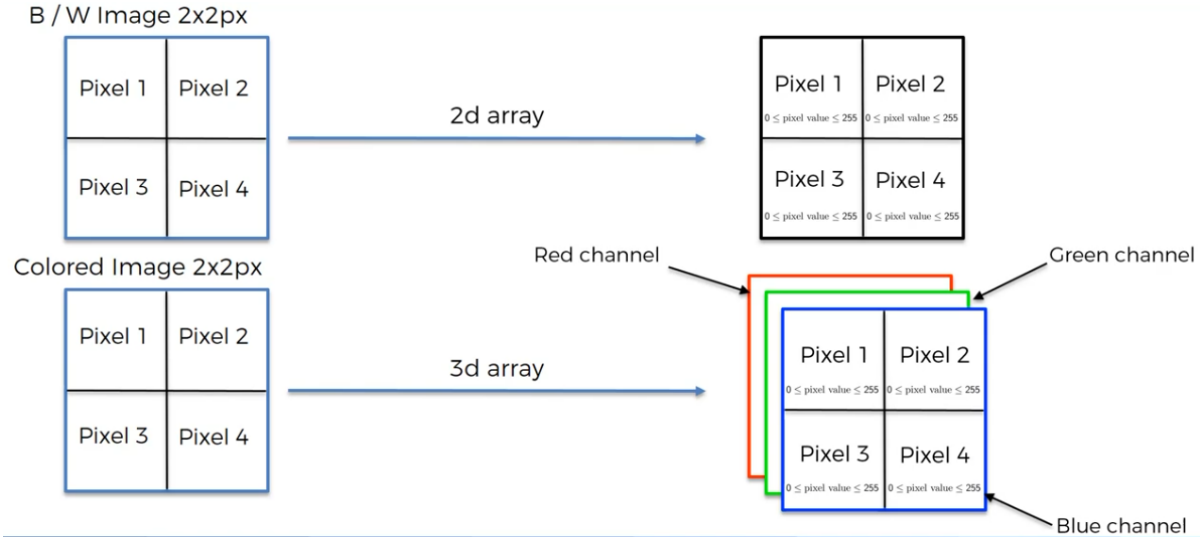
# CNN - Convolutional Neural Networks

A Convolution Neural Network (CNN) is just an Artificial Neural Network (ANN), on which we use this convolution trick to add some convolution layers. Convolution layers are added to preserve the spatial structure in the images.

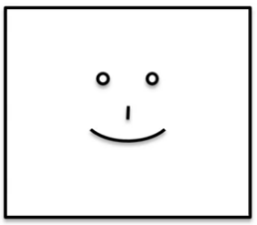
In deep learning, a **convolutional neural network** (**CNN**, or **ConvNet**) is a class of artificial neural network (**ANN**), most commonly applied to analyze visual imagery. [1] CNNs are also known as **Shift Invariant** or **Space Invariant Artificial Neural Networks** (**SIANN**), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps.

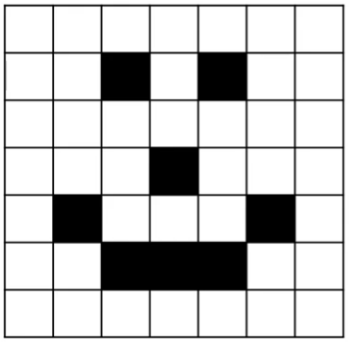



In the image data, we will have man pixels, in which each pixel will have some value attached to it.
For Black & White image, the image will be represented as a 2d array, with each pixel having a value between 0 to 255 (grayscale range).
0 ------> represents complete black pixel
255 ---> represents complete white

For Coloured image, we wil have a 3-d array. The image will have 3 layers, red, blue, green layer. Basically a pixel has 3 values (ranging from 0 to 255) assigned to it. These values represents the intensity of each colour in that pixel.

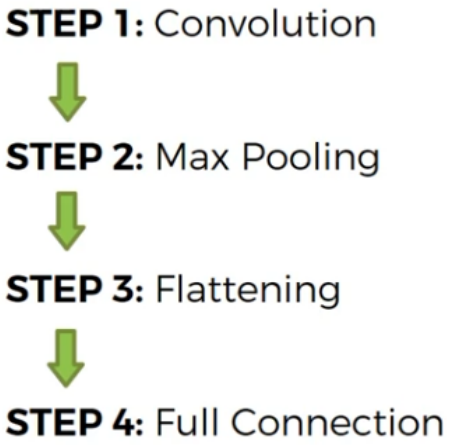
For example when we zoom a simple image, it has the pixels marked like that and that grid can be represented as a 2d array like the last image below.





| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**CNN Steps**

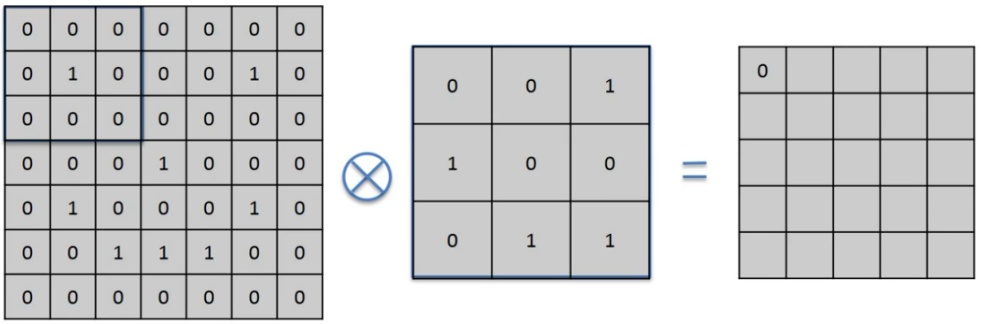



**Step-1: Convolution**

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

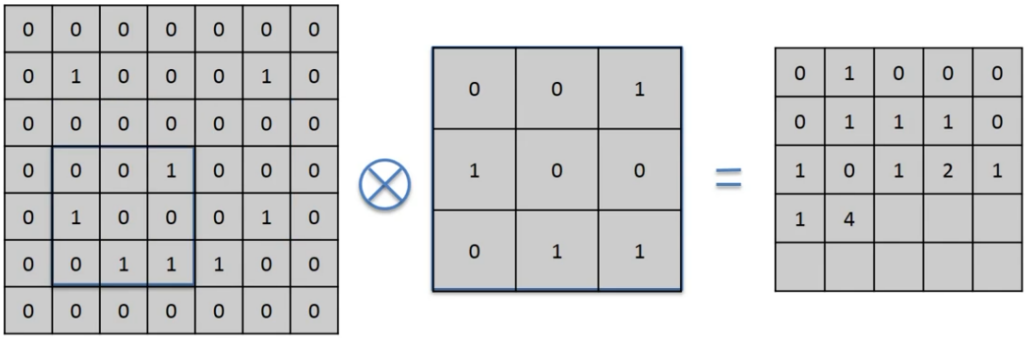
A Convolution is basically a combined integration of 2 functions and it shows you how one function modifies the shape of the other.

Feature Detectors can also be called as filters. They can be of any size. Like 3x3, 4x4, 7x7 and so on.




Put the feature detector on top of the image and do an element wise multiplication. Whenever a match, mark it as 1 in the feature map. If two values matchup then put 2 in the Feature map and so on.

**Stride :** It is **the number of pixels shifts over the input matrix**. The number of pixel shifts you are doing while moving the feature detector on the input image is called stride. You can use any no.of strides 1, 2,3,4, etc.




The above image shows there are places where 2 elements position matched and a place where 4 elements matched.

The Feature Map is also sometimes called as **Activation Mapor Convolved Feature.**

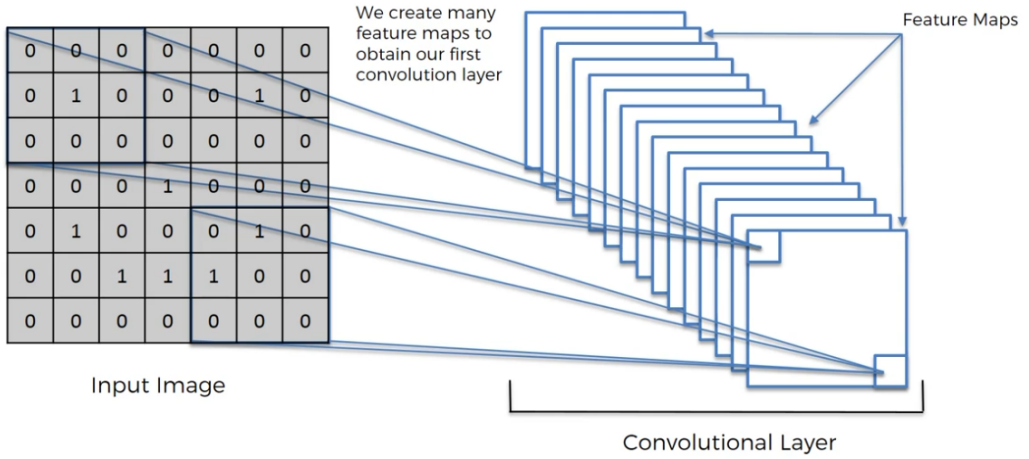
Here in the above example the image is reduced in size. The same original image is represented by the Feature Map. Here we have used the stride of 1. But if we use higher strides of 2, 3 etc then the Feature Map will further be reduced in size. The whole idea of convolution is to reduce the size of the image. So, that the computation will be faster.

If we take a real picture of just 300 x 300 pixels then we will have 90,000 pixels to process for just one image, which is very high computational overhead. So, by

using the feature detector we are reducing the image size and learning the important features.

**Q: Do we loose information while applying the feature detector?**
Yes, we will loose some of the information. But important information is captured using the feature detectors by matching the patterns. Features is how we detect and process images. By applying many feature detectors, we are trying to capture different features/patterns. So, this will compensate the loss in information.

We apply many different filters/feature detectors and create many Feature Maps as shown below. Each feature map will contain certain type of features/patterns.




**Example**

Sharpen:

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 |
| 0 | -1 | 5 | -1 | 0 |
| 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

In the above image, by applying the above filter,  it sharpens the image.

## Blur:

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

By applying the above filter we get a blur image.

## Edge Enhance:

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 0 | 0 | 0 | |
| | -1 | 1 | 0 | |
| | 0 | 0 | 0 | |
| | | | | |

## Edge Detect:

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 0 | 1 | 0 | |
| | 1 | -4 | 1 | |
| | 0 | 1 | 0 | |
| | | | | |

By reducing the middle pixel and making the surrounding we try to detect the Edges of the input image.

Emboss:

| | | | | |
|---|---|---|---|---|
| | -2 | -1 | 0 | |
| | -1 | 1 | 1 | |
| | 0 | 1 | 2 | |
| | | | | |

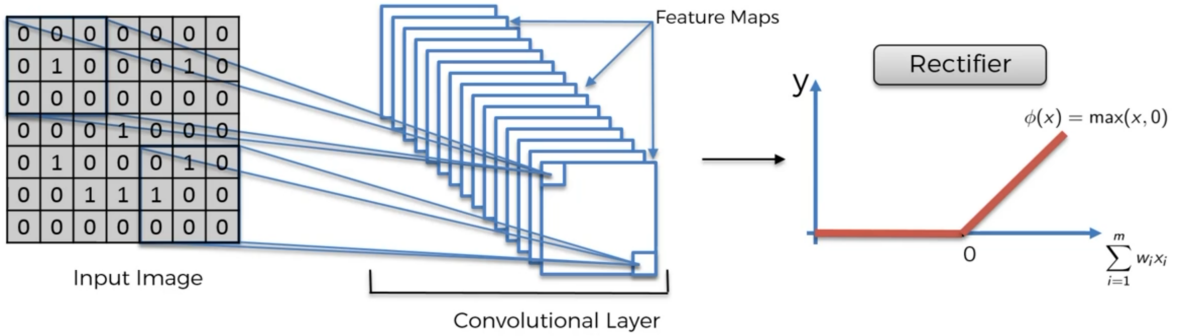
From the above example, with a single input image, by applying different feature detectors we are getting feature maps indicating different features of the input image. Certain things like the Edge Enhance, Emboss might not be useful for us in CNN, but certain filters are important for us like the Edge Detect, Sharpen etc. The neural nets will decide what feature maps are important. Human eye might NOT be able to understand the importance.

