Lab 6 - 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 : 30/09

- 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
- Pássaros

Importações e configurações

```
# This Colab requires TF 2.5.
!pip install -U "tensorflow>=2.5"
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
```

```
Requirement already satisfied: tensorflow>=2.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/li
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.
Requirement already satisfied: keras<2.11,>=2.10.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: tensorboard<2.11,>=2.10 in /usr/local/lib/python3.7/d
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: tensorflow-estimator<2.11,>=2.10.0 in /usr/local/lib/
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/py
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/l
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/pytho
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/d
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/
```

```
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')
```

Utilidades

Execute a seguinte célula para criar uma função necessária.

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_512x5
'CenterNet HourGlass104 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/hourg
'CenterNet HourGlass104 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hourglass_102
'CenterNet HourGlass104 Keypoints 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hou
'CenterNet Resnet50 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v1_f
'CenterNet Resnet50 V1 FPN Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/re
'CenterNet Resnet101 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet101v1
```

```
  'CenterNet Resnet50 V2 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v2_512x5
  'CenterNet Resnet50 V2 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet
  '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_640x64
  'SSD MobileNet V2 FPNLite 320x320' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlit
  'SSD MobileNet V2 FPNLite 640x640' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlit
  'SSD ResNet50 V1 FPN 640x640 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/resr
  'SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/re
  'SSD ResNet101 V1 FPN 640x640 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/re
  'SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/
  'SSD ResNet152 V1 FPN 640x640 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/re
  'SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/
  'Faster R-CNN ResNet50 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1
  'Faster R-CNN ResNet50 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_
  'Faster R-CNN ResNet50 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v
  'Faster R-CNN ResNet101 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_
  'Faster R-CNN ResNet101 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet10
  'Faster R-CNN ResNet101 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101
  'Faster R-CNN ResNet152 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_
  'Faster R-CNN ResNet152 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet15
  'Faster R-CNN ResNet152 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152
  'Faster R-CNN Inception ResNet V2 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/inc
  'Faster R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/i
  'Mask R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/mask_rcnn/incep
  }

  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_
    'Beatles' : 'https://upload.wikimedia.org/wikipedia/commons/1/1b/The_Coleoptera_of_the_B
    # By Américo Toledano, Source: https://commons.wikimedia.org/wiki/File:Biblioteca_Maim%C
    'Phones' : 'https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/Biblioteca_Maim%C3
    # Source: https://commons.wikimedia.org/wiki/File:The_smaller_British_birds_(8053836633)
    'Birds' : 'https://upload.wikimedia.org/wikipedia/commons/0/09/The_smaller_British_birds
  }

  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)]
```

```
# Dicionário para medir o custo computacional
import timeit
"""#exemplo de uso da timeit
inicio = timeit.default_timer()
alguma_funcao()
fim = timeit.default_timer()
print ('duracao: %f' % (fim - inicio))"""
dicionario_de_tempo = {"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": {}}
```

Ferramentas de Visualização

Para visualizar as imagens com as caixas detectadas, pontos-chave e segmentação adequados, usaremos a API TensorFlow Object Detection. Para instalá-lo vamos clonar o repositório.

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

```
fatal: destination path 'models' already exists and is not an empty directory.
```

Instalando a API de detecção de objetos

```
%%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: tensorflow-estimator<2.11,>=2.10.0 in /usr/local/li
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python
```

```
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/pyt
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: dm-tree~=0.1.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pydot<2,>=1.2.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: pymongo<4.0.0,>=3.8.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: fastavro<2,>=0.23.6 in /usr/local/lib/python3.7/dis
Requirement already satisfied: cloudpickle<3,>=2.1.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: hdfs<3.0.0,>=2.1.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: proto-plus<2,>=1.7.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: dill<0.3.2,>=0.3.1.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: crcmod<2.0,>=1.7 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyarrow<8.0.0,>=0.15.1 in /usr/local/lib/python3.7/
Requirement already satisfied: orjson<4.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: docopt in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: opencv-python>=4.1.0.25 in /usr/local/lib/python3.7
Requirement already satisfied: cycler>=0.10.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: kiwisolver>=1.1.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: scikit-learn>=0.21.3 in /usr/local/lib/python3.7/di
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: toml in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.7/dis
Requirement already satisfied: promise in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.7/dis
Requirement already satisfied: etils[epath] in /usr/local/lib/python3.7/dist-packa
Building wheels for collected packages: object-detection
  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.w
  Stored in directory: /tmp/pip-ephem-wheel-cache-hs00ghqr/wheels/fa/a4/d2/e9a5057
Successfully built object-detection
Installing collected packages: object-detection
  Attempting uninstall: object-detection
    Found existing installation: object-detection 0.1
    Uninstalling object-detection-0.1:
      Successfully uninstalled object-detection-0.1
Successfully installed object-detection-0.1

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-plac
   pip 21.3 will remove support for this functionality. You can find discussion re
```

Agora podemos importar as dependências que precisaremos mais tarde

```
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
```

Carregar dados do mapa de rótulos (para plotagem).

```
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_di
```

Crie um modelo de detecção e carregue pesos de modelo pré-treinados

Escolher a Arquitetura Utilizada

EfficientDet D0 512x512

Model Selection

model_display_name: EfficientDet D0 512x512

Mostrar código

```
Selected model:EfficientDet D0 512x512
Model Handle at TensorFlow Hub: https://tfhub.dev/tensorflow/efficientdet/d0/1
```

Carregando o modelo selecionado do TensorFlow Hub

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

```
loading model...
WARNING:absl:Importing a function (__inference___call___32344) with ops with unsaved
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return
WARNING:absl:Importing a function (__inference_bifpn_layer_call_and_return_condition
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return
```

```
WARNING:absl:Importing a function (__inference_bifpn_layer_call_and_return_condition
model loaded!
```

Carregando uma imagem

Praia

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

selected_image: Beach

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['raw_detection_scores', 'detection_boxes', 'detection_classes', 'raw_dete
```

Visualizando os resultados

```
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()
```

Alguns objetos não foram detectados

Carregando uma imagem

Cachorro

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

selected_image: Dogs

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

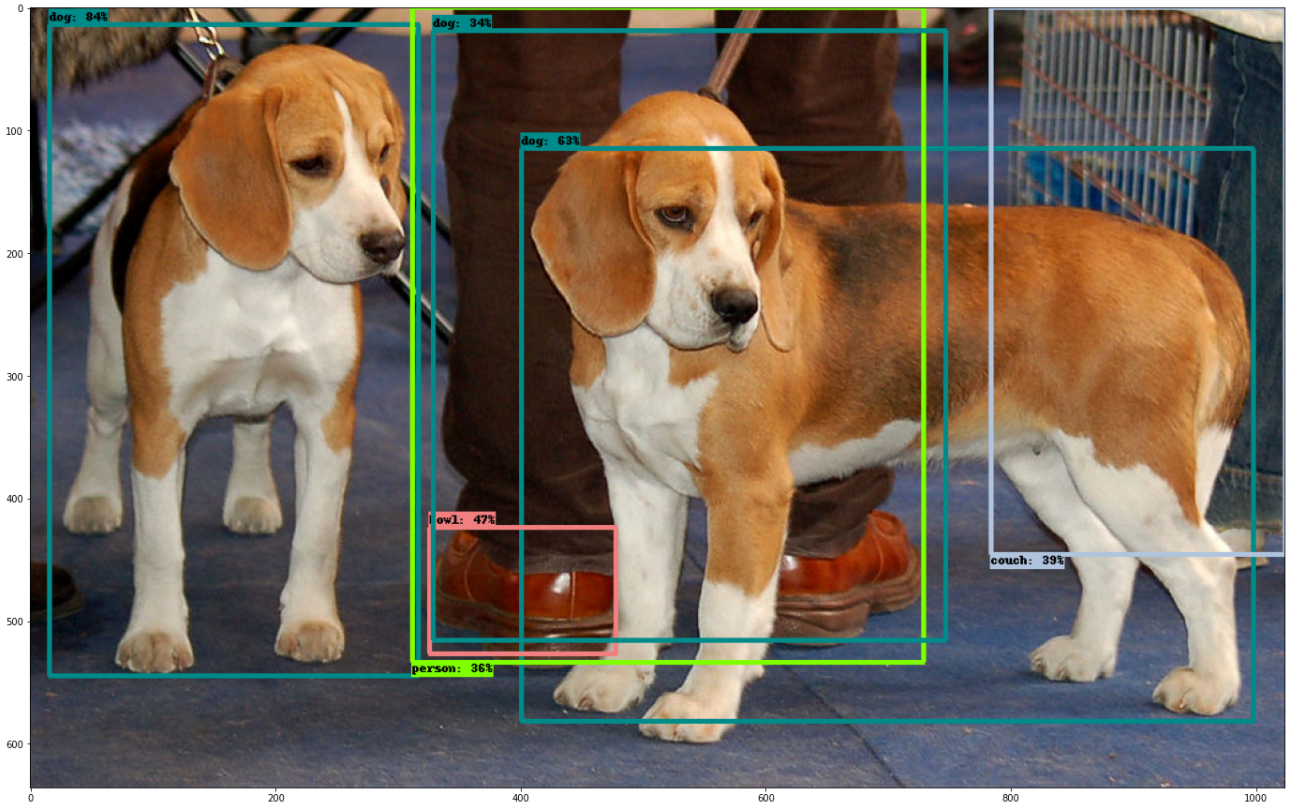
```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
dict_keys(['raw_detection_scores', 'detection_boxes', 'detection_classes', 'raw_dete
```

Visualizando os resultados

```
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()
```



Detectou bem os cachorros.

Carregando uma imagem

Passáros

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

selected_image: Birds

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

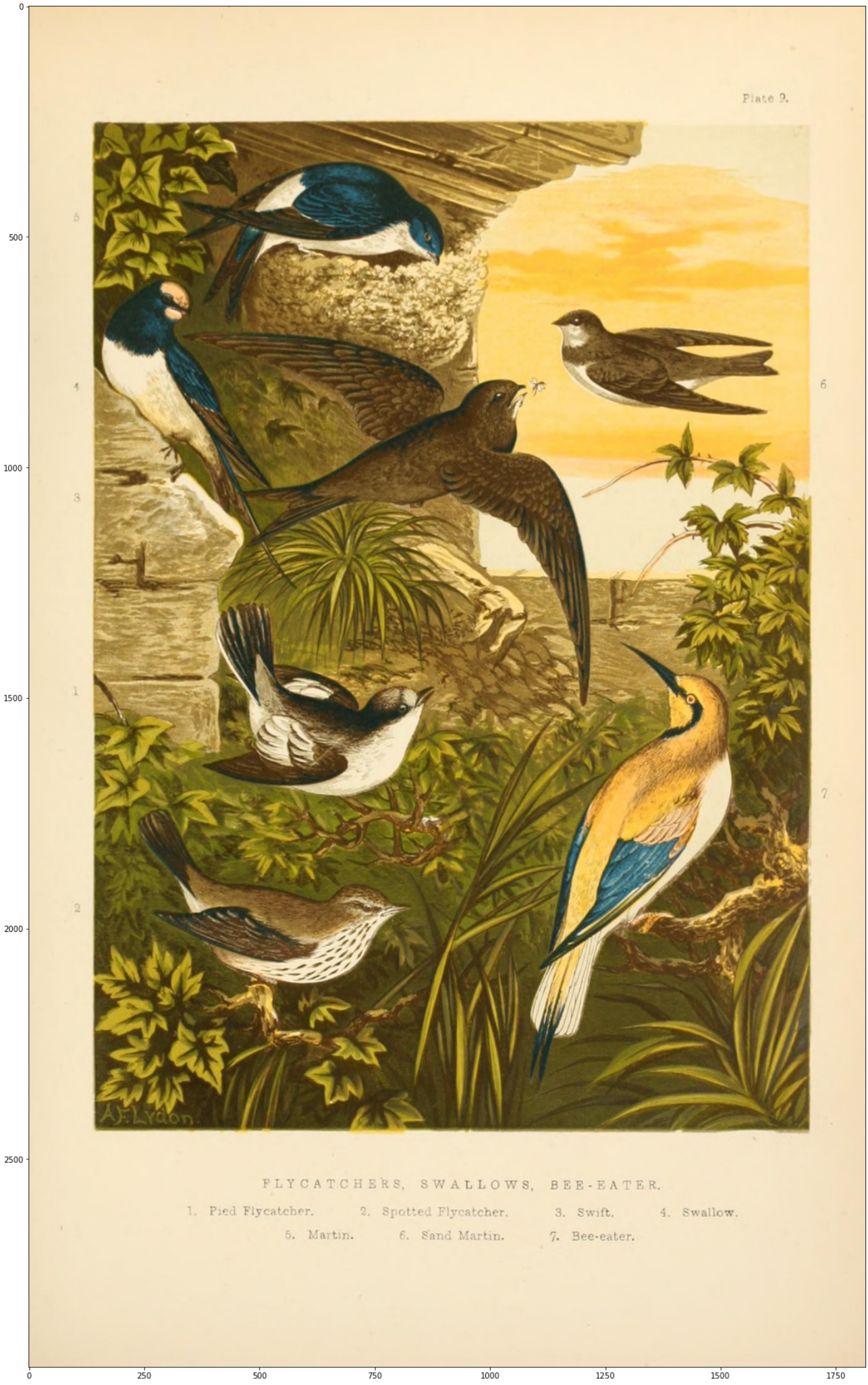


Plate 9.
A.F. Lydon.
FLYCATCHERS, SWALLOWS, BEE-EATER.
1. Pied Flycatcher. 2. Spotted Flycatcher. 3. Swift. 4. Swallow.
5. Martin. 6. Sand Martin. 7. Bee-eater.
0
500
1000
1500
2000
2500
0
250
500
750
1000
1250
1500
1750

