

---

# **Software Design**

# **Specification**

**for**

# **AEye**

**Vidhu Chaudhary**

**Department of Computer Science**

**Banasthali Vidyapith**

**07-02-2021**

# Table of Contents

<b>1. Introduction</b>	<b>1</b>
1.1 Purpose of this document	1
1.2 Scope of the development project	2
1.3 Definitions, acronyms, and abbreviations	2
1.4 References	3
1.5 Overview of document	3
<b>2. System architecture description</b>	<b>4</b>
2.1 Overview of modules / components	4
2.2 Structure and relationships	5
2.3 User Interface issues	9
<b>3. Detailed Description of components</b>	<b>12</b>
3.1 Object	12
3.2 Microsoft COCO Dataset	13
3.3 YOLO (You Only Look Once)	14
<b>4. Reuse and relationship to other products</b>	<b>16</b>
<b>5. Pseudocode for components</b>	<b>16</b>
<b>6. Appendices</b>	<b>22</b>

# **1. Introduction**

This software design specification (SDS) report is a detailed description for the system called AEye - Artificial Eye, which is planned to be developed by our project team. AEye is basically a mobile application project whose aim is to detect common day to day objects in one's vicinity in real time. Although, its end users are mainly specified as visually impaired people, anyone having a mobile phone with an android operating system will be able to use this application.

Visually Impaired People confront many problems in identifying objects in their surroundings. A solution which is easily available is needed to solve the problems of visually Impaired people. This project tries to transform the visual world into the audio world with the potential to inform visually impaired people about the objects in their local environment.

Mobile applications are software designed applications that are mainly designed to run on several devices like smartphones, tablets or computers. The aim of the team was to develop an application which will be used by visually impaired people for detecting objects easily around them. For Object detection Component, the team was required to have a clear, well-defined feasibility study because of several reasons. Firstly, since the aim of the AEye project is to develop a mobile application that can be run on smartphones, processors to run the object detection algorithms were restricted. Despite these limitations, OpenCV provided a nice library for the applications that will run on the Android operating system. For example, to track the points between video frames, an optical flow algorithm can be implemented easily for Android devices. Furthermore, several edge detection and enhancement methods can be developed and used on a smart phone. The other most important component for application is the text to speech API functions through which the users of the application will be informed about the objects.

Our system consists of several modules. Video is captured with an inbuilt android camera on the client side, and is streamed to the server for real-time image recognition with existing object detection models (YOLO). The 3D objects are then determined by bounding boxes from the detection algorithm in addition to an audio speech to recognize the objects with their names. Audios containing the name of the object are played at the interval of a few seconds, when the recognized object differs from the previous one.

## **1.1 Purpose of this document**

The purpose of this SDS report is to provide a detailed description of an object detection system. This document will present the detailed description of all the components being used to develop the application AEye. Moreover, this report will also contain the constraints under which this system will be operating.

Thus, in this SDS a model has been proposed which makes the use of smartphones and technology to make an application which can help the visually impaired users detect objects in their surroundings.

In this SDS, the model of an android application has been proposed which is designed to detect multiple objects in real-time with the simplest interface that can be easily used by visually impaired people i.e. the target users of our application.

## 1.2 Scope of the development project

“AEye”, is the name of the project which is planned to be developed by our team. The product will be an android application designed for the detection of multiple objects in real time. The system is an aided technology for the assistance of visually impaired people.

The aim of the project is to develop an application that can detect the objects in the user's surroundings. It identifies objects familiar to the user in his daily life, and then tells the user of these detected objects to aid him in his daily life routines. The reason it is more reliable is because it is developed on the Android operating system and Android-based smartphones are very common and highly available almost everywhere. In fact, it's one of the most used mobile operating systems. This makes the application convenient to get. However, it is focused on identifying the most common things in one's day to day life, which are thought to be essential by the team.

The application will not require from the users any personal information to be inserted to the application. There will not be any login page or password requirements to use the application. However, to download the application from Google Store, a Gmail account is required. This obligation is applied by Google, not by the team developing the application. Also, basic permissions such as access to the camera will be required.

## 1.3 Definitions, acronyms, and abbreviations

### 1.3.1 Definitions:

- **API:** An application programming interface, is a computing interface that defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc.
- **Android Operating System:** Android is a mobile operating system based on a modified version of the Linux kernel and other open-source software, designed primarily for touchscreen mobile devices such as smartphones and tablets.
- **Convolution Neural Network:** A Convolutional Neural Network is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other.
- **Image Segmentation:** Image segmentation is the process of partitioning a digital image into multiple segments. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. It is typically used to locate objects and boundaries (lines, curves, etc.) in images and assign a label to every pixel in an image such that pixels with the same label share certain characteristics.
- **Feature Extraction Segmentation:** Feature extraction in image processing is a method of transforming large redundant data into a reduced data representation. Transforming the input data into the set of features is called feature extraction.
- **Dataset:** A data set is a collection of related, discrete items of related data that may be accessed individually or in combination or managed as a whole entity.

### 1.3.2 Acronyms:

Acronyms	Definition
YOLO	You Only Look Once
OpenCV	Open Source Computer Vision library
ConvNet	Convolution Network
GUI	Graphical User Interface
DPM	Deformable Parts Model
R-CNN	Region Based Convolutional Neural Networks
MS-COCO	The Microsoft Common Objects in Context

### 1.4 References

- [1] Pressman Roger S., Software Engineering “A practitioner’s Approach” Fifth Edition , Publication
- [2] R. Rajwani , D. Purswani , P. Kalinani , D. Ramchandani, I. Dokare ,  
“ Proposed System on Object Detection for Visually Impaired People ”, *International Journal of Information Technology*, vol. 4, no. 1, Mar ,2018. [Online serial]. Available: <http://www.ijitjournal.org/volume-4/issue-2/IJIT-V4I2P1.pdf>
- [3] <https://github.com/Garima13a/YOLO-Object-Detection>

### 1.5 Overview of document

Visually Impaired People confront many problems in identifying objects in their surroundings. A solution which is easily available is needed to solve the problems of visually Impaired people. This project tries to transform the visual world into the audio world with the potential to inform visually impaired people about the objects in their local environment.

