



UNIVERSITATEA DIN
BUCUREŞTI



FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

ASISTENT INTELIGENT ÎN NUTRIȚIE

Absolvent
Tălpigă Andrei-Vlad

Coordonator științific
Lect. dr. Dobrovăț Anca Mădălina

București, iunie 2024

Rezumat

În ultimii ani, inteligența artificială a cunoscut o evoluție exponențială, transformând profund aspecte ale vieții noastre cotidiene. Modelele de învățare automată au devenit un sprijin esențial în numeroase domenii, precum medicină și nutriție, dar și educație. Aceste tehnologii inovatoare ajută la descoperirea unor soluții eficiente, la automatizarea sarcinilor repetitive și la personalizarea experiențelor utilizatorilor.

Obiectivul acestui proiect este dezvoltarea unui asistent nutrițional intelligent, bazat pe un model de învățare automată. Acesta are capacitatea de a recunoaște felurile de mâncare dintr-o poză atașată de către utilizator și de a prezice numărul de calorii și macronutrienți.

Abstract

The evolution of artificial intelligence has been exponential in recent years, transforming aspects of our daily lives profoundly. Machine learning models have become an essential support in many areas, such as medicine and nutrition, as well as education. These innovative technologies help discover effective solutions, automate repetitive tasks and personalize user experiences.

The goal of this project is to develop an intelligent nutritional assistant using a machine learning model. It has the ability to recognize dishes from a photo attached by the user and predict the number of calories and macronutrients.

Cuprins

1 Introducere	5
1.1 Motivație	5
1.2 Domenii abordate	5
1.3 Obiectivul lucrării	5
1.4 Structura lucrării	6
1.4.1 Modelul de clasificare	6
1.4.2 Aplicația Web	6
2 Clasificarea felurilor de mâncare	7
2.1 Setul de date	7
2.1.1 Descrierea setului de date	7
2.1.2 Preprocesarea datelor	7
2.1.3 Augmentarea setului de date	8
2.2 Arhitectura modelului	8
2.3 Configurații (fine-tuning)	13
2.3.1 Greutăți preantrenate	13
2.3.2 Straturile clasificatorului	15
2.3.3 Funcții de activare	16
2.4 Antrenarea modelului	18
2.4.1 Cross validare	19
2.4.2 Optimizatori	20
2.4.3 Funcții de pierdere	23
2.4.4 Antrenarea pe GPU	24
2.5 Testarea modelului	25
2.5.1 Analizarea rezultatelor	25
2.5.2 Testarea pe exemple reale	28
2.6 Extinderea modelului	28
2.6.1 Clasificarea mâncărurilor internaționale	28
2.6.2 Clasificarea mâncărurilor românești	30
3 Aplicația Web	31
3.1 Front-End	31

3.1.1	Bara de navigare	31
3.1.2	Pagina Home	32
3.1.3	Pagina Scan	33
3.1.4	Responsive Design	34
3.2	Back-End	36
4	Concluzii și perspective	39
4.1	Limitări și probleme întâmpinate	39
4.2	Îmbunătățiri și idei viitoare	39
Listă de figuri		41
Bibliografie		43

Capitolul 1

Introducere

1.1 Motivație

Nutriția reprezintă unul dintre aspectele esențiale ale unei vieți sănătoase. Alimentele consumate zilnic au un impact direct asupra sănătății fizice și mintale. O dietă echilibrată nu numai că susține funcționarea optimă a organismului, dar poate preveni o multitudine de afecțiuni, astfel prelungind semnificativ durata de viață.

În ciuda tuturor beneficiilor, mulți oameni nu își monitorizează consumul de calorii și macronutrienți, astfel observând prea târziu dacă apare un deficit sau un surplus. Obiectivul acestei lucrări este de a realiza o aplicație care ușurează tot acest proces, pentru ca mai multe persoane să fie dispuse să aibă grijă de sănătatea proprie. În plus, spre deosebire de alte aplicații sau site-uri similare, platforma va fi disponibilă gratuit, indiferent de regiunea utilizatorului sau dispozitivul de pe care este accesată.

1.2 Domenii abordate

Această lucrare include următoarele domenii:

- **Deep Learning** : antrenarea unei rețele neurale adânci pentru clasificarea felurilor de mâncare cu o acuratețe cât mai mare.
- **Tehnici Web** : realizarea site-ului folosind diverse limbaje și tehnologii. Construirea unei aplicații web robuste și funcționale pe orice tip de dispozitiv.

1.3 Obiectivul lucrării

Lucrarea are ca scop automatizarea analizei dietei și promovarea obiceiurilor alimentare sănătoase. Prin dezvoltarea unui model de recunoaștere a felurilor de mâncare, se

îmbunătățește confortul utilizatorilor, aceștia fiind nevoiți doar să încarce o poză pentru a-și înregistra numărul de calorii și macronutrienți consumați la o masă.

Astfel, această aplicație are obiectivul de a permite oamenilor să își monitorizeze cantitatea de grăsimi, proteine și carbohidrați într-un mod simplu și rapid, fără a fi nevoiți să le adauge manual. Datorită acestei funcționalități, mai puțini utilizatori sunt predispuși să renunțe în drumul spre atingerea obiectivelor personale, obținând rezultate mai bune pe termen lung.

1.4 Structura lucrării

Lucrarea de licență este alcătuită din două părți:

- **Modelul de clasificare** al unui fel de mâncare dintr-o imagine. Modelul primește o poză ca dată de intrare și returnează clasa cu numele mâncării detectate.
- **Aplicația Web** unde utilizatorul poate interacționa cu modelul și are posibilitatea de a vizualiza detalii despre mâncarea consumată.

1.4.1 Modelul de clasificare

Pentru clasificarea în mod corect a unei imagini, au fost încercate diverse arhitecturi și modele preantrenate, configurate în multiple moduri, pentru a găsi structura optimă a unui clasificator de mâncăruri. Așadar, după mai multe teste, a fost aleasă rețea neurală care prezice cele mai precise rezultate și a fost antrenată pe întreg setul de date cu 101 clase de feluri de mâncare.

1.4.2 Aplicația Web

Aplicația web este realizată în HTML, CSS, JavaScript și Flask (framework scris în Python). Aceasta conține două pagini principale:

- **Pagina Home**, unde se află o scurtă descriere a aplicației și un câmp în care utilizatorul își poate scrie numărul de calorii pe care și-l propune să îl consume pe zi.
- **Pagina Scan**, unde utilizatorul poate încărca o imagine cu un fel de mâncare, poza va fi trimisă către server, unde se află modelul antrenat, iar acesta va întoarce clientului rezultatul, care reprezintă numele dish-ului. După aceea, pe pagină apar și informațiile nutriționale ale mâncării detectate, pentru o cantitate de 100 de grame. Se poate introduce și un alt număr de grame pentru care vor fi calculate automat numărul de calorii, proteine, grăsimi și carbohidrați.

Capitolul 2

Clasificarea felurilor de mâncare

2.1 Setul de date

2.1.1 Descrierea setului de date

Obținerea datelor necesare este un proces destul de delicat, care implică verificarea fiecărei imagini și poate afecta rezultatele dacă datele nu sunt etichetate corespunzător. În plus, pentru ca setul să fie cât mai realist, datele trebuie să conțină și puțin zgomot, astfel încât să simuleze inexactități umane normale. Pentru antrenarea cât mai reușită a modelului am folosit setul de date Food-101 [3], care cuprinde cele mai populare 101 feluri de mâncare din lume. Acesta are un număr mare de imagini, verificate și etichetate. În total setul de date are 101.000 de imagini, fiecare clasă conținând câte 1000 de poze, dintre care 750 sunt pentru antrenare, iar 250 pentru testare. În acest dataset nu există duplicate și dimensiunea maximă a unei imagini este de 512 pixeli înălțime, respectiv 512 pixeli lățime.



Figura 2.1: Exemple de imagini din setul de date

2.1.2 Preprocesarea datelor

Preprocesarea datelor se referă la diverse tehnici utilizate pentru a transforma datele brute într-un format adecvat pentru analiza ulterioară. Scopul principal al preprocesării

este de a îmbunătăți calitatea setului de date și de a elimina problemele care pot afecta performanța modelelor și acuratețea rezultatelor.

