



ДИПЛОМНА РАБОТА

Тема:

Проектиране и разработване на система за разпознаване на обекти в изображение

Студент: Любомир Руменов Ламбрев

Факултетен номер: 614973

Специалност: Компютърни системи и технологии, ОКС Магистър

Научен ръководител: гл. ас. д-р Веселка Петрова-Димитрова

Консултант: д-р Ваня Маркова

Катедра: Компютърни системи и технологии

Съдържание

Увод	4
Глава 1 – Обзор.....	5
1.1 - Състояние на проблема	5
1.2 - Преглед на уеб-базирана система за машинно зрение.....	6
1.3 - Съществуващи системи за детекция и сегментация на обекти.....	7
1.3.1 – Ultralytics YOLO	9
1.3.2 - SSD (Single Shot MultiBox Detector).....	10
1.3.3 - Faster R-CNN	11
Глава 2 - Теоретично решение на поставената задача	13
2.1 – Компютърно зрение (Computer Vision).....	13
2.1.1 – Филтри в обработката на изображения	14
2.2 – Дълбоко обучение (Deep Learning)	18
2.2.1 – Разпознаване на образи в дълбоко обучение	19
2.3 – Концептуален модел на подсистемата за обучение	21
2.4 – Концептуален модел на подсистемата за разпознаване на обекти	24
2.5 – Архитектура на системата	26
2.5.1 – Архитектура на софтуерната система.....	29
2.5.2 – Концептуален модел на системата за потребителя	31
2.6 – Софтуерни инструменти и библиотеки	33
2.6.1 – Python.....	33
2.6.2 – Streamlit	33
2.6.3 – OpenCV.....	34
2.6.4 – MongoDB	34
2.6.5 – MongoDB Compass.....	35
2.6.6 – PyMongo	35
2.6.7 – Bcrypt	36
2.6.8 – Pandas	36
2.6.9 – PIL (Python Imaging Library)	37
2.6.10 – Datetime	37
2.6.11 – UUID.....	38
2.6.12 – Streamlit-cookie-manager	38
2.6.13 – St_img_pastebutton.....	39
2.6.14 – SMTPlib	39

Глава 3 - Описание на използваната апаратна / схемна / софтуерна част	40
3.1 – Псевдокод на подсистемата за обучение	40
3.2 – Псевдокод на подсистемата за разпознаване на обекти	47
3.3 – Експеримент	54
3.3.1 – Резултати.....	54
Глава 4 – Изчислителна част/проектиране на блок схеми на алгоритми за софтуерната част/функционално тестване	57
4.1 – Система за вход	57
4.1.1 – Регистрация на потребител.....	57
4.1.2 – Забравена парола	58
4.1.3 – Забравено потребителско име.....	58
4.2 – Промяна на парола	59
4.3 – Промяна на имейл	59
4.4 – Работа с изображения.....	60
4.4.1 – Качване на изображение (Upload an image)	60
4.4.2 – Поставяне чрез clipboard base64/URL	61
4.4.3 – Поставяне чрез clipboard изображение	61
4.4.4 – Преоразмеряване на изображение	62
4.5 – Детекция и сегментация	62
4.5.1 – Избор на модел (Detection / Segmentation).....	62
4.5.2 – Праг на увереност (Confidence Threshold).....	63
4.5.3 – Визуализиране на резултати	63
4.6 – Запазване в база данни.....	64
4.6.1 – Структура на документа за детекция.....	64
4.6.2 – Структура на колекциите.....	65
4.7 – История и статистика	66
4.7.1 – Таблица с обекти и броя им	66
4.7.2 – Обобщена таблица с обекти и броя им.....	67
4.7.3 – История на детекциите.....	68
4.7.4 – Визуализация на изображения от минали детекции	68
Глава 5 – Приложимост на дипломната работа.....	69
Глава 7 – Изводи и претенции за самостоятелно получени резултати	70
Литературни източници	71
Приложения	72

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 е разработена 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

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 видео потоци. 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[14,11].

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 [14].

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 [5,7,13]. 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 [10,11,12].

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 [15].

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 [14].

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. Стремeжът е насочен 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 пригодност 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 [11]. SSD заема междинно място, като съчетава добра скорост и точност, както 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)	Image Net (mAP)	Open Images (mAP)	Inference Speed (FPS)	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 [16]

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 видео потока 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)**

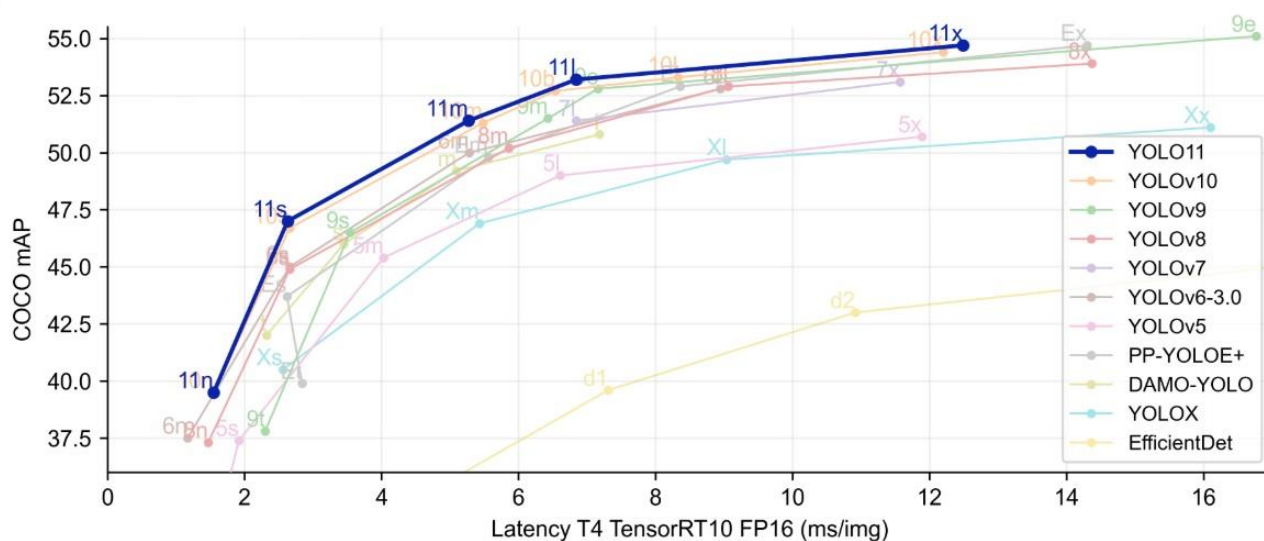
Моделът постига значителни подобрения по отношение на точност, скорост и изчислителна ефективност, в сравнение спрямо предходните модели.

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 [17]

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 [12].

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 [10]. 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 [16]. 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 [13]. 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 определи 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 [7]. 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 – Компютърно зрение (Computer Vision)

Компютърното зрение е научна и приложна област, която изучава методите за автоматизирано възприемане и интерпретация на изображения от компютър. По аналогия с човешкото зрение, където процесът включва формиране на зрителен образ в очите и осъзнаване на видяното в мозъка, в компютърното зрение основната задача е регистриране на изображения и извличане на информация от тях. Системите могат да бъдат изградени в два основни варианта – интерактивни, при които част от решенията се вземат от потребителя, и напълно автоматизирани, при които крайните решения се правят от машината. Независимо от вида, процесът на обучение преминава през етапи на работа с регистрирани изображения, обработка за подчертаване на специфични особености, извличане на признаци и избор на методи за класификация. В резултат се създава необходимият набор от апаратни и алгоритмични средства за функциониране на системата [2].

Фундаменталните основи на компютърното зрение могат да се обобщят в няколко направления. На първо място, то разчита на преобразуване на оптични сигнали в електрически, последвано от цифрова обработка на двумерни сигнали, т.е. изображения. Към това се добавя статистическа и вероятностна обработка на данни, свързани с конкретния обект на изследване [2]. Така се осигурява възможност за извличане на информация и надеждно разпознаване на обекти в различни условия на регистрация.

В практическо отношение всяка система за компютърно зрение започва със създаване на цифрово изображение. Това най-често се реализира с помощта на камера, която има четири основни елемента: оптична система, фото-приемна матрица, електронна обработваща част и интерфейс за предаване на данните към компютър. Камерите могат да се класифицират според различни признаци – по спектрален диапазон, по конструкция на матрицата, по метод за сканиране или

според начина на управление [2].

Най-разпространеното цветово представяне е RGB моделът, където всеки пиксел съдържа стойности за червено (Red), зелено (Green) и синьо (Blue). Съществуват и други цветови пространства, като HSV (Hue, Saturation, Value) или YCbCr, които често се използват за специфични задачи поради по-добра съвместимост с човешкото възприятие. За да се съхраняват и обменят изображения, са разработени различни файлови формати. Сред най-популярните са:

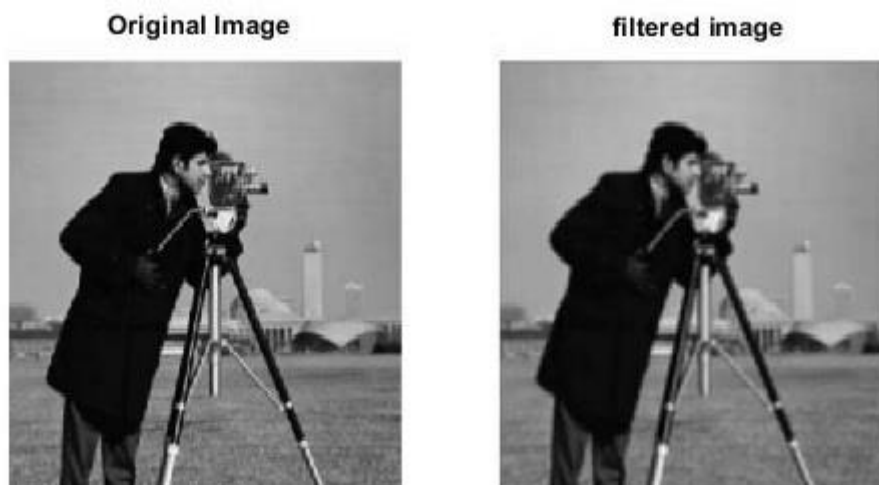
- **BMP** – базов формат, който запазва пикселите без компресия, подходящ за обработка, но с голям размер;
- **JPEG** – широко използван компресиран формат, който намалява размера чрез загуба на част от информацията;
- **PNG** – компресиран формат без загуба на качество, често използван в уеб приложения;
- **TIFF** – гъвкав формат, често използван в научни изследвания и медицинско изображение, заради поддръжката на високо качество и различни дълбочини на цвета.

2.1.1 – Филтри в обработката на изображения

Филтрирането на изображения е една от най-важните операции в компютърното зрение и цифровата обработка. То представлява прилагане на математически операции върху пикселите с цел подобряване на качеството на изображението, извличане на важни характеристики или потискане на шум. В основата на филтрирането стои идеята за *маска* (kernel), която се „плъзга“ върху изображението и модифицира стойностите на пикселите според предварително дефинирано правило [1].

Най-простата група са линейните филтри, които изчисляват новата стойност на даден пиксел като линейна комбинация на съседните му пиксели.

Пример за такъв филтър е осредняващият филтър, който изглажда изображението чрез замяна на стойността на пиксела със средната стойност на неговото обкръжение. Този подход е ефективен за премахване на случаен шум, но води и до замъгляване на ръбовете. Показано на фиг. 2.



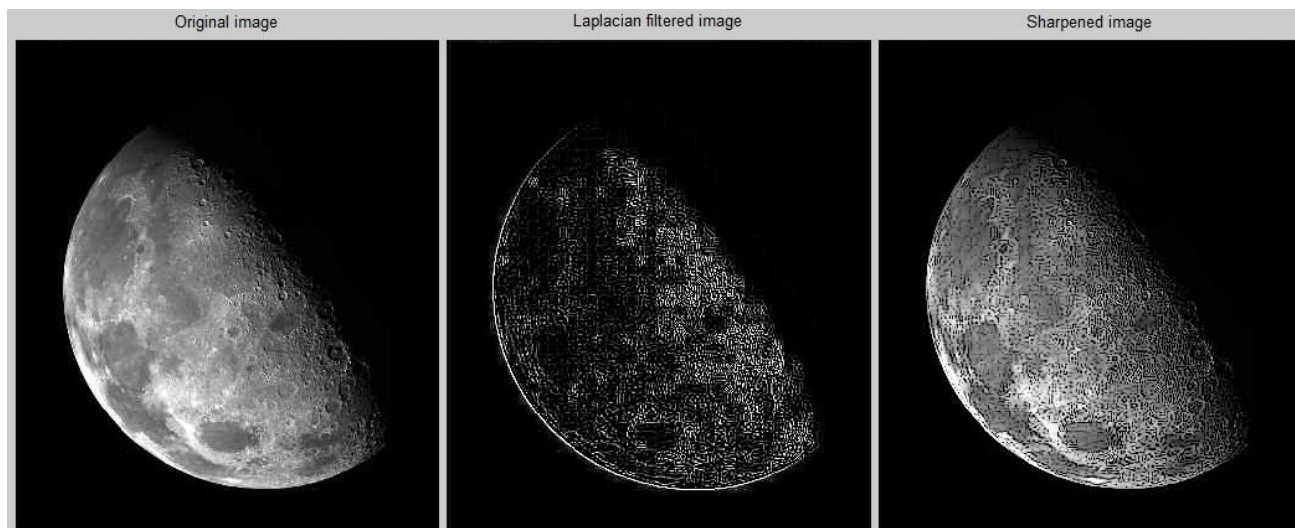
Фиг. 2 Визуално представяне на изображение след осредняващия филтър [4]

Друг често използван е Гаусовият филтър, който също има за цел изглаждане, но за разлика от простото усредняване, използва нормално разпределение за теглата в маската. Така по-близките пиксели оказват по-голямо влияние от по-далечните, което позволява по-естествено и контролирано замъгляване (фиг. 3). Гаусовите филтри са особено важни при предварителна обработка преди алгоритми за откриване на ръбове или сегментация.



Фиг. 3 Визуално представяне след Гаусова филтрация на изображение [4]

Освен за изглаждане, филтрите се използват и за изостряне. Пример е Лапласовият филтър, който подчертава ръбовете в изображението, като изчислява втората производна на яркостта. Така се акцентира върху области със силна промяна, но същевременно се усилюва и шумът, което изисква комбинация с изглаждащи методи. Показано на фиг. 4.



Фиг. 4 Визуално представяне след Лапласова филтрация на изображение [4]

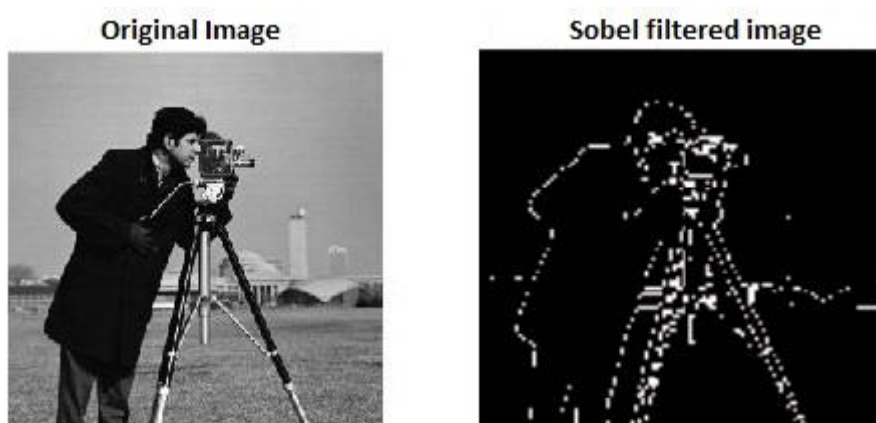
Към нелинейните филтри спада медианният филтър, който заменя стойността на пиксела с медианата от неговите съседни пиксели. За разлика от линейните методи, медианният филтър е особено ефективен при премахване на импулсен („salt & pepper“) шум, без да замъглява толкова силно ръбовете (фиг. 5). Това го прави широко използван в медицинската обработка и в системи за разпознаване на обекти.



Фиг. 5 Визуално представяне след медианна филтрация на изображение [4]

Филтрите намират изключително широко приложение при решаването на практически задачи в компютърното зрение. Те не само подобряват качеството на изображенията, но и служат като инструмент за извличане на структурна информация, необходима за по-сложни алгоритми.

Една от класическите задачи е откриването на ръбове. Ръбовете представляват граници между области с различна яркост или цвят и съдържат ключова информация за формата и структурата на обектите. За тяхното намиране се използват специализирани филтри като операторът на Собел (фиг 6). Операторът на Собел изчислява първата производна по хоризонтала и вертикала и комбинира резултатите, за да подчертае областите с рязка промяна в интензитета.



Фиг. 6 Визуално представяне след филтрация на изображение с оператор на Собел [4]

Друга важна област е сегментацията на изображения, при която филтрите се използват за разделяне на изображението на смислово значими области – например фон и обекти. Чрез изглаждащи филтри се намалява шумът, който може да доведе до неправилна сегментация. В комбинация с прагови техники или алгоритми за групиране, филтрирането позволява по-точно отделяне на обектите от околната среда.

Филтрите са незаменими и при потискане на шум, особено в ситуации, където изображенията са получени при неблагоприятни условия като слаба

светлина, атмосферни влияния или технически ограничения на сензора. Гаусовите и медианните филтри са стандартни подходи за премахване на шум, но съвременните методи включват и адаптивни филтри, които се настройват според локалните характеристики на изображението [9].

В заключение, компютърното зрение обединява хардуерни и софтуерни технологии, които имат за цел да направят машините способни да възприемат, анализират и разпознават обекти в заобикалящия ги свят по начин, наподобяващ човешкото зрение.

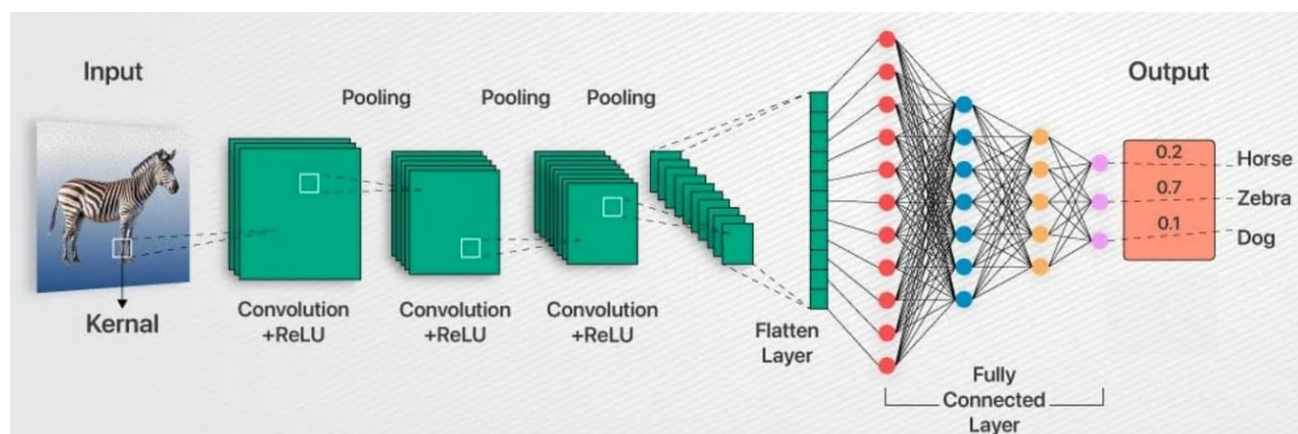
2.2 – Дълбоко обучение (Deep Learning)

Дълбокото обучение е направление в изкуствения интелект, което използва многослойни изкуствени невронни мрежи за извличане и анализ на сложни зависимости в данните. Неговата сила се състои в способността автоматично да се откриват йерархии от признаци, без необходимост от предварително ръчно дефиниране на характеристики, както е при класическите подходи. Това го прави особено ефективен метод при задачи, свързани с разпознаване на изображения и откриване на обекти [8].

Невронните мрежи, на които се основава дълбокото обучение, се състоят от слоеве взаимно свързани изчислителни единици. Всеки неврон приема вход, извършва трансформация чрез активационна функция и предава резултат към следващ слой. Натрупването на такива слоеве позволява на мрежата да извлича все по-абстрактни и устойчиви признаци, което води до надеждни класификации и детекции.