Our system consists of several modules. Video is captured with an inbuilt android camera on the client side, and is streamed to the server for real-time image recognition with existing object detection models (YOLO). The 3D objects are then determined by bounding boxes from the detection algorithm in addition to an audio speech to recognize the objects with their names. Audios containing the name of the object are played at the interval of a few seconds, when the recognized object differs from the previous one.

## 2. System architecture description

### 2.1 Overview of modules / components

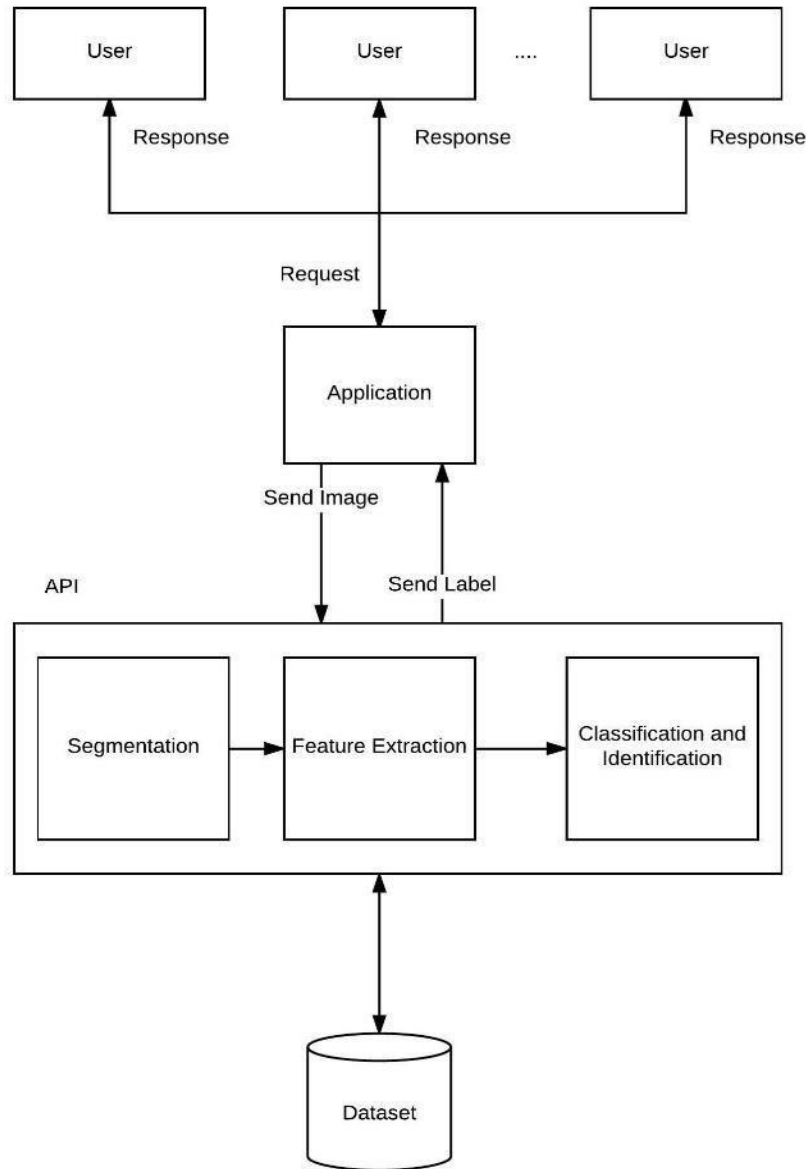
To successfully detect surrounding objects, we investigate several existing detection systems that could classify objects and evaluate them at various locations in an image. Deformable Parts Model (DPM) uses root filters that slides detection windows over the entire image. RCNN uses region proposal methods to generate possible bounding boxes in an image. Then, it applies various ConvNets to classify each box. The results are then post processed and output finer boxes. The slow test-time, complex training pipeline and the large storage does not fit into

our application. Fast R-CNN max-pools proposed regions and combines the computation of ConvNet for each proposal of an image and outputs features of all regions at once. Based on Fast R-CNN, Faster R-CNN inserts a region proposal network after the last layer of ConvNet.

Both methods speed up the computational time and improve the accuracy. The pipelines of these methods are still relatively complex and hard to optimize. Considering the requirement of real-time objective detection, in this project, we use You Only Look Once (YOLO) model. YOLO could efficiently provide relatively good objective detection with extremely fast speed.

The prototype we build successfully recognizes visual objects and presents the detection information as 3D sound, giving the user a sense of “augmented reality”. However, the prototype suffers from the following limitations. First, it is common for a user to focus on a certain object from afar and navigate to a location close to the object. In this task, the user needs a consistent instruction of the target object from approximately 10 m away to only 20 cm away. That imposes a very high requirement to the object detection model. In our experience, YOLO can correctly detect objects, such as a chair, within a range about 2-5 m away. Objects that are outside this range are either unrecognized or misclassified. One approach to solve this issue is to incorporate training images with greater scale ranges (e.g., include a chair picture captured from 20 cm away and 10 m away). However, it may be difficult for object detection models to classify the object from a picture of extreme scale (too close or too far). Another approach to solve this is to use object tracking algorithm to track the object (e.g. a chair) once the user has identified it as the target. These two approaches are worth exploring in the future work. The second issue reported by the blind user is the blocking of ambient sound by using earbuds. However, this can be solved by using bone conduction earphones, which leave ears open for hearing surrounding sounds. The third issue reported by the blind user is “information overload” when the system is trying to notify users of multiple objects at the same time. This can be solved by delayed notifications. For example, the system can sequentially notify the user of the object from left to right. However, this solution requires the user stands still while playing the 3D sounds. Moreover, blind people usually do not want to know every object in his “eyesight”, but instead want to know objects that are pertinent to their immediate need. For example, they may want to find a particular room in a building, or find food and drinks during a conference. In this regard, the system should have three modes: exploration mode where users are notified with every detected object, search mode where the system only notify users of the object they are looking for, and navigation mode where only the target object and obstacle objects are notified to users in real time. In sum, extensive work is required to analyze users’ needs if one would like to stem from this prototype to a really helpful assistive product.

