# 1 Welcome to computer vision

***This chapter covers***

- Components of the vision system
- Applications of computer vision
- Understanding the computer vision pipeline
- Preprocessing images and extracting features
- Using classifier learning algorithms

Hello! I'm very excited that you are here. You are making a great decision—to grasp deep learning (DL) and computer vision (CV). The timing couldn't be more perfect. CV is an area that's been advancing rapidly, thanks to the huge AI and DL advances of recent years. Neural networks are now allowing self-driving cars to figure out where other cars and pedestrians are and navigate around them. We are using CV applications in our daily lives more and more with all the smart devices in our homes—from security cameras to door locks. CV is also making face recognition work better than ever: smartphones can recognize faces for unlocking, and smart locks can unlock doors. I wouldn't be surprised if sometime in the near future, your couch or television is able to recognize specific people in your house and react according to their personal preferences. It's not just about recognizing

objects—DL has given computers the power to imagine and create new things like artwork; new objects; and even unique, realistic human faces.

The main reason that I'm excited about deep learning for computer vision, and what drew me to this field, is how rapid advances in AI research are enabling new applications to be built every day and across different industries, something not possible just a few years ago. The unlimited possibilities of CV research is what inspired me to write this book. By learning these tools, perhaps you will be able to invent new products and applications. Even if you end up not working on CV per se, you will find many concepts in this book useful for some of your DL algorithms and architectures. That is because while the main focus is CV applications, this book covers the most important DL architectures, such as artificial neural networks (ANNs), convolutional networks (CNNs), generative adversarial networks (GANs), transfer learning, and many more, which are transferable to other domains like natural language processing (NLP) and voice user interfaces (VUIs).

The high-level layout of this chapter is as follows:

- *Computer vision intuition*—We will start with visual perception intuition and learn the similarities between humans and machine vision systems. We will look at how vision systems have two main components: a sensing device and an interpreting device. Each is tailored to fulfill a specific task.
- *Applications of CV*—Here, we will take a bird's-eye view of the DL algorithms used in different CV applications. We will then discuss vision in general for different creatures.
- *Computer vision pipeline*—Finally, we will zoom in on the second component of vision systems: the interpreting device. We will walk through the sequence of steps taken by vision systems to process and understand image data. These are referred to as a *computer vision pipeline*. The CV pipeline is composed of four main steps: image input, image preprocessing, feature extraction, and an ML model to interpret the image. We will talk about image formation and how computers see images. Then, we will quickly review image-processing techniques and extracting features.

Ready? Let's get started!

## 1.1 Computer vision

The core concept of any AI system is that it can perceive its environment and take actions based on its perceptions. *Computer vision* is concerned with the visual perception part: it is the science of perceiving and understanding the world through images and videos by constructing a physical model of the world so that an AI system can then take appropriate actions. For humans, vision is only one aspect of perception. We perceive the world through our sight, but also through sound, smell, and our other senses. It is similar with AI systems—vision is just one way to understand the world. Depending on the application you are building, you select the sensing device that best captures the world.

### 1.1.1 What is visual perception?

*Visual perception*, at its most basic, is the act of observing patterns and objects through sight or visual input. With an autonomous vehicle, for example, visual perception means understanding the surrounding objects and their specific details—such as pedestrians, or whether there is a particular lane the vehicle needs to be centered in—and detecting traffic signs and understanding what they mean. That's why the word *perception* is part of the definition. We are not just looking to capture the surrounding environment. We are trying to build systems that can actually understand that environment through visual input.

### 1.1.2 Vision systems

In past decades, traditional image-processing techniques were considered CV systems, but that is not totally accurate. A machine processing an image is completely different from that machine understanding what's happening within the image, which is not a trivial task. Image processing is now just a piece of a bigger, more complex system that aims to interpret image content.

#### HUMAN VISION SYSTEMS

At the highest level, vision systems are pretty much the same for humans, animals, insects, and most living organisms. They consist of a sensor or an eye to capture the image and a brain to process and interpret the image. The system then outputs a prediction of the image components based on the data extracted from the image (figure 1.1).

Let's see how the human vision system works. Suppose we want to interpret the image of dogs in figure 1.1. We look at it and directly understand that the image consists of a bunch of dogs (three, to be specific). It comes pretty natural to us to classify

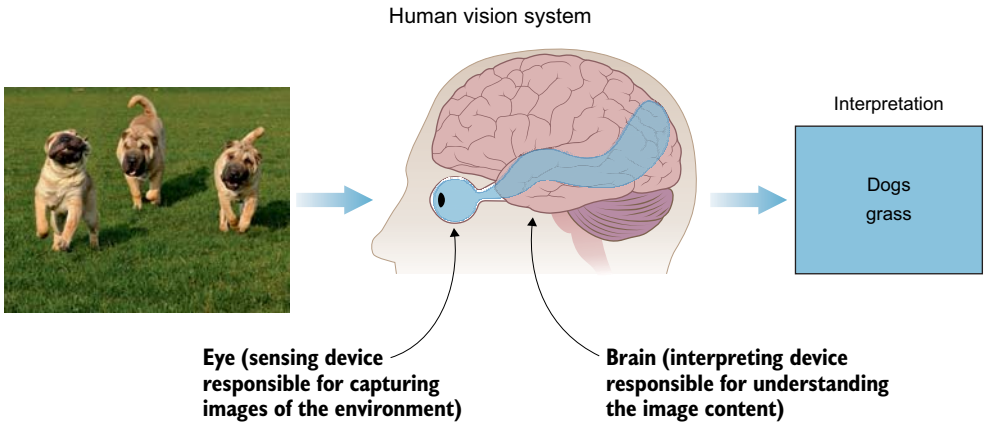



**Figure 1.1 The human vision system uses the eye and brain to sense and interpret an image.**

and detect objects in this image because we have been trained over the years to identify dogs.

Suppose someone shows you a picture of a dog for the first time—you definitely don't know what it is. Then they tell you that this is a dog. After a couple experiments like this, you will have been trained to identify dogs. Now, in a follow-up exercise, they show you a picture of a horse. When you look at the image, your brain starts analyzing the object features: hmmm, it has four legs, long face, long ears. Could it be a dog? "Wrong: this is a horse," you're told. Then your brain adjusts some parameters in its algorithm to learn the differences between dogs and horses. Congratulations! You just trained your brain to classify dogs and horses. Can you add more animals to the equation, like cats, tigers, cheetahs, and so on? Definitely. You can train your brain to identify almost anything. The same is true of computers. You can train machines to learn and identify objects, but humans are much more intuitive than machines. It takes only a few images for you to learn to identify most objects, whereas with machines, it takes thousands or, in more complex cases, millions of image samples to learn to identify objects.

### The ML perspective

Let's look at the previous example from the machine learning perspective:

- You learned to identify dogs by looking at examples of several dog-labeled images. This approach is called *supervised learning.*
- *Labeled data* is data for which you already know the target answer. You were shown a sample image of a dog and told that it was a dog. Your brain learned to associate the features you saw with this label: dog.
- You were then shown a different object, a horse, and asked to identify it. At first, your brain thought it was a dog, because you hadn't seen horses before, and your brain confused horse features with dog features. When you were told that your prediction was wrong, your brain adjusted its parameters to learn horse features. "Yes, both have four legs, but the horse's legs are longer. Longer legs indicate a horse." We can run this experiment many times until the brain makes no mistakes. This is called *training by trial and error*.

#### AI VISION SYSTEMS

Scientists were inspired by the human vision system and in recent years have done an amazing job of copying visual ability with machines. To mimic the human vision system, we need the same two main components: a sensing device to mimic the function of the eye and a powerful algorithm to mimic the brain function in interpreting and classifying image content (figure 1.2).

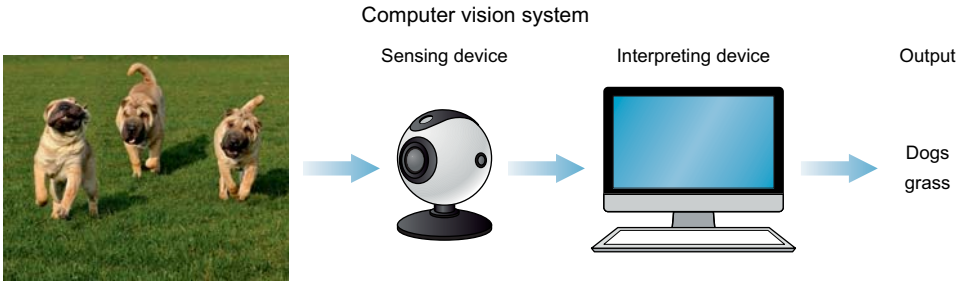



**Figure 1.2 The components of the computer vision system are a sensing device and an interpreting device.**

### 1.1.3 Sensing devices

Vision systems are designed to fulfill a specific task. An important aspect of design is selecting the best sensing device to capture the surroundings of a specific environment, whether that is a camera, radar, X-ray, CT scan, Lidar, or a combination of devices to provide the full scene of an environment to fulfill the task at hand.

Let's look at the autonomous vehicle (AV) example again. The main goal of the AV vision system is to allow the car to understand the environment around it and move from point A to point B safely and in a timely manner. To fulfill this goal, vehicles are equipped with a combination of cameras and sensors that can detect 360 degrees of movement—pedestrians, cyclists, vehicles, roadwork, and other objects—from up to three football fields away.

Here are some of the sensing devices usually used in self-driving cars to perceive the surrounding area:

- Lidar, a radar-like technique, uses invisible pulses of light to create a high-resolution 3D map of the surrounding area.
- Cameras can see street signs and road markings but cannot measure distance.
- Radar can measure distance and velocity but cannot see in fine detail.

