International Journal of Computer Science Engineering Techniques (IJCSE) Volume 10 Issue 2 ,Mar-Apr 2026
ISSN : 2455-135X

"Real Time Object Detection System Using Machine"

Haidara Ahmad Alkenj[1*], B.L. Pal[2]

[1*]M.Tech., Faculty of Engineering & Technology Department of Computer Science & Engineering, Mewar University, Chittorgarh (Raj.) 312901.

[2] Associate Prof., Faculty of Engineering & Technology Department of Computer Science & Engineering, Mewar University, Chittorgarh (Raj.) 312901.

Corresponding Author: **Haidara Ahmad Alkenj**
E-mail: haidaraalkenj55@gmail.com

ABSTRACT

Detecting objects is a basic computer vision task that entails the location and identification of objects in images or video frames. As the use of digital imaging devices (surveillance cameras, smartphones, and autonomous systems) grows fast, automated and effective visual analysis has gained a significant role. Conventional object detection systems used manual designed features and classical machine learning algorithms, which were frequently not able to cope with the complexity of the environment, lighting variations and various objects in a scene. Object detection systems have become more accurate and faster to detect objects because of the emergence of deep learning, specifically Convolutional Neural Networks (CNNs).

This paper is a project report on a real-time object detector based on the YOLO (You Only Look Once) algorithm. YOLO is a neural network model that uses the deep learning system to simultaneously predict object categories and their positioning in the form of a bounding box on one pass through an image. The proposed system is based on Python and deep learning frameworks like OpenCV and TensorFlow to develop and implement a model. The model is trained on the dataset of COCO (Common Objects in Context), including many labeled pictures of various objects categories.

The system takes the input in the form of a camera or video stream and conducts real-time detection, drawing a bounding box and labels on the types of objects that are recognized. The analysis of performance shows that the YOLO-based solution is very accurate in the detection rate and has a high processing rate, and hence is applicable in real-time scenarios like surveillance systems, traffic systems, driverless cars, and smart security systems. The article has brought to light the advantages of deep learning methods in enhancing automated visual recognition and has helped in the development of real-time object detection systems.

Keywords: Object Detection, machine learning, YOLO algorithm, computer vision, deep learning.

1 Introduction

Object detection is among the most significant fields of investigation in computer vision and artificial intelligence. It is concerned with the search and discovery of objects in pictures or video streams [1]. As digital imaging devices, including smartphones, surveillance cameras, and autonomous vehicles, are growing at a rapid pace, the volume of visual information has grown exponentially. To take the important data amid these huge amounts of

visual information one will need to have intelligent systems that can automatically identify and interpret objects. Conventional object detection methods were based on a large number of features that are designed manually and classical machine learning methods [2]. These techniques were mostly difficult to execute by humans and were restricted in the accuracy when it came to complex scenes. Nevertheless, with the advent of the concept of deep learning and specifically, the Convolutional Neural Networks (CNNs), circulating in the industry have reinvented the concept of computer vision [2]. Deep learning object detection algorithms can learn sophisticated features automatically with massive datasets and this results in dramatic change in detection accuracy and speed. One of the most popular of these modern techniques is the YOLO (You Only Look Once) algorithm which has the advantage of detecting objects in real time. Unlike the old techniques, which processed images in various steps, the YOLO processed images once in a neural network [3]. This enables the system to simultaneously forecast object categories and the location of bounding boxes thereby being very efficient in real-time environment where it is applicable to surveillance, autonomous driving, traffic scrutiny, and intelligent security systems, among others. This project is based on a real-time object detector with the help of machine learning applications and the YOLO algorithm. The system takes the input of cameras and classifies and detects objects in real time by placing bounding box on the identified objects and labeling them. The dataset employed to train the model is very large like the COCO dataset, and it is implemented in Python and deep learning frameworks like OpenCV and TensorFlow. The offered system will serve to showcase the possibility of a successful application of machine learning and computer vision strategies to track objects in real-time settings. This study will help to enhance automated visual recognition systems and emphasize the essential practice of deep learning-based object detection models [4].

1.1 Problem summary:

Compared to the simple image classification, object detection is much more complicated as it does not only have to recognize the type of an object but also its location in a picture. Images in the real world usually have more than one object that is of various sizes, positions and orientation thus making the detection process difficult.

The key issues that involve object detection are some of the following ones:

- The ability to locate several objects in the same image.
- Detection of objects at varying levels of size and distance.
- Having a large detection accuracy and fast processing speed.
- Dealing with differences in lighting, orientation of objects and the complexity of their backgrounds.

- The limited or unbalanced data during training.

Conventional computer vision techniques which use handcrafted features are not as efficient in these issues. Thus, modern algorithms that are able to learn complicated patterns autonomously and identify objects immediately are needed..

1.2 Aim and objective:

Aim

The overall purpose of the given project is to design and develop a real-time object detection system based on the machine learning approach and the YOLO algorithm to identify and track the objects in the pictures or video streams correctly.

Objectives

1. To examine the computer vision and machine learning concepts regarding the object detection concepts.

2. To evaluate various methods used in detecting objects and come up with the most effective method of object detection which may be implemented in real time.

3. The purpose of the research is to apply the YOLO algorithm to multiple objects that should be detected in images or video streams.

4. To generate the detection model with the help of an appropriate dataset like the COCO dataset.

5. To create a system that is able to identify real time objects based on camera input.

6. To measure the accuracy of the proposed system in detecting and processing images in terms of accuracy and speed.

1.3 Problem specification:

The issue of identifying and labeling a variable number of objects on a picture is known as object detection. It is the significant variance that is the variable part. Unlike such issues as classification, the length of the output of object detection can vary, as the quantity of detected objects might vary image to image [5].

Object detection is the issue of seeking and classifying variable number of objects in an image. The significant distinction is the part variable. The output of object detection, unlike issues such as classification, is variable in length, due to the possibility of the number of objects detected in an image varying across images [6].

In case there are several object to be localized on an image, then we apply multiple object detection. Neural network, as in the case of the localization of objects, will result in the generation of 7 output vectors, although on a

grid-by-grid basis. A grid 4 divides one image by 4 or 16 divides it by 16 etc.. This example has 4 x 4 grid [7].

We take into consideration the issue about recognizing a great amount of various types of objects in ambiguous situations. Multi-class object detection includes sub-sequentially applying a battery of various classifiers, on the image, at different locations, and scales.

Deep learning object detection offers a rapid and accurate method to forecast object position of an image. Deep learning is an effective machine learning method where the object detector or classifier learns automatically the features found in the image need to process them in detection [8].

1.4 Literature review:

Since the problem of object detection with the help of machine learning algorithms is incredibly important in the various spheres of our life there are too many researches in the last 10 years focus on this problem and implemented it during the research and they offered a traditional approach to detect the objects in the images with Hough transform and they performed the approach with the SVM classifier and they obtained a good accuracy which is about 91 percent [9]. Through this paper they have reviewed all the deep learning methods which applied the deep learning methods to identify objects in images and videos and they have concluded that yolo algorithm is the best among all as it is a high-resolution object detection at low cost through a small number of network applications [10]. Through this research we learnt how to implement the neural network to object detection and have known the importance of deep learning in object detection a multiscale inference procedure that is capable of generating high-resolution object detection at a low cost by a small number of network applications. The analytical excellence of the method is demonstrated on Pascal VOC [11]. Propose a saliency-inspired neural network framework to detection which appraises a pool of class-agnostic bounding boxes and one score per box, indicating its probability of understanding any object of interest. The model itself assumes variable number of instances per class and also supports cross class generalization at the utmost levels of the network [12]. They compare the performance of deep learning models to identify objects in real time video stream in mobile devices and the inference delay with object detection performance as an end-toend solution or feature extractor to the performance of existing algorithms. Our findings indicate that there is a greater improvement in the object detection performance compared to the existing algorithms through application of transfer learning on mobile-based neural networks [13]. The classification of objects in the environment is done using Convolutional Neural Networks (CNN). Two states of art models are compared which are Single Shot Multi-Box Detector (SSD) using MobileNetV1 and a Faster Region-based Convolutional Neural Network (Faster-RCNN)

using InceptionV2. It can be seen that result is that one model would suit real-time application due to speed and the other would be applicable in more specific detection of objects [14]. But also define a trainable object detector and the detectors used to detect faces and cars of any size, location and pose. In order to adapt to change in the object orientation, the detector employs multiple classifiers, each covering a variable range of orientation. All these classifiers establish the presence of an object at a given size within a predetermined size image window. In order to locate the object in any given location and size, these classifiers exhaustively scan off the image [15]. Details a machine learning technique of detecting visual objects that will be capable of processing pictures at very high rates and still attains high detection rates. There are three notable contributions in this work [16].

An object detection task is among the essential tasks of computer vision. The most widespread paradigm to deal with this issue is to train object detectors functioning on an extra sub-image and exhaustively apply these detectors on all locations and scale. A discriminatively trained Deformable Part Model (DPM) successfully used this paradigm to obtain state-of-art detectors [17]. The large-scale and exhaustive search of all potential locations and scale is a computational problem. It is further complicated by the fact that this task is more difficult when a larger number of classes is involved because most of the methods are trained to use a different detector per class. To deal with this problem various approaches were suggested ranging between detector cascades, to applying segmentation to provide a few object hypotheses.

The amount of literature that has studied object detection is extensive, and here we will concentrate on a set of approaches that utilized class-agnostic concepts and overcame scalability issues. Several of the suggested detection methods are founded on 1 part-based models [18], which, more recently, have performed significantly when using discriminative learning and well-thought out features [19]. Such techniques are however based on the protracted use of the part templates across many scales and therefore costly. In addition, they grow in-proportion to the number of classes, making this a problem with large-scale datasets like ImageNet 1. As a solution to the former problem, Lampert et al. [20] employ a branch-bound methodology that does not require determining the locations of all possible objects. In order to solve the latter problem, Song et al. [21] apply lowdimensional part basis, which is common to all object classes. Hashing based method of efficient detection of parts has also performed well [22]. Another branch of research, more similar to ours, is that which is founded on the notion that it is indeed possible to localize objects without learning what they are. Part of these techniques are based on bottom-up classless segmentation [23]. Top-down feedback can be seen as the way to score the segments that were obtained in this manner [17, 2, 4]. Based on the identical motivation, the Alexe et al. [1] score

the object hypotheses using a cheap classifier with either an object or not to decrease the amount of location in the subsequent detection steps in this manner. The approaches may be considered as multi-layered models, where the segmentation is assumed as the first layer and a segment classification as the second layer. Although they represent established rules of perception, we will demonstrate that, under better conditions, having deeper models that have been acquired in full can result in better performance. Lastly, we are utilizing the current developments in Deep Learning, the most conspicuous being the one by Krizhevsky et al. [23]. Our scaling problem is the extension of their regression approach to bounding box using detections to the case of multiple object handling. Szegedy et al. [24] examine the DNN-based regression on object masks. The latter has state-of-art detection performance on VOC2007 but can not scale to many classes because the cost of a single mask regression has to be paid at inference-time: in such a system, at inference-time one must run 5 networks each, which is not scalable to many classes. real-world applications.

