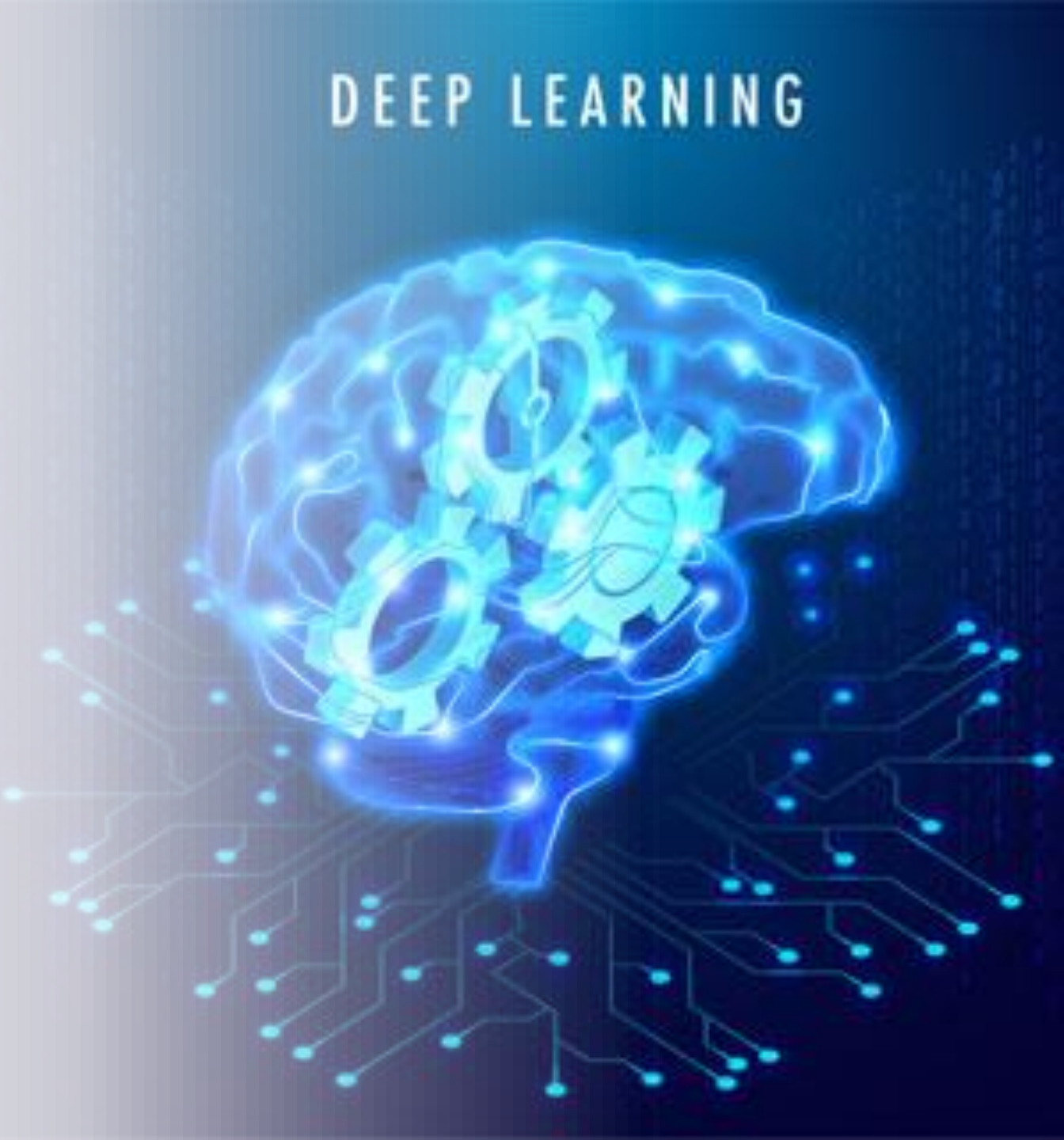
Convolutional Neural Networks

Dr. Huseyin Kusetogullari

Previously on DV2586

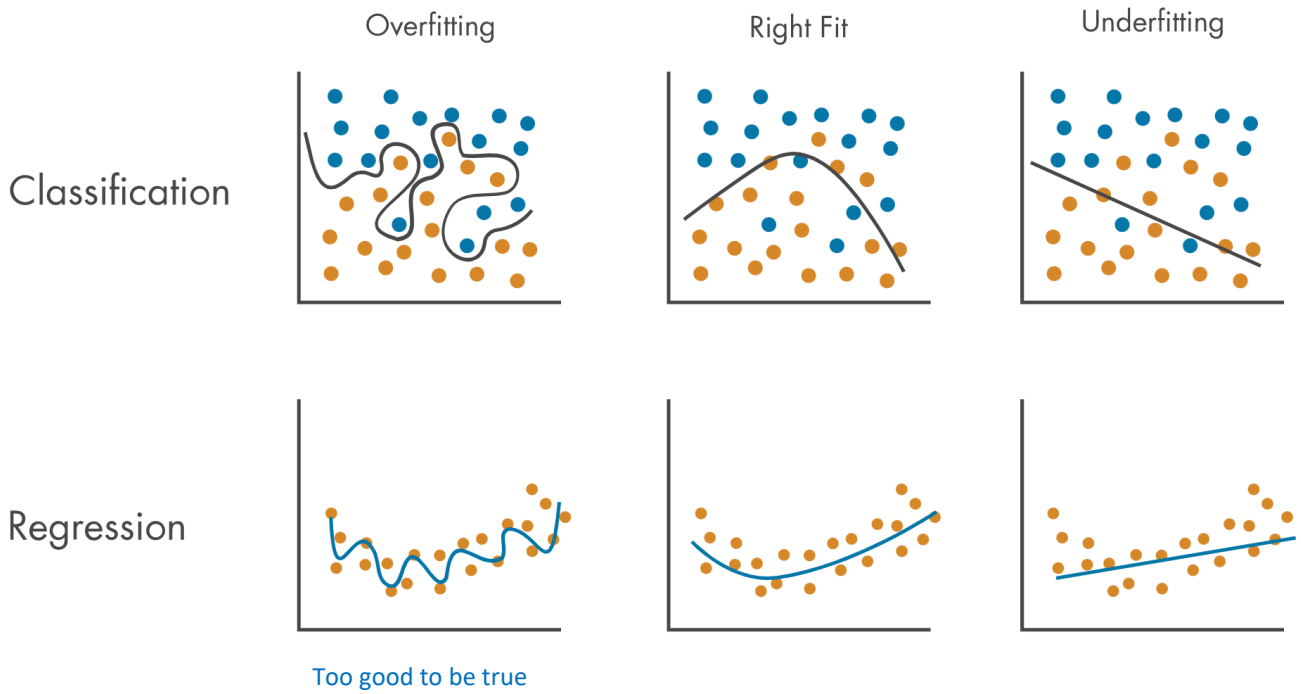
- Normalization
- Activation functions and vanishing problem
- Gradient Descents
- Cost function
- Adam optimizer
- Softmax



News

- Assignments will be available tomorrow
- Project topic searching
  - We will have 1 hour discussion about the project topics in the 5th Lecture (April 14th)

Overfitting - Underfitting



Too good to be true

When only looking at the computed error of a machine learning model for the training data, overfitting is harder to detect than underfitting. So, to avoid overfitting, it is important to validate a deep learning model before using it on test data.

Regularization is a technique used to prevent statistical overfitting in a deep learning model

* Overfitting occurs when a model learns the training data too well, capturing noise and random fluctuations in the training data instead of the actual underlying patterns.
* As a result, the model performs well on the training data but poorly on new, unseen data.

https://www.mathworks.com/discovery/overfitting.html

* How to prevent overfitting?

Regularization

- Regularization is a technique used in deep learning to prevent overfitting and improve the generalization performance of a model.

- Overfitting occurs when a model becomes too complex and learns to fit the training data too well, resulting in poor performance on new, unseen data.

- There are several reasons why regularization is important in deep learning:

- **Complexity of Models:** Deep learning models, especially deep neural networks, have a large number of parameters. This complexity makes them highly flexible and capable of learning intricate patterns in the data. However, it also makes them prone to overfitting, especially when the amount of training data is limited compared to the number of parameters.

- **Generalization:** The ultimate goal of a machine learning model is to make accurate predictions on new, unseen data. Regularization encourages the model to focus on the most important patterns and to generalize better from the training data to unseen data.

- **Noise in Data:** Real-world data often contains noise. Without regularization, a model might learn to fit the noise rather than the underlying data distribution, which can degrade its performance on new data.

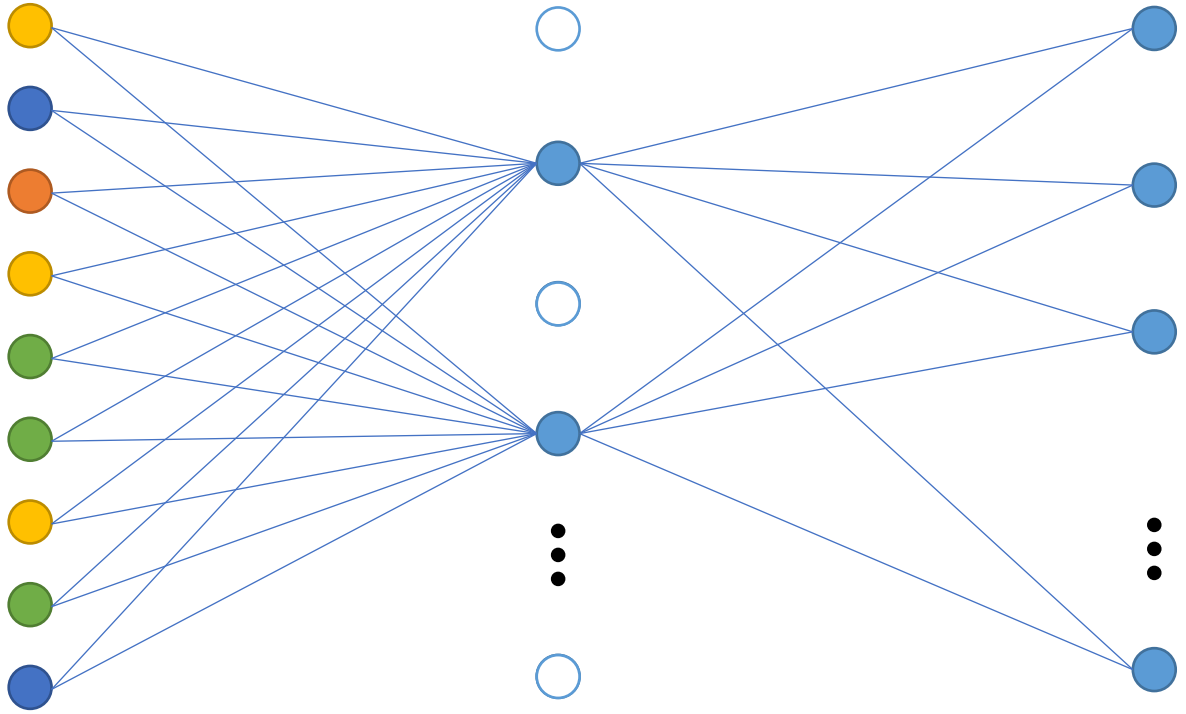
Cont.

- There are several regularization techniques used in deep learning, including dropout and batch normalization.

- 1-Dropout: Dropout is a technique where randomly selected neurons are dropped out or ignored during training. This forces the network to learn more robust features, as it cannot rely on any single neuron to always be present.

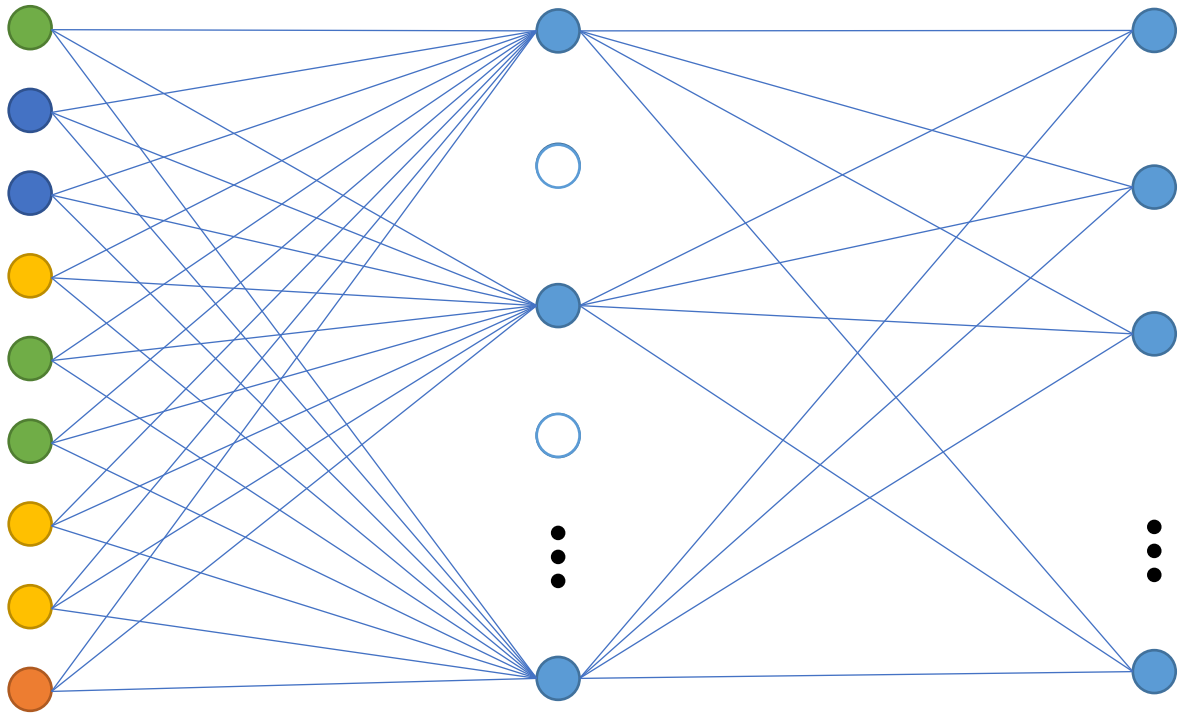
Dropout –randomly drops activations during training

Instance 1



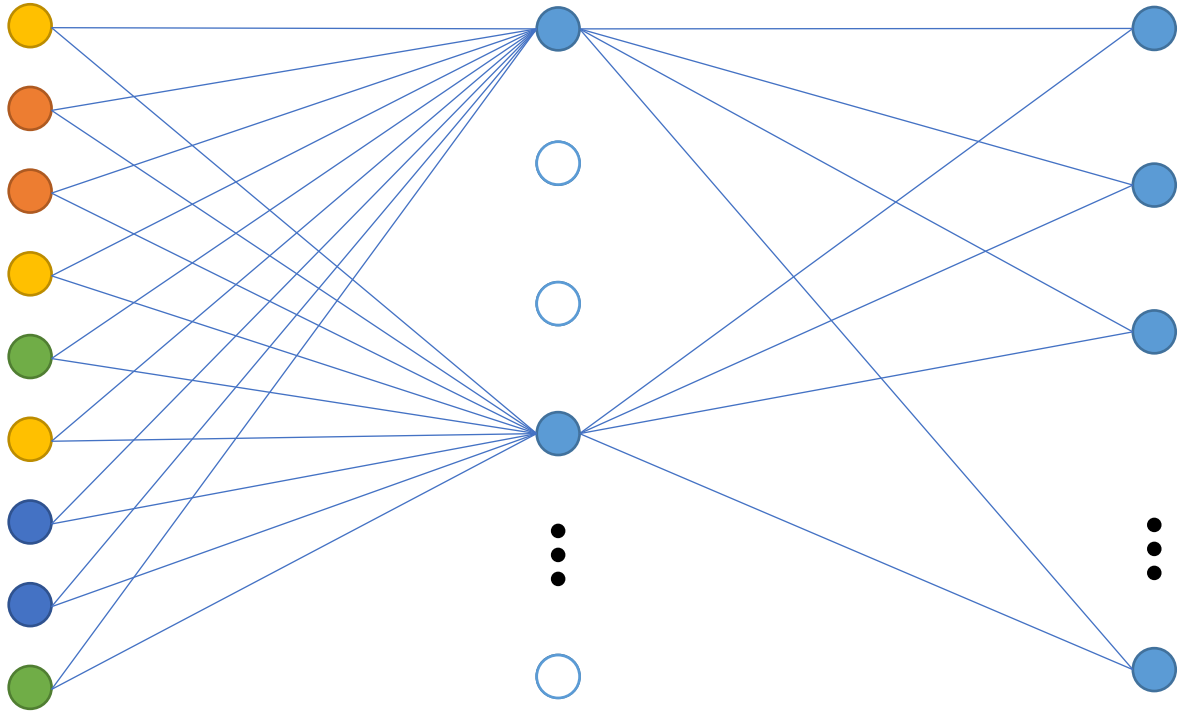
Dropout –randomly drops activations during training

