

Технически университет - София  
Филиал Пловдив

# Дипломна работа

Тема:

*Система за разпознаване на обект в изображение*

Студент: Любомир Ламбрев  
Фак №: 614973

Специалност: КСТ

Образователна степен: ОКС Магистър

Факултет: ФЕА

ТУ – София, Филиал Пловдив, 2025 г.

# Съдържание

Увод.....	4
Глава 1 – Обзор .....	5
1.1 - Състояние на проблема по литературни данни .....	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 .....	12
2.1 – Компютърно зрение (Computer Vision) .....	12
2.1.1 – Филтри в обработката на изображение .....	14
2.2 –Дълбоко обучение (Deep Learning).....	17
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 – Концептуален модел на архитектурата на софтуерната система .....	29
2.5.2 – Концептуален модел на системата за потребителя.....	31
Глава 3 - Описание на използваната апаратна / схемна / софтуерна част.....	32
3.1 – Система за вход.....	32
3.1.1 – Регистрация на потребител .....	33
3.1.2 – Забравена парола .....	33
3.1.2 – Забравено потребителско име .....	33
3.2 – Работа с изображения .....	34
3.2.1 – Качване на изображение (Upload an image).....	34
3.2.2 – Поставяне чрез clipboard base64/URL .....	35
3.2.3 – Поставяне чрез clipboard изображение.....	35
3.2.4 – Преоразмеряване на изображение.....	36
3.3 –Детекция и сегментация .....	36
3.3.1 – Избор на модел (Detection / Segmentation) .....	36
3.3.2 – Праг на увереност (Confidence Threshold) .....	37
3.3.3 – Визуализиране на резултати.....	37
3.4 – Запазване в база данни .....	38
3.4.1 – Структура на документа за детекция .....	38

3.4.2 – Структура на колекциите .....	39
3.5 – История и статистика .....	40
3.5.1 – Таблица с обекти и броя им .....	41
3.5.2 – Обобщена таблица с обекти и броя им .....	41
3.5.2 – История на детекциите .....	42
3.5.3 – Визуализация на изображения от минали детекции .....	42
3.6 – Софтуерни инструменти и библиотеки .....	43
3.6.1 – Python .....	43
3.6.2 – Streamlit .....	44
3.6.3 – OpenCV .....	44
3.6.4 – MongoDB .....	45
3.6.5 – Mongo Compass .....	45
3.6.6 – PyMongo .....	46
3.6.7 – Bcrypt .....	46
3.6.8 – Pandas .....	46
3.6.9 – PIL (Python Imaging Library) .....	47
3.6.10 – Datetime .....	48
3.6.11 – UUID .....	48
3.6.12 – Streamlit-cookie-manager .....	48
3.6.13 – St_img_pastebutton .....	49
3.6.14 – SMTPlib .....	49
Глава 4 – Изчислителна част/проектиране на блок схема на алгоритми за софтуерната част/функционално тестване .....	50
4.1 – Псевдокод на подсистемата за обучение .....	50
4.2 – Псевдокод на подсистемата за разпознаване на обекти .....	57
Глава 5 – Приложимост на дипломната работа .....	60
Глава 6 – Оценка на икономическата ефективност на разработката (ако е приложимо) .....	60
Глава 7 – Изводи и претенции за самостоятелно получени резултати .....	60

## Увод

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

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

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

# Глава 1 – Обзор

## 1.1 - Състояние на проблема по литературни данни

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

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

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

Съвременните изследвания насочват усилията към подобряване на ефективността, устойчивостта и възможността за работа с ограничени изчислителни ресурси. Голямо внимание се отделя на леките архитектури (MobileNet, EfficientDet), които са оптимизирани за мобилни устройства и вградени системи. Наред с това се разработват и хибридни подходи, съчетаващи класическа обработка на изображения с невронни мрежи за постигане на баланс между точност и изчислителна сложност [11].

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

## **1.2 - Преглед на Уеб базирана система за машинно зрение**

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

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

Все по-често се срещат и системи с отворен код, които позволяват

персонализация и обучение на собствени модели – например базирани на TensorFlow, PyTorch или YOLO архитектури. Тези решения се използват от разработчици и изследователи за изграждане на гъвкави и адаптивни приложения, включително и уеб интерфейси с функционалности за качване, анализ и визуализация на резултати. Стремежа е насочена към комбиниране на локални и облачни изчисления, като се постига баланс между бързодействие, сигурност на данните и удобство за крайния потребител.

### 1.3 - Съществуващи системи за детекция и сегментация на обекти

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

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

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

На пазара се предлагат множество готови платформи за детекция и сегментация.

Сред комерсиалните решения се открояват Google Cloud Vision API, Amazon Rekognition и Microsoft Azure Custom Vision, които предоставят лесна интеграция чрез уеб услуги и API. Тези платформи са оптимизирани за мащабна обработка, но често са ограничени откъм персонализация и изискват заплащане за по-голям обем заявки. Отворените решения като OpenCV, Detectron2 и MMDetection предоставят на разработчиците пълен контрол върху модела, настройките и обучението, като в същото време позволяват интеграция в уеб или локални системи.

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

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

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



### 1.3.1 – Ultralytics YOLO

Ultralytics YOLO (You Only Look Once) е една от най-широко използваните съвременни системи за детекция и сегментация на обекти в изображения и видеа. Основната идея на YOLO моделите е едноетапната обработка на изображенията, при която цялото изображение се анализира за обекти в рамките на една невронна мрежа.

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

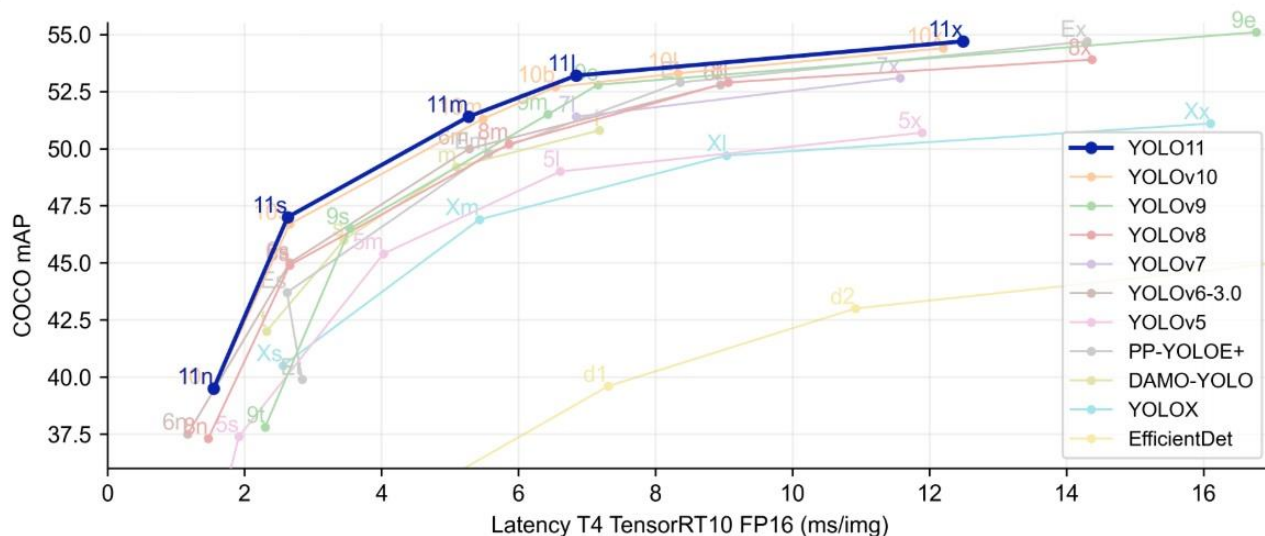
- **Детекция (Detection)**
- **Сегментация (Segmentation)**
- **Класификация (Classification)**
- **Оценка на позата (Pose Estimation)**
- **Ориентирани ограничителни кутии (Oriented Bounding Boxes)**

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

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

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

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



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

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

### 1.3.2 - SSD (Single Shot MultiBox Detector)

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

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

- Детекция (Detection)
- Класификация на обекти в сцени (Object Classification in Scenes)

- **Откриване на обекти в различни мащаби чрез мулти-мащабни характеристики (Multi-Scale Detection)**

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

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

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

### **1.3.3 - Faster R-CNN**

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

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

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

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

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

## **Глава 2 - Теоретично решение на поставената задача**

### **2.1 – 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 [8].

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 [8]. 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, фотоприемна 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 [8].

