Simple Introduction to Convolutional Neural Networks

Matthew Stewart, PhD Researcher Feb 26, 2019

In this article, I will explain the concept of convolution neural networks (CNN's) using many swan pictures and will make the case of using CNN's over regular multilayer perceptron neural networks for processing images.

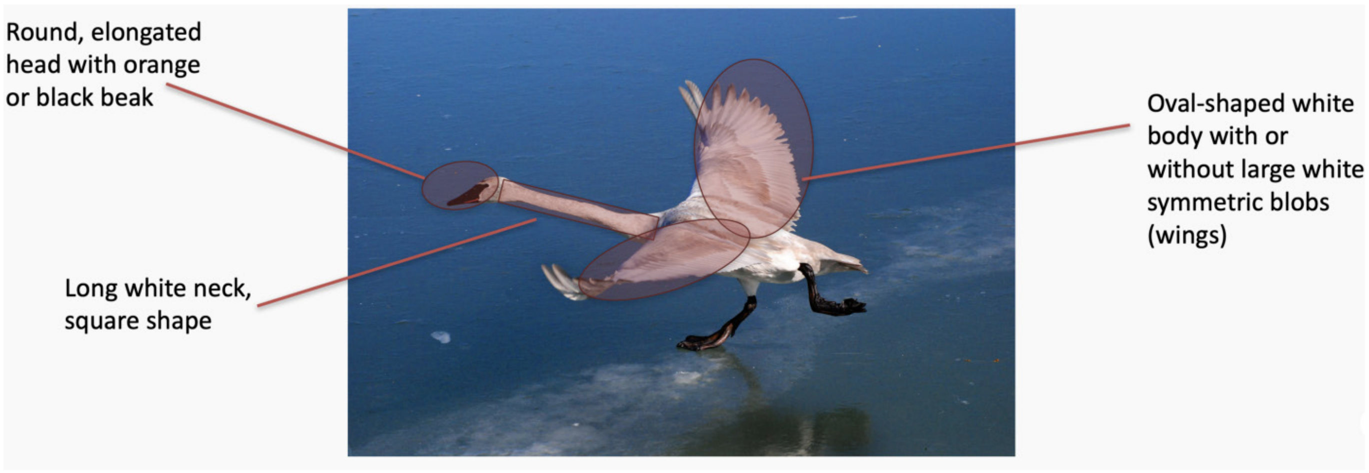
Image Analysis

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.

Oval-shaped white blob (body)
Round, elongated oval with orange protuberance
Long white rectangular shape (neck)

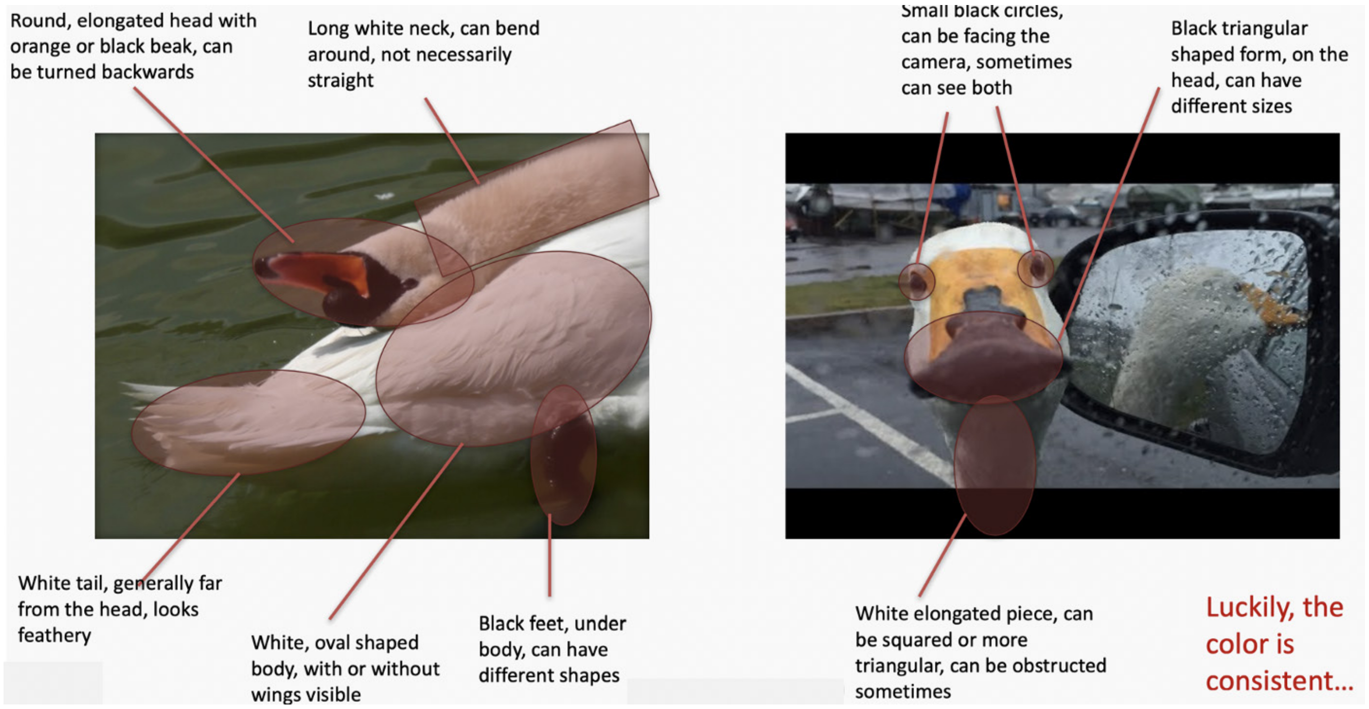
A swan has certain characteristic features that can be used for recognition purposes.

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


Round, elongated head with orange or black beak
Long white neck, square shape
Oval-shaped white body with or without large white symmetric blobs (wings)

Harder to classify swan image.

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.

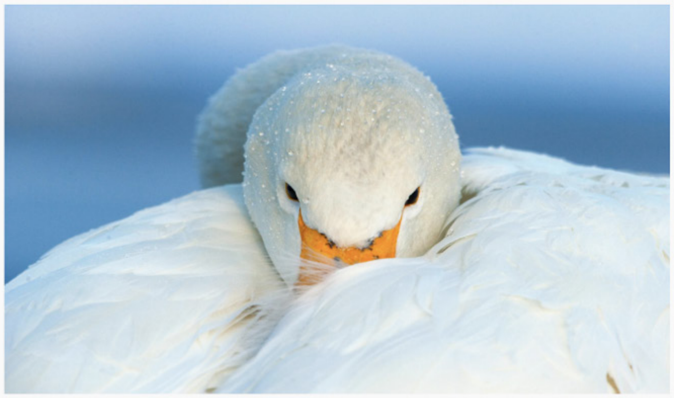

Round, elongated head with orange or black beak, can be turned backwards
Long white neck, can bend around, not necessarily straight
White tail, generally far from the head, looks feathery
White, oval shaped body, with or without wings visible
Black feet, under body, can have different shapes
Small black circles, can be facing the camera, sometimes can see both
Black triangular shaped form, on the head, can have different sizes
White elongated piece, can be squared or more triangular, can be obstructed sometimes
Luckily, the color is consistent…

Extreme cases of swan classification.

