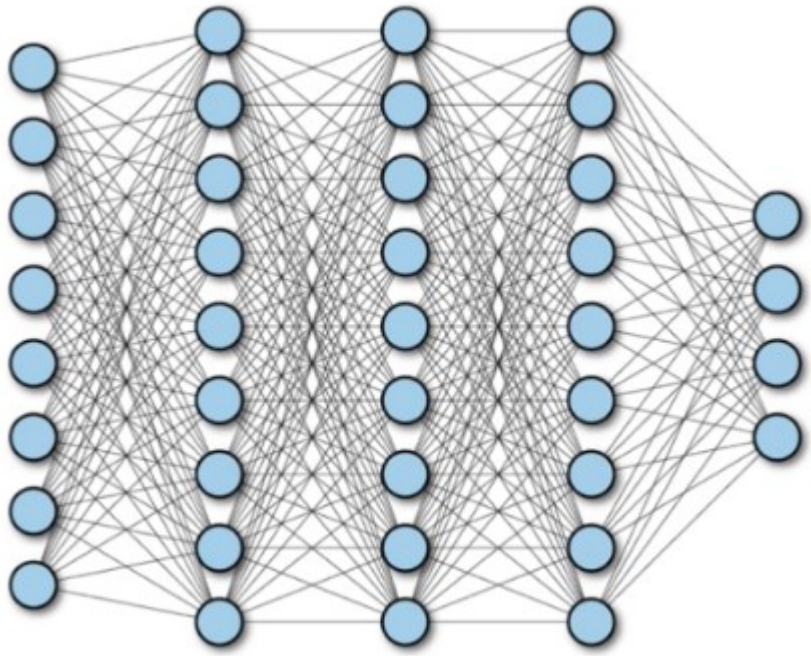
Convolution Neural Networks

Introduction

- Convolutional neural network (CNN, or ConvNet) is a class of artificial neural network, most commonly applied to analyze visual imagery
- CNNs are regularized versions of multilayer perceptrons.
- The "full connectivity" of these networks make them prone to overfitting data.
- Typical ways of regularization, or preventing overfitting, include: weight decay, skipped connections, dropout, etc.
- CNNs take a different approach towards regularization: they take advantage of the **hierarchical pattern** in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters.

Motivation

- The structure of our neural network treats all inputs interchangeably.
- No relationships between the individual inputs
- If order does not matter this kind of architecture is fine
- But image data is different



Motivation

- For Computer vision tasks to feed 2D image(which have spatial structure) to fully connected dense network
- We need to collapse it down to one dimensional vector
- Then all the 2D structure in that image will be gone
- So we lost all the spatial structure in that image

Motivation

- Fully connected network would require a vast number of parameters
- the full connectivity between nodes caused the curse of dimensionality
- For example, in CIFAR-10, images are only of size 32×32×3
- so a single fully connected neuron in the first hidden layer of a regular neural network would have 32*32*3 = 3,072 weights.
- A 200×200 image, however, would lead to neurons that have 200*200*3 = 120,000 weights.

Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Motivation

- Also, such network architecture does not take into account the spatial structure of data
- We need to take advantage of spatial structure while designing the architecture of a Neural Network.
- How can we keep spatial structure in the input while building architecture of network

Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

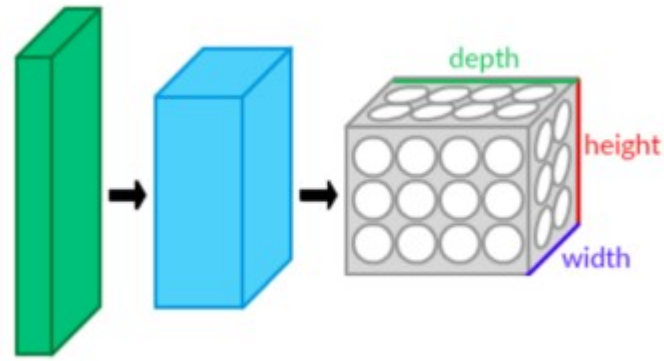
Motivation

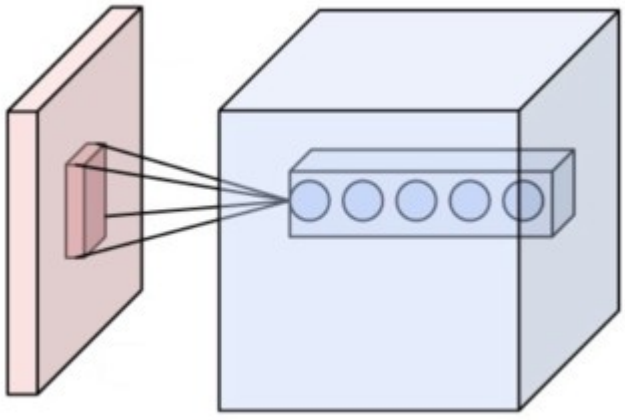
- Convolutional neural networks are the models mitigate the challenges posed by the MLP architecture by exploiting the strong spatially local correlation present in natural images.
- As opposed to MLPs, CNNs have the following distinguishing features:
  - 3D volumes of neurons
  - Local connectivity
  - Shared weights
  - Pooling

Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

3D volumes of neurons

- The layers of a CNN have neurons arranged in 3 dimensions: width, height and depth.
- Each neuron inside a convolutional layer is connected to only a **small region** of the layer before it, called a **receptive field**
- Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.

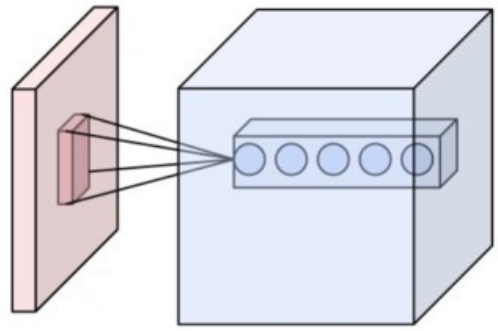


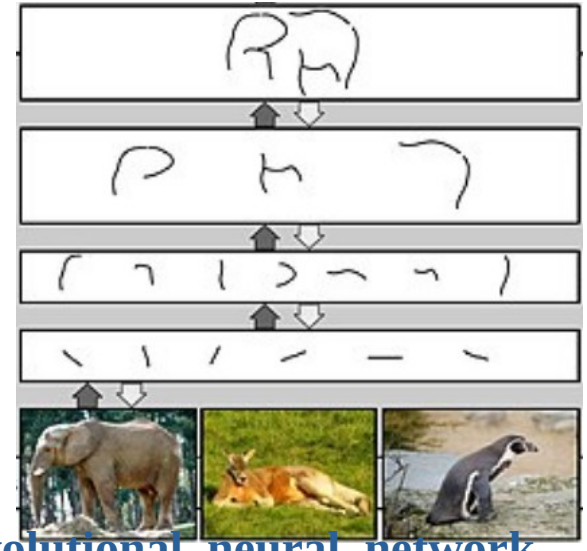


Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Local connectivity

- CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers.
- The architecture thus ensures that the learned "filters" produce the strongest response to a spatially local input pattern.
- Stacking many such layers leads to non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.

