DETECTION OF ANIMALS USING MACHINE LEARNING TO PREVENT INTRUSION AT ANY SPECIFIED PLACE

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

KODAVALLA DURGA AVINASH
(Reg. No.: 123003114, B. Tech CSE)
POTTURI G V S ASRITH
(Reg. No.: 123003194, B. Tech CSE)
YERASI VENKATA NITHIN REDDY
(Reg. No.: 123003277, B. Tech CSE)

June 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING
THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**Detection of Animals using Machine Learning to Prevent Intrusion at any Specified Place**" submitted as a requirement for the course, CSE300 / INT300 / ICT300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Kodavalla Durga Avinash (Reg No. 123003114, CSE), Mr. Potturi G V S Asrith (Reg No. 123003194, CSE), and Mr. Yerasi Venkata Nithin Reddy (Reg No. 123003277, CSE)** during the academic year 2021-22, in the School of Computing, under my supervision.

Signature of Project Supervisor :
Name with Affiliation : **Prof. Ezhilarasie .R (Asst. Professor – III)**
Date : **29th June 2022**

Mini Project *Viva-voce* held on **29th June 2022 .**

Examiner 1 **Examiner 2**

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First of all, we would like to thank God Almighty for his endless blessings.

We would like to express my sincere gratitude to **Dr. S. Vaidyasubramaniam, Vice – Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

We would like to forever remain grateful to **Dr. S. Swaminathan, Dean, Planning & Development** for encouraging, providing me the required support during the span of these past three years of my academic years at SASTA Deemed University.

We would forever remain grateful and I would like to thank **R. Chandramouli, Registrar** and **Dr. A. Umamakeswari, Dean, School of Computing,** for their overwhelming support provided during my course span in SASTRA Deemed University.

We were extremely grateful to **Dr. Shankar Sriram, Associate Dean, School of Computing** for his constant support, motivation, and academic help extended for the past three years of my life in the School of Computing.

We would specially thank and express my gratitude to **Dr. R. Ezhilarasie, Asst Professor – III, School of Computing** for providing me an opportunity to do this project and for her guidance and support to complete the project.

We also thank all the teaching and non–teaching faculty, and all other people who have directly or indirectly helped me through their support, encouragement, and all other assistance extended for the completion of my project and successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, We thank my parents and all others who helped me acquire this interest in the project and aided me in completing it within the deadline without much struggle.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

SSD	Single Shot Multi-Box Detector
CNN	Convolution Neural Network
ML	Machine Learning
AI	Artificial Intelligence
API	Application Programming Interface
CV	Computer Vision
IP	Image Processing
IoT	Internet of Things
TPU	Tensor Processing Unit
GPU	Graphics Processing Unit
CPU	Central Processing Unit

ABSTRACT

Animal intrusion is a serious issue in many places like agricultural fields, highways, remote villages, and base stations of various camps. This affects food security as the intrusion of animals will damage yield in agricultural lands, thus reducing farmers' profit. Also, intrusion on highways where vehicles move at a faster pace might come into collision with them, leading to disastrous accidents, and people in remote villages and camps are prone to wild animals invading unknowingly. There must be some way to detect the animals before they come to areas where they shouldn't be and take necessary precautions to prevent their intrusion and its consequences. In this work, a solution using the advancements in Machine Learning algorithm techniques, to detect animals is proposed. Machine Learning algorithms like Single Shot Detection are used to detect the animals and classify them based on their category to know whether it's a threat or not to take necessary precautions in advance. Also, integrating the already made open-source APIs into this project improves further in-depth analysis of the situations. This Machine Learning solution can be used to detect animals in advance and take necessary precautions such that animal intrusion damage consequences will be avoided.

KEYWORDS:

Single Shot Detector (SSD), Convolutional Neural Network (CNN), TensorFlow, Machine Learning, Application Programming Interface (API), Image Processing, Animal Intrusion, Computational.

CHAPTER 1
SUMMARY OF THE BASE PAPER & INTRODUCTION

Animal intrusion is a serious issue in many parts of the country, especially in the agricultural sector. As greater than 50% of the country's population is indirectly or directly dependent on this agricultural section, it is important to prevent animal intrusion as it's one of the major issues in this field. The intrusion of animal cause damage to the yields produced. If this isn't prevented then huge losses would incur and production of yields will be decreased in near future. This will definitely have a significant impact on our country's economy. Also, in places like remote areas and national highways through forested areas, animals do tend to come in between these operations and might cause damage to people and to themselves.

To date, there are traditional methods like shotguns, strings, electrified welding fences, etc implemented to prevent animal intrusions but these are not at all effective in stopping them and protecting crops. Our model uses Machine Learning concepts to detect animal intrusions at any specified place which can be further used in ARM-based devices like IoT [7] boards and mobile devices. Our ML model will have the capability to detect animals and also people intrusion in a quick way (less processing time) and label them. Detection of these animals and also human subjects will de be done using a camera.

Machine learning is an arm of artificial intelligence that is used for data analysis to automate the analytical model building to identify the pattern and objects to make a decision. In this work, a deep machine learning algorithm [2], a single-shot multi-box detector with the help of TensorFlow APIs is used for object detection and the classification models are trained and tested. All this work is to open up an effective model to protect the crops and people from animal intrusion through current technology and methods.

CHAPTER 2
MERITS AND DEMERITS OF THE BASE PAPER

2.1 Merits

The base paper's main work is to detect an animal intrusion in agricultural fields using any video capturing device with Machine Learning models. This detection of animals and sending alerts to the user will enable people to quickly react to animal intrusions in their fields so that their crop damage will be avoided. As there are many cases in which crops get damaged due to the intrusion of animals in the country which indirectly affects the economy and the lives of people dependent on the agricultural sector. Around 55% of the country's population is dependent on and generates their income from this agricultural sector and damage to these crops will have huge losses on their income leading to many other issues. Also, scarcity of food will be avoided with the detection of these animals. The machine learning model used in the base paper is very appropriate for this use case. The single-shot multi-box detection model is a type of convolutional neural network which is used in cases where we need faster detection but compromise on the accuracy front. As faster detection of animals is required in this case, then this type of machine learning model is perfect. Accuracy need not matter as long as the detection of animals and their classification is done.

