Convolutional Neural Networks

Prof. Gheith Abandah

Reference: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly), 2017, 978-1-491-96229-9.

Introduction

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

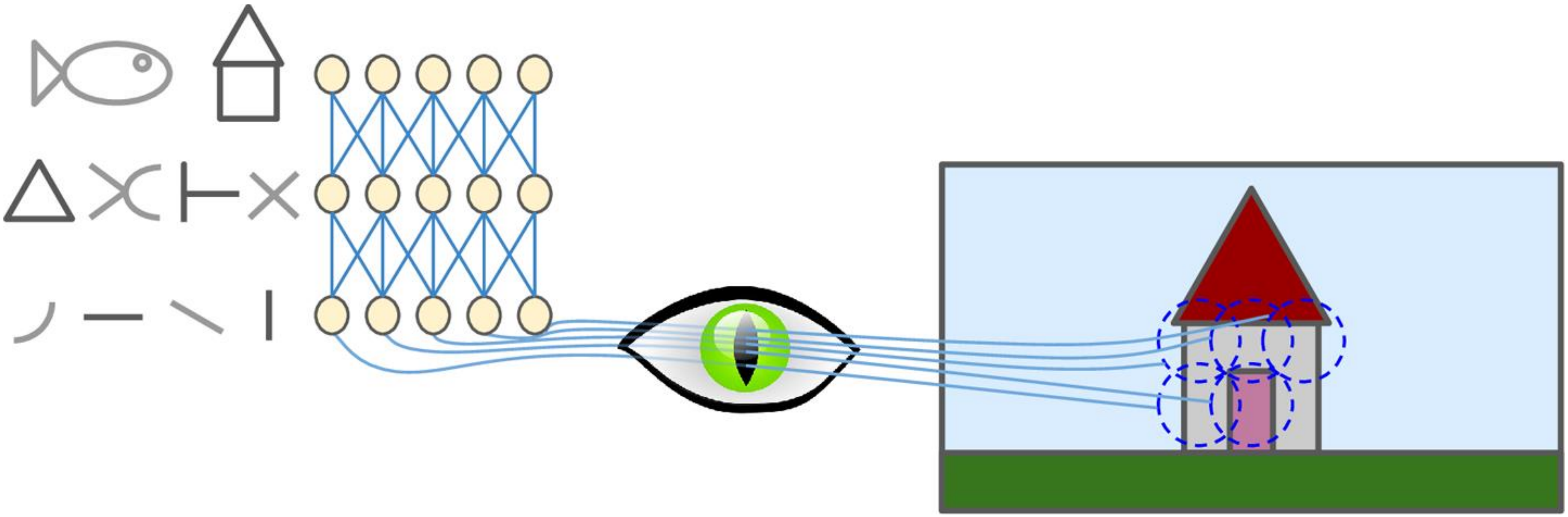
https://youtu.be/YRhxdVk_sIs

Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. TensorFlow implementation
3. Pooling layer
4. CNN architectures
5. Exercises