At least the color is consistent, right? Or is it…

Don’t forget about those black swans!

Can it get any worse? It definitely can.



Man in swan tent photographing swans

OK, enough with the swan pictures now. 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.

- What if we learned the features to detect?
- We need a system that can do Representation Learning (or Feature Learning).

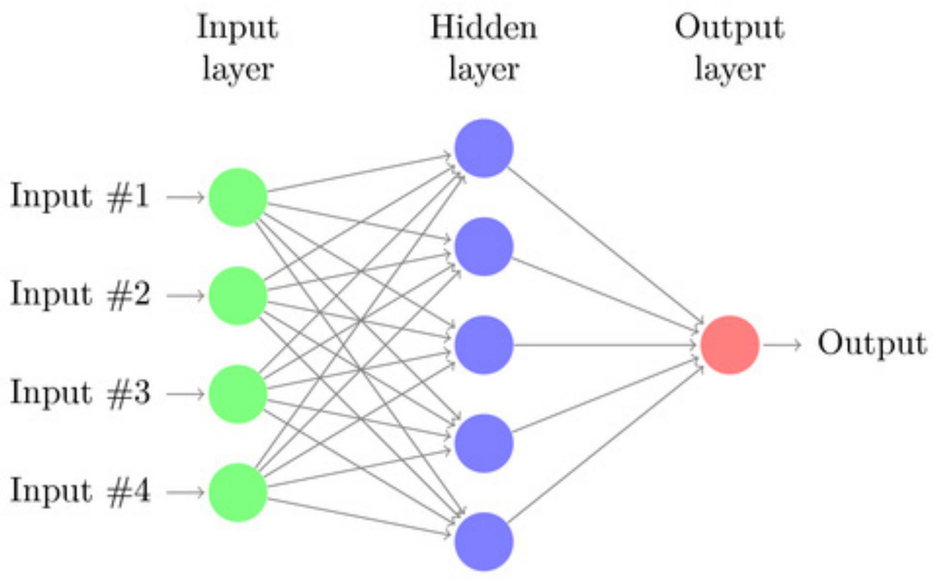
Representation Learning is a technique that allows a system to automatically find relevant features for a given task. Replaces manual feature engineering. There are several techniques for this:

- Unsupervised (K-means, PCA, …)
- Supervised (Sup. Dictionary learning, Neural Networks!)

The Problem with Traditional Neural Networks

I will assume that you are already familiar with traditional neural networks called the multilayer perceptron (MLP). If you are not familiar with these, there are hundreds of tutorials on Medium outlining how MLPs work. 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.

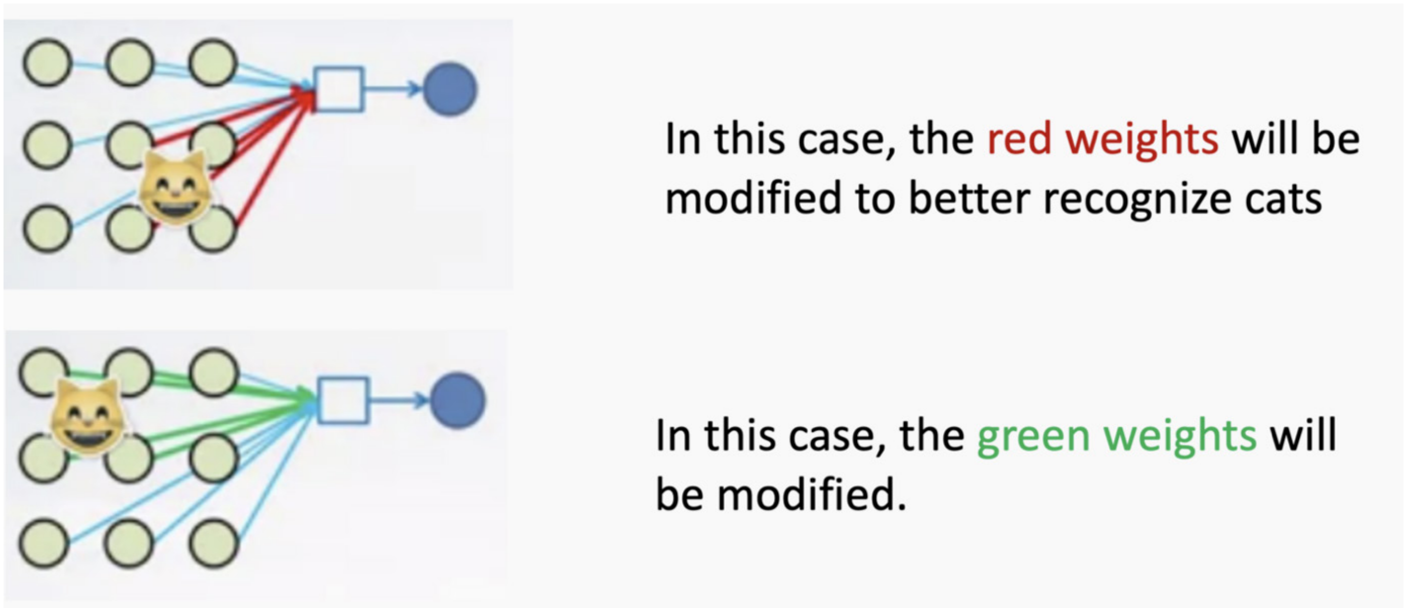
A standard multilayer perceptron (traditional neural network).



There are several drawbacks of MLP’s, especially when it comes to image processing. MLPs use one perceptron for each input (e.g. pixel in an image, multiplied by 3 in RGB case). 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 whilst training and overfitting can occur.

Another common problem is that 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.

