Derin Öğrenme ve OpenCV ile Zehirli Balıkların Gerçek Zamanlı veya Resimler ile Tespiti

Bu çalışmada opencv ve derin makina öğrenmesi kullanılarak ülkemizde yaşayan bazı zehirli balıklar tespit edilmiştir.Kullanılan yazılım dili python, kullanılan ide ise jupyter notebooktur.

Dataset:

Dataset google,bing search api ve facebooktan toplanmıştır.Toplam sınıf sayısı 5(aslan,sokar,trakonya,iskorpit ve balon),toplam resim sayısı ise 740(200 aslan,40 sokar,200 trakonya,100 iskorpit,200 balon) dır.

Aslan Balığı:

Üst kısımlarında bulunan iğnelerin insan ile teması sonucunda birkaç gün süren yanma, terleme ve solunum zorluğu görülebilir, hatta ölüme bile neden olabilir.

Balon Balığı:

Balon balığı zehri bünyesinde besleyen bir türdür. Karaciğerlerinde “tetrodotoksin” maddesi vardır. Bu madde ise zehirli yönüyle dikkat çeker.
TTX koduyla bilimsel kaynaklarda yer alan tetrodotoksin, balon balığını tüketen kişinin vücuduna yayılır.
Zehirli bir özelliği olan TTX kimyasalı, siyanürden daha etkili bir zehirdir.

Trakonya Balığı:

Trakonya, halk arasında Çarpan balığı Akdeniz, Marmara ve Ege denizlerinde yaşayan zehirli balık türüdür. Türkiye sularında çok fazla görülen bu balık türü bazı insanların dokunması ve teması halinde kalp krizi geçirmesine sebep olup ölümüne yol açabilecek derece toksin zehir salgılamaktadır.

Sokar Balığı:

Kare kuyruklu tavşan balığı olarak da bilinen esmer omurga ayağı, Siganidae familyasına ait bir tavşan balığı olan deniz ışınlı yüzgeçli bir balık türüdür. Süveyş Kanalı yoluyla Akdeniz'e yayılan batı Hint Okyanusu'na özgüdür. Yüzgeç dikenleri zehir içerir.

İskorpit Balığı:

İskorpit, Scorpaenidae familyasından bir balık türü. Yaşamı aynı aileden olan Lipsos'un aynıdır. Farkları İskorpit'in Lipsos'a göre daha küçük olması ve renginin koyuluğudur. Eti oldukça lezzetlidir ancak dikenleri çok zehirli olduğundan yakalandığında dikkat edilmelidir.

Uygulanan Adımlar:

- **Veri setinin oluşturulması :**

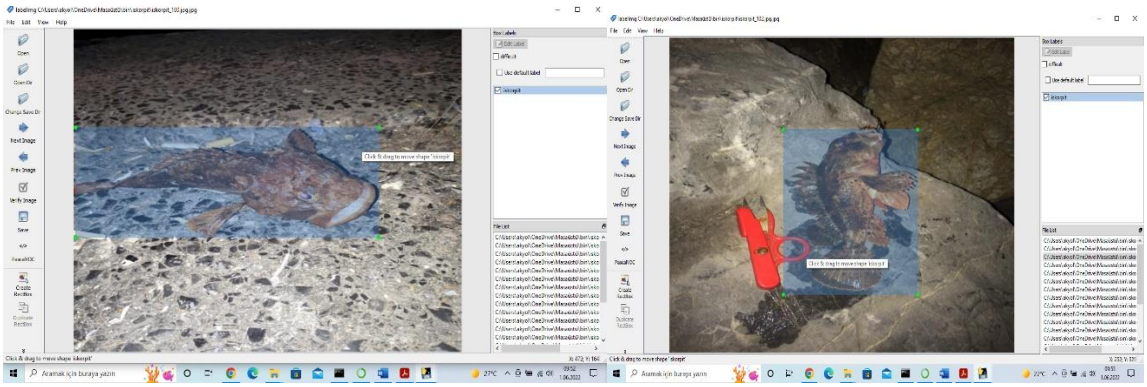
  Dataset Google, bing search api ve facebooktan toplanmıştır.

- **Veri setinin düzenlenmesi(Resimlerin boyutu ve formatı eğitim süresini düşürmek için düzenlendi) :**

  Resimler 640x480x3 formatına dönüştürülmüştür.

- **Veri setinin Labelimg programı ile etiketlenmesi :**

  Veriler LabelImg programı ile etiketlenmiştir.LabelImg balığın hangi koordinatlarda bulunduğunu xml formatında kaydetmemizi sağlamaktadır.



- **Veri setinin train ve test olmak üzere 2 klasörde toplanması**

  Verilerin %80 train yüzde %20'lik kısmı test için ayrılmıştır.

- **Tensorflow object detection apı ve daha önce eğitilmiş modellerinden olan mobilenet ssd'in indirilip kurulması :**

  Tensorflow Object Detection API önceden eğitilmiş modelleri kullanmamızı sağlayan bir nesne tanıma aracıdır. Bu modeller COCO veri seti ile eğitilmiş olup günlük hayatta fazlaca karşılaştığımız 80 tane nesneyi tanıyabilir. MobileNet ise kısaca bahsetmek gerekirse bir resim sınıflandırma kütüphanesidir. Aşağıdaki resimde tensorflow 2.0 zoo içinde bulunan daha önce eğitilmiş modeller gösterilmektedir. Speed sütunundaki değerler modellerin "ms" cinsinden hızlarını temsil ediyor. Map sütunundaki değerler ise modellerin COCO veri setindeki nesneleri tanıma başarısını temsil ediyor. Ben buradan mobilenet v2 modelini seçtim.

