Words	Visuals
In the previous example you saw how you could create a neural network -- called a 'Deep Neural Network' to pattern match a set of images of fashion items to labels. In just a couple of minutes you were able to train it to classify with about a 72% accuracy on the training set, and a little less on the test set.	Talking head
Now one of the things you would have seen when you looked at the images is that there's a lot of wasted space in each image -- and while they're only 784 pixels, it would be interesting to see if there was a way that we could condense the image down to the important features in it that distinguish what makes it a shoe or a handbag	Taking head
And that's where Convolutions come in. So what's a convolution? Well -- if you've ever done any kind of image processing it usually involves having a filter and passing that filter over the image in order to change the underlying image. The process works a little bit like this. For every pixel, take its value, and the value of its neighbors. If our filter is 3x3, then we take each immediate neighbor so that we have a corresponding 3x3. Then, to get the new value for the pixel we simply multiply each neighbor by the corresponding value in the filter. So, for example, in this case our pixel has value 192, and its upper left neighbor has the value 0. The upper left value in the filter is -1, so we would multiply 0 by -1. Then we would do the same for the upper neighbor. It's value is 64, and the corresponding filter value is 0. So we would multiply them out. Repeating for each neighbor and each corresponding filter value, we would then have the new pixel be the sum of each of the neighbor values multiplied by their corresponding filter value. And that's a convolution. It's really as simple as that!	SLIDE 1
The idea here is that some convolutions will change the image in such a way that certain features in the image get emphasised. So, for example, if you look at this filter, then the vertical lines in the image pop out.	SLIDE 2
And with this filter, the horizontal lines pop out.	SLIDE 3
That's a very basic introduction to what convolutions do. And when they're combined with something called pooling, they become really powerful. Put simply, pooling is a way of compressing an image. A quick and easy way to do it is to go over the image 4 pixels at a time -- i.e. the current pixel, and it's neighbors underneath and to the right of it. Pick the biggest value, and keep that.	Talking head
For example, you can see it here. My 16 pixels on the left are turned into the 4 pixels on the right by looking at them in 2x2 grids and picking the biggest value. This will preserve the features that were highlighted by the convolution while simultaneously quartering the size of the image -- we halve the horizontal and vertical axis.	SLIDE 4
So now let's take a look at Convolutions and Pooling in Code. We don't have to do all the math for filtering and compressing -- we simply define Convolutional and Pooling layers to do the job for us.	Talking Head
So here's our code from the earlier example -- where we defined out neural network to have an input layer in the shape of our data, an output layer in the shape of the number of categories we're trying to define, and a hidden layer in the	Slide 5

middle. The Flatten takes our square 28x28 images and turns them into a 1 dimensional array.	
To add convolutions to it, you use code like this. You'll see that the last 3 lines are the same -- the flatten, the dense hidden layer with 128 neurons and the dense output layer with 10 neurons. What's different is what has been added on top of this. Let's take a look at this line by line.	Slide 6
Here we are specifying the first convolution. We're asking Keras to generate 64 filters for us. These filters are 3x3. Their activation is 'relu', which means that negative values will be thrown away, and finally the input shape is as before 28x28. The '1' just means that we're using a single byte for color depth. As we saw before the images are greyscale. Now of course you might wonder what the 64 filters are. It's a little beyond the scope of this class to define them, but they aren't random -- they start with a set of known, good, filters and the ones that work from that set are 'learned' over time. In a similar way to the pattern fitting you saw earlier, the filters taht give good results are learned over time.	Slide 7
For more details on Convolutions and how they work, there's a great set of resources here	Slide 8 - link to deeplearning.ai playlist
This next line of code then creates a pooling layer. It's MAX pooling because we are going to take the maximum value. We're saying it's a 2x2 pool, so, for every 4 pixels, the biggest one will survive, as shown earlier.	Slide 9
We then add another convolutional layer, and another max pooling layer, so that the network can learn another set of convolutions on top of the existing one, and again pool to reduce the size. So, by the time the image gets to the flatten to go into the dense layers, it's already much smaller -- it's been quartered, and then quartered again, and its content is greatly simplified -- the goal being that the convolutions will filter it to features that determine the output.	Slide 10
A really useful method on the model is the model.summary() method. This allows you to inspect the layers of the model, and see the journey of the image through the convlutions	Slide 11
And here is the output -- it's a nice table showing us the layers, and some details about them, including the output shape.	Slide 12
It's important to keep an eye on the output shape column. When you first look at this it can be a bit confusing and feel like a bug. After all, isn't the data 28x28, so why is the output 26x26?	Slide 13
The key to this is remembering that the filter is a 3x3 filter. Consider what happens when you start scanning thorugh an image, starting on the top left. So for example, with this image of the dog on the right, you can see zoomed into the pixels at its top left corner	Slide 14
You can't calculate the filter for the pixel in the top left, because it doesn't have any neighbors above it or to its left.	Slide 15
In a similar fashion, the next pixel to the right won't work either, because it doesn't have any neighbors above it	Slide 16
So logically, the first pixel you can do calculations on is this one...	Slide 17
Because it has all 8 neighbors that a 3x3 filter needs. This, when you think about it,	Slide 18

you can't use a 1 pixel margin all around the image, so the output of the convolution will be 2 pixels smaller on x, and 2 pixels smaller on y. If your filter is 5x5, for similar reasons, your output will be 4 smaller on x, and 4 smaller on y	
So that's why our output from the 28x28 image is now 26x26.	Slide 19
Next is the first of the Max Pooling layers. Remember that we specified it to be 2x2, thus turning 4 pixels into 1, and halving our X and Y dimensions, so our output gets reduced to 13x13	Slide 20
The convolutions then operate on that, and of course we lose the 1 pixel margin as before, so we're down to 11x11	Slide 21
And another 2x2 Max Pooling halves this, rounding down, so we're down to 5x5 images.	Slide 22
So now our dense neural network is the same as before, but it is being fed with 5x5 images instead of 28x28 ones. But, remember that it's not just 1 compressed 5x5 image instead of the original 28x28...there are a number of convolutions per image that we specified -- in this case 64. So ther are 64 'new' images of 5x5 being fed in. Flatten that out and you have 25 pixels times 64, which is 1600. So the new flattened layer has 1600 elements in it, as opposed to 784 that you had previously. This number is impacted by the parameters that you set when defining the conv2d layer. Later when you experiment, see what the impact of setting other values for the number of convolutions will have -- and in particular see what happens when you're feeding less than 784 overall pixels in. Training should be faster, but is there a sweet spot where it's more accurate?	Slide 23
Let's switch to the workbook, and try it for ourselves!	