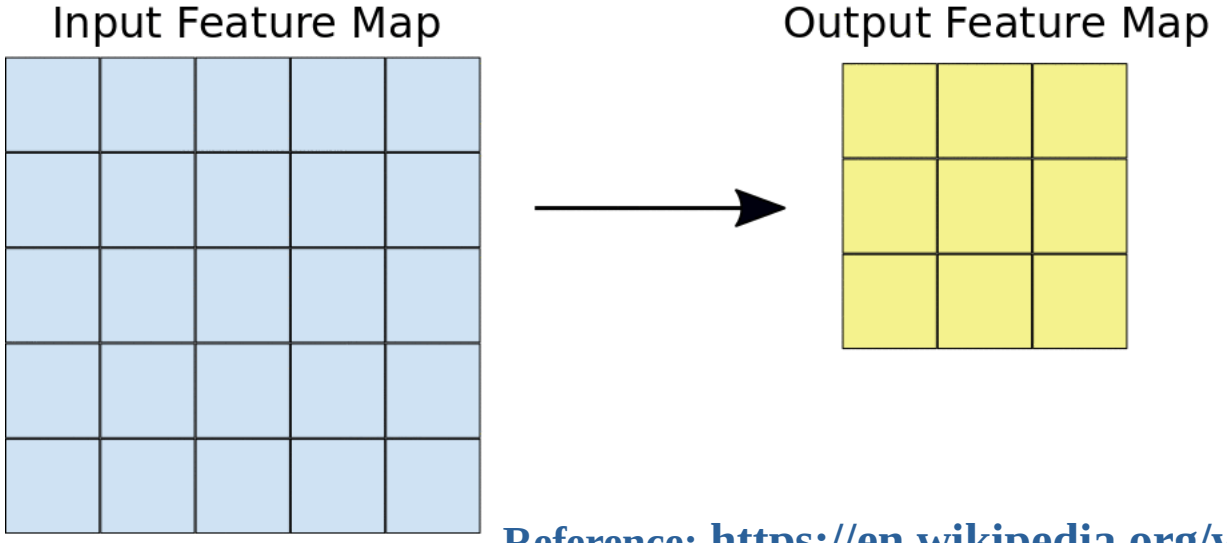




Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Shared weights

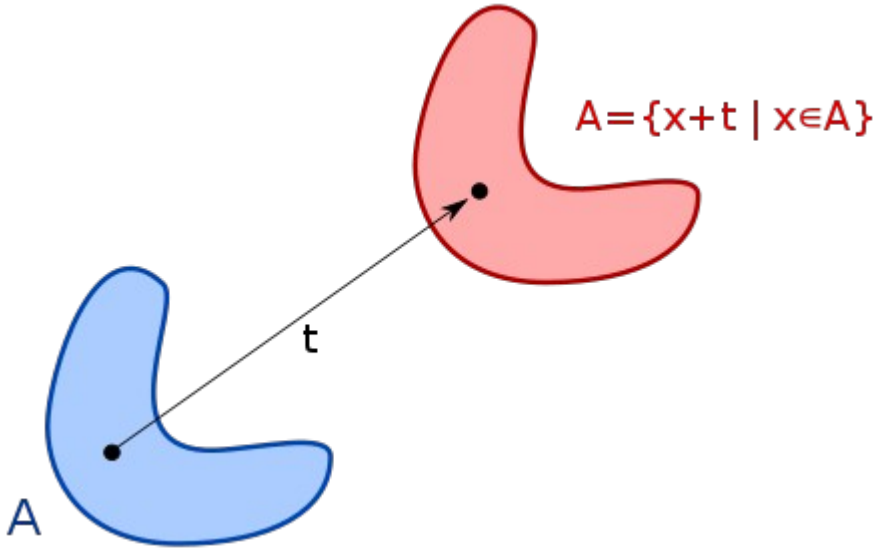
- In CNNs, each filter is replicated across the entire visual field.
- These replicated units share the same parameterization (weight vector and bias) and form a feature map.

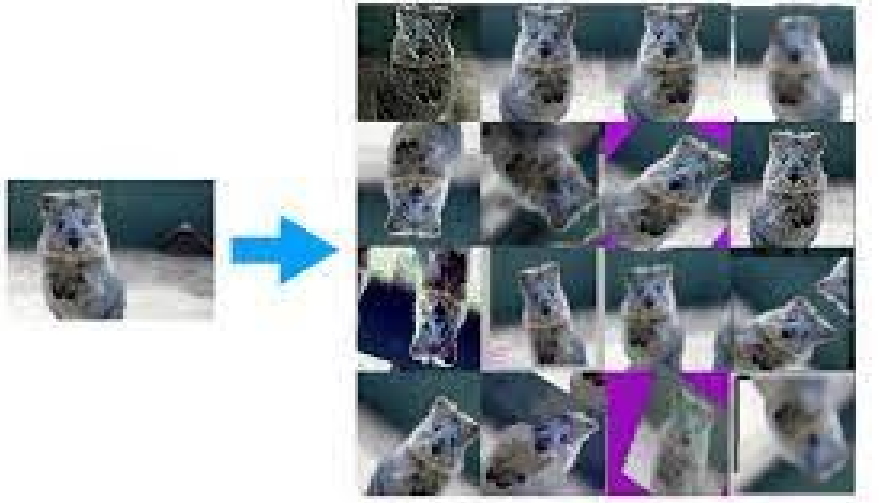


Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Shared weights

- Replicating units in this way allows for the resulting activation map to be equivariant under shifts of the locations of input features in the visual field

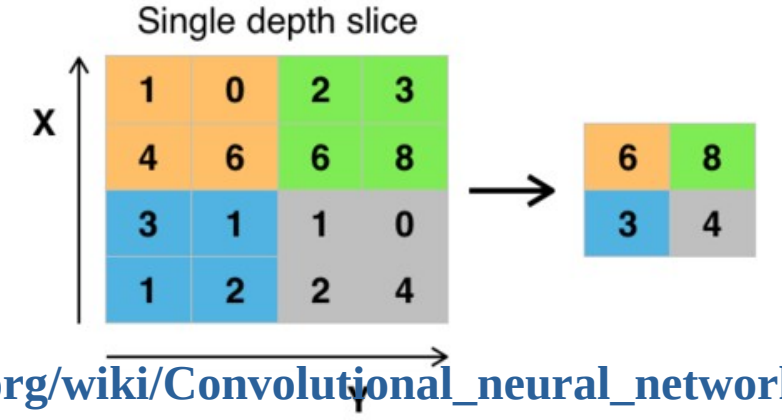




Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Pooling

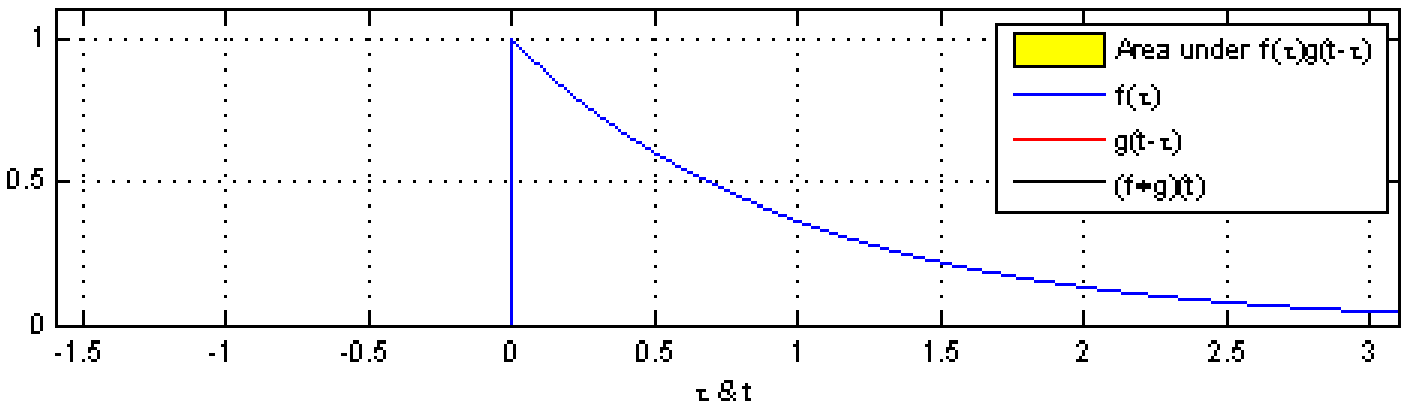
- In a CNN's pooling layers, feature maps are divided into rectangular sub-regions, and the features in each rectangle are independently down-sampled to a single value, commonly by taking their average or maximum value.
- In addition to reducing the sizes of feature maps, the pooling operation grants a degree of local translational invariance to the features contained therein, allowing the CNN to be more robust to variations in their positions.



Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Convolution Operation

- Convolution is mathematical operation it measures how much two functions overlap as one passes over the other.



Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Kernel

- A kernel is a grid of weights “overlaid” on image, centered on one pixel
- Each weight multiplied with pixel underneath it
- Output over the centered pixel is $\sum_{p=1}^{P} w_p . pixel_p$
- Used for traditional image processing techniques:
  - Blur
  - Sharpen
  - Edge detectio

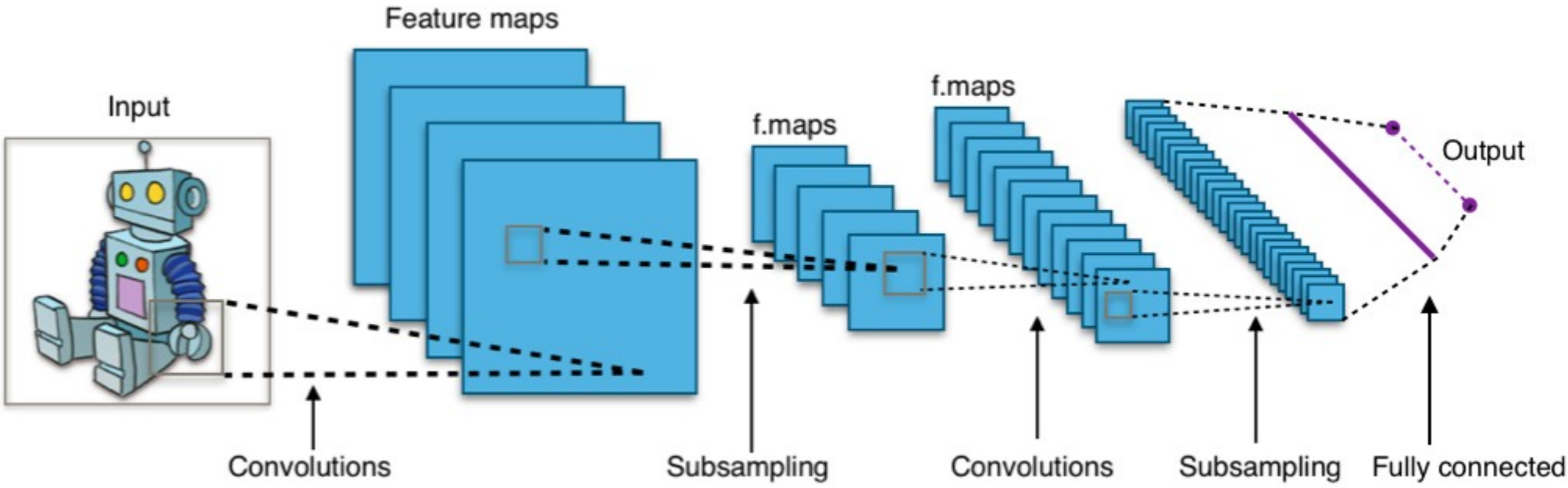
Input Feature Map

3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

Convolutional Filter

1	0	0
1	1	0
0	0	1

Typical CNN architecture



Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Convolution

- A convolution extracts tiles of the input feature map
- And performs a dot product of the convolution kernel with the layer's input matrix extracted tiles
- Convolutions are defined by two parameters:
  - **Size of the tiles** that are extracted (typically 3x3 or 5x5 pixels).
  - The **depth of the output feature map**, which corresponds to the number of filters that are applied.

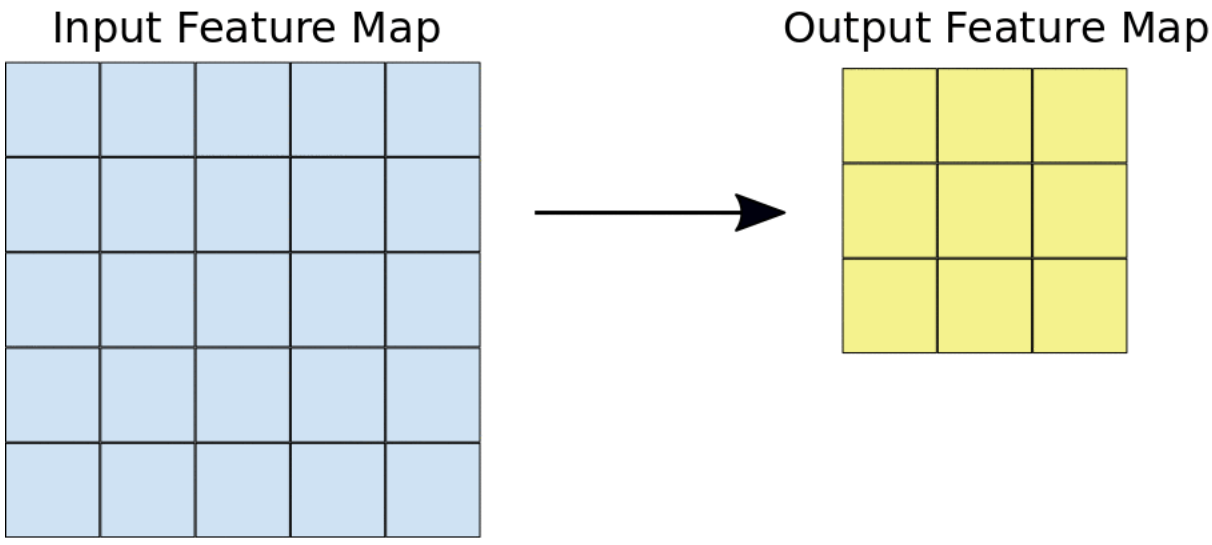
Reference: **https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks**

Convolution

- During a convolution, the filters effectively slide over the input feature map's grid horizontally and vertically, one pixel at a time, extracting each corresponding tile

3x3 convolution of depth 1 performed over a 5x5 input feature map, also of depth 1.

There are nine possible 3x3 locations to extract tiles from the 5x5 feature map



Reference: **https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks**

Convolution

Input Feature Map

3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

5x5 Input Feature map (depth 1)

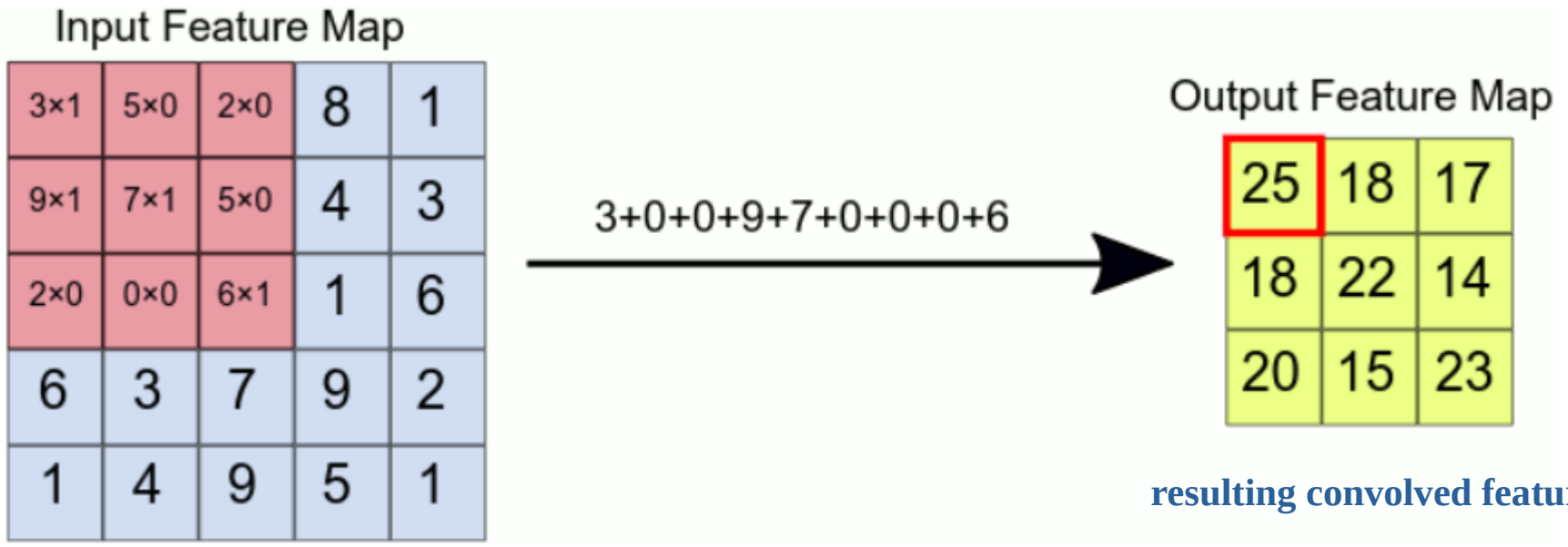
Convolutional Filter

1	0	0
1	1	0
0	0	1

3x3 Convolutional Filter (depth 1)

