CSE4060 - Intelligent Robots and Drone Technology

Project Report

Object Detection on Endangered Animals

By

19BAI1118	Ashwin U Iyer
19BAI1129	Hitesh Goyal
19BAI1141	Sila Shivanandan

B. Tech Computer Science and Engineering

Submitted to

Dr. Sathian D

School of Computer Science and Engineering

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

July 2022

DECLARATION

I hereby declare that the report titled **“Object Detection on Endangered Animals”** submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr. Sathian D**, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai.

ABSTRACT

Due to the development of civilization, animals are losing their natural habitats. This has caused major disruption in food chains and entire ecosystems.

Over 7,000 species of animals around the world are considered endangered. Thus, monitoring the status of wildlife has never been more necessary or more difficult.

We choose to use drones and object detection to detect and track endangered wildlife in an effort to better protect and prevent them.

CONTENTS

1. Dataset

Animal Detection Images: Collection of wild animal species with annotations from Kaggle. We used this dataset to create a proof of concept model showing that given images and annotations, we can use an object detection model which can recognize animals.

The dataset contains about 80 different animal species. There are multiple train and test images for all of these classes. The dataset comes along with annotations which are normalized coordinates of bounding box corners which can tell us where the animal is.

2. Literature Survey

A) Object Detection with Deep Learning: A Review

Zhong-Qiu Zhao, *Member, IEEE,* Peng Zheng, Shou-tao Xu, and Xindong Wu, *Fellow, IEEE*

Due to object detection's close relationship with video analysis and image understanding, it has attracted much research attention in recent years. Traditional object detection methods are built on handcrafted features and shallow trainable architectures. Their performance easily stagnates by constructing complex ensembles which combine multiple low-level image features with high-level context from object detectors and scene classifiers. With the rapid development in deep learning, more powerful tools, which are able to learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures. These models behave differently in network architecture, training strategy and optimization function, etc. In this paper, we provide a review on deep learning based object detection frameworks. Our review begins with a brief introduction on the history of deep learning and its representative tool, namely Convolutional Neural Network (CNN). Then we focus on typical generic object detection architectures along with some modifications and useful tricks to improve detection performance further. As distinct specific detection tasks exhibit different characteristics, we also briefly survey several specific tasks, including salient object detection, face detection and pedestrian detection. Experimental analyses are also provided to compare various methods and draw some meaningful conclusions. Finally, several promising directions and tasks are provided to serve as guidelines for future work in both object detection and relevant neural network based learning systems.

B) MobileNetV2: Inverted Residuals and Linear Bottlenecks

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen

In this paper we describe a new mobile architecture, MobileNetV2, that improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. We also describe efficient ways of applying these mobile models to object detection in a novel framework we call SSDLite. Additionally, we demonstrate how to build mobile semantic segmentation models through a reduced form of DeepLabv3 which we call Mobile DeepLabv3.

The MobileNetV2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer. Additionally, we find that it is important to remove non-linearities in the narrow layers in order to maintain representational power. We demonstrate that this improves performance and provide an intuition that led to this design. Finally, our approach allows decoupling of the input/output domains from the expressiveness of the transformation, which provides a convenient framework for further analysis. We measure our performance on Imagenet classification, COCO object detection, VOC image segmentation. We evaluate the trade-offs between accuracy, and number of operations measured by multiply-adds (MAdd), as well as the number of parameters

C) Deep Learning Methods for Animal Recognition and Tracking to Detect Intrusions