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 изображение

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.

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 усредняващият 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.

Original Image



filtered image



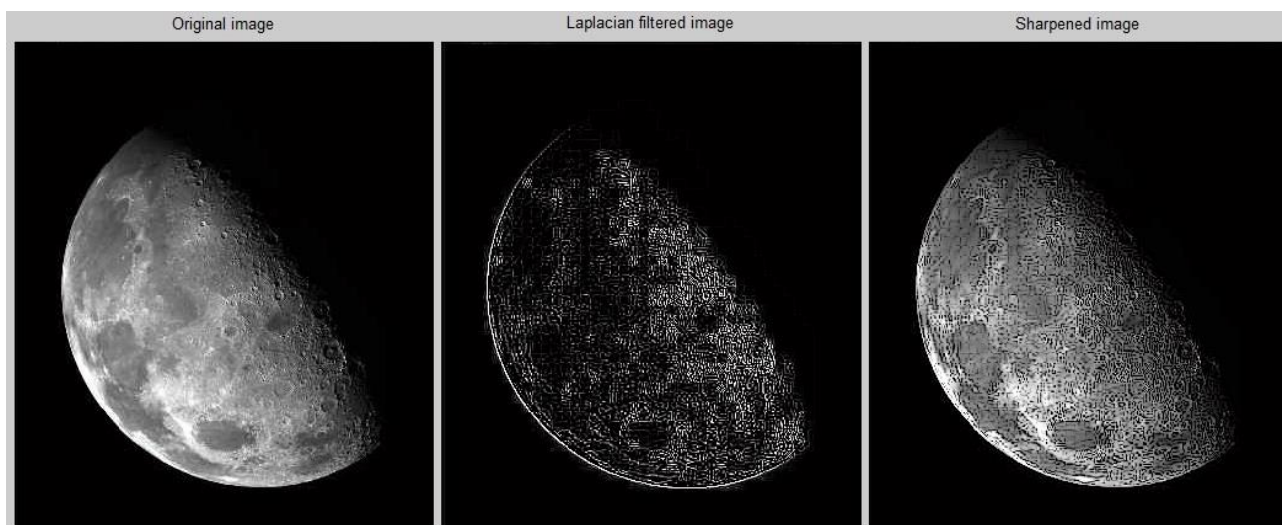
Фиг. 2 Визуално представяне на изображение след усредняващият филтър [18]

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. 2). 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 [18]

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.



Фиг. 4 Визуално представяне след Лапласова филтрация на изображение [18]

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

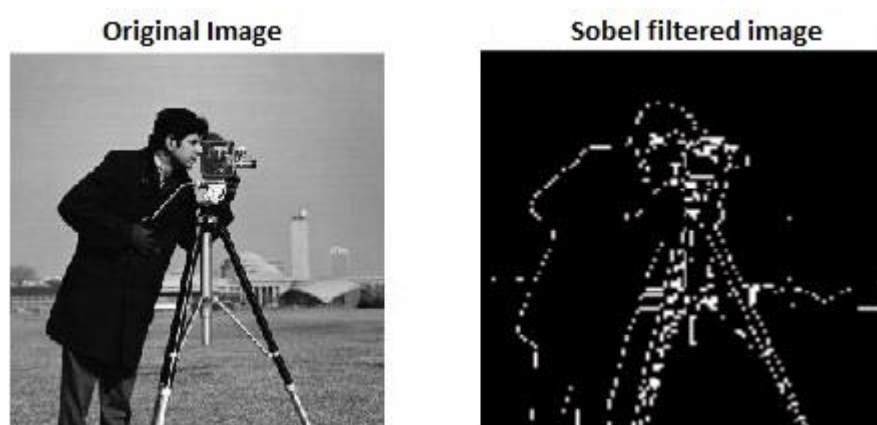


Фиг. 5 Визуално представяне след Медианна филтрация на изображение [18]

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. 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 изображения с оператор на Собел [18]

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 – 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

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.

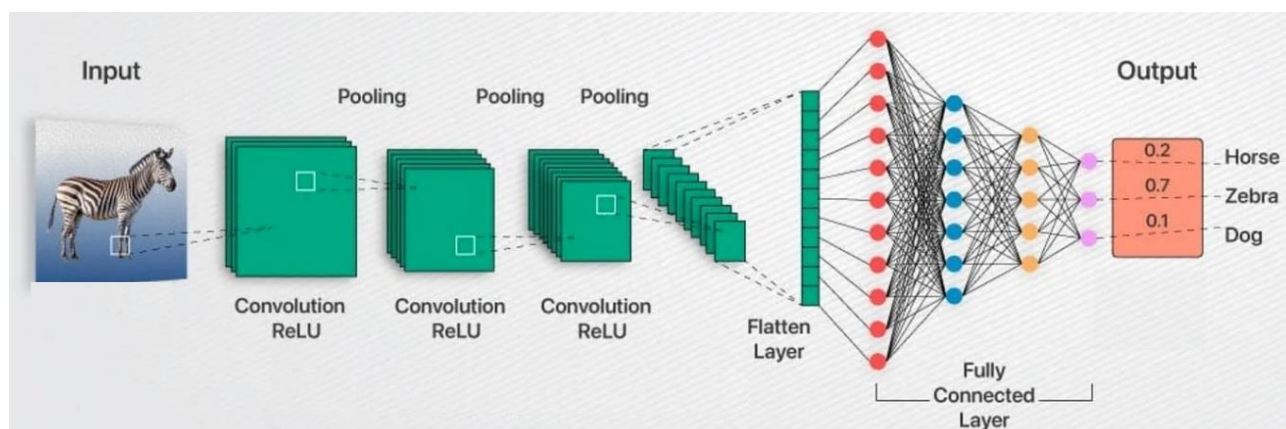
## 2.2 –Дълбоко обучение (Deep Learning)

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 иерархии 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 [9].

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 [9].



Фиг.7 Обща структура на конволюционните невронни мрежи [14]

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.

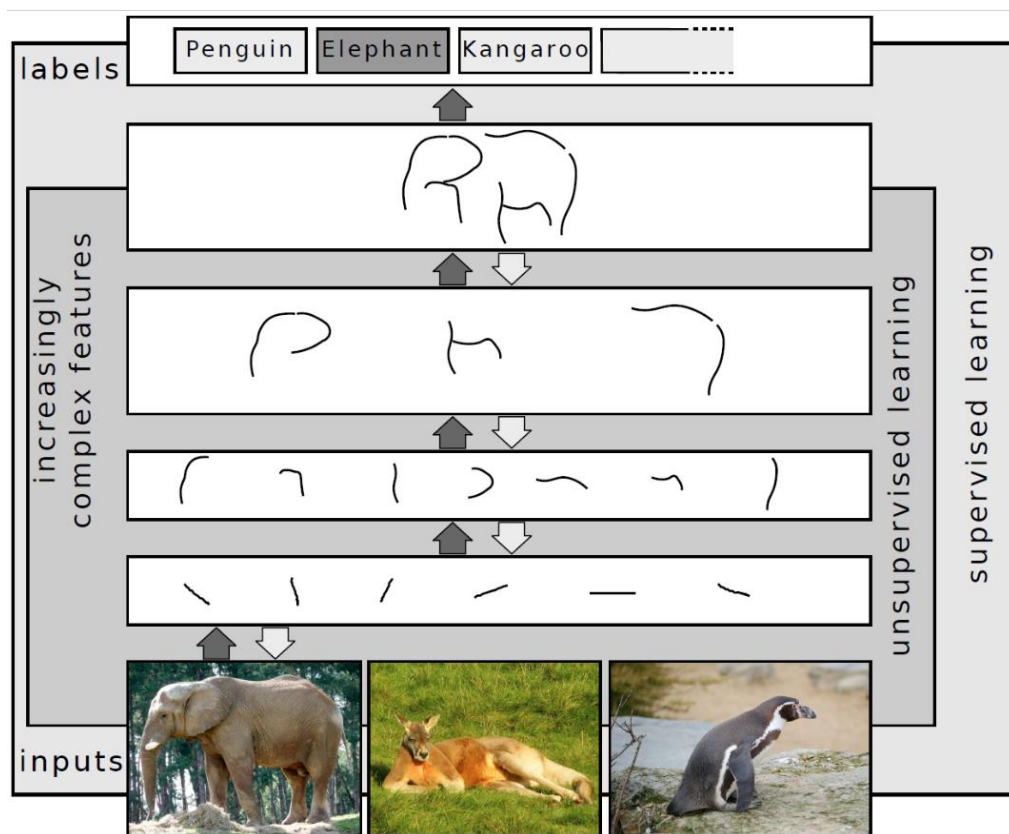
Разликата се използва за корекция на теглата чрез алгоритъма за обратно разпространение на грешката. С всяка итерация точността се повишава, докато моделът не постигне стабилни резултати.