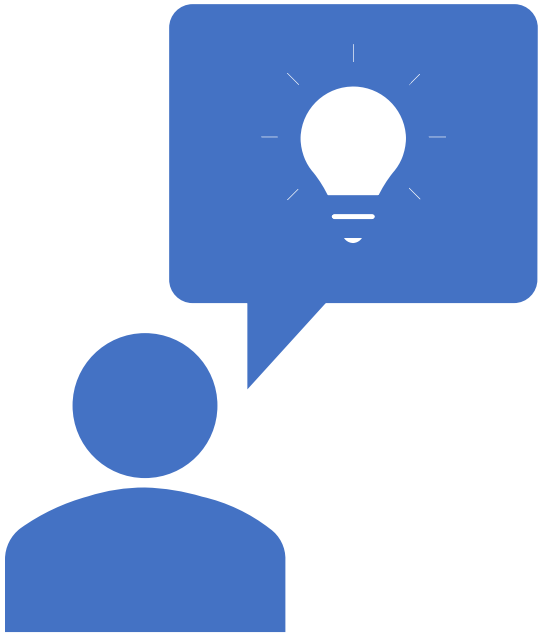
Instance 2



Dropout –randomly drops activations during training

Instance 3





Dropout

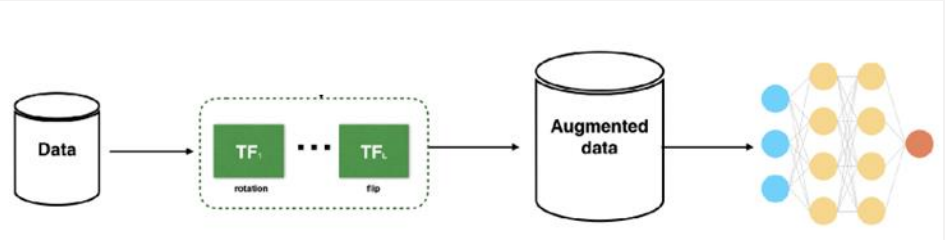
- Idea: randomly drop activations during training
- Benefit: reduces overfitting and improves generalization
- Dropout forces a neural network to learn more robust features
  - 0.2<= Dropout rate <=0.5
- Can be implemented as a layer

2-Early Stopping

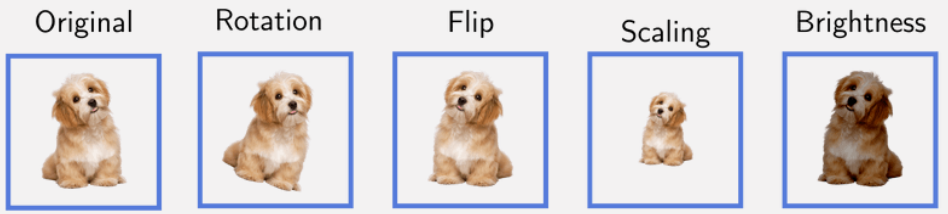
- Another regularization technique to prevent overfitting.
- Early stopping involves monitoring the model's performance on a validation set and stopping training when the performance starts to degrade (e.g., when the validation error begins to increase), effectively preventing overfitting.

- 1. Train the model.
- 2. Monitor validation loss (or accuracy).
- 3. If the model stops improving for a number of consecutive epochs (called patience), stop training early.
- 4. Optionally, restore the best weights.

4-Data Augmentation



- In data augmentation, the training data is artificially increased by applying random transformations (e.g., rotation, scaling, cropping).
- This increases the amount of data available for training and can help to prevent overfitting.
- This can help the model generalize better, effectively serving as a form of regularization.



Data Augmentation

```python
from PIL import Image, ImageOps
import matplotlib.pyplot as plt

def show_images(images, titles=None):
    """Utility function to display images in a grid."""
    n = len(images)
    fig, axs = plt.subplots(1, n, figsize=(15, 5))
    for i in range(n):
        axs[i].imshow(images[i], cmap='gray')
        axs[i].axis('off')
        if titles:
            axs[i].set_title(titles[i])
    plt.show()

def augment_image(image_path):
    """Load an image and apply augmentation."""
    # Load the image
    original_image = Image.open(image_path)

    # Rotate the image
    rotated_image = original_image.rotate(45) # Rotate by 45 degrees

    # Flip the image horizontally
    flipped_image = ImageOps.mirror(original_image)

    # Display the original and augmented images
    show_images([original_image, rotated_image, flipped_image],
                titles=['Original', 'Rotated (45°)', 'Flipped (Horizontal)'])

# Replace 'path/to/your/image.jpg' with the actual path to your image file
image_path = 'path/to/your/image.jpg'
```

Others

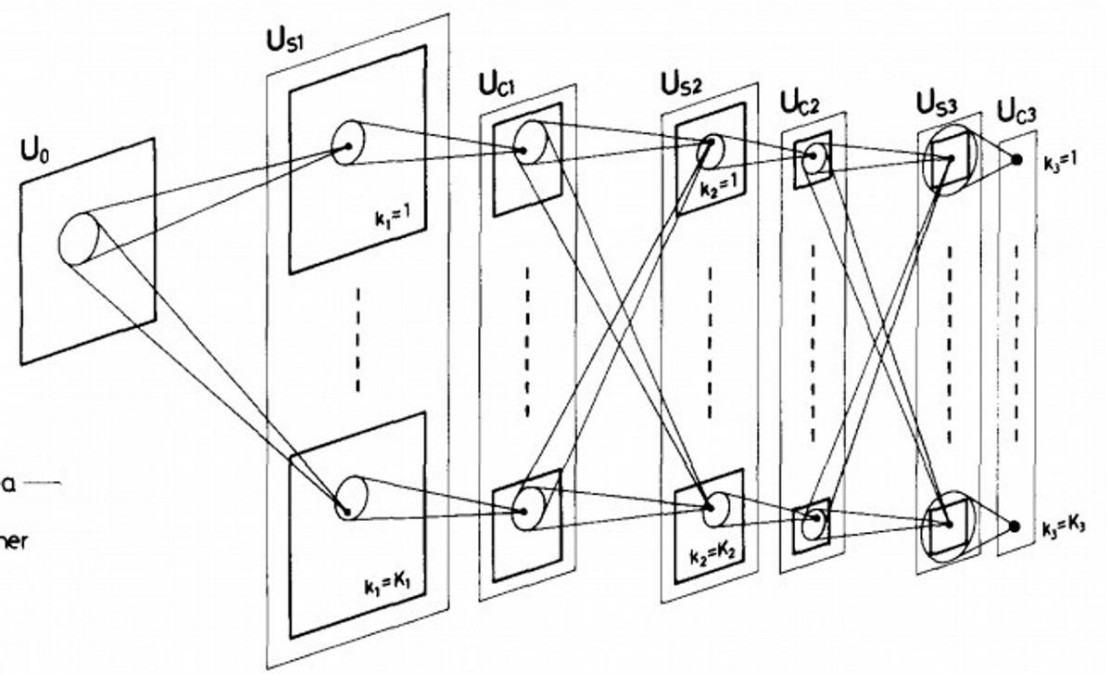
- L1/L2 Regulazation
- Elastic Net
- Noise Injection

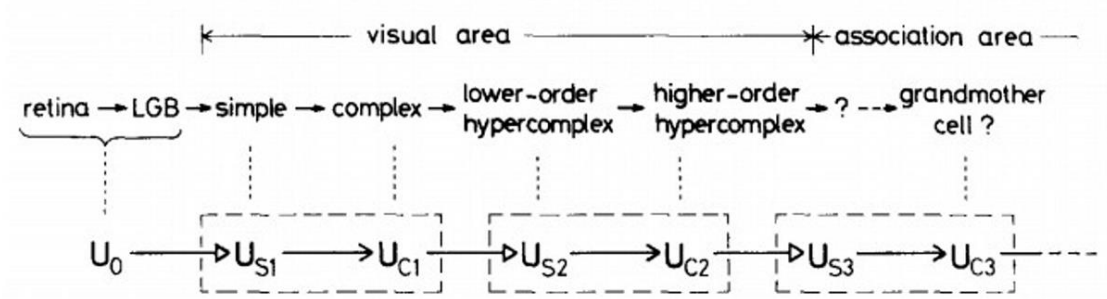
These techniques can be used in combination or individually depending on the specific problem and the characteristics of the data.

How to design better deep learning models?

Convolutional Neural Networks (CNNs)

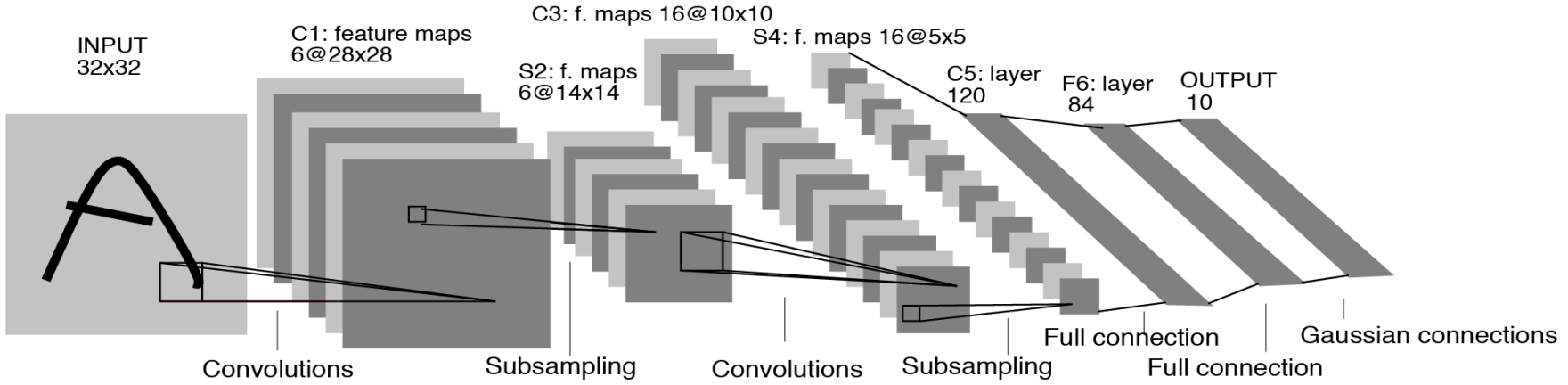
- Neocognitron model by Fukushima (1980)
- The first convolutional neural network (CNN) model
- so-called "sandwich" architecture
  - simple cells act like filters
  - complex cells perform pooling
- Difficult to train
  - No backpropagation yet





CNNs

- One of the first true CNNs by Yann LeCun in LeNet-5
- Designed for digit recognition (e.g., reading ZIP codes and checks).
- Combined convolutional layers, subsampling (pooling), and fully connected layers.
- Training used backpropagation, a major milestone in deep learning.



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE. **86** (11): 2278–2324, 1998.

2012 – AlexNet Breakthrough

- **AlexNet**, created by **Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton**, won the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** by a wide margin.
- Introduced innovations like:
  - **ReLU** activations
  - **Dropout** for regularization
  - GPU training
- This was the **turning point** for deep learning and CNN adoption.

CNNs

- CNNs are very similar to the ordinary NNs.
- They are also made up of neurons that have learnable weights and biases.
- Steps in CNN:
  - **INPUT**
  - **CONVOLUTIONAL layer**
  - **RECTIFIED LINEAR UNITS (RELU)**
  - **POOLING Layer**
  - **FULLY CONNECTED** (i.e. fully-connected NNs) **layer**
  - **Softmax Layer**

Convolution

- Convolution = Spatial filtering

$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

- Different filters (weights) reveal a different characteristics of the input.

$*^{1/8}$

0	1	0
1	4	1
0	1	0

Convolution

- Convolution = Spatial filtering

$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

- Different filters (weights) reveal a different characteristics of the input.

$*$

0	-1	0
-1	4	-1
0	-1	0

Convolution

- Convolution = Spatial filtering

$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

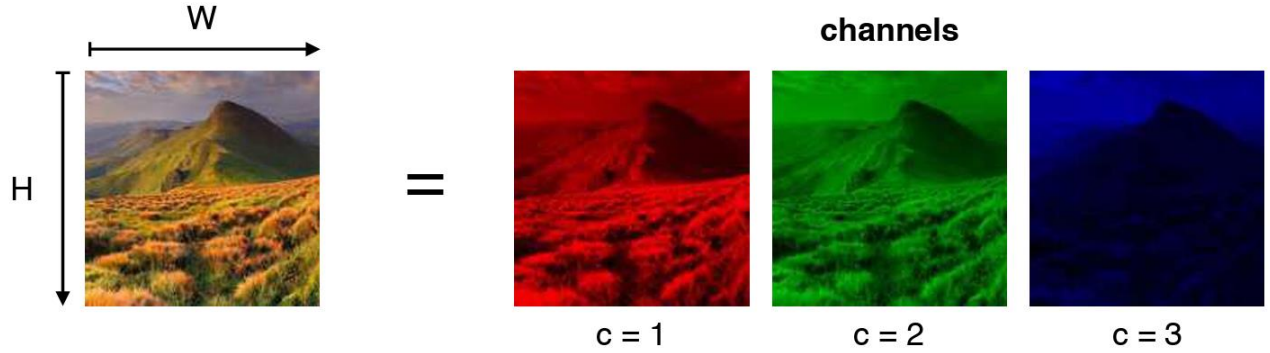
- Different filters (weights) reveal a different characteristics of the input.

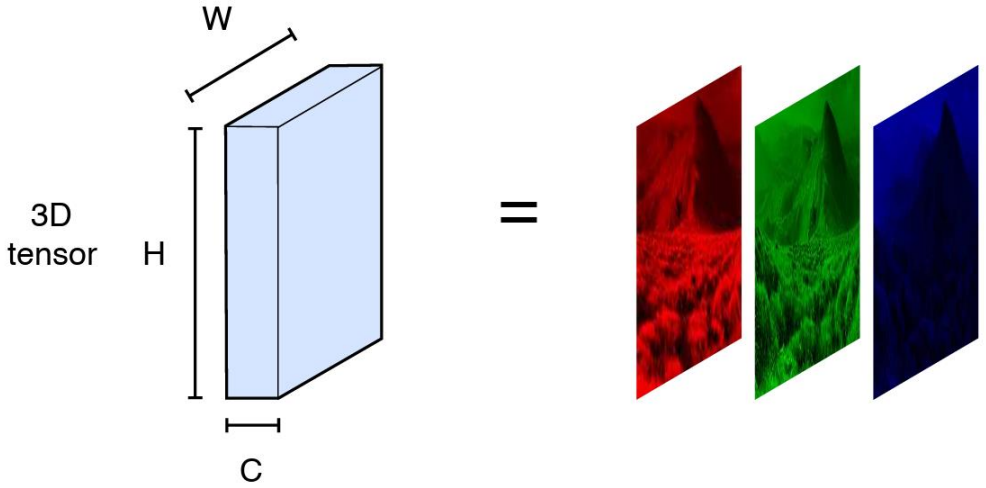
$*$

1	0	-1
2	0	-2
1	0	-1

Data = 3D Tensors

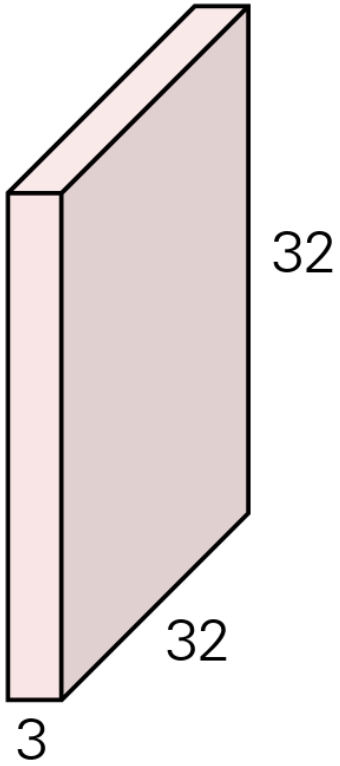
- There is a vector of feature channels (e.g. RGB) at each spatial location (pixel).



