Week5 CNN

Tutor:
Email:
Tutorial:
Code: https://github.com/Jinxu-Lin/COMP5329

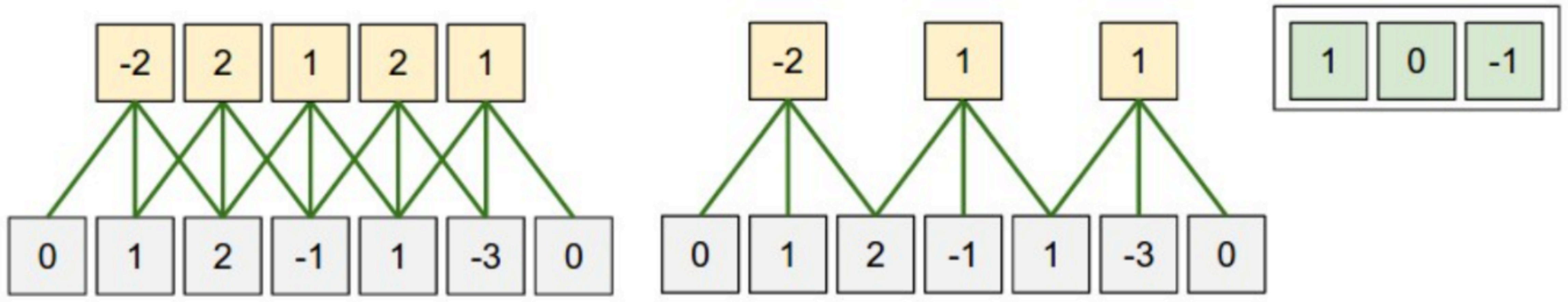
Convolutional Neural Network

Convolution

- Convolution operation
- Popular filters/kernels in Computer Vision
- Convolutional layer
- Stride and Padding
- Pooling layer

Convolution Operation

- $(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$



- Refer to this link to see example: (https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1)

Convolution Operation

- Before Neural Network why does this operation matter?
    - Traditional Computer Vision (CV) goes around the act of designing the filters so that it can extract desirable features.
    - 1st question: "What features do I want to extract?" e.g Key points, edges, blurr.etc.
    - 2nd question: "How do I extract them?" e.g Kalman filter, Gaussian filter,.etc.

Convolution Operation

- Before Neural Network why does this operation matter?
  - Traditional Computer Vision (CV) goes around the act of designing the filters so that it can extract desirable features.
  - 1st question: "What features do I want to extract?" e.g Key points, edges, blurr.etc.
  - 2nd question: "How do I extract them?" e.g Kalman filter, Gaussian filter,.etc.

