Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
По дисциплине: «Обработка изображений в интеллектуальных системах»
Тема: “Трекинг множественных объектов”

Выполнил:
Студент 4 курса
Группы ИИ-24
Крупич Д.Д.
Проверила:
Андренко К. В.

Брест 2025

Цель: исследовать применение алгоритмов трекинга на базе обученной сети-детектора объектов.

Общее задание

1. Используя сеть-детектор, обученный в ЛР 3, реализовать логику для отслеживания множественных объектов, используя библиотеку Ultralytics YOLO;
2. Применять алгоритмы BoT-Sort и ByteTrack (задействовать соответствующие конфигурационные файлы);
3. Исследовать изменения параметров в конфигурационных файлах и их влияние на качество трекинга;
4. В качестве исходных видеоматериалов для экспериментов использовать видео-ролики из сети (например, из YouTube), содержащие множественные объекты классов из ЛР 3;
5. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github.

Ход работы:

Код программы:

```
"""
Лабораторная работа №4: Трекинг множественных объектов
Использование алгоритмов BoT-SORT и ByteTrack с YOLO детектором
"""

import tkinter as tk
from tkinter import filedialog, Label, Button, Frame, ttk, messagebox
from PIL import Image, ImageTk
from ultralytics import YOLO
import cv2
import os
import time
import queue

class TrackerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("ЛР4: Трекинг множественных объектов")
        self.root.geometry("1200x800")
        self.root.configure(bg="#2b2b2b")
```

```
        # Загрузка модели
        self.model = YOLO("best.pt")

        # Переменные
        self.video_path = None
        self.is_tracking = False
        self.is_paused = False
        self.cap = None
        self.tracker_var = tk.StringVar(value="botsort")
        self.current_tracker = "botsort"  # Для отслеживания смены алгоритма

        # Статистика
        self.unique_ids = set()
        self.frame_count = 0
        self.start_time = None
        self.total_frames = 0

        # Очередь для передачи кадров между потоками
        self.frame_queue = queue.Queue(maxsize=2)

        self.setup_ui()

        # Запуск обновления UI
        self.update_display()

    def setup_ui(self):
        # Заголовок
        title = Label(
            self.root,
            text="Трекинг множественных объектов",
            font=("Arial", 20, "bold"),
            bg="#2b2b2b", fg="#ffffff"
        )
        title.pack(pady=15)

        # Панель управления
        control_frame = Frame(self.root, bg="#3c3c3c", padx=20, pady=15)
        control_frame.pack(fill="x", padx=20)

        # Кнопка загрузки видео
        self.btn_load = Button(
            control_frame,
            text="Загрузить видео",
            command=self.load_video,
            font=("Arial", 12),
            bg="#4a9eff", fg="white",
            padx=15, pady=8
        )
        self.btn_load.pack(side="left", padx=10)
```

```
# Выбор трекера
tracker_label = Label(
    control_frame,
    text="Алгоритм:",
    font=("Arial", 12),
    bg="#3c3c3c", fg="white"
)
tracker_label.pack(side="left", padx=(20, 5))

self.tracker_combo = ttk.Combobox(
    control_frame,
    textvariable=self.tracker_var,
    values=["botsort", "bytetrack"],
    state="readonly",
    width=12,
    font=("Arial", 11)
)
self.tracker_combo.pack(side="left", padx=5)
self.tracker_combo.bind("<<ComboboxSelected>>", self.on_tracker_changed)

# Кнопка запуска трекинга
self.btn_track = Button(
    control_frame,
    text="Запустить трекинг",
