Computer Vision

Introduction

Lara WEHBE - TheAIEngineers - September 2024

Outline

Introduction

Introduction

Computer vision is a field of artificial intelligence (AI) that uses machine learning and neural networks to teach computers and systems to derive meaningful information from digital images, videos and other visual inputs—and to make recommendations or take actions when they see defects or issues.

Introduction

How it works?

CV needs a lot of data.
It works by analyzing the frames of an image or video.
These frames are represented as arrays of numbers.

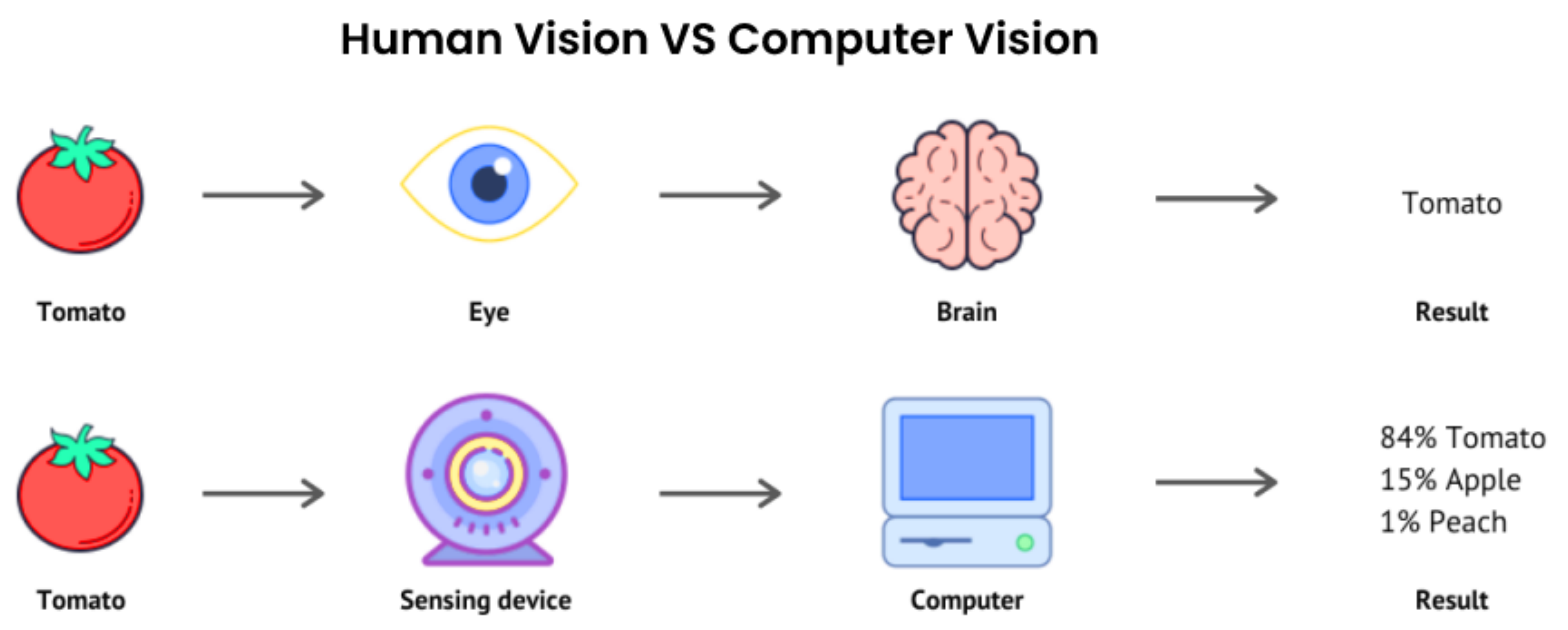


Image Representation and Processing

Image Processing
Definitions

Image processing is a method used to perform operations on images in order to enhance them or extract useful information.

Image Processing
Definitions

The applications of image processing are vast and span across many fields such as:

- **Medical Field**: Enhancing X-ray images, MRI scans, or creating 3D models of organs.
- **Remote Sensing**: Satellite image analysis for weather forecasting, agriculture, and land use mapping.
- **Machine Vision**: Used in automated inspection systems in manufacturing processes.
- **Facial Recognition**: Security systems and user authentication methods.
- **Automotive Safety**: Driver assistance systems like lane departure warnings and automatic braking systems.

Image Processing

Definitions

The history of image processing dates back to the 1920s with the development of television systems. However, it wasn’t until the invention of digital computers in the 1950s and 1960s that digital image processing began to emerge as a field of study. The first successful application was in the early 1960s, involving the enhancement of moon pictures transmitted by the Ranger 7 spacecraft. Since then, the field has grown exponentially with the advancement of computer technology, leading to sophisticated techniques and applications in various domains.

The applications of image processing are vast and span across many fields such as:

- **Medical Field**: Enhancing X-ray images, MRI scans, or creating 3D models of organs.
- **Remote Sensing**: Satellite image analysis for weather forecasting, agriculture, and land use mapping.
- **Machine Vision**: Used in automated inspection systems in manufacturing processes.
- **Facial Recognition**: Security systems and user authentication methods.
- **Automotive Safety**: Driver assistance systems like lane departure warnings and automatic braking systems.

Image Representations

Definitions

Images have been represented in different formats.

There are 4 main representations:

1. Image as a function

2. Image as Pixels

3. Image as a Cartesian Coordinate system

4. Color Images

Image Representations

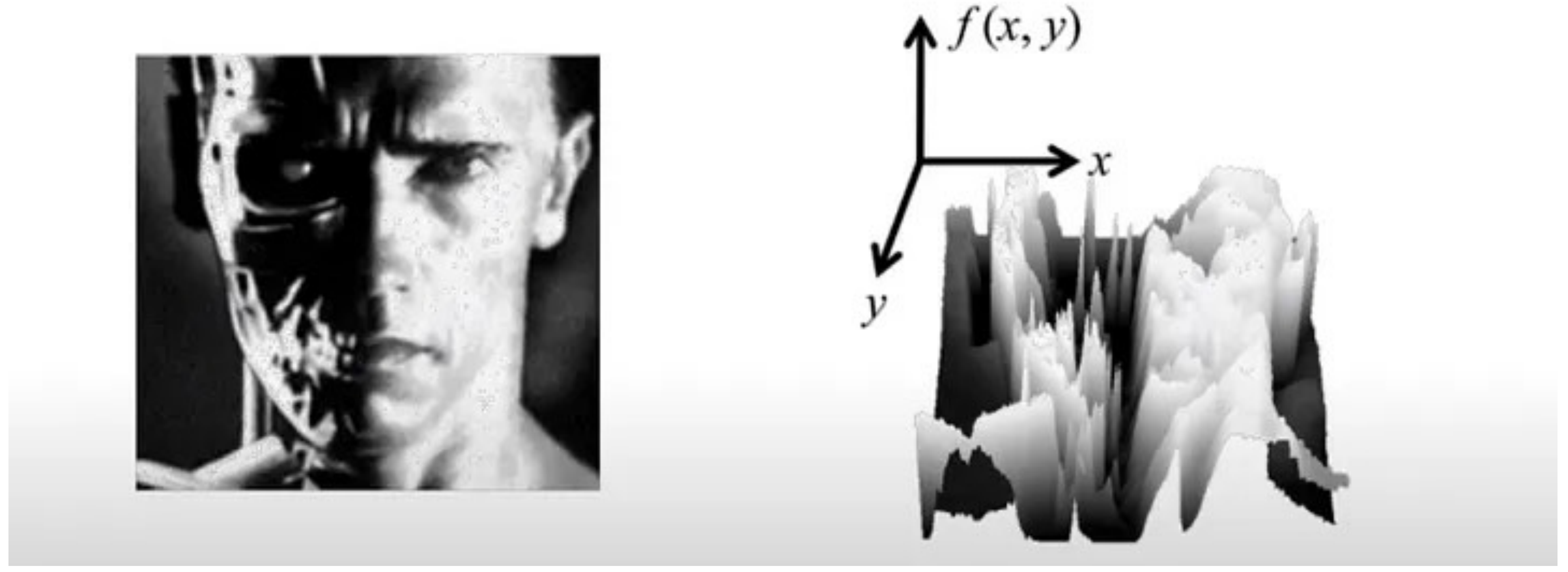
Image as a function

Image as a function:

Representing an image as a function makes it easier to apply any operation on the image.

For example, if we want to increase the image by 2, and the image is represented by a function f(x,y), hence the image increasing operation will simply be:

f(x,y) * 2.

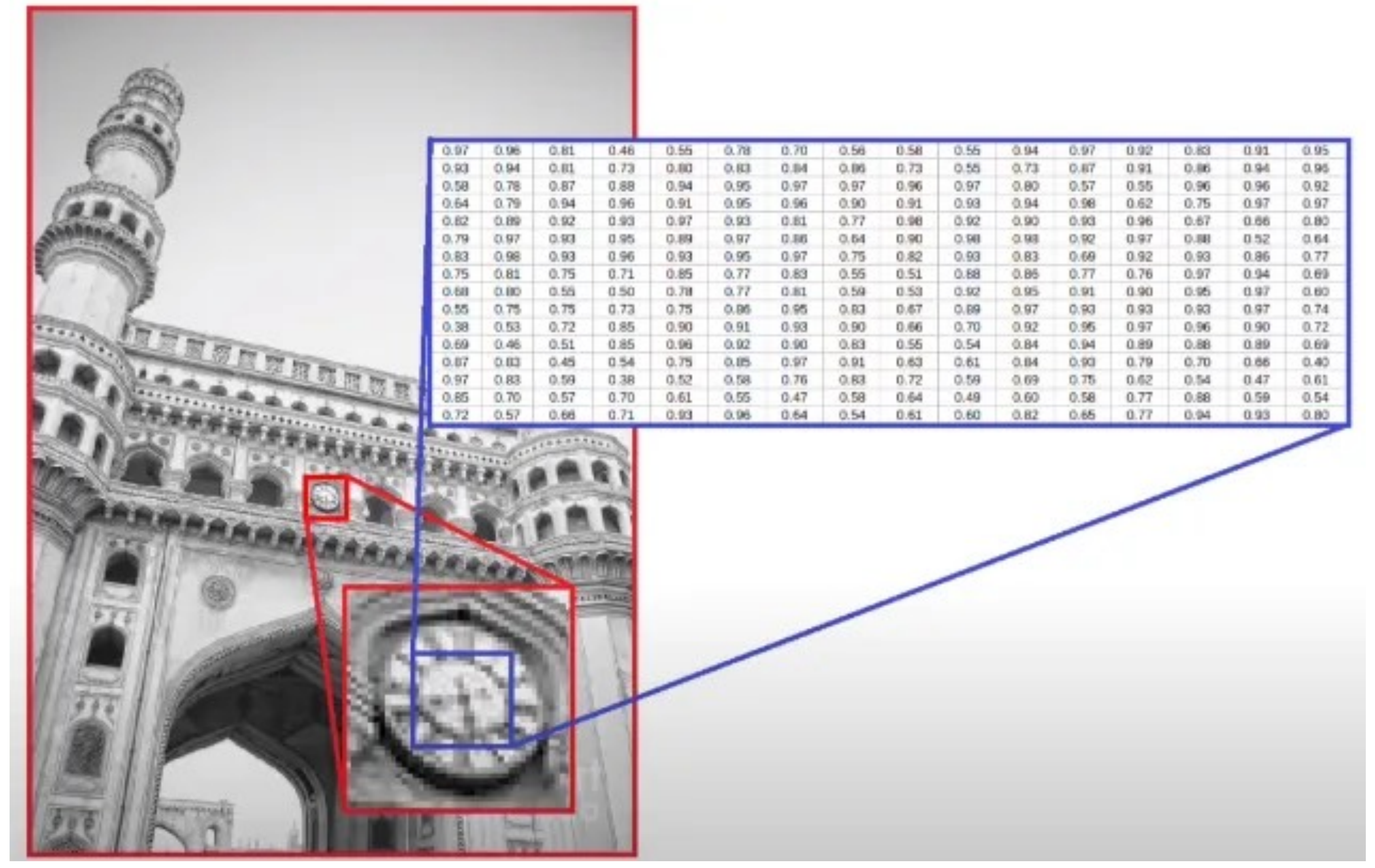


(Image credits: Noah Snavely, Cornell University)

Image Representations

Image as Pixels

Simply put, an image is represented as a matrix of pixels.



0.97	0.96	0.81	0.46	0.55	0.78	0.70	0.56	0.58	0.55	0.94	0.97	0.92	0.83	0.91	0.95
0.93	0.94	0.81	0.73	0.80	0.83	0.84	0.86	0.73	0.55	0.73	0.87	0.91	0.86	0.94	0.96
0.58	0.78	0.87	0.88	0.94	0.95	0.97	0.97	0.96	0.97	0.80	0.57	0.55	0.96	0.96	0.92
0.64	0.79	0.94	0.96	0.91	0.95	0.96	0.90	0.91	0.93	0.94	0.98	0.62	0.75	0.97	0.97
0.82	0.89	0.92	0.93	0.97	0.93	0.81	0.77	0.98	0.92	0.90	0.93	0.96	0.67	0.66	0.80
0.79	0.97	0.93	0.95	0.89	0.97	0.86	0.64	0.90	0.98	0.98	0.92	0.97	0.88	0.52	0.64
0.83	0.98	0.93	0.96	0.93	0.95	0.97	0.75	0.82	0.93	0.83	0.69	0.92	0.93	0.86	0.77
0.75	0.81	0.75	0.71	0.85	0.77	0.83	0.55	0.51	0.88	0.86	0.77	0.76	0.97	0.94	0.69
0.68	0.80	0.55	0.50	0.78	0.77	0.81	0.59	0.53	0.92	0.95	0.91	0.90	0.95	0.97	0.60
0.55	0.75	0.75	0.73	0.75	0.86	0.95	0.83	0.67	0.89	0.97	0.93	0.93	0.93	0.97	0.74
0.38	0.53	0.72	0.85	0.90	0.91	0.93	0.90	0.66	0.70	0.92	0.95	0.97	0.96	0.90	0.72
0.69	0.46	0.51	0.85	0.96	0.92	0.90	0.83	0.55	0.54	0.84	0.94	0.89	0.88	0.89	0.69
0.87	0.83	0.45	0.54	0.75	0.85	0.97	0.91	0.63	0.61	0.84	0.93	0.79	0.70	0.66	0.40
0.97	0.83	0.59	0.38	0.52	0.58	0.76	0.83	0.72	0.59	0.69	0.75	0.62	0.54	0.47	0.61
0.85	0.70	0.57	0.70	0.61	0.55	0.47	0.58	0.64	0.49	0.60	0.58	0.77	0.88	0.59	0.54
0.72	0.57	0.66	0.71	0.93	0.96	0.64	0.54	0.61	0.60	0.82	0.65	0.77	0.94	0.93	0.80

Image Representations

Image as Cartesian Coordinate System

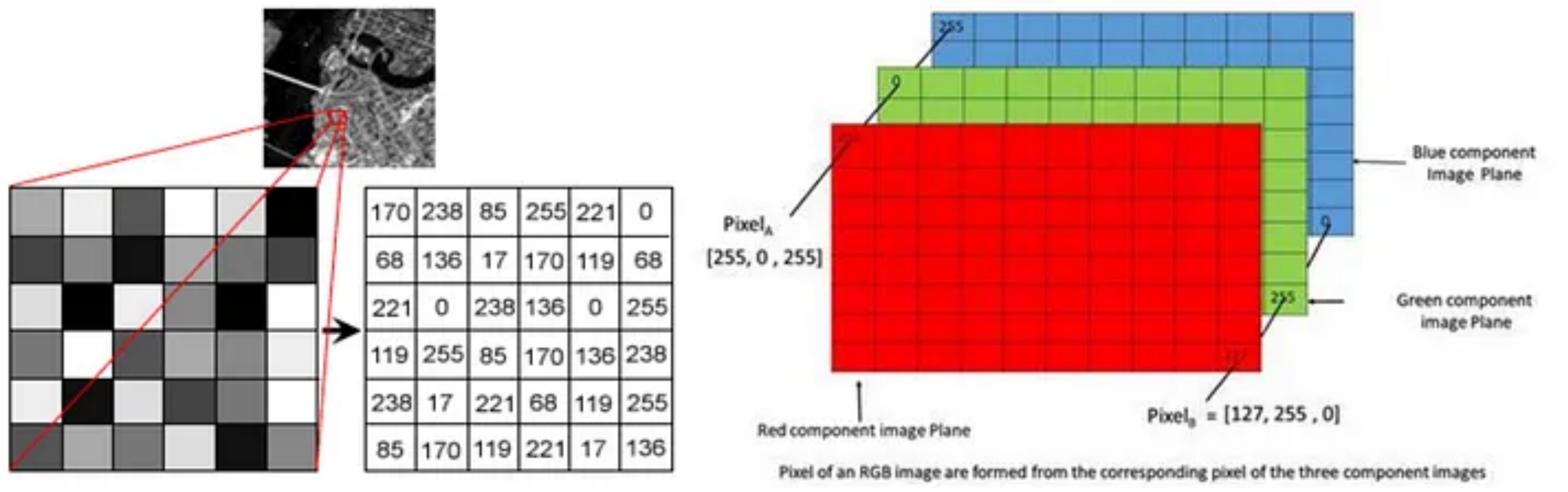
Since an image is a matrix, and the function controls how we represent the matrix by coordinates x,y (for simplicity, only x,y now), we use Cartesian Coordinate System to visualize an image.

Image Representations

Color Images

Images can belong to different color channels such as RGB or gray scale (More on this in the coming lectures).

Pixels are not singular values but triplets, representing the intensities of red, green, and blue (RGB). This trichromatic model enables the representation of a broad spectrum of colors.

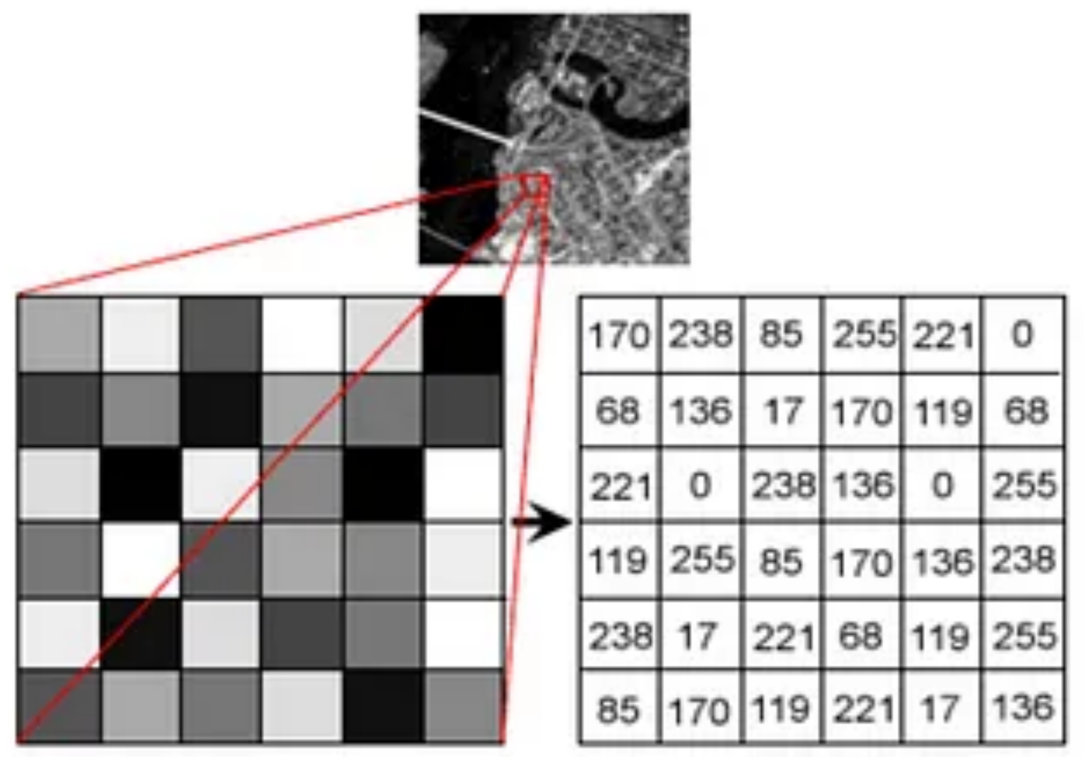

255
0
Pixel$_A$
[255, 0 , 255]
Blue component Image Plane
0
255
Green component image Plane
127
Red component image Plane
Pixel$_B$ = [127, 255 , 0]
Pixel of an RGB image are formed from the corresponding pixel of the three component images

170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

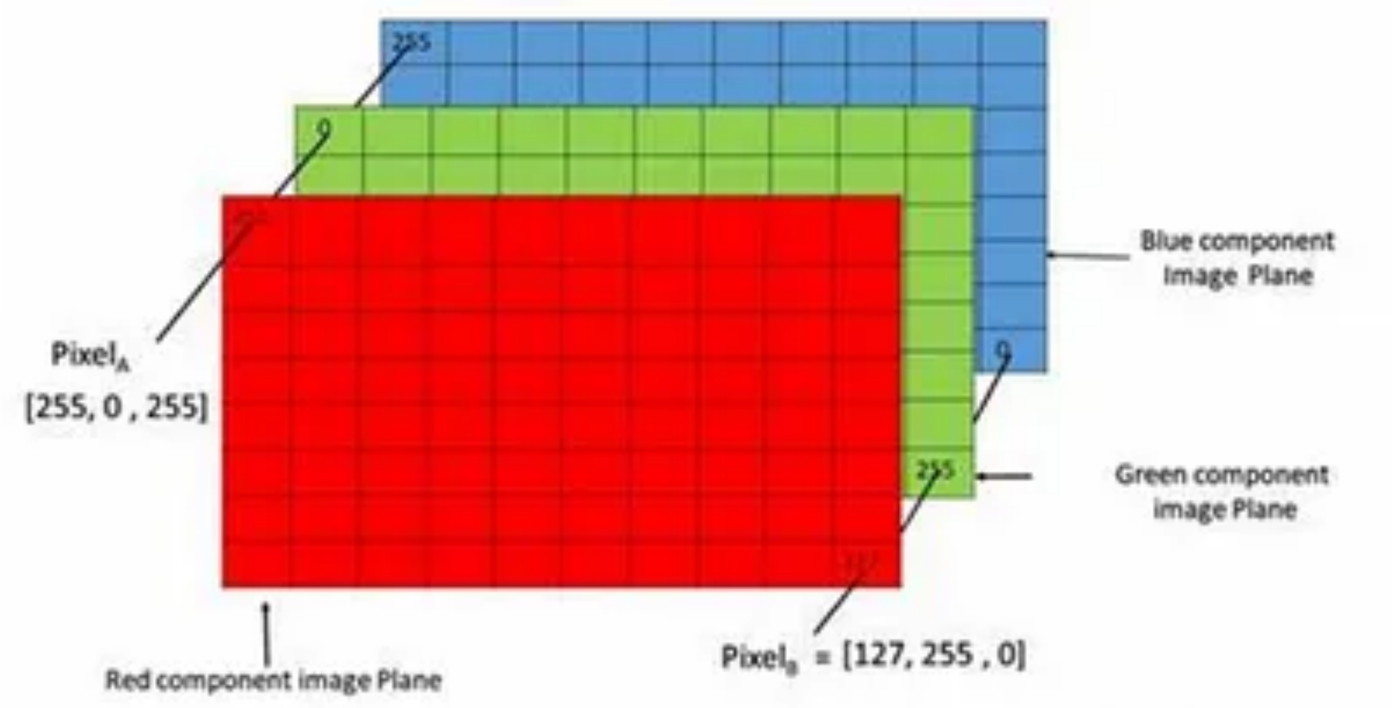
Image Representations

Color Images

- Grayscale - A pixel is an integer with a value between 0 to 255 (0 is completely black and 255 is completely white).
- RGB - A pixel is made up of 3 integers between 0 to 255 (the integers represent the intensity of red, green, and blue).
- RGBA - It is an extension of RGB with an added alpha field, which represents the opacity of the image.