Особено важен клас архитектури са конволюционните невронни мрежи (CNN). Те са създадени специално за обработка на двумерни данни като изображения и комбинират процесите по извличане на признаци и обучение в една обща структура. Основните операции включват конволюция за откриване на локални особености, нелинейни функции (ReLU) за въвеждане на сложност, обединяване (Pooling) за редуциране на размерността и напълно свързани слоеве

за крайна класификация [8]. Показано на фиг. 7.



Фиг. 7 Обща структура на конволюционните невронни мрежи [18]

Обучението на такива модели обикновено е с "учител". При него всяко входно изображение е съпроводено от известен очакван резултат (етикет), което позволява сравнение между предсказанието на мрежата и реалната стойност. Разликата се използва за корекция на теглата чрез алгоритъма за обратно разпространение на грешката. С всяка итерация точността се повишава, докато моделът не постигне стабилни резултати.

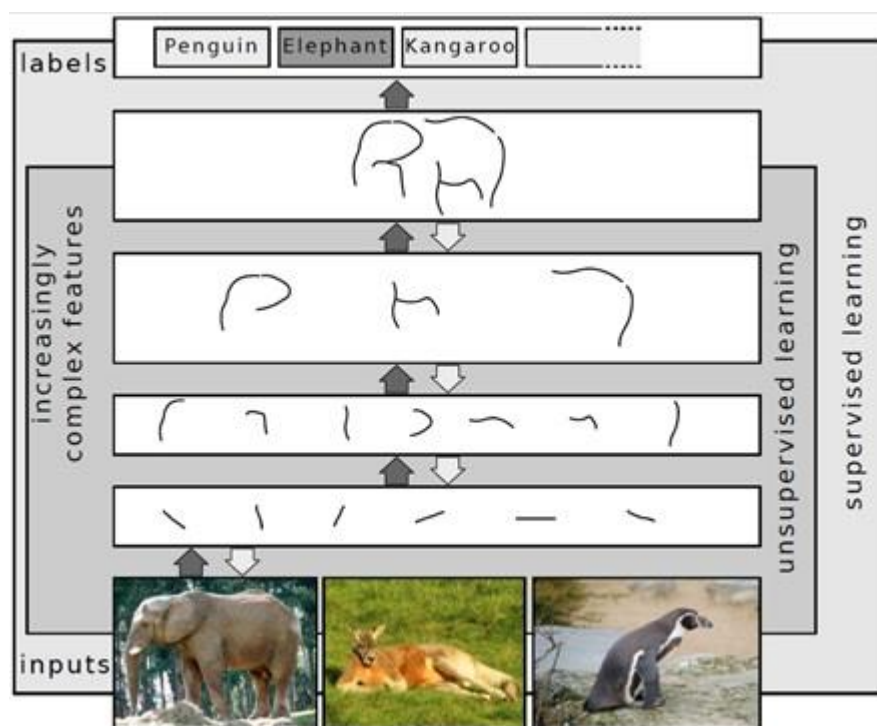
Приложенията на дълбокото обучение в компютърното зрение са изключително разнообразни – от разпознаване на пръстови отпечатъци до лицева идентификация. Характерно е, че веднъж обучени върху големи набори от данни, моделите могат да бъдат адаптирани и към нови задачи чрез техники като трансферно обучение.

В заключение, дълбокото обучение може да се определи като един от ключовите фактори за успеха на съвременните системи за компютърно зрение, тъй като съчетава висока точност, адаптивност и способност за автоматично извличане на признаци от сложни данни.

2.2.1 – Разпознаване на образи в дълбоко обучение

Под „разпознаване“ се разбира процесът, при който система, обучена върху масиви от данни, получава ново изображение и автоматично определя какъв обект

съдържа то, както и неговите характеристики. Процесът на разпознаване обикновено започва с предварителна обработка на изображението – нормализация, промяна на размера и понякога отстраняване на шумове. След това данните се подават към дълбока невронна мрежа. Тези мрежи имат способността да извличат локални и глобални признаци директно от пикселната структура на изображението, като в по-ранните слоеве се улавят прости особености (ръбове, линии, цветови контрасти), а в по-дълбоките – по-сложни структури и абстракции (части от обекти, форми, контури) както е показано в фиг. 8.



Фиг. 8 Представяне на изображения на множество слоеве на абстракция в дълбокото обучение [19]

Ключов момент в обучението на мрежите за разпознаване на образи е алгоритъмът за обратно разпространение на грешката. При всяка итерация системата сравнява своето предсказание със зададения етикет и изчислява грешка (loss). Тази грешка се използва за актуализиране на теглата по слоевете, така че при следващи итерации моделът да се приближава към правилния резултат. По този начин системата се самоусъвършенства, докато достигне стабилно ниво на точност [3].

Разпознаването на образи чрез дълбоко обучение не се ограничава само до класификация на цели изображения. При локализацията задачата е да се открият координатите на конкретен обект в изображението и да се обособи неговото местоположение чрез ограничителна кутия (bounding box). При сегментацията обаче изискването е още по-високо – да се идентифицират пикселите, които принадлежат на даден обект, като по този начин се разпознава неговата форма и граници.

Въпреки впечатляващите резултати, дълбокото обучение при разпознаване на образи среща и предизвикателства. Моделите са силно зависими от количеството и качеството на данните, като при недостатъчно разнообразие могат да проявят склонност към грешки. Освен това, необходимостта от значителни изчислителни ресурси поставя ограничения за тяхното приложение на устройства с ограничени възможности.

2.3 – Концептуален модел на подсистемата за обучение

Концептуалният модел е представяне на системата. Той се състои от концепции, използвани за разбиране или симулиране на процеса на системата. Концептуалният модел на подсистемата за обучение на модела е показан на фиг.9.

В първата стъпка – **Подготовка на данните** – изображенията се подготвят и техните анотации, които описват местоположението и класа на обектите в изображенията. Тези данни обикновено са в стандартен формат (например xml или txt файлове, генерирани от инструменти като LabelImg).

В следващата стъпка – **Въвеждане на изображения и анотации** - Съответните изображения се въвеждат със съответните анотации в подсистемата.

В следващата стъпка – **Анализ и корекции на изображения** - Подсистемата превръща анотациите в унифицирани структури (таблицы/датафреймове), съдържащи информация за координатите на ограничителните кутии (bounding boxes) и съответните класове. На този етап се

извършва и нормализация на координатите спрямо размерите на всяко изображение. Така се постига мащабна независимост, позволяваща моделът да работи върху изображения с различни размери. Допълнително изображенията се преобразуват чрез операции като промяна на размер (resize) , нормализиране на пикселните стойности в диапазон $[0,1]$ и други операции. Като тези стъпки гарантират, че входните данни са в стандартизирана форма, подходяща за подаване към невронна мрежа.

В следващата стъпка – **Изграждане на обучаваща среда** – включва структуриране на данните в тренировъчни и валидационни множества. Това разделение има за цел да се гарантира, че моделът не само „запаметява“ данните, но и придобива способност за обобщаване върху нови, невиджани примери. Обикновено тренировъчният дял е по-голям, докато валидационният се използва за следене на процеса и предотвратяване на пренастройване (overfitting). Данните в тези множества се организират като потоци (datasets), които се подават на малки пакети (batch). Това позволява оптимално използване на паметта и ускорява изчисленията, като едновременно с това осигурява статистическа стабилност на обучението.

В следващата стъпка – **Процес на обучение** - подготвените изображения и техните анотации се подават към невронната мрежа. Може да се разгледа като серия от трансформации върху данните:

- **Формиране на входни признаци** – всяко изображение, вече нормализирано и оразмерено, постъпва във входния слой на модела под формата на числови стойности. Тези стойности съдържат информация за интензитета на пикселите и формират „суровите признаци“ (raw features), които мрежата трябва да обработи.
- **Извличане на признаци** – конволюционните слоеве, които се намират в тялото на модела, действат като автоматизирани филтри. Те преобразуват входните данни в по-абстрактни представяния, идентифицирайки ръбове, текстури и по-сложни структури. Така

мрежата изгражда вътрешна йерархия на признаците, която е подходяща за задачата по локализация и класификация на обекти.

- **Разклоняване на изходи** – архитектурата на модела съдържа две паралелни „глави“. Първата отговаря за регресията на координатите на ограничителните кутии (bounding box regression), докато втората реализира класификацията на обекта. Това разделяне гарантира, че моделът може едновременно да прогнозира къде се намира обектът и какъв клас представлява.

Обучението се осъществява чрез функция на загуба, която измерва разликата между предсказанията и истинските стойности. По време на обучението системата непрекъснато сравнява резултатите върху тренировъчния и валидационния набор. Това служи като механизъм за обратна връзка, който помага да се идентифицират признаци на пренастройване или недообучаване. Допълнителни техники като ранно спиране (early stopping) подsigуряват, че процесът ще бъде прекратен, когато се достигне оптимален баланс между точност и обобщаваща способност.

След завършване на основния цикъл на обучение, подсистемата преминава към **оценяване и анализ на резултатите**. Тази стъпка има за цел да даде отговор на два ключови въпроса: доколко моделът е научил връзката между изображенията и техните анотации и доколко е способен да се справя с нови данни. Оценяването започва с подаване на валидационни или тестови данни, които не са били използвани в процеса на обучение. Моделът генерира изходи – предсказани координати на ограничителните кутии и вероятности за принадлежност към даден клас. Тези изходи се сравняват със съответните истини (Анотирани изображения).

На последно място стои **съхранението и интеграцията на обучен модел**. Това означава, че параметрите на мрежата (тегла, архитектурна структура) се запазват във файл, който може да бъде използван от други подсистеми. Така

обучаващата подсистема е завършена и моделът вече може да се прилага в реални приложения.



Фиг. 9 Концептуален модел на подсистемата за обучение

2.4 – Концептуален модел на подсистемата за разпознаване на обекти

Концептуалният модел на подсистемата за разпознаване на обекти е показан на фиг. 10.

първата стъпка – **Въвеждане на данни** – Системата получава изображение в стандартен цифров формат (например JPEG или PNG), което представлява двумерна матрица от пикселни стойности. Тези стойности отразяват интензитета на цветовете и светлината в сцената.

В следващата стъпка - **Предварителна обработка на изображението** - Извършват се няколко трансформации, които имат за цел да подготвят данните за

подаване към модела. Типични операции са промяна на размера (resize) до фиксирани входни размери, нормализация на пикселните стойности в диапазон $[0,1]$ или $[-1,1]$, както и евентуално конвертиране в определен брой канали (например RGB или grayscale). Това преобразува данните от „неструктурирани“ пикселни стойности в унифицирана форма, съвместима с входния слой на невронната мрежа.

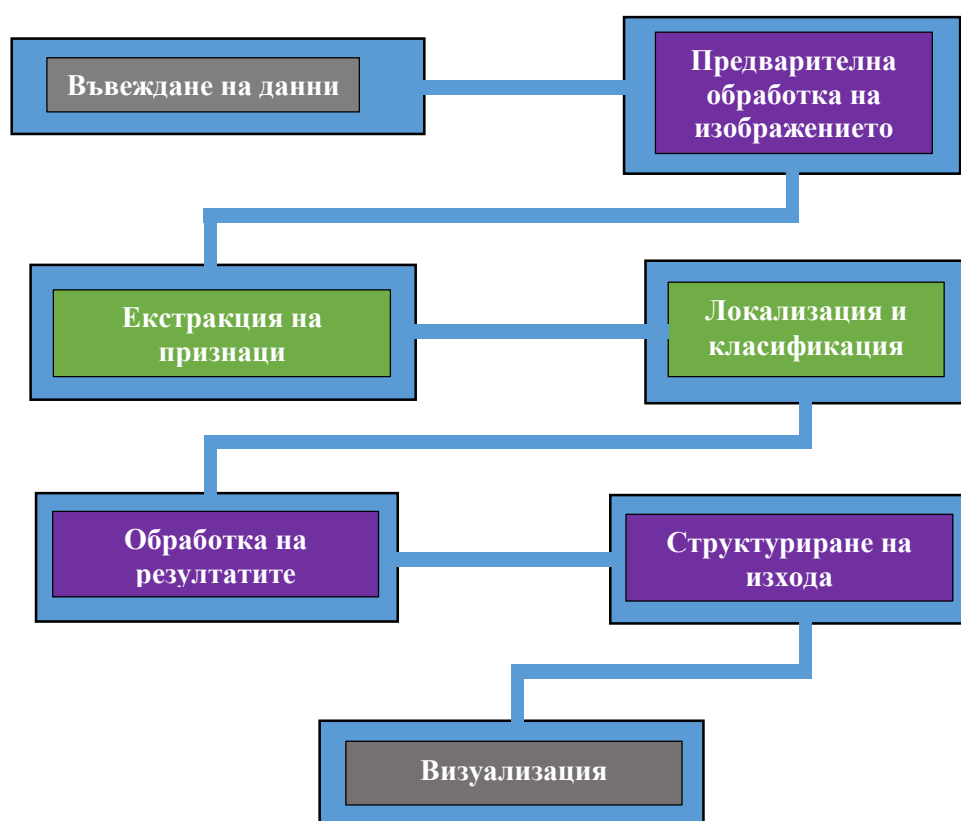
В следващата стъпка – **Екстракция на признаци** - Изображението се подава към конволюционна невронна мрежа, която чрез серия от филтри и нелинейни трансформации извлича релевантни характеристики – ръбове, текстури, контури и по-сложни структури. Може да се разглежда като автоматично „превеждане“ на изображението в многомерно пространство от признаци, където обектите могат да бъдат по-лесно разграничавани.

В следващата стъпка – **Локализация и класификация** - Подсистемата генерира хипотези за потенциалните обекти в изображението под формата на ограничителни кутии (bounding boxes). Всяка област получава предсказание за принадлежност към определен клас и вероятност (confidence score). Така данните се преобразуват от непрекъснато поле от пиксели в дискретен набор от структурирани описания: *обект – координати – клас – вероятност*.

В следващата стъпка – **Обработка на резултатите** - Подсистемата прилага методи за филтриране и оптимизиране на прогнозите, за да се получи по-надеждна интерпретация. Най-често използван подход е *Non-Maximum Suppression (NMS)*, при който се премахват излишните предсказани кутии, които се припокриват в значителна степен. Това означава, че системата редуцира множеството от хипотези до малък набор от уникални и най-вероятни обекти.

В следващата стъпка – **Структуриране на изхода** - Вече филтрираните предсказания се подреждат в унифициран формат. За всеки разпознат обект се запазват координатите на ограничителната кутия, името или индексът на класа и стойността на увереност.

В следващата стъпка – **Визуализация** – Подсистемата превежда числовите резултати в човешки-разбираема форма. Върху изображението се изчертават ограничителни кутии около обектите и се изписват предсказаните класове. Така се получава визуален слой, който позволява бърза оценка на качеството на системата.



Фиг. 10 Концептуален модел на системата за разпознаване на обекти

2.5 – Архитектура на системата

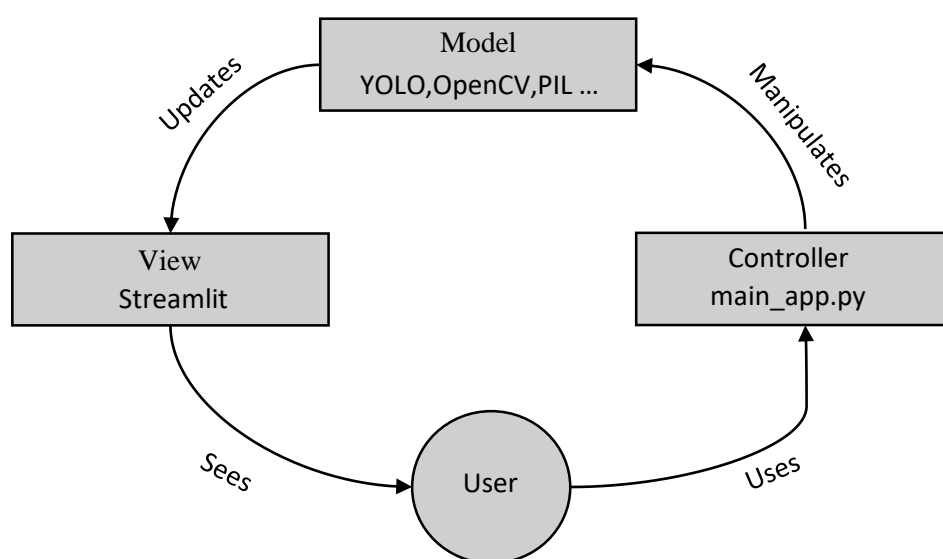
Model–View–Controller (MVC) е софтуерен архитектурен модел, често използван за разработка на потребителски интерфейси, който разделя свързаната програмна логика на три взаимосвързани елемента. Това се прави, за да се отделят вътрешните представяния на информацията от начина, по който тя се представя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

В контекста на тази система MVC е показан на фиг. 11. В **модела** се събират всички инструменти, свързани с данните и тяхната обработка:

- **MongoDB (pymongo)** - използва се за съхранение на информация за потребители и за съхранение на резултатите от разпознавания.
- **YOLO** - това е основният инструмент за разпознаване на обекти. Зареждат се обучени модели, които при подаване на изображение връщат bounding boxes, класове и confidence стойности.
- **Tensorflow** - инструмент за разпознаване на обекти. Зареждат се обучени модели, които при подаване на изображение връщат bounding boxes.

- **OpenCV (cv2)** - участват в обработката на изображенията преди и след разпознаване. Служи за преоразмеряване и работа с масиви от пиксели.
- **PIL (Pillow)** - се използва за отваряне, преоразмеряване и конвертиране в base64.
- **Bcrypt** - осигурява хеширане и проверка на пароли при регистрация и логин.
- **Pandas и Numpy** - служат за представяне на статистически данни и подготовка на таблици, които после се показват в Streamlit.

В **изгледа** основната роля е да представя интерфейса и визуализацията на данни за потребителя:

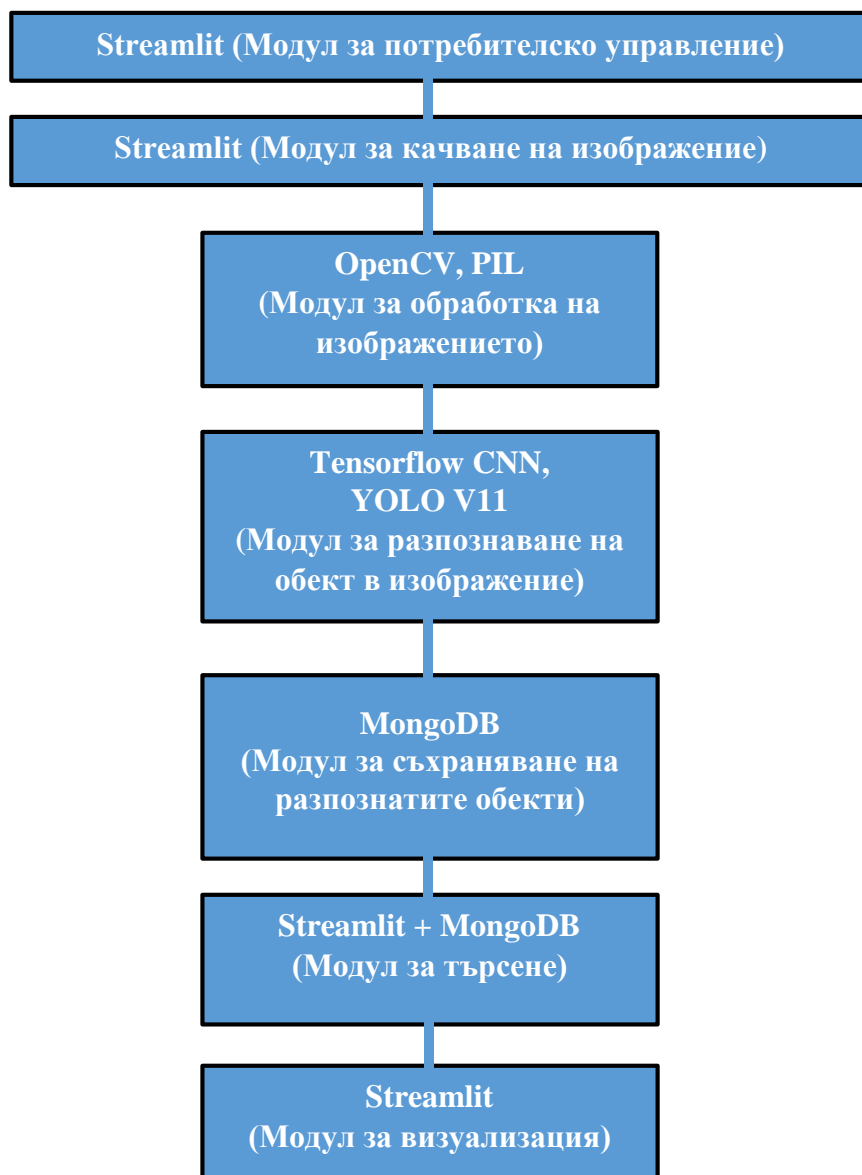
- **Streamlit** - Използва се за създаване на интерактивни елементи: бутони, текстови полета, селектори, слайдери, табове и страници. Също така визуализира изображения (оригинални и обработени), таблици с данни (pandas DataFrame) и графики (bar_chart).
- **CSS** - използван за стилизиране на интерфейса: цветове, фон, бутони, размери и визуални ефекти.

