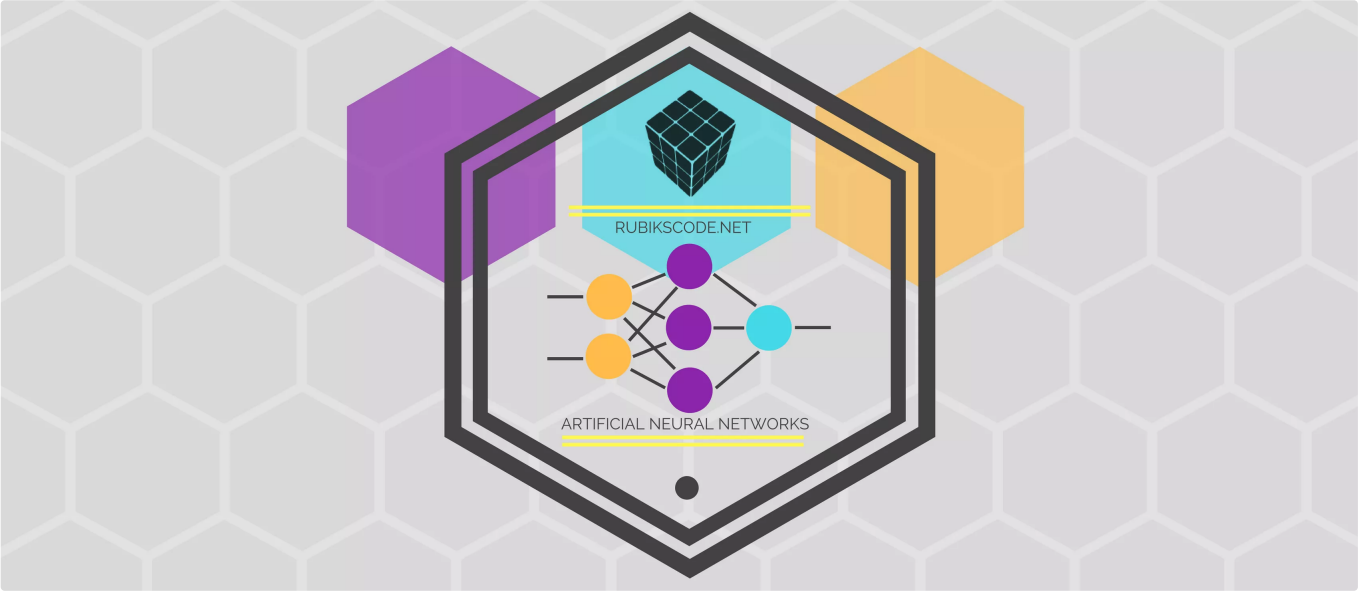
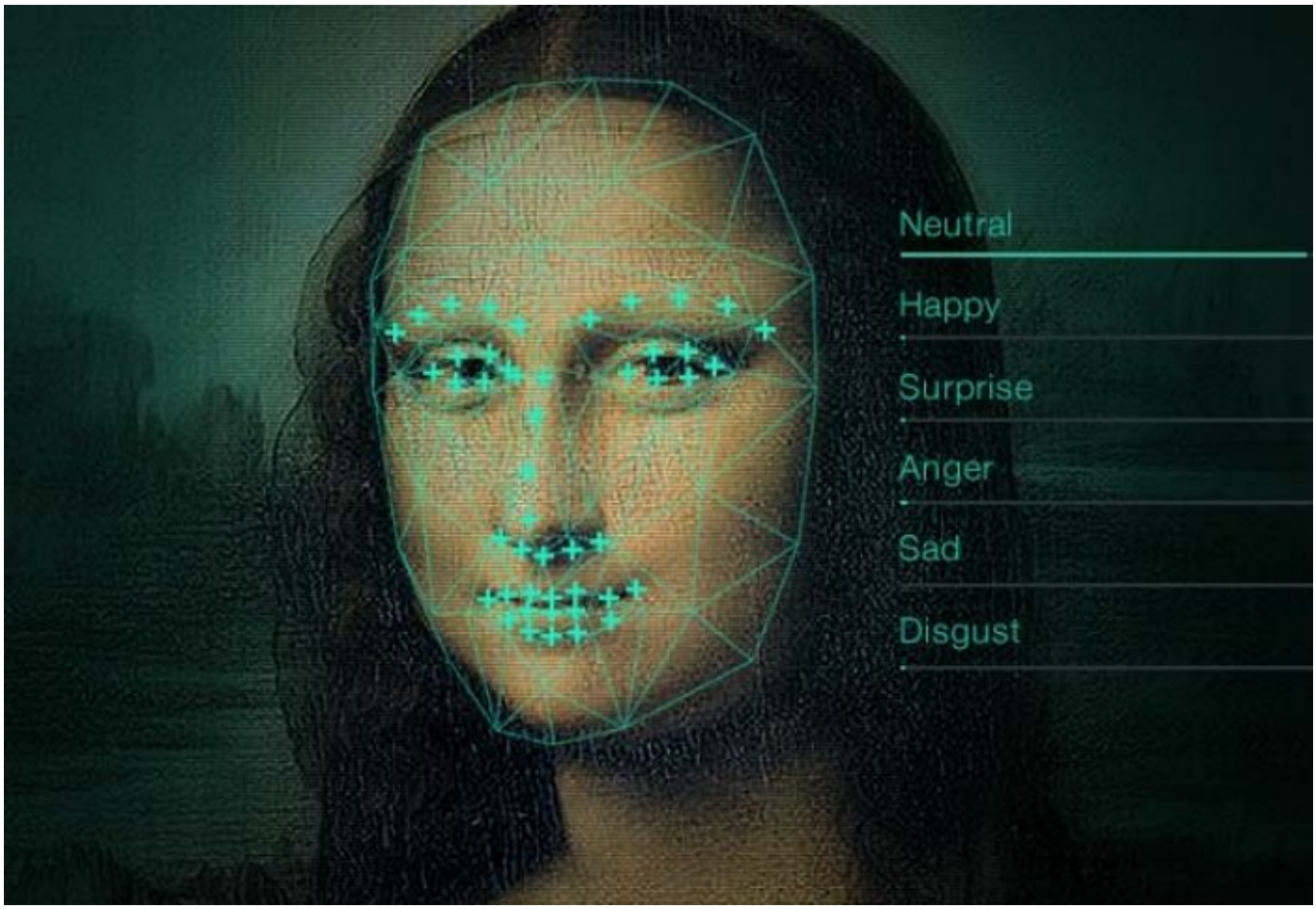
Introduction to Convolutional Neural Networks

