Machine Learning

Lecture 9: Convolutional Neural Networks

Instructor: Dr. Farhad Pourkamali Anaraki

Overview

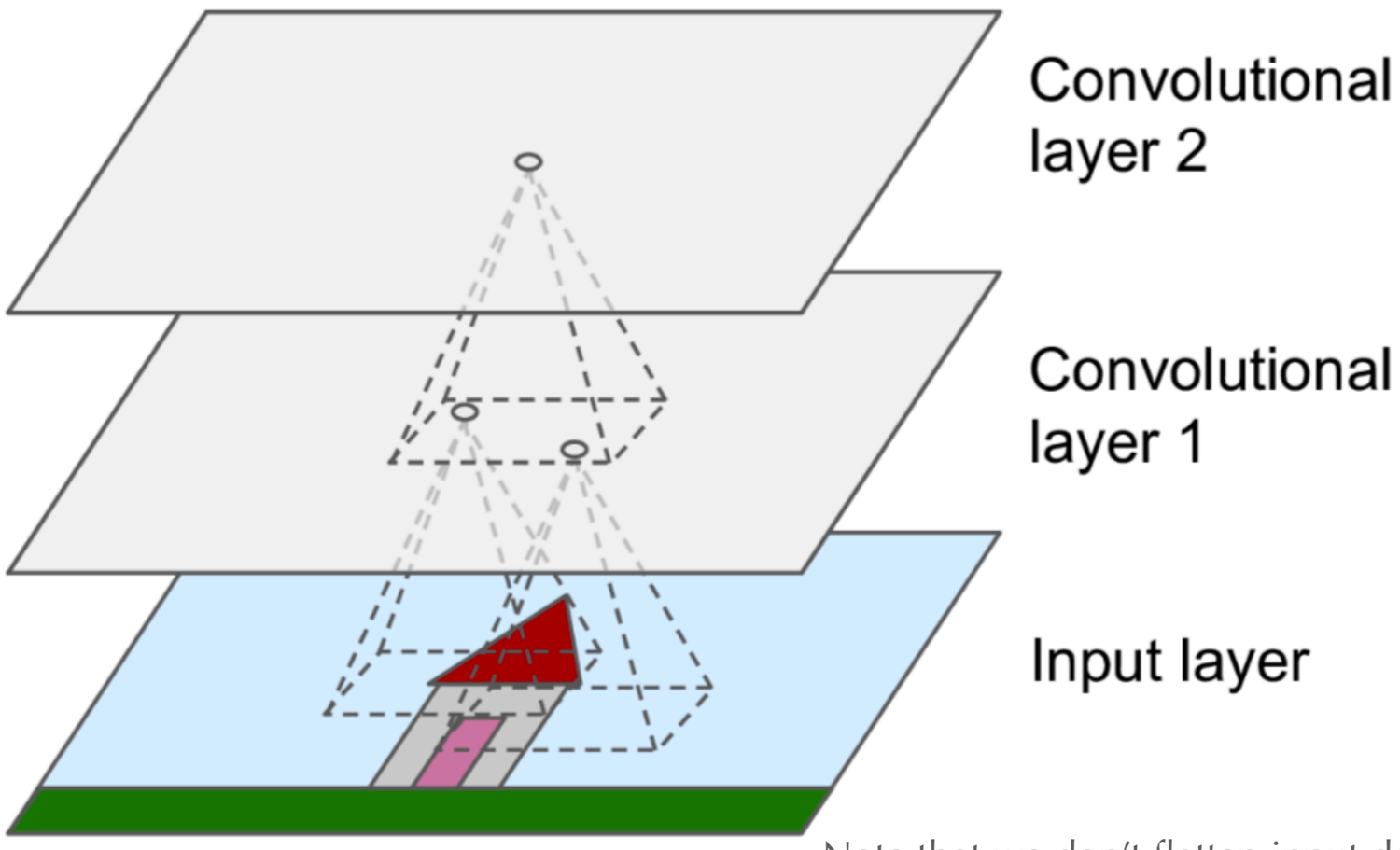
- Thanks to the increase in computational power and the amount of available training data, convolutional neural networks (CNNs) have achieved great performance on complex visual tasks
  - Image search services, self-driving cars, video classification systems, etc.
  - Not restricted to visual applications, e.g., voice recognition
- We explain building blocks of CNNs and how to implement them using Keras
  - Convolutional layers
  - Pooling layers
  - CNN architectures

Convolutional layers

- Neurons in the first *convolutional layer* are **not** connected to every single pixel in the input image, but only to pixels in their receptive fields

- Each neuron in the second *convolutional layer* is connected only to neurons located within a small rectangle in the first layer

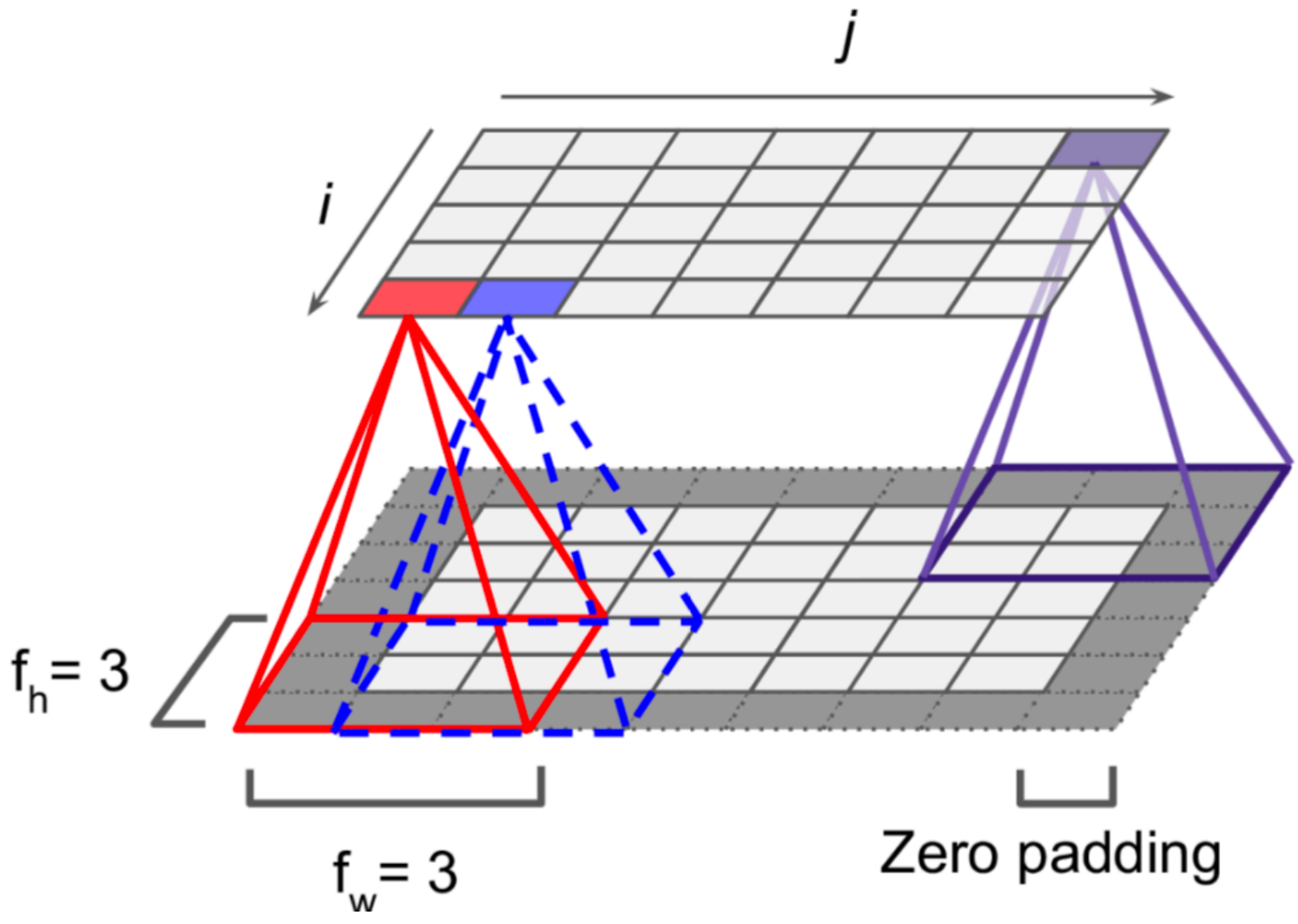
- Hierarchical structure: low-level features to high-level features



