Технически университет - София
Филиал Пловдив

Дипломна работа

Тема:

Система за разпознаване на обект в изображение

Студент: Любомир Ламбрев
Фак №: 614973

Специалност: КСТ

Образователна степен: ОКС Магистър

Факултет: ФЕА

ТУ – София, Филиал Пловдив, 2025 г.

Съдържание

Увод

Развитието на технологиите за изкуствен интелект и машинно обучение доведе до значителен напредък в областта на компютърното зрение. Разпознаването на обекти в изображения намира широко приложение в различни индустрии, включително сигурност, автоматизация, медицина и търговия. Съвременните системи за разпознаване на обекти предоставят възможност за автоматична идентификация и класификация на визуални елементи, което значително улеснява анализа на изображения и ускорява обработката на информация.

В настоящата дипломна работа се разработва уеб базирана система за разпознаване на обекти в изображения. Основната ѝ цел е да даде възможност на потребителите да качват изображения, които да бъдат анализирани чрез алгоритми за машинно обучение, а разпознатите обекти да се съхраняват в база данни за последващо търсене и визуализация.

Системата ще бъде реализирана с помощта на съвременни уеб технологии, включително Streamlit за изграждане на потребителския интерфейс, както и OpenCV за обработка на изображения. За съхранение на данните ще бъде използвана Mongodb за нерелационна база от данни.

Глава 1 – Обзор

1.1 - Състояние на проблема по литературни данни

Разпознаването на обекти в изображения е ключова област в съвременната компютърна визия, която намира широко приложение в индустрията, медицината, сигурността и автономните системи. Основната цел е автоматичното идентифициране и локализиране на обекти от различни класове в цифрови изображения или видеопотоци. През последните десетилетия подходите за решаване на този проблем претърпяват значително развитие – от класически методи за обработка на изображения до усъвършенствани модели, базирани на дълбоко обучение[10,11].

В ранните години се използват техники, основани на ръчни характеристики като контури, ръбове, текстурни дескриптори и цветови хистограми. Въпреки че тези методи предоставят определено ниво на надеждност, те са ограничени от сложността на сцената и вариациите в осветлението, мащаба и ъгъла на гледане. Тези недостатъци стимулират преминаването към по-автоматизирани решения, при които се прилагат алгоритми за машинно обучение, обучени върху набори от предварително извлечени характеристики [10].

С навлизането на дълбоките невронни мрежи и по-специално свързващите (convolutional) архитектури, се постига значителен пробив. Модели като R-CNN, Fast R-CNN и Faster R-CNN поставят основите на модерните методи, като комбинират регионални предложения и свързващи слоеве за висока точност [1,2,3]. Последващото развитие води до системи в реално време като YOLO (You Only Look Once) и SSD (Single Shot Multibox Detector), които значително намаляват времето за обработка, без да жертват съществено точността [4,5,7].

Съвременните изследвания насочват усилията към подобряване на ефективността, устойчивостта и възможността за работа с ограничени изчислителни ресурси. Голямо внимание се отделя на леките архитектури (MobileNet, EfficientDet), които са оптимизирани за мобилни устройства и

вградени системи. Наред с това се разработват и хибридни подходи, съчетаващи класическа обработка на изображения с невронни мрежи за постигане на баланс между точност и изчислителна сложност [11].

В заключение може да се каже, че състоянието на проблема в областта на разпознаването на обекти е силно повлияно от динамичното развитие на дълбокото обучение. Основните предизвикателства днес са свързани с оптимизацията на скоростта, адаптивността към нови класове и устойчивостта при условия на шум и промяна в средата [10].

1.2 - Преглед на Уеб базирана система за машинно зрение

Уеб базираните системи за машинно зрение представляват интеграция между технологии за обработка на изображения и уеб среди, която позволява достъп до алгоритми за анализ и разпознаване на обекти директно чрез браузър или клиентско приложение. Основното им предимство е, че не изискват инсталиране на сложен софтуер на локалния компютър – обработката се извършва на сървърна страна, което осигурява по-голяма производителност, мащабируемост и възможност за работа на устройства с ограничени ресурси.

На пазара съществуват множество платформи с различна насоченост. Някои от тях са предназначени за общо приложение и поддържат широк набор от модели за детекция, сегментация и класификация, докато други са фокусирани върху специфични индустрии като медицинска диагностика, автоматизация на производството или системи за сигурност. Пример за широко използвани решения са Google Cloud Vision, Amazon Rekognition и Microsoft Azure Computer Vision, които предоставят API за интеграция и работят с различни формати изображения и видеа.

