Deep learning for Computer Vision

Convolutional Neural Networks (convnets)



APIs for building models in Keras

1. The Sequential model, it is basically a Python list;
   It is limited to simple stacks of layers.

2. The Functional API, which focuses on graph-like model architectures.
   It represents a nice mid-point between usability and flexibility.

3. Model subclassing, a low-level option where you write everything
   yourself from scratch. This is ideal if you want full control over every little thing.

The Sequential model

```
from tensorflow import keras
from tensorflow.keras import layers
```

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

Building the same model incrementally via the add() method:

```
model = keras.Sequential()
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(10, activation="softmax"))
```

Technical note: *Models that aren't yet built have no weights*

```
>>> model.weights                                    ❶
ValueError: Weights for model sequential_1 have not yet been created.
```

```
>>> model.build(input_shape=(None, 3))               ❶
>>> model.weights                                    ❷
```

Batch size

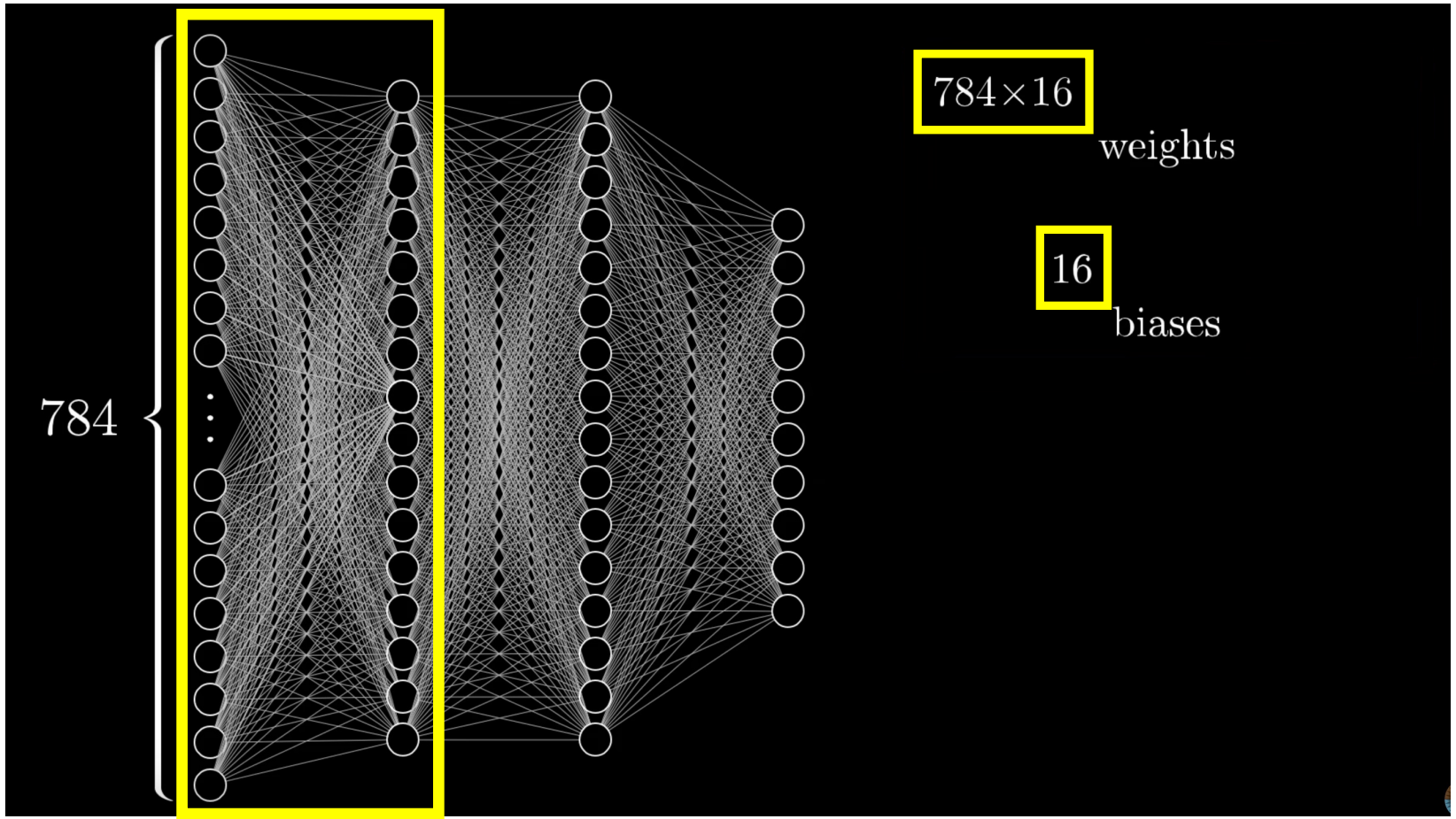
Builds the model and now the model will expect samples (input) of shape (3,)

The Sequential model

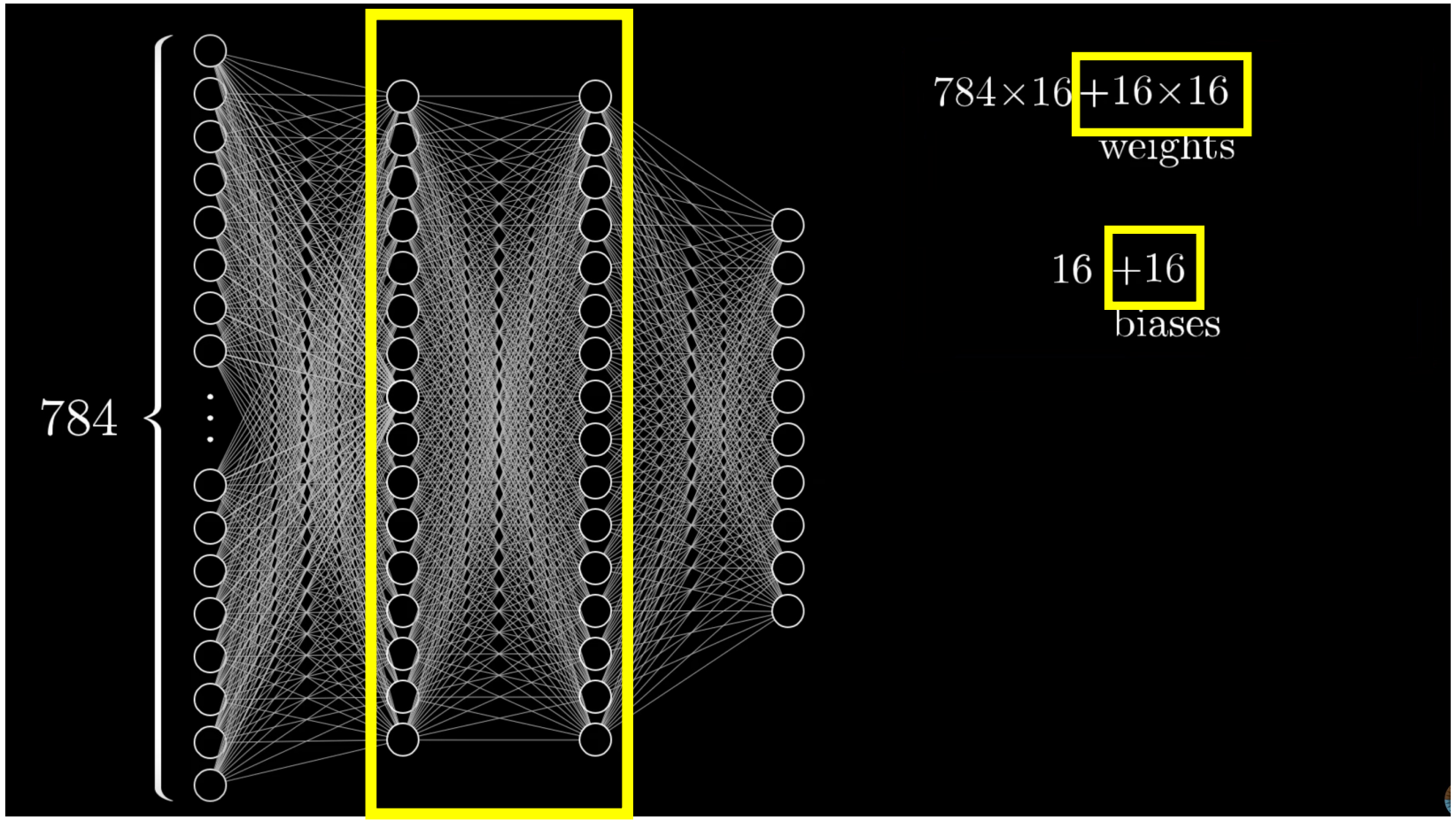
The summary() method: display its contents

```
>>> model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 64)                256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 906
Trainable params: 906
Non-trainable params: 0
_________________________________________________________________
```

