

FAST APPROACH FOR EFFICIENT VEHICLE COUNTING

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in ELECTRONICS AND COMMUNICATION ENGINEERING

By

JONNALAGADDA LEELA VENKATA SIVA PRASAD(38130091)

GUJJULA HEMANTH REDDY(38130076)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

SCHOOL OF ELECTRONICS & COMMUNICATION

ENGINEERING



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project is the bonafide work of JONNALAGADDA
LEELA VENKATA SIVA PRASAD(38130091)& GUJJULA HEMANTH
REDDY(38130076) carried out the project entitled as "FAST APPROACH FOR
EFFICIENT VEHICLE COUNTING." under our supervision from 09/12/2021 to 01/03/2022.

Internal Guide

DR.VINO

External Guide

Head of the Department

(DR. T. RAVI)

*Submitted for the viva voice exam held on 04/05

Internal examiner

DECLARATION

External examiner

We JONNALAGADDA LEELA VENKATA SIVA PRASAD & GUJJULA HEMANTH REDDY hereby declare that the project report entitled "FAST APPROACH FOR EFFICIENT VEHICLE COUNTING "done by us under the guidance of ----- and ----- at SATHYABAMA UNIVERSITY is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering in ELECTRONICS AND COMMUNICATION ENGINEERING.

Date:- 21/04/2022
VIJAYAWADA

Signature of the candidate Place:-
1.
2.

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to DR. N. M. NANDHITHA, Dean, School of Electrical Engineering and DR. T. RAVI, Head of the Department, Department of ELECTRONICS AND COMMUNICATION ENGINEERING for providing me necessary support and details at the right time during the progressive reviews

I would like to express my sincere and deep sense of gratitude to my Project Guide DR.T.VINO sir for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of E.C.E who were helpful in many ways for the completion of the project.

CONTENTS

CHAPTER: 1

INTRODUCTION

CHAPTER:2

LITERATURE SURVEY

CHAPTER: 3

3.1 EXISTING SYSTEM 3.2 PROPOSED SYSTEM 3.3 BLOCK DIAGRAM

3.4 SOFTWARE REQUIREMENTS **CHAPTER:**

4

4.1 MORE MODULES

4.1.1 EXECUTING MODULES AS SCRIPTS

4.1.2 THE MODULE SEARCH PATH

4.1.3 COMPILED PYTHON FILES

4.2 STANDARD MODULES 4.3 THE DIR() FUNCTION

4.4 PACKAGES

4.4.1 IMPORTING *FROMA PACKAGES

4.4.2 INTRA PACKAGE REFERENCES

4.4.3 PACKAGES IN MULTIPLE DIRECTORIES

CHAPTER: 5

5.1 CODE DESCRIPTION 5.2 PERFORMANCE & RESULTS

CHAPTER: 6

CONCLUSION

Fast approach for efficient vehicle counting

Abstract:

- Vehicles have been a big part of many lives; from the time it is invented and as it increases popularity in the 20th century. Though they offer the benefit of convenience, they also have certain negative effects as they add to air pollution and global warming, as well as risks when they are not handled properly. In recent years, the number of vehicles on the road is rapidly increasing and it causes different major concerns.
- One of the major effects of this increasing volume of vehicles is the traffic congestion it caused on our roads especially in the urban areas. This traffic congestion became one of the major problems in many cities in the world including Metro Manila, Philippines. Many options are discussed and implemented by the traffic management but it seems that it is still unsolved.

- In recent years, traffic congestions became unpredictable, there are parts of the cities that don't experience traffic congestion then suddenly traffic builds up to that area. Also, traffic congestion might happen every hour of the day.
- With this concern, the study proposed a system for vehicle headlight recognition for on-road vehicle detection and counting. This study focused on the detection of the headlight of every vehicle that will be seen on the perimeter of the installed camera.
- The system can detect headlight vehicles during daytime and nighttime as we trained the AI to recognized the headlights in these scenarios. Possible applications of this system can be in monitoring the volume of vehicles within the area and it can be used by traffic management authority in monitoring the buildup of traffic or traffic situation in an area so that they can provide an immediate solution, such as re-routing, one-way street conversions, etc.

Chapter 1

Introduction:

- Vehicle use has now become an integral part of our lives. These vehicles provide us with convenience as we travel to work and transport a variety of food and goods from different parts of the country.
- From the invention of the vehicle until it became popular in the 20th century, the increase in the number of vehicles on our roads has increased dramatically, especially in urban areas.

- Though they offer the benefit of convenience, they also have certain negative effects as they add to air pollution and global warming, as well as risks when they are not handled properly.
- In recent years, With the growing number of cars on our roads, one of its main consequences is the occurrence of severe traffic congestion. And traffic congestion nowadays becomes unpredictable, traffic can build up anytime and anywhere.

Chapter 2 Literature

survey:

1 Vehicle Detection and Tracking Techniques: A Concise Review

Raad Ahmed Hadi, Ghazali Sulong, Loay E. George

Vehicle detection and tracking applications play an important role for civilian and military applications such as in highway traffic surveillance control, management and urban

traffic planning. Vehicle detection process on road are used for vehicle tracking, counts, average speed of each individual vehicle, traffic analysis and vehicle categorizing objectives and may be implemented under different environments changes.

2 Automatic Vehicle Identification by Plate Recognition

Serkan Ozbay, and Ergun Ercelebi

Automatic Vehicle Identification (AVI) has many applications in traffic systems (highway electronic toll collection, red light violation enforcement, border and customs checkpoints, etc.). License Plate Recognition is an effective form of AVI systems. In this study, a smart and simple algorithm is presented for vehicle's license plate recognition system. The proposed algorithm consists of three major parts: Extraction of plate region, segmentation of characters and recognition of plate characters.

3 Vehicle Identification Based on the Model

T. Sridevi; P. Swapna; Kuruva Harinath

The main objective of this project is vehicle identification. Vehicles can be identified by performing recognition of its iconic license plate but the License Plate Recognition System does not work when license plate is manipulated, missing or covered. Another important attribute of a vehicle is its logo which contains important information about the car and as it cannot be changed easily. Logo plays an important role in recognition of vehicles.

4 Traffic Surveillance using Vehicle License Plate Detection and Recognition in Bangladesh

Md. Saif Hassan Onim¹ , Muhaiminul Islam Akash² , Mahmudul Haque³ , Raiyan Ibne Hafiz

Computer vision coupled with Deep Learning (DL) techniques bring out a substantial prospect in the field of traffic control, monitoring and law enforcing activities. This paper presents a YOLOv4 object detection model in which the Convolutional Neural Network (CNN) is trained and tuned for detecting the license plate of the vehicles of Bangladesh and recognizing characters using tesseract from the detected license plates. Here we also present a Graphical User Interface (GUI) based on Tkinter, a python package.

5 Matlab Based Vehicle Number Plate Recognition

1Ms. Shilpi Chauhan and 2Vishal Srivastava

The ANPR (Automatic Number plate Recognition) system is based on image processing technology. It is one of the necessary systems designed to detect the vehicle number plate. In today's world with the increasing number of vehicle day by day it's not possible to manually keep a record of the entire vehicle. With the development of this system it becomes easy to keep a record and use it whenever required. The main objective here is to design an efficient automatic vehicle identification system by using vehicle number plate

6 An automated license plate detection and recognition system based on wavelet decomposition and CNN

Ibtissam Slimani, Abdelmoghith Zaarane, Wahban Al Okaishil, ssamAtouf,

Abdellatif Hamdoun

The License Plate detection and recognition (LPDR) is a challenging task that plays a significant role in intelligent transportation systems (ITS). Where it could be used as a core in various applications, such as security, traffic control, and electronic payment systems (e.g. freeway toll payment and parking fee payment). A variety of algorithms are developed for this work and each one has advantages and disadvantages for extracting plates in images under different circumstances. However, the complexity of some methods requires a high calculation cost and this could be time-consuming.

