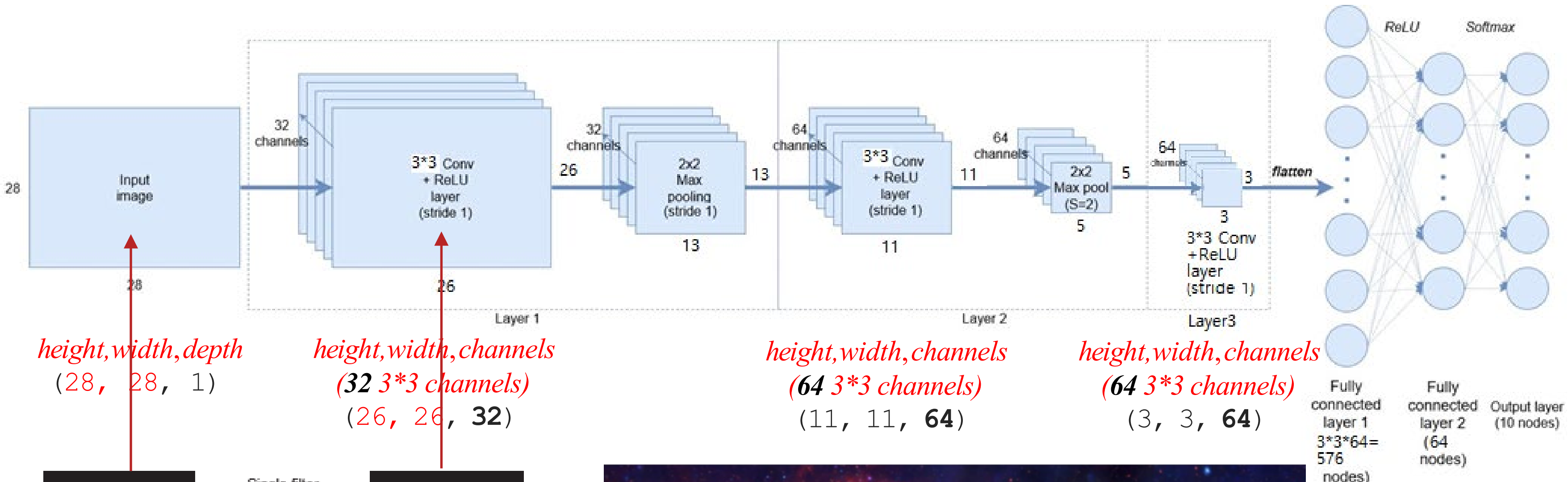
5장 *Deep learning for computer vision*

“Out of the Box”

This chapter covers

- Understanding convolutional neural networks (convnets)
- Using data augmentation to mitigate overfitting
- Using a pretrained convnet to do feature extraction
- Fine-tuning a pretrained convnet
- Visualizing what convnets learn and how they make classification decisions

5.1 Introduction to convnets



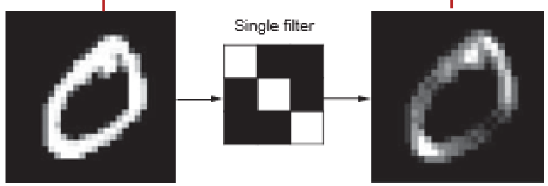


Figure 5.3 The concept of a response map

Architecture of the Convolutional neural network

5.1 Introduction to convnets

- a convnet to classify MNIST digits - its accuracy will blow out of the water that of the densely connected model
- a basic convnet - a stack of Conv2D and MaxPooling2D layers

Listing 5.1 Instantiating a small convent

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',    # The number of channels
                  input_shape=(28, 28, 1))) # (image_height, image_width, image_channels)
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'))
# padding = 'valid', stride = 1   # default
>>> 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
```

5.1 Introduction to convnets

- flatten the 3D outputs to 1D, and then add a few `Dense` layers on top.
- feed the last output tensor (of shape `(3, 3, 64)`) into a densely connected classifier network: a stack of `Dense` layers.

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
================================================================
```

5.1 Introduction to convnets

Listing 5.3 Training the convnet on MNIST images

```
from keras.datasets import mnist
from keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop',
     loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

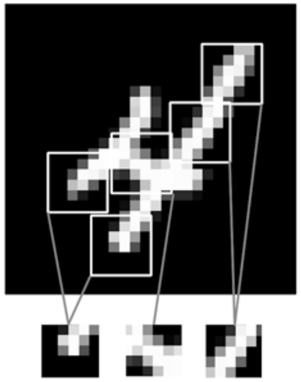
- Let's evaluate the model on the test data:

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> test_acc
0.9908
```

- Whereas the densely connected network had a test accuracy of 97.8%, the basic convnet has a test accuracy of 99.3%: we decreased the error rate by 68% (relative). Not bad!
- Why? Let's dive into what the `Conv2D` and `MaxPooling2D` layers do.