Приложенията на дълбокото обучение в компютърното зрение са изключително разнообразни – от разпознаване на пръстови отпечатыци 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) показано в Фиг. 8



Фиг.8 Представяне на изображения на множество слоеве на абстракция в дълбокото обучение [15]

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.

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



Фиг. 9 Концептуален модел на подсистемата за обучение

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, невиждани 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 по-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. Концептуално това служи като механизъм за обратна връзка, който помага да се идентифицират признаци на пренастройване или недообучаване. Допълнителни техники като ранно спиране (early stopping) подсигурият, 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.

## 2.4 – Концептуален модел на подсистемата за разпознаване на обекти



Фиг. 10 Концептуален модел на системата за разпознаване на обекти

В 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.

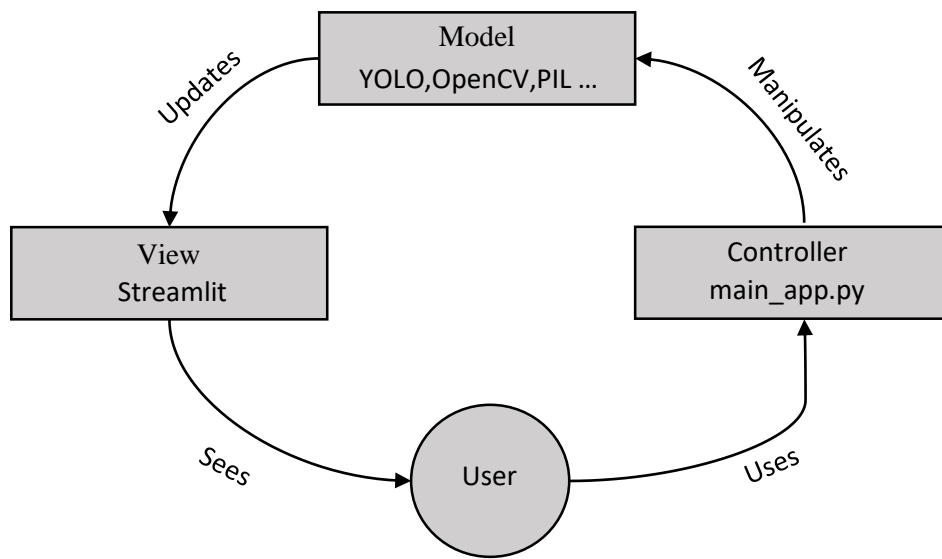
В следващата стъпка – **Визуализация** – Подсистемата превежда числовите резултати в човешки-разбираема форма. Върху изображението се изчертават ограничителни кутии около обектите и се изписват предсказаните класове. Така се получава визуален слой, който позволява бърза оценка на качеството на системата.

## 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 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.

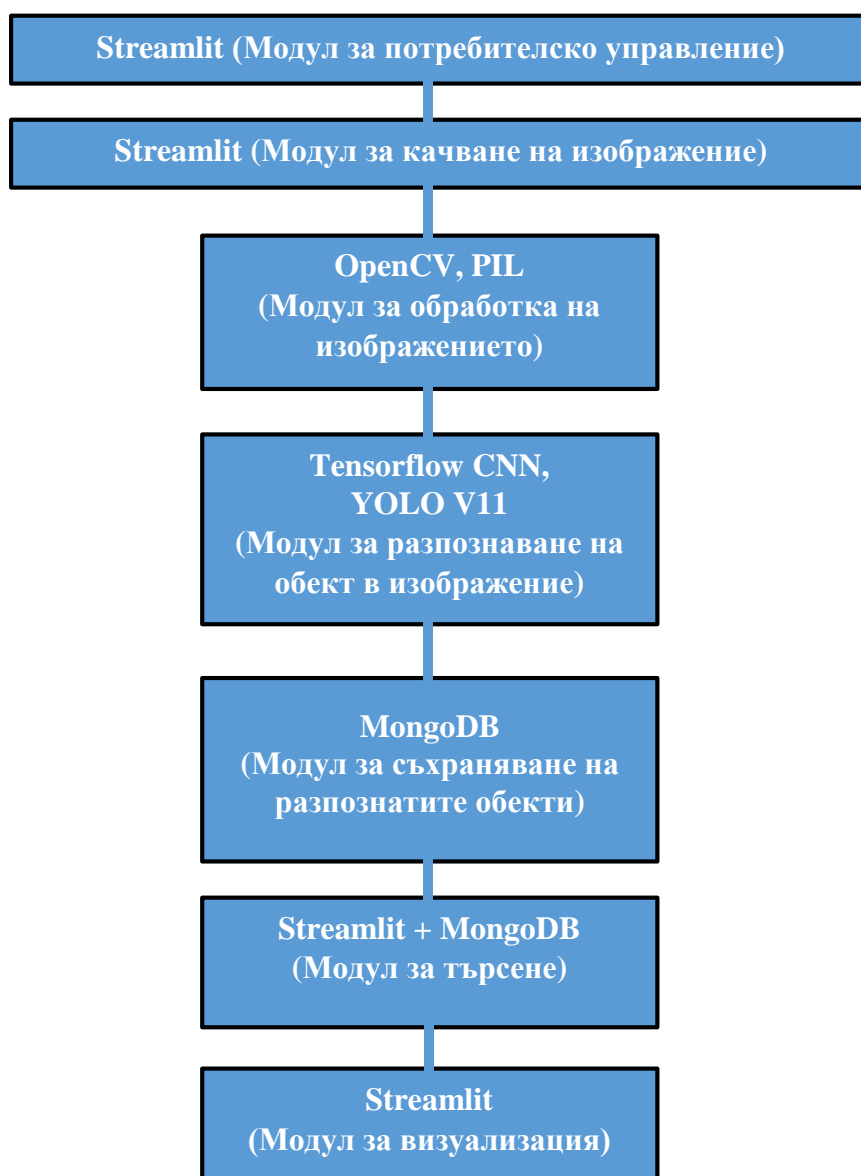
В **изгледа** основната роля е да представя интерфейса и визуализацията на данни за потребителя:

- **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 **контролерът** 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 – Концептуален модел на архитектурата на софтуерната система



Фиг. 12 Концептуален модел на архитектурата на софтуерната система

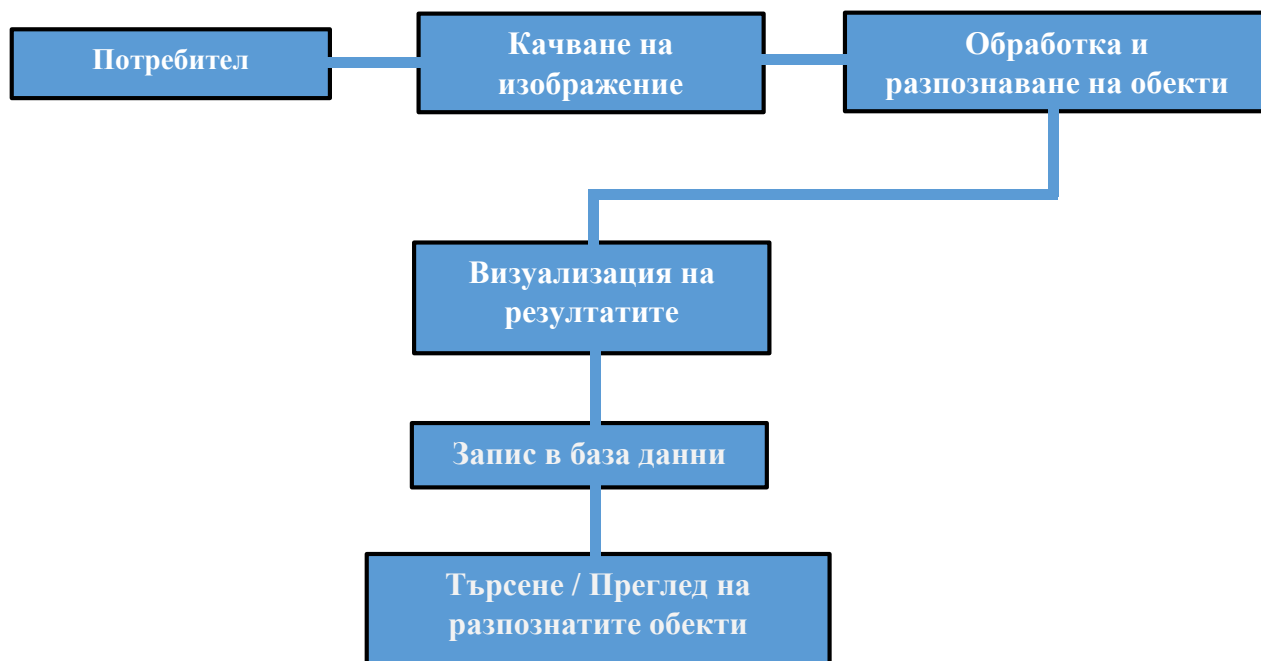
Архитектурата на софтуерната система се разделя на седем основни модула:

- **Модул за потребителско управление** - Отговаря за регистрацията, логина, управлението на сесии, смяната на парола и възстановяването 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 избора 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** - избрания 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



Фиг. 13 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.

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.

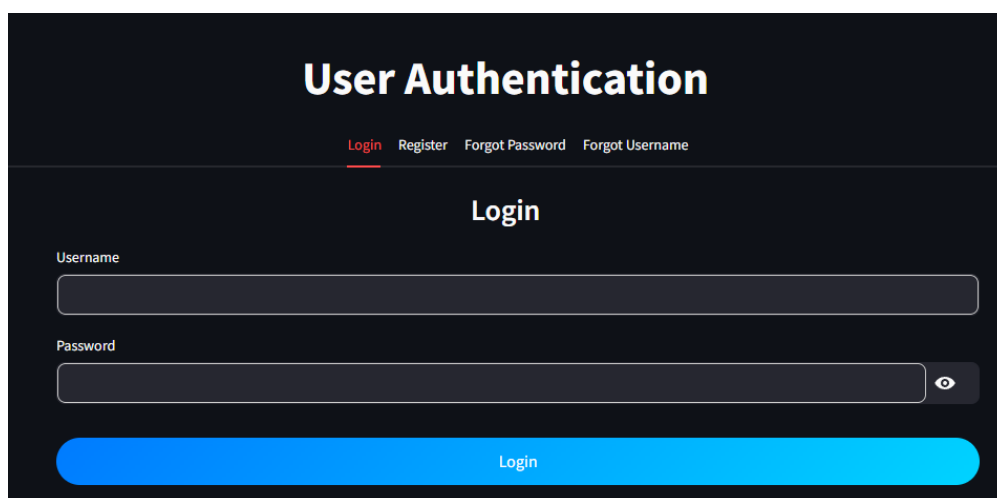
В следващата стъпка - **Запис в база данни** - Всички извършени детекции се съхраняват автоматично в база данни (MongoDB). Записът включва изображението, информация за модела, избраните параметри и разпознатите обекти. Това гарантира, че историята на действията се пази и може да бъде използвана за последващи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

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

### 3.1 – Система за вход

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



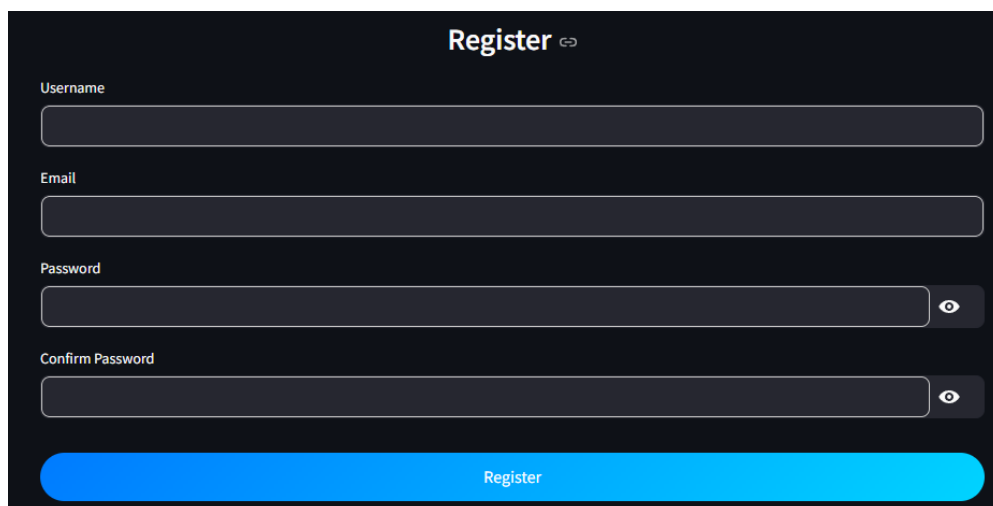
The image shows a dark-themed user authentication interface. At the top, the title "User Authentication" is displayed in white. Below it, there are four links: "Login" (highlighted in red), "Register", "Forgot Password", and "Forgot Username". The main section is titled "Login" and contains two input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

Фиг. 14 Вход за удостоверяване на потребител



### 3.1.1 – Регистрация на потребител

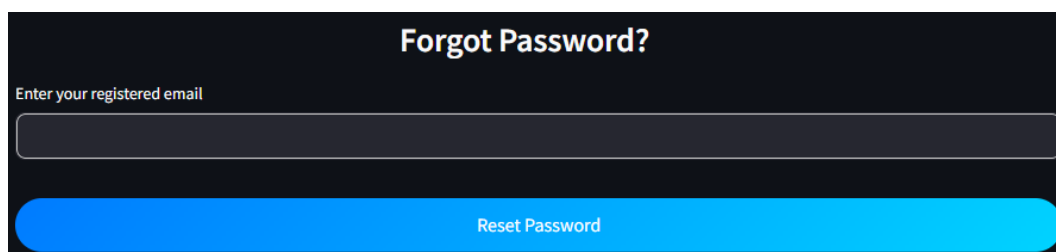
Функционалността започва с процеса по регистрация на нови потребители. При създаване на профил системата извършва серия от проверки: дали паролата съвпада с полето за потвърждение, дали вече съществува потребител със същото име или имейл, както и дали въведените данни отговарят на минимални изисквания.

A screenshot of a web form titled "Register" with a back arrow icon. The form has four input fields: "Username", "Email", "Password", and "Confirm Password". The "Password" and "Confirm Password" fields have eye icons to toggle visibility. A large blue button labeled "Register" is at the bottom.

Фиг. 15 Регистрация на потребител

### 3.1.2 – Забравена парола

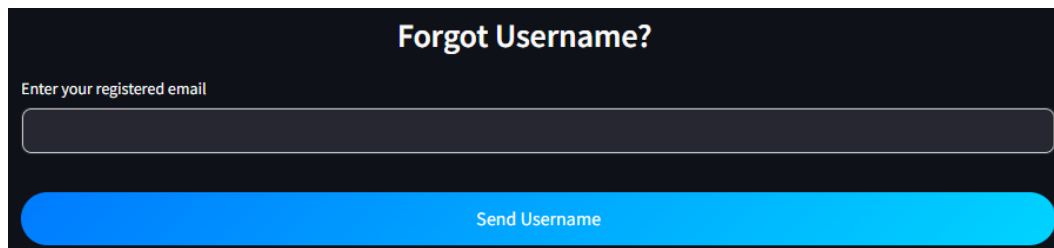
При забравена парола потребителят въвежда регистрирания си имейл и системата генерира временна парола, която автоматично се хешира и записва в базата. Потребителят получава новата парола по имейл чрез интеграция със SMTP сървър на Gmail.

A screenshot of a web form titled "Forgot Password?". It has a single input field labeled "Enter your registered email". A large blue button labeled "Reset Password" is at the bottom.

Фиг. 16 Забравена парола

### 3.1.2 – Забравено потребителско име

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

A dark-themed web form titled "Forgot Username?". It contains a text input field with the placeholder text "Enter your registered email". Below the input field is a large, rounded blue button with the text "Send Username" in white.