Ashwini V. Sayagavi, Sudarshan Tsb, Prashanth C Ravoor

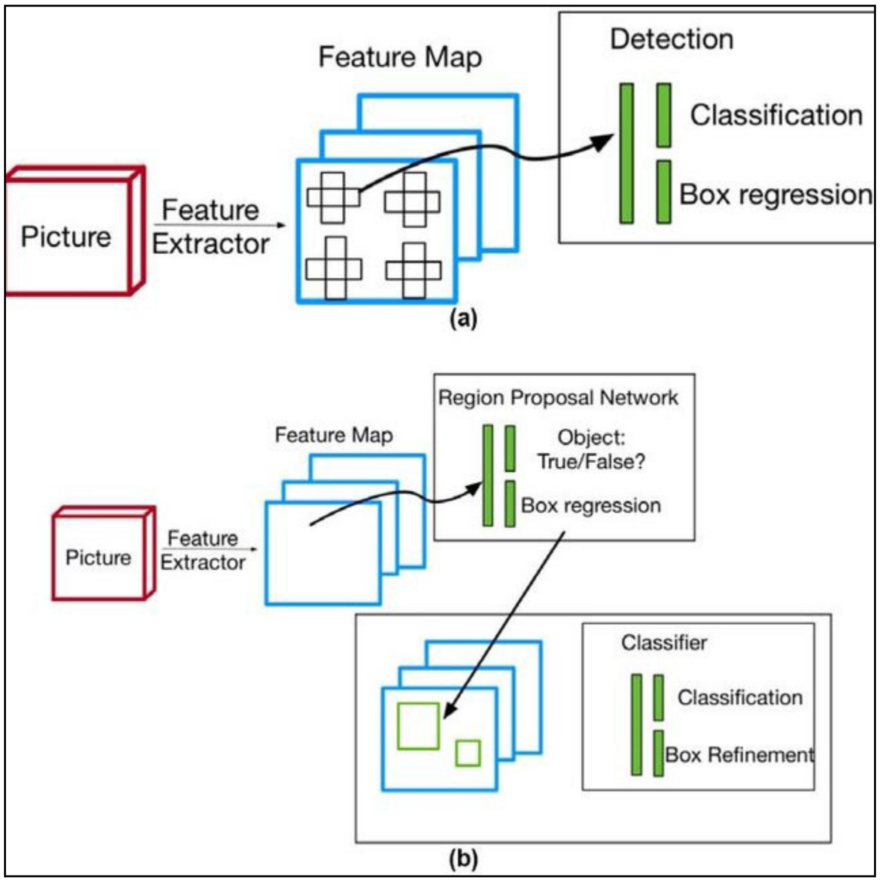
Over the last few years, there has been a steady rise in number of reported human–animal conflicts. While there are several reasons for increase in such conflicts, foremost among them is the reduction in forest cover. Animals stray close to human settlements in search of food, and often end up raiding crops or preying on cattle. There are at times human causalities as well. Proficient, reliable, and autonomous monitoring of human settlements bordering forest areas can help reduce such animal–human conflicts. A broad range of techniques in computer vision and deep-learning has shown enormous potential to solve such problems. In this paper, a novel, efficient, and reliable system is presented which automatically detects wild-animals using computer vision. The proposed method uses the YOLO object detection model to ascertain presence of wild animals in images. The model is fine-tuned for identifying six different entities—humans, and five different types of animals (elephant, zebra, giraffe, lion, and cheetah). Once detected, the animal is tracked using CSRT to determine its intentions, and based on the perceived information, notifications are sent to alert the concerned authorities. The design of a prototype for the proposed solution is also described, which uses Raspberry Pi devices equipped with cameras. The proposed method achieves an accuracy of 98.8% and 99.8% to detect animals and humans, respectively.

3.About Model

Object detection is a computer vision technology related to image processing that deals with detecting instances of semantic objects of a certain class in digital images and videos. With the advent of deep neural networks, object detection has taken the center stage in the development of computer vision with many models developed such as R-CNN (Regional Convolutional Neural Network) and it's variant (Faster-RCNN), Single Shot Detectors (SSD) models as well as the famous You Only Look Once (YOLO) models and it’s many versions.