Toate imaginile vor fi redimensionate astfel încât să aibă înălțimea și lățimea de 512 pixeli. Apoi poza va fi transformată în tensor, iar acesta va avea o dimensiune de 512x512x3. Valorile reprezintă înălțimea, lățimea, respectiv canalele de culoare - 3 în acest caz, deoarece imaginea este color, exprimată prin valori RGB.

Normalizarea implică ajustarea valorilor pixelilor astfel încât acestea să aparțină unui interval dorit [12]. Folosim următoarea funcție din PyTorch pentru a normaliza imaginile.

```
1 torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406],  
    std=[0.229, 0.224, 0.225])
```

Această funcție normalizează un tensor cu valoarea medie și deviația standard. Astfel, pentru fiecare pixel se va aplica formula următoare:

$$\text{output}[c] = \frac{\text{input}[c] - \text{mean}[c]}{\text{std}[c]}, \quad \text{pentru } c = 1, 2, \dots, n,$$

variabila n fiind egală cu 3 în cazul nostru, deoarece imaginea are 3 canale (RGB - roșu, verde, albastru) [5].

2.1.3 Augmentarea setului de date

Augmentarea datelor este o tehnică artificială de mărire a setului de antrenament prin crearea de copii ale datelor originale cu mici modificări. În cazul augmentării imaginilor, se realizează transformări geometrice, precum răsturnarea fotografiei sau rotirea sa cu un anumit număr de grade la stânga sau la dreapta (funcția „RandomRotation”). De altfel, se pot transforma și luminozitatea, contrastul, saturarea sau nuantele imaginii (funcția „ColorJitter”) pentru a adăuga zgomot, făcând setul de antrenament mai realist și diversificat [2].

După încercarea mai multor variante de augmentări ale setului de date, cele mai bune rezultate s-au obținut atunci când s-au folosit, pentru antrenarea modelului, datele originale în combinație cu setul de date augmentat prin următoarea funcție:

```
1 torchvision.transforms.RandomHorizontalFlip()
```

Aceasta sugerează faptul că fiecare imagine are o probabilitate de 50% de a fi răsturnată vertical. Astfel, se pot simula cazuri din viața cotidiană când, din greșeala, fotografia poate fi capturată invers.

2.2 Arhitectura modelului

Alegerea arhitecturii modelului de clasificare este o sarcină importantă în această lucrare. Fiecare tip de rețea neurală adâncă, are avantajele și dezavantajele sale, astfel, prin

mai multe încercări și comparații, s-a reușit găsirea modelului optim pentru clasificarea felurilor de mâncare. Acesta este modelul preantrenat ResNet-50 fine-tuned, configurația sa și straturile adăugate modelului de bază vor fi detaliate ulterior. În acest capitol, se vor prezenta concepțele care stau la baza arhitecturilor testate cu obiectivul clasificării fotografiilor cu mâncare.

Arhitectura ResNet

În anul 2012, AlexNet [10], prima arhitectură bazată pe rețele neurale convecționale [13], a câștigat concursul ImageNet 2012 [4]. De atunci, fiecare arhitectură câștigătoare utilizează mai multe straturi într-o rețea neurală adâncă pentru a reduce rata de eroare. Acest lucru, funcționează pentru un număr mai mic de straturi, însă atunci când se crește considerabil numărul acestora, apare o problemă numită vanishing/exploding gradient. Acest lucru face ca gradientul să devină 0 sau prea mare. Astfel, atunci când se mărește numărul de straturi al unei rețele neurale convecționale, crește și rata de eroare la antrenare și testare.

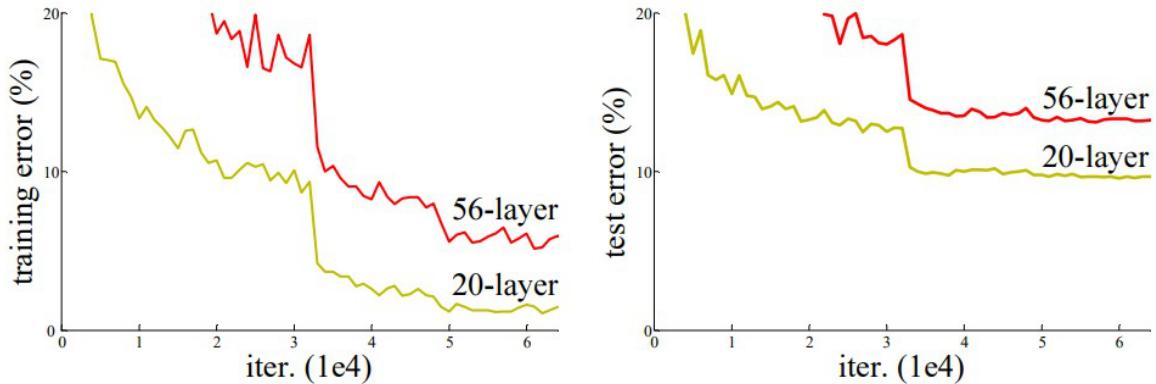


Figura 2.2: Comparație între o arhitectură CNN cu 20 de straturi și una cu 56 de straturi

Astfel, cu scopul combaterii acestei probleme, cercetătorii de la Microsoft au introdus o nouă arhitectură numită Residual Network (rețea reziduală) [7]. Pentru a rezolva problema dispariției/explodării gradientului, această arhitectură a introdus un nou concept numit bloc rezidual. În această rețea, se folosește o tehnică numită skip connections, care conectează activările unui strat la alte straturi, sărind peste unele între ele. Aceasta formează un bloc rezidual. Resnet-urile sunt realizate prin stivuirea acestor blocuri reziduale împreună. Abordarea din spatele acestei rețele este că, în loc de straturi care învăță cartografierea de bază (underlying mapping), putem permite rețelei să se ajusteze cu cartografierea reziduală (residual mapping).

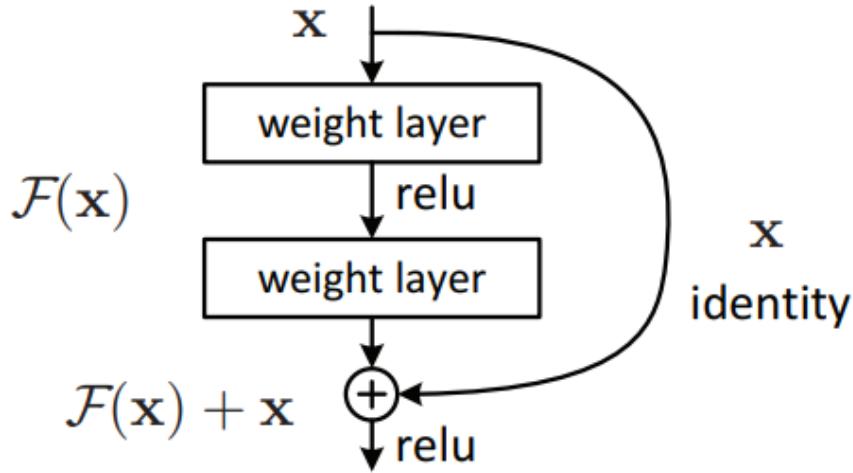


Figura 2.3: Bloc Rezidual (Conexiune skip)

Avantajul adăugării acestui tip de conexiune este că, dacă un strat afectează performanța arhitecturii, atunci acesta va fi omis prin regularizare.

Așadar, există multe variante ale arhitecturii ResNet, care urmează același concept, însă au un număr diferit de straturi, cele mai populare fiind: ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer	
conv1	112×112			$7 \times 7, 64, \text{stride } 2$			
				$3 \times 3 \text{ max pool, stride } 2$			
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$	
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$	
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	
	1×1			average pool, 1000-d fc, softmax			

Figura 2.4: Diferite versiuni ale arhitecturii ResNet

ResNet-18

Modelul ResNet-18 este rețeaua reziduală cu structura cea mai simplă, însă are și câteva avantaje față de celelalte ResNet-uri. Acesta este mai puțin complex și mai rapid de antrenat în comparație cu variante mai adânci, precum ResNet-50 sau ResNet-101. Atinge performanțe bune în sarcinile de clasificare a imaginilor cu mai puțini parametri, făcându-l potrivit pentru scenarii în care resursele computaționale sunt limitate.

ResNet-34

Rețeaua VGG-19 este o rețea neurală conoluțională adâncă, cu 19 straturi, cunoscută pentru simplitatea sa în design, folosind filtre mici (3×3) de conoluție [17]. Rețeaua simplă cu 34 de straturi („34-layer plain”) este inspirată de această filozofie de utilizare a straturilor de conoluție simple și uniforme, dar extinde adâncimea la 34 de straturi (Figura 2.5). Arhitectura Resnet-34 este obținută prin adăugarea conexiunilor reziduale în această rețea simplă.