Chapter 3

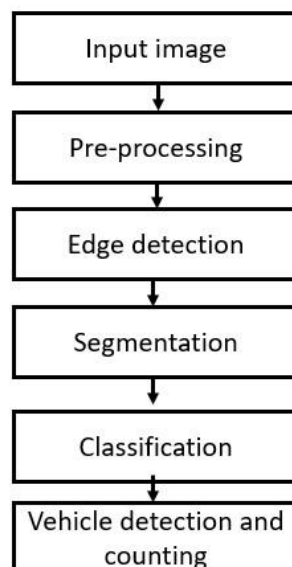
Existing system

- In existing to done using Matlab, here there is no live recording of road side, they used data set of images to identifies the vehicle.
- In existing system can be said that one of the most important researches in object detection fields, which has resulted in the auto scope video detection systems is introduced in.

Proposed system:

- The system proposed in the study, when integrated into CCTV Camera can be used in measuring the volume of the vehicles as well as traffic monitoring.
- Other studies, used the whole vehicle as their object of interest for object detection, while the proposed study only used the headlight for object detection, which makes the detection of the vehicle during nighttime can be possible and more accurate.
- Like other studies, this proposed system has limitations and enhancement. Other vehicles captured by the camera are not detected, however, increasing the number of datasets as well as increasing the anchor boxes and decreasing the threshold might generate a better result.
- Additional datasets can increase the detection and precision accuracy of the system. Also, hardware, where this system can be integrated, can be created to make this system a whole package application. Or a mobile software application, so anyone anywhere can report the traffic condition or vehicle volume on a certain place.

Block diagram:



SOFTWARE REQUIREMENTS

Introduction

In this tutorial, we are going to learn how we can perform image processing using the Python language. We are not going to restrict ourselves to a single library or framework; however, there is one that we will be using the most Open CV frequently, the library. We will start off by talking a little about image processing and then we will move on to see different applications/scenarios where image processing can come in handy. So, let's begin!

What is Image Processing?

It is important to know what exactly image processing is and what is its role in the bigger picture before diving into its how's. Image Processing is most commonly termed as 'Digital Image Processing' and the domain in which it is frequently used is 'Computer Vision'. Don't be confused - we are going to talk about both of these terms and how they connect. Both Image Processing algorithms and Computer Vision (CV) algorithms take an image as input; however, in image processing, the output is also an *image*, whereas in computer vision the output can be some *features/information about* the image.

Why do we need it?

The data that we collect or generate is mostly raw data, i.e. it is not fit to be used in applications directly due to a number of possible reasons. Therefore, we need to analyze it first, perform the necessary pre-processing, and then use it.

For instance, let's assume that we were trying to build a cat classifier. Our program would take an image as input and then tell us whether the image contains a cat or not. The first step for building this classifier would be to collect hundreds of cat pictures. One common issue is that all the pictures we have scraped would not be of the same size/dimensions, so before feeding them to the model for training, we would need to resize/pre-process them all to a standard size.

This is just one of many reasons why image processing is essential to any computer vision application.

Prerequisites

Before going any further, let's discuss what you need to know in order to follow this tutorial with ease. Firstly, you should have some basic programming knowledge in any language. Secondly, you should know what machine learning is and the basics of how it works, as we will be using some machine learning algorithms for image processing in this article. As a bonus, it would help if you have had any exposure to, or basic knowledge of, Open CV before going on with this tutorial. But this is not required.

One thing you should definitely know in order to follow this tutorial is how exactly an image is represented in memory. Each image is represented by a set of pixels i.e. a matrix of

pixel values. For a grayscale image, the pixel values range from 0 to 255 and they represent the intensity of that pixel. For instance, if you have an image of 20 x 20 dimensions, it would be represented by a matrix of 20x20 (a total of 400-pixel values).

If you are dealing with a colored image, you should know that it would have three channels - Red, Green, and Blue (RGB). Therefore, there would be three such matrices for a single image.

Installation

Note: Since we are going to use OpenCV via Python, it is an implicit requirement that you already have Python (version 3) already installed on your workstation.

Windows

```
$ pip install opencv-python
```

MacOS

```
$ brew install opencv3 --with-contrib --with-python3
```

Linux

```
$ sudo apt-get install libopencv-dev python-opencv
```

To check if your installation was successful or not, run the following command in either a Python shell or your command prompt:

```
import cv2
```

Some Basics You Should Know

Before we move on to using Image Processing in an application, it is important to get an idea of what kind of operations fall into this category, and how to do those operations. These operations, along with others, would be used later on in our applications. So, let's get to it.

For this article we'll be using the following image:



Note: The image has been scaled for the sake of displaying it in this article, but the original we are using is about 1180x786.

probably noticed that the image is currently colored, which means it is represented in three color channels i.e. Red, Green, and Blue. We will be converting the image to grayscale, as well as splitting the image into its individual channels using the code below.

Getting Image Details

After loading the image with the `imread()` function, we can then retrieve some simple properties about it, like the number of pixels and dimensions:

```
import cv2
```

```
img = cv2.imread('rose.jpg')
```

```
print("Image Properties") print(
```

```
" + str (img.size))
```

```
print(" - Shape/Dimensions: " + str (img.shape))
```

Output:

```
Image Properties
```

- Number of Pixels: 2782440

- Shape/Dimensions: (1180, 786, 3)

Splitting an Image into Individual Channels

Now we'll split the image in to its red, green, and blue components using OpenCV and display them:

```
from google.colab.patches import cv2_imshow
blue, green, red = cv2.split(img)
```

```
img_gs = cv2.imread('rose.jpg', cv2.IMREAD_GRAYSCALE) # Convert image to grayscale
```

```
cv2_imshow(red) # Display the red channel in the image
cv2_imshow(blue) # Display the blue channel in the image
cv2_imshow(green) # Display the green channel in the image
cv2_imshow(img_gs) # Display the grayscale version of image
```

For brevity, we'll just show the grayscale image.

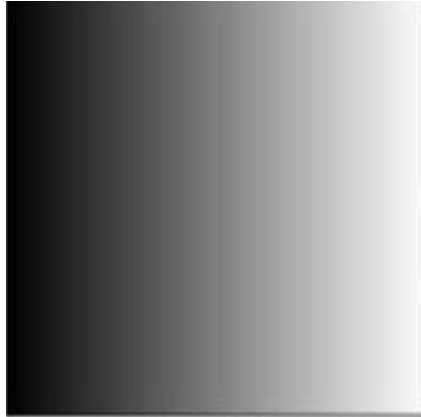
Grayscale Image:



Image Thresholding

The concept of thresholding is quite simple. As discussed above in the image representation, pixel values can be any value between 0 to 255. Let's say we wish to convert an image into a binary image i.e. assign a pixel either a value of 0 or 1. To do this, we can perform thresholding. For instance, if the Threshold (T) value is 125, then all pixels with values greater than 125 would be assigned a value of 1, and all pixels with values lesser than or equal to that would be assigned a value of 0. Let's do that through code to get a better understanding.

Image used for Thresholding:



```
import cv2
```

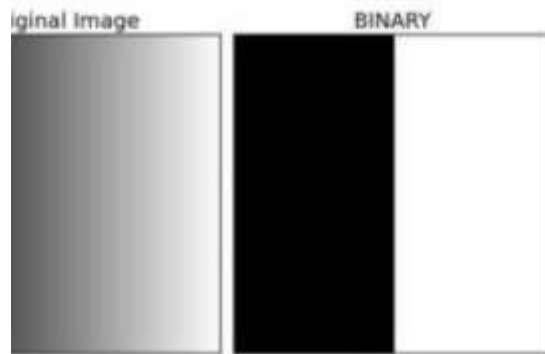
```
# Read image img = cv2.imread('img.jpg')
```

```
# Perform binary thresholding on the image with T = 125
```

```
_, threshold = cv2.threshold(img, 125, 255, cv2.THRESH_BINARY)
```

```
cv2.imshow('threshold', threshold)
```

Output:



As we can see, in the resultant image, two regions have been established, i.e. the black region (pixel value 0) and white region (pixel value 1). Turns out, the threshold we set is in the middle of the image, which is why the black and white values are divided.

Removing Noise from an Image

Now that you have got a basic idea of what image processing is and what it is used for, let's move ahead and learn about some of its specific applications.

In many cases, the raw data that we gather has noise in it i.e. unwanted features that make the image hard to perceive. Although these images can be used directly for feature extraction, the accuracy of the algorithm would suffer greatly. This is why image denoising is applied to the image *before* passing it to the algorithm to get better results.

