Chapter 9 : Part 2

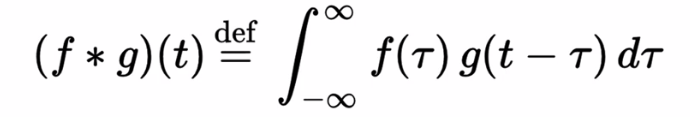
**Deep Learning**

**CNN: Convolution Operations**

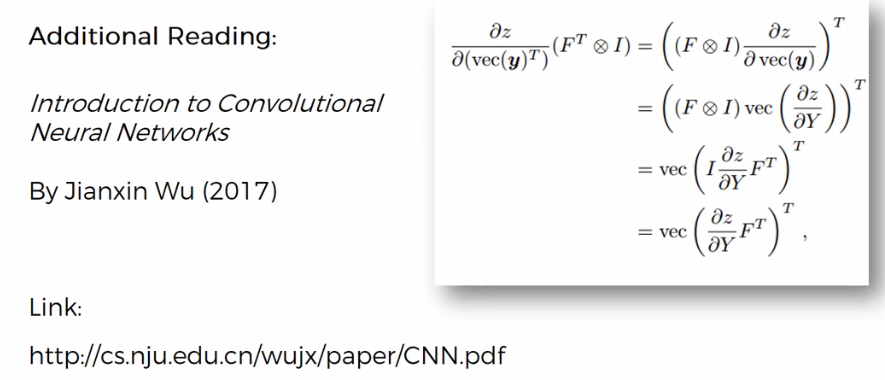
**& ReLU layer**

**9.2.1 Convolution**

* *Convolution:* Following is the ***convolution*** ***function***. A ***convolution*** is basically a combined integration of the two functions and it shows you *how one Function modifies the other* or (modifies the shape of the other). If you've done any Signal Processing or Electrical Engineering then you are already familiar with it.

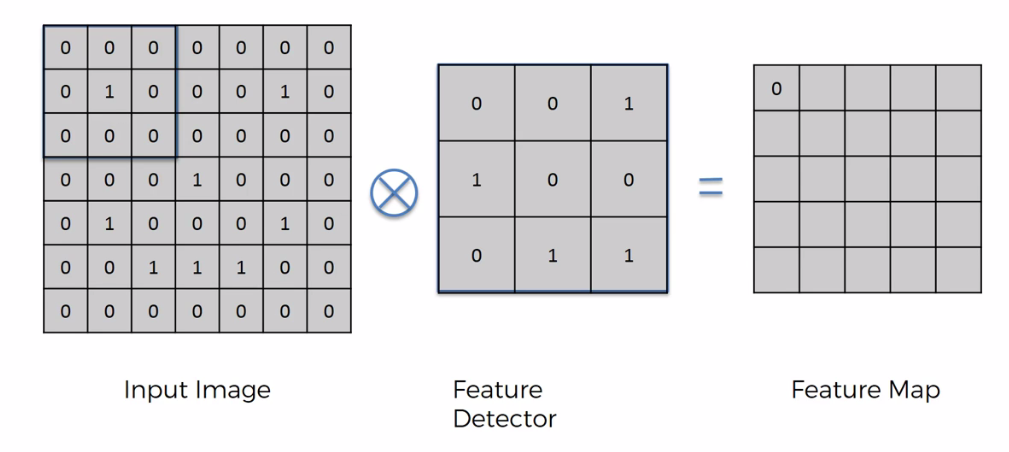


* Additional Reading: If you want to know the mathematics of ***convolution*** ***function*** and CNN read the article below. is called *"Introduction to convolutional neural networks"* by Jianxin Wu who is a professor at Nanjing University in China. It is oriented specifically at people who are beginners to know CNN so the mathematics there should be accessible.

