Lab 7 - BCC406

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Detecção e Segmentação de objetos

Prof. Eduardo e Prof. Pedro

Objetivos:

- Parte I : Detecção de objetos
- Parte II : Segmentação de imagens na Oxford Pet Dataset

Data da entrega : 04/12

- Este notebook é baseado em tensorflow e Keras.
- Execute todo notebook e salve tudo em um PDF **nomeado** como "NomeSobrenome-LabX.pdf"
- Envie o PDF via google FORM

Parte I - Detecção de Objetos (60pt)

Execute o tutorial do link. Faça um teste com os seguintes modelos:

- EfficientDet D0 512x512
- SSD MobileNet V2 FPNLite 320x320
- SSD ResNet50 V1 FPN 640x640 (RetinaNet50)
- Faster R-CNN ResNet50 V1 640x640
- Mask R-CNN Inception ResNet V2 1024x1024

Teste com imagens de:

- Praia
- Cachorros
- Pássartos

Imports and Setup

Let's start with the base imports.

```
# This Colab requires a recent numpy version.
!pip install numpy==1.24.3
!pip install protobuf==3.20.3
```

```
Collecting numpy==1.24.3
  Downloading numpy-1.24.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 64.3 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.23.5
    Uninstalling numpy-1.23.5:
      Successfully uninstalled numpy-1.23.5
ERROR: pip's dependency resolver does not currently take into account all the package
lida 0.0.10 requires fastapi, which is not installed.
lida 0.0.10 requires kaleido, which is not installed.
lida 0.0.10 requires python-multipart, which is not installed.
lida 0.0.10 requires uvicorn, which is not installed.
Successfully installed numpy-1.24.3
WARNING: The following packages were previously imported in this runtime:
  [numpy]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
Requirement already satisfied: protobuf==3.20.3 in /usr/local/lib/python3.10/dist-pac
```

```
import os
import pathlib

import matplotlib
import matplotlib.pyplot as plt

import io
import scipy.misc
import numpy as np
from six import BytesIO
from PIL import Image, ImageDraw, ImageFont
from six.moves.urllib.request import urlopen

import tensorflow as tf
import tensorflow_hub as hub

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

Utilities

Run the following cell to create some utils that will be needed later:

```
Helper method to load an image
Map of Model Name to TF Hub handle
List of tuples with Human Keypoints for the COCO 2017 dataset. This is needed for models with keypoints.
```

Run this!!

```
# @title Run this!!

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)


ALL_MODELS = {
'CenterNet HourGlass104 512x512' : 'https://tfhub.dev/tensorflow/centernet/hourglass_512x512/1',
'CenterNet HourGlass104 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/hourglass_512x512_kpts/1',
'CenterNet HourGlass104 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hourglass_1024x1024/1',
'CenterNet HourGlass104 Keypoints 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hourglass_1024x1024_kpts/1',
'CenterNet Resnet50 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v1_fpn_512x512/1',
'CenterNet Resnet50 V1 FPN Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v1_fpn_512x512_kpts/1',
'CenterNet Resnet101 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet101v1_fpn_512x512/1',
'CenterNet Resnet50 V2 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v2_512x512/1',
'CenterNet Resnet50 V2 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v2_512x512_kpts/1',
'EfficientDet D0 512x512' : 'https://tfhub.dev/tensorflow/efficientdet/d0/1',
'EfficientDet D1 640x640' : 'https://tfhub.dev/tensorflow/efficientdet/d1/1',
'EfficientDet D2 768x768' : 'https://tfhub.dev/tensorflow/efficientdet/d2/1',
'EfficientDet D3 896x896' : 'https://tfhub.dev/tensorflow/efficientdet/d3/1',
'EfficientDet D4 1024x1024' : 'https://tfhub.dev/tensorflow/efficientdet/d4/1',
'EfficientDet D5 1280x1280' : 'https://tfhub.dev/tensorflow/efficientdet/d5/1',
'EfficientDet D6 1280x1280' : 'https://tfhub.dev/tensorflow/efficientdet/d6/1',
'EfficientDet D7 1536x1536' : 'https://tfhub.dev/tensorflow/efficientdet/d7/1',
'SSD MobileNet v2 320x320' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2',
'SSD MobileNet V1 FPN 640x640' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v1/fpn_640x640/1',
'SSD MobileNet V2 FPNLite 320x320' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_320x320/1',
```

```
'SSD MobileNet V2 FPNLite 640x640' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_640x640/1',
'SSD ResNet50 V1 FPN 640x640 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_640x640/1',
'SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_1024x1024/1',
'SSD ResNet101 V1 FPN 640x640 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/resnet101_v1_fpn_640x640/1',
'SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/resnet101_v1_fpn_1024x1024/1',
'SSD ResNet152 V1 FPN 640x640 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/resnet152_v1_fpn_640x640/1',
'SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/resnet152_v1_fpn_1024x1024/1',
'Faster R-CNN ResNet50 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1',
'Faster R-CNN ResNet50 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_1024x1024/1',
'Faster R-CNN ResNet50 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_800x1333/1',
'Faster R-CNN ResNet101 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_640x640/1',
'Faster R-CNN ResNet101 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_1024x1024/1',
'Faster R-CNN ResNet101 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_800x1333/1',
'Faster R-CNN ResNet152 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_640x640/1',
'Faster R-CNN ResNet152 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_1024x1024/1',
'Faster R-CNN ResNet152 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_800x1333/1',
'Faster R-CNN Inception ResNet V2 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/inception_resnet_v2_640x640/1',
'Faster R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/inception_resnet_v2_1024x1024/1',
'Mask R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1'
}