В **контролера** е връзката между изгледа и модела, т.е. логиката, която обработва действията.

- **Python** - Всички функции, условни проверки, обработка на данни, управление на сесии и извиквания към външни библиотеки са реализирани в Python.
- **Streamlit** - се използва за обработка на събитията (бутони натиснати, изображение качено, избран модел). Контролерът „чува“ тези събития чрез Streamlit и ги пренасочва към подходящите функции.

2.5.1 – Архитектура на софтуерната система

Архитектурата на системата е представена на фиг. 12. Тя включва отделни модули със специфични роли, които работят съвместно за реализиране на пълния процес. Всеки модул изпълнява ясно дефинирани задачи и обменя данни с останалите чрез стандартизирани интерфейси.



Фиг. 12 Архитектура на софтуерната система

Архитектурата на системата се разделя на седем основни модула:

- **Модул за потребителско управление** - Отговаря за регистрация, логин, управление на сесии, смяна на парола и възстановяване на достъп. Входните данни се валидират и паролите се хешират. Модулът е свързан с персистентното хранилище за потребители и предоставя API за идентификация (`user_id`, `username`) на останалата част от системата. Сигурността и управлението на сесии са му основен фокус.
- **Модул за качване на изображения** - Приема изображения от различни източници (качване, поставяне като `base64` или `URL`, `clipboard`). Извършва базова валидация на формати, конвертира входовете в потоци/обекти за по-нататъчна обработка и предава изображението към обработващия модул. Основният фокус е изборът на източник и метаданни.
- **Модул за обработка на изображения** - Подготвя изображението за разпознаване: отваряне (`PIL`), евентуално преоразмеряване (`OpenCV`), нормализация и форматиране за подаване към модела. Генерира копия за визуализация (`preview`) и за запамятаване (`compressed/base64`). Управлява грешки при формати и размери и връща метаданни за резолюция и източник.
- **Модул за разпознаване на обект в изображение** - избраният модел за детекция връща `bounding boxes`, класове, `confidence` и визуално означено изображение. Този модул работи като логическо ядро за извличане на обекти и предоставя структурирани резултати (`object_counts`, `objects`) към следващите стъпки.
- **Модул за съхранение на разпознатите обекти** - Получава резултатите и метаданните от детекцията и ги записва в хранилище (колекции по класове, `multiclass` колекция или `no_detections` колекция). Съхранява изображение като `base64`, `timestamp`, потребителски идентификатор и параметри на модела.

- **Модул за търсене** - Предоставя възможности за извличане на информация от предишни детекции, филтрация по клас или източник. Записите съдържат метаданни (време на детекцията, използван модел, източник, праг на увереност) и резултатите от разпознаването (класове и брой обекти). Данните се сортират по дата (най-новите първи).
- **Модул за визуализация** - Всеки запис се показва в две колони. Лява колона: име на изображението, визуализация на самото изображение. Дясна колона: подробна информация за детекцията – дата и час (конвертирани в локалната часова зона), модел, източник, праг на увереност и списък с разпознати класове и техните бройки.

2.5.2 – Концептуален модел на системата за потребителя

Системата е изградена като последователност от стъпки, които водят потребителя от момента на вход в приложението до получаване на резултати и управление на данните. Показано във фиг. 13.

В първата стъпка – **Потребител** - Всеки потребител преминава през процес на регистрация и вход в системата. За целта се използва защитена автентикация с криптирани пароли и проверка на имейл. След успешен вход, потребителят получава достъп до основните функции на приложението чрез персонализиран интерфейс.

В следващата стъпка - **Качване на изображение** - Потребителят може да предостави изображение по няколко начина – качване от устройство или поставяне от клипборд. Това изображение е входната точка за процеса на анализ.

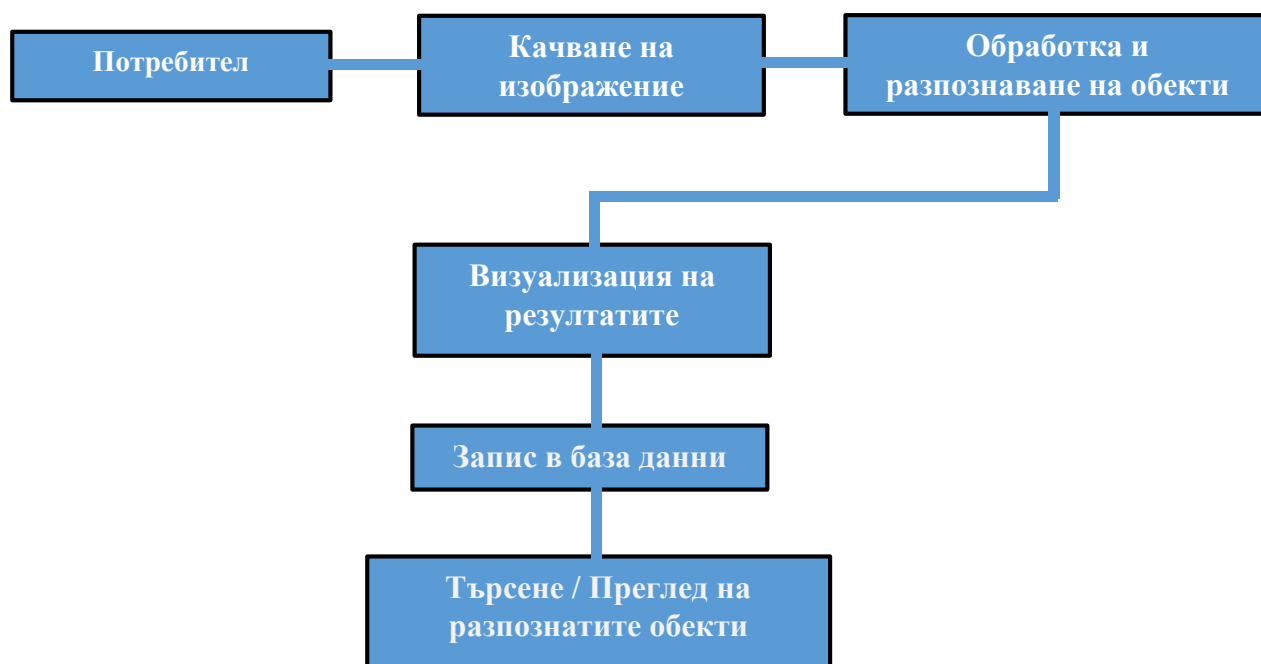
В следващата стъпка - **Обработка и разпознаване на обекти** - След като изображението бъде избрано, системата го предава към модел, който анализира съдържанието и извлича информация за наличните обекти. Всеки обект се описва с клас, ниво на увереност и координати на обектната рамка.

В следващата стъпка - **Визуализация на резултатите** - Резултатите от анализа се визуализират в удобен за потребителя формат (маркирани обекти

върху оригиналното изображение), статистика за броя на разпознатите класове, средна увереност и разпределение по категории. Потребителят може да прегледа резултатите и да изследва детайлна информация за всяко засечено откритие.

В следващата стъпка - **Запис в база данни** - Всички извършени детекции се съхраняват автоматично в база данни (MongoDB). Записът включва изображението, информация за модела, избраните параметри и разпознатите обекти. Това гарантира, че историята на действията се пази и може да бъде използвана за последващи справки и анализи.

В следващата стъпка - **Търсене / Преглед на разпознатите обекти** - Потребителят може да преглежда своята история на детекции чрез удобен интерфейс с филтри по класове или източник на изображение. Системата предоставя и обобщена статистика за всички заснети обекти – например най-често срещани категории, средна увереност.



Фиг. 13 Концептуален модел на системата за потребителя

2.6 – Софтуерни инструменти и библиотеки

2.6.1 – 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.

2.6.2 – 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.

2.6.3 – 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.

2.6.4 – 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.

2.6.5 – 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 от 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.

2.6.6 – 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.

2.6.7 – 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.

2.6.8 – 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 оптимизирани 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 включва информация за клас на обект, координати на ограничителни рамки и ниво на увереност. Те се записват 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.

2.6.9 – 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 в последствие 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.

2.6.10 – Datetime

Datetime е една от стандартните библиотеки за време, предназначена за работа с дати и часове. Тя предоставя богат набор от класове и функции, чрез които могат да се представят времеви стойности, да се извършват изчисления върху тях и да се преобразуват в различни формати. Основното предимство на datetime е, че съчетава висока прецизност с лесен за употреба интерфейс, което я прави предпочитан инструмент за работа с времеви данни. В приложение datetime се използва за създаване на времеви печати при всяка детекция на обекти.

2.6.11 – UUID

Универсално уникалният идентификатор (UUID) е 128-битово число, което е проектирано да бъде уникално само по себе си. UUID се използва за маркиране и съхраняване на резултатите от извършените детекции. При всяко ново качване на изображение и последваща обработка от модела за разпознаване на обекти, системата създава уникален идентификатор, който се асоциира с конкретния запис в базата данни. Основната причина за избора на UUID пред други библиотеки или механизми за идентификация е гаранцията за глобална уникалност, без да е необходимо поддържане на централен брояч или база данни, която да следи използваните стойности. UUID предоставя 128-битови идентификатори, които могат да бъдат генерирани локално и с изключително нисък риск от колизии.

2.6.12 – Streamlit-cookie-manager

Streamlit-cookie-manager е външна библиотека за Streamlit, която позволява работа с „cookies” директно в приложения. Основната причина за избора на streamlit-cookie-manager пред алтернативи е неговата интеграция със Streamlit. Други библиотеки за управление на „cookies“ като Flask-Session или Django sessions предлагат по-богата функционалност, но изискват тежка архитектура и внедряване на допълнителни сървърни компоненти, което би усложнило и забавило разработката на приложението. Streamlit-cookie-manager от своя страна е леко решение, което работи директно в средата на Streamlit, без необходимост от външни зависимости, и е оптимизирано за сценарии, където се търси простота и бърза интеграция. Библиотеката е използвана за управление на потребителските сесии след успешен вход в системата. При логин приложението записва информация за текущата сесия в „cookie“, като например уникален токен или идентификатор на потребителя. По този начин се избягва необходимостта от повторно въвеждане на данни при всяко действие, а системата може надеждно да разпознае кой потребител извършва дадена операция.

2.6.13 – St_img_pastebutton

St_img_pastebutton е външен компонент за библиотеката Streamlit, който позволява директно поставяне на изображения от клипборда чрез бутон в потребителския интерфейс. Чрез този компонент потребителят може лесно да вмъква изображения за детекция, независимо дали са взети от интернет страница, научна статия или генерирани от собственото му устройство. Тази функционалност допринася за по-интуитивна и бърза работа със системата, като същевременно минимизира риска от загуба на време и грешки при работа с файлове. Изборът на st_img_pastebutton пред алтернативни решения е взет от неговата интеграция със Streamlit и лекотата, с която може да бъде внедрен в съществуващи приложения. Докато други библиотеки за работа с изображения (например OpenCV или Pillow) предоставят богати възможности за обработка, те не решават проблема с директното въвеждане на данни от клипборда в уеб интерфейса

2.6.14 – SMTPlib

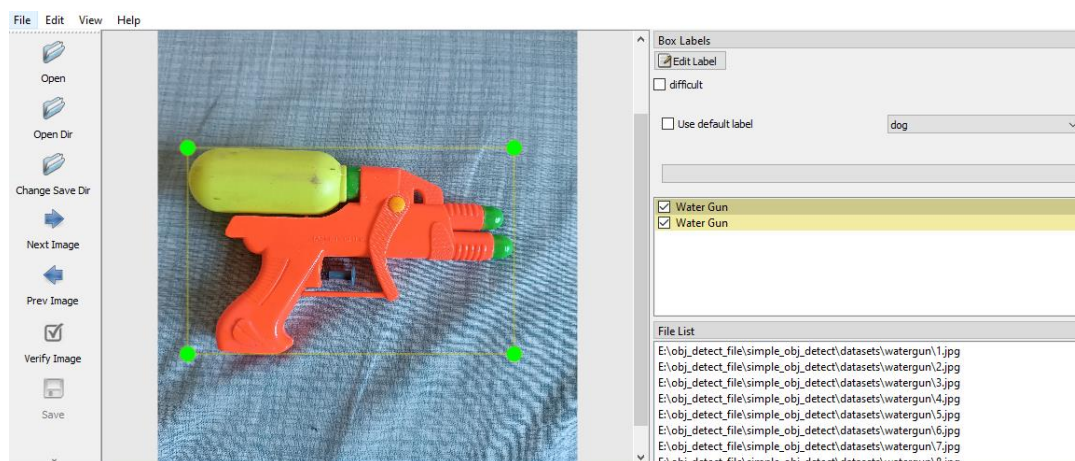
SMTPlib е вградена в Python библиотека, която предоставя работа с протокола Simple Mail Transfer Protocol (SMTP). Чрез тази библиотека се изпращат имейли директно от Python приложението, без да е необходимо използването на външни пакети или сложни конфигурации. Изборът на smtpplib пред други решения се основава на това, че е стандартна библиотека в Python, което означава, че е налична по подразбиране и не изисква допълнителна инсталация. Това улеснява приложението, тъй като намалява зависимостите от външни пакети.

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

В тази глава ще се разгледат важни моменти и етапи в кода на подсистемата представен като псевдокод, както и експерименталната част, свързана с обучението и оценката на модела.

3.1 – Псевдокод на подсистемата за обучение

Започваме с - **Подготовка на данни** – Ние ръчно чрез програмата Labellmg задаваме координатите на обектите и се подготвят техните анотации, които описват местоположението и класа на обектите в изображенията. Показано във фиг. 14 .



Фиг. 14 Анотация на обект чрез програмата Labellmg

Следва – **Въвеждане на изображения** – В променливите DIR_ROOT, DIR_IMAGE и DIR_ANNOTATIONS указваме съответно основната директория с данните , поддиректорията с изображенията и поддиректорията с анотациите.

Algorithm set_directories is:

Вход: Главната директория на проекта (DIR_ROOT), поддиректорията с изображенията (DIR_IMAGES) , поддиректорията с анотациите (DIR_ANNOTATIONS)

Изход: пълните пътища към папката с изображения и папката с анотации

Стъпка 1: Задаваме DIR_ROOT като


```
'/obj_detect_file/simple_obj_detect'
```

Стъпка 2: Задаваме DIR_IMAGES като DIR_ROOT +

```
'/datasets/watergun'
```

Стъпка 3: Задаваме DIR_ANNOTATIONS като DIR_ROOT +

```
'/datasets/watergun annotations'
```

Стъпка 4: Връщаме готовите пътища DIR_IMAGES и DIR_ANNOTATIONS

Следва – **Анализ и корекции на изображения** - В променливата CLASSES се съхраняват всички допустими класове обекти за разпознаване. Функцията parse_annotation чете XML анотация и извлича информация за файла, размерите на изображението, класа и координатите на ограничителната кутия (bounding box). След това данните се зареждат в dataframe, като координатите на рамката се нормализират в интервала [0,1].

Algorithm parse_and_prepare_annotations is:

Вход: Папка с XML анотации (DIR_ANNOTATIONS), списък с класове (CLASSES)

Изход: DataFrame с информация за изображенията и нормализирани bounding box координати

Стъпка 1: Дефинираме CLASSES = ["Water Gun"]

Стъпка 2: Дефинираме функция parse_annotation(root):

2.1. Създаваме празен речник annotation

2.2. Извличаме името на файла от XML и го съхраняваме в annotation['filename']

2.3. Извличаме ширината и височината на изображението -> annotation['width'], annotation['height']

2.4. Намираме обекта (object) в XML

2.5. Извличаме името на класа (class_name) и премахваме празните символи

2.6. Ако class_name не принадлежи към CLASSES -> хвърли грешка

2.7. Записваме индекса на класа -> annotation['class_id']

2.8. Извличаме координати на bounding box:

- xmin, ymin, xmax, ymax

- записваме ги в annotation

2.9. Връщаме речника annotation

Стъпка 3: Извикваме `load_xml_annotations(DIR_ANNOTATIONS, parse_annotation)` за да заредят всички анотации

Стъпка 4: Превръщаме списъка от анотации в DataFrame -> `dataframe_original`

Стъпка 5: Прави копие `df = dataframe_original.copy()`

Стъпка 6: Нормализираме координатите на bounding box към интервала `[0,1]`:

- `xmin_n = xmin / width`
- `ymin_n = ymin / height`
- `xmax_n = xmax / width`
- `ymax_n = ymax / height`

Стъпка 7: За всяка нормализирана координата (`xmin_n`, `ymin_n`, `xmax_n`, `ymax_n`):

- Ограничаваме стойностите да са в границите `[0.0, 1.0]`

Стъпка 8: Връщаме готовия DataFrame `df` с нормализираните координати

Следва – **Изграждане на обучаваща среда** - В променливите `BUFFER_SIZE`, `BATCH_SIZE` и `SIZE` задаваме параметри за обработка на изображенията – колко примера да се разбъркват, колко примера да има в една партида и какъв е крайният размер на изображението (в пиксели). Данните се разделят на тренировъчен и валидационен набор, след което се създават TensorFlow Dataset обекти, които се нормализират, разбъркват, групират и подготвят за обучение.

Algorithm prepare_datasets is:

Вход: DataFrame с изображения и анотации (`df`), параметри `BUFFER_SIZE`, `BATCH_SIZE`, `SIZE`

Изход: тренировъчен dataset (`train_ds`) и валидационен dataset (`val_ds`) готови за подаване към модела

Стъпка 1: Дефинираме параметри:

```
BUFFER_SIZE = 200
BATCH_SIZE = 10
SIZE = 340
```

Стъпка 2: Разделяме df на тренировъчен (train_df) и валидационен (val_df) набор:

- използваме train_test_split с test_size = 0.2 (80% train, 20% val)
- запазваме пропорцията на класовете (stratify по class_id)
- random_state = SEED за повторимост
- shuffle = True за разбъркване

Стъпка 3: Създаваме тренировъчен dataset (train_ds):

- 3.1. Създаваме dataset от train_df чрез create_dataset_from_dataframe
- 3.2. Дефинираме map_func: която зарежда изображението и го нормализира до размер SIZE×SIZE с 1 канал
- 3.3. Прилагаме map върху train_ds (с num_parallel_calls = AUTOTUNE)
- 3.4. Разбъркваме dataset с BUFFER_SIZE
- 3.5. Кешираме dataset в паметта
- 3.6. Повтаряме dataset безкрайно (repeat)
- 3.7. Групираме данните на партии (batch) с размер BATCH_SIZE
- 3.8. Използваме prefetch за оптимизация на входа (AUTOTUNE)
- 3.9. Определяме броя на стъпките за епоха: train_steps = max(1, len(train_df) // BATCH_SIZE)

Стъпка 4: Създаваме валидационен dataset (val_ds):

- 4.1. Създаваме dataset от val_df чрез create_dataset_from_dataframe
- 4.2. Прилагаме същата map_func за нормализация
- 4.3. Кешираме dataset в паметта
- 4.4. Повтаряме dataset безкрайно (repeat)
- 4.5. Групираме данните на партии (batch) с размер BATCH_SIZE
- 4.6. Използваме prefetch за оптимизация
- 4.7. Определяме броя на стъпките за епоха: val_steps = max(1, len(val_df) // BATCH_SIZE)

Стъпка 5: Извеждаме броя на елементите в `train_df` и `val_df`, както и броя стъпки `train_steps` и `val_steps`

Стъпка 6: Връщаме `train_ds` и `val_ds`

Следва – **Процес на обучение** - В нашия код архитектурата е сравнително проста CNN. Има три convolution слоя, след това плоска част с плътни (dense) слоеве, и накрая два изхода. В този блок от кода се дефинират параметрите за модела, създава се невронна мрежа за детекция на обекти, компилира се с подходяща функция на загуба и метрики, и след това моделът се тренира върху тренировъчния и валидационния набор.