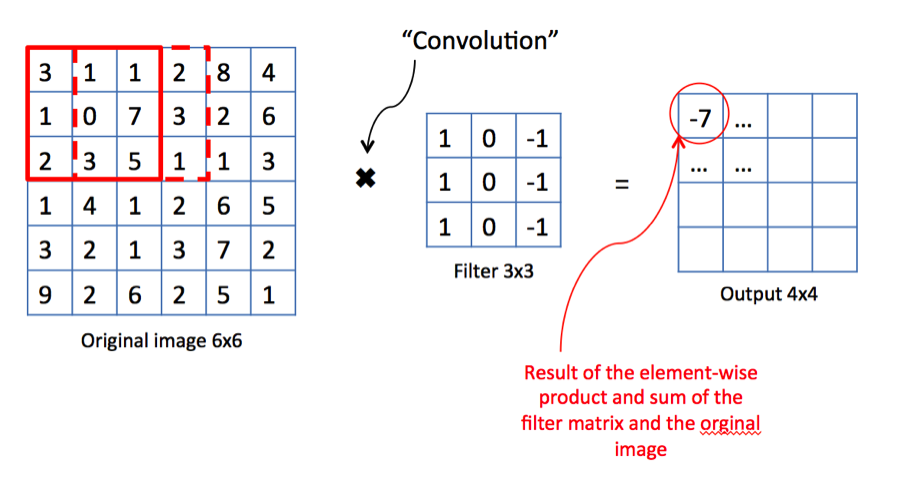


<https://cs.nju.edu.cn/wujx/index.htm>

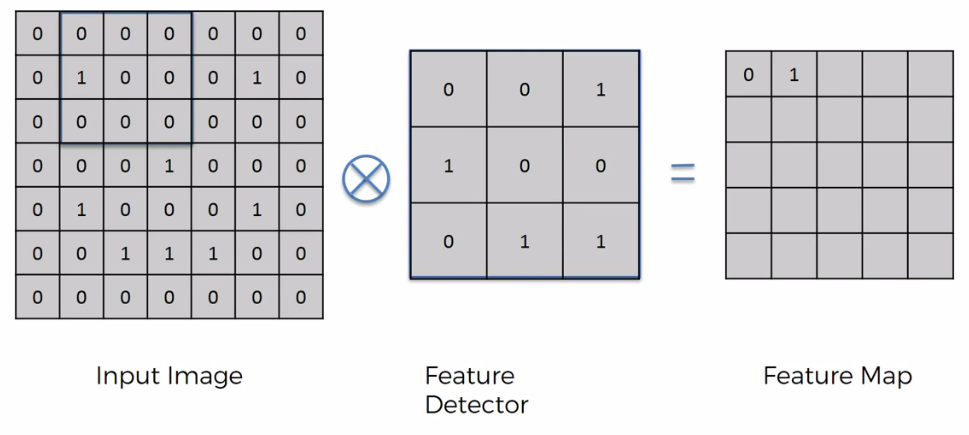
|  |  |
| --- | --- |
| * Convolution in intuitive terms: Here we got an input image, just ***ones*** and ***zeros*** to simplify things. You can see the smiley face there. * Feature Detector: We've also got a feature detector. This feature detectors a Matrix. However it could be any size. Also the feature detectors called ***Kernel*** or you might hear it being called ***Filter***. We're going to be using either ***filter*** or a ***feature*** ***detector*** interchangeably. And a ***Convolution*** ***operation*** is signified by ***"x in a circle"***. |  |



* How Feature Detector works: Here we are going to see what is actually happening in the background rather than the mathematics.
* You take the ***feature*** ***detector*** or ***filter*** and you put it on your image (like you see on the left). For instance in above case the top left corner-nine pixels and you basically multiply *each* *value* by *respective value*. It's not the matrix multiplication, its actually ***"element-wise multiplication and sum of the values"***. So
* Actually we are looking for how many matches are found in ***"Input image"*** w.r.t. ***"Feature Detector"***. So in this first case nothing matches up (it's always either or by ) so the result is ***0***.
* Feature detector can also contain negative values:



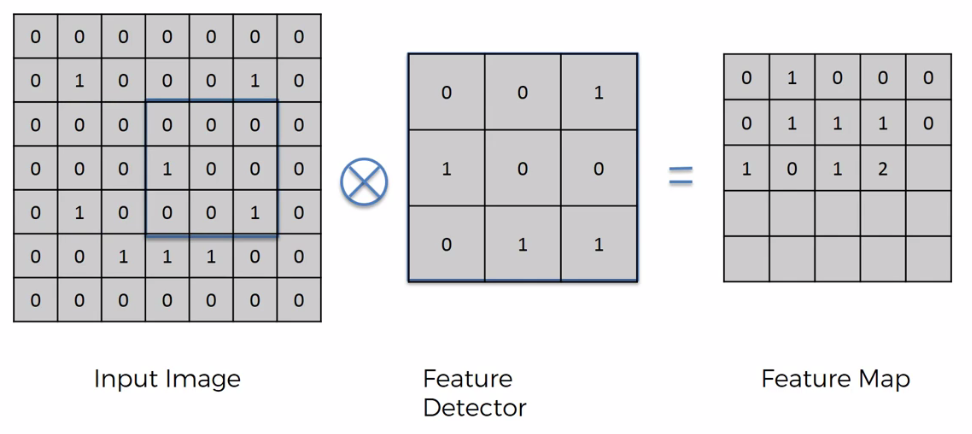
* If we shift 1px to right we can see that one of the 1 is matched up And therefore we've got a 1 here.



