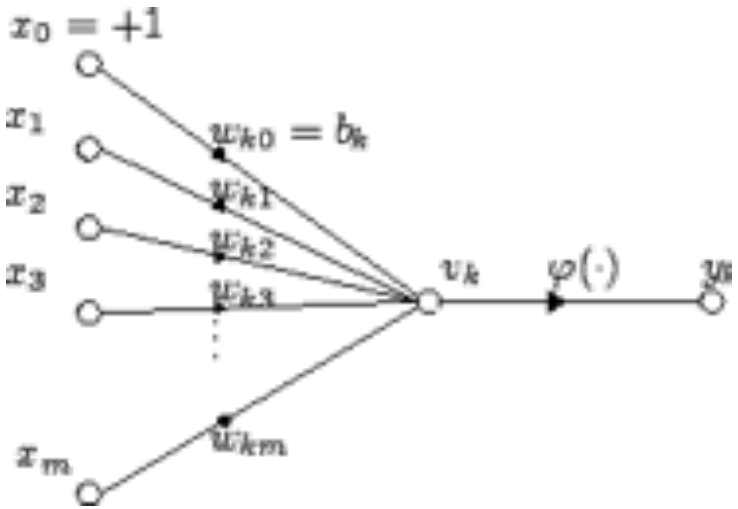
UNIVERSITÀ DEGLI STUDI DI SALERNO
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA
BIP LAB
Carmen Bisogni, PhD Student

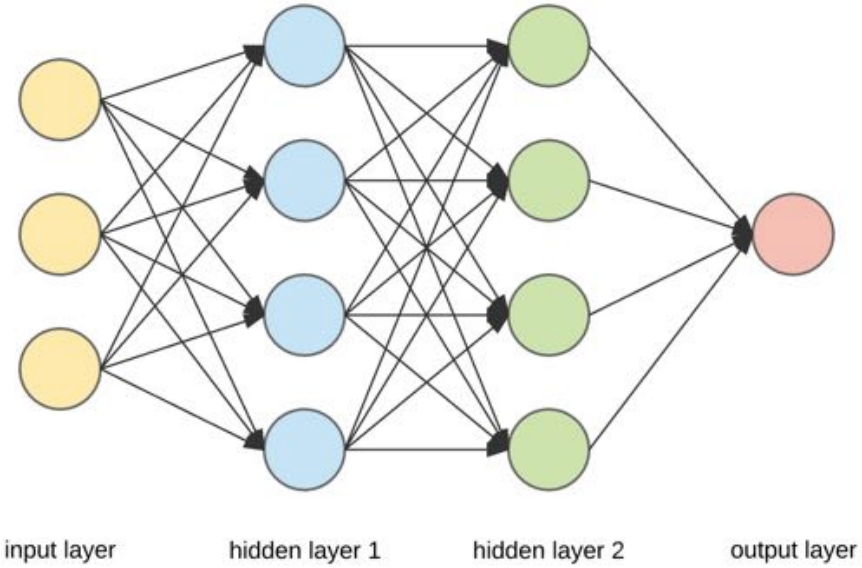
C.A.S.A. Course

CONTEXT AWARE SECURITY ANALYTICS IN COMPUTER VISION
Lesson 5

NN Theory

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

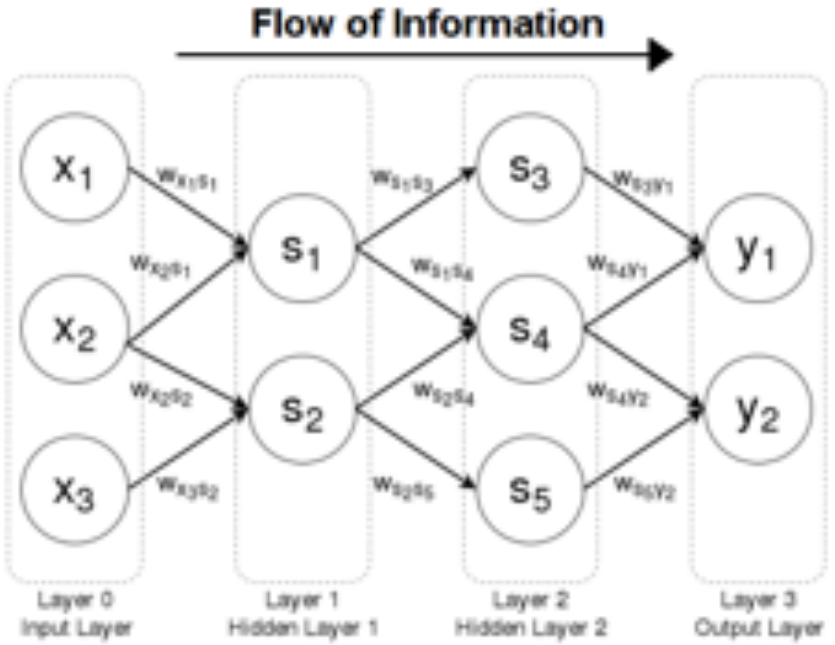




How the Neural Network predicts?

NN Theory: Feedforward

In a multilayer feedforward ANN, the neurons are ordered in layers, starting with an input layer and ending with an output layer. Between these two layers are a number of hidden layers. Connections in these kinds of network only go forward from one layer to the next.

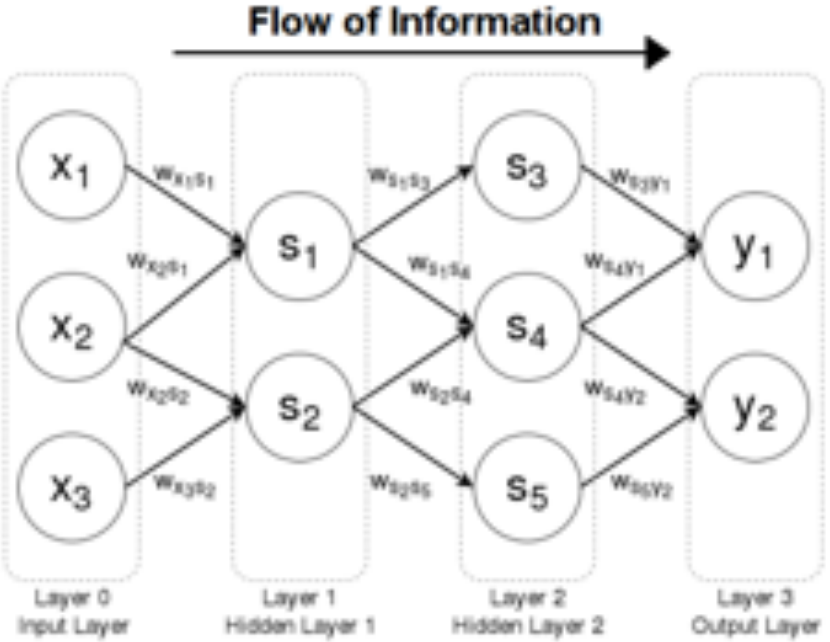


NN Theory: Feedforward

Step in a feedforward network:

- An input is presented to the input layer
- The input is propagated through all the layers
- The input reaches the output layer

In feedforward network there are no loop!



NN Theory:
Feedforward details

This process of Forward propagation is actually getting the Neural Network output value based on a given input. This algorithm is used to calculate the cost value.
As we already discuss, we have this connection between neurons:

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

After we got the h(x) value (hypothesis) we use the Cost function equation to calculate the cost for the given set of inputs.
θ are the weights.
Here we can notice how forward propagation works and how a Neural Network generates the predictions.

$$x = a^{(1)}$$

$$z^{(j+1)} = \theta^{(j)} a^{(j)}$$

$$a^{(j+1)} = \sigma(z^{(j+1)})$$

$$h_\theta(x) = a^{(L)} = \sigma(z^{(L)})$$




Is it sufficient?

NN Theory:
Feedforward direction

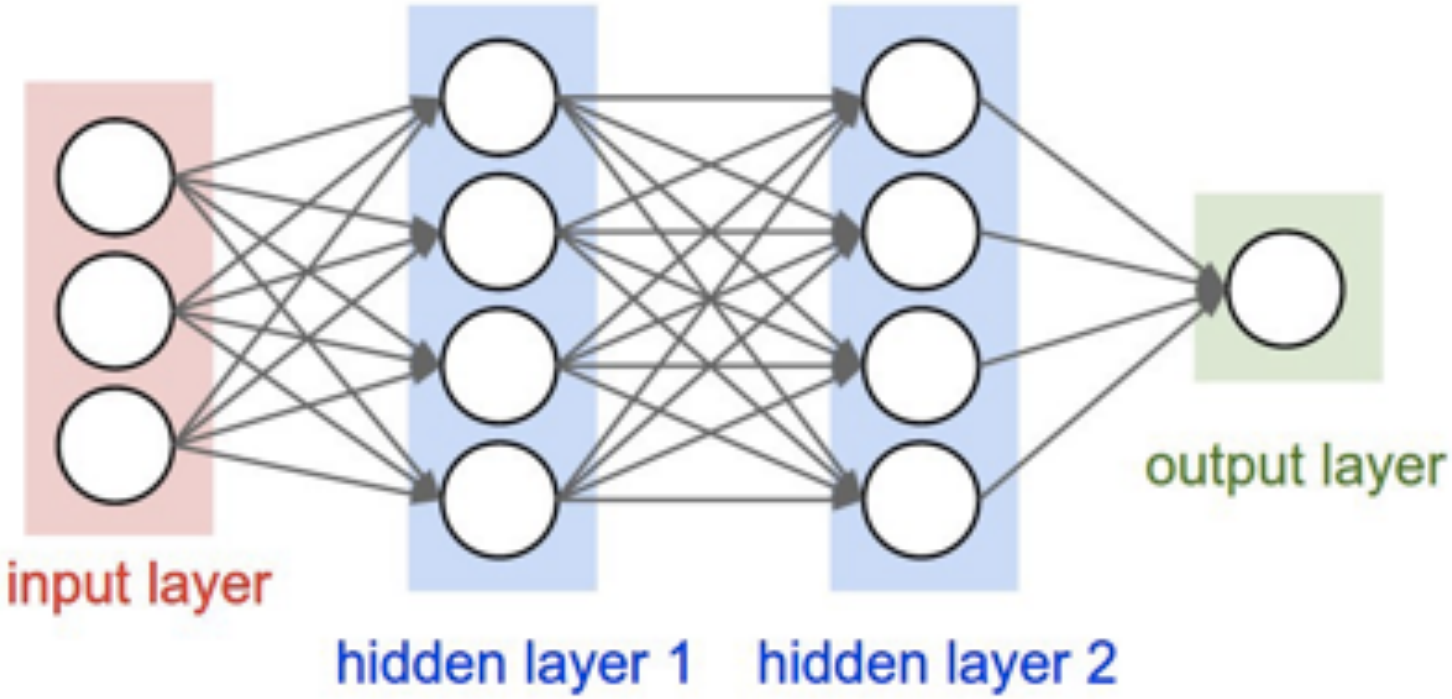
What is missing?