Algorithm build_and_train_model is:

Вход: тренировъчен dataset (`train_ds`), валидационен dataset (`val_ds`), брой епохи (`FIT_EPOCHS`), форма на входа (`INPUT_SHAPE`)

Изход: обучен модел (`model`), история от тренировката (`history`)

Стъпка 1: Дефинираме параметри на модела:

`INPUT_SHAPE = (SIZE, SIZE, 1)`

`FIT_EPOCHS = 20`

Стъпка 2: Дефинираме callbacks:

- `EarlyStopping`: наблюдава `'val_loss'`
- `patience = 75` (ако няма подобрене след 75 епохи -> (спиране)

- `min_delta = 0.001` (минимално подобрене за да се брои)

- `restore_best_weights = True` (възстановява най-добрите (тежести)

- `verbose = 1` (отпечатва съобщения)

Стъпка 3: Създаваме архитектурата на модела (`backbone`):

3.1. Входен слой: `Input(shape = INPUT_SHAPE)`

3.2. `Conv2D(16` филтър, `kernel 3×3`, `relu` активация, `padding='same')`

3.3. `MaxPool2D`

3.4. `Conv2D(32` филтър, `kernel 3×3`, `relu` активация, `padding='same')`

3.5. `MaxPool2D`

3.6. Conv2D(64 филтъра, kernel 3×3, relu активация, padding='same')

3.7. MaxPool2D

3.8. Flatten (преобразува в едномерен вектор)

3.9. Dense(512 неврона, relu)

3.10. Dense(128 неврона, relu)

Стъпка 4: Създаваме изходни „глави“ на модела:

- bbox_output: Dense(4 неврона, activation='linear') -> предсказва координати на bounding box

- class_output: Dense(len(CLASSES), activation='softmax') -> предсказва класа на обекта

Стъпка 5: Дефинираме модела:

inputs = входен слой

outputs = [bbox_output, class_output]

име = 'object_detection_3class'

Стъпка 6: Компилираме модела:

- Optimizer: Adam

- Загуба (loss):

* bbox -> MeanSquaredError

* class -> SparseCategoricalCrossentropy

- Загубите имат еднаква тежест (1.0)

- Метрики:

* bbox -> MSE

* class -> Accuracy

Стъпка 7: Извеждаме архитектурата на модела (model.summary)

Стъпка 8: Обучение на модела (model.fit):

- вход: train_ds

- steps_per_epoch = train_steps

- validation_data = val_ds

- validation_steps = val_steps

- epochs = FIT_EPOCHS

- callbacks = FIT_CALLBACKS (EarlyStopping)

- verbose = 1 (подробни съобщения)

Стъпка 9: Запазваме историята от обучението в променлива history

Стъпка 10: Връщаме обучен модел и историята от тренировката

Следва – **Оценяване и анализ на резултатите** - Тук проверяваме доколко добре се е научил моделът – гледаме точност на класовете, грешка при bounding boxes и обща загуба. После визуализираме историята на обучението (train/val loss).

Algorithm evaluate_trained_model is:

Вход: обучен модел (model), тренировъчен dataset (train_df), валидационен dataset (val_df), размер на изображение (SIZE), размер на batch (BATCH_SIZE), история от тренировката (history)

Изход: метрики от оценката (model_evaluation_info), графика с историята на обучението и резултатите

Стъпка 1: Дефинираме функция eval_ds_from_df(dframe):

- Създаваме dataset от подадения DataFrame чрез create_dataset_from_dataframe(dframe, use_norm_cols=True)
- Преобразуваме всеки елемент със load_normalize_image(path, размер = [SIZE, SIZE], 1 канал)
- Групираме елементите по batch (BATCH_SIZE)
- Връщаме подготвения dataset

Стъпка 2: Създаваме eval dataset-и (без повторение):

```
train_eval_ds = eval_ds_from_df(train_df)
val_eval_ds   = eval_ds_from_df(val_df)
```

Стъпка 3: Оценяваме модела върху dataset-ите:

```
eval_train = model.evaluate(train_eval_ds, verbose=2,
return_dict=True)
eval_val   = model.evaluate(val_eval_ds,   verbose=2,
return_dict=True)
```

Стъпка 4: Събираме резултатите в текстов отчет model_evaluation_info:

Включи:

- class_accuracy (точност на класификацията)
- bbox_mse (грешка за координати на bounding box)
- total_loss (обща функция на загуба)

Стъпка 5: Отпечатваме резултатите (model_evaluation_info)

Стъпка 6: Визуализираме историята на обучението и резултатите:

```
plot_history(history, model_evaluation_info)
```

Стъпка 7: Връщаме отчет за оценката и графика на резултатите

Следва – **Съхранение и интеграция на обучен модел** - Тук запазваме модела във .h5 файл. След това този файл може да бъде използван в други Python скриптове или в приложения.

Algorithm save_trained_model is:

Вход: обучен модел (final_model), път за запазване (path)

Изход: файл с обучен модел (.h5 формат)

Стъпка 1: Дефинираме път за запазване:

```
path =  
'/obj_detect_file/simple_obj_detect/working/simple_object_detection  
.h5'
```

Стъпка 2: Извикваме save(path) върху final_model:

- сериализираме структурата на невронната мрежа
- запазваме обучените тегла
- запазваме настройките за компилация (optimizer, loss, metrics)

Стъпка 3: Създаваме файл .h5 на зададеното място

Стъпка 4: Връщаме готов модел

3.2 – Псевдокод на подсистемата за разпознаване на обекти

Започваме с - **Въвеждане на данни** - Ако потребителят избере опцията "Upload an image", се извиква компонент за качване на файл, който позволява избор само на определени типове изображения. Каченото изображение се запазва в променливата source_image за последваща обработка.

Algorithm image_upload is:

Вход: избор на източник на изображение (image_source)

Изход: качено изображение (source_image)

Стъпка 1: Проверяваме дали `image_source` е равно на "Upload an image".

Стъпка 2: Ако условието е вярно, стартираме компонент за качване на изображение (`file_uploader`).

Стъпка 3: Ограничаваме типовете файлове до `jpg`, `png`, `jpeg`, `bmp` и `webp`.

Стъпка 4: Указваме уникален ключ „`file_uploader`“.

Стъпка 5: Ако потребителят качи изображение, запазваме файла в променливата `source_image`.

Стъпка 6: Връщаме стойността на `source_image`.

Ако потребителят избере опцията "**Paste from clipboard (text input)**", се предоставя текстово поле, в което той може да постави изображение като `base64` низ или `URL`. След това приложението обработва въведените данни и ги преобразува в изображение, което се запазва в променливата `source_image`. Ако данните са невалидни или не могат да бъдат обработени, се показва съобщение за грешка.

Algorithm paste_image is:

Вход: избор на източник на изображение (`image_source`)

Изход: качено изображение (`source_image`), ако данните са валидни

Стъпка 1: Проверяваме дали `image_source` е равно на "Paste from clipboard (text input)".

Стъпка 2: Ако условието е вярно, показваме текстово поле (`text_area`), където потребителят може да постави изображение в `base64` или като `URL`.

Стъпка 3: Ако `paste_data` не е празно:

3.1: Опитваме се да обработим съдържанието.

3.2: Ако започва с "`data:image`", разделяме `header` и кодираната част, декодираме `base64` и създаваме поток `BytesIO` като `source_image`.

3.3: Ако започва с "`http://`" или "`https://`", изтегляме съдържанието чрез `HTTP` заявка и запазваме в `BytesIO` като `source_image`.

3.4: В противен случай приемаме, че е чист `base64` низ,

декодираме и запазваме в BytesIO като `source_image`.

Стъпка 4: Ако възникне грешка при обработката, показваме съобщение „Could not process the pasted image. Please try another method.“

Ако потребителят избере опцията "**Paste from clipboard (button)**", се показва бутон, който позволява директно поставяне на изображение от клипборда. Поставените данни се обработват като base64, преобразуват се в двоичен формат и се запазват като изображение в променливата `source_image`. Ако обработката е успешна, изображението се визуализира в приложението. При грешка се показва съобщение с информация за проблема.

Algorithm paste_image_button is:

Вход: избор на източник на изображение (`image_source`), данни от клипборда (`image_data`)

Изход: визуализирано изображение (`source_image`), ако данните са валидни

Стъпка 1: Проверяваме дали `image_source` е равно на "Paste from clipboard (button)".

Стъпка 2: Ако условието е вярно, показваме бутон „Paste from Clipboard“.

Стъпка 3: Ако `image_data` не е None (потребителят е поставил данни):

3.1: Опитваме се да разделим данните на header и кодираната част.

3.2: Декодираме base64 съдържанието до двоичен формат (`binary_data`).

3.3: Създаваме поток BytesIO от `binary_data` и го запазваме в `source_image`.

3.4: Визуализираме изображението с надпис „Pasted from Clipboard“.

Стъпка 4: Ако възникне грешка при обработката, показваме съобщение „Failed to process clipboard image“ с информация за грешката.

Следва – **Предварителна обработка на изображението** – Програмата проверява дали подаденото изображение е поток от тип BytesIO, и ако е така, връща указателя му в началото. След това изображението се отваря с PIL, преобразува се в масив и в цветови формат BGR (за работа с OpenCV). Накрая изображението се преоразмерява според зададените ширина и височина от session_state.

Algorithm prepare_and_resize_image is:

Вход: източник на изображение (source_image), целеви размери (resize_width, resize_height)

Изход: преоразмерено изображение (resized_img)

Стъпка 1: Опитваме се да изпълним обработката (try).

Стъпка 2: Ако source_image е обект от тип BytesIO, връщаме указателя му в началото (seek(0)).

Стъпка 3: Отваряме изображението със библиотеката PIL след което получаваме img_pil.

Стъпка 4: Преобразуваме img_pil в NumPy масив и го конвертираме от RGB към BGR след което получаваме img_np.

Стъпка 5: Преоразмеряваме img_np до размерите (resize_width, resize_height) с помощта на OpenCV и получаваме resized_img.

Стъпка 6: Връщаме resized_img като резултат.

Следва – **Екстракция на признаци** – Това действие се извършва от невронната мрежа (YOLO или TensorFlow модел).

- Предсказване с YOLO – Изображението, качено от потребителя, се подава на обученния YOLO модел. Моделът извършва предсказване върху изображението.

Algorithm yolo_predict is:

Вход: качено изображение (uploaded_image), праг на увереност (confidence_value)

Изход: резултат от предсказването (result)

Стъпка 1: Подаваме uploaded_image на YOLO модела заедно с confidence_value.

Стъпка 2: YOLO извършва откриване на обекти.

Стъпка 3: Връщаме резултат с намерените обекти и техните характеристики като `result`.

- Предсказване с TensorFlow – Изображението се преобразува в масив (`img_arr`) и се подава на TensorFlow модела за предсказване. Моделът връща ограничителните кутии (`bounding boxes`) и стойности за класовете (`pred_class_logits`).

Algorithm tensorflow_predict is:

Вход: изображение като масив (`img_arr`)

Изход: ограничителни кутии (`pred_bbox`), стойностите за класове (`pred_class_logits`)

Стъпка 1: Подаваме `img_arr` на TensorFlow модела чрез метода `predict`.

Стъпка 2: Моделът изчислява позициите на обектите като `pred_bbox`.

Стъпка 3: Моделът изчислява вероятностите на стойностите за класовете като `pred_class_logits`.

Стъпка 4: Връщаме `pred_bbox` и `pred_class_logits`.

Следва – **Локализация и класификация** – След като моделите направят предсказване, от резултатите се извличат откритите обекти.

- Извличане от YOLO резултат

Algorithm yolo_boxes is:

Вход: резултат от YOLO предсказване (`result`)

Изход: координати на ограничителни кутии (`boxes`)

Стъпка 1: Достъпваме първия елемент от резултата (`result[0]`).

Стъпка 2: Извличаме списъка с `bounding boxes` от `result[0].boxes`.

Стъпка 3: Връщаме `boxes`.

- Извличане от TensorFlow резултат

Algorithm tensorflow_boxes is:

Вход: предсказани ограничителни кутии (`pred_bbox`), предсказани

класове (pred_class_logits)

Изход: обработени ограничителни кутии (pred_bbox), избран клас (pred_class)

Стъпка 1: Прилагаме операция squeeze върху pred_bbox, за да премахнем излишните размерности.

Стъпка 2: Намираме индекса на класа с най-висока стойност в pred_class_logits чрез argmax по последната ос.

Стъпка 3: Вземаме първия елемент от резултата на argmax като pred_class.

Стъпка 4: Връщаме обработените pred_bbox и pred_class.

Следва – **Обработка на резултатите** – Прилага се филтриране на обектите чрез праг на увереност и чрез Non-Maximum Suppression (NMS) което премахва припокриващи се кутии. След като бъдат извлечени ограничителните кутии от резултатите на модела, всяка кутия съдържа информация за класа на открития обект. Чрез class_map се превежда class_id към име на класа, след което в речник (class_counts) се натрупва броят на срещанията за всеки клас.

Algorithm count_detected_classes is:

Вход: списък от ограничителни кутии (boxes), речник за съпоставяне на клас ID към име (class_map)

Изход: речник с брой на обекти по класове (class_counts)

Стъпка 1: Създаваме празен речник class_counts.

Стъпка 2: За всеки box в списъка boxes:

2.1: Извличаме идентификатора на класа → class_id.

2.2: Намираме името на класа чрез class_map[class_id] → class_name.

2.3: Увеличаваме брояча за class_name в class_counts с 1 (ако липсва, започва от 0).

Стъпка 3: Връщаме речника class_counts.

Следва – **Структуриране на изхода** – Създава се структура (речник), която съдържа информация за извършената детекция: времето, използвания модел, праг на увереност, източника и името на изображението, неговата резолюция, броя на откритите обекти и идентификатора на потребителя.

Algorithm detection_data is:

Вход: тип модел (`model_type`), праг на увереност (`confidence_value`), източник на изображение (`image_source`), изображение (`source_image`), размери на изображение (`image_width`, `image_height`), брое на класове (`class_counts`), списък с ограничителни кутии (`boxes`) и потребителски ID (`user_id`)

Изход: речник с информация за детекцията (`detection_data`)

Стъпка 1: Създаваме `detection_data` с нужните полета

Стъпка 2: За всеки `box` в `boxes` с индекс `i`:

2.1: Извличаме класа на обекта от `CLASSES[int(box.cls)]`.

2.2: Извличаме увереността на модела като `float(box.conf)`.

2.3: Извличаме координатите на bounding box във формат (`x`, `y`, `w`, `h`) като `box.xywh.tolist()[0]`.

2.4: Добавяме нов речник в `detection_data["objects"]` със следните полета:

- `object_id = i`
- `class = име на класа`
- `confidence = увереност`
- `box_xywh = координати на кутията`

Стъпка 3: Връщаме готовия `detection_data`.

Следва – **Визуализация** - След като моделът извърши детекция и крайните резултати са записани, резултатите се изобразяват върху изображението.

- **Визуализация на резултати от YOLO**

Algorithm yolo_visualization is:

Вход: резултат от YOLO предсказване (`result`)

Изход: визуализирано изображение с отбелязани обекти

Стъпка 1: Използваме метода `plot()` върху `result[0]`, за да получим изображение с нарисувани ограничителни кутии и класове.

Стъпка 2: Преобразуваме цветния формат на изображението от BGR към RGB.

Стъпка 3: Показваме обработеното изображение в Streamlit със заглавие „Detection Results“.

- Визуализация на резултати от TensorFlow

Algorithm tensorflow_visualization is:

Вход: изображение като NumPy масив (`img_np`), координати на bounding box (`xmin`, `ymin`, `xmax`, `ymax`), етикет на класа (`label`)

Изход: визуализирано изображение с отбелязани обекти

Стъпка 1: Начертаваме правоъгълник около открития обект върху `img_np` с помощта на `cv2.rectangle`.

Стъпка 2: Добавяме текст с името на класа (`label`) над bounding box с помощта на `cv2.putText`.

Стъпка 3: Показваме изображението в Streamlit със заглавие „Detection Results“.

В рамките на експерименталната част е извършена серия от обучения на обектно-детекционния модел използван в нашата система с различен брой епохи, като целта е да се сравни ефективността на модела в зависимост от дължината на тренировъчния процес. Основният критерий за оценка е средноквадратичната грешка (`bbox_mse`), която съвпада с общата загуба (`total_loss`) както за тренировъчния, така и за валидационния набор.

3.3 – Експеримент

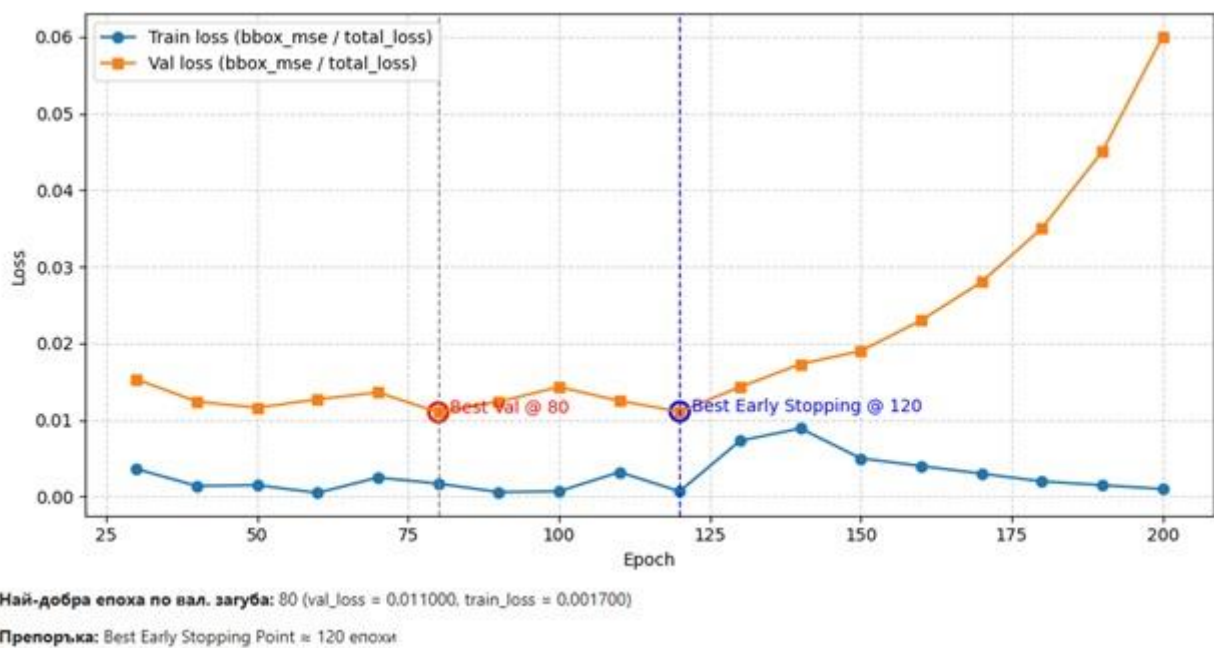
Експериментът е проведен с брой епохи в интервала от 30 до 200, като се използва стъпка от 10 епохи. По този начин са получени 18 различни конфигурации на обучение. За всяка конфигурация са записани стойностите на тренировъчната и валидационната загуба.

3.3.1 – Резултати

Стойностите на тренировъчната и валидационната загуба от епохите са показани на фиг. 15, а резултатите от експеримента са показани на фиг. 16.

	epoch	train_loss	val_loss	train/val_ratio
0	30	0.003600	0.015300	0.235294
1	40	0.001400	0.012400	0.112903
2	50	0.001500	0.011600	0.129310
3	60	0.000500	0.012700	0.039370
4	70	0.002500	0.013600	0.183824
5	80	0.001700	0.011000	0.154545
6	90	0.000600	0.012400	0.048387
7	100	0.000700	0.014300	0.048951
8	110	0.003200	0.012500	0.256000
9	120	0.000700	0.011100	0.063063
10	130	0.007300	0.014300	0.510490
11	140	0.008900	0.017300	0.514451
12	150	0.005000	0.019000	0.263158
13	160	0.004000	0.023000	0.173913
14	170	0.003000	0.028000	0.107143
15	180	0.002000	0.035000	0.057143
16	190	0.001500	0.045000	0.033333
17	200	0.001000	0.060000	0.016667

Фиг. 15 Таблица с конфигурации на обучение и техните стойности



Фиг. 16 Сравнителна графика на: Train vs Val loss по епохи

Наблюдавани са следните събития: при малък брой епохи (30–50) валидационната загуба намалява значително, което показва, че моделът бързо се адаптира.