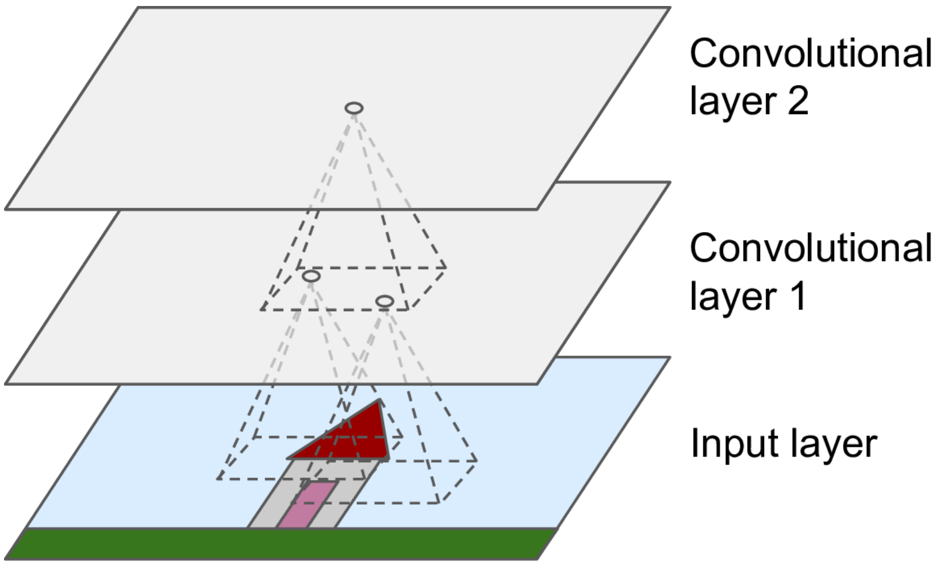
1. Introduction

- *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.*



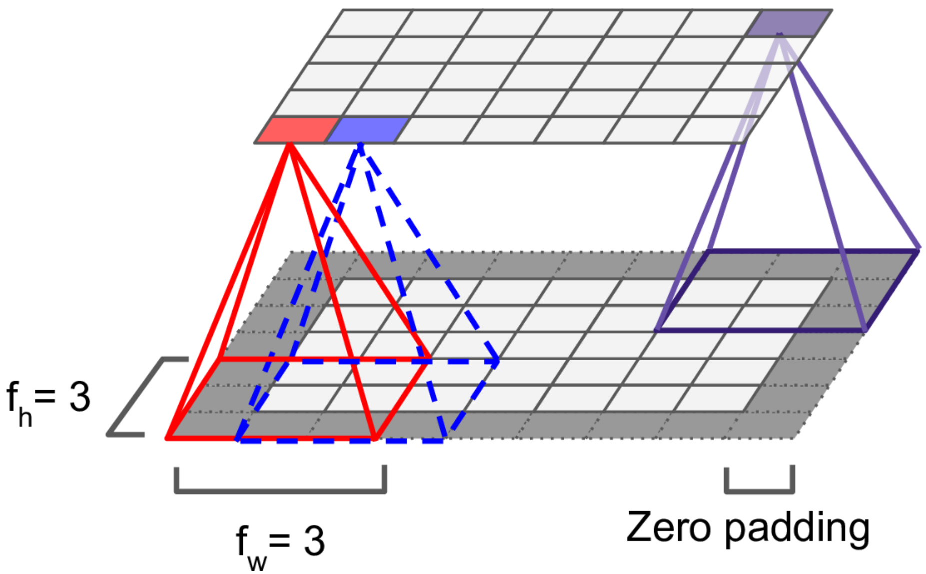
2. Convolutional Layer

- 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.



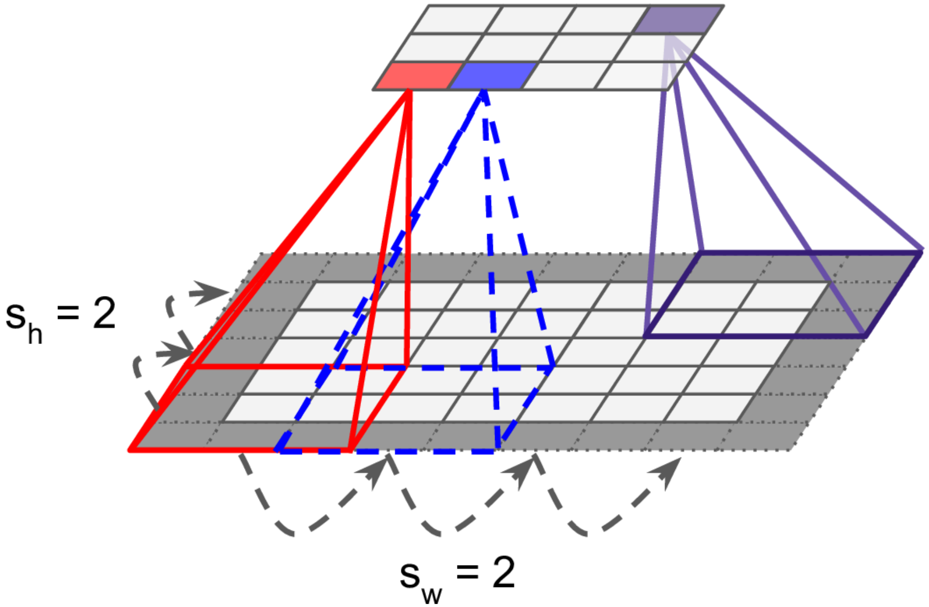
2. Convolutional Layer

- 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.