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
CenterNet MobileNetV2 FPN 512x512	6	23.4	Boxes
CenterNet MobileNetV2 FPN Keypoints 512x512	6	41.7	Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes

- **Etiket haritası oluşturulması**

  Toplam 5 adet sınıfımız var bunlar aslan balığı,balon balığı,trakonya balığı,sokar balığı ve iskorpit balığıdır. labels = [{'name':'balon', 'id':1}, {'name':'iskorpit', 'id':2}, {'name':'sokar', 'id':3}, {'name':'aslan', 'id':4}, {'name':'trakonya', 'id':5}]

- **Verilerin TensorflowRecord formatına dönüştürülmesi**

  Verilerimizi eğitime sokmadan önce txt formatında veya csv oluşturduğumuz label mapı tfrecord formatına dönüştürüyoruz.Tfrecord formatı Büyük veri kümeleriyle çalışıyorsanız, verilerinizin depolanması için ikili dosya biçiminin kullanılması, içe

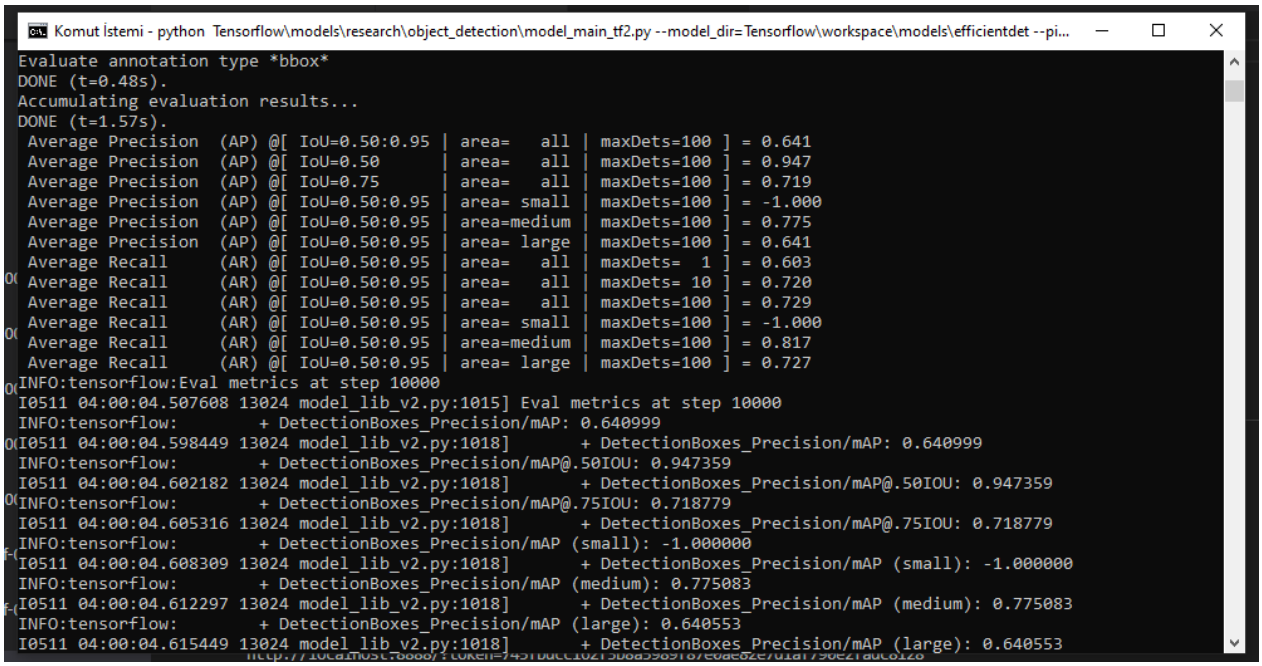
aktarma işlem hattınızın performansı ve bunun sonucunda modelinizin eğitim süresi üzerinde önemli bir etkiye sahip olmaktadır.

