4.3. Руководство к практическому занятию 2.

Содержание

Что такое TensorFlow

Установка пакета TensorFlow

Модель SSD MobileNet

Функция загрузки модели нейронной сети

Карта загрузочных меток

Сегментация(выделение) распознанных экземпляров

Дообучение сети

Маркировка объектов(элементов) на изображении

Проект в Google Colab

OPENCV ДЛЯ РАСПОЗНАВАНИЯ ЭЛЕМЕНТОВ (ОБЪЕКТОВ) В ВИДЕОПОТОКЕ

Что такое OpenCV

Установка и настройка пакета

Программа поиска объектов веб-камерой. Основной алгоритм

Основной цикл программы

Запуск программы

Код программы

Что такое TensorFlow

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для R, C Sharp, C++, Haskell, Java, Go и Swift. Изначально TensorFlow была разработана командой Google Brain для внутреннего использования в Google, но в 2015 году система была переведена в свободный доступ с открытой лицензией Apache 2.0.

TensorFlow может работать на многих параллельных процессорах, как CPU, так и GPU, опираясь на архитектуру CUDA для поддержки вычислений общего назначения на графических процессорах. TensorFlow доступна для 64-разрядных Linux, macOS, Windows, и для мобильных вычислительных платформ, включая Android и iOS.

Вычисления TensorFlow выражаются в виде потоков данных через граф состояний. Название TensorFlow происходит от операций с многомерными массивами данных, которые также называются «тензорами».

В мае 2016 года Google сообщила о применении для задач глубинного обучения аппаратного ускорителя собственной разработки — тензорного процессора (TPU) — специализированной интегральной схемы, адаптированной под задачи для TensorFlow, и обеспечивающей высокую производительность в арифметике пониженной точности (например, для 8-битной архитектуры) и направленной скорее на применение моделей, чем на их обучение.

Сообщалось, что после использования TPU в собственных задачах Google по обработке данных удалось добиться на порядок лучших показателей продуктивности на ватт затраченной энергии.

Установка пакета TensorFlow

```
pip install -U --pre tensorflow=="2.*"

pip install tf_slim
```

```
Successfully installed tf-estimator-nightly-2.8.0.dev2021122109
Collecting tf_slim
  Downloading tf_slim-1.1.0-py2.py3-none-any.whl (352 kB)
     |████████████████████████████████| 352 kB 4.7 MB/s
Requirement already satisfied: absl-py>=0.2.2 in /usr/local/lib/pyt
Requirement already satisfied: six in /usr/local/lib/python3.7/dist
Installing collected packages: tf-slim
Successfully installed tf-slim-1.1.0
```

Будем использовать датасет – MS-COCO (Microsoft Common Objects in Context – общие объекты в контенте). Цель выпуска этого набора фирмой Microsoft – способствовать развитию распознавания объектов, помещая распознавание объектов в более широкую проблему понимания сцены. Набор данных построен путем сбора изображений сложных сцен повседневной жизни на естественном фоне. Все экземпляры объектов на изображении индивидуально сегментированы и помечены, что помогает повысить точность позиционирования объекта. Этот набор данных содержит изображения 91 типа объектов, которые легко распознаются 4-летними детьми. Набор данных содержит 328 000 изображений и 2,5 миллиона помеченных примеров.

Для работы с этим дататсетом необходим фреймворк `PYCOCOTOOLS.` Устанавливаем его:

```
pip install pycocotools
```

Далее проверим установлены ли модели в родительском каталоге и если еще не установлены – клонируем их с GITHUB-узла моделей фреймворка TensorFlow:

```
import os

import pathlib


if "models" in pathlib.Path.cwd().parts:

  while "models" in pathlib.Path.cwd().parts:

    os.chdir('..')

elif not pathlib.Path('models').exists():

  !git clone --depth 1 https://github.com/tensorflow/models
```

Теперь необходимо скомпилировать модели с помощью команты protoc:

```
%%bash

cd models/research/

protoc object_detection/protos/*.proto --python_out=.
```

Инсталлируем пакет object_detection package:

```
%%bash

cd models/research/

protoc object_detection/protos/*.proto --python_out=.

cp object_detection/packages/tf2/setup.py .
```

```
python -m pip install .
```

Импортируем все нужные библиотеки:

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display
```

Также импортируем из object_detection package:

```
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils
                                    as vis_util

