Note: This article can be better read on my Kaggle Notebook, 🧠 **Convolutional Neural Network From Scratch**.

Introduction

Convolutional Neural Networks (CNNs or ConvNets) are specialized neural architectures that are predominantly used for several **computer vision** tasks, such as image classification and object recognition. These neural networks harness the power of *Linear Algebra,* specifically through convolution operations, to identify patterns within images.

Convolutional neural networks have three main kinds of layers, which are:

• Convolutional layer

• Pooling layer

• Fully-connected layer

The convolutional layer is the first layer of the network, while the fully-connected layer is the final layer, responsible for the output. The first convolutional layer may be followed by several additional convolutional layers or pooling layers; and with each new layer, the more complex is the CNN.

As the CNN gets more complex, the more it excels in identifying greater portions of the image. Whereas earlier layers focus on the simple features, such as colors and edges; as the image progresses through the network, the CNN starts to recognize larger elements and shapes, until finally reaching its main goal.

The image below displays the structure of a CNN. We have an input image, followed by Convolutional and Pooling layers, where the feature learning process happens. Later on, we have the layers responsible for the task of classifying whether the vehicle in the input data is a car, truck, van, bicycle, etc.

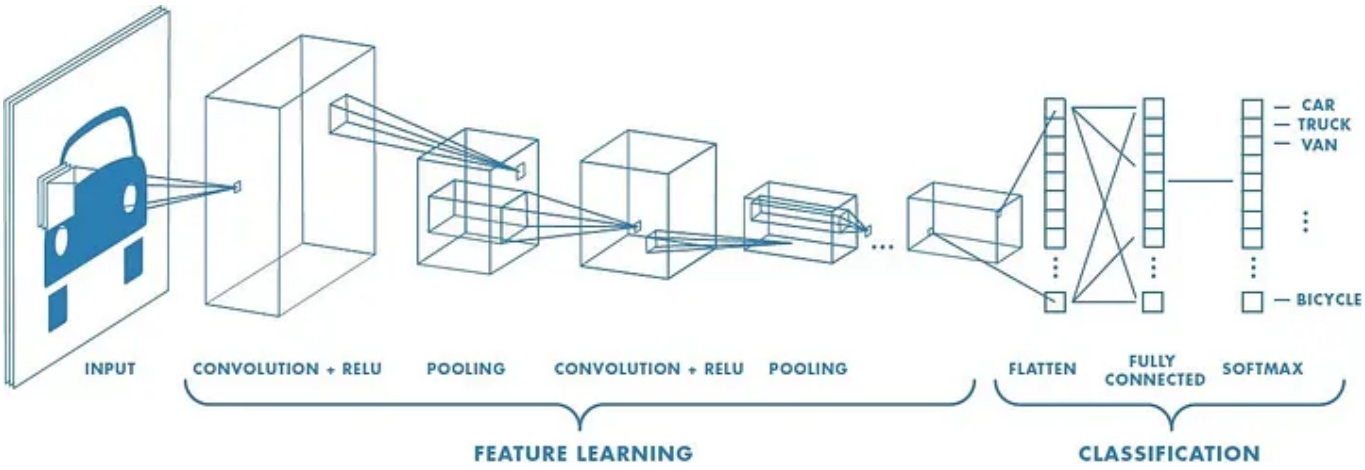


Image displaying the structure of a Convolutional Neural Network.
Source: Understanding of Convolutional Neural Network (CNN) — Deep Learning

Convolutional Layer

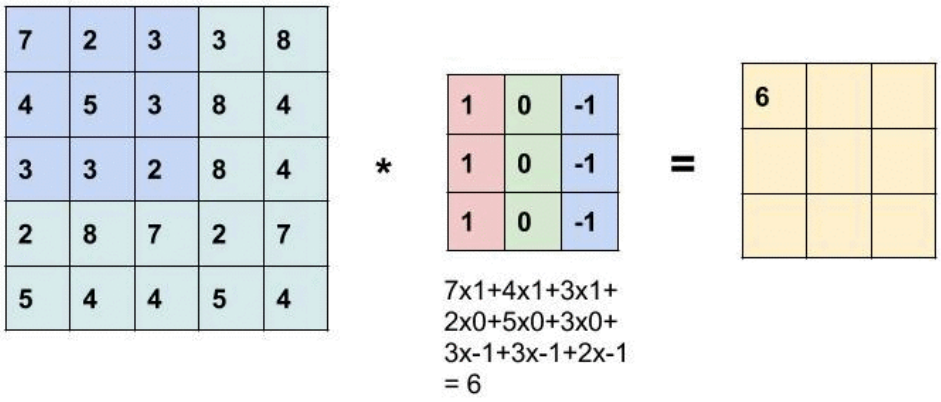
The convolutional layer is the most important layer of a CNN; responsible for dealing with the major computations. The convolutional layer includes **input data**, **a filter**, and **a feature map**.

