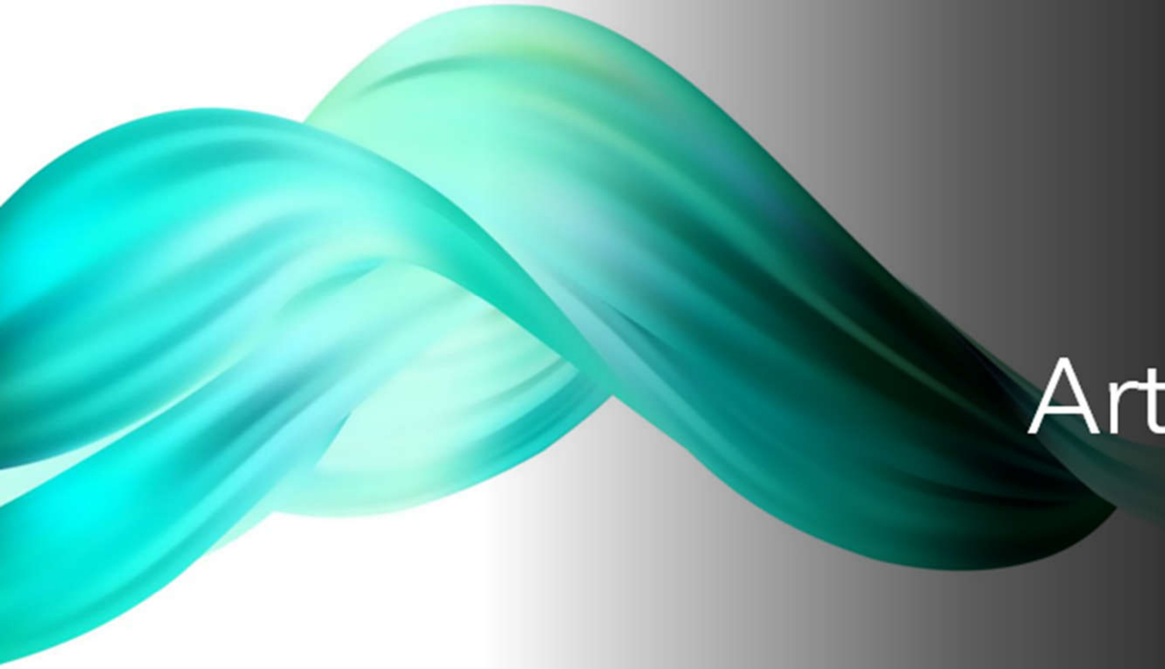


Artificial Intelligence

Dr. Mubashir Ahmad (Ph.D.)

CNN

- Since the 1950s, the early days of artificial intelligence, computer scientists have been trying to build computers that can make sense of visual data. In the ensuing decades, the field, which has become known as computer vision, saw incremental advances. In 2012, computer vision took a quantum leap when a group of researchers from the University of Toronto developed an AI model that surpassed the best image recognition algorithms by a large margin.
- The AI system, which became known as AlexNet (named after its main creator, Alex), won the 2012 ImageNet computer vision contest with an amazing 85 percent accuracy. The runner-up scored a modest 74 percent on the test.

A brief history of convolutional neural networks

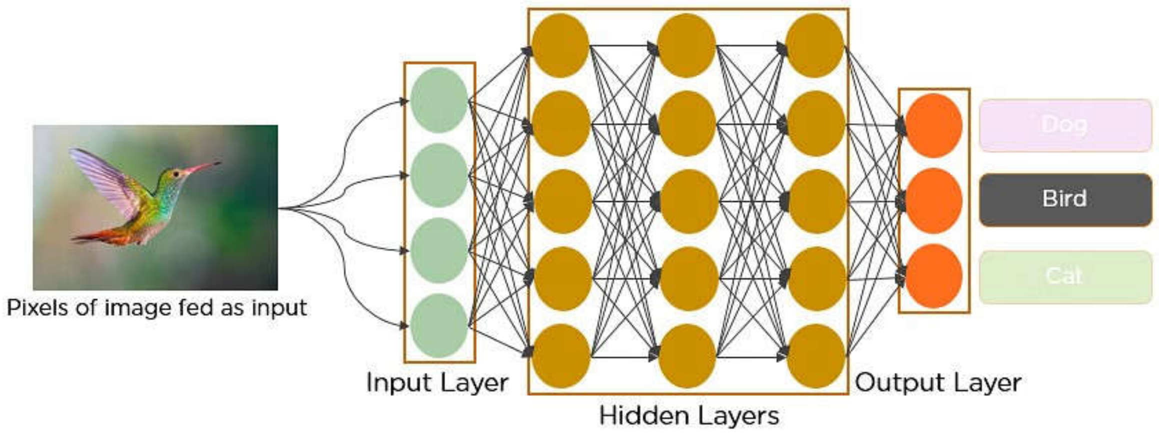
- Convolutional neural networks, also called ConvNets, were first introduced in the 1980s by Yann LeCun, a postdoctoral computer science researcher. LeCun had built on the work done by Kunihiko Fukushima, a Japanese scientist who, a few years earlier, had invented the neocognitron, a very basic image recognition neural network.
- The early version of CNNs, called LeNet (after LeCun), could recognize handwritten digits. CNNs found a niche market in banking and postal services and banking, where they read zip codes on envelopes and digits on checks.

A brief history of convolutional neural networks

- But despite their ingenuity, ConvNets remained on the sidelines of computer vision and artificial intelligence because they faced a serious problem: They could not scale. CNNs needed a lot of data and compute resources to work efficiently for large images. At the time, the technique was only applicable to images with low resolutions.
- In 2012, AlexNet showed that perhaps the time had come to revisit deep learning, the branch of AI that uses multi-layered neural networks. The availability of large sets of data, namely the ImageNet dataset with millions of labeled pictures, and vast compute resources enabled researchers to create complex CNNs that could perform computer vision tasks that were previously impossible.

How do CNNs work?

- Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value.