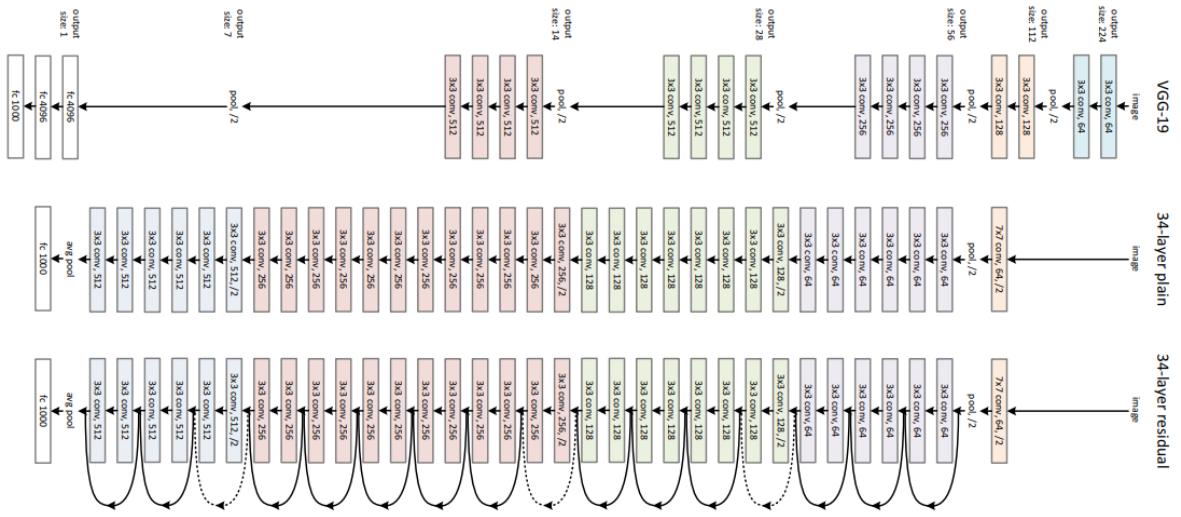


Figura 2.5: Arhitecturile VGG-19, „34-layer plain”, respectiv ResNet-34

ResNet-50

ResNet-50 este o variantă a arhitecturii populare ResNet, eficientă în sarcinile de clasificare a imaginilor, care conține 50 de straturi.

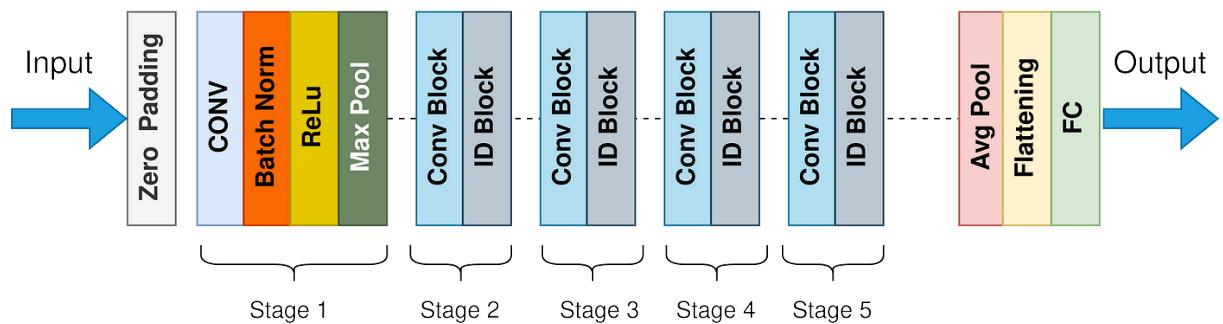


Figura 2.6: Arhitectura modelului ResNet-50

Această versiune este obținută prin înlocuirea fiecărui bloc cu 2 straturi din ResNet-34 (Figura 2.5) cu un bloc rezidual „bottleneck” cu 3 straturi, rezultând, astfel, un ResNet cu 50 de straturi.

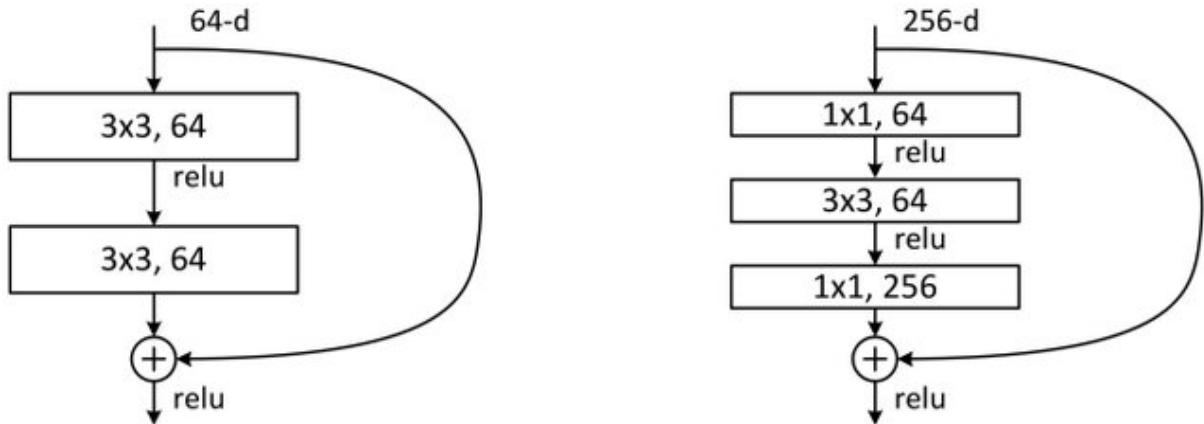


Figura 2.7: Stânga: un Bloc Standard Rezidual pentru ResNet-34. Dreapta: un Bloc „Bottleneck” Rezidual pentru ResNet-50/101/150

Structura „bottleneck”-ului reduce dimensiunea hărților de caracteristici (feature maps) înainte de a le extinde înapoi, micșorând numărul de parametri și costul computațional. În arhitectura ResNet-50, un astfel de bloc este alcătuit din trei straturi convoluționale:

- Convoluție 1x1: Reduce dimensiunea caracteristicilor de intrare (compresie).
- Convoluție 3x3: Procesează caracteristicile (convoluție spațială).
- Convoluție 1x1: Restabilește dimensiunea originală (extindere).

Identitatea este adăugată direct la ieșirea celui de-al treilea strat convoluțional. Acest design facilitează antrenamentul mai ușor al rețelelor neurale adânci și precizia în sarcini de clasificare a imaginilor [15], abordând problema dispariției/explodării gradientului și îmbunătățind fluxul acestora în timpul propagării inverse (backpropagation).

Funcția de activare ReLU (Rectified Linear Unit) [1] se aplică după fiecare strat convoluțional. ReLU permite trecerea doar valorilor pozitive, introducând neliniaritatea în rețea, care este esențială pentru învățarea șabloanelor (pattern-urilor) complexe în date. Această funcție de activare mai are și avantajul că este eficientă computațional în comparație cu alte funcții, precum Sigmoid sau Tanh. Dacă valoarea de intrare în funcția ReLU este mai mare sau egală cu zero, ieșirea este egală cu valoarea de intrare. Dacă valoarea de intrare este negativă, ieșirea este zero. Matematic, funcția ReLU poate fi reprezentată astfel:

$$f(x) = \max(0, x)$$

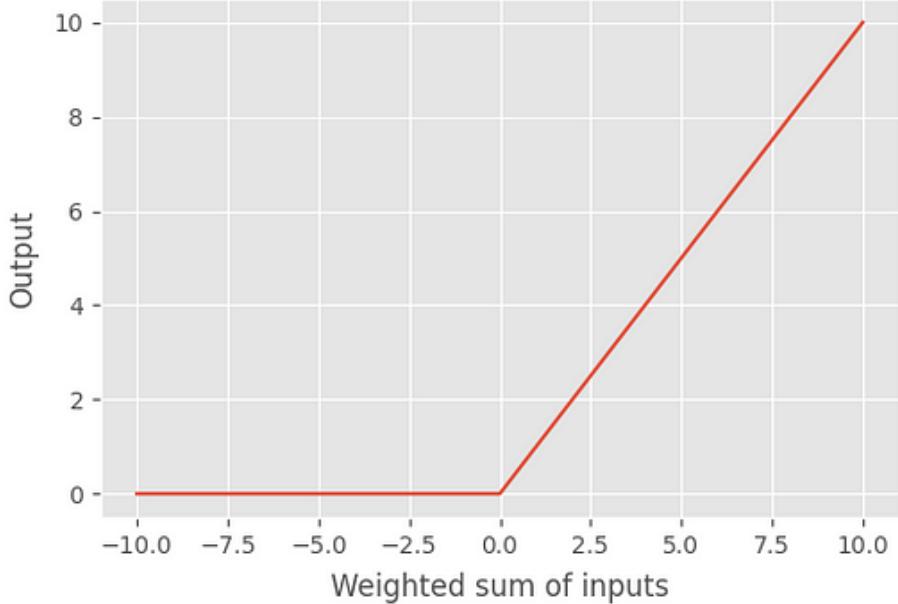


Figura 2.8: Funcția de activare ReLU

2.3 Configurații (fine-tuning)

Noțiunea de fine-tuning în domeniul învățării automate reprezintă procesul de adaptare al unui model preantrenat pentru sarcini specifice și cazuri de utilizare. După mai multe teste și comparații între modelele preantrenate de rețele neurale adânci, a fost aleasă arhitectura ResNet-50, pentru a fi prelucrată ulterior.

2.3.1 Greutăți preantrenate

ImageNet1K, cunoscut și sub numele de ILSVRC (ImageNet Large Scale Visual Recognition Challenge) dataset, este un punct de referință utilizat pe scară largă în computer vision. Este un subset al setului de date ImageNet și a avut un rol esențial în dezvoltarea și evaluarea modelelor avansate de clasificare a imaginilor [4].

În această lucrare se vor folosi greutățile preantrenate pe setul de date ImageNet1K pentru arhitectura ResNet-50. Greutățile preantrenate accelerează procesul de antrenament, deoarece modelul a învățat deja caracteristici „low-level”, cum ar fi marginile, texturile și formele. Astfel, duce adesea la o performanță mai bună în comparație cu antrenarea de la zero, mai ales atunci când setul de date este limitat în dimensiune.

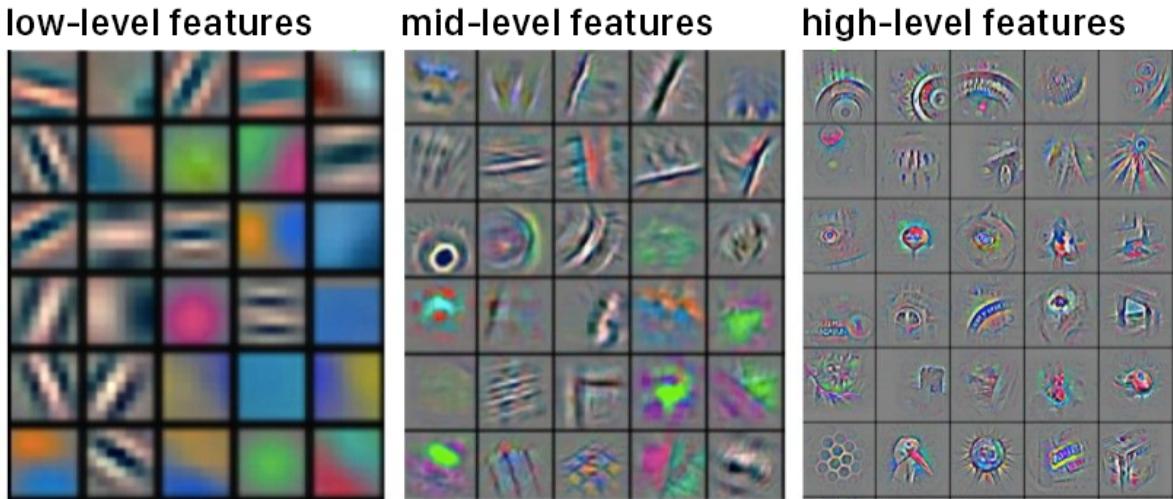


Figura 2.9: Vizualizarea hărților de caracteristici (feature maps)

În această lucrare se utilizează biblioteca PyTorch, din Python, pentru deep learning. Aceasta oferă două versiuni pentru încărcarea unui model ResNet-50 cu greutățile preantrenante pe setul de date ImageNet1K [16]. Prima versiune are o acuratețe de 76.13%, iar a doua, unde s-a folosit o rețetă mai complexă de antrenare, are o acuratețe de 80.85%. În cazul acestui proiect, pentru clasificarea felurilor de mâncare, s-au obținut rezultate mai bune folosind prima variantă (linia 2 de cod).

```
1 from torchvision.models import resnet50, ResNet50_Weights
2 model = resnet50(weights=ResNet50_Weights.IMGNET1K_V1)
3 model = resnet50(weights=ResNet50_Weights.IMGNET1K_V2)
```

Listing 2.1: Încarcarea modelului preantrenat ResNet-50 în PyTorch

După încărcarea modelului preantrenat ResNet-50, straturile se vor converti într-o listă și se va elimina ultimul strat fully connected.

```
1 model = nn.Sequential(*list(resnet.children())[:-1])
```

Înghețarea parametrilor

Prin înghețarea parametrilor se garantează că greutățile preantrenate rămân neschimbăte, de obicei necesitând mai puține resurse și timp de antrenament. Acest lucru este util în scenariile de învățare prin transfer (Transfer Learning [20]), unde se dorește utilizarea caracteristicilor învățate de o rețea preantrenată, fără a fi modificate. Apoi, se adaugă propriile straturi pentru învățarea specifică a feature-urilor „high-level” dorite.

```
1 for param in model.parameters():
2     param.requires_grad = False
```

Listing 2.2: Înghețarea parametrilor modelului preantrenat

2.3.2 Straturile clasificatorului

Clasificatorul procesează caracteristicile de ieșire din feature extractor (ResNet-50) și efectuează împărțirea finală pe clase. Acesta este alcătuit din următoarele tipuri de straturi:

1. **Adaptive Average Pooling:** Acest strat reduce dimensiunile spațiale ale hărților caracteristicilor de intrare, în cazul acesta la 1x1, indiferent de dimensiunea originală. Efectuează operația de average pooling, care calculează media tuturor elementelor din fiecare hartă a caracteristicilor.
2. **Flatten:** Acest strat convertește caracteristicile 2D într-un tensor unidimensional. Aplatizarea este necesară pentru a pregăti datele pentru straturile liniare complet conectate.
3. **Linear:** Stratul liniar, fully connected, primește ca primi doi parametri numărul de caracteristici la intrare, respectiv numărul de caracteristici de ieșire.
4. **Batch Normalization:** Acest strat normalizează output-urile stratului liniar precedent astfel încât valoarea medie să fie 0 și deviația standard să fie 1. Astfel, modelul este mai stabil la antrenare și viteza de convergență spre loss-ul minim crește [8].
5. **Dropout:** Dropout-ul este o tehnică de regularizare care setează la zero, aleatoriu, un procent dat ca parametru din pixeli. Aceasta reduce overfitting-ul și îmbunătățește performanța de generalizare modelului.

```
1 self.classifier = nn.Sequential(  
2     nn.AdaptiveAvgPool2d((1, 1)),  
3     nn.Flatten(),  
4     nn.Linear(2048, 512),  
5     nn.ReLU(),  
6     nn.BatchNorm1d(512),  
7     nn.Dropout(0.25),  
8     nn.Linear(512, 128),  
9     nn.ReLU(),  
10    nn.BatchNorm1d(128),  
11    nn.Dropout(0.1),  
12    nn.Linear(128, 101)  
13 )
```

Listing 2.3: Structura clasificatorului

În final, modelul fine-tuned ajunge la un tensor de dimensiune 101 (mai exact batch_size x 101, dimensiunea batch-ului este declarată la crearea data loader-ului), reprezentând valorile predicțiilor pentru fiecare clasă, astfel modelul va returna clasa cu probabilitatea cea mai mare pentru imaginea dată ca input.

2.3.3 Funcții de activare

În deep learning, funcțiile de activare sunt operații matematice aplicate la ieșirea unui neuron, având scopul de a introduce neliniaritate în rețea neurală. Prin calcularea sumei ponderate și adăugând „bias”-ul, o astfel de funcție decide dacă un neuron trebuie, sau nu, activat [6].

