A Project Report
on

OBJECT DETECTION AND LOCALIZATION

submitted in partial fulfillment of the requirements for the award of the degree
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

by

17WH1A0564	**Ms. G.VAISHNAVI**
17WH1A0580	**Ms. NAKHVEEN AHMED**
17WH1A0583	**Ms. B.ANJANI**

under the esteemed guidance of
Ms. Suparna Das
Assistant Professor

Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

May, 2021

DECLARATION

We hereby declare that the work presented in this project entitled **"OBJECT DETECTION AND LOCALIZATION"** submitted towards completion of Project Work in IV year of B.Tech., CSE at 'BVRIT HYDERABAD College of Engineering For Women', Hyderabad is an authentic record of our original work carried out under the guidance of Ms. Suparna Das, Assistant Professor, Department of CSE.

Sign. with date:
Ms. G.VAISHNAVI
(17WH1A0564)

Sign. with date:
Ms. NAKHVEEN AHMED
(17WH1A0580)

Sign. with date:
Ms. B.ANJANI
(17WH1A0583)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering

Certificate

This is to certify that the Project Work report on "**OBJECT DETECTION AND LOCALIZATION**" is a bonafide work carried out by Ms. G.VAISHNAVI(17WH1A0564) ; Ms. NAKHVEEN AHMED (17WH1A0580) ; Ms. B.ANJANI (17WH1A0583) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department
Dr. K.Srinivasa Reddy
Professor and HoD,
Department of CSE

Guide
Ms.Suparna Das
Assistant Professor

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal**, **BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. K.Srinivasa Reddy, Professor**, Department of CSE, **BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Ms. Suparna Das, Assistant Professor**, Department of CSE**, BVRIT HYDERABAD College of Engineering for Women** for his constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. G.VAISHNAVI
(17WH1A0564)

Ms. NAKHVEEN AHMED
(17WH1A0580)

Ms. B.ANJANI
(17WH1A0583)

Contents

ABSTRACT

Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the map of other real-time detectors. An innovative approach for detecting and localizing duplicate objects in pick-and-place applications under extreme conditions of occlusion, where standard appearance-based approaches are likely to be ineffective. The approach exploits SIFT key point extraction and mean shift clustering to partition the correspondences between the object model and the image onto different potential object instances with real-time performance. Then, the hypotheses of the object shape are validated by a projection with a fast Euclidean transform of some delimiting points onto the current image.

LIST OF FIGURES

1. INTRODUCTION

Object detection and localization model deals with images of different objects and uses Machine learning to identify objects in the given image and give audio of that image. Machine learning is a use of computer systems to automatically learn, improve and implement the experience without following explicit instructions. The techniques in this model are used to identify the object in the image, localize it with a bounding box and also produce an audio with object names detected.

1.1 Objectives

Object detection is very common for a human being. Every person does it while seeing something. The main objective here is to make the system detect the objects present in an image which reduces human effort in many cases. Object detection and localization model allows a user to give an image and get the image with labels and bounding boxes of the respective objects and also an audio stating the objects, which can be customized. YOLO(You only look once) is a

1.2 Methodology

Object localization refers to identifying the location of one or more objects in an image and drawing a bounding box around their extent. Object detection combines these two tasks and localizes and classifies one or more objects in an image.In this section the methodology followed is discussed in detail

1.2.1 Dataset

In this section, we present the details on the collection, annotation, statistics, and quality of the dataset respectively.

Data Collection

- Data Source To make the image sources more diverse, we collect images mainly from Flicker 1 . All the images conform to licensing for research purposes. Sample images can be found on our website2 .
- Object Categories Based on the collected images, we first select eleven supercategories which are common and diverse to cover most object instances. They are: human and related accessories, living room, clothes, kitchen, instrument, transportation, bathroom, electronics, food (vegetables), office supplies, and animals. Based on the super-categories, we further propose 442 categories which widely exists in our daily lives. As some of the object

categories are rarely found, we first annotate all 442 categories in the first 100K images and then select the most frequent 365 object categories as our target objects. Also, to be compatible with the existing object detection benchmarks, the 365 categories include the categories defined in PASCAL VOC [8] and COCO [24] benchmarks.

- Non-Iconic Images As our Objects365 dataset focuses on object detection, we eliminate those images which are only suitable for image classification. For example, the image only contains one object instance around the image center. This filtering process was first adopted in COCO [24].

Annotation

As there are a large number of images and object categories, a good annotation process is of great importance to ensure high quality and efficiency.

Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image. More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

1.2.2 Unified Detection

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously.

This means our network reasons globally about the full image and all the objects in the image. Our system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.

At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Classi}|\text{Object}) * \Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(\text{Classi}) * \text{IOU}^{\text{truth}}_{\text{pred}} \quad (1)$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

