Ungraded Lab: Mask R-CNN Image Segmentation Demo

In this lab, you will see how to use a Mask R-CNN model from Tensorflow Hub for object detection and instance segmentation. This means that aside from the bounding boxes, the model is also able to predict segmentation masks for each instance of a class in the image. You have already encountered most of the commands here when you worked with the Object Dection API and you will see how you can use it with instance segmentation models. Let's begin!

Note: You should use a TPU runtime for this colab because of the processing requirements for this model. We have already enabled it for you but if you'll be using it in another colab, you can change the runtime from `Runtime --> Change runtime type` then select `TPU`.

Installation

As mentioned, you will be using the Tensorflow 2 Object Detection API. You can do that by cloning the Tensorflow Model Garden and installing the object detection packages just like you did in Week 2.

[1]
```
# Clone the tensorflow models repository
!git clone --depth 1 https://github.com/tensorflow/models
```

```
Cloning into 'models'...
remote: Enumerating objects: 2826, done.
remote: Counting objects: 100% (2826/2826), done.
remote: Compressing objects: 100% (2365/2365), done.
remote: Total 2826 (delta 724), reused 1276 (delta 426), pack-reused 0
Receiving objects: 100% (2826/2826), 32.83 MiB | 25.70 MiB/s, done.
Resolving deltas: 100% (724/724), done.
```

[2]
```
%%bash
sudo apt install -y protobuf-compiler
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
cp object_detection/packages/tf2/setup.py .
python -m pip install .
```

```
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (from tensorflow-addons->tf-models-official>=2.5.1->object-detection==0.1) (2.7.1)
Requirement already satisfied: promise in /usr/local/lib/python3.7/dist-packages (from tensorflow-datasets->tf-models-official>=2.5.1->object-detection==0.1) (2.3)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.7/dist-packages (from tensorflow-datasets->tf-models-official>=2.5.1->object-detection==0.1) (5.2.0)
Requirement already satisfied: attrs>=18.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-datasets->tf-models-official>=2.5.1->object-detection==0.1) (21.2.0)
Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.7/dist-packages (from tensorflow-datasets->tf-models-official>=2.5.1->object-detection==0.1) (1.1.0)
Building wheels for collected packages: object-detection, py-cpuinfo, avro-python3, dill, future, seqeval
  Building wheel for object-detection (setup.py): started
  Building wheel for object-detection (setup.py): finished with status 'done'
  Created wheel for object-detection: filename=object_detection-0.1-py3-none-any.whl size=1660307 sha256=d195ce9e7af48962bfceac4cf8d16577c46cc9ae64ce2f90c58dc4c88040d965
  Stored in directory: /tmp/pip-ephem-wheel-cache-06n08upe/wheels/fa/a4/d2/e9a5057e414fd46c8e543d2706cd836d64e1fcd9eccceb2329
  Building wheel for py-cpuinfo (setup.py): started
  Building wheel for py-cpuinfo (setup.py): finished with status 'done'
  Created wheel for py-cpuinfo: filename=py_cpuinfo-8.0.0-py3-none-any.whl size=22257 sha256=8ed161887de5321aab116638312d409bc0adab85d8eb95145dc76db23e0c8684
  Stored in directory: /root/.cache/pip/wheels/d2/f1/1f/041add21dc9c4220157f1bd2bd6afe1f1a49524c3396b94401
  Building wheel for avro-python3 (setup.py): started
  Building wheel for avro-python3 (setup.py): finished with status 'done'
  Created wheel for avro-python3: filename=avro_python3-1.9.2.1-py3-none-any.whl size=43512 sha256=7e1aca1daa575ff2c99d6a0723ba58201ecd7d514fb11cb52d89680367b5492c
  Stored in directory: /root/.cache/pip/wheels/bc/49/5f/fdb5b9d85055c478213e0158ac122b596816149a02d82e0ab1
  Building wheel for dill (setup.py): started
  Building wheel for dill (setup.py): finished with status 'done'
  Created wheel for dill: filename=dill-0.3.1.1-py3-none-any.whl size=78544 sha256=04dd22f7e4618d4d85874b13a75d2c13f204472c92ad89d5aa3eb891359ae575
  Stored in directory: /root/.cache/pip/wheels/a4/61/fd/c57e374e580aa78a45ed78d5859b3a44436af17e22ca53284f
  Building wheel for future (setup.py): started
  Building wheel for future (setup.py): finished with status 'done'
  Created wheel for future: filename=future-0.18.2-py3-none-any.whl size=491070 sha256=00bb73453db69c09aac64ab753dc1afa957ea427a34f6fabc24cc3e4e3fbef77
  Stored in directory: /root/.cache/pip/wheels/56/b0/fe/4410d17b32f1f0c3cf54cdfb2bc04d7b4b8f4ae377e2229ba0
  Building wheel for seqeval (setup.py): started
  Building wheel for seqeval (setup.py): finished with status 'done'
  Created wheel for seqeval: filename=seqeval-1.2.2-py3-none-any.whl size=16181 sha256=2681c14fa36074a899067157bf34598909074b08e6f6259c9661807e10001672
  Stored in directory: /root/.cache/pip/wheels/05/96/ee/7cac4e74f3b19e3158dce26a20a1c86b3533c43ec72a549fd7
Successfully built object-detection py-cpuinfo avro-python3 dill future seqeval
Installing collected packages: requests, portalocker, future, dill, colorama, tf-slim, tensorflow-model-optimization, tensorflow-addons, seqeval, sentencepiece, sacrebleu, pyyaml, py-cpui
  Attempting uninstall: requests
    Found existing installation: requests 2.23.0
    Uninstalling requests-2.23.0:
      Successfully uninstalled requests-2.23.0
  Attempting uninstall: future
    Found existing installation: future 0.16.0
    Uninstalling future-0.16.0:
      Successfully uninstalled future-0.16.0
  Attempting uninstall: dill
    Found existing installation: dill 0.3.4
    Uninstalling dill-0.3.4:
      Successfully uninstalled dill-0.3.4
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed apache-beam-2.31.0 avro-python3-1.9.2.1 colorama-0.4.4 dill-0.3.1.1 fastavro-1.4.4 future-0.18.2 hdfs-2.6.0 lvis-0.5.3 object-detection-0.1 opencv-python-headless-4

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

  DEPRECATION: A future pip version will change local packages to be built in-place without first copying to a temporary directory. We recommend you use --use-feature=in-tree-build to tes
    pip 21.3 will remove support for this functionality. You can find discussion regarding this at https://github.com/pypa/pip/issues/7555.
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
multiprocess 0.70.12.2 requires dill>=0.3.4, but you have dill 0.3.1.1 which is incompatible.
google-colab 1.0.0 requires requests~=2.23.0, but you have requests 2.26.0 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
```