Bounding box: we represent the top-left and bottom-right corner of each box using four node values that could be represented as a vector li [?]. R 4 . These coordinates are brought to normalized w. r. t. image dimensions to make them independent of actual image size. The results of the linear transformation of the final hidden layer are the normalized coordinates. Confidence: the confidence of the box that contains an object is also coded in a single node value ci [?] [0, 1]. Such value is generated by linear transformation of the final hidden layer and a sigmoid. We may take a combination of the bounding box locations li, i [?] {1, . . . K}, as one linear layer. Just as we may do with collection of all confidences ci, i [?] {1, . . . K= b(L= 1 sigmoid output. Both these output layers are linked to the final hidden layers. On the inference time, our algorithm generates K bounding boxes. We take K =100 and 200 in our experiments. We can retrieve fewer high-confidence boxes at inference time, though, as desired, by simply using the confidence scores and non-maximum suppression. These are boxes that are expected to symbolize objects. They may thus be grouped together with a second group to enable object recognition. The size of the number of boxes is extremely small, making it possible to afford powerful classifiers. We employ second DNN in classification in our experiments [24]. Training Objective We train a DNN to achieve bounding boxes and its corresponding confidence score on each training image so that the top-ranking bounding boxes align well with the ground truth object bounding boxes on the image.

1.5 Plan of the work:

Considering the background work we have found that the most popular deep learning algorithm used in the detection of object is known as yolo (You Only Look Once) reasons being that we will start building the yolo algorithm model with its dataset and then train the model then test it in real time and then we

will build an application to detect object in real time.

1.6 Tools and requirements:

1.6.1 Laptop with high specification.

1.6.2 Camera.

1.6.3 Python.

1.6.4 Yolo algorithm

YOLO is an algorithm that is based on neural networks to achieve real-time object detection. It is a popular algorithm due to the speed and accuracy. It has been applied over all in different aspects to discern traffic lights, individuals, parking machines, and beasts.

1.6.5 Coco dataset

CoCo (official site) dataset, or Common Objects In Context, is a set of challenging, high quality, datasets in computer vision, mostly the latest neural networks. A format of the said datasets is also referred to as this name. COCO dataset consists of difficult and high quality visual datasets on computer vision which are primarily state of the art neural networks. As an example, to compare the performance of real time object detection, algorithms are often compared to COCO.

1.7 computer vision:

Computer vision is a branch under artificial intelligence (AI) that allows computers and systems to extract meaningful data based on the nature of the digital data (images, videos and other visual representations) inputted into them, and act, or give recommendations on information obtained. When AI provides computers with the ability to think, the vision grants them the ability to see, observe, and comprehend. Computer vision operates in a highly similar way as the human vision, only it has an edge due to human presence. The human eye enjoys the privilege of decades of experience of how we know how to discriminate things, how distant they are, whether they are moving or not, and whether there is something wrong in a picture. Computer vision is predicting machines to do these things, however it must do so in a significantly shorter time using cameras, data and algorithms instead of retinas, optic nerves and a visual cortex. Since a system that is trained to scan the products or monitor a given production unit can scan thousands of products or processes per minute, detecting even imperceptible defects or problems, it can easily outperform the human beings.

1.8 How computer vision work:

Lots of data is required in computer vision. It carries out analysis of data repeatedly until it identifies differences and eventually identify images. As an illustration, a computer that will be used to identify the tires of a vehicle must be exposed to extensive amounts of tire pictures and objects of tires to comprehend the variations and understand a tire, and more so one without flaws.

This is done with two fundamental technologies: a form of machine learning endowed by the title of deep learning and a convolutional neural network (CNN).

Machine learning involves algorithmic models, which allow the computer to learn itself on the contextual information a visual data. When sufficient data is inputted to the model, the computer will look at the data and learn to distinguish one image as opposed to another. Algorithms allow the machine to self learn, and not is it programmed to identify an image.

A CNN assists a machine learning or deep learning model to "stare at images by dividing the images into pixels, and assigning the tags or labels. It comes up with predictions of what it is seeing using the labels to do convolutions (mathematically view a two-function to form a third-function), and predictions. The neural network applies convolution to the input and verifies whether the predictions it does is correct in a sequence of iterations until the predictions begin to materialize. It is also perceiving or visualising images in a similar manner as human beings.

Similar to a human in a process of making out an image at a distance, CNN initially identifies hard edges and simple shapes, and then provides information over as it iterates its predictions. Single images are comprehended with the help of a CNN. A similar app is a recurrent neural network (RNN) applied to video in order to assist computers in comprehending the connection between pictures in a sequence of frames between each other.

1.9 The history of computer vision:

Efforts to make machines perceive and decode visual information have been ongoing in the last 6 decades with scientists and engineers working to create a process through which machines can see. It was first experimented upon in 1959, after neurophysiology presented a series of images to a cat, in the attempt of trying to match a reaction within its brain. They found out that it is attracted to hard edges or lines and scientifically that meant that image processing began with simple shapes such as straight edges.

Around the same period, the first computers image scanning technology was invented and computers now digitized and obtained images. The other milestone was achieved in the year 1963 where computing technology enabled the ability to convert 2-dimensional images into threedimensional images. The birth of AI as an academic discipline of study came in the 1960s and the first attempt at finding a solution to the problem of solving the human vision problem was also initiated.

In 1974 they came up with optical character recognition (OCR) technology that had the ability to identify text that was printed in any font of typeface. Equally, intelligent character recognition (ICR) had the ability to read hand-written text by neural networks. OCR and ICR have since been used in document and invoice processing, vehicle plate

recognition, mobile payments, machine translation and other applications since that time.

Neuroscientist David Marr in 1982 developed the idea of the hierarchical nature of vision, and algorithms to make machines perceive edges, corners, curves etc. and the like. At the same time computer scientist Kunihiko Fukushima invented a network of cells that had the ability to identify patterns. The scheme was named the Neocognitron, and featured convolutional layers in a neural network.

By 2000, the recognition of the objects was the subject of research, and the first applications in the sphere of face recognition were developed in 2001. The process of standardization of visual data sets tagging and annotation was developed until the 2000s. ImageNet data set was made available in the year 2010. It had millions of tagged images in a thousand classes of objects and forms a basis of CNNs and deep learning models in use today. In 2012 a university of Toronto crew had put a CNN into a contest on image recognition. The image recognition error rate dropped considerably thanks to the model which was referred to as AlexNet. Since this breakthrough, the rates of errors have decreased down to few percent.

1.10Application of computer vision:

The computer vision field has a lot of research being conducted; however, research is not the only thing that is being conducted. Real-life applications show the significance of the computer vision to business, entertainment, transport, health and daily life ventures. One of the major stimulating factors leading to the development of these applications is the deluge of visual data streaming out of smartphones, security systems, traffic cameras and other gadgets that are visually instrumented. This data has the potential to become significant to the work in industries, however, as it is today not utilized. The intelligence provides a training ground to deploy computer vision programs and a catapult to them taking off into a variety of human pursuits:

The IBM vision was intended to develop My Moments in the 2018 Masters golf tournament. IBM Watson also had hundreds of hours of Masters footage, and could recognize the sights (and sounds) of important shots. It edited these highlights and presented it to the fans as a customized highlight reel.

The Google Translate application allows one to point a smartphone camera at any sign in a different language and nearly instantly receive a translated version of the sign in the language of his or her preference.

The creation of autonomous vehicles will require the use of computer vision to translate the information a car camera provides, among other visual sensors. The identification of all other cars, traffic lights, the lane markers, pedestrians, bicycles and other visuals that a person encounters on the road is necessary.

IBM is using computer vision technology along with other companies such as Verizon

to take intelligent AI to the edge, and to assist autopilot companies in the industry to detect quality flaws in a vehicle prior to its coming out of a production facility.

1.11Object detection:

Object Detection is an example of deep learning technology, the things, human, building, cars can be recognized as an object in picture and video. Object detection is simply to identify the object with the bounding box in the image whereas in image classification, we can simply either categorize(classify) that is an object in the image or not in terms of the likely (Probability). Others: The SoftMax function can make us know we could produce that picture of that kitten which is bound and one which is not bound which makes the whole difference between Image classification and Object detection.

1.12Machine learning and object detection:

Object detection method through machine learning must extract the feature manually by implementing Image based feature extraction method (Histogram of oriented gradient, Speeded-up robust features, Local binary patterns, Haar wavelets, Color histogram etc.) whereas Deep learning based object detection method automatically generates the feature of the image namely, edge, shape etc.

1.13How machine learning work with object detection:

In the case where the phenomenon being studied is to classify the class of the image as an input of the CNN, this is the problem of image classification and the probability values of the image classifying the image to any of the desired classes are given as shown in the figure below. Issue of Object detection has presupposed that even several classes of objects can be observed within an image simultaneously.

- It can also be viewed as such that there were two kinds of problems that one is multi label classification (multiple class in one image).
- Bounding Box (Regression Problem) where we need to estimate the values of coordinates of the bounding box, in terms of.

Figure 5 is an example of the case where an object appears in one image, yet there should be the possibility of detecting even several objects in one image.

1.14Object localization:

Object localization is a task that involves making predictions of an object in an image and its boundaries. The distinction between objects localization and objects detection is slight.

In simple terms, the object localization seeks to identify the foremost (or most conspicuous) object in a photograph whereas object

detection endeavours to determine all the objects and their edges.

The classification or image recognition model provides merely the likelihood of the presence of an object in an image. Contrary to this is object localization where the location of the object in the image was determined. An object localization algorithm will provide values of the position of an object relative to an image. The most used means of localization in an image is thus the use of bounding boxes to depict the location of an object in computer vision.

