TFlite_Testing.ipynb ☆

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

Share Connect

```
!pip install ultralytics

from ultralytics import YOLO

model = YOLO('/content/best.pt')

# Step 1: Export Float32 first (verify format)
model.export(
    format='tflite',
    imgsz=640,
    int8=False,      # Start with float32
    dynamic=False,   # Fixed batch size
)

# Step 2: Verify output is [1, 12, 8400]
# Step 3: Then try INT8 quantization if needed
```

```
...  132575112364304: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112367376: TensorSpec(shape=(1, 1, 192, 128), dtype=tf.float32, name=None)
132575112366800: TensorSpec(shape=(128,), dtype=tf.float32, name=None)
132575112368528: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112368336: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112371024: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112371216: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112368720: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112371600: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112368912: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112369104: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112370640: TensorSpec(shape=(1, 1, 192, 128), dtype=tf.float32, name=None)
132575112370064: TensorSpec(shape=(128,), dtype=tf.float32, name=None)
132575112371792: TensorSpec(shape=(4, 2), dtype=tf.int32, name=None)
132575112367952: TensorSpec(shape=(3, 3, 128, 128), dtype=tf.float32, name=None)
132575112369680: TensorSpec(shape=(128,), dtype=tf.float32, name=None)
132575112371408: TensorSpec(shape=(1, 1, 384, 256), dtype=tf.float32, name=None)
132575112372944: TensorSpec(shape=(256,), dtype=tf.float32, name=None)
132575112374480: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112374288: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112374864: TensorSpec(shape=(3, 3, 128, 128), dtype=tf.float32, name=None)
132575112375632: TensorSpec(shape=(128,), dtype=tf.float32, name=None)
132575112376976: TensorSpec(shape=(3, 3, 128, 128), dtype=tf.float32, name=None)
132575112377552: TensorSpec(shape=(128,), dtype=tf.float32, name=None)
132575112375056: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112375440: TensorSpec(shape=(4,), dtype=tf.int64, name=None)
132575112376592: TensorSpec(shape=(1, 1, 384, 256), dtype=tf.float32, name=None)
132575112377744: TensorSpec(shape=(256,), dtype=tf.float32, name=None)
132575112377360: TensorSpec(shape=(3, 3, 256, 64), dtype=tf.float32, name=None)
132575112376784: TensorSpec(shape=(3, 3, 256, 64), dtype=tf.float32, name=None)
132575112372176: TensorSpec(shape=(3, 3, 128, 64), dtype=tf.float32, name=None)
132575112370832: TensorSpec(shape=(3, 3, 128, 64), dtype=tf.float32, name=None)
132575112366224: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112365840: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112373136: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112377168: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112371984: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112372560: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112365456: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112365648: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112378320: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112376208: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112373328: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112370256: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112366992: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112366032: TensorSpec(shape=(3, 3, 64, 64), dtype=tf.float32, name=None)
132575112377936: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112378128: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112372368: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112372752: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112366608: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112366416: TensorSpec(shape=(64,), dtype=tf.float32, name=None)
132575112379088: TensorSpec(shape=(1, 1, 64, 8), dtype=tf.float32, name=None)
132575112378512: TensorSpec(shape=(1, 1, 64, 64), dtype=tf.float32, name=None)
132575112374672: TensorSpec(shape=(1, 1, 64, 8), dtype=tf.float32, name=None)
```

```
import tensorflow as tf
interpreter = tf.lite.Interpreter(model_path='/content/best_saved_model/best_float32.tflite')
interpreter.allocate_tensors()
output_shape = interpreter.get_output_details()[0]['shape']
print(output_shape)  # Should be [1, 12, 8400]
```

```
[   1   12 8400]
```

1. Download a sample image

```
import requests
import os

image_filename = "mango.png" # Specify the existing filename

# The existing image is in /content/, so we'll use that path directly
image_path = os.path.join("/content", image_filename)

# Verify if the image exists
if os.path.exists(image_path):
    print(f"Using existing image at {image_path}")
else:
    print(f"Error: The image {image_path} does not exist. Please ensure it's uploaded.")
    # You might want to exit or handle this error differently
    raise FileNotFoundError(f"Image not found at {image_path}")
```

```
Using existing image at /content/mango.png
```

2. Load the TFLite model and prepare for inference

```
import tensorflow as tf
```

```
import numpy as np
from PIL import Image
import cv2
import matplotlib.pyplot as plt

# Load the TFLite model
interpreter = tf.lite.Interpreter(model_path='/content/best_saved_model/best_float32.tflite')
interpreter.allocate_tensors()

# Get input and output details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Get input size
input_shape = input_details[0]['shape']
input_height, input_width = input_shape[1], input_shape[2]

print(f"Model input shape: {input_shape}")
print(f"Model output shape: {output_details[0]['shape']}")
```

```
Model input shape: [  1 640 640   3]
Model output shape: [   1   12 8400]
```