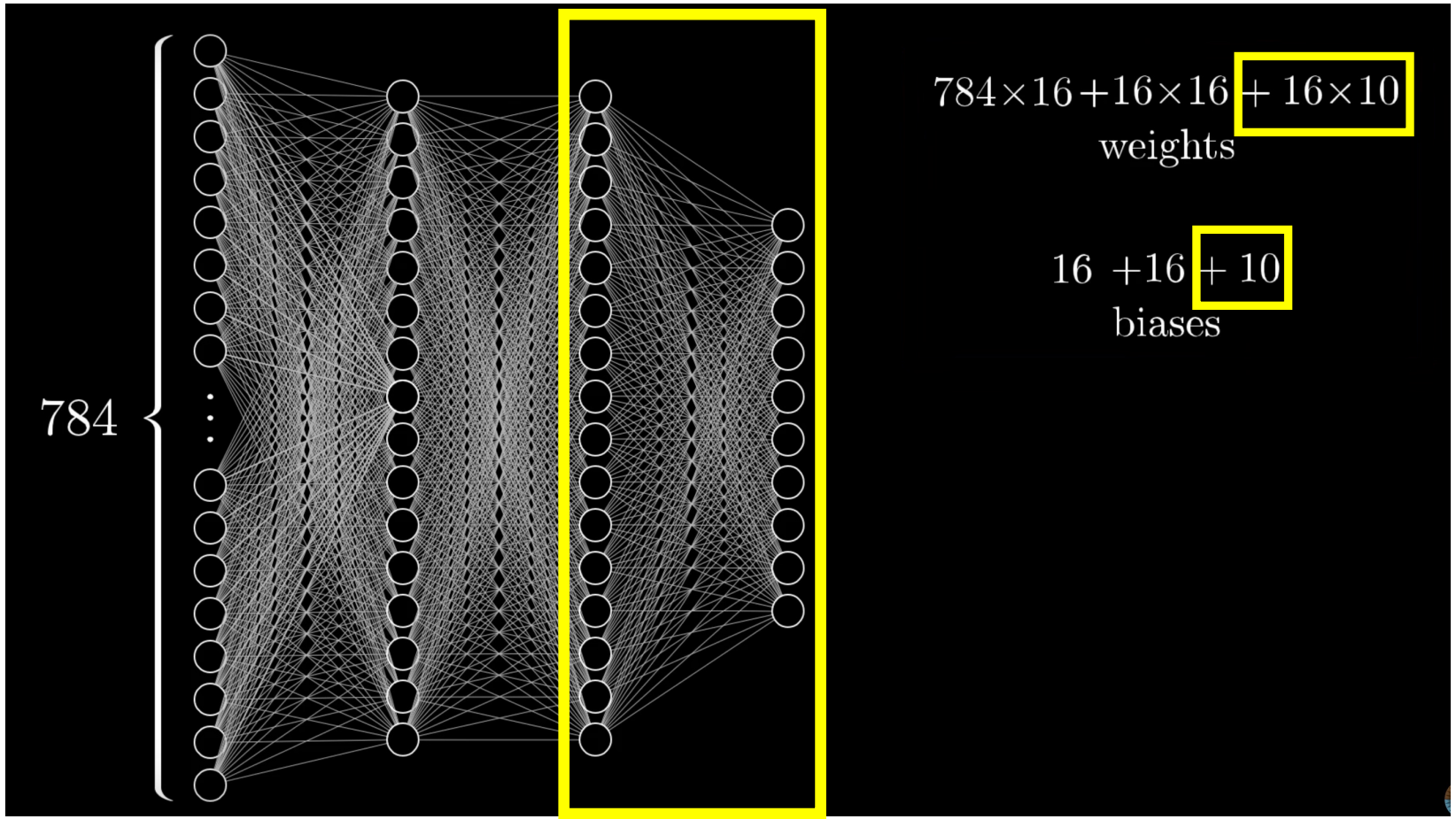
Counting weights and biases



Counting weights and biases



Counting weights and biases



The Sequential model

The summary() method: display its contents

```
>>> model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 64)                256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 906
Trainable params: 906
Non-trainable params: 0
_________________________________________________________________
```

$= 3 \times 64 + 64$

$= 64 \times 10 + 10$

The Functional API

A simple Functional model with two Dense layers

```
inputs = keras.Input(shape=(3,), name="my_input")
features = layers.Dense(64, activation="relu")(inputs)
outputs = layers.Dense(10, activation="softmax")(features)
model = keras.Model(inputs=inputs, outputs=outputs)
```

"It feels like playing with LEGO bricks"

The Functional API

A simple Functional model with two Dense layers

```
inputs = keras.Input(shape=(3,), name="my_input")
features = layers.Dense(64, activation="relu")(inputs)
outputs = layers.Dense(10, activation="softmax")(features)
model = keras.Model(inputs=inputs, outputs=outputs)
```

"It feels like playing with LEGO bricks"

The Functional API

A simple Functional model with two Dense layers

```
inputs = keras.Input(shape=(3,), name="my_input")
features = layers.Dense(64, activation="relu")(inputs)
outputs = layers.Dense(10, activation="softmax")(features)
model = keras.Model(inputs=inputs, outputs=outputs)
```

"It feels like playing with LEGO bricks"

The Functional API

A simple Functional model with two Dense layers

```
inputs = keras.Input(shape=(3,), name="my_input")
features = layers.Dense(64, activation="relu")(inputs)
outputs = layers.Dense(10, activation="softmax")(features)
model = keras.Model(inputs=inputs, outputs=outputs)
```

"It feels like playing with LEGO bricks"

The Functional API

A simple Functional model with two Dense layers

Inputs object holds information about the shape and dtype of the data that the model will process

```
>>> inputs.shape
(None, 3)          ❶
>>> inputs.dtype   ❷
float32
```

❶ The model will process batches where each sample has shape (3,). The number of samples per batch is variable (indicated by the None batch size).

❷ These batches will have dtype float32.

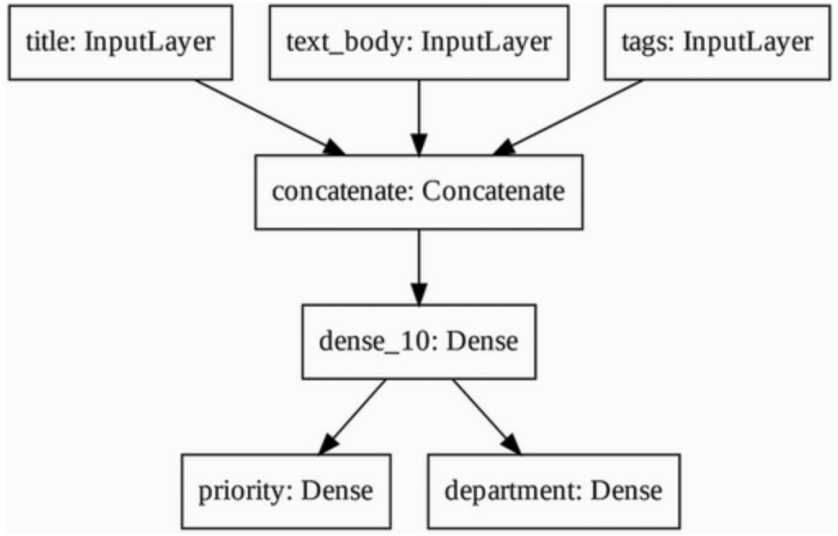
```
>>> features.shape
(None, 64)
```

"It feels like playing with LEGO bricks"

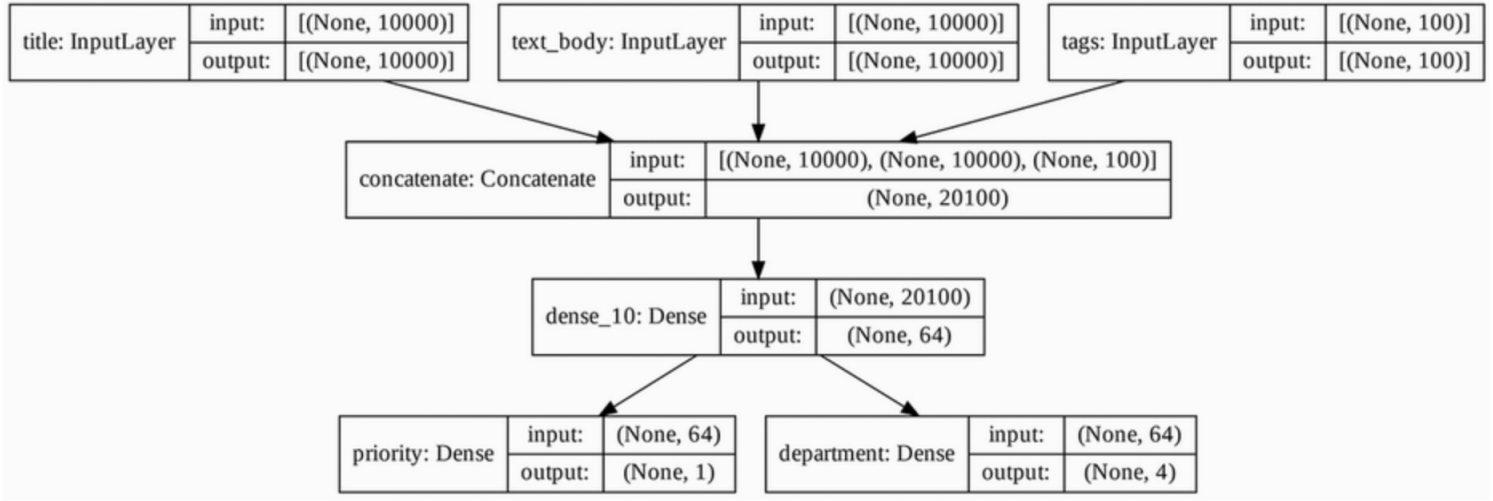
The Functional API:

Access to layer connectivity: plot_model()

```
keras.utils.plot_model(model, "ticket_classifier.png")
```




```
keras.utils.plot_model(
    model, "ticket_classifier_with_shape_info.png", show_shapes=True)
```



The "**None**" in the tensor shapes represents the batch size: this model allows batches of any size.

