**Computer Vision on the Raspberry Pi 4**

[**https://github.com/LinkedInLearning/computer-vision-3153829**](https://github.com/LinkedInLearning/computer-vision-3153829)

### **Getting started with computer vision**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Matt] The Raspberry Pi is the most popular, single board computer in the world. Despite its low cost and small size, it provides an amazing amount of computing power. Developers have written many great applications for it. I'm going to show you how to code applications capable of computer vision. These applications can capture images from a camera and then detect or recognize objects using machine learning. My name is Matt Scarpino. I'm the author of "TensorFlow for Dummies," and I've been writing software for over 20 years. I've always been fascinated with computer vision and I'm thrilled that the Raspberry Pi is powerful enough to run real world computer vision applications. If you want to teach a Raspberry Pi how to detect and recognize objects, then join me in my LinkedIn course on computer vision on the Raspberry Pi 4.

### **What you should know**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] This course explains how to implement machine learning applications on the Raspberry Pi. Therefore, the first requirement is that you own a Raspberry Pi single-board computer, and that you've connected it to the internet. Some of the example applications process images captured by a camera. If you'd like to run these applications, you'll need to connect a compatible camera to the Raspberry Pi. This course doesn't require any background in machine learning, but you should have a solid understanding of Python programming. That is, you should be comfortable working with classes, methods, modules, and packages. If you need to brush up on Python, I recommend the Python Essential Training course by Bill Weinman.

### **Using the exercise files**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] The files for this course are provided on GitHub. To access them on a Raspberry PI, you'll need to open a terminal by clicking the right most icon in the main menu. You probably already have Git installed, but just in case, you can install it with the following command. sudo apt install git To clone the repository for this course, you'll need to enter the following command. git clone https://github.com/LinkedInLearning/ computer-vision-3153829.git. This creates a new directory with the name of the repository. I'll change to this directory with a CD command. The files for this course are provided in different branches. You can see the list of remote tracking branches using the command, git branch -a. Each branch is named according to the chapter and video. Branches ending in B contain preliminary code and branches ending in E can contain finished code. For example, suppose you want to access the final code in the fifth video of chapter two, the branch containing the code is 02\_05e, and you can access it with git checkout. Now, when you look in the directory, you'll see the files rpi4.jpg and opencv\_conv2d.py. As another example, suppose you want to access the beginning code of the third video of chapter five. In this case, you'd enter the command, git checkout 05\_03b. Now you can see a file called train\_cnn.py and a folder called train images. I'll have much more to say about these files in the third video of chapter five. If you run into trouble cloning the repository or accessing its branches, I recommend searching the LinkedIn Learning Library for the Git Essential Training Videos.

### **Introducing the Raspberry Pi 4**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] This course explains how to code computer vision applications for the Raspberry Pi single-board computer. Since it's released in 2012, the Raspberry Pi has become famous because it provides the capabilities of a traditional computer at low cost, and a tiny form factor. Despite having the size of a credit card, it's powerful enough to run a full operating system, surf the internet, and analyze images from a camera. Before I explain the Raspberry Pi software, I'd like to take some time to discuss its hardware. This slide shows the components that make up the Raspberry Pi for Model B. The most important component is the processor, Broadcom BCM2711 runs at 1.5 gigahertz, contains a 64 bit arm processor, and can access up to eight gigabytes of high-speed Ram. On the right, IO ports make it possible to communicate with other devices. You can connect a keyboard, and a mouse using the USB ports, and you can connect to a network using the RJ45 network jack. In the lower left, the USB-C connector provides the board with power. It's recommended to connect this to a power supply capable of delivering 5.1 volts at a maximum of three amps. To the right of the power connector, two micro HTMI ports, make it possible to connect the board to monitors. To the right of the HTMI connectors, the 15 pin connector makes it possible to connect the board to a camera. To derive the most benefit from this course, I recommend connecting a compatible camera, such as the Raspberry Pi high quality or HQ camera. This slide shows how I've connected the Raspberry Pi for running computer vision applications. I've connected my keyboard, and mouse to the USB ports, and I've connected an ethernet cable to access my home network. I've connected a power supply to the lower left, and a micro HTMI cable to the left HTMI connector. To the right of the HTMI connectors, I've connected the Raspberry Pi high-quality camera module. The Raspberry Pi doesn't have a traditional hard drive. Instead, it reads and writes data from a micro SD card on the rear of the board. This must be programmed with a suitable operating system, and this can be easily accomplished using the Raspberry Pi imager, which is free for download from RaspberryPi.org The Raspberry Pi can run several different operating systems, including Ubuntu and Android. But for this course, I'll assume that you're running the official Raspberry Pi OS, formerly called Raspbian. The Raspberry Pi provides a tremendous amount of power in a small package. It won't replace your smartphone or desktop PC, but it's capable of running a full operating system, and computer vision applications.

### **Setting up the environment**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] When you apply power to the Raspberry Pi and connect it to a display, the desktop should look similar to what you see here. This is the Raspberry Pi desktop environment. Along the top, you can see four icons. The leftmost icon opens the Applications menu. The next icon opens the Chromium web browser. The next icon opens the File Manager, and the last icon opens a terminal for entering commands. Before you start programming, you'll need to set up the environment to support computer vision. This means installing new packages and upgrading existing packages. To enter commands, click the rightmost icon to open a terminal. At this point, I'd like to provide some background on the operating system. Raspberry Pi OS is based on a Linux distribution called Debian. First released in 1993, Debian has become the basis of over 100 Linux distributions, including Ubuntu and Kali Linux. One major reason for Debian's popularity is that users can customize the OS using packages. A package contains all the files needed to run a program or a set of programs. You can access your system's packages using the advanced package tool or APT. For example, you can list the installed packages by entering the command apt list --installed. As you can see, the Raspberry Pi comes with hundreds of packages. Debian currently provides over 51,000 packages and they're all free. To change your packages, you'll need administrative privileges, which aren't available by default. Therefore, you'll have to begin commands with sudo which allows you to perform operations as a super user. With this in mind, you can update your list of packages with the command sudo apt update. This checks to see if there are any additional packages that should be downloaded. It may take a few minutes depending on your system and network connection. After updating the package list, the next step is to upgrade packages to their latest versions. **This can be accomplished with the command sudo apt full-upgrade. Computer vision requires a lot of math and graphical opeor image processing. We can install both with one command, sudo apt install libatlas base-dev libjasper dev. All of the code in this course is written in Python, so the most important package is Python 3. We can verifrations, so we're going to install two packages. The first Atlas is needed for linear algebra and the second Jasper is needed fy that it's present with the command apt list python3.** As shown, my system has Python 3.7 installed. You'll probably have a later version. To run the computer vision applications in this course, you'll need to install OpenCV. This is a Python package, not a Debian package, so you'll have to use Python's pip3 tool. The package's name is OpenCV Python and you can install it with the command pip3 install opencv-python. This is provided as a whl file, which is why the downloaded file has the whl suffix. Once the download is finished, OpenCV is installed and you're ready to start coding. This video has discussed the Raspberry Pi OS which can be customized using packages. We've also seen how to install Debian and Python packages needed to run computer vision applications.

### **Using the Thonny IDE**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] The Raspberry Pi provides two development environments for Python, and you can find them by opening the Applications menu, and going to the Programming sub menu. The two options are Geany and Thonny. Geany has more features, but Thonny is easier to use. For this reason, I'll use Thonny throughout the course. If you open Thonny, you'll see that the environment has three parts: the toolbar, the editor area, and the shell area. The editor occupies most of the screen. And, at the bottom, the shell displays messages from the Python interpreter. When an application prints a message, or throws an error, the shell displays the result. To see how Thonny works let's write a simple program that uses OpenCV to create and display an image. We can access OpenCV's capabilities with the statement import cv2. In addition to OpenCV, we'll need the numeric Python package, or NumPy. We can import NumPy with the statement, import numpy as np Now, that we've imported OpenCV and NumPy, let's create a 600 by 400 image whose pixels are set to 0.5. We'll call the array, img\_array and initialize it by calling NumPy's full function with the array size 400 by 600, (typing) and the value of each element, 0.5. Next, we'll convert the array to an image, and display the image by calling OpenCV's I am show function (typing) This accepts a name for the window, (typing) and the array. (typing) I'd like the window to stay open until a key is pressed. This requires calling the wait key function (typing) and the destroy all windows function. (typing) Now, I'll press the Run button in the toolbar. A window appears and displays a gray image. I'll close it by pressing a key. This video has explained how to use Thonny. Thonny isn't the most powerful Python environment, but it makes it easy to edit and run programs. We'll use Thonny throughout this course to demonstrate Python development.

