Convolutional Neural Networks (Covnets)

Prof. Gheith Abandah

References:

- *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly), 2017, 978-1-491-96229-9.
- François Chollet, *Deep Learning with Python*, Manning Pub. 2018

Introduction

- YouTube Video: *Convolutional Neural Networks (CNNs) explained* from Deeplizard

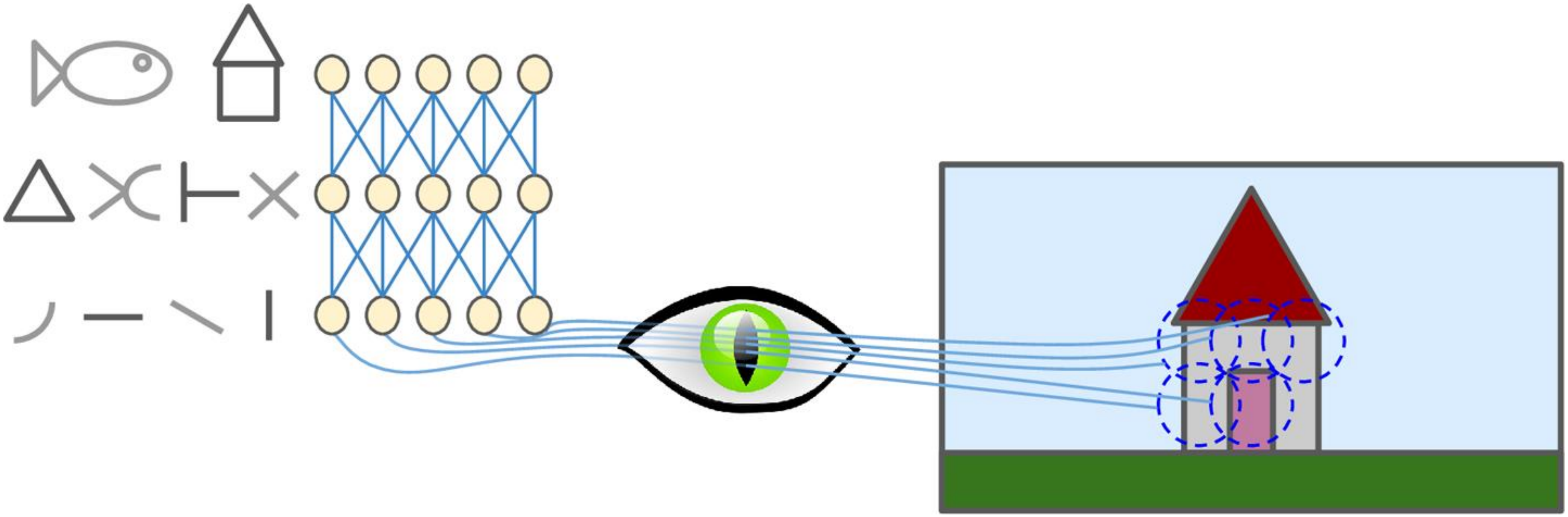
https://youtu.be/YRhxdVk_sIs

Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
3. Pooling layer
4. CNN architectures
5. Keras example
6. Exercises