$*$

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

## Step 1(B) - ReLU Layer

This is a small additional step on convolution.
We apply ReLU on the convolution layer, in to introduce/increase the non-linearity in the network.
Also ReLU is faster to compute than Sigmoid or tanh.



## Step-2: Max Pooling

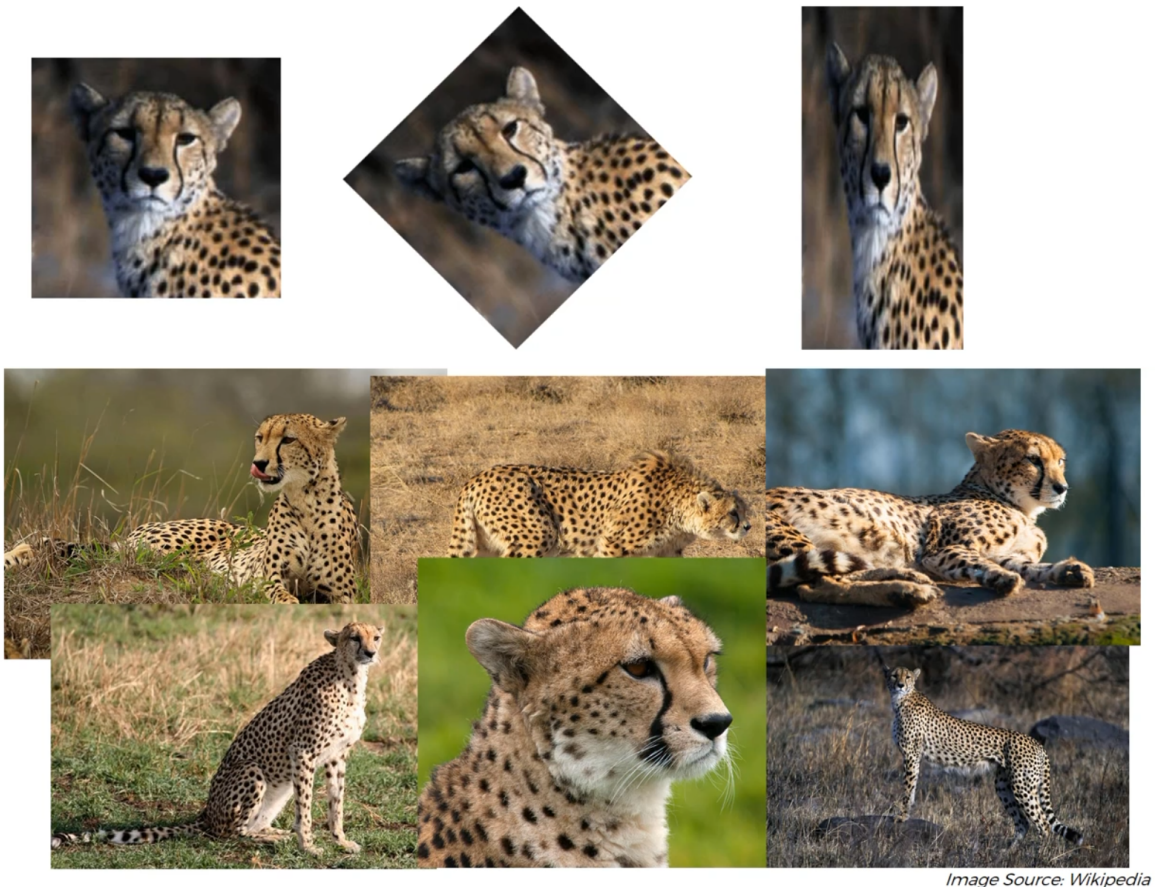
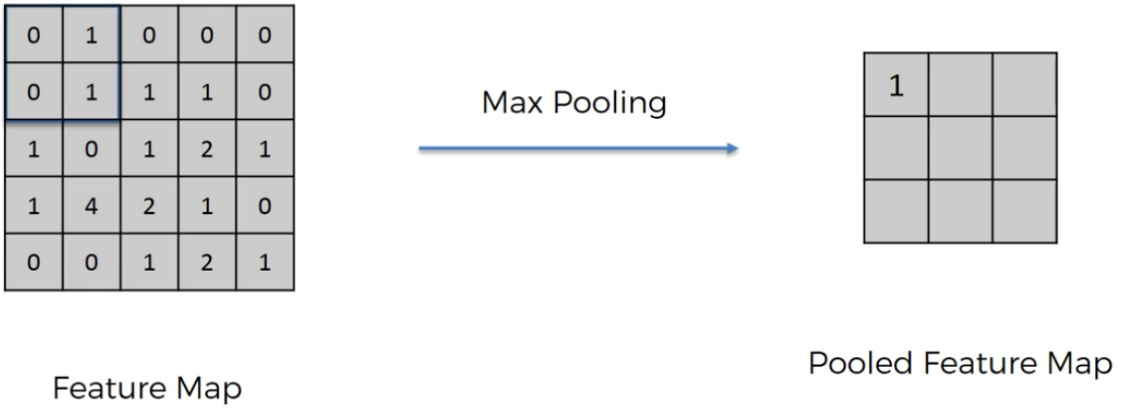


