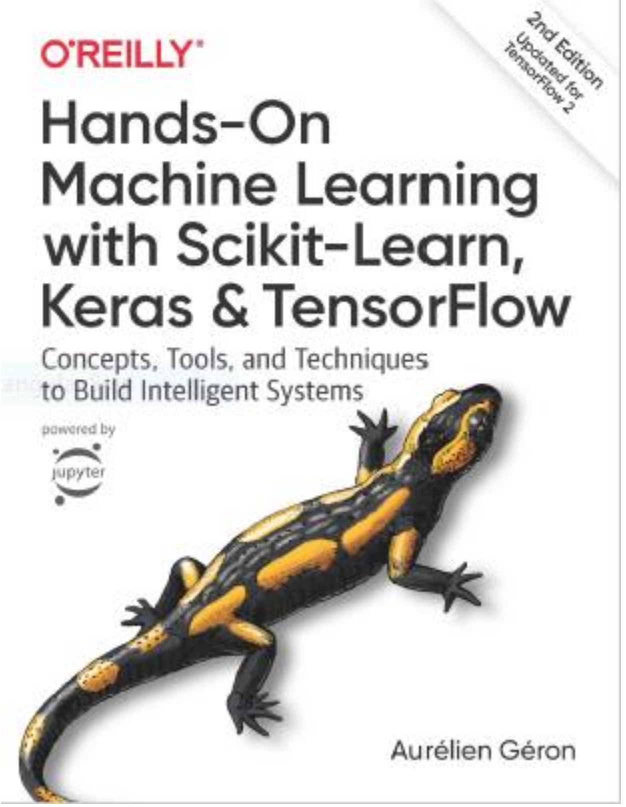
Deep Computer Vision Using Convolutional Neural Networks

Prof. Gheith Abandah

Reference

- Chapter 14: **Deep Computer Vision Using Convolutional Neural Networks**



O'REILLY

2nd Edition Updated for TensorFlow 2

Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow

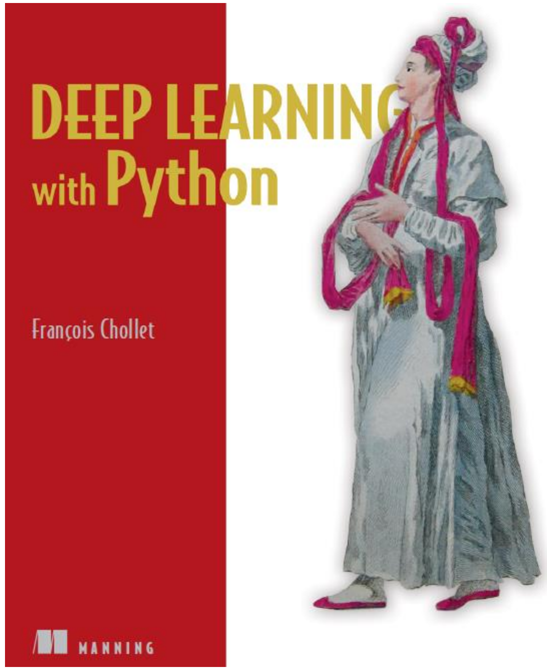
Concepts, Tools, and Techniques to Build Intelligent Systems

powered by jupyter

Aurélien Géron

- Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow**, O'Reilly, 2nd Edition, 2019
  - Material: https://github.com/ageron/handson-ml2

Reference



- **Deep Learning with Python**, by François Chollet, Manning Pub. 2018

Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises

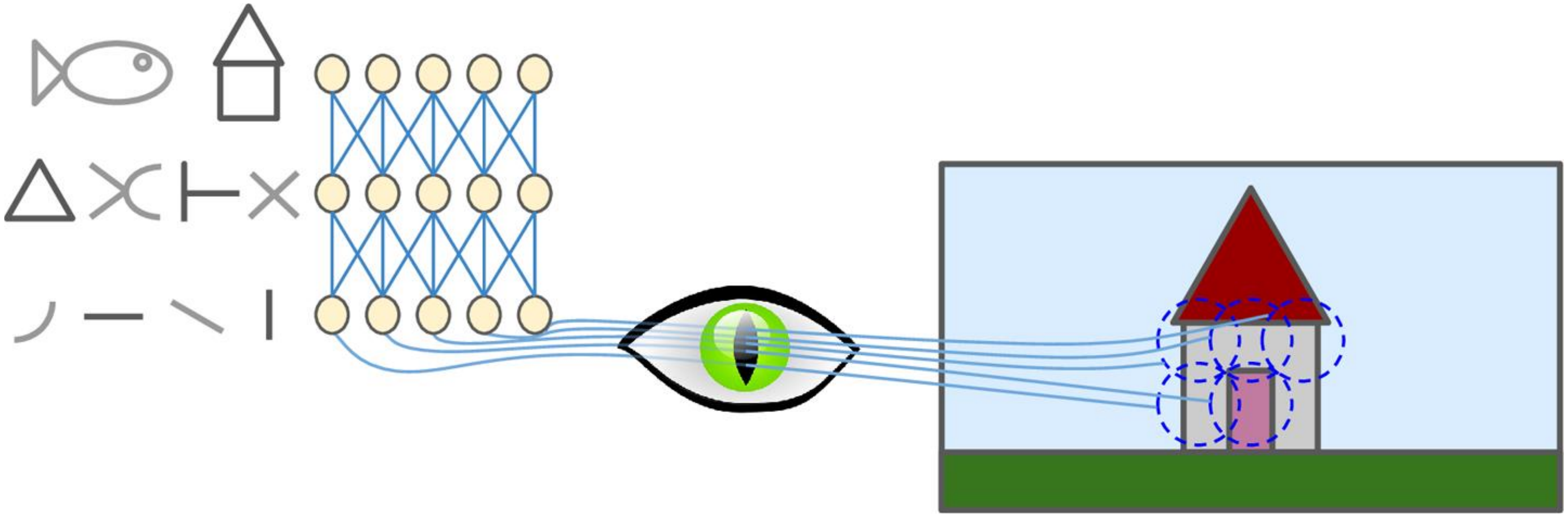
Introduction

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

https://youtu.be/YRhxdVk_sIs

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**.