Note that we don't flatten input data

Convolutional layers

- To be more formal, each neuron in row i and column j of a given layer is connected to the outputs of the neurons in the previous layer located in rows i to $i + f_h - 1$ and columns j to $j + f_w - 1$, where f_h and f_w are the height and width of the receptive field

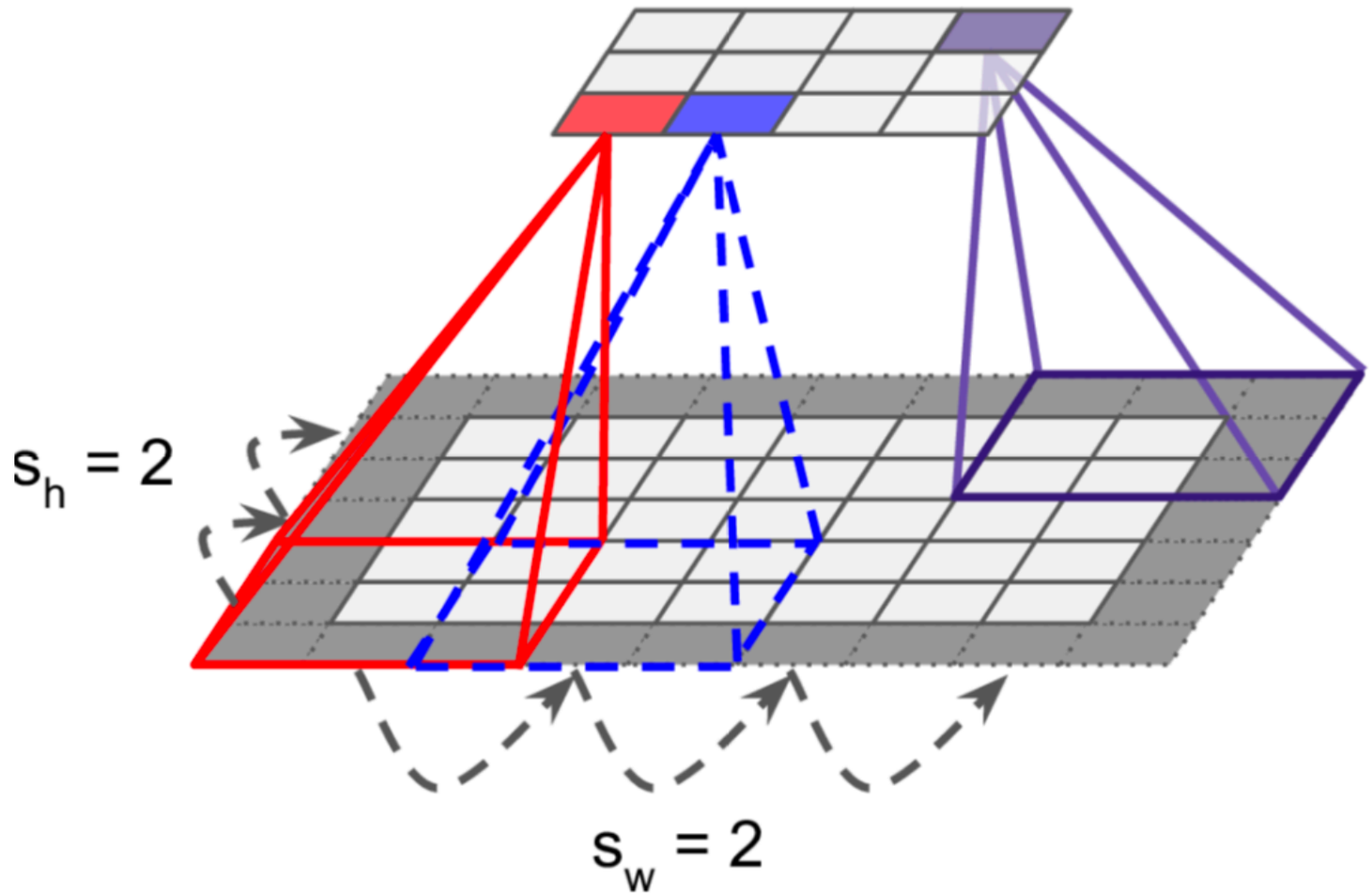


- We add zeros around the inputs to have the same number of rows and columns

Stride

- The shift from one receptive field to the next is called the **stride**

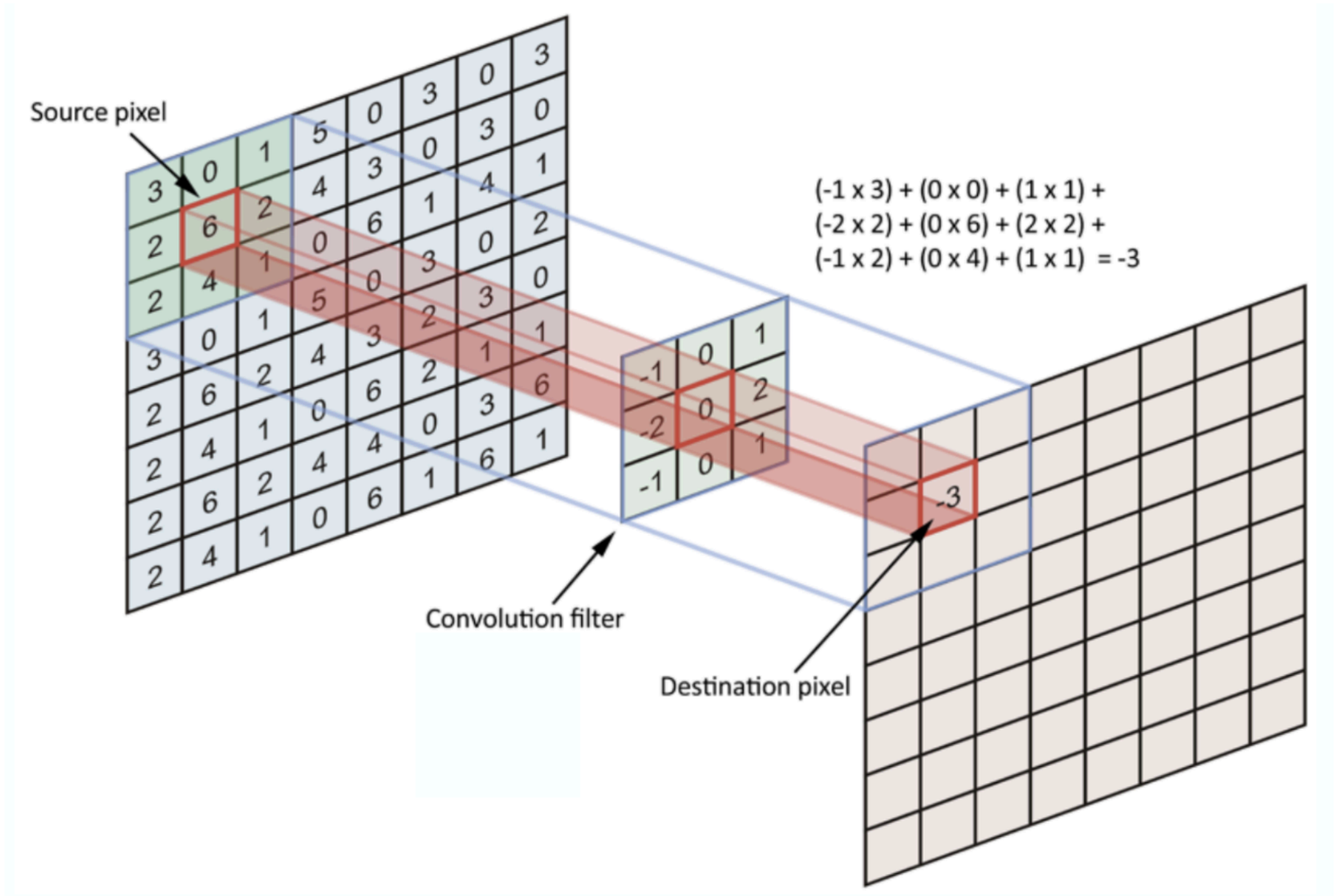
- 3x3 receptive fields and a stride of 2