## **Question 1 of 3**

How does the Raspberry Pi access its operating system?

* through HDMI
* through the SD card  
  Correct  
  The Raspberry Pi accesses its OS through the SD card reader on the rear of the board.
* through the RAM chip
* through a USB port

## **Question 2 of 3**

What utility does the Raspberry Pi use to manage packages?

* apt  
  Correct  
  The Raspberry Pi relies on APT (Advanced Package Tool) to manage packages.
* npm
* upm
* yum

## **Question 3 of 3**

Which part of Thonny displays messages from the Python interpreter?

* shell area  
  Correct  
  Thonny displays interpreter messages in the shell area.
* toolbar
* editor area

### **Introducing OpenCV**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Earlier, I explained how to install OpenCV and use some of its basic features. Now, I'd like to take a step back and provide a proper introduction. OpenCV's full name is the Open Source Computer Vision Library, and Intel released the alpha version in 2000. Their goal was to advance vision research by providing a common infrastructure that developers could build on. Since then, developers have added many new capabilities to OpenCV. These include stereoscopic vision, video stabilization, and facial analysis. In this course, we'll focus on using OpenCV for object detection. Object detection determines if an object is present in an image, and if so, where it's located. Most of the OpenCV applications in this course perform the same set of operations. First, the application reads an image by calling imread or by calling the capture function of the PiCamera package. Next, the application examines the image using OpenCV's analysis capabilities. These include support vector machines, neural networks, and decision trees. When the analysis is complete, applications can display the results by drawing graphics on the image, such as a rectangle around a detected object. Then, the resulting image can be saved to a file by calling imwrite. Toward the end of the course, I'll explain how to capture images from a camera. Until then, we'll read images from files by calling imread. This accepts two arguments, the path of the file containing the image and the image's type. All of the images in this course will be color, so we'll set the type to IMREAD\_COLOR. For example, if you want to read a color image from smiley.jpeg, you'd call imread with the first argument set to the full path of smiley.jpeg and the second argument set to IMREAD\_COLOR. When an image is read from a file, OpenCV returns the image's data in a NumPy ndarray. This stands for n-dimensional array. For a color image, imread will return an array with three dimensions, one for the image's height, one for its width, and one for each channel in the image's pixels. This slide shows what happens when imread reads the color image on the left. The array has three dimensions, but you can think of it as having three two-dimensional arrays, one for each color channel in the image's pixels. Most programs access color in RGB order with red first, green second, and blue last, but OpenCV stores colors in BGR order with blue first. After an application analyzes an image, it can use OpenCV's drawing functions to draw graphics. These graphics may include boundary lines, bounding boxes, or text. The three main drawing functions are line, rectangle, and putText. Each is straightforward to understand, and I'll demonstrate how they're used later on. I like to use putText when I need to figure out why my application isn't analyzing the right region of the image. Unfortunately, the font selection is limited. Two options are FONT\_HERSHEY\_SIMPLEX, which is normal-sized sans-serif, and FONT\_HERSHEY\_COMPLEX, which is normal-sized serif. After an application analyzes an image and draws graphics, it can save the image to a file by calling imwrite. The first argument identifies the name of the file, and OpenCV can create several types of images, including JPEGs, PNGs, TIFs, and Windows Bitmaps. The second argument identifies the ndarray containing the image's data. This example code shows how you can save the data from image\_array to an image file named out\_image.png. OpenCV is a powerful toolset that provides a wealth of capabilities for computer vision. This video has presented many of the simple functions, and later videos will present more advanced features.

### **NumPy array operations**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In an earlier video, I explained that OpenCV stores image data in NumPy arrays. When it comes to crunching numbers, these arrays are much more efficient than regular Python containers. Also, NumPy provides several capabilities that aren't available when using lists and dictionaries. It would take an entire course to present all of NumPy's features. For this course, you only need to know four of them: types and type conversion, slicing, centering, and standardization. Elements in a Python list can have any type, but elements in a NumPy array always have the same type. The arrays in this course all contain numbers, so we'll be concerned with signed integers, unsigned integers, and floating point values. Each type is represented by a field of the NumPy module. Integers are given by int or uint and the number of bits in each value. Floating point values are given by float32 or float64. In many cases, we'll need to convert an array from one type to another. This is accomplished by calling the astype method with the desired type. For example, if you want to convert int\_array to double\_array, you'd call astype with the np.float64 field. If you want to access multiple values of a NumPy array, you can take advantage of slicing. This defines a range of indexes using two values separated by a colon. The first value is the first index of interest and the second value is the value after the last index of interest. For example, you can access the second element of num\_array by inserting one in square brackets. You can access the third, fourth, and fifth elements by following num\_array with 3:6 in square brackets. If the value before the colon is omitted, NumPy will set it equal to zero. If the value after the colon is omitted, NumPy will set it equal to the last index in the array. If both values are omitted, NumPy will assume you want to read the entire array. Computer vision applications commonly analyze several images in a sequence. To ensure that images are statistically similar, it's common to set their mean values to zero and their standard deviations to one. The first step is called centering. An array is centered if the mean of its elements is zero. An easy way to center a NumPy array is to call np.mean and subtract this from the elements of the array. For example, you can center num\_array using num\_array minus equals np.mean of num\_array. Keep in mind the np.means returns a float64 and the subtraction can only be performed if num\_array contains float64 values. In addition to making sure images have the same mean, we want their values to have the same average distance from the mean. The average distance from the mean is called standard deviation. And an array is standardized if its standard deviation is one. To standardize an array, call np.std to obtain the standard deviation, then divide each element of the array by this value. For example, you can standardize num\_array using num\_array slash equals np.std of num\_array. When this is executed, the average distance of num\_array's values from the mean will be one. This video has presented four features of NumPy that are commonly encountered in computer vision applications. Later videos will rely on these operations to analyze images.

### **Running a simple image processing example**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] To see how OpenCV and NumPy process images. It helps to work through a simple example. This video shows how to read an image from a file, modify its content and save it to another file. To be specific, I'm going to perform five operations. I'll start by removing the blue content from the image and making its first 10 rows gray. Then I'll center it by subtracting it's mean and standardize it by dividing it by its standard deviation. To start, I'll create an image named image array. I'll read the image data by calling CV2.imread. This accepts a file name, which I'll set to car.jpg and an image type, which all set to CV2.imread\_color. You can think of image array as a set of two dimensional arrays, one for blue, one for green, and one for red. To remove the blue content from the image, the third array must be set to zero and the pixels must be set to zero. The first dimension of the array corresponds to the images height. This means we can make the first 10 rows gray by slicing the first dimension, and setting its pixels to 128. Before I can center the array, it needs to be converted to a floating point array. I'll do this by calling the as type method with np.float64. And then I'll subtract the mean. Standardizing the image is just as simple. Just divide the array, by its standard deviation. To make sure that the centering and the standardization worked properly, I'll create a string named stats and I'll have this display the mean and the standard deviation. And I'll compute these by calling np.mean and to np.std. After pre-processing, most of the arrays images lie between minus three and three. This is great for computer vision, but before we can write the array to a file, we need to make sure its values range from zero to 255. I'll do this by calling the normalize method. This accepts an input array and an output array, which all set to none. Then it accepts the minimum of value, which I'll set to zero and the maximum, which I'll set to 255. The next argument is the norm type, which I'll set to CV2.norm\_minmax. The last argument, sets the type of the values to be returned. I want the output array to contain unsigned bytes. So I'll set this to CV2.CV\_8U. Now I'll draw a string on the image by calling put text. This accepts the input array and the string to be printed. The next argument takes the coordinates of the lower left corner, and I'll set this to 10, 40. I'll set the font to CV2.font-hershey\_simplex. I'll set the scaling to 0.4 and I'll set the color to white, which has given us 255, 255, 255. For the last step, I'll save the array to a file by calling CV2.imwrite. This accepts the file name, which I'll set to newimage.jpg and the array imagery. When I run this, it will create the new output image file. This slide shows what the processed image looks like. The first 10 rows are gray and the blue content has been completely removed. As the text shows the mean is very close to zero and the standard deviation is very close to one. This video has demonstrated how OpenCV and NumPy worked together for image processing. Computer vision relies on both packages and we'll use their capabilities throughout the course.

