

1/14 3:18:43 ***



Machine learning

A gentle introduction





Types of AI

Here is one way [after Arend Hintze] to classify AI types (not just techniques!)..

Type I AI: **Reactive machines** - make optimal moves - no memory, no past 'experience'.

Type II AI: **Limited memory** - human-compiled , one-shot 'past' 'experiences' are stored for lookup.

 Type III AI: **Theory of Mind** - "the understanding that people, creatures and objects in the world can have thoughts and emotions that affect the AI programs' own behavior".


 Type IV AI: **Self-awareness** - machines that have consciousness, that can form representations about themselves (and others).



As of now, types III and IV are in the realm of speculation and science-fiction, but appear to be certainty in the near term, in the general public's mind :)



What is ML?



Machine learning focuses on the construction and study of systems that can **learn from data** to optimize a **performance function**, such as optimizing the expected reward or minimizing loss functions. The goal is to **develop deep insights from data assets faster, extract knowledge from data with greater precision, improve the bottom line and reduce risk.** 

- Wayne Thompson, SAS



Types of ML

The following notes are from various sources..

The key types of machine learning include: 

- Supervised learning 
- Unsupervised learning
- Semisupervised learning
- Reinforcement learning 

 Supervised learning algorithms are "trained" using examples where in addition to features [inputs], the **desired output** [label, aka target] is known. 



 Unsupervised learning is a type of machine learning where the **system operates on unlabeled examples**. In this case, the system is **not told the "right answer."** The algorithm tries to find a hidden structure or manifold in unlabeled data. The goal of **unsupervised learning** is to explore the data to find 



- ☞ intrinsic structures within it using methods like clustering or dimension reduction.
- ☞ For Euclidian space data: k-means clustering, Gaussian mixtures and principal component analysis (PCA)
- ☞ For non-Euclidian space data: ISOMAP, local linear embedding (LLE), Laplacian eigenmaps, kernel PCA.



Use matrix factorization, topic models/graphs for social media data.

- ☞ Semisupervised learning is used for the same applications as supervised learning. But this technique **uses both labeled and unlabeled data for training** - typically, a small amount of labeled data with a large amount of unlabeled data. The **primary goal is unsupervised learning** (clustering, for example), and labels are viewed as side information (cluster indicators in the case of clustering) to help the algorithm find the right intrinsic data structure. image analysis - for

example, identifying a person's face on a webcam - textual analysis, and disease detection.


With  reinforcement learning, the  algorithm discovers for itself which actions yield the greatest rewards through trial and error. Reinforcement learning has three primary components:

1. The agent - the **learner** or decision maker.
-  2. The environment - everything the agent interacts with.
-  3. Actions - what the agent can do.


 The objective is for the agent to choose **actions** that  **maximize the expected reward** over a given period of time. The agent will reach the goal much quicker by following a good policy, so the goal in reinforcement learning is to learn the best policy. Reinforcement learning is often used for robotics and navigation.


Markov decision processes (MDPs) are popular models used in reinforcement learning. MDPs assume the state of the environment is perfectly observed by the agent. When this is not the case, we can use a more general model called partially observable MDPs (or POMDPs).

Here is a WIRED mag writeup on unsupervised learning..

Let's say, for example, that you're a researcher who wants to learn more about human personality types. You're awarded an extremely generous grant that allows you to give  200,000 people a 500-question personality test, with answers that vary on a scale from one to 10. Eventually you find yourself with 200,000 data points in 500 virtual "dimensions" - one dimension for each of the original questions on the personality quiz. These points, taken together, form a lower-dimensional "surface" in the 500-dimensional space in the same way that a simple plot of

elevation across a mountain range creates a two-dimensional surface in three-dimensional space.










What you would like to do, as a researcher, is identify this lower-dimensional surface, thereby reducing the personality portraits of the 200,000 subjects to their essential properties - a task that is similar to finding that two variables suffice to identify any point in the mountain-range surface. Perhaps the personality-test surface can also be described with a simple function, a connection between a number of variables that is significantly smaller than 500. This function is likely to reflect a hidden structure in the data. 

In the last 15 years or so, researchers have created a number of tools to probe the geometry of these hidden structures. For example, you might build a model of the surface by first zooming in at many different points. At each point, you  would place a drop of virtual **ink** on the surface and watch

how it spread out. Depending on how the surface is curved at each point, the ink would diffuse in some directions but not in others. If you were to connect all the drops of ink, you would get a pretty good picture of what the surface looks like as a whole. And with this information in hand, you would no longer have just a collection of data points. Now you would start to see the connections on the surface, the interesting loops, folds and kinks. This would give you a map.


Common ML algorithms

To reiterate, these are the most common ML algorithms:

- Linear Regression 
- Logistic Regression 
- Decision Trees 
- SVM 
- Naive Bayes 
- KNN 
- K-Means Clustering 
- Random Forest 
- Gradient Boost, Adaboost 
- **Neural nets ("NN, ANN, DNN, CNN, RNN..")**



What is Deep Learning?


"Deep learning is currently one of the best providers of solutions regarding problems in image recognition, speech recognition, object recognition, and natural language with its increasing number of libraries that are available in Python. The aim of deep learning is to develop deep neural networks by increasing and improving the number of training layers for each network, so that a machine learns more about the data until it's as accurate as possible. Developers can avail the techniques provided by deep learning to accomplish complex machine learning tasks, and train AI networks to develop deep levels of perceptual recognition." 

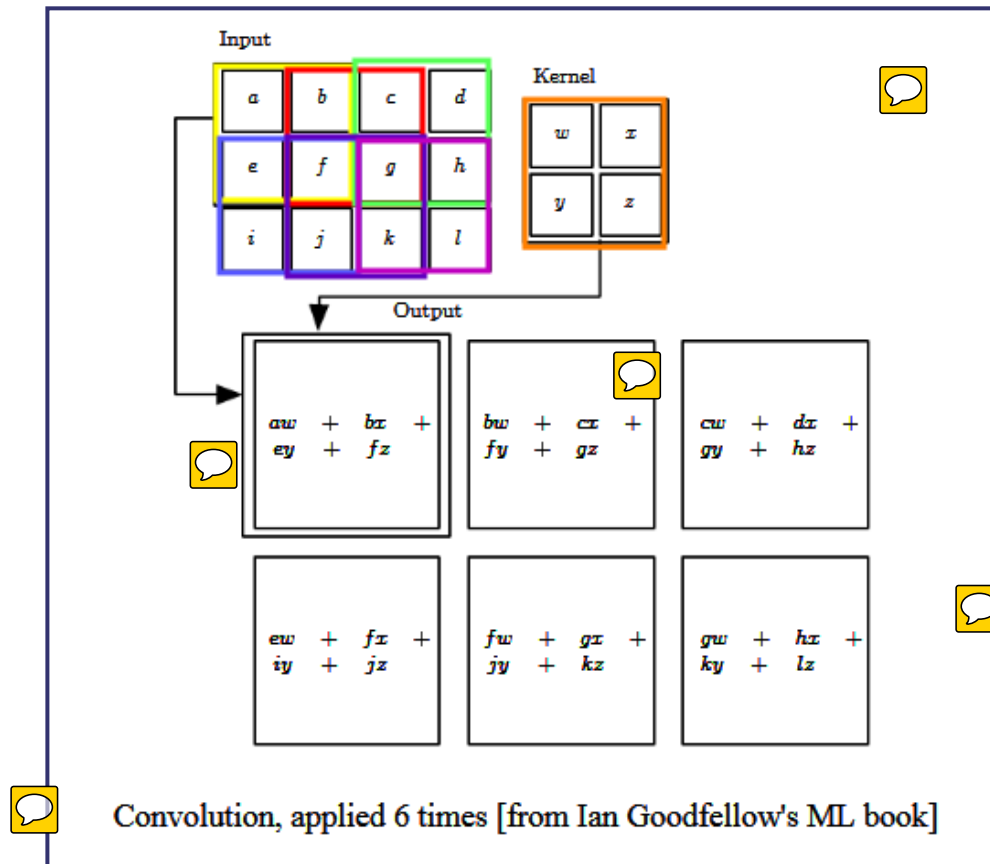
Q: so what makes it 'deep'? A: the number of intermediate layers of neurons. 

Convolution

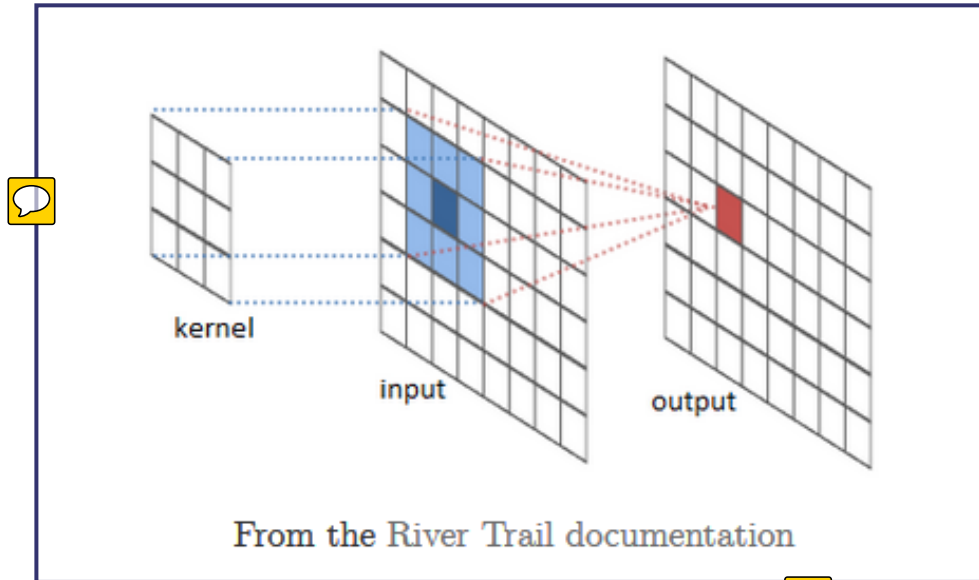


A convolution is a **blending (or integrating) operation between two functions** (or signals or numerical arrays) - one function is convolved  (pointwise-multiplied) with another, and **the results summed.** 

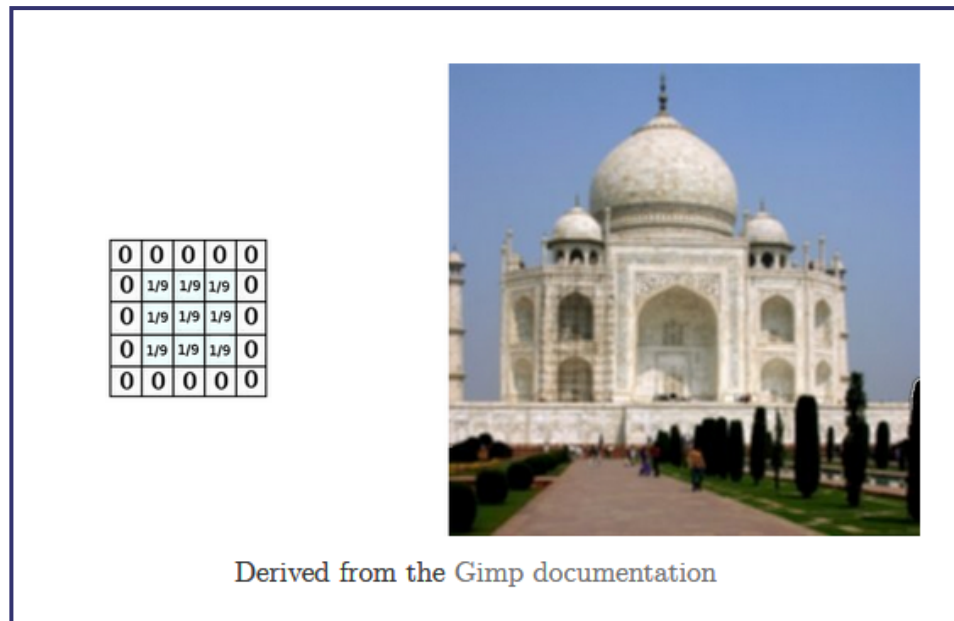
Here is an example of convolution - the 'Input' function [with discrete array-like values] is convolved with a 'Kernel'  function [also with a discrete set of values] to produce a result; **here this is done six times:**



Convolution is used heavily in creating image-processing filters for blurring, sharpening, edge-detection, etc. The to-be-processed image represents the convolved function, and a 'sliding' "mask" (grid of weights), the convolving function (aka convolution kernel):



Here is [the result of] a **blurring** operation:



Here you can fill in your own weights for a kernel, and examine the resulting convolution.

So - how does this relate to neural nets? In other words, what are CNNs?

CNN (Convolutional Neural Network, aka ConvoNet)

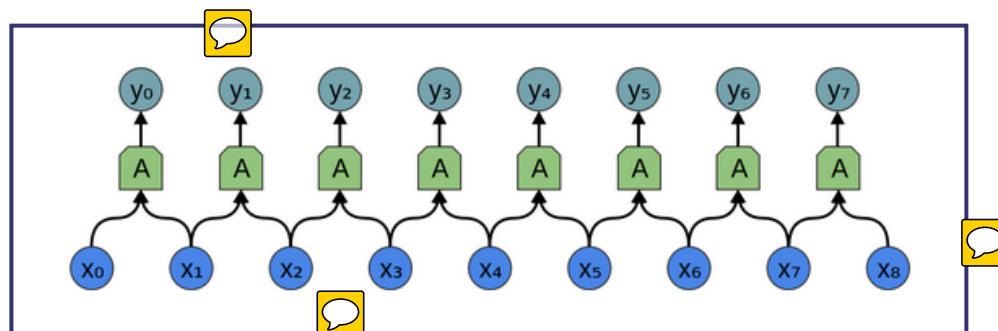
In essence, a CNN is where we represent a neuron's weights as a matrix (kernel), and slide it (IP-style) over an input (an image, a piece of speech, text, etc.) to produce a convolved output.

In what sense is a neuron's weights, a convolution kernel?

We know that for an individual neuron, its output y is expressed by

$y = x_0 \cdot w_0 + w_1 \cdot x_1 + \dots + w_n \cdot x_n + b$, where the w s represent the neuron's weights, and the x s, the incoming signals [b is the neuron's activation bias]. The multiplications and summations resemble a convolution! The incoming 'function' is $[x_0, x_1, x_2, \dots, x_n]$, and the neuron's kernel 'function', $[w_0, w_1, w_2, \dots, w_n]$.

