

REAL TIME OBJECT DETECTION USING DEEP LEARNING

A PROJECT REPORT

SUBMITTED BY:

PRINCE KUMAR

(730921201044)

SUBMITTED TO:

PEGASUS AEROSPACE SYSTEM

ERODE, TAMIL NADU – 638002

UNDER THE ESTIMATED GUIDANCE

MR.VIKRAM M



PEGASUS AEROSPACE SYSTEM

Erode, Tamil Nadu

EXCEL ENGINEERING COLLEGE

(Approved by A.I.C.T.E , New Delhi & Affiliated To AnnaUniversity, Chennai)

Komarapalyam – 638183, Namakkal (dist), Tamilnadu.



DEPARMENT OF
ARTIFICIAL INTELLIGENCE & DATA SCIENCE

CERTIFICATE

This is to certify that the project report entitled “REAL TIME OBJECT DETECTION USING DEEP LEARNING” submitted by PRINCE KUMAR (730321201044) in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in ARTIFICIAL INTELLIGENCE & DATA SCIENCE of EXCEL ENGINEERING COLLEGE(Autonomous) , NAMAKKAL ,TAMIL NADU is a record of bonafide work carried out under my guidance and supervision.

Project Guide
Mr. PANKAJ PATIL

CEO
ROOPESH RAVICHANDRAN

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr. PANKAJ PATIL**, His guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Mr. VIKRAM M** , for providing us with the required facilities for the completion of the project work. We express our thanks to all teaching faculty of Department of AI&DS , whose suggestions during reviews helped us in accomplishment of our project. We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

ABSTRACT

Real time object detection is a vast, vibrant and complex area of computer vision. If there is a single object to be detected in an image, it is known as Image Localization and if there are multiple objects in an image, then it is Object Detection. This detects the semantic objects of a class in digital images and videos. The applications of real time object detection include tracking objects, video surveillance, pedestrian detection, people counting, self-driving cars, face detection, ball tracking in sports and many more. Convolution Neural Networks is a representative tool of Deep learning to detect objects using OpenCV (Open source Computer Vision), which is a library of programming functions mainly aimed at real time computer vision. Keywords: Computer vision, Deep Learning, Convolution Neural Networks.

INTRODUCTION

Project Objective:

The motive of object detection is to recognize and locate all known objects in a scene. Preferably in 3D space, recovering pose of objects in 3D is very important for robotic control systems. Imparting intelligence to machines and making robots more and more autonomous and independent has been a sustaining technological dream for the mankind. It is our dream to let the robots take on tedious, boring, or dangerous work so that we can commit our time to more creative tasks. Unfortunately, the intelligent part seems to be still lagging behind. In real life, to achieve this goal, besides hardware development, we need the software that can enable robot the intelligence to do the work and act independently. One of the crucial components regarding this is vision, apart from other types of intelligences such as learning and cognitive thinking. A robot cannot be too intelligent if it cannot see and adapt to a dynamic environment. The searching or recognition process in real time scenario is very difficult. So far, no effective solution has been found for this problem. Despite a lot of research in this area, the methods developed so far are not efficient, require long training time, are not suitable for real time application, and are not scalable to large number of classes. Object detection is relatively simpler if the machine is looking for detecting one particular object. However, recognizing all the objects inherently requires the skill to differentiate one object from the other, though they may be of same type. Such problem is very difficult for machines, if they do not know about the various possibilities of objects.

Motivation:

Blind people do lead a normal life with their own style of doing things. But, they definitely face troubles due to inaccessible infrastructure and social challenges. The biggest challenge for a blind person, especially the one with the complete loss of vision, is to navigate around places. Obviously, blind people roam easily around their house without any help because they know the position of everything in the house. Blind people have a tough time finding objects around them. . So we decided to make a REAL TIME OBJECT DETECTION System. We are interested in this project after we went through few papers in this area. As a result we are highly motivated to develop a system that recognizes objects in the real time environment.

OBJECT DETECTION

INTRODUCTION TO OBJECT DETECTION

Object Detection is the process of finding and recognizing real-world object instances such as car, bike, TV, flowers, and humans out of an images or videos. An object detection technique lets you understand the details of an image or a video as it allows for the recognition, localization, and detection of multiple objects within an image.

It is usually utilized in applications like image retrieval, security, surveillance, and advanced driver assistance systems (ADAS).Object Detection is done through many ways:

- Feature Based Object Detection
- Viola Jones Object Detection
- SVM Classifications with HOG Features
- Deep Learning Object Detection

Object detection from a video in video surveillance applications is the major task these days. Object detection technique is used to identify required objects in video sequences and to cluster pixels of these objects.

The detection of an object in video sequence plays a major role in several applications specifically as video surveillance applications.

Object detection in a video stream can be done by processes like pre-processing, segmentation, foreground and background extraction, feature extraction.

Humans can easily detect and identify objects present in an image. The human visual system is fast and accurate and can perform complex tasks like identifying multiple objects with little conscious thought. With the availability of large amounts of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy.

DIGITAL IMAGE PROCESSING

Computerized picture preparing is a range portrayed by the requirement for broad test work to build up the practicality of proposed answers for a given issue. A critical trademark hidden the plan of picture preparing frameworks is the huge level of testing and experimentation that Typically is required before touching base at a satisfactory arrangement. This trademark infors that the capacity to plan approaches and rapidly model hopeful arrangements by and large assumes a noteworthy part in diminishing the cost and time required to land at a suitableframework execution.

WHAT IS DIP?

A picture might be characterized as a two-dimensional capacity $f(x, y)$, where x , y are spatial directions, and the adequacy off at any combine of directions (x, y) is known as the power or dark level of the picture by then. Whenever x , y and the abundance estimation of are all limited discrete amounts, we call the picture a computerized picture. The field of DIP alludes to preparing advanced picture by methods for computerized PC. Advanced picture is made out of a limited number of components, each of which has a specific area and esteem. The components are called pixels. Vision is the most progressive of our sensor, so it is not amazing that picture play the absolute most imperative part in human observation. Be that as it may, dissimilar to people, who are constrained to the visual band of the EM range imaging machines cover practically the whole EM range, going from gamma to radio waves. They can work likewise on pictures produced by sources that people are not acclimated to partner with picture. There is no broad understanding among creators in regards to where picture handling stops and other related territories, for example, picture examination and PC vision begin. Now and then a qualification is made by characterizing picture handling as a teach in which both the info and yield at a procedure are pictures. This is constraining and to some degree manufactured limit. The range of picture investigation is in the middle of picture preparing and PC vision. There are no obvious limits in the continuum from picture handling toward one side to finish vision at the other. In any case, one helpful worldview is to consider three sorts

of mechanized procedures in this continuum: low, mid and abnormal state forms. Low-level process includes primitive operations, for example, picture preparing to decrease commotion differentiate upgrade and picture honing. A lowlevel process is described by the way that both its sources of info and yields are pictures. Mid-level process on pictures includes assignments, for example, division, depiction of that 11 Question diminish them to a frame reasonable for PC handling and characterization of individualarticles

A mid-level process is portrayed by the way that its sources of info by and large are pictures however its yields are properties removed from those pictures. At long last more elevated amount handling includes "Understanding an outlet of perceived items, as in picture examination and at the farthest end of the continuum playing out the intellectual capacities typically connected with human vision. Advanced picture handling, as effectively characterized is utilized effectively in a wide scope of regions of outstanding social and monetary esteem.

WHAT IS AN IMAGE?

A picture is spoken to as a two dimensional capacity $f(x, y)$ where x and y are spatial co-ordinates and the adequacy of "T" at any match of directions (x, y) is known as the power of the picture by then.



Fig. 1.1 digital image

Processing on image:

Processing on image can be of three types They are low-level, mid-level, high level.

Low-level Processing:

- Preprocessing to remove noise.
- Contrast enhancement.
- Image sharpening.

Medium Level Processing:

- Segmentation.
- Edge detection
- Object extraction.

High Level Processing:

- Image analysis
- Scene interpretation

Why Image Processing?

Since the digital image is invisible, it must be prepared for viewing on one or more output device (laser printer, monitor at).The digital image can be optimized for the application by enhancing the appearance of the structures within it.

There are three of image processing used. They are

- Image to Image transformation
- Image to Information transformations
- Information to Image transformations

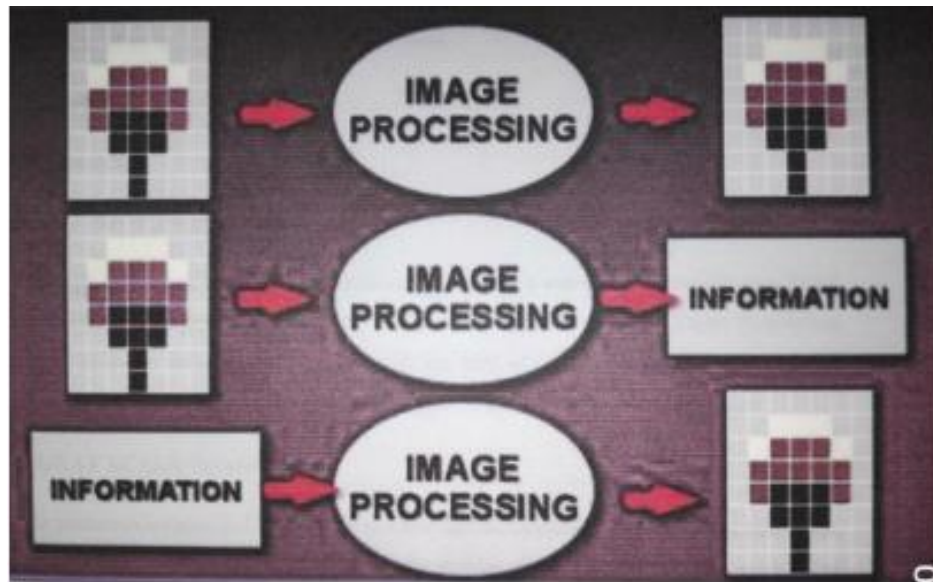


Fig. 1.2 Types of Image Processing

Pixel :

Pixel is the smallest element of an image. Each pixel correspond to any one value. In an 8-bit gray scale image, the value of the pixel between 0 and 255. Each pixel store a value proportional to the light intensity at that particular location. It is indicated in either Pixels per inch or Dots per inch.