Все по-често се срещат и системи с отворен код, които позволяват персонализация и обучение на собствени модели – например базирани на TensorFlow, PyTorch или YOLO архитектури. Тези решения се използват от разработчици и изследователи за изграждане на гъвкави и адаптивни приложения, включително и уеб интерфейси с функционалности за качване,

анализ и визуализация на резултати. Стремежа е насочена към комбиниране на локални и облачни изчисления, като се постига баланс между бързодействие, сигурност на данните и удобство за крайния потребител.

1.3 - Съществуващи системи за детекция и сегментация на обекти

Едни от най-популярните архитектури за детекция на обекти са YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector) и Faster R-CNN. YOLO се отличава с висока скорост и подходящост за приложения в реално време, докато Faster R-CNN осигурява по-висока точност при сложни сцени, но за сметка на по-голямо време за обработка [4]. SSD заема междинно място, като съчетава добра скорост и точност, показано в таблица 1. В областта на сегментацията доминират модели като Mask R-CNN, U-Net и DeepLab, които позволяват пикселно ниво на класификация и са особено подходящи за медицински изображения, автономни превозни средства и роботика.

Model	**Pascal VOC (mAP)**	**COCO (mAP)**	**ImageNet (mAP)**	**Open Images (mAP)**	**Inference Speed (mAP)**	**Model Size (MB)**
RCNN	66%	54%	60%	55%	~5 FPS	200
Fast RCNN	70%	59%	63%	58%	~7 FPS	150
Faster RCNN	75%	65%	68%	63%	~10 FPS	180
Mask RCNN	76%	66%	69%	64%	~8 FPS	230
YOLO	72.5%	58.5%	61.5%	57.5%	~45 - 60 FPS	145
SSD	75%	63.5%	66.5%	61.5%	~19 – 46 FPS	145

Таблица 1. Количествено сравнение на производителността на модели за откриване на обекти на различни набор от данни [12]

На пазара се предлагат множество готови платформи за детекция и сегментация. Сред комерсиалните решения се открояват Google Cloud Vision API, Amazon Rekognition и Microsoft Azure Custom Vision, които предоставят лесна интеграция чрез уеб услуги и API. Тези платформи са оптимизирани за мащабна обработка, но често са ограничени откъм персонализация и изискват заплащане за по-голям

обем заявки. Отворените решения като OpenCV, Detectron2 и MMDetection предоставят на разработчиците пълен контрол върху модела, настройките и обучението, като в същото време позволяват интеграция в уеб или локални системи.

В последните години се развиват и хибридни подходи, които комбинират предварително обучени модели с възможност за дообучаване върху специфични набори от данни. Това позволява на компаниите да адаптират системите към конкретни условия – например разпознаване на дефекти в производствена линия или класификация на специфични видове растения. В допълнение, внедряването на оптимизационни техники като квантоване и хардуерно ускорение с GPU и TPU прави възможна реализацията на такива системи дори върху мобилни и вградени устройства.

Сегментацията на обекти, като по-сложна задача от детекцията, изисква по-големи изчислителни ресурси и оптимизация на алгоритмите. Например, в автономното шофиране е критично всяка рамка от видеопотока да бъде обработена в реално време с минимална латентност, като едновременно се откриват пешеходци, превозни средства, пътни знаци и препятствия. В медицинската диагностика обаче се търси максимална точност и устойчивост на модела, което налага използване на по-сложни архитектури и по-дълго време за обработка.

Развитието на машинното зрение показва, че бъдещето на тези системи ще бъде в интеграцията им в уеб платформи с лесен достъп, автоматично мащабиране и възможност за съвместна работа с други технологии като анализ на видео потоци, 3D реконструкция и комбиниране на данни от различни сензори. Това ще позволи по-бързо внедряване на решения в различни сектори и ще улесни преминаването от експериментални прототипи към реални продукти.

1.3.1 – Ultralytics YOLO

Ultralytics YOLO (You Only Look Once) е една от най-широко използваните съвременни системи за детекция и сегментация на обекти в изображения и видеа. Основната идея на YOLO моделите е едноетапната обработка на изображенията,

при която цялото изображение се анализира за обекти в рамките на една невронна мрежа.

