Vol 57, No 1—February 2023

Introduction to Convolutional Neural Networks: What Is Machine Learning?—Part 1

Ole Dreessen, Staff Engineer, Field Applications

Abstract

The world of artificial intelligence (AI) is rapidly evolving, and AI is increasingly enabling applications that were previously unattainable or very difficult to implement. This series of articles explains convolutional neural networks (CNNs) and their significance in machine learning within AI systems. CNNs are powerful tools for extracting features from complex data. This includes, for example, complex pattern recognition in audio signals or images. This article discusses the advantages of CNNs vs. classic linear programming. A subsequent article, "Training Convolutional Neural Networks: What Is Machine Learning?—Part 2" will discuss how CNN models are trained. Part 3 will examine a specific use case to test the model using a dedicated AI microcontroller.

What Are Convolutional Neural Networks?

Neural networks are systems, or structures of neurons, that enable AI to better understand data, allowing it to solve complex problems. While there are numerous network types, this series of articles will solely focus on convolutional neural networks (CNNs). The main application areas for CNNs are pattern recognition and classification of objects contained in input data. CNNs are a type of artificial neural network used in deep learning. Such networks are composed of an input layer, several convolutional layers, and an output layer. The convolutional layers are the most important components, as they use a unique set of weights and filters that allow the network to extract features from the input data. Data can come in many different forms, such as images, audio, and text. This feature extraction process enables the CNN to identify patterns in the data. By extracting features from data, CNNs enable engineers to create more effective and efficient applications. To better understand CNNs, we will first discuss classic linear programming.

Linear Program Execution in Classic Control Engineering

In control engineering, the task lies in reading data from one or more sensors, processing it, responding to it according to rules, and displaying or forwarding the results. For example, a temperature regulator measures temperature every second through a microcontroller unit (MCU) that reads the data from the temperature sensor. The values derived from the sensor serve as input data for the closed-loop control system and are compared with the setpoint temperature in a loop. This is an example of a linear execution, run by the MCU. This technique delivers conclusive outcomes based on a set of preprogrammed and actual values. In contrast, probabilities play a role in the operation of AI systems.

Complex Pattern and Signal Processing

There are also numerous applications that work with input data that first must be interpreted by a pattern recognition system. Pattern recognition can be applied to different data structures. In our examples, we restrict ourselves to one- and two-dimensional data structures. Some examples are as follows: audio signals, electrocardiograms (ECGs), photoplethysmographs (PPGs), vibrations for one-dimensional data and images, thermal images, and waterfall charts for two-dimensional data.

In pattern recognition used for the cases mentioned, conversion of the application in classic code for the MCU is extremely difficult. An example is the recognition of an object (for example, a cat) in an image. In this case, it doesn't make a difference if the image to be analyzed is from an earlier recording or one just read by a camera sensor. The analysis software performs a rules-based search for patterns that can be attributed to those of a cat: the typical pointed ears, the triangular nose, or the whiskers. If these features can be recognized in the image, the software reports a cat find. Some questions arise here: What would the pattern recognition system do if the cat is only shown from the back? What would happen if it didn't have any whiskers or lost its legs in an accident? Despite the unlikelihood of these exceptions, the pattern recognition code would have to check a large number of additional rules covering all possible anomalies. Even in our simple example, the rules set by the software would quickly become extensive.

How Machine Learning Replaces Classic Rules

The idea behind AI is to mimic human learning on a small scale. Instead of formulating a large number of if-then rules, we model a universal pattern recognition machine. The key difference between the two approaches is that AI, in contrast to a set of rules, does not deliver a clear result. Instead of reporting "I recognized a cat in the image," machine learning yields the result "There is a 97.5% probability that the image shows a cat. It could also be a leopard (2.1%) or a tiger (0.4%)." This means that the developer of such an application must decide at the end of the pattern recognition process. A decision threshold is used for this.

Another difference is that a pattern recognition machine is not equipped with fixed rules. Instead, it is trained. In this learning process, a neural network is shown a large number of cat images. In the end, this network is capable of independently recognizing whether there is a cat in an image or not. The crucial point is that future recognition is not restricted to already known training images. This neural network needs to be mapped into an MCU.

What Exactly Does a Pattern Recognition Machine Look Like on the Inside?

A network of neurons in AI resembles its biological counterpart in the human brain. A neuron has several inputs and just one output. Basically, such a neuron is nothing other than a linear transformation of the inputs—multiplication of the inputs by numbers (weights, w) and addition of a constant (bias, b)—followed by a fixed nonlinear function that is also known as an activation function.[1] This activation function, as the only nonlinear component of the network, serves to define the value range in which an artificial neuron fires. The function of a neuron can be described mathematically as

$$Out = f(w*x+b) \tag{1}$$

where f = the activation function, w = weight, x = input data, and b = bias. The data can occur as individual scalars, vectors, or in matrix form. Figure 1 shows a neuron with three inputs and a ReLU[2] activation function. Neurons in a network are always arranged in layers.