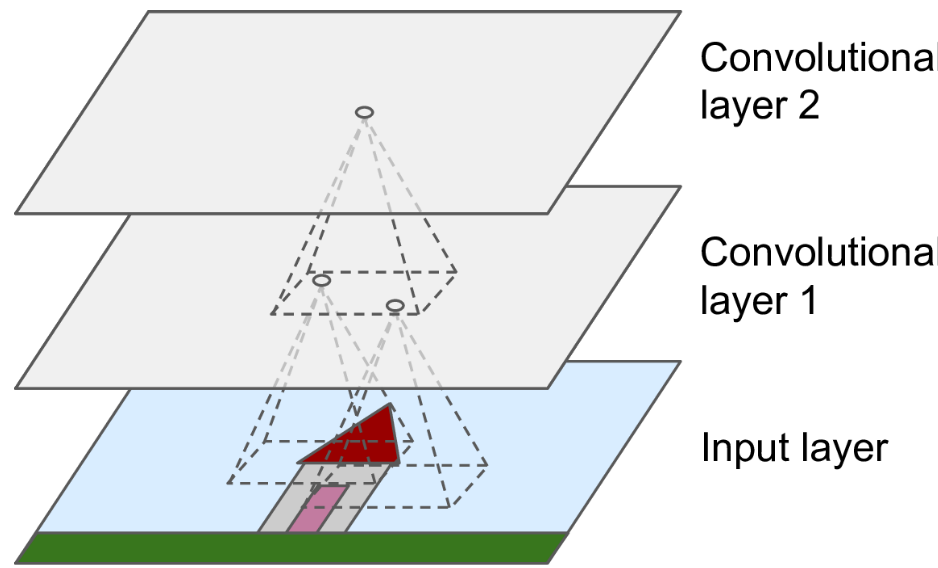
1. Introduction

- *Convolutional neural networks (CNNs)* emerged from the study of the brain's visual cortex.
- Many neurons in the visual cortex have a small *local receptive field.*



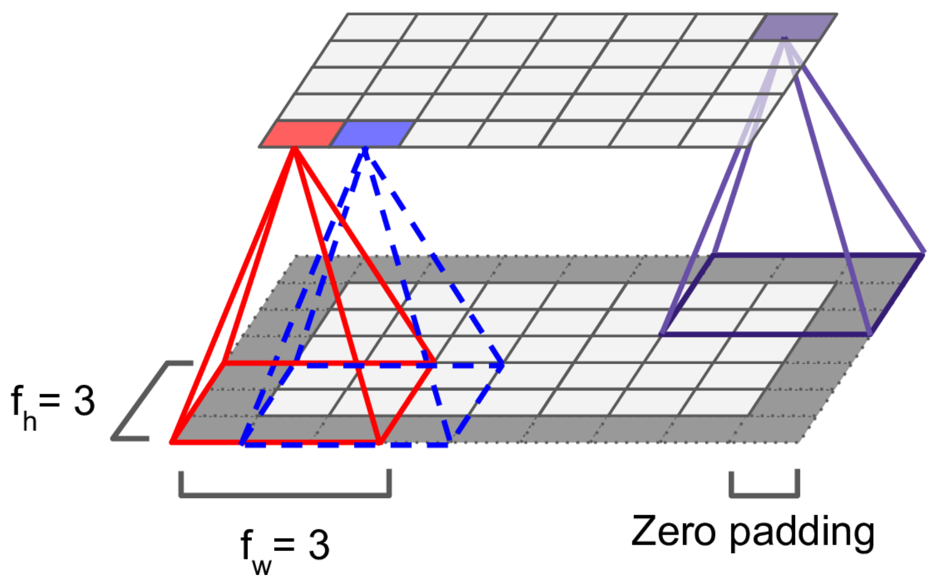
2. Convolutional Layer

- Neurons in one layer are not connected to every single pixel/neuron in the previous layer, but only to pixels/neurons in their receptive fields.
- This architecture allows the network to concentrate on low-level features in one layer, then assemble them into higher-level features in the next layer.
- Each layer is represented in 2D.