### **Theory of convolution**

Selecting transcript lines in this section will navigate to timestamp in the video

- Now that you understand basic image processing. It's time to look at convolution. Convolution plays a central role in computer vision, and the better you understand it, the better you'll understand the content of this course. Convolution accepts an image and a two dimensional array called a kernel. It produces a second image with the same size as the original. If you've used tools like blur and sharpen in Photoshop, then you should have an idea of what convolution can do. Many sources present convolution using equations, but I'm going to approach the topic more gradually I'll start by discussing dot products in one and two dimensions. Once this is clear, I'll explain how convolution works. The dot product is a common operation in linear algebra and graphical rendering. It measures the similarity of two arrays by multiplying their corresponding elements and adding the products together. This slide computes the dot product of two arrays named A and B. Many of the elements have opposite signs. So the result is negative. If the elements all have the same signs, the dot product would be positive. A two dimensional dot product is similar to a one-dimensional dot product, but multiplies elements of two dimensional arrays and adds the products together. This slide illustrates the dot product of two, three by three arrays. In this example, the elements in the arrays all have the same sign, as a result, The dot product must be positive. In essence, convolution is just a series of two dimensional dot products for each pixel in an image convolution computes the dot product of the images values with values from a two dimensional array called a kernel. The result is stored as a pixel and the output image at the location where the dot product was taken. Colonels are much smaller than the image, and they're almost always square. The Colonel's values, make it possible to measure relationships between adjacent pixels in the image. The best way to understand convolution is to look at some examples, in the following discussion, I'll explain how convolution can be used to blur and sharpen images. The simplest way to blur an image is to take the average of each pixel and its surrounding pixels, and then replace the pixel with this average, the average can be computed by summing all the values and dividing by nine. The same result can be obtained by setting the values of a three by three Colonel equal to one ninth and taking the two dimensional dot products. If an image is converged with this kernel, then each pixel will be replaced with the corresponding average. This operation is called a box blur. This slide shows what a box blur looks like. The image on the right is the blurred version of the image on the left. Blurring works by reducing differences between adjacent pixels. We can also sharpen images by accentuating pixel differences. In this example, the Colonel multiplies the central pixel by five and subtracts adjacent pixels. On the right, you can see the result. Sharp differences in the input image produce very bright pixels or very dark pixels in the output. This video has explained the basics of convolution, which performs a series of two-dimensional dot products. Convolution makes it possible to blur and sharpen images. In later videos we'll present more interesting applications.

### **Convolution in OpenCV**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] This video demonstrates how convolution can be performed using OpenCV. The most important function to know is filter2D. And in most applications, you only need to be concerned with the first three arguments. Src identifies the NumPy array containing the input image. Ddepth sets the data type of the output values. If this is set to minus one, the output values will all have the same type as the input values. Kernel identifies the NumPy array containing the kernel values. The function's return value is the image produced by the convolution. To start the program, I'll create an array named img\_array and call cv2.imread to read data from rpi4.jpg and I'll set the image's type to cv2.IMREAD\_COLOR. Then I'll call the display function to present the image in a window. This accepts the window's name, which I'll call Original and the array containing the image's pixels. To demonstrate box blurring, I'll create box\_kernel by calling NumPy's full function. This accepts the size of the array, which I'll set to 3, 3, and the value of each element 1/9. To perform convolution with box\_kernel, I'll set box\_result equal to cv2.filter2D. This accepts the img\_array, a value of minus one, and box\_kernel. Then I'll call the display function to present the image in a window. This time, I'll call the window Box Blur and I'll pass it box\_result. Box Blurs are simple to understand, but we can usually get a better smoothing result using a Gaussian blur. To demonstrate this, I'll set gaussian\_kernel equal to np.array with the values of the kernel. The first row of the kernel is 1, 2, 1. The second row is 2, 4, 2. And the last row is 1, 2, 1. And all the values need to be divided by 16.0. To obtain the convolution result, I'll set gaussian\_result equal to cv2.filter2D. This accepts img\_array, a value of minus one, and gaussian\_kernel. Then I'll call the display function to present the image in a window. I'll call the window Gaussian Blur and display gaussian\_result. In addition to blurring images, convolution can sharpen images by magnifying differences between pixels. To demonstrate, I'll set sharpen\_kernel equal to np.array with the kernel's values. The first row of the kernel is 0, -1, 0. The second row is minus 1, 5, -1. And the last row is 0, -1, 0. To obtain the convolution result, I'll set sharpen\_result equal to cv2.filter2D. This accepts the img\_array, a value of minus one, and sharpen\_kernel. Then I'll call the display function to present the image in a window. I'll call the window Sharpen and pass in sharpen\_result. At this point, the program is coded to perform three convolutions. Now I'll run the script. Here's the original image, and here's the result of the box blur. It does a good job at blurring most of the image, but if you look closely, you'll see that many of the corners are jagged. Here's the result of the Gaussian blur. As you can see, it does a better job of smoothing the entire image. Finally, the sharpen operation makes small details more apparent because it accentuates differences between pixels. Convolution is very useful and can be used for blurring and sharpening images. This video has demonstrated how it can be performed using OpenCV's filter2D function.

## **Question 1 of 5**

Which OpenCV function performs image convolution?

* imread
* cvtColor
* tensorConv
* filter2D  
  Correct  
  To perform image convolution with OpenCV, an application needs to call the filter2D function.

## **Question 2 of 5**

What color format does OpenCV use when accessing pixels?

* BRG (blue-red-green)
* BGR (blue-green-red)  
  Correct  
  OpenCV accesses image data using the BGR (blue-green-red) color format.
* RGB (red-green-blue)
* RBG (red-blue-green)

## **Question 3 of 5**

Convolution computes dot products of pixels in an image and values in a \_\_\_\_\_.

* rect
* box  
  Incorrect  
  Boxes are not involved in convolution.
* kernel  
  Correct  
  Convolution computes dot products of pixels in an image and values in a kernel.

## **Question 4 of 5**

When OpenCV reads an image from a file, what data structure does it use to hold the data?

* Python list
* TensorFlow tensor
* NumPy array  
  Correct  
  In Python code, OpenCV relies on NumPy arrays to hold image data.

## **Question 5 of 5**

What happens when you center your data?

* You set the standard deviation to 1
* You set the minimum value to 0
* You set the mean to zero  
  Correct  
  Data is centered if its mean has been set to 0.
* You set the maximum value to 255

### **Computing image gradients**

Selecting transcript lines in this section will navigate to timestamp in the video

- At this point, you should understand what convolution is and how it can be used to blur and sharpen images in this video, I'll explain what image gradients are and show how they can be computed, Using convolution. A gradient is a special kind of vector. And before I go further, I'd like to explain the basics of vectors. A vector is a quantity with magnitude and direction. If you've taken courses in physics or graphics, you've probably seen vectors represented by arrows. The arrows length identifies the vectors magnitude and its direction identifies the vectors direction. We described vectors in terms of their components. This slide illustrates a two dimensional vector named V and its component VX and VY VX is the component in the X axis. And V Y is the component in the y-axis. The magnitude of V equals the square root of the sum, of VX squared and VY squared. These orientation is the angle the vector makes with the horizontal axis. This is denoted theta, and it equals the inverse tangent of VY. Over VX. A gradient is a vector measured on a changing surface. The gradients magnitude is the greatest rate of change, and it points in the direction of the greatest rate of change. To understand this, imagine that you're standing at a point P on this hill. If you want to get up the hill in the least time, you'll climb in the direction of the vector G. this is the direction of greatest descent. So G is the gradient at point P Ge's magnitude equals the slope of the surface at point P. slope equals rise over run, which is DY over DX in this illustration. An image gradient at a pixel is the vector that points in the direction of the greatest change of intensity. change of intensity is positive, If the pixel colors change from dark to light. The image on the left presents an example. intensity increases the most, when you move from left to right. So the gradient G at point P points from left to right on the right intensity increases the most when you move from top to bottom. So the gradient G at point P points downward. to compute image gradients, we use convolution. one popular method is to convolve an image with the two kernels defined by the Nobel operator. These kernels are called SX and SY. And this slide shows what they look like. convolution with SX tells us how intensity changes along the horizontal axis. The result is called GX and a positive value implies that intensity changes from left to right. convolution with SY tells us how intensity changes, along the vertical axis. The result is called GY and a positive value implies that intensity increases from top to bottom. GX and GY are the components of the image grading at G. Ge's magnitude equals the square root of GX squared, plus GY squared. and its orientation equals the inverse tangent of GY. Over GX. This video has explained what image gradients are and how they can be computed using convolution. later on I'll explain how these gradients make it possible to detect in objects and images.