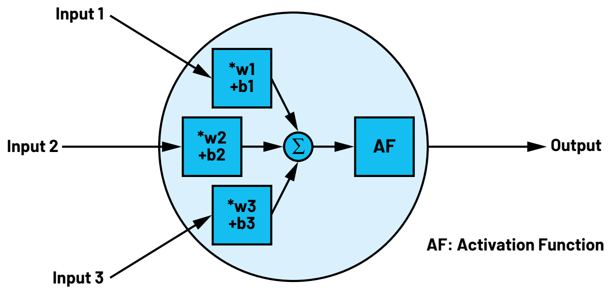

Figure 1. A neuron with three inputs and one output.

As mentioned, CNNs are used for pattern recognition and classification of objects contained in input data. CNNs are divided into various sections: one input layer, several hidden layers, and one output layer. A small network with three inputs, one hidden layer with five neurons, and one output layer with four outputs can be seen in Figure 2. All neuron outputs are connected to all inputs in the next layer. The network shown in Figure 2 is not able to process meaningful tasks and is used here for demonstration purposes only. Even in this small network, there are 32 biases and 32 weights in the equation used to describe the network.

A CIFAR neural network is a type of CNN that is widely used in image recognition tasks. It consists of two main types of layers: convolutional layers and pooling layers, which are both utilized to great effect in the training of neural networks. The convolutional layer uses a mathematical operation called *convolution* to identify patterns within an array of pixel values. Convolution occurs in hidden layers, as can be seen in Figure 3. This process is repeated multiple times until the desired level of accuracy is achieved. Note that the output value from a convolution operation is always especially high if the two input values to be compared (image and filter, in this case) are similar. This is called a filter matrix, which is also known as a filter kernel or just a filter. The results are then passed into the pooling layer, which generates a feature map—a representation of the input data that identifies important features. This is considered to be another filter matrix. After training—in the operational state of the network—these feature maps are compared with the input data. Because the feature maps hold object class-specific characteristics that are compared with the input images, the output of the neurons will only trigger if the contents are alike. By combining these two approaches, the CIFAR network can be used to recognize and classify various objects in an image with high accuracy.

[1] The functions sigmoid, tanh, or ReLU are typically used.
[2] ReLU: rectified linear unit. Negative input values for this function are output as zero and values greater than zero with their input values.

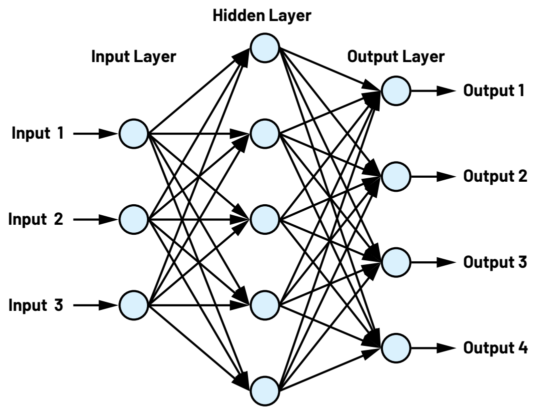

Figure 2. A small neural network.

CIFAR-10 is one specific dataset commonly used for training CIFAR neural networks. It consists of 60,000 32 × 32 color images broken up into 10 classes that were collected from various sources like web pages, newsgroups, and personal imagery collections. Each class has 6000 images divided equally between training, testing, and validation sets, making it an ideal set for testing new computer vision architectures and other machine learning models.

The main difference between convolutional neural networks and other types of networks is the way in which they process data. Through filtering, the input data are successively examined for their properties. As the number of convolutional layers connected in series increases, so does the level of detail that can be recognized. The process starts with simple object properties, such as edges or points, after the first convolution and goes on to detailed structures, such as corners, circles, rectangles, etc., after the second convolution. After the third convolution, features represent complex patterns that resemble parts of objects in images and that usually are unique to the given object class. In our initial example, these are the whiskers or ears of a cat. Visualization of the feature maps—which can be seen in Figure 4—is not necessary for the application itself, but it helps in the understanding of the convolution.

Even small networks such as CIFAR consist of hundreds of neurons in each layer and many layers connected in series. The number of necessary weights and biases grows rapidly with increasing complexity and size of the network. In the CIFAR-10 example pictured in Figure 3, there are already 200,000 parameters that require a determined set of values during the training process. The feature maps can be further processed by pooling layers that reduce the number of parameters that need to be trained while still preserving important information.