- Neuron located in row i and column j is connected to rows $i \times s_h$ to $i \times s_h + f_h - 1$ and columns $j \times s_w$ to $j \times s_w + f_w - 1$

Filters

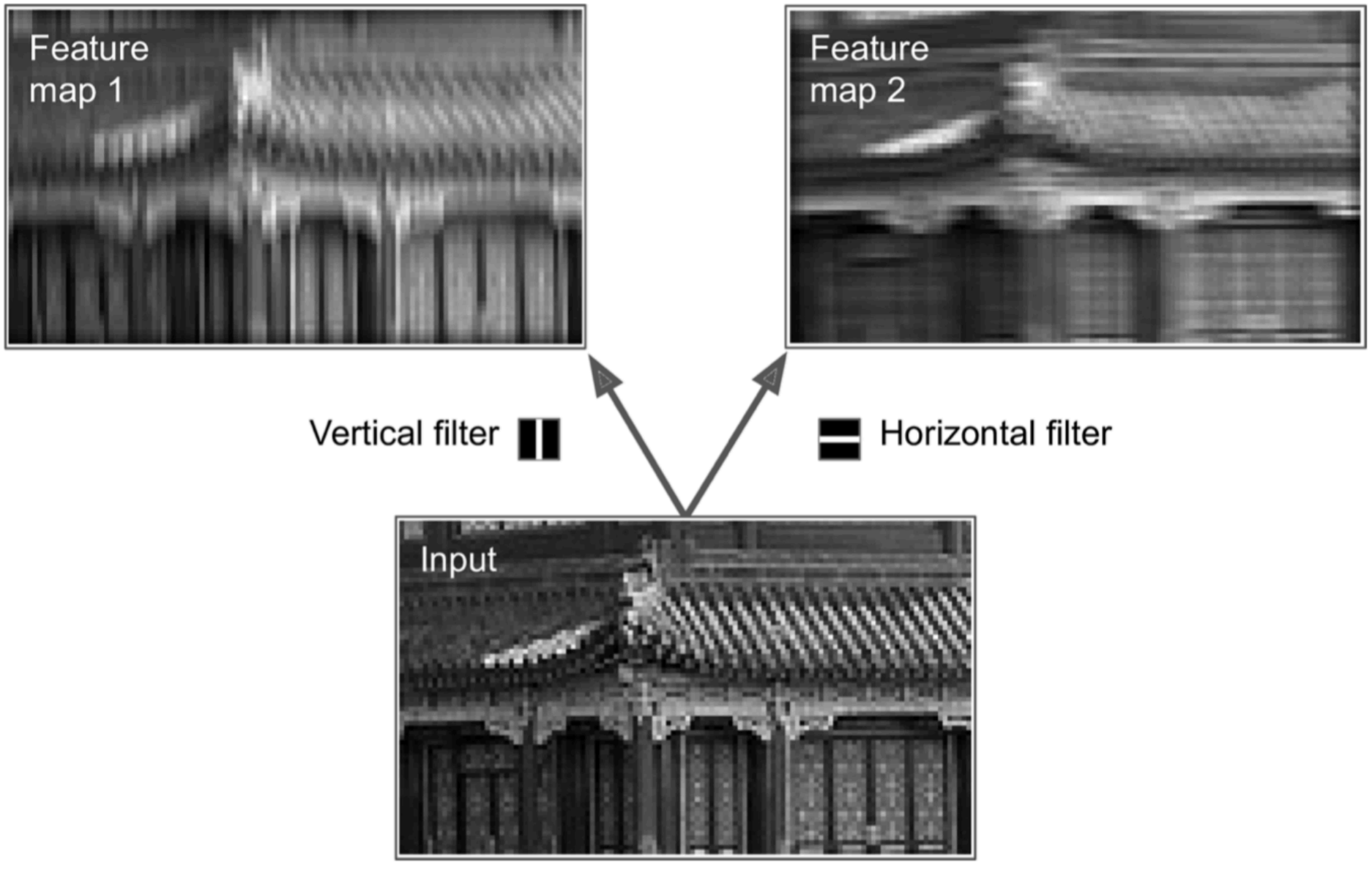
- A neuron's weight can be represented as a small image of the size of the receptive field
  - We refer to sets of weights as filters or convolutional kernels



The convolution operation.