Reference: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Convolution

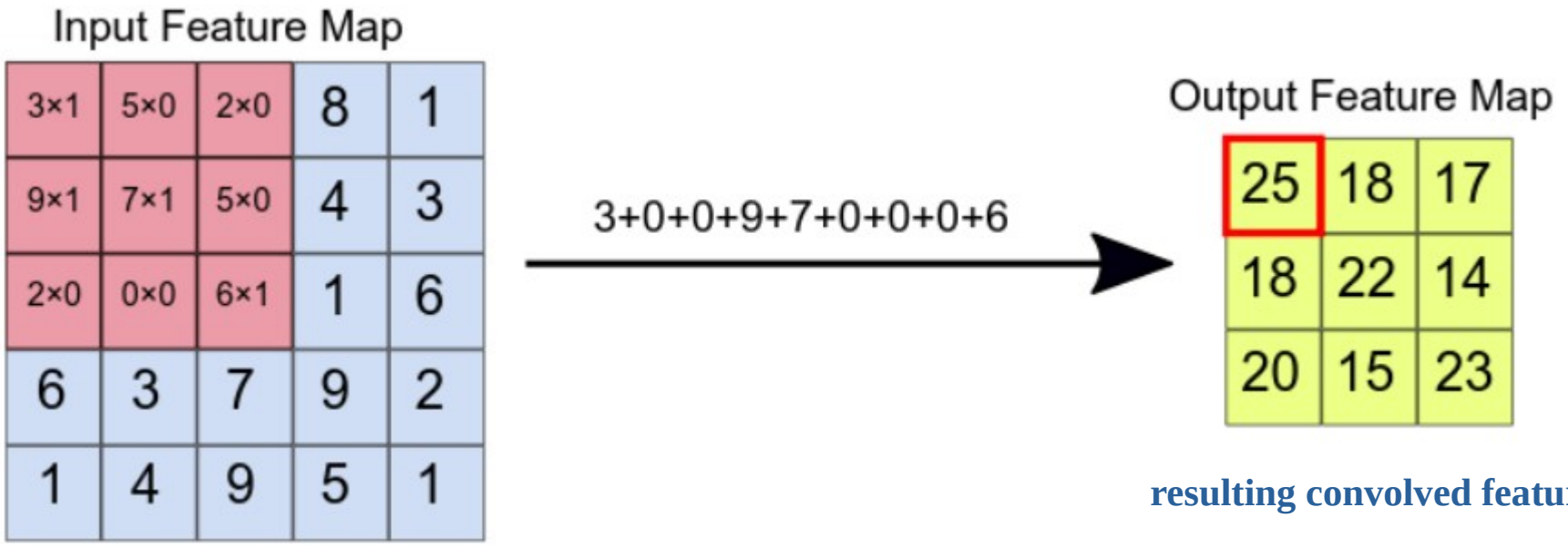


resulting convolved feature

3x3 convolution is performed on the 5x5 input feature map

Reference: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Convolution



resulting convolved feature

3x3 convolution is performed on the 5x5 input feature map

Reference: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Kernels as Feature Detectors

- Can think of kernels as a ”local feature detectors”

-1	1	-1
-1	1	-1
-1	1	-1

Vertical Line Detector

-1	-1	-1
1	1	1
-1	-1	-1

Horizontal Line Detector

-1	-1	1
-1	1	-1
1	-1	-1

Diagonal Line Detector

Reference: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Kernels as Feature Detectors

0	0	10	0	0	0
0	0	10	0	0	0
0	0	10	0	0	0
10	10	10	0	0	0
0	0	10	0	0	0
0	0	10	0	0	0

-1	1	-1
-1	1	-1
-1	1	-1

-30	30	-30	0
-30	20	-30	0
-30	20	-30	0
-30	20	-30	0

-1	-1	-1
1	1	1
-1	-1	-1

-10	-10	-10	0
-30	-20	-10	0
10	0	-10	0
-30	-20	-10	0

Stride

- The ”step size” as the kernel moves across the image
- Can be different for vertical and horizontal steps (but usually is the same value)
- When stride is greater than 1, it scales down the output dimension