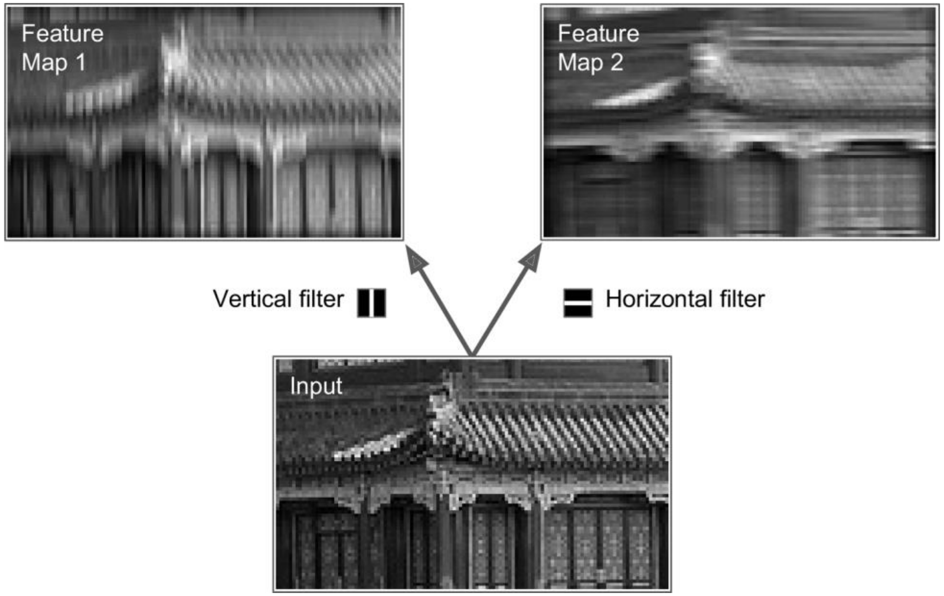
2. Convolutional Layer

- 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 $i \times s_h + f_h - 1$, columns $j \times s_w$ to $j \times s_w + f_w - 1$.