Fazendo a inferência

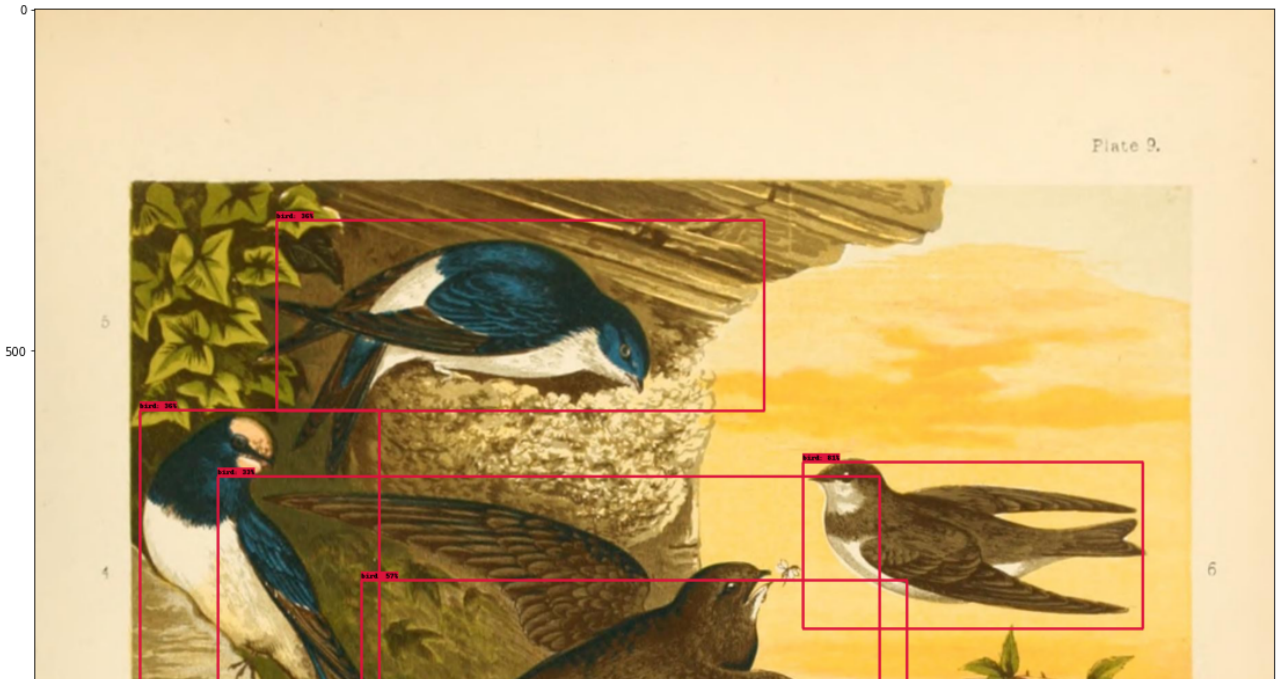
```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
dict_keys(['raw_detection_scores', 'detection_boxes', 'detection_classes', 'raw_dete
```

Visualizando os resultados

```
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()
```



Conseguiu detectar bem os pássaros.

SSD MobileNet V2 FPNLite 320x320

Model Selection

model_display_name: SSD MobileNet V2 FPNLite 320x320

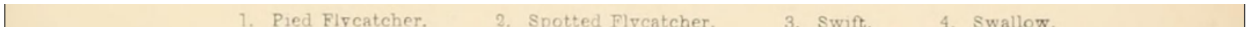
Mostrar código

```
Selected model:SSD MobileNet V2 FPNLite 320x320
Model Handle at TensorFlow Hub: https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlit
```

Carregando o modelo selecionado do TensorFlow Hub

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

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



Carregando uma imagem

Praia

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

selected_image: Beach

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['detection_anchor_indices', 'raw_detection_scores', 'num_detections', 'de
```

Visualizando os resultados

```
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()
```

Alguns objetos não foram detectados

Carregando uma imagem

Cachorro

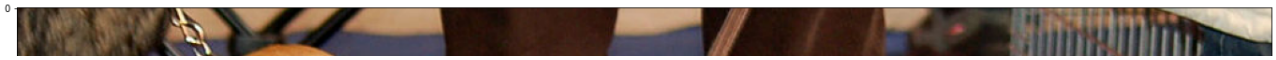
Image Selection (don't forget to execute the cell!)

selected_image: Dogs

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código



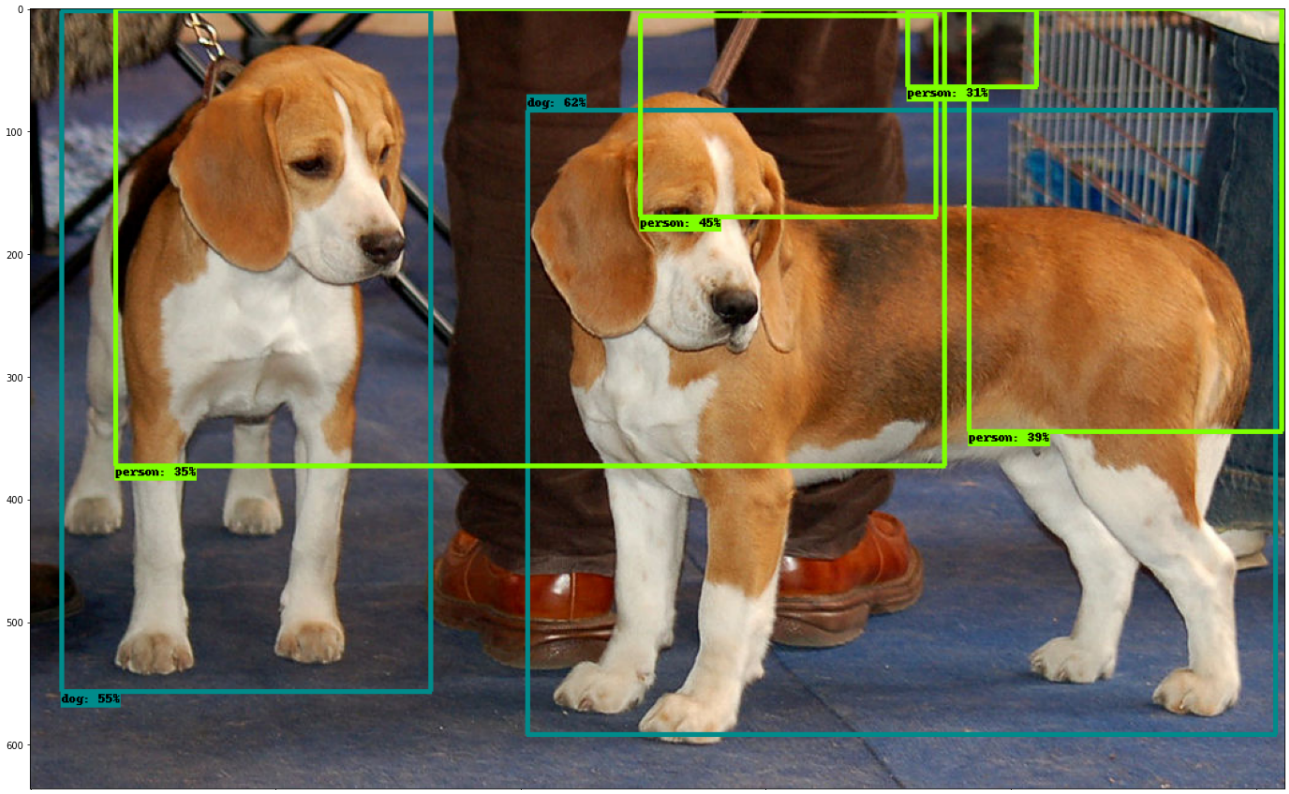
Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['detection_anchor_indices', 'raw_detection_scores', 'num_detections', 'de
```

Visualizando os resultados

```
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()
```



Detectou bem os cachorros.

Carregando uma imagem

Passáros

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

selected_image: Birds

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

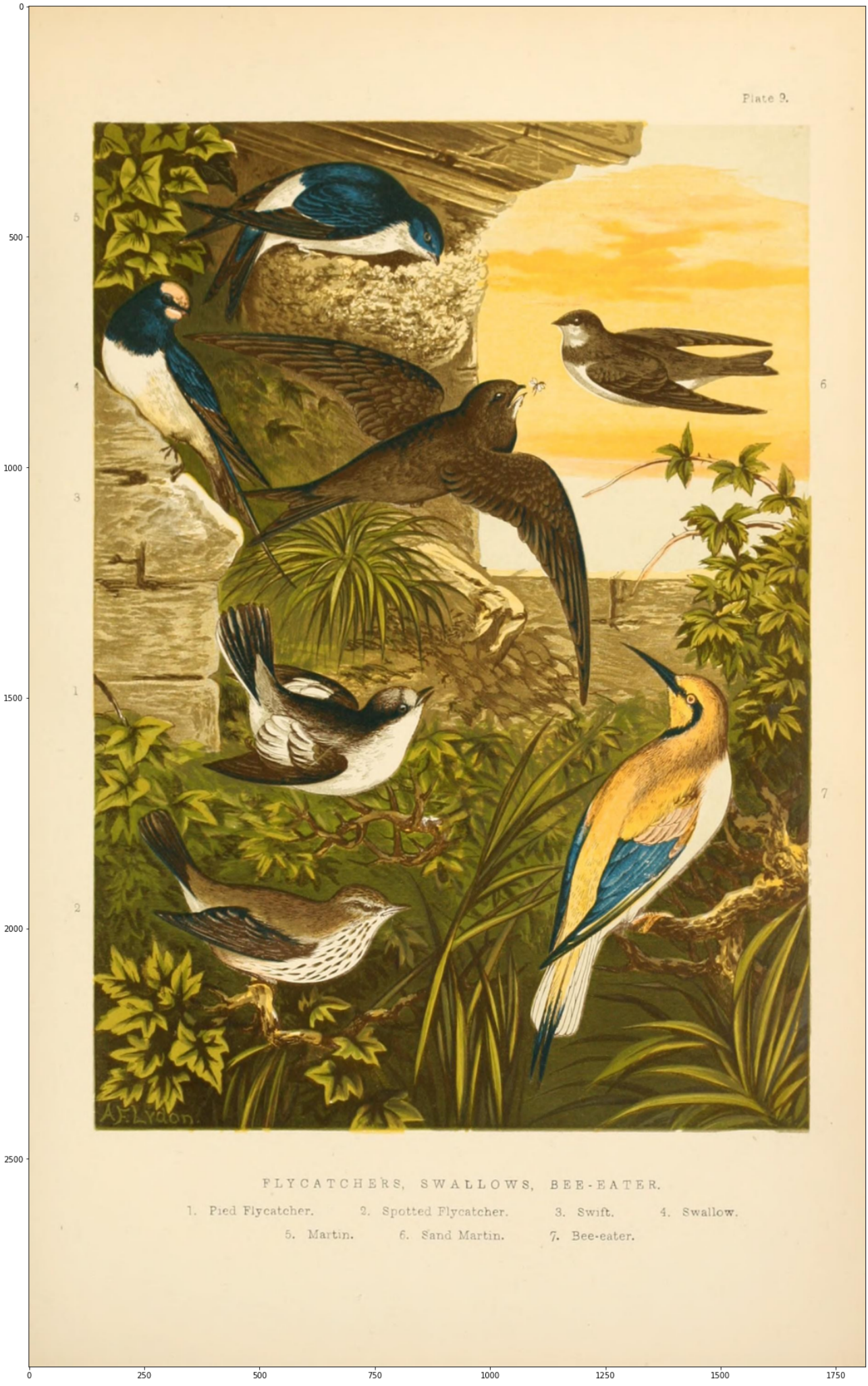


Plate 9.
A.F. Lydon.
FLYCATCHERS, SWALLOWS, BEE-EATER.
1. Pied Flycatcher. 2. Spotted Flycatcher. 3. Swift. 4. Swallow.
5. Martin. 6. Sand Martin. 7. Bee-eater.
0
500
1000
1500
2000
2500
0
250
500
750
1000
1250
1500
1750

Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['detection_anchor_indices', 'raw_detection_scores', 'num_detections', 'de
```

Visualizando os resultados