Popular filters/kernels

- Edge Detection Filters
  - Horizontal

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

  - Vertical

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

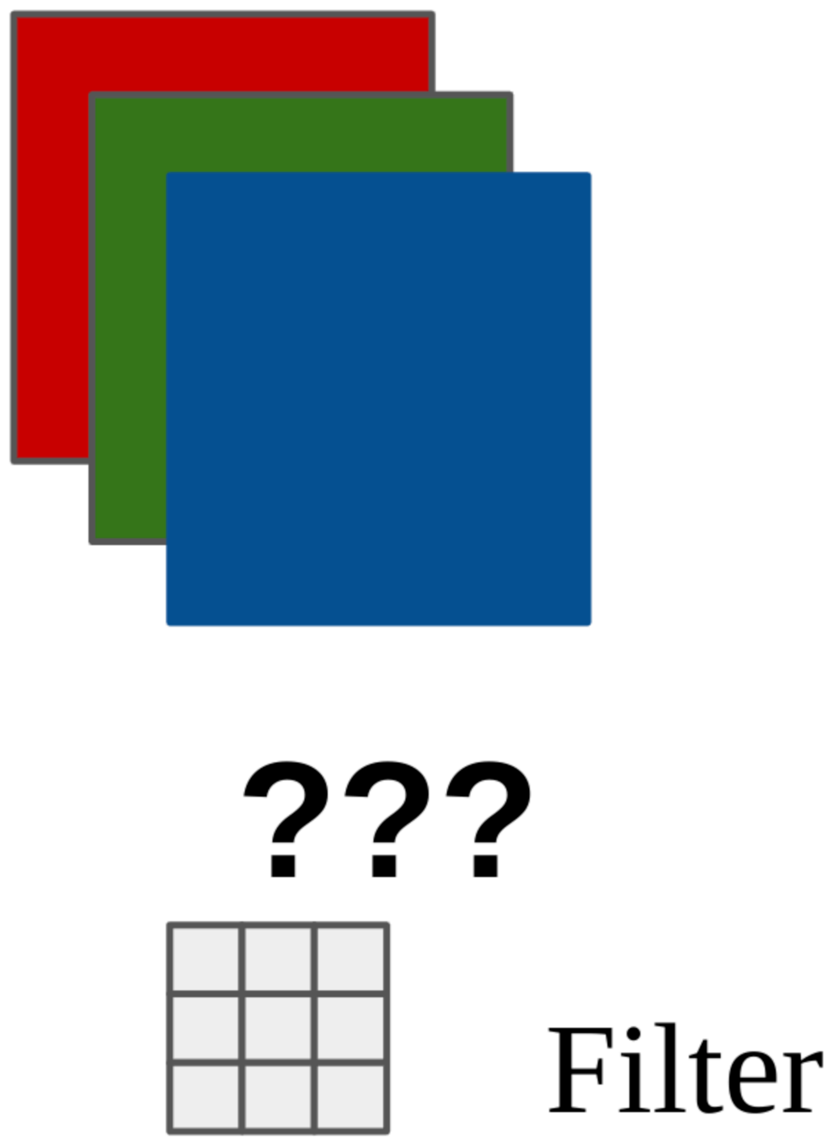
  - …

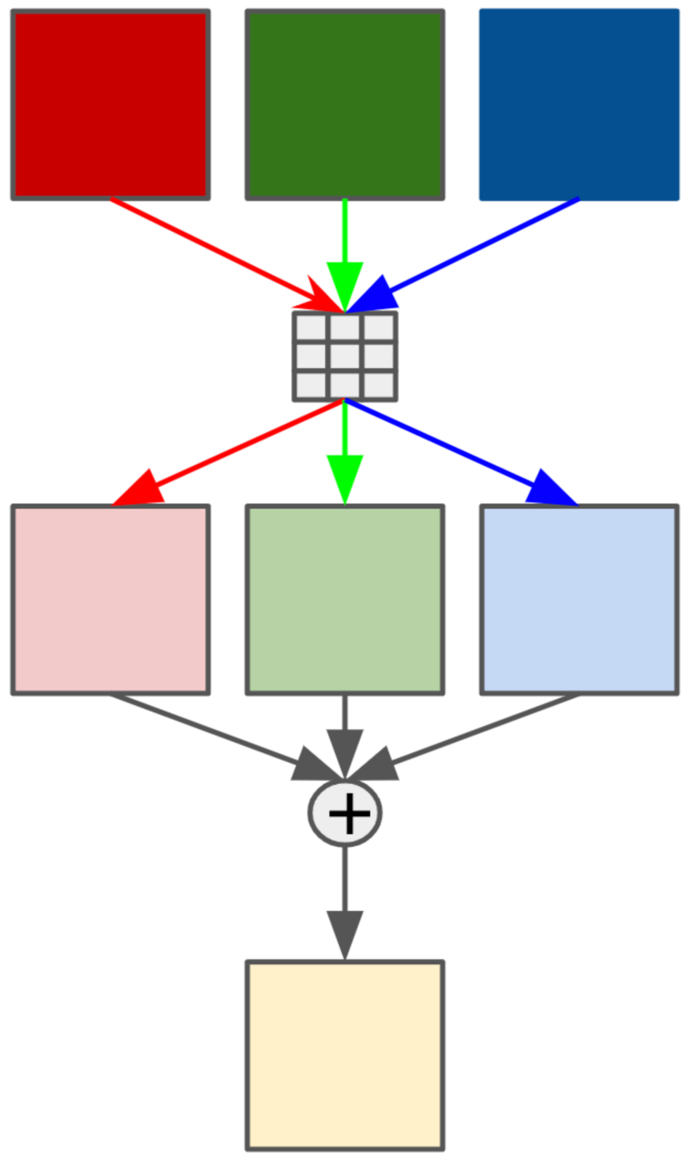
Popular filters/kernels

- What is the problem?
  - Manually design the convolution filters is an exhaustive task
  - Not effective enough as some features are not easily to understand to human eyes.
  - How to arrange the orders of filters to achieve the best results
