Lecture 11

Learning outcomes:

- *Introduction to SymPy for symbolic mathematics:*
- *Simulating simple engineering systems (e.g., pendulum, electrical circuits)*
- *Introduction to OpenCV or similar tools for basic image analysis in engineering.*

Introduction to SymPy for Symbolic Mathematics

- In the realm of mathematics and computer science, symbolic computation is an area that deals with the manipulation of mathematical symbols and expressions.
- Unlike numerical computation, which deals with specific numbers, symbolic computation allows for the manipulation of algebraic expressions, equations, and other mathematical objects in a symbolic form.
- One of the most powerful libraries available for symbolic computation in Python is SymPy.

What is SymPy?

- SymPy is an open-source Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible to be comprehensible and easily extensible.

- SymPy is written entirely in Python and can be used in both interactive and programmatic environments. Its primary goal is to provide a robust and efficient symbolic computation library that integrates seamlessly with the Python ecosystem.

Basic Symbolic Computation

❑ Defining Symbols

- The foundation of symbolic computation is the ability to define and manipulate symbols. In SymPy, we use the symbols function to define symbolic variables:

```
from sympy import symbols

x, y, z = symbols('x y z')
```

- Now, x, y, and z are symbolic variables that we can use in mathematical expressions.

Basic Operations

- With symbols defined, we can perform basic arithmetic operations just like we would with numerical variables:

```
from sympy import symbols

x, y = symbols('x y')


# Addition
expr = x + y
print(expr) # Output: x + y
```

```
# Subtraction
expr = x - y
print(expr) # Output: x - y

# Multiplication
expr = x * y
print(expr) # Output: x*y

# Division
expr = x / y
print(expr) # Output: x/y

# Power
expr = x**2 + y**2
print(expr) # Output: x**2 + y**2
```

Algebraic Manipulations

- Expanding and Factoring Expressions

SymPy provides powerful tools for algebraic manipulations, such as expanding and factoring expressions.

```
from sympy import expand, factor

expr = (x + y)**2

# Expand the expression
expanded_expr = expand(expr)
print(expanded_expr) # Output: x**2 + 2*x*y + y**2

# Factor the expression
factored_expr = factor(expanded_expr)
print(factored_expr) # Output: (x + y)**2
```

- Simplifying Expressions

SymPy can simplify complex mathematical expressions to their simplest form using the simplify function:

```
from sympy import simplify

expr = (x**2 + 2*x*y + y**2) / (x + y)

# Simplify the expression
simplified_expr = simplify(expr)
print(simplified_expr) # Output: x + y
```

Solving Equations Symbolically

- One of the most powerful features of SymPy is its ability to solve equations symbolically.

➢ ***Solving Algebraic Equations***

To solve algebraic equations, we use the solve function. For example, let's solve the equation $x^2-4=0$:

```
from sympy import solve

# Define the equation
equation = x**2 - 4

# Solve the equation
solutions = solve(equation, x)
print(solutions) # Output: [-2, 2]
```

➢ *Solving Systems of Equations*

SymPy can also solve systems of equations. For example, let's solve the system:

$$\begin{cases} 2x + y = 1 \\ x - y = 3 \end{cases}$$

```
# Define the equations
equation1 = 2*x + y - 1
equation2 = x - y - 3

# Solve the system of equations
solutions = solve((equation1, equation2), (x, y))
print(solutions) # Output: {x: 4/3, y: -5/3}
```

Applications in Simplifying Complex Mathematical Expressions

- SymPy's ability to simplify complex mathematical expressions is invaluable in various fields, including physics, engineering, and computer science. Let's explore a few examples.

➢ ***Simplifying Trigonometric Expressions***

Trigonometric expressions can often be simplified using SymPy's trigsimp function:

```
from sympy import trigsimp, sin, cos

expr = sin(x)**2 + cos(x)**2

# Simplify the trigonometric expression
simplified_expr = trigsimp(expr)
print(simplified_expr) # Output: 1
```

➢ ***Simplifying Logarithmic Expressions***

Logarithmic expressions can be simplified using the logcombine function

```
from sympy import log, logcombine

expr = log(x) + log(y)

# Simplify the logarithmic expression
simplified_expr = logcombine(expr)
print(simplified_expr) # Output: log(x) + log(y)
```

➢ ***Simplifying Exponential Expressions***

Exponential expressions can be simplified using the simplify function:

```
from sympy import exp, simplify

expr = exp(x) * exp(y)

# Simplify the exponential expression
simplified_expr = simplify(expr)
print(simplified_expr) # Output: exp(x + y)
```

Advanced Features of SymPy

- SymPy provides robust support for calculus, including differentiation and integration.

