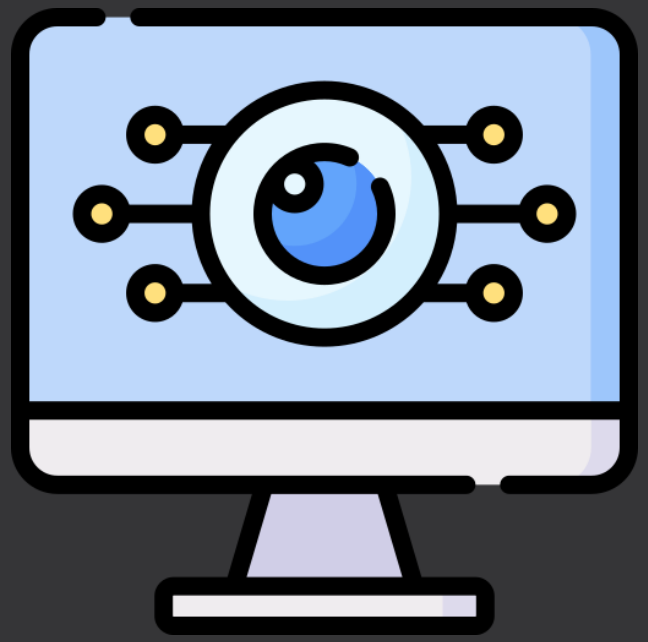
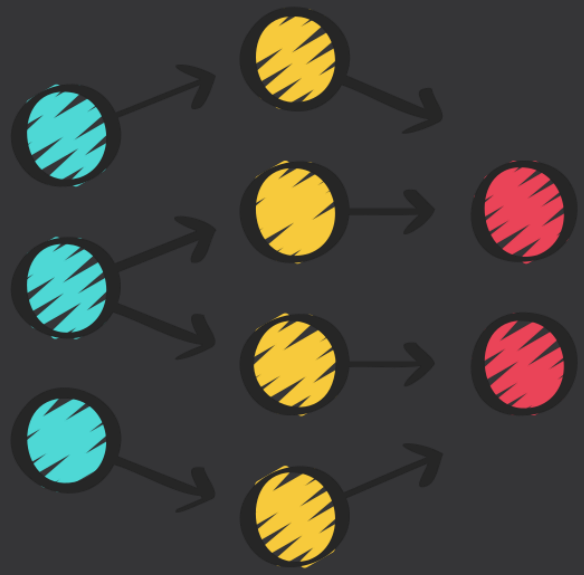




Convolutional Neural Networks

Lecture 16 – HCCDA-AI

Dr. Muhammad Sajjad

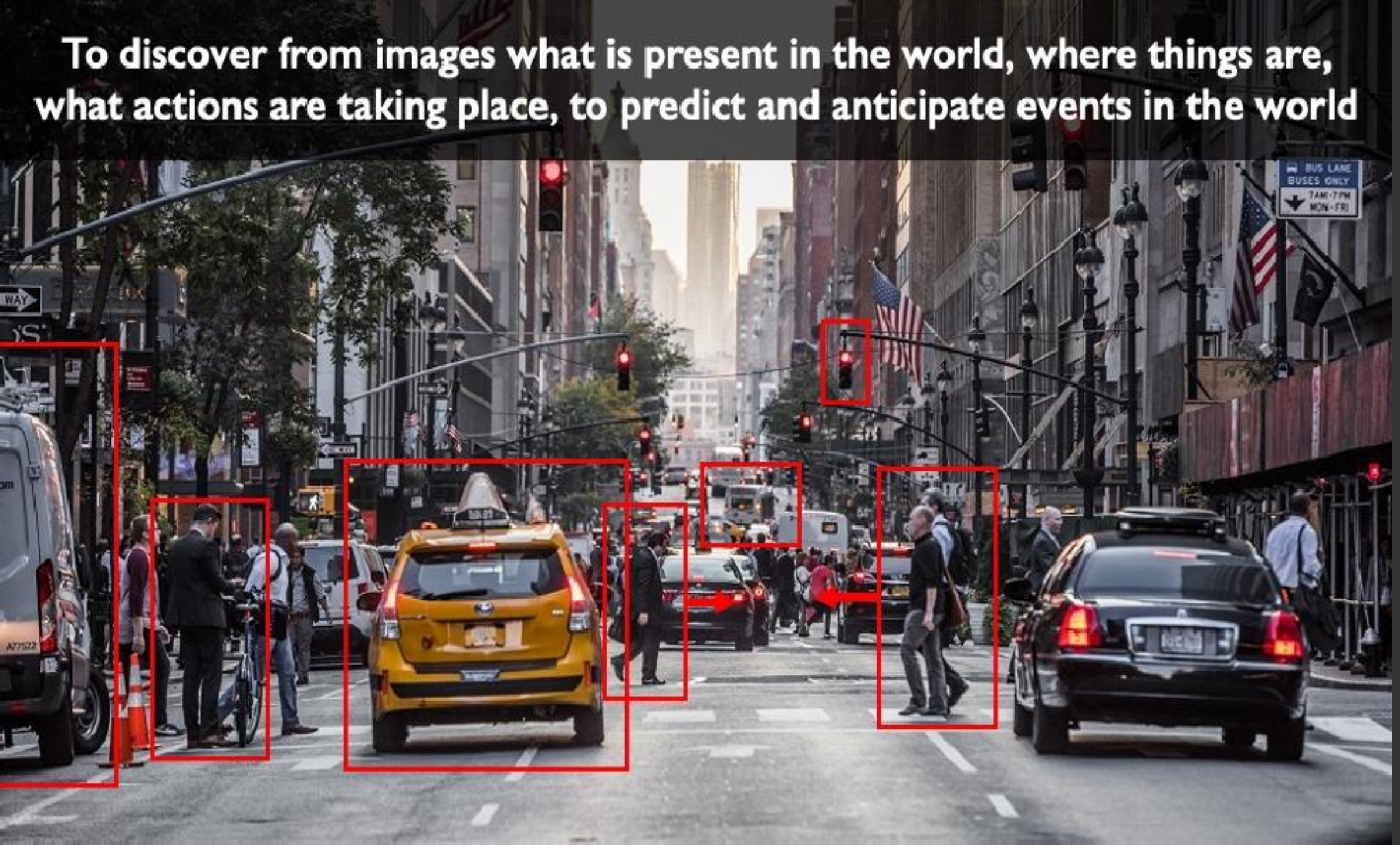
R.A: Kaleem Ullah

R.A: Imran Nawar

27 July 2025

“To know what is where by looking”

To discover from images what is present in the world, where things are, what actions are taking place, to predict and anticipate events in the world



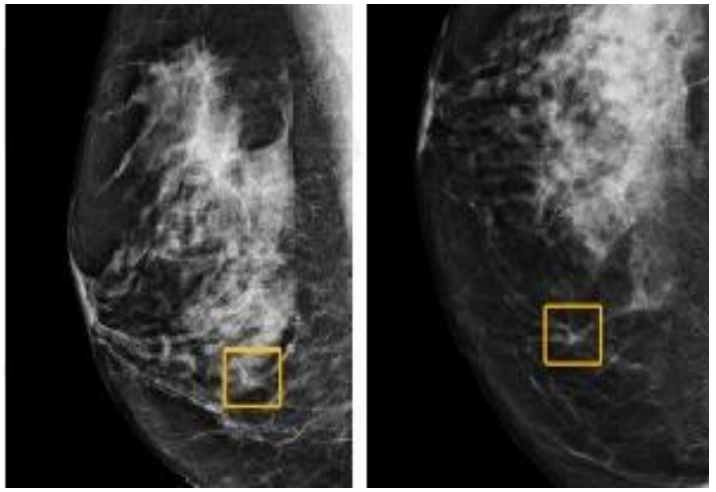
The rise and impact of Computer Vision

Robotics

Mobile Computing

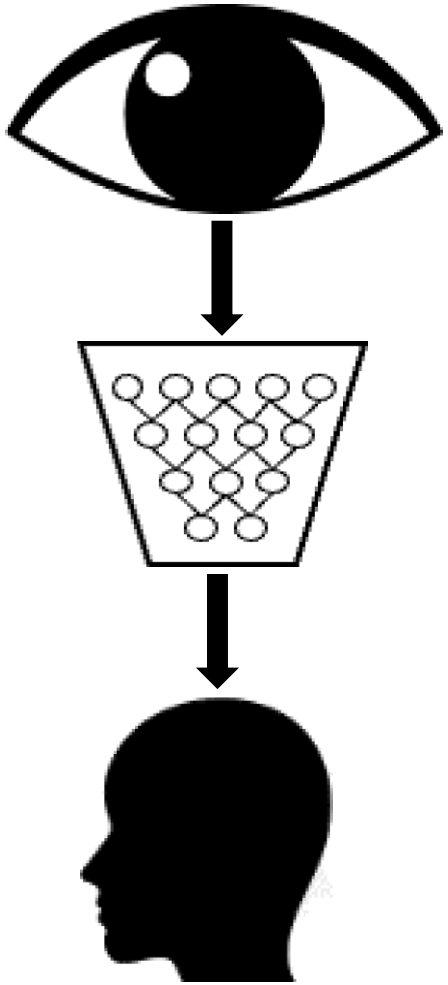
Accessibility

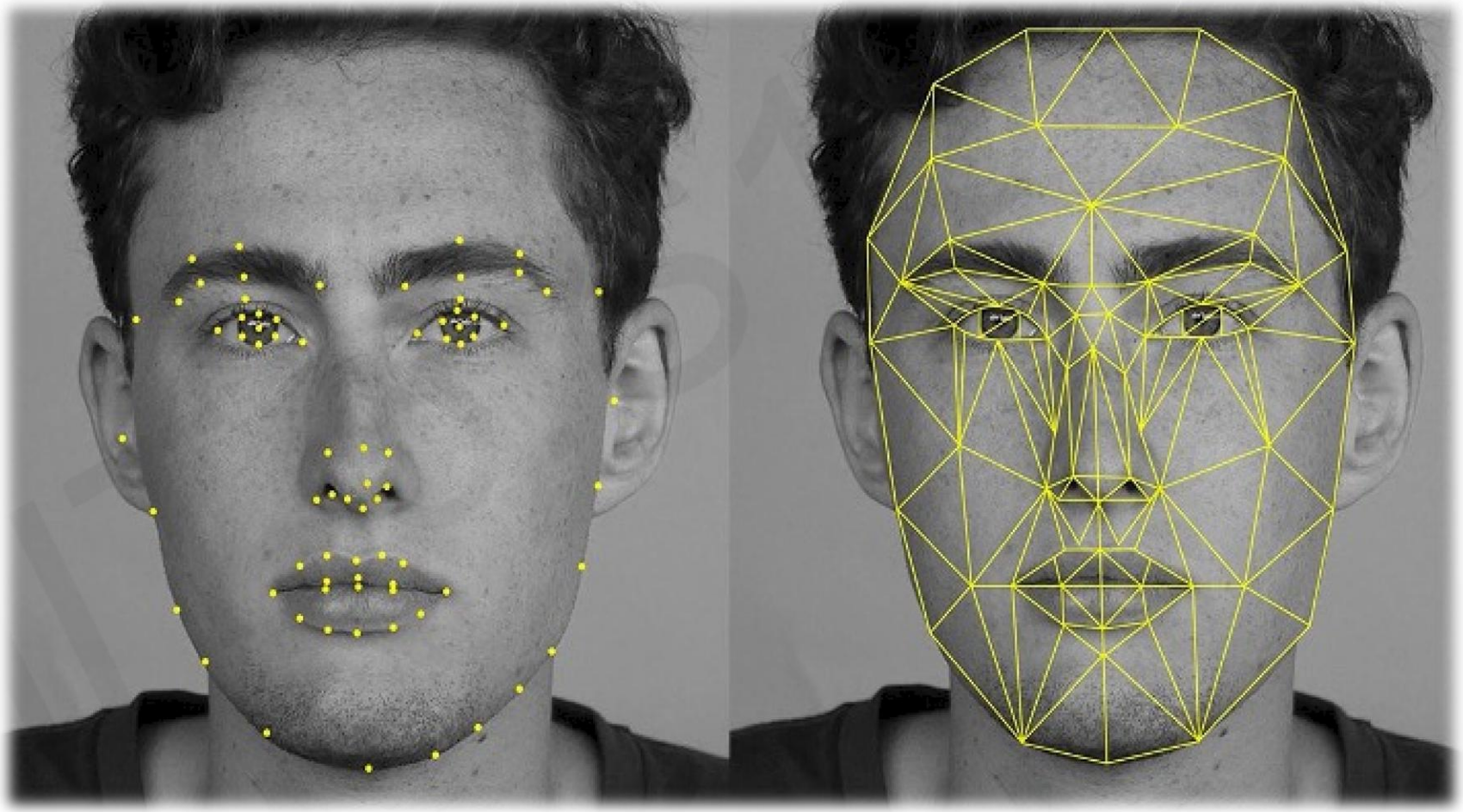
Biology & Medicine



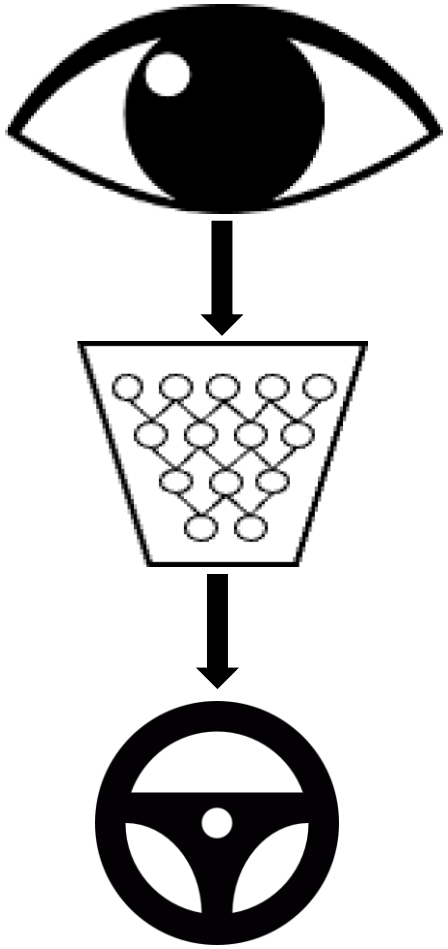
Autonomous driving

Impact: Facial Detection & Recognition



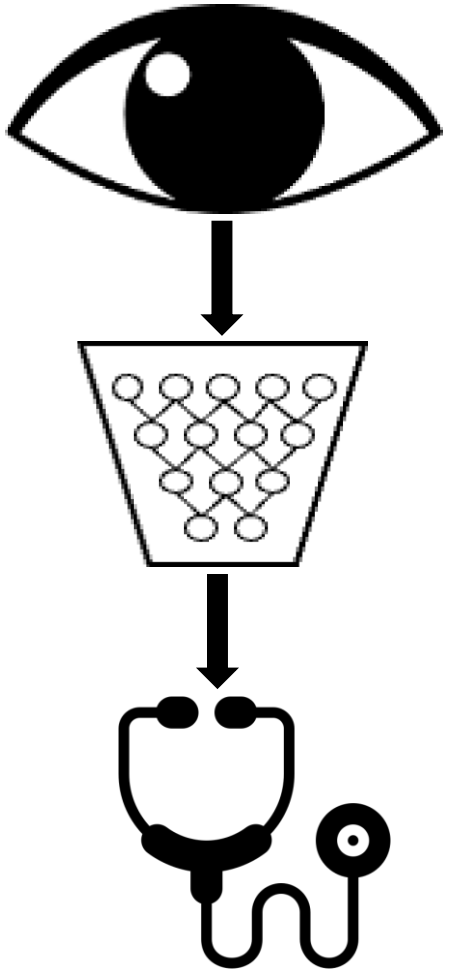


Impact: Self-Driving Cars

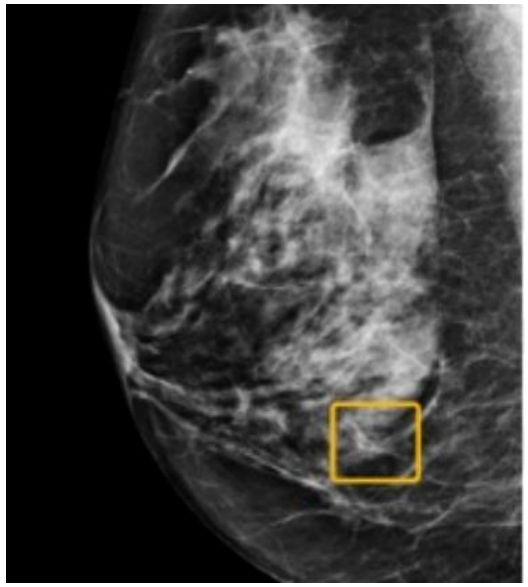


[Courtesy of Alexander Amini]

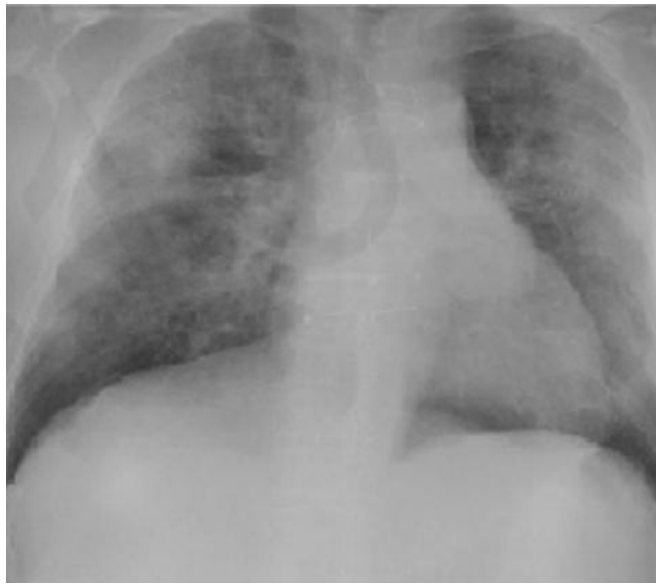
Impact: Medicine, Biology, Healthcare



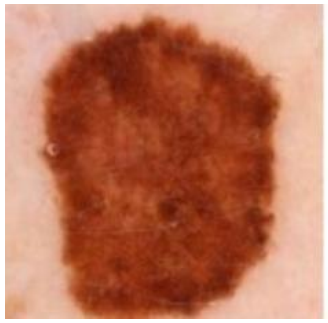
Breast Cancer

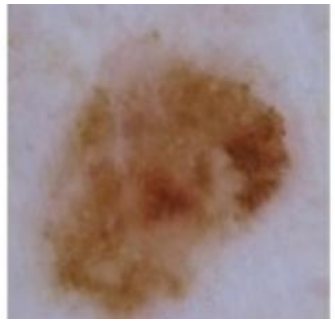


COVID-19

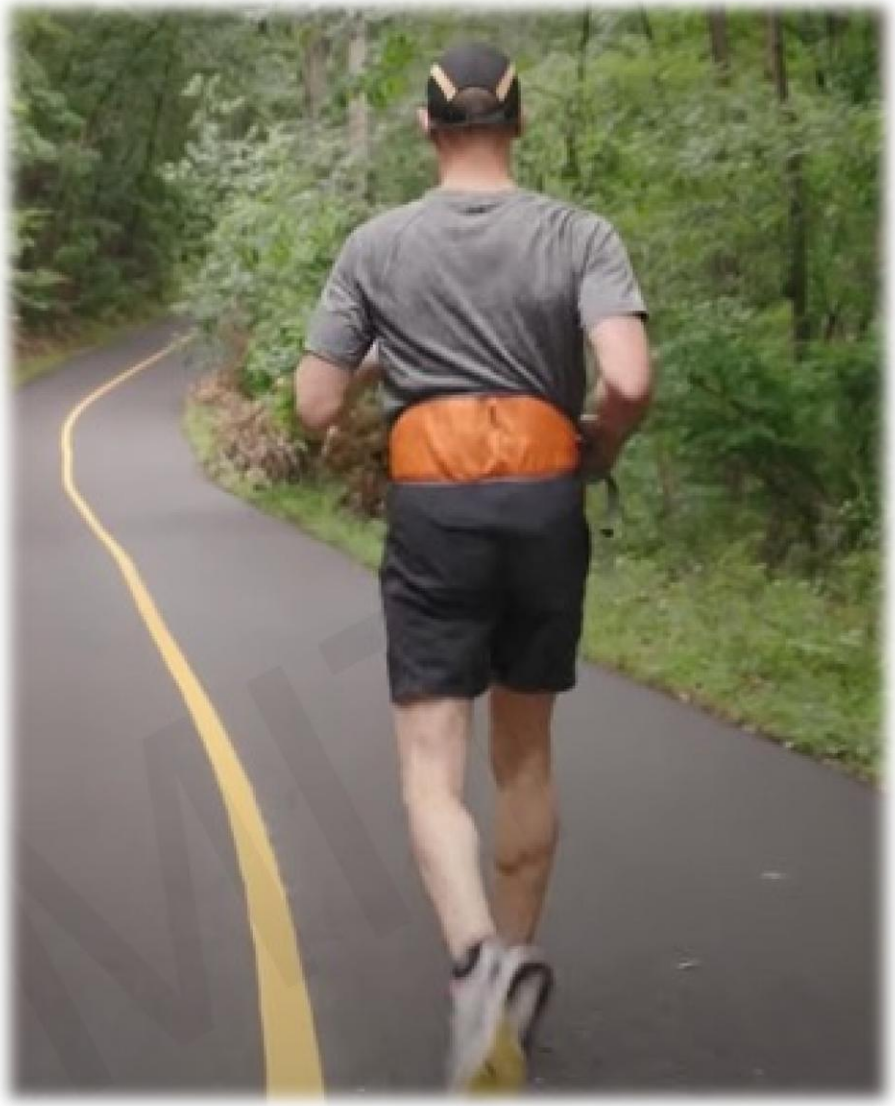


Skin Cancer



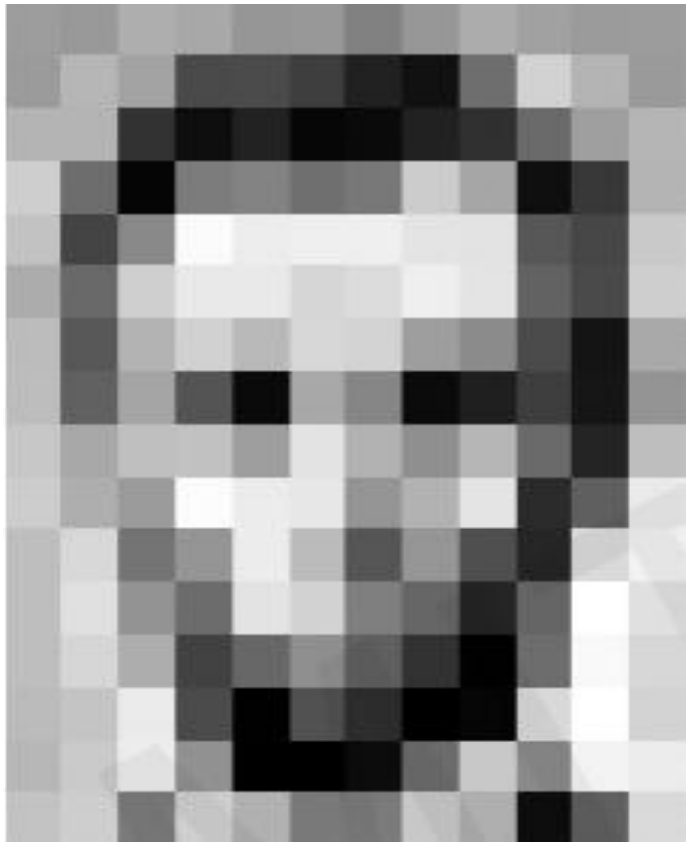


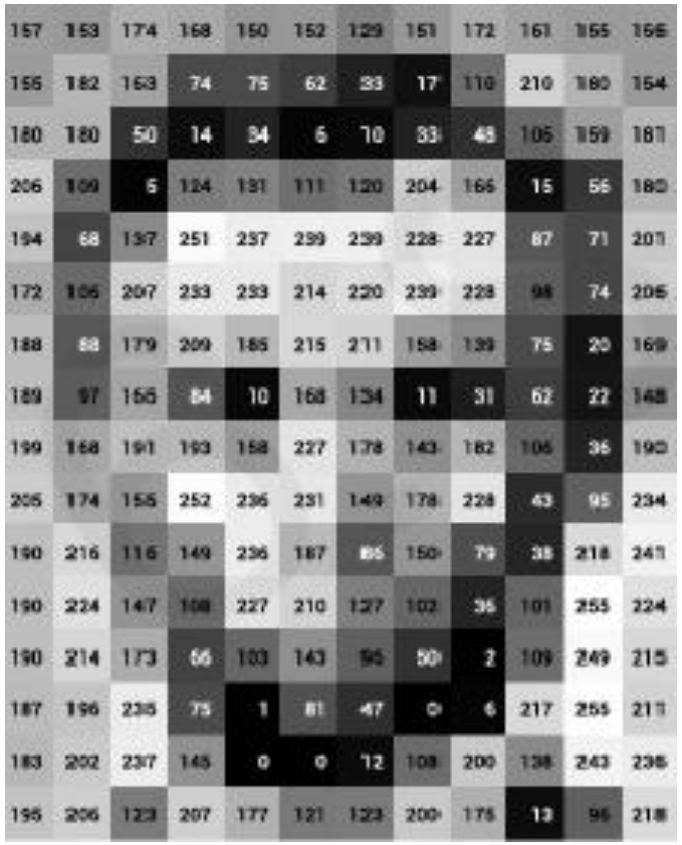
Impact: Accessibility



What Computers “See”

Images are Numbers





What the computer sees

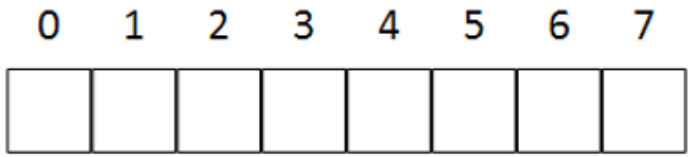
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0, 255]
i.e., 1080x1080x3 for an RGB image

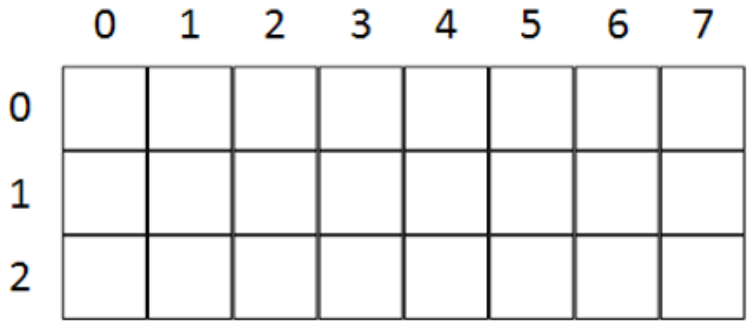
Digital Images Format

Images are stored as **multidimensional arrays**, where pixel values are represented in **2D (grayscale)** or **3D (RGB channels)** matrices.

