Convolutional Neural Network (CNN)

What is a Convolutional Neural Network (CNN)?

A Convolutional Neural Network (CNN), also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others.

Key Components of a CNN

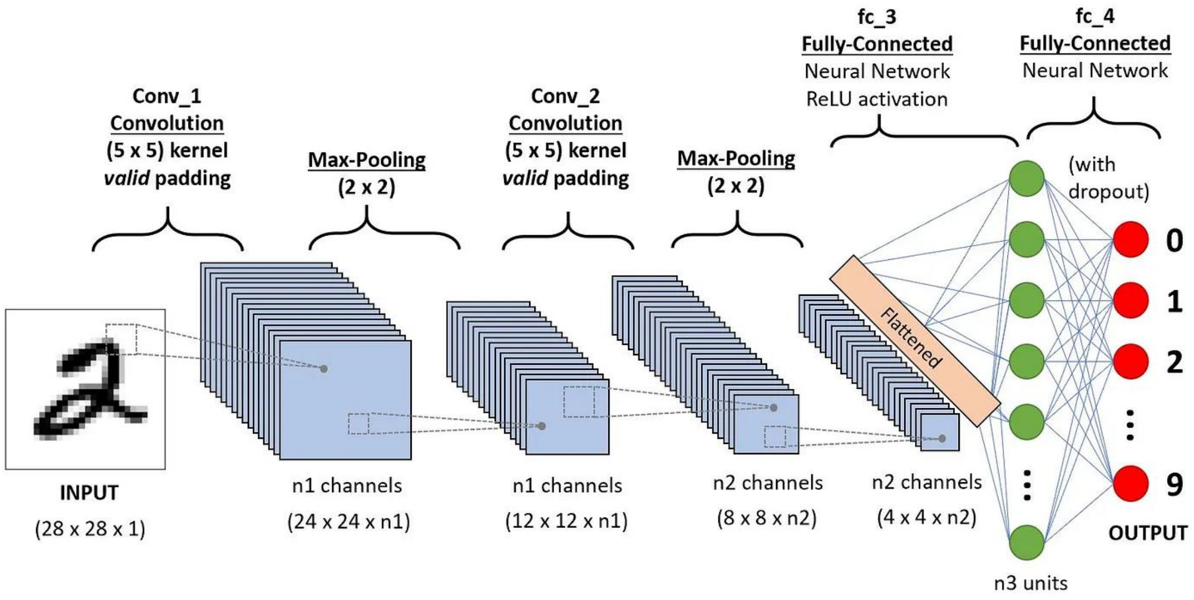
The convolutional neural network is made of four main parts.

But how do CNNs Learn with those parts?

They help the CNNs mimic how the human brain operates to recognize patterns and features in images:

- Convolutional layers
- Rectified Linear Unit (ReLU for short)
- Pooling layers
- Fully connected layers

This section dives into the definition of each one of these components through the example of the following example of classification of a handwritten digit.



Architecture of the CNNs applied to digit recognition

Convolution layers

This is the first building block of a CNN. As the name suggests, the main mathematical task performed is called convolution, which is the application of a sliding window function to a matrix of pixels representing an image. The sliding function applied to the matrix is called kernel or filter, and both can be used interchangeably.

In the convolution layer, several filters of equal size are applied, and each filter is used to recognize a specific pattern from the image, such as the curving of the digits, the edges, the whole shape of the digits, and more.

Put simply, in the convolution layer, we use small grids (called filters or kernels) that move over the image. Each small grid is like a mini magnifying glass that looks for specific patterns in the photo, like lines, curves, or shapes. As it moves across the photo, it creates a new grid that highlights where it found these patterns.

For example, one filter might be good at finding straight lines, another might find curves, and so on. By using several different filters, the CNN can get a good idea of all the different patterns that make up the image.

Let's consider this 32x32 grayscale image of a handwritten digit. The values in the matrix are given for illustration purposes.

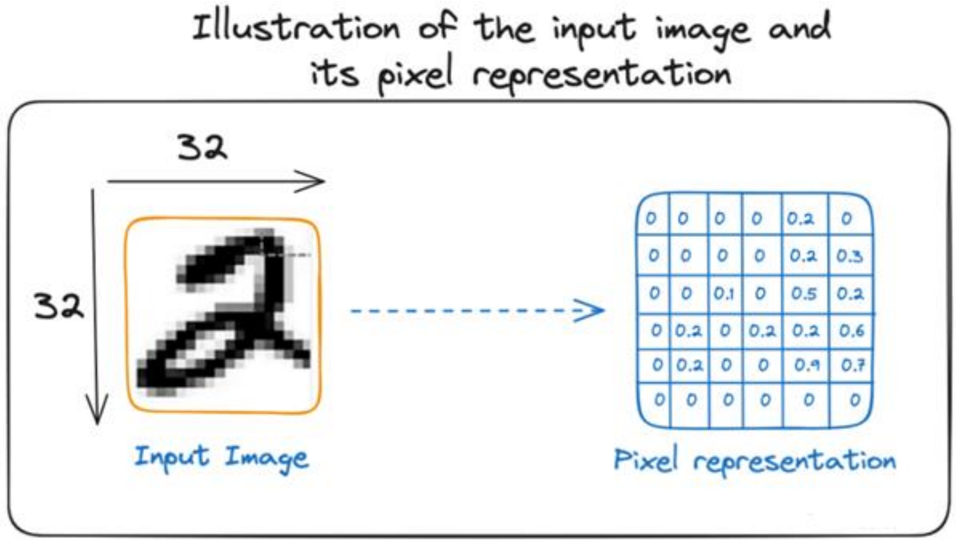


Illustration of the input image and its pixel representation
32
32
Input Image
Pixel representation

0	0	0	0	0.2	0
0	0	0	0	0.2	0.3
0	0	0.1	0	0.5	0.2
0	0.2	0	0.2	0.2	0.6
0	0.2	0	0	0.9	0.7
0	0	0	0	0	0

Illustration of the input image and its pixel representation

Also, let's consider the kernel used for the convolution. It is a matrix with a dimension of 3x3. The weights of each element of the kernel is represented in the grid. Zero weights are represented in the black grids and ones in the white grid.

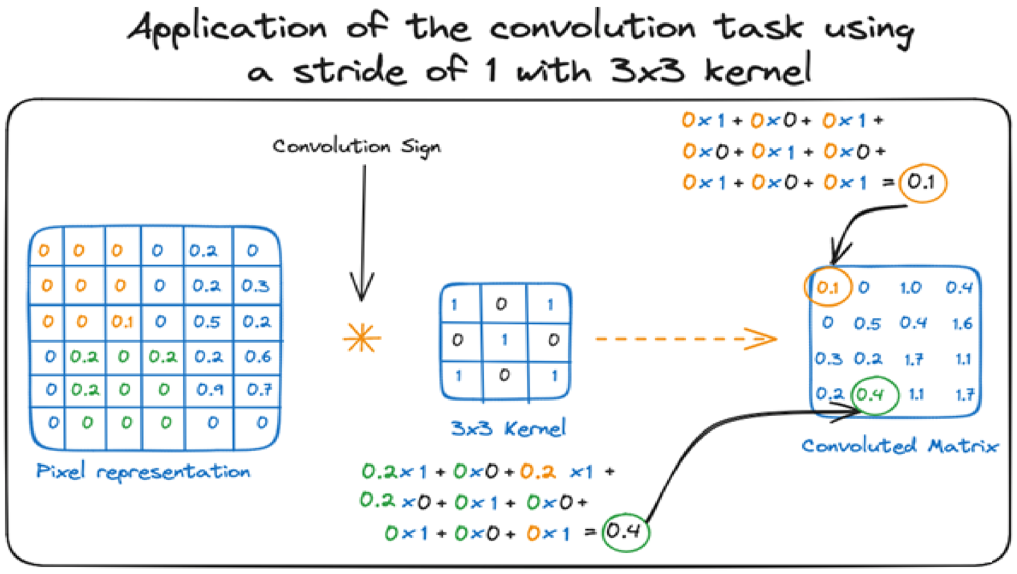
Do we have to manually find these weights?