Ultralytics YOLO11 представлява най-новия модел в серията YOLO (You Only Look Once) за задачи от областта на компютърното зрение като:

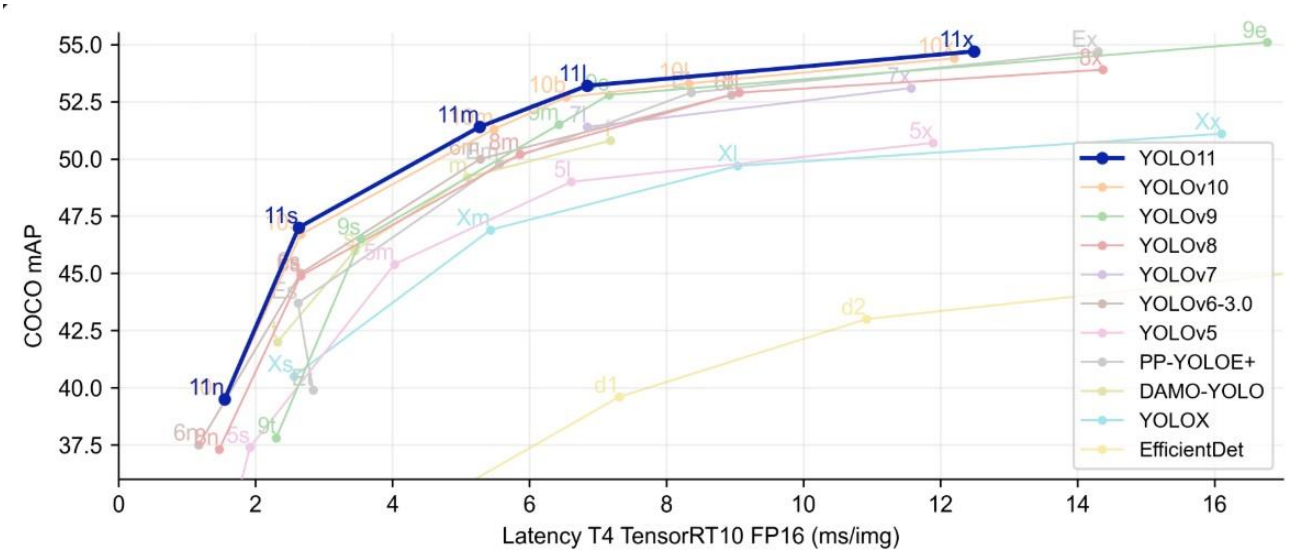
- **Детекция (Detection)**
- **Сегментация (Segmentation)**
- **Класификация (Classification)**
- **Оценка на позата (Pose Estimation)**
- **Ориентирани ограничителни кутии (Oriented Bounding Boxes)**

Моделът постига значителни подобрения по отношение на точност, скорост и изчислителна ефективност, в сравнение спрямо предходните модели.

Ultralytics YOLO11 предлага модулен подход към жизнения цикъл на модела чрез няколко режима:

- **Train**: обучение върху собствен или стандартен набор от данни
- **Val**: валидация за оценка на mAP и контрол на преобучението
- **Predict**: високо-производителен извод върху изображения и видеа
- **Export**, **Track**, **Benchmark**: съответно за експорт на модели, проследяване на обекти и измерване на производителност

YOLO11 постига по-голяма точност с по-малко параметри чрез подобрения в проектирането на модели и техниките за оптимизация. Подобрената архитектура позволява ефективно извличане и обработка на характеристики, което води до по-висока средна точност (mAP) върху набори от данни като COCO, като същевременно използва 22% по-малко параметри от предишните модели. Това прави YOLO11 изчислително ефективен без компромис с точността, което го прави подходящ за внедряване на устройства с ограничени ресурси (фиг.1).



Фиг.1 Сравнителни показатели на YOLO11 спрямо други модели [13]

Един от недостатъците на YOLO моделите е когато обектите се припокриват значително. YOLO може да срещне проблеми при сцените със силно застъпващи се обекти, където локализацията става по-малко надеждна. Това може да доведе до неточни прогнози за ограничаващи рамки и присвоявания на класове.

1.3.2 - SSD (Single Shot MultiBox Detector)

SSD (Single Shot MultiBox Detector) също така е една от най-широко използваните системи за детекция на обекти в изображения и видеа. Моделът е проектиран така, че в рамките само на едно преминаване през конволюционната невронна мрежа извежда едновременно координатите на ограничителни кутии и вероятностите за принадлежност към даден клас. Това го отличава от по-ранните двустъпкови подходи като R-CNN, Fast R-CNN и Faster R-CNN които първо генерират предложения за региони и след това ги класифицират.