    command=self.start_tracking,
    font=("Arial", 12),
    bg="#4caf50", fg="white",
    padx=15, pady=8,
    state="disabled"
)
self.btn_track.pack(side="left", padx=10)

# Кнопка паузы
self.btn_pause = Button(
    control_frame,
    text="Пауза",
    command=self.toggle_pause,
    font=("Arial", 12),
    bg="#ff9800", fg="white",
    padx=15, pady=8,
    state="disabled"
)
self.btn_pause.pack(side="left", padx=10)

# Кнопка остановки
self.btn_stop = Button(
    control_frame,
    text="Стоп",
    command=self.stop_tracking,
    font=("Arial", 12),
```

```
    bg="#f44336", fg="white",
    padx=15, pady=8,
    state="disabled"
)
self.btn_stop.pack(side="left", padx=10)

# Кнопка сохранения
self.btn_save = Button(
    control_frame,
    text="Сохранить видео",
    command=self.save_video,
    font=("Arial", 12),
    bg="#9c27b0", fg="white",
    padx=15, pady=8,
    state="disabled"
)
self.btn_save.pack(side="left", padx=10)

# Область отображения видео
self.video_frame = Frame(self.root, bg="#1e1e1e")
self.video_frame.pack(expand=True, fill="both", padx=20, pady=15)

self.video_label = Label(
    self.video_frame,
    text="Загрузите видео для начала работы",
    font=("Arial", 16),
    bg="#1e1e1e", fg="#888888"
)
self.video_label.pack(expand=True)

# Панель статистики
stats_frame = Frame(self.root, bg="#3c3c3c", padx=20, pady=10)
stats_frame.pack(fill="x", padx=20, pady=(0, 15))

self.lbl_video_info = Label(
    stats_frame,
    text="Видео: не загружено",
    font=("Arial", 11),
    bg="#3c3c3c", fg="#aaaaaa"
)
self.lbl_video_info.pack(side="left", padx=20)

self.lbl_tracker_info = Label(
    stats_frame,
    text="Трекер: -",
    font=("Arial", 11),
    bg="#3c3c3c", fg="#aaaaaa"
)
self.lbl_tracker_info.pack(side="left", padx=20)
```

```
        self.lbl_ids = Label(
            stats_frame,
            text="Уникальных ID: 0",
            font=("Arial", 11, "bold"),
            bg="#3c3c3c", fg="#4caf50"
        )
        self.lbl_ids.pack(side="left", padx=20)

        self.lbl_fps = Label(
            stats_frame,
            text="FPS: -",
            font=("Arial", 11),
            bg="#3c3c3c", fg="#aaaaaa"
        )
        self.lbl_fps.pack(side="left", padx=20)

        self.lbl_frame = Label(
            stats_frame,
            text="Кадр: 0 / 0",
            font=("Arial", 11),
            bg="#3c3c3c", fg="#aaaaaa"
        )
        self.lbl_frame.pack(side="left", padx=20)

        self.lbl_status = Label(
            stats_frame,
            text="",
            font=("Arial", 11, "bold"),
            bg="#3c3c3c", fg="#ffeb3b"
        )
        self.lbl_status.pack(side="left", padx=20)

    def on_tracker_changed(self, event=None):
        """При смене алгоритма сбрасываем модель и ID"""
        new_tracker = self.tracker_var.get()
        if new_tracker != self.current_tracker:
            self.current_tracker = new_tracker
            # Перезагружаем модель для сброса трекера
            self.model = YOLO("best.pt")
            self.unique_ids = set()
            self.lbl_ids.config(text="Уникальных ID: 0")
            self.lbl_tracker_info.config(text=f"Трекер: {new_tracker.upper()} (сброшен)")

    def load_video(self):
        path = filedialog.askopenfilename(
            filetypes=[("Video files", "*.mp4 *.avi *.mov *.mkv *.webm")]
        )
        if not path:
            return
```

```
        self.video_path = path
        self.cap = cv2.VideoCapture(path)

