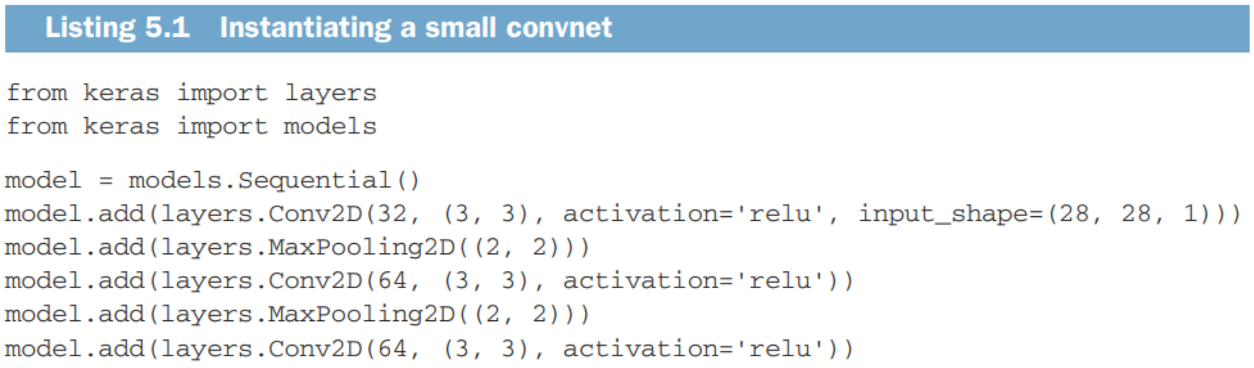
Deep learning for computer vision

2021.6.19

5.1 Intro to convnets

The following lines of code show what a basic convnet looks like. It's a stack of Conv2D and MaxPooling2D layers.

