

Министерство науки и высшего образования Российской Федерации  
ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики  
Кафедра вычислительной техники и электроники (ВТиЭ)

Отчёт по производственной практике  
АВТОМАТИЗАЦИЯ РЕШЕНИЯ САРТСНА В ФОРМАТЕ ИЗОБРАЖЕНИЙ

Выполнил студент 5.306М группы:

\_\_\_\_\_ А. В. Лаптев

Проверил: доц. каф. ВТиЭ

\_\_\_\_\_ А. В. Калачев

«\_\_\_» \_\_\_\_\_ 2025 г.

Барнаул 2025 г.

## РЕФЕРАТ

Полный объём работы составляет 24 страницы, включая 6 рисунков и 1 листинг.

Данная работа посвящена разработке и обучению нейросетевой модели для автоматического распознавания объектов на изображениях CAPTCHA. Целью исследования является создание системы, способной интерпретировать задания CAPTCHA и выбирать нужные объекты на изображениях в автоматическом режиме. В рамках проекта был собран собственный датасет CAPTCHA-изображений, проведена их разметка, обучена модель YOLOv8 с поддержкой сегментации и реализован скрипт автоматического взаимодействия с web-интерфейсами. Результаты тестирования модели подтвердили её способность эффективно решать задания CAPTCHA, что делает разработку применимой в задачах автоматизации тестирования и обхода визуальных защит.

Ключевые слова: CAPTCHA, нейронная сеть, YOLOv8, сегментация, компьютерное зрение, Selenium, CVAT.

## СОДЕРЖАНИЕ

Введение . . . . .	4
1. Выбор модели нейронной сети для обучения . . . . .	5
2. Парсинг изображений CAPTCHA и их предобработка для создания датасета . . . . .	8
2.1. Парсинг реальных CAPTCHA с различных web-ресурсов . . . . .	8
2.2. Предварительная обработка изображений датасета . . . . .	9
3. Обучение и тестирование модели YOLOv8 на реальных CAPTCHA	12
Заключение . . . . .	16
Список использованной литературы . . . . .	18
Приложение . . . . .	19

## ВВЕДЕНИЕ

В последние годы CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) остаётся одним из наиболее распространённых способов защиты web-ресурсов от автоматических скриптов. Одной из наиболее сложных форм CAPTCHA являются изображения, содержащие множество объектов с размытыми контурами, шумами и низким разрешением, что затрудняет автоматическое распознавание.

Сложность таких изображений делает их трудными не только для ботов, но и для современных систем компьютерного зрения. При этом в процессе автоматизированного тестирования web-приложений возникает необходимость обхода подобных CAPTCHA, что требует разработки устойчивых и точных методов распознавания визуального контента.

Целью данной работы является разработка и обучение нейронной сети с поддержкой сегментации, способной автоматически распознавать объекты на изображениях CAPTCHA и выполнять задания, формируемые системой защиты.

Для достижения поставленной цели необходимо решить следующие задачи:

1. проанализировать типы CAPTCHA, применяемых на web-ресурсах;
2. выбрать подходящую архитектуру нейронной сети, обеспечивающую высокую скорость и точность;
3. собрать и разметить датасет реальных CAPTCHA с изображениями объектов;
4. провести предварительную обработку изображений и формирование структуры датасета;
5. обучить выбранную модель на собранных данных;
6. разработать скрипт для автоматизированного прохождения CAPTCHA с использованием обученной модели;
7. протестировать модель в реальных условиях и оценить её эффективность.

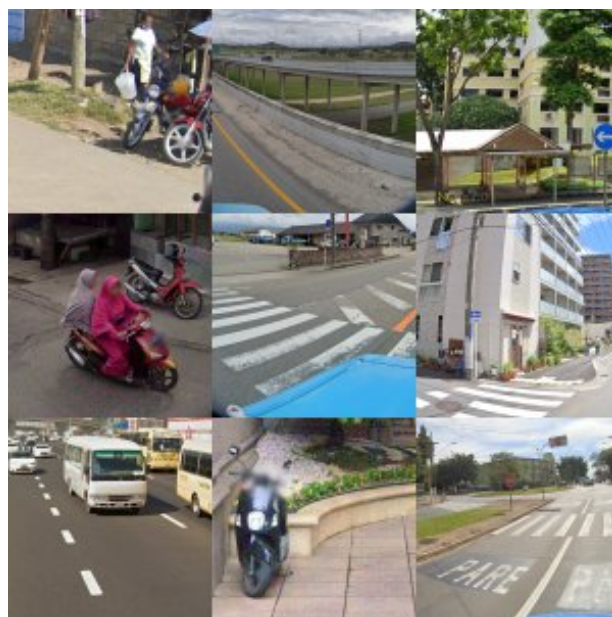
## 1. ВЫБОР МОДЕЛИ НЕЙРОННОЙ СЕТИ ДЛЯ ОБУЧЕНИЯ

САРТСНА в формате изображений широко используется для защиты ресурсов от автоматизированных ботов и может быть реализована несколькими способами. Как правило, такие САРТСНА направлены на проверку способности пользователя распознавать и интерпретировать объекты на изображении. Наиболее распространены два варианта реализации (оба варианта реализации проиллюстрированы на рис. 1.1):

1. цельное изображение, содержащее несколько объектов, частично размытых или искажённых, при этом изображение разбито на сетку  $3 \times 3$  или  $4 \times 4$ . Пользователю предлагается выбрать ячейки, содержащие объекты определённого класса (например, автобусы или светофоры);
2. составное изображение, сформированное из 9 или 12 отдельных фрагментов (изображений), каждый из которых представляет собой независимое изображение – зачастую низкого качества, с наложением артефактов или шумов. Задача пользователя – выбрать те изображения, где присутствует нужный объект.



а)



б)

Рис. 1.1 Изображения САРТСНА с размером сетки  $3 \times 3$ : а) – цельное, б) – составное.

