Program wykrywa obiekty na obrazie, oraz wypisuje czym one są oraz jaka jest szansa, że program co do danego obiektu się nie pomylił.

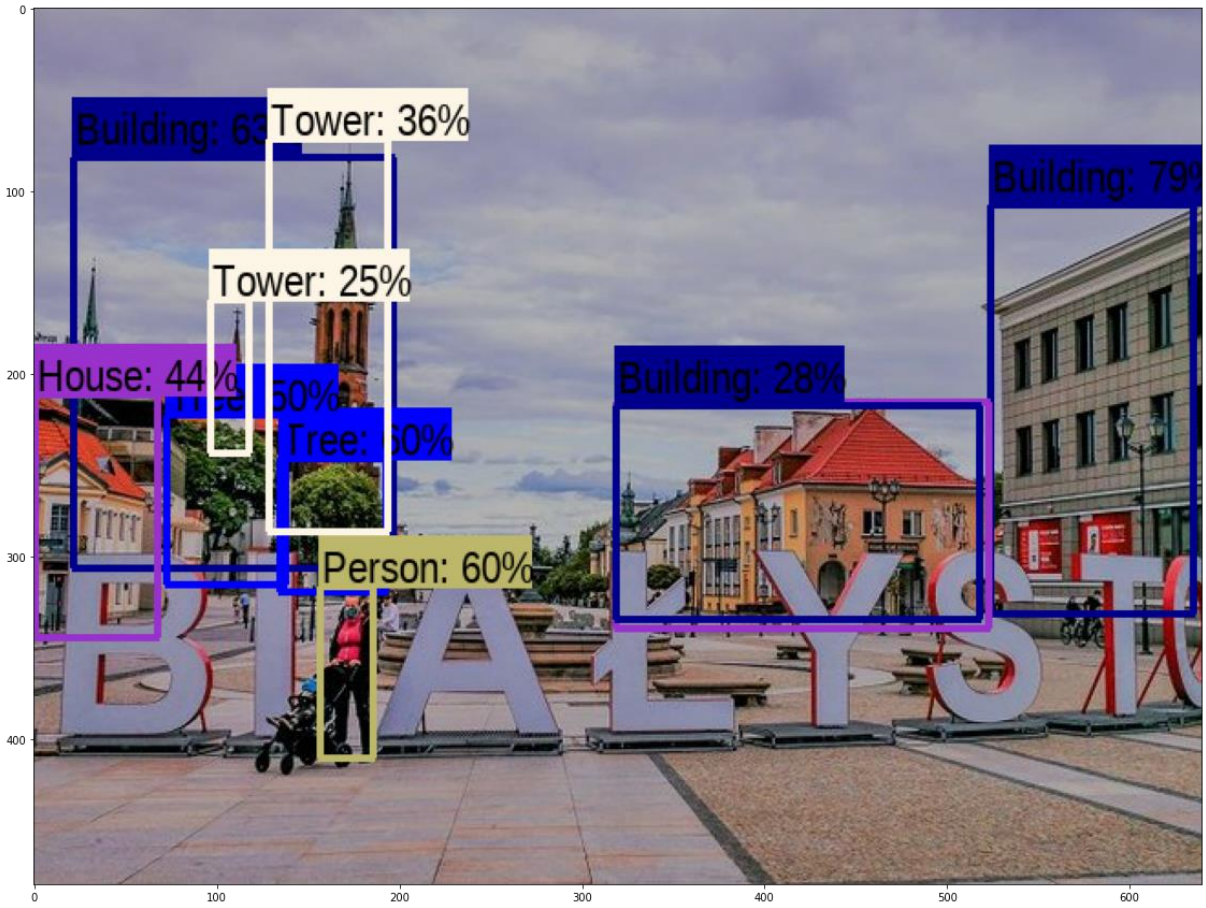
Oryginalny obraz (uwb):

Zdjęcie już po wykryciu obiektów:

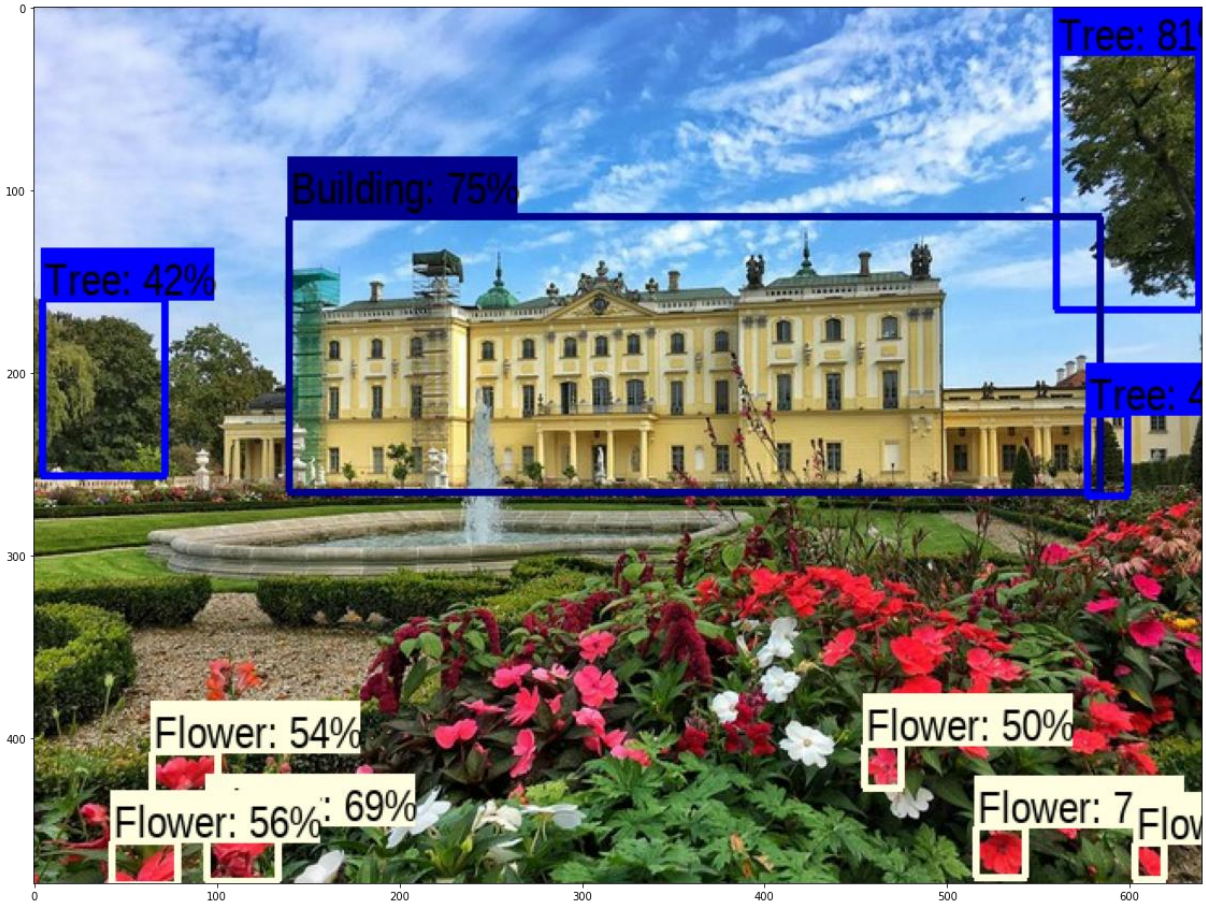
```
Found 100 objects.
Inference time:  40.51938605308533
```

Jak widzimy, program pozwala nawet na wykrycie, czy na zdjęciu znajduje się kobieta, bądź mężczyzna, a gdy nie jest pewny, to podpisuje daną postać jako osoba. Co ciekawe program wykrywa nawet buty na człowieku. Czas szukania obiektów to 40 sekund.