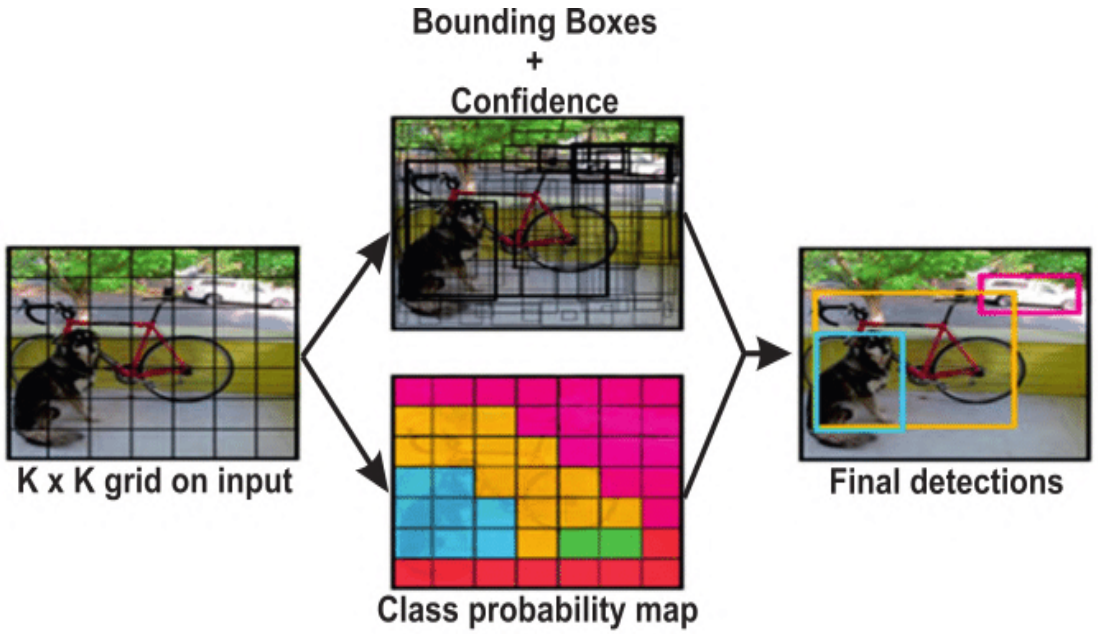


Fig 1.2.1: The Model

Our system models detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B ∗ 5 + C) tensor.

For evaluating YOLO on PASCAL VOC, we use S = 7, B = 2. PASCAL VOC has 20 labelled classes so C = 20. Our final prediction is a 7 × 7 × 30 tensor.

Network Design

We implement this model as a convolutional neural network and evaluate it on the PASCAL VOC detection dataset. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.

Our network architecture is inspired by the Google Net model for image classification. Our network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by Google Net, we simply use 1 × 1 reduction layers followed by 3 × 3 convolutional layers.

We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO.

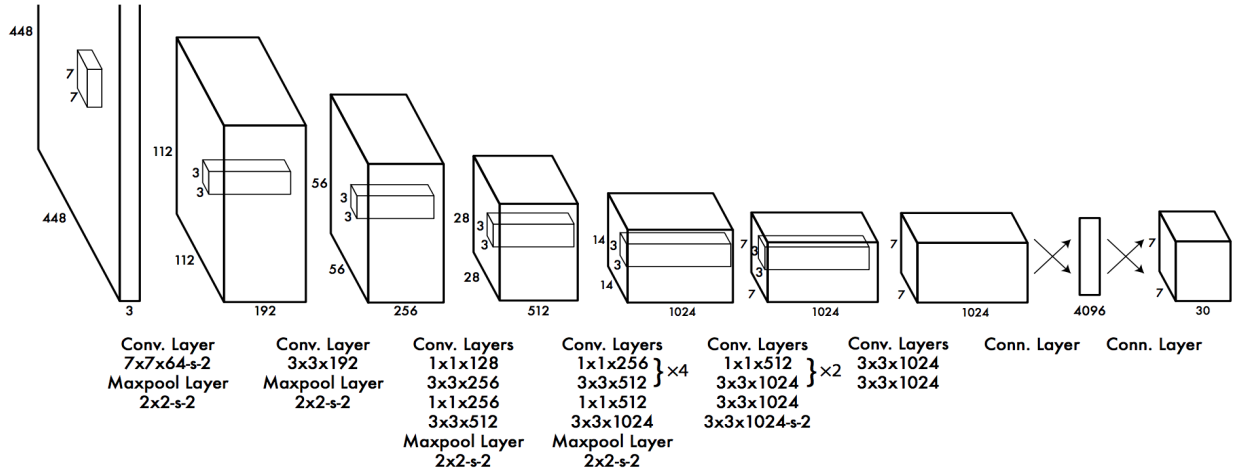


Fig 1.2.2: The Architecture

The final output of our network is the 7 × 7 × 30 tensor of predictions.

Training

We pre train our convolutional layers on the ImageNet 1000-class competition dataset [30]. For pre-training we use the first 20 convolutional layers from Figure 3 followed by an average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the Google Net models in Caffe's Model Zoo.We use the Dark net framework for all training and inference.

We add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from 224 × 224 to 448 × 448. Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1.

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be “responsible” for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or classes of object, improving overall recall.

INFERENCE

Just like in training, predicting detections for a test image only requires one network evaluation. On PASCAL VOC the network predicts 98 bounding boxes per image and class probabilities for each box. YOLO is extremely fast at test time since it only requires a single network evaluation, unlike classifier-based methods. The grid design enforces spatial diversity in the bounding box predictions. Often it is clear which grid cell an object falls into and the network only predicts one box for each object. However, some large objects or objects near the border of multiple cells can be well localized by multiple cells. Non-maximal suppression can be used to fix these multiple detections. While not critical to performance as it is for R-CNN or DPM, non-maximal suppression adds 2- 3% in the map.

Limitations

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image.

Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a

small box has a much greater effect on IOU. Our main source of error is incorrect localizations.