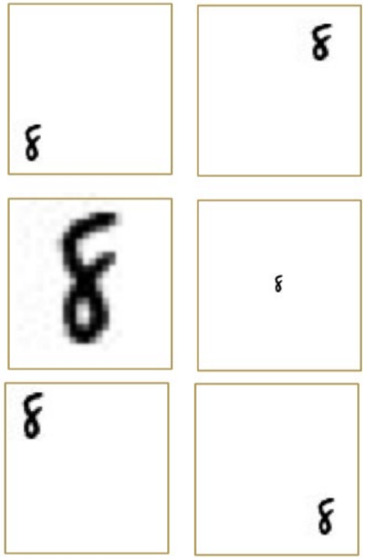
5.1 Introduction to convnets

Figure 5.1 Images can be broken into local patterns such as edges, textures, and so on.



5.1.1 The convolution operation

- `Dense` layers - learn global patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels),
- convolution layers - learn local patterns (see figure 5.1): in the case of images, patterns found in small 2D windows (3×3, etc.) of the inputs.
- This key characteristic gives convnets two interesting properties:
  - *translation invariant* - After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere.
  - A densely connected network would have to learn the pattern a new if it appeared at anew location ***vs.*** Convnets need fewer training samples to learn representations that have **generalization power**.



5.1 Introduction to convnets

5.1.1 The convolution operation

- *They can learn spatial hierarchies of patterns (see figure 5.2).*
- A first convolution layer - learn small local patterns such as edges,
- A second convolution layer - learn larger patterns made of the features of the first layers. This allows convnets to efficiently learn increasingly complex and abstract visual concepts (because *the visual world is fundamentally spatially hierarchical*).

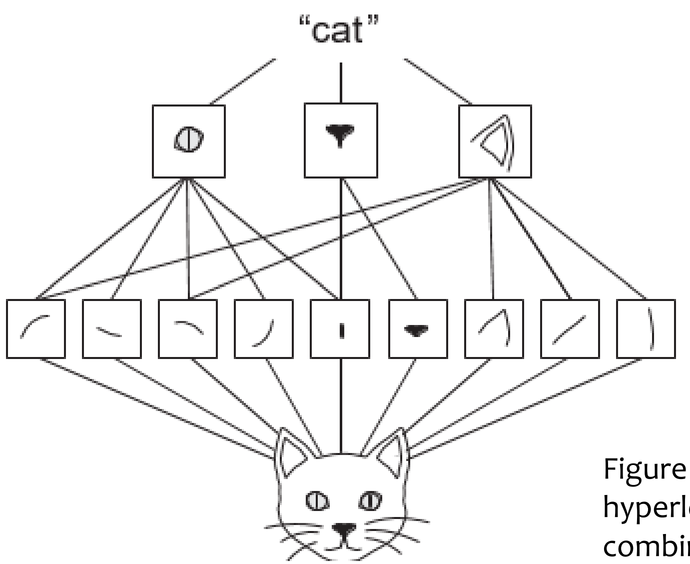


Figure 5.2 The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as "cat."