- We need something stronger and less dependent on human!!!!
  - Actually, it is just a set of filters whose parameters are optimized via training of neural network.
  - Make the parameters of convolution as the parameters of neural networks

Multi-channels with 1 filters

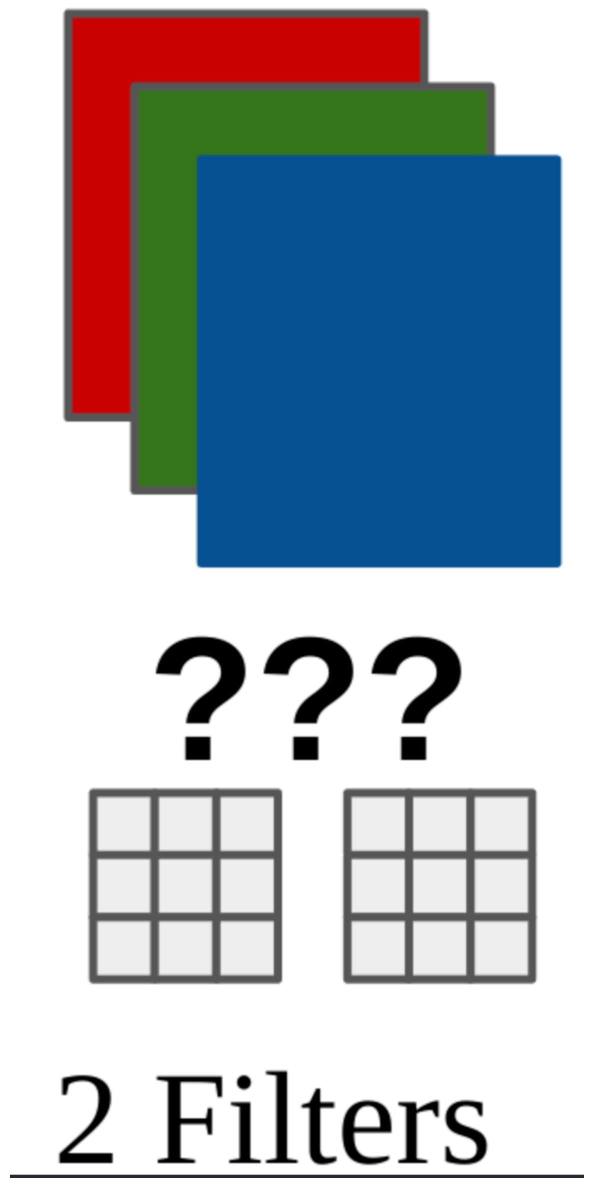
- Previously, we know how to apply the filter on gray image. How about color image (multi-channels)?
- Multi-channels input: conduct convolution separately
- 1 filters: 1 channels of output
- Result of each input channels: Add