### **Forming histograms of gradients (HOGs)**

Selecting transcript lines in this section will navigate to timestamp in the video

- Now that you understand what image gradients are, you're ready to see how they're used in computer vision. This video presents an important operation called the Histogram of Oriented Gradients method. Also called the HOG method. This has three steps, and the first is to compute image gradients for the pixels in a region. Then it combines these image gradients to form histograms, and concatenates the histograms to create a feature descriptor for the region. Like a fingerprint, a feature descriptor serves as a unique identifier for a set of pixels. Object detection relies on these feature descriptors to determine if an object is present in the region before proceeding further, I'd like to review the topic of histograms. A histogram divides a range of values into intervals and counts how many values fall into each interval. For example, suppose you own a store and you want to study the ages of your customers a graph that displays every age won't be helpful. So you split the range into five-year intervals. The resulting histogram shows the age distribution, the Y axis counts how many values belong to each interval. Histogram intervals are commonly called bins. In this example, the histogram splits values into 12 bins and each bin has a width of 5. To form a histogram with oriented gradients. We start by computing the image gradient of each pixel in a region. Then we split the gradients into bins, according to their orientation. It's common to use 9 bins that are each 20 degrees wide. A major difference between HOGs and regular histograms is that HOGs don't count how many gradients belong to each bin. Instead we add each gradients magnitude to the value of its bin. The bottom part of this slide presents an example on the left. We've computed the gradients magnitude and orientation in 4 adjacent pixels. The HOG on the right shows how the gradients have been combined. The first gradient has an orientation of 25 and the fourth gradient has an orientation of 35, both belong to the second bin, which ranges from 20 degrees to 39 degrees. The bins value 10 equals the sum of their magnitudes 8 and 2 the maximum orientation is 180 because OpenCV uses unsigned gradients by default. This means a gradient with 270 degree orientation will be processed like a gradient with 90 degree orientation. In practice. We don't compute a histogram for an entire image at once. Instead we create a sliding window and repeatedly move its position throughout the image. A window is made up of square regions called cells. In this slide, the window contains a rectangular grid of cells and each cell contains a square grid of pixels. The HOG method computes a histogram for each cell and concatenates them together to form a single vector that serves as the windows feature descriptor. Then, the method determines if an object is present by comparing the windows feature descriptor with the objects feature descriptor. To make object detection independent of lighting changes. OpenCV uses a method called block normalization. This splits each feature descriptor into subvectors and normalizes each subvector, normalization divides each of the subvectors components by the square root of the sum of the components squared. The equations at the bottom of the slide shows how this works. If a V has components V0 through V5, block normalization divides each component by K, where K is the square root of the sum of V0 through V5 squared. The HOG method plays a central role in the object detection. This video has explained how HOGs are computed and how they're used to detect objects.

### **Forming histograms of gradients (HOGs)**

Selecting transcript lines in this section will navigate to timestamp in the video

- Now that you understand what image gradients are, you're ready to see how they're used in computer vision. This video presents an important operation called the Histogram of Oriented Gradients method. Also called the HOG method. This has three steps, and the first is to compute image gradients for the pixels in a region. Then it combines these image gradients to form histograms, and concatenates the histograms to create a feature descriptor for the region. Like a fingerprint, a feature descriptor serves as a unique identifier for a set of pixels. Object detection relies on these feature descriptors to determine if an object is present in the region before proceeding further, I'd like to review the topic of histograms. A histogram divides a range of values into intervals and counts how many values fall into each interval. For example, suppose you own a store and you want to study the ages of your customers a graph that displays every age won't be helpful. So you split the range into five-year intervals. The resulting histogram shows the age distribution, the Y axis counts how many values belong to each interval. Histogram intervals are commonly called bins. In this example, the histogram splits values into 12 bins and each bin has a width of 5. To form a histogram with oriented gradients. We start by computing the image gradient of each pixel in a region. Then we split the gradients into bins, according to their orientation. It's common to use 9 bins that are each 20 degrees wide. A major difference between HOGs and regular histograms is that HOGs don't count how many gradients belong to each bin. Instead we add each gradients magnitude to the value of its bin. The bottom part of this slide presents an example on the left. We've computed the gradients magnitude and orientation in 4 adjacent pixels. The HOG on the right shows how the gradients have been combined. The first gradient has an orientation of 25 and the fourth gradient has an orientation of 35, both belong to the second bin, which ranges from 20 degrees to 39 degrees. The bins value 10 equals the sum of their magnitudes 8 and 2 the maximum orientation is 180 because OpenCV uses unsigned gradients by default. This means a gradient with 270 degree orientation will be processed like a gradient with 90 degree orientation. In practice. We don't compute a histogram for an entire image at once. Instead we create a sliding window and repeatedly move its position throughout the image. A window is made up of square regions called cells. In this slide, the window contains a rectangular grid of cells and each cell contains a square grid of pixels. The HOG method computes a histogram for each cell and concatenates them together to form a single vector that serves as the windows feature descriptor. Then, the method determines if an object is present by comparing the windows feature descriptor with the objects feature descriptor. To make object detection independent of lighting changes. OpenCV uses a method called block normalization. This splits each feature descriptor into subvectors and normalizes each subvector, normalization divides each of the subvectors components by the square root of the sum of the components squared. The equations at the bottom of the slide shows how this works. If a V has components V0 through V5, block normalization divides each component by K, where K is the square root of the sum of V0 through V5 squared. The HOG method plays a central role in the object detection. This video has explained how HOGs are computed and how they're used to detect objects.

### **Understanding Support Vector Machines (SVMs)**

Selecting transcript lines in this section will navigate to timestamp in the video

- Once you've obtained a set of feature descriptors, the next step is to send them to a classifier, such as a support vector machine or SVM. This video explains what SVMs are and how they work. To use open CV's SVM class, you only need three lines of code. The first creates the SVM and the second calls it to train method. Train, accepts a set of vectors and their labels, and it teaches the SVM how to classify vectors into categories. The last step calls predict to classify an unknown vector. If a vector has 'n' element, you can think of it as a point 'n' dimensional space. This slide illustrates points in two sets, A and B, which are separated by lines L1 and L2. L1 and L2 are called boundaries because they separate the points into two sets. L2 is a better boundary because it's points are further away from this point in set A and set B. The goal of an SVM is to find the best boundary that separates sets of points. This maximizes the distance from the boundaries points to the points in each set. This boundary will be align if the points are two dimensional, and a plane, if the points are three-dimensional. If points have more than three dimensions, the SVM will compute a hyperplane to serve as the boundary. In practice, SVMs always create hyper planes, but when you're learning the theory, it's easier to work with three-dimensional points in planes. To see how an SVM computes boundaries, you need to be familiar with plain geometry. In this diagram, 'n' represents any vector perpendicular to the plane and 'X' is a vector from the origin to any point P on the plane. This plane intersects the origin. So 'X' always lies in the plane. This means 'n' is always perpendicular to X, and this can be expressed mathematically by saying that the dot product of n and x is zero. This serves as the equation of a plane that passes through the origin. If a plane doesn't pass through the origin, x doesn't lie in the plane, and n and x aren't perpendicular. In this case, we draw a new vector called 'n prime', which is perpendicular to the plane and passes through the origin. Subtracting x from n prime, produces a vector that lies in the plane. Therefore, the dot product of n and n prime minus x is zero. If we set b equal to minus n prime dot n we arrive at a new plane equation, n dot x plus b equals zero. Two planes are parallel if they have the same vector n, but different values of b. If one plane has equation N dot x0 plus b0 equals zero and another has equation N dot x1 plus b1 equals zero, The two planes are parallel. In this diagram, x1 extends from the origin to the top plane and x0 extends from the origin to the bottom plane. The vector kn points from the bottom plane to the top plane. So kn equals x1 minus x0. Replacing this relationship in our equations, the distance between the two planes equals b0 minus b1 over the length of n. This slide shows how parallel planes are used to separate points. We want to find one plane that bounced the points in set A, and the parallel plane, that bounced the points in set B. The ideal boundary will lie halfway between them. To find these planes, an SVM adds a dimension called 'y', to each point and plane in the space. y equals minus one for each point in set A, one for each point in set B, and zero for each point on the boundary plane. For the left plane, we want n dot x plus b to be less than or equal to minus one. For the right plane. We want n dot x plus b to be greater than or equal to one. We also want to maximize the distance between these planes. This distance is called the margin, and it equals to, over the length of n. Now that we've obtained these relationships, we can use optimization theory to find the best values for n and b. Then we can classify unknown points by computing n dot x plus b. A positive value, implies that the point belongs to set A. A negative value, implies that it belongs to set B. Object detection relies on SVMs to classify feature descriptors. This video has explained how SVMs can be trained to perform this classification.