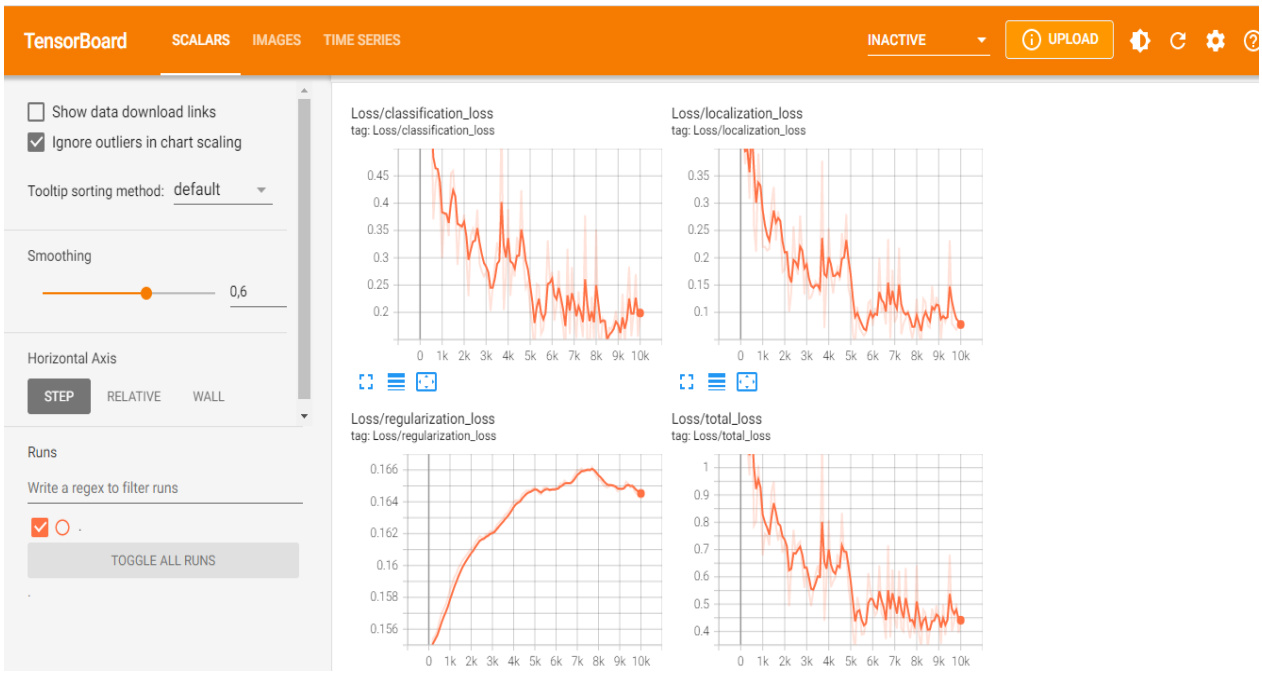
- **Modelin eğitilmesi**

  Model toplam 10000 tur eğitilmiştir.

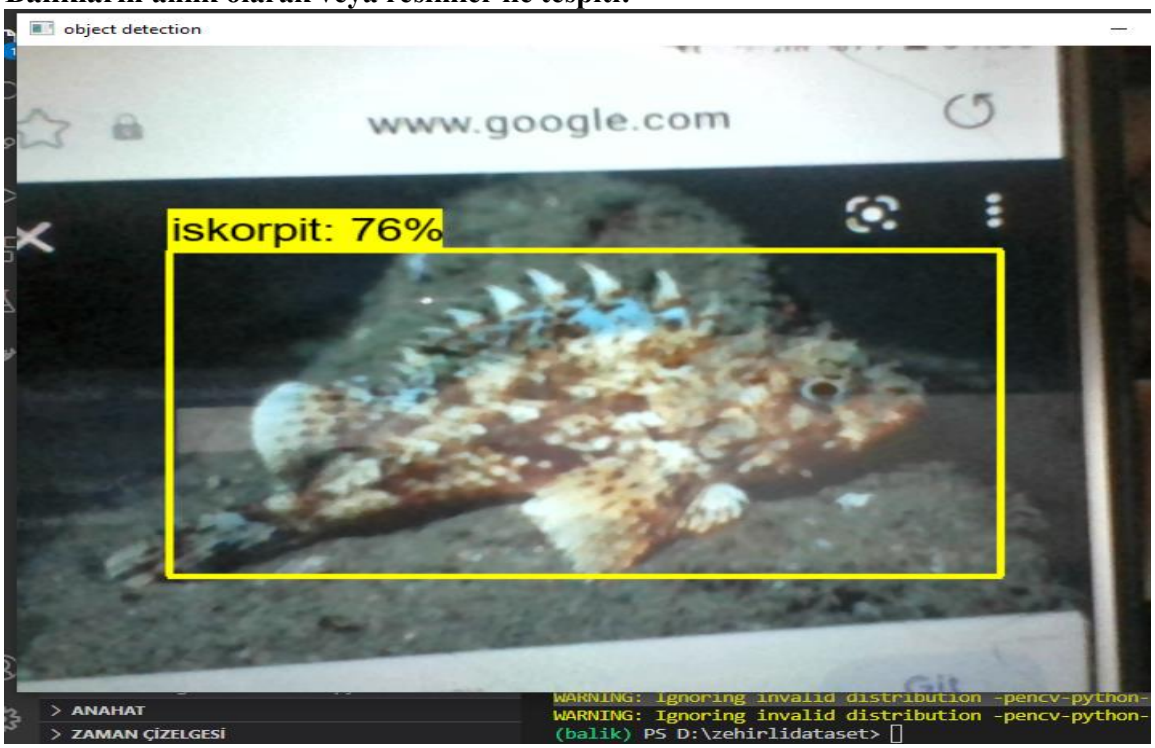