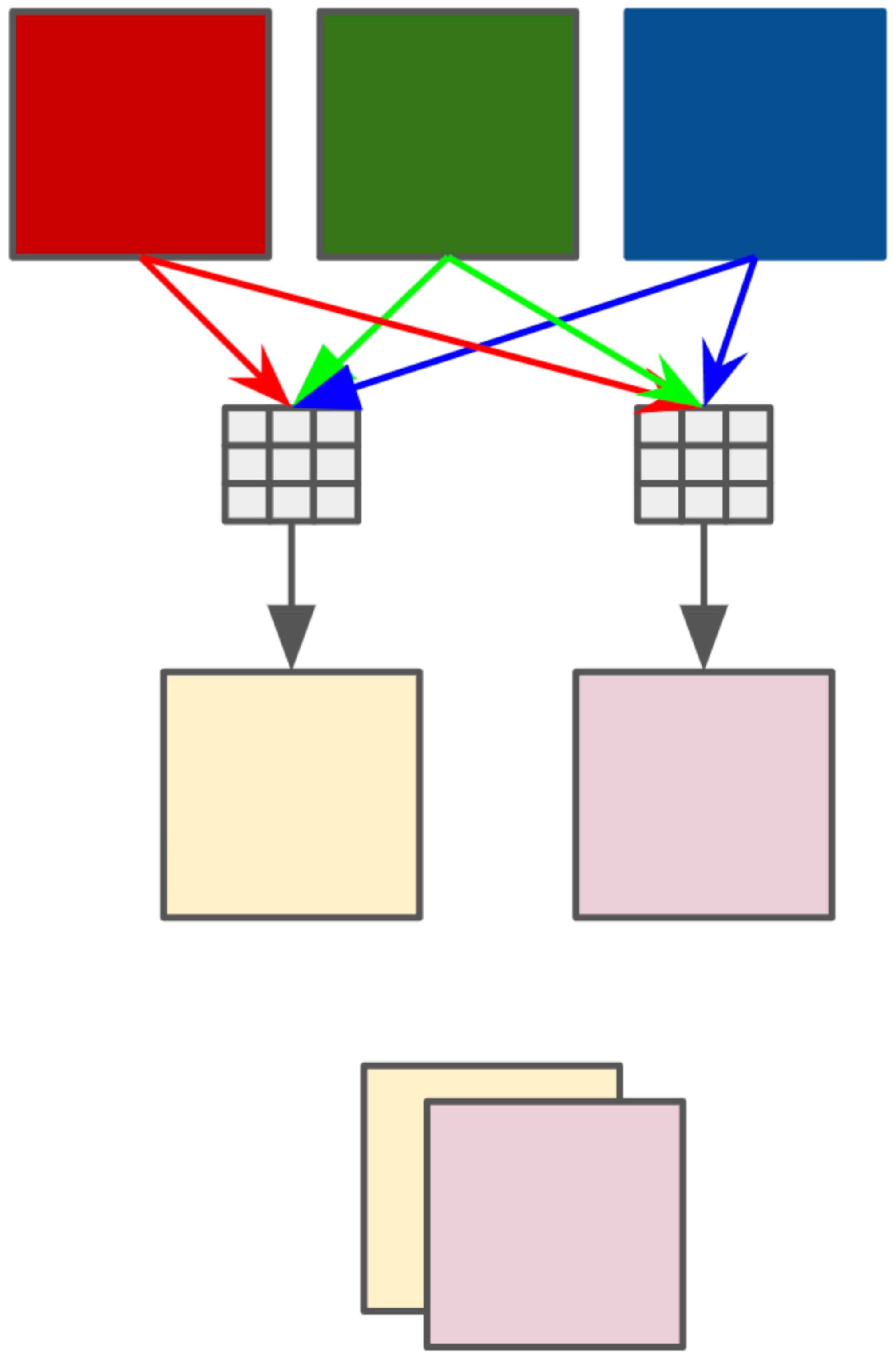




Multi-channels with multi-filters

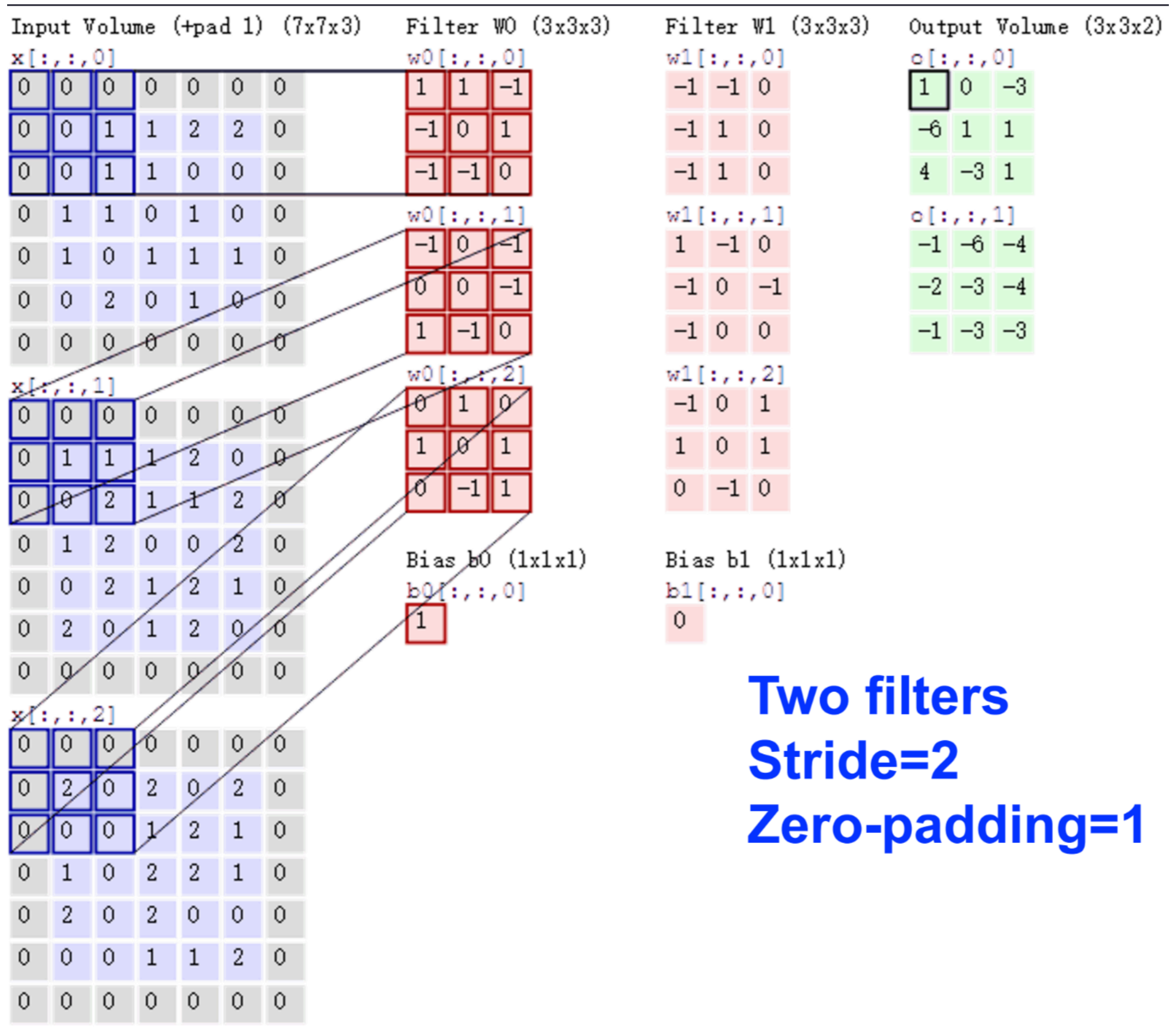
- Multi-filters: Multi channels of output