IMAGES_FOR_TEST = {
  'Beach' : 'models/research/object_detection/test_images/image2.jpg',
  'Dogs' : 'models/research/object_detection/test_images/image1.jpg',
  # By Heiko Gorski, Source: https://commons.wikimedia.org/wiki/File:Naxos_Taverna.jpg
  'Naxos Taverna' : 'https://upload.wikimedia.org/wikipedia/commons/6/60/Naxos_Taverna.jpg',
  # Source: https://commons.wikimedia.org/wiki/File:The_Coleoptera_of_the_British_islands_(Plate_125)_(8592917784).jpg
  'Beatles' : 'https://upload.wikimedia.org/wikipedia/commons/1/1b/The_Coleoptera_of_the_British_islands_%28Plate_125%29_%288592917784%29
  # By Américo Toledano, Source: https://commons.wikimedia.org/wiki/File:Biblioteca_Maim%C3%B3nides,_Campus_Universitario_de_Rabanales_00
  'Phones' : 'https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/Biblioteca_Maim%C3%B3nides%2C_Campus_Universitario_de_Rabanales_0
  # Source: https://commons.wikimedia.org/wiki/File:The_smaller_British_birds_(8053836633).jpg
  'Birds' : 'https://upload.wikimedia.org/wikipedia/commons/0/09/The_smaller_British_birds_%288053836633%29.jpg',
}

COCO17_HUMAN_POSE_KEYPOINTS = [(0, 1),
 (0, 2),
 (1, 3),
 (2, 4),
 (0, 5),
 (0, 6),
 (5, 7),
 (7, 9),
 (6, 8),
 (8, 10),
 (5, 6),
 (5, 11),
 (6, 12),
 (11, 12),
 (11, 13),
 (13, 15),
 (12, 14),
 (14, 16)]
```

Visualization tools

To visualize the images with the proper detected boxes, keypoints and segmentation, we will use the TensorFlow Object Detection API. To install it we will clone the repo.

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

```
Cloning into 'models'...
remote: Enumerating objects: 4056, done.
remote: Counting objects: 100% (4056/4056), done.
remote: Compressing objects: 100% (3083/3083), done.
remote: Total 4056 (delta 1185), reused 1939 (delta 916), pack-reused 0
Receiving objects: 100% (4056/4056), 54.70 MiB | 26.37 MiB/s, done.
Resolving deltas: 100% (1185/1185), done.
Updating files: 100% (3669/3669), done.
```

Intalling the Object Detection API

```
%%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: matplotlib in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (3.7.1)
Requirement already satisfied: Cython in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (3.0.5)
Requirement already satisfied: contextlib2 in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (21.6.0)
Requirement already satisfied: tf-slim in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (1.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (1.16.0)
Requirement already satisfied: pycocotools in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (2.0.7)
Collecting lvis (from object-detection==0.1)
  Downloading lvis-0.5.3-py3-none-any.whl (14 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (1.11.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (1.5.3)
Collecting tf-models-official>=2.5.1 (from object-detection==0.1)
  Downloading tf_models_official-2.15.0-py2.py3-none-any.whl (2.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 26.6 MB/s eta 0:00:00
Collecting tensorflow_io (from object-detection==0.1)
  Downloading tensorflow_io-0.34.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (28.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 28.8/28.8 MB 15.5 MB/s eta 0:00:00
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (2.14.0)
Collecting pyparsing==2.4.7 (from object-detection==0.1)
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.8/67.8 kB 1.8 MB/s eta 0:00:00
Collecting sacrebleu<=2.2.0 (from object-detection==0.1)
  Downloading sacrebleu-2.2.0-py3-none-any.whl (116 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.6/116.6 kB 7.6 MB/s eta 0:00:00
Collecting portalocker (from sacrebleu<=2.2.0->object-detection==0.1)
  Downloading portalocker-2.8.2-py3-none-any.whl (17 kB)
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packages (from sacrebleu<=2.2.0->object-detection==0.1) (2
Requirement already satisfied: tabulate>=0.8.9 in /usr/local/lib/python3.10/dist-packages (from sacrebleu<=2.2.0->object-detectio
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from sacrebleu<=2.2.0->object-detection==0
Collecting colorama (from sacrebleu<=2.2.0->object-detection==0.1)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Requirement already satisfied: gin-config in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->object-dete
Requirement already satisfied: google-api-python-client>=1.6.7 in /usr/local/lib/python3.10/dist-packages (from tf-models-officia
Collecting immutabledict (from tf-models-official>=2.5.1->object-detection==0.1)
  Downloading immutabledict-3.0.0-py3-none-any.whl (4.0 kB)
Requirement already satisfied: kaggle>=1.3.9 in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->object-d
Requirement already satisfied: oauth2client in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->object-de
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1-
Requirement already satisfied: psutil>=5.4.3 in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->object-d
Requirement already satisfied: py-cpuinfo>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->obje
Requirement already satisfied: pyyaml>=6.0.0 in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->object-d
Collecting sentencepiece (from tf-models-official>=2.5.1->object-detection==0.1)
  Downloading sentencepiece-0.1.99-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 23.1 MB/s eta 0:00:00
Collecting seqeval (from tf-models-official>=2.5.1->object-detection==0.1)
  Downloading seqeval-1.2.2.tar.gz (43 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.6/43.6 kB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: tensorflow-datasets in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->ob
Requirement already satisfied: tensorflow-hub>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from tf-models-official>=2.5.1->
Collecting tensorflow-model-optimization>=0.4.1 (from tf-models-official>=2.5.1->object-detection==0.1)
  Downloading tensorflow_model_optimization-0.7.5-py2.py3-none-any.whl (241 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 241.2/241.2 kB 20.6 MB/s eta 0:00:00
Collecting tensorflow-text~=2.15.0 (from tf-models-official>=2.5.1->object-detection==0.1)
  Downloading tensorflow_text-2.15.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.2/5.2 MB 45.3 MB/s eta 0:00:00
Collecting tensorflow~=2.15.0 (from tf-models-official>=2.5.1->object-detection==0.1)
```

Now we can import the dependencies we will need later