In real life, the weights of the kernels are determined during the training process of the neural network.

Using these two matrices, we can perform the convolution operation by applying the dot product, and work as follows:

1. **Apply the kernel matrix from the top-left corner to the right.**
2. **Perform element-wise multiplication.**
3. **Sum the values of the products.**
4. **The resulting value corresponds to the first value (top-left corner) in the convoluted matrix.**
5. **Move the kernel down with respect to the size of the sliding window.**
6. **Repeat steps 1 to 5 until the image matrix is fully covered.**

The dimension of the convoluted matrix depends on the size of the sliding window. The higher the sliding window, the smaller the dimension.



Application of the convolution task using a stride of 1 with 3x3 kernel

Another name associated with the kernel in the literature is feature detector because the weights can be fine-tuned to detect specific features in the input image.

For instance:

- **Averaging neighboring pixels kernel can be used to blur the input image.**
- **Subtracting neighboring kernel is used to perform edge detection.**

The more convolution layers the network has, the better the layer is at detecting more abstract features.

Activation function

A ReLU activation function is applied after each convolution operation. This function helps the network learn non-linear relationships between the features in the image, hence making the network more robust for identifying different patterns. It also helps to mitigate the vanishing gradient problems.

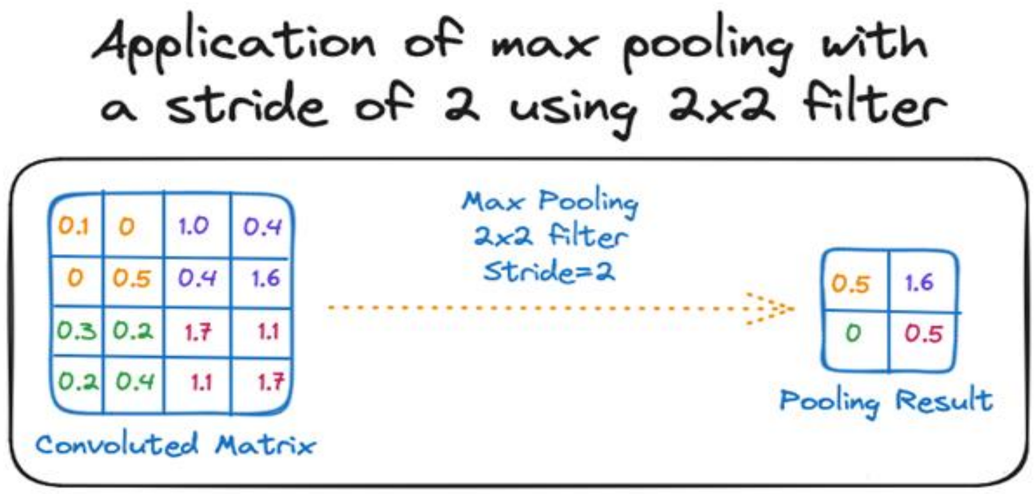
Pooling layer

The goal of the pooling layer is to pull the most significant features from the convoluted matrix. This is done by applying some aggregation operations, which reduce the dimension of the feature map (convoluted matrix), hence reducing the memory used while training the network. Pooling is also relevant for mitigating overfitting.

The most common aggregation functions that can be applied are:

- **Max pooling, which is the maximum value of the feature map**
- **Sum pooling corresponds to the sum of all the values of the feature map**
- **Average pooling is the average of all the values.**

Below is an illustration of each of the previous example:



Application of max pooling with a stride of 2 using 2x2 filter

Also, the dimension of the feature map becomes smaller as the pooling function is applied.

The last pooling layer flattens its feature map so that it can be processed by the fully connected layer.

Fully connected layers

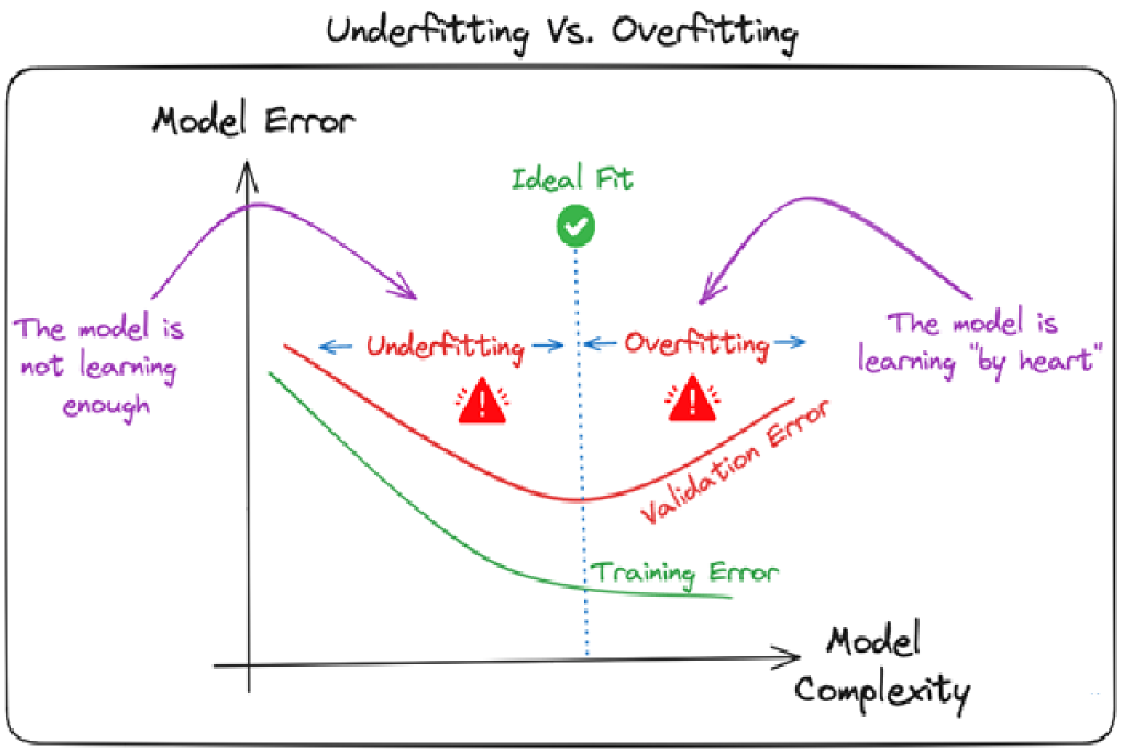
These layers are in the last layer of the convolutional neural network, and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer. ReLU activations functions are applied to them for non-linearity.

Finally, a softmax prediction layer is used to generate probability values for each of the possible output labels, and the final label predicted is the one with the highest probability score.