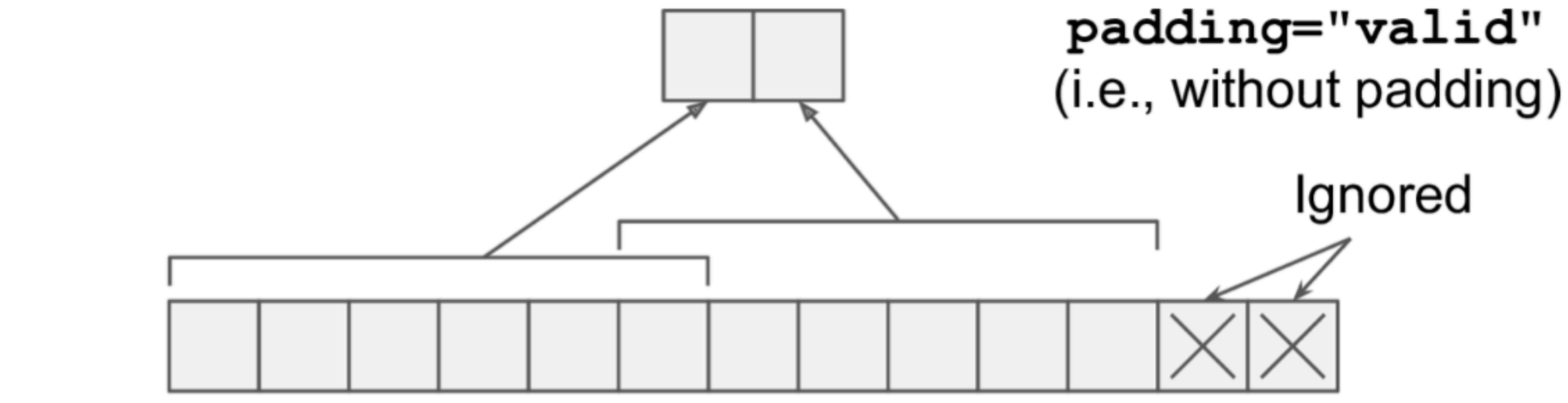
Examples

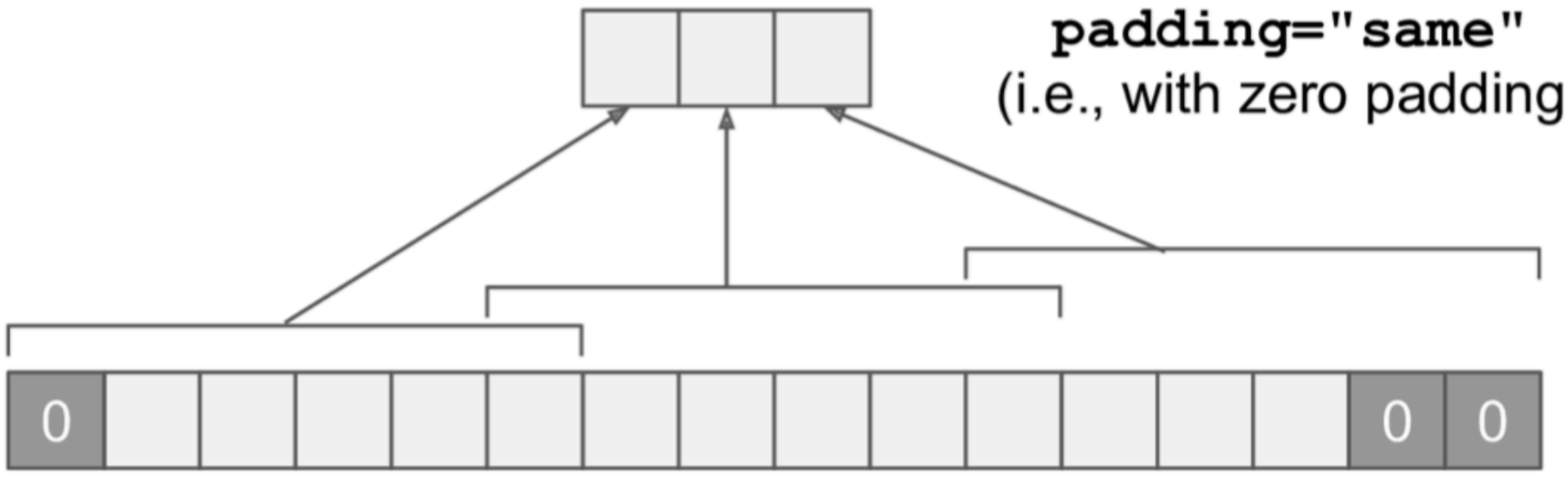
- Let us consider two possible sets of weights, i.e., filters
  - We obtain feature maps, highlighting the areas in an image that activate the filter the most



Two different kinds of padding

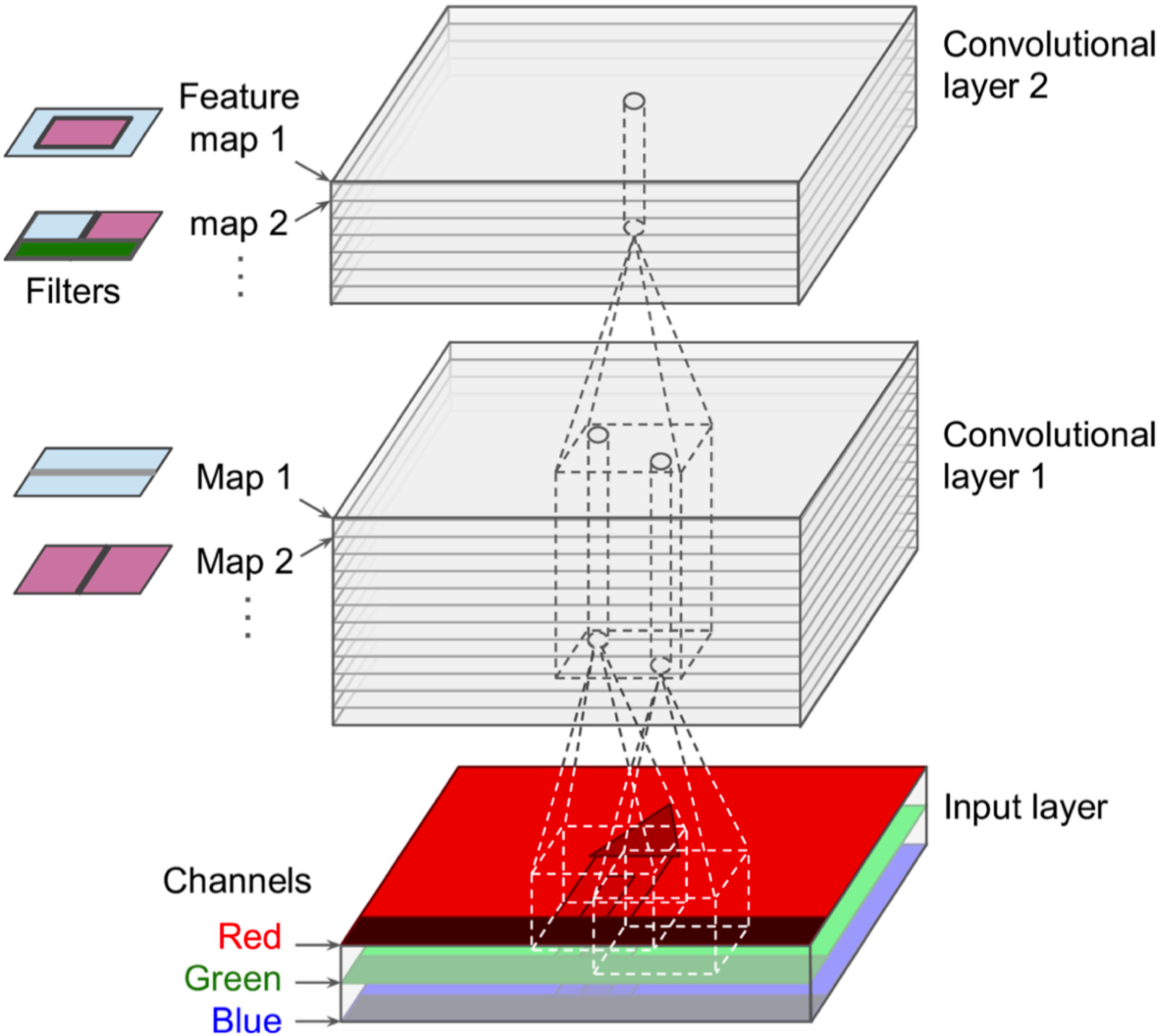
- "SAME" vs "VALID" padding (input width=13, filter width=6, stride=5)





Stacking multiple feature maps

- The number of parameters in CNNs is dramatically reduced because all neurons in a feature map share the same parameters for each filter



Mathematical representation

- We calculate the weighted sum of all the inputs, plus the bias term

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

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

- $x_{i',j',k'}$ is the output of the neuron located in layer $l - 1$, row i', column j', feature map k' (or channel k' if the previous layer is the input layer).
- b_k is the bias term for feature map k (in layer l). You can think of it as a knob that tweaks the overall brightness of the feature map k.
- $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer l and its input located at row u, column v (relative to the neuron's receptive field), and feature map k'.

Keras Implementation

- We can use the keras.layers.Conv2D

```
conv = keras.layers.Conv2D(filters=32, kernel_size=3, strides=1,
                           padding="same", activation="relu")
```

- 32 filters each 3x3 using a stride of 1 (both horizontally and vertically) and “same” padding, and then we apply the ReLU activation function
- Number of filters, their height and width
- Strides
- Padding type

Pooling layer

- Like convolutional layer, each neuron in a pooling layer is connected to a small rectangular receptive field. However, its goal is to aggregate the inputs using a function such as the max or mean

- The idea is to shrink the input image to reduce the computational load and memory usage

