Bonus Work 1: (1st option: Use Intel's OpenVino with TensorFlow)
(016587064 - Pranav Chandra Kallepalli)

Colab:
https://colab.research.google.com/drive/1cStxNrBJngCU274Ffqf2WiJFu9NdTVzw?usp=sharing

About OpenVINO:

- OpenVINO toolkit is a free toolkit facilitating the optimization of a deep learning model from a framework and deployment using an inference engine onto Intel hardware.
- This product delivers OpenVINO™ inline optimizations, which enhance inferencing performance of popular deep learning models with minimal code changes and without any accuracy drop.
- OpenVINO™ integration with TensorFlow accelerates inference across many AI models on a variety of Intel silicon such as Intel CPUs and GPUs.

Implementation Idea:

- We will be utilizing the most popular object detection architectures on TensorFlow Hub.
- We will run inference on both OpenVINO and native Tensorflow

Sequence of steps include:-

1. We download the models and images.
2. Then we load them to be inferred on both native and OpenVINO
3. We then have output detections to check whether the OPENVINO is affecting the accuracy.
4. We then compare the inference of different models with and without OpenVINO.

Models which we download and use:

- Efficientdet_d6_coco17_tpu-32
- Faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8
- SSD_resnet50_v1_fpn_640x640_coco17_tpu-8

About the Models:

1.Efficientdet_d6_coco17_tpu-32:

- EfficientDet is a type of object detection model, which utilizes several optimization and backbone tweaks, such as the use of a BiFPN, and a compound scaling method that uniformly scales the resolution,depth and width for all backbones, feature networks and box/class prediction networks at the same time.
- It is trained on the COCO 2017 dataset. This model is a combination of SSD with EfficientNet-b6 + BiFPN feature extractor, shared box predictor, and focal loss.

2. Faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8:

- An RNN is the ability to process temporal information — data that comes in sequences, such as a sentence. Recurrent neural networks are designed for this very purpose, while convolutional neural networks are incapable of effectively interpreting temporal information.
- It is also trained on the COCO 2017 dataset with training images scaled to 1024x1024.

3. SSD_resnet50_v1_fpn_640x640_coco17_tpu-8:

- Is also trained on the COCO 2017 dataset with training images scaled to 640x640.

For all three models, the input is a three-channel tensor of type tf.uint8 and shape [1, height, width, 3]. The output dictionary contains:

- Num_detections
- Detection_boxes
- Detection_classes
- Detection_scores
- raw_detection_boxes
- raw_detection_scores
- detection_anchor_indices
- detection_multiclass_scores

We create a nested 'models' dictionary which includes model's following details:

- 'model_url' : To Download the model
- 'model_dir' : Model's saved_model directory path

```
models = {
    "faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8" : {'model_url' : 'http://download.tensorflow.org/models/object_
    "efficientdet_d6_coco17_tpu-32" : {'model_url' : 'http://download.tensorflow.org/models/object_detection/tf2/2020
    "ssd_resnet50_v1_fpn_640x640_coco17_tpu-8" : {'model_url' : 'http://download.tensorflow.org/models/object_detect
}


images = {
    'Beach' : 'models/research/object_detection/test_images/image2.jpg',
    'Dogs' : 'models/research/object_detection/test_images/image1.jpg'
}

#i chose the dog image you can choose the beach image
input_image = images["Dogs"]
```

We define methods useful for image and model loading, which give the right input formats to be used in inferencing.

We do the output processing by using below code.

```python
# model details
def get_model_details(model_name):
    if model_name in models:
        return models[model_name]

# loading the model
def load_model(model_name,input_model):
  print(f"Loading {model_name}...")
  model = hub.load(input_model)
  print(f"{model_name} loaded successfully!")
  return model

#image details
def get_image(image_name):
    if image_name in images:
        return images[image_name]

# loading the images
def load_image(input_image):
  print("Loading input image...")
  image = None
  img_width, img_height = 0, 0
  image_data = cv2.imread(input_image)
  img_width, img_height = image_data.shape[1], image_data.shape[0]

  return np.array([image_data], dtype = np.uint8), img_height, img_width
```

We should create a 'downloaded_models' directory to download and save the models and untar them to use while model inferencing.