Overfitting and Regularization in CNNs

Overfitting is a common challenge in machine learning models and CNN deep learning projects. It happens when the model learns the training data too well (“learning by heart”), including its noise and outliers. Such a learning leads to a model that performs well on the training data but badly on new, unseen data.

This can be observed when the performance on training data is too low compared to the performance on validation or testing data, and a graphical illustration is given below:


Underfitting Vs. Overfitting
Model Error
Ideal Fit
The model is not learning enough
The model is learning "by heart"
Underfitting
Overfitting
Validation Error
Training Error
Model Complexity

Underfitting Vs. Overfitting

Deep learning models, especially Convolutional Neural Networks (CNNs), are particularly susceptible to overfitting due to their capacity for high complexity and their ability to learn detailed patterns in large-scale data.

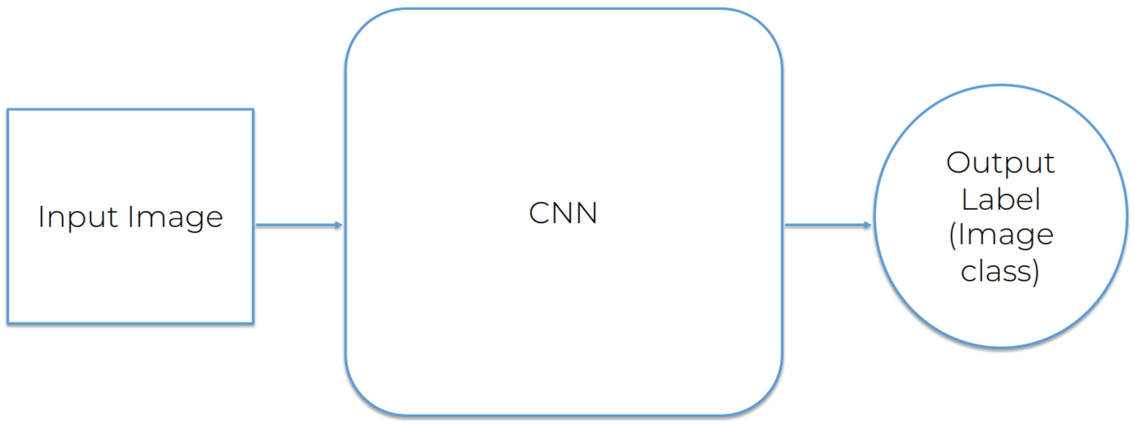
Several regularization techniques can be applied to mitigate overfitting in CNNs, and some are illustrated below:

7 strategies to mitigate overfitting in CNNs

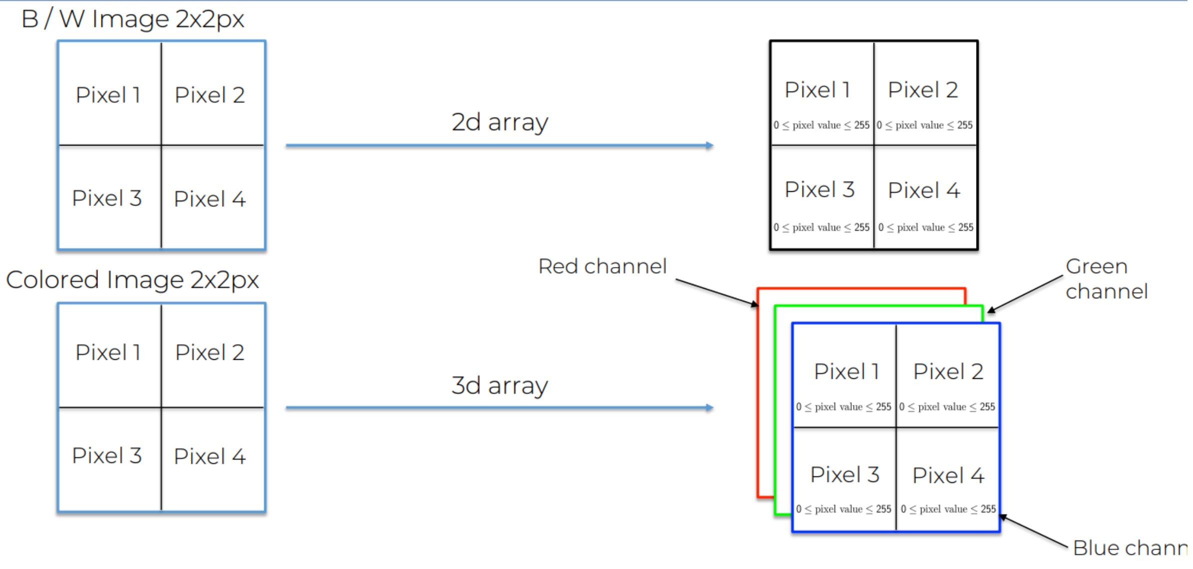
- **Dropout: This consists of randomly dropping some neurons during the training process, which forces the remaining neurons to learn new features from the input data.**
- **Batch normalization: The overfitting is reduced at some extent by normalizing the input layer by adjusting and scaling the activations. This approach is also used to speed up and stabilize the training process.**
- **Pooling Layers: This can be used to reduce the spatial dimensions of the input image to provide the model with an abstracted form of representation, hence reducing the chance of overfitting.**
- **Early stopping: This consists of consistently monitoring the model's performance on validation data during the training process and stopping the training whenever the validation error does not improve anymore.**

- **Noise injection: This process consists of adding noise to the inputs or the outputs of hidden layers during the training to make the model more robust and prevent it from a weak generalization.**
- **L1 and L2 normalizations: Both L1 and L2 are used to add a penalty to the loss function based on the size of weights. More specifically, L1 encourages the weights to be spare, leading to better feature selection. On the other hand, L2 (also called weight decay) encourages the weights to be small, preventing them from having too much influence on the predictions.**
- **Data augmentation: This is the process of artificially increasing the size and diversity of the training dataset by applying random transformations like rotation, scaling, flipping, or cropping to the input images.**

Convolutional Neural Networks



Convolutional Neural Networks



Convolutional layer:

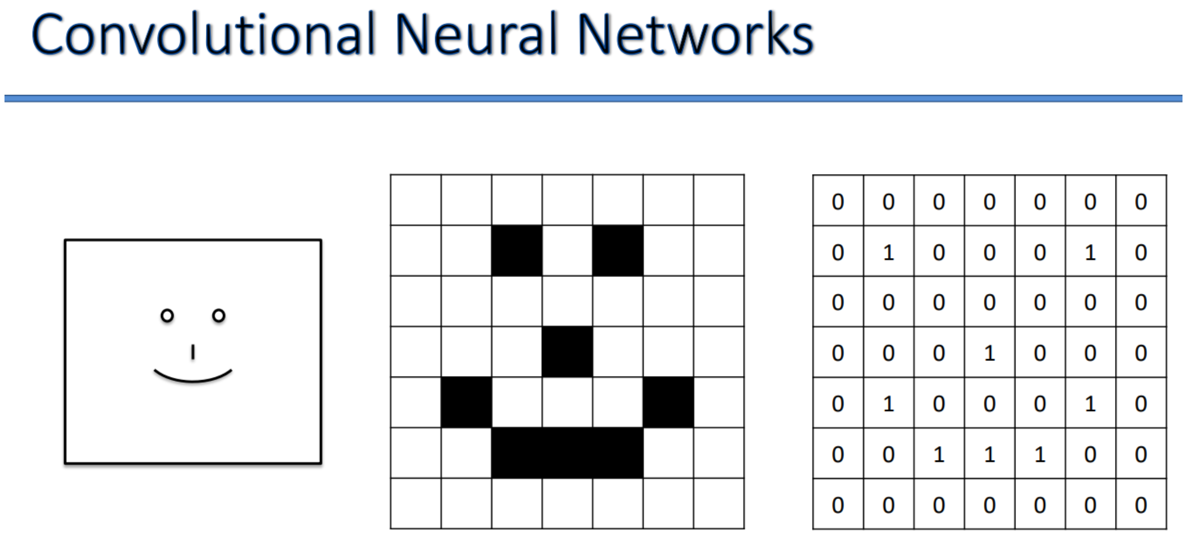
In a CNN, the input is a tensor with shape:

(number of inputs) × (input height) × (input width) × (input channels)

After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape:

(number of inputs) × (feature map height) × (feature map width) × (feature map channels).