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136


255
0
Blue component Image Plane
Pixel$_A$
[255, 0 , 255]
0
255
Green component image Plane
Red component image Plane
Pixel$_B$ = [127, 255 , 0]

Pixel of an RGB image are formed from the corresponding pixel of the three component images

Color Channels

Color Images

Color digital images are made of pixels, and pixels are made of combinations of primary colors represented by a series of code. A channel in this context is the grayscale image of the same size as a color image, made of just one of these primary colors.

Image Formats

Image formats play a vital role today in computer vision.

Let’s explore how many image formats we have, what they mean, and why it’s important in computer vision

Image Formats
JPEG

JPEG (Joint Photographic Experts Group):

JPEG is a widely used image format that employs lossy compression.

While it achieves smaller file sizes, some image quality is sacrificed in the compression process.

Image Formats

PNG

PNG (Portable Network Graphics):
PNG is a lossless image format known for its support for transparency. It employs a compression algorithm that preserves all the image data without compromising quality. P

Image Formats
GIF

GIF (Graphics Interchange Format):
GIF is commonly used for simple animations and low-resolution images. It employs lossless compression but is limited to a maximum of 256 colors.

Image Formats

BMP

BMP (Bitmap):
BMP is a basic and uncompressed image format that stores each pixel's color information individually. As a result, BMP files tend to have large sizes. BMP is commonly used in Windows environments but is less popular for web and digital applications due to its size.

Image Formats

TIFF

TIFF (Tagged Image File Format):
TIFF is a versatile image format that supports lossless compression. It is widely used in professional printing and publishing. TIFF files can store multiple layers, different color spaces, and a wide range of image metadata.

Image Processing

Types of Image Processing

There are five main types of image processing:

- Visualization - Find objects that are not visible in the image
- Recognition - Distinguish or detect objects in the image
- Sharpening and restoration - Create an enhanced image from the original image
- Pattern recognition - Measure the various patterns around the objects in the image
- Retrieval - Browse and search images from a large database of digital images that are similar to the original image

Image Manipulation

Resizing: Show how to resize images, maintaining aspect ratio, and explain interpolation methods (nearest neighbor, bilinear, bicubic).

Cropping: Teach students how to crop an image to focus on a specific region.

Flipping and Rotation: Introduce methods to flip and rotate images, useful for image augmentation.

Color Space Conversion: Explain different color spaces (RGB, HSV, grayscale) and how to convert between them using libraries like OpenCV.

- **Practical Demo**: Convert an image from RGB to grayscale and HSV.

Image Manipulation

3. Histogram and Image Statistics

- **Histograms: Explain how histograms represent the distribution of pixel intensities in an image.**
    - **Practical: Show how to plot histograms of images and discuss what they reveal about brightness and contrast.**
- **Histogram Equalization: Introduce techniques like histogram equalization to improve the contrast of an image.**

Image Manipulation

4. Smoothing and Sharpening Filters

- **Blurring**: Introduce basic image filters for blurring, like Gaussian blur, to reduce noise.
- **Sharpening**: Explain the concept of sharpening an image by highlighting edges using filters like the Laplacian.
  - Practical: Show examples of how blurring helps in reducing noise and how sharpening enhances image details.

Image Manipulation

Tools and Libraries:

- **OpenCV**: For implementing basic image processing operations.
- **PIL (Python Imaging Library)**: To open, manipulate, and convert image formats.

Suggested Code Examples:

- Load an image using OpenCV, resize it, crop it, and convert it to grayscale.
- Plot the histogram of an image using Matplotlib.
- Apply Gaussian blurring and a sharpening filter to see the difference.

Edge Detection and Filtering

Why do we need edge detection?

Discontinuities in depth, surface orientation, scene illumination variations, and material properties changes lead to discontinuities in image brightness. We get the set of curves that indicate the boundaries of objects and surface markings, and curves that correspond to discontinuities in surface orientation.
Thus, applying an edge detection algorithm to an image may significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant while preserving the important structural properties of an image.
As you can see in fig 1.1, the structural properties of an image are captured through edge detection.

Image Gradient

Image is a matrix of pixel values representing various intensity level values. A pixel is the building block of an image. The gradient can be defined as the change in the direction of the intensity level of an image.

Image Gradient

Let’s Compute Gradients

Consider a 3*3 Image

Find an edge by using the image gradient.

(i) P(x, y-1) top pixel
(ii) P(x+1,y) right pixel
(iii) P(x-1,y) left pixel
(iv) P(x,y+1) bottom pixel

Image Gradient

Let’s Compute Gradients

Change of intensity in the X direction is given by:

Gradient in Y direction = PR - PL

Change of intensity in the Y direction is given by:

Gradient in Y direction = PB - PT

Gradient for the image function is given by:

$\Delta I = [\delta I/\delta x, \delta I/\delta y]$

Image Gradient

Let's Compute Gradients

GX = PR - PL

Gy = PB - PT

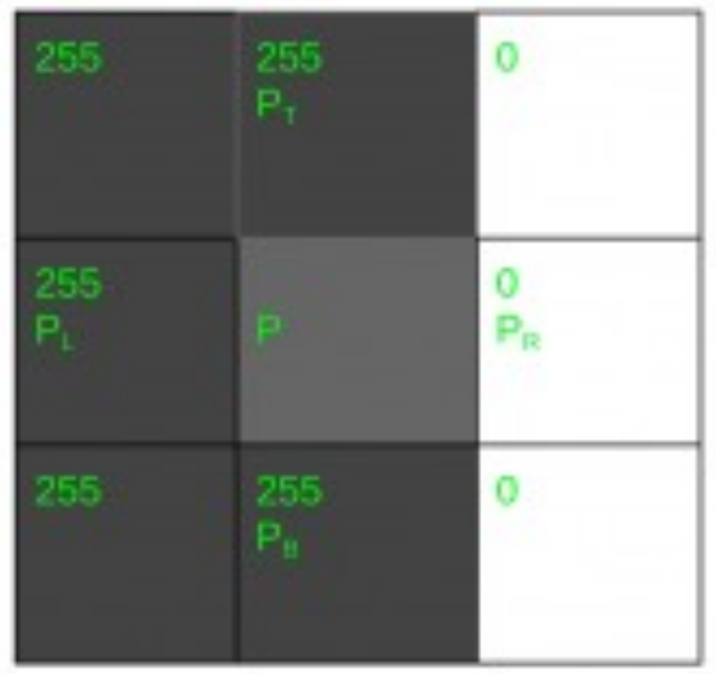


Image Gradient

Let’s Compute Gradients

GX = 0-255 = -255

Gy = 255 - 255 = 0

Gradient for this Image function will be:
$\Delta I = [\,-255, 0]$

Hence, the image is changing in the X direction, then there is a filter in the X direction as well

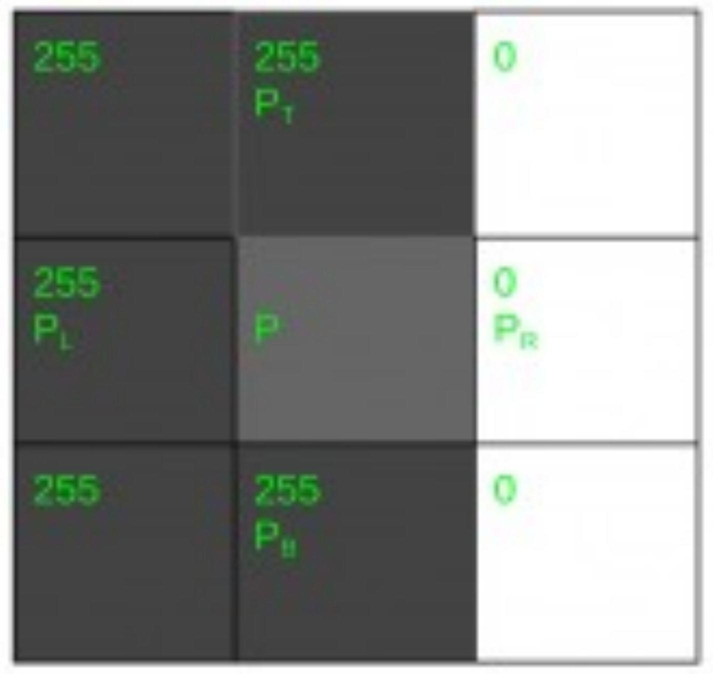


Image Gradient

Let's Compute Gradients

Now that we have found the gradient values, let me introduce you to two new terms:

- Gradient magnitude
- Gradient orientation

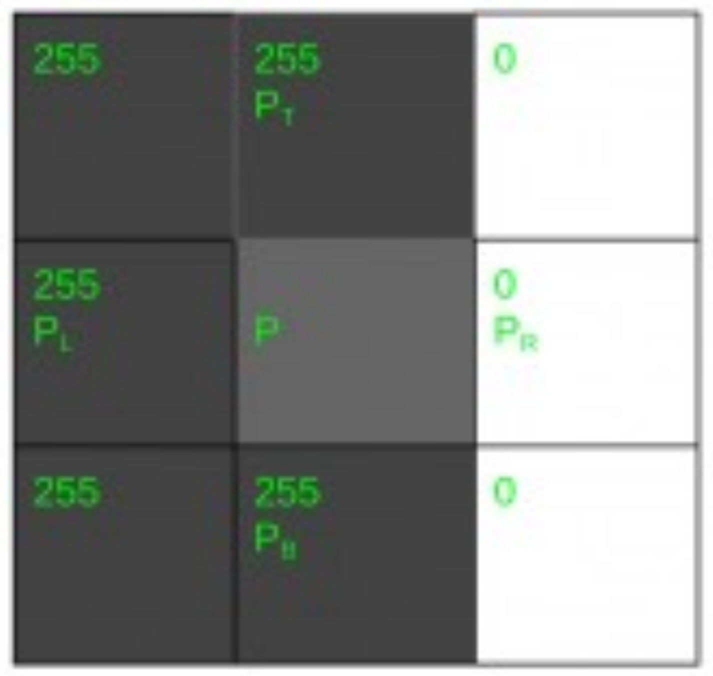


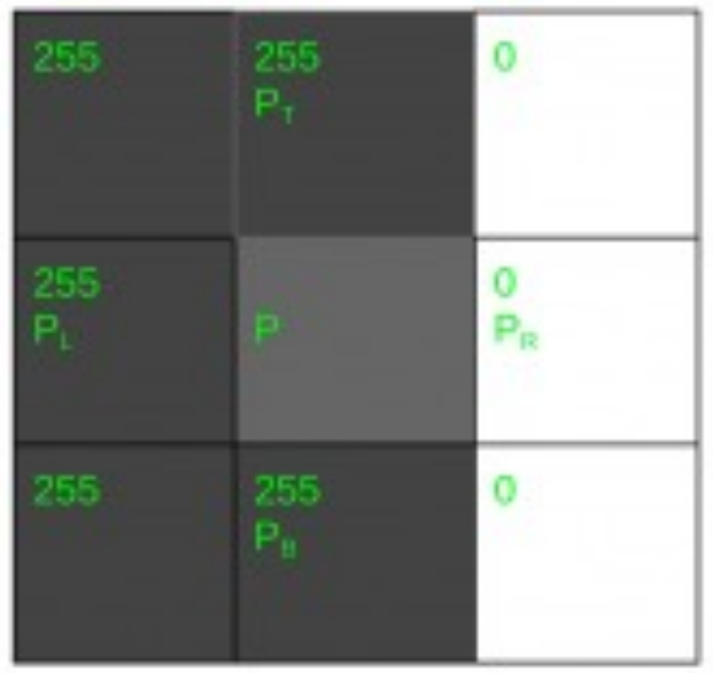
Image Gradient

Let’s Compute Gradients

Gradient magnitude represents the strength of the change in the intensity level of the image. It is calculated by the given formula:
Gradient Magnitude: $\sqrt{(\text{change in x})^2 + (\text{change in Y})^2}$

The higher the *Gradient magnitude*, the stronger the change in the image intensity

Gradient Orientation represents the direction of the change of intensity levels in the image. We can find out gradient orientation by the formula given below:
Gradient Orientation: $\tan^{-1}\left((\delta I/\delta y) / (\delta I/\delta x) \right) * (180/\pi)$

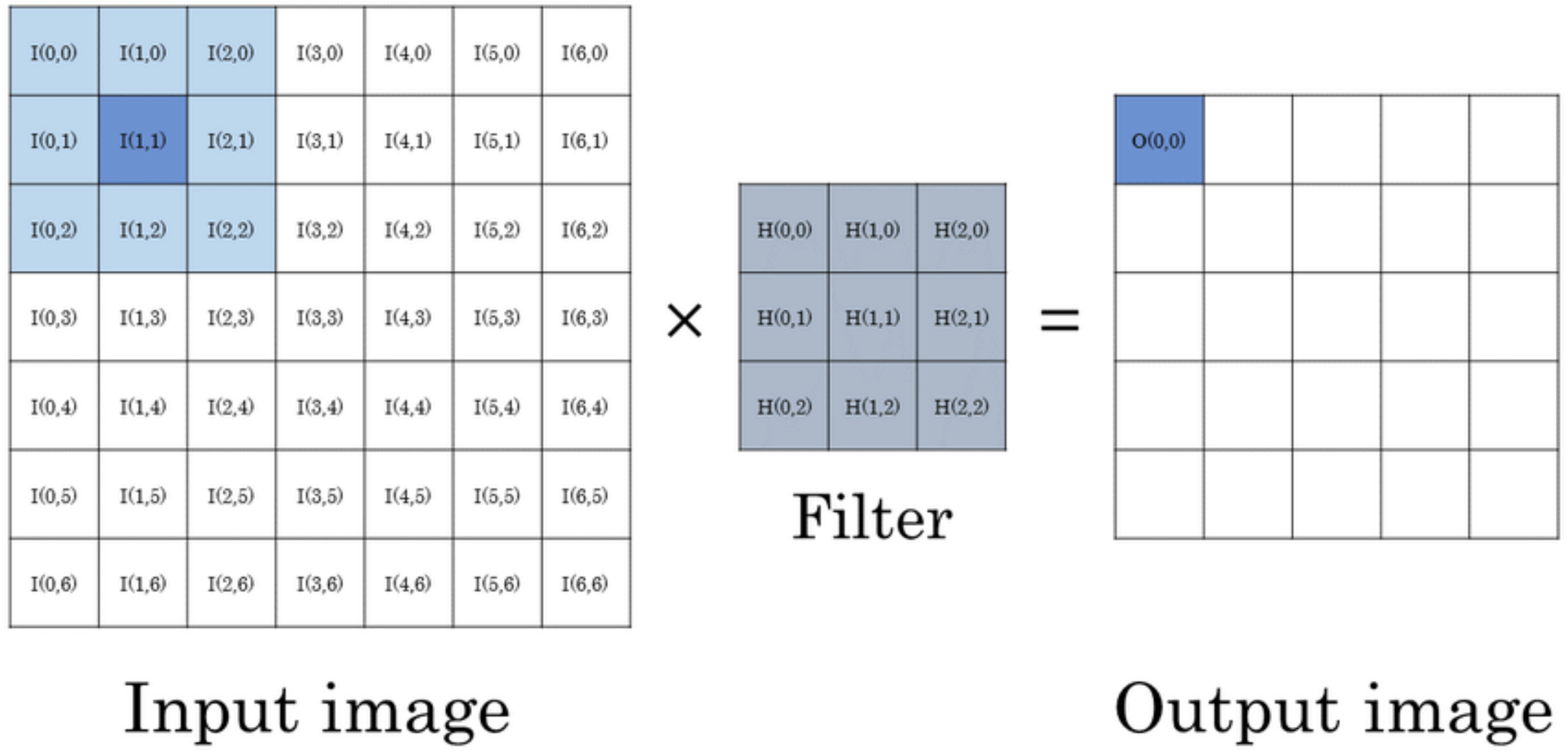


Convolution and Filter!

Let’s Compute Gradients

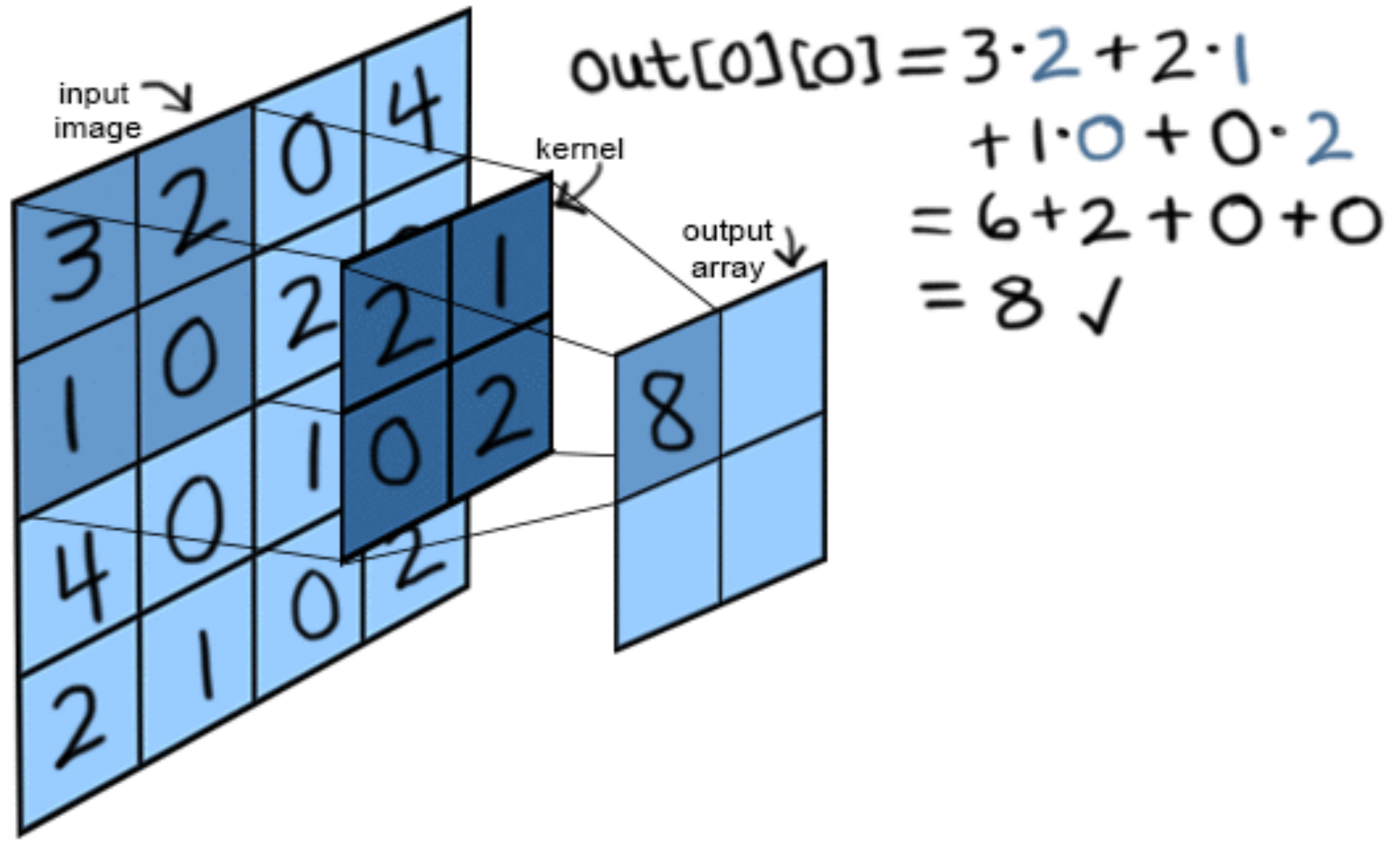
Are we going to calculate the gradients each time??
What if the image is 300*300 that is 150,000 pixels??