```
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

%matplotlib inline
```

Load label map data (for plotting).

Label maps correspond index numbers to category names, so that when our convolution network predicts 5, we know that this corresponds to airplane. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine.

We are going, for simplicity, to load from the repository that we loaded the Object Detection API code

```
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)
```

Build a detection model and load pre-trained model weights

Here we will choose which Object Detection model we will use. Select the architecture and it will be loaded automatically. If you want to change the model to try other architectures later, just change the next cell and execute following ones.

Tip: if you want to read more details about the selected model, you can follow the link (model handle) and read additional documentation on TF Hub. After you select a model, we will print the handle to make it easier.

Model Selection

model_display_name: Mask R-CNN Inception ResNet V2 1024x1024

Mostrar código

```
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
```

Loading the selected model from TensorFlow Hub

Here we just need the model handle that was selected and use the Tensorflow Hub library to load it to memory.

```
print('loading model...')
hub_model = hub.load(model_handle)
print('model loaded!')
```

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

Loading an image

Let's try the model on a simple image. To help with this, we provide a list of test images.

Here are some simple things to try out if you are curious:

```
Try running inference on your own images, just upload them to colab and load the same way it's done in the cell below.
Modify some of the input images and see if detection still works. Some simple things to try out here include flipping the image horizontally, or c
```

Be careful: when using images with an alpha channel, the model expect 3 channels images and the alpha will count as a 4th.

Image Selection (don't forget to execute the cell!)

```
#@title Image Selection (don't forget to execute the cell!) { display-m
selected_image = 'Beach' # @param ['Beach', 'Dogs', 'Naxos Taverna', 'B
flip_image_horizontally = False #@param {type:"boolean"}
convert_image_to_grayscale = False #@param {type:"boolean"}

image_path = IMAGES_FOR_TEST[selected_image]
image_np = load_image_into_numpy_array(image_path)

# Flip horizontally
if(flip_image_horizontally):
  image_np[0] = np.fliplr(image_np[0]).copy()

# Convert image to grayscale
if(convert_image_to_grayscale):
  image_np[0] = np.tile(
    np.mean(image_np[0], 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

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

selected_image: Beach

flip_image_horizontally:

convert_image_to_grayscale:

Doing the inference

To do the inference we just need to call our TF Hub loaded model.

Things you can try:

```
Print out result['detection_boxes'] and try to match the box locations to the boxes in the image. Notice that coordinates are given in normalized
inspect other output keys present in the result. A full documentation can be seen on the models documentation page (pointing your browser to the m
```

Declarando uma variável para calcular o tempo de inferência e guardar em um vetor

```
import timeit

resultados = []

#inicio = timeit.default_timer()
#alguma_funcao()
#fim = timeit.default_timer()
#print ('duracao: %f' % (fim - inicio))
```

```
# running inference

inicio = timeit.default_timer() # inicio da contagem do tempo

results = hub_model(image_np)

# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}


fim = timeit.default_timer() #Tempo final

resultados.append(fim - inicio) #Salvando tempo de inferência na variavel resultados

print ('duracao: %f' % (fim - inicio))

print(result.keys())
```

```
duracao: 68.791482
dict_keys(['anchors', 'rpn_box_predictor_features', 'detection_scores', 'raw_detection_boxes', 'proposal_boxes', 'detection_anchor_i
```

Visualizing the results

Here is where we will need the TensorFlow Object Detection API to show the squares from the inference step (and the keypoints when available).

the full documentation of this method can be seen here

Here you can, for example, set min_score_thresh to other values (between 0 and 1) to allow more detections in or to filter out more detections.