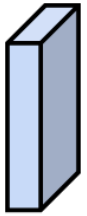


Convolutional Layer

32x32x3 input

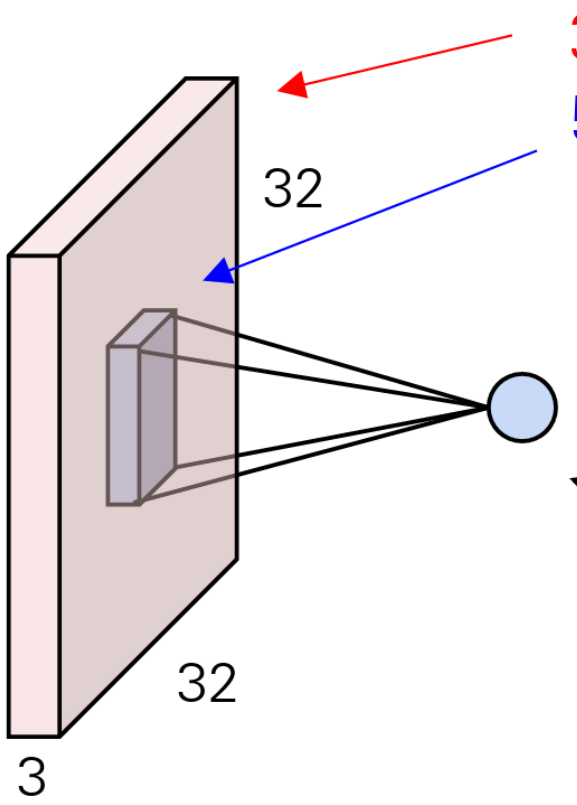


5x5x3 filter



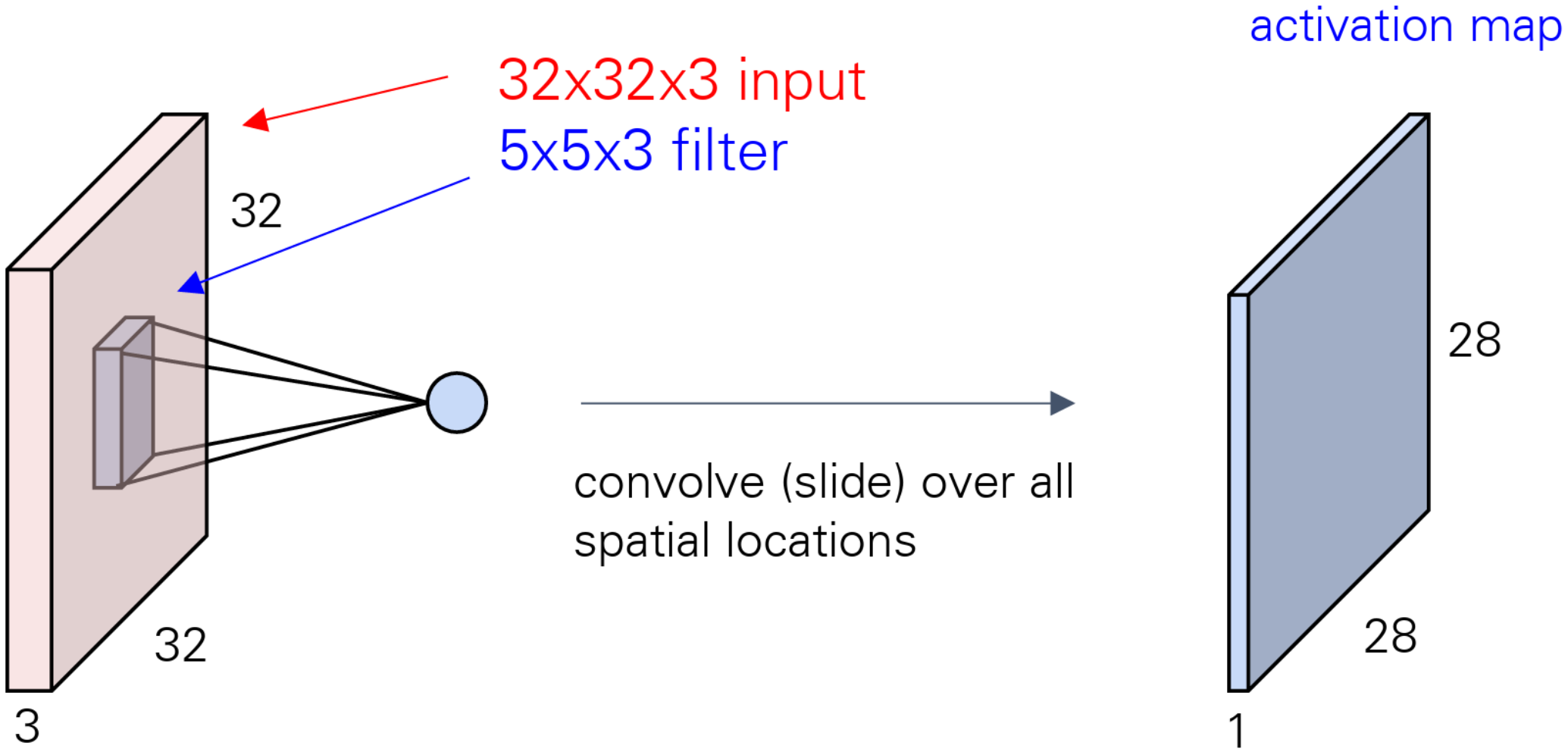
Convolve the filter with the input
i.e. “slide over the image spatially,
computing dot products”

Convolutional Layer



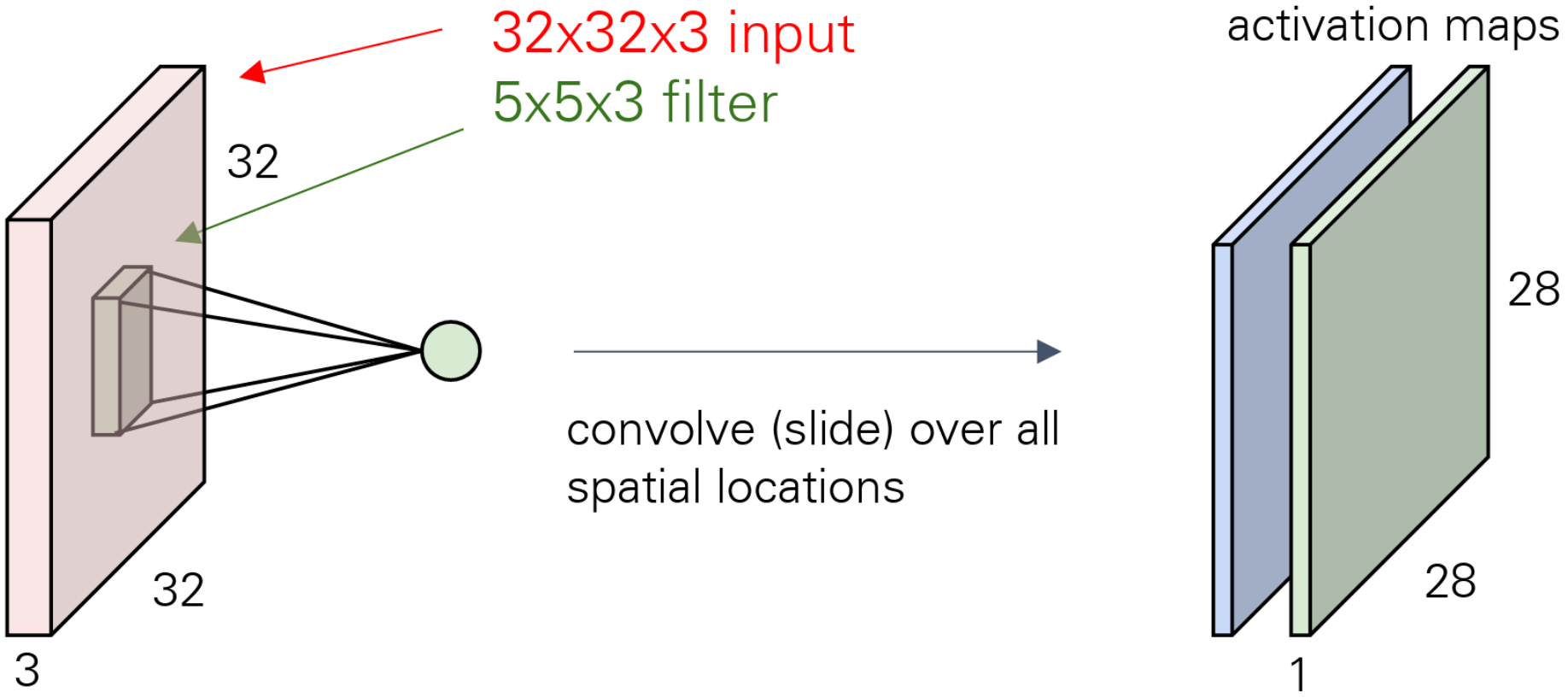
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the input
(i.e. 5*5*3 = 75-dimensional dot product + bias)

Convolutional Layer



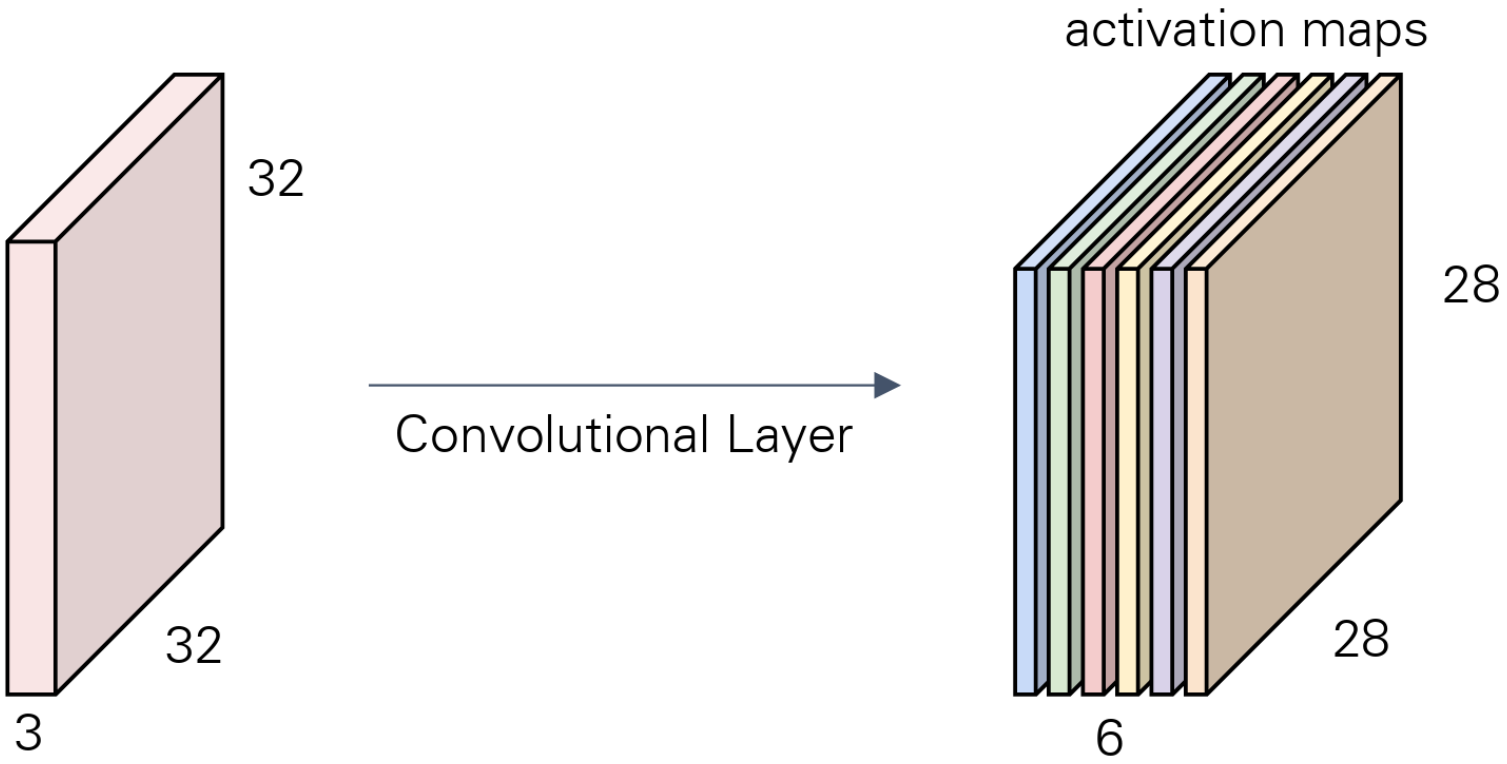
Convolutional Layer

consider a second, green filter



Convolutional Layer

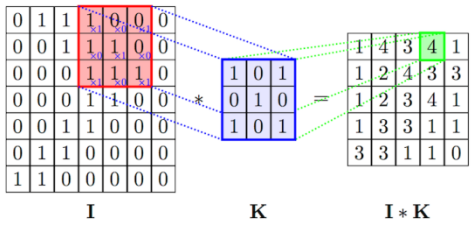
- Multiple filters produce multiple output channels
- For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



Convolution Layer

- Filters or Kernels are playing a key role in convolutional neural networks
- Kernels are helping to create a feature set of the input images.
- The kernel size can be 3x3, 5x5 or 7x7 etc.
- Any number of kernel can be selected 6, 12, 25, 36, 128, etc.
- The kernel weights or values are randomly initialized and then updated based on the Gradient Descent method. Thus, optimal kernel filters will be obtained.