Такие САРТСНА требуют от системы автоматического анализа способности как к глобальному восприятию изображения, так и к локальной интерпретации его фрагментов. Соответственно, модель, предназначенная для решения данной задачи, должна поддерживать:

1. классификацию объектов на уровне отдельных изображений (для CAPTCHA, основанных на отдельных картинках в сетке);
2. локализацию и сегментацию объектов с высокой точностью, чтобы корректно определить границы объектов в пределах ячеек, особенно в случаях, когда объект может частично заходить за границу между ячейками.

Для решения этих задач были рассмотрены следующие современные архитектуры нейронных сетей:

1. YOLO (You Only Look Once) – однопроходная модель, объединяющая классификацию и регрессию ограничивающих рамок в одной свёрточной архитектуре. Отличается высокой скоростью и хорошей точностью [1; 2];
2. Faster R-CNN – двухступенчатая модель, в которой сначала генерируются области предложений, а затем выполняется классификация и уточнение рамок. Обладает высокой точностью, но уступает в скорости [3];
3. DETR (DEtection TRansformer) – основана на архитектуре трансформеров, что позволяет эффективно моделировать глобальные взаимосвязи между объектами. Подходит для задач с большим количеством контекстных зависимостей, но требует больше ресурсов для обучения [4].

Среди этих архитектур было принято решение использовать YOLOv8 по следующим причинам [5]:

1. высокая производительность: YOLOv8 показывает высокую скорость обработки изображений без значительного ущерба для точности, что критично в условиях, когда необходимо обрабатывать CAPTCHA в реальном времени;
2. гибкость и масштабируемость: модель предоставляет множество предобученных вариантов с различной глубиной и числом параметров (версии n, s, m, l, x), что позволяет использовать как на слабых, так и на производительных устройствах;
3. широкая поддержка и документация: YOLOv8 имеет активное сообщество, подробную документацию и регулярно обновляется, что значительно упрощает интеграцию и адаптацию модели под пользовательские задачи;
4. поддержка сегментации: в отличие от более ранних версий, YOLOv8 поддерживает не только детекцию, но и сегментацию объектов, что

особенно важно для задач, где необходимо точно определить область объекта внутри изображения;

5. дообучение на пользовательских данных: YOLOv8 позволяет эффективно дообучать модель на собственных датасетах, что особенно важно при работе с CAPTCHA-изображениями, содержащими специфические классы объектов и нестандартные искажения.

Кроме того, модель YOLOv8 была успешно протестирована в задачах, близких по структуре к CAPTCHA: детекции дорожных знаков, транспортных средств, пешеходов и других объектов в сложных условиях съёмки, что подтверждает её универсальность и применимость к рассматриваемой задаче.

Таким образом, YOLOv8 является наиболее сбалансированным выбором, обеспечивающим как точную классификацию, так и локализацию объектов в условиях ограниченных ресурсов и с возможностью адаптации под специфику визуальных CAPTCHA.

## 2. ПАРСИНГ ИЗОБРАЖЕНИЙ CAPTCHA И ИХ ПРЕДОБРАБОТКА ДЛЯ СОЗДАНИЯ ДАТАСЕТА

### 2.1. Парсинг реальных CAPTCHA с различных web-ресурсов

Большинство предобученных моделей компьютерного зрения, таких как YOLOv8, обучены на датасете COCO [6], содержащем изображения высокого качества с чёткими контурами и однозначной аннотацией объектов. Однако CAPTCHA с изображениями имеют принципиально иные характеристики: они могут включать в себя размытие, наложенные артефакты, искажения, шумы, повторяющиеся элементы и искусственно пониженное разрешение. Всё это снижает эффективность использования стандартных датасетов и моделей, не адаптированных под такие условия.

Для обеспечения высокой точности в задаче автоматического решения CAPTCHA необходимо подготовить собственный набор данных, приближённый к реальным условиям использования. Наиболее эффективным методом является автоматизированный парсинг изображений CAPTCHA, представленных на веб-сайтах, использующих визуальные CAPTCHA-решения, такие как Google reCAPTCHA v2.

Использование реальных CAPTCHA, собранных в автоматическом режиме, имеет ряд преимуществ по сравнению с синтетической генерацией данных:

1. изображения содержат разнообразные сцены, освещение, углы обзора и уровни шума, что положительно влияет на способность модели к обобщению;
2. присутствует большое количество уникальных объектов на фоне, в том числе в частично перекрытых и смазанных вариантах;
3. отсутствует необходимость в ручной генерации изображений и создании дополнительных искажений для повышения реалистичности;
4. возможно извлекать текстовые инструкции к CAPTCHA, что позволяет соотносить каждое изображение с требуемым классом.

Для парсинга CAPTCHA был реализован автоматизированный сценарий взаимодействия с браузером с использованием библиотеки Selenium [7]. Данный подход позволяет воспроизвести действия пользователя при работе с CAPTCHA, обходя при этом ручной ввод. Для обеспечения стабильной работы и масштабируемости процесса применялась браузерная автоматизация через WebDriver (в частности, ChromeDriver).

Функциональность парсера включает следующие ключевые этапы:



1. поиск `iframe`-элемента, содержащего чекбокс «Я не робот», и эмуляция клика по нему для инициирования визуальной CAPTCHA;
2. ожидание загрузки CAPTCHA и извлечение изображения с заданием (включая его URL или пиксельный снимок);
3. извлечение информации о структуре сетки (количество строк и столбцов), на которую разбито изображение CAPTCHA;
4. получение текста задания, содержащего имя объекта (например, «выберите все изображения с мотоциклами»), для последующего использования в аннотации данных.

Типичная CAPTCHA представляет собой изображение, разделённое на сетку из  $3 \times 3$  или  $4 \times 4$  ячеек, каждая из которых может содержать фрагмент сцены. При этом пользователю предлагается выбрать ячейки, в которых присутствует объект заданного класса. Процесс парсинга может быть представлена блок-схемой на рис. 2.1.