$$x = a^{(1)}$$

$$z^{(j+1)} = \theta^{(j)} a^{(j)}$$

$$a^{(j+1)} = \sigma(z^{(j+1)})$$

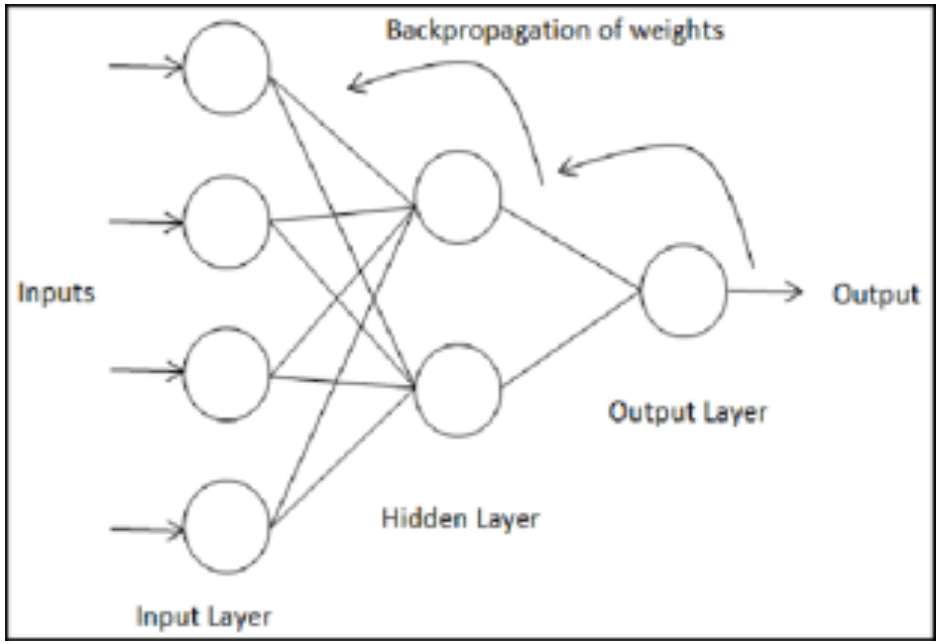
$$h_\theta(x) = a^{(L)} = \sigma(z^{(L)})$$



NN Theory:
Training details

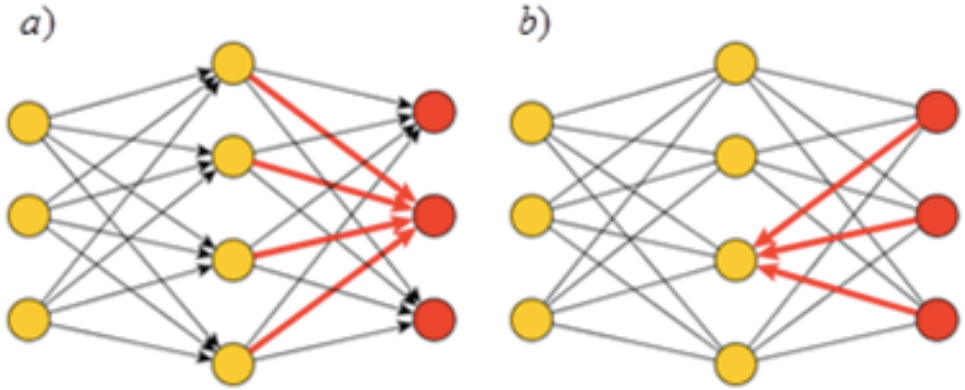
In neural networks, you forward propagate to get the output and compare it with the real value to get the error.

Now, to minimize the error, you propagate backwards by finding the derivative of error with respect to each weight and then subtracting this value from the weight value.



NN Theory:
Feedforward vs Backpropagation

The basic learning that has to be done in neural networks is training neurons when to get activated. Each neuron should activate only for particular type of inputs and not all inputs. Therefore, by propagating forward you see how well your neural network is behaving and find the error. After you find out that your network has error, you back propagate and use a form of **gradient descent** to update new values of weights. Then, you will again forward propagate to see how well those weights are performing and then will backward propagate to update the weights. This will go on until you reach some minima for error value.



NN Theory: Backpropagation

«**Gradient descent** is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. If, instead, one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent. Gradient descent was originally proposed by Cauchy in 1847.»

θ are weights
J(θ) is the cost function to minimize

$$
\begin{aligned}
&Repeat\ \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\
&\}
\end{aligned}
$$

NN Theory: Backpropagation

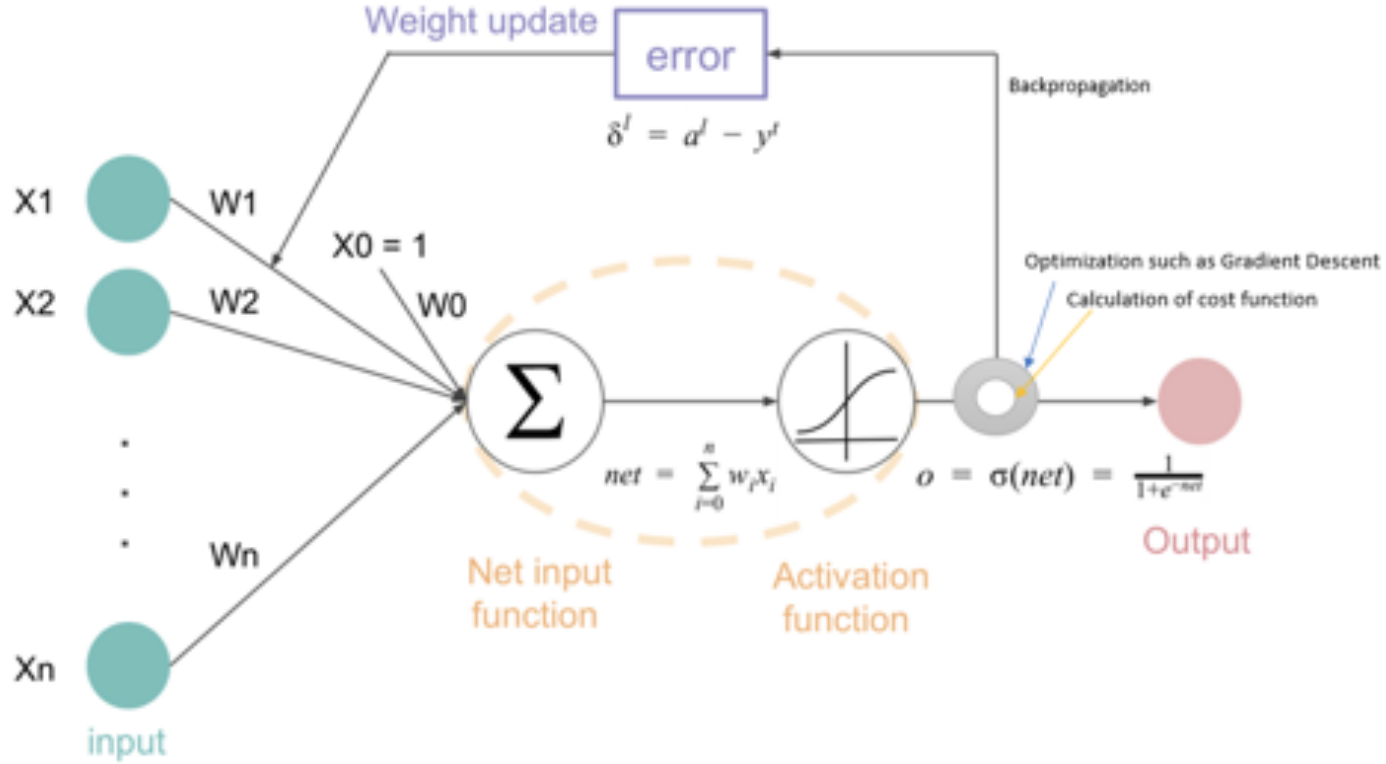
Backpropagation algorithm has 5 steps:

- Set a(1) = X; for the training examples
- Perform forward propagation and compute a(l) for the other layers (l = 2...L)
- Use y (output) and compute the delta value for the last layer $\delta(L) = h(x) — y$
- Compute the $\delta(l)$ values backwards for each layer (next slides)
- Calculate derivative values $\Delta(l) = (a(l))^T \circ \delta(l+1)$ for each layer, which represent the derivative of cost $J(\theta)$ with respect to $\theta(l)$ for layer l

Backpropagation is about determining how changing the weights impact the overall cost in the neural network.