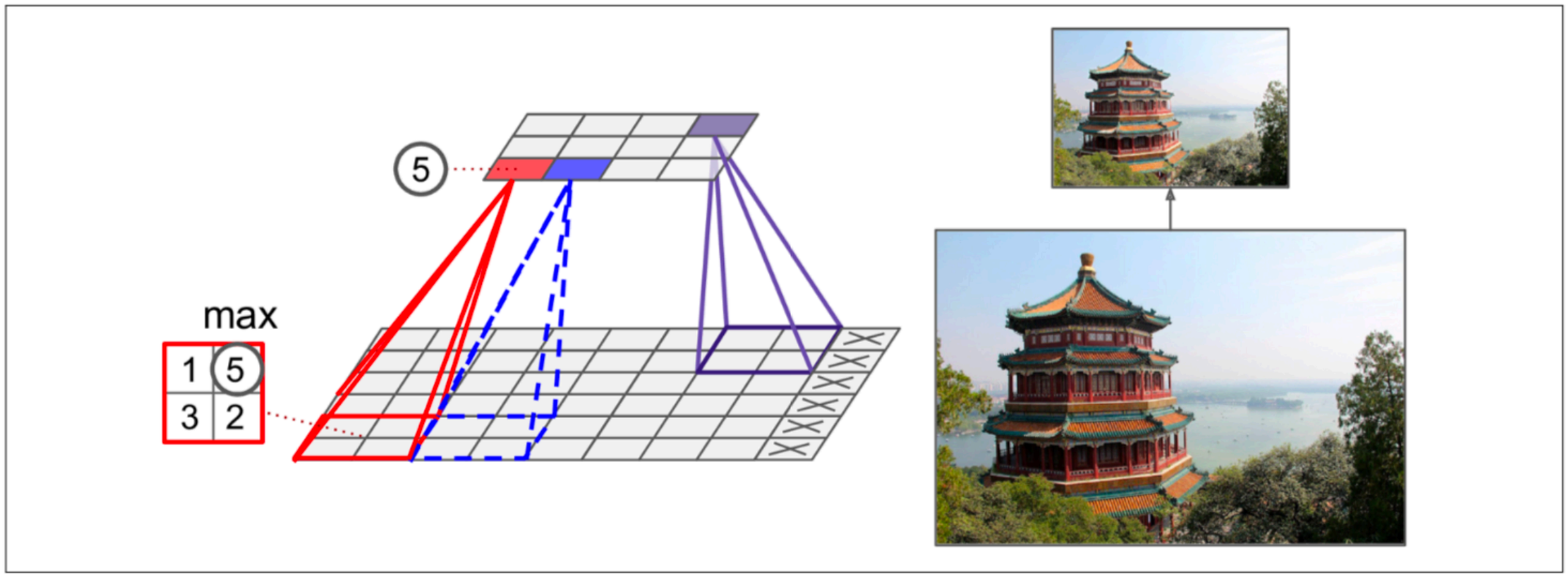
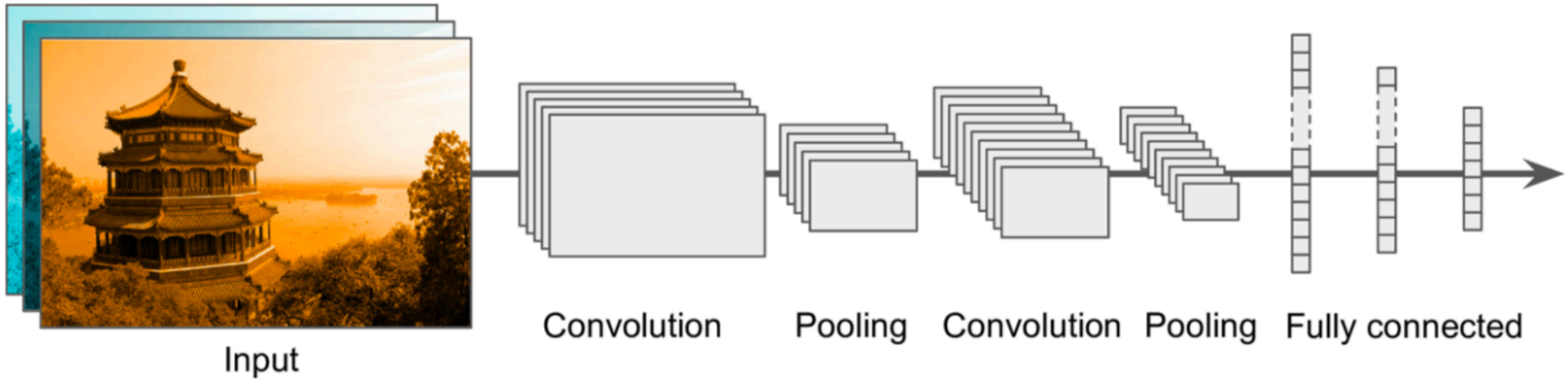


Figure 14-8. Max pooling layer (2 × 2 pooling kernel, stride 2, no padding)

```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```

CNN architectures

- Typically, a few convolutional layers, then a pooling layer, and then convolutional layers,…
- A regular feedforward neural network is added composed of fully connected layers
- Final layer outputs the prediction (e.g., a softmax layer that outputs estimated class probabilities)

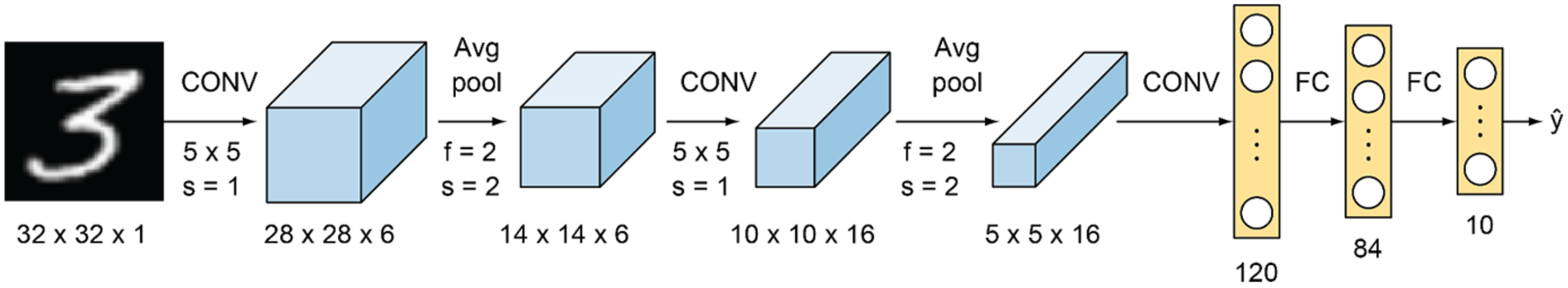


Keras implementation

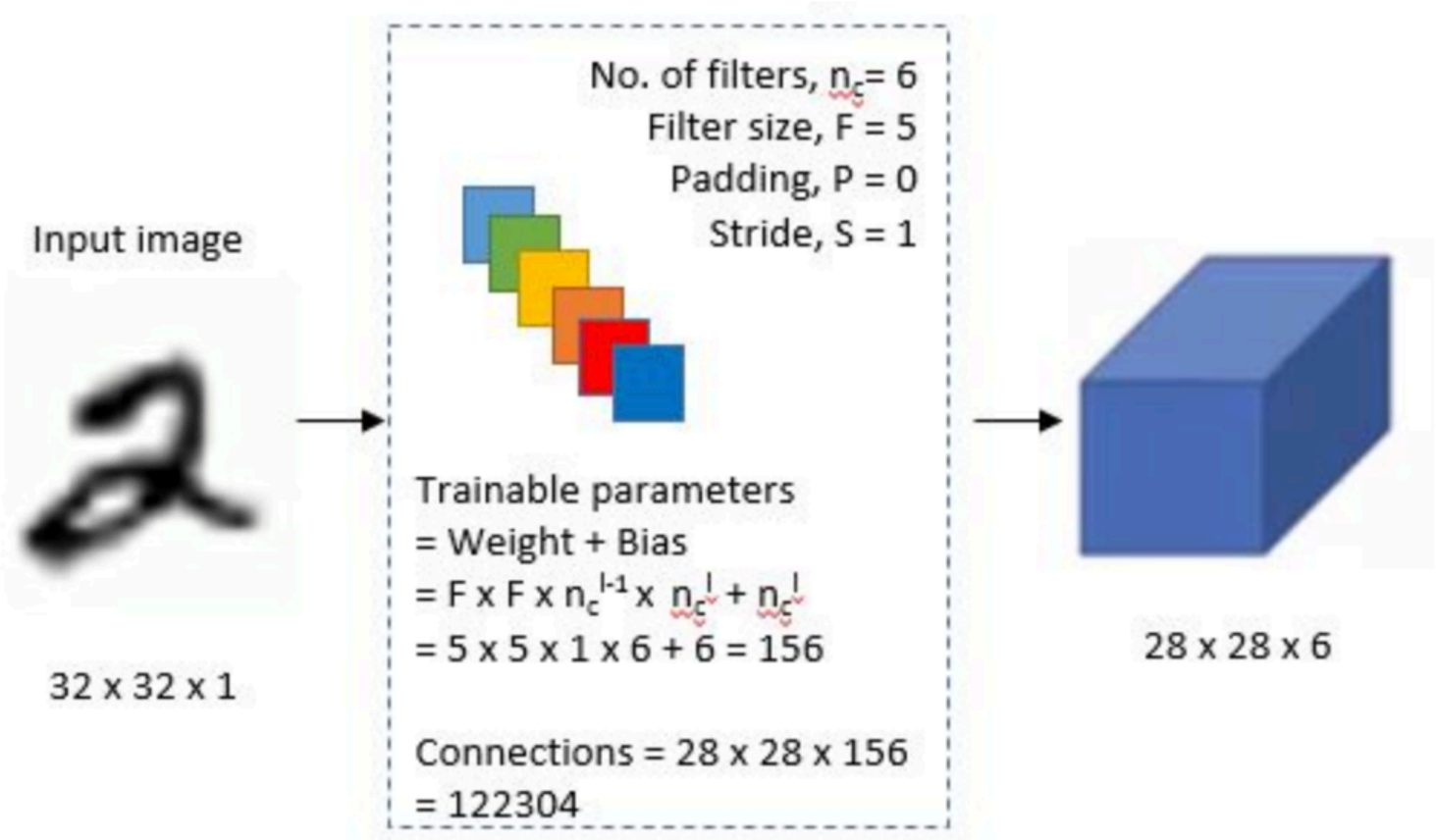
```
model = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same",
                        input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu",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