În această lucrare, modelul a fost antrenat folosind cele mai populare funcții de activare. S-au efectuat comparații între ele pentru a determina cea mai bună variantă pentru cazul clasificării felurilor de mâncare. În continuare, se vor prezenta aceste funcții, precum și formulele lor matematice:

- **ReLU** (Rectified Linear Unit): Aceasta este, în prezent, cea mai utilizată funcție de activare în modelele de deep learning. În structura clasificatorului din acest proiect s-a folosit ReLU (Listing 2.3), iar prezentarea detaliată a acestei funcții se află în secțiunea ResNet-50 (ReLU: 2.2).
- **Leaky ReLU** (Leaky Rectified Linear Unit): este un tip de funcție de activare, bazată pe ReLU, dar are o pantă mică pentru valori negative, în loc de o pantă plană.

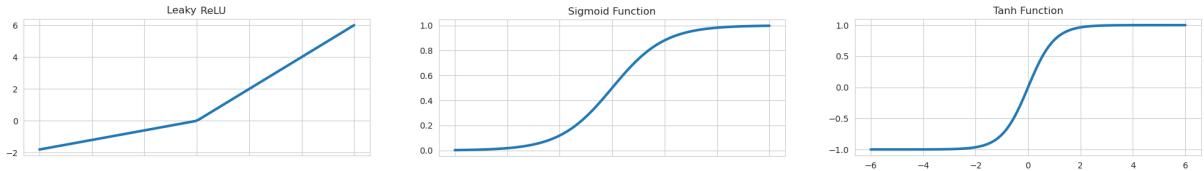
$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0, \alpha \text{ este pantă negativă primită ca parametru} \end{cases}$$

- **Sigmoid**: Ieșirea funcției sigmoide variază între 0 și 1, tindând spre 1 atunci când x are valori foarte mari pozitive, respectiv tindând spre 0 atunci când x are valori foarte mici negative. Acest lucru îl face potrivit pentru aplicații în care ieșirile trebuie interpretate ca probabilități.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Tanh** (Tangenta hiperbolică): Output-ul funcției variază între -1 și 1, acest lucru asigurând faptul că media activărilor este centralizată în jurul lui 0.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



- **GELU** (Gaussian Error Linear Unit):

$$\text{GELU}(x) = x \cdot \frac{1}{2} \left(1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

- **ELU** (Exponential Linear Unit):

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

- **SELU** (Scaled Exponential Linear Unit):

$$\text{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0, \alpha \approx 1.6733, \lambda \approx 1.0507 \end{cases}$$



Pentru accelerarea procesului de găsire a funcției de activare optime pentru modelul fine-tuned, s-a antrenat pe o cincime din setul de date (150 de imagini pentru antrenare și 50 de testare). Din acest motiv, acuratețea a scăzut, însă acest experiment a fost executat doar cu scopul de a compara. Rezultatele au demonstrat faptul că ReLU este cea mai bună opțiune pentru acest clasificator.

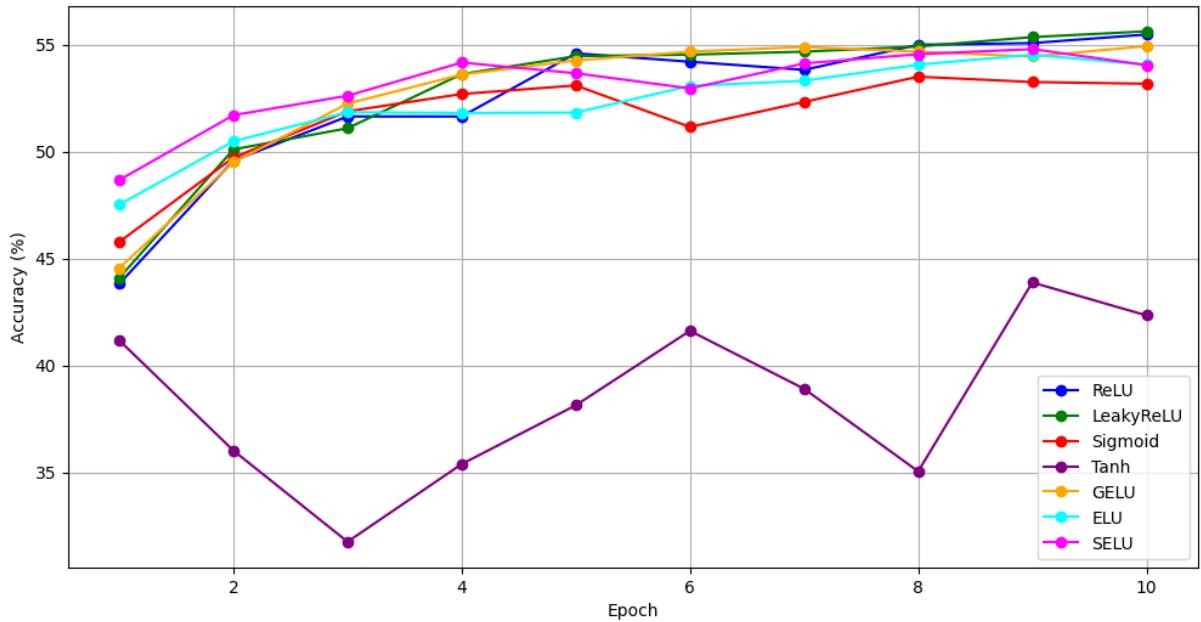


Figura 2.10: Acuratețea modelului pentru diferite funcții de activare

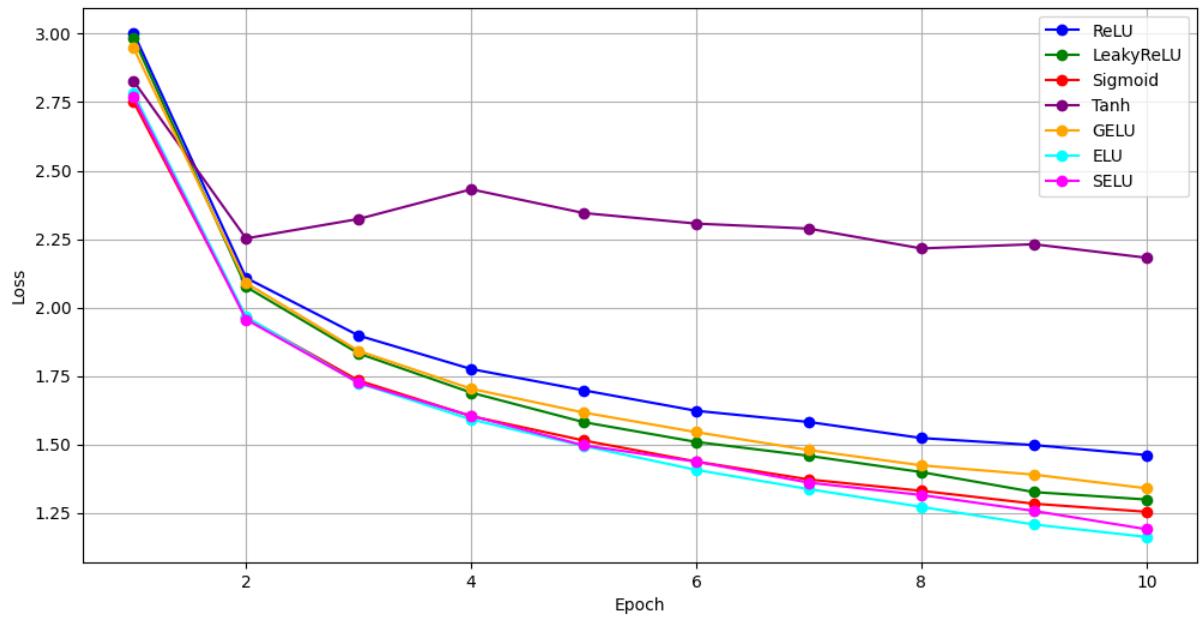


Figura 2.11: Loss-ul modelului pentru diferite funcții de activare

2.4 Antrenarea modelului

Înainte de antrenarea finală a modelului fine-tuned pe întreg setul de date, s-au testat mai multe configurații, cu diferenți parametri, pentru a se găsi varianta optimă. Acest proces a fost efectuat pentru varianta modelului care conține straturi de dropout (4), dar și pentru o variantă care nu conține. Adăugând starturi de dropout se pierde puțină informație din date, însă se reduce probabilitatea ca modelul să facă overfitting, astfel

generalizând mai bine. Datorită rezultatelor mai precise, s-a ales modelul care conține straturi de dropout. Așadar, în acest capitol se vor prezenta tehniciile încercate în căutarea configurației optime.

