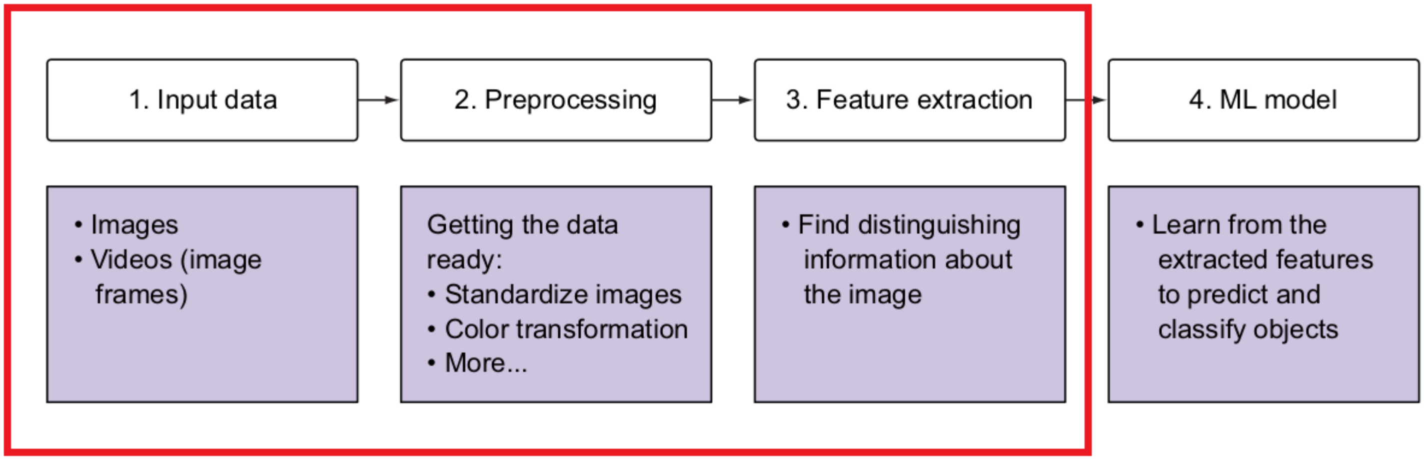
Computer Vision

Lecture 10-11: Convolutional Neural Networks

Yuriy Kochura
iuriy.kochura@gmail.com
@y_kochura

The computer vision pipeline

Last time



—
Credits: Mohamed Elgendy. Deep Learning for Vision Systems, 2020.

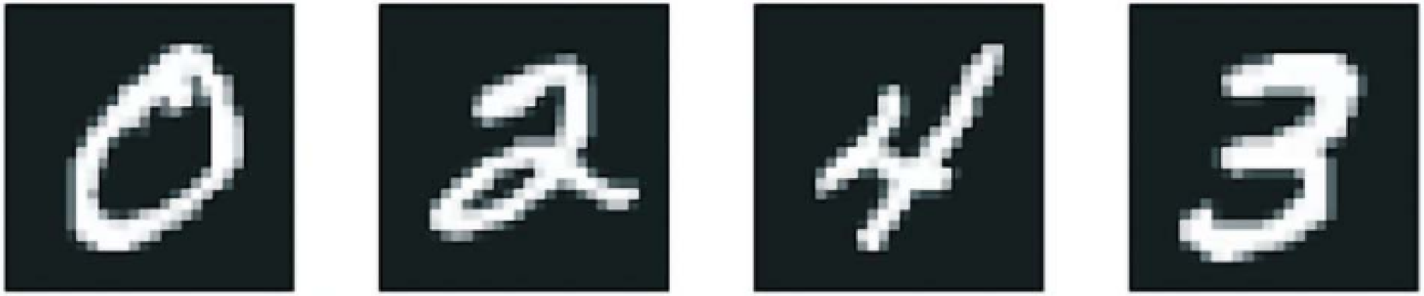
Today

Understanding convolutional neural networks (convnets)

- Fully connected NNs
- Convolutional NNs:
  - The convolution operation
  - Understanging border effects
  - Understanging padding
  - Understanging convolution strides
  - Understanging max-pooling operation

Fully connected NNs

MNIST sample digits



Note! In machine learning, a category in a classification problem is called a class. Data points are called samples. The class associated with a specific sample is called a label.

