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 CAPTCHA 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 27 страниц, включая 3 рисунка и 1 таблицу.

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.

Отчёт оформлен с помощью системы компьютерной вёрстки TEX и его расширения XETEX из дистрибутива *TeX Live*.

СОДЕРЖАНИЕ

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

CAPTCHA в формате изображений широко используется для защиты ресурсов от автоматизированных ботов и может быть реализована несколькими способами. Как правило, такие CAPTCHA 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 3×3 или 4×4. 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 CAPTCHA с размером сетки 3×3: а) – цельное, б) – составное.

Такие CAPTCHA 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:

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 [5];
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 РЕАЛЬНЫХ CAPTCHA ДЛЯ СОЗДАНИЯ ДАТАСЕТА

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.

2.1. Преимущества парсинга реальных CAPTCHA

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.

2.2. Автоматизация сбора данных

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 ра-

боты 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 3×3 или 4×4 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.

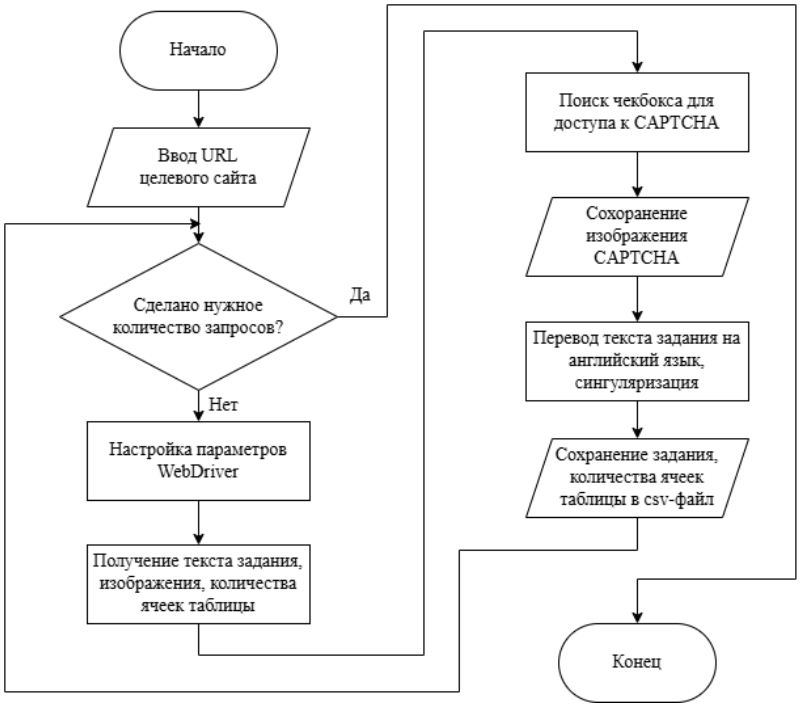

Рис. 2.1 Блок-схема процесса парсинга CAPTCHA.

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.

3. ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ ДАТАСЕТА

После получения достаточного количества изображений для составления датасета необходимо провести их предварительную обработку и разметку. Это один из самыхважных 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 3.1 Параметры конфигурации для обучения модели

```yaml
path: ../datasets/image_dataset
train: images/train
val: images/val

nc: 9  # Количество классов
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 CAPTCHA-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. 3.1 Пример разметки изображения с тестовой CAPTCHA.

Кроме того, разметка позволяет учесть сразу несколько объектов разных классов на одном изображении, что особенно характерно для CAPTCHA, 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 удается обойтись без аугментации.

4. ОБУЧЕНИЕ И ТЕСТИРОВАНИЕ МОДЕЛИ YOLOV8 НА РЕАЛЬНЫХ CAPTCHA

4.1. Обучение модели

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 CAPTCHA, 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:

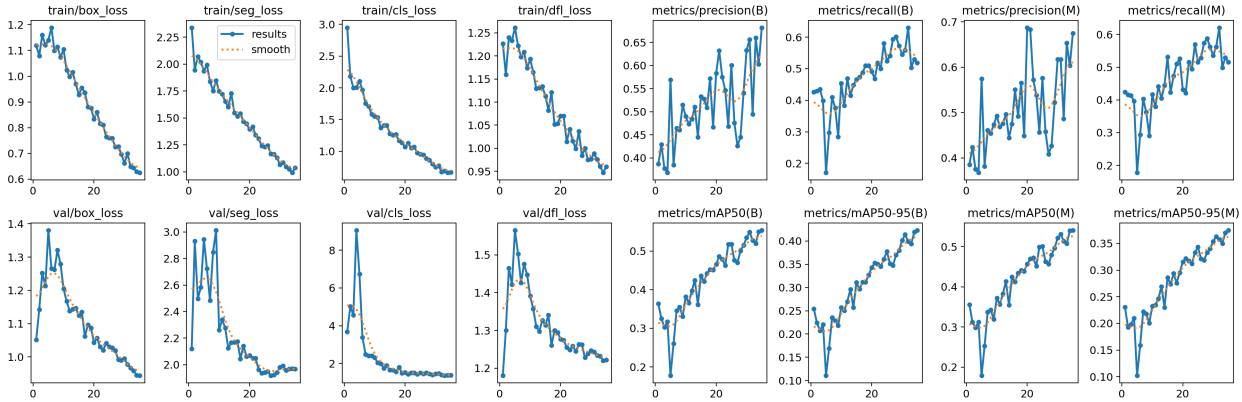

Рис. 4.1 Изменение ключевых метрик в процессе обучения.

Также, была построена нормализованная матрица ошибок для определения точности предсказания необходимых классов на валидационной выборке, которая представлена на рис. 4.2.

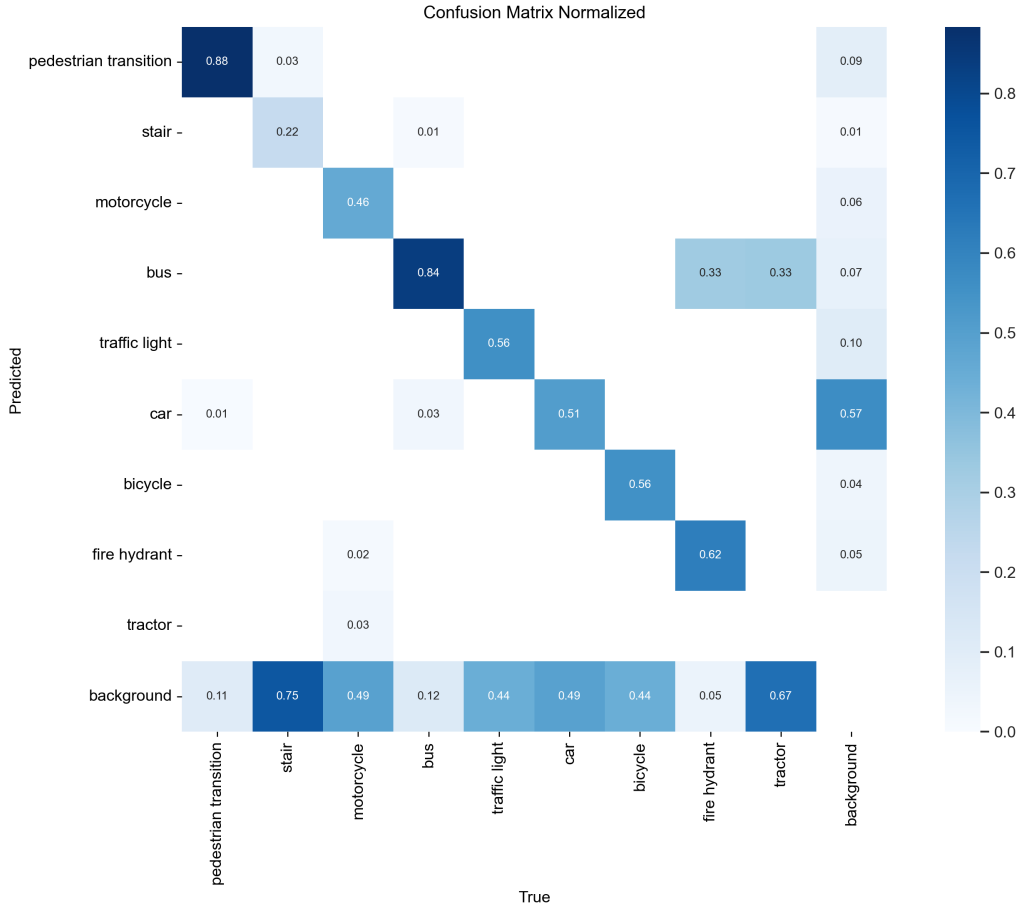

Рис. 4.2 Матрица ошибок для изображений валидационной выборки.

4.2. Тестирование модели

После завершения обучения модель была протестирована на реальных CAPTCHA, 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 — 3×3 или 4×4);
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.

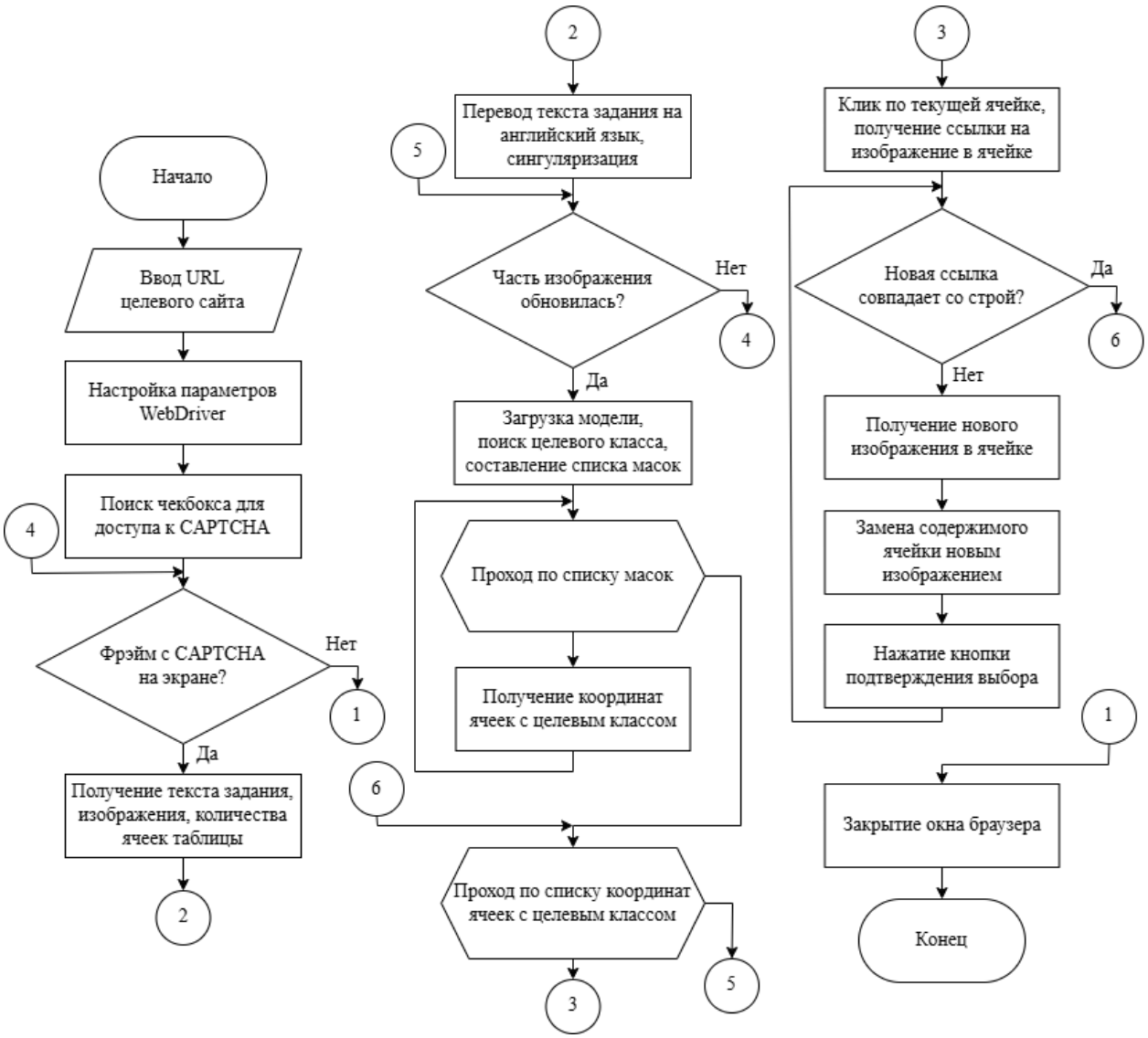

Рис. 4.3 Блок-схема процесса прохождения CAPTCHA.

