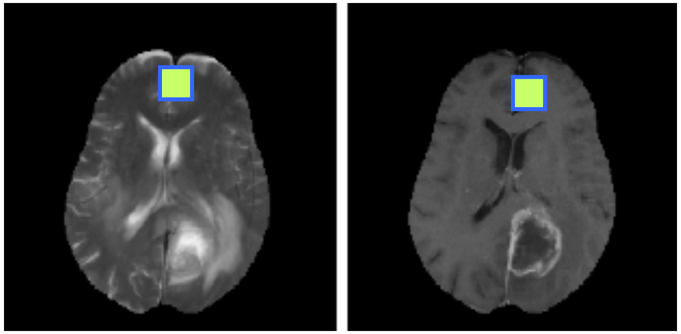
***net* = .3*.28 + .3*-.29 = -.003**

x_1	x_2	t
.8	.2	1
.4	.3	0

Training Set

x_1	x_2	t
.9	.2	1
.3	.3	0

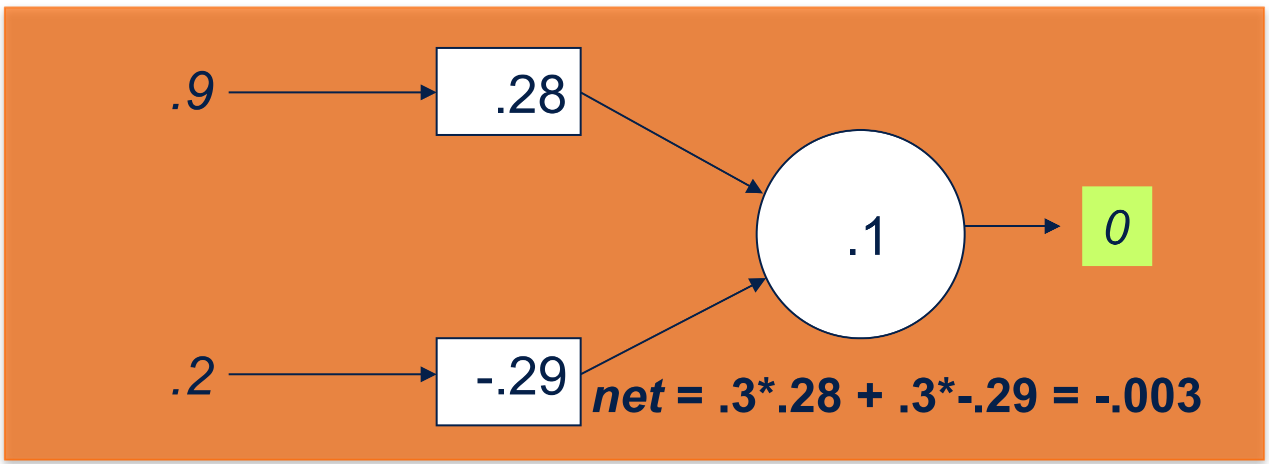
Test Set



Which pattern did we learn ?

$$\Delta w_i = c(t - z)\, x_i$$

$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$

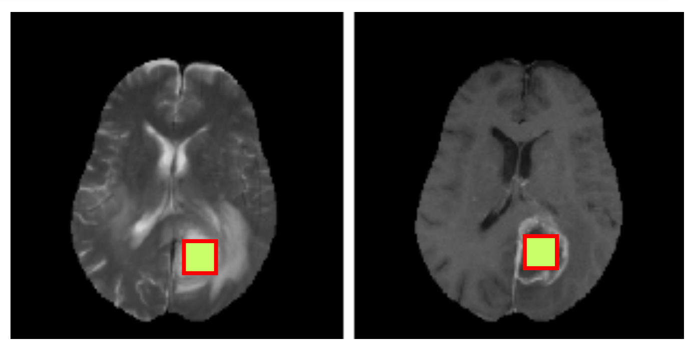


x_1	x_2	t
.8	.2	1
.4	.3	0

Training Set

x_1	x_2	t
.9	.2	1
.3	.3	0

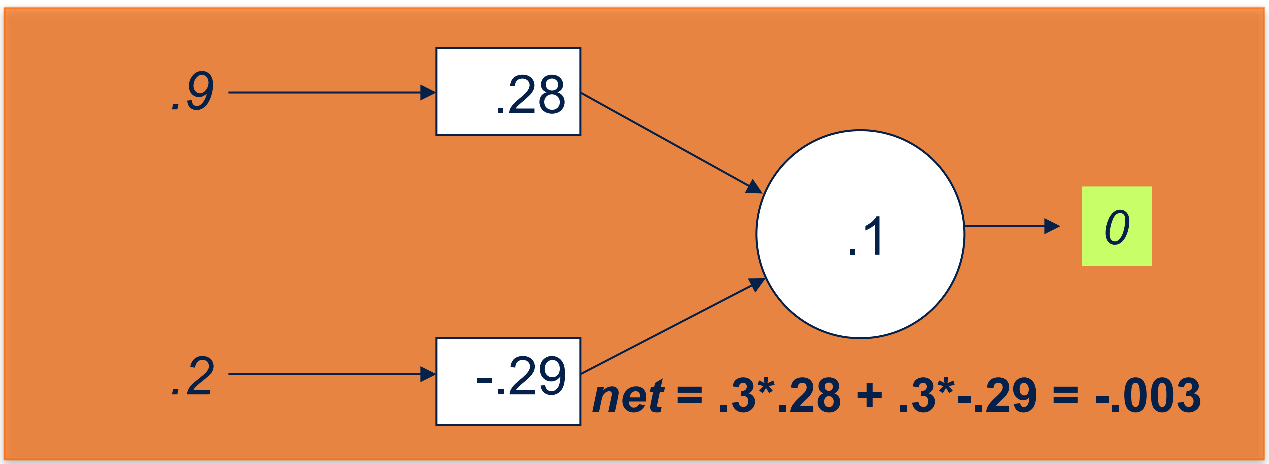
Test Set

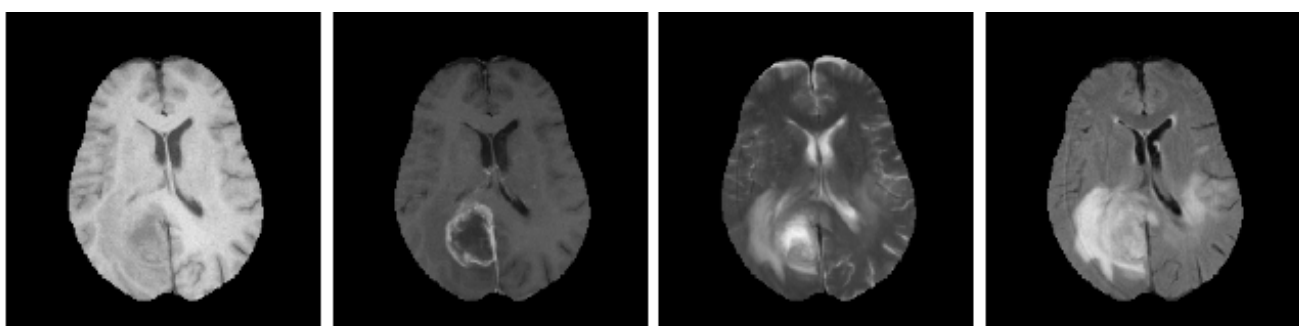


Add complexity: *input channels*

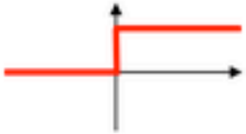
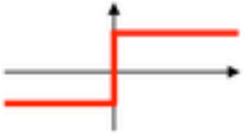
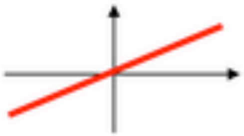
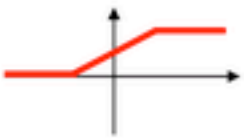
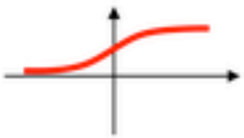
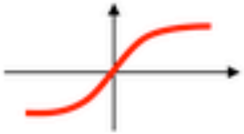
$$\Delta w_i = c(t - z)\, x_i$$

$$\Delta w_1 = 0.3*(0 - 1)*0.4 = -0.12$$
$$\Delta w_2 = 0.3*(0 - 1)*0.3 = -0.09$$