Reference: **https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks**

Output Size

Output size: $\frac{N-F}{S}+1$

N = 7, F = 3

Stride S =1 => ((7-3)/1)+1 =5

Stride S =2 => ((7-3)/2)+1 =3

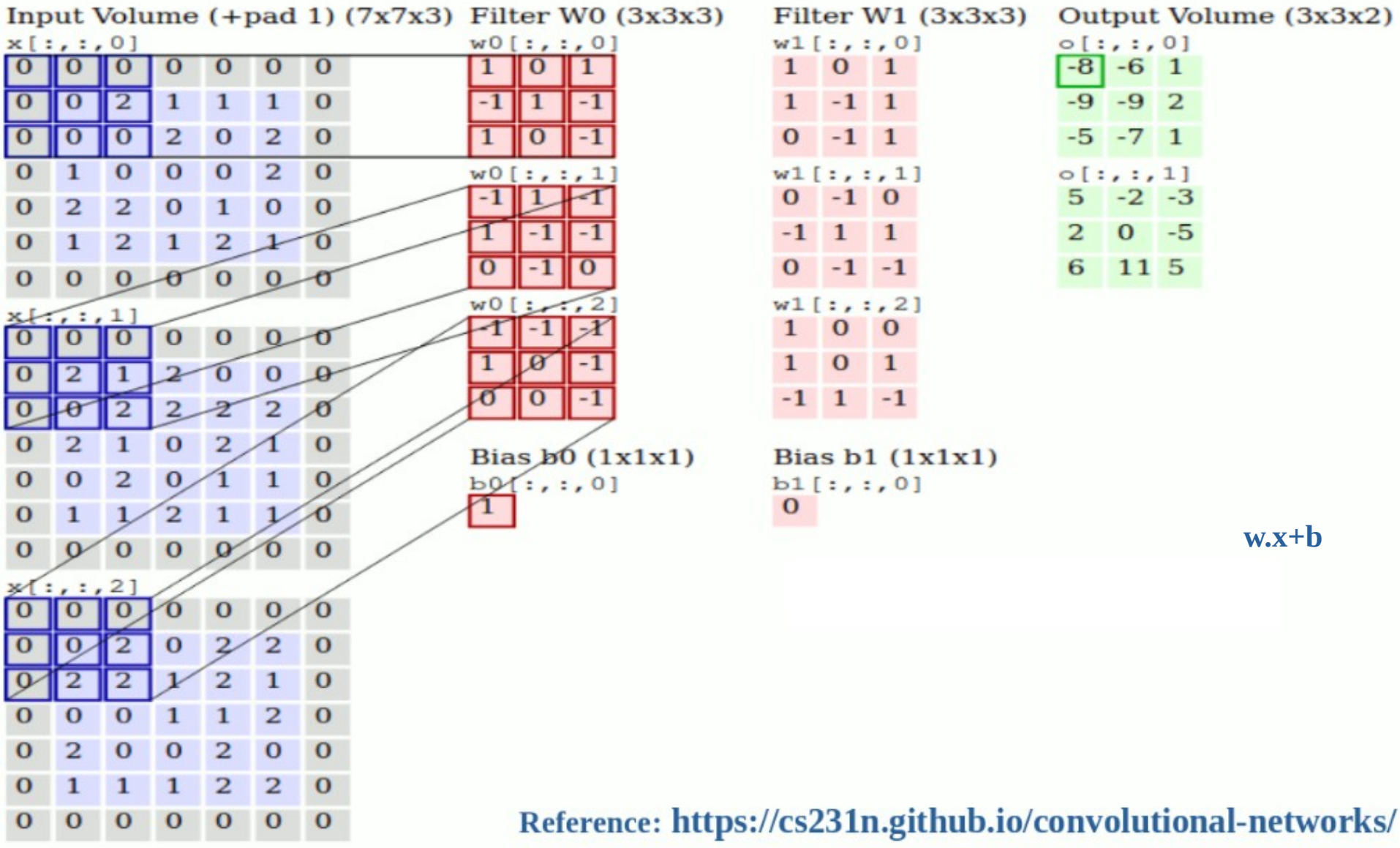
Stride S =3 => ((7-3)/3)+1 =2.33

Reference: Andrej Karpathy https://cs231n.github.io/convolutional-networks/

Padding

- The output feature map (5x5) is smaller than the input feature map (7x7).
- Input convoluted repeatedly shrink output volumes spatially
- Shrinking too fast is not good, does not work well
- If you instead want the output feature map to have the same dimensions as the input feature map
- you can add zero-padding

Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 2 1 1 1 0
0 0 0 2 0 2 0
0 1 0 0 0 2 0
0 2 2 0 1 0 0
0 1 2 1 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 1 2 0 0 0
0 0 2 2 2 2 0
0 2 1 0 2 1 0
0 0 2 0 1 1 0
0 1 1 2 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 0 2 2 0
0 2 2 1 2 1 0
0 0 0 1 1 2 0
0 2 0 0 2 0 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 0 1
-1 1 -1
1 0 -1
w0[:,:,1]
-1 1 -1
1 -1 -1
0 -1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 0 1
1 -1 1
0 -1 1
w1[:,:,1]
0 -1 0
-1 1 1
0 -1 -1
w1[:,:,2]
1 0 0
1 0 1
-1 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 -6 1
-9 -9 2
-5 -7 1
o[:,:,1]
5 -2 -3
2 0 -5
6 11 5
w.x+b
Reference: https://cs231n.github.io/convolutional-networks/

Padding

- Using Kernels directly, there will be an “edge effect”
- Pixels near the edge will not be used as “center pixels” since there are not enough surrounding pixels
- Padding adds extra pixels around the frame
- So every pixel of the original image will be a center pixel as the kernel moves across the image

Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Output size with padding

Output size: $\frac{N-F+2P}{S}+1$

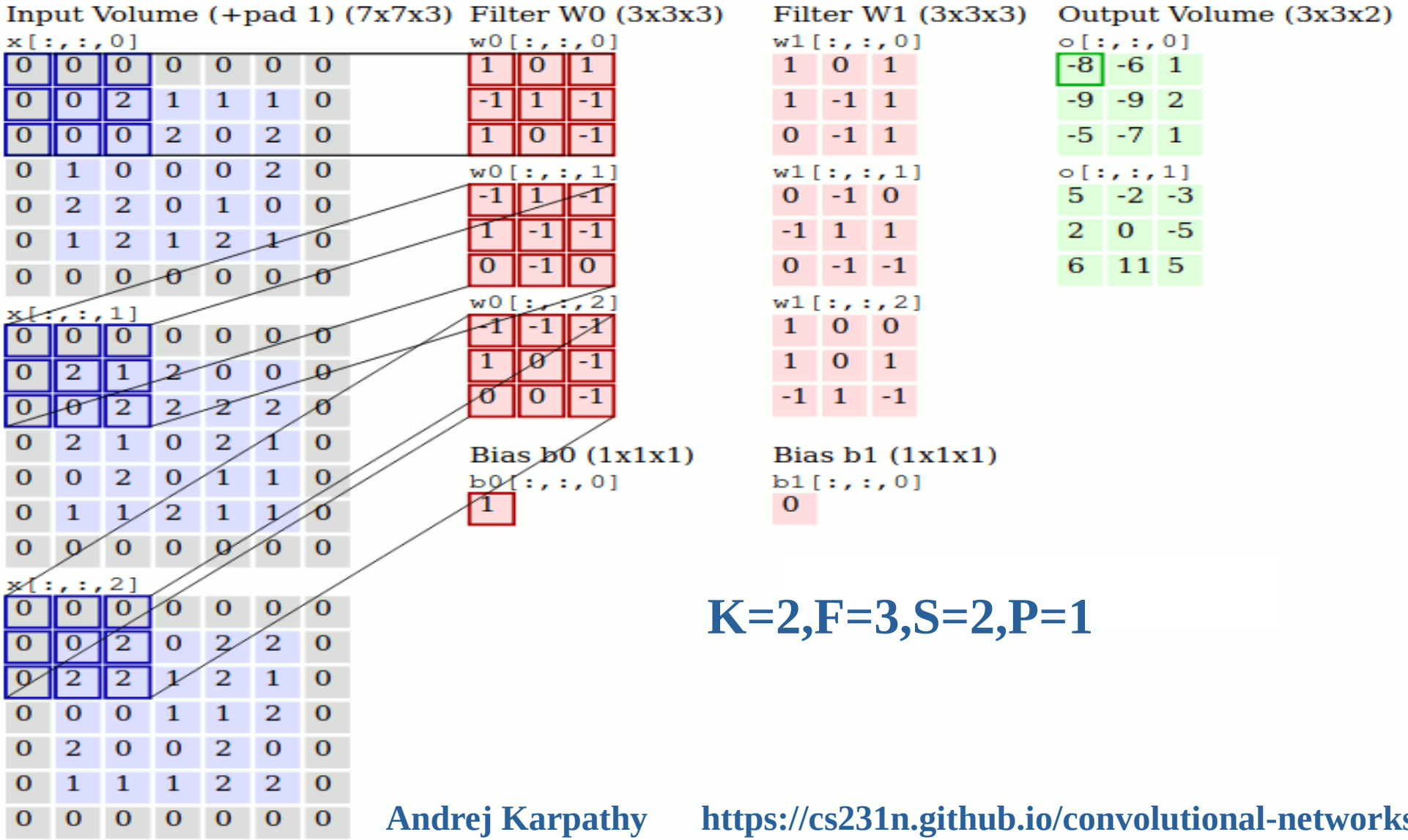
N = 7, F = 3, P=1

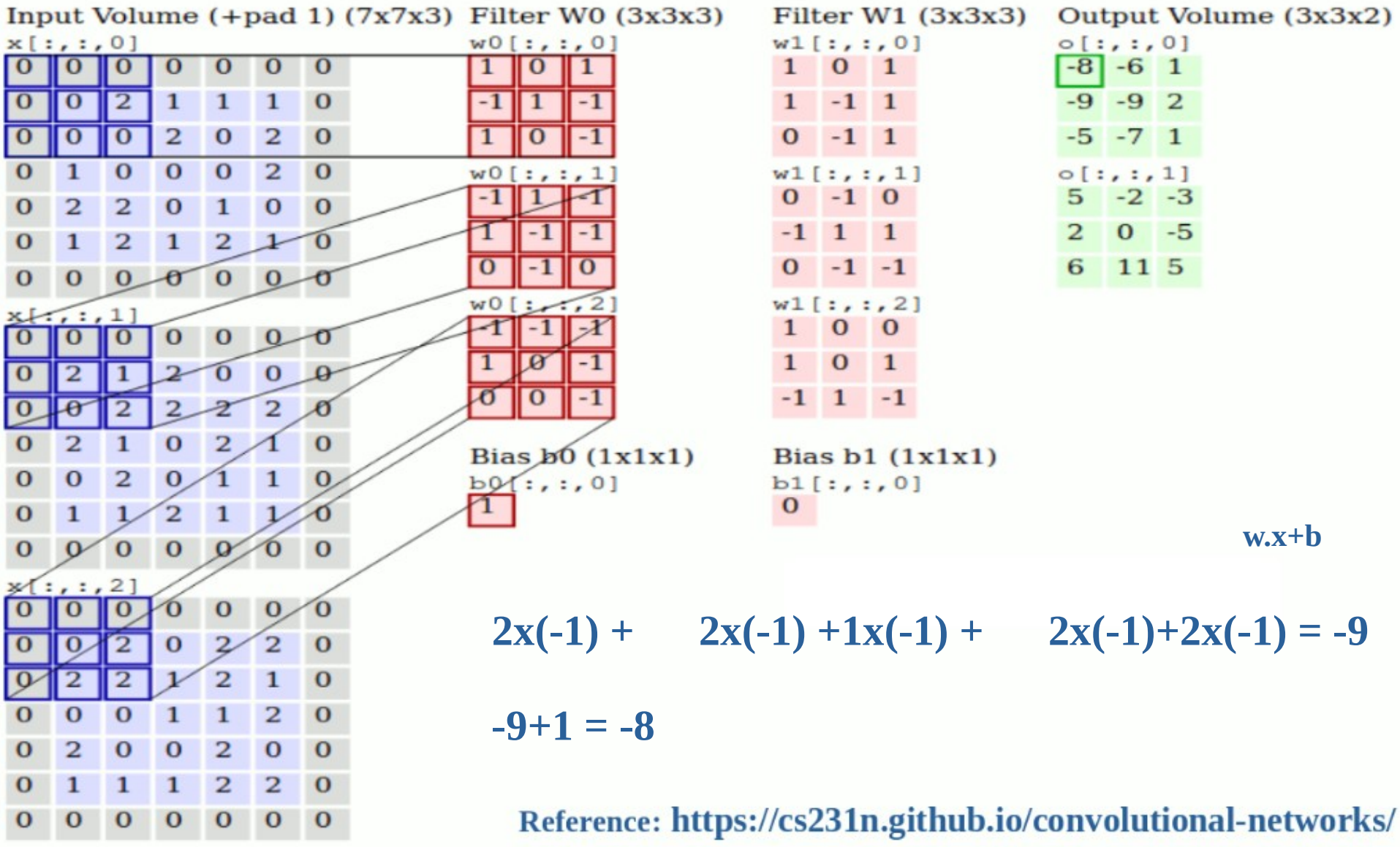
Stride S =1 => ((7-3+2)/1)+1 =7

Stride S =2 => ((7-3+2)/2)+1 =4

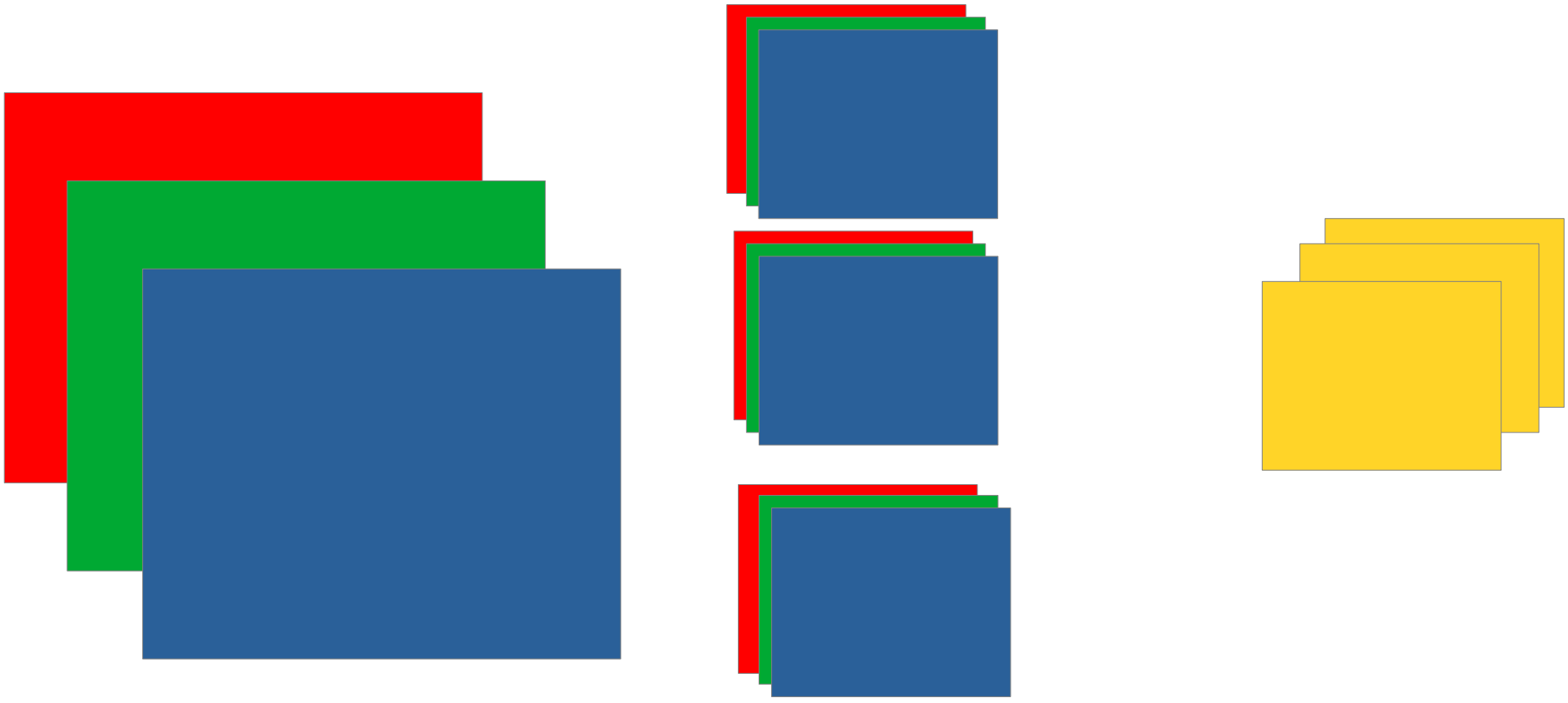
Stride S =3 => ((7-3+2)/3)+1 =3

Reference: Andrej Karpathy https://cs231n.github.io/convolutional-networks/

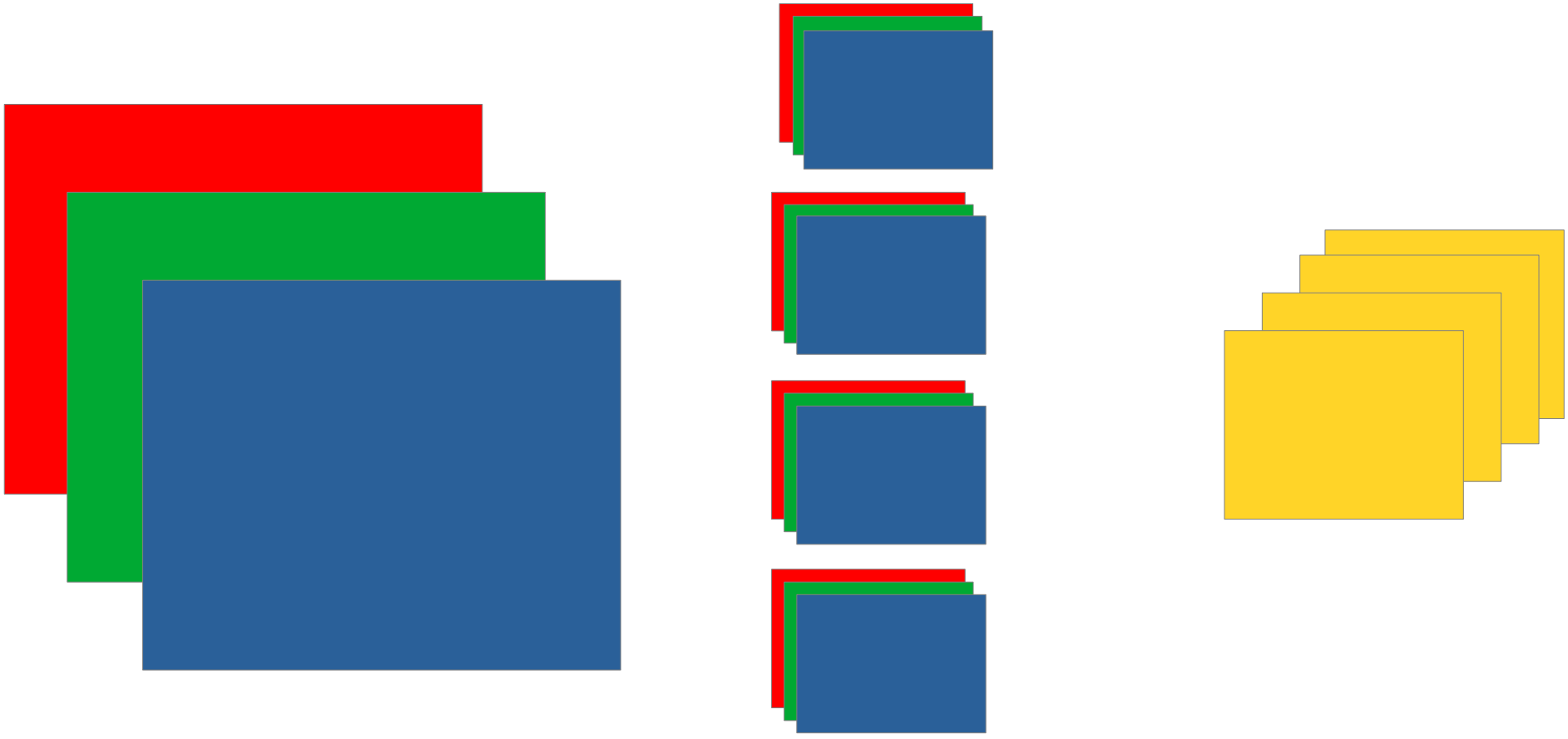

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 2 1 1 1 0
0 0 0 2 0 2 0
0 1 0 0 0 2 0
0 2 2 0 1 0 0
0 1 2 1 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 1 2 0 0 0
0 0 2 2 2 2 0
0 2 1 0 2 1 0
0 0 2 0 1 1 0
0 1 1 2 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 0 2 2 0
0 2 2 1 2 1 0
0 0 0 1 1 2 0
0 2 0 0 2 0 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 0 1
-1 1 -1
1 0 -1
w0[:,:,1]
-1 1 -1
1 -1 -1
0 -1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 0 1
1 -1 1
0 -1 1
w1[:,:,1]
0 -1 0
-1 1 1
0 -1 -1
w1[:,:,2]
1 0 0
1 0 1
-1 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 -6 1
-9 -9 2
-5 -7 1
o[:,:,1]
5 -2 -3
2 0 -5
6 11 5
K=2,F=3,S=2,P=1
Andrej Karpathy
https://cs231n.github.io/convolutional-networks/


