Общество с ограниченной ответственностью
«НЕЙРОТЕХ»
Разработки на базе искусственного интеллекта
614077, г. Пермь, бул. Гагарина, д. 58В, оф. 25
neurotech.pro · 1@neurotech.pro · +7 (902) 642-8658
ИНН 5906172713 / КПП 590601001 / ОГРН 1225900006155

ПРАВООБЛАДАТЕЛЬ:

Общество с ограниченной ответственностью

«Нейротех»

ПРОГРАММА ДЛЯ ЭВМ:

«Анти-Дрон»

АВТОРЫ:

1. Сивков Сергей Павлович

2. Шалимов Антон Владимирович

3. Шалимова Мария Анатольевна

4. Терехова Мария Александровна

5. Шалимов Александр Антонович

6. Шалимов Борис Антонович

Исходный текст программы на 14 листах.

г. Пермь.

Код определяет функцию, которая принимает две строки в качестве аргументов: model и zoo. Функция используется для генерации строки аргументов, которые будут использоваться в других частях программы для настройки параметров обработки видео и звука, а также параметров модели нейронной сети.

def gen_preproc_arguments(model, zoo):

```
return (

    gen_argument("delay", "Frame to frame delay for video files
loading in microseconds.", model, zoo) +

    gen_argument("screenX", "default screen layout width to preserve
original video aspect ratio.", model, zoo) +

    gen_argument("screenY", "default screen layount height to preserve
original video aspect ratio.", model, zoo) +

    gen_argument("width", "Preprocess input image by resizing to a
specific width.", model, zoo, ' ', "-1") +

    gen_argument("height", "Preprocess input image by resizing to a
```

```
specific height.", model, zoo, ' ', "-1") +

    gen_argument("sound", "Path to a sound file for alarm sound (.wav,
44100Hz, mono). "

                 "It should be a WAV file, 8 bit, 44100Hz, mono, with
play length at least 8 seconds.", model, zoo) +

    gen_argument("alarm-path", "Path to an alarm history.", model,
zoo) +

    gen_argument("opt-net", "Indicate that model has optimized NN
kernels definition file.", model, zoo, ' ', "fp16-4.rt") +

    gen_argument("opt-names", "Class definitions file.", model, zoo, '
', "obj.names") +

    gen_argument("opt-classes", "Indicate that model works with subset
of available classes.", model, zoo, ' ', "1") +

    gen_argument("opt-batch", "Batch size for optimized model.",
model, zoo, ' ', "4")

)
```

Код создания GUI-приложения с использованием библиотеки Tkinter. Он создает окно с видео, обрабатывает и показывает кадры видео с помощью класса Alerts.

import cv2

import tkinter as tk

from PIL import Image, ImageTk

import threading

VIDEO_WIDTH = 300

VIDEO_HEIGHT = 200

class Alerts:

```
def __init__(self, window):

    self.window = window

    self.started = False

    self.path_ = ""
```

```
    self.descr_ = ""

    self.frame = None

    self.canvas = tk.Canvas(window, width=VIDEO_WIDTH,
height=VIDEO_HEIGHT)

    self.canvas.pack()

    self.update_video()

def add_frame(self, src, cam, ts, path):

    sz = (VIDEO_WIDTH, VIDEO_HEIGHT)

    dst = cv2.resize(src, sz)

    self.frame = dst

    # Здесь должна быть функция обновления пользовательского
интерфейса, аналогичная tk_pwin->call_function

    self.window.refresh(cam, ts, path)

def update_alarm_history(self, frame, cam, ts, path):

    # Реализуйте функцию update_alarm_history согласно требованиям

    pass

def process_frames(self, src, bitmask):

    i = 0

    ts = ""

    path = ""

    for v in src:

        if bitmask & (1 << i):

            self.update_alarm_history(v, i + 1, ts, path)

            self.add_frame(v, i + 1, ts, path)
```

```
        i += 1
def update_video(self):
    if self.started:
        self.show_frame(self.frame)
    self.window.after(20, self.update_video)
def show_frame(self, frame):
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = Image.fromarray(frame)
    frame = ImageTk.PhotoImage(frame)
    self.canvas.create_image(0, 0, anchor=tk.NW, image=frame)
    self.canvas.image = frame
def last(self, path, descr):
    self.path_ = path
    self.descr_ = descr
    if not self.started:
        self.started = True
def get_path(self):
    return self.path_
def get_descr(self):
    return self.descr_
class VideoApp(tk.Tk):
def __init__(self):
    super().__init__()
    self.title("Video App")
```

```
    self.alerts = Alerts(self)

def refresh(self, cam, ts, path):

    # Реализуйте функцию обновления пользовательского интерфейса,

    pass
```