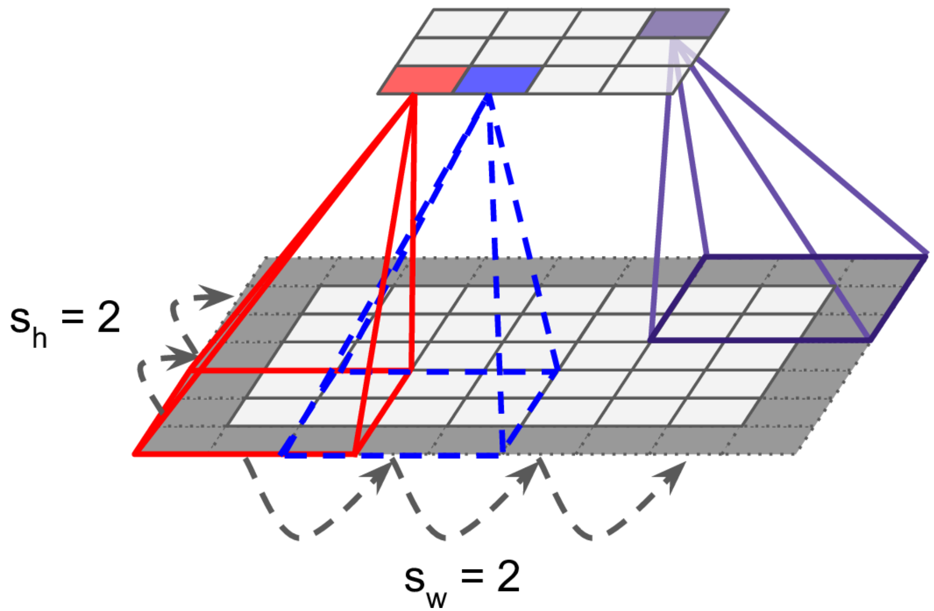
2. Convolutional Layer

- f_h and f_w are the height and width of the receptive field.
- *Zero padding:* In order for a layer to have the same height and width as the previous layer, it is common to add zeros around the inputs.
- Keras default is no padding



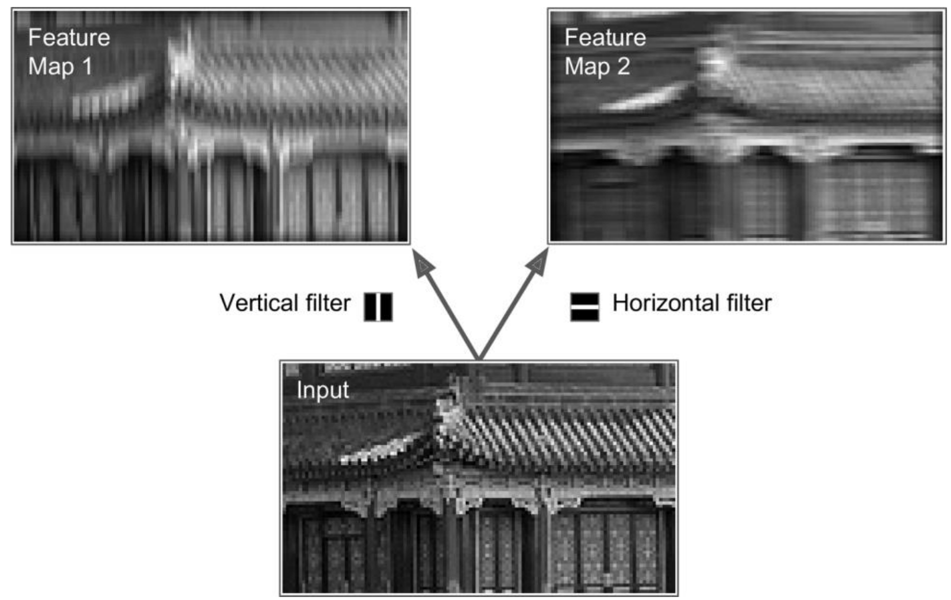
2. Convolutional Layer

- It is also possible to connect a large input layer to a smaller layer by spacing out the receptive fields.
- The distance between two consecutive receptive fields is called the *stride*.
- A neuron located in row i, column j is connected to the neurons in the previous layer located in:
  - Rows: $i \times s_h$ to $i \times s_h + f_h - 1$
  - Cols: $j \times s_w$ to $j \times s_w + f_w - 1$