2.2 Demerits

The base paper's work is to detect animal intrusion in agricultural fields only. This can be extended to animal intrusion detection on national highways, remote villages, base stations, etc. This model can be implemented at any specified place to find the animal intrusions and get alerted to take necessary actions. Also, accuracy might not matter but the accuracy results shown is very low which is around 50% to 60% in a few cases using a single-shot multi-box detection machine learning model. This is due to minimal datasets being used to train the model. This can be improved by training the model with more datasets.

CHAPTER 3
PROPOSED FRAMEWORK

The protection of agricultural fields, accidents on national highways, and injuries to people in remote areas due to the intrusion of animals are done using this proposed model using Machine Learning methodologies. The machine learning algorithm runs well on ARM-based devices and runs less effectively on x86-based devices using emulation. Any USB camera or webcam is used to capture the animals intruding in the specified place where they shouldn't be like agricultural fields, national highways, and remote villages & base stations. This live video stream is broken into images frame by frame and examined by the machine learning algorithm working on any particular device. Finally, if there is any detection then the conclusion frame is drawn on the output screen or video playback screen. If an intruder is detected then it will be shown on screen. This machine learning model for animal and human subjects' detection will be trained using custom datasets manually. Single-shot multi-box detector (SSD) machine learning model is used for object detection and the model is trained using TensorFlow APIs [9]. This model also uses other TensorFlow APIs to detect objects from the data stream.

3.1 Hardware & Software

An ARM-based device or any x86-based device with 64-bit architecture is required to test or demonstrate this model.

If using an x86-based device, then Intel bridge technology to emulate ARM-based commands on this device is needed to run this model.

A USB camera or webcam is required for video live capture to detect objects. [6]

Python version 3, TensorFlow, TensorFlow Lite, Numpy, and Computer Vision cv2 are the software required to train and use this model for object detection. [9]

A display to view the video data stream and point out the frame where the animal is detected. That frame also gives an insight into which animal and the accuracy of that animal's prediction.

3.2 Single Shot Multi-Box Detection (SSD) Model

Many object detector convolution neural networks CNNs are divided into two major groups. One type of CNNs is models that reach high accuracy but are very slow in detection and the other type are the models that have lower accuracy but are significantly faster. Single Shot Multi-Box detectors (SSD) model [8] is of type 2 CNNs, which have lower accuracy but detects the objects quickly which in our case is necessary. This model assumes object detection as a simple regression problem that takes in an image and learns the bounding box coordinates and class probabilities.

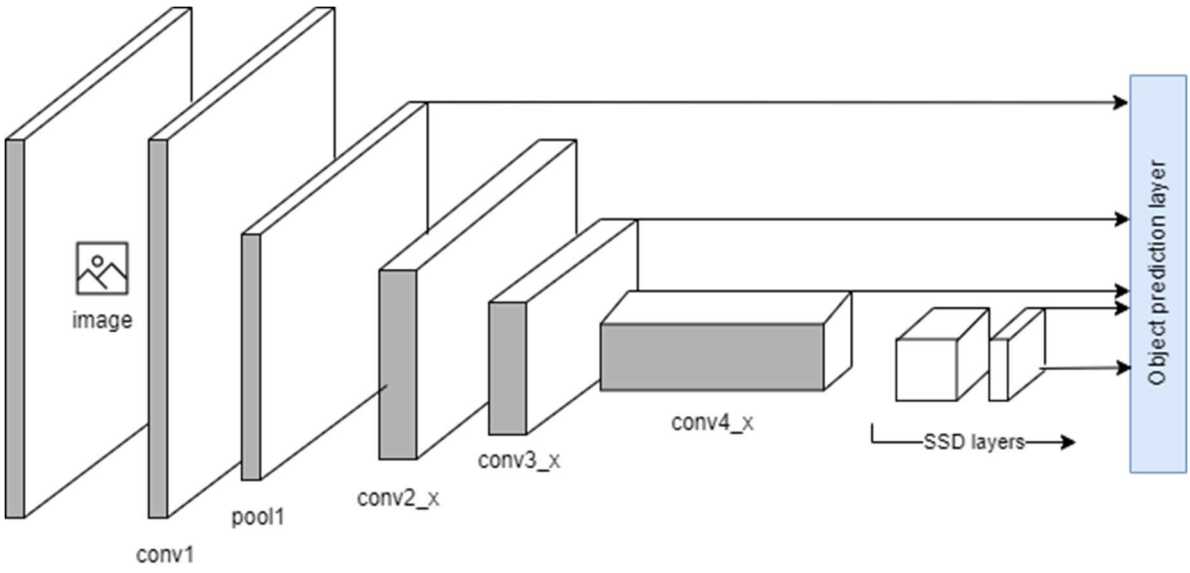

Fig. 1. The architecture of a CNN with an SSD detector

As shown in Fig. 1, this model consists of components like a ground network block and many multiscale feature blocks. The features of images are taken out using that base network block which is based on a deep CNN. There are many multiscale feature blocks in this model that are used to reduce the size and detect objects of different various sizes based on bounding box prediction and anchor boxes. In such a way, the detection of objects is faster in this model but compromises accuracy.

3.3 Google TensorFlow

TensorFlow is an open-source library developed by Google Inc. This is used to construct a numerical computation for the deep machine learning models. This library has many stages of abstraction to build and train models using APIs [9](Application Programming Interface). This is useful for debugging our model and training it in an easy manner, that gives flexibility.