There are many different types of noise, like Gaussian noise, salt and pepper noise, etc. To remove that noise from an image by applying a filter which removes that noise, or at very least, minimizes its effect. There are a lot of options when it comes to filters as each of them has different strengths, and hence is the best for a specific kind of noise.

Let's

Understand this properly, we are going to add 'salt and pepper' noise to the grayscale

Applications of the rose image that we considered above, and then try to remove that noise to improve accuracy. We will add noisy image using different filters and see which one is best-fit for that type.

```
import cv2
import numpy as np
```

```
# Adding salt & pepper noise to an image
```

```
def salt_pepper(prob):
```

```
    # Extract image dimensions
```

```
    row, col = img_gs.shape
```

```
    # Declare salt & pepper noise ratio
```

```
    s_vs_p = 0.5
```

```
    output = np.copy(img_gs)
```

```
    # Apply salt noise on each pixel individually
```

```
    num_salt = np.ceil(prob * img_gs.size * s_vs_p)
```

```
    coords = [np.random.randint(0, i - 1, int(num_salt))
```

```
               for i in img_gs.shape]
```

```
    output[coords] = 1
```

```
    # Apply pepper noise on each pixel individually
```

```
    num_pepper = np.ceil(prob * img_gs.size * (1. - s_vs_p))
```

```
    coords = [np.random.randint(0, i - 1, int(num_pepper))
```

```
               for i in img_gs.shape]
```

```
    output[coords] = 0
```

```
    cv2_imshow(output)
```

```
    return output
```

```
# Call salt & pepper function with probability = 0.5
```

```
# on the grayscale image of rose
```

```
sp_05 = salt_pepper(0.5)
```

```
# Store the resultant image as 'sp_05.jpg'
```

```
cv2.imwrite('sp_05.jpg', sp_05)
```

Alright, we have added noise to our rose image, and this is what it looks like now:

Noisy Image:



Lets now apply different filters on it and note down our observations i.e. how well each filter reduces the noise.

Arithmetic Filter with Sharpening Kernel

```
# Create our sharpening kernel, the sum of all values must equal to one for uniformity
```

```
kernel_sharpening = np.array([[[-1, -1, -1, -1, -1],
```

```
[-1, -1, -1, -1, -1],
```

```
# Applying the sharpening kernel to the grayscale image & displaying it.
```

```
print( '\n\n--- Effects on S&P Noise Image with Probability 0.5 ---\n\n" )
```

```
# Applying filter on image with salt & pepper noise
```

```
sharpened_img = cv2.filter2D(sp_05, -1, kernel_sharpening)
```

```
cv2.imshow(sharpened_img)
```

The resulting image, from applying arithmetic filter on the image with salt and pepper noise, is shown below. Upon comparison with the original grayscale image, we can see that it brightens the image too much and is unable to highlight the bright spots on the rose

as well. Hence, it can be concluded that arithmetic filter fails to remove salt and pepper noise.

Arithmetic Filter Output:



Midpoint Filter

```
from scipy.ndimage import maximum_filter, minimum_filter
```

```
def midpoint(img):
```

```
    maxf = maximum_filter(img, (3, 3))
```

```
    minf = minimum_filter(img, (3, 3))
```

```
    midpoint = (maxf + minf) / 2
```

```
    cv2_imshow(midpoint)
```

```
print("\n\n---Effects on S&P Noise Image with Probability 0.5---\n\n")
```

```
midpoint(sp_05)
```

The resulting image, from applying the Midpoint Filter on the image with salt and pepper noise, is shown below. Upon comparison with the original grayscale image, we can see that, like the kernel method above, brightens the image too much; however, it is able to highlight the bright spots on the rose. Therefore, we can say that it is a better choice than the arithmetic filter, but still it does not recover the original image completely.

Midpoint Filter Output:



Contraharmonic Mean Filter

Note: The implementations of these filters can be found online easily and how exactly they work is out of scope for this tutorial. We will be looking at the applications from an abstract/higher level.

```
def contraharmonic_mean(img, size, Q):
    num = np.power(img, Q + 1)
    denom = np.power(img, Q)
    kernel = np.full(size, 1.0)
    result = cv2.filter2D(num, -1, kernel) / cv2.filter2D(denom, -1, kernel)
    return result
```

```
print("\n\n--- Effects on S&P Noise Image with Probability 0.5 ---\n\n")
cv2.imshow('Contraharmonic Mean', cv2.filter2D(sp_05, -1, kernel) / cv2.filter2D(sp_05, -1, kernel))
```

Contraharmonic Mean

The resulting image, from applying Harmonic Mean Filter on the image with salt and pepper noise, is the exact same image as the original image, and it highlights the bright regions in the image. The harmonic mean filter is

used to remove salt and pepper noise.

Contraharmonic Mean Filter Output:



Now that we have found the best filter to recover the original image from a noisy one, we can move on to our next application.

#2: Edge Detection using Canny Edge Detector

The rose image that we have been using so far has a constant background i.e. black, therefore, we will be using a different image for this application to better show the algorithm's capabilities. The reason is that if the background is constant, it makes the edge detection task rather simple, and we don't want that.

We talked about a cat classifier earlier in this tutorial, let's take that example forward and see how image processing plays an integral role in that.

In a classification algorithm, the image is first scanned for 'objects' i.e. when you input an image, the algorithm would find all the objects in that image and then compare them against the features of the object that you are trying to find. In case of a cat classifier, it would compare all objects found in an image against the features of a cat image, and if a match is found, it tells us that the input image contains a cat.

Since we are using the cat classifier as an example, it is only fair that we use a cat image going forward. Below is the image we will be using:

Image used for Edge Detection:



```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Declaring the output graph's size
plt.figure(figsize=(16, 16))

# Reading the image in grayscale
img_gs = cv2.imread('cat.jpg',
                    cv2.IMREAD_GRAYSCALE)
cv2.imwrite('cat_gs.jpg', img_gs)

# Apply canny edge detector algorithm on the image to find edges
edges = cv2.Canny(img_gs, 100, 300)

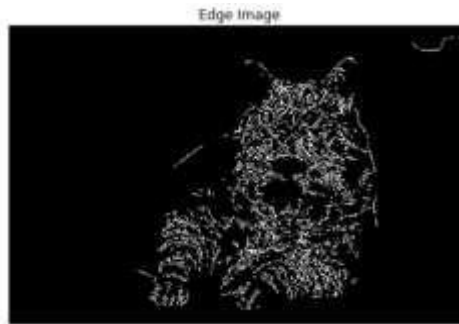
# Plot the original image against the edges
plt.subplot(121), plt.imshow(img_gs)
plt.title('Original Gray Scale Image')
plt.subplot(122), plt.imshow(edges)
plt.title('Edge Image')

# Display the two images
```

```
plt.show()
```

Edge Detection Output:

Canny Edge



cat, has been what is the now.

of the image which contains an object, which in this case is a cat. The image is processed through edge detection. Now you must be wondering, how and how did it make this happen; so let's discuss that

To understand the

th

process, there are three key steps that need to be discussed. First, it involves finding edges on the image in a similar manner that we discussed previously. To find edges, the derivative at each pixel is calculated to find edges. The logic behind this is that if an edge exists, there is an abrupt intensity change, which causes a

val

for it to be classified as

a sharp value, hence making that pixel an 'edge pixel'.

identify edges in an

image using hysteresis thresholding; we said above that there's a spike in the intensity at an edge, but we did not state 'how high' the spike needs to be to be considered an edge - this is called a threshold! Earlier in this tutorial we discussed thresholding. Hysteresis thresholding is an improvement on that, using two threshold values instead of one. The reason behind that is, if the threshold is set too low, we might miss some actual edges (true negatives) and if the threshold is set too high, we might get a lot of points classified as edges that actually are not edges. One threshold value is set high, and one is set low. All points which are above the high threshold value are identified as edges, then all points which are between the high and low threshold value are evaluated; they are either confirmed as edges or are neighbors of, points which have been identified as edges, and the rest are discarded.

Conclusion

In this tutorial, we discussed the concepts/methods that Canny Edge Detector algorithm uses

how to verify that the installation was successful.

Next, we will discuss how to install OpenCV, the most popular library for image processing, on different platforms like Windows, MacOS, and Linux, as well as

We went on to discuss what Image Processing is and its uses in the computer vision domain of Machine Learning. We talked about some common types of noise and how we

can remove it from our images using different filters, before using the images in our applications.