Medical diagnosis applications use X-rays or CT scans as sensing devices. Or maybe you need to use some other type of radar to capture the landscape for agricultural vision systems. There are a variety of vision systems, each designed to perform a particular task. The first step in designing vision systems is to identify the task they are built for. This is something to keep in mind when designing end-to-end vision systems.

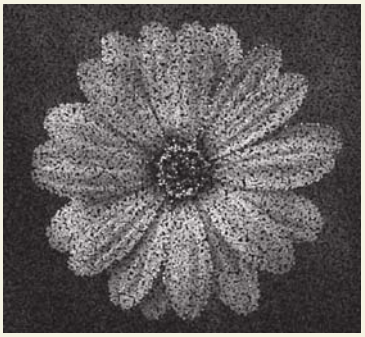
#### Recognizing images

Animals, humans, and insects all have eyes as sensing devices. But not all eyes have the same structure, output image quality, and resolution. They are tailored to the specific needs of the creature. Bees, for instance, and many other insects, have compound

***(continued)***

eyes that consist of multiple lenses (as many as 30,000 lenses in a single compound eye). Compound eyes have low resolution, which makes them not so good at recognizing objects at a far distance. But they are very sensitive to motion, which is essential for survival while flying at high speed. Bees don't need high-resolution pictures. Their vision systems are built to allow them to pick up the smallest movements while flying fast.

Compound eyes



How bees see a flower

**Compound eyes are low resolution but sensitive to motion.**

### 1.1.4 Interpreting devices

Computer vision algorithms are typically employed as interpreting devices. The interpreter is the brain of the vision system. Its role is to take the output image from the sensing device and learn features and patterns to identify objects. So we need to build a brain. Simple! Scientists were inspired by how our brains work and tried to reverse engineer the central nervous system to get some insight on how to build an artificial brain. Thus, *artificial neural networks (ANNs)* were born (figure 1.3).

In figure 1.3, we can see an analogy between biological neurons and artificial systems. Both contain a main processing element, a *neuron*, with input signals ($x_1$, $x_2$, …, $x_n$) and an output.

The learning behavior of biological neurons inspired scientists to create a network of neurons that are connected to each other. Imitating how information is processed in the human brain, each artificial neuron fires a signal to all the neurons that it's connected to when enough of its input signals are activated. Thus, neurons have a very simple mechanism on the individual level (as you will see in the next chapter); but when you have millions of these neurons stacked in layers and connected together, each neuron is connected to thousands of other neurons, yielding a learning behavior. Building a multilayer neural network is called *deep learning* (figure 1.4).

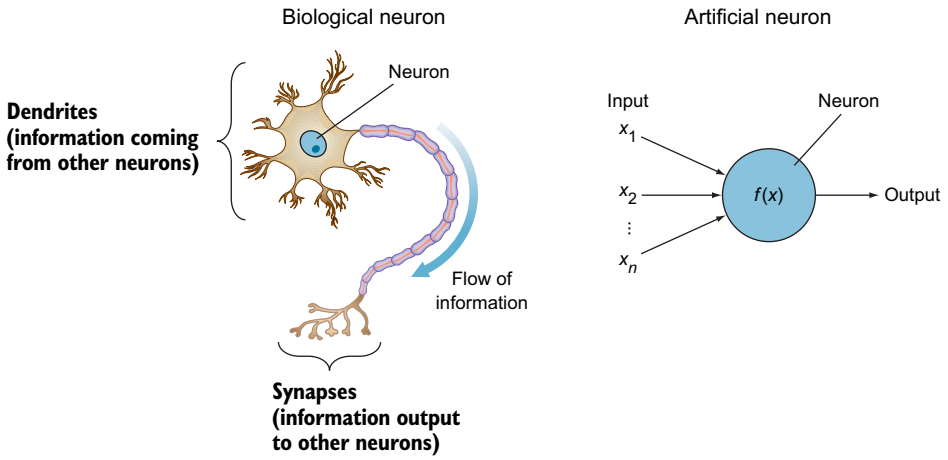



**Figure 1.3 The similarities between biological neurons and artificial systems**

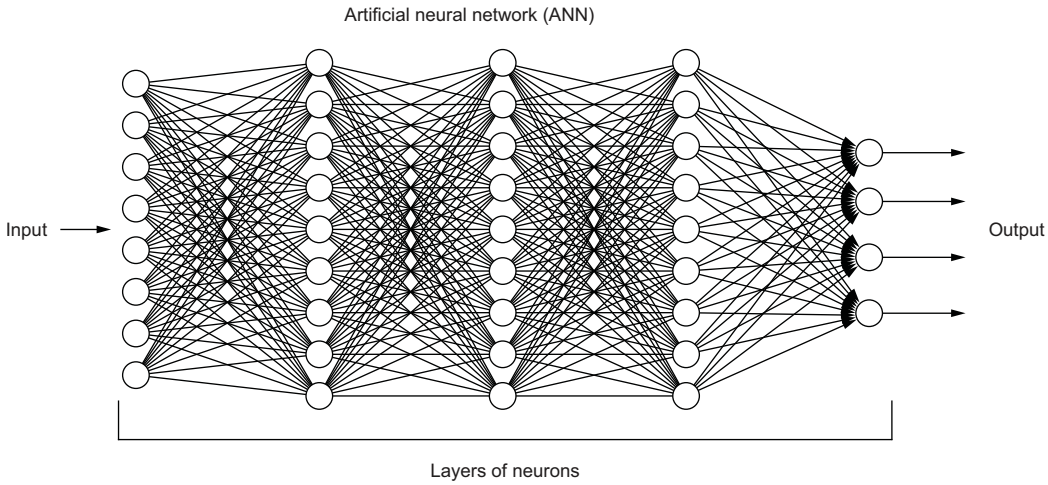



**Figure 1.4 Deep learning involves layers of neurons in a network.**

DL methods learn representations through a sequence of transformations of data through layers of neurons. In this book, we will explore different DL architectures, such as ANNs and convolutional neural networks, and how they are used in CV applications.

#### CAN MACHINE LEARNING ACHIEVE BETTER PERFORMANCE THAN THE HUMAN BRAIN?

Well, if you had asked me this question 10 years ago, I would've probably said no, machines cannot surpass the accuracy of a human. But let's take a look at the following two scenarios:

- Suppose you were given a book of 10,000 dog images, classified by breed, and you were asked to learn the properties of each breed. How long would it take you to study the 130 breeds in 10,000 images? And if you were given a test of 100 dog images and asked to label them based on what you learned, out of the 100, how many would you get right? Well, a neural network that is trained in a couple of hours can achieve more than 95% accuracy.
- On the creation side, a neural network can study the patterns in the strokes, colors, and shading of a particular piece of art. Based on this analysis, it can then transfer the style from the original artwork into a new image and create a new piece of original art within a few seconds.

Recent AI and DL advances have allowed machines to surpass human visual ability in many image classification and object detection applications, and capacity is rapidly expanding to many other applications. But don't take my word for it. In the next section, we'll discuss some of the most popular CV applications using DL technology.

## 1.2 Applications of computer vision

Computers began to be able to recognize human faces in images decades ago, but now AI systems are rivaling the ability of computers to classify objects in photos and videos. Thanks to the dramatic evolution in both computational power and the amount of data available, AI and DL have managed to achieve superhuman performance on many complex visual perception tasks like image search and captioning, image and video classification, and object detection. Moreover, deep neural networks are not restricted to CV tasks: they are also successful at natural language processing and voice user interface tasks. In this book, we'll focus on visual applications that are applied in CV tasks.

DL is used in many computer vision applications to recognize objects and their behavior. In this section, I'm not going to attempt to list all the CV applications that are out there. I would need an entire book for that. Instead, I'll give you a bird's-eye view of some of the most popular DL algorithms and their possible applications across different industries. Among these industries are autonomous cars, drones, robots, in-store cameras, and medical diagnostic scanners that can detect lung cancer in early stages.

### 1.2.1 Image classification

*Image classification* is the task of assigning to an image a label from a predefined set of categories. A *convolutional neural network* is a neural network type that truly shines in processing and classifying images in many different applications:

- *Lung cancer diagnosis*—Lung cancer is a growing problem. The main reason lung cancer is very dangerous is that when it is diagnosed, it is usually in the middle or

late stages. When diagnosing lung cancer, doctors typically use their eyes to examine CT scan images, looking for small nodules in the lungs. In the early stages, the nodules are usually very small and hard to spot. Several CV companies decided to tackle this challenge using DL technology.

Almost every lung cancer starts as a small nodule, and these nodules appear in a variety of shapes that doctors take years to learn to recognize. Doctors are very good at identifying mid- and large-size nodules, such as 6–10 mm. But when nodules are 4 mm or smaller, sometimes doctors have difficulty identifying them. DL networks, specifically CNNs, are now able to learn these features automatically from X-ray and CT scan images and detect small nodules early, before they become deadly (figure 1.5).