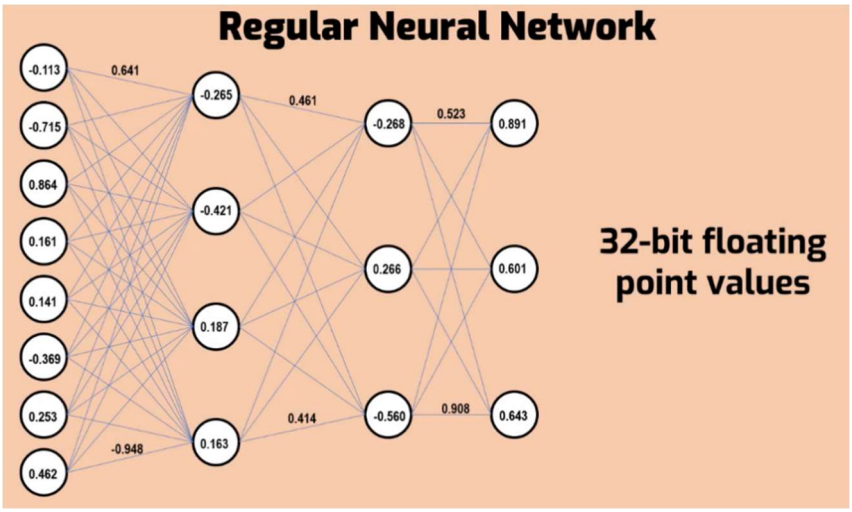

Fig. 2. TensorFlow Regular Neural Network

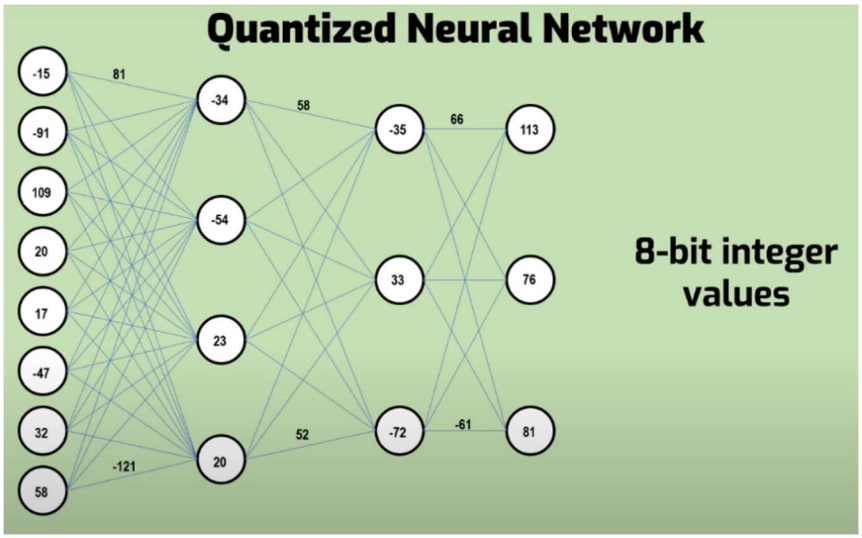

Fig. 3. TensorFlow Lite Quantized Neural Network

Google TensorFlow's regular neural network uses 32-bit floating-point values to detect objects using machine learning models. This is very time-consuming and high computation power required by the device to detect the object on the device whereas small IoT and ARM-based devices used for animal detection at any specified place might not have such power. Hence, TensorFlow Lite is used which is also developed by Google Inc, for low-powered ARM-based devices for object detection in a quicker manner converting the traditional TensorFlow model to 8-bit integer values.

3.4 Datasets

In this work different animal and human images are collected from the web sources like Google Images, Pinterest, etc for training our machine learning model using TensorFlow APIs. For experimentation in this work, considered 10 classes of animals such as birds, cats, dogs, horses, sheep, cows, elephants, bears, zebras, and giraffes. Also, human as a class is considered to show human as an intruder. Around 50-70 images per class were collected for training the model.

3.5 Training the Datasets

All the images are placed in the root directory of the TensorFlow folder as a sub-directory as training data. Then "labelImg" python application is launched in which we open up each and every image we considered for training the model and select the boundary of that object. After selecting the boundary, label the object and move to the next image. Similarly, labeling (class) must be done for all the images. After labeling, the application generates the XML file with all the data of images, boundaries, and labels. This file holds useful information for training for the machine learning model [4]. This XML file is now converted to XLSX (Excel Sheet) using another python code that does this conversion. Now, this excel sheet is passed as an argument with TensorFlow training python code. After the training is successfully done, TensorFlow opens up a webpage with all the graphs related to our training data. From that webpage, .tflite file can be downloaded that holds the neural network graph and class label file to determine the class.

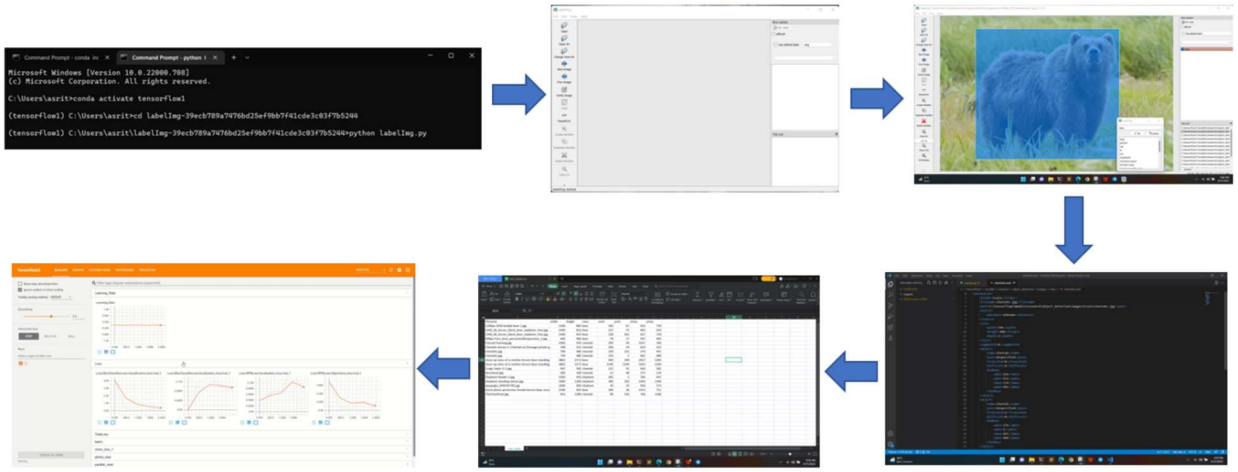

Fig. 4. Sample flow chart of training datasets

3.6 Algorithm

Below is the algorithm to detect animals and human subjects using a single-shot multi-box detector machine learning model using a camera live feed stream.

