



SISTEM DE ANALIZĂ NUTRIȚIONALĂ ASISTATĂ DE VIZIUNEА COMPUTERIZATĂ

LUCRARE DE LICENȚĂ

Absolvent: **Ioan-Gabriel NISTOR**

Coordonator **Conf. dr. ing. Tiberiu MARITA**
științific:

2025

Cuprins

Capitolul 1 Introducere	1
Capitolul 2 Obiectivele proiectului	3
2.1 Configurarea mediului de lucru	3
2.2 Definirea arhitecturii aplicației	3
2.3 Crearea și consolidarea seturilor de date	3
2.4 Antrenarea modelelor de detecție și clasificare	3
2.5 Dezvoltarea componentei de analiză vizuală	4
2.6 Crearea bazei de date a claselor	4
2.7 Dezvoltarea componentei de gestionare a utilizatorilor	4
2.8 Dezvoltarea componentei UI (User Interface)	4
2.9 Integrarea componentelor	4
2.10 Testarea și validarea	4
Capitolul 3 Studiu bibliografic	5
3.1 MyFitnessPal – Analiză funcțională și contextuală	5
3.1.1 Context general	5
3.1.2 Funcționalități principale	5
3.2 FoodVisor – Analiză funcțională și contextuală	6
3.2.1 Context general	6
3.2.2 Funcționalități principale	7
3.2.3 Metodologia dezvoltării aplicației FoodVisor	7
3.2.4 Relevanța pentru aplicația propusa	8
3.3 Studiu de caz: Alma – Analiză funcțională și contextuală	8
3.3.1 Context general	8
3.3.2 Funcționalități principale	9
3.3.3 Arhitectura aplicației Alma	9
3.3.4 Relevanța pentru aplicația propusă	9
3.4 Noom – Analiză funcțională și contextuală	9
3.4.1 Context general	9
3.4.2 Funcționalități principale	10
3.4.3 Arhitectura aplicației Noom	10
3.4.4 Relevanța pentru aplicația propusă	10
3.5 Analiza comparativă a aplicațiilor studiate	10
3.5.1 Relevanța pentru aplicația propusă	10
3.5.2 Concluzii	11
Capitolul 4 Analiză și fundamentare teoretică	12
4.1 Analiza obiectivelor proiectului	12
4.1.1 Cerințe funcționale	12
4.1.2 Cerințe non-funcționale	14
4.1.3 Diagrame use-case	15
4.1.4 Scenarii de utilizare	17
4.2 Recunoașterea obiectelor din imagini	19
4.2.1 Învățarea folosind Machine Learning	19

4.2.2	Învățarea folosind Deep Learning	20
4.3	Rețele neuronale convoluționale (CNN)	21
4.3.1	Arhitectura Rețelelor Neuronale Convoluționale	22
4.4	Fundamentarea procesului de recunoaștere vizuală	27
4.4.1	Rolul, structura și preprocesarea seturilor de date	27
4.4.2	Preprocesarea și augmentarea datelor	27
4.4.3	Modelul abstract al detecției de obiecte (YOLO)	28
4.4.4	Arhitectura modelului ResNet (Residual Network)	30
4.5	Aplicațiile Client–Server	31
4.5.1	Interacțiunea prin API-uri (Application Programming Interfaces)	31
4.6	Justificarea soluției propuse	32
4.6.1	Tehnologii folosite	32
Capitolul 5 Proiectare de detaliu și implementare		33
5.1	Arhitectura generală a sistemului propus	33
5.1.1	Componentele aplicației	34
5.1.2	Backend - componenta de analiză vizuală	34
5.1.3	Backend – componenta de gestionare a datelor	35
5.1.4	Baza de date	35
5.1.5	Frontend	36
5.2	Compunerea seturilor de date	37
5.2.1	Pregătirea și preprocesarea setului de date destinat detecției obiectelor de interes.	37
5.2.2	Structura și caracteristicile setului de date	38
5.2.3	Pregătirea și preprocesarea setului de date destinat clasificării reuniunilor de interes.	38
5.2.4	Concluzia compunerii setului de date	39
5.3	Localizarea obiectelor de interes	40
5.3.1	Mediul folosit	40
5.3.2	Configurarea modelului	41
5.3.3	5.4.3. Antrenarea modelului	41
5.4	Clasificarea imaginilor	42
5.4.1	Mediul folosit	42
5.4.2	Configurarea modelului	42
5.4.3	Antrenarea modelului	42
5.4.4	Procesul de fine-tuning	44
5.5	Componența de analiză vizuală	46
5.5.1	Încărcarea modelelor	47
5.5.2	Detectia și segmentarea obiectelor de interes	47
5.5.3	Transformările imaginilor segmentate	47
5.5.4	Clasificarea imaginilor	47
5.6	Componența de gestionare a datelor	48
5.6.1	Baza de date nutrițională	48
5.6.2	Stratul de Repository	49
5.6.3	Stratul de Service	49
5.6.4	Startul de prezentare (Controller)	50

Capitolul 6 Testare și validare	51
6.1 Evaluarea modelului de detecție a obiectelor de interes	51
6.2 Evaluarea modelului de clasificare a obiectelor	53
6.2.1 Analiza matricilor de confuzie	53
6.2.2 Compararea F1-score per clasă	55
6.2.3 Rezumatul performanțelor	56
6.3 Validarea practică a pipeline-ului de analiză vizuală	56
6.4 Testarea componentei de gestionare a datelor	59
Capitolul 7 Manual de instalare și utilizare	60
7.1 Instalare	60
7.1.1 Structura sistemului și containerizare	60
7.1.2 Instalare și rulare aplicație	60
7.2 Ghid de utilizare	61
7.2.1 Autentificare și înregistrare	61
7.2.2 Adăugarea unei mese pe baza unei imagini	61
7.2.3 Vizualizarea datelor legate de alimente	61
7.2.4 Vizualizarea statisticilor nutriționale	61
7.2.5 Setarea obiectivelor personale	61
7.2.6 Administratorul gestioneaza userii	61
Capitolul 8 Concluzii	62
Bibliografie	64
Anexa A Secțiuni relevante din cod	67
Anexa B Alte informații relevante	70

Capitolul 1. Introducere

În ultimele decenii alimentația și stilul de viață sănătos au devenit subiecte de interes în discursul public, lucrările științifice și în politicile de sănătate. Apariția tot mai frecventă a bolilor netransmisibile, precum obezitatea, diabetul de tip 2 sau afecțiunile cardiovasculare sunt strâns legate de alimentația dezechilibrată și de carentele din planul nutrițional. În acest context, devine crucială adoptarea unei diete sănătoase prin urmărirea constantă a obiceiurilor alimentare pentru a reduce riscul acestor probleme de sănătate.

În ultimii ani, dispozitivele mobile inteligente au devenit tot mai accesibile, iar aplicațiile legate de sănătate s-au dezvoltat rapid. Acest lucru a oferit multe oportunități pentru digitalizarea modului în care ne urmărim stilul de viață. Printre cele mai populare soluții se numără aplicațiile mobile de nutriție, care au cunoscut o răspândire tot mai mare. Ele îi ajută pe utilizatori să țină evidența caloriilor consumate și a conținutului nutrițional al meselor, să-și stabilească obiective alimentare și să găsească informații detaliate despre alimente.

Potrivit estimărilor Statista Market Insights (2024) [1], aplicațiile mobile dedicate nutriției cunosc o evoluție semnificativă la nivel global. Pentru anul 2025, valoarea totală a acestei piețe este de aproximativ 6,05 miliarde USD, semnalând un interes în creștere pentru soluțiile digitale care sprijină monitorizarea alimentației și adoptarea unui stil de viață sănătos. Ritmul acestei dezvoltări este susținut și de o rată anuală medie de creștere estimată la 10,89% pentru perioada 2025–2029. În același timp, și utilizarea acestor aplicații se extinde constant: aproximativ 4,57% din populația globală ar putea apela la astfel de instrumente până în 2025, procent care ar putea ajunge la 5,36% spre finalul deceniului. Aceste date reflectă nu doar o expansiune economică, ci și o integrare treptată a aplicațiilor de nutriție în viața cotidiană ca parte din strategiile personale de sănătate.

Cu toate acestea, cercetările recente arată că deși mulți utilizatori pornesc cu un interes ridicat, aceștia renunță adesea destul de repede la aplicațiile de nutriție – uneori chiar înainte să vadă rezultate reale în comportamentul lor alimentar. Un studiu realizat de van der Haar (2023) [2] indică faptul că motivul principal al acestei renunțări rapide este timpul mare necesar pentru utilizarea constantă, mai ales din cauza introducerii manuale a alimentelor consumate. Pe lângă acest aspect au fost identificate și alte limitări precum lipsa personalizării, baze de date incomplete sau neactualizate interfețe greu de folosit și absența unor obiective clare, ușor de adaptat în funcție de utilizator. În același studiu au fost propuse 46 de funcționalități considerate esențiale pentru a influența pozitiv decizia de a începe și a continua utilizarea unei astfel de aplicații, acoperind toate cele trei etape de folosire: începutul, utilizarea zilnică și menținerea pe termen lung. În faza inițială, utilizatorii consideră importantă o introducere clară în aplicație și flexibilitatea în metodele de înregistrare a alimentelor. În etapa de utilizare activă, sunt apreciate navigarea intuitivă, baza de date completă și minimizarea elementelor de publicitate. În faza finală, menținerea interesului este influențată de posibilitatea setării unor obiective realiste, adaptabile și de accesul constant la informații actualizate. Lipsa acestor funcționalități poate duce utilizării aplicației să renunțe la folosirea acesteia, aspect

confirmat atât de analiza calitativă (interviuri și grupuri de discuție), cât și de studiul cantitativ realizat pe un eșantion reprezentativ și prezentat în articolul citat.

Aceste concluzii subliniază faptul că succesul unei aplicații de nutriție nu este determinat exclusiv de prezența unor funcționalități de bază, ci de integrarea intelligentă a unor mecanisme care reduc munca utilizatorului, oferind de asemenea informații relevante, precise și personalizate.

În acest context, tot mai multe lucrări de cercetare se concentrează pe automatizarea procesului de recunoaștere a alimentelor din imagini, folosind tehnici de viziune artificială și învățare automată. Aceste direcții sunt susținute de expansiunea accelerată a pietei globale de e-health și a inteligenței artificiale. Segmentul de aplicații AI în domeniul sănătății este estimat să crească cu peste 37% până în 2030. Această tendință evidențiază nevoia de soluții automatizate care să reducă semnificativ necesitatea introducerii manuale a datelor de către utilizator, fapt care contribuie la o interacțiune mai fluentă și eficientizează interacțiunea utilizator–sistem în domeniul sănătății digitale.

Prezenta lucrare se înscrie în sfera aplicațiilor nutriționale care integrează inteligența artificială pentru a sprijini monitorizarea alimentației și aduce un plus față de aplicațiile existente prin integrarea unui pipeline de analiză vizuală, care permite atât detectarea multiplă a alimentelor dintr-o imagine complexă, cât și clasificarea precisă a fiecărui aliment în parte. Acest mecanism automatizat de analiză vizuală este integrat într-o aplicație web complet funcțională, ce include funcționalități precum: jurnal alimentar, obiective personalizate, statistici zilnice. Tema propusă vizează procesarea automată a imaginilor și dezvoltarea unei aplicații software orientate către domeniul sănătății publice și nutriției.

Relevanța temei este susținută de interesul crescut pentru soluțiile digitale din domeniul nutriției, atât în mediul academic, cât și în cel comercial. Implementarea unor funcționalități care pot identifica automat alimentele și estima valorile nutriționale contribuie la reducerea efortului utilizatorului și la creșterea frecvenței de utilizare. În acest cadru, lucrarea urmărește proiectarea și implementarea unei aplicații care să răspundă acestor cerințe. Astfel, soluția propusă combină performanța modelelor de deep learning cu accesibilitatea unei interfețe intuitive, contribuind la dezvoltarea de instrumente mai eficiente pentru promovarea alimentației echilibrate în mediul digital.

Pentru a înțelege mai bine contextul actual și soluțiile deja existente în domeniul aplicațiilor nutriționale, capitolul trei oferă o analiză detaliată a celor mai relevante platforme disponibile pe piață. Această analiză va evidenția funcționalitățile comune, punctele forte și limitările fiecărei aplicații, punând astfel în evidență necesitatea și originalitatea soluției propuse în cadrul prezentei lucrări.

Capitolul 2. Obiectivele proiectului

Proiectul propune dezvoltarea unei aplicații de tip client-server care integrează un sistem inteligent de analiză vizuală pentru recunoașterea alimentelor. Scopul principal este de a-i ajuta pe utilizatori să își monitorizeze alimentația zilnică într-un mod mai simplu și eficient. Aplicația își propune să automatizeze înregistrarea meselor, reducând necesitatea introducerii manuale a datelor și oferind funcționalități clare pentru urmărirea aportului caloric, a compoziției nutriționale și a istoricului alimentar.

Proiectul se fundamentează pe două direcții complementare: dezvoltarea unui serviciu de analiză vizuală, bazat pe tehnologii din domeniul viziunii artificiale, și integrarea acestuia într-o platformă web modernă, destinată gestionării profilului utilizatorului și a datelor nutriționale.

2.1. Configurarea mediului de lucru

Scopul acestui obiectiv este crearea unui mediu de dezvoltare stabil și eficient, care să permită implementarea, testarea și rularea aplicației în condiții cât mai bune. Etapa include instalarea și configurarea tuturor instrumentelor necesare — biblioteci, dependințe, platforme de lucru — organizarea clară a structurii proiectului, într-un mod modular și ușor de extins, de asemenea, sunt alese și configurate uneltele de dezvoltare, modelele de inteligență artificială cu care se va lucra în procesul de detecție.

2.2. Definirea arhitecturii aplicației

Definirea unei arhitecturi robuste este esențială în buna funcționare a aplicației, de aceea aceasta va fi definită și descrisă utilizând diagrame logice, diagrame use-case și descrieri detaliate ale fiecărei componente pentru a defini într-un mod clar funcționalitățile acestora și modul în care interacționează între ele.

2.3. Crearea și consolidarea seturilor de date

Pentru o recunoaștere corectă și precisă a obiectelor de interes, este esențială construirea de seturi de date potrivite scopului propus. Acestea trebuie să reflecte cât mai bine situațiile reale în care aplicația va fi folosită, inclusiv o varietate suficient de mare de imagini care acoperă diferite tipuri de obiecte și condiții diverse.

Procesul presupune identificare și selectarea surselor publice care conțin imagini relevante pentru domeniul abordat. Ulterior, aceste imagini sunt analizate, filtrate și uniformizate, păstrându-le doar pe cele care respectă criteriile stabilite legate de calitatea, volumul datelor și consistența și corectitudinea etichetelor.

2.4. Antrenarea modelelor de detecție și clasificare

Pentru implementarea componentei vizuale, sunt antrenate două modele bazate pe rețele neuronale convoluționale (CNN): un model pentru detecția zonelor de interes (bounding boxes), responsabil cu identificarea alimentelor din imagine, și un al doilea model pentru clasificarea acestor regiuni în funcție de tipul de aliment. Obiectivul este obținerea

unei recunoașteri precise care să stea la baza estimării automate a valorilor nutriționale și să contribuie la monitorizarea eficientă a obiceiurilor alimentare ale utilizatorului.

2.5. Dezvoltarea componentei de analiză vizuală

Unul dintre obiectivele centrale ale proiectului este dezvoltarea unei componente care să poată identifica automat alimentele din imaginile încărcate de utilizatori. Pentru a realiza acest lucru, este nevoie de integrarea modelelor antrenate să detecteze obiectele din imagine și să le asocieze cu tipul corect de aliment. Scopul este de a obține o recunoaștere precisă, care să fie suficient de fiabilă pentru a permite estimarea valorilor nutriționale și pentru a ajuta utilizatorii să își urmărească mai ușor obiceiurile alimentare.

2.6. Crearea bazei de date a claselor

Pentru a susține procesul de recunoaștere și analiză nutrițională a alimentelor, este necesară definirea și organizarea unei baze de date care să conțină toate clasele de obiecte recunoscute de sistem. Aceste clase corespund diferitelor tipuri de alimente și sunt asociate cu informații detaliate despre compozitia lor nutrițională.

2.7. Dezvoltarea componentei de gestionare a utilizatorilor

Obiectivul va include dezvoltarea părții dedicate gestionării utilizatorilor, responsabilă cu administrarea conturilor și a datelor asociate acestora. Aceasta va permite înregistrarea, actualizarea informațiilor de profil, stergerea profilului și accesul la datele acestuia. Obiectivul va presupune dezvoltarea părții dedicate gestionării informațiilor nutriționale, care va avea ca obiectiv principal calcularea și monitorizarea valorilor energetice și nutriționale. Aceasta va integra baza de date proprie cu valori nutriționale standardizate, asociate fiecărui aliment, și va permite salvarea și gestionarea meselor consumate, setarea și modificarea obiectivelor nutriționale personale, generarea de statistici zilnice, săptămânale privind aportul alimentar;

2.8. Dezvoltarea componentei UI (User Interface)

Pentru a facilita interacțiunea utilizatorilor cu aplicația, proiectul presupune dezvoltarea unei interfețe grafice intuitive. Interfața utilizatorului va fi implementată sub forma unei aplicații web.

2.9. Integrarea componentelor

Integrarea componentelor dezvoltate este o etapă esențială pentru asigurarea funcționării corecte și coerente a întregului sistem. Aplicația este construită pe o arhitectură de tip client-server, în care interfața web (front-end) comunică cu un back-end, care include componentele de detectie și gestionarea utilizatorilor, printr-un set de API-uri (Application Programming Interface).

2.10. Testarea și validarea

Pentru a asigura funcționarea corectă a aplicației și îndeplinirea cerințelor funcționale, proiectul va include o etapă dedicată testării și validării sistemului. Testarea va fi realizată pe mai multe niveluri: unitar, end-to-end. Testarea se va realiza pe parcursul dezvoltării aplicației, fiecare componentă va fi testată individual după implementare.

Capitolul 3. Studiu bibliografic

În contextul dezvoltării unei aplicații inteligente pentru recunoașterea alimentelor și estimarea valorilor nutriționale, este esențială analiza soluțiilor existente care au abordat problematica nutriției digitale din diverse perspective tehnologice și funcționale. Studiul bibliografic realizat în această secțiune are ca scop identificarea caracteristicilor comune, a limitărilor și a direcțiilor de inovare prezente în aplicațiile nutriționale moderne.

Analiza funcționalităților, a tehnologiilor folosite și a modului în care au fost construite aceste aplicații are rolul de a susține direcțiile tehnologice alese în cadrul proiectului. De asemenea, această abordare ajută la evidențierea aspectelor care diferențiază soluția propusă și care pot aduce un plus de valoare utilizatorilor, comparativ cu alte produse existente.

3.1. MyFitnessPal – Analiză funcțională și contextuală

Informațiile prezentate în această secțiune sunt sintetizate pe baza articolului “The Magic of MyFitnessPal: How One App Can Help You Achieve Your Fitness Dreams” (TechAhead, 2024) [3], care oferă o perspectivă detaliată asupra funcționalităților, arhitecturii și impactului aplicației MyFitnessPal în domeniul managementului nutrițional și al activităților de fitness.

3.1.1. Context general

MyFitnessPal este o aplicație mobilă lansată în anul 2005, proiectată pentru platformele Android și iOS. Scopul principal al aplicației este sprijinirea utilizatorilor în monitorizarea aportului caloric și a activității fizice, în vederea atingerii unor obiective personalizate de sănătate.

3.1.2. Funcționalități principale

Aplicația oferă un set de funcționalități integrate care sunt clasificate în trei categorii principale:

Monitorizarea nutrițională

- Bază de date cu peste 18 milioane de produse alimentare;
- Scanare coduri de bare pentru introducerea rapidă a produselor ambalate;
- Calcul automat al macronutrienților (proteine, carbohidrați, grăsimi) și micronutrienților (zahăr, colesterol etc.);
- Posibilitatea personalizării ţintelor nutriționale zilnice.

Monitorizarea activității fizice

- Înregistrarea exercițiilor fizice și a caloriilor arse;
- Integrare cu dispozitive externe de tip fitness tracker;
- Planuri predefinite de antrenament.

Suport și obiective

- Vizualizare grafică a progresului (calorii, greutate, nutrienți);
- Posibilitatea exportării datelor pentru uz medical;
- Interacțiune socială prin comunități.

Arhitectura aplicației MyFitnessPal

Conform informațiilor publice disponibile despre arhitectura MyFitnessPal (TechAhead, 2024) [4], aplicația adoptă un model modular distribuit, bazat pe microservicii, care asigură scalabilitate, disponibilitate ridicată și integrare facilă cu servicii terțe.

Structura generală a sistemului este alcătuită din următoarele componente principale:

- **Aplicația client (mobilă și web)** – construită folosind framework-uri moderne, precum *React* și *React Native*;
- **Gateway API** – componentă intermediară responsabilă de rutarea cererilor, autentificare și aplicarea limitărilor de acces (*rate limiting*);
- **Microservicii backend** – fiecare microserviciu gestionează o funcționalitate specifică (de exemplu: autentificare, monitorizarea nutriției, analiza activităților fizice);
- **Baze de date** – sunt utilizate atât baze de date relationale (ex. *SQL Server*, *MySQL*) pentru date structurate (utilizatori, setări), cât și baze de date NoSQL (ex. *MongoDB*, *Redis*) pentru date nestructurate (jurnale alimentare, fluxuri de activitate);
- **Module de Machine Learning** – oferă recomandări personalizate de mese și analize predictive privind aportul caloric și progresul în greutate. Modelele de ML sunt construite cu ajutorul *TensorFlow*;
- **Servicii de integrare terță** – permit comunicarea cu API-uri externe (de exemplu, aplicații de monitorizare a activităților fizice sau baze de date nutriționale comerciale).

Relevanța pentru aplicația propusă

Aplicația MyFitnessPal oferă un punct de referință solid în ceea ce privește structura și funcționalitățile necesare unei aplicații de nutriție moderne. Cu toate acestea, soluția propusă în cadrul acestei lucrări urmărește să aducă un plus de valoare prin automatizarea procesului de colectare a datelor nutriționale, folosind un modul intelligent de recunoaștere vizuală a alimentelor. Această abordare are potențialul de a reduce considerabil efortul utilizatorului și de a crește gradul de utilizare constantă a aplicației.

3.2. FoodVisor – Analiză funcțională și contextuală

Informațiile prezentate în această secțiune sunt sintetizate pe baza lucrării științifice “FoodVisor: A Food Calorie Estimation System”, publicată în cadrul conferinței IEEE ICAIT 2024 [5]. Această lucrare propune un sistem de estimare a valorii calorice a alimentelor pe baza imaginilor, utilizând modele de deep learning, în special Convolutional Neural Networks (CNN) și ResNet50 pentru extragerea caracteristicilor vizuale.

3.2.1. Context general

FoodVisor este o aplicație web dezvoltată cu scopul de a sprijini utilizatorii în monitorizarea aportului caloric și a conținutului nutrițional al alimentelor, pe baza ima-

ginilor încărcate. Abordarea răspunde nevoii de automatizare a procesului de identificare a alimentelor, eliminând necesitatea introducerii manuale a acestora.

3.2.2. Funcționalități principale

Aplicația studiată oferă un set de funcționalități concepute pentru automatizarea procesului de recunoaștere a alimentelor, de estimare a valorilor nutriționale, iar interacțiunea cu sistemul fiind posibilă prin intermediul unei interfețe web oferite de platformă. Aceste funcționalități sunt grupate în trei categorii principale:

Recunoaștere vizuală a alimentelor

- Aplicația identifică automat tipul de aliment prezent în imagine, utilizând un model CNN antrenat pe arhitectura ResNet50;
- După identificare, sistemul estimează valoarea calorică a alimentului, exprimată în kcal/100g;
- Modelul utilizat a demonstrat o acuratețe de aproximativ 77,97% și un scor F1 de 0,776, ceea ce indică o performanță bună în condiții reale de utilizare.

Procesare automată a imaginilor

- Imaginile încărcate sunt prelucrate automat prin pași standard: redimensionare, normalizare și aplicarea de augmentări;
- Aceste etape contribuie la îmbunătățirea robustei modelului și la obținerea unor rezultate mai fiabile;
- Încărcarea imaginilor se face direct din interfața aplicației, iar analiza este declanșată automat, fără intervenții suplimentare.

Interfață de utilizator

- Platforma este disponibilă sub formă de aplicație web, compatibilă cu principalele browsere și ușor de accesat;
- După încărcarea unei imagini, utilizatorul primește imediat rezultatul: denumirea alimentului și valoarea calorică estimată;
- Interfața este intuitivă și oferă feedback rapid, fiind concepută pentru a fi utilizabilă chiar și de persoane fără pregătire tehnică.

3.2.3. Metodologia dezvoltării aplicației FoodVisor

Metodologia adoptată urmează un lanț de procesare logic, organizat pe mai multe etape funcționale, ilustrate și în Figura 3.1.

1. Construirea și procesarea setului de date

- Utilizarea datasetului Food-101, care conține 101.000 imagini distribuite în 101 clase alimentare;
- Separarea în seturi de antrenare și validare, urmată de preprocesări (redimensionare, flip, normalizare);
- Standardizarea imaginilor pentru compatibilitatea cu arhitectura modelului.

2. Modelele de Deep Learning

- Extragerea caracteristicilor vizuale utilizând modelul ResNet50 prin tehnica transfer learning;

- Antrenarea unei rețele CNN pentru clasificarea alimentelor și estimarea valorii calorice.

3. Performanță

- Comparativ cu alte modele (Inception V4 și VGG16), ResNet50 a obținut cele mai bune rezultate;
- Acuratețe în antrenare: 77,97%, acuratețe pe setul de validare: 75,77%;
- Evaluare suplimentară prin metrii precum precizie, scor F1 și recall — toate cu valori medii în jurul pragului de 0,77.

4. Dezvoltarea aplicației web

- Integrarea modelului de inferență într-o aplicație web interactivă;
- Utilizatorii încarcă imagini cu alimente, iar sistemul identifică tipul de aliment și oferă o estimare a valorii calorice (în kcal/100g).