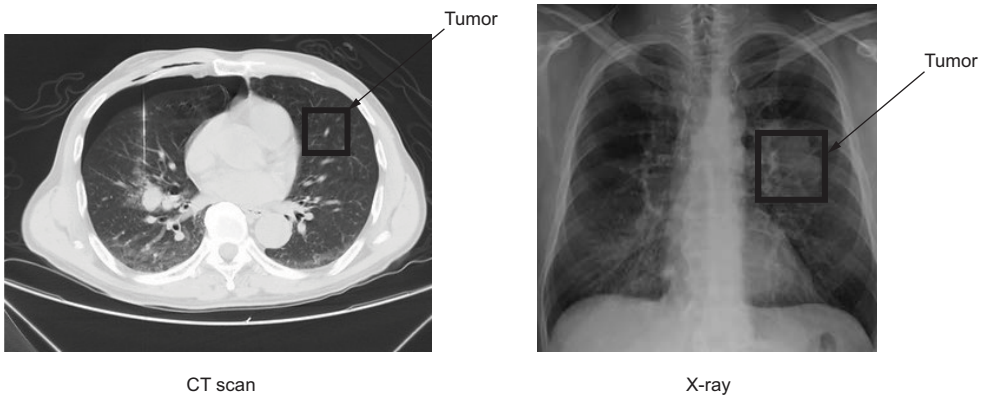



**Figure 1.5 Vision systems are now able to learn patterns in X-ray images to identify tumors in earlier stages of development.**

- *Traffic sign recognition*—Traditionally, standard CV methods were employed to detect and classify traffic signs, but this approach required time-consuming manual work to handcraft important features in images. Instead, by applying DL to this problem, we can create a model that reliably classifies traffic signs, learning to identify the most appropriate features for this problem by itself (figure 1.6).

**NOTE** Increasing numbers of image classification tasks are being solved with convolutional neural networks. Due to their high recognition rate and fast execution, CNNs have enhanced most CV tasks, both pre-existing and new. Just like the cancer diagnosis and traffic sign examples, you can feed tens or hundreds of thousands of images into a CNN to label them into as many classes as you want. Other image classification examples include identifying people and objects, classifying different animals (like cats versus dogs versus horses), different breeds of animals, types of land suitable for agriculture, and so on. In short, if you have a set of *labeled* images, convolutional networks can classify them into a set of predefined classes.


**Figure 1.6 Vision systems can detect traffic signs with very high performance.**

### 1.2.2 Object detection and localization

Image classification problems are the most basic applications for CNNs. In these problems, each image contains only one object, and our task is to identify it. But if we aim to reach human levels of understanding, we have to add complexity to these networks so they can recognize multiple objects and their locations in an image. To do that, we can build object detection systems like YOLO (you only look once), SSD (single-shot detector), and Faster R-CNN, which not only classify images but also can locate and detect each object in images that contain multiple objects. These DL systems can look at an image, break it up into smaller regions, and label each region with a class so that a variable number of objects in a given image can be localized and labeled (figure 1.7). You can imagine that such a task is a basic prerequisite for applications like autonomous systems.

### 1.2.3 Generating art (style transfer)

*Neural style transfer*, one of the most interesting CV applications, is used to transfer the style from one image to another. The basic idea of style transfer is this: you take one image—say, of a city—and then apply a style of art to that image—say, *The Starry Night* (by Vincent Van Gogh)—and output the same city from the original image, but looking as though it was painted by Van Gogh (figure 1.8).

This is actually a neat application. The astonishing thing, if you know any painters, is that it can take days or even weeks to finish a painting, and yet here is an application that can paint a new image inspired by an existing style in a matter of seconds.

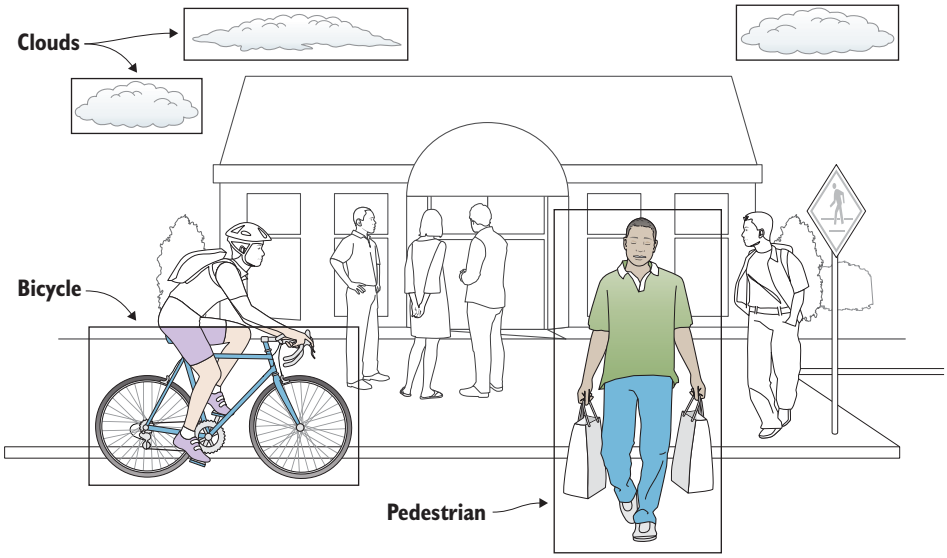



Figure 1.7 Deep learning systems can segment objects in an image.


Figure 1.8 Style transfer from Van Gogh's *The Starry Night* onto the original image, producing a piece of art that feels as though it was created by the original artist

### 1.2.4 Creating images

Although the earlier examples are truly impressive CV applications of AI, this is where I see the real magic happening: the magic of creation. In 2014, Ian Goodfellow invented a new DL model that can imagine new things called generative adversarial networks (GANs). The name makes them sound a little intimidating, but I promise you that they are not. A GAN is an evolved CNN architecture that is

considered a major advancement in DL. So when you understand CNNs, GANs will make a lot more sense to you.

GANs are sophisticated DL models that generate stunningly accurate synthesized images of objects, people, and places, among other things. If you give them a set of images, they can make entirely new, realistic-looking images. For example, StackGAN is one of the GAN architecture variations that can use a textual description of an object to generate a high-resolution image of the object matching that description. This is not just running an image search on a database. These "photos" have never been seen before and are totally imaginary (figure 1.9).


**Figure 1.9 Generative adversarial networks (GANS) can create new, "made-up" images from a set of existing images.**

The GAN is one of the most promising advancements in machine learning in recent years. Research into GANs is new, and the results are overwhelmingly promising. Most of the applications of GANs so have far have been for images. But it makes you wonder: if machines are given the power of imagination to create pictures, what else can they create? In the future, will your favorite movies, music, and maybe even books be created by computers? The ability to synthesize one data type (text) to another (image) will eventually allow us to create all sorts of entertainment using only detailed text descriptions.

### GANs create artwork

In October 2018, an AI-created painting called *The Portrait of Edmond Belamy* sold for $432,500. The artwork features a fictional person named Edmond de Belamy, possibly French and—to judge by his dark frock coat and plain white collar—a man of the church.

**AI-generated artwork featuring a fictional person named Edmond de Belamy sold for $432,500.**

The artwork was created by a team of three 25-year-old French students using GANs. The network was trained on a dataset of 15,000 portraits painted between the fourteenth and twentieth centuries, and then it created one of its own. The team printed the image, framed it, and signed it with part of a GAN algorithm.

### 1.2.5 Face recognition

Face recognition (FR) allows us to exactly identify or tag an image of a person. Day-to-day applications include searching for celebrities on the web and auto-tagging friends and family in images. Face recognition is a form of fine-grained classification.

The famous *Handbook of Face Recognition* (Li et al., Springer, 2011) categorizes two modes of an FR system:

- *Face identification*—Face identification involves one-to-many matches that compare a query face image against all the template images in the database to determine the identity of the query face. Another face recognition scenario involves a watchlist check by city authorities, where a query face is matched to a list of suspects (one-to-few matches).
- *Face verification*—Face verification involves a one-to-one match that compares a query face image against a template face image whose identity is being claimed (figure 1.10).

### 1.2.6 Image recommendation system

In this task, a user seeks to find similar images with respect to a given query image. Shopping websites provide product suggestions (via images) based on the selection of a particular product, for example, showing a variety of shoes similar to those the user selected. An example of an apparel search is shown in figure 1.11.