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 рамках данной работы была реализована система автоматического распознавания и прохождения CAPTCHA с изображениями, основанная на использовании нейросетевой модели YOLOv8 с поддержкой сегментации.

В ходе исследования были решены следующие задачи:

1. проанализированы основные форматы CAPTCHA и выделены ключевые требования к модели;
2. выбран подходящий вариант модели – YOLOv8m-seg, обеспечивающий баланс между точностью и производительностью;
3. собран и размечен собственный датасет CAPTCHA-изображений с использованием инструментов Selenium и CVAT;
4. реализована и протестирована система обучения и предобработки данных;
5. создан скрипт автоматизированного взаимодействия с CAPTCHA в браузере;
6. проведено тестирование модели в реальных условиях, продемонстрировавшее достаточную точность распознавания объектов и успешность прохождения CAPTCHA.

Результаты работы подтверждают применимость современных моделей компьютерного зрения для решения задач, связанных с автоматизацией взаимодействия с защищёнными web-ресурсами. Разработанная система может использоваться как в рамках автоматического тестирования web-интерфейсов, так и в исследованиях в области распознавания сложных визуальных паттернов.

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. YOLOv8 - 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. *Карион Н. Масса Ф.* End-to-End Object Detection with Transformers // European Conference on Computer Vision (ECCV). — 2020.
5. *Бочковский А. Ван Ч.-Я.* YOLOv4: Optimal speed and accuracy of object detection // arXiv:2004.10934 [Электронный ресурс]. — 2020.
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/opencv/cvat.

ПРИЛОЖЕНИЕ

Листинг 4.1 Исходный код получения CAPTCHA с целевого сайта