To create a Bounding Box with the parameters, which are provided as follows:

bx, by: the position of the positioning box center.

bw strength of the bounding box with reference to the image.

bh The present value of height of the bounding box with respect to image height.

2 analysis design methodology and implementation strategy:

2.1 observation matrix:

2.1.1 install tensorflow

There are two methods of installing TensorFlow Object Detection API using Python Package Installer (pip) or Docker, a free open source, platform that allows installed applications the ability to run in a container. To run Local Object Detection API of the Tensorflow API, Docker is suggested. Even when you are not introduced to Docker, perhaps finding its installation using pip is a better thing to do.

2.1.2 Docker installation:

From the root of the git
repository docker build -f
research/object
detection/docker
files/tf1/Docker file -t od.
docker run -it od's

2.1.4 Gathering data:

We collect our data from coco dataset.

2.2 Database design:

The first one is the configuration file the second file is a data file used to implement YOLO algorithm.

2.2.1 Net Section:

\\ batch=64- number of samples that will be processed in a single batch. • subdivisions=16- mini batch count in a batch; mini batch samples are run immediately through the GPUs; mini batches samples weights are modified at most 1 step run batch images. • width=608 -all the images will be resized to this value in the course of training and testing. height= 608- all the images will be downsized both during training and testing to this figure. • channels=3- in both training and testing, all images will be reduced to this number.

2.2.2 Optimization Section:

momentum=0.9 is hyperparameter of optimizer which determines the extent of the history that will affect additional weight updating. decay= 0.0005: decays the learning rate through the training time period learning rate=0.001: initial learning rate during training.

2.2.3 Training:

- angle=0 – the training parameter randomly rotates images.
- saturation=1.5 - parameter that varies randomly saturation of the images during the training.
- exposure=1.5 - randomly varying parameter of brightness of pictures in training.
- hue=.1- parameter that changes the hue of images randomly during training. 2.2.4 Weight file: this file has all weights of the neuron within yolo algorithm.

2.2.4 Weight file: this file contains all the weights for the neuron inside yolo algorithm.

2.3 System design:

2.3.1 Object:

The meaning of object here is purposely left as free of applications as possible. The object is believed to be a collection of samples which have some similar sense to the user. The samples are supposed to consist of fragments of parts of visible objects of interest (in a common sense) or represent special situations of interest as they are portrayed in the scanned scenes. The major aim of image mapping phase is to classify samples as part, or not part, of objects of certain classes as exhibited in the image. There are numerous benefits of this universal perception of objects which are discussed in more details below.

2.3.2 Training phase:

A session, during which a user determines the classes names (tag) of certain objects and feeds the system with tagged exemplars of the same that will then be used in key phase of classification in the image mapping processing to classify similar samples. A user is normally advised that so selected samples must show different sections of the object, and which will be recognizable without further information. The training phase is generally used through the iterative approach in which the user is free to form and refine training set that can result in improved classification. The final stage of the training phase is the initiation of special machine learning mechanism to produce the classifier.

2.3.3 Training set:

Sample set, which has class labels.

2.3.4 Feature vector:

Number or Boolean valued vectors, the products of a sample as determined by special algorithms of numerical features on a sample. Algorithms of numerical features used in the multipurpose system are designed and implemented further as discussed below. Image grid: Image locations (points), which are located at a constant step in both horizontal and vertical directions. Image mapping phase Under image mapping phase centers such positions are normally viewed as sample centres in automatic classification by

classifier. The position of image grid will be provided in our figures below. Image mapping phase: Classify samples, described by image grid points, by a classifier with a map of samples being supplied automatically. So formed class tags are normally accompanied with probability assessment values created by the classifier.

2.3.5 Classifier:

On A given feature vector input, a procedure, which employs special numeric feature algorithms to compute the feature vector, automatically finds the most probable tag of the class with the largest number of features set up. A special machine learning process typically generates a classifier, based on training set and the numerical feature algorithms. The probability assessment value is normally provided together with the class tag. Moreover, each sample will be classified by the probabilities that such a sample can be classified in all other classes available.