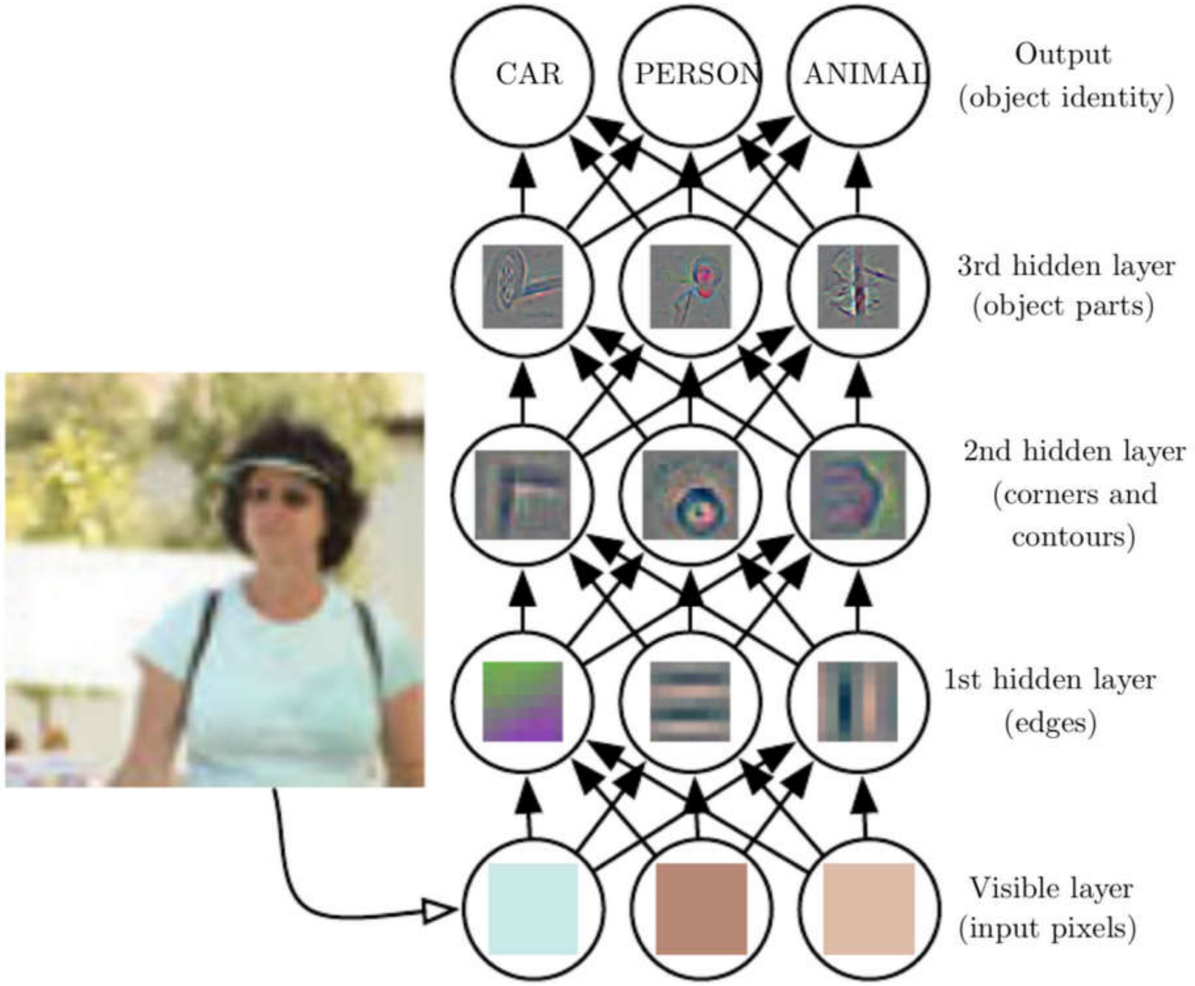
How do CNNs work?

- When you input an image into a ConvNet, each of its layers generates several activation maps. Activation maps highlight the relevant features of the image. Each of the neurons takes a patch of pixels as input, multiplies their color values by its weights, sums them up, and runs them through the activation function.
- The first (or bottom) layer of the CNN usually detects basic features such as horizontal, vertical, and diagonal edges. The output of the first layer is fed as input of the next layer, which extracts more complex features, such as corners and combinations of edges. As you move deeper into the convolutional neural network, the layers start detecting higher-level features such as objects, faces, and more.

How do CNNs work?

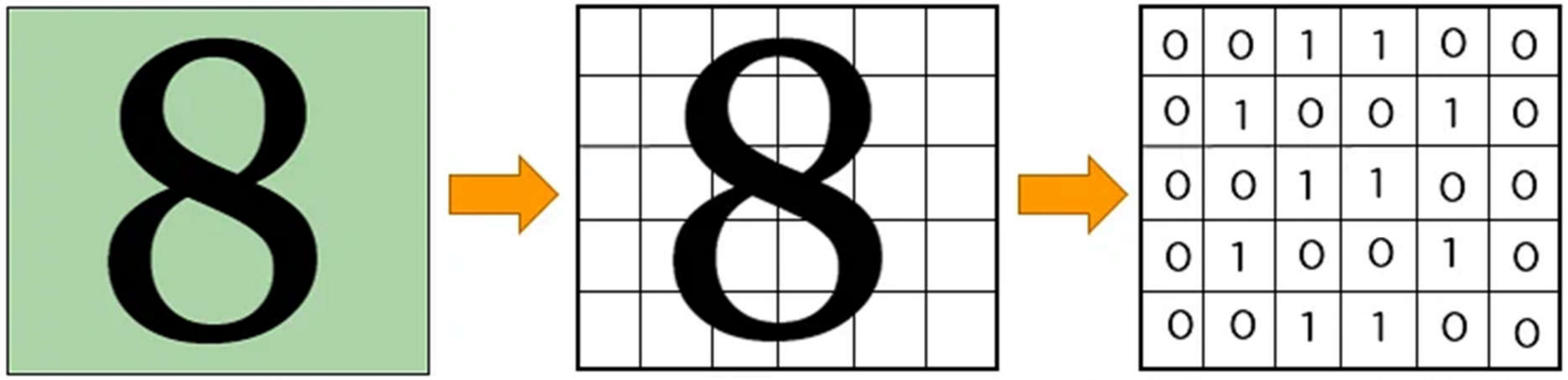
- The operation of multiplying pixel values by weights and summing them is called “convolution” (hence the name convolutional neural network). A CNN is usually composed of several convolution layers, but it also contains other components. The final layer of a CNN is a classification layer, which takes the output of the final convolution layer as input (remember, the higher convolution layers detect complex objects).
- Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a “class.” For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

How do CNNs work?



How do CNNs work?

- In CNN, every image is represented in the form of an array of pixel values.