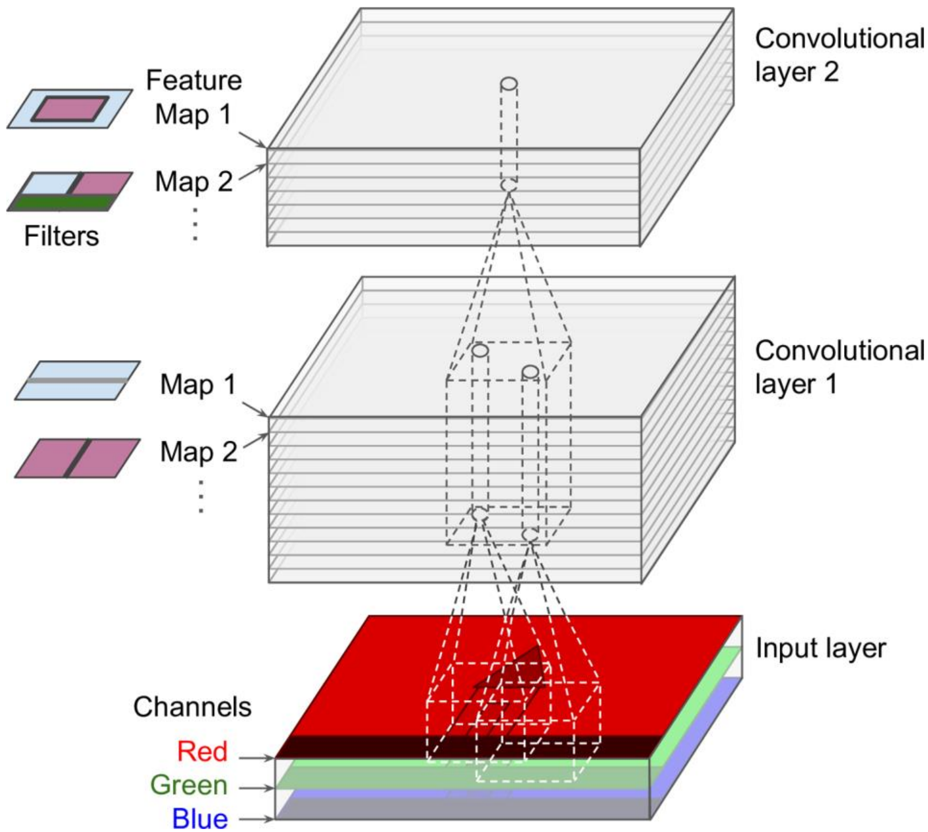
2.1. Filters

- 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

2.4 TensorFlow Implementation

```
import numpy as np
from sklearn.datasets import load_sample_images

# Load sample images
dataset = np.array(load_sample_images().images, dtype=np.float32)
batch_size, height, width, channels = dataset.shape

# Create 2 filters
filters_test = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters_test[:, 3, :, 0] = 1  # vertical line
filters_test[3, :, :, 1] = 1  # horizontal line

# Create a graph with input X plus a convolutional layer applying the 2 filters
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
convolution = tf.nn.conv2d(X, filters, strides=[1,2,2,1], padding="SAME")

with tf.Session() as sess:
    output = sess.run(convolution, feed_dict={X: dataset})
```

(s_h and s_w)

SAME: Zero padding
VALID: No padding

2.4 TensorFlow Implementation

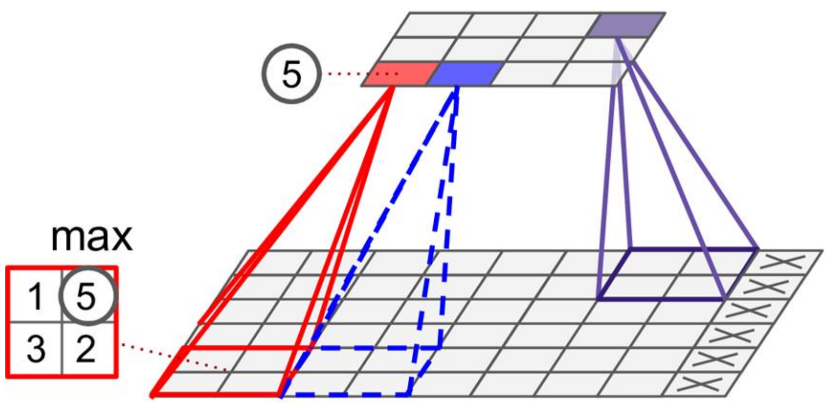
```
plt.imshow(output[0, :, :, 1])  # plot 1st image's 2nd feature map
plt.show()
```

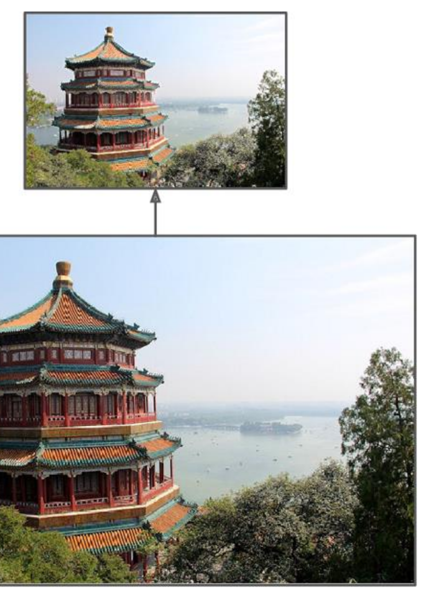
Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. TensorFlow implementation
3. Pooling layer
4. CNN architectures
5. 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.





3. Pooling Layer

- The following code creates a max pooling layer using a 2 × 2 kernel, stride 2, and no padding, then applies it to all the images in the dataset.

```
[...] # load the image dataset, just like above

# Create a graph with input X plus a max pooling layer
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
max_pool = tf.nn.max_pool(X, ksize=[1,2,2,1], strides=[1,2,2,1],padding="VALID")

with tf.Session() as sess:
    output = sess.run(max_pool, feed_dict={X: dataset})
```

[batch size, height, width, channels]