Import libraries

[3]
```
import tensorflow as tf
import tensorflow_hub as hub
```

```
import matplotlib
import matplotlib.pyplot as plt

import numpy as np
from six import BytesIO
from PIL import Image
from six.moves.urllib.request import urlopen

from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.utils import ops as utils_ops

tf.get_logger().setLevel('ERROR')

%matplotlib inline
```

Utilities

For convenience, you will use a function to convert an image to a numpy array. You can pass in a relative path to an image (e.g. to a local directory) or a URL. You can see this in the `TEST_IMAGES` dictionary below. Some paths point to test images that come with the API package (e.g. `Beach`) while others are URLs that point to images online (e.g. `Street`).

```
[4] def load_image_into_numpy_array(path):
      """Load an image from file into a numpy array.

      Puts image into numpy array to feed into tensorflow graph.
      Note that by convention we put it into a numpy array with shape
      (height, width, channels), where channels=3 for RGB.

      Args:
        path: the file path to the image

      Returns:
        uint8 numpy array with shape (img_height, img_width, 3)
      """
      image = None
      if(path.startswith('http')):
        response = urlopen(path)
        image_data = response.read()
        image_data = BytesIO(image_data)
        image = Image.open(image_data)
      else:
        image_data = tf.io.gfile.GFile(path, 'rb').read()
        image = Image.open(BytesIO(image_data))

      (im_width, im_height) = (image.size)
      return np.array(image.getdata()).reshape(
          (1, im_height, im_width, 3)).astype(np.uint8)


    # dictionary with image tags as keys, and image paths as values
    TEST_IMAGES = {
      'Beach' : 'models/research/object_detection/test_images/image2.jpg',
      'Dogs' : 'models/research/object_detection/test_images/image1.jpg',
      # By Américo Toledano, Source: https://commons.wikimedia.org/wiki/File:Biblioteca_Maim%C3%B3nides,_Campus_Universitario_de_Rabanales_007.jpg
      'Phones' : 'https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/Biblioteca_Maim%C3%B3nides%2C_Campus_Universitario_de_Rabanales_007.jpg/1024px-Biblioteca_Maim%C3%B3nides%2C_Campus_
      # By 663highland, Source: https://commons.wikimedia.org/wiki/File:Kitano_Street_Kobe01s5s4110.jpg
      'Street' : 'https://upload.wikimedia.org/wikipedia/commons/thumb/0/08/Kitano_Street_Kobe01s5s4110.jpg/2560px-Kitano_Street_Kobe01s5s4110.jpg'
    }
```