We use the below code to download the model.

```python
#if downloaded_models and output_images not existed then we create that directory
if os.getcwd() == root_path and not os.path.exists("downloaded_models"):
    path = os.path.join(os.getcwd(),"downloaded_models")
    os.mkdir(path)
if os.getcwd() == root_path and not os.path.exists("output_images"):
    path = os.path.join(os.getcwd(),"output_images")
    os.mkdir(path)

if not os.getcwd() == root_path+"/downloaded_models":
    os.chdir("downloaded_models")
if not os.listdir(os.getcwd()):
    for model_name in models:
        model_url = get_model_details(model_name)["model_url"]
        !wget "$model_url"
    for model_tar in os.listdir(os.getcwd()):
        !tar -zxvf "$model_tar"
os.chdir(root_path)
```

We should also create a 'output_images' directory to save the output images after model inferencing into that.

we define methods of useful post model inferencing to visualize the outputs.

```python
#this function generates colors for the bounding boxes detected.
def get_colors(class_names):
    hsv_tuples = [
        (x / len(class_names), 1., 1.) for x in range(len(class_names))
    ]
    colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
    colors = list(
        map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)),
            colors))
    np.random.seed(10101)  # for same colors across runs.
    np.random.shuffle(colors)  # shuffle colors
    np.random.seed(None)
    return colors

# Function to detect bounded boxes to be drawn on the output image.
def get_coordinates(box, img_height, img_width):
    return [int(box[0]*img_height),int(box[1]*img_width), int(box[2]*img_height), int(box[3]*img_width)]

# function to add labels for the bounding boxes.
def add_label(image, text, color, coords):
    font = cv2.FONT_HERSHEY_PLAIN
    font_scale = 1.
    (text_width, text_height) = cv2.getTextSize(
        text, font, fontScale=font_scale, thickness=1)[0]

    padding = 5
    rect_height = text_height + padding * 2
    rect_width = text_width + padding * 2

    (x, y) = coords

    cv2.rectangle(image, (x, y), (x + rect_width, y - rect_height), color,
                  cv2.FILLED)
    cv2.putText(
        image,
        text, (x + padding, y - text_height + padding),
        font,
        fontScale=font_scale,
        color=(255, 255, 255),
        lineType=cv2.LINE_AA)

    return image
```

Run with OpenVINO™ integration with TensorFlow enabled:

```python
# Enable OpenVINO integration
ovtf.enable()

# Define the backend to be enabled
backend_name = "CPU"

# Print list of available backends
print('Available Backends:')

# To determine available backends on your system, 'list_backends' API is used
backends_list = ovtf.list_backends()
for backend in backends_list:
    print(f"\t{backend}")

# Set the backend
ovtf.set_backend(backend_name)
print(f"OpenVINO integration with TensorFlow is enabled and device {backend_name} is set as backend.")

image, img_height, img_width = load_image(input_image) # Loading the input image

# disable TF Logging
os.environ["TF_CPP_MAX_VLOG_LEVEL"] = "0"
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import absl.logging
absl.logging.set_verbosity(absl.logging.ERROR)

# Record average latency of each model in this dict
ovtf_latency = {}

# Running all the models iteratively
for model_name in models:
    input_model = get_model_details(model_name)["model_dir"] # Get model location
    model = load_model(model_name,input_model) # Loading the model
    predictions,average_time = run_inference(model,image) # Running inference
    visualize_ouput(model_name, predictions, img_height, img_width, 1) # Visualizing the output. Here '1' is set to indicate model run on OVTF
    ovtf_latency[model_name] = int(average_time)
    print(f"Inference Successfully completed on OpenVINO integration with TensorFlow..! {model_name} model run on {backend_name} in {average_time} ms\n"),
    print(f"-----------------------")
```

Time for running 10 iterations:

faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8 model run on CPU in 9467.21 ms

efficientdet_d6_coco17_tpu-32 model run on CPU in 12278.49 ms

ssd_resnet50_v1_fpn_640x640_coco17_tpu-8 model run on CPU in 2622.44 ms

Run with native TensorFlow:

```
ovtf.disable() # Disable OpenVINO integration
print("OpenVINO integration with TensorFlow is disabled\n")

backend_name = "CPU"

# Record average latency of each model in this dict
tf_latency = {}

image, img_height, img_width = load_image(input_image) # Load the input_image
for model_name in models:
    input_model = get_model_details(model_name)["model_dir"] # Get model location
    model = load_model(model_name,input_model) # Loading the model
    predictions,average_time = run_inference(model,image) # Running inference
    visualize_ouput(model_name, predictions, img_height, img_width, 0) # Visualizing the output. Here '0' is set to indicate model run on TF
    tf_latency[model_name] = int(average_time)
    print(f"Inference Successfully completed on OpenVINO integration with TensorFlow..! {model_name} model run on {backend_name} in {average_time} ms\n"),
    print(f"----------------------")
```