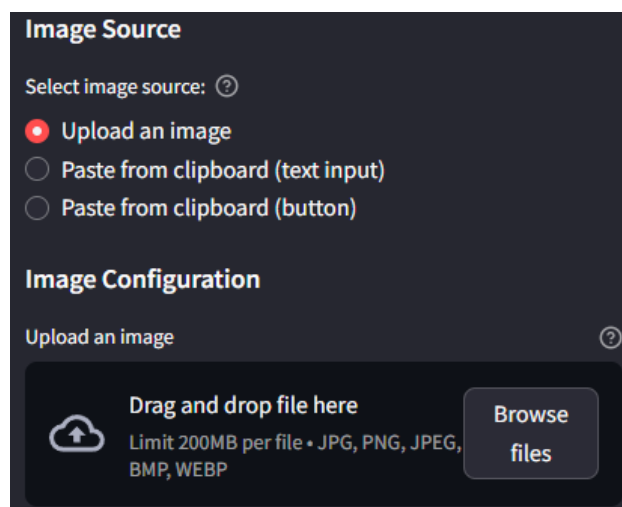
Фиг. 17 Забравено потребителско име

## 3.2 – Работа с изображения

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

### 3.2.1 – Качване на изображение (Upload an image)

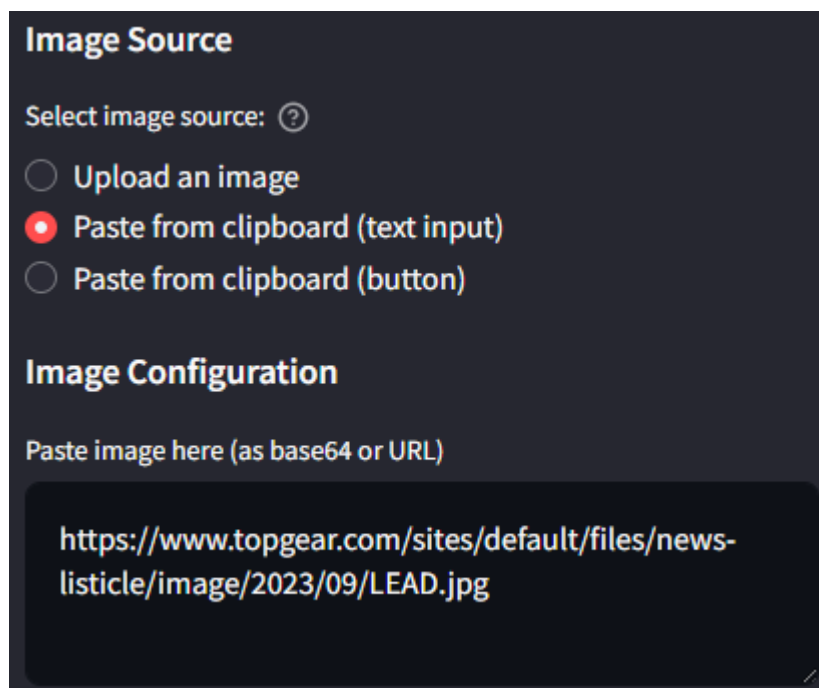
Приложението използва streamlit file\_uploader с ограничение по разширения (jpg, png, jpeg, bmp, webp), осигурявайки валиден формат на входа. При качване файлът се отваря с PIL, а за унифициране на обработката се поддържа BytesIO. Преди всяка последваща операция указателят се връща в началото, за да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

A dark-themed web interface for image upload. It has two main sections: "Image Source" and "Image Configuration". In the "Image Source" section, there is a label "Select image source: ?" followed by three radio button options: "Upload an image" (which is selected), "Paste from clipboard (text input)", and "Paste from clipboard (button)". The "Image Configuration" section has a label "Upload an image ?" and a large area for file upload. This area includes a cloud icon with an upward arrow, the text "Drag and drop file here", a file size limit "Limit 200MB per file • JPG, PNG, JPEG, BMP, WEBP", and a "Browse files" button.

Фиг. 18 Качване на изображение чрез upload

### 3.2.2 – 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 с съобщение за грешка, без да се компрометира стабилността на сесията.



**Image Source**

Select image source: ?

☐ Upload an image

☒ Paste from clipboard (text input)

☐ Paste from clipboard (button)

**Image Configuration**

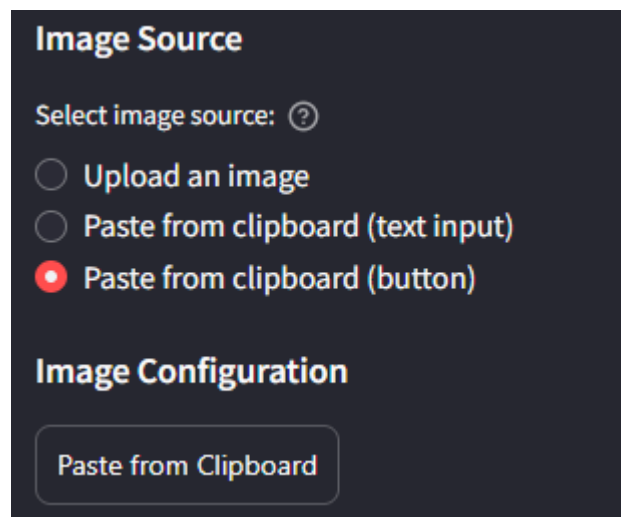
Paste image here (as base64 or URL)

`https://www.topgear.com/sites/default/files/news-listicle/image/2023/09/LEAD.jpg`

Фиг. 19 Поставяне на изображение чрез копираният адрес

### 3.2.3 – Поставяне чрез clipboard изображение

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



Фиг. 20 Поставяне на изображение чрез clipboard бутон

### 3.2.4 – Преоразмеряване на изображение

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

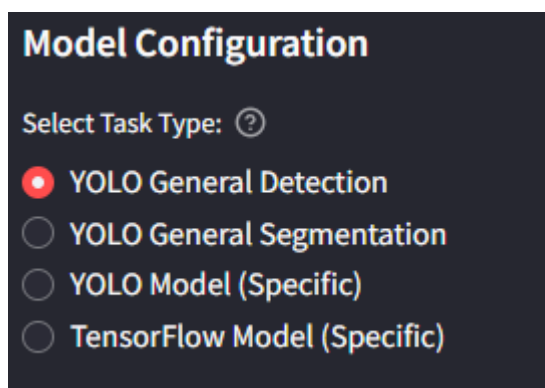
## 3.3 – Детекция и сегментация

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

### 3.3.1 – Избор на модел (Detection / Segmentation)

В приложението е реализиран механизъм за избор на режим на работа, който се осъществява чрез радио бутони в страничното меню. Потребителят може да избере дали да използва стандартен модел за детекция или модел за

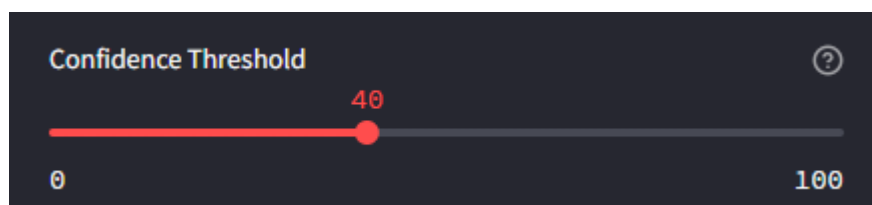
сегментация. При детекция се извършва разпознаване на обекти чрез ограничаващи рамки около тях. При сегментация позволява по-прецизно отделяне на формата на обектите чрез сегментационни маски.



