from collections import Counter

import torch

# Assuming you have an image and a predictor already set up

outputs = predictor(image)

# Convert image to RGB format for visualization if it's not already

if image.shape[2] == 3: # Check if the image has 3 channels

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

else:

image\_rgb = image # If it's already RGB or grayscale, no need to convert

# Get the bounding boxes

boxes = outputs["instances"].pred\_boxes.tensor.to("cpu").numpy()

nb\_box=0

# Draw each bounding box on the image

for box in boxes:

# Convert box coordinates from float to int

box = box.astype(np.int32)

# Draw rectangle on the image

cv2.rectangle(image\_rgb, (box[0], box[1]), (box[2], box[3]), (0, 255, 0), 2)

nb\_box+=1

# Convert image back to BGR for displaying with cv2

image = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2BGR)

cv2\_imshow(image)

instances = outputs["instances"]

types=['apple', 'fraise', 'kiwi', 'lemon', 'orange']

pred\_classes = instances.pred\_classes

# Count the number of occurrences of each value

unique\_classes, counts = torch.unique(pred\_classes, return\_counts=True)

# Find the value with the max count

max\_count\_val = unique\_classes[counts.argmax()]

# print(f"Unique classes: {unique\_classes}")

# print(f"Counts: {counts}")

type\_fruit = types[max\_count\_val]

print(f"The type is: {type\_fruit}")

vizualize with scores   
# We can use `Visualizer` to draw the predictions on the image.

import copy

image2 = copy.deepcopy(image)

outputs1 = predictor(image2)

metadata = MetadataCatalog.get("dataset\_train6")

v1 = Visualizer(image2[:, :, ::-1], metadata=metadata) # Assuming `img` is in BGR format, as read by OpenCV.

out = v1.draw\_instance\_predictions(outputs1["instances"].to("cpu"))

cv2\_imshow(out.get\_image()[:, :, ::-1]) # Convert from RGB back to BGR for displaying with OpenCV.