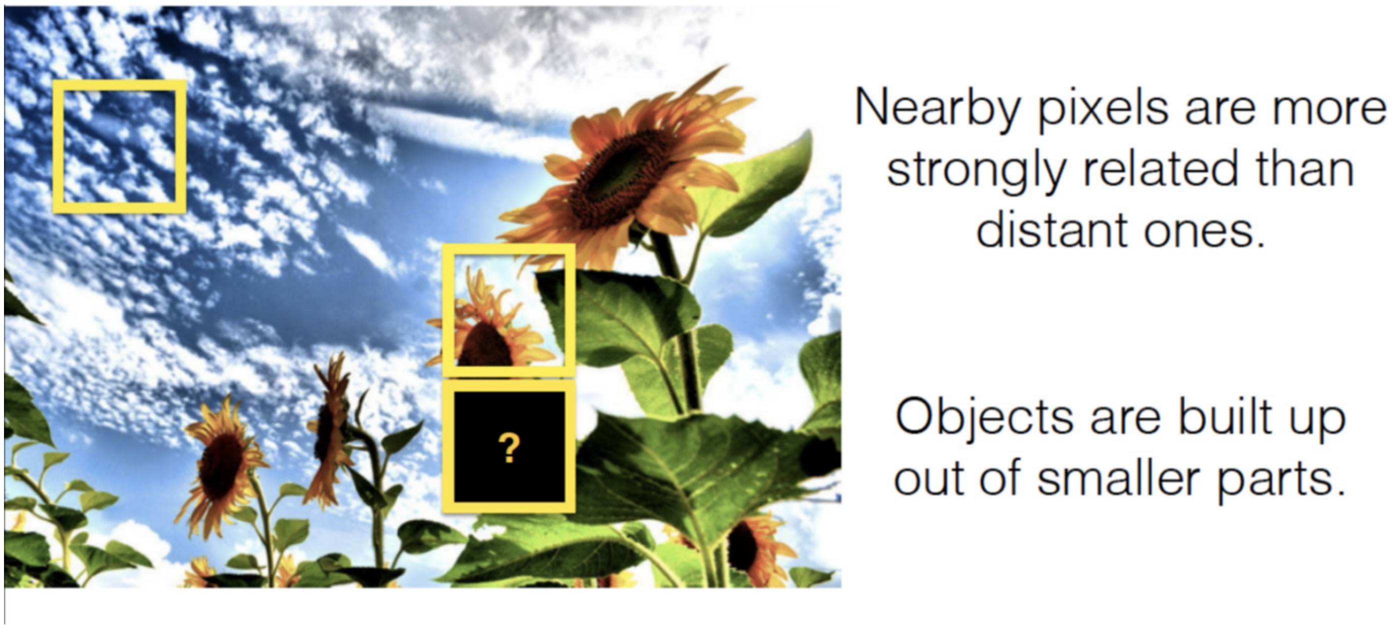
Clearly, MLPs are not the best idea to use for image processing. One of the main problems is that spatial information is lost when the image is flattened into an MLP. Nodes that are close together are important because they help to define the features of an image. We thus need a way to leverage the spatial correlation of the image features (pixels) in such a way that we can see the cat in our picture no matter where it may appear. In the below image, we are learning redundant features. The approach is not robust, as cats could appear in yet another position.


In this case, the red weights will be modified to better recognize cats
In this case, the green weights will be modified.

A cat detector using an MLP which changes as the position of the cat changes.

Enter the Convolutional Neural Network

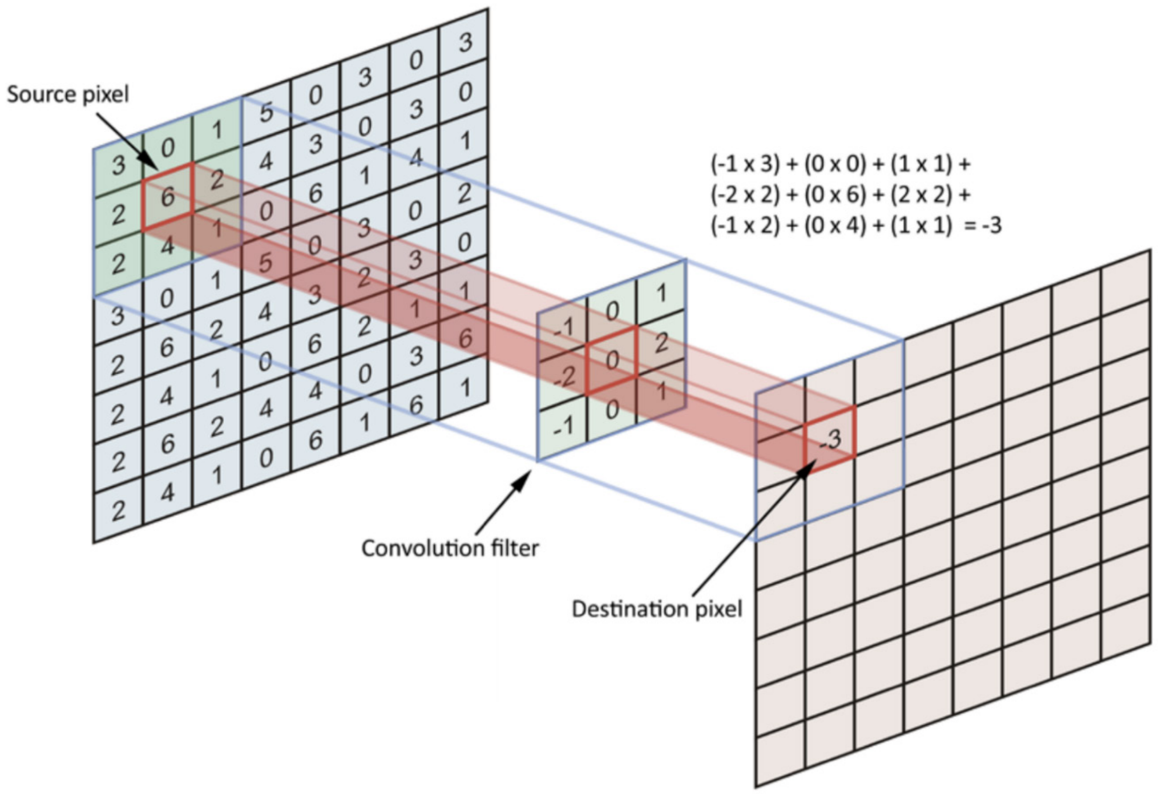
I hope the case is clear why MLPs are a terrible idea to use for image processing. Now let us move on and discuss how CNN's can be used to solve most of our problems.


?
Nearby pixels are more strongly related than distant ones.
Objects are built up out of smaller parts.

CNN's leverage the fact that nearby pixels are more strongly related than distant ones.

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.

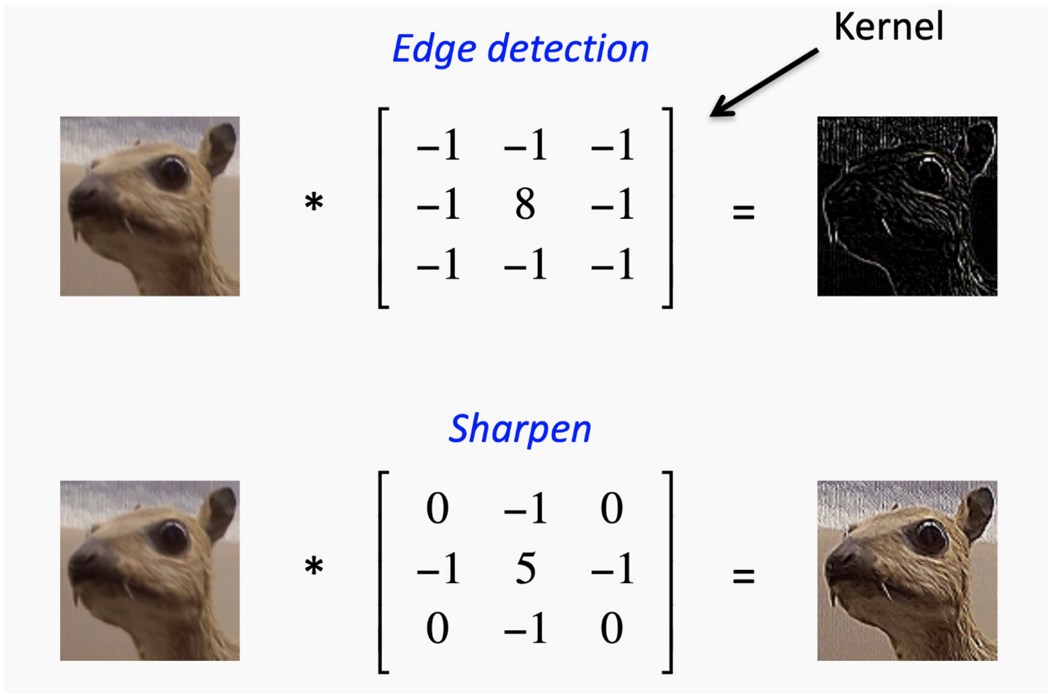
A filter could be related to anything, for pictures of humans, one filter could be associated with seeing noses, and our nose filter would give us an indication of how strongly a nose seems to appear in our image, and how many times and in what locations they occur. This reduces the number of weights that the neural network must learn compared to an MLP, and also means that when the location of these features changes it does not throw the neural network off.