SSD може да се използва в широк кръг задачи от областта на компютърното зрение, включително:

- **Детекция (Detection)**
- **Класификация на обекти в сцени (Object Classification in Scenes)**
- **Откриване на обекти в различни мащаби чрез мулти-мащабни характеристики (Multi-Scale Detection)**

Основната иновация на SSD е използването на множество слоеве от мрежата за

предсказване на обекти с различни размери. По-ранните слоеве са по-чувствителни към малки обекти, докато по-дълбоките слоеве засичат по-големи структури [7]. Освен това, SSD използва предварително дефинирани **"default boxes"** (кутии с различни размери и пропорции), които улесняват детекцията на обекти с различна форма и мащаб.

Архитектурата на SSD е проектирана така, че да бъде изчислително ефективна. Например, версията SSD300 (с входни изображения 300x300 px) може да обработва над 59 кадъра в секунда върху GPU, като постига високи стойности на средна точност (mAP) върху наборите PASCAL VOC и COCO показано в таблица 1 [12]. По-големият вариант SSD512 предлага още по-добра точност за сметка на скоростта. Комбинацията от бързодействие и висока точност прави SSD особено ценен за приложения като видеонаблюдение и мобилни устройства, където времето за реакция е съществено важно.

Недостатък на SSD е, че представянето му при много малки обекти не е толкова добро, колкото при по-късни архитектури като RetinaNet или EfficientDet, които използват допълнителни техники за балансиране на класификацията и локализацията или по-ефективни архитектурни подобрения. Въпреки това, SSD остава важен междинен етап в развитието на моделите за детекция и продължава да бъде база за много доработки и усъвършенствания.

1.3.3 - Faster R-CNN

Faster R-CNN представлява усъвършенствана версия на предходните модели R-CNN и Fast R-CNN, насочена към по-бърза и точна детекция на обекти [3]. Основната идея зад модела е въвеждането на **Region Proposal Network (RPN)** – мрежа, която автоматично генерира кандидати за региони (region proposals) директно от конволюционните характеристики. Това позволява елиминирането на външни алгоритми като Selective Search, използвани в предишни архитектури, което значително ускорява процеса на откриване на обекти[6].

Faster R-CNN може да се използва в широк кръг задачи от областта на компютърното зрение, включително:

- **Детекция (Detection)**

- **Локализация чрез ограничителни кутии (Bounding Box Localization)**
- **Класификация на обекти в изображения (Object Classification)**

Faster R-CNN е двуетапен детектор: първо RPN предлага региони от интерес, а след това мрежа за класификация и регресия обработва тези региони, за да определи класа на обекта и да прецизира неговата позиция. Това го прави по-бавен от едностъпковите методи, но значително по-точен, особено при работа с големи и сложни набори от данни като COCO и PASCAL VOC.

Недостатъците на ранните версии R-CNN и Fast R-CNN са свързани с високата изчислителна цена и времето за обучение. Например, R-CNN изисква генериране на около 2000 предложения за региони чрез Selective Search за всяко изображение, след което всяко предложение се подава в отделна CNN за извличане на характеристики [6]. Това води до изключително бавна обработка и големи изисквания към паметта. Fast R-CNN оптимизира част от този процес, но все още разчита на Selective Search, което ограничава скоростта [2]. Faster R-CNN решава този проблем чрез RPN, но остава по-бавен в сравнение с едностъпкови модели като SSD и YOLO.

Глава 2 - Теоретично решение на поставената задача

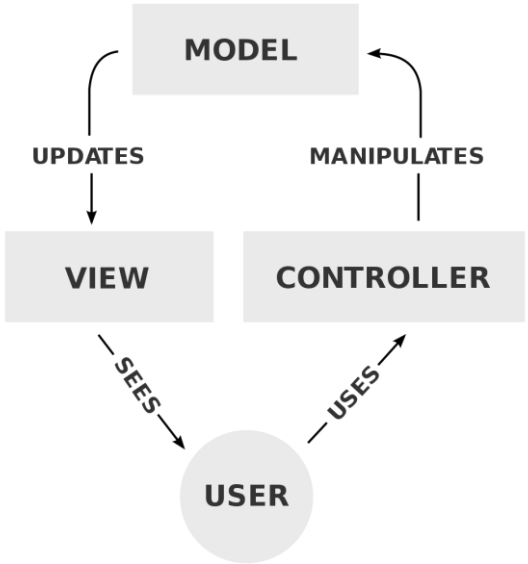
2.1 – Основна Теория

Model–View–Controller (MVC) е софтуерен архитектурен модел, често използван за разработка на потребителски интерфейси, който разделя свързаната програмна логика на три взаимосвързани елемента. Това се прави, за да се отделят вътрешните представяния на информацията от начина, по който тя се представя и приема от потребителя.