Multi-channels with multi-filters

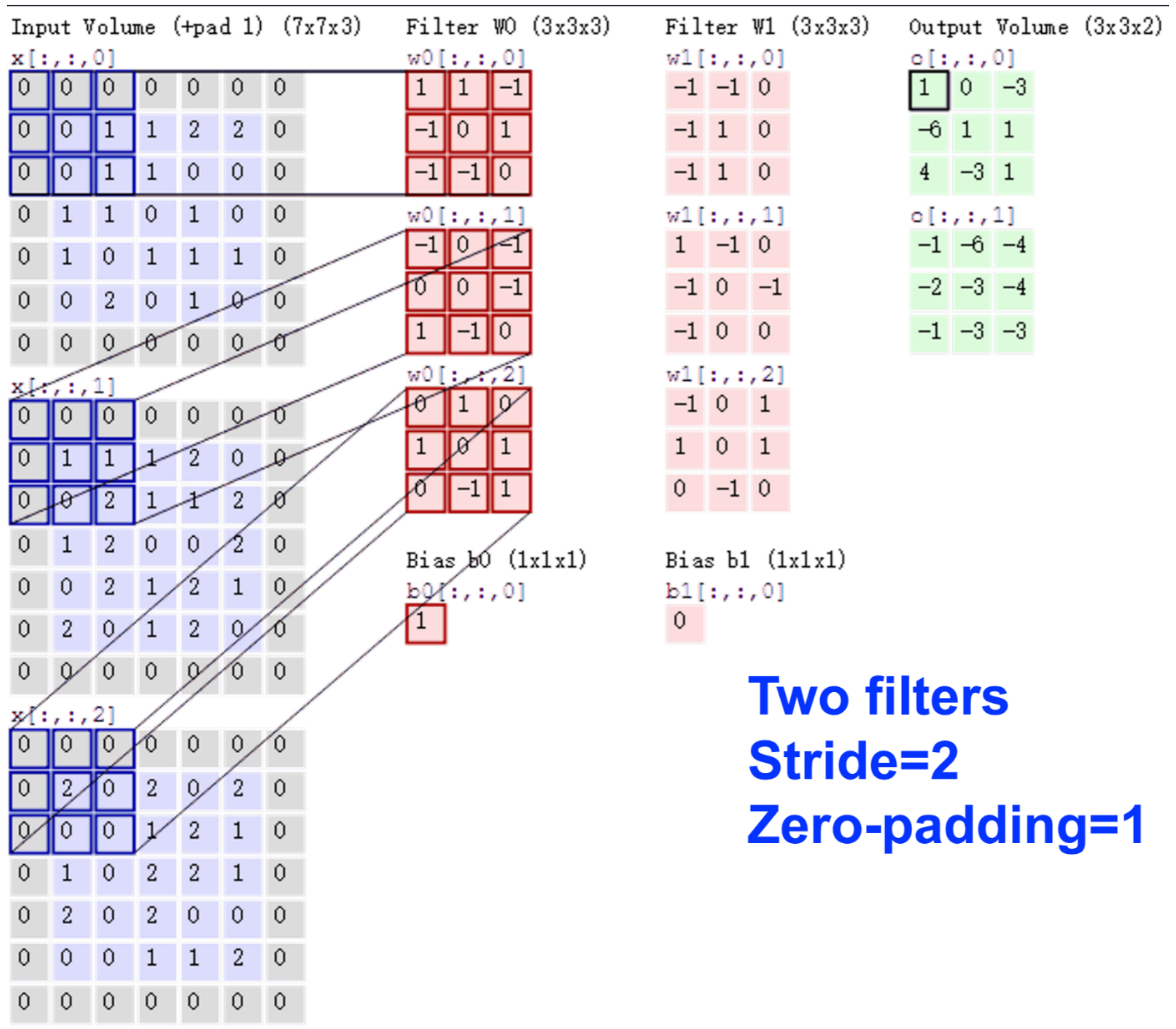
- Previous examples are just for illustration for dealing with multiple input channels and multiple filters. In CNN, we will employ the multi-channel filters like image below:



Two filters
Stride=2
Zero-padding=1

Multi-channels with multi-filters

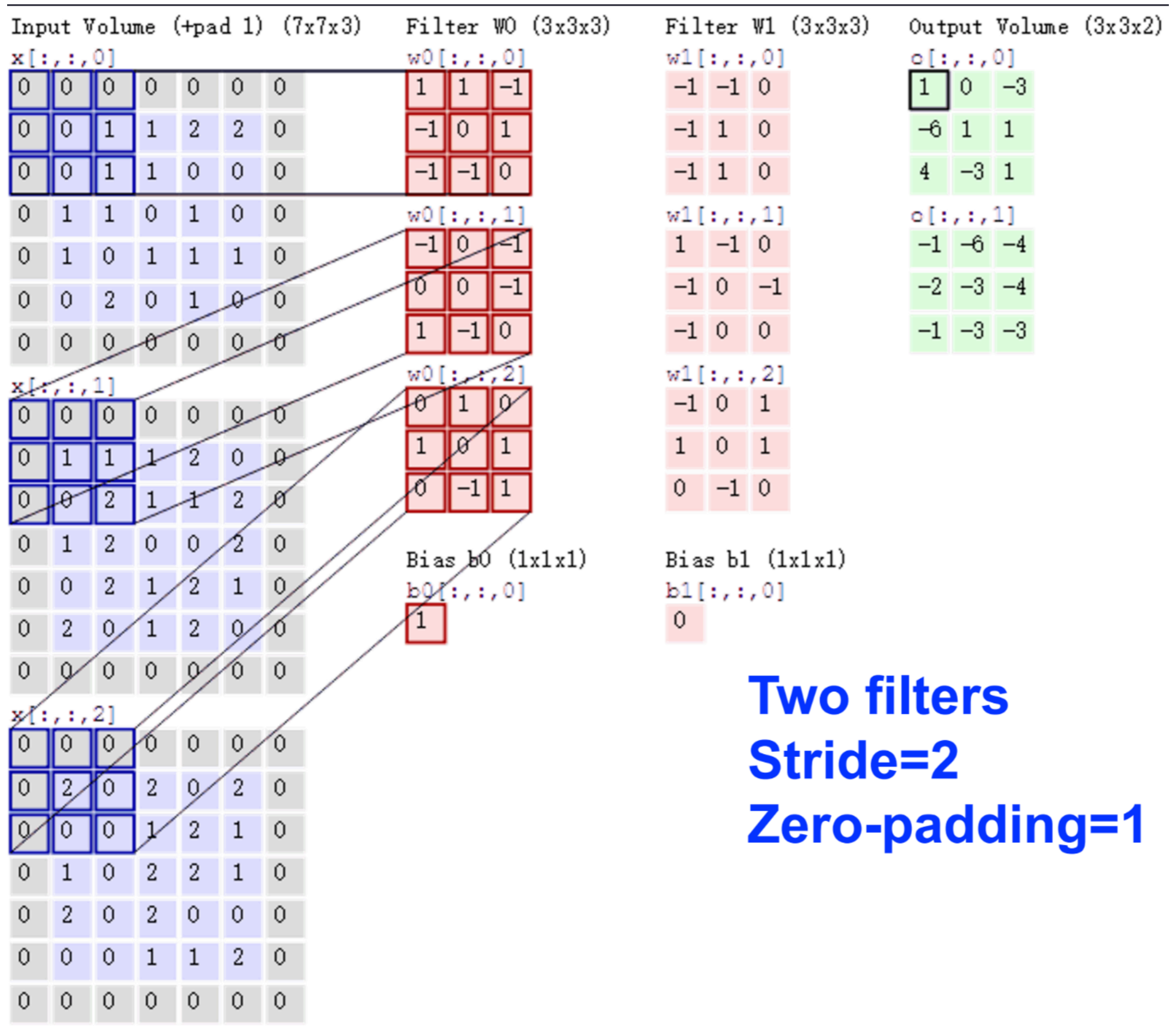
- The number of channels of the output = the number of filters
- The number of channels of the input = the number of channels in each filter