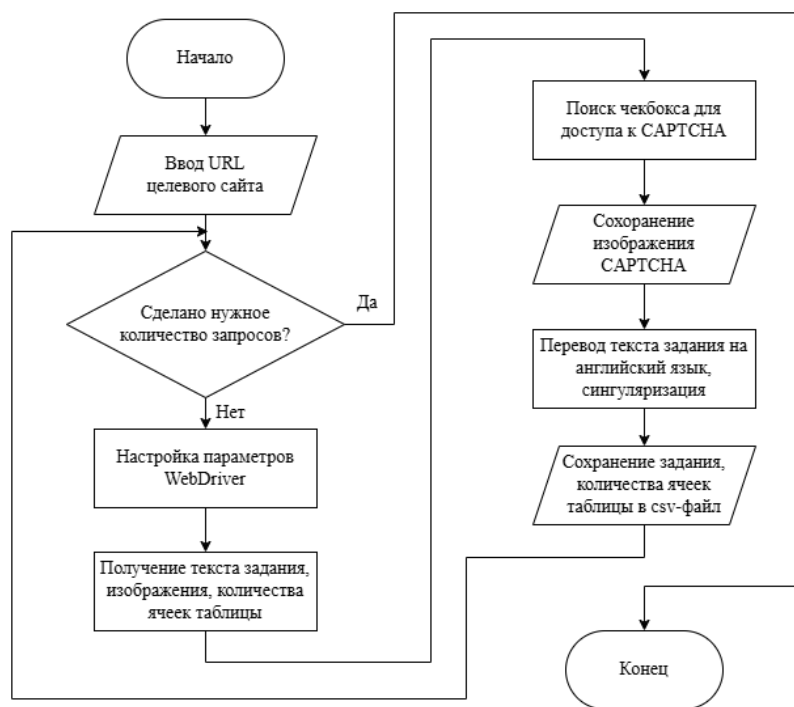


Рис. 2.1 Блок-схема процесса парсинга CAPTCHA.

Полученные изображения и метаданные (включая текст задания и параметры сетки) используются для формирования обучающего датасета, пригодного для дообучения модели YOLOv8 в задачах классификации и сегментации объектов.

## 2.2. Предварительная обработка изображений датасета

После получения достаточного количества изображений для составления датасета необходимо провести их предварительную обработку и разметку.

ку. Это один из самых важных этапов работы, поскольку от качества разметки напрямую зависит точность и эффективность последующей работы модели.

Для корректной работы модели YOLO требуется создать иерархическую структуру папок, в которой изображения и соответствующие метки будут разделены на тренировочную и валидационную выборки. Стандартная структура включает следующие директории:

1. Директория `train` – содержит тренировочную выборку:
  - 1.1. `images` – изображения;
  - 1.2. `labels` – метки к изображениям.
2. Директория `val` – содержит валидационную выборку:
  - 2.1. `images` – изображения;
  - 2.2. `labels` – метки к изображениям.

Набор классов, пути к выборкам и параметры конфигурации задаются в YAML-файле, который передается при обучении модели. Содержимое такого файла для данной модели:

Листинг 2.1

#### Параметры конфигурации для обучения модели

---

```

1 path: ../datasets/image_dataset
2 train: images/train
3 val: images/val
4
5 nc: 9 # Количество классов
6 names: ['pedestrian transition', 'stair', 'motorcycle', 'bus', 'traffic light', 'car', 'bicycle', 'fire hydrant', 'tractor']

```

---

Для создания меток используется инструмент CVAT (Computer Vision Annotation Tool) – многофункциональное веб-приложение с поддержкой аннотации объектов с помощью полигонов, прямоугольников и других форм. CVAT позволяет экспортировать разметку напрямую в формат, совместимый с YOLO [8].

Поскольку САРТСНА-изображения часто содержат объекты с нечёткими контурами, наложением и визуальными искажениями, особенно важно использовать ручную точную разметку, а не ограничиваться автоматическими методами. Выделение объектов должно проводиться как можно точнее, с учётом геометрии контуров. На рисунке ниже представлен пример изображения с размеченными объектами:



Рис. 2.2 Пример разметки изображения с тестовой САРТСНА.

Кроме того, разметка позволяет учесть сразу несколько объектов разных классов на одном изображении, что особенно характерно для САРТСНА, где в одной сетке могут одновременно находиться, например, автомобили и автобусы. Такой подход положительно влияет на обобщающую способность модели.

В случае, если количество данных по отдельным классам окажется недостаточным, можно дополнительно использовать методы аугментации: вращение, масштабирование, искажение цвета и контраста. Однако при хорошо организованном парсинге и разметке зачастую удастся обойтись без аугментации.

### 3. ОБУЧЕНИЕ И ТЕСТИРОВАНИЕ МОДЕЛИ YOLOV8 НА РЕАЛЬНЫХ САПТСНА

В качестве основной архитектуры была выбрана модель YOLOv8m-seg, поддерживающая сегментацию объектов. Она представляет собой сбалансированное решение между качеством распознавания, производительностью и требованиями к аппаратному обеспечению. Благодаря своей универсальности, модель подходит как для задач классификации, так и для задач детектирования и сегментации, что особенно важно при работе с САПТСНА, содержащими зашумлённые или плохо различимые объекты.

Преимущества YOLOv8m-seg заключаются в следующем:

1. наличие встроенной поддержки сегментации объектов, что особенно важно при необходимости выделения фрагментов изображений;
2. возможность использования предобученных весов, сокращающих время на обучение и повышающих стартовую точность;
3. высокая скорость инференса по сравнению с другими моделями сегментации (например, Mask R-CNN или DETR);
4. встроенные средства аугментации (изменения яркости, повороты, масштабирование и пр.);
5. удобный интерфейс через библиотеку ultralytics, позволяющий быстро запускать обучение, логировать метрики и визуализировать результаты;
6. полная совместимость с аннотациями в формате YOLO, полученными из CVAT.