## 2.2 Structure and relationships



The above diagram is the architectural / conceptual diagram of our system. It is a four layered architecture. Through this conceptual design we are trying to show the interaction between the different layers of our system, mainly dividing into four main components as mentioned above.

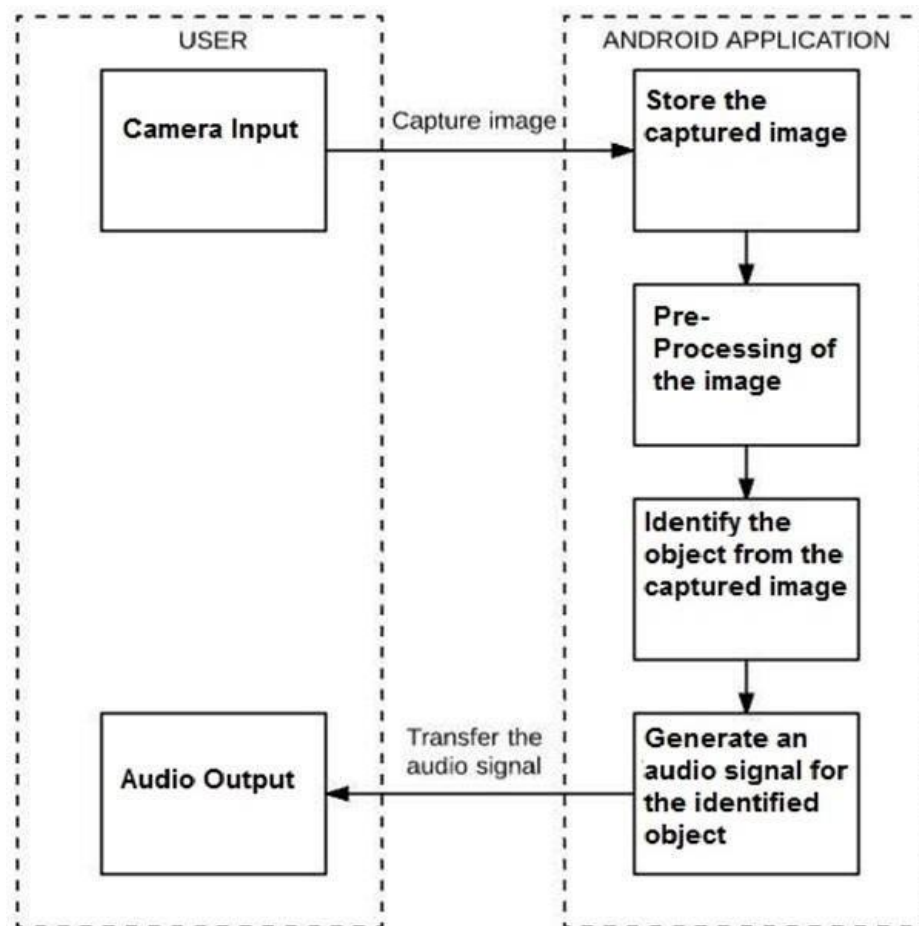
The first layer depicts the users, which is the external layer of our entire architecture. The figure shows an interaction between the users and the middle layer which is the Android application.

All the requests of the users are taken by the Android application and are given to the API for processing. The API uses a dataset which contains thousands of labelled images and compares the current image with the images in the dataset.

After classification and identification, the API sends the label of the current image to the application.

The application mainly uses Android along with many supported libraries. The camera on an Android smartphone will be used to capture an image of the surrounding which will be stored in Android's memory. This image will be processed by using libraries like OpenCV and Google Cloud Vision API.

Google Cloud Vision API uses Google Cloud. The Image is sent to the cloud through the internet. It uses the COCO dataset to compare the input image with millions of other images. The process gets completed and the objects are identified in the image. The user gets informed about the identified objects present in his surroundings via an audio output. Fig. 2 ahead shows the Module Diagram of the system.