Resolution :

The resolution can be defined in many ways. Such as pixel resolution, spatial resolution, temporal resolution, spectral resolution. In pixel resolution, the term resolution refers to the total number of count of pixels in an digital image. For example, If an image has M rows and N columns, then its resolution can be defined as $M \times N$. Higher is the pixel resolution, the higher is the quality of the image

Resolution of an image is of generally two types.

- Low Resolution image
- High Resolution

Since high resolution is not a cost effective process It is not always possible to achieve high resolution images with low cost. Hence it is desirable Imaging. In Super Resolution imaging, with the help of certain methods and algorithms we can be able to produce high resolution images from the low resolution image from the low resolution images.

GRAY SCALE IMAGE

A gray scale picture is a capacity $I(x,y)$ of the two spatial directions of the picture plane. $I(x,y)$ is the force of the picture force of picture at the point (x, y) on the picture plane. $I(x,y)$ take nonnegative expect the picture is limited by a rectangle

COLOR IMAGE

It can be spoken to by three capacities, $R(x,y)$ for red, $G(x,y)$ for green and $B(x,y)$ for blue. A picture might be persistent as for the x and y facilitates and furthermore in adequacy. Changing over 14 such a picture to advanced shape requires that the directions and the adequacy to be digitized. Digitizing the facilitate's esteems is called inspecting. Digitizing the adequacy esteems is called quantization.

RELATED TECHNOLOGY:

R-CNN

R-CNN is a progressive visual object detection system that combines bottom-up region proposals with rich options computed by a convolution neural network. R-CNN uses region proposal ways to initial generate potential bounding boxes in a picture and then run a classifier on these proposed boxes.

SINGLE SIZE MULTI BOX DETECTOR

SSD discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At the time of prediction the network generates scores for the presence of each object category in each default box and generates adjustments to the box to better match the object shape.

Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes.

TENSOR FLOW

Tensor flow is an open source software library for high performance numerical computation. It allows simple deployment of computation across a range of platforms (CPUs, GPUs, TPUs) due to its versatile design also from desktops to clusters of servers to mobile and edge devices. Tensor flow was designed and developed by researchers and engineers from the Google Brain team at intervals Google's AI organization, it comes with robust support for machine learning and deep learning and the versatile numerical computation core is used across several alternative scientific domains.

To construct, train and deploy Object Detection Models TensorFlow is used that makes it easy and also it provides a collection of Detection Models pre-trained on the COCO dataset, the Kitti dataset, and the Open Images dataset. One among the numerous Detection Models is that the combination of Single Shot Detector (SSDs) and Mobile Nets architecture that is quick, efficient and doesn't need huge computational capability to accomplish the object Detection.

APPLICATION OF OBJECT DETECTION

The major applications of Object Detection are:

FACIAL RECOGNITION

“Deep Face” is a deep learning facial recognition system developed to identify human faces in a digital image. Designed and developed by a group of researchers in Facebook. Google also has its own facial recognition system in Google Photos, which automatically separates all the photos according to the person in the image. There are various components involved in Facial Recognition or authors could say it focuses on various aspects like the eyes, nose, mouth and the eyebrows for recognizing a faces.

PEOPLE COUNTING

People counting is also a part of object detection which can be used for various purposes like finding person or a criminal; it is used for analysing store performance or statistics of crowd during festivals. This process is considered a difficult one as people move out of the frame quickly.

INDUSTRIAL QUALITY CHECK

Object detection also plays an important role in industrial processes to identify or recognize products. Finding a particular object through visual examination could be a basic task that's involved in multiple industrial processes like sorting, inventory management, machining, quality management, packaging and so on. Inventory management can be terribly tough as things are hard to trace in real time. Automatic object counting and localization permits improving inventory accuracy.

SELF DRIVING CARS

Self-driving is the future most promising technology to be used, but the working behind can be very complex as it combines a variety of techniques to perceive their surroundings, including radar, laser light, GPS, odometer, and computer vision. Advanced control systems interpret sensory info to allow navigation

methods to work, as well as obstacles and it. This is a big step towards Driverless cars as it happens at very fast speed.

SECURITY

Object Detection plays a vital role in the field of Security; it takes part in major fields such as face ID of Apple or the retina scan used in all the sci-fi movies. Government also widely use this application to access the security feed and match it with their existing database to find any criminals or to detecting objects like car number involved in criminal activities. The applications are limitless.

OBJECT DETECTION WORKFLOW AND FEATURE EXTRACTION

Every Object Detection Algorithm works on the same principle and it's just the working that differs from others. They focus on extracting features from the images that are given as the input at hands and then it uses these features to determine the class of the image.

DEEP LEARNING

INTRODUCTION

Deep learning is a machine learning technique. It teaches a computer to filter inputs through layers to learn how to predict and classify information.

Observations can be in the form of images, text, or sound. The inspiration for deep learning is the way that the human brain filters information. Its purpose is to mimic how the human brain works to create some real magic. In the human brain, there are about 100 billion neurons. Each neuron connects to about 100,000 of its neighbors. We're kind of recreating that, but in a way and at a level that works for machines. In our brains, a neuron has a body, dendrites, and an axon. The signal from one neuron travels down the axon and transfers to the dendrites of the next neuron. That connection where the signal passes is called a synapse. Neurons by themselves are kind of useless. But when you have lots of them, they work together to create some serious magic. That's the idea behind a deep learning algorithm! You get input from observation and you put your input into one layer. That layer creates an output which in turn becomes the input for the next layer, and so on. This happens over and over until your final output signal! The neuron (node) gets a signal or signals (input values), which pass through the neuron. That neuron delivers the output signal.

Think of the input layer as your senses: the things you see, smell, and feel, for example. These are independent variables for one single observation. This information is broken down into numbers and the bits of binary data that a computer can use. You'll need to either standardize or normalize these variables so that they're within the same range. They use many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output of the previous layer for its input. What they learn forms a hierarchy of concepts. In this hierarchy, each level learns to transform its input data into a more and more abstract and composite representation. That means that for an image, for example, the input might be a matrix of pixels. The first layer might encode the edges and compose the pixels. The next layer might compose an arrangement of edges. The next layer might encode a nose and eyes. The next layer might recognize that the image contains a face, and so on.

What happens inside the neuron?

The input node takes in information in a numerical form. The information is presented as an activation value where each node is given a number. The higher the number, the greater the activation. Based on the connection strength (weights) and transfer function, the activation value passes to the next node. Each of the nodes sums the activation values that it receives (it calculates the weighted sum) and modifies that sum based on its transfer function. Next, it applies an activation function. An activation function is a function that's applied to this particular neuron. From that, the neuron understands if it needs to pass along a signal or not.

Each of the synapses gets assigned weights, which are crucial to Artificial Neural Networks (ANNs). Weights are how ANNs learn. By adjusting the weights, the ANN decides to what extent signals get passed along. When you're training your network, you're deciding how the weights are adjusted. The activation runs through the network until it reaches the output nodes. The output nodes then give us the information in a way that we can understand. Your network will use a cost function to compare the output and the actual expected output. The model performance is evaluated by the cost function. It's expressed as the difference between the actual value and the predicted value.

There are many different cost functions you can use, you're looking at what the error you have in your network is. You're working to minimize loss function. (In essence, the lower the loss function, the closer it is to your desired output). The information goes back, and the neural network begins to learn with the goal of minimizing the cost function by tweaking the weights. This process is called backpropagation.

In forward propagation, information is entered into the input layer and propagates forward through the network to get our output values. We compare the values to our expected results. Next, we calculate the errors and propagate the info backward. This allows us to train the network and update the weights. (Backpropagation allows us to adjust all the weights simultaneously.) During this process, because of the way the algorithm is structured, you're able to adjust all of the weights simultaneously. This allows you to see which part of the error each of your weights in the neural network is responsible for.

When you've adjusted the weights to the optimal level, you're ready to proceed to the testing phase!

How does an artificial neural network learn?

There are two different approaches to get a program to do what you want. First, there's the specifically guided and hard-programmed approach. You tell the program exactly what you want it to do. Then there are neural networks. In neural networks, you tell your network the inputs and what you want for the outputs, and then you let it learn on its own.

By allowing the network to learn on its own, you can avoid the necessity of entering in all of the rules. You can create the architecture and then let it go and learn. Once it's trained up, you can give it a new image and it will be able to distinguish output.

Feedforward and feedback networks

A feedforward network is a network that contains inputs, outputs, and hidden layers. The signals can only travel in one direction (forward). Input data passes into a layer where calculations are performed. Each processing element computes based upon the weighted sum of its inputs. The new values become the new input values that feed the next layer (feed-forward). This continues through all the layers and determines the output. Feedforward networks are often used in, for example, data mining.

A feedback network (for example, a recurrent neural network) has feedback paths. This means that they can have signals traveling in both directions using loops. All possible connections between neurons are allowed. Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium. Feedback networks are often used in optimization problems where the network looks for the best arrangement of interconnected factors.