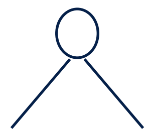
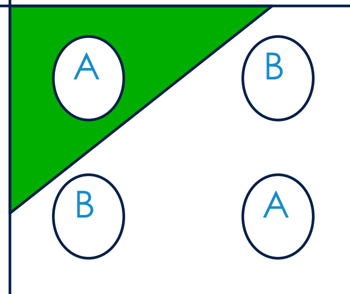
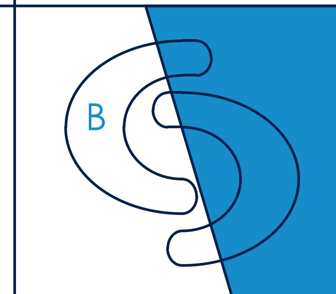
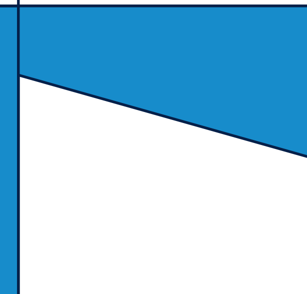
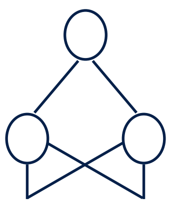
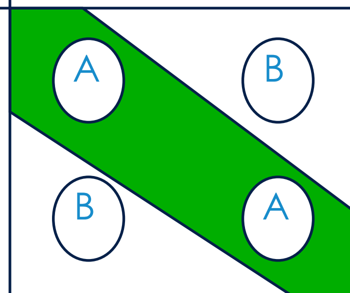
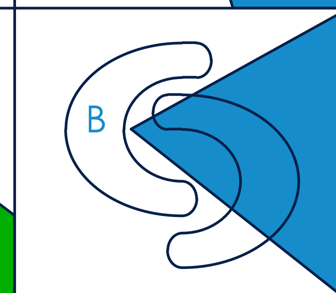
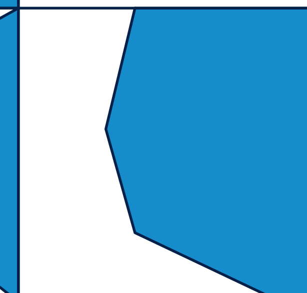
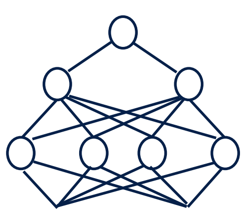
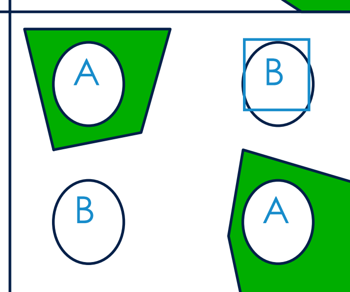
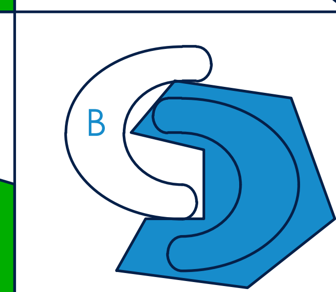
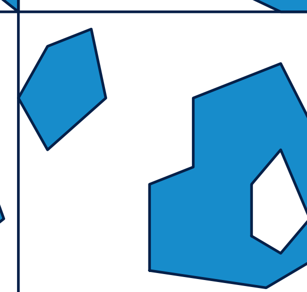


Add complexity: *activation functions*

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

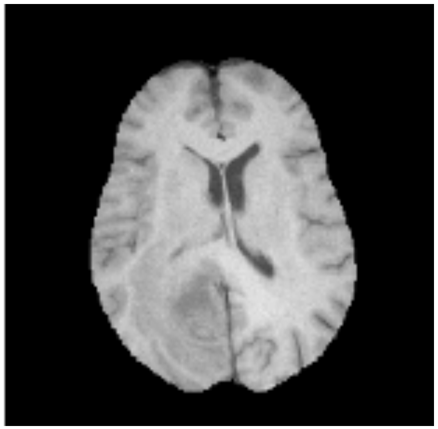
UCSF

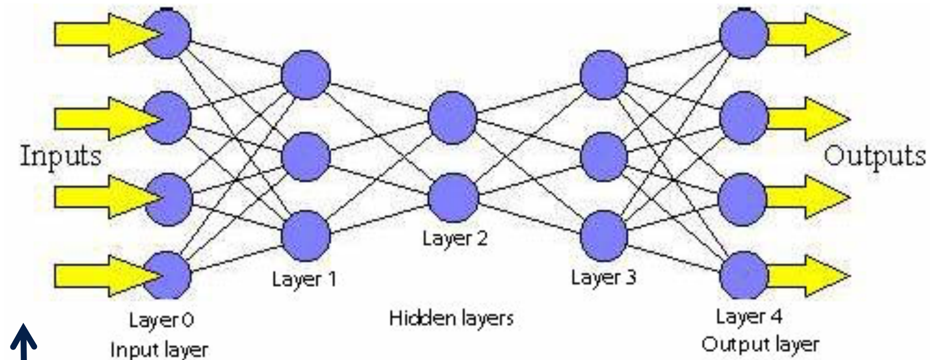
Multilayer Neural Network ...Why deep is better …

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyper plane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

UCSF

Data coming from pixel in a image are not **IID**

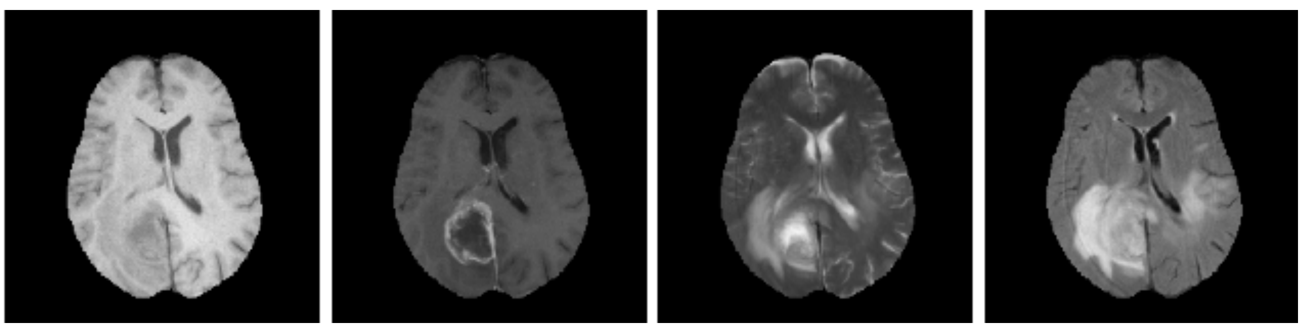




Classification

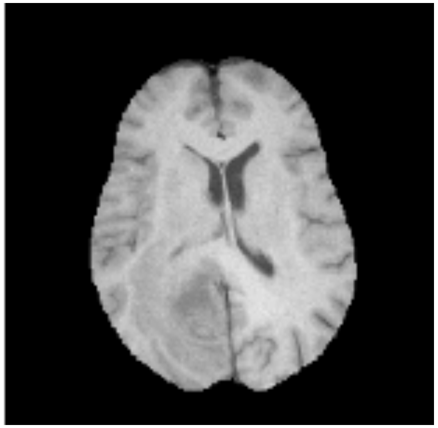
…in computer vision/medical imaging

???