Method:
I: Input image
K: filter



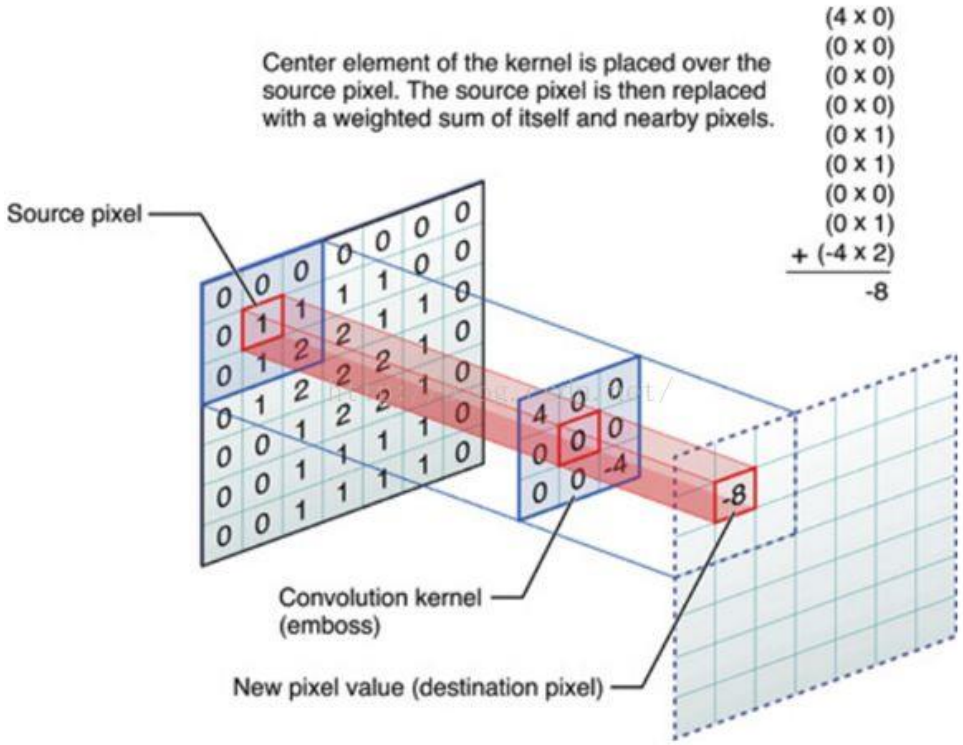
Conv1D layer

```
>>> # The inputs are 128-length vectors with 10 timesteps, and the batch size
>>> # is 4.
>>> input_shape = (4, 10, 128)
>>> x = tf.random.normal(input_shape)
>>> y = tf.keras.layers.Conv1D(
... 32, 3, activation='relu',input_shape=input_shape[1:])(x)
>>> print(y.shape)
(4, 8, 32)
```

Conv2D layer

```
>>> # The inputs are 28x28 RGB images with `channels_last` and the batch
>>> # size is 4.
>>> input_shape = (4, 28, 28, 3)
>>> x = tf.random.normal(input_shape)
>>> y = tf.keras.layers.Conv2D(
... 2, 3, activation='relu', input_shape=input_shape[1:])(x)
>>> print(y.shape)
(4, 26, 26, 2)
```

Convolutional kernel



Padding on the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input

Example

Example: An image is given with 5x5x1. Kernel size is 3x3x1 and two kernels have been used, stride=2 and padding=1.

Input Image: 5x5x1

x[:,:]

0	1	1	1	1
1	2	1	1	1
1	0	0	0	0
0	0	0	1	0
0	1	0	0	0

Filter W0 (3x3x1)

w0[:,:]

-1	1	0
-1	0	-1
-1	1	0

Filter W1 (3x3x1)

w1[:,:]

0	-1	-1
-1	1	1
1	-1	1

Convolution Layer: Example

Input Image: 5x5x1
Padding +1
The image will be 7x7x1
x[:,:]

0	0	0	0	0	0	0
0	0	1	1	1	1	0
0	1	2	1	1	1	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x1)
w0[:,:]

-1	1	0
-1	0	-1
-1	1	0

Filter W1 (3x3x1)
w1[:,:]

0	-1	-1
-1	1	1
1	-1	1

Input Image: 5x5x1
Padding +1
The image will be 7x7x1
x[:,:]

Filter W0 (3x3x1)
w0[:,:]

-1	1	0
-1	0	-1
-1	1	0

Bias b0 (1x1x1)
b0

1

Filter W1 (3x3x1)
w1[:,:]

0	-1	-1
-1	1	1
1	-1	1

Bias b1 (1x1x1)
b1

0

Output Volume (3x3x2)
o[:,:]

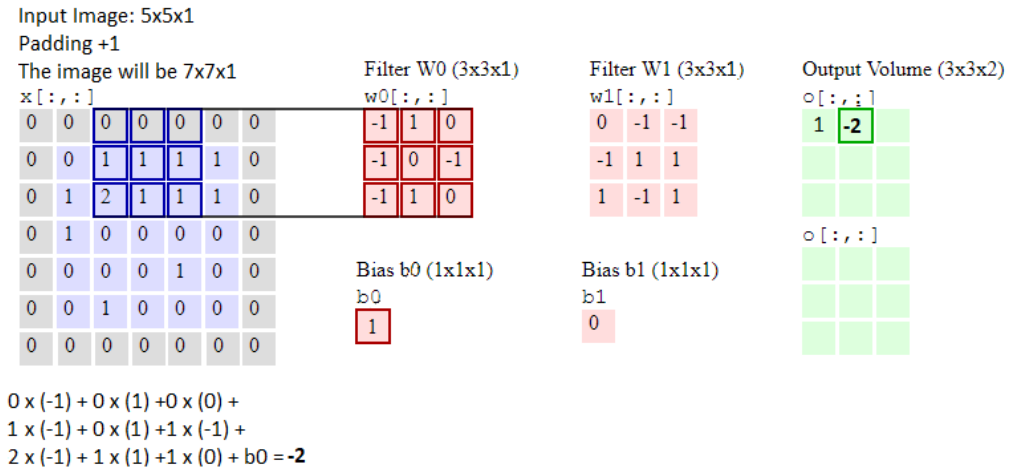
1		

o[:,:]

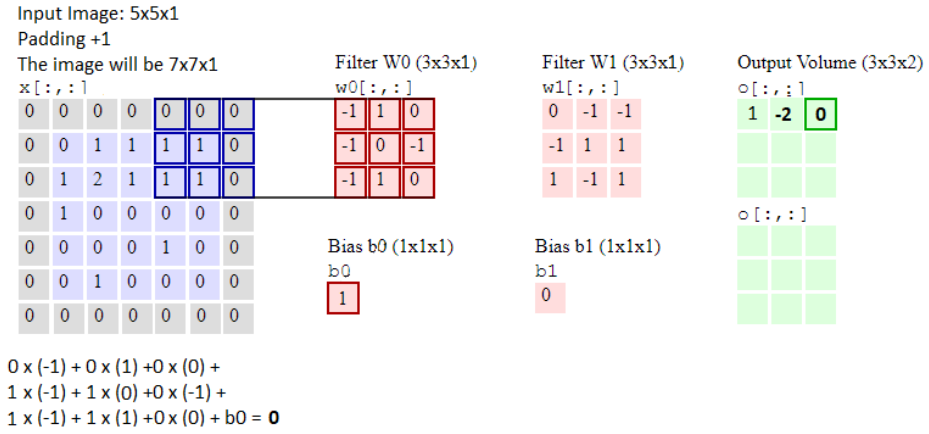
Stride 2 →

0 x (-1) + 0 x (1) +0 x (0) +
0 x (-1) + 0 x (0) +1 x (-1) +
0 x (-1) + 1 x (1) +2 x (0) + b0 = **1**