В интервала между 80 и 120 епохи се достига минимална валидационна грешка (около 0.0110–0.0111), което показва най-добър баланс на обучение.

След 120-140 епохи започва отчетлив процес на свръхобучение (overfitting). Тренировъчната загуба продължава да спада, но валидационната загуба нараства постепенно, като достига стойности над 0.02 при 150 епохи и над 0.06 при 200 епохи. Това ясно демонстрира необходимостта от използването на механизъм за ранно спиране (early stopping), който да прекрати обучението в момента, когато се достигне най-ниската валидационна загуба. За целите на експеримента е изчислена и препоръчителна точка за ранно спиране. Тя съвпада с епохата, след която започва устойчиво нарастване на валидационната загуба.

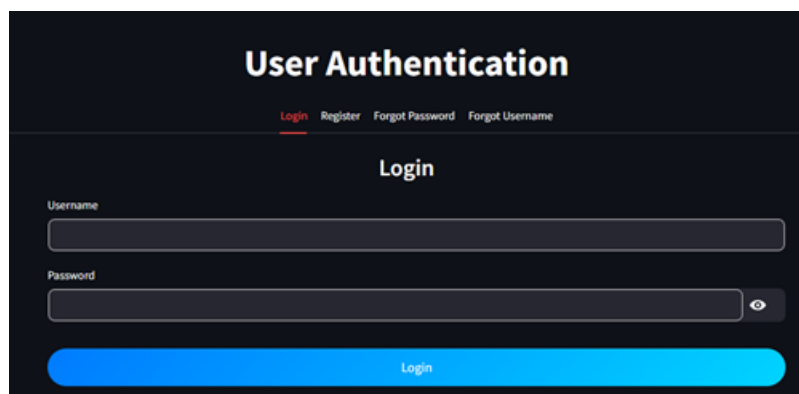
В заключение, конкретният случай най-добрата епоха е около 80, а препоръчителната точка за ранно спиране е приблизително след 120 епохи. Оптималният брой епохи за разглеждания модел е в диапазона 80–120.

Глава 4 – Изчислителна част/проектиране на блок схеми на алгоритми за софтуерната част/функционално тестване

В тази глава ще се разгледа структурата на този проект, както и неговият интерфейс и система за автентикация.

4.1 – 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. Във фиг. 17 е показана страницата за вход.

The image shows a web interface for user authentication. At the top, the title "User Authentication" is displayed in white text on a dark background. Below the title, there are four links: "Login" (highlighted in red), "Register", "Forgot Password", and "Forgot Username". The main section is titled "Login" and contains two input fields: "Username" and "Password". The "Password" field has a toggle icon (an eye) to the right of it. At the bottom of the form is a large blue button labeled "Login".

Фиг. 17 Вход за удостоверяване на потребител

4.1.1 – 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. Във фиг. 18 е показана страницата за регистрация.

A registration form titled "Register" with a double arrow icon. It contains four input fields: "Username", "Email", "Password", and "Confirm Password". The "Password" and "Confirm Password" fields have eye icons to toggle visibility. A blue "Register" button is at the bottom.

Фиг. 18 Регистрация на потребител

4.1.2 – Забравена парола

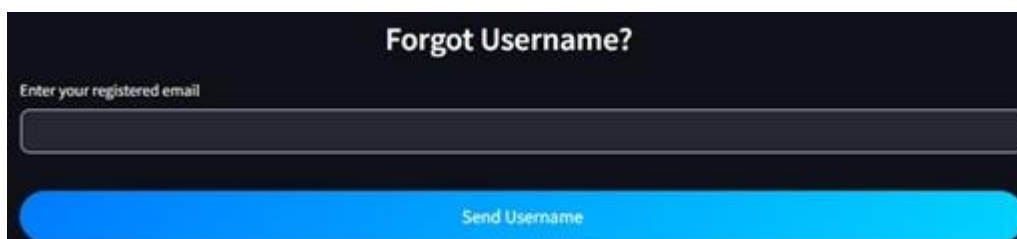
При забравена парола потребителят въвежда регистрирания си имейл и системата генерира временна парола, която автоматично се хешира и записва в базата. Потребителят получава новата парола по имейл чрез интеграция със SMTP сървър на Gmail. Във фиг. 19 е показана страницата за забравена парола.

A form titled "Forgot Password?". It has a label "Enter your registered email" above a single input field. A blue "Reset Password" button is at the bottom.

Фиг. 19 Забравена парола

4.1.3 – Забравено потребителско име

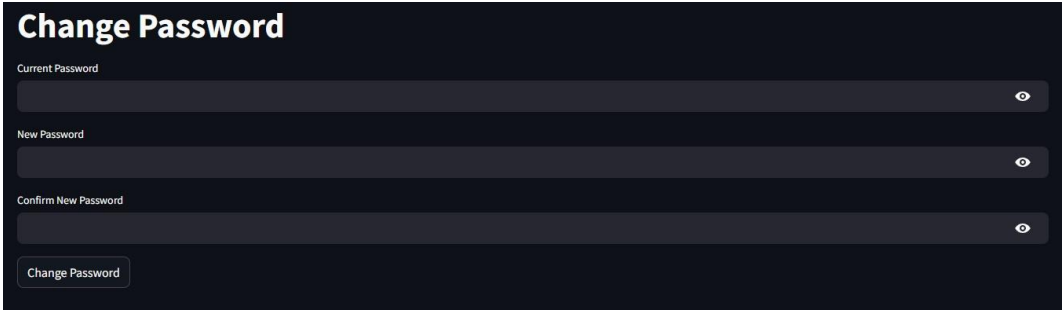
При забравено потребителско име се въвежда регистрираният имейл и системата автоматично изпраща на потребителите напомняне за своето потребителско име по електронна поща. Във фиг. 20 е показана страницата за забравено потребителско име.

A form titled "Forgot Username?". It has a label "Enter your registered email" above a single input field. A blue "Send Username" button is at the bottom.

Фиг. 20 Забравено потребителско име

4.2 – Промяна на парола

Функционалността за промяна на парола позволява на потребителите да актуализират своите идентификационни данни, като процесът е защитен чрез проверка на текущата парола. За целта потребителят въвежда старата си парола и нова парола с потвърждение. Системата проверява дали старата парола съвпада с тази в базата данни и дали новата отговаря на минимални изисквания за сигурност. При успех паролата се хешира и записва в MongoDB, като се уведомява потребителят за успешна промяна. Във фиг. 21 е показана страницата за промяна на парола.

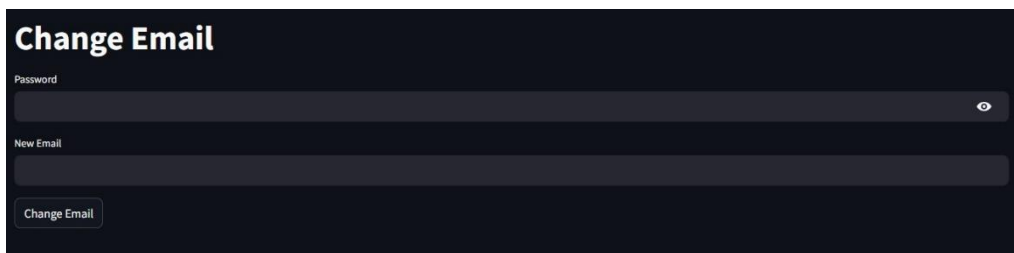


The image shows a 'Change Password' form with a dark theme. The title 'Change Password' is at the top. Below it are three input fields: 'Current Password', 'New Password', and 'Confirm New Password'. Each field has a small eye icon on the right to toggle visibility. At the bottom left, there is a button labeled 'Change Password'.

Фиг. 21 Промяна на парола

4.3 – Промяна на имейл

Модулът за промяна на имейл дава възможност на потребителите да актуализират своята електронна поща. За да се осигури сигурност, при всяка промяна е необходимо въвеждане на текущата парола. Системата валидира новия имейл, проверява дали той не е вече регистриран и дали има валиден формат. Ако проверките преминат успешно, имейлът се обновява в базата данни MongoDB, а потребителят получава съобщение за успешна промяна. При некоректна парола или вече съществуващ/невалиден имейл, приложението визуализира съответното предупреждение. Във фиг. 22 е показана страницата за промяна на имейл.



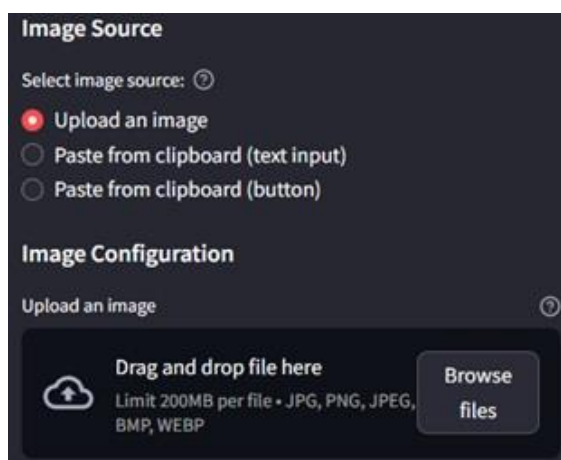
Фиг. 22 Промяна на имейл

4.4 – 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.

4.4.1 – Качване на изображение (Upload an image)

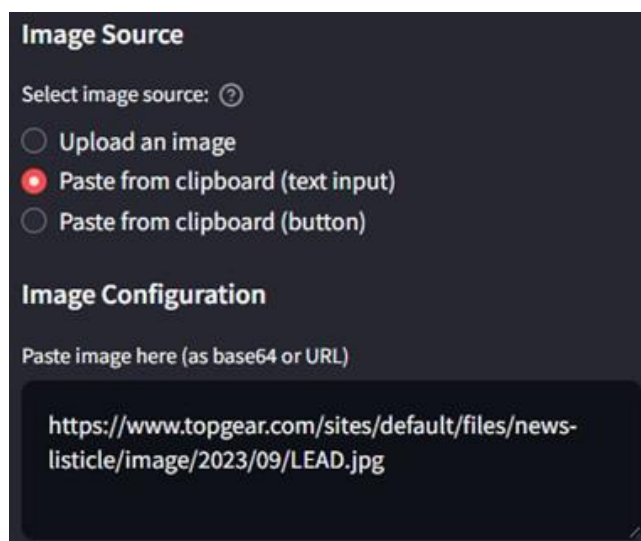
Приложението използва streamlit file_uploader с ограничение по разширения (jpg, png, jpeg, bmp, webp), осигурявайки валиден формат на входа. При качване файлът се отваря с PIL, а за унифициране на обработката се поддържа BytesIO. Преди всяка последваща операция указателят се връща в началото, за да се избегнат грешки при многократно четене. При неуспех се изписва ясна грешка и процесът се прекъсва. Във фиг. 23 е показано качване на изображение чрез upload.



Фиг. 23 Качване на изображение чрез upload

4.4.2 – Поставяне чрез clipboard base64/URL

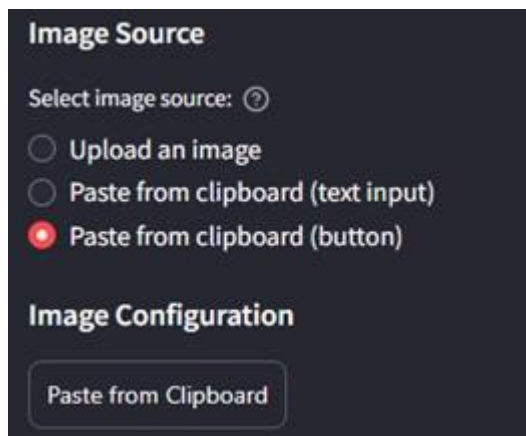
Вариантът за поставяне приема чист base64 или копираният адрес на изображението. Логиката детектира формата, декодира съдържанието чрез base64 или изтегля изображението с requests и го капсулира в BytesIO. Невалидни входни данни се отхвърлят със съобщение за грешка, без да се компрометира стабилността на сесията. Във фиг. 24 е показано поставяне на изображение чрез копирания адрес.



Фиг. 24 Поставяне на изображение чрез копирания адрес

4.4.3 – 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. Във фиг. 25 може да се разгледа това поставяне.



Фиг. 25 Поставяне на изображение чрез clipboard бутон

4.4.4 – Преоразмеряване на изображение

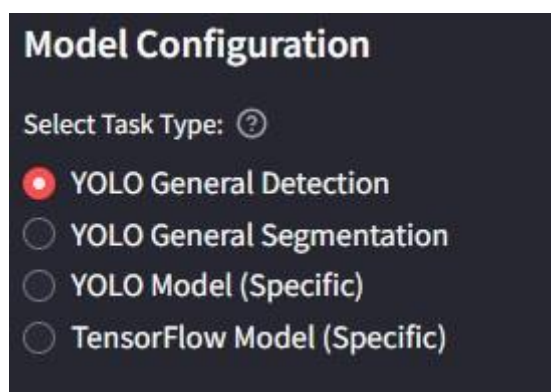
Преоразмеряването е управлявано от радио бутон „Enabled/Disabled“ в страничната лента и се активира само при налично изображение. При „Enabled“ се извличат оригиналните размери и се инициират контролите за ширина/височина. Самото мащабиране се изпълнява с OpenCV, като се осигурява коректна конверсия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

4.5 – 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.

4.5.1 – 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. Видимо е на фиг. 26. 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.



Фиг. 26 Избор на модел за разпознаване на обект

4.5.2 – 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. Показано на фиг. 27. 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.



Фиг. 27 Чувствителността на модела за разпознаване на обекти

4.5.3 – 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. Показано на фиг. 28.



Фиг. 28 Визуализация на разпознатите обекти в изображения

4.6 – 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. Показано на фиг. 29.



Фиг. 29 Структура на запазените данни като общ изглед

4.6.1 – Структура на документа за детекция

Всеки резултат от детекция се запазва като отделен документ. Документът съдържа метайнформация за изображението и конкретните открити обекти. Показано на фиг. 30. Основните полета са:

- ***_id*** - уникален идентификатор, който се асоциира с конкретния запис на направената детекция.
- ***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*** - броят обекти от всеки клас.

- **Objects** - списък с подробни данни за всяка детекция (идентификатор, клас, увереност, координати на bounding box).
- **user_id** - уникален идентификатор на потребителя, за да може историята да бъде персонализирана.
- **image_bytes** - изображението с нанесени маркировки, съхранено като base64 формат за директна визуализация.

```

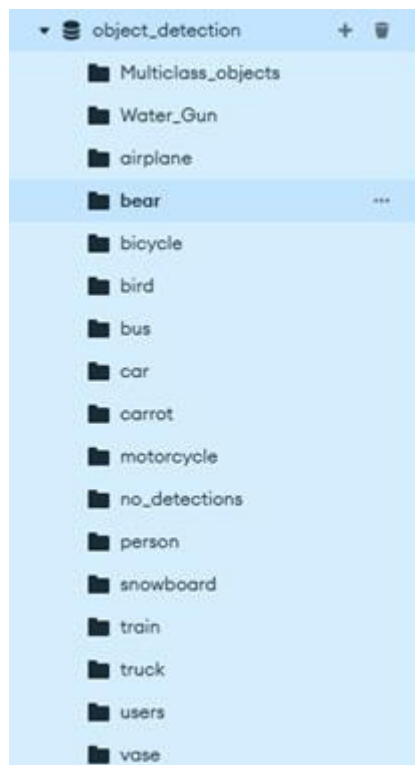
_id: ObjectId('68bc6db3f0cd189e0624f374')
timestamp : 2025-09-06T17:21:55.137+00:00
model_type : "Detection"
confidence_threshold : 0.4
source : "Paste from clipboard (button)"
image_name : "pasted_image"
* image_resolution : Object
  width : 3000
  height : 4000
* object_counts : Object
  person : 1
* objects : Array (1)
  * 0: Object
    object_id : 0
    class : "person"
    confidence : 0.4463236331939697
    * box_xywh : Array (4)
      0: 1500.185302734375
      1: 2017.769287109375
      2: 2987.7451171875
      3: 3964.46142578125
    user_id : "c95a2c46-4403-4c22-8af1-e774a0b303e2"
    image_bytes : "/9j/4AAQSkZJRgABAQAAQABAAQ/2wBDAAAGBGgcGBQgHBwcJCQgKDBQNDAsLDBkSEwSUHR..."

```

Фиг. 30 Структура на документа за детекция

4.6.2 – 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**. Показано на фиг. 31.



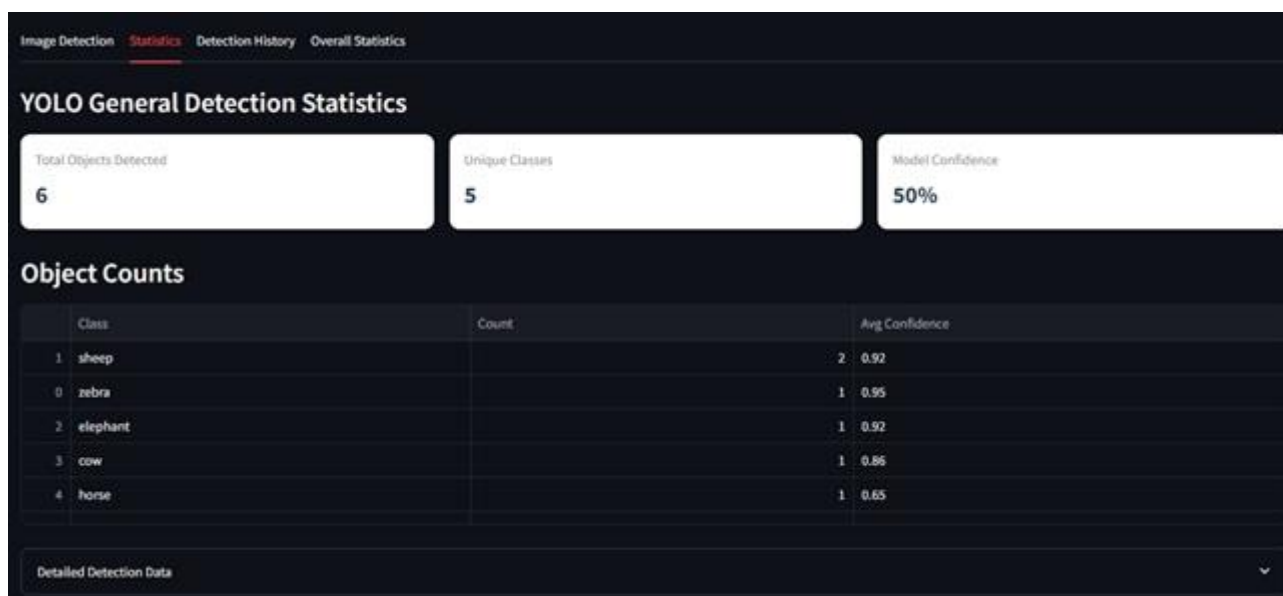
Фиг. 31 Структура на колекциите

4.7 – 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.