Comparison to Other Detection Systems

Object detection is a core problem in computer vision. Detection pipelines generally start by extracting a set of robust features from input images. Then, classifiers or localizers are used to identify objects in the feature space. These classifiers or localizers are run either in sliding window fashion over the whole image or on some subset of regions in the image. We compare the YOLO detection system to several top detection frameworks, highlighting key similarities and differences.

YOLO is originally trained with a COCO dataset with 9000 images containing 80 different objects.

- **Deformable parts models.** Deformable parts models (DPM) use a sliding window approach to object detection. DPM uses a disjoint pipeline to extract static features, classify regions, predict bounding boxes for high scoring regions, etc.

- **R-CNN.** R-CNN and its variants use region proposals instead of sliding windows to find objects in images. Selective Search generates potential bounding boxes, a convolutional network extracts features, an SVM scores the boxes, a linear model adjusts the bounding boxes, and non-max suppression eliminates duplicate detections. Each stage of this complex pipeline must be precisely tuned independently and the resulting system is very slow, taking more than 40 seconds per image at test time.

- **Other Fast Detectors** Fast and Faster R-CNN focus on speeding up the R-CNN framework by sharing computation and using neural networks to propose regions instead of Selective Search. While they offer speed and accuracy improvements over R-CNN, both still fall short of real-time performance.

- **Deep Multi Box.** Unlike R-CNN, train a convolutional neural network to predict regions of interest [8] instead of using Selective Search. Multi Box can

also perform single object detection by replacing the confidence prediction with a single class prediction. However, Multi Box cannot perform general object detection and is still just a piece in a larger detection pipeline, requiring further image patch classification. Both YOLO and Multi Box use a convolutional network to predict bounding boxes in an image but YOLO is a complete detection system.

· **Over Feat.** Train a convolutional neural network to perform localization and adapt that localizer to perform detection. Over Feat efficiently performs sliding window detection but it is still a disjoint system.

Experiments

First we compare YOLO with other real-time detection systems on PASCAL VOC 2007. To understand the differences between YOLO and R-CNN variants we explore the errors on VOC 2007 made by YOLO and Fast R-CNN, one of the highest performing versions of R-CNN. Based on the different error profiles we show that YOLO can be used to rescore Fast R-CNN detections and reduce the errors from background false positives, giving a significant performance boost. We also present VOC 2012 results and compare maps to current state-of-the-art methods. Finally, we show that YOLO generalizes to new domains better than other detectors on two artwork datasets.

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Operating system : Windows 10, Ubuntu

Web browser : Google chrome

2.1.2 Hardware Requirements

Processor : Intel Core i5 @ 1.60 GHz

Memory : 8 GB (RAM)

Hard disk : 240 GB

2.2 Technologies Description

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious

repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform Tensor flow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

gtts(Google Text-to-Speech):

gTTS (Google Text-to-Speech), a Python library and CLI tool to interface with Google Translate's text-to-speech API. Writes spoken mp3 data to a file, a file-like object (bytestring) for further audio manipulation, or stdout. It features flexible pre-processing and tokenizing.

We will import the gTTS library from the gtts module which can be used for speech translation.

A variable is used to perform the Google text-to-speech translation on the user's input. The output of the converted text is stored in the form of speech in that variable.

The save function allows us to save the converted speech in a format that allows us to play sounds. Speech can be stored in a .mp3 format. Other formats like .wav format can also be used.

IPython.display.Audio

IPython.display.Audio lets you play audio directly in an IPython notebook.

The speech file that was generated by gtts library is played using the Audio function class in this model.

There is also an attribute where we can specify if we want it to autoplay.

Jupyter Notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

3. DESIGN

3.1 Introduction

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Once system requirements have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed, reviewed and documented. System design can be viewed from either a technical or project management perspective. From the technical

point of view, design consists of four activities – architectural design, data structure design, interface design and procedural design.

3.2 UML Diagrams

3.2.1 Use Case Diagram

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

Only static behavior is not sufficient to model a system; rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagrams are one of them. Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consist of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

Hence to model the entire system, a number of use case diagrams are used.

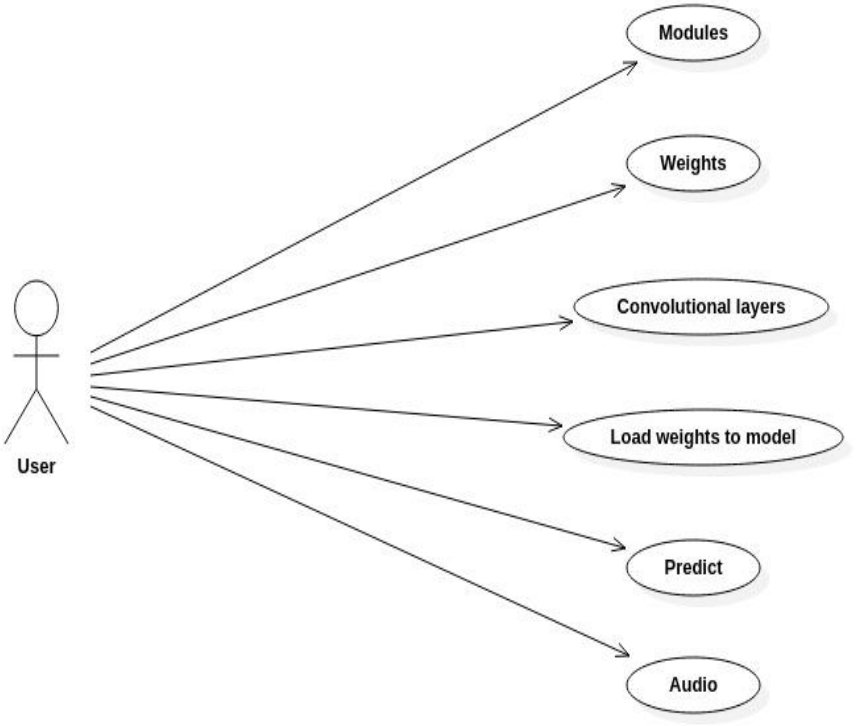


Fig3.2.1: Use Case Diagram

3.2.2 Sequence Diagram

Sequence Diagrams Represent the objects participating in the interaction horizontally and time vertically. A Use Case is a kind of behavioral classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling.