- **Modelin derlenmesi**

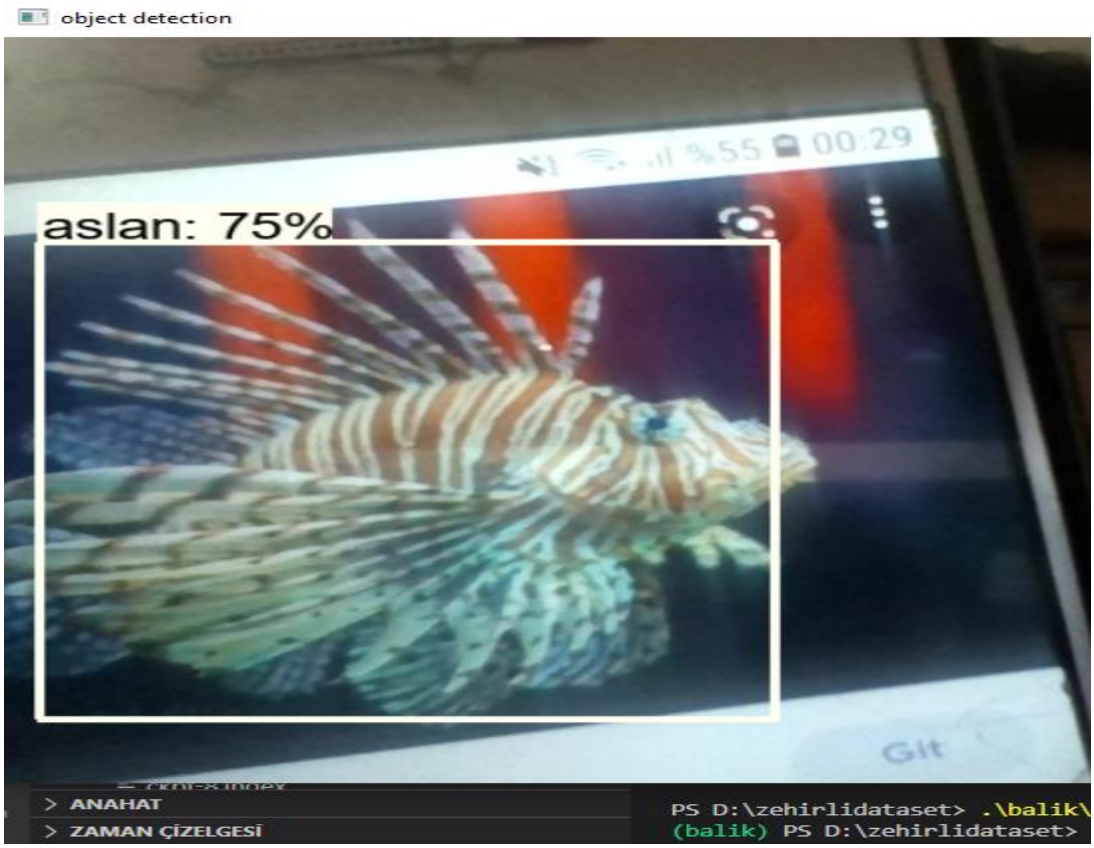
  Modelin başarı oranı (%64) ve kayıplar aşağıdaki resimde gösterilmiştir.