➢ ***Differentiation***

To compute the derivative of an expression, use the diff function:

```
from sympy import diff

expr = x**3 + 3*x**2 + 3*x + 1

# Compute the derivative
derivative = diff(expr, x)
print(derivative) # Output: 3*x**2 + 6*x + 3
```

➢ ***Integration***

To compute the integral of an expression, use the integrate function:

```
from sympy import integrate

# Compute the indefinite integral
indefinite_integral = integrate(expr, x)
print(indefinite_integral) # Output: x**4/4 + x**3 + 3*x**2/2 + x

# Compute the definite integral
definite_integral = integrate(expr, (x, 0, 1))
print(definite_integral) # Output: 15/4
```

➢ ***Limits***

SymPy can also compute limits using the limit function:

```
from sympy import limit

expr = (sin(x) / x)

# Compute the limit as x approaches 0
limit_value = limit(expr, x, 0)
print(limit_value) # Output: 1
```

Solving Differential Equations

SymPy provides tools for solving ordinary differential equations (ODEs):

```
from sympy import Function, dsolve, Eq

f = Function('f')
eq = Eq(f(x).diff(x, x) + 9*f(x), 1)

# Solve the differential equation
solution = dsolve(eq)
print(solution) # Output: Eq(f(x), C1*sin(3*x) + C2*cos(3*x) + 1/9)
```

Simulating simple engineering systems (e.g., pendulum, electrical circuits)

- Using **SymPy**, a Python library for symbolic mathematics, we can simulate engineering systems by deriving and solving differential equations symbolically.

- SymPy excels at analytical solutions, making it ideal for studying system behavior and generating exact solutions.

SymPy Code for Solving the Pendulum Equation

```
from sympy import symbols, Function, Eq, dsolve, sin, simplify

# Define variables and functions
t = symbols('t')
theta = Function('theta')(t)
g, L = symbols('g L')

# Define the linearized pendulum equation
pendulum_eq = Eq(theta.diff(t, 2) + (g / L) * theta, 0)
```

```
# Solve the differential equation
solution = dsolve(pendulum_eq, theta)
simplified_solution = simplify(solution)

# Display the solution
print("Pendulum Solution:")
print(simplified_solution)
```

✓**Output**:

```
Pendulum Solution:
Eq(theta(t), C1*exp(-t*sqrt(-g/L)) + C2*exp(t*sqrt(-g/L)))
```