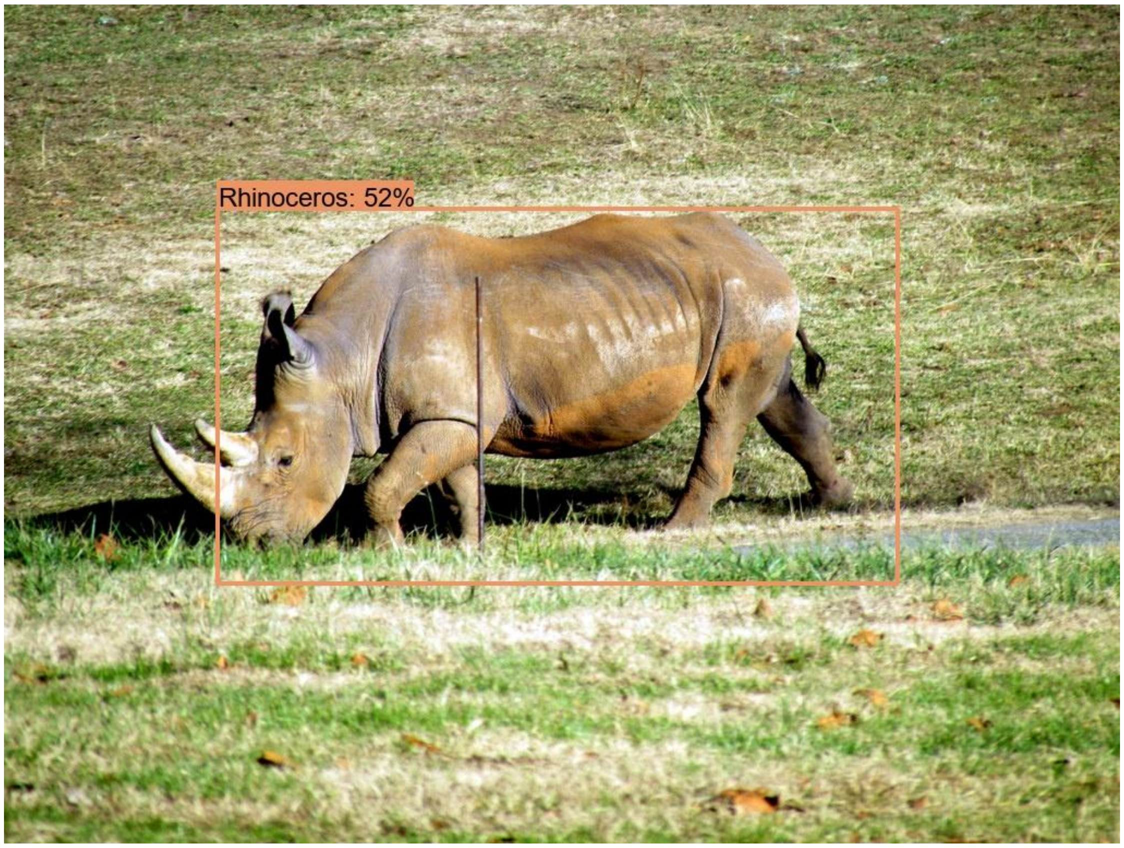
Typically object detection models are categorized into two major architectural types: one (single) stage object detectors such as YOLO and SSD and two (dual) stage object detectors such as R-CNN. The major difference between the two is that in the two-stage object detection models, the region of interest is first determined and the detection is then performed only on the region of interest. This implies that the two-stage object detection models are generally more accurate than the one-stage ones but require more computational resources and are slower.