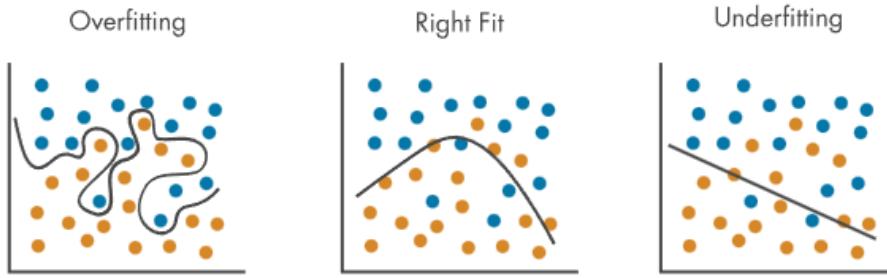


Figura 2.12: Vizualizarea overfitting-ului și underfitting-ului la antrenarea unui clasificator

2.4.1 Cross validare

Validarea încrușiată este o tehnică utilizată în învățarea automată și știința datelor, pentru a facilita calcularea hiperparametrilor optimi ai unui model. Aceasta implică partionarea setului de date în mai multe subseturi, antrenarea modelului pe unele dintre aceste subseturi (setul de antrenare) și testarea acestuia pe subseturile rămase (setul de validare). Procesul se repetă de mai multe ori cu partiții diferite pentru a se asigura că performanța modelului nu depinde de un anumit subset de date.

K-Fold Cross Validation

Setul de date este împărțit în K seturi de dimensiuni egale. Modelul este antrenat de K ori, de fiecare dată folosind K-1 partiții pentru antrenament și partiția rămasă pentru validare. Performanța medie în toate iterările K este utilizată pentru a estima performanța modelului. Numărul de partiții ales pentru acest studiu este egal cu 5. Pentru fiecare clasă vom folosi cele 1000 de imagini din setul de date Food-101, cu scopul testării și validării. La prima iteratie setul de validare constă în primele 200 de imagini, restul fiind de antrenare. La a doua iteratie, setul de validare conține următoarele 200 de imagini (de la imaginea 200 la 400), restul fiind de antrenare. Acest procedeu se mai aplică de trei ori, până la ultima iteratie. În final, modelul va fi testat pe un alt set de date cu aproximativ 100 de imagini pe clasă (în total fiind în jur de 1000 de date pentru testare).

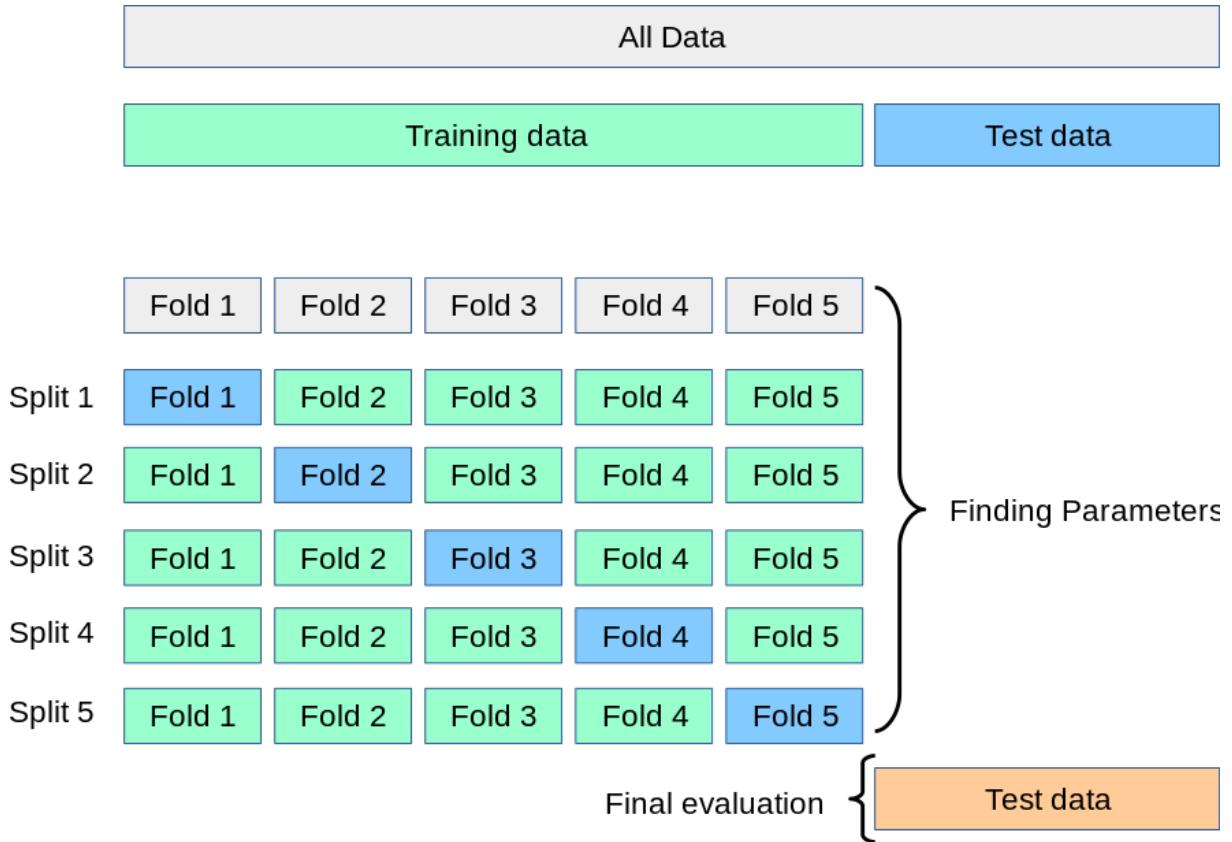


Figura 2.13: K-Fold Cross Validation

Acest algoritm se repetă pentru fiecare configurație care se dorește a fi evaluată. Așadar, se obține o comparație reală între precizia diferitelor configurații, găsind, în final, soluția optimă.

2.4.2 Optimizatori

Optimizatorii sunt algoritmi sau metode utilizate pentru a ajusta atributele rețelei neurale, cum ar fi greutățile și rata de învățare, în scopul minimizării loss-ului din timpul antrenării. Aceștia joacă un rol crucial în determinarea felului în care modelul învăță și converge către o soluție optimă, ajutând la obținerea mai rapidă a rezultatelor.

În aceasta lucrare s-au testat trei tipuri de optimizatori: SGD, Adam și AdamW. Pentru fiecare dintre aceștia, s-a antrenat modelul, cu diferite configurații, pe o epocă, folosind tehnica de cross validare pe 5 split-uri diferite. Astfel, în această secțiune, se vor prezenta toate rezultatele obținute pentru modelul ResNet-50 fine-tuned cu acești optimizatori.

Gradient Descent

Gradient Descent este un algoritm de optimizare bazat pe o funcție convexă, care își ajustează parametrii iterativ, prin deplasarea în direcția opusă celei mai abrupte ascensiuni. Depinde de derivatele funcției de pierdere pentru găsirea minimelor [14].

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla f(\mathbf{w}_t), \text{ unde:}$$

- \mathbf{w}_{t+1} este vectorul parametrului actualizat la iterația $t + 1$,
- \mathbf{w}_t este vectorul parametrului curent la iterația t ,
- α este rata de învățare,
- $\nabla f(\mathbf{w}_t)$ este gradientul funcției f în raport cu parametrii \mathbf{w}_t .

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent este o variantă a algoritmului Gradient Descent care este utilizată pentru optimizarea modelelor de învățare automată. În SGD, se procesează doar un exemplu la un moment dat pentru a se efectua un singur pas. Așadar, dacă setul de date conține 100.000 de rânduri, SGD va actualiza parametrii modelului de 100.000 de ori într-un singur ciclu, spre deosebire de o singură dată în cazul lui Gradient Descent.