5.1 Introduction to convnets

5.1.1 The convolution operation

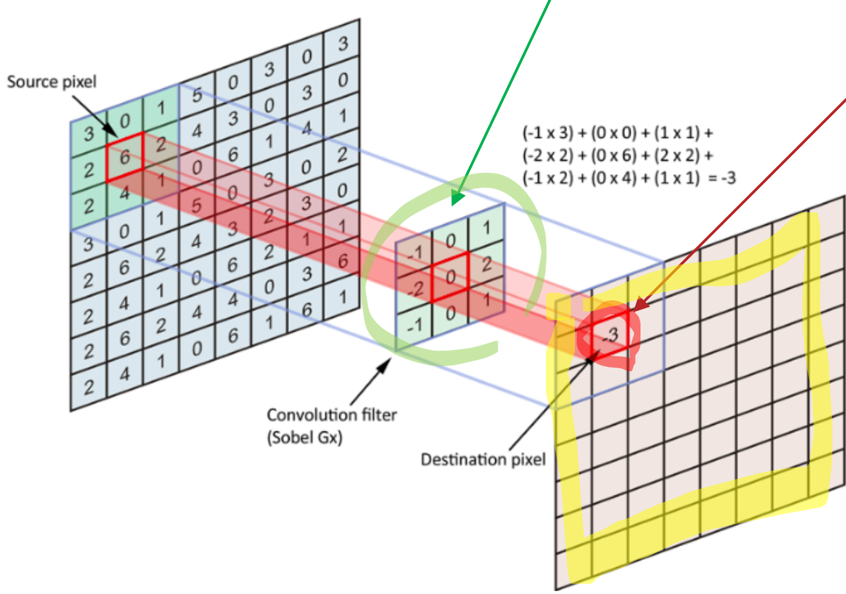
- The convolution operation: input feature map → *output feature* with 3D tensor (height, width, channels)
- The first convolution layer in the MNIST : a feature map of size `(28, 28, 1)` → outputs a feature map of size `(26, 26, 32)`
- Convolutions are defined by two key parameters:
  - *Size of the filters* — 3 × 3 or 5 × 5
  - *number of* channels — **32** or 64
- In Keras `Conv2D` layers, these parameters are the first arguments passed to the layer:

```
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
```

5.1 Introduction to convnets

5.1.1 The convolution operation

- 3D output map of shape (height, width, output_depth) → 1D vector of shape (output_depth,)
- For instance, with 3 × 3 windows, the vector output[i, j, :] comes from the 3D patch input[i-1:i+1, j-1:j+1, :]
  - Border effects, which can be countered by padding the input feature map
  - The use of *strides*, which I'll define in a second



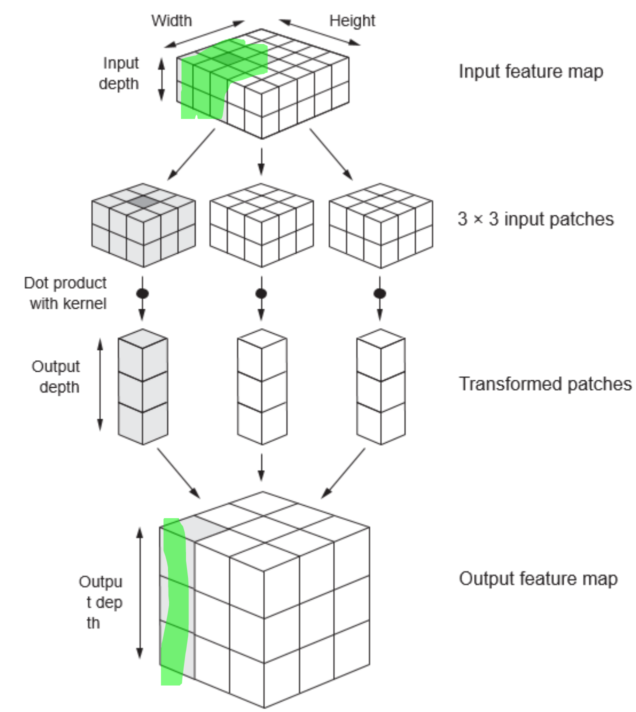


Figure 5.4 How convolution works

5.1 Introduction to convnets

5.1.1 The convolution operation

UNDERSTANDING BORDER EFFECTS AND PADDING

- border effect - 5 × 5 feature map (25 tiles total) → output feature map 3 × 3 by 3 × 3 filter
- 28 × 28 inputs → 26 × 26 after the first convolution
- padding argument: "valid", which means no padding (only valid window locations will be used); and "same", which means "pad in such a way as to have an output with the same width and height as the input." The padding argument defaults to "valid".

```
model.add(layers.Conv2D(32, (3, 3), padding='valid',
                   input_shape=(28, 28, 1), activation='relu'))
```