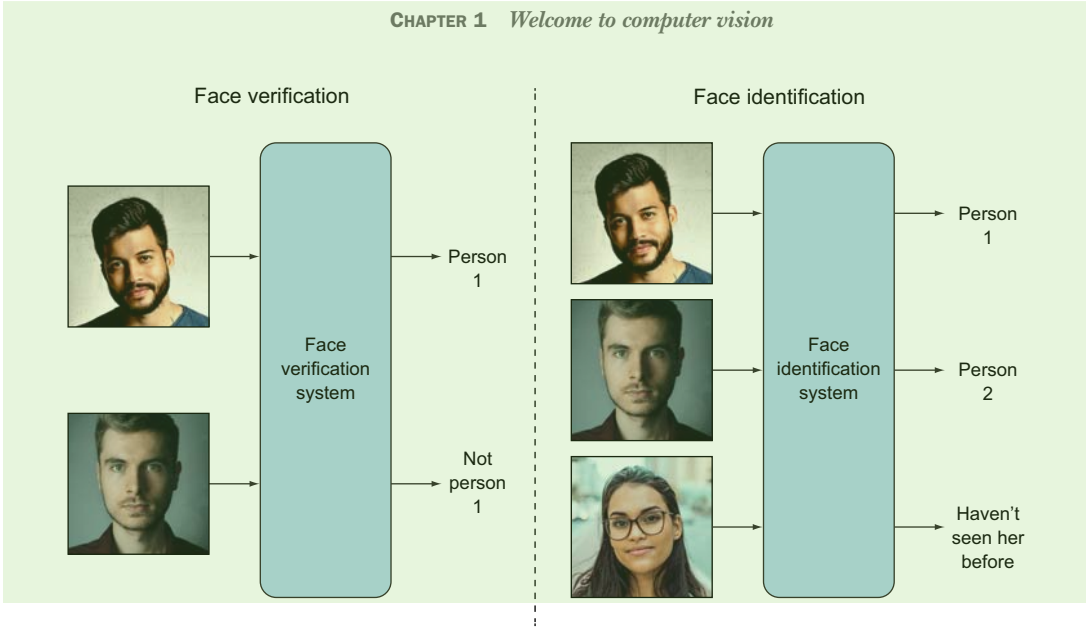



**Figure 1.10 Example of face verification (left) and face recognition (right)**

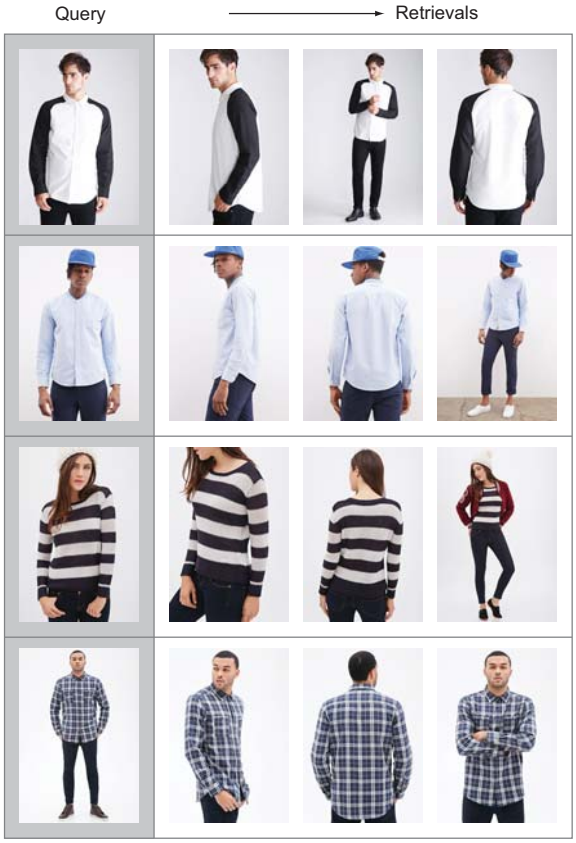



**Figure 1.11 Apparel search. The leftmost image in each row is the query/clicked image, and the subsequent columns show similar apparel. (*Source*: Liu et al., 2016.)**

## 1.3 Computer vision pipeline: The big picture

Okay, now that I have your attention, let's dig one level deeper into CV systems. Remember that earlier in this chapter, we discussed how vision systems are composed of two main components: sensing devices and interpreting devices (figure 1.12 offers a reminder). In this section, we will take a look at the pipeline the interpreting device component uses to process and understand images.

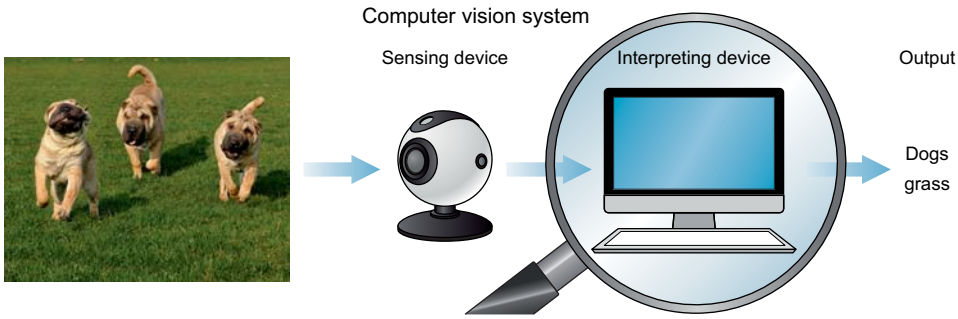



**Figure 1.12 Focusing on the interpreting device in computer vision systems**

Applications of CV vary, but a typical vision system uses a sequence of distinct steps to process and analyze image data. These steps are referred to as a *computer vision pipeline*. Many vision applications follow the flow of acquiring images and data, processing that data, performing some analysis and recognition steps, and then finally making a prediction based on the extracted information (figure 1.13).

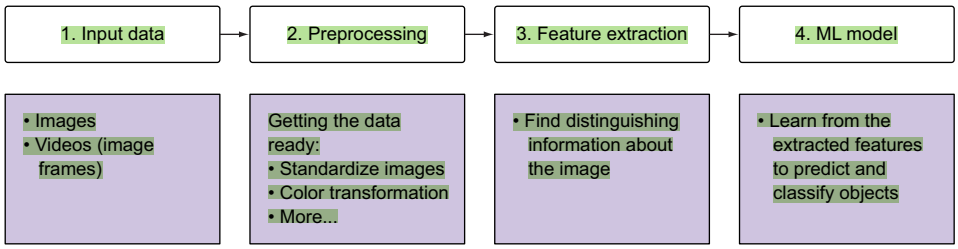



**Figure 1.13 The computer vision pipeline, which takes input data, processes it, extracts information, and then sends it to the machine learning model to learn**

Let's apply the pipeline in figure 1.13 to an image classifier example. Suppose we have an image of a motorcycle, and we want the model to predict the probability of the object from the following classes: motorcycle, car, and dog (see figure 1.14).

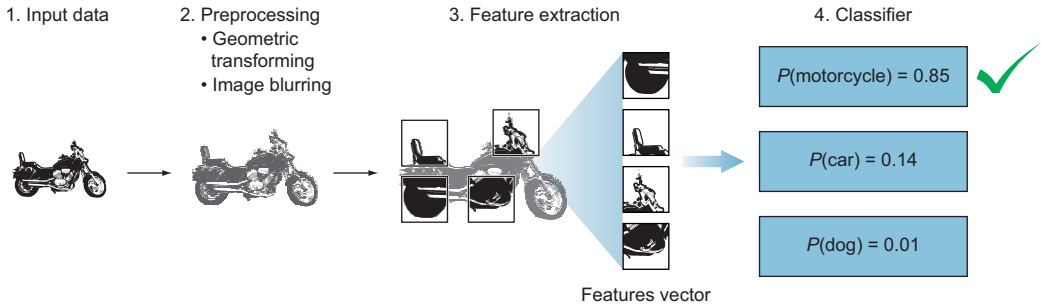



**Figure 1.14 Using the machine learning model to predict the probability of the motorcycle object from the motorcycle, car, and dog classes**

**DEFINITIONS** An *image classifier* is an algorithm that takes in an image as input and outputs a label or "class" that identifies that image. A *class* (also called a *category*) in machine learning is the output category of your data.

Here is how the image flows through the classification pipeline:

1. A computer receives visual input from an imaging device like a camera. This input is typically captured as an image or a sequence of images forming a video.
2. Each image is then sent through some preprocessing steps whose purpose is to standardize the images. Common preprocessing steps include resizing an image, blurring, rotating, changing its shape, or transforming the image from one color to another, such as from color to grayscale. Only by standardizing the images—for example, making them the same size—can you then compare them and further analyze them.
3. We extract features. *Features* are what help us define objects, and they are usually information about object shape or color. For example, some features that distinguish a motorcycle are the shape of the wheels, headlights, mudguards, and so on. The output of this process is a *feature vector* that is a list of unique shapes that identify the object.
4. The features are fed into a *classification model*. This step looks at the feature vector from the previous step and predicts the class of the image. Pretend that you are the classifier model for a few minutes, and let's go through the classification process. You look at the list of features in the feature vector one by one and try to determine what's in the image:
   a. First you see a *wheel* feature; could this be a car, a motorcycle, or a dog? Clearly it is not a dog, because dogs don't have wheels (at least, normal dogs, not robots). Then this could be an image of a car or a motorcycle.
   b. You move on to the next feature, the *headlights*. There is a higher probability that this is a motorcycle than a car.
   c. The next feature is *rear mudguards*—again, there is a higher probability that it is a motorcycle.

d The object has only two wheels; this is closer to a motorcycle.
e And you keep going through all the features like the body shape, pedal, and so on, until you arrive at a best guess of the object in the image.

The output of this process is the probability of each class. As you can see in our example, the dog has the lowest probability, 1%, whereas there is an 85% probability that this is a motorcycle. You can see that, although the model was able to predict the right class with the highest probability, it is still a little confused about distinguishing between cars and motorcycles—it predicted that there is a 14% chance this is an image of a car. Since we know that it is a motorcycle, we can say that our ML classification algorithm is 85% accurate. Not bad! To improve this accuracy, we may need to do more of step 1 (acquire more training images), or step 2 (more processing to remove noise), or step 3 (extract better features), or step 4 (change the classifier algorithm and tune some hyperparameters), or even allow more training time. The many different approaches we can take to improve the performance of our model all lie in one or more of the pipeline steps.

That was the big picture of how images flow through the CV pipeline. Next, we'll zoom in one level deeper on each of the pipeline steps.

## 1.4 Image input

In CV applications, we deal with images or video data. Let's talk about grayscale and color images for now, and in later chapters, we will talk about videos, since videos are just stacked sequential frames of images.

### 1.4.1 Image as functions

An image can be represented as a function of two variables $x$ and $y$, which define a two-dimensional area. A digital image is made of a grid of pixels. The *pixel* is the raw building block of an image. Every image consists of a set of pixels in which their values represent the *intensity* of light that appears in a given place in the image. Let's take a look at the motorcycle example again after applying the pixel grid to it (figure 1.15).