2.3.6 Object oriented design:

2.3.6.1 UML Activity Diagram:

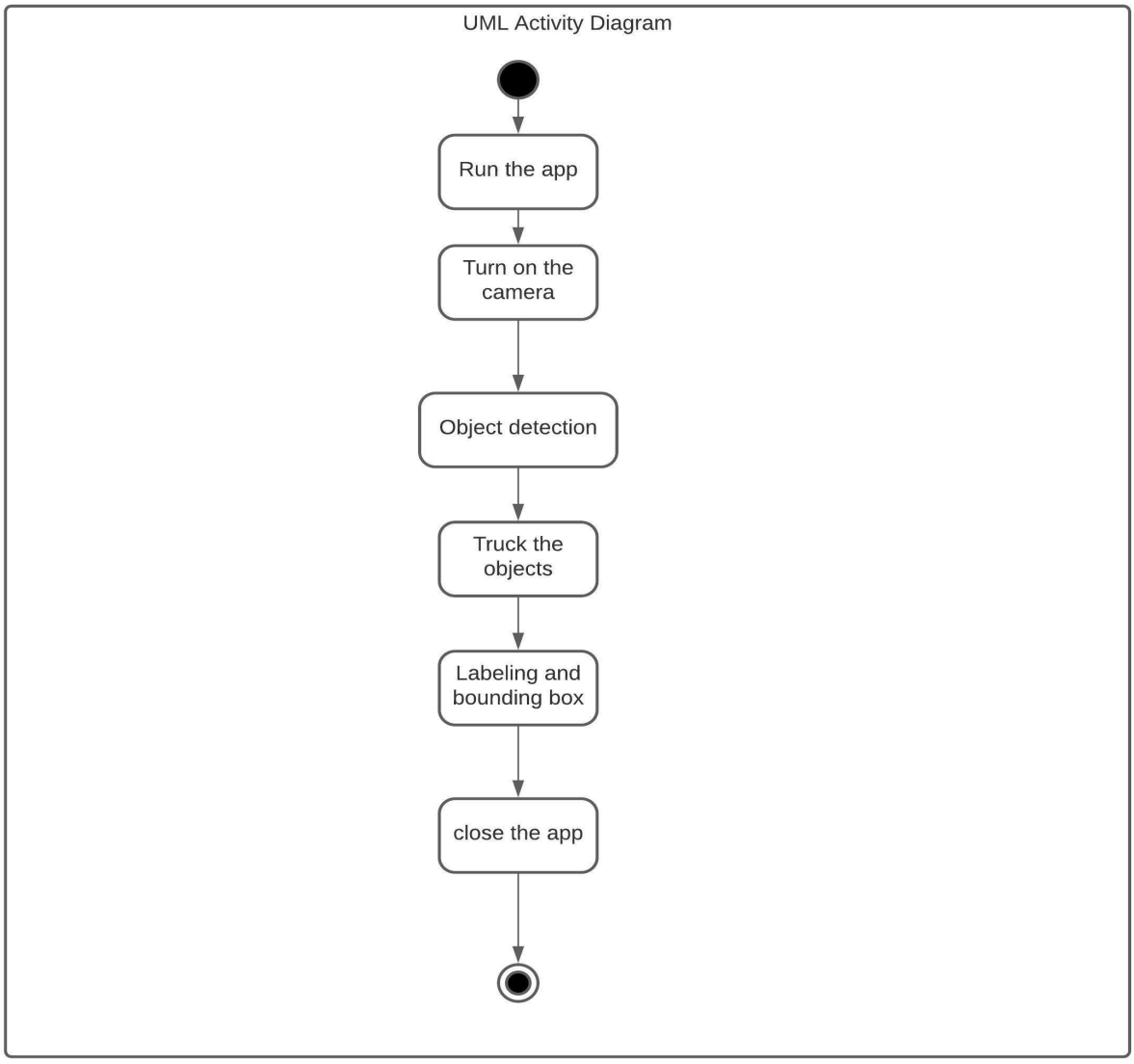

Figure 1 UML activity diagram

2.3.6.2 Case diagram:

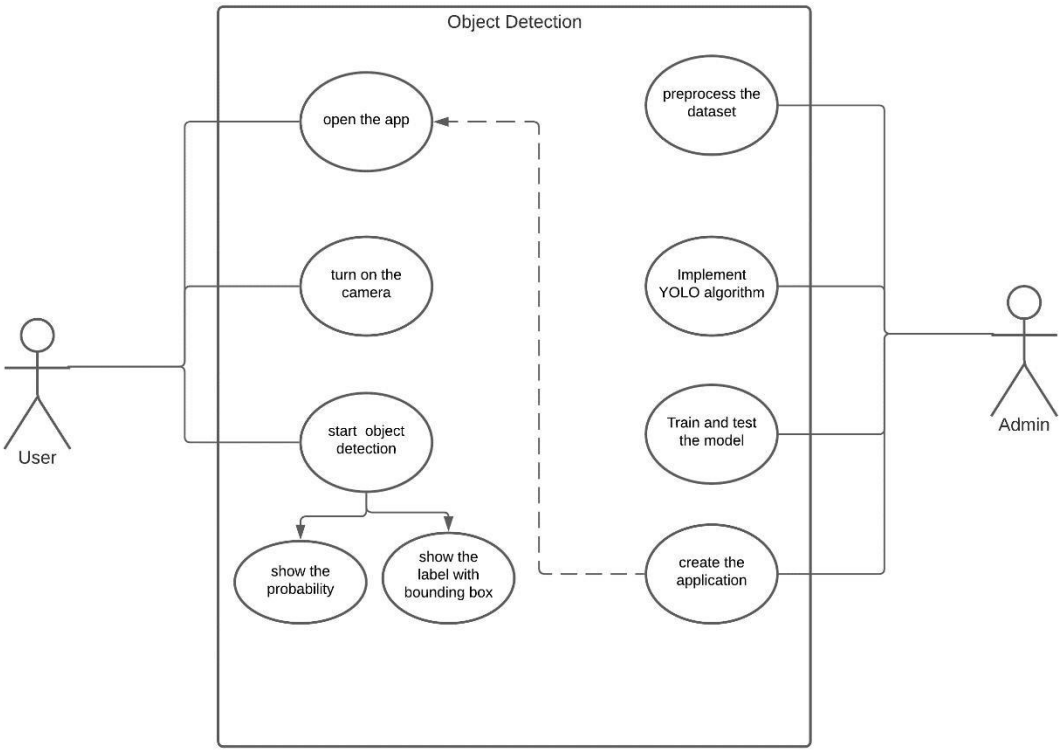

Figure 2 case diagram

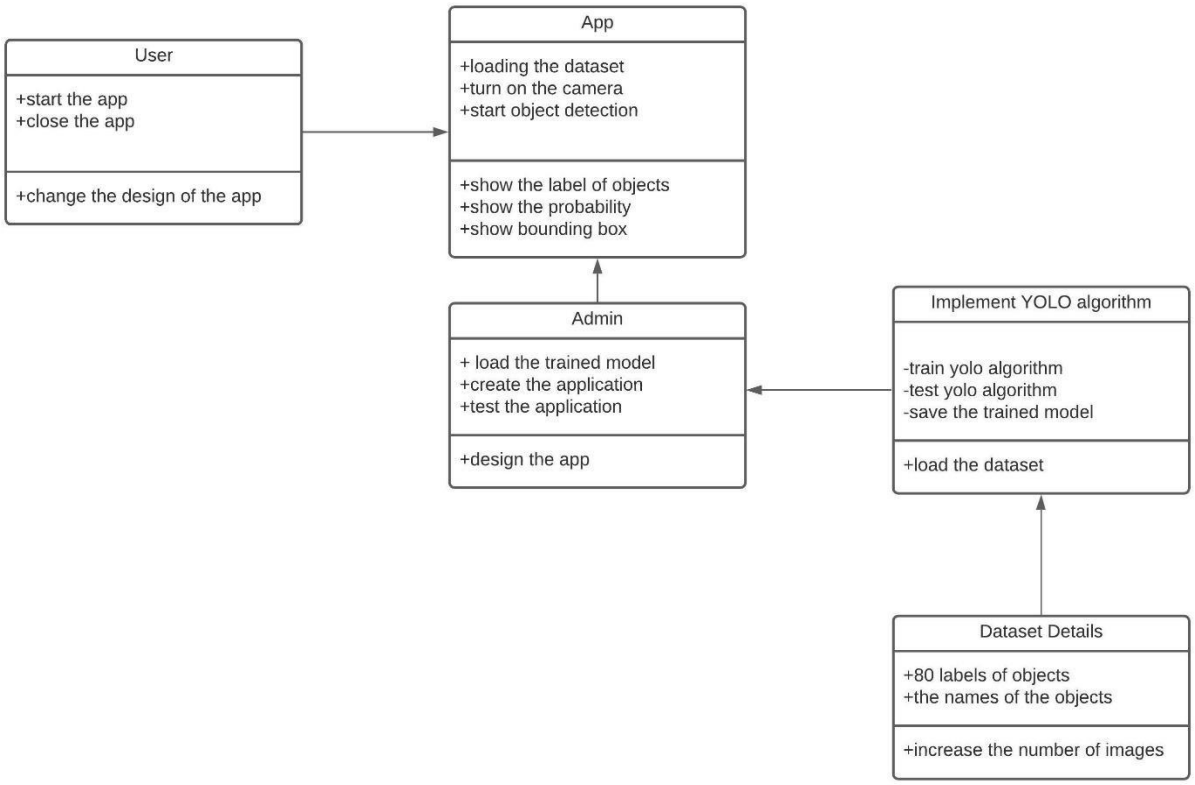

Figure 3 class diagram

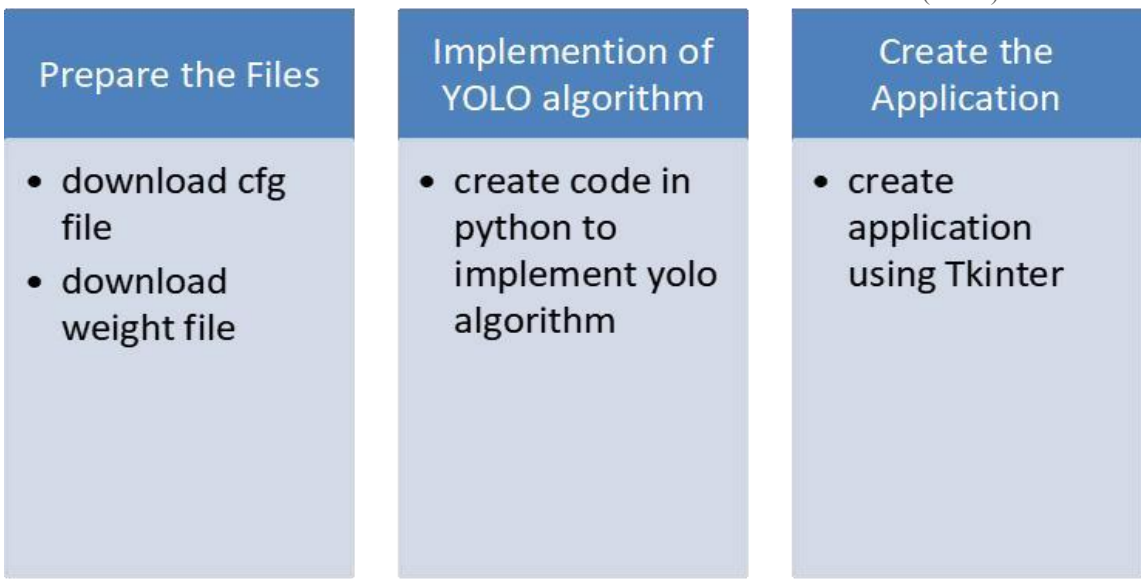

Figure -4 show the steps of implementation of our project

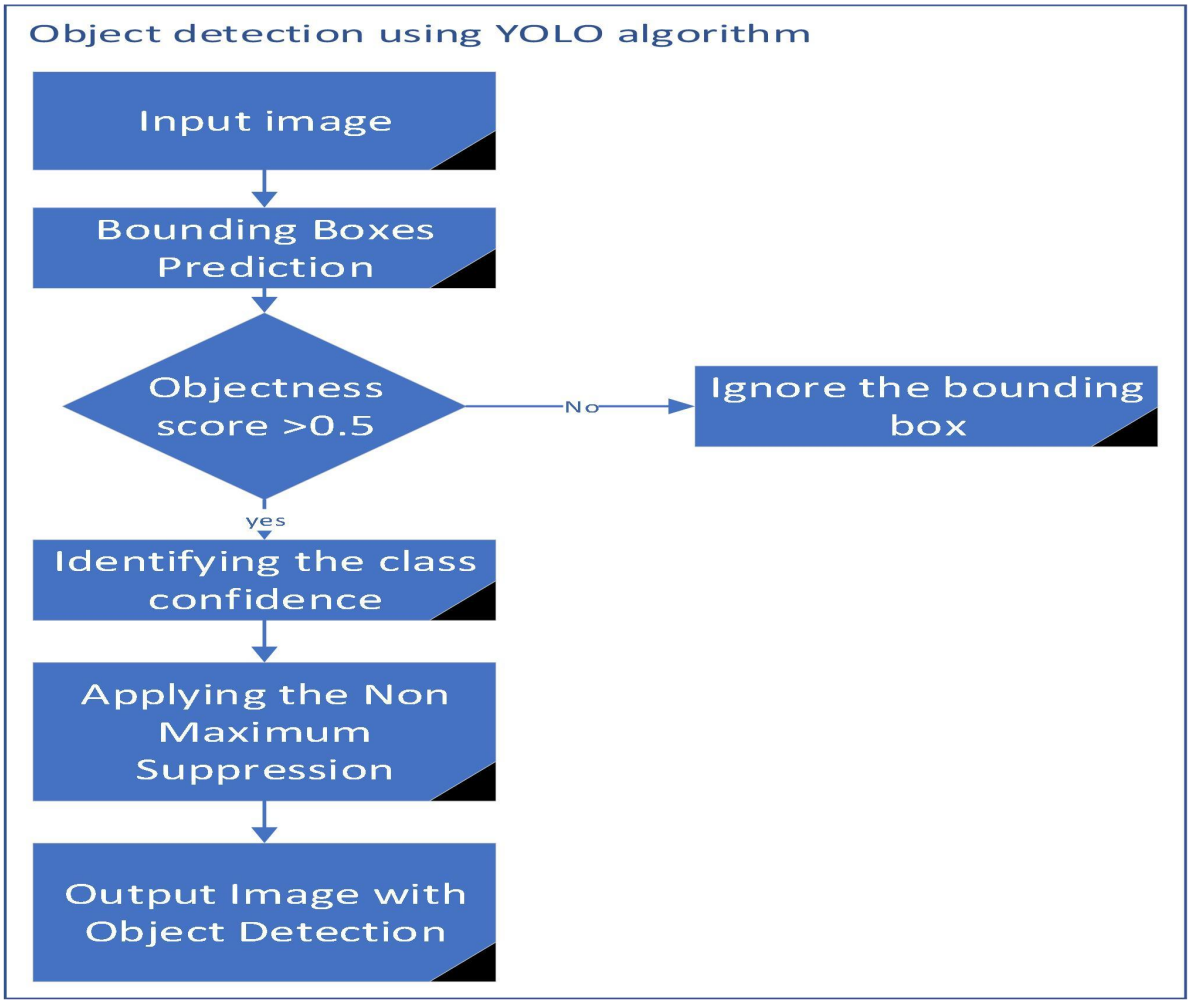

Figure -5 The steps of object detection using yolo algorithm

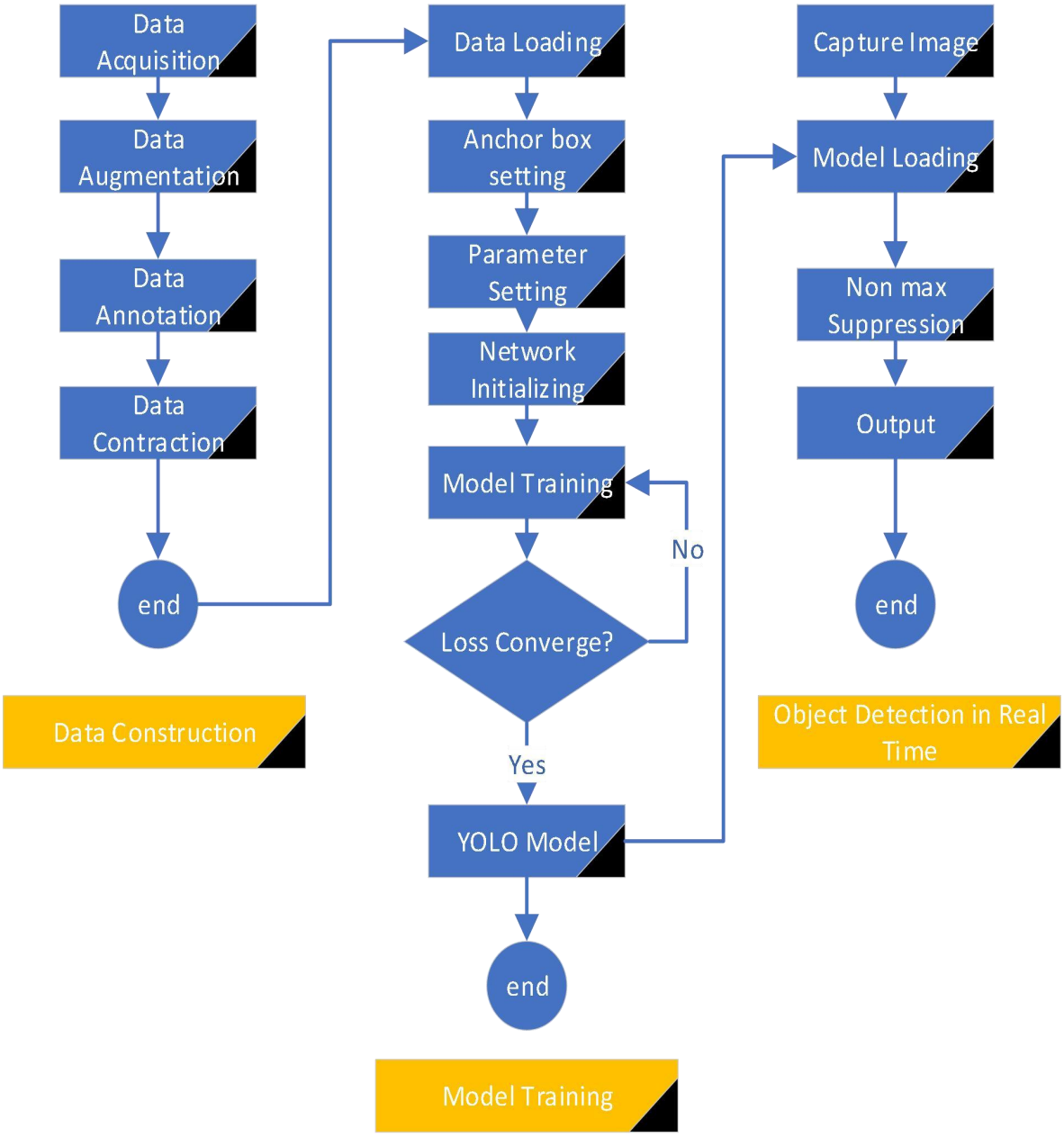

Figure -6 Flowchart of Implementation The algorithm

3 Implementation:

3.1 Implementation functionality:

Yolo is a neural network-based algorithm which gives real-time detection of objects. The popularity of this algorithm is due to its speed and accuracy. Other applications of it have been in the detection of traffic lights, human beings, parking machines and even animals. The YOLO (You Only Look Once) framework on the other hand directly approaches object detection differently. It learns the full image once and the coordinates of the bounding box and the probability of the classes of these boxes. The python code is used to implement the algorithm.

3.2 Yolo algorithm:

YOLO is an acronym of You Only Look Once and it employs the convolutional neural networks (CNN) to detect objects. In the case of YOLO, it is used to predict classification tags as well as find the position of objects. This is why, YOLO has the ability to recognize several objects in a single picture. The meaning of the name of

the algorithm implies that one network is utilized on entire image only once. YOLO splits up an image into regions and predicts bounding box on each of the regions and a probability. YOLO also estimates confidence of all bounding boxes providing information that object is actually contained in this specific box, and probability of object contained in binding box being of a specific type. Subsequently, bounding boxes are rejected using method known as non-maximum suppression which rejects certain bounding boxes in case of low confidence, or when there is a different bounding box on this region that possesses a better confidence. The latest version is YOLO-3 which uses successive 3x3 and 1x1 convolutional layers. It has a total of 53 convolutional layers with architecture as illustrated on the. Each of the layers is preceded by batch normalization and Leaky ReLU activation.

3.2.1 Importing needed libraries import

numpy as np import cv2 import time

Reading stream video from camera
"""

3.2.2 Defining 'Video Capture' object

3.2.3 reading stream video from camera

camera = cv2.VideoCapture(0)

3.2.4 Preparing variables for spatial dimensions of the frames h, w =

None, None **3.2.5 Loading COCO class labels from file**

```
# Opening file
# Pay attention! If you're using Windows, yours path might looks like: # r'yolo-
coco-data\coco.names' # or:
# 'yolo-coco-data\\coco.names' with open('yolo-coco-
data/coco.names') as f:
    # Getting labels reading every line # and
    putting them into the list labels = [line.strip()
    for line in f]
```

3.2.6 Check point

```
# print('List with labels names:')
# print(labels)

# Loading trained YOLO v3 Objects Detector
# with the help of 'dnn' library from OpenCV
# Pay attention! If you're using Windows, yours paths might look like:
# r'yolo-coco-data\yolov3.cfg' # r'yolo-coco-
data\yolov3.weights' # or:
```

```
# 'yolo-coco-data\\yolov3.cfg' # 'yolo-coco-
data\\yolov3.weights'
network = cv2.dnn.readNetFromDarknet('yolo-coco-data/yolov3.cfg',
                                     'yolo-coco-data/yolov3.weights')
```

3.2.7 Getting list with names of all layers from YOLO v3 network

```
layers_names_all =
network.getLayerNames()

# # Check point
# print()
# print(layers_names_all)

# Getting only output layers' names that we need from YOLO v3 algorithm # with
function that returns indexes of layers with unconnected outputs layers_names_output
= \
    [layers_names_all[i[0] - 1] for i in network.getUnconnectedOutLayers()] # # Check
point
# print()
# print(layers_names_output) # ['yolo_82', 'yolo_94', 'yolo_106']
```

3.2.8 Setting minimum probability to eliminate weak predictions

```
probability_minimum = 0.5
```

3.2.9 Setting threshold for filtering weak bounding boxes

```
# with non-maximum suppression threshold = 0.3
```