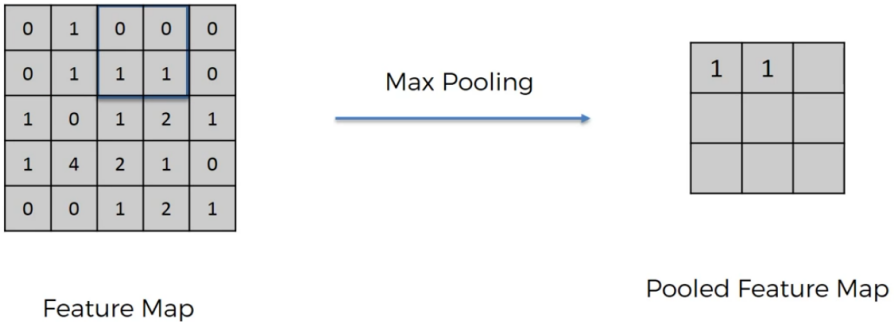
*Image Source: Wikipedia*

If the neural network is trained on the first image of cheetah and it might not be able to recognise the cheetah in the other images as they are in different postures and different positions in the picture and in different angles. **So, we have to make sure that our neural network has a property called Spatial Variance.** Meaning that it doesn't care where the features are located, if the features are bit tilted, distorted, different in texture etc. Our neural network should have the flexibility to deal with these variations. That is what pooling is all about.

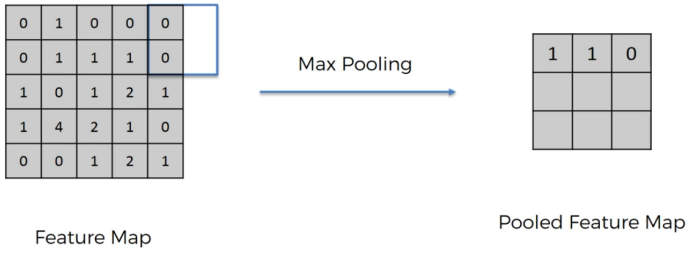
The pooling is applied on the Feature Map i.e. after the convolution.
We take a box of 2x2 pixel and you place it on the top of the Feature Map and identify the maximum value in that box from the Feature Map. So, put that number on the Pooled Feature Map. Disregard the other numbers.

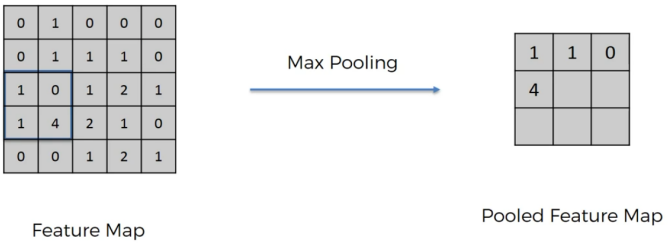


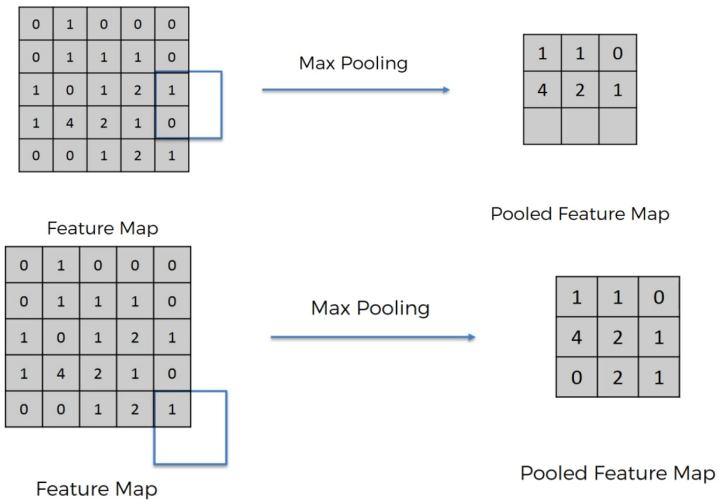

Select a stride before hand and move your box by that stride. Here we selected a stride of 2.




Even if you cross over the Feature Map, it doesn't matter, just continue to do so.






Here we are pooling the features. After the pooling, we will be able to preserve the important features by getting rid of the unimportant information. Although we are keeping only 25% of the info, we are focussing on the important 25% of the info. For example if we consider the number 4 in Feature Map in the above example as the cheetah's eyes. Suppose if the cheetah's image is a bit rotated and the 4 ended up in the near by boxes around 4 in the Feature Map, while applying the Max pooling we will still have 4 in the same place of the Pooled Feature Map. So, in this way we are pooling the important features.

**Advantages**

By pooling we are reducing the size by 75%.

We are also reducing the no.of parameters also. By reducing parameters/information we are reducing over-fitting. Human eyes disregard most of the information while looking at an image and only looks for the features to recognise the image. Similar to human eye, here also by disregarding the non-important information, we are preventing over-fitting.

**Other Pooling Options**

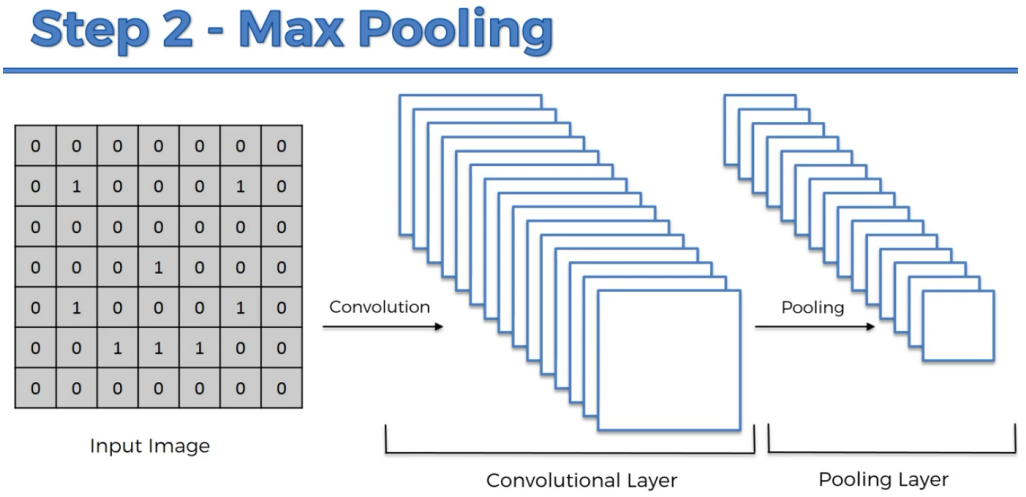
In the max pooling after we put a box on the Feature Map we take the maximum value for the Pooled Feature Map. But instead of taking maximum value, we have many other pooling options as below.