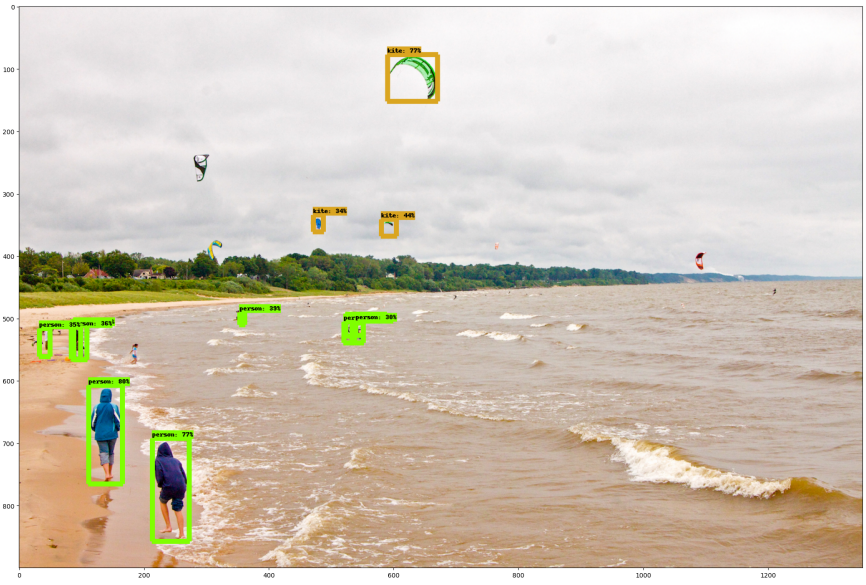
```
label_id_offset = 0
image_np_with_detections = image_np.copy()

# Use keypoints if available in detections
keypoints, keypoint_scores = None, None
if 'detection_keypoints' in result:
  keypoints = result['detection_keypoints'][0]
  keypoint_scores = result['detection_keypoint_scores'][0]

viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_detections[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=200,
      min_score_thresh=.30,
      agnostic_mode=False,
      keypoints=keypoints,
      keypoint_scores=keypoint_scores,
      keypoint_edges=COCO17_HUMAN_POSE_KEYPOINTS)

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

[Optional]

Among the available object detection models there's Mask R-CNN and the output of this model allows instance segmentation.

To visualize it we will use the same method we did before but adding an aditional parameter:
instance_masks=output_dict.get('detection_masks_reframed', None)

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

if 'detection_masks' in result:
  # we need to 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 bbox 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])
  detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                     tf.uint8)
  result['detection_masks_reframed'] = detection_masks_reframed.numpy()

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=200,
      min_score_thresh=.30,
      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()
```

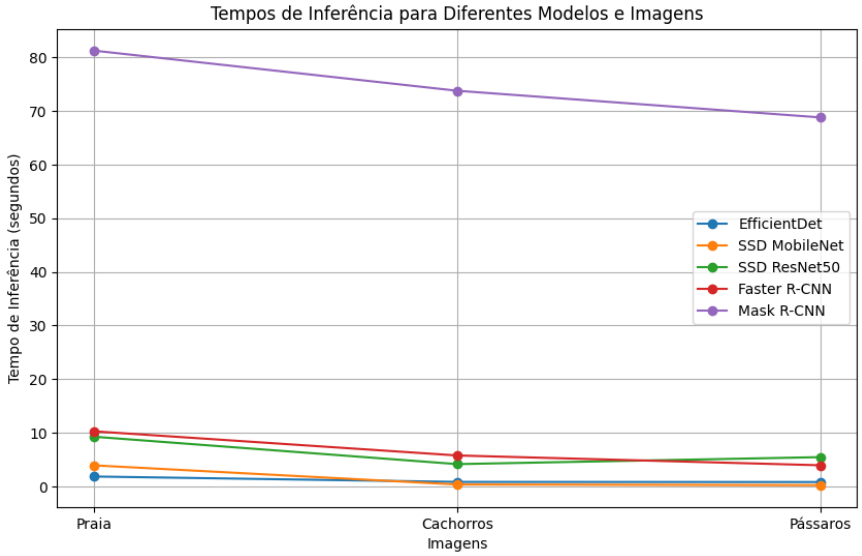


ToDo : Custo computacional (30pt)

Compute o custo computacional (tempo de inferência) de cada modelo acima

Dica : Use o método "default_timer" da biblioteca "timeit"

```
#exemplo de uso da timeit
#import timeit

#inicio = timeit.default_timer()
#alguma_funcao()
#fim = timeit.default_timer()
import matplotlib.pyplot as plt
import numpy as np
from timeit import default_timer as timer
import tensorflow as tf
import cv2

#print ('duracao: %f' % (fim - inicio))
# Plotando os resultados


modelos = ['EfficientDet', 'SSD MobileNet', 'SSD ResNet50', 'Faster R-CNN', 'Mask R-CNN']
imagens = ['Praia', 'Cachorros', 'Pássaros']

# Criando uma matriz 2D para os tempos de inferência
tempos_matriz = np.array(resultados).reshape(len(modelos), len(imagens))

# Plotando o gráfico
plt.figure(figsize=(10, 6))
for i, modelo in enumerate(modelos):
    plt.plot(imagens, tempos_matriz[i], label=modelo, marker='o')

plt.title('Tempos de Inferência para Diferentes Modelos e Imagens')
plt.xlabel('Imagens')
plt.ylabel('Tempo de Inferência (segundos)')
plt.legend()
plt.grid(True)
plt.show()
```



