Ex No: 8 **OBJECT DETECTION WITH YOLO3**

AIM:

To build an object detection model with YOLO3 using Keras/TensorFlow.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

PROGRAM:

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.


import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil


CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'data-for-yolo-v3-
kernel:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-
sets%2F81753%2F300187%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%3DGOOG4-RSA-
SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-
161607.iam.gserviceaccount.com%252F20241013%252Fauto%252Fstorage%252Fgoog4_request%2
```

```
6X-Goog-Date%3D20241013T134721Z%26X-Goog-Expires%3D259200%26X-Goog-
SignedHeaders%3Dhost%26X-Goog-
Signature%3D111d66e74f67e64fdba7c945042efbdae1215da134d52cf0c52c6a96cc4cde60f3b80f1ea6
e5820082e23d78f1c059e97b37381c855e53751064f7320567256db1283ba5484fadb539ff5b705b7fbef
6d59ba32b07900a140e7eca2dde2de99473d64369dc2f5d58c8dca00f63932deec3ba9c64effb6e1c4a22
156bf2241f36a2531348072fd38f36b3a9f54dd833383251f53462ccf2e402d42d3d15c231384cb8b8957
94710e7e83114cc26b134b8a1ad396c3126240d3328e4d2849790c95feb4b1fdb92fda78b5715af082c9
94d7d031a91744795141c700e68cdd8e0c159fcbca9acae1116b2fa43b0068ca1df76ff39f9b9242cd9806
b509e726ebac1'


KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'


!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)


try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass


for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
```

```
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')

import os
import numpy as np
import pandas as pd
import struct
```

```
import scipy.io
import scipy.misc
import PIL
import cv2
from skimage.transform import resize

import tensorflow as tf
from keras import backend as K
from keras.layers import Input, Lambda, Conv2D, BatchNormalization, LeakyReLU,
ZeroPadding2D, UpSampling2D
from keras.models import load_model, Model
from keras.layers import add, concatenate
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array

import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from matplotlib.patches import Rectangle

class Read_Weights:

    def __init__(self, file_name):

        with open(file_name, '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

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 darknet prefers
left and top

        x = Conv2D(conv['filter'],
                   conv['kernel'],
                   strides = conv['stride'],
                   padding = 'valid' if conv['stride'] > 1 else 'same', # peculiar padding as darknet prefers left
and top
                   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['layer_idx']))(x)
        if conv['leaky']: x = LeakyReLU(alpha = 0.1, name = 'leaky_' + str(conv['layer_idx']))(x)

    return add([skip_connection, x]) if skip else x

def make_yolov3_model():

    input_image = Input(shape=(None, None, 3))

    # Layers 0 to 4
    x = conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 0},
                                 {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
                                 {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
                                 {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])

    # Layers 5 to 8
    x = conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                       {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
                       {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])

    # Layers 9 to 11
```

```
    x = conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                       {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])

    # Layers 12 to 15
    x = conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                       {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
                       {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])

    # Layers 16 to 36
    for i in range(7):
        x = conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 16+i*3},
                           {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 17+i*3}])
    skip_36 = x

    # Layers 37 to 40
    x = conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 37},
                       {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 38},
                       {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 39}])

    # Layers 41 to 61
    for i in range(7):
        x = conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 41+i*3},
                           {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 42+i*3}])
    skip_61 = x

    # Layers 62 to 65
    x = conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 62},
                       {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 63},
                       {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 64}])

    # Layers 66 to 74
```

```
    for i in range(3):
        x = conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 
66+i*3},
                            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 
67+i*3}])

    # Layers 75 to 79
    x = conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 75},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 76},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 77},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 78},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 79}], 
skip=False)

    # Layers 80 to 82
    yolo_82 = conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True, 
'layer_idx': 80},
                             {'filter':  255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx': 81}], 
skip=False)

    # Layers 83 to 86
    x = conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 84}], 
skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_61])

    # Layers 87 to 91
    x = conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 87},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 88},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 89},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 90},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 91}], 
skip=False)

    # Layers 92 to 94
```

```
    yolo_94 = conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 92},
                                                                                    {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False,
'leaky': False, 'layer_idx': 93}], skip=False)