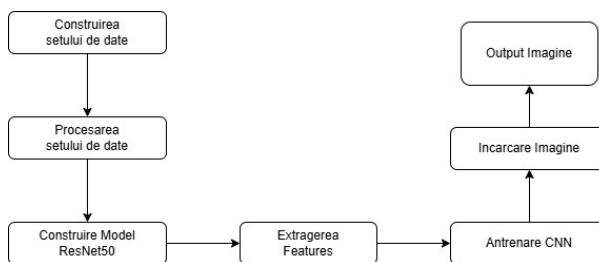


Figura 3.1: Metodologia dezvoltării aplicației FoodVisor

3.2.4. Relevanța pentru aplicația propusa

FoodVisor demonstrează fezabilitatea utilizării rețelelor neuronale conveționale pentru estimarea valorii calorice a alimentelor pe baza imaginilor, însă este limitată la clasificarea unui singur aliment per imagine și nu oferă suport pentru gestionarea nutriției în ansamblu. Prin urmare, aplicația propusă extinde acest concept prin integrarea unei componente de detecție a mai multor obiecte dintr-o singură imagine și combinată cu clasificarea eficientă oferită de ResNet50. De asemenea conceptul este extins prin integrarea componente de gestionare care permite jurnalizarea automată a meselor și personalizarea interacțiunii în funcție de utilizator.

3.3. Studiu de caz: Alma – Analiză funcțională și contextuală

Informațiile prezentate în această secțiune sunt sintetizate pe baza comunicatului oficial de lansare al aplicației Alma (Alma Nutrition, 2025) și a articolului publicat de TechCrunch [6, 7]. Alma reprezintă o nouă generație de asistenți digitali în nutriție, construită în jurul tehnologiilor de inteligență artificială conversațională și vizuie computațională, având ca scop simplificarea procesului de urmărire alimentară și oferirea de recomandări nutriționale personalizate.

3.3.1. Context general

Aplicația Alma a fost fondată în 2024 de Rami Alhamad, fost vicepreședinte de produs la compania Whoop, și a fost lansată public în februarie 2025. Se adresează

utilizatorilor care doresc să își îmbunătățească obiceiurile alimentare fără a fi nevoiți să interacționeze cu interfețe complicate sau să introducă manual fiecare aliment. Alma funcționează ca un asistent nutrițional intelligent, capabil să înțeleagă limbajul natural și să interacționeze prin voce, text sau imagine.

3.3.2. Funcționalități principale

Aplicația oferă un set extins de funcționalități bazate pe inteligență artificială, printre care:

- Logarea meselor prin mai multe metode: voce, text, imagini sau scanare de coduri de bare;
- Recunoașterea vizuală a alimentelor din fotografii;
- Estimarea automată a porțiilor și a valorii calorice;
- Calcularea unui scor nutrițional zilnic, bazat pe aportul de calorii și macronutrienți;
- Adaptarea progresivă la preferințele utilizatorului, pe baza interacțiunilor repetate cu asistentul AI.

3.3.3. Arhitectura aplicației Alma

Aplicația este disponibilă, momentan, exclusiv pe platforma iOS și utilizează un sistem hibrid de modele AI, care îmbină tehnici de procesare a limbajului natural (NLP), învățare automată (machine learning) și viziune computațională. Arhitectura se bazează pe date validate științific, combinate cu interacțiuni multimodale (voce, imagine, text), într-un flux conversațional fluid și natural.

3.3.4. Relevanța pentru aplicația propusă

Alma reprezintă un exemplu actual și relevant de aplicație de nutriție centrată pe utilizator și susținută de AI generativ. Aceasta demonstrează potențialul real al noilor tehnologii în simplificarea și personalizarea procesului de monitorizare alimentară. Integrarea unor componente similare, precum recunoașterea vizuală sau interacțiunea naturală, poate aduce beneficii semnificative aplicației propuse în cadrul acestei lucrări.

3.4. Noom – Analiză funcțională și contextuală

Informațiile prezentate în această secțiune sunt sintetizate pe baza articolului “The Growth Machine: How Noom Runs 365 Landing Page Experiments Per Year” [8], dar și a articolului “How to develop a fitness app like Noom” [9], care oferă o perspectivă detaliată asupra funcționalităților, arhitecturii și impactului aplicației Noom în domeniul nutriției și wellness-ului digital.

3.4.1. Context general

Noom este o aplicație mobilă concepută pentru a sprijini utilizatorii să adopte un stil de viață sănătos, în vederea schimbării comportamentelelor alimentare într-un mod sustenabil. Aplicația combină planuri nutriționale personalizate cu elemente de coaching psihologic. Lansată în 2008, Noom a cunoscut o creștere accelerată în ultimii ani datorită adoptării unei strategii bazate pe date și utilizării tehnologiilor de învățare automată.

3.4.2. Funcționalități principale

Noom oferă o suită de funcționalități centrate pe personalizare, monitorizare și sprijin comportamental:

- Planuri alimentare personalizate, generate în funcție de profilul utilizatorului;
- Monitorizarea progresului privind greutatea și obiceiurile alimentare;
- Jurnal alimentar și integrare cu dispozitive de fitness;
- Coaching digital asistat de AI, dar și suport uman prin antrenori certificați;
- Utilizarea tehniciilor de terapie cognitiv-comportamentală (CBT) pentru sprijinirea schimbărilor de comportament.

3.4.3. Arhitectura aplicației Noom

Conform surselor publice, Noom implementează o arhitectură modulară, distribuită, bazată pe microservicii, ce asigură scalabilitate, disponibilitate ridicată și integrare ușoară cu servicii externe.

Structura generală a sistemului este susținută de un ecosistem de experimentare continuă:

- Utilizarea *React.lazy* și a importurilor dinamice pentru încărcarea modulară și condiționată a componentelor UI;
- Backend construit pe un framework precum *Django*, cu suport din partea unei infrastructuri cloud scalabile.

3.4.4. Relevanța pentru aplicația propusă

Noom constituie un exemplu relevant de integrare a inteligenței artificiale în aplicații mobile de nutriție. Totuși, accentul pus de Noom pe personalizarea prin NLP și coaching conversațional îl diferențiază de abordarea vizuală propusă în cadrul acestei lucrări. Cu toate acestea, ambele soluții urmăresc același obiectiv general: sprijinirea utilizatorilor în îmbunătățirea obiceiurilor alimentare prin intermediul tehnologiei într-un mod automatizat.

3.5. Analiza comparativă a aplicațiilor studiate

În această secțiune este realizată o analiză comparativă a celor patru aplicații descrise anterior: **MyFitnessPal**, **FoodVisor**, **Noom** și **Alma** în comparație cu aplicația propusă în această lucrare. Aceste aplicații propun abordări diferite pentru monitorizarea nutriției, integrând tehnologii variate precum inteligența artificială conversațională, viziunea computațională, psihologia comportamentală sau baze de date nutriționale extinse.

3.5.1. Relevanța pentru aplicația propusă

Pentru a evidenția poziționarea aplicației propuse în contextul soluțiilor existente, a fost realizată o analiză comparativă pe baza a cinci criterii esențiale, prezentate în Tabelul 3.1:

- Modalitatea de introducere a datelor alimentare: acest criteriu reflectă nivelul de efort necesar din partea utilizatorului pentru înregistrarea meselor.
- Gradul de automatizare a procesului de recunoaștere și estimare;
- Nivelul de personalizare a recomandărilor;
- Tipul de inteligență artificială utilizat;

- Funcționalități suplimentare oferite utilizatorului.

Prin urmare, din analiza comparativă rezultă că aplicația dezvoltată în cadrul acestei lucrări combină punctele forte ale abordărilor existente, oferind o experiență echilibrată între automatizare, acuratețea recunoașterii vizuale și funcționalități utile pentru utilizatorul final.

Tabela 3.1: Comparatie între aplicațiile analizate

Criteriu de comparatie	MyFitnessPal	Foodvisor	Noom	Alma	Aplicatia propusa
Modalitate de introducere a datelor	Manual	Imagine	Manual, voce	Voce, text, imagine	Imagine, manual
Grad de automatizare a salvării meselor	Scăzut	Mediu	Scăzut	Ridicat	Mediu
Recunoaștere vizuală a alimentelor	Nu	Partial (1)	Nu	Da	Da
Tipuri de inteligență artificială utilizate	Nu	CNN	NLP, ML	NLP, CV, AI generativ	CNN (detecție + clasificare)
Existența jurnalului alimentar	Da	Nu	Da	Da	Da
Obiective nutriționale personalizate	Da	Nu	Da (AI + coaching)	Da	Da

(1) Foodvisor permite recunoașterea alimentului principal, dar nu detecție multiplă.

3.5.2. Concluzii

- **MyFitnessPal** rămâne un standard în domeniul aplicațiilor nutriționale tradiționale, însă nivelul de automatizare și personalizare este limitat.
- **FoodVisor** introduce funcționalități importante în recunoașterea vizuală, dar este restricționată la identificarea unui singur aliment per imagine și nu oferă funcții avansate de gestionare a dietei.
- **Noom** se axează pe schimbarea comportamentului alimentar prin integrarea psihologiei cognitiv-comportamentale și AI conversațională, dar nu include recunoaștere vizuală.
- **Alma** oferă cea mai avansată integrare tehnologică, combinând NLP, Computer Vision și AI generativă pentru o experiență complet automatizată și personalizată.

Aplicația propusă în cadrul acestei lucrări se poziționează între FoodVisor și Alma, oferind o soluție robustă de tip client-server, bazată pe arhitectura YOLOv11 + ResNet50 pentru detecția și clasificarea automată a alimentelor din imagini. Aceasta integrează funcționalități suplimentare precum jurnal alimentar, profiluri de utilizator și setarea de obiective nutriționale, contribuind astfel la o experiență completă și adaptabilă de monitorizare a alimentației.

Capitolul 4. Analiză și fundamentare teoretică

Acest capitol are rolul de a oferi o prezentare structurată a conceptelor teoretice care stau la baza proiectului propus, precum și a justificării alegerilor tehnologice luate. Sunt analizate aspecte esențiale din domenii precum procesarea imaginilor, recunoașterea vizuală cu ajutorul rețelelor neuronale, arhitectura aplicațiilor web, comunicarea între componente client-server, gestionarea datelor. De asemenea, se argumentează necesitatea utilizării unor modele specializate de inteligență artificială, precum și modul în care acestea pot fi integrate într-un sistem complet, capabil să ofere utilizatorului informații utile într-un mod automatizat.

Prin această fundamentare teoretică, se construiește cadrul necesar înțelegerei arhitecturii generale a aplicației, alături de raționamentele care au stat la baza deciziilor luate privind procesarea datelor, alegerile arhitecturale și structurarea bazei de date.

4.1. Analiza obiectivelor proiectului

Această secțiune are ca scop definirea și detalierea obiectivelor urmărite în cadrul proiectului, în urma cărora derivă un set de cerințe funcționale și non-funcționale esențiale sistemului. Prin analizarea acestor cerințe se obține o imagine clară asupra direcției de dezvoltare a aplicației, a rezultatelor așteptate și a modului în care fiecare componentă contribuie la atingerea scopului general al proiectului.

În completarea acestei analize, sunt utilizate diagrame de tip use-case, care au rolul de a reprezenta interacțiunile principale dintre utilizatori și sistem. Aceste diagrame oferă o perspectivă vizuală asupra funcționalităților esențiale și asupra modului în care diferite tipuri de utilizatori (precum utilizatori finali, administratori) interacționează cu aplicația. Ele contribuie la o înțelegere mai bună a cerințelor funcționale și susțin procesul de proiectare ulterioară a sistemului.

De asemenea, în această secțiune vor fi analizate în detaliu și componentele unei rețele neuronale, acestea stau la baza modulului de recunoaștere vizuală a alimentelor din cadrul acestui proiect. Înțelegerea structurii interne și a modului de funcționare al rețelelor neuronale este esențială pentru justificarea alegerilor tehnologice.

4.1.1. Cerințe funcționale

Cerințele funcționale ale unui sistem descriu comportamentele și acțiunile pe care aplicația trebuie să le îndeplinească pentru a răspunde nevoilor utilizatorilor. Acestea definesc în mod concret ce poate face sistemul, ce funcționalități trebuie implementate și cum va interacționa utilizatorul cu platforma.

Cerințele funcționale au fost identificate pornind de la scopul principal al aplicației, acela de a permite recunoașterea automată a alimentelor din imagini și de a sprijini monitorizarea nutriției zilnice. Pe baza acestui obiectiv, aplicația trebuie să includă următoarele funcționalități esențiale pentru cele două tipuri principale de utilizatori:

- **Utilizator** – persoană care folosește aplicația pentru monitorizarea alimentației proprii;
- **Administrator** – utilizator cu privilegii extinse, care are acces la funcționalități

suplimentare de gestionare a sistemului.

Funcționalități pentru utilizatorul obișnuit Utilizatorul are acces la un set restrâns de funcționalități care sunt esențiale și necesare pentru o bună monitorizare a aportului nutrițional, fiind esențiale pentru gestionarea profilului energetic personal:

- **Încărcarea de imagini cu alimente:** utilizatorul trebuie să aibă posibilitatea de a încărca imaginea cu obiectele pe care dorește să le identifice. Ulterior aplicația identifică automat alimentele din imagini folosind un model de viziune computațională, clasifică alimentele detectate în funcție de tipul din care face parte, utilizând un model de clasificare specializat și estimează valorile calorice, nutriționale corespunzătoare alimentelor recunoscute creand un sumar care conține toate detaliile mesei pe care utilizatorul a luat-o.
- **Gestionarea meselor într-un jurnal alimentar zilnic:** utilizatorul poate adăuga, sterge sau edita mesele, oferind astfel o flexibilitate în modul de monitorizare a datelor.
- **Vizualizarea istoricului alimentar:** utilizatorul poate să vadă statisticile detaliate ale nutrientilor și macronutrientilor consumați de acesta în funcție de zile.
- **Definirea și urmărirea unor obiective nutriționale personalizate:** utilizatorul poate să își seteze borne pentru fiecare valoare în parte.
- **Vizualizarea informațiilor claselor de interes:** utilizatorul poate vedea toate informațiile detaliate legate de alimente.
- **Crearea și autentificarea conturilor de utilizator**
- **Schimbarea parolei utilizatorului:** Utilizatorul are posibilitatea de a resetă printr-un link trimis automat pe adresa de email asociată contului.
- **Vizualizarea informațiilor utilizatorului:** utilizatorul poate să își vadă informațiile personale asociate contului de user.
- **Interfață web prietenoasă și intuitivă:** aplicația oferă o interfață accesibilă și modernă, optimizată pentru o utilizare ușoară și eficientă.

Funcționalități specifice administratorului Administratorul are acces la un set extins de funcționalități, pe lângă cele disponibile utilizatorului obișnuit. Aceste funcții sunt esențiale pentru gestionarea conținutului, a utilizatorilor și pentru menținerea unei bune funcționări a aplicației:

- **Vizualizarea tuturor utilizatorilor înregistrati:** administratorul poate accesa o listă completă a conturilor de utilizator, inclusiv informații precum nume, email, data înregistrare.
- **Gestionarea conturilor de utilizator:** permite crearea, editarea sau ștergerea conturilor utilizatorilor.
- **Gestionarea claselor de alimente:** administratorul poate adăuga, modifica sau elimina tipuri de alimente existente în sistem. Acest lucru include denumirea clasei, descrierea, imaginea reprezentativă.
- **Editarea valorilor nutriționale asociate alimentelor:** aplicația oferă administratorului posibilitatea de a actualiza datele nutriționale (kcal, proteine, carbohidrați, grăsimi, fibre, etc.) pentru fiecare aliment în parte, asigurând astfel corectitudinea informațiilor furnizate utilizatorilor.
- **Acces la interfața de administrare:** pagini dedicate cu funcții specifice rolului de administrator, separat de interfața utilizatorului final, pentru o utilizare clară și sigură.

Acstea cerințe stau la baza arhitecturii sistemului și definesc cadrul funcțional necesar pentru ca aplicația să își îndeplinească scopul propus.

4.1.2. Cerințe non-funcționale

Cerințele non-funcționale descriu atributile calitative ale sistemului care nu privesc funcționalitatea specifică a sistemului și stabilesc criterii esențiale pentru performanță, scalabilitate, modularitate, disponibilitate și experiența generală a utilizatorului. Aceste cerințe nu privesc acțiunile directe ale sistemului, ci condițiile în care acesta trebuie să funcționeze eficient și fiabil.

- **Performanță:** sistemul trebuie să răspundă prompt la solicitările utilizatorului, fapt care poate fi imbunătățit prin aplicarea diverselor tehnici de optimizare. Acest punct fiind esențial în funcționarea fluidă a aplicației.
Performanța unui sistem software reprezintă capacitatea acestuia de a executa operațiuni într-un timp cât mai scurt și cu un consum optim de resurse, asigurând un răspuns rapid și fluent la solicitările utilizatorului.
- **Robustete:** performanțele sistemului trebuie să fie robuste, fiind esențial ca sistemul să poată lucra eficient și precis în diferite circumstanțe. Acesta trebuie să gestioneze corect situațiile imprevizibile și să revina într-o stare de stabilitate cât mai rapid posibil.
Robustetea se referă la abilitatea sistemului de a funcționa corect și stabil chiar și în condiții nefavorabile, precum introducerea de date eronate, erori interne sau factori externi imprevizibili, fără a se bloca sau compromite integritatea aplicației.
- **Scalabilitate:** aplicația trebuie să fie capabilă să poată gestiona creșteri ale numărului de utilizatori sau ale datelor procesata fără a-și pierde stabilitatea și de a fi afectată într-un mod semnificativ.
Scalabilitatea unui sistem software reprezintă capacitatea acestuia de a gestiona eficient creșteri ale încărcării — fie că este vorba despre un număr mai mare de utilizatori, date sau cereri — fără a compromite performanța sau stabilitatea funcționării. Un sistem scalabil poate fi extins ușor, atât pe orizontală (prin adăugarea de resurse), cât și pe verticală (prin îmbunătățirea componentelor existente).
- **Modularitate:** sistemul trebuie să fie dezvoltat și extins în module separate și să aibă un grad ridicat de izolare pentru a favoriza extinderea și nu modificarea în procesul de adaugare de noi funcționalități.
Modularitatea este proprietatea arhitecturii software de a organiza funcționalitățile în componente independente (module), care pot fi dezvoltate, testate, întreținute și reutilizate separat. Un sistem modular este mai flexibil, mai ușor de extins și separă clar responsabilitățile.
- **Securitate:** toate datele utilizatorilor trebuie protejate prin criptare, iar accesul la funcționalitățile aplicației se va face pe baza mecanismelor de autentificare și autorizare.
- **Fiabilitate:** aplicația trebuie să funcționeze corect în condiții normale de utilizare, iar rezultatele oferite (recunoaștere aliment, estimări nutriționale) trebuie să fie consistente și predictibile.
- **Ușurință în utilizare:** interfața aplicației trebuie să fie intuitivă, bine structurată, astfel încât utilizatorii să poată accesa rapid funcționalitățile principale fără cunoștințe tehnice avansate.

4.1.3. Diagrame use-case

Diagramele de tip use-case oferă o viziune de ansamblu asupra interacțiunii dintre utilizatori și sistem, evidențiind funcționalitățile disponibile în funcție de tipul de utilizator. În cadrul aplicației, au fost identificate trei categorii principale de utilizatori: utilizatorul neautentificat, utilizatorul autentificat și administratorul autentificat. În continuare sunt prezentate diagramele use-case corespunzătoare fiecărui tip de utilizator.

Diagrama Use Case pentru utilizatorul neautentificat (User) Această diagramă corespunzătoare utilizatorului neautentificat Figura 4.1 prezintă funcționalitățile accesibile unui utilizator care nu s-a autentificat în aplicație. Acestea includ înregistrarea unui cont nou, autentificarea, uitarea parolei, precum și accesul la informații generale, alimentele disponibile.

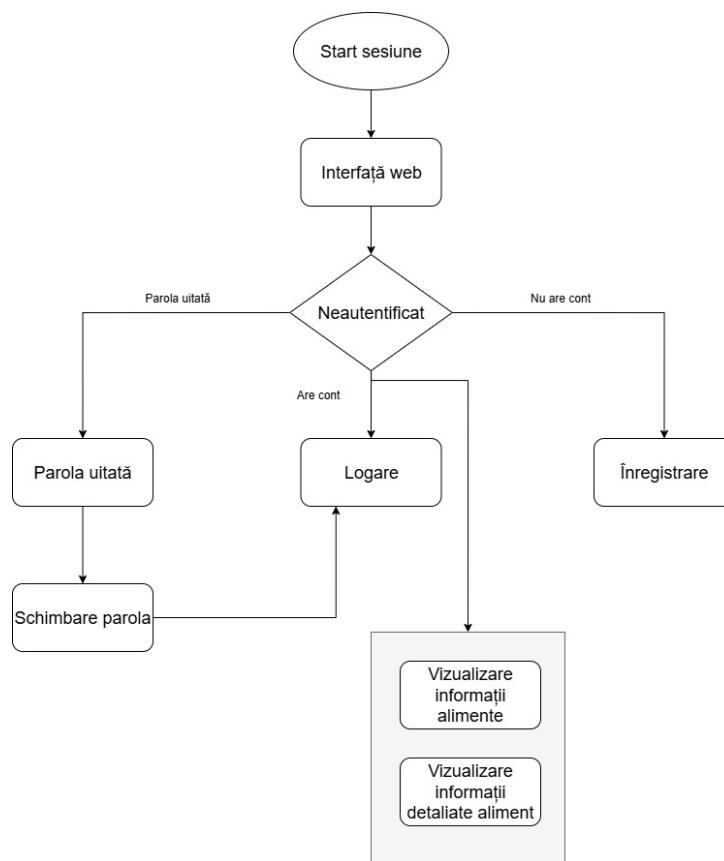


Figura 4.1: Diagrama use-case pentru utilizatorul neautentificat

Diagrama Use Case pentru utilizatorul autentificat (User) Această diagramă corespunzătoare userului Figura 4.2 ilustrează principalele funcționalități disponibile pentru utilizatorul obișnuit după autentificare. Printre acestea se numără, încărcarea imaginilor cu alimente pentru recunoașterea automată a alimentelor, vizualizarea jurnalului alimentar, precum și setarea și urmărirea obiectivelor nutriționale.

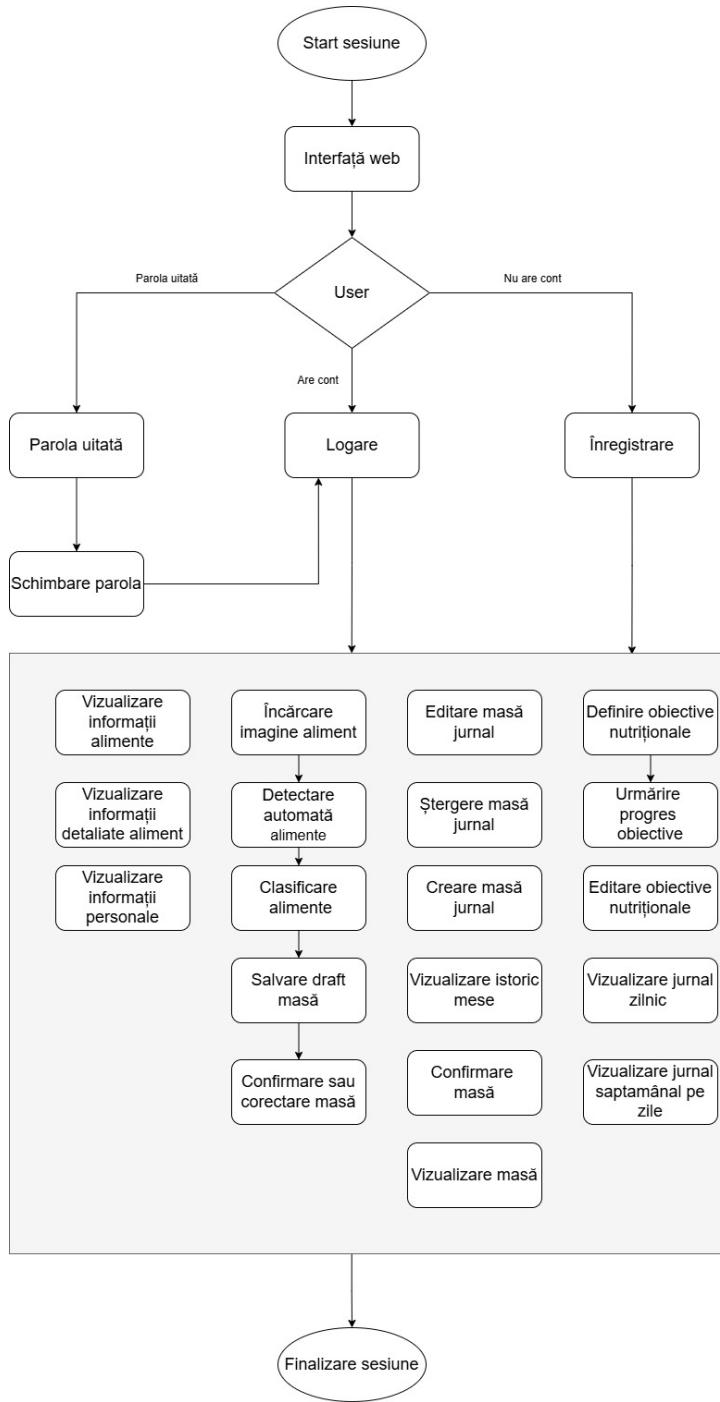


Figura 4.2: Diagrama use-case pentru utilizator(User)

Diagrama Use Case pentru utilizatorul autentificat (Admin) Această diagramă corespunde administratorului. Figura 4.3 evidențiază funcționalitățile specifice rolului de administrator al aplicației. Administratorul are acces la acțiuni precum gestionarea conturilor (creare, vizualizare, editare, ștergere), gestionarea datelor nutriționale introduse în sistem, precum și administrarea bazei de date cu alimente. Aceste funcționalități sunt esențiale pentru menținerea corectitudinii și funcționării optime a aplicației.