Two filters
Stride=2
Zero-padding=1

Multi-channels with multi-filters

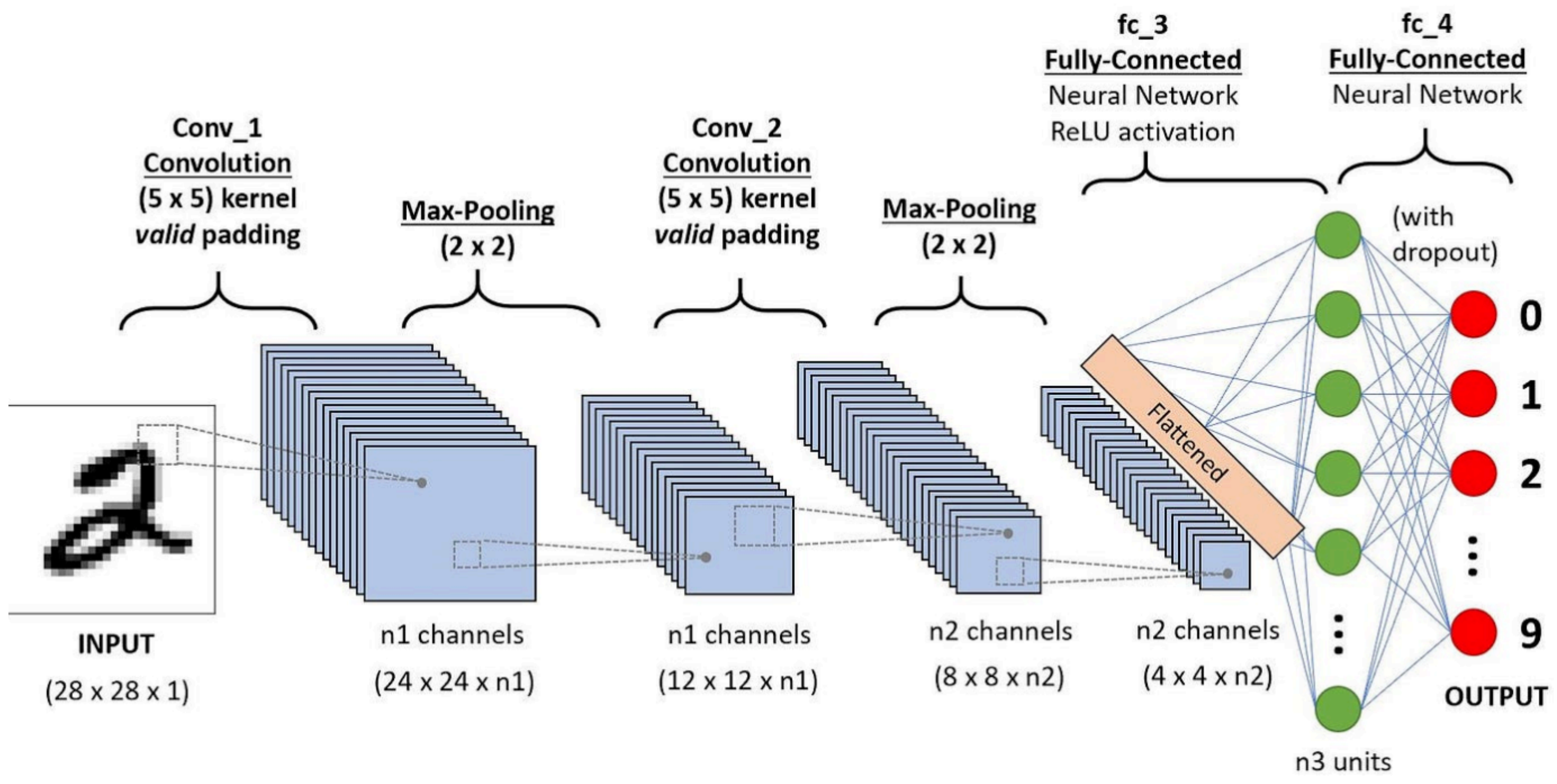
- Filter: (input_channels, output_channels, width, height)
- Filter: (number of channels in each filter, the number of filters, width, height)