Furthermore, we learned how image processing plays an integral part in high-end applications like Object Detection or classification. Do note that this article was just the tip of the iceberg, and Digital Image Processing has a lot more in the store that cannot possibly be covered in a single tutorial. Reading this should enable you to dive deeper and learn about other advanced concepts related to image processing. Good Luck!

Python 3.8.0:

The Python Tutorial

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth. This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will

be ready to learn more about the various Python library modules described in The Python Standard Library.

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the [Python Package Index](#).

The Python Language Reference

This reference manual describes the syntax and “core semantics” of the language. It is terse, but attempts to be exact and complete. The semantics of non-essential built-in object types and of the built-in functions and modules are described in The Python Standard Library. For an informal introduction to the language, see The Python Tutorial. For C or C++ programmers, two additional manuals exist: Extending and Embedding the Python Interpreter describes the high-level picture of how to write a Python extension module, and the Python/C API Reference Manual describes the interfaces available to C/C++ programmers in detail.

Python Setup and Usage

This part of the documentation is devoted to general information on the setup of the Python environment on different platforms, the invocation of the interpreter and things that make working with Python easier.

Command line and environment

The CPython interpreter scans the command line and the environment for various settings.

CPython implementation detail: Other implementations' command line schemes may differ. See Alternate Implementations for further resources. **Command line**

When invoking Python, you may specify any of these options:

```
python [-bBdEhiLOqsSuvVWx?] [-c command | -m module-name | script | - ] [args]
```

The most common use case is, of course, a simple invocation of a script:

```
python myscript.py
```

Interface options

The interpreter interface resembles that of the UNIX shell, but provides some additional methods of invocation:

- When called with standard input connected to a tty device, it prompts for commands and executes them until an EOF (an end-of-file character, you can produce that with Ctrl-D on UNIX or Ctrl-Z, Enter on Windows) is read.
- When called with a file name argument or with a file as standard input, it reads and executes a script from that file.
- When called with a directory name argument, it reads and executes an appropriately named script from that directory.
- When called with `-c command`, it executes the Python statement(s) given as *command*. Here *command* may contain multiple statements separated by newlines. Leading whitespace is significant in Python statements!
- When called with `-m module-name`, the given module is located on the Python module path and executed as a script.

In non-interactive mode, the entire input is parsed before it is executed.

An interface option terminates the list of options consumed by the interpreter, all consecutive arguments will end up in `sys.argv` – note that the first element, subscript zero (`sys.argv[0]`), is a string reflecting the program's source.

-c <command>

Execute the Python code in *command*. *command* can be one or more statements separated by newlines, with significant leading whitespace as in normal module code.

If this option is given, the first element of `sys.argv` will be `"-c"` and the current directory will be added to the start of `sys.path` (allowing modules in that directory to be imported as top level modules).

Raises an auditing event `cpython.run_command` with argument `command`.

-m <module-name>

Search `sys.path` for the named module and execute its contents as the `__main__` module.

Since the argument is a *module* name, you must not give a file extension (`.py`). The module name should be a valid absolute Python module name, but the implementation may not always enforce this (e.g. it may allow you to use a name that includes a hyphen).

Package names (including namespace packages) are also permitted. When a package name is supplied instead of a normal module, the interpreter will execute `<pkg>.__main__` as the main module. This behaviour is deliberately similar to the handling of directories and zipfiles that are passed to the interpreter as the script argument.

Note

This option cannot be used with built-in modules and extension modules written in C, since they do not have Python module files. However, it can still be used for precompiled modules, even if the original source file is not available.

If this option is given, the first element of `sys.argv` will be the full path to the module file (while the module file is being located, the first element will be set to `"-m"`). As with the `-c` option, the current directory will be added to the start of `sys.path`.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the current directory nor the user's site-packages directory. All PYTHON* environment variables are ignored, too.

Many standard library modules contain code that is invoked on their execution as a script. An example is the `timeit` module:

```
python -mtimeit -s 'setup here' 'benchmarked code here' python  
-mtimeit -h # for details
```

Raises an auditing event `cpython.run_module` with argument `module-name`.

See also

`runpy.run_module()`

Equivalent functionality directly available to Python code

PEP 338 – Executing modules as scripts

Changed in version 3.1: Supply the package name to run a `__main__` submodule.

Changed in version 3.4: namespace packages are also supported

-

Read commands from standard input (`sys.stdin`). If standard input is a terminal, it is implied.

If this option is given, the first element of `sys.argv` will be `"-"` and the current directory will be added to the start of `sys.path`.

Raises an auditing event `cpython.run_stdin` with no arguments.

<script>

Execute the Python code contained in *script*, which must be a filesystem path (absolute or relative) referring to either a Python file, a directory containing a `__main__.py` file, or a zipfile containing a `__main__.py` file.

If this option is given, the first element of `sys.argv` will be the script name as given on the command line.

If the script name refers directly to a Python file, the directory containing that file is added to the start of `sys.path`, and the file is executed as the `__main__` module.

If the script name refers to a directory or zipfile, the script name is added to the start of `sys.path` and the `__main__.py` file in that location is executed as the `__main__` module.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the script's directory nor the user's site-packages directory. All PYTHON* environment variables are ignored, too.

Raises an auditing event `cpython.run_file` with argument `filename`.

CHAPTER 4

MODULES

If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and

running it with that file as input instead. This is known as creating a *script*. As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a *module*; definitions from a module can be *imported* into other modules or into the *main* module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

A module is a file containing Python definitions and statements. The file name is the module name with the suffix `.py` appended. Within a module, the module's name (as a string) is available as the value of the global variable `__name__`. For instance, use your favorite text editor to create a file called `fibonacci.py` in the current directory with the following contents:

```
# Fibonacci numbers module

def fib(n): # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
        print()

def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Now enter the Python interpreter and import this module with the following command:

```
>>>
```

```
>>> import fibo
```

This does not enter the names of the functions defined in `fibo` directly in the current symbol table; it only enters the module name `fibo` there. Using the module name you can access the functions:

```
>>>
```

```
>>> fibo.fib(1000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

If you intend to use a function often you can assign it to a local name:

```
>>>
```

```
>>> fib = fibo.fib
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

4.1. More on Modules

A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only the *first* time the module name is encountered in an import statement. 1 (They are also run if the file is executed as a script.)

Each module has its own private symbol table, which is used as the global symbol table by all functions defined in the module. Thus, the author of a module can use global variables in the module without worrying about accidental clashes with a user's global variables. On the other hand, if you know what you are doing you can touch a module's global variables with the same notation used to refer to its functions, `modname.itemname`.

Modules can import other modules. It is customary but not required to place all import statements at the beginning of a module (or script, for that matter). The imported module names are placed in the importing module's global symbol table.

There is a variant of the import statement that imports names from a module directly into the importing module's symbol table. For example:

```
>>>
```

```
>>> from fibo import fib, fib2
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

This does not introduce the module name from which the imports are taken in the local symbol table (so in the example, `fibo` is not defined).

There is even a variant to import all names that a module defines:

```
>>>
```

```
>>> from fibo import *  
>>> fib(500)  
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

This imports all names except those beginning with an underscore (`_`). In most cases Python programmers do not use this facility since it introduces an unknown set of names into the interpreter, possibly hiding some things you have already defined.

Note that in general the practice of importing `*` from a module or package is frowned upon, since it often causes poorly readable code. However, it is okay to use it to save typing in interactive sessions.

If the module name is followed by `as`, then the name following `as` is bound directly to the imported module.

```
>>>
```

```
>>> import fibo as fib  
>>> fib.fib(500)  
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

This is effectively importing the module in the same way that `import fibo` will do, with the only difference of it being available as `fib`.