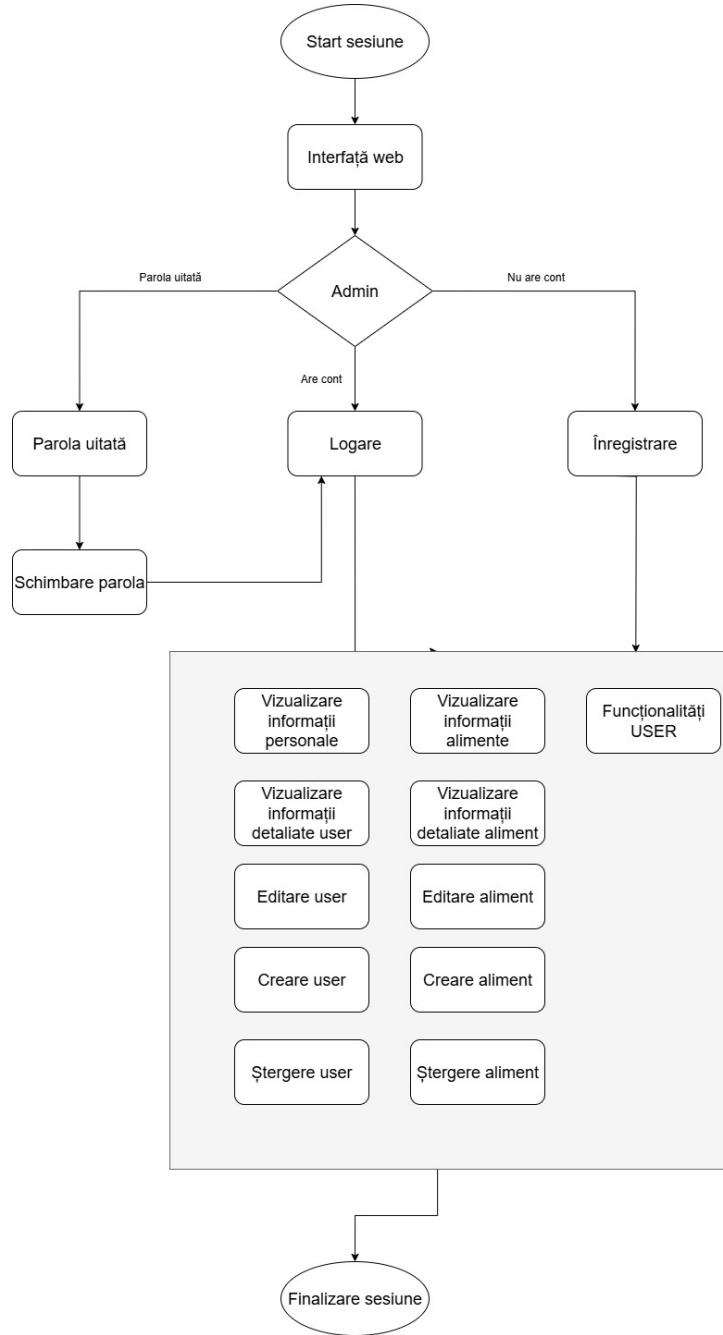


Figura 4.3: Diagrama use-case pentru administrator(Admin)

4.1.4. Scenarii de utilizare

Descrierea scenariilor de utilizare (*use case*) este esențială pentru a înțelege modul în care actorii interacționează cu sistemul și comportamentul acestuia ca răspuns la acțiunile actorilor. În această secțiune sunt prezentate principalele scenarii de utilizare pentru cei doi actori: **Utilizatorul (User)** și **Administratorul (Admin)**.

Fiecare scenariu este descris printr-o succesiune de pași care evidențiază interacțiunea dintre actor și sistem, scopul urmărit și rezultatul așteptat.

Scenariul 1: Încărcarea unei imagini cu alimente

- **Actor principal:** Utilizator

- **Scop:** Identificarea automată a alimentelor și estimarea valorilor nutriționale
- **Pași:**
 1. Utilizatorul se loghează
 2. Utilizatorul accesează pagina aplicației pentru salvarea unei mese dintr-o imagine.
 3. Apasă pe butonul pentru încărcarea imaginii.
 4. Selectează o imagine cu o masă din galeria personală.
 5. Sistemul trimite imaginea către backend.
 6. Backend-ul crează un template automat al mesei cu alimentele identificate și tipul mesei.
 7. Statisticile nutriționale legate de masa sunt calculate.
 8. Aplicația afisează lista alimentelor recunoscute și valorile nutriționale estimate.
 9. Utilizatorul are posibilitatea să editeze predicția dacă au intervenit erori.
 10. Utilizatorul confirmă masa în jurnal sau o șterge.

Scenariul 2: Vizualizarea istoricului meselor

- **Actor principal:** Utilizator
- **Scop:** Vizualizarea istoricului meselor
- **Pași:**
 1. Utilizatorul se autentifică în aplicație.
 2. Accesează pagina destinată vizualizării tuturor meselor.
 3. Sistemul afișează lista meselor într-o formă simplificată.
 4. Utilizatorul poate accesa o masă specifică și va vedea toate detaliile acelei mese, poate să steargă o masă, să o editeze și confirme.
 5. Utilizatorul poate să creeze o masă manual folosind formularul.

Scenariul 3: Vizualizarea istoricului alimentar

- **Actor principal:** Utilizator
- **Scop:** Vizualizarea jurnalului alimentar zilnic/săptămânal și al statisticilor
- **Pași:**
 1. Utilizatorul se autentifică în aplicație.
 2. Accesează pagina destinată sumarului alimentar.
 3. Selectează o zi din calendar sau ultima săptămână.
 4. Sistemul afișează totalul nutrientilor și macronutrientilor din acea zi sau ultimele zile, împreună cu valorile obiectivelor.
 5. Utilizatorul poate modifica obiectivele.
 6. Sistemul afișează o comparație directă între valorile dorite și valorile înregistrate

Scenariul 4: Vizualizarea tuturor alimentelor

- **Actor principal:** utilizator neautentificat
- **Scop:** Vizualizarea alimentelor disponibile și informațiile aferente
- **Pași:**
 1. Accesează pagina destinată alimentelor.
 2. Este afișată lista de alimente.
 3. Utilizatorul poate vedea valorile nutriționale detaliate ale fiecărui aliment, dacă îl accesează.

Scenariul 5: Schimbarea parolei

- **Actor principal:** utilizator neautentificat
- **Scop:** Schimbarea parolei
- **Pași:**
 1. Accesează pagina destinată schimbarii parolei.
 2. Se introduce adresa de email.
 3. Pe adresa de email se primește un link care va fi accesat pentru a finaliza procesul de schimbare a parolei.
 4. Utilizatorul este redirecționat pe o pagina unde își cere noua parolă.

Scenariul 6: Gestionarea utilizatorilor

- **Actor principal:** Administrator
- **Scop:** Vizualizarea, crearea, editarea sau ștergerea conturilor de utilizatori
- **Pași:**
 1. Administratorul se autentifică.
 2. Accesează pagina destinată vizualizării tuturor utilizatorilor.
 3. Sistemul afișează lista utilizatorilor înregistrati.
 4. Administratorul poate vizualiza un utilizator, crea un utilizator, edita informațiile acestuia sau șterge contul respectiv.
 5. Sistemul salvează modificările și actualizează baza de date.

Scenariul 7: Gestionarea tipurilor de alimente

- **Actor principal:** Administrator
- **Scop:** Adăugarea, modificarea sau ștergerea informațiilor despre alimente (tip, valori nutriționale)
- **Pași:**
 1. Administratorul se loghează
 2. Administratorul accesează pagina destinație alimentelor.
 3. Vizualizează lista de alimente existente.
 4. Adaugă un aliment nou, editează un aliment existent sau șterge un aliment existent.
 5. Sistemul salvează modificările și actualizează baza de date.

4.2. Recunoașterea obiectelor din imagini

Recunoașterea imaginilor este o ramură importantă a viziunii artificiale, care urmărește identificarea automată a obiectelor, persoanelor, locurilor, textului sau acțiunilor din imagini și secvențe video. Această tehnologie stă la baza multor aplicații moderne, regăsindu-se în diverse domenii precum asistența medicală bazată pe AI, controlul calității în procesele industriale, conducerea autonomă sau interfețele de realitate augmentată.

Recunoașterea imaginilor este una dintre cele mai importante aplicații ale inteligenței artificiale, cu precădere în ceea ce privește utilizarea metodelor de machine learning și deep learning. Așa cum este subliniat și în articolul publicat de IBM [10], există două direcții principale în abordarea acestei probleme.

4.2.1. Învățarea folosind Machine Learning

Machine learning (ML) este o ramură a inteligenței artificiale care se concentrează pe replicarea modului în care oamenii învăță pentru a permite sistemelor informaticice să

execute sarcini în mod automat și fără a fi programate explicit pentru fiecare situație și permitând învățarea din seturile de date. Prin utilizarea datelor, aceste sisteme pot să-și îmbunătățească performanțele în timp și să se adapteze în funcție de experiență. Așa cum este menționat și în articolul publicat de IBM, “*What is machine learning?*” [11]. Există mai multe tipuri de Machine Learning, fiecare cu aplicabilitate specifică:

- **Învățarea supervizată** presupune utilizarea unor date etichetate, unde fiecare exemplu este asociat cu un răspuns așteptat(etichetă). Modelul învăță să coreleze datele de intrare cu etichetele, ajustându-și parametrii pentru a minimiza eroarea de predicție.
- **Învățarea nesupervizată** se aplică pe seturi de date fără etichete, iar algoritmul identifică automat structuri latente, tipare sau grupări în cadrul datelor.
- **Învățarea semi-supervizată** presupune combinarea unui set mic de date etichetate cu un set mare de date neetichetate, astfel modelul învăță structura generală a datelor și poate să extrapoleze.
- **Învățarea prin recompensă (Reinforcement Learning)** modelul învăță prin interacțiunea cu mediul sau prin penalizari, astfel algoritmul alege acțiunile care maximizează recompensa.

ML permite clasificarea și identificarea obiectelor, acțiunilor sau contextelor vizuale, pe baza unui set de caracteristici extrase din imagini.

În abordarea tradițională de machine learning, procesul de recunoaștere a imaginilor se bazează pe extragerea manuală de către ingineri a caracteristicilor relevante din imagini prin preprocesarea acestora și analizarea lor în concordanță cu obiectivele propuse, urmată de antrenarea unui model de clasificare pe baza acestor caracteristici.

Deși este mai puțin flexibilă decât abordările moderne bazate pe rețele neuronale deep learning, ea oferă un control mai mare asupra procesului de selecție a trăsăturilor și este mai eficientă din punct de vedere computațional pentru seturi de date mici. Dar necesită o implicare umană semnificativă în definirea caracteristicilor și este mai puțin scalabilă în fața problemelor complexe.

4.2.2. Învățarea folosind Deep Learning

Deep learning este o subramură a învățării automate (Machine Learning) care utilizează rețele neuronale artificiale în care arhitectura este structurată în mai multe straturi (denumite *deep neural networks*) pentru a simula procesele decizionale complexe ale creierului uman. Spre deosebire de modelele clasice, care au nevoie de extragerea manuală a trăsăturilor relevante pentru procesul de învățare din date, modelele de *deep learning* pot învăța automat direct din date brute, cum ar fi imaginile, și își pot construi reprezentări interne prin antrenare. Acest principiu este prezentat și în articolul publicat de IBM, “*What is deep learning?..*” [12].

Deep neural networks (rețelele neuronale adânci) sunt alcătuite din multiple straturi de noduri interconectate, unde fiecare strat prelucrează și îmbunătățește datele primite de la stratul anterior într-un proces denumit *forward propagation*. Începând cu stratul de intrare, care primește datele brute, și finalizând cu stratul de ieșire, care generează predicția finală. Acest proces de forward propagation este completat de un alt proces numit *back propagation*, unde folosind algoritmi precum gradient descent calculează erorile și ajustează weight-urile funcției mergând înapoi prin straturi pentru a antrena modelul, erorile sunt propagate înapoi în rețea, iar weighturile (ponderile) sunt actualizate.

Există mai multe tipuri de rețele neuronale profunde, fiecare adaptată unui anumit tip de date sau sarcini, cum ar fi:

- **Rețele neuronale convoluționale (CNN – Convolutional Neural Networks)**
Sunt utilizate în special în procesarea imaginilor și clasificarea vizuală, fiind extrem de eficiente în detectarea trăsăturilor și pattern-urilor din imagini și videoclipuri. Acestea sunt formate din straturi care colaborează pentru a extrage și interpreta automat caracteristici vizuale relevante. CNN-urile reduc nevoia de extractie manuală de trăsături și sunt esențiale în aplicațiile de recunoașterea facială, detecția de obiecte, analiza imaginilor.
- **Rețele neuronale recurente (RNN – Recurrent Neural Networks)**
Acestea sunt proiectate pentru a lucra cu date sevențiale, având bucle de feedback care permit memorarea informației din pașii anteriori. Sunt utilizate pentru sarcini precum traducerea automată, recunoașterea vorbirii și analiza temporală a datelor.
- **Autoencodere variaționale (VAE – Variational Autoencoders)**
Acestea funcționează prin codificarea datelor neetichetate într-o reprezentare comprimată, iar apoi decodifica datele înapoi la forma originală. Acestea sunt folosite în reconstruirea imaginilor corupte sau blurate, acestea nu au doar abilitate de a reconstrui date, dar și de a genera variații ale datelor originale. Sunt formate din blocuri de codificare și de decodificare.
- **Rețele generative adversariale (GAN – Generative Adversarial Networks)**
GAN-urile constă în două rețele care se antrenează împreună: una generatoare, care produce conținut sintetic (de exemplu, imagini), și una discriminatoare, care încearcă să distingă între conținutul real și cel generat. Prin această competiție, generatorul devine din ce în ce mai performant. GAN-urile sunt folosite în aplicații precum generarea de imagini realiste, stilizarea artistică sau simulările virtuale.

4.3. Rețele neuronale convoluționale (CNN)

CNN-urile sunt modele avansate de rețele neuronale special concepute pentru procesarea datelor tridimensionale, acestea sunt utilizate în general în rezolvarea sarcinilor care presupun clasificarea de imagini și recunoașterea de obiecte.

Rețelele neuronale sunt un subdomeniu al machine learning-ului și stau la baza algoritmilor de deep learning. Structura arhitecturală a acestora este reprezentată din mai multe straturi alcătuite din noduri (neuroni artificiali), printre care se numără stratul de intrare (input layer), unul sau mai multe straturi ascunse (hidden layer) și un strat de ieșire (output layer). Nodurile sunt conectate între ele și fiecare asociere are alocată un weight și un threshold (prag).

Dacă valoarea oricărui output este peste valoarea threshold-ului specificat atunci nodul este activat, iar datele se trimit următorului layer din rețea. Altfel, nu se trimit nimic spre stratul următor al rețelei și semnalul este blocat. Acest proces este descris în detaliu și în articolul “What are convolutional neural networks?”, publicat de IBM. [13]

CNN-urile sunt o extensie a rețelelor neuronale artificiale (ANN), arhitectura acesteia fiind detaliată în Figura 4.4, concepute special pentru a lucra cu date de tip matrice, cum ar fi imaginile sau videoclipurile, unde modelele de date joacă un rol semnificativ. Aceste rețele sunt folosite pentru a extrage trăsături din imagini și pentru a le clasifica sau interpreta, procesul este detaliat de asemenea în articolul publicat pe site-ul Geeks for Geeks, “Introduction to Convolution Neural Network”. [14]

Rețelele neuronale artificiale (ANN – Artificial Neural Networks) sunt inspirate din modul de funcționare al creierului uman. Așa cum neuronii biologici primesc, procesează și transmit semnale, neuronii artificiali sunt organizați în noduri conectate între ele, care preiau datele de intrare, le prelucrează și generează o ieșire.

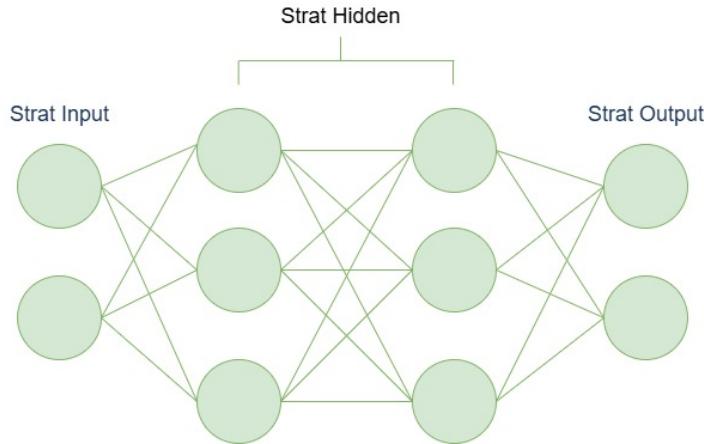


Figura 4.4: Arhitectura generică a unei rețele neuronale artificiale

Conexiunile dintre noduri sunt controlate de valori numerice denumite *weights*, care determină cât de mult influențează o intrare rezultatul final. Pe parcursul procesului de antrenare a rețelei neuronale, valorile weight-urilor sunt modificate automat pentru ca modelul să poată învăța din date și să genereze predicții corecte.

Un rol esențial în acest proces îl joacă *funcțiile de activare*, care decid dacă semnalul generat de un nod este suficient de puternic pentru a fi transmis mai departe. Acest mecanism permite rețelelor neuronale să identifice modele complexe și să realizeze sarcini precum recunoașterea imaginilor sau clasificarea datelor.

4.3.1. Arhitectura Rețelelor Neuronale Convoluționale

O rețea neuronală convoluțională (CNN) are la bază o arhitectură compusă din mai multe tipuri de straturi, fiecare contribuind în mod particular la procesarea datelor de intrare. Acestea includ stratul de intrare (*Input Layer*), straturile convoluționale (*Convolutional Layers*), straturile de subeșantionare (*Pooling Layers*), straturile complet conectate (*Fully Connected Layers*) și stratul de ieșire (*Output Layer*). O reprezentare vizuală a unei astfel de arhitecturi poate fi observată în Figura 4.5.

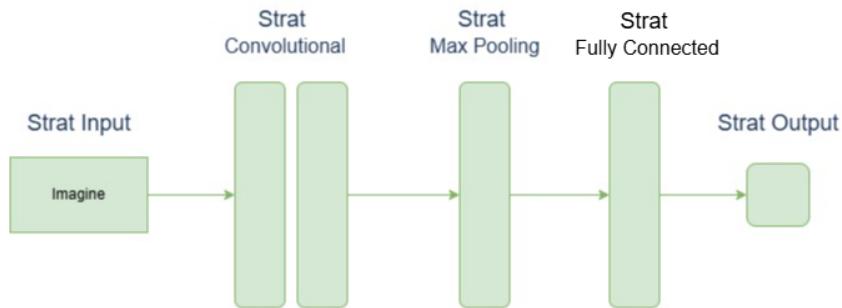


Figura 4.5: Arhitectura conceptuală a rețelelor neuronale convoluționale

Convolutional Layer (Stratul Convolutional) Stratul convolutional reprezintă elementul de bază al unei rețele neuronale convoluționale (CNN) și este stratul unde are loc cea mai mare parte a procesării. Acest strat are nevoie de trei componente principale pentru a funcționa: datele de intrare, un filtru (sau kernel) și o hartă de caracteristici (feature

map). O imagine de intrare color este reprezentată sub forma unei matrice tridimensională de pixeli, cu înălțime, lățime și adâncime (corespunzătoare canalelor RGB).

Procesul de conoluție este reprezentat de glisarea unui filtru de caracteristici, cunoscut sub denumirea de **kernel** sau filtru, pe imagine și verificând dacă caracteristicile căutate sunt prezente.

Detectorul de caracteristici este vector bidimensional (2D) de weight-uri, care reprezintă o parte din imagine. Aceste kernel-uri sunt de obicei o matrice de 3x3. În fiecare zonă analizată, se calculează *produsul scalar* între valorile pixelilor și filtrul aplicat, iar rezultatul este înscris într-un array de output. Acest proces este repetat pe întreaga imagine, mutând filtrul cu un pas fix numit *stride*, proces ilustrat și în Figura 4.6. Output-ul final format din seria de produse scalare realizate se numește **feature map**, **activation map** sau **convolved map**. În loc de R, G și B acum sunt mai multe canale, dar cu lungimi, lățim și înălțimi mai mici, fapt descris și în articolele deja menționate.

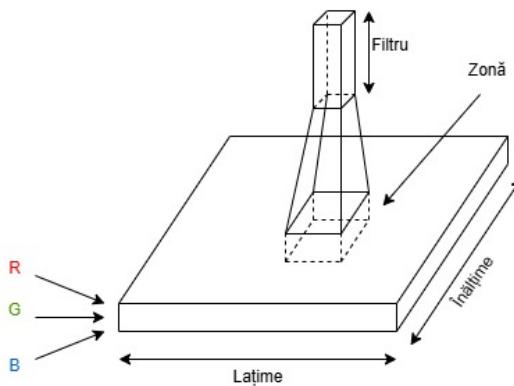


Figura 4.6: Procesul de conoluție

$$\text{Output}(i, j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \text{Input}(i + m, j + n) \cdot \text{Kernel}(m, n) \quad (4.1)$$

În timpul procesului de conoluție se aplică regula partajării parametrilor (parameter sharing), ceea ce înseamnă că același filtru este utilizat pe întregul câmp receptiv al imaginii. Anumiți parametri precum valorile weight-urilor vor fi ajustate în timpul antrenării prin procesul de backpropagation și gradient descent.

Dimensiunea datelor de ieșire generate de un strat conoluțional este determinată de trei hiperparametri principali:

- **Numărul de filtre:** influențează profunzimea volumului de ieșire, fiecare filtru generând o hartă de caracteristici;
- **Stride:** reprezintă numărul de pixeli cu care filtrul (kernel) se deplasează la fiecare pas pe imaginea de intrare;
- **Padding:** controlează modul în care sunt tratate marginile imaginii. Se disting mai multe tipuri:
 - *Valid*: fără padding – dimensiunea ieșirii este mai mică;
 - *Same*: padding astfel încât dimensiunea ieșirii să rămână identică cu cea a intrării;
 - *Full*: se adaugă padding suplimentar pentru ca dimensiunea ieșirii să fie mai mare decât cea a intrării.

După fiecare operație de conoluție, se aplică pe rezultatul obținut (feature map) o **funcție de activare ReLU (Rectified Linear Unit)** introducând nelinearitatea în model pentru a permite învățarea relațiilor mai complexe, fapt ilustrat în Figura 4.7.

$$\text{ReLU}(x) = \begin{cases} x, & \text{dacă } x > 0 \\ 0, & \text{altfel} \end{cases} \quad (4.2)$$

Această funcție este simplă și eficientă din punct de vedere computațional, fiind utilizată frecvent în rețelele neuronale convoluționale (CNN) datorită capacitatii sale de a accelera procesul de antrenare și de a reduce riscul gradientului de dispariție. Prin faptul că păstrează valorile pozitive și anulează cele negative, ReLU ajută la menținerea activă a unor canale informative relevante, în timp ce filtrează activările inutile.

Funcțiile de activare permit ca procesul de backpropagation să se realizeze prin calcularea *gradientilor*, esențiali în actualizarea parametrilor. În cazul ReLU, derivata este simplă și are o formă de tip "treaptă":

$$\frac{d}{dx} \text{ReLU}(x) = \begin{cases} 1, & \text{dacă } x > 0 \\ 0, & \text{dacă } x \leq 0 \end{cases} \quad (4.3)$$

Această derivată este folosită în timpul algoritmului de backpropagation pentru a propaga eroarea înapoi prin rețea. Doar neuronii activați (cu ieșiri pozitive) transmit gradientul mai departe, ceea ce face ReLU eficientă din punct de vedere al învățării. Astfel, funcțiile de activare joacă un rol esențial în învățarea rețelei, asigurând capacitatea acesteia de a aproxima relații nelineare complexe între datele de intrare și ieșire. [15]

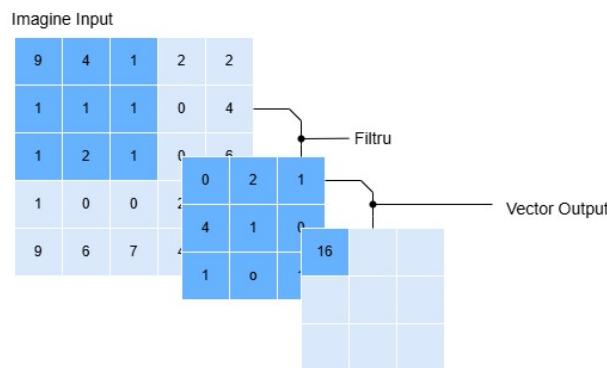


Figura 4.7: Funcția de activare

Rețeaua poate include mai multe straturi convoluționale, ceea ce duce la formarea unei structuri ierarhice. Astfel, straturile superioare pot învăța să recunoască combinații de trăsături detectate de straturile inferioare.

Pooling Layer

Stratul de pooling, cunoscut și sub denumirea de *downsampling*, are rolul de a reduce dimensiunile volumului de date. Deci contribuie la reducerea complexității modelului și la creșterea eficienței, ceea ce duce la scăderea numărului total de parametri ai rețelei. Prin această reducere, modelul devine mai eficient din punct de vedere computațional și mai ușor de antrenat.

Deși principiul de funcționare este asemănător celui al stratului conoluțional, prin glisarea unui kernel peste o regiune a imaginii. Stratul de pooling nu conține weight-uri. În schimb, aplică o funcție de agregare asupra valorilor din regiunea analizată. Acest proces fiind descris și în articolul “Introduction to Pooling Layer” [16].

Principalele beneficii ale stratului de pooling sunt:

- **Reducerea dimensionalității (Dimensionality Reduction):** scade dimensiunea feature map-urilor, ceea ce duce la un model mai rapid și mai eficient;
- **Invarianță la translații (Translation Invariance):** ajuta ca rețeaua să nu fie afectată de mici zgomote și asigură stabilitatea rețelei la mici deplasări sau distorsiuni apărute în imaginea de intrare;
- **Prevenirea overfitting-ului:** reducerea dimensiunilor previne overfitting-ul prin aducerea unei forme de regularizare.;
- **Ierarhizarea trăsăturilor (Feature Hierarchy):** construiește o reprezentarea ierarhica a caracteristicilor, unde straturile inferioare identifică detaliile mai fine, iar straturile superioare identifică caracteristicile mai globale și mai abstracte.

Există două tipuri principale de pooling:

- **Max Pooling:** Când filtrul glisează pe imaginea de input, se va selecta valoarea cea mai mare a regiunii studiate pentru a fi trimisă spre array-ul de output. Conservă cel mai bine caracteristicile esențiale (muchii, texturi, etc.) și oferă mai bune performanțe în cele mai multe cazuri.
- **Average Pooling:** Când filtrul glisează pe imaginea de input, se va selecta valoarea medie a regiunii studiate pentru a fi trimisă spre array-ul de output. Oferă o reprezentare mai generală a datelor de intrare, este util în cazurile în care se dorește o conservare a contextului global.