3.2.10 Generating colours for representing every detected object

```
# with function randint(low, high=None, size=None, dtype='l') colours =
np.random.randint(0, 255, size=(len(labels), 3), dtype='uint8')
```

3.2.11 Check point

```
# print()
# print(type(colours)) # <class 'numpy.ndarray'>
# print(colours.shape) # (80, 3)
# print(colours[0]) # [172 10 127]

"""
End of:
Loading YOLO v3 network
""" """
Start of:
Reading frames in the loop
"""
```

3.2.12 Defining loop for catching frames

```
while True:
```

```
# Capturing frame-by-frame from camera _, frame =
camera.read()

# Getting spatial dimensions of the frame
# we do it only once from the very beginning # all
other frames have the same dimension if w is None or
h is None:
    # Slicing from tuple only first two elements h, w =
    frame.shape[:2]
""" Start of:
```

3.2.13 Getting blob from current frame

```
"""

# Getting blob from current frame
# The 'cv2.dnn.blobFromImage' function returns 4-dimensional blob from current


# frame after mean subtraction, normalizing, and RB channels swapping # Resulted
shape has number of frames, number of channels, width and height # E.G.:
# blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean, swapRB=True) blob =
cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True, crop=False)

"""
End of:
Getting blob from current frame
```

3.2.14 Implementing forward pass with our blob and only through output layers #

```
Calculating at the same time, needed time for forward pass network.setInput(blob) # setting

blob as input to the network start = time.time() output_from_network =

network.forward(layers_names_output) end = time.time()
```

3.2.15 Showing spent time for single current frame print('Current frame

```
took {:.5f} seconds'.format(end - start))
```

3.2.16 Preparing lists for detected bounding boxes, # obtained

```
confidences and class's number bounding_boxes = []

confidences = [] class_numbers = []
```

3.2.17 Going through all output layers after feed forward pass for result in

```
output_from_network:
```

```
# Going through all detections from current output layer for

detected_objects in result:


    # Getting 80 classes' probabilities for current detected object scores =

    detected_objects[5:]

    # Getting index of the class with the maximum value of probability

    class_current = np.argmax(scores) # Getting value of probability for defined

    class confidence_current = scores[class_current]

    # # Check point

    # # Every 'detected_objects' numpy array has first 4 numbers with

    # # bounding box coordinates and rest 80 with probabilities

    # # for every class

    # print(detected_objects.shape) # (85,)
```

3.2.18 Eliminating weak predictions with minimum probability

```
    if confidence_current > probability_minimum:

        # Scaling bounding box coordinates to the initial frame size

        # YOLO data format keeps coordinates for center of bounding box

        # and its current width and height

        # That is why we can just multiply them elementwise

        # to the width and height

        # of the original frame and in this way get coordinates for center # of

        bounding box, its width and height for original frame box_current =

        detected_objects[0:4] * np.array([w, h, w, h]) Now, from YOLO data

        format, we can get top left corner coordinates
```

```
        # that are x_min and y_min x_center, y_center, box_width,

        box_height = box_current x_min = int(x_center - (box_width / 2))

        y_min = int(y_center - (box_height / 2)) # Adding results into

        prepared lists bounding_boxes.append([x_min, y_min,

                                int(box_width), int(box_height)])

        confidences.append(float(confidence_current)) class_numbers.append(class_current)
```

"""

End of:

Getting bounding boxes

"""

3.2.19 Implementing non-maximum suppression of given bounding boxes

```
# With this technique we exclude some of bounding boxes if their

# corresponding confidences are low or there is another

# bounding box for this region with higher confidence

# It is needed to make sure that data type of the boxes is 'int'

# and data type of the confidences is 'float'

# https://github.com/opencv/opencv/issues/12789

results = cv2.dnn.NMSBoxes(bounding_boxes, confidences,

                probability_minimum, threshold)
```