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field.



Convolutional Neural Networks

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

Convolutional Neural Networks

STEP 1: Convolution

⬇

STEP 2: Max Pooling

⬇

STEP 3: Flattening

⬇

STEP 4: Full Connection

Convolutional Neural Networks

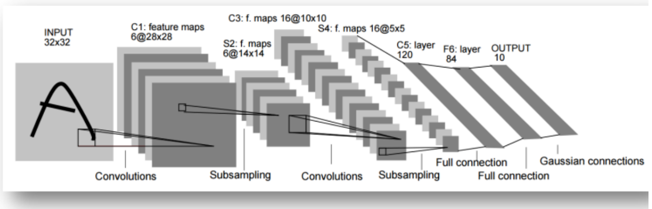
Additional Reading:

Gradient-Based Learning Applied to Document Recognition

By Yann LeCun et al. (1998)

Link:

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf



Step 1 - Convolution

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

Step 1 - Convolution

Additional Reading:

Introduction to Convolutional Neural Networks

By Jianxin Wu (2017)

$$\begin{aligned}\frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{y})^T)}(F^T \otimes I) &= \left((F \otimes I)\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})}\right)^T \\ &= \left((F \otimes I)\operatorname{vec}\left(\frac{\partial z}{\partial Y}\right)\right)^T \\ &= \operatorname{vec}\left(I\frac{\partial z}{\partial Y}F^T\right)^T \\ &= \operatorname{vec}\left(\frac{\partial z}{\partial Y}F^T\right)^T,\end{aligned}$$

