**ПРИЛОЖЕНИЕ А**

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

# КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ТРЕКИНГ ВЫБРАННЫХ ОБЪЕКТОВ В ВИДЕОПОТОКЕ

**ТЕКСТ ПРОГРАММЫ**

КП.ПО7.200149 – 04 12 00

Листов 7

|  |  |
| --- | --- |
| Руководитель | А.А. Крощенко |
| Выполнил | А.Е. Комиссаров |
| Консультант |  |
| по ЕСПД | А.А. Крощенко |
|  |  |

2022

**СОДЕРЖАНИЕ**

**main.py –** код программы, содержащий в себе запуск натренированной модели и установку свойств графического интерфейса.

**main.py**

import cv2

import sys

from PyQt5 import QtTest

from PyQt5.QtWidgets import \*

from PyQt5.QtCore import QThread, Qt, pyqtSignal, pyqtSlot, QTimer

from PyQt5.QtGui import QImage, QPixmap

import os

import numpy as np

import tensorflow as tf

from object\_detection.utils import label\_map\_util

from object\_detection.utils import config\_util

from object\_detection.utils import visualization\_utils as viz\_utils

from object\_detection.utils.visualization\_utils import STANDARD\_COLORS

from object\_detection.builders import model\_builder

from PIL import ImageColor

MODELS\_DIR = os.path.join(os.path.join(os.getcwd(), 'data'), 'models')

MODEL\_NAME = 'efficientdet'

PATH\_TO\_CKPT = os.path.join(MODELS\_DIR, os.path.join(MODEL\_NAME, 'checkpoint/'))

PATH\_TO\_CFG = os.path.join(MODELS\_DIR, os.path.join(MODEL\_NAME, 'pipeline.config'))

PATH\_TO\_LABELS = os.path.join(MODELS\_DIR, os.path.join(MODEL\_NAME, 'label\_map.pbtxt'))

TRACKED\_NAME = 'person'

LAST\_TRACKED\_NAME = 'person'

class Thread(QThread):

changePixmap = pyqtSignal(QImage)

not\_found\_signal = pyqtSignal(int)

loaded = pyqtSignal()

source = 0

stopvar = 0

def stop(self):

self.stopvar = 1

def run(self):

gpus = tf.config.experimental.list\_physical\_devices('GPU')

for gpu in gpus:

tf.config.experimental.set\_memory\_growth(gpu, True)

configs = config\_util.get\_configs\_from\_pipeline\_file(PATH\_TO\_CFG)

model\_config = configs['model']

detection\_model = model\_builder.build(model\_config=model\_config, is\_training=False)

ckpt = tf.compat.v2.train.Checkpoint(model=detection\_model)

ckpt.restore(os.path.join(PATH\_TO\_CKPT, 'ckpt-0')).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, prediction\_dict, tf.reshape(shapes, [-1])

category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)

cap = cv2.VideoCapture(0)

detect\_history = []

class Overlay():

lines = []

def \_\_init\_\_(self, x1, y1, x2, y2, color):

self.x1 = x1

self.y1 = y1

self.x2 = x2

self.y2 = y2

self.color = color

def addLine(self):

Overlay.lines.append([self.x1, self.y1, self.x2, self.y2, self.color])

def saveDetect(name, curX, curY, color):

global TRACKED\_NAME

global LAST\_TRACKED\_NAME

if TRACKED\_NAME != LAST\_TRACKED\_NAME:

Overlay.lines.clear()

LAST\_TRACKED\_NAME = TRACKED\_NAME

return

if name == LAST\_TRACKED\_NAME:

if len(detect\_history) == 0:

detect\_history.append([name, curX, curY, color])

else:

buf = 0

for i in range(len(detect\_history)):

if detect\_history[i][0] == name:

prevX = detect\_history[i][1]

prevY = detect\_history[i][2]

color = detect\_history[i][3]