—
Credits: François Chollet. Deep Learning with Python, 2021.

Loading the MNIST dataset in Keras

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

train_images and train_labels form the training set, the data that the model will learn from. The model will then be tested on the test set, test_images and test_labels.

—
Credits: François Chollet. Deep Learning with Python, 2021.

Train and Test data

```
>>> train_images.shape
(60000, 28, 28)
>>> len(train_labels)
60000
>>> train_labels
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

And here's the test data:

```
>>> test_images.shape
(10000, 28, 28)
>>> len(test_labels)
10000
>>> test_labels
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

—
Credits: François Chollet. Deep Learning with Python, 2021.

The network architecture

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

The workflow will be as follows: First, we'll feed the neural network the training data, train_images and train_labels. The network will then learn to associate images and labels. Finally, we'll ask the network to produce predictions for test_images, and we'll verify whether these predictions match the labels from test_labels.

—

Credits: François Chollet. Deep Learning with Python, 2021.

Before training a model

To make the model ready for training, we need to pick three more things as part of the compilation step:

- An optimizer — The mechanism through which the model will update itself based on the training data it sees, so as to improve its performance.
- A loss function — How the model will be able to measure its performance on the training data, and thus how it will be able to steer itself in the right direction.
- Metrics to monitor during training and testing — Here, we'll only care about accuracy (the fraction of the images that were correctly classified).

—
Credits: François Chollet. Deep Learning with Python, 2021.

The compilation step

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

Preparing the image data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

Previously, our training images were stored in an array of shape $(60000, 28, 28)$ of type uint8 with values in the $[0, 255]$ interval. We'll transform it into a float32 array of shape $(60000, 28 * 28)$ with values between 0 and 1.

“Fitting” the model

```
>>> model.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [===========================] - 5s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [=====================>.....] - ETA: 1s - loss: 0.1035 - acc: 0.9692
```

—
Credits: François Chollet. Deep Learning with Python, 2021.

Using the model to make predictions

```
>>> test_digits = test_images[0:10]
>>> predictions = model.predict(test_digits)
>>> predictions[0]
array([1.0726176e-10, 1.6918376e-10, 6.1314843e-08, 8.4106023e-06,
       2.9967067e-11, 3.0331331e-09, 8.3651971e-14, 9.9999106e-01,
       2.6657624e-08, 3.8127661e-07], dtype=float32)
```

Each number of index i in that array corresponds to the probability that digit image test_digits[0] belongs to class i.

—
Credits: François Chollet. Deep Learning with Python, 2021.

This first test digit has the highest probability score (0.99999106, almost 1) at index 7, so according to our model, it must be a 7:

```
>>> predictions[0].argmax()
7
>>> predictions[0][7]
0.99999106
```

We can check that the test label agrees:

```
>>> test_labels[0]
7
```

—
Credits: François Chollet. Deep Learning with Python, 2021.

Evaluating the model on new data

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"test_acc: {test_acc}")
test_acc: 0.9785
```

—

Credits: François Chollet. Deep Learning with Python, 2021.

Convolutional NNs

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)
```

Importantly, a convnet takes as input tensors of shape (image_height, image_width, image_channels), not including the batch dimension. In this case, we'll configure the convnet to process inputs of size $(28, 28, 1)$, which is the format of MNIST images.

—
Credits: François Chollet. Deep Learning with Python, 2021.