    # Layers 95 to 98
    x = conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,   'layer_idx':
96}], skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_36])


    # Layers 99 to 106
    yolo_106 = conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 99},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 100},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 101},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 102},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 103},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 104},
                               {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_idx': 105}],
skip=False)


    model = Model(input_image, [yolo_82, yolo_94, yolo_106])
    return model


# define the yolo v3 model
yolov3 = make_yolov3_model()


# load the weights
weight_reader = Read_Weights("../input/data-for-yolo-v3-kernel/yolov3.weights")


# set the weights
weight_reader.load_weights(yolov3)
```

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

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 = np.expand_dims(image, 0)
  return image, width, height

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

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)

def interval_overlap(interval_a, interval_b):
```

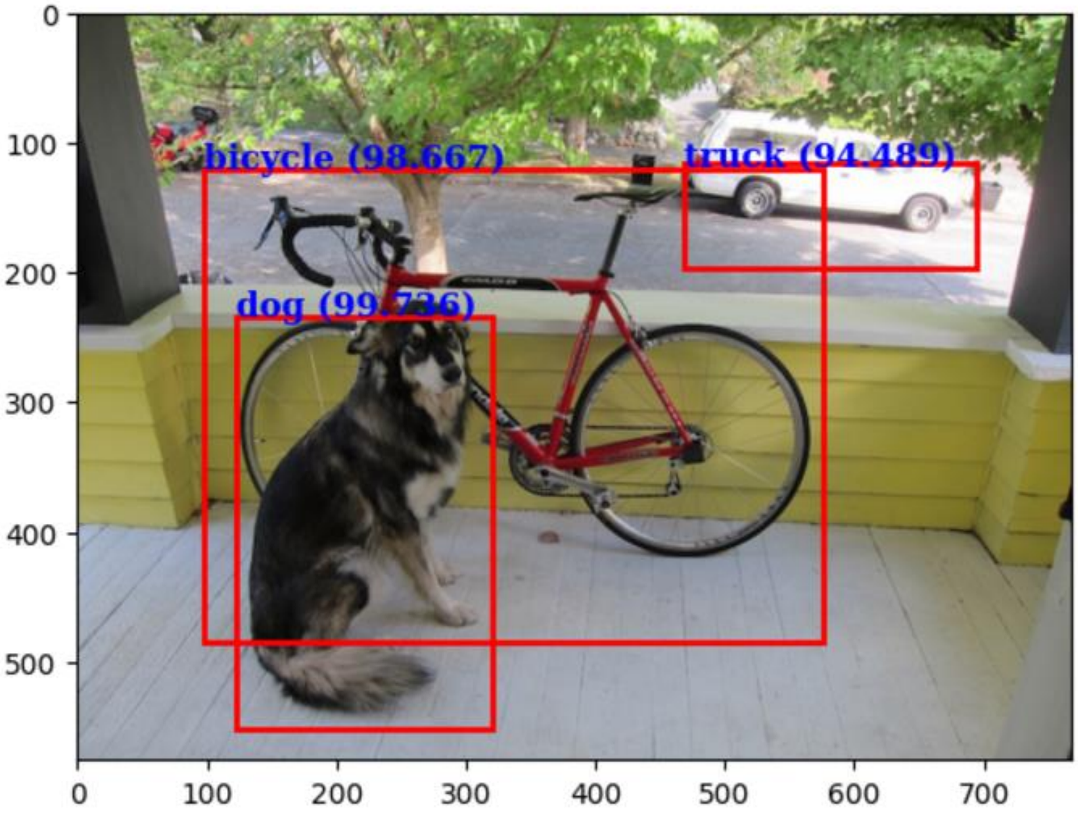
```
    x1, x2 = interval_a
    x3, x4 = interval_b

    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 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
```