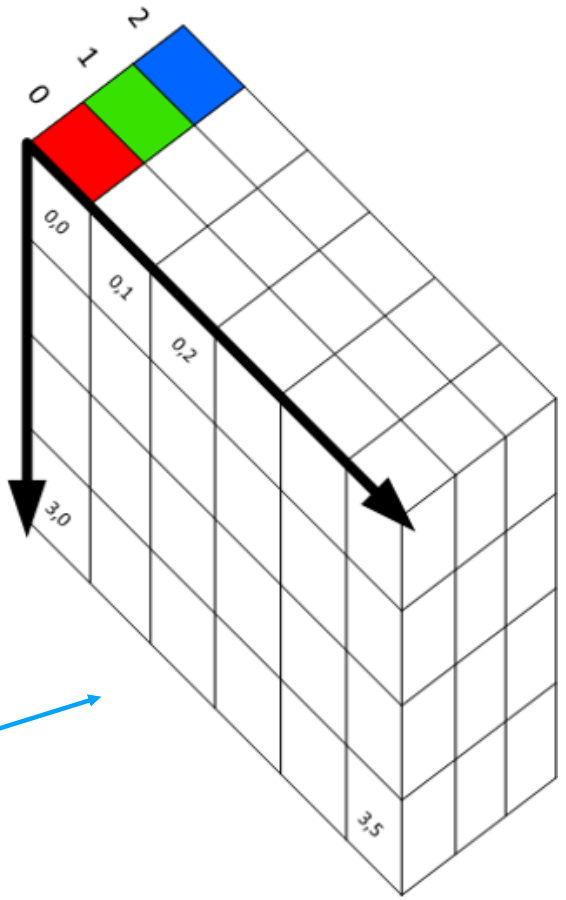
- A 1-Dimensional array looks like this



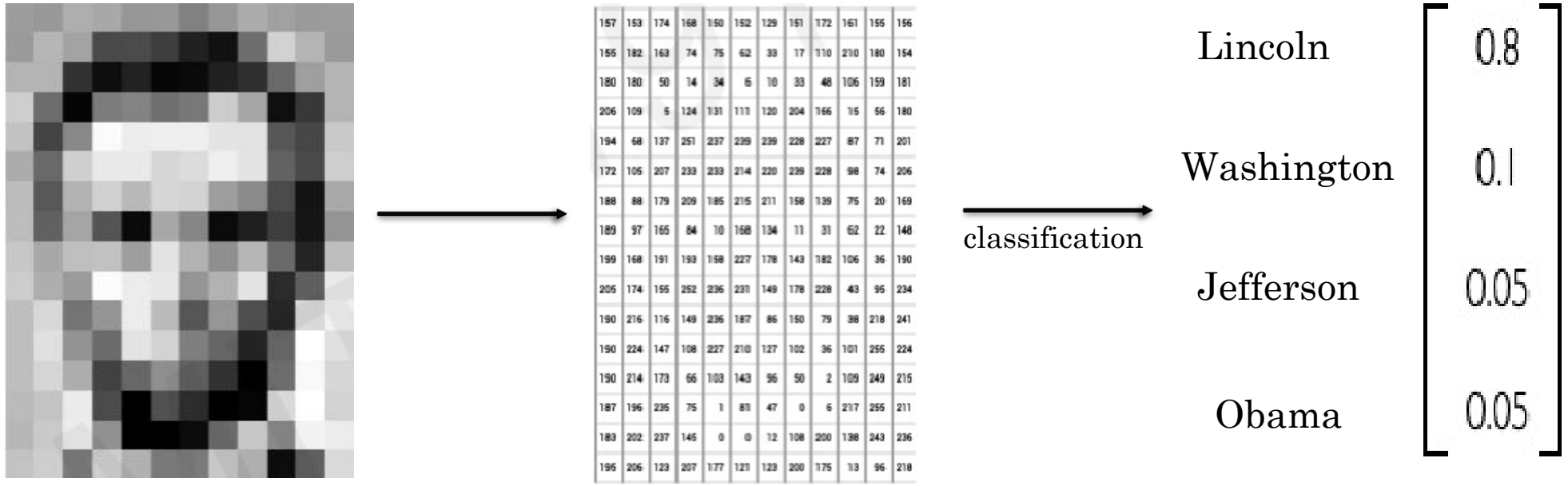
- A 2-Dimensional array looks like this



- A **3-Dimensional** array looks like this



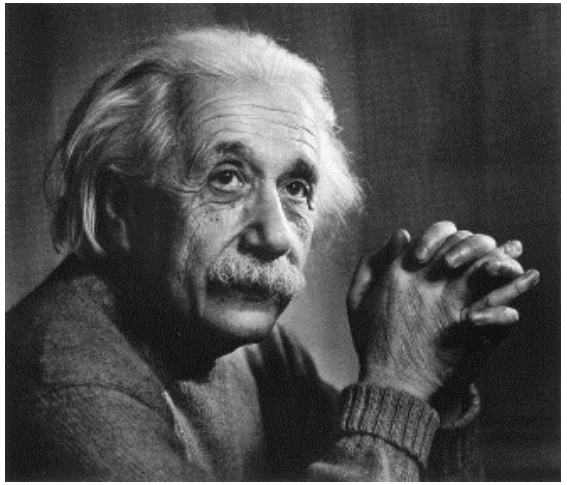
Tasks in Computer Vision



- **Regression:** output variable takes continuous value
- **Classification:** output variable takes class label. Can produce probability of belonging to a particular class

High Level Feature Detection

Let's identify key features in each image category



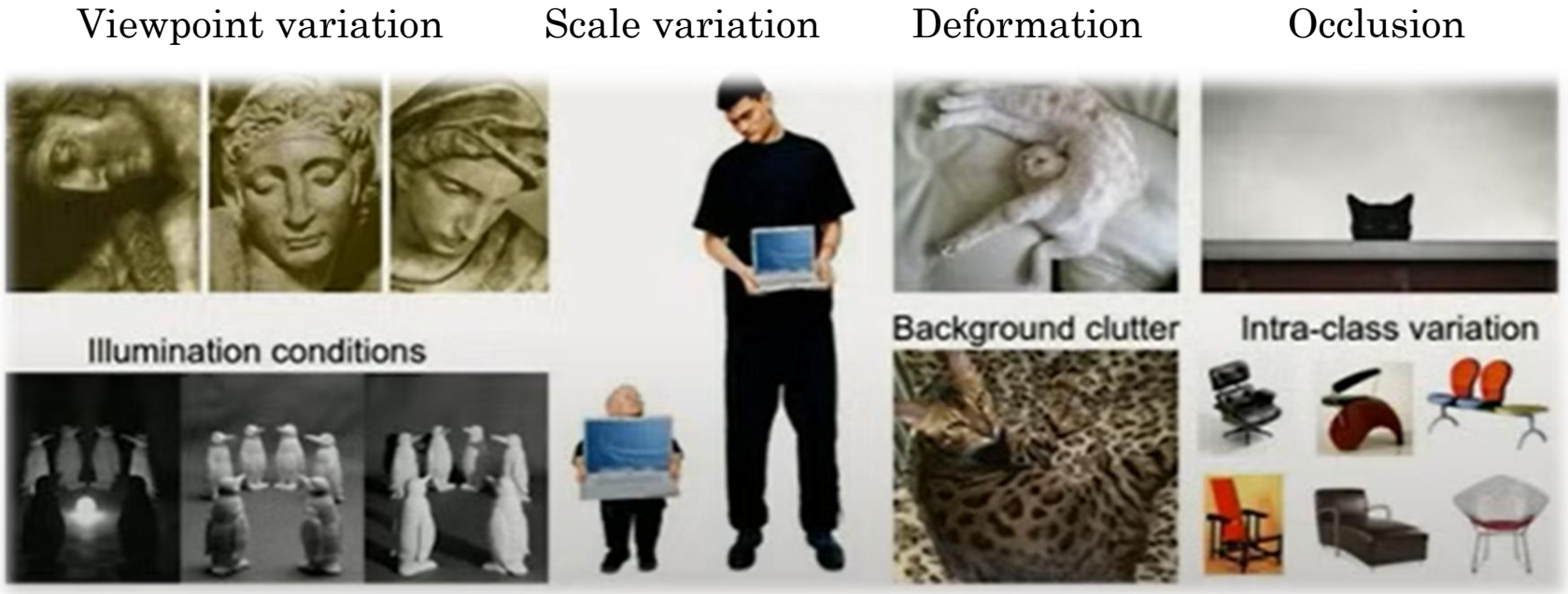
Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

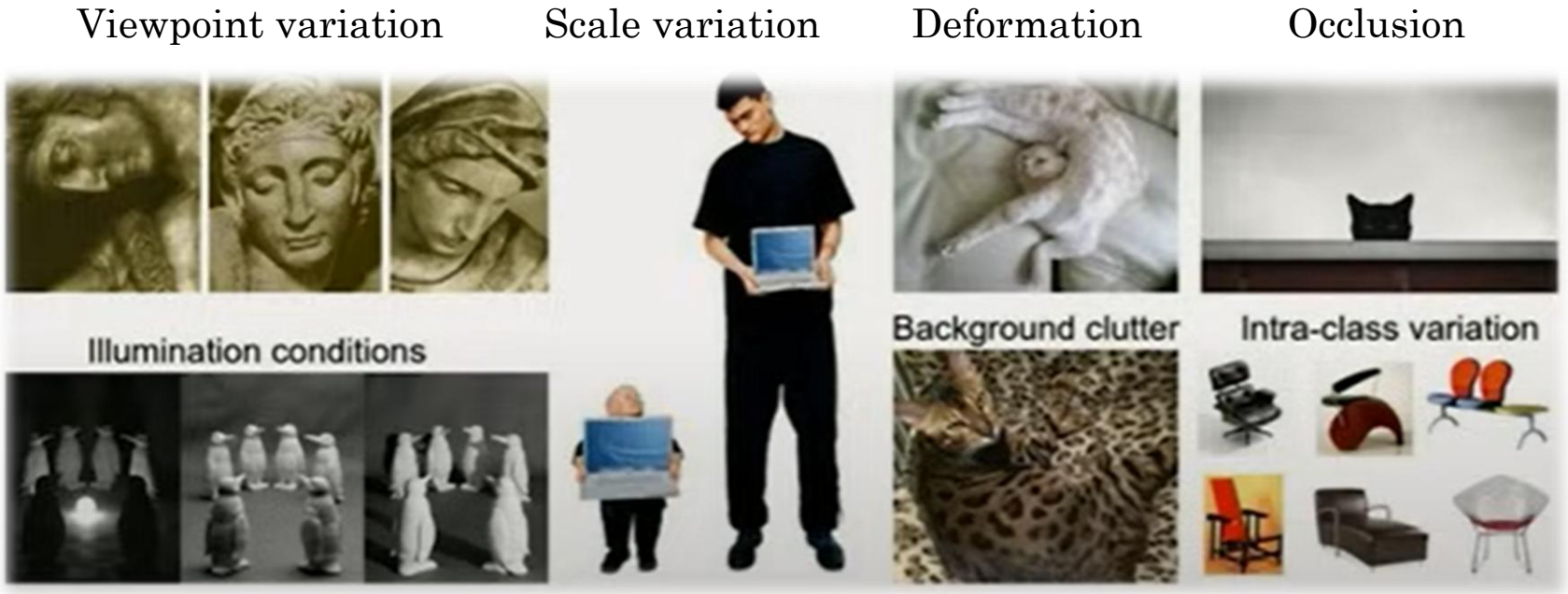
Manual Feature Extraction

Domain Knowledge → Define Features → Detect Features to classify



Manual Feature Extraction

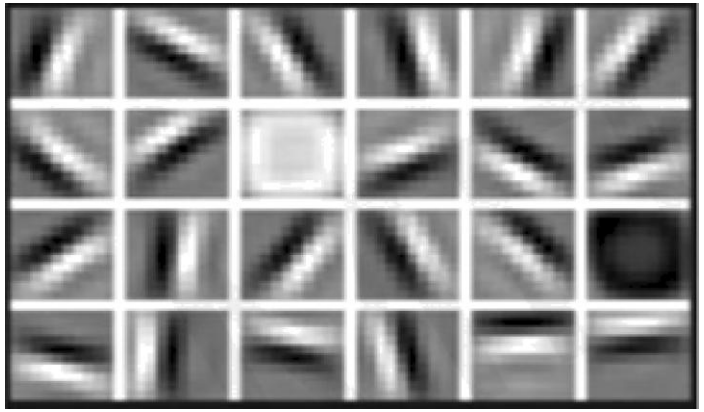
Domain Knowledge → Define Features → Detect Features to classify



Learning Feature Representations

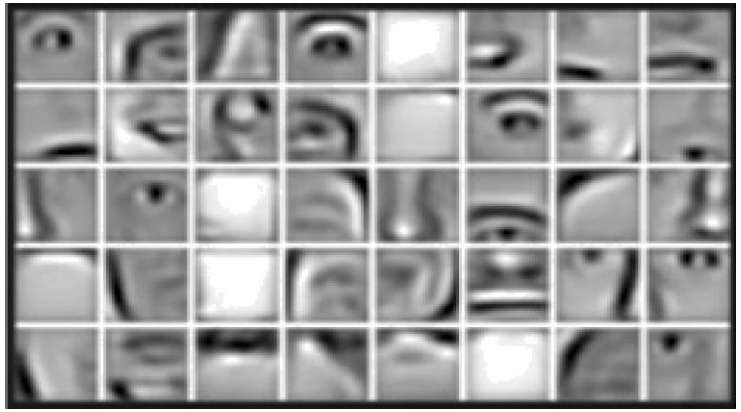
Can we learn the **hierarchy of features** directly from the data instead of hand engineering?

Low Level Features



Lines & Edges

Mid Level Features

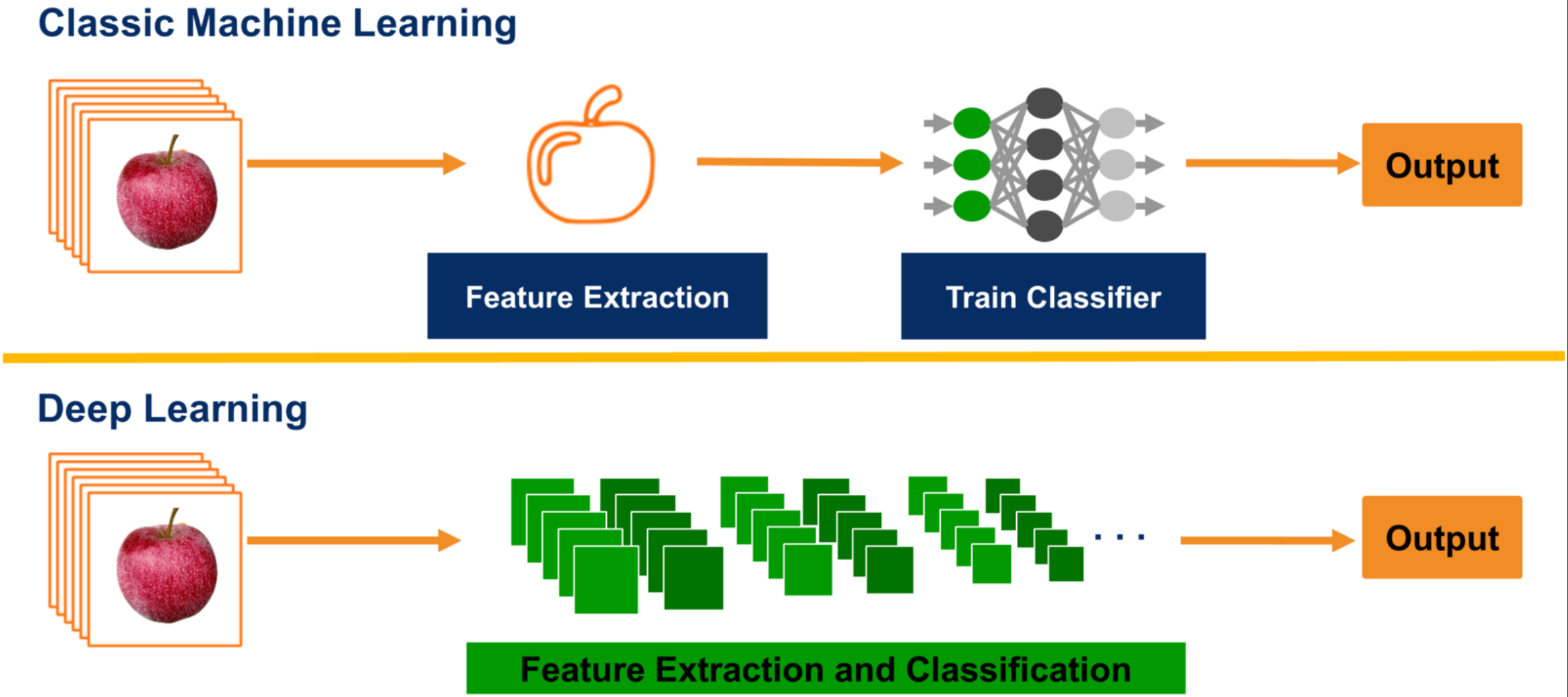


Eyes & Nose & Ears

High Level Features

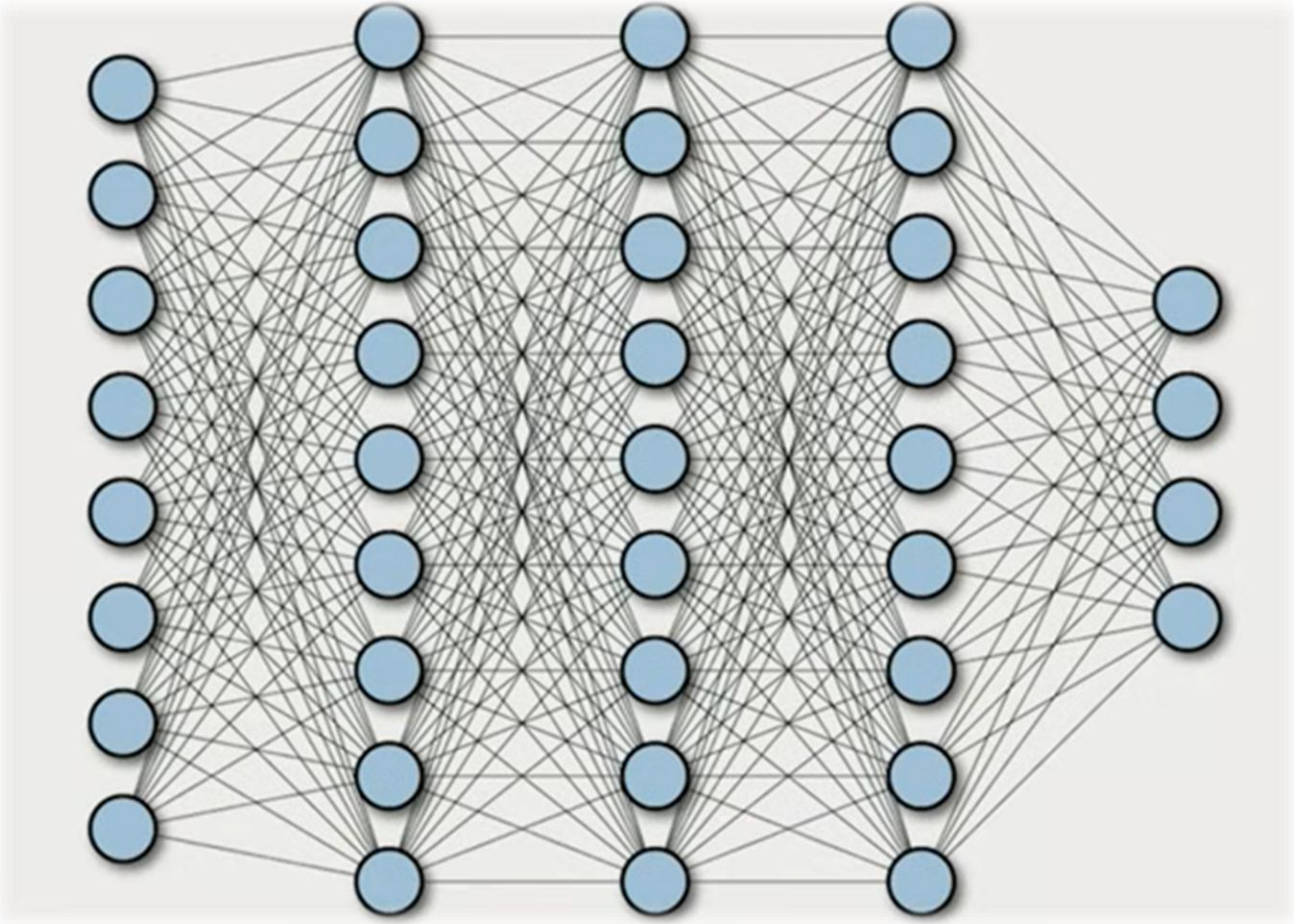
Facial Structure

Classic machine vision vs. deep learning



Learning Visual Features

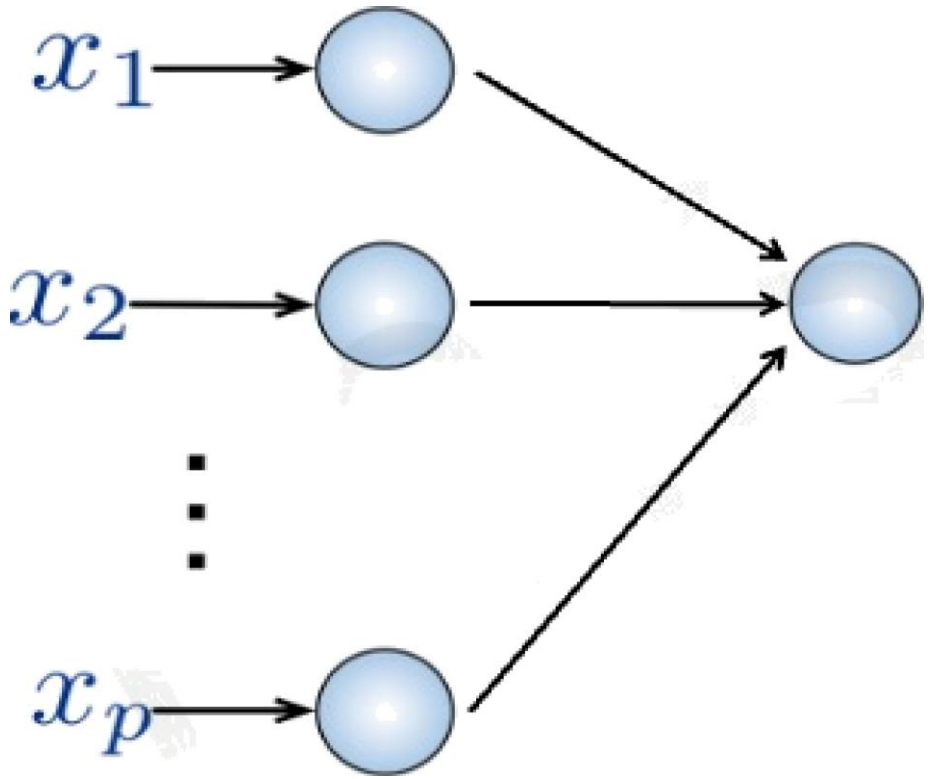
Fully Connected Neural Network



Fully Connected Neural Network

Input

- 2D image
- Vector of pixel values



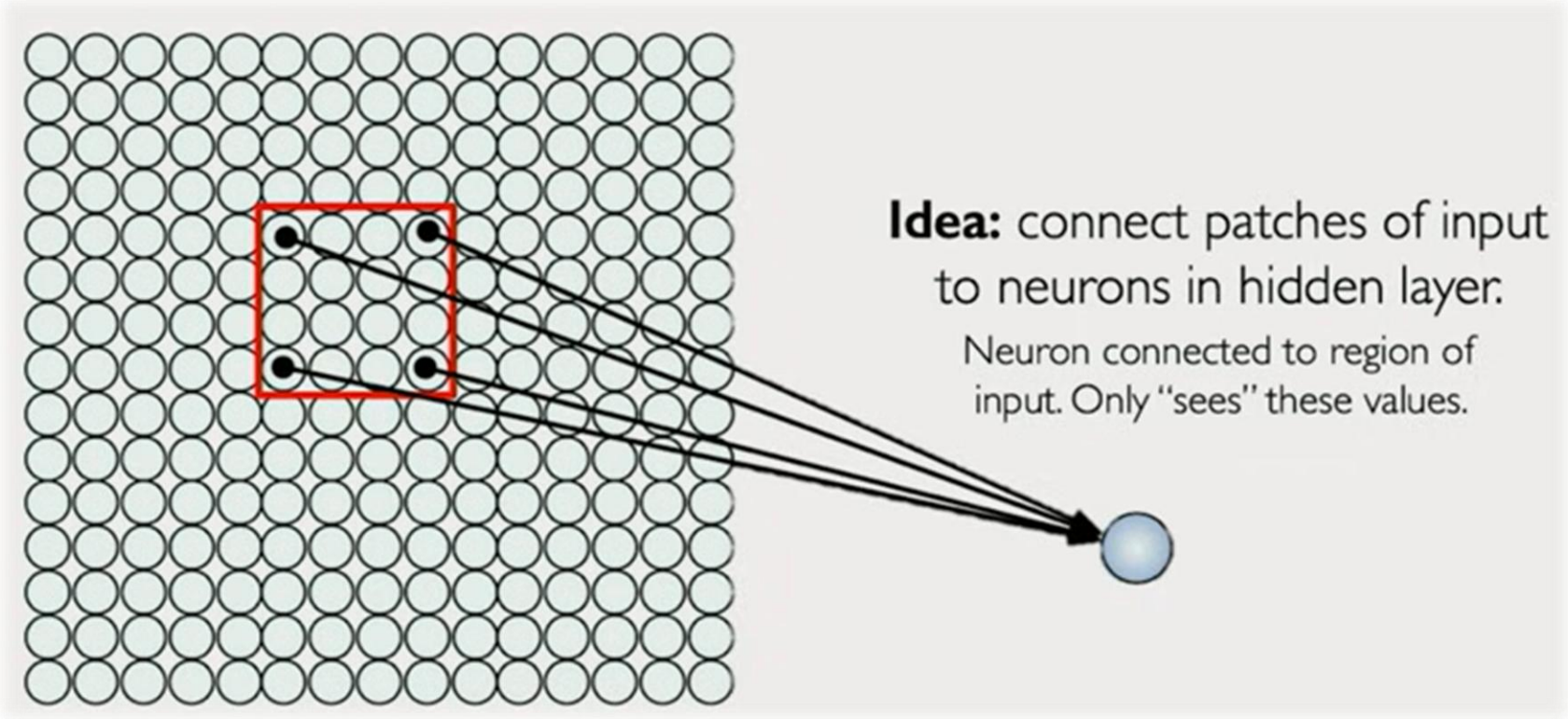
Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

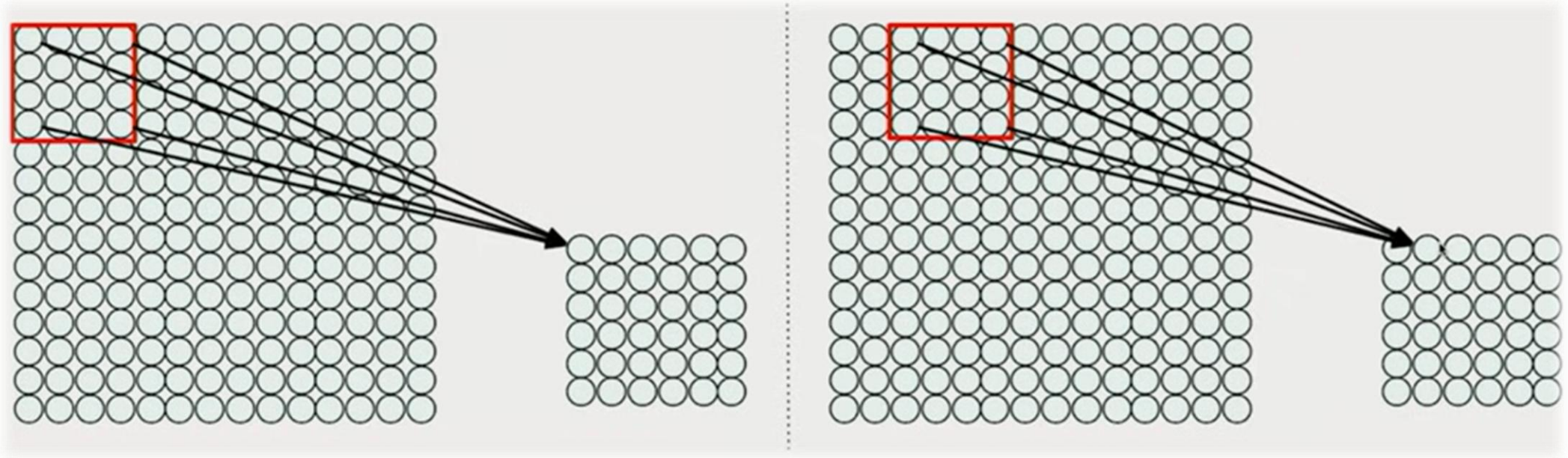
How can we use **spatial structure** in the input to inform the architecture of the network?