```python
# Подключение библиотек для работы с браузером
from selenium import webdriver
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.common.by import By

# Подключение библиотек для работы с текстом заданиия captcha
from deep_translator import GoogleTranslator
import inflect

# Библиотека для парсинга HTML
from bs4 import BeautifulSoup

from random import randint
import time
import requests
import os
import csv


class GetCaptcha():
    '''
    Основной класс проекта, который управляет вызовом дочерних
    ↪ классов для решения определенных видов captcha
    На начальном этапе здесь также будет все, что касатеся
    ↪ получения captcha с веб-страницы
    '''

    def __init__(self, browser: WebDriver):
        '''Конструктор класса'''
        super().__init__()
        self.browser = browser


    def get_captcha(self, link: str, cnt: int) -> tuple[str, str,
    ↪ str]:
        '''Метод получения captcha со страницы'''
        # Проходим по ссылке
        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(
40              By.XPATH,
41              '//*[@id="g-recaptcha"]/div/div/iframe'
42          ))
43          # Кликаем по чекбоксу "Я не робот"
44          self.browser.find_element(
45              By.XPATH,
46              '/html/body/div[2]/div[3]/div[1]/div/div/span'
47          ).click()
48          time.sleep(randint(3, 5))
49
50          # Переключаемся на обычную web-страницу
51          self.browser.switch_to.default_content()
52          # Переключаемся на фрейм с картинкой captcha
53          self.browser.switch_to.frame(self.browser.find_element(
54              By.XPATH,
55              '/html/body/div[2]/div[4]/iframe'
56          ))
57          # Находим элемент, содержащий ссылку на исходное
            ↪  изображение
58          image = self.browser.find_element(
59              By.XPATH,
60              '//*[@id="rc-imageselect-target"]/table/tbody'+
61              '/tr[1]/td[1]/div/div[1]/img'
62          ).get_attribute('src')
63          # Делаем запрос для получения файла
64          response = requests.get(image)
65          response.raise_for_status()
66
67          # Получаем название объекта, который надо найти
68          object_name = self.browser.find_element(
69              By.XPATH,
70              '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]'+
71              '/div/strong'
72          ).text
73
74          # Получаем таблицу с кусочками изображения
75          table = self.browser.find_element(
76              By.XPATH,
77              '//*[@id="rc-imageselect-target"]/table'
78          ).get_attribute('outerHTML')
79
```