Procesul de Flattening Flattening-ul reprezintă etapa de conversie a datelor multidimensionale rezultate într-un vector unidimensional. Această transformare este esențială pentru a permite conectarea la stratul fully connected, care are nevoie de vectori ca date de intrare. Scopul acestuia este de a converti datele de ieșire primite de la stratul de pooling, care sunt date reprezentate sub o formă multidimensională (2D), într-un format acceptat de stratul fully connected care acceptă vectori ca input (1D), fapt remarcat și în Figura 4.8, și descris în articolul “Convolutional Neural Network — Lesson 8: Fully Connected Layers and Flattening”.[17].

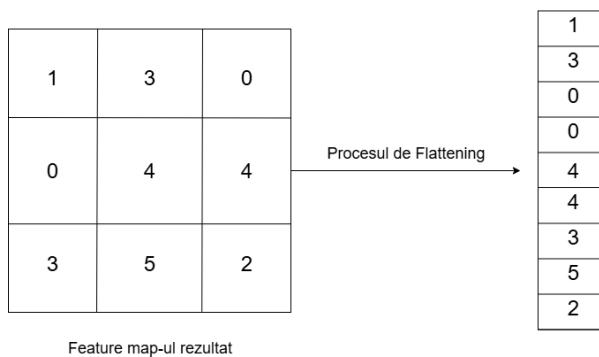


Figura 4.8: Procesul de Flattening

Fully-Connected Layer Fiecare nod dintr-un strat al rețelei din acest bloc este conectat cu fiecare nod aflat în stratul anterior. Aceasta are ca scop efectuarea sarcinilor de clasificare bazate pe caracteristicile extrase din straturile anterioare, procedeu descris și în articolul “What is Fully Connected Layer in Deep Learning?” [18], acesta interpretează feature map-urile generate anterior și oferă predicția finală a categoriei din care se află obiectul.

Stratul fully-connected aplică o funcție de activare specială, denumită Softmax, pentru a realiza clasificarea corectă a datelor de intrare.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (4.4)$$

Unde:

- z_i este logitul (ieșirea brută a stratului anterior) pentru clasa i ;
- K este numărul total de clase;
- e^{z_i} reprezintă exponentiala valorii de ieșire pentru clasa i ;
- $\sum_{j=1}^K e^{z_j}$ este suma tuturor exponentialei logitilor pentru toate clasele — asigură că ieșirile sunt normalizate și însumate la 1.

Softmax transformă valorile brute (logits) generate de rețea într-o distribuție de probabilități, unde suma tuturor probabilităților este egală cu 1. Aceasta funcționează aplicând o funcție exponențială fiecărui logit și normalizând rezultatele, astfel încât să se obțină o distribuție probabilistică asupra tuturor claselor. Astfel, pe baza valorilor generate, rețeaua poate selecta clasa cu cea mai mare probabilitate ca predicție finală, precum reiese și din articolul „Softmax Activation Function in Neural Networks”. [19]

Alte tipuri de funcții de activare sunt:

- **Funcția Sigmoid** este frecvent folosită în clasificarea binară, deoarece transformă valorile într-un interval cuprins între 0 și 1. Totuși, nu normalizează rezultatul pe toate clasele, ceea ce înseamnă că suma probabilităților nu este neapărat egală cu 1.
- **Funcția ReLU (Rectified Linear Unit)** este una dintre cele mai utilizate în hidden layer-ele rețelelor neuronale. Ea păstrează doar valorile pozitive și le anulează pe cele negative. Nu este potrivită pentru stratul de ieșire într-o problemă de clasificare.
- **Funcția Tanh (Tangenta Hiperbolică)** funcționează similar cu Sigmoid, dar returnează valori între -1 și 1, ceea ce poate duce la o convergență mai rapidă în unele cazuri.
- **Funcția Identity** este folosită în probleme de regresie, unde rețeaua trebuie să returneze o valoare numerică continuă, fără nicio transformare.

În funcție de natura problemei și a algoritmului folosit, se pot aplica următoarele forme de regresie:

- **Regresia liniară (Linear Regression)** – utilizată pentru predicții continue. În acest caz, se folosește funcția identitate sau nu se aplică nicio funcție de activare.
- **Regresia logistică (Logistic Regression)** – folosită pentru clasificarea binară, implică aplicarea funcției Sigmoid asupra rezultatului final.
- **Regresia Softmax (Softmax Regression)** – potrivită pentru clasificare multi-clasă, utilizează funcția Softmax pentru a genera probabilități distribuite pe toate clasele.

Procesul de învățare al rețelei continuă după aplicarea funcției de activare, iar pentru ca modelul să învețe, este necesar un mecanism care să măsoare cât de eronate

sunt predicțiile față de valorile reale. Acest rol este îndeplinit de **funcția de pierdere (loss)**, aplicată imediat după ieșirea Softmax. În cazul problemelor de clasificare care conțin mai mult de o singură clasă, cea mai utilizată funcție de pierdere este *Cross-Entropy Loss*, care compara distribuția probabilistică generată de Softmax cu eticheta reală exprimată în format one-hot. Scopul rețelei este să minimizeze această pierdere în timpul antrenării, proces realizat prin algoritmul de backpropagation, care ajustează greutățile modelului în sensul reducerii erorii la fiecare iterație.

$$\mathcal{L} = - \sum_{i=1}^K y_i \log(\hat{y}_i) \quad (4.5)$$

4.4. Fundamentarea procesului de recunoaștere vizuală

Procesul de recunoaștere vizuală în cadrul aplicației propuse se bazează pe un pipeline de prelucrare și procesare a imaginilor, în care fiecare componentă are un rol specific în analiza și interpretarea datelor vizuale. Principalele două etape sunt detectarea obiectelor și clasificarea acestora, realizate prin inferența modelelor de deep learning de tip YOLO și ResNet, astfel se va urmări prezentarea structurii seturilor de date, principiilor de funcționare, arhitectura specifică a fiecărui model, precum și justificarea alegerii acestora în contextul aplicației.

4.4.1. Rolul, structura și preprocesarea seturilor de date

Primul set de date utilizat urmează formatul standard compatibil cu modelul Ultralytics YOLO. Structura datasetului este organizată prin împărțirea datelor în trei subseturi principale:

- *train/* – setul de antrenare;
- *val/* – setul de validare;
- *test/* – setul de testare.

Fiecare subset conține imagini color în format *.jpg*, cu dimensiuni standardizate la 640×640 pixeli, pentru a asigura compatibilitatea cu dimensiunea de intrare a modelului YOLOv11 utilizat. Fiecărei imagini îi corespunde un fișier de etichete *.txt*, care conține, pe linii separate, clasa obiectului și coordonatele normalizate ale bounding box-ului (*x_center*, *y_center*, *width*, *height*), toate exprimate în valori relative față de dimensiunile imaginii.

Structura organizatorică prezentată facilitează separarea clară a procesului de învățare de cel de validare și testare, contribuind la evaluarea obiectivă a performanței modelului, așa cum este descris și în articolul “Object Detection Datasets Overview” [20].

Pentru clasificarea imaginilor cu ajutorul unei rețele ResNet, s-a construit un alt doilea set de date, derivat din primul, prin decuparea automată a regiunilor de interes (bounding boxes) generate de modelul YOLO. Aceste imagini decupate sunt compatibile cu cerințele arhitecturilor de clasificare precum ResNet, după cum este menționat și în articolul “ResNet and ResNetV2” [21]. Fiecare imagine decupată corespunde unei instanțe izolate a unui obiect alimentar, având dimensiuni standardizate pentru a respecta cerințele de intrare ale rețelei ResNet de 224×224 pixeli, 3 canale RGB.

4.4.2. Preprocesarea și augmentarea datelor

Preprocesarea și augmentarea datelor au fost aplicate cu scopul de a îmbunătăți performanța modelelor și de a reduce overfitting-ul. Augmentarea constă în generarea unor versiuni noi ale imaginilor existente, prin aplicarea unor transformări aleatorii care

cresc variația setului de date.

Preprocesare În etapa de preprocesare, imaginile sunt modificate pentru a standardiza valorile și a introduce variații naturale controlate. Tehnicile aplicate sunt descrise și în articolul “Preprocess Images” [22] și includ:

- **Normalizare HSV** (hsv_h , hsv_s , hsv_v), ajustarea canalelor de culoare Hue, Saturation și Value pentru a simula condiții diferite de iluminare și a crește robustețea modelului la variații de culoare;
- **Scalare și translatăre** ($scale$, $translate$), modificarea dimensiunii și poziției obiectelor în imagine, pentru a învăța modelul să recunoască obiectele în forme și poziții variate;
- **Rotiri aleatorii** ($degrees$), aplicarea de rotiri în jurul centrului imaginii pentru a simula diverse orientări ale obiectelor;
- **Shearing** ($shear$), transformări geometrice care înclină imaginea pe axele orizontală sau verticală, simulând distorsiuni sau perspective reale.

Augmentare Pentru a mări diversitatea setului de antrenament vor fi aplicate tehnici variate de augmentare. Aceste tehnici sunt descrise și în articolul “Create Augmented Images” [23] și includ:

- **Flip orizontal și vertical**, inversarea imaginilor pe axa orizontală sau verticală;
- **Mosaic**, combinarea a patru imagini într-un singur cadru, fiecare ocupând un sfert din imaginea finală;
- **MixUp** – fuzionarea a două imagini și a etichetelor lor prin interpolare liniară;
- **Copy-Paste și Close-Mosaic** – inserarea de obiecte decupate dintr-o imagine în alta, menținând corect etichetele și adnotările aferente;
- **Overlap Mask & Mask Ratio** – acoperirea parțială a obiectelor din imagine;

4.4.3. Modelul abstract al detecției de obiecte (YOLO)

YOLO este un model de detecție a obiectelor cunoscut pentru viteza și eficiența sa, deoarece analizează întreaga imagine într-o singură etapă și oferă simultan atât locația obiectelor (prin bounding box-uri), cât și clasele acestora.

Arhitectura YOLOv11 Arhitectura YOLOv11 este compusă din trei componente principale: backbone, neck și head, fiecare având roluri esențiale în procesul de prelucrare a imaginilor și de generare a predicțiilor.

Arhitectura modelului YOLOv11

Arhitectura YOLOv11 este compusă din trei componente principale: **backbone**, **neck** și **head**, fiecare participând în procesul de prelucrare a imaginilor și generare a predicțiilor. Diagrama arhitecturii fiind reprezentată în Figura 4.10

1. Backbone – Extrația caracteristicilor

Backbone-ul YOLOv11 este responsabil pentru extragerea caracteristicilor relevante din imaginea de intrare. Acesta utilizează o serie de blocuri și straturi optimizate pentru a reduce complexitatea computațională, fără a compromite performanța:

- *Blocul C3k2 (Cross Stage Partial with 2-kernel convolution)* – reprezintă o versiune optimizată a blocului CSP (Cross Stage Partial). În loc să folosească o

singură conoluție mare (care este costisitoare din punct de vedere al resurselor), C3k2 împarte operația în două conoluții mai mici, aplicate în paralel sau secvențial. Acest lucru reduce numărul total de parametri ai rețelei. Principiul de funcționare fiind reprezentat în Figura 4.9.

- *SPPF(Spatial Pyramid Pooling - Fast) și C2PSA(Cross Stage Partial Spatial Attention)* – YOLOv11 păstrează un mecanism rapid de extragere a informației numit SPPF în care se aplică max pooling, care ajută la rezumarea trăsăturilor importante din imagine. Pe lângă acesta, introduce și un nou bloc numit C2PSA, care are rolul de a „ghida atenția” modelului spre zonele esențiale din imagine. Astfel, rețeaua devine atentă la obiectele mici sau parțial ascunse, crescând șansele ca acestea să fie detectate corect. Principiul de funcționare fiind reprezentat în Figura 4.9.

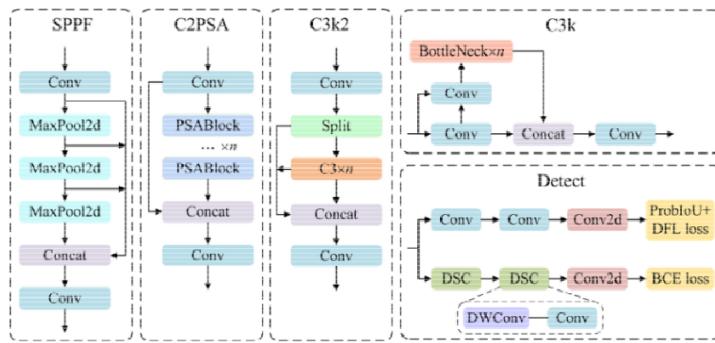


Figura 4.9: Structura modulelor C3K2, SPFF, C2PSA, C3K, Detect, diagrama preluată [24].

2. Neck – Agregarea caracteristicilor

Componenta *neck* combină caracteristicile extrase la diferite rezoluții de către *backbone* și le pregătește pentru generarea predicțiilor. YOLOv11 utilizează optimizări similare cu cele din *backbone*:

- *Blocul C3k2* – înlocuiește blocurile tradiționale, oferind procesare mai rapidă și mai eficientă;
- *C2PSA* – adaugă un mecanism de atenție asupra regiunilor-cheie, permitând o mai bună integrare a caracteristicilor extrase la diferite scări. Astfel, modelul devine mai capabil să detecteze obiecte de dimensiuni variabile.

3. Head – Generarea predicțiilor

Componenta *head* este responsabilă de realizarea predicțiilor finale, care includ atât localizarea (bounding boxes), cât și clasificarea obiectelor:

- *Blocul C3k2* – este folosit și în această etapă pentru a menține eficiența procesării;
- *Stratul Detect* – similar cu cel utilizat în YOLOv8, acest strat produce predicții la trei niveluri de rezoluție: $P3$ (obiecte mici), $P4$ (medii) și $P5$ (mari). Acest mecanism permite detectarea obiectelor la diferite scări, oferind acuratețe ridicată atât pentru obiectele mici și rapide, cât și pentru cele mari și detaliate.

Arhitectura descrisă mai sus este reprezentată în Figura 4.10.

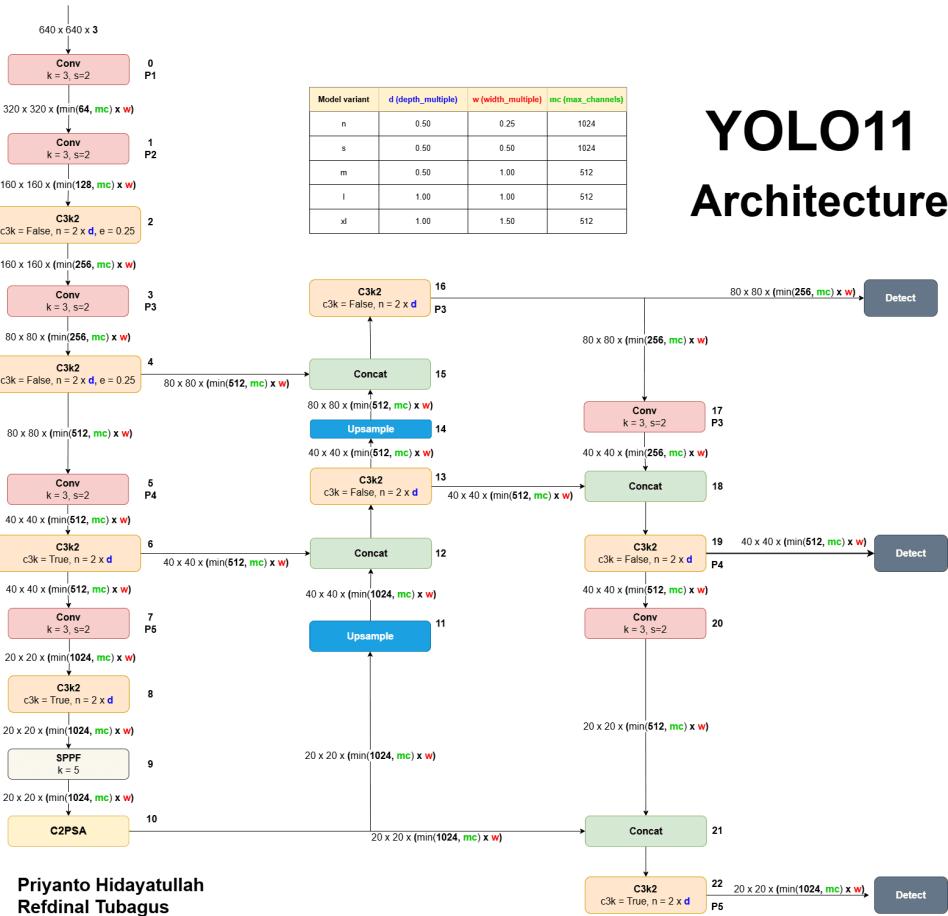


Figura 4.10: Arhitectura YOLOv11, diagrama preluată [25].

4.4.4. Arhitectura modelului ResNet (Residual Network)

ResNet (Residual Network) este o arhitectură de rețea neuronală care a adus o contribuție importantă în domeniul recunoașterii imaginilor. ResNet a fost creată pentru a rezolva o problemă frecvent întâlnită atunci când rețelele neuronale devin foarte adânci: în loc să învețe mai bine, modelul ajunge uneori să obțină rezultate mai slabe, din cauza dificultăților în transmiterea corectă a semnalului înapoi prin rețea, degradarea gradientului. Această abordare este explicată pe larg în articolul "Residual Networks (ResNet)" [26].

Principiul de funcționare: Conexiuni reziduale(Skip Connections) Ideea centrală a arhitecturii ResNet constă în introducerea conexiunilor shortcut. Spre deosebire de rețele neuronale tradiționale în care fluxul de date este transmis secvențial, ResNet permite input-ul unui bloc de layers să fie adăugat la outputul aceluiași bloc. Asta creează blocurile reziduale unde straturile învață o mapare reziduală dată de diferența dintre input și output-ul dorit. Dacă funcția optimală este mai aproape de maparea identitate (unde output-ul va fi asemănător cu input-ul), este mult mai eficient pentru rețeaua neuronală să facă reziduul 0, decât să învețe maparea identitate prin straturile neliniare. Deci în loc ca fiecare strat să învețe o mapare completă a datelor de intrare:

$$H(x) \quad (4.6)$$

ResNet restructurează învățarea sub forma unei mapări reziduale:

$$F(x) = H(x) - x \quad (4.7)$$

Prin urmare, ieșirea devine:

$$H(x) = F(x) + x \quad (4.8)$$

Fapt reprezentat și în Figura 4.11.

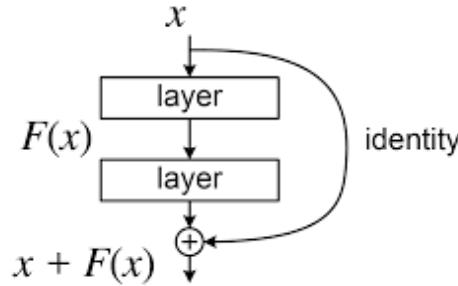


Figura 4.11: Arhitectura Bloc Reziduală [27].

Arhitectura de bază a rețelei ResNet este descrisă și în articolul „Resnet Architecture Explained” [28], aceasta este inspirată din VGG-19 și constă inițial dintr-o rețea conlovuțională „simplă” (plain network) cu 34 de straturi, fără conexiuni reziduale. Ulterior, prin adăugarea conexiunilor shortcut (skip connections), această arhitectură este transformată într-o rețea reziduală (Residual Network). Conexiunile shortcut permit modelului să transmită direct activările între straturi, facilitând propagarea gradientului și evitarea degradării performanței în rețelele adânci, cum se poate observa și în Figură B.1 – Arhitectura ResNet.resnetskipconnectionimg

4.5. Aplicațiile Client–Server

Sistemele informaticice moderne se bazează pe modelul **client–server** pentru a asigura comunicarea între componente. În această arhitectură, funcțiile aplicației sunt împărțite între un *client*, care gestionează interfața cu utilizatorul și solicitările, și un *server*, care procesează cererile și gestionează resursele de date. Această separare permite scalabilitate, modularitate și o mai bună întreținere a aplicațiilor, după cum reiese și din articolul „Client–Server Model” [29].

Modelul client–server se bazează pe următoarele concepte fundamentale:

- **Clientul** este responsabil de interacțiunea cu utilizatorul și de transmiterea cererilor către server;
- **Serverul** primește cererile, le procesează și returnează răspunsurile corespunzătoare;
- **Comunicarea** între client și server se realizează prin protocole standardizate.

4.5.1. Interacțiunea prin API-uri (Application Programming Interfaces)

Pentru a asigura comunicarea între client și server, aplicațiile utilizează API-uri (Application Programming Interfaces). Un API definește un set de reguli prin care componentele aplicației interacționează între ele. În special, *RESTful APIs* sunt utilizate în acest proiect datorită eficienței în transferul de date între client și server, prin operații HTTP standard (GET, POST, PUT, DELETE).

4.6. Justificarea soluției propuse

Soluția propusă se bazează pe integrarea a două tehnologii avansate din domeniul viziunii artificiale: YOLOv11 pentru detecția obiectelor și ResNet-50 pentru clasificarea acestora. Alegerea acestei combinații este motivată atât de performanța ridicată demonstrată, cât și de flexibilitatea și scalabilitatea acestor modele. YOLO a fost selectat pentru capacitatea sa de a detecta obiectele într-un mod eficient, cu o acuratețe ridicată și latență minimă.

ResNet-50 a fost ales ca rețea de clasificare datorită arhitecturii sale reziduale care permite antrenarea eficientă a rețelelor adânci fără probleme de alterare a gradientului. Astfel, ResNet poate învăța reprezentări complexe ale imaginilor și asigură o clasificare robustă chiar și în cazul unor variații subtile între clase.

Această soluție este compatibilă cu infrastructuri moderne de inferență și poate fi integrată cu ușurință în aplicații web, ceea ce o face potrivită pentru implementarea practică a unui sistem intelligent de recunoaștere a alimentelor și estimare nutrițională.

4.6.1. Tehnologii folosite

Pentru dezvoltarea aplicației, au fost utilizate mai multe tehnologii moderne, adaptate fiecărei componente funcționale a sistemului. Alegerea acestora s-a bazat pe criterii precum performanța, scalabilitatea, interoperabilitatea și suportul comunității.

- **Spring Boot** – Framework Java utilizat pentru dezvoltarea back-end-ului aplicației. Oferă suport pentru crearea rapidă a aplicațiilor RESTful, securitate și interacțiune cu baza de date.
- **React** – Bibliotecă Typescript utilizată pentru construirea interfeței grafice (front-end). Permite crearea unei interfețe dinamice, modulare, facilitând interacțiunea utilizatorului cu funcționalitățile aplicației.
- **Python** – Limbaj de programare utilizat pentru dezvoltarea componentei de inferență vizuală, datorită ecosistemului său extins în domeniul inteligenței artificiale și machine learning.
- **PyTorch** – Bibliotecă open-source de machine learning dezvoltată de Meta AI, folosită pentru antrenarea și inferența modelelor de clasificare și detecție a alimentelor. PyTorch oferă suport pentru rețele neuronale dinamice și integrare nativă cu GPU.
- **TensorFlow** – Utilizat experimental pentru testarea alternativelor de inferență vizuală și procesare paralelă, mai ales pentru evaluarea performanței pe arhitecturi diverse.
- **CUDA (Compute Unified Device Architecture)** – Platformă de calcul paralel dezvoltată de NVIDIA, folosită pentru accelerarea antrenării modelelor de deep learning pe GPU-uri compatibile. Permite reducerea semnificativă a timpului de antrenare și inferență.
- **Kaggle** – Platformă online utilizată pentru antrenarea și testarea modelelor de inteligență artificială, oferind suport gratuit pentru execuția pe GPU cu o performanță putere de procesare.
- **PostgreSQL** – Sistemul de gestionare a bazei de date relaționale utilizat pentru stocarea informațiilor. Aceasta oferă suport pentru interogări complexe, tranzacții sigure și scalabilitate pentru volume mari de date.

Capitolul 5. Proiectare de detaliu și implementare

Această secțiune are rolul de a transpune obiectivele teoretice și cerințele funcționale într-o formă concretă, tehnică și implementabilă. După stabilirea arhitecturii generale a aplicației și definirea componentelor esențiale, urmează etapa în care fiecare modul este proiectat în detaliu.

Se vor prezenta structurile de date utilizate, logica din spatele fluxurilor de date, precum și modul în care diversele componente ale sistemului (front-end, back-end) interacționează între ele. Tot în această etapă sunt detaliate deciziile de implementare privind tehnologii alese, structura proiectului, integrarea modelelor de inteligență artificială, gestionarea bazei de date și interfața cu utilizatorul.

5.1. Arhitectura generală a sistemului propus

Arhitectura aplicației propuse urmează modelul client-server și este organizată modular, astfel încât să permită dezvoltarea, testarea și extinderea facilă a componentelor. Sistemul este compus din mai multe părți interconectate: interfața utilizator (frontend), componența de procesare și gestionare a datelor, componența pentru analiză vizuală, și o bază de date pentru stocarea informațiilor nutriționale și a datelor utilizatorilor.

Interacțiunea dintre componente se face prin API-uri REST, iar datele sunt transmise în format JSON. Componența de inferență este responsabilă de procesarea imaginilor încărcate de utilizatori și identificarea alimentelor din imagine.

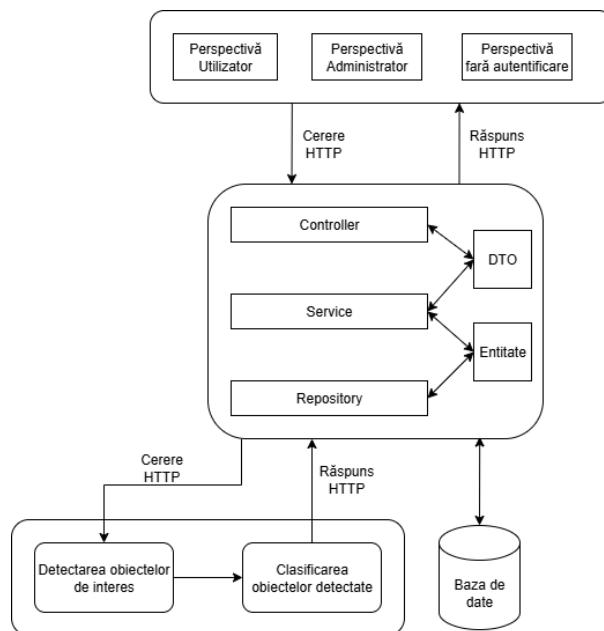


Figura 5.1: Arhitectura generală a sistemului propus

După cum se poate observa în Figura 5.1, sistemul este compus din patru componente principale:

- **Frontend** – interfața cu utilizatorul, unde pot fi încărcate imagini, consultat jurnalul alimentar și setate obiective;
- **Backend - componenta de gestionare a datelor** – gestionează autentificarea, datele utilizatorului, gestionarea datelor alimentelor, obiectivele nutriționale și comunicația cu componenta de analiză vizuală;
- **Backend - componenta de analiză vizuală** – efectuează procesarea imaginilor și returnează rezultatele de clasificare și detectie a zonelor de interes;
- **Baza de date** – stochează informații legate de utilizatori, mese, alimente și date nutriționale.