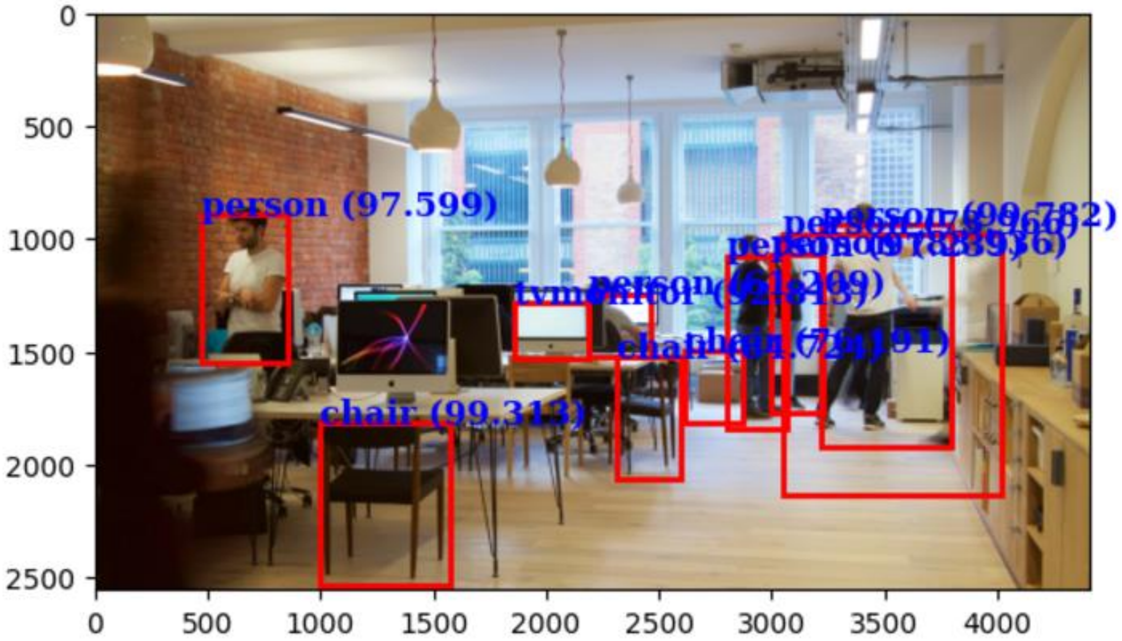
```
import numpy as np

def draw_boxes(filename, v_boxes, v_labels, v_scores):
    data = plt.imread(filename)
    print(f"Image Shape: {data.shape}")  # Debugging image shape

    # Convert grayscale to RGB if necessary
    if len(data.shape) == 2:  # Grayscale image
        data = np.stack([data] * 3, axis=-1)

    plt.imshow(data)
    ax = plt.gca()

    # Plot each box
    for i in range(len(v_boxes)):
        box = v_boxes[i]

        # Get coordinates and ensure they are floats
        y1, x1, y2, x2 = float(box.ymin), float(box.xmin), float(box.ymax), float(box.xmax)
        width, height = x2 - x1, y2 - y1

        # Debugging: Check types and box values
        print(f"Box: {box}, x1: {x1}, y1: {y1}, width: {width}, height: {height}")
        print(f"Label: {v_labels[i]}, Score: {v_scores[i]}, Type of Score: {type(v_scores[i])}")
        print(f"x1: {x1}, y1: {y1}, Type of x1: {type(x1)}, Type of y1: {type(y1)}")

        # Create the shape
        rect = plt.Rectangle((x1, y1), width, height, fill=False, color='red', linewidth=2)

        # Draw the box
        ax.add_patch(rect)
```

```
        # Format the label
        label = f"{v_labels[i]} ({v_scores[i]:.3f})"  # Ensure label and score are formatted correctly

        # Draw text and score in the top left corner
        plt.text(x1, y1, label, color='b', fontsize=12, family='serif', fontweight='bold')

    # Show the plot
    plt.show()

# 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", "banana",
    "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
    "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse",
    "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator",
    "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]

ls, v_scores)
image_names = ["../input/data-for-yolo-v3-kernel/dog.jpg", "../input/data-for-yolo-v3-
kernel/office.jpg"]
predict_boxes(image_names)
```

OUTPUT:




```
x1: 996.0, y1: 1815.0, Type of x1: <class 'float'>, Type of y1: <class 'float'>
```



RESULT:

Thus an object detection model with YOLO3 using Keras/TensorFlow is built.