Eg. if the kernel function is $[0, 0, 0 \dots w_0, w_1, 0, 0 \dots]$ [where we only process our two nearest inputs], the equivalent network would look like so [fig from Chris Olah]:



The above could be considered one 'layer' of neurons, in a multi-layered network. The convolution (each neuron's application of w_0 and w_1 to its inputs) would produce the following:

$$y_0 = x_0 \cdot w_0 + x_1 \cdot w_1 + b_0$$

$$y_1 = x_1 \cdot w_0 + x_2 \cdot w_1 + b_1$$

$$y_2 = x_2 \cdot w_0 + x_3 \cdot w_1 + b_2$$

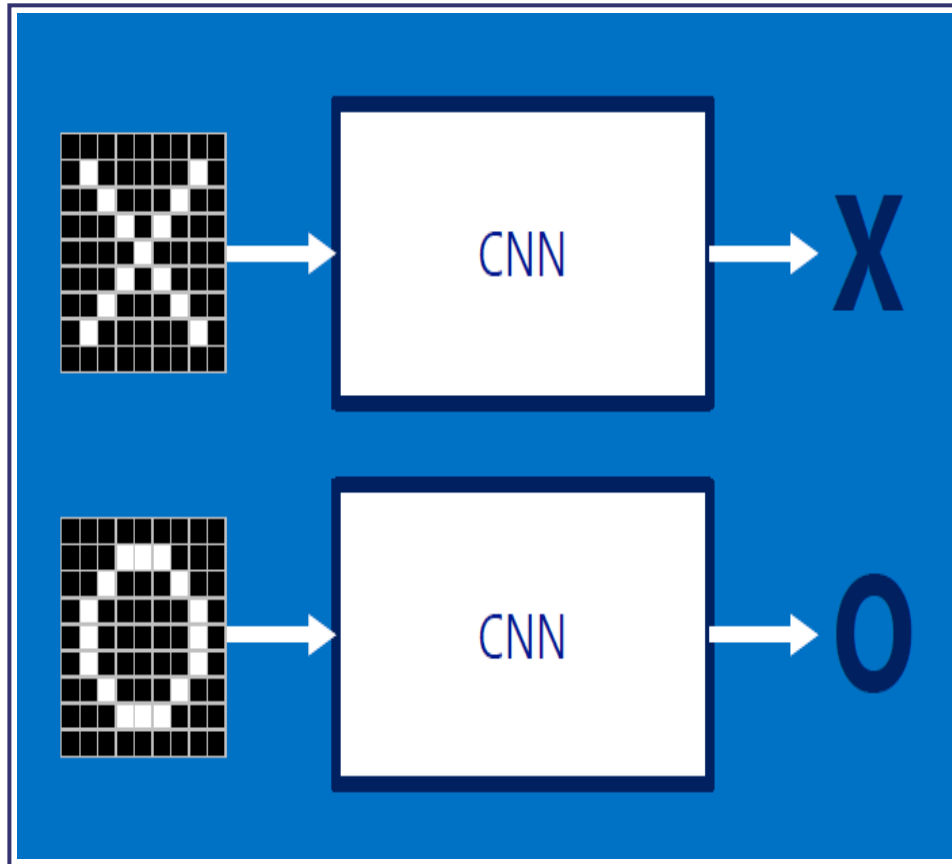
....

Pretty cool, right? Treating the neuron as a kernel function provides a convenient way to represent its weights as an array. For 2D inputs such as images, speech and text, the kernels would be 2D arrays that are coded to detect specific features (such as a vertical edge, color..).

EACH NEURON IS CONVOLVED OVER THE ENTIRE INPUT (again, IP-style), **AND AN OUTPUT IS GENERATED FROM ALL THE CONVOLUTIONS.** The output gets 'normalized' (eg. clamped), and 'collapsed' (reduced in size, aka 'pooling'), and the process repeats down several layers of neurons: input -> convolve -> normalize -> reduce/pool -> convolve -> normalize -> reduce/pool -> ... -> output.

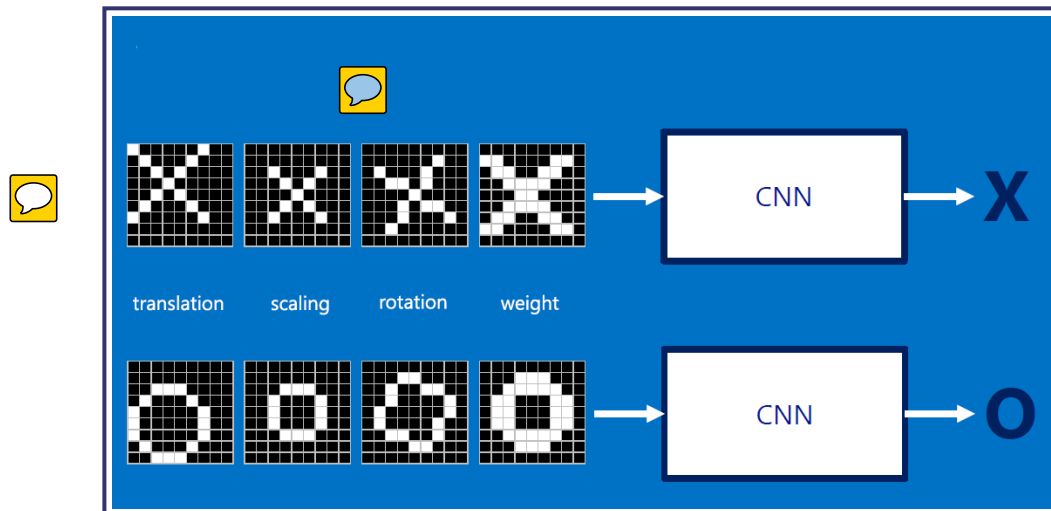
The following pics are from a talk by Brandon Rohrer (Microsoft).

What we want:

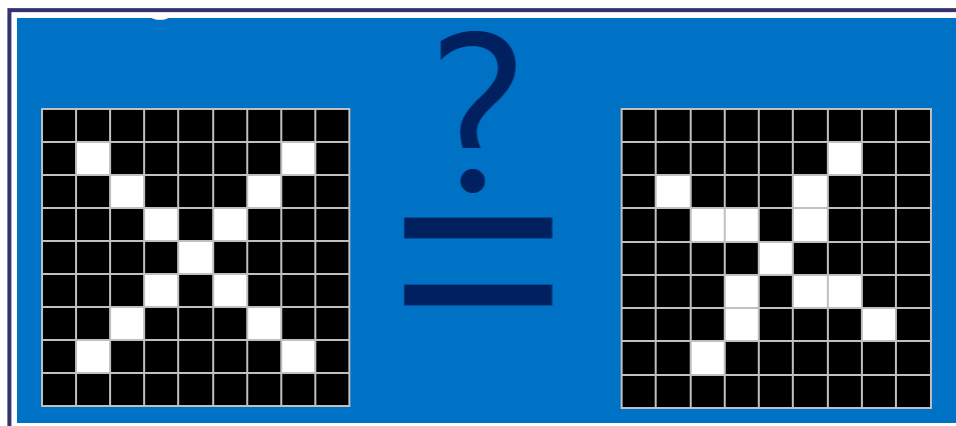


The input can be RST of the original:

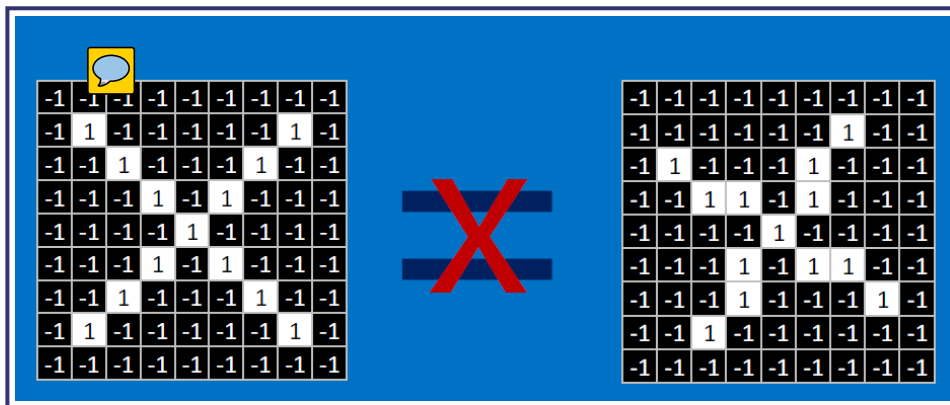
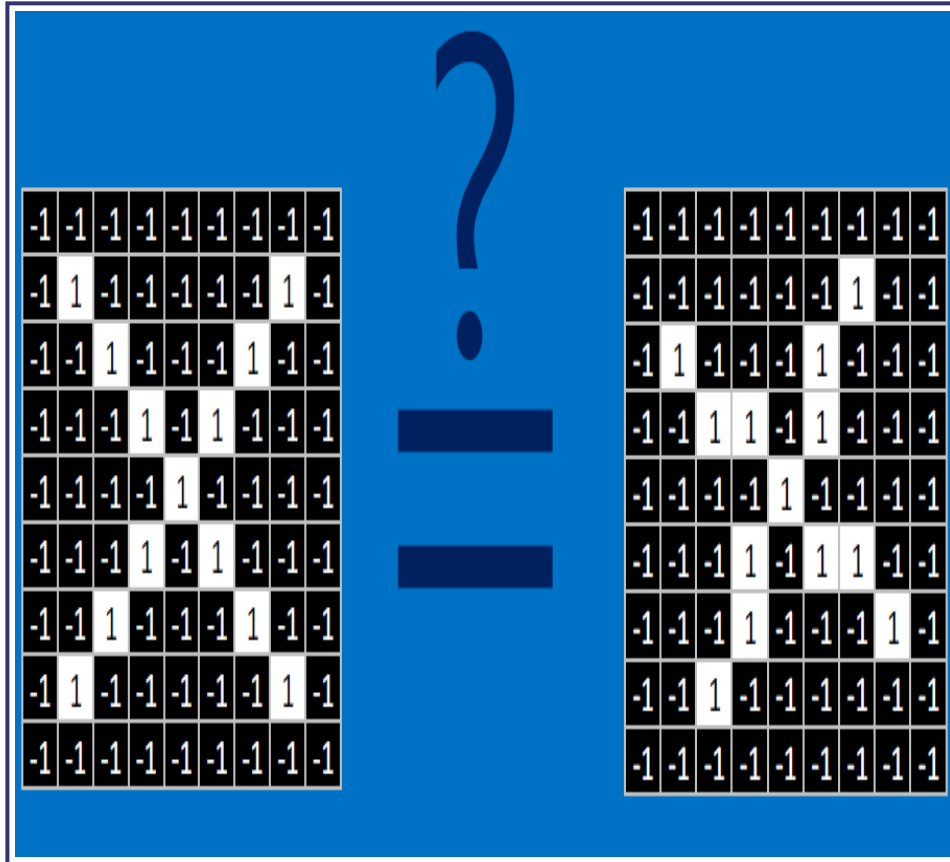




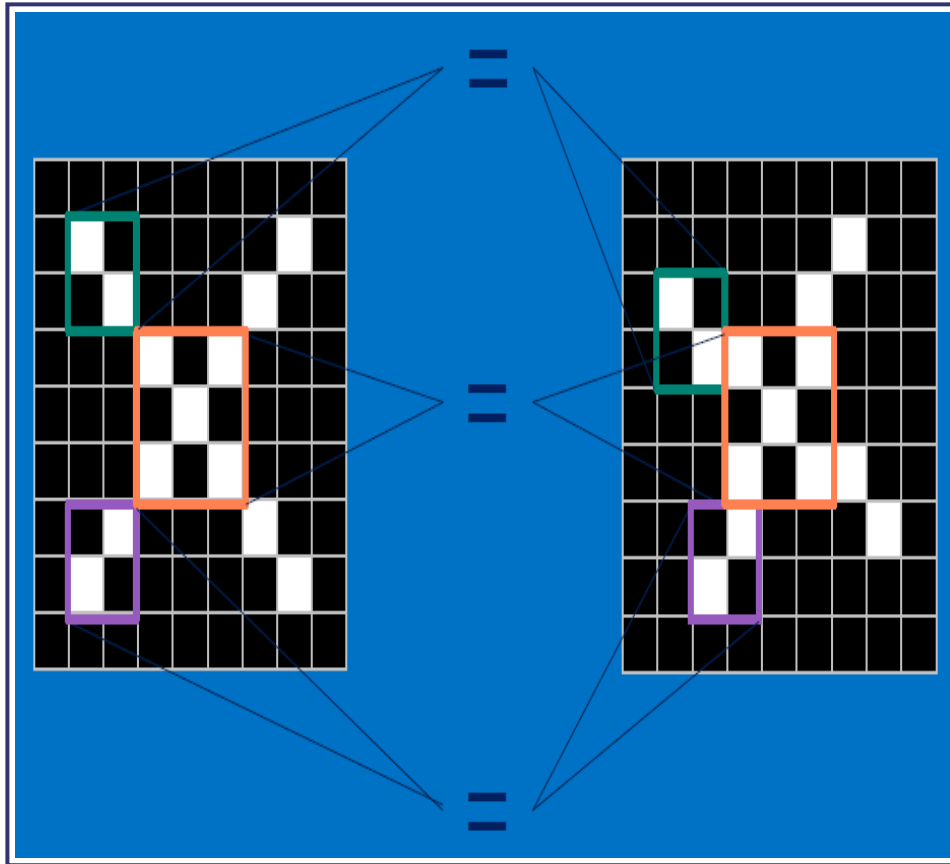
How can we compute similarity, but not LITERALLY (ie without pixel by pixel comparison)?



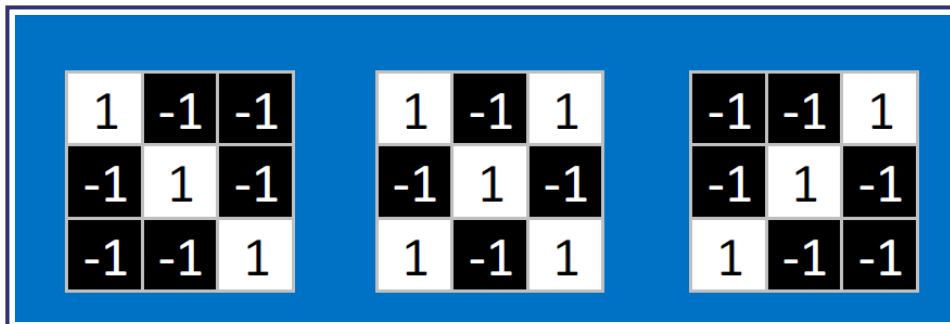
Useful pixels are 1, background pixels are -1:

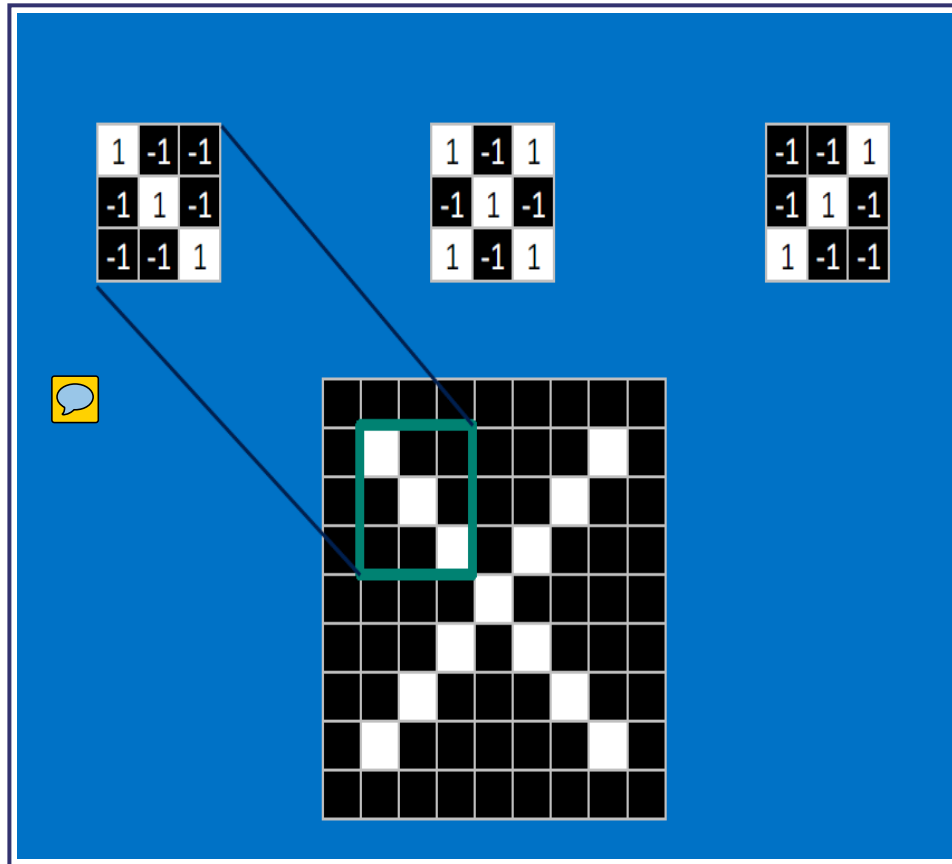


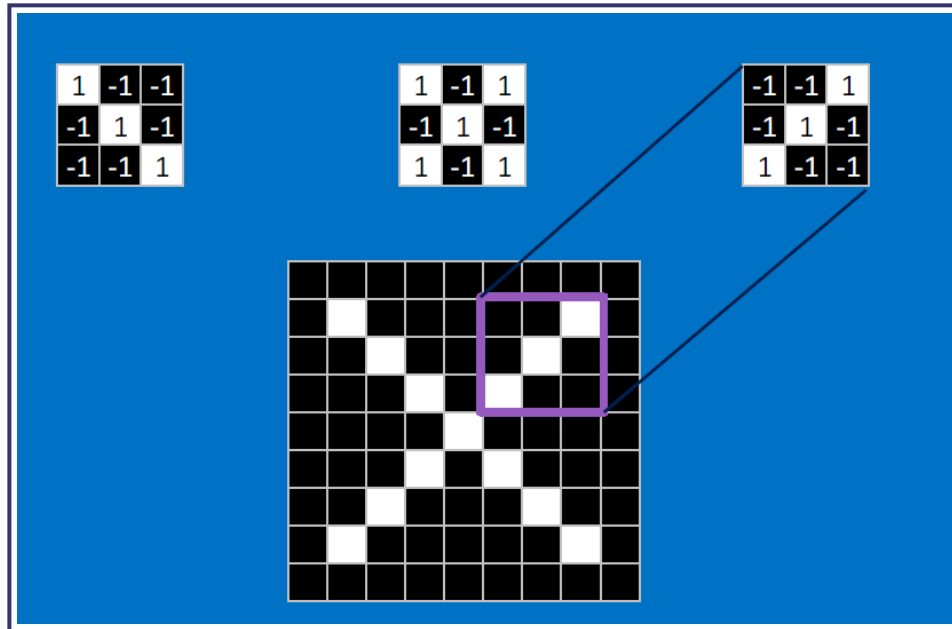
We match SUBREGIONS:

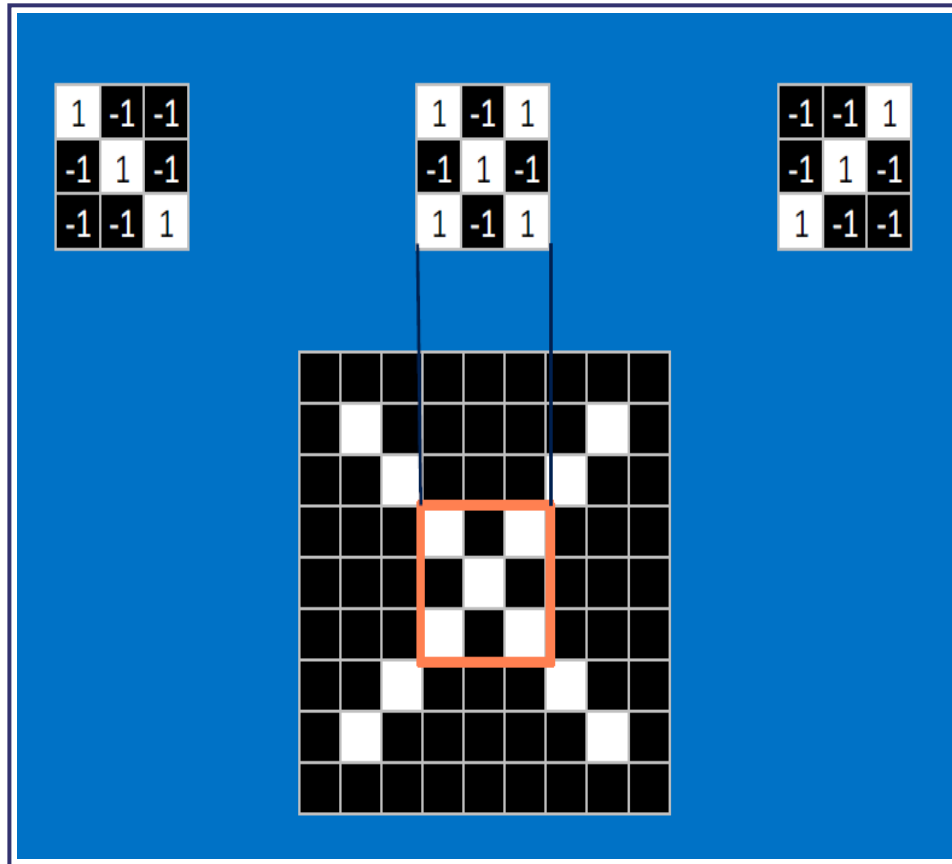


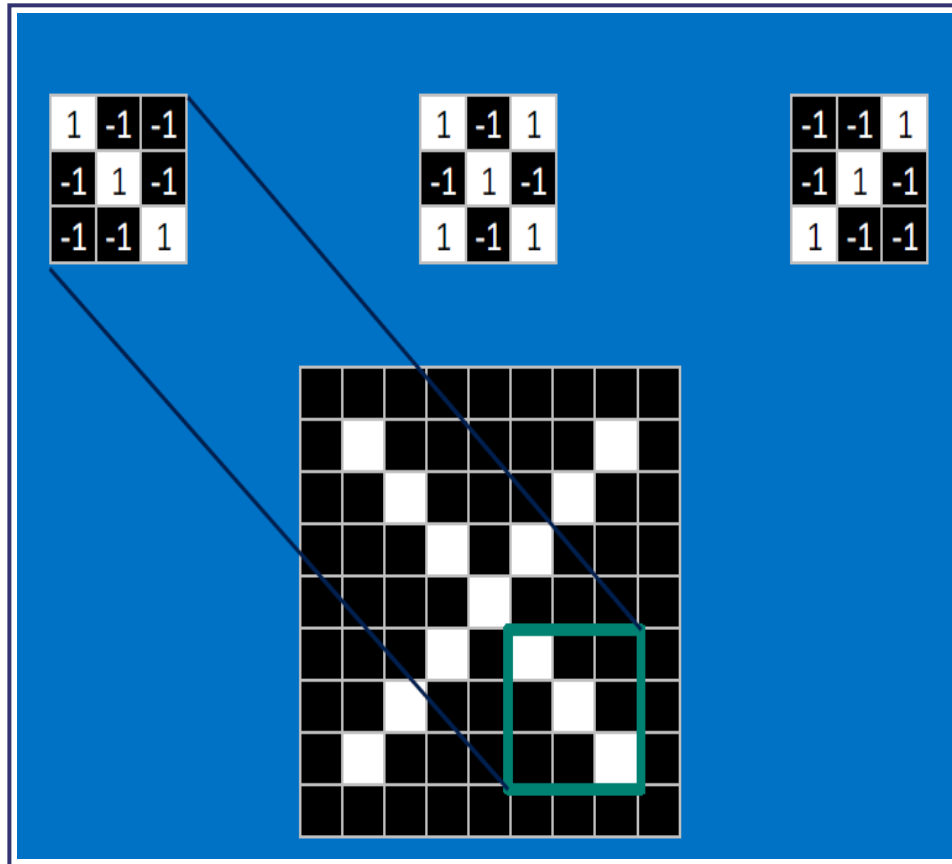
Convolutional neurons that check for these three features:

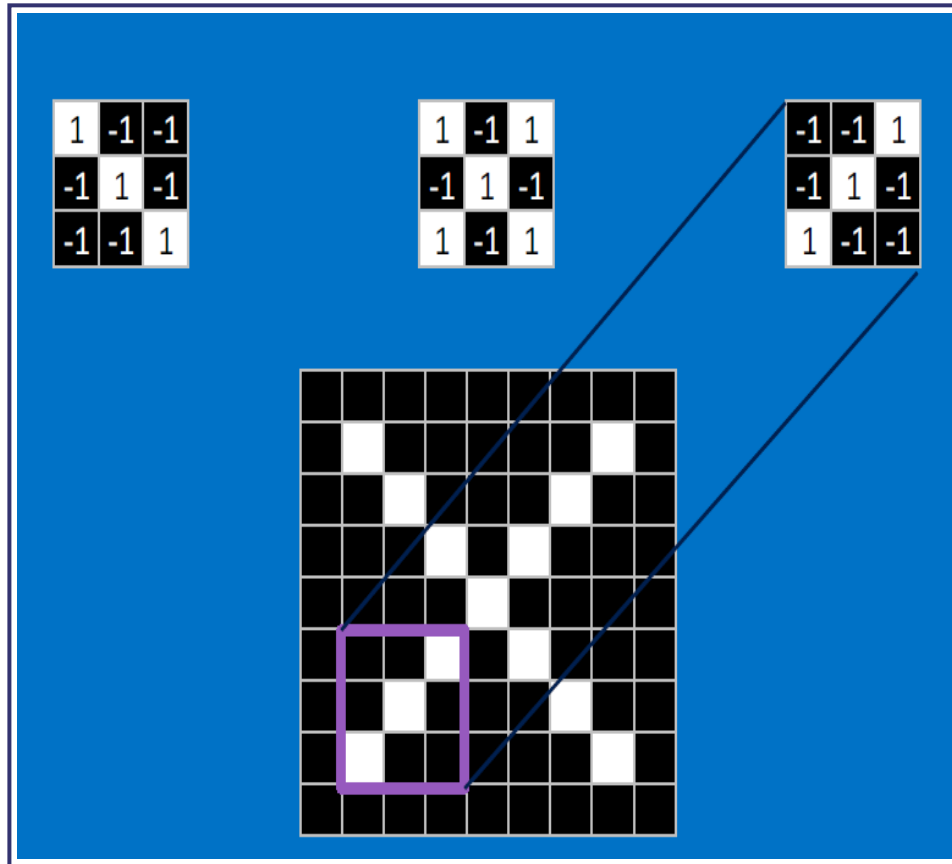






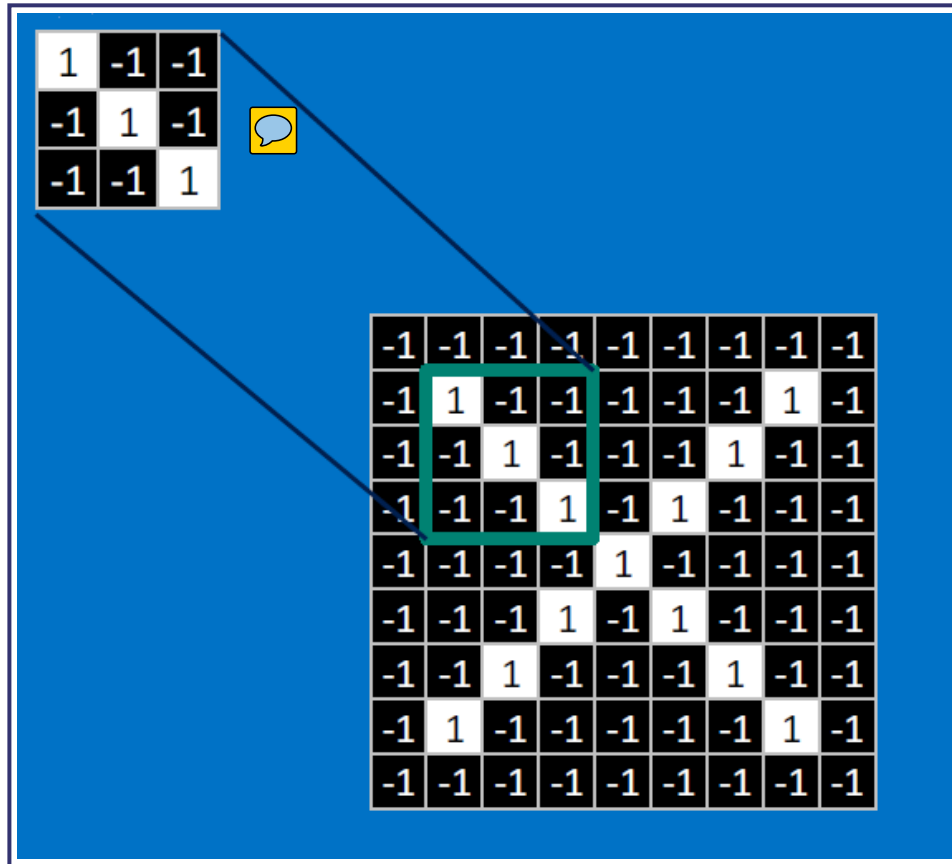


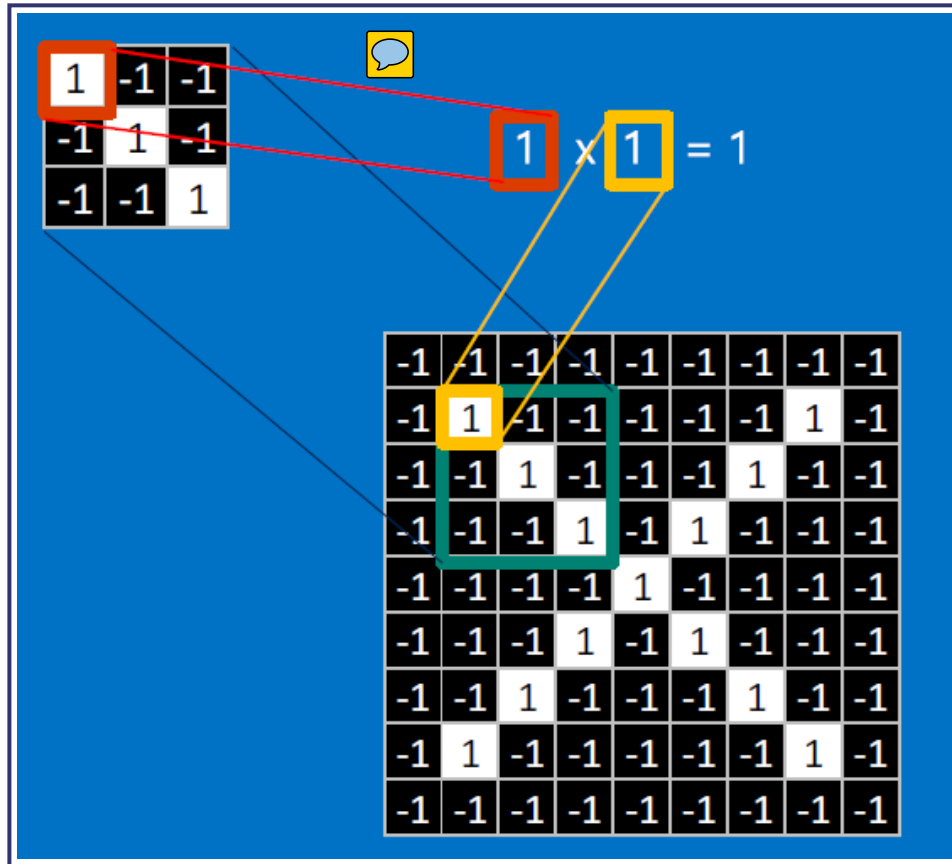


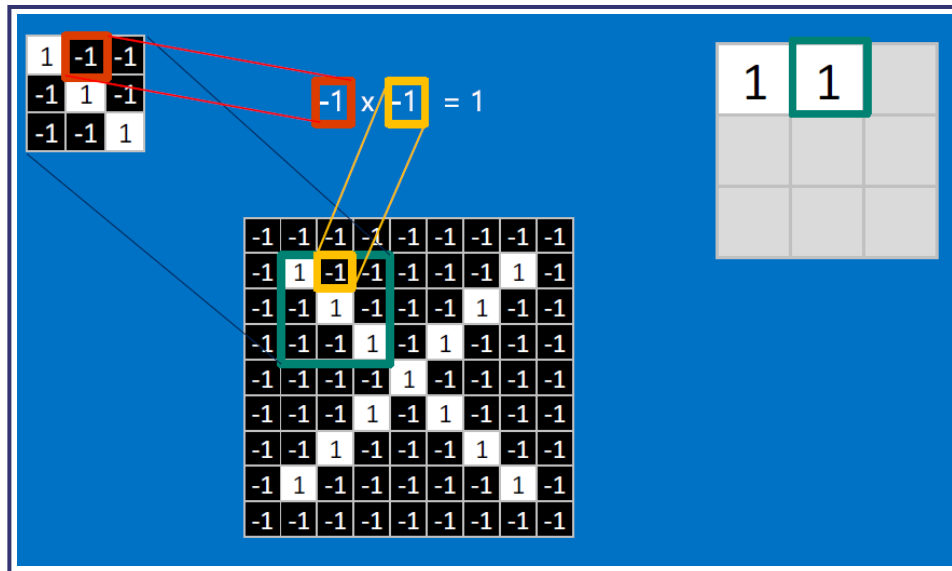
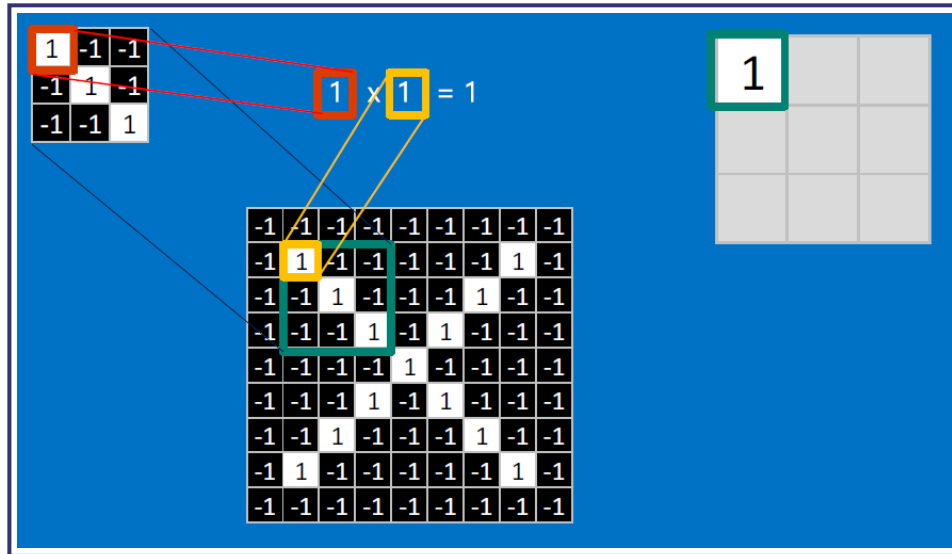


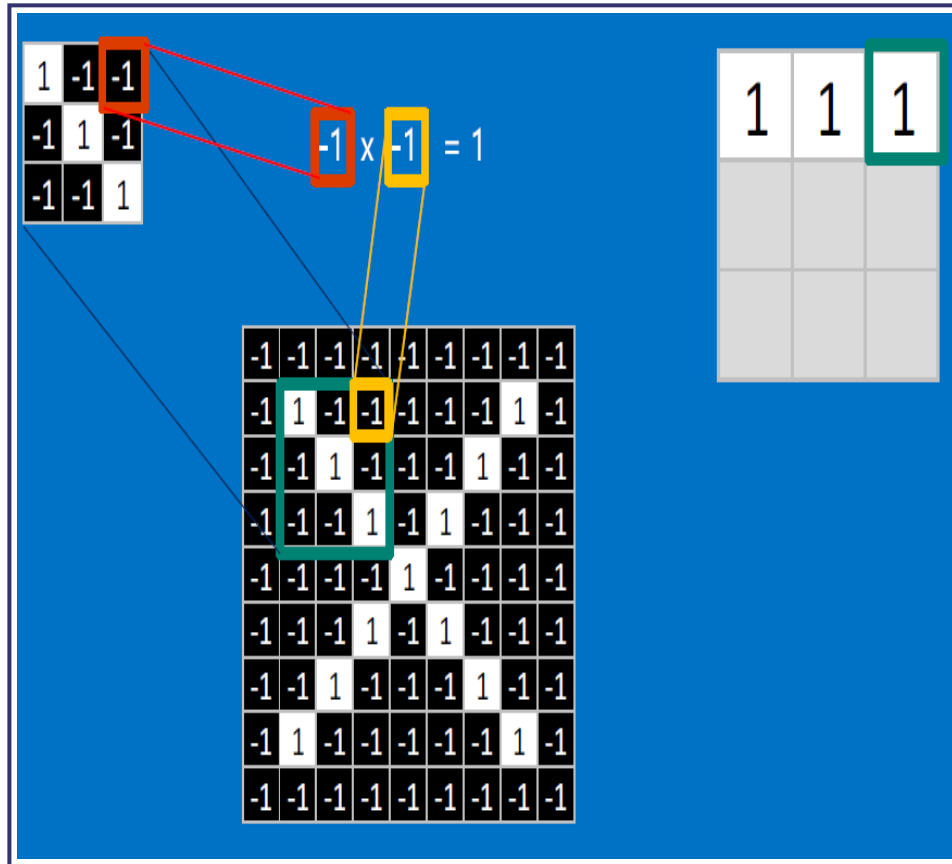
CONVOLVE, ie. do $x_i \cdot w_i$, then average, output a value:

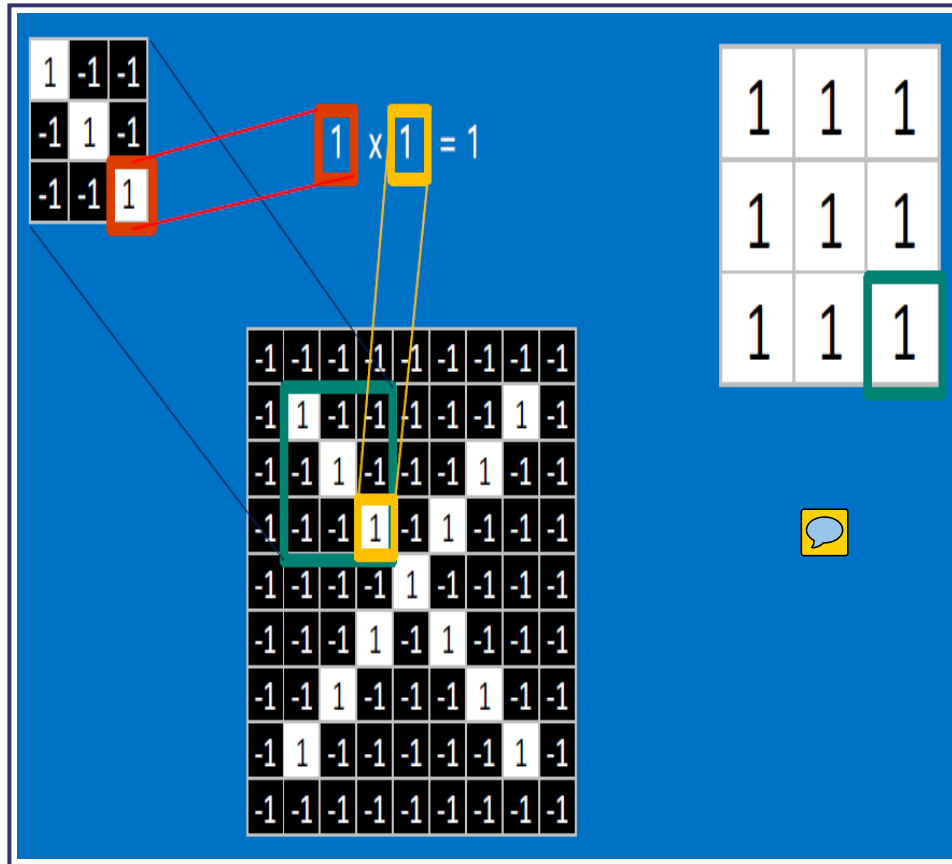


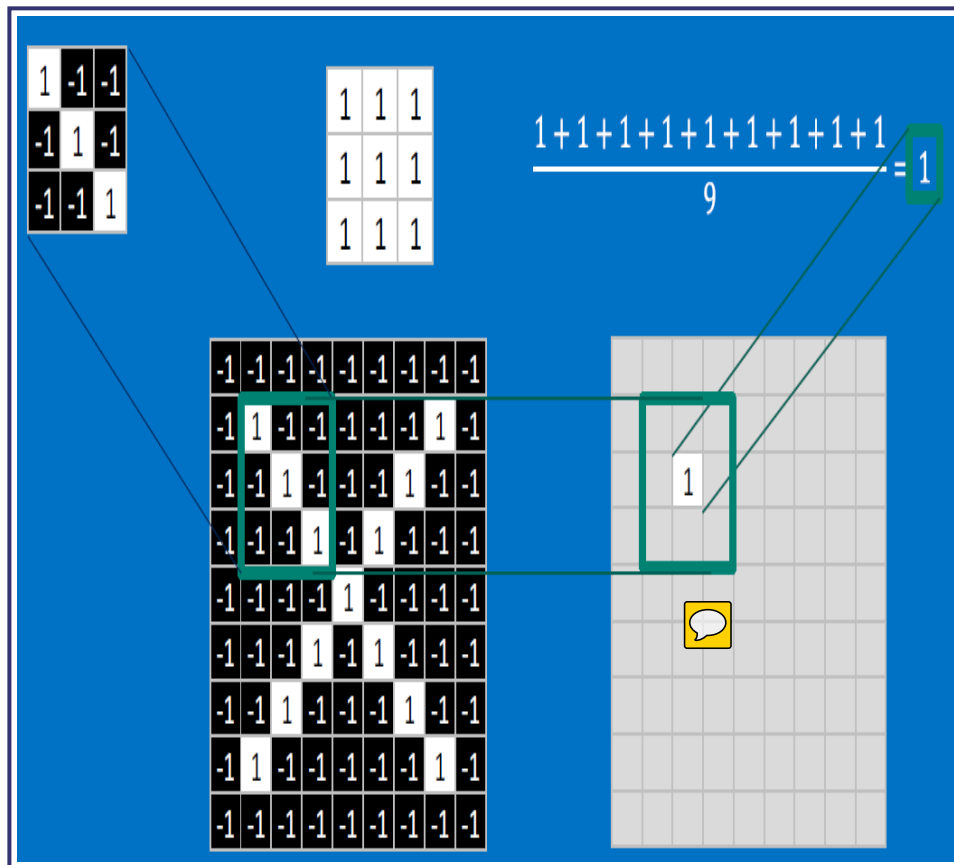




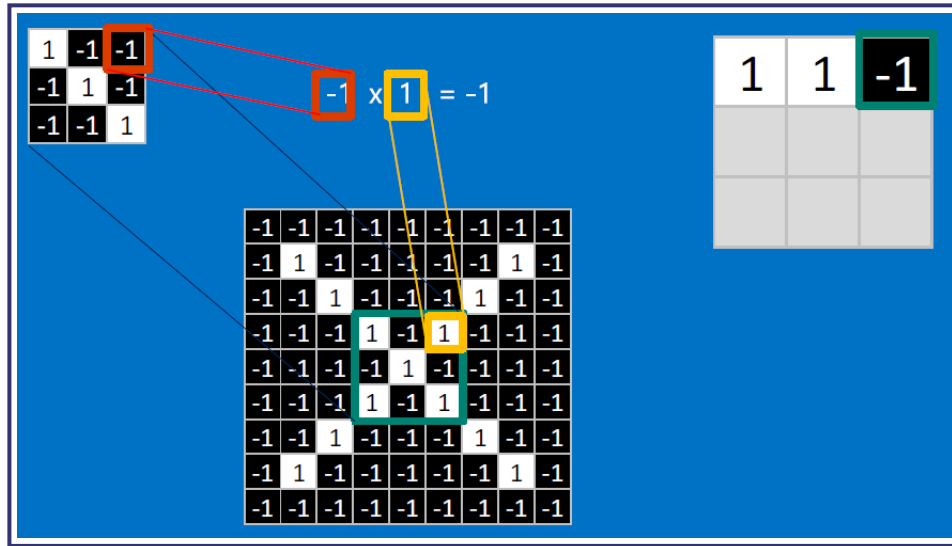


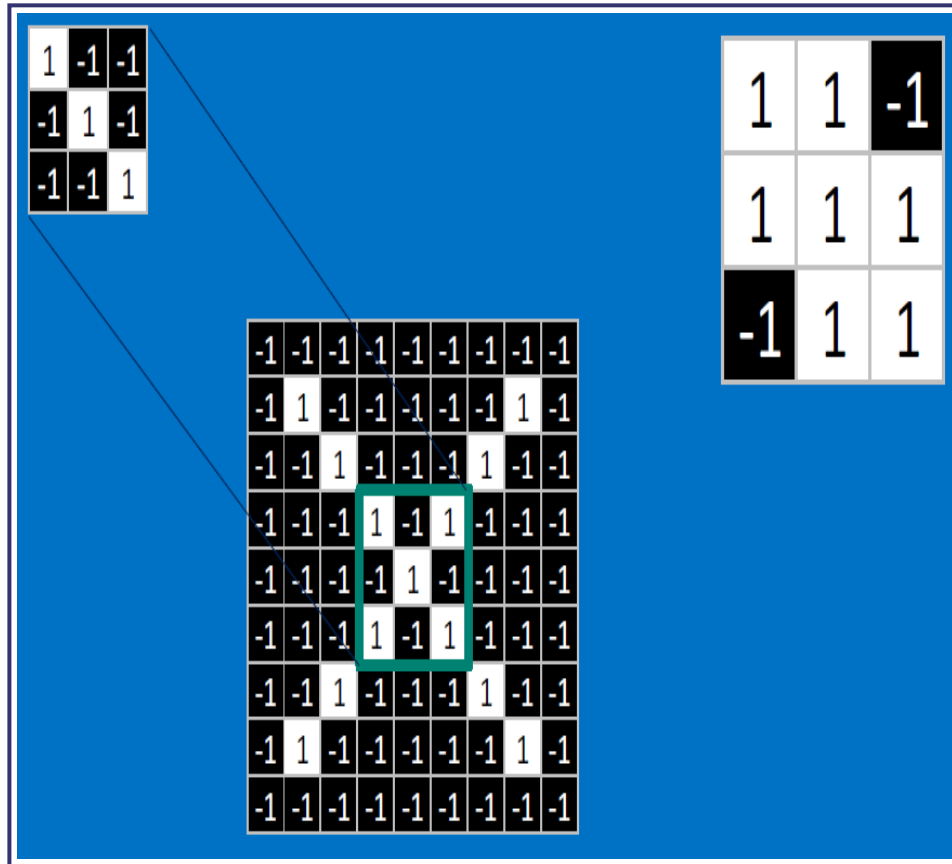


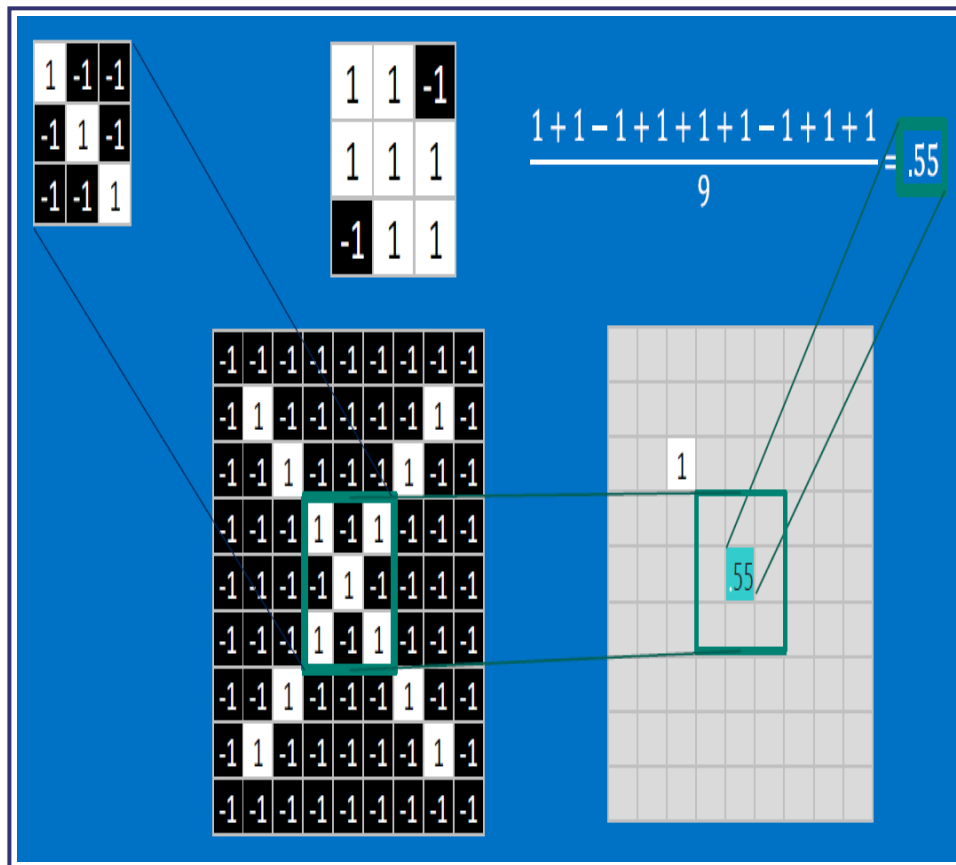




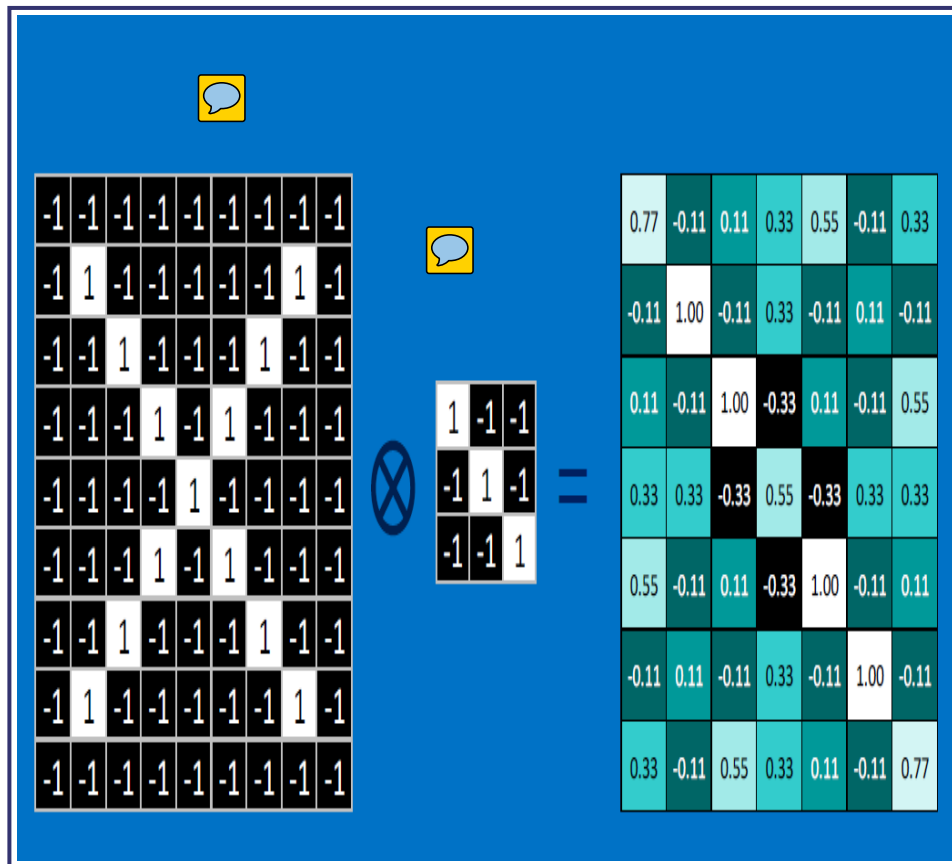
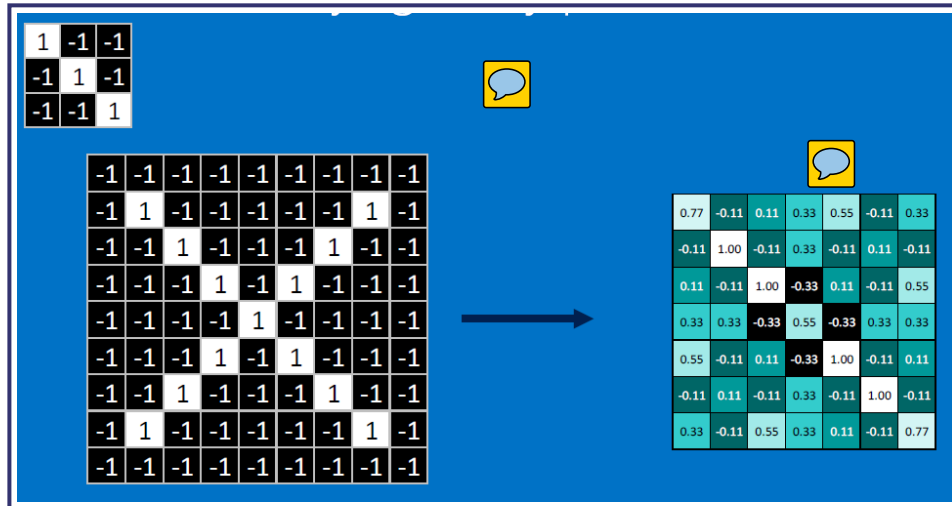
Need to **center the kernel at EVERY pixel** (except at the edges) and compute a value for that pixel!



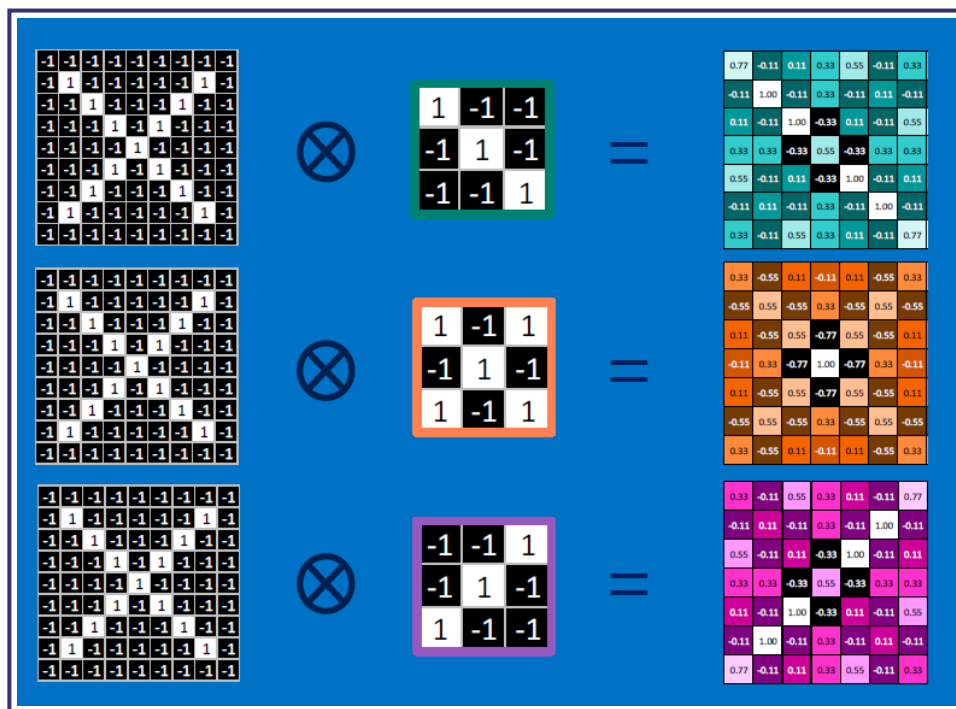




We end up with a 7x7 output grid, just for this (negative slope diagonal) feature:

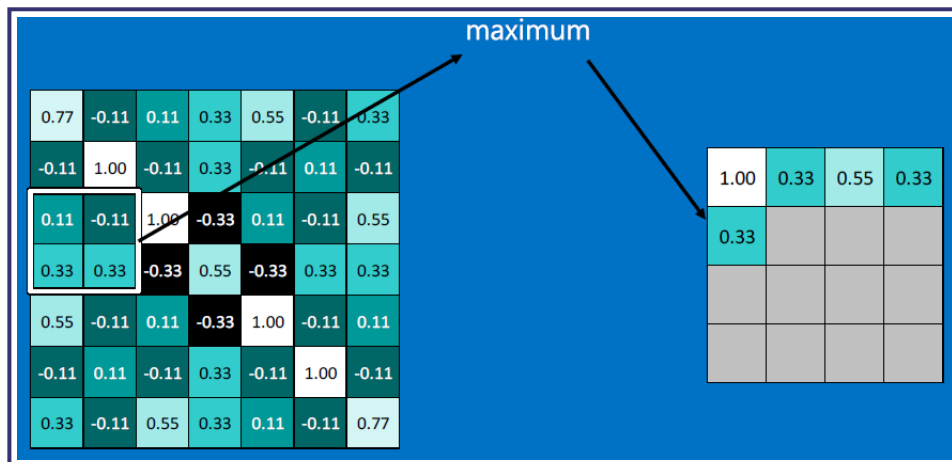
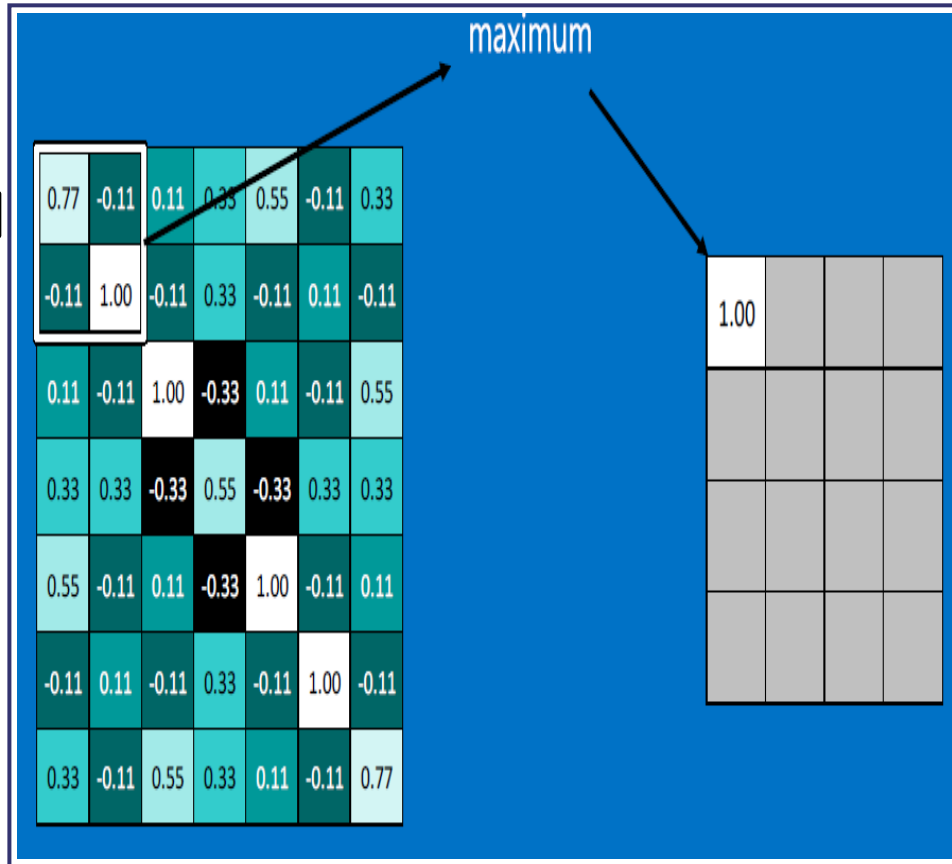