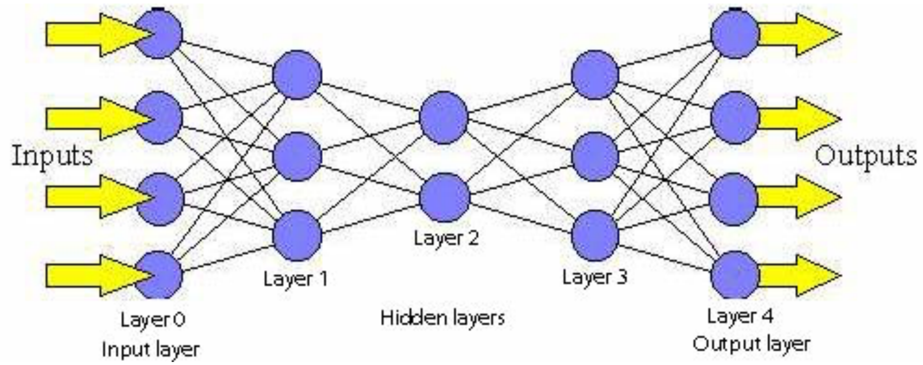


Maybe is not even 1 image but a stack of images or volumes …

Handcraft Feature vs Feature Learning



…in computer vision/medical imaging

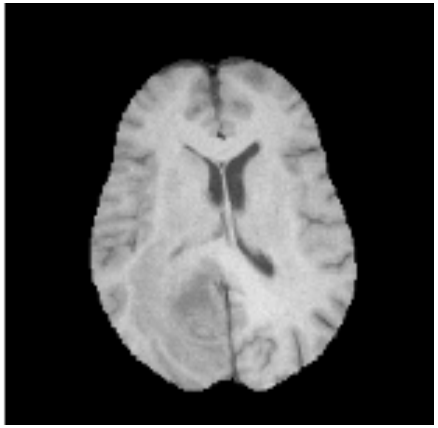


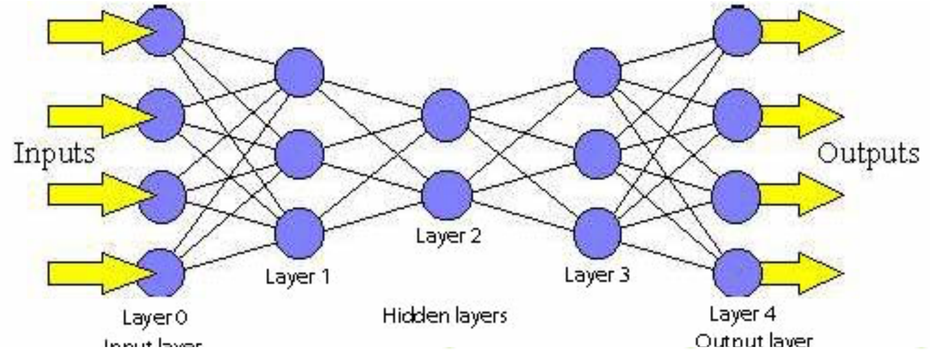
Classification

Feature Engineering

Moments: Average, std, entropy, kurtosis
Domain Transformation: FT, Wavelet, Spherical Harmonics
Filtering: Gabor Filters Fun Filters
Scale invariant feature transform **SIFT**