        # Получаем информацию о видео
        self.total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
        fps = self.cap.get(cv2.CAP_PROP_FPS)
        width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

        self.lbl_video_info.config(
            text=f"Видео: {os.path.basename(path)} | {width}x{height} | {fps:.1f} FPS |
{self.total_frames} кадров"
        )
        self.lbl_frame.config(text=f"Кадр: 0 / {self.total_frames}")

        # Показываем первый кадр
        ret, frame = self.cap.read()
        if ret:
            self.display_frame(frame)

        self.cap.release()
        self.btn_track.config(state="normal")
        self.btn_save.config(state="normal")

    def display_frame(self, frame):
        """Отображение кадра в интерфейсе"""
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Масштабирование под размер окна
        max_width = self.video_frame.winfo_width() - 40
        max_height = self.video_frame.winfo_height() - 40

        if max_width > 100 and max_height > 100:
            h, w = frame_rgb.shape[:2]
            scale = min(max_width / w, max_height / h)
            if scale < 1:
                new_w, new_h = int(w * scale), int(h * scale)
                frame_rgb = cv2.resize(frame_rgb, (new_w, new_h))

        img = Image.fromarray(frame_rgb)
        self.photo = ImageTk.PhotoImage(img)
        self.video_label.config(image=self.photo, text="")

    def update_display(self):
        """Обновление дисплея из очереди (вызывается в главном потоке)"""
        try:
            while not self.frame_queue.empty():
                data = self.frame_queue.get_nowait()
                if data is not None:
                    frame, frame_num, unique_count = data
```

```
                self.display_frame(frame)

                elapsed = time.time() - self.start_time if self.start_time else 0
                fps = frame_num / elapsed if elapsed > 0 else 0

                self.lbl_ids.config(text=f"Уникальных ID: {unique_count}")
                self.lbl_fps.config(text=f"FPS: {fps:.1f}")
                self.lbl_frame.config(text=f"Кадр: {frame_num} / {self.total_frames}")
        except:
            pass

        # Планируем следующее обновление
        self.root.after(10, self.update_display)

    def start_tracking(self):
        if not self.video_path:
            return

        # Сброс модели для нового трекинга (ID начнутся с 1)
        self.model = YOLO("best.pt")

        self.is_tracking = True
        self.is_paused = False
        self.unique_ids = set()
        self.frame_count = 0
        self.start_time = time.time()
        self.current_tracker = self.tracker_var.get()

        self.btn_track.config(state="disabled")
        self.btn_pause.config(state="normal")
        self.btn_stop.config(state="normal")
        self.btn_load.config(state="disabled")
        self.tracker_combo.config(state="disabled")
        self.lbl_status.config(text="")

        tracker = self.tracker_var.get()
        self.lbl_tracker_info.config(text=f"Трекер: {tracker.upper()}")

        # Запуск в отдельном потоке
        import threading
        self.tracking_thread = threading.Thread(target=self.run_tracking, daemon=True)
        self.tracking_thread.start()

    def run_tracking(self):
        """Основной цикл трекинга"""
        tracker_config = f"{self.tracker_var.get()}.yaml"

        cap = cv2.VideoCapture(self.video_path)

        while self.is_tracking:
```