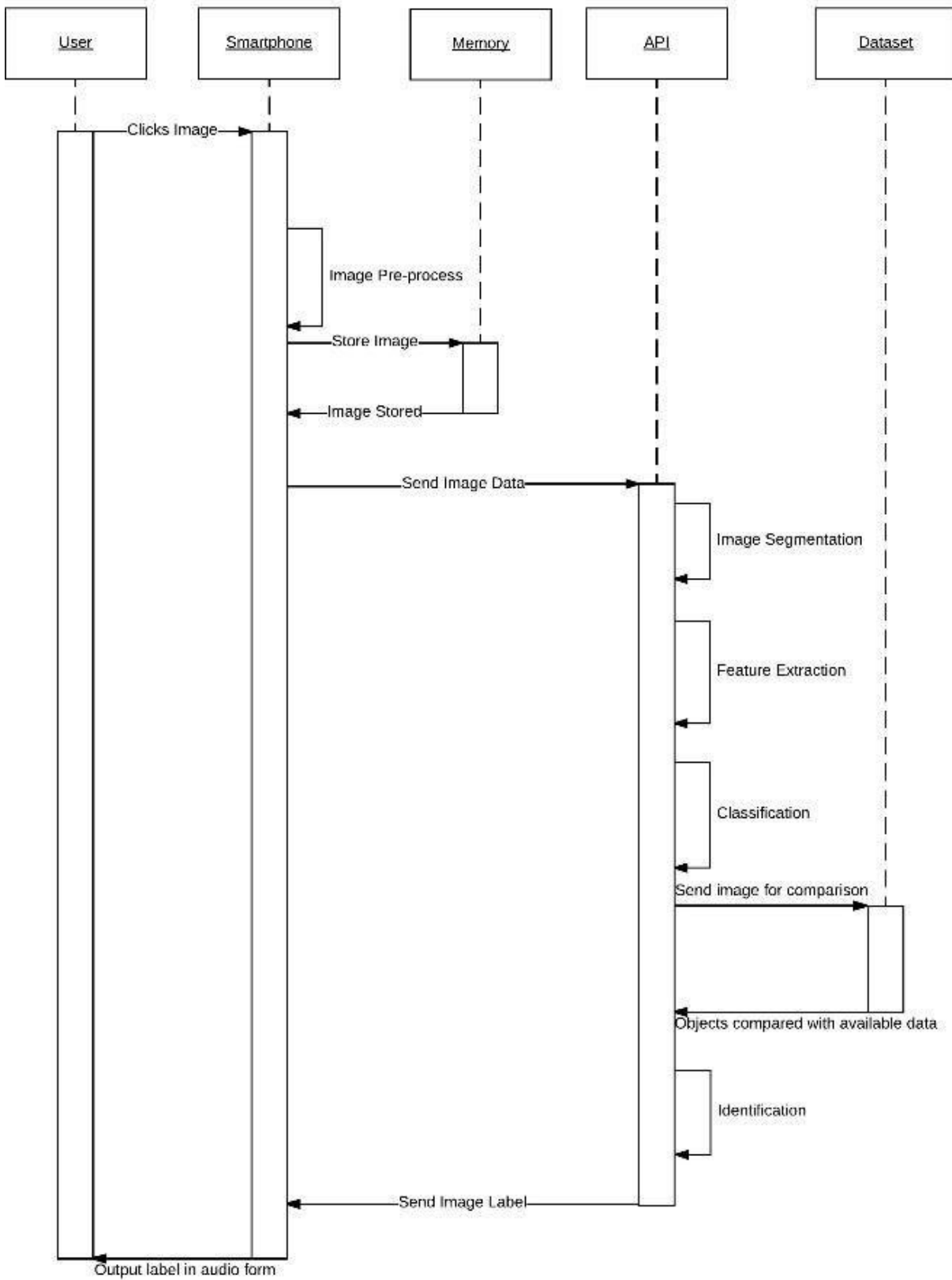


Sequence diagram is a communication outline that shows how objects work with each other and in what order. It is a development of a message sequence chart. A sequence diagram indicates question cooperation masterminded in time arrangement. It portrays the items and classes engaged with the



situation and the arrangement of messages traded between the articles expected to complete the usefulness of the situation.

Sequence diagrams are normally connected with use case realizations in the Logical View of the framework, a work in progress. Sequence diagrams are ordinarily connected with utilize cases acknowledged in the Logical View of the framework being worked on. Sequence diagrams are sometimes called event diagrams or event scenarios.



The above sequence diagram of our framework clarifies the stream of the framework, that is, what action takes place first and what action will follow the previous action. First, the user starts the application and captures the image of the surrounding in front of him or of the object in front of him which he wants to identify. The application digitizes and stores the captured image in the memory. It is then used to detect objects in other images. This image will be processed by using libraries like OpenCV and Google Cloud Vision API.

The OpenCV library contains many Image Processing algorithms and Google Cloud Vision API has the power to compare the input image with millions of other images using Microsoft's COCO Dataset. The API receives image data, performs image segmentation, feature extraction, classification & identification functions and uses the COCO dataset to get the image label. The application then converts the label into audio and sends it via the speaker.

The user gets informed about the identified objects present in his surroundings via an audio output.

## **2.3 User interface issues**

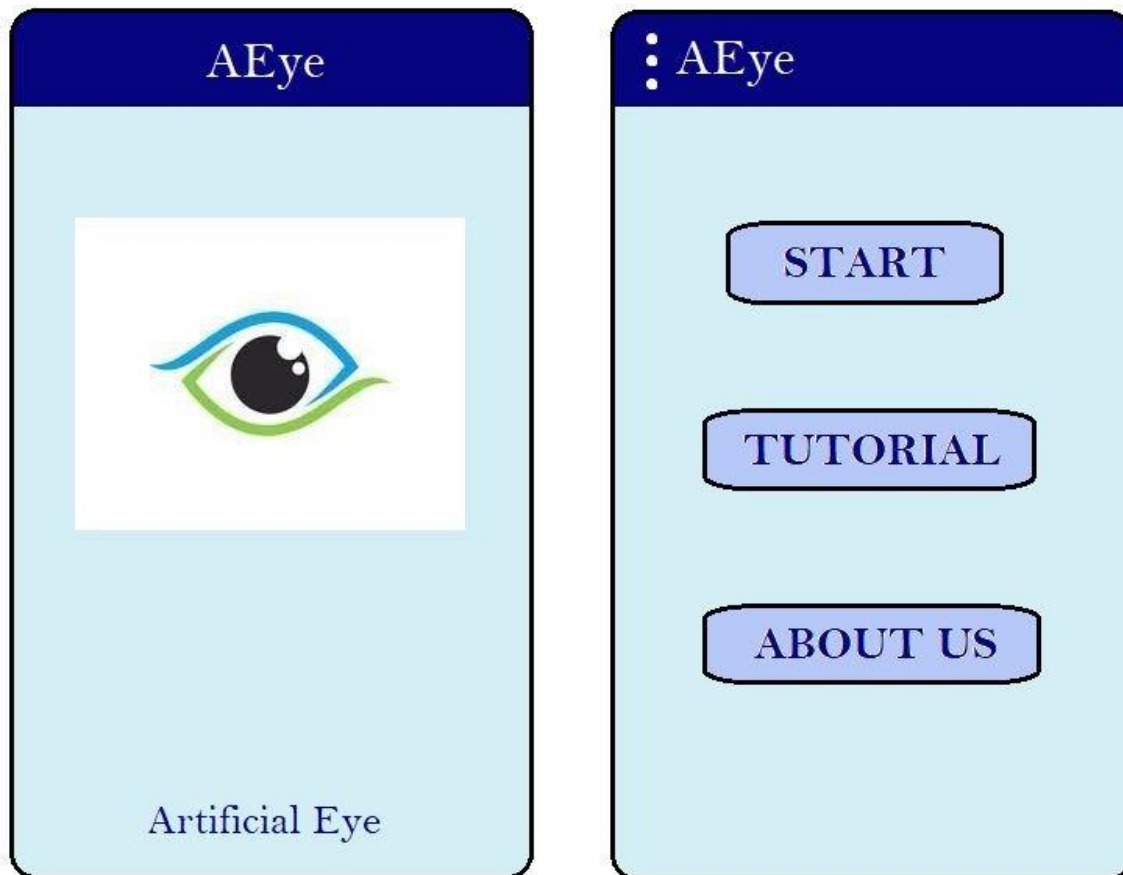
There will be one graphical user interface for the use of people who are not visually impaired. There will be two basic pages belonging to this GUI. This GUI will be responsible for providing communication between the user and the application. Before mentioning the specific features of each page, there are some common features of all pages in the user interface of AEye. Firstly, links to all these pages will be displayed on the screen via buttons. The contents of the pages will be displayed in English language.