Weighted Sum

Inputs to a neuron can either be features from a training set or outputs from the neurons of a previous layer. Each connection between two neurons has a unique synapse with a unique weight attached. If you want to get from one neuron to the next, you have to travel along the synapse and pay the "toll" (weight). The

neuron then applies an activation function to the sum of the weighted inputs from each incoming synapse. It passes the result on to all the neurons in the next layer. When we talk about updating weights in a network, we're talking about adjusting the weights on these synapses.

A neuron's input is the sum of weighted outputs from all the neurons in the previous layer. Each input is multiplied by the weight associated with the synapse connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weights: one for each synapse.

In a nutshell, the activation function of a node defines the output of that node.

The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It's an abstraction that represents the rate of action potential firing in the cell. It's a number that represents the likelihood that the cell will fire. At it's simplest, the function is binary: yes (the neuron fires) or no (the neuron doesn't fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range. If you were using a function that maps a range between 0 and 1 to determine the likelihood that an image is a cat, for example, an output of 0.9 would show a 90% probability that your image is, in fact, a cat.

Activation function

In a nutshell, the activation function of a node defines the output of that node.

The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It's an abstraction that represents the rate of action potential firing in the cell. It's a number that represents the likelihood that the cell will fire. At it's simplest, the function is binary: yes (the neuron fires) or no (the neuron doesn't fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range.

What options do we have? There are many activation functions, but these are the four very common ones:

Thresholdfunction

This is a step function. If the summed value of the input reaches a certain threshold the function passes on 0. If it's equal to or more than zero, then it would pass on 1. It's a very rigid, straightforward, yes or no function.

Sigmoid function

This function is used in logistic regression. Unlike the threshold function, it's a smooth, gradual progression from 0 to 1. It's useful in the output layer and is used heavily for linear regression.

Hyperbolic Tangent Function

This function is very similar to the sigmoid function. But unlike the sigmoid function which goes from 0 to 1, the value goes below zero, from -1 to 1. Even though this isn't a lot like what happens in a brain, this function gives better results when it comes to training neural networks. Neural networks sometimes get "stuck" during training with the sigmoid function. This happens when there's a lot of strongly negative input that keeps the output near zero, which messes with the learning process.

Rectifier function

This might be the most popular activation function in the universe of neural networks. It's the most efficient and biologically plausible. Even though it has a kink, it's smooth and gradual after the kink at 0. This means, for example, that your output would be either "no" or a percentage of "yes." This function doesn't require normalization or other complicated calculations.

The field of artificial intelligence is essential when machines can do tasks that typically require human intelligence. It comes under the layer of machine learning, where machines can acquire skills and learn from past experience without any involvement of human. Deep learning comes under machine learning where artificial neural networks, algorithms inspired by the human brain, learn

from large amounts of data. The concept of deep learning is based on humans' experiences; the deep learning algorithm would perform a task continuously so that it can improve the outcome. Neural networks have various (deep) layers that enable learning. Any drawback that needs "thought" to work out could be a drawback deep learning can learn to unravel.

PROGRAM

Model.py

```
from sklearn.svm import LinearSVC

import numpy as np
import cv2 as cv
import PIL

class Model:

    def __init__(self):
        self.model = LinearSVC()

    def train_model(self, counters):
        img_list = np.array([])
        class_list = np.array([])

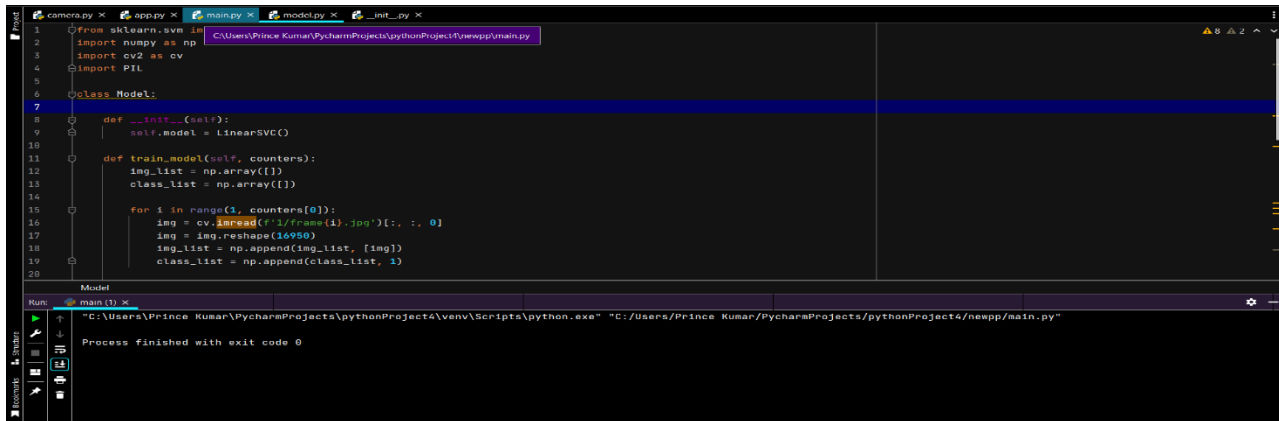
        for i in range(1, counters[0]):
            img = cv.imread(f'1/frame{i}.jpg')[:, :, 0]
            img = img.reshape(16950)
            img_list = np.append(img_list, [img])
            class_list = np.append(class_list, 1)

        for i in range(1, counters[1]):
            img = cv.imread(f'2/frame{i}.jpg')[:, :, 0]
            img = img.reshape(16950)
            img_list = np.append(img_list, [img])
            class_list = np.append(class_list, 2)

        img_list = img_list.reshape(counters[0] - 1 + counters[1] - 1, 16950)
```

```
self.model.fit(img_list, class_list)

print("Model successfully trained!")
```



```
1 from sklearn.svm import LinearSVC
2 import numpy as np
3 import cv2 as cv
4 import PIL
5
6 class Model:
7
8     def __init__(self):
9         self.model = LinearSVC()
10
11     def train_model(self, counters):
12         img_list = np.array([])
13         class_list = np.array([])
14
15         for i in range(1, counters[0]):
16             img = cv.imread(f'1/frame{i}.jpg')[ :, :, 0]
17             img = img.reshape(16950)
18             img_list = np.append(img_list, [img])
19             class_list = np.append(class_list, 1)
20
21
22 if __name__ == '__main__':
23     model = Model()
24     model.train_model(counters)
```

The code above creates a machine learning model using the Linear Support Vector Classifier (LinearSVC) algorithm from the Scikit-learn library. The model is trained on a set of image data, where the images are divided into two classes - represented by the numbers 1 and 2.

The computer vision Library OpenCV is used to read in the image data, and the images are converted to grayscale by indexing only the first channel (i.e., `cv.imread(f'1/frame{i}.jpg')[:, :, 0]`). These grayscale images are then flattened to a one-dimensional array, with a length of 16950.

The flattened image data is stored in an array 'img_list', and the corresponding class (i.e., the number 1 or 2) is stored in another array, 'class_list'. The img_list contains both classes' image data, and the class_list represents which class each image belongs to.

The model is fit using the img_list and class_list data via the fit function of the LinearSVC class. The model is trained to predict the class of a new image data by using the previously trained data to recognize patterns and create a decision boundary between the two classes.

Finally, once the model is trained, a success message is printed on the screen.