Convolution Layer: Example

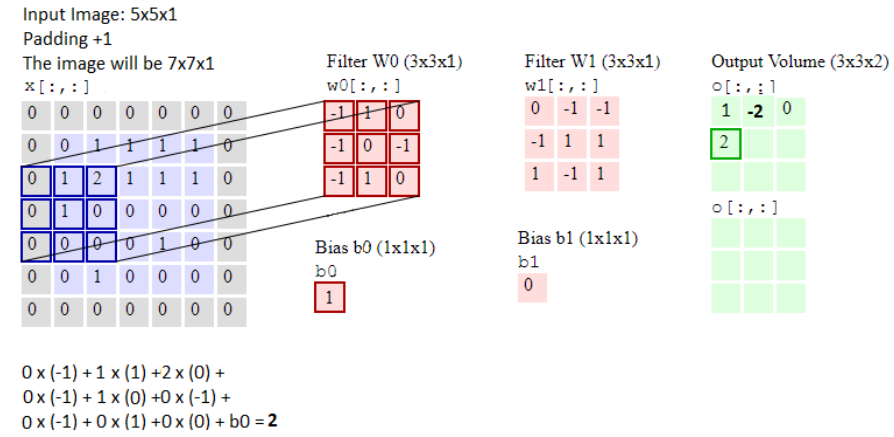


Stride 2

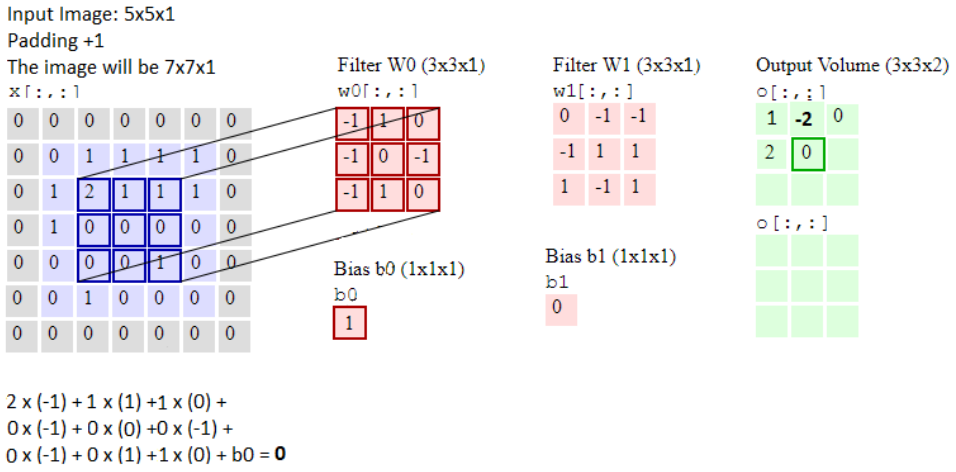


Stride 2

Convolution Layer: Example

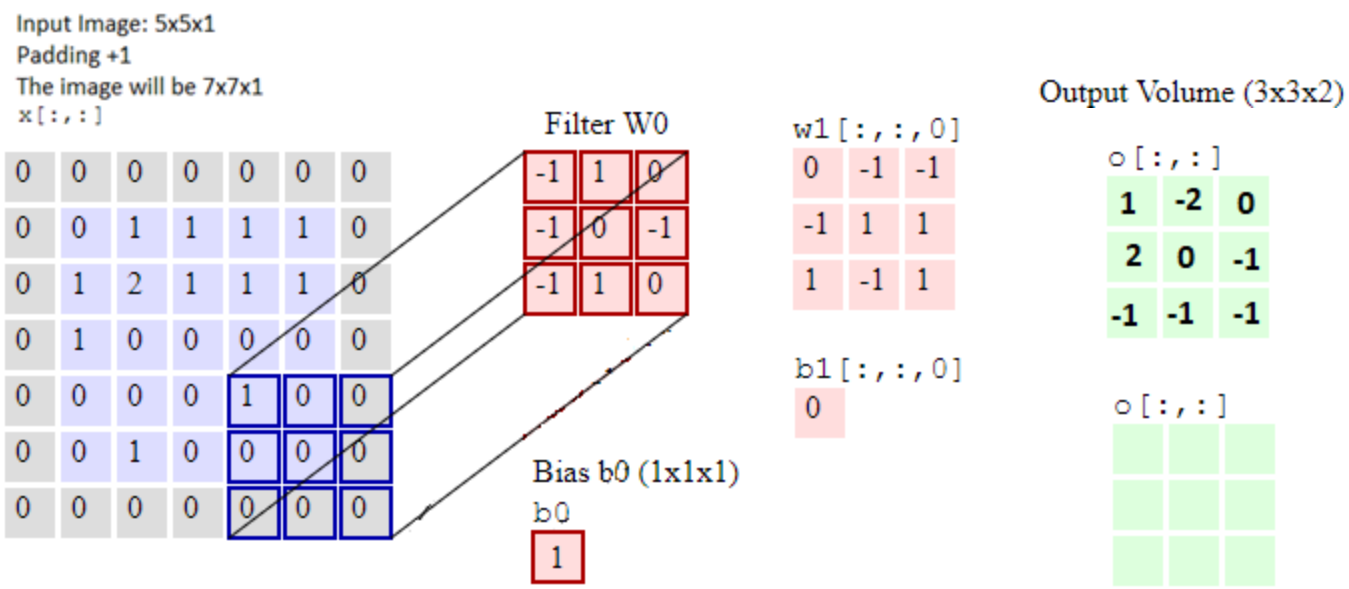


Stride 2

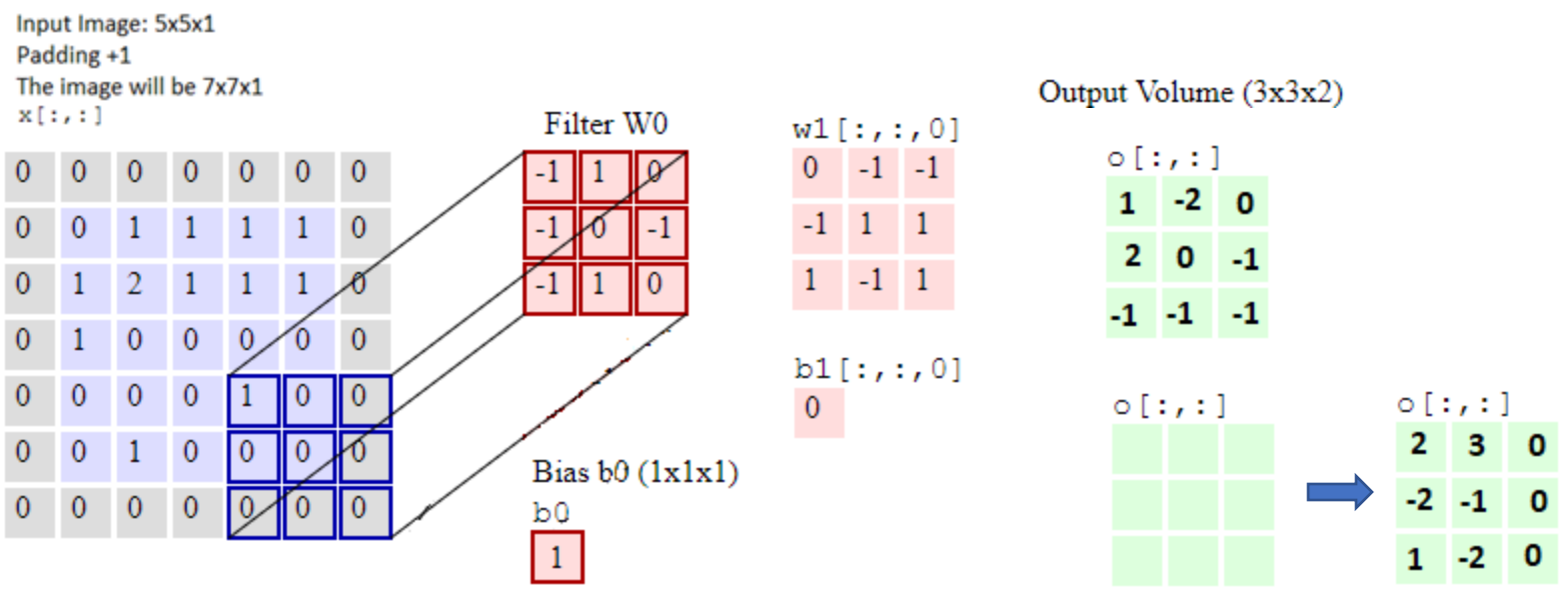


Stride 2

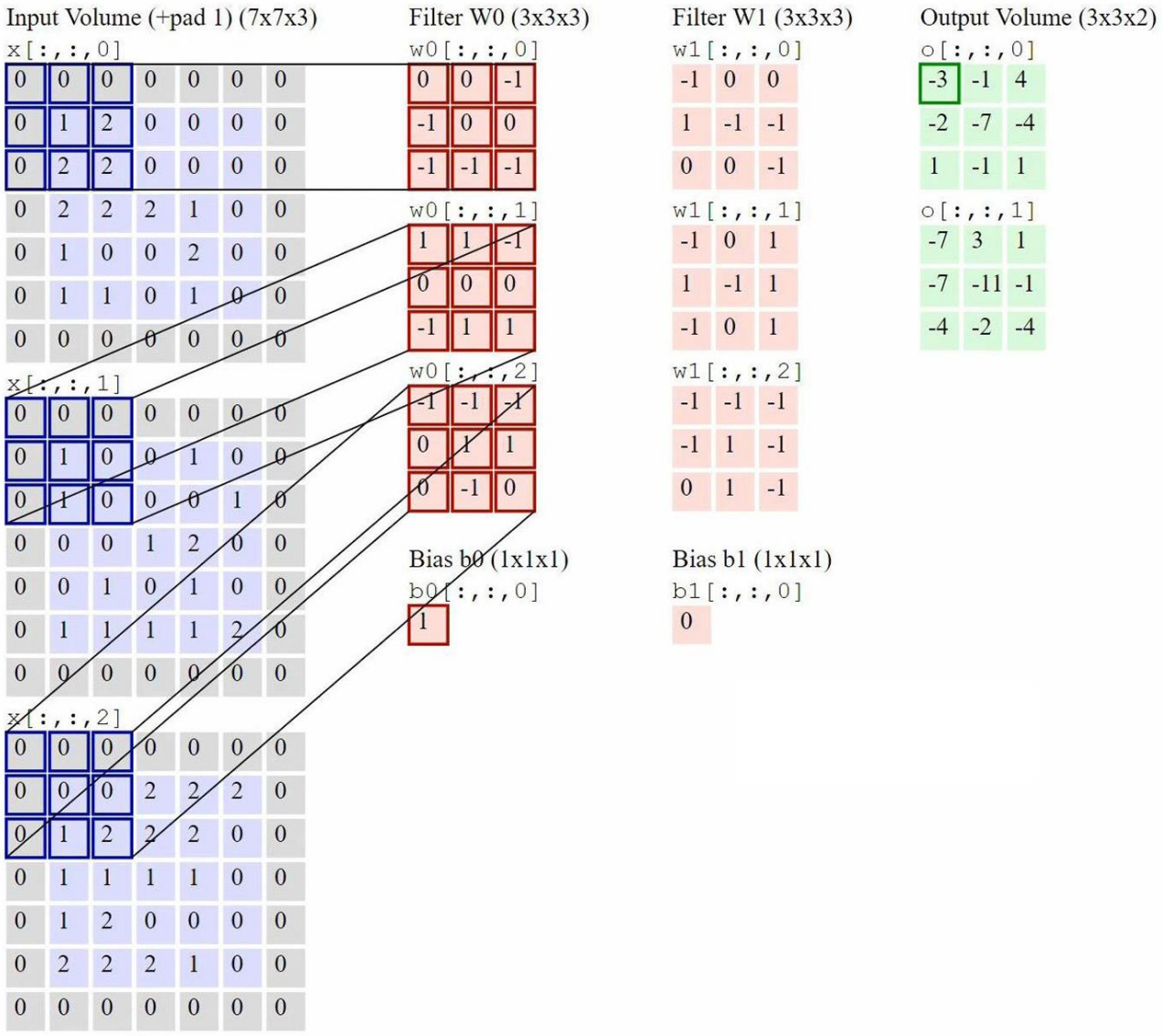
Convolution Layer: Example

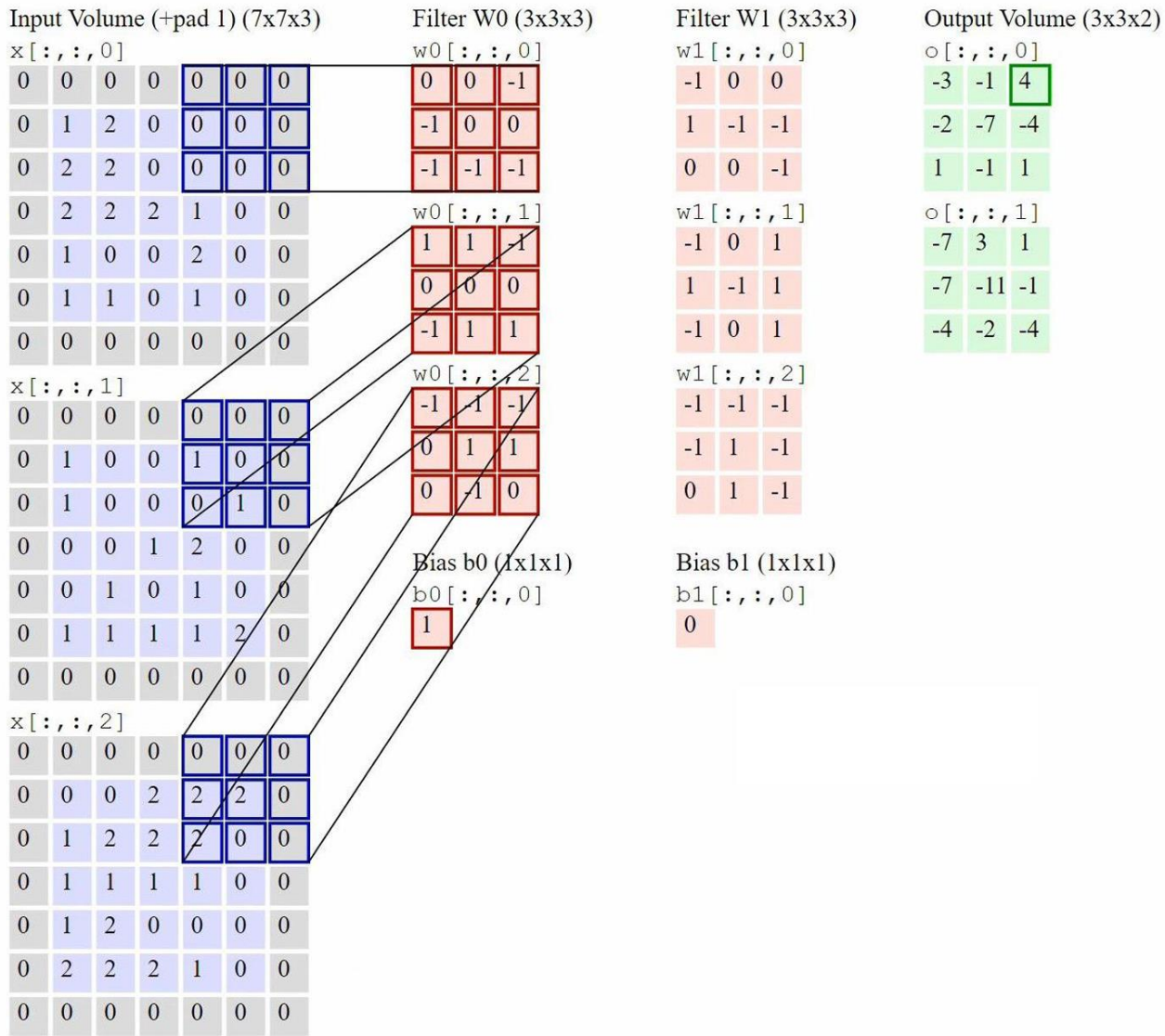


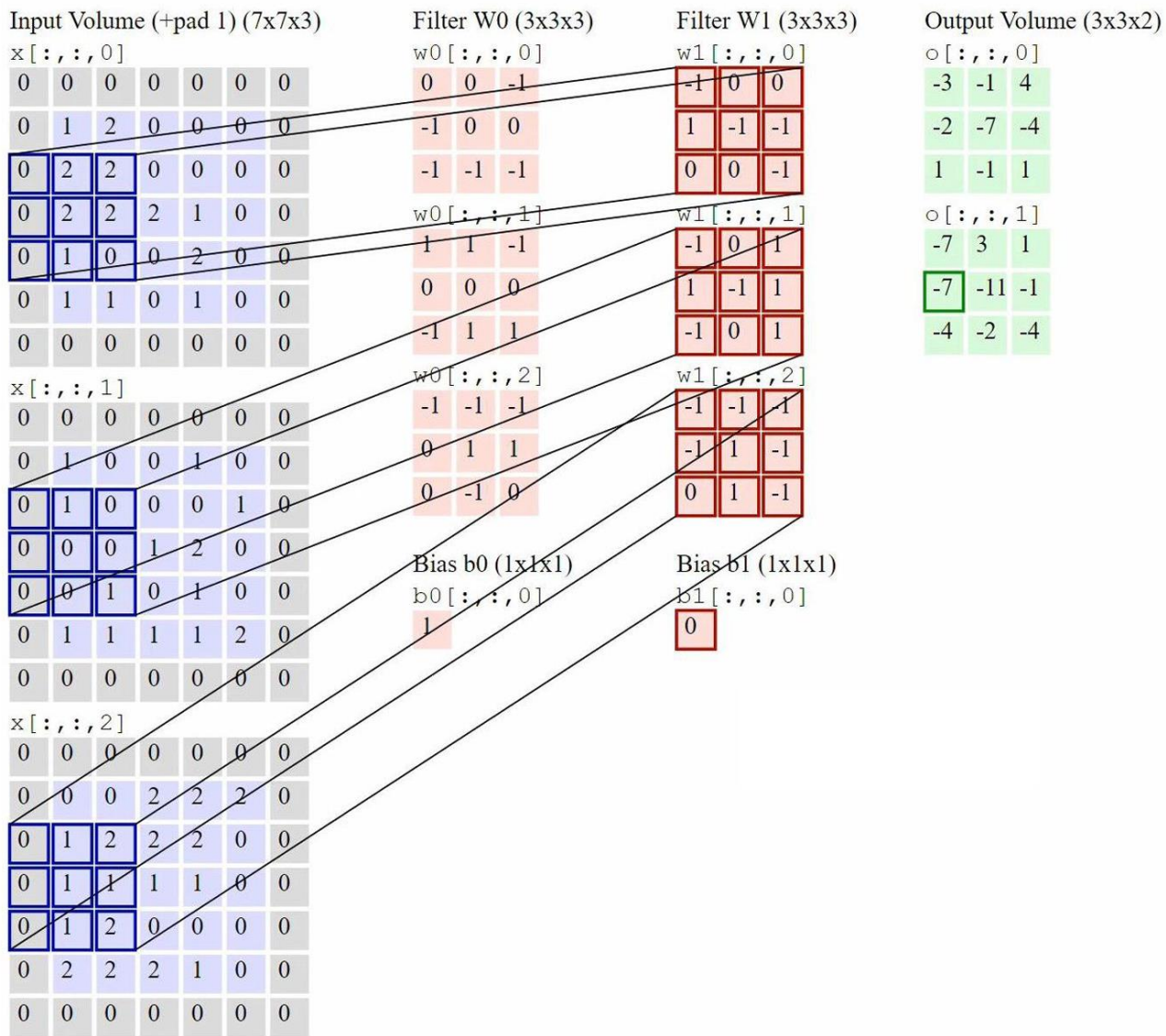
Convolution Layer: Example

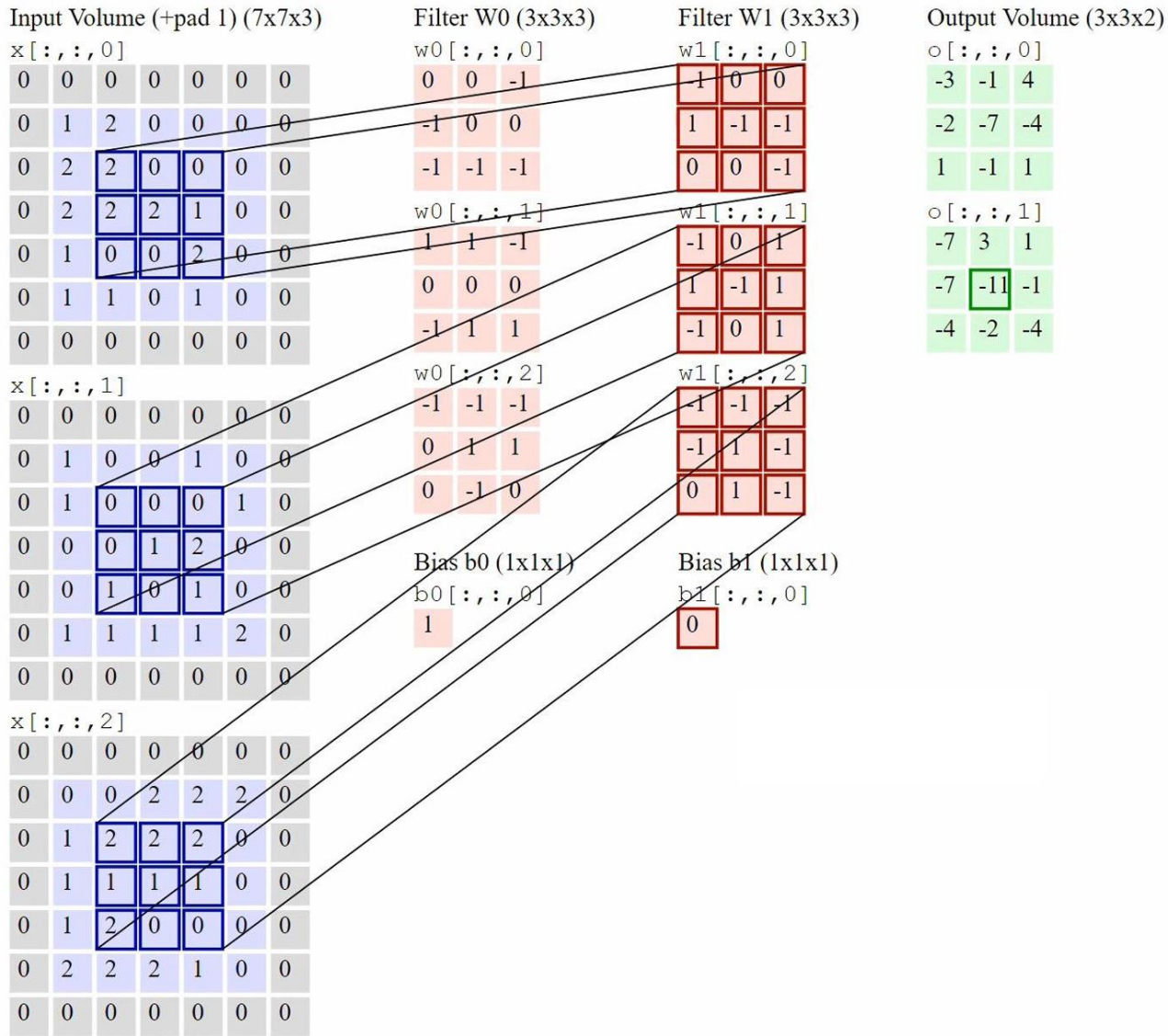


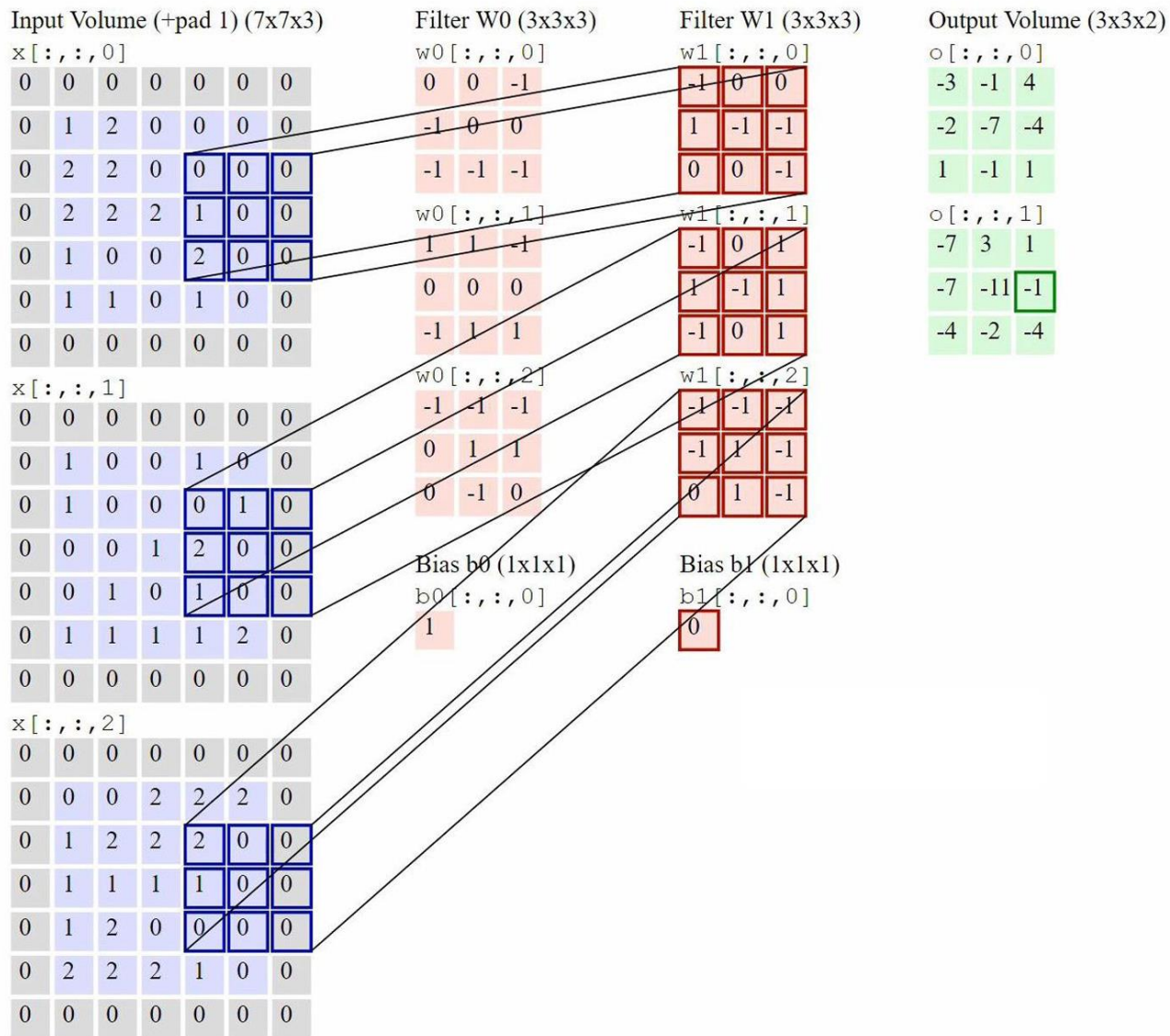
Example 2

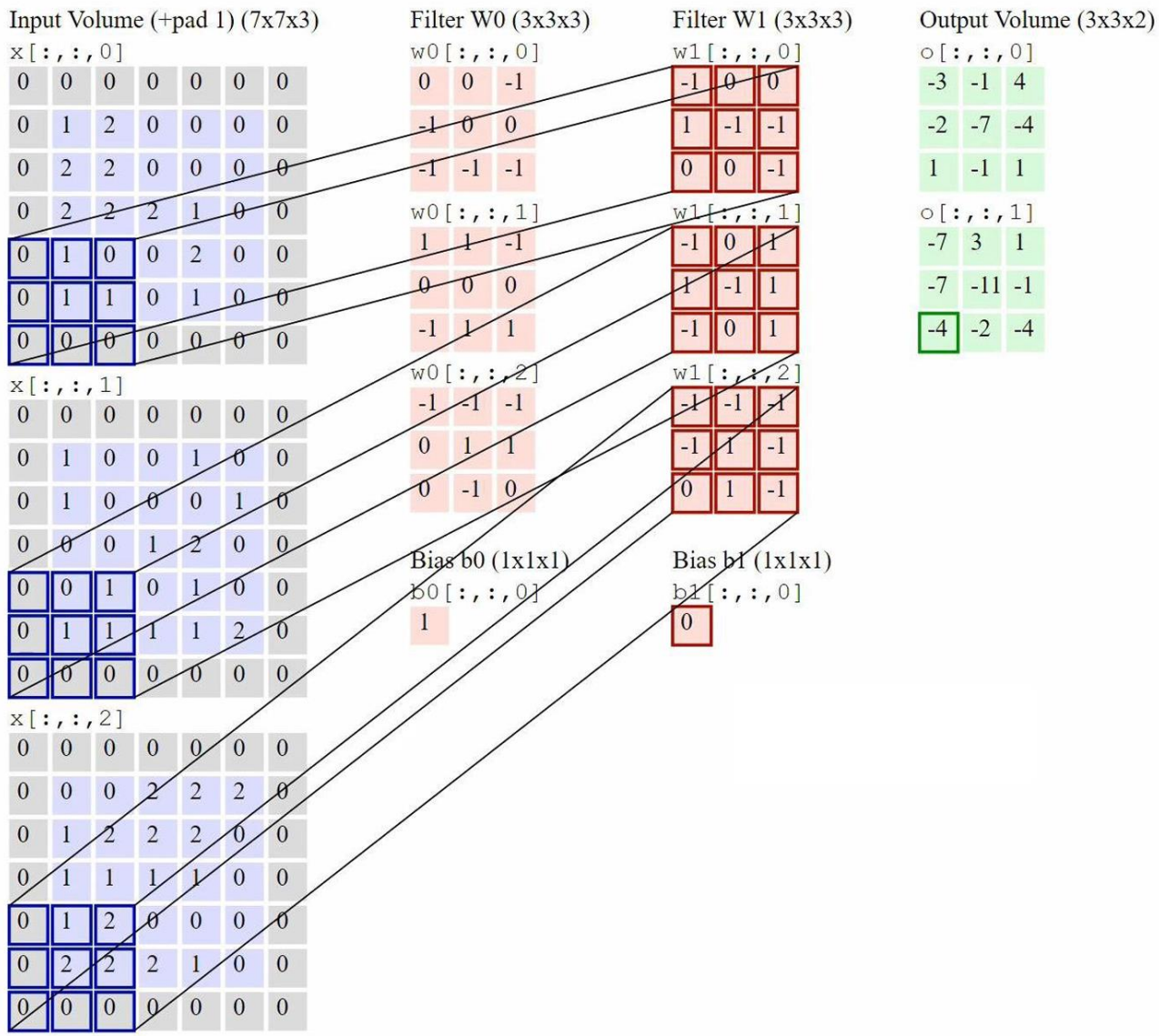

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

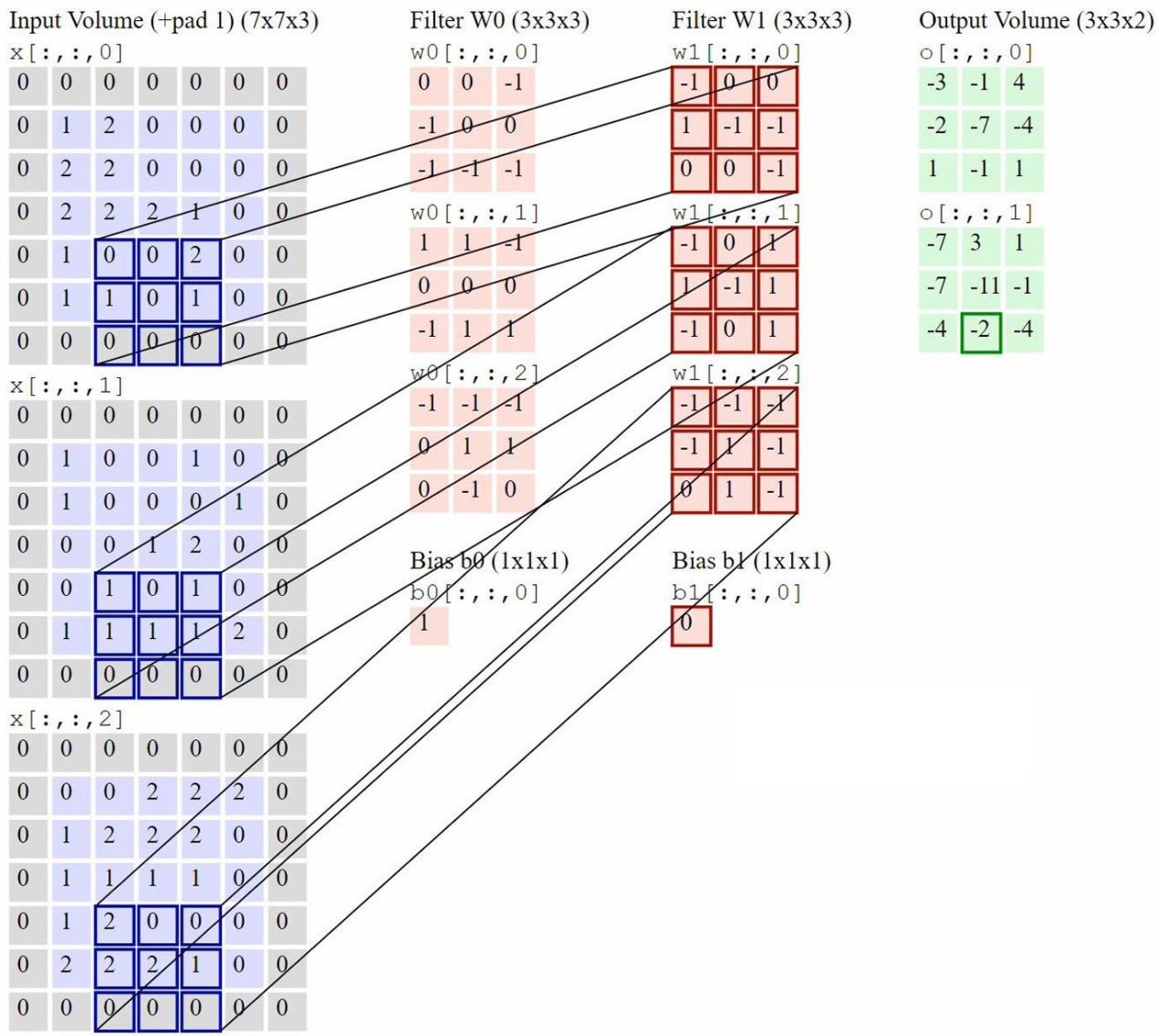

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

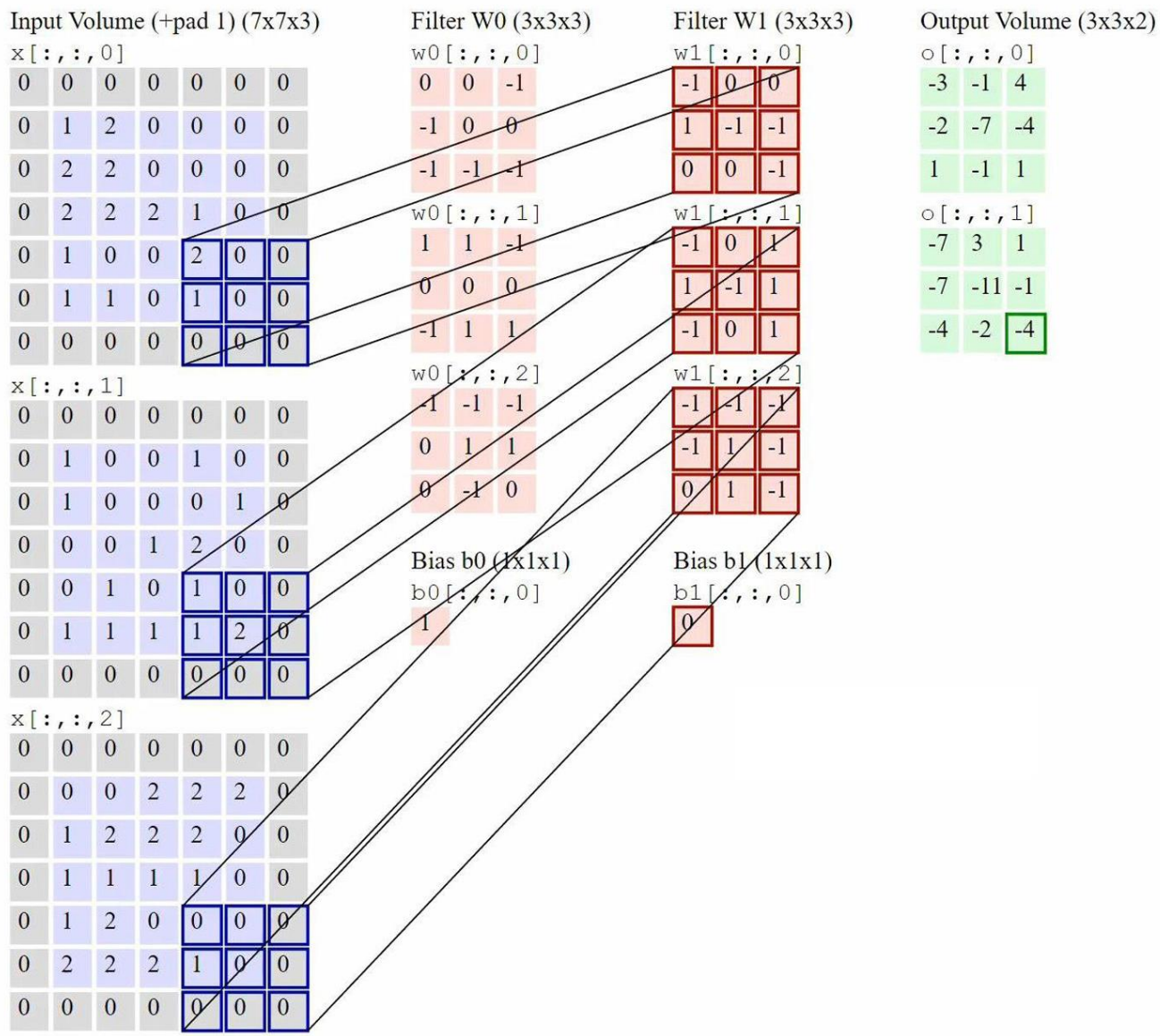

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

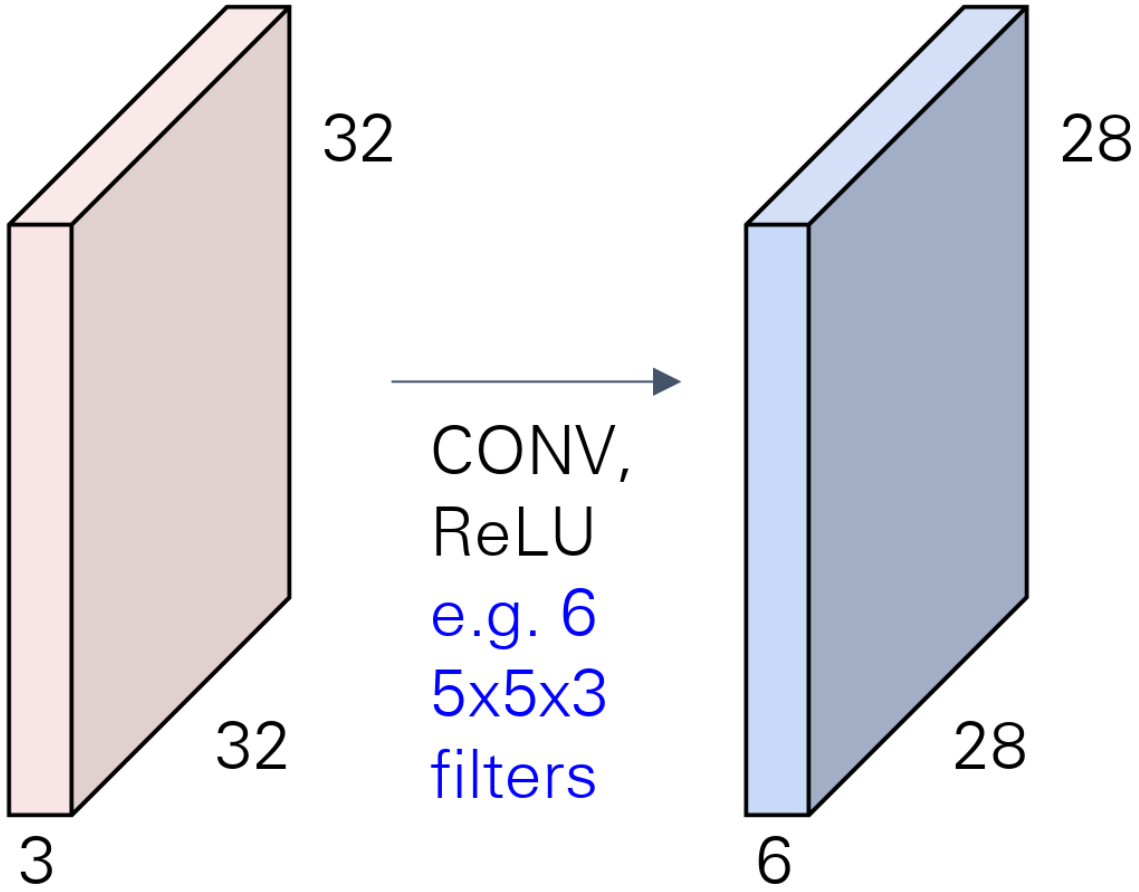

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

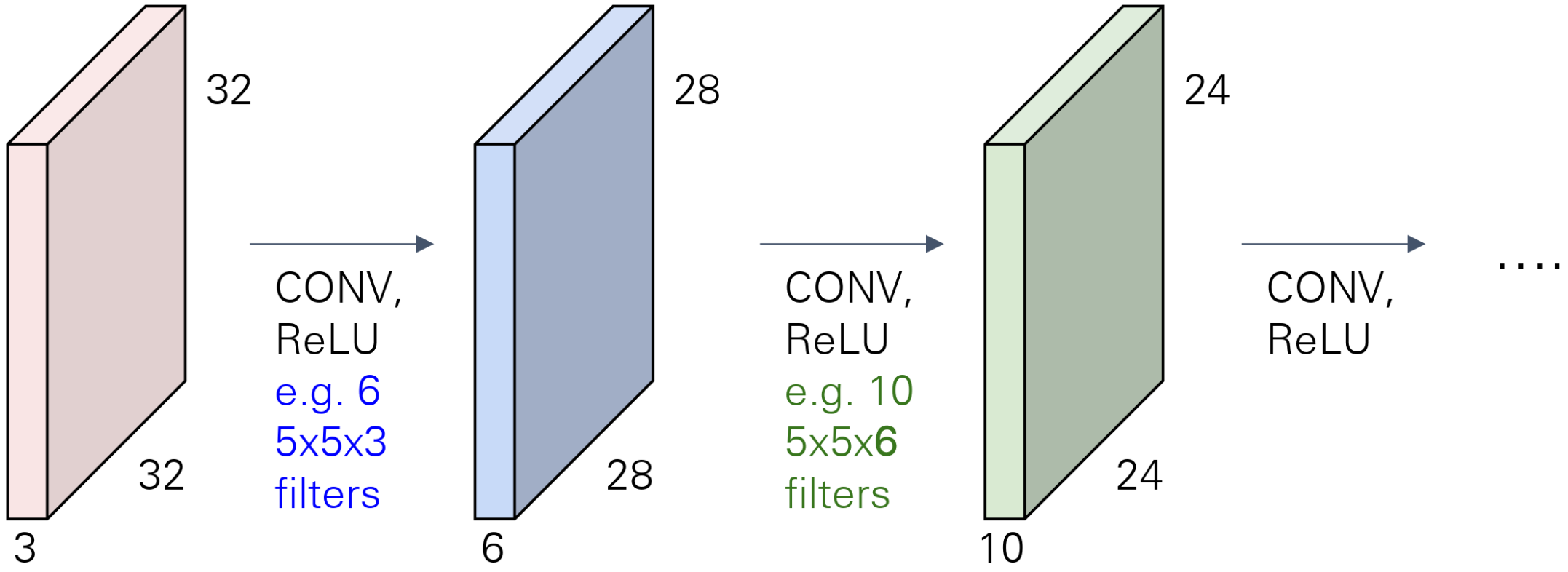