5.1.1. Componentele aplicației

Aplicația este construită pe o arhitectură modulară, ceea ce înseamnă că fiecare parte a sistemului are un rol bine definit și funcționează independent, dar coordonat cu celelalte componente. În acest mod, dezvoltarea și mențenanța sunt simplificate, iar scalabilitatea pe termen lung este mai ușor de gestionat.

Componentele principale sunt împărțite în trei părți: interfața cu utilizatorul (front-end), serviciul principal de procesare a datelor și serviciul de analiză vizuală, care integrează modelele de inteligență artificială. Această structură reflectă nevoia de separare între logica de prezentare, logica de business și componentele specializate în procesare AI.

În cele ce urmează, vom detalia cum contribuie fiecare componentă la realizarea obiectivelor generale ale sistemului.

5.1.2. Backend - componenta de analiză vizuală

Procesul de recunoaștere vizuală Procesul de recunoaștere vizuală în cadrul aplicației propuse se bazează pe un *pipeline* de prelucrare și procesare a imaginilor, în care fiecare componentă are un rol specific în analiza și interpretarea datelor vizuale.

Etapele procesului Etapele propuse pe care o imagine le parcurge pentru ca informațiile necesare să fie extrase, iar mai apoi prelucrate pentru a fi utilizate în procesele ulterioare sunt:

1. **Primirea imaginii:** Imaginea introdusă de către utilizator este preluată și transmisă către serviciul responsabil pentru analiză.
2. **Detectarea obiectelor:** Modelul de detectie analizează întreaga imagine și returnează coordonatele (*bounding boxes*) ale obiectelor identificate.
3. **Extragerea obiectelor:** Fiecare regiune delimitată de un bounding box este extrasă și transformată într-o imagine individuală, respectând cerințele și standardele modelului de clasificare.
4. **Clasificarea obiectelor:** Modelul de clasificare procesează fiecare obiect extras și atribuie o etichetă care corespunde clasei alimentare din care face parte.
5. **Returnarea rezultatelor:** Etichetele și numărul de obiecte sunt returnate într-un răspuns care este transmis către componenta care a inițiat cererea.

Această succesiune de etape permite o interpretare automată a imaginilor, cu un grad ridicat de acuratețe. Reprezentarea grafică a acestui pipeline este ilustrată în Figura 5.2.

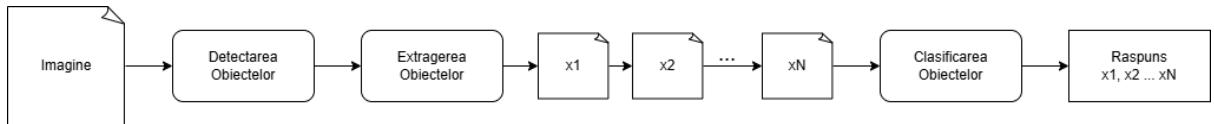


Figura 5.2: Etapele procesului de analiză vizuală

5.1.3. Backend – componenta de gestionare a datelor

Componenta de gestionare a datelor din backend este responsabilă pentru logica de manipulare, validare și persistare a informațiilor primite din partea utilizatorilor sau a altor componente ale aplicației. Arhitectura monolitică a fost aleasă datorită eficienței în contextul unei aplicații cu complexitate redusă și cerințe de scalabilitate limitate.

Aceasta este implementată conform unei arhitecturi stratificate (*layered architecture*), care oferă claritate, modularitate și separare a responsabilităților.

Arhitectura este compusă din următoarele layere:

- **Repository** – Interfață cu baza de date. Fiecare entitate este asociată unui repository care oferă metode CRUD (Create, Retreat, Update, Delete) standard, extinse cu interogări personalizate atunci când este necesar.
- **Service** – Conține logica de business. Aici sunt procesate datele primite din controller, se aplică validări și se coordonează apelurile către repository.
- **Controller** – Acesta primește datele de la client, le validează și le transmite către stratul de service.
- **Entități (Entities)** – Reprezintă structura obiectelor din baza de date. Acestea sunt clasele care modelează tabelele corespunzătoare din baza de date.
- **DTO (Data Transfer Objects)** – Sunt folosite pentru transferul de date între client și server într-un mod controlat și securizat, evitând expunerea directă a entităților.

Structura backend-ului este organizată modular, în pachete dedicate pentru fiecare funcționalitate majoră a aplicației:

- **Pachetul user** – Conține clasele și serviciile asociate gestionării utilizatorilor.
- **Pachetul food** – Găzduiește funcționalitățile legate de gestionarea alimentelor: adăugare, editare, clasificare și asociere cu valorile nutriționale.
- **Pachetul meal** – Se ocupă de gestionarea și compunerea meselor și jurnalul alimentar al utilizatorului.
- **Pachetul goal** – Se ocupă de logica de gestionare a obiectivelor utilizatorilor.
- **Pachetul security** – Conține componentele responsabile de gestionarea autentificării și autorizării utilizatorilor.

Această structură modulară facilitează extinderea sistemului, testarea unitară a fiecărei componente și menținerea clară a separării responsabilităților în cod.

5.1.4. Baza de date

Baza este o componentă esențială în cadrul aplicației, fiind responsabilă cu păstrarea și gestionarea informațiilor într-un mod organizat și sigur. Aici sunt stocate date despre utilizatori, mesele înregistrate, alimentele recunoscute, obiectivele nutriționale stabilite și istoricul alimentar zilnic.

Designul bazei de date reflectă o organizare clară care urmărește logica generală a sistemului.

Modelul a fost gândit astfel încât să permită acces rapid la date și, totodată, să ofere flexibilitatea necesară pentru a putea adăuga ușor noi funcționalități pe viitor.

Diagrama bazei de date, prezentată în Figura 5.3, oferă o imagine de ansamblu asupra principalelor entități și a modului în care acestea sunt conectate între ele.

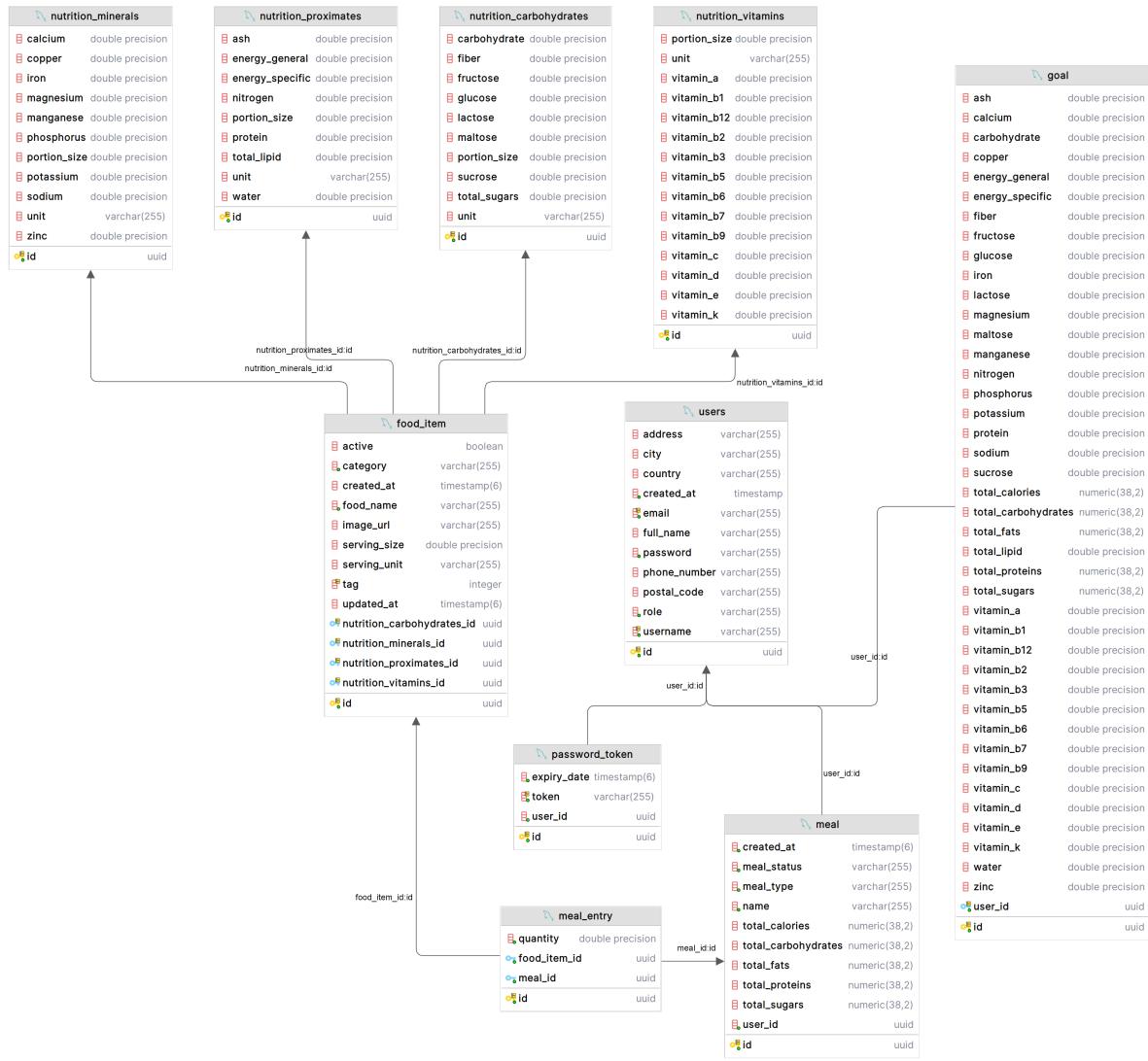


Figura 5.3: Diagrama bazei de date a aplicației

5.1.5. Frontend

Interfața utilizator (frontend-ul) aplicației este realizată folosind biblioteca React, este organizată pe componente clare și reutilizabile, ceea ce face ca dezvoltarea și întreținerea să fie mai ușoare pe termen lung. Aplicația este structurată în pagini bine definite – fiecare pagină conține propriile componente care se ocupă de afișarea datelor, de interacțiunea cu utilizatorul și de trimiterea cererilor către server printr-un serviciu API dedicat.

Navigarea între pagini este gestionată cu ajutorul react-router-dom, folosind un sistem centralizat de rute care leagă adresele URL de componentele corespunzătoare. Astfel, aplicația acoperă toate funcționalitățile importante: autentificarea, gestionarea utilizatorilor, alimentele, mesele zilnice, profilurile personale și statisticile nutriționale.

Pentru a păstra starea utilizatorului chiar și după ce pagina este închisă sau reîncărcată, aplicația salvează datele esențiale (cum ar fi token-ul JWT și informațiile de profil)

în localStorage. Acest mecanism asigură o experiență fluidă, fără a obliga utilizatorul să se reconecteze de fiecare dată.

Accesul la anumite secțiuni este limitat în funcție de rol, prin componente speciale de tip PrivateRoute și AdminRoute, care verifică dacă utilizatorul este autentificat sau are privilegii de administrator.

Logica de comunicare cu serverul este izolată într-un service de tip API, care oferă funcții ușor de folosit pentru a trimite și primi date. Această separare ajută la organizarea codului și face ca aplicația să fie mai clară, mai ușor de testat și mai flexibilă atunci când trebuie extinsă.

Componentele amintite aici vor fi detaliate în subcapitolele care urmează.

5.2. Compunerea seturilor de date

Procesul de construire a celor două seturi de date utilizate în cadrul aplicației propuse este descris în aceasta secțiune:

- Setul de date destinat detecției regiunilor de interes (ROI) corespunzătoare alimentelor;
- Setul de date utilizat pentru clasificarea obiectelor detectate în clase specifice;

5.2.1. Pregătirea și preprocesarea setului de date destinat detecției obiectelor de interes.

Setul de date utilizat pentru antrenarea modelului YOLOv11 a fost construit prin combinarea mai multor seturi de date etichetate, disponibile online. Pentru a obține un dataset unificat și compatibil cu obiectivul propus, au fost aplicate o serie de prelucrări esențiale.

- Fișierele de etichete care nu conțineau adnotări (adică cele care nu identificau niciun obiect) au fost detectate și eliminate, împreună cu imaginile pe care le reprezentau. Această curățare a avut ca scop eliminarea zgomotului din date și îmbunătățirea calității setului de antrenament.

Pentru a automatiza acest proces, a fost utilizat un script care parurge directorul de etichete asociat setului de testare și identifică fișierele .txt cu dimensiune zero (fără conținut/adnotări). Un fișier de etichete gol indică faptul că imaginea asociată nu conține obiecte de interes (fructe sau legume) și, prin urmare, este irelevantă pentru antrenarea unui model de detecție.

Pașii efectuați în procesul de eliminare a imaginilor neetichetate:

1. Definește calea către fișierele de etichete (*labels*) și imagini (*images*);
2. Caută toate fișierele .txt cu dimensiune zero;
3. Pentru fiecare fișier gol identificat:
 - a. Șterge fișierul .txt din directorul de etichete;
 - b. Caută imaginea corespunzătoare (cu același nume, extensie .jpg) în directorul de imagini și o șterge, dacă există.

Această etapă de filtrare a fost esențială pentru asigurarea calității setului de date și evitarea antrenării modelului YOLOv11 pe imagini negative (fără obiecte întă).

- Clasele originale, care reprezentau diverse tipuri de fructe sau legume denumite inconsistent, au fost convertite într-o singură clasă generală denumită "roi" (region of interest). Această conversie s-a realizat printr-un script care a rescris fiecare fișier

de etichete, înlocuind identificatorul clasei originale cu 0. Astfel, modelul a fost antrenat să detecteze orice obiect relevant (fruct sau legumă), fără a diferenția între tipuri. Pentru a realiza această conversie, a fost utilizat un script automatizat care parcurge fiecare fișier de etichete în format .txt și înlocuiește identificatorul clasei inițiale cu o valoare unică 0, corespunzătoare noii clase generice.

Pașii efectuați în procesul de uniformizare ai etichetelor setului de date:

1. Parcurgerea fișierelor .txt din director. Fiecare fișier conține una sau mai multe linii, corespunzătoare obiectelor detectate într-o imagine.
2. Pentru fiecare linie dintr-un fișier:
 - a. Se împarte linia în componente (prin spațiu): prima componentă este ID-ul clasei, iar restul sunt coordonatele normalizate ale obiectului (*x_center*, *y_center*, *width*, *height*).
 - b. Se înlocuiește ID-ul clasei cu 0, păstrând restul informației neschimbată.
 - c. Se reconstruiește linia și se adaugă în lista rezultată.
3. Se rescrie fișierul original cu noile linii în care toate clasele au fost înlocuite cu 0.
4. Se repetă acest proces pentru toate fișierele de etichete, astfel încât întregul set de date să conțină o singură clasă etichetă.

Această etapă de uniformizare a etichetelor a fost esențială pentru asigurarea robustei setului de date și evitarea antrenării modelului YOLOv11 pe date inconsecvente.

5.2.2. Structura și caracteristicile setului de date

Setul de date este organizat conform formatului standard YOLO, fiind împărțit în trei subseturi principale: *train*, *valid* și *test*, fiecare conținând imagini și fișiere de etichete asociate (.txt). Etichetele respectă structura YOLO, fiecare linie din fișier conținând informații despre pozițiile normalizate ale obiectelor detectate în imagine sub forma: *<class_id> <x_center> <y_center> <width> <height>*.

Caracteristici statistice ale setului de date:

- Număr total de imagini: 29.883;
- Număr total de adnotări (bounding boxes): 73.903;
 - Medie: aproximativ 2.5 obiecte per imagine;
- Număr de clase: 1 (monoclasă);
- Dimensiune medie a imaginilor: aproximativ 0.41 megapixeli;
- Rezoluție mediană a imaginilor: 640×640 pixeli;
- Calitate generală:
 - Fișiere fără adnotări: 0;
 - Fișiere nule sau corupte: 0.

5.2.3. Pregătirea și preprocesarea setului de date destinat clasificării regiunilor de interes.

Pentru antrenarea modelului de clasificare ResNet50, a fost necesar un set de date care să conțină imagini centrate pe un singur aliment, etichetate corespunzător cu clasa corespunzătoare. Întrucât imaginile brute conțineau multiple obiecte sau fundaluri

care nu reprezentau un interes în procesul de antrenare, s-a optat pentru extragerea automată a regiunilor de interes (ROI) folosind un model antrenat anterior pentru detecția alimentelor.

A fost creat un set de date personalizat, compus din 30 de clase de fructe și legume.

Procesul de creare a setului de date pentru clasificare s-a bazat pe extragerea și etichetarea automată a imaginilor pe baza detecțiilor realizate de modelul de detecție. Etapele acestui proces sunt descrise în detaliu mai jos:

1. Detectarea obiectelor în imaginile brute:

- (a) Fiecare imagine dintr-un director sursă a fost procesată cu modelul de clasificare;
- (b) Modelul a returnat o listă de regiuni de interes (*bounding boxes*) și etichete de clasă (*class_id*) pentru fiecare obiect detectat.

2. Extragerea regiunilor de interes (cropping):

- (a) Pentru fiecare bounding box, s-a extras zona corespunzătoare din imaginea originală pe baza coordonatelor normalizate (x_c, y_c, w, h);
- (b) Coordonatele au fost convertite din formatul centru-lățime-înălțime în formatul colț-stânga-sus / colț-dreapta-jos ($x_{\min}, y_{\min}, x_{\max}, y_{\max}$) pentru a permite decuparea efectivă a regiunii.

3. Salvarea imaginilor decupate:

- (b) Imaginile au fost salvate într-o structură de directoare, unde fiecare subdirector corespunde unei clase (0, 1, ..., 29).

4. Etichetare implicită prin structură de directoare:

- (a) Fiecare imagine este etichetată automat în funcție de apartenența sa la un anumit director, conform convenției acceptate de bibliotecile PyTorch sau TensorFlow pentru clasificare supravegheată.

5. Verificarea setului de date:

- (a) Corectitudinea repartizării imaginilor a fost verificată vizual și corectată unde era cazul.

Datasetul rezultat a fost împărțit în proporție de 80%-20% pentru antrenare și validare. Transformările aplicate includ:

- Antrenare: redimensionare aleatorie, flip orizontal, jitter de culoare, rotații, affine, perspective.
- Validare: redimensionare + crop central, normalizare.

Acest set de date a fost utilizat în procesul de antrenare a modelului *ResNet*. Distribuția claselor din cadrul acestui set este redată în Tabelul 5.1, fiecare clasă fiind asociată cu un identificator unic numeric.

5.2.4. Concluzia compunerii setului de date

Procesul de construire a seturilor de date a fost esențial pentru buna funcționare a arhitecturii propuse, permitând o separare clară între detecție și clasificare.

Pentru antrenarea modelului de detecție a obiectelor de interes s-a folosit un set de imagini etichetate cu o singură clasă generală, cu scopul exclusiv de a localiza alimentele.

Modelul de clasificare a fost antrenat pe un set de date personalizat, de clasificare, obținut prin decuparea automată a alimentelor detectate, imaginile fiind organizate în cele 30 de clase definite.

Tabela 5.1: Distribuția claselor în setul de date

ID	Clasă	Număr imagini	ID	Clasă	Număr imagini
0	apple	2697	15	gigner	319
1	avocado	1100	16	guava	1300
2	banana	4200	17	lemon	1998
3	beetroot	453	18	lettuce	2004
4	bell_pepper	2708	19	mushroom	845
5	bread	98	20	onion	1409
6	broccoli	2197	21	orange	1953
7	cabbage	1359	22	pear	1101
8	carrot	1461	23	pineapple	2108
9	cauliflower	1293	24	potato	1460
10	cucumber	1125	25	strawberry	1620
11	dragon_fruit	2519	26	sugar_apple	2500
12	egg	2381	27	tomato	1582
13	eggplant	1018	28	watermelon	1533
14	garlic	546	29	zucchini	1305

5.3. Localizarea obiectelor de interes

Pentru detectia regiunilor de interes (ROI) din imaginile brute a fost utilizat modelul YOLOv11. Obiectivul principal a fost identificarea automată a zonelor care conțin fructe sau legume, fără a diferenția între tipurile acestora, pentru a permite extragerea regiunilor relevante care urmează a fi clasificate ulterior cu ajutorul modelului ResNet.

5.3.1. Mediul folosit

Pentru procesul de antrenare a fost utilizată platforma **Kaggle Notebooks**, care pune la dispoziție acceleratoare grafice *NVIDIA Tesla T4* compatibile cu arhitectura *CUDA 12.6*. Aceste GPU-uri oferă o memorie VRAM de 15.360 MB, fiind potrivite pentru sarcini de antrenare intensivă.

Modelul utilizat a fost **YOLOv11x**, cea mai performantă versiune din seria YOLOv11 dezvoltată de Ultralytics. Conform articolului „Ultralytics YOLO11” [30], această versiune aduce îmbunătățiri semnificative față de iterațiile anterioare, printre care se remarcă:

- **Backbone și neck optimizate** – componente arhitecturale responsabile pentru extragerea caracteristicilor vizuale au fost îmbunătățite pentru a capta mai bine detaliile esențiale din imagini;
- **Timp redus de inferență** – modelul YOLOv11x este conceput să ofere răspunsuri rapide, ceea ce îl face potrivit pentru aplicații în timp real;
- **Dimensiune (640 pixeli)** – rezoluția standard de input utilizată pentru antrenare și inferență. O dimensiune mai mare poate îmbunătăți acuratețea, dar crește și costul computațional;
- **mAP val (54.7)** – *mean Average Precision* pe setul de validare, un indicator global al performanței modelului în detectarea obiectelor;
- **Viteză CPU ONNX (462.8 ± 6.7 ms)** – timpul mediu necesar pentru o inferență pe un procesor CPU folosind formatul ONNX, exprimat în milisecunde;
- **Parametrii (56.9 M)** – numărul total de parametri antrenabili ai modelului, exprimat în milioane. Acest număr reflectă complexitatea modelului;

- **FLOPs (194.9 B)** – *Floating Point Operations*, exprimat în miliarde, reprezintă costul computațional per inferență.

5.3.2. Configurarea modelului

Pentru desfășurarea procesului de antrenare, a fost utilizat un mediu de execuție configurat pe platforma Kaggle, care oferă acces gratuit la resurse GPU. Codul de inițializare a inclus următorii pași esențiali:

- **Verificarea disponibilității GPU (NVIDIA T4)**: folosind comanda `!nvidia-smi`;
- **Stabilirea directorului de lucru și afișarea fișierelor disponibile** în setul de date;
- **Instalarea bibliotecilor necesare**: au fost instalate următoarele biblioteci esențiale:
 - *ultralytics* – pentru utilizarea modelului *YOLOv11*;
 - *supervision* – pentru manipularea și vizualizarea etichetelor;
 - *ray[tune]* – pentru posibile optimizări automate ale hiperparametrilor.
- **Validarea mediului de lucru Ultralytics**: comanda `ultralytics.checks()` a fost utilizată pentru a verifica compatibilitatea versiunilor de biblioteci și drivere.

5.3.3. 5.4.3. Antrenarea modelului

Antrenarea modelului s-a realizat în două etape succesive, fiecare constând în câte 20 de epoci. Weight-urile obținute la finalul primei etape au fost utilizate drept punct de plecare pentru cea de-a doua etapă, optimizând astfel timpul de convergență și performanța finală.

Antrenarea a fost configurată pentru o durată de 20 de epoci, cu o dimensiune a imaginilor (imgsz) setată la 640×640 pixeli, standard pentru YOLOv11. Batch size-ul utilizat a fost de 22, iar numărul de workeri pentru încărcarea datelor a fost stabilit la 8, pentru a asigura un flux constant și eficient de date către model în timpul antrenării.

Optimizarea a fost realizată utilizând algoritmul *AdamW*, un optimizator adaptativ cu regularizare prin penalizare a normei L2 (weight decay). Parametrii de antrenare au fost configurați astfel:

- Rata de învățare inițială (*lr0*): 0.0035, cu o rată de învățare finală (*lrf*): 0.015. Ajustată printr-un mecanism de scădere exponențială;
- *Momentum*: 0.94, pentru stabilizarea convergenței;
- *Weight decay*: 0.0004, pentru prevenirea overfitting-ului;
- *Dropout*: 0.05;
- Epoci de încălzire (*warmup_epochs*): 3, utilizat pentru o tranziție lină în faza inițială a învățării;
- Parametri warmup: *warmup_momentum* = 0.85, *warmup_bias_lr* = 0.1.

Pentru a crește diversitatea datelor și capacitatea de generalizare a modelului, au fost aplicate o serie de tehnici de augmentare a imaginilor:

- Augmentări cromatice: ajustarea nuanței (*hsb_h* = 0.015), saturăției (*hsb_s* = 0.5) și luminozității (*hsb_v* = 0.3);
- Transformări geometrice: rotații (*d'*), translații (*translate* = 0.1), scalări (*scale* = 0.3) și forfecări (*shear* = 2.0);
- Flipuri aleatorii: vertical (*flipud* = 0.1) și orizontal (*fliplr* = 0.4);
- Tehnici avansate de compozitie: *mosaic* = 0.8, *mixup* = 0.1, *copy_paste* = 0.1, și *close_mosaic* = 2 pentru combinarea imaginilor într-o manieră realistă;
- Suport pentru mascarea obiectelor suprapuse: *overlap_mask* = True, cu *mask_ratio* = 4.

Validarea modelului a fost activată la fiecare epocă (*val=True*), iar salvarea automată a rezultatelor și generarea de grafice (*save=True*, *plots=True*) au fost de asemenea activate.

5.4. Clasificarea imaginilor

Scopul acestei etape a fost antrenarea unui model de clasificare bazat pe rețeaua ResNet-50 destinată recunoașterii a 30 de clase de fructe și legume.

5.4.1. Mediul folosit

Procesul de antrenare al modelului ResNet-50 a fost realizat pe platforma Kaggle Notebooks, care oferă suport pentru accelerare hardware prin GPU-uri NVIDIA Tesla T4, compatibile cu framework-ul PyTorch. În timpul execuției, a fost activată automat detectia dispozitivelor CUDA, permitând antrenarea modelului pe GPU. În cazul detecției multiplelor dispozitive, s-a activat optional suportul pentru antrenare multi-GPU.