Two filters
Stride=2
Zero-padding=1

Multi-channels with multi-filters

- How it will benefit the neural network training?
  - Help to reduce trainable parameters
  - Help the model to focus on local features which might benefit the prediction
  - Keep the trainable parameters the same with different size of images



Reduce trainable parameters

- Number of trainable parameters are equal to the number of parameters of filters.
- Number of trainable parameters $\sim$ number of previous channels * size of filters * number of filters
    - d: the number of previous channels
    - w: is the width of filter
    - h: is the size of filter
    - k: the number of filters
    - The trainable parameters of each layer: $n_t = ((d * w * h) + 1) * k$

Reduce trainable parameters

- The trainable parameters of each layer: $n_t = ((d * w * h) + 1) * k$
- Example:
  - Given image $(3,32,32)$, we want to achieve the neural in the next layer has the shape of $(100,32,32)$
  - We have:
    - The input size: $3 \times 32 \times 32 = 3072$; The output size $100 \times 32 \times 32 = 102400$
  - Linear Operation: $Y = X * W + B$, where W have the shape of $3072 * 102400$ = 314572800 ~ 314m parameters

Reduce trainable parameters

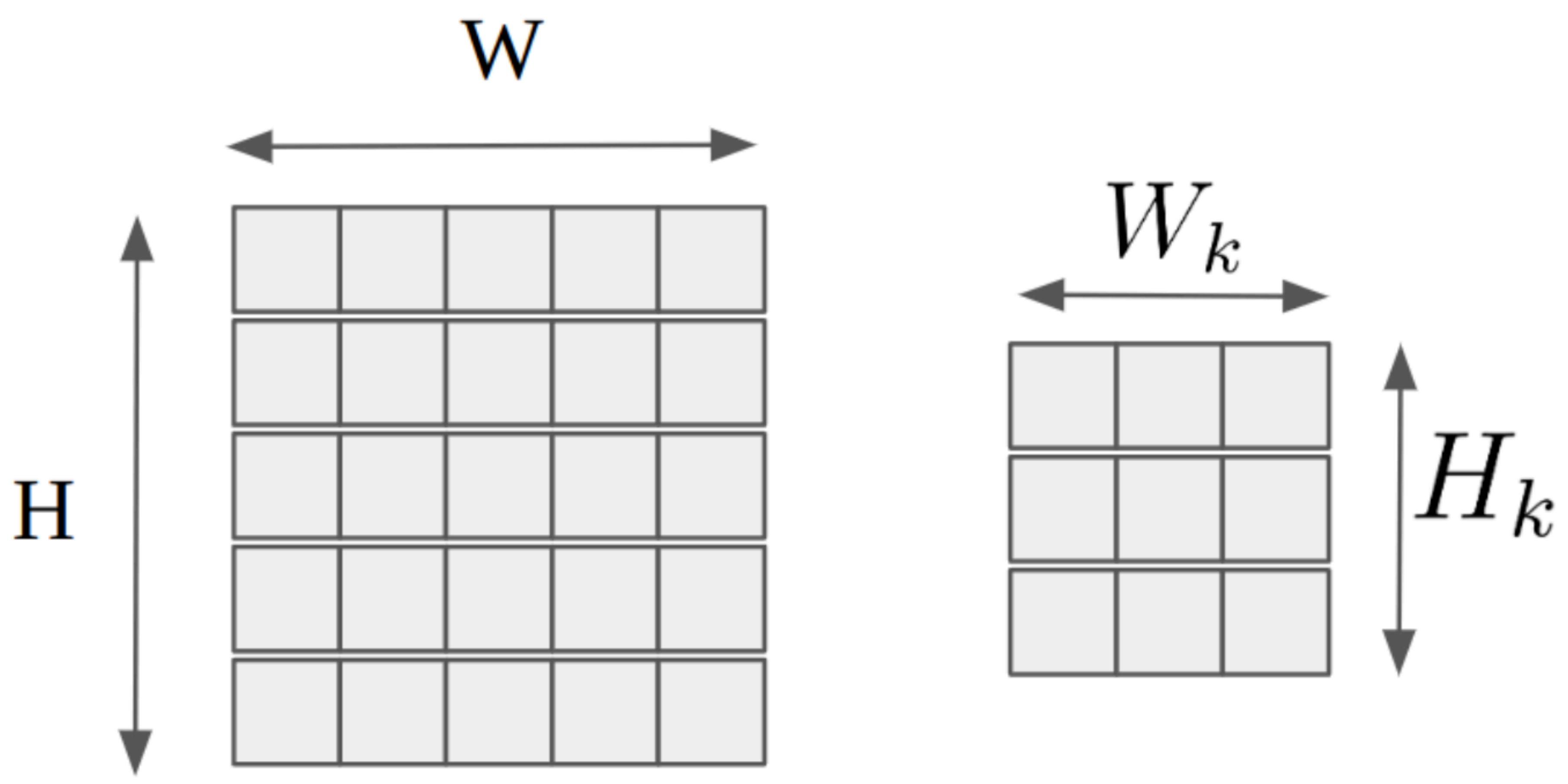
- Example: given image $(3,32,32)$, we want to achieve the neural in the next layer has the shape of $(100,32,32)$
  - Convolution: We need the Convolution layer with size (100, 3, 3) with padding=1 and stride=1 to transform from $(3,32,32)$ to $(100,32,32)$.
  - We have the number of trainable parameters is : $n_t = ((d \times w \times h) + 1) \times k = ((3 \times 3 \times 3) + 1) \times 100 = 1000$
- Comparison
  - MLP: 314572800 parameters V.S. CNN: 1000 parameters