Перед запуском обучения структура данных была организована в соответствии с требованиями YOLOv8: директории train и val содержали соответствующие изображения и файлы разметки, а в .yaml файле конфигурации были указаны пути к выборкам и список классов.

Обучение проводилось на 35 эпохах при размере изображений 640×640 пикселей и размере батча 8. Использование предобученных весов позволило достичь стабильного снижения функции потерь с первых эпох, а встроенные механизмы аугментации способствовали улучшению обобщающей способности модели.

Результаты обучения отслеживались по ключевым метрикам (IoU, Precision, Recall, Loss), которые визуализировались автоматически. Примеры графиков с результатами обучения приведены ниже:

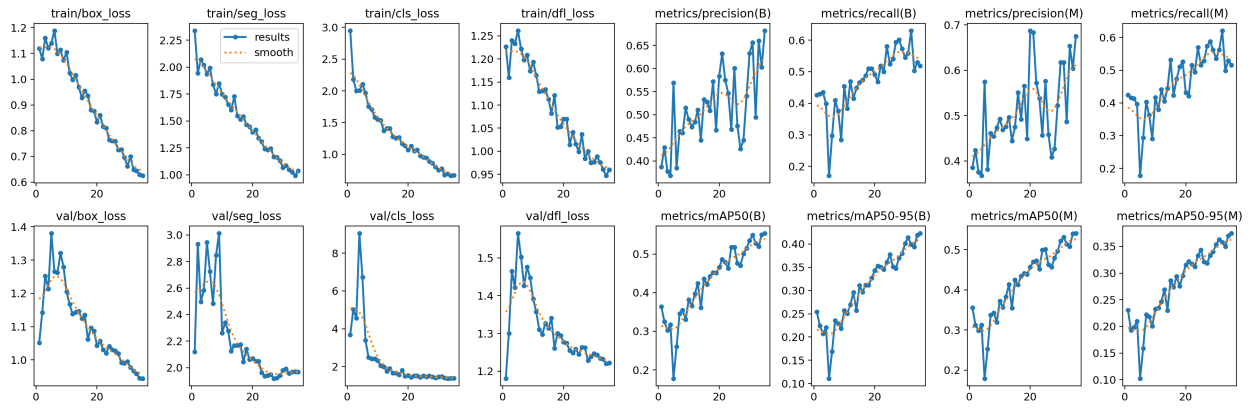


Рис. 3.1 Изменение ключевых метрик в процессе обучения.

Также, была построена нормализованная матрица ошибок для определения точности предсказания необходимых классов на валидационной выборке, которая представлена на рис. 3.2.

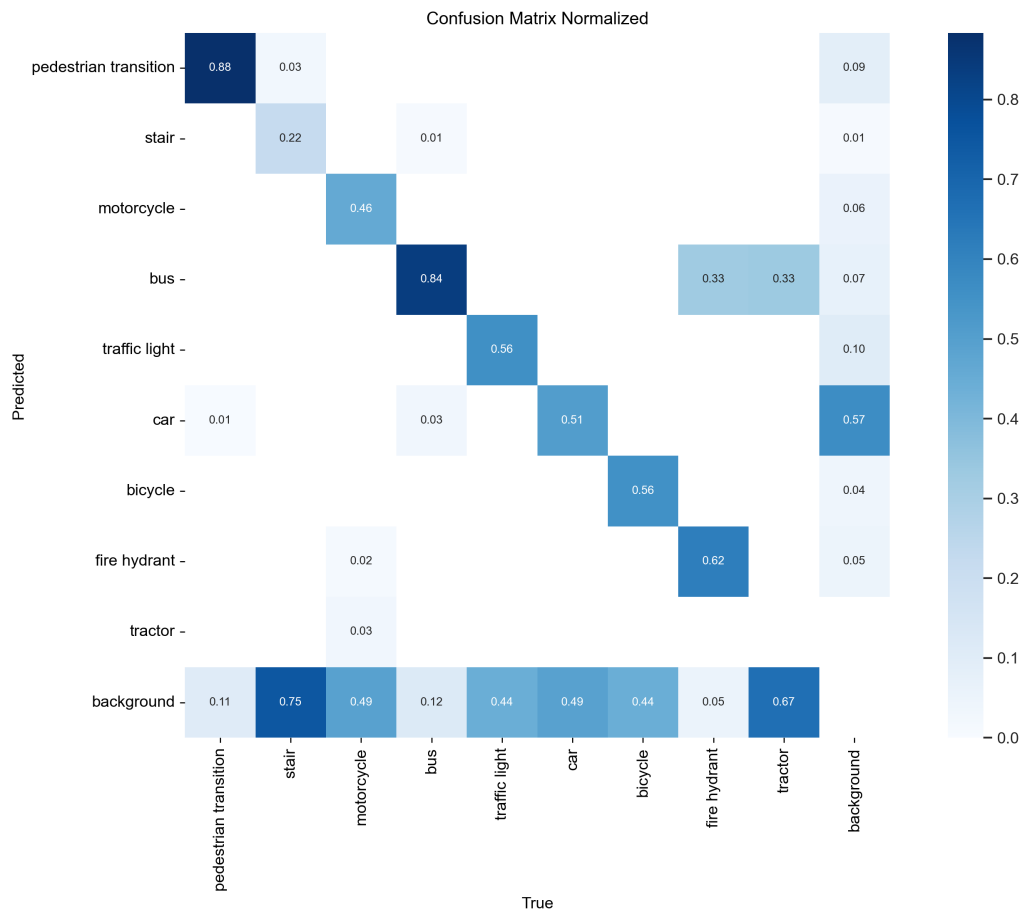


Рис. 3.2 Матрица ошибок для изображений валидационной выборки.

После завершения обучения модель была протестирована на реальных САРТСНА, собранных с помощью автоматического парсера, реализованного на базе библиотеки Selenium. Тестирование проводилось в автоматическом

режиме, имитируя реальные действия пользователя в браузере, что позволило оценить работоспособность системы в условиях, приближенных к реальной эксплуатации.