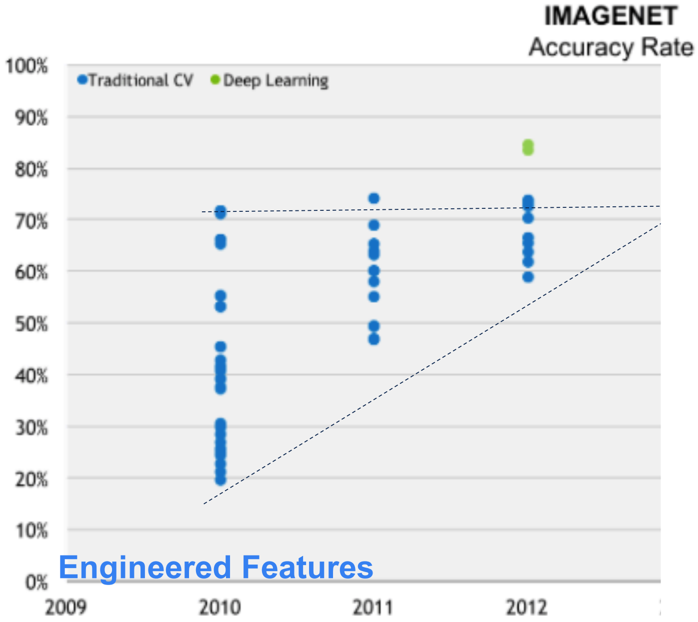
Handcraft Feature vs Feature Learning





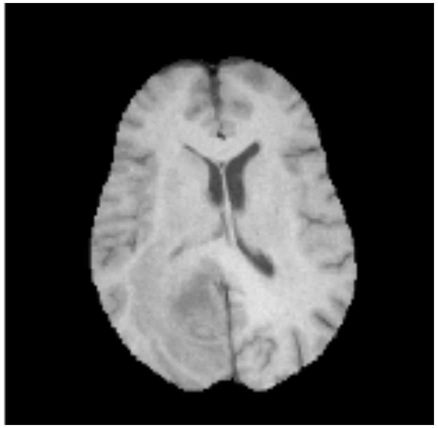
Classification

…in computer vision/medical imaging



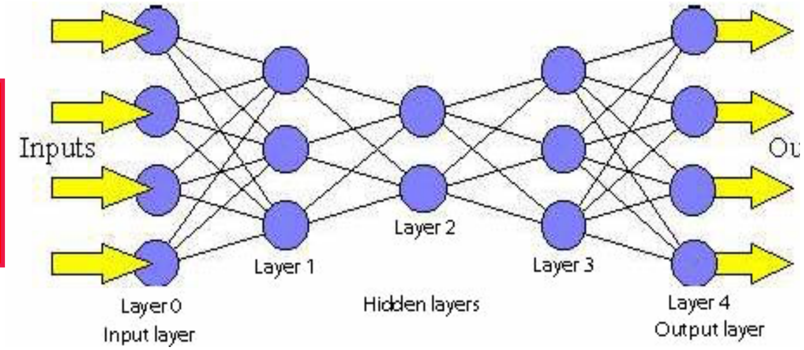
UCSF

Handcraft Feature vs Feature Learning



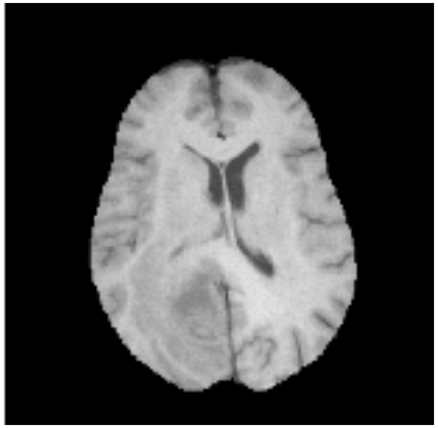
…in computer vision/medical imaging

Feature Learning



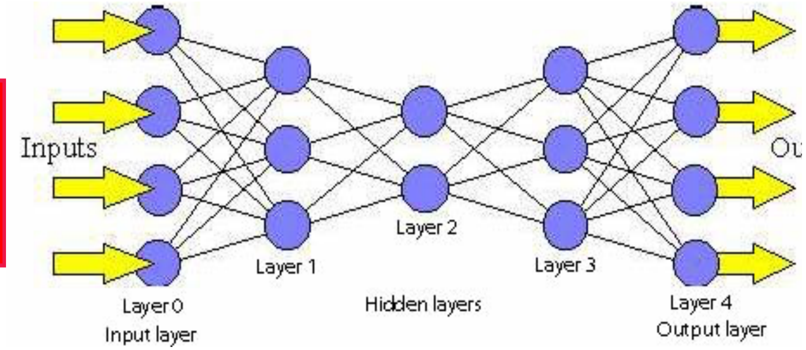
Classification

Handcraft Feature vs Feature Learning

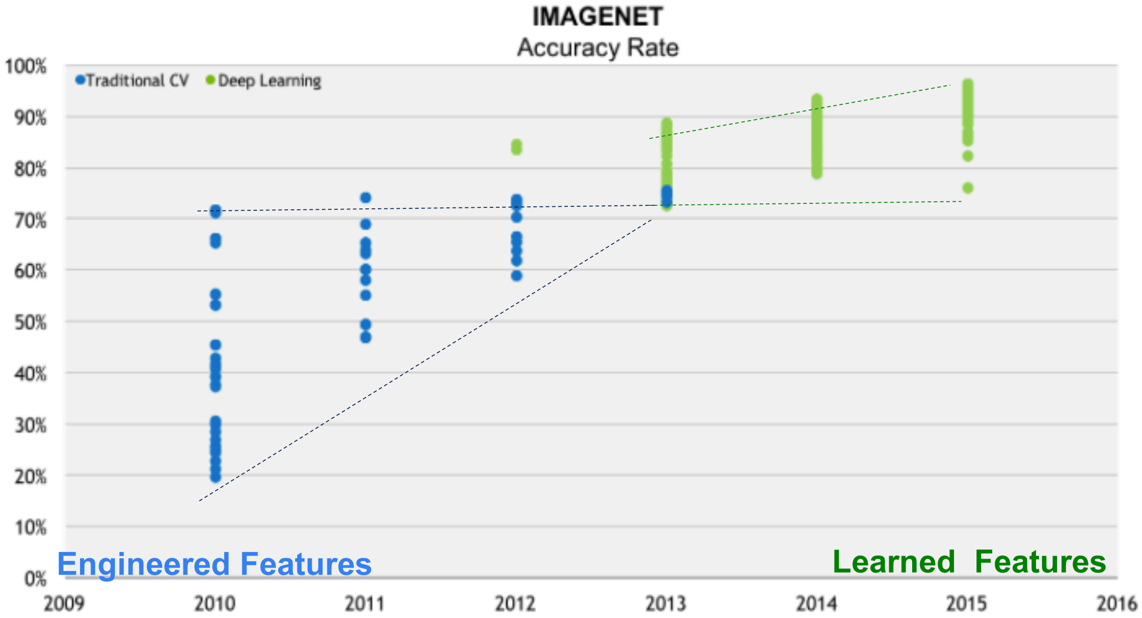


…in computer vision/medical imaging

Feature Learning



Classification



Convolutional Layers

- The core layer of CNNs
- The convolutional layer consists **of a set of filters**.
  - Each filter covers a spatially small portion of the input data.
- Each filter **is convolved across the dimensions of the input data**, producing a **multidimensional** feature map.
  - As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.
- the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.

Convolutional Layer

UCSF

Conv Hyperparameters

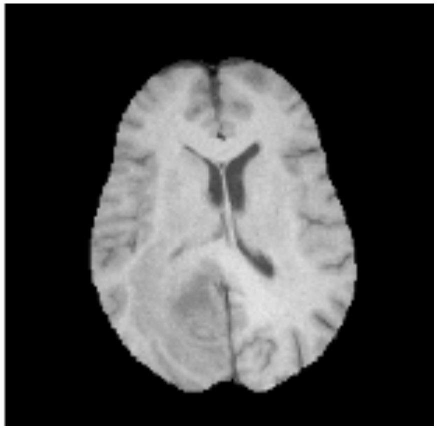
<u>Kernel Size:</u> controls how much **contextual information** is used to build feature mas

<u>Depth</u>: Depth corresponds to the **number of filters** we use for the convolution operation.

<u>Stride:</u> Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. **Having a larger stride will produce smaller feature maps.**

<u>Zero-padding:</u> Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero padding is that it allows us to control the size of the feature maps.
Adding zero-padding is also called ***wide convolution***, and not using zero-padding would be a ***narrow convolution***.

CNN Architecture



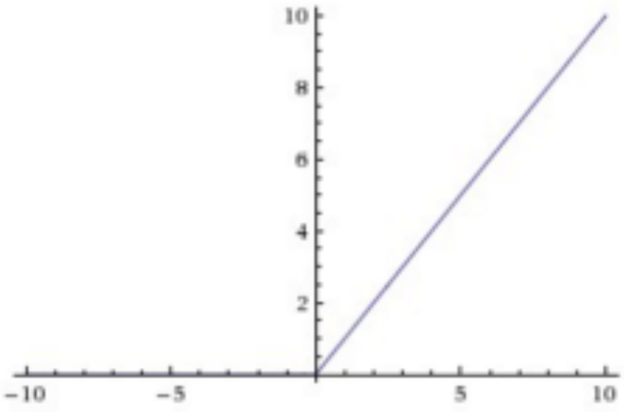
…in computer vision/medical imaging

5x5x20
Stride 2
Pad 0

Rectified Linear Unit

An additional operation called ReLU has been used after every Convolution operation ReLU stands for **Re**ctified **L**inear **U**nit and is a non-linear operation

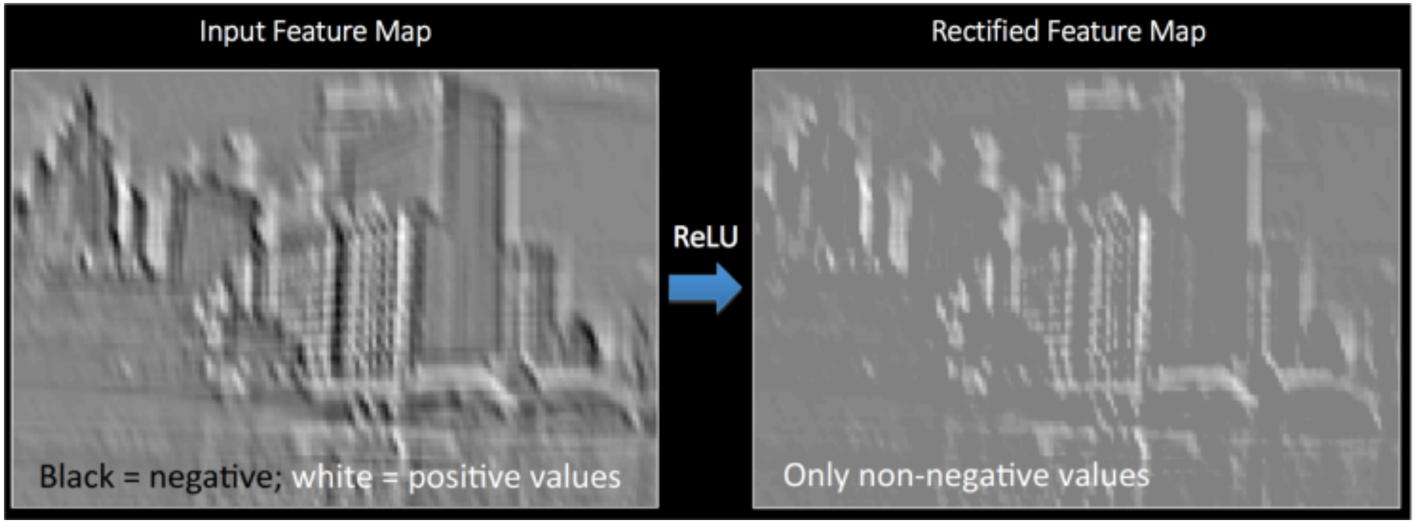
Output = Max(zero, Input)



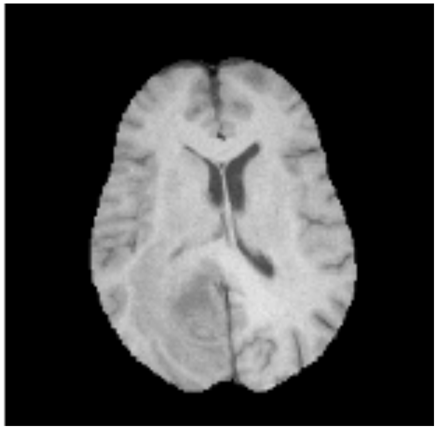
Rectified Linear Unit

ReLU is an element wise operation (**applied per pixel**)

The purpose of ReLU is to **introduce non-linearity** in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).