**A. Sum Pooling :** We just sum the values with in the box and put that number in the Pooled Feature Map.

**B. Average/Mean Pooling :** We just average the values with in the box and put that number in the Pooled Feature Map.

**C. Subsampling :** It is a generalization of mean pooling. It is more generalized approach of averaging of the values. (Have to read more about it).

**Summary**



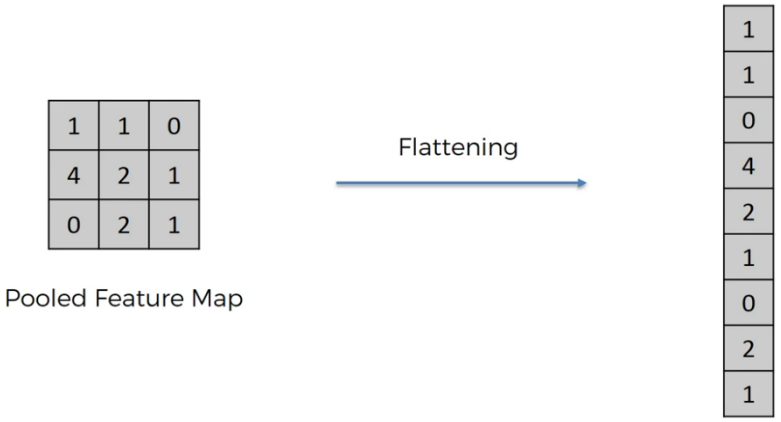


The bottom most image is the original image. Last but one layer is the Feature Maps generated by applying different Feature Detectors on the input image. The third from last layer is the Max-Pooled Feature Maps obtained on the Feature Maps. Notice that after pooling we are still able to preserve the important features even after the reduction in size of the image.

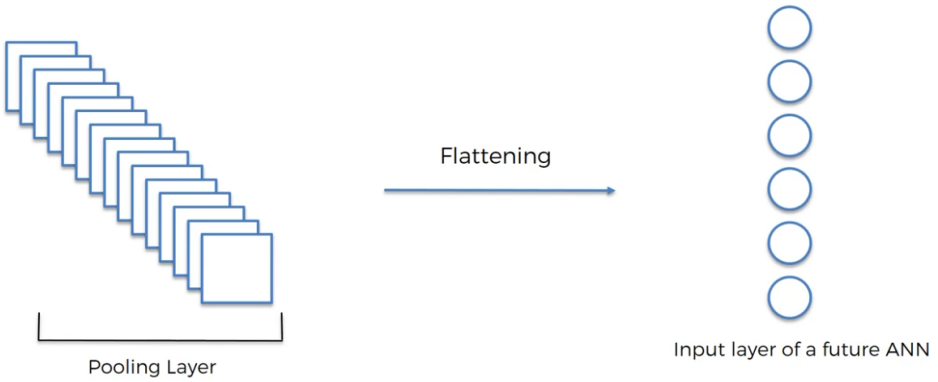

The neural net is trained on the digits from 0 to 9. But we drew a smiley face in the above image. As it's job is to classify the input image as one of the digits, it detects the input as '3' with some probability. Similarly humans also, when we see a fruit that we have never seen before, we will try to classify it as a similar fruit which our brain is trained off in the past.

## Step-3: Flattening

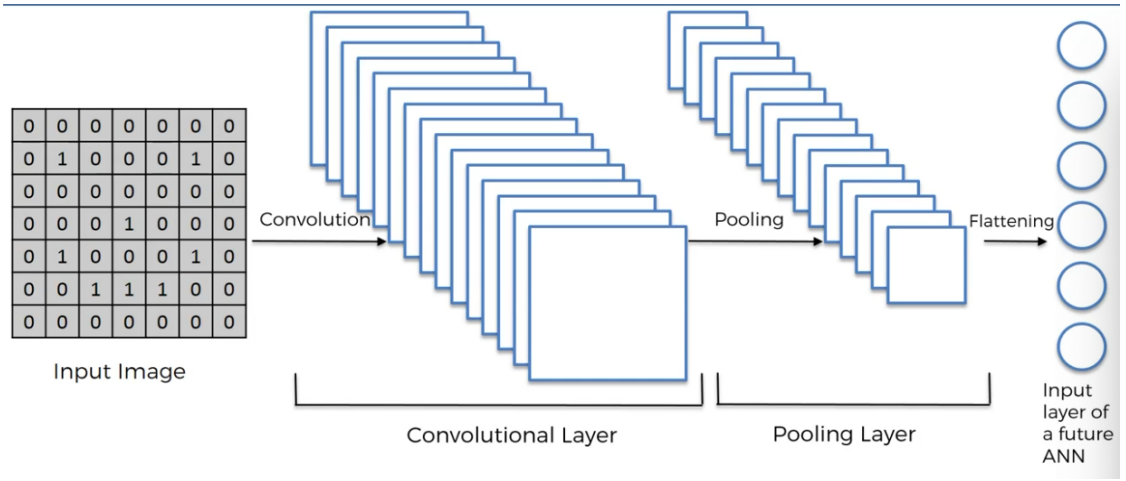
We take the Pooled Feature Map and flatten them out in this step as shown below. We just put all the numbers of the Pooled Feature Map in to a column one-by-one in each row.