Сценарий тестирования предусматривал выполнение следующих шагов:

1. автоматический переход к странице с CAPTCHA и активация чекбокса «Я не робот»;
2. извлечение изображения CAPTCHA (включая структуру сетки и текст задания);
3. определение целевого объекта из текста задания (например, «выберите все изображения с автобусами»);
4. разбиение изображения CAPTCHA на ячейки (в зависимости от размера сетки —  $3 \times 3$  или  $4 \times 4$ );
5. применение обученной модели для сегментации и классификации каждого изображения или фрагмента;
6. определение ячеек, содержащих нужный класс, и программная симуляция кликов по ним;
7. повторная попытка прохождения CAPTCHA в случае, если результат оказался некорректным (что также фиксировалось в логах).

Тестирование было организовано в виде цикла, позволяющего автоматически проходить CAPTCHA до тех пор, пока не будет достигнут положительный результат. Это позволило зафиксировать частоту ошибок модели и определить случаи, в которых требуются дообучение или оптимизация.

Рабочий процесс тестирования и взаимодействия модели с CAPTCHA представлен на блок-схеме ниже.

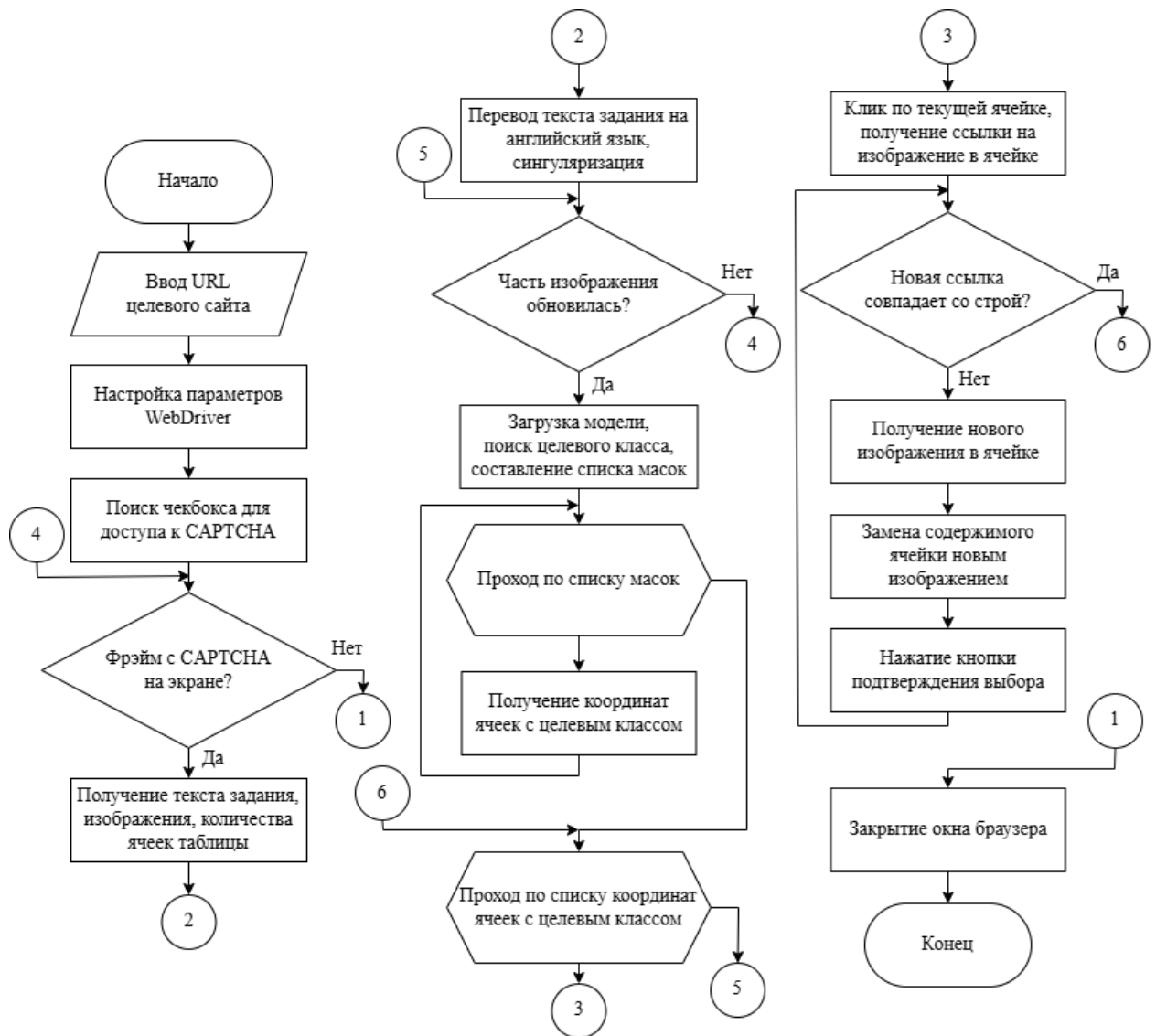


Рис. 3.3 Блок-схема процесса прохождения CAPTCHA.

Полученные данные используются для последующего анализа качества модели и корректировки процесса обучения. Основное внимание при анализе будет уделено типам ошибок, сложности распознаваемых объектов и влиянию качества исходного изображения на точность сегментации.

## ЗАКЛЮЧЕНИЕ

В данной работе была реализована система автоматического распознавания и прохождения графических CAPTCHA с использованием нейросетевой архитектуры YOLOv8, поддерживающей сегментацию объектов. Основное внимание уделялось решению задачи классификации и локализации визуальных элементов, характерных для современных CAPTCHA, применяемых на web-ресурсах.