CNN Architecture



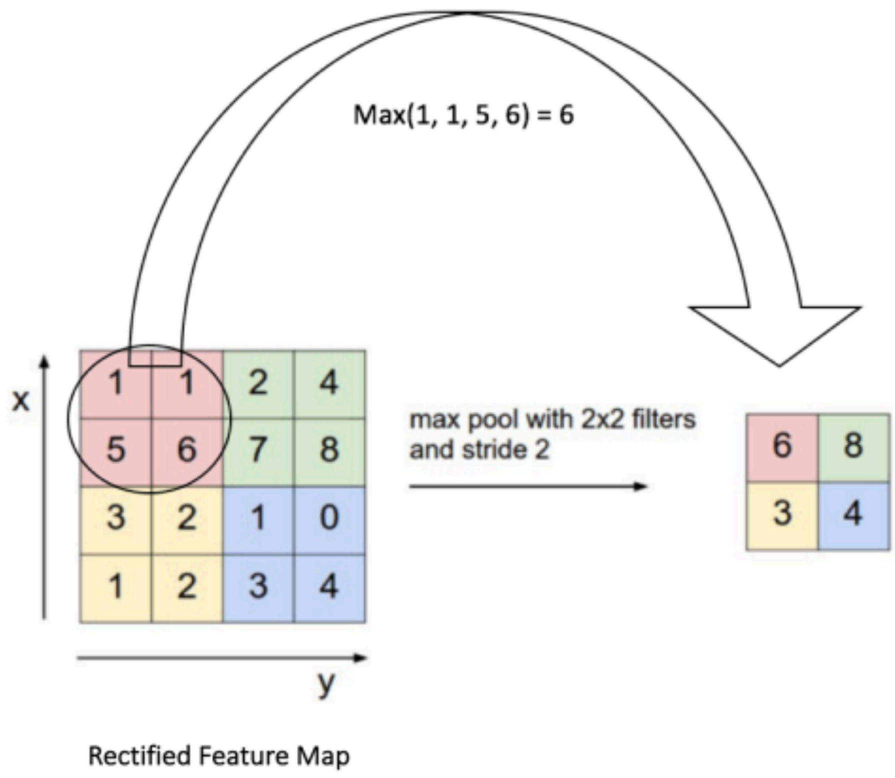
…in computer vision/medical imaging

ReLU

5x5x20
Stride 2
Pad 0

Pooling/Downsampling

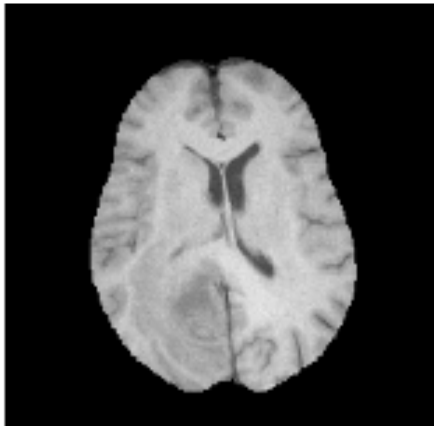
- Spatial Pooling (also called subsampling or downsampling) **reduces the dimensionality of each feature map** but retains the most important information.
- Spatial Pooling can be of different types: **Max, Average, Sum etc**.
- Can be learned
- It is not strictly necessary
- **<u>Hyperparameters</u>**:
  - Type, Filer Size, Stride



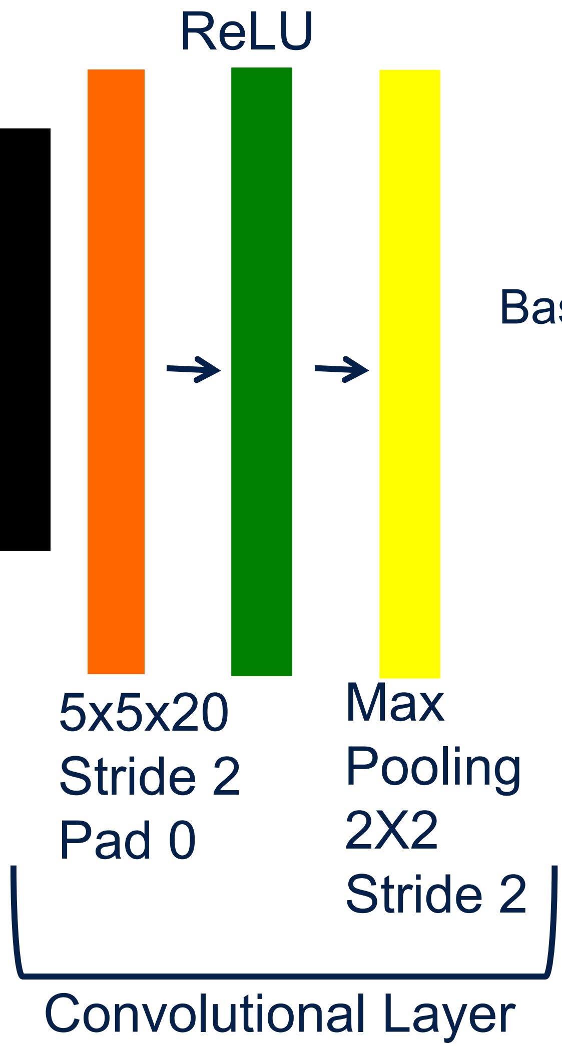
Pooling/Downsampling

- Pooling makes the input representations (feature dimension) smaller and more manageable.
- Reduces the number of parameters and computations in the network, therefore, controlling overfitting
- Makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).

CNN Architecture

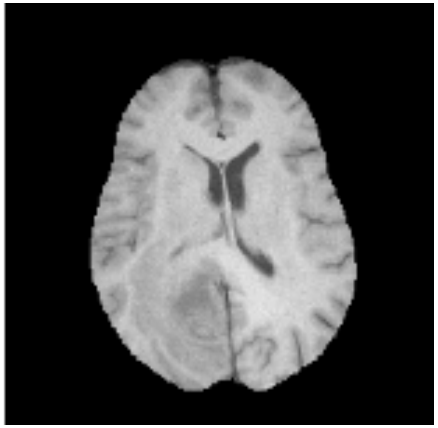


…in computer vision/medical imaging

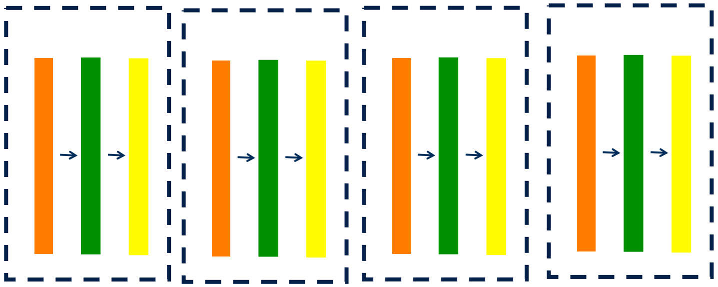


Basic building blocks of any CNN

CNN Architecture

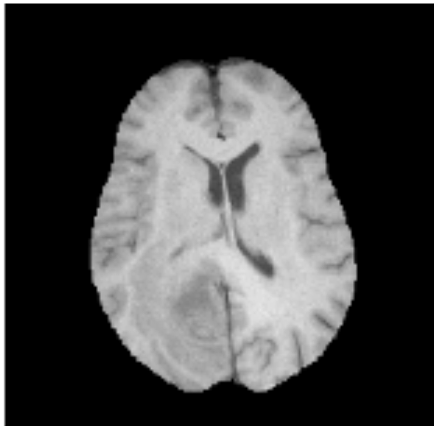


…in computer vision/medical imaging

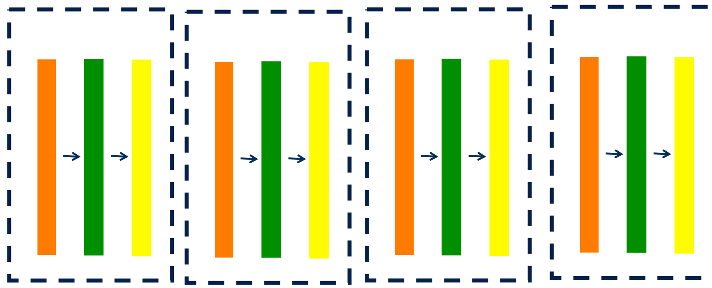


Several convolutional layers are used to build a **Deep** CNN with different hyper parameters