Фиг. 21 Избор на модел за разпознаване на обект

### 3.3.2 – Праг на увереност (Confidence Threshold)

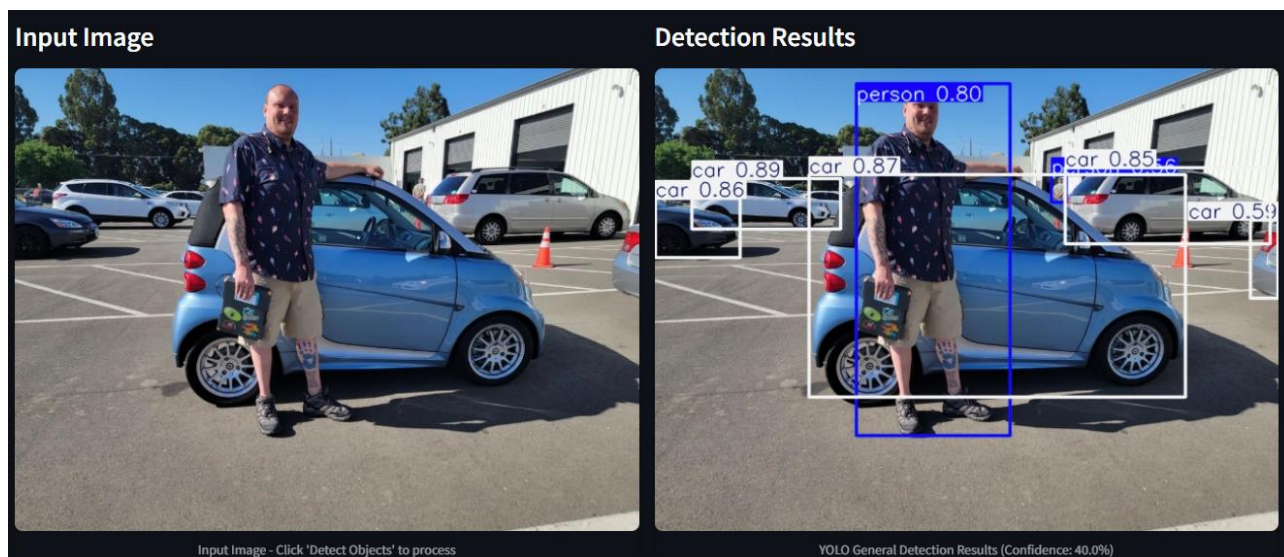
Основен параметър при анализа е прагът на увереност, който контролира чувствителността на модела. В интерфейса този параметър се настройва чрез плъзгач със стойности от 0 до 100 процента, които в кода се преобразуват в десетична стойност и се подават към модела. По този начин потребителят може сам да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



Фиг. 22 Чувствителността на модела за разпознаване на обекти

### 3.3.3 – Визуализиране на резултати

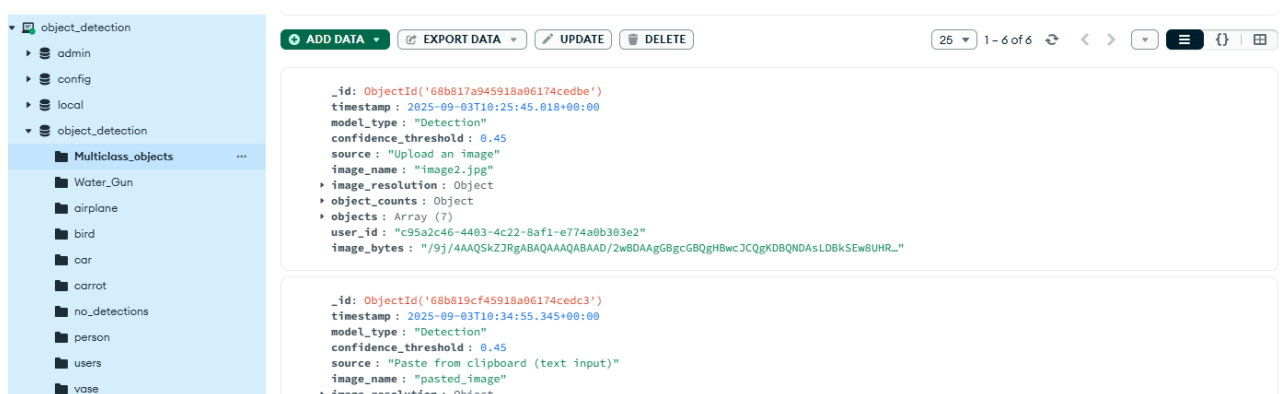
Резултатите от анализа се представят на потребителя чрез визуализация директно в приложението. В системата първо се показва оригиналното изображение, а след това неговата обработена версия с нанесени рамки или маски върху откритите обекти. Освен това към всеки обект се добавя текстова анотация за класа,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



Фиг 22 Визуализация на разпознатите обекти в изображения

### 3.4 – Запазване в база данни

В системата е критично резултатите да бъдат съхранявани надеждно и структурирано в база данни. Това позволява лесно извличане, анализ, визуализация и проследяване на детекциите във времето. В този случай е използвана MongoDB, която е документно-ориентирана база, подходяща за съхранение на разнообразни данни с различна структура.



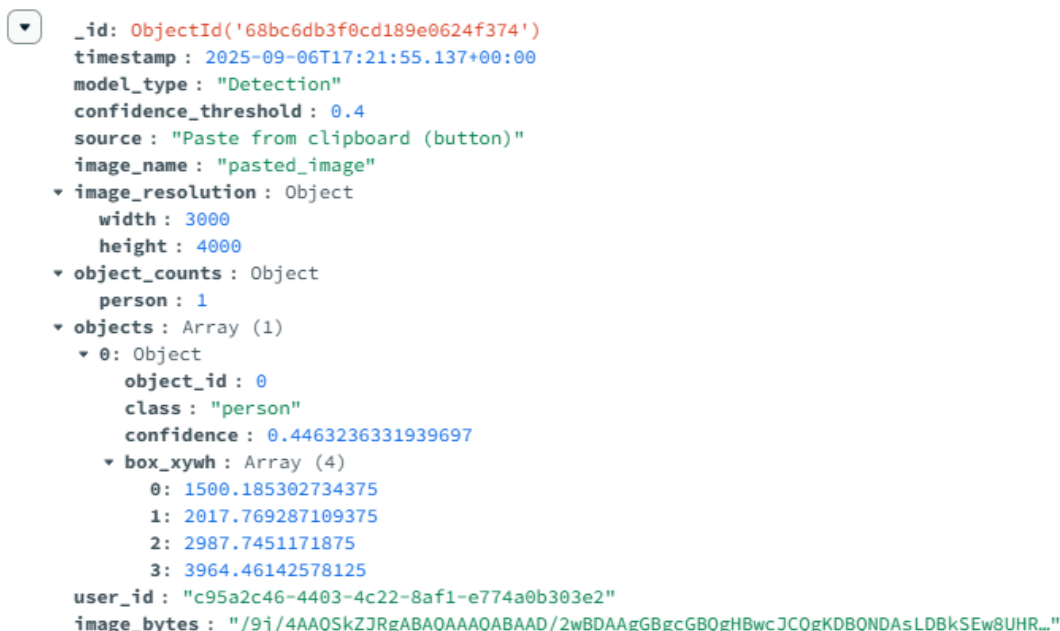
#### 3.4.1 – Структура на документа за детекция

Всеки резултат от детекция се запазва като отделен документ. Документът съдържа метаданни за изображението и конкретните открити обекти. 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** - броя 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.

```
A screenshot of a JSON document structure for a detection. The document is a single object with the following fields: _id (ObjectId), timestamp (ISO 8601 string), model_type (string), confidence_threshold (float), source (string), image_name (string), image_resolution (object with width and height), object_counts (object with person count), objects (array of object objects), user_id (string), and image_bytes (base64 string). The objects array contains one object with fields: object_id, class, confidence, and box_xywh (array of four coordinates).