if **name** == “**main**”:

```
app = VideoApp()

app.mainloop()
```

Код создает приложение с двумя вкладками, например: “History View” и “Last Alerts View”. На вкладке “History View” есть таблица с заголовками “index”, “tstamp” и “cam”. На вкладке “Last Alerts View” есть 4 контейнера, представляющие видеопотоки от 4 видеокамер, и текстовый виджет для вывода информации.

import tkinter as tk

from tkinter import ttk

class VideoApp(tk.Tk):

```
def __init__(self):

    super().__init__()

    self.title("Video App")

    # Создание вкладок

    self.tab_control = ttk.Notebook(self)

    # Вкладка "History View"

    self.history_tab = ttk.Frame(self.tab_control)

    self.tab_control.add(self.history_tab, text="History View")

    self.create_history_view(self.history_tab)

    # Вкладка "Last Alerts View"

    self.alerts_tab = ttk.Frame(self.tab_control)
```

```
    self.tab_control.add(self.alerts_tab, text="Last Alerts View")
    self.create_last_alerts_view(self.alerts_tab)
    self.tab_control.pack(expand=1, fill='both')
def create_history_view(self, tab):
    # Создание виджета таблицы
    columns = ("index", "tstamp", "cam")
    self.tree = ttk.Treeview(tab, columns=columns, show="headings")
    for col in columns:
        self.tree.heading(col, text=col)
        self.tree.column(col, width=100)
    self.tree.pack(side='left', fill='both', expand=True)
def create_last_alerts_view(self, tab):
    # Создание виджетов для отображения видеопотоков
    self.cam_frames = []
    for i in range(1, 5):
        cam_frame = tk.LabelFrame(tab, text=f"Cam #{i}", width=300,
height=200)
        cam_frame.pack(side="left", padx=5, pady=5)
        self.cam_frames.append(cam_frame)
    # Создание виджета вывода информации
    self.info_output = tk.Text(tab, wrap="word")
    self.info_output.pack(side="bottom", fill="both", expand=True)
```

if **name** == “**main**”:

```
app = VideoApp()

app.mainloop()
```

Код создает поток, который будет воспроизводить звуковой файл с изменяемой частотой дискретизации, в зависимости от количества распознанных объектов на камерах (переменная alarm_detections). Частота дискретизации выбирается из списка rates на основе текущей статистики распознавания.

import alsaaudio

import numpy as np

import time

from threading import Thread

from collections import deque

process = True

alarm_detections = 0

rates = [22050, 44100, 88200, 132300, 196400]

rates_size = len(rates)

sound_len_ms = 2000 # 2s

def get_rate():

```
global alarm_detections

detections = alarm_detections

alarm_detections = 0

frames = detections & 0xFFFF

detections >>= 16

if frames == 0 or detections == 0:

    return 0

rate = 100.0 * detections / frames / (100.0 / rates_size)

for i in range(rates_size):
```

```
    if rate <= i:
        return rates[i]
return rates[rates_size - 1]
```