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel

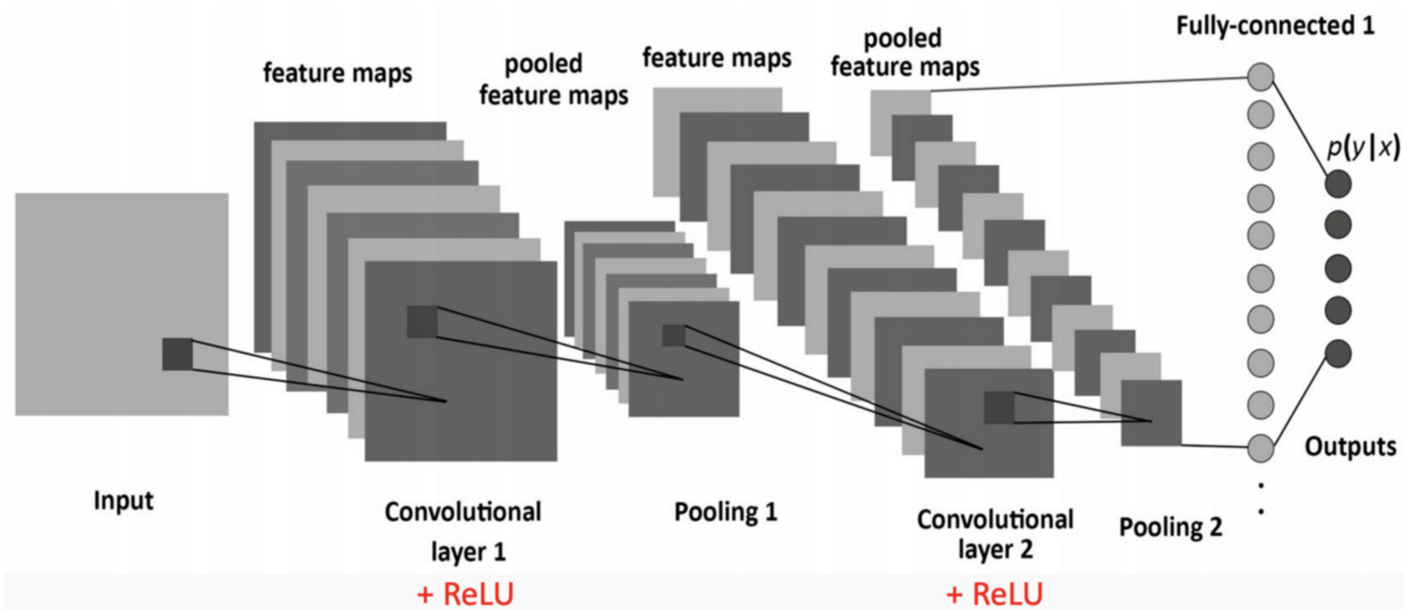
The convolution operation.

If you are wondering how the different features are learned by the network, and whether it is possible that the network will learn the same features (having 10 nose filters would be kind of redundant), this is highly unlikely to happen. 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 examples of filters, or kernels as we call them, are given below.


Edge detection
Kernel
$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$
Sharpen
$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$

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. We can also use pooling layers in order to select the largest values on the feature maps and use these as inputs to subsequent layers. In theory, any type of operation can be done in pooling layers, but in practice, only max pooling is used because we want to find the outliers — these are when our network sees the feature!



An example CNN with two convolutional layers, two pooling layers, and a fully connected layer which decides the final classification of the image into one of several categories.

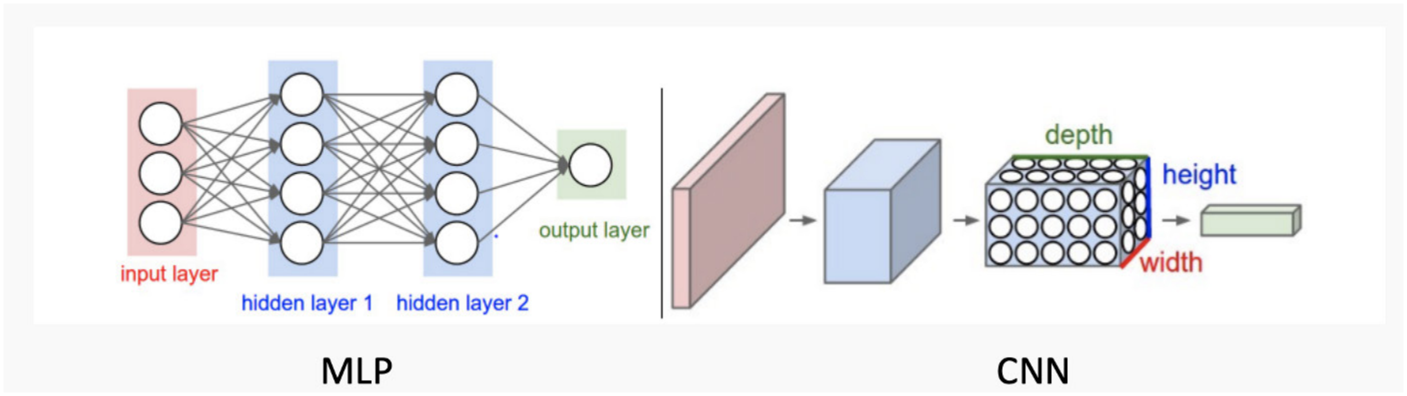
Just to reiterate what we have found so far. We know that MLPs:

- Do not scale well for images

- Ignore the information brought by pixel position and correlation with neighbors
- Cannot handle translations