```
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()
```

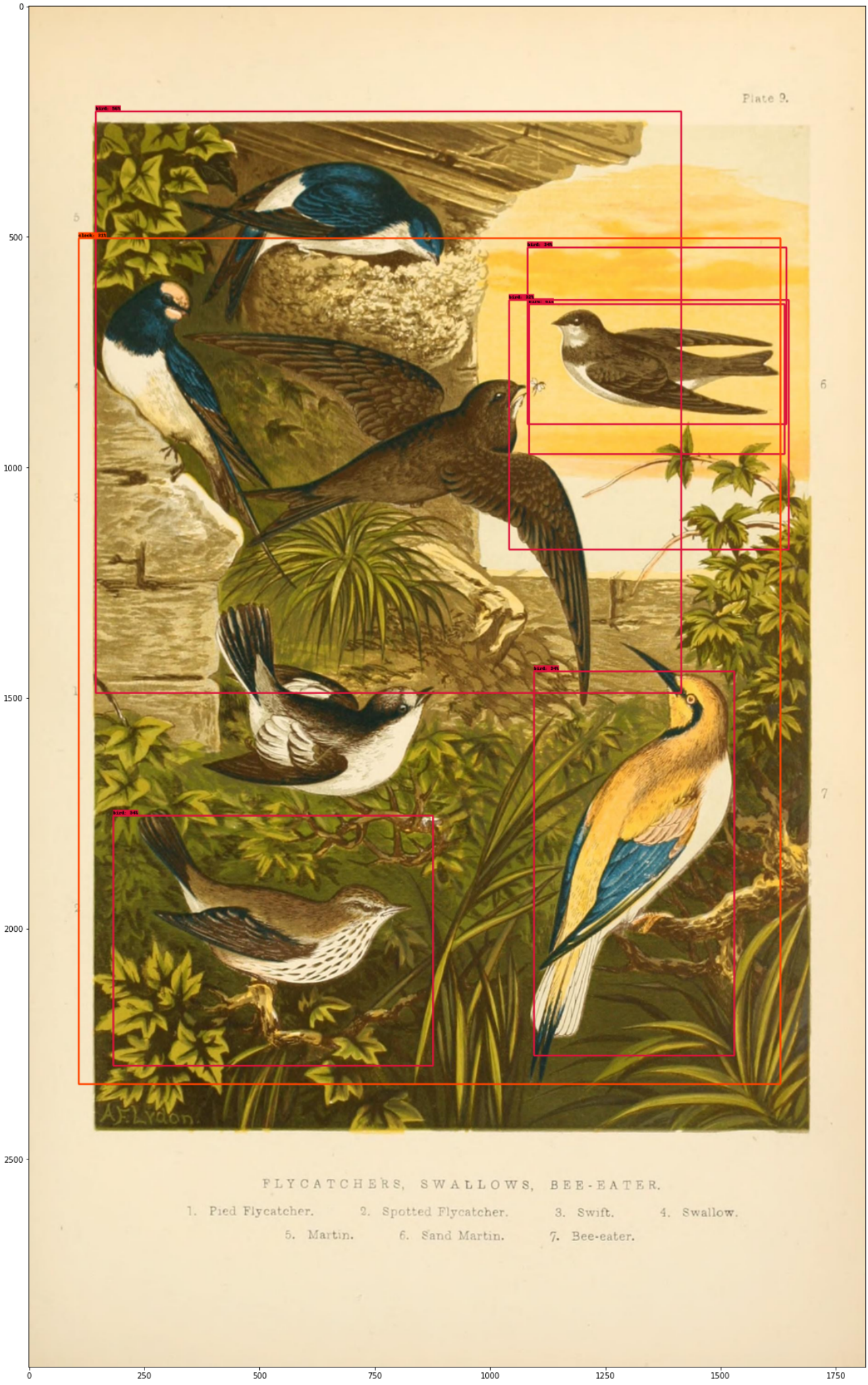


Plate 9.
FLYCATCHERS, SWALLOWS, BEE-EATER.
1. Pied Flycatcher. 2. Spotted Flycatcher. 3. Swift. 4. Swallow.
5. Martin. 6. Sand Martin. 7. Bee-eater.
A.F. Lydon.
0
500
1000
1500
2000
2500
0
250
500
750
1000
1250
1500
1750

Conseguiu detectar bem os pássaros, mas não separadamente alguns.

SSD ResNet50 V1 FPN 640x640 (RetinaNet50)

Model Selection

model_display_name: SSD ResNet50 V1 FPN 640x640 (RetinaNet50)

Mostrar código

```
Selected model:SSD ResNet50 V1 FPN 640x640 (RetinaNet50)
Model Handle at TensorFlow Hub: https://tfhub.dev/tensorflow/retinanet/resnet50_v1_f
```

Carregando o modelo selecionado do TensorFlow Hub

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

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

Carregando uma imagem

Praia

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

selected_image: Beach

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['detection_multiclass_scores', 'raw_detection_scores', 'detection_scores'
```

Visualizando os resultados

```
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()
```

Conseguiu detectar bem os objetos, mas alguns objetos não foram detectados

Carregando uma imagem

Cachorro

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

selected_image: Dogs

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

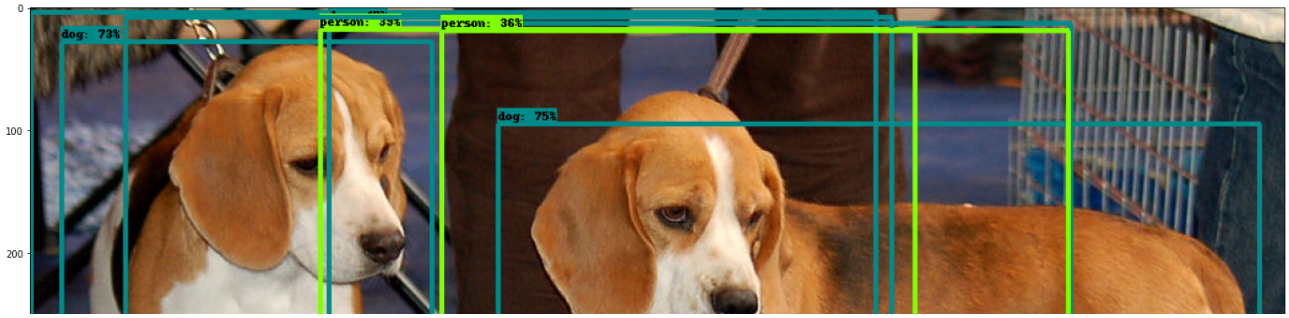
```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
```

```
print(result.keys())
```

```
dict_keys(['detection_multiclass_scores', 'raw_detection_scores', 'detection_scores'
```

Visualizando os resultados

```
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()
```



Detectou bem os cachorros.

Carregando uma imagem

Passáros

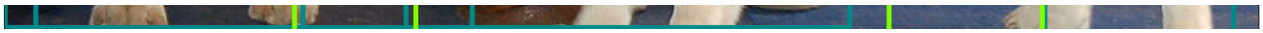


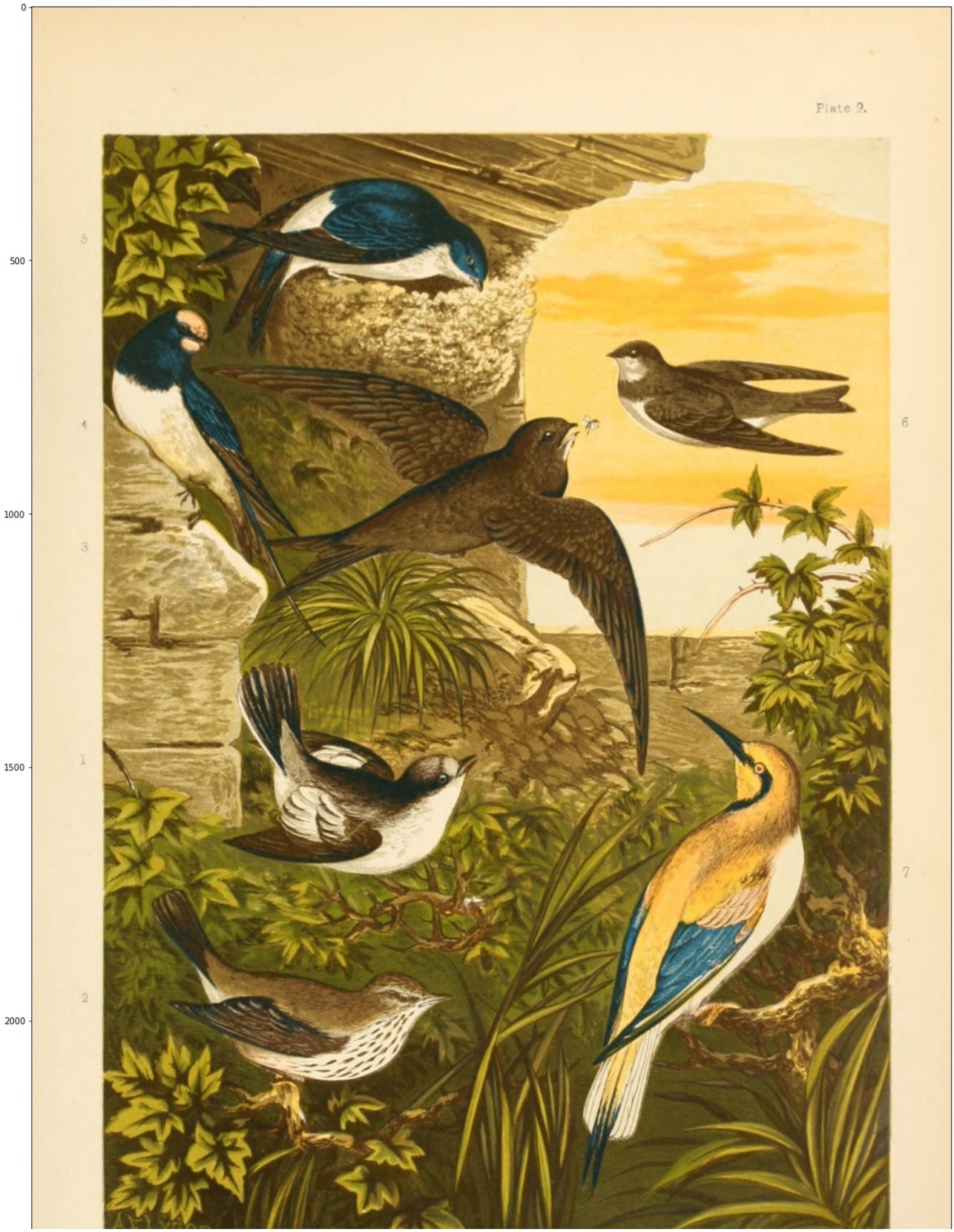
Image Selection (don't forget to execute the cell!)

selected_image: Birds

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código



Fazendo a inferência

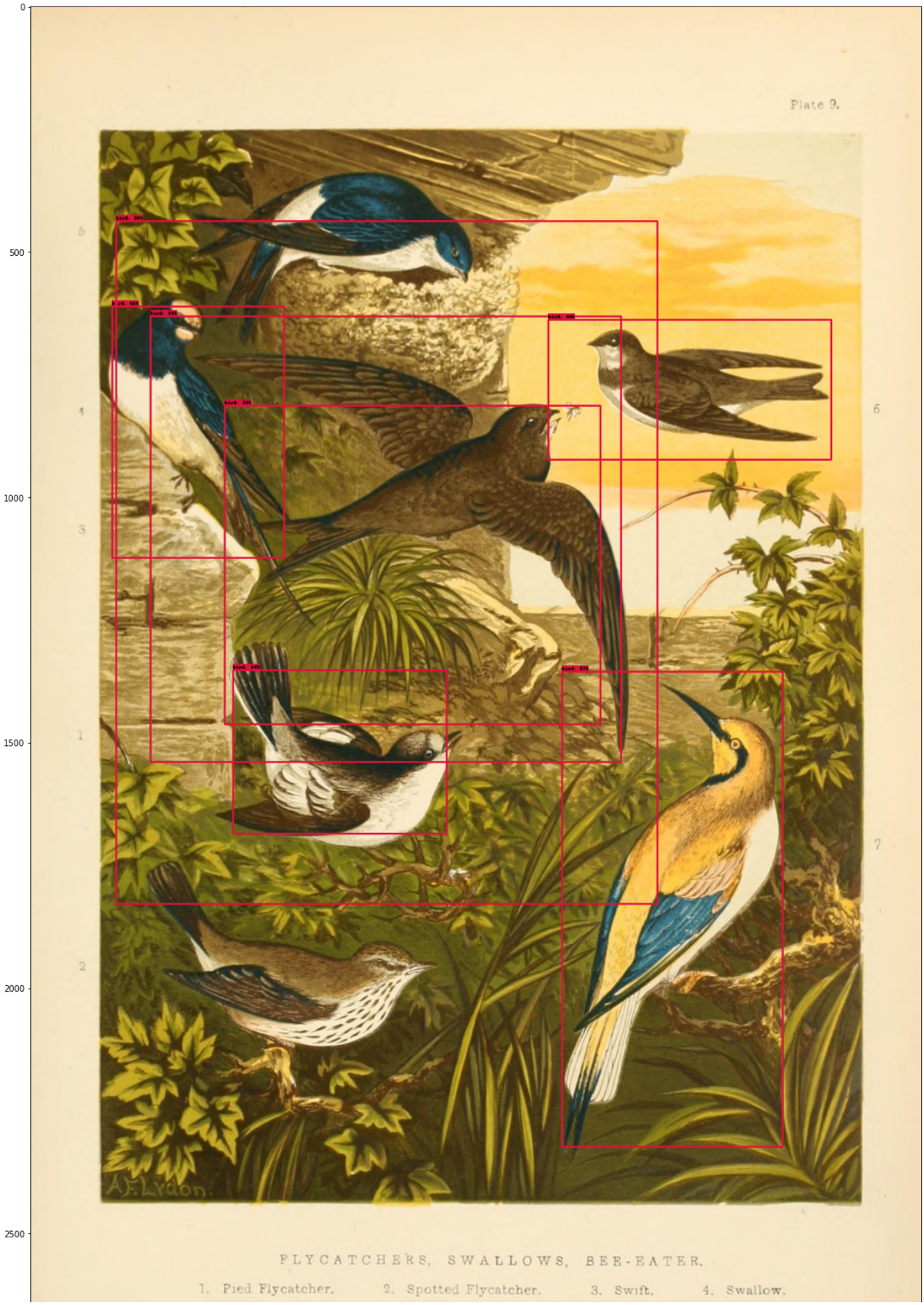
```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
print(result.keys())
```

```
dict_keys(['detection_multiclass_scores', 'raw_detection_scores', 'detection_scores'
```

Visualizando os resultados

```
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()
```



Conseguiu detectar alguns pássaros, mas ficou um sem classificar mesmo que estava de fácil classificação.