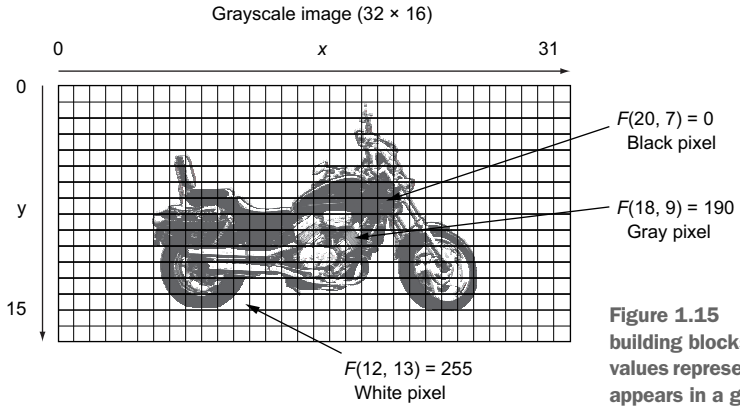



**Figure 1.15 Images consists of raw building blocks called *pixels*. The pixel values represent the *intensity* of light that appears in a given place in the image.**

The image in figure 1.14 has a size of 32 × 16. This means the dimensions of the image are 32 pixels wide and 16 pixels tall. The x-axis goes from 0 to 31, and the y-axis from 0 to 16. Overall, the image has 512 (32 × 16) pixels. In this grayscale image, each pixel contains a value that represents the *intensity of light* on that specific pixel. The pixel values range from 0 to 255. Since the pixel value represents the intensity of light, the value 0 represents very dark pixels (black), 255 is very bright (white), and the values in between represent the intensity on the grayscale.

You can see that the image coordinate system is similar to the Cartesian coordinate system: images are two-dimensional and lie on the x-y plane. The origin (0, 0) is at the top left of the image. To represent a specific pixel, we use the following notations: $F$ as a function, and $x$, $y$ as the location of the pixel in x- and y-coordinates. For example, the pixel located at $x = 12$ and $y = 13$ is white; this is represented by the following function: $F(12, 13) = 255$. Similarly, the pixel (20, 7) that lies on the front of the motorcycle is black, represented as $F(20, 7) = 0$.

```
Grayscale => F(x, y) gives the intensity at position (x, y)
```

That was for grayscale images. How about color images?

In color images, instead of representing the value of the pixel by just one number, the value is represented by three numbers representing the intensity of each color in the pixel. In an RGB system, for example, the value of the pixel is represented by three numbers: the intensity of red, intensity of green, and intensity of blue. There are other color systems for images like HSV and Lab. All follow the same concept when representing the pixel value (more on color images soon). Here is the function representing color images in the RGB system:

```
Color image in RGB => F(x, y) = [ red (x, y), green (x, y), blue (x, y) ]
```

Thinking of an image as a function is very useful in image processing. We can think of an image as a function of $F(x, y)$ and operate on it mathematically to transform it to a new image function $G(x, y)$. Let's take a look at the image transformation examples in table 1.1.

**Table 1.1 Image transformation example functions**

| Application | Transformation |
|---|---|
| Darken the image. | `G(x, y) = 0.5 * F(x, y)` |
| Brighten the image. | `G(x, y) = 2 * F(x, y)` |
| Move an object down 150 pixels. | `G(x, y) = F(x, y + 150)` |
| Remove the gray in an image to transform the image into black and white. | `G(x, y) = { 0 if F(x, y) < 130, 255 otherwise }` |

### 1.4.2 How computers see images

When we look at an image, we see objects, landscape, colors, and so on. But that's not the case with computers. Consider figure 1.16. Your human brain can process it and immediately know that it is a picture of a motorcycle. To a computer, the image looks like a 2D matrix of the pixels' values, which represent intensities across the color spectrum. There is no context here, just a massive pile of data.

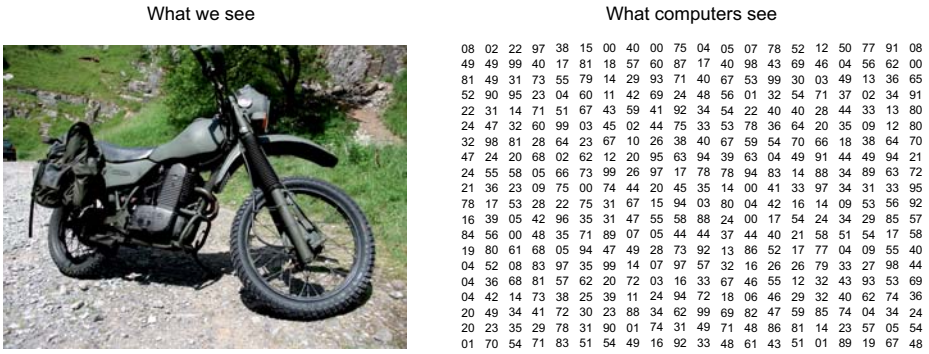



**Figure 1.16 A computer sees images as matrices of values. The values represent the intensity of the pixels across the color spectrum. For example, grayscale images range between pixel values of 0 for black and 255 for white.**

The image in figure 1.16 is of size 24 × 24. This size indicates the width and height of the image: there are 24 pixels horizontally and 24 vertically. That means there is a total of 576 (24 × 24) pixels. If the image is 700 × 500, then the dimensionality of the matrix will be (700, 500), where each pixel in the matrix represents the intensity of brightness in that pixel. Zero represents black, and 255 represents white.

### 1.4.3 Color images

In grayscale images, each pixel represents the intensity of only one color, whereas in the standard RGB system, color images have three channels (red, green, and blue). In other words, color images are represented by three matrices: one represents the intensity of red in the pixel, one represents green, and one represents blue (figure 1.17).

As you can see in figure 1.17, the color image is composed of three channels: red, green, and blue. Now the question is, how do computers see this image? Again, they see the matrix, unlike grayscale images, where we had only one channel. In this case, we will have three matrices stacked on top of each other; that's why it's a 3D matrix. The dimensionality of 700 × 700 color images is (700, 700, 3). Let's say the first matrix represents the red channel; then each element of that matrix represents an intensity of red color in that pixel, and likewise with green and blue. Each pixel in a color

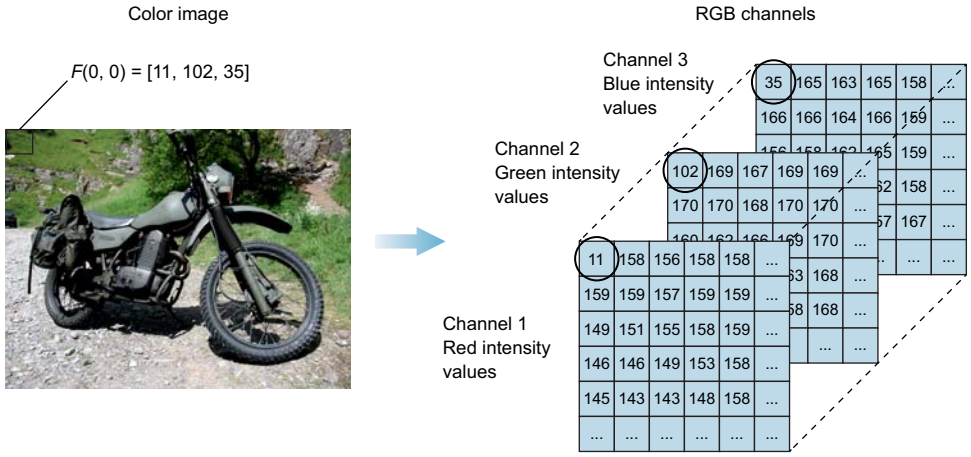



**Figure 1.17 Color images are represented by red, green, and blue channels, and matrices can be used to indicate those colors' intensity.**

image has three numbers (0 to 255) associated with it. These numbers represent intensity of red, green, and blue color in that particular pixel.

If we take the pixel (0,0) as an example, we will see that it represents the top-left pixel of the image of green grass. When we view this pixel in the color images, it looks like figure 1.18. The example in figure 1.19 shows some shades of the color green and their RGB values.

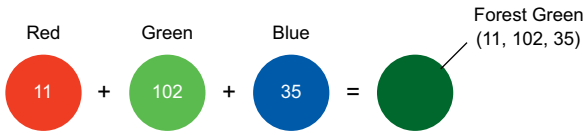



**Figure 1.18 An image of green grass is actually made of three colors of varying intensity.**


**Figure 1.19 Different shades of green mean different intensities of the three image colors (red, green, blue).**

**How do computers see color?**

Computers see an image as matrices. Grayscale images have one channel (gray); thus, we can represent grayscale images in a 2D matrix, where each element represents the intensity of brightness in that particular pixel. Remember, 0 means black and 255 means white. Grayscale images have one channel, whereas color images have three channels: red, green, and blue. We can represent color images in a 3D matrix where the depth is three.

We've also seen how images can be treated as functions of space. This concept allows us to operate on images mathematically and change or extract information from them. Treating images as functions is the basis of many image-processing techniques, such as converting color to grayscale or scaling an image. Each of these steps is just operating mathematical equations to transform an image pixel by pixel.

- Grayscale: $f(x, y)$ gives the intensity at position $(x, y)$
- Color image: $f(x, y)$ = [ red $(x, y)$, green $(x, y)$, blue $(x, y)$ ]

## 1.5 Image preprocessing

In machine learning (ML) projects, you usually go through a data preprocessing or cleaning step. As an ML engineer, you will spend a good amount of time cleaning up and preparing the data before you build your learning model. The goal of this step is to make your data ready for the ML model to make it easier to analyze and process computationally. The same thing is true with images. Based on the problem you are solving and the dataset in hand, some data massaging is required before you feed your images to the ML model.