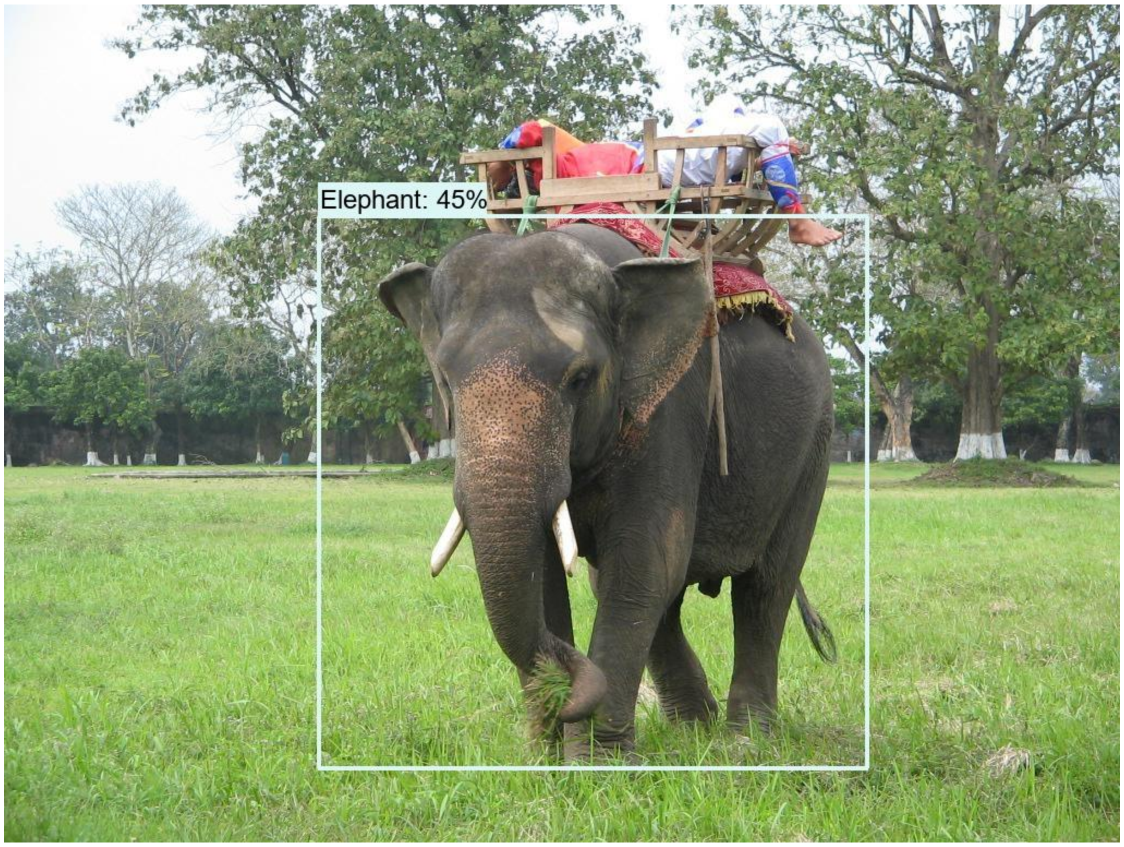
The below figure shows a representation of the two types of object detection models with (a) being one-stage detection and (b) being a two-stage detection model.