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

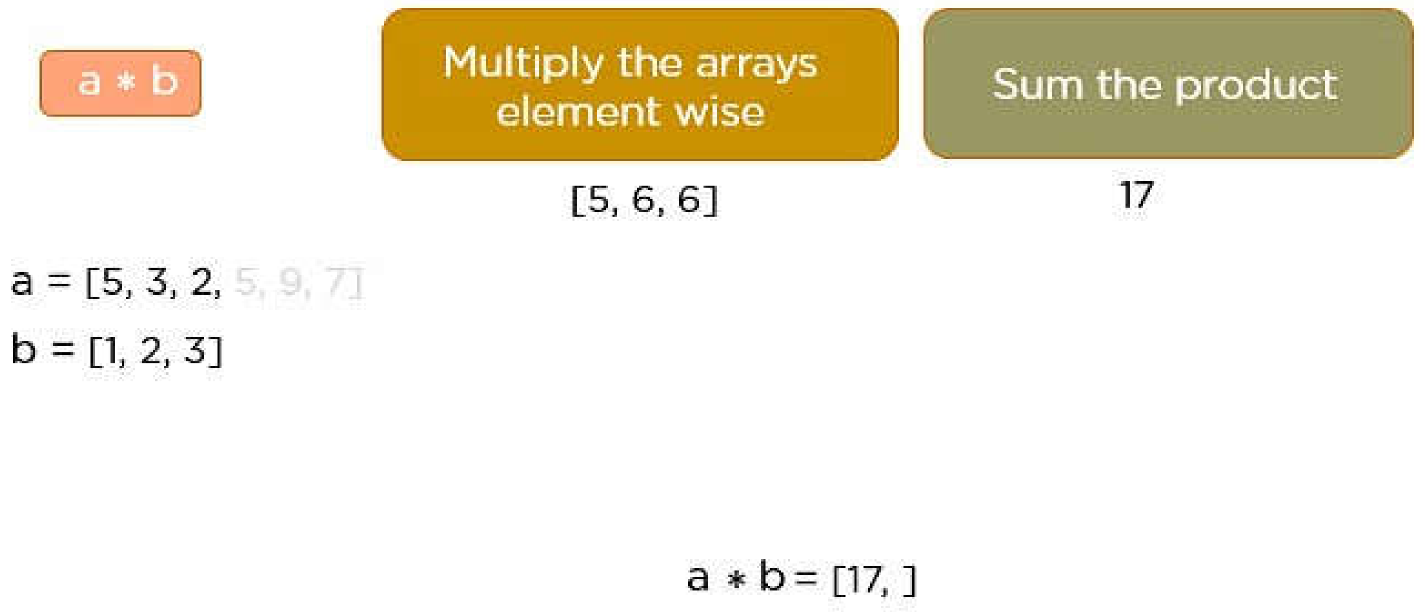
Real Image of the digit 8

Represented in the form of an array

Digit 8 represented in the form of pixels of 0's and 1's

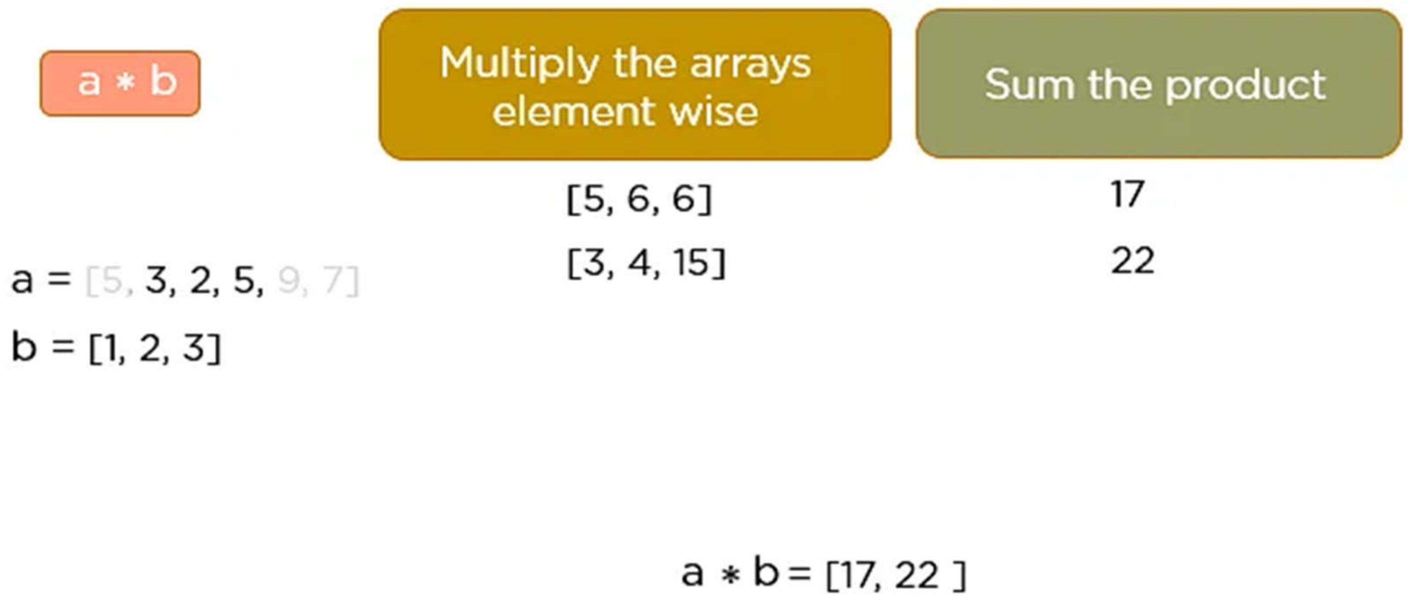
How do CNNs work?

- The convolution operation forms the basis of any convolutional neural network. Let’s understand the convolution operation using two matrices, a and b, of 1 dimension.
- a = [5,3,2,5,9,7]
- b = [1,2,3]
- In convolution operation, the arrays are multiplied element-wise, and the product is summed to create a new array, which represents a*b. The first three elements of the matrix a are multiplied with the elements of matrix b. The product is summed to get the result.



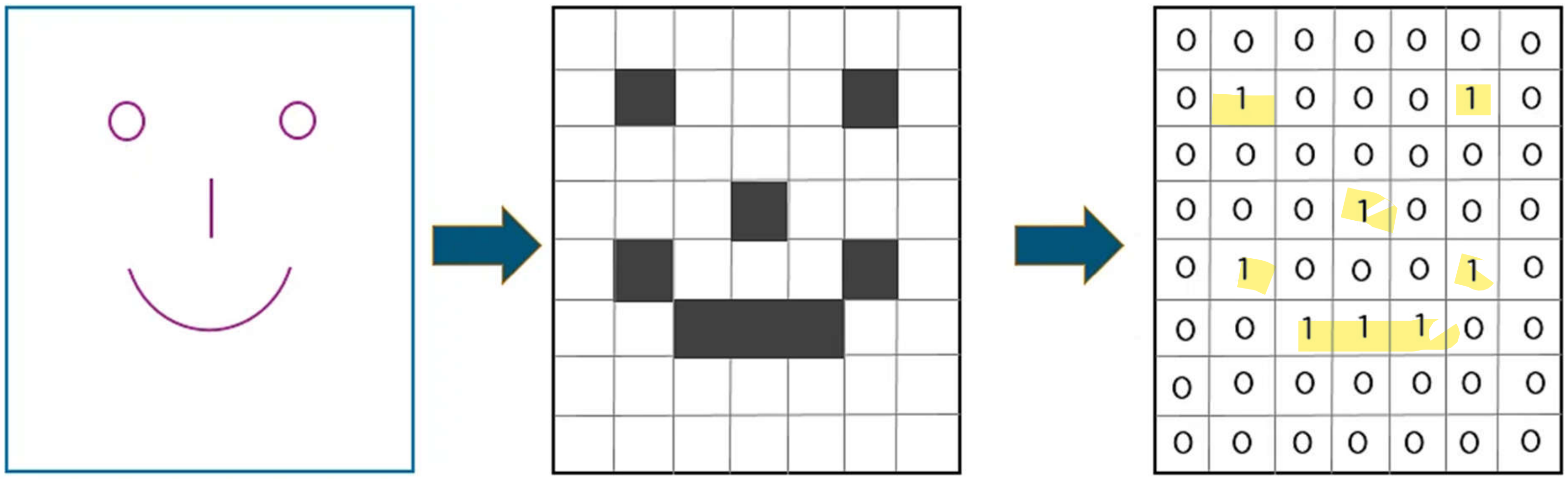
How do CNNs work?

- The next three elements from the matrix a are multiplied by the elements in matrix b, and the product is summed up.



How do CNNs work?