NN Theory: Backpropagation

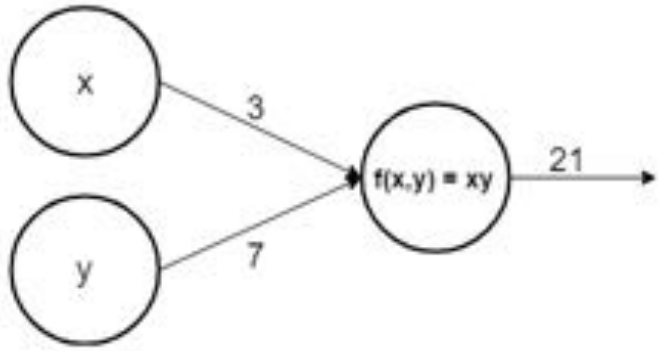
What it does is propagating the "error" backwards in the neural network. On the way back it is finding how much each weight is contributing in the overall "error". The weights that contribute more to the overall "error" will have larger derivation values, which means that they will change more (when computing Gradient descent).



Why derivatives?

NN Theory: Backpropagation

The derivative of a function (in our case J(θ)) on each variable (in our case weight θ) tells us the sensitivity of the function with respect to that variable or how changing the variable impacts the function value.
Let's look at a simple example neural network:



There are two input nodes x and y. The output function is calculating the product x and y. We can now compute the partial derivatives for both nodes.

$$\frac{df}{dy} = x \qquad \frac{df}{dx} = y$$

$$\frac{df}{dx} = 7 \qquad \frac{df}{dy} = 3$$

NN Theory: Backpropagation

The partial derivative with respect to x is saying that if x value increase for some value ε then it would increase the function (product xy) by 7ε and the partial derivative with respect to y is saying that if y value increase for some value ε then it would increase the function by 3ε.

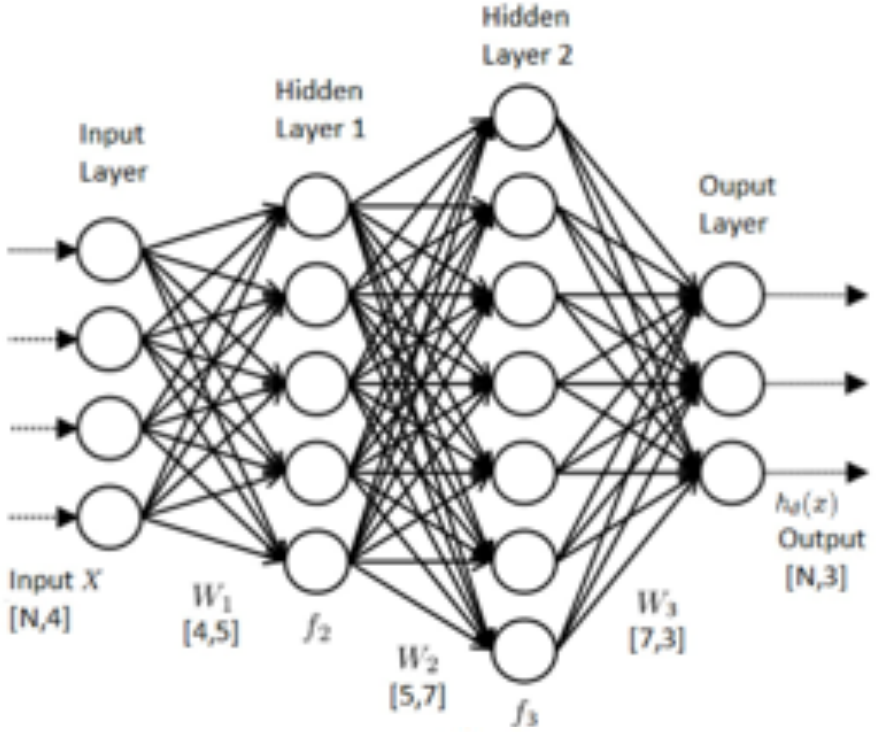
$\frac{df}{dy} = x$ $\frac{df}{dx} = y$

$\frac{df}{dx} = 7$ $\frac{df}{dy} = 3$

As we defined, Backpropagation algorithm is calculating the derivative of cost function with respect to each θ weight parameter. By doing this we determine how sensitive is the cost function J(θ) to each of these θ weight parameters. It also help us determine how much we should change each θ weight parameter when computing the Gradient descent. So at the end we get model that best fits our data.

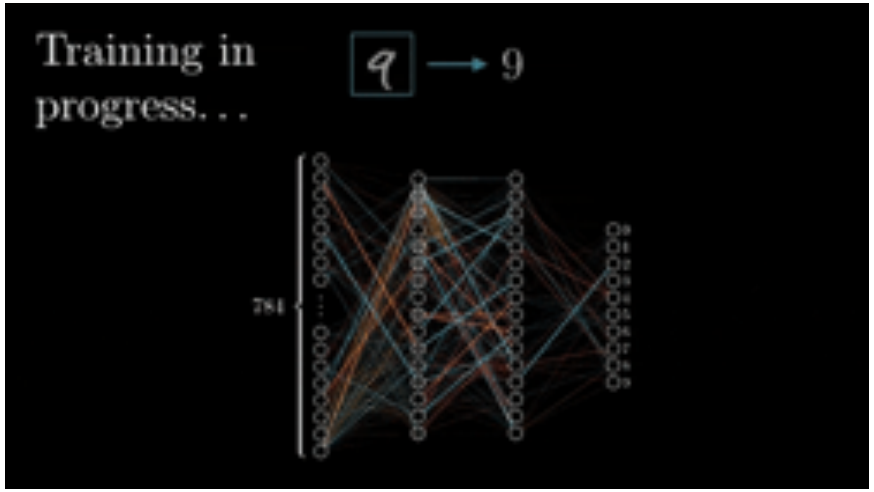
NN Theory: Backpropagation details

If we take, as an example, the following network. In this model we got 3 output nodes (K) and 2 hidden layers. As previously defined, the cost function for the neural network is J(θ).
What we need is to compute the partial derivative of J(θ) with respect to each θ parameters.
For math details see:
https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