CNN Architecture



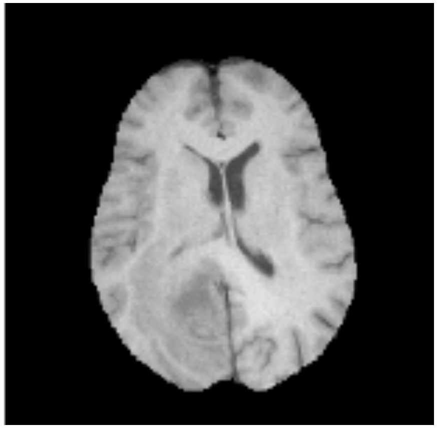
…in computer vision/medical imaging



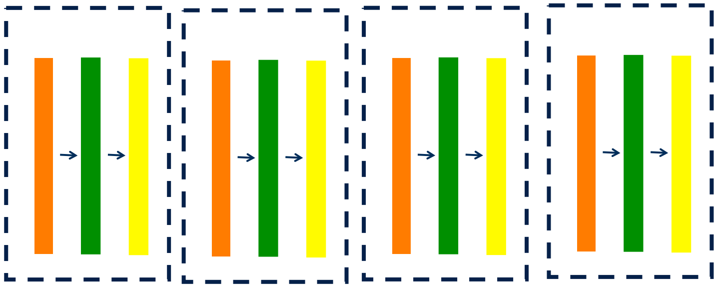
Several convolutional layers are used to build a **Deep** CNN with different hyper parameters

Feature Learning

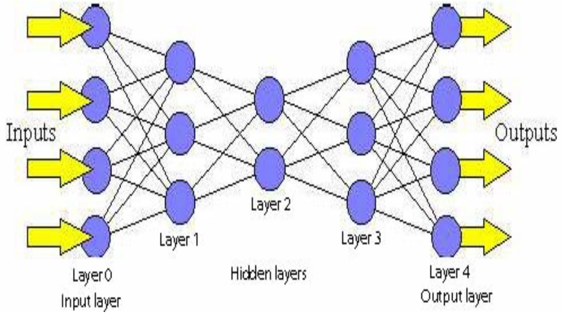
CNN Architecture



…in computer vision/medical imaging



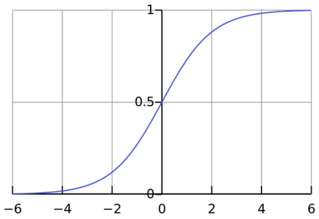
Several convolutional layers are used to build a **Deep** CNN with different hyper parameters



Fully Connected Layer

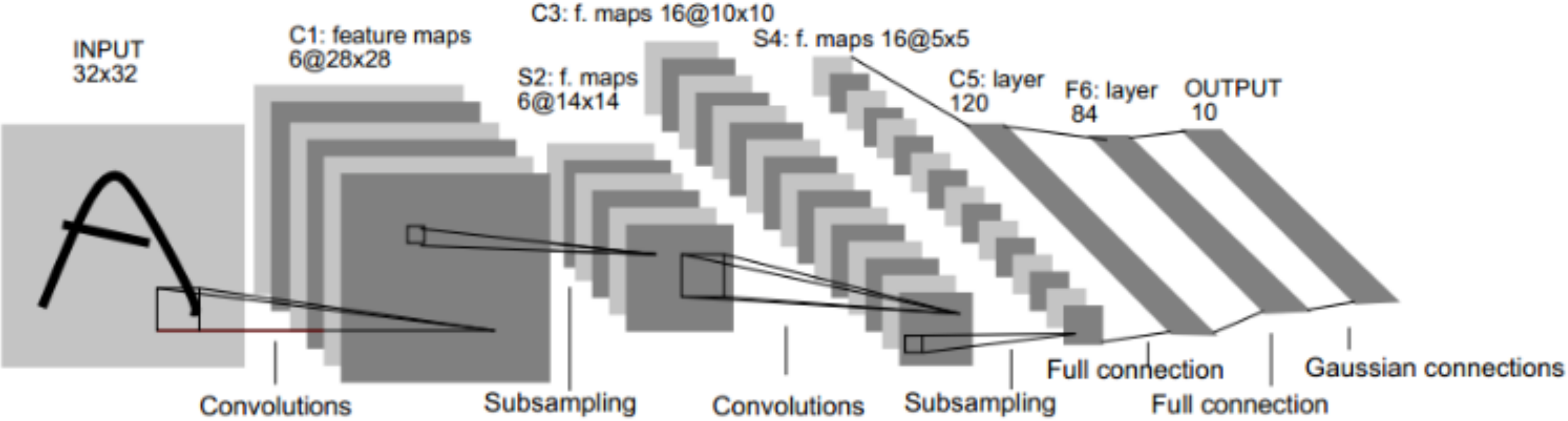
Feature Learning

Classifier



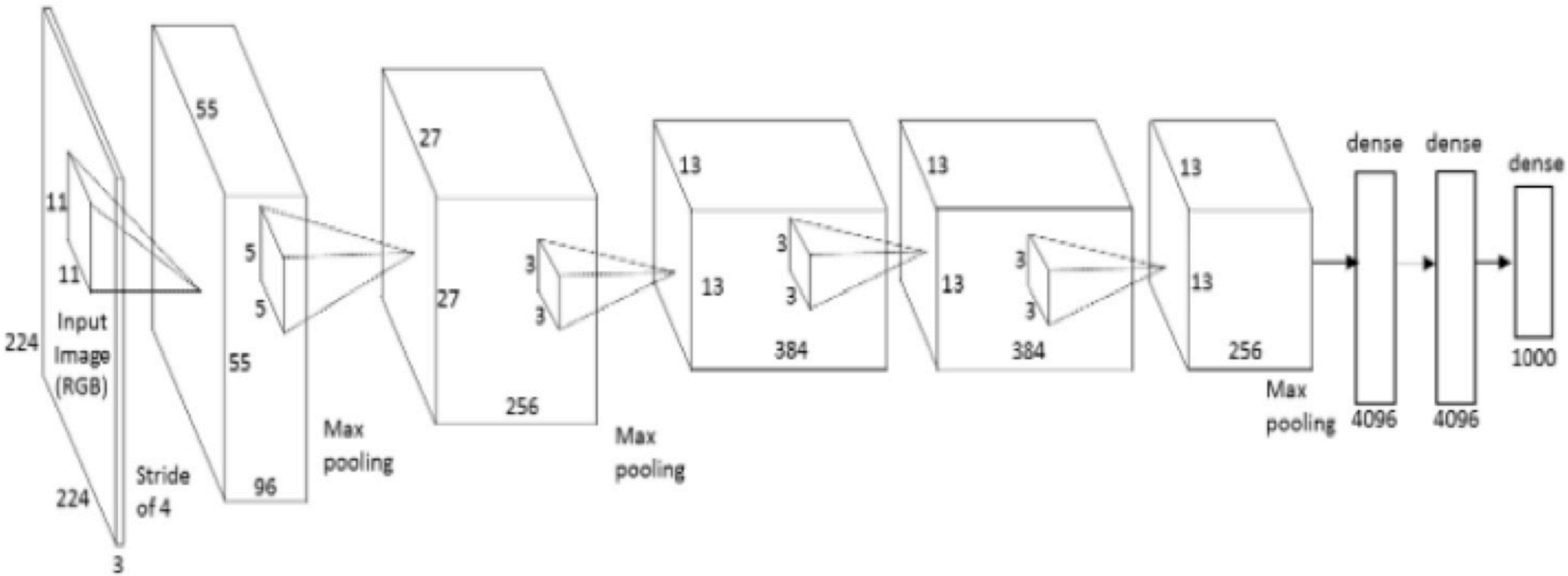
The Fully Connected layer is a traditional **Multi Layer Perceptron** that uses a **Softmax** activation function in the output layer