Instantiating a small convnet

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Instantiating a small convnet

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 10)                11530

=================================================================
Total params: 104202 (407.04 KB)
Trainable params: 104202 (407.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training the convnet on MNIST images

```
from tensorflow.keras.datasets import mnist

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

Evaluating the convnet on MNIST images

Evaluating the convnet

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc:.3f}")
```

```
313/313 [==============================] - 4s 12ms/step - loss: 0.0264 - accuracy: 0.9920
Test accuracy: 0.992
```

Generic model (chapter 2)

```
model = keras.Sequential([
layers.Dense(512, activation="relu"),
layers.Dense(10, activation="softmax")
])
```

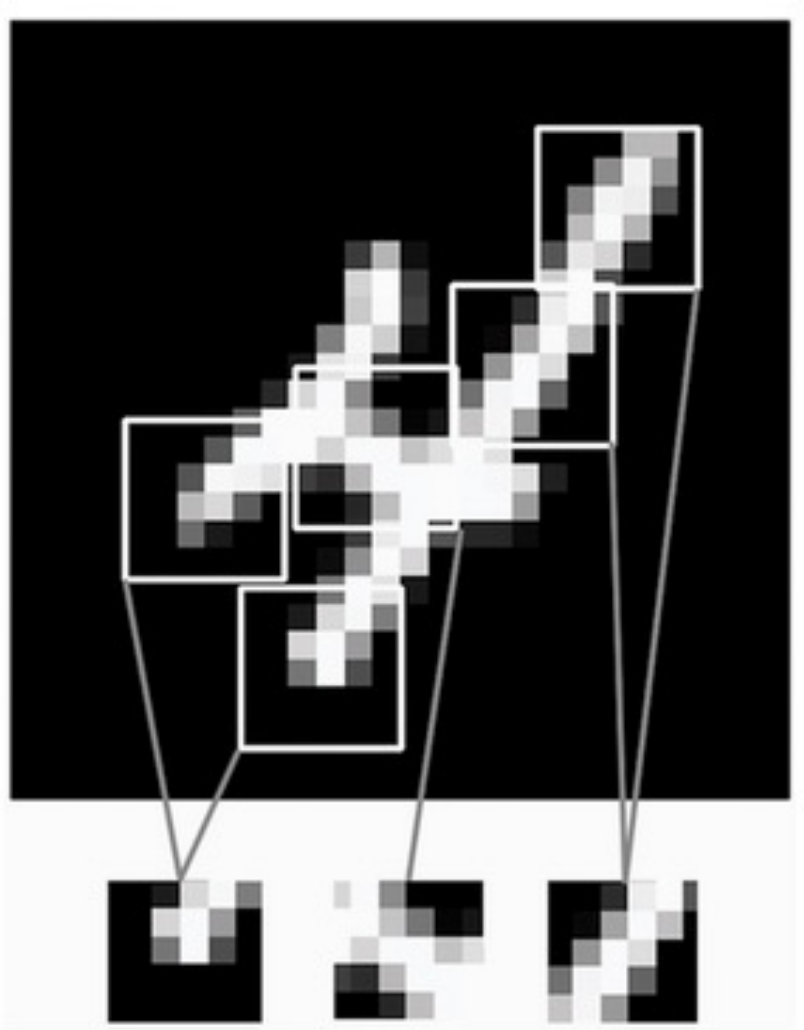
```
313/313 [==============================] - 3s 7ms/step - loss: 0.0661 - accuracy: 0.9801
test_acc: 0.9800999760627747
```

The convolution operation

- The fundamental difference between a densely connected layer and a convolution layer is this:

- **Dense layers** learn global patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels)

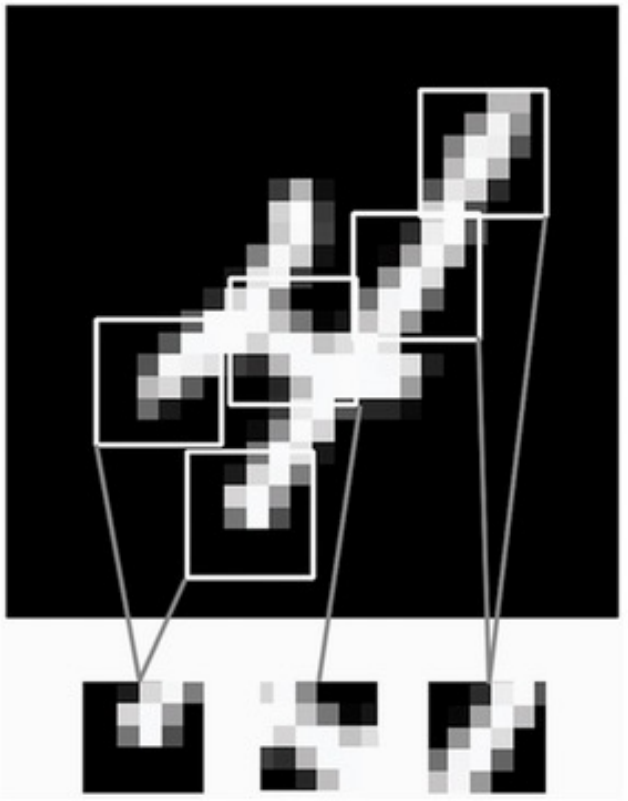
- **Convolution layers** learn local patterns—in the case of images, patterns found in small 2D windows of the inputs.



The convolution operation

The patterns they learn are translation-invariant

- After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere: for example, in the upper-left corner;
- A densely connected model would have to learn the pattern anew if it appeared at a new location;
- This makes convnets data-efficient when processing images;
- They need fewer training samples to learn representations that have generalization power.

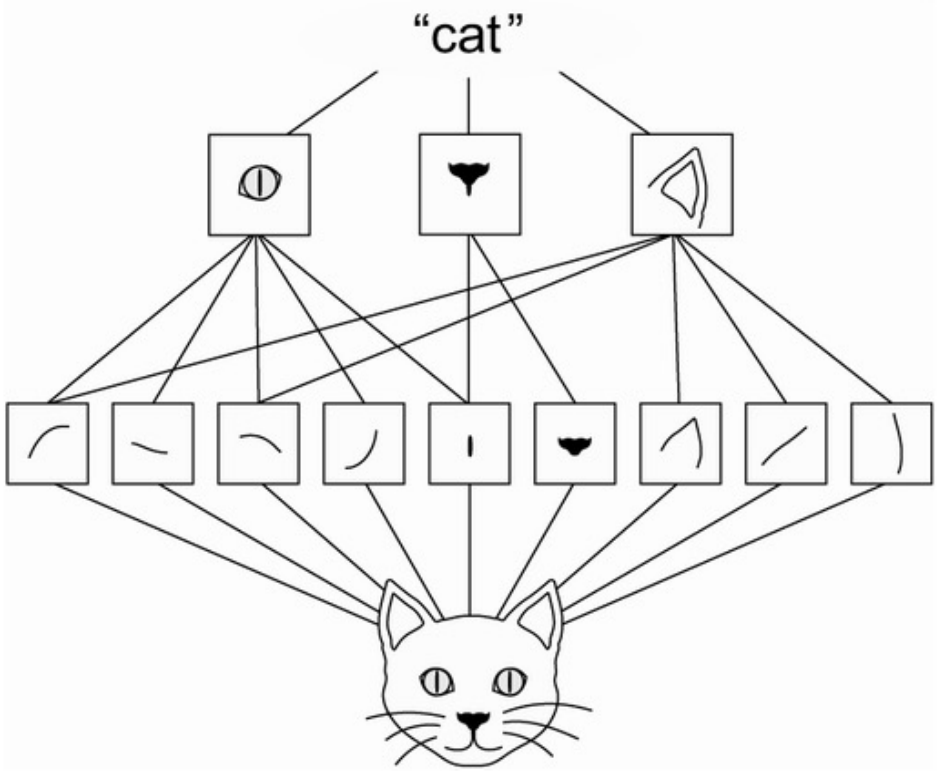


The *visual world is fundamentally translation-invariant*

The convolution operation

They can learn spatial hierarchies of patterns

- A first convolution layer will learn small local patterns such as edges;
- a second convolution layer will learn larger patterns made of the features of the first layers, and so on;
- This allows convnets to efficiently learn increasingly complex and abstract visual concepts;



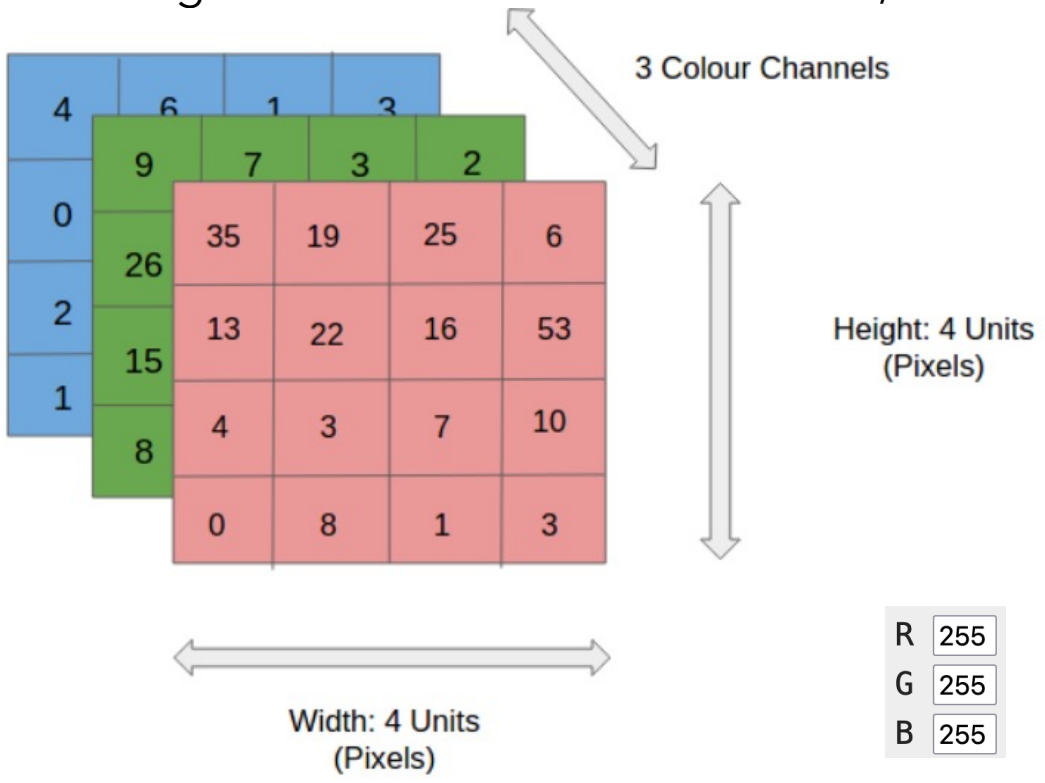
The visual world is *fundamentally spatially hierarchical*