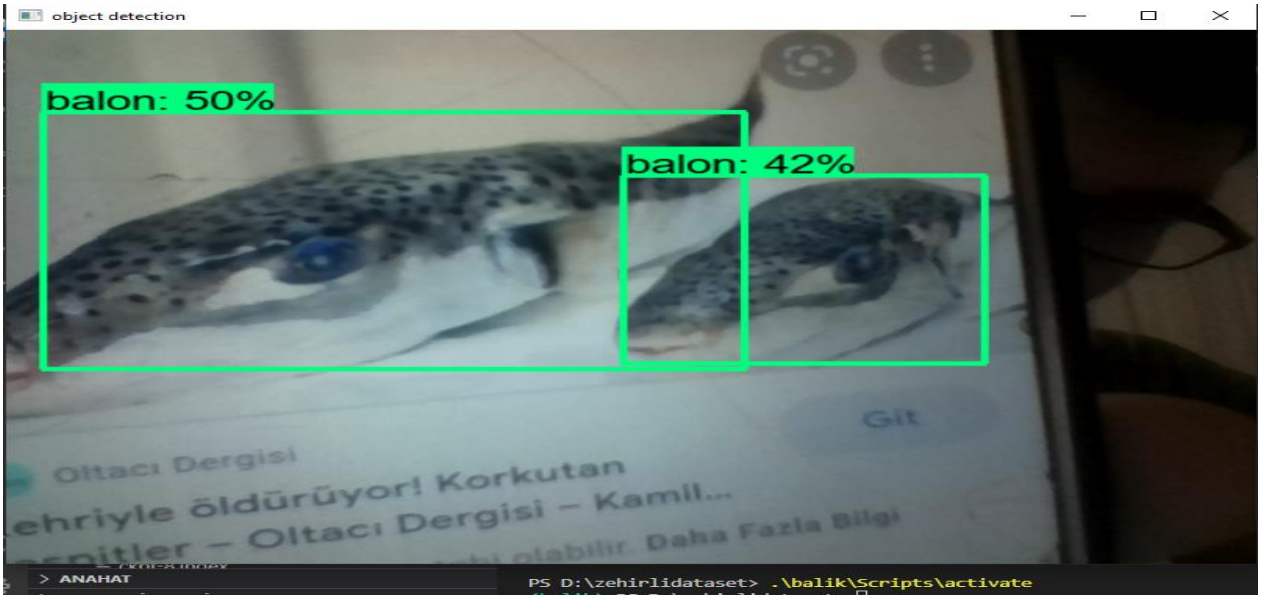


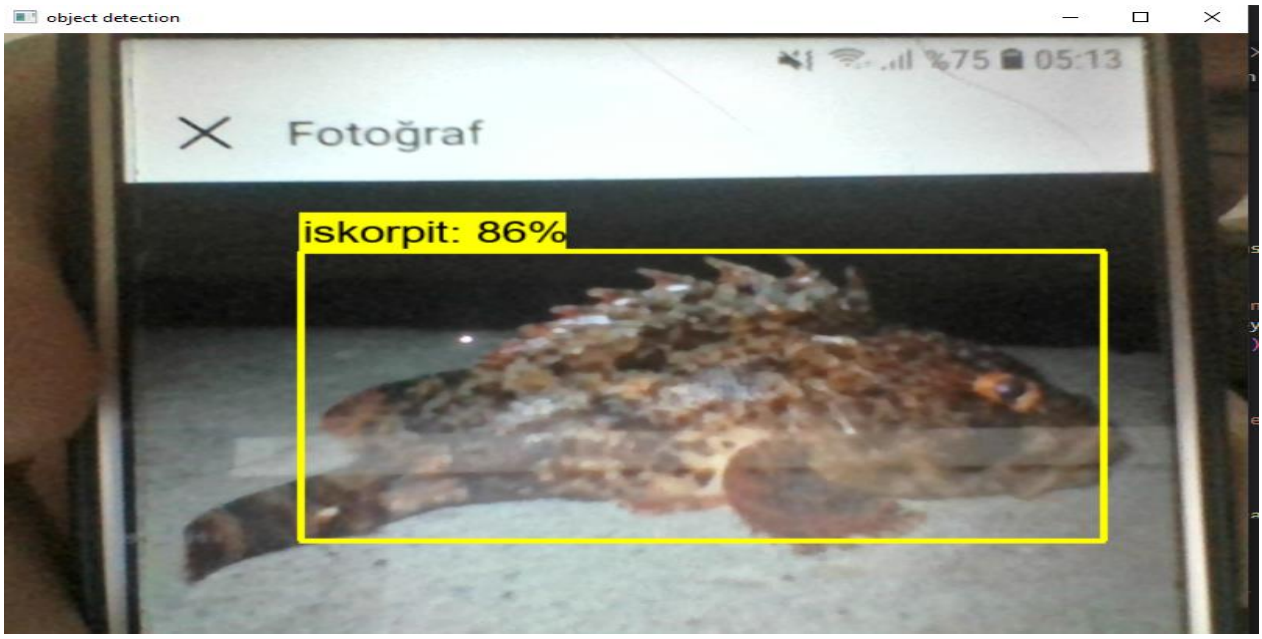


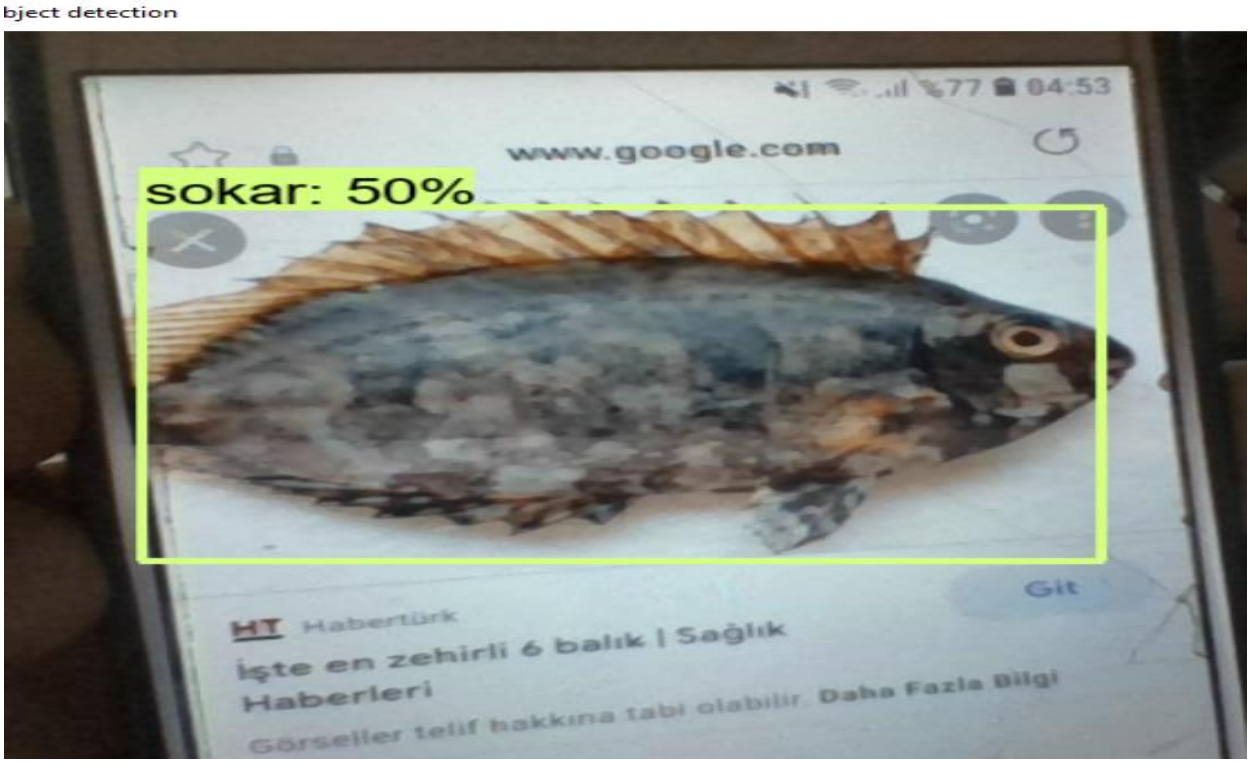
- **Balıkların anlık olarak veya resimler ile tespiti:**