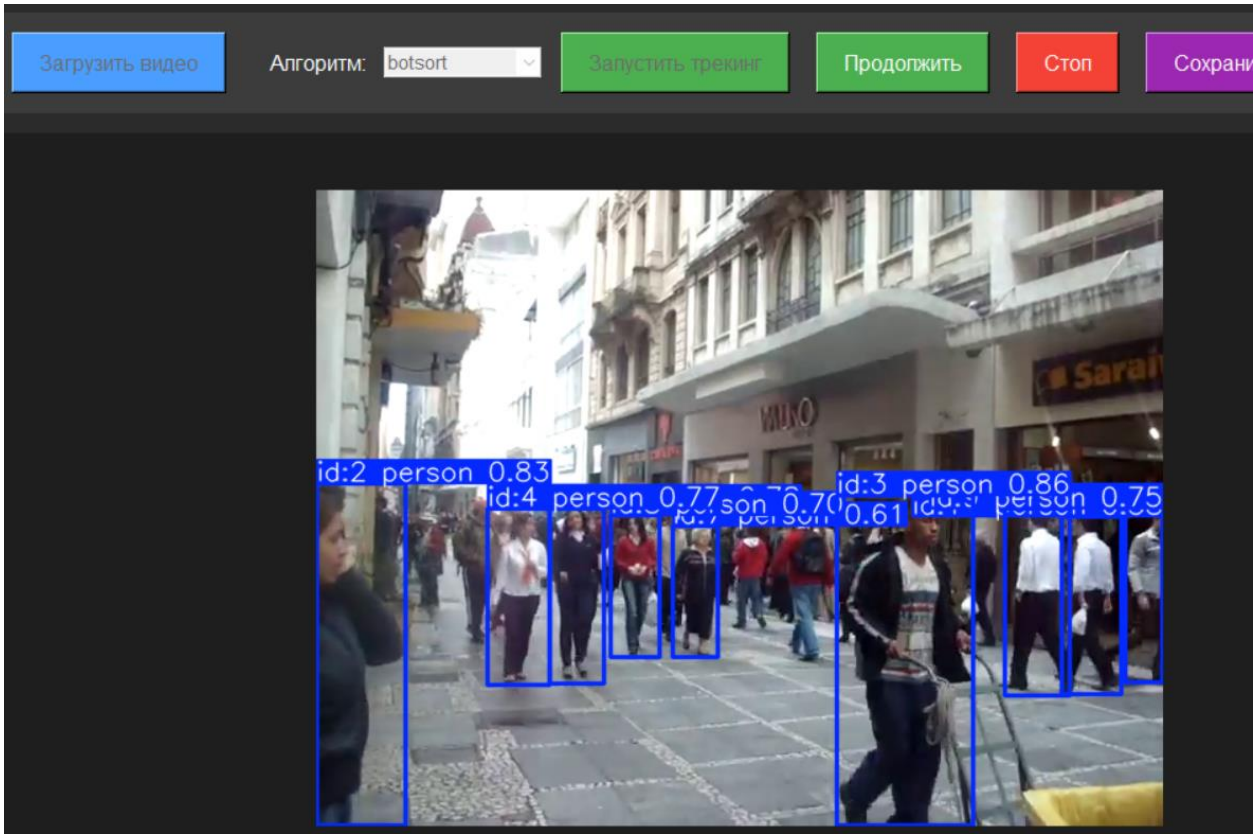
```
            # Пауза
            if self.is_paused:
                time.sleep(0.1)
                continue

            ret, frame = cap.read()
            if not ret:
                break

            self.frame_count += 1

            # Трекинг с выбранным алгоритмом
            results = self.model.track(
                frame,
                tracker=tracker_config,
                persist=True,
                conf=0.5,
                verbose=False
            )

            # Получаем ID объектов
            if results[0].boxes.id is not None:
                ids = results[0].boxes.id.cpu().numpy().astype(int)
                self.unique_ids.update(ids)

            # Отрисовка результата
            annotated_frame = results[0].plot()

            # Отправляем кадр в очередь для отображения
            try:
                if not self.frame_queue.full():
                    self.frame_queue.put_nowait((annotated_frame, self.frame_count, len(self.unique_ids)))
            except:
                pass

        cap.release()

        # Сигнал о завершении
        self.is_tracking = False
        self.root.after(0, self.tracking_finished)

    def toggle_pause(self):
        """Переключение паузы"""
        self.is_paused = not self.is_paused
        if self.is_paused:
            self.btn_pause.config(text="Продолжить", bg="#4caf50")
            self.lbl_status.config(text="ПАУЗА")
        else:
            self.btn_pause.config(text="Пауза", bg="#ff9800")
            self.lbl_status.config(text="")
```