Here comes the convolution and the filter.



Convolution and Filter!

Let's Compute Gradients



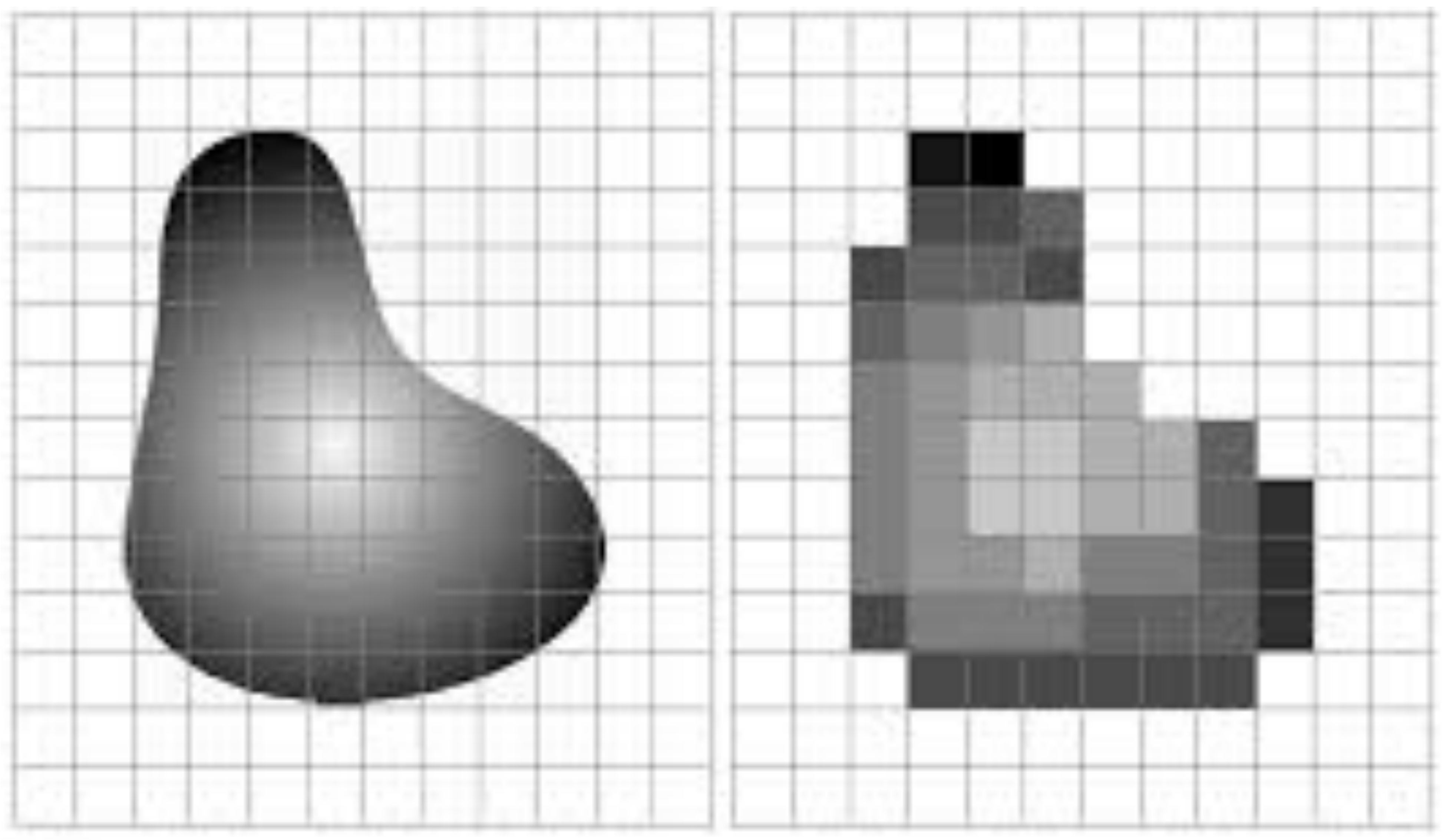
$$\text{out}[0][0] = 3\cdot 2 + 2\cdot 1 + 1\cdot 0 + 0\cdot 2 = 6+2+0+0 = 8 ✓$$

Filters

Overview

We cannot calculate gradients manually, specially that images are not as simple as an image of size 3*3.

Enter Filters.



a b

FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.

Filters

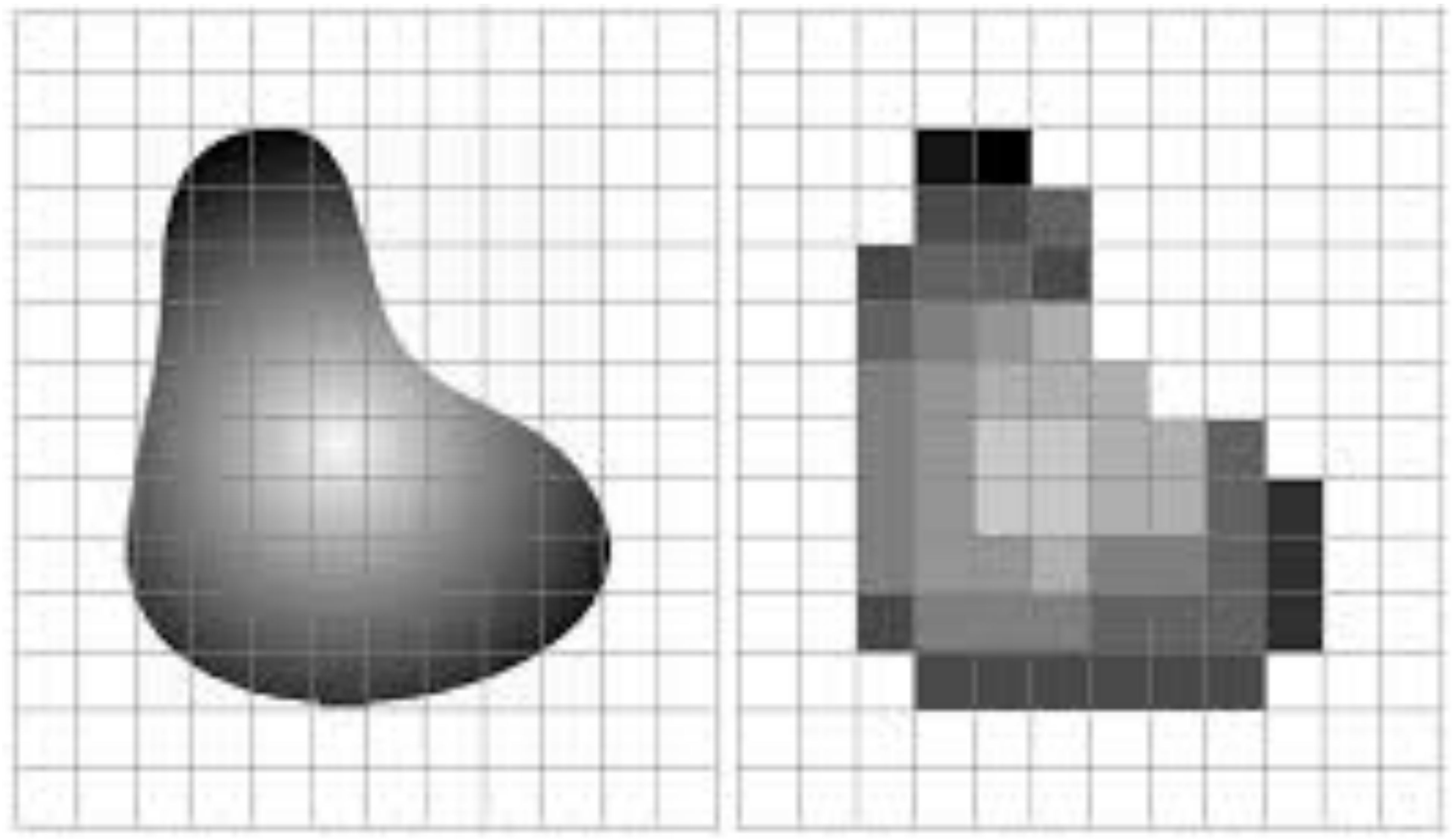
Overview

3 main types of filters:

1. Roberts Filter

2. Prewitt Filter

3. Sobel Filter



a b

FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.

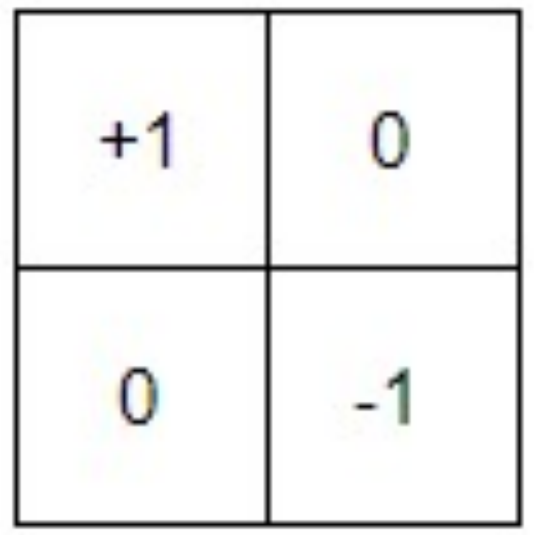
Roberts Filter

Definition

Roberts Filter is a 2*2 filter.

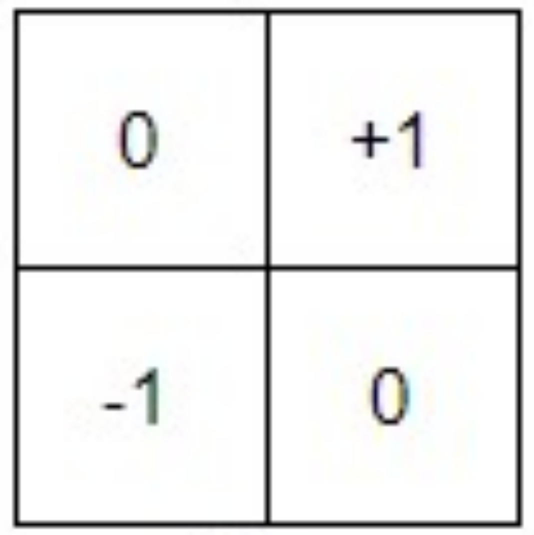
Gx = 100 *1 + 200*0 + 150*0 - 35*1

Gx = 65



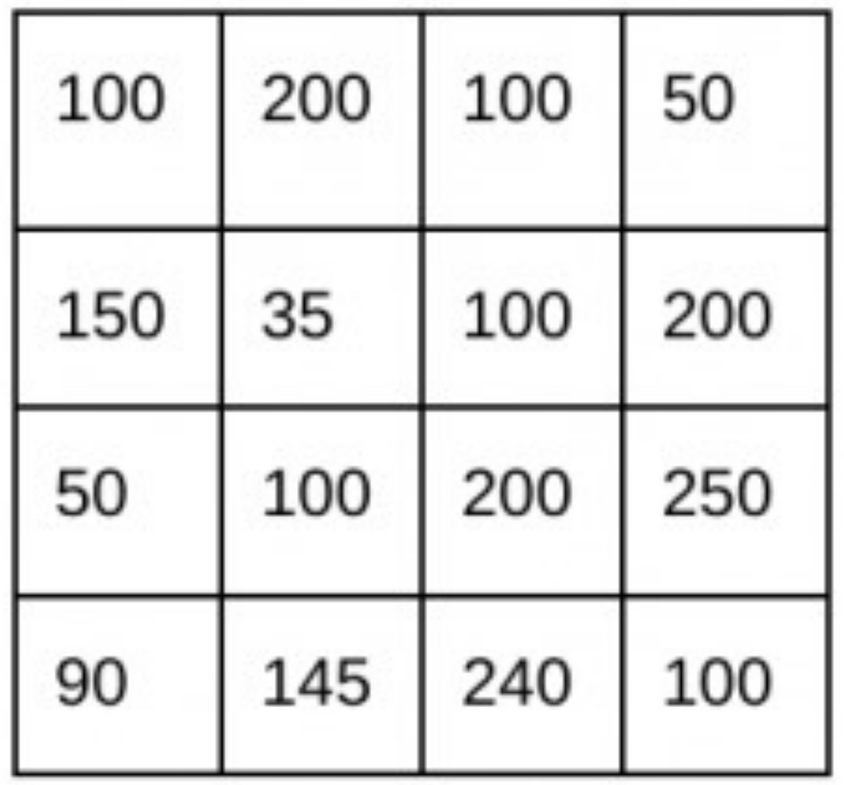
+1	0
0	-1

Gx



0	+1
-1	0

Gy

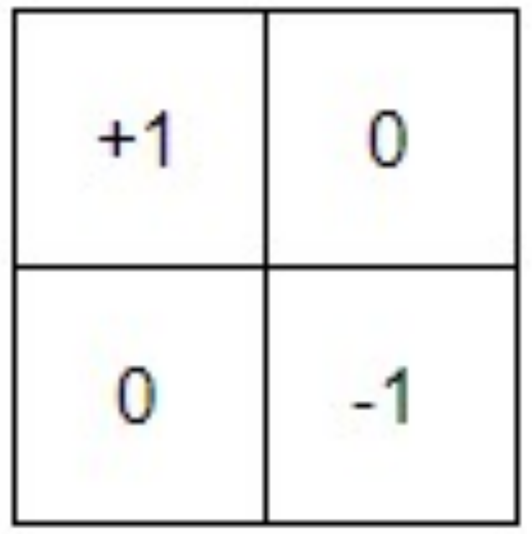


100	200	100	50
150	35	100	200
50	100	200	250
90	145	240	100

Roberts Filter

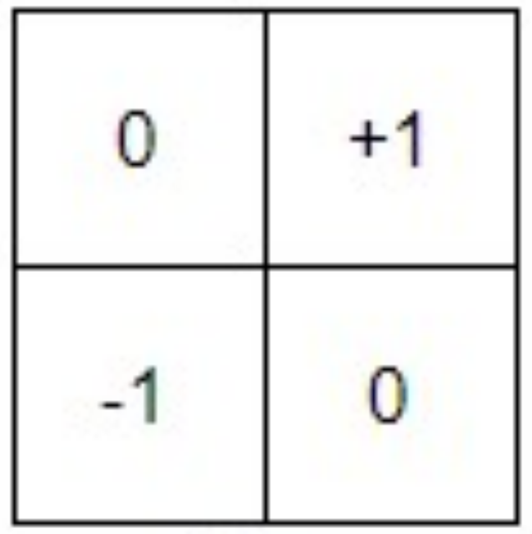
Definition

Roberts Filter is a 2*2 filter.



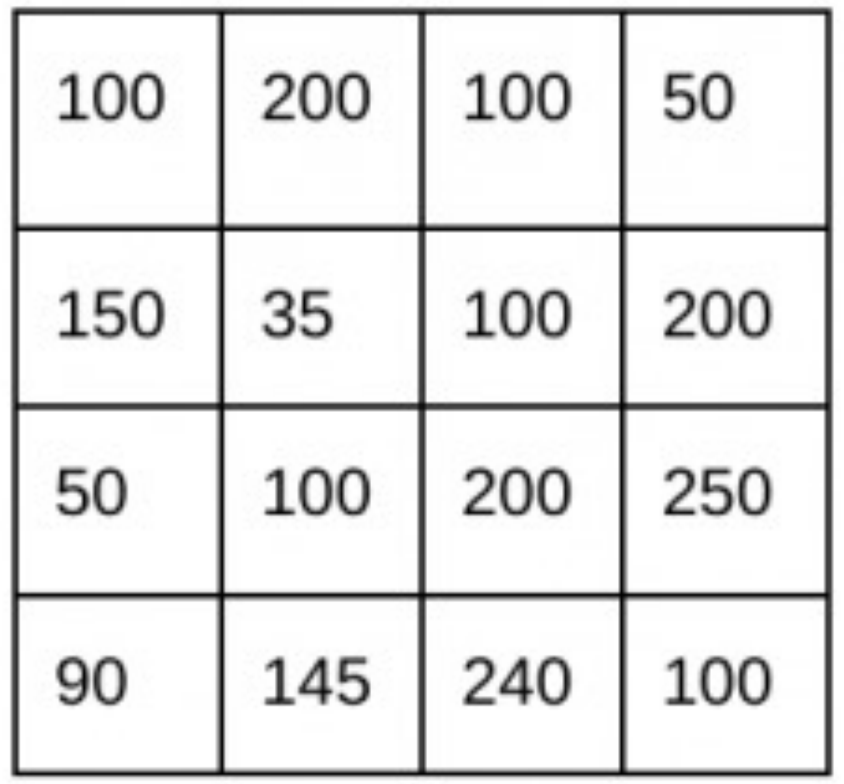
+1	0
0	-1

Gx

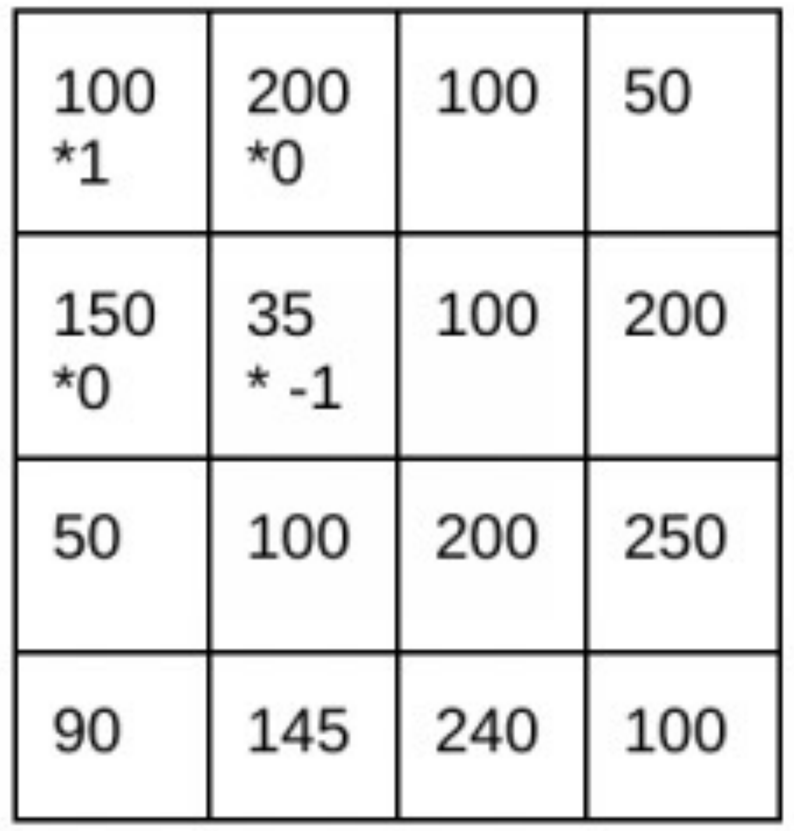


0	+1
-1	0

Gy



100	200	100	50
150	35	100	200
50	100	200	250
90	145	240	100



100 *1	200 *0	100	50
150 *0	35 * -1	100	200
50	100	200	250
90	145	240	100

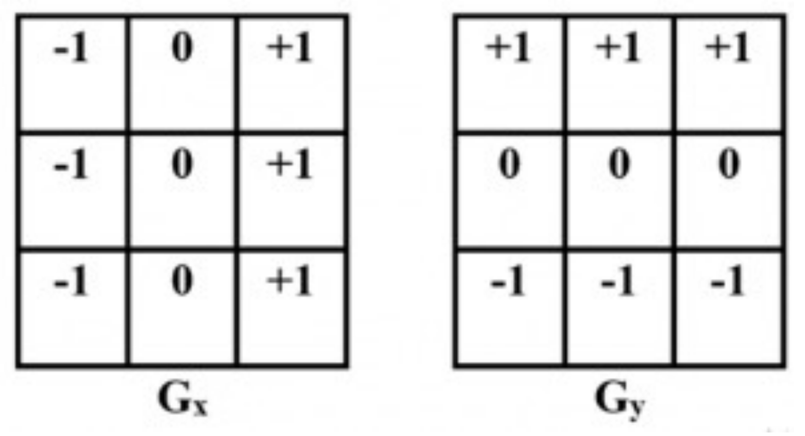
Prewitt Filter

Definition

Prewitt Filter is a 3*3 filter.