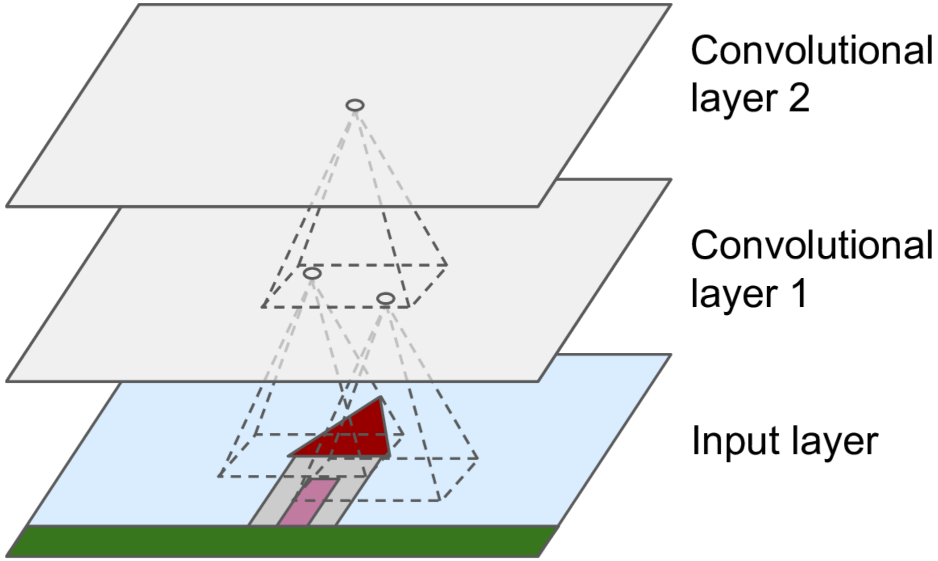


Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises

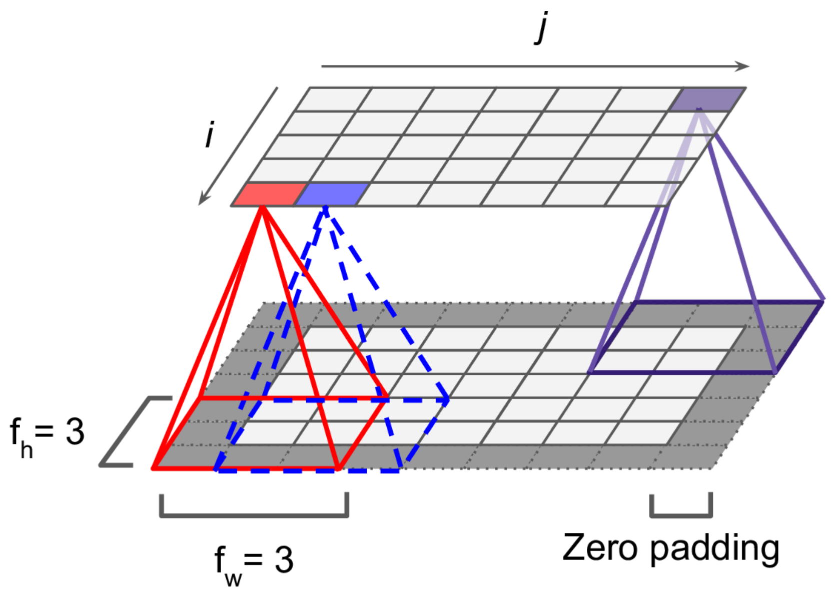
2. Convolutional Layer

- **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**.



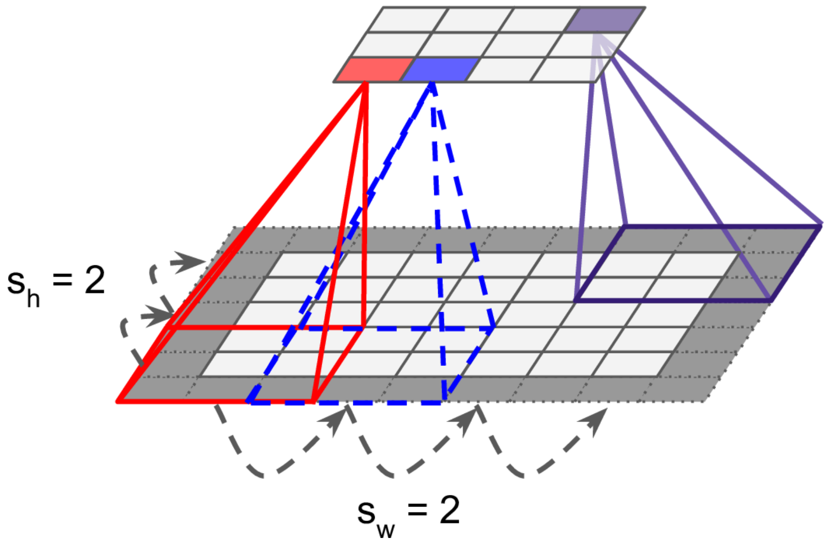
2. Convolutional Layer

- 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.



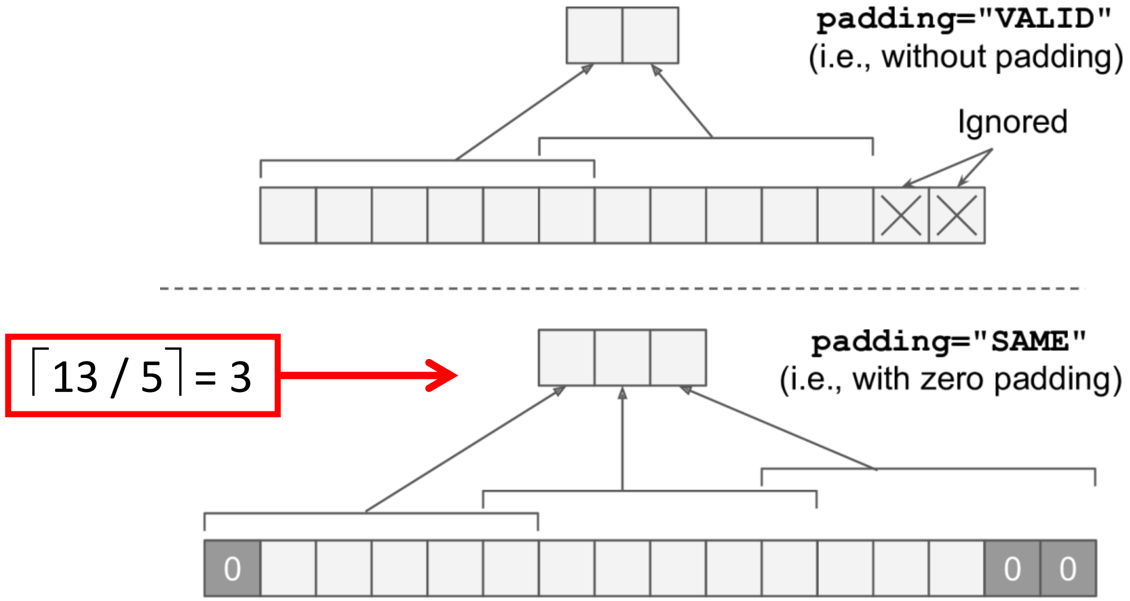
2. Convolutional Layer

- 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. Convolutional Layer

- Keras supports
  - **No padding** (default)
    `padding="VALID"`
  - **Zero padding**
    `padding="SAME"`
- Example:
  - Input width: 13
  - Filter width: 6
  - Stride: 5



