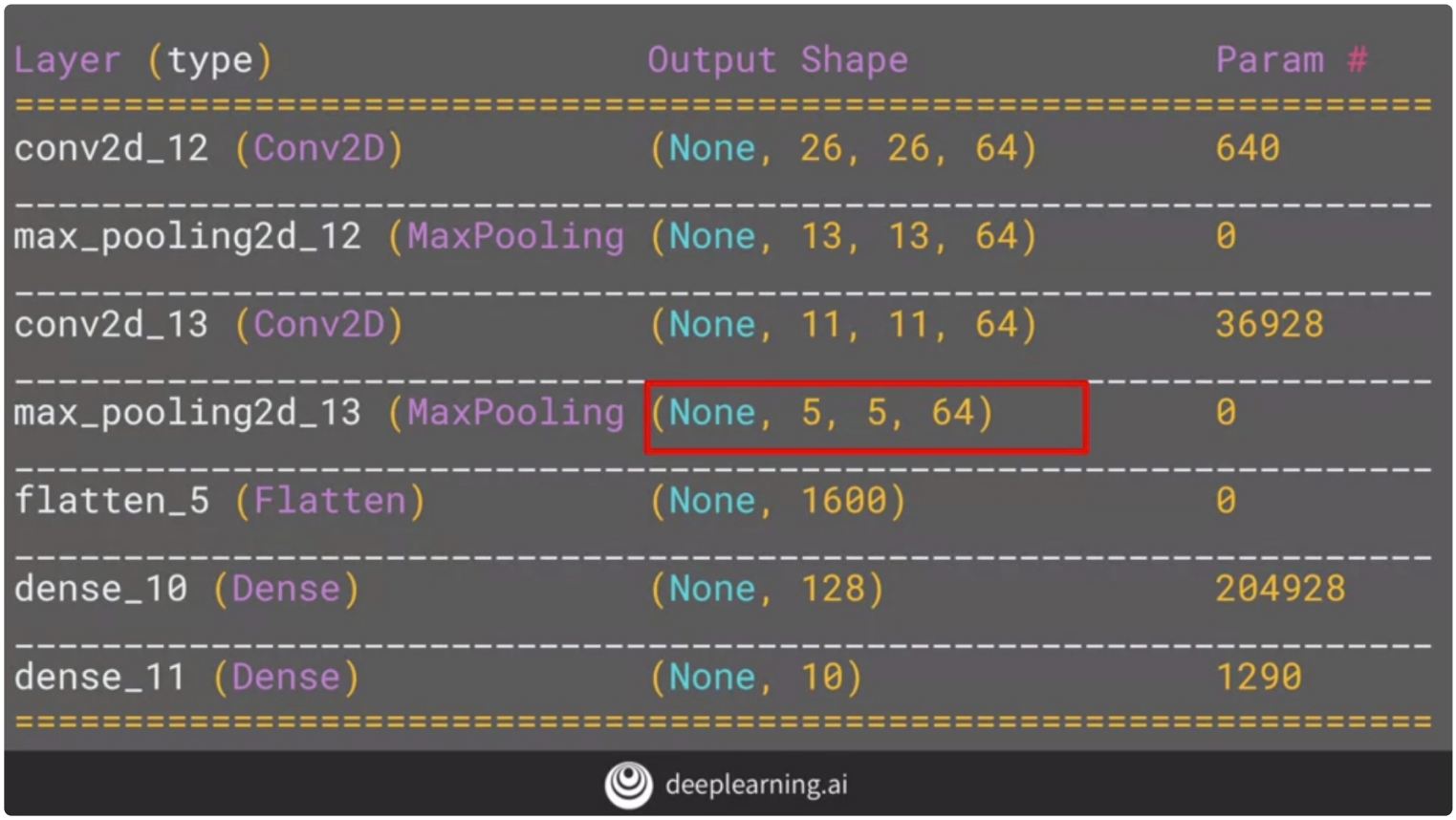

Why was convolution needed for image related problems?

One take on this question can be:

1. It reduces the overall paramter requirement for deeper NN as for Convolution layers one has a kernel being shared by the whole of image.
2. Another important take from a logical POV: if we take MNIST-fashion data, it contains a lot of spaces, extra features like color, design differences of apparel, etc. The convolution layers try to extract the essential features using image processing (by use of learned kernels or filters) before moving to Fully connected NN architecture as before. In doing so as shown in the above `model.summary()` output the image sizes are reduced from 28x28 to 5x5, however, the feature maps or the number of channels just before flattening has increased from 3 to 64. It's the increase in these feature maps or the number of images wherein lies the extracted information for easier and better classification of the image by Fully connected NN layers at the end. That's the concept of Convolutional Neural Networks. Add some layers to do convolution before you have the dense layers, and then the information going to the dense layers is more focused and possibly more accurate. Convolution layers often highlights features that distinguish one item from another. Moreover, the amount of information needed is then much less because you'll just train on the highlighted features.**Using convolutions might make your training faster or slower, and a poorly designed Convolutional layer may even be less efficient than a plain DNN!**

The convolution is followed with a MaxPool2D layer which is designed to compress the image, while maintaining the content of the features that were highlighted by the convolution. Pooling reduces information without removing all of the features.

[Conv2D] [MaxPooling2D]

You can call model.summary() to see the size and shape of the network

[model summary blog post]

Points on filters used for convolution: [Source]

1. There are a few rules about the filter:
   - ts size has to be uneven, so that it has a center, for example 3x3, 5x5 and 7x7 are ok.
   - It doesn't have to, but the sum of all elements of the filter should be 1 if you want the resulting image to have the same brightness as the original.
   - If the sum of the elements is larger than 1, the result will be a brighter image, and if it's smaller than 1, a darker image. If the sum is 0, the resulting image isn't necessarily completely black, but it'll be very dark.
2. if you're for example calculating a pixel on the left side, there are no more pixels to the left of it while these are required for the convolution (edge case for convolution). You can either use value 0 here (padding) or wrap it around to the other side of the image.
3. The resulting pixel values after applying the filter can be negative or larger than 255, if that happens you can truncate them so that values smaller than 0 are made 0, and values larger than 255 are set to 255. For negative values, you can also take the absolute value instead.
4. image filters aren't feasible for real-time applications and games yet, but they're useful in image processing.
5. Digital audio and electronic filters work with convolution as well, but in 1D.
6. Blurring is done for example by taking the average of the current pixel and its 4 neighbors. For a 3x3 filter, take the sum of the current pixel and its 4 neighbors, and divide it through 5, or thus fill in 5 times the value 0.2 in the filter. The more blur you want, the bigger the filter has to be, or you can apply the same small blur filter multiple times. In the blurring above, the kernel we used is rather harsh. A much smoother blur is achieved with a gaussian kernel. With a Gaussian kernel, the value exponentially decreases as we go away from the center.