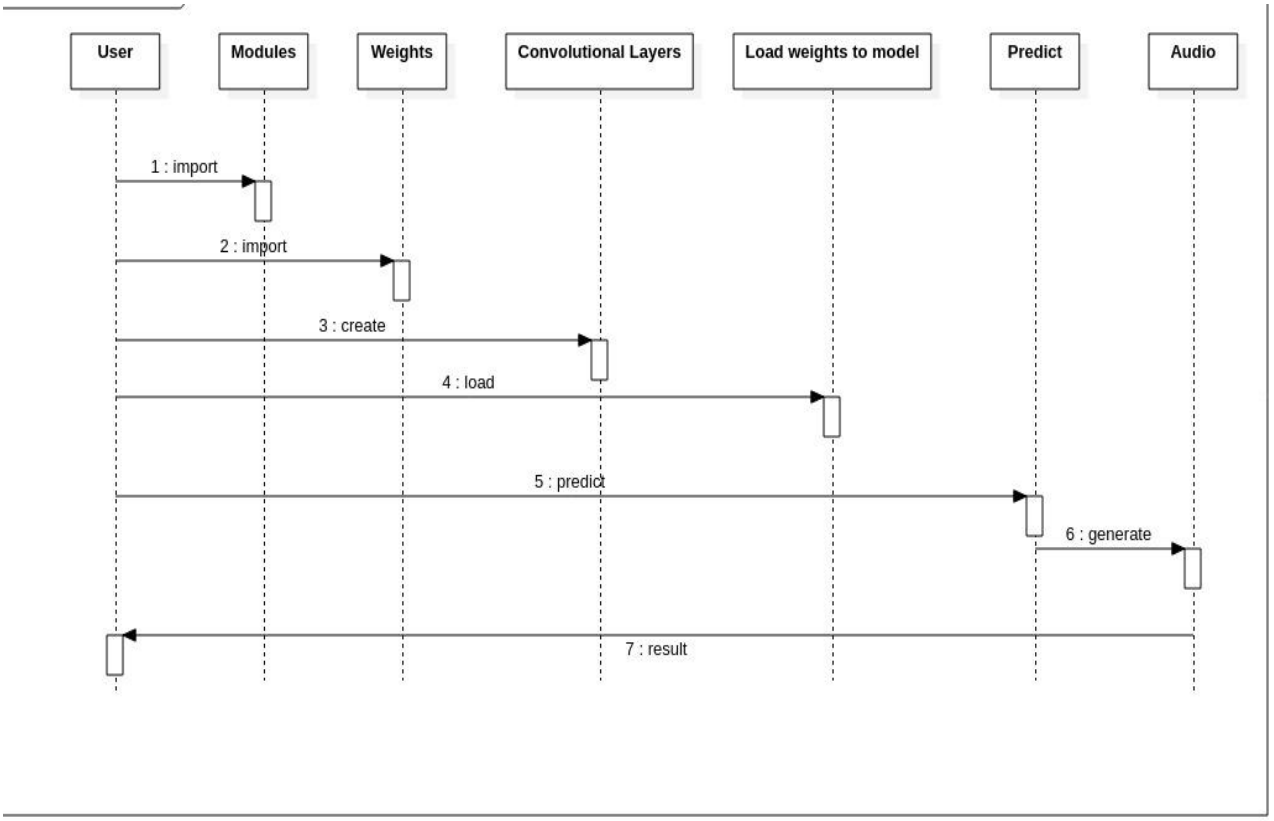


Fig 3.2.2: Sequence Diagram

3.2.3 Activity Diagram

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

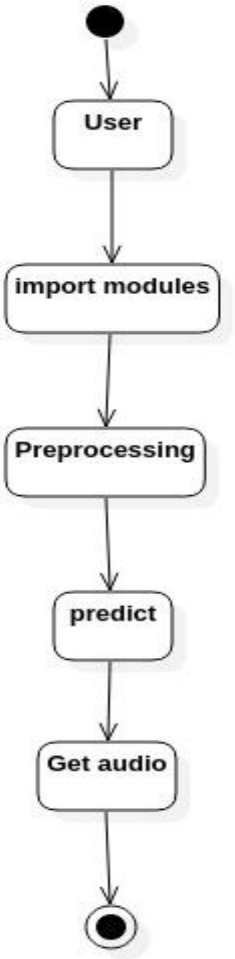


Fig 3.2.3: Activity Diagram

3.2.4 Collaboration Diagram

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing.