5.4.2. Configurarea modelului

Pentru a pregăti datele necesare antrenării modelului ResNet-50, s-a utilizat setul de imagini clasificate în 30 de clase. Structura folderului corespunde formatului ImageFolder din PyTorch, unde fiecare subdirector reprezintă o clasă distinctă.

Etapele configurației și pregătirii mediului pentru începerea antrenării sunt:

- Augmentarea și preprocesarea datelor: Au fost definite două seturi de transformări – unul pentru antrenare și altul pentru validare. Imaginilor care nu au o formă patrată le-au fost adăugate margini pentru a nu strica raportul obiectelor. Transformările aplicate imaginilor de antrenament includ tehnici de augmentare vizuală (rotații, translatații, distorsiuni de perspectivă etc.) pentru a crește diversitatea și robustețea modelului. Pentru validare, imaginile au fost doar redimensionate și normalizate pentru a reflecta fidel condițiile reale de inferență.
- Încărcarea datasetului și împărțirea în subseturi: Setul de date a fost încărcat folosind ImageFolder, metodă compatibilă cu o structură de directoare în care fiecare subfolder reprezintă o clasă. Ulterior, datele au fost împărțite în 80% pentru antrenare și 20% pentru validare, utilizând funcția random split. S-a asigurat ca fiecare subset să primească transformările corespunzătoare (train sau val), prin atribuirea explicită a câmpului `.transform`.
- Inițializarea DataLoader-elor: Au fost create obiecte DataLoader pentru încărcarea batch-urilor de imagini în timpul antrenării și validării. Parametrii utilizați optimizează fluxul de date:
 - `batch_size = 128` – echilibrul între memorie și viteza de procesare;
 - `shuffle = True` pentru antrenament – asigură o distribuție aleatorie a datelor în fiecare epocă;
 - `pin_memory = True` – permite copierea mai rapidă a tensorilor către GPU;
 - `num_workers` – configurat în funcție de disponibilitatea hardware-ului.

5.4.3. Antrenarea modelului

Modelul utilizat pentru clasificarea imaginilor este ResNet-50, o rețea convecțională adâncă cu arhitectură reziduală, pre-antrenată pe setul de date ImageNet (IMAGENET1K_V2).

Pentru a adapta modelul la setul de date utilizat în proiect, compus din 30 de clase

distincte de fructe și legume, a fost necesară modificarea stratului fully connected (fc) din ResNet-50. În mod implicit, acest strat are dimensiunea de ieșire 1000, corespunzătoare claselor din ImageNet. Aceasta a fost înlocuit cu un strat nn.Linear(2048, 30), unde 2048 reprezintă dimensiunea vectorului de caracteristici extrase de blocurile convoluționale, iar 30 este numărul de clase din datasetul personalizat.

Configurarea procesului de antrenare Pentru instruirea modelului, au fost utilizate următorii parametri și metode de optimizare:

- Funcția de pierdere (Loss function): CrossEntropyLoss, este funcția folosită în antrenarea acestui model de clasificare de obiecte ale mai multor clase.
- Optimizator: Adam, cu o rată de învățare inițială setată la 0.001, ales pentru convergența sa rapidă și stabilitate în antrenarea modelelor profunde.
- Scheduler: StepLR reduce rata de învățare cu un factor de 0.1 la fiecare 3 epoci, pentru a permite o rafinare a învățării pe măsură ce modelul se apropiie de convergență.
- Batch size: 128, ales pentru a echilibra între viteza de antrenare și utilizarea memoriei GPU.
- Mixed precision training: activat prin modulul torch.amp, care permite executarea anumitor operații în precizie redusă (float16) pentru a accelera antrenarea și a reduce consumul de memorie fără pierderi semnificative în precizie.

Antrenarea modelului Modelul a fost antrenat pe o perioadă de 10 epoci, fiecare epocă constând în două faze principale:

1. Faza de antrenare (*model.train()*)
 - Modelul a procesat fiecare mini-batch de imagini.
 - Pentru fiecare iterare s-au efectuat următorii pași: *forward pass*, calculul eroarei, *backward pass* și actualizarea parametrilor.
 - A fost utilizată tehnica mixed precision training, folosind *torch.amp.autocast* și *GradScaler*, pentru creșterea performanței și reducerea consumului de memorie GPU.
2. Faza de validare (*model.eval()*)
 - Modelul a fost evaluat pe un set de validare separat, fără actualizarea parametrilor.
 - Au fost calculate pierderea medie și acuratețea pe epocă, pentru a evalua performanța modelului și a detecta eventualul fenomen de *overfitting*.

Toate modelele au fost salvate la finalul fiecărei epoci, cu nume corespunzătoare, iar modelul final antrenat a fost salvat separat sub denumirea *resnet_fruits_final.pth*. Această practică asigură atât posibilitatea reluării antrenării de la o anumită epocă, cât și conservarea celui mai performant model pentru utilizare ulterioară. Implementarea fiind prezentă în Figura 5.4.

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
from torchvision import datasets, models, transforms
import torch.amp

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

### Initialize pre-trained ResNet-50 model
model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(dataset.classes))

model = model.to(device)

### Define Function and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
scheduler = StepLR(optimizer, step_size=3, gamma=0.1) # Reduce lr for every 3 epochs

### Mixed Precision Training
scaler = torch.amp.GradScaler()

num_epochs = 10

### Model Training
for epoch in range(num_epochs):
    print("-" * 50)
    print(f"\n START Epoch [{epoch+1}/{num_epochs}]")
    print("-" * 50)

    model.train()
    running_loss = 0.0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True)

        optimizer.zero_grad() # Reset gradients

        # Mixed Precision
        with torch.amp.autocast(device_type='cuda'): # Activate mixed precision
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        # Backpropagation cu mixed precision
        scaler.scale(loss).backward() # Calculate gradients
        scaler.step(optimizer) # Optimizer steps
        scaler.update() # Update scaler for mixed precision

        running_loss += loss.item()

    scheduler.step() # Update scheduler for lr

    ##### Evaluate the data set for validation
    model.eval()
    val_loss, correct, total = 0.0, 0, 0

    with torch.no_grad():
        for inputs, labels in val_loader: # No gradient in evaluation mode
            inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True)

            with torch.amp.autocast(device_type='cuda'): # Activate mixed precision
                outputs = model(inputs)
                loss = criterion(outputs, labels)

            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels.sum()).item()
            total += labels.size(0)

        val_acc = correct / total
        print("-" * 50)
        print(f"\n Epoch [{epoch+1}/{num_epochs}] COMPLETED")
        print(f"\n Train Loss: {running_loss/len(train_loader):.4f}")
        print(f"\n Val Loss: {val_loss/len(val_loader):.4f} | Accuracy: {val_acc:.4%}")
        print("-" * 50)

    ### Model is saved for every epoch
    model_save_path = f"resnet_fruits_epoch_{epoch+1}.pth"
    torch.save(model.state_dict(), model_save_path)
    print(f"\n Model salvat: {model_save_path}")

    ### Final model is saved after the training process is completed
    model_save_path_final = "resnet_fruits_final.pth"
    torch.save(model.state_dict(), model_save_path_final)
    print(f"\n Model final salvat: {model_save_path_final}")

```

Figura 5.4: Antrenare model de clasificare

5.4.4. Procesul de fine-tuning

După obținerea unui modelului antrenat inițial pe imagini decupate automat, s-a realizat o etapă suplimentară de fine-tuning, cu scopul de a rafina capacitatea de clasificare. Această etapă a vizat exclusiv ajustarea stratului final fully-connected, permitând

adaptarea la distribuția specifică a datelor, cu un cost computațional minim.

1. Încărcarea modelului pre-antrenat

- A fost utilizat modelul *ResNet-50* pre-antrenat pe ImageNet
- Weighturile antrenate anterior au fost încărcate din fișierul *resnet_fruits_epoch_6.pth*.

2. Înghetarea straturilor

- S-a aplicat o strategie de înghetare totală a parametrilor, cu excepția stratului final *fc*.
- Această abordare asigură păstrarea reprezentărilor vizuale învățate în etapele anterioare, concentrând învățarea doar asupra deciziei finale de clasificare.

3. Optimizator și funcția de pierdere

- Optimizarea a fost realizată cu algoritmul *Adam*, aplicat exclusiv parametrilor stratului *fc*, cu o rată de învățare de 1×10^{-4} și regularizare L2 (*weight_decay = 1e-4*).
- Pentru calculul erorii, s-a utilizat funcția de pierdere *CrossEntropyLoss*, adecvată clasificării cu clase multiple.

4. Scheduler-ul de rată de învățare

- A fost integrat scheduler-ul *ReduceLROnPlateau*, configurat să reducă rata de învățare cu un factor de 0,3 în cazul stagnării acurateței pe setul de validare.

5. Antrenare cu *mixed precision*

- Antrenarea s-a desfășurat pe durata a 3 epoci, folosind *mixed precision training* prin *torch.amp*, pentru reducerea timpului de execuție și a utilizării memoriei GPU.

6. Evaluare și salvarea modelului

- La finalul epocilor, modelul a fost evaluat pe setul de validare, iar acuratețea și loss-ul au fost monitorizate.
- Modelul antrenat a fost salvat după fiecare epocă.
- Evoluția acurateței a demonstrat o îmbunătățire progresivă, confirmând eficiența fine-tuning-ului asupra stratului de decizie.

Implementarea întregului proces fiind prezentă în Figura 5.5

```

import torch
import torch.nn as nn
import torchvision.models as models
from torch.optim.lr_scheduler import ReduceLROnPlateau

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#Load the pre-trained ResNet50 model and replace the classification head
model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 30)

#Load the weights from the previously trained model
model.load_state_dict(torch.load("/kaggle/input/resnet_fruits_best_6/pytorch/default/1/resnet_fruits_epoch
_6.pth", map_location=device))
model = model.to(device)
print("Model loaded!")

#Freeze all layers except for the final fully-connected (fc) layer
for name, param in model.named_parameters():
    param.requires_grad = name.startswith("fc")

#Define optimizer for the final layer only with low learning rate
optimizer = torch.optim.Adam(model.fc.parameters(), lr=1e-4, weight_decay=1e-4)
criterion = nn.CrossEntropyLoss()

#Learning rate scheduler based on validation accuracy
scheduler = ReduceLROnPlateau(optimizer, mode='max', factor=0.3, patience=1, threshold=0.002)
scaler = torch.amp.GradScaler()

num_epochs = 3

#Fine-tuning loop (only for the final layer)
for epoch in range(num_epochs):
    print(f"\nEpoch {epoch+1}/{num_epochs}")
    model.train()
    train_loss = 0.0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()

        with torch.amp.autocast(device_type='cuda'):
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        train_loss += loss.item()

    #Evaluation on validation set
    model.eval()
    val_loss, correct, total = 0.0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            with torch.amp.autocast(device_type='cuda'):
                outputs = model(inputs)
                loss = criterion(outputs, labels)

            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

    val_acc = correct / total
    scheduler.step(val_acc)

    print(f"Train Loss: {train_loss/len(train_loader):.4f}")
    print(f"Val Loss: {val_loss/len(val_loader):.4f} | Accuracy: {val_acc:.4%}")

    #Save model after each epoch
    torch.save(model.state_dict(), f"resnet_fruits_finetuned_fc_only_epoch_{epoch+1}.pth")

#Save the final fine-tuned model
torch.save(model.state_dict(), "resnet_fruits_finetuned_fc_only_best.pt")
print("Fine-tuning completed and model saved!")

```

Figura 5.5: Fine tuning al modelului de clasificare

5.5. Componenta de analiză vizuală

Această componentă integrează modelele antrenate pentru detectia obiectelor de interes și clasificarea acestora în clasele corespunzătoare.

Aplicația este expusă ca API REST folosind framework-ul FastAPI, permitând utilizatorilor să trimită imagini și să primească etichetele clasificate pentru fiecare obiect detectat.

5.5.1. Încărcarea modelelor

Funcția care încarcă cele două modele se numește `load_model`:

- Modelul de detecție al obiectelor de interes: încărcat dintr-un fișier `.pt` cu ajutorul bibliotecii `ultralytics`. Acest model detectează obiectele din imagine și returnează coordonatele bounding box-urilor.
- Modelul de clasificare: modificat pentru a avea 30 clase corespunzătoare datasetului utilizat. Weight-urile sunt încărcate dintr-un fișier `.pth`.

Ambele modele sunt setate în modul de evaluare (`eval()`) și mutate pe dispozitivul disponibil (CPU sau GPU).

5.5.2. Detectia și segmentarea obiectelor de interes

Procesul începe prin încărcarea imaginii și convertirea acesteia într-un format RGB, pentru a asigura compatibilitatea cu modelul de detecție. Ulterior, imaginea este analizată folosind modelul de detecție, care identifică bounding box-urile care corespund obiectelor detectate. Pentru fiecare dintre aceste zone, se extrage porțiunea relevantă din imagine prin decuparea obiectului recunoscut. La final, se returnează o listă cu toate imaginile decupate. În cazul în care modelul nu detectează niciun obiect, imaginea originală este returnată ca atare.

5.5.3. Transformările imaginilor segmentate

Pentru ca imaginile decupate de modelul de detecție să poată fi utilizate în etapa de clasificare folosind rețea de clasificare, este necesar un proces de preprocesare care să asigure compatibilitatea cu cerințele modelului. Transformările aplicate au rolul de a aduce toate imaginile la același format și de a le normaliza în mod corespunzător.

Primul pas constă în redimensionarea imaginii astfel încât cea mai lungă latură să aibă dimensiunea dorită (implicit 224 pixeli), menținând proporțiile originale. După redimensionare, imaginea poate avea o formă dreptunghiulară, motiv pentru care se adaugă margini (padding) negre pentru a o transforma într-un pătrat perfect. Se calculează dimensiunile noii imagini și diferențele față de dimensiunea tintă pe fiecare axă, pentru a determina cât padding trebuie aplicat pe fiecare latură. Acest padding este distribuit simetric (stânga-dreapta și sus-jos), iar zonele adăugate sunt umplute cu negru (valoare RGB: 0), pentru a nu influența vizual modelul de clasificare.

După redimensionare, imaginea este transformată într-un tensor — adică într-o structură numerică specifică PyTorch, ce poate fi procesată de modelul de deep learning.

În final, valorile pixelilor sunt normalizate utilizând media și deviația standard aferente setului de date ImageNet, pe care modelul ResNet a fost inițial antrenat. Acest pas este crucial pentru ca modelul să poată interpreta corect noile date, care trebuie să respecte aceeași distribuție statistică ca datele originale.

Prin aplicarea acestor transformări, imaginile devin compatibile cu ResNet și sunt pregătite pentru etapa de inferență, unde se va realiza clasificarea lor.

5.5.4. Clasificarea imaginilor

Odată aplicate transformările, imaginea este transmisă către modelul de clasificare.

Modelul încărcat în modul de inferență, fără activarea mecanismului de învățare, procesează imaginea și generează o predicție. Pentru a evita calculul gradientului, procesul are loc în contextul `torch.no_grad()`, ceea ce reduce consumul de memorie și accelerează inferența.

În urma procesării, modelul returnează un vector de probabilități asociate fiecarei clase. Se selectează clasa cu probabilitatea cea mai mare (prin *argmax*), iar indexul obținut este mapat la o etichetă corespunzătoare.

Această componentă returnează o listă de obiecte JSON care indică rezultatele inferenței vizuale. Pentru fiecare clasă identificată în imagine, este furnizat un identificator numeric (tag) și numărul total de apariții (count). Clasificarea se face pe baza imaginilor decupate de YOLO și analizate cu ResNet. Astfel, rezultatul final reflectă frecvența fiecărei clase de obiecte detectate în imaginea analizată.

5.6. Componenta de gestionare a datelor

Componenta de gestionare a datelor are un rol esențial în aplicație, aceasta ocupându-se de salvarea, actualizarea și oferirea datelor nutriționale, a informațiilor despre utilizatori, mese și obiective alimentare. Funcționalitățile sunt integrate într-o aplicație Spring Boot de tip monolit, structurată clar pe straturi: controller, service și repository-ul care interacționează cu baza de date, acestă subsecțiune își propune descrierea acestora.

5.6.1. Baza de date nutrițională

Un element central al acestei componente este baza de date nutrițională, proiectată și populată printr-un script SQL în care se inserează informațiile fiecărei clase, pentru a reflecta fidel compoziția alimentelor analizate în aplicație. Structura bazei include următoarele categorii de informații:

- **Proximate composition** – include date despre umiditate, energie (energy_general, energy_specific), proteine, lipide totale, azot și conținutul de cenușă; Valori energetice – exprimate atât în kcal (energy_general), cât și în kJ (energy_specific) per 100g produs;
- **Carbohidrați** – carbohidrați totali, fibre, zaharuri simple (glucoză, fructoză, sucroză, lactoză, maltoză) și zaharuri totale (total_sugars);
- **Minerale** – minerale esențiale precum calciu, fier, magneziu, fosfor, potasiu, sodiu;
- **Vitamine** – vitaminele B1, B2, B3, B5, B6, B7, B9, B12, C, A, D, E și K;
- **Metadate alimentare** – denumirea (food_name), categoria (category), porția standard (serving_size, serving_unit), imagine ilustrativă (image_url) și starea (active).

Fiecare categorie este organizată în tabele separate pentru proximate, carbohidrați, vitamine și minerale, fiecare aliment denumit (FoodItem) în cadrul aplicației, având relații *OneToOne* cu aceste tabele, un exemplu de inserare a unui obiect este reprezentată și în Figura 5.6. Această separare permite gestionarea detaliată a fiecărei componente nutriționale și facilitează extinderea ulterioară a bazei de date.

Normalizarea și modelarea relațiilor Structura generală a bazei de date a fost proiectată conform principiilor normalizării. Această abordare asigură eliminarea redundanțelor, menținerea consistenței datelor și creșterea eficienței în operațiile de actualizare. Figura 5.3 Fiecare tabel este specializat într-o categorie distinctă de informații (ex. *proximate*, *carbohidrați*, *minerale*, *vitamine*), astfel încât fiecare atribut să depindă funcțional doar de cheia primară a tabelei în care se află. De exemplu, valorile de tip „proteină”, „grăsimi” și „energie” sunt stocate exclusiv în tabelul *nutrition_proximates*, evitând replicarea acelorași câmpuri în alte contexte.

Aceleasi principii au fost aplicate și pentru celelalte entități și informații structurate în tabele astfel baza de date devine scalabilă, clar structurată și ușor de întreținut,

facilitând totodată integrarea ulterioară a unor noi funcționalități.

```
-- Apple

INSERT INTO nutrition_proximates (
    id, portion_size, unit, water, energy_general, energy_specific,
    nitrogen, protein, total_lipid, ash
) VALUES (
    '1da71bb6-6737-42ec-99a7-010965000000', 100.0, 'g', 84.7, 62.0, 56.0,
    0.03, 0.19, 0.21, 0.15
);

INSERT INTO nutrition_minerals (
    id, portion_size, unit, calcium, iron, magnesium, phosphorus,
    potassium, sodium, zinc, copper, manganese
) VALUES (
    '1da71bb6-6737-42ec-99a7-010965000001', 100.0, 'mg', 5.0, 0.1, 4.7, 9.0,
    95.0, 1.0, 0.02, 0.024, 0.029
);

INSERT INTO nutrition_carbohydrates (
    id, portion_size, unit, carbohydrate, fiber, total_sugars, sucrose,
    glucose, fructose, maltose, lactose
) VALUES (
    '1da71bb6-6737-42ec-99a7-010965000002', 100.0, 'g', 14.8, 2.0, 12.2, 1.32,
    3.09, 7.81, 0.15, 0.15
);

INSERT INTO nutrition_vitamins (
    id, portion_size, unit, vitamin_a, vitamin_b1, vitamin_b2,
    vitamin_b3, vitamin_b5, vitamin_b6, vitamin_b7,
    vitamin_b9, vitamin_b12, vitamin_c, vitamin_d,
    vitamin_e, vitamin_k
) VALUES (
    '1da71bb6-6737-42ec-99a7-010965000003', 100.0, 'mg', NULL, 0.009, 0.066,
    0.09, NULL, 0.021, NULL,
    6.0, NULL, NULL, NULL,
    NULL, NULL
);