Using Spatial Structure

Input: 2D image.
Array of pixel values

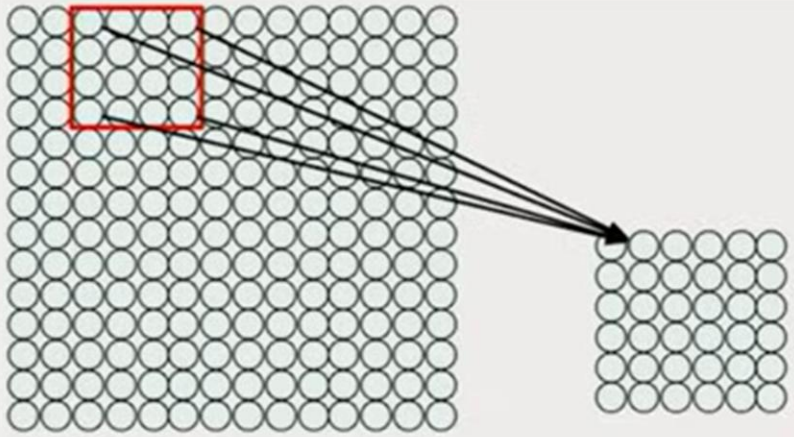


Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** the patch to detect particular features?

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

- **Convolutional Neural Networks**, commonly referred to as **CNNs** are a specialized type of neural network designed to process and classify images.

- There are three main operations to CNNs.
  1. **Convolution:** Apply filters to generate feature maps.
  2. **Non-linearity:** Often ReLU
  3. **Pooling:** Down sampling operation on each feature map.

torch.nn.Conv2d

torch.nn.ReLU, …

torch.nn.MaxPool2d


torch.nn.Conv2d(in_channels=3, out_channels=d, kernel_size=(h,w), stride=s)


tf.keras.layers.Conv2D(filters=d, kernel_size=(h,w), strides=s)

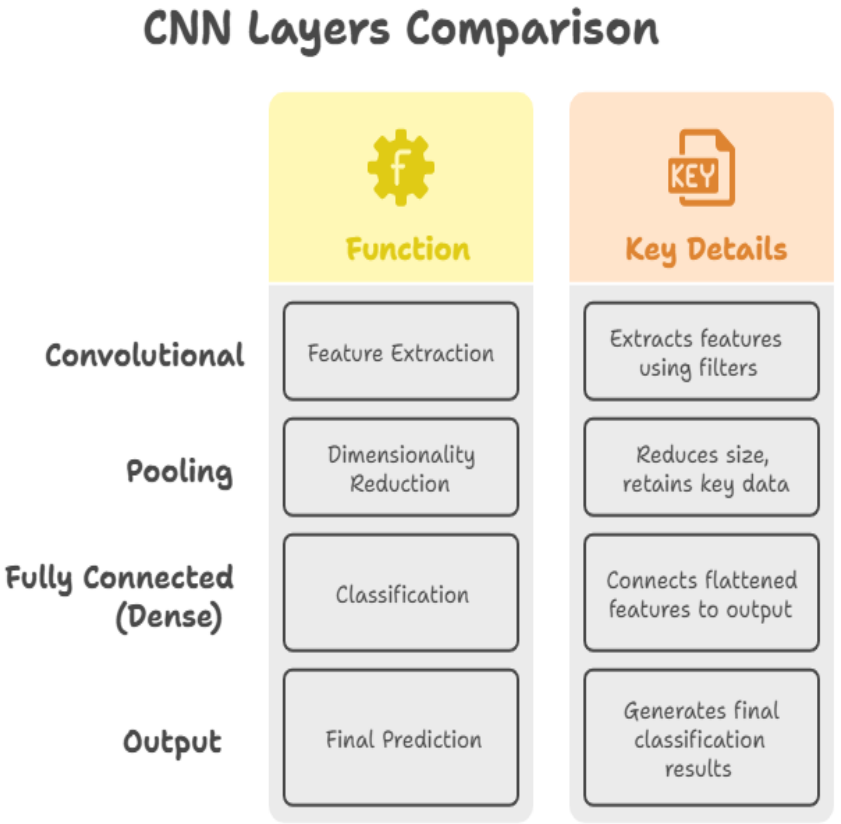
Need to specify input dimensionality!

Convolutional Neural Networks (CNNs)

Types of Layers

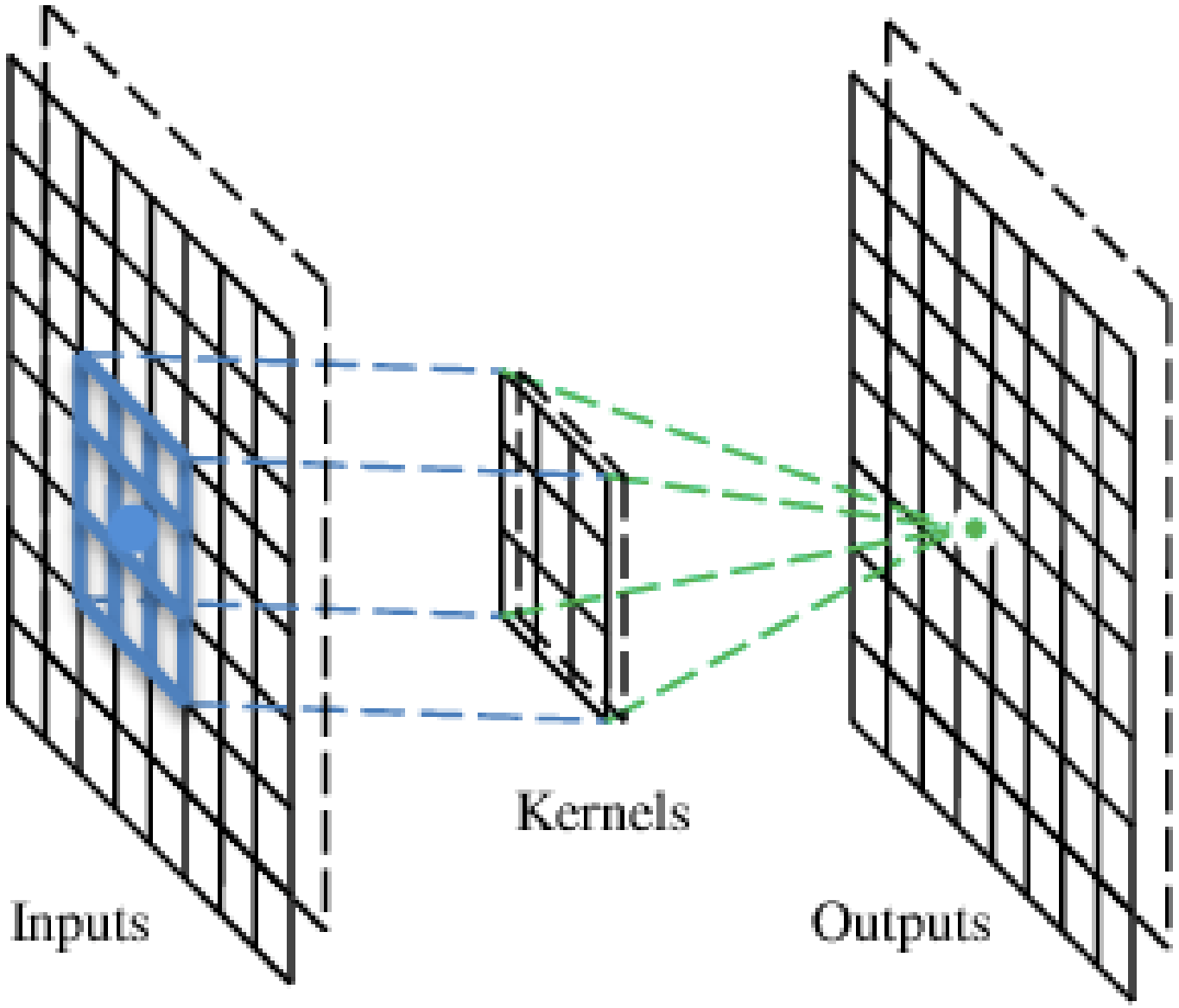
CNNs consist of specialized layers that efficiently process spatial data like images.

1) **Convolutional Layer**
   - Extracts features (edges, textures, patterns) using filters.
   - Preserves spatial structure.
   - Activation functions (e.g., ReLU) are applied after this layer.
   - Helps learn complex patterns.
2) **Pooling Layer**
   - Reduces dimensionality while retaining key features.
   - **Common types**: Max Pooling, Average Pooling.
3) **Fully Connected (Dense) Layer**
   - Flattens feature maps and connects to output neurons.
   - Used for classification tasks.
4) **Output Layer**
   - Produces final predictions (e.g., Softmax for multi-class classification).



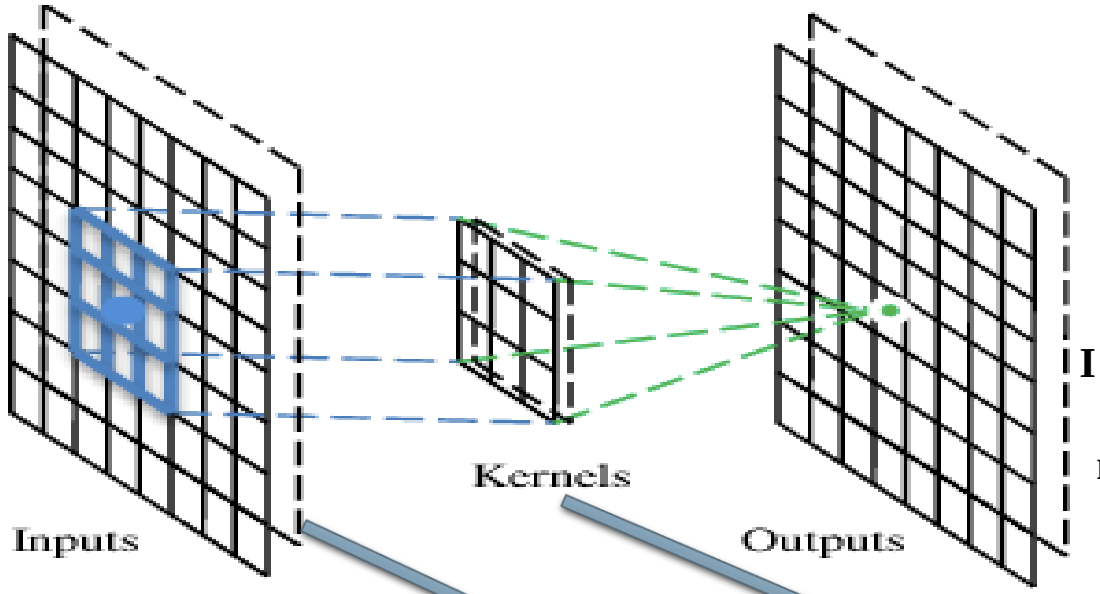
A convolutional layer

- A CNN is a neural network with some convolutional layers
- (and some other layers). A convolutional layer has a number
- of filters that does convolutional operation.



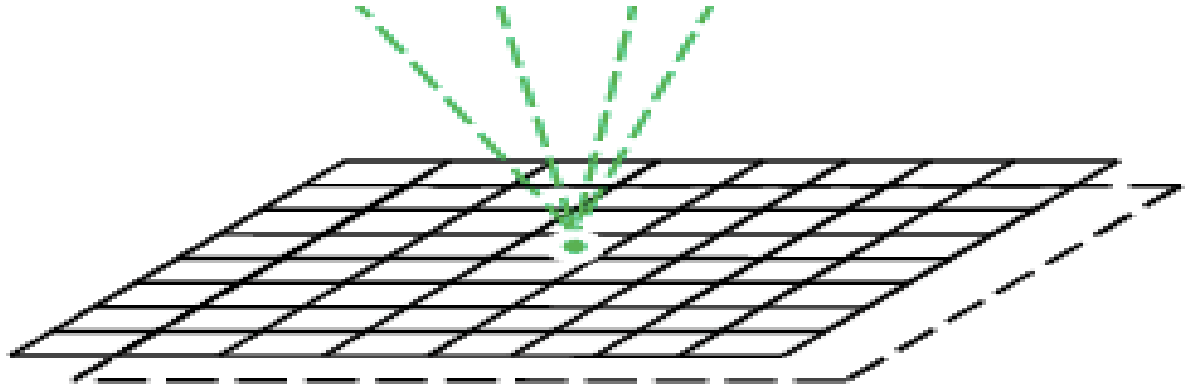
A convolutional layer

- In CNNs, the convolution operation is the core building block
- computing dot products between the filter and local regions of image
- filter's weights extracts features such as edges, textures or patterns.

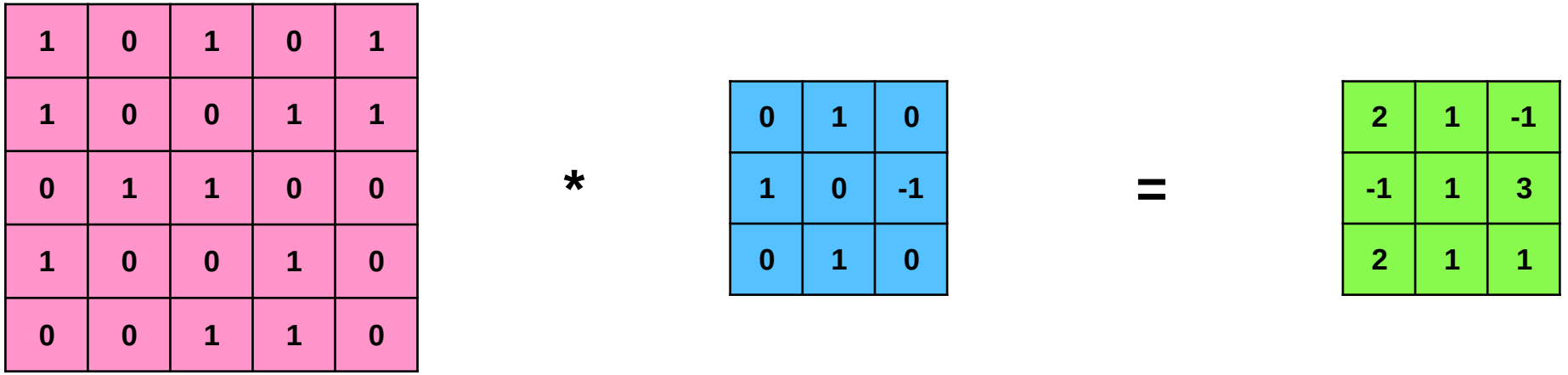


I is input Image
K kernel
m, n are filter indices

$$(I * K)(i,j) = \sum_m \sum_n I(i-m, j-n).K(m,n)$$



Example of a Convolution Operation



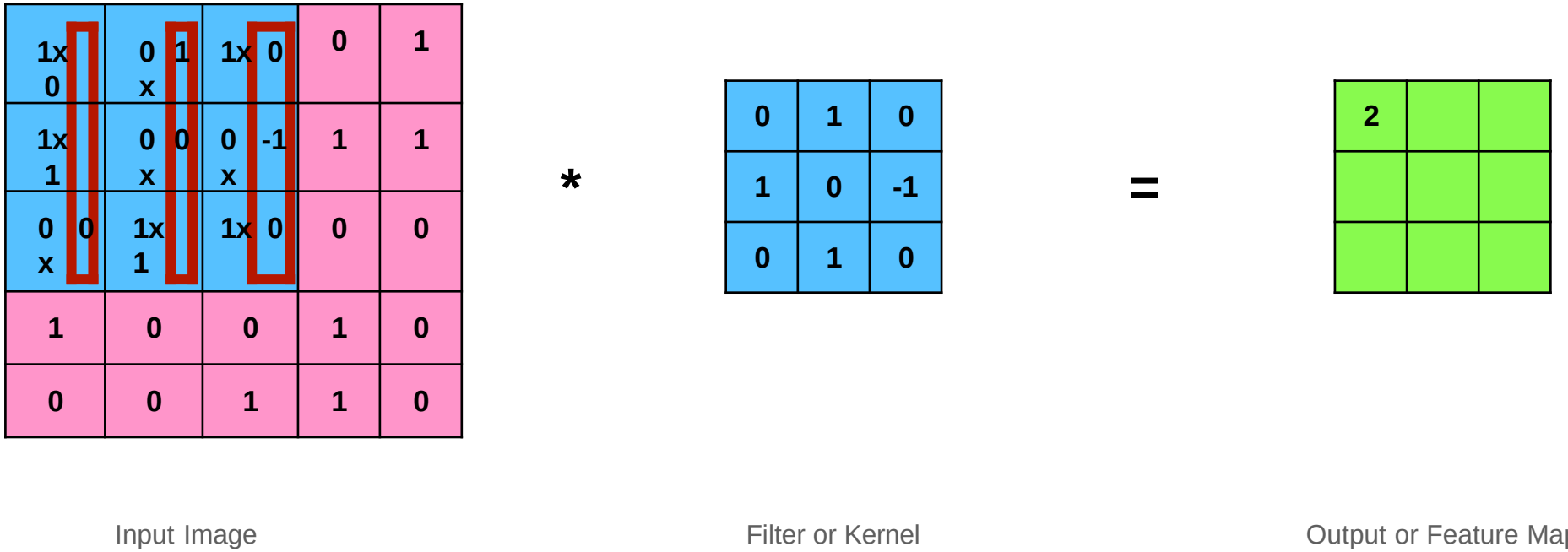
Input Image

Filter or Kernel

Output or Feature Map

Example of a Convolution Operation

$(1 \times 0) + (0 \times 1) + (1 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times -1) + (0 \times 0) + (1 \times 1) + (1 \times 0) = 2$



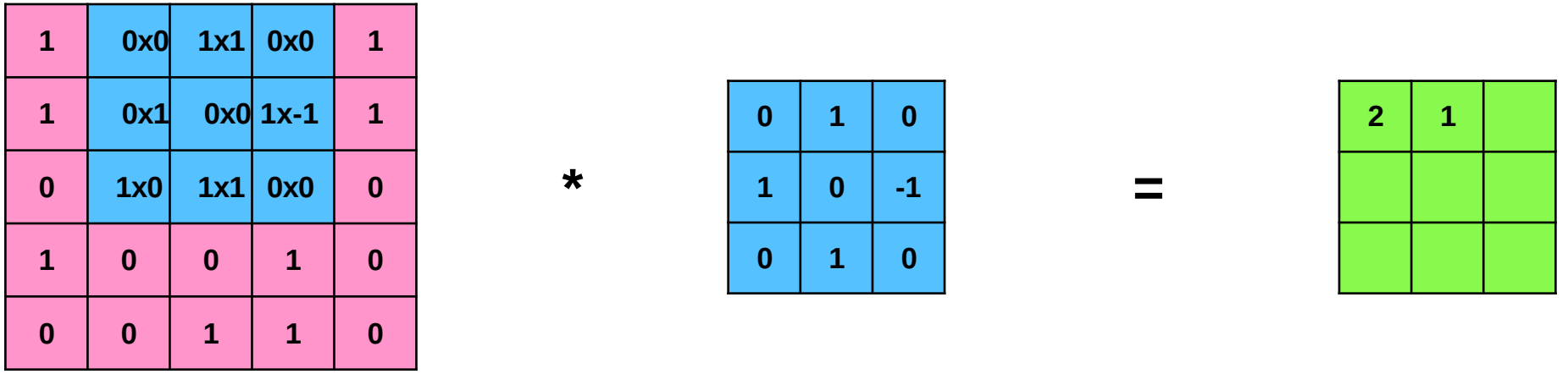
Input Image

Filter or Kernel

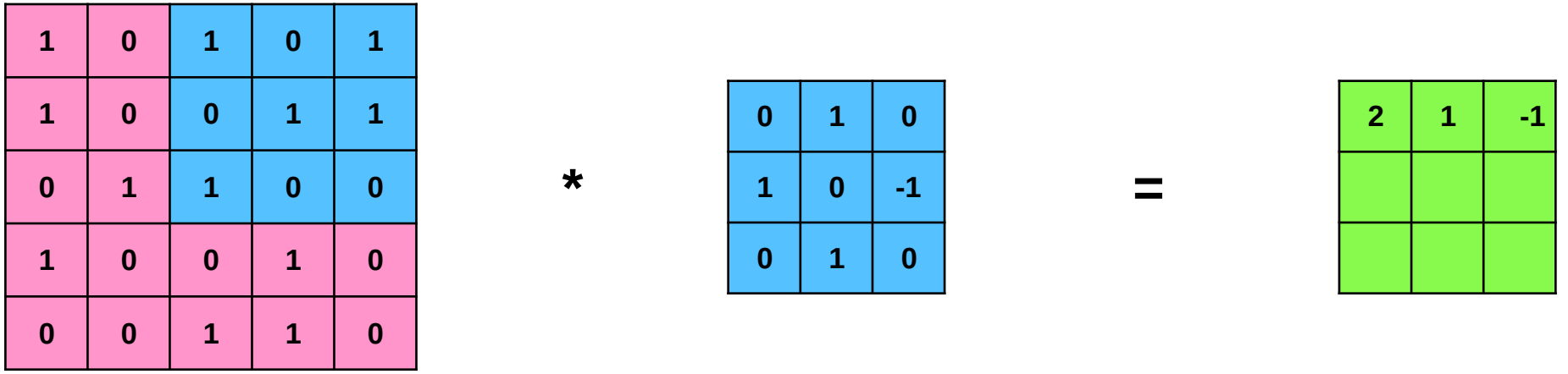
Output or Feature Map

Example of a Convolution Operation

$(0 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times -1) + (1 \times 0) + (1 \times 1) + (0 \times 0) = 1$