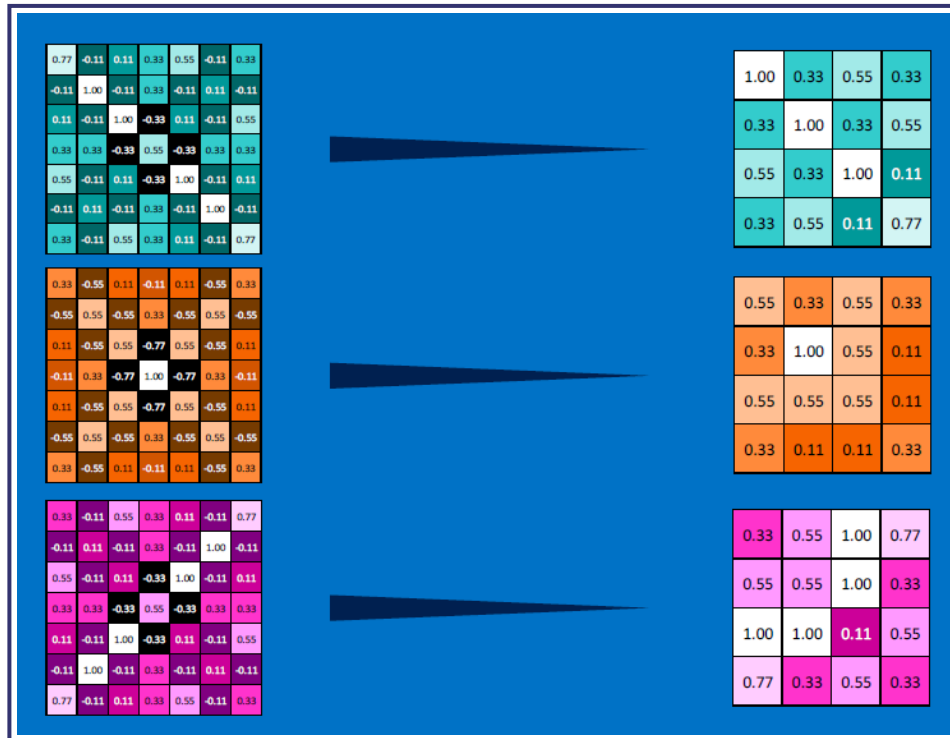
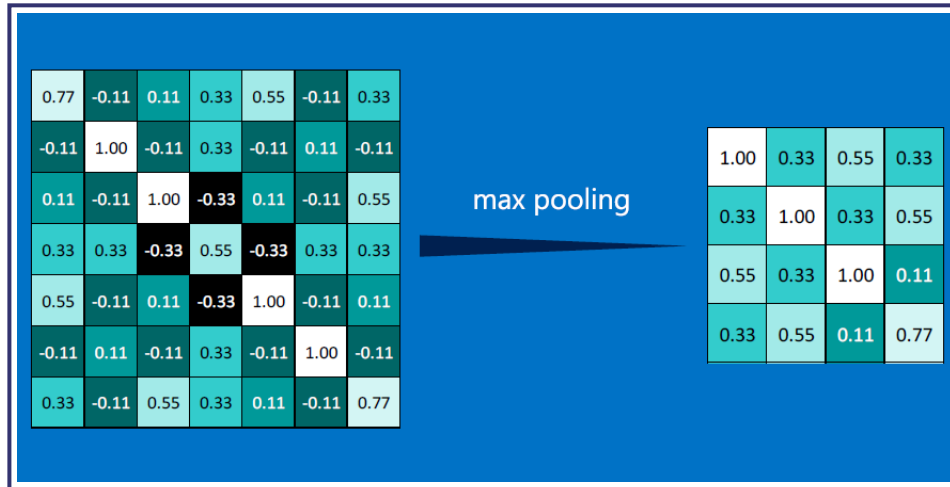
Each neuron (feature detector) produces an output - so a single input image produces a **STACK** of output images:

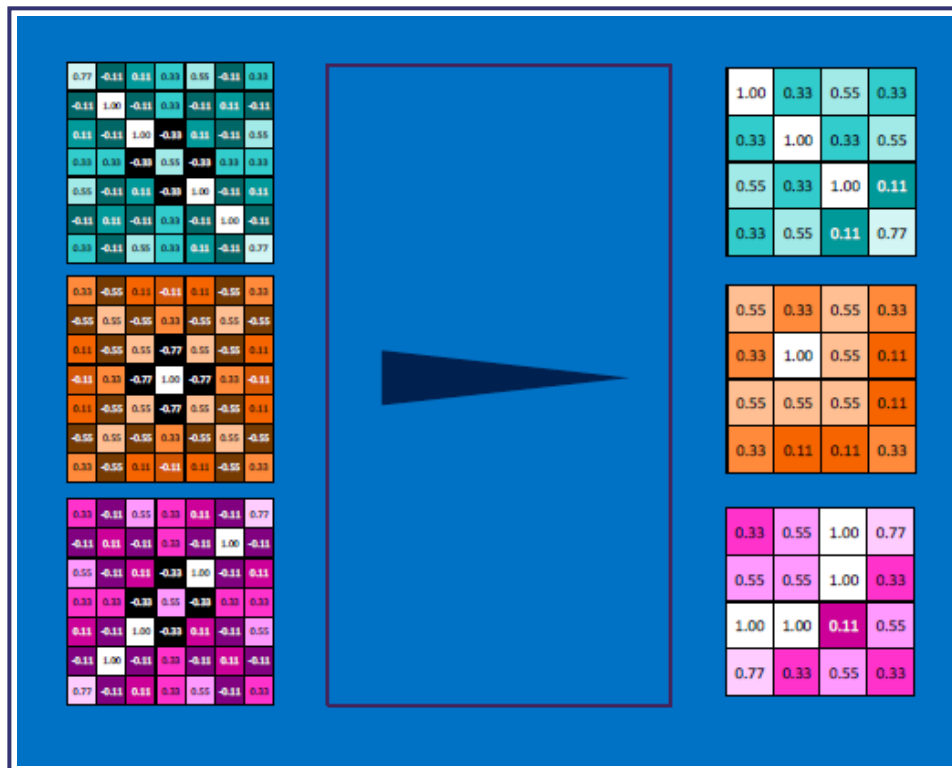




To collapse the outputs, we do **'max pooling'** - replace an $m \times n$ (eg. 2×2) neighborhood of pixels with a single value, the max of all the $m \times n$ pixels.

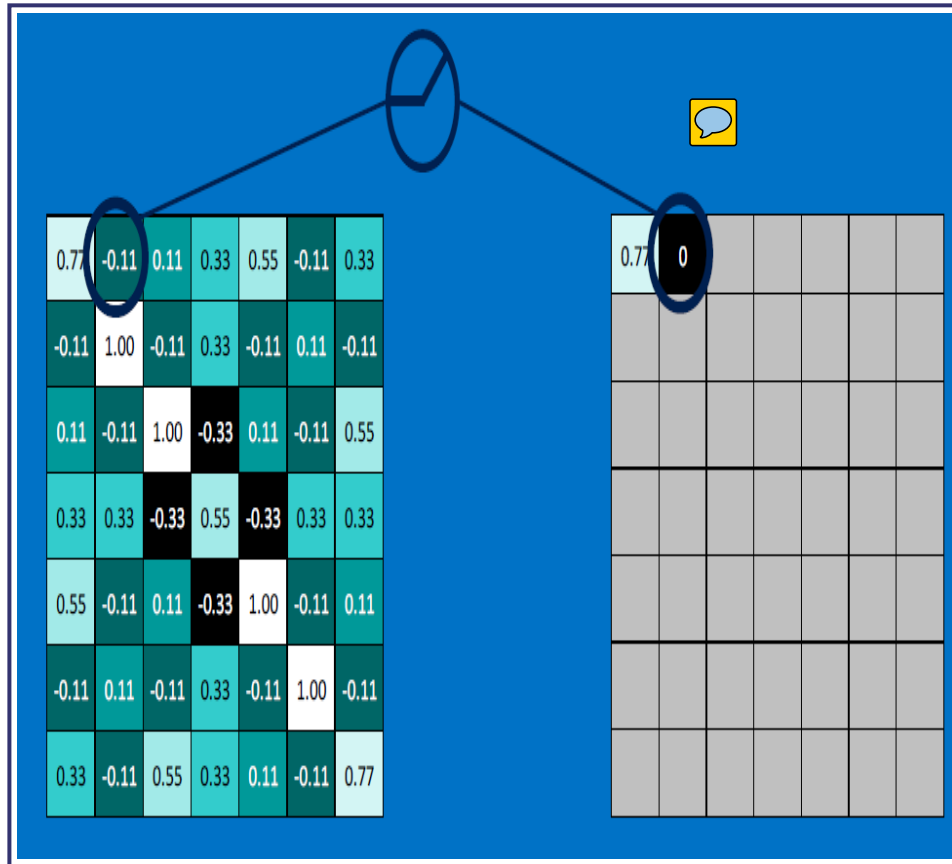


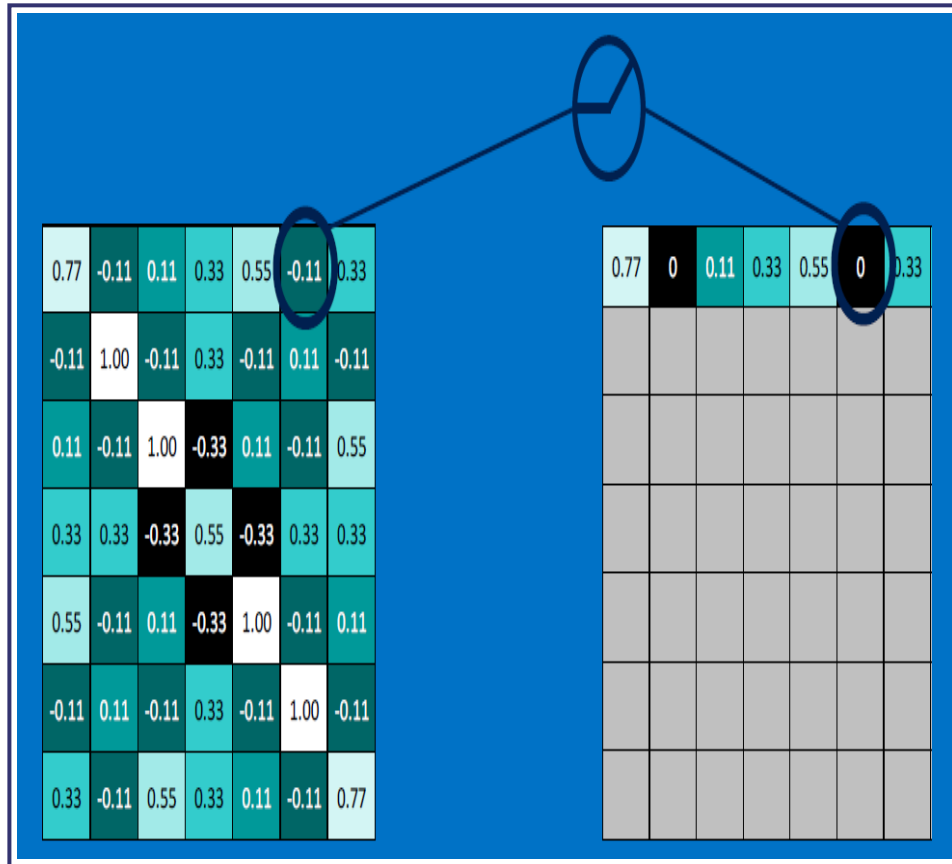


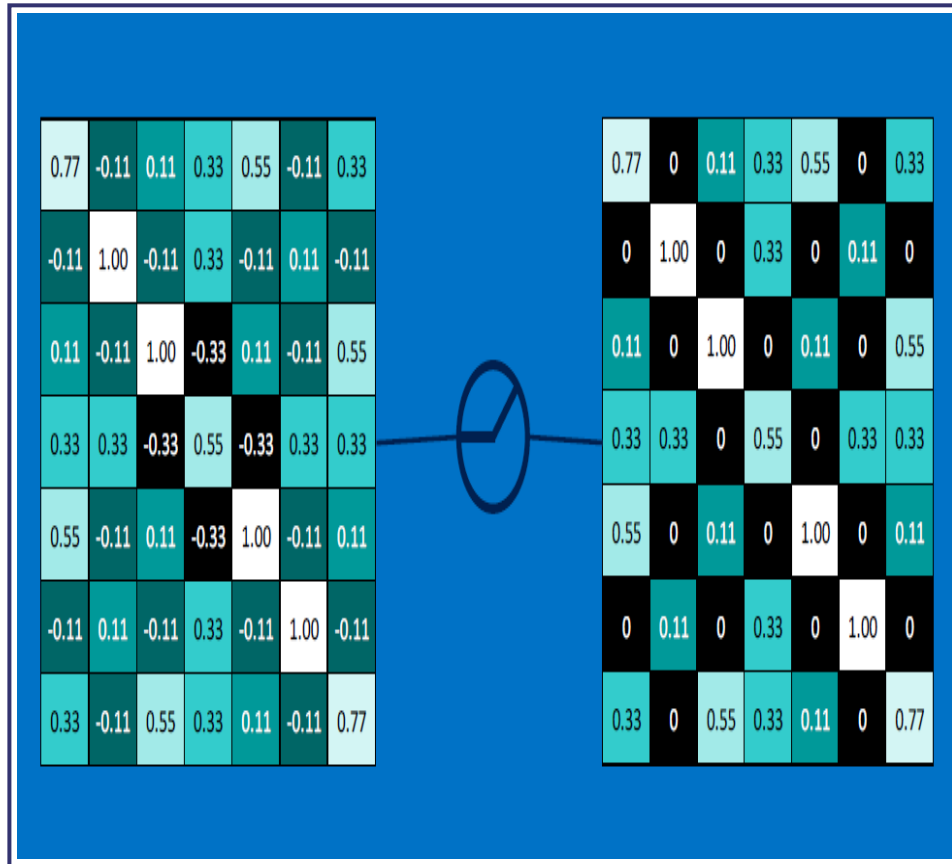


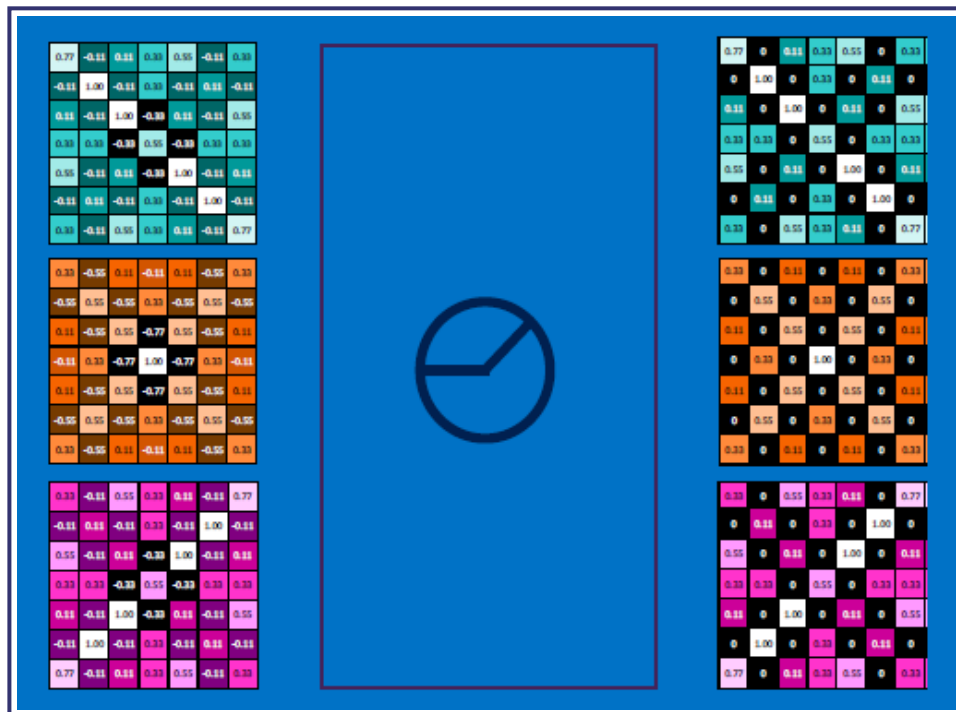
Next, create a ReLU - rectified linear unit - **replace negative values with 0s:**