Image processing could involve simple tasks like image resizing. Later, you will learn that in order to feed a dataset of images to a convolutional network, the images all have to be the same size. Other processing tasks can take place, like geometric and color transformation, converting color to grayscale, and many more. We will cover various image-processing techniques throughout the chapters of this book and in the projects.

The acquired data is usually messy and comes from different sources. To feed it to the ML model (or neural network), it needs to be standardized and cleaned up. Preprocessing is used to conduct steps that will reduce the complexity and increase the accuracy of the applied algorithm. We can't write a unique algorithm for each of the conditions in which an image is taken; thus, when we acquire an image, we convert it into a form that would allow a general algorithm to solve it. The following subsections describe some data-preprocessing techniques.

### 1.5.1 Converting color images to grayscale to reduce computation complexity

Sometimes you will find it useful to remove unnecessary information from your images to reduce space or computational complexity. For example, suppose you want to convert your colored images to grayscale, because for many objects, color is not

necessary to recognize and interpret an image. Grayscale can be good enough for recognizing certain objects. Since color images contain more information than black-and-white images, they can add unnecessary complexity and take up more space in memory. Remember that color images are represented in three channels, which means that converting them to grayscale will reduce the number of pixels that need to be processed (figure 1.20).

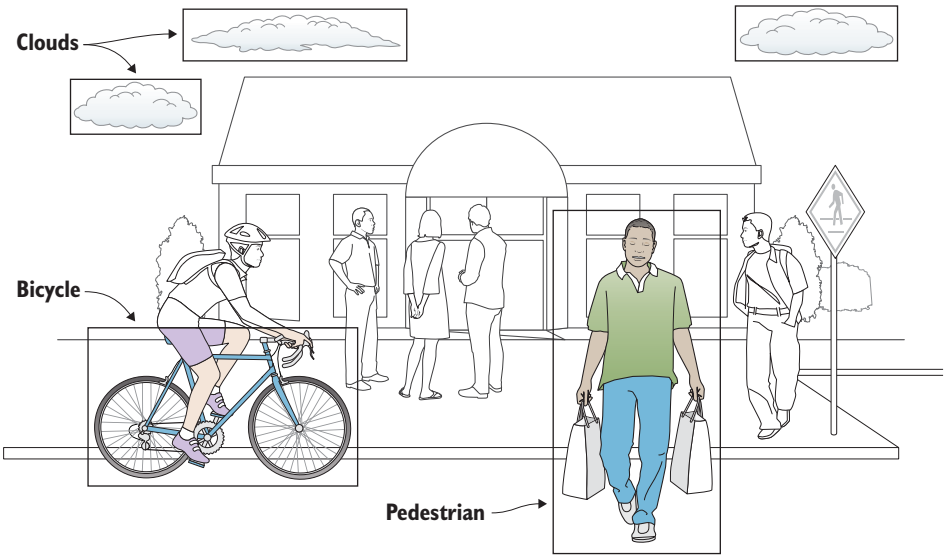



**Figure 1.20 Converting color images to grayscale results in a reduced number of pixels that need to be processed. This could be a good approach for applications that do not rely a lot on the color information loss due to the conversion.**

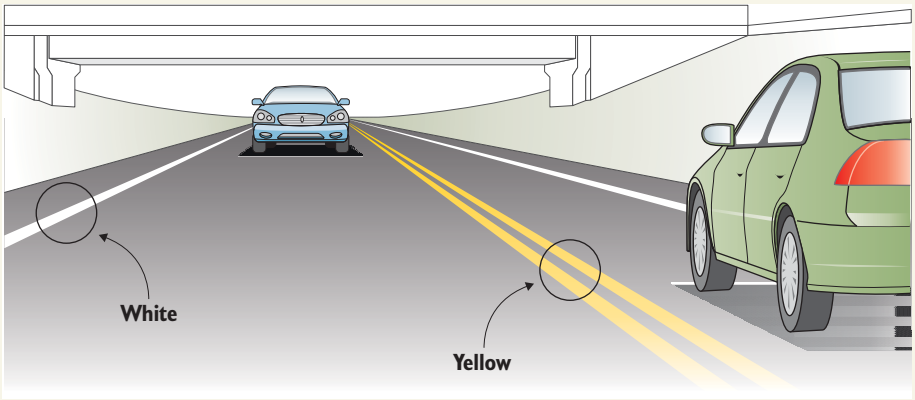
In this example, you can see how patterns of brightness and darkness (intensity) can be used to define the shape and characteristics of many objects. However, in other applications, color is important to define certain objects, like skin cancer detection, which relies heavily on skin color (red rashes).

- *Standardizing images*—As you will see in chapter 3, one important constraint that exists in some ML algorithms, such as CNNs, is the need to resize the images in your dataset to unified dimensions. This implies that your images must be preprocessed and scaled to have identical widths and heights before being fed to the learning algorithm.
- *Data augmentation*—Another common preprocessing technique involves augmenting the existing dataset with modified versions of the existing images. Scaling, rotations, and other affine transformations are typically used to enlarge your dataset and expose the neural network to a wide variety of variations of

### When is color important?

Converting an image to grayscale might not be a good decision for some problems. There are a number of applications for which color is very important: for example, building a diagnostic system to identify red skin rashes in medical images. This application relies heavily on the intensity of the red color in the skin. Removing colors from the image will make it harder to solve this problem. In general, color images provide very helpful information in many medical applications.

Another example of the importance of color in images is lane-detection applications in a self-driving car, where the car has to identify the difference between yellow and white lines, because they are treated differently. Grayscale images do not provide enough information to distinguish between the yellow and white lines.




**Grayscale-based image processors cannot differentiate between color images.**

The rule of thumb to identify the importance of colors in your problem is to look at the image with the human eye. If you are able to identify the object you are looking for in a gray image, then you probably have enough information to feed to your model. If not, then you definitely need more information (colors) for your model. The same rule can be applied for most other preprocessing techniques that we will discuss.

your images. This makes it more likely that your model will recognize objects when they appear in any form and shape. Figure 1.21 shows an example of image augmentation applied to a butterfly image.

- *Other techniques*—Many more preprocessing techniques are available to get your images ready for training an ML model. In some projects, you might need to remove the background color from your images to reduce noise. Other projects might require that you brighten or darken your images. In short, any adjustments that you need to apply to your dataset are part of preprocessing. You will select

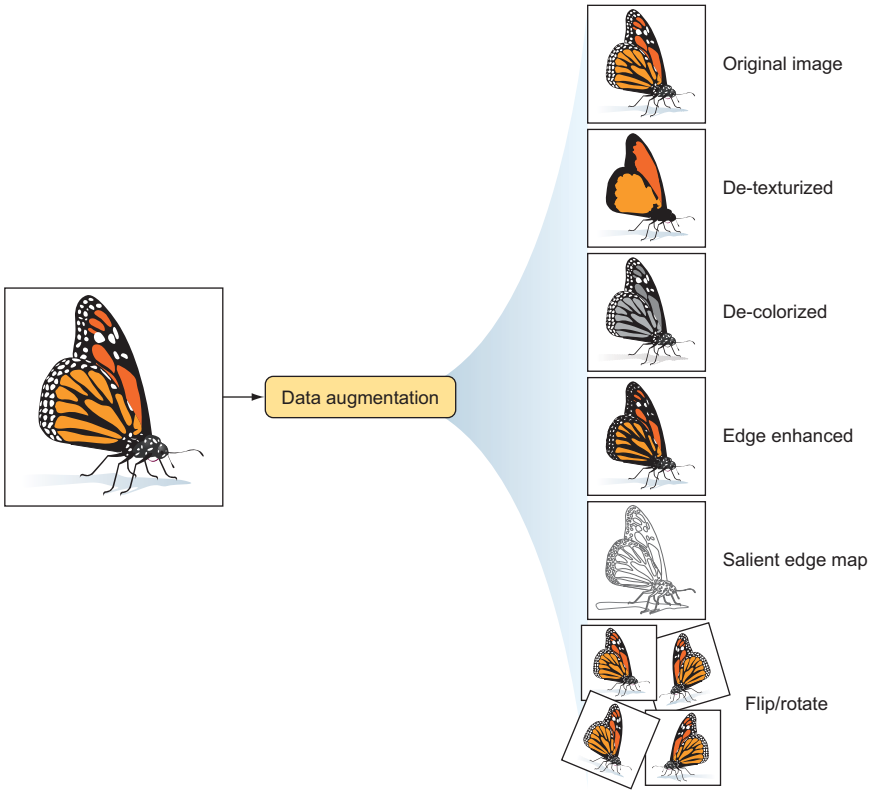



**Figure 1.21 Image-augmentation techniques create modified versions of the input image to provide more examples for the ML model to learn from.**

the appropriate processing techniques based on the dataset at hand and the problem you are solving. You will see many image-processing techniques throughout this book, helping you build your intuition of which ones you need when working on your own projects.

### No free lunch theorem

This is a phrase that was introduced by David Wolpert and William Macready in "No Free Lunch Theorems for Optimizations" (*IEEE Transactions on Evolutionary Computation* 1, 67). You will often hear this said when a team is working on an ML project. It means that no one prescribed recipe fits all models. When working on ML projects, you will need to make many choices like building your neural network architecture, tuning hyperparameters, and applying the appropriate data preprocessing techniques. While there are some rule-of-thumb approaches to tackle certain problems, there is really no single recipe that is guaranteed to work well in all situations.