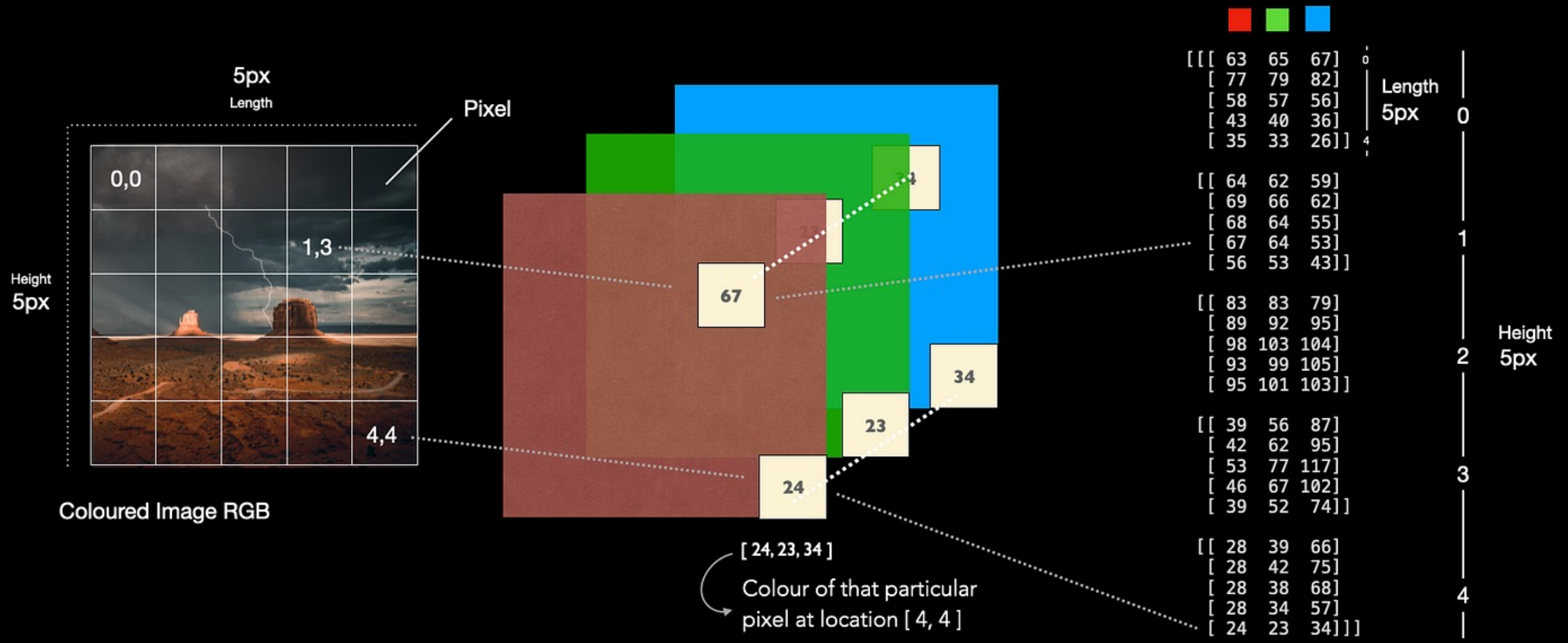
The convolution operation

Convolutions operate over rank-3 tensors called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis);

For an RGB image, the dimension of the depth axis is 3, because the image has three color channels: red, green, and blue.

For a black-and-white picture, like the MNIST digits, the depth




5px
Length
Pixel
Height
5px
0,0
1,3
4,4
Coloured Image RGB
67
34
23
24
[24, 23, 34]
Colour of that particular
pixel at location [4, 4]
[[[63 65 67]
[77 79 82]
[58 57 56]
[43 40 36]
[35 33 26]]
[[64 62 59]
[69 66 62]
[68 64 55]
[67 64 53]
[56 53 43]]
[[83 83 79]
[89 92 95]
[98 103 104]
[93 99 105]
[95 101 103]]
[[39 56 87]
[42 62 95]
[53 77 117]
[46 67 102]
[39 52 74]]
[[28 39 66]
[28 42 75]
[28 38 68]
[28 34 57]
[24 23 34]]]
Length
5px
Height
5px
0
1
2
3
4

The convolution operation

Convolutions operate over rank-3 tensors called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis);

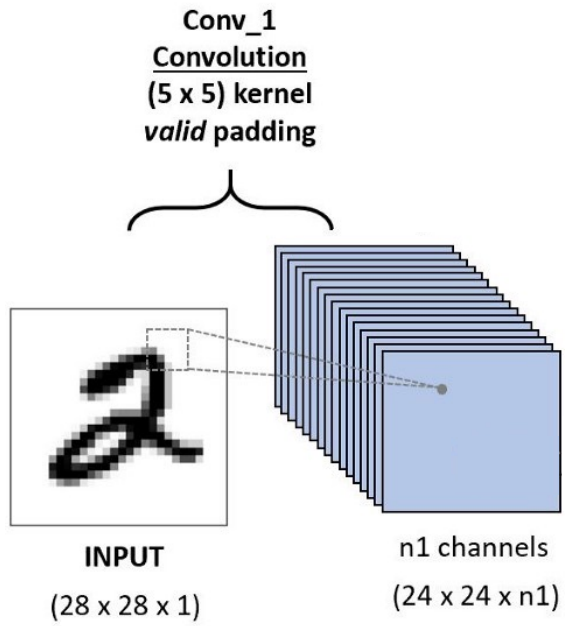
For an RGB image, the dimension of the depth axis is 3, because the image has three color channels: red, green, and blue.

For a black-and-white picture, like the MNIST digits, the depth is 1 (levels of gray).

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 rank-3 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.

Filters encode specific aspects of the input data: at a high level, a single filter could encode the concept “presence of a face in the input,” for instance;



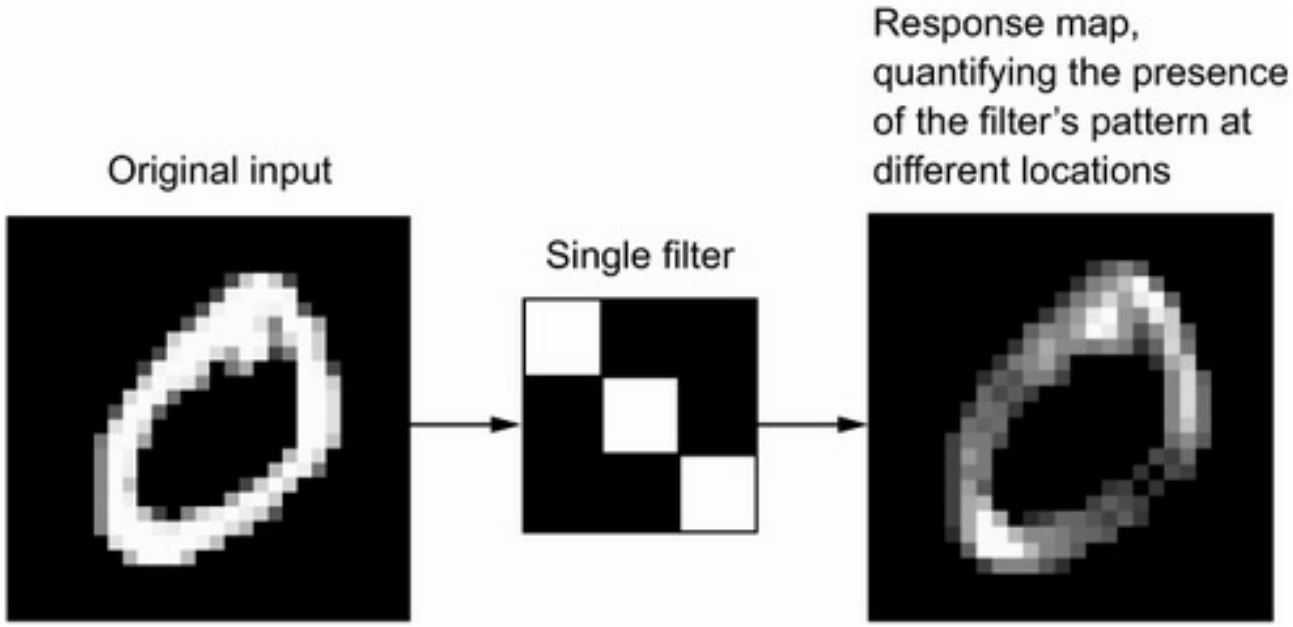
The convolution operation

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;