1. Import required packages
2. Create Videostream class to handle streaming
    - 2.1 Initialization method
        - 2.1.1 Initialize USB Camera/ Pi Camera/ Webcam and the camera image stream
        - 2.1.2 Read the first frame from the stream
    - 2.2 Start method
        - 2.2.1 Start the thread that reads frames from the video stream
    - 2.3 Update method
        - 2.3.1 Keep looking until the thread is stopped
            - 2.3.1.1 If a thread is stopped
                - 2.3.1.1.1 Close camera resources
            - 2.3.1.2 Grab the next frame from the video stream

-

2.4 Read method

2.4.1 Returns the most recent frame

2.5 Stop method

2.5.1 Indicate the camera and thread should be stopped

3. Define and parse input arguments
4. Import Google's TensorFlow libraries and use the tflite_runtime interpreter
5. Get the paths from input arguments
6. Load the label map
7. Load the TensorFlow Lite SSD model
8. Get the model details
9. Use TensorFlow 2 if available else use version 1
10. Initialize frame rate calculation
11. Initialize video stream class
12. While capturing data continuously is true

12.1 start the timer

12.2 Grab the frame from a video stream

12.3 Acquire frame and resize to expected shape

12.4 Normalize pixel values if the model is Non-quantized

12.5 Perform detection by running the model with an image as input

12.6 Retrieve detection results

12.7 Loop all detections and draw detection box

12.7.1 Get bounding box coordinates to draw the box

12.7.2 Draw the label

12.8 Draw the frame rate count in corner of the frame

12.9 All the results have been drawn on the frame and displayed now

12.10 Calculate the frame rate

13. Clean up the data and memory

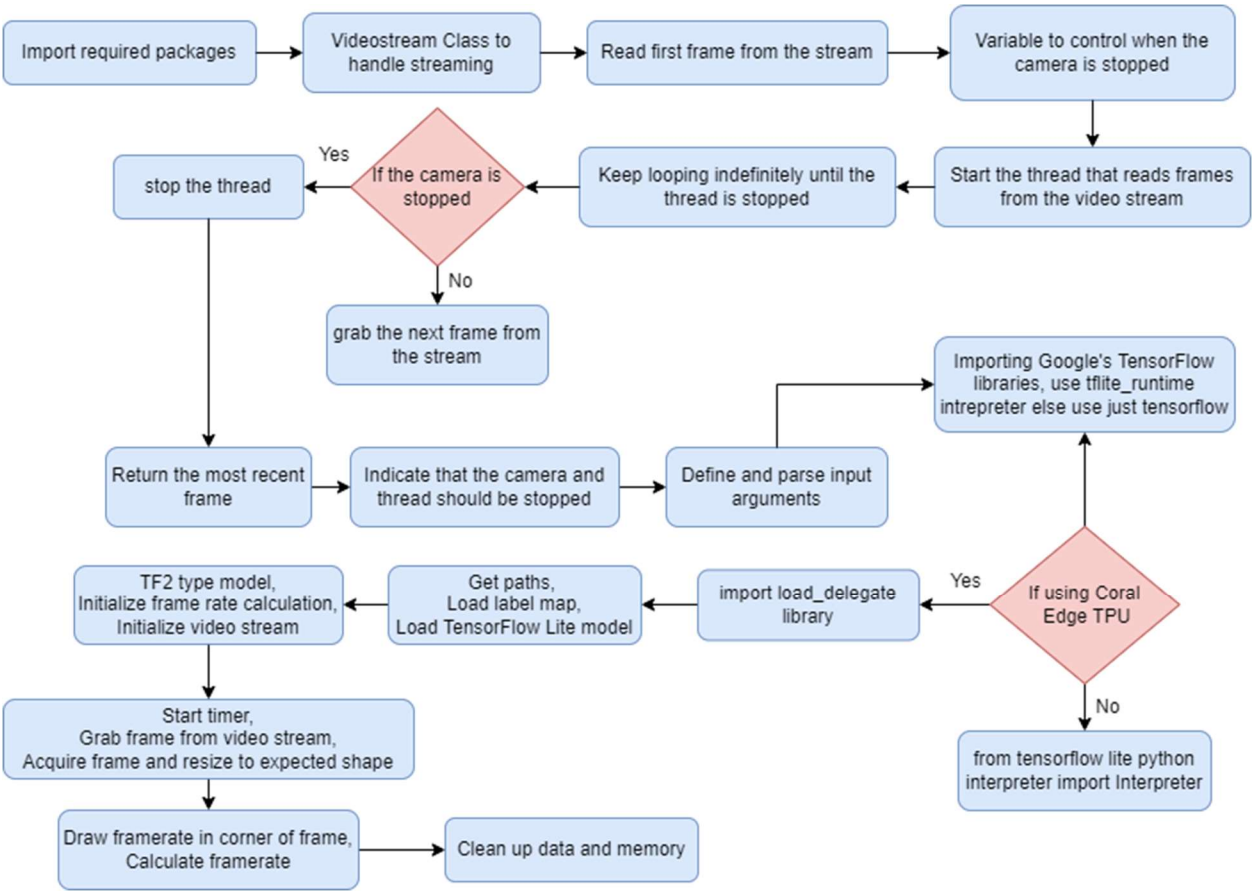

Fig. 5. Flowchart of Process Flow

Using tflite_interpreter 's set_tensor() method will start the actual detection by running the Single-Shot Multi-Box Detection (SSD) machine learning model. This is already available TensorFlow Lite API to detect the objects using the custom-made .tflite file which holds the neural network graph data and the label map which holds the class names.