You must make certain assumptions about the dataset and the problem you are trying to solve. For some datasets, it is best to convert the colored images to grayscale, while for other datasets, you might need to keep or adjust the color images.

The good news is that, unlike traditional machine learning, DL algorithms require minimum data preprocessing because, as you will see soon, neural networks do most of the heavy lifting in processing an image and extracting features.

## 1.6 Feature extraction

*Feature extraction* is a core component of the CV pipeline. In fact, the entire DL model works around the idea of extracting useful features that clearly define the objects in the image. So we'll spend a little more time here, because it is important that you understand what a feature is, what a vector of features is, and why we extract features.

**DEFINITION** A *feature* in machine learning is an individual measurable property or characteristic of an observed phenomenon. Features are the input that you feed to your ML model to output a prediction or classification. Suppose you want to predict the price of a house: your input features (properties) might include `square_foot`, `number_of_rooms`, `bathrooms`, and so on, and the model will output the predicted price based on the values of your features. Selecting good features that clearly distinguish your objects increases the predictive power of ML algorithms.

### 1.6.1 What is a feature in computer vision?

In CV, a *feature* is a measurable piece of data in your image that is unique to that specific object. It may be a distinct color or a specific shape such as a line, edge, or image segment. A good feature is used to distinguish objects from one another. For example, if I give you a feature like a wheel and ask you to guess whether an object is a motorcycle or a dog, what would your guess be? A motorcycle. Correct! In this case, the wheel is a strong feature that clearly distinguishes between motorcycles and dogs. However, if I give you the same feature (a wheel) and ask you to guess whether an object is a bicycle or a motorcycle, this feature is not strong enough to distinguish between those objects. You need to look for more features like a mirror, license plate, or maybe a pedal, that collectively describe an object. In ML projects, we want to transform the raw data (image) into a feature vector to show to our learning algorithm, which can learn the characteristics of the object (figure 1.22).

In the figure, we feed the raw input image of a motorcycle into a feature extraction algorithm. Let's treat the feature extraction algorithm as a black box for now, and we will come back to it. For now, we need to know that the extraction algorithm produces a vector that contains a list of features. This feature vector is a 1D array that makes a robust representation of the object.

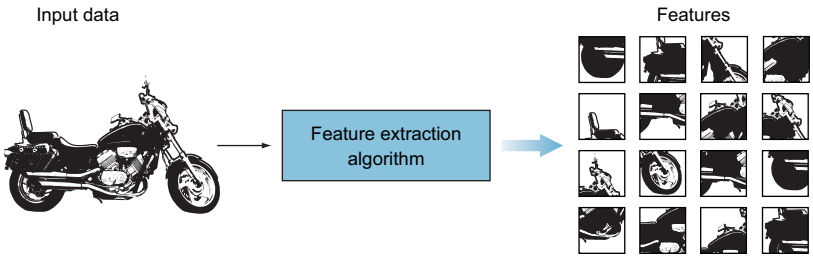
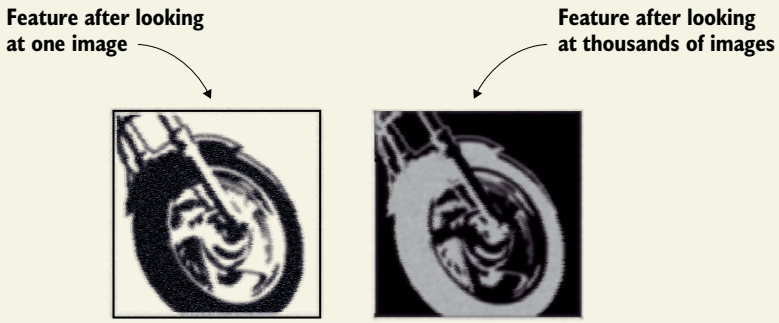



**Figure 1.22 Example input image fed to a feature-extraction algorithm to find patterns within the image and create the feature vector**

### Feature generalizability

It is important to point out that figure 1.22 reflects features extracted from just one motorcycle. A very important characteristic of a feature is *repeatability*. The feature should be able to detect motorcycles in general, not just this specific one. So in real-world problems, a feature is not an exact copy of a piece of the input image.




**Features need to detect general patterns.**

If we take the *wheel* feature, for example, the feature doesn't look exactly like the wheel of one particular motorcycle. Instead, it looks like a circular shape with some patterns that identify wheels in all images in the training dataset. When the feature extractor sees thousands of images of motorcycles, it recognizes patterns that define wheels in general, regardless of where they appear in the image and what type of motorcycle they are part of.

## 1.6.2 What makes a good (useful) feature?

Machine learning models are only as good as the features you provide. That means coming up with good features is an important job in building ML models. But what makes a good feature? And how can you tell?

Let's discuss this with an example. Suppose we want to build a classifier to tell the difference between two types of dogs: Greyhound and Labrador. Let's take two features—the dogs' height and their eye color—and evaluate them (figure 1.23).


**Figure 1.23 Example of Greyhound and Labrador dogs**

Let's begin with height. How useful do you think this feature is? Well, on average, Greyhounds tend to be a couple of inches taller than Labradors, but not always. There is a lot of variation in the dog world. So let's evaluate this feature across different values in both breeds' populations. Let's visualize the height distribution on a toy example in the histogram in figure 1.24.

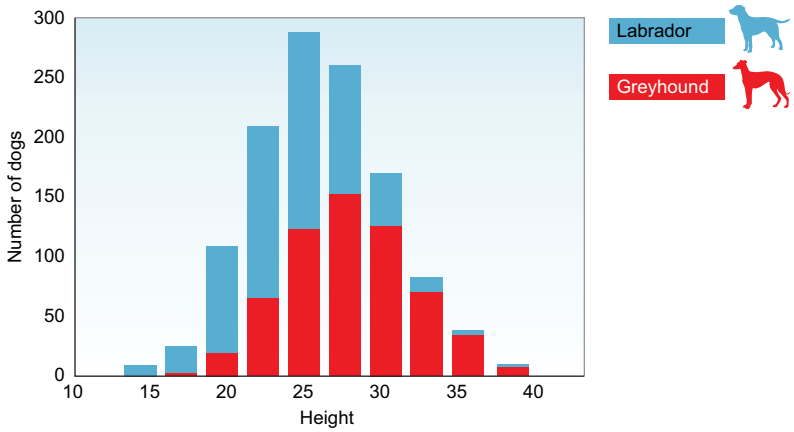



**Figure 1.24 A visualization of the height distribution on a toy dogs dataset**

From the histogram, we can see that if the dog's height is 20 inches or less, there is more than an 80% probability that the dog is a Labrador. On the other side of the histogram, if we look at dogs that are taller than 30 inches, we can be pretty confident

the dog is a Greyhound. Now, what about the data in the middle of the histogram (heights from 20 to 30 inches)? We can see that the probability of each type of dog is pretty close. The thought process in this case is as follows:

if height ≤ 20:

    return higher probability to Labrador

if height ≥ 30:

    return higher probability to Greyhound

if 20 < height < 30:

    look for other features to classify the object

So the height of the dog in this case is a useful feature because it helps (adds information) in distinguishing between both dog types. We can keep it. But it doesn't distinguish between Greyhounds and Labradors in all cases, which is fine. In ML projects, there is usually no one feature that can classify all objects on its own. That's why, in machine learning, we almost always need multiple features, where each feature captures a different type of information. If only one feature would do the job, we could just write `if-else` statements instead of bothering with training a classifier.

> **TIP** Similar to what we did earlier with color conversion (color versus grayscale), to figure out which features you should use for a specific problem, do a thought experiment. Pretend you are the classifier. If you want to differentiate between Greyhounds and Labradors, what information do you need to know? You might ask about the hair length, the body size, the color, and so on.

For another quick example of a non-useful feature to drive this idea home, let's look at dog eye color. For this toy example, imagine that we have only two eye colors, blue and brown. Figure 1.25 shows what a histogram might look like for this example.

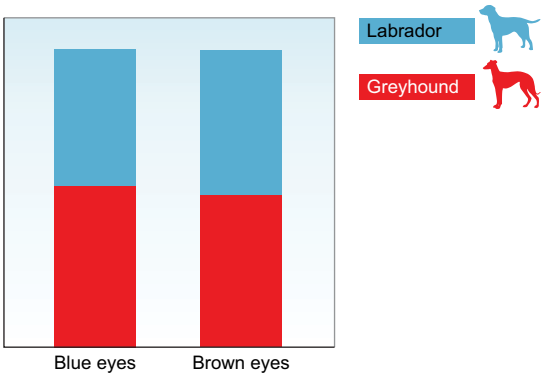



**Figure 1.25 A visualization of the eye color distribution in a toy dogs dataset**

It is clear that for most values, the distribution is about 50/50 for both types. So practically, this feature tells us nothing, because it doesn't correlate with the type of dog. Hence, it doesn't distinguish between Greyhounds and Labradors.

**What makes a good feature for object recognition?**

A good feature will help us recognize an object in all the ways it may appear. Characteristics of a good feature follow:

- Identifiable
- Easily tracked and compared
- Consistent across different scales, lighting conditions, and viewing angles
- Still visible in noisy images or when only part of an object is visible

### 1.6.3 Extracting features (handcrafted vs. automatic extracting)

This is a large topic in machine learning that could take up an entire book. It's typically described in the context of a topic called *feature engineering*. In this book, we are only concerned with extracting features in images. So I'll touch on the idea very quickly in this chapter and build on it in later chapters.