4.7.1 – Таблица с обекти и броя им

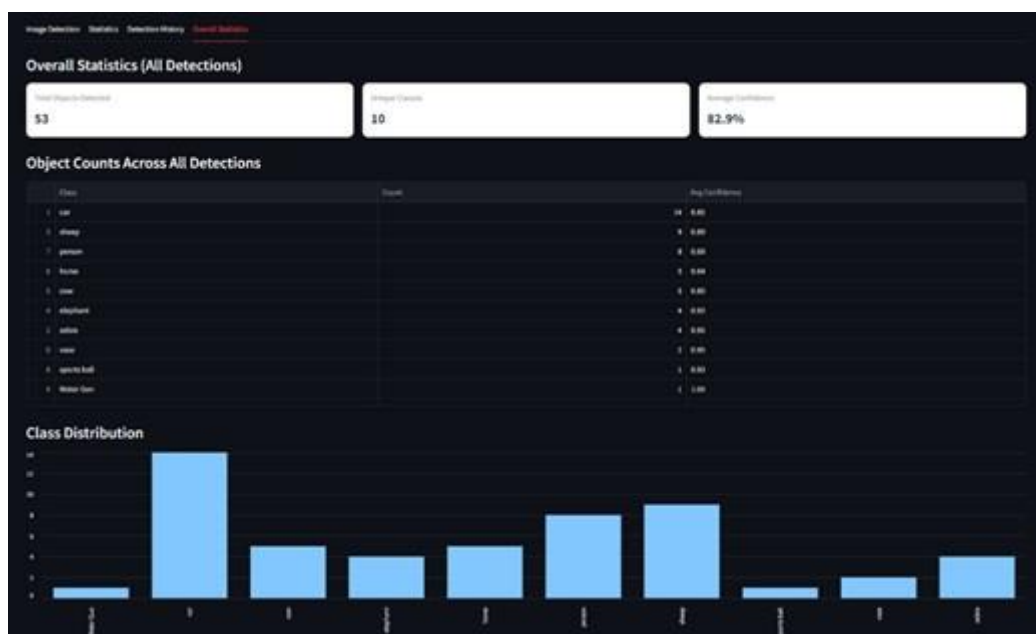
Системата генерира таблица, съдържаща информация за класовете обекти, които са били засечени в рамките на една детекция. Таблицата включва колони за името на класа, броя на срещанията му и средната увереност на модела при разпознаването. Чрез сортиране потребителят може лесно да идентифицира кои обекти се срещат най-често в анализирания изображения. Показано на фиг. 32.



Фиг. 32 Таблица с информация за името на класа, броя на срещанията му и средната увереност на модела

4.7.2 – 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. Показано на фиг. 33



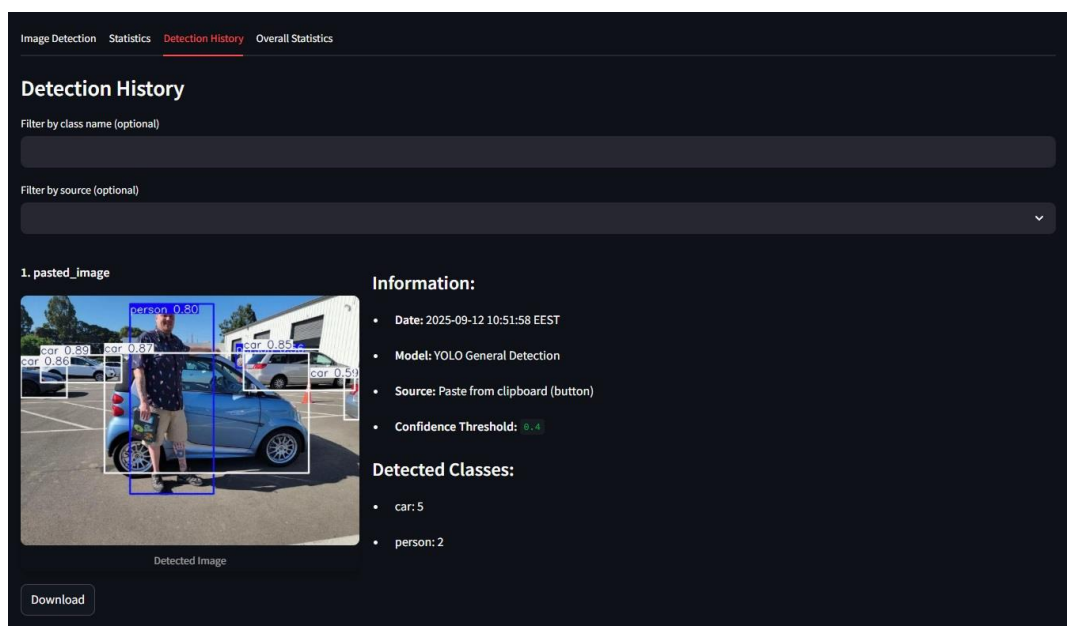
Фиг. 33 Таблица с информация за всички извършени налични записи

4.7.3 – История на детекциите

Системата осигурява 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.

4.7.4 – 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. Показано на фиг. 34.



Фиг. 34 Визуализация и информация на предишни детекции

Глава 5 – Приложимост на дипломната работа

Тази дипломна работа е създадена с цел да демонстрира приложението на съвременни технологии за компютърно зрение и изкуствен интелект в практическа среда.

Приложението може директно да се въведе като допълнителен инструмент в различни области. Например в индустриални условия може да подпомага процеси по автоматизация чрез откриване на дефекти, преброяване на готова продукция или следене на производствени линии.

С наличието на постоянен достъп до интернет, резултатите могат да бъдат съхранени и преглеждани по всяко време и от всяко място. Информацията е лесно четлива благодарение на статистическите модули и визуализациите, което позволява бърз анализ и вземане на решения.

Не е задължително системата да се прилага само в дадена конкретна сфера. Тя може да бъде модифицирана и адаптирана за нуждите на потребителя. По този начин проектът може да бъде основа за по-специализирани приложения като инвентаризация в складове, разпознаване на продукти в магазини или първоначална обработка на медицински изображения.

Възможностите на тази разработка позволяват лесно надграждане и интеграция с вече съществуващи системи. Тя може да служи както като самостоятелен инструмент, така и като допълнение, което подобрява ефективността на работата.

Глава 7 – Изводи и претенции за самостоятелно получени резултати

Заклучение

В настоящата работа беше разработена уеб-базирана система за разпознаване на обекти в изображения, реализирана чрез интегриране на съвременни методи за компютърно зрение и дълбоко обучение. Използвани бяха технологии и инструменти като Python, OpenCV, Streamlit и MongoDB, които осигуриха стабилна основа за изграждане на мащабируема система.

В резултат на извършената работа могат да се направят следните изводи:

- В теоретичния анализ е проведен подробен обзор на съвременното състояние на изследванията в областта на компютърното зрение, като са идентифицирани основните подходи и техните предимства и недостатъци.
- В методология на решението са дефинирани и описани концептуални модели за подсистемите за обучение и разпознаване на обекти, включващи етапите на предварителна обработка, извличане на признаци, обучение и класификация.
- В програмната реализация е изградена функционална уеб-базирана система с модулна архитектура, която позволява гъвкаво разширяване и адаптация към нови изисквания.
- В експерименталната проверка са проведени тестове с реални данни, които установиха, че най-добър баланс между точност и обобщаваща способност се постига в диапазона между 80 и 120 епохи, докато след 120 започва ясно изразено свръхобучение.

Като бъдещо развитие на системата може да се добави поддръжка за видео потоци в реално време и интеграция с облачни услуги за съхранение.

Литературни източници

1. Павлова П., Цифрова обработка на изображения (учебно пособие), Фондация физика, инженерство и медицина XXI, Пловдив 2005
2. Павлова П., Н. Шакев, Компютърно зрение, ТУ София филиал Пловдив, 2018
3. Canny, J. "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986
4. Forsyth, D., Ponce, J. "Computer Vision: A Modern Approach", Pearson, 2011.
5. Girshick, R. *"Rich feature hierarchies for accurate object detection and semantic segmentation"* (R-CNN). CVPR 2014
6. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.
7. Girshick, R. *"Fast R-CNN"*. ICCV 2015
8. Goodfellow, I., Bengio, Y., Courville, A. "Deep Learning." MIT Press, 2016
9. R. Gonzalez, R. Woods, Digital Image Processing, 3rd Edition, Prentice Hall, 2007
10. Liu, W., Anguelov, D., Erhan, D. et al. *"SSD: Single Shot MultiBox Detector"*. ECCV 2016
11. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. *"You Only Look Once: Unified, Real-Time Object Detection"*. CVPR 2016
12. Redmon, J., Farhadi, A. *"YOLO9000: Better, Faster, Stronger"*. CVPR 2017
13. Ren, S., He, K., Girshick, R., Sun, J. *"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"*. NeurIPS 2015
14. Zhao, Z.Q. et al. *"Object Detection with Deep Learning: A Review."* IEEE Transactions on Neural Networks and Learning Systems, 2019
15. Zou, Z. et al. *"Object Detection in 20 Years: A Survey."* Proceedings of the IEEE, 2019
16. <https://arxiv.org/html/2412.05252v1#S5>
17. <https://docs.ultralytics.com/models/yolo11>
18. <https://ravjot03.medium.com/decoding-cnns-a-beginners-guide-to-convolutional-neural-networks-and-their-applications-1a8806cbf536>
19. https://en.wikipedia.org/wiki/Deep_learning

Приложения

Main_app.py

```
import streamlit as st

st.set_page_config(

    page_title="Object Detection",

    layout="wide",

    initial_sidebar_state="expanded"

)

from main_page import main_page

from mongo import users_collection,db

from utils import

login_user,register_user,reset_password,send_username,check_password,is_strong_password,hash_password,is_valid_email

from streamlit_cookies_manager import EncryptedCookieManager

# Session management

cookies = EncryptedCookieManager(password="super-secret-key")

if not cookies.ready():

    st.stop()

# Persistent session check using cookies

if "authenticated" not in st.session_state:
```



```
user_id = cookies.get("user_id")

username = cookies.get("username")

if user_id and username:

    st.session_state.authenticated = True

    st.session_state.username = username

    st.session_state.user_id = user_id

else:

    st.session_state.authenticated = False

    st.session_state.username = None

def centered_input(label, **kwargs):

    col1, col2, col3 = st.columns([1,2,1])

    with col2:

        return st.text_input(label, **kwargs)

def auth_page():

    st.markdown("""

    <style>

    /* Background */

    body {

        background: linear-gradient(135deg, #1e3c72, #2a5298);
```

```
background-attachment: fixed;

}

.centered-title {

    text-align: center;

    font-family: 'Segoe UI', sans-serif;

    color: #222;

}

div[data-baseweb="tab-list"] {

    display: flex;

    justify-content: center;

}

div[data-testid="stTextInput"] {

    margin: 0 auto;

    width: 50% !important;

}

div[data-testid="stTextInput"] > div > div > input {

    border: 1px solid #ccc;

    border-radius: 8px;

    padding: 0.6rem;
```

```
    font-size: 1rem;

}

div[data-testid="stTextInput"] > div > div > input:focus {

    border: 1px solid #007bff;

    box-shadow: 0 0 5px rgba(0,123,255,0.5);

    outline: none;

}

div.stButton > button {

    display: block;

    margin: 1rem auto;

    width: 50% !important;

    border-radius: 25px;

    background: linear-gradient(135deg, #007bff, #00d4ff);

    color: white;

    font-weight: 600;

    padding: 0.6rem;

    border: none;

    transition: 0.3s ease;

}
```

```
div.stButton > button:hover {  
  
    background: linear-gradient(135deg, #0056b3, #0099cc);  
  
    transform: scale(1.02);  
  
}
```

```
label {  
  
    display: block;  
  
    text-align: center;  
  
    font-weight: 600;  
  
}
```

```
</style>
```

```
""", unsafe_allow_html=True)
```

```
st.markdown("<h1 class='centered-title'>User Authentication</h1>",  
unsafe_allow_html=True)
```

```
tab1, tab2, tab3, tab4 = st.tabs(["Login", "Register", "Forgot Password", "Forgot  
Username"])
```

```
# --- Login ---
```

```
with tab1:
```

```
st.markdown("<h3 class='centered-title'>Login</h3>", unsafe_allow_html=True)
```

```
login_username = st.text_input("Username", key="login_user")
```

```
login_password = st.text_input("Password", type="password", key="login_pass")
```

```
if st.button("Login"):
```

```
    user = login_user(login_username, login_password)
```

```
    if user:
```

```
        st.session_state.authenticated = True
```

```
        st.session_state.username = user["username"]
```

```
        st.session_state.user_id = user["user_id"]
```

```
        cookies["user_id"] = user["user_id"]
```

```
        cookies["username"] = user["username"]
```

```
        st.rerun()
```

```
    else:
```

```
        st.error("Invalid username or password.")
```

```
# --- Register ---
```

```
with tab2:
```

```
    st.markdown("<h3 class='centered-title'>Register</h3>",  
unsafe_allow_html=True)
```

```
    reg_username = st.text_input("Username", key="reg_user")
```

```
    reg_email = st.text_input("Email", key="reg_email")
```

```
    reg_password = st.text_input("Password", type="password", key="reg_pass")
```

```
    reg_confirm_password = st.text_input("Confirm Password", type="password",  
key="reg_confirm_pass")
```

```
if st.button("Register"):

    status, msg = register_user(reg_username, reg_email, reg_password,
reg_confirm_password)

    if status:

        st.success("Register Successful")

    else:

        st.error("Register Failed: " + msg)

# --- Forgot Password ---

with tab3:

    st.markdown("<h3 class='centered-title'>Forgot Password?</h3>",
unsafe_allow_html=True)

    reset_email = st.text_input("Enter your registered email", key="reset_email")

    if st.button("Reset Password"):

        success, msg = reset_password(reset_email)

        if success:

            st.success("Password has been sent " + msg)

            st.info("Temporary password printed in console for dev/testing.")

        else:

            st.error("Error " + msg)

# --- Forgot Username ---
```

with tab4:

```
st.markdown("<h3 class='centered-title'>Forgot Username?</h3>",
unsafe_allow_html=True)

lookup_email = st.text_input("Enter your registered email", key="lookup_email")

if st.button("Send Username"):

    success, msg = send_username(lookup_email)

    if success:

        st.success(msg)

        st.info("Check your inbox for your username. (Also printed in console for
dev/testing.)")

    else:

        st.error("Error: " + msg)

def change_password_page():

    st.title("Change Password")

    current_password = st.text_input("Current Password", type="password",
key="cp_current")

    new_password = st.text_input("New Password", type="password", key="cp_new")

    confirm_new_password = st.text_input("Confirm New Password",
type="password", key="cp_confirm")

    if st.button("Change Password", key="cp_button"):

        user = users_collection.find_one({"user_id": st.session_state.user_id})
```

```

if user and check_password(current_password, user['password']):

    if new_password == confirm_new_password and
is_strong_password(new_password):

        hashed = hash_password(new_password)

        users_collection.update_one(

            {"user_id": user["user_id"]},

            {"$set": {"password": hashed}}

        )

        st.success("Password updated successfully.")

    else:

        st.error("New passwords do not match or are not strong enough.")

else:

    st.error("Current password is incorrect.")

def change_email_page():

    st.title("Change Email")

    password_for_email = st.text_input("Password", type="password",
key="ce_password")

    new_email = st.text_input("New Email", key="ce_new_email")

    if st.button("Change Email", key="ce_button"):

        user = users_collection.find_one({"user_id": st.session_state.user_id})

```



```

if user and check_password(password_for_email, user['password']):

    if is_valid_email(new_email) and not users_collection.find_one({"email":
new_email}):

        users_collection.update_one(

            {"user_id": user["user_id"]},

            {"$set": {"email": new_email}}

        )

        st.success("Email updated successfully.")

    else:

        st.error("Invalid or already registered email.")

else:

    st.error("Password is incorrect.")

def logout_page():

    if st.session_state.get("authenticated", False):

        st.write(f"В момента сте влезли като: **{st.session_state.get('username',
'Неизвестен')}**")

        st.write(f"User ID: `{st.session_state.get('user_id', 'N/A')}`")

        user_id = st.session_state.get("user_id")

        if user_id:

            user_data = users_collection.find_one({"user_id": user_id}, {"email": 1})

```

```
if user_data and "email" in user_data:

    st.write(f"Email: {user_data['email']}")

else:

    st.write("Email: (няма намерен имейл)")

else:

    st.write("Email: (липсва user_id)")

else:

    st.info("Не сте влезли в профил.")

st.write("Натиснете бутона по-долу, за да излезете от профила си.")

if st.button("Logout", key="logout_button"):

    try:

        del cookies["user_id"]

        del cookies["username"]

    except Exception:

        pass

    st.session_state.authenticated = False

    st.session_state.username = None

    st.session_state.user_id = None

    st.success("You have been logged out.")
```

```
st.rerun()

if not st.session_state.authenticated:

    auth_page()

else:

    pages = {

        "Your account": [

            st.Page(main_page, title="Home"),

            st.Page(change_password_page, title="Change Password"),

            st.Page(change_email_page, title="Change Email"),

            st.Page(logout_page, title="Logout"),

        ]

    }

    pg = st.navigation(pages, position="top",expanded=False)

    pg.run()
```

utils.py

```
import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

import streamlit as st
```

```
import bcrypt

import string

import re

import random

import uuid

from mongo import users_collection

EMAIL_REGEX = r"^[\\w\\.]+@[\\w\\.]+\\.\\w+$"

PASSWORD_REGEX = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).{6,}$"

def is_valid_email(email):

    return re.match(EMAIL_REGEX, email)

def is_strong_password(password):

    return re.match(PASSWORD_REGEX, password)

def hash_password(password):

    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

def check_password(password, hashed):

    return bcrypt.checkpw(password.encode('utf-8'), hashed)

def register_user(username, email, password, confirm_password):

    username = username.lower().strip()
```

```
if password != confirm_password:

    return False, "Passwords do not match."

if users_collection.find_one({"username": username}):

    return False, "Username already exists."

if users_collection.find_one({"email": email}):

    return False, "Email already registered."

if not is_valid_email(email):

    return False, "Invalid email format."

if not is_strong_password(password):

    return False, "Password must contain at least one uppercase letter, one lowercase
letter, one number, and be at least 6 characters long."

hashed = hash_password(password)

user_id = str(uuid.uuid4())

users_collection.insert_one({

    "user_id": user_id,

    "username": username,

    "email": email,

    "password": hashed

})

return True, "User registered successfully."
```

```
def login_user(username, password):

    username = username.lower().strip()

    user = users_collection.find_one({"username": username})

    if user and check_password(password, user['password']):

        return user

    return None

def get_username_by_email(email):

    user = users_collection.find_one({"email": email})

    if user:

        return user["username"]

    return None

def send_username(email):

    user = users_collection.find_one({"email": email})

    if not user:

        return False, "Email not found."

    username = user["username"]

    subject = "Username Recovery - Object Detection System"

    body = f""""Hello,

Your registered username is: {username}"""
```

If you didn't request this, please ignore this email.