# настроим опции tf1 в `utils.ops`
# tf.compat - модуль совместимости, позволяет писать код, который
# работает, как в TensorFlow 1.x, так и в 2.x.
utils_ops.tf = tf.compat.v1

# настроим путь к файлу gfile
tf.gfile = tf.io.gfile
```

Основное назначение модуля tf.gfile:

1. Предоставить API, близкий к файловым объектам Python
2. Предоставить реализацию на основе API файловой системы TensorFlow C ++. Для того, чтобы C ++ FileSystem API поддерживает несколько реализаций файловых систем, включая локальные файлы, Google Cloud Storage (начиная с gs: //) и HDFS (начиная с hdfs: /). TensorFlow экспортирует их как tf.gfile, чтобы можно было использовать эти реализации для сохранения и загрузки контрольных точек, записи журналов TensorBoard и доступа к данным обучения (среди других целей). Однако, если все файлы являются локальными, вы можете использовать обычный файловый API Python без каких-либо проблем.

Модель SSD MobileNet

SSD (Single Shot MultiBox Detector) - это новый тип алгоритма обнаружения цели в области глубокого обучения. За последние несколько международных соревнований алгоритм SSD добился отличных результатов с точки зрения скорости и точности, что когда-то было далеко от других алгоритмов обнаружения. Поток алгоритма SSD может быть сведен к нескольким этапам: создание области кандидата, выбор кадра, оценка и фильтрация. Среди них алгоритмы для генерации областей-кандидатов, выбора кадра и фильтрации являются фиксированными, и для данной области-кандидата необходимо использовать разные модели, чтобы определить, является ли изображение в области целью, подлежащей обнаружению. Часто с этим алгоритмом используются модели распознавания образов – VGG, Inception, ResNet, MobileNet и другие. Среди них модель SSD на основе MobileNet имеет самую быструю скорость обнаружения.

Функция загрузки модели нейронной сети

Любая модель, экспортируемая с помощью export_inference_graph.py можно загрузить в следующей функции, просто изменив путь. По умолчанию мы используем здесь модель "SSD with Mobile net". Список других моделей, которые можно запускать "из коробки" с различной скоростью и точностью, см. в зоопарке моделей обнаружения.

```
def load_model(model_name):

  base_url = 'http://download.tensorflow.org/models/object_detection/'

  model_file = model_name + '.tar.gz'

  model_dir = tf.keras.utils.get_file( fname=model_name, origin=base_url + model_file,untar=True)
```

```
  model_dir = pathlib.Path(model_dir)/"saved_model"

  model = tf.saved_model.load(str(model_dir))

  return model
```

Карта загрузочных меток

Карты меток сопоставляют индексы с названиями категорий, так что, когда наша сеть свертки предсказывает число 5, мы знаем, что это соответствует, например, самолету. Здесь мы используем внутренние служебные функции, возвращающие словарь, сопоставляющий целые числа с соответствующими строковыми метками.

```
# Список строк, которые используются для добавления правильной метки для

# каждого поля

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)
```

Для простоты мы проведем тестирование на 3х изображениях:

```
# Если вы хотите протестировать код с вашими изображениями,

# просто добавьте путь к изображениям в TEST_IMAGE_PATHS.

PATH_TO_TEST_IMAGES_DIR = pathlib.Path('models/research/object_detection/test_images')

TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.jpg")))

TEST_IMAGE_PATHS
```

```
[PosixPath('models/research/object_detection/test_images/image1.jpg'),
 PosixPath('models/research/object_detection/test_images/image2.jpg'),
 PosixPath('models/research/object_detection/test_images/image3.jpg')]
```

Например, если хотим проверить свои фото на гугл-диске – подключаем его:

```
from google.colab import drive

drive.mount("/content/drive/")

!ls /content/drive/MyDrive/TestRecPics

IMG_1.jpg  IMG_2.jpg


PATH_TO_TEST_IMAGES_DIR = pathlib.Path('/content/drive/MyDrive/TestRecPics')

TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.jpg")))

TEST_IMAGE_PATHS

                IMG_1.jpg  IMG_2.jpg
```

Загружаем модель нейронной сети «SSD with Mobile net»:

```
model_name = 'ssd_mobilenet_v1_coco_2017_11_17'
```

```
detection_model = load_model(model_name)
```

```
Downloading data from http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_2017_11_17.tar.gz
76537856/76534733 [==============================] - 2s 0us/step
76546048/76534733 [==============================] - 2s 0us/step
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
```

Проверяем входную сигнатуру модели, она ожидает пакет из 3-цветных изображений типа uint8:

