# 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",

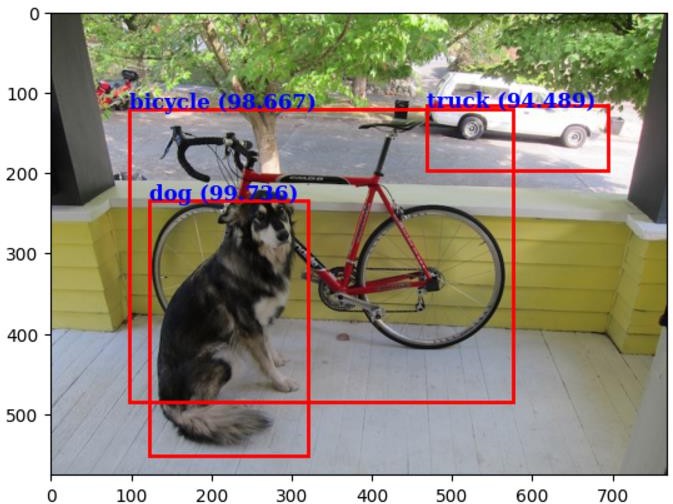
"chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse", "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]

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:



**RESULT:**

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