```
_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/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHR..."
```

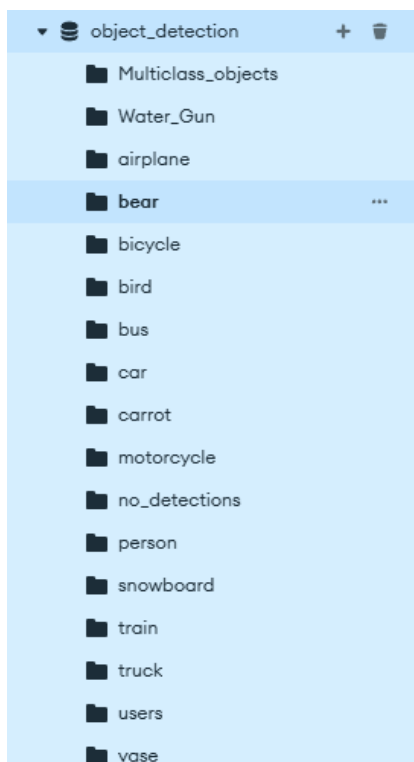

```

Фиг. 24 Структура на документа за детекция

### 3.4.2 – 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**.



Фиг. 25 Структура на колекциите

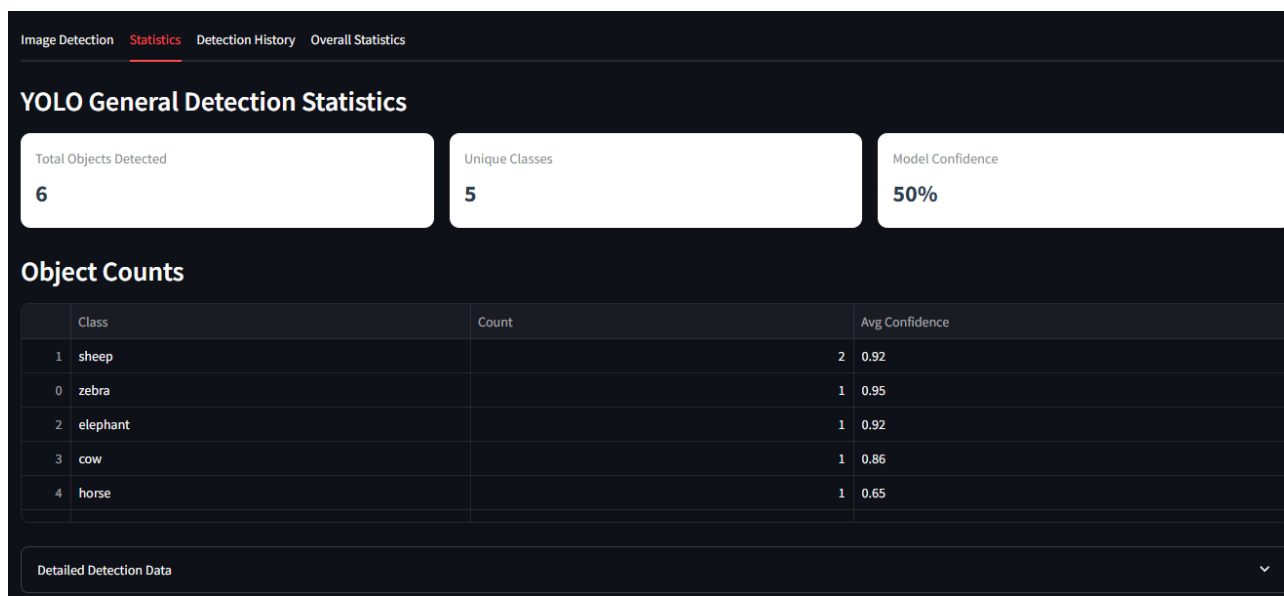
### 3.5 – История и статистика

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



### 3.5.1 –Таблица с обекти и броя им

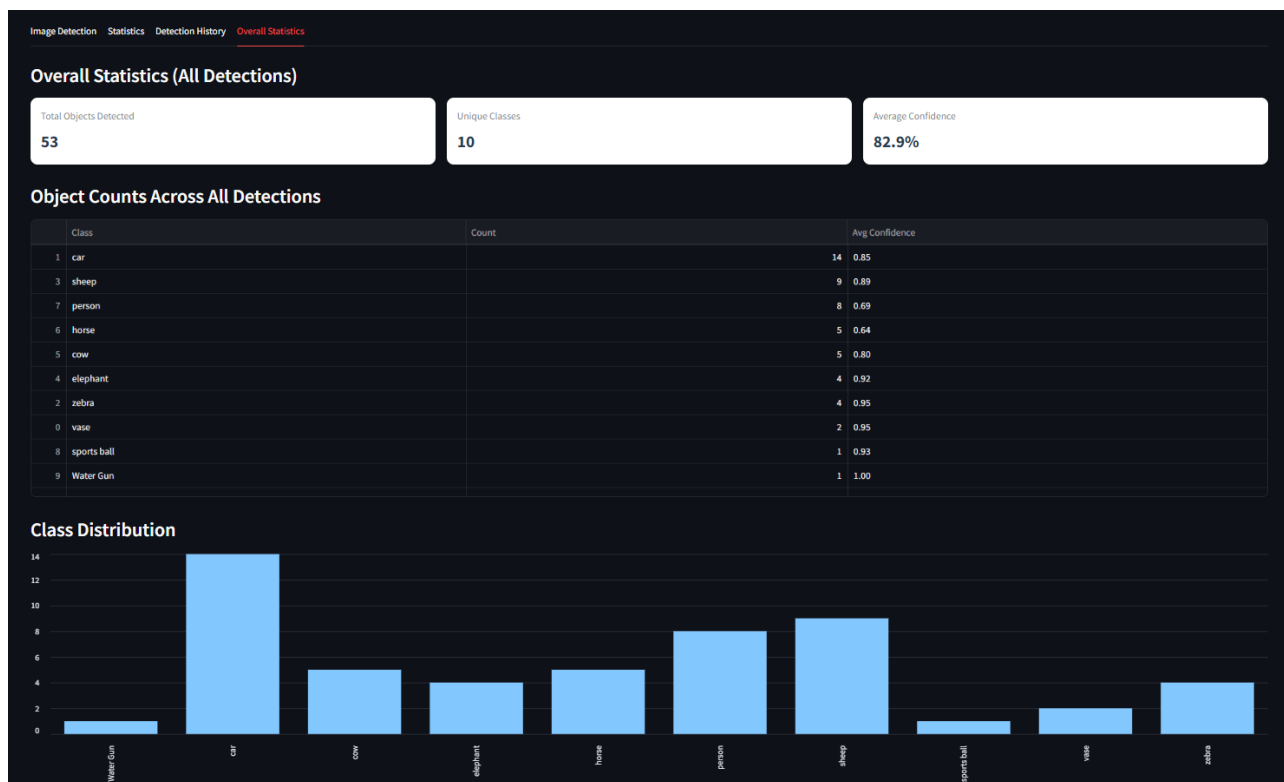
системата генерира таблица, съдържаща информация за класовете обекти, които са били засечени в рамките на една детекция. Таблицата включва колони за името на класа, броя на срещанията му и средната увереност на модела при разпознаването. Чрез сортиране потребителят може лесно да идентифицира кои обекти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



Фиг. 26 Таблица с информация за името на класа, броя на срещанията му и средната увереност на модела