App.py

```
import tkinter as tk
from tkinter import simpledialog
import cv2 as cv
import os
import PIL.Image, PIL.ImageTk
import model
import camera

class App:

    def __init__(self, window=tk.Tk(), window_title="Camera Classifier"):

        self.window = window
        self.window_title = window_title

        self.counters = [1, 1]

        self.model = model.Model()

        self.auto_predict = False

        self.camera = camera.Camera()

        self.init_gui()
```

```
self.delay = 15
```

```
self.update()
```

```
self.window.attributes("-topmost", True)
```

```
self.window.mainloop()
```

```
def init_gui(self):
```

```
    self.canvas = tk.Canvas(self.window, width=self.camera.width,  
height=self.camera.height)
```

```
    self.canvas.pack()
```

```
    self.btn_toggleauto = tk.Button(self.window, text="Auto Prediction",  
width=50, command=self.auto_predict_toggle)
```

```
    self.btn_toggleauto.pack(anchor=tk.CENTER, expand=True)
```

```
    self.classname_one = simplifiedialog.askstring("Classname 1st", "Enter the  
name of the first class:", parent=self.window)
```

```
    self.classname_two = simplifiedialog.askstring("Classname 2nd", "Enter the  
name of the second class:", parent=self.window)
```

```
    self.btn_class_one = tk.Button(self.window, text=self.classname_one,  
width=50, command=lambda: self.save_for_class(1))
```

```
    self.btn_class_one.pack(anchor=tk.CENTER, expand=True)
```

```
    self.btn_class_two = tk.Button(self.window, text=self.classname_two,  
width=50, command=lambda: self.save_for_class(2))
```

```
    self.btn_class_two.pack(anchor=tk.CENTER, expand=True)
```

```
self.btn_train = tk.Button(self.window, text="Train Model", width=50,
command=lambda: self.model.train_model(self.counters))

self.btn_train.pack(anchor=tk.CENTER, expand=True)


self.btn_predict = tk.Button(self.window, text="Predict", width=50,
command=self.predict)

self.btn_predict.pack(anchor=tk.CENTER, expand=True)


self.btn_reset = tk.Button(self.window, text="Reset", width=50,
command=self.reset)

self.btn_reset.pack(anchor=tk.CENTER, expand=True)


# creator Prince PRK
self.class_label = tk.Label(self.window, text="Created by Prince PRK")
self.class_label.config(font=("Arial", 20))
self.class_label.pack(anchor=tk.CENTER, expand=True)


self.class_label = tk.Label(self.window, text="CLASS")
self.class_label.config(font=("Arial", 20))
self.class_label.pack(anchor=tk.CENTER, expand=True)


def auto_predict_toggle(self):
    self.auto_predict = not self.auto_predict


def save_for_class(self, class_num):
    ret, frame = self.camera.get_frame()
    if not os.path.exists("1"):
```

```
    os.mkdir("1")
if not os.path.exists("2"):
    os.mkdir("2")

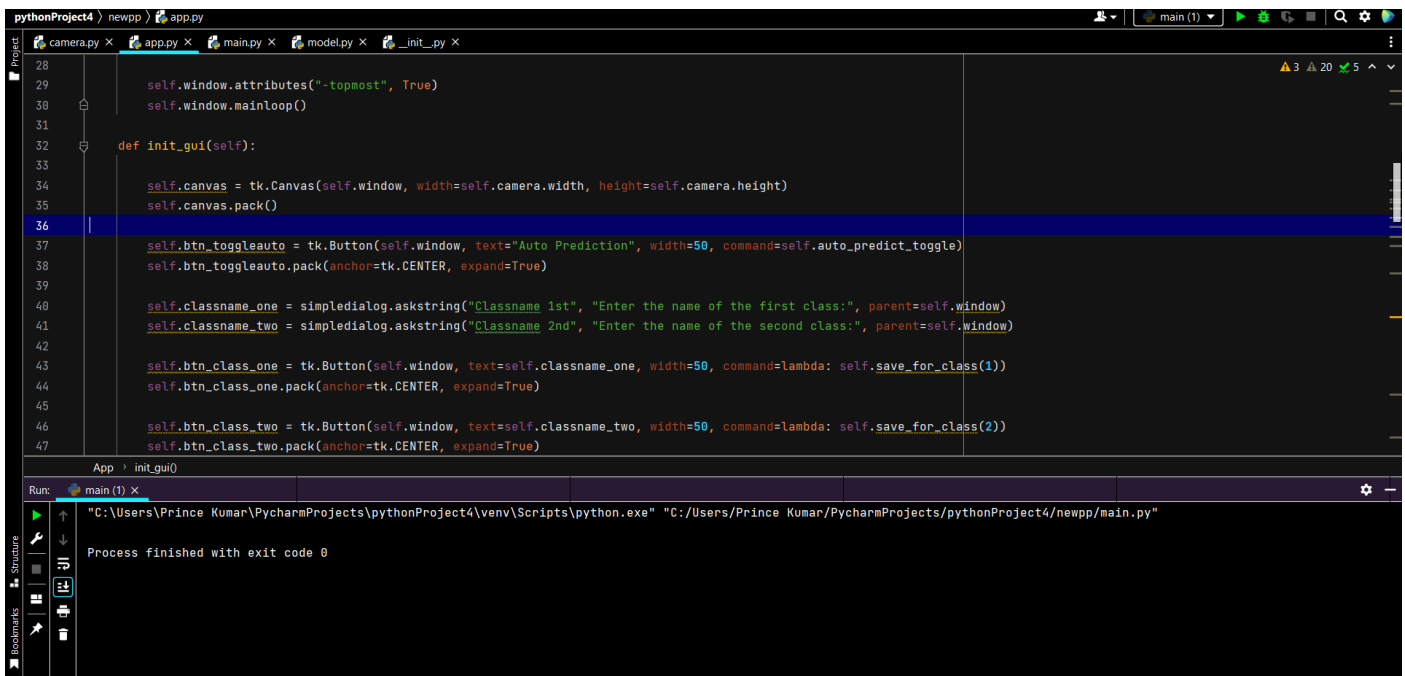
    cv.imwrite(f'{class_num}/frame{self.counters[class_num-1]}.jpg',
cv.cvtColor(frame, cv.COLOR_RGB2GRAY))

    img = PIL.Image.open(f'{class_num}/frame{self.counters[class_num -
1]}.jpg')
    img.thumbnail((150, 150), PIL.Image.ANTIALIAS)
    img.save(f'{class_num}/frame{self.counters[class_num - 1]}.jpg')

self.counters[class_num - 1] += 1

def reset(self):
    for folder in ['1', '2']:
        for file in os.listdir(folder):
            file_path = os.path.join(folder, file)
            if os.path.isfile(file_path):
                os.unlink(file_path)

self.counters = [1, 1]
self.model = model.Model()
self.class_label.config(text="CLASS")
```



```
pythonProject4 newpp app.py
28
29 self.window.attributes("-topmost", True)
30 self.window.mainloop()
31
32 def init_gui(self):
33
34     self.canvas = tk.Canvas(self.window, width=self.camera.width, height=self.camera.height)
35     self.canvas.pack()
36
37     self.btn_toggleauto = tk.Button(self.window, text="Auto Prediction", width=50, command=self.auto_predict_toggle)
38     self.btn_toggleauto.pack(anchor=tk.CENTER, expand=True)
39
40     self.classname_one = simpledialog.askstring("Classname 1st", "Enter the name of the first class:", parent=self.window)
41     self.classname_two = simpledialog.askstring("Classname 2nd", "Enter the name of the second class:", parent=self.window)
42
43     self.btn_class_one = tk.Button(self.window, text=self.classname_one, width=50, command=lambda: self.save_for_class(1))
44     self.btn_class_one.pack(anchor=tk.CENTER, expand=True)
45
46     self.btn_class_two = tk.Button(self.window, text=self.classname_two, width=50, command=lambda: self.save_for_class(2))
47     self.btn_class_two.pack(anchor=tk.CENTER, expand=True)
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

App › init_gui()

Run: main (1) ×

"C:\Users\Prince Kumar\PycharmProjects\pythonProject4\venv\Scripts\python.exe" "C:\Users\Prince Kumar\PycharmProjects\pythonProject4\newpp\main.py"

Process finished with exit code 0

The above code defines a simple GUI application for capturing images from a camera and classifying them into two classes using a pre-trained model. The GUI is implemented using the tkinter library, and the image capture is performed using the OpenCV library. The application allows the user to manually capture and save images for training the model, train the model, and perform real-time classification of captured images.

The main class is the `App` class, which initializes the GUI, sets up the camera and the model, and defines the various GUI elements such as buttons, labels, and canvas for displaying the captured image. The `init_gui` method sets up the various GUI elements, including buttons for capturing and saving images for training, buttons for training the model and performing real-time classification, and labels for displaying the class name and the creator's name.

The `auto_predict_toggle` method is used to toggle automatic prediction of captured images, which is triggered by a button click event. When this option is enabled, the `predict` method is called automatically for each captured frame.

The `save_for_class` method saves a captured frame to a folder corresponding to its class. The class number is passed as an argument, and the `counters` list is used to keep track of the number of images saved for each class.

The ``reset`` method deletes all the previously saved images, resets the ``counters`` list, and creates a new instance of the model.

The ``update`` method is called repeatedly with a small delay to update the GUI and capture new frames from the camera. The captured frame is displayed on the canvas, and the ``predict`` method is called if the auto-prediction option is enabled.

The ``predict`` method performs classification of the captured frame using the pre-trained model and updates the class label accordingly. The method returns the predicted class name for further processing, such as logging or triggering other actions.

Overall, the code provides a simple and user-friendly interface for capturing and classifying images using a pre-trained model. However, it can be improved by adding error handling and better GUI design.

Camera.py