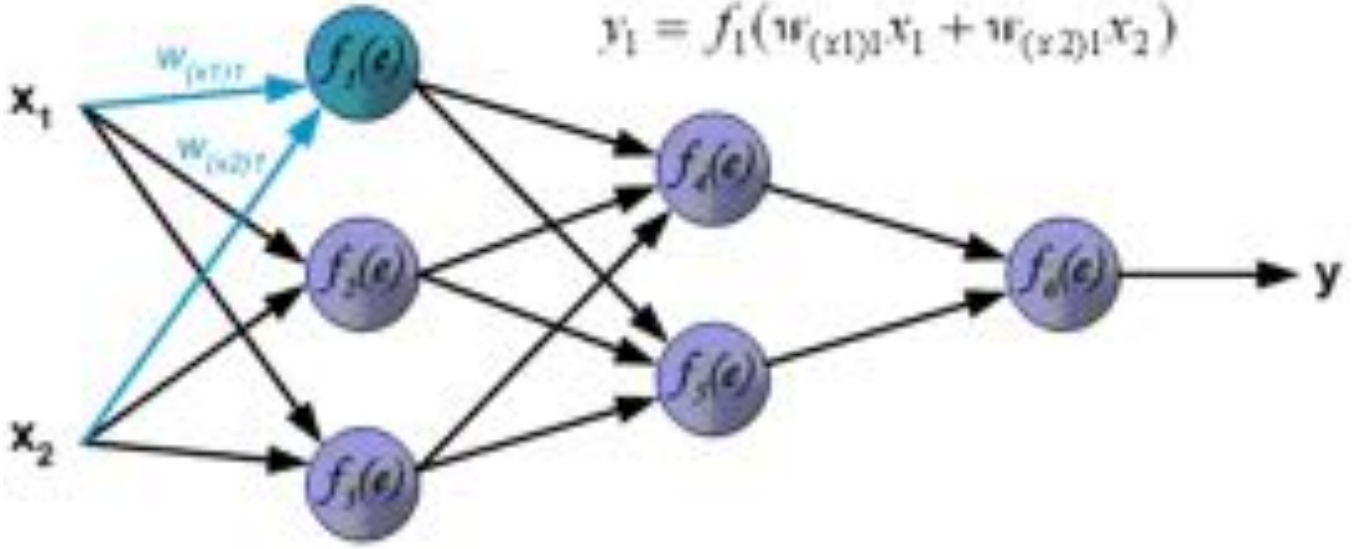


$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

NN Theory: Backpropagation



FP



Neural Network:
How to deal with complexity

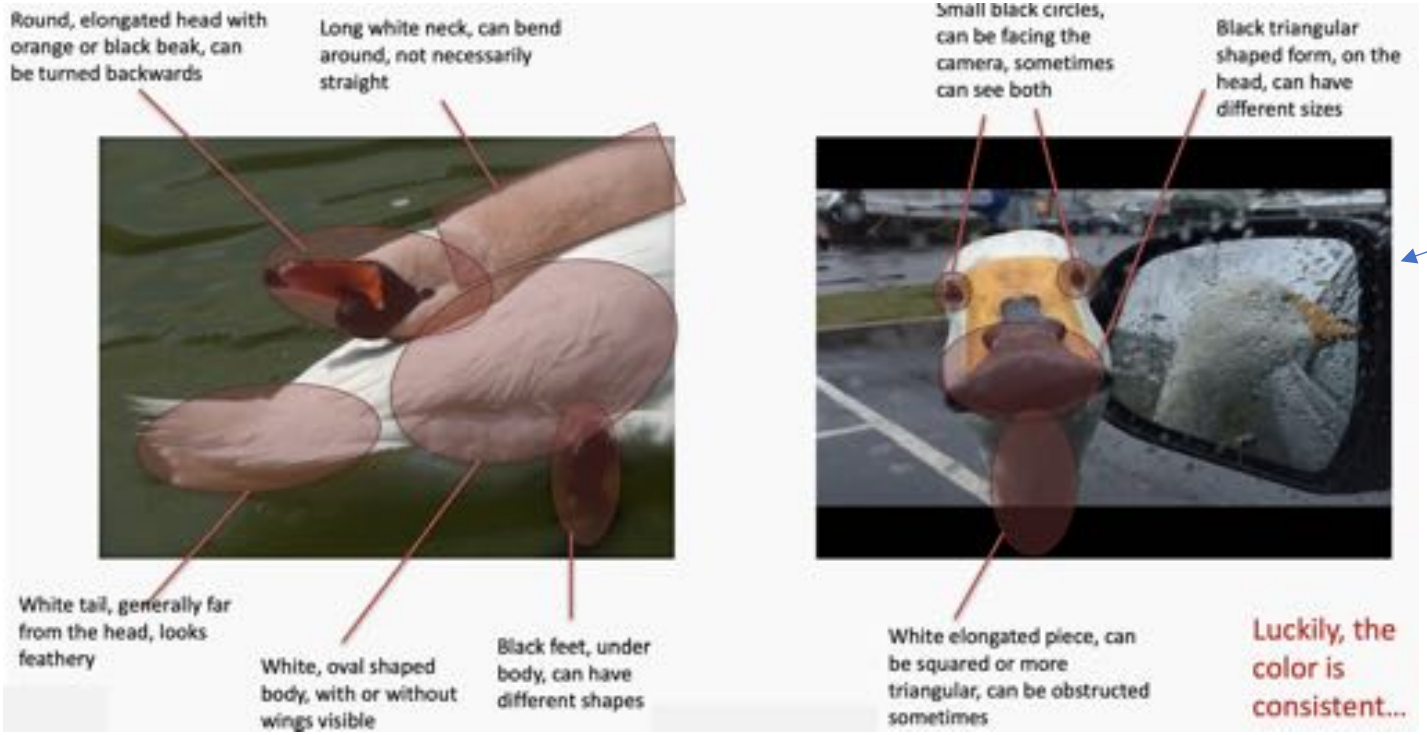
Let us assume that we want to create a neural network model that is capable of recognizing swans in images. The swan has certain characteristics that can be used to help determine whether a swan is present or not, such as its long neck, its white color, etc.

Neural Network:
How to deal with complexity

For some images, it may be more difficult to determine whether a swan is present, consider the following image.

Neural Network:
How to deal with complexity

The features are still present in the above image, but it is more difficult for us to pick out these characteristic features. Let us consider some more extreme cases.



The Perspective we told about in the first lesson

Neural Network:
How to deal with complexity

There are also a lot of inter-class variation.

Man in swan tent photographing swans

Neural Network:
How to deal with complexity

There are also a lot of inter-class variation.
Let's talk about neural networks. We've been basically talking about detecting features in images, in a very naïve way. Researchers built multiple computer vision techniques to deal with these issues: SIFT, FAST, SURF, BRIEF, etc. However, similar problems arose: the detectors were either too general or too over-engineered. Humans were designing these feature detectors, and that made them either too simple or hard to generalize.

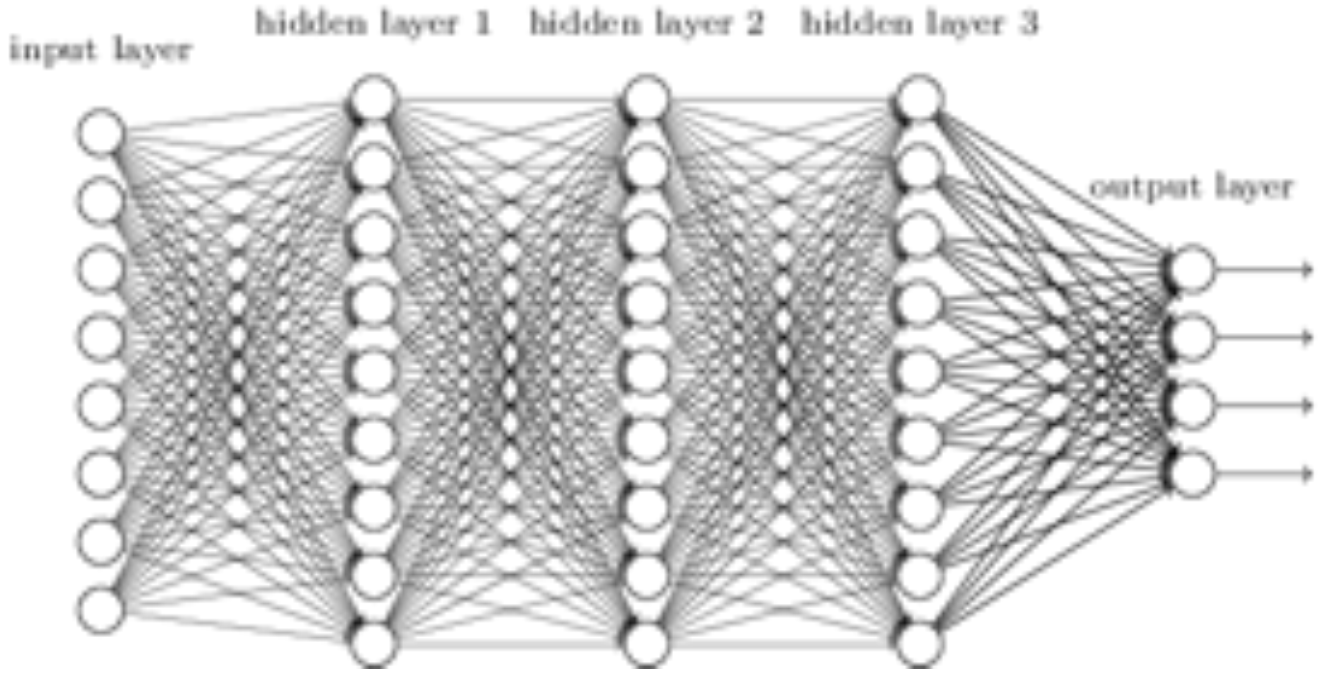
Traditional Neural Network are also called multilayer perceptron (MLP). These are modeled on the human brain, whereby neurons are stimulated by connected nodes and are only activated when a certain threshold value is reached…but…




What happens when the task to solve is difficult?
(A lot of inter and intra class variation)

Neural Network:
How to deal with complexity

- The amount of weights rapidly becomes unmanageable for large images. For a 224 x 224 pixel image with 3 color channels there are around 150,000 weights that must be trained! As a result, difficulties arise while training and overfitting can occur.

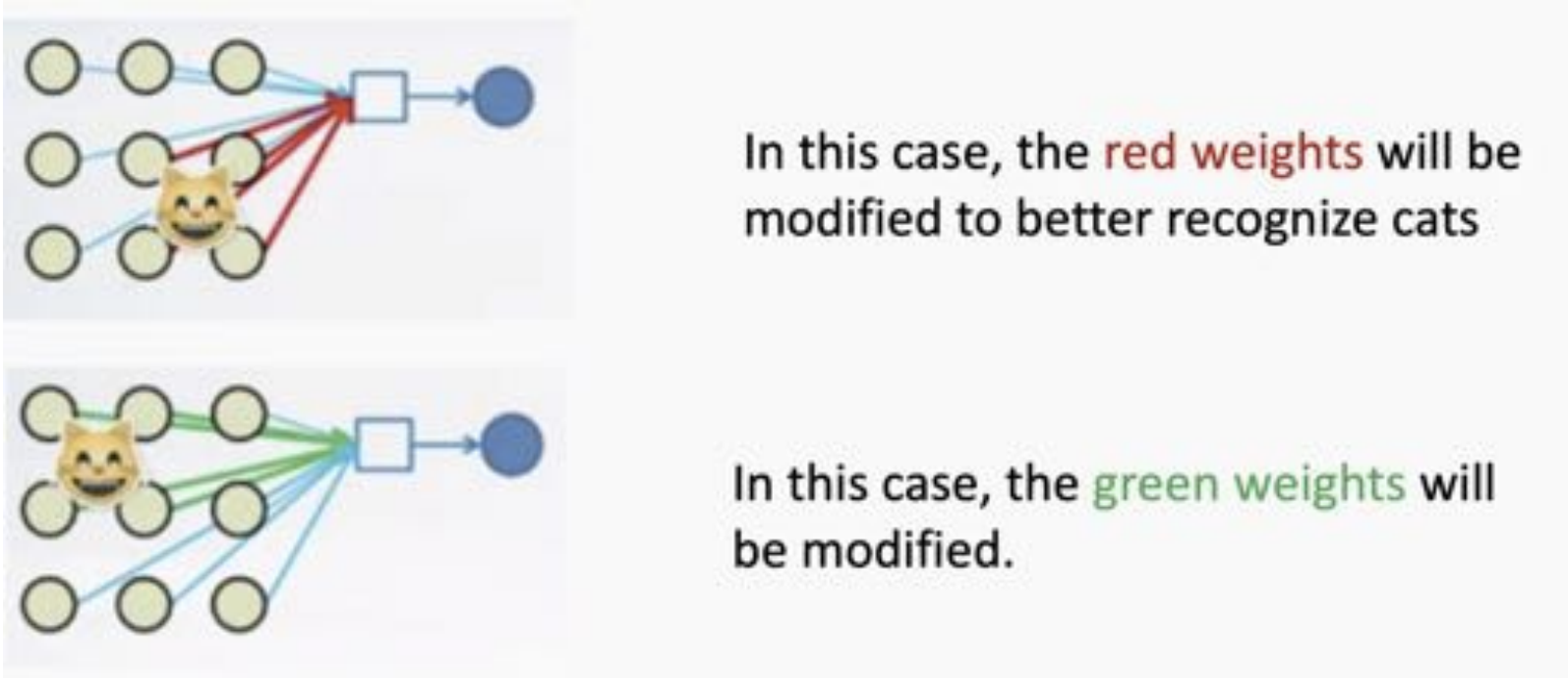


Neural Network:
How to deal with complexity

- MLPs react differently to an input (images) and its shifted version — they are not translation invariant. For example, if a picture of a cat appears in the top left of the image in one picture and the bottom right of another picture, the MLP will try to correct itself and assume that a cat will always appear in this section of the image.