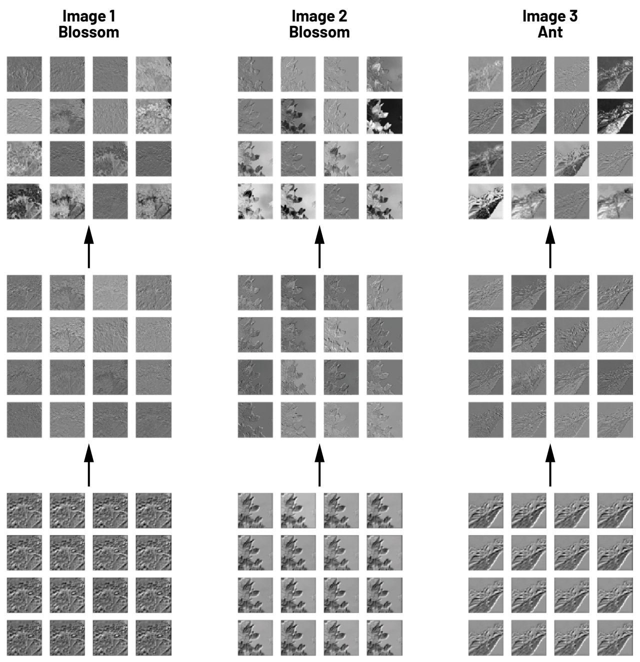

Figure 4. Feature maps for a CNN.

As mentioned, after every convolution in a CNN, pooling, often also referred to in literature as subsampling, often occurs. This serves to reduce the dimensions of the data. If you look at the feature maps in Figure 4, you notice that large regions contain little to no meaningful information. This is because the objects

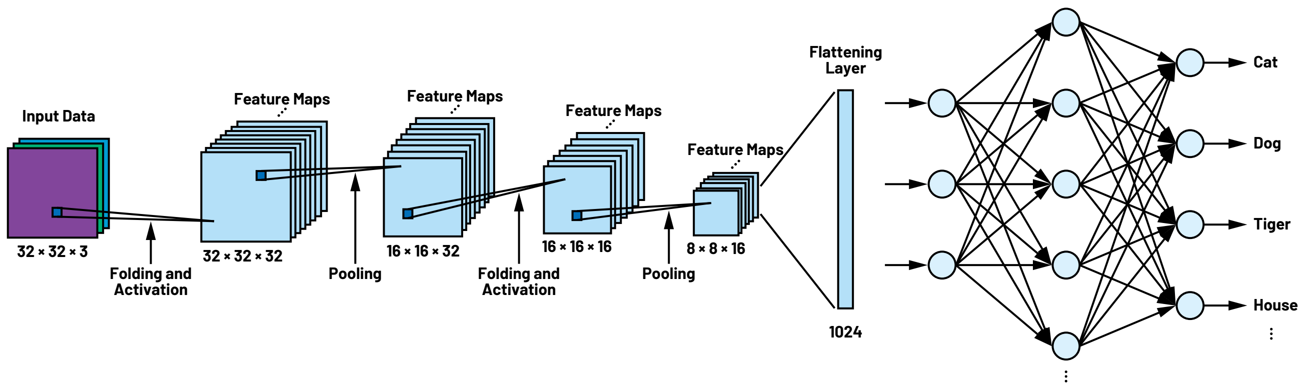

Figure 3. A model of the CIFAR network trained with the CIFAR-10 data set.

do not make up the entire image, but only a small part of it. The remaining part of the image is not used in this feature map and is hence not relevant for the classification. In a pooling layer, both the pooling type (maximum or average) and the window matrix size are specified. The window matrix is moved in a stepwise manner across the input data during the pooling process. In maximum pooling, for example, the largest data value in the window is taken. All other values are discarded. In this way, the data is continuously reduced in number, and in the end, it forms, together with the convolutions, the unique properties of the respective object class.

However, the result of these convolution and pooling groups is a large number of two-dimensional matrices. To achieve our actual goal of classification, we convert the two-dimensional data to a long one-dimensional vector. The conversion is done in a so-called flattening layer, which is followed by one or two fully connected layers. The neurons in the last two layer types are similar to the structure shown in Figure 2. The last layer of our neural network has exactly as many outputs as there are classes to be distinguished. In addition, in the last layer, the data are also normalized to yield a probability distribution (97.5% cat, 2.1% leopard, 0.4% tiger, etc.).

This concludes the modeling of our neural network. However, the weights and contents of the kernel and filter matrices are still unknown and must be determined through network training in order for the model to work. This will be explained in the subsequent article, "Training Convolutional Neural Networks: What Is Machine Learning?—Part 2." Part 3 will explain the hardware implementation for the neural network we have discussed (for cat recognition, as an example). For this, we will use the MAX78000 artificial intelligence microcontroller with a hardware-based CNN accelerator developed by Analog Devices.

About the Author

Ole Dreessen is a field applications staff engineer at Analog Devices. Prior to joining ADI in 2014, he held positions at Avnet Memec and Macnica, supporting communication technologies and high performance microprocessors. Ole has broad expertise in microcontroller and security topics and is an experienced presenter at conferences and distribution events. In his free time, he is an active member of the Chaos Computer Club, where he has worked on concepts such as reverse engineering and embedded security.