To illustrate how it works, let's assume we have a color image as input. This image is made up of a matrix of pixels in 3D, representing the three dimensions of the image: height, width, and depth.

The filter — which is also referred to as kernel — is a two-dimensional array of weights, and is typically a 3×3 matrix. It is applied to a specific area of the image, and a **dot product** is computed between the input pixels and the

weights in the filter. Subsequently, the filter shifts by a stride, and this whole process is repeated until the kernel slides through the entire image, resulting in an output array.

The resulting output array is also known as a feature map, activation map, or convolved feature.



GIF displaying the convolutional process. First, we have a 5×55×5 matrix — pixels in the input image — with a 3×33×3 filter. The result of the operation is the output array.
Source: Convolutional Neural Networks

It is important to note that the weights in the filter remain fixed as it moves across the image. The weight values are adjusted during the training process due to backpropagation and gradient descent.

Besides the weights in the filter, we have other three important parameters that need to be set before the training begins:

• **Number of Filters:** This parameter is responsible for defining the depth of the output. If we have three distinct filters, we have three different feature maps, creating a depth of three.

• **Stride:** This is the distance, or number of pixels, that the filter moves over the input matrix.

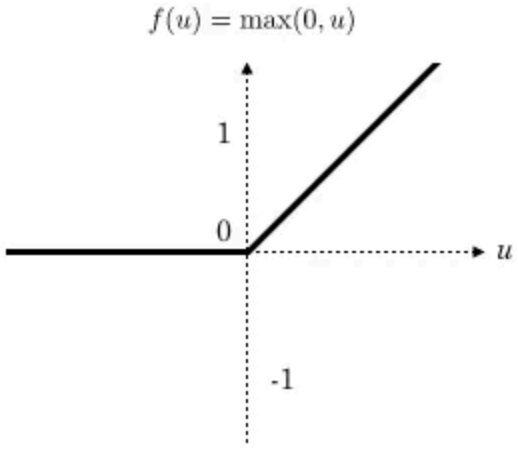
• **Zero-padding:** This parameter is usually used when the filters do not fit the input image. This sets all elements outside the input matrix to zero, producing a larger or equally sized output. There are three different kinds of padding:

■ **Valid padding:** Also known as *no padding*. In this specific case, the last convolution is dropped if the dimensions do not align.

■ **Same padding:** This padding ensures that the output layer has the exact same size as the input layer.

■ **Full padding:** This kind of padding increases the size of the output by adding zeros to the borders of the input matrix.

After each convolution operation, we have the application of a ***Rectified Linear Unit (ReLU)*** function, which transforms the feature map and introduces nonlinearity.



ReLU activation function:

$$f(u) = \begin{cases} 0 & \text{if } u \leq 0 \\ u & \text{if } u > 0 \end{cases}$$

Source: ResearchGate

As mentioned earlier, the initial convolutional layer can be followed by additional convolutional layers.

The subsequent convolutional layers can see the pixels within the receptive fields of the prior layers, which helps to extract and interpret additional patterns.

Pooling Layer

The pooling layer is responsible for reducing the dimensionality of the input. It also slides a filter across the entire input — without any weights — to populate the output array. We have two main types of pooling:

• **Max Pooling:** As the filter slides through the input, it selects the pixel with the highest value for the output array.

• **Average Pooling:** The value selected for the output is obtained by computing the average within the receptive field.