$\lceil 13 / 5 \rceil = 3$

2. Convolutional Layer

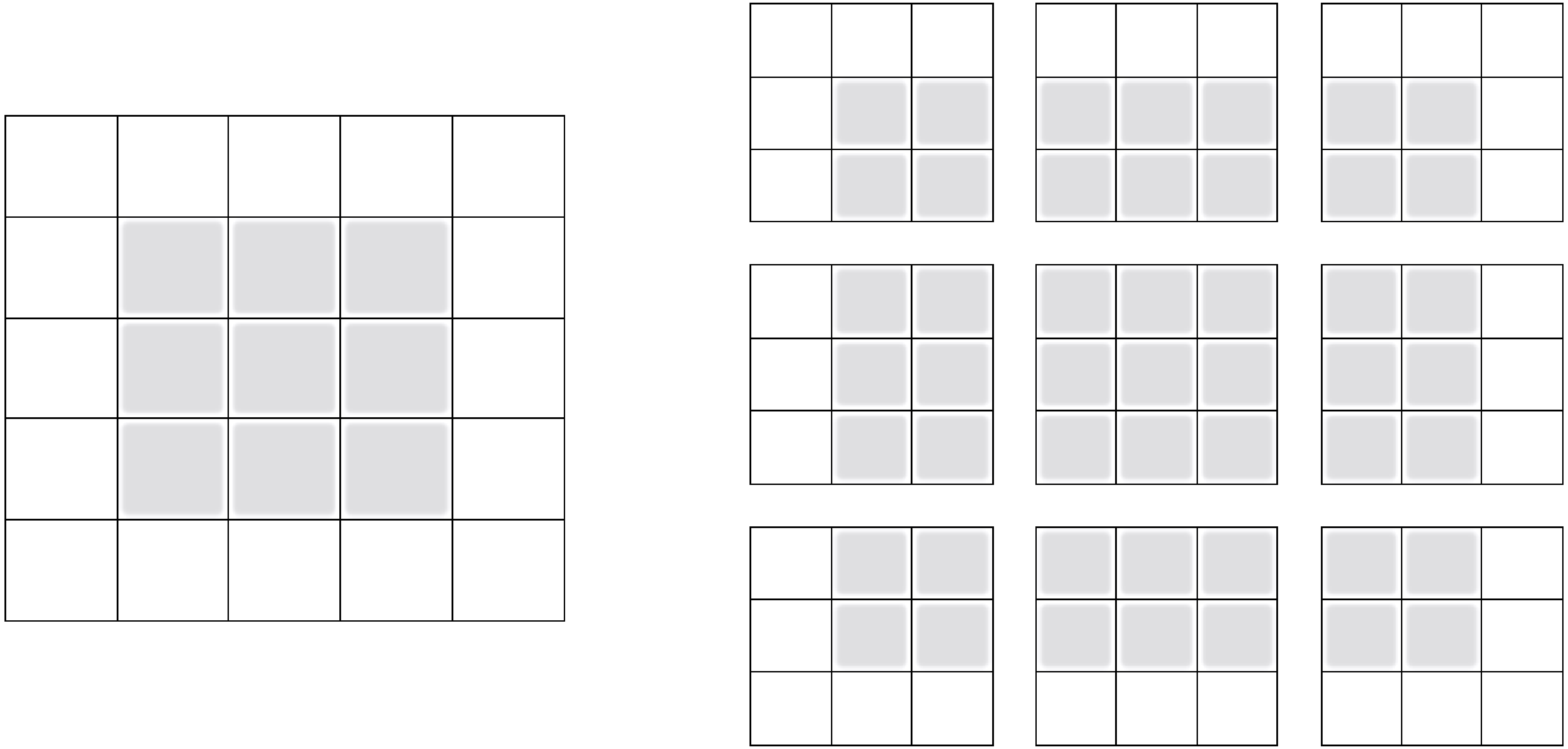
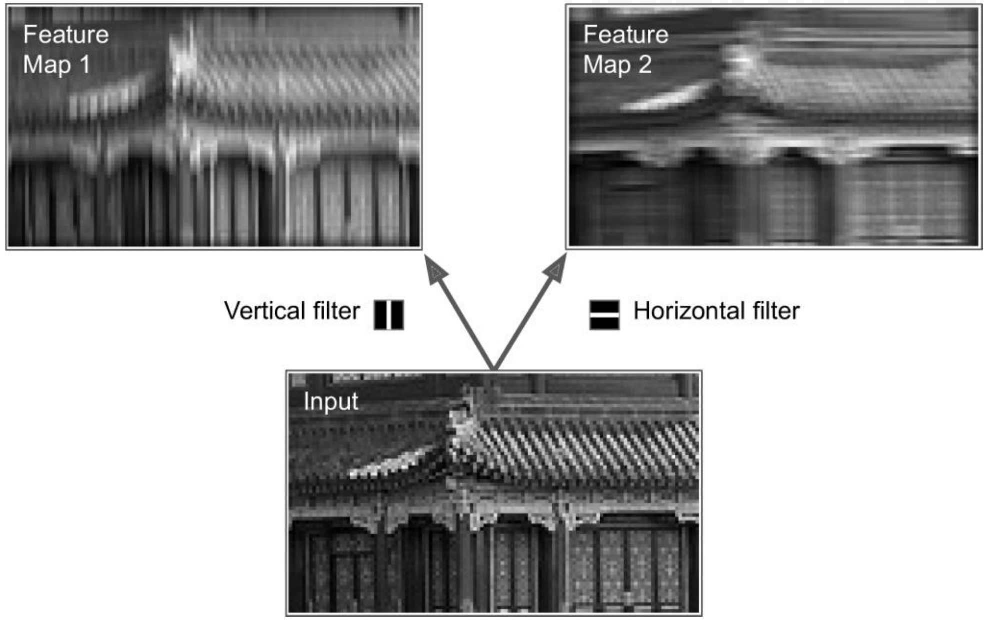


Figure 5.5 Valid locations of 3 × 3 patches in a 5 × 5 input feature map

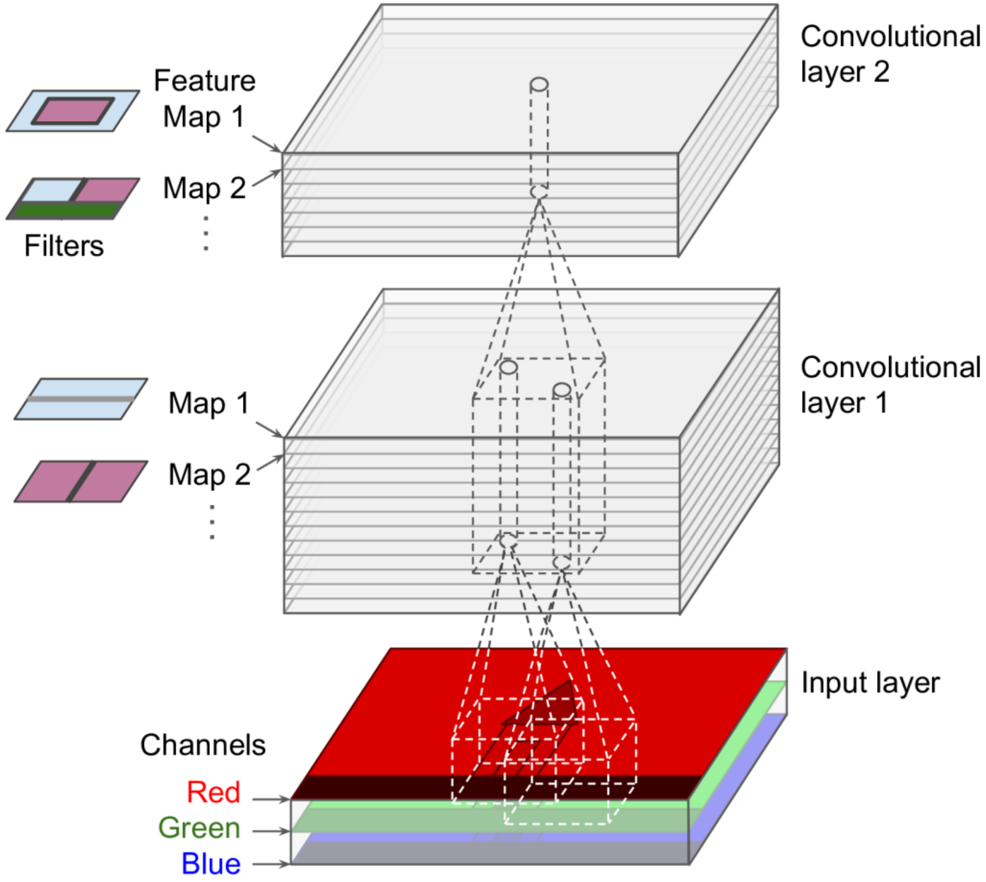
2.1 Filters

- 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.

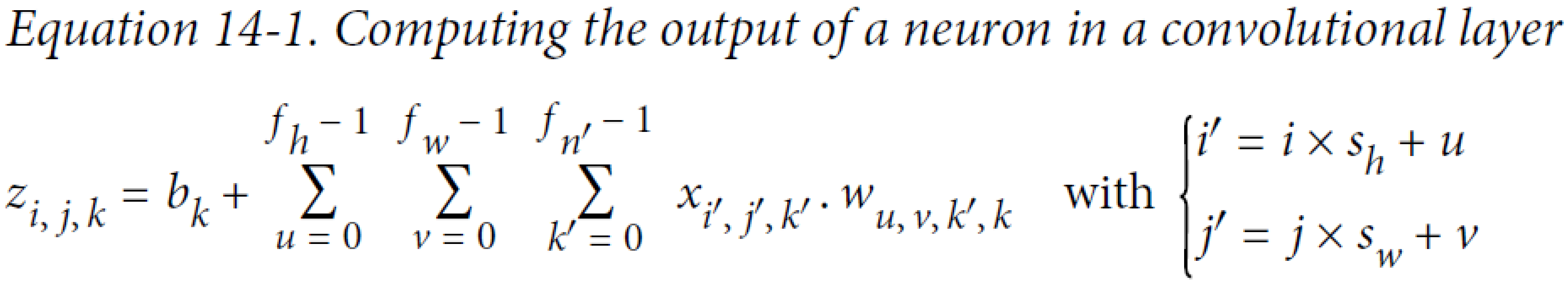


2.2 Stacking Feature Maps

- 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 14-1. Computing the output of a neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

- $z_{i,j,k}$ 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

2.4 Memory Requirements

- Convolutional layers require a **huge amount of RAM**.