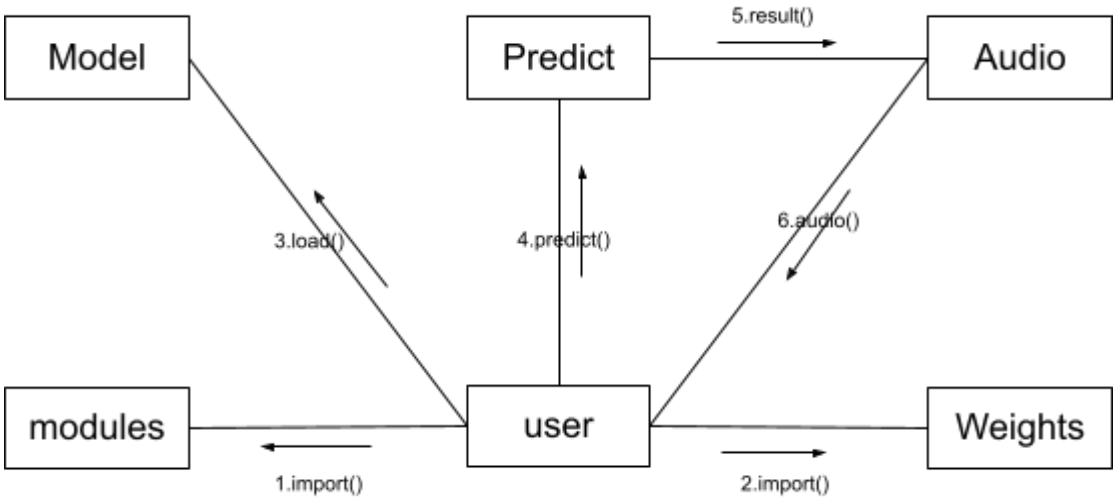


Fig 3.2.4:Collaboration Diagram

3.2.5 Class Diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the system of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