We pass this as input to the ANN for further processing. We will flatten out all the Pooled Feature Maps and pass in to the ANN sequentially one after the other. By this we get one huge vector of inputs for the ANN.

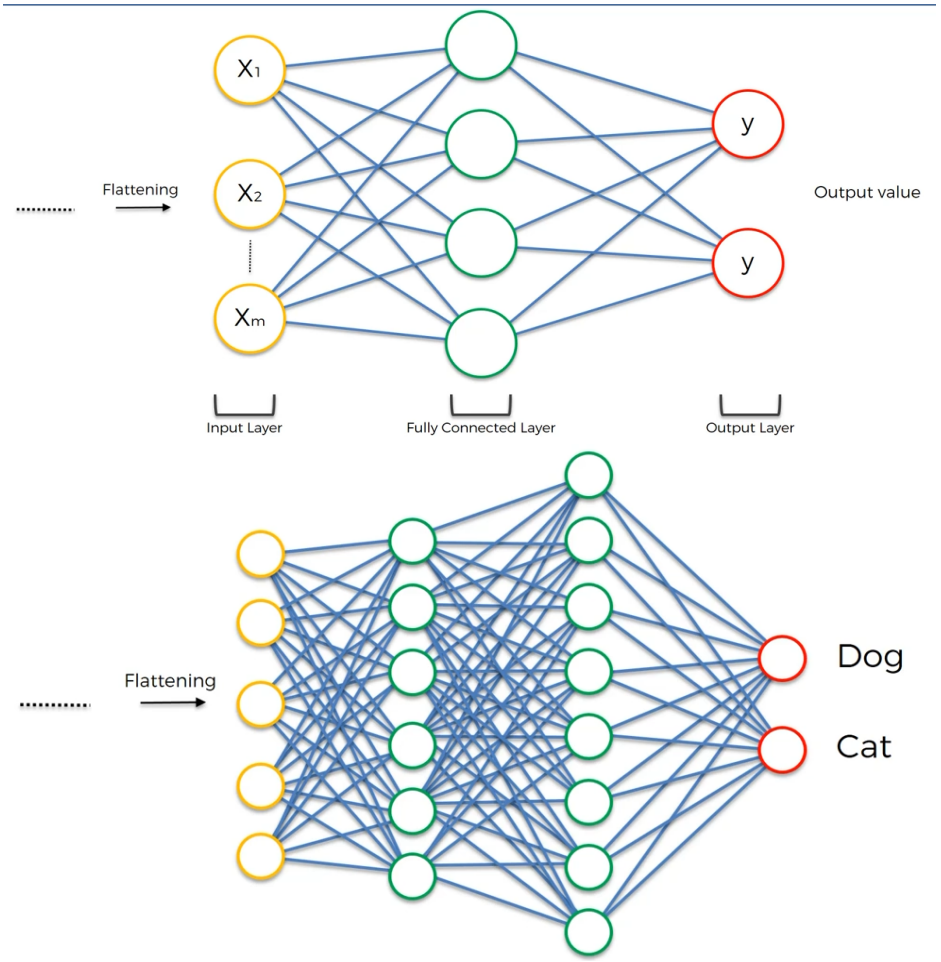



## Summary

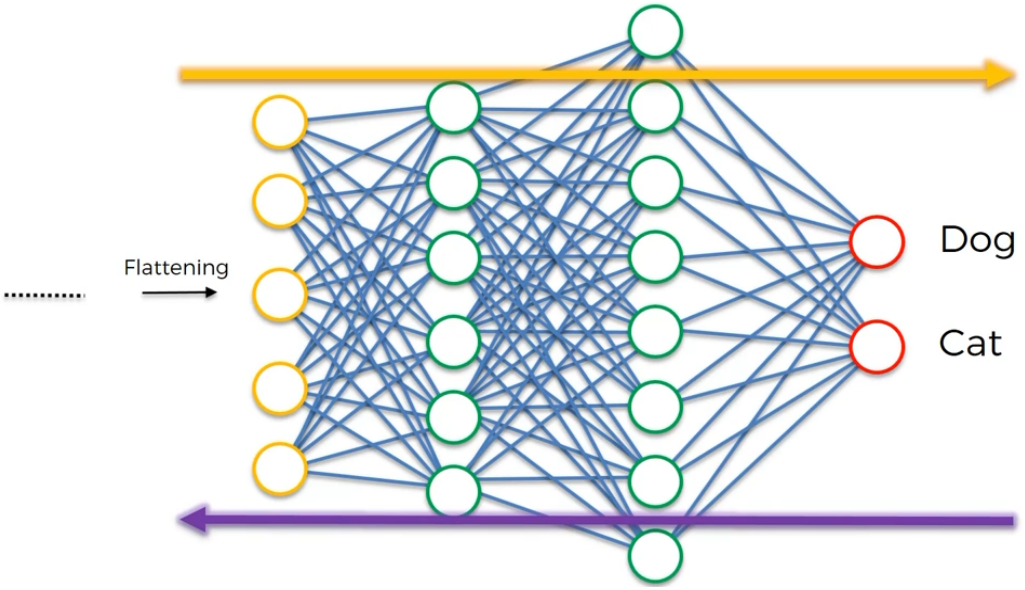



## Step-4: Full Connection

We connect the ANN with full connection to the CNN. In the ANN we usually might use full connection or not. But here in the CNN we usually do the full connection in the hidden layers.




During regression we will have one neuron in the output layer. But for the classification we need one neuron per each class. For Binary we can have only one neuron.