FEBRUARY 26, 2018 — 9 COMMENTS



Have you ever wondered how Facebook knows how to suggest the right friend to tag? Speaking of it, how does the Google's image search algorithm work? Yes, you are right, there is a neural network involved in all those tasks. To be more precise, we are talking about Convolutional Neural Networks. Even though it sounds like a weird mixture of biology and computer science (everything related to neural networks kinda sound like that) this is one very effective mechanism used for image recognition. Of course, it is motivated by biological systems and the ways the brain works, specifically visual cortex.



Biology Behind the Idea

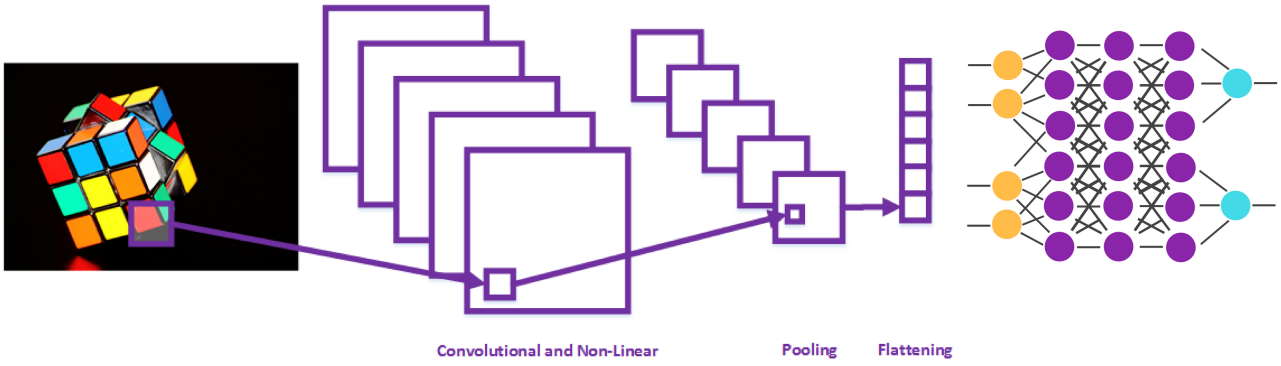
Individual neurons in this section of the brain respond to stimuli only in a restricted region of the visual field known as the receptive field. Because these fields of different neurons overlap, together they make the entire visual field. This effectively means that certain neurons were activated only if there is a certain attribute in the visual field, for example, horizontal edge. So, different neurons will be “fired up” if there is a horizontal edge in your visual field, and different neurons will be activated if there is, let’s say a vertical edge in our visual field. In a nutshell, if there is a certain feature in our visual field, specific neurons will be activated and other won’t. This was proven by a fascinating experiment done by Hubel and Wiesel in 1962. Results of this experiment can be checked out in **this video**.

So, how are Convolutional Neural Networks using this for image recognition? Well, they use this idea to differentiate between given images and figure out the unique features that make a plane a plane or a snake – a snake. This process is happening in our minds subconsciously. For example, when we take a look at the picture of a plane, we can identify it as a plane by distinguishing features like two wings, tale, windows, etc. Convolutional Neural Networks do the same thing, but they are first detecting lower level features like curves and edges and then they build it up to more abstract concepts.

Structure of Convolutional Neural Networks

In order to achieve the functionality we talked about, Convolutional Neural Network processes image through several layers. We will examine them in detail in next chapters of this article, but for now, let's just do an overview of them and their purposes:

- Convolutional Layer – Used to detect features
- Non-Linearity Layer – Introducing non-linearity to the system
- Pooling (Downsampling) Layer – Reduces the number of weights and controls overfitting
- Flattening Layer – Prepares data for Classical Neural Network
- Fully-Connected Layer – Standard Neural Network used for classification



Basically, in the end, Convolutional Neural Network uses standard Neural Network for solving classification problem, but it uses other layers to prepare data and detect certain features before that. Let's dive into details of each layer and their functionalities.

Convolutional Layer

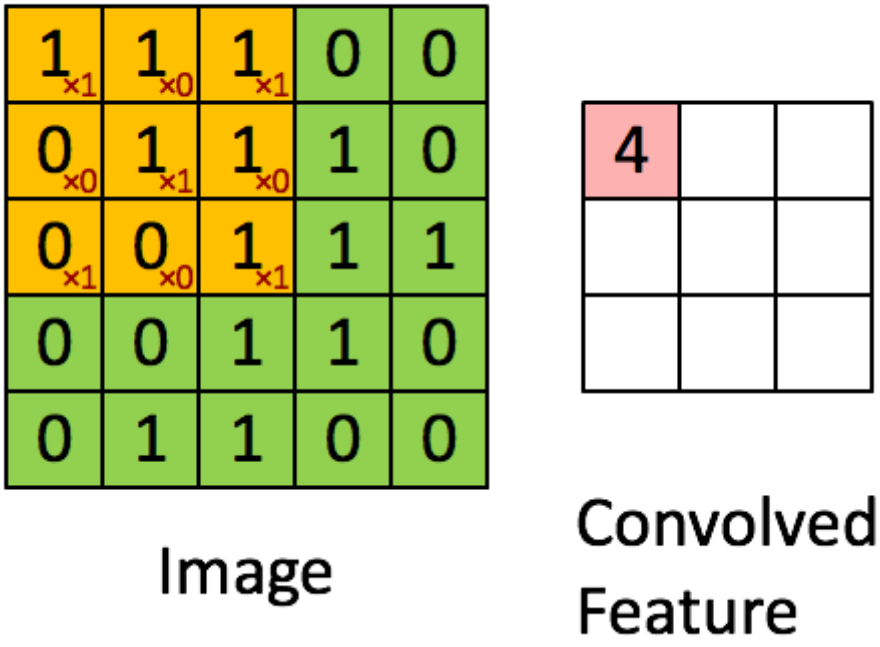
This is the main building block of Convolutional Neural Networks. It is doing the heavy lifting without which the rest of the activities would be impossible. As mentioned already, it is in charge of detecting features. This is done by applying a certain filter to the image that will extract some of the low level and later higher level of attributes on some image. For example, we can use a filter that will detect edges. A filter is usually a multi-dimensional array of pixel values – for example, 5x5x3. Numbers 5 are used to present height and width in pixels, while number 3 is used to present depth because images have three color channels. So, how do we apply the filter? Take a look at this image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