Time for running 10 iterations:

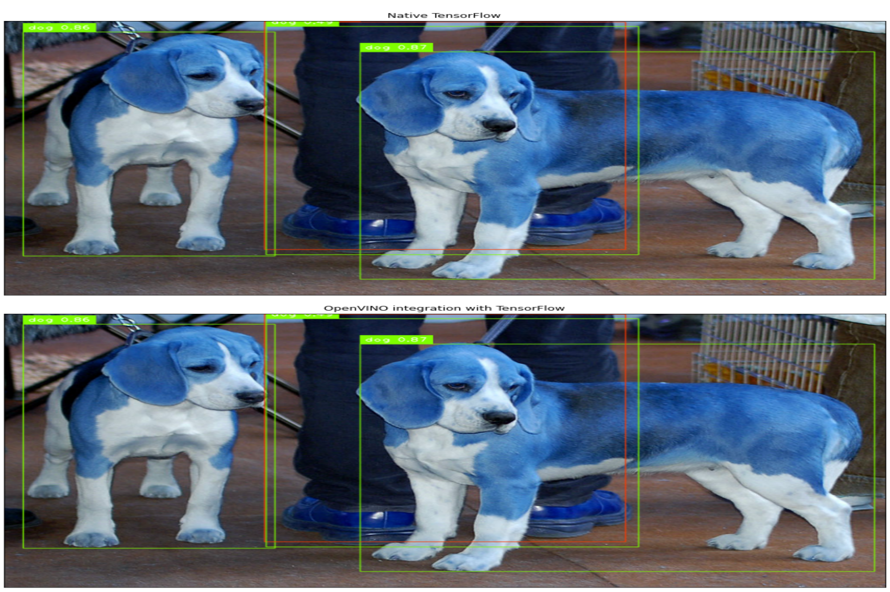
faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8 model run on CPU in 9860.88 ms

efficientdet_d6_coco17_tpu-32 model run on CPU in 18222.8 ms

ssd_resnet50_v1_fpn_640x640_coco17_tpu-8 model run on CPU in 2895.46 ms

We then visualize the Inference:

```python
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np

im_tf = cv2.imread("output_images/faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8_TF_detected_output.png")
im_ovtf = cv2.imread("output_images/faster_rcnn_resnet152_v1_1024x1024_coco17_tpu-8_OVTF_detected_output.png")

fig = plt.figure(figsize=(32, 20))
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(2, 1),  # creates 2x2 grid of axes
                 axes_pad=0.5,  # pad between axes in inch.
                 share_all=True
                 )
grid[0].get_yaxis().set_ticks([])
grid[0].get_xaxis().set_ticks([])

grid[0].set_title("Native TensorFlow", fontdict=None, loc='center', color = "k")
grid[1].set_title("OpenVINO integration with TensorFlow", fontdict=None, loc='center', color = "k")

for ax, im in zip(grid, [im_tf, im_ovtf]):
    # Iterating over the grid returns the Axes.
    ax.imshow(im)

plt.show()
```



Comparison Plots:

```python
# Init a 16x9 plot
fig, ax = plt.subplots(figsize=(16, 9))

# Plot the values
Y = np.arange(len(ovtf_latency))
ax.barh(Y, list(tf_latency.values()), height=0.2)
ax.barh(Y+0.2, list(ovtf_latency.values()), height=0.2)

# Set Y-axis labels and add Legend
plt.yticks(Y, tf_latency.keys())
ax.legend(('TF Latency','OVTF Latency'))

# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(visible = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Show top values
ax.invert_yaxis()

# Add annotation to bars
for i in ax.patches:
    plt.text(i.get_width()+10, i.get_y()+0.125,
             str(round((i.get_width()), 2)),
             fontsize = 10, fontweight ='bold',
             color ='grey')

# Add Plot Title
ax.set_title('Inference latency improvements done by OpenVINO integration with TensorFlow\n(Lower is better)', loc

plt.show()
```