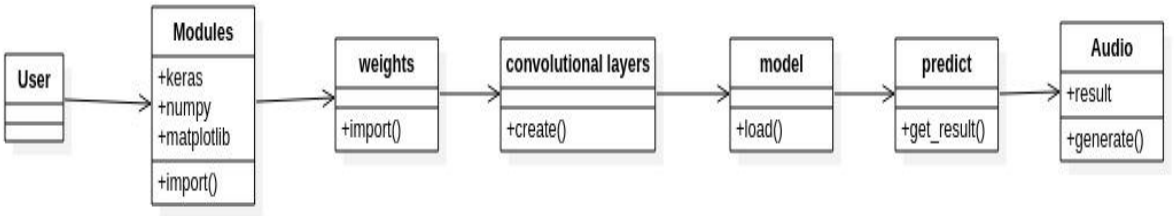


Fig 3.2.5: Class Diagram

4. IMPLEMENTATION

4.1 Coding

```
#First Let us import all the libraries that are required
import struct
import numpy as np
from numpy import expand_dims
from keras.layers import Conv2D
from keras.layers import Input
from keras.layers import BatchNormalization
from keras.layers import LeakyReLU
from keras.layers import ZeroPadding2D
from keras.layers import UpSampling2D
from keras.layers.merge import add, concatenate
from keras.models import Model
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from matplotlib import pyplot
from matplotlib.patches import Rectangle


  #This function is used to create convolutional layers
def _conv_block(inp, convs, skip=True):
 x = inp
 count = 0
 for conv in convs:
  if count == (len(convs) - 2) and skip:
   skip_connection = x
  count += 1
    if conv['stride'] > 1: x = ZeroPadding2D(((1,0),(1,0)))(x) # peculiar padding as
darknet prefer left and top
  x = Conv2D(conv['filter'],
        conv['kernel'],
        strides=conv['stride'],
          padding='valid' if conv['stride'] > 1 else 'same', # peculiar padding as darknet
prefer left and top
        name='conv_' + str(conv['layer_idx']),
        use_bias=False if conv['bnorm'] else True)(x)
```

```
        if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' + str(conv['layer_idx']))(x)
                if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' + str(conv['layer_idx']))(x)
  return add([skip_connection, x]) if skip else x
#'''This function define the Keras model for YOLOv3.'''
def make_yolov3_model():
  input_image = Input(shape=(None, None, 3))
  # Layer  0 => 4
   x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0},
                 {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
                 {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
                 {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])
  # Layer  5 => 8
   x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
          {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
          {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])
  # Layer  9 => 11
   x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                 {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])
  # Layer 12 => 15
   x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
          {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
              {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])
  # Layer 16 => 36
  for i in range(7):
     x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 16+i*3},
```

```
                {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
17+i*3}])
  skip_36 = x
  # Layer 37 => 40
   x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 37},
           {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 38},
                {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
39}])
  # Layer 41 => 61
  for i in range(7):
     x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 41+i*3},
                {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
42+i*3}])
  skip_61 = x
  # Layer 62 => 65
   x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 62},
              {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
63},
             {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
64}])
  # Layer 66 => 74
  for i in range(3):
     x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 66+i*3},
                {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
67+i*3}])
  # Layer 75 => 79
   x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 75},
             {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
76},
              {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
77},
```

```
            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
78},
             {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
79}], skip=False)
 # Layer 80 => 82
   yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky': True,  'layer_idx': 80},
             {'filter':  255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx':
81}], skip=False)
 # Layer 83 => 86
  x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 84}], skip=False)
 x = UpSampling2D(2)(x)
 x = concatenate([x, skip_61])
 # Layer 87 => 91
  x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 87},
         {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 88},
         {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 89},
         {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 90},
             {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
91}], skip=False)
 # Layer 92 => 94
  yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky':
True,  'layer_idx': 92},
             {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx':
93}], skip=False)
 # Layer 95 => 98
  x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 96}], skip=False)
 x = UpSampling2D(2)(x)
 x = concatenate([x, skip_36])
 # Layer 99 => 106
   yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,
'leaky': True,  'layer_idx': 99},
             {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx':
100},
```

```
                {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx':
101},
                {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx':
102},
                {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx':
103},
                {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx':
104},
                {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx':
105}], skip=False)
 model = Model(input_image, [yolo_82, yolo_94, yolo_106])
 return model

#This class reads and weights from file and rather manually
#decoding the weights from yolo weights file than we can use it.
class WeightReader:
 def __init__(self, weight_file):
   with open(weight_file, 'rb') as w_f:
     major,  = struct.unpack('i', w_f.read(4))
     minor,  = struct.unpack('i', w_f.read(4))
     revision, = struct.unpack('i', w_f.read(4))
     if (major*10 + minor) >= 2 and major < 1000 and minor < 1000:
       w_f.read(8)
     else:
       w_f.read(4)
     transpose = (major > 1000) or (minor > 1000)
     binary = w_f.read()
   self.offset = 0
   self.all_weights = np.frombuffer(binary, dtype='float32')

 def read_bytes(self, size):
   self.offset = self.offset + size
   return self.all_weights[self.offset-size:self.offset]

 def load_weights(self, model):
   for i in range(106):
     try:
```

```
        conv_layer = model.get_layer('conv_' + str(i))
        print("loading weights of convolution #" + str(i))
        if i not in [81, 93, 105]:
          norm_layer = model.get_layer('bnorm_' + str(i))
          size = np.prod(norm_layer.get_weights()[0].shape)
          beta  = self.read_bytes(size) # bias
          gamma = self.read_bytes(size) # scale
          mean  = self.read_bytes(size) # mean
          var   = self.read_bytes(size) # variance
          weights = norm_layer.set_weights([gamma, beta, mean, var])
        if len(conv_layer.get_weights()) > 1:
          bias   = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape))
          kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
          kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
          kernel = kernel.transpose([2,3,1,0])
          conv_layer.set_weights([kernel, bias])
        else:
          kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
          kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
          kernel = kernel.transpose([2,3,1,0])
          conv_layer.set_weights([kernel])
      except ValueError:
        print("no convolution #" + str(i))