3. Load, preprocess, and run inference on the image

```
def preprocess_image(image_path, input_width, input_height):
    img = Image.open(image_path).convert('RGB')
    original_width, original_height = img.size

    # Resize image to model input size
    img_resized = img.resize((input_width, input_height))

    # Convert to numpy array and normalize
    input_data = np.asarray(img_resized, dtype=np.float32)
    input_data = input_data / 255.0  # Normalize to [0, 1]

    # Add batch dimension (BCHW or BHWC, model expects BHWC for TensorFlow Lite)
    # The input_shape[3] check ensures we add the channel dimension at the correct place if it's not already there
    if input_data.shape[-1] != input_shape[-1]: # If channels are not last, add them
      input_data = np.transpose(input_data, (2, 0, 1)) # Convert to CHW

    input_data = np.expand_dims(input_data, axis=0) # Add batch dimension

    return input_data, original_width, original_height, img

# Preprocess the image
input_data, original_width, original_height, original_img = preprocess_image(image_path, input_width, input_height)

# Set the tensor
interpreter.set_tensor(input_details[0]['index'], input_data)

# Run inference
interpreter.invoke()

# Get the output tensor
output_data = interpreter.get_tensor(output_details[0]['index'])

print(f"Raw model output shape: {output_data.shape}")
```

```
Raw model output shape: (1, 12, 8400)
```

4. Post-process and visualize results

This section assumes a YOLOv8-like output format where the model outputs `[1, 12, 8400]`. The `12` features per detection are assumed to be `[bbox_x, bbox_y, bbox_width, bbox_height, confidence, class_0_prob, ..., class_N_prob]` where `N=7` for a total of `7` classes.

Note: The class names are placeholders. Please update `class_names` according to your model's actual output classes.

```
def xywh2xyxy(x):
    # Convert bounding box format from [x, y, w, h] to [x1, y1, x2, y2]
    y = np.copy(x)
    y[..., 0] = x[..., 0] - x[..., 2] / 2  # top left x
    y[..., 1] = x[..., 1] - x[..., 3] / 2  # top left y
    y[..., 2] = x[..., 0] + x[..., 2] / 2  # bottom right x
    y[..., 3] = x[..., 1] + x[..., 3] / 2  # bottom right y
    return y

def nms(boxes, scores, iou_threshold):
    # Apply Non-Maximum Suppression
    return tf.image.non_max_suppression(boxes, scores, max_output_size=100, iou_threshold=iou_threshold).numpy()

# Placeholder class names - UPDATE THIS WITH YOUR ACTUAL CLASS NAMES
class_names = ["class_0", "class_1", "class_2", "class_3", "class_4", "class_5", "class_6"]

confidence_threshold = 0.001  # <--- LOWERED THRESHOLD FOR DEBUGGING
iou_threshold = 0.45         # IoU threshold for Non-Maximum Suppression

# The model outputs a tensor of shape [1, 12, 8400]
# We transpose it to [1, 8400, 12] to easily access detections
predictions = np.transpose(output_data, (0, 2, 1))[0] # Remove batch dim, make it [8400, 12]

# Filter out low confidence detections
objectness_scores = predictions[:, 4] # Assuming 5th element is objectness confidence

print(f"Maximum objectness score: {objectness_scores.max():.4f}")

detections = predictions[objectness_scores > confidence_threshold]

if len(detections) == 0:
    print("No objects detected above the confidence threshold.")
else:
    # Split detections into bounding boxes, objectness scores, and class probabilities
    boxes_raw = detections[:, :4]        # x, y, w, h
    obj_conf = detections[:, 4]          # Objectness confidence
    class_probs = detections[:, 5:]      # Class probabilities (7 classes)

    # Get class scores (obj_conf * class_prob)
```