3.2.20 Checking if there is at least one detected object # after

```
non-maximum suppression if len(results) > 0:

    # Going through indexes of results

    for i in results.flatten():
```

```
        # Getting current bounding box coordinates,

        # its width and height x_min, y_min = bounding_boxes[i][0],

        bounding_boxes[i][1] box_width, box_height = bounding_boxes[i][2],

        bounding_boxes[i][3]



        # Preparing colour for current bounding box # and converting

        from numpy array to list colour_box_current =

        colours[class_numbers[i]].tolist()

        # # # Check point

        # print(type(colour_box_current)) # <class 'list'>

        # print(colour_box_current) # [172 , 10, 127] # Drawing

        bounding box on the original current frame cv2.rectangle(frame,

        (x_min, y_min),

                        (x_min + box_width, y_min + box_height),

                colour_box_current, 2)

# Preparing text with label and confidence for current bounding box text_box_current = '{}:

                        {:.4f}'.format(labels[int(class_numbers[i])],

                                confidences[i])

        # Putting text with label and confidence on the original image cv2.putText(frame,

        text_box_current, (x_min, y_min - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,

        colour_box_current,

    End of:

    Drawing bounding boxes and label
```

Course: Training YOLO v3 for Objects Detection with Custom Data

Section-2

Objects Detection on Video with YOLO v3 and OpenCV

File: yolo-3-video.py

"""

Detecting Objects on Video with OpenCV deep learning library

#

Algorithm:

Reading input video --> Loading YOLO v3 Network -->

--> Reading frames in the loop --> Getting blob from the frame -->

--> Implementing Forward Pass --> Getting Bounding Boxes -->

--> Non-maximum Suppression --> Drawing Bounding Boxes with Labels -->

--> Writing processed frames

#

Result:

New video file with Detected Objects, Bounding Boxes and Labels

Importing needed libraries import

numpy as np import cv2 import time

"""

Start of:

Reading input video

"""

```
# Defining 'VideoCapture' object

# and reading video from a file

# Pay attention! If you're using Windows, the path might looks like:

# r'videos\traffic-cars.mp4' # or:

# 'videos\\traffic-cars.mp4'

video = cv2.VideoCapture('videos/traffic-cars.mp4')

# Preparing variable for writer # that we will use to

write processed frames writer = None

# Preparing variables for spatial dimensions of the frames

h, w = None, None

"""



End of: Reading input vide

Start of:

Loading YOLO v3 network

# Loading COCO class labels from file

# Opening file

# Pay attention! If you're using Windows, yours path might looks like:

# r'yolo-coco-data\coco.names' # or:

#  'yolo-coco-data\\coco.names'  with  open('yolo-coco-

data/coco.names') as f:

    # Getting labels reading every line

    # and putting them into the list
```

```
    labels = [line.strip() for line in f]

# # Check point

# print('List with labels names:')

# print(labels)

# Loading trained YOLO v3 Objects Detector

# with the help of 'dnn' library from OpenCV

# Pay attention! If you're using Windows, yours paths might look like:

# r'yolo-coco-data\yolov3.cfg' # r'yolo-coco-

data\yolov3.weights' # or:

# 'yolo-coco-data\\yolov3.cfg' # 'yolo-coco-data\\yolov3.weights' network =

cv2.dnn.readNetFromDarknet('yolo-coco-data/yolov3.cfg',

                           'yolo-coco-data/yolov3.weights')


# Getting list with names of all layers from YOLO v3 network layers_names_all =

network.getLayerNames()

# # Check point

# print()

# print(layers_names_all)

# Getting only output layers' names that we need from YOLO v3 algorithm # with function that

returns indexes of layers with unconnected outputs layers_names_output = \

        [layers_names_all[i[0] - 1] for i in network.getUnconnectedOutLayers()]

# # Check point

# print()
```

```
# print(layers_names_output) # ['yolo_82', 'yolo_94', 'yolo_106']



# Setting minimum probability to eliminate weak predictions probability_minimum = 0.5

# Setting threshold for filtering weak bounding boxes

# with non-maximum suppression threshold = 0.3

# Generating colours for representing every detected object # with function

randint(low, high=None, size=None, dtype='l') colours = np.random.randint(0,

255, size=(len(labels), 3), dtype='uint8')

# # Check point

# print()

# print(type(colours)) # <class 'numpy.ndarray'>

# print(colours.shape) # (80, 3)

# print(colours[0]) # [172 10 127]

"""

End of:

Loading YOLO v3 network

"""

"""

Start of:

Reading frames in the loop

"""



# Defining variable for counting frames
```

```
# At the end we will show total amount of processed frames f = 0

# Defining variable for counting total time

# At the end we will show time spent for processing all frames t = 0

# Defining loop for catching frames while True:

    # Capturing frame-by-frame ret, frame =

    video.read()

    # If the frame was not retrieved

    # e.g.: at the end of the video,

    # then we break the loop

    if not ret:

        break

    # Getting spatial dimensions of the frame

    # we do it only once from the very beginning # all

    other frames have the same dimension if w is None or

    h is None:

        # Slicing from tuple only first two elements h, w =

        frame.shape[:2]

    """




    Start of:

    Getting blob from current frame

    """
```

```
# Getting blob from current frame

# The 'cv2.dnn.blobFromImage' function returns 4-dimensional blob from current

# frame after mean subtraction, normalizing, and RB channels swapping # Resulted

shape has number of frames, number of channels, width and height

# E.G.:

# blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean, swapRB=True) blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),

                    swapRB=True, crop=False)

"""

End of:

Getting blob from current frame

"""

"""

Start of:

Implementing Forward pass

"""



# Implementing forward pass with our blob and only through output layers #

Calculating at the same time, needed time for forward pass network.setInput(blob) #

setting blob as input to the network start = time.time() output_from_network =

network.forward(layers_names_output) end = time.time()

# Increasing counters for frames and total time

f += 1
```

```
t += end - start

# Showing spent time for single current frame print('Frame number {0} took

{1:.5f} seconds'.format(f, end - start))

"""

End of:

Implementing Forward pass

"""

"""



Start of:

Getting bounding boxes

"""



# Preparing lists for detected bounding boxes, #

obtained confidences and class's number

bounding_boxes = [] confidences = [] class_numbers =

[]

# Going through all output layers after feed forward pass for result in

output_from_network:

    # Going through all detections from current output layer

    for detected_objects in result:

        # Getting 80 classes' probabilities for current detected object scores =

        detected_objects[5:]
```

```
# Getting index of the class with the maximum value of probability

class_current = np.argmax(scores) # Getting value of probability for defined

class confidence_current = scores[class_current]

# # Check point

# # Every 'detected_objects' numpy array has first 4 numbers with

# # bounding box coordinates and rest 80 with probabilities

# # for every class

# print(detected_objects.shape) # (85,)


# Eliminating weak predictions with minimum probability if

confidence_current > probability_minimum:

    # Scaling bounding box coordinates to the initial frame size

    # YOLO data format keeps coordinates for center of bounding box

    # and its current width and height

    # That is why we can just multiply them elementwise

    # to the width and height

    # of the original frame and in this way get coordinates for center

    # of bounding box, its width and height for original frame box_current =

    detected_objects[0:4] * np.array([w, h, w, h])

                # Now, from YOLO data format, we can get top left corner coordinates

    # that are x_min and y_min x_center, y_center, box_width,

    box_height = box_current x_min = int(x_center - (box_width / 2))
```

```
            y_min = int(y_center - (box_height / 2)) # Adding results into

            prepared lists bounding_boxes.append([x_min, y_min,

                                    int(box_width), int(box_height)])

            confidences.append(float(confidence_current)) class_numbers.append(class_current)



"""

End of:

Getting bounding boxes

"""

"""

Start of:

Non-maximum suppression

"""

# Implementing non-maximum suppression of given bounding boxes

# With this technique we exclude some of bounding boxes if their

# corresponding confidences are low or there is another

# bounding box for this region with higher confidence

# It is needed to make sure that data type of the boxes is 'int'

# and data type of the confidences is 'float' #

https://github.com/opencv/opencv/issues/12789 results =

cv2.dnn.NMSBoxes(bounding_boxes, confidences,

                            probability_minimum, threshold)
```

```
"""
End of:
Non-maximum suppression
"""



"""
Start of:
Drawing bounding boxes and labels
"""
# Checking if there is at least one detected object # after
non-maximum suppression if len(results) > 0:
    # Going through indexes of results
    for i in results.flatten():
        # Getting current bounding box coordinates,
        # its width and height x_min, y_min = bounding_boxes[i][0],
        bounding_boxes[i][1] box_width, box_height = bounding_boxes[i][2],
        bounding_boxes[i][3]
        # Preparing colour for current bounding box # and converting
        from numpy array to list colour_box_current =
        colours[class_numbers[i]].tolist()
        # # # Check point
        # print(type(colour_box_current)) # <class 'list'>
        # print(colour_box_current) # [172 , 10, 127]
```

```
        # Drawing bounding box on the original current frame

        cv2.rectangle(frame, (x_min, y_min), (x_min + box_width,

        y_min + box_height), colour_box_current, 2)

        # Preparing text with label and confidence for current bounding box text_box_current = '{}:
        {:.4f}'.format(labels[int(class_numbers[i])],

                                        confidences[i])

        # Putting text with label and confidence on the original image cv2.putText(frame,

        text_box_current, (x_min, y_min - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,

        colour_box_current, 2)
"""



End of:

Drawing bounding boxes and labels

"""

"""

Start of:

Writing processed frame into the file

"""



# Initializing writer

# we do it only once from the very beginning # when we

get spatial dimensions of the frames if writer is None:

    # Constructing code of the codec

    # to be used in the function VideoWriter
```

```
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')

    # Writing current processed frame into the video file

    # Pay attention! If you're using Windows, yours path might looks like:

    # r'videos\result-traffic-cars.mp4' # or:

    #   'videos\\result-traffic-cars.mp4'   writer   =   cv2.VideoWriter('videos/result-

    traffic-cars.mp4', fourcc, 30,

                        (frame.shape[1], frame.shape[0]), True)

  # Write processed current frame to the file writer.write(frame)

  """


  End of:

  Writing processed frame into the file

  """

"""


End of:

Reading frames in the loop

"""

# Printing final results

print() print('Total number of frames', f) print('Total amount of

time {:.5f} seconds'.format(t)) print('FPS:', round((f / t), 1))

# Releasing video reader and writer

video.release() writer.release()
```

"""

Some comments

FOURCC is short for "four-character code" - an identifier for a video codec, compression format, colour or pixel format used in media files.

http://www.fourcc.org

Parameters for cv2.VideoWriter():

filename - Name of the output video file.

fourcc - 4-character code of codec used to compress the frames. fps - Frame rate of the created video. frameSize - Size of the video frames. isColor - If it True, the encoder will expect and encode colour frames.