Компоненти на MVC:

- **Модел (Model)** – Централният компонент на модела. Това е динамичната структура от данни на приложението, независима от потребителския интерфейс. Той директно управлява данните, логиката и правилата на приложението.

- **Изглед (View)** – Всяко представяне на информация, като графика, диаграма или таблица. Възможни са множество изгледи на една и съща информация, например стълбовидна диаграма за ръководството и табличен изглед за счетоводителите.
- **Контролер (Controller)** – Контролерът е частта от приложението, която обработва взаимодействието с потребителя. Той интерпретира входните данни от потребителя, като информира модела и изгледа да се променят според тях, и ги преобразува в команди за модела или изгледа.



Фиг.2 *Model-View-Controller*

Компютърна графика,дълбоко обучение,концептуални модели,архитектури на системата

Глава 3 - Описание на използваната апаратна / схемна / софтуерна част

3.1 – Система за вход

Системата за вход в приложението е изградена с цел да осигури надеждна автентикация и персонализация на потребителите. Тя е реализирана чрез комбинация от потребителски интерфейс в Streamlit и база данни MongoDB, в

която се съхраняват данните за регистрация.

3.1.1 – Регистрация на потребител

Функционалността започва с процеса по регистрация на нови потребители. При създаване на профил системата извършва серия от проверки: дали паролата съвпада с полето за потвърждение, дали вече съществува потребител със същото име или имейл, както и дали въведените данни отговарят на минимални изисквания.

3.1.2 – Забравена парола

При забравена парола потребителят въвежда регистрирания си имейл и системата генерира временна парола, която автоматично се хешира и записва в базата. Потребителят получава новата парола по имейл чрез интеграция със SMTP сървър на Gmail.

3.1.2 – Забравено потребителско име

При забравено потребителско име се въвежда регистрираният имейл и системата автоматично изпраща на потребителите напомняне за своето потребителско име по електронна поща.

3.2 – Работа с изображения

Модулът за работа с изображения осигурява подготовка на данните преди детекция/сегментация, като покрива три канала за въвеждане и контролирано преоразмеряване. При липса на вход се визуализират изображения (“Input Image” и “Detection Results”), което гарантира предвидимо поведение на интерфейса и улеснява тестването.

3.2.1 – Качване на изображение (Upload an image)

Приложението използва streamlit file_uploader с ограничение по разширения (jpg, png, jpeg, bmp, webp), осигурявайки валиден формат на входа. При качване файлът се отваря с PIL, а за унифициране на обработката се

поддържа BytesIO. Преди всяка последваща операция указателят се връща в началото, за да се избегнат грешки при многократно четене. При неуспех се изписва ясна грешка и процесът се прекъсва.

3.2.2 – Поставяне чрез clipoard base64/URL

Вариантът за поставяне приема чист base64 или копираният адрес на изображението. Логиката детектира формата, декодира съдържанието чрез base64 или изтегля изображението с requests и го капсулира в BytesIO. Невалидни входни данни се отхвърлят с съобщение за грешка, без да се компрометира стабилността на сесията.

3.2.3 – Поставяне чрез clipboard изображение

Чрез бутона „Paste from clipboard“ се улавя изображение от клипборда като data URI. Съдържанието се декодира до бинарен поток и се визуализира незабавно в интерфейса.

3.2.4 – Преоразмеряване на изображение

Преоразмеряването е управлявано от радиобутон „Enabled/Disabled“ в страничната лента и се активира само при налично изображение. При „Enabled“ се извличат оригиналните размери и се инициират контролите за ширина/височина. Самото мащабиране се изпълнява с OpenCV, като се осигурява коректна конверсия между PIL и NumPy. Резултатът се материализира в буфер (BytesIO) и замества текущото изображние, така че следващите стъпки (детекция/сегментация) работят върху размерите подадени от потребителя.

3.3 –Детекция и сегментация

Функцията за детекция и сегментация е ключов елемент в разработената система. Чрез нея приложението позволява на потребителя да обработва изображения, да избира между различни режими на анализ и да визуализира резултатите.

3.3.1 – Избор на модел (Detection / Segmentation)

В приложението е реализиран механизъм за избор на режим на работа, който се осъществява чрез радио бутони в страничното меню. Потребителят може да избере дали да използва стандартен модел за детекция или модел за сегментация. При детекция се извършва разпознаване на обекти чрез ограничаващи рамки около тях. При сегментация позволява по-прецизно отделяне на формата на обектите чрез сегментационни маски.