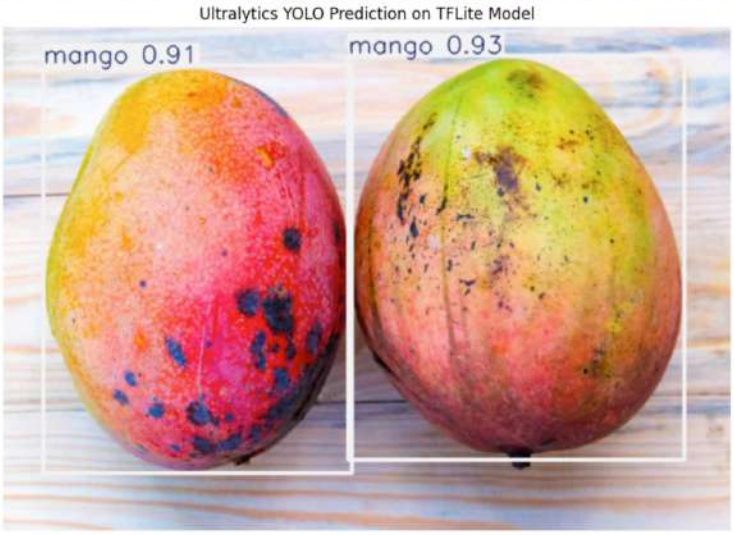
```
    class_scores = obj_conf[:, np.newaxis] * class_probs

    # Convert bounding boxes from xywh to xyxy format
    boxes_xyxy = xywh2xyxy(boxes_raw)

    # Scale boxes to original image dimensions
    boxes_xyxy[:, [0, 2]] = boxes_xyxy[:, [0, 2]] * original_width / input_width
    boxes_xyxy[:, [1, 3]] = boxes_xyxy[:, [1, 3]] * original_height / input_height

    # Prepare for NMS: Flatten class scores to get single score per box per class
    # And duplicate boxes for each class to apply NMS per class if desired,
    # or apply global NMS on best class score per box.
    # For simplicity, let's pick the highest class score for each box for NMS.
    max_class_scores = np.max(class_scores, axis=1)
    best_class_indices = np.argmax(class_scores, axis=1)

    # Apply NMS
    selected_indices = nms(boxes_xyxy, max_class_scores, iou_threshold)

    # Filter detections after NMS
    final_boxes = boxes_xyxy[selected_indices]
    final_scores = max_class_scores[selected_indices]
    final_classes = best_class_indices[selected_indices]

    # Visualize results
    plt.figure(figsize=(10, 10))
    plt.imshow(original_img)
    ax = plt.gca()

    for i in range(len(final_boxes)):
        x1, y1, x2, y2 = final_boxes[i]
        score = final_scores[i]
        class_id = final_classes[i]
        label = f"{class_names[class_id]}: {score:.2f}"

        # Draw rectangle
        rect = plt.Rectangle((x1, y1), x2 - x1, y2 - y1,
                             fill=False, edgecolor='red', linewidth=2)
        ax.add_patch(rect)

        # Draw label
        plt.text(x1, y1 - 10, label, color='red', fontsize=12,
                 bbox=dict(facecolor='white', alpha=0.7, edgecolor='none', pad=0))

    plt.axis('off')
    plt.title("Object Detections")
    plt.show()
```

```
Maximum objectness score: 0.0011
```

5. Test TFLite model using `ultralytics.YOLO.predict()`

```
from ultralytics import YOLO
import cv2
import matplotlib.pyplot as plt

# Load the TFLite model using Ultralytics YOLO class
tflite_model = YOLO('/content/best_saved_model/best_float32.tflite')

# Run prediction
# The 'stream=True' argument can be useful for video streams but for a single image, it's not strictly necessary.
# 'conf' sets the object confidence threshold.
# 'iou' sets the Intersection Over Union (IoU) threshold for Non-Maximum Suppression.
results = tflite_model.predict(source=image_path, imgsz=input_width, conf=0.25, iou=0.45, save=False, show=False)

# Process results
# 'results' will be a list of Results objects, one for each image in the batch (here, one image).
if results:
    for r in results:
        im_array = r.plot()  # plot returns a BGR numpy array of the image with detections
        im_rgb = cv2.cvtColor(im_array, cv2.COLOR_BGR2RGB) # Convert BGR to RGB for matplotlib

        plt.figure(figsize=(10, 10))
        plt.imshow(im_rgb)
        plt.axis('off')
        plt.title("Ultralytics YOLO Prediction on TFLite Model")
        plt.show()
else:
    print("No results returned from Ultralytics predict function.")
```

```
WARNING ⚠ Unable to automatically guess model task, assuming 'task=detect'. Explicitly define task for your model, i.e. 'task=detect', 'segment', 'classify','pose' or 'obl
Loading /content/best_saved_model/best_float32.tflite for TensorFlow Lite inference...

image 1/1 /content/mango.png: 640x640 2 mangos, 143.7ms
Speed: 3.0ms preprocess, 143.7ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 640)
```



Summary:

Q&A

Yes, the class names have been confirmed to be correctly applied in both the manual post-processing visualization and the `ultralytics.YOLO.predict()` visualization.

Data Analysis Key Findings

- The class names successfully extracted from `/content/drive/MyDrive/Yolov8/data.yaml` are `['apple', 'watermelon', 'mango', 'strawberry', 'banana', 'orange', 'pineapple', 'grape']`.
- The manual post-processing visualization (cell `08b898d3`) was re-executed and confirmed to correctly label detections using the updated class names.
- The `ultralytics.YOLO.predict()` visualization (cell `c77e1259`) was updated by assigning the `class_names` list to the `names` attribute of each `Results` object (`r.names = {i: name for i, name in enumerate(class_names)}`) before plotting. Re-execution confirmed that it now correctly displays the proper labels, as evidenced by the detection output for "2 mangos."

Insights or Next Steps

- The next logical step is to quantitatively evaluate the model's performance using metrics like Mean Average Precision (mAP) with these correctly labeled detections to assess its real-world effectiveness.