Faster R-CNN ResNet50 V1 640x640

Model Selection

model_display_name: Faster R-CNN ResNet50 V1 640x640

Mostrar código

```
Selected model:Faster R-CNN ResNet50 V1 640x640
Model Handle at TensorFlow Hub: https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1
```

Carregando o modelo selecionado do TensorFlow Hub

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

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

Carregando uma imagem

Praia

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

selected_image: Beach

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['raw_detection_boxes', 'detection_boxes', 'detection_multiclass_scores',
```

Visualizando os resultados

```
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()
```

Conseguiu detectar muito bem os objetos !!!

Carregando uma imagem

Cachorro

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

selected_image: Dogs

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

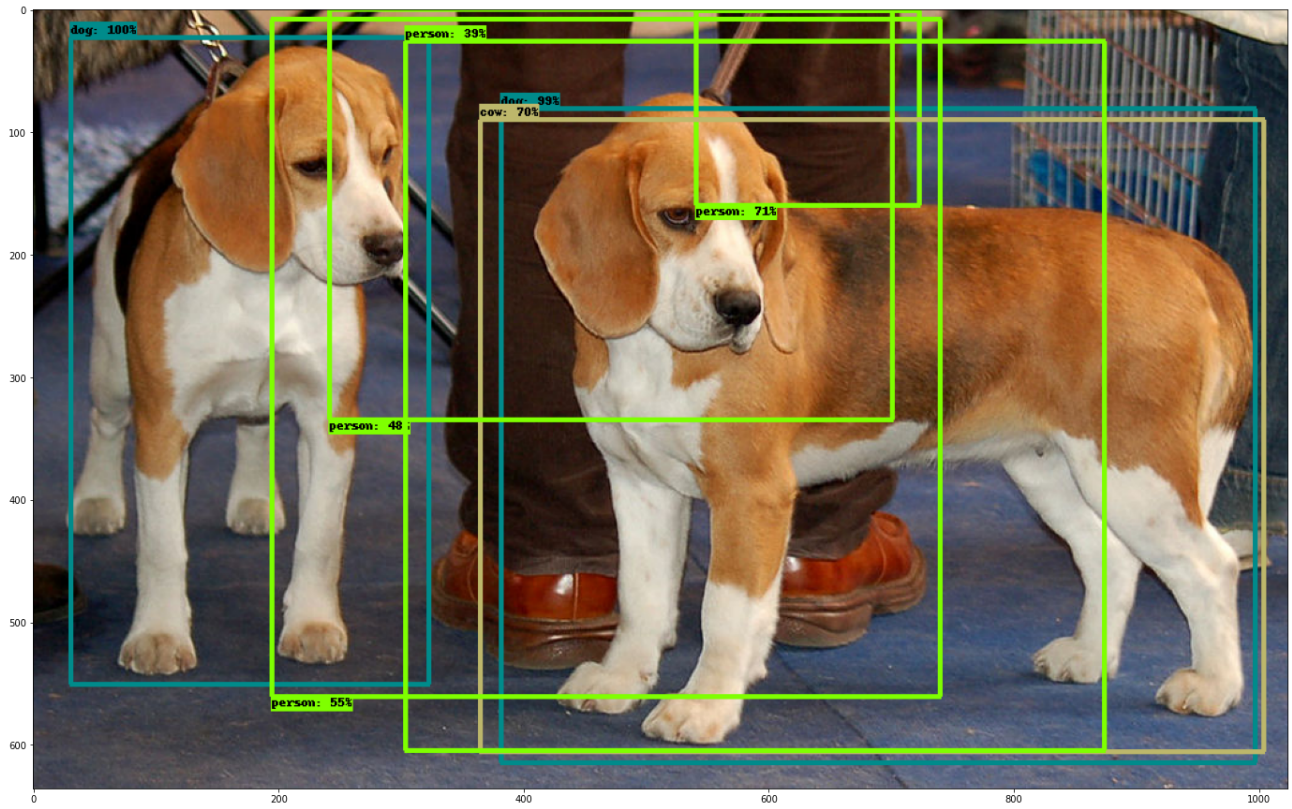
```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
dict_keys(['raw_detection_boxes', 'detection_boxes', 'detection_multiclass_scores',
```

Visualizando os resultados

```
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()
```



Detectou bem os cachorros.

Carregando uma imagem

Passáros

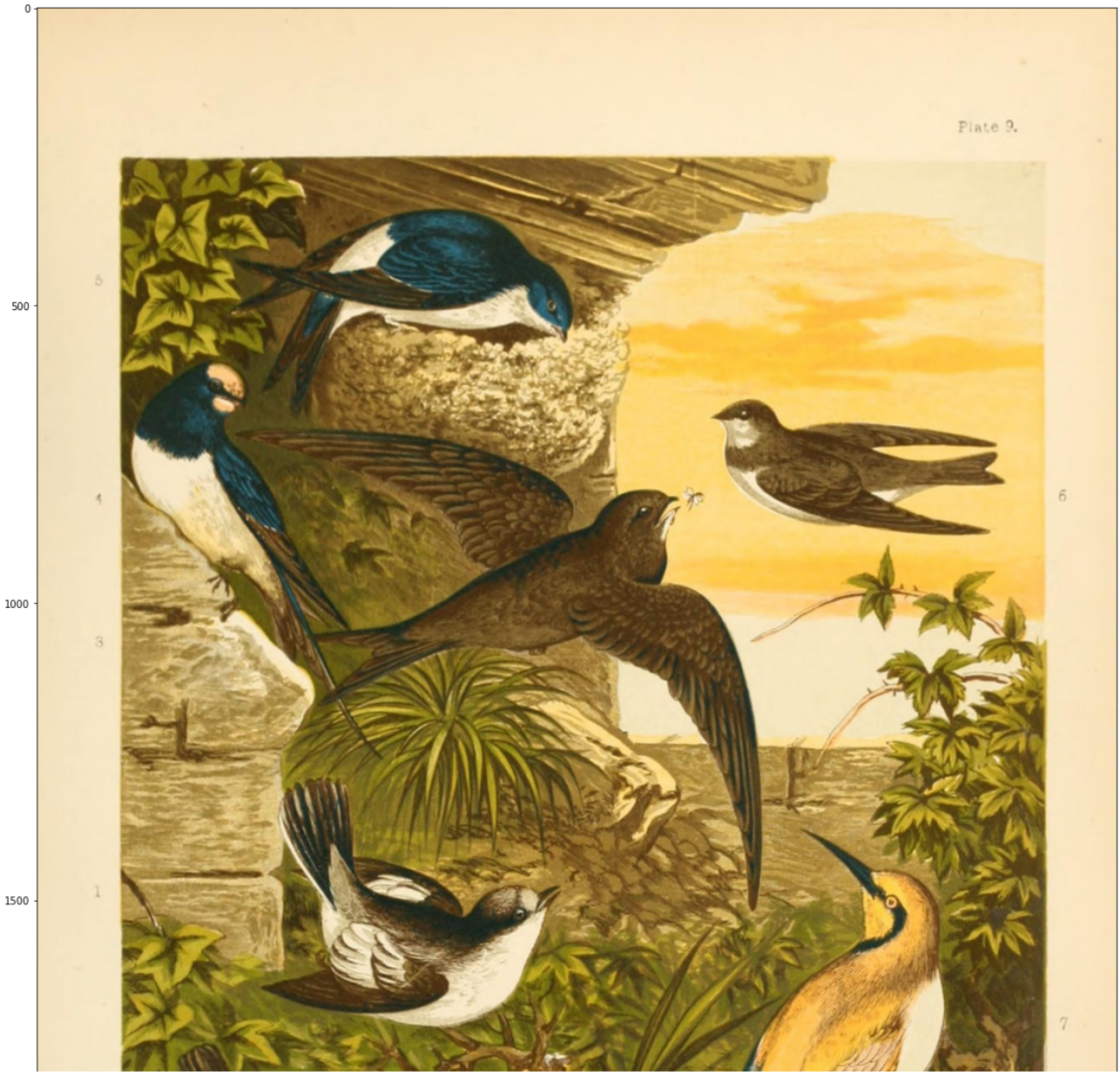
Image Selection (don't forget to execute the cell!)

selected_image: Birds

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código



Fazendo a inferência

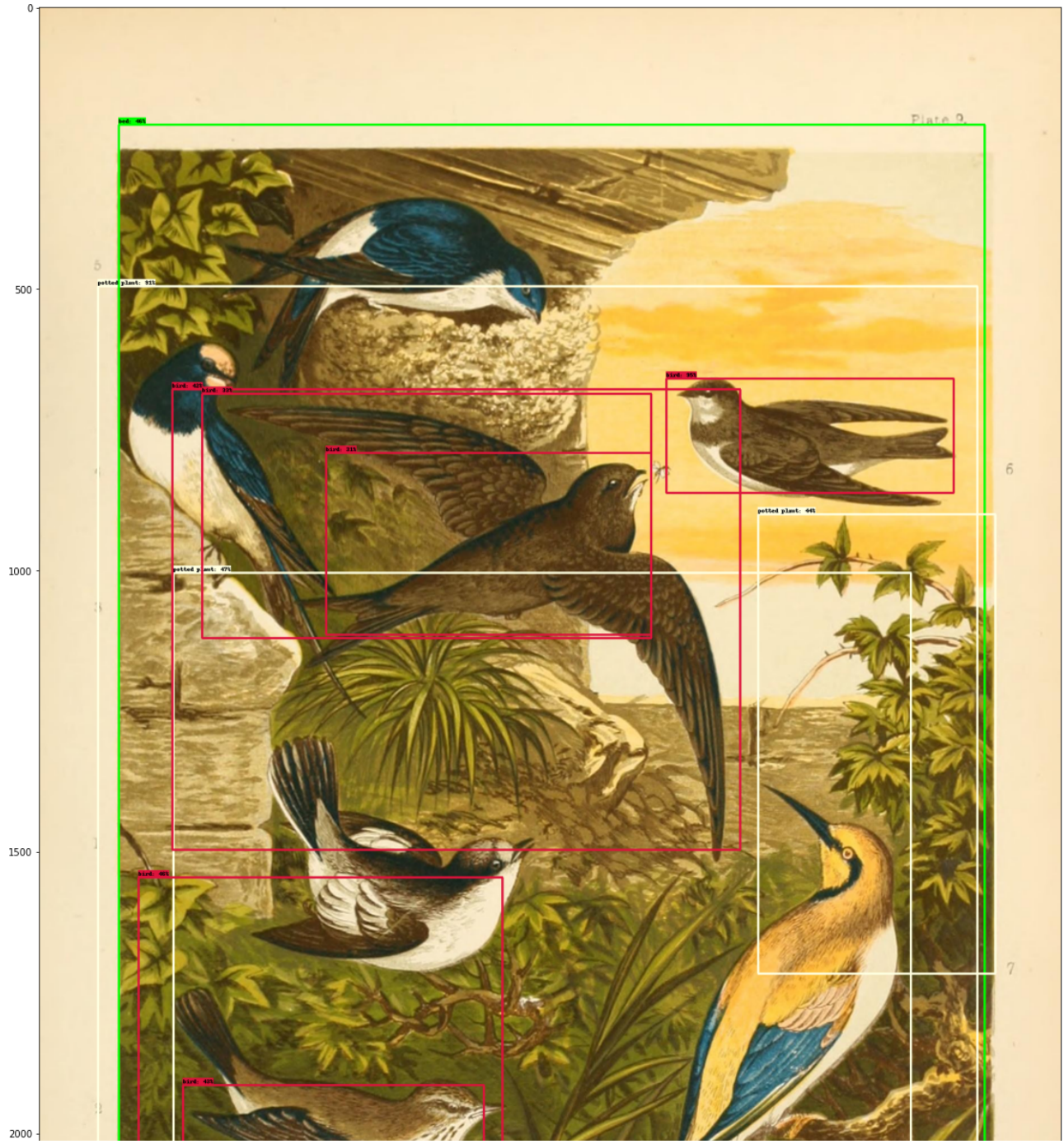
```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
dict_keys(['raw_detection_boxes', 'detection_boxes', 'detection_multiclass_scores',
```

Visualizando os resultados

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

# Use keypoints if available in detections
```

```
    # 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()
```



Conseguiu detectar bem os pássaros, mas faltou algumas caixas para alguns.

