from torchvision.ops import nms

import torch

from collections import Counter

# Assuming `image` and `predictor` are already defined and set up

outputs = predictor(image)

instances = outputs["instances"].to("cpu")

# 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 and scores

boxes = instances.pred\_boxes.tensor

scores = instances.scores

pred\_classes = instances.pred\_classes

# Apply NMS

iou\_threshold = 0.75 # Intersection Over Union threshold

keep\_indices = nms(boxes, scores, iou\_threshold)

# Filter boxes, scores, and classes based on NMS

boxes\_after\_nms = boxes[keep\_indices].numpy()

scores\_after\_nms = scores[keep\_indices]

classes\_after\_nms = pred\_classes[keep\_indices]

# Draw each bounding box on the image

nb\_boxes = 0

for box in boxes\_after\_nms:

box = box.astype(np.int32) # Convert box coordinates from float to int

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

nb\_boxes +=1

# Convert image back to BGR for displaying with cv2

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

cv2\_imshow( image\_bgr)

# Assuming 'types' corresponds to the indices in 'pred\_classes'

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

# Count the number of occurrences of each value in classes\_after\_nms

counts = Counter(classes\_after\_nms.numpy())

# Find the most common class

most\_common\_class, most\_common\_count = counts.most\_common(1)[0]

type\_fruit = types[most\_common\_class]

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

print(f"Number of {type\_fruit}s is : {nb\_boxes}")

https://detectron2.readthedocs.io/en/latest/\_modules/detectron2/layers/nms.html

The provided code snippet is a part of Detectron2, a Facebook AI Research library for object detection and segmentation. This code defines functions related to Non-Maximum Suppression (NMS) for handling bounding boxes in object detection tasks, specifically for batched and rotated boxes scenarios. Let's break it down for easier understanding:

### 1. `batched\_nms` Function:

- \*\*Purpose\*\*: Performs Non-Maximum Suppression (NMS) on a set of bounding boxes and scores, but it operates on batches where each bounding box is associated with a category or class index. NMS is used to reduce the number of overlapping bounding boxes, keeping only the ones with the highest scores. This function ensures that NMS is applied separately for each category, preventing suppression across different object classes.

- \*\*Parameters\*\*:

- `boxes`: A tensor containing the coordinates of the bounding boxes.

- `scores`: The confidence scores for each bounding box.

- `idxs`: Category indices for each bounding box, ensuring that NMS is class-wise.

- `iou\_threshold`: The Intersection Over Union (IoU) threshold for determining when one box should suppress another.

- \*\*How It Works\*\*: It leverages the `torchvision.ops.boxes.batched\_nms` function for efficiency but ensures the input is in float precision to handle the range correctly.

### 2. `nms\_rotated` Function:

- \*\*Purpose\*\*: Similar to the standard NMS, but designed for rotated bounding boxes. Rotated bounding boxes are often used in tasks like Object Detection in scenes where objects are not aligned with the image axes, and also in tasks like text detection where the text might be oriented arbitrarily.

- \*\*Details\*\*: It considers the rotation of bounding boxes when calculating overlaps and deciding which boxes to keep. This is crucial for applications like OCR (Optical Character Recognition), where the orientation of the text can significantly change the meaning.

- \*\*Parameters\*\*: Similar to `batched\_nms` but with boxes expected to include an additional rotation angle.

### 3. `batched\_nms\_rotated` Function:

- \*\*Purpose\*\*: Combines the concepts of `batched\_nms` and `nms\_rotated` to perform NMS on batches of rotated bounding boxes, with class-wise separation.

- \*\*How It Works\*\*: It employs a clever trick of adding class-wise offsets to box coordinates before applying rotated NMS, ensuring that NMS is performed independently per class. This is particularly useful when dealing with datasets where objects can be in various orientations and belong to different categories.

In summary, these functions are critical for efficient object detection and segmentation in images, especially in complex scenes where objects can overlap, be of different classes, and have various orientations. They ensure that the final predictions are sparse, with reduced redundancy, and respect the class and orientation of the objects detected.