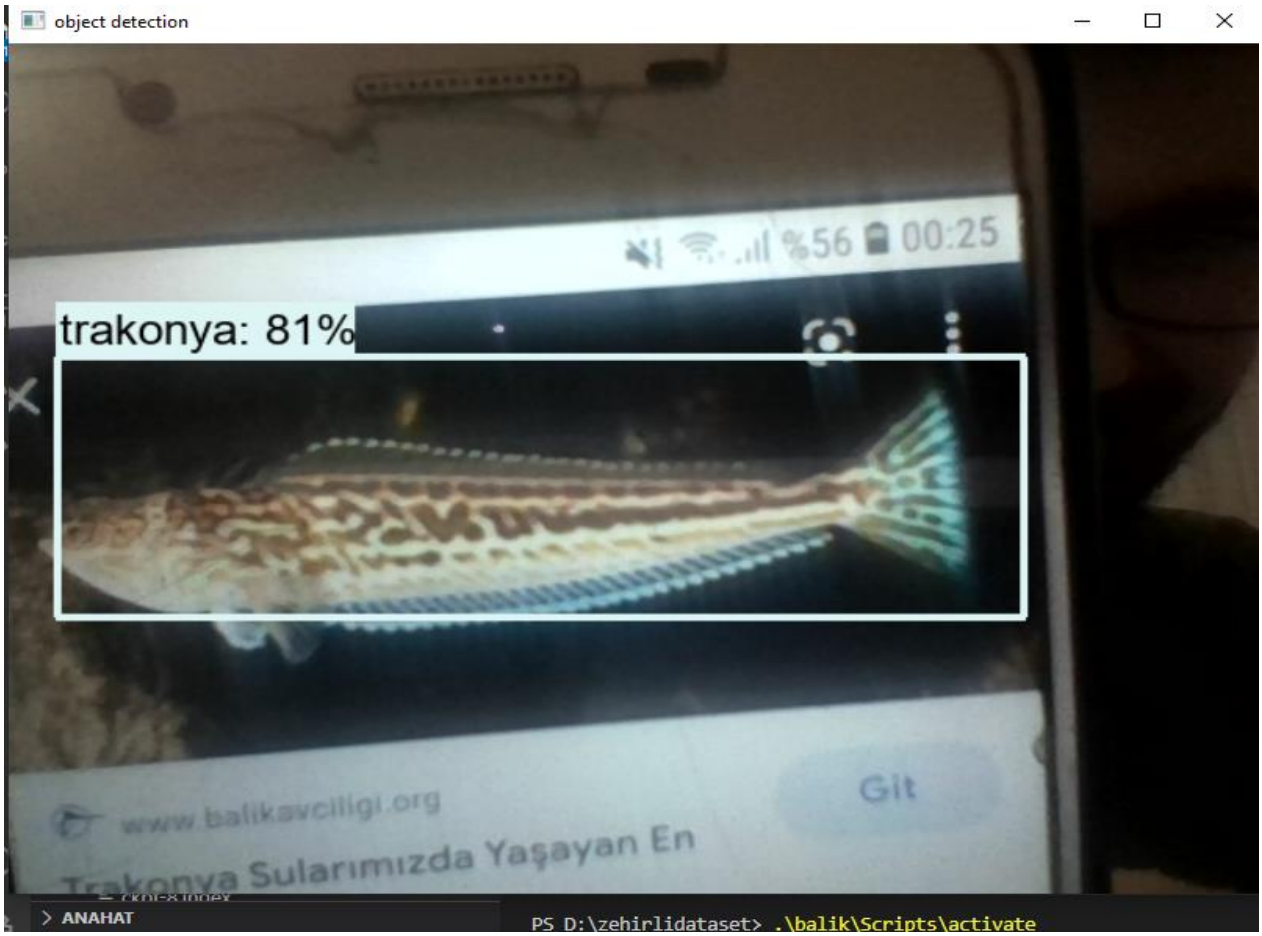



object detection
aslan: 75%
> ANAHAT
> ZAMAN ÇİZELGESİ
PS D:\zehirlidataset> .\balik\
(balik) PS D:\zehirlidataset>


object detection
balon: 50%
balon: 42%
> ANAHAT
PS D:\zehirlidataset> .\balik\Scripts\activate


object detection
%75 05:13
Fotoğraf
iskorpit: 86%


bject detection
%77 04:53
www.google.com
sokar: 50%
Git
HT Habertürk
İşte en zehirli 6 balık | Sağlık Haberleri
Görseller telif hakkına tabi olabilir. Daha Fazla Bilgi



Kodlar:

- **Bing Search API ile Veri çekilmesi:**
- ```
  from requests import exceptions
  import argparse
  import requests
  import cv2
  import os
  ap = argparse.ArgumentParser()
  ap.add_argument("-q", "--query", required=True,
  	help="search query to search Bing Image API for")
  ap.add_argument("-o", "--output", required=True,
  	help="path to output directory of images")
  args = vars(ap.parse_args())
  API_KEY = "API KEYINIZ"
  MAX_RESULTS = 250
  GROUP_SIZE = 50
  URL = "https://api.bing.microsoft.com/v7.0/images/search"
  EXCEPTIONS = set([IOError, FileNotFoundError,
  	exceptions.RequestException, exceptions.HTTPError,
  	exceptions.ConnectionError, exceptions.Timeout])
  term = args["query"]
  headers = {"Ocp-Apim-Subscription-Key" : API_KEY}
  ```
  ```

```
params = {"q": term, "offset": 0, "count": GROUP_SIZE}
print("[INFO] searching Bing API for '{}'".format(term))
search = requests.get(URL, headers=headers, params=params)
search.raise_for_status()
results = search.json()
estNumResults = min(results["totalEstimatedMatches"], MAX_RESULTS)
print("[INFO] {} total results for '{}'".format(estNumResults,
      term))
total = 0
for offset in range(0, estNumResults, GROUP_SIZE):
      # update the search parameters using the current offset, then
      # make the request to fetch the results
      print("[INFO] making request for group {}-{} of {}...".format(
            offset, offset + GROUP_SIZE, estNumResults))
      params["offset"] = offset
      search = requests.get(URL, headers=headers, params=params)
      search.raise_for_status()
      results = search.json()
      print("[INFO] saving images for group {}-{} of {}...".format(
            offset, offset + GROUP_SIZE, estNumResults))
 # loop over the results
      for v in results["value"]:
            # try to download the image
            try:
                  # make a request to download the image
                  print("[INFO] fetching: {}".format(v["contentUrl"]))
                  r = requests.get(v["contentUrl"], timeout=30)
                  # build the path to the output image
                  ext = v["contentUrl"][v["contentUrl"].rfind("."):]
                  p = os.path.sep.join([args["output"], "{}{}".format(
                        str(total).zfill(8), ext)])
                  # write the image to disk
                  f = open(p, "wb")
                  f.write(r.content)
                  f.close()
            # catch any errors that would not unable us to download the
            # image
            except Exception as e:
                  # check to see if our exception is in our list of
                  # exceptions to check for
                  if type(e) in EXCEPTIONS:
                        print("[INFO] skipping:
{}".format(v["contentUrl"]))
                        continue
# try to load the image from disk
            image = cv2.imread(p)
            # if the image is `None` then we could not properly load
the
            # image from disk (so it should be ignored)
            if image is None:
```