Properties of each page of the application are listed below:

### **Main Page:**

Basic graphical user interface for the main page of the application will be in English as mentioned above. There will be a start button on this page which on clicking will redirect the user to the second page of the application which is the object detection page. In addition, buttons for navigating to other pages of the application such as tutorial, user manual and brief description about the application will also be present on the main page. Moreover, a pop up window requesting access to the camera will be prompted at the start of the application for the user to allow the functioning of the object detection feature.

A simple layout of the main page for the application is:



### Object Detection Page:

As soon as the user is directed to the second page of the application the camera is launched at the page and it starts detecting the objects that are captured. As the surrounding objects are detected one by one the application will prompt the user about the identity of the object in an audio form. The page will also provide a stop button to stop the object detection being processed by the application.

### Events:

***startApplication:*** All users of the application will be able to start the application via clicking the application icon on the device.

***exitApplication:*** All users will be able to exit and close the application via a physical button of the device.

***startObjectDetection:*** The users will be informed about the occurrence of an object captured in the camera via speech.

**stopObjectDetection:** The users will be able to stop the object detecting component without having to exit the application.

A simple layout of the object detection page for the application is:



### 3. Detailed description of components

#### 3.1 Object

<b>Identification</b>	Objects that can be detected with the help of the dataset and the system.
<b>Type</b>	Object (Subjected as an input for the system)
<b>Purpose</b>	Serves as a subject to perform detection on and also as an input for the system via the camera.
<b>Function</b>	Acts as an input that is taken in the form of an image and then identified by the detection system.
<b>Subordinates</b>	NA
<b>Dependencies</b>	Objects are taken as an input by all the other components and the processing module processes the identification of these objects relative to the dataset used.
<b>Interfaces</b>	NA
<b>Resources</b>	Detection is performed with the help of COCO Dataset as a resource for identification.
<b>Processing</b>	NA

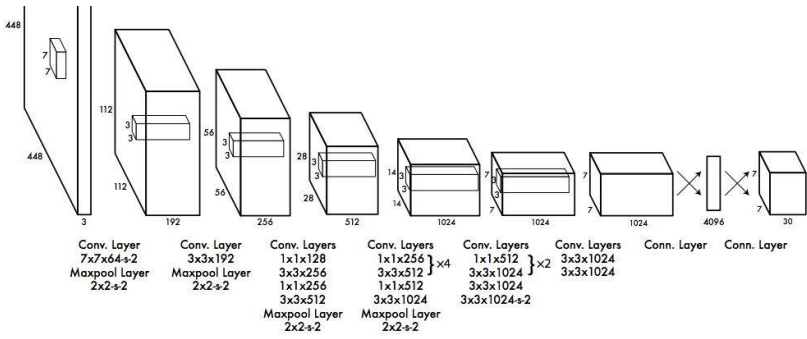
### 3.2 Microsoft COCO Dataset

<b>Identification</b>	Microsoft COCO Dataset
<b>Type</b>	Dataset
<b>Purpose</b>	Dataset acts as a reference to match and identify the objects to be detected.
<b>Function</b>	Dataset is used to recognize and classify the objects taken as input.
<b>Subordinates</b>	The dataset consists of images and their labels with details for stating that an object belonging to a specified class is present, and localizing it in the image. The location of an object is typically represented by a bounding box.
<b>Dependencies</b>	The dataset is used by the YOLO algorithm to classify and recognize the objects that need to be detected by the system.
<b>Interfaces</b>	NA
<b>Resources</b>	It itself acts as a resource for the system.
<b>Processing</b>	NA
<b>Data</b>	The data represented in the dataset is represented in columns of image, image filename, image id, panoptic image, panoptic image filename, panoptic objects: area, bbox, id, label.

### 3.3 YOLO (You Only Look Once) Model

<b>Identification</b>	YOLO Model.
<b>Type</b>	Module
<b>Purpose</b>	To successfully detect surrounding objects, we use the existing detection system, YOLO that could classify objects and evaluate it at various locations in an image. Considering the requirement of real-time object detection, in this project, we use You Only Look Once (YOLO) model. YOLO could efficiently provide relatively good object detection with extremely fast speed.
<b>Function</b>	The YOLO model is used for object detection purpose. It provides a pretrained neural network to detect objects. In our project we use the MS-COCO dataset to train the network. The module takes an image as input, identifies the object(s) in the image and then builds a grid around the identified objects with their name and confidence score.