Displaying the model's summary

```
>>> 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 (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 10)                11530
=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

—

Credits: François Chollet. Deep Learning with Python, 2021.

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)
```

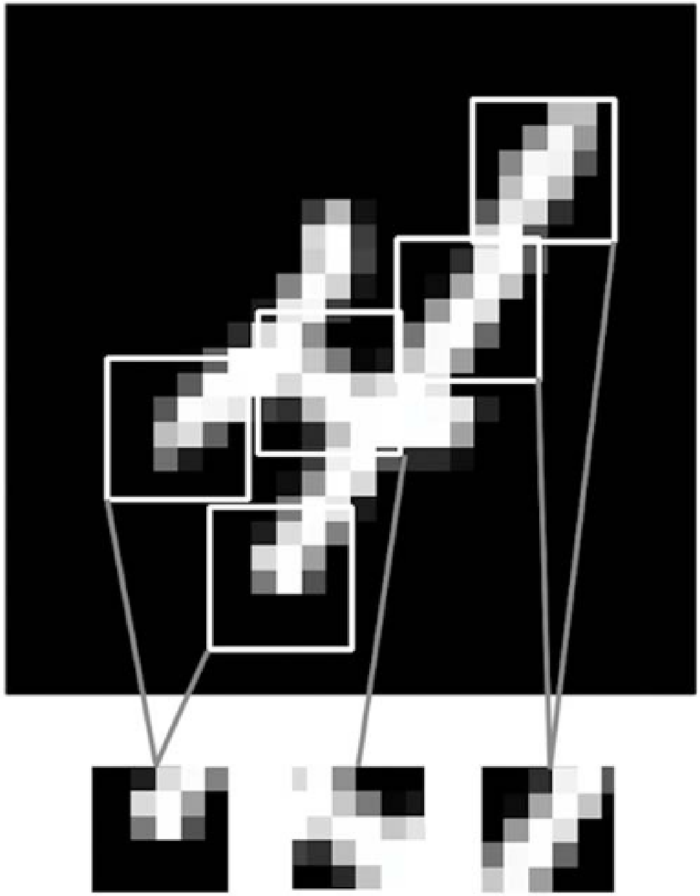
—
Credits: François Chollet. Deep Learning with Python, 2021.