В процессе выполнения работы были решены ключевые задачи:

1. проведён анализ распространённых форматов графических CAPTCHA, что позволило сформулировать требования к точности и скорости работы модели;
2. выбрана и адаптирована модель YOLOv8m-seg, которая обеспечила оптимальный баланс между производительностью и точностью распознавания;
3. сформирован собственный набор данных: собраны реальные CAPTCHA-изображения, проведена их разметка с использованием CVAT, а также реализован инструмент сбора изображений с помощью Selenium;
4. настроен процесс обучения модели с учётом особенностей структуры CAPTCHA, включая разнообразие классов объектов (транспорт, пешеходы, светофоры и др.);
5. разработан скрипт для автоматического взаимодействия с браузером, обеспечивающий эмуляцию действий пользователя и автоматическую отправку распознанных ответов;
6. проведено тестирование модели в условиях, приближенных к реальным, что подтвердило достаточную точность сегментации объектов и эффективность системы при прохождении CAPTCHA.

Полученные результаты продемонстрировали применимость современных нейросетевых моделей компьютерного зрения, в частности YOLOv8 с сегментацией, для решения практических задач в области автоматизации взаимодействия с защищёнными web-интерфейсами. Разработанная система может быть эффективно использована в процессе автоматического тестирования web-приложений, где требуется обход CAPTCHA для имитации пользовательских сценариев, а также в исследовательских целях – для анализа устойчивости визуальных CAPTCHA к автоматическим методам распознавания.



Возможными направлениями дальнейшего развития проекта являются: расширение классов распознаваемых объектов, интеграция с другими методами (например, OCR для комбинированных CAPTCHA), а также оптимизация модели под работу в условиях ограниченных вычислительных ресурсов (например, в мобильных или браузерных приложениях).

Таким образом, поставленная цель была достигнута, а разработанная система подтвердила свою эффективность в реальных условиях эксплуатации.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. *Редмон Дж. Фархади А.* YOLO9000: Better, Faster, Stronger // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). — 2017.
2. YOLO - официальная документация [Электронный ресурс]. — URL: <https://docs.ultralytics.com>.
3. *Жэнь Ш. Гиршик Р.* Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks // Advances in Neural Information Processing Systems (NeurIPS). Т. 28. — 2015.
4. *Каруон Н. Масса Ф.* End-to-End Object Detection with Transformers // European Conference on Computer Vision (ECCV). — 2020.
5. Обзор модели Ultralytics YOLOv8 [Электронный ресурс]. — URL: <https://docs.ultralytics.com/ru/models/yolov8/>.
6. COCO Dataset: Common Objects in Context [Электронный ресурс]. — URL: <https://cocodataset.org/>.
7. Selenium WebDriver — автоматизация браузера [Электронный ресурс]. — URL: <https://www.selenium.dev/>.
8. CVAT — инструмент аннотирования изображений [Электронный ресурс]. — URL: <https://github.com/openai/cvat>.

## ПРИЛОЖЕНИЕ

## Листинг 1

## Исходный код получения CAPTCHA с целевого сайта

---

```

1  # Подключение библиотек для работы с браузером
2  from selenium import webdriver
3  from selenium.webdriver.remote.webdriver import WebDriver
4  from selenium.webdriver.common.by import By
5
6  # Подключение библиотек для работы с текстом задания captcha
7  from deep_translator import GoogleTranslator
8  import inflect
9
10 # Библиотека для парсинга HTML
11 from bs4 import BeautifulSoup
12
13 from random import randint
14 import time
15 import requests
16 import os
17 import csv
18
19
20 class GetCaptcha():
21     """
22     Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
23     На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
24     """
25
26     def __init__(self, browser: WebDriver):
27         """Конструктор класса"""
28         super().__init__()
29         self.browser = browser
30
31
32     def get_captcha(self, link: str, cnt: int) -> tuple[str, str, str]:
33         """Метод получения captcha со страницы"""
34         # Проходим по ссылке
35         self.browser.get(link)
36         time.sleep(randint(3, 5))
37
38         # Переключаемся на фрейм с чекбоксом captcha
39         self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '//*[@id="g-recaptcha"]/div/div/iframe'))
40         # Кликаем по чекбоксу "Я не робот"
41         self.browser.find_element(By.XPATH, '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
42         time.sleep(randint(3, 5))
43
44         # Переключаемся на обычную веб-страницу
45         self.browser.switch_to.default_content()
46         # Переключаемся на фрейм с картинкой captcha
47         self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '/html/body/div[2]/div[4]/iframe'))
48         # Находим элемент, содержащий ссылку на исходное изображение
49         image = self.browser.find_element(By.XPATH,
50     ↪ '//*[@id="rc-imageselect-target"]/table/tbody/tr[1]/td[1]/div/div[1]/img').get_attribute('src')
51         # Делаем запрос для получения файла
52         response = requests.get(image)
53         response.raise_for_status()
54
55         # Получаем название объекта, который надо найти
56         object_name = self.browser.find_element(By.XPATH, '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]/div/strong').text
57
58         # Получаем таблицу с кусочками изображения
59         table = self.browser.find_element(By.XPATH, '//*[@id="rc-imageselect-target"]/table').get_attribute('outerHTML')