- Here is another example to depict how CNN recognizes an image:


Real Image

Represented in the form of black and white pixels

Image represented in the form of a matrix of numbers

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
0	0	0	0	0	0	0

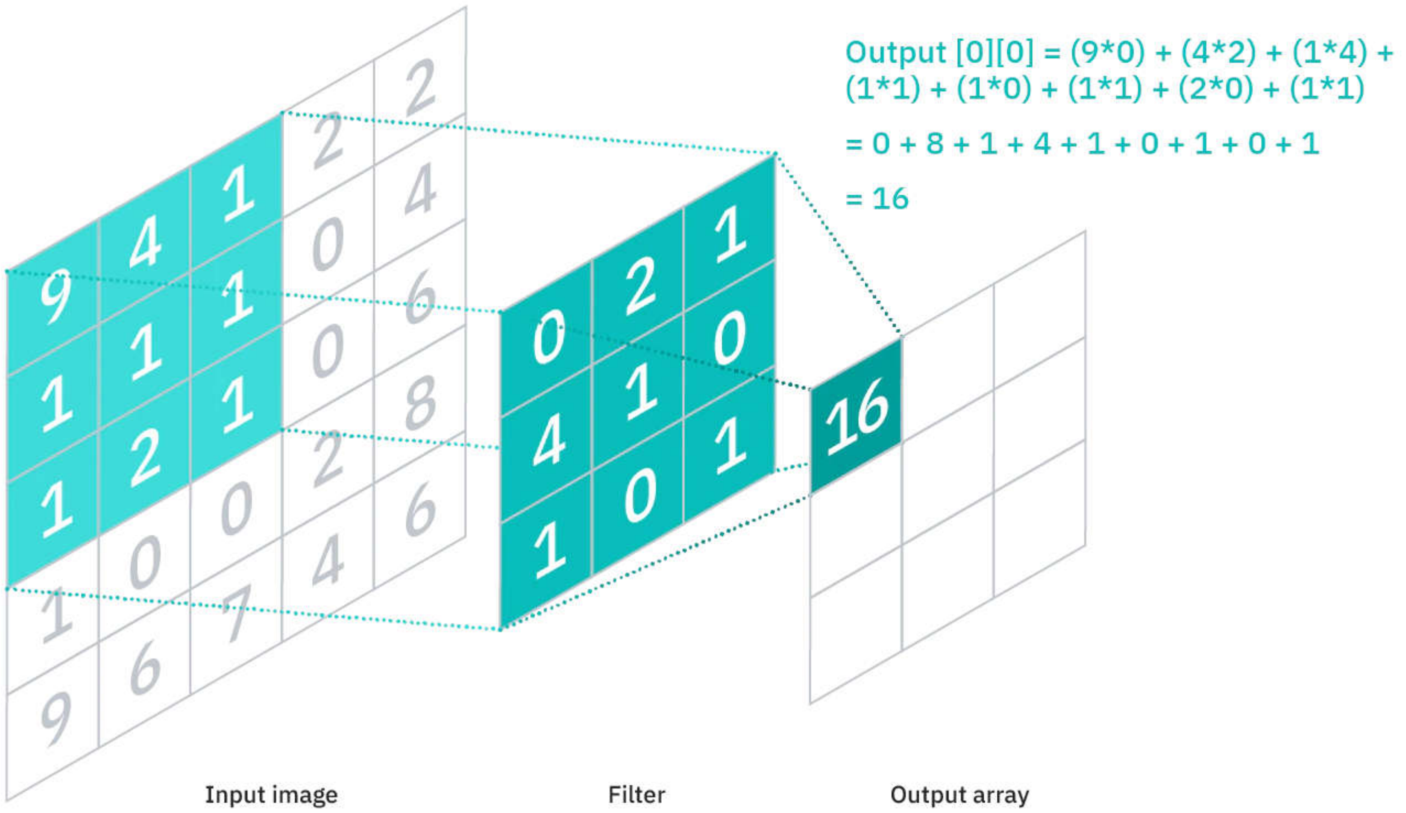
Layers used to build ConvNets

- a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet **architecture**.

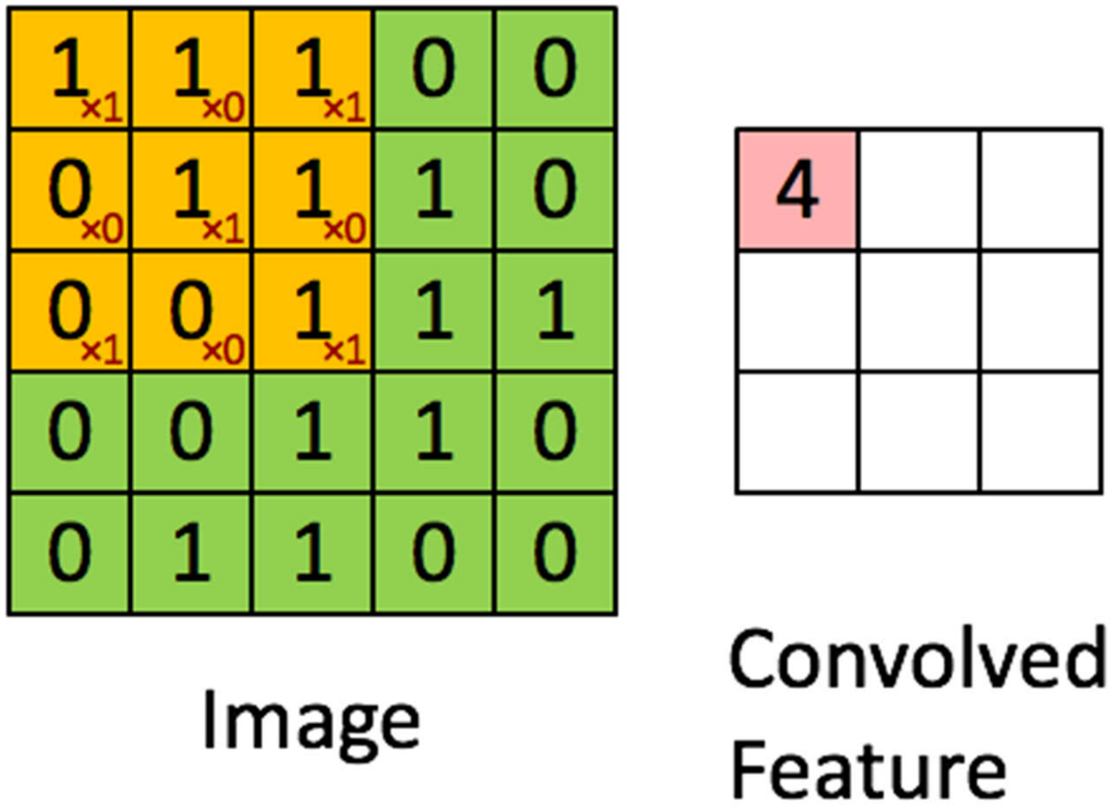
Convolutional Layer

- The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let’s assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.
- The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.

Convolutional Layer



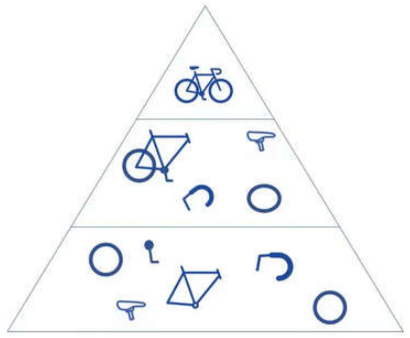
Convoluted features

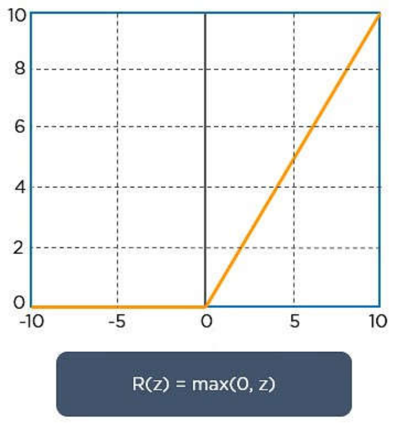


Convolutional Layer

- **Stride** is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.
- **Same padding**: This padding ensures that the output layer has the same size as the input layer
- **Full padding:** This type of padding increases the size of the output by adding zeros to the border of the input.