Neural Network:
How to deal with complexity

- Spatial information is lost when the image is flattened into an MLP. Nodes that are close together are important because they help to define the features of an image. The approach is not robust, as cats could appear in yet another position.



Convolutional Neural Network: Introduction

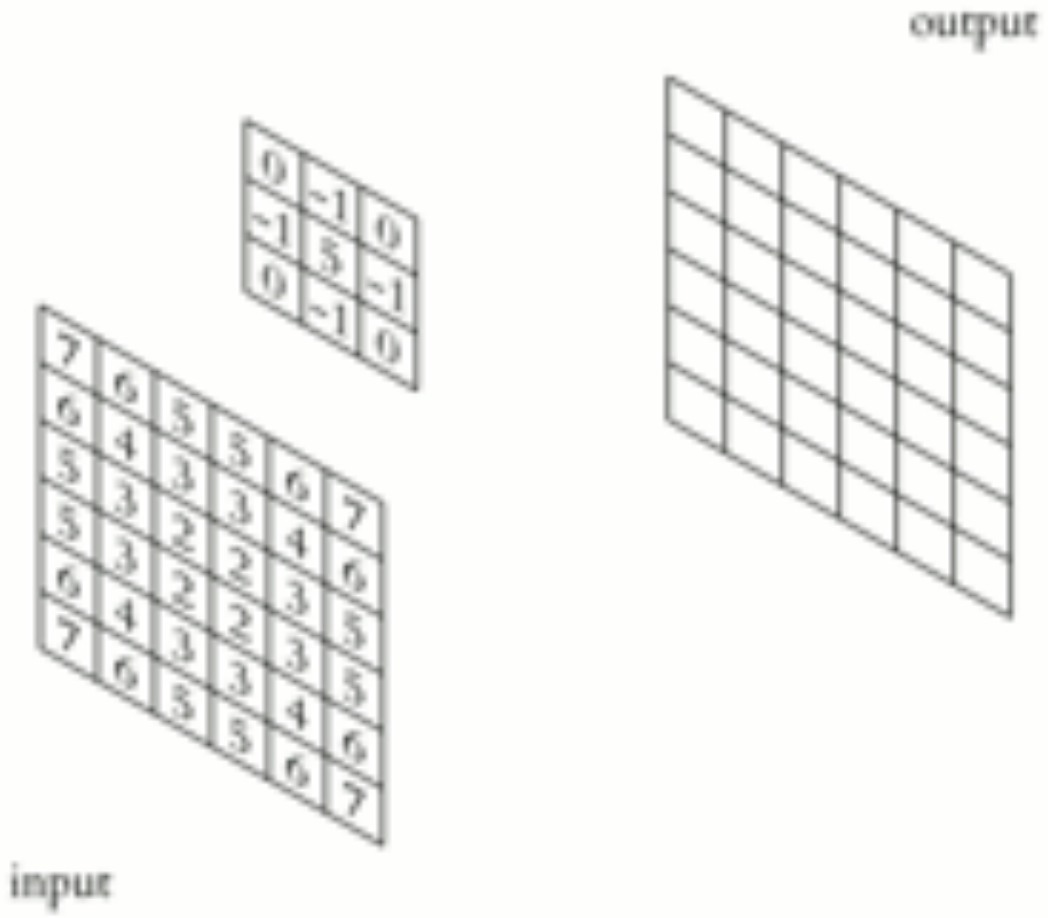
CNN's can be used to solve most of our problems.

Nearby pixels are more strongly related than distant ones.

Objects are built up out of smaller parts.

Convolutional Neural Network: Introduction

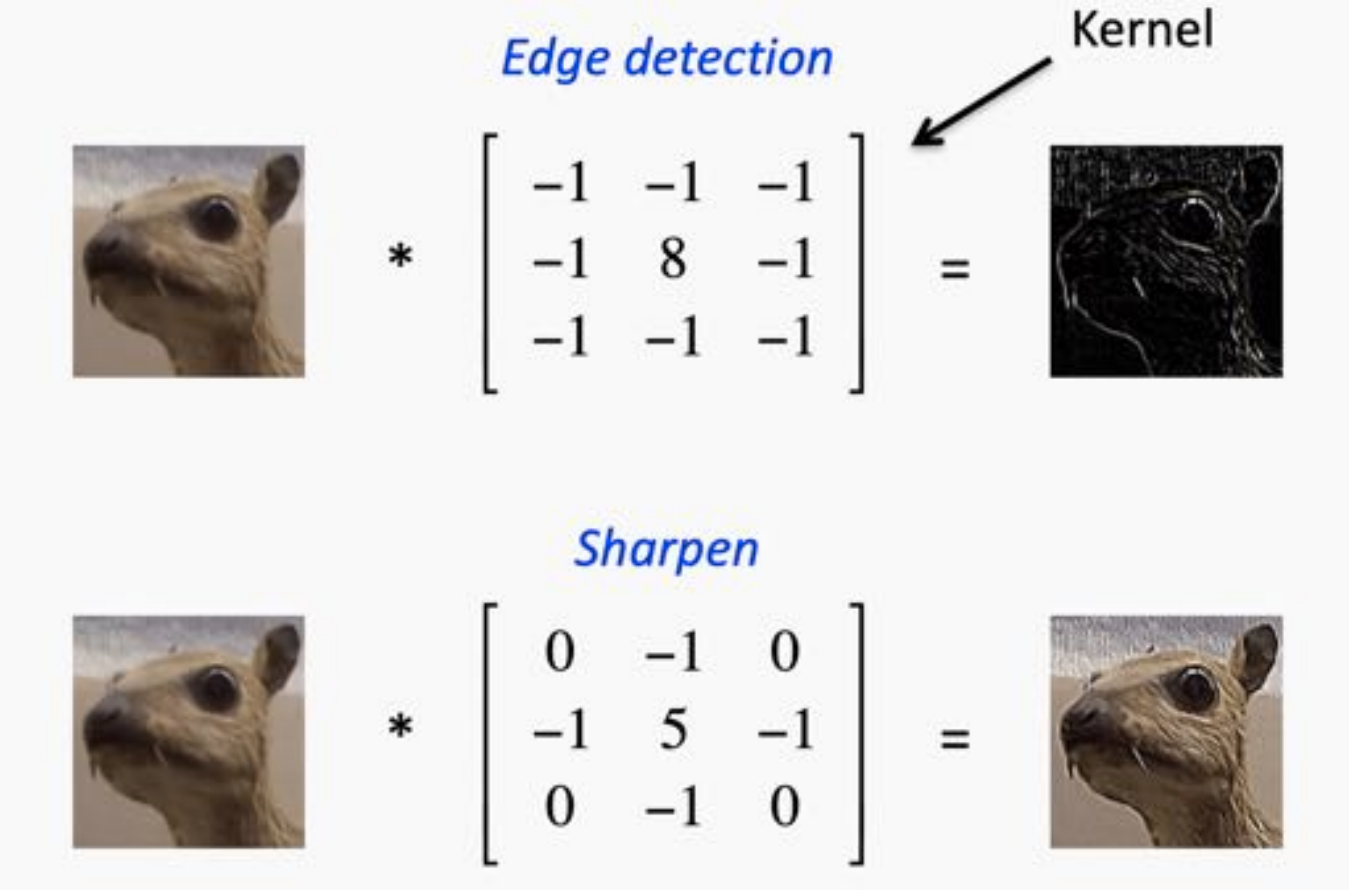
We analyze the influence of nearby pixels by using something called a filter. A filter is exactly what you think it is, in our situation, we take a filter of a size specified by the user (a rule of thumb is 3x3 or 5x5) and we move this across the image from top left to bottom right. For each point on the image, a value is calculated based on the filter using a convolution operation.