Link:
http://cs.nju.edu.cn/wujx/paper/CNN.pdf

Step 1 - Convolution

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

Input Image

0	0	1
1	0	0
0	1	1

Feature Detector

Step 1- Convolution

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

Input Image

$\otimes$

0	0	1
1	0	0
0	1	1

Feature Detector

$=$

0				

Feature Map

Step 1 - Convolution

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

Input Image

$\otimes$

0	0	1
1	0	0
0	1	1

Feature Detector

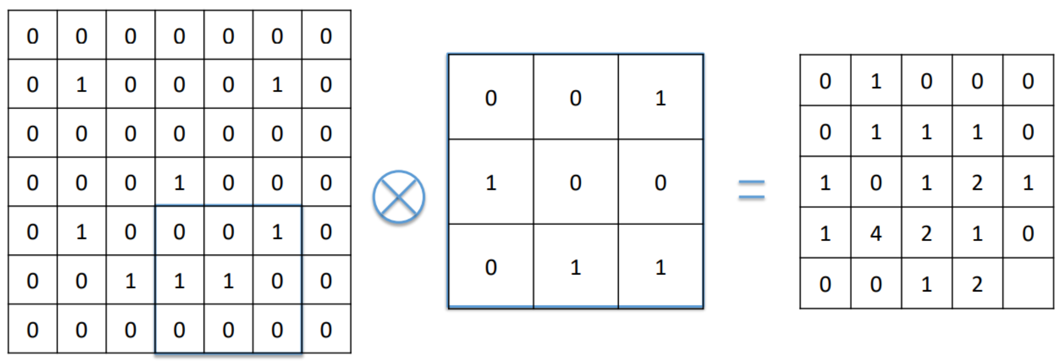
$=$

0	1			

Feature Map

Same way last but one Step

Step 1 - Convolution



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

Input Image

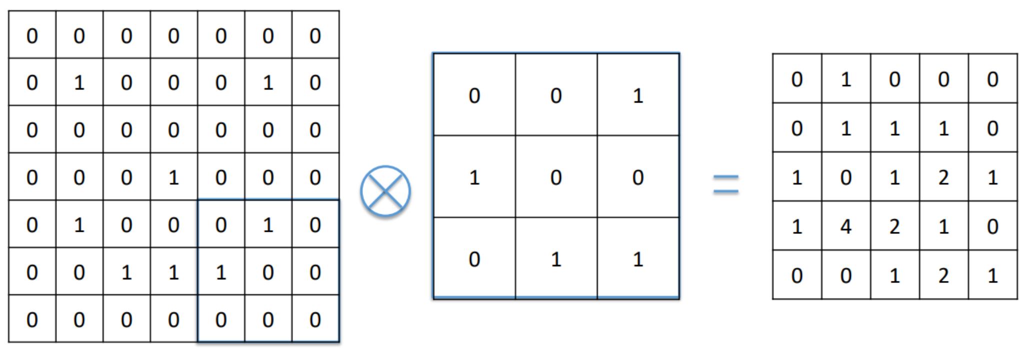
0	0	1
1	0	0
0	1	1

Feature Detector

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	

Feature Map

Step 1 - Convolution



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

Input Image

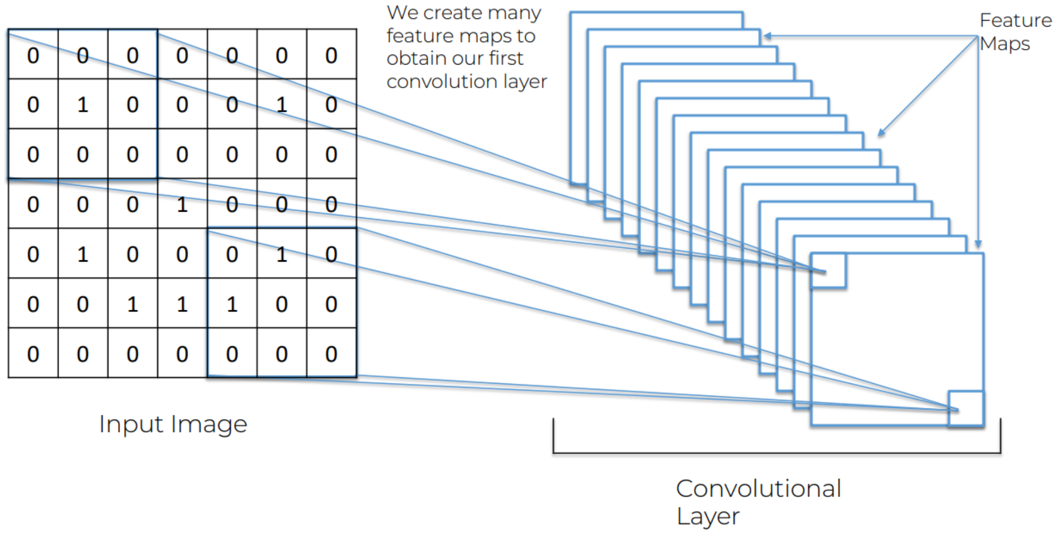
0	0	1
1	0	0
0	1	1

Feature Detector

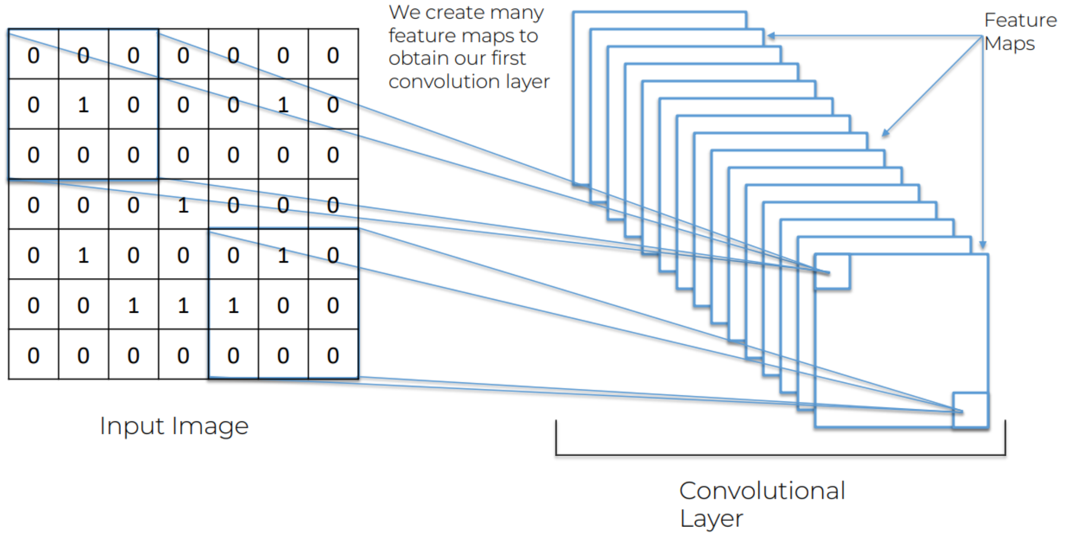
0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Step 1 - Convolution



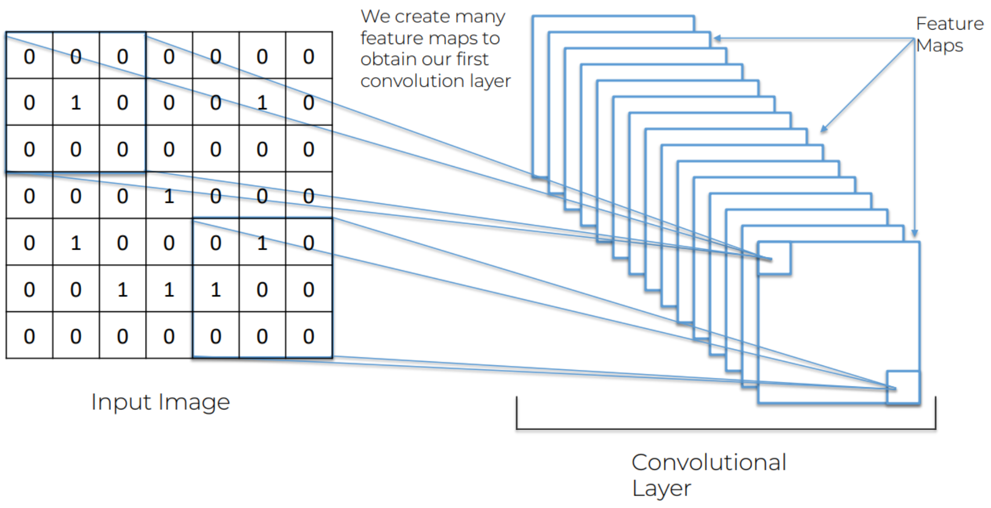
Step 1 - Convolution



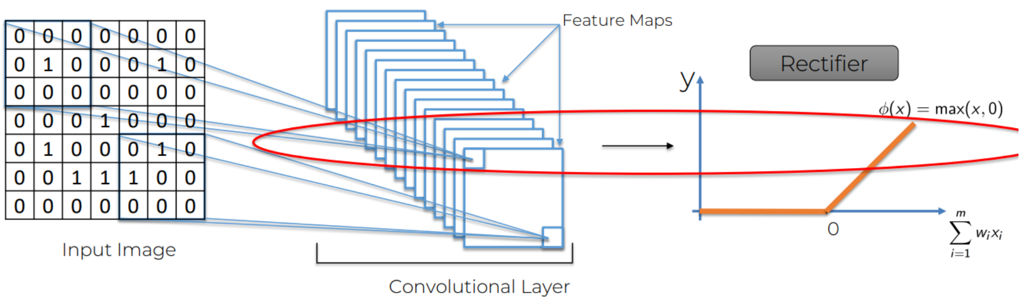
ReLU layer

ReLU applies the non-saturating activation function. It effectively removes negative values from an activation map by setting them to zero.[91] It introduces nonlinearity to the decision function and in the overall network without affecting the receptive fields of the convolution layers.

Step 1(B) – ReLU Layer


We create many feature maps to obtain our first convolution layer
Feature Maps
Input Image
Convolutional Layer

Step 1(B) – ReLU Layer

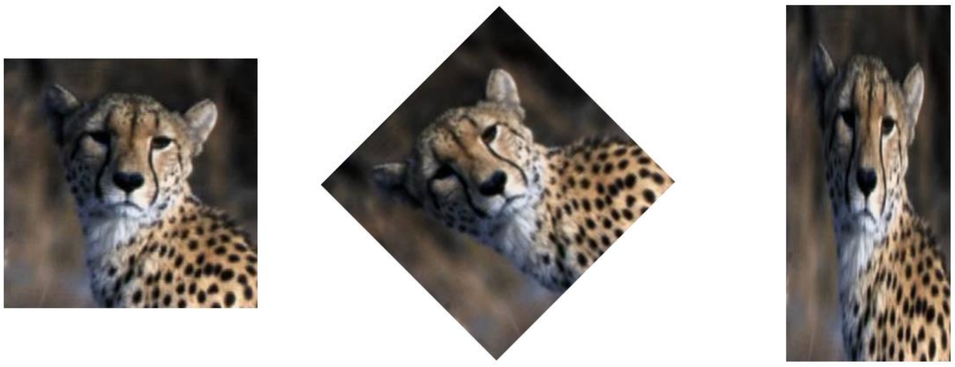


<u>Step-2 : Max Pooling</u>

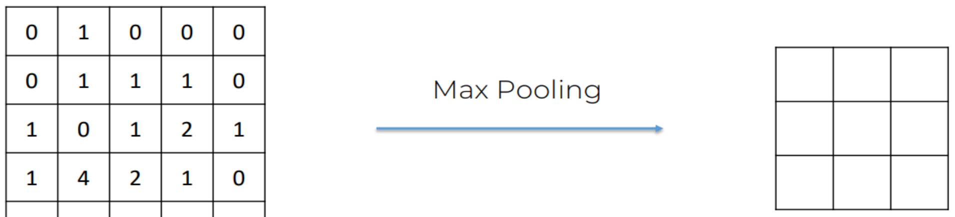
Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling, where *max pooling* is the most common.
It partitions the input image into a set of rectangles and, for each such sub-region, outputs the maximum.