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320
```



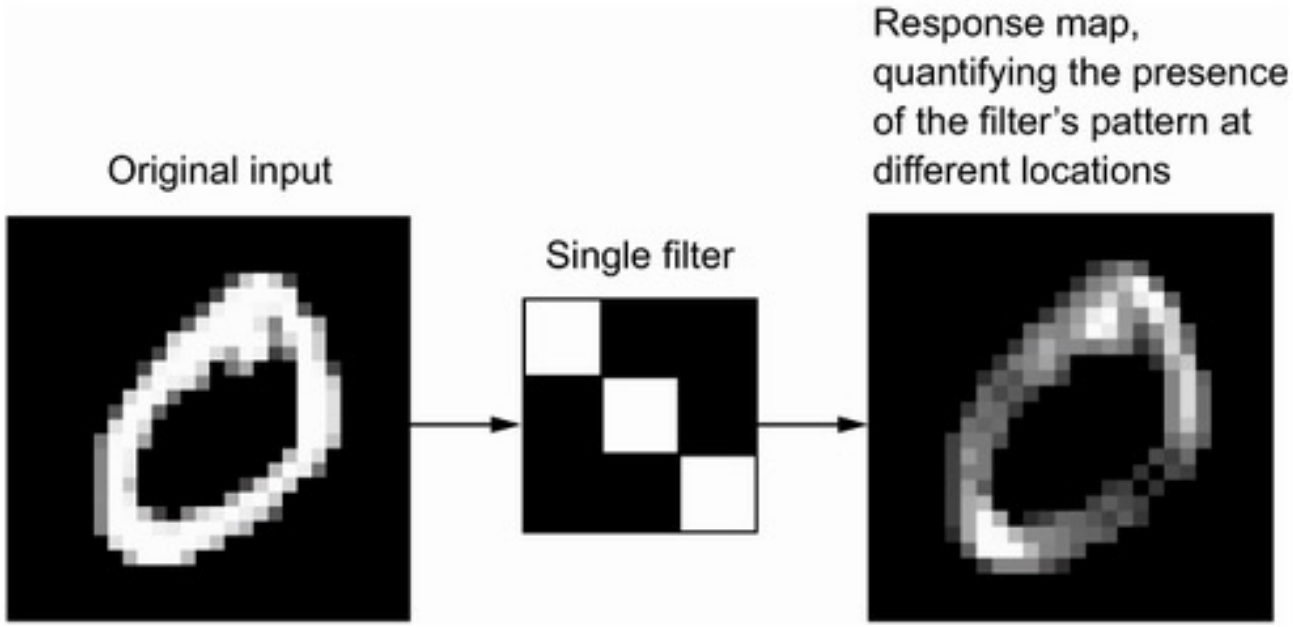
The convolution operation

The term feature map means: every dimension in the depth axis is a feature (or filter), and the rank-2 tensor **output[:, :, n]** is the 2D spatial map of the response of this filter over the input;

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320
```



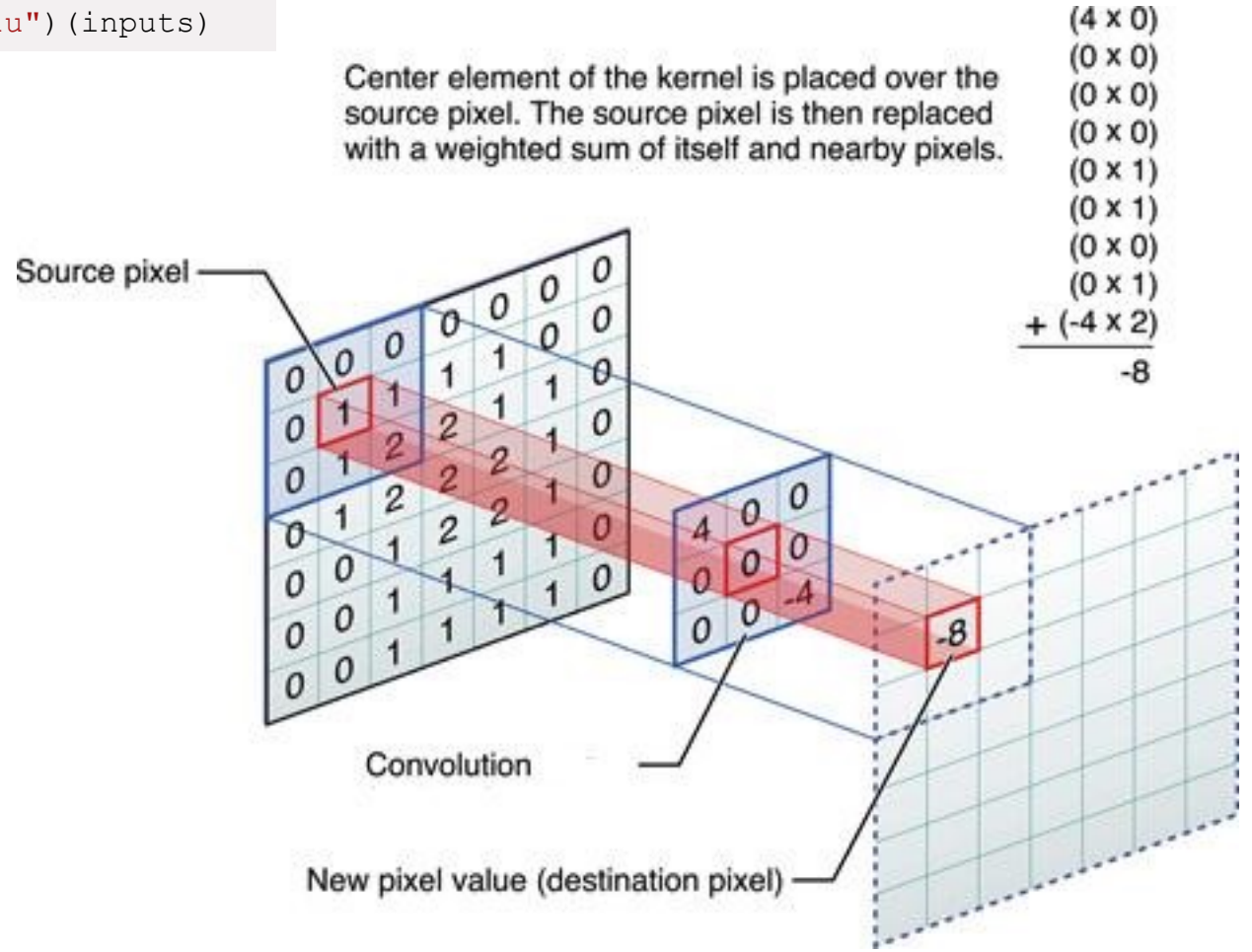
The convolution operation

```
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
```

Convolutions are defined by two key parameters:

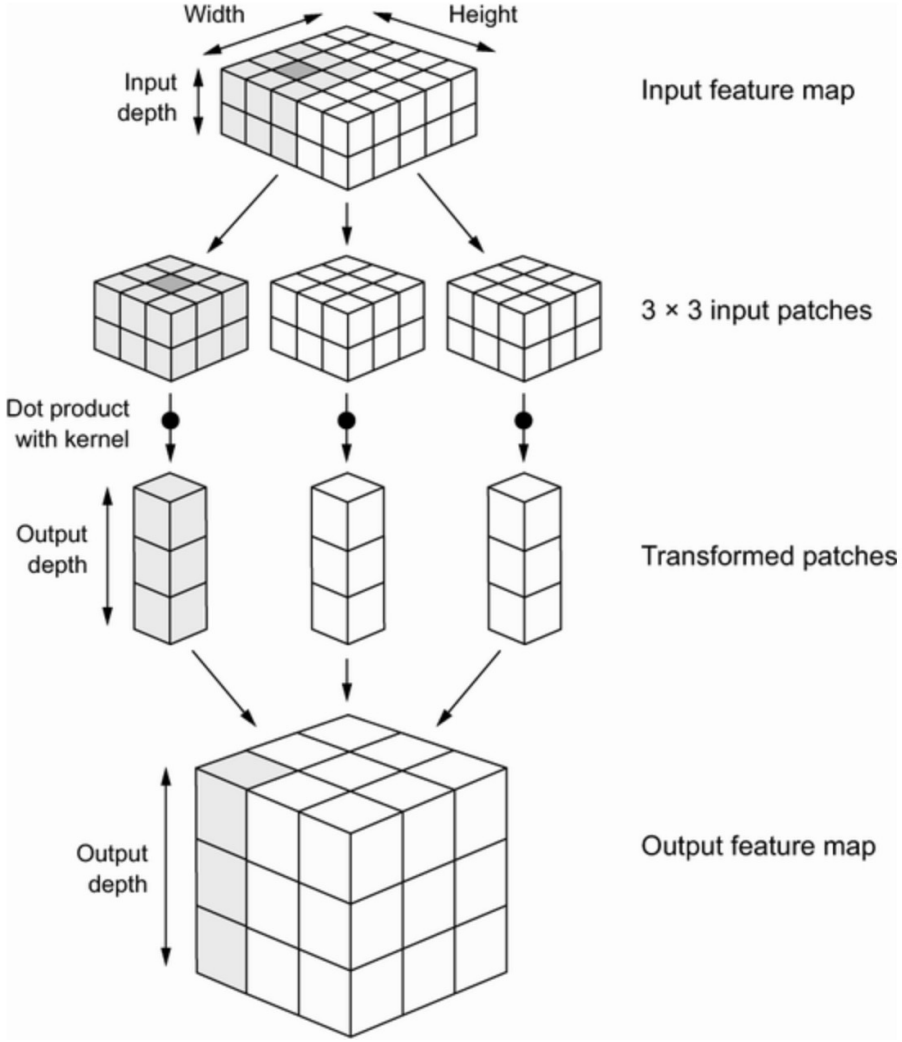
Size of the patches extracted from the inputs:These are typically 3 × 3 or 5 × 5;

Depth of the output feature map: This is the number of filters computed by the convolution. The example started with a depth of 32 and ended with a depth of 64.