It can also be used when utilising `from` with similar effects:

```
>>>
```

```
>>> from fibo import fib as fibonacci  
>>> fibonacci(500)  
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Note

For efficiency reasons, each module is only imported once per interpreter session. Therefore, if you change your modules, you must restart the interpreter – or, if it's just one module you want to test interactively, use `importlib.reload()`, e.g. `import importlib; importlib.reload(modulename)`.

4.1.1. Executing modules as scripts When

you run a Python module with

```
python fibo.py <arguments>
```

the code in the module will be executed, just as if you imported it, but with the `__name__` set to `"__main__"`. That means that by adding this code at the end of your module:

```
if __name__ == "__main__":  
    import sys    fib(int(sys.argv[1]))
```

you can make the file usable as a script as well as an importable module, because the code that parses the command line only runs if the module is executed as the “main” file:

```
$ python fibo.py 50  
0 1 1 2 3 5 8 13 21 34
```

If the module is imported, the code is not run:

```
>>>
```

```
>>> import fibo  
>>>
```

This is often used either to provide a convenient user interface to a module, or for testing purposes (running the module as a script executes a test suite).

4.1.2. The Module Search Path

When a module named `spam` is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named `spam.py` in a list of directories given by the variable `sys.path`. `sys.path` is initialized from these locations:

- The directory containing the input script (or the current directory when no file is specified).
- `PYTHONPATH` (a list of directory names, with the same syntax as the shell variable `PATH`).
- The installation-dependent default.

Note

On file systems which support symlinks, the directory containing the input script is calculated after the symlink is followed. In other words the directory containing the symlink is **not** added to the module search path.

After initialization, Python programs can modify `sys.path`. The directory containing the script being run is placed at the beginning of the search path, ahead of the standard library path. This means that scripts in that directory will be loaded instead of modules of the

same name in the library directory. This is an error unless the replacement is intended. See section Standard Modules for more information.

4.1.3. “Compiled” Python files

To speed up loading modules, Python caches the compiled version of each module in the `__pycache__` directory under the name `module.version.pyc`, where the version encodes the format of the compiled file; it generally contains the Python version number. For example, in CPython release 3.3 the compiled version of `spam.py` would be cached as `__pycache__/spam.cpython-33.pyc`. This naming convention allows compiled modules from different releases and different versions of Python to coexist.

Python checks the modification date of the source against the compiled version to see if it's out of date and needs to be recompiled. This is a completely automatic process. Also, the compiled modules are platform-independent, so the same library can be shared among systems with different architectures.

Python does not check the cache in two circumstances. First, it always recompiles and does not store the result for the module that's loaded directly from the command line. Second, it does not check the cache if there is no source module. To support a nonsource (compiled only) distribution, the compiled module must be in the source directory, and there must not be a source module.

Some tips for experts:

- You can use the `-O` or `-OO` switches on the Python command to reduce the size of a compiled module. The `-O` switch removes assert statements, the `-OO` switch removes both assert statements and `__doc__` strings. Since some programs may rely on having these available, you should only use this option if you know what you're doing. “Optimized” modules have an `opt-` tag and are usually smaller. Future releases may change the effects of optimization.
- A program doesn't run any faster when it is read from a `.pyc` file than when it is read from a `.py` file; the only thing that's faster about `.pyc` files is the speed with which they are loaded.
- The module `compileall` can create `.pyc` files for all modules in a directory.
- There is more detail on this process, including a flow chart of the decisions, in **PEP 3147**.

4.2. Standard Modules

Python comes with a library of standard modules, described in a separate document, the Python Library Reference (“Library Reference” hereafter). Some modules are built into the interpreter; these provide access to operations that are not part of the core of the language but are nevertheless built in, either for efficiency or to provide access to

operating system primitives such as system calls. The set of such modules is a configuration option which also depends on the underlying platform. For example, the winreg module is only provided on Windows systems. One particular module deserves some attention: sys, which is built into every Python interpreter. The variables sys.ps1 and sys.ps2 define the strings used as primary and secondary prompts:

```
>>> import sys
>>> sys.ps1
'>>> '
>>> sys.ps2
'...'
>>> sys.ps1 = 'C> ' C> print('Yuck!') Yuck! C>
```

These two variables are only defined if the interpreter is in interactive mode.

The variable sys.path is a list of strings that determines the interpreter's search path for modules. It is initialized to a default path taken from the environment variable PYTHONPATH, or from a built-in default if PYTHONPATH is not set. You can modify it using standard list operations:

```
>>>
```

```
>>> import sys
>>> sys.path.append('/ufs/guido/lib/python')
```

4.3. The dir() Function

The built-in function dir() is used to find out which names a module defines. It returns a sorted list of strings:

```
>>>
```

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fib', 'fib2']
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__loader__', '__name__',
'__package__', '__stderr__', '__stdin__', '__stdout__',
'__clear_type_cache__', '__current_frames__', '__debugmallocstats__', '__getframe__',
'__home__', '__mercurial__', '__xoptions__', 'abiflags', 'api_version', 'argv',
'base_exec_prefix', 'base_prefix', 'builtin_module_names', 'byteorder',
'call_tracing', 'callstats', 'copyright', 'displayhook',
'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix',
'executable', 'exit', 'flags', 'float_info', 'float_repr_style',
```

```
'getcheckinterval', 'getdefaultencoding', 'getdlopenflags',  
'getfilesystemencoding', 'getobjects', 'getprofile', 'getrecursionlimit',  
'getrefcount', 'getsizeof', 'getswitchinterval', 'gettotalrefcount',  
'gettrace', 'hash_info', 'hexversion', 'implementation', 'int_info',  
'intern', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path',  
'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'ps1',  
'setcheckinterval', 'setdlopenflags', 'setprofile', 'setrecursionlimit',  
'setswitchinterval', 'settrace', 'stderr', 'stdin', 'stdout', 'thread_info', 'version', 'version_info',  
'warnoptions']
```

Without arguments, `dir()` lists the names you have defined currently:

```
>>>  
  
>>> a = [1, 2, 3, 4, 5]  
>>> import fibo  
>>> fib = fibo.fib  
>>> dir()  
['__builtins__', '__name__', 'a', 'fib', 'fibo', 'sys']
```

Note that it lists all types of names: variables, modules, functions, etc.

`dir()` does not list the names of built-in functions and variables. If you want a list of those, they are defined in the standard module `builtins`:

```
>>>  
  
>>> import builtins  
>>> dir(builtins)  
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',  
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',  
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
```

```
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError',
'MemoryError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented',
'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'ZeroDivisionError', '_', '__build_class__',
'__debug__', '__doc__', '__import__', '__name__', '__package__', 'abs',
'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable',
'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits',
'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit',
'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass',
'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview',
'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property',
'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars',
'zip']
```

4.4. Packages

Packages are a way of structuring Python's module namespace by using "dotted module names". For example, the module name A.B designates a submodule named B in a package named A. Just like the use of modules saves the authors of different modules from having to worry about each other's global variable names, the use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other's module names.

Suppose you want to design a collection of modules (a "package") for the uniform handling of sound files and sound data. There are many different sound file formats (usually recognized by their extension, for example: .wav, .aiff, .au), so you may need to create and maintain a growing collection of modules for the conversion between the various file formats. There are also many different operations you might want to perform on sound data (such as mixing, adding echo, applying an equalizer function, creating an artificial stereo effect), so in addition you will be writing a never-ending stream of modules to perform these operations. Here's a possible structure for your package (expressed in terms of a hierarchical filesystem):

sound/	Top-level package
__init__.py	Initialize the sound package
formats/	Subpackage for file format conversions
__init__.py	
wavread.py	
wavwrite.py	
aiffread.py	
aiffwrite.py	
auread.py	auwrite.py
...	
effects/	Subpackage for sound effects
__init__.py	
echo.py	
surround.py	
reverse.py	
...	
filters/	Subpackage for filters
__init__.py	
equalizer.py	
vocoder.py	
karaoke.py	
...	

When importing the package, Python searches through the directories on `sys.path` looking for the package subdirectory.

The `__init__.py` files are required to make Python treat directories containing the file as packages. This prevents directories with a common name, such as `string`, unintentionally hiding valid modules that occur later on the module search path. In the simplest case, `__init__.py` can just be an empty file, but it can also execute initialization code for the package or set the `__all__` variable, described later.