```

```

60     # Создаем папку для хранения временных файлов
61     if not os.path.isdir('./datasets/imagecaptcha_dataset'):
62         os.mkdir('./datasets/imagecaptcha_dataset')
63     path_to_file = f'./datasets/imagecaptcha_dataset/{cnt}.jpg'
64     # Сохраняем файл
65     with open(f'{path_to_file}', 'wb') as imageCaptcha:
66         imageCaptcha.write(response.content)
67
68     return object_name, path_to_file, table
69
70
71 def get_number_of_cells(self, table:str) -> tuple[int, int]:
72     '''Метод для получения количества ячеек таблицы для последующего разбиения изображения на части'''
73     # Парсинг HTML
74     soup = BeautifulSoup(table, 'lxml')
75
76     # Получаем количество строк
77     number_of_rows = len(soup.find_all('tr'))
78
79     # Получаем количество столбцов
80     number_of_columns = len(soup.find('tr').find_all(['td', 'th']))
81
82     return number_of_rows, number_of_columns
83
84
85 if __name__ == "__main__":
86     # Целевой сайт
87     target_link = 'https://rucaptcha.com/demo/recaptcha-v2'
88     for cnt in range(463, 638):
89         # Настройку user agent
90         USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
91         ↪ Safari/537.36"
92         options = webdriver.ChromeOptions()
93
94         options.add_experimental_option("excludeSwitches", ["enable-automation"])
95         options.add_experimental_option('useAutomationExtension', False)
96         options.add_argument(f"user-agent={USER_AGENT}")
97         options.add_argument("--disable-blink-features=AutomationControlled")
98
99         # Передача параметров
100        browser = webdriver.Chrome(options=options)
101        browser.implicitly_wait(30)
102
103        captcha = GetCaptcha(browser)
104        # Получение captcha и объекта для поиска
105        task_object, image, table = captcha.get_captcha(target_link, cnt)
106
107        # Перевод названия объекта на английский и сохранение его в единственном числе
108        task_object = GoogleTranslator(source='auto', target='en').translate(task_object)
109        singular = inflect.engine()
110        if len(task_object) > 3:
111            # Исключаем ошибки с множественным числом для слов, которые не могут быть во множественном числе из-за малого
112            ↪ количества символов
113            task_object = singular.singular_noun(task_object)
114            if task_object.lower() == 'hydrant':
115                task_object = 'fire hydrant'
116
117        # Получаем количество ячеек
118        rows, columns = captcha.get_number_of_cells(table)
119
120        # Запись полученных параметров в csv-файл
121        with open('images_for_captcha.csv', 'a') as datasetFile:
122            csv_rows = csv.writer(datasetFile, quoting=csv.QUOTE_NONE)
123            csv_rows.writerow([task_object, image, rows, columns])

```

---

## Листинг 2

## Исходный код дообучения модели на датасете

---

```

1 from ultralytics import YOLO
2
3 # Загрузка модели
4 model = YOLO("yolov8m-seg.pt") # Загрузка предобученной лёгкой модели
5 # Дообучение модель
6 model.train(
7     data="./datasets/image_dataset/image_captcha.yaml", # Путь к файлу конфигурации
8     epochs=35,
9     imgsz=640,
10    batch=8,
11    workers=4,
12    device="cpu",
13    name="captcha_seg" # Название директории для сохранения результатов обучения
14 )

```

---

## Листинг 3

## Исходный код автоматизированного решения CAPTCHA на сайте

---

```

1 # Подключение библиотек для работы с браузером
2 from selenium import webdriver
3 from selenium.webdriver.remote.webdriver import WebDriver
4 from selenium.webdriver.common.by import By
5 from selenium.common.exceptions import ElementClickInterceptedException
6
7 # Подключение библиотек для работы с текстом задания captcha
8 from deep_translator import GoogleTranslator
9 import inflect
10
11 # Библиотека для парсинга HTML
12 from bs4 import BeautifulSoup
13
14 # Библиотека для работы с изображениями
15 from ultralytics import YOLO
16 import cv2
17 import numpy as np
18
19 from random import randint
20 import time
21 import requests
22
23
24 class SolveCaptcha():
25     """
26     Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
27     На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
28     """
29
30     def __init__(self, browser: WebDriver):
31         """Конструктор класса"""
32         super().__init__()
33         self.browser = browser
34
35
36     def find_captcha(self, link: str):
37         # Проходим по ссылке
38         self.browser.get(link)
39         time.sleep(randint(3, 5))
40
41         # Переключаемся на фрейм с чекбоксом captcha
42         self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '//*[@id="g-recaptcha"]/div/div/iframe'))

```

```

43 # Кликаем по чекбоксу "Я не робот"
44 self.browser.find_element(By.XPATH, '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
45 time.sleep(randint(3, 5))
46
47 # Переключаемся на обычную web-страницу
48 self.browser.switch_to.default_content()
49 # Переключаемся на фрейм с картинкой captcha
50 self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '/html/body/div[2]/div[4]/iframe'))
51
52
53 def get_captcha(self) -> tuple[str, str, str, np.ndarray]:
54     "Метод получения captcha со страницы"
55     # Находим элемент, содержащий ссылку на исходное изображение
56     src_image = self.browser.find_element(By.XPATH,
57     ↪ '//*[@id="rc-imageselect-target"]/table/tbody/tr[1]/td[1]/div/div[1]/img').get_attribute('src')
58     # Делаем запрос для получения файла
59     response = requests.get(src_image)
60     response.raise_for_status()
61
62     # Получаем название объекта, который надо найти
63     object_name = self.browser.find_element(By.XPATH, '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]/div/strong').text
64
65     # Получаем таблицу с кусочками изображения
66     table = self.browser.find_element(By.XPATH, '//*[@id="rc-imageselect-target"]/table').get_attribute('outerHTML')
67
68     # Преобразование байтовой последовательности в изображение
69     image = cv2.imdecode(np.frombuffer(response.content, np.uint8), cv2.IMREAD_COLOR)
70
71     return object_name, table, src_image, image
72
73 def get_properties_for_recognition(self, task_object: str, table: str) -> tuple[str, int, int]:
74     "Метод для получения необходимых параметров для распознавания на картинке"
75     # Перевод названия объекта на английский и сохранение его в единственном числе
76     task_object = GoogleTranslator(source='auto', target='en').translate(task_object)
77     singular = inflect.engine()
78     if len(task_object) > 3:
79         # Исключаем ошибки с множественным числом для слов, которые не могут быть во множественном числе из-за малого
80         ↪ количества символов
81         task_object = singular.singular_noun(task_object)
82         if task_object.lower() == 'hydrant':
83             task_object = 'fire hydrant'
84
85     # Парсинг HTML
86     soup = BeautifulSoup(table, 'lxml')
87     # Получаем количество строк
88     number_of_rows = len(soup.find_all('tr'))
89     # Получаем количество столбцов
90     number_of_columns = len(soup.find('tr').find_all(['td', 'th']))
91
92     return task_object, number_of_rows, number_of_columns
93
94 def predict_class(self, image: np.ndarray, task_object: str) -> list:
95     "Метод для получения масок для изображения с необходимым классом"
96     # Передаем в предобученную модель изображение для поиска нужного объекта
97     results = model(image)[0]
98     class_names = model.names
99
100     # Получаем идентификатор нужного класса
101     for id, name in class_names.items():
102         if name == task_object.lower():
103             class_id = id
104             break
105
106     # Получаем все маски для классов
107     masks = results.masks.data.cpu().numpy()