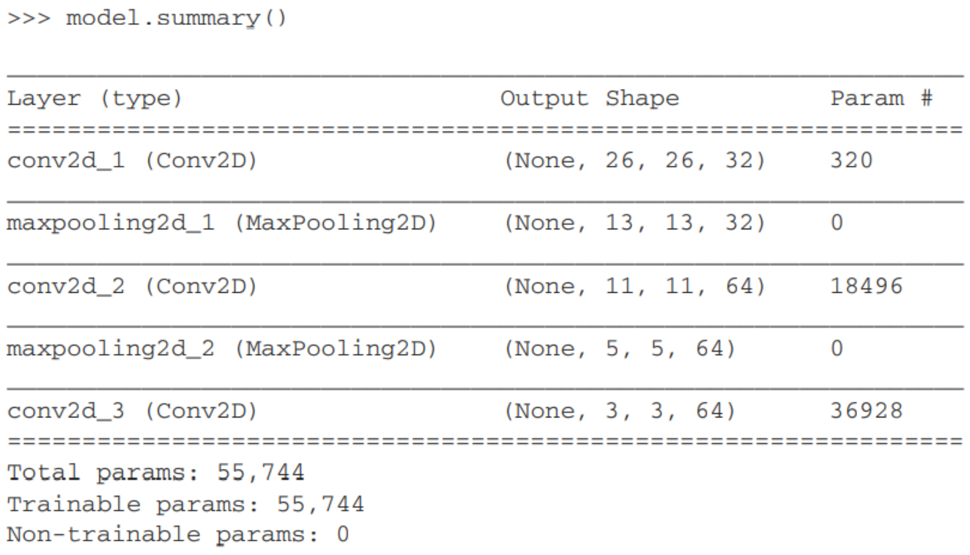


Listing 5.1 Instantiating a small convnet

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

A convnet takes as input tensors of shape (*image_height*, *image_width*, *image_channels*). In the case above, we'll configure the convnet to process input of size (28, 28, 1). We'll do this by passing the argument input_shape=(28, 28, 1) to the first layer.
Display the architecture of the convnet so far.



```
>>> model.summary()

________________________________________________________________
Layer (type)                     Output Shape          Param #
================================================================
conv2d_1 (Conv2D)                (None, 26, 26, 32)    320
________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 13, 13, 32)    0
________________________________________________________________
conv2d_2 (Conv2D)                (None, 11, 11, 64)    18496
________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 5, 5, 64)      0
________________________________________________________________
conv2d_3 (Conv2D)                (None, 3, 3, 64)      36928
================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

We can see that the output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as going deeper in the network. The number of channels is controlled by the first argument passed to the Conv2D layers.
The next step is to feed the last output tensor into a densely connected classifier network - A stack of Dense layers.
These classifiers process vectors, which are 1D, whereas the current output is a 3D tensor. Firstly we need to flatten the 3D outputs to 1D, and then add a few Dense layers on top.

Listing 5.2 Adding a classifier on top of the convnet

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
>>> model.summary()

Layer (type)                     Output Shape          Param #
================================================================
conv2d_1 (Conv2D)                (None, 26, 26, 32)    320
________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 13, 13, 32)    0
________________________________________________________________
conv2d_2 (Conv2D)                (None, 11, 11, 64)    18496
________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 5, 5, 64)      0
________________________________________________________________
conv2d_3 (Conv2D)                (None, 3, 3, 64)      36928
________________________________________________________________
flatten_1 (Flatten)              (None, 576)           0
________________________________________________________________
dense_1 (Dense)                  (None, 64)            36928
________________________________________________________________
dense_2 (Dense)                  (None, 10)            650
================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

We can find that (3, 3, 64) outputs are flattened into vectors of shape (576,) before going through two Dense layers.

There will be an little coding example here

5.1.1 Convolution operation

The fundamental difference between a densely connected layer and a convolution layer is: Dense layers learn **global patterns** in their input feature space, whereas convolution layers learn **local patterns**.
This key characteristic gives convnets two interesting properties:
1. The patterns NN learn are translation invariant.
2. NN can learn spatial heirarchies of patterns.

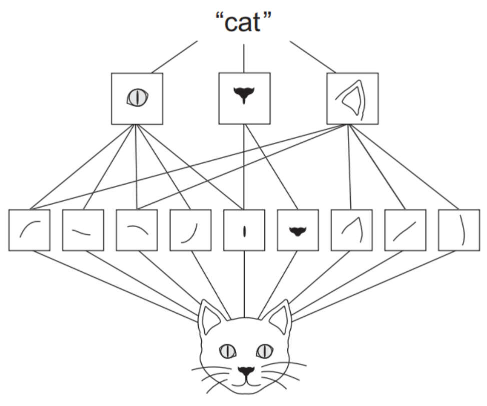


Figure 5.2 The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as "cat."

Convolutions operate over 3D tensors, called *feature maps*,

with two spatial axes (height and width) as well as a depth axis (also called *channels axis*).
The convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an *output feature map.* This output feature map is still a 3D tensor: it has a width and a height. Its depth can be arbitrary, because the output depth is a parameter of the layer, and the different channels in that depth axis no longer stand for specific colors as in RGB input; rather, they stand for filters.
In the MNIST example, the first convolution layer takes a feature map of size (28, 28, 1) and outputs a feature map of size (26, 26, 32): it computes 32 filters over its input. Each of these 32 output channels contains a 26 × 26 grid of values, which is a *response map* of the filter over the input, indicating the response of that filter pattern at different locations in the input.
That is what the term feature map means: every dimension in the depth axis is a feature (or filter), and the 2D tensor output[:, :, n] is the 2D spatial map of the response of this filter over the input.

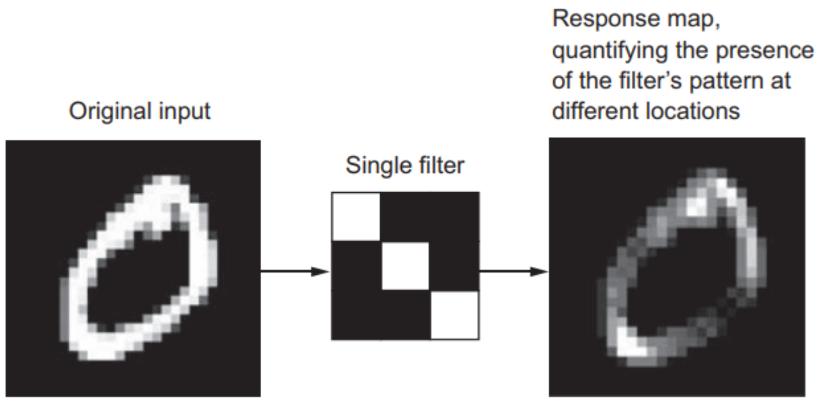

Original input
Single filter
Response map, quantifying the presence of the filter's pattern at different locations

Figure 5.3 The concept of a *response map*: a 2D map of the presence of a pattern at different locations in an input

Convolutions are defined by two key parameters:
1. Size of the patches extracted from the inputs.
2. Depth of the output feature map.

A convolution works by sliding these windows of size 3 × 3 or 5 × 5 over the 3D input feature map, stopping at every possible location, and extracting the 3D patch of surrounding features (shape (window_height, window_width, input_depth)). Each such 3D patch is then transformed into a 1D vector of shape (output_depth,). All of these vectors are then spatially reassembled into a 3D output map of shape (height, width, output_depth). Every spatial location in the output feature map corresponds to the same location in the input feature map.

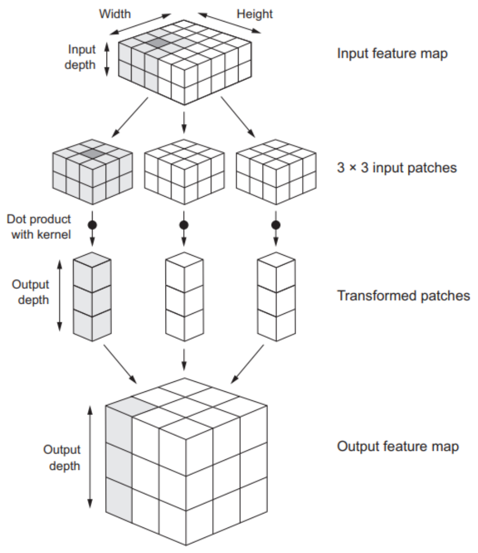


Figure 5.4 How convolution works

Note that the output width and height may differ from the input width and height due to *border effect* and *strides*.

5.1.2 The max-pooling operation

We found that before the first MaxPooling2D layers, the feature map is 26 × 26, but the max-pooling operation halves it to 13 × 13. That's the role of max pooling: to aggressively downsample feature maps. Max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel. They're transformed via a hardcoded **max** tensor operation. A big difference from convolution is that max pooling is usually done with 2 × 2 windows, in order to downsample the feature maps by a factor of 2. On the other hand, convolution is typically done with 3 × 3 windows.

At this point, we should understand the basics of convnets—feature maps, convolution, and max pooling. Now let's move on to more useful, practical applications.

5.2 Training a convet from a scratch on a small dataset