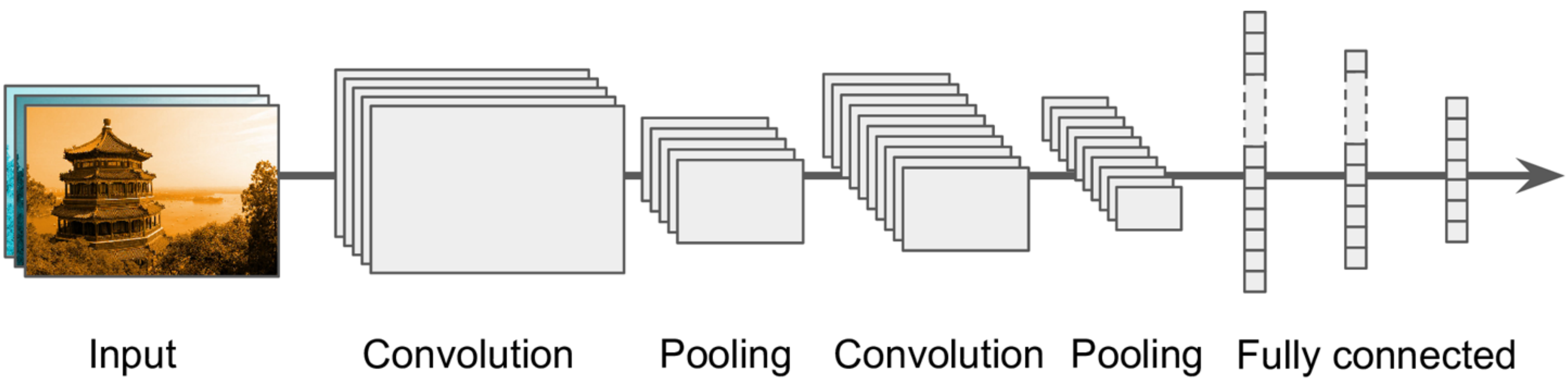
There is also avg_pool()

Outline

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. TensorFlow implementation
3. Pooling layer
4. CNN architectures
5. 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.



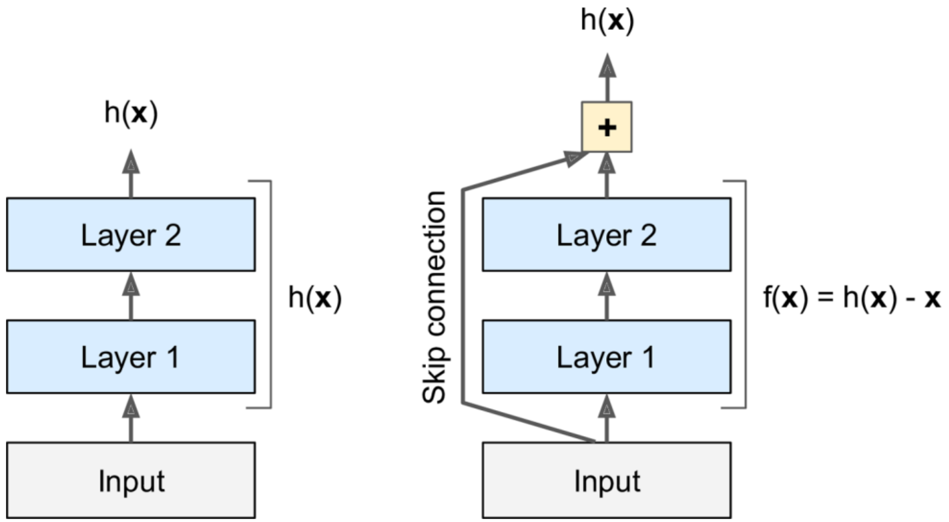
4.1. LeNet-5 (LENET Section)

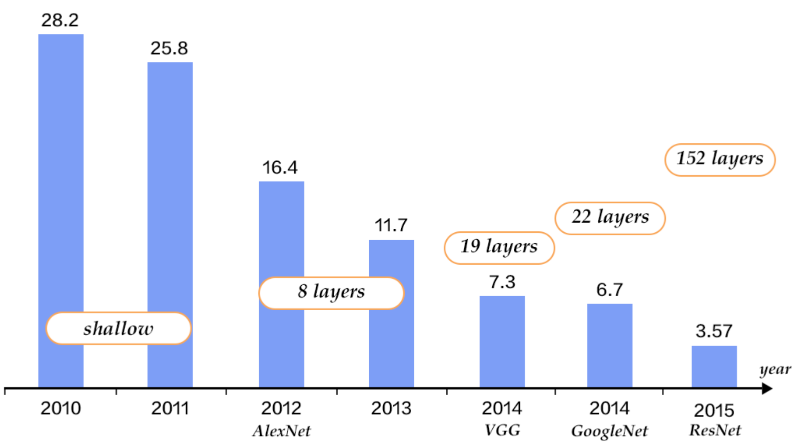
- It was created by Yann LeCun in 1998 and widely used for handwritten digit recognition (MNIST).