Astfel, în loc să se utilizeze întregul set de date pentru fiecare iterație, este selectat aleatoriu un singur exemplu pentru antrenament (sau un batch mic) pentru a calcula gradientul și a actualiza parametrii modelului. Această selecție introduce aleatorismul în procesul de optimizare, de unde vine și termenul „stochastic”. Avantajul utilizării SGD este eficiența sa computațională, în special atunci când lucrează cu seturi mari de date [18].

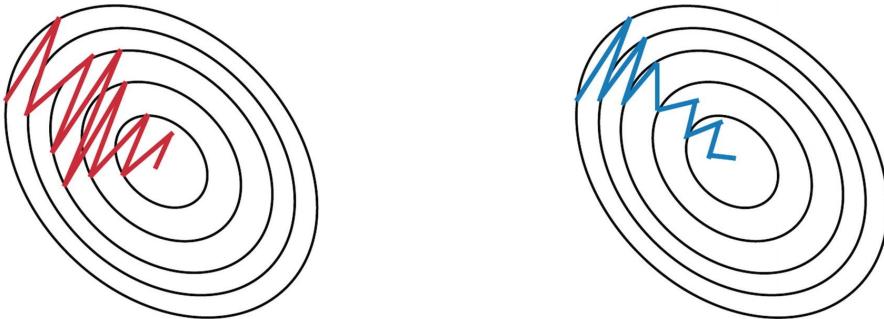


Figura 2.14: Stânga: SGD fără Momentum. Dreapta: SGD cu Momentum

Stochastic Gradient Descent cu Momentum

În SGD nu se calculează derivata precisă a funcției de pierdere. În schimb, se estimează, folosind un batch mic. Acest lucru are ca rezultat deriveate „zgomotoase”, ceea ce înseamnă că nu se deplasează întotdeauna în direcția optimă. Pentru a rezolva această problemă și a atenua zgomotul, momentum-ul a fost introdus în SGD. Accelerează convergența către direcția relevantă și diminuează fluctuațiile în direcții irelevante.

Conceptul din spatele momentum-ului implică eliminarea zgomotului derivatelor prin folosirea unei medii ponderate exponențială, prin atribuirea unei ponderi mai mari actualizărilor recente în comparație cu cele anterioare.

momentum	0,9	0,95	0,99
weight decay			
0,001	41,71%	50,68%	51,35%
0,0001	47,61%	50,65%	51,54%
0,00001	47,74%	50,75%	51,24%

Tabela 2.1: Acuratețea optimizatorului SGD pentru learning rate = 0,001 și diferite valori de momentum și weight decay

Adam

Optimizatorul AdaGrad (Adaptive Gradient) adaptează rata de învățare pentru fiecare parametru în funcție de frecvența și magnitudinea actualizărilor acestuia. Parametrii care primesc actualizări frecvente au ratele de învățare reduse, iar cei cu actualizări rare au ratele de învățare crescute. Acest lucru face ca AdaGrad să fie deosebit de eficient pentru a trata date și caracteristici rare, deoarece se adaptează în mod natural la importanța fiecărui parametru, asigurând o convergență mai rapidă pentru caracteristicile care apar frecvent și permitând caracteristicilor rare să primească actualizări mai mari atunci când apar.

Optimizatorul RMSProp (Root Mean Square Propagation) este un algoritm cu learning rate adaptiv, conceput pentru a aborda limitările lui AdaGrad, împiedicând ratele sale de învățare să devină prea mici. RMSProp menține o medie mobilă a gradienților pătrați pentru fiecare parametru și utilizează aceste informații pentru a ajusta rata de învățare. Acest lucru ajută la stabilizarea și accelerarea procesului de antrenament prin atenuarea oscilațiilor în actualizările gradienților.

Adam (Adaptive Moment Estimation) este un algoritm avansat de optimizare utilizat în antrenarea modelelor de învățare automată, în special a rețelelor neurale adânci. Combină avantajele altor două extensii ale optimizatorului SGD: AdaGrad și RMSProp. Adam calculează ratele de învățare adaptive pentru fiecare parametru, menținând mediile curente atât ale gradienților, cât și ale valorilor lor pătrate. Aceste medii de funcționare sunt corectate pentru a îmbunătăți stabilitatea și performanța, făcând optimizatorul Adam deosebit de eficient pentru seturi mari de date și modele complexe [9].

learning rate	0,01	0,005	0,001	0,0009	0,0005	0,0001
accuracy	47,66%	49,04%	51,67%	51,28%	51,84%	47,35%

Tabela 2.2: Acuratețea optimizatorului Adam pentru diferite valori de learning rate

AdamW

AdamW (Adaptive Moment Estimation with Weight Decay) este o variantă a optimizatorului Adam care decuplează scăderea greutății (regularizarea L2) de actualizarea gradientului. În Adam, scăderea greutății este implicit inclusă în actualizarea parametrelor, ceea ce poate duce la convergență ineficientă. AdamW abordează această problemă, scăzând separat, în mod explicit, termenul de weight decay din actualizarea gradientului, ceea ce duce la o performanță mai bună și un antrenament mai stabil. Această decuplare ajută la obținerea unei mai bune generalizări și este deosebit de eficientă în reducerea overfitting-ului [11].

weight decay	betas	(0,9, 0,999)	(0,95, 0,999)	(0,85, 0,999)
0,01		51,30%	51,70%	51,15%
0,001		51,64%	51,64%	50,96%
0,0001		51,37%	51,93%	51,14%

Tabela 2.3: Acuratețea optimizatorului AdamW pentru learning rate = 0,001 și diferite valori de betas și weight decay

În concluzie, cea mai bună acuratețe (51,93%) a fost obținută folosind optimizatorul AdamW, având configurația următoare: learning rate = 0,001, weight decay = 0,0001 și betas = (0,95, 0,999).

2.4.3 Funcții de pierdere

Funcțiile de pierdere (loss functions) în rețelele neurale sunt funcții matematice utilizate pentru a compara diferența dintre rezultatul prezis de model și valorile țintă reale. Acestea ghidează procesul de optimizare, măsurând cât de bine modelează rețeaua neurală datele de antrenament. În timpul antrenamentului, optimizatorul ajustează parametrii modelului pentru a minimiza această pierdere [19]. Cele mai folosite funcții de pierdere în domeniul învățării automate sunt următoarele:

- **Mean Squared Error Loss (MSE):** calculează media diferențelor pătrate dintre valorile prezise și cele reale, fiind folosit în sarcini de regresie.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- **Mean Absolute Error Loss (MAE)**: calculează media valorilor absolute ale diferențelor dintre valorile prezise și cele reale, fiind o altă opțiune comună pentru sarcinile de regresie.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- **Categorical Cross-Entropy Loss**: calculează pierderea pentru fiecare predicție, fiind folosit pentru sarcini de clasificare cu mai multe clase.

Categorical Cross-Entropy = $-\sum_{i=1}^n \sum_{j=1}^c y_{i,j} \log(\hat{y}_{i,j})$, unde c este numărul de clase

- **Hinge Loss**: este folosită, în primul rând, pentru formarea clasificatorilor, în special în Support Vector Machines (SVM). Aceasta calculează pierderea pe baza marjei dintre clasele anticipate și cele reale.

$$\text{Hinge Loss} = \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i \cdot \hat{y}_i)$$

În această lucrare, s-a utilizat funcția de pierdere Categorical Cross Entropy pentru clasificarea modelului în cele 101 clase. Această funcție a furnizat cele mai bune rezultate în timpul antrenării rețelei neurale adânci ResNet-50 fine-tuned.

```
1 loss_fn = torch.nn.CrossEntropyLoss()
```

Listing 2.4: Încărcarea funcției Cross Entropy Loss în Pytorch

2.4.4 Antrenarea pe GPU

Antrenarea rețelelor neurale pe GPU-uri folosind CUDA (Compute Unified Device Architecture) accelerează semnificativ procesul de antrenament, permitând gestionarea mai eficientă a seturilor mari de date și a modelelor complexe decât pe procesoare (CPU). CUDA este o platformă de calcul și un model de interfață de programare a aplicațiilor (API) creat de NVIDIA. Permite dezvoltatorilor să folosească GPU-uri NVIDIA pentru procesarea de uz general, valorificând puterea masivă de calcul în paralel pentru sarcini care erau gestionate în mod tradițional de procesoare. Mai multe framework-uri populare pentru deep learning, precum PyTorch, acceptă CUDA și simplifică procesul de antrenare a rețelelor neurale pe unități de procesare grafică.

```
1 device = torch.device('cuda' if torch.cuda.is_available() else
                      'cpu')
2 model.to(device)
```

Listing 2.5: Mutarea modelului pe GPU

2.5 Testarea modelului

Testarea unei rețele neurale presupune evaluarea performanței acesteia pe un set de date pe care nu l-a văzut în timpul fazei de antrenament. Aceasta este un pas esențial în dezvoltarea și implementarea rețelelor neurale, deoarece asigură faptul că modelul generalizează bine pe date nevăzute și funcționează cu acuratețe în scenariile din lumea reală.