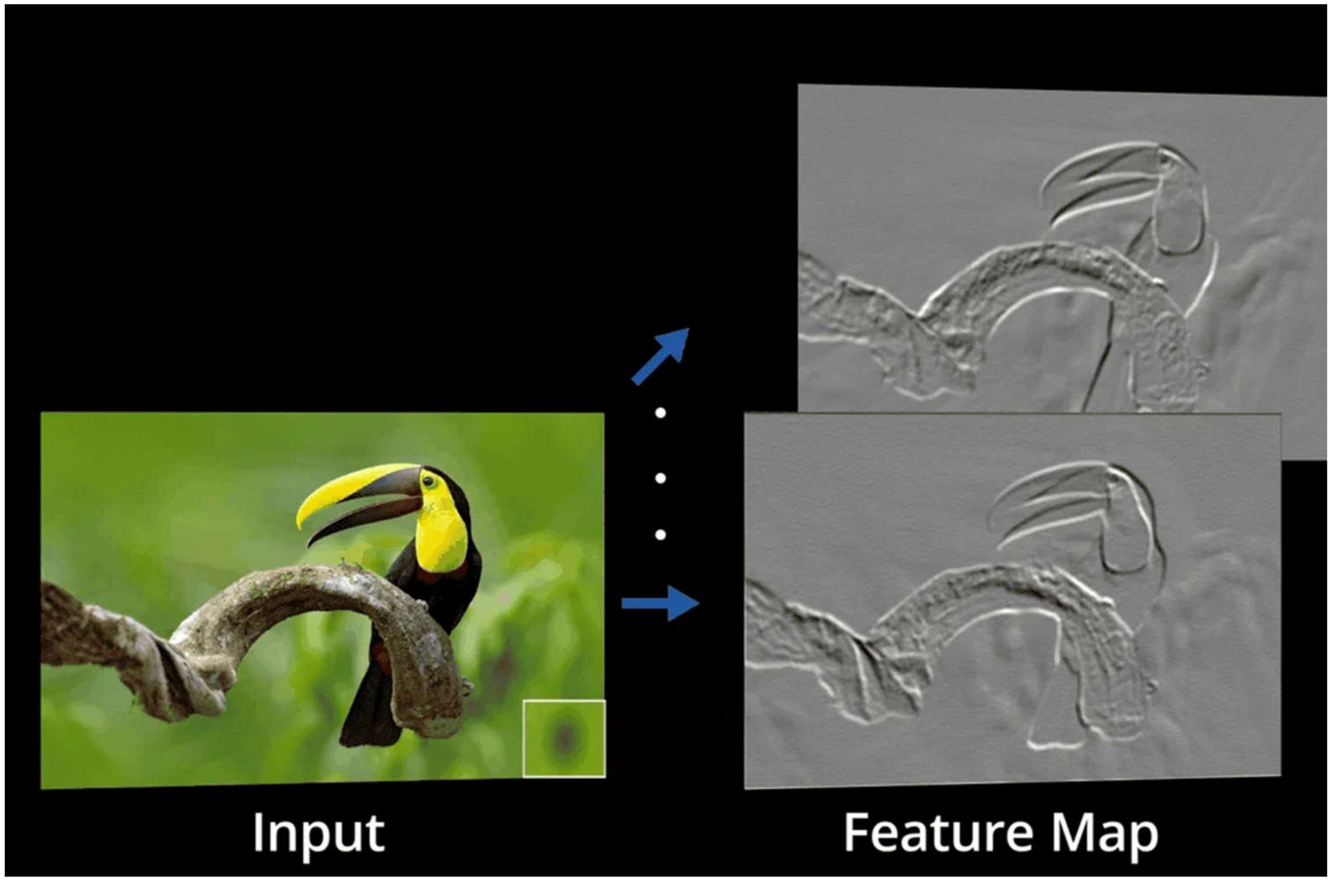
ReLU





- After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.
- As we mentioned earlier, another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers. As an example, let’s assume that we’re trying to determine if an image contains a bicycle. You can think of the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, etc. Each individual part of the bicycle makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.