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

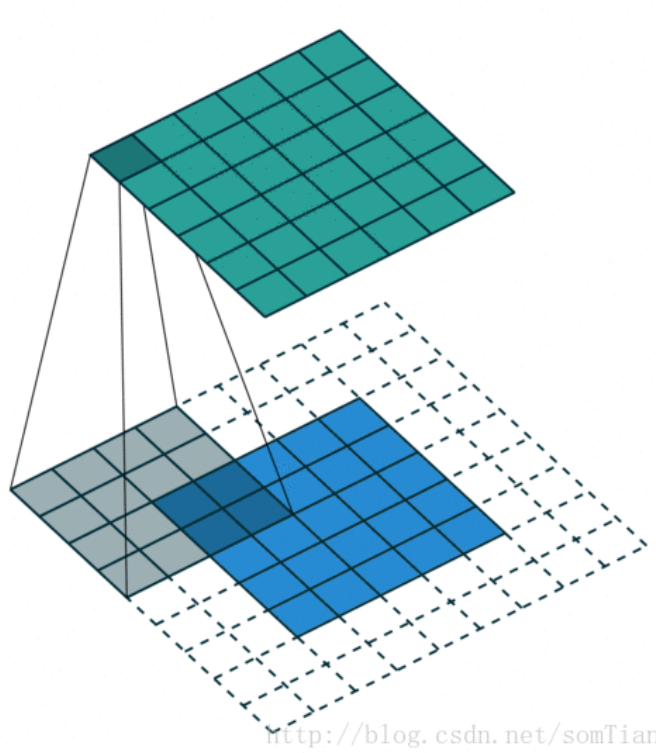
Convolutional layers



Repeat linear / non-linear operators



Convolutional kernel



Strides and Padding

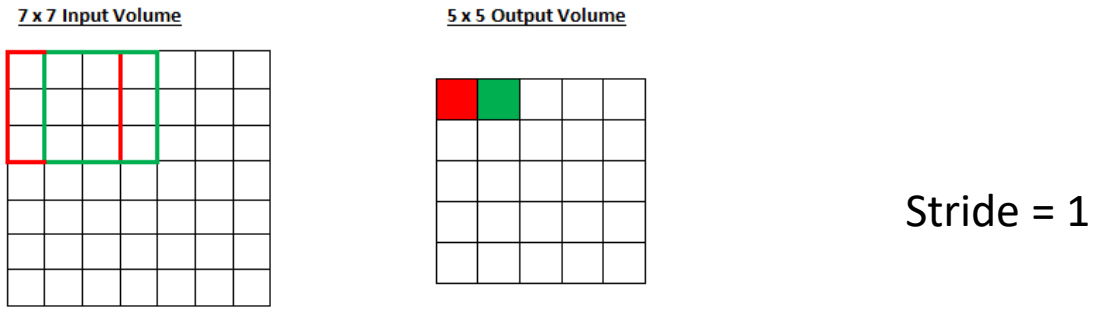
- There are 2 main parameters that we can change to modify the behaviour of each layer.

- After we choose the number of filters and filter size, we also have to choose the **stride** and the **padding**.

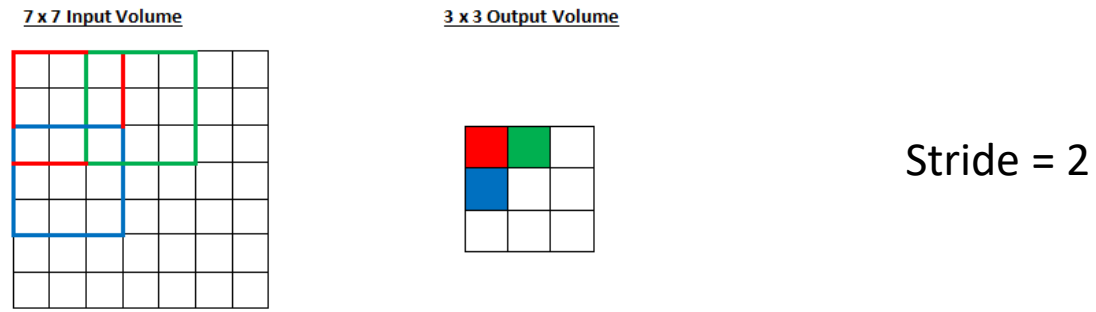
- **Stride**: controls how the filter convolves around the input volume. In the example, the filter convolves around the input volume by shifting by adjusted value (e.g., **two)** unit at a time. The amount by which the filter shifts is the stride.

Stride and Padding

- Let’s look at an example. Let’s imagine a 7 x 7 input volume, a 3 x 3 filter (Disregard the 3rd dimension for simplicity), and a stride of 1. This is the case that we’re accustomed to.



Stride = 1

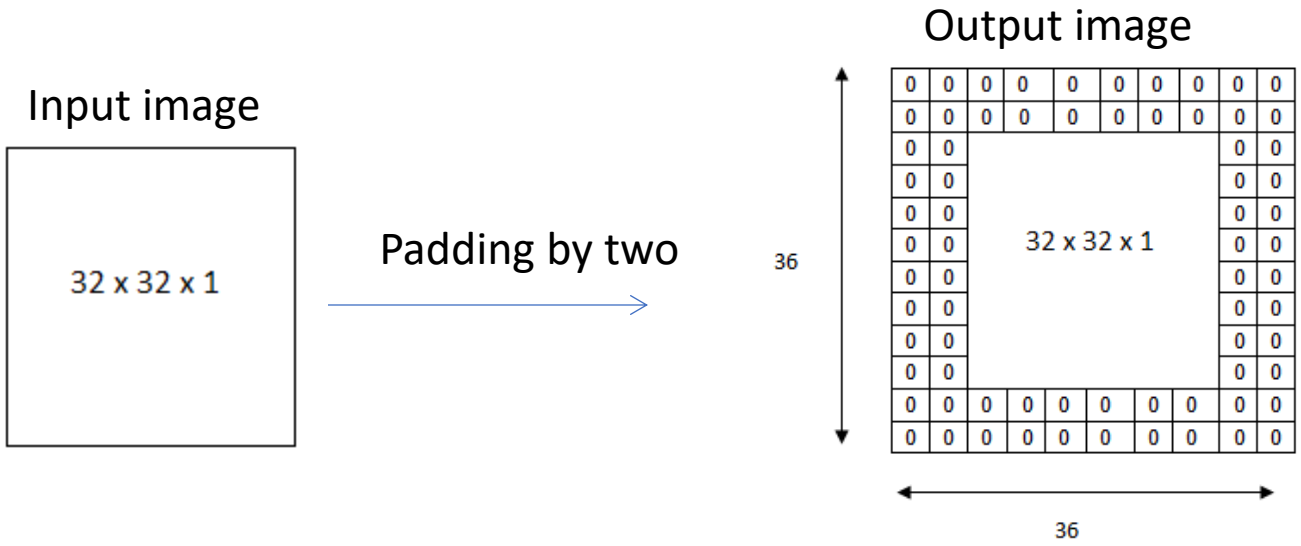