Reduce trainable parameters

- Benefit of reducing trainable parameters:
  - It means the model is much faster to train
  - We can stack much more number of layers (in MLP only 1 or 2 layers, in CNN we can stack 20 - 50 layers).
  - Reduce the model complexity -> reduce overfitting

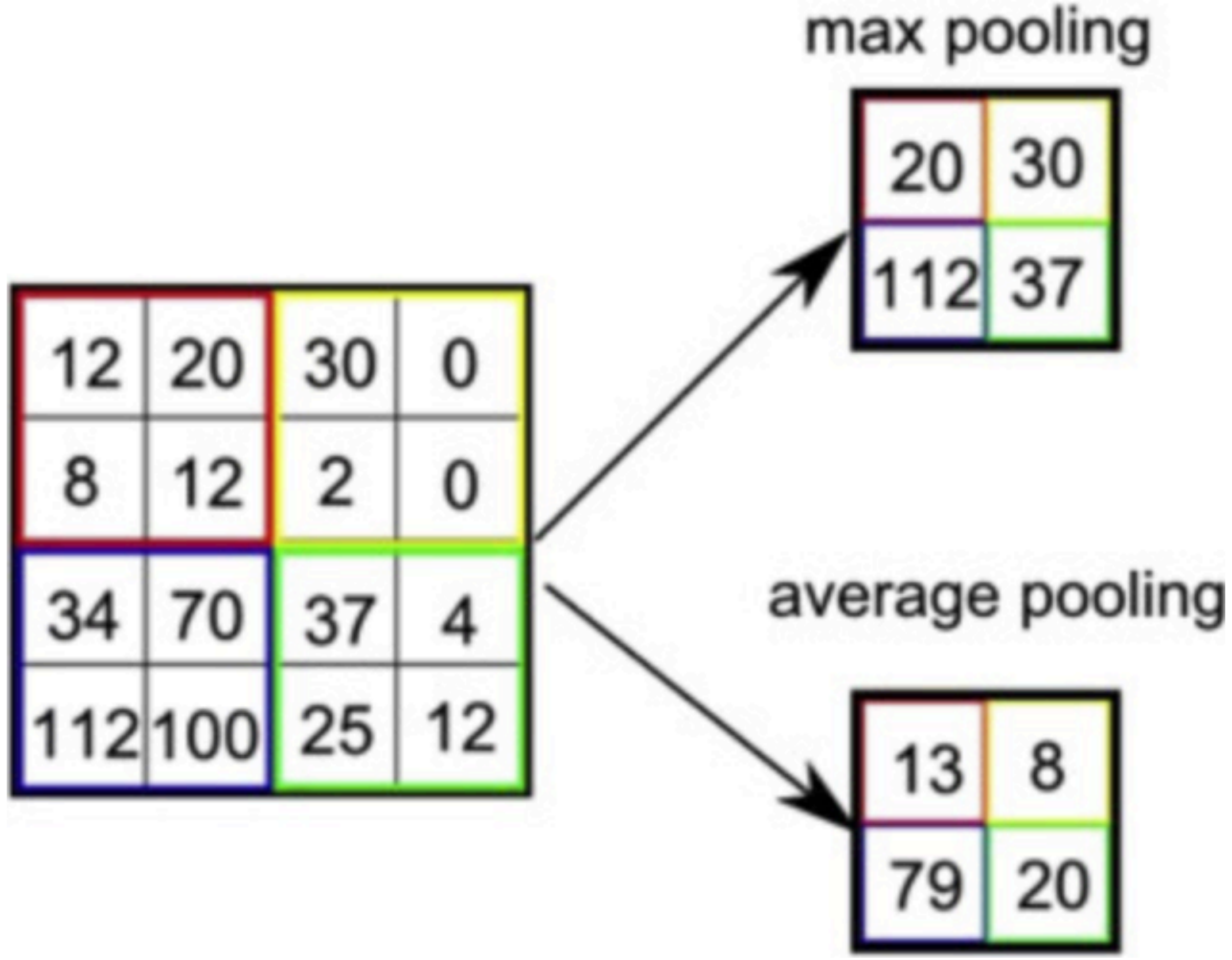
Output Shape after Convolution

- Stride: s; Padding: p;
- Output size:
  - $OS_H = \dfrac{H - H_k + P * 2}{S} + 1$
  - $OS_W = \dfrac{W - W_k + P * 2}{S} + 1$



$P = 1, S = 1$

Pooling layer



Exam-style Question

Code

Code

- ./materials/Week5_CNN/Week5_CNN.ipynb
- ./ResNet/Models/cnn.py
- ./ResNet/Models/resnet.py