Gx = 100 *(-1) + 200*0 + 100*1 + 150*(-1) + 35*0 + 100*1 + 50*(-1) + 100*0 + 200*1

Gx = 100

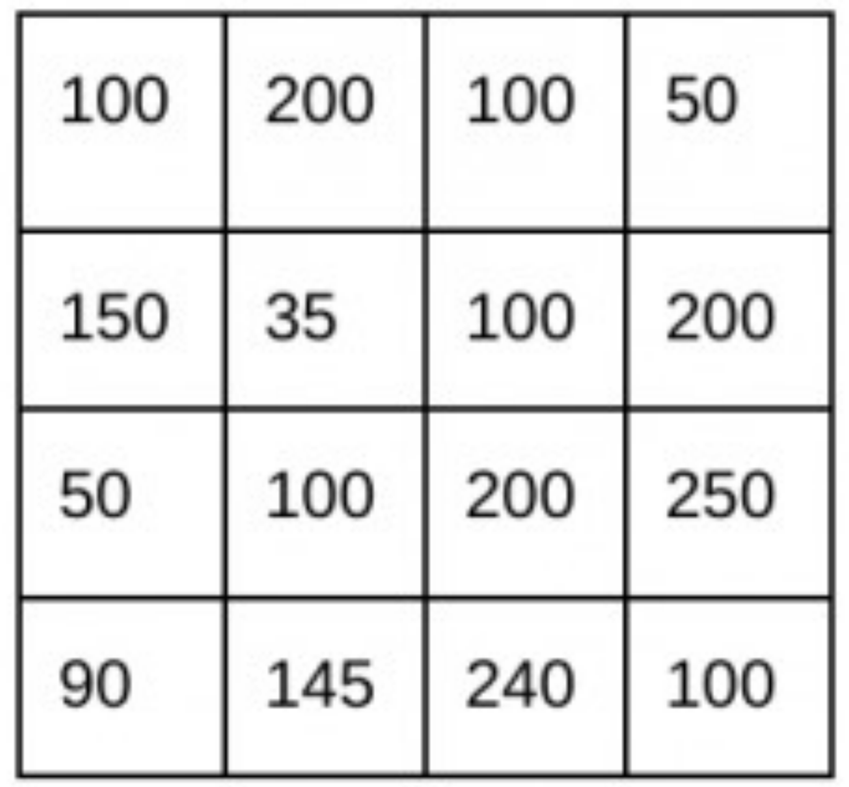


-1	0	+1
-1	0	+1
-1	0	+1

G_x

+1	+1	+1
0	0	0
-1	-1	-1

G_y

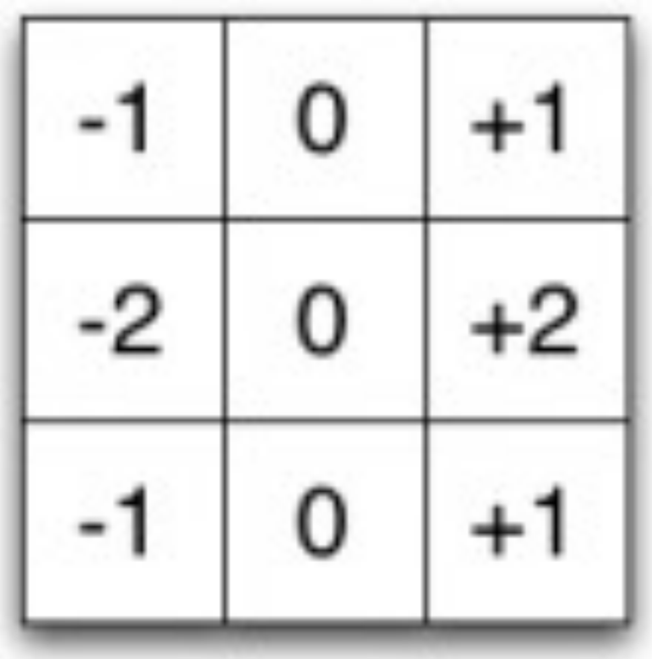


100	200	100	50
150	35	100	200
50	100	200	250
90	145	240	100

Sobel Filter

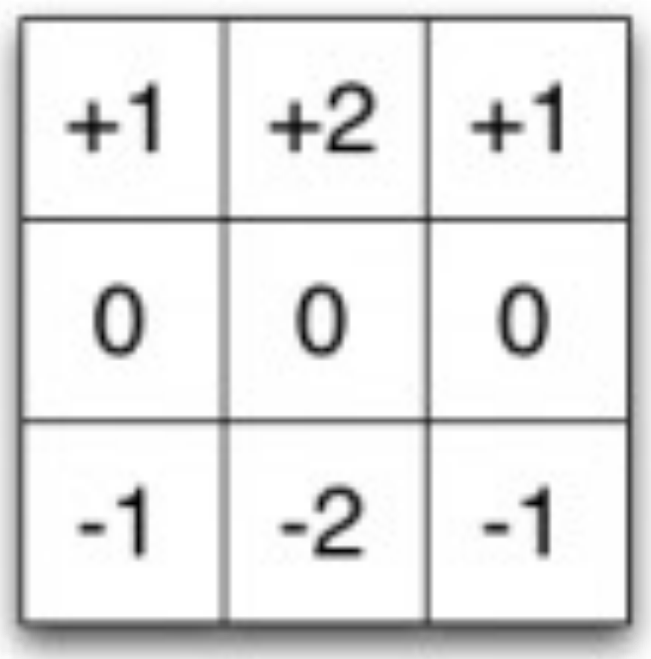
Definition

Sobel Filter is a 3*3 filter like Prewitt Filter, just the center 2 values are changed from 1 to 2 and -1 to -2 in both the filters used for horizontal and vertical edge detection.



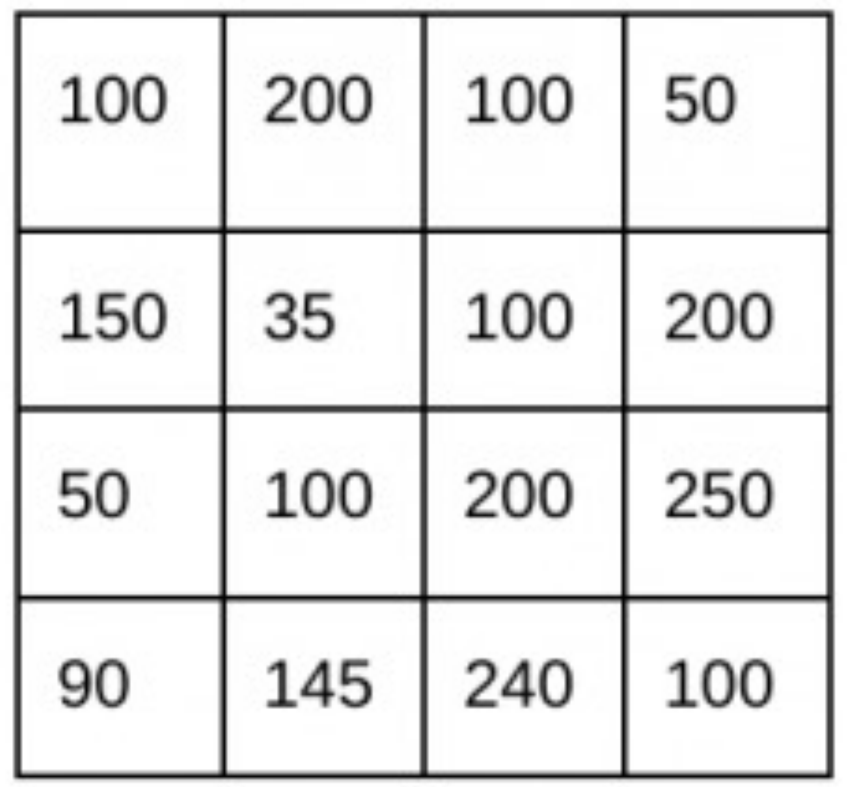
-1	0	+1
-2	0	+2
-1	0	+1

x filter



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

y filter



100	200	100	50
150	35	100	200
50	100	200	250
90	145	240	100

Sobel Filter

Definition

Coding now

Canny Edge Detector

Definition

The Canny Edge Detection algorithm is a widely used edge detection algorithm in today's image processing applications.

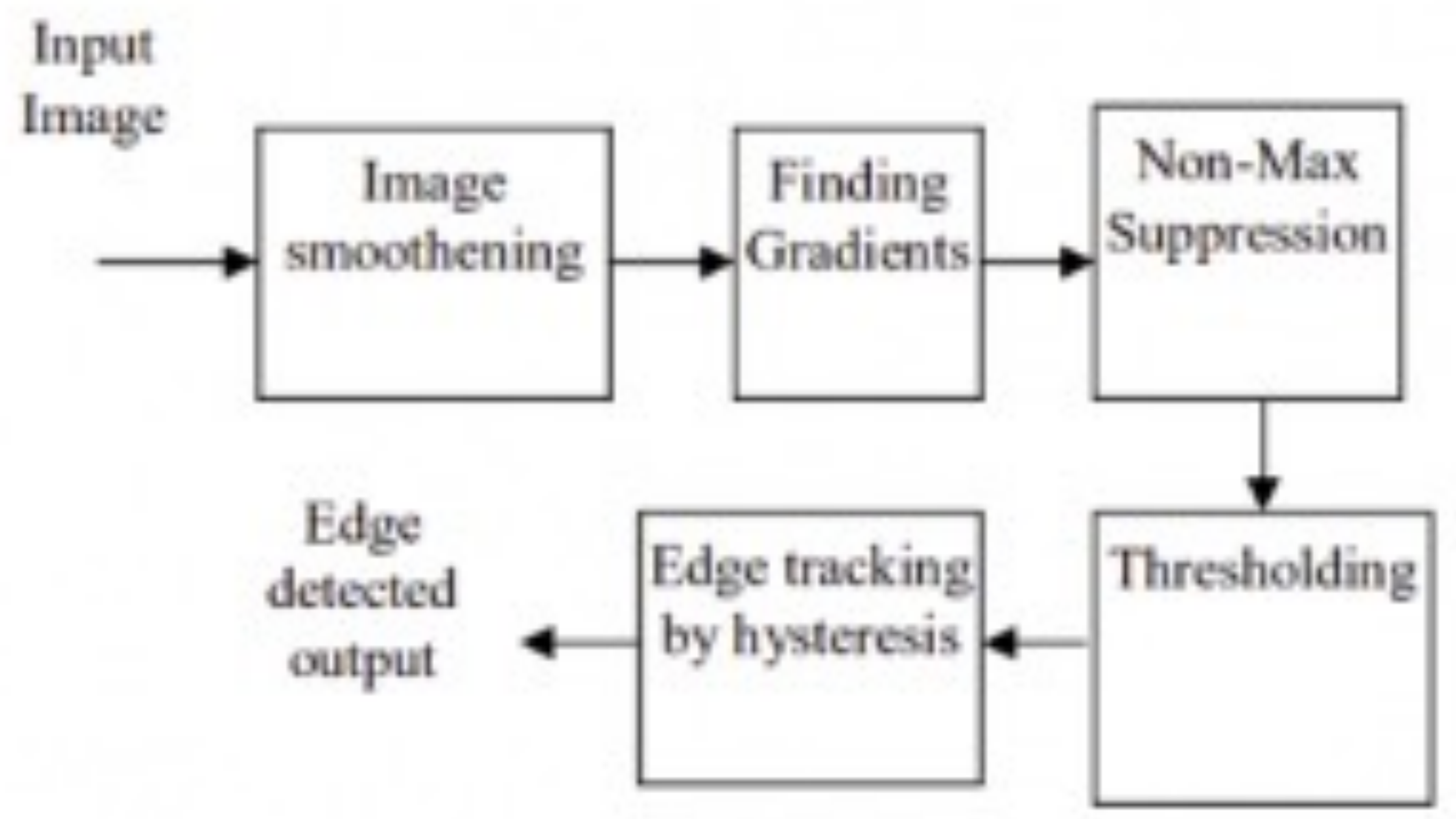


Fig 1.2: Canny Edge Detection

Feature Detection and Matching

Features

Definition

A feature is a piece of information which is relevant for solving the computational task related to a certain application.

They can be in two categories:

1. keypoint features: The features that are in specific locations of the images, such as mountain peaks, building corners, doorways, or interestingly shaped patches of snow. (Localized Features)

2. Edge Features: The features that can be matched based on their orientation and local appearance (edge profiles) are called **edges** and they can also be good indicators of object boundaries and occlusion events in the image sequence.

Feature Detection

Applications

Application Of Feature Detection And Matching

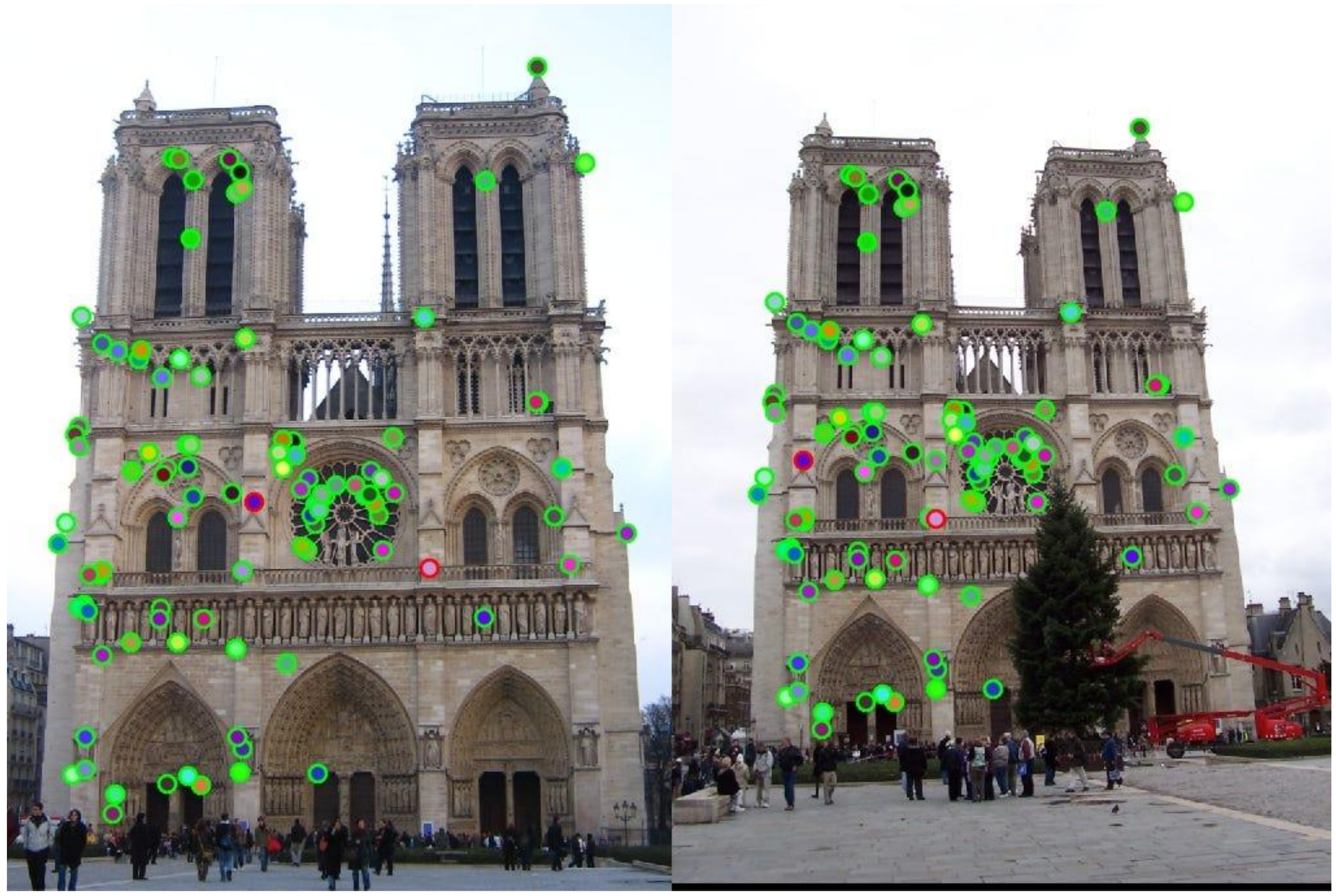
- Automate object tracking
- Point matching for computing disparity
- Stereo calibration(Estimation of the fundamental matrix)
- Motion-based segmentation
- Recognition
- 3D object reconstruction
- Robot navigation
- Image retrieval and indexing

Main Component Of Feature Detection And Matching

- **Detection:** Identify the **Interest Point**
- **Description:** The local appearance around each feature point is described in some way that is (ideally) invariant under changes in illumination, translation, scale, and in-plane rotation. We typically end up with a descriptor vector for each feature point.
- **Matching:** Descriptors are compared across the images, to identify similar features. For two images we may get a set of pairs (***Xi, Yi***) ↔ (***Xi`, Yi`***), where (***Xi, Yi***) is a feature in one image and (***Xi`, Yi`***) its matching feature in the other image.

Main Component Of Feature Detection And Matching

Interest point or Feature Point is the point which is expressive in texture. Interest point is the point at which the direction of the boundary of the object changes abruptly or intersection point between two or more edge segments.



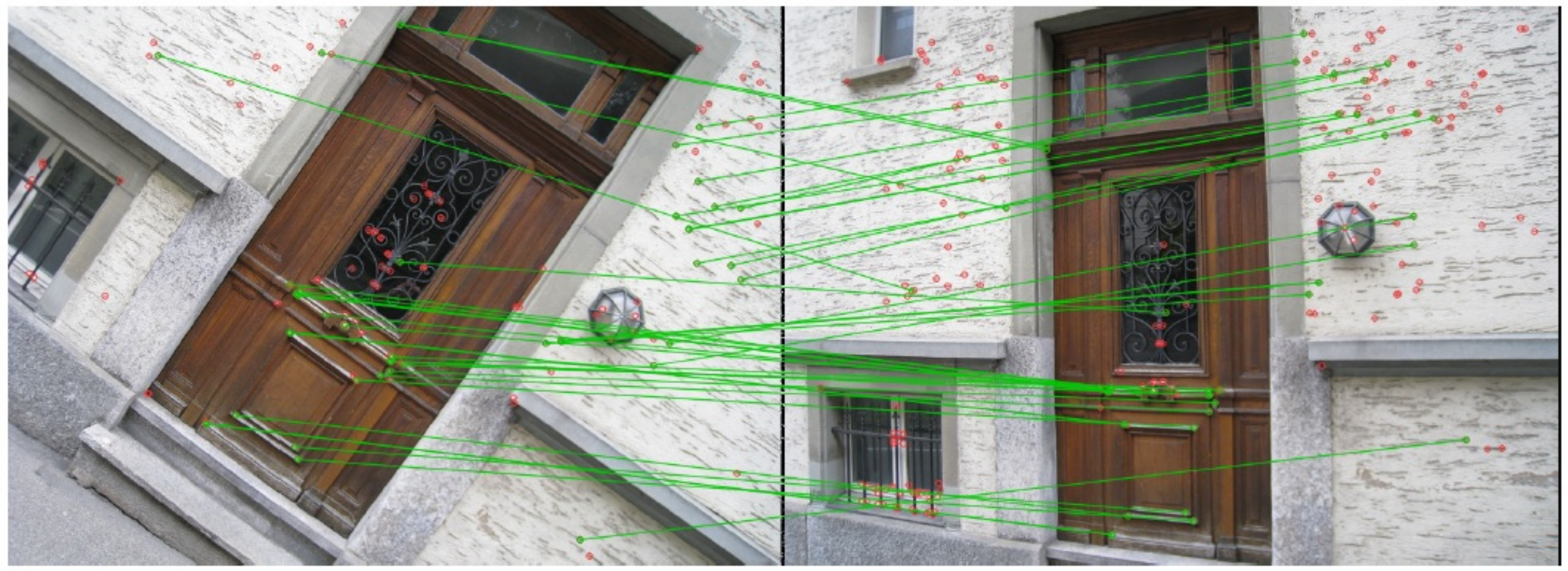
Keypoints and Descriptors

Definition

Keypoints and descriptors are essential in computer vision for detecting and describing unique features in images.

Keypoints are distinct points or regions in an image that are invariant to scale, rotation, and illumination.

Descriptors provide a numerical representation of these keypoints, enabling the comparison of features between images.



Feature Descriptors

A feature descriptor is an algorithm which takes an image and outputs feature descriptors/feature vectors. Feature Descriptors can be categorized into:

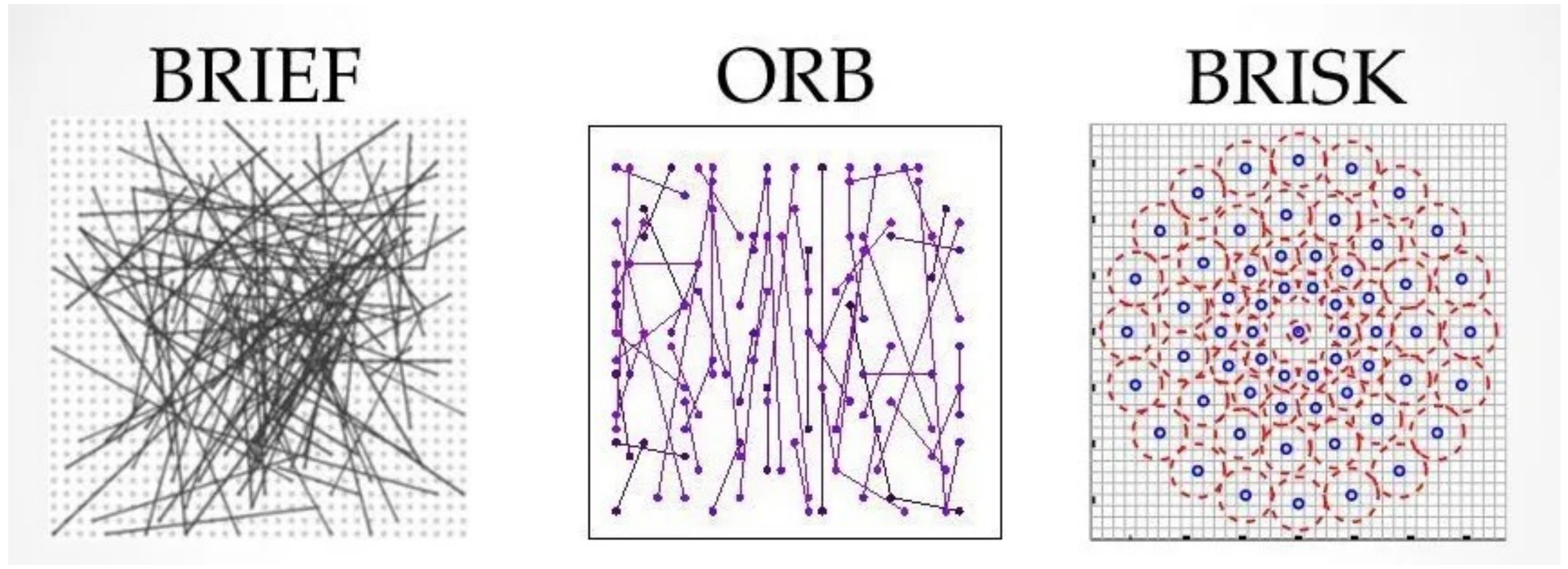
- **Local Descriptor:** It is a compact representation of a point's local neighborhood. Local descriptors try to resemble shape and appearance only in a local neighborhood around a point and thus are very suitable for representing it in terms of matching.