Popular architectures

- LeNet-5 Architecture

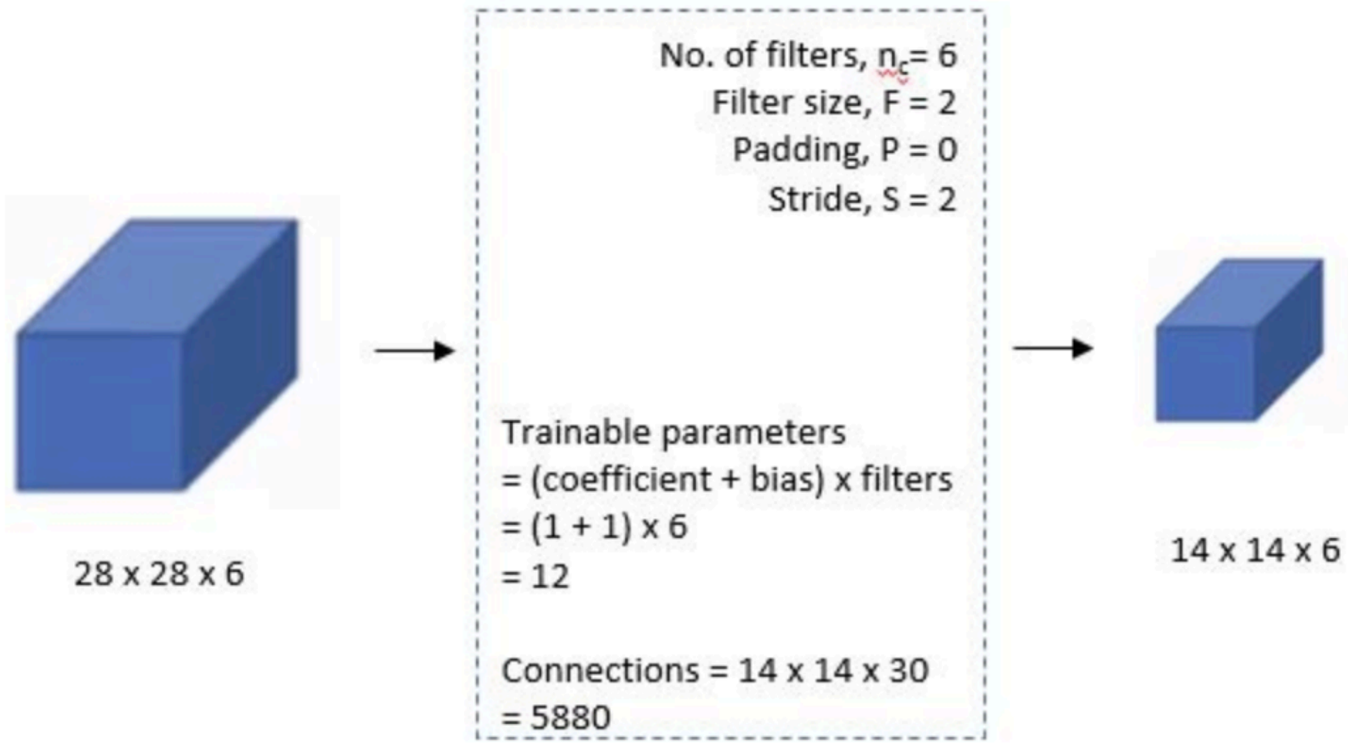


Some known architectures



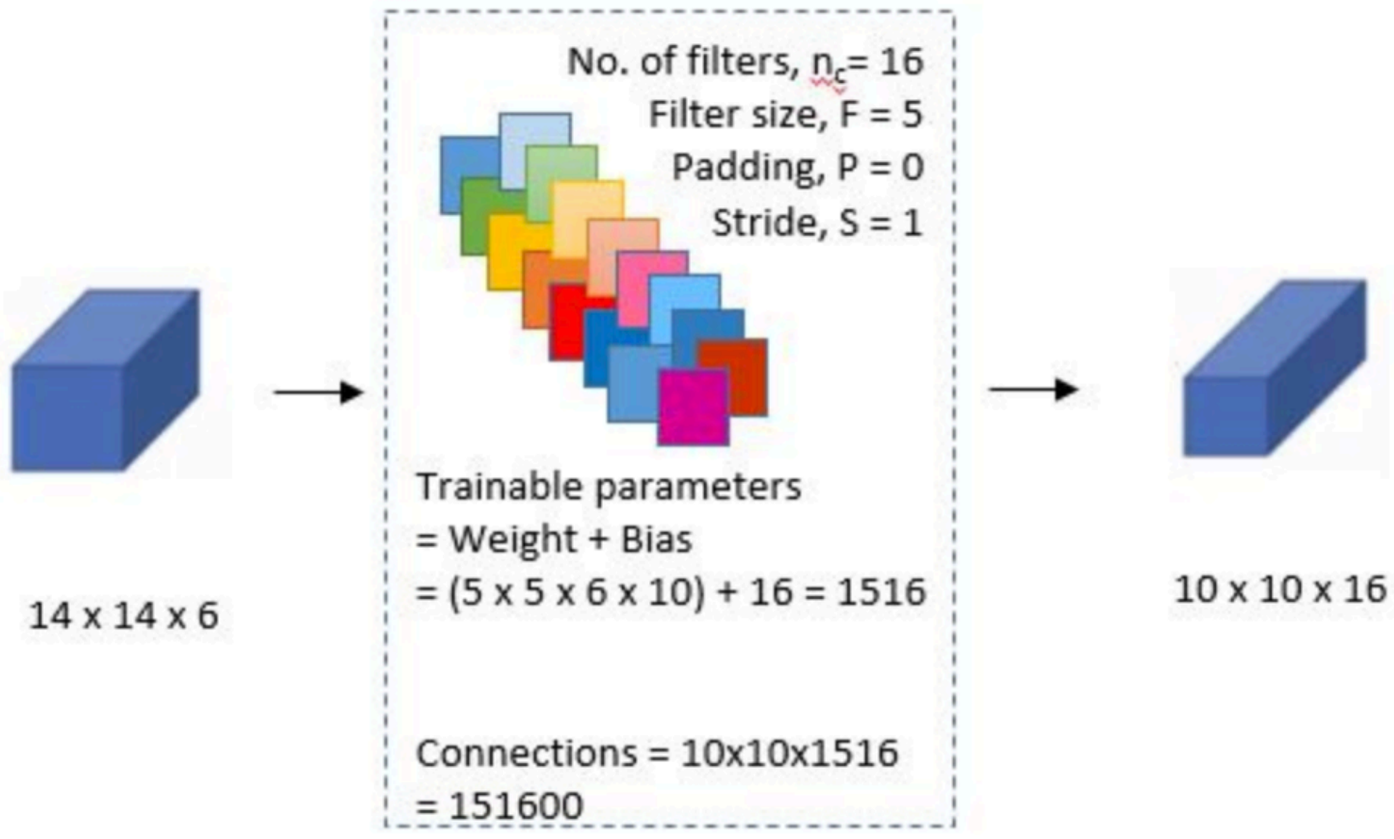
C1: Convolutional Layer

Some known architectures



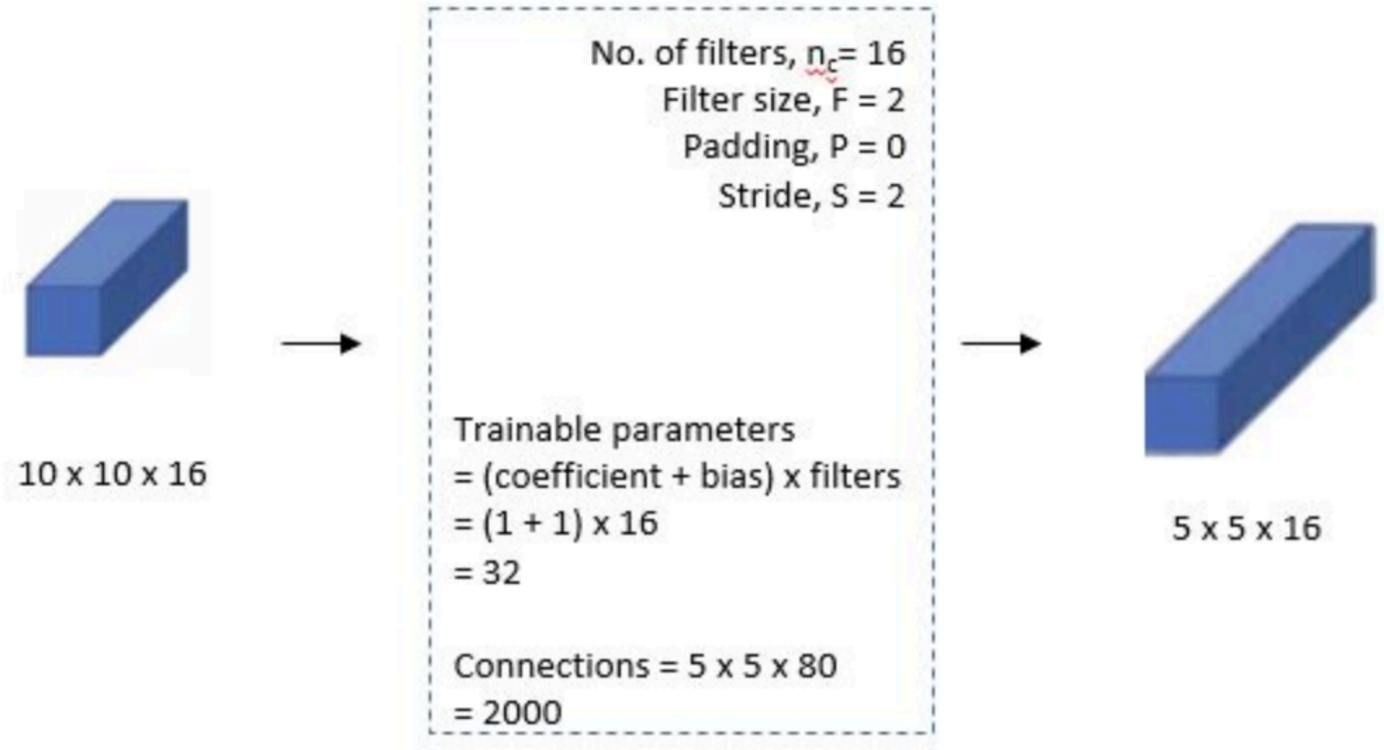
S2: Average Pooling Layer

Some known architectures