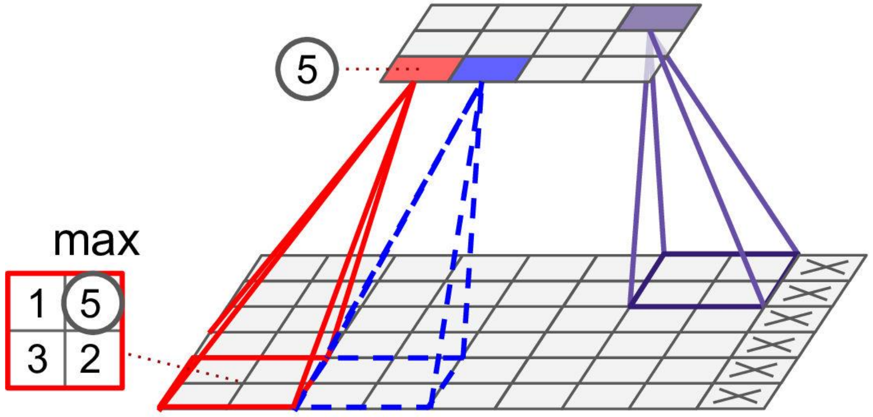
- **Example**: Convolutional layer with 5 × 5 filters, 200 feature maps of size 150 × 100, with stride 1 and "`same`" padding. Input is RGB image (three channels).
  - Parameters = (5 × 5 × 3 + 1) × 200 = 15,200
  - Size of feature maps (single precision) = 200 × 150 × 100 × 4 = 12 MB of RAM
  - 1.2 GB of RAM for a mini batch of 100 instances

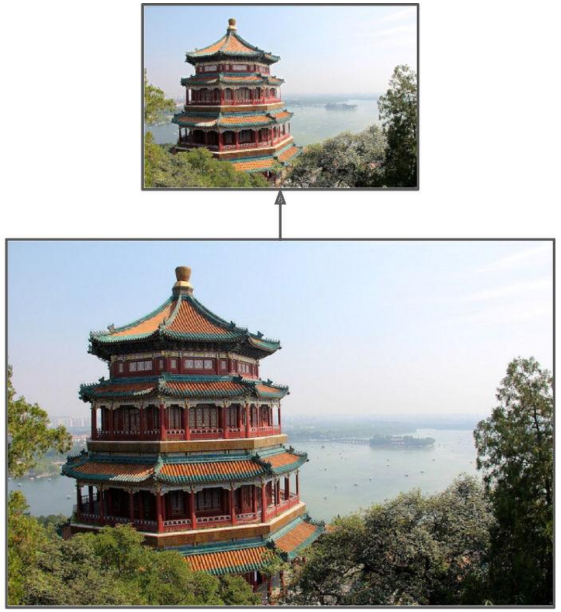
Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. 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**.



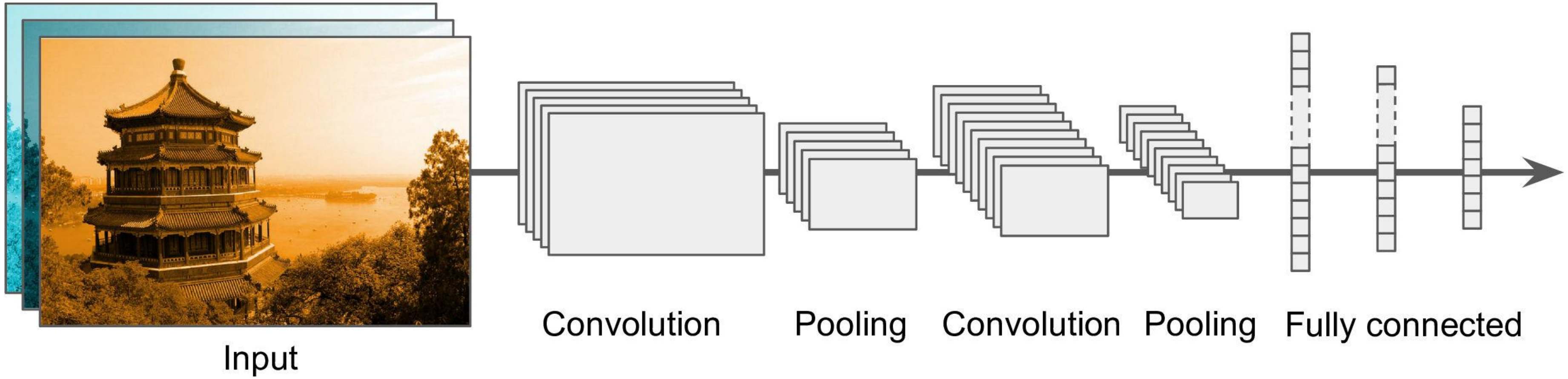


Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. 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 **dense** NN is added.



4.1 Example – Fashion MNIST

```
model = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same",
        input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
])
```

Filter size

Feature maps

2×2 window and stride 2

4.1 Example – Fashion MNIST

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

history = model.fit(X_train, y_train, epochs=10,
       validation_data=(X_valid, y_valid))
Train on 55000 samples, validate on 5000 samples
Epoch 1/10 55000/55000 [==============================] - 51s 923us/sample - loss:
0.7183 - accuracy: 0.7529 - val_loss: 0.4029 - val_accuracy: 0.8510
…
Epoch 10/10
55000/55000 [==============================] - 50s 911us/sample - loss: 0.2561 -
accuracy: 0.9145 - val_loss: 0.2891 - val_accuracy: 0.9036
```

4.1 Example – Fashion MNIST

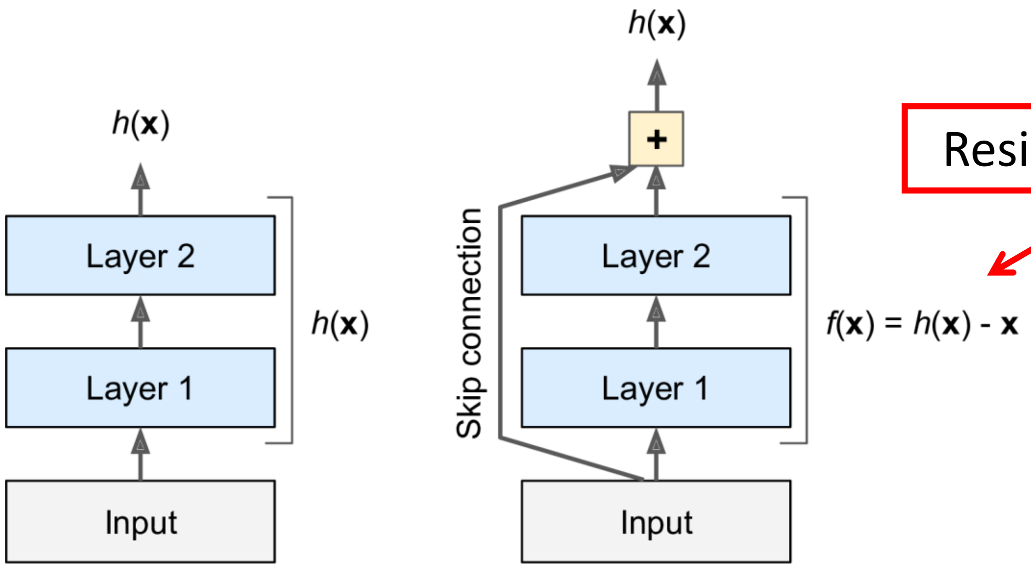
```
score = model.evaluate(X_test, y_test)
X_new = X_test[:10] # pretend we have new images
y_pred = model.predict(X_new)
```