```
80          # Создаем папку для хранения временных файлов
81          if not os.path.isdir('../datasets/imagecaptcha_dataset'):
82              os.mkdir('../datasets/imagecaptcha_dataset')
83          path_to_file =
             ↪  f'../datasets/imagecaptcha_dataset/{cnt}.jpg'
84          # Сохраняем файл
85          with open(f'{path_to_file}', 'wb') as imageCaptcha:
86              imageCaptcha.write(response.content)
87
88          return object_name, path_to_file, table
89
90
91      def get_number_of_cells(self, table:str) -> tuple[int, int]:
92          '''Метод для получения колличества ячеек таблицы для
             ↪  последующего разбиения изображения на части'''
93          # Парсинг HTML
94          soup = BeautifulSoup(table, 'lxml')
95
96          # Получаем количество строк
97          number_of_rows = len(soup.find_all('tr'))
98
99          # Получаем количество столбцов
100         number_of_columns = len(soup.find('tr').find_all(['td',
             ↪  'th']))
101
102         return number_of_rows, number_of_columns
103
104
105 if __name__ == "__main__":
106     # Целевой сайт
107     target_link = 'https://rucaptcha.com/demo/recaptcha-v2'
108     for cnt in range(463, 638):
109         # Настройки user agent
110         USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
             ↪  AppleWebKit/537.36 (KHTML, like Gecko)
             ↪  Chrome/127.0.0.0 Safari/537.36"
111         options = webdriver.ChromeOptions()
112
113         options.add_experimental_option("excludeSwitches",
             ↪  ["enable-automation"])
114         options.add_experimental_option('useAutomationExtension',
             ↪  False)
115         options.add_argument(f"user-agent={USER_AGENT}")
116         options.add_argument(
117             "--disable-blink-features=AutomationControlled"
```

```
        )

        # Передача параметров
        browser = webdriver.Chrome(options=options)
        browser.implicitly_wait(30)

        captcha = GetCaptcha(browser)
        # Получение captcha и объекта для поиска
        task_object, image, table =
        ↪  captcha.get_captcha(target_link, cnt)

        # Перевод названия объекта на английский и сохранение его
        ↪  в единственном числе
        task_object = GoogleTranslator(source='auto',
        ↪  target='en').translate(task_object)
        singular = inflect.engine()
        if len(task_object) > 3:
            # Исключаем ошибки с множественным числом для слов,
            ↪  которые не могут быть во множественном числе
            ↪  из-за малого количества символов
            task_object = singular.singular_noun(task_object)
            if task_object.lower() == 'hydrant':
                task_object = 'fire hydrant'

        # Получаем количество ячеек
        rows, columns = captcha.get_number_of_cells(table)

        # Запись полученных параметров в csv-файл
        with open('images_for_captcha.csv', 'a') as datasetFile:
            csv_rows = csv.writer(datasetFile,
            ↪  quoting=csv.QUOTE_NONE)
            csv_rows.writerow([task_object, image, rows, columns])
```

Листинг 4.2 Исходный код дообучения модели на датасете

```
from ultralytics import YOLO

# Загрузка модели
model = YOLO("yolov8m-seg.pt")  # Загрузка предобученной лёгкой
↪  модели
# Дообучение модель
model.train(
    data="../datasets/image_dataset/image_captcha.yaml",  # Путь
    ↪  к файлу конфигурации
```

```
    epochs=35,
    imgsz=640,
    batch=8,
    workers=4,
    device="cpu",
    name="captcha_seg"  # Название директории для сохранения
     ↪  результатов обучения
)
```

Листинг 4.3 Исходный код автоматизированного решения CAPTCHA на сайте

```
# Подключение библиотек для работы с браузером
from selenium import webdriver
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.common.by import By
from selenium.common.exceptions import
 ↪  ElementClickInterceptedException

# Подключение библиотек для работы с текстом задания captcha
from deep_translator import GoogleTranslator
import inflect

# Библиотека для парсинга HTML
from bs4 import BeautifulSoup

# Библиотека для работы с изображениями
from ultralytics import YOLO
import cv2
import numpy as np

from random import randint
import time
import requests


class SolveCaptcha():
    '''
    Основной класс проекта, который управляет вызовом дочерних
     ↪  классов для решения определенных видов captcha
    На начальном этапе здесь также будет все, что касается
     ↪  получения captcha с веб-страницы
    '''

```

```
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(
43              By.XPATH,
44              '//*[@id="g-recaptcha"]/div/div/iframe'
45          ))
46          # Кликаем по чекбоксу "Я не робот"
47          self.browser.find_element(By.XPATH,
            ↪  '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
48          time.sleep(randint(3, 5))
49
50          # Переключаемся на обычную web-страницу
51          self.browser.switch_to.default_content()
52          # Переключаемся на фрейм с картинкой captcha
53          self.browser.switch_to.frame(self.browser.find_element(
54              By.XPATH,
55              '/html/body/div[2]/div[4]/iframe'
56          ))
57
58
59      def get_captcha(self) -> tuple[str, str, str, np.ndarray]:
60          '''Метод получения captcha со страницы'''
61          # Находим элемент, содержащий ссылку на исходное
            ↪  изображение
62          src_image = self.browser.find_element(
63              By.XPATH,
64              '//*[@id="rc-imageselect-target"]/table/tbody/'+
65              'tr[1]/td[1]/div/div[1]/img'
66          ).get_attribute('src')
67          # Делаем запрос для получения файла
68          response = requests.get(src_image)
69          response.raise_for_status()
70
71          # Получаем название объекта, который надо найти
72          object_name = self.browser.find_element(
```