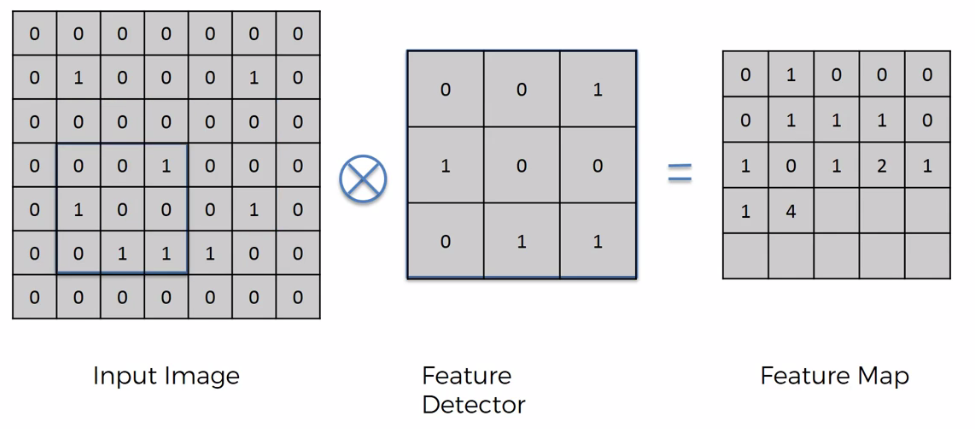
Hence we get the followings

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

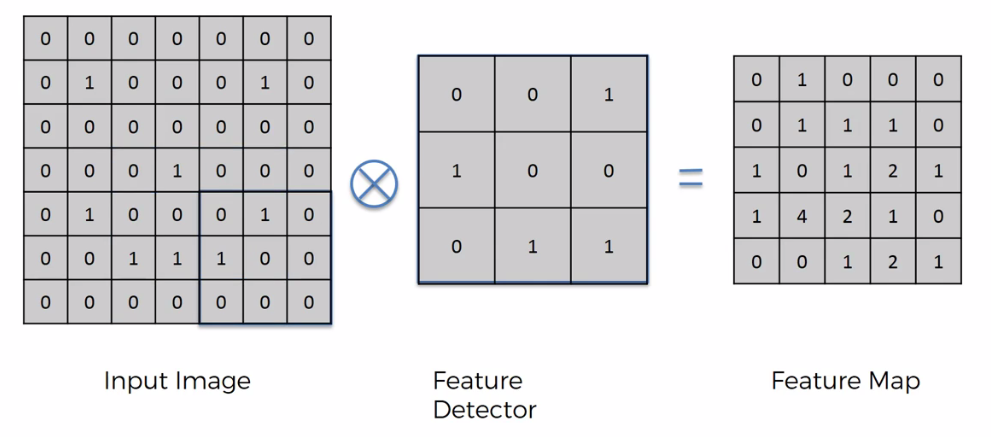
* Notice that, following has matching ***two 1's*** hence ***sum*** is 2.



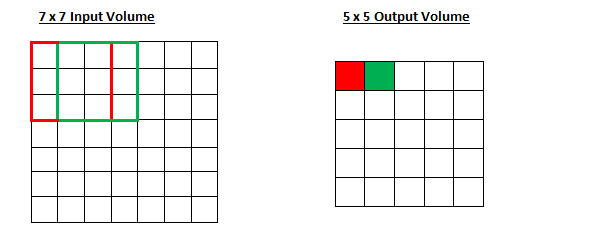
* Also notice that we got All Matching four 1's. Hence the sum is 4.