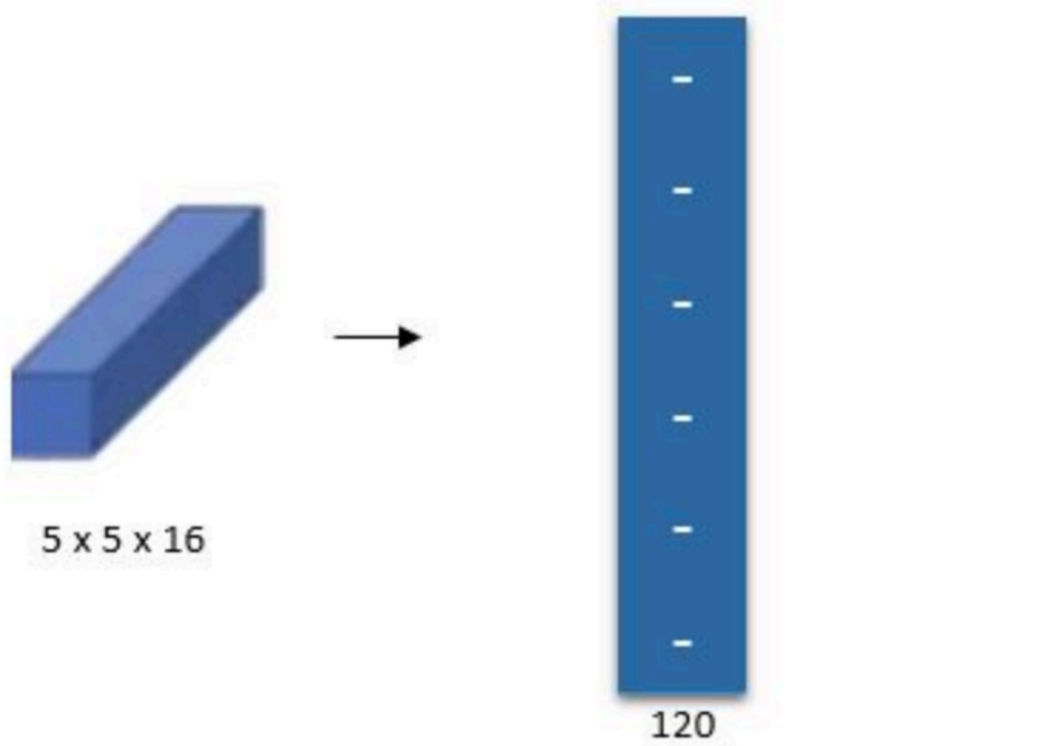
C3: Convolutional Layer

Some known architectures



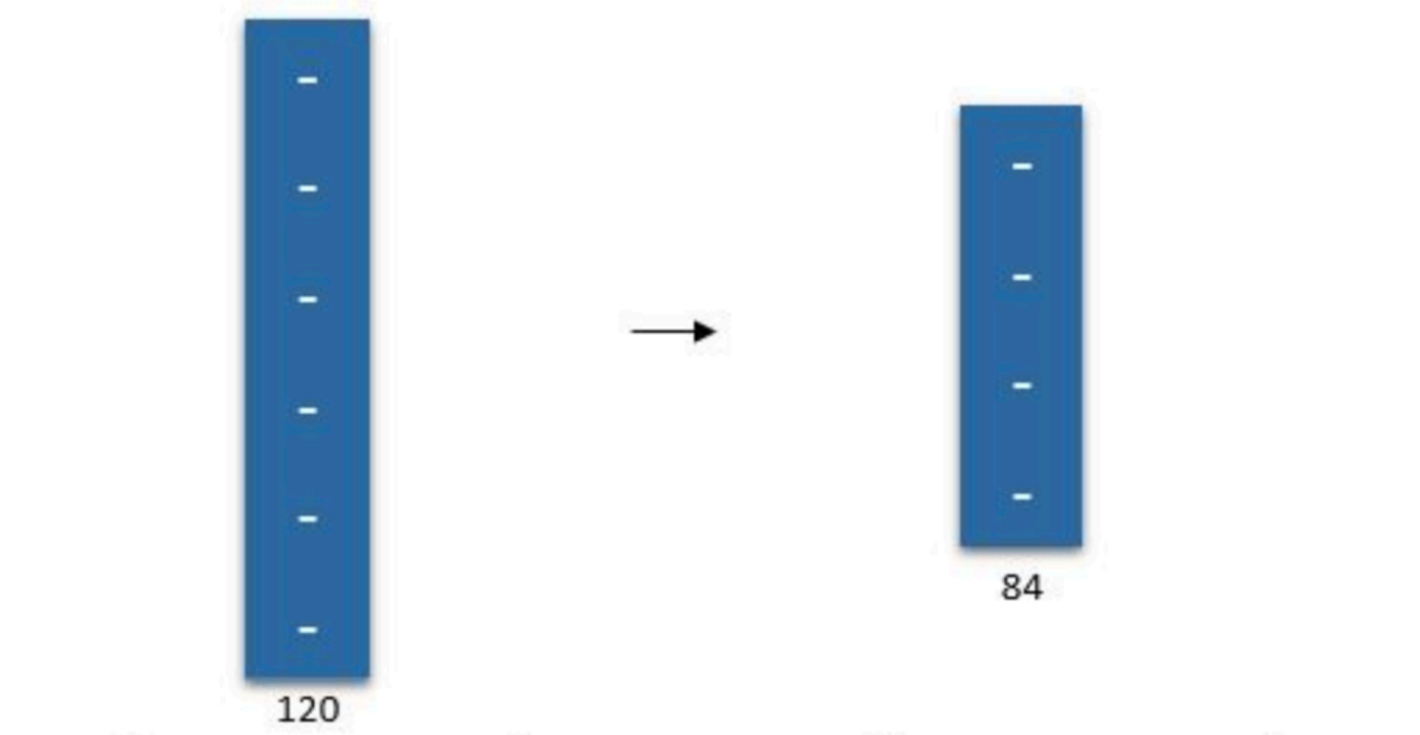
S4: Average Pooling Layer

Some known architectures



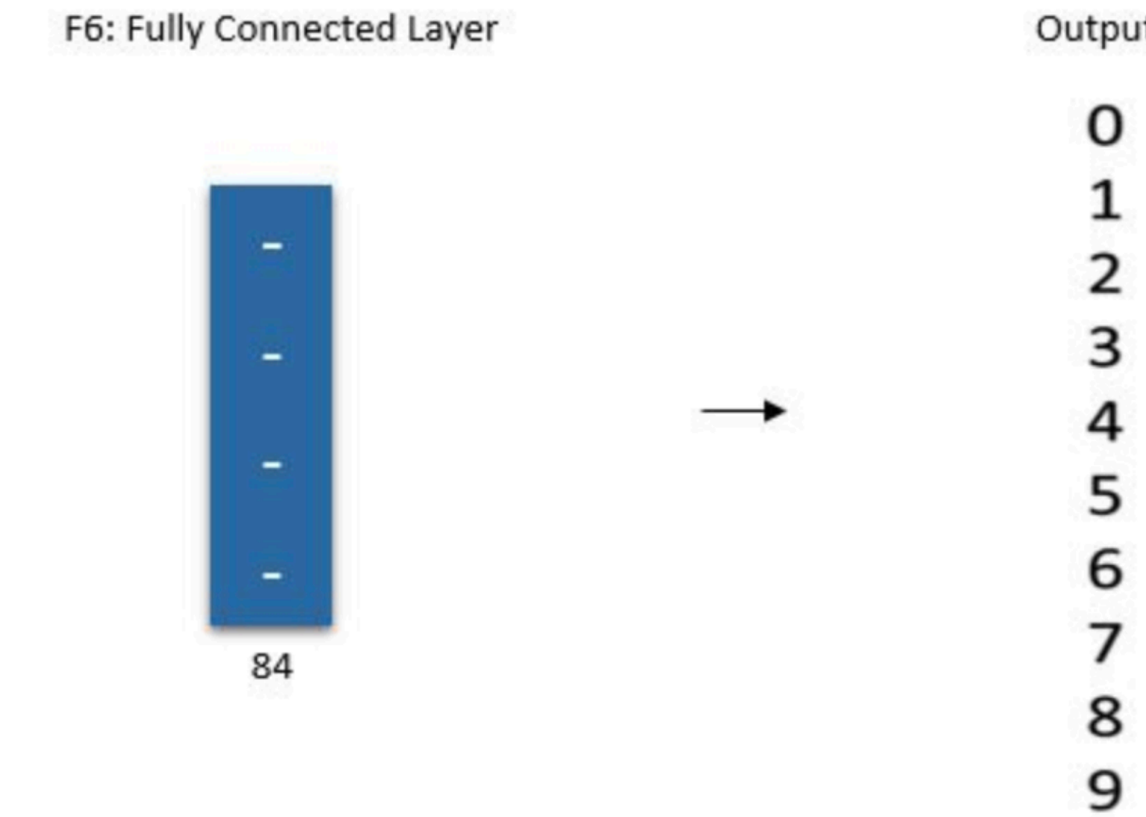
C5: Fully Connected Convolutional Layer

Some known architectures