```
print(detection_model.signatures['serving_default'].inputs)
```

```
[<tf.Tensor 'image_tensor:0' shape=(None, None, None, 3) dtype=uint8>]
```

Эта модель возвращает несколько выходных данных:

```
detection_model.signatures['serving_default'].output_dtypes
```

```
{'detection_boxes': tf.float32,
 'detection_classes': tf.float32,
 'detection_scores': tf.float32,
 'num_detections': tf.float32}
```

```
detection_model.signatures['serving_default'].output_shapes
```

```
{'detection_boxes': TensorShape([None, 100, 4]),
 'detection_classes': TensorShape([None, 100]),
 'detection_scores': TensorShape([None, 100]),
 'num_detections': TensorShape([None])}
```

Добавим функцию-оболочку для вызова модели и очистки выходных данных:

```
def run_inference_for_single_image(model, image):
  image = np.asarray(image)
  # На входе должен быть тензор,
  # конвертируем image используя `tf.convert_to_tensor`.
  input_tensor = tf.convert_to_tensor(image)

  # модель ожидает пакет изображений,
  # поэтому добавляем ось(измерение)
  # с помощью метода with `tf.newaxis`.
  input_tensor = input_tensor[tf.newaxis,...]

  # запускаем нейросеть для вывода результатов
  model_fn = model.signatures['serving_default']
  output_dict = model_fn(input_tensor)
  # Все выходные данные являются пакетом тензоров пакетов.
```

```
  # Преобразуем в массивы numpy и возьмем  индекс [0],
  # чтобы удалить пакетное измерение.
  # Нас интересуют только первое измерение - num_detections.

  num_detections = int(output_dict.pop('num_detections'))
  output_dict = {key:value[0, :num_detections].numpy()
                 for key,value in output_dict.items()}
  output_dict['num_detections'] = num_detections
  # detection_classes должен быть целочисленным
  output_dict['detection_classes']=output_dict['detection_classes'].astype(np.int64)

   # Обработаем модель с помощью масок:
   if 'detection_masks' in output_dict:
     # Изменим формат маски bbox в соответствии
     # с размером изображения.
     detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
               output_dict['detection_masks'], output_dict['detection_boxes'],
               image.shape[0], image.shape[1])
     detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5, tf.uint8)
     output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()
  return output_dict
```

Теперь напишем функцию, которая запускает предыдущую на каждом тестовом изображении и показывает результаты:

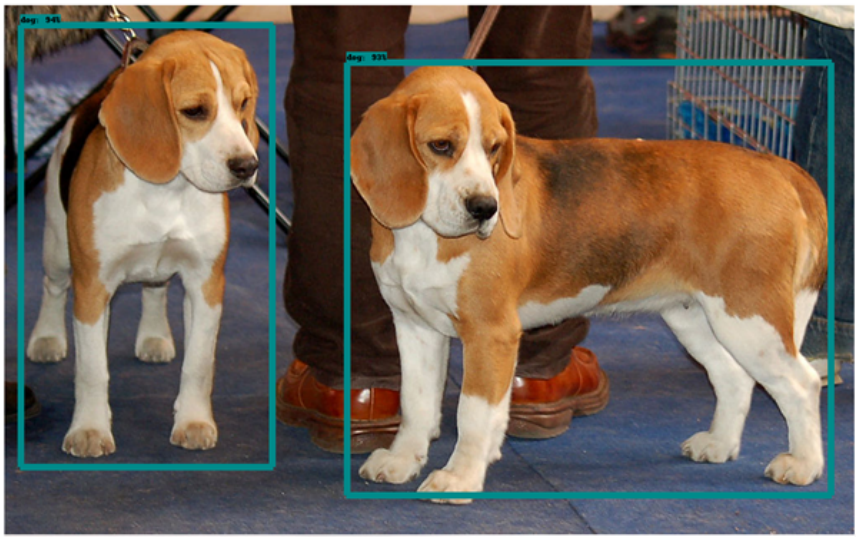
```
def show_inference(model, image_path):
  # представление изображения на основе массива
  # будет использовано позже для подготовки
  # результирующего изображения с рамками и надписями на них.
  image_np = np.array(Image.open(image_path))
  # Запускаем распознавание
  output_dict = run_inference_for_single_image(model, image_np)
  # Визуализируем результаты распознавания
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks_reframed', None),
```