Mask R-CNN Inception ResNet V2 1024x1024

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_res
```

Carregando o modelo selecionado do TensorFlow Hub

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

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

Carregando uma imagem

Praia

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

selected_image: Beach

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

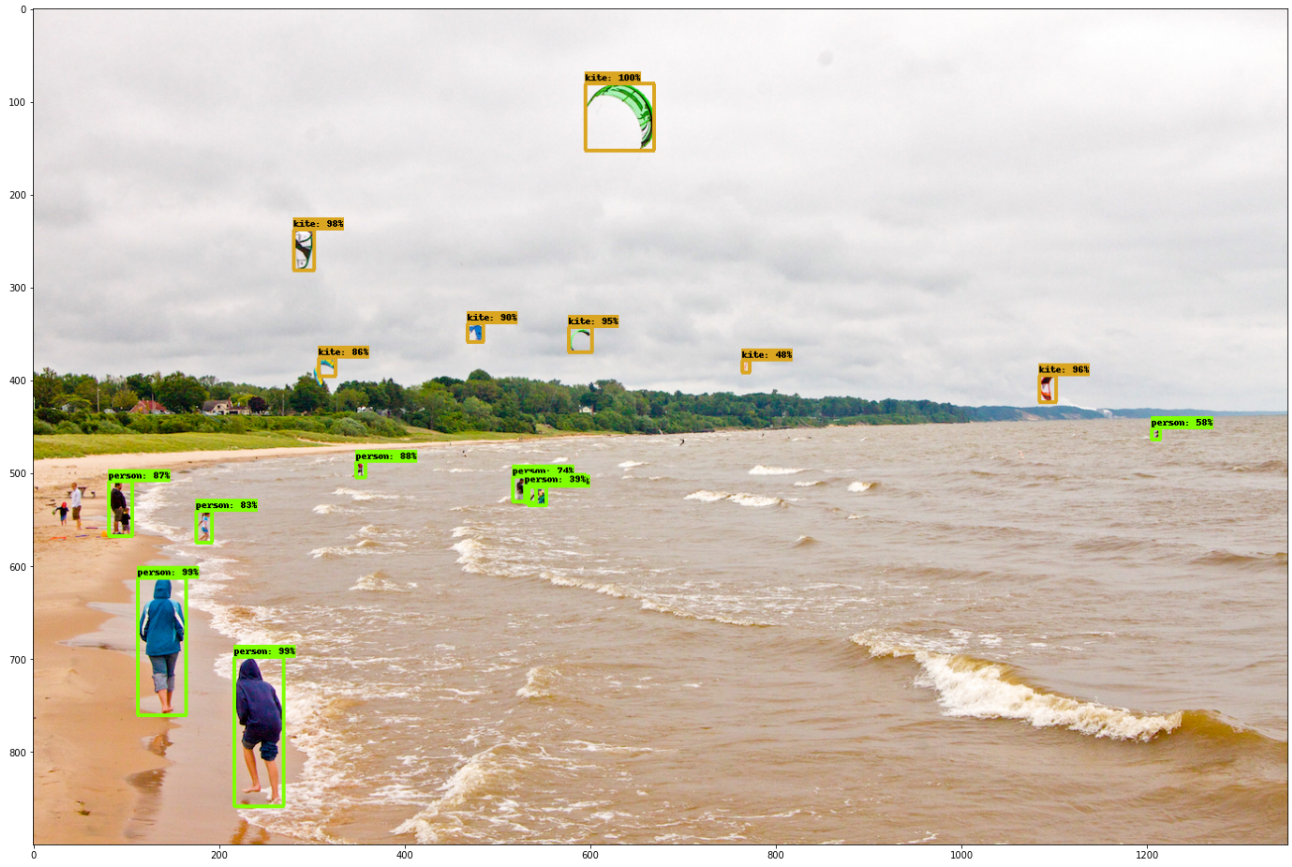
Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['box_classifier_features', 'num_proposals', 'raw_detection_scores', 'imag
```

Visualizando os resultados

```
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()
```



Detectou muito bem os objetos.

Carregando uma imagem

Cachorro

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

selected_image: Dogs

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

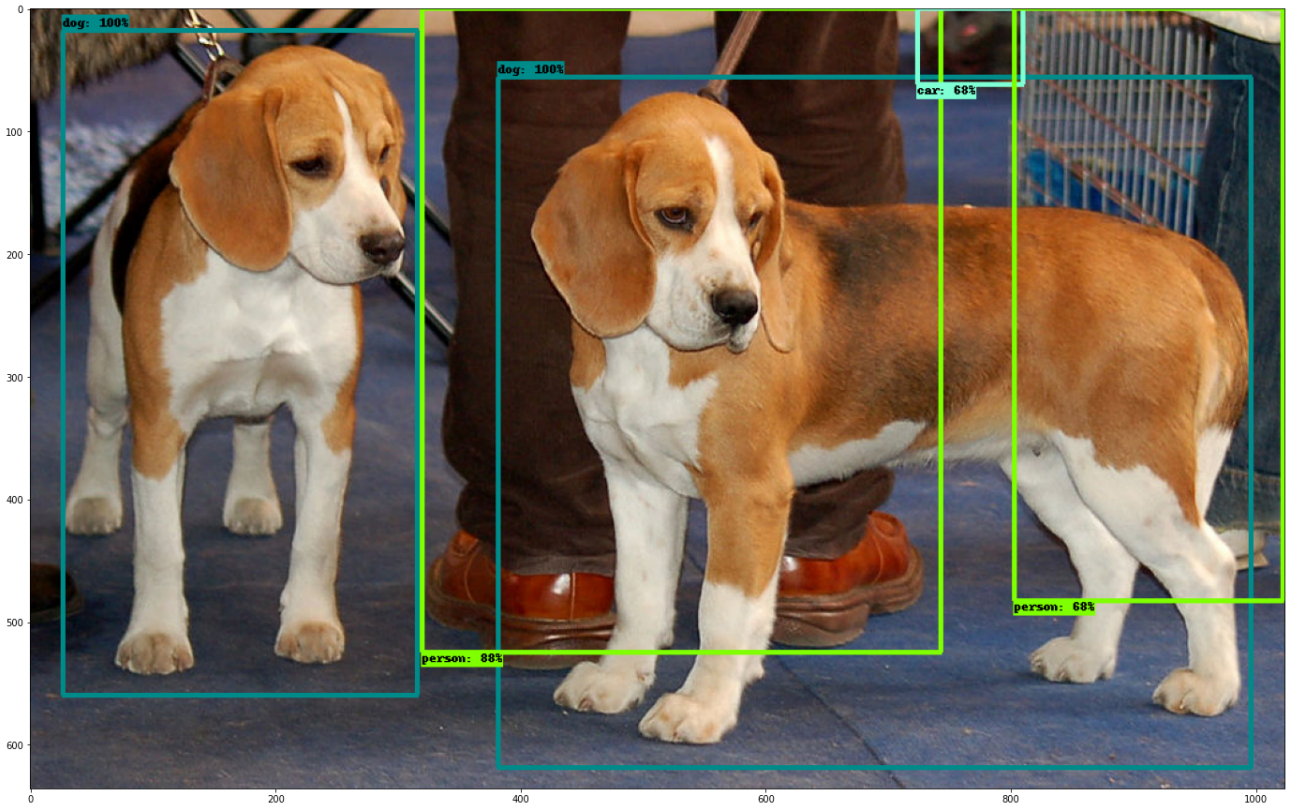
```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
dict_keys(['box_classifier_features', 'num_proposals', 'raw_detection_scores', 'imag
```

Visualizando os resultados

```
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()
```



Detectou bem os objetos

Carregando uma imagem

Passáros

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

selected_image: Birds

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

Mostrar código

Fazendo a inferência

```
# running inference
inicio = timeit.default_timer()
results = hub_model(image_np)
dicionario_de_tempo[model_display_name][selected_image] = timeit.default_timer() - inicio
# 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()}
print(result.keys())
```

```
dict_keys(['box_classifier_features', 'num_proposals', 'raw_detection_scores', 'imag
```

Visualizando os resultados

```
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()
```

Conseguiu detectar muito bem os pássaros. Somente um ficou sem detectar

Conclusão

Os modelos Mask R-CNN Inception ResNet V2 1024x1024 e Faster R-CNN ResNet50 V1 640x640 foram os melhores modelos dentre os testados, pois conseguiram classificar e detectar muito bem os objetos nas imagens de teste.

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-221-9b54273a1b94> in <module>
      3
      4 inicio = timeit.default_timer()
----> 5 alguma_funcao()
      6 fim = timeit.default_timer()
      7 print ('duracao: %f' % (fim - inicio))

NameError: name 'alguma_funcao' is not defined
```

SEARCH STACK OVERFLOW

```
for modelo_tempo in dicionario_de_tempo.keys():
   print(modelo_tempo)
   for imagem_tempo, tempo_gasto in dicionario_de_tempo[modelo_tempo].items():
     print(imagem_tempo, ":", tempo_gasto)

   print()
```

```
EfficientDet D0 512x512
Beach : 14.251046642999427
Dogs : 1.5044406110009731
Birds : 1.268514769999456

SSD MobileNet V2 FPNLite 320x320
Beach : 5.689724038998975
Dogs : 0.3652632419998554
Birds : 0.478123893000884

SSD ResNet50 V1 FPN 640x640 (RetinaNet50)
Beach : 13.449030376999872
Dogs : 3.8460350679997646
Birds : 3.905161828999553

Faster R-CNN ResNet50 V1 640x640
Beach : 16.668734711998695
Dogs : 7.8982937289983965
Birds : 4.5323845830007485

Mask R-CNN Inception ResNet V2 1024x1024
Beach : 95.075538744999
Dogs : 84.44044439199934
Birds : 94.32926981799937
```

Percebe-se que o modelo EfficientDet D0 512x512 é o mais rápido e modelo Mask R-CNN Inception ResNet V2 1024x1024 é o mais lento.

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.

Importações

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive
```

```
import os
import scipy.io
import scipy.misc
import numpy as np
import pandas as pd
import PIL
import struct
import cv2
from numpy import expand_dims
import tensorflow as tf
from skimage.transform import resize
from keras import backend as K
from keras.layers import Input, Lambda, Conv2D, BatchNormalization, LeakyReLU, ZeroPadding
from keras.models import load_model, Model
from tensorflow.keras.layers import add, concatenate
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from matplotlib import pyplot
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from matplotlib.patches import Rectangle
%matplotlib inline
```

Etapa 1: A classe WeightReader é usada para analisar o arquivo "yolov3.weights" e carregar os pesos do modelo na memória em um formato que podemos definir no modelo keras

```
class WeightReader:
    def __init__(self, weight_file):
        with open(weight_file, 'rb') as w_f:
            major,  = struct.unpack('i', w_f.read(4))
            minor,  = struct.unpack('i', w_f.read(4))
            revision, = struct.unpack('i', w_f.read(4))
            if (major*10 + minor) >= 2 and major < 1000 and minor < 1000:
                w_f.read(8)
            else:
                w_f.read(4)
```

```
            transpose = (major > 1000) or (minor > 1000)
            binary = w_f.read()
        self.offset = 0
        self.all_weights = np.frombuffer(binary, dtype='float32')

    def read_bytes(self, size):
        self.offset = self.offset + size
        return self.all_weights[self.offset-size:self.offset]

    def load_weights(self, model):
        for i in range(106):
            try:
                conv_layer = model.get_layer('conv_' + str(i))
                print("loading weights of convolution #" + str(i))
                if i not in [81, 93, 105]:
                    norm_layer = model.get_layer('bnorm_' + str(i))
                    size = np.prod(norm_layer.get_weights()[0].shape)
                    beta  = self.read_bytes(size) # bias
                    gamma = self.read_bytes(size) # scale
                    mean  = self.read_bytes(size) # mean
                    var   = self.read_bytes(size) # variance
                    weights = norm_layer.set_weights([gamma, beta, mean, var])
                if len(conv_layer.get_weights()) > 1:
                    bias   = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape))
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shap
                    kernel = kernel.transpose([2,3,1,0])
                    conv_layer.set_weights([kernel, bias])
                else:
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shap
                    kernel = kernel.transpose([2,3,1,0])
                    conv_layer.set_weights([kernel])
            except ValueError:
                print("no convolution #" + str(i))

    def reset(self):
        self.offset = 0
```

Etapa 2:

- _conv_block(input, convs, skip=True) é uma função para criar camada convolucional
- make_yolov3_model() é uma função para criar camadas de convolução e empilhar juntas como um modelo yolo completo

```
def _conv_block(inp, convs, skip=True):
    x = inp
    count = 0
    for conv in convs:
        if count == (len(convs) - 2) and skip:
            skip_connection = x
        count += 1
```

```
            if conv['stride'] > 1: x = ZeroPadding2D(((1,0),(1,0)))(x) # peculiar padding as d
            x = Conv2D(conv['filter'],
                       conv['kernel'],
                       strides=conv['stride'],
                       padding='valid' if conv['stride'] > 1 else 'same', # peculiar padding a
                       name='conv_' + str(conv['layer_idx']),
                       use_bias=False if conv['bnorm'] else True)(x)
            if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' + str(conv['
            if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' + str(conv['layer_idx']))
        return add([skip_connection, x]) if skip else x

    def make_yolov3_model():
        input_image = Input(shape=(None, None, 3))
        # Layer  0 => 4
        x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True,
                                       {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True,
                                       {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True,
                                       {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True,
        # Layer  5 => 8
        x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky':
                            {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                            {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
        # Layer  9 => 11
        x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                            {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
        # Layer 12 => 15
        x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky':
                            {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
        # Layer 16 => 36
        for i in range(7):
            x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leak
                                {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leak
        skip_36 = x
        # Layer 37 => 40
        x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky':
                            {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                            {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
        # Layer 41 => 61
        for i in range(7):
            x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leak
                                {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leak
        skip_61 = x
        # Layer 62 => 65
        x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky':
                            {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
        # Layer 66 => 74
        for i in range(3):
            x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'lea
                                {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'lea
        # Layer 75 => 79
        x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
                            {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
```

```
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
    # Layer 80 => 82
    yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,  '
                              {'filter':  255, 'kernel': 1, 'stride': 1, 'bnorm': False, '
    # Layer 83 => 86
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_61])
    # Layer 87 => 91
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
    # Layer 92 => 94
    yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,  'l
                              {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'l
    # Layer 95 => 98
    x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_36])
    # Layer 99 => 106
    yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  '
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  '
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  '
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  '
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  '
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  '
                               {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, '
    model = Model(input_image, [yolo_82, yolo_94, yolo_106])
    return model
```

Etapa 3:

- Definir o modelo
- Carregar o peso
- Salvar o modelo

```
# define the yolo v3 model
yolov3 = make_yolov3_model()

# load the weights
weight_reader = WeightReader('/content/gdrive/My Drive/Colab Dataset/yolov3.weights')

# set the weights
weight_reader.load_weights(yolov3)

# save the model to file
yolov3.save('model.h5')
```

```
     loading weights of convolution #47
     loading weights of convolution #48
     no convolution #49
     loading weights of convolution #50
```

```
loading weights of convolution #51
no convolution #52
loading weights of convolution #53
loading weights of convolution #54
no convolution #55
loading weights of convolution #56
loading weights of convolution #57
no convolution #58
loading weights of convolution #59
loading weights of convolution #60
no convolution #61
loading weights of convolution #62
loading weights of convolution #63
loading weights of convolution #64
no convolution #65
loading weights of convolution #66
loading weights of convolution #67
no convolution #68
loading weights of convolution #69
loading weights of convolution #70
no convolution #71
loading weights of convolution #72
loading weights of convolution #73
no convolution #74
loading weights of convolution #75
loading weights of convolution #76
loading weights of convolution #77
loading weights of convolution #78
loading weights of convolution #79
loading weights of convolution #80
loading weights of convolution #81
no convolution #82
no convolution #83
loading weights of convolution #84
no convolution #85
no convolution #86
loading weights of convolution #87
loading weights of convolution #88
loading weights of convolution #89
loading weights of convolution #90
loading weights of convolution #91
loading weights of convolution #92
loading weights of convolution #93
no convolution #94
no convolution #95
loading weights of convolution #96
no convolution #97
no convolution #98
loading weights of convolution #99
loading weights of convolution #100
loading weights of convolution #101
loading weights of convolution #102
loading weights of convolution #103
loading weights of convolution #104
loading weights of convolution #105
```

Etapa 4: Decodificar a saída da previsão para as coordenadas do retângulo

- A classe BoundBox é usada para retornar as coordenadas da caixa delimitadora do objeto, o nome do objeto e a pontuação do limite
- A função decode_netout é usada para decodificar a saída de previsão para as coordenadas do retângulo

```
class BoundBox:
  def __init__(self, xmin, ymin, xmax, ymax, objness = None, classes = None):
    self.xmin = xmin
    self.ymin = ymin
    self.xmax = xmax
    self.ymax = ymax
    self.objness = objness
    self.classes = classes
    self.label = -1
    self.score = -1

  def get_label(self):
    if self.label == -1:
      self.label = np.argmax(self.classes)

    return self.label

  def get_score(self):
    if self.score == -1:
      self.score = self.classes[self.get_label()]
    return self.get_score

def _sigmoid(x):
  return 1. /(1. + np.exp(-x))

def decode_netout(netout, anchors, obj_thresh, net_h, net_w):
    grid_h, grid_w = netout.shape[:2]
    nb_box = 3
    netout = netout.reshape((grid_h, grid_w, nb_box, -1))
    nb_class = netout.shape[-1] - 5
    boxes = []
    netout[..., :2]  = _sigmoid(netout[..., :2])
    netout[..., 4:]  = _sigmoid(netout[..., 4:])
    netout[..., 5:]  = netout[..., 4][..., np.newaxis] * netout[..., 5:]
    netout[..., 5:] *= netout[..., 5:] > obj_thresh

    for i in range(grid_h*grid_w):
        row = i / grid_w
        col = i % grid_w
        for b in range(nb_box):
            # 4th element is objectness score
            objectness = netout[int(row)][int(col)][b][4]
            if(objectness.all() <= obj_thresh): continue
            # first 4 elements are x, y, w, and h
            x, y, w, h = netout[int(row)][int(col)][b][:4]
            x = (col + x) / grid_w # center position, unit: image width
            y = (row + y) / grid_h # center position, unit: image height
            w = anchors[2 * b + 0] * np.exp(w) / net_w # unit: image width
            h = anchors[2 * b + 1] * np.exp(h) / net_h # unit: image height
```

```
            # last elements are class probabilities
            classes = netout[int(row)][col][b][5:]
            box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
            boxes.append(box)
    return boxes
```

Passo 5: Esticar a caixa para se ajustar à forma normal da imagem

```
def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
    new_w, new_h = net_w, net_h
    for i in range(len(boxes)):
        x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w
        y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h
        boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
        boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
        boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
        boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)
```

Etapa 6: Implementação da IOU

```
def _interval_overlap(interval_a, interval_b):
    x1, x2 = interval_a
    x3, x4 = interval_b
    if x3 < x1:
        if x4 < x1:
            return 0
        else:
            return min(x2,x4) - x1
    else:
        if x2 < x3:
             return 0
        else:
            return min(x2,x4) - x3

def bbox_iou(box1, box2):
    intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
    intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])
    intersect = intersect_w * intersect_h
    w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin
    w2, h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin
    union = w1*h1 + w2*h2 - intersect
    return float(intersect) / union

def do_nms(boxes, nms_thresh):
    if len(boxes) > 0:
        nb_class = len(boxes[0].classes)
    else:
        return
    for c in range(nb_class):
        sorted_indices = np.argsort([-box.classes[c] for box in boxes])
        for i in range(len(sorted_indices)):
            index_i = sorted_indices[i]
```

```
            if boxes[index_i].classes[c] == 0: continue
            for j in range(i+1, len(sorted_indices)):
                index_j = sorted_indices[j]
                if bbox_iou(boxes[index_i], boxes[index_j]) >= nms_thresh:
                    boxes[index_j].classes[c] = 0

# get all of the results above a threshold
def get_boxes(boxes, labels, thresh):
    v_boxes, v_labels, v_scores = list(), list(), list()
    # enumerate all boxes
    for box in boxes:
        # enumerate all possible labels
        for i in range(len(labels)):
            # check if the threshold for this label is high enough
            if box.classes[i] > thresh:
                v_boxes.append(box)
                v_labels.append(labels[i])
                v_scores.append(box.classes[i]*100)
                # don't break, many labels may trigger for one box
    return v_boxes, v_labels, v_scores

# draw all results
def draw_boxes(filename, v_boxes, v_labels, v_scores):

    # load the image
    data = pyplot.imread(filename)
    # plot the image
    pyplot.imshow(data)
    # get the context for drawing boxes
    ax = pyplot.gca()
    # plot each box
    for i in range(len(v_boxes)):
        box = v_boxes[i]
        # get coordinates
        y1, x1, y2, x2 = box.ymin, box.xmin, box.ymax, box.xmax
        # calculate width and height of the box
        width, height = x2 - x1, y2 - y1
        # create the shape
        rect = Rectangle((x1, y1), width, height, fill=False, color='red', linewidth = '2'
        # draw the box
        ax.add_patch(rect)
        # draw text and score in top left corner
        label = "%s (%.3f)" % (v_labels[i], v_scores[i])
        pyplot.text(x1, y1, label, color='red')
    # show the plot
    pyplot.show()
```

Etapa 7: Declarar Configurações e Rótulos

```
# define the anchors
anchors = [[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]]

# define the probability threshold for detected objects
```

```
class_threshold = 0.6

# define the labels
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck",
  "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench",
  "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
  "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard",
  "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard",
  "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banar
  "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake'
  "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", '
  "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerato
  "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]
```

Etapa 8: Carregando a imagem para modelar e fazer a previsão

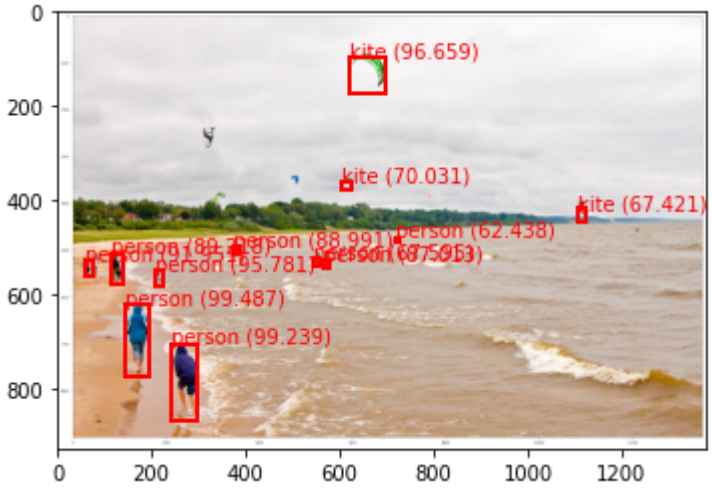
```
def load_image_pixels(filename, shape):
  # load image to get its shape
  image = load_img(filename)
  width, height = image.size

  # load image with required size
  image = load_img(filename, target_size=shape)
  image = img_to_array(image)

  # grayscale image normalization
  image = image.astype('float32')
  image /= 255.0

  # add a dimension so that we have one sample
  image = expand_dims(image, 0)
  return image, width, height
```

Etapa 9: Fazer Predições

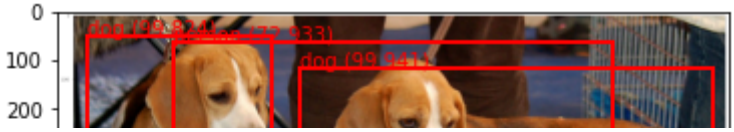
```
#Dicionario de tempos
dicionario_de_tempo["YoloV3"] = {}


paths = ["beach.png", "dog.png", "bird.png"]

for fn in paths:
  photo_filename = '/content/gdrive/MyDrive/' + fn

  # define the expected input shape for the model
  input_w, input_h = 416, 416

  image, image_w, image_h = load_image_pixels(photo_filename, (input_w, input_h))

  inicio = timeit.default_timer()

  # make prediction
```

```
yhat = yolov3.predict(image)

dicionario_de_tempo["YoloV3"][fn] = timeit.default_timer() - inicio

# summarize the shape of the list of arrays
print([a.shape for a in yhat])

boxes = list()
for i in range(len(yhat)):
  # decode the output of the network
  boxes += decode_netout(yhat[i][0], anchors[i], class_threshold, input_h, input_w)

# correct the sizes of the bounding boxes for the shape of the image
correct_yolo_boxes(boxes, image_h, image_w, input_h, input_w)

# suppress non-maximal boxes
do_nms(boxes, 0.5)

# get the details of the detected objects
v_boxes, v_labels, v_scores = get_boxes(boxes, labels, class_threshold)

# summarize what we found
for i in range(len(v_boxes)):
  print(v_labels[i], v_scores[i])

# draw what we found
draw_boxes(photo_filename, v_boxes, v_labels, v_scores)
```

```
1/1 [==============================] - 6s 6s/step
[(1, 13, 13, 255), (1, 26, 26, 255), (1, 52, 52, 255)]
person 99.48738813400269
person 99.23863410949707
kite 96.65899276733398
kite 70.03083825111389
kite 67.42089986801147
person 62.43835091590881
person 88.9911413192749
person 67.59470701217651
person 87.01285719871521
person 91.95093512535095
person 89.21782970428467
person 95.78072428703308
```

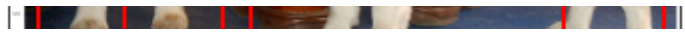



```
1/1 [==============================] - 1s 1s/step
[(1, 13, 13, 255), (1, 26, 26, 255), (1, 52, 52, 255)]
dog 99.82355833053589
person 72.93270826339722
dog 99.94120597839355
```



O modelo YoloV3 detectou e classificou bem os objetos, mas não os pássaros.