```
10000/10000 [==============================] - 2s 239us/sample - loss: 
0.2972 - accuracy: 0.8983
```

Can reach 92% with more epochs

4.2 ResNet

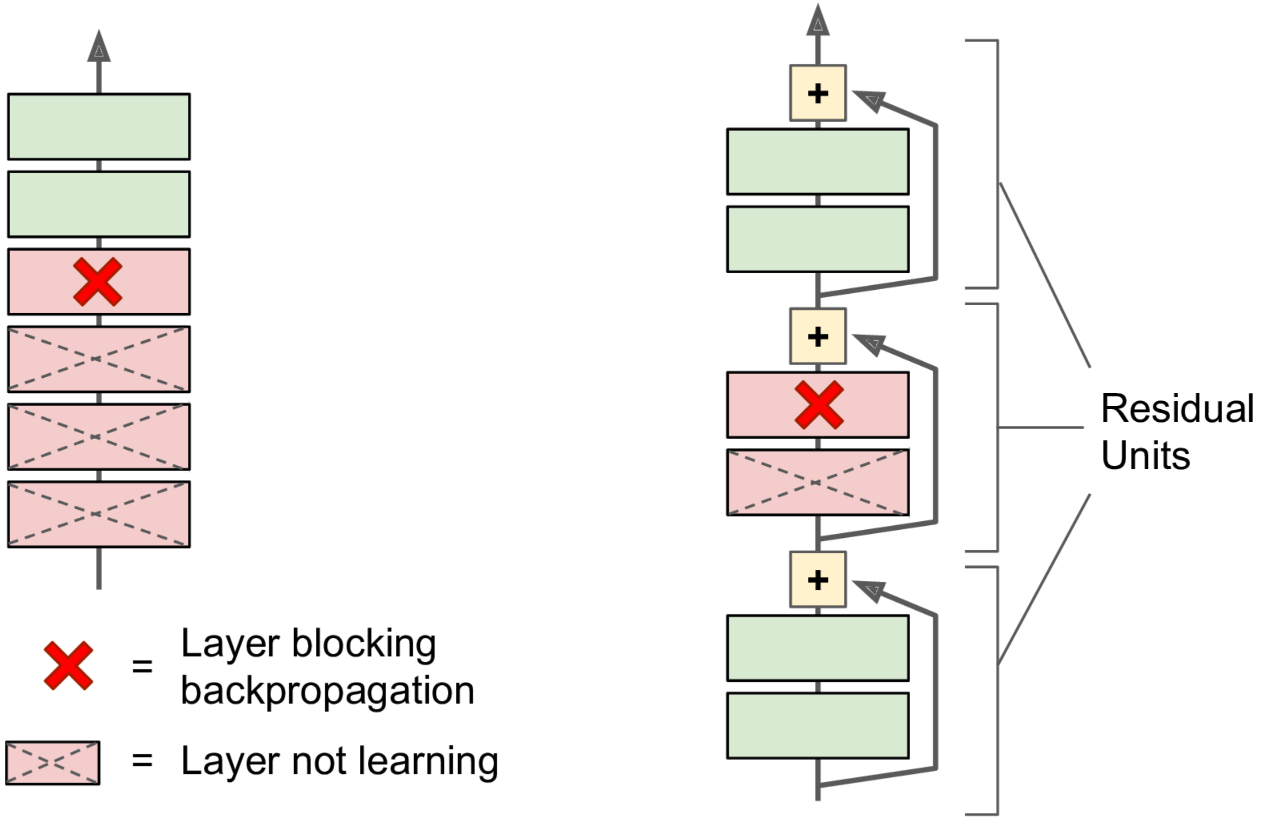
- **Residual Network** (or ResNet) won the ILSVRC 2015 challenge.
- Top-5 error rate under 3.6%, using an extremely deep CNN composed of **152 layers**.
- To train such a deep network, it uses **skip connections**.



Residual Learning

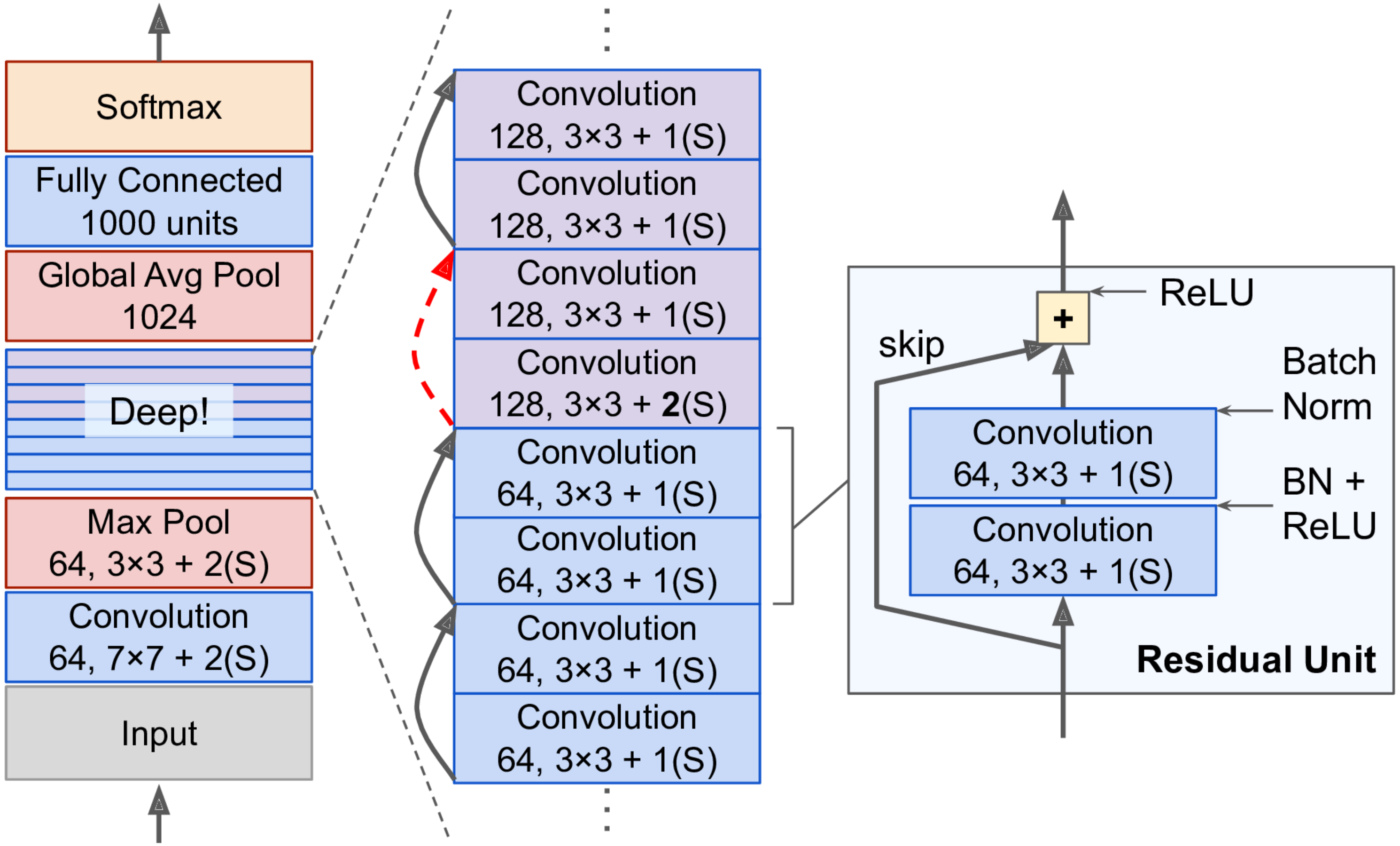
4.2 ResNet

- The network can start making progress even if several layers have not started learning yet.



4.2 ResNet

- ResNet is a **stack** of residual units.



Softmax
Fully Connected 1000 units
Global Avg Pool 1024
Deep!
Max Pool 64, 3×3 + 2(S)
Convolution 64, 7×7 + 2(S)
Input
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 2(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
ReLU
+
skip
Batch Norm
Convolution 64, 3×3 + 1(S)
BN + ReLU
Convolution 64, 3×3 + 1(S)
Residual Unit

Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises

5. Using Pretrained Models

- Pretrained networks are readily available from the **`keras.applications`** package.
- Check https://github.com/keras-team/keras-applications

- You can load the **ResNet-50** model, pretrained on **ImageNet**, with the following line of code:

```
model = keras.applications.resnet50.ResNet50(weights="imagenet")
```

5. Using Pretrained Models

```
# Input: 224 × 224-pixel images
images_resized = tf.image.resize(images, [224, 224])

# Preprocess images, should be scaled 0-255
inputs = keras.applications.resnet50.preprocess_input(
      images_resized * 255)

Y_proba = model.predict(inputs)

# Get top predictions out of the 1000-class probs.
top_K = keras.applications.resnet50.decode_predictions(Y_proba, top=3)
```

5. Using Pretrained Models

```
# Print results
for image_index in range(len(images)):
   print("Image #{}".format(image_index))
   for class_id, name, y_proba in top_K[image_index]:
      print(" {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
   print()
```

```
Image #0
  n03877845 - palace       42.87%
  n02825657 - bell_cote    40.57%
  n03781244 - monastery    14.56%

Image #1
  n04522168 - vase         46.83%
  n07930864 - cup          7.78%
  n11939491 - daisy        4.87%
```

Correct Class

Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises

6. Pretrained Models for Transfer Learning

- Training a pretrained network (**Xception**) for a dataset from TFDS (https://www.tensorflow.org/datasets).
- `tf_flowers`: 3670 images, 5 classes

```
# Load the dataset
import tensorflow_datasets as tfds

dataset, info = tfds.load("tf_flowers",
      as_supervised=True, with_info=True)

dataset_size = info.splits["train"].num_examples # 3670
n_classes = info.features["label"].num_classes   # 5
class_names = info.features["label"].names
```

6. Pretrained Models for Transfer Learning

```
# Relooad the dataset with three splits tf.data.Dataset
test_set_raw, valid_set_raw, train_set_raw = tfds.load(
      "tf_flowers", split=["train[:10%]",
      "train[10%:25%]", "train[25%:]"],
      as_supervised=True)

# Define the preprocessing function
def preprocess(image, label):
    resized_image = tf.image.resize(image, [224, 224])
    final_image = keras.applications.xception.preprocess_input(
      resized_image)
    return final_image, label
```

6. Pretrained Models for Transfer Learning

```
# Apply this preprocessing function to the 3 datasets
# Shuffle the training set
# Add batching and prefetching to all the datasets
batch_size = 32
train_set = train_set_raw.shuffle(3000).repeat()
train_set = train_set.map(preprocess).batch(
       batch_size).prefetch(1)
valid_set = valid_set_raw.map(preprocess).batch(
       batch_size).prefetch(1)
test_set = test_set_raw.map(preprocess).batch(
       batch_size).prefetch(1)
```

6. Pretrained Models for Transfer Learning

```
# Load an Xception model, pretrained on ImageNet
#  excluding the global avg pool. and dense o/p layers
base_model = keras.applications.xception.Xception(
      weights="imagenet", include_top=False)

# Add global avg pool. layer based on model output
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
output = keras.layers.Dense(n_classes, # Add desnse o/p
      activation="softmax")(avg)
model = keras.models.Model(inputs=base_model.input,
      outputs=output) # Create the Keras Model
```

6. Pretrained Models for Transfer Learning

```
# Freeze the weights of the pretrained layers
for layer in base_model.layers:
    layer.trainable = False

# Compile the model and start training
optimizer = keras.optimizers.SGD(lr=0.2, momentum=0.9,
        decay=0.01) # LR=0.2 with scheudle, k=1/0.01
model.compile(loss="sparse_categorical_crossentropy",
        optimizer=optimizer, metrics=["accuracy"])
history = model.fit(train_set, epochs=5,
        validation_data=valid_set) # Tops at 75-80% acc.
```

$$\boxed{\eta(t) = \eta_0 \,/\, (1 + t/k)}$$

6. Pretrained Models for Transfer Learning

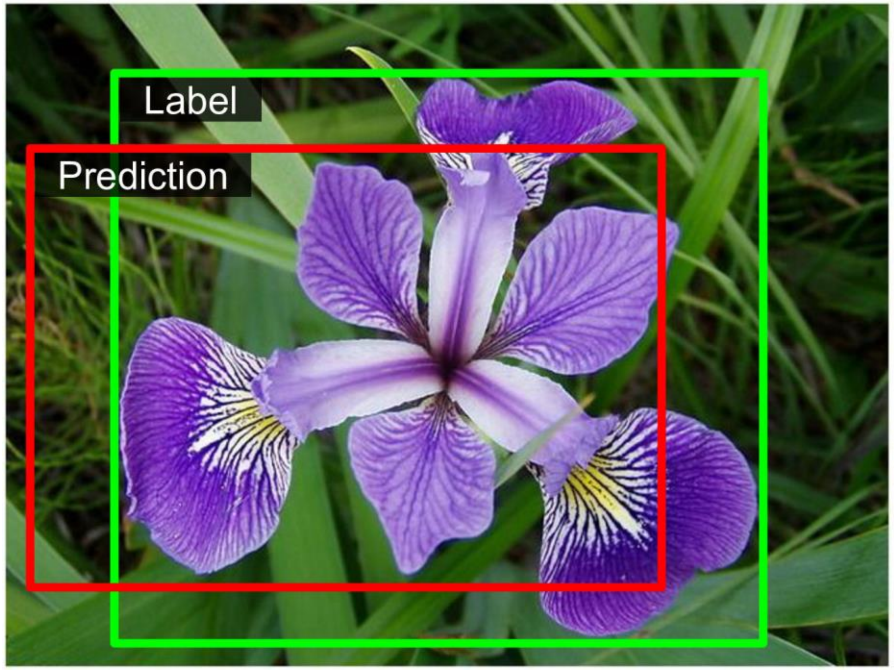
```
# Unfreeze the weights of the pretrained layers
for layer in base_model.layers:
       layer.trainable = True