```
    use_normalized_coordinates=True,
    line_thickness=8)
  display(Image.fromarray(image_np))
```

Запускаем все в работу на распознавание:

```
for image_path in TEST_IMAGE_PATHS:
  show_inference(detection_model, image_path)
```

Получаем следующий результат:



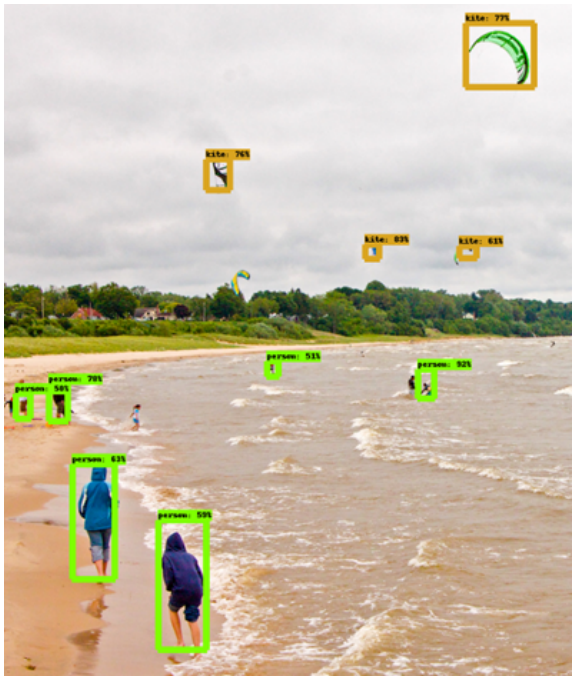


Рис.1. Результат классификации элементов на изображении в TensorFlow.

Сегментация(выделение) распознанных экземпляров

Выбираем модель нейронной сети с сегментацией:

```
model_name = "mask_rcnn_inception_resnet_v2_atrous_coco_2018_01_28"

masking_model = load_model(model_name)
```

Модель сегментации экземпляра включает в себя выходные данные detection_masks:

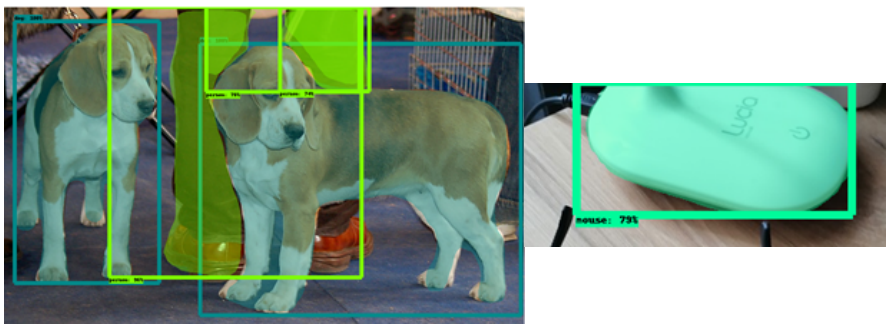
```
masking_model.signatures['serving_default'].output_shapes
```

```
{'detection_boxes': TensorShape([None, 100, 4]),
 'detection_classes': TensorShape([None, 100]),
 'detection_masks': TensorShape([None, None, 33, 33]),
 'detection_scores': TensorShape([None, 100]),
 'num_detections': TensorShape([None])}
```

Запускаем модель с сегментацией:

```
    for image_path in TEST_IMAGE_PATHS:
      show_inference(masking_model, image_path)
```

Получаем следующий результат:



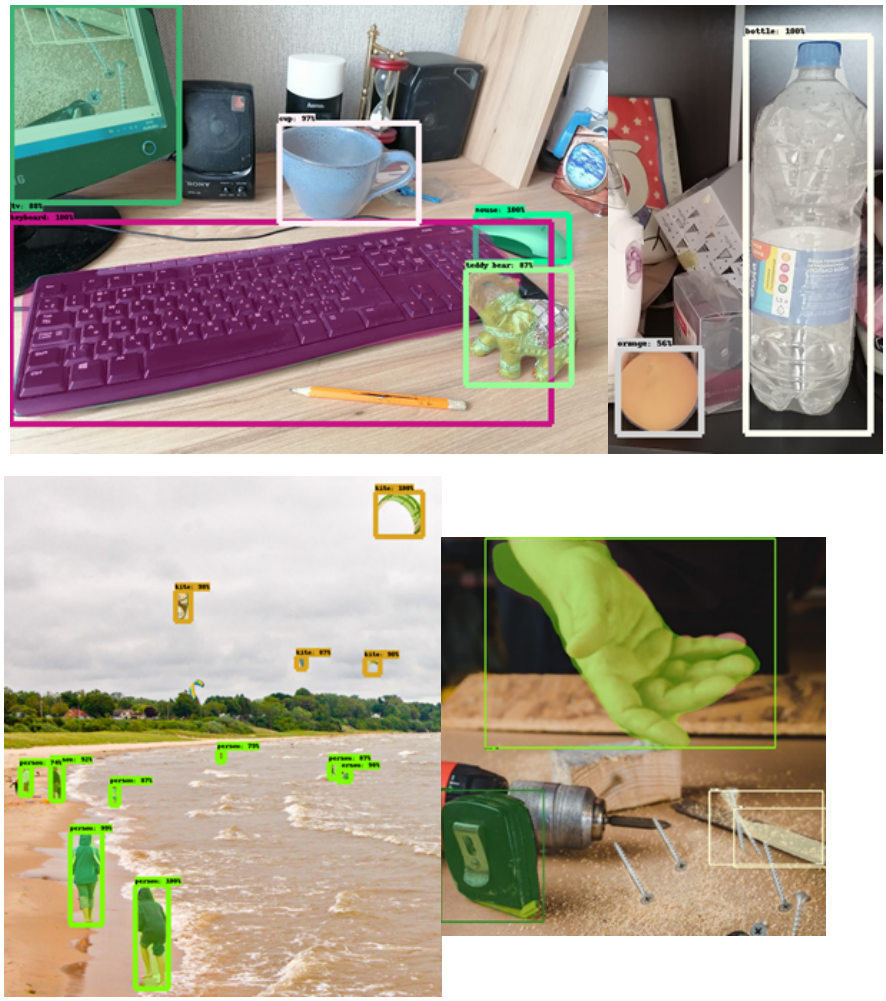


Рис.2. Результат сегментации элементов на изображении в TensorFlow.

Дообучение сети

Установим программу, которая позволит грузить картинки Google по запросу, сначала клонируем (скачаем) ее:

```
!git clone https://github.com/Joeclinton1/google-images-download.git
```

Перейдем в папку для установки:

```
%cd google-images-download/
```

Устанавливаем пакет:

```
!python setup.py install
```

Переходим в папку, где хранится скрипт загрузки файлов:

```
%cd google_images_download/
```

```
/content/google-images-download/google_images_download
```

Загружаем 100 картинок карандашей:

```
!python google_images_download.py --keywords "pencil"
                                  --limit 100 --format jpg
```

Поскольку картинки имеют различный размер, напишем функцию, которая преобразуем их к одинаковому размеру:

```
from PIL import Image

import os

import argparse

def rescale_images(directory, size):
```

```
    print(directory,' ',size)
    print(os.listdir(directory))
    i=1
    for img in os.listdir(directory):
      im = Image.open(directory+img)
      print(i,': '+directory+img,'\n')
      im_resized = im.resize(size, Image.ANTIALIAS)
      im_resized.save(directory+img)
      i+=1
```

И далее выполняем:

```
rescale_images('/content/google-images-
  download/google_images_download/downloads/pencil/', (800,600))
```

Теперь нам нужно переместить около 80% изображений в каталог **/content/models/research/object_detection/images/train** и 20% изображений в каталог **/content/models/research/object_detection/images/test**

Маркировка объектов(элементов) на изображении

Для маркировки элементов на изображении нужно установить дополнительное программное обеспечение **LabelImg** (официальныйы сайт):

На Линуксе:

```
!pip install pyqt5
!git clone https://github.com/tzutalin/labelImg.git
```

Можно установить в Windows в виртуальное окружение **PyCharm**:

```
pip install pyqt5
pip install pyqt5-tools
pip install lxml
pip install labelImg
cd venv\lib\site-packages\labelimg
python labelimg.py
```

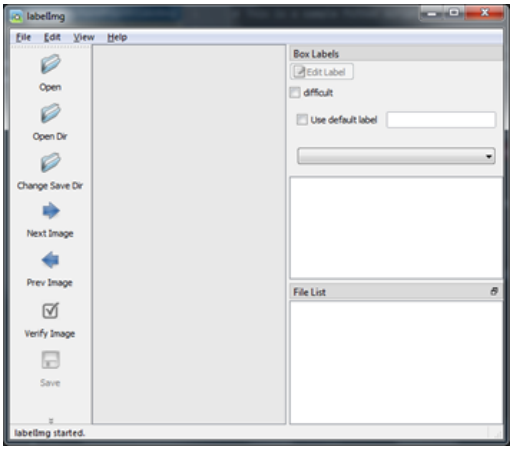


Рис.3. Интерфейс программы LabelImg.

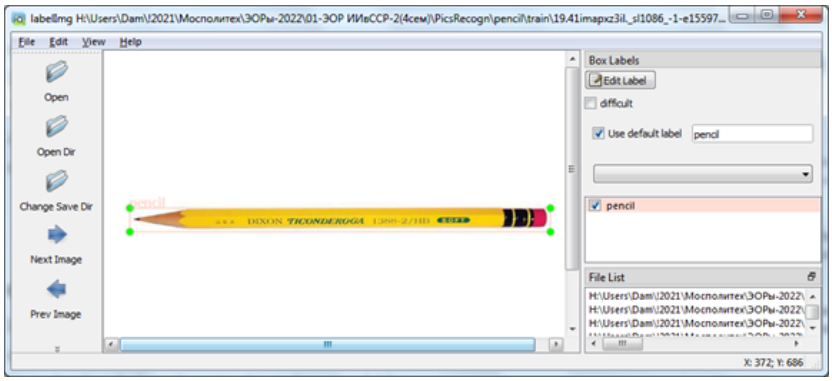


Рис.4. Пример маркировки элемента изображения (карандаш) в LabelImg.

После разметки элементов используются скрипты *xml_to_csv.py* и *generate_tfrecord.py* из репозитория https://github.com/datitran/raccoon_dataset для генерации csv файлов. Будут созданы файлы: **test_labels.csv**, и **train_labels.csv**.

На основе csv-файлов нужно создать TFRecords с помощью команд:

```
# train tfrecord
python generate_tfrecord.py --csv_input=images/train_labels.csv

 --image_dir=images/train --output_path=train.record

# test tfrecord
python generate_tfrecord.py --csv_input=images/test_labels.csv

 --image_dir=images/test --output_path=test.record
```

Создаются 2 файла: **train.record** и **test.record**, которые потом будут использованы в модели нейросети для распознавания элементов (например, наши карандаши) на изображении.

Более подробно дальнейший процесс дообучения нейросети рассмотрен тут.

Проект в Google Colab

Все вышеизложенные действия можно повторять и отлаживать работу, копируя инструкции и команды непосредственно в ячейки ноутбука в Google Colab. Наверное, это наиболее эффективный способ разобраться детально в каждой команде и результатах ее выполнения. Однако, можно использовать за основу уже готовый ноутбук с указанными действиями. В этом случае предварительно нужно пройти по ссылке ниже и сохранить копию ноутбука на своем Google-диске. Далее работать со своей копией ноутбука и отлаживать работу в ней. Вот **ссылка на проект в Google Colab.** Оригинальный проект в Google Colab можно посмотреть тут.

OPENCV ДЛЯ РАСПОЗНАВАНИЯ ЭЛЕМЕНТОВ (ОБЪЕКТОВ) В ВИДЕОПОТОКЕ

Что такое OpenCV

OpenCV — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD.

OpenCV используется везде, где нужно компьютерное зрение. Это позволяют устройству «увидеть», распознать и описать изображение. Компьютерное зрение дает точную информацию о том, что изображено на картинке, с описанием, характеристиками и размерами (с определенной степенью достоверности). В библиотеке есть модуль **DNN** (Deep Neural Network) для работы с нейронными сетями и машинным обучением. Для распознавания элементов в **OpenCV** используются очертания объектов, сегментация по цветам, встроенные методы распознавания, которые можно настраивать в зависимости от объекта и чувствительности алгоритма.

Новые версии библиотеки поддерживают работу не только с картинками, но и с видео. Они могут считывать ролики с использованием кодеков, анализировать происходящее в них, отслеживать движения и элементы. Это полезно, например, при программировании движущегося робота или создании ПО для камеры видеонаблюдения

Установка и настройка пакета

Чтобы сделать детектор объектов в реальном времени, потребуется:

- PyCharm+Python
- Получить доступ к веб-камере/видео потоку.
- Применить распознавание объекта для каждого кадра.

Для начала импортируем все нужные библиотеки для python через командную строку (pip install):

- TensorFlow
- OpenCV (pip install opencv-python)
- Keras (pip install keras)
- Matplotlib (pip install matplotlib)
- H5py(pip install h5py)
- imutils (pip install imutils)
- Скачать из репозитория https://github.com/C-Aniruddh/realtime_object_recognition два файла модели нейронной сети и поместить в папку проекта:
  - MobileNetSSD_deploy.caffemodel
  - MobileNetSSD_deploy.prototxt.txt

Программа поиска объектов веб-камерой. Основной алгоритм

1. Загружаем сериализованную модель нейронной сети, предоставляя ссылки на файл prototxt и файл модели.
2. Затем инициализируем видео поток (это может быть видеофайл или веб-камера). Для камеры сначала запускаем VideoStream, делаем паузу, пока камера включится, и, наконец, начинаем отсчёт кадров в секунду. Классы VideoStream и FPS являются частью пакета imutils.
3. Начинаем проходить циклами через detections – обнаруженные объекты, помня, что несколько объектов могут быть восприняты как единое изображение. Также делаем проверку на валидность (т.е. вероятность) для каждого обнаружения. Если валидность достаточно велика (т.е. выше заданного порога при вызове программы), отображаем предсказание в терминале, а также рисуем на видео потоке предсказание (обводим объект в цветной прямоугольник и вешаем лейбл).
4. В целом, нужно, чтобы лейбл располагался над цветным прямоугольником, однако, может возникнуть такая ситуация, что сверху будет недостаточно места, поэтому в таких случаях выводим лейбл под верхней стороной прямоугольника.
5. При выходе из цикла, останавливаем счётчик FPS и выводим информацию о конечном значении FPS в терминал.
6. Закрываем окно программы, прекращая видео поток.

В начале программы анализируем аргументы командной строки:

```
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
        help="путь к архитектуре сети Caffe - prototxt file")
ap.add_argument("-m", "--model", required=True,
        help="путь к модели сети (коэффициенты связей) Caffe")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
        help="минимальная вероятность, чтобы отбросить")
args = vars(ap.parse_args())
```

Затем инициализируем список классов и набор цветов:

```
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
```

```
    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

Теперь загрузим модель и настроим видео поток:

```
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
fps = FPS().start()
```

Основной цикл программы

```
while True:
    frame = vs.read()
    frame = cv2.flip(frame, -1)  # Flip camera vertically
    frame = imutils.resize(frame, width=400)

    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
        0.007843, (300, 300), 127.5)

    net.setInput(blob)
    detections = net.forward()

    for i in np.arange(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]

        if confidence > args["confidence"]:
            idx = int(detections[0, 0, i, 1])
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            label = "{}: {:.2f}%".format(CLASSES[idx],
                confidence * 100)
            cv2.rectangle(frame, (startX, startY), (endX, endY),
                COLORS[idx], 2)
            y = startY - 15 if startY - 15 > 15 else startY + 15
            cv2.putText(frame, label, (startX, y),
```

```
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
            break

        fps.update()
```

Запуск программы

Запускаем программу с помощью команды:

```
python main.py --prototxt MobileNetSSD_deploy.prototxt.txt
    --model MobileNetSSD_deploy.caffemodel
```

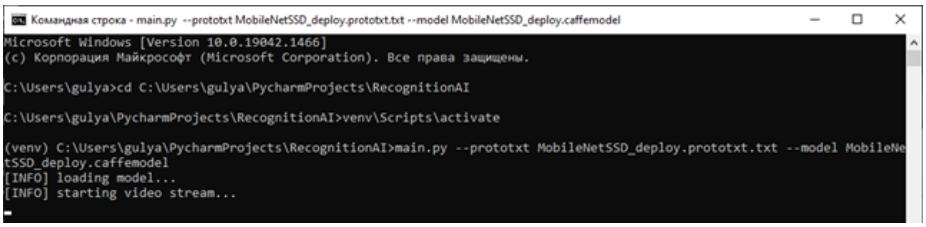
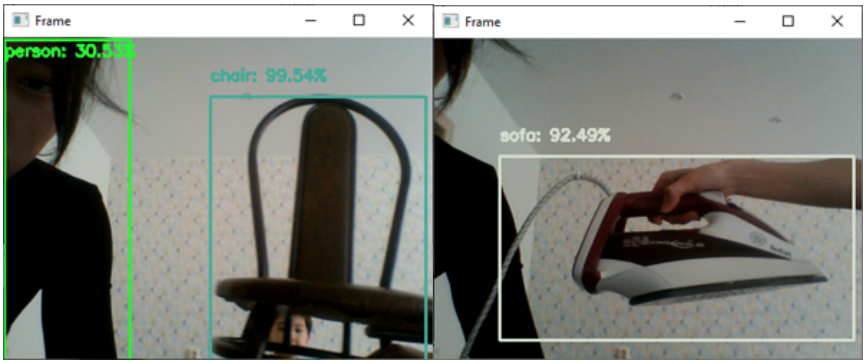


Рис.5. Запуск программы через командную строку.