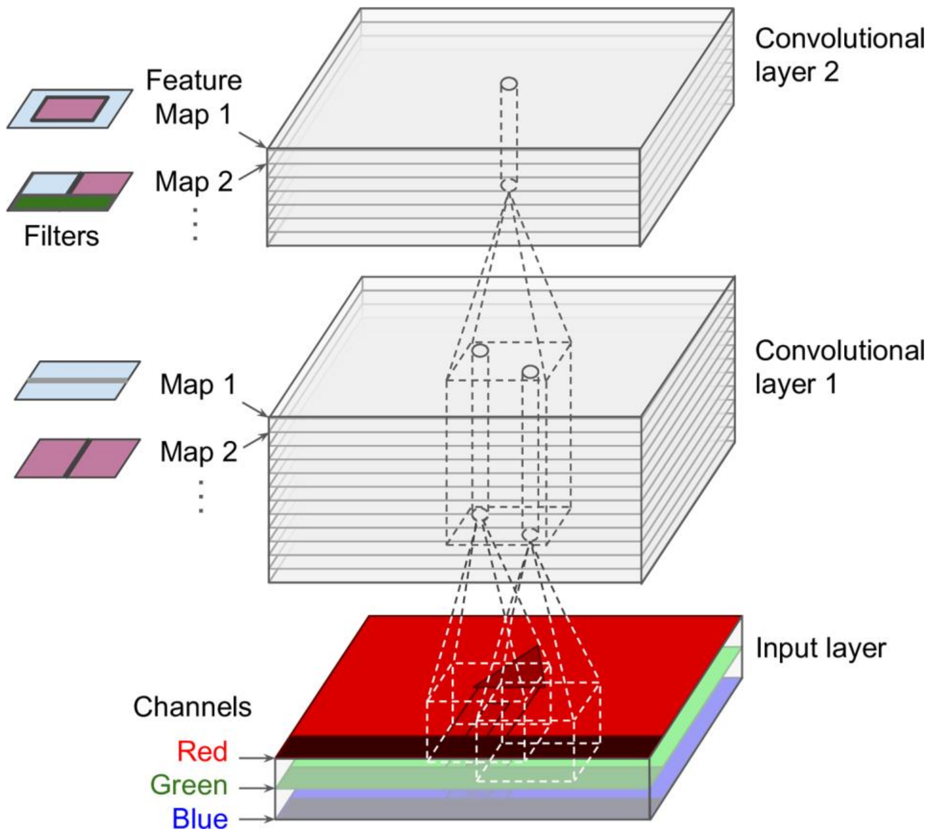
2.1. Filters

- A neuron's weights can be represented as a small image the size of the receptive field, called *filters*.
- When all neurons in a layer use the same line filters, we get the *feature maps* on the top.



2.2. Stacking Feature Maps

- In reality, each layer is *3D* composed of several feature maps of equal sizes.
- Within one feature map, all neurons share the same parameters, but different feature maps may have different parameters.
- Once the CNN has learned to recognize a pattern in one location, it can recognize it in any other location.



2.3. Mathematical Summary

Equation 13-1. Computing the output of a neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f_{n'}} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = u \cdot s_h + f_h - 1 \\ j' = v \cdot s_w + f_w - 1 \end{cases}$$

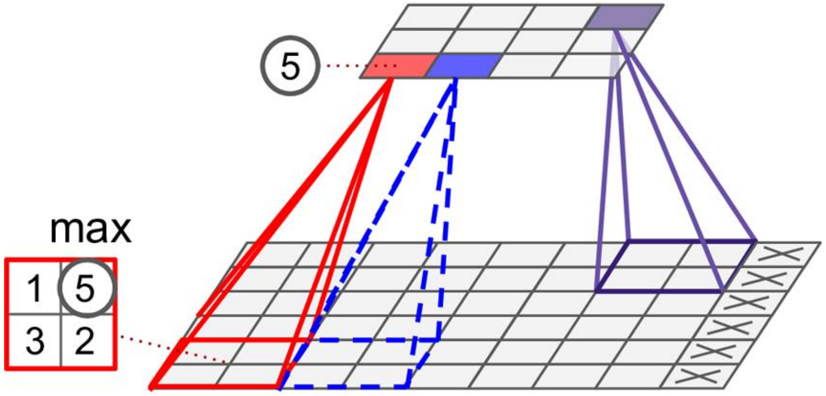
- $z_{i,j,k}$ is the output of the neuron located in row *i*, column *j* in feature map *k*
- $f_{n'}$ is the number of feature maps in the previous layer