### **Detecting objects with HOGs and SVMs**

Selecting transcript lines in this section will navigate to timestamp in the video

- At this point, you should be familiar with how feature descriptors are computed and classified. In this video, I'm going to walk through the full object detection process. Here's the source code from an earlier video before proceeding, I'm going to remove the last two lines and add a line that updates the train labels list. Each training file starts with a number that identifies whether the toy car is present. So I'll split the file name, and cast the result to an integer. (typing) Next, I'll create the support vector machine by setting SVM equal to cv2.ml.SVM\_create. Hog descriptors can only use linear SBMs, so I'll call set Kernel with cv2.ml.SVM\_linear. Now I'll train the SVM with the training features and labels. (typing) I'll save the trained model to an XML file by calling the SVMs safe method. (typing) The next step associates, the SVM with the hog descriptor. This isn't easy because the hog descriptor won't accept the SVM instance. Instead, I need to extract the SVMs support vectors. (typing) I'll also access the first value in the SPM's decision function. (typing) Now, I'll create a vector that combines this information and use it to update the hog descriptor. (typing) Now that the hog descriptor is ready, we'll use it to detect objects in the test images. To begin, I'll set image dir equal to test images. Then I'll create a loop to iterate through the test images. (typing) To prepare each image for detection, I'll set image to the result of pre-process, which accepts the image directory and the image file. To detect objects, I'll set rests equal to the return value of detect multi-scale. (typing) Rest consists of two lists. The first contains rectangles and the second contains confidence values. I'm only interested in rectangles if the confidence value is greater than 1.2 and I'll test this with an if statement. (typing) If this condition is met, I want to draw the rectangle containing the object on the image. I'll find the index of the greatest confidence by setting index equal to np.argmax. (typing) Then I'll obtain the dimensions of the rectangle by accessing the first list. (typing) To draw the rectangle I'll call CV2.rectangle. This accepts the image, the lower left corner, which I'll set to X Y. The upper right corner, which I'll set to X + W, Y + H, the color blue and a line thickness of two. Now I'll present each image in a window by calling display. This accepts a name for the window and the array of pixels. Now I'll run the application. This is clearly an error, but this looks good. The rectangle clearly overlaps the toy car. This is acceptable. This is good. This is acceptable. This is very good. Good, acceptable, good and good. This video has walked through the development of an application that the text, whether a toy car is present in an image, the results are good, but could be improved with more training.

Question 1 of 5

What two quantities does every vector have?

direction, area

magnitude, mean

magnitude, direction

Correct

Every vector has a magnitude and a direction.

area, mean

Question 2 of 5

The goal of a support vector machine (SVM) is to compute \_\_\_\_\_ for sets of data.

boundaries

Correct

The goal of a support vector machine (SVM) is to compute boundaries for sets of data.

sums

nearest neighbors

standard deviations

Question 3 of 5

Which OpenCV class plays a central role in object detection?

ObjFinder

CvHistogram

HOGDescriptor

Correct

OpenCV's HOGDescriptor class plays a central role in object detection.

KalmanFilter

Question 4 of 5

What are the intervals of a histogram commonly called?

ranges

blocks

bins

Correct

The intervals of a histogram are commonly called bins.

gaps

Question 5 of 5

If the window of a grayscale image has a 3-by-4 grid of cells and each cell's histogram has 9 values, how many values are in the window's feature descriptor?

108

96

Incorrect

If a window has a 3-by-4 grid of cells and each cell's histogram has 9 values, its feature descriptor will not have 96 values.

48

256

Incorrect

If a window has a 3-by-4 grid of cells and each cell's histogram has 9 values, its feature descriptor will not have 256 values.

### **Introducing neural networks**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video, we'll put aside object detection and start pursuing object recognition. To be specific, we're going to use convolutional neural networks, or CNNs, to recognize objects in an image. CNNs are very powerful and are used in applications from facial recognition to medical analysis, but they're not easy to understand. For this reason, I'm going to approach the topic gradually. This video presents the basics of neural networks and later videos will show how they can be used to recognize objects. In my opinion, the best way to introduce neural networks is to compare them with neurons. Our brains have millions of neurons, and this slide presents a simplified view of what one looks like. This neuron accepts a set of input signals and produces one output. At a physical level, signals are represented by electrical pulses. If the sum of the input voltages exceeds a threshold voltage, the neuron fires or produces an output signal. If the sum of the inputs is less than the threshold, the neuron doesn't fire at all. Inspired by neurons, Frank Rosenblatt devised a learning algorithm based on circuits called perceptrons. Like a neuron, a perceptron accepts multiple inputs, denoted xi, and adds them together. Then it compares the sum to a threshold, denoted y. If the sum of xi is greater than y, the perceptron fires and produces a value of one. Otherwise, it produces a value of zero. Researchers have improved on the perceptron in many ways and one major improvement involves adding weights and a bias. In this diagram, each input xi has a weight, wi. Now the perceptron multiplies each xi by wi and adds the products together. Researchers have also replaced the threshold with a new input called the bias or b. Now the perceptron fires if the sum of xi times wi plus b is greater than or equal to zero. Rather than limit the output to zero or one, each perceptron is assigned a function that produces a range of output values. This is called the activation function, and researchers have devised several activation functions for neural networks. This slide presents two of the most popular. The first is the rectified linear unit function or ReLU. This returns the input value if it's positive and zero otherwise. The second activation function is the hyperbolic tangent. This is a continuous function whose output ranges from minus one for significantly negative inputs to plus one for significantly positive inputs. Perceptrons can be very useful when combined into structures called neural networks, such as the one illustrated here. To describe this, I need to present a few new terms. Each perceptron in a neural network is called a node, and each column of nodes is called a layer. A layer is dense if every node is connected to every node in the next layer. This network has four layers, starting with the input layer and ending with the output layer. The layers in between are called hidden layers and a neural network with hidden layers is said to be deep. You'll see the acronym DNN frequently in machine learning and this stands for deep neural network. DNNs have proven effective in many applications. Two famous examples include Google's AlphaGo program, which uses a DNN to beat professional Go players, and Google's 2012 application that used a DNN to recognize cat videos. This video has introduced neural networks by explaining their basis in neurons and perceptrons. As we'll see, these networks make it possible to perform sophisticated machine learning tasks.

### **Training neural networks**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] To understand neural networks, it helps to view them mathematically. Every neural network embodies an equation that relates outputs to inputs. For example, the network in this slide accepts four inputs, feeds them into three nodes and produces one output. The output can be expressed as a function, determined by the network's weights, inputs, and activation functions. This serves as our model of how the output is determined by the input. Many aspects of the model are selected in advance, such as the network structure and its activation functions. The inputs are determined by the system, so the only way to improve the model is to change the weights. The process of finding the best weights is called training. The most popular method of training a neural network is backpropagation. This requires five steps that are repeated several times. The first step computes the network's output for a set of inputs and weights. Then it finds the difference between this output and the actual system output. This difference is called loss. To determine how the weights need to be changed, backpropagation finds the partial derivative of the loss with respect to each weight. Using these partial derivatives, it updates the weights to reduce the loss. Then it computes a new output value and repeats the process. The simplest method of updating weights is the gradient descent algorithm. As discussed earlier, a gradient is the vector that points in the direction of greatest steepness. Backpropagation computes the gradient of the loss function by taking partial derivatives with respect to the weights. To clarify how the gradient relates to partial derivatives, I've illustrated the function f of xy equals 16 minus four x squared, minus two y squared. At the point one, two, both partial derivatives equal minus eight. So the gradient of F at one, two is a vector with components minus eight and minus eight. Our goal is to reduce the loss. So we update weights using the negative gradient. If you think of the loss function as a hill, then following the negative gradient is the fastest path downward. Each training step moves further down the hill. And when the loss reaches a minimum, training is complete. This slide presents a more mathematical view of the training process. The height step changes Wi to Wi + 1. And it does this by setting Wi + 1 = Wi minus the gradient of the loss, denoted grad L times the constant called the learning rate denoted Ada. The image on the left shows how this works. The gradient at point P is positive. So Wi + 1 will be smaller than Wi. As a result, the loss decreases. The learning rate is a major concern in the training process. If it's too high, the algorithm will take large steps down the hill, but it may step around the minimum. If it's too small, the algorithm will step much more precisely, but it will take more time to reach the bottom. If trading brings the loss down to point Q, training will stop because the gradient is zero. At this point, loss has reached a minimum, but there's a problem, point R is even lower than point Q. Which means Q is a local minimum, not a global minimum. The distinction between local minima and global minima is a source of constant frustration. Neural networks have been researched for several decades, but training them is still more or less an art than a science. This video has presented the basics of training with backpropagation.

