```
"""Object detection

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1i4XC3Oz9FNNUT8n01WbL3hdJXMqPKq-Q
"""

import tensorflow as tf
print(tf.__version__)

import tensorflow_hub as hub

import matplotlib.pyplot as plt
import tempfile
from six.moves.urllib.request import urlopen
from six import BytesIO

import numpy as np
from PIL import Image
from PIL import ImageColor
from PIL import ImageDraw
from PIL import ImageFont
from PIL import ImageOps

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.ANTIALIAS)
  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.getsize(ds)[1] 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]:
    text_width, text_height = font.getsize(display_str)
    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/LiberationSansNarrow-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

module_handle="https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1"
detector = hub.load(module_handle).signatures['default']

image_url="https://th.bing.com/th/id/OIP.XKhnoT3WTBVHMw3IfpEnZQHaE7?w=800&h=533&rs=1&pid=ImgDetMain"
downloaded_image_path=download_and_resize_image(image_url,1280,1000,True)

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)

run_detector(detector, downloaded_image_path)
```