<b>Subordinates</b>	 <p>The ConvNet architecture is shown in Figure above. The network has 24 convolutional layers with 2 fully connected layers. The ConvNet is to extract features from input images and the fully connected layers are to predict the probability of the boxes coordinates and confidence score. The accuracies of the predictions also depend on the architecture of the network. The loss function of the final output depends on the x, y, w, h, prediction of classes and overall probabilities. In our project, we use pre-trained YOLO weight to detect objects.</p>
<b>Dependencies</b>	<p>Our project is based on a platform that is capable of processing real-time image. A pipeline is developed that enables us to communicate quickly. A program in local machine extracts raw image from a camera, encodes it into a string and sends through a client to a server. The server decodes it and uses trained object detection engine to return detected items.</p>
<b>Interfaces</b>	<p>NA</p>
<b>Resources</b>	<p>A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services.</p>

	The YOLO model requires some resources like: Execution time, Both primary and secondary memory, buffers, I/O channels, the OpenCV(cv2), matplotlib, darknet libraries and the MS-COCO Dataset.
<b>Processing</b>	The processing of the YOLO model has been explained in detail in the section 5.0 along with the pseudocode.
<b>Data</b>	NA

## 4. Reuse and relationships to other products

Our team has heavily incorporated the idea of reuse in the development of this project. The application prominently uses resources such as APIs, dataset and libraries for our processing module. The team has tried to make use of open source resources as much as possible so that if need be, then other developers can also contribute to this project.

## 5. Pseudocode for components

### Pseudocode for YOLO Algorithm:

#### Introduction

As you learned in the previous lessons, YOLO is a state-of-the-art, real-time object detection algorithm. In this notebook, we will apply the YOLO algorithm to detect objects in images. We have provided a series of images that you can test the YOLO algorithm on. Below is a list of the available images that you can load:

- cat.jpg
- city\_scene.jpg
- dog.jpg
- dog2.jpg
- eagle.jpg
- food.jpg
- giraffe.jpg
- horses.jpg
- motorbike.jpg
- person.jpg
- surf.jpg
- wine.jpg

## Importing Resources ¶

We will start by loading the required packages into Python. We will be using *OpenCV* to load our images, *matplotlib* to plot them, *utils* module that contains some helper functions, and a modified version of *Darknet*. YOLO uses *Darknet*, an open source, deep neural network framework written by the creators of YOLO. The version of *Darknet* used in this notebook has been modified to work in PyTorch 0.4 and has been simplified because we won't be doing any training. Instead, we will be using a set of pre-trained weights that were trained on the Common Objects in Context (COCO) database. For more information on *Darknet*, please visit [Darknet](#).

```
In [2]: import cv2
import matplotlib.pyplot as plt

from utils import *
from darknet import Darknet
```

## Setting Up The Neural Network

We will be using the latest version of YOLO, known as YOLOv3. We have already downloaded the `yolov3.cfg` file that contains the network architecture used by YOLOv3 and placed it in the `/cfg/` folder. Similarly, we have placed the `yolov3.weights` file that contains the pre-trained weights in the `/weights/` directory. Finally, the `/data/` directory, contains the `coco.names` file that has the list of the 80 object classes that the weights were trained to detect.

In the code below, we start by specifying the location of the files that contain the neural network architecture, the pre-trained weights, and the object classes. We then use *Darknet* to setup the neural network using the network architecture specified in the `cfg_file`. We then use the `.load_weights()` method to load our set of pre-trained weights into the model. Finally, we use the `load_class_names()` function, from the `utils` module, to load the 80 object classes.

```
In [3]: # Set the location and name of the cfg file
cfg_file = './cfg/yolov3.cfg'

# Set the location and name of the pre-trained weights file
weight_file = './weights/yolov3.weights'

# Set the location and name of the COCO object classes file
namesfile = 'data/coco.names'

# Load the network architecture
m = Darknet(cfg_file)

# Load the pre-trained weights
m.load_weights(weight_file)
```

```
# Load the COCO object classes
class_names = load_class_names(namesfile)
```

Loading weights. Please Wait...100.00% Complete

## Taking a Look at The Neural Network

Now that the neural network has been setup, we can see what it looks like. We can print the network using the `.print_network()` function.

```
In [4]: # Print the neural network used in YOLOv3
m.print_network()
```

layer	filters	size	input	output
0 conv	32	3 x 3 / 1	416 x 416 x 3	-> 416 x 416 x 32
1 conv	64	3 x 3 / 2	416 x 416 x 32	-> 208 x 208 x 64
2 conv	32	1 x 1 / 1	208 x 208 x 64	-> 208 x 208 x 32
3 conv	64	3 x 3 / 1	208 x 208 x 32	-> 208 x 208 x 64
4 shortcut 1				
5 conv	128	3 x 3 / 2	208 x 208 x 64	-> 104 x 104 x 128
6 conv	64	1 x 1 / 1	104 x 104 x 128	-> 104 x 104 x 64
7 conv	128	3 x 3 / 1	104 x 104 x 64	-> 104 x 104 x 128
8 shortcut 5				
9 conv	64	1 x 1 / 1	104 x 104 x 128	-> 104 x 104 x 64
10 conv	128	3 x 3 / 1	104 x 104 x 64	-> 104 x 104 x 128
11 shortcut 8				