Example of a Convolution Operation

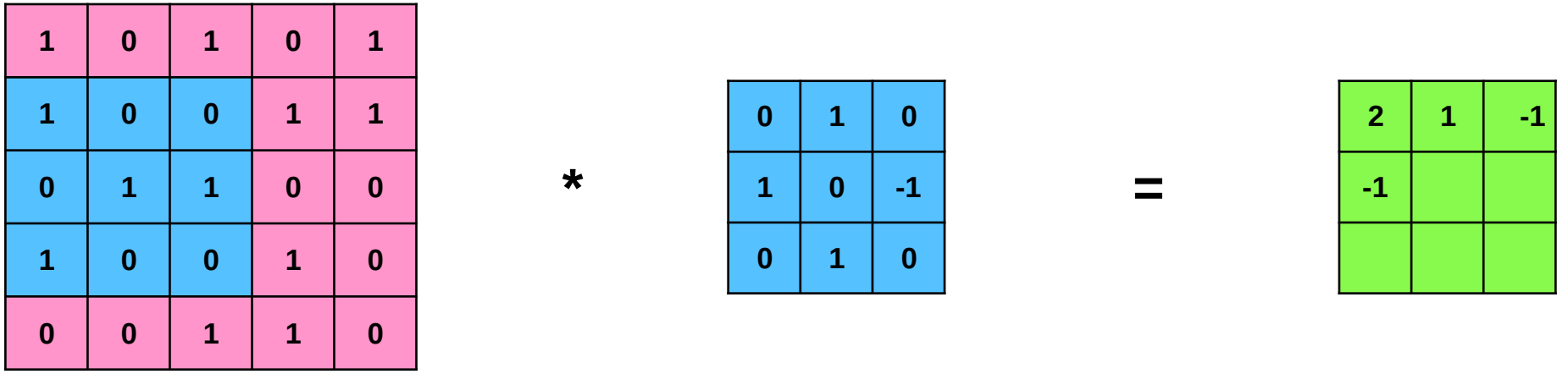


Input Image

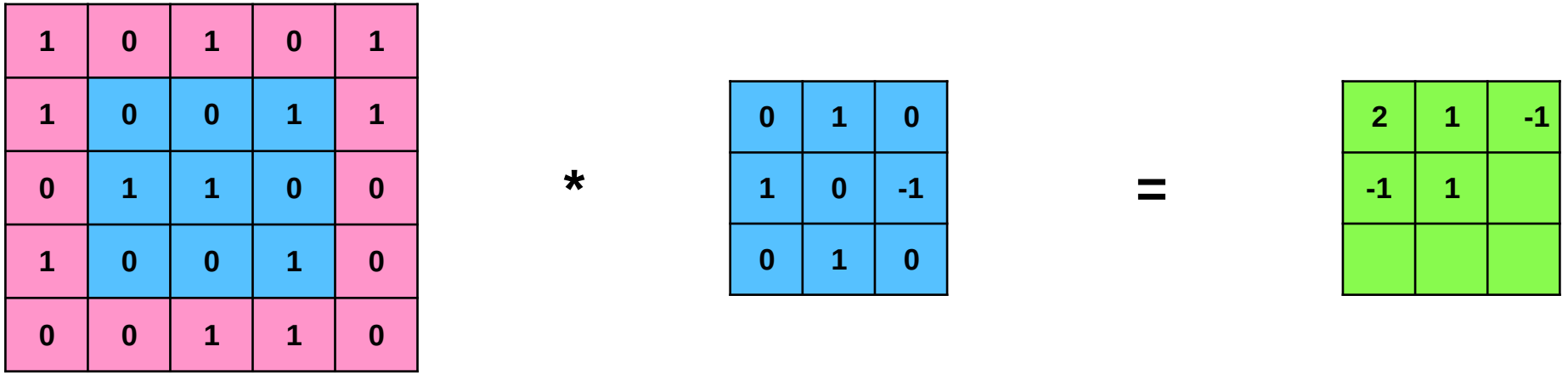
Filter or Kernel

Output or Feature Map

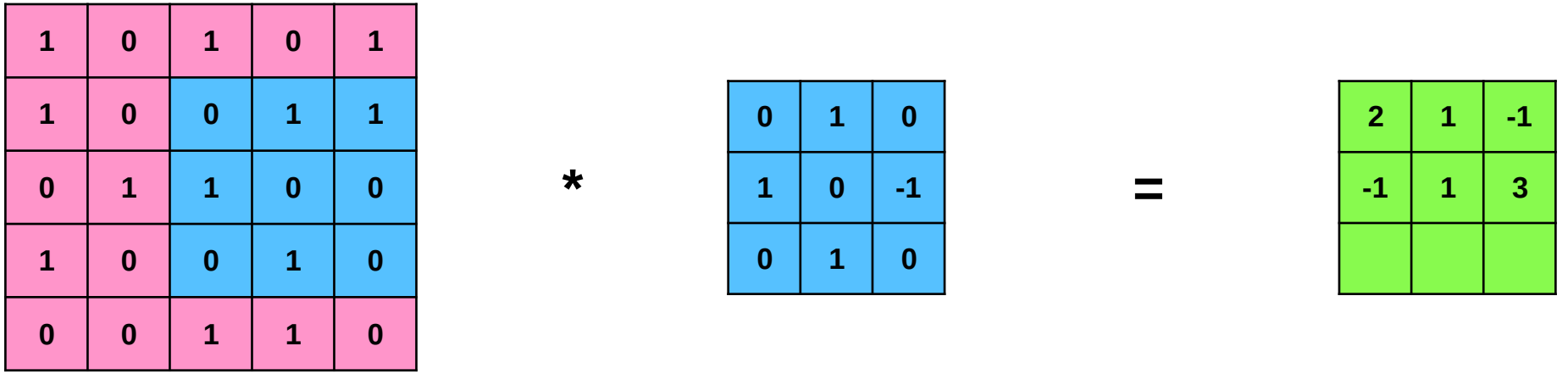
Example of a Convolution Operation



Example of a Convolution Operation



Example of a Convolution Operation

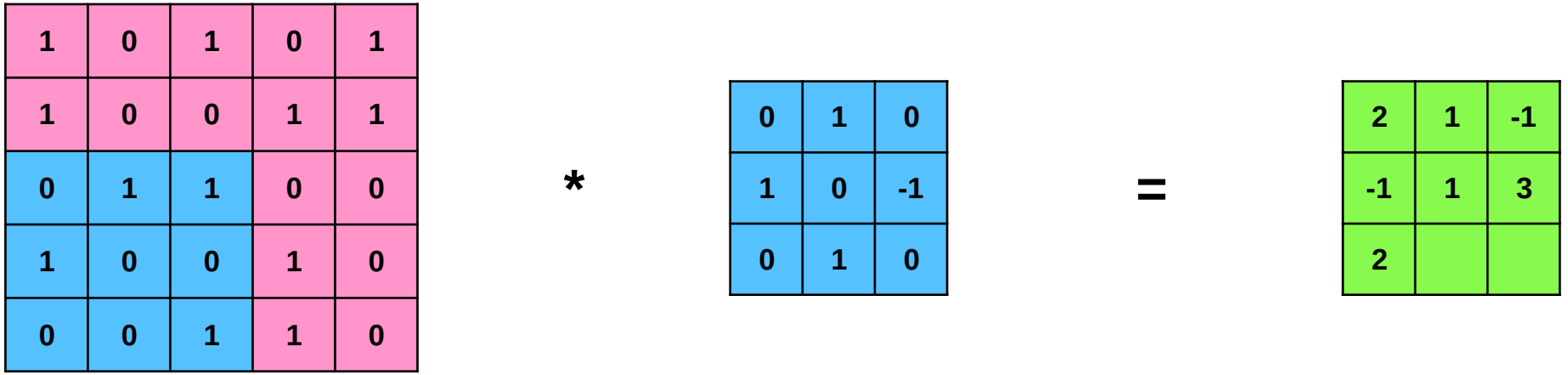


Input Image

Filter or Kernel

Output or Feature Map

Example of a Convolution Operation

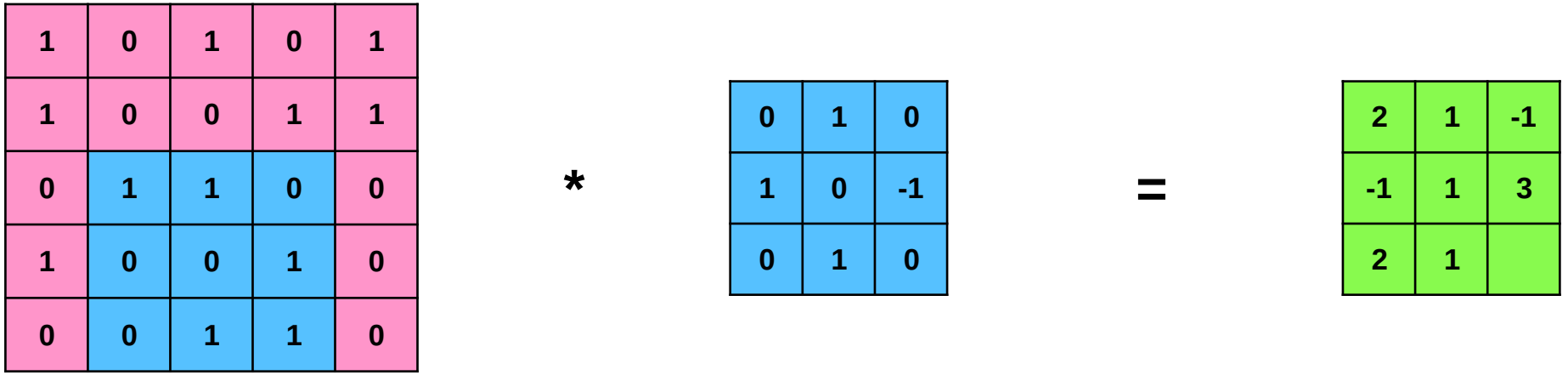


Input Image

Filter or Kernel

Output or Feature Map

Example of a Convolution Operation

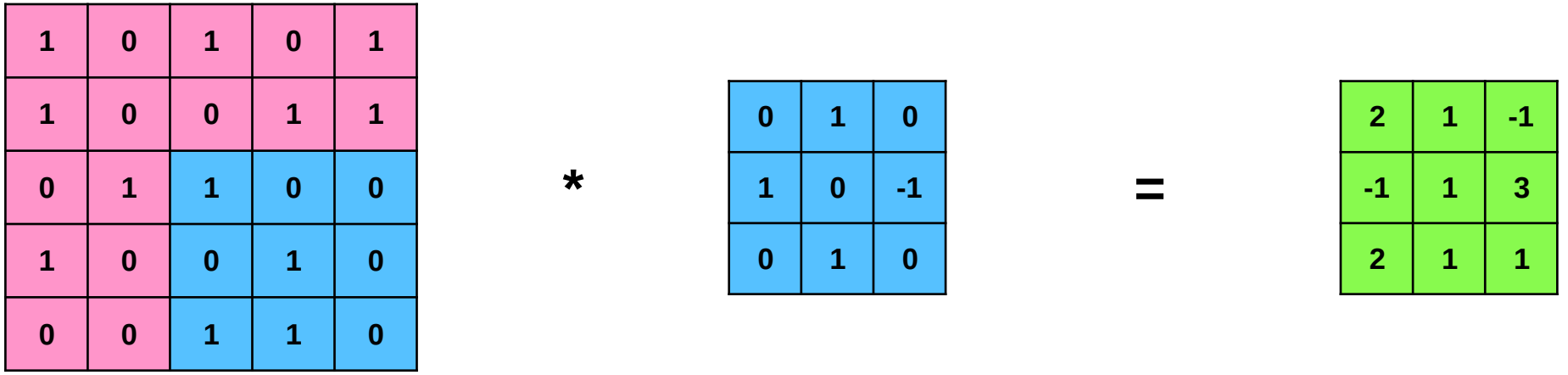


Input Image

Filter or Kernel

Output or Feature Map

Example of a Convolution Operation



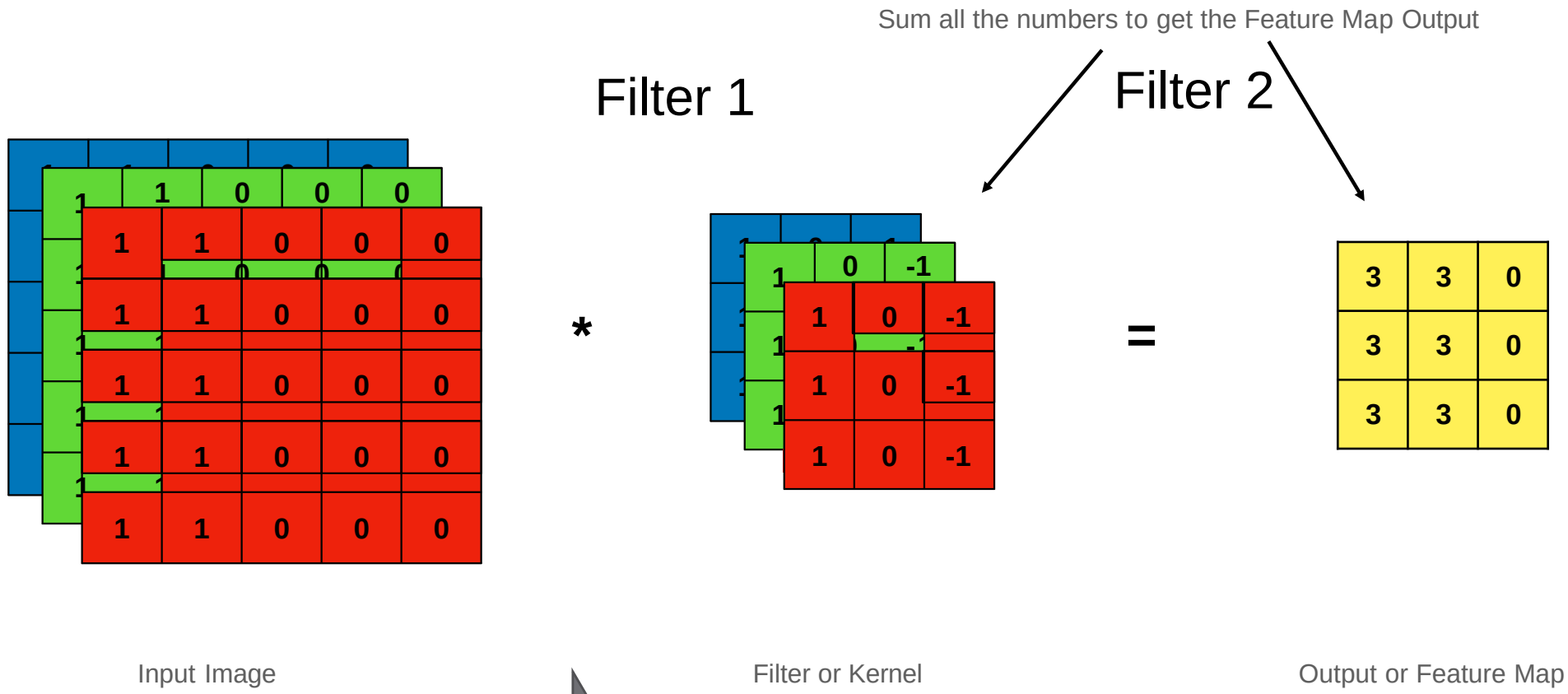
Input Image

Filter or Kernel

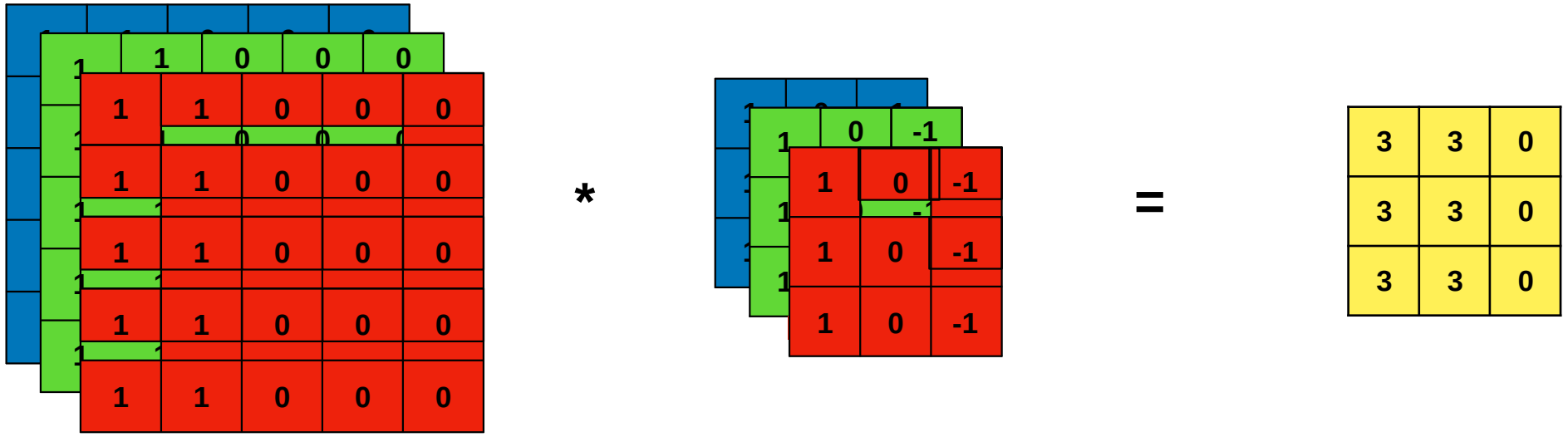
Output or Feature Map

Convolution Operations on Color Images

Color image: RGB 3 channels



Advantages of Having a Filter For Each Colour



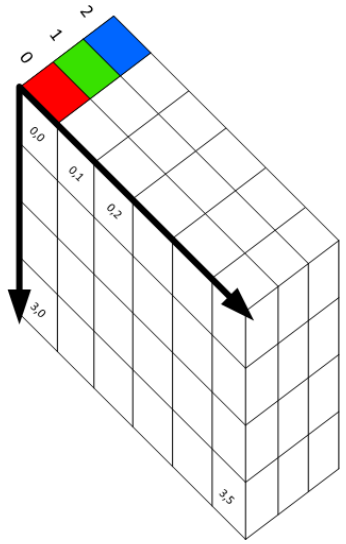
Input Image

Filter or Kernel

Output or Feature Map

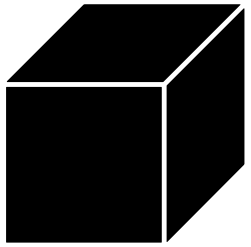
- We can detect features that are specific to a colour

Considered 3D Volumes



Input Image
5 x 5 x **3**

*



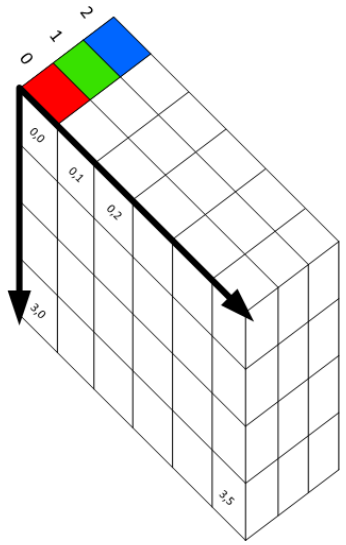
Filter or Kernel
3 x 3 x **3**

=

3	3	0
3	3	0
3	3	0

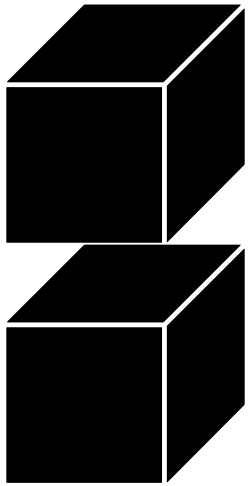
Output or Feature Map
3 x 3

How Multiple Filters Affect Our Output



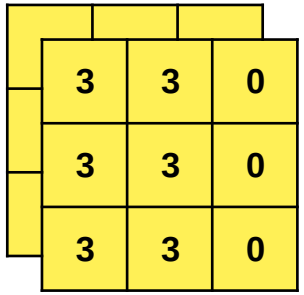
Input Image
5 x 5 x **3**

*



2 Filters or Kernels
3 x 3 x **3** x **2**

=

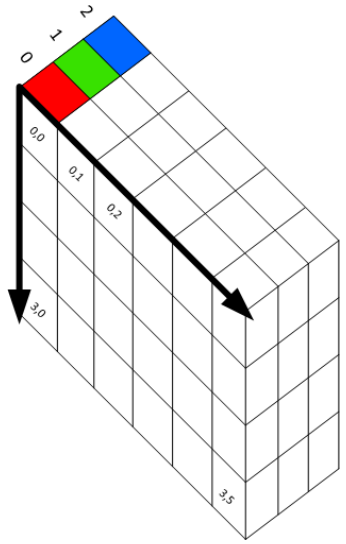


Output or Feature Map
3 x 3 x **2**

Calculating Output Size for 3D Conv Volumes

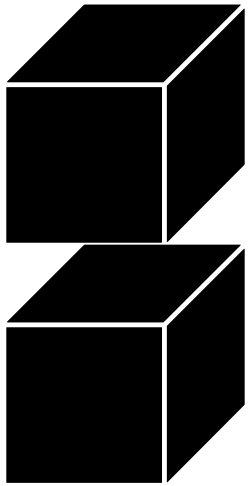
$$(n \times n \times n_c) * (f \times f \times n_c) = (n - f + 1) \times (n - f + 1) \times n_f$$

$$(5 \times 5 \times 3) * (3 \times 3 \times 3) = 3 \times 3 \times 2$$



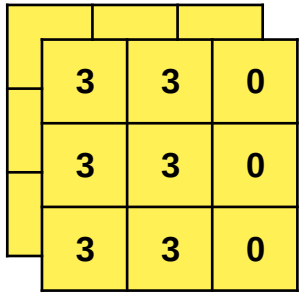
Input Image
5 x 5 x **3**

*



2 Filters or Kernels
3 x 3 x **3** x **2**

=

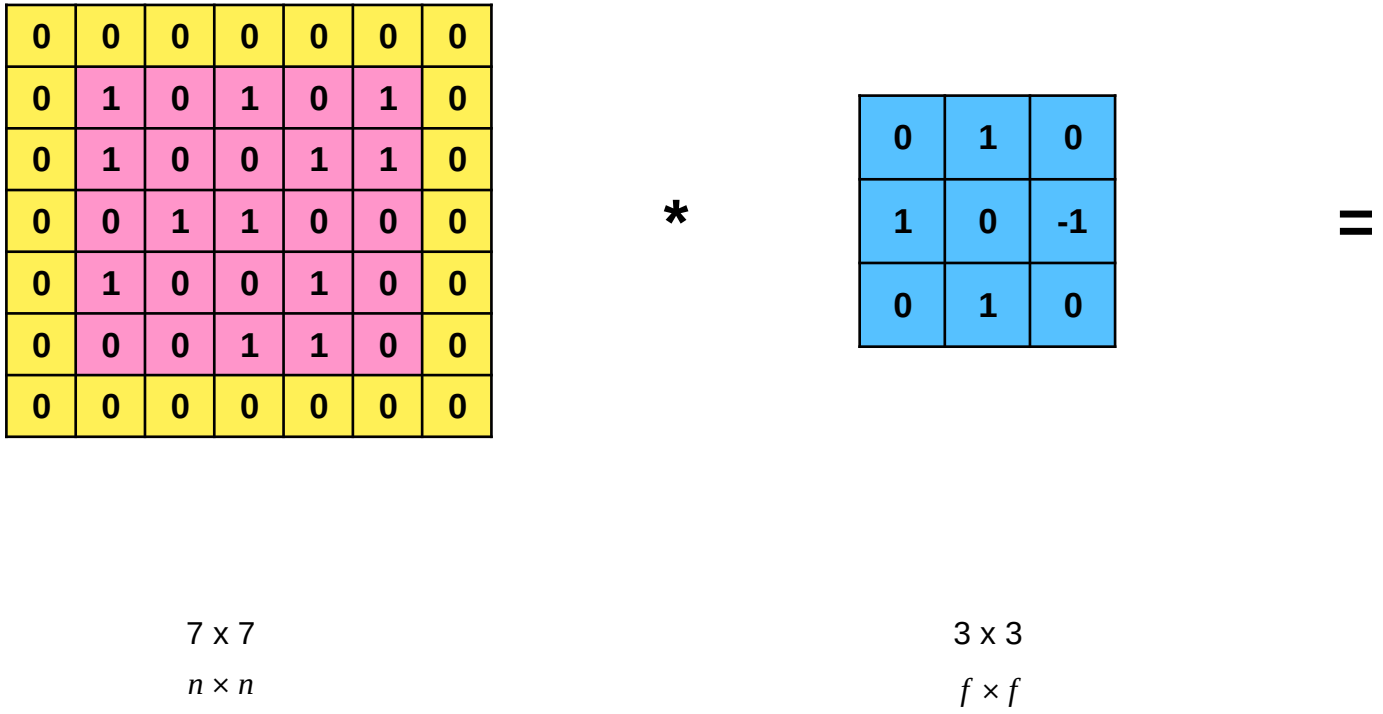


Output or Feature Map
3 x 3 x **2**

Consecutive Conv layers would keep shrinking the output

Can we preserve our image size?

We've added a 1 pixel pad of zeros (zero padding) around our input



7 x 7

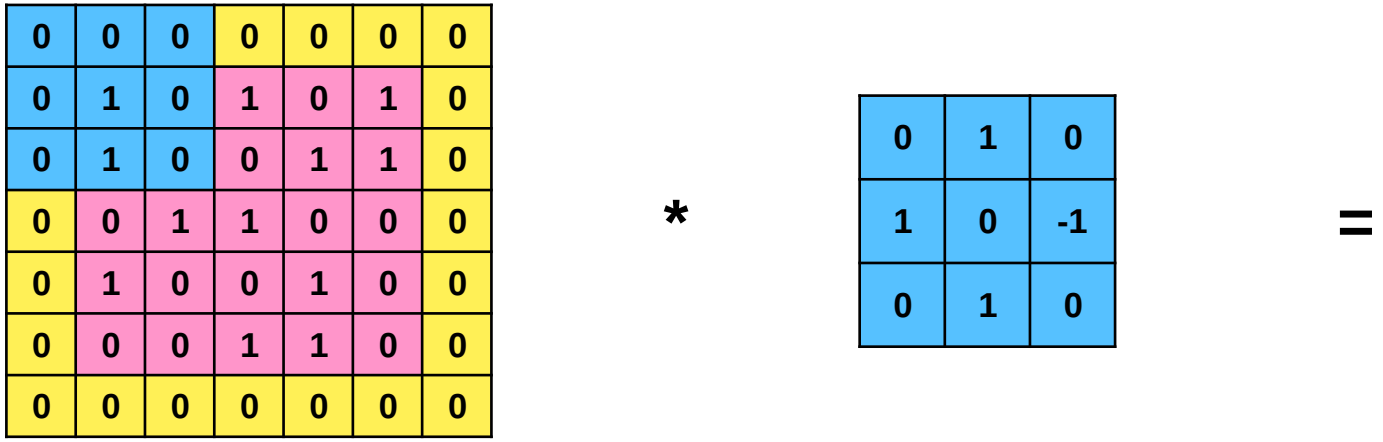
$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding



7 x 7

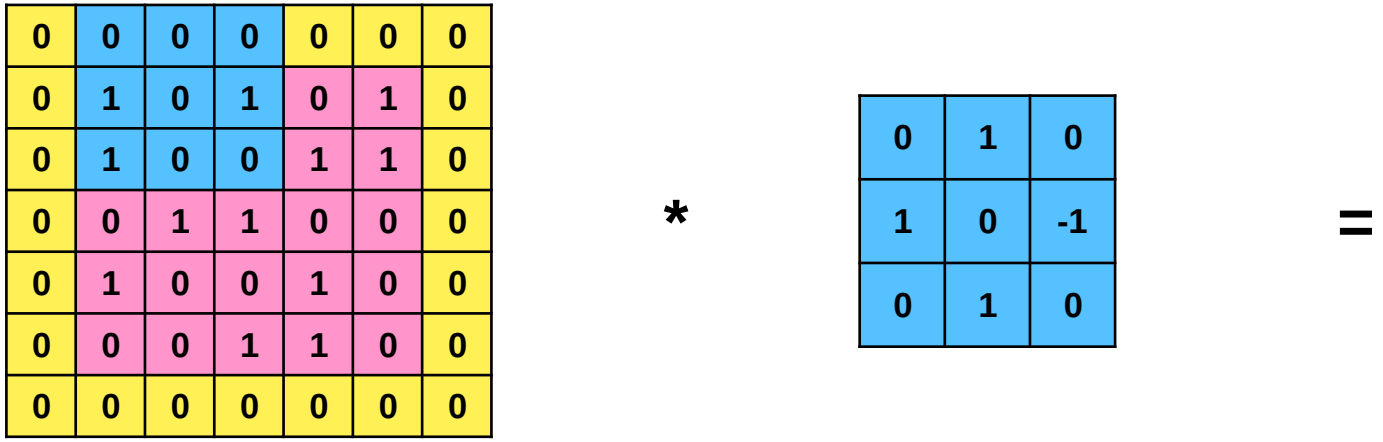
$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding



7 x 7

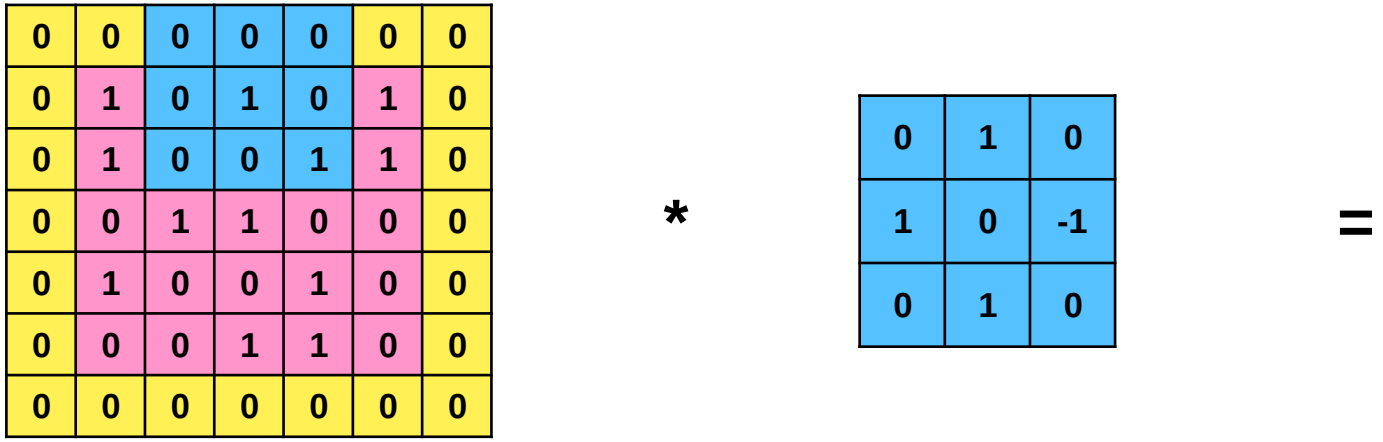
$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding



7 x 7

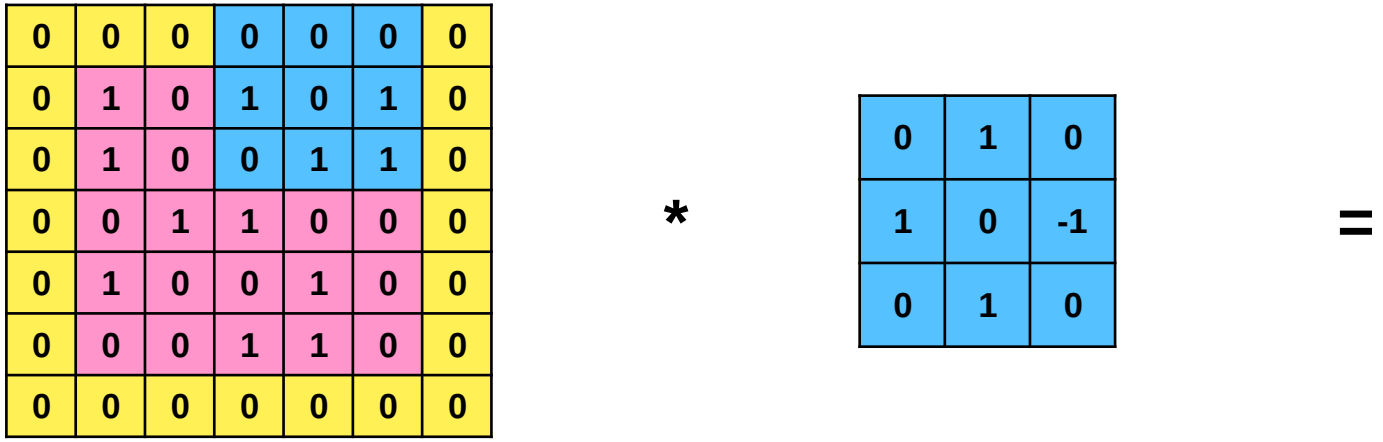
$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding



7 x 7

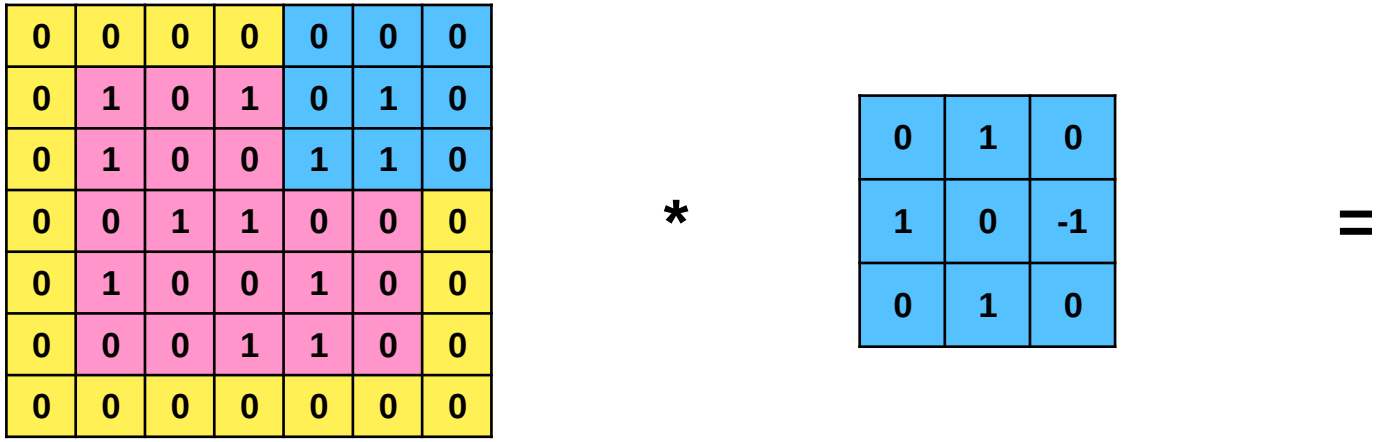
$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding



7 x 7

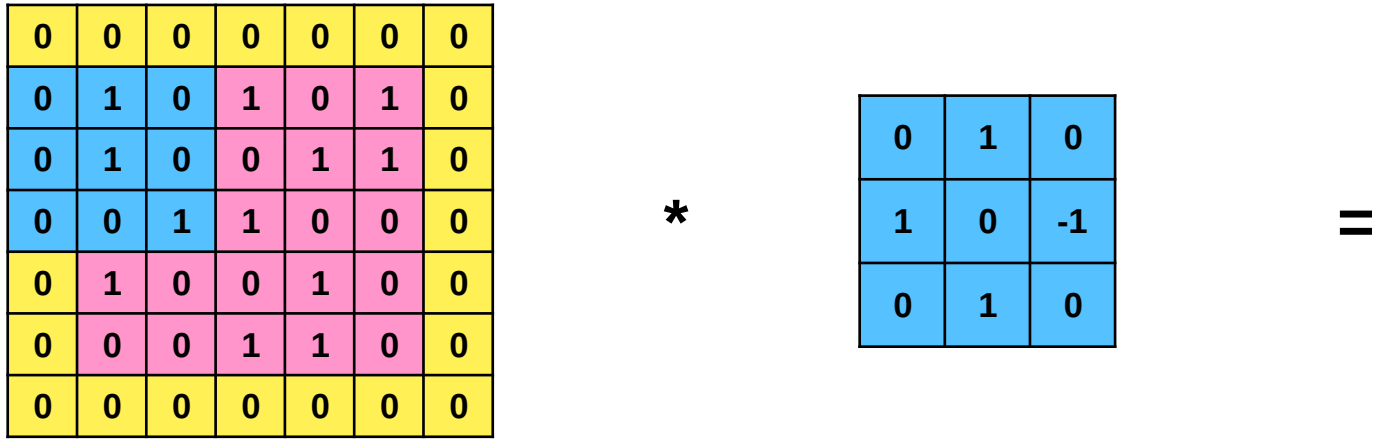
$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding



7 x 7

$n \times n$

3 x 3

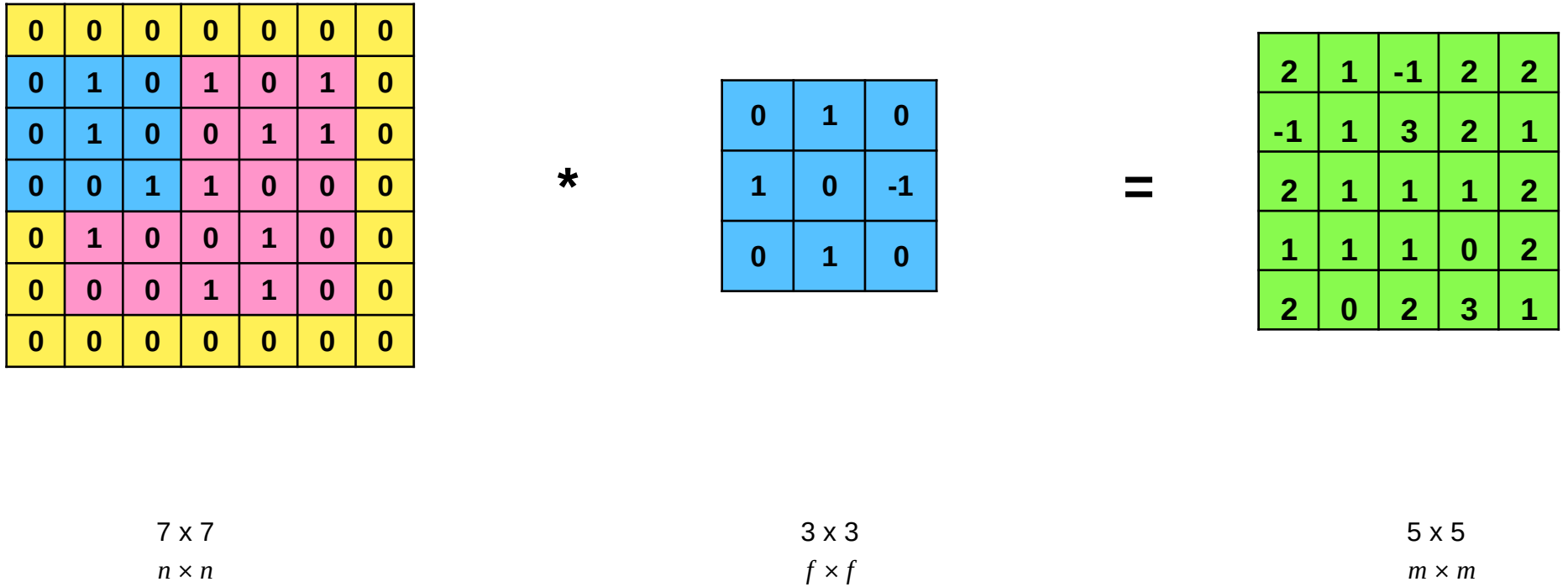
$f \times f$

Padding

Let's Perform our Convolution with the Padding

Feature Map Size = $n - f + 1 = m$

Feature Map Size = $7 - 3 + 1 = 5$



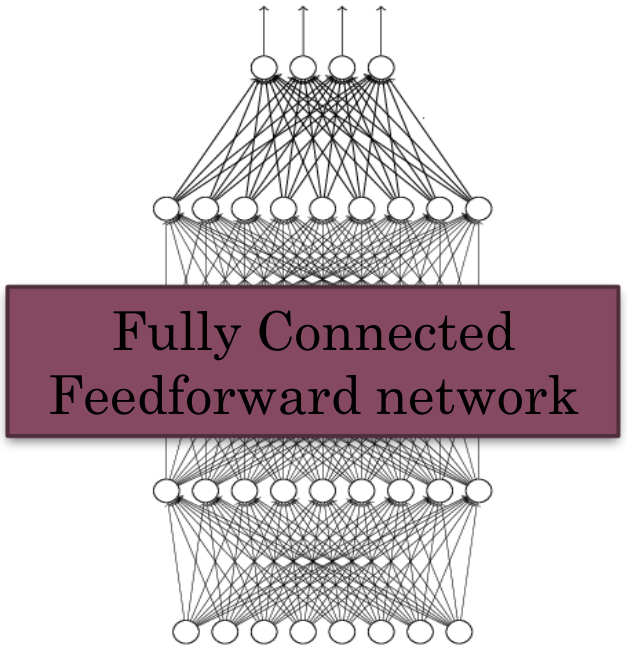
7 x 7
$n \times n$

3 x 3
$f \times f$

5 x 5
$m \times m$

The whole CNN

cat dog ……



Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flattened

Pooling

4	123	1	34
56	99	222	253
45	122	165	12
21	187	133	124

Max pool with 2x2 filters and stride 2

123	253
187	165

1. Reduced dimensionality
2. Spatial invariance

```
torch.nn.MaxPool2d(
    kernel_size=(2,2),
    stride=2
)
```

```
tf.keras.layers.MaxPool2D(
    pool_size=(2,2),
    strides=2
)
```

Why Pooling

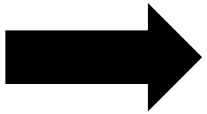
- Subsampling pixels will not change the object

bird

Subsampling

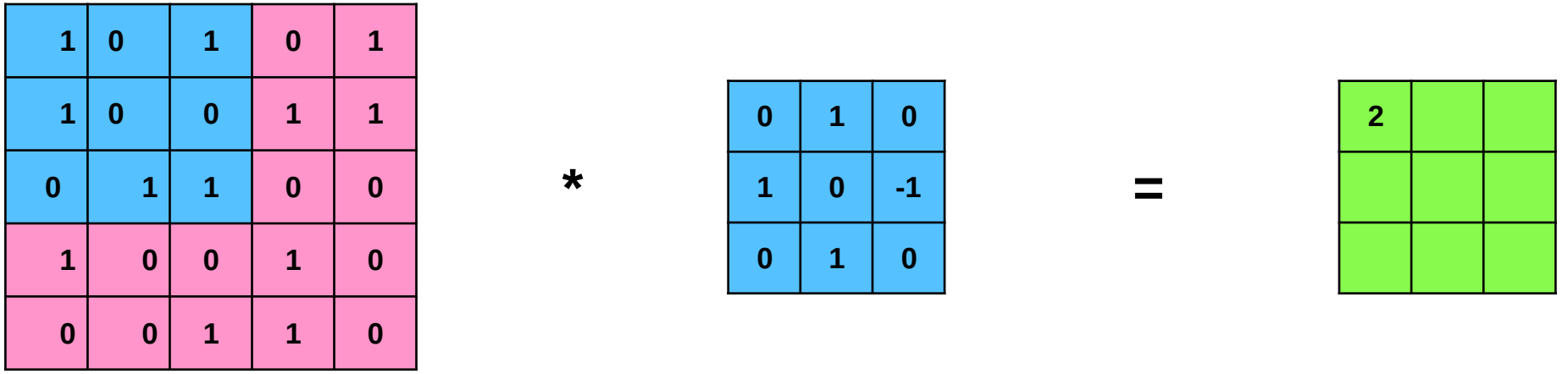
bird

We can subsample the pixels to make image smaller



fewer parameters to characterize the image

What a Stride of 1 Looks Like

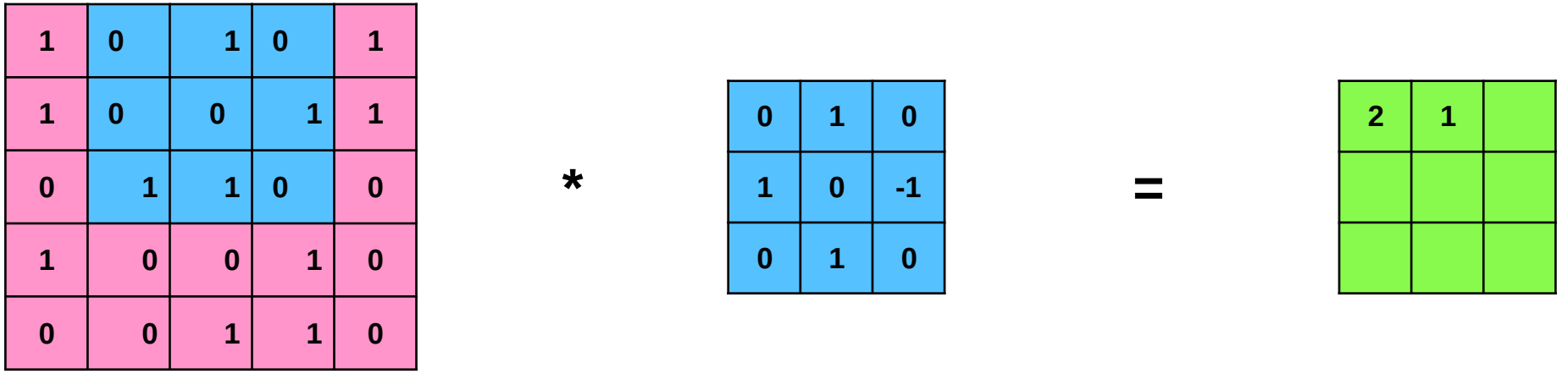


Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 1 Looks Like

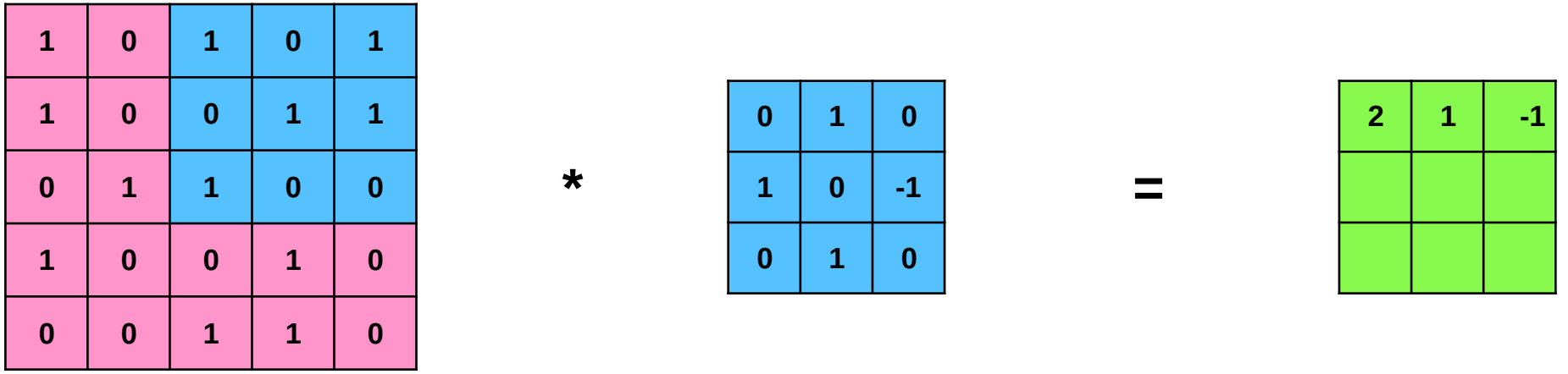


Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 1 Looks Like

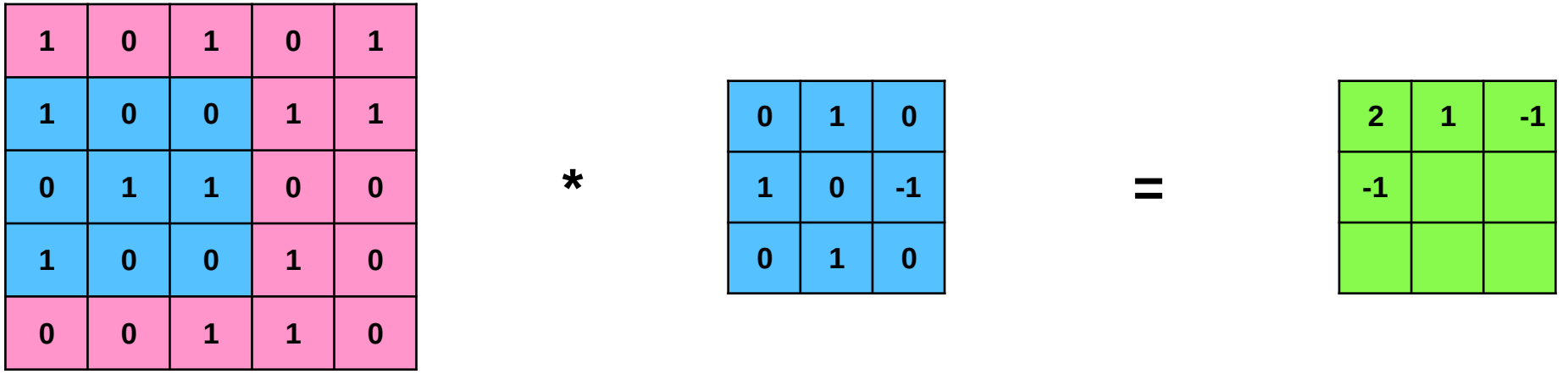


Input Image

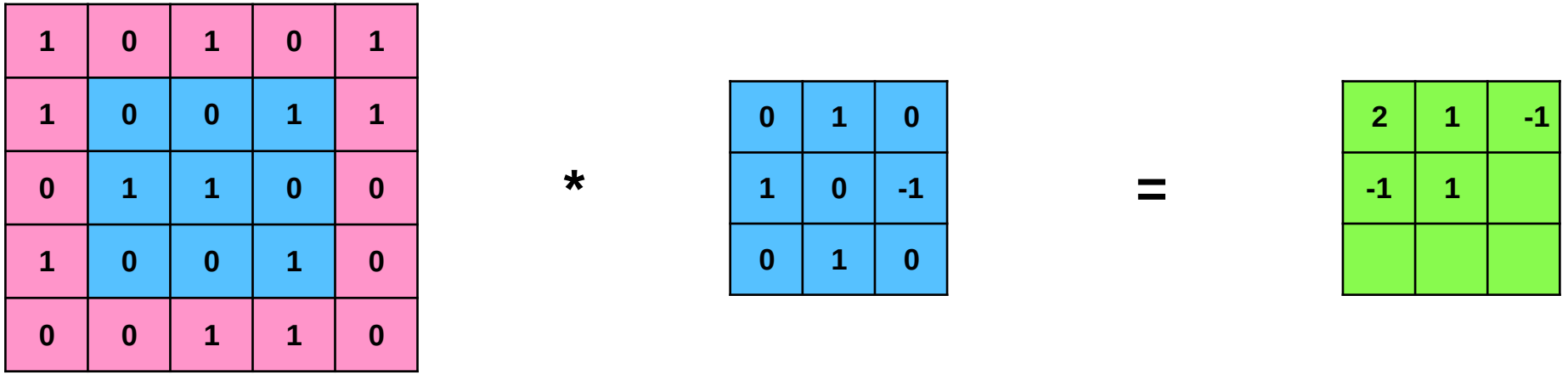
Filter or Kernel

Output or Feature Map

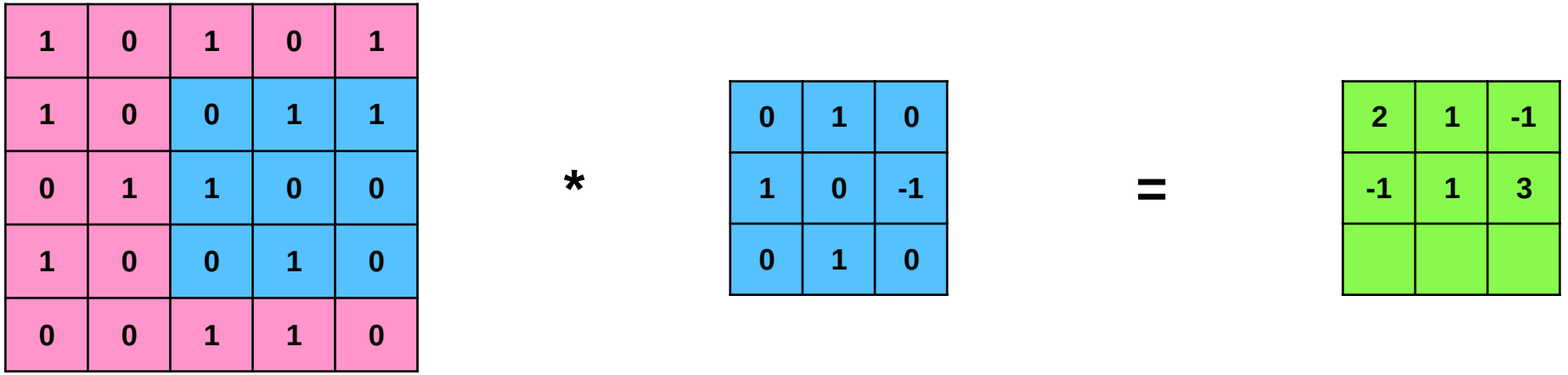
What a Stride of 1 Looks Like



What a Stride of 1 Looks Like



What a Stride of 1 Looks Like

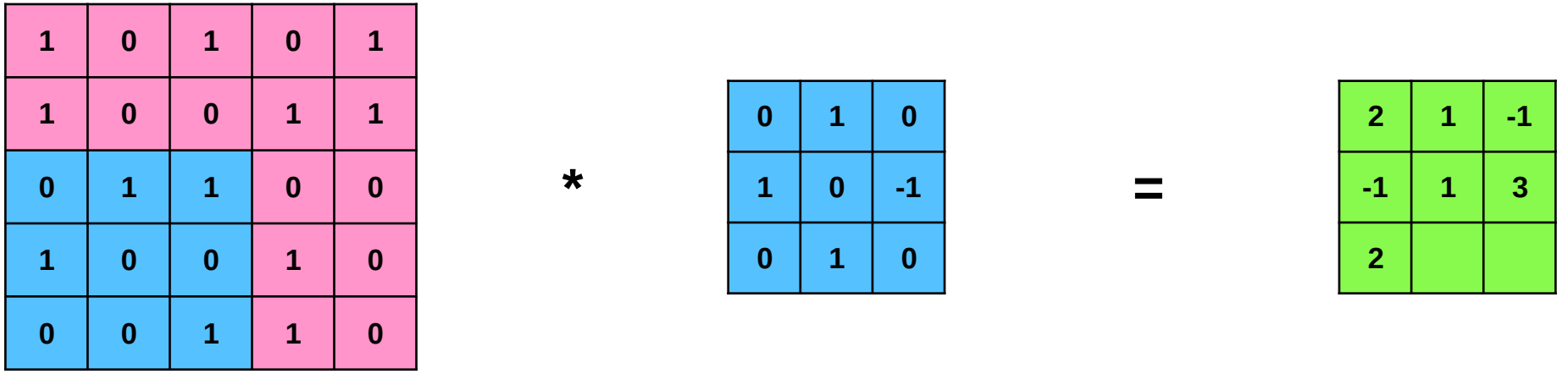


Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 1 Looks Like

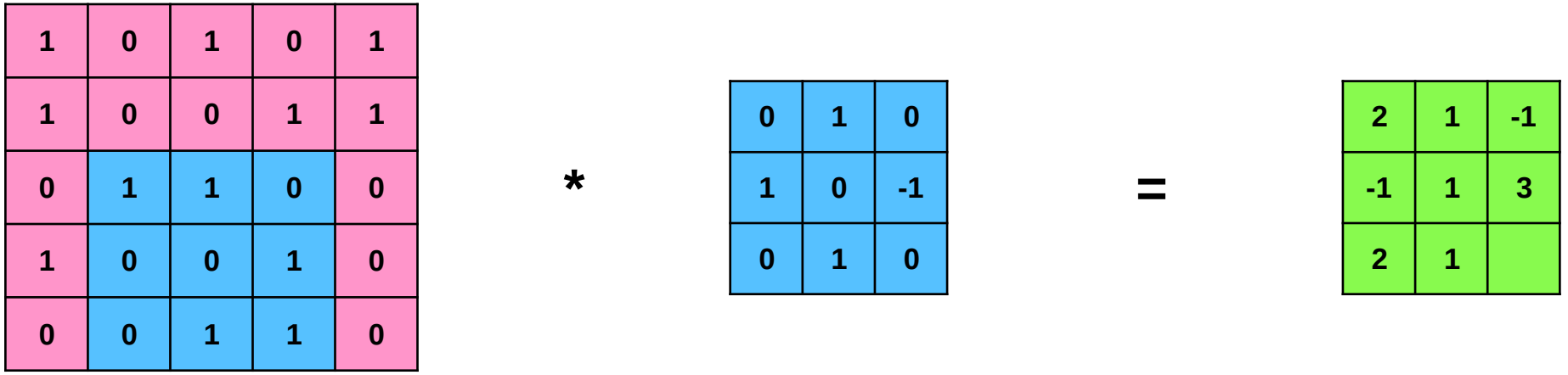


Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 1 Looks Like

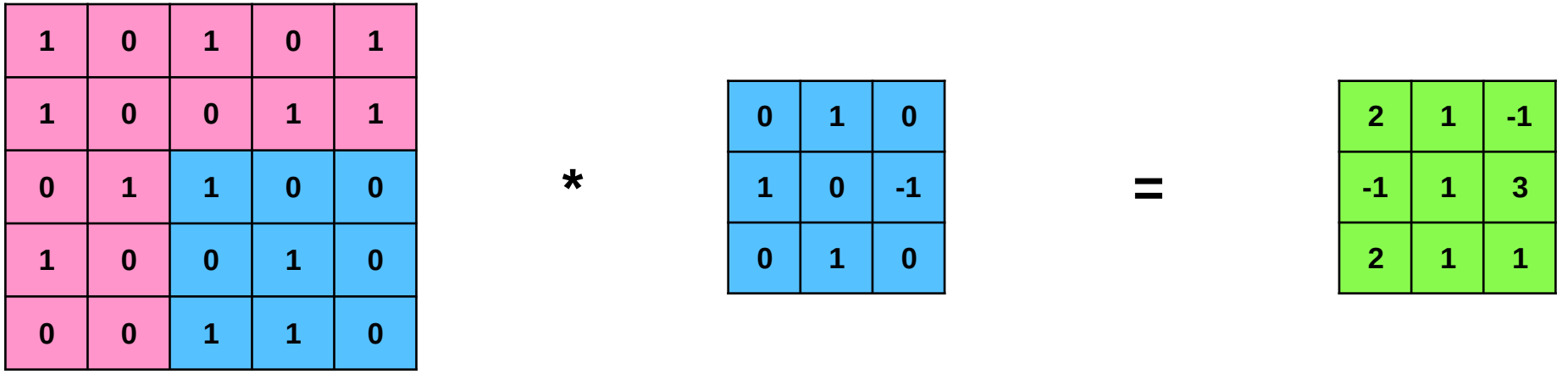


Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 1 Looks Like



Input Image

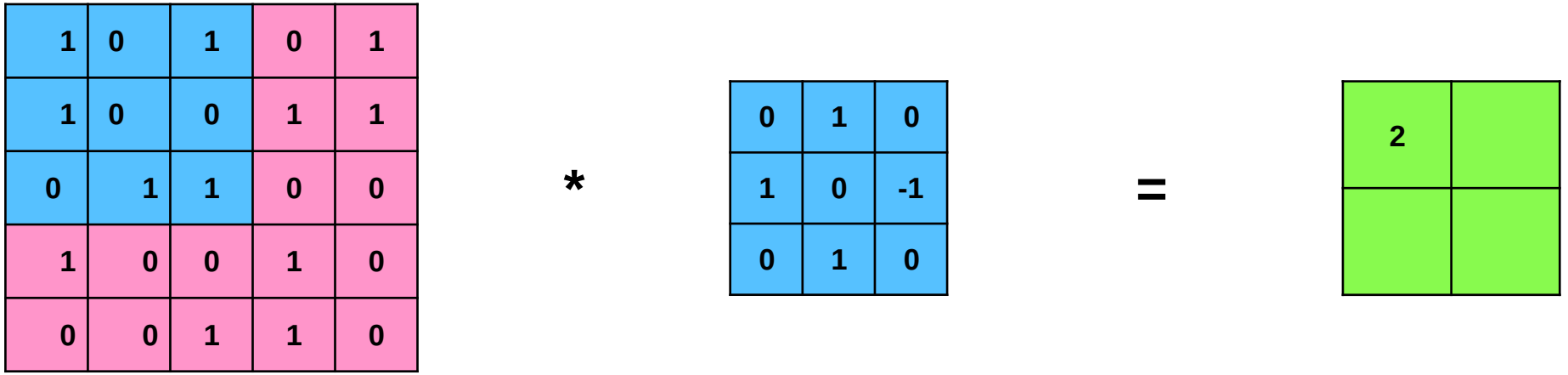
Filter or Kernel

Output or Feature Map

What about a Stride of 2 ?