ToDo : YoloV3 (30pt)

Carregue o YoloV3 pré-treinado (ver link) e execute a inferência nas mesmas imagens testadas com os modelos acima. Calule o custo computacional e compare contra os modelos acima. Qual a sua conclusão? Justifique.

O modelo YoloV3 é o mais rápido de todos testados até agora no quesito tempo computacional.

ToDo : Detectando objetos com dados próprios (Opcional / 20 Pontos Extra)

1 célula oculta

Part II - Segmentação (40pt)

ToDo : Rodando um tutorial (15pt)

Estude o tutorial do link. Rode o código e veja o resultado.

```
pip install -q git+https://github.com/tensorflow/examples.git
```

```
  Preparing metadata (setup.py) ... done
  Building wheel for tensorflow-examples (setup.py) ... done
```

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

```
from __future__ import absolute_import, division, print_function, unicode_literals

from tensorflow_examples.models.pix2pix import pix2pix

import tensorflow_datasets as tfds
tfds.disable_progress_bar()

from IPython.display import clear_output
import matplotlib.pyplot as plt
```

```
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

```
Downloading and preparing dataset 773.52 MiB (download: 773.52 MiB, generated: 774.69 MiB, total: 1.51 GiB) to /root/tensorflow_data
Dataset oxford_iiit_pet downloaded and prepared to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0. Subsequent calls will reuse this
```

```
def normalize(input_image, input_mask):
  input_image = tf.cast(input_image, tf.float32) / 255.0
  input_mask -= 1
  return input_image, input_mask
```

```
@tf.function
def load_image_train(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  if tf.random.uniform(()) > 0.5:
    input_image = tf.image.flip_left_right(input_image)
    input_mask = tf.image.flip_left_right(input_mask)

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask
```

```
def load_image_test(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask
```

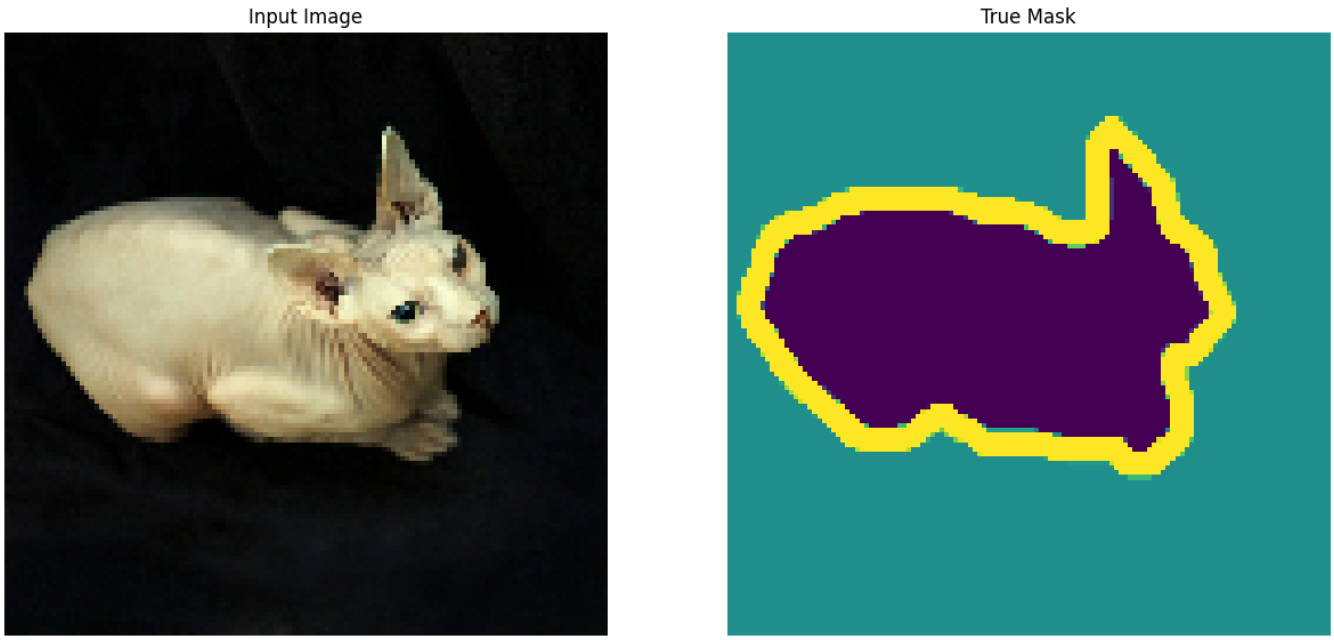
```
TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 64
BUFFER_SIZE = 1000
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
```