Users of the package can import individual modules from the package, for example:

```
import sound.effects.echo
```

This loads the submodule `sound.effects.echo`. It must be referenced with its full name.

```
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

An alternative way of importing the submodule is:

```
from sound.effects import echo
```

This also loads the submodule `echo`, and makes it available without its package prefix, so it can be used as follows:

```
echo.echofilter(input, output, delay=0.7, atten=4)
```

Yet another variation is to import the desired function or variable directly:

```
from sound.effects.echo import echofilter
```

Again, this loads the submodule `echo`, but this makes its function `echofilter()` directly available:

```
echofilter(input, output, delay=0.7, atten=4)
```

Note that when using `from package import item`, the item can be either a submodule (or subpackage) of the package, or some other name defined in the package, like a function, class or variable. The import statement first tests whether the item is defined in the package; if not, it assumes it is a module and attempts to load it. If it fails to find it, an `ImportError` exception is raised.

Contrarily, when using syntax like `import item.subitem.subsubitem`, each item except for the last must be a package; the last item can be a module or a package but can't be a class or function or variable defined in the previous item.

4.4.1. Importing * From a Package

Now what happens when the user writes `from sound.effects import *`? Ideally, one would hope that this somehow goes out to the filesystem, finds which submodules are present in the package, and imports them all. This could take a long time and importing submodules might have unwanted side-effects that should only happen when the submodule is explicitly imported.

The only solution is for the package author to provide an explicit index of the package.

defines a list named `__all__`, it is taken to be the list of module names that should be imported when `from package import *` is encountered. It is up to the package author to keep this list up-to-date when a new version of the package is released. Package authors The import statement uses the following convention: if a package's `__init__.py` code may

also decide not to support it, if they don't see a use for importing `*` from their package. For

example, the file `sound/effects/__init__.py` could contain the following code:

```
__all__ = ["echo", "surround", "reverse"]
```

This would mean that `from sound.effects import *` would import the three named submodules of the sound package.

If `__all__` is not defined, the statement `from sound.effects import *` does *not* import all submodules from the package `sound.effects` into the current namespace; it only ensures that the package `sound.effects` has been imported (possibly running any initialization code in `__init__.py`) and then imports whatever names are defined in the package. This includes any names defined (and submodules explicitly loaded) by `__init__.py`. It also includes any submodules of the package that were explicitly loaded by previous import statements. Consider this code:

```
import sound.effects.echo import
sound.effects.surround from
sound.effects import *
```

In this example, the `echo` and `surround` modules are imported in the current namespace because they are defined in the `sound.effects` package when the `from...import` statement is executed. (This also works when `__all__` is defined.)

Although certain modules are designed to export only names that follow certain patterns when you use `import *`, it is still considered bad practice in production code.

Remember, there is nothing wrong with using `from package import specific_submodule!` In fact, this is the recommended notation unless the importing module needs to use submodules with the same name from different packages.

4.4.2. Intra-package References

When packages are structured into subpackages (as with the `sound` package in the example), you can use absolute imports to refer to submodules of siblings packages. For example, if the module `sound.filters.vocoder` needs to use the `echo` module in the `sound.effects` package, it can use `from sound.effects import echo`.

You can also write relative imports, with the `from module import name` form of import statement. These imports use leading dots to indicate the current and parent packages involved in the relative import. From the `surround` module for example, you might use:

```
from . import echo from ..
import formats from ..filters
import equalizer
```

Note that relative imports are based on the name of the current module. Since the name

of the main module is always `"__main__"`, modules intended for use as the main module of a Python application must always use absolute imports.

4.4.3. Packages in Multiple Directories

Packages support one more special attribute, `__path__`. This is initialized to be a list containing the name of the directory holding the package's `__init__.py` before the code in that file is executed. This variable can be modified; doing so affects future searches for modules and subpackages contained in the package.

While this feature is not often needed, it can be used to extend the set of modules found in a package.

CHAPTER 5

CODE DESCRIPTION :

Python Programming Language

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

This specially designed Python tutorial will help you learn Python Programming Language in most efficient way, with the topics from basics to advanced (like Web-scraping, Django, Deep-Learning, etc.) with examples.

Below are some facts about Python Programming Language:

1. Python is currently the most widely used multi-purpose, high-level programming language.
2. Python allows programming in Object-Oriented and Procedural paradigms.
3. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
4. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber... etc.
5. The biggest strength of Python is huge collection of standard library which can be used for the following:
 - Machine Learning
 - GUI Applications (like Kivy, Tkinter, PyQt etc.)
 - Web frameworks like Django (used by YouTube, Instagram, Dropbox)
 - Image processing (like OpenCV, Pillow)

- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia
- Scientific computing
- Text processing and many more.. **Recent Articles on Python !**

Python Programming Examples

Python Output & Multiple Choice Questions

Basics, Input/Output, Data Types, Variables, Operators, Control Flow, Functions, Object Oriented Concepts, Exception Handling, Python Collections, Django Framework, Data Analysis, Numpy, Pandas, Machine Learning with Python, Python GUI, Modules in Python, Working with Database, Misc, Applications and Projects, Multiple Choice Questions

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions: **Python 2 and Python 3**. Both are quite different.

Beginning with Python programming:

1) Finding an Interpreter:

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like

<https://ide.geeksforgeeks.org/>, <http://ideone.com/> or <http://codepad.org/> that can be used to run Python programs without installing an interpreter.

Windows: There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) that comes bundled with the Python software downloaded from <http://python.org/>.

Linux: Python comes preinstalled with popular Linux distros such as Ubuntu and Fedora. To check which version of Python you're running, type "python" in the terminal emulator. The interpreter should start and print the version number.

macOS: Generally, Python 2.7 comes bundled with macOS. You'll have to manually install Python 3 from <http://python.org/>.

2) Writing our first program:

Just type in the following code after you start the interpreter.