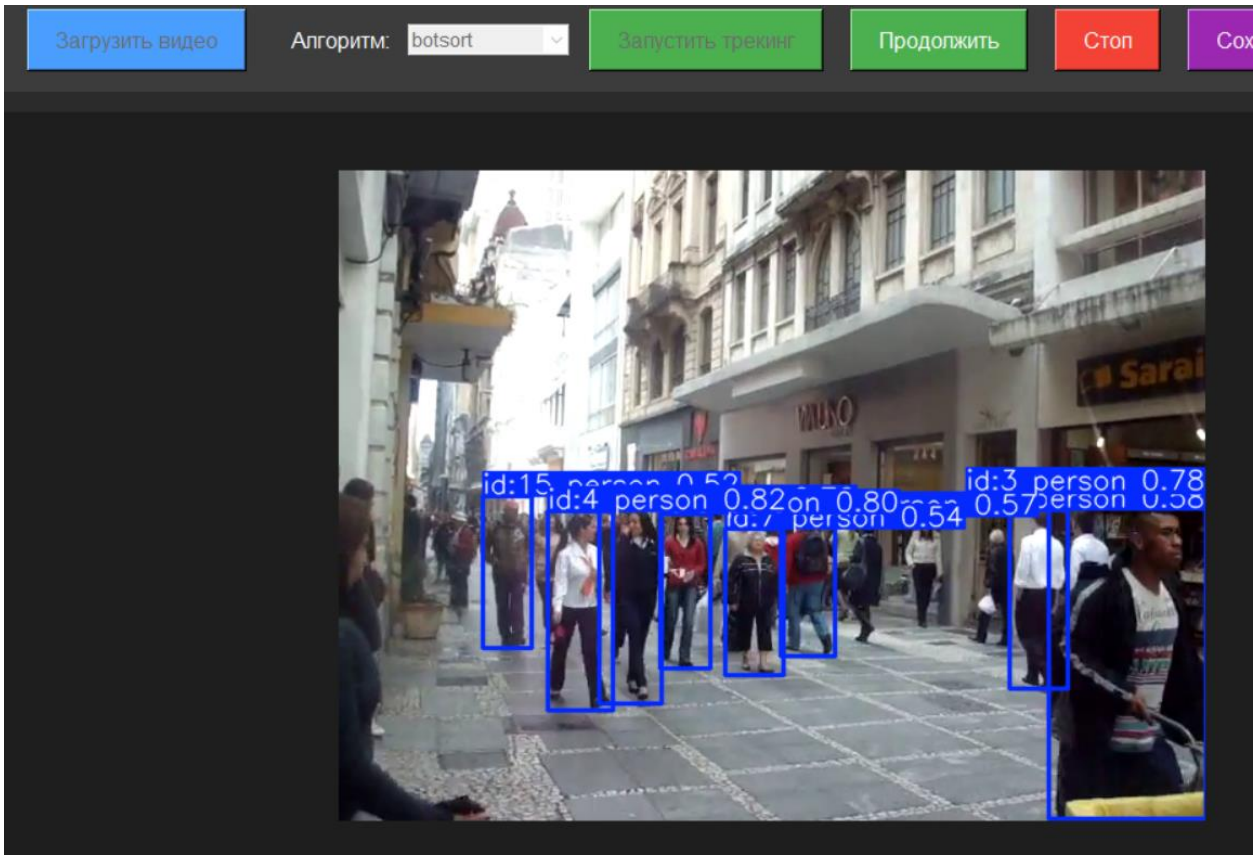
```
def tracking_finished(self):
    """Вызывается по завершении трекинга"""
    self.is_tracking = False
    self.is_paused = False
    self.btn_track.config(state="normal")
    self.btn_pause.config(state="disabled", text="Пауза", bg="#ff9800")
    self.btn_stop.config(state="disabled")
    self.btn_load.config(state="normal")
    self.tracker_combo.config(state="readonly")
    self.lbl_status.config(text="ЗАВЕРШЕНО")

    messagebox.showinfo(
        "Трекинг завершён",
        f"Обработано кадров: {self.frame_count}\n"
        f"Уникальных объектов: {len(self.unique_ids)}"
    )

def stop_tracking(self):
    self.is_tracking = False
    self.is_paused = False

def save_video(self):
    """Сохранение видео с трекингом"""
    if not self.video_path:
        return

    save_path = filedialog.asksaveasfilename(
        defaultextension=".mp4",
        filetypes=[("MP4", "*.mp4"), ("AVI", "*.avi")],
        initialfile=f"tracked_{self.tracker_var.get()}.mp4"
    )

    if not save_path:
        return

    tracker_config = f"{self.tracker_var.get()}.yaml"

    # Сброс модели для нового трекинга
    self.model = YOLO("best.pt")

    cap = cv2.VideoCapture(self.video_path)
    fps = cap.get(cv2.CAP_PROP_FPS)
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(save_path, fourcc, fps, (width, height))
```

```
        frame_num = 0
        self.unique_ids = set()

        self.lbl_status.config(text="СОХРАНЕНИЕ...")

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            frame_num += 1

            results = self.model.track(
                frame,
                tracker=tracker_config,
                persist=True,
                conf=0.5,
                verbose=False
            )

            if results[0].boxes.id is not None:
                ids = results[0].boxes.id.cpu().numpy().astype(int)
                self.unique_ids.update(ids)

            annotated = results[0].plot()
            out.write(annotated)

            self.lbl_frame.config(text=f"Сохранение: {frame_num} / {total_frames}")
            self.root.update()

        cap.release()
        out.release()

        self.lbl_ids.config(text=f"Уникальных ID: {len(self.unique_ids)}")
        self.lbl_status.config(text="СОХРАНЕНО")

        messagebox.showinfo(
            "Сохранено",
            f"Видео сохранено: {save_path}\n"
            f"Уникальных объектов обнаружено: {len(self.unique_ids)}"
        )

if __name__ == "__main__":
    root = tk.Tk()
    app = TrackerApp(root)
    root.mainloop()
```