### 3.5.2 –Обобщена таблица с обекти и броя им

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



Фиг. 27 Таблица с информация за всички извършени налични записи

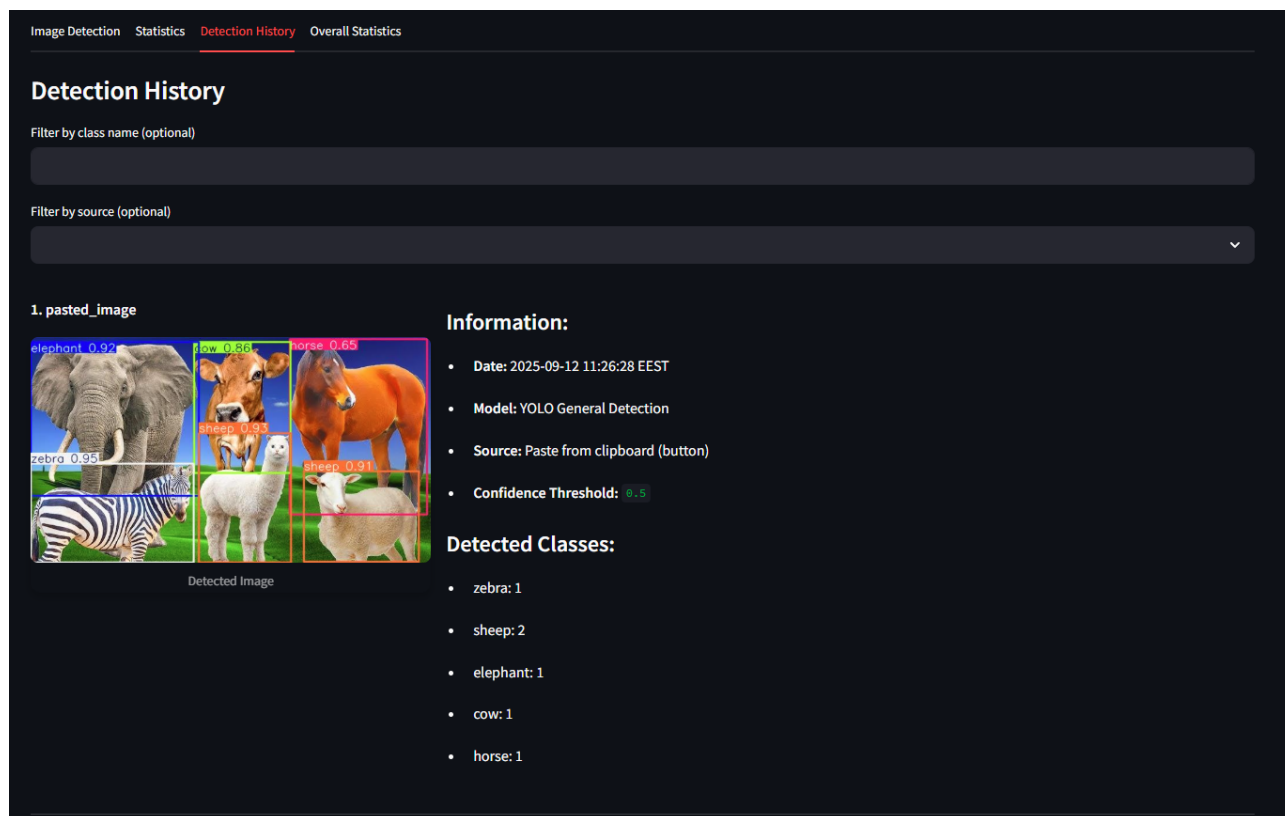
### 3.5.2 – История на детекциите

Достъп до пълната история на извършените детекции за конкретен потребител. Системата съхранява всяка детекция в база данни, като записва дата и час на обработката, използвания модел, избрания източник на изображение и зададения праг на увереност. Освен това, всеки запис съдържа информация за засечените класове и техния брой. Възможни са филтри по класове и източници, което улеснява преглеждането на историята според конкретни критерии.

### 3.5.3 – Визуализация на изображения от минали детекции

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

като доказателствен материал за извършените анализи.



Фиг. 28 Визуализация и информация на предишни детекции

## 3.6 – Софтуерни инструменти и библиотеки

### 3.6.1 – 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.

### 3.6.2 – 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.

### 3.6.3 – 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.

### **3.6.4 – 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.

### **3.6.5 – Mongo Compass**

Освен самата база данни MongoDB, е използван и MongoDB Compass който е официалният графичен интерфейс за управление на базата данни. MongoDB Compass предоставя визуален достъп до документно-ориентираната база, което значително улеснява работата с нея. Вместо да се използват само конзолни команди, потребителят получава удобен графичен изглед за данните, индексирането и структурата на колекциите. Това позволява бързо валидиране на коректността на записите – например дали резултатите детекциите съдържат очакваните класове, координати и праг на увереност. По този начин Compass се използва като инструмент за верификация и отстраняване на грешки при работата с базата.

### **3.6.6 – 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.

### **3.6.7 – 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.

### **3.6.8 – 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 – включващ информация за клас на обект, координати на ограничителни рамки и ниво на увереност. Те се записва първоначално в MongoDB. След това тези данни се извличат и трансформират в Pandas DataFrame, което позволява по-нататъшни операции като статистически анализ, филтрация и агрегиране по класове. В допълнение, Pandas се използва и за експортиране на резултатите в по-удобни формати (например CSV), което улеснява както визуализацията, така и последващата обработка от страна на потребителя.

### **3.6.9 – 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.

### **3.6.10 – 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.

### **3.6.11 – 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.

### **3.6.12 –Streamlit-cookie-manager**

Streamlit-cookies-manager е външна библиотека за Streamlit, която позволява работа с „cookies” директно в приложения. Основната причина за избора на streamlit-cookies-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-cookies-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.

### **3.6.13 – 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

### **3.6.14 – SMTPlib**

smtpplib 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

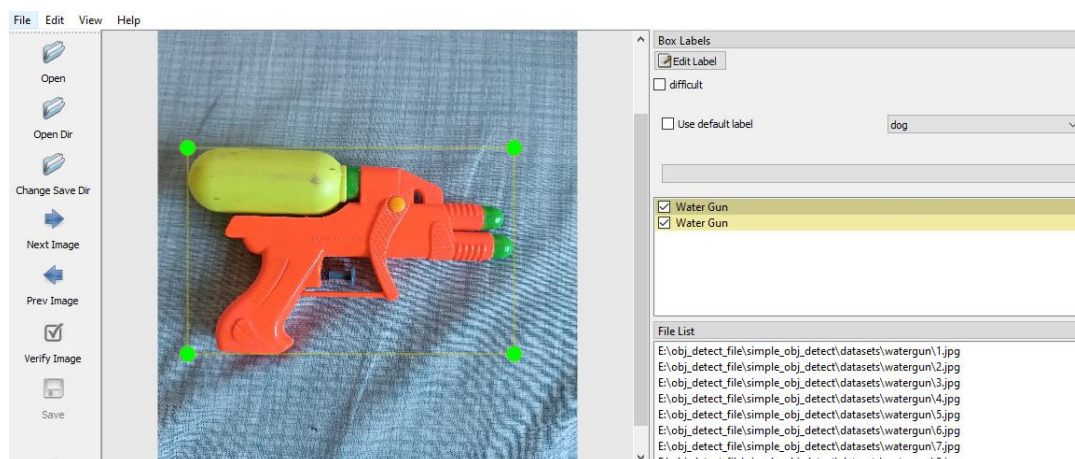
пред други решения се основава на това че е стандартна библиотека в Python, което означава, че е налична по подразбиране и не изисква допълнителна инсталация. Това улеснява приложението, тъй като намалява зависимостите от външни пакети.

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

В тази глава ще се разгледат важни моменти и етапи в кода на подсистемата представен като псевдокод.

### 4.1 – Псевдокод на подсистемата за обучение

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



Следва – **Въвеждане на изображениа** – 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\_IMAGE), 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 съхрани в annotation['filename']
- 2.3. Извличаме ширината и височината на изображението -> annotation['width'], annotation['height']
- 2.4. Намериме 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:** Създаме архитектурата на модела (backbone):

**3.1.** Входен слой: Input(shape = INPUT\_SHAPE)

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

**3.3.** MaxPool2D

**3.4.** Conv2D(32 филтъра, 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

## 4.2 – 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.

## Глава 5 – Приложимост на дипломната работа

## Глава 6 – Оценка на икономическата ефективност на разработката (ако е приложимо)

## Глава 7 – Изводи и претенции за самостоятелно получени резултати

## Литературни източници:

1. Girshick, R. *"Rich feature hierarchies for accurate object detection and semantic segmentation"* (R-CNN). CVPR 2014.
2. Girshick, R. *"Fast R-CNN"*. ICCV 2015.
3. Ren, S., He, K., Girshick, R., Sun, J. *"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"*. NeurIPS 2015.
4. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. *"You Only Look Once: Unified, Real-Time Object Detection"*. CVPR 2016
5. Redmon, J., Farhadi, A. *"YOLO9000: Better, Faster, Stronger"*. CVPR 2017.
6. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.
7. Liu, W., Anguelov, D., Erhan, D. et al. *"SSD: Single Shot MultiBox Detector"*. ECCV 2016.
8. Павлова П., Н. Шакев, Компютърно зрение, ТУ София филиал Пловдив, 2018
9. Goodfellow, I., Bengio, Y., Courville, A. *"Deep Learning."* MIT Press, 2016.
10. Zhao, Z.Q. et al. *"Object Detection with Deep Learning: A Review."* IEEE Transactions on Neural Networks and Learning Systems, 2019.
11. Zou, Z. et al. *"Object Detection in 20 Years: A Survey."* Proceedings of the IEEE, 2019.
12. <https://arxiv.org/html/2412.05252v1#S5> - таблица 1
13. <https://docs.ultralytics.com/models/yolo11>
14. <https://ravjot03.medium.com/decoding-cnns-a-beginners-guide-to-convolutional-neural-networks-and-their-applications-1a8806cbf536>
15. [https://en.wikipedia.org/wiki/Deep\\_learning](https://en.wikipedia.org/wiki/Deep_learning)

16. Canny, J. "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986.

17.

18. Forsyth, D., Ponce, J. "Computer Vision: A Modern Approach", Pearson, 2011.