INSERT INTO food_item (
    id, tag, food_name, category, image_url, serving_size, serving_unit,
    created_at, updated_at, active,
    nutrition_proximates_id, nutrition_minerals_id, nutrition_carbohydrates_id, nutrition_vitamins_id
) VALUES (
    '1da71bb6-6737-42ec-99a7-010965000004', 0, 'Apple', 'Fruit',
    'https://example.com/images/apple.jpg', 200.0, 'g',
    '2025-06-18T14:00:00.000+00:00', '2025-06-18T14:00:00.000+00:00', TRUE,
    '1da71bb6-6737-42ec-99a7-010965000000', '1da71bb6-6737-42ec-99a7-010965000001',
    '1da71bb6-6737-42ec-99a7-010965000002', '1da71bb6-6737-42ec-99a7-010965000003'
);
```

Figura 5.6: Inserarea unui aliment în baza de date

5.6.2. Stratul de Repository

Pentru accesul la date, aplicația utilizează stratul *repository* implementat cu ajutorul framework-ului Spring Data JPA.

Pentru a optimiza încărcarea datelor au fost definite proiecții personalizate prin interfețe *FoodItemSimpleProjection*, utilizate în interogările care nu necesită toate câmpurile entității complete. Acest mecanism reduce volumul de date transferat și timpul de procesare, fiind util în special pentru afișările în listă (ex. tabele paginate sau componente de tip preview).

5.6.3. Stratul de Service

Stratul de *service* reprezintă centrul logică de business al aplicației, fiind responsabil pentru coordonarea operațiilor dintre datele persistente și logica funcționalităților oferite. Un exemplu relevant este *FoodItemService*, care include funcționalitățile de bază care se regăsesc pentru toate entitățile:

- operații CRUD (creare, citire, actualizare, stergere) asupra alimentelor;
- conversia datelor între DTO-uri și entități persistente;
- utilizarea paginării și folosirea proiecțiilor pentru optimizarea răspunsurilor;

Procesul automatizat de adăugare a unei mese, gestionat MealService. Aceasta se declanșează automat în momentul în care utilizatorul încarcă o imagine cu o masă. Imaginea este transmisă către un endpoint dedicat, care comunică direct cu microserviciul de inferență vizuală prin RestTemplate, microserviciu care analizează imaginea.

În urma inferenței, sistemul returnează o listă cu alimentele detectate, alături de estimări cantitative pentru fiecare. Pe baza acestor informații, aplicația generează automat o propunere de masă ce conține structurate în MealEntry-uri:

- denumirile alimentelor identificate;
- cantitățile estimate;
- valorile nutriționale aggregate;

Iar în entitatea Meal, tipul mesei (mic dejun, prânz, cină etc.), determinat în funcție de ora încărcării. Statisticile calculate dacă nu sunt deja salvate.

Utilizatorul poate edita această propunere înainte de salvare, beneficiind astfel de un echilibru între automatizare și control personalizat asupra datelor.

Stratul *service* integrează funcționalități suplimentare precum generarea statisticilor nutriționale zilnice și săptămânale (inclusiv cantitatea totală de calorii, macronutrienți și micronutrienți) calculate dinamic și salvate în cahce până la modificări ulterioare, gestionarea obiectivelor nutriționale ale utilizatorului prin urmărirea automată a progresului.

Partea de securitate este dezvoltată cu ajutorul framework-ului Spring Security, fiind implementată autentificarea și autorizarea utilizatorilor, gestionarea sesiunilor prin tokenuri JWT, înregistrarea conturilor noi, precum și procesul de resetare a parolei. Funcționalitatea de „forgot password” este completată de un sistem de trimitere automată a e-mailurilor cu link-uri temporare de resetare, iar cererile expirate sunt eliminate periodic printr-un *cron job* dedicat.

5.6.4. Startul de prezentare (Controller)

Controlurile REST expun rutele prin care se pot accesa și modifica informațiile. Fiecare entitate are propriul controller dedicat:

- *UserController* – pentru operații legate de utilizatori.
- *SecurityController* - pentru acțiunile de logare, înregistrare sau uitare de parola
- *FoodItemController* – pentru gestionarea alimentelor și valorilor nutriționale.
- *MealController* – pentru gestionarea meselor și calculul statisticilor.
- *GoalController* – pentru gestionarea obiectivelor nutriționale personalizate.

Controller-ele folosesc clasele de service pentru logica aplicației și apelează DTO-urile și validatorii corespunzători pentru gestionarea datelor. Acest mod de organizare respectă principiile arhitecturii MVC (Model-View-Controller) și contribuie la scalabilitatea și testabilitatea întregului sistem.

Tabela 5.2: Lista endpointurilor disponibile în aplicație

Controller	Endpointuri (I)	Endpointuri (II)
user-controller	GET /users POST /users PATCH /users	GET /users/{id} DELETE /users/{id} GET /users/personal
meal-controller	GET /meals POST /meals PATCH /meals POST /meals/request PATCH /meals/{id}/finalize	GET /meals/{id} DELETE /meals/{id} GET /meals/statistics/week GET /meals/statistics/day
goal-controller	GET /goals POST /goals PATCH /goals	GET /goals/{id} DELETE /goals/{id} GET /goals/user
food-item-controller	GET /food-items POST /food-items PATCH /food-items GET /food-items/{id}	DELETE /food-items/{id} GET /food-items/simple GET /food-items/simple/{id}
auth-controller	POST /auth/reset-password POST /auth/register	POST /auth/login POST /auth/forgot-password

Capitolul 6. Testare și validare

Strategia de testare a fost concepută pentru a asigura funcționarea corectă a aplicației atât din punct de vedere tehnic, cât și practic, în scenarii reale. Au fost testate modelele de recunoaștere vizuală pe imagini din setul de validare, iar rezultatele au fost analizate.

Pentru partea de backend - gestionarea datelor, au fost folosite teste unitare care verifică dacă aplicația se comportă într-un mod asteptat și corect în situații concrete.

Testarea manuală a fost o parte importantă în acest proces de testare, folosind imagini obișnuite, ca un utilizator real. A fost verificat pas cu pas cum se comportă aplicația: dacă imaginea este procesată corect, dacă tipul mesei este completat automat în funcție de momentul zilei și dacă datele afișate corespund cu ceea ce era în imagine. Toate aceste verificări au oferit o perspectivă exhaustivă a modului în care funcționează aplicația.

6.1. Evaluarea modelului de detecție a obiectelor de interes

În procesul de evaluare al modelului YOLOv11, s-au utilizat o serie de metrii standard pentru detecția obiectelor. Acestea oferă o imagine clară asupra preciziei și robusteștii modelului în identificarea alimentelor din imagini. Mai jos sunt explicate cele mai importante dintre ele:

- **Precizia (Precision)** – măsoară cât de multe dintre predicțiile modelului sunt corecte. Din toate obiectele pe care modelul le-a identificat, câte au fost într-adevăr corecte. O valoare mare indică un număr redus de greșeli.
- **Recall-ul** – indică cât de multe dintre obiectele reale din imagine au fost detectate de model.
- **mAP@0.5** – reprezintă media preciziei la un prag de suprapunere (IoU – Intersection over Union) de 0.5. Cu alte cuvinte, considerăm o predicție ca fiind corectă dacă bounding box-ul generat se suprapune cel puțin 50% cu cel real. Este o metrică standard în detecția obiectelor.
- **mAP@0.5:0.95** – o versiune a mAP-ului care calculează media preciziei la mai multe praguri de suprapunere, de la 0.5 la 0.95, din 0.05 în 0.05. Oferă o imagine mai completă asupra performanței modelului, penalizând predicțiile aproximative.
- **Curba F1** – reprezintă un echilibru între precizie și recall, oferind o imagine de ansamblu asupra performanței generale a modelului. Este utilă mai ales în situațiile în care avem dezechilibru între clase sau când dorim un compromis între detecțiile ratate și cele false. Valoarea F1 este maximă atunci când atât precizia, cât și recall-ul sunt ridicate.

Nota: Forma curbei F1 în timpul antrenării oferă informații esențiale despre stabilitatea modelului – o creștere constantă sugerează o învățare eficientă, iar o scădere bruscă poate semnala overfitting sau probleme de generalizare.

Evaluarea modelului YOLOv11 s-a realizat pe setul de validare, utilizând aceste metrii pentru a urmări evoluția performanței în timp. Au fost oferite și reprezentări

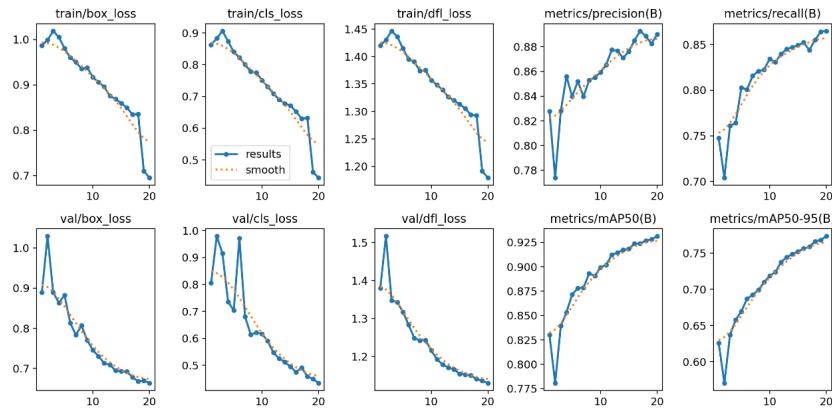


Figura 6.1: Metriki de performanță, etapa 1 (epocile 0-19)

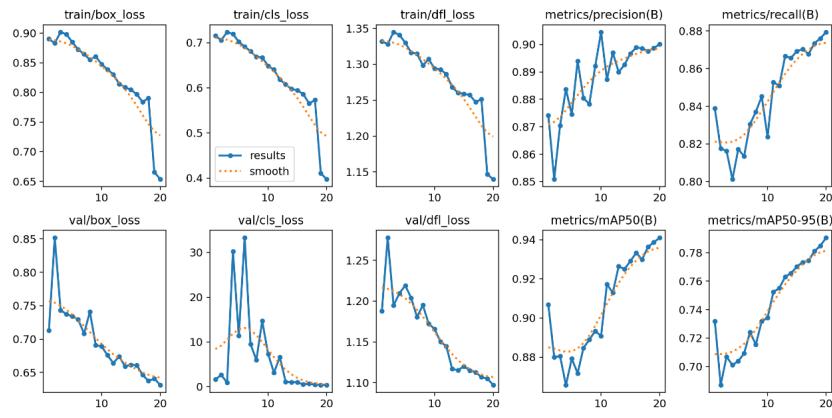


Figura 6.2: Metriki de performanță, etapa 2 (epocile 20-39)

grafice, precum curbele F1, distribuția etichetelor și matricea de confuzie, pentru o analiză vizuală detaliată a rezultatelor.

În figurile Figura 6.1 și Figura 6.2 sunt prezentate variațiile acestor metriki în funcție de epoci și etape de antrenare. Toate aceste metriki au înregistrat o creștere constantă, ceea ce indică stabilitate și progres pe parcursul antrenării:

- În etapa 1, modelul a atins o valoare mAP@0.5 de peste 92%, iar scorul F1 maxim a fost de 0.88 la un prag de încredere de 0.475.
- În etapa 2, după continuarea antrenării, mAP@0.5 a crescut la peste 94%, iar F1 a atins 0.89 la un prag optim de 0.454, ceea ce confirmă o îmbunătățire a preciziei și stabilității detecției.

Matricile de confuzie evidențiază performanța de clasificare pentru clasa "roi":

- În prima respectiv a doua etapă modelul a detectat corect 9.415 respectiv 9.478 regiuni "roi" din totalul instantelor reale.
- 2.080 în prima etapă respectiv 1.797 regiuni au fost fals detectate ca "roi" (false positive).
- 878 regiuni respectiv 815 "roi" reale au fost ratate (false negative).

Curba F1 în funcție de confidence threshold Curba F1 (Figura 6.3) evidențiază echilibrul dintre precizie și recall în funcție de pragul de încredere (confidence threshold):

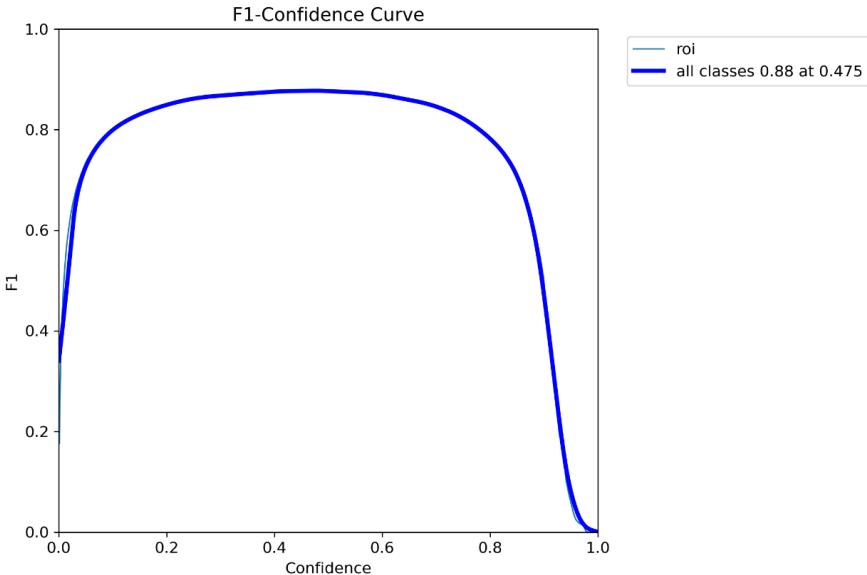


Figura 6.3: Curba F1 în funcție de confidence threshold

- Scorul maxim F1 în prima etapă a fost de 0.88 la un prag de încredere de aproximativ 0.475.
- Scorul maxim F1 în a doua etapă a fost de 0.89 la un prag de încredere de aproximativ 0.454.

6.2. Evaluarea modelului de clasificare a obiectelor

După finalizarea etapelor de antrenare și fine-tuning, performanța modelului ResNet-50 a fost evaluată prin intermediul unui set de teste, folosind un set de date de validare compus din 9.661 de imagini. Pentru interpretarea performanței, au fost prezentate matrice de confuzie și diagrame F1-score pentru ambele variante ale modelului: cel antrenat inițial și cel ajustat prin fine-tuning.

6.2.1. Analiza matricilor de confuzie

În Figura 6.4 și Figura 6.5 sunt prezentate matricile de confuzie aferente celor două versiuni ale modelului. Acestea evidențiază modul în care sunt clasificate corect sau greșit imaginile pentru fiecare dintre cele 30 de clase.

În urma fine-tuning-ului, se poate observa o scădere a numărului de clasificări greșite pentru clase precum *ginger*, *garlic* sau *zucchini*, unde anterior au fost identificate confuzii frecvente. De asemenea, s-a menținut o precizie ridicată pentru clase cu suport mare, precum *banana*, *dragon_fruit* sau *strawberry*.

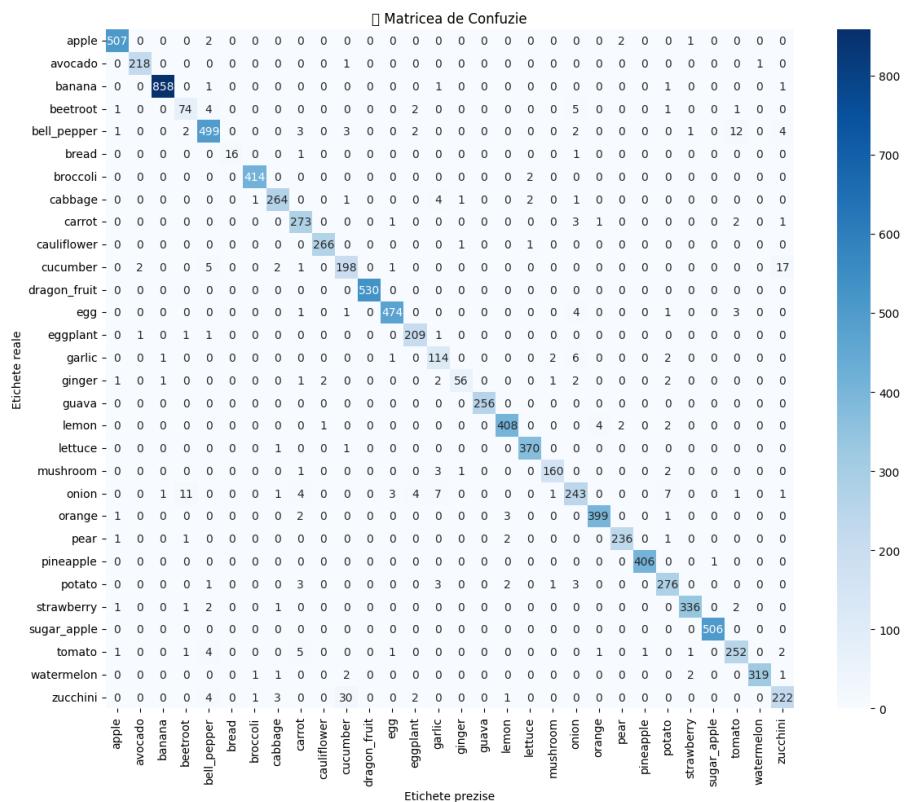


Figura 6.4: Matricea de confuzie pentru modelul ResNet-50 antrenat initial

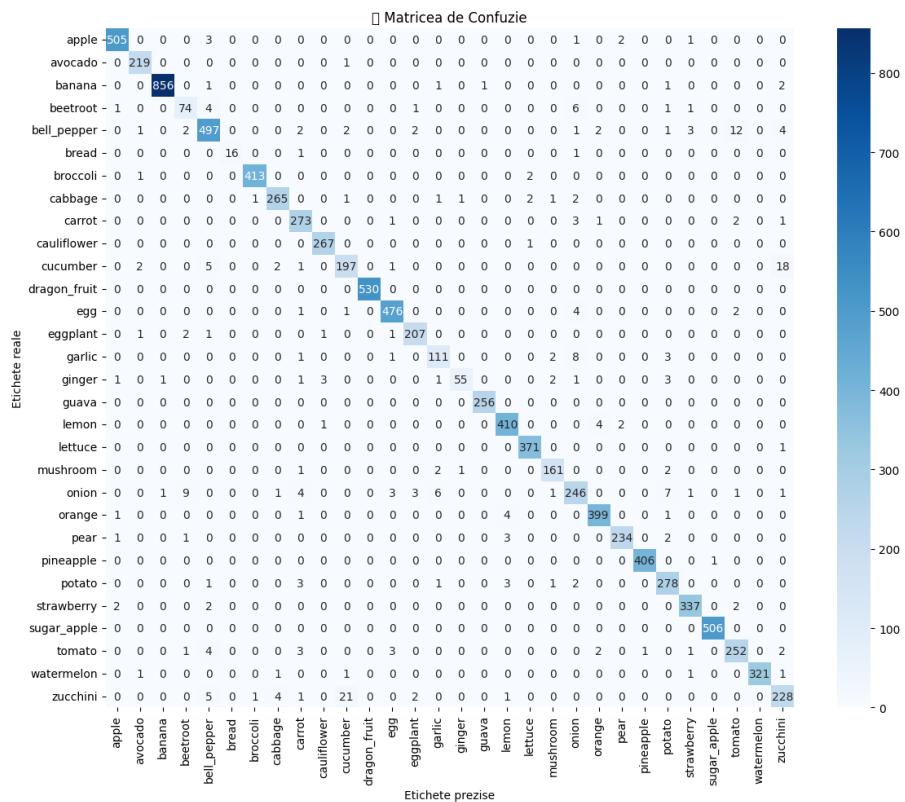


Figura 6.5: Matricea de confuzie pentru modelul ResNet-50 după fine-tuning

6.2.2. Compararea F1-score per clasă

Pentru a evidenția diferențele de performanță la nivel de clasă, în Figura 6.6 și Figura 6.7 sunt prezentate barele de F1-score pentru fiecare dintre cele 30 de clase.

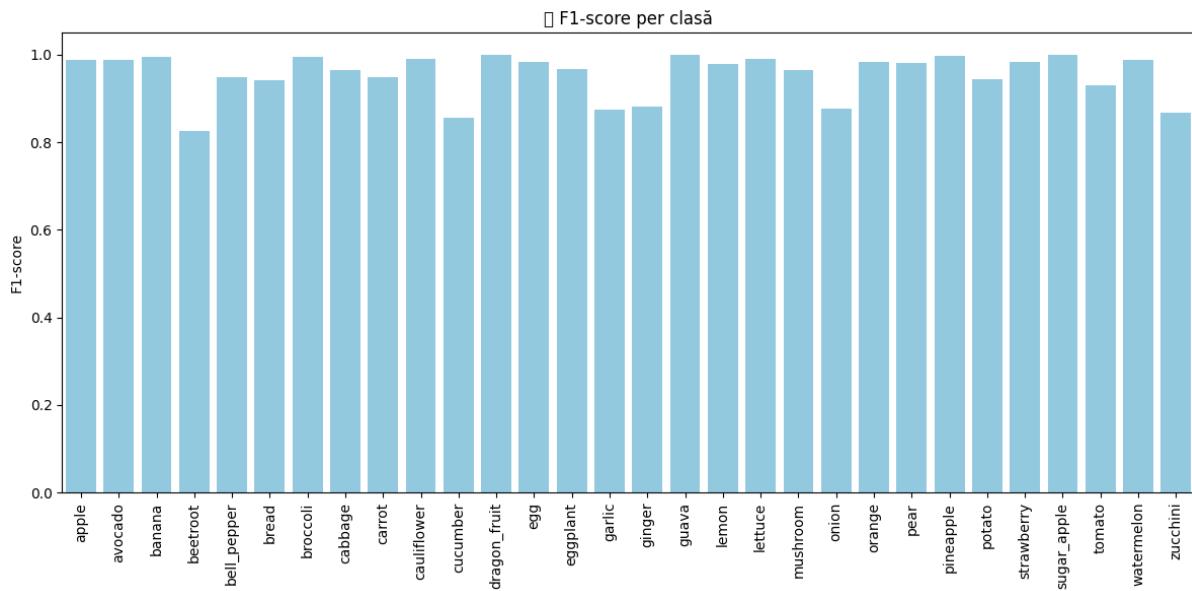


Figura 6.6: F1-score pe clasă pentru modelul ResNet-50 antrenat inițial

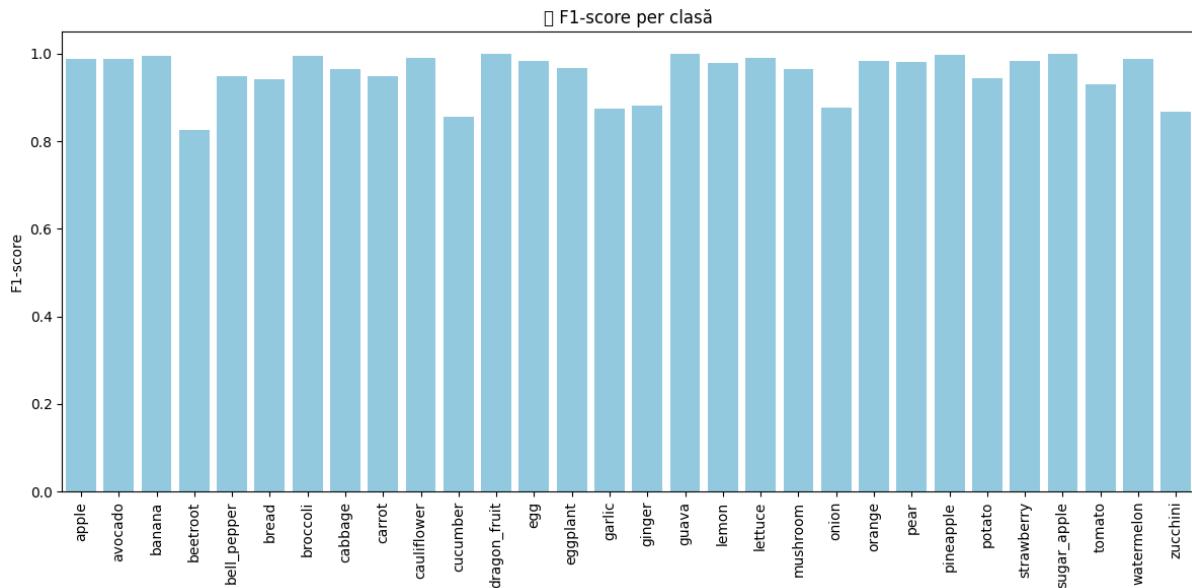


Figura 6.7: F1-score pe clasă pentru modelul ResNet-50 după fine-tuning

În mod global, se remarcă o creștere ușoară dar semnificativă a valorilor F1-score pentru clasele problematice. Clasa *bread*, de exemplu, care avea un suport redus și era predispusă la overfitting, a înregistrat o valoare F1 de 0.94. Alte clase, precum *ginger* și *onion*, au avut îmbunătățiri clare în acuratețea generală și în echilibrul între precizie și recall.

6.2.3. Rezumatul performanțelor

În Tabelul 6.1 sunt sintetizate valorile de *precision*, *recall*, *f1-score* și *support* pentru fiecare clasă, conform evaluării finale a modelului ResNet-50 fine-tuned.

Clasă	Precizie	Recall	F1-score	Support
apple	0.988	0.986	0.987	512
avocado	0.973	0.995	0.984	220
banana	0.998	0.993	0.995	862
beetroot	0.831	0.841	0.836	88
bell_pepper	0.950	0.940	0.945	529
bread	1.000	0.889	0.941	18
broccoli	0.995	0.993	0.994	416
cabbage	0.971	0.967	0.969	274
carrot	0.932	0.972	0.951	281
cauliflower	0.982	0.996	0.989	268
cucumber	0.879	0.872	0.876	226
dragon_fruit	1.000	1.000	1.000	530
egg	0.979	0.983	0.981	484
eggplant	0.963	0.972	0.967	213
garlic	0.902	0.881	0.892	126
ginger	0.965	0.809	0.880	68
guava	0.996	1.000	0.998	256
lemon	0.974	0.983	0.979	417
lettuce	0.987	0.997	0.992	372
mushroom	0.958	0.964	0.961	167
onion	0.895	0.866	0.880	284
orange	0.978	0.983	0.980	406
pear	0.983	0.971	0.977	241
pineapple	0.998	0.998	0.998	407
potato	0.930	0.962	0.946	289
strawberry	0.977	0.983	0.980	343
sugar_apple	0.998	1.000	0.999	506
tomato	0.930	0.937	0.933	269
watermelon	1.000	0.985	0.992	326
zucchini	0.884	0.867	0.875	263
Acuratețe	0.969			
Media	0.960	0.953	0.956	9661
Media ponderată	0.969	0.969	0.969	9661

Tabela 6.1: Rezultatele finale ale clasificării cu ResNet-50 fine-tuned

6.3. Validarea practică a pipeline-ului de analiză vizuală

Pentru a valida funcționalitatea practică a sistemului de analiză vizuală, au fost realizate teste pe imagini reale, care simulează scenarii uzuale din viața cotidiană, cum ar fi mesele consumate acasă. Procesul de inferență este împărțit în două etape distințe, corespunzătoare celor două modele utilizate: modelul de detecție a obiectelor (YOLO) și modelul de clasificare (ResNet-50).

În cadrul testelor efectuate pe un **laptop fără placă video dedicată**, timpul mediu de inferență pentru o imagine completă este de **aproximativ 2 secunde**. Această durată include atât detecția obiectelor, cât și clasificarea fiecărui obiect identificat, iar limitarea se datorează lipsei unui GPU, ceea ce impune procesarea exclusiv pe CPU.

Pentru a evalua impactul resurselor hardware asupra performanței, au fost realizate teste suplimentare pe un alt **laptop echipat cu o placă video dedicată**, dar care nu este performantă. În acest caz, timpul de inferență a fost semnificativ redus, ajungând la **sub 1 secundă** pentru imagini similare, ceea ce confirmă că viteza de procesare poate fi îmbunătățită considerabil prin utilizarea unui accelerator hardware.

Acste rezultate demonstrează că, deși sistemul este funcțional și pe echipamente modeste, performanța inferenței poate fi scalată în funcție de infrastructura disponibilă, ceea ce îl face potrivit atât pentru utilizatori individuali, cât și pentru integrarea în soluții cloud cu suport GPU.

Detectia obiectelor de interes. Imaginele brute, fără preprocesări artificiale, au fost introduse în modelul de detectie a obiectelor de interes (YOLOv11). Aceasta a identificat automat regiunile relevante din imagine (bounding box-uri) care încadrează obiectele vizibile, precum fructe sau legume. Pentru fiecare imagine procesată:

- Regiunile identificate au fost extrase automat;
- Fiecare regiune a fost returnată sub forma unei imagini decupate, utilizată ulterior pentru clasificare.

Această etapă confirmă capacitatea modelului de detectie de a localiza obiectele într-un cadru real, diferit de condițiile ideale ale setului de antrenament, după cum este reprezentat și în Figura 6.8.

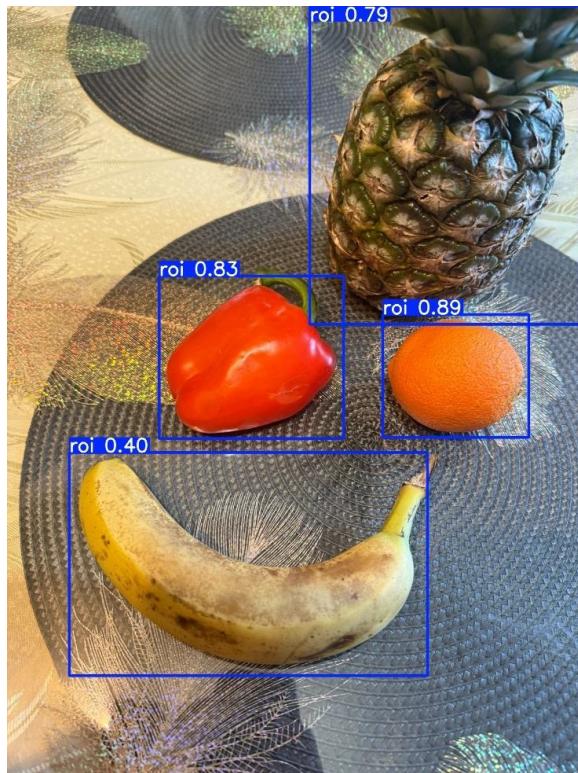


Figura 6.8: Identificarea obiectelor de interes ale unei imagini reale folosind modelul de detectie a obiectelor

Curl

```
curl -X 'POST' \
'http://127.0.0.1:8000/predict/' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiJ9eyJ' \
-H 'Content-Type: multipart/form-data' \
-F 'file=@14.jpg;type=image/jpeg'
```

Request URL

<http://127.0.0.1:8000/predict/>

Server response

Code	Details
200	Response body <pre>[{ "tag": 21, "count": 1 }, { "tag": 4, "count": 1 }, { "tag": 23, "count": 1 }, { "tag": 2, "count": 1 }]</pre> Response headers <pre>content-length: 83 content-type: application/json date: Tue, 08 Jul 2025 12:56:24 GMT server: uvicorn</pre>

Figura 6.9: Clasificarea obiectelor de interes ale unei imagini reale folosind modelul de clasificare a obiectelor

Clasificarea obiectelor. Regiunile decupate din imaginile originale, generate în urma inferenței modelului de detectie, au fost utilizate ca input pentru modelul de clasificare a obiectelor (ResNet-50). Acest model a atribuit fiecărei regiuni o etichetă corespunzătoare uneia dintre cele 30 de clase predefinite de alimente, reprezentând diverse fructe și legume.

Clasificarea a fost aplicată asupra obiectelor extrase din imaginea prezentată în Figura 6.8, fiecare regiune fiind analizată separat, iar pentru fiecare obiect segmentat,

modelul a returnat automat clasa prezisă care a fost ulterior comparată vizual cu imaginea reală, Figura 6.9:

- Eticheta 21: *orange*
- Eticheta 4: *bell_pepper*
- Eticheta 23: *pineapple*
- Eticheta 2: *banana*

Distribuția completă a claselor poate fi consultată în Tabelul 5.1, unde sunt prezentate toate cele 30 de categorii utilizate în procesul de clasificare.

Rezultatele obținute pentru un set suplimentar de imagini reale sunt prezentate în Figura 6.10, unde se poate observa atât segmentarea obiectelor de interes (rândul de sus), cât și clasificarea acestora (rândul de jos). Tag reprezintă eticheta cu sub care a fost clasificat obiectul, iar count numărul de obiecte găsite.



Figura 6.10: Alte teste pentru segmentare (sus) și rezultatele clasificării (jos)

Pipeline-ul propus funcționează eficient, dar pot apărea erori de identificare a obiectelor de interes sau de clasificare în cazul unor obiecte vizual foarte asemănătoare (de exemplu, între anumite fructe cu formă și culoare apropiată), sau în cazul în care imaginile sunt de o calitate slabă, unde sunt deteriorate sau în condiții de lumină neprielnice.

6.4. Testarea componentei de gestionare a datelor

Componenta de gestionare a datelor a fost testată printr-un set de teste unitare, concentrate pe logica de business implementată în clasele de tip *service*. Scopul acestor teste a fost verificarea corectitudinii funcționalităților implementate în logica de business.

Testele au fost dezvoltate cu ajutorul framework-ului *JUnit 5*, utilizând *Mockito* pentru a crea obiecte mock ale componentelor externe, în special repository-urile. Această abordare a permis izolarea logicii din serviciile testate. Structura testelor a urmat modelul *Given-When-Then*, facilitând claritatea cazurilor de test și trasabilitatea acestora.

Fiecare clasă de service a fost testată individual, acoperind atât scenariile pozitive (cazuri corecte de utilizare), cât și pe cele negative. Pentru metodele care implică transformări sau calcule (ex. calculul statisticilor nutriționale sau generarea automată a meselor), au fost verificate explicit rezultatele returnate.

Funcționalități testate

- **Security** – autentificare, înregistrare, resetare a parolei cu generare de token și verificarea validității acestuia;
- **User** – creare cont, editare date personale, ștergere și vizualizare detaliu utilizator;
- **Meal** – adăugare masă, modificare, ștergere și generare automată a unei mese pe baza unei imagini mock, vizualizare masă;
- **FoodItem** – operații CRUD asupra alimentelor, validarea câmpurilor obligatorii;
- **Goal** – setarea, modificarea și ștergerea obiectivelor nutriționale;
- **Statistics** – agregarea corectă a valorilor nutriționale zilnice/săptămânale și compararea acestora cu obiectivele;
- **Error handling** – tratarea exceptiilor precum și cazuri cu input lipsă sau invalid.

Prin rularea constantă a acestor teste în timpul dezvoltării, se menține un nivel ridicat al calității codului și se facilitează introducerea de noi funcționalități fără afectarea comportamentului existent.

Capitolul 7. Manual de instalare și utilizare

7.1. Instalare

Cerințe hardware

- Procesor: minim Intel i3 sau echivalent;
- Memorie RAM: 8 GB (ideal 16 GB pentru o experiență fluidă);
- Spațiu pe disc: cel puțin 3 GB liber pentru containere și date;
- Placă video cu suport CUDA (optional, pentru rulare locală performantă).

Cerințe software

- Sistem de operare: Windows 10+, Linux sau macOS;
- Docker, Docker Compose, Java21 instalate;
- Browser modern.

7.1.1. Structura sistemului și containerizare

Pentru a asigura o instalare rapidă și fără conflicte între dependințe, întreaga aplicație a fost containerizată folosind Docker. Fiecare componentă rulează într-un container izolat, cu propriile resurse și configurări. Structura generală este următoarea:

- PostgreSQL (v15) – baza de date relațională pentru stocarea alimentelor, utilizatorilor, meselor etc.;
- nutrition-app – componeta backend realizat în Spring Boot cu Java 21;
- inference-app – componenta de analiză vizuală scris în FastAPI (Python 3.10) are urmatoarele depenințe: fastapi, uvicorn, opencv-python-headless, torch, torchvision, ultralytics, python-jose, python-multipart, pillow;
- frontend – interfață React compilată prin NGINX;
- Mailhog – serviciu de testare a emailurilor (ex: resetare parolă).

Continutul aplicației

Arhiva *nutrition-app.zip* include:

- codul sursă pentru toate componente;
- fișierul docker-compose.yml;
- fișierul .env pentru configurații;
- scriptul SQL de inițializare a bazei de date.

7.1.2. Instalare și rulare aplicație

Înainte de a porni aplicația cu Docker Compose, este necesară generarea fișierul *.jar* pentru serviciul Java din folderul aplicației nutrition-app, dacă nu este deja generat.