- **Global Descriptor**: A global descriptor describes the whole image. They are generally not very robust as a change in part of the image may cause it to fail as it will affect the resulting descriptor.

Descriptors Algorithms

Algorithms

- SIFT(Scale Invariant Feature Transform)
- SURF(Speeded Up Robust Feature)
- BRISK (Binary Robust Invariant Scalable Keypoints)
- BRIEF (Binary Robust Independent Elementary Features)
- ORB(Oriented FAST and Rotated BRIEF)



SIFT

Algorithm Explanation

SIFT (Scale-Invariant Feature Transform) works by detecting keypoints and computing descriptors in images, making it scale and rotation invariant. It involves:

1. **Keypoint Detection**: The algorithm uses Gaussian filters to detect distinctive points at different scales by computing differences between images.

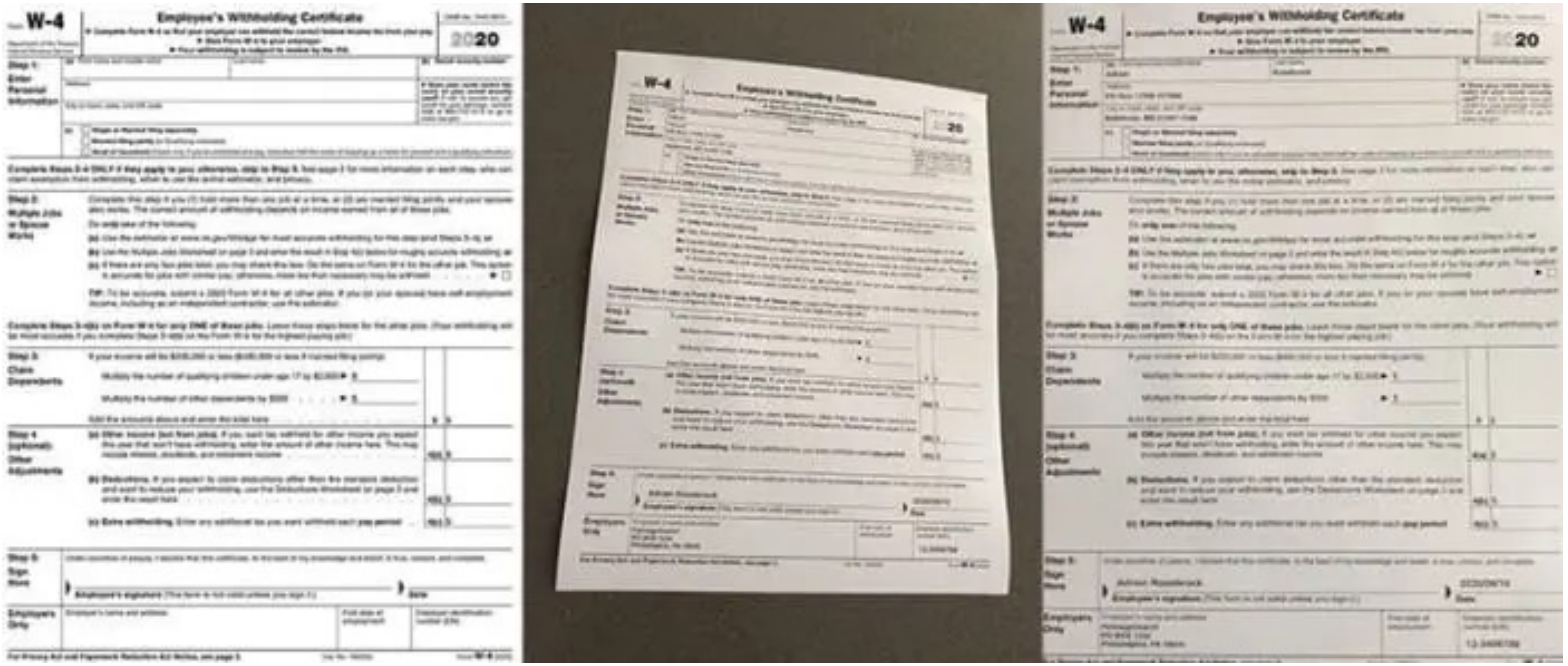
2. **Orientation Assignment**: Each keypoint is assigned a dominant orientation based on local image gradients.

3. **Descriptor Computation**: Descriptors are calculated by forming histograms of gradient orientations within neighborhoods around each keypoint, producing a feature vector for comparison.

Alignment Algorithms

The most popular image alignment algorithms are **feature-based** and include keypoint detectors (DoG, Harris, GFFT, etc.), local invariant descriptors (SIFT, SURF, ORB, etc.), and keypoint matching (RANSAC and its variants).

Image Matching Example


W-4
Employee's Withholding Certificate
2020
W-4
Employee's Withholding Certificate
2020
W-4
Employee's Withholding Certificate
2020

Detect Features

In order to detect features, we need to follow three steps:

1. Detect keypoints and extract descriptors
2. Match keypoints
3. Estimate homography

Keypoints and Descriptors

Keypoints and Descriptors

- **Keypoints**: Explain that keypoints are distinctive locations in an image, such as corners or blobs, where feature extraction occurs.
- **Descriptors**: Once keypoints are identified, descriptors provide a description of the local image structure around each keypoint, which can be used for matching.
- Popular Keypoint Detectors and Descriptors:
    - **SIFT (Scale-Invariant Feature Transform)**: Explain that SIFT detects keypoints at different scales and orientations, making it robust to image transformations like rotation and scaling.
    - **SURF (Speeded Up Robust Features)**: Similar to SIFT, but optimized for speed, making it faster and more efficient.
    - **ORB (Oriented FAST and Rotated BRIEF)**: ORB is an open-source alternative to SIFT and SURF, using FAST for keypoint detection and BRIEF for descriptors. It is faster and suitable for real-time applications.
- **Practical Example**: Detect and describe keypoints using SIFT, SURF, and ORB with OpenCV's `cv2.SIFT_create()`, `cv2.SURF_create()`, and `cv2.ORB_create()` functions.

Feature Matching

. Feature Matching

- **Brute Force Matching**: Explain brute force matching, where descriptors from one image are compared to descriptors from another using a distance metric (e.g., Euclidean distance). The closest matches are returned.
- **FLANN (Fast Library for Approximate Nearest Neighbors)**: Introduce FLANN as an efficient, approximate method for feature matching, especially for large datasets.
  Practical Example: Implement brute force matching and FLANN matching in OpenCV using `cv2.BFMatcher()`and `cv2.FlannBasedMatcher()`.

Applications of Feature Matching

- Image Stitching:
  - Explain how feature matching is used in applications like panoramic image stitching. Keypoints are detected in overlapping regions of multiple images, and the images are aligned using matched features.
  - **Practical Example**: Perform image stitching by detecting and matching keypoints between two images, then blending them to create a panorama.
- Object Recognition:
  - Feature matching is often used in object recognition, where keypoints from a known object are matched with keypoints in a scene to detect and locate the object.

Practical Part

Suggested Code Examples:

- Detect keypoints and compute descriptors using SIFT, SURF, and ORB in OpenCV.
- Match features using brute force matching and FLANN.
- Implement an image stitching example using feature matching.

Object Detection and Recognition

Video Basics with Python and OpenCV

We will learn:

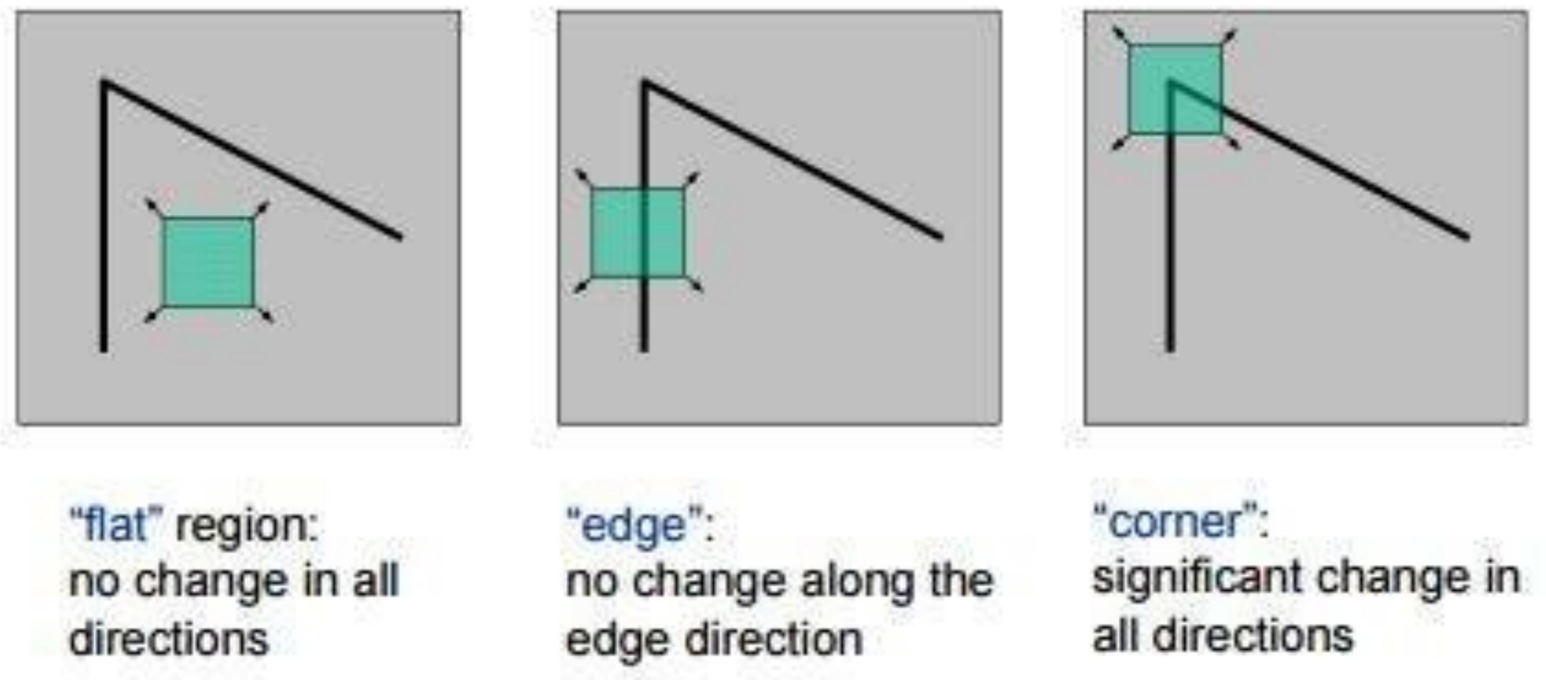
- How to connect OpenCV to a WebCam
- How to use OpenCV to open a video file
- How to draw shapes on video
- How to interact with video

Note: *You will need a webcam to follow along with certain parts of this section*

Corner Detection

Definition

Corner: Point where two edges meet. i.e., rapid changes of image intensity in two directions within a small region


"flat" region:
no change in all directions
"edge":
no change along the edge direction
"corner":
significant change in all directions

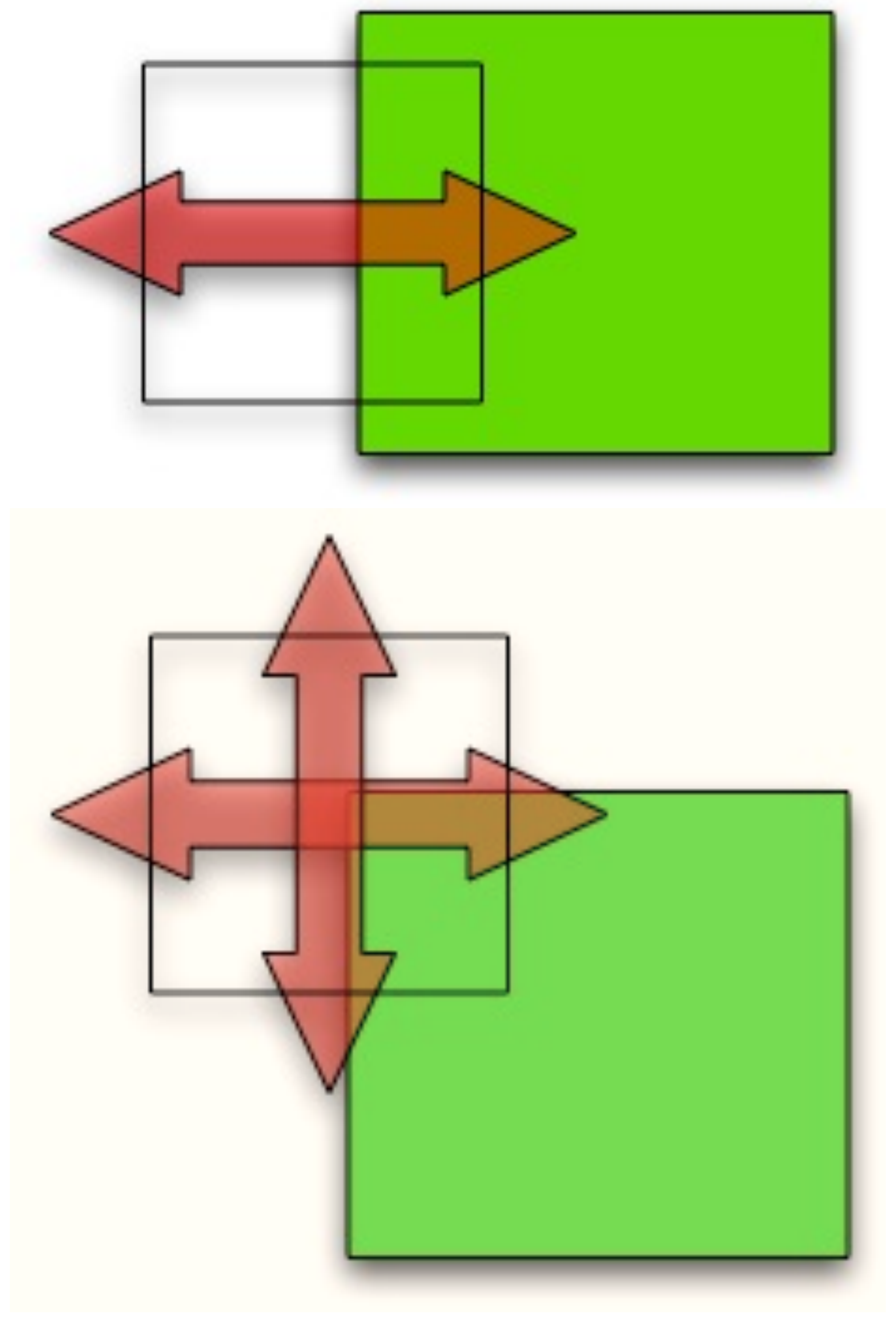
Corner Detection: Corner detection in computer vision is a technique used to identify points in an image where two edges meet, forming a "corner." These corners are important because they often contain a lot of useful information about the shape and structure of objects in the image.

Corner Detection

How it works

Corner detection works on the principle that if you place a small window over an image, if that window is placed on a corner then if it is moved in any direction there will be a large change in intensity.

- Flat regions will have no change in all directions
- Edges won't have a major change along the direction of the edge
- Corners will have a change in all directions, and therefore we know there must be a corner



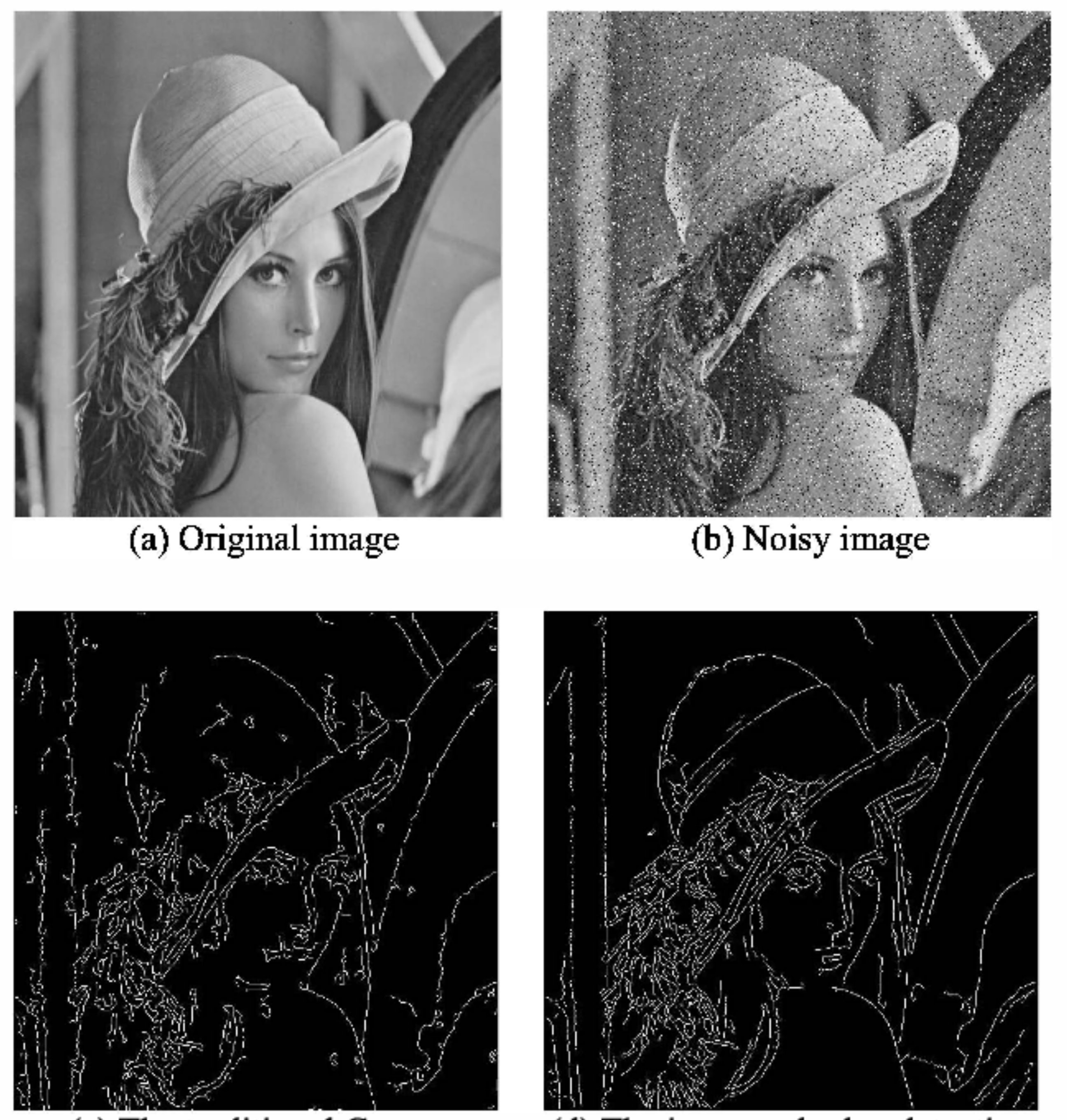
Canny Edge Detection

Definition

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.

Process:

1. **Gradient Calculation**: Compute edge intensity and direction .
2. **Non-Maximum Suppression:** Thin out edges by retaining local maxima
3. **Double Thresholding:** Classify edges into strong, weak, and non-relevant based on two thresholds.
4. **Edge Tracking by Hysteresis:** Connect weak edges to strong edges to finalize detection.
5. **Noise Reduction:** Smooth the image with a Gaussian filter.