Load the Model

Tensorflow Hub provides a Mask-RCNN model that is built with the Object Detection API. You can read about the details here. Let's first load the model and see how to use it for inference in the next section.

```
[5] model_display_name = 'Mask R-CNN Inception ResNet V2 1024x1024'
    model_handle = 'https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1'

    print('Selected model:'+ model_display_name)
    print('Model Handle at TensorFlow Hub: {}'.format(model_handle))
```

```
Selected model:Mask R-CNN Inception ResNet V2 1024x1024
Model Handle at TensorFlow Hub: https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1
```

```
[6] # This will take 10 to 15 minutes to finish
    print('loading model...')
    hub_model = hub.load(model_handle)
    print('model loaded!')
```

```
loading model...
model loaded!
```

Inference

You will use the model you just loaded to do instance segmentation on an image. First, choose one of the test images you specified earlier and load it into a numpy array.

```
[7] # Choose one and use as key for TEST_IMAGES below:
    # ['Beach', 'Street', 'Dogs','Phones']

    image_path = TEST_IMAGES['Street']

    image_np = load_image_into_numpy_array(image_path)

    plt.figure(figsize=(24,32))
    plt.imshow(image_np[0])
    plt.show()
```

You can run inference by simply passing the numpy array of a *single* image to the model. Take note that this model does not support batching. As you've seen in the notebooks in Week 2, this will output a dictionary containing the results. These are described in the `Outputs` section of the documentation

[8]
```
# run inference
results = hub_model(image_np)

# output values are tensors and we only need the numpy()
# parameter when we visualize the results
result = {key:value.numpy() for key,value in results.items()}

# print the keys
for key in result.keys():
  print(key)
```

```
proposal_boxes_normalized
final_anchors
raw_detection_scores
mask_predictions
anchors
rpn_box_encodings
rpn_features_to_crop
num_proposals
rpn_box_predictor_features
detection_boxes
raw_detection_boxes
proposal_boxes
box_classifier_features
refined_box_encodings
detection_scores
detection_anchor_indices
detection_classes
detection_masks
num_detections
rpn_objectness_predictions_with_background
detection_multiclass_scores
image_shape
class_predictions_with_background
```

Visualizing the results

You can now plot the results on the original image. First, you need to create the `category_index` dictionary that will contain the class IDs and names. The model was trained on the COCO2017 dataset and the API package has the labels saved in a different format (i.e. `mscoco_label_map.pbtxt`). You can use the create_category_index_from_labelmap internal utility function to convert this to the required dictionary format.

[9]
```
PATH_TO_LABELS = './models/research/object_detection/data/mscoco_label_map.pbtxt'
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)

# sample output
print(category_index[1])
print(category_index[2])
print(category_index[4])
```

```
{'id': 1, 'name': 'person'}
```