### **Creating neural networks in OpenCV**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In OpenCV, neural networks are represented by instances of the ANN\_MLP class, where ANN stands for Artificial Neural Network, and MLP stands for Multi-Layer Perceptron. Using this class is easy. You create an instance by calling ANN\_MLP\_create and then set its structure by calling setLayerSizes with a list that identifies the number of nodes in each layer. After creating the network, you can train it by calling train with two arrays, one containing feature data, and one containing labels. A label is an integer that identifies the category of the corresponding feature. When training is complete, the predict method accepts data of an unknown feature and returns a prediction or classification. The ANN\_MLP class provides three methods that control the training process. The first is setTrainingMethod, which can be set to simulate at annealing, backpropagation, or resilient backpropagation. The second is setActivationFunction, which assigns the activation function for every node in the network. This can be set to ReLU, Leaky ReLU, Gaussian, and symmetric sigmoid. The last training method is setTermCriteria, which identifies when training should stop. This has two fields and the first, MAX\_ITER, sets the maximum number of training steps. The second, EPS, sets the minimum absolute change in loss from one step to the next. In the following video, I'll demonstrate how to code a neural network that classifies irises. Specifically, we want the network to tell us whether an Iris belongs to the setosa, versicolor, or virginica species. We'll provide the network with four observations. The Iris is sepal width, sepal length, petal width, and petal length. We don't have a clear mathematical relationship between these variables and the irises species, so we'll train a neural network and use it for classification. This slide illustrates the structure of the network we're going to use. On the left, we provide the network with an irises physical characteristics. This information passes through two hidden layers, and the output layer provides the probabilities that the Iris belongs to each species. The input data is contained in a file called Iris.CSV, which has 150 rows of data. The first 140 rows will be used for training. And the last 10 will be used for testing. This slide shows what the first six rows look like. The first four columns identify the irises sepal length, sepal width, petal length, and petal width in centimeters. The last column contains an integer that identifies the irises species. A zero corresponds to setosa, a one corresponds to versicolor, and a two corresponds to virginica. This video has explained how to create neural networks in OpenCV. The next video will demonstrate how to code an application that uses a neural network to classify irises.

### **Classifying irises with a neural network**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor/Educator]To start neural network development. I'll open ('iris.csv') as cvsfile. I'll set reader = csv.reader, which accepts the csvfile. Now I'll iterate through the rows. I want to use the first 140 rows for training and the last ten for testing. If I is less than 140, I'll add the first four values of each row to train features and the last value to train labels And I'll do the same thing for the test data. Now I'll create the neural network by setting net = cv2.ml.ANN\_MPL\_create{}. To set the network structure, I'll call net.setLayerSizes and provide an array containing the layer sizes, which I'll set to four, five, five, and three. Now I'll set the training method by calling net.setTrainMethod[CV2.ml.ANN\_MLP\_BACKPROP] I'll set the activation function by calling net.setActivationFunction[cv2.ml.ANN\_MLP\_SIGMOID\_SYM] To train the network, I'll call the train method. This accepts the train features and the train labels. Next I'll call the network's predict method to classify the test features. To check these results. I'll use numpy to get the mean square error. Now I'll print the error. Ah, I mistyped a function. And when I run the application, I get an error of approximately 0.0057. This video has demonstrated how irises can be classified with a neural network. The accuracy is good, but could be improved with a different structure or training criteria.

## **Question 1 of 4**

In a neural network, what is a column of nodes called?

* block  
  Incorrect  
  A column of nodes in a neural network is not called a block.
* layer  
  Correct  
  A column of nodes in a neural network is called a layer.
* region
* div

## **Question 2 of 4**

Which OpenCV class represents a neural network?

* ANN\_NET
* CvNetwork
* NeuralNet
* ANN\_MLP  
  Correct  
  OpenCV's ANN\_MLP class represents a neural network.

## **Question 3 of 4**

What are the elements of a neural network called?

* points
* nodes  
  Correct  
  The elements of a neural network are called nodes.
* scalars
* pixels

## **Question 4 of 4**

What's the most popular method of training a neural network?

* minimization
* normalization
* backpropagation  
  Correct  
  The most popular method of training a neural network is backpropagation.
* standardization

### **Introducing convolutional neural networks (CNNs)**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video, I'll introduce a new type of neural network called a convolutional neural network, or CNN. CNNs are used extensively in machine learning, and we're going to use them to recognize objects. In earlier videos, I presented a neural network that returned a set of probabilities. CNNs accomplish essentially the same result. If a CNN is trained to recognize n objects, it will return n probabilities. If a probability is close to one, it means the CNN recognizes the corresponding object. The main difference between CNNs and regular neural networks is that CNNs accept images as input and process them using convolution. The values in the convolution kernels serve as the network's weights. Another difference between CNNs and regular neural networks involves training. When training a CNN, the goal is to determine which kernels are best suited for recognizing the objects of interest. This can be hard to grasp, so I'll compare CNN training to regular convolution. In regular convolution, we take an image and a kernel and produce a new image. When training a CNN, we provide a set of images and a set of labels that identify which objects are present. The training process produces kernels that will enable the CNN to recognize these objects. CNNs contain different types of layers than regular neural networks, and the most important new layer is the convolution layer. Each node of a convolution layer accepts an image and convolves it with a different kernel. As a result, each node produces a new image. Convolution layers produce a lot of data, so these layers are commonly followed with pooling layers. Pooling splits an image into blocks and converts each block into a single pixel. The first pooling method is maximum pooling, which sets the output pixel equal to the largest pixel in each block. The second method is average pooling, which sets the output pixel equal to the average of the pixels in each block. This slide illustrates the structure of a basic CNN. The first layer convolves the image with three kernels and produces three images. That's a lot of data, so it's followed by a pooling layer that reduces the size of each image. The pooled images pass into another convolution layer. Now, there are four kernels, so the layer produces four images. These pass through another pooling layer. The next layer is called the flatten layer because it combines the images into a one-dimensional array. This serves as the input of the final layer, which is a dense layer that contains one node for each probability. So far, all of the code I've presented in this course has relied on OpenCV running on the Raspberry Pi, but now we reach a problem. OpenCV doesn't have functions for creating or training CNNs. Also, training a CNN requires a lot of processing, so using the Raspberry Pi isn't a good idea. For this reason, the next video will explain how to train CNNs on a PC using a package called Keras. This is part of a larger package called TensorFlow. Once the CNN is trained, the model can be saved to a file and copied to a Raspberry Pi. Then, we can execute the model using the stripped down version of TensorFlow called TensorFlow Lite. This course performs object recognition using convolutional neural networks, or CNNs. This video has explained what CNNs accomplished and how they're structured.

### **Creating CNNs with Keras**

Selecting transcript lines in this section will navigate to timestamp in the video

- In this video, we'll move away from OpenCV and look at a package called Keras. Unlike OpenCv, Keras focuses solely on neural networks and it provides a simple API for creating training and running many different types of neural networks. Because of its simplicity, Google hired Keras's creator and incorporated Keras into its TensorFlow package. Today, Keras is the default neural network API in TensorFlow, our usage of Keras will consist of four steps, the first is to create an instance of the Sequential class and to add layers that represent the layers of a neural network. The compile method defines the training process and the fit method, trains the network. Instead of executing the model, we're going to convert it to the TensorFlow Lite format. Then we'll save the model to a file that can be copied to the Raspberry Pi. Each neural network layer corresponds to a Keras class. The two convolution layers are represented by instances of Keras Conv2D class. And the two MaxPooling layers are represented by instances of the MaxPooling2D class. The Flatten layer is represented by an instance of the Flatten class. And the final layer is represented by an instance of the Dense class. This slide shows how Keras classes are used in code. The first line creates an instance of the Sequential class. This represents a neural network made up of networks and sequence. Each line afterward adds a new layer by calling model to add with a constructor for the appropriate layer, this code adds four layers to the model, it creates a convolutional layer by calling the conv2D constructor. A MaxPooling layer by calling the MaxPooling2D constructor. A Flatten layer by calling the Flatten constructor and a Dense layer by calling the Dense constructor. Keras provides many more layers than those shown here. You can create sophisticated neural networks by calling model.ad with the right constructors. For this course, the goal of our CNN is to recognize toy cars, to be specific. We want to know which of these three cars is present in an image, or if none of them are present. The CNN provides this information with four values. The first value is the probability that none of the three cars are present. The second, is the probability that the black car is present. The third, is the probability that the blue car is present. And the last is the probability that the green car is present. To train the CNN, we'll use a total of 400 images with 100 images for each class. The left part of this slide shows what 16 of the training images look like. Unlike the training images for object detection, these images present the objects in different sizes and orientations. Each image has the same size, which is 256-by-256 pixels. Also each file name starts with the corresponding label. For example, image files starting with two all contain blue cars and image files starting with zero don't contain any cars at all. This video has introduced the Keras package and its classes, it is also presented the method we'll use to train a CNN to recognize toy cars.