We are considering just one channel of this image for the demonstration purposes. It is the 5×5 image where each pixel has value 1 or 0. We will use the 3×3 filter that is simple and it looks like this:

1	0	1
0	1	0
1	0	1

Now take a look how this process of applying the filter, also known as convolution, is done:



Convolution process – Source

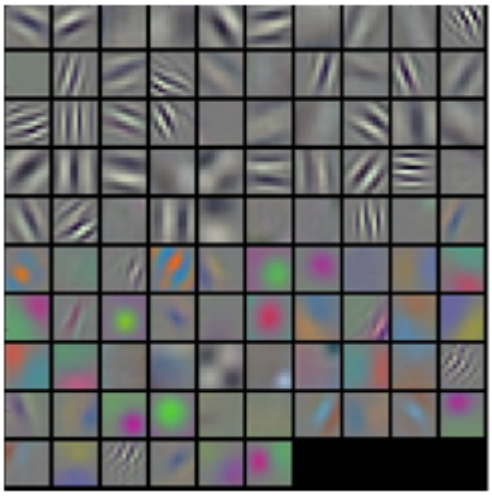
Firstly, we position the filter in the first location of the image, top left corner. There we use element-wise multiplication between two matrixes and store the result to the output matrix. Then we move that filter to the right by 1 pixel (also known as “stride”) and repeat the process. We do that as long as we can move our filter to the right. When we cannot do that anymore, we

go to the next row and apply the filter in the same way. We do this until our output matrix is not complete. The reason our output matrix is 3×3 is that there are 9 positions in which you can fit the 3×3 feature in the 5×5 image. These 9 numbers are mapped to the 3×3 matrix.

What is our output matrix telling to us? This matrix is often called Feature Map. It indicates where the feature, represented by the filter, is located in the image. In a nutshell, by moving the filter through the image and using simple matrix multiplication, we detect our features. Usually, we use more than one filter and detect multiple features, which means that we have more than one convolutional layer in the network. Take a look at the animation below, where this process is a little bit more visual:

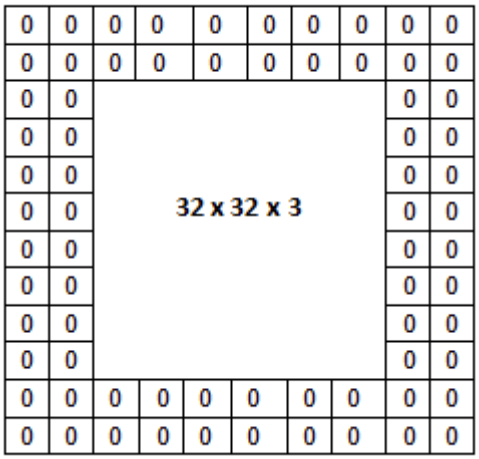
Convolution process – Source

When we apply the first filter we are creating one Feature Map and detect one kind of feature. Then we use the second filter to create a second Feature Map that detects another kind of feature. These filters can be simple as we could see in the example, but they can get quite complicated if we want to extract some complex features from the image. Take a look at the image below, where multiple filters are represented.