```
73              By.XPATH,
74              '//*[@id=”rc-imageselect”]/div[2]/div[1]/div[1]/'+
75              'div/strong'
76          ).text
77
78          # Получаем таблицу с кусочками изображения
79          table = self.browser.find_element(
80              By.XPATH,
81              '//*[@id=”rc-imageselect-target”]/table'
82          ).get_attribute('outerHTML')
83
84          # Преобразование байтовой последовательности в изображение
85          image = cv2.imdecode(np.frombuffer(response.content,
            ↪ np.uint8), cv2.IMREAD_COLOR)
86
87          return object_name, table, src_image, image
88
89
90      def get_properties_for_recognition(self, task_object: str,
        ↪ table: str) -> tuple[str, int, int]:
91          '''Метод для получения необходимых параметров для
            ↪ распознавания на картинке'''
92          # Перевод названия объекта на английский и сохранение его
            ↪ в единственном числе
93          task_object = GoogleTranslator(source='auto',
            ↪ target='en').translate(task_object)
94          singular = inflect.engine()
95          if len(task_object) > 3:
96              # Исключаем ошибки с множественным числом для слов,
                ↪ которые не могут быть во множественном числе
                ↪ из-за малого количества символов
97              task_object = singular.singular_noun(task_object)
98              if task_object.lower() == 'hydrant':
99                  task_object = 'fire hydrant'
100
101         # Парсинг HTML
102         soup = BeautifulSoup(table, 'lxml')
103         # Получаем количество строк
104         number_of_rows = len(soup.find_all('tr'))
105         # Получаем количество столбцов
106         number_of_columns = len(soup.find('tr').find_all(['td',
            ↪ 'th']))
107
108         return task_object, number_of_rows, number_of_columns
109
```

```
110
111     def predict_class(self, image: np.ndarray, task_object: str)
        ↪  -> list:
112         '''Метод для получения масок для изображения с необходимым
            ↪  классом'''
113         # Передаем в предобученную модель изображение для поиска
            ↪  нужного объекта
114         results = model(image)[0]
115         class_names = model.names
116
117         # Получаем идентификатор нужного класса
118         for id, name in class_names.items():
119             if name == task_object.lower():
120                 class_id = id
121                 break
122
123         # Получаем все маски для классов
124         masks = results.masks.data.cpu().numpy()
125         classes = results.boxes.cls.cpu().numpy()
126
127         # Получаем список масок для нужного класса
128         selected_masks = [masks[i] for i in range(len(classes)) if
            ↪  int(classes[i]) == class_id]
129
130         return selected_masks
131
132
133     def get_cells_with_mask(self, cells_with_object: list,
        ↪  coords_cells: list, mask: np.ndarray, grid_size: tuple,
        ↪  threshold: float) -> list:
134         '''Метод для получения ячеек таблицы, содержащих объект'''
135         # Определяем размер ячейки
136         cell_height, cell_width = int(mask.shape[0] /
            ↪  grid_size[0]), int(mask.shape[1] / grid_size[1])
137         idx_cell = 0
138
139         for i in range(grid_size[0]):
140             for j in range(grid_size[1]):
141                 # Координаты прямоугольника, соответствующего
                    ↪  ячейке
142                 y1, y2 = i * cell_height, (i + 1) * cell_height
143                 x1, x2 = j * cell_width, (j + 1) * cell_width
144
145                 # Вырезаем часть маски, соответствующую ячейке
146                 cell_mask = mask[y1:y2, x1:x2]
```