3.2.21 Showing results obtained from camera in Real Time

```
# Showing current frame with detected objects
# Giving name to the window with current frame # And specifying that window is resizable
cv2.namedWindow('YOLO v3 Real Time Detections', cv2.WINDOW_NORMAL)
# Pay attention! 'cv2.imshow' takes images in BGR format cv2.imshow('YOLO v3 Real Time Detections', frame)
# Breaking the loop if 'q' is pressed if cv2.waitKey(1) & 0xFF == ord('q'):
    break
"""
End of:
Showing processed frames in OpenCV Window
"""
```

3.2.22 Releasing camera

camera.release()

Destroying all opened OpenCV windows cv2.destroyAllWindows()

"""

Some comments

cv2.VideoCapture(0)

To capture video, it is needed to create Video Capture object.

Its argument can be camera's index or name of video file.

Camera index is usually 0 for built-in one.

Try to select other cameras by passing 1, 2, 3, etc.

"""

Section-2

Objects Detection in Real Time with YOLO v3 and OpenCV

3.2.23 File: yolo-3-camera.py

"""

3.2.24 Detecting Objects in Real Time with OpenCV deep learning library

#

Algorithm:

Reading stream video from camera --> Loading YOLO v3 Network -->

3.2.25 Reading frames in the loop --> Getting blob from the frame -->

3.2.26 Implementing Forward Pass --> Getting Bounding Boxes -->

3.2.27 Non-maximum Suppression --> Drawing Bounding Boxes with Labels -->

3.2.28 Showing processed frames in OpenCV Window

#

Result:

3.2.29 Window with Detected Objects, Bounding Boxes and Labels in Real Time

3.2.30 Importing needed libraries import

numpy as np import cv2 import time

"""

Start of:

Reading stream video from camera

"""

Defining 'VideoCapture' object # and reading

stream video from camera camera =

cv2.VideoCapture(0)

Preparing variables for spatial dimensions of the frames

h, w = None, None

"""

End of:

Reading stream video from came

Start of:

3.2.31 Loading YOLO v3 network

3.2.32 Loading COCO class labels from file

Opening file

Pay attention! If you're using Windows, your path might looks like:

Yolo-coco-data\coco. Names' # Or:

```
# 'yolo-coco-data\\Coco. Names' with open ('yolo-coco-

data/Coco. Names') as f:

    # Getting labels reading every line # and

    putting them into the list labels = [line. Strip()

    for line in f]

# # Check point

# print('List with labels names:')

# print(labels)
```

3.2.33 Loading trained YOLO v3 Objects Detector

```
# with the help of 'dnn' library from OpenCV

# Pay attention! If you're using Windows, yours paths might look like:

# r'yolo-coco-data\yolov3.cfg'

# r'yolo-coco-data\yolov3.weights'

# or:

# 'yolo-coco-data\\yolov3.cfg' # 'yolo-coco-data\\yolov3.weights' network =

cv2.dnn.readNetFromDarknet('yolo-coco-data/yolov3.cfg',

                           'yolo-coco-data/yolov3.weights')



# Getting list with names of all layers from YOLO v3 network layers_names_all =

network.getLayerNames()

# # Check point

# print()
```

print(layers_names_all)

Getting only output layers' names that we need from YOLO v3 algorithm # with

function that returns indexes of layers with unconnected outputs layers_names_output

= \

[layers_names_all[i[0] - 1] for i in network.getUnconnectedOutLayers()]

Check point

print()

print(layers_names_output) # ['yolo_82', 'yolo_94', 'yolo_106'] # Setting

minimum probability to eliminate weak predictions probability_minimum =

0.5

Setting threshold for filtering weak bounding boxes

with non-maximum suppression threshold = 0.3

3.2.34 Generating colours for representing every detected object # with

function randint(low, high=None, size=None, dtype='l') colours =

np.random.randint(0, 255, size=(len(labels), 3), dtype='uint8')

Check point

print()

print(type(colours)) # <class 'numpy.ndarray'>

print(colours.shape) # (80, 3)

print(colours[0]) # [172 10 127]

3.2.35 Defining loop for catching frames while True:

Capturing frame-by-frame from camera

_, frame = camera.read()

```
# Getting spatial dimensions of the frame

# we do it only once from the very beginning # all

other frames have the same dimension if w is None or

h is None:

   # Slicing from tuple only first two elements h, w =

   frame. Shape[:2]
```

3.2.36 Getting blob from current frame

```
# The 'cv2.dnn.blobFromImage' function returns 4-dimensional blob from current

# frame after mean subtraction, normalizing, and RB channels swapping # Resulted

shape has number of frames, number of channels, width and height

# E.G.:

# blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean, swapRB=True) blob =

cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),

                      swapRB=True, crop=False)

"""

End of:

Getting blob from current frame

"""



"""

Start of:

Implementing Forward pass

"""
```

Implementing forward pass with our blob and only through output layers

Calculating at the same time, needed time for forward pass network.setInput(blob) #

setting blob as input to the network start = time.time() output_from_network =

network.forward(layers_names_output) end = time.time()

Showing spent time for single current frame print('Current frame

took {:.5f} seconds'.format(end - start))

"""

End of:

Implementing Forward pass

"""

"""

Start of:

Getting bounding boxes

"""

3.2.37 Preparing lists for detected bounding boxes, # obtained

confidences and class's number bounding_boxes = []

confidences = []

class_numbers = []

Going through all output layers after feed forward pass for result in

output_from_network:

Going through all detections from current output layer for

detected_objects in result:

ISSN : 2455-135X

```
# Getting 80 classes' probabilities for current detected object scores =

detected_objects[5:]

# Getting index of the class with the maximum value of probability

class_current = np.argmax(scores) # Getting value of probability for defined

class confidence_current = scores[class_current]

# # Check point

# # Every 'detected_objects' numpy array has first 4 numbers with

# # bounding box coordinates and rest 80 with probabilities

# # for every class

# print(detected_objects.shape) # (85,)

# Eliminating weak predictions with minimum probability if

confidence_current > probability_minimum:

        # Scaling bounding box coordinates to the initial frame size

            # YOLO data format keeps coordinates for center of bounding box

  # and its current width and height

  # That is why we can just multiply them elementwise

  # to the width and height

  # of the original frame and in this way get coordinates for center # of

  bounding box, its width and height for original frame box_current =

  detected_objects[0:4] * np.array([w, h, w, h])



  # Now, from YOLO data format, we can get top left corner coordinates
```

that are x_min and y_min x_center, y_center, box_width,

box_height = box_current x_min = int(x_center - (box_width / 2))

y_min = int(y_center - (box_height / 2)) # Adding results into

prepared lists bounding_boxes.append([x_min, y_min,

int(box_width), int(box_height)])

confidences.append(float(confidence_current)) class_numbers.append(class_current)

3.2.38 Implementing non-maximum suppression of given bounding boxes

With this technique we exclude some of bounding boxes if their

corresponding confidences are low or there is another

bounding box for this region with higher confidence

It is needed to make sure that data type of the boxes is 'int'

and data type of the confidences is 'float'

https://github.com/opencv/opencv/issues/12789 results =

cv2.dnn.NMSBoxes(bounding_boxes, confidences,

probability_minimum, threshold)

"""

End of:

Non-maximum suppression

"""

"""

Start of:

Drawing bounding boxes and labels

```
"""

# Checking if there is at least one detected object # after
non-maximum suppression if len(results) > 0:
    # Going through indexes of results
    for i in results.flatten():
        # Getting current bounding box coordinates,
        # its width and height
        x_min, y_min = bounding_boxes[i][0], bounding_boxes[i][1] box_width,
        box_height = bounding_boxes[i][2], bounding_boxes[i][3]
        # Preparing colour for current bounding box # and converting
        from numpy array to list colour_box_current =
        colours[class_numbers[i]].tolist()
        # # # Check point
        # print(type(colour_box_current)) # <class 'list'>
        # print(colour_box_current) # [172 , 10, 127]

        # Drawing bounding box on the original current frame
        cv2.rectangle(frame, (x_min, y_min), (x_min + box_width,
        y_min + box_height), colour_box_current, 2)
# Preparing text with label and confidence for current bounding box text_box_current = '{}:
                {:.4f}'.format(labels[int(class_numbers[i])],
                confidences[i])
```

```
            # Putting text with label and confidence on the original image cv2.putText(frame,

            text_box_current, (x_min, y_min - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,

            colour_box_current, 2)

    """

    End of:

    Drawing bounding boxes and labels

    """

    """

    Start of:

    Showing processed frames in OpenCV Window

    """



    # Showing results obtained from camera in Real Time



    # Showing current frame with detected objects

    # Giving name to the window with current frame # And specifying that window is resizable

    cv2.namedWindow('YOLO v3 Real Time Detections', cv2.WINDOW_NORMAL)

    # Pay attention! 'cv2.imshow' takes images in BGR format cv2.imshow('YOLO v3 Real

    Time Detections', frame)

    # Breaking the loop if 'q' is pressed if

    cv2.waitKey(1) & 0xFF == ord('q'):

        break # Releasing

camera camera.release()
```

Destroying all opened OpenCV windows

cv2.destroyAllWindows() cv2.VideoCapture(0)

To capture video, it is needed to create Video Capture object.

Its argument can be camera's index or name of video file.

Camera index is usually 0 for built-in one.

Try to select other cameras by passing 1, 2, 3, etc.

"""

3.3 Testing and verification:

Testing YOLO on manually taken images A very positive result of these test is the section of "false positives". A false positive result is an object the object detection algorithm tells it would be another object. As errors in picking are expensive this score must be low.