The convolution operation

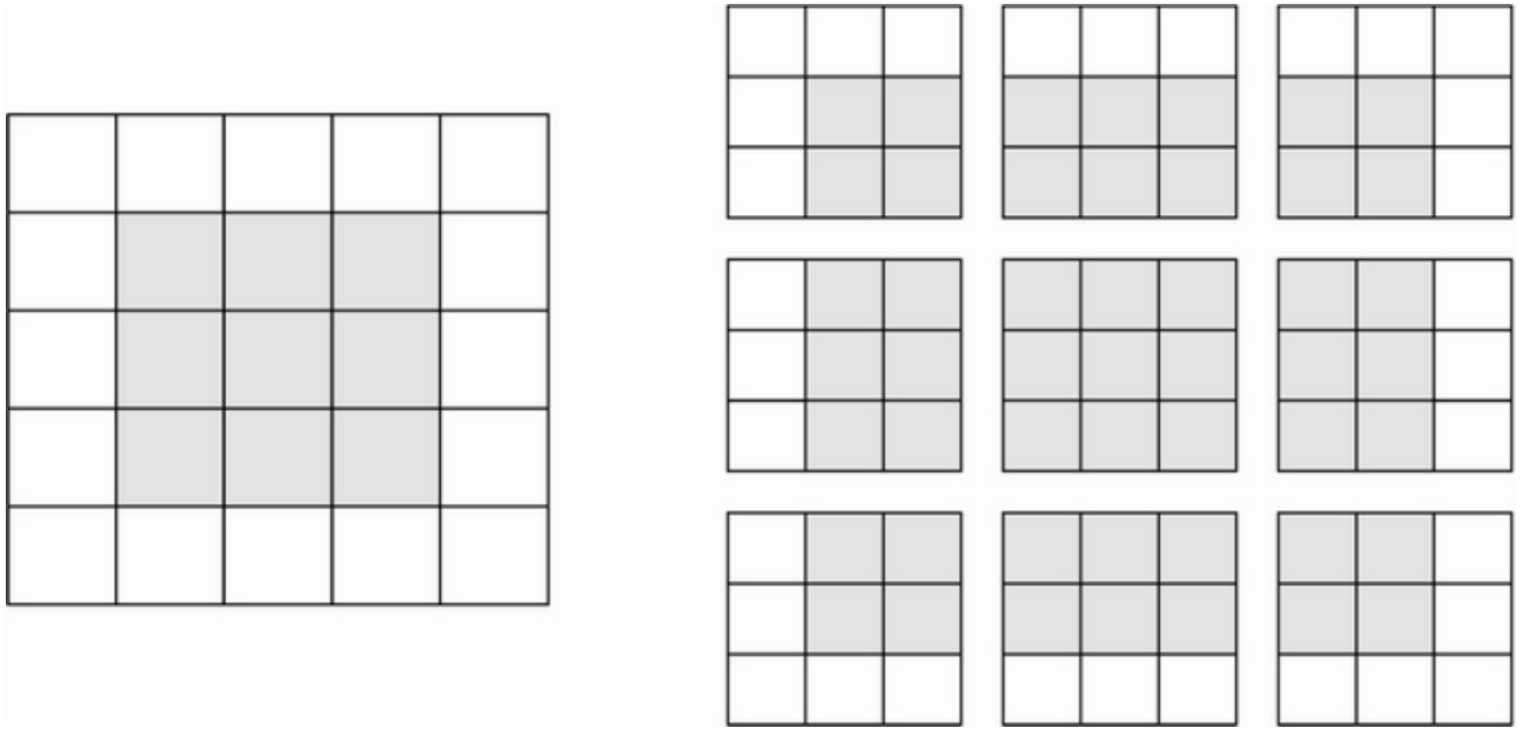
- A convolution works by sliding these windows of size 3 × 3 or 5 × 5 over the 3D input feature map, stopping at every possible location, and extracting the 3D patch of surrounding features (shape (window_height, window_width, input_depth)). Each such 3D patch is then transformed into a 1D vector of shape (output_depth,), which is done via a tensor product with a learned weight matrix, called the "convolution kernel—the same kernel is reused across every patch.



The convolution operation

Border effects

Consider a 5 × 5 feature map (25 tiles total). There are only 9 tiles around which you can center a 3 × 3 window, forming a 3 × 3 grid. Hence, the output feature map will be 3 × 3. It shrinks a little: by exactly two tiles alongside each dimension, in this case.

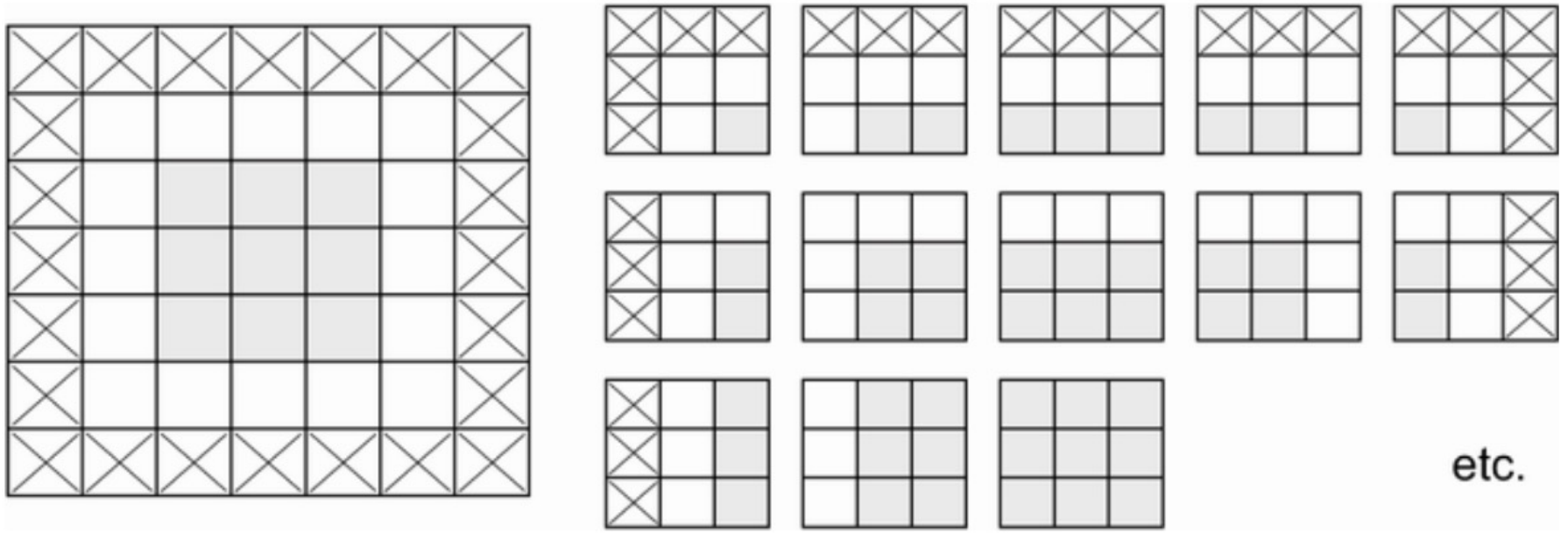


Valid locations of 3 × 3 patches in a 5 × 5 input feature map

The convolution operation

Padding

Padding consists of adding an appropriate number of rows and columns on each side of the input feature map so as to make it possible to fit center convolution windows around every input tile.”

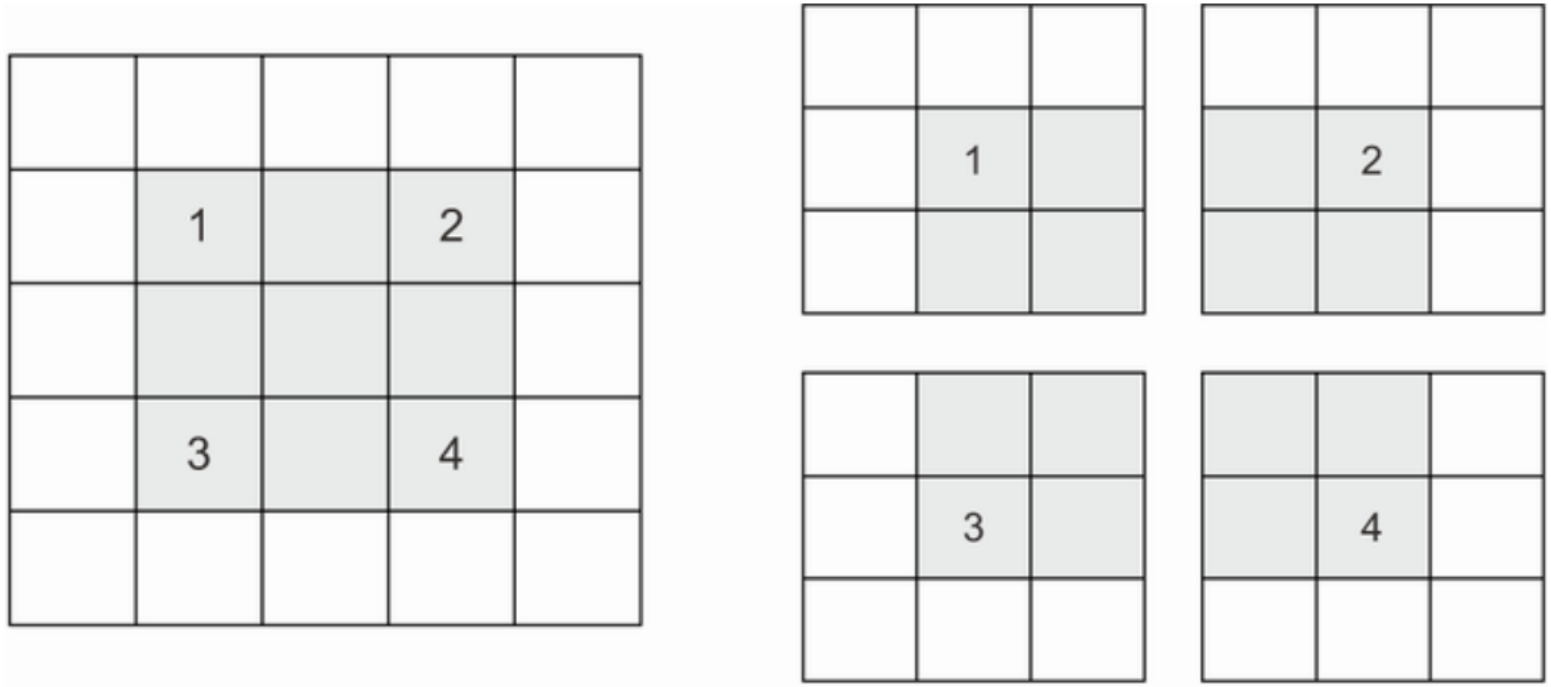


Padding a 5 × 5 input in order to be able to extract 25 3 × 3 patches

The convolution operation

Convolution 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: convolutions with a stride higher than 1.