Evaluating the convnet

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"Test accuracy: {test_acc:.3f}")
Test accuracy: 0.991
```

—
Credits: François Chollet. Deep Learning with Python, 2021.

The convolution operation



—
Credits: François Chollet. Deep Learning with Python, 2021.

This key characteristic (convolution layers learn local patterns) gives convnets two interesting properties:

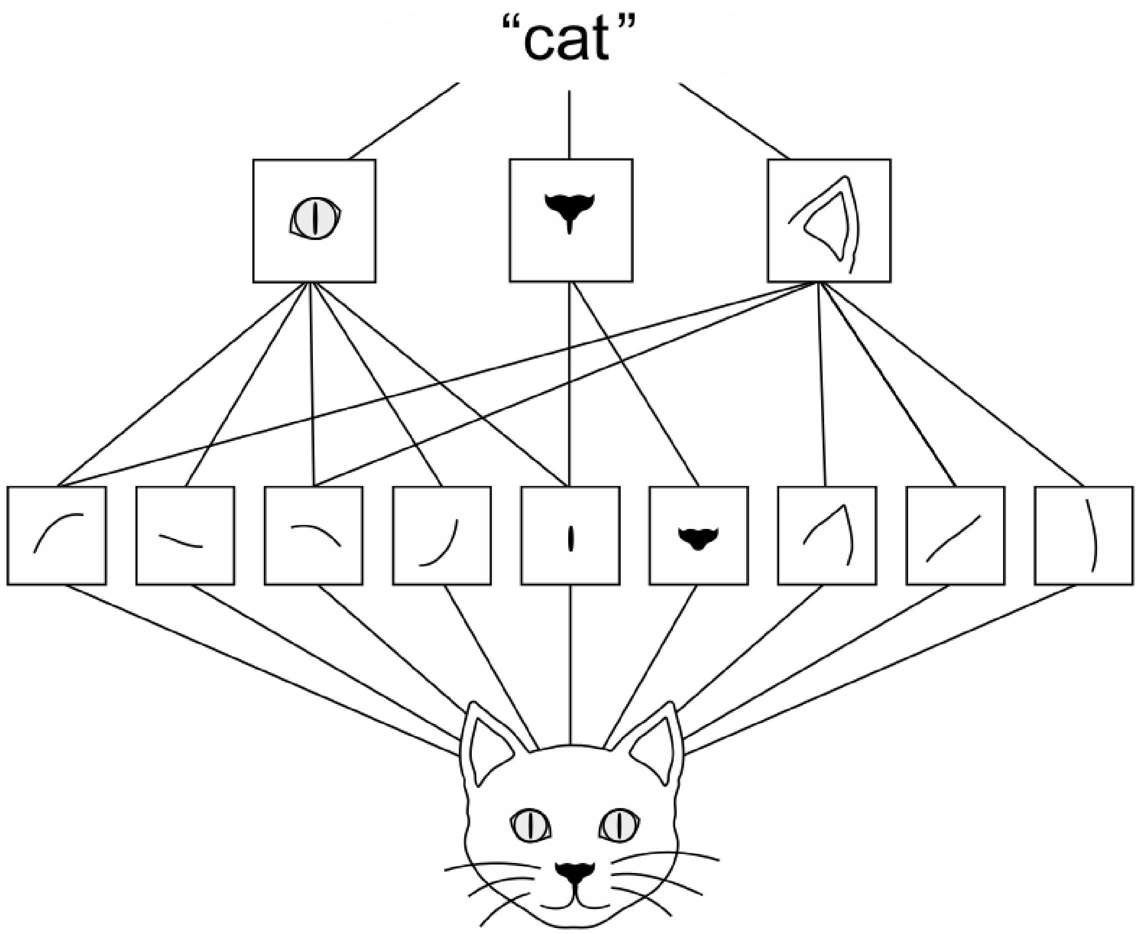
- The patterns they learn are translation-invariant. After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere: for example, in the upper-left corner. A densely connected model would have to learn the pattern anew if it appeared at a new location. This makes convnets data-efficient when processing images (because the visual world is fundamentally translation-invariant): they need fewer training samples to learn representations that have generalization power.