3.3.2 – Праг на увереност (Confidence Threshold)

Основен параметър при анализа е прагът на увереност, който контролира чувствителността на модела. В интерфейса този параметър се настройва чрез плъзгач със стойности от 0 до 100 процента, които в кода се преобразуват в десетична стойност и се подават към модела. По този начин потребителят може сам да избере дали да работи с по-точни резултати, при които се отчитат само високонадеждни детекции, или да допуска повече разпознавания, но с риск от по-ниска точност.

3.3.3 – Визуализиране на резултати

Резултатите от анализа се представят на потребителя чрез визуализация директно в приложението. В системата първо се показва оригиналното изображение, а след това неговата обработена версия с нанесени рамки или маски върху откритите обекти. Освен това към всеки обект се добавя текстова анотация за класа, което улеснява идентификацията.

3.4 – Запазване в база данни

В системата е критично резултатите да бъдат съхранявани надеждно и структурирано в база данни. Това позволява лесно извличане, анализ, визуализация и проследяване на детекциите във времето. В този случай е

използвана MongoDB, която е документно-ориентирана база, подходяща за съхранение на разнообразни данни с различна структура.

3.4.1 – Структура на документа за детекция

Всеки резултат от детекция се запазва като отделен документ. Документът съдържа метаинформация за изображението и конкретните открити обекти. Основните полета са:

- ***timestamp*** - времето на извършване на детекцията (UTC).
- ***model_type*** – използваният тип модел (например detection или segmentation).
- ***confidence-threshold*** - прагът на увереност, зададен при анализа.
- ***source*** - източникът на изображението (качено, поставено от клипборда или взето от URL).
- ***image_name*** - име на изображението, ако е качено от файл.
- ***image_resolution*** - ширина и височина на изображение.
- ***object_counts*** - броя обекти от всеки клас.
- ***Objects*** - списък с подробни данни за всяка детекция (идентификатор, клас, увереност, координати на bounding box).
- ***image_bytes*** - изображението с нанесени маркировки, съхранено като base64 за директна визуализация.
- ***user_id*** - уникален идентификатор на потребителя, за да може историята да бъде персонализирана.

3.4.1 – Структура на колекциите

За по-добра организация и по-бързо търсене документите се разпределят в различни колекции в зависимост от класа на откритите обекти. Ако е засечен само един клас, например „person“ или „car“, документът се запазва в едноименна колекция. При наличие на повече от един клас детекцията се записва в колекцията **Multiclass_objects**. Това позволява ефективно филтриране – например бързо извличане на всички случаи с автомобили, без да се претърсват

останалите записи. Ако не са засечени обекти, данните се запазват в колекция **no_detections**.

3.5 – История и статистика

Модулът за „История“ и „Статистика“ има за цел да предостави на потребителя възможност не само да наблюдава резултати от извършените детекции, но и да прави количествен и качествен анализ на обектите. Тази част от системата е изградена така, че да съчетава удобство, гъвкавост и яснота, като комбинира таблично представяне, филтриране по критерии и визуализация на изображения от предишни сесии.

3.5.1 –Таблица с обекти и броя им

системата генерира таблица, съдържаща информация за класовете обекти, които са били засечени в рамките на една детекция. Таблицата включва колони за името на класа, броя на срещанията му и средната увереност на модела при разпознаването. Чрез сортиране потребителят може лесно да идентифицира кои обекти се срещат най-често в анализираните изображения.

3.5.2 –Обобщена таблица с обекти и броя им

Системата предоставя информация за всички извършени налични записи. Таблицата включва колони за името на класа, броя на срещанията му и средната увереност на модела при вече разпознати обекти. Обобщената статистика предоставя цялостен поглед върху работата на системата и е полезна за оценка на нейната ефективност при различни сценарии на използване.

3.5.2 – История на детекциите

Достъп до пълната история на извършените детекции за конкретен потребител. Системата съхранява всяка детекция в база данни, като записва дата и час на обработката, използвания модел, избрания източник на изображение и

зададения праг на увереност. Освен това, всеки запис съдържа информация за засечените класове и техния брой. Възможни са филтри по класове и източници, което улеснява преглеждането на историята според конкретни критерии.

3.5.3 – Визуализация на изображения от минали детекции

Освен табличните данни и текстовата информация, системата предоставя и визуализация на изображенията от предишни детекции. Към всеки запис се съхранява изображение с нанесени върху него резултати от детекцията – рамки около откритите обекти и техните етикети. Потребителят може да разгледа тези изображения директно в интерфейса, като по този начин получава по-ясна представа за качеството на разпознаването и точността на модела в реални ситуации. Тази визуализация служи не само като удобен начин за проверка, но и като доказателствен материал за извършените анализи.