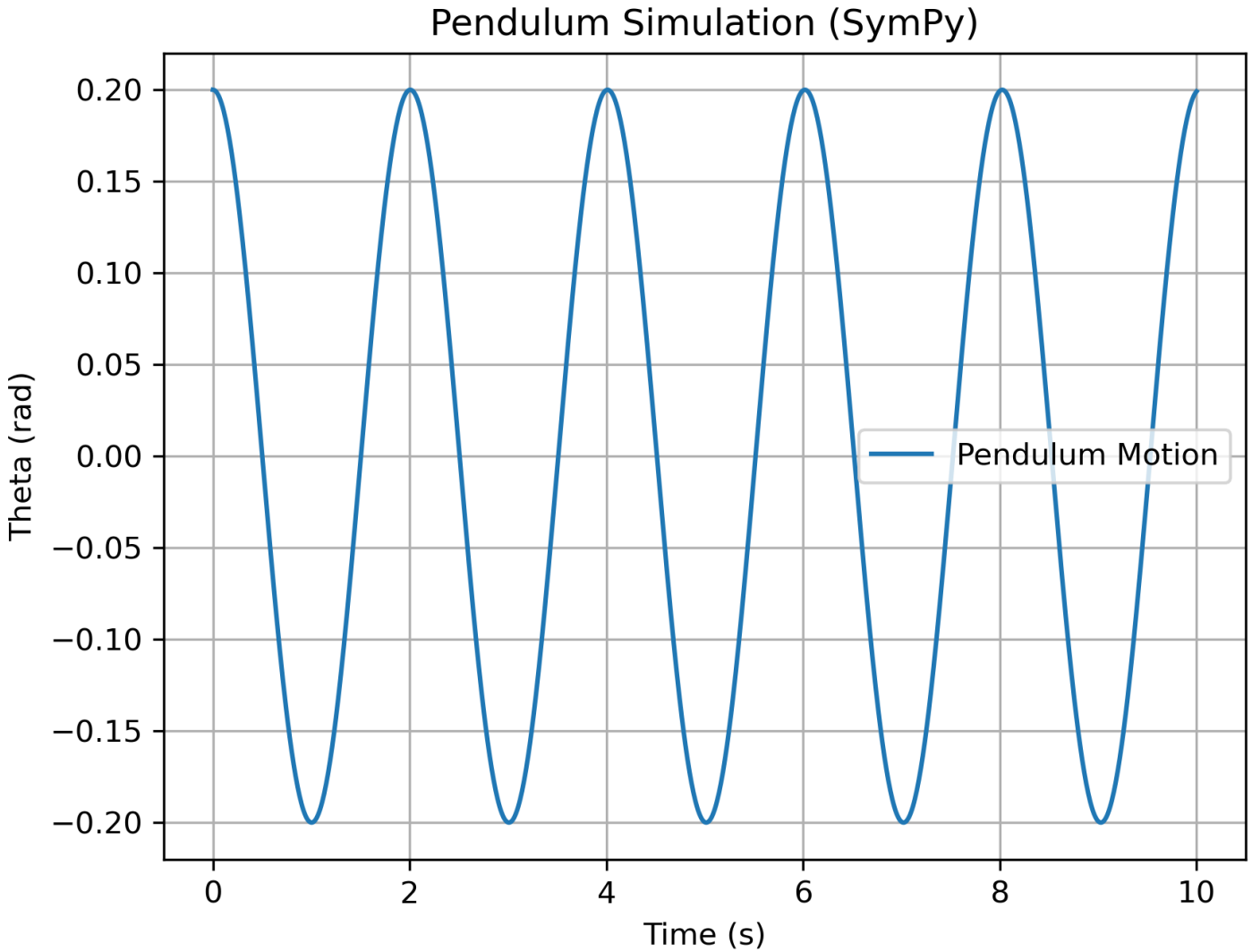
Visualizing the Solutions

- SymPy provides integration with Matplotlib to plot the results.

➢ Example Visualization of the Pendulum Solution

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

# Define symbolic variables
t = sp.Symbol('t')  # Time
g = sp.Symbol('g')  # Gravity
L = sp.Symbol('L')  # Length of pendulum

# Define the analytical solution (Example: Small-angle approximation)
theta_0 = 0.2  # Initial angle (radians)
omega = sp.sqrt(g / L)  # Natural frequency
simplified_solution = theta_0 * sp.cos(omega * t)  # Solution for small angles
```

```
# Substitute parameters
params = {g: 9.81, L: 1}  # Gravity = 9.81 m/s², Length = 1m
sol_expr = simplified_solution.subs(params)

# Convert symbolic solution to numerical function
theta_func = sp.lambdify(t, sol_expr, modules=['numpy'])

# Generate time values
time = np.linspace(0, 10, 1000)  # 10 seconds, 1000 points
theta_vals = theta_func(time)

# Plot the results
plt.plot(time, theta_vals, label="Pendulum Motion")
plt.title("Pendulum Simulation (SymPy)")
plt.xlabel("Time (s)")
plt.ylabel("Theta (rad)")
plt.grid()
plt.legend()
plt.show()
```


Pendulum Simulation (SymPy)
Theta (rad)
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
2
4
6
8
10
Time (s)
Pendulum Motion

SymPy Code for Solving the RC Circuit Equation

```
from sympy import symbols, Function, Eq, dsolve, simplify
# Define variables and functions
t = symbols('t')
V_C = Function('V_C')(t)
R, C, V_in = symbols('R C V_in')
# Define the RC circuit equation
rc_eq = Eq(V_C.diff(t), (V_in - V_C) / (R * C))
# Solve the differential equation
rc_solution = dsolve(rc_eq, V_C)
simplified_rc_solution = simplify(rc_solution)