(a) Original image (b) Noisy image

(c) The traditional Canny edge detection (d) The improved edge detection

Canny Edge Detection

Implementation with OpenCV

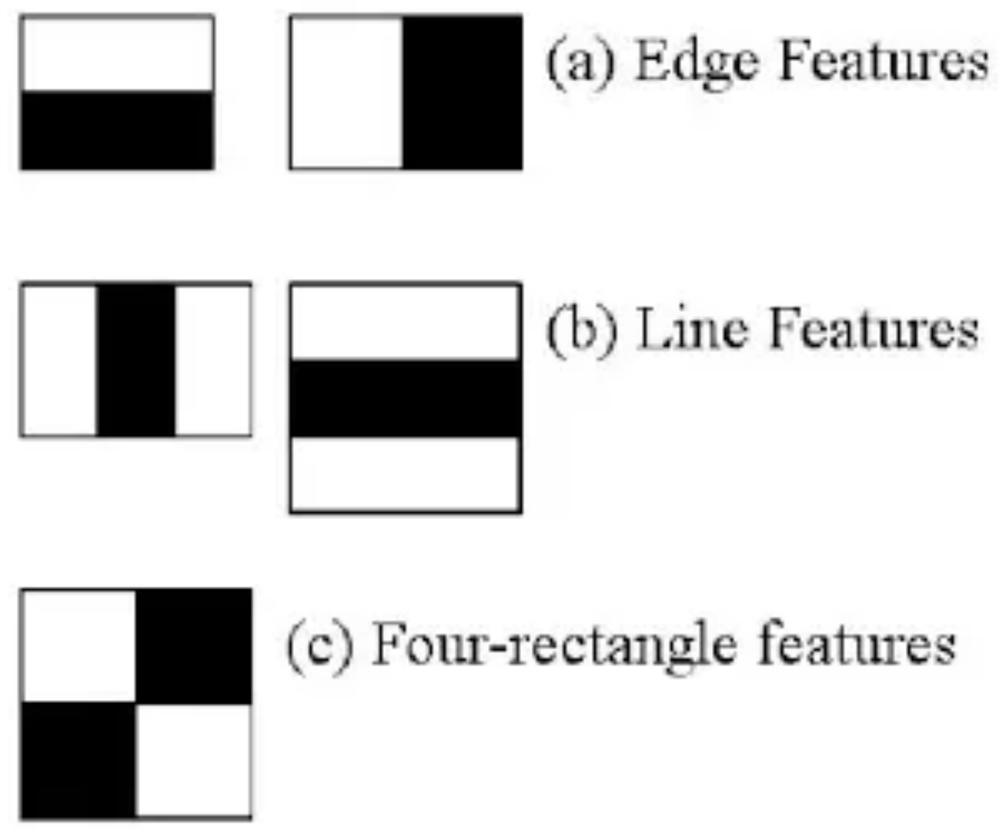
1. **Load Image:** Import your image using OpenCV.
2. **Convert to Grayscale:** Canny works on grayscale images.
3. **Apply Canny Edge Detection:** Use *cv2.Canny()* with two thresholds:
    - Lower Threshold: Set around 70% of the median pixel value
    - Upper Threshold: Set around 130% of the median pixel value.
4. **Adjust Thresholds:** Experiment with different threshold values to achieve the desired edge detection results

Basic Techniques

Haar Cascades

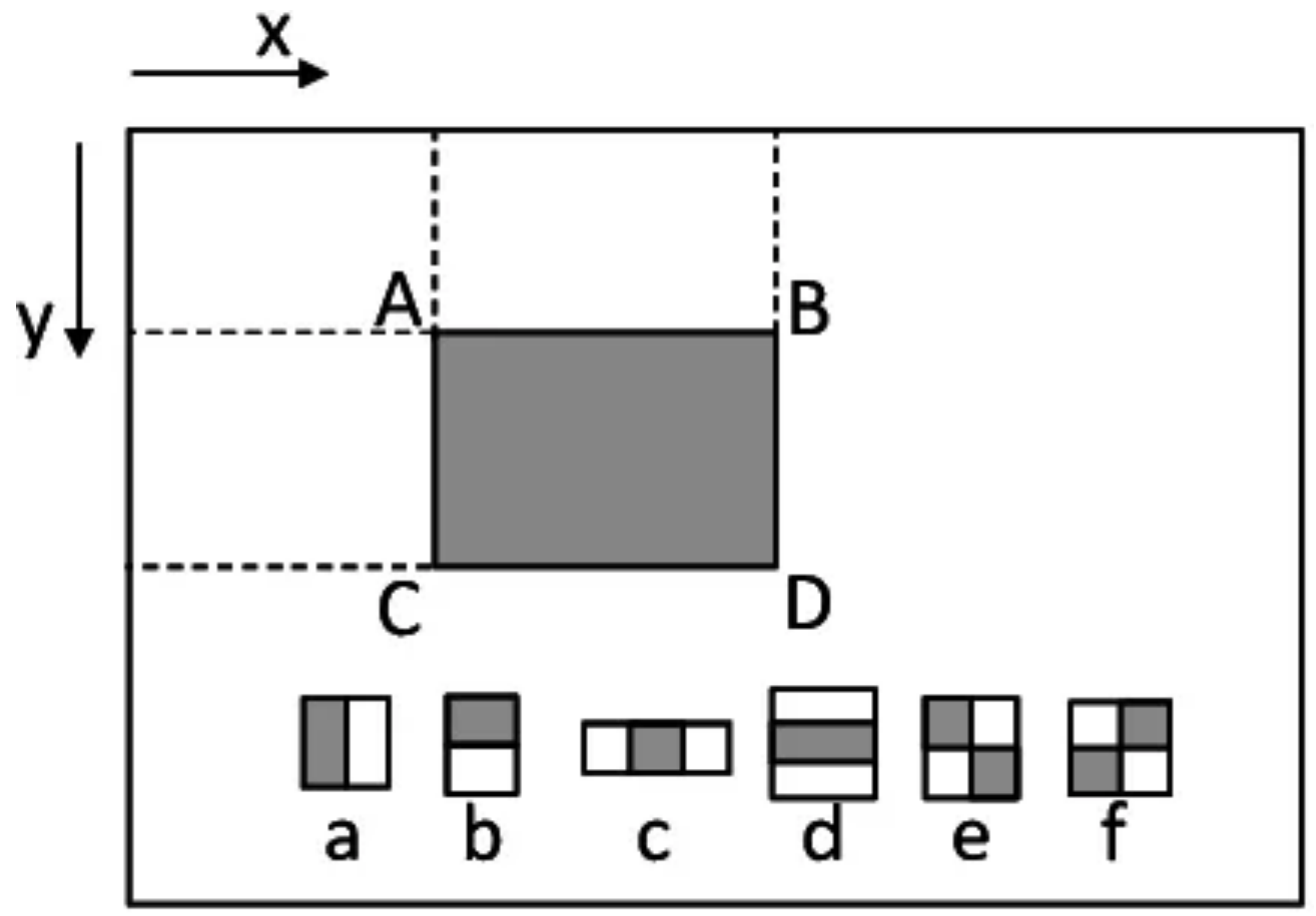
What is Haar Cascade?

- Traditional Object Detection technique based on machine learning
- Utilizes a classifier trained on features like edges, lines, and rectangles
- Commonly used for detecting faces, eyes, and other simple objects



How Haar Cascade Algorithm Works?

1. **Haar Features Calculation:** Identifies object features by comparing pixel intensity differences in adjacent regions.
2. **Integral Image:** Speeds up feature calculation by using summed pixel intensities over sub-rectangles.

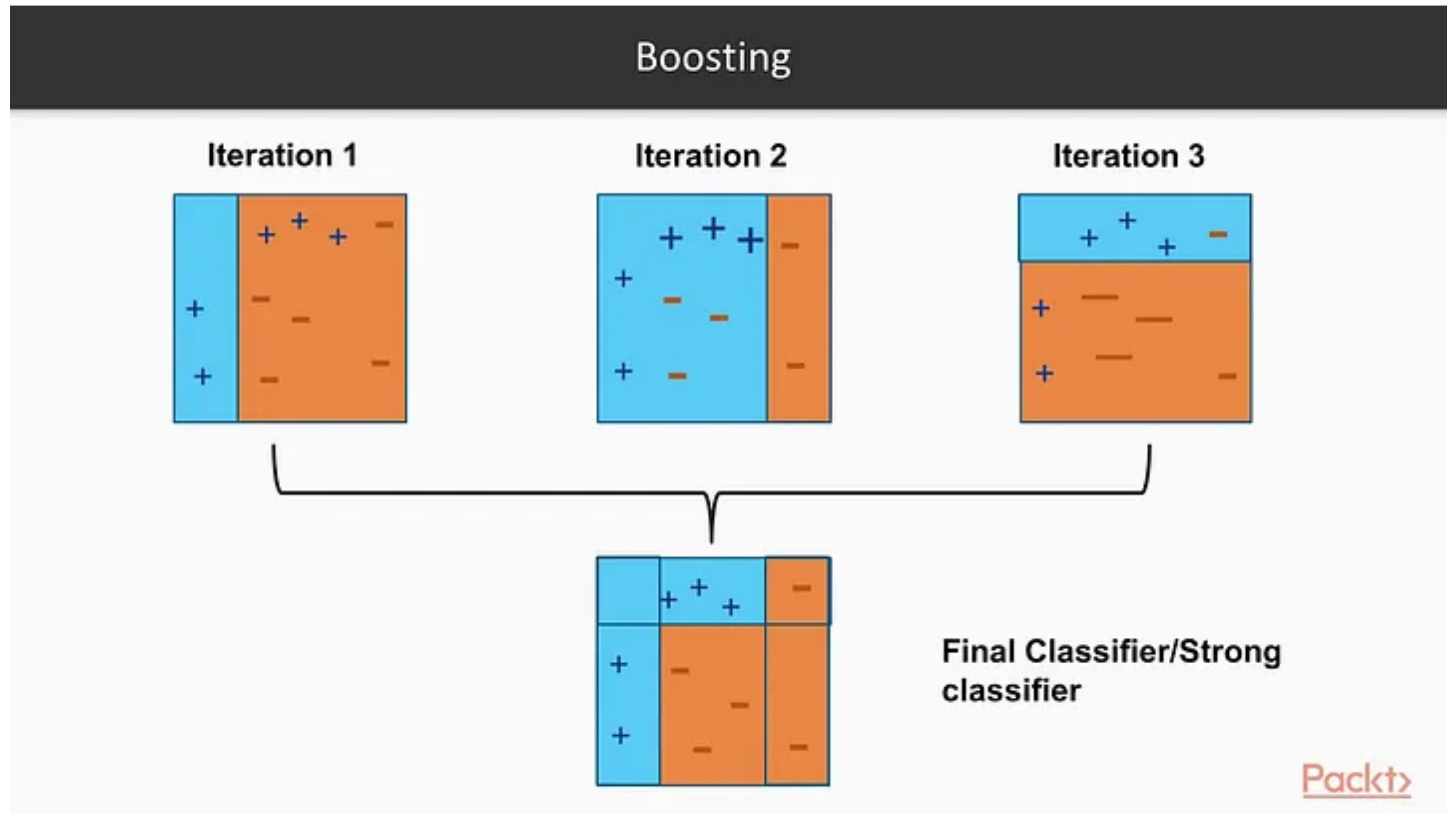


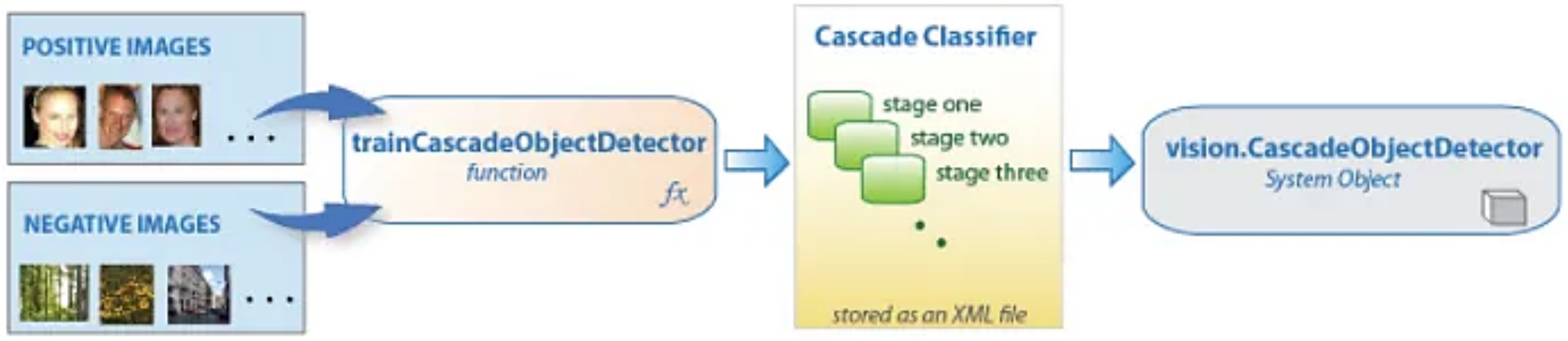
Basic Techniques

Haar Cascades

How Haar Cascade Algorithm Works

3. **Adaboost Training:** Combines weak classifiers to create a strong classifier, selecting the most relevant features
4. **Cascading Classifiers:** Sequentially filters regions, quickly discarding non-objects, improving speed and accuracy





Basic Techniques

Haar Cascades

Practical Example

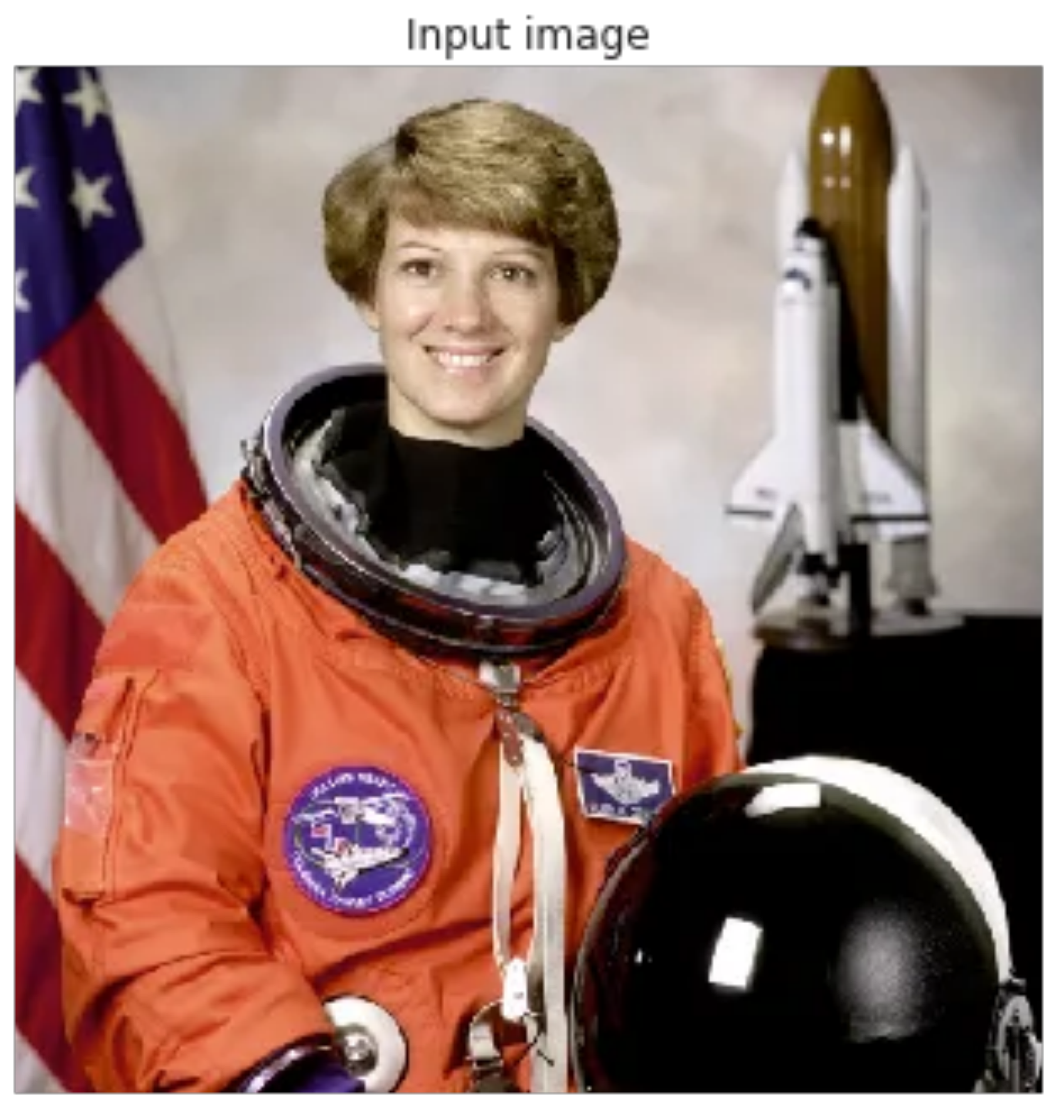
```python
import cv2

# Load pre-trained face detection classifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Load image
image = cv2.imread('image.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw rectangles around faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

# Display result
cv2.imshow('Detected Faces', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

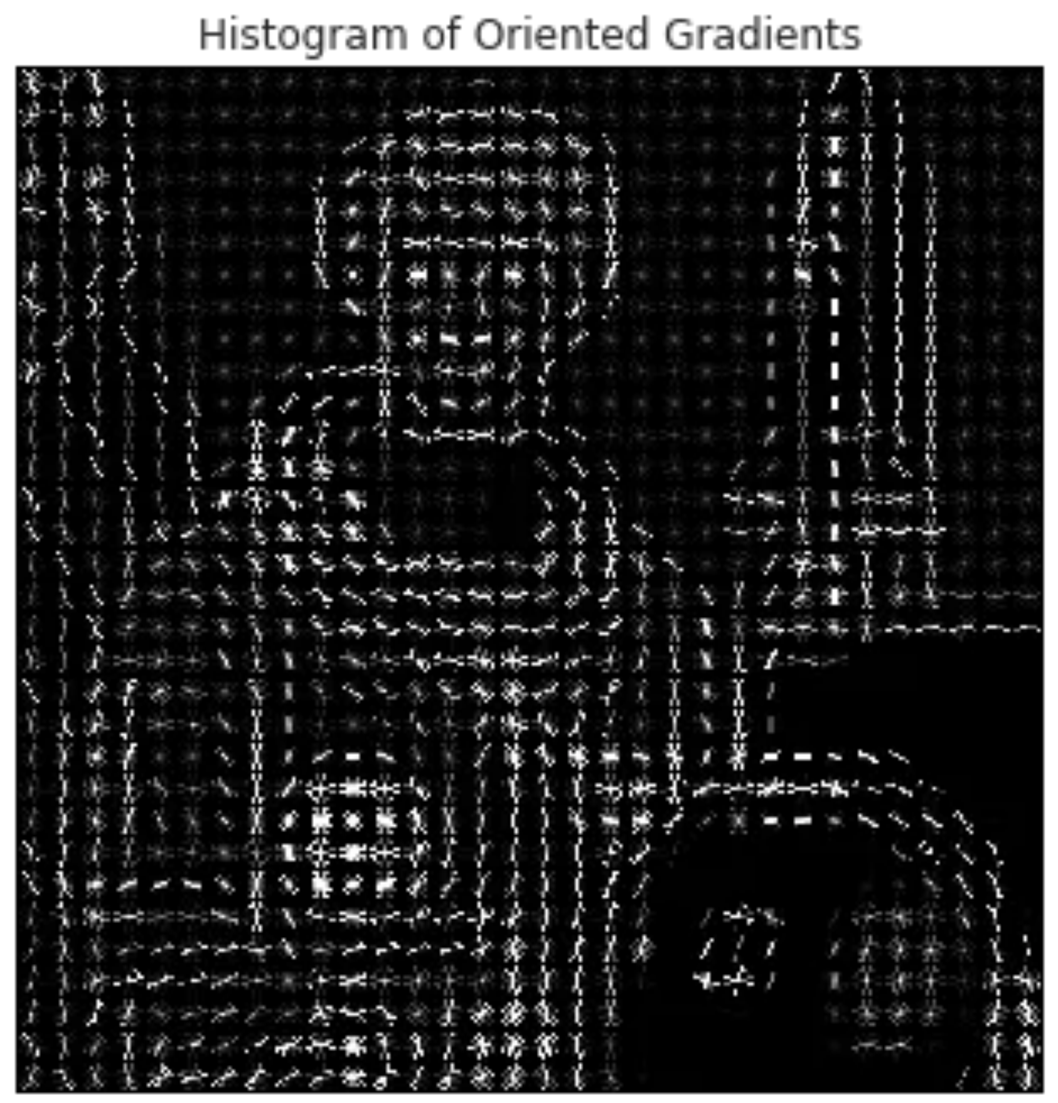
Basic Techniques

HOG (Histogram of Oriented Gradients)

What is HOG?

- HOG is a feature descriptor used for object detection, particularly for pedestrians.
- Focuses on the distribution of gradients (edge orientations) in localized regions of the image.





Basic Techniques

HOG (Histogram of Oriented Gradients)

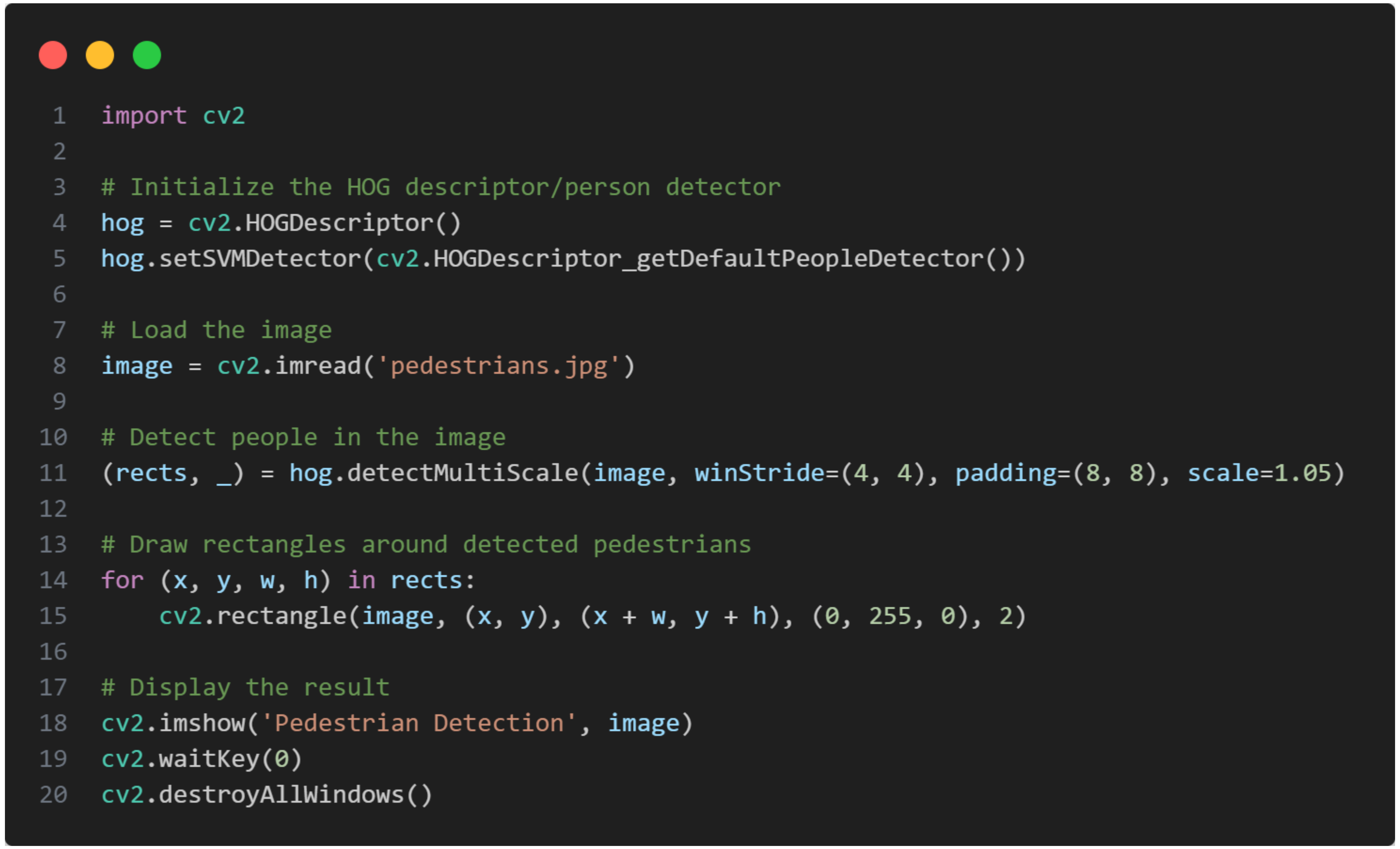
How it Works?

- **Feature Extraction**: HOG extracts key information from images by focusing on the edges and gradients, ignoring unnecessary background details.
- **Gradient Calculation**: The image is filtered to calculate horizontal and vertical gradients, identifying areas of intensity change, which signify edges and corners.
- **Histogram Creation**: Each image is divided into small cells (e.g., 8x8 pixels). For each cell, a histogram of gradient directions is calculated, with gradients binned based on their orientation (e.g., 0-160 degrees).
- **Block Normalization**: To reduce sensitivity to lighting and contrast variations, gradient values are normalized across larger blocks (e.g., 16x16 pixels), improving robustness.
- **Visualization**: The final HOG descriptor highlights key shapes in the image, making it suitable for object detection and classification algorithms like SVM.

Basic Techniques

HOG (Histogram of Oriented Gradients)

Practical Example

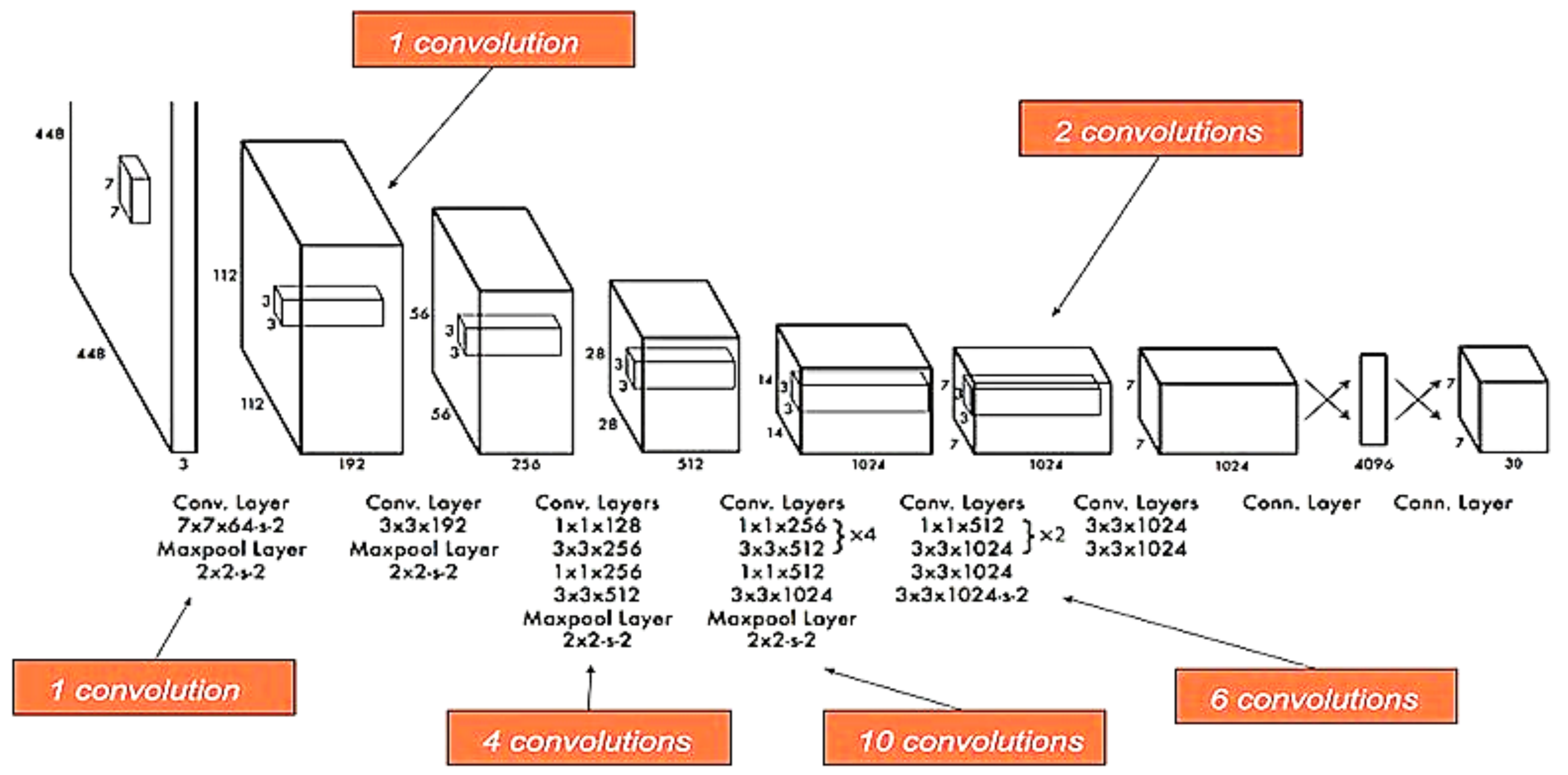