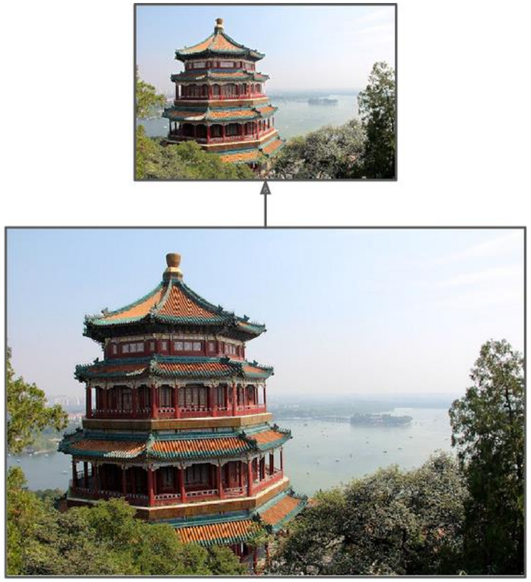
Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
3. Pooling layer
4. CNN architectures
5. Keras example
6. Exercises

3. Pooling Layer

- Its goal is to *subsample* (i.e., shrink) the input image in order to reduce the computational load, the memory usage, and the number of parameters.
- It aggregates the inputs using max or mean.



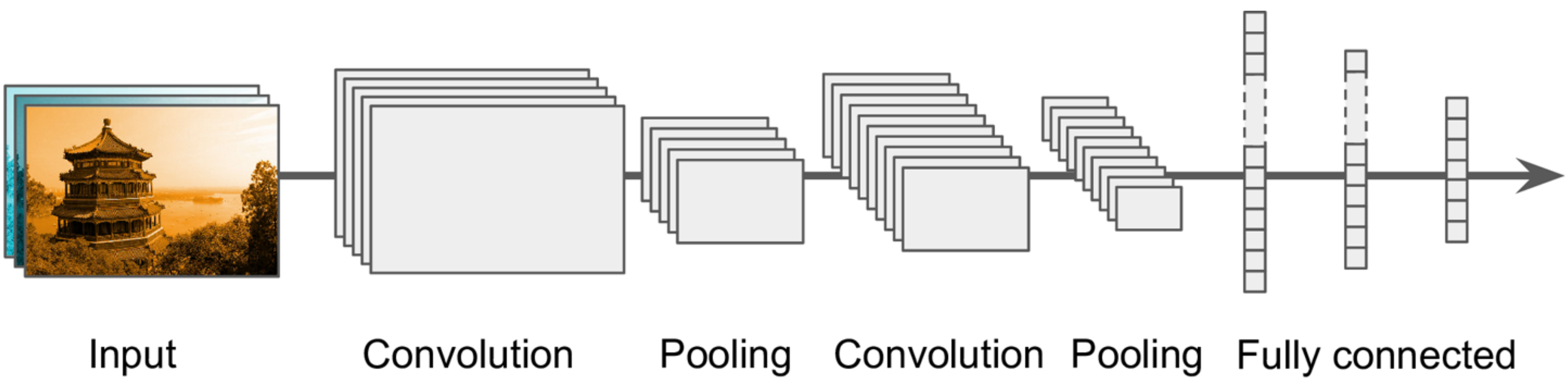


Outline

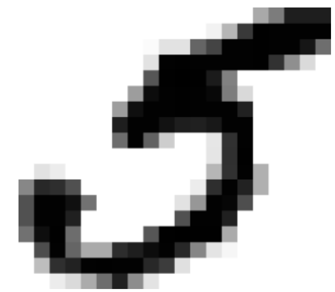
1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
3. Pooling layer
4. CNN architectures
5. Keras example
6. Exercises

4. CNN Architectures

- Stack few convolutional layers (each one generally followed by a ReLU layer), then a pooling layer, then another few convolutional layers, then another pooling layer, and so on. The image gets smaller and smaller, but it also gets deeper and deeper. At the end, a regular NN is added.



5. Keras Example - MNIST



```python
from keras import models
from keras import layers

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'))
```