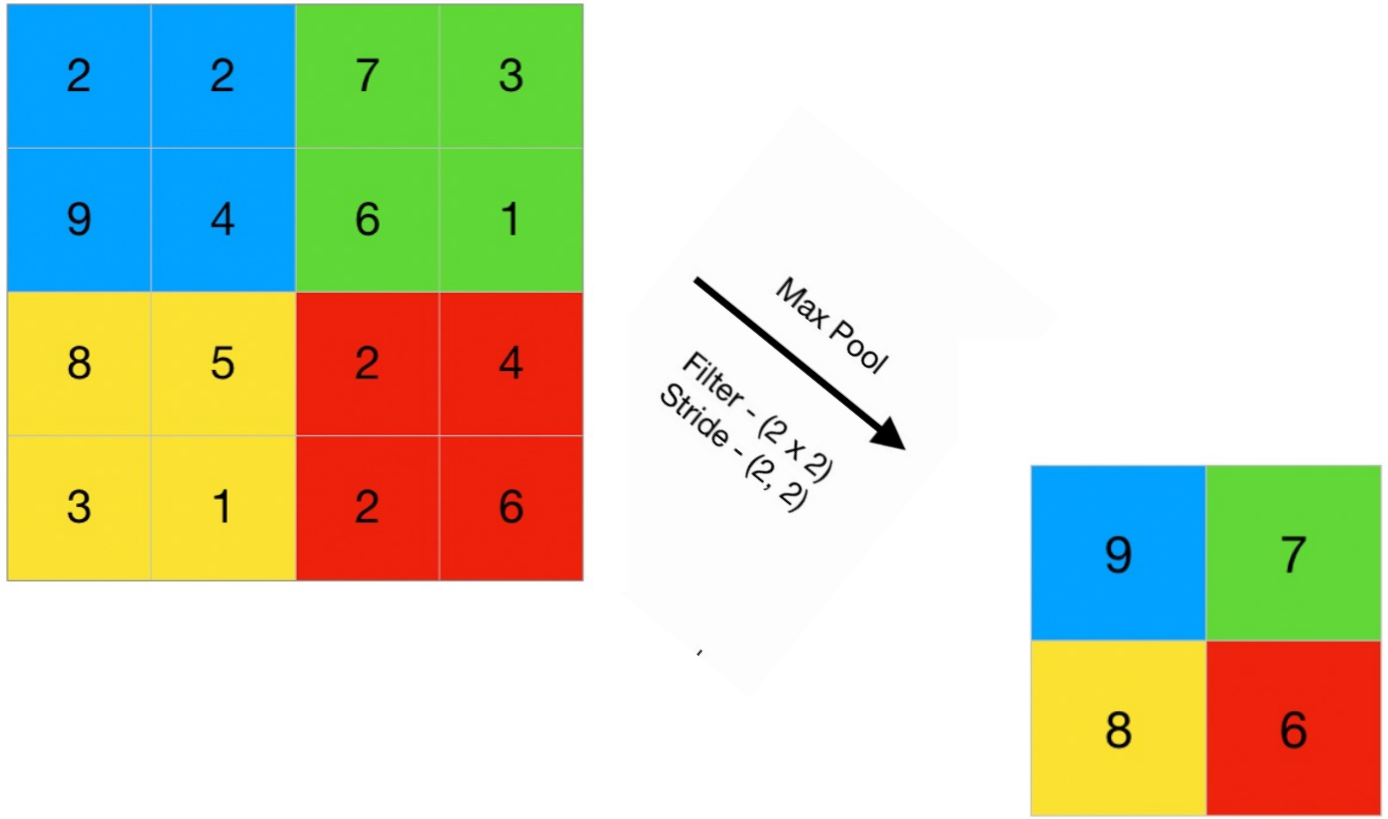
3 × 3 convolution patches with 2 × 2 strides

The max-pooling operation

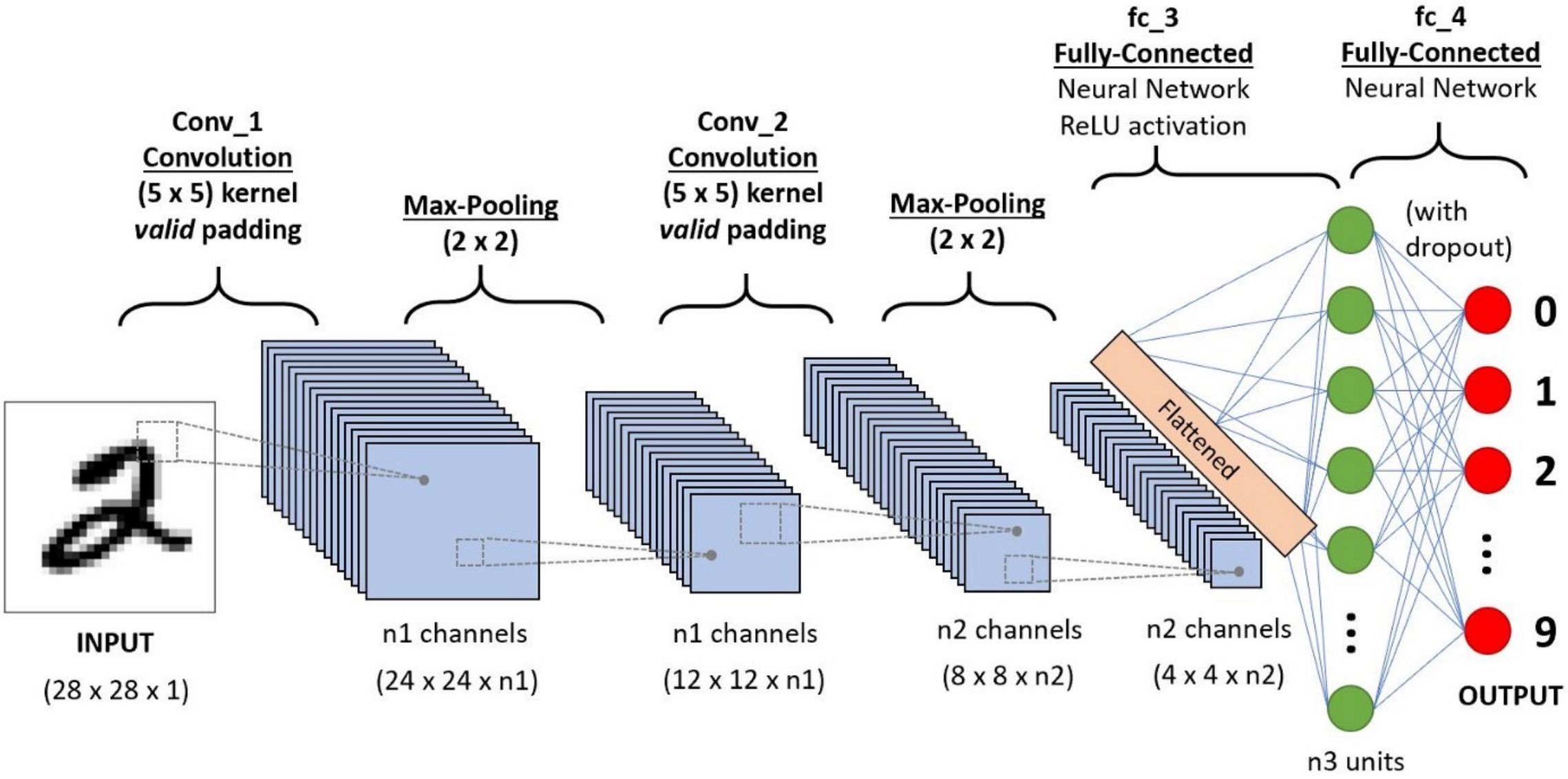
The role of **max pooling** is to aggressively downsample feature maps

Max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel.

It's conceptually similar to convolution, except that instead of transforming local patches via a learned *linear transformation* (the convolution kernel), they're transformed via a hardcoded max tensor operation.