# Display the solution
print("RC Circuit Solution:")
print(simplified_rc_solution)
```

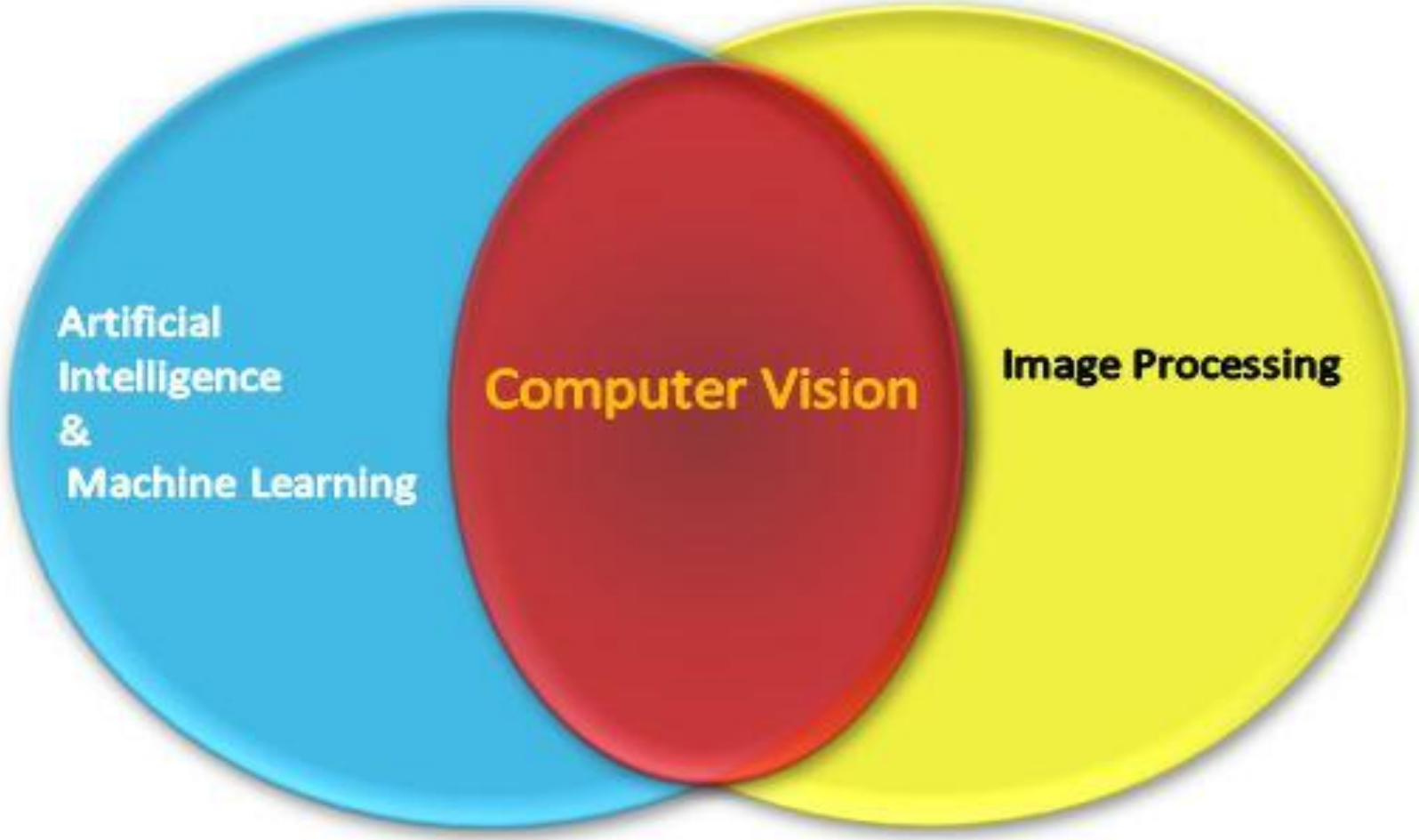
✓ **Output**:

```
RC Circuit Solution:
Eq(V_C(t), C1*exp(-t/(C*R)) + V_in)
```

Introduction to OpenCV or similar tools for basic image analysis in engineering

Computer Vision (CV)

- Computer Vision is a branch of Computer Science, which aims to build up intelligent systems that can understand the content in images as they are perceived by humans. The data may be presented in different modalities such as sequential (video) images from multiple sensors (cameras) or multidimensional data from a biomedical camera, and so on. It is the discipline that integrates the methods of acquiring, processing, analyzing and understanding large-scale images from the real world.
- Computer Vision is one aspect of Artificial Intelligence and Image Processing, which generally aims to simulate intelligent human capabilities. In computer Vision concept, object recognition is one of the fundamental tasks, which depends on how these objects are defined, whether in the form of images or video sequences, and human beings are able to recognize many entities, even if these objects, which are images, vary greatly in size and lighting.


Artificial
Intelligence
&
Machine Learning
Computer Vision
Image Processing

Some examples of Computer Vision applications:

- Any application that can recognize objects or humans in an image;
- Automatic control applications (industrial robots, vehicles);
- Object construction models (industrial inspection, medical image analysis);
- Applications make it possible to track a moving object.

Computer vision tasks:

- There are many computer vision tasks including Image Recognition, Semantic Segmentation, Image Retrieval, Image Restoration, Object Recognition, Video Tracking, and so on.

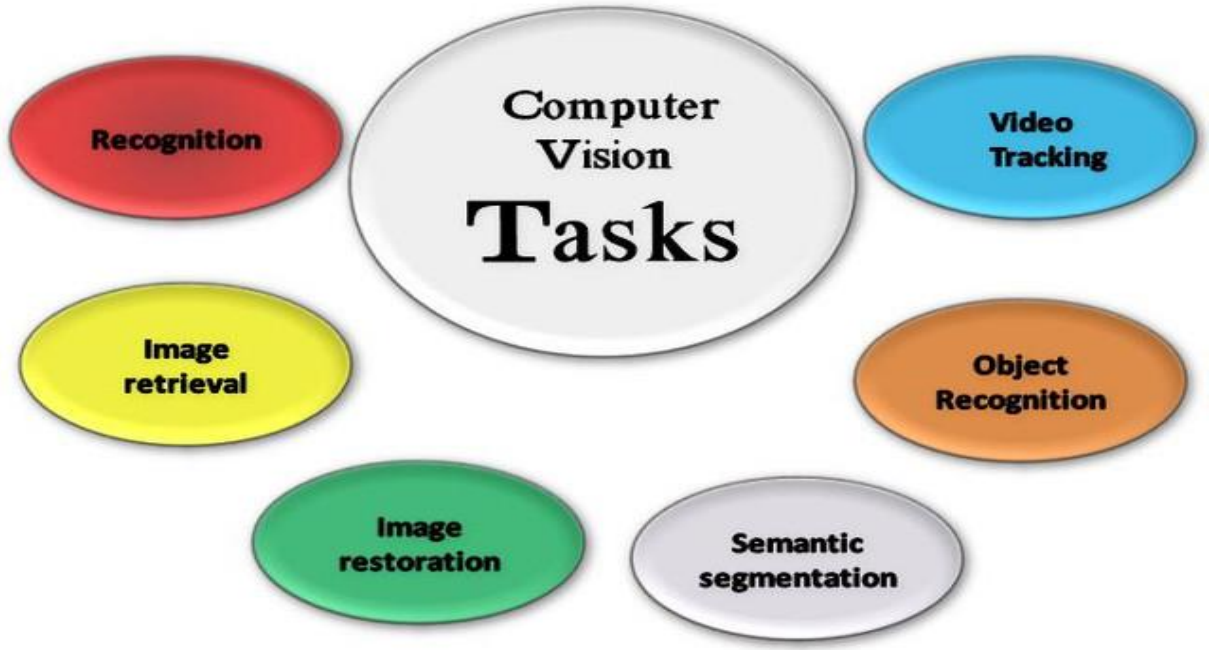


Image Recognition

- Traditionally, Computer Vision is about deciding whether or not the image contains an object.

- This task can be solved simply with little effort by human beings, but a certain activity is still not solved effectively and finely by computer in its general state.

- The only way to solve this issue is to find the best solutions to match certain features (edges, shapes, etc), and in some cases only, often with specific lighting conditions, a background and a certain position for the camera.

Types of recognition:

A - Identification: Predefined objects are often identified from different viewpoints of the camera in their different locations.

B - Selection: Define a unique identifier in the shape. For example: identify a person's face or identify the specific type of a person or car.

C - Examination: Image data is treated for a specific object. For example: check for the presence of diseased cells in medical form, check if a car is present on a highway.

Computer Vision Systems

- Computer vision systems are very diverse and are divided into large and sophisticated systems that perform general and complete tasks as well as small systems that perform specific and simple ones. Most computer vision systems mainly include the following:

➢Collecting images

- The image is generated by using one or more image sensors. These include many digital camera sensors, distance sensors, radars, and ultrasonic cameras.

➢ Pre-processing operations

- Before applying the computer vision algorithm in order to extract valuable information, it is necessary to perform prior data operations to ensure that the data are consistent with the algorithm's specific hypotheses. Some examples of these processes include:

1. Select the image resolution to confirm that its coordinate system is correct.
2. Reduce the interference to ensure that the sensor does not provide inaccurate information.
3. Increase the variance in order to ensure that the required information will be available.

➢ Features extraction

- Visual data features are extracted at different levels of abstraction from data raw. These benchmarks are categorized into:

1. Global features such as color and shape.
2. Local features such as edges and points. More complex features related to colors and patterns can be obtained.

➢ Segmentation

- All zones of the image can be recognized as important locations for subsequent operations. For example: select a set of key points, divide one or more images that contain the region of interest. 3.5. High-level processing operations At this stage, the input data consists of a small set of data, such as a set of points or a portion of the image that is suspected to contain the interest object. The other operations are:

1. Ensure that the collected data are consistent with the hypotheses of the intended application.
2. Evaluate the transaction values assigned to the request, such as steering or shape size.
3. Classify the recognized objects into several classes

Python libraries for Computer Vision

- The main toolkits for image processing in python are OpenCV, scikit-image and Pillow.
- The most general Python libraries (Numpy and Scipy) also provide some image processing tools.
- All these libraries can easily dialog with each other due to the common use of Numpy arrays to store images.
- A grayscale image is usually stored in a 2-dimensional integer or real value Numpy array with H rows and W columns (W=width,H=height). A color image is stored in a 3-dimensional Numpy array (H, W, 3).

Installing and Importing the OpenCV Image Preprocessing Package

- OpenCV in deep learning is an extremely important important aspect of many Machine Learning algorithms. OpenCV basics is an open-source library (package) for computer vision, machine learning, and image processing applications that run on the CPU exclusively. It works with many different programming languages, including Python. It can be imported with single line command as being depicted below.

✓pip install opencv-python

- A package in Python is a collection of modules that contain pre-written programmes. These packages allow you to import modules separately or in their whole. Importing the package is as simple as calling the “cv2” module as seen below:

✓import cv2 as cv

Processing images with OpenCV

- Reading images in Python

To read an image, we have the imread () function. It should be mentioned that previously, we have moved to the directory that contains the image.

image = cv2.imread ('image.jpg')

- As an alternative, it is also possible to pass a value for a flag, which is the second argument

✓cv2.IMREAD_COLOR: For loading a color image by overlooking existing transparency; cv2.IMREAD_GRAYSCALE: For loading a grayscale image; cv2.IMREAD_UNCHANGED: For loading an image that includes an alpha channel It is possible to use integers 1, 0 or -1:

image = cv2. imread ('image.jpg', 0)

- ***Note that sending an invalid image path does not result in any errors.***

Displaying images in Python

- The cv2.imshow () function enables to display an image in a frame that can be adjusted to its size. The first argument is the name of the frame and the second one is the image.

✓image = cv2. imread ('image.jpg')

✓cv2.imshow('Images', image)

- Note that we have two frames at once as we have not attempted to title them in the same way. cv2.destroyAllWindows () function is another function that destroys all the frames that we have already created. cv2.destroyWindow () also destroys a specific frame

Creating images in Python

- To do this, there is the cv2.imwrite ()function. The first argument is the file name and the second one is the image to be saved.

✓cv2.imwrite('image.png', image)

This will store the grayscale image named "image.png" in the current location.

Displaying images using Matplotlib

- By using Matplotlib (opens new window)library, we can display that image.