#### TRADITIONAL MACHINE LEARNING USING HANDCRAFTED FEATURES

In traditional ML problems, we spend a good amount of time in manual feature selection and engineering. In this process, we rely on our domain knowledge (or partner with domain experts) to create features that make ML algorithms work better. We then feed the produced features to a classifier like a support vector machine (SVM) or AdaBoost to predict the output (figure 1.26). Some of the handcrafted feature sets are these:

- Histogram of oriented gradients (HOG)
- Haar Cascades
- Scale-invariant feature transform (SIFT)
- Speeded-Up Robust Feature (SURF)

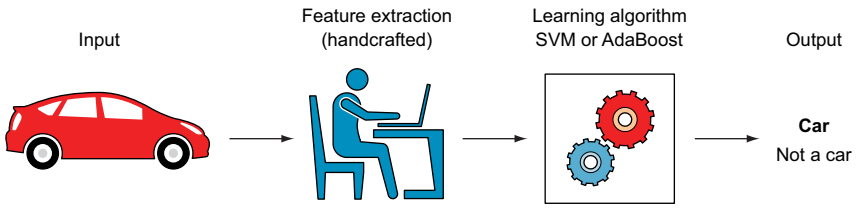



**Figure 1.26 Traditional machine learning algorithms require handcrafted feature extraction.**

### DEEP LEARNING USING AUTOMATICALLY EXTRACTED FEATURES

In DL, however, we do not need to manually extract features from the image. The network extracts features automatically and learns their importance on the output by applying weights to its connections. You just feed the raw image to the network, and while it passes through the network layers, the network identifies patterns within the image with which to create features (figure 1.27). Neural networks can be thought of as feature extractors plus classifiers that are end-to-end trainable, as opposed to traditional ML models that use handcrafted features.

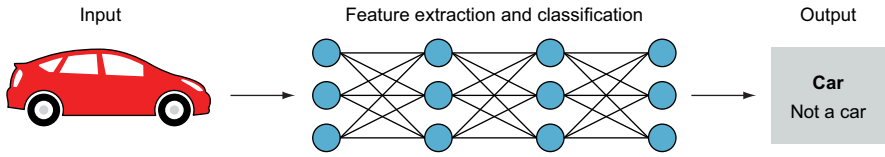
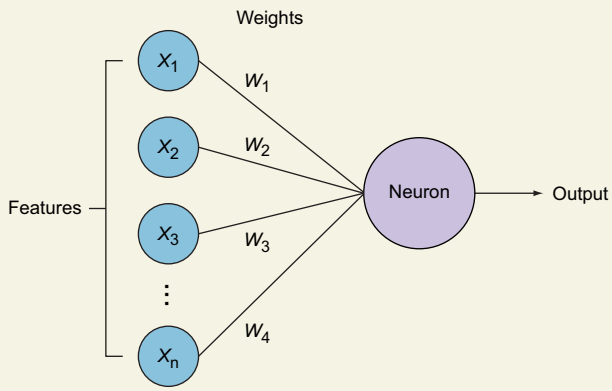



**Figure 1.27 A deep neural network passes the input image through its layers to automatically extract features and classify the object. No handcrafted features are needed.**

### How do neural networks distinguish useful features from non-useful features?

You might get the impression that neural networks only understand the most useful features, but that's not entirely true. Neural networks scoop up all the features available and give them random weights. During the training process, the neural network adjusts these weights to reflect their importance and how they should impact the output prediction. The patterns with the highest appearance frequency will have higher weights and are considered more useful features. Features with the lowest weights will have very little impact on the output. This learning process will be discussed in deeper detail in the next chapter.




**Weighting different features to reflect their importance in identifying the object**

### WHY USE FEATURES?

The input image has too much extra information that is not necessary for classification. Therefore, the first step after preprocessing the image is to simplify it by extracting the important information and throwing away nonessential information. By extracting important colors or image segments, we can transform complex and large image data into smaller sets of features. This makes the task of classifying images based on their features simpler and faster.

Consider the following example. Suppose we have a dataset of 10,000 images of motorcycles, each of 1,000 width by 1,000 height. Some images have solid backgrounds, and others have busy backgrounds of unnecessary data. When these thousands of images are fed to the feature extraction algorithms, we lose all the unnecessary data that is not important to identify motorcycles, and we only keep a consolidated list of useful features that can be fed directly to the classifier (figure 1.28). This process is a lot simpler than having the classifier look at the raw dataset of 10,000 images to learn the properties of motorcycles.

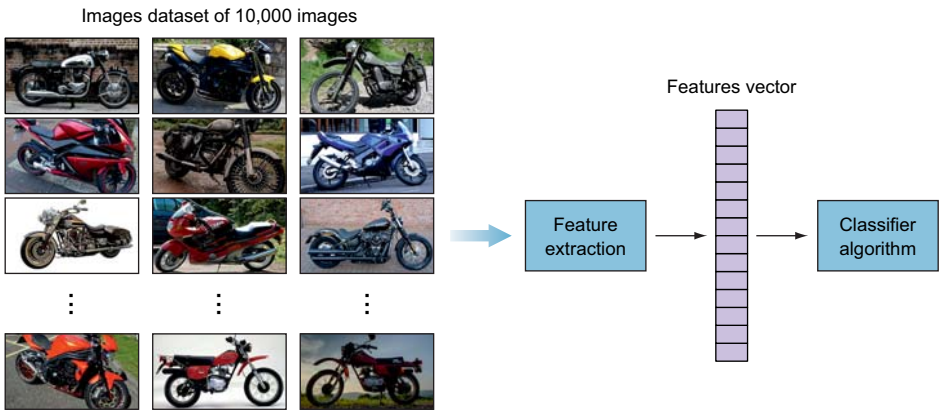



**Figure 1.28 Extracting and consolidating features from thousands of images in one feature vector to be fed to the classifier**

## 1.7 Classifier learning algorithm

Here is what we have discussed so far regarding the classifier pipeline:

- *Input image*—We've seen how images are represented as functions, and that computers see images as a 2D matrix for grayscale images and a 3D matrix (three channels) for colored images.
- *Image preprocessing*—We discussed some image-preprocessing techniques to clean up our dataset and make it ready as input to the ML algorithm.
- *Feature extraction*—We converted our large dataset of images into a vector of useful features that uniquely describe the objects in the image.

Now it is time to feed the extracted feature vector to the classifier to output a class label for the images (for example, motorcycle or otherwise).

As we discussed in the previous section, the classification task is done one of these ways: traditional ML algorithms like SVMs, or deep neural network algorithms like CNNs. While traditional ML algorithms might get decent results for some problems, CNNs truly shine in processing and classifying images in the most complex problems.

In this book, we will discuss neural networks and how they work in detail. For now, I want you to know that neural networks *automatically* extract useful features from your dataset, and they act as a classifier to output class labels for your images. Input images pass through the layers of the neural network to learn their features layer by layer (figure 1.29). The deeper your network is (the more layers), the more it will learn the features of the dataset: hence the name *deep learning*. More layers come with some trade-offs that we will discuss in the next two chapters. The last layer of the neural network usually acts as the classifier that outputs the class label.

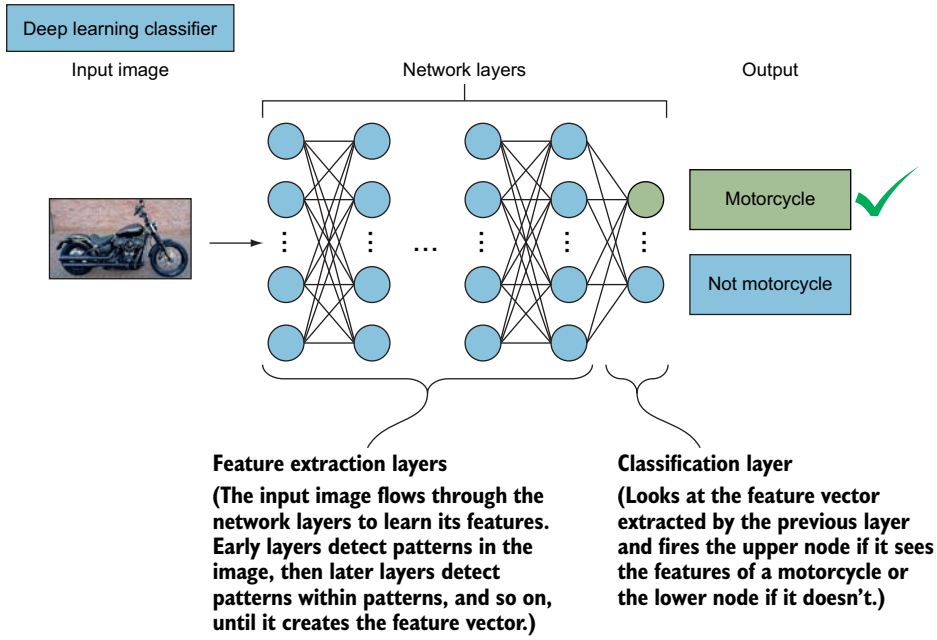



**Figure 1.29 Input images pass through the layers of a neural network so it can learn features layer by layer.**

## Summary

- Both human and machine vision systems contain two basic components: a sensing device and an interpreting device.
- The interpreting process consists of four steps: input the data, preprocess it, do feature extraction, and produce a machine learning model.

- An image can be represented as a function of *x* and *y*. Computers see an image as a matrix of pixel values: one channel for grayscale images and three channels for color images.
- Image-processing techniques vary for each problem and dataset. Some of these techniques are converting images to grayscale to reduce complexity, resizing images to a uniform size to fit your neural network, and data augmentation.
- Features are unique properties in the image that are used to classify its objects. Traditional ML algorithms use several feature-extraction methods.