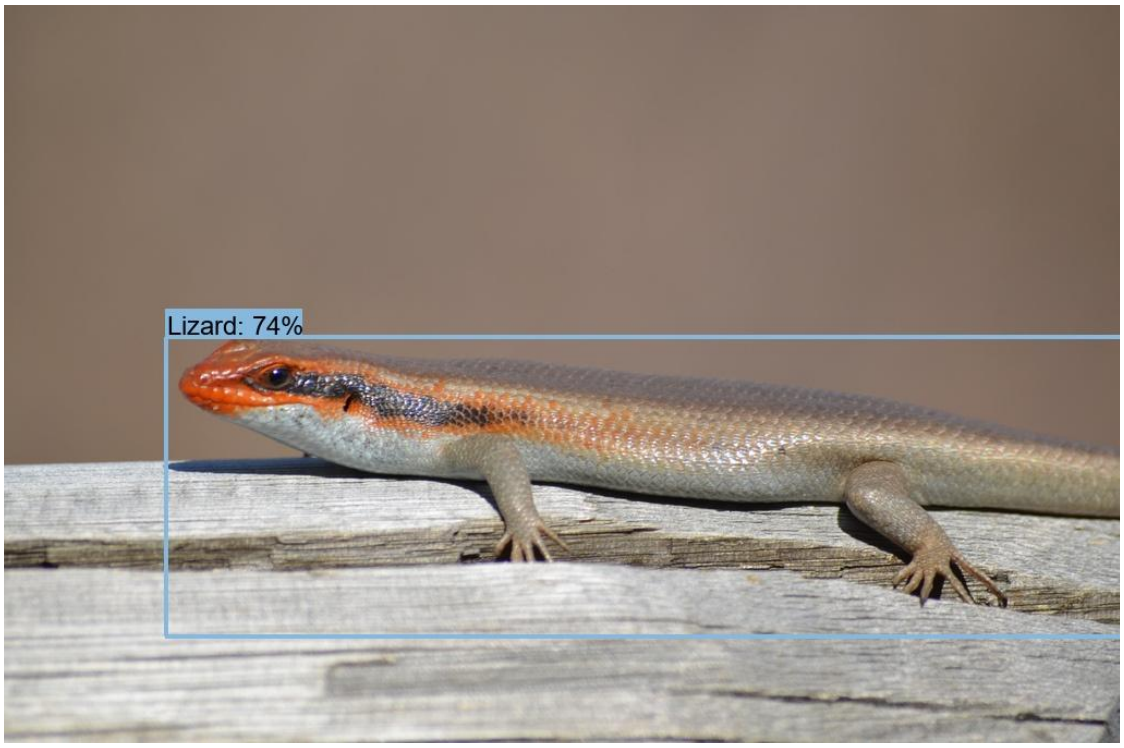


4. Results

The trained SSD MobileNet V2 model seems to perform the task of object detection relatively well, reaching a localization loss of 0.502 on validation. It manages to provide the bounding boxes with a reasonable degree of confidence, ~40% for each sample. The threshold for drawing the bounding boxes has been set at 30%. Below are a few sample outputs:




Elephant: 45%


Lizard: 74%

5. REFERENCES

- https://www.kaggle.com/datasets/antoreepjana/animals-detection-images-dataset
- https://www.v7labs.com/blog/object-detection-guide#h4
- https://vidishmehta204.medium.com/object-detection-using-ssd-mobilenet-v2-7ff3543d738d

6. Code

Train.py

```
import os
import object_detection

CUSTOM_MODEL_NAME = "my_ssd_mobnet"
PRETRAINED_MODEL_NAME                                                          =
"ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8"
PRETRAINED_MODEL_URL                                                           =
"http://download.tensorflow.org/models/object_detection/tf2/202007
11/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz"
TF_RECORD_SCRIPT_NAME = "generate_tfrecord.py"
LABEL_MAP_NAME = "label_map.pbtxt"


os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
paths = {
    "WORKSPACE_PATH": os.path.join("Tensorflow", "workspace"),
    "SCRIPTS_PATH": os.path.join("Tensorflow", "scripts"),
    "APIMODEL_PATH": os.path.join("Tensorflow", "models"),
       "ANNOTATION_PATH":  os.path.join("Tensorflow",  "workspace",
"annotations"),
          "IMAGE_PATH":   os.path.join("Tensorflow",   "workspace",
"images"),
          "MODEL_PATH":   os.path.join("Tensorflow",   "workspace",
"models"),
    "PRETRAINED_MODEL_PATH": os.path.join(
        "Tensorflow", "workspace", "pre-trained-models"
    ),
    "CHECKPOINT_PATH": os.path.join(
        "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME
    ),
    "OUTPUT_PATH": os.path.join(
          "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"export"
    ),
    "TFJS_PATH": os.path.join(
            "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"tfjsexport"
    ),
    "TFLITE_PATH": os.path.join(
            "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"tfliteexport"
    ),
    "PROTOC_PATH": os.path.join("Tensorflow", "protoc"),
}

files = {
    "PIPELINE_CONFIG": os.path.join(
            "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"pipeline.config"
    ),
           "TF_RECORD_SCRIPT":  os.path.join(paths["SCRIPTS_PATH"],
TF_RECORD_SCRIPT_NAME),
               "LABELMAP":   os.path.join(paths["ANNOTATION_PATH"],
LABEL_MAP_NAME),
}

TRAINING_SCRIPT = os.path.join(
         paths["APIMODEL_PATH"],  "research",  "object_detection",
"model_main_tf2.py"
)

NUM_STEPS = 10000
command = (
            "python  {}  --model_dir={}  --pipeline_config_path={}
--num_train_steps={}".format(
                       TRAINING_SCRIPT,  paths["CHECKPOINT_PATH"],
files["PIPELINE_CONFIG"], NUM_STEPS
    )
)
print(command)
os.system(command)

os.system(f"tar -czf models.tar.gz {paths['CHECKPOINT_PATH']}")
```