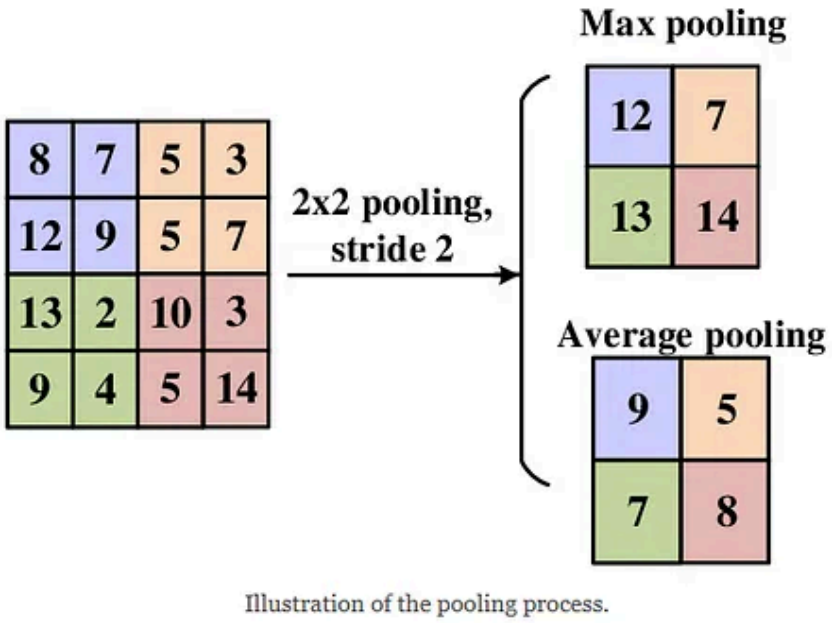


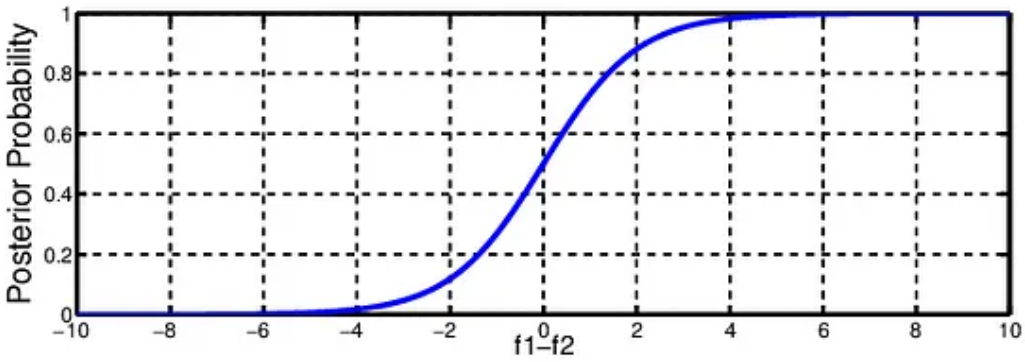
Illustration of the pooling process.

Source: ResearchGate

The pooling layer serves the purpose of reducing complexity, improving efficiency, and limiting the risk of overfitting.

Fully-connected Layer

This is the layer responsible for performing the task classification based on the features extracted during the previous layers. While both convolutional and pooling layers tend to use ***ReLU*** functions, fully-connected layers use the ***Softmax*** activation function for classification, producing a probability from *0* to *1*.



Softmax activation function graph.

Source: ResearchGate

CNNs and Computer Vision

Due to its power in image recognition tasks, CNNs have been highly effective in many fields related to ***Computer Vision***.

Computer Vision is a field of AI that enables computers to extract information from digital images, videos, and other visual inputs. Some common applications of computer vision today can be seen across several industries, including the following:

• **Social Media:** *Google*, *Meta*, and *Apple* use these systems to identify people in a photograph, making it easier to organize photo albums and tag friends.

• **Healthcare:** Computer vision models have been used to help doctors identify cancerous tumors in patients, as well as other conditions.

• **Agriculture:** Drones equipped with cameras can monitor the health of vast farmlands to identify areas that need more water or fertilizers.

• **Security:** Surveillance systems can detect unusual and suspect activities in real time.

• **Finance:** Computer vision models may be used to identify relevant patterns in candlestick charts to predict price movements.

• **Automotive:** Computer vision is an essential component of the research leading to self-driving cars.

This Article

Nowadays, there are several *pre-trained* CNNs available for many tasks. Models like ***ResNet, VGG16, InceptionV3,*** as well as many others, are highly efficient in most computer vision tasks we currently perform across industries.

In this article, however, I would like to explore the process of building a simple, yet effective, Convolutional Neural Network from scratch. For this task, I will use ***Keras*** to help us build a neural network that can accurately identify diseases in a plant through images.

I am going to use the Plant Disease Recognition Dataset, which contains 1,530 images divided into train, test, and validation sets. The images are labeled as *"Healthy", "Rust",* and *"Powdery"* to describe the conditions of the plants.

Very briefly, each class means the following:

• **Rust:** These are plant diseases caused by Pucciniales fungi, which cause severe deformities to the plant.

• **Powdery:** Powdery mildews are caused by Erysphales fungi, posing a threat to agriculture and horticulture by reducing crop yields.