Lab 7 - BCC406

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Detecção e Segmentação de objetos

Prof. Eduardo e Prof. Pedro

Objetivos:

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

Data da entrega : 04/12

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

Parte I - Detecção de Objetos (60pt)

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

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

Teste com imagens de:

- Praia
- Cachorros
- Pássartos

```
#instalando algumas dependências e carregando os utilitários necessários
!pip install numpy==1.24.3
!pip install protobuf==3.20.3

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')

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

```
Requirement already satisfied: numpy==1.24.3 in /usr/local/lib/python3.10/dist-packages (1.24.3)
Requirement already satisfied: protobuf==3.20.3 in /usr/local/lib/python3.10/dist-packages (3.20.3)
fatal: destination path 'models' already exists and is not an empty directory.
```

```
# Installing the Object Detection API
%%bash
sudo apt install -y protobuf-compiler
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
cp object_detection/packages/tf2/setup.py .
python -m pip install .
```

```
    Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle>=1.3.9->tf-models-official>=2.5.1-
    Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle>=1.3.9->tf-models-official>=2.5.1->ob
    Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle>=1.3.9->tf-models-official>
    Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle>=1.3.9->tf-models-official>=2.5.1-
    Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle>=1.3.9->tf-models-official>=2.5.1->
    Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from pymongo<5.0.0,>=3.8.0->a
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.24.0-
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.24.0->apache-beam
    Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models-o
    Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-model
    Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2
    Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models
    Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models-officia
    Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models-of
    Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models-of
    Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models-o
    Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models-official
    Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models-of
    Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-models
    Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.15.0->tf-mo
    Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow~=2.
    Requirement already satisfied: tf-keras>=2.14.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow-hub>=0.6.0->tf-models
    Requirement already satisfied: dm-tree~=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow-model-optimization>=0.4
    Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.10/dist-packages (from oauth2client->tf-models-official>=2
    Requirement already satisfied: pyasn1-modules>=0.0.5 in /usr/local/lib/python3.10/dist-packages (from oauth2client->tf-models-off
    Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from oauth2client->tf-models-official>=2.5.
    Requirement already satisfied: scikit-learn>=0.21.3 in /usr/local/lib/python3.10/dist-packages (from seqeval->tf-models-official>
    Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from tensorflow-datasets->tf-models-official>=2.
    Requirement already satisfied: etils[enp,epath,etree]>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow-datasets
    Requirement already satisfied: promise in /usr/local/lib/python3.10/dist-packages (from tensorflow-datasets->tf-models-official>=
    Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.10/dist-packages (from tensorflow-datasets->tf-model
    Requirement already satisfied: toml in /usr/local/lib/python3.10/dist-packages (from tensorflow-datasets->tf-models-official>=2.5
    Requirement already satisfied: array-record>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow-datasets->tf-model
    Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow~
    Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from etils[enp,epath,etree]>=0.9.0->tensorflow-
    Requirement already satisfied: importlib_resources in /usr/local/lib/python3.10/dist-packages (from etils[enp,epath,etree]>=0.9.0
    Requirement already satisfied: zipp in /usr/local/lib/python3.10/dist-packages (from etils[enp,epath,etree]>=0.9.0->tensorflow-da
    Requirement already satisfied: googleapis-common-protos<2.0.dev0,>=1.56.2 in /usr/local/lib/python3.10/dist-packages (from google
    Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.0dev,>=1.1
    Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21.3->seqeval->tf-m
    Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21.3->seqeva
    Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2
    Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorfl
    Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard
    Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorfl
    Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle>=1.3.9->tf-models-off
    Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle>=1.3.9
    Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>
    Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.
    Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-
    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.whl size=1697356 sha256=ee205577c113416f501449bf
      Stored in directory: /tmp/pip-ephem-wheel-cache-oeetkjsn/wheels/53/dd/70/2de274d6c443c69d367bd6a5606f95e5a6df61aacf1435ec0d
```

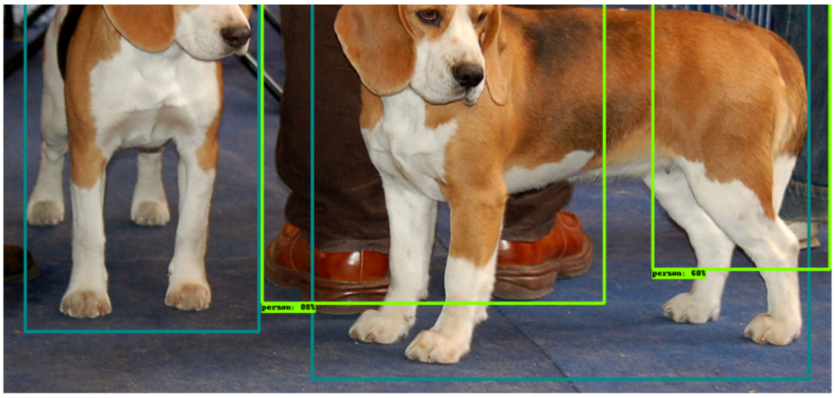
```python
# Importing dependencies
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.utils import ops as utils_ops