```python
import cv2

# Initialize the HOG descriptor/person detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

# Load the image
image = cv2.imread('pedestrians.jpg')

# Detect people in the image
(rects, _) = hog.detectMultiScale(image, winStride=(4, 4), padding=(8, 8), scale=1.05)

# Draw rectangles around detected pedestrians
for (x, y, w, h) in rects:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the result
cv2.imshow('Pedestrian Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

YOLO (You Only Look Once)

Overview

- YOLO is a series of real-time object detection systems based on convolutional neural networks Introduced by Joseph Redmon et al. in 2015
- The name "You Only Look Once" refers to the fact that the algorithm requires only one forward propagation pass through the neural network to make predictions
- In comparison with other object detection algorithms, YOLO proposes the use of an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once
- Inspired by the GoogleNet, the first Yolo architecture has a total of 24 convolutional layers with 2 fully connected layers at the end



YOLO (You Only Look Once)

Key Components

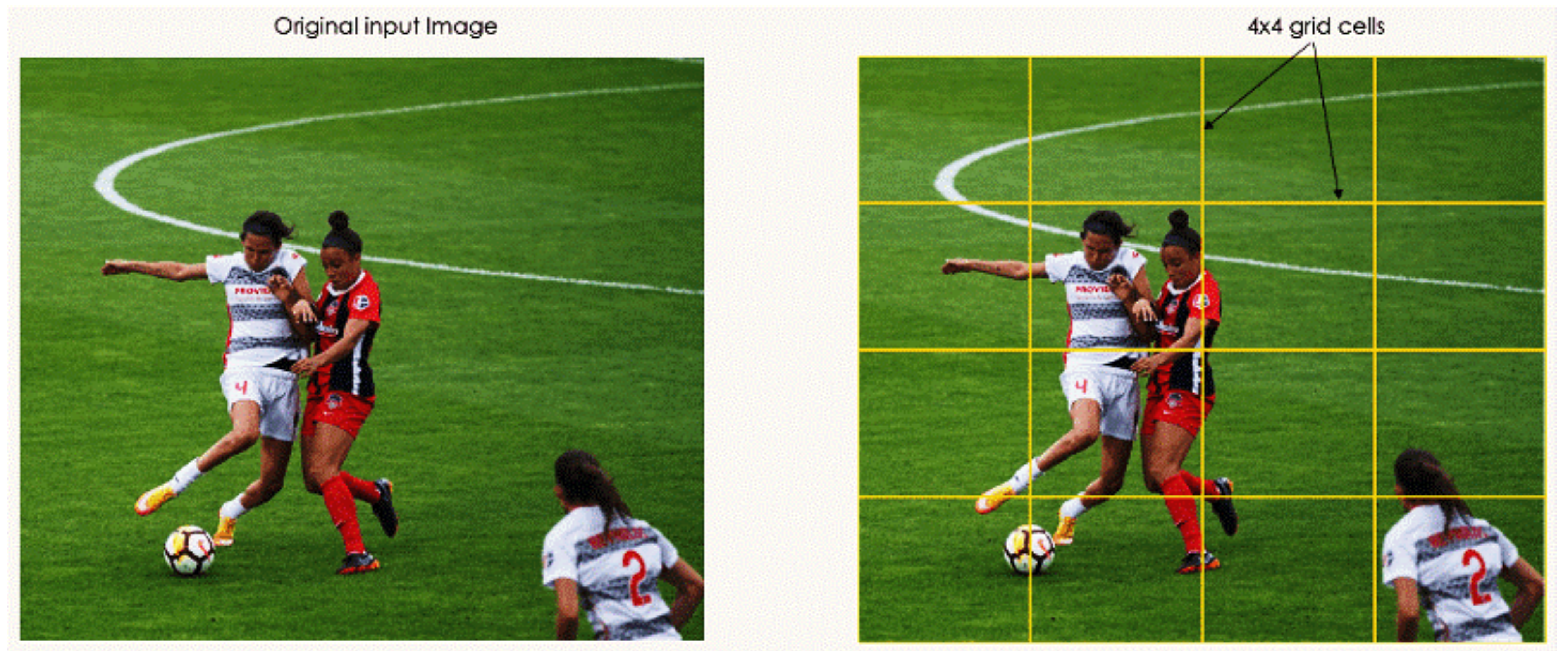
1. **Residual blocks:**
   - Divide the input image into S × S grid.
   - Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value
2. **Bounding Box Regression:**
   - Each object is surrounded by a rectangular box
   - YOLO predicts values like position (x, y), size (width, height), and class for each bounding box

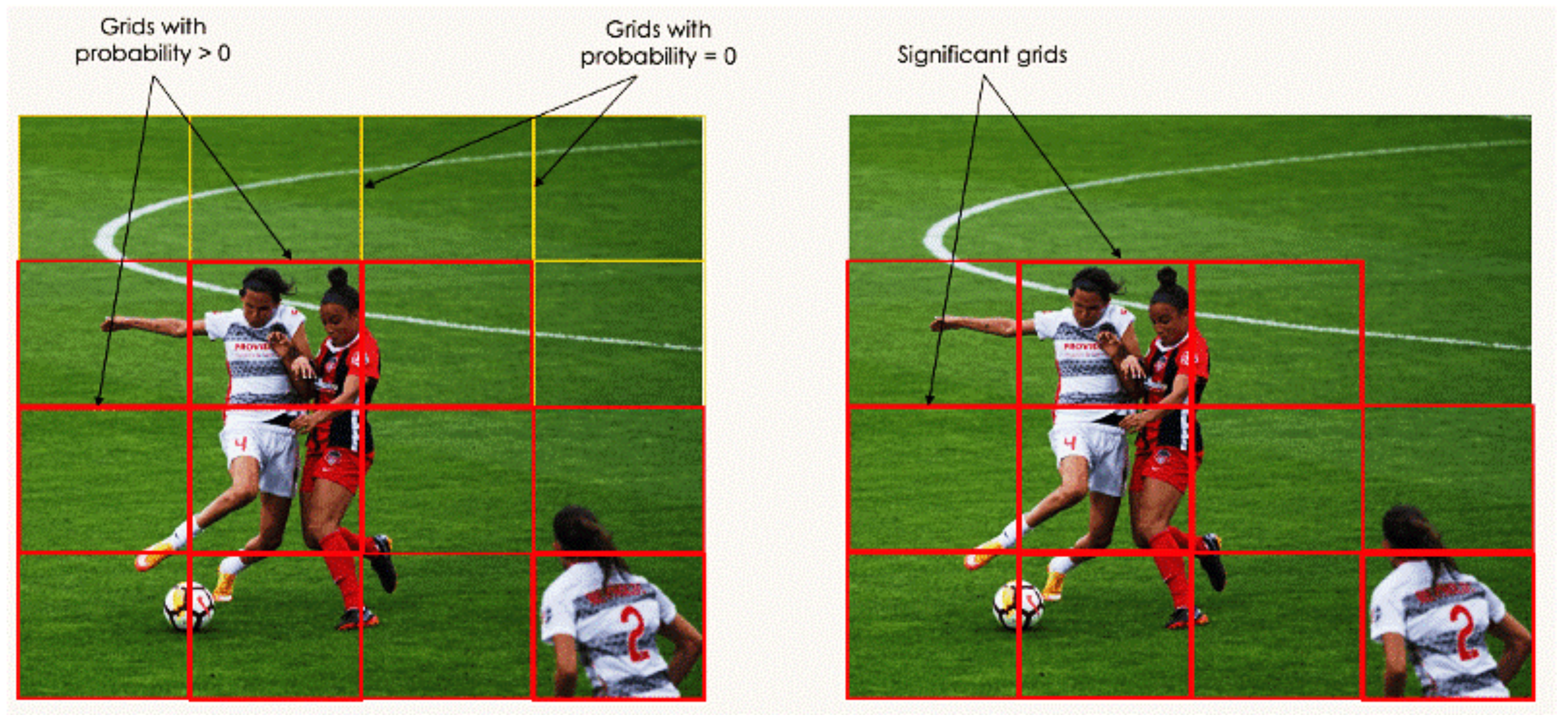
   Example: Y = [pc, bx, by, bh, bw, c1, c2]

   pc: probability score of the grid containing an object.

   bx,by,bh,bw: Coordinates and dimensions of the bounding box

   c1,c2: Object classes, e.g., player or ball.



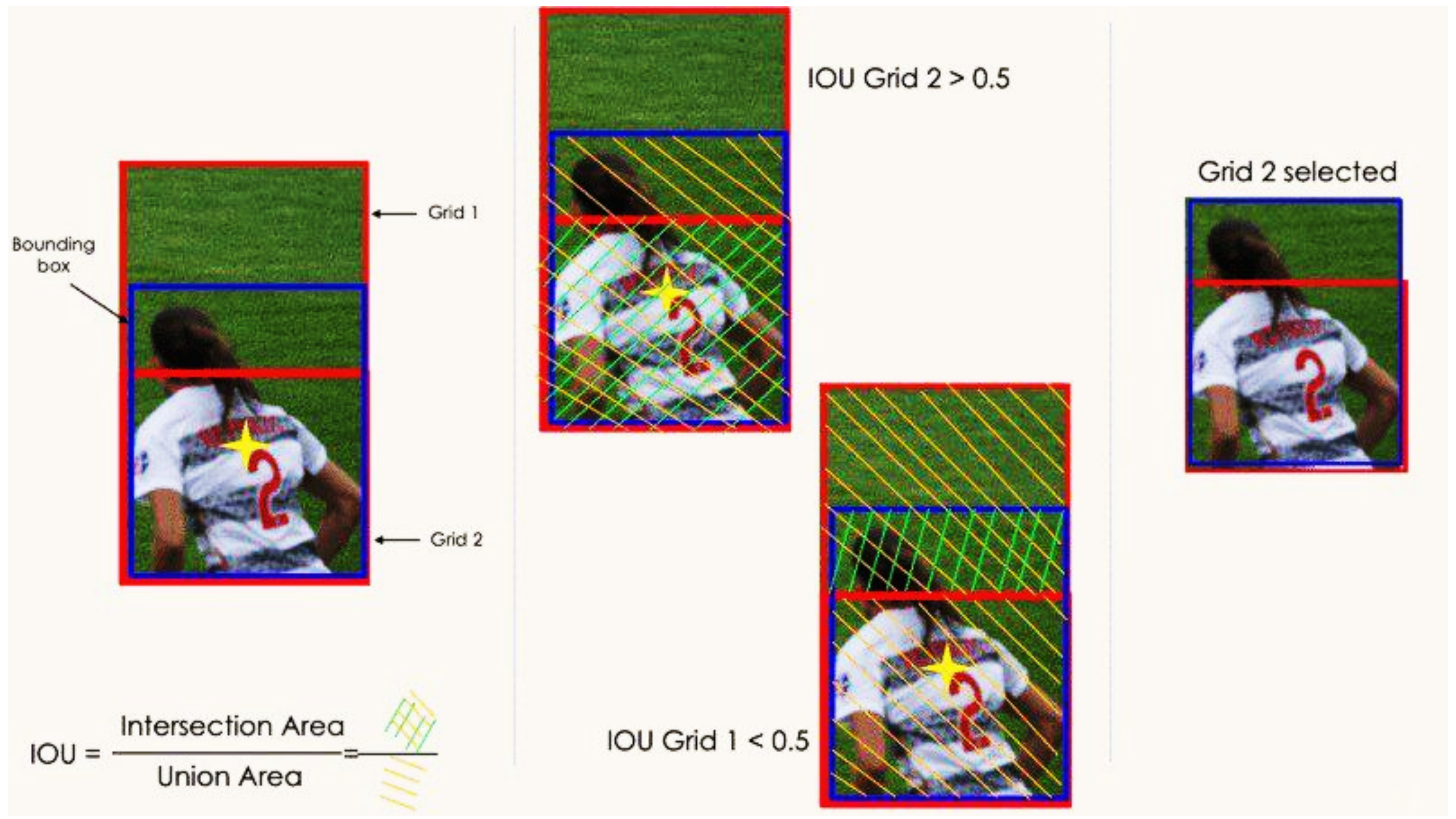


YOLO (You Only Look Once)

Key Components

3. Intersection Over Union (IOU):

- IOU (a value between 0 and 1) is used to discard irrelevant boxes with low overlap
- Process:
  - Set an IOU threshold (e.g., 0.5)
  - YOLO computes the IOU of each grid cell which is the Intersection area divided by the Union Area
  - Keep boxes with IOU > threshold, discard those ≤ threshold.



4. Non-Maximum Suppression (NMS)

- Even after setting an IOU threshold, multiple high-scoring boxes may still overlap for the same object
- NMS keeps only the box with the highest confidence score, suppressing others

YOLO (You Only Look Once)

Versions

YOLOv1

Introduced the concept of a single-stage object detector

2015

YOLOv2

Improved accuracy and speed with batch normalization, higher resolution, and anchor boxes

2017

YOLOv3

Added objectness score, multi-scale predictions, and connections to the backbone

2018

YOLOv4-5

Imroved feature aggregation
Highly customizable

2020

YOLOv6-8

Hardware-minded design
Introduced E-ELAN for improved accuracy and speed
Anchor-free training

2022-202

YOLOv9-10

Improved accuracy and speed using programmable gradient information.
State-of-the-art performance with fewer parameters and lower latency

2024

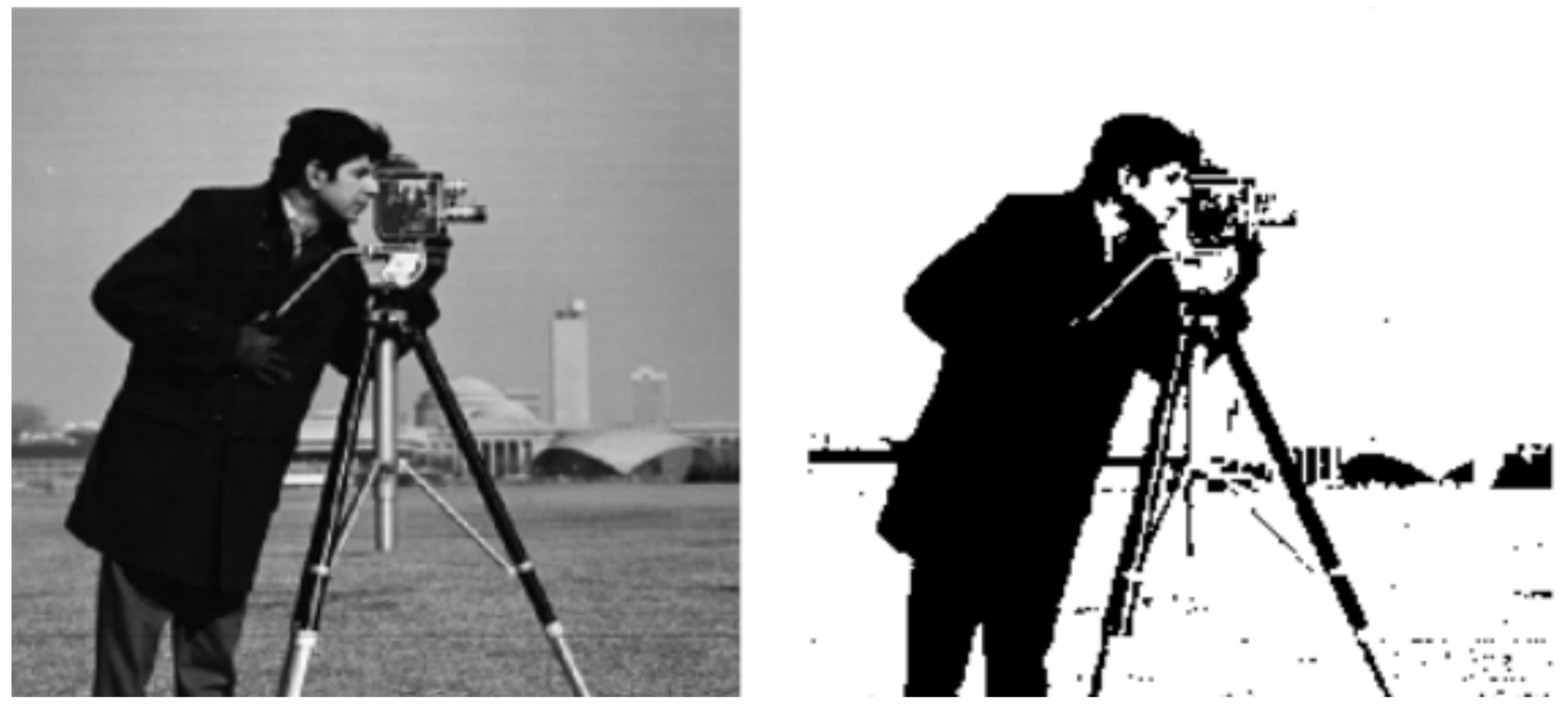
Image Segmentation

Thresholding

Intro

Thresholding is the simplest form of image segmentation. Its idea is to separate image pixels into groups (like foreground & background) based on pixels' intensity values and certain thresholds.