LeNet



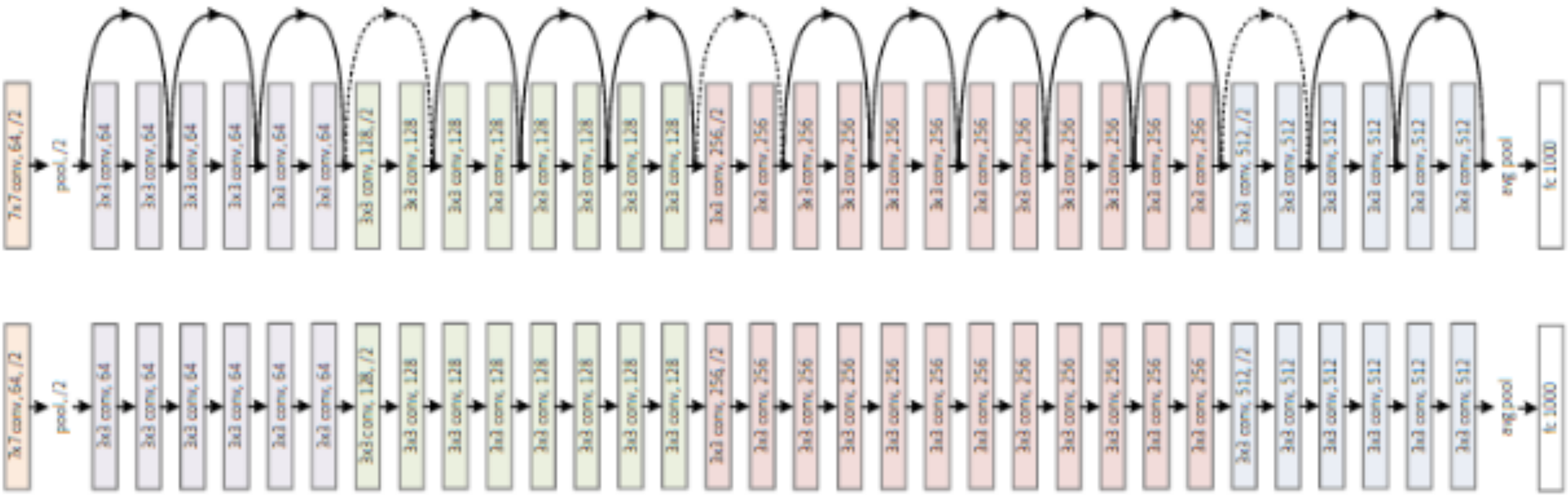
INPUT 32x32
C1: feature maps 6@28x28
S2: f. maps 6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer 120
F6: layer 84
OUTPUT 10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

AlexNet

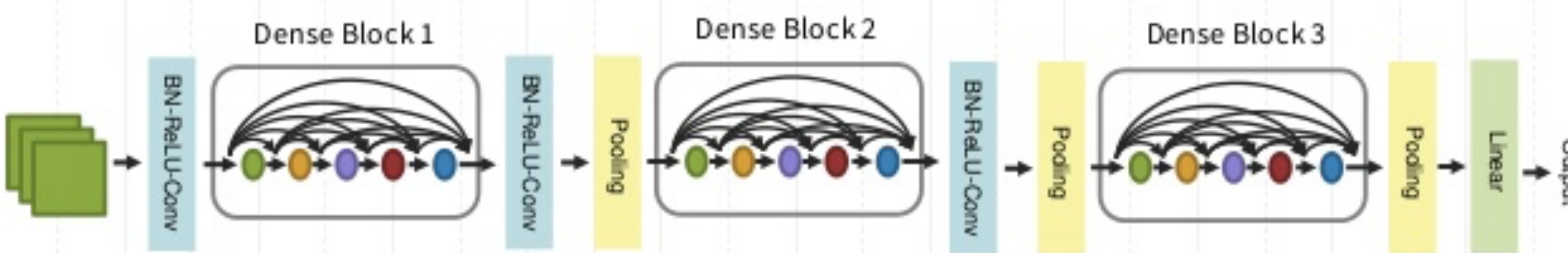


11
11
224
Input
Image
(RGB)
224
3
Stride
of 4
55
5
5
55
96
Max
pooling
27
27
256
3
3
Max
pooling
13
13
384
3
3
13
13
384
3
3
13
13
256
Max
pooling
dense
dense
dense
4096
4096
1000