%matplotlib inline
```

```python
#carregando o mapa de rótulos e definindo os modelos e imagens para os testes
# Load label map data
PATH_TO_LABELS = './models/research/object_detection/data/mscoco_label_map.pbtxt'
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)

# Define models and images for testing
models_to_test = {
    'EfficientDet D0 512x512': 'https://tfhub.dev/tensorflow/efficientdet/d0/1',
    'SSD MobileNet V2 FPNLite 320x320': 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_320x320/1',
    'SSD ResNet50 V1 FPN 640x640 (RetinaNet50)': 'https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_640x640/1',
    'Faster R-CNN ResNet50 V1 640x640': 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1',
    'Mask R-CNN Inception ResNet V2 1024x1024': 'https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1'
}

images_to_test = {
    'Praia': 'models/research/object_detection/test_images/image2.jpg',
    'Cachorros': 'models/research/object_detection/test_images/image1.jpg',
    'Pássaros': 'https://upload.wikimedia.org/wikipedia/commons/0/09/The_smaller_British_birds_%288053836633%29.jpg',
}
```

```python
def load_image_into_numpy_array(path):
```

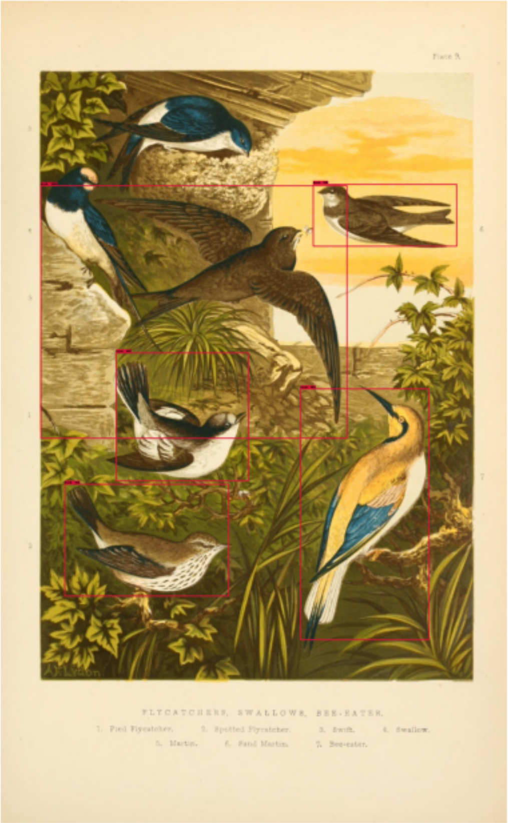
```
def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

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

    Args:
      path: the file path to the image

    Returns:
      uint8 numpy array with shape (1, 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)
```

```
# Para cada modelo e imagem, vamos carregar o modelo, selecionar a imagem, executar a inferência e visualizar os resultados
# Iterate over each model and image for testing
for model_name, model_handle in models_to_test.items():
    print(f"Testing model: {model_name}")
    print(f"Model handle: {model_handle}")
    hub_model = hub.load(model_handle)

    for image_name, image_path in images_to_test.items():
        print(f"Testing image: {image_name}")

        # Load the image
        image_np = load_image_into_numpy_array(image_path)

        # Running inference
        results = hub_model(image_np)
        result = {key: value.numpy() for key, value in results.items()}

        # Visualizing the results
        image_np_with_detections = image_np.copy()
        label_id_offset = 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=0.30,
            agnostic_mode=False)

        # Plot the image with detections
        plt.figure(figsize=(12, 8))
        plt.imshow(image_np_with_detections[0])
        plt.title(f"{model_name} - {image_name}")
        plt.axis('off')
        plt.show()
```