```
import cv2 as cv
```

```
class Camera:
```

```
    def __init__(self):
```

```
        self.camera = cv.VideoCapture(0)
```

```
        if not self.camera.isOpened():
```

```
            raise ValueError("Unable to open camera!")
```

```
        self.width = self.camera.get(cv.CAP_PROP_FRAME_WIDTH)
```

```
        self.height = self.camera.get(cv.CAP_PROP_FRAME_HEIGHT)
```

```
    def __del__(self):
```

```
        if self.camera.isOpened():
```

```
            self.camera.release()
```

```
    def get_frame(self):
```

```
        if self.camera.isOpened():
```

```
            ret, frame = self.camera.read()
```

```
            if ret:
```

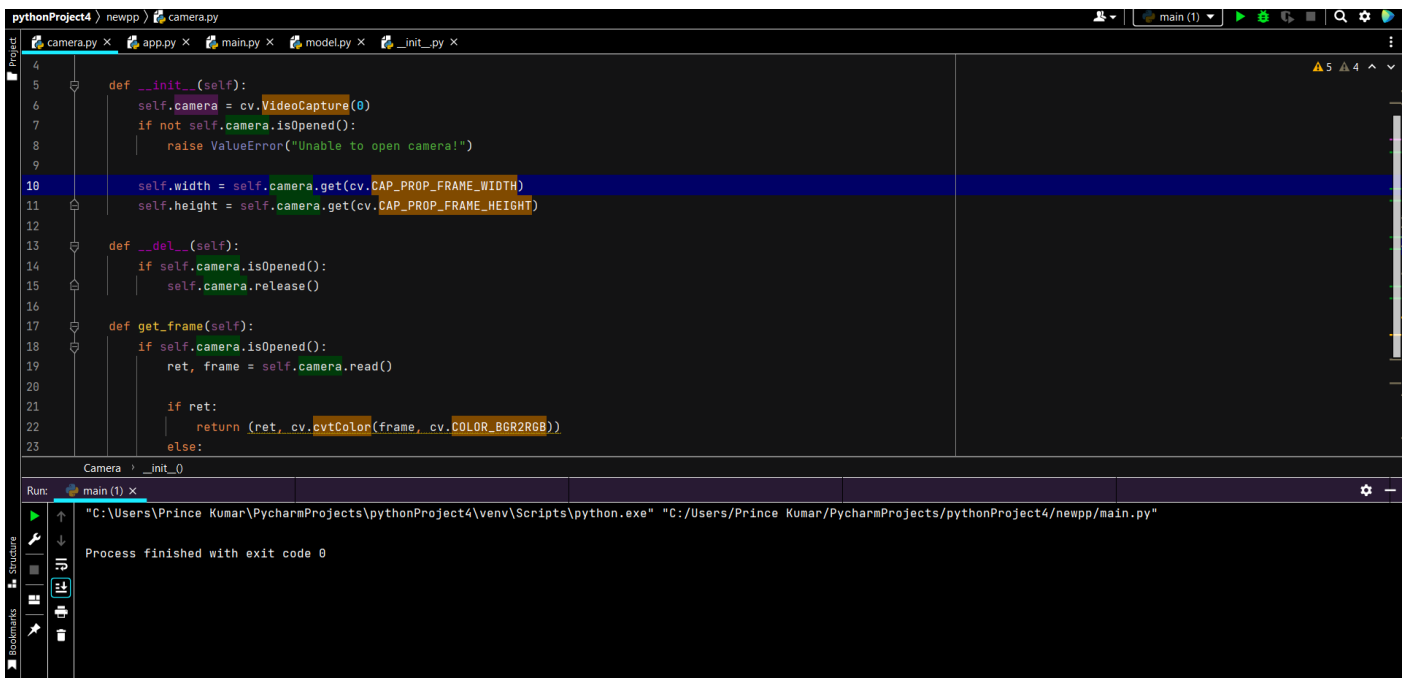
```
                return (ret, cv.cvtColor(frame, cv.COLOR_BGR2RGB))
```

```
            else:
```

```
                return (ret, None)
```

```
        else:
```

return None

The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for running, debugging, and other IDE functions. The main editor window displays a Python file named 'camera.py' with the following code:

```
4
5 def __init__(self):
6     self.camera = cv.VideoCapture(0)
7     if not self.camera.isOpened():
8         raise ValueError("Unable to open camera!")
9
10    self.width = self.camera.get(cv.CAP_PROP_FRAME_WIDTH)
11    self.height = self.camera.get(cv.CAP_PROP_FRAME_HEIGHT)
12
13 def __del__(self):
14     if self.camera.isOpened():
15         self.camera.release()
16
17 def get_frame(self):
18     if self.camera.isOpened():
19         ret, frame = self.camera.read()
20
21         if ret:
22             return (ret, cv.cvtColor(frame, cv.COLOR_BGR2RGB))
23         else:
24             return None
```

The bottom panel shows the 'Run' tab with the command: `"C:\Users\Prince Kumar\PycharmProjects\pythonProject4\venv\Scripts\python.exe" "C:\Users\Prince Kumar\PycharmProjects\pythonProject4\newpp/main.py"`. Below the command, it states: `Process finished with exit code 0`.

The above code defines a 'Camera' class that is used to capture frames from the default camera attached to the computer.

The `'__init__'` method initializes the camera by creating a `'VideoCapture'` object with the index `'0'`, which is the index of the default camera on most systems. If the camera cannot be opened, a `'ValueError'` is raised. The width and height of the frames captured by the camera are also initialized and stored as instance variables.

The `'__del__'` method releases the camera when the object is destroyed.

The `'get_frame'` method captures a frame from the camera and returns a tuple containing a Boolean value indicating whether the capture was successful and the captured frame as a NumPy array in RGB format. If the capture was not successful, `'None'` is returned.

Overall, this code provides a simple interface for capturing frames from the default camera, which can be useful for a variety of computer vision tasks.

Main.py

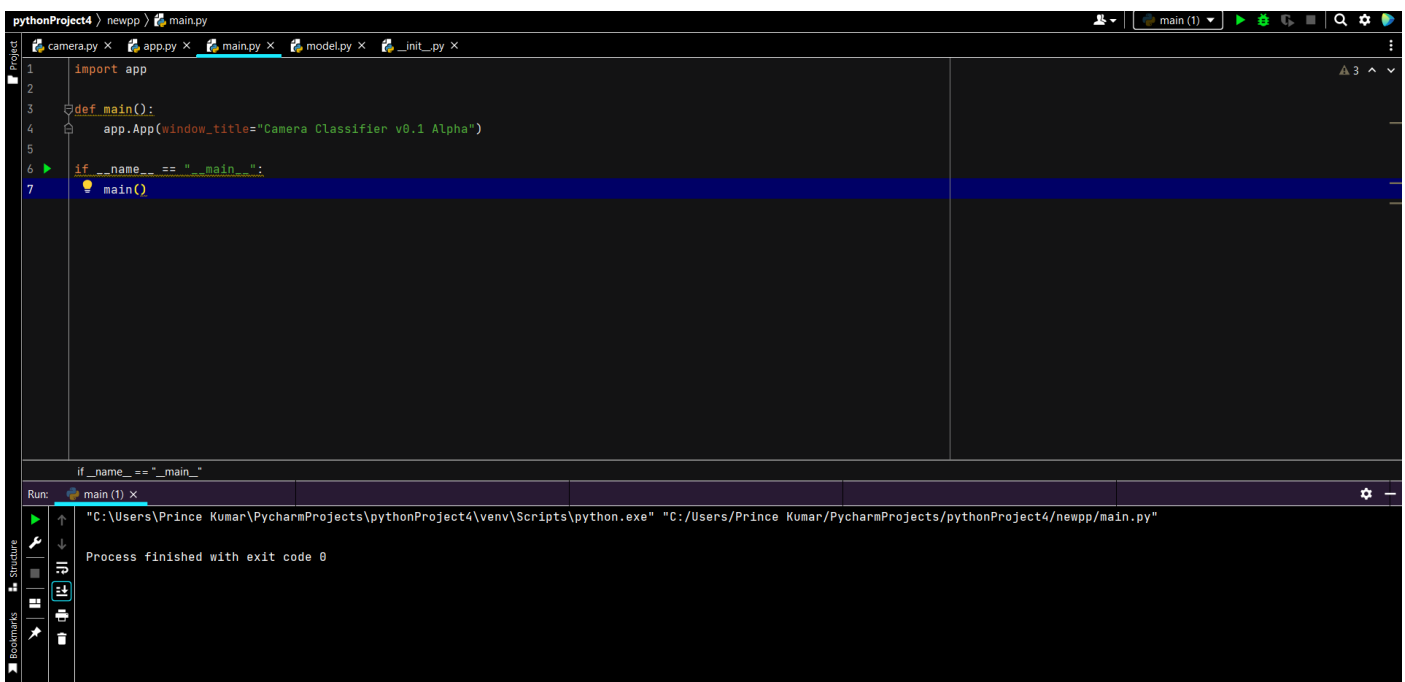
```
import app
```