What a Stride of 2 Looks Like

We start off in the same position



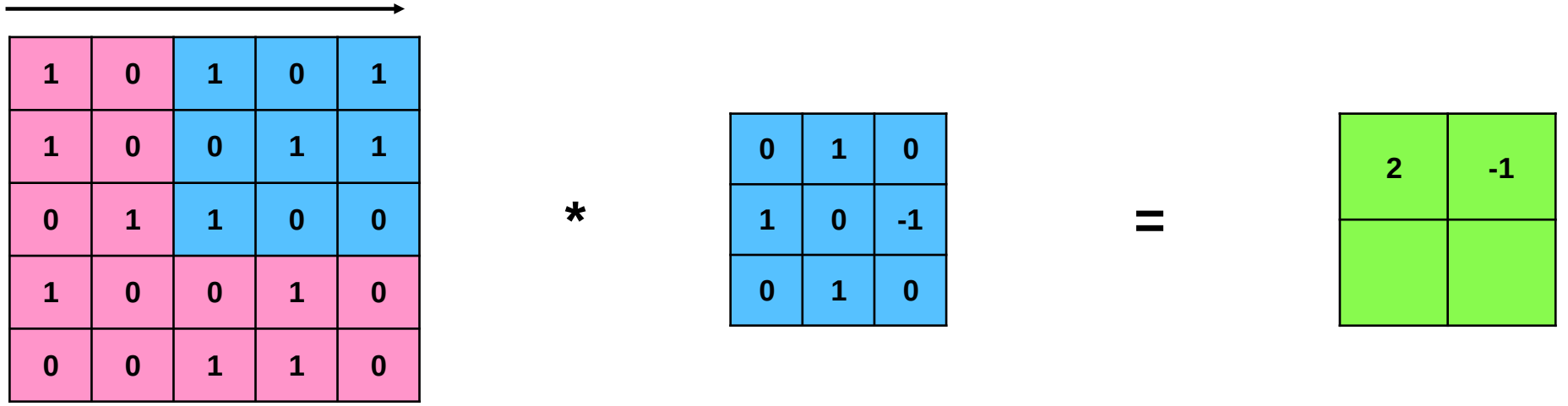
Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 2 Looks Like

Now we jump two spots to the left



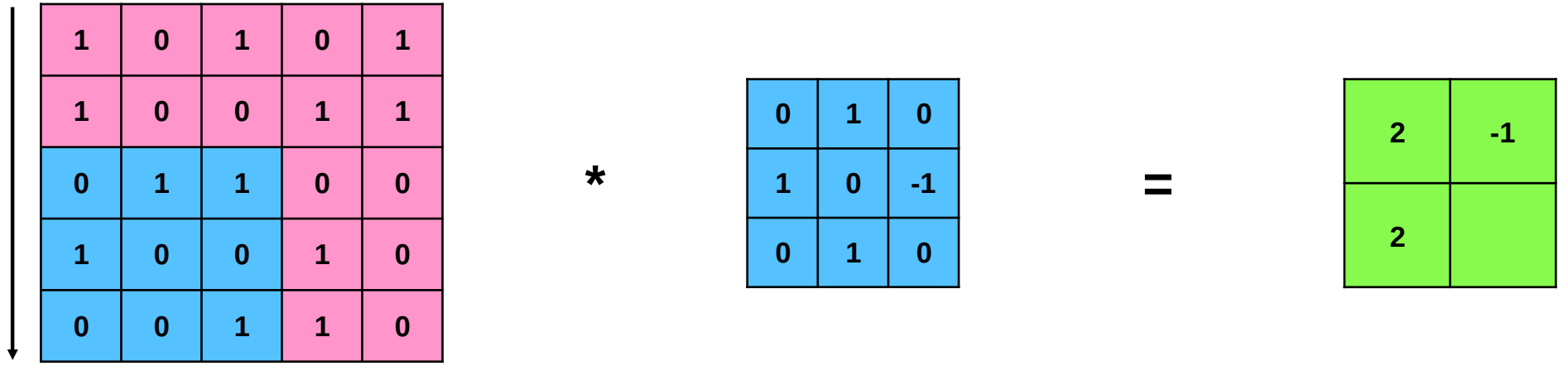
Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 2 Looks Like

Now we go down, but by also 2 spots



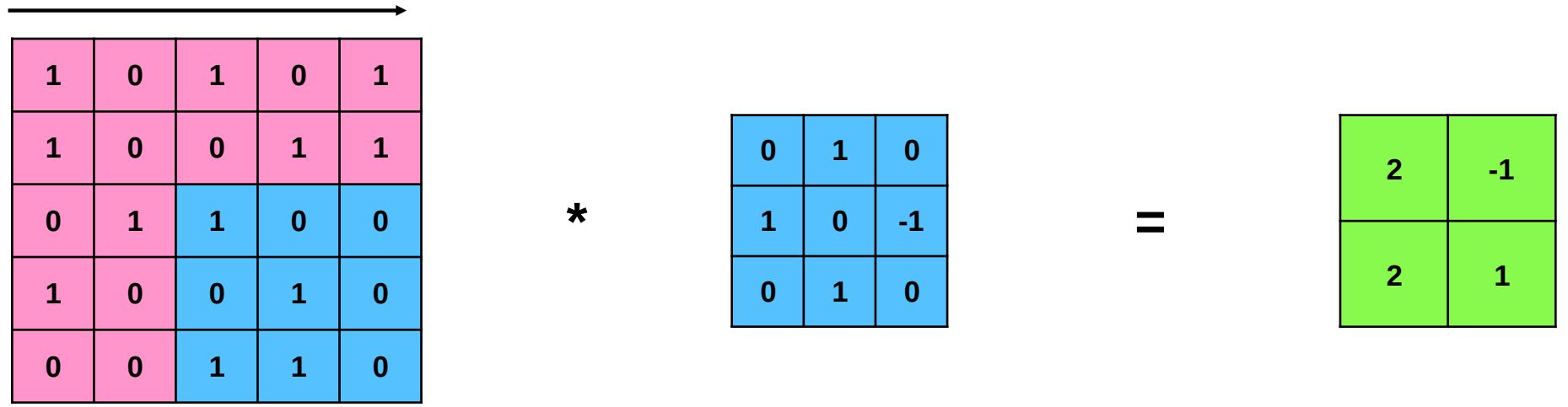
Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 2 Looks Like

Now we jump two spots to the left



Input Image

Filter or Kernel

Output or Feature Map

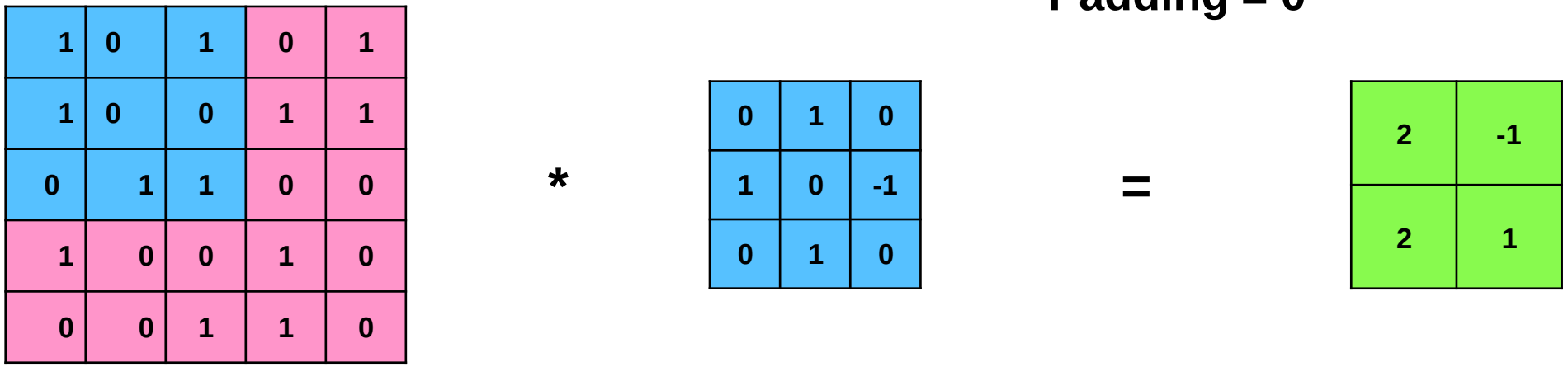
Stride Observations

- A larger Stride produced a **smaller** Feature Map output
- Larger Stride has **less overlap**
- We can use stride to **control the size of the Feature Map output**

Calculating Output Size

Using Stride and Padding

Stride = 2
Padding = 0



Input Image
5 x 5

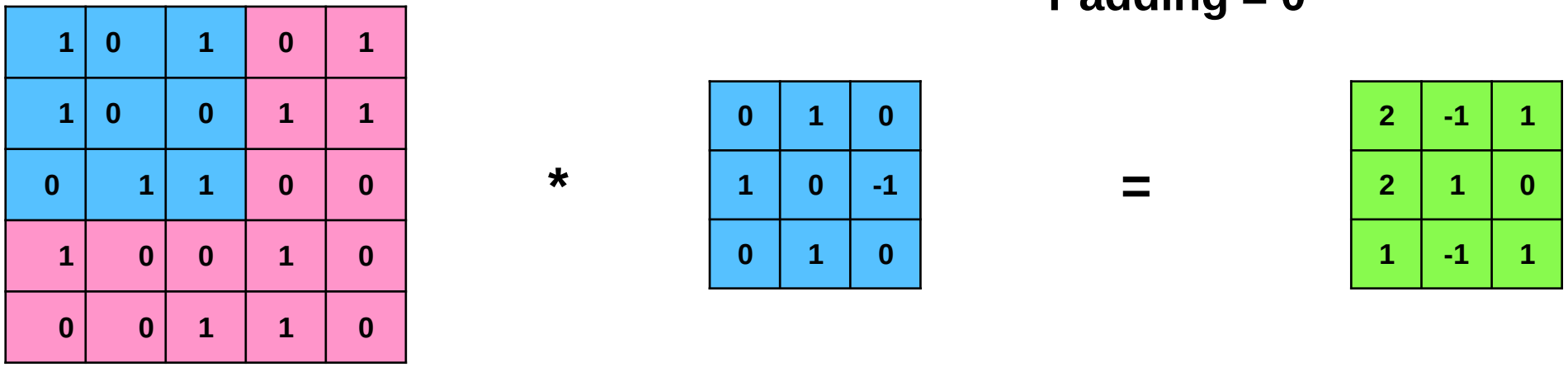
Filter or Kernel
3 x 3

$$(n \times n) * (f \times f) = (\frac{n + 2p - f}{s} + 1) \times (\frac{n + 2p - f}{s} + 1) = (\frac{5 + (2 \times 0) - 3}{2} + 1) \times (\frac{5 + (2 \times 0) - 3}{2} + 1) = 2 \times 2$$

Calculating Output Size

Using Stride and Padding

Stride = 1
Padding = 0



Input Image
5 x 5

Filter or Kernel
3 x 3

$$(n \times n) * (f \times f) = (\frac{n + 2p - f}{s} + 1) \times (\frac{n + 2p - f}{s} + 1) = (\frac{5 + (2 \times 0) - 3}{1} + 1) \times (\frac{5 + (2 \times 0) - 3}{1} + 1) = 3 \times 3$$

Calculating Output Size

Using Stride and Padding

Stride = 1
Padding = 1

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

Input Image
7 x 7

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel
3 x 3

=

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

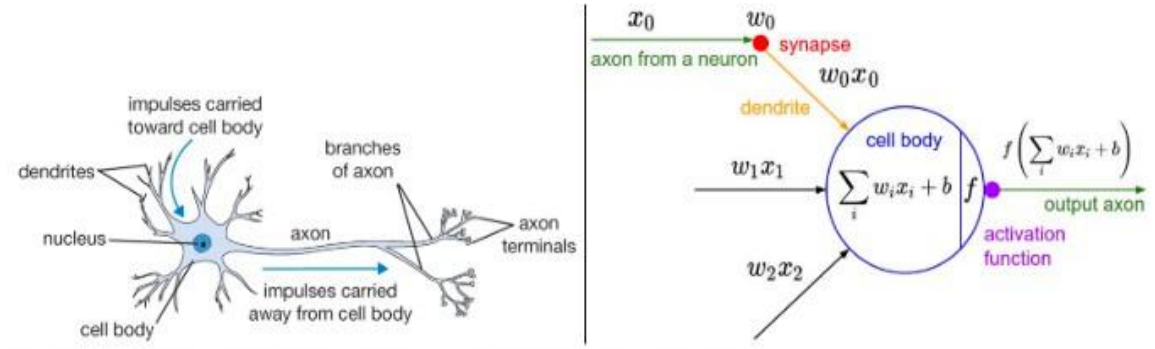
$$(n \times n) * (f \times f) = (\frac{n + 2p - f}{s} + 1) \times (\frac{n + 2p - f}{s} + 1) = (\frac{5 + (2 \times 1) - 3}{1} + 1) \times (\frac{5 + (2 \times 1) - 3}{1} + 1) = 5 \times 5$$

Note if we get non integer value for our output size, we found it down to the nearest integer

Purpose of Activation Functions

To enable the learning of **complex patterns** in our data

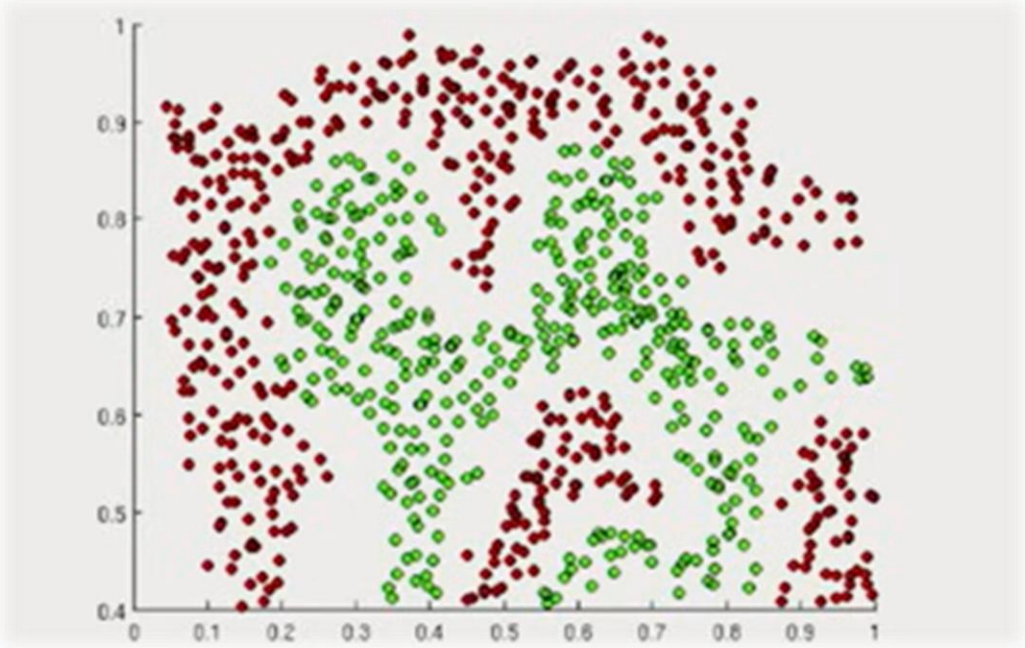
- Biological neurons fire (activate) on certain inputs, these are then fed into other neurons
- Introduces **non-linearity** to our network
- This allows a non-linear decision boundary via non-linear combinations of the weight and inputs



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Importance Activation Functions

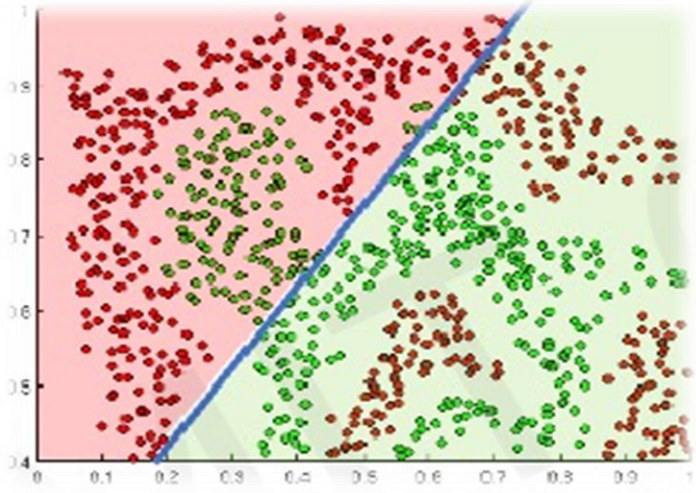
The purpose of activation functions is **to introduce non-linearity** into the network



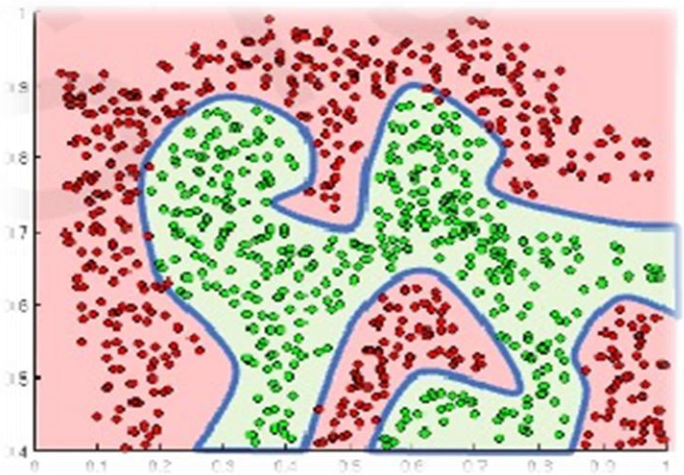
What if we wanted to build a neural network
to distinguish green vs red points?

Importance Activation Functions

The purpose of activation functions is **to introduce non-linearity** into the network



Linear activation functions produce linear decisions no matter the network size



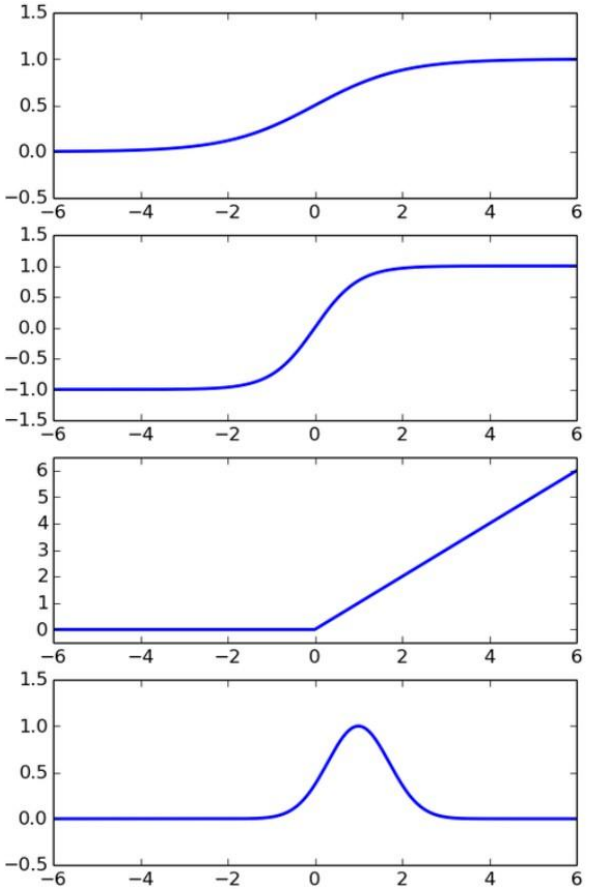
Non-linearities allow us to approximate arbitrarily complex functions

Types of Activation Functions

There are several activation functions we can use in our CNN. However, Rectified Linear Units (ReLU) have become the activation function of choice for CNNs.

ReLU is advantageous in CNN Training:

- Simple Computation (fast to train)
- Does not saturate



Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Hyperbolic Tangent

$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Rectified Linear

$$\phi(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

Radial Basis Function

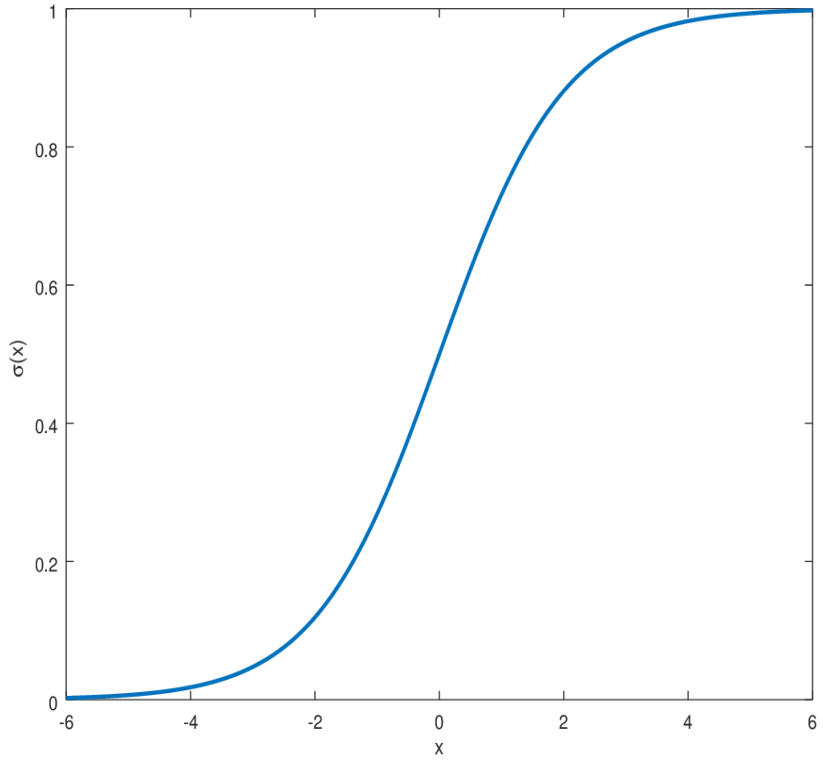
$$\phi(z, c) = e^{-(\epsilon \|z - c\|)^2}$$

Sigmoid Function (Logistic)

- **Explanation**: The sigmoid function squashes the input values between 0 and 1, making it useful in binary classification problems where we need to produce probabilities.
- **Formula**:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Usage**: Commonly used in the output layer for binary classification tasks, where it transforms the network's raw output into probabilities.



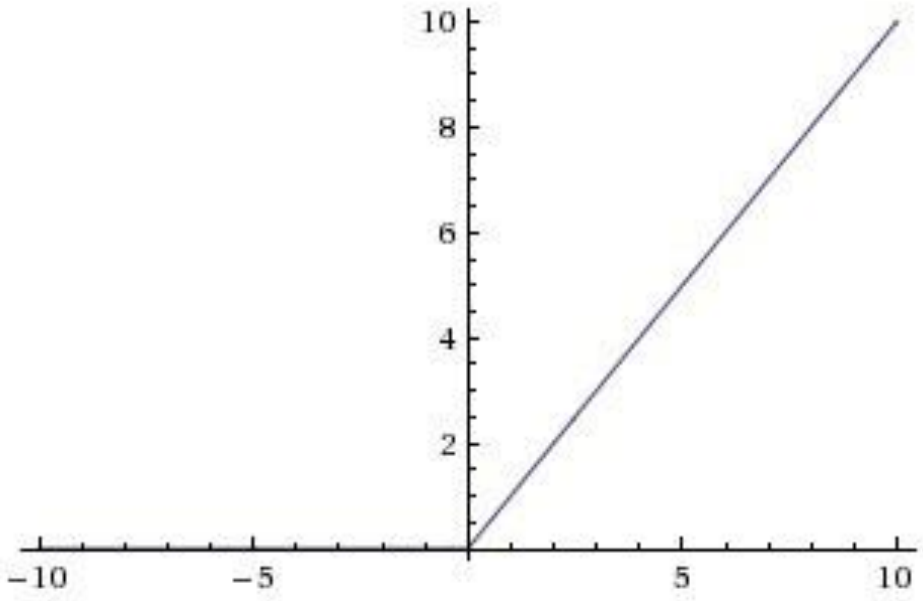
The ReLU Operation

- Change all negative values to 0
- Leave all positive Values alone

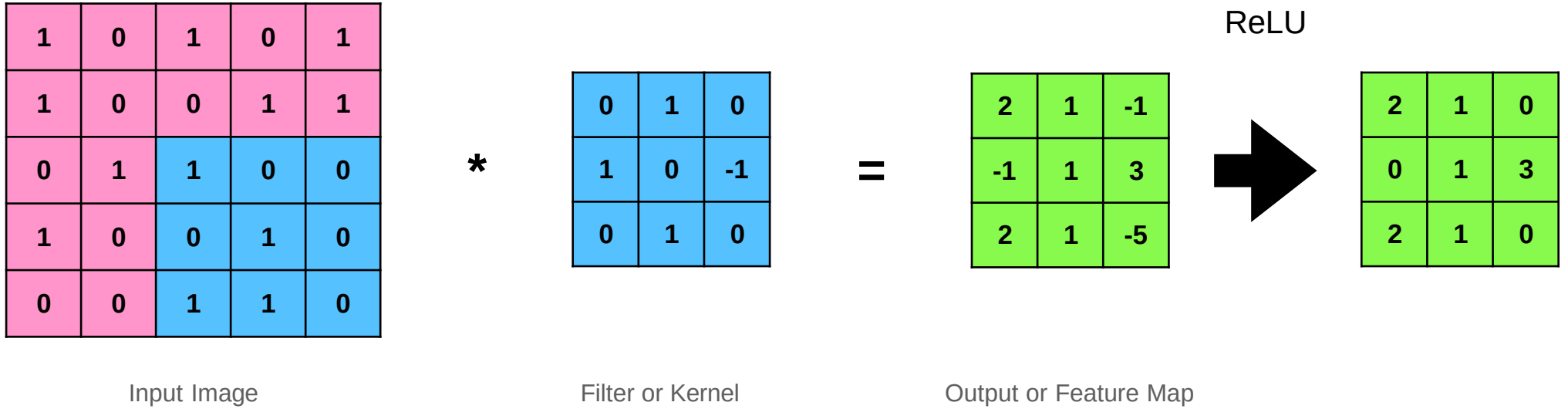
Formula:

$$f(x) = max(0,x)$$

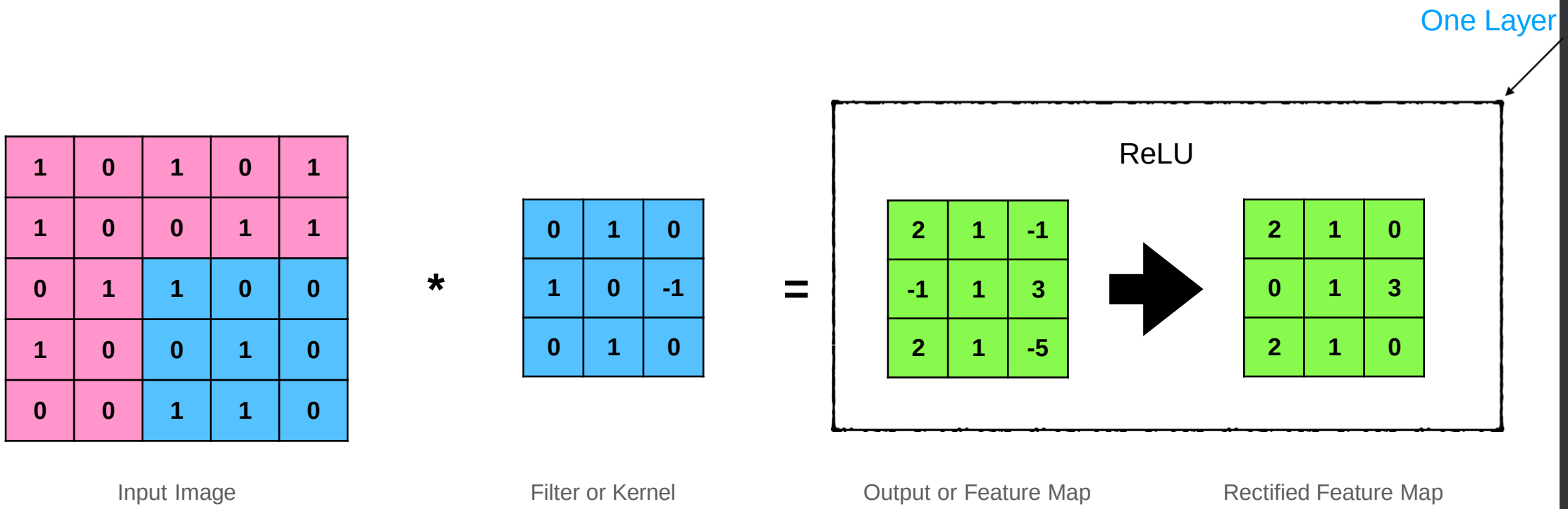
$f(x) = max(0,x)$



Applying the ReLU Activation



Applying the ReLU Activation



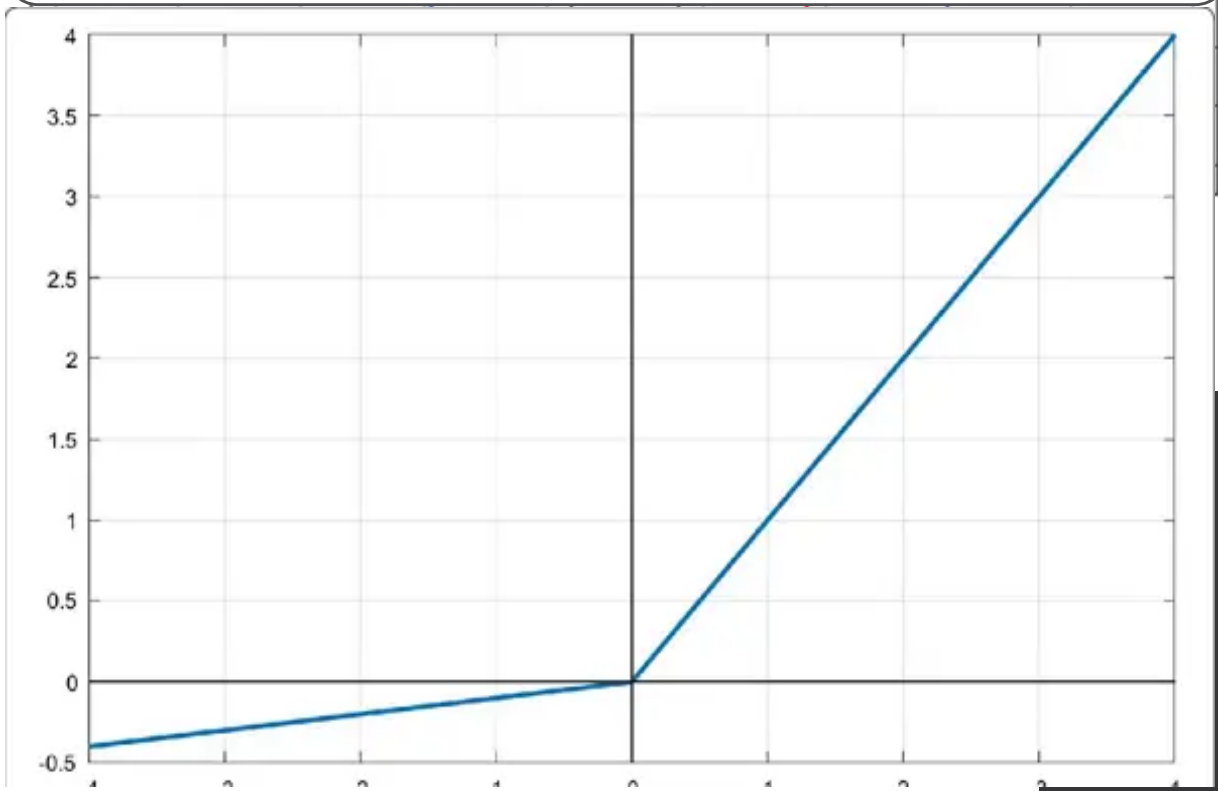
Leaky ReLU

- Dying ReLU → neurons using the ReLU activation function become inactive during training and never recover.
- Leaky ReLU is similar to ReLU but has a small slope for negative inputs, which helps mitigate the “dying ReLU” problem.
- **Formula**:

$$f(x) = max\ (0.1x, x)$$

- **Usage**: Leaky ReLU addresses the “dying ReLU” problem by allowing a small gradient for negative inputs, which can be beneficial in training deep networks.