Stride = 2

```
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape))
```

Stride and Padding

- Padding: Zero padding pads the input volume with zeros around the border. If we think about a zero padding of two, then:



- Padding gives advantage to the method to keep the all features of the input image when convolution is applied to the input image.

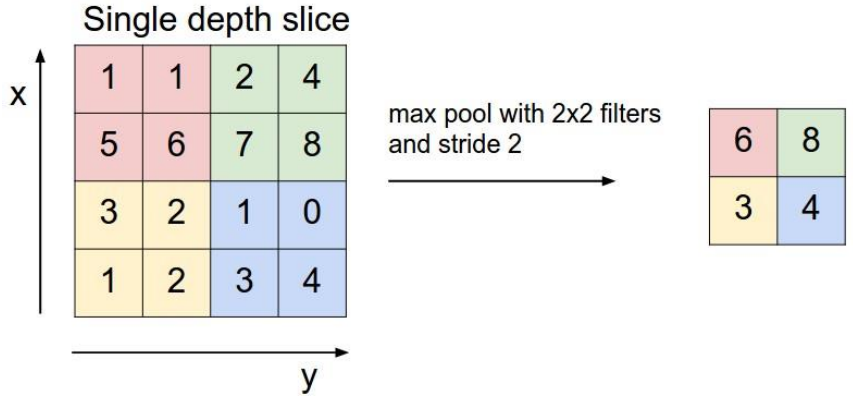
SAME = use zero-padding

VALID = no zero-padding

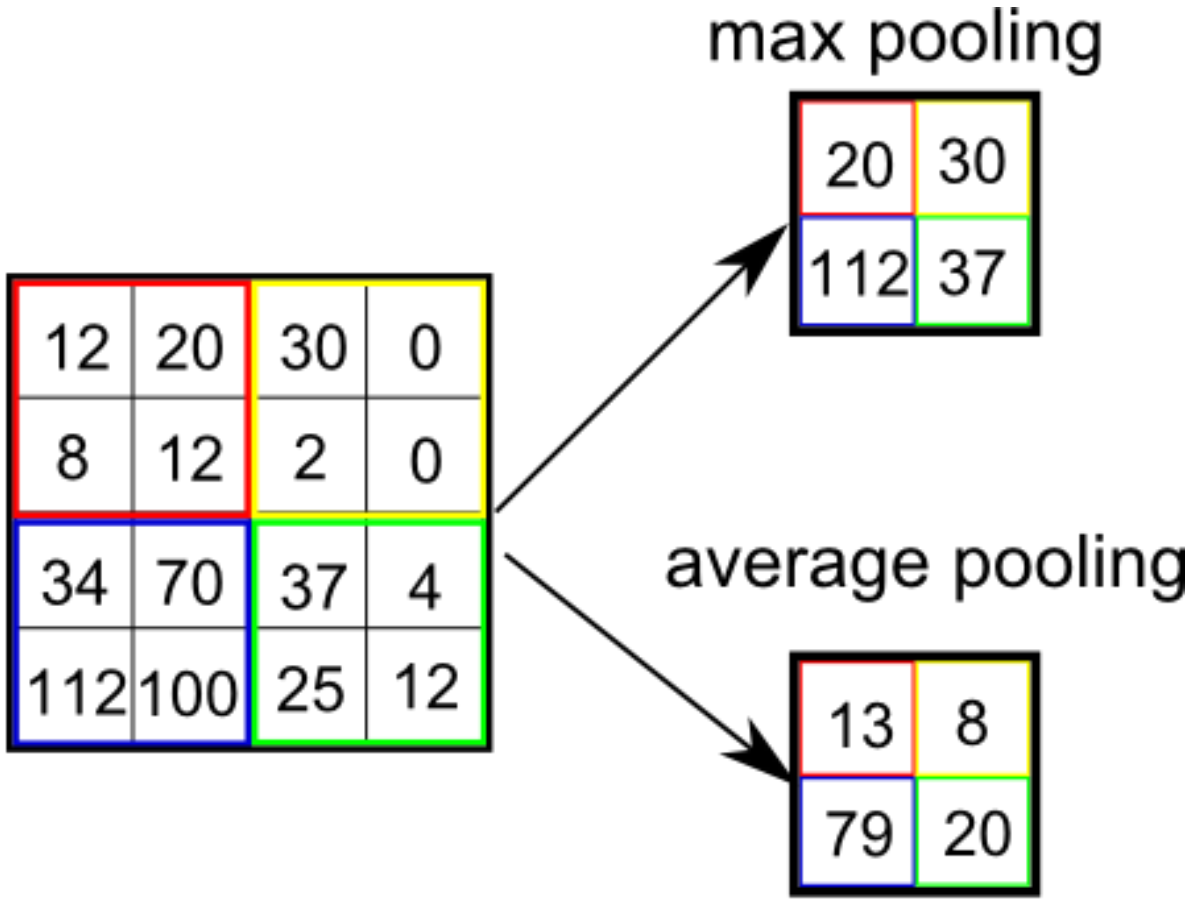
```
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='valid'))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding=same'))
```

Pooling Layer

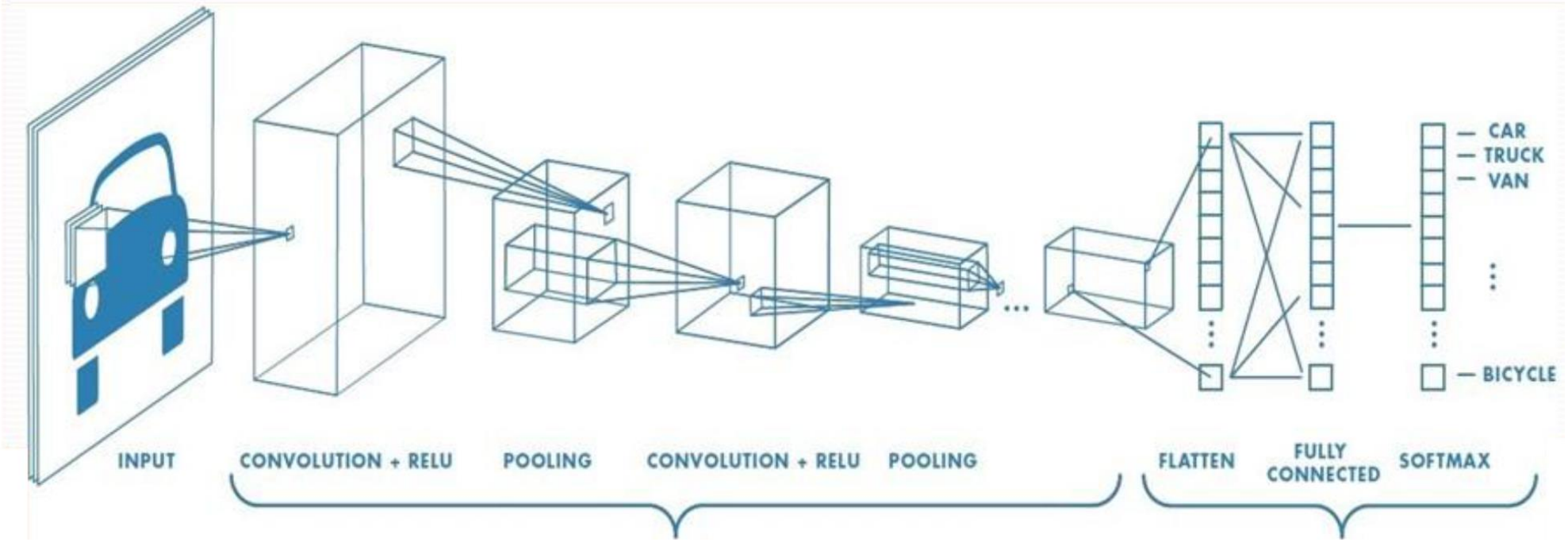
- performs a downsampling operation along the spatial dimensions (width, height).
- maxpooling being the most popular. Another one is the averaging pool layer.
- This layer drastically reduces the spatial dimension
- This layer also assumes that the most important features are the maximum values.