```
# Script Begins print("GeeksQuiz")
```

```
# Scripts Ends
```

Output:

GeeksQuiz

Let's analyze the script line by line.

Line 1: [# Script Begins] In Python, comments begin with a #. This statement is ignored by the interpreter and serves as documentation for our code.

Line 2: [print("GeeksQuiz")] To print something on the console, print() function is used. This function also adds a newline after our message is printed(unlike in C). Note that in Python 2, "print" is not a function but a keyword and therefore can be used without parentheses. However, in Python 3, it is a function and must be invoked with parentheses.

Line 3: [# Script Ends] This is just another comment like in Line 1.

Python Language advantages and applications

- Difficulty Level : Basic
- Last Updated : 30 Jun, 2021

Python is a high-level, interpreted, and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. Python ranks among the most popular and fastest-growing languages in the world. Python is a powerful, flexible, and easy-touse language. In addition, the community is very active there. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

Advantages :

1. Presence of third-party modules
2. Extensive support libraries(NumPy for numerical calculations, Pandas for data analytics etc)
3. Open source and community development
4. Versatile, Easy to read, learn and write
5. User-friendly data structures
6. High-level language
7. Dynamically typed language(No need to mention data type based on the value assigned, it takes data type)
8. Object-oriented language
9. Portable and Interactive
10. Ideal for prototypes – provide more functionality with less coding

11. Highly Efficient(Python's clean object-oriented design provides enhanced process control, and the language is equipped with excellent text processing and integration capabilities, as well as its own unit testing framework, which makes it more efficient.)
12. (IoT)Internet of Things Opportunities
13. Interpreted Language
14. Portable across Operating systems

Applications :

1. GUI based desktop applications
2. Graphic design, image processing applications, Games, and Scientific/ computational Applications
3. Web frameworks and applications
4. Enterprise and Business applications
5. Operating Systems
6. Education
7. Database Access
8. Language Development
9. Prototyping
10. Software Development

Organizations using Python :

1. Google(Components of Google spider and Search Engine)
2. Yahoo(Maps)
3. YouTube
4. Mozilla
5. Dropbox
6. Microsoft
7. Cisco
8. Spotify
9. Quora

Download and Install Python 3 Latest Version

How to Download and Install Python 3 Latest Version? In this article, you will get the answer to all your questions related to installing Python on Windows/Linux/macOS.

Python was developed by Guido van Rossum in the early 1990s and its latest version is 3.7.4, we can simply call it as Python3.

To understand how to install Python You need to know *What Python is* and where it is actually installed in your system.

Let's consider a few points:

- Python is a widely-used general-purpose, high-level programming language.
- Every Release of Python is open-source. Python releases have also been GPLcompatible.

- Any version of Python can be downloaded from **Python Software Foundation** website at python.org.
- Most of the languages, notably Linux provide a package manager through which you can directly install Python on your Operating System

In this Python tutorial of Installation and Setup, you'll see how to install Python on Windows, macOS, Linux, iOS, and Android.

Python Latest Version Installation and Setup

Here you can choose your OS and see the corresponding tutorial,

☐ [Windows](#) ☐

[Linux](#) ☐ [macOS /](#)

[Mac OS X](#) ☐

[Android](#) ☐ [iOS](#)

[\(iPhone / iPad\)](#)

☐ [Online Interpreters of Python](#)

How to install Python on Windows?

Since windows don't come with Python preinstalled, it needs to be installed explicitly. Here we will define step by step tutorial on How to install Python on Windows.

Follow the steps below :

Download Python Latest Version from python.org

- First and foremost step is to open a browser and open <https://www.python.org/downloads/windows/>



- Underneath the **Python Releases for Windows** find **Latest Python 3 Release – Python 3.7.4** (latest stable release as of now is Python 3.7.4).
- On this page move to **Files** and click on **Windows x86-64 executable installer** for 64-bit or **Windows x86 executable installer** for 32-bit.

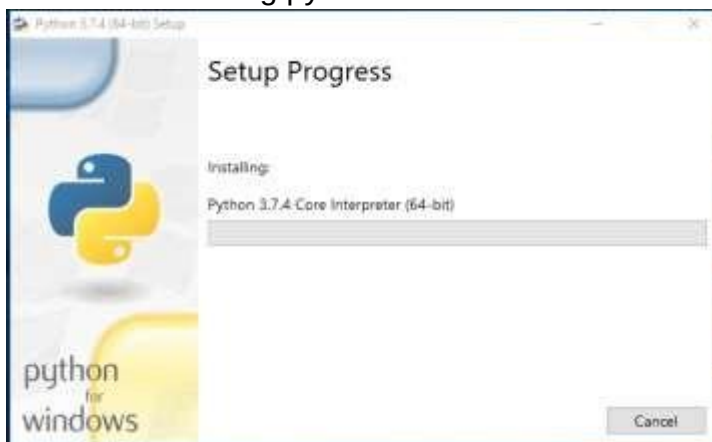
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abc83184a40729a2	7504391	SiG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76debd63043a583e563400	26680368	SiG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c508bbd73ae8e53a3bd351b4bd2	1362904	SiG
Windows x86 embeddable zip file	Windows		9fab3b81f8941879fda94133574139d0	6741626	SiG
Windows x86 executable installer	Windows		33c602042a54446a3d6451476394789	25863848	SiG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SiG

Install Python 3.7.4 Latest Version on Windows

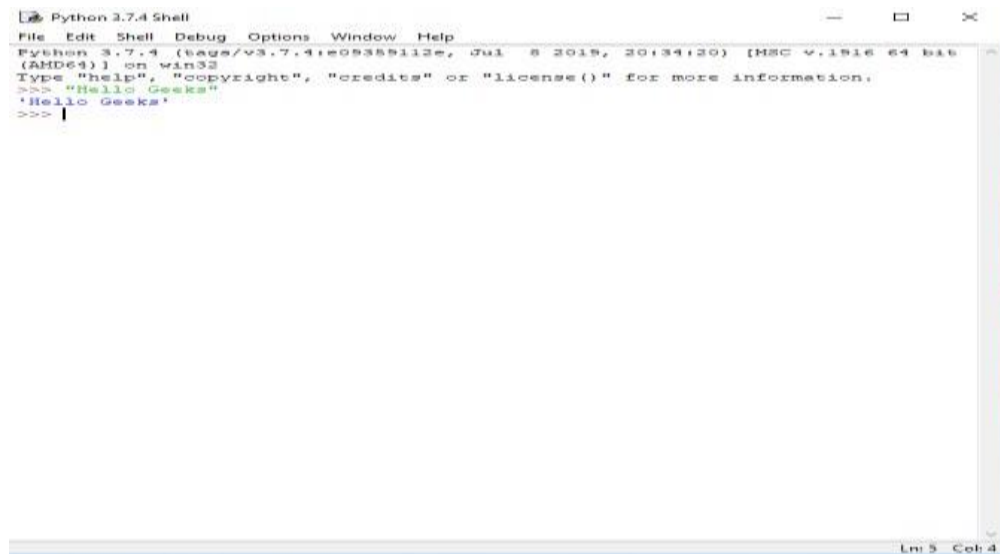
- Run the Python Installer from downloads folder



- Make sure to mark **Add Python 3.7 to PATH** otherwise you will have to do it explicitly. It will start installing python on windows.



- After installation is complete click on **Close**.
Bingo...!! Python is installed. Now go to windows and type IDLE.

A screenshot of a Python 3.7.4 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the Python version and build information: 'Python 3.7.4 (tags/v3.7.4:ec0355b13e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32'. It then prompts the user to type 'help', 'copyright', 'credits', or 'license()' for more information. The user has entered '>>> "Hello Geeks"' and the output is '>>> Hello Geeks'. The status bar at the bottom right shows 'Ln: 5 Col: 4'.

This is Python Interpreter. I printed Hello geeks, python is working smoothly.

How to install Python on Linux?

On every linux system including following OS,

☐ Ubuntu

☐ Linux Mint ☐ Debian ☐

openSUSE ☐ CentOS ☐

Fedora ☐ and my favourite one,

Arch Linux.

You will find Python already installed. You can check it using the following command from the terminal

\$ python --version

To check latest version of python 2.x.x :

\$ python2 --version

To check latest version of python 3.x.x :

\$ python3 --version

```
naveen@naveen-HP-Notebook: ~  
File Edit View Search Terminal Help  
naveen@naveen-HP-Notebook:~$ python --version  
Python 2.7.15+  
naveen@naveen-HP-Notebook:~$ _
```

Clearly it won't be the latest version of python. There can be multiple methods to install python on a linux base system and it all depends on your linux system. For almost every Linux system, the following commands would work definitely.

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
```

```
$ sudo apt-get update
```

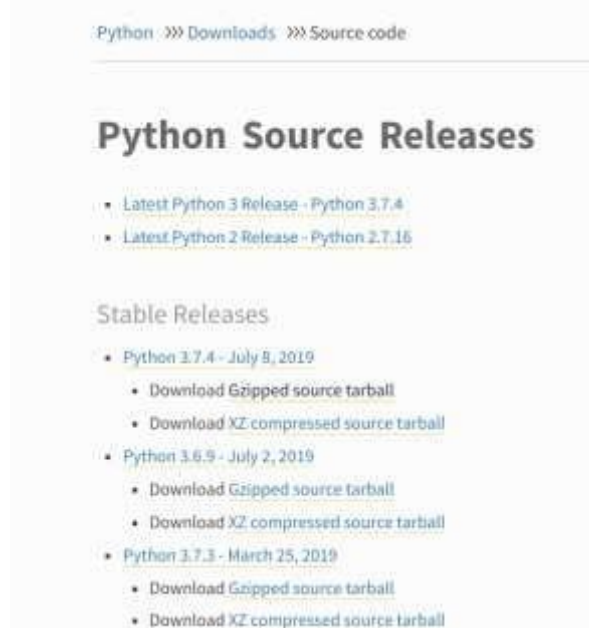
```
$ sudo apt-get install python3.7
```

Download and install Python Latest Version on Linux

To install the latest version from source code of Python follow below steps

Download Python Latest Version from python.org

- First and foremost step is to open a browser and open <https://www.python.org/downloads/source/>



- Underneath the **Stable Releases** find **Download Gzipped source tarball** (latest stable release as of now is Python 3.7.4).
You can do all the above steps in a single command

```
$ wget https://www.python.org/ftp/python/3.7.4/Python-3.7.4.tgz
```

Install Python 3.7.4 Latest Version on Linux

For installing Python successfully on Linux, Enter Following command to get the prerequisites and other source files

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

```
$ sudo apt-get install -y make build-essential libssl-dev zlib1g-dev libbz2-dev  
libreadline-dev libsqlite3-dev wget curl llvm libncurses5-dev libncursesw5-dev  
xzutils tk-dev
```

Now we are all ready to unpack the file downloaded from the python official website'
Move to downloads directory using cd downloads in terminal and
then enter following commands

```
$ tar xvf Python-3.6.5.tgz
```

```
$ cd Python-3.6.5
```

```
$ ./configure --enable-optimizations --with-ensurepip=install
```

```
$ make -j 8
```

```
$ sudo make altinstall
```

Bingo..!! The latest version of Python language is installed on your Linux system. You can confirm it using the below command.

```
python --version
```

How to install Python on macOS / Mac OS X ?