Predict.py

```
import os
from black import out

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
import tensorflow as tf
from object_detection.utils import label_map_util
from   object_detection.utils   import   visualization_utils   as
viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
import cv2
import numpy as np
import matplotlib.pyplot as plt

CUSTOM_MODEL_NAME = "my_ssd_mobnet"
PRETRAINED_MODEL_NAME                                          =
"ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8"
PRETRAINED_MODEL_URL                                           =
"http://download.tensorflow.org/models/object_detection/tf2/202007
11/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz"
TF_RECORD_SCRIPT_NAME = "generate_tfrecord.py"
LABEL_MAP_NAME = "label_map.pbtxt"


os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
paths = {
    "WORKSPACE_PATH": os.path.join("Tensorflow", "workspace"),
    "SCRIPTS_PATH": os.path.join("Tensorflow", "scripts"),
    "APIMODEL_PATH": os.path.join("Tensorflow", "models"),
       "ANNOTATION_PATH":  os.path.join("Tensorflow",  "workspace",
"annotations"),
          "IMAGE_PATH":   os.path.join("Tensorflow",   "workspace",
"images"),
          "MODEL_PATH":   os.path.join("Tensorflow",   "workspace",
"models"),
    "PRETRAINED_MODEL_PATH": os.path.join(
        "Tensorflow", "workspace", "pre-trained-models"
    ),
    "CHECKPOINT_PATH": os.path.join(
        "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME
    ),
    "OUTPUT_PATH": os.path.join(
           "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"export"
    ),
    "TFJS_PATH": os.path.join(
           "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"tfjsexport"
    ),
    "TFLITE_PATH": os.path.join(
           "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"tfliteexport"
    ),
    "PROTOC_PATH": os.path.join("Tensorflow", "protoc"),
}

files = {
    "PIPELINE_CONFIG": os.path.join(
           "Tensorflow", "workspace", "models", CUSTOM_MODEL_NAME,
"pipeline.config"
    ),
          "TF_RECORD_SCRIPT":  os.path.join(paths["SCRIPTS_PATH"],
TF_RECORD_SCRIPT_NAME),
              "LABELMAP":   os.path.join(paths["ANNOTATION_PATH"],
LABEL_MAP_NAME),
}

TRAINING_SCRIPT = os.path.join(
         paths["APIMODEL_PATH"],  "research",  "object_detection",
"model_main_tf2.py"
)

# Load pipeline config and build a detection model
configs                                                          =
config_util.get_configs_from_pipeline_file(files["PIPELINE_CONFIG"
])
```

```
detection_model                                                     =
model_builder.build(model_config=configs["model"],
is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths["CHECKPOINT_PATH"],
"ckpt-4")).expect_partial()


@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
        detections  =  detection_model.postprocess(prediction_dict,
shapes)
                                              print("Detections:",
detections["detection_multiclass_scores"])
    return detections


category_index                                                      =
label_map_util.create_category_index_from_labelmap(files["LABELMAP
"])


def get_coordinates(x, y, bounding_boxes, padding=0):
    ymin = max(round(bounding_boxes[0] * x) - padding, 0)
    xmin = max(round(bounding_boxes[1] * y) - padding, 0)
    ymax = max(round(bounding_boxes[2] * x) + padding, 0)
    xmax = max(round(bounding_boxes[3] * y) + padding, 0)

    return np.array([ymin, xmin, ymax, xmax])


def get_multiple_coordinates(x, y, arr_bb: np.array):
     return np.array([get_coordinates(x, y, i, padding=5) for i in
arr_bb])
```

```
def   output_detection_boxes_with_score(IMAGE_PATH,   num_boxes=8,
print_detections=False):
    img = cv2.imread(IMAGE_PATH)
    x, y, _ = img.shape
    image_np = np.array(img)
      input_tensor  =  tf.convert_to_tensor(np.expand_dims(image_np,
0), dtype=tf.float32)