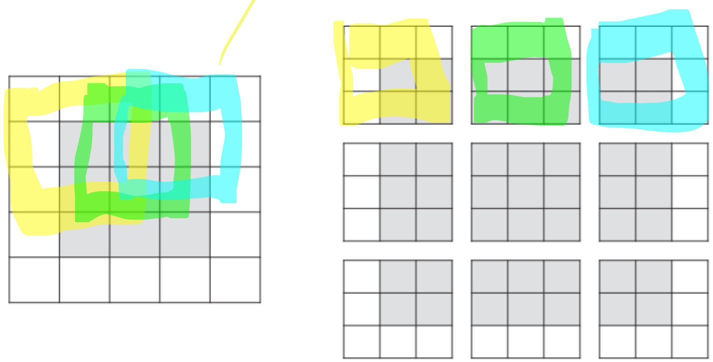


Figure 5.5 Valid locations of 3 × 3 patches in a 5 × 5 input feature map

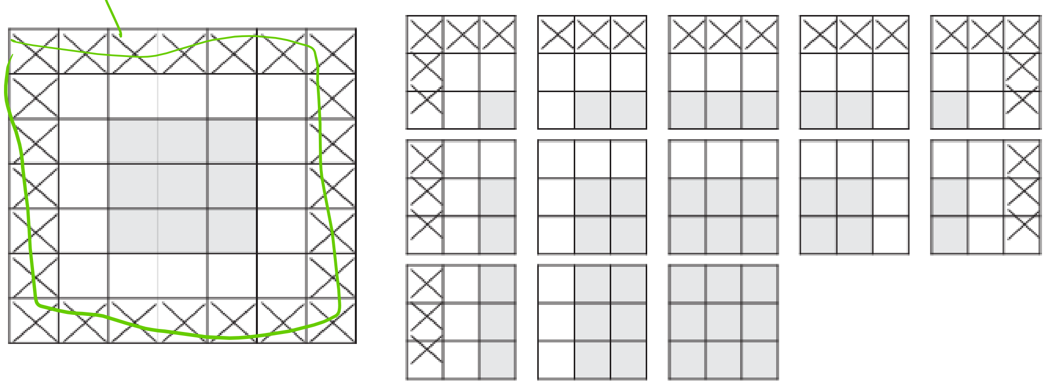


Figure 5.6 Padding a 5 × 5 input in order to be able to extract 25 3 × 3 patches

5.1 Introduction to convnets

5.1.1 The convolution operation

UNDERSTANDING CONVOLUTION STRIDES

- *strides* - The distance between two successive windows is a parameter of the convolution, called its *stride*, which defaults to 1.
- It's possible to have *strided convolutions* : 3 × 3 convolution with stride 2 over a 5 × 5 input (without padding).

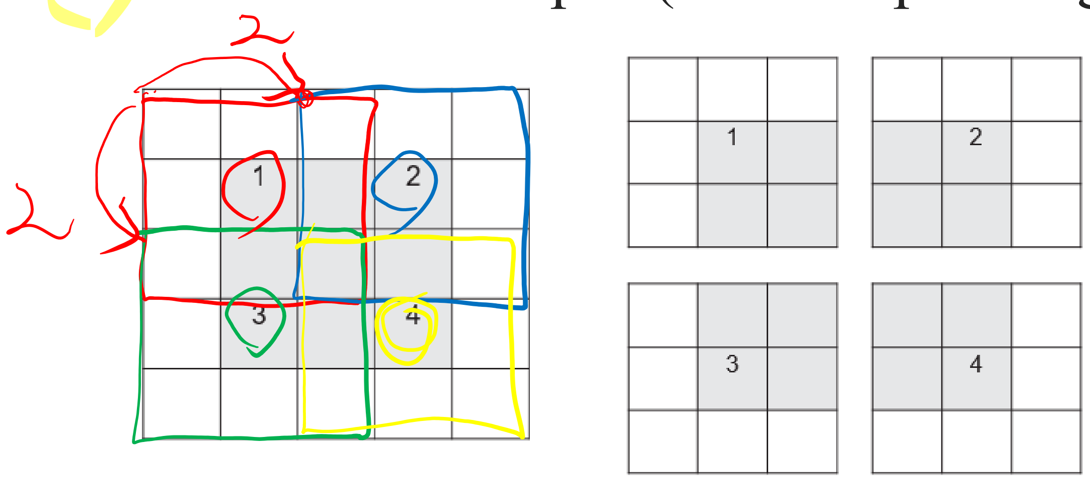


Figure 5.7 3 × 3 convolution patches with 2 × 2 strides

5.1.2 The max-pooling operation

- max-pooling operation - feature map 26 × 26 → 13 × 13
- role of max pooling: aggressively downsample feature maps, much like strided convolutions.
- max pooling is usually done with 2 × 2 windows and stride 2.

```
model.add(MaxPooling2D(pool_size=(2, 2),
                strides=(2, 2)))
```

- contain information about the totality of the input

5.1 Introduction to convnets

5.1.2 The max-pooling operation