### **Training CNNs with TensorFlow**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Training a CNN requires a lot of processing, so this video explains how to train CNNs on a PC. OpenCV doesn't have functions for creating and training CNNs, so I'll install TensorFlow with the command pip3 install tensorflow. This can take some time, but in my case, the package is already present so I'll launch thonny. To begin, I'll set the number of classes that the network should recognize. There's one class for each toy car and one class for no cars. Next, I'll load images and labels by calling the load\_data function. This accepts the directory containing training files and the number of classes. To create the neural network, I'll set model equal to models.Sequential. For the first convolution layer, I'll call model.add with layers.Conv2D. I'll set the number of kernels to 16, the size of each kernel to 3,3, the activation function to relu, and the input\_shape to the shape of the first image. For the first pooling layer, I'll call model.add with layers.MaxPooling2D and I'll set the block\_size to 2,2. Then I'll create another convolution layer and another pooling layer. To flatten the pooled images, I'll call model.add with layers.Flatten. For the final layer, I'll call model.add with layers.Dense. I'll set the number of nodes to the number of classes. And because the network performs classification, I'll set the activation function to softmax. Now I'll compile the network by calling model.compile. There are a number of classes, so I'll set loss to categorical \_crossentropy and metrics to accuracy. To train the model, I'll call model.fit. This accepts an array of images, an array of labels, and the number of epochs. An epoch represents a complete pass through the input data. And for this example, 10 should be sufficient. I'll also set verbose equal to zero. After training the model, I'll save it to a file that can be accessed by TensorFlow Lite. To do this, I'll set converter equal to tf.lite.TFLiteConverter.from\_keras\_model(model). Now I'll open a file named object\_recognition.tflite and write the converted result. Now I'll train the CNN. When training is finished, I'll check that the model file is present by importing os and running os.listdir. Here, you can see that the model file is present in the directory. Before proceeding to the next video, I recommend that you copy this to the Raspberry Pi. This video has shown how to create and train a CNN using Keras. I've also shown how to save the CNN to a file that can be accessed by TensorFlow Lite.

### **Executing models with TensorFlow Lite**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] An earlier video demonstrated how to train a CNN, and save it to a file that can be accessed by TensorFlow Lite. The goal of this video is to provide a proper introduction to TensorFlow Lite. Once you understand this, you'll be able to execute CNNs on a Raspberry Pi. The complete TensorFlow package is very large. So Google provides a stripped down version that only occupies about one megabyte of memory. This is ideal for smartphones, microcontrollers, and single-board computers like the Raspberry Pi. TensorFlow Lite doesn't have the resources to create or train neural networks, but it can execute models at high speed. To see how this works, you need to become familiar with the interpreter class. An interpreter takes care of loading the model, receiving input data and executing the model. In this course, we'll access the interpreter in a five-step process. The first step creates an interpreter instance by calling the constructor with the name of the model file. The next step reserves memory by calling allocate tensors. Next set sensor provides the interpreter with input data. In TensorFlow, tensors are similar to NumPy arrays. We'll assign the second argument of set tensor to a NumPy array containing pixels. Once the input data is available, the invoke method tells the interpreter to process the input by executing the model. Then the model's output can be obtained by calling get tensor. Every interpreter has two lists of dictionaries that provide information about its tensors. For input tensors, the list can be obtained by calling, get input details. For output tensors, the list can be obtained by calling, get output details. These methods are important because they enable us to access the index values required by get tensor and set tensor. The code in this slide shows how this works. To get the index of the first input tensor. You need to call, get input details, access the first element, and obtain the value corresponding to the index key. The process is similar for obtaining output index values. This video has discussed that TensorFlow Lite package, and its interpreter class. The following video, we'll use TensorFlow Lite on the Raspberry Pi to recognize objects.

## **Question 1 of 5**

What type of layer reduces the amount of data in a convolutional neural network (CNN)?

* recurrent
* convolution
* pooling  
  Correct  
  A pooling layer reduces the amount of data in a convolutional neural network (CNN).
* dense

## **Question 2 of 5**

What is the central class of the TensorFlow Lite package?

* Interpreter  
  Correct  
  The Interpreter class is the central class of the TensorFlow Lite package.
* Monitor
* Supervisor
* Communicator

## **Question 3 of 5**

In machine learning, what is the value that identifies a classification category?

* qualifier
* marker
* modifier
* label  
  Correct  
  In machine learning, labels are used to identify classification categories.

## **Question 4 of 5**

Which of the following is not a Keras class that represents a network layer?

* MaxPooling2D
* Flatten  
  Incorrect  
  Flatten is a Keras class that represents a flattening layer.
* Activation2D  
  Correct  
  Activation2D is not a Keras class that represents a network layer.
* Conv2D

## **Question 5 of 5**

What type of layer is commonly used as the last layer of a convolutional neural network (CNN)?

* convolution  
  Incorrect  
  The last layer of a convolutional neural network (CNN) usually isn't a convolution layer.
* dense  
  Correct  
  The last layer of a convolutional neural network (CNN) is commonly a dense layer.
* dropout
* pooling  
  Incorrect  
  The last layer of a convolutional neural network (CNN) usually isn't a pooling layer.

### **Introducing the picamera package**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] This video introduces the Pi Camera package, which makes it possible to capture images from a camera. The Raspberry Pi supports a few different cameras. And on my system, I'm using the Raspberry Pi high-quality or HQ camera. The left image shows what the board looks like with the HQ camera module connected. No matter what camera you use, it's important to attach a good lens. This focuses the light received by the sensor and without it, your images will be blurry. The image on the right shows the lens that I'm using. This is a 16 millimeter telephoto lens that connects to the camera using a C mount. To access the camera in Python, you need to install the Pi Camera package. This provides the PiCamera class, which plays a central role in this discussion. In this course, we'll access the PiCamera class using five steps. First, we'll call the constructor to create a PiCamera instance. Then we'll configure the camera's operation by setting the instance's properties. After configuring the camera, we'll call the capture method, which reads a single image. We'll pass the image to a machine learning model for analysis. When analysis is complete, we'll draw the results on the image and then save it to a file. Cameras have many settings that determine how images are taken, and we can configure them by setting properties of the PiCamera instance. The resolution property determines the dimensions of the display. On my system, the default is 1920 by 1080, but I get better results setting this to a lower value. Brightness makes the image's pixels lighter or darker. This can be set to a value between 0 and 100 and the default is 50. I usually set it higher. Contrast affects differences in the image's colors, and it can be set to a value between minus 100 and 100. The default is zero, but I like to set it a little higher. ISO determines how sensitive a camera's sensor is to light, and it can be set to a value between 0 and 1600. I usually set it to the maximum value of 1600. After the camera is configured, the capture method can be called to read images. If it's called with a filename, the image will be saved to the file according to the format given by the file suffix. It can also be called with a stream and an argument that identifies the image's format. The Pi Camera package provides custom stream classes for accessing data. And for this discussion, the most important is picamera.module.PiRGBArray. This provides pixels in a NumPy array. And if the format is set to bgr, the array can be accessed by OpenCV. This video has introduced the Pi Camera package and the all-important PiCamera class. Later videos will use this class to access and analyze camera images.

### **Accessing a Raspberry Pi camera in Python**

Selecting transcript lines in this section will navigate to timestamp in the video

- Before you can access the camera, you need to configure the Raspberry Pi to read data from the camera's connection. To do this, open the leftmost menu in the desktop and go to preferences. Then Raspberry Pi configuration. If you open the interfaces tab, the first interface is for the camera. Make sure this is set to enable and then click okay. It's also a good idea to make sure that the pi camera and pi camera dot array packages are installed. To check, open a terminal and enter sudo pip3 install pi camera array. These packages are already installed on my system. If they're not installed on your system, that command will install them. To start coding, I'll create an instance of the PI camera class by setting camera equal to PI camera dot PI camera. The default resolution is 1920 by 1080, but I get better results by setting it to 640 by 480. I'll configure this by setting camera dot resolution, to 640 480. The default brightness is 50, but I prefer to increase it to 60. So I'll set camera dot brightness to 60. The default contrast is zero, but I get better results by increasing it to 20. So I'll set camera dot contrast to 20. Lastly, my camera images usually come out somewhat dark, so I prefer to set the ISO value to the maximum. I'll set camera dot ISO to 1600. Now I'll create a stream that provides image data in a pi array. Stream equals pi camera dot array dot PI RGB array. And I'll pass the camera. To read 10 images from the camera, I'll create a four loop. Now I'll read an image from the camera by calling the camera dot capture method. This accepts the stream and the format. For open CV compatibility, I'll set format equal to BGR. To read an image from the stream, I'll set image equal to stream dot array. Now I'll save the array to a file by calling CV 2 dot I M write. This accepts a name for the file, and the image. Before another image can be captured, the streams position must be reset. Now I'll tell the camera to wait half a second before capturing the next image. Now I'll run the application. Now that it's finished, it's produced 10 JPEGs in the working directory. The PI camera package provides the PI camera class, which makes it possible to capture images from a camera. This video has explained how PI camera can be used and has demonstrated how image capture works.