# Recompile with lower LR and decay
optimizer = keras.optimizers.SGD(lr=0.01, momentum=0.9,
        nesterov=True, decay=0.001)
model.compile(loss="sparse_categorical_crossentropy",
       optimizer=optimizer, metrics=["accuracy"])
history = model.fit(train_set, epochs=40,
       validation_data=valid_set) # Result: 95% acc.
```

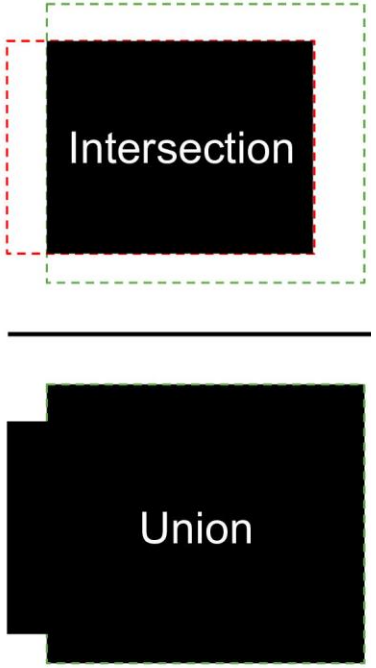
Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises

7. Classification and Localization

- **Localizing** an object in a picture can be expressed as a **regression** task.
- Predict the horizontal and vertical coordinates of the object's center and its height and width.





Common metric: the Intersection over Union (IoU)

7. Classification and Localization

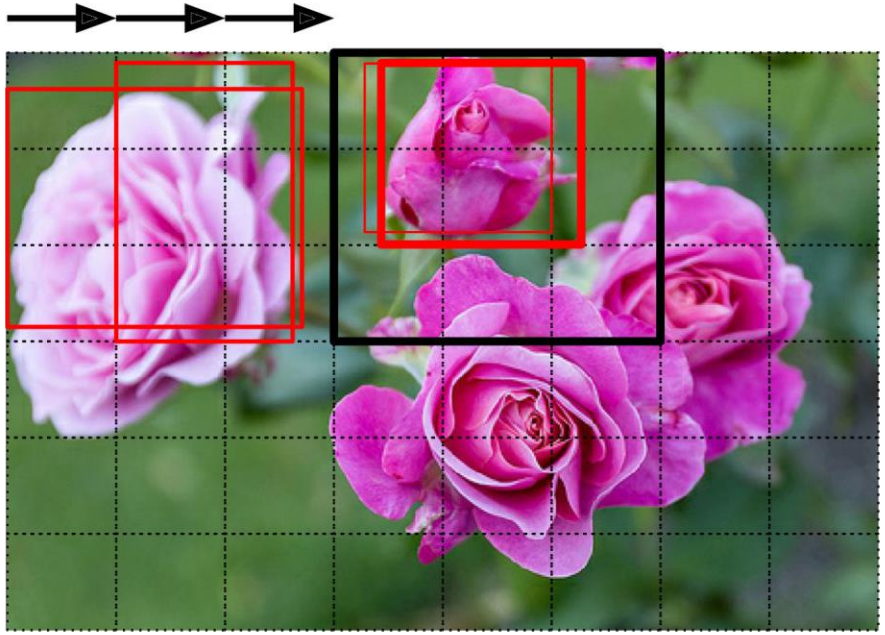
```
base_model = keras.applications.xception.Xception(
      weights="imagenet", include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
class_output = keras.layers.Dense(n_classes, activation="softmax")(avg)
loc_output = keras.layers.Dense(4)(avg)
model = keras.Model(inputs=base_model.input,
      outputs=[class_output, loc_output])