32 feature maps

Filter size

2×2 window and stride 2

5. Keras Example

```
# add a classifier

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

5. Keras Example

```
>>> Model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (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
```

5. Keras Example

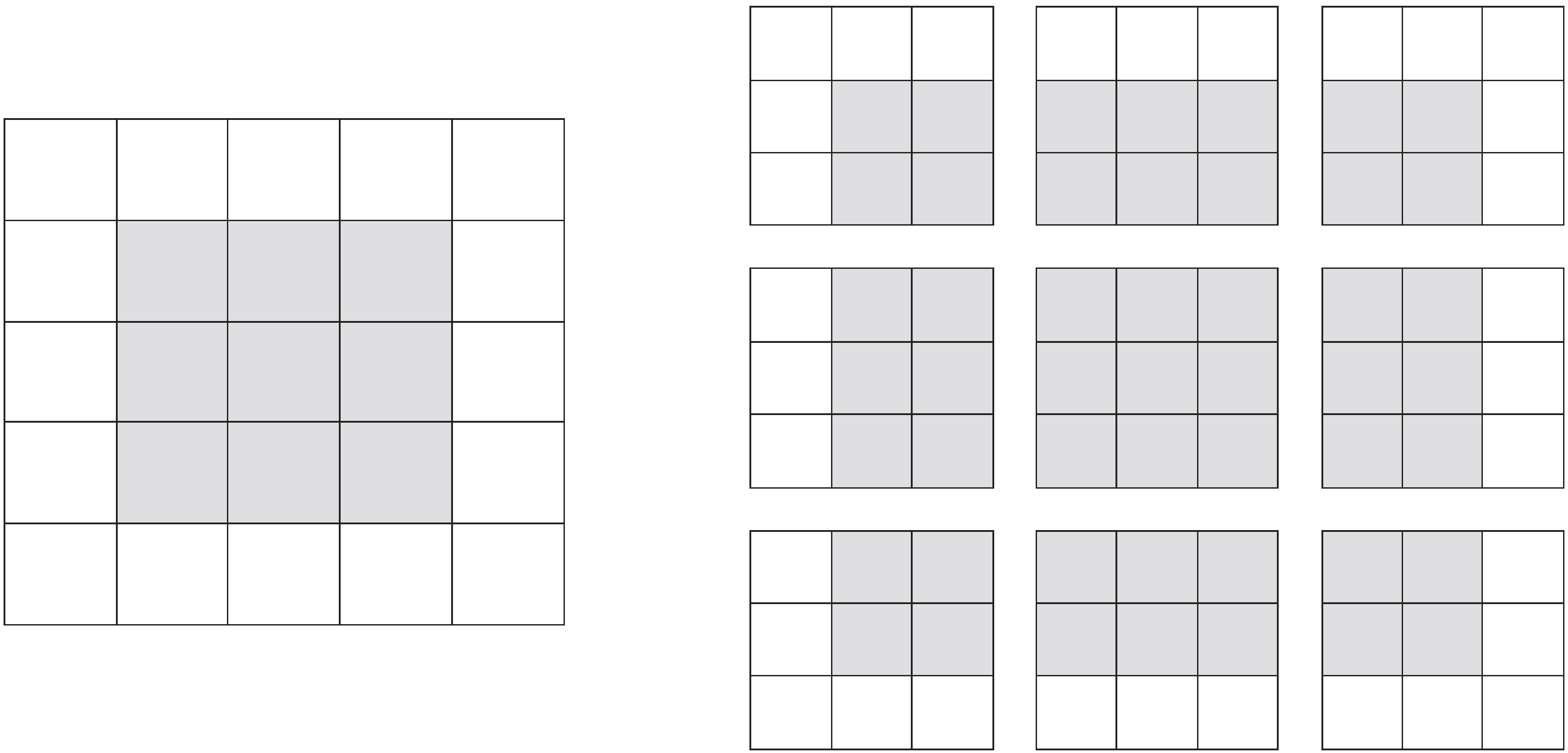


Figure 5.5 Valid locations of 3 × 3 patches in a 5 × 5 input feature map

5. Example – Prepare the data

```
from keras.datasets import mnist
(train_images, train_labels), (test_images,
      test_labels) = mnist.load_data()
#(60000, 28, 28), (6000), #(10000, 28, 28), (1000)
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

from keras.utils import to_categorical   #one hot
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

5. Keras Example

```
# Compile, train and evaluate

model.compile(optimizer='rmsprop',
       loss='categorical_crossentropy',
       metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5,
       batch_size=64)
…
Epoch 5/5
60000/60000 [===] - 7s - loss: 0.0187 - acc: 0.9943

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

0.9913

Summary

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
3. Pooling layer
4. CNN architectures
5. Keras example
6. Exercises

Exercises

From Chapter 13, solve exercises:

- 2
- 7