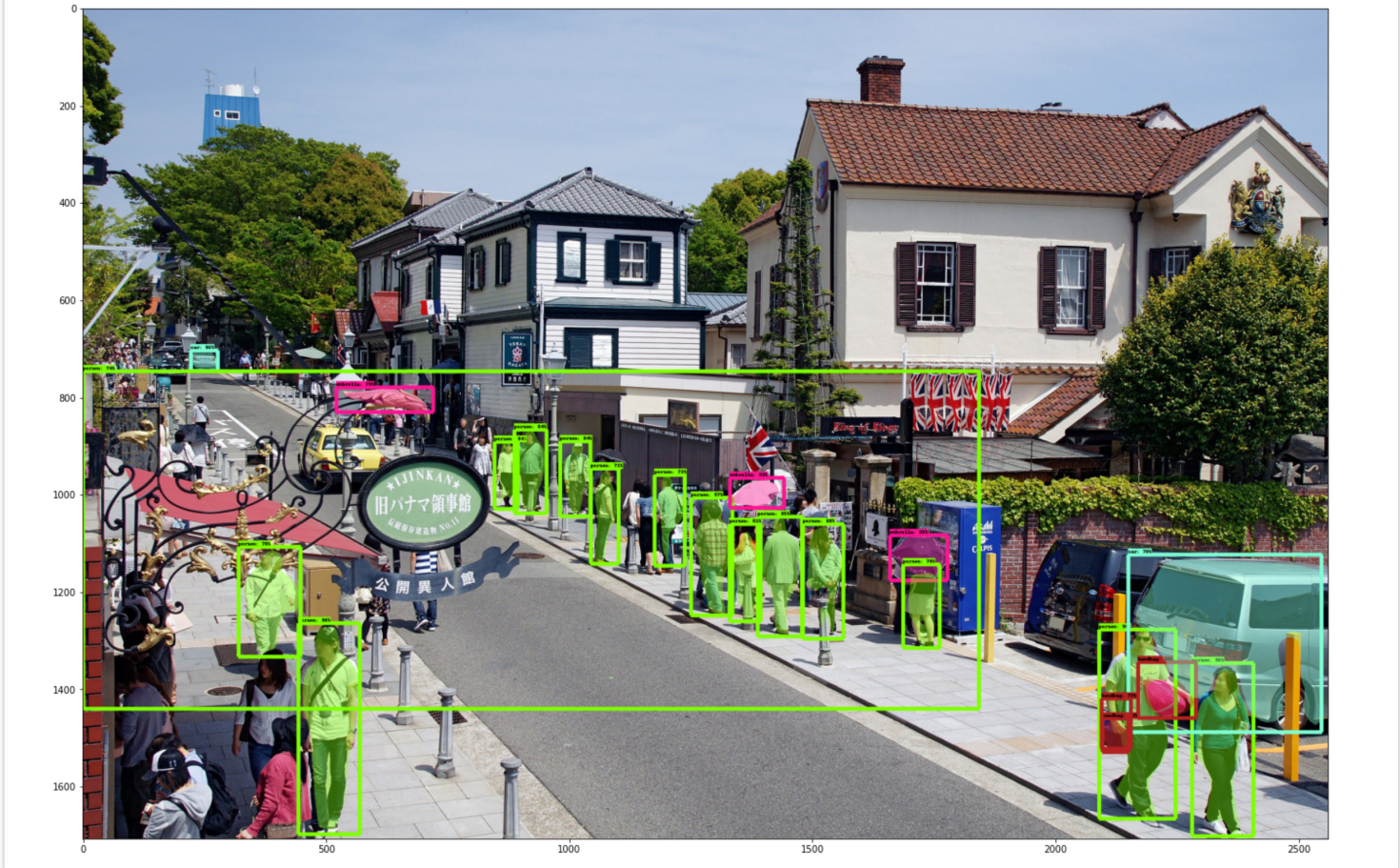
```
{'id': 2, 'name': 'bicycle'}
{'id': 4, 'name': 'motorcycle'}
```

Next, you will preprocess the masks then finally plot the results.

- The result dictionary contains a `detection_masks` key containing segmentation masks for each box. That will be converted first to masks that will overlay to the full image size.
- You will also select mask pixel values that are above a certain threshold. We picked a value of `0.6` but feel free to modify this and see what results you will get. If you pick something lower, then you'll most likely notice mask pixels that are outside the object.
- As you've seen before, you can use `visualize_boxes_and_labels_on_image_array()` to plot the results on the image. The difference this time is the parameter `instance_masks` and you will pass in the reframed detection boxes to see the segmentation masks on the image.

You can see how all these are handled in the code below.

```
# Handle models with masks:
label_id_offset = 0
image_np_with_mask = image_np.copy()

if 'detection_masks' in result:

  # convert np.arrays to tensors
  detection_masks = tf.convert_to_tensor(result['detection_masks'][0])
  detection_boxes = tf.convert_to_tensor(result['detection_boxes'][0])

  # reframe the the bounding box mask to the image size.
  detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes,
              image_np.shape[1], image_np.shape[2])

  # filter mask pixel values that are above a specified threshold
  detection_masks_reframed = tf.cast(detection_masks_reframed > 0.6,
                                     tf.uint8)

  # get the numpy array
  result['detection_masks_reframed'] = detection_masks_reframed.numpy()

# overlay labeled boxes and segmentation masks on the image
viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_mask[0],
      result['detection_boxes'][0],
      (result['detection_classes'][0] + label_id_offset).astype(int),
      result['detection_scores'][0],
      category_index,
      use_normalized_coordinates=True,
      max_boxes_to_draw=100,
      min_score_thresh=.70,
      agnostic_mode=False,
      instance_masks=result.get('detection_masks_reframed', None),
      line_thickness=8)

plt.figure(figsize=(24,32))
plt.imshow(image_np_with_mask[0])
plt.show()
```



✓ 11s completed at 2:10 PM