```
147                     # Рассчитываем какую часть ячейки занимает объект
148                     coverage_area = np.sum(cell_mask) / cell_mask.size
149 
150                     # Проверяем, есть ли объект в ячейке
151                     if coverage_area >= threshold:
152                         # Сохраняем данные о ячейке
153                         cells_with_object.append(idx_cell)
154                         coords_cells.append((i, j))
155 
156                     idx_cell += 1
157 
158             return cells_with_object, coords_cells
159 
160 
161 if __name__ == "__main__":
162     # Загружаем модель
163     model = YOLO('best.pt')
164 
165     # Целевой сайт
166     target_link = 'https://rucaptcha.com/demo/recaptcha-v2'
167 
168     # Настройки user agent
169     USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
        ↪ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
        ↪ Safari/537.36"
170     options = webdriver.ChromeOptions()
171 
172     options.add_experimental_option("excludeSwitches",
        ↪ ["enable-automation"])
173     options.add_experimental_option('useAutomationExtension',
        ↪ False)
174     options.add_argument(f"user-agent={USER_AGENT}")
175     options.add_argument(
176         "--disable-blink-features=AutomationControlled"
177     )
178 
179     # Передача параметров
180     browser = webdriver.Chrome(options=options)
181     browser.implicitly_wait(30)
182 
183     captcha = SolveCaptcha(browser)
184     # Находим фрейм с captcha (автоматизация клика на чекбокс)
185     captcha.find_captcha(target_link)
186 
187     # Выполняем распознавание до тех пор, пока фрейм не исчезнет
```

```
188         while True:
189             try:
190                 # Получение изображения captcha и объекта для поиска
191                 task_object, table, src_image, image =
                    ↪ captcha.get_captcha()
192                 # Полчаем необходимые параметры captcha
193                 task_object, rows, columns =
                    ↪ captcha.get_properties_for_recognition(task_object,
                    ↪ table)
194
195                 RECURSIVE_CAPTCHA = True    # Флаг для captcha, в
                    ↪ которых вместо выбранных изображений появляются
                    ↪ новые
196                 while RECURSIVE_CAPTCHA:
197                     # Сбрасываем флаг рекурсии
198                     RECURSIVE_CAPTCHA = False
199                     # Находим нужный класс на изображении
200                     selected_masks = captcha.predict_class(image,
                        ↪ task_object)
201
202                     cells_with_object, coords = [], []
203                     for mask in selected_masks:
204                         # Проходим по выбранным маскам для определения
                            ↪ клетки к которой она принадлежит
205                         resized_mask = cv2.resize(mask,
                            ↪ (image.shape[1], image.shape[0]),
                            ↪ interpolation=cv2.INTER_NEAREST)
206                         cells_with_object, coords =
                            ↪ captcha.get_cells_with_mask(cells_with_object,
                            ↪ coords, resized_mask, (rows, columns),
                            ↪ 0.05)
207
208                     # Кликаем по ячейкам с уникальными индексами
209                     for cell, coord in list(set(zip(cells_with_object,
                        ↪ coords))):
210                         captcha.browser.find_elements(By.TAG_NAME,
                            ↪ 'td')[cell].click()
211                         time.sleep(randint(2, 3))
212                         # Проверяем наличие новых изображений в данной
                            ↪ ячейке
213                         src_cell =
                            ↪ captcha.browser.find_elements(By.TAG_NAME,
                            ↪ 'td')[cell].find_element(By.TAG_NAME,
                            ↪ 'img').get_attribute('src')
214                         if src_cell != src_image:
```

```
215                         # Делаем запрос для получения изображения
216                         response = requests.get(src_cell)
217                         response.raise_for_status()
218                         cell_image = cv2.imdecode(
219                             np.frombuffer(response.content,
                                ↪  np.uint8),
220                             cv2.IMREAD_COLOR
221                         )
222
223                         # Заменяем в исходном изображении старую
                            ↪  ячейку на новую
224                         x1, x2 = coord[0] * cell_image.shape[0],
                            ↪  (coord[0] + 1) * cell_image.shape[0]
225                         y1, y2 = coord[1] * cell_image.shape[1],
                            ↪  (coord[1] + 1) * cell_image.shape[1]
226                         image[x1:x2, y1:y2] = cell_image
227
228                         # Устанавливаем флаг рекурсии
229                         RECURSIVE_CAPTCHA = True
230
231                 # Находим кнопку подтверждения выбора и кликаем по ней
232                 captcha.browser.find_element(By.XPATH,
                    ↪  '//*[@id="recaptcha-verify-button"]').click()
233                 time.sleep(randint(3, 5))
234             except ElementClickInterceptedException:
235                 captcha.browser.quit()
236                 break
237
```