```
train = dataset['train'].map(load_image_train, num_parallel_calls=tf.data.experimental.AUTOTUNE)
test = dataset['test'].map(load_image_test)
```

```
train_dataset = train.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
test_dataset = test.batch(BATCH_SIZE)
```

```
def display(display_list):
  plt.figure(figsize=(15, 15))

  title = ['Input Image', 'True Mask', 'Predicted Mask']

  for i in range(len(display_list)):
    plt.subplot(1, len(display_list), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.preprocessing.image.array_to_img(display_list[i]))
    plt.axis('off')
  plt.show()
```

```
for image, mask in train.take(1):
  sample_image, sample_mask = image, mask
display([sample_image, sample_mask])
```




```
OUTPUT_CHANNELS = 3
```

```
base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], include_top=False)

# Use as ativações dessas camadas
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
layers = [base_model.get_layer(name).output for name in layer_names]

# Crie o modelo de extração de características
down_stack = tf.keras.Model(inputs=base_model.input, outputs=layers)

down_stack.trainable = False
```

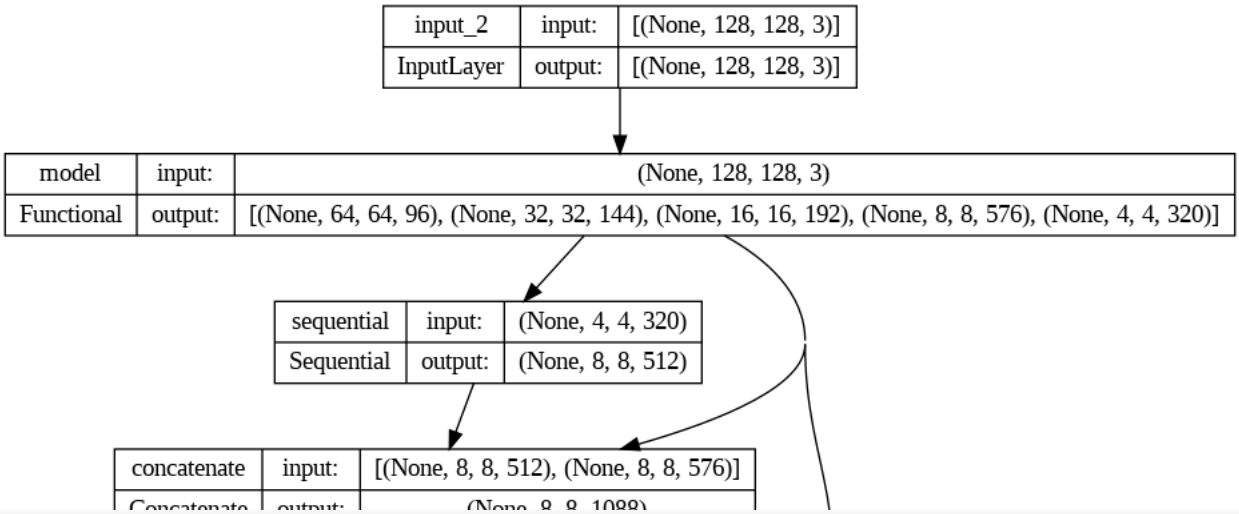
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering
9406464/9406464 [==============================] - 0s 0us/step
```

```
up_stack = [
    pix2pix.upsample(512, 3),  # 4x4 -> 8x8
    pix2pix.upsample(256, 3),  # 8x8 -> 16x16
    pix2pix.upsample(128, 3),  # 16x16 -> 32x32
    pix2pix.upsample(64, 3),   # 32x32 -> 64x64
]
```

```
def unet_model(output_channels):

  # Esta é a última camada do modelo
  last = tf.keras.layers.Conv2DTranspose(
      output_channels, 3, strides=2,
      padding='same', activation='softmax')  #64x64 -> 128x128

  inputs = tf.keras.layers.Input(shape=[128, 128, 3])
  x = inputs

  # Downsampling através do modelo
  skips = down_stack(x)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling e estabelecimento das conexões de salto
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