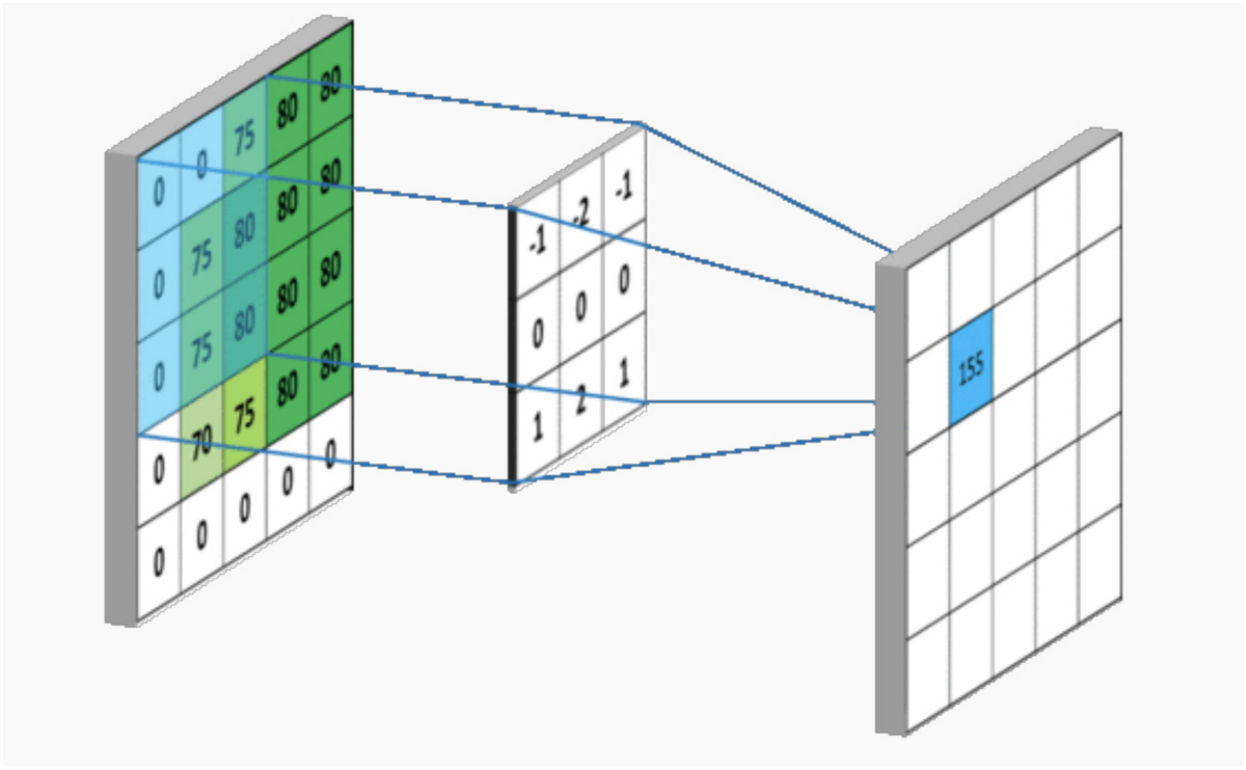
The general idea of CNN's is to intelligently adapt to the properties of images:

- Pixel position and neighborhood have semantic meanings
- Elements of interest can appear anywhere in the image



Comparison of architecture for MLP and CNN.

CNN's are also composed of layers, but those layers are not fully connected: they have filters, sets of cube-shaped weights that are applied throughout the image. Each 2D slice of the filters are called kernels. These filters introduce translation invariance and parameter sharing. How are they applied? Convolutions!



Example of how convolutions are applied to images using kernel filters.

A good question to have right now is what happens at the edges of the image? If we apply convolutions on a normal image, the result will be down-sampled by an amount depending on the size of the filter. What do we do if we don't want this to happen? We can use padding.

Padding

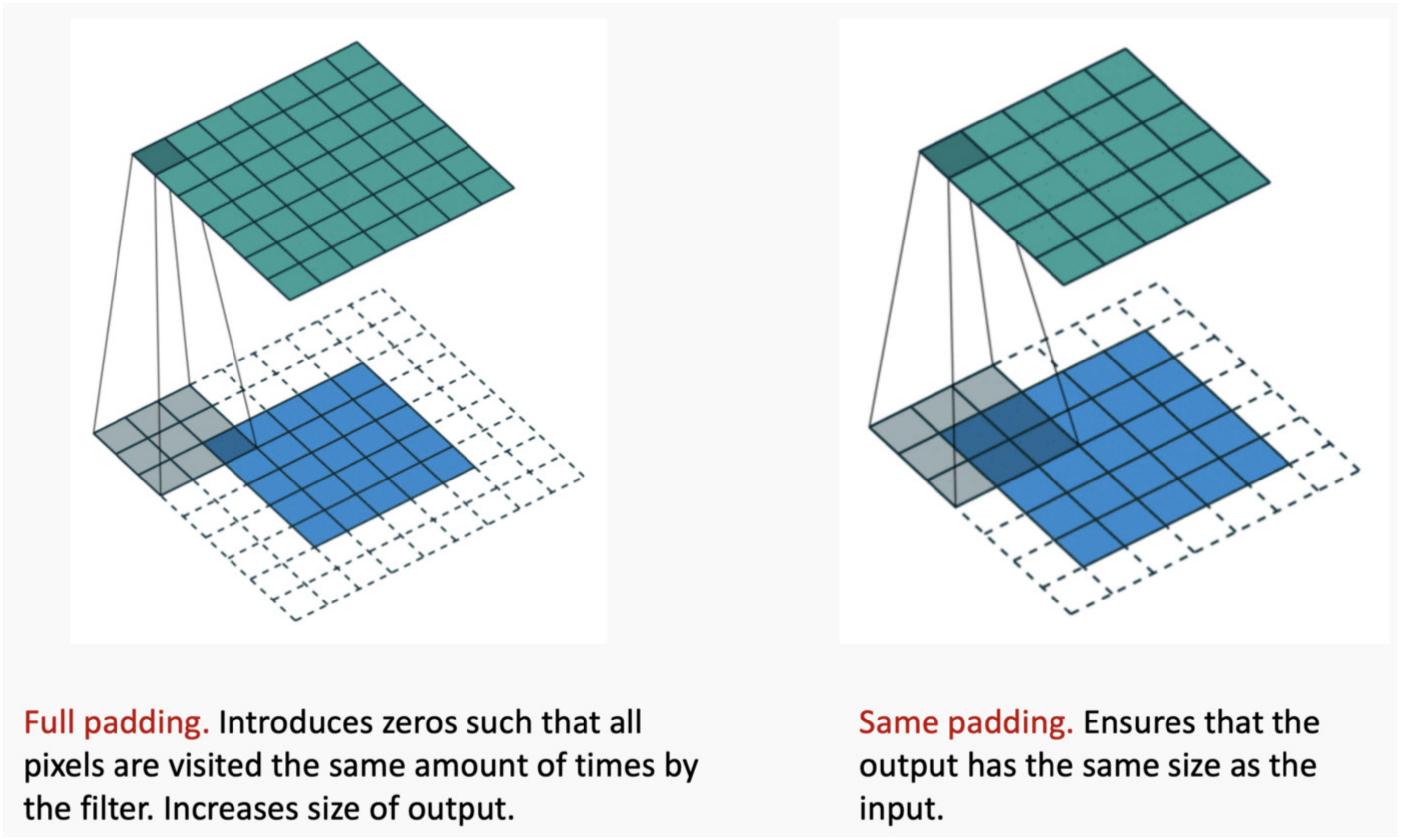
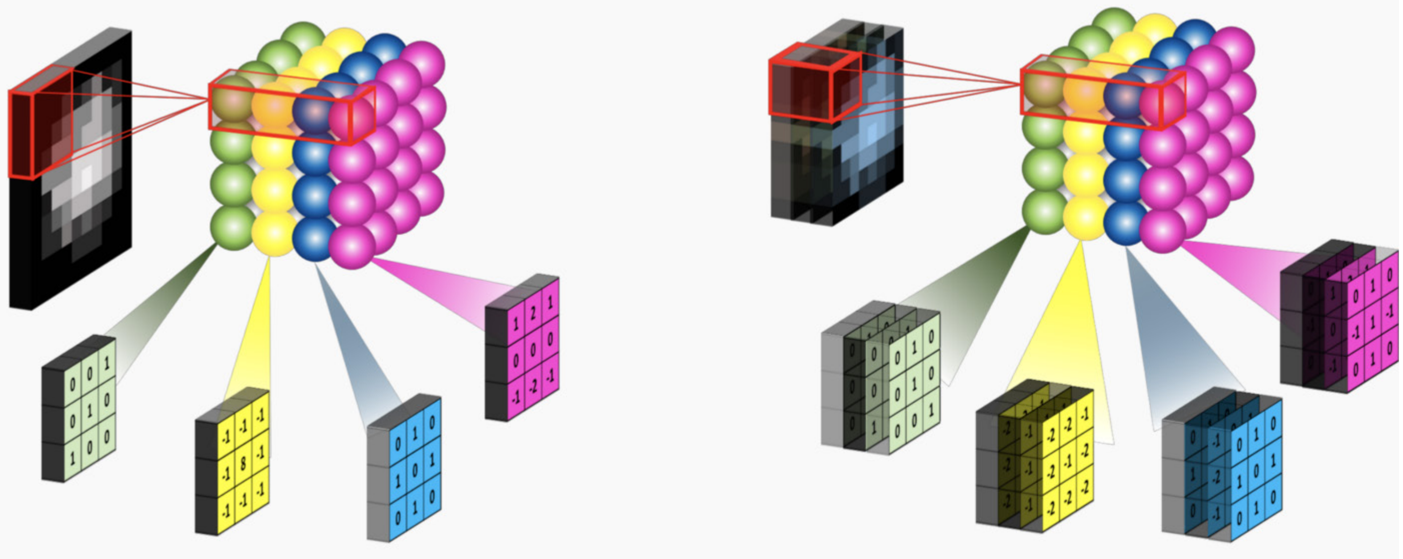

Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.
Same padding. Ensures that the output has the same size as the input.

Illustration of how full padding and same padding are applied to CNN's.

Padding essentially makes the feature maps produced by the filter kernels the same size as the original image. This is very useful for deep CNN's as we don't want the output to be reduced so that we only have a 2x2 region left at the end of the network upon which to predict our result.

How do we connect our filters together?

If we have many feature maps, how are these combined in our network to help us get our final result?



Convolutional layer with four 3x3 filters on a black and white image (just one channel)

Convolutional layer with four 3x3 filters on an RGB image. As you can see, the filters are now cubes, and they are applied on the full depth of the image..

To be clear, each filter is convolved with the entirety of the 3D input cube but generates a 2D feature map.

- Because we have multiple filters, we end up with a 3D output: one 2D feature map per filter
- The feature map dimension can change drastically from one convolutional layer to the next: we can enter a layer with a 32x32x16 input and exit with a 32x32x128 output if that layer has 128 filters.
- Convolving the image with a filter produces a feature map that highlights the presence of a given feature in the image.

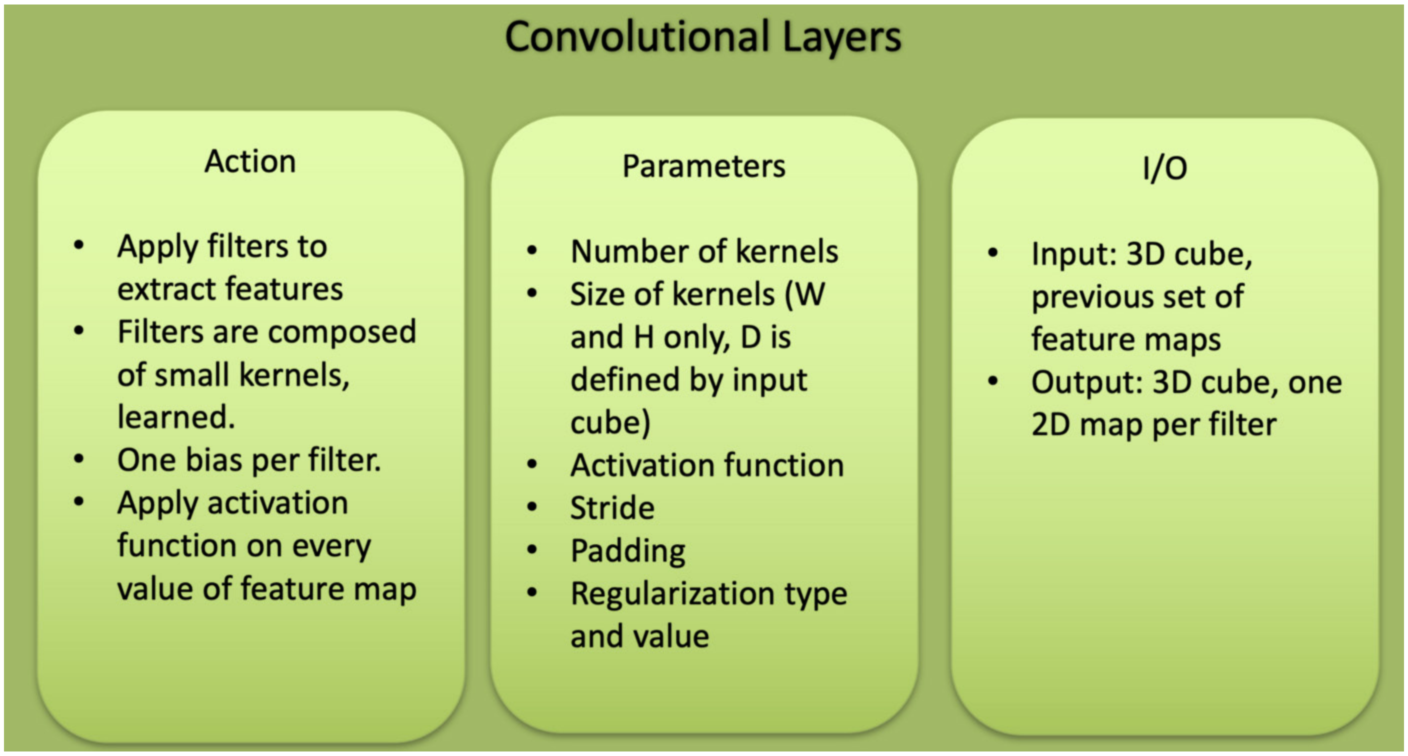
In a convolutional layer, we are basically applying multiple filters at over the image to extract different features. But most importantly, we are learning those filters! One thing we're missing: non-linearity.