12 conv	256	3 x 3 / 2	104 x 104 x 128	->	52 x 52 x 256
13 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
14 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
15 shortcut	12				
16 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
17 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
18 shortcut	15				
19 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
20 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
21 shortcut	18				
22 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
23 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
24 shortcut	21				
25 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
26 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
27 shortcut	24				
28 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
29 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
30 shortcut	27				
31 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
32 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
33 shortcut	30				
34 conv	128	1 x 1 / 1	52 x 52 x 256	->	52 x 52 x 128
35 conv	256	3 x 3 / 1	52 x 52 x 128	->	52 x 52 x 256
36 shortcut	33				
37 conv	512	3 x 3 / 2	52 x 52 x 256	->	26 x 26 x 512
38 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
39 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
40 shortcut	37				
41 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
42 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
43 shortcut	40				
44 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
45 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
46 shortcut	43				
47 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
48 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
49 shortcut	46				
50 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
51 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
52 shortcut	49				
53 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
54 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
55 shortcut	52				
56 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
57 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
58 shortcut	55				
59 conv	256	1 x 1 / 1	26 x 26 x 512	->	26 x 26 x 256
60 conv	512	3 x 3 / 1	26 x 26 x 256	->	26 x 26 x 512
61 shortcut	58				
62 conv	1024	3 x 3 / 2	26 x 26 x 512	->	13 x 13 x1024
63 conv	512	1 x 1 / 1	13 x 13 x1024	->	13 x 13 x 512
64 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x1024
65 shortcut	62				
66 conv	512	1 x 1 / 1	13 x 13 x1024	->	13 x 13 x 512
67 conv	1024	3 x 3 / 1	13 x 13 x 512	->	13 x 13 x1024
68 shortcut	65				

```

69 conv 512 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 512
70 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024
71 shortcut 68
72 conv 512 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 512
73 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024
74 shortcut 71
75 conv 512 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 512
76 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024
77 conv 512 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 512
78 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024
79 conv 512 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 512
80 conv 1024 3 x 3 / 1 13 x 13 x 512 -> 13 x 13 x1024
81 conv 255 1 x 1 / 1 13 x 13 x1024 -> 13 x 13 x 255
82 detection
83 route 79
84 conv 256 1 x 1 / 1 13 x 13 x 512 -> 13 x 13 x 256
85 upsample * 2 13 x 13 x 256 -> 26 x 26 x 256
86 route 85 61
87 conv 256 1 x 1 / 1 26 x 26 x 768 -> 26 x 26 x 256
88 conv 512 3 x 3 / 1 26 x 26 x 256 -> 26 x 26 x 512
89 conv 256 1 x 1 / 1 26 x 26 x 512 -> 26 x 26 x 256
90 conv 512 3 x 3 / 1 26 x 26 x 256 -> 26 x 26 x 512
91 conv 256 1 x 1 / 1 26 x 26 x 512 -> 26 x 26 x 256
92 conv 512 3 x 3 / 1 26 x 26 x 256 -> 26 x 26 x 512
93 conv 255 1 x 1 / 1 26 x 26 x 512 -> 26 x 26 x 255
94 detection
95 route 91
96 conv 128 1 x 1 / 1 26 x 26 x 256 -> 26 x 26 x 128
97 upsample * 2 26 x 26 x 128 -> 52 x 52 x 128

97 upsample * 2 26 x 26 x 128 -> 52 x 52 x 128
98 route 97 36
99 conv 128 1 x 1 / 1 52 x 52 x 384 -> 52 x 52 x 128
100 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256
101 conv 128 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 128
102 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256
103 conv 128 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 128
104 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256
105 conv 255 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 255
106 detection

```

As we can see, the neural network used by YOLOv3 consists mainly of convolutional layers, with some shortcut connections and upsample layers. For a full description of this network please refer to the [YOLOv3 Paper](#).

## Loading and Resizing Our Images

In the code below, we load our images using OpenCV's `cv2.imread()` function. Since, this function loads images as BGR we will convert our images to RGB so we can display them with the correct colors.

As we can see in the previous cell, the input size of the first layer of the network is 416 x 416 x 3. Since images have different sizes, we have to resize our images to be compatible with the input size of the first layer in the network. In the code below, we resize our images using OpenCV's `cv2.resize()` function. We then plot the original and resized images.

```

In [8]: # Set the default figure size
plt.rcParams['figure.figsize'] = [24.0, 14.0]

# Load the image
img = cv2.imread('./images/dog.jpg')

# Convert the image to RGB
original_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

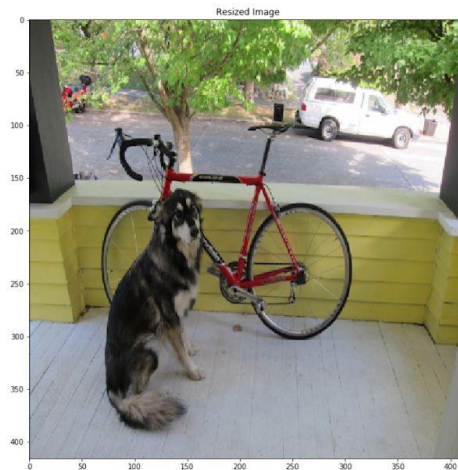
# We resize the image to the input width and height of the first Layer of the network.
resized_image = cv2.resize(original_image, (m.width, m.height))

# Display the images
plt.subplot(121)
plt.title('Original Image')
plt.imshow(original_image)
plt.subplot(122)
plt.title('Resized Image')
plt.imshow(resized_image)

```



```
plt.imshow(resized_image)
plt.show()
```



## Setting the Non-Maximal Suppression Threshold

As you learned in the previous lessons, YOLO uses **Non-Maximal Suppression (NMS)** to only keep the best bounding box. The first step in NMS is to remove all the predicted bounding boxes that have a detection probability that is less than a given NMS threshold. In the code below, we set this NMS threshold to 0.6. This means that all predicted bounding boxes that have a detection probability less than 0.6 will be removed.

```
In [ ]: # Set the NMS threshold
nms_thresh = 0.6
```

## Setting the Intersection Over Union Threshold

After removing all the predicted bounding boxes that have a low detection probability, the second step in NMS, is to select the bounding boxes with the highest detection probability and eliminate all the bounding boxes whose **Intersection Over Union (IOU)** value is higher than a given IOU threshold. In the code below, we set this IOU threshold to 0.4. This means that all predicted bounding boxes that have an IOU value greater than 0.4 with respect to the best bounding boxes will be removed.