  def reset(self):
    self.offset = 0


   class BoundBox:
  def __init__(self, xmin, ymin, xmax, ymax, objness = None, classes = None):
    self.xmin = xmin
    self.ymin = ymin
    self.xmax = xmax
    self.ymax = ymax
    self.objness = objness
    self.classes = classes
    self.label = -1
    self.score = -1
```

```
  def get_label(self):
    if self.label == -1:
      self.label = np.argmax(self.classes)

    return self.label

  def get_score(self):
    if self.score == -1:
      self.score = self.classes[self.get_label()]

    return self.score

def _sigmoid(x):
  return 1. / (1. + np.exp(-x))

def decode_netout(netout, anchors, obj_thresh, net_h, net_w):
  grid_h, grid_w = netout.shape[:2]
  nb_box = 3
  netout = netout.reshape((grid_h, grid_w, nb_box, -1))
  nb_class = netout.shape[-1] - 5
  boxes = []
  netout[..., :2]  = _sigmoid(netout[..., :2])
  netout[..., 4:]  = _sigmoid(netout[..., 4:])
  netout[..., 5:]  = netout[..., 4][..., np.newaxis] * netout[..., 5:]
  netout[..., 5:] *= netout[..., 5:] > obj_thresh

  for i in range(grid_h*grid_w):
    row = i / grid_w
    col = i % grid_w
    for b in range(nb_box):
      # 4th element is objectness score
      objectness = netout[int(row)][int(col)][b][4]
      if(objectness.all() <= obj_thresh): continue
      # first 4 elements are x, y, w, and h
      x, y, w, h = netout[int(row)][int(col)][b][:4]
      x = (col + x) / grid_w # center position, unit: image width
```

```
      y = (row + y) / grid_h # center position, unit: image height
      w = anchors[2 * b + 0] * np.exp(w) / net_w # unit: image width
      h = anchors[2 * b + 1] * np.exp(h) / net_h # unit: image height
      # last elements are class probabilities
      classes = netout[int(row)][col][b][5:]
      box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
      boxes.append(box)
  return boxes

#The correct_yolo_boxes() function to perform the translation of bounding box
#coordinates, taking the list of bounding boxes, the original shape of our
#loaded photograph, and the shape of the input to the network as arguments.
#The coordinates of the bounding boxes are updated directly.

def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
  new_w, new_h = net_w, net_h
  for i in range(len(boxes)):
    x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w
    y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h
    boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
    boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
    boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
    boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)

def _interval_overlap(interval_a, interval_b):
  x1, x2 = interval_a
  x3, x4 = interval_b
  if x3 < x1:
    if x4 < x1:
      return 0
    else:
      return min(x2,x4) - x1
  else:
    if x2 < x3:
       return 0
    else:
      return min(x2,x4) - x3
```

```
def bbox_iou(box1, box2):
  intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
  intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])
  intersect = intersect_w * intersect_h
  w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin
  w2, h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin
  union = w1*h1 + w2*h2 - intersect
  return float(intersect) / union



    def do_nms(boxes, nms_thresh):
  if len(boxes) > 0:
   nb_class = len(boxes[0].classes)
  else:
   return
  for c in range(nb_class):
   sorted_indices = np.argsort([-box.classes[c] for box in boxes])
   for i in range(len(sorted_indices)):
    index_i = sorted_indices[i]
    if boxes[index_i].classes[c] == 0: continue
    for j in range(i+1, len(sorted_indices)):
     index_j = sorted_indices[j]
     if bbox_iou(boxes[index_i], boxes[index_j]) >=nms_thresh:
      boxes[index_j].classes[c] = 0


    # get all of the results above a threshold
def get_boxes(boxes, labels, thresh):
  v_boxes, v_labels, v_scores = list(), list(), list()
  # enumerate all boxes
  for box in boxes:
   # enumerate all possible labels
   for i in range(len(labels)):
    # check if the threshold for this label is high enough
    if box.classes[i] > thresh:
```

```
            v_boxes.append(box)
            v_labels.append(labels[i])
            v_scores.append(box.classes[i]*100)
            # don't break, many labels may trigger for one box
    return v_boxes, v_labels, v_scores

# draw all results
def draw_boxes(filename, v_boxes, v_labels, v_scores):
    pyplot.figure(figsize=(20,10))
    # load the image
    data = pyplot.imread(filename)
    # plot the image
    pyplot.imshow(data)
    # get the context for drawing boxes
    ax = pyplot.gca()
    # plot each box
    for i in range(len(v_boxes)):
        box = v_boxes[i]
        # get coordinates
        y1, x1, y2, x2 = box.ymin, box.xmin, box.ymax, box.xmax
        # calculate width and height of the box
        width, height = x2 - x1, y2 - y1
        # create the shape
        rect = Rectangle((x1, y1), width, height, fill=False, color='blue',lw=3)
        # draw the box
        ax.add_patch(rect)
        # draw text and score in top left corner
        label = "%s (%.3f)" % (v_labels[i], v_scores[i])
        pyplot.text(x1-20, y1-20, label, color='red',fontsize=16)
    # show the plot
    pyplot.show()


    # load and prepare an image
def load_image_pixels(filename, shape):
        # load the image to get its shape
        image = load_img(filename)
```

```
    width, height = image.size
    # load the image with the required size
    image = load_img(filename, target_size=shape)
    # convert to numpy array
    image = img_to_array(image)
    # scale pixel values to [0, 1]
    image = image.astype('float32')
    image /= 255.0
    # add a dimension so that we have one sample
    image = expand_dims(image, 0)
    return image, width, height

        # define the model
    model = make_yolov3_model()

   #read weights from yolov3 weights file provided in the data

weight_reader=WeightReader('/content/gdrive/MyDrive/majorproject/yolov3.weights'
)




# set the model weights into the model
weight_reader.load_weights(model)

# define the expected input shape for the model
input_w, input_h = 416, 416
# define our new photo
photo_filename = '/content/drive/MyDrive/majorproject/office.jpg'
# load and prepare image
image, image_w, image_h = load_image_pixels(photo_filename, (input_w, input_h))
# make prediction
yhat = model.predict(image)
# summarize the shape of the list of arrays
print([a.shape for a in yhat])
# define the anchors
```

```
anchors = [[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]]
# define the probability threshold for detected objects
class_threshold = 0.7
boxes = list()
for i in range(len(yhat)):
  # decode the output of the network
  boxes += decode_netout(yhat[i][0], anchors[i], class_threshold, input_h, input_w)
# correct the sizes of the bounding boxes for the shape of the image
correct_yolo_boxes(boxes, image_h, image_w, input_h, input_w)
# suppress non-maximal boxes
do_nms(boxes, 0.4)
# define the labels
labels = []
with open("/content/drive/MyDrive/majorproject/coco.names", "r") as f:
    labels = [line.strip() for line in f.readlines()]
v_boxes, v_labels, v_scores = get_boxes(boxes, labels, class_threshold)
# summarize what we found
for i in range(len(v_boxes)):
  print(v_labels[i], v_scores[i])
# draw what we found
draw_boxes(photo_filename, v_boxes, v_labels, v_scores)

cprint(v_labels)

!pip install gTTS

!pip install playsound

!pip install pyttsx3

import gtts
from IPython.display import Audio

print(v_labels)
res = "This image contains " + ' '.join([i for i in v_labels]) + '.'
print(res)
```

```
audio = gtts.gTTS(res)
audio.save('file.mp3')
Audio('file.mp3', autoplay=True)
```