```
model = unet_model(OUTPUT_CHANNELS)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
tf.keras.utils.plot_model(model, show_shapes=True)
```

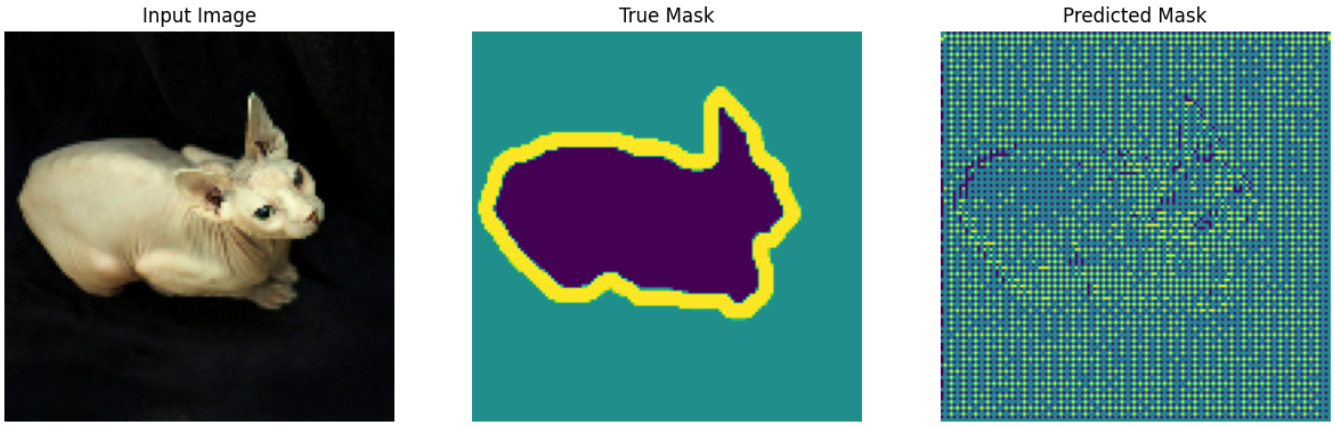



```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]
```

```
def show_predictions(dataset=None, num=1):
  if dataset:
    for image, mask in dataset.take(num):
      pred_mask = model.predict(image)
      display([image[0], mask[0], create_mask(pred_mask)])
  else:
    display([sample_image, sample_mask,
             create_mask(model.predict(sample_image[tf.newaxis, ...]))])
```

```
show_predictions()
```

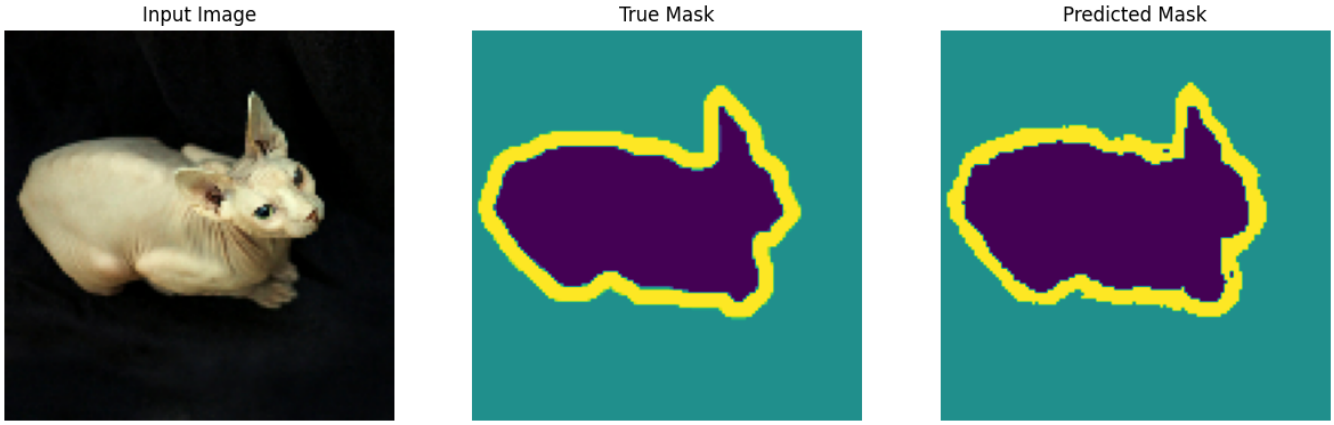
```
1/1 [==============================] - 1s 1s/step
```




```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    clear_output(wait=True)
    show_predictions()
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))
```

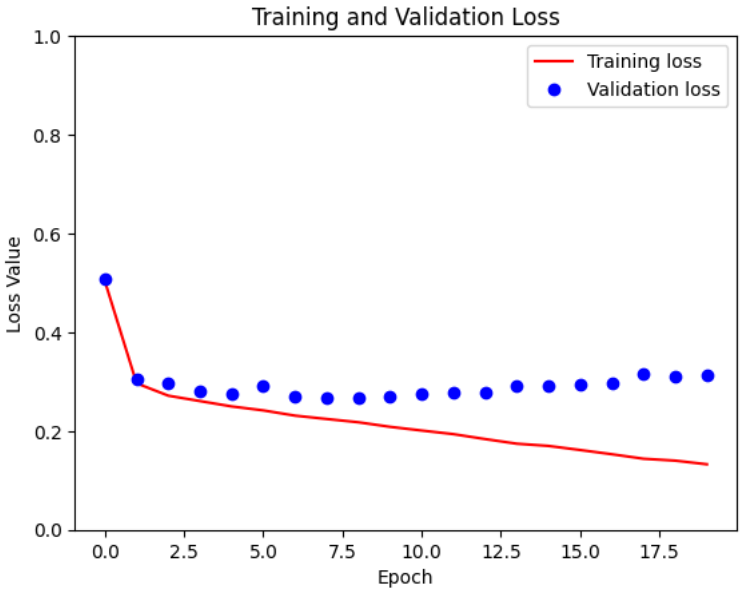
```
EPOCHS = 20
VAL_SUBSPLITS = 5
VALIDATION_STEPS = info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLITS

model_history = model.fit(train_dataset, epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_steps=VALIDATION_STEPS,
                          validation_data=test_dataset,
                          callbacks=[DisplayCallback()])
```