```
Testing image: Pássaros
```

Mask R-CNN Inception ResNet V2 1024x1024 - Pássaros



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))
```

```
import timeit

# Iterate over each model and image for testing
for model_name, model_handle in models_to_test.items():
    print(f"Testing model: {model_name}")
    print(f"Model handle: {model_handle}")
    hub_model = hub.load(model_handle)

    for image_name, image_path in images_to_test.items():
        print(f"Testing image: {image_name}")

        # Load the image
        image_np = load_image_into_numpy_array(image_path)

        # Measure inference time
        start_time = timeit.default_timer()

        # Running inference
        results = hub_model(image_np)

        end_time = timeit.default_timer()
        inference_time = end_time - start_time
        print(f"Inference time for {model_name} with {image_name}: {inference_time} seconds")
```

```
Testing model: EfficientDet D0 512x512
Model handle: https://tfhub.dev/tensorflow/efficientdet/d0/1
WARNING:absl:Importing a function (__inference___call___32344) with ops with unsaved custom gradients. Will likely fail if a gradien
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return_conditional_losses_97451) with ops with unsaved
WARNING:absl:Importing a function (__inference_bifpn_layer_call_and_return_conditional_losses_77595) with ops with unsaved custom gr
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return_conditional_losses_103456) with ops with unsave
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return_conditional_losses_93843) with ops with unsaved
WARNING:absl:Importing a function (__inference_EfficientDet-D0_layer_call_and_return_conditional_losses_107064) with ops with unsave
WARNING:absl:Importing a function (__inference_bifpn_layer_call_and_return_conditional_losses_75975) with ops with unsaved custom gr
Testing image: Praia
Inference time for EfficientDet D0 512x512 with Praia: 7.7485818359998575 seconds
Testing image: Cachorros
Inference time for EfficientDet D0 512x512 with Cachorros: 0.7046013189999485 seconds
Testing image: Pássaros
Inference time for EfficientDet D0 512x512 with Pássaros: 0.8734544639999058 seconds
Testing model: SSD MobileNet V2 FPNLite 320x320
Model handle: https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_320x320/1
Testing image: Praia
Inference time for SSD MobileNet V2 FPNLite 320x320 with Praia: 5.147866380000096 seconds
Testing image: Cachorros
Inference time for SSD MobileNet V2 FPNLite 320x320 with Cachorros: 0.21391165099998943 seconds
Testing image: Pássaros
Inference time for SSD MobileNet V2 FPNLite 320x320 with Pássaros: 0.4779764200000045 seconds
Testing model: SSD ResNet50 V1 FPN 640x640 (RetinaNet50)
Model handle: https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_640x640/1
Testing image: Praia
Inference time for SSD ResNet50 V1 FPN 640x640 (RetinaNet50) with Praia: 9.685456734000127 seconds
Testing image: Cachorros
Inference time for SSD ResNet50 V1 FPN 640x640 (RetinaNet50) with Cachorros: 3.1633863020001627 seconds
Testing image: Pássaros
Inference time for SSD ResNet50 V1 FPN 640x640 (RetinaNet50) with Pássaros: 3.696379805999868 seconds
Testing model: Faster R-CNN ResNet50 V1 640x640
Model handle: https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1
Testing image: Praia
Inference time for Faster R-CNN ResNet50 V1 640x640 with Praia: 12.790792164999857 seconds
Testing image: Cachorros
Inference time for Faster R-CNN ResNet50 V1 640x640 with Cachorros: 4.114566305999915 seconds
Testing image: Pássaros
Inference time for Faster R-CNN ResNet50 V1 640x640 with Pássaros: 5.174774370000023 seconds
Testing model: Mask R-CNN Inception ResNet V2 1024x1024
Model handle: https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1
Testing image: Praia
Inference time for Mask R-CNN Inception ResNet V2 1024x1024 with Praia: 78.20360851300006 seconds
Testing image: Cachorros
Inference time for Mask R-CNN Inception ResNet V2 1024x1024 with Cachorros: 73.08541657799992 seconds
Testing image: Pássaros
Inference time for Mask R-CNN Inception ResNet V2 1024x1024 with Pássaros: 59.02453252200007 seconds
```

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.

```
import struct
```

```
# Pre - Passo 0: Funções e estruturas auxiliares
# A classe WeightReader é usada para analisar o arquivo "yolov3.weights" e carregar os pesos do modelo na memória em um formato que pode
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].shape)))
                    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].shape)))
                    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


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.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 = len(anchors) // 2
    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[row, col, b, 4]
            if(objectness <= obj_thresh): continue
            # first 4 elements are x, y, w, and h
            x, y, w, h = netout[row, col, b, :4]
            x = (col + x) / grid_w
            y = (row + y) / grid_h
            # convert w and h from log to regular scale
            w = anchors[2 * b] * np.exp(w) / net_w
            h = anchors[2 * b + 1] * np.exp(h) / net_h
            classes = netout[row, col, b, 5:]

            # Atualiza a lista de caixas delimitadoras
            box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
            boxes.append(box)
    return boxes


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)):
        # Escala e deslocamento para redimensionar os boxes de acordo com as dimensões da imagem de entrada
        x_scale = float(net_w) / image_w
        y_scale = float(net_h) / image_h
        # Ajustar as coordenadas dos boxes
        boxes[i].xmin = int(boxes[i].xmin / x_scale)
        boxes[i].xmax = int(boxes[i].xmax / x_scale)
        boxes[i].ymin = int(boxes[i].ymin / y_scale)
        boxes[i].ymax = int(boxes[i].ymax / y_scale)

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

# Função para processar as saídas da rede neural e obter as caixas delimitadoras, labels e pontuações
def get_boxes(outputs, labels, class_threshold, image_shape):
    boxes = []
    confidences = []
```