- 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 (see figure on next slide). This allows convnets to efficiently learn increasingly complex and abstract visual concepts, because the visual world is fundamentally spatially hierarchical.

—

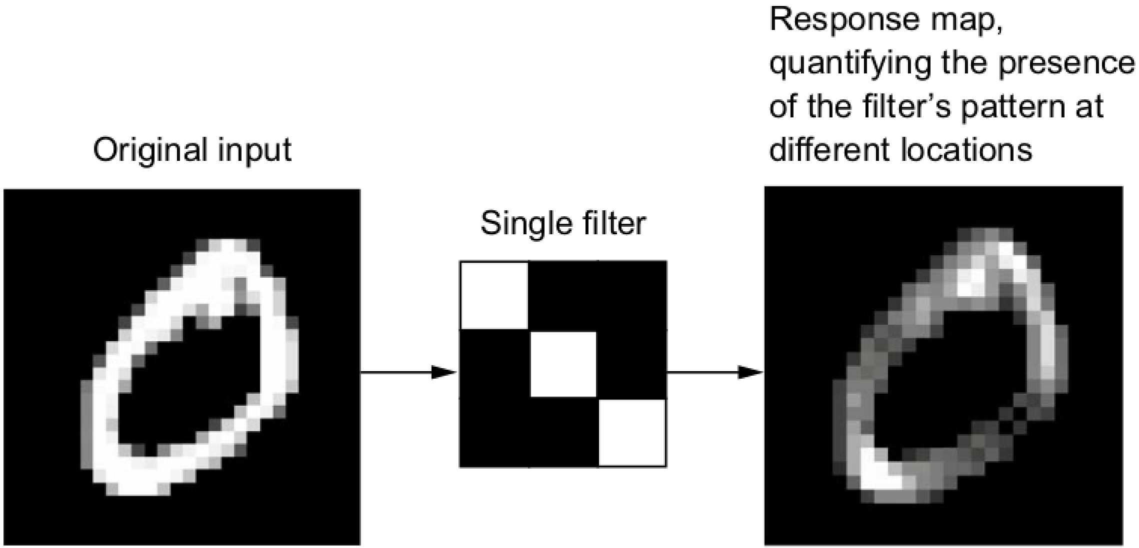
Credits: François Chollet. Deep Learning with Python, 2021.

The convolution operation



—
Credits: François Chollet. Deep Learning with Python, 2021.

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



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 (see figure above).

—

Credits: François Chollet. Deep Learning with Python, 2021.

Demo

How convolution works in a convolutional layer?

Convolutions are defined by two key parameters:

- Size of the patches extracted from the inputs — These are typically 3×3 or 5×5. In the example, they were 3×3, which is a common choice.
- 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.

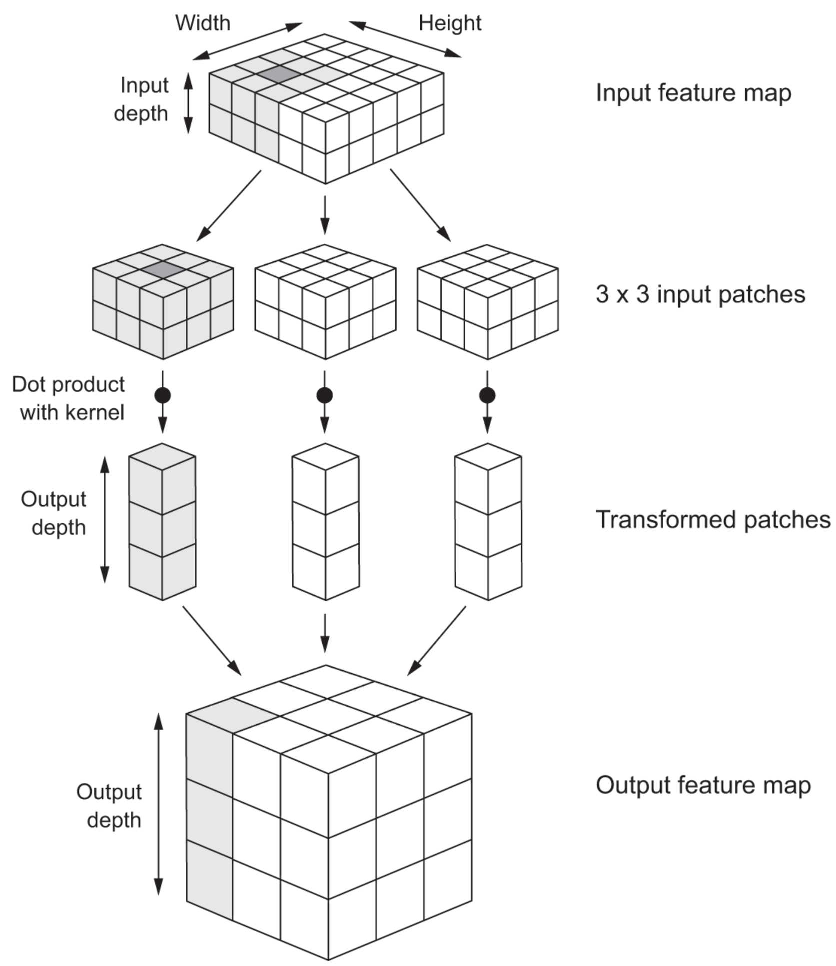
—
Credits: François Chollet. Deep Learning with Python, 2021.