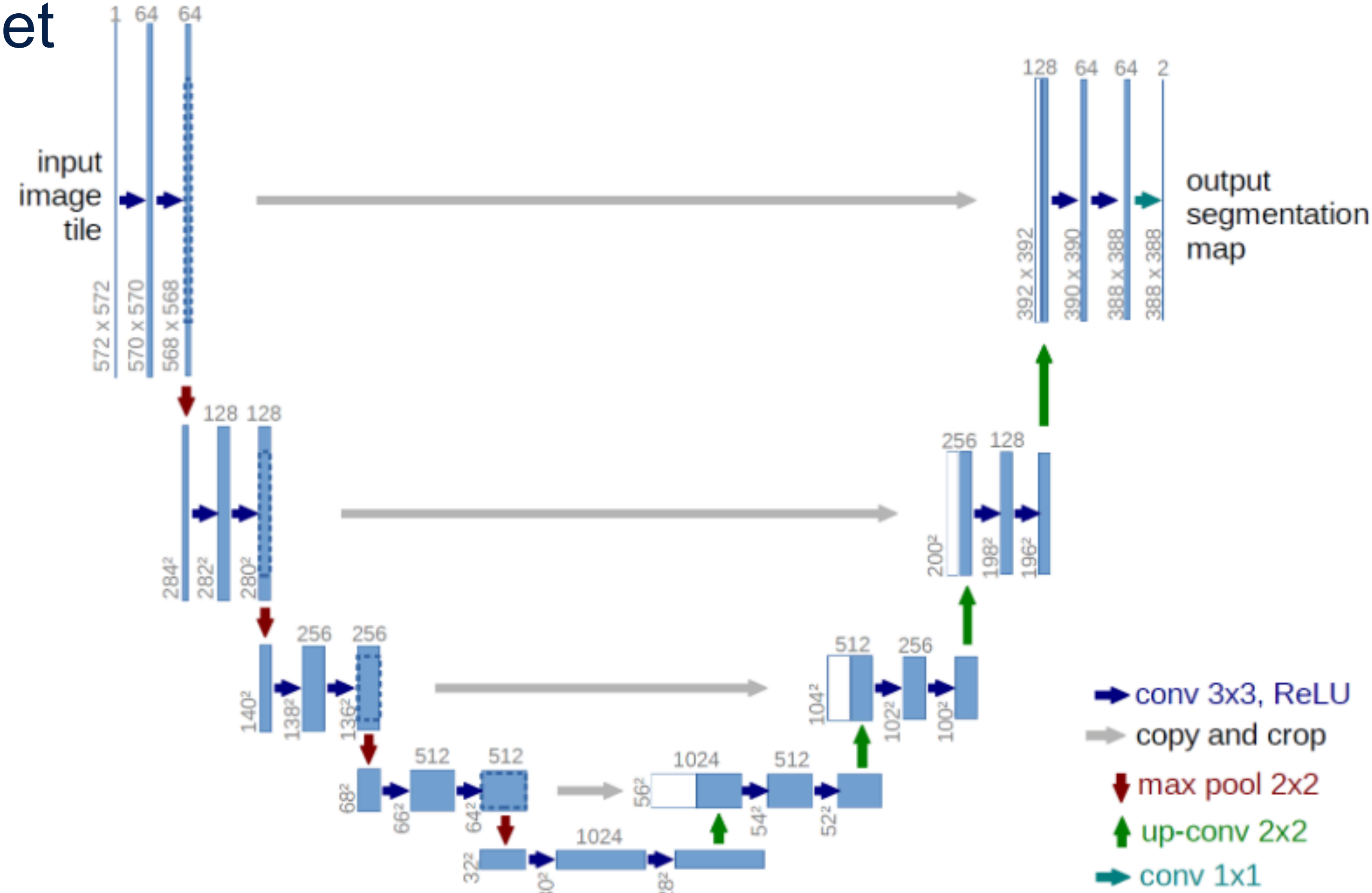
ResNet



DenseNet

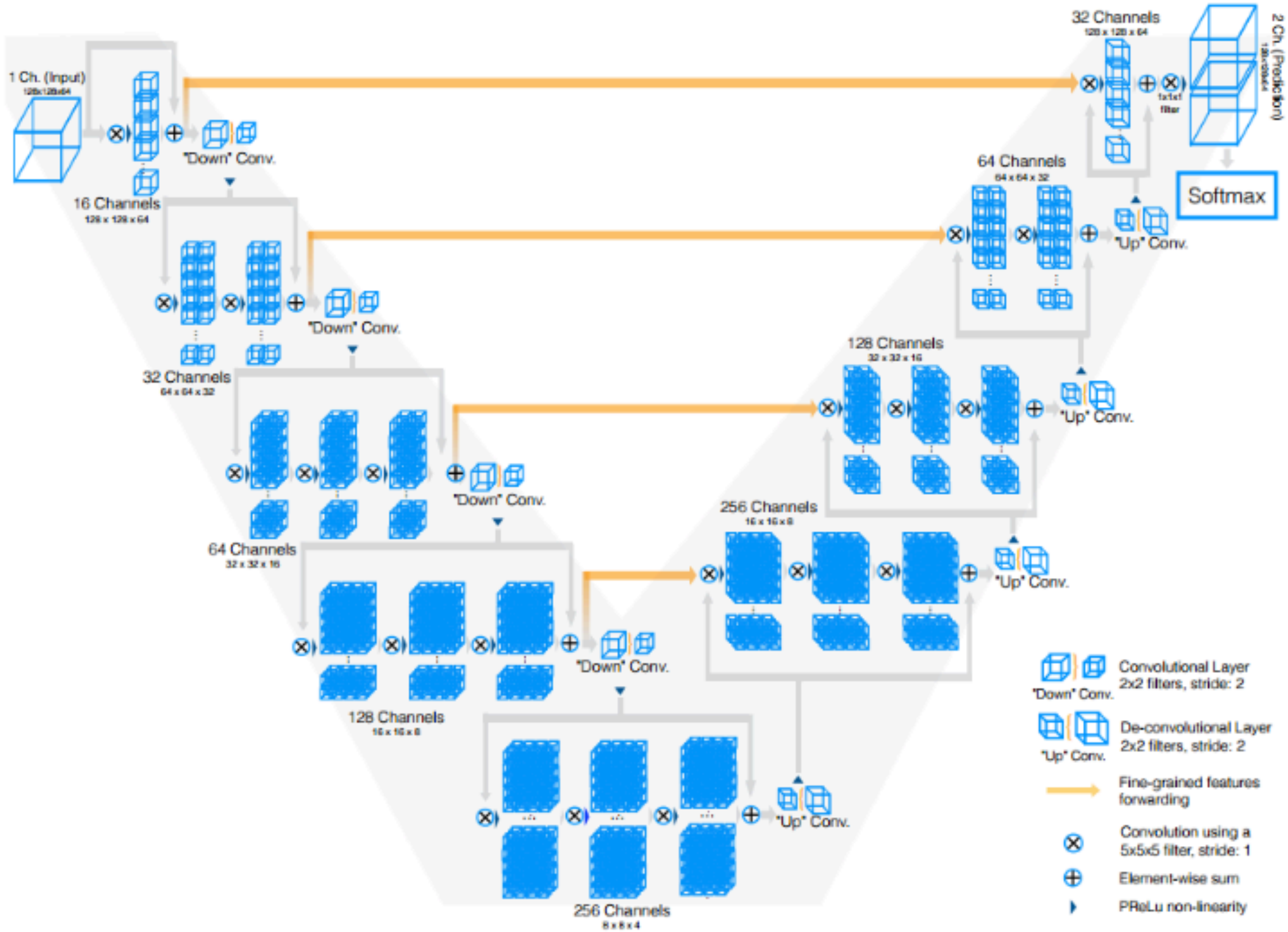


U-net



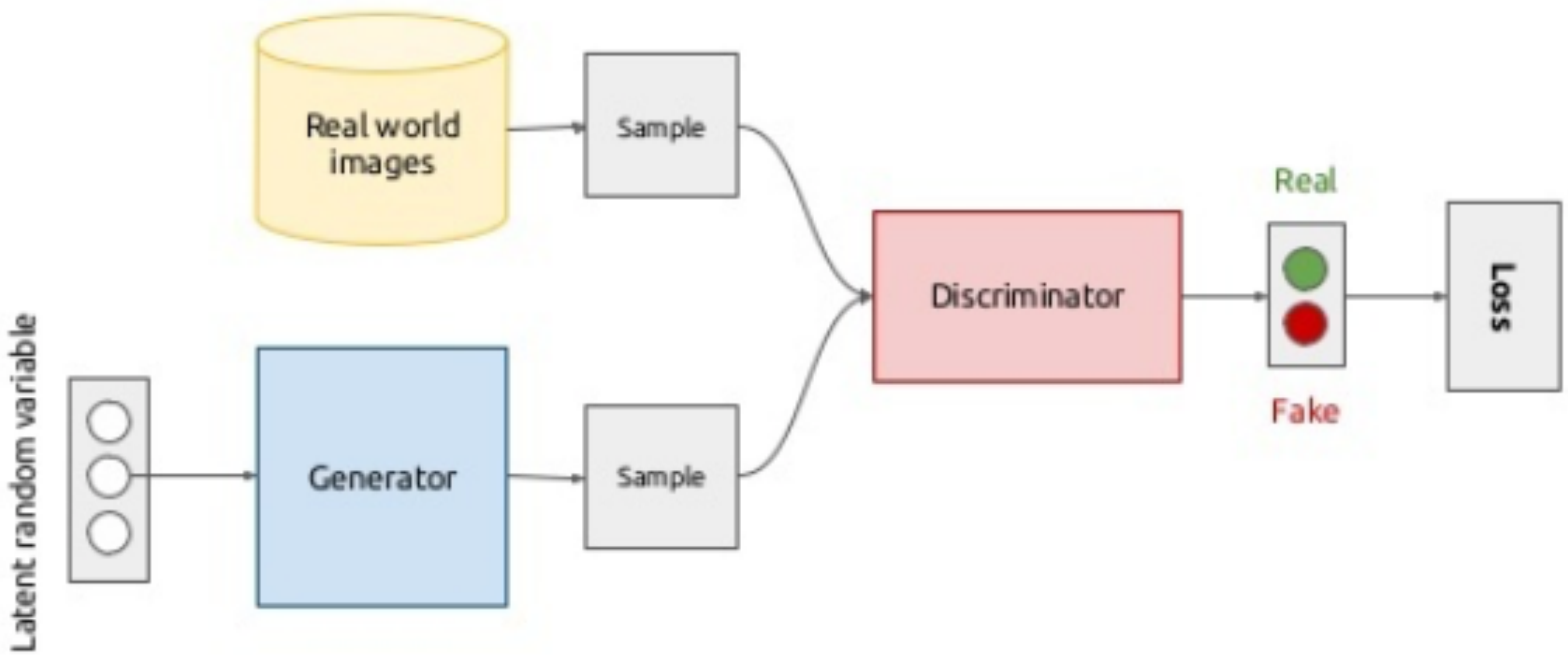
1
64
64
128
64
64
2
input
image
tile
output
segmentation
map
572 x 572
570 x 570
568 x 568
392 x 392
390 x 390
388 x 388
388 x 388
128
128
256
128
284²
282²
280²
200²
198²
196²
256
256
512
256
140²
138²
136²
104²
102²
100²
512
512
1024
512
68²
66²
64²
56²
54²
52²
1024
32²
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

V-net



1 Ch. (Input)
16 Channels
128 x 128 x 64
"Down" Conv.
32 Channels
64 x 64 x 32
64 Channels
32 x 32 x 16
128 Channels
16 x 16 x 8
256 Channels
8 x 8 x 4
256 Channels
16 x 16 x 8
"Up" Conv.
128 Channels
32 x 32 x 16
64 Channels
64 x 64 x 32
32 Channels
128 x 128 x 64
1x1x1 filter
2 Ch. (Prediction)
Softmax
Convolutional Layer
2x2 filters, stride: 2
De-convolutional Layer
2x2 filters, stride: 2
Fine-grained features forwarding
Convolution using a 5x5x5 filter, stride: 1
Element-wise sum
PReLu non-linearity

Generative adversarial network (**GAN**)



2D-3D, Deep and Machine learning hybrid pipeline