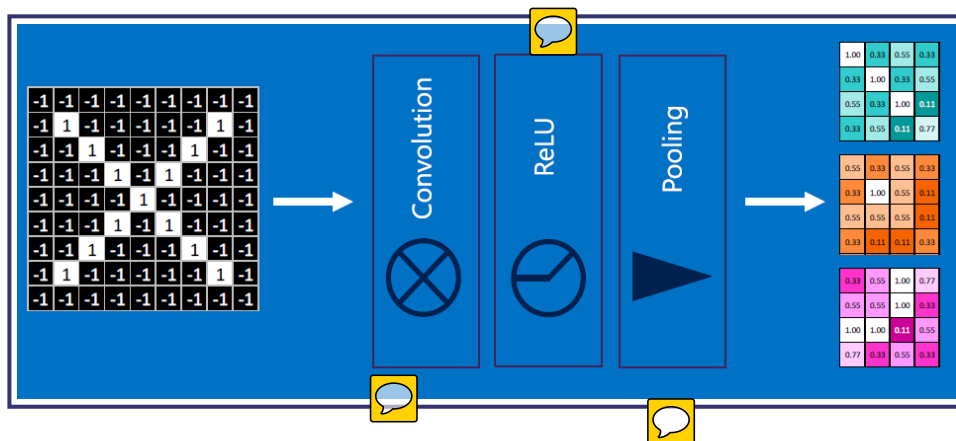




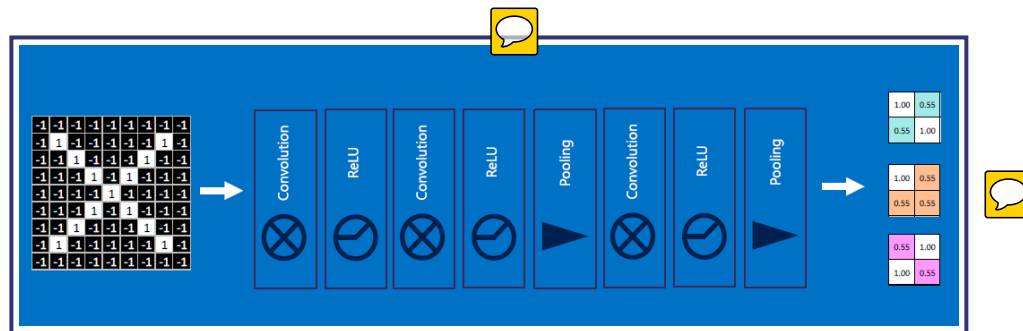




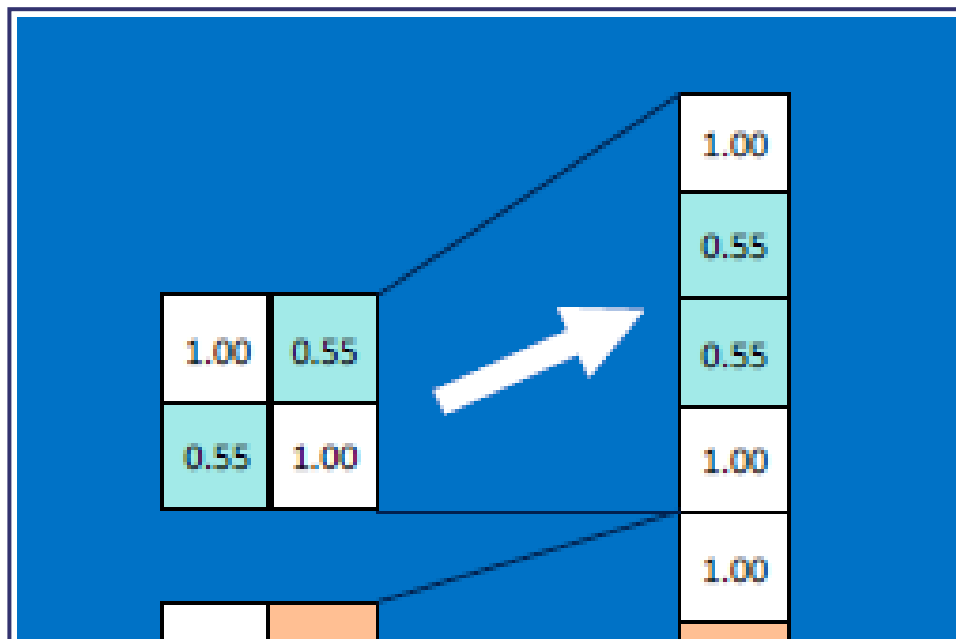
After a single stage of convolution, **ReLU**, pooling (or eqvt'ly, convolution, pooling, ReLU):

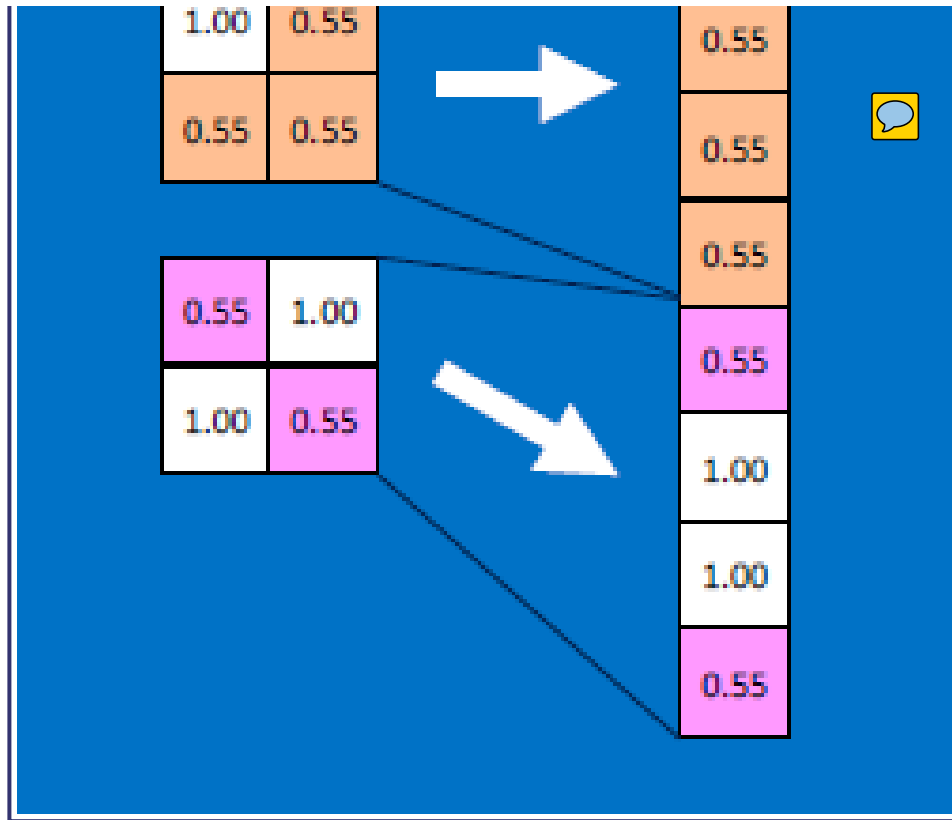


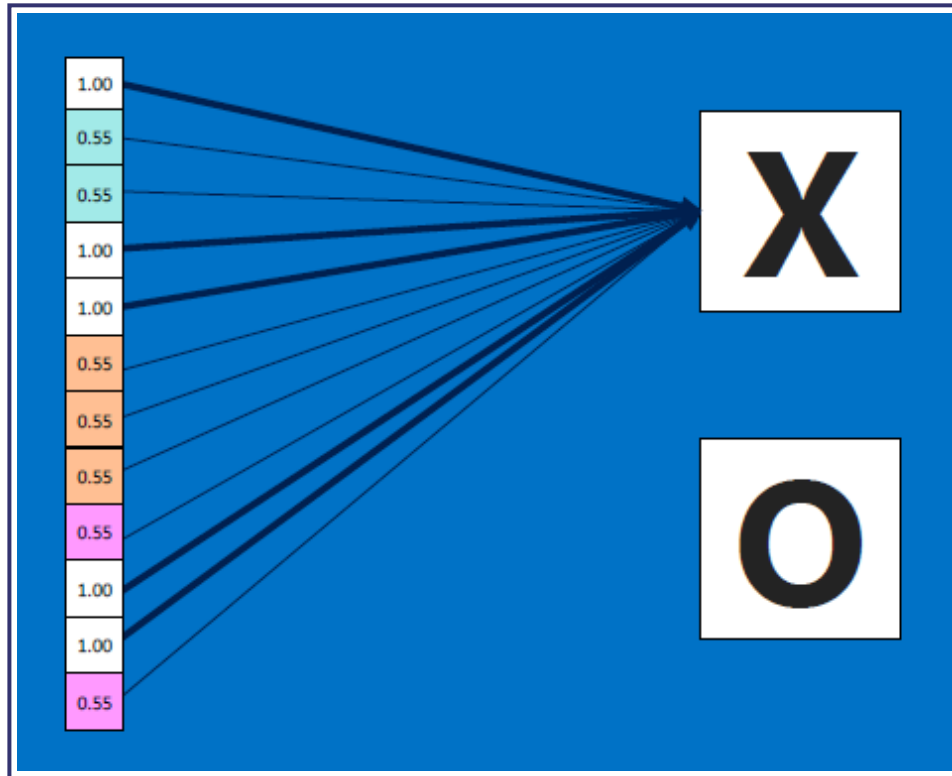
Usually there are multiple stages:

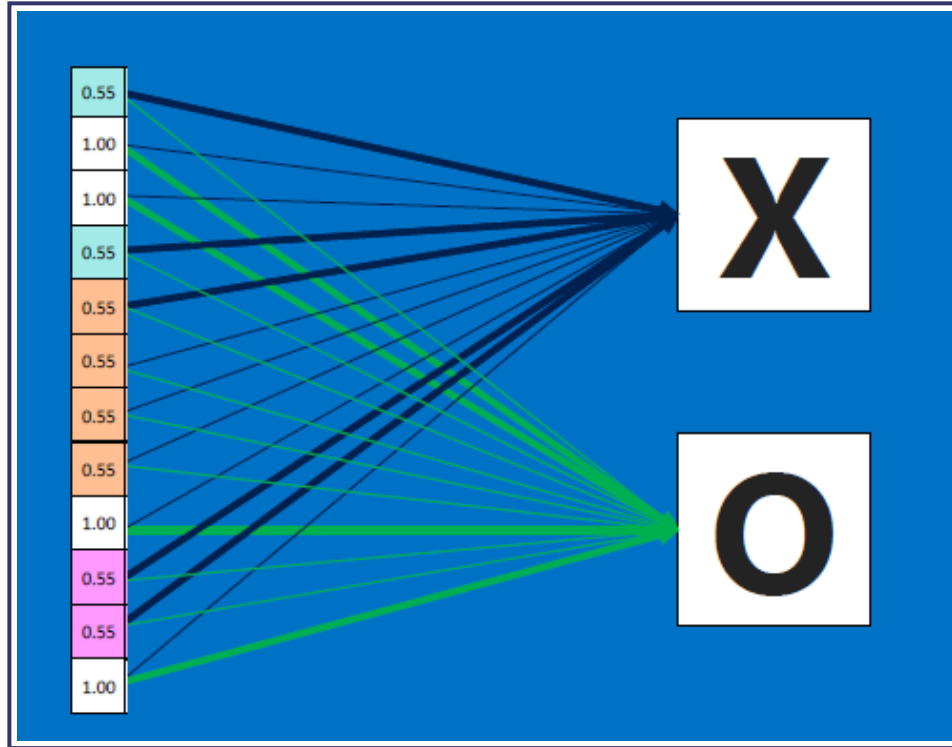


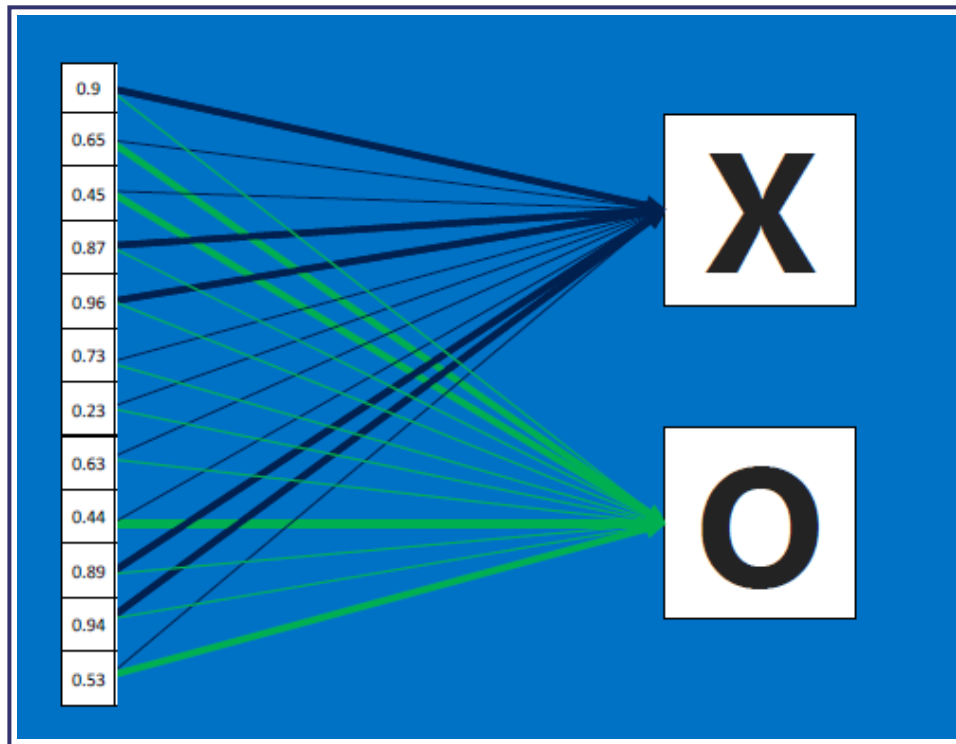
The resulting output values (12 in our case) are equivalent to VOTES: values at #0, #3, #4, #9, #10 contribute to voting for an 'X', and the rest, for 'O' (we'd train the network to detect an 0 as well):



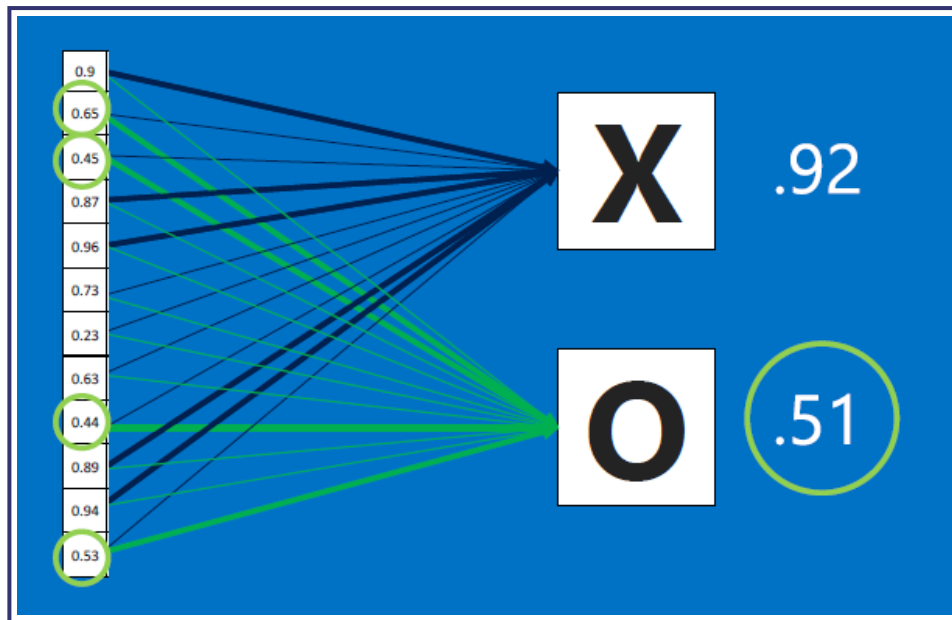
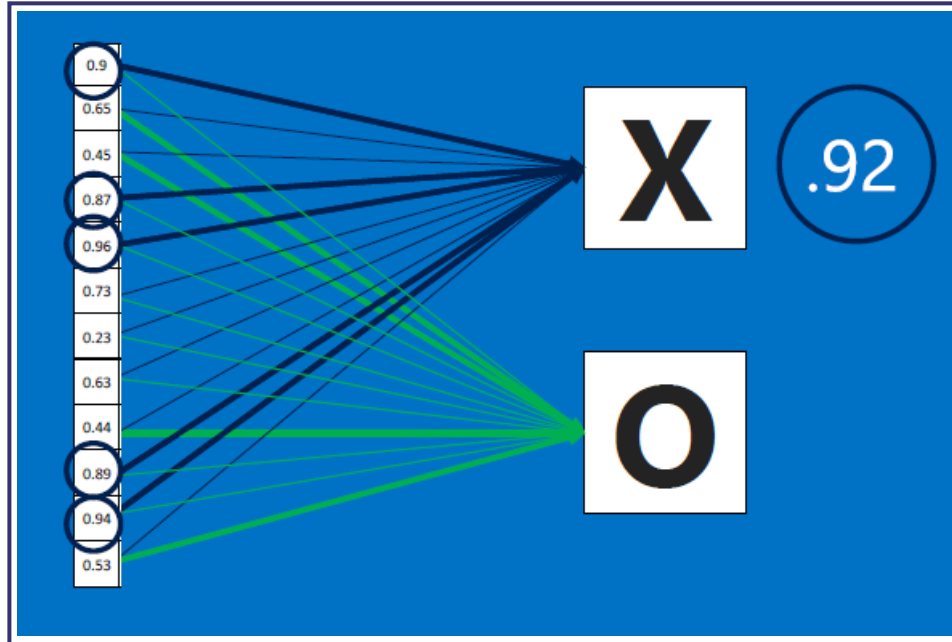