```
import matplotlib.pyplot as plt
plt.imshow(image, cmap = "gray", interpolation = "bilinear")
plt.xticks([]), pl.ticks ([])
(([], ), ([], ))
plt.display ()
```

Core operations on images

❑Let's now look at the basic operations applicable on the image.

```
import cv2
image = cv2.imread (' image.jpg')
y, x = 100,50
```

▪ Reading of color values at positions y, x:

```
(b, g, r) = image[y,x]
```

- Region of interest at (x, y) whose dimensions are 100x100:

```
roi = image[y:y+100,x:x+100]
cv2.imshow ('image', image)
cv2.imshow('ROI', roi)
```

- Pixelization of the new color:

```
roi[:,:]= (55,44,87)
cv2.imshow('New image', image)
```

Practical examples

Reading an Image

- First of all, we will import cv2 module and then read the input image using cv2’s imread() method. Then extract the height and width of the image.

```
# Importing the OpenCV library
import cv2
# Reading the image using imread() function
image = cv2.imread('image.jpg')

# Creating GUI window to display an image on screen
cv2.imshow("My first image", image)
cv2.waitKey(0)
```

✓**Output:**

```
# Extracting the height and width of an image
h, w = image.shape[:2]
# Displaying the height and width
print("Height = {}, Width = {}".format(h, w))
```

✓**Output**:

```
Height = 735, Width = 1100
```

Extracting the BGR (Blue-Green-Red)Values of a Pixel

- Now we will focus on extracting the RGB values of an individual pixel. OpenCV arranges the channels in BGR order. So the 0th value will correspond to the Blue pixel and not the Red.