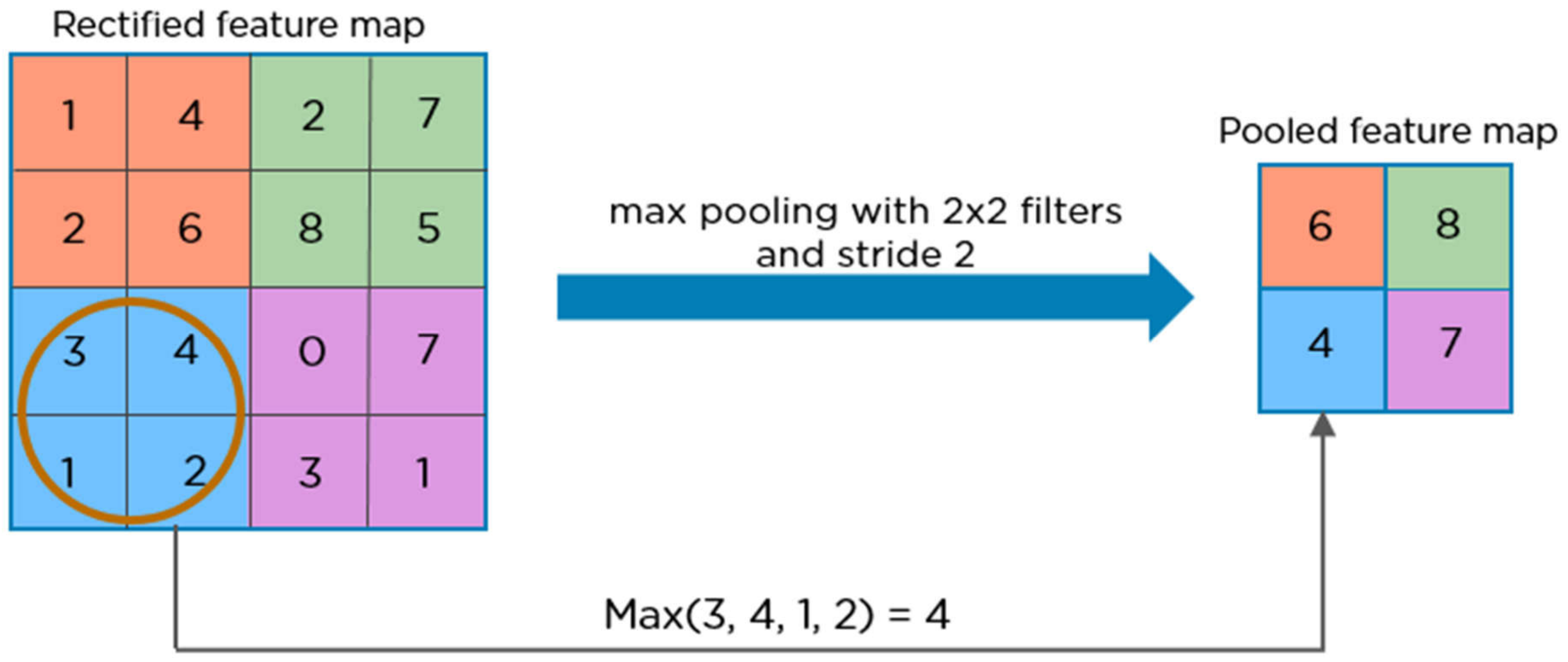
Convolutional Layer



Pooling Layer

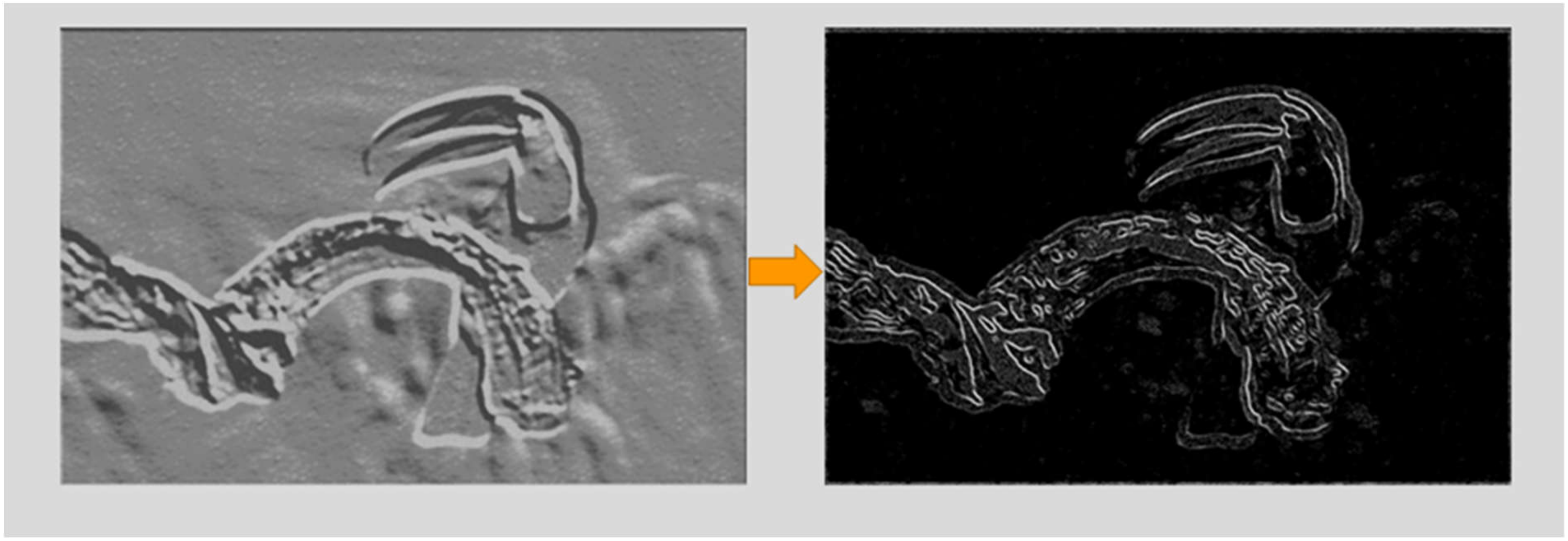
- Pooling layers, also known as downsampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:
- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.
- While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.

Pooling Layer



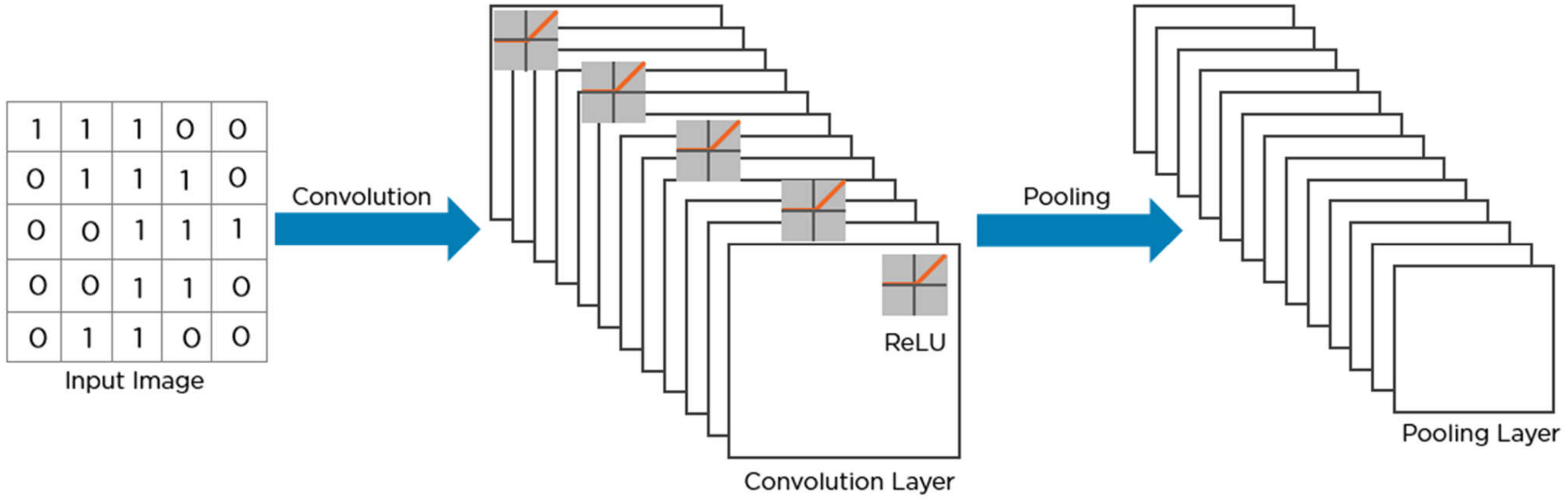
Pooling Layer

- The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes.



Pooling Layer

- Here’s how the structure of the convolution neural network looks so far:

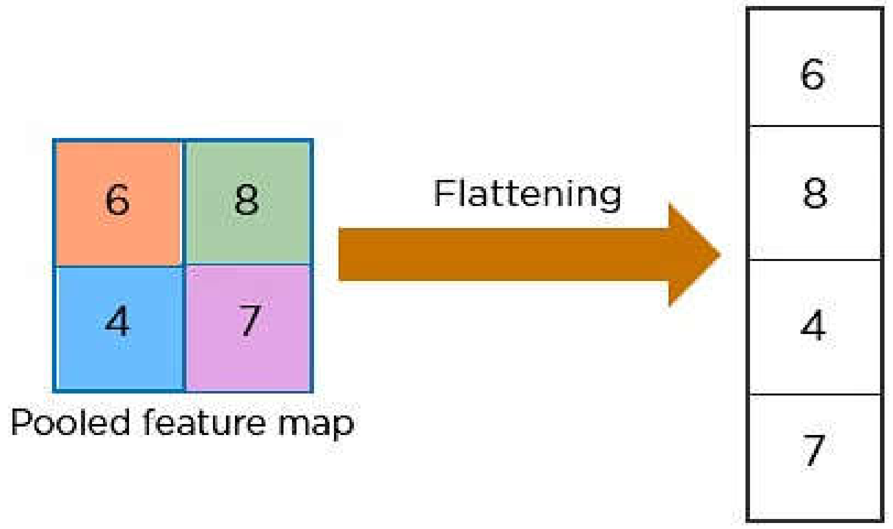


Fully-Connected Layer

- As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.
- This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