A channel max pooling (CMP) operation layer conducts the MP operation along the channel side among the corresponding positions of the consecutive feature maps for the purpose of redundant information elimination. The CMP makes the significant features gather together within fewer channels, which is important for fine-grained image classification that needs more discriminating features. Meanwhile, another advantage of the CMP operation is to make the channel number of feature maps smaller before it connects to the first fully connected (FC) layer. Similar to the MP operation, we denote the input feature maps and output feature maps of a CMP layer as F ∈ R(C×M×N) and C ∈ R(c×M×N), respectively, where C and c are the channel numbers of the input and output feature maps, M and N are the widths and the height of the feature maps, respectively.

Step 2 - Max Pooling



Step 2 - Max Pooling

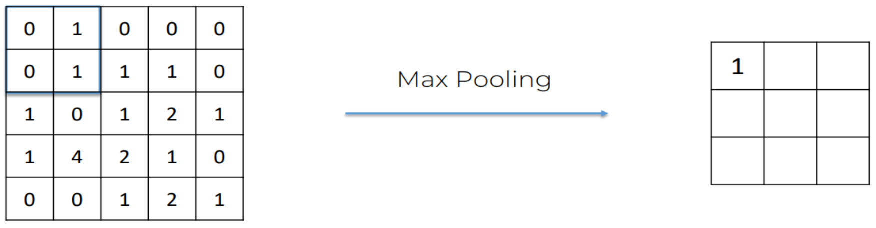


0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Pooled Feature Map

Step 2 - Max Pooling

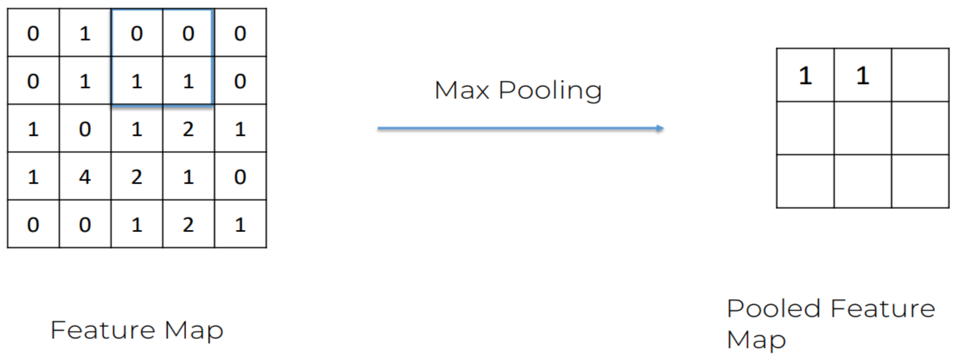


0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Pooled Feature Map

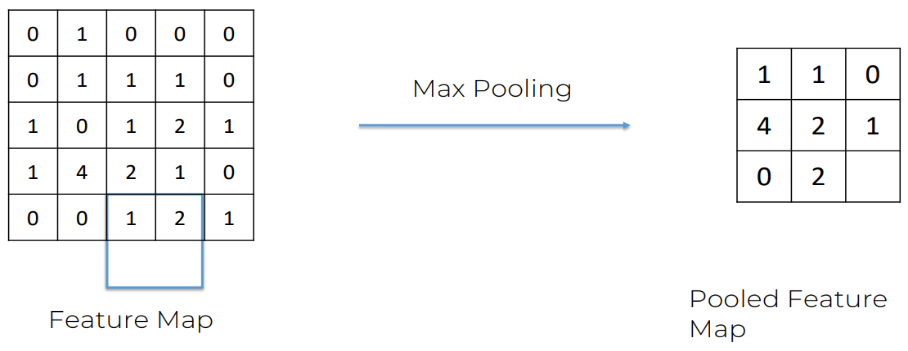
Step 2 - Max Pooling



0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

1	1	

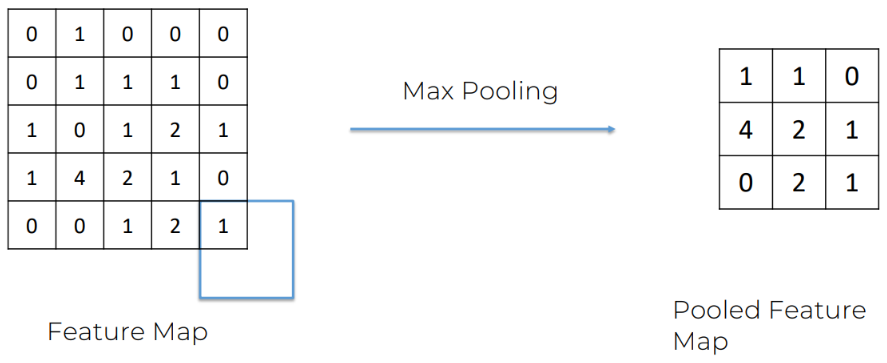
Step 2 - Max Pooling



0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

1	1	0
4	2	1
0	2	

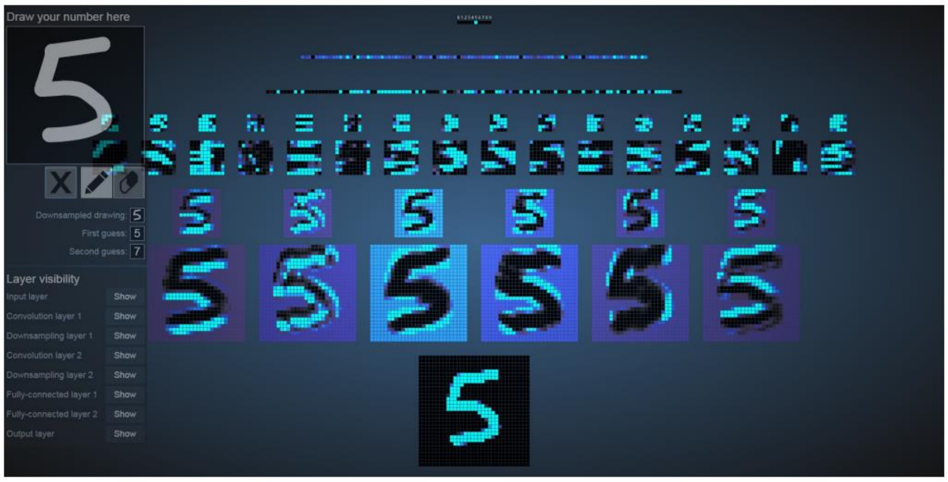
Step 2 - Max Pooling



0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

1	1	0
4	2	1
0	2	1

Example