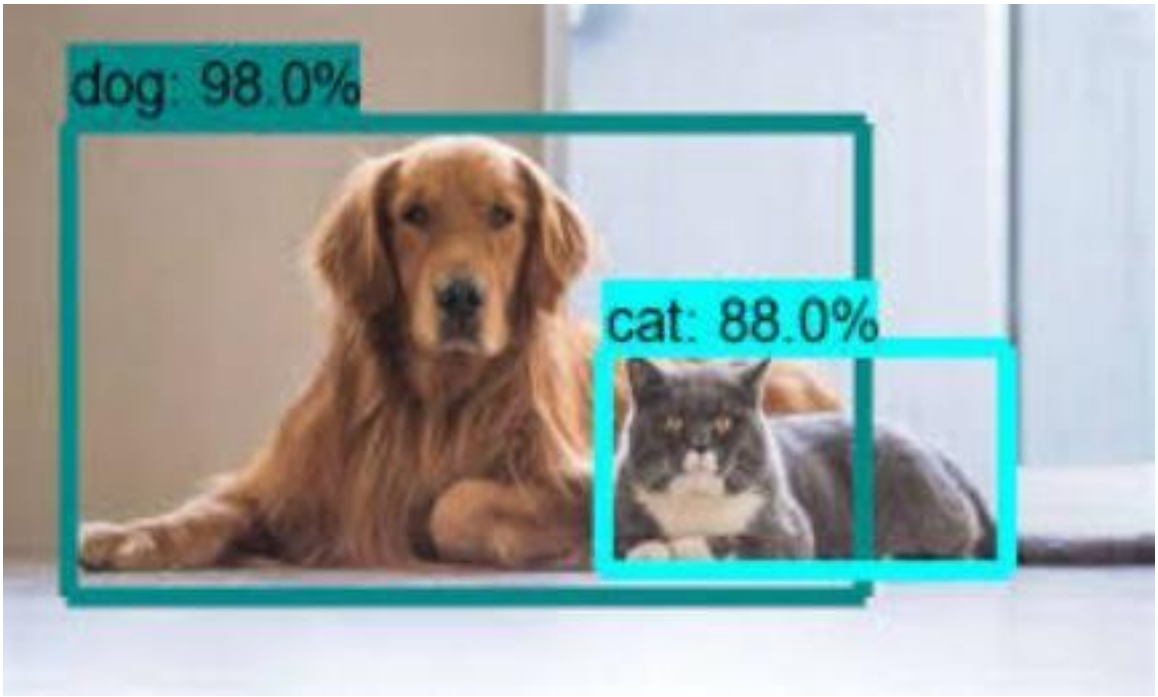

Figure 7 the output

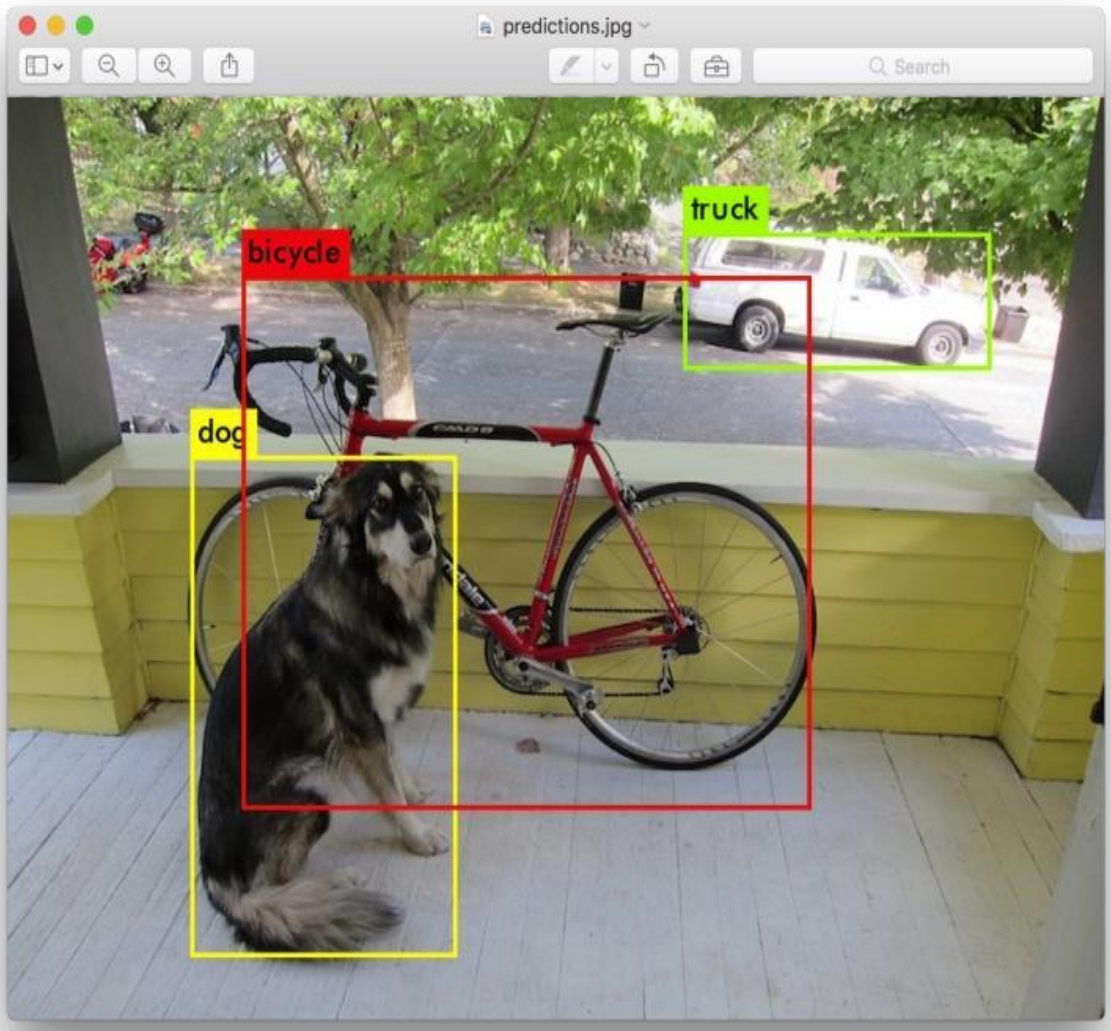

Figure 8 OUTPUT

4 Conclusion:

4.1 Summary of results:

Object detection is one of the most significant processes in machine learning and it fundamentally found its way in solving too many problems which related to computer vison problems in our project, we use one of the most popular machine learning algorithms in object detection yolo algorithm and we find out that we can use it in an efficient way to detect the objects.

4.2 Advantage of the work:

Object detection is fully interconnected with other related computer vision methods like image segmentation and image recognition that help us to perceive and comprehend the objects in the videos and photos.

4.3 Scope of the future work:

Object detection This is a computer vision method that enables us to recognize and find objects in an image or a video. Through such identification and localization, object detection can be applied to count the number

of objects on a scene and calculate and trace their exact location of objects, all the time giving them their accurate names.

4.4 Unique feature of our project:

It has applications in computer vision problems like image annotation, vehicle counting, activity recognition, face detection, face recognition, video object co-segmentation.

References:

1. Wang, X. (2016). Deep learning in object recognition, detection, and segmentation. *Foundations and Trends& in Signal Processing*, *8*(4), 217-382.
2. Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, *30*(11), 3212-3232.
3. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).
4. Sanchez, S. A., Romero, H. J., & Morales, A. D. (2020, May). A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework. In *IOP conference series: materials science and engineering* (Vol. 844, No. 1, p. 012024). IOP Publishing.
5. Yan, J., Yu, Y., Zhu, X., Lei, Z., & Li, S. Z. (2015). Object detection by labeling superpixels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5107-5116).
6. Schneiderman, H., & Kanade, T. (2004). Object detection using the statistics of parts. *International Journal of Computer Vision*, *56*(3), 151-177.
7. Dou, F., Lu, J., Xu, T., Huang, C. H., & Bi, J. (2020). A bisection reinforcement learning approach to 3-D indoor localization. *IEEE Internet of Things Journal*, *8*(8), 6519-6535.
8. Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep learning* (Vol. 1, pp. 23-24). Cambridge, MA, USA: MIT press.
9. S. Maji and J. Malik, "Object detection using a max-margin hough transform," *2009 IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2009*, pp. 1038–1045, 2009, doi: 10.1109/CVPRW.2009.5206693.
10. Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212– 3232, 2019, doi: 10.1109/TNNLS.2018.2876865.
11. R. Pierdicca, M. Paolanti, A. Felicetti, F. Piccinini, and P. Zingaretti, "Automatic faults detection of photovoltaic farms: Solair, a deep learning-based system for thermal images," *Energies*, vol. 13, no. 24, pp. 1–17, 2020, doi: 10.3390/en13246496.
12. D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2155–2162, 2014, doi: 10.1109/CVPR.2014.276.
13. O. Alsing, "Mobile Object Detection using TensorFlow Lite and Transfer Learning TT - Objektigenkänning i mobila enheter med TensorFlow Lite (swe)," *TritaEecs-Ex Nv - 2018535*, vol. Independen, 2018, [Online]. Available: http://kth.divaportal.org/smash/get/diva2:1242627/FULLTEXT01.pdf%0Ahttp://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-233775.

14. R. L. Galvez, A. A. Bandala, E. P. Dadios, R. R. P. Vicerra, and J. M. Z. Maningo, "Object Detection Using Convolutional Neural Networks," *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, vol. 2018-Octob, no. October, pp. 2023–2027, 2019, doi: 10.1109/TENCON.2018.8650517.
15. H. Schneiderman and T. Kanade, "Object detection using the statistics of parts,"
16. *Int. J. Comput. Vis.*, vol. 56, no. 3, pp. 151–177, 2004, doi: 10.1023/B:VISI.0000011202.85607.00.
17. P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," Cvpr 2001, vol. 1, pp. I-511-I–
18. R. B. Girshick, P. F. Felzenszwalb, and D. McAllester. Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/ rbg/latentrelease5/. [10] C. Gu, J. J. Lim, P. Arbelaez, and J. Malik. Recognition ´ using regions. In CVPR, 2009.
19. A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1106–1114, 2012.
20. C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In CVPR, 2008.
21. P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229, 2013.
22. H. O. Song, S. Zickler, T. Althoff, R. Girshick, M. Fritz, C. Geyer, P. Felzenszwalb, and T. Darrell. Sparselet models for efficient multiclass object detection. In ECCV. 2012. [15] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In Advances in Neural Information Processing Systems (NIPS), 2013
23. C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In Advances in Neural Information Processing Systems (NIPS), 2013.
24. J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. International journal of computer vision, 104(2):154–171, 2013.K. E. van de Sande, J. R. Uijlings, T. Gevers, and A. W. Smeulders. Segmentation as selective search for object recognition. In ICCV, 2011.