F6: Fully Connected Layer

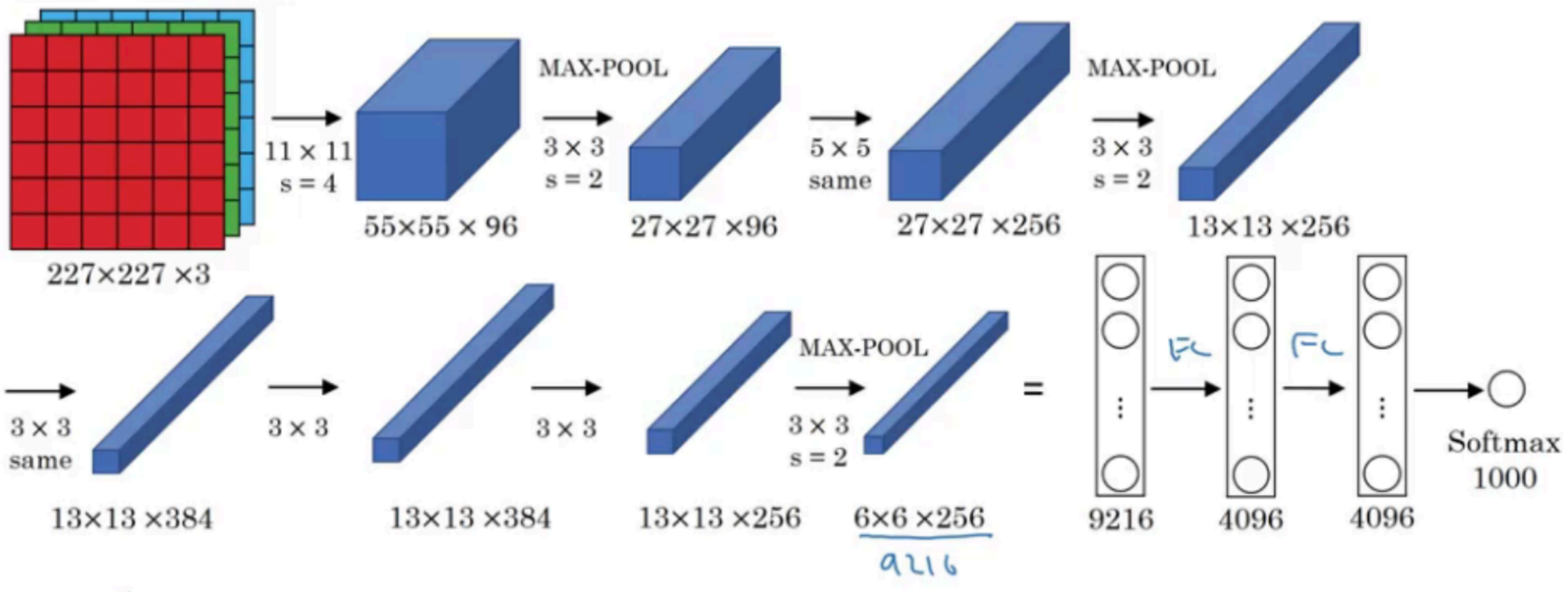
Some known architectures



Fully Connected Output Layer

Popular architectures

AlexNet



This network is similar to LeNet-5 with just more convolution and pooling layers:

- **Parameters:** 60 million
- **Activation function:** ReLu