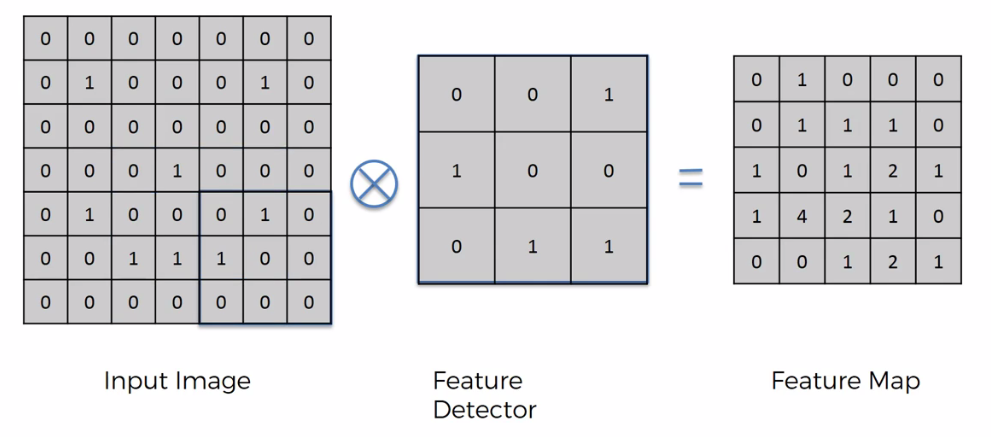
* Finally we got the following ***feature map***.



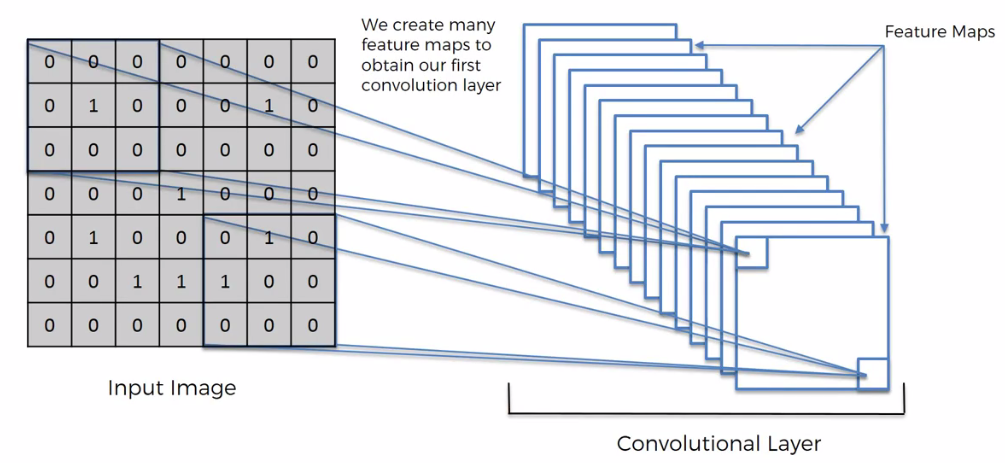
* Stride: *Steps* at which we're moving this whole filter is called the stride. So here we have a stride of one pixel. Stride is a component of CNN, or neural networks tuned for *the Compression of Images* and *Video* data. Stride is a ***parameter*** of the neural network's ***filter*** that modifies *the* ***amount of movement*** *over the image* or *video*. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.



* Imagine a convolutional neural network is taking an image and analyzing the content. If the filter size is 3x3 pixels, the contained nine pixels will be converted down to 1 pixel in the output layer. Naturally, as the stride, or movement, is increased, the resulting output will be smaller.
* Feature map: The image on the right is called a ***feature*** ***map*** also called ***convolved*** ***feature*** can also be called the ***activation*** ***map***. So when you apply ***convolution*** ***operator*** to something, it *doesn't* become *convoluted* it becomes ***convolved***.