Overlay.addLine(Overlay(prevY, prevX, curY, curX, color))

detect\_history[i][1] = curX

detect\_history[i][2] = curY

buf = i

if buf == (len(detect\_history)-1):

detect\_history.append([name, curX, curY, color])

cap = cv2.VideoCapture(self.source)

self.loaded.emit()

not\_found = 0

while True:

ret, image\_np = cap.read()

image\_np = cv2.resize(image\_np, (640,480))

input\_tensor = tf.convert\_to\_tensor(value=np.expand\_dims(image\_np, 0), dtype=tf.float32)

detections, predictions\_dict, shapes = detect\_fn(input\_tensor)

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'][0].numpy(), (detections['detection\_classes'][0].numpy() + label\_id\_offset).astype(int),

detections['detection\_scores'][0].numpy(), category\_index, use\_normalized\_coordinates=True,

max\_boxes\_to\_draw=10, min\_score\_thresh=.6, agnostic\_mode=False)

is\_found = False

for i in range(6):

if (detections['detection\_scores'][0][i].numpy() > 0.6):

#print("name: ", category\_index.get((detections['detection\_classes'][0][i].numpy() + label\_id\_offset).astype(int)).get('name'))

yCur = (detections['detection\_boxes'][0][i][0].numpy() + detections['detection\_boxes'][0][i][2].numpy()) / 2

xCur = (detections['detection\_boxes'][0][i][1].numpy() + detections['detection\_boxes'][0][i][3].numpy()) / 2

box\_color = ImageColor.getrgb(STANDARD\_COLORS[((detections['detection\_classes'][0].numpy() + label\_id\_offset).astype(int)[i])% len(STANDARD\_COLORS)])

box\_title = category\_index.get((detections['detection\_classes'][0][i].numpy() + label\_id\_offset).astype(int)).get('name')

if box\_title == TRACKED\_NAME:

is\_found = True

saveDetect(box\_title, round(yCur\*480), round(xCur\*640), box\_color)

if not is\_found:

not\_found +=1

print("Selected object not found, refreshing... ", not\_found)

else:

not\_found = 0

if not\_found > 5:

Overlay.lines.clear()

self.not\_found\_signal.emit(1)

else:

self.not\_found\_signal.emit(0)

for line in Overlay.lines:

cv2.line(image\_np\_with\_detections, (line[0],line[1]), (line[2],line[3]), line[4], 3)

cv2.putText(image\_np\_with\_detections , 'KAMISARAU, PO-7, 2022', (20,460), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255,255,255), 1, 2)

if cv2.waitKey(25) & 0xFF == ord('q'):

break

if self.stopvar == 1:

break

if ret:

# https://stackoverflow.com/a/55468544/6622587

rgbImage = cv2.cvtColor(image\_np\_with\_detections, cv2.COLOR\_BGR2RGB)

h, w, ch = rgbImage.shape

bytesPerLine = ch \* w

convertToQtFormat = QImage(rgbImage.data, w, h, bytesPerLine, QImage.Format\_RGB888)

p = convertToQtFormat.scaled(640, 480, Qt.KeepAspectRatio)

self.changePixmap.emit(p)

cap.release()

print("finished thread")

self.stopvar = 0

class Placeholder(QLabel):

def \_\_init\_\_(self, parent):

super(Placeholder,self).\_\_init\_\_(parent)

self.i = 0

self.timer=QTimer()

self.timer.setInterval(400)

self.timer.timeout.connect(self.animateText)

def stopTimer(self):

QtTest.QTest.qWait(40000)

self.timer.stop()

def animateText(self):

self.i += 1

self.i = self.i%4

self.setText("Loading" + ("."\*self.i))

class App(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title = 'PyQt5 Video'

self.left = 100

self.top = 100

self.width = 640

self.height = 500

self.initUI()

def notFoundText(self, msg):

if msg == 0:

self.labelNotFound.setText("")

else:

self.labelNotFound.setText("Selected object not found")