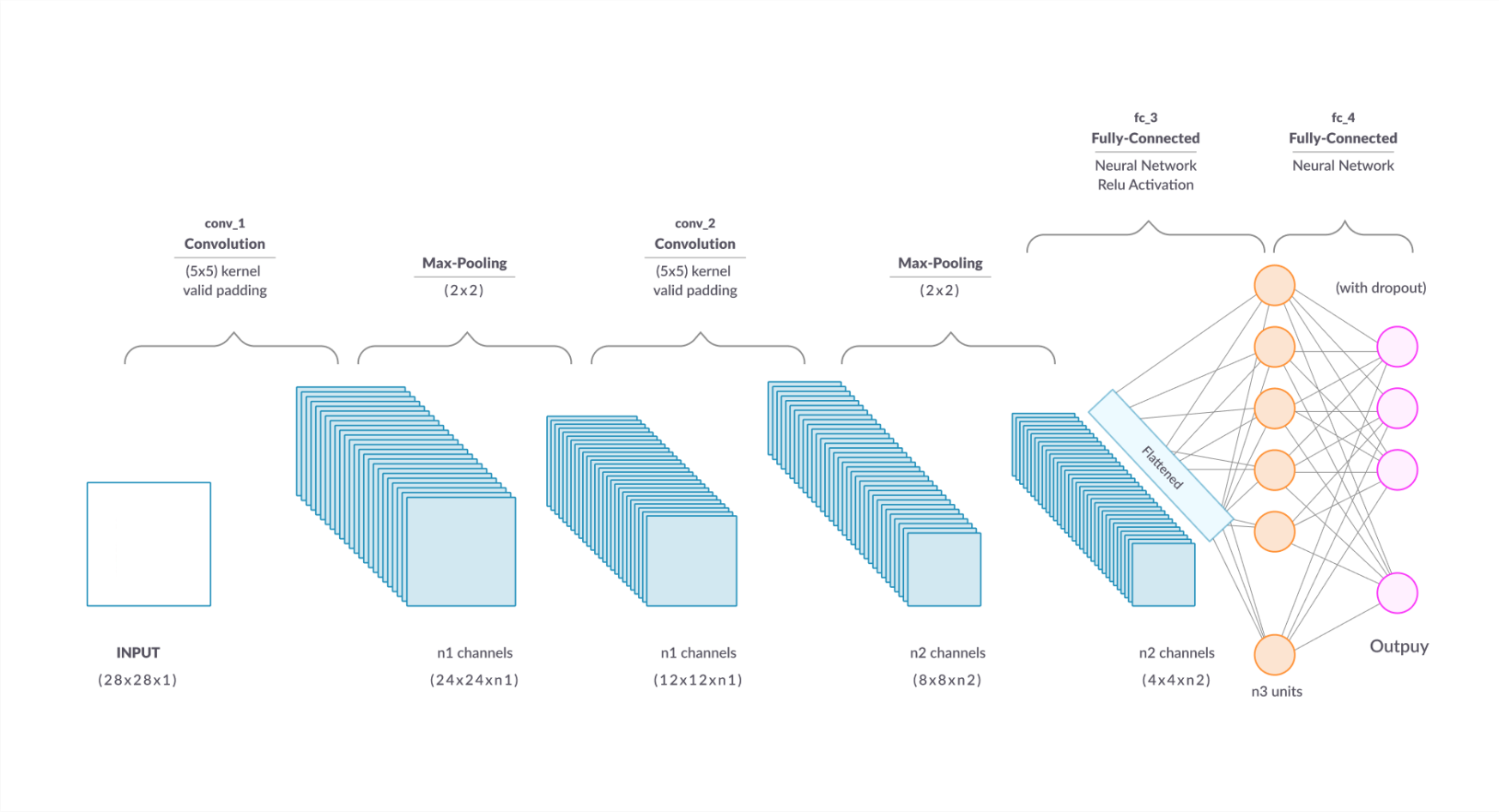
Pooling


max pooling
20 30
112 37
12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
average pooling
13 8
79 20

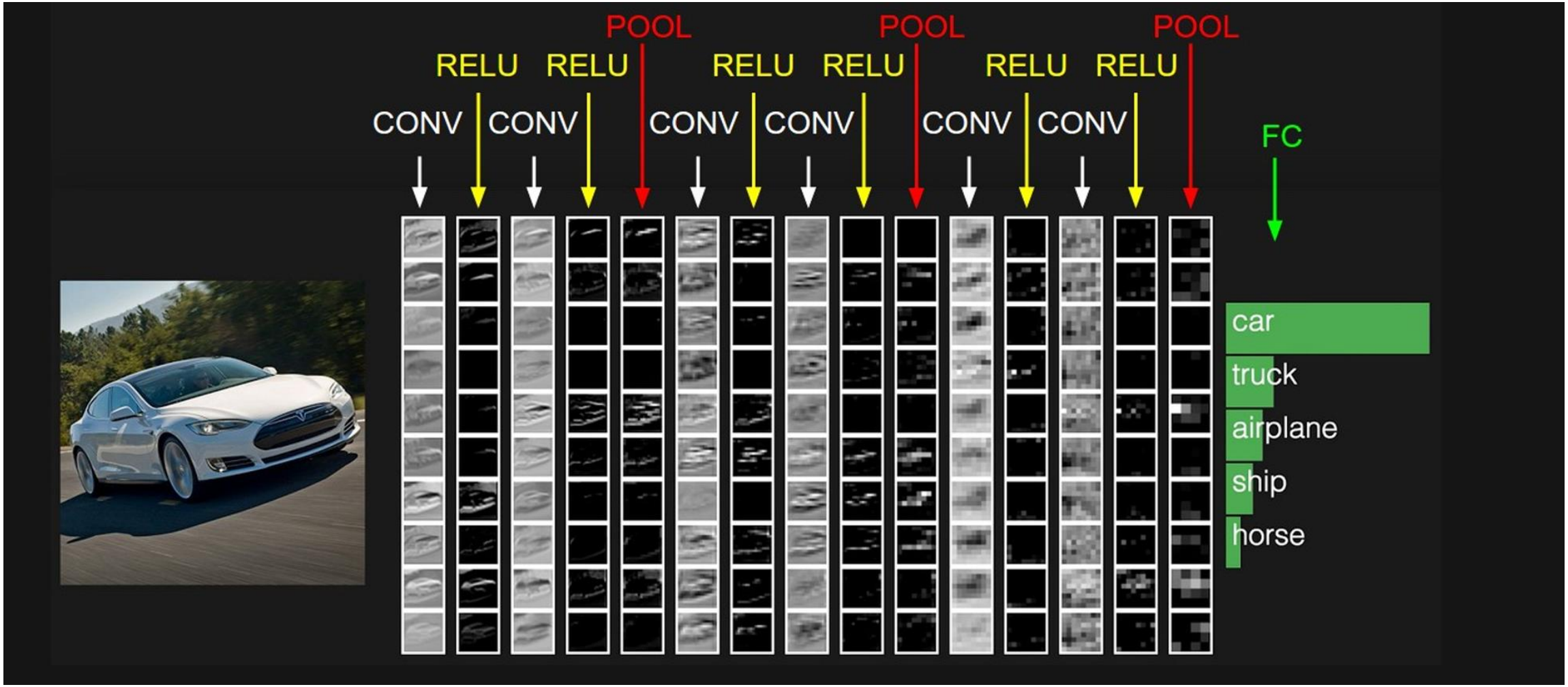
Convolutional Neural Networks (CNNs)



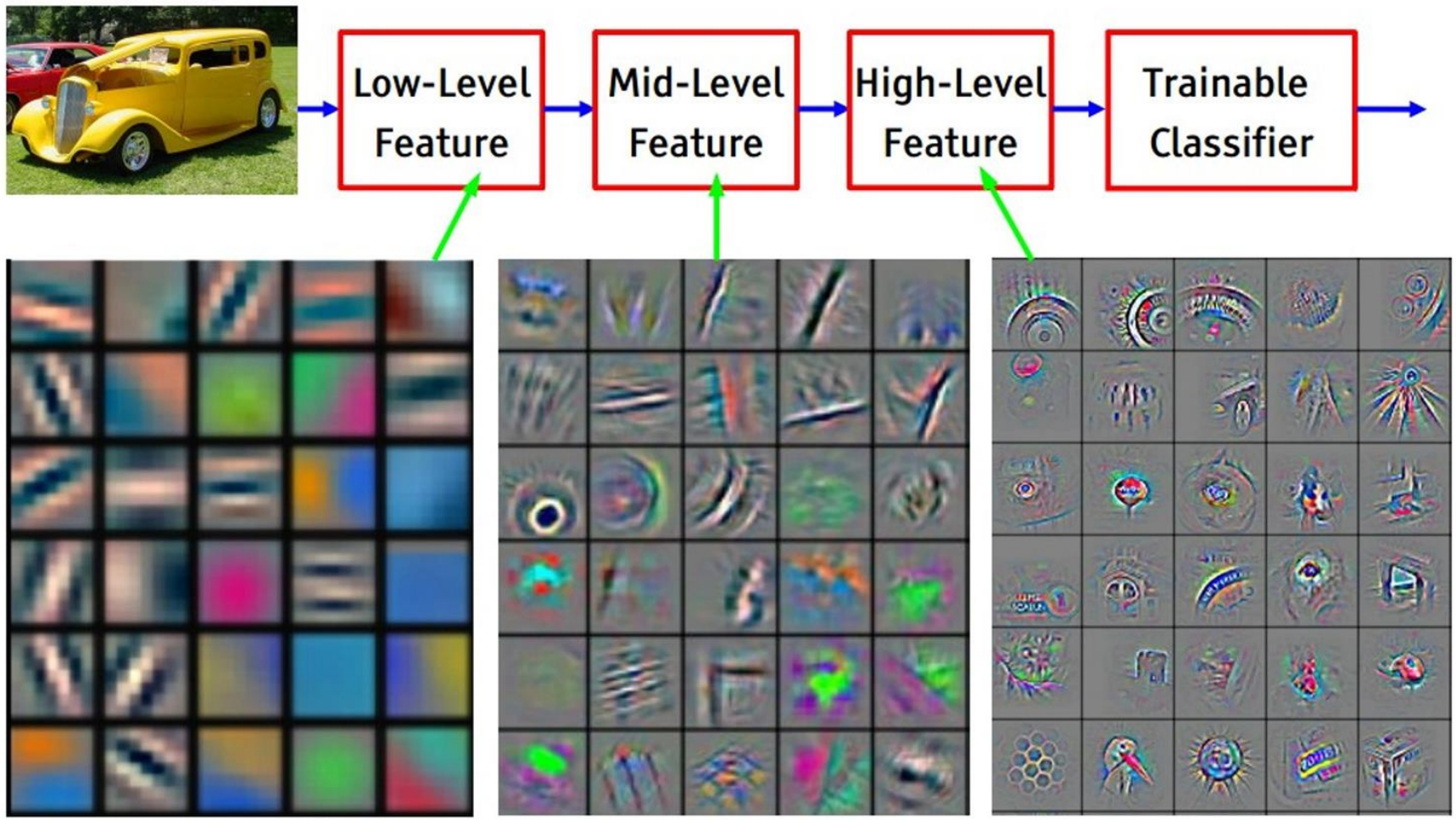
Convolutional Neural Networks (CNNs)



Feature Learning



Hierarchical layer structure allows to learn hierarchical filters (features).



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Hyperparameters of the model

- Number of layers
- Size of the layers
  - Must evaluate an array of different architectures of the right model size for the data.
  - Start from relatively few layers and parameters.
- Learning rate
- Activation functions
- Dropout
- Batch size
- No. of epochs

LeNet-5 Architecture Overview

- Input: 32×32 grayscale image (e.g., a digit image from MNIST, originally 28×28, padded to 32×32).
- Total layers: 7 layers (excluding input), each with learnable parameters.

Cont.

- Assume input image: 32x32 pixels (MNIST digits are 28x28, but zero-padded to 32x32 in LeNet)

Layer	Type	Output Size	Details
Input	—	32x32x1	Grayscale image
C1	Convolution	28x28x6	6 filters (5x5), stride 1, no padding
S2	Average Pooling	14x14x6	Subsampling (2x2), stride 2
C3	Convolution	10x10x16	16 filters (5x5), some filters connected to subsets of input maps
S4	Average Pooling	5x5x16	Again, 2x2 pooling
C5	Convolution	1x1x120	120 filters (5x5), fully connected to previous layer
F6	Fully Connected	84	Fully connected layer (like traditional MLP)
Output	Fully Connected	10	Softmax output for 10 digit classes (0–9)

Demonstration

Quiz

1. What is the main purpose of a pooling layer in a CNN?

A) To reduce overfitting by randomly dropping neurons
B) To increase the number of parameters in the model
C) To reduce the spatial dimensions and retain important features
D) To convert images into grayscale

C

2. In a 2D convolution operation with a filter of size 3x3, stride of 1, and padding of 1, what happens to the spatial dimensions of the output feature map compared to the input?

A) They become smaller
B) They remain the same
C) They double
D) They become zero

B

Q3. What is the main purpose of the convolution operation in a CNN layer?

A. To flatten the image
B. To reduce the number of input features
C. To extract spatial features from the input image
D. To increase the dimensionality of the image

C

Q4. Which of the following is a characteristic of max pooling?

A. It increases the number of parameters in the model
B. It selects the minimum value in each region
C. It reduces spatial dimensions while retaining important features
D. It increases the resolution of the feature map

C

Discussion

How does data augmentation improve the performance of a CNN, and what are some common techniques used in data augmentation?

Data augmentation helps prevent overfitting and improves generalization by artificially increasing the size and diversity of the training dataset. This is especially useful when you have limited training data. Instead of collecting more images, you make smarter use of the ones you already have.

Common Data Augmentation Techniques:

Rotation: Slightly rotate images by a few degrees (e.g., ±15°).

Flipping: Horizontally (and sometimes vertically) flip the image.

Cropping: Randomly crop a region from the image.

Zooming: Slightly zoom in or out of the image.

Translation: Shift the image along the X or Y axis.

Brightness Adjustment: Modify brightness to simulate different lighting.

Thank you for listening