Convolutional Neural Network: Introduction

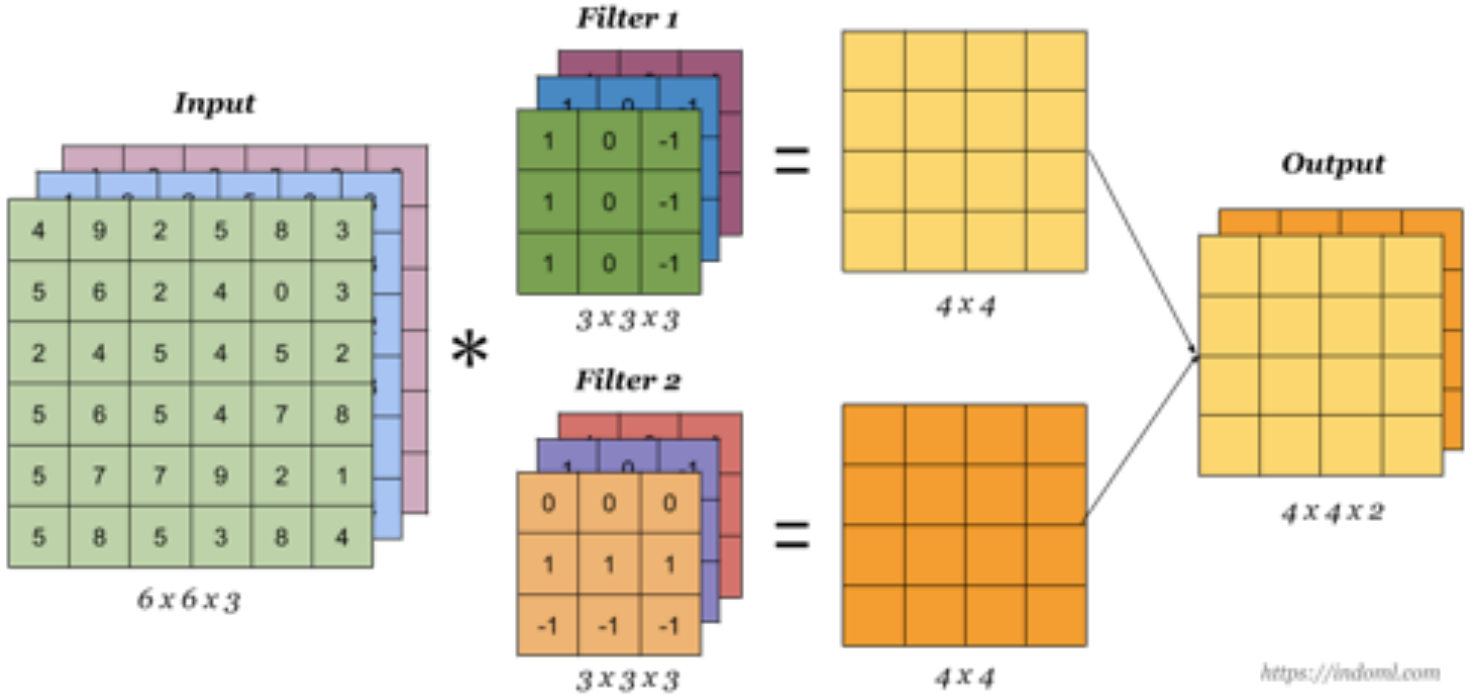
When building the network, we randomly specify values for the filters, which then continuously update themselves as the network is trained. It is very very unlikely that two filters that are the same will be produced unless the number of chosen filters is extremely large.

Some example of filter ---->



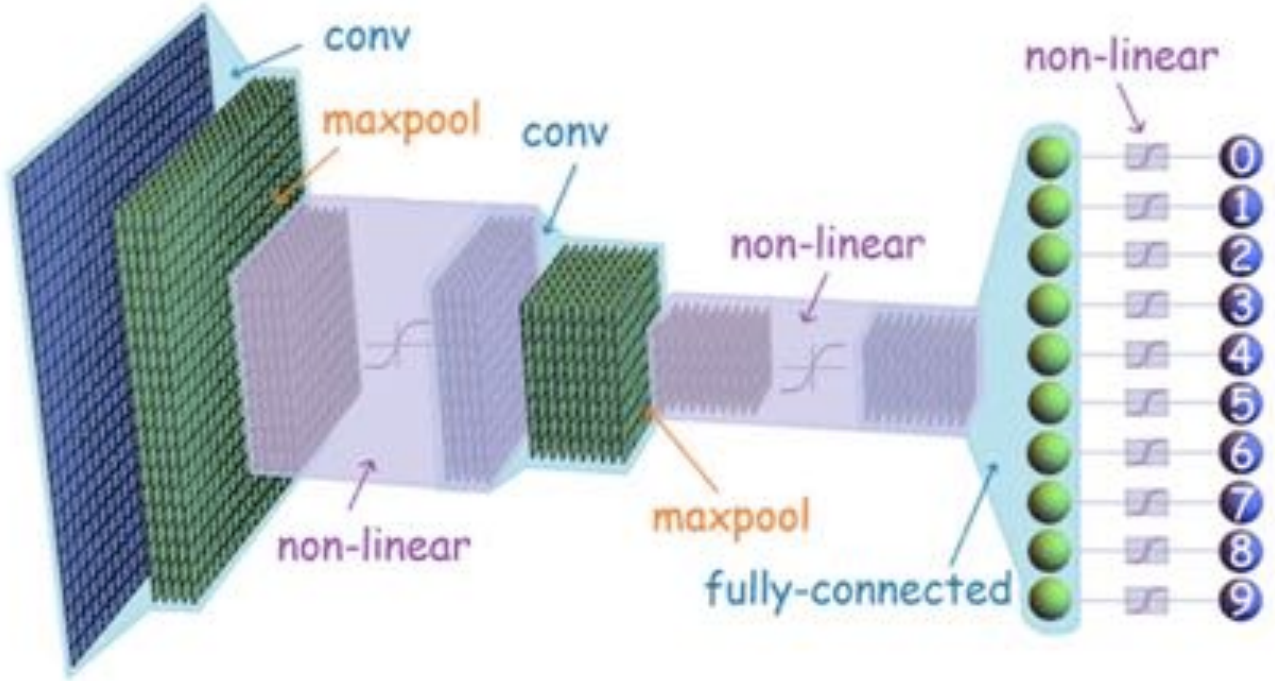
Convolutional Neural Network: Introduction

After the filters have passed over the image, a feature map is generated for each filter. These are then taken through an activation function, which decides whether a certain feature is present at a given location in the image. We can then do a lot of things, such as adding more filtering layers and creating more feature maps, which become more and more abstract as we create a deeper CNN.



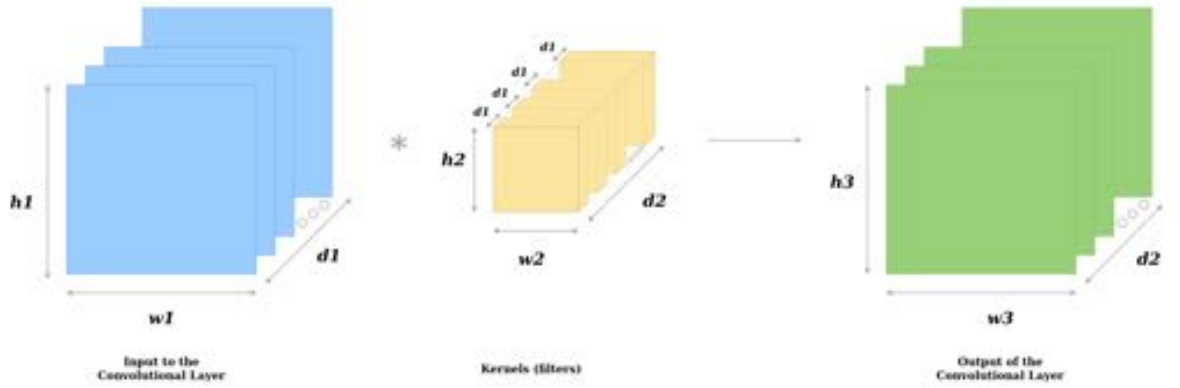
Convolutional Neural Network: Structure

A ConvNet usually has 3 types of main layers:
1)Convolutional Layer (CONV)
2) Pooling Layer (POOL)
3) Fully Connected Layer (FC)



Convolutional Neural Network: Structure

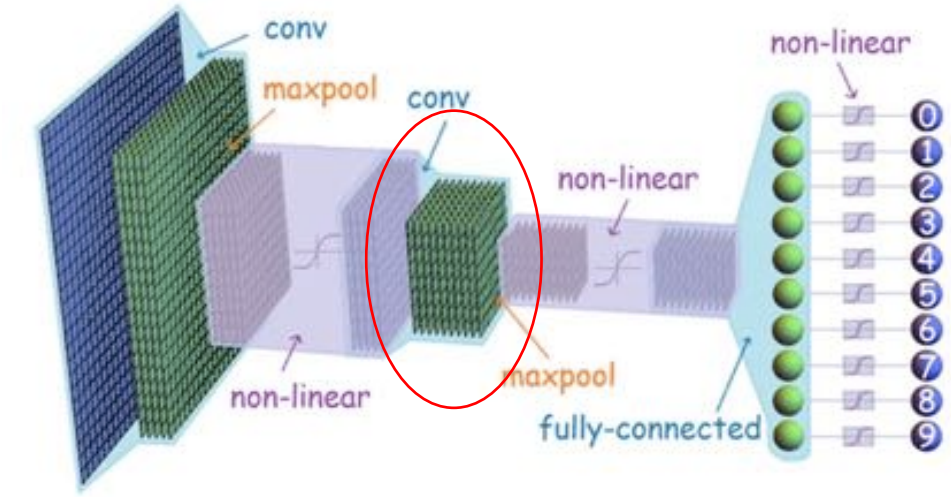
1) Convolutional Layer is the first layer in a CNN.
It gets as input a matrix of the dimensions [h1 * w1 * d1], which is the blue matrix in the above image.
Next, we have kernels (filters) we discussed above. A kernel is a matrix with the dimensions [h2 * w2 * d1], which is one yellow cuboid of the multiple cuboid (kernels) stacked on top of each other (in the kernels layer) in the above image.
And then, we have a output for this layer, the green matrix, which has dimensions [h3 * w3 * d2].