    detections = detect_fn(input_tensor)
    num_detections = int(detections.pop("num_detections"))
    detections = {
          key: value[0, :num_detections].numpy() for key, value in
detections.items()
    }
    detections["num_detections"] = num_detections
    # detection_classes should be ints.
                            detections["detection_classes"]       =
detections["detection_classes"].astype(np.int64)
    detections["detection_multiclass_scores"] = detections[
        "detection_multiclass_scores"
    ].astype(np.float64)
    result = {
        "detection_boxes": get_multiple_coordinates(
            x, y, detections["detection_boxes"][:num_boxes]
        ),
                                                "detection_scores":
detections["detection_scores"][:num_boxes],
        "detection_classes": detections["detection_classes"],
        "detection_multiclass_scores": np.argmax(
            detections["detection_multiclass_scores"], axis=1
        ),
    }
    if print_detections:
        print(result)
    return result

def draw_predictions(IMAGE_PATH, result, threshold=0, padding=0):
    img = cv2.imread(IMAGE_PATH)
    # print(result)
                boxes,    score    =    result["detection_boxes"],
result["detection_scores"]
```

```
    for i in range(len(score)):
        if float(score[i]) < threshold:
            continue
        y, x, h, w = boxes[i]
        cv2.rectangle(
                img, (x - padding, y - padding), (w + padding, h +
padding), (0, 255, 0), 2
        )
        # print(x-padding,y-padding,w+padding,h+padding)
        return img

def   save_output_to_folder(IMAGE_PATH,   path_to_save,   results,
threshold=0.2, values=[]):
    # detections = output_detection_boxes_with_score(IMAGE_PATH)
                boxes,    score    =    results["detection_boxes"],
results["detection_scores"]
    img_name = IMAGE_PATH.split("\\")[-1]
    img = cv2.imread(IMAGE_PATH)
    s = path_to_save

    for cnt, num in enumerate(boxes):
        if threshold > score[cnt]:
            continue
        cropped_image = img[num[0] : num[2], num[1] : num[3]]
        try:
            if len(values) > 0:
                path_to_save = os.path.join(
                    path_to_save, (str(values[cnt]) if values[cnt]
>= 0 else "UNK")
                )
        except:
            continue

        cv2.imwrite(
                os.path.join(path_to_save, img_name + "_img_" +
str(cnt) + ".jpg"),
            cropped_image,
        )
        path_to_save = s
```

```
if __name__ == "__main__":
    for idx, img in enumerate(os.listdir("./data/test")):
        i = np.random.randint(0, len(os.listdir("./data/test")))
        input_img = os.listdir("./data/test")[i]
                                                    IMAGE_PATH     =
f"C:/Users/Ashwin/Projects/Object-Detection-on-Animals/data/test/{
input_img}"
        results = output_detection_boxes_with_score(
            IMAGE_PATH,
            num_boxes=1,
        )
        output_image = draw_predictions(IMAGE_PATH, results)
        # label_id_offset = 1
        # image_np = cv2.imread(IMAGE_PATH)

        # image_np_with_detections = image_np.copy()
                                                #     output_image     =
viz_utils.visualize_boxes_and_labels_on_image_array(
        #     image_np_with_detections,
        #     results["detection_boxes"],
        #     results["detection_classes"] + label_id_offset,
        #     results["detection_scores"],
        #     category_index,
        #     use_normalized_coordinates=False,
        #     max_boxes_to_draw=8,
        #     min_score_thresh=0.3,
        #     agnostic_mode=False,
        # )
        cv2.imwrite(

f"C:/Users/Ashwin/Projects/Object-Detection-on-Animals/output/pred
icted_{input_img}___{idx}.jpg",
            output_image,
        )
```

GITHUB LINK:

https://github.com/ashwiniyer176/Object-Detection-on-Endangered-Animals