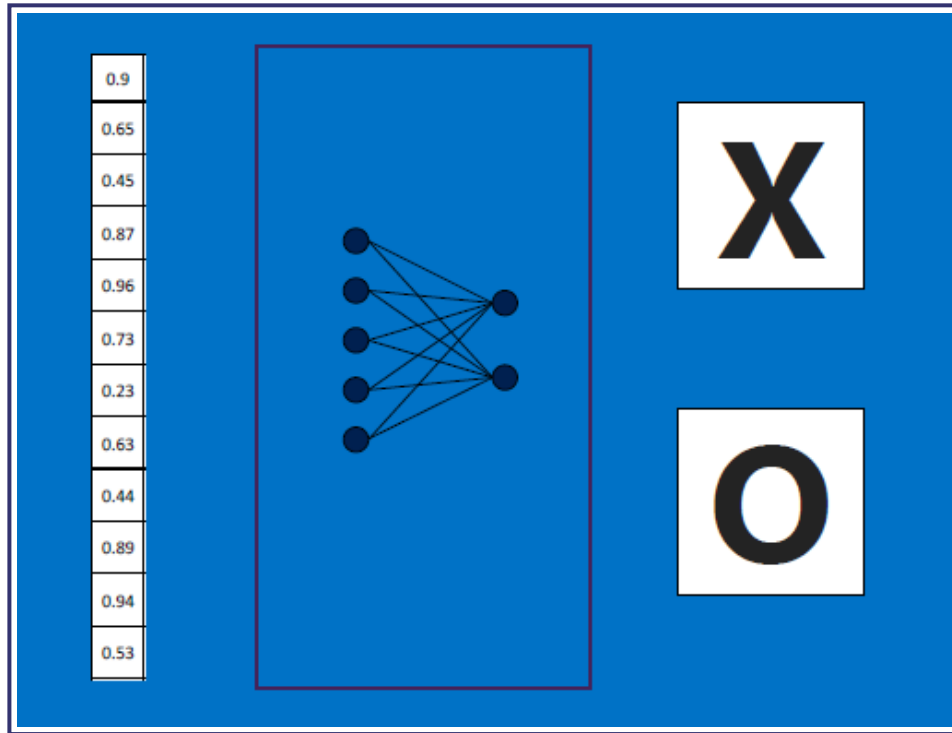






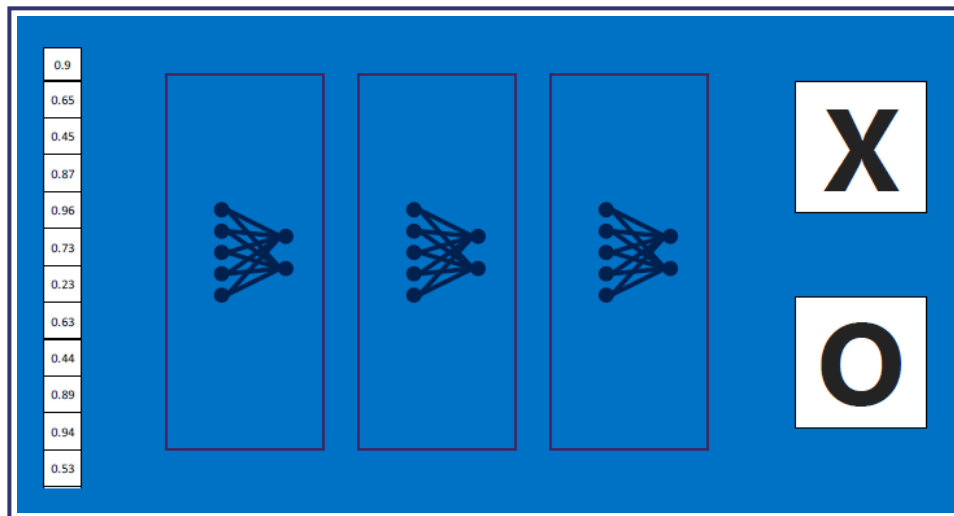
A real-world non-training input might produce this output - the results average to 0.91 for X, and 0.52 for O - the NN would classify this as an X:



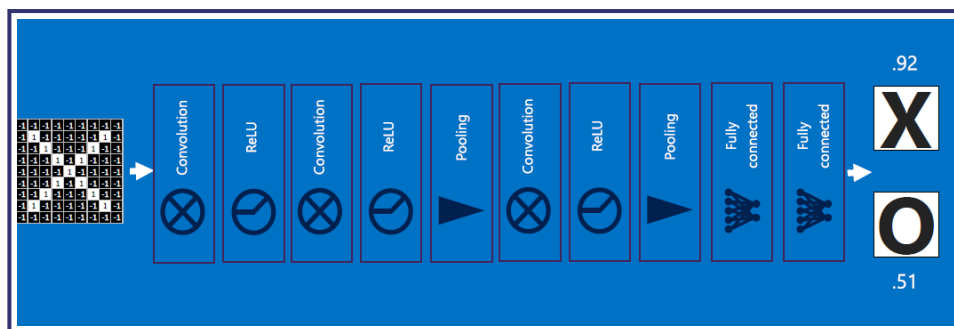


In real world situations, these voting outputs can also be cascaded:





'All together now':

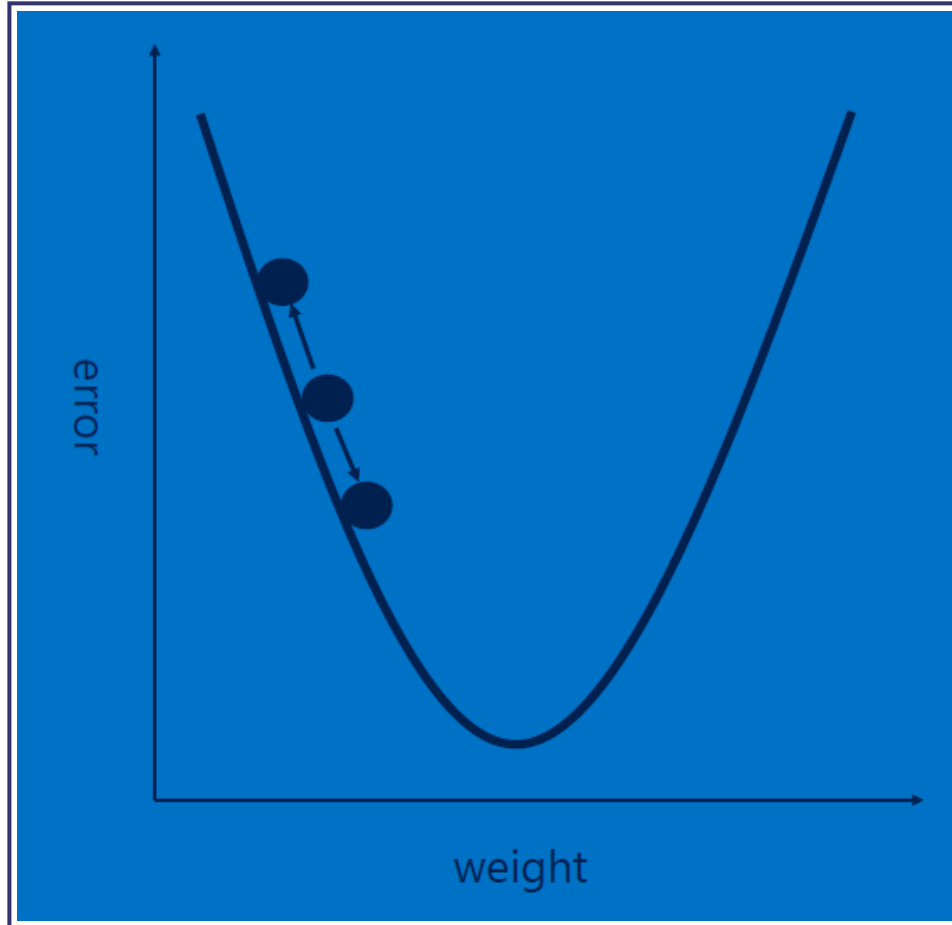


Errors are reduced via **backpropagation**. **Error** is computed by taking the absolute **differences'** sums between expected and observed outputs:

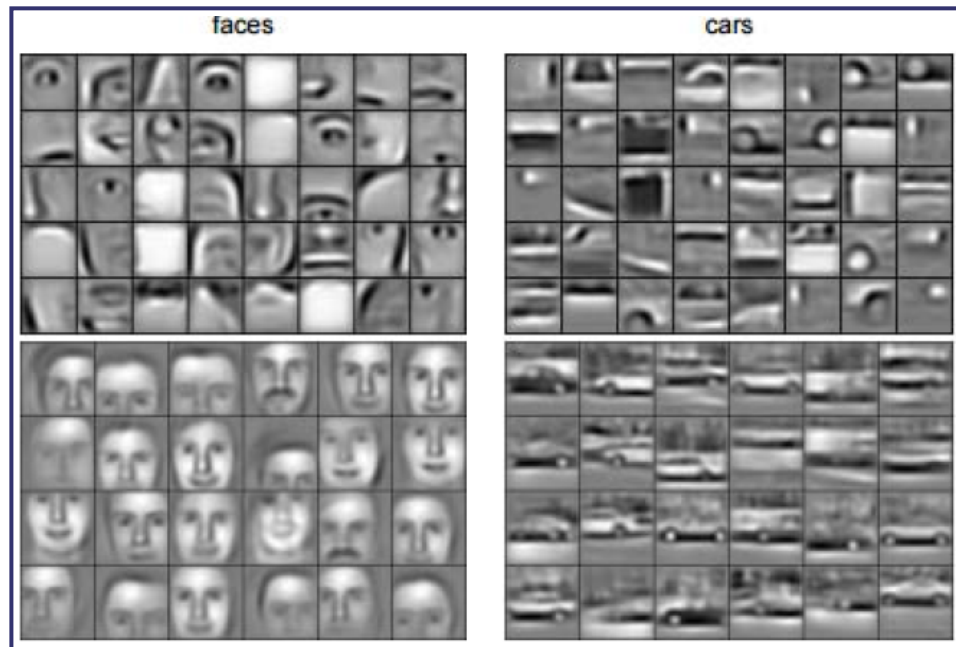




For each feature, each weight (one at a time) is adjusted slightly from its current value, with the goal of reducing the error (use the modified weights to classify, recompute error, modify weights, iterate till convergence) - this is backpropagation:



That was a whirlwind tour of the world of CNNs! Now you can start to understand how an NN can detect faces, cars...:



Want to 'learn' more? This 2009 [paper](#) has details.

When is a CNN ****not**** a good choice? Answer: when data is not spatially laid out, ie. scrambling rows and columns of the data would still keep the data intact (like in a relational table) but would totally throw off the convolutional neurons!



Finding out more about NNs

Look up papers/blogs by:

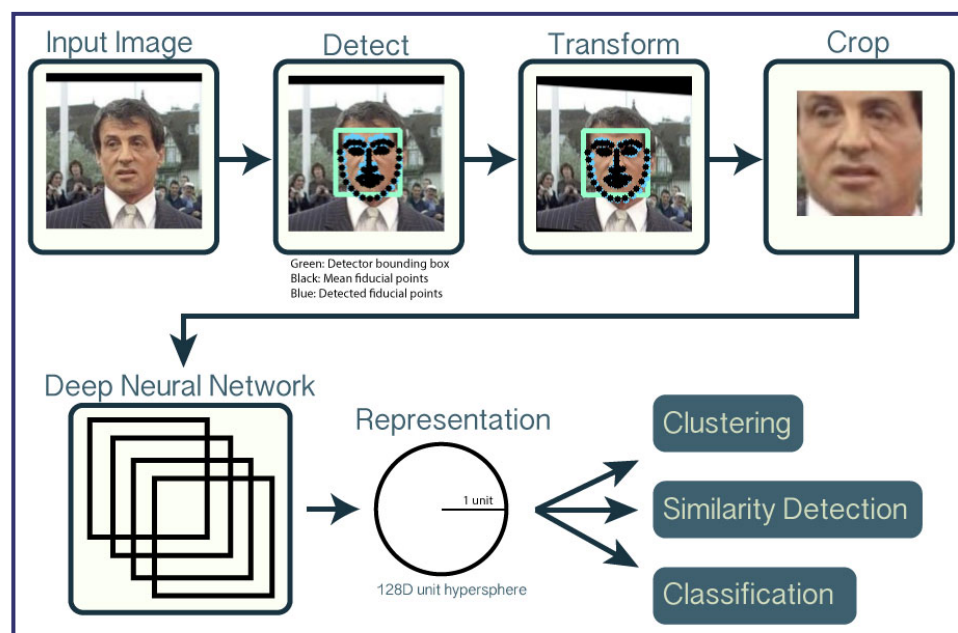
- Andrew Ng
- Yann LeCun
- Andrej Karpathy
- Chris Olah
- Brandon Rohrer

Also, [this](#) is a VERY comprehensive portal on DNN.

NN libraries

As you can imagine, there is a wide variety of libraries available.

OpenFace is specifically for face detection:



The following are general purpose ML frameworks:

- CAFFE
- CNTK

- **Deeplearning4j**
- **Keras**
- **TensorFlow**
- **Theano**
- **Torch**

NN in hardware

The following are GPU-based NN implementations:

- **NVIDIA**
- **Fujitsu**
- **Inspur**

In addition, Amazon's AWS now includes **cloud-GPU** computing!

And IBM's **TrueNorth** is a neural-architecture-based chip, meant for general purpose computation.

What is **old** is new again :)

Baby X

Baby X is an NN-driven **brain simulation!**

Can deep learning reach brain-computing levels?

NOT by a long shot, says this researcher (Informatics student).

More to look up

- RNNs
- adversarial learning
- Incep'ism
- Google Draw: <https://quickdraw.withgoogle.com/#>
- DL portrait morph: <https://www.youtube.com/watch?v=-oLemNTo7YU>