Tflite_interpreter's invoke() method will run the detection model on the frameset using the set_tensor() method initially. Finally, tflite_interpreter's get_tensor() will get the detection results such as boundary boxes, classes, and scores (accuracy) of the detected object(s).

Using computer vision, break down the live video capture using the camera into image frames sending it into the TensorFlow API libraries for object detection and using the boundary box data to draw the boxes and label data to the label on the video frame. Also, showing the frame rates.

CHAPTER 4
SOURCE CODE

4.1 TFLite_Camera.py

To run the code:

>py TFLite_Camera.py –modeldir=<give model directory here (folder directory)>

</> CODE

```
# Object (Animal) Detection using TensorFlow Custom Trained Classifier in Camera
Capture
# Import required packages
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util

# Videostream Class to handle streaming
class VideoStream:
    """Camera object that controls video streaming from the Picamera"""
    def __init__(self,resolution=(640,480),framerate=30):
        # Initialize the PiCamera and the camera image stream
        self.stream = cv2.VideoCapture(0)
        ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        ret = self.stream.set(3,resolution[0])
        ret = self.stream.set(4,resolution[1])

        # Read first frame from the stream
        (self.grabbed, self.frame) = self.stream.read()

# Variable to control when the camera is stopped
        self.stopped = False

    def start(self):
# Start the thread that reads frames from the video stream
        Thread(target=self.update,args=()).start()
        return self

    def update(self):
        # Keep looping indefinitely until the thread is stopped
        while True:
            # If the camera is stopped, stop the thread
            if self.stopped:
                # Close camera resources
                self.stream.release()
```

```
                return

            # Otherwise, grab the next frame from the stream
            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
# Return the most recent frame
        return self.frame

    def stop(self):
# Indicate that the camera and thread should be stopped
        self.stopped = True

#Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than
labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying
detected objects',
                    default=0.5)
parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the
webcam does not support the resolution entered, errors may occur.',
                    default='1280x720')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up
detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH = int(resW), int(resH)
use_TPU = args.edgetpu

# Importing Google's TensorFlow libraries
# Use tflite_runtime intrepreter else use just tensorflow
# If using Coral Edge TPU then import load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
```

```
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

if use_TPU:
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get paths
CWD_PATH = os.getcwd()
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]
if labels[0] == '???':
    del(labels[0])

# Load TensorFlow Lite model
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,
                              experimental_delegates=[load_delegate('libedgetpu.so.1.
0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)

interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean = 127.5
input_std = 127.5

outname = output_details[0]['name']

# TF2 type model
if ('StatefulPartitionedCall' in outname):
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # TF1 type model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()
```

```
# Initialize video stream
videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
time.sleep(1)

#for frame1 in camera.capture_continuous(rawCapture,
format="bgr",use_video_port=True):
while True:

    # Start timer (for calculating frame rate)
    t1 = cv2.getTickCount()

    # Grab frame from video stream
    frame1 = videostream.read()

    # Acquire frame and resize to expected shape [1xHxWx3]
    frame = frame1.copy()
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_resized = cv2.resize(frame_rgb, (width, height))
    input_data = np.expand_dims(frame_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-
quantized)
    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding
box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class
index of detected objects
    scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] #
Confidence of detected objects

    # Loop over all detections and draw detection box if confidence is above minimum
threshold
    for i in range(len(scores)):
        if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image
dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

            # Draw label
```

```
            object_name = labels[int(classes[i])] # Look up object name from "labels"
array using class index
            label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example:
'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX,
0.7, 2) # Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label
too close to top of window
            cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10),
(xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw
white box to put label text in
            cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 0), 2) # Draw label text

    # Draw framerate in corner of frame
    cv2.putText(frame,'FPS:
{0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),2,cv2
.LINE_AA)

    # All the results have been drawn on the frame, so it's time to display it.
    cv2.imshow('Object detector', frame)

    # Calculate framerate
    t2 = cv2.getTickCount()
    time1 = (t2-t1)/freq
    frame_rate_calc= 1/time1

    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break

# Clean up data and memory
cv2.destroyAllWindows()
videostream.stop()
```

4.2 TFLite_Video.py

To run the code:

>py TFLite_Video.py –modeldir=<Provide model directory> --video=<Provide video directory>

</>CODE

```
# Object (Animal) Detection using TensorFlow Custom Trained Classifier in Videos
# Import required packages
import os
import argparse
import cv2
import numpy as np
import sys
import importlib.util
```

```
#Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than
labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying
detected objects',
                    default=0.5)
parser.add_argument('--video', help='Name of the video file',
                    default='test.mp4')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up
detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
VIDEO_NAME = args.video
min_conf_threshold = float(args.threshold)
use_TPU = args.edgetpu

# Importing Google's TensorFlow libraries
# Use tflite_runtime intrepreter else use just tensorflow
# If using Coral Edge TPU then import load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

if use_TPU:
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get paths
CWD_PATH = os.getcwd()
VIDEO_PATH = os.path.join(CWD_PATH,VIDEO_NAME)
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]
```

```
if labels[0] == '???':
    del(labels[0])

# Load TensorFlow Lite model
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,
                              experimental_delegates=[load_delegate('libedgetpu.so.1.
0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)

interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean = 127.5
input_std = 127.5

outname = output_details[0]['name']

# TF2 type model
if ('StatefulPartitionedCall' in outname):
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # TF1 type model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2

# Open Video File
video = cv2.VideoCapture(VIDEO_PATH)
imW = video.get(cv2.CAP_PROP_FRAME_WIDTH)
imH = video.get(cv2.CAP_PROP_FRAME_HEIGHT)
while(video.isOpened()):
    # Acquire frame and resize to expected shape [1xHxWx3]
    ret, frame = video.read()
    if not ret:
      print('Reached the end of the video!')
      break
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_resized = cv2.resize(frame_rgb, (width, height))
    input_data = np.expand_dims(frame_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-
quantized)
    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as input
```

```
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding
box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class
index of detected objects
    scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] #
Confidence of detected objects

    # Loop over all detections and draw detection box if confidence is above minimum
threshold
    for i in range(len(scores)):
        if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image
dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 4)

            # Draw label
            object_name = labels[int(classes[i])] # Look up object name from "labels"
array using class index
            label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example:
'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX,
0.7, 2) # Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label
too close to top of window
            cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10),
(xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw
white box to put label text in
            cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 0), 2) # Draw label text

    # All the results have been drawn on the frame, so it's time to display it.
    cv2.imshow('Object detector', frame)

    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break

# Clean up data and memory
video.release()
cv2.destroyAllWindows()
```