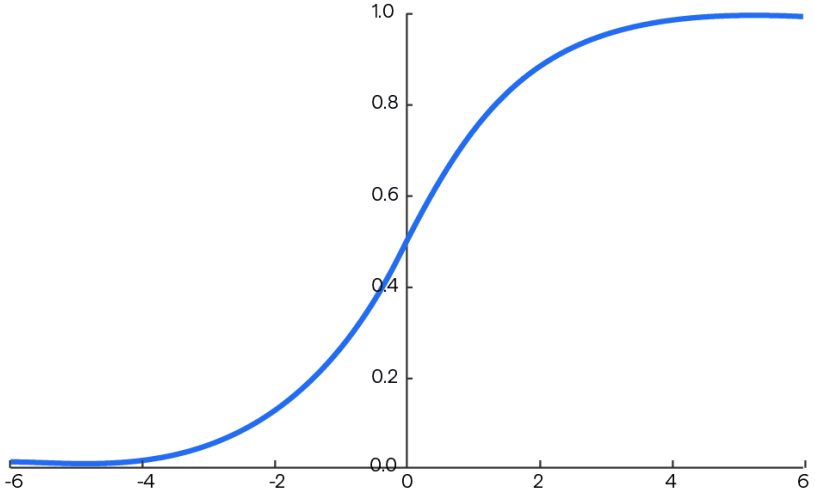
Solution



Softmax Function

- **Explanation**: The softmax function is commonly used in the output layer of neural networks for multi-class classification problems. It converts raw scores (logits) into probabilities, ensuring that the sum of the probabilities for all classes is equal to 1.
- **Formula**:

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$



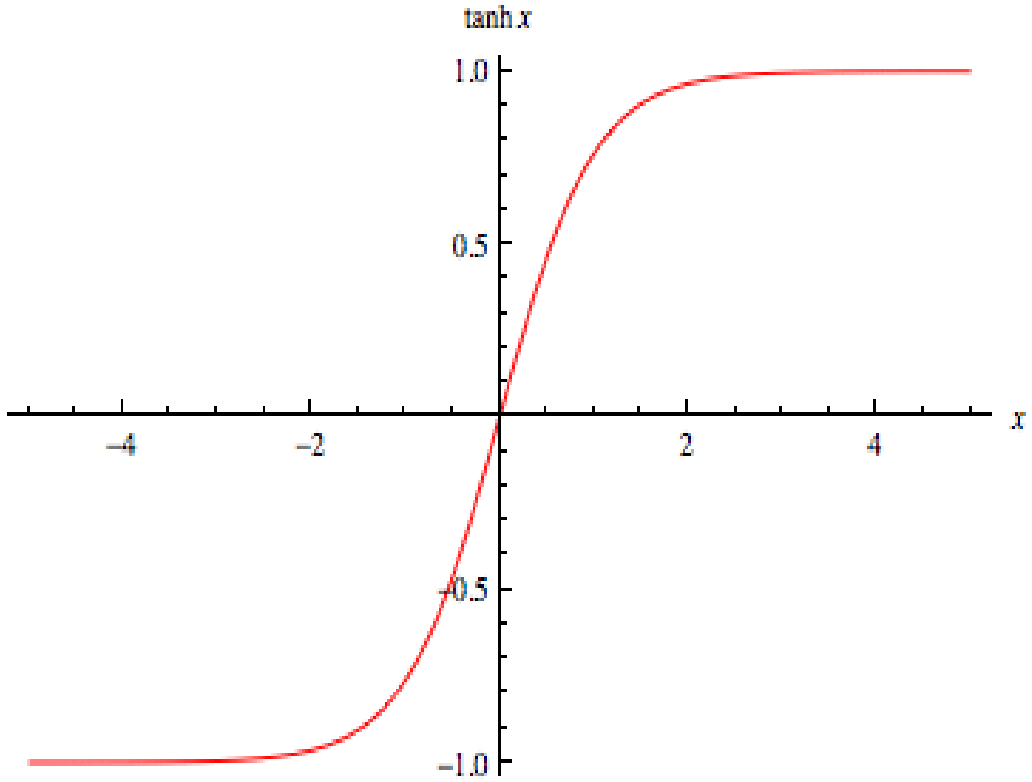
- **Usage**: Softmax transforms the final layer activations into a probability distribution, allowing the model to make predictions about the likelihood of each class.

Hyperbolic Tangent Function (Tanh)

- **Explanation**: Tanh function squashes the input values between -1 and 1, which can be useful for normalization and also in classification tasks.
- **Formula**:

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

- **Usage**: Similar to other activations function, tanh is used in the hidden layers of neural networks. Specifically useful in GANs and GenAI.



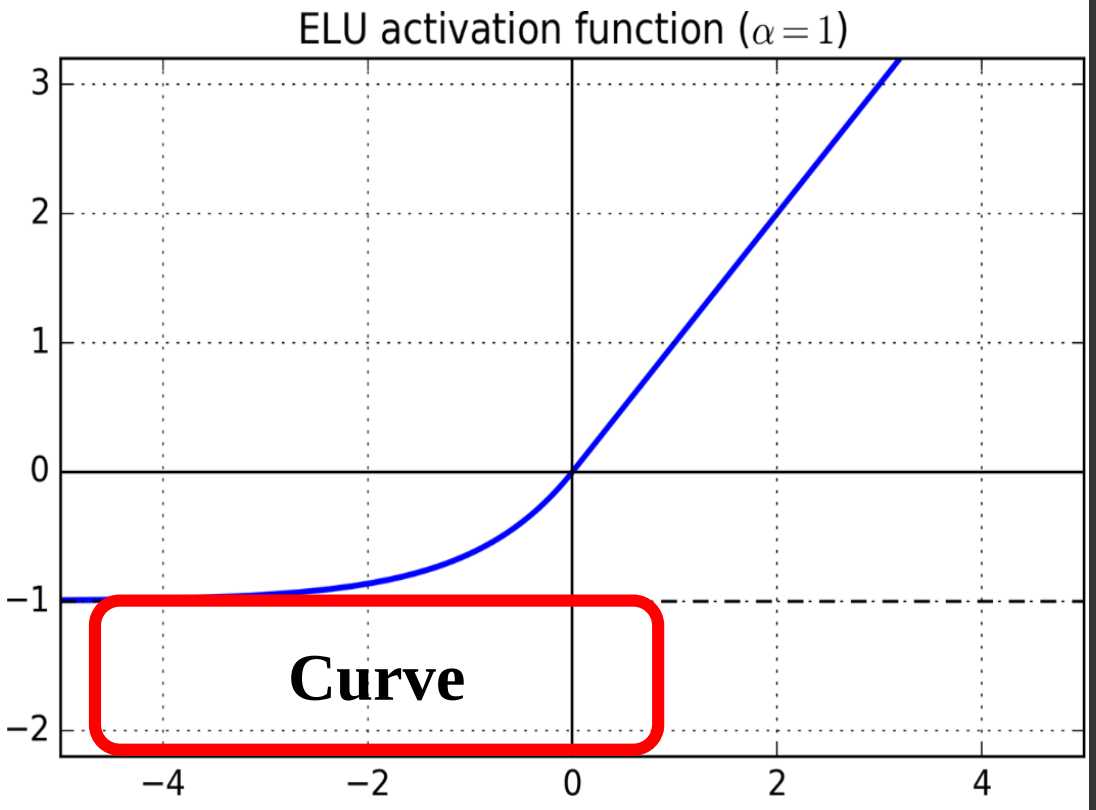
Exponential Linear Unit (ELU)

- **Explanation**: ELU is similar to ReLU for positive inputs but allows negative values with a smooth curve, aiming to address some limitations of ReLU.

- **Formula**

$$\text{elu}(x) = \begin{cases} x, & x > 0 \\ \alpha\left(\exp(x) - 1\right), & x \leq 0 \end{cases}$$

- **Usage**: ELU mitigates the limitations of ReLU by handling negative inputs with a smooth curve, which can improve the robustness of deep networks.

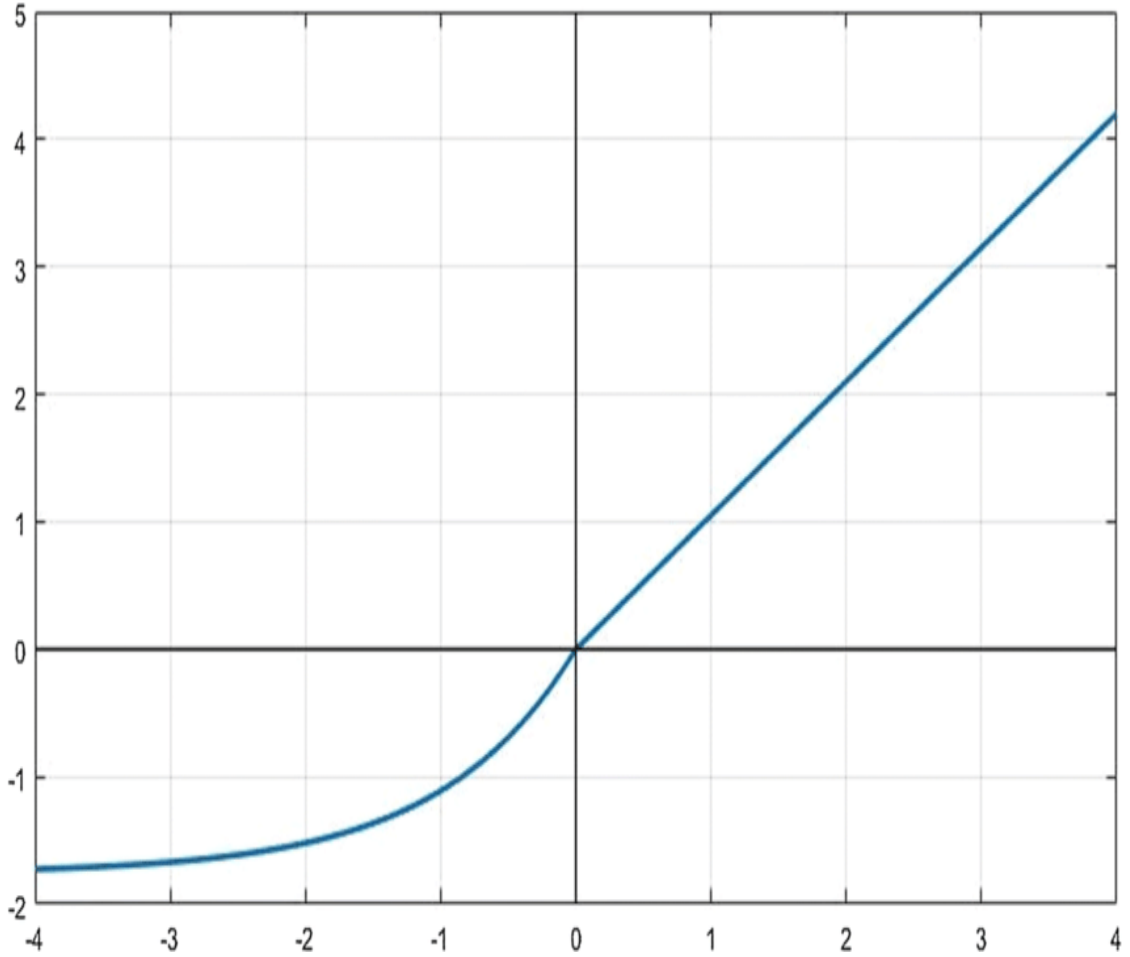


Scaled Exponential Linear Unit (SELU)

- **Explanation**: SELU is designed to maintain the mean and variance of the activations close to 0 and 1 respectively, aiding convergence.
- **Formula**

$$f(\alpha, x) = \lambda \begin{cases} \alpha\left(e^x - 1\right) & for\ x < 0 \\ x & for\ x \geqslant 0 \end{cases}$$

- **Usage**: SELU is designed to maintain the stability of activations throughout the network, often leading to better convergence and performance in deep architectures.

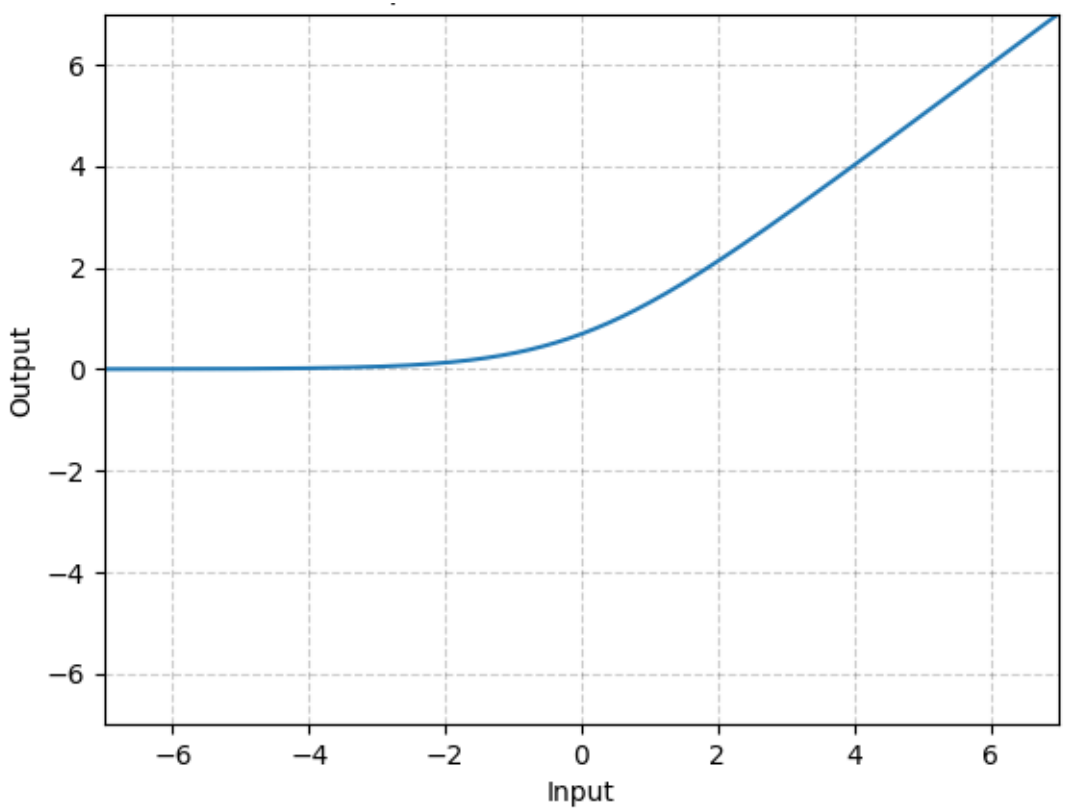


Softplus Function

- **Explanation**: Softplus is a smooth version of ReLU and can be used as an alternative activation function in some cases.
- **Formula**

$$f(x) = In(1 + e^x)$$

- **Usage**: Softplus is a smooth approximation of ReLU and can be used in scenarios where a differentiable activation function is required.

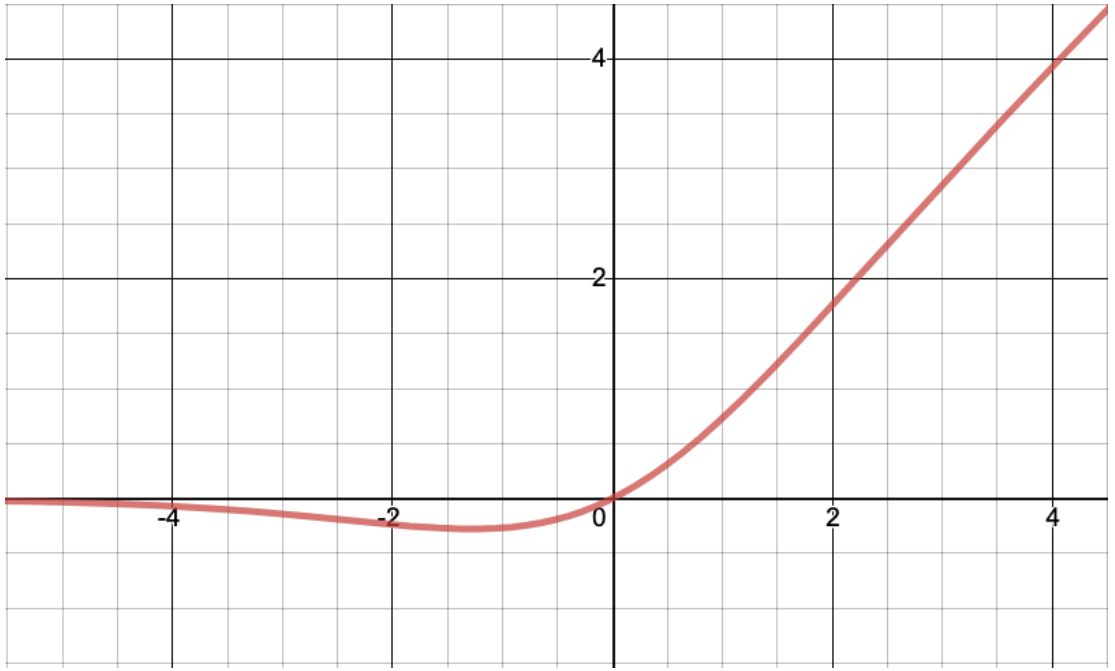


Swish Function

- **Explanation**: Swish function is a recently proposed activation function that tends to perform better than ReLU in certain scenarios.
- **Formula**

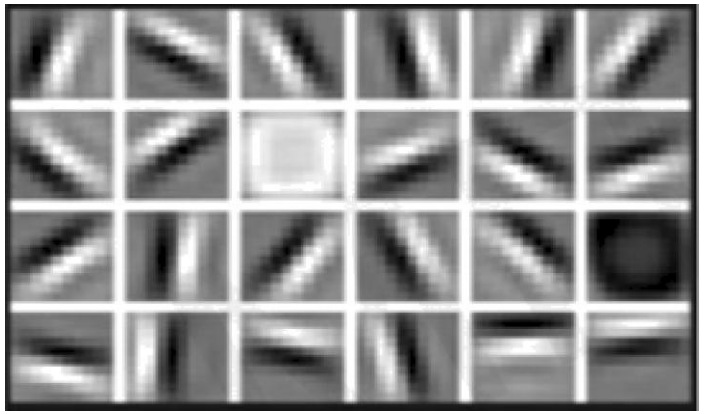
$$f(x)=x\cdot\text{sigmoid}(x)$$

- **Usage**: Swish is an alternative to ReLU, offering potentially better performance, especially in large-scale datasets and deeper networks.



Representation Learning in Deep CNNs

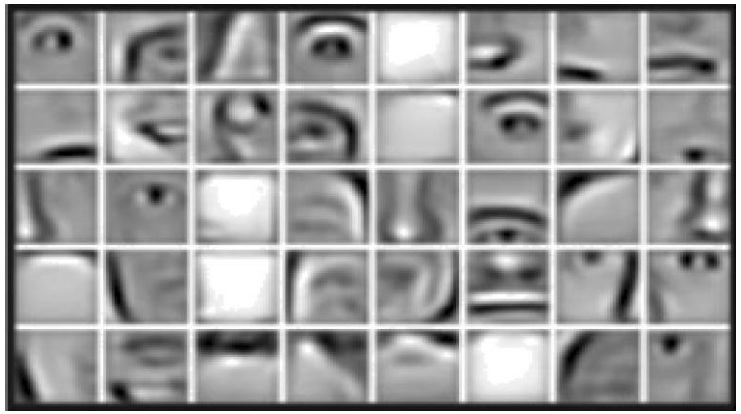
Low Level Features



Edges, dark spots

Conv Layer 1

Mid Level Features



Eyes & Nose & Ears

Conv Layer 2

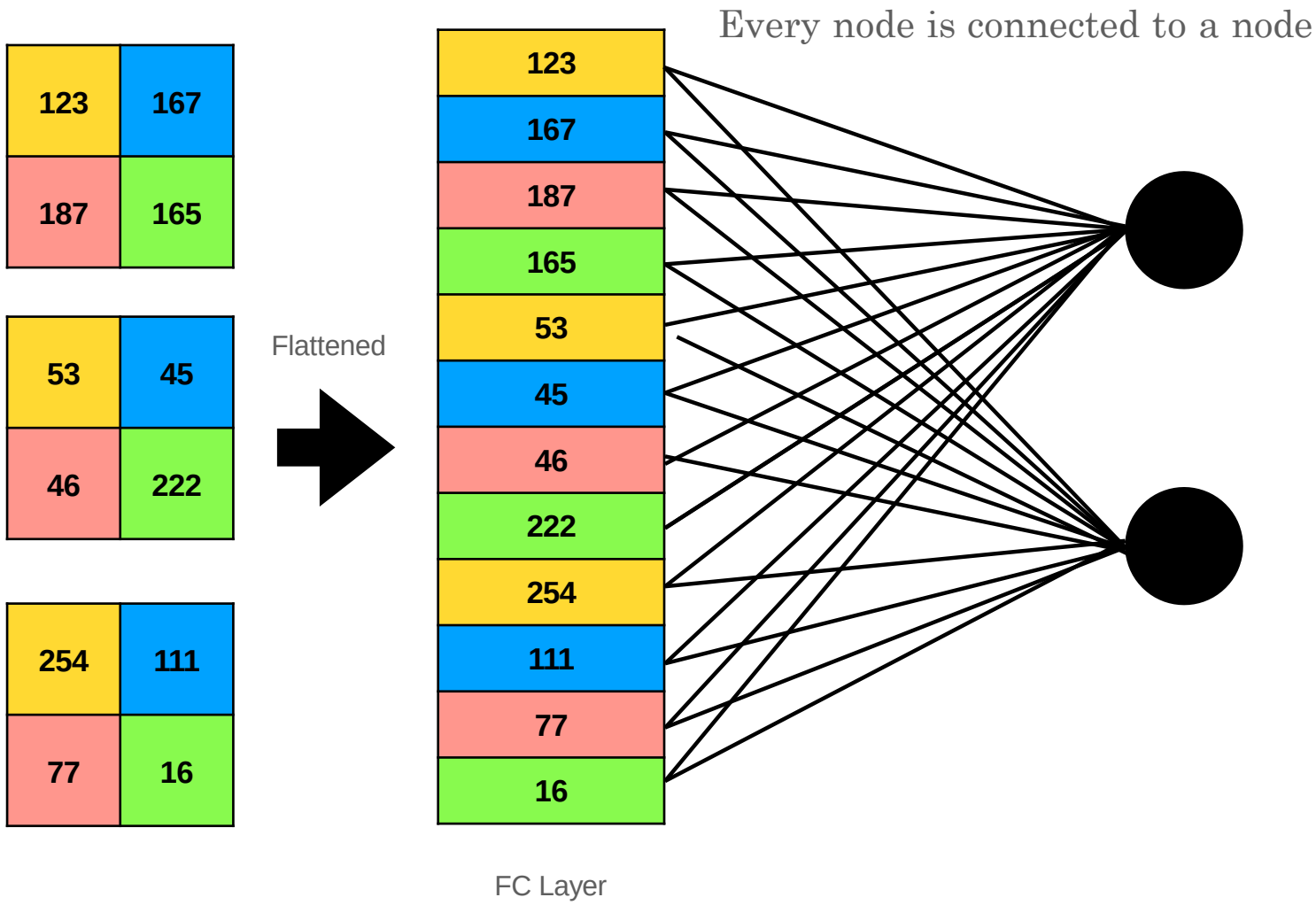
High Level Features

Facial Structure

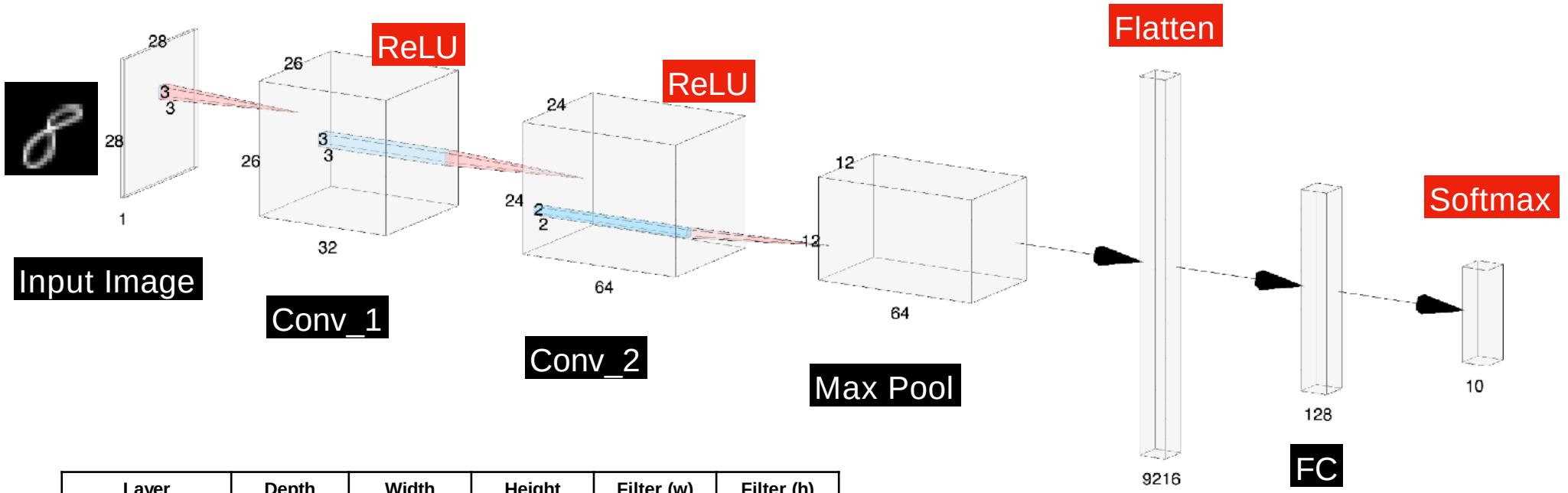
Conv Layer 3

Fully Connected (Dense) Layer

The Max Pool Layer is Flattened



Another Representation

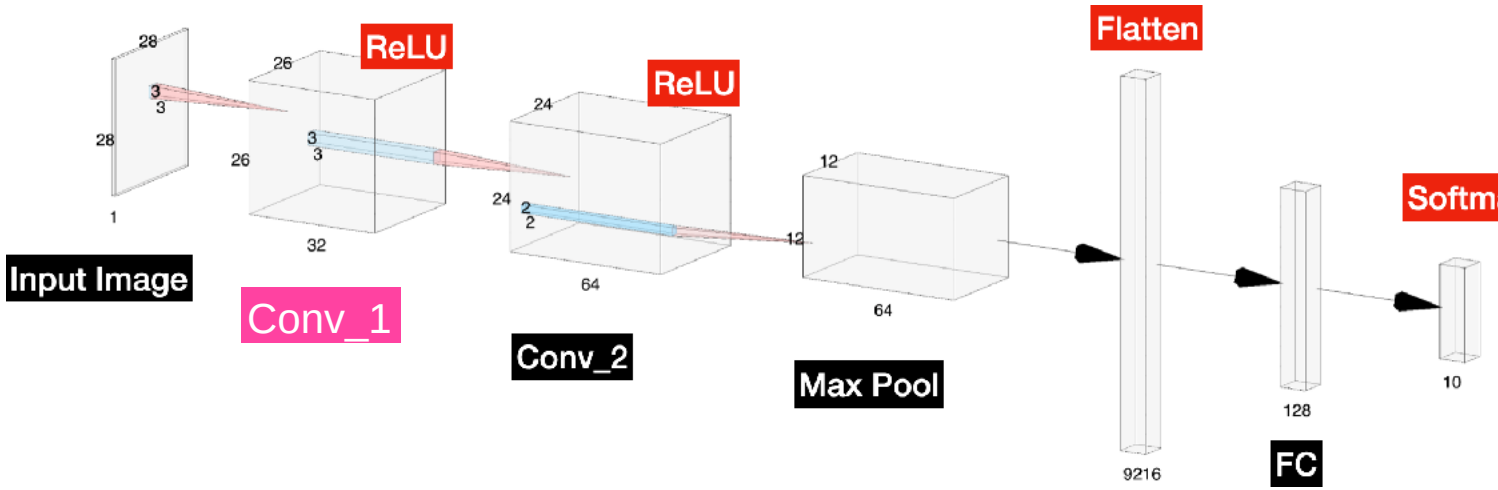


Layer	Depth	Width	Height	Filter (w)	Filter (h)
Input	1	28	28		
Conv_1	32	26	26	3	3
Conv_2	64	24	24	3	3
Max Pool	64	12	12	2	2
Flatten	9216	1	1		
Fully Connected	128	1	1		
Output	10	1	1		