Convolutional Neural Networks (convnets)

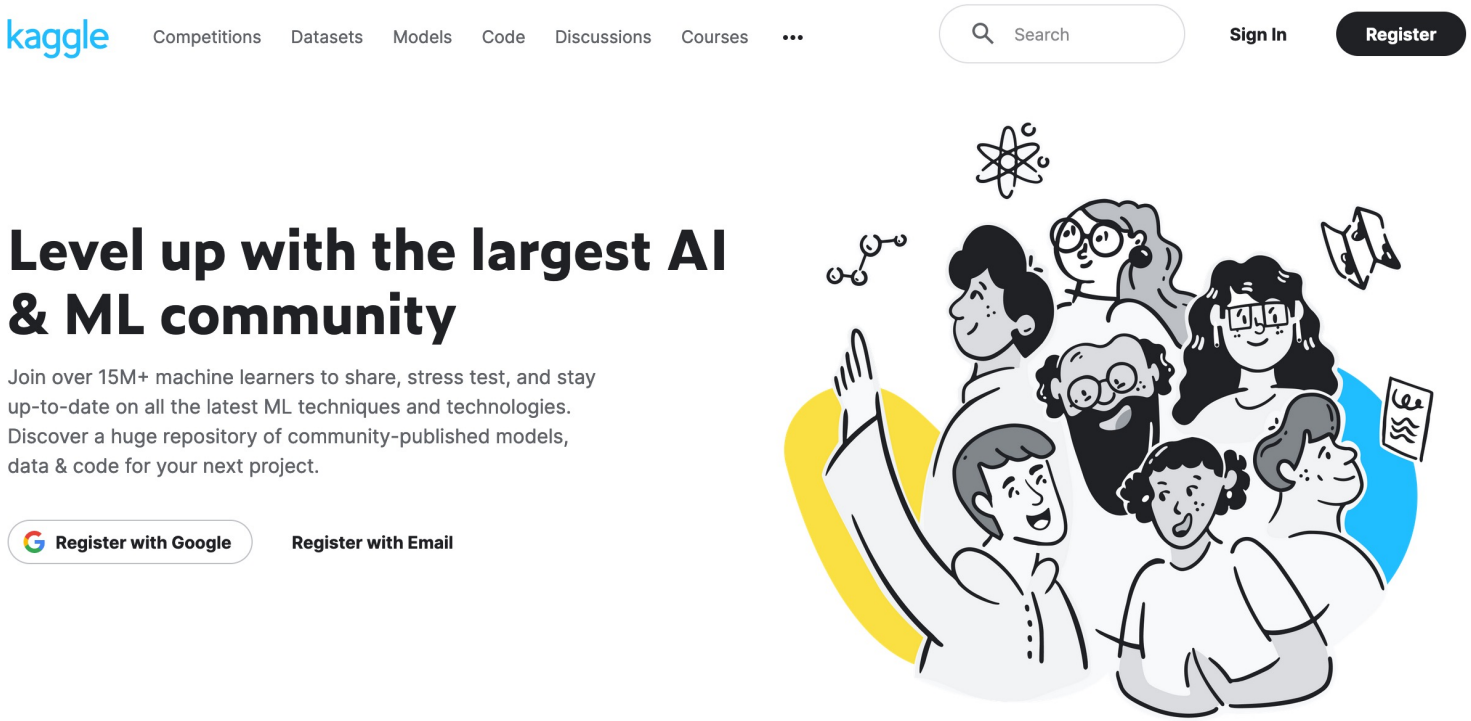


Training a convnet from scratch
on a small dataset

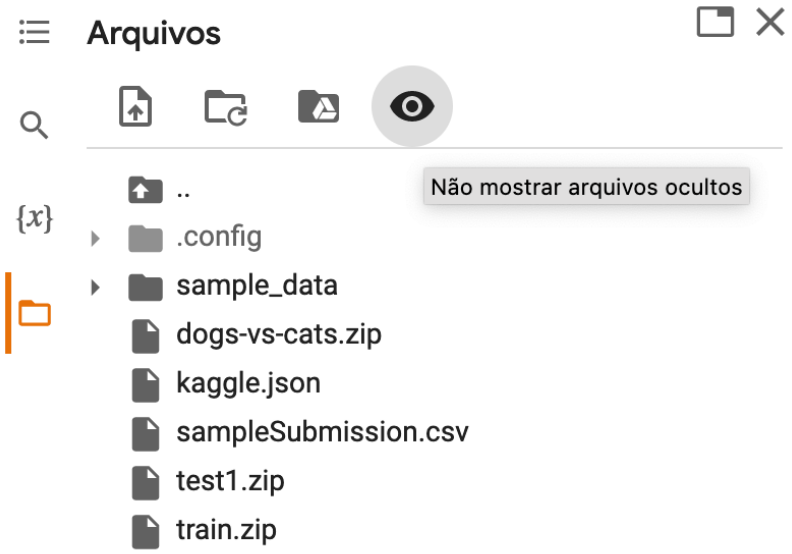
Practical example: classifying images as dogs or cats in a dataset containing **5,000** pictures of cats and dogs (**2,500** cats, **2,500** dogs). We'll use 2,000 pictures for training, **1,000** for validation, and **2,000** for testing.

To download the data you will need to create a Kaggle accout

https://www.kaggle.com/



Linux Virtual Machine - *Colab*




```
!ls -lh
```

```
total 1.6G
-rw-r--r-- 1 root root 813M Dec 11  2019 dogs-vs-cats.zip
-rw-r--r-- 1 root root   73 Oct 10 14:26 kaggle.json
drwxr-xr-x 1 root root 4.0K Oct  6 13:25 sample_data
-rw-r--r-- 1 root root  87K Dec 11  2019 sampleSubmission.csv
-rw-r--r-- 1 root root 272M Dec 11  2019 test1.zip
-rw-r--r-- 1 root root 544M Dec 11  2019 train.zip
```

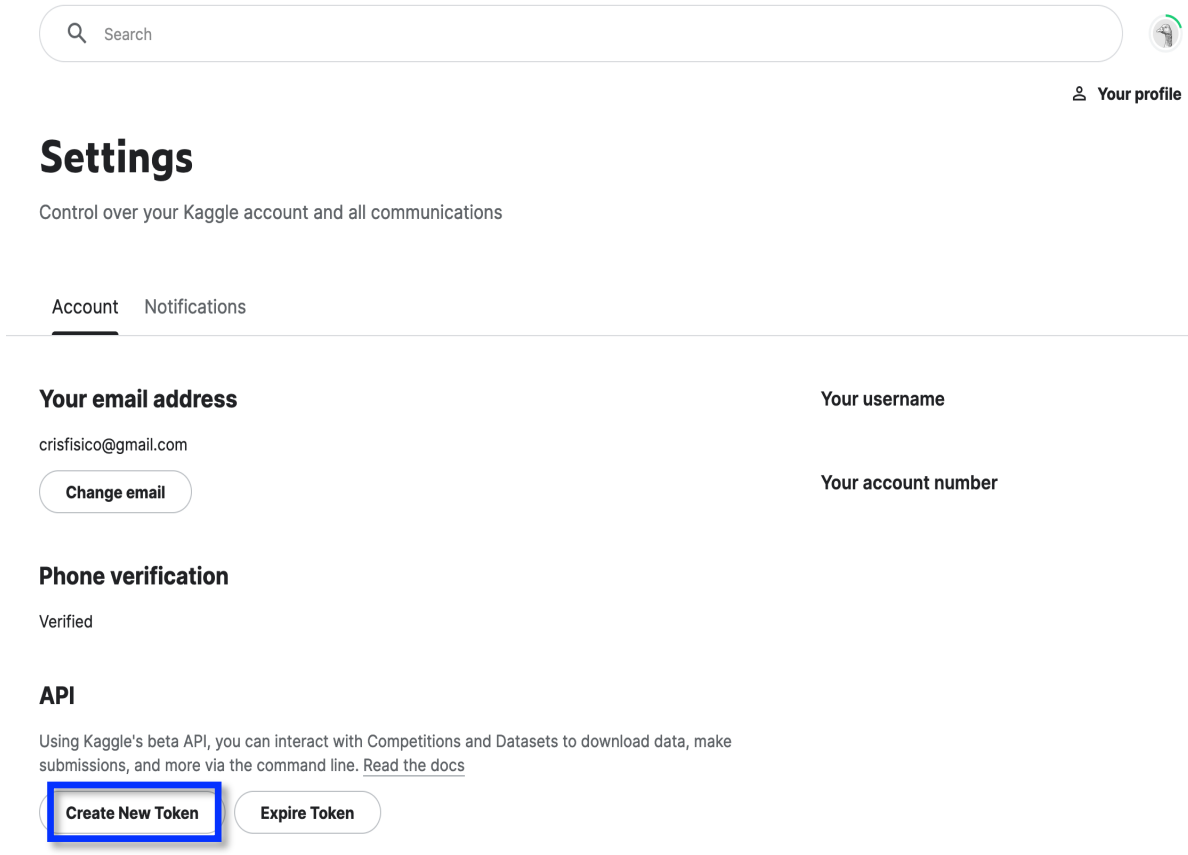
```
!ls -lha ~/
```

```
total 72K
drwx------ 1 root root 4.0K Oct 10 14:25 .
drwxr-xr-x 1 root root 4.0K Oct 10 14:07 ..
-r-xr-xr-x 1 root root 1.2K Jan  1  2000 .bashrc
drwxr-xr-x 1 root root 4.0K Oct  6 13:58 .cache
drwx------ 4 root root 4.0K Oct  6 13:57 .config
drwxr-xr-x 5 root root 4.0K Oct  6 13:42 .ipython
drwx------ 1 root root 4.0K Oct  6 13:42 .jupyter
drwxr-xr-x 2 root root 4.0K Oct 10 14:26 .kaggle
drwxr-xr-x 2 root root 4.0K Oct  6 13:39 .keras
drwx------ 3 root root 4.0K Oct  6 13:15 .launchpadlib
drwxr-xr-x 1 root root 4.0K Oct  6 13:42 .local
drwxr-xr-x 4 root root 4.0K Oct  6 13:56 .npm
-rw-r--r-- 1 root root  161 Jul  9  2019 .profile
-r-xr-xr-x 1 root root  254 Jan  1  2000 .tmux.conf
-rw-r--r-- 1 root root  165 Oct  6 13:56 .wget-hsts
```

Kaggle account

Access to the API is restricted to Kaggle users, so in order to download the data, you first need to authenticate yourself. The kaggle package will look for your login credentials in a JSON file located at ~/.kaggle/kaggle.json

Step 1



Step 2

```
from google.colab import files
files.upload()
```

Procurar... kaggle.json

Step 3

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Step 4

```
!kaggle competitions download -c dogs-vs-cats
```

```
Downloading dogs-vs-cats.zip to /content
 99% 800M/812M [00:06<00:00, 176MB/s]
100% 812M/812M [00:06<00:00, 129MB/s]
```

Step 5

```
!unzip -qq dogs-vs-cats.zip
```

```
!unzip -qq train.zip
```