def alarm_sound(buff):

```
pcm = alsaaudio.PCM(type=alsaaudio.PCM_PLAYBACK,
mode=alsaaudio.PCM_NORMAL, device="default")
pcm.setformat(alsaaudio.PCM_FORMAT_S16_LE)
channels = 1
pcm.setchannels(channels)
started = False
while process:
    rate = get_rate()
    if rate == 0:
        time.sleep(sound_len_ms / 1000)
        continue
    if not started:
        pcm.setrate(rate)
        started = True
    pcm.write(buff)
```

if **name** == “**main**”:

```
# Загрузите звуковой файл в буфер, используйте numpy
# buff = ...
# Запускаем поток для воспроизведения звука
alarm_thread = Thread(target=alarm_sound, args=(buff,))
alarm_thread.start()
```

```
# Ожидаем завершения потока (например, по нажатию клавиши или другому событию)

alarm_thread.join()
```

Код коммуникации эффекторов нейросети с пользовательским интерфейсом:

import cv2

import numpy as np

import time

import threading

from queue import Queue

from collections import deque

import datetime

import pygtk

pygtk.require('2.0')

import gtk

Определение класса QueueFPS

class QueueFPS(Queue):

```
def __init__(self):

    Queue.__init__(self)

    self.counter = 0

    self.fps = 0

    self.start_time = time.time()

def put(self, item):

    Queue.put(self, item)

    self.counter += 1
```

```
    self.fps = self.counter / (time.time() - self.start_time)
def getFPS(self):
    return self.fps
```

def callback(pos):

```
global confThreshold
confThreshold = pos * 0.01
```

def update_alarm_history(frame, cam, ts, path):

```
global alarm_path, alarmQueue
now = datetime.datetime.now()
milliseconds = now.microsecond // 1000
ts_buf = now.strftime("%Y-%m-%d-%H-%M-%S-") + f"{milliseconds:03d}"
buf = f"{alarm_path}/{ts_buf}-{cam}.png"
ts = ts_buf
path = buf
alarmQueue.put((path, frame))
```

def alarm_history():

```
global process, alarmQueue
while process:
    if not alarmQueue.empty():
        src = alarmQueue.get()
        cv2.imwrite(src[0], src[1])
    else:
        time.sleep(0.001)
print("alarm_history: stopped")
```

def make_capture_thread(cap, framesQueue, tag, delay):

```
readed = 0

while process:

    ret, frame = cap.read()

    if ret:

        readed += 1

        framesQueue.put(frame.copy())

        if tag == "i":

            time.sleep(delay / 1000000.0)

    else:

        print(f"make_capture_thread: unable to cap.read on {tag} after
{readed}")

        process = False

        break

print(f"make_capture_thread: {tag} stopped")
```

Определение остальных функций

...

def initGTK():

```
gtk.gdk.threads_init()

gtk.gdk.threads_enter()

gtk.main()

gtk.gdk.threads_leave()
```

def metakom_ui(needs_debugger, show, record):

```
if show:

    initGTK()
```

if **name** == “**main**”:

```
metakom_ui(False, True, False)
```

Код отвечает за инициализацию пайплайна обработки на основе параметров backend, cap2, cap3 и cap4, создает и запускает различные обработчики для комбинации камер и опций, создает окно отображения результатов обнаружения объектов, выполняет цикл обработки кадров, в котором извлекаются обнаружения из очереди predictions_queue, подготавливаются к отображению и выводятся на экран. Кадры также записываются в видеофайл.

import cv2

import threading

import time

from queue import Queue

функции и классы обработки, например:

tk_processing1_thread, tk_processing2_thread, tk_processing3_thread, tk_processing4_thread

cv_processing_thread, make_hjoin_thread, make_vjoin_thread, make_join3_thread, make_join4_thread, make_fakejoin_thread