Notes:

- We choose 32 & 64 Filters or Kernels for Conv_1 & Conv_2
- The Feature Maps are shown as Conv_1 and Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

Calculating the Output Size of Conv_1



Notes:

- We choose 32 Filters or Kernels for Conv_1
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{28 + (2 \times 0) - 3}{1} + 1\right) \times \left(\frac{28 + (2 \times 0) - 3}{1} + 1\right) = 26 \times 26$$

Where

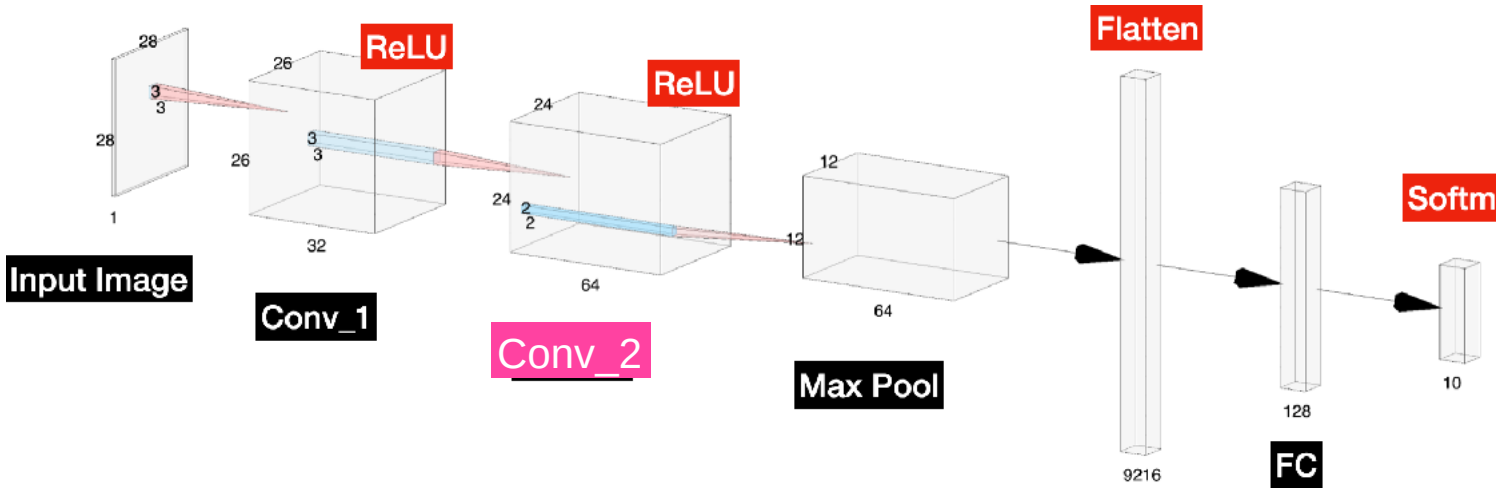
$n = 28$

$f = 3$

$s = 1$

$p = 0$

Calculating the Output Size of Conv_2



Notes:

- We choose 64 Filters or Kernels for Conv_2
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{26 + (2 \times 0) - 3}{1} + 1\right) \times \left(\frac{26 + (2 \times 0) - 3}{1} + 1\right) = \text{24 x 24}$$

Where

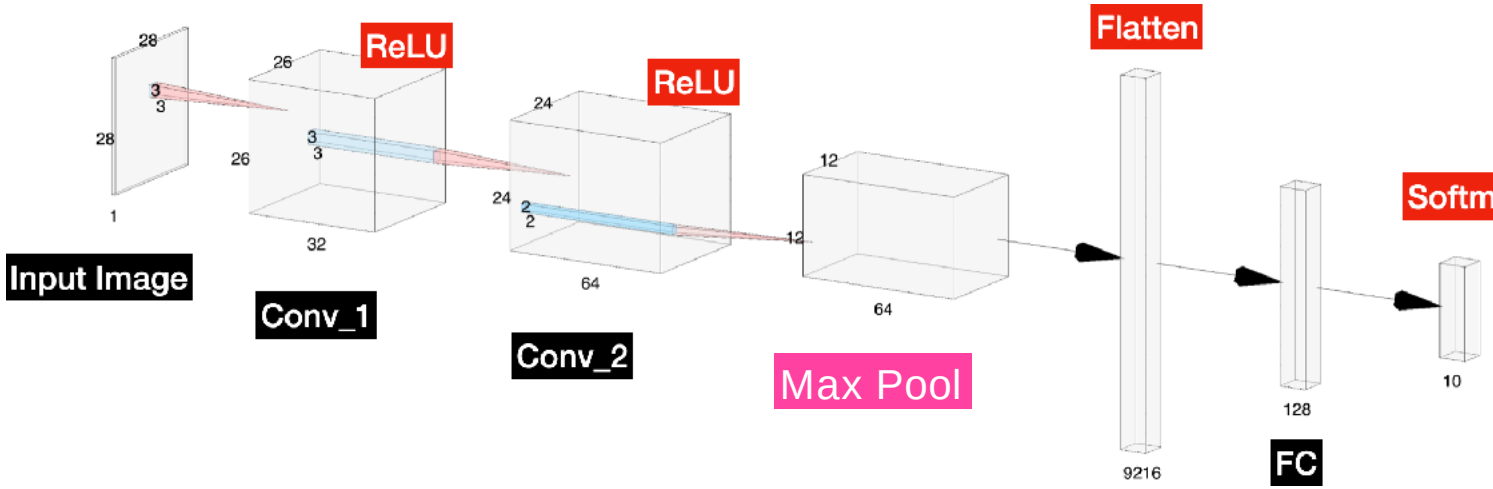
$n = 26$

$f = 3$

$s = 1$

$p = 0$

Calculating the Output Size of the Max Pool Layer



Notes:

- We choose 64 Filters or Kernels for Conv_2
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{24 + (2 \times 0) - 2}{2} + 1\right) \times \left(\frac{24 + (2 \times 0) - 2}{2} + 1\right) = 12 \times 12$$

12 x 12

Where

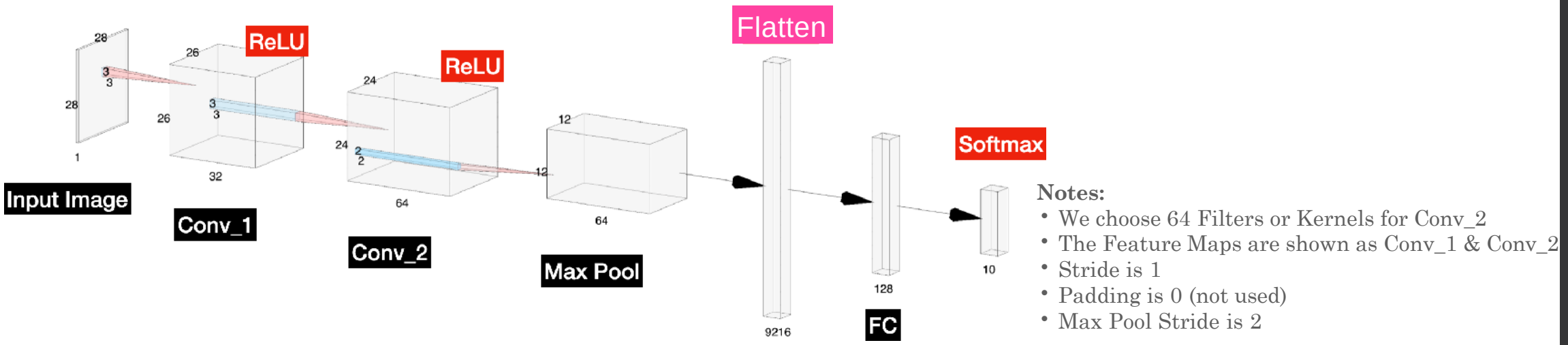
$n = 24$

$f = 2$

$s = 2$

$p = 0$

Calculating the Output Size of Flattened Layer



Notes:

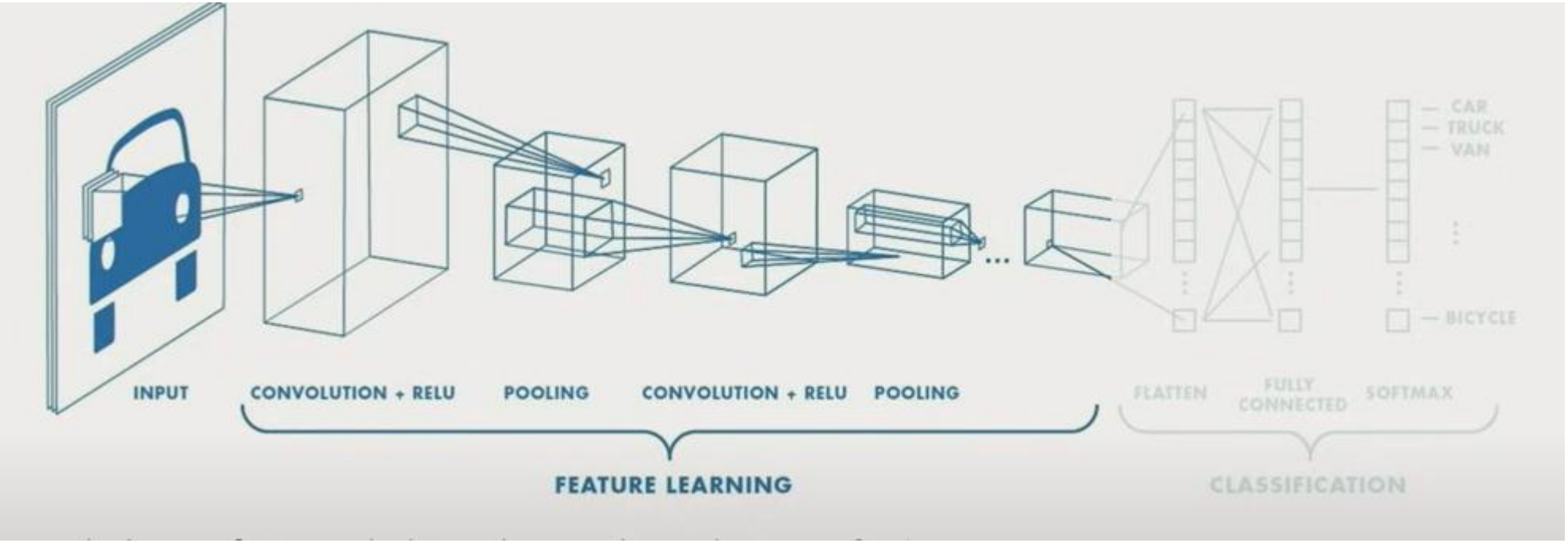
- We choose 64 Filters or Kernels for Conv_2
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$12 \times 12 \times 64 = 9216$$

Where

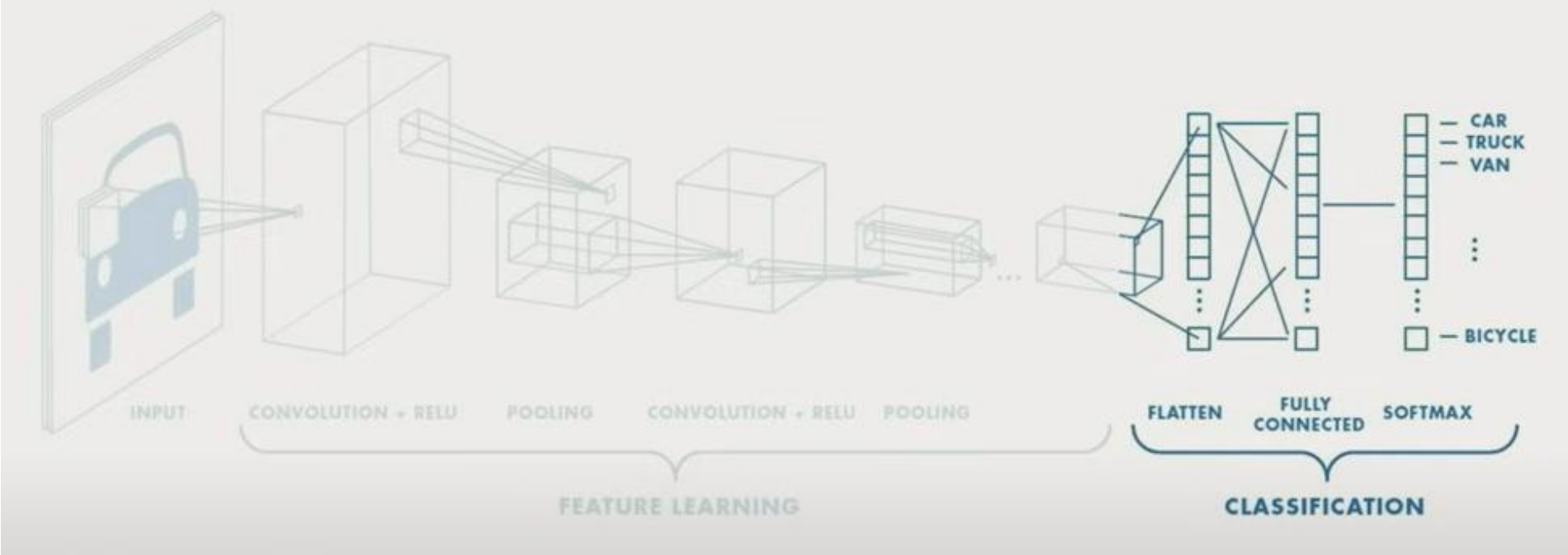
$n = 12$

CNN for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce non-linearity through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with pooling

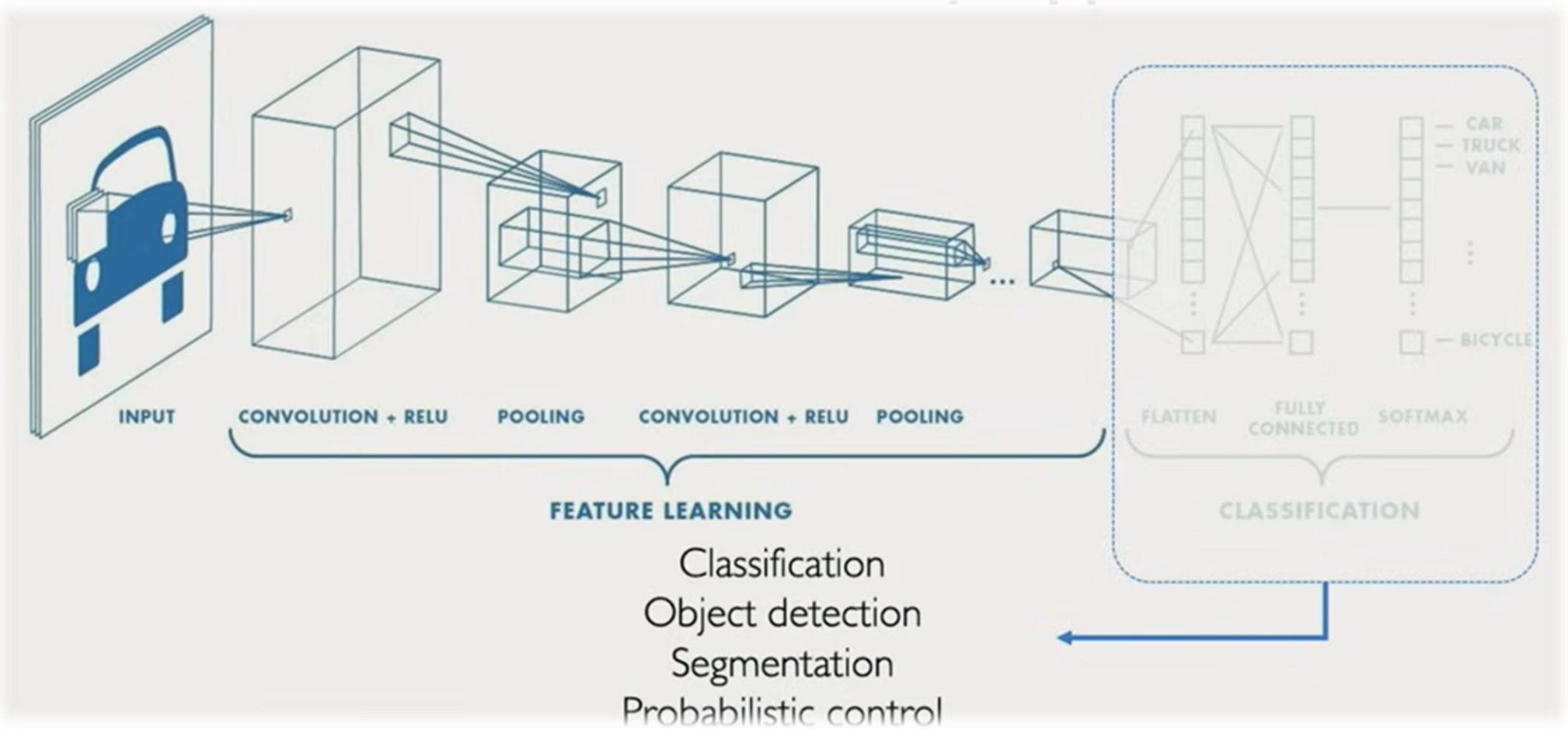
CNN for Classification: Class Probabilities



$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

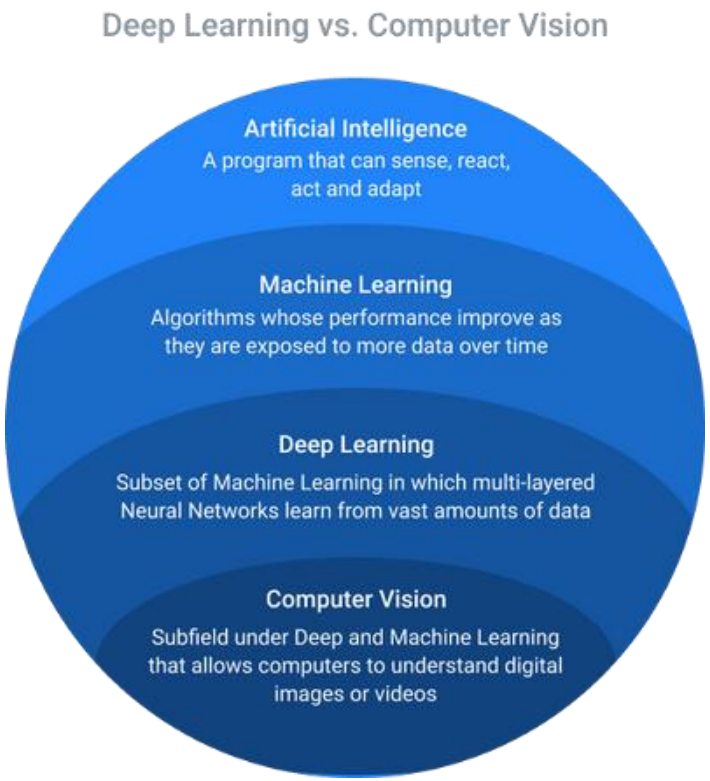
1. CONV and POOL layers output high-level features of input
2. Fully connected layer uses these features for classifying input image
3. Express output as probability of image belonging to a particular class

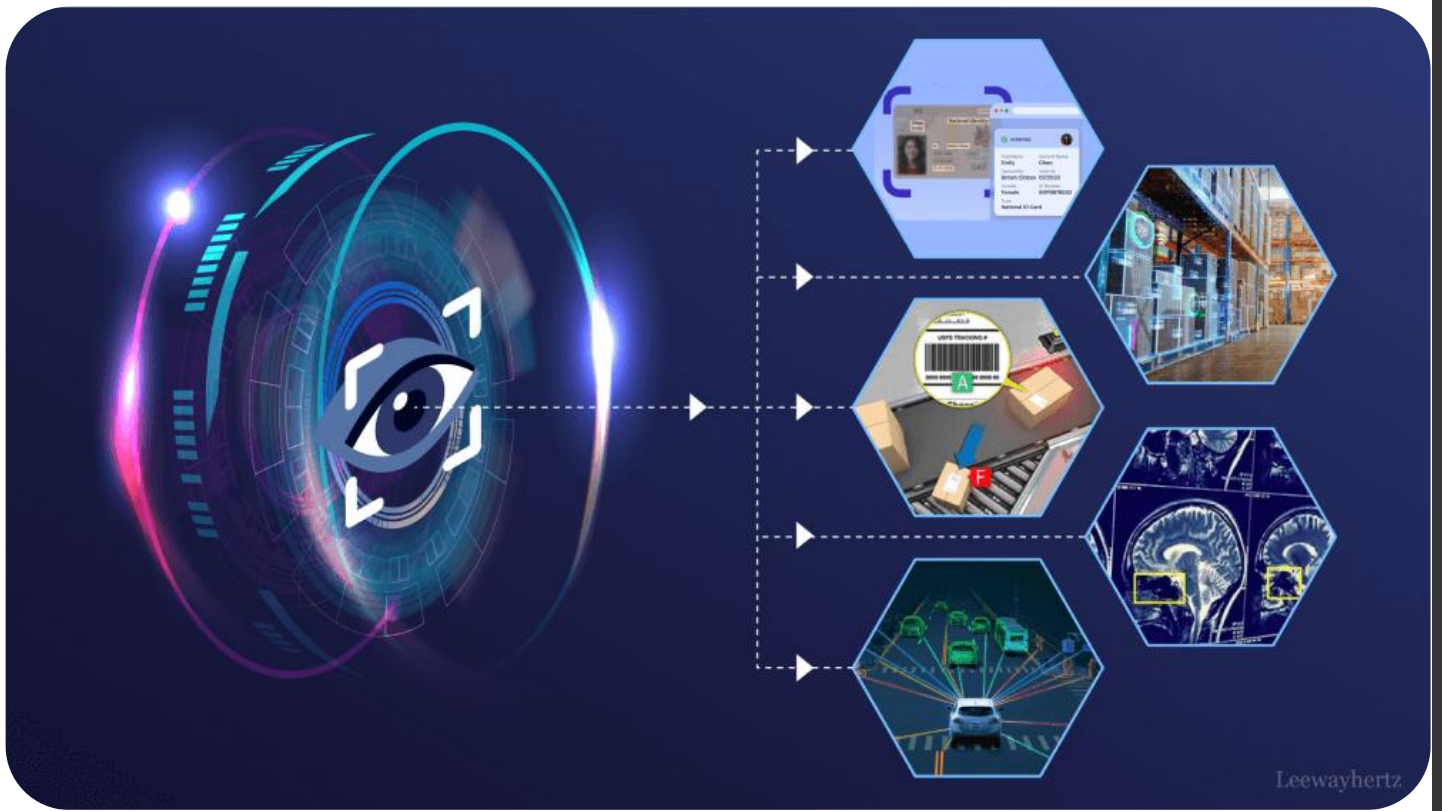
An architecture for Many Applications



Computer Vision

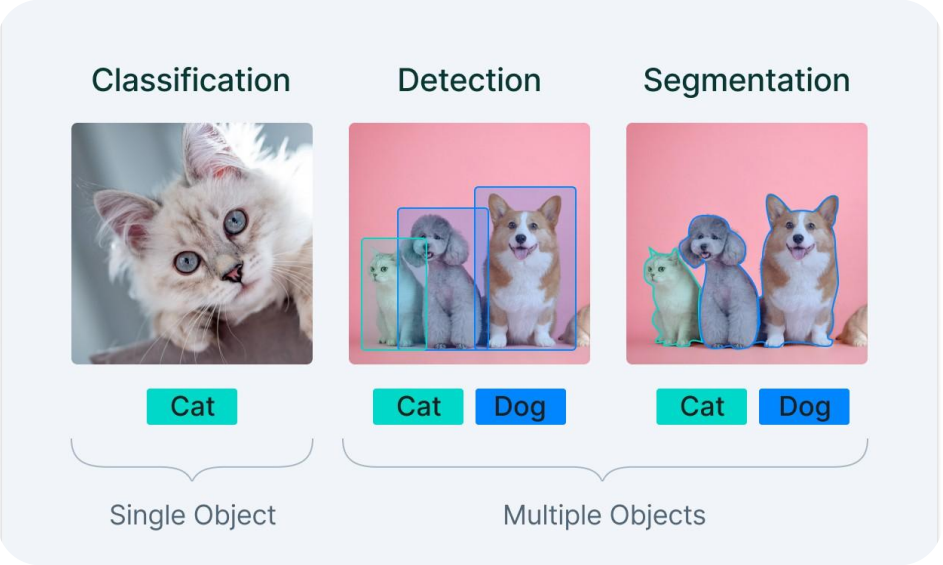
- Computer vision is focuses on enabling computers to identify and understand objects and people in images and videos.
- Computer vision seeks to perform and automate tasks that replicate human capabilities.





Common Tasks in Computer Vision

- **Image Classification** – Assigning a label to an entire image
- **Object Detection** – Identifying and locating multiple objects within an image
- **Image Segmentation** – Classifying each pixel (e.g., semantic or instance segmentation)
- **Facial Recognition** – Identifying or verifying people in images
- **Pose Estimation** – Estimating human body/keypoint positions
- **Image Captioning** – Generating descriptive text for an image
- **Image Generation** – Creating new images using generative models



Deep Learning Architectures Used in Computer Vision

Category	Purpose	Examples
CNNs	Core for image classification	LeNet, VGG, ResNet, EfficientNet
Object Detection	Locate & classify objects	YOLO, SSD, Faster R-CNN
Segmentation	Pixel-wise labeling of images	U-Net, Mask R-CNN, DeepLab
Transformers	Global attention over image patches	ViT, Swin Transformer
Generative Models	Generate or enhance images	GANs, Diffusion Models
VLMs	Vision + language tasks	CLIP, BLIP, GPT-4V

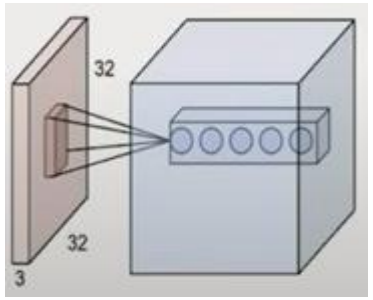
Deep Learning for Computer Vision: Summary

Foundations

- Why Computer vision?
- Representing images
- Convolutions for feature extraction

CNNs

- CNN architecture
- Application to classification
- ImageNet



Applications

- Segmentation, image captioning, control
- Security, medicine, robotics

Thank You