Filters – Source

Another thing that we already mentioned, but didn't explain in detail is stride. This term is usually used in combination with the term padding. Stride controls how the filter is convolved around the input image. In the example above, stride was 1 pixel, but it can be larger. This affects the size of our output, of our Feature Map. At the early stages of the Convolution Neural Network, when we are applying our first layers we want to preserve as much information as possible for other Convolutional Layers. That is why padding is used. You may notice that Feature Map is smaller than the original input image. Padding would add zero values to this map to preserve the size, like on the image below:



Padding example

Non-linearity

After every convolutional layer, we usually have a non-linearity layer. Why is linearity in the image a problem? The problem is that our Neural Network would behave just like a single perception, because the sum of all the layers would still be a linear function, meaning the output could be

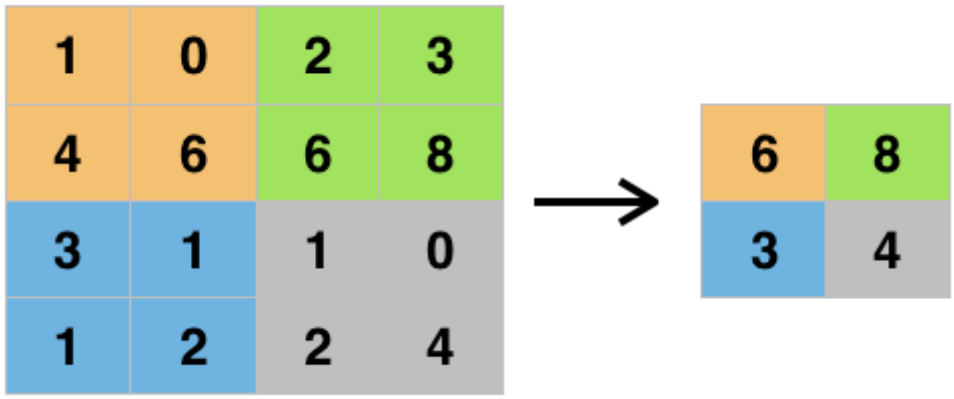
calculated as the linear combination of the outputs. This layer is also called the activation layer because we use one of the **activation functions**. In the past, nonlinear functions like sigmoid and *tahn* were used, but it turned out that the function that gives the best results when it comes to the speed of training of the Neural Network is **Rectifier function**. So, this layer is often ReLU Layer, that removes linearity by setting values that are below 0 to 0 since Rectifier function is described with the – $f(x) = max(0, x)$. Here is how it looks once applied to one of the feature maps:

On the second image, the feature map, black values are negative ones, and after we apply the rectifier function, black ones are removed from the image.

Pooling Layer

This layer is commonly inserted between successive convolutional layers in Convolutional Neural Networks. The functionality of this layer is to reduce the spatial size of the representation and with, that the number of parameters and computation in the network. This way we are also controlling over-fitting in our network. There are different pooling operations, but the most popular one is called max pooling and we will examine it in this article. The other pooling algorithms, like average pooling, or L2-norm pooling, work on the same principle, one which we will examine shortly.

Here we will have a sort of a filter once again. Take a look at the picture below. We used the maxpooling filter size 2×2 on the 4×4 image. As you already guessed, filter picks the largest number of the part of the image it covers. This way we end up with smaller representations that contain enough information for our Neural Network to make correct decisions.

