OBJECT DETECTION

PROBLEM STATEMENT:

Given an image or video, develop a system that can identify and locate all instances of objects of specific classes.

This typically involves two key aspects:

Object Classification:

The system needs to correctly classify the object into a predefined category (e.g., car, person, dog).

Object Localization:

The system needs to draw a bounding box around each detected object, indicating its precise location within the image or video frame.

PROGRAM:

```
import tensorflow as tf
import tensorflow_hub as hub
from PIL import Image, ImageDraw, ImageFont, ImageColor, ImageOps
from io import BytesIO
from urllib.request import urlopen
import tempfile
import matplotlib.pyplot as plt
import numpy as np
import time

def display_image(image):
    fig = plt.figure(figsize=(20, 15))
    plt.grid(False)
    plt.imshow(image)

def download_and_resize_image(url, new_width=256, new_height=256,
display=False):
    _, filename = tempfile.mkstemp(suffix=".jpg")
    response = urlopen(url)
    image_data = response.read()
    image_data = BytesIO(image_data)
    pil_image = Image.open(image_data)
    pil_image = ImageOps.fit(pil_image, (new_width, new_height),
Image.LANCZOS)
    pil_image_rgb = pil_image.convert("RGB")
    pil_image_rgb.save(filename, format="JPEG", quality=90)
    print("Image downloaded to %s." % filename)
    if display:
        display_image(pil_image)
    return filename

def draw_bounding_box_on_image(image, ymin, xmin, ymax, xmax, color,
font, thickness=4, display_str_list=()):
    """Adds a bounding box to an image."""
    draw = ImageDraw.Draw(image)
    im_width, im_height = image.size
    (left, right, top, bottom) = (xmin * im_width, xmax * im_width,
ymin * im_height, ymax * im_height)
    draw.line([(left, top), (left, bottom), (right, bottom), (right,
top), (left, top)],
              width=thickness, fill=color)

    # If the total height of the display strings added to the top of
the bounding
    # box exceeds the top of the image, stack the strings below the
bounding box
    # instead of above.
    display_str_heights = [font.getbbox(ds)[3] for ds in
display_str_list]
    # Each display_str has a top and bottom margin of 0.05x.
    total_display_str_height = (1 + 2 * 0.05) *
sum(display_str_heights)

    if top > total_display_str_height:
        text_bottom = top
    else:
        text_bottom = top + total_display_str_height
    # Reverse list and print from bottom to top.
    for display_str in display_str_list[::-1]:
        bbox = font.getbbox(display_str)
        text_width, text_height = bbox[2], bbox[3]
        margin = np.ceil(0.05 * text_height)
        draw.rectangle([(left, text_bottom - text_height - 2 * margin),
                        (left + text_width, text_bottom)],
                       fill=color)
        draw.text((left + margin, text_bottom - text_height - margin),
                  display_str, fill="black", font=font)
        text_bottom -= text_height - 2 * margin

def draw_boxes(image, boxes, class_names, scores, max_boxes=10,
min_score=0.1):
    """Overlay labeled boxes on an image with formatted scores and
label names."""
    colors = list(ImageColor.colormap.values())
    try:
        font =
ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationSans
Narrow-Regular.ttf", 25)
    except IOError:
        print("Font not found, using default font.")
        font = ImageFont.load_default()

    for i in range(min(boxes.shape[0], max_boxes)):
        if scores[i] >= min_score:
            ymin, xmin, ymax, xmax = tuple(boxes[i])
            display_str = "{}:
{}%".format(class_names[i].decode("ascii"), int(100 * scores[i]))
            color = colors[hash(class_names[i]) % len(colors)]
            image_pil = Image.fromarray(np.uint8(image)).convert("RGB")
            draw_bounding_box_on_image(image_pil, ymin, xmin, ymax,
xmax, color, font, display_str_list=[display_str])
            np.copyto(image, np.array(image_pil))
    return image

# Load the pre-trained detector model
module_handle =
"https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2
/1"
detector = hub.load(module_handle).signatures['default']

def load_img(path):
    img = tf.io.read_file(path)
    img = tf.image.decode_jpeg(img, channels=3)
    return img

def run_detector(detector, path):
    img = load_img(path)
    converted_img = tf.image.convert_image_dtype(img,
tf.float32)[tf.newaxis, ...]
    start_time = time.time()
    result = detector(converted_img)
    end_time = time.time()
    result = {key: value.numpy() for key, value in result.items()}
    print("Found %d objects." % len(result["detection_scores"]))
    print("Inference time: ", end_time - start_time)
    image_with_boxes = draw_boxes(img.numpy(),
result["detection_boxes"], result["detection_class_entities"],
result["detection_scores"])
    display_image(image_with_boxes)

def detect_img(image_url):
    start_time = time.time()
    image_path = download_and_resize_image(image_url, 640, 480)
    run_detector(detector, image_path)
    end_time = time.time()
    print("Inference time:", end_time - start_time)

# Image URLs for detection
image_urls = [
    "https://upload.wikimedia.org/wikipedia/commons/6/60/Naxos_Taverna.jpg",
    "https://upload.wikimedia.org/wikipedia/commons/1/1b/The_Coleoptera_of_the_British_islands_%28Plate_125%29_%288592917784%29.jpg",
    "https://upload.wikimedia.org/wikipedia/commons/0/09/The_smaller_British_birds_%288053836633%29.jpg"
]

# Perform object detection on each image URL
for url in image_urls:
    detect_img(url)
```