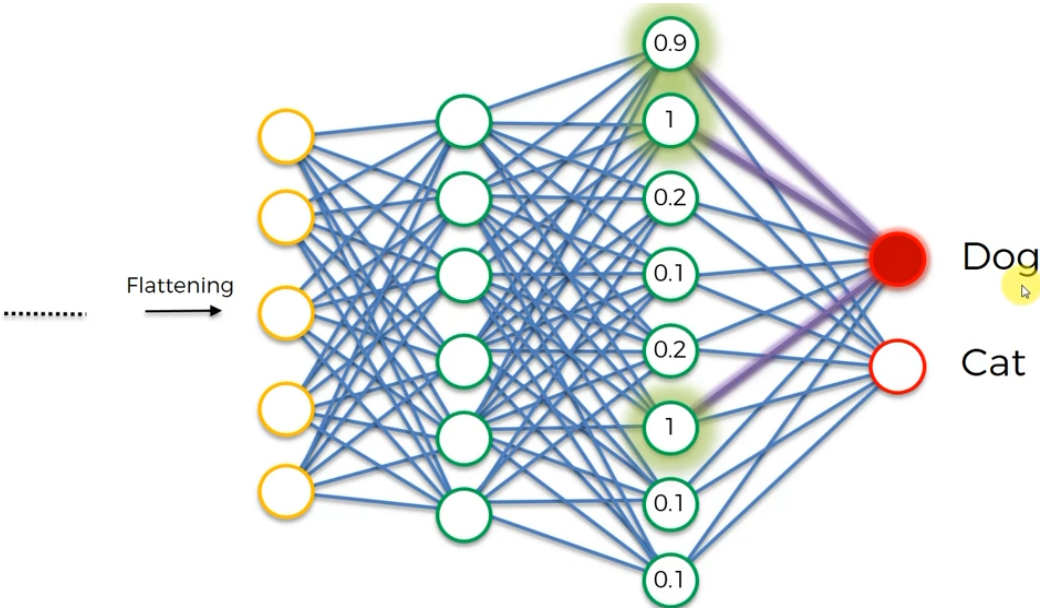



After the error is calculated and while the back propagation in the CNN, along with the weights the feature detectors are also adjusted. So, that new feature detectors

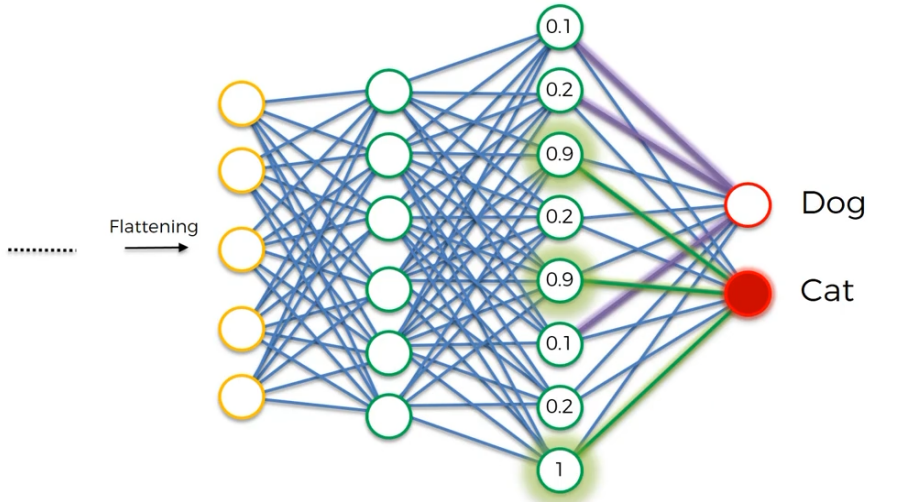
are placed and convolution is done and new Feature Maps are generated, Pooling is done and new Pooled Feature Maps are generated, flattened and passed to the ANN.

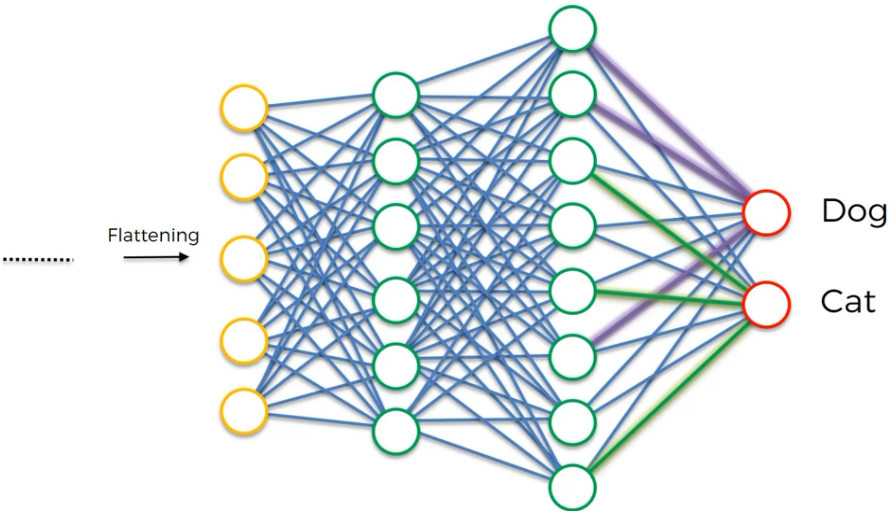
In the above example, if the dog is detected as cat, then there might be problem with the feature detectors like what if we are looking for the wrong features. So, to correct that, the back propagation adjusts the feature detectors as well.

Lets examine how the output layer's two neurons are helpful for the binary classification. **Why having 2 neurons in the output layer is helpful than one neuron for binary classification?**
Consider the following example.  During the training, after many iterations and epochs the Dog neuron learned that the highlighted neurons are more useful for calculating the probability of the input as Dog. The purple colour highlighted synopses will have the weights, i.e. importance values for those neurons. So, by looking at those weights the Dog neuron treat the output of those neurons as important for it.  The numbers in the hidden layer neurons are just for illustration here.