In Keras Conv2D layers, these parameters (Size of the patches extracted from the inputs, Depth of the output feature map) are the first arguments passed to the layer:

- Conv2D(output_depth, (window_height, window_width))

How convolution works

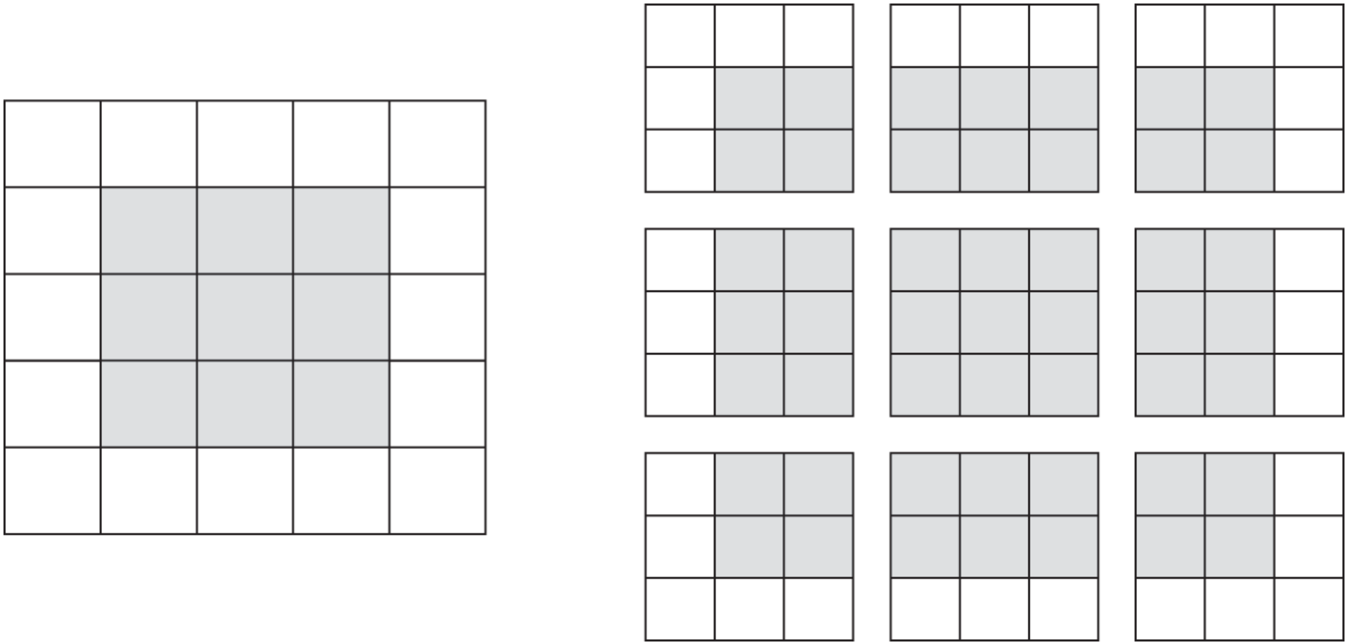
For instance, with 3×3 windows, the vector $output[i, j, :]$ comes from the 3D patch $input[i-1 : i+1, j-1 : j+1, :]$. The full process is detailed in figure below.



—
Credits: François Chollet. Deep Learning with Python, 2021.

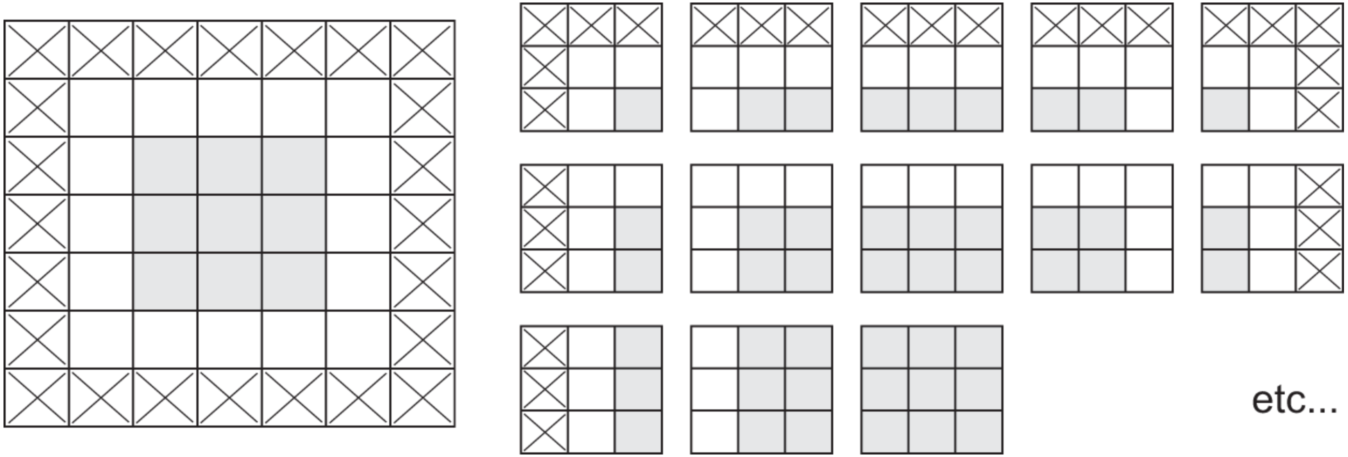
Understanging border effects

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



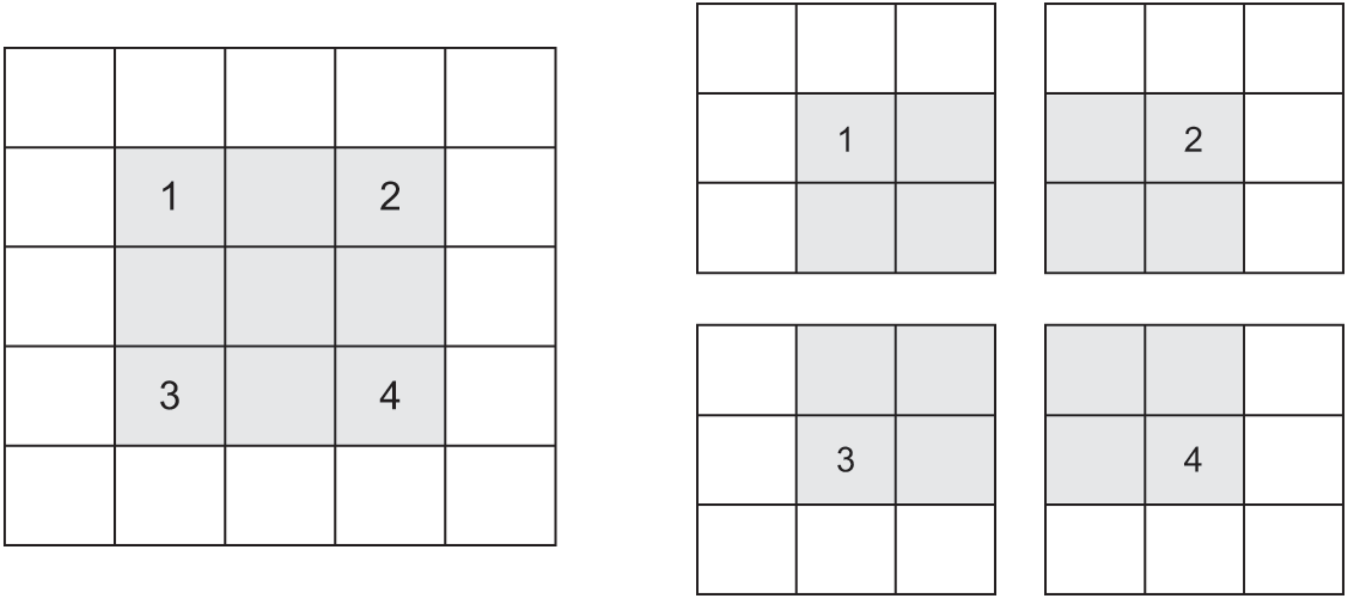
Understanging padding

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



Understanging convolution strides

3×3 convolution patches with 2×2 strides



Credits: François Chollet. Deep Learning with Python, 2021.

Understanging max-pooling operation

In the convnet example, you may have noticed that the size of the feature maps is halved after every MaxPooling2D layer. For instance, 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, much like strided convolutions.

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.

—

Credits: François Chollet. Deep Learning with Python, 2021.

Understanging max-pooling operation

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model_no_max_pool = keras.Model(inputs=inputs, outputs=outputs)
```

Here's a summary of the model:

```
>>> model_no_max_pool.summary()
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 128)       73856
_________________________________________________________________
flatten_1 (Flatten)          (None, 61952)             0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                619530
=================================================================
Total params: 712,202
Trainable params: 712,202
Non-trainable params: 0
_________________________________________________________________
```

—
Credits: François Chollet. Deep Learning with Python, 2021.

Demo

CNN vs FCN

The end