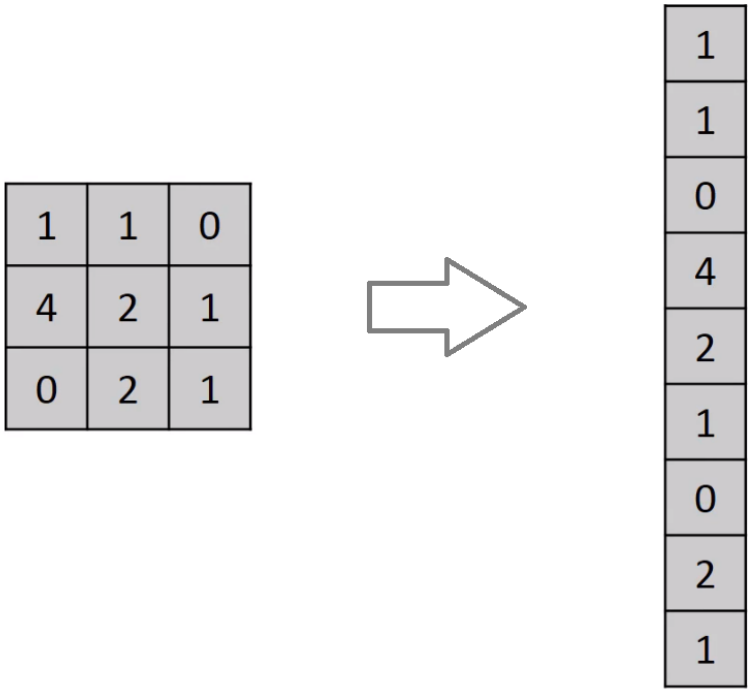


Maxpooling process

However, a lot of people is against pooling layers, and replace them with additional Convolutional Layers with a larger stride. Also, newer generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs), discard pooling layers completely. It seems that this layer will soon be obsolete.

Flattening Layer

This is a simple layer that is used to prepare data to be the input of the final and most important layer – Fully-Connected Layer (which is our Neural Network). Since in general, neural networks receive data in one dimension in a form of an array of values, this layer uses data passed from pooling layer or convolutional layer and squashed the matrixes into arrays. Then, these values are used as an input to the neural network. Here is the visual representation of the flattening process:



Flattening data

Fully-Connected Layer

The final layer and the layer that does the actual classification is the so-called Fully-Connected layer. This layer takes input from the flattening process and feeds and forwards it through the Neural Network. What we have done basically, is that we fed this network with the detection of the features. If you need more information of how neural networks work you can check previous articles from **this series**.

Architectures of Convolutional Neural Networks

A common way of building Convolutional Neural Networks is to stack a few Convolutional Layers and after each of them add ReLU layer. After that, they are followed by pool layers and the flattening layer. Finally, several Fully-connected layers along with additional ReLU layers are added. Keep in mind that in the end there should always be fully-connected layers, which looks something like this:

Input Image -> [[Conv -> ReLU]*N -> Pool?]*M -> Flattening -> [FC -> ReLU]*K -> FC

Some of the architectures in the field of Convolutional Networks are quite famous and have a name:

- **LeNet** – This was the first successful application of Convolutional Networks. It was developed by Yann LeCun in 1990's and it was used to read zip codes, simple digits, etc.
- **AlexNet** – This was the network that was presented in the ImageNet ILSVRC challenge back in 2012. It is actually the network that popularized the Convolutional Networks since it outperformed all other contestants by far. It was developed by developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton.
- **GoogLeNet** – The winner of the ILSVRC 2014 winner was a Convolutional Network from Google. They used average pooling layers to dramatically minimize the number of parameters in the network. There are several follow-up versions to the GoogLeNet.
- **VGGNet** – Convolutional Neural Network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. This network proved that depth of the network that is crucial for good performances. It has 16 convolutional layers.
- **ResNet** – Developed by Kaiming He et al. was the winner of ILSVRC 2015. You can watch **this cool video** in which topic is described in depth.

Conclusion

In this article, we covered quite the ground. We examined biological motivations behind this type of Neural Networks as well as the mechanisms they work on. There are a lot of topics in here that we just scratched the surface of, since the field is wide and somewhat complicated. Still, it is a nice starting point for going deeper into the wide world of Convolutional Neural Networks.

Thanks for reading!

This article is a part of Artificial Neural Networks Series, which you can check out **here**.

Read more posts from the author at **Rubik's Code**.