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

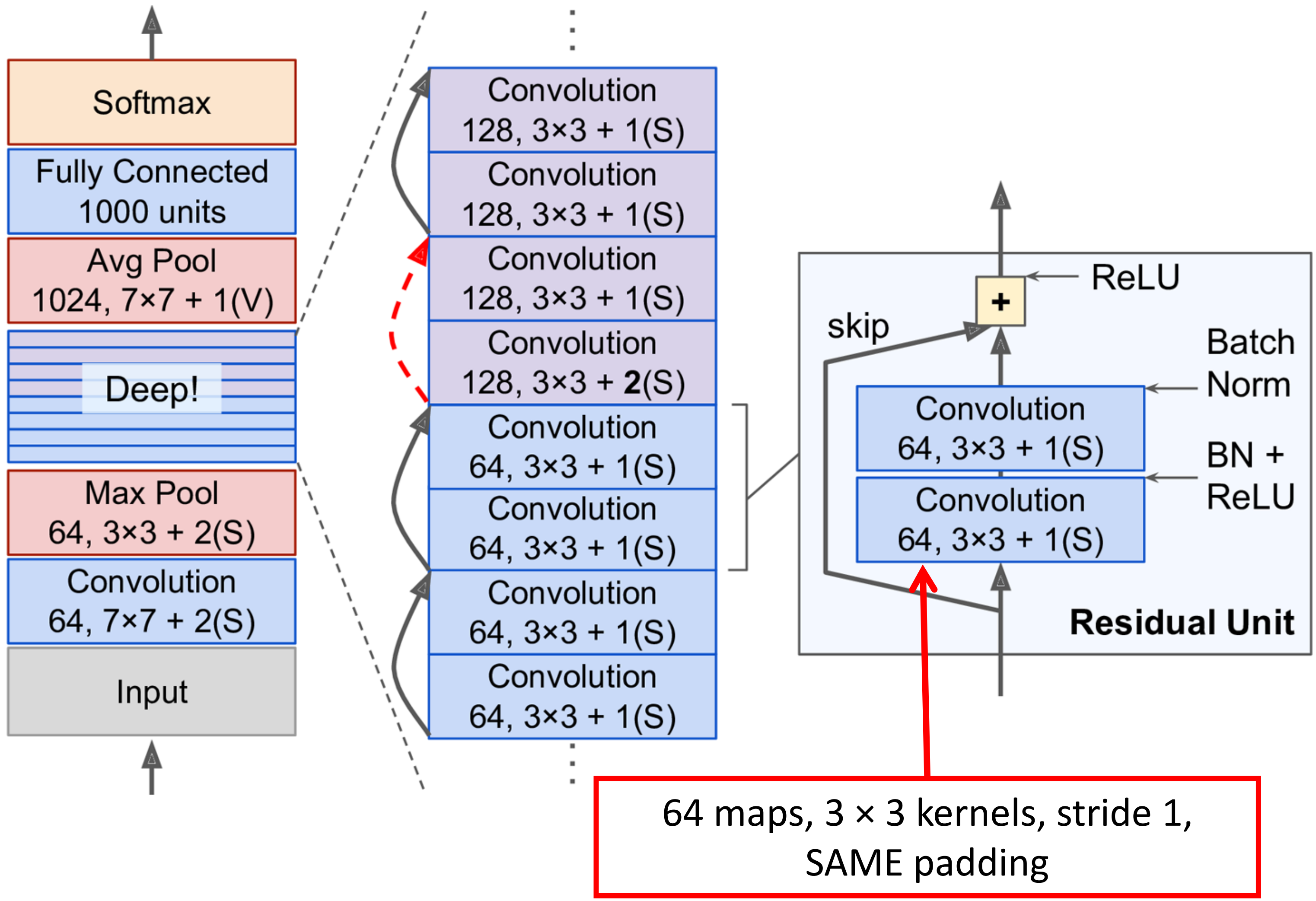
4.2. ResNet: Residual Network

- The winner of ILSVRC 2015, developed by Kaiming He et al., delivered top-5 error rate under 3.6%, using an extremely deep CNN of 152 layers.
- Uses *skip connections* (*shortcut connections*)
- *ILSVRC*: ImageNet Large Scale Visual Recognition Challenge





4.2. ResNet: Residual Network



Summary

1. Introduction
2. Convolutional layer
    1. Filters
    2. Stacking feature maps
    3. Mathematical summary
    4. TensorFlow implementation
3. Pooling layer
4. CNN architectures
5. Exercises

Exercises

From Chapter 13, solve exercises:

- 2
- 7