Инициализация пайплайна обработки

if backend == METAKOM_OPT_MODEL:

```
opt_net_path = parser["opt-net"] if "opt-net" in parser else ""

name_path = parser["opt-names"] if "opt-names" in parser else ""

opt_classes = parser["opt-classes"]

opt_batch = parser["opt-batch"]
```

```
if opt_batch < 1 or opt_batch > 16:  # 64 - это наша собственная
теоретическая верхняя граница

    print(f"Incorrect batch dimension: {opt_batch}")

    exit(1)

assert opt_net_path != "" and name_path != ""

assert not cap2 or (cap2 and not cap3 and opt_batch >= 2) or (cap2 and
cap3 and not cap4 and opt_batch >= 2) or (cap2 and cap3 and cap4 and
opt_batch >= 4)

if not cap2:

    joiner = threading.Thread(target=tk_processing1_thread,
args=(opt_net_path, config_path, name_path, opt_classes,
frames_queue1, sz))

elif cap2 and not cap3:

    joiner = threading.Thread(target=tk_processing2_thread,
args=(opt_net_path, config_path, name_path, opt_classes,
frames_queue1, frames_queue2, sz, out_sz))

elif cap2 and cap3 and not cap4:

    joiner = threading.Thread(target=tk_processing3_thread,
args=(opt_net_path, config_path, name_path, opt_classes,
frames_queue1, frames_queue2, frames_queue3, sz, out_sz))

else:

    joiner = threading.Thread(target=tk_processing4_thread,
args=(opt_net_path, config_path, name_path, opt_classes,
frames_queue1, frames_queue2, frames_queue3, frames_queue4, sz,
out_sz))

generation_thread = tk_generation_thread

metakom_ui(ac, av, debug, show_sc, video and video.isOpened())
```

else:

```
if cap2 and not cap3:

    joiner = threading.Thread(target=make_hjoin_thread,
```

Общество с ограниченной ответственностью «НЕЙРОТЕХ»
Разработки на базе искусственного интеллекта
614077, г. Пермь, бул. Гагарина, д. 58В, оф. 25
neurotech.pro · 1@neurotech.pro · +7 (902) 642-8658
ИНН 5906172713 / КПП 590601001 / ОГРН 1225900006155

```
args=(frames_queue1, frames_queue2, sz, out_sz))

elif not cap2 and cap3:

    joiner = threading.Thread(target=make_vjoin_thread,
args=(frames_queue1, frames_queue3, sz, out_sz))

elif cap2 and cap3 and not cap4:

    joiner = threading.Thread(target=make_join3_thread,
args=(frames_queue1, frames_queue2, frames_queue3, sz, out_sz))

elif cap2 and cap3

and cap4:

    joiner = threading.Thread(target=make_join4_thread,
args=(frames_queue1, frames_queue2, frames_queue3, frames_queue4, sz,
out_sz))

else:

    joiner = threading.Thread(target=make_fakejoin_thread,
args=(frames_queue1, out_sz))

processing_thread = threading.Thread(target=cv_processing_thread,
args=(config_path, model_path, backend, target, (inp_width,
inp_height)))

if show_sc:

   generation_thread = cv_generation_thread

    metakom_ui(ac, av, debug, show_sc, video and video.isOpened())

    print("metakom_ui done")

else:

    win_name = "Shortgun/Rifle object detection demonstration"

    if show_cv:

        cv2.namedWindow(win_name, cv2.WINDOW_AUTOSIZE)

        initial_conf = int(conf_threshold * 100)
```

```
        cv2.createTrackbar("Confidence threshold, %", win_name,
initial_conf, 99, callback)

    while process:

        if predictions_queue.empty():

            time.sleep(0.001)

            continue

        detections = predictions_queue.get()

        frame = processed_frames_queue.get()

        cv_render_prepare(detections, frame)

        if show_cv:

            cv2.imshow(win_name, frame)

        if video:

            video.write(frame)

        frame_idx += 1

        if cv2.waitKey(1) & 0xFF == ord('q'):

            break

    process = False

    processing_thread.join()
```