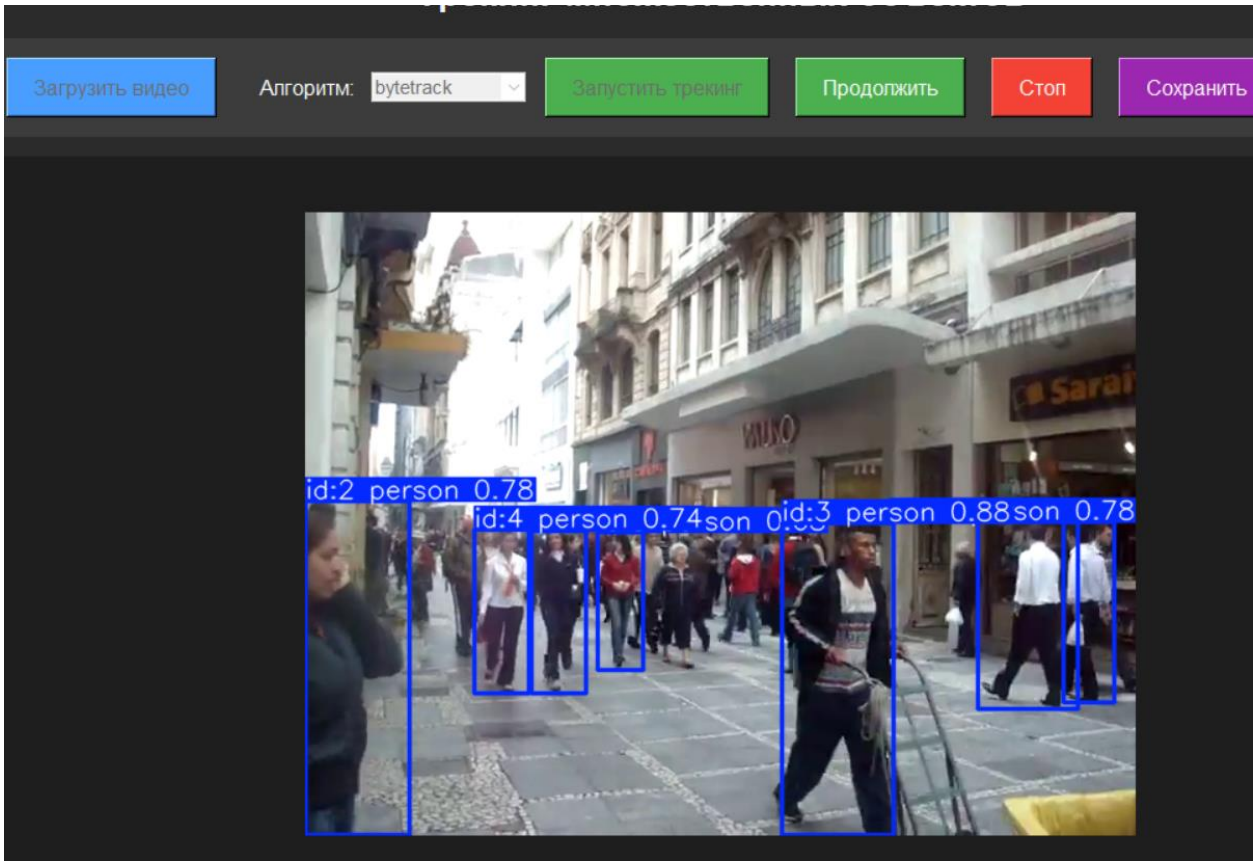
Результат:

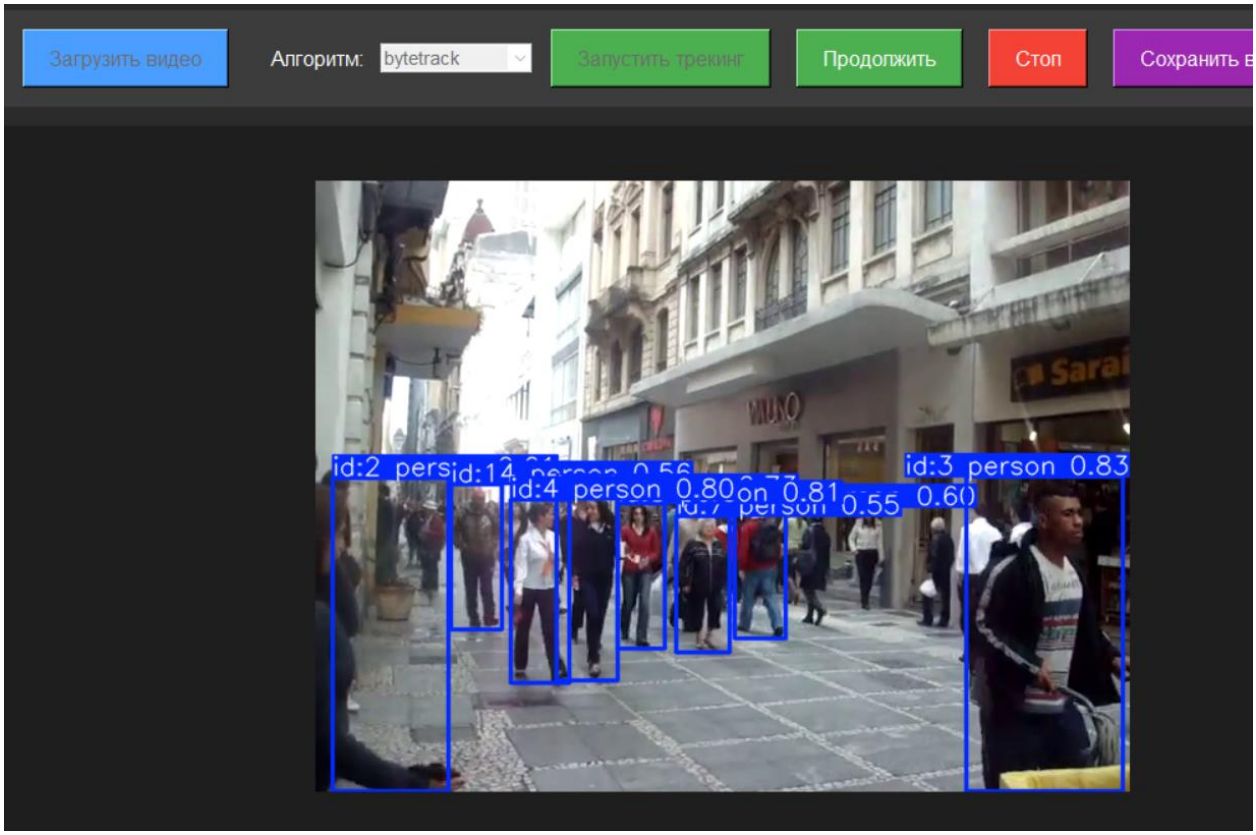
BoT-SORT:





ByteTrack:


Загрузить видео
Алгоритм:
bytetrack
Запустить трекинг
Продолжить
Стоп
Сохранить
id:2 person 0.78
id:4 person 0.74
id:3 person 0.88
son 0.78



Разница алгоритмов
BoT-SORT показал лучшую стабильность трекинга благодаря использованию GMC (компенсации движения камеры), но работает медленнее (3.8 FPS). ByteTrack быстрее (5.2 FPS), но чаще теряет объекты.

Вывод: исследовал применение алгоритмов трекинга на базе обученной сети-детектора объектов.