```

```

108 classes = results_boxes.cls.cpu().numpy()
109
110 # Получаем список масок для нужного класса
111 selected_masks = [masks[i] for i in range(len(classes)) if int(classes[i]) == class_id]
112
113 return selected_masks
114
115
116 def get_cells_with_mask(self, cells_with_object: list, coords_cells: list, mask: np.ndarray, grid_size: tuple, threshold: float) -> list:
117     "Метод для получения ячеек таблицы, содержащих объект"
118     # Определяем размер ячейки
119     cell_height, cell_width = int(mask.shape[0] / grid_size[0]), int(mask.shape[1] / grid_size[1])
120     idx_cell = 0
121
122     for i in range(grid_size[0]):
123         for j in range(grid_size[1]):
124             # Координаты прямоугольника, соответствующего ячейке
125             y1, y2 = i * cell_height, (i + 1) * cell_height
126             x1, x2 = j * cell_width, (j + 1) * cell_width
127
128             # Вырезаем часть маски, соответствующую ячейке
129             cell_mask = mask[y1:y2, x1:x2]
130             # Рассчитываем какую часть ячейки занимает объект
131             coverage_area = np.sum(cell_mask) / cell_mask.size
132
133             # Проверяем, есть ли объект в ячейке
134             if coverage_area >= threshold:
135                 # Сохраняем данные о ячейке
136                 cells_with_object.append(idx_cell)
137                 coords_cells.append((i, j))
138
139             idx_cell += 1
140
141     return cells_with_object, coords_cells
142
143
144 if __name__ == "__main__":
145     # Загружаем модель
146     model = YOLO('best.pt')
147
148     # Целевой сайт
149     target_link = 'https://rucaptcha.com/demo/recaptcha-v2'
150
151     # Настройки user agent
152     USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
153     ↪ Safari/537.36"
154
155     options = webdriver.ChromeOptions()
156
157     options.add_experimental_option("excludeSwitches", ["enable-automation"])
158     options.add_experimental_option('useAutomationExtension', False)
159     options.add_argument(f"user-agent={USER_AGENT}")
160     options.add_argument("--disable-blink-features=AutomationControlled")
161
162     # Передача параметров
163     browser = webdriver.Chrome(options=options)
164     browser.implicitly_wait(30)
165
166     captcha = SolveCaptcha(browser)
167     # Находим фрейм с captcha (автоматизация клика на чекбокс)
168     captcha.find_captcha(target_link)
169
170     # Выполняем распознавание до тех пор, пока фрейм не исчезнет
171     while True:
172         try:
173             # Получение изображения captcha и объекта для поиска
174             task_object, table, src_image, image = captcha.get_captcha()
175             # Получаем необходимые параметры captcha

```

```

174 task_object, rows, columns = captcha.get_properties_for_recognition(task_object, table)
175
176 RECURSIVE_CAPTCHA = True # Флаг для captcha, в которых вместо выбранных изображений появляются новые
177 while RECURSIVE_CAPTCHA:
178     # Сбрасываем флаг рекурсии
179     RECURSIVE_CAPTCHA = False
180     # Находим нужный класс на изображении
181     selected_masks = captcha.predict_class(image, task_object)
182
183     cells_with_object, coords = [], []
184     for mask in selected_masks:
185         # Проходим по выбранным маскам для определения клетки к которой она принадлежит
186         resized_mask = cv2.resize(mask, (image.shape[1], image.shape[0]), interpolation=cv2.INTER_NEAREST)
187         cells_with_object, coords = captcha.get_cells_with_mask(cells_with_object, coords, resized_mask, (rows, columns), 0.05)
188
189     # Кликаем по ячейкам с уникальными индексами
190     for cell, coord in list(set(zip(cells_with_object, coords))):
191         captcha.browser.find_elements(By.TAG_NAME, 'td')[cell].click()
192         time.sleep(randint(2, 3))
193         # Проверяем наличие новых изображений в данной ячейке
194         src_cell = captcha.browser.find_elements(By.TAG_NAME, 'td')[cell].find_element(By.TAG_NAME, 'img').get_attribute('src')
195         if src_cell != src_image:
196             # Делаем запрос для получения изображения
197             response = requests.get(src_cell)
198             response.raise_for_status()
199             cell_image = cv2.imdecode(
200                 np.frombuffer(response.content, np.uint8),
201                 cv2.IMREAD_COLOR
202             )
203
204             # Заменяем в исходном изображении старую ячейку на новую
205             x1, x2 = coord[0] * cell_image.shape[0], (coord[0] + 1) * cell_image.shape[0]
206             y1, y2 = coord[1] * cell_image.shape[1], (coord[1] + 1) * cell_image.shape[1]
207             image[x1:x2, y1:y2] = cell_image
208
209             # Устанавливаем флаг рекурсии
210             RECURSIVE_CAPTCHA = True
211
212     # Находим кнопку подтверждения выбора и кликаем по ней
213     captcha.browser.find_element(By.XPATH, '//*[@id="recaptcha-verify-button"]').click()
214     time.sleep(randint(3, 5))
215 except ElementClickInterceptedException:
216     captcha.browser.quit()
217 break
218

```

---