3.2 – Python

Python е език за програмиране от високо ниво с общо предназначение. Неговото проектиране набляга на четимостта на кода с използването на значителни отстъпи. Python е избран като основен инструмент. Използван е в няколко основни направления: за създаване на потребителска автентикация и управление на сесии, за обработка на изображения в реално време, за работа с база данни с цел съхраняване на резултати от детекции, както и за статистически анализ и визуализация. Причина за това е неговата гъвкавост, лекота на употреба и широката поддръжка на библиотеки, които улесняват интеграцията между различни компоненти – база данни, сървърна логика и обработка на

изображения. За разлика от други езици като C или Pascal, Python предлага значително по-бързо време за прототипиране и по-добра четимост на кода, което е от ключово значение при разработка на комплексни приложения в ограничени времеви рамки.

2.3 – Streamlit

Streamlit е Python библиотека с отворен код, която улеснява създаването и споделянето на персонализирани уеб приложения. Streamlit е използван за изграждане на цялостния потребителски интерфейс – регистрация и вход на потребители, настройка на параметри за детекция, качване на изображения и преглед на резултатите. Допълнително, чрез него се реализират модули за преглед на историята на детекциите, както и статистическа обработка на събраните данни. Streamlit е избран, тъй като той предоставя бързо и ефективно решение за разработка, прототипиране и тестване на системата която е базирана на Python, като същевременно осигурява достатъчно функционалности за крайния потребител. За разлика от класически уеб framework като *Django* или *Flask*, които изискват ръчна интеграция с HTML, CSS и JavaScript, Streamlit предлага високо ниво на абстракция и позволява директно изграждане на динамични интерфейси чрез Python код. Това прави процеса на разработка значително по-ускорен и подходящ за системи, в които потребителското взаимодействие е пряко свързано с обработка на данни и визуализация.

2.4 – OpenCV

OpenCV (Open Source Computer Vision Library) е библиотека с отворен код за компютърно зрение и машинно обучение. OpenCV се използва като подпомагащ инструмент за интеграция с модела за детекция и сегментация. След като се извърши разпознаване на обектите, библиотеката служи за последваща обработка на изображенията: преоразмеряване, конвертиране между цветови пространства, визуализация на резултатите и съхранение на крайния файл. Освен това чрез OpenCV се реализира и динамично преоразмеряване на изображенията

в Streamlit интерфейса, което позволява на потребителя да избира желаната резолюция преди стартиране на детекцията. Основната причина да бъде предпочетена пред други библиотеки е нейната оптимизация за работа с изображения в реално време.

2.5 – MongoDB

MongoDB е база от данни, в която може да се управлява, да се съхранява или извлича документно-ориентирана информация. Решението да се използва именно MongoDB пред алтернативи като MySQL или PostgreSQL е поради спецификата на съхраняваната информация. Данните, които системата генерира, включват динамични JSON структури – резултати от детекция с различен брой обекти, класове, координати и метаданни за изображенията. MongoDB, като NoSQL документно-ориентирана база данни, е изключително подходяща за съхранение на данни с гъвкава и изменяща се структура. Класическите релационни бази данни предполагат предварително дефинирана схема, което ограничава добавянето на нови атрибути без промяна на цялата база. В контекста на нашата система това би довело до затруднения при интеграция на нови модели за детекция или допълнителни характеристики.

С MongoDB можем да съхраняваме резултатите в JSON-подобни BSON документи**,** които позволяват лесно добавяне на нови полета без нарушаване на съществуващите данни.

2.6 – MongoDB Compass

Освен самата база данни MongoDB, е използван и MongoDB Compass който е официалният графичен интерфейс за управление на базата данни. MongoDB Compass предоставя визуален достъп до документно-ориентираната база, което значително улеснява работата с нея. Вместо да се използват само конзолни команди, потребителят получава удобен графичен изглед за данните, индексирането и структурата на колекциите. Това позволява бързо валидиране на коректността на записите – например дали резултатите детекциите съдържат

очакваните класове, координати и праг на увереност. По този начин Compass се използва като инструмент за верификация и отстраняване на грешки при работата с базата.