There exists several thresholding methods, each specifying number of groups or thresholds' values in a different way.



Thresholding

Global

Simple (Global) thresholding uses a single intensity value as a threshold to separate pixels into 2 groups, depending whether each pixel intensity is below/above this threshold (usually 2 groups are black/white)

$$I(x,y) = \begin{cases} 0 & \text{if } I(x,y) < T \\ 255 & \text{if } I(x,y) \geq T \end{cases}$$

where T is the threshold (usually 128 for grayscale case), and I(x,y) is intensity of pixel at (x,y) coordinates.

Thresholding

Adaptive

Adaptive thresholding computes a threshold for each pixel based on the intensities of its neighboring pixels within a small window (block size), as:

$$T(x,y) = \frac{1}{N} \sum_{(i,j)\in\mathcal{N}(x,y)} I(i,j) - C$$

where N(x,y) is neighbourhood of (x,y)

This method is particularly useful when there is non-uniform lighting or shadows across different parts of the image, which can cause global thresholding to fail.

Thresholding
Otsu

Otsu’s method determines the optimal threshold to divide the image pixels into 2 groups, foreground and background.

It automatically does this in a way that maximizes the variance between those 2 groups, and minimizes the variance within them.

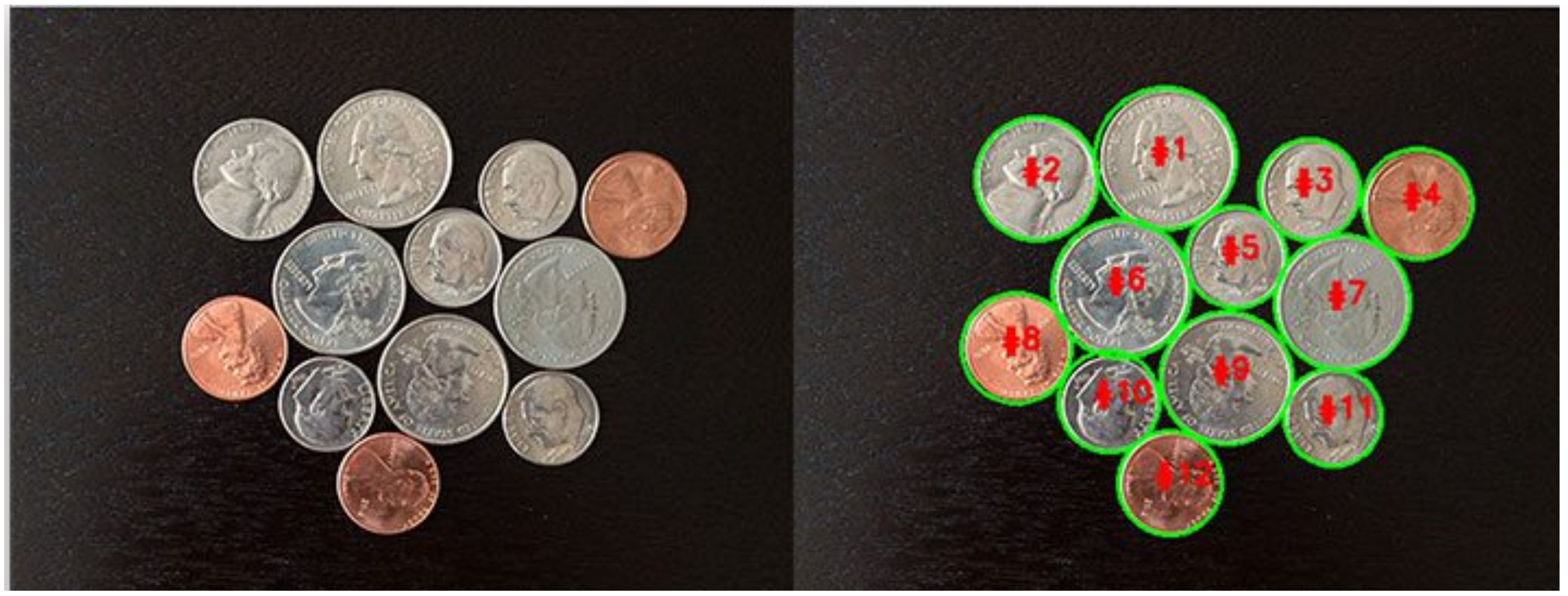
$$T^* = \arg\max_T \sigma_b^2(T)$$

where σ_b^2 is variance between classes

Watershed Algorithm

Watershed Segmentation treats grayscale images like topographic maps, where regions of high intensity are peaks and low intensity are valleys. The algorithm floods basins starting from predefined markers, expanding the regions until the image is segmented.

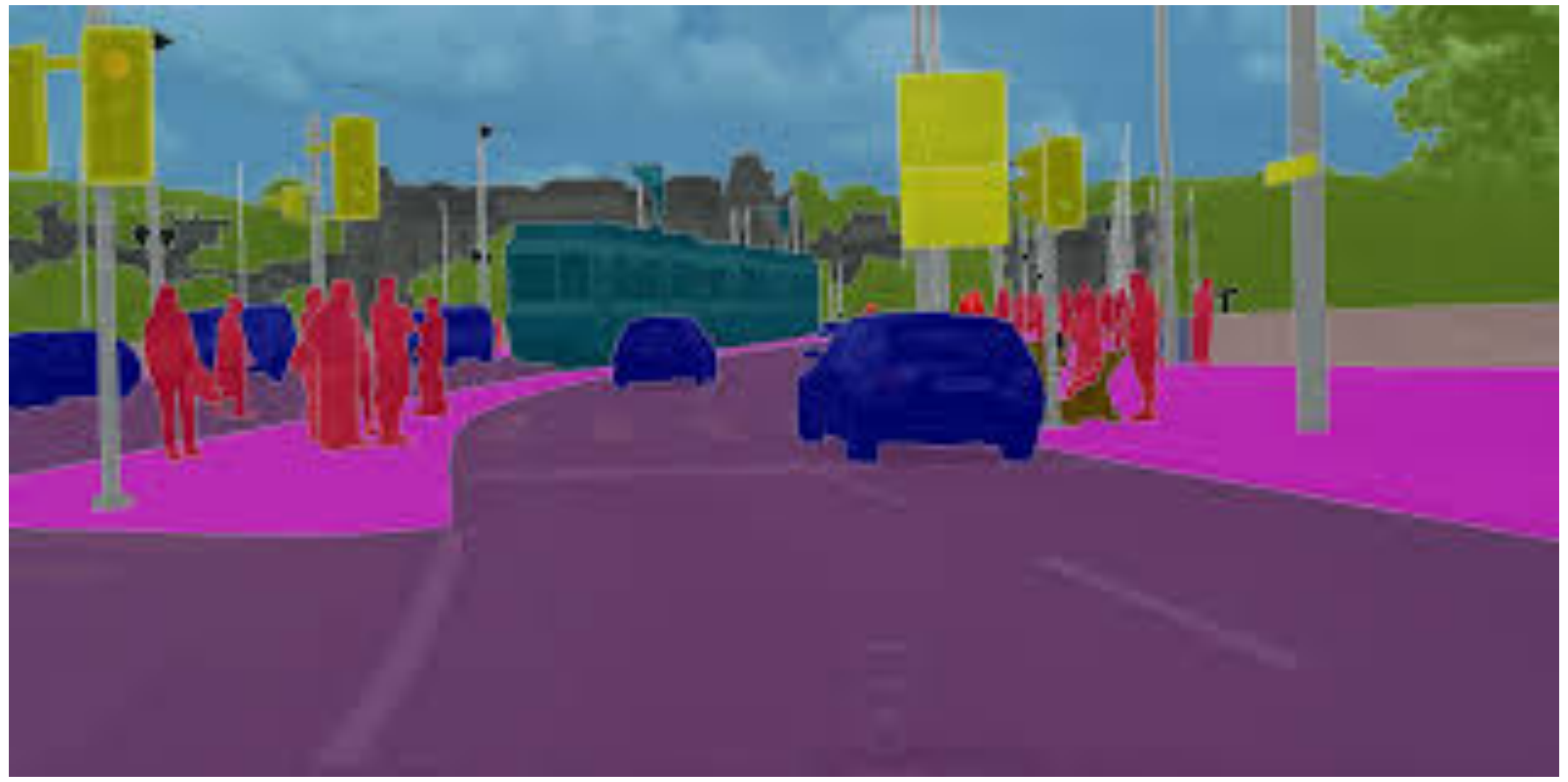
This algorithm is useful for separating objects that are touching or overlapping, such as separating coins in an image.



Semantic Segmentation with Deep Learning

While Contouring detects and outlines the boundaries of objects within an image, **Semantic Segmentation** classifies each pixel in an image into predefined categories.

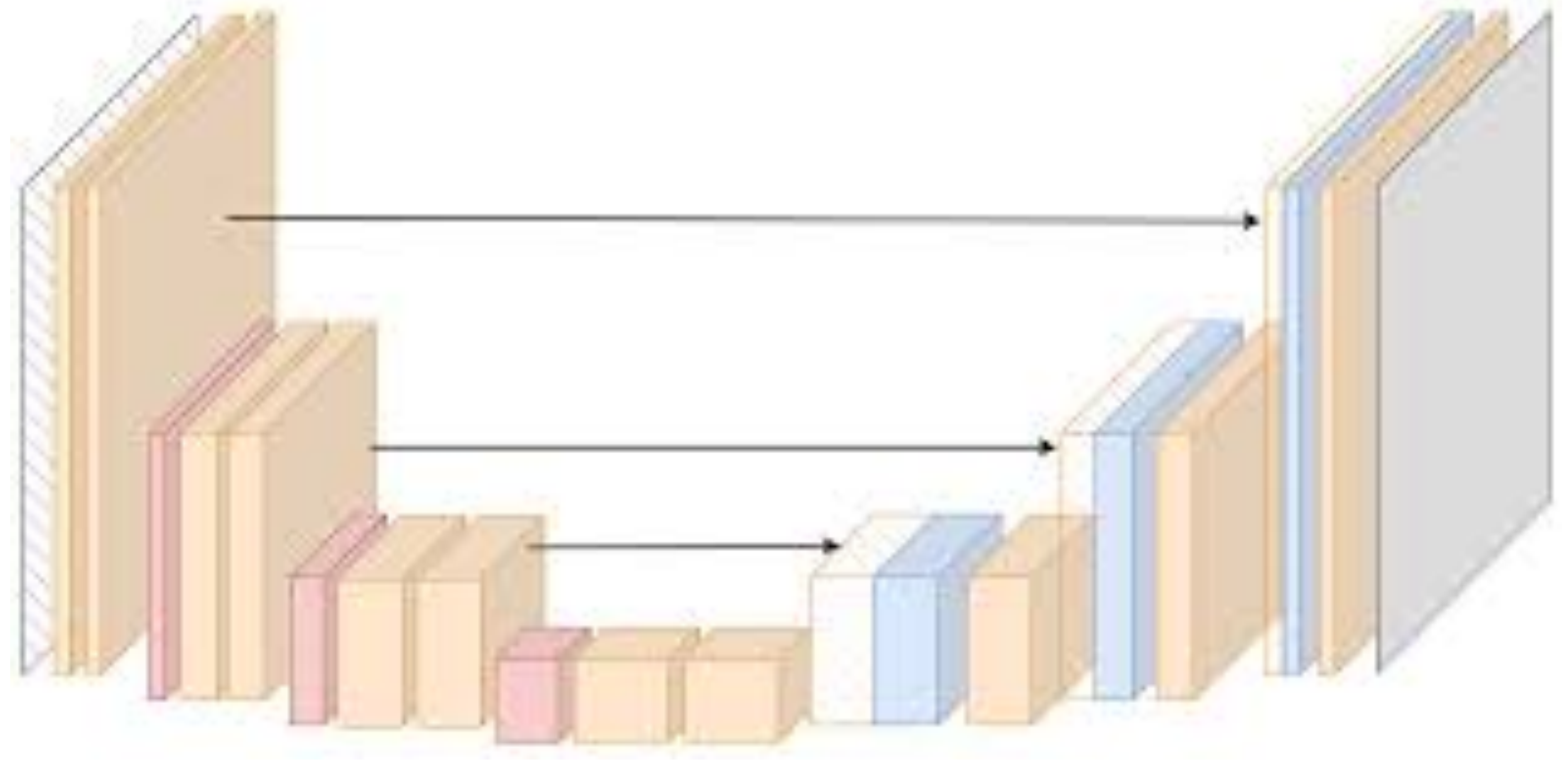
Usually done using CNNs and their variants, like U-Net, Mask R-CNN or SegNet..



U-Net

A popular architecture for image segmentation, based on an encoder-decoder structure, where the input image is down-sampled (size decreases but channels increase) and then up-sampled to generate the segmented image, passing through many convolutional layers along the way.

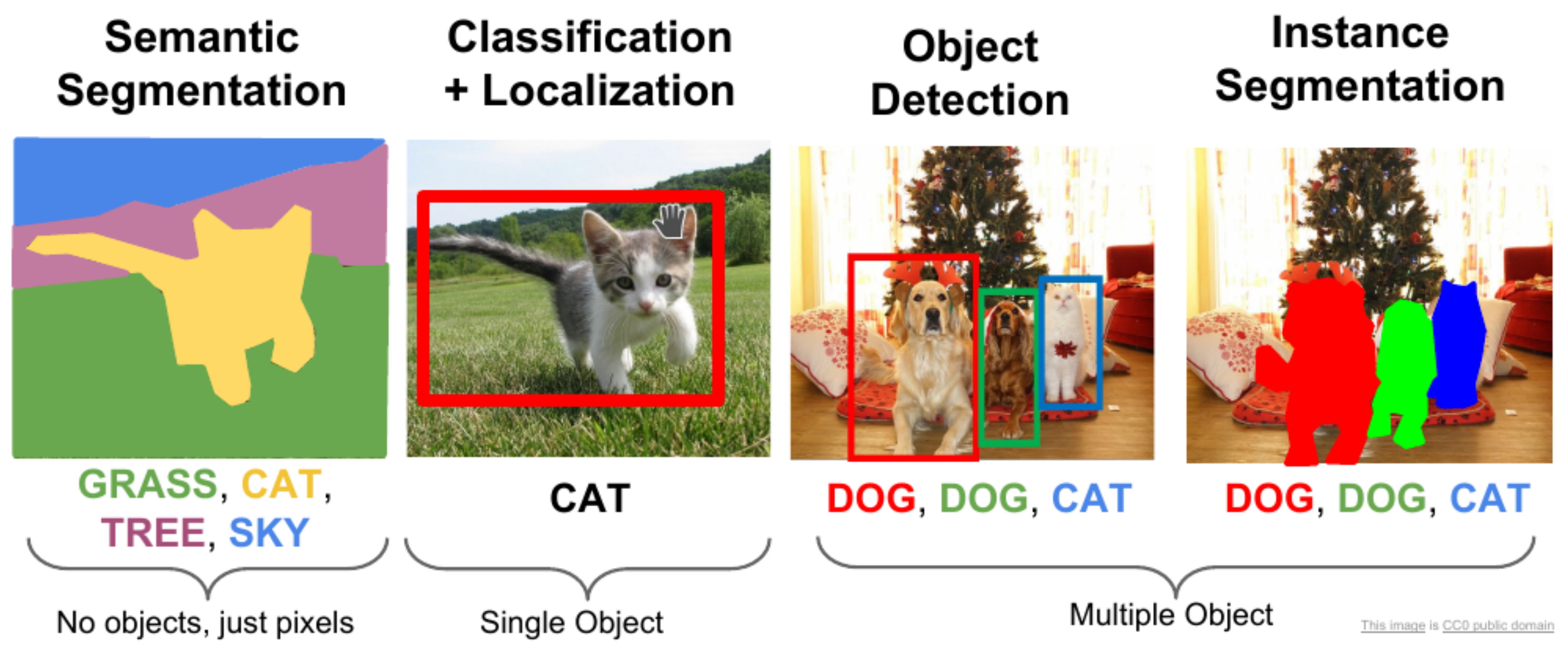
The output of U-NET is a binary segmentation, which marks whether each pixel represents a background region or not.



Mask R-CNN

Mask R-CNN is an extension of Faster R-CNN that adds a branch for predicting segmentation masks on top of bounding boxes and class labels.

This allows for instance segmentation, which can detect multiple objects of the same class in an image and segment each one individually.



Convolutional Neural Networks

Brief from last course

Transfer Learning in Computer Vision

. Pre-trained Models (VGG, ResNet, EfficientNet)

- Introduction to Transfer Learning:
  - Explain **transfer learning**, where models pre-trained on large datasets (like ImageNet) are reused to solve a different, smaller task. Instead of training a model from scratch, you leverage the knowledge learned by these models.
- VGG (Visual Geometry Group):
  - Introduce **VGG**, a deep convolutional neural network model that uses 16 or 19 layers of convolutions followed by fully connected layers.
  - Discuss its strengths (simplicity, ease of use) and limitations (large model size, slower inference).
  - **Practical Example**: Load a pre-trained VGG16 model using Keras or PyTorch and use it for image classification.
- ResNet (Residual Networks):
  - Explain **ResNet**, a deeper network that solves the vanishing gradient problem by introducing residual connections. These "skip connections" allow the model to train effectively even with hundreds of layers.
  - **Practical Example**: Load a pre-trained ResNet model using Keras or PyTorch and classify images.
- EfficientNet:
  - Introduce **EfficientNet**, a family of models that scale both the depth, width, and resolution of networks more efficiently than previous models, achieving better accuracy with fewer parameters.
  - **Practical Example**: Implement an EfficientNet model using TensorFlow or Keras for a custom task.

Freezing and Unfreezing Layers:

- Explain the process of fine-tuning, where the base layers of a pre-trained model are "frozen" to retain their learned features, and only the final layers are trained on the new task. Once the model stabilizes, more layers can be "unfrozen" for further training.
- **Practical Example**: Demonstrate how to freeze and unfreeze layers in Keras or PyTorch when fine-tuning a pre-trained ResNet or EfficientNet model for a custom dataset.

Customizing the Final Layers:

- Teach students how to modify the last few layers of a pre-trained model to suit their specific task (e.g., change the output layer to have a different number of classes).
- **Practical Example**: Modify the final layer of a pre-trained model to classify a custom dataset with fewer classes than the original dataset (e.g., binary classification).

Optimizing for the Task:

- Discuss how to adjust learning rates, optimizers, and loss functions when fine-tuning pre-trained models to optimize for the specific task.
- **Practical Example**: Use different optimizers like Adam or SGD and adjust learning rates during the fine-tuning process.

Suggested Code Examples:

- Load and fine-tune a pre-trained VGG16, ResNet, or EfficientNet model in Keras.
- Customize the final layers of a pre-trained model and fine-tune it for a custom task.
- Implement freezing and unfreezing of layers during fine-tuning to improve model performance.

GenAI in Computer Vision

1. Autoencoders for Image Generation

- Introduction to Autoencoders:
  - Explain that autoencoders are neural networks designed to compress (encode) input data into a lower-dimensional representation and then reconstruct (decode) it back to its original form. They are often used for tasks like image denoising and generating images.
  - **Encoder-Decoder Architecture**: Describe the two main parts of an autoencoder — the encoder that compresses the image and the decoder that reconstructs it.
- Variational Autoencoders (VAEs):
  - Introduce **VAEs**, a type of autoencoder that learns a probability distribution over latent spaces, allowing for more varied image generation. VAEs are useful for generating new images by sampling from the latent space.
  - **Practical Example**: Show how to implement a VAE in TensorFlow or PyTorch to generate new images (e.g., MNIST digits).

Generative Adversarial Networks (GANs)

- Introduction to GANs:
  - Explain that **GANs** consist of two neural networks: a generator that creates images and a discriminator that evaluates how realistic the generated images are. The two networks are trained in a competitive manner, with the generator aiming to fool the discriminator.
  - **Architecture Overview**: Discuss how the generator and discriminator work together. The generator creates images from random noise, and the discriminator tries to distinguish real images from fake ones.

Practical Example:

- Implement a simple GAN using TensorFlow or PyTorch. Use it to generate images from a random distribution (e.g., generating fake MNIST digits).
- Discuss common challenges like mode collapse and ways to improve training stability in GANs (e.g., using Wasserstein GANs).

Applications of Generative Models

- Style Transfer:
  - Introduce **neural style transfer**, where the style of one image is applied to the content of another (e.g., making a photo look like a Van Gogh painting). This is done by optimizing the content and style representations extracted from a pre-trained network like VGG.
  - **Practical Example**: Implement style transfer using a pre-trained VGG19 model in TensorFlow or PyTorch.
- Super-Resolution:
  - Explain **super-resolution** as the process of generating high-resolution images from low-resolution inputs using deep learning techniques.
  - **SRGAN (Super-Resolution GAN)**: Introduce SRGAN, a GAN-based approach for super-resolution, which enhances the details of low-resolution images.
  - **Practical Example**: Use a pre-trained SRGAN model to upscale a low-resolution image.

Suggested Code Examples:

- Implement a simple autoencoder and variational autoencoder (VAE) for image reconstruction and generation.
- Build a basic GAN for generating new images and discuss improvements with Wasserstein GAN (WGAN).
- Apply neural style transfer to blend the style of one image with the content of another.
- Use a pre-trained SRGAN model to perform image super-resolution.