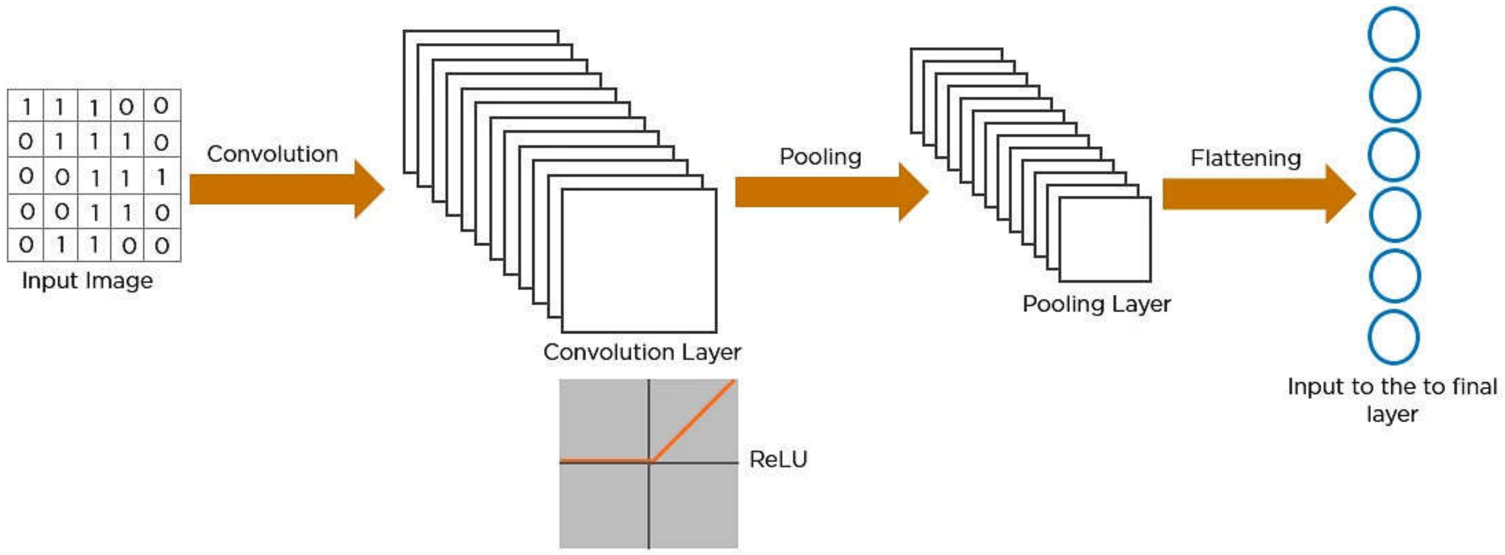
Fully-Connected Layer

- The next step in the process is called flattening. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.2-dimensional

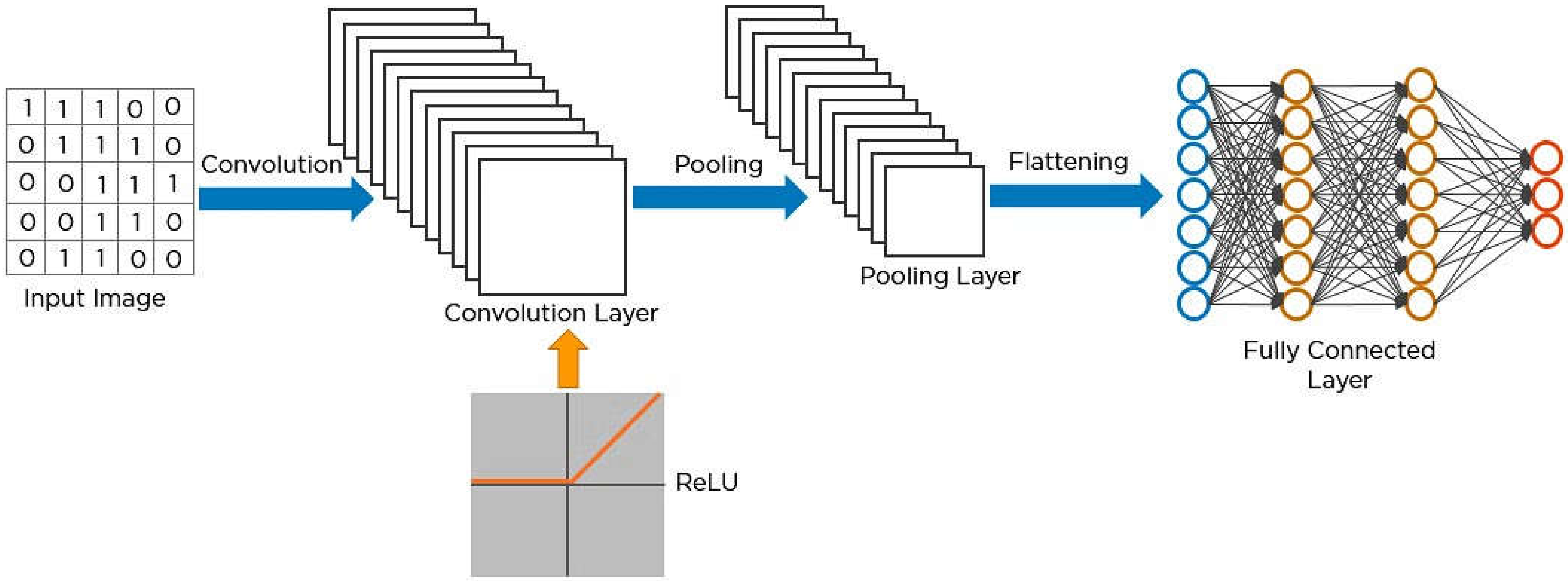


Fully-Connected Layer

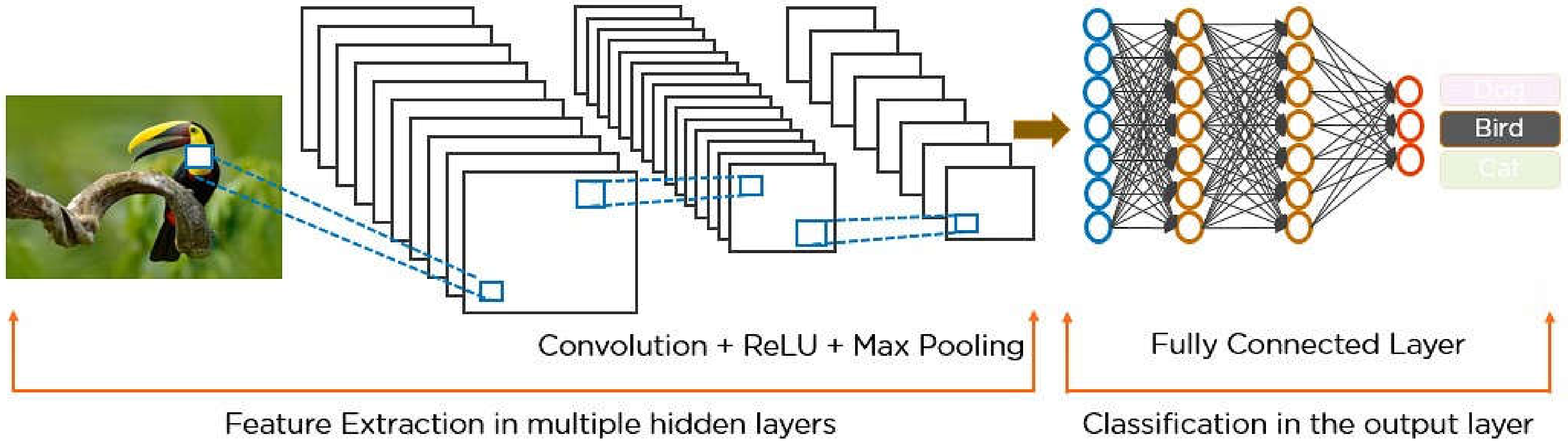
- The flattened matrix is fed as input to the fully connected layer to classify the image.



