- [Instructor] So far, we've created the neural network with densely connected layers. Now we're ready to add convolutional layers to make it better at finding patterns in images. Let's open up 05\_convolutional\_layers.py. To be able to recognize images officially, we'll add convolutional layers before our densely connected layers.Convolutional layers are able to look for patterns in an image, no matter where the pattern appears in the image. Let's go down to line 22, this is where we'll insert a convolutional layer. First, to add the layer, we'll call model.add.

Now there's two types of convolutional layers: 1D and 2D. Since we're working with images, we'll want to add the two dimensional convolutional layer. For some kinds of data, like sound waves, you can use one dimensional convolutional layers, but typically you'll be working with 2D layers. To create one, we just create a new Conv2D object and then pass in the parameters. The first parameter is how many different filters should be in the layer? Each filter will be capable of detecting one pattern in the image.

We'll start with 32. Next, we need to pass in the size of the window that we'll use when creating image tiles from each image. Let's use a window size of three pixels by three pixels. So to do that, we pass in an array of three comma three. This will split up the original image into three by three tiles. When we do that, we have to decide what to do with the edges of the image. If the image size isn't exactly divisible by three, we'll have a few extra pixels left over on the edge. We can either throw that information away, or we can add padding to the image.

Padding is just extra zeros added to the edge of the image to make the math work out. The terminology that Keras uses here is a bit confusing. If we want to add extra padding to the image, it's called same padding. There's complex historical reasons why researchers used the term same, but it's easier just to memorize it. For this layer, we do want to have padding, so we'll pass in a parameter padding equals, and the string same, and just like the normal dense layer, convolutional layers also need an activation function. And just like dense layers, we almost always use the relu activation function because of its efficiency.

So I'll pass in activation equals relu. And that's it for adding this layer, but there's one more tweak we need to make. Let's look at the next line. This dense layer is no longer the first layer in the neural network, so it shouldn't have an input shape defined anymore. Let's just cut and paste this input shape, and move it up to the convolutional layer because it's now the first layer. To make our neural network more powerful, let's add a few more convolutional layers the same way.

First, let's add another one with the same settings, 32 filters and a three by three window size. So we'll say model.add, we'll pass in Conv2D, we'll say 32 filters, and the three by three window size, and we'll also add an activation function, we'll use relu again, activation equals relu. Now in this layer we won't have the image, so we don't need to pass in the padding parameter. Now let's add two more layers with 64 filters each.

First we'll add one with padding, so we'll say model.add Conv2D, say 64 filters, I'll use a three by three tile size again. I'll pass in padding equals same, and in activation function we'll use relu. And now we'll do one more without padding, but also with 64 filters. So we can just cut and paste this, paste it here, and just remove the padding.

Alright, there's just one thing left to do. Whenever we transition between convolutional layers and dense layers, we need to tell Keras that we're no longer working with 2D data. To do that we need to create a flattened layer and add it to our network. We can do that by calling model.add, and creating a new flattened layer, and there's no parameters required for a flattened layer. Alright, if you look down at line 35, we can see that we're printing out the summary of the neural network structure, so let's run this code and see what it looks like.

Right click and choose run. Alright, we can see the neural network now has seven layers. We have four convolutional layers, the flattened layer, and then our two dense layers. Notice that each layer also has a number of parameters listed. This is the total number of weights in that layer. There's also a total number at the bottom for the whole network. As we add more layers that total number will keep increasing. This is the size or complexity of our neural network. The larger the number, the longer it'll take to train and the more data we'll need to train it.

It's a good idea to keep an eye on this number as you add layers to your neural network. As you test and refine your neural network, you might find that you can get good results even after you remove some of your layers and reduce this number. When you can do that, that means you'll need less powerful hardware to run your neural network, so it's always a good goal.