```
def main():
```

```
    app.App(window_title="Camera Classifier v0.1 Alpha")
```

```
if __name__ == "__main__":
```

```
    main()
```



The above code is a simple Python script that creates an instance of the `App` class defined in the `app` module and starts the GUI application.

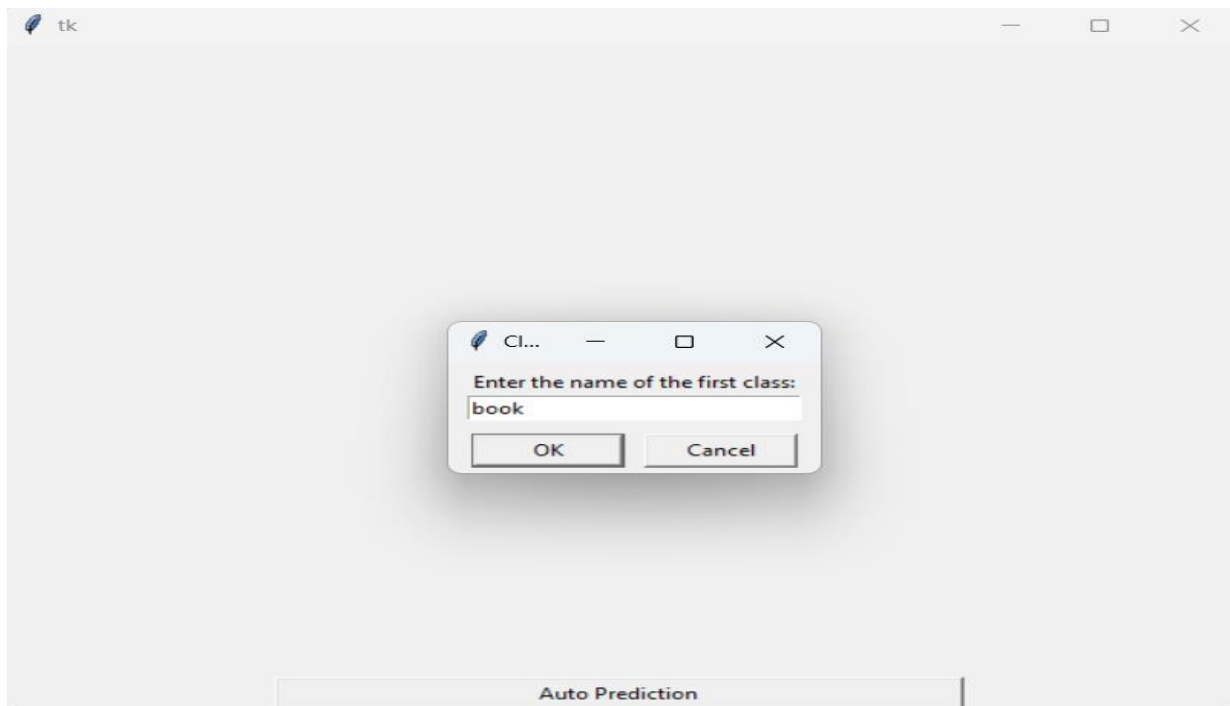
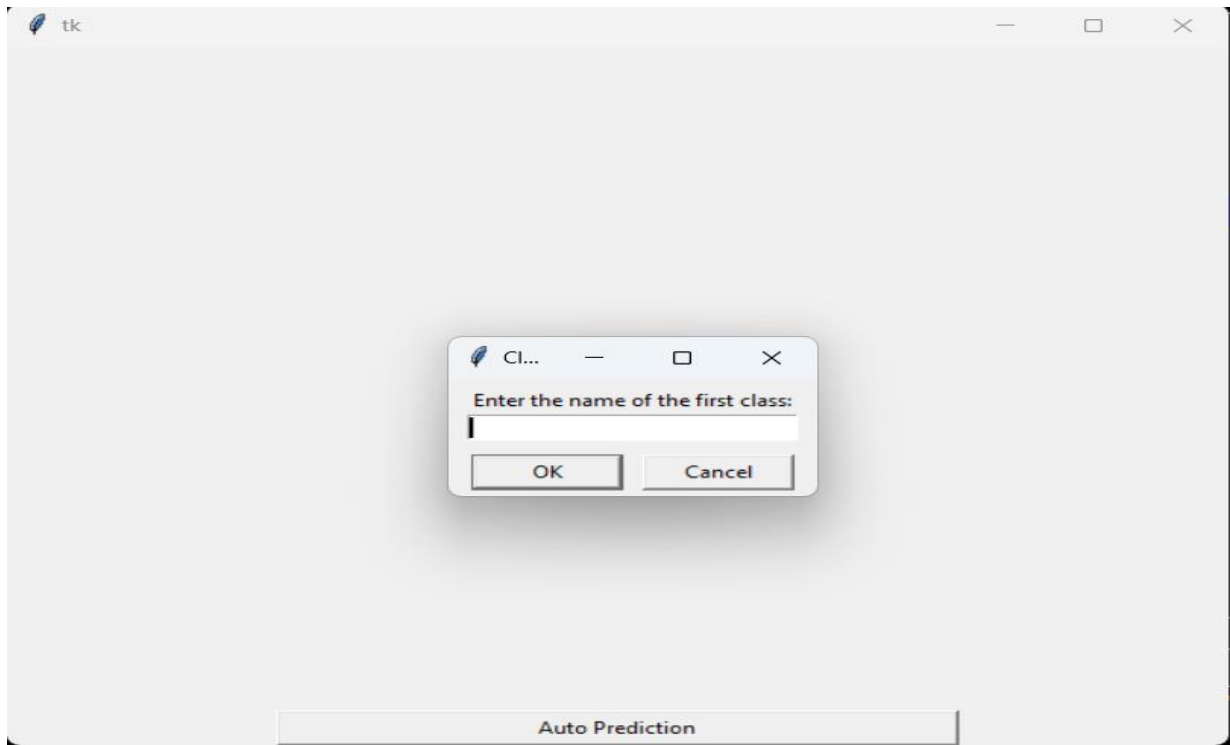
The `main()` function is defined to create the instance of the `App` class with the `window_title` argument set to "Camera Classifier v0.1 Alpha". Then, it calls the `main()` function when the script is executed.

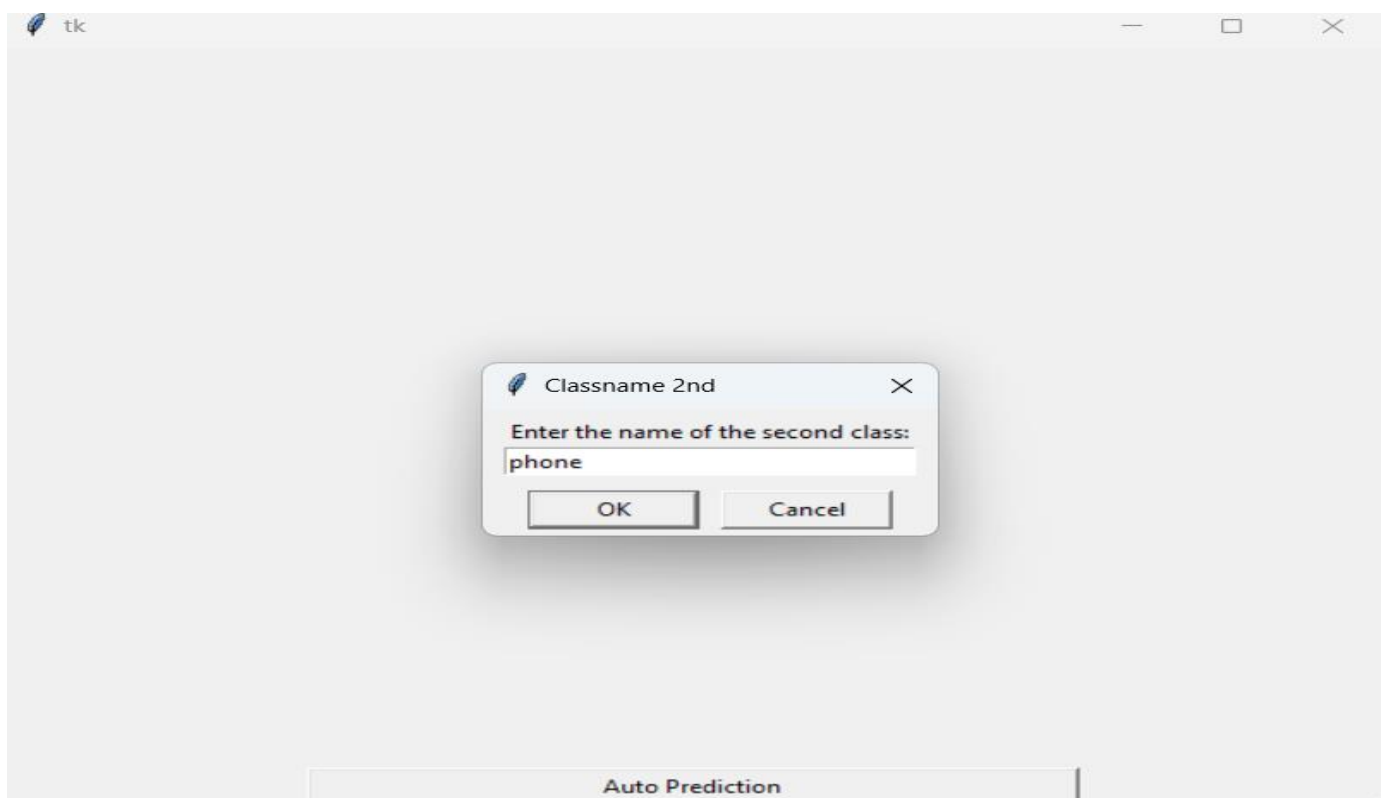
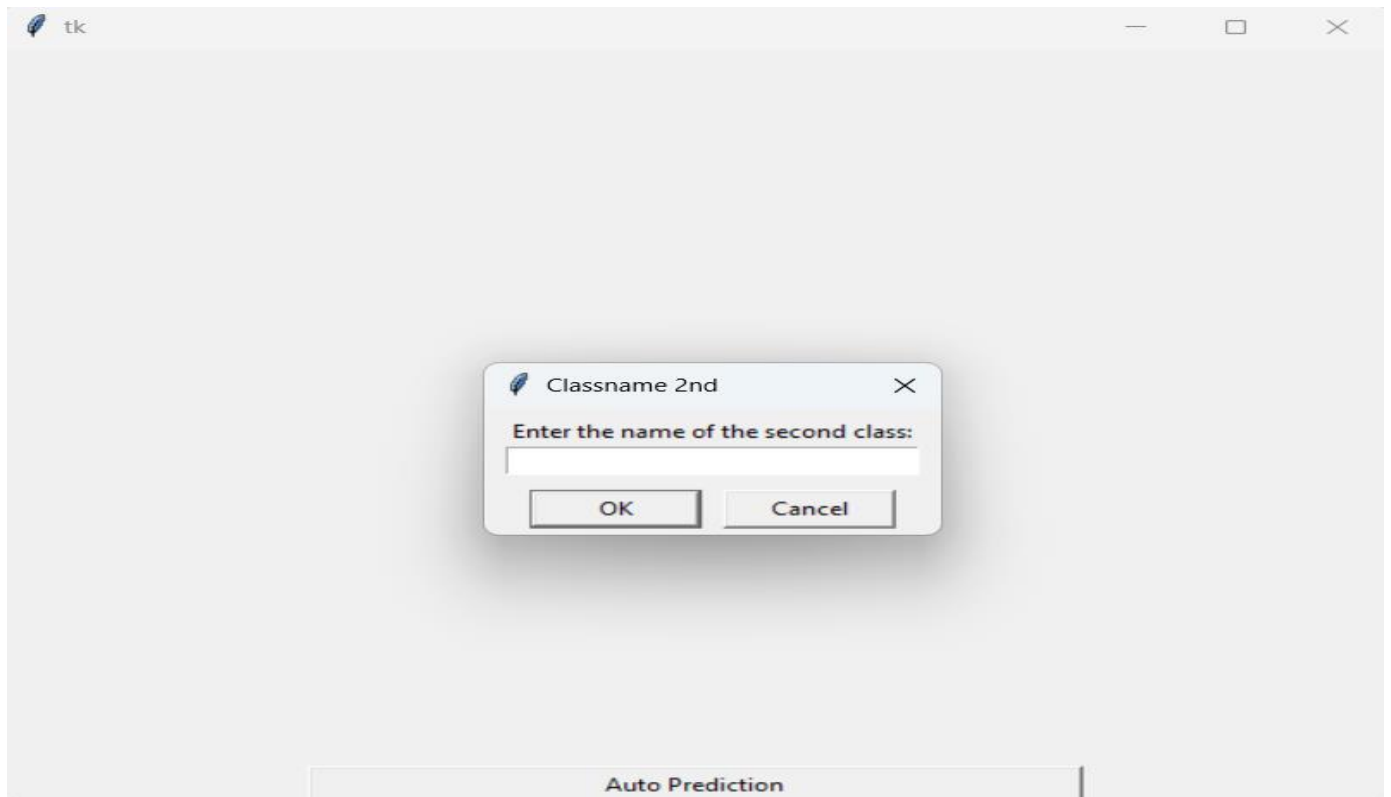
When the `App` instance is created, the `_init_()` method is called. This method initializes the `App` class and its attributes. It also creates an instance of the

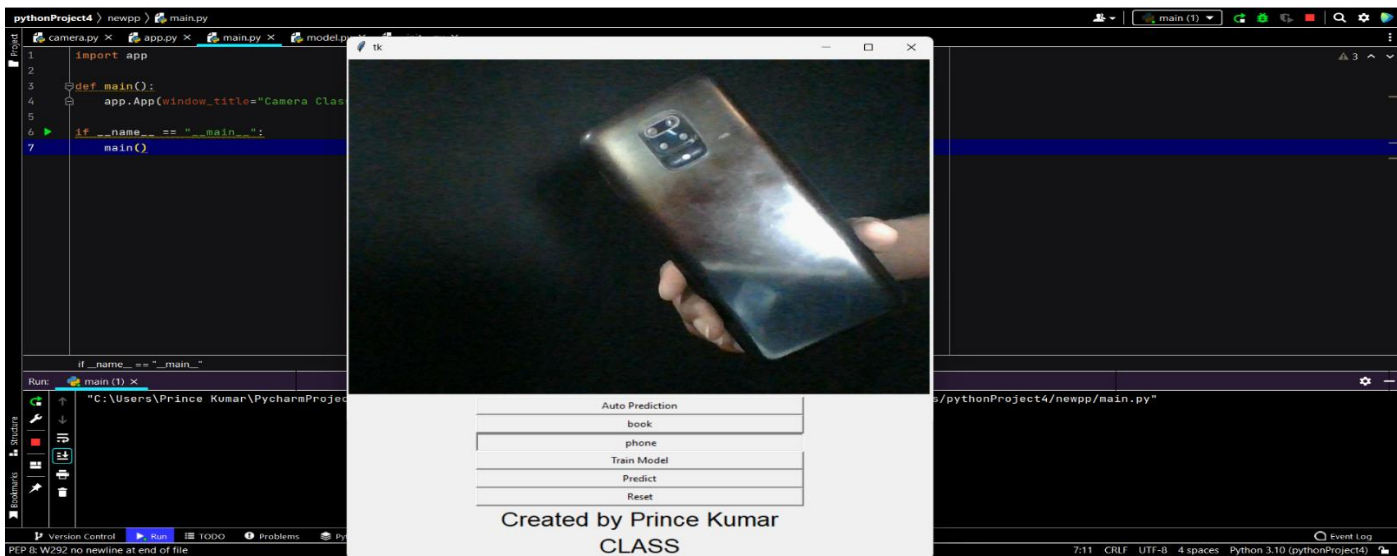
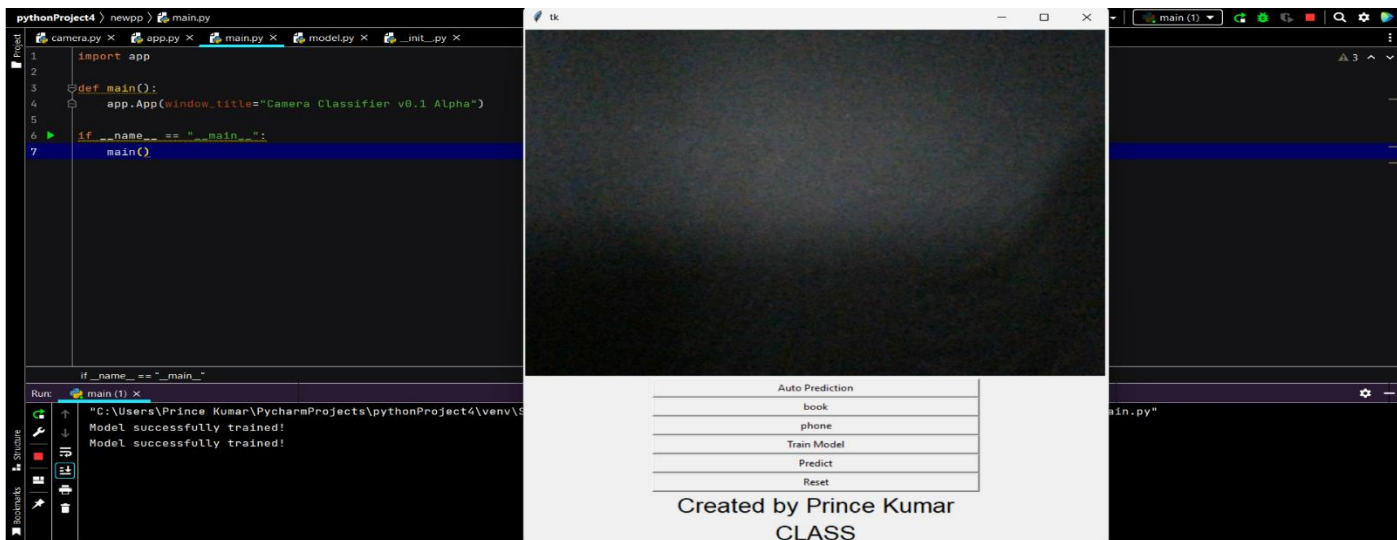
`Model` and `Camera` classes, sets up the GUI, and starts the main loop for the application.

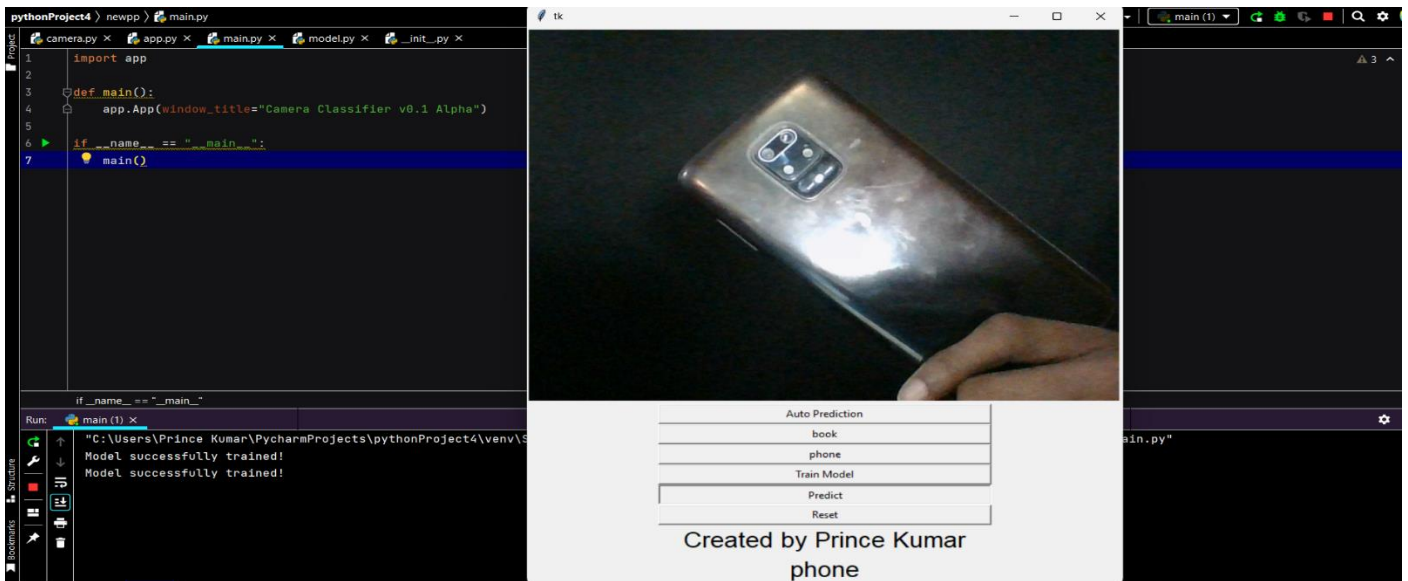
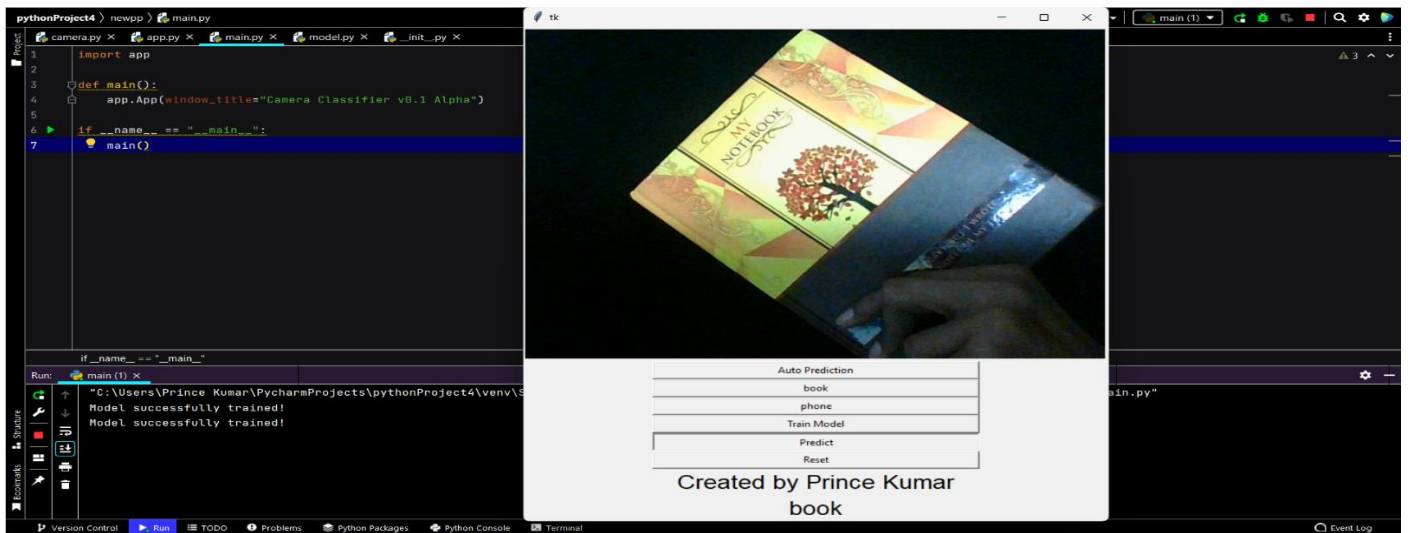
In summary, this script is a simple driver program that creates and starts a GUI application that uses the computer's camera to classify images.

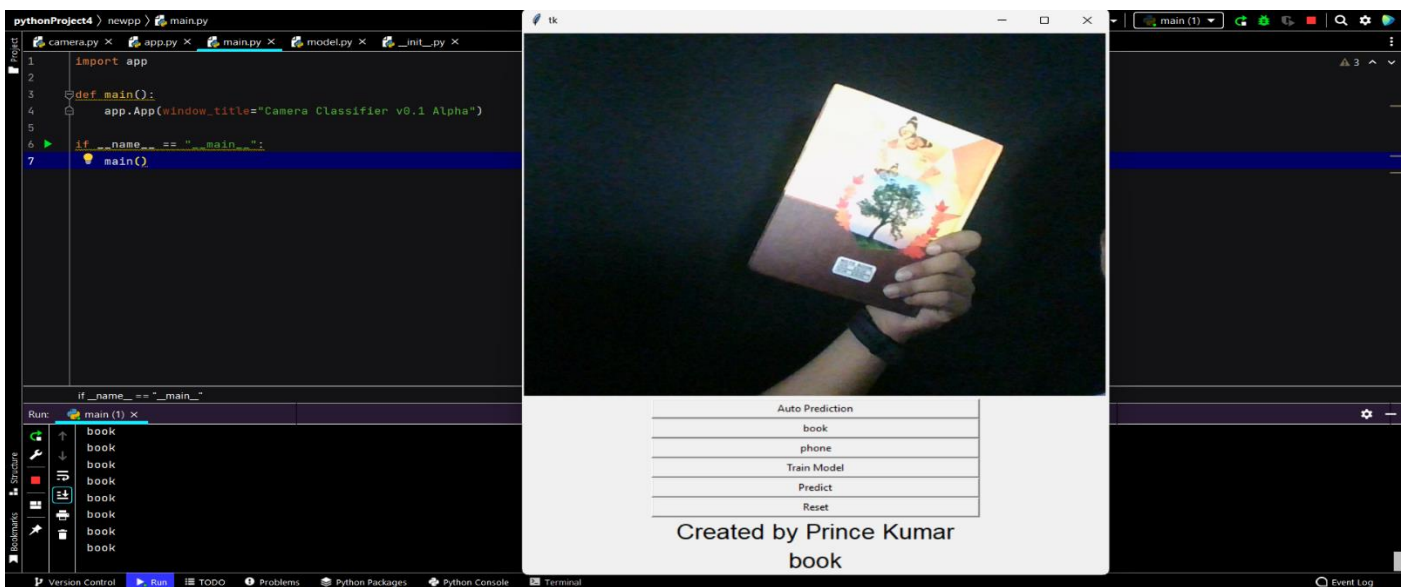
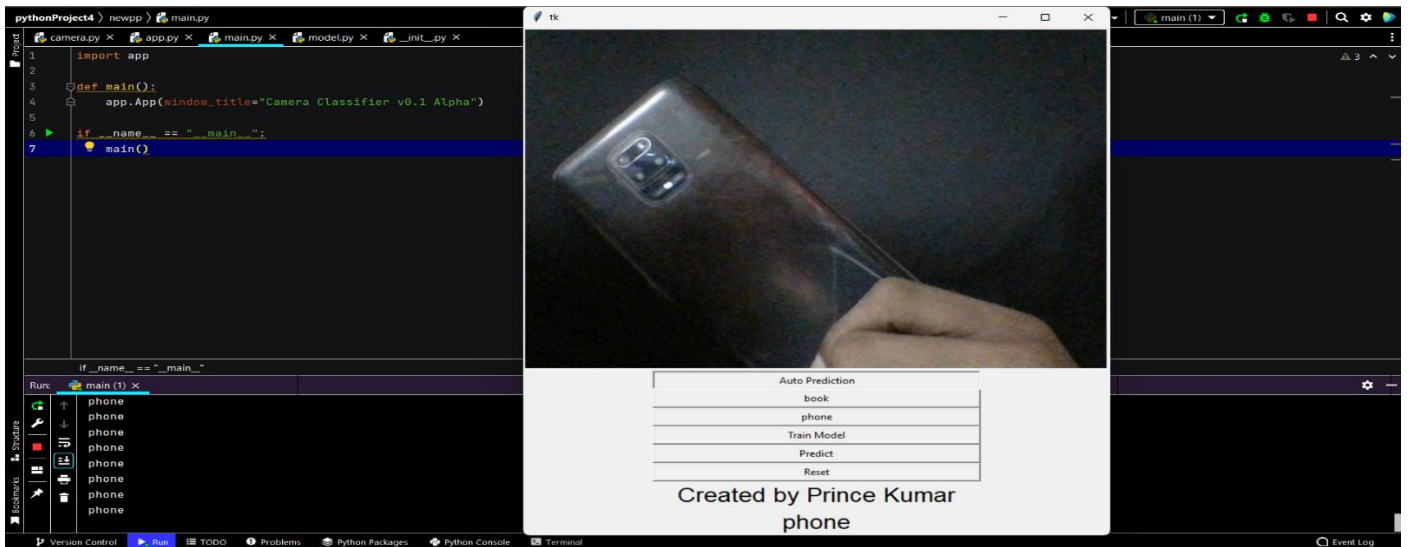
OUTPUT











APPLICATIONS

Here are a some of the future implementation of object detection.

1. Face detections and recognition:

Face detection perhaps be a separate class of object detection. We wonder how some applications like Facebook, Faceapp, etc., detect and recognize our faces. this is often a sample example of object detection in our day to day life. Face detection is already in use in our lifestyle to unlock our mobile phones and for other security systems to scale back rate .

2. Object tracking:

Object detection is additionally utilized in tracking objects like tracking an individual and his actions, continuously monitoring a ball within the game of Football or Cricket. As there's an enormous interest for people in these games, these tracking techniques enables them to know it during a better way and obtain some additional information. Tracking of the ball is of maximal importance in any ball-based games to automatically record the movement of the ball and adjust the video frame accordingly.

3. Self-driving cars:

This is often one among the main evolutions of the planet and is that the best example why we'd like object detection. so as for a car to travel to the specified destination automatically with none human interference or to form decisions whether to accelerate or to use brakes and to spot the objects around it. this needs object detection.

4. Emotions detection:

This permits the system to spot the type of emotion the person puts on his face. the corporate Apple has already tried to use this by detecting the emotion of the user and converting it into a respective emoji within the smart phone.

5. Biometric identification through retina scan:

Retina scan through iris code is one among the techniques utilized in high security systems because it is one among the foremost accurate and unique biometric.

6. Smart text search and text selection (Google lens):

In recent times, we've encountered an application in smart phones called google lens. this will recognize the text and also images and search the relevant information within the browser without much effort.

CONCLUSION

Deep-learning based object detection has been a search hotspot in recent years. This project starts on generic object detection pipelines which give base architectures for other related tasks. With the assistance of this the 3 other common tasks, namely object detection, face detection and pedestrian detection, are often accomplished. Authors accomplished this by combining 2 things: Object detection with deep learning and OpenCV and Efficient, threaded video streams with OpenCV. The camera sensor noise and lightening condition can change the result because it can create problem in recognizing the objects. generally, this whole process requires GPU's rather than CPU's. But we've done using CPU's and executes in much less time, making it efficient. Object Detection algorithms act as a mixture of both image classification and object localization. It takes the given image as input and produces the output having the bounding boxes adequate to the amount of objects present within the image with the category label attached to every bounding box at the highest. It projects the scenario of the bounding box up the shape of position, height and width.

REFERENCES

1. Google
2. Github
3. Stackoverflow
4. Youtube