In the `utils` module you will find the `nms` function, that performs the second step of Non-Maximal Suppression, and the `boxes_iou` function that calculates the Intersection over Union of two given bounding boxes. You are encouraged to look at these functions to see how they work.

```
In [6]: # Set the IOU threshold
iou_thresh = 0.4
```

## Object Detection

Once the image has been loaded and resized, and you have chosen your parameters for `nms_thresh` and `iou_thresh`, we can use the YOLO algorithm to detect objects in the image. We detect the objects using the `detect_objects(m, resized_image, iou_thresh, nms_thresh)` function from the `utils` module. This function takes in the model `m` returned by *Darknet*, the resized image, and the NMS and IOU thresholds, and returns the bounding boxes of the objects found.

Each bounding box contains 7 parameters: the coordinates  $(x, y)$  of the center of the bounding box, the width  $w$  and height  $h$  of the bounding box, the confidence detection level, the object class probability, and the object class id. The `detect_objects()` function also prints out the time it took for the YOLO algorithm to detect the objects in the image and the number of objects detected. Since we are running the algorithm on a CPU it takes about 2 seconds to detect the objects in an image, however, if we were to use a GPU it would run much faster.

Once we have the bounding boxes of the objects found by YOLO, we can print the class of the objects found and their corresponding object class probability. To do this we use the `print_objects()` function in the `utils` module.

Finally, we use the `plot_boxes()` function to plot the bounding boxes and corresponding object class labels found by YOLO in our image. If you set the `plot_labels` flag to `False` you will display the bounding boxes with no labels. This makes it easier to view the bounding boxes if your `nms_thresh` is too low. The `plot_boxes()` function uses the same color to plot the bounding boxes of the same object class. However, if you want all bounding boxes to be the same color, you can use the `color` keyword to set the desired color. For example, if you want all the bounding boxes to be red you can use:

```
plot_boxes(original_image, boxes, class_names, plot_labels = True, color = (1,0,0))
```

You are encouraged to change the `iou_thresh` and `nms_thresh` parameters to see how they affect the YOLO detection algorithm. The default values of `iou_thresh = 0.4` and `nms_thresh = 0.6` work well to detect objects in different kinds of images. In the cell below,

You are encouraged to change the `iou_thresh` and `nms_thresh` parameters to see how they affect the YOLO detection algorithm. The default values of `iou_thresh = 0.4` and `nms_thresh = 0.6` work well to detect objects in different kinds of images. In the cell below, we have repeated some of the code used before in order to prevent you from scrolling up down when you want to change the `iou_thresh` and `nms_thresh` parameters or the image. Have Fun!

```
In [9]: # Set the default figure size
plt.rcParams['figure.figsize'] = [24.0, 14.0]

# Load the image
img = cv2.imread('./images/dog.jpg')

# Convert the image to RGB
original_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# We resize the image to the input width and height of the first layer of the network.
resized_image = cv2.resize(original_image, (m.width, m.height))

# Set the IOU threshold. Default value is 0.4
iou_thresh = 0.4

# Set the NMS threshold. Default value is 0.6
nms_thresh = 0.6

# Detect objects in the image
boxes = detect_objects(m, resized_image, iou_thresh, nms_thresh)

# Print the objects found and the confidence level
print_objects(boxes, class_names)

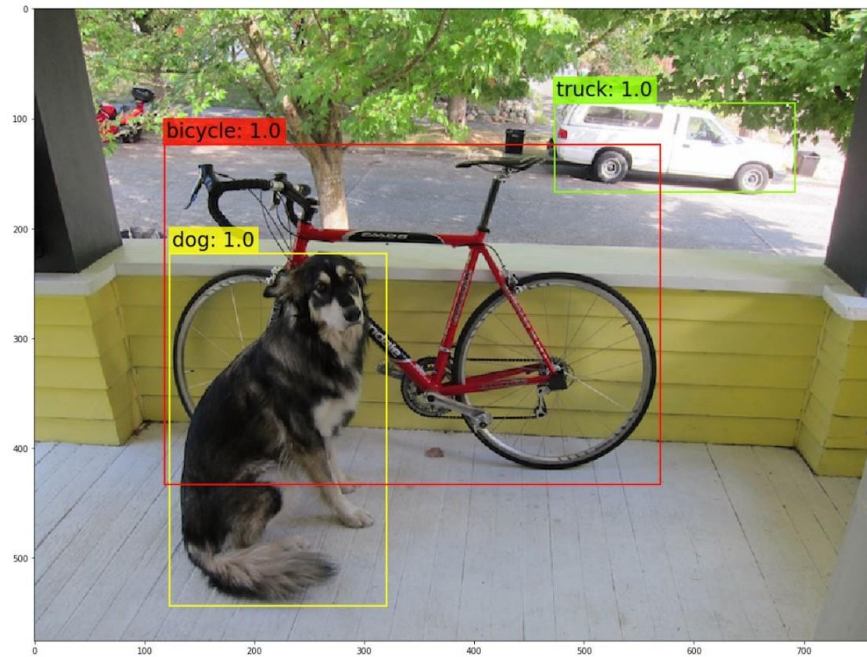
#Plot the image with bounding boxes and corresponding object class labels
plot_boxes(original_image, boxes, class_names, plot_labels = True)
```

It took 2.083 seconds to detect the objects in the image.

Number of Objects Detected: 3

Objects Found and Confidence Level:

```
1. dog: 0.999997
2. truck: 0.991125
3. bicycle: 0.999998
```



## 6. Appendices (if any)

There is no available appendix for this SDS report.