model.compile(loss=["sparse_categorical_crossentropy "mse"],
       loss_weights=[0.8, 0.2],
      optimizer=optimizer, metrics=["accuracy"])
```

Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. Memory requirements
3. Pooling layer
4. CNN architectures
    1. Example – Fashion MNIST
    2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises

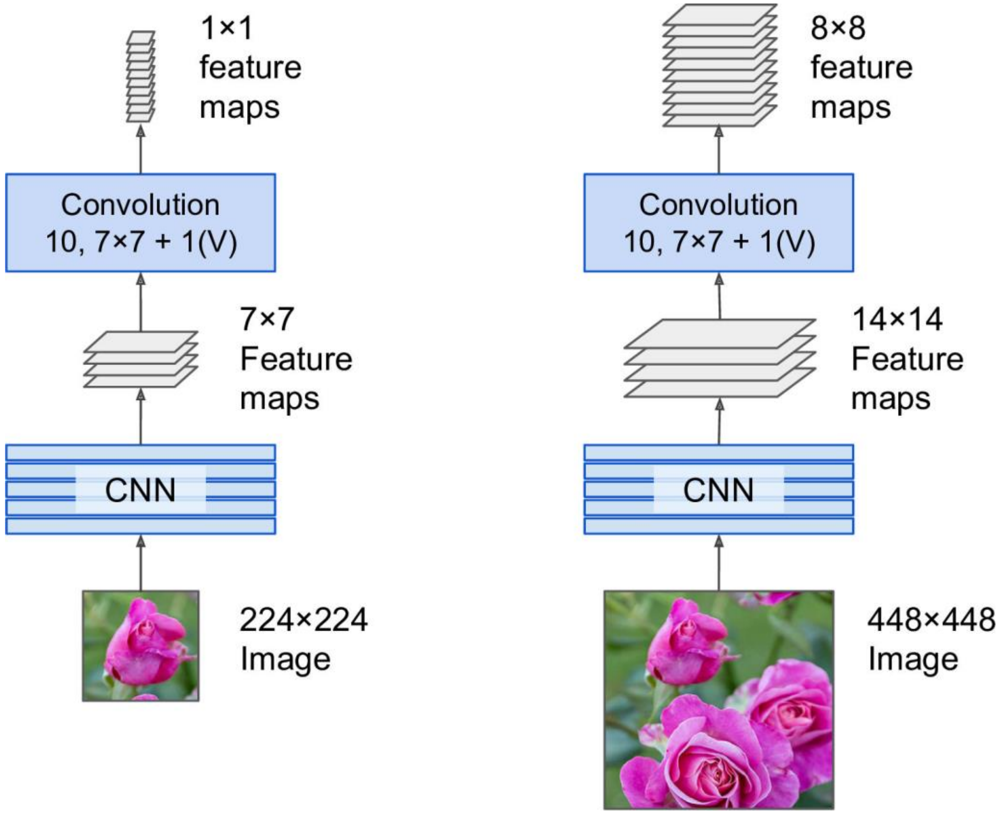
8. Object detection

- The task of **classifying and localizing** multiple objects in an image.
- A **slow** approach is use a CNN trained to classify and locate a single object, then **slide** it across the image.



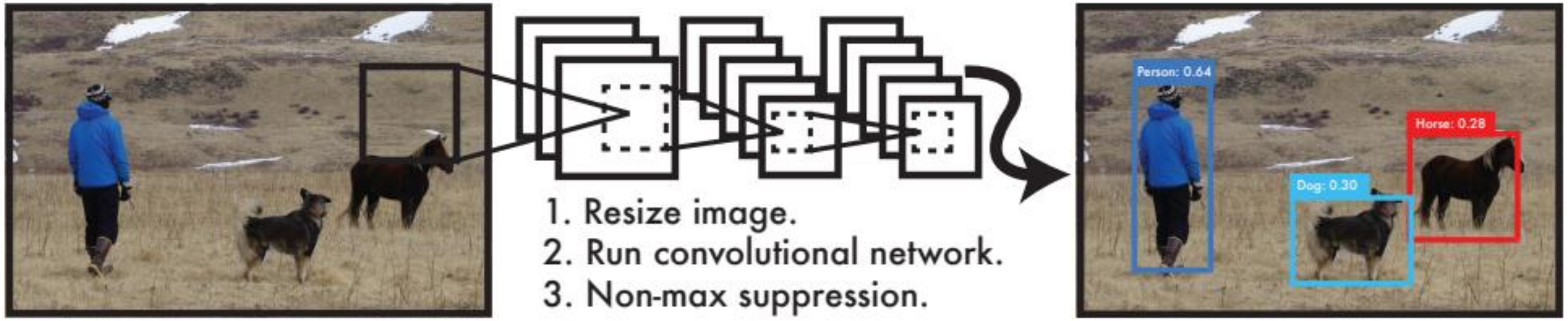
8.1 Fully Convolutional Networks

- FCN has also a **convolution layer at the output** with **`valid`** padding.
- FCN can process images of **any size**.
- **Example**:
  - Train the CNN for classification and localization on small images, 10 outputs.
  - For larger image, it output 8 × 8 grid where each cell contains 10 numbers.



8.2 You Only Look Once (YOLO)

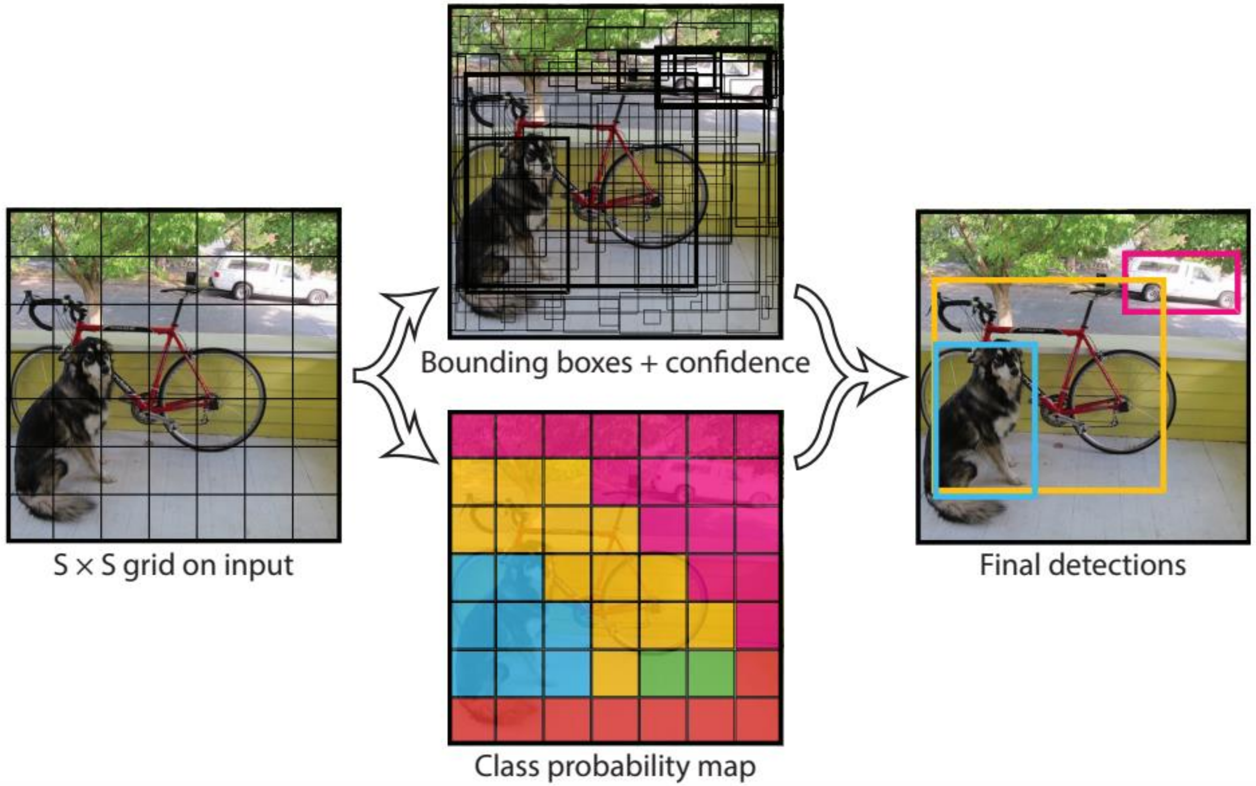
- **YOLO** is an extremely fast and accurate object detection architecture.
  1. Resizes the input image to 448 × 448
  2. Runs a single convolutional network on the image
  3. Thresholds the resulting detections by the model’s confidence.



8.2 You Only Look Once (YOLO)

- Models detection as a regression problem. It divides the image into an $S \times S$ grid.

- For each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities.

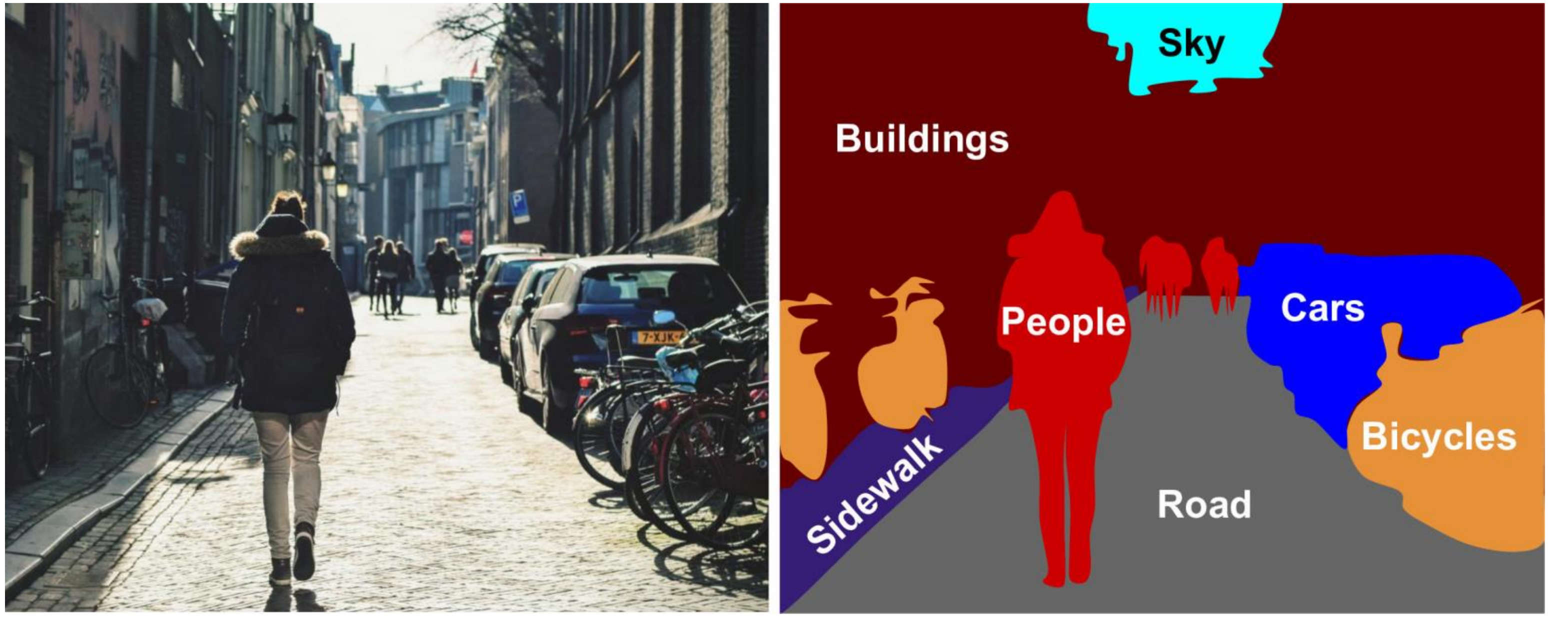


Outline

1. Introduction
2. Convolutional layer
   1. Filters
   2. Stacking feature maps
   3. Mathematical summary
   4. Memory requirements
3. Pooling layer
4. CNN architectures
   1. Example – Fashion MNIST
   2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises

9. Semantic Segmentation

- Each pixel is classified according to the class of the object it belongs to.
- Can use **FCN** followed by up **sampling** layers.



Exercises

14.9. Build your own CNN from scratch and try to achieve the highest possible accuracy on **MNIST**.

14.10. Use **transfer learning** for large image classification, going through these steps:

a) Create a training set containing at least 100 images per class. For example, you could classify your own pictures based on the location (beach, mountain, city, etc.), or alternatively you can use an existing dataset (e.g., from TensorFlow Datasets).

b) Split it into a training set, a validation set, and a test set.

c) Build the input pipeline, including the appropriate preprocessing operations, and optionally add data augmentation.

d) Fine-tune a pretrained model on this dataset.

Summary

1. Introduction
2. Convolutional layer
   1. Filters
   2. Stacking feature maps
   3. Mathematical summary
   4. Memory requirements
3. Pooling layer
4. CNN architectures
   1. Example – Fashion MNIST
   2. ResNet
5. Using pretrained models
6. Pretrained models for transfer learning
7. Classification and localization
8. Object detection
9. Semantic segmentation
10. Exercises