4.3 TFLite_Image.py

To run the code:

>py TFLite_Image.py –modeldir=<Provide model directory> --Image=<Provide folder directory containing only images>

</>CODE

```
# Object (Animal) Detection using TensorFlow Custom Trained Classifier in Images
# Import required packages
import os
import argparse
import cv2
import numpy as np
import sys
import glob
import importlib.util

#Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than
labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying
detected objects',
                    default=0.5)
parser.add_argument('--image', help='Name of the single image to perform detection
on. To run detection on multiple images, use --imagedir',
                    default=None)
parser.add_argument('--imagedir', help='Name of the folder containing images to
perform detection on. Folder must contain only images.',
                    default=None)
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up
detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
use_TPU = args.edgetpu

# Parse image name and directory
IM_NAME = args.image
IM_DIR = args.imagedir
if (IM_NAME and IM_DIR):
```

```
    print('Error! Please only use the --image argument or the --imagedir argument,
not both. Issue "python TFLite_detection_image.py -h" for help.')
    sys.exit()
if (not IM_NAME and not IM_DIR):
    IM_NAME = 'test1.jpg'

# Importing Google's TensorFlow libraries
# Use tflite_runtime intrepreter else use just tensorflow
# If using Coral Edge TPU then import load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate
if use_TPU:
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get paths
CWD_PATH = os.getcwd()
if IM_DIR:
    PATH_TO_IMAGES = os.path.join(CWD_PATH,IM_DIR)
    images = glob.glob(PATH_TO_IMAGES + '/*')

elif IM_NAME:
    PATH_TO_IMAGES = os.path.join(CWD_PATH,IM_NAME)
    images = glob.glob(PATH_TO_IMAGES)
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]
if labels[0] == '???':
    del(labels[0])

# Load TensorFlow Lite model
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,
                              experimental_delegates=[load_delegate('libedgetpu.so.1.
0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)

interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
```

```
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean = 127.5
input_std = 127.5

outname = output_details[0]['name']

# TF2 type model
if ('StatefulPartitionedCall' in outname):
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # TF1 type model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2

# Loop over every image and perform detection
for image_path in images:

    # Load image and resize to expected shape [1xHxWx3]
    image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    imH, imW, _ = image.shape
    image_resized = cv2.resize(image_rgb, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-
quantized)
    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding
box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class
index of detected objects
    scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] #
Confidence of detected objects

    # Loop over all detections and draw detection box if confidence is above minimum
threshold
    for i in range(len(scores)):
        if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image
dimensions, need to force them to be within image using max() and min()
```

```
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

            # Draw label
            object_name = labels[int(classes[i])] # Look up object name from "labels"
array using class index
            label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example:
'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX,
0.7, 2) # Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label
too close to top of window
            cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10),
(xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw
white box to put label text in
            cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 0), 2) # Draw label text

    # All the results have been drawn on the image, now display the image
    cv2.imshow('Object detector', image)

    # Press any key to continue to next image, or press 'q' to quit
    if cv2.waitKey(0) == ord('q'):
        break

# Clean up data and memory
cv2.destroyAllWindows()
```

4.4 Important Instructions

Arguments needed while running the code:

1. --modeldir	Provide the folder directory where .tflite and label map files are available.
2. --video	Provide the video file directory to detect objects.
3. --image	Provide a folder directory where all images are present.
4. --imagedir	Provide an image directory to detect objects.
5. --edgetpu	Provide Edge TPU directory if available.