@pyqtSlot(QImage)

def setImage(self, image):

self.label.setPixmap(QPixmap.fromImage(image))

def initUI(self):

self.setWindowTitle(self.title)

self.setGeometry(self.left, self.top, self.width, self.height)

self.resize(850, 550)

self.labelNotFound = QLabel(self)

self.labelNotFound.setText("")

self.labelNotFound.setStyleSheet('''QLabel{text-align: center; font-size: 10px; color: red;}''')

self.labelNotFound.setGeometry(360,520,120,16)

self.labelExit = QLabel(self)

self.labelExit.setText("Press to exit")

self.labelExit.setGeometry(30,420,120,16)

self.buttonExit = QPushButton(self)

self.buttonExit.clicked.connect(self.close)

self.buttonExit.setGeometry(30,440,140,50)

self.buttonExit.setText("Exit")

self.buttonExit.show()

self.labelSelect = QLabel(self)

self.labelSelect.setText("Select item to track")

self.labelSelect.setGeometry(30,320,120,16)

self.selection = QComboBox(self)

self.selection.setGeometry(30,340,140,50)

for item in self.listOptions():

self.selection.addItem(item)

self.selection.activated[str].connect(self.onChanged)

self.labelCamera = QLabel(self)

self.labelCamera.setText("Press to load camera feed")

self.labelCamera.setGeometry(30,220,140,16)

self.buttonCamera = QPushButton(self)

self.buttonCamera.clicked.connect(self.openCamera)

self.buttonCamera.setGeometry(30,240,140,50)

self.buttonCamera.setText("Webcam")

self.buttonCamera.show()

self.labelFile = QLabel(self)

self.labelFile.setText("Press to open a video file")

self.labelFile.setGeometry(30,120,120,16)

self.buttonFile = QPushButton(self)

self.buttonFile.clicked.connect(self.openFile)

self.buttonFile.setGeometry(30,140,140,50)

self.buttonFile.setText("Video file")

self.buttonFile.show()

self.labelStop = QLabel(self)

self.labelStop.setText("Press to stop everything")

self.labelStop.setGeometry(30,10,140,16)

self.buttonStop = QPushButton(self)

self.buttonStop.clicked.connect(self.stopAll)

self.buttonStop.setGeometry(30,30,140,50)

self.buttonStop.setText("Stop")

self.buttonStop.show()

self.placeholder = Placeholder(self)

self.placeholder.setText("Waiting for command")

self.placeholder.setGeometry(191,11,638,478)

self.placeholder.setStyleSheet('''QWidget{text-align: center; font-size: 30px; border: 1px solid black;}''')

self.placeholder.show()

self.label = QLabel(self)

self.label.setStyleSheet('''QWidget{border: 1px solid black;}''')

self.label.move(190, 10)

self.label.resize(640, 480)

self.show()

self.thC = Thread(self)

self.thC.source = 0

self.thC.loaded.connect(self.placeholder.stopTimer)

self.thC.changePixmap.connect(self.setImage)

self.thC.not\_found\_signal.connect(self.notFoundText)

self.thF = Thread(self)

self.thF.loaded.connect(self.placeholder.stopTimer)

self.thF.changePixmap.connect(self.setImage)

self.thF.not\_found\_signal.connect(self.notFoundText)

def openFileNameDialog(self):

options = QFileDialog.Options()

options |= QFileDialog.DontUseNativeDialog

fileName, \_ = QFileDialog.getOpenFileName(self,"QFileDialog.getOpenFileName()", "","Video files (\*.mp4)", options=options)

if fileName:

return fileName

def openCamera(self):

self.label.show()

self.placeholder.timer.start()

self.thF.stop()

self.thC.stopvar = 0

self.thC.start()

def openFile(self):

file = self.openFileNameDialog()

self.thF.source = file

self.label.show()

self.placeholder.timer.start()

self.thC.stop()

self.thF.stopvar = 0

self.thF.start()

def stopAll(self):

self.placeholder.setText("Waiting for command")

self.label.hide()

self.thC.stop()

self.thF.stop()

def onChanged(self, text):

global TRACKED\_NAME

TRACKED\_NAME = text

def listOptions(self):

li = []

category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)

for a in category\_index:

li.append(category\_index.get(a).get('name'))

return li

app = QApplication(sys.argv)

application = App()

app.exec\_()