Like Linux, macOS also comes with Python pre-installed on the system. It might be Python version 2 or some similar outdated version. To update to the latest version, we will use the Homebrew Package manager. It is one of the best and convenient methods to install Python on macOS.

To know more about Homebrew Package manager, [visit here](#)

- *Download and install Homebrew Package Manager*

If you don't have homebrew installed on your system, follow the steps below Open the Terminal Application of macOS from Application -> Utilities. Bash terminal will open where you can enter commands Enter following command in macOS terminal

```
/usr/bin/ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

```

Last login: Mon Sep 9 12:28:42 on console
Sierras-MBP: ~ sierra$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
This script will install:
  /usr/local/bin/brew
  /usr/local/etc/bash_completion.d/brew
  /usr/local/etc/man/core.1
  /usr/local/etc/man/site-functions/_brew
  /usr/local/etc/bash_completion.d/brew
  /usr/local/homebrew
  The following existing directories will be made group writable:
  /usr/local/lib
  The following existing directories will have their owner set to g0cccc:
  /usr/local/lib
  The following new directories will be created:
  /usr/local/bin
  /usr/local/etc
  /usr/local/include
  /usr/local/sbin
  /usr/local/share
  /usr/local/var
  /usr/local/opt
  /usr/local/share/man
  /usr/local/share/site-functions
  /usr/local/var/homebrew
  /usr/local/var/homebrew/linked
  /usr/local/Cellar
  /usr/local/Frameworks
  /usr/local/Homebrew
  The Xcode Command Line Tools will be installed.
Press RETURN to continue or any other key to abort

```

Enter the system password if prompted. This will install the Homebrew package Manager on your OS.

After you see a message called “Installation Successful”. You are ready to install python version 3 on your macOS.

```

sierra — bash — 80x24
https://github.com/Homebrew/brew#donations
==> Tapping homebrew/core
Cloning into '/usr/local/Homebrew/Library/Taps/homebrew/homebrew-core'...
remote: Enumerating objects: 5042, done.
remote: Counting objects: 100% (5042/5042), done.
remote: Compressing objects: 100% (4841/4841), done.
remote: Total 5042 (delta 46), reused 565 (delta 10), pack-reused 0
Receiving objects: 100% (5042/5042), 4.03 MiB | 1.59 MiB/s, done.
Resolving deltas: 100% (46/46), done.
Tapped 2 commands and 4830 formulae (5,083 files, 12.5MB).
Already up-to-date.
==> Installation successful!

==> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
  https://docs.brew.sh/Analytics

==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brew#donations
==> Next steps:
- Run 'brew help' to get started
- Further documentation:
  https://docs.brew.sh
Sierras-MBP:~ sierra$

```

- *Install Python Latest Version on macOS / macOS X*

To install python simple open Terminal app from Application -> Utilities and enter following command brew install python3

After command processing is complete, Python’s version 3 would be installed on your mac.

```
sierra — bash — 80x24
echo 'export PATH="/usr/local/opt/sqlite/bin:$PATH"' >> ~/.bash_profile

For compilers to find sqlite you may need to set:
export LDFLAGS="-L/usr/local/opt/sqlite/lib"
export CPPFLAGS="-I/usr/local/opt/sqlite/include"

=> python
Python has been installed as
/usr/local/bin/python3

Unversioned symlinks 'python', 'python-config', 'pip' etc. pointing to
'python3', 'python3-config', 'pip3' etc., respectively, have been installed into
/usr/local/opt/python/libexec/bin

If you need Homebrew's Python 2.7 run
brew install python@2

You can install Python packages with
pip3 install <package>
They will install into the site-package directory
/usr/local/lib/python3.7/site-packages

See: https://docs.brew.sh/Homebrew-and-Python
Sierras-MBP:~ sierra$
```

To verify the installation enter following commands in your Terminal app
python
pip3

```
sierra — bash — 80x24
Last login: Tue Oct 1 12:07:58 on ttys000
Sierras-MBP:~ sierra$ python3 --version
Python 3.7.4
Sierras-MBP:~ sierra$
```

Bingo...!! Python is installed on your computer. You can explore more about python [here](#)

How to install Python on Android ?

Python can run on Android through various apps from play store library. This tutorial will explain how to run python on Android using Pydroid 3 – IDE for Python 3 application.



Features :

- Offline Python 3.7 interpreter: no Internet is required to run Python programs.
- Pip package manager and a custom repository for prebuilt wheel packages for enhanced scientific libraries, such as numpy, scipy, matplotlib, scikit-learn and jupyter.
- Tensorflow is now also available.
- Examples available out-of-the-box for quicker learning.
- Complete Tkinter support for GUI.
- Full-featured Terminal Emulator, with a readline support (available in pip). *Download Pydroid 3 – IDE for Python 3 app from Play store*
- To install Pydroid app go to play store link here – [Pydroid 3 – IDE for Python 3](#)



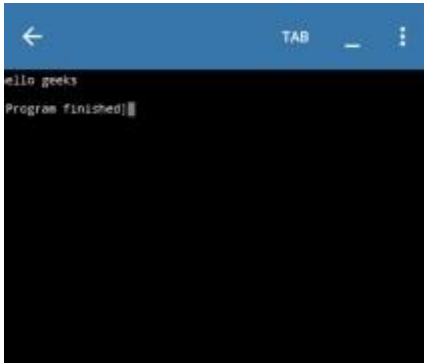
- After installation is complete, run the app and it will show as installing python.



- Wait for a minute and it will show the ide. Here you can enter the Python code.



- Click on the yellow button to run the code.



Python is installed successfully. You can check more features of this app [here](#)

How to install Python on iOS (iPhone / iPad)?

On iOS platform, Python can be installed using various apps from app store. One of the most popular app is [Pythonista](#). Pythonista is a complete development environment for writing Python™ scripts on your iPad or iPhone. Lots of examples are included — from games and animations to plotting, image manipulation, custom user interfaces, and automation scripts.

You can download and buy Pythonista app from [here](#)



Since most of the apps are paid on IOS and it doesn't allow any interpreters officially. One can run Python from online IDEs and ide.geeksforgeeks.org.

Online Interpreters of Python

In this modern era of digital technologies, one can run Python directly from its browser without explicitly installing Python on OS. Here is a list of famous IDEs for python.

- GeeksforGeeks IDE – ide.geeksforgeeks.org
- Python Fiddle: pythonfiddle.com
- Python Anywhere: www.pythonanywhere.com
- Online gdp compiler – onlinegdb.com

For expensive computations for deep learning libraries like TensorFlow, following IDEs can be used

- kaggle – kaggle.com

- Jupyter/IPython Notebook – jupyter.org
- Google Colab – colab.research.google.com

These interpreters can run Python codes easily except for complex Django codes or TensorFlow libraries. To run such advanced applications, you need to install Python explicitly.

Result screen shots:



CHAPTER 6

Conclusion: A fast approach for efficient vehicle counting was proposed. A background model was created for only a small area. When vehicles pass through this narrow area, they are detected as foreground objects. An efficient method was introduced to count vehicles employing the extracted objects. A major difference from other schemes is that no need for a tracking step. Obtained results confirmed the excellent performance of the proposed method. The execution time is very short, which promotes the proposed approach for real-time implementations.

References:

1. K. M. Babu and M. V. Raghunadh. Vehicle number plate detection and recognition using bounding box method. In 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), pages 106–110, May 2016.
2. J. A. Mayan, K. A. Deep, M. Kumar, L. Alvin, and S. P. Reddy. Number plate recognition using template comparison for various fonts in matlab. In 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pages 1– 6, Dec 2016.
3. S. Singh and B. Kaur. Number plate recognition through image using morphological algorithm. In 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pages 3157–3160, March 2016.
4. H. V. Vala and A. Baxti. A review on otsu image segmentation algorithm. International Journal of Advanced Research in Computer Engineering and Technology, page 387, 2013.