Downsampled drawing: 5
First guess: 5
Second guess: 7

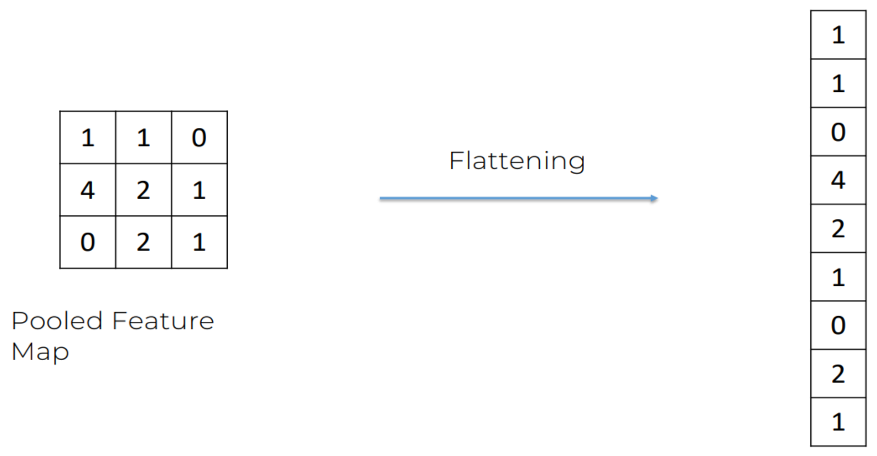
Step-3: Flattening

Step 3 - Flattening

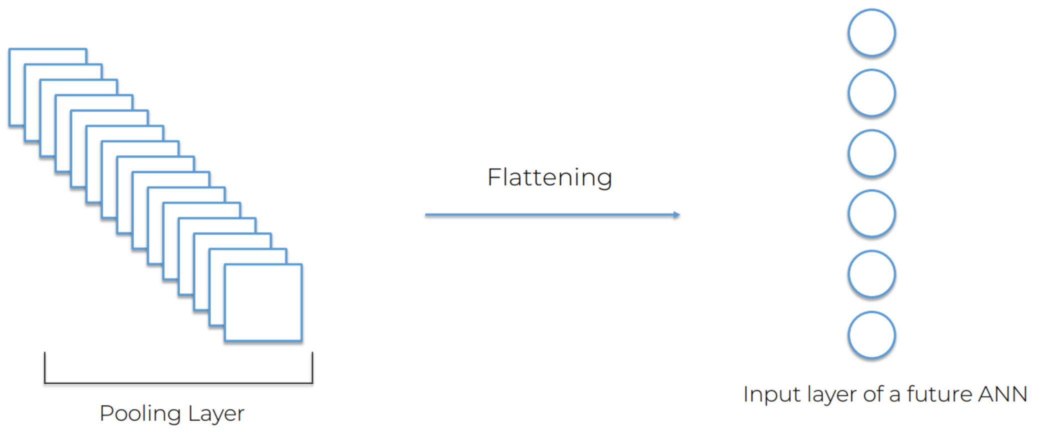
1	1	0
4	2	1
0	2	1

Pooled Feature Map

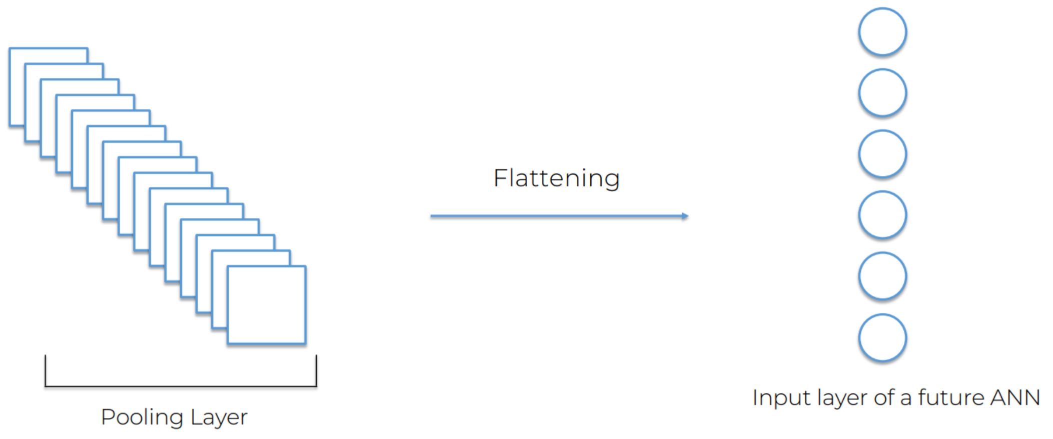
Step 3 - Flattening



Step 3 - Flattening



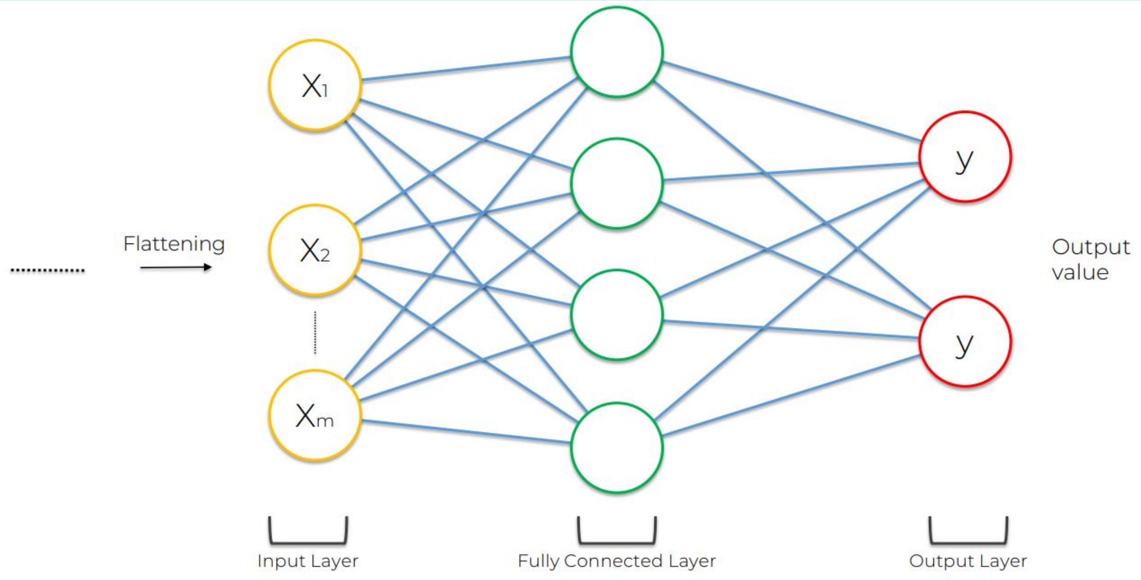
Step 3 - Flattening



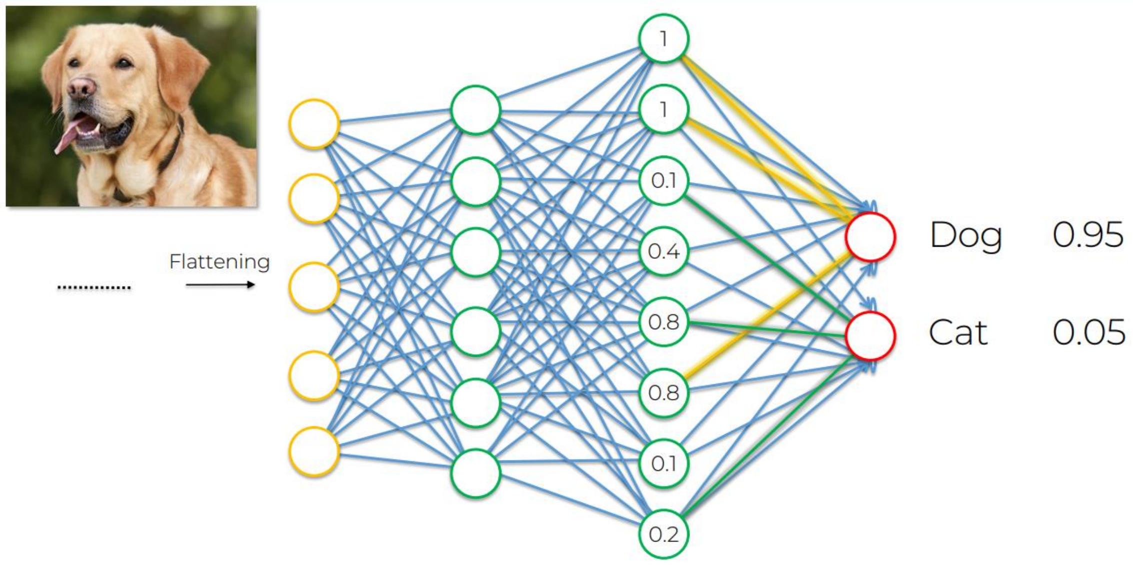
Full Connection

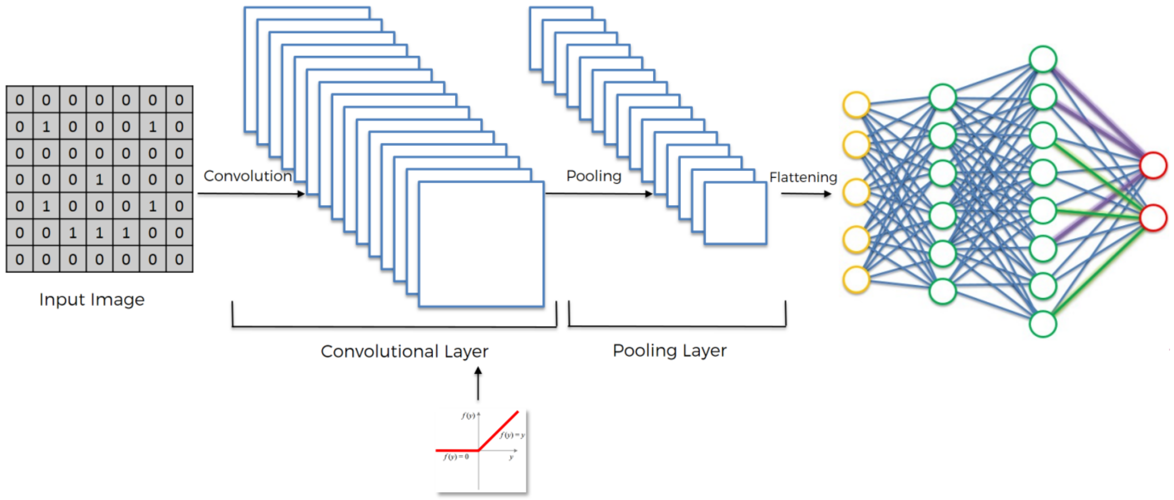
The purpose of the fully connected layer in a convolutional neural network is to detect certain features in an image. More specifically, each neuron in the fully connected layer corresponds to a specific feature that might be present in an image. The value that the neuron passes on to the next layer represents the probability that the feature is contained in the image.

Step 4 - Full Connection



Step 4 - Full Connection





Softmax and Cross-Entropy

Multiclass classification is an application of deep learning/machine learning where the model is given input and renders a categorical output corresponding to one of the labels that form the output. For example, providing a set of images of animals and classifying it among cats, dogs, horses, etc.

For this purpose, where the model outputs multiple outputs for each class, a simple logistic function (or sigmoid function) cannot be used. Thus, another activation function called the Softmax function is used along with the cross-entropy loss.

Softmax Function:

The softmax formula is represented as:

softmax function image