```
# Extracting RGB values.
# Here we have randomly chosen a pixel
# by passing in 100, 100 for height and width.
(B, G, R) = image[100, 100]
# Displaying the pixel values
print("R = {}, G = {}, B = {}".format(R, G, B))
```

✓**Output**:

R = 239, G = 200, B = 157

```
# We can also pass the channel to extract
# the value for a specific channel
B = image[100, 100, 0]
print("B = {}".format(B))
```

✓**Output**:

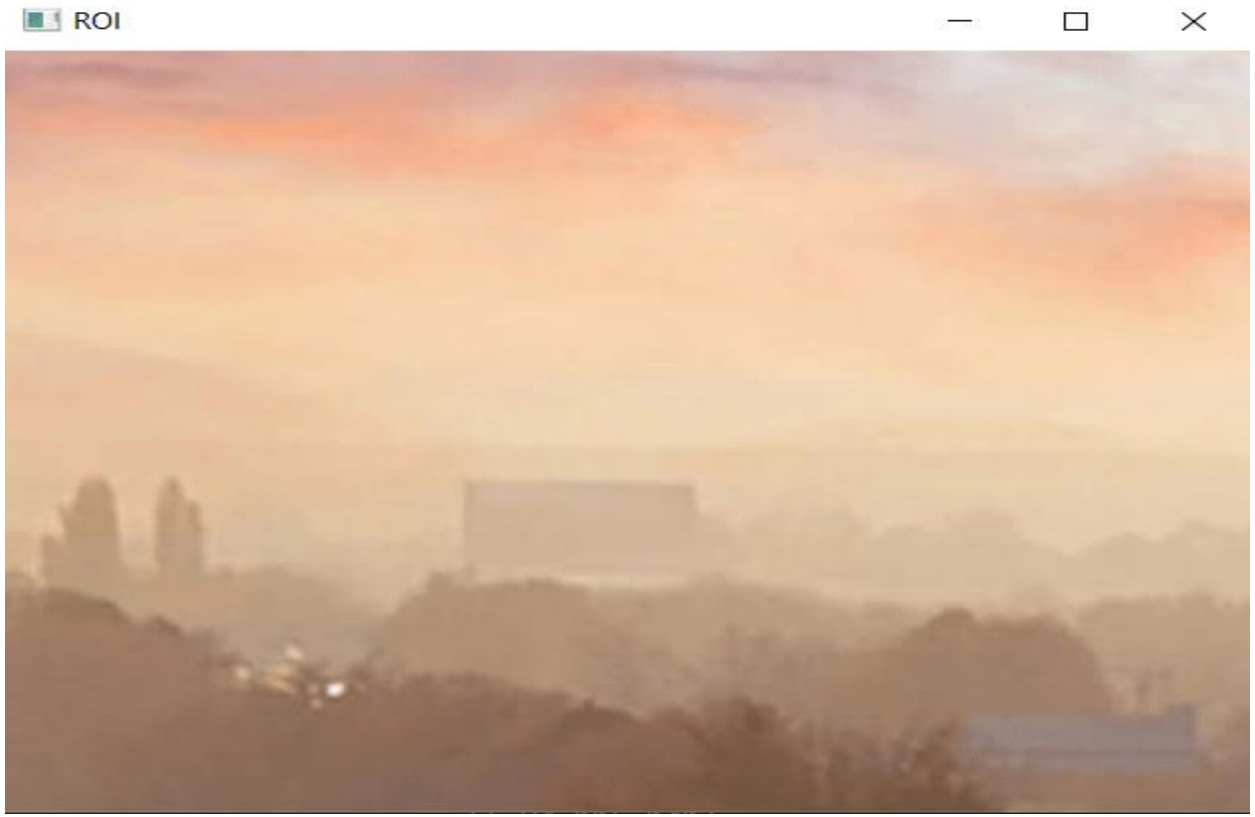
```
B = 157
```

Extracting the Region of Interest (ROI)

- Sometimes we want to extract a particular part or region of an image. This can be done by slicing the pixels of the image