"""

email_sent = send_email(email, subject, body)

if email_sent:

return True, "Your username has been sent to your email."

else:

print(f"[DEBUG] Username for {email}: {username}")

return False, "We couldn't send the email. Your username is printed in the server console."

def reset_password(email):

user = users_collection.find_one({"email": email})

if not user:

return False, "Email not found."

new_pass = generate_temp_password()

hashed = hash_password(new_pass)

users_collection.update_one({"email": email}, {"\$set": {"password": hashed}})

subject = "Password Reset - Object Detection System"

body = f""""Hello {user['username']},

Your temporary password is: {new_pass}

Log in with this password and change it immediately.

```
"""

email_sent = send_email(email, subject, body)

if email_sent:

    return True, "A new temporary password has been sent to your email."

else:

    print(f"[DEBUG] Temporary password for {email}: {new_pass}")

    return False, "We couldn't send the email. Use the temporary password printed in
the server console."

def generate_temp_password(length=10):

    return ''.join(random.choices(string.ascii_letters + string.digits, k=length))

SMTP_SERVER = "smtp.gmail.com"

SMTP_PORT = 587

SENDER_EMAIL = "objectdetectionsystem1@gmail.com"

SENDER_PASSWORD = "tzvgqobgzseiyyng" #AppPassword

def send_email(recipient_email, subject, body):

    try:

        msg = MIMEMultipart()

        msg["From"] = SENDER_EMAIL

        msg["To"] = recipient_email

        msg["Subject"] = subject
```



```

msg.attach(MIMEText(body, "plain"))

with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:

    server.ehlo()

    server.starttls()

    server.ehlo()

    server.login(SENDER_EMAIL, SENDER_PASSWORD)

    server.send_message(msg)

return True

except Exception as e:

    error_msg = f"[EMAIL ERROR] {type(e).__name__}: {str(e)}"

    print(error_msg)

    st.error(error_msg)

return False

```

mongo.py

```

from pymongo import MongoClient
# MongoDB Configuration
MONGO_URI = "mongodb://localhost:27017"
DB_NAME = "object_detection"
# Connect to MongoDB
client = MongoClient(MONGO_URI)
db = client[DB_NAME]
USERS_COLLECTION = "users"

```

```
users_collection = db[USERS_COLLECTION]
```

main_page.py

```
import cv2
import sys
from pathlib import Path
from ultralytics import YOLO
from PIL import Image
from datetime import datetime, timezone
from tzlocal import get_localzone
from st_img_pastebutton import paste
import pytz
import pandas as pd
import numpy as np
import io
import tensorflow as tf
import base64
import streamlit as st
from utils import check_password, is_valid_email
from mongo import users_collection, db

FILE = Path(__file__).resolve()
ROOT = FILE.parent
if ROOT not in sys.path:
    sys.path.append(str(ROOT))
ROOT = ROOT.relative_to(Path.cwd())

# Image Config
IMAGES_DIR = ROOT / 'images'
DEFAULT_IMAGE = IMAGES_DIR / 'image2.jpg'
```

```

DEFAULT_DETECT_IMAGE = IMAGES_DIR / 'detectedimage2.jpg'
# Model Configurations
MODEL_DIR = ROOT / 'weights'
DETECTION_MODEL = MODEL_DIR / 'yolo11n.pt'
SEGMENTATION_MODEL = MODEL_DIR / 'yolo11n-seg.pt'
CUSTOM_YOLO_MODEL = MODEL_DIR / 'my_model.pt'
CUSTOM_MODEL_PATH = MODEL_DIR / 'simple_object_detection.h5'
@st.cache_resource
def load_yolo_model(path):
    return YOLO(path)
@st.cache_resource
def load_tf_model(path):
    return tf.keras.models.load_model(path, compile=False)
# preload models
yolo_det_model = load_yolo_model(DETECTION_MODEL)
yolo_seg_model = load_yolo_model(SEGMENTATION_MODEL)
yolo_custom_model = load_yolo_model(CUSTOM_YOLO_MODEL)
try:
    tf_custom_model = load_tf_model(CUSTOM_MODEL_PATH)
except Exception as e:
    tf_custom_model = None
def main_page():
    # CSS for styling
    st.markdown("""
        <style>
            .main {
                background-color: #f8f9fa;
            }
            .sidebar .sidebar-content {

```

```
    background-color: #343a40;
    color: white;
}

.stButton>button {
    background-color: #4CAF50;
    color: white;
    border-radius: 5px;
    padding: 10px 24px;
    font-weight: bold;
    width: 100%;
}

.stButton>button:hover {
    background-color: #45a049;
}

.stSelectbox, .stSlider {
    margin-bottom: 20px;
}

.stImage {
    border-radius: 10px;
    box-shadow: 0 4px 8px 0 rgba(0,0,0,0.2);
}

.stDataFrame {
    border-radius: 10px;
    box-shadow: 0 4px 8px 0 rgba(0,0,0,0.1);
}

.header-text {
    font-size: 2.5rem;
    font-weight: 700;
    color: #2c3e50;
```

```
        margin-bottom: 1rem;
    }
    .subheader-text {
        font-size: 1.2rem;
        color: #7f8c8d;
        margin-bottom: 2rem;
    }
    .metric-card {
        background: white;
        border-radius: 10px;
        padding: 15px;
        box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        margin-bottom: 20px;
    }
    .metric-title {
        font-size: 0.9rem;
        color: #7f8c8d;
        margin-bottom: 5px;
    }
    .metric-value {
        font-size: 1.5rem;
        font-weight: bold;
        color: #2c3e50;
    }
    .detection-button-container {
        margin-top: 2rem;
        text-align: center;
    }
    .paste-container {
```

```
border: 2px dashed #ccc;
border-radius: 5px;
padding: 20px;
text-align: center;
margin-bottom: 20px;
}
```

```
</style>
```

```
""", unsafe_allow_html=True)
```

```
if "page" not in st.session_state:
```

```
    st.session_state.page = "home"
```

```
if "account_action" not in st.session_state:
```

```
    st.session_state.account_action = "Home"
```

```
elif st.session_state.page == "change_email":
```

```
    st.title("Change Email")
```

```
    password_for_email = st.text_input("Password", type="password")
```

```
    new_email = st.text_input("New Email")
```

```
if st.button("Change Email"):
```

```
    user = users_collection.find_one({"user_id": st.session_state.user_id})
```

```
    if user and check_password(password_for_email, user['password']):
```

```
        if is_valid_email(new_email) and not users_collection.find_one({"email":
new_email}):
```

```
            users_collection.update_one(
                {"user_id": user["user_id"]},
                {"$set": {"email": new_email}}
            )
```

```
            st.success("Email updated successfully.")
```

```
        else:
```

```
            st.error("Invalid or already registered email.")
```

```

        else:
            st.error("Password is incorrect.")
        st.stop()
# SideBar
with st.sidebar:
    st.markdown("""
        <style>
            .sidebar .sidebar-content {
                background-image: linear-gradient(#343a40,#2c3e50);
                color: white;
            }
            .sidebar .stRadio label {
                color: white;
            }
            .sidebar .stSlider label {
                color: white;
            }
            .sidebar .stFileUploader label {
                color: white;
            }
        </style>
        """, unsafe_allow_html=True)
    st.header("Model Configuration")

# Choose Model: Detection or Segmentation
model_type = st.radio(
    "Select Task Type:",
    ["YOLO General Detection", "YOLO General Segmentation", "YOLO Model
(Specific)", "TensorFlow Model (Specific)"],

```

```

    index=0,
    help="Choose between object detection and instance segmentation"
)
st.markdown("---")
# Select Confidence Value
confidence_value = st.slider(
    "Confidence Threshold",
    min_value=0,
    max_value=100,
    value=40,
    help="Adjust the minimum confidence level for detections"
)
confidence_value = float(confidence_value) / 100
# Class Selection
CLASSES = [
    "person", "bicycle", "car", "motorcycle", "airplane", "bus", "train", "truck",
"boat",
    "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird",
"cat",
    "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
"backpack",
    "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports
ball",
    "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", "tennis
racket",
    "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana",
"apple",
    "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
"chair",

```



```

        "couch", "potted plant", "bed", "dining table", "toilet", "tv", "laptop", "mouse",
"remote",
        "keyboard", "cell phone", "microwave", "oven", "toaster", "sink",
"refrigerator", "book",
        "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"
    ]
    # Image Source Selection
    st.subheader("Image Source")
    image_source = st.radio(
        "Select image source:",
        ["Upload an image", "Paste from clipboard (text input)", "Paste from clipboard
(button)"],
        index=0,
        help="Choose how to provide the image for detection"
    )

    # Image Upload/Paste Section
    st.subheader("Image Configuration")
    source_image = None
    if image_source == "Upload an image":
        source_image = st.file_uploader(
            "Upload an image",
            type=("jpg", "png", "jpeg", "bmp", "webp"),
            help="Upload an image for object detection",
            key="file_uploader"
        )
    elif image_source == "Paste from clipboard (text input)":
        paste_data = st.text_area("Paste image here (as base64 or URL)", "",
height=100, key="paste_area")

```

```

if paste_data:
    try:
        if paste_data.startswith("data:image"):
            header, encoded = paste_data.split(",", 1)
            image_data = base64.b64decode(encoded)
            source_image = io.BytesIO(image_data)
        elif paste_data.startswith(("http://", "https://")):
            import requests
            from io import BytesIO
            response = requests.get(paste_data)
            source_image = BytesIO(response.content)
        else:
            image_data = base64.b64decode(paste_data)
            source_image = io.BytesIO(image_data)
    except:
        st.error("Could not process the pasted image. Please try another method.")
elif image_source == "Paste from clipboard (button)":
    image_data = paste(label="Paste from Clipboard", key="image_clipboard")
    if image_data is not None:
        try:
            header, encoded = image_data.split(",", 1)
            binary_data = base64.b64decode(encoded)
            source_image = io.BytesIO(binary_data)
            st.image(source_image, caption="Pasted from Clipboard",
use_container_width=True)
        except Exception as e:
            st.error(f"Failed to process clipboard image: {e}")

# Selecting Detection or Segmentation Model
if model_type == 'YOLO General Detection':

```

```

    model = yolo_det_model
    class_map = model.names
elif model_type == 'YOLO General Segmentation':
    model = yolo_seg_model
    class_map = model.names
elif model_type == 'YOLO Model (Specific)':
    model = yolo_custom_model
    class_map = model.names
elif model_type == 'TensorFlow Model (Specific)':
    model = tf_custom_model
    class_map = None
tab1, tab2, tab3, tab4 = st.tabs(["Image Detection", "Statistics", "Detection History",
"Overall Statistics"])
with tab1:
    col1, col2 = st.columns(2)
    with col1:
        st.subheader("Input Image")
        try:
            if source_image is None:
                default_image_path = str(DEFAULT_IMAGE)
                default_image = Image.open(default_image_path)
                st.image(default_image_path,
                    caption="Default Image - Upload or paste your own image to see
detection results",
                    use_container_width=True)
            else:
                if isinstance(source_image, io.BytesIO):
                    # Reset pointer to start if it's a BytesIO object
                    source_image.seek(0)

```

```

        uploaded_image = Image.open(source_image)
        st.image(uploaded_image,
                  caption="Input Image - Click 'Detect Objects' to process",
                  use_container_width=True)
    except Exception as e:
        st.error("Error Occurred While Opening the Image")
        st.error(e)
with col2:
    st.subheader("Detection Results")
    try:
        if source_image is None:
            default_detected_image_path = str(DEFAULT_DETECT_IMAGE)
            default_detected_image = Image.open(default_detected_image_path)
            st.image(default_detected_image_path,
                      caption="Sample Detection - Upload or paste your own image to see
live results",
                      use_container_width=True)
        except Exception as e:
            st.error("Error Occurred While Processing the Image")
            st.error(e)
    st.markdown("---")
def update_width_from_slider():
    st.session_state.resize_width = st.session_state.width_slider
    st.session_state.width_input = st.session_state.width_slider
def update_width_from_input():
    st.session_state.resize_width = st.session_state.width_input
    st.session_state.width_slider = st.session_state.width_input
def update_height_from_slider():
    st.session_state.resize_height = st.session_state.height_slider

```

```

    st.session_state.height_input = st.session_state.height_slider
def update_height_from_input():
    st.session_state.resize_height = st.session_state.height_input
    st.session_state.height_slider = st.session_state.height_input
# Sidebar toggle
resize_mode = st.sidebar.radio(
    "Image Resize Mode",
    ("Disabled", "Enabled"),
    index=0
)
if source_image is not None and resize_mode == "Enabled":
    st.subheader("Image Resize")
    # Get original image size
    try:
        if isinstance(source_image, io.BytesIO):
            source_image.seek(0)
            temp_img = Image.open(source_image)
            orig_width, orig_height = temp_img.size
        except Exception:
            orig_width, orig_height = 640, 480 # fallback
    st.info(f"Original Size: {orig_width} x {orig_height} px")
    # Initialize session state
    if "resize_width" not in st.session_state:
        st.session_state.resize_width = orig_width
    if "resize_height" not in st.session_state:
        st.session_state.resize_height = orig_height
    # Checkbox for aspect ratio
    keep_aspect = st.checkbox("Keep Aspect Ratio", value=True)
    aspect_ratio = orig_width / orig_height

```

```

if keep_aspect:
    # --- Single width slider (separate key) ---
    new_width = st.slider(
        "Width (px)",
        min_value=50,
        max_value=5000,
        value=st.session_state.get("resize_width", orig_width),
        step=1,
        key="aspect_width_slider"
    )
    new_height = int(new_width / aspect_ratio)
    st.session_state.resize_width = new_width
    st.session_state.resize_height = new_height
    st.write(f"Auto-scaled Height: {new_height}px")
else:
    # --- Independent width/height sliders ---
    col_w1, col_w2 = st.columns([3, 1])
    with col_w1:
        st.slider(
            "Width (px)",
            min_value=50,
            max_value=5000,
            value=st.session_state.resize_width,
            step=1,
            key="width_slider",
            on_change=update_width_from_slider
        )
    with col_w2:
        st.number_input(

```

```

        "Width",
        min_value=50,
        max_value=5000,
        value=st.session_state.resize_width,
        step=1,
        key="width_input",
        on_change=update_width_from_input
    )
col_h1, col_h2 = st.columns([3, 1])
with col_h1:
    st.slider(
        "Height (px)",
        min_value=50,
        max_value=5000,
        value=st.session_state.resize_height,
        step=1,
        key="height_slider",
        on_change=update_height_from_slider
    )
with col_h2:
    st.number_input(
        "Height",
        min_value=50,
        max_value=5000,
        value=st.session_state.resize_height,
        step=1,
        key="height_input",
        on_change=update_height_from_input
    )

```

```

# --- Apply resize ---
try:
    if isinstance(source_image, io.BytesIO):
        source_image.seek(0)
        img_pil = Image.open(source_image)
        img_np = cv2.cvtColor(np.array(img_pil), cv2.COLOR_RGB2BGR)
        resized_img = cv2.resize(
            img_np,
            (st.session_state.resize_width, st.session_state.resize_height),
            interpolation=cv2.INTER_AREA
        )
        resized_pil = Image.fromarray(cv2.cvtColor(resized_img,
cv2.COLOR_BGR2RGB))
        buf = io.BytesIO()
        resized_pil.save(buf, format="JPEG")
        buf.seek(0)
        source_image = buf # update source image
        st.image(
            resized_pil,
            caption=f"Resized Image
({st.session_state.resize_width}x{st.session_state.resize_height})",
            use_container_width=True
        )
    except Exception as e:
        st.error(f"Error resizing image: {e}")
    if source_image is not None:
        st.markdown('<div class="detection-button-container">',
unsafe_allow_html=True)

```



```

if st.button(f"Detect Objects ({model_type})", key="detect_button"):
    with st.spinner(f"Processing {model_type}..."):
        try:
            uploaded_image = Image.open(source_image)

            # ----- YOLO (Detection / Segmentation / Custom YOLO) -----

            if model_type in ["YOLO General Detection", "YOLO General
Segmentation", "YOLO Model (Specific)"]:
                result = model.predict(uploaded_image, conf=confidence_value)
                boxes = result[0].boxes
                result_plotted = result[0].plot()[ :, :, ::-1]
                with tab1:
                    with col2:
                        st.image(result_plotted,
                                caption=f"{model_type} Results (Confidence:
{confidence_value*100:.1f}%)",
                                use_container_width=True)

                class_counts = { }
                for box in boxes:
                    class_id = int(box.cls)
                    class_name = class_map[class_id]
                    class_counts[class_name] = class_counts.get(class_name, 0) + 1
                image_width, image_height = uploaded_image.size
                detection_data = {
                    "timestamp": datetime.now(timezone.utc),
                    "model_type": model_type,
                    "confidence_threshold": confidence_value,
                    "source": image_source,
                    "image_name": source_image.name if image_source == "Upload an
image" and hasattr(source_image, 'name') else "pasted_image",

```

```

        "image_resolution": {"width": image_width, "height":
image_height},
        "object_counts": class_counts,
        "objects": [],
        "user_id": st.session_state.user_id
    }

    for i, box in enumerate(boxes):
        detection_data["objects"].append({
            "object_id": i,
            "class": class_map[int(box.cls)],
            "confidence": float(box.conf.item() if hasattr(box.conf, "item")
else box.conf),
            "box_xywh": box.xywh.tolist()[0]
        })
    image_pil = Image.fromarray(result_plotted)
    buffered = io.BytesIO()
    image_pil.save(buffered, format="JPEG")
    detection_data["image_bytes"] =
base64.b64encode(buffered.getvalue()).decode("utf-8")
    unique_classes = list(class_counts.keys())
    if len(unique_classes) == 0:
        target_collection_name = "no_detections"
    elif len(unique_classes) == 1:
        target_collection_name = unique_classes[0].replace(" ", "_")
    else:
        target_collection_name = "Multiclass_objects"
    try:
        db[target_collection_name].insert_one(detection_data)

```

```

        st.success(f"Detection saved to database collection")
except Exception as e:
    st.error("Failed to store detection in MongoDB:")
    st.error(e)
st.session_state.detection_results = {
    "image": result_plotted,
    "counts": class_counts,
    "boxes": boxes,
    "model_type": model_type,
    "class_map": class_map
}

# ----- Custom TensorFlow Model -----
elif model_type == "TensorFlow Model (Specific)":
    if model is None:
        st.error("Custom model not loaded.")
    else:
        img = uploaded_image.convert("L")
        img = img.resize((340, 340))
        img_arr = np.array(img) / 255.0
        img_arr = np.expand_dims(img_arr, axis=(0, -1))

        pred_bbox, pred_class_logits = model.predict(img_arr)
        pred_bbox = np.squeeze(pred_bbox)
        pred_class = np.argmax(pred_class_logits, axis=-1)[0]

        w, h = uploaded_image.size
        xmin, ymin, xmax, ymax = (
            int(pred_bbox[0] * w),
            int(pred_bbox[1] * h),

```

```

        int(pred_bbox[2] * w),
        int(pred_bbox[3] * h),
    )
    img_np = np.array(uploaded_image)
    cv2.rectangle(img_np, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)
    label = "Water Gun"
    cv2.putText(img_np, label, (xmin, max(ymin - 10, 0)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 255), 2)
    with tab1:
        with col2:
            st.image(img_np, caption="Custom Model Detection",
use_container_width=True)
    detection_data = {
        "timestamp": datetime.now(timezone.utc),
        "model_type": model_type,
        "confidence_threshold": confidence_value,
        "source": image_source,
        "image_name": source_image.name if hasattr(source_image,
'name') else "pasted_image",
        "image_resolution": {"width": w, "height": h},
        "object_counts": {"Water Gun": 1},
        "objects": [{
            "object_id": 0,
            "class": "Water Gun",
            "confidence": float(np.max(pred_class_logits)),
            "box_xyxy": [xmin, ymin, xmax, ymax]
        }],
        "user_id": st.session_state.user_id
    }

```

```

        buffered = io.BytesIO()
        Image.fromarray(img_np).save(buffered, format="JPEG")
        detection_data["image_bytes"] =
base64.b64encode(buffered.getvalue()).decode("utf-8")
        try:
            db["Water_Gun"].insert_one(detection_data)
            st.success("Detection saved to MongoDB collection:
'Water_Gun'")
        except Exception as e:
            st.error("Failed to store detection in MongoDB:")
            st.error(e)
    except Exception as e:
        st.error("Error during detection:")
        st.error(e)
    st.markdown('</div>', unsafe_allow_html=True)
with tab2:
    if 'detection_results' not in st.session_state:
        st.warning("Run a detection first to see statistics")
    else:
        results = st.session_state.detection_results
        st.subheader(f'{results["model_type"]} Statistics')
        col1, col2, col3 = st.columns(3)
        with col1:
            st.markdown('<div class="metric-card"><div class="metric-title">Total
Objects Detected</div>'
                        f'<div class="metric-value">{sum(results.get("counts",
{ }).values())}</div></div>',
                        unsafe_allow_html=True)
        with col2:

```

```

        st.markdown('<div class="metric-card"><div class="metric-title">Unique
Classes</div>'
                    f'<div class="metric-value">{len(results.get("counts",
{}))}</div></div>',
                    unsafe_allow_html=True)

    with col3:
        st.markdown('<div class="metric-card"><div class="metric-title">Model
Confidence</div>'
                    f'<div class="metric-
value">{confidence_value*100:.0f}%</div></div>',
                    unsafe_allow_html=True)

    # ---- Statistics display Object Counts in a Table with sorting ----
    st.subheader("Object Counts")
    # Collect confidences per class (works even if boxes empty)
    confidences_by_class = { }
    for box in results.get("boxes", []):
        cls_map = results.get("class_map", CLASSES)
        cls_name = cls_map[int(box.cls)]
        conf = float(box.conf.item() if hasattr(box.conf, "item") else box.conf)
        confidences_by_class.setdefault(cls_name, []).append(conf)

    # Build dataframe
    data_rows = []
    for cls_name, count in results.get("counts", { }).items():
        if isinstance(cls_name, int):
            cls_name = CLASSES[cls_name]
            avg_conf = np.mean(confidences_by_class.get(cls_name, [])) if cls_name in
confidences_by_class else 0
            data_rows.append({

```

```

        "Class": cls_name,
        "Count": count,
        "Avg Confidence": f"{avg_conf:.2f}"
    })
count_df = pd.DataFrame(data_rows)
if not count_df.empty and "Count" in count_df.columns:
    st.dataframe(
        count_df.sort_values("Count", ascending=False),
        use_container_width=True,
        height=min(400, 50 + 35 * len(count_df))
    )
else:
    st.info("No objects detected in this image.")
    st.dataframe(pd.DataFrame(columns=["Class", "Count", "Avg
Confidence"])),
        use_container_width=True,
        height=100)
with st.expander("Detailed Detection Data"):
    if not results.get("boxes"):
        st.write("No detection boxes available.")
    else:
        st.write("Raw detection data from the model:")
        for i, box in enumerate(results["boxes"]):
            st.json({
                "object_id": i,
                "class": CLASSES[int(box.cls)],
                "confidence": float(box.conf),
                "coordinates": box.xywh.tolist()
            })