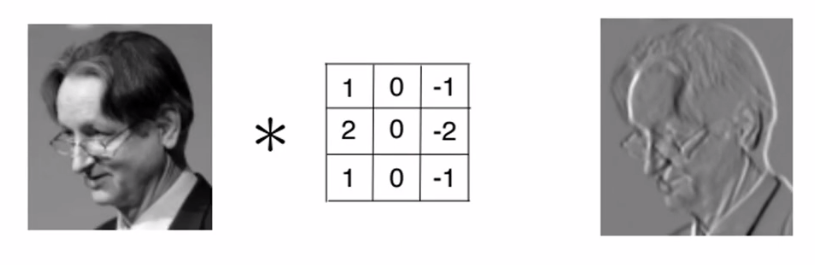
* Here we've actually reduced the size of the image. If you have a stride of one you can see the image reduced a bit, but if you have stride of two the image is going to reduce more, so the ***feature-map*** is going to be even smaller.
* The purpose of this whole convolution step is to make the image smaller, so that, it'll be easier & faster to process it. Therefore feature detectors will *reduce* the *size* of the image and therefore stride of two is actually beneficial.
* Information loss: Some information we are losing because we have less values in resulting matrix. But at the same time the purpose of the feature detector is to detect certain features at certain parts of the image that are integral.
* For instance if you think about it this way: the *feature* *detector* has a certain *pattern* on it, the highest number in your feature map is when that *pattern* *matches* up. As in our case it is 4 the highest value, because feature-detector fully matched.
* It helps us to ***focus*** on the ***features***, as for detect a "***Cheetah***" we look at its "***eye mark***". We don’t consider all of the pixels.
* Multiple FEATURE MAPS with different FILTERS/FEATURE-DETECTOR: For our input image, and we create multiple feature maps because we use different filters/feature-detector. That's another way that we preserve lots of the information. (Kind of extracting features).
* Here we don't just have one feature map. Basically we want to detect features. Each *layer* has its *own* *feature* *map* for certain *features*.
* When we look for *certain* *features,* the *network* *decides* through its *training* which features are important for certain types/categories and it looks for them using different *filters/feature-detectors*. It'll apply these *filters/feature-detectors* so to get certain *feature* *map*.
* That's why the term Feature Detector is better than the term Filters.



|  |  |  |
| --- | --- | --- |
| * Lets see some example of filters for an image app. We are using ***Gimp.org***. Here we have a picture of the ***Taj Mahal*** and you can choose ***which*** ***filter*** you want to apply. So if you download this program and you upload a photo into it and then you can actually start a Convolution-matrix and apply filters. Following are some commonly used filters.      * Sharpen: If we apply this filter, we can see that it sharpens the image. 5 in the middle as main pixel and four -1's. Kind of reduces the pixels around. | |  |
| * Blur: All six 1's. Basically takes equal significant to all of the pixels and therefore it combines them together and you get a blur. |  | |

|  |  |
| --- | --- |
| * Edge Enhance: So here you can see that's -1 and 1 and then you get all 0's (i.e. remove the pixels around the main one). It gives you an edge. |  |
| * Edge Detect: So here we are detecting edge. You reduce the middle one as -4. And use 1's around it. |  |
| * Emboss: So the key point here is that this matrix is asymmetrical and you can see the image becomes asymmetrical. |  |

* So we can see these are great examples of the same image but we're getting different ***feature maps*** using different ***filters/feature-detectors***.
* *But these terms:* Emboss/Sharpen are not applicable to us (we are not working with Gimp). We are not going to use those terms in CNN. However, ***Edge-Detect*** is quiet important for CNN but not the others.
* CNN will decide for itself what's important what's not and it probably won't be even recognizable to the human eye. You won't be able to understand what those features mean.
* That's the beauty of CNN. They can process so many different things and understand without even having that explanation *why they will understand which features are important to them* whether we have a term/name for those features or not.