CHAPTER 5
SNAPSHOTS

5.1 Animal & Person Detection in Image Files

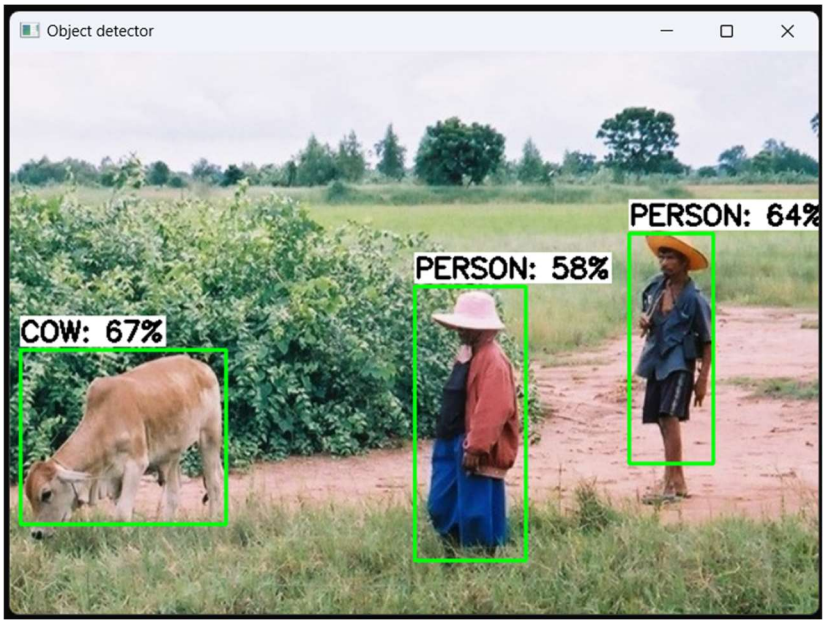

Fig. 6. Cow and Person detection in an Image

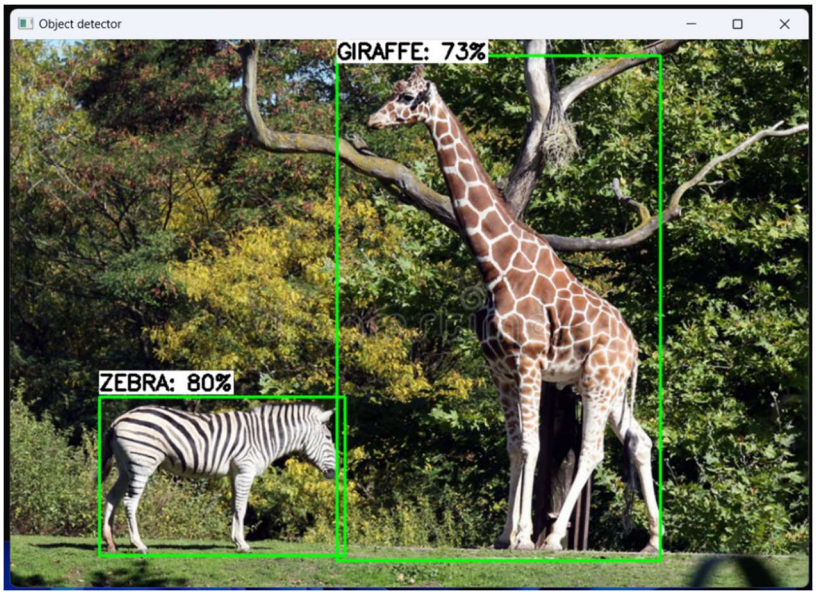

Fig. 7. Zebra and Giraffe detection in an Image

Fig. 8. Bear detection in an Image

5.2 Animal Detection in Video Files

Fig. 9. Cow detection in a Video

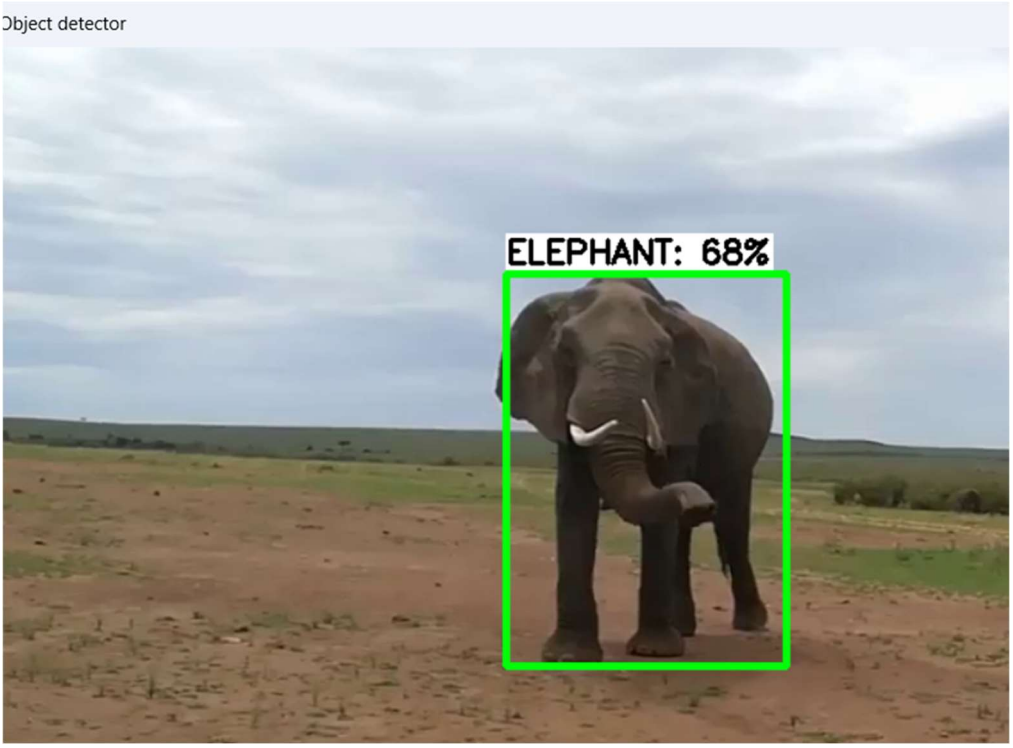

Fig. 10. Elephant detection in a Video

5.3 Animal or Person detection using Camera

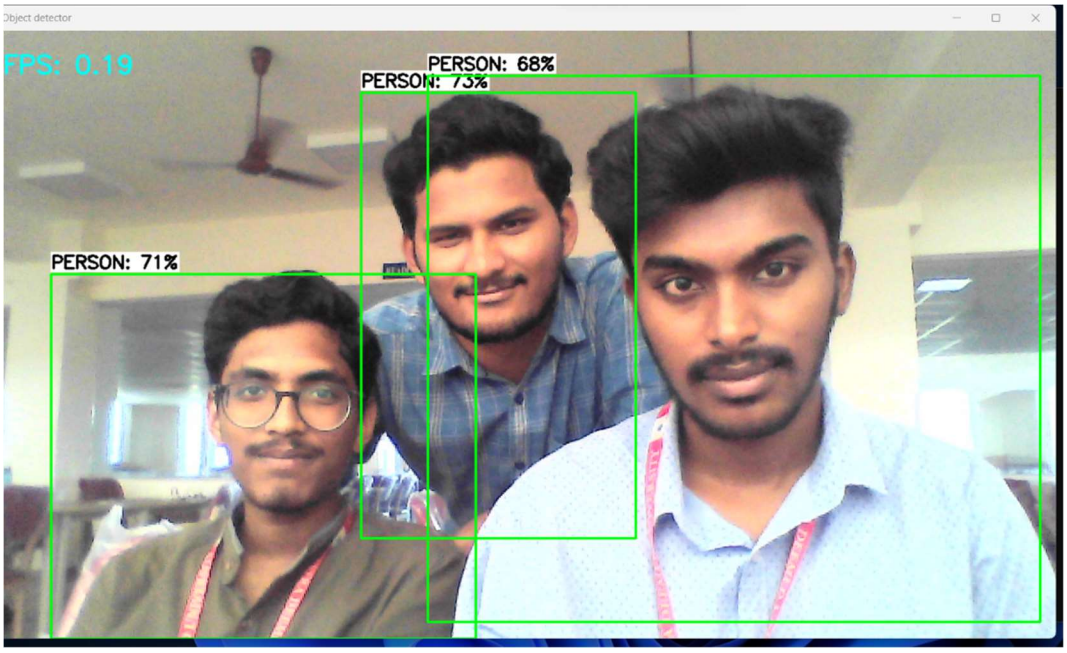

Fig. 11. Person detection using Camera (Live Feed)

5.4 Result Analysis

Single-Shot Multi-Box Detector machine learning model ensures faster object detection but compromises accuracy, it is clearly observed in this work. The accuracy of the object detected is not very low. It is between 65% to 80% in most of the cases and between 55% to 65% in complex scenario cases. This is clearly enough accuracy to immediately detect animal intrusion quickly and let the necessary people know about the intrusion.

Object Detected	**Approximate Accuracy**	**Detection Speed**
Cow	65% to 80%	Very Fast
Zebra	72% to 85%	Very Fast
Giraffe	70% to 85%	Very Fast
Bear	70% to 80%	Very Fast
Elephant	60% to 75%	Very Fast
Person	58% to 75%	Very Fast
Other Animals	60% to 80%	Very Fast

Table. 1. Accuracy and Detection Information of Objects Detected

CHAPTER 6
CONCLUSION AND FUTURE PLANS

Agricultural yield protection, avoiding accidents on national highways, and injuries in remote areas like villages & base stations because of animal intrusion will be detected beforehand with the use of the Single-Shot Multi-Box Detector machine learning model with the use of Google's TensorFlow APIs. SSD algorithm performance is good to detect animals and persons quickly in real-time using the camera's live feed but the accuracy of the detected object will not be near 100%. Compared to other types of a convolutional neural network like R-CNN which mainly focuses on accuracy rather than speed of detection, it will not be so useful for IoT or other mobile ARM-based devices for this application as processor power in these small devices will be low. SSD has the perfect qualities to work with these types of devices for faster detection and identifying objects without any miss in any frame. Hence, the animal intrusion is prevented with the proposed work in this paper.

FUTURE PLANS

An app-based prototype can be developed such that it will be user-friendly and can be more mobile so that it can be taken anywhere for the detection of animal intrusion.

CHAPTER 7
REFERENCES

[1] Balakrishna, K., Mohammed, F., Ullas, C. R., Hema, C. M., & Sonakshi, S. K. (2021). Application of IOT and machine learning in crop protection against animal intrusion. Global Transitions Proceedings, 2(2), 169-174.