Получаем следующий результат:



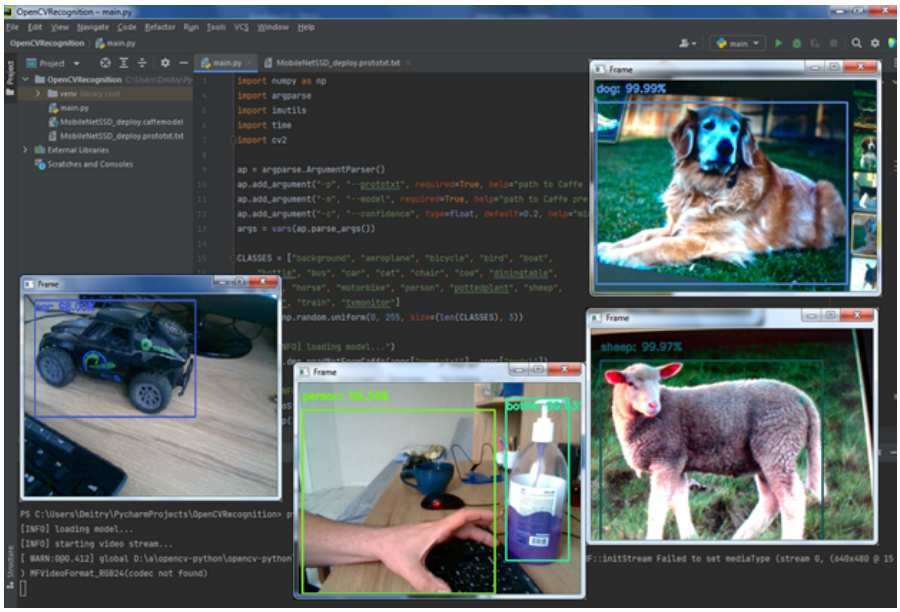


Рисунок 6. Работа программы

Код программы

Листинг 1. Main.py

```
from imutils.video import VideoStream

from imutils.video import FPS

import numpy as np

import argparse

import imutils

import time

import cv2


ap = argparse.ArgumentParser()

ap.add_argument("-p", "--prototxt", required=True, help="path to Caffe 'deploy' prototxt file")

ap.add_argument("-m", "--model", required=True, help="path to Caffe pre-trained model")

ap.add_argument("-c", "--confidence", type=float, default=0.2, help="minimum probability to filter weak detections")

args = vars(ap.parse_args())


CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",

    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",

    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",

    "sofa", "train", "tvmonitor"]

COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))


print("[INFO] loading model...")

net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

```
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
fps = FPS().start()

while True:
    frame = vs.read()
    frame = cv2.flip(frame, -1)  # Flip camera vertically
    frame = imutils.resize(frame, width=400)

    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
        0.007843, (300, 300), 127.5)

    net.setInput(blob)
    detections = net.forward()

    for i in np.arange(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]

        if confidence > args["confidence"]:
            idx = int(detections[0, 0, i, 1])
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            label = "{}: {:.2f}%".format(CLASSES[idx],
                confidence * 100)
            cv2.rectangle(frame, (startX, startY), (endX, endY),
                COLORS[idx], 2)
            y = startY - 15 if startY - 15 > 15 else startY + 15
            cv2.putText(frame, label, (startX, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break
    fps.update()
fps.stop()
```

```
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))

print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))


cv2.destroyAllWindows()

vs.stop()
```

Последнее изменение: Суббота, 14 мая 2022, 22:05

◄ 4.2. Видеолекция: Фреймворк TensorFlow для обнаружения и локализации объектов на основе нейронных сетей

Перейти на...

4.4. Практическое занятие 2. Разработка приложения для обнаружения и локализации объектов на изображениях и в видеопотоке ►