```
                print("[INFO] deleting: {}".format(p))
                os.remove(p)
                continue
            # update the counter
            total += 1
```

Tfrecord formatına dönüştürmek için kodlar:

```
import os
import glob
import pandas as pd
import io
import xml.etree.ElementTree as ET
import argparse

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow
logging (1)
import tensorflow.compat.v1 as tf
from PIL import Image
from object_detection.utils import dataset_util, label_map_util
from collections import namedtuple
```

Initiate argument parser

```
parser = argparse.ArgumentParser(
    description="Sample TensorFlow XML-to-TFRecord converter")
parser.add_argument("-x",
                    "--xml_dir",
                    help="Path to the folder where the input .xml files
are stored.",
                    type=str)
parser.add_argument("-l",
                    "--labels_path",
                    help="Path to the labels (.pbtxt) file.", type=str)
parser.add_argument("-o",
                    "--output_path",
                    help="Path of output TFRecord (.record) file.",
type=str)
parser.add_argument("-i",
                    "--image_dir",
                    help="Path to the folder where the input image
files are stored. "
                         "Defaults to the same directory as XML_DIR.",
                    type=str, default=None)
parser.add_argument("-c",
                    "--csv_path",
                    help="Path of output .csv file. If none provided,
then no file will be "
                         "written.",
                    type=str, default=None)

args = parser.parse_args()
```

```
if args.image_dir is None:
    args.image_dir = args.xml_dir

label_map = label_map_util.load_labelmap(args.labels_path)
label_map_dict = label_map_util.get_label_map_dict(label_map)

def xml_to_csv(path):
    """Iterates through all .xml files (generated by labelImg) in a
given directory and combines
    them in a single Pandas dataframe.
    Parameters:
    ----------
    path : str
        The path containing the .xml files
    Returns
    -------
    Pandas DataFrame
        The produced dataframe
    """
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[5][0].text),
                     int(member[5][1].text),
                     int(member[5][2].text),
                     int(member[5][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height',
                   'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

def class_text_to_int(row_label):
    return label_map_dict[row_label]

def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in
zip(gb.groups.keys(), gb.groups)]

def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path,
'{}'.format(group.filename)), 'rb') as fid:
```

```
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size
    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []
    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))
    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin':
dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax':
dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin':
dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax':
dataset_util.float_list_feature(ymaxs),
        'image/object/class/text':
dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label':
dataset_util.int64_list_feature(classes),
    }))
    return tf_example

def main(_):
    writer = tf.python_io.TFRecordWriter(args.output_path)
    path = os.path.join(args.image_dir)
    examples = xml_to_csv(args.xml_dir)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())
    writer.close()
```

```
    print('Successfully created the TFRecord file:
{}'.format(args.output_path))
    if args.csv_path is not None:
        examples.to_csv(args.csv_path, index=None)
        print('Successfully created the CSV file:
{}'.format(args.csv_path))

if __name__ == '__main__':
    tf.app.run()
```

Eğitim ve Balık Tespiti kodlar:

0. Yolların Oluşturulması

```
import os

CUSTOM_MODEL_NAME = 'mobilnet'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL =
'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite
_320x320_coco17_tpu-8.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
#.\balik\Scripts\activate


paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME,
'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME,
'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }

files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models',
CUSTOM_MODEL_NAME, 'pipeline.config'),
```

```
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'],
TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}

for path in paths.values():
    if not os.path.exists(path):
            !mkdir {path}
```

1. Tensorflow object detection apı ve daha önce eğitilmiş modelin indirilmesi

```
# https://www.tensorflow.org/install/source_windows


!pip install wget
import wget
```

```
Requirement already satisfied: wget in d:\zehirlidataset\balik\lib\site-packages (3.2)
```

```
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}


url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-
win64.zip"
wget.download(url)
!move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
!cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'],
'bin'))
!cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. &&
copy object_detection\\packages\\tf2\\setup.py setup.py && python setup.py build && python
setup.py install
!cd Tensorflow/models/research/slim && pip install -e .

VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'builders', 'model_builder_tf2_test.py')
# Verify Installation
!python {VERIFICATION_SCRIPT}

import object_detection


wget.download(PRETRAINED_MODEL_URL)
!move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
!cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf
{PRETRAINED_MODEL_NAME+'.tar.gz'}
```

2. Etiket haritası oluşturulması

```
labels = [{'name':'balon', 'id':1}, {'name':'iskorpit', 'id':2}, {'name':'sokar', 'id':3}, {'name':'aslan', 'id':4}, {'name':'trakonya', 'id':5}]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

3. TFrecords oluşturulması

```
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}

!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')}
```

```
Successfully created the TFRecord file: Tensorflow\workspace\annotations\train.record
Successfully created the TFRecord file: Tensorflow\workspace\annotations\test.record
```

4. Model bilgilerinin train klasörüne taşınması

```
    !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}
```

```
        1 file(s) copied.
```