```
./mvnw clean package -DskipTests
```

După generarea fișierului *.jar*, rulați comanda următoare în folderul rădăcină:

```
docker-compose up --build
```

Accesează aplicația

- Frontend (UI): <http://localhost:5173>
- Backend API: <http://localhost:8080>
- Serviciu inferență: <http://localhost:8000>
- Mailhog (test email): <http://localhost:8025>

7.2. Ghid de utilizare

7.2.1. Autentificare și înregistrare

La accesarea aplicației, utilizatorul este întâmpinat de o interfață de autentificare.

Sunt disponibile opțiunile:

- Înregistrare: completarea unui formular cu email, parolă și username.
- Autentificare: introducerea datelor deja înregistrate.
- Uitare Parola: introducerea emailui, accesarea link-ului din mail și introducerea noii parole.

7.2.2. Adăugarea unei mese pe baza unei imagini

1. Utilizatorul accesează pagina dedicată salvării imaginii dintr-o imagine.
2. Încarcă o imagine cu alimentele consumate.
3. Backend-ul de detecție identifică obiectele din imagine și estimează obiectele, cantitățile.
4. În funcție de ora, aplicația sugerează automat tipul mesei.
5. Utilizatorul poate revizui sau edita alimentele înainte de salvare.

7.2.3. Vizualizarea datelor legate de alimente

1. Utilizatorul poate vedea toate informațiile legate de alimente.
2. Utilizatorul accesează pagina dedicată vizualizării alimentelor.
3. Utilizatorul poate vedea informațiile detaliate pentru fiecare aliment, accesând butonul view.
4. Administratorul poate gestiona alimentele.

7.2.4. Vizualizarea statisticilor nutriționale

- Utilizatorul accesează secțiunea dedicată statisticilor.
- Tabel cu date zilnice sau săptămânale.
- Comparație între aportul realizat și obiectivele personale.

7.2.5. Setarea obiectivelor personale

1. Utilizatorul accesează secțiunea dedicată obiectivelor.
2. Introduce țintele zilnice pentru calorii și nutrienți.
3. Aplicația utilizează aceste valori pentru a compara.

7.2.6. Administratorul gestionează userii

1. Administratorul accesează secțiunea dedicată utilizatorilor.
2. Lista utilizatorilor apare structurată într-un tabel.
3. Administratorul poate să vizualizeze, editeze, steargă sau crea un nou utilizator.

Capitolul 8. Concluzii

Lucrarea de față a avut ca obiectiv dezvoltarea unei aplicații inteligente care să permită recunoașterea automată a alimentelor din imagini și estimarea valorilor nutriționale corespunzătoare. Aplicația combină metode moderne de viziune computerizată cu o arhitectură modulară, de tip client-server, oferind o soluție completă și extensibilă pentru urmărirea alimentației în viața de zi cu zi.

Contribuții personale

În cadrul acestei lucrări de licență, am realizat procesele de proiectare, dezvoltare și testare ale aplicației, contribuind semnificativ la fiecare componentă a sistemului. Contribuțiiile personale pot fi detaliate astfel:

- Am compus setul de date pentru detectia obiectelor, prin selectarea și uniformizarea unor surse publice, eliminarea imaginilor irelevante sau neetichetate și reconfigurarea datasetului pentru a include o singură clasă generală (clasa 0), destinată obiectelor alimentare;
- Am generat un al doilea set de date, derivat din primul, prin decuparea automată a obiectelor de interes (bounding boxes) și încadrarea lor în clase alimentare specifice pentru antrenarea modelului de clasificare;
- Am antrenat modelul de detectie vizuală YOLOv11 pe primul set de date personalizat, obținând un model capabil să localizeze fructele și legumele în imagini reale;
- Am antrenat modelul de clasificare ResNet-50 pe al doilea set de date provenit din imaginile decupate, optimizând performanța prin fine-tuning pentru cele 30 de clase alimentare;
- Am realizat integrarea celor două modele într-un pipeline complet de procesare a imaginilor, care detectează automat obiectele alimentare și le clasifică pentru a extrage informațiile nutriționale relevante;
- Am construit o bază de date nutrițională, completată cu informații detaliate despre fructe și legume (macronutrienți, vitamine, minerale), inserate printr-un script SQL personalizat pentru consistență și acuratețe;
- Am dezvoltat aplicația back-end folosind Spring Boot, implementând un sistem monolic care cuprinde toate funcționalitățile de gestionare a datelor, statistici nutriționale, obiective zilnice și creare automată de mese pe baza imaginilor, precum și integrarea cu microserviciul de inferență vizuală;
- Am realizat interfața front-end în React, concepută într-un mod modern și intuitiv, care integrează toate funcționalitățile implementate în backend: autentificare și gestionare a contului, administrarea alimentelor și meselor, setarea și urmărirea obiectivelor nutriționale, afișarea statisticilor zilnice și săptămânale, precum și încărcarea imaginilor pentru procesare automată prin pipeline-ul de detectie și clasificare.
- Am testat aplicația atât prin teste unitare, cât și prin testare vizuală, verificând rezultatele cu imagini reale de fructe și legume pentru a evalua corectitudinea

detectiei, clasificarii si calculului valorilor nutriționale.

Analiza rezultatelor

Rezultatele obținute sunt încurajatoare. Sistemul a atins o precizie de detectie (94% mAP) și clasificare (97% acuratețe) pe setul de validare, cu un timp de răspuns bun pentru procesarea imaginilor de aproximativ o secundă. Detectia obiectelor a fost stabilă în condiții optime de iluminare, iar aplicația a fost capabilă să genereze statistici zilnice și săptămânale privind aportul de calorii, macronutrienți și micronutrienți. Totuși, în anumite situații, cum ar fi imaginile cu fundal aglomerat sau alimente vizual similare, au apărut erori de detectie sau clasificare, ceea ce indică nevoia de diversificare a datelor de antrenament.

Dezvoltări ulterioare

Pentru continuarea și îmbunătățirea acestei aplicații, se pot lua în considerare mai multe direcții de dezvoltare, cu scopul de a extinde funcționalitatea și să crească utilitatea practică a sistemului:

- Extinderea bazei de date nutriționale: Adăugarea unui număr mai mare de alimente, inclusiv preparate combinate sau produse procesate, ar permite utilizatorilor să obțină informații nutriționale mai complete și adaptate consumului real.
- Îmbunătățirea componentei obiective: Funcționalitatea actuală ar putea fi dezvoltată prin introducerea unor recomandări inteligente, generate automat pe baza istoricului nutrițional al utilizatorului. De asemenea, sistemul ar putea sugera ajustări ale obiectivelor zilnice în funcție de progresul utilizatorului, nevoile calorice estimate sau nivelul de activitate fizică.
- Recomandări alimentare personalizate: Un modul dedicat ar putea oferi sugestii de mese și alimente în funcție de preferințele și obiectivele declarate ale utilizatorului (de exemplu: menținere, slabire, creștere în greutate, dietă vegetariană etc.). Acest sistem ar putea folosi reguli nutriționale, dar și algoritmi de învățare automată pentru a îmbunătăți precizia recomandărilor.
- Confirmarea manuală a rezultatelor: În anumite cazuri, utilizatorii ar putea avea posibilitatea să valideze sau să corecteze manual etichetele generate de modelul de clasificare. Acest tip de feedback ar contribui la perfecționarea modelelor prin învățare continuă și ar permite adaptarea sistemului la variații specifice ale imaginilor din lumea reală.
- Integrarea cu dispozitive inteligente: Conectarea aplicației la senzori purtabili (wearables), cum ar fi brățările de fitness sau ceasurile inteligente, ar permite corelația alimentației cu nivelul de activitate fizică și alți parametri fiziologici. Astfel, aplicația ar deveni o unealtă completă pentru monitorizarea stilului de viață, nu doar a nutriției.

În concluzie, această lucrare evidențiază potențialul real al integrării tehnologijilor de viziune computerizată cu sisteme software moderne în dezvoltarea aplicațiilor dedicate nutriției. Chiar dacă mai există aspecte care pot fi optimizate, soluția implementată constituie un punct de plecare solid pentru explorări viitoare în direcția digitalizării sănătății cu sprijinul inteligenței artificiale.

Bibliografie

- [1] "Nutrition apps - worldwide," 2024. [Online]. Available: <https://www.statista.com/outlook/hmo/digital-health/digital-fitness-well-being/health-wellness-coaching/nutrition-apps/worldwide>
- [2] S. M. S. v. d. H. Ireen Raaijmakers, Muriel C D Verain, "Jmir publications," Jun 2023. [Online]. Available: <https://mhealth.jmir.org/2023/1/e39515>
- [3] V. Kaushik, "The magic of myfitnesspal: How one app can help you achieve your fitness dreams," April 2024. [Online]. Available: <https://www.techaheadcorp.com/blog/the-magic-of-myfitnesspal-how-one-app-can-help-you-achieve-your-fitness-dreams/>
- [4] "System architecture of myfitnesspal," may 2025. [Online]. Available: <https://prezi.com/p/vhxyxxuj8s1k/system-architecture-of-myfitnesspal/>
- [5] V. M. M. H. Mohd Sarim, Garima Singh, "Foodvisor:afood calorie estimation system," *2024 Second International Conference on Advances in Information Technology (ICAIT-2024)*, 2024.
- [6] I. A. Nutrition, "Alma launches first ai-powered nutrition companion app to make healthy eating simple, personalized, and effortless," February 2025. [Online]. Available: <https://www.prnewswire.com/news-releases/alma-launches-first-ai-powered-nutrition-companion-app-to-make-healthy-eating-simple-personalized-301651111.html>
- [7] I. Mehta, "Former whoop exec's new app alma uses ai for all things nutrition," February 2025. [Online]. Available: <https://techcrunch.com/2025/02/05/former-whoop-execs-new-app-alma-uses-ai-for-all-things-nutrition/>
- [8] P. Lee, "The growth machine: How noom runs 365 landing page experiments per year," August 2022. [Online]. Available: <https://medium.com/noom-engineering/the-growth-machine-how-noom-runs-365-landing-page-experiments-per-year-1e098ea33354>
- [9] "How to develop a fitness app like noom," may 2025. [Online]. Available: <https://www.gmtasoftware.com/blog/develop-a-fitness-app-like-noom/>
- [10] T. Mucci, "What is image recognition?" Noiembrie 2024. [Online]. Available: <https://www.ibm.com/think/topics/image-recognition#:~:text=Image%20recognition%20is%20an%20application,in%20digital%20images%20or%20video>.
- [11] "What is machine learning?" September 2025. [Online]. Available: <https://www.ibm.com/think/topics/machine-learning>
- [12] J. H. și Mark Scapicchio, "What is deep learning?" Iunie 2024. [Online]. Available: <https://www.ibm.com/think/topics/deep-learning>

- [13] "What are convolutional neural networks?" Aprilie 2025. [Online]. Available: <https://www.ibm.com/think/topics/convolutional-neural-networks>
- [14] "Introduction to convolution neural network," Aprilie 2025. [Online]. Available: <https://www.geeksforgeeks.org/introduction-convolution-neural-network/>
- [15] "Activation functions in neural networks," Aprilie 2025. [Online]. Available: <https://www.geeksforgeeks.org/activation-functions-neural-networks/>
- [16] "Introduction to pooling layer," aprilie 2025. [Online]. Available: <https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/>
- [17] M. L. in Plain English, "Convolutional neural network — lesson 8: Fully connected layers and flattening," iunie 2023. [Online]. Available: [ConvolutionalNeuralNetworkâ€ťLesson8:FullyConnectedLayersandFlattening](#)
- [18] "What is fully connected layer in deep learning?" Mai 2024. [Online]. Available: <https://www.geeksforgeeks.org/what-is-fully-connected-layer-in-deep-learning/>
- [19] "Softmax activation function in neural networks," Noiembrie 19. [Online]. Available: <https://www.geeksforgeeks.org/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/>
- [20] "Object detection datasets overview," may 2025. [Online]. Available: <https://docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format>
- [21] "Resnet and resnetv2," may 2025. [Online]. Available: <https://keras.io/api/applications/resnet/>
- [22] "Preprocess images," may 2025. [Online]. Available: <https://docs.roboflow.com/datasets/image-preprocessing>
- [23] "Create augmented images," may 2025. [Online]. Available: <https://docs.roboflow.com/datasets/image-augmentation>
- [24] R. or Deepfake Face Detection in Images and V. D. using YOLO11 Algorithm. (2025) The optimized module structure of yolo11 network. Accessed: 3 Jul 2025. [Online]. Available: https://www.researchgate.net/figure/The-Optimized-Module-Structure-of-YOLO11-Network-11_fig2_389027619
- [25] [Online]. Available: <https://learnopencv.com/wp-content/uploads/2024/10/yolo11-architecture.png>
- [26] "Residual networks (resnet) – deep learning," may 2025. [Online]. Available: <https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/>
- [27] "Resnet (34, 50, 101): Residual cnns for image classification tasks," may 2025. [Online]. Available: <https://neurohive.io/wp-content/uploads/2019/01/resnet-e1548261477164.png>
- [28] S. Bangar, "Resnet architecture explained," july 2022. [Online]. Available: <https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d>

- [29] Geeks for Geeks, “Client-server model,” <https://www.geeksforgeeks.org/client-server-model/>, Apr. 2025, accesat la 28 aprilie 2025.
- [30] “Ultralytics yolo11,” may 2025. [Online]. Available: <https://docs.ultralytics.com/models/yolo11/>
- [31] O. Aliyev, “Fruits and vegetables dataset (roboflow universe),” 2024. [Online]. Available: <https://universe.roboflow.com/orkhan-aliyev-8nktf/fruits-and-vegetables-2vf7u>
- [32] “Fruits and vegetables dataset (datasets-wqqwv),” 2024. [Online]. Available: <https://universe.roboflow.com/datasets-wqqwv/fruits-and-vegetables-un1ow>
- [33] ishop, “Fruits and vegetables dataset (ishop),” 2024. [Online]. Available: <https://universe.roboflow.com/ishop/fruits-and-vegetables-im3lj>
- [34] roboscale, “Vegetables and fruits dataset,” 2024. [Online]. Available: <https://universe.roboflow.com/roboscale/vegetables-and-fruits-egjaa>
- [35] fruitsandvegetables 4atqi, “Fruits and vegetables dataset (hoye8),” 2024. [Online]. Available: <https://universe.roboflow.com/fruitsandvegetables-4atqi/fruits-and-vegetables-hoye8>
- [36] project 6000-agriculture, “Fruits and vegetables dataset (agriculture project),” 2024. [Online]. Available: <https://universe.roboflow.com/project-6000-agriculture/fruits-and-vegetables-17y0t>
- [37] ece4078 mpxui, “Lemon dataset,” 2024. [Online]. Available: <https://universe.roboflow.com/ece4078-mpxui/lemon-jlkzg>
- [38] S. Jayasinghe, “Lettuce harvest prediction dataset,” 2024. [Online]. Available: <https://universe.roboflow.com/samandini-jayasinghe/lettuce-harvest-prediction>
- [39] “vegetables computer vision project,” 2024. [Online]. Available: <https://universe.roboflow.com/robouserflow/vegetables-cg8tb>
- [40] “Residual networks (resnet) – deep learning,” may 2025. [Online]. Available: <https://media.geeksforgeeks.org/wp-content/uploads/20200424011138/ResNet.PNG>

Anexa A. Secțiuni relevante din cod

```
/** ANTRENAREA MODELULUI YOLO PENTRU DETECȚIA OBIECTELOR DE INTERES**/

!yolo task=detect mode=train
model=/kaggle/input/run_roi_2/pytorch/default/1/last.pt \
data=/kaggle/input/fruitsandvegetablesroi/data.yaml epochs=20 \
imgsz=640 batch=22 workers=8 device=0,1 optimizer=AdamW \
lr0=0.002 lrf=0.01 momentum=0.937 weight_decay=0.0004 \
warmup_epochs=3 warmup_momentum=0.8 warmup_bias_lr=0.1 \
cache=False dropout=0.05 \
hsv_h=0.015 hsv_s=0.4 hsv_v=0.3 \
degrees=2 translate=0.05 scale=0.2 shear=1.0 \
flipud=0.05 fliplr=0.2 mosaic=0.3 mixup=0.05 copy_paste=0.05 \
close_mosaic=2 overlap_mask=True mask_ratio=4 \
patience=10 save=True plots=True val=True


/** PIPELINE-UL DE ANALIZĂ VIZUALĂ **/

@app.post("/predict/")
async def predict(
    file: UploadFile = File(...),
    payload: dict = Depends(verify_jwt)
):
    temp_file = f"temp_{file.filename}"
    try:
        with open(temp_file, "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)

        cropped_images = detect_and_crop(temp_file, yolo_model)

        labels = [
            classify_with_resnet(img, resnet_model, device, class_names)
            for img in cropped_images
        ]
        counts = Counter(labels)

        return
        [{"tag": int(tag), "count": count} for tag, count in counts.items()]

    except Exception:
        raise HTTPException(status_code=500, detail="Error during inference")
```

```

finally:
    if os.path.exists(temp_file):
        os.remove(temp_file)

/* IMPLEMENTAREA METOLDELOR FOLOSITE ÎN PROCESUL DE ANALIZĂ VIZUALĂ*/

def resize_with_padding(image, target_size=224):
    w, h = image.size
    scale = target_size / max(w, h)
    new_w, new_h = int(w * scale), int(h * scale)
    image = tf.resize(image, (new_h, new_w))

    # Calculate padding
    pad_left = (target_size - new_w) // 2
    pad_top = (target_size - new_h) // 2
    pad_right = target_size - new_w - pad_left
    pad_bottom = target_size - new_h - pad_top

    # Add black padding (0)
    image = tf.pad(image, (pad_left, pad_top, pad_right, pad_bottom), fill=0)
    return image

resnet_transform = transforms.Compose([
    transforms.Lambda(resize_with_padding),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])

def detect_and_crop(image_path, model_yolo):
    image = Image.open(image_path).convert("RGB")
    results = model_yolo(image_path, imgsz=640)
    cropped_images = []

    for result in results:
        for bbox in result.boxes.xywh:
            x_c, y_c, w, h = bbox
            x_min = int((x_c - w / 2))
            y_min = int((y_c - h / 2))
            x_max = int((x_c + w / 2))
            y_max = int((y_c + h / 2))
            cropped_img = image.crop((x_min, y_min, x_max, y_max))
            cropped_images.append(cropped_img)

    return cropped_images if cropped_images else [image]

```

```
def classify_with_resnet(image, model_resnet, device, class_names):
    image = resnet_transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        output = model_resnet(image)
        predicted_class = torch.argmax(output, dim=1).item()
    return class_names[predicted_class]

/** CREAREA UNEI MESE APELAND SERVICIUL DE ANALIZĂ VIZUALĂ **/

@Transactional
public Optional<CreateMealRequest> createMealDraft(final MultipartFile image) {
    List<DetectedObject> detectedObjects = inferenceClient.predict(image);

    if (detectedObjects == null || detectedObjects.isEmpty()) {
        return Optional.empty();
    }

    Meal meal = initializeMealDraft();

    List<MealEntry> entries = detectedObjects.stream()
        .map(detectedObject -> {
            FoodItem foodItem = findFoodItemByTagOrThrow(
                detectedObject.getTag()
            );
            return buildMealEntry(
                meal, foodItem, detectedObject.getCount(
            );
        })
        .toList();

    meal.setEntries(entries);

    return Optional.of(mapMealToCreateMealRequest(meal));
}
```

Anexa B. Alte informații relevante

```
/** CREAREA SETULUI DE DATE DESTINAT CLASIFICĂRII **/


# Path to trained YOLO model
yolo_model_path = "E:/anul 4/LICENTA INVATARE/runs/detect/train2/weights/best.pt"
yolo_model = YOLO(yolo_model_path)
yolo_model.eval()


# Path to save cropped images
output_dir = "E:/anul 4/LICENTA_NISTOR_IOAN_GABRIEL/datasets/yolo_crops/"


# Transform for saved images
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])


image_dir = "E:/anul 4/licenta/dataset/zucchini/"
images = [f for f in os.listdir(image_dir) if f.endswith('.jpg')]

for img_name in images:
    image_path = os.path.join(image_dir, img_name)
    image = Image.open(image_path).convert("RGB")
    width, height = image.size

    results = yolo_model(image_path)

    for result in results:
        for bbox, cls in zip(result.boxes.xywh, result.boxes.cls):
            x_c, y_c, w, h = bbox
            class_id = int(cls.item())

            x_min = int((x_c - w / 2))
            y_min = int((y_c - h / 2))
            x_max = int((x_c + w / 2))
            y_max = int((y_c + h / 2))

            cropped_img = image.crop((x_min, y_min, x_max, y_max))


            class_folder = os.path.join(output_dir, str(class_id))
            os.makedirs(class_folder, exist_ok=True)
```

```

        save_path = os.path.join(class_folder, f"{img_name}_crop.jpg")
        cropped_img.save(save_path)

print("Cropped images saved!")

```

Pentru instruirea modelelor de detecție și clasificare, au fost folosite mai multe seturi de date publice printre care se numără și: [[31, 32, 33, 34, 35, 36, 37, 38, 39]].

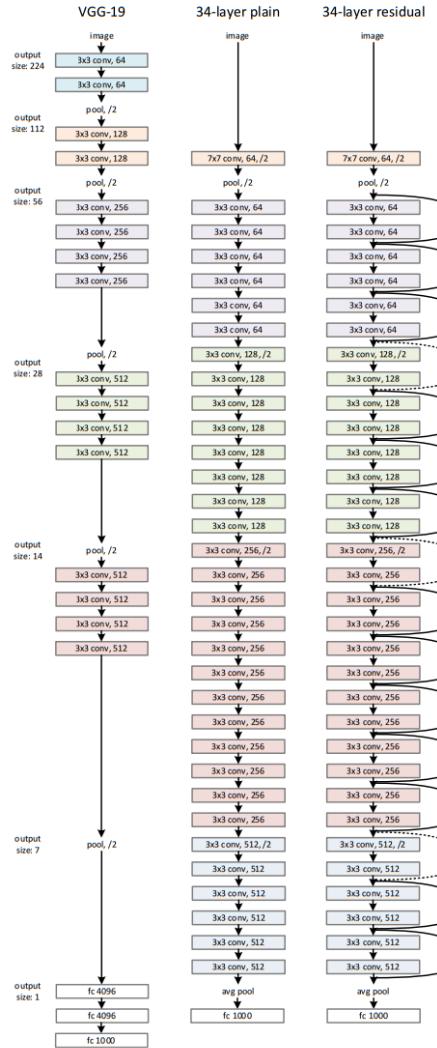


Figura B.1: Arhitectura ResNet [40].