2.7 – PyMongo

За да се осъществи връзката между Python приложението и базата данни MongoDB, е избрана библиотеката PyMongo. Тя представлява официалния Python драйвер за MongoDB, който предоставя пълен достъп до всички функции на базата чрез лесен за употреба API. PyMongo се използва за управление на потребителски данни и детекции, съхранявани в MongoDB. Чрез него се реализират основните операции: добавяне на нови потребители с криптирани пароли, записване на резултатите от детекциите, извличане на история на предишни детекции и анализ на данните. Благодарение на PyMongo тези процеси са интегрирани директно в Python кода на приложението, без необходимост от външни инструменти.

2.8 – bcrypt

Bcrypt е криптографски алгоритъм, създаден специално за хеширане на пароли, като неговата основна сила произтича от две ключови характеристики – добавянето на случайна стойност, наречена "сол" (salt), и възможността за настройка на изчислителната сложност чрез броя на итерациите. Солта предотвратява използването на т.нар. rainbow таблици, тъй като прави всяка парола уникална дори ако потребителите въвеждат една и съща стойност. bcrypt се използва като основен механизъм за защита на пароли в приложението. При регистрация на нов потребител неговата парола се хешира с bcrypt и полученият хеш се съхранява в базата от данни MongoDB. Така оригиналната парола никога не се записва в системата, което елиминира риска от директно изтичане. При вход в системата, въведената от потребителя парола отново се хешира и резултатът се сравнява с вече съхранения хеш. Ако стойностите съвпадат, достъпът се разрешава.

2.9 – Pandas

Pandas е една от най-популярните библиотеки в Python за работа с таблични данни. Изборът на Pandas пред други библиотеки се дължи на няколко ключови предимства. Първо, Pandas комбинира удобството на високо ниво операции с ефективността на оптимизираните алгоритми за обработка на големи масиви от данни. Второ, библиотеката предоставя вградени функции за обработка на липсващи стойности, групиране, обединяване на таблици и филтриране, което улеснява бързото изграждане на аналитични решения. Pandas се използва като междинен слой за обработка и структуриране на данни, генерирани от модела за детекция. Всеки резултат от обектна детекция – включващ информация за клас на обект, координати на ограничителни рамки и ниво на увереност. Те се записва първоначално в MongoDB. След това тези данни се извличат и трансформират в Pandas DataFrame, което позволява по-нататъшни операции като статистически анализ, филтрация и агрегиране по класове. В допълнение, Pandas се използва и за експортиране на резултатите в по-удобни формати (например CSV), което улеснява както визуализацията, така и последващата обработка от страна на потребителя.

2.10 – PIL (Python Imaging Library)

PIL представлява една от най-разпространените библиотеки за обработка на изображения в Python, която добавя поддръжка за отваряне, манипулиране и запазване на много различни формати на файлове с изображения.PIL се използва за подготовка на изображенията, които впоследствие се подават към алгоритмите за детекция на обекти. Например, когато потребителят качи снимка или я постави от клипборда, тя първо се зарежда с помощта на PIL, което гарантира коректното ѝ декодиране независимо от формата. Изборът на PIL пред други решения е мотивиран именно от тази гъвкавост и интеграция с останалите библиотеки в проекта. Докато OpenCV също е използван за определени операции, PIL играе ключова роля като междинен слой между входа на потребителя и модела, осигурявайки стабилност и лесна обработка на изображенията. Така се гарантира

надеждност на целия процес и удобство за бъдещо разширяване на системата.

Литературни източници:

1. Girshick, R. *"Rich feature hierarchies for accurate object detection and semantic segmentation"* (R-CNN). CVPR 2014.

2. Girshick, R. *"Fast R-CNN"*. ICCV 2015.

3. Ren, S., He, K., Girshick, R., Sun, J. *"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"*. NeurIPS 2015.

4. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. *"You Only Look Once: Unified, Real-Time Object Detection"*. CVPR 2016

5. Redmon, J., Farhadi, A. *"YOLO9000: Better, Faster, Stronger"*. CVPR 2017.

6. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.

7. Liu, W., Anguelov, D., Erhan, D. et al. *"SSD: Single Shot MultiBox Detector"*. ECCV 2016.

8. Lin, T.Y. et al. *"Microsoft COCO: Common Objects in Context"*. ECCV 2014.

9. Deng, J. et al. *"ImageNet: A large-scale hierarchical image database"*. CVPR 2009.

10. Zhao, Z.Q. et al. *"Object Detection with Deep Learning: A Review."* IEEE Transactions on Neural Networks and Learning Systems, 2019.

11. Zou, Z. et al. *"Object Detection in 20 Years: A Survey."* Proceedings of the IEEE, 2019.

12.https://arxiv.org/html/2412.05252v1#S5 - таблица 1

13. https://docs.ultralytics.com/models/yolo11