```
1/1 [==============================] - 0s 64ms/step
```



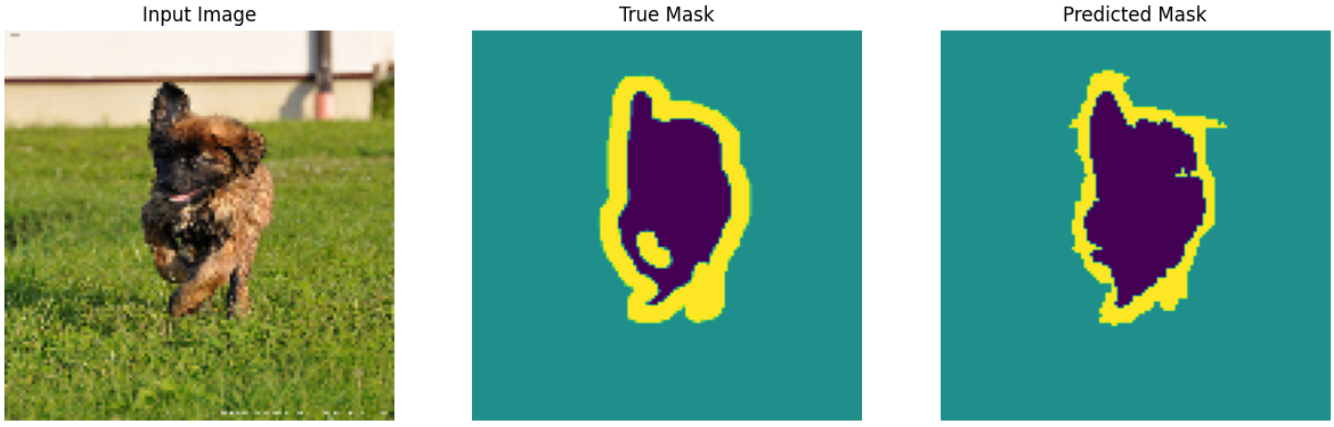

```
Sample Prediction after epoch 20
```

```
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']

epochs = range(EPOCHS)

plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'bo', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.ylim([0, 1])
plt.legend()
plt.show()
```

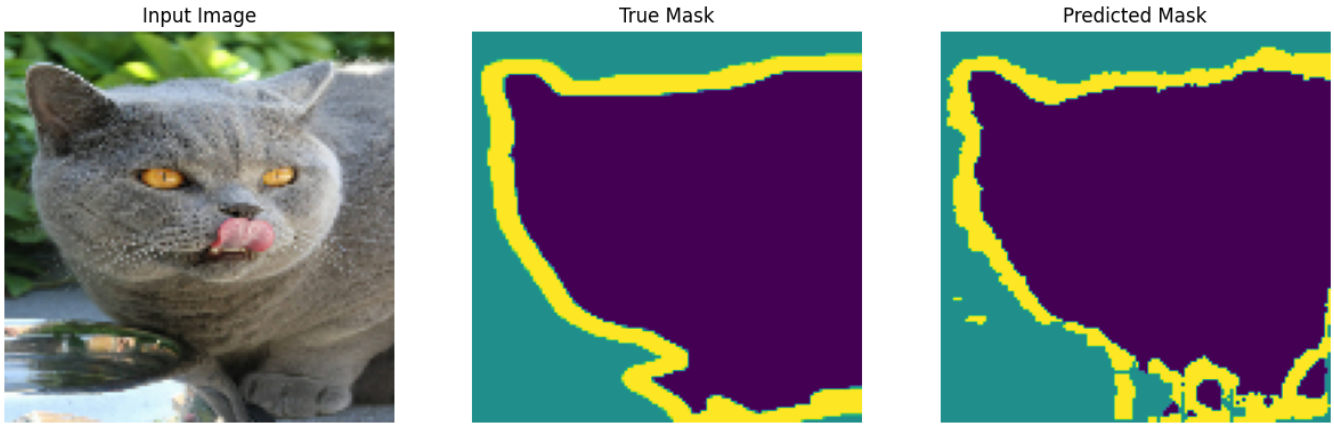



```
show_predictions(test_dataset, 3)
```

```
2/2 [==============================] - 3s 1s/step
```




```
2/2 [==============================] - 2s 1s/step
```




```
2/2 [==============================] - 2s 1s/step
```

ToDo : Melhorando o modelo (25pt)

Use das operações que você conhece para construir uma rede melhor:

- Dropout
- Convolution2D, Dense, (várias funções de ativação como GeLu, LeakyReLu, etc)
- Flatten, GlobalAveragePooling2D, GlobalMaxPooling2D, etc.
- Use outras arquiteturas para o encoder: inception, Xception, VGG16, EfficientNet