Custo Computacional



```
for modelo_tempo in dicionario_de_tempo.keys():
   print(modelo_tempo)
   for imagem_tempo, tempo_gasto in dicionario_de_tempo[modelo_tempo].items():
     print(imagem_tempo, ":", tempo_gasto)

   print()
```

```
  EfficientDet D0 512x512
  Beach : 14.251046642999427
  Dogs : 1.5044406110009731
  Birds : 1.268514769999456

  SSD MobileNet V2 FPNLite 320x320
  Beach : 5.689724038998975
```

```
Dogs : 0.3652632419998554
Birds : 0.478123893000884

SSD ResNet50 V1 FPN 640x640 (RetinaNet50)
Beach : 13.449030376999872
Dogs : 3.8460350679997646
Birds : 3.905161828999553

Faster R-CNN ResNet50 V1 640x640
Beach : 16.668734711998695
Dogs : 7.8982937289983965
Birds : 4.5323845830007485

Mask R-CNN Inception ResNet V2 1024x1024
Beach : 95.075538744999
Dogs : 84.44044439199934
Birds : 94.32926981799937

YoloV3
beach.png : 8.1613365409994
dog.png : 1.5318640129989944
bird.png : 2.608251036999718
```

O YoloV3 gasta praticamente o mesmo tempo para classificar qualquer imagem e não é um tempo muito elevado comparado aos demais modelos, quase 3 segundos.

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

Caso você queira usar as técnicas de detecção de objetos em uma base de dados própria, siga o tutorial do link. Você também pode se basear no trabalho do link.

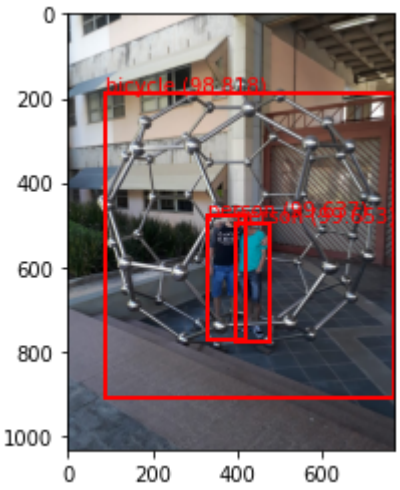
Relate sua experiência e anexe aqui os resultados.

Detectando eu e meu pai dentro de um grafo gigante na UFMG utilizando o modelo pré-treinado YoloV3.

```
photo_filename = '/content/gdrive/MyDrive/grafo.jpeg'

# define the expected input shape for the model
input_w, input_h = 416, 416

image, image_w, image_h = load_image_pixels(photo_filename, (input_w, input_h))

 # make prediction
yhat = yolov3.predict(image)

# summarize the shape of the list of arrays
print([a.shape for a in yhat])
```

```
boxes = list()
for i in range(len(yhat)):
  # decode the output of the network
  boxes += decode_netout(yhat[i][0], anchors[i], class_threshold, input_h, input_w)

 # correct the sizes of the bounding boxes for the shape of the image
correct_yolo_boxes(boxes, image_h, image_w, input_h, input_w)

# suppress non-maximal boxes
do_nms(boxes, 0.5)

# get the details of the detected objects
v_boxes, v_labels, v_scores = get_boxes(boxes, labels, class_threshold)

# summarize what we found
for i in range(len(v_boxes)):
  print(v_labels[i], v_scores[i])

# draw what we found
draw_boxes(photo_filename, v_boxes, v_labels, v_scores)
```

```
1/1 [==============================] - 1s 1s/step
[(1, 13, 13, 255), (1, 26, 26, 255), (1, 52, 52, 255)]
bicycle 98.81848692893982
person 99.63719248771667
person 99.6533751487732
```



Conseguiu detectar as pessoas corretamente, porém confundiu o grafo com uma bicicleta.

Part II - Segmentação (40pt)

ToDo : Rodando um tutorial (15pt)

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

Importações e Configurações

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

```
  Building wheel for tensorflow-examples (setup.py) ... done
  WARNING: Built wheel for tensorflow-examples is invalid: Metadata 1.2 mandates PEP
    Running setup.py install for tensorflow-examples ... done
  DEPRECATION: tensorflow-examples was installed using the legacy 'setup.py install'
```

```
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
```

Baixar o conjunto de dados Oxford-IIIT Pets

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

```
Downloading and preparing dataset 773.52 MiB (download: 773.52 MiB, generated: 774.6
Dataset oxford_iiit_pet downloaded and prepared to ~/tensorflow_datasets/oxford_iiit
```

Ajustar Dataset

```
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))
```

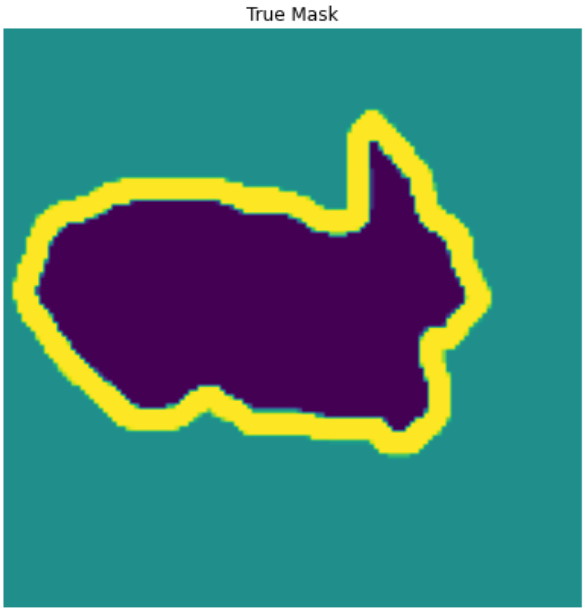
```
  if tf.random.uniform(()) > 0.5:
    input_image = tf.image.flip_left_right(input_image)
    input_mask = tf.image.flip_left_right(input_mask)

  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
```

Divisão do Conjunto - Teste e Treinamento

```
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.AUT
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)
```

Visualizar Instâncias do Dataset

```
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])
```



Definir o modelo Codificador

```
OUTPUT_CHANNELS = 3


base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], include_top=Fals

# 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/m
9412608/9406464 [==============================] - 0s 0us/step
9420800/9406464 [==============================] - 0s 0us/step
```

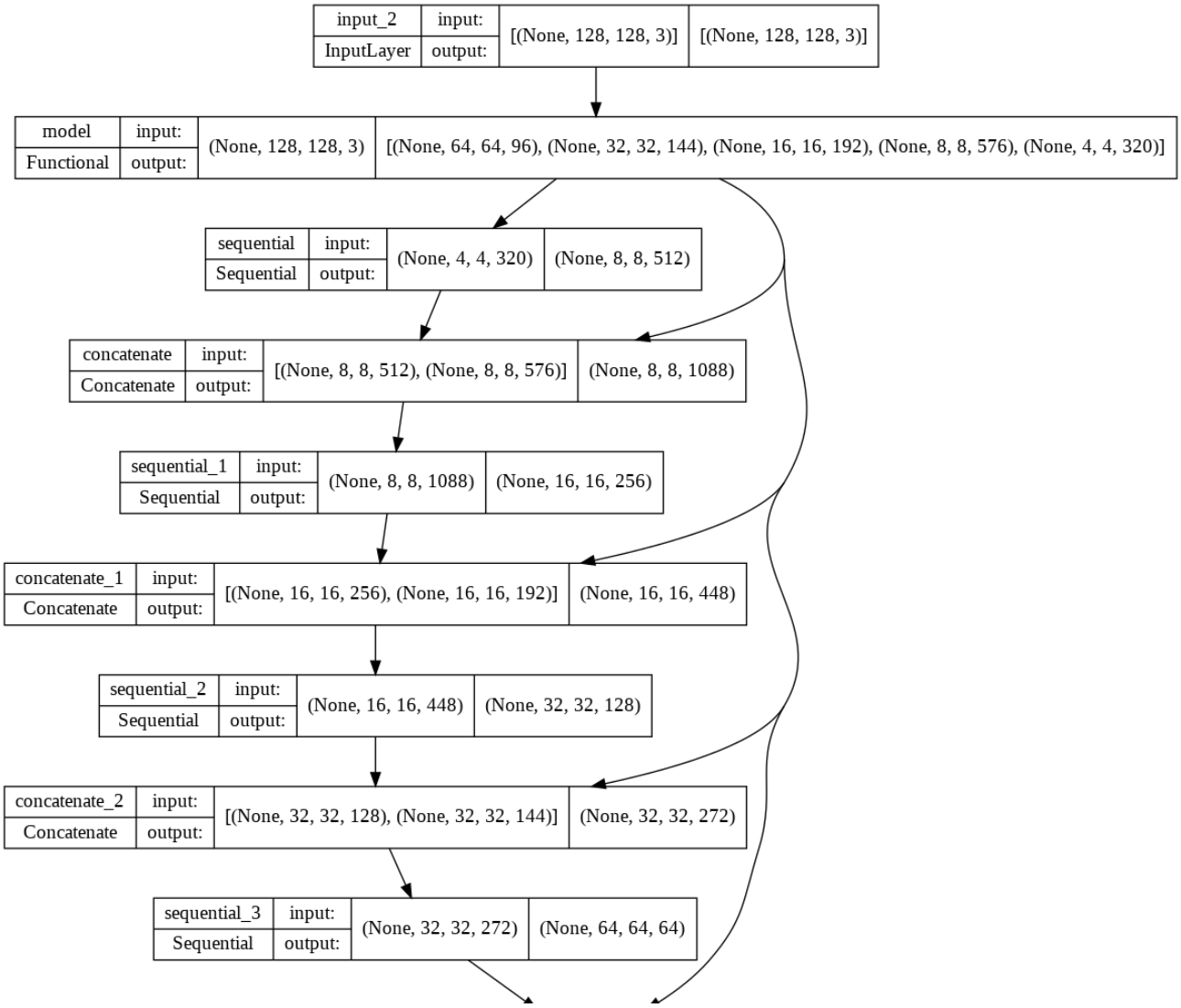
Definir o modelo Decodificador

```
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)
```

Treinar Modelo

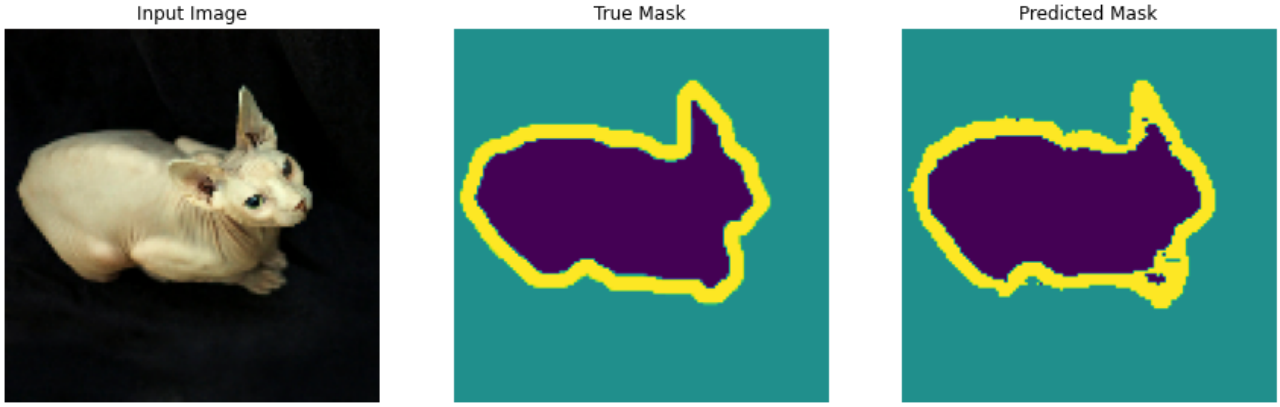
```
model = unet_model(OUTPUT_CHANNELS)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


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()
```

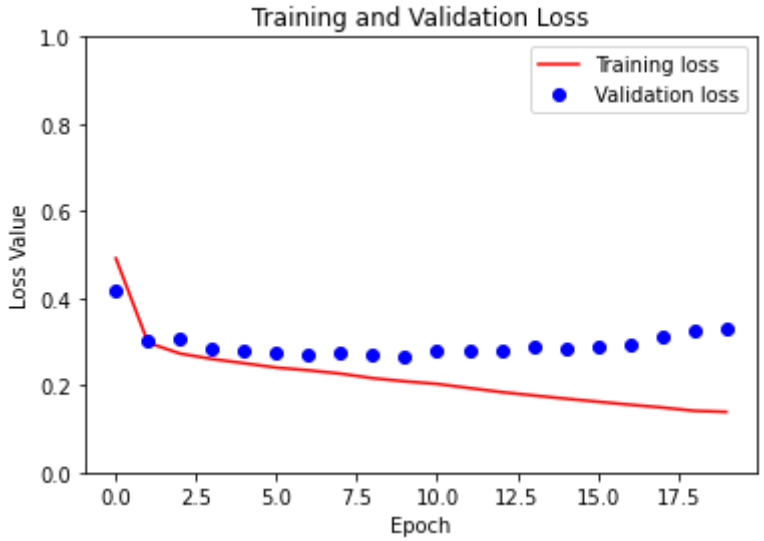

```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    clear_output(wait=True)
    show_predictions()
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))


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()])
```




```
Sample Prediction after epoch 20

57/57 [==============================] - 9s 160ms/step - loss: 0.1391 - accuracy: 0.
```