[2] Iniyaa, K. K., Divya, J. K., Devdharshini, S., & Sangeethapriya, R. (2021). Crop Protection from Animals Using Deep Learning. International Journal of Progressive Research in Science and Engineering, 2(3), 41-44.

[3] Gavrilescu, R., Zet, C., Foșalău, C., Skoczylas, M., & Cotovanu, D. (2018, October). Faster R-CNN: an approach to real-time object detection. In 2018 International Conference and Exposition on Electrical And Power Engineering (EPE) (pp. 0165-0168).

[4] Iniyaa, K. K., Divya, J. K., Devdharshini, S., & Sangeethapriya, R. (2021). Crop Protection from Animals Using Deep Learning. International Journal of Progressive Research in Science and Engineering, 2(3), 41-44. This paper has works related to animal detection using a deep learning model which is an advanced sub-branch of machine learning model

[5] D. Renard, D. Tilman, National food production stabilized by crop diversity, Nature 571 (2019) 257–260, doi:10.1038/s41586-019-1316-y. This paper has works related to crop production stability using crop diversity with machine learning models to determine health.

[6] H.S. Kanyal, M. Goel, A.S. Tomar, H.K. Yadav, K. Singh, Object recognition and security improvement by enhancing the features of CCTV, in: 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), 2020, pp. 245–248, doi:10.1109/SMART50582.2020.9337065. This paper has work related to object detection using machine learning algorithms with a camera. Live feed is captured using a camera (CCTV) and that is used as data to detect objects

[7] R R., P S., A cohesive farm monitoring and wild animal warning prototype system using IoT and machine learning, in: 2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE), 2020, pp. 472–476, doi:10.1109/ICSTCEE49637.2020.9277267. This paper has works related to alerting the user by a warning when an animal is detected. Detection is done using machine learning models and IoT devices are used to detect and capture data required for detection.

[8] X. Miao, X. Liu, J. Chen, S. Zhuang, J. Fan, H. Jiang, Insulator detection in aerial images for transmission line inspection using single shot multibox detector, IEEE Access 7 (2019) 9945–9956, doi:10.1109/ACCESS.2019.2891123. This paper has works related to a single-shot multi-box detector machine learning model to detect objects in aerial images to inspect transmission lines. This also uses the camera as a live capture feed as a data stream

[9] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, R. Sharma, CrypTFlow: secure tensorflow inference, in: 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 336–353, doi:10.1109/SP40000.2020.00092. This paper has works related to using machine learning models with the help of TensorFlow APIs provided by Google Inc.