```

with tab3:

```
st.subheader("Detection History")
```

```
local_tz = get_localzone()
```

```
import history
```

```
user_id = st.session_state.user_id
```

```
class_filter = st.text_input("Filter by class name (optional)")
```

```
source_filter = st.selectbox(
```

```
    "Filter by source (optional)",
```

```
    [
```

```
        "",
```

```
        "Upload an image",
```

```
        "Paste from clipboard (text input)",
```

```
        "Paste from clipboard (button)"
```

```
    ]
```

```
)
```

```
history_docs = history.get_detection_history(
```

```
    user_id=user_id,
```

```
    class_filter=class_filter if class_filter else None,
```

```
    source_filter=source_filter if source_filter else None,
```

```
    limit=100
```

```
)
```

```
history_docs = sorted(
```

```
    history_docs,
```

```
    key=lambda doc: doc.get("timestamp",
```

```
datetime.min.replace(tzinfo=timezone.utc)),
```

```
    reverse=True
```

```
)
```

```
if not history_docs:
```



```

        st.info("No detection history found.")
    else:
        for i, doc in enumerate(history_docs):
            col1, col2 = st.columns([1, 2])
            with col1:
                st.markdown(f"***{i+1}. {doc.get('image_name', 'Unnamed Image')}***")
                image_data = doc.get("image_bytes")
                if image_data:
                    try:
                        decoded_image = base64.b64decode(image_data)
                        image = Image.open(io.BytesIO(decoded_image))
                        st.image(image, caption="Detected Image",
use_container_width=True)

                        # --- Download button ---
                        st.download_button(
                            label="Download",
                            data=decoded_image,
                            file_name=doc.get("image_name", f"detection_{i+1}.jpg"),
                            mime="image/jpeg",
                            key=f"download_btn_{i}",
                            use_container_width=False
                        )
                    except Exception as e:
                        st.warning(f"Failed to decode or display image: {e}")

            with col2:
                st.markdown("#### Information:")
                utc_time = doc.get("timestamp")
                if utc_time.tzinfo is None:
                    utc_time = utc_time.replace(tzinfo=pytz.UTC)

```

```

local_time = utc_time.astimezone(local_tz)
local_time_str = local_time.strftime("%Y-%m-%d %H:%M:%S %Z")
st.markdown(f"- **Date:** {local_time_str}")
st.markdown(f"- **Model:** {doc.get('model_type')}")
st.markdown(f"- **Source:** {doc.get('source')}")
confidence_threshold = doc.get("confidence_threshold", "N/A")
st.write("- **Confidence Threshold:**", confidence_threshold)
st.markdown("#### Detected Classes:")
for cls, count in doc.get("object_counts", {}).items():
    st.markdown(f"- {cls}: {count}")
st.markdown("---")
def get_all_collections():
    return db.list_collection_names()
def get_detection_history(user_id, class_filter=None, source_filter=None,
limit=100):
    results = []
    for col_name in get_all_collections():
        if col_name == "users":
            continue
        query = {"user_id": user_id}
        if class_filter:
            query["object_counts." + class_filter] = {"$exists": True}
        if source_filter:
            query["source"] = source_filter
        collection = db[col_name]
        docs = collection.find(query).sort("timestamp", -1).limit(limit)
        for doc in docs:
            doc["collection"] = col_name
            results.append(doc)

```

```

    return results

def history_to_dataframe(docs):
    rows = []
    for doc in docs:
        timestamp = doc.get("timestamp")
        model_type = doc.get("model_type")
        source = doc.get("source")
        image_name = doc.get("image_name")
        collection = doc.get("collection", "")
        for cls, count in doc.get("object_counts", {}).items():
            rows.append({
                "Date": timestamp,
                "Class": cls,
                "Count": count,
                "Model": model_type,
                "Source": source,
                "Image": image_name,
                "Collection": collection
            })
    return pd.DataFrame(rows)

with tab4:
    st.subheader("Overall Statistics (All Detections)")
    user_id = st.session_state.user_id
    all_docs = get_detection_history(user_id=user_id, limit=500)
    if not all_docs:
        st.warning("No detection history found for this user.")
    else:
        total_objects = 0
        class_counts = {}

```

```

confidences_by_class = {}
for doc in all_docs:
    counts = doc.get("object_counts", {})
    for cls, cnt in counts.items():
        total_objects += cnt
        class_counts[cls] = class_counts.get(cls, 0) + cnt

    for obj in doc.get("objects", []):
        cls = obj["class"]
        conf = obj["confidence"]
        if cls not in confidences_by_class:
            confidences_by_class[cls] = []
        confidences_by_class[cls].append(conf)
col1, col2, col3 = st.columns(3)
with col1:
    st.markdown('<div class="metric-card"><div class="metric-title">Total
Objects Detected</div>'
                f'<div class="metric-value">{total_objects}</div></div>',
    unsafe_allow_html=True)
with col2:
    st.markdown('<div class="metric-card"><div class="metric-title">Unique
Classes</div>'
                f'<div class="metric-value">{len(class_counts)}</div></div>',
    unsafe_allow_html=True)
with col3:
    avg_conf = np.mean([c for lst in confidences_by_class.values() for c in lst])
    if confidences_by_class else 0
    st.markdown('<div class="metric-card"><div class="metric-title">Average
Confidence</div>'

```

```

        f'<div class="metric-value">{avg_conf*100:.1f}%</div></div>',
unsafe_allow_html=True)
    rows = []
    for cls, count in class_counts.items():
        avg_conf = np.mean(confidences_by_class.get(cls, [])) if cls in
confidences_by_class else 0
        rows.append({
            "Class": cls,
            "Count": count,
            "Avg Confidence": f"{avg_conf:.2f}"
        })
    df_overall = pd.DataFrame(rows)
    st.subheader("Object Counts Across All Detections")
    st.dataframe(
        df_overall.sort_values("Count", ascending=False),
        use_container_width=True,
        height=min(500, 50 + 35 * len(df_overall))
    )
    st.subheader("Class Distribution")
    st.bar_chart(df_overall.set_index("Class")["Count"])

```

Simple_obj_detect_gun_v2.ipynb

```

import os
import random
import xml.etree.ElementTree as ET
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt

```

```
import matplotlib.patches as patches
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
import skimage
from skimage import io
import bs4
from bs4 import BeautifulSoup

# Print versions
versions = {
    'numpy': np.__version__,
    'tensorflow': tf.__version__,
    'matplotlib': matplotlib.__version__,
    'pandas': pd.__version__,
    'sklearn': sklearn.__version__,
    'skimage': skimage.__version__,
    'beautifulsoup4': bs4.__version__
}
for lib, version in versions.items():
    print(f'{lib}: {version}')

# Determinism
tf.config.experimental.enable_op_determinism
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

def load_xml_annotations(directory, fn_parse_annotation):
    annotations = []
```

```

for filename in os.listdir(directory):
    if filename.endswith('.xml'):
        file_path = os.path.join(directory, filename)
        tree = ET.parse(file_path)
        annotation_item = fn_parse_annotation(tree.getroot())
        annotations.append(annotation_item)
    return annotations

def display_images(df_original, num_elements):
    if num_elements == 1:
        fig, ax = plt.subplots(1, 2, figsize=(10, 4))
        ax[0].set_title('Input'); ax[1].set_title('Expected result')
    else:
        fig, ax = plt.subplots(num_elements, 2, figsize=(10, 4 * num_elements))
        ax[0, 0].set_title('Input'); ax[0, 1].set_title('Expected result')
    for i in range(num_elements):
        display_data = df_original.iloc[i]
        image = io.imread(display_data["filename"])
        if num_elements == 1:
            ax[0].imshow(image); ax[1].imshow(image)
            w = display_data['xmax'] - display_data['xmin']
            h = display_data['ymax'] - display_data['ymin']
            ax[1].add_patch(patches.Rectangle((display_data['xmin'],
display_data['ymin']),
                                            w, h, fill=False, color='lime', linewidth=2))
            for a in ax: a.set_xticks([]); a.set_yticks([])
        else:
            ax[i, 0].imshow(image); ax[i, 1].imshow(image)
            w = display_data['xmax'] - display_data['xmin']
            h = display_data['ymax'] - display_data['ymin']

```

```

ax[i, 1].add_patch(patches.Rectangle((display_data['xmin'],
display_data['ymin'],
w, h, fill=False, color='lime', linewidth=2))

for a in (ax[i,0], ax[i,1]): a.set_xticks([]); a.set_yticks([])
plt.tight_layout(); plt.show()
def plot_history(history, model_evaluation_info=""):
    epochs_range = range(1, len(history.history['loss'])+1)
    plt.figure(num='Training Result', figsize=(12, 6))
    if model_evaluation_info:
        plt.suptitle(model_evaluation_info, fontsize=13)
    # Loss (total)
    plt.subplot(1, 2, 2)
    plt.title('Total Loss')
    plt.plot(epochs_range, history.history['loss'], label='Train Loss')
    plt.plot(epochs_range, history.history['val_loss'], label='Val Loss')
    plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend(loc='upper right')
    plt.tight_layout(); plt.subplots_adjust(wspace=0.2); plt.show()
def load_normalize_image(image_path, img_size, channels=1):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=channels)
    image = tf.image.resize(image, size=img_size)
    image = tf.cast(image, tf.float32) / 255.0
    return image
def create_dataset_from_dataframe(dataframe, use_norm_cols=True):
    image_paths = tf.cast(dataframe['filename'].values, tf.string)
    if use_norm_cols:
        cols = ['xmin_n', 'ymin_n', 'xmax_n', 'ymax_n']
    else:
        cols = ['xmin', 'ymin', 'xmax', 'ymax']

```



```

image_coordinates = tf.cast(dataframe[cols].values, tf.float32)
class_ids = tf.cast(dataframe['class_id'].values, tf.int32)
# Output structure must match model outputs order: ('bbox', 'class')
ds = tf.data.Dataset.from_tensor_slices((image_paths, (image_coordinates,
class_ids)))

return ds

def plot_predictions(pred_bboxes, pred_classes, dataframe_original, box_color='lime',
class_names=None):
    num_predictions = len(pred_bboxes)
    fig, ax = plt.subplots(num_predictions, 2, figsize=(12, num_predictions * 4))
    if num_predictions == 1:
        ax = np.array([[ax[0], ax[1]]]) # normalize shape
    for i in range(num_predictions):
        expected = dataframe_original.iloc[i]
        image = io.imread(expected['filename'])
        h, w = image.shape[:2] # actual dims
        # Scale predicted normalized bbox back to original image size
        pb = pred_bboxes[i] * np.array([w, h, w, h])
        pw, ph = pb[2] - pb[0], pb[3] - pb[1]
        # Scale expected normalized bbox back to original image size
        eb = np.array([
            expected['xmin_n'] * w,
            expected['ymin_n'] * h,
            expected['xmax_n'] * w,
            expected['ymax_n'] * h
        ])
        ew, eh = eb[2] - eb[0], eb[3] - eb[1]
        # Left: prediction
        ax[i, 0].imshow(image)

```

```

ax[i, 0].add_patch(patches.Rectangle((pb[0], pb[1]), pw, ph, fill=False,
color=box_color, linewidth=2))
pred_label = int(pred_classes[i])
pred_text = class_names[pred_label] if class_names else str(pred_label)
ax[i, 0].text(pb[0], max(pb[1]-5, 0), pred_text, fontsize=10,
              bbox=dict(facecolor='white', alpha=0.7, edgecolor='none'))
# Right: ground truth
ax[i, 1].imshow(image)
ax[i, 1].add_patch(patches.Rectangle((eb[0], eb[1]), ew, eh, fill=False,
color=box_color, linewidth=2))
true_text = class_names[expected['class_id']] if class_names else
str(expected['class_id'])
ax[i, 1].text(eb[0], max(eb[1]-5, 0), true_text, fontsize=10,
              bbox=dict(facecolor='white', alpha=0.7, edgecolor='none'))
for a in (ax[i,0], ax[i,1]):
    a.set_xticks([]); a.set_yticks([])
plt.tight_layout(); plt.show()
# Directories
DIR_ROOT = '/obj_detect_file/simple_obj_detect'
DIR_IMAGES = f'{DIR_ROOT}/datasets/watergun'
DIR_ANNOTATIONS = f'{DIR_ROOT}/datasets/watergun annotations'
CLASSES = ["Water Gun"]
def parse_annotation(root):
    annotation = { }
    annotation['filename'] = f'{DIR_IMAGES}/{root.find("filename").text}'
    annotation['width'] = int(root.find('size/width').text)
    annotation['height'] = int(root.find('size/height').text)
    # single object per image assumption
    obj = root.find('object')

```

```

class_name = obj.find('name').text.strip()
if class_name not in CLASSES:
    raise ValueError(f'Class '{class_name}' not in CLASSES {CLASSES}. Fix your
list or labels.")
    annotation['class_id'] = CLASSES.index(class_name)
    bb = obj.find('bndbox')
    annotation['xmin'] = int(bb.find('xmin').text)
    annotation['ymin'] = int(bb.find('ymin').text)
    annotation['xmax'] = int(bb.find('xmax').text)
    annotation['ymax'] = int(bb.find('ymax').text)
    return annotation
# Load & build dataframe
annotations = load_xml_annotations(DIR_ANNOTATIONS, parse_annotation)
dataframe_original = pd.DataFrame(annotations)
# Normalize bbox to [0,1] using original image dimensions (robust to any size)
df = dataframe_original.copy()
df['xmin_n'] = df['xmin'] / df['width']
df['ymin_n'] = df['ymin'] / df['height']
df['xmax_n'] = df['xmax'] / df['width']
df['ymax_n'] = df['ymax'] / df['height']
# sanity clip
for c in ['xmin_n', 'ymin_n', 'xmax_n', 'ymax_n']:
    df[c] = df[c].clip(0.0, 1.0)
df.head()
print("TOTAL IMAGES:", len(df))
sample_shape = io.imread(df.iloc[0]['filename']).shape
print('SAMPLE IMAGE SHAPE:', sample_shape)
display_images(dataframe_original, min(2, len(df)))
import matplotlib.pyplot as plt

```

```

import matplotlib.patches as patches
from skimage import io
row = df.sample(1).iloc[0]
image = io.imread(row['filename'])
h, w = image.shape[:2]
x1, y1, x2, y2 = row['xmin_n']*w, row['ymin_n']*h, row['xmax_n']*w,
row['ymax_n']*h
fig, ax = plt.subplots(1)
ax.imshow(image)
ax.add_patch(patches.Rectangle((x1,y1), x2-x1, y2-y1, fill=False, color='red'))
plt.show()

# Parameters
BUFFER_SIZE = 200
BATCH_SIZE = 10
SIZE = 340 # final resize target (square)

# Train/Val split
train_df, val_df = train_test_split(df, test_size=0.2, random_state=SEED,
shuffle=True, stratify=df['class_id'])

# Train Dataset
train_ds = create_dataset_from_dataframe(train_df, use_norm_cols=True)
map_func = lambda path, labels: (load_normalize_image(path, [SIZE, SIZE],
channels=1), labels)
train_ds = (train_ds
            .map(map_func=map_func, num_parallel_calls=tf.data.AUTOTUNE)
            .shuffle(BUFFER_SIZE)
            .cache()
            .repeat()
            .batch(BATCH_SIZE)
            .prefetch(buffer_size=tf.data.AUTOTUNE))

```

```

train_steps = max(1, len(train_df) // BATCH_SIZE)
# Val Dataset
val_ds = create_dataset_from_dataframe(val_df, use_norm_cols=True)
val_ds = (val_ds
          .map(map_func=map_func, num_parallel_calls=tf.data.AUTOTUNE)
          .cache()
          .repeat()
          .batch(BATCH_SIZE)
          .prefetch(buffer_size=tf.data.AUTOTUNE))
val_steps = max(1, len(val_df) // BATCH_SIZE)
print("Elements in train:", len(train_df), "| steps:", train_steps)
print("Elements in val :", len(val_df), "| steps:", val_steps)
train_df.head()
# Model params
INPUT_SHAPE = (SIZE, SIZE, 1)
FIT_EPOCHS = 20
# Callbacks
FIT_CALLBACKS = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=75, min_delta=1e-3,
                                     restore_best_weights=True, verbose=1)
]
# Backbone
inputs = tf.keras.Input(shape=INPUT_SHAPE, name='input_layer')
x = tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu',
name='conv1')(inputs)
x = tf.keras.layers.MaxPool2D(name='pool1')(x)
x = tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu', name='conv2')(x)
x = tf.keras.layers.MaxPool2D(name='pool2')(x)
x = tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu', name='conv3')(x)

```

```

x = tf.keras.layers.MaxPool2D(name='pool3')(x)
x = tf.keras.layers.Flatten(name='flatten')(x)
x = tf.keras.layers.Dense(512, activation='relu', name='dense1')(x)
x = tf.keras.layers.Dense(128, activation='relu', name='dense2')(x)

# Heads
bbox_output = tf.keras.layers.Dense(4, activation='linear', name='bbox')(x)
class_output = tf.keras.layers.Dense(len(CLASSES), activation='softmax',
name='class')(x)
model = tf.keras.Model(inputs=inputs, outputs=[bbox_output, class_output],
name='object_detection_3class')
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss={'bbox': tf.keras.losses.MeanSquaredError(),
        'class': tf.keras.losses.SparseCategoricalCrossentropy()},
    loss_weights={'bbox': 1.0, 'class': 1.0},
    metrics={'bbox': ['mse'], 'class': ['accuracy']})
model.summary()

# Train
history = model.fit(
    train_ds,
    steps_per_epoch=train_steps,
    validation_data=val_ds,
    validation_steps=val_steps,
    epochs=FIT_EPOCHS,
    callbacks=FIT_CALLBACKS,
    verbose=1
)

# Non-repeating eval datasets

```

```

def eval_ds_from_df(dframe):
    ds = create_dataset_from_dataframe(dframe, use_norm_cols=True)
    ds = ds.map(lambda p, y: (load_normalize_image(p, [SIZE, SIZE], channels=1), y),
                  num_parallel_calls=tf.data.AUTOTUNE)
    ds = ds.batch(BATCH_SIZE)
    return ds

train_eval_ds = eval_ds_from_df(train_df)
val_eval_ds = eval_ds_from_df(val_df)
eval_train = model.evaluate(train_eval_ds, verbose=2, return_dict=True)
eval_val = model.evaluate(val_eval_ds, verbose=2, return_dict=True)
model_evaluation_info = f"""
MODEL EVALUATION
Train -> class_acc: {eval_train.get('class_accuracy', float('nan')):.4f} | bbox_mse:
{eval_train.get('bbox_mse', float('nan')):.4f} | total_loss: {eval_train.get('loss',
float('nan')):.4f}
Val -> class_acc: {eval_val.get('class_accuracy', float('nan')):.4f} | bbox_mse:
{eval_val.get('bbox_mse', float('nan')):.4f} | total_loss: {eval_val.get('loss',
float('nan')):.4f}
"""

print(model_evaluation_info)
plot_history(history, model_evaluation_info)
# Predict on first N items from the (normalized) dataframe
N = min(10, len(df))
pred_bboxes = []
pred_classes = []
for i in range(N):
    image_path = df.iloc[i]['filename']
    image = load_normalize_image(image_path, [SIZE, SIZE], channels=1)
    logits_bbox, logits_class = model.predict(tf.expand_dims(image, 0), verbose=0)

```

```

    pred_bboxes.append(np.squeeze(logits_bbox))          # normalized [0,1]
    pred_classes.append(np.argmax(np.squeeze(logits_class)))
pred_bboxes = np.array(pred_bboxes, dtype=np.float32)
pred_classes = np.array(pred_classes, dtype=np.int32)
# Visualize predictions vs ground truth
plot_predictions(
    pred_bboxes=pred_bboxes,
    pred_classes=pred_classes,
    dataframe_original=df,      # uses normalized cols inside
    box_color='lime',
    class_names=["Water Gun"]
)
path = '/obj_detect_file/simple_obj_detect/working/simple_object_detection.h5'
final_model.save(path)

```