2.5.1 Analizarea rezultatelor

Înaintea antrenării, setul de date Food-101 a fost divizat, având 75% dintre imagini pentru antrenare și restul de 25% pentru testare (2.1.1). Astfel, în această secțiune, se va prezenta acuratețea obținută pentru diferitele arhitecturi încercate.

ResNet-18

Rețeaua neurală ResNet-18 (2.2) fine-tuned a furnizat cea mai bună acuratețe cu următoarea configurație:

```
1 self.classifier = nn.Sequential(  
2     resnet18,  
3     nn.Flatten(),  
4     nn.Linear(512, 256),  
5     nn.ReLU(),  
6     nn.BatchNorm1d(256),  
7     nn.Dropout(0.25),  
8     nn.Linear(256, 128),  
9     nn.ReLU(),  
10    nn.BatchNorm1d(128),  
11    nn.Dropout(0.1),  
12    nn.Linear(128, 101)  
13 )
```

Listing 2.6: Structura clasificatorului bazat pe ResNet-18

În timpul antrenării acestui model, s-a utilizat funcția de pierdere Cross Entropy și optimizatorul Adam (2.4.2) cu rata de învățare egală cu 0,001.

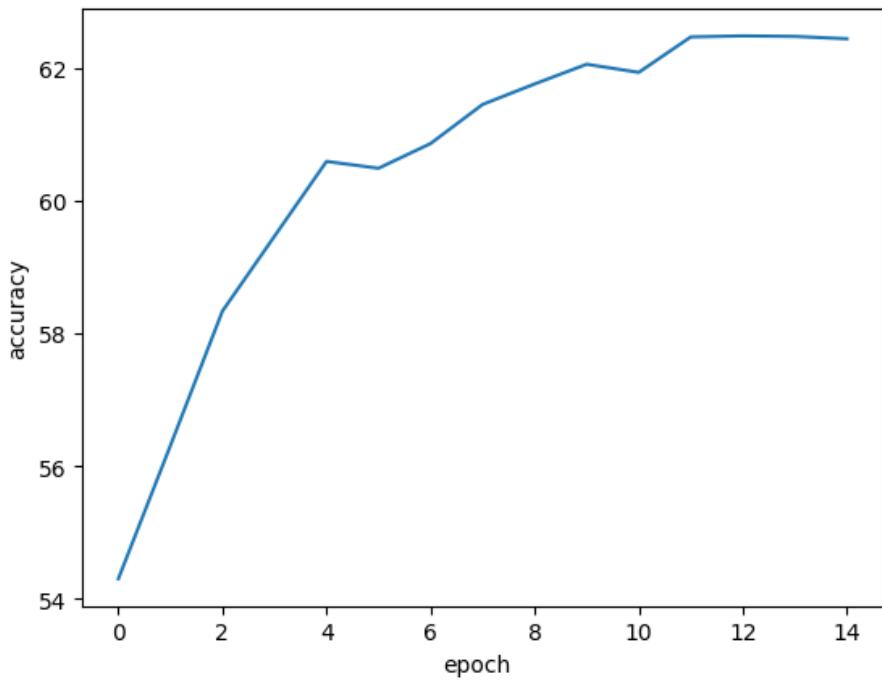


Figura 2.15: Acuratețea modelului ResNet-18 fine-tuned pe 15 epoci

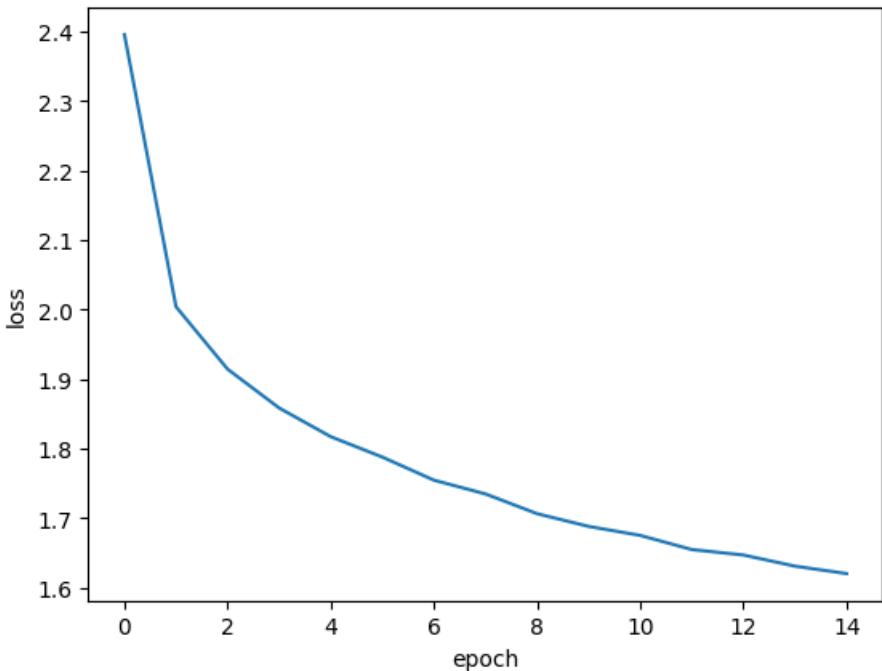


Figura 2.16: Loss-ul modelului ResNet-18 fine-tuned pe 15 epoci

ResNet-50

În această lucrare, cea mai ridicată acuratețe a fost obținută folosind, la bază, modelul ResNet-50 (2.2) pentru extragerea caracteristicilor fiecărei imagini, urmat de un

clasificator cu straturi fully connected, prezentat în secțiunea „Straturile clasificatorului” (2.3).

Configurația optimă pentru antrenarea acestui model este reprezentată de funcția Cross Entropy Loss, respectiv optimizatorul AdamW cu learning rate = 0,001, weight decay = 0,0001 și betas = (0,95, 0,999).

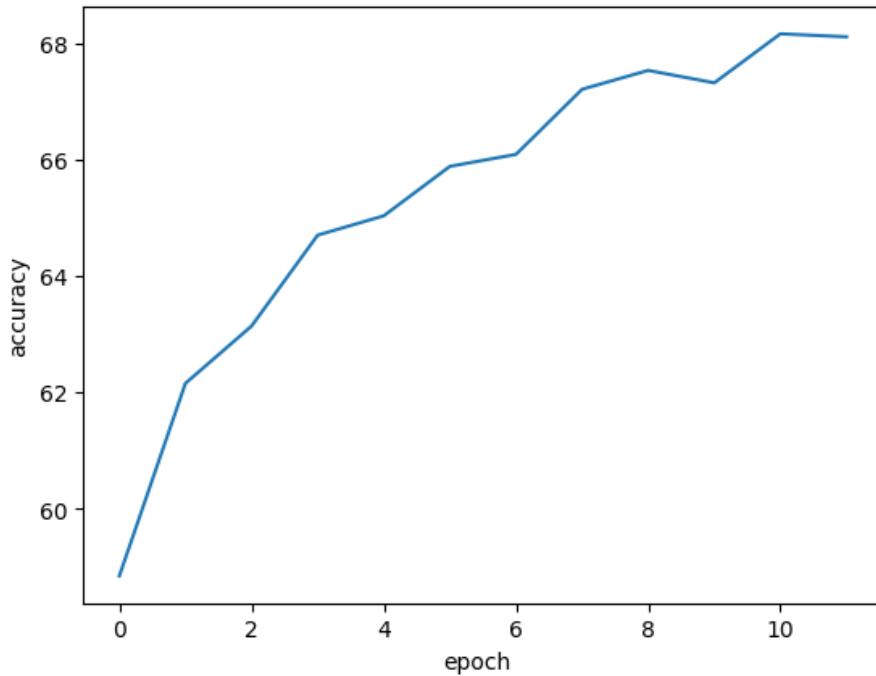


Figura 2.17: Acuratețea modelului ResNet-50 fine-tuned pe 12 epoci