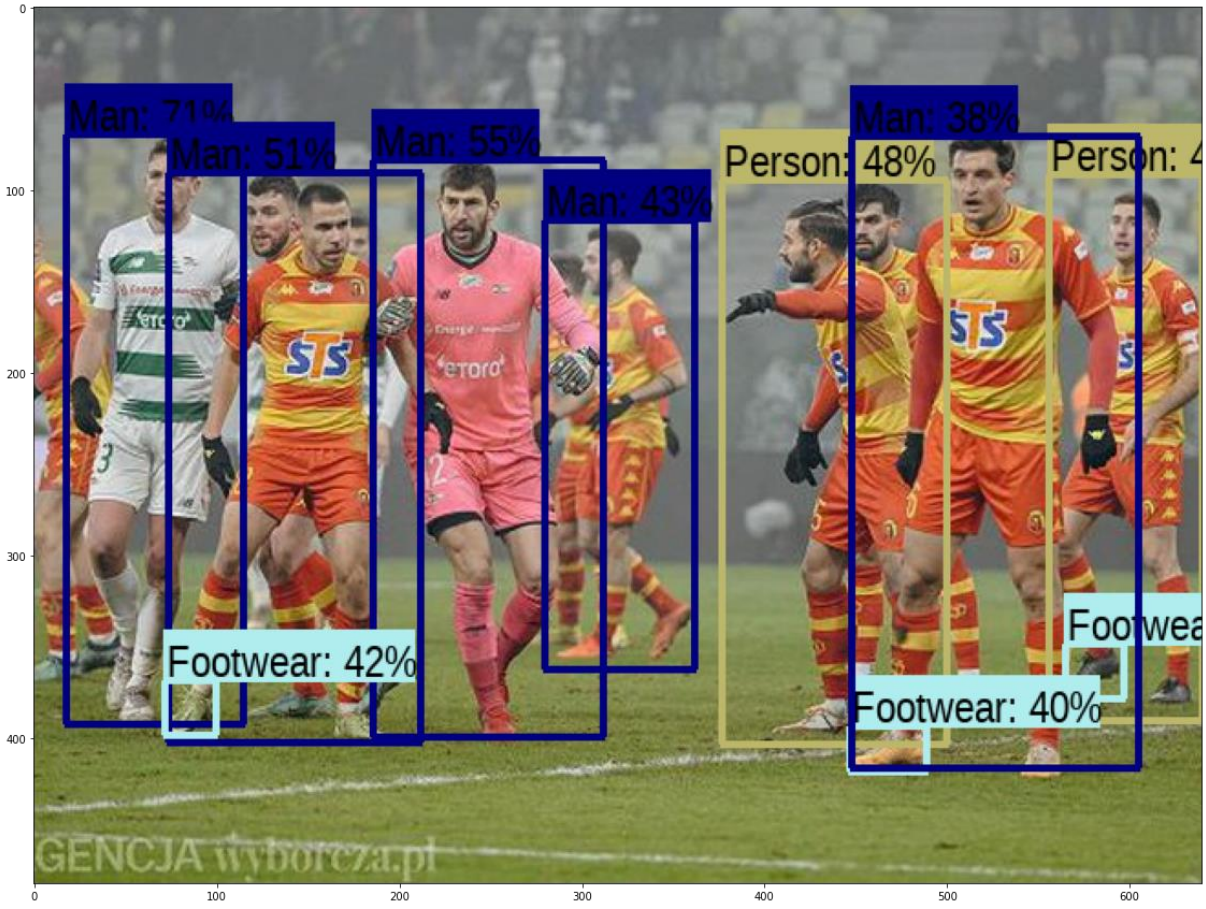
Inne przykłady:




```
Inference time:  3.028883218765259
Inference time: 4.244513511657715
```




```
Image downloaded to /tmp/tmpjl7qh9v7.jpg.
Found 100 objects.
Inference time:  1.6270663738250732
Inference time: 3.239697217941284
```




```
Inference time:  1.243797779083252
Inference time: 2.561464786529541
```

Im mniej obiektów, tym program działa szybciej. Najwolniej zadziałał na pierwszym zdjęciu, ponieważ jest tam najwięcej obiektów, jednak odpalając te zdjęcie ponownie, czas się skrócił. Zauważyłem, że zawsze pierwsze zdjęcie działa najwolniej.

KOD:

Poniższe funkcje służą do pobierania obrazów i wizualizacji

```
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/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


Tutaj zapisujemy obraz i wyświetlamy

image_url =
"https://upload.wikimedia.org/wikipedia/commons/6/60/Naxos_Taverna.jpg"
downloaded_image_path = download_and_resize_image(image_url, 1280, 856,
True)
```

Pozostałe funkcje służą do zastosowania modółów aby móc wykrywać dane obiekty.