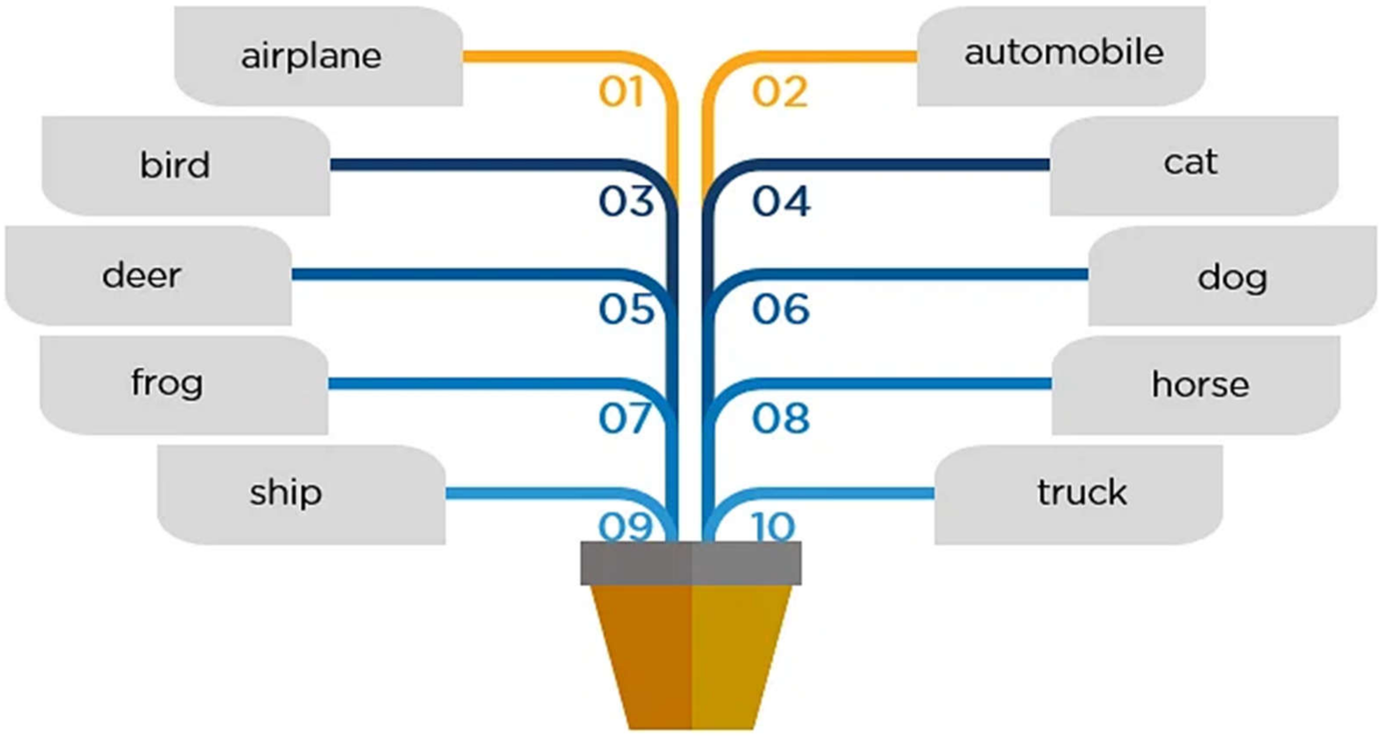
Fully-Connected Layer



- Here's how exactly CNN recognizes a bird:
- The pixels from the image are fed to the convolutional layer that performs the convolution operation
- It results in a convolved map
- The convolved map is applied to a ReLU function to generate a rectified feature map
- The image is processed with multiple convolutions and ReLU layers for locating the features
- Different pooling layers with various filters are used to identify specific parts of the image
- The pooled feature map is flattened and fed to a fully connected layer to get the final output


Dog
Bird
Cat
Convolution + ReLU + Max Pooling
Fully Connected Layer
Feature Extraction in multiple hidden layers
Classification in the output layer

Use case implementation using CNN-CIFAR-10 dataset



Designing of a CNN Model

- Make a model with input size of 32*32 with channel 1.
- First convolutional layer has a filter size of 7*7, stride 2, zero padding none with channel 32.
- First Maxpooling layer with filter size of 3*3, stride 2 and zero padding none with channel 32.
- Second convolutional layer having a filter size of 5*5, 1 stride, zero padding is 2 with channel 192.
- Second Maxpooling layer with filter size of 3*3, stride 2 and zero padding non with channel 192.
- Third convolutional layer having a filter size of 3*3, 1 stride, zero padding is 1 with channel 256.
- Third Maxpooling layer with filter size of 2*2, stride 2 and zero padding non with channel 256.
- Then 2 fully connected layers with channel 4096.
- What will be the output of each convolutional and pooling layer?
- What will be the total parameters of all layers?

Formulas to calculate the output size

- Convolutional layer = [(W−K+2P)/S]+1
- W is the size of an image
- K is the kernel or filter
- P is the padding
- S is the stride
- [(32 - 7 + 2*0) / 2] + 1 = 13*13
- [(13 - 3 + 2*0) / 2] + 1 = 10/2 + 1 = 6*6
- [(6 - 5 + 2*2) / 1] + 1 = 5 + 1 = 6*6
- [(6 - 3 + 2*0) / 2] + 1 = 3/2 + 1 = 2*2
- [(2 - 1 + 2*1) / 2 + 1 = 1 + 1 = 2*2
- [(2 - 2 + 2*0) / 2] + 1 = 1*1
- Formula for calculating parameters = Input depth × kernel × kernel × output depth + output depth

- D_{in} as the input depth (number of input channels),
- D_{out} as the output depth (number of filters in the convolutional layer),
- K as the size of the square filter (kernel),
- B as a binary value representing the use (1) or absence (0) of bias terms.

The number of learnable parameters in a convolutional layer is calculated as follows:

$$\text{Number of parameters} = D_{\text{in}} \times D_{\text{out}} \times K^2 + B \times D_{\text{out}}$$

For a fully connected layer, the number of parameters is calculated as follows:

$$\text{Number of parameters} = (\text{input depth} + 1) \times \text{output depth}$$

Layer Type	Kernel size	Stride	Padding	Output	Depth	Parameters
Input	-	-	-	32 x 32	1	
Convolution	7	2	0	13 x 13	32	1×7×7×32 + 32 = 1600
Max-Pooling	3	2	0	6 x 6	32	
Convolution	5	1	2	6 x 6	192	32×5×5×192 + 192 = 153792
Max-Pooling	3	2	0	2 x 2	192	
Convolution	3	1	1	2 x 2	256	192×3×3×256 + 256 = 442560
Max-Pooling	2	2	0	1 x 1	256	
Fully connected	1	-	-	1 x 1	4096	(256+1)×4096 = 1,052,672
Fully connected	1	-	-	1 x 1	2	(4096 +1)* 2= 8,194
Softmax	-	-	-	1 x 1	2	
Total	-	-	-	1 x 1	2	1658818