### **Object detection with a Raspberry Pi camera**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] This video demonstrates how the Raspberry Pi can be programmed to detect objects in camera images. Before I start coding, I'd like to review how object detection works. The application looks at one window of an image at a time, each window is split into the cells, and we use a HOGDescriptor to compute the histogram of oriented gradients for each cell. These histograms are concatenated to form the windows feature descriptor. Next, a support vector machine or SVM checks if the windows feature descriptor matches that of the toy car. If there's a match, the object is detected and the application draws a rectangle around the window. To obtain the HOGDescriptor, all set hog equal to the get descriptor function. This creates the same HOGDescriptor that we used earlier for object detection. To analyze images, the detector needs a trained SVM. In an earlier video, we saved the SVM to svm.xml. So I'll set svm equal to cv2 .ml .SVM\_load, and svm.xml. To extract data from the SVM, I'll set vec equal to svm.getSupportVectors. Now I'll get the first value of the SVM decision function. To configure the descriptor with the SVM data, all call hog.setSVMDetector and append vec and minus rho. Next, I'll set camera and stream equal to get\_camera\_stream. This returns the same Pi camera and Pi RGB array that we created in the proceeding video. I want to capture 10 images, so I'll create a for loop. To capture an image, I'll call camera.capture. This accepts the image stream and the format, which all set to BGR. To prepare an image for analysis, I'll set test\_image equal to preprocess, which accepts the image array and the desired size, which I'll set to 213, and 160. Now I'll get the object detection result by setting res equal to hog.detectMultiScale and I'll pass in the image. Res consists of two lists. The first contains rectangles and the second contains confidence values. I'm only interested in rectangles, if the confidence value is greater than 1.2. And I'll check this with an if statement. If this condition is met, I'll draw a rectangle on the image. To get the index of greatest confidence, I'll set index equal to np.argmax. Then I'll obtain the location of the rectangle. Next, I'll draw the rectangle by calling cv2.rectangle. This accepts the image, the lower left corner, which all set to x, y, the upper right corner, which all set to x + w, y + h, the color, which all set to blue, and the line width, which all set 2. To save the image, I'll call cv2.imwrite. This accepts the name of the file and the image. Now I'll reset the stream by calling stream.seek and zero. And I'll delay processing by half a second. Now I'll run the application. When the application is finished, it'll produce 10 different images that will contain rectangles if the object was detected. In my tests, detecting objects in the camera images works about 73% of the time. More tests are needed, but it looks like the problems involved the camera and the lens. Despite changing the lens setting, I could never get the focus as clear as with a regular camera. And even with the ISO value set to the maximum, the images are still very dark. Object detection can be performed on the Raspberry Pi by combining the Pi camera package and OpenCV. This video has explained the detection process and has demonstrated how it can be performed.

### **Object recognition with a Raspberry Pi camera**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] This video shows how the Raspberry Pi can recognize objects in camera images. The goal is to obtain four probabilities that identify recognition states. No cars, black car, blue car, and green car. Each image will be analyzed by the convolutional neural network that we save to the object\_recognition.tflite file. Because of the Raspberry Pi has limited resources, we'll rely on TensorFlow Lite and the interpreter class. To start coding I'll set ip equal to the tflite .interpreter constructor. I'll set the model path argument to object\_recognition .tflite. Next, I'll reserve memory by calling ip.allocate\_tensors. To access input and output tensors, I'll need their index values. I'll get the index value of the input tensor by calling get\_input\_details. I'll access the first element and the value corresponding to the index key. Then I'll do the same thing for the output index. Next, I'll set camera and stream equal to get\_camera\_stream. This configures the camera with the same settings as those we discussed earlier. As before, I'll analyze 10 images by creating a for-loop. To read an image, I'll call camera.capture, and pass in the stream. I'll set the format equal to JPEG. Next I'll reset the stream by calling stream.seek with zero. TensorFlow Lite works very well with the Python Imaging Library. So I'll set img equal to Image.open and pass in the stream, then I'll crop the central square. The recognition model requires input data to be provided as 256 by 256 images. So I'll set img equal to img.resize and I'll pass in 256 and 256. To prepare the input tensor, I'll set tensor equal to preprocess and img. This converts the image to a NumPy array, subtracts the mean, and divides the standard deviation. Now I'll send the tensor to the interpreter by calling ip.set\_tensor. This accepts the index of the input tensor, and the tensor. Now I'll analyze the image by calling ip.invoke. To access the result, I'll set pred equal to get\_tensor With the output index. Pred contains four probabilities corresponding to the different recognition states. To draw the largest probability on the image, I'll set draw equal to ImageDraw.Draw and pass in the image. Now I'll create a string named msg. This displays the classification with the highest probability, and the probability. I'll draw this text on the image by calling draw.text with the coordinates 10, 10, and the message. Now I'll save the image to a file by calling img.save, which accepts the name of the file. Lastly, I'll add a half second delay before capturing more images. Now I'll run the application. Ah. Ha. Now that the application is finished, it's produced 10 images. Each has a message that identifies the predicted category and the computer probability. This slide gives an idea of the results from my system. These four images were all classified successfully. But overall, the application only worked about 79% of the time. The black car, class one, was the easiest to classify, and the blue car, class two, was the hardest. The Raspberry Pi can recognize objects in an image by combining Pi camera, TensorFlow Lite, and the Python Imaging Library. This video has explained the recognition process and has demonstrated how it can be performed in code.

## **Question 1 of 4**

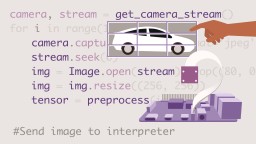
Which method of the HOGDescriptor class produces the object detection result?

* analyzeImage
* readScale
* detectMultiScale  
  Correct  
  The detectMultiClass method of the HOGDescriptor class produces the object detection result.
* findObject

## **Question 2 of 4**

Which method of the Interpreter class reserves memory for data?

* apportion\_images
* designate\_lists  
  Incorrect  
  The Interpreter class doesn't have a designate\_lists method.
* allocate\_tensors
* reserve\_arrays  
  Incorrect  
  The Interpreter class doesn't have a reserve\_arrays method.



Replay

Review this video

Object recognition with a Raspberry Pi camera

5m 15s

## **Question 3 of 4**

Which camera parameter determines the sensitivity of the image sensor?

* contrast
* ISO  
  Correct  
  A camera's ISO parameter determines the sensitivity of its image sensor.
* resolution  
  Incorrect  
  Resolution affects the number of pixels in an image, not the sensitivity of the camera's sensor.
* intensity

## **Question 4 of 4**

What method of the Picamera class is used to read one image at a time?

* readImage  
  Incorrect  
  The PiCamera class doesn't have a readImage method.
* capture  
  Correct  
  The capture method of the Picamera class is used to read one image at a time.
* takePicture
* getFrame

### **Next steps**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Ten Years ago, the idea of performing real-world machine learning on a single board computer was unheard of. But the Raspberry Pi is so powerful that applications can capture images from a camera, compute histograms of oriented gradients, and then execute convolutional neural networks. This course has explained how to use these capabilities to detect and recognize objects. I've presented a few different methods of machine learning. But the subject is far more extensive than what I've described here. New research papers are released every day. And if you're interested in pursuing object detection and recognition, I strongly recommend that you look into advanced methods. Like the Scale-invariant feature transform or SIFT. And You only look once, or YOLO detection. You can find thousands of machine learning papers for free by looking through archive.org. You can also learn more about the Open CV Library at opencv.org. And you can learn more about TensorFlow at tensorflow.org. TensorFlow provides a vast number of features and it grows more powerful with each new version. Most important, be sure to experiment with your own machine learning code, try new neural network structures, new training methods and new image processing algorithms. As your experience grows, your applications will become more accurate and more precise.