5. Bilgilerin transfer learning için güncellenmesi

```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format

config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])

config
```

```
{'model': ssd {
  num_classes: 90
  image_resizer {
    fixed_shape_resizer {
      height: 320
      width: 320
```

```
  }
}
feature_extractor {
  type: "ssd_mobilenet_v2_fpn_keras"
  depth_multiplier: 1.0
  min_depth: 16
  conv_hyperparams {
    regularizer {
      l2_regularizer {
        weight: 3.9999998989515007e-05
      }
    }
    initializer {
      random_normal_initializer {
        mean: 0.0
        stddev: 0.009999999776482582
      }
    }
    activation: RELU_6
    batch_norm {
      decay: 0.996999979019165
      scale: true
      epsilon: 0.0010000000474974513
    }
  }
  use_depthwise: true
  override_base_feature_extractor_hyperparams: true
  fpn {
    min_level: 3
    max_level: 7
    additional_layer_depth: 128
  }
}
box_coder {
  faster_rcnn_box_coder {
    y_scale: 10.0
    x_scale: 10.0
    height_scale: 5.0
    width_scale: 5.0
  }
}
matcher {
  argmax_matcher {
    matched_threshold: 0.5
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
```

```
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.9999998989515007e-05
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 128
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
    share_prediction_tower: true
    use_depthwise: true
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    scales_per_octave: 2
  }
```

```
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 9.99999993922529e-09
      iou_threshold: 0.6000000238418579
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: false
    }
    score_converter: SIGMOID
  }
  normalize_loss_by_num_matches: true
  loss {
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    classification_loss {
      weighted_sigmoid_focal {
        gamma: 2.0
        alpha: 0.25
      }
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  encode_background_as_zeros: true
  normalize_loc_loss_by_codesize: true
  inplace_batchnorm_update: true
  freeze_batchnorm: false
},
'train_config': batch_size: 128
data_augmentation_options {
  random_horizontal_flip {
  }
}
data_augmentation_options {
  random_crop_image {
    min_object_covered: 0.0
    min_aspect_ratio: 0.75
    max_aspect_ratio: 3.0
    min_area: 0.75
    max_area: 1.0
    overlap_thresh: 0.0
  }
}
sync_replicas: true
```

```
optimizer {
  momentum_optimizer {
    learning_rate {
      cosine_decay_learning_rate {
        learning_rate_base: 0.07999999821186066
        total_steps: 50000
        warmup_learning_rate: 0.026666000485420227
        warmup_steps: 1000
      }
    }
    momentum_optimizer_value: 0.8999999761581421
  }
  use_moving_average: false
}
fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED"
num_steps: 50000
startup_delay_steps: 0.0
replicas_to_aggregate: 8
max_number_of_boxes: 100
unpad_groundtruth_tensors: false
fine_tune_checkpoint_type: "classification"
fine_tune_checkpoint_version: V2,
'train_input_config': label_map_path: "PATH_TO_BE_CONFIGURED"
tf_record_input_reader {
  input_path: "PATH_TO_BE_CONFIGURED"
},
'eval_config': metrics_set: "coco_detection_metrics"
use_moving_averages: false,
'eval_input_configs': [label_map_path: "PATH_TO_BE_CONFIGURED"
shuffle: false
num_epochs: 1
tf_record_input_reader {
  input_path: "PATH_TO_BE_CONFIGURED"
}
],
'eval_input_config': label_map_path: "PATH_TO_BE_CONFIGURED"
shuffle: false
num_epochs: 1
tf_record_input_reader {
  input_path: "PATH_TO_BE_CONFIGURED"
}}
```

```
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
```

```
pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint =
os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
'checkpoint', 'ckpt-0')
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'test.record')]

config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)
```

6. Eğitim

```
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection',
'model_main_tf2.py')

command = "python {} --model_dir={} --pipeline_config_path={} --
num_train_steps=10000".format(TRAINING_SCRIPT,
paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])

print(command)
```

```
python Tensorflow\models\research\object_detection\model_main_tf2.py --
model_dir=Tensorflow\workspace\models\efficientdet --
pipeline_config_path=Tensorflow\workspace\models\efficientdet\pipeline.config --
num_train_steps=10000
```

```
!{command}
```

7. Modelin Derlenmesi

```
command = "python {} --model_dir={} --pipeline_config_path={} --
checkpoint_dir={}".format(TRAINING_SCRIPT,
paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])

print(command)
```

```
python Tensorflow\models\research\object_detection\model_main_tf2.py --
model_dir=Tensorflow\workspace\models\efficientdet --
pipeline_config_path=Tensorflow\workspace\models\efficientdet\pipeline.config --
checkpoint_dir=Tensorflow\workspace\models\efficientdet
```

```
!{command}
```

8. Modelin son checkpoint noktasından cagrılması

```
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util

configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)


ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-11')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

9. Resimden Tespit

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])

IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'trakonya_171.jpg')

img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections

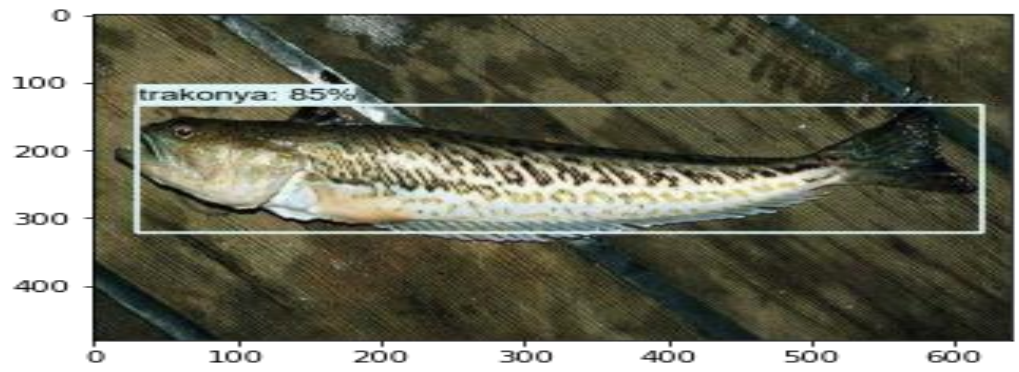
# detection_classes should be ints.
```

```
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes']+label_id_offset,
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=5,
        min_score_thresh=.4,
        agnostic_mode=False)

plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```



10. Webcamdan tespit

```
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections
```

```
# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes']+label_id_offset,
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=5,
        min_score_thresh=.4,
        agnostic_mode=False)

cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

if cv2.waitKey(10) & 0xFF == ord('q'):
    cap.release()
    cv2.destroyAllWindows()
    break
```