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 2 1 1 1 0
0 0 0 2 0 2 0
0 1 0 0 0 2 0
0 2 2 0 1 0 0
0 1 2 1 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 1 2 0 0 0
0 0 2 2 2 2 0
0 2 1 0 2 1 0
0 0 2 0 1 1 0
0 1 1 2 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 0 2 2 0
0 2 2 1 2 1 0
0 0 0 1 1 2 0
0 2 0 0 2 0 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 0 1
-1 1 -1
1 0 -1
w0[:,:,1]
-1 1 -1
1 -1 -1
0 -1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 0 1
1 -1 1
0 -1 1
w1[:,:,1]
0 -1 0
-1 1 1
0 -1 -1
w1[:,:,2]
1 0 0
1 0 1
-1 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 -6 1
-9 -9 2
-5 -7 1
o[:,:,1]
5 -2 -3
2 0 -5
6 11 5
w.x+b
2x(-1) + 2x(-1) +1x(-1) + 2x(-1)+2x(-1) = -9
-9+1 = -8
Reference: https://cs231n.github.io/convolutional-networks/

Convolution



Convolution



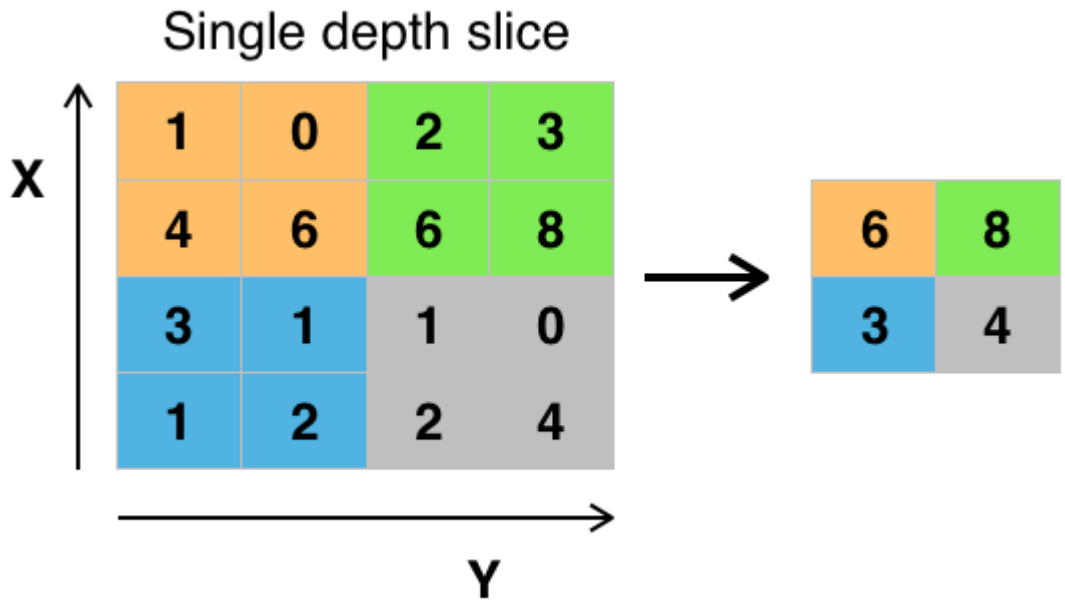
Convolution layers activation function is commonly ReLU.

Pooling Layer

- **Idea:** Reduce the image size by mapping a patch of pixels to a single value.
- Pooling layers reduce the dimensions of data
- to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- Does not have parameters
- There are two common types of pooling in popular use: max and average pooling

Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Pooling Layer



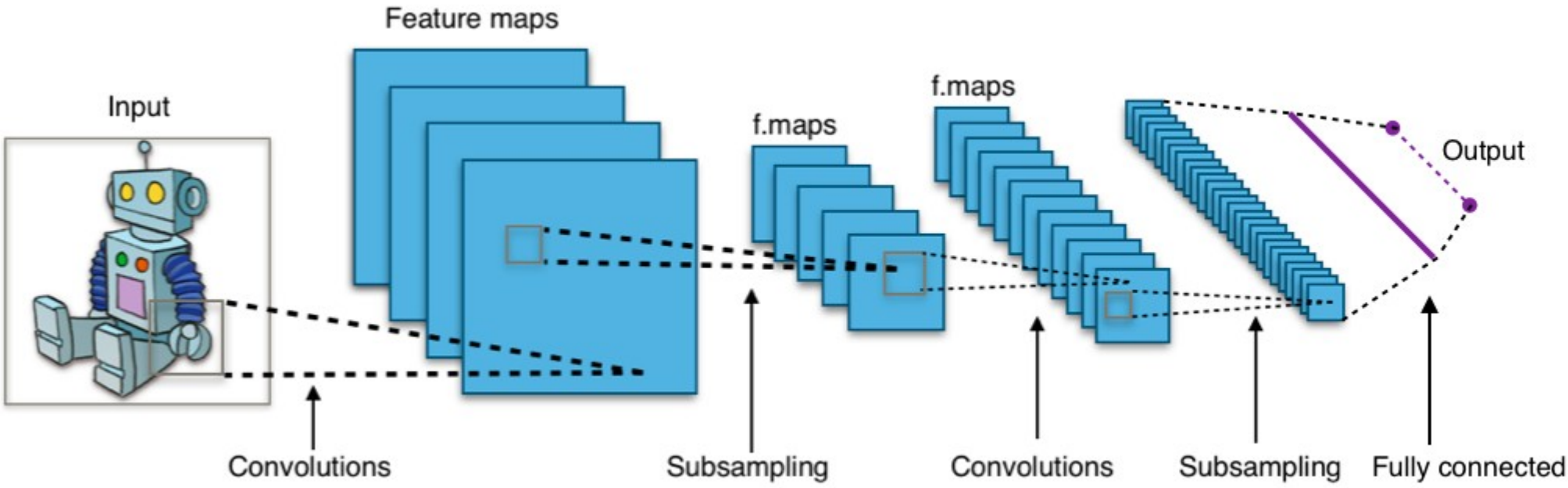
Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Fully-connected layer

- At the end of a convolutional neural network are one or more fully connected layers (when two layers are "fully connected," every node in the first layer is connected to every node in the second layer).
- Their job is to perform classification based on the features extracted by the convolutions.
- Typically, the final fully connected layer contains a softmax activation function, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict.

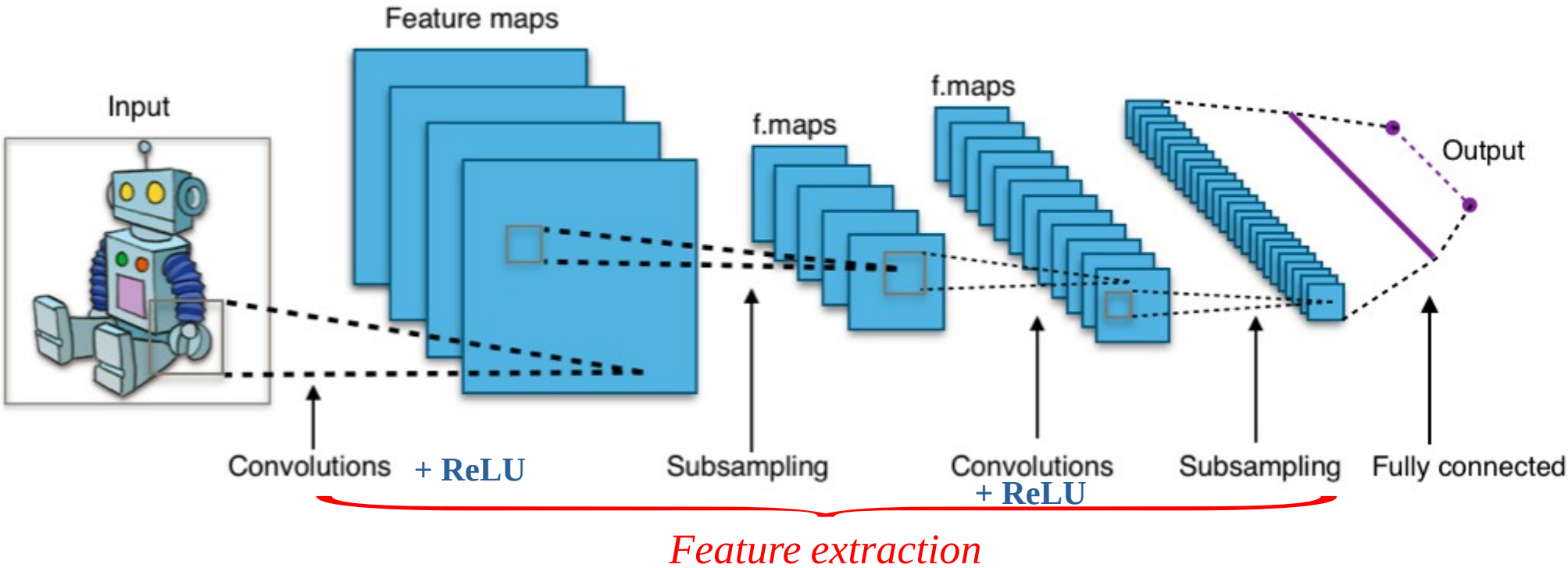
Reference: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Typical CNN architecture



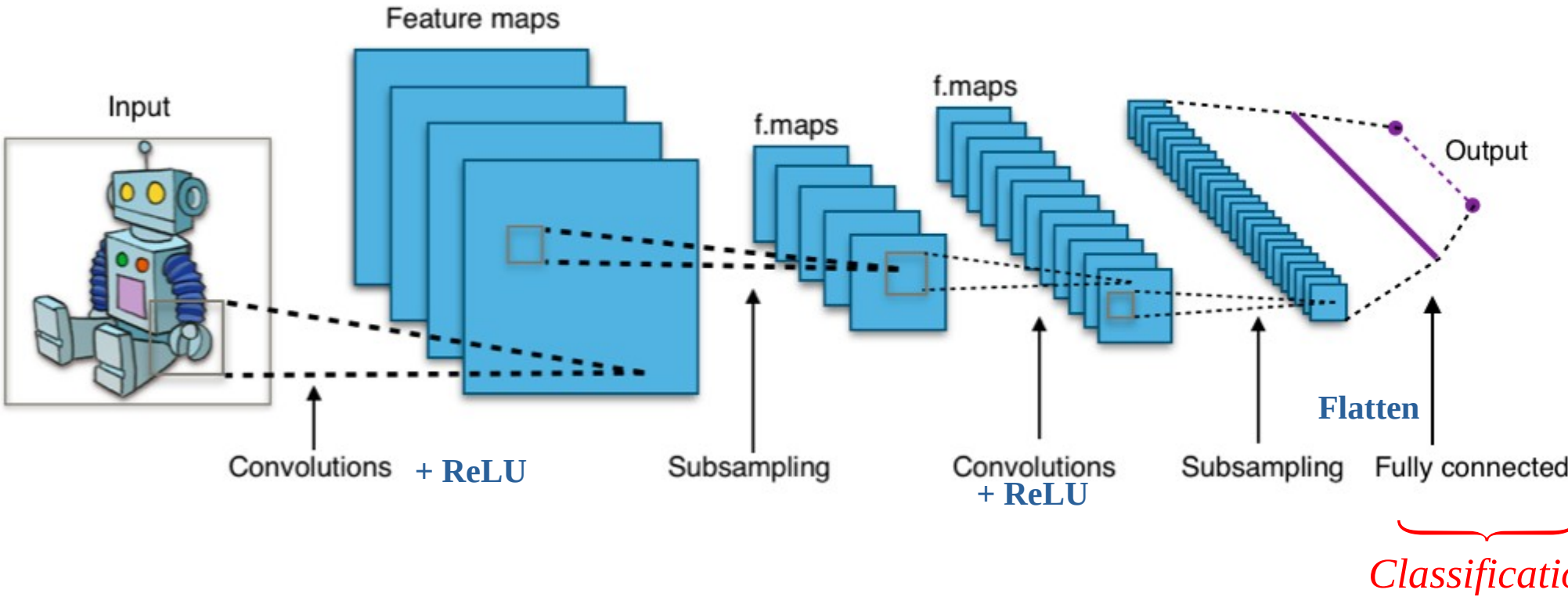
Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Typical CNN architecture



Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network

Typical CNN architecture



Reference: https://en.wikipedia.org/wiki/Convolutional_neural_network