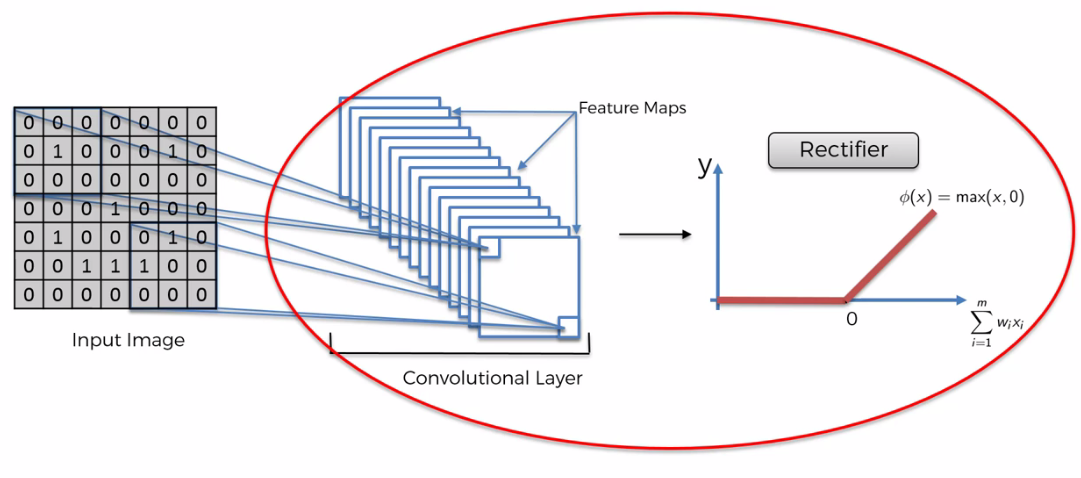


Here's a image of Geoffrey Hinton passed through one of these filters.

* The key-point is: the primary purpose of ***Convolution*** is to find ***features*** in your image using the ***feature detector*** put them into a ***feature map*** by preserving the spatial relationships between pixels (if they are completely jumbled up then we've lost the pattern).
* It's also important to understand that most of the time the ***features*** a neural network will detect/use to recognize certain images and classes, it will mean nothing to humans.

**9.2.2 Rectified Linear Unit (ReLU)**

ReLU (Rectified Linear Unit) is an *additional step* on top of our ***Convolution*** step. Here we have our input image and we have all Convolution Layer then on top of that we're going to apply rectifier function. The reason why we're applying the rectifier we want to Increase Non-Linearity in our image or in our CNN. And rectifier acts as that filter or that function which breaks up that linearity.



* Why increase Non-linearity: We want to increase nonlinearity in our network is because *images* themselves *are* highly *non-linear* especially if you're *recognizing* different *objects* *next to each other* .
* Images are going to have lots of *nonlinear elements* and the *transition* between *pixels-adjacent pixels* is often would be nonlinear, because of its borders, different colors, different elements in your images.
* When we're applying a *mathematical* *operation* such as *convolution*, and running this *feature detection* to create our *feature* *maps* there is a possibility that we might create something *linear* and therefore we need to break up the linearity.

|  |  |
| --- | --- |
| * Example: Here is a image an original image. Now when we apply a ***feature detector*** to this image we get following: |  |
| * You can see that *black* is *negative* and *white* is *positive*. When you apply a feature detector to a real-life image, (not just 1 or 0 but lots of different values, including negative values) we get this kind of result. |  |

* Now a Rectified Linear Unit (ReLU) function removes all the black (anything below zero it turns into zero).



* Here it's pretty hard to see what exactly is the benefit in terms of breaking up linearity. This is a very mathematical concept and would have to go into a lot of math to really explain what is going on. Now we try to figure out without math.
* What is LINEARITY in here: For instance consider the shadows of the white building. In *black-negative* and *white-positive* version notice the light. You see that it's White (the reflection of the light) and then it's a Gray and then it gets Darker and then it gets Darker again. It looks like when you go from White to Gray the next step would be Black. So it's a Linear Progression from Bright to Dark and therefore this is kind of ***linear*** situation.
* When you take out the ***black*** you ***break up*** the ***linearity***. There won't be any gradual progression like ***White-Gray-Black***.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Real image | Linear: Bk = -ve, W=+ve | Non-Linear: only +ve |

* Like having an *abrupt* *change*, which helps to introduce *non-linearity* into your *image*.
* Additional reading: But if you'd like to learn more there's a good paper. This one is by C.C. Jay Kuo from the University of California and it's called *"Understanding Convolutional neural networks with a mathematical model"*.
* There he answers this question: Why a *nonlinear activation function* is essential at the *filter* *output* of all *intermediate* *layers*. It explains it in a bit more detail both in terms of intuition and mostly in terms of ***mathematics***.



* And if you really want to dig in and explore more. Then there's another paper that you might be interested in. It's called ***"Delving Deep Into Rectifiers: Surpassing Human Level Performance on ImageNet classification"*** by Kaiming He from ***Microsoft*** ***Research***.
* They proposed a different type of ReLU function which you see here on the right. Which is parametric ReLU function. And they argue that it delivers better results Without Sacrificing PERFORMANCE.