Convolutional Neural Network: Structure

2) Pooling Layer (POOL): There are two types of pooling: Max Pooling; Average Pooling. The main purpose of a pooling layer is to reduce the number of parameters of the input tensor (is a mathematical object that has n indices and m^n components and obeys certain transformation rules.) and thus

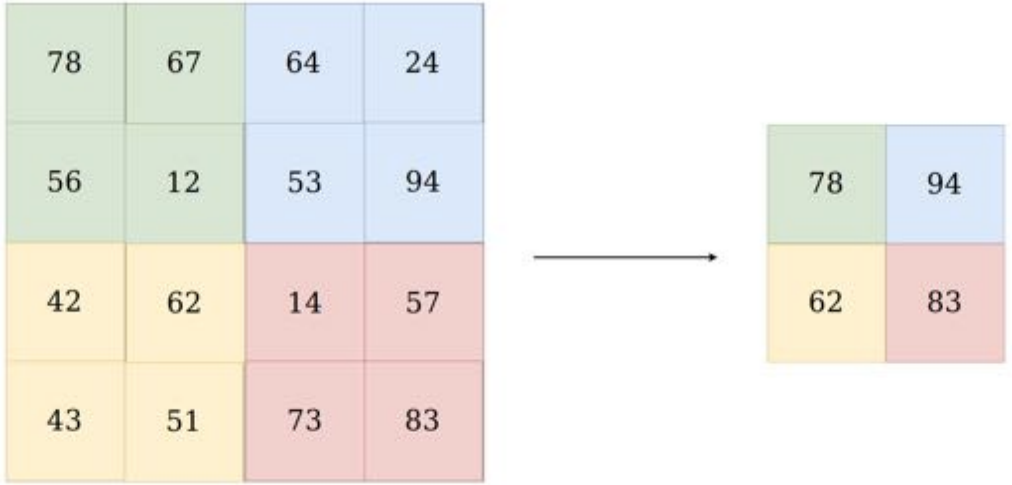
- Helps reduce overfitting
- Extract representative features from the input tensor
- Reduces computation and thus aids efficiency



Convolutional Neural Network: Structure

In case of Max Pooling, a kernel of size n*n (2x2 in the above example) is moved across the matrix and for each position the max value is taken and put in the corresponding position of the output matrix.

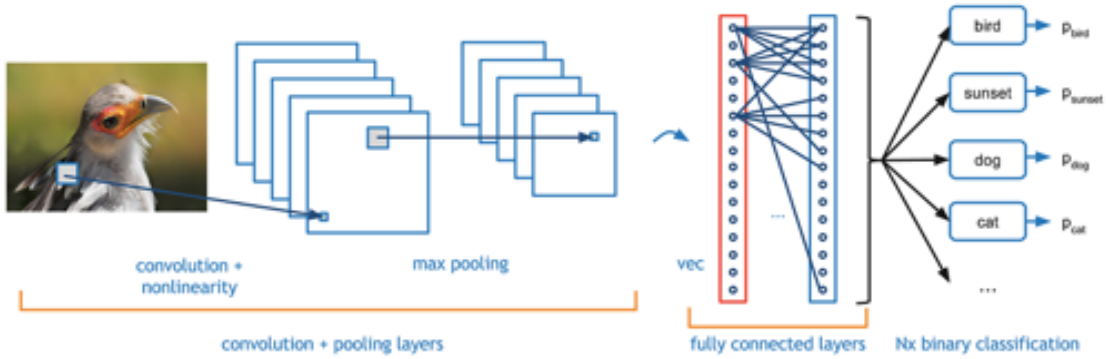
In case of Average Pooling, a kernel of size n*n is moved across the matrix and for each position the average is taken of all the values and put in the corresponding position of the output matrix. This is repeated for each channel in the input tensor. And so we get the output tensor. So, a thing to note is, Pooling downsamples the image in its height and width but the number of channels(depth) stays the same.



Convolutional Neural Network: Structure

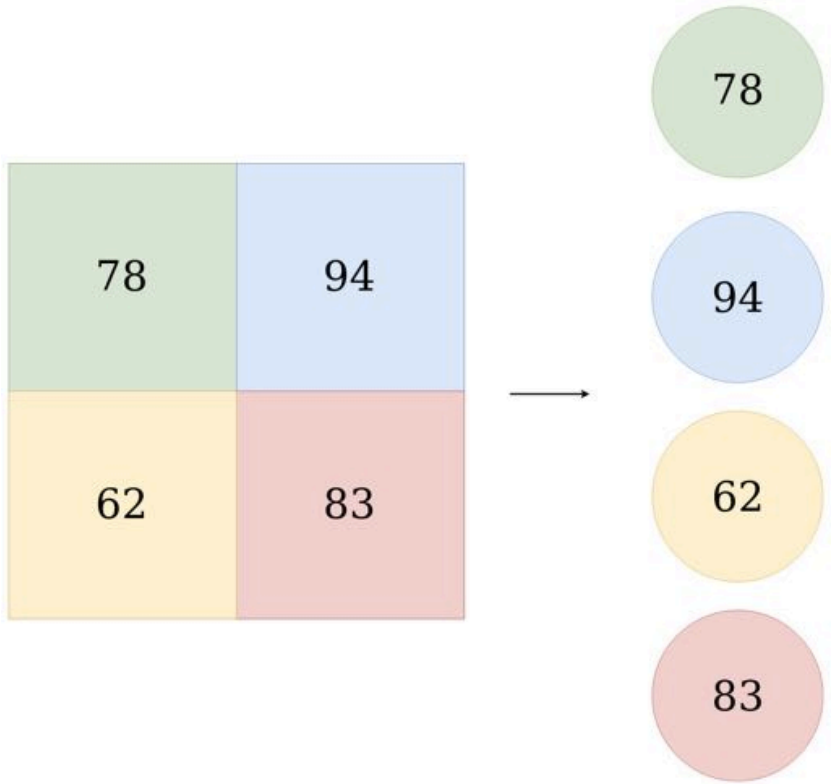
3) Fully Connected Layer (FC): Fully Connected Layer is simply, feed forward neural networks. Fully Connected Layers form the last few layers in the network.
The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

Note: Flattened->The output from the final (and any) Pooling and Convolutional Layer is a 3-dimensional matrix, to flatten that is to unroll all its values into a vector.



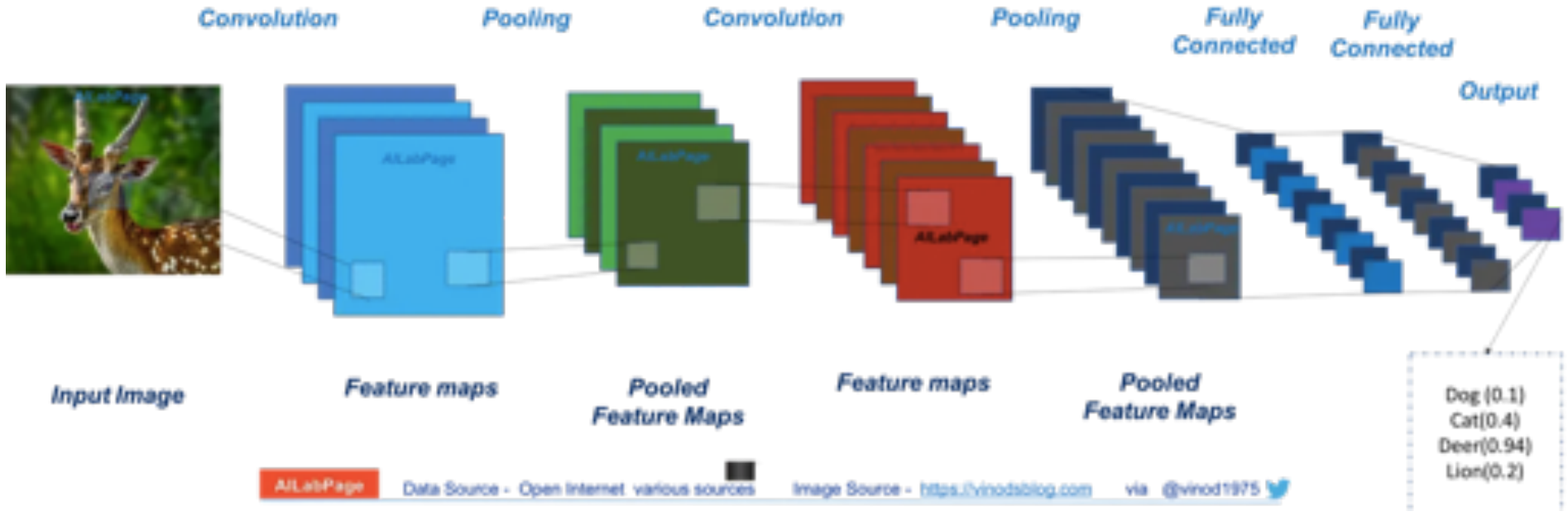
Convolutional Neural Network: Structure

This Flattened vector is then connected to a few fully connected layers which are same as Artificial Neural Networks and perform the same mathematical operations!
This calculation is repeated for each layer.



Convolutional Neural Network: Structure

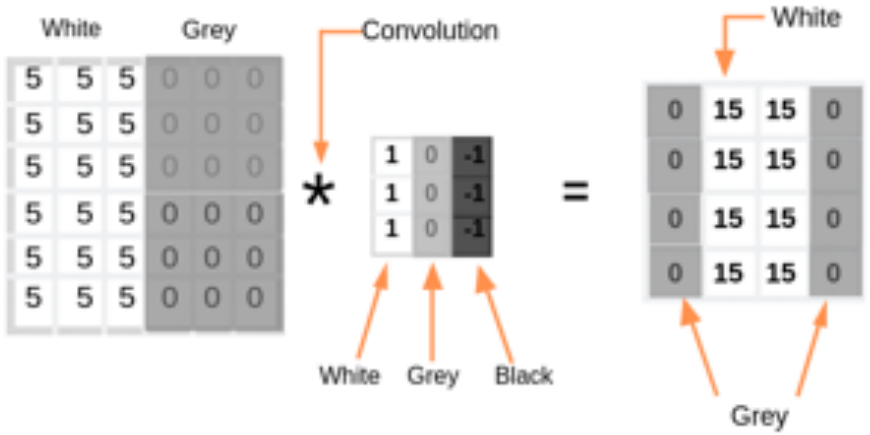
After passing through the fully connected layers, the final layer uses the softmax activation function (instead of ReLU) which is used to get probabilities of the input being in a particular class (classification).
And so finally, we have the probabilities of the object in the image belonging to the different classes!