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where the values of z_i are the elements of the input vector and they can take any real value. The denominator of the formula is normalised term which guarantees that all the output values of the function will sum to 1, thus making it a valid probability distribution.

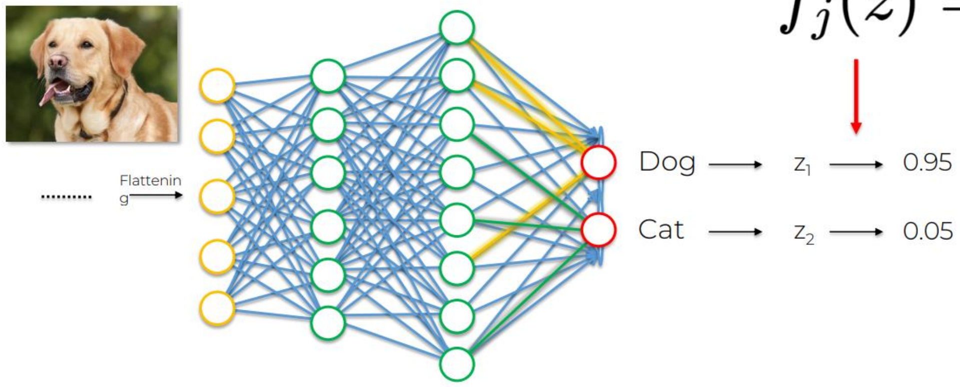
The softmax function and the sigmoid function are similar to each other. Softmax operates on vector values while the sigmoid takes scalar values. Thus, we can say that sigmoid function is a specific case of the softmax function and it is for a classifier with only two input classes. The logistic function, often known as the logistic sigmoid function, is the most common object of the word “sigmoid function” in the context of machine learning. Mathematically, it is defined by:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

The above function is used for classification between 2 classes, i.e., 1 and 0. In the case of Multiclass classification, the softmax function is used. The softmax converts the output for each class to a probability value (between 0-1), which is exponentially normalized among the classes.

Softmax & Cross-Entropy

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

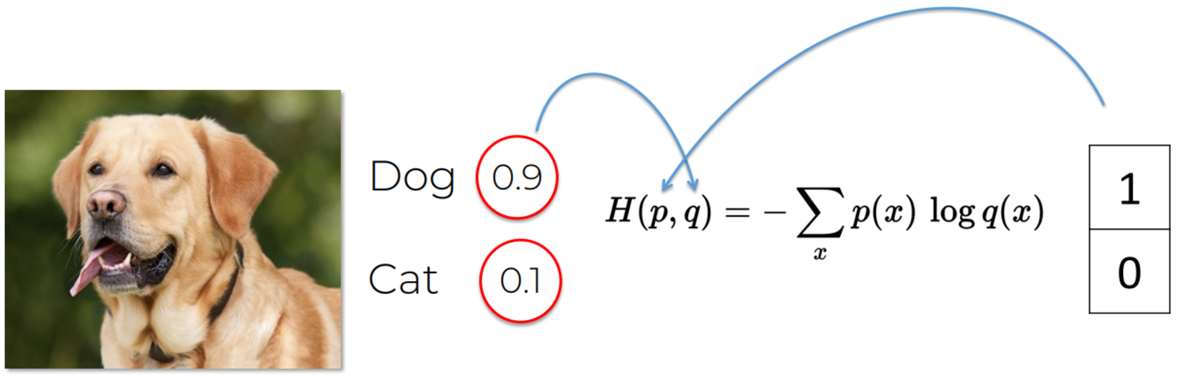


Softmax & Cross-Entropy

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

$$H(p,q) = -\sum_x p(x)\,\log q(x)$$

Softmax & Cross-Entropy



$$H(p,q) = -\sum_x p(x)\,\log q(x)$$

Softmax & Cross-Entropy

			NN1	NN2
	Dog	1	0.9	0.6
	Cat	0	0.1	0.4
	Dog	0	0.1	0.3
	Cat	1	0.9	0.7
	Dog	1	0.4	0.1
	Cat	0	0.6	0.9