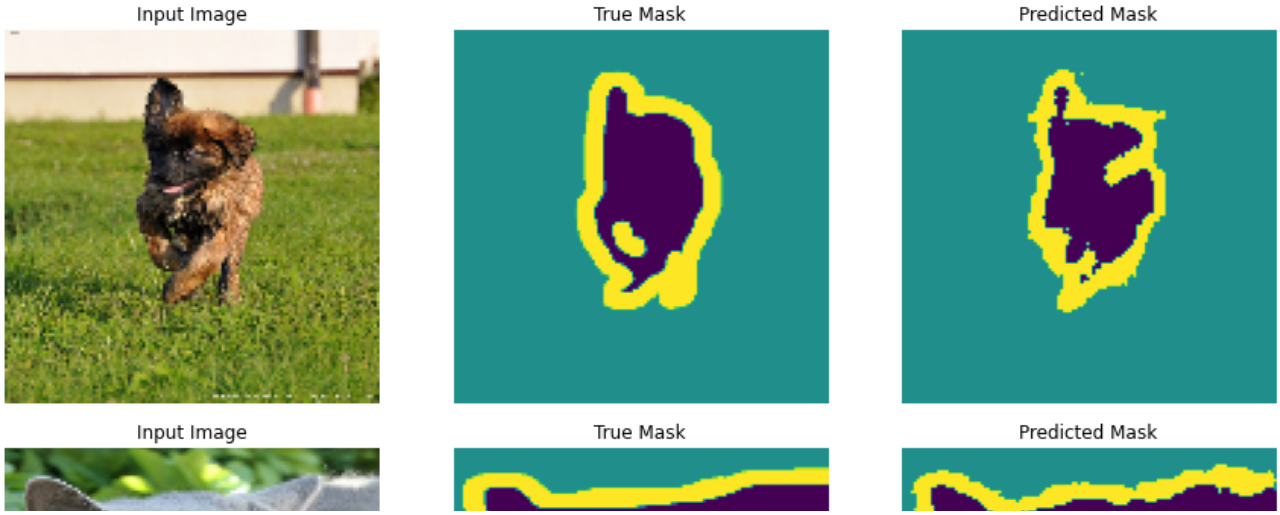
```
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']

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()
```



Fazer Predições

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



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

Importações

```
# Importações
import numpy as np
import os
import cv2
from keras.models import Sequential
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report
from keras.layers import Conv2D, MaxPool2D, Dropout, Conv2DTranspose,  GlobalAveragePoolir
from tensorflow.keras.applications import MobileNet, InceptionV3, Xception, VGG16, Efficie
```

MobileNet

Definir o modelo Codificador

```
OUTPUT_CHANNELS = 3

base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], include_top=Fals

# 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
```

Definir o modelo Decodificador

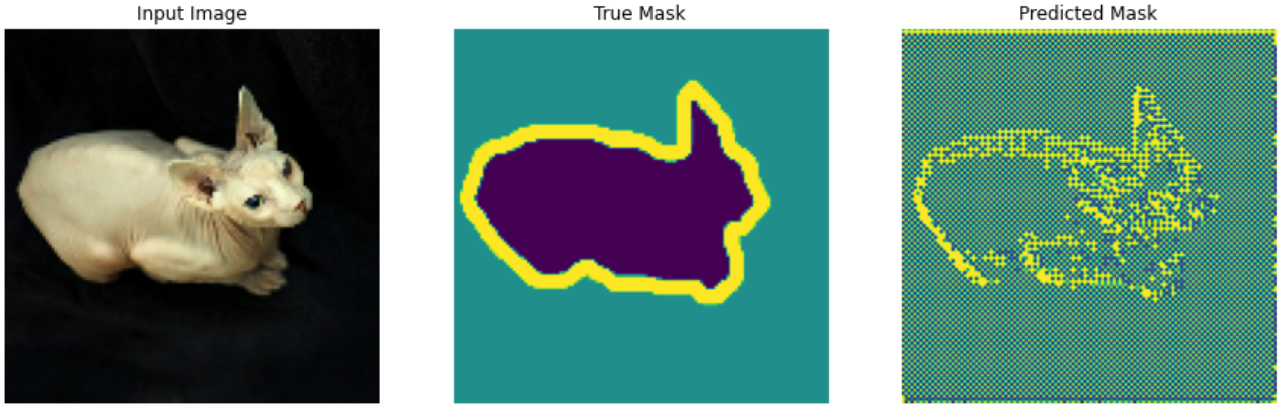
```
up_stack = [
    pix2pix.upsample(512, 3),  # 4x4 -> 8x8
    pix2pix.upsample(256, 3),  # 8x8 -> 16x16
    pix2pix.upsample(128, 3),  # 16x16 -> 32x32
    pix2pix.upsample(64, 3),   # 32x32 -> 64x64
]


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
  x = Dropout(0.05)(x) # Adicionei Dropout

  # 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)
```

Treinar Modelo

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


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

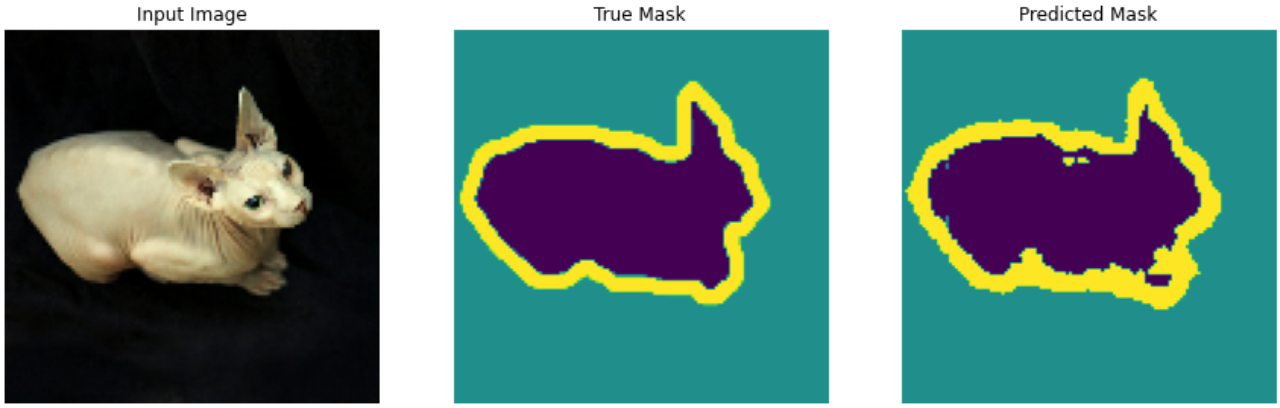
```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]
```

```
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()
```



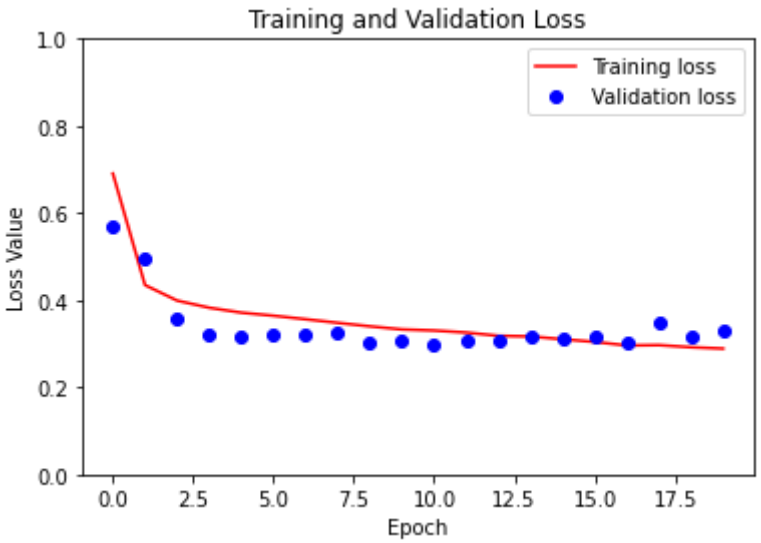
[(None, 64, 64, 64), (None, 64, 64, 96)] | (None, 64, 64, 160) |

```
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()])
```




```
Sample Prediction after epoch 20

57/57 [==============================] - 9s 163ms/step - loss: 0.2889 - accuracy: 0.
```

```
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()
```



Fazer Predições

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

VGG

Definir o modelo Codificador

```
OUTPUT_CHANNELS = 3

for i in base_model.layers:
  print(i.name)
```

```
      input_60
      block1_conv1
      block1_conv2
      block1_pool
      block2_conv1
      block2_conv2
      block2_pool
      block3_conv1
      block3_conv2
      block3_conv3
      block3_pool
      block4_conv1
      block4_conv2
      block4_conv3
      block4_pool
      block5_conv1
      block5_conv2
      block5_conv3
      block5_pool
```

```
base_model = tf.keras.applications.VGG16(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(layer.name).output for layer in base_model.layers if "pool

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

down_stack.trainable = False
```

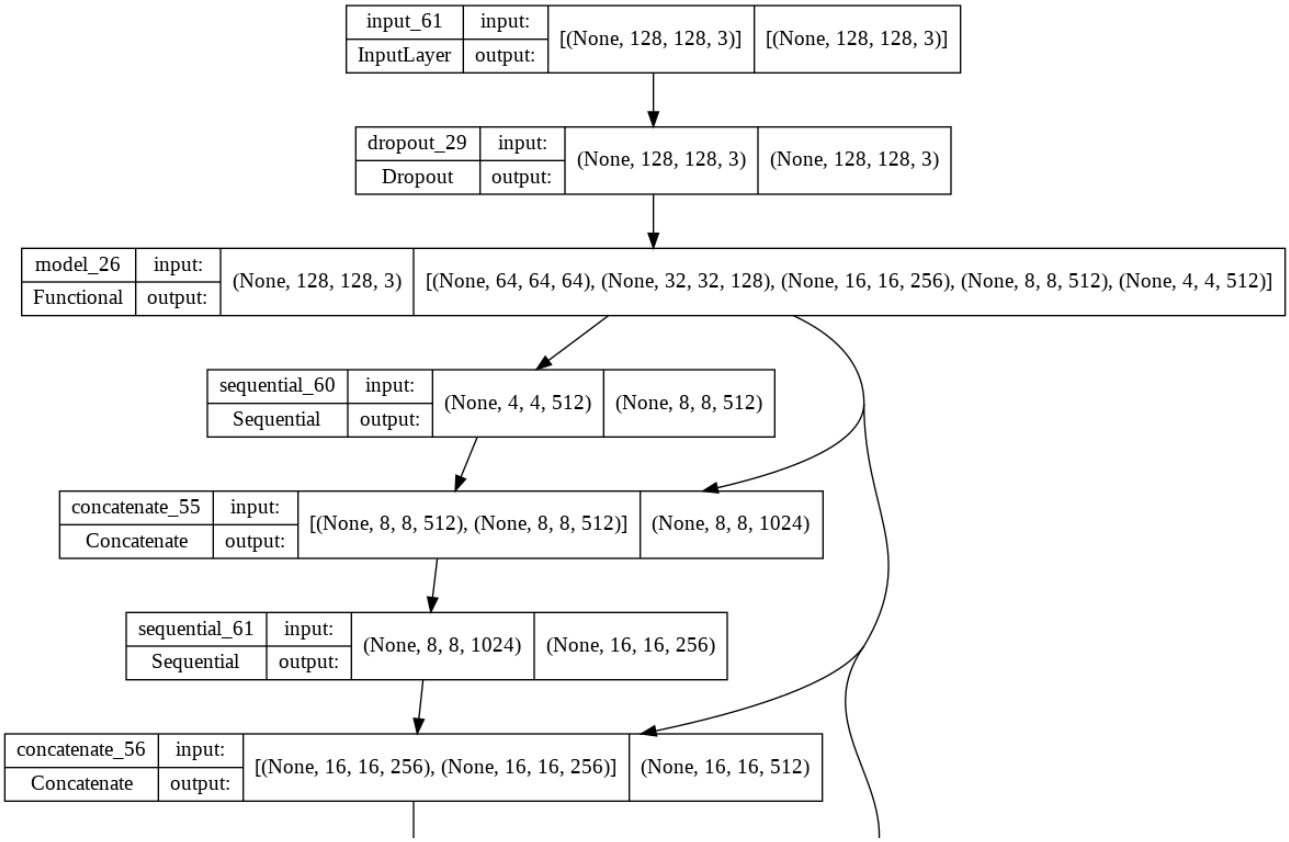
Definir o modelo Decodificador

```
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
  x = Dropout(0.05)(x) # Adicionei Dropout

  # 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)
```

Treinar Modelo

```
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()
```