Introducing ReLU

The most successful non-linearity for CNN's is the Rectified Non-Linear unit (ReLU), which combats the vanishing gradient problem occurring in sigmoids. ReLU is easier to compute and generates sparsity (not always beneficial).

Comparison of Different Layers

There are three types of layers in a convolutional neural network: convolutional layer, pooling layer, and fully connected layer. Each of these layers has different parameters that can be optimized and performs a different task on the input data.



Convolutional Layers

Action

- Apply filters to extract features
- Filters are composed of small kernels, learned.
- One bias per filter.
- Apply activation function on every value of feature map

Parameters

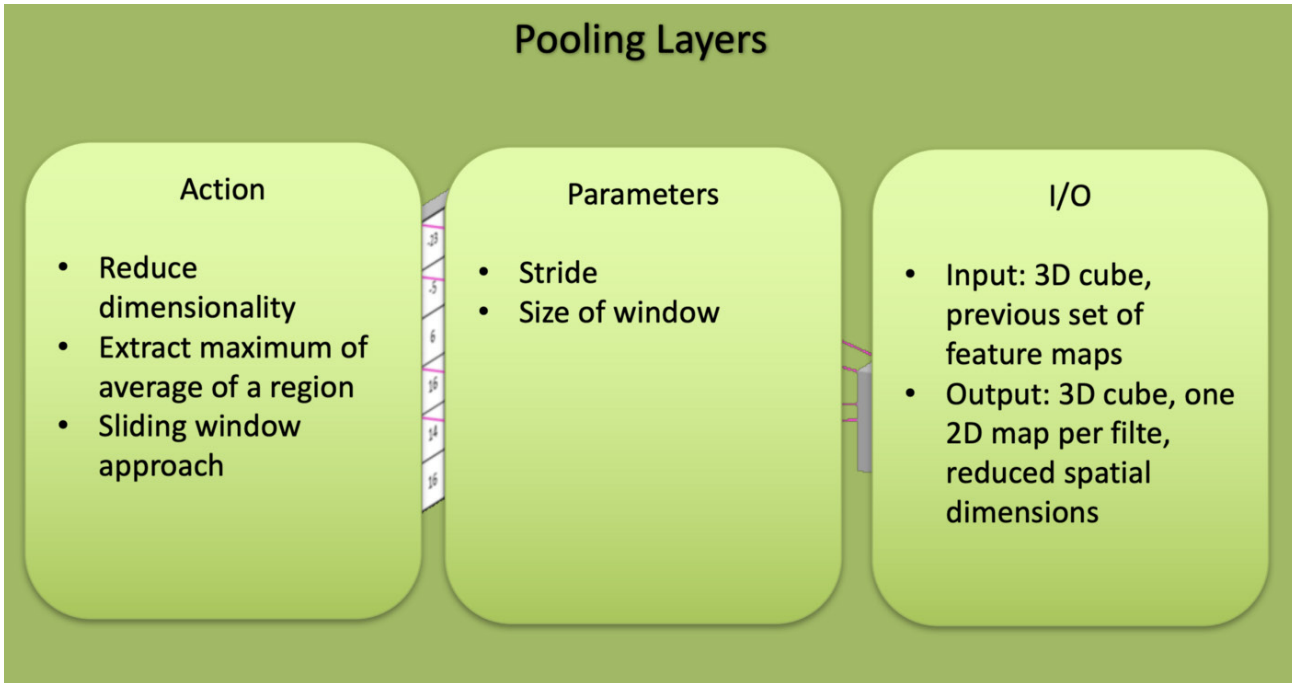
- Number of kernels
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding
- Regularization type and value

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Features of a convolutional layer.

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.



Pooling Layers

Action

- Reduce dimensionality
- Extract maximum of average of a region
- Sliding window approach

Parameters

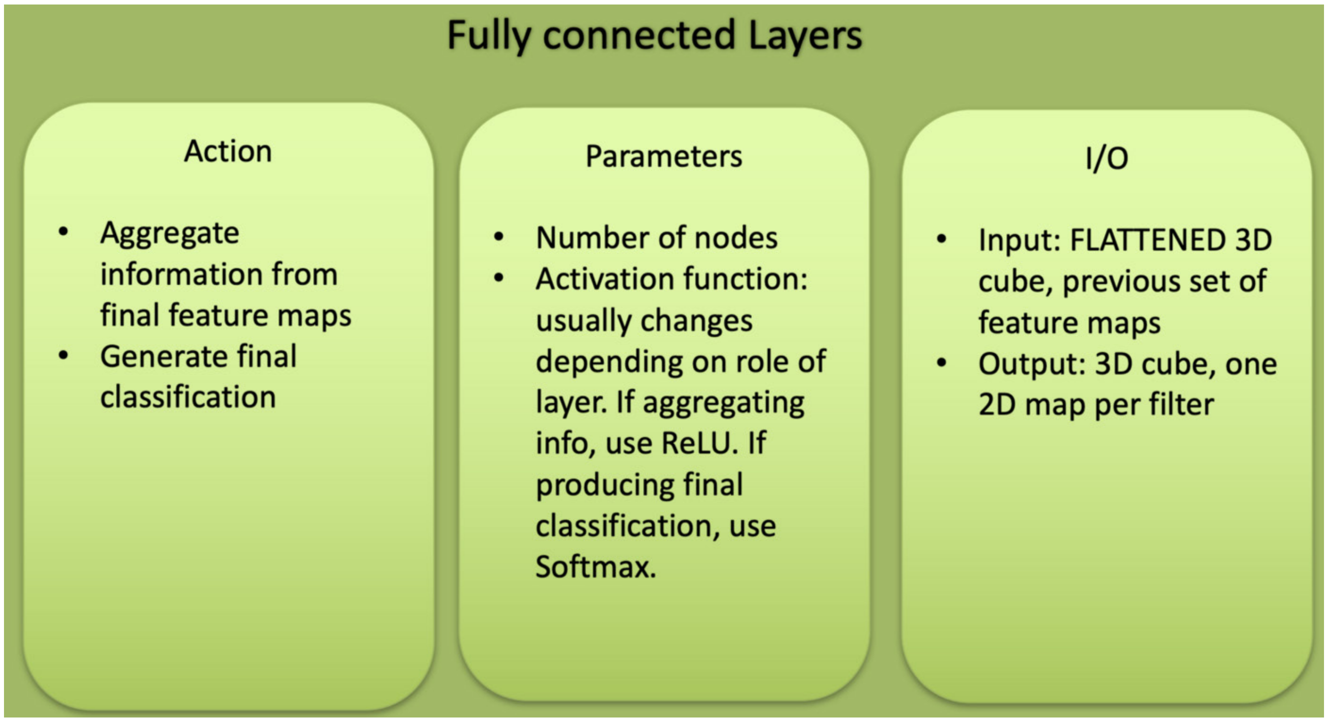
- Stride
- Size of window

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filte, reduced spatial dimensions

Features of a pooling layer

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.



Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification

Parameters

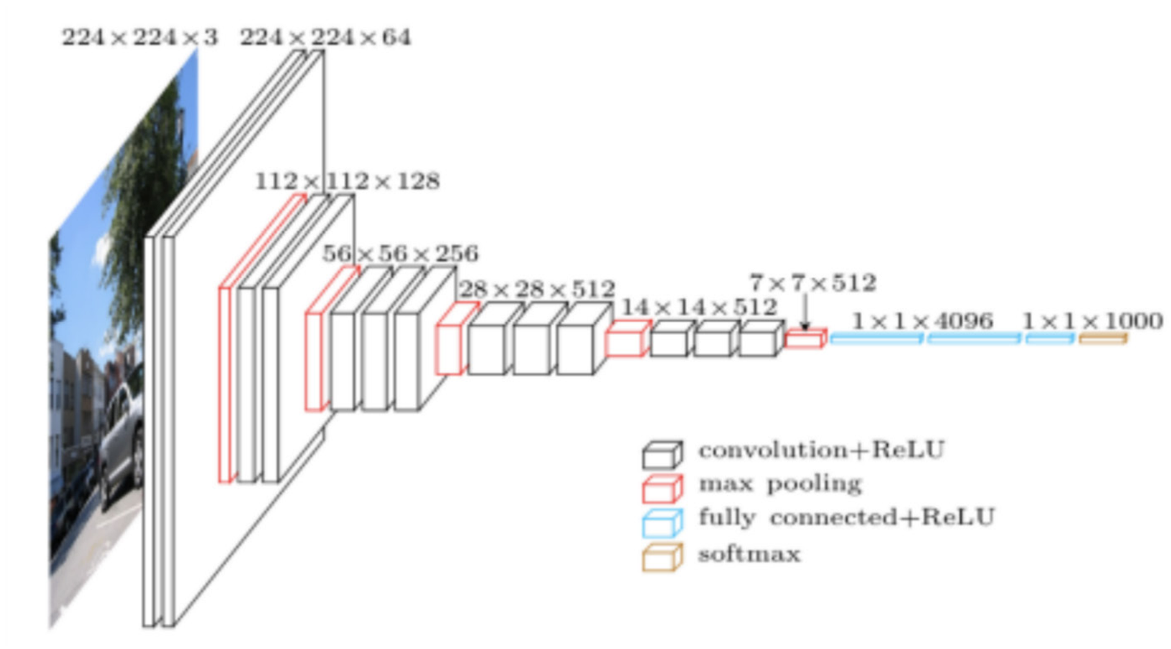
- Number of nodes
- Activation function: usually changes depending on role of layer. If aggregating info, use ReLU. If producing final classification, use Softmax.

I/O

- Input: FLATTENED 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Features of a fully connected layer.

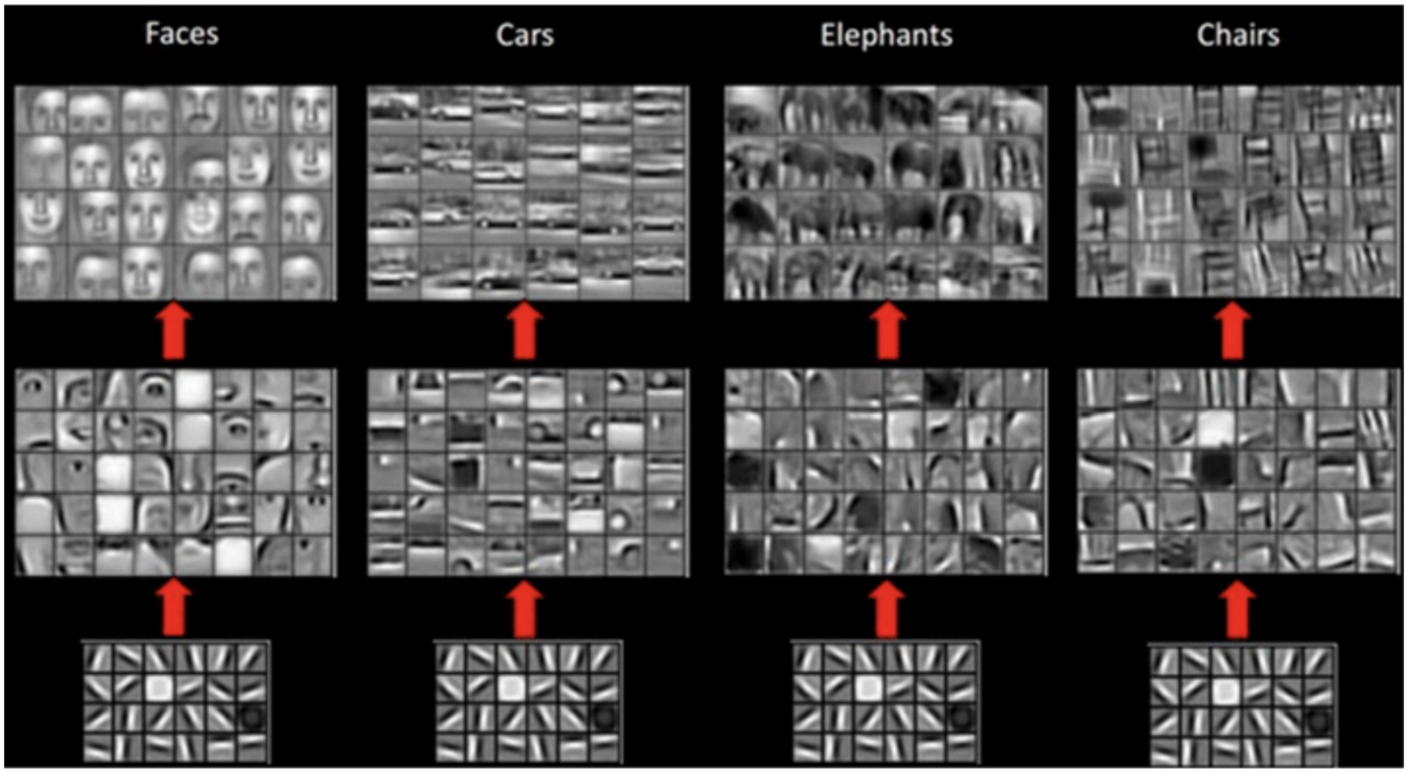
Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.



The architecture of a standard CNN.

What do CNN layers learn?

- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters: edges, corners, etc
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions



Examples of CNN's trained to recognize specific objects and their generated feature maps.

To see a 3D example of a CNN working in practice, check out the following link here.