OUTPUT

The output of the model used here is an image passed with labels of the objects in the image and a bounding box for the objects labelled. The next part would be an audio of the objects labeled in the image passed to the model.

The model can label upto 80 different types of objects. They are listed below.

Person
bicycle
car
motorbike
aeroplane
bus
train
truck
boat
traffic light
fire hydrant
stop sign
parking meter
bench
bird
cat
dog
horse
sheep

cow

elephant

bear

zebra

giraffe

backpack

umbrella

handbag

tie

suitcase

frisbee

skis

snowboard

sports ball

kite

baseball bat

baseball glove

skateboard

surfboard

tennis racket

bottle

wine glass

cup

fork

knife

spoon

bowl

banana

apple

sandwich

orange

broccoli

carrot

hot dog

pizza

donut

cake

chair

sofa

potted plant

bed

dining table

toilet

tv monitor

laptop

mouse

remote

keyboard

cell phone

microwave

oven

toaster

sink

refrigerator

book

clock

vase

scissors

teddy bear

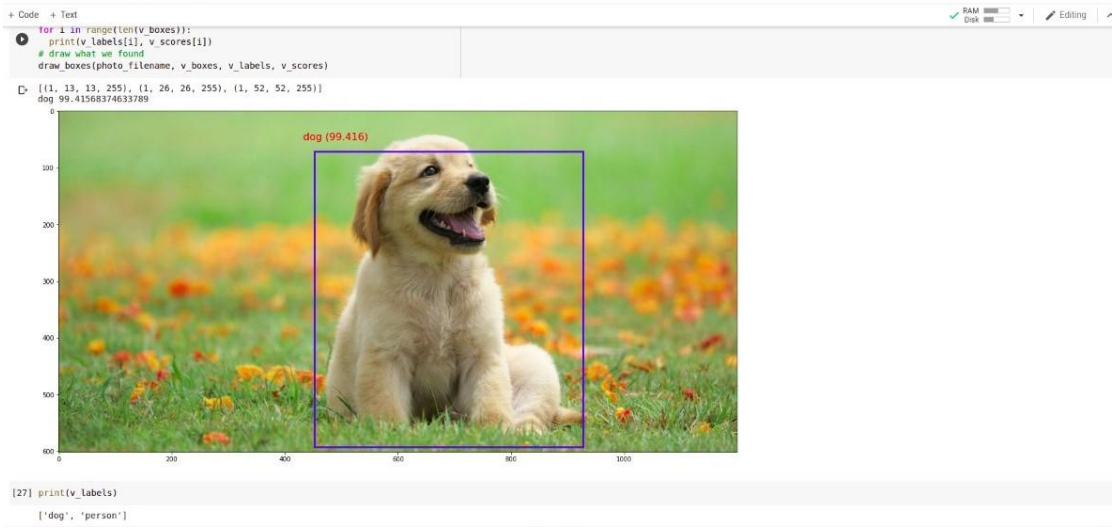
hair drier

toothbrush

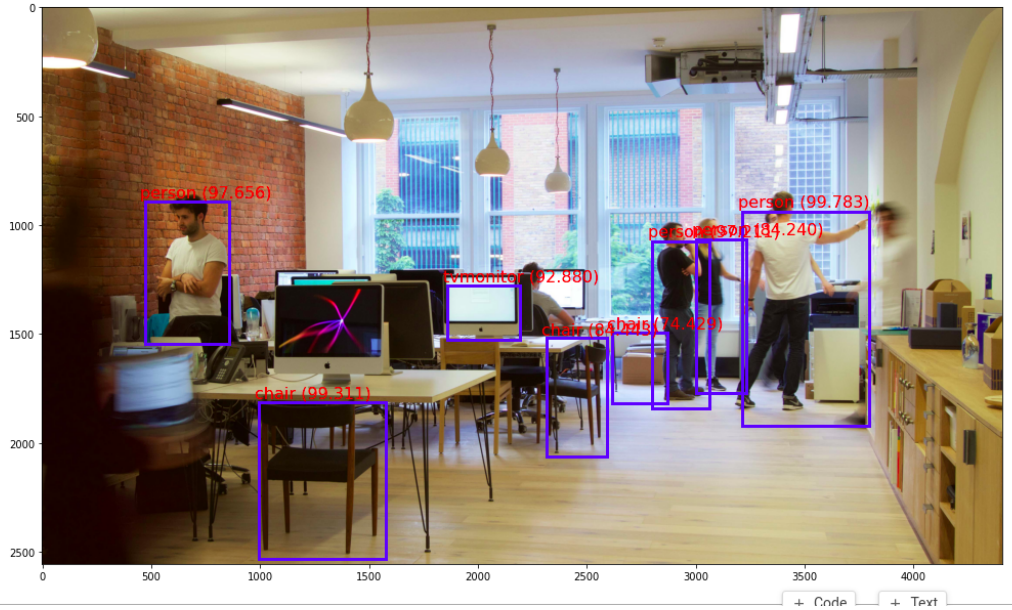
The screenshot given below is the output image obtained by passing an image of a dog. The output is the same image with a label of dog, and accuracy or a weight of 99.416 and a bounding box of colour blue around it.

The next screenshot shows the generation of audio files using gtts library.

Images with multiple objects can also be given



```
[(1, 13, 13, 255), (1, 26, 26, 255), (1, 52, 52, 255)]
person 97.65632748603821
tvmonitor 92.88041591644287
person 84.24032330513
person 99.7834324836731
person 97.210693359375
chair 74.42907094955444
chair 84.44294929504395
chair 99.31058883666992
```



5. CONCLUSION AND FUTURE SCOPE

The model helps to identify the objects in any image and saves so much time in finding out manually.

Adding audio feature also helps people to hear what they see or people who can't see can hear what is in there.

There are many future enhancements that can be done with this model.

- An own dataset can be prepared with the images of some selected objects which are not identified by the yolo model and the model can be trained specifically on those images. This will be helpful when the model is used in situations like identifying parts in an IC or in a movie scene.
- This model can be added with an external camera and can be made into an IOT project which can help blind people or people with poor eyesight to dynamically take a photo of something and can hear what is in the image.

6. REFERENCES

[1] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In Computer Vision– ECCV 2008, pages 2–15. Springer, 2008.

[2] S. Gidaris and N. Komodakis. Object detection via a multiregion & semantic segmentation-aware CNN model. CoRR, abs/1505.01749, 2015.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 580–587. IEEE, 2014.

[4] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv preprint arXiv:1506.01497, 2015