```
# We will calculate the region of interest
# by slicing the pixels of the image
roi = image[100 : 500, 200 : 700]
cv2.imshow("ROI", roi)
cv2.waitKey(0)
```

✓**Output**:

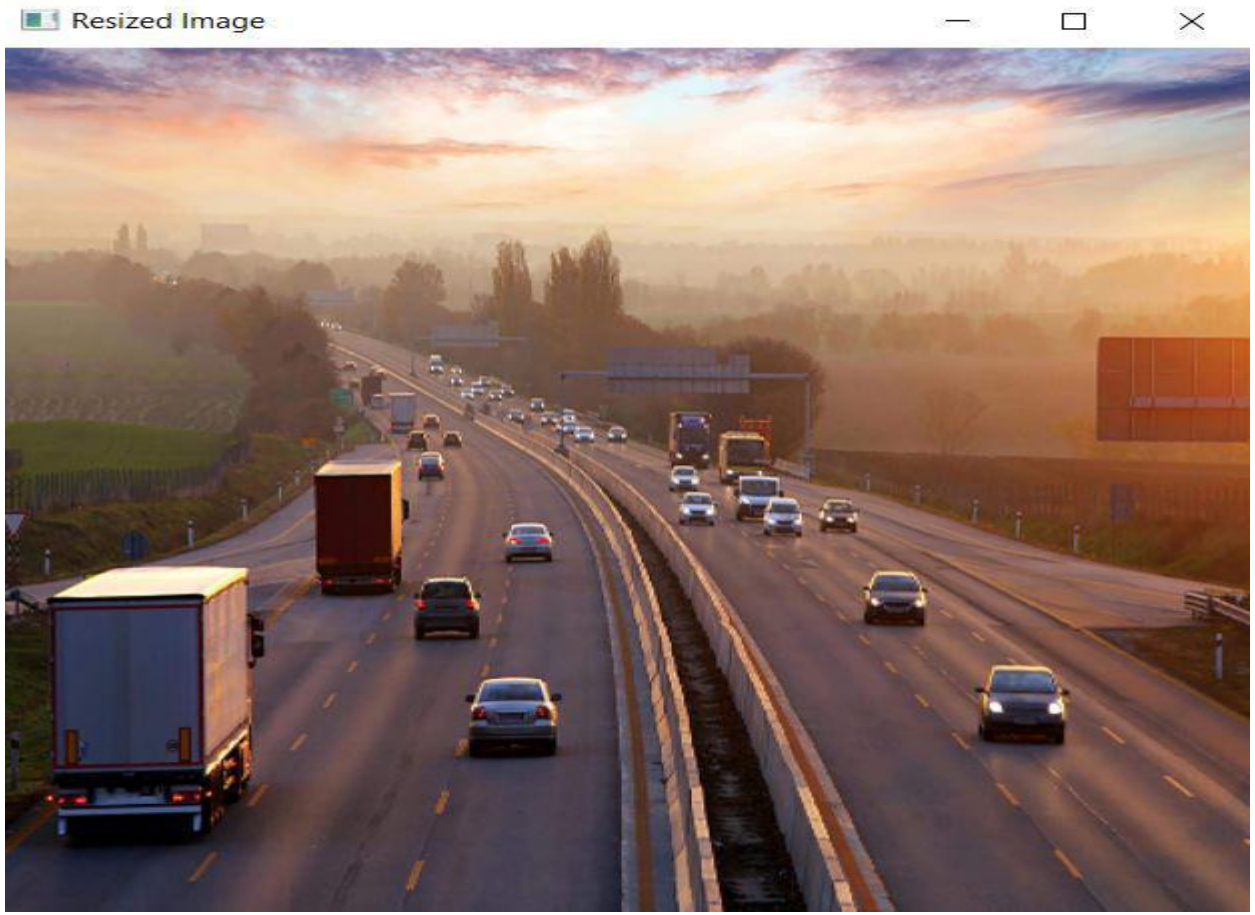


Resizing the Image

- We can also resize an image in Python using resize() function of the cv2 module and pass the input image and resize pixel value.

```
# resize() function takes 2 parameters,
# the image and the dimensions
resize = cv2.resize(image, (500, 500))
cv2.imshow("Resized Image", resize)
cv2.waitKey(0)
```

✓**Output:**



- The problem with this approach is that the aspect ratio of the image is not maintained. So we need to do some extra work in order to maintain a proper aspect ratio.
- This ensures that the image does not get distorted when resized.

```
# Calculating the ratio
ratio = 800 / w

# Creating a tuple containing width and height
dim = (800, int(h * ratio))

# Resizing the image
resize_aspect = cv2.resize(image, dim)
cv2.imshow("Resized Image", resize_aspect)
cv2.waitKey(0)
```

✓**Output:**

Drawing a Rectangle

- We can draw a rectangle on the image using rectangle() method. It takes in 5 arguments:

- Image
- Top-left corner co-ordinates
- Bottom-right corner co-ordinates
- Color (in BGR format)
- Line width

```
# We are copying the original image,
# as it is an in-place operation.
output = image.copy()

# Using the rectangle() function to create a rectangle.
rectangle = cv2.rectangle(output, (1500, 900), (600, 400), (255, 0, 0), 2)
cv2.imshow("Drawing a Rectangle", rectangle)
cv2.waitKey(0)
```

✓**Output:**

Displaying text

- It is also an in-place operation that can be done using the putText() method of OpenCV module. It takes in 7 arguments:

- Image
- Text to be displayed
- Bottom-left corner co-ordinates, from where the text should start
- Font
- Font size
- Color (BGR format)
- Line width

```
# Copying the original image
output = image.copy()


# Adding the text using putText() function
text = cv2.putText(output, 'OpenCV Demo', (500, 550), cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 0, 0), 2)
cv2.imshow("Displaying text", text)
cv2.waitKey(0)
```

✓**Output:**