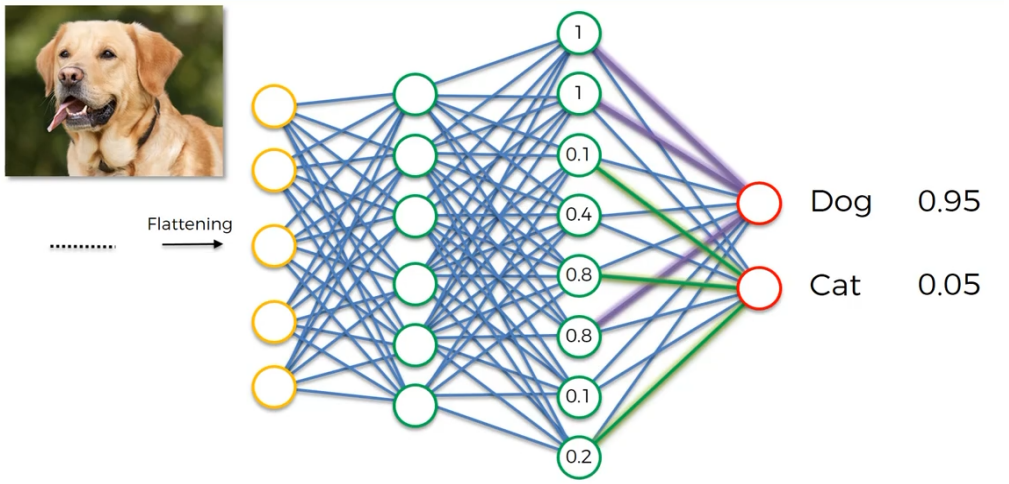


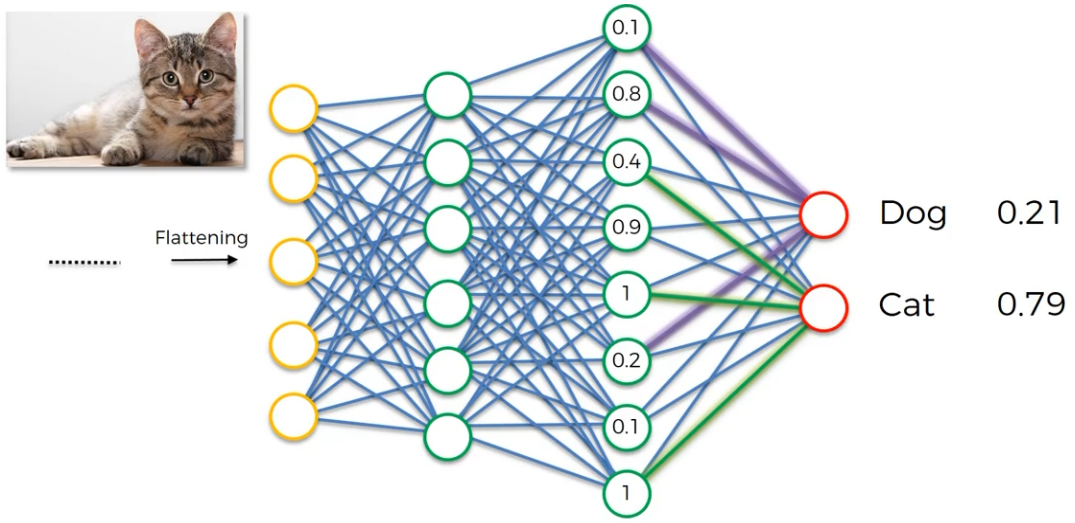

Similarly, during the training, after many iterations and epochs the Cat neuron learned that the highlighted neurons are more useful for calculating the probability of the input as Cat. The green colour highlighted synopses will have the weights, i.e. importance values for those neurons. So, by looking at those weights the Cat neuron treat the output of those neurons as important for it. The numbers in the hidden layer neurons are just for illustration here.



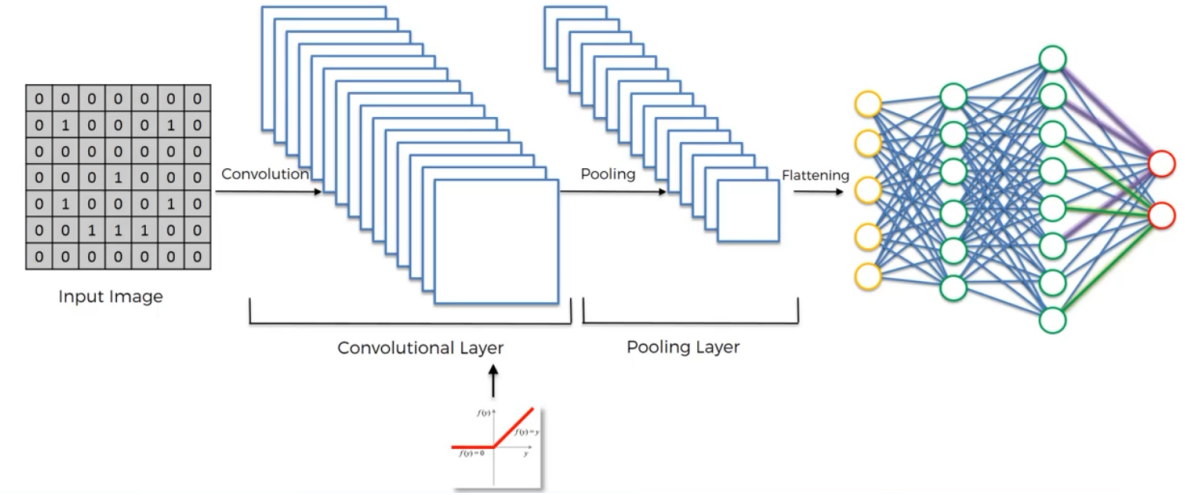




So, above is the final network after training.


Flattening
1
1
0.1
0.4
0.8
0.8
0.1
0.2
Dog 0.95
Cat 0.05


Flattening
0.1
0.8
0.4
0.9
1
0.2
0.1
1
Dog 0.21
Cat 0.79

## Summary


Input Image
Convolution
Pooling
Flattening
Convolutional Layer
Pooling Layer