Convolutional Neural Network: Image Processing & Filtering

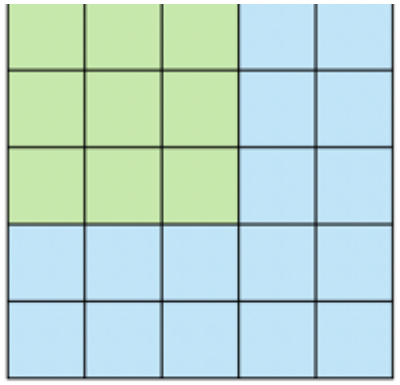
Other important concepts in CNN and image processing in general:

- **Edge detection**: Every image has vertical and horizontal edges which actually combining to form a image. Convolution operation is used with some filters for detecting edges. Suppose you have gray scale image with dimension 6 x 6 and filter of dimension 3 x 3(say). When 6x6 greyscale image convolve with 3x3 filter, we get 4x4image.

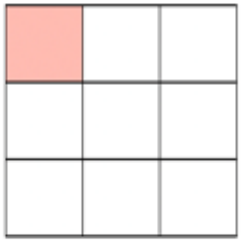


Convolutional Neural Network: Image Processing & Filtering

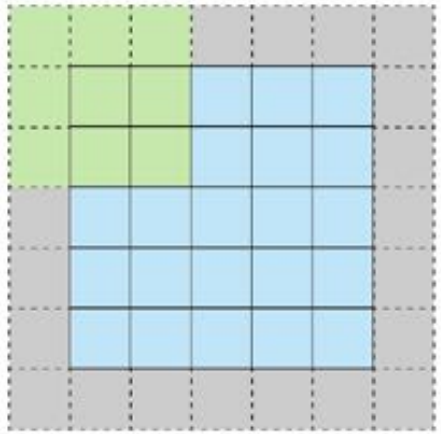
- **Stride and Padding:** Stride denotes how many steps we are moving in each steps in convolution.By default it is one. We can observe that the size of output is smaller that input. To maintain the dimension of output as in input , we use padding. Padding is a process of adding zeros to the input matrix symmetrically. In the following example,the extra grey blocks denote the padding. It is used to make the dimension of output same as input.



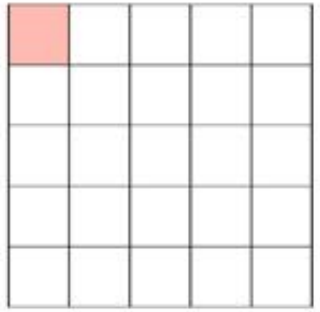
Stride 1



Feature Map



Stride 1 with Padding



Feature Map

Examples of CNN

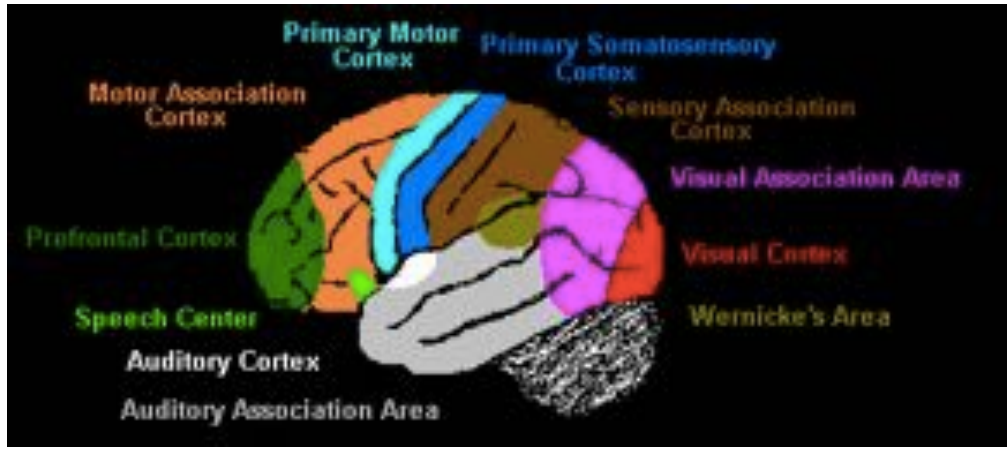
https://www.youtube.com/watch?v=ptzpJwtbPp0

https://www.youtube.com/watch?v=vdDqMtdyUYU

How the CNN is related to Human Visual System or Brain?

CNN & Brain

Everytime we build a network we want to imitate the human brain.
The visual information is passed from one cortical area to another and each cortical area is more specialized than the last one. The neurons in the specific field only respond to the specific actions.

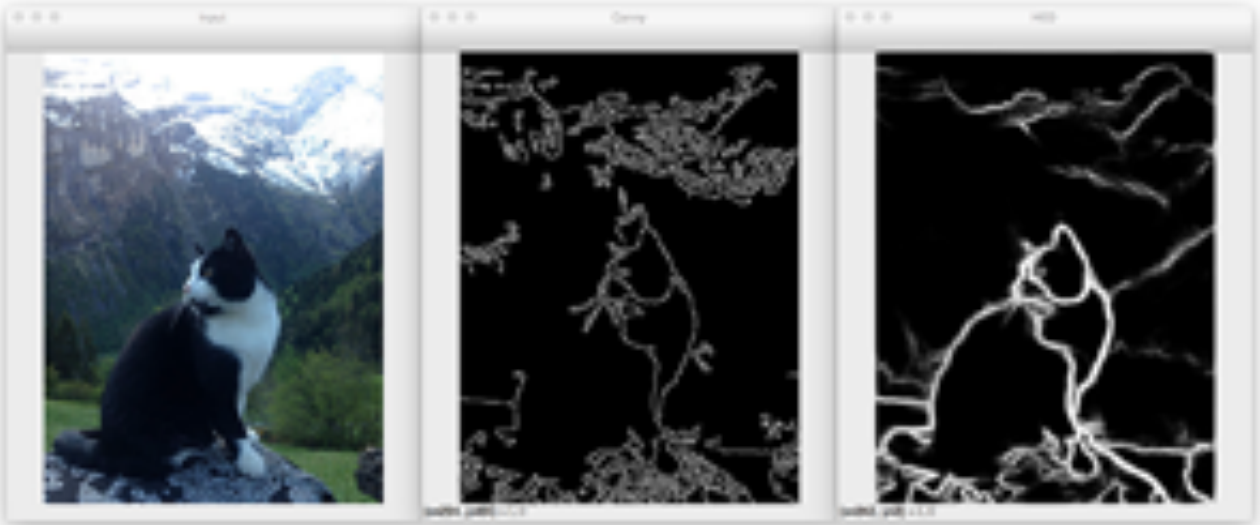


The above visual cortex acts as layers of the CNN. Lets take scenarios as Edge Detection, Face Detection, In-variance detection (i.e. Rotated Face Detection, Large or Small Face Detection)

CNN & Brain

Edge Detection : Using convolution operation on image with Sobel Kernel we can detect the edges.

Max Pooling : It is used to detect, where the objects are located in the Image based on the output of the each cluster of neurons in previous layer. As the face is detected where ever it is; doesn't depend on the location of face in image.






CNN & Brain

ReLU (Rectified Linear Unit) : As human brain never stops learning, it (brain) always learns from the observations and experiences; i.e. the inputs which it receives form the sensory organs, are utilized at some or another point; but the learning never becomes “Zero”. To add this feature to the neural networks ReLU is used. The activation function is : f(x) = max(0,x). For any activation function, we must be able to take the derivative of that function and with ReLU we can do that. But the derivative at zero is not defined for the ReLU. Due to zero we can have the problem of dead activation state. This implies there will be no weight change meaning no learning. But in humans it doesn’t happens often. To tackle this problem the concept of Leaky ReLU is used.

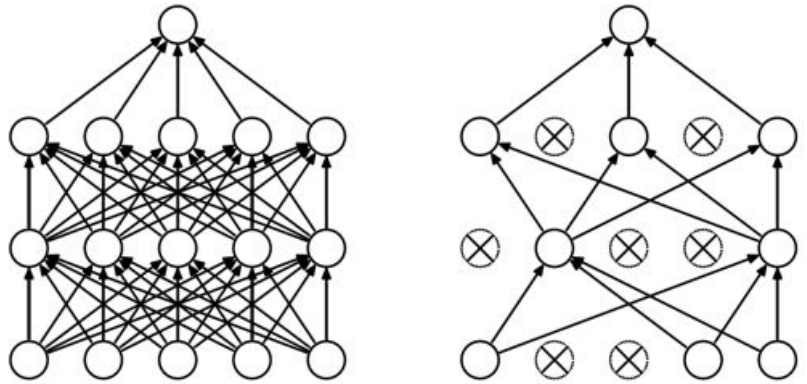
Leaky ReLU : The function is : f(x) = x if x>0, a small fraction of x if x>0 . With this we are avoiding the problem of dead states. That is the network can continue to learn; but it can face the problem of Vanishing gradient.

CNN & Brain

Data Augmentation : We humans can recognize the face even if it is inverted, rotated,flipped, reflected or skewed. Using Data Augmentation technique we can convert a single image into different types of image and use the newly formed images for training the CNN. After that, the CNN will be able to detect In-variance based data such as rotated faces, large and small faces, flipped faces etc (i.e. the objects will be recognized even if they are not in their original position).

CNN & Brain

Dropouts : Does all the neurons present in our brain fire to learn something? The answer is ‘’NO’’. It is not necessary that they fire in a linear fashion or in back-propagation. Some of the neurons may stay inactive in one phase of learning and may get active in another phase of learning or vice-versa. This gives the capability of independent learning to the neurons. To have this in the networks, the concept of dropouts is introduced. After applying dropout with probability p, the randomly selected individual nodes/neurons are dropped out of that epoch for learning process and the respective incoming and outgoing edges are also dropped out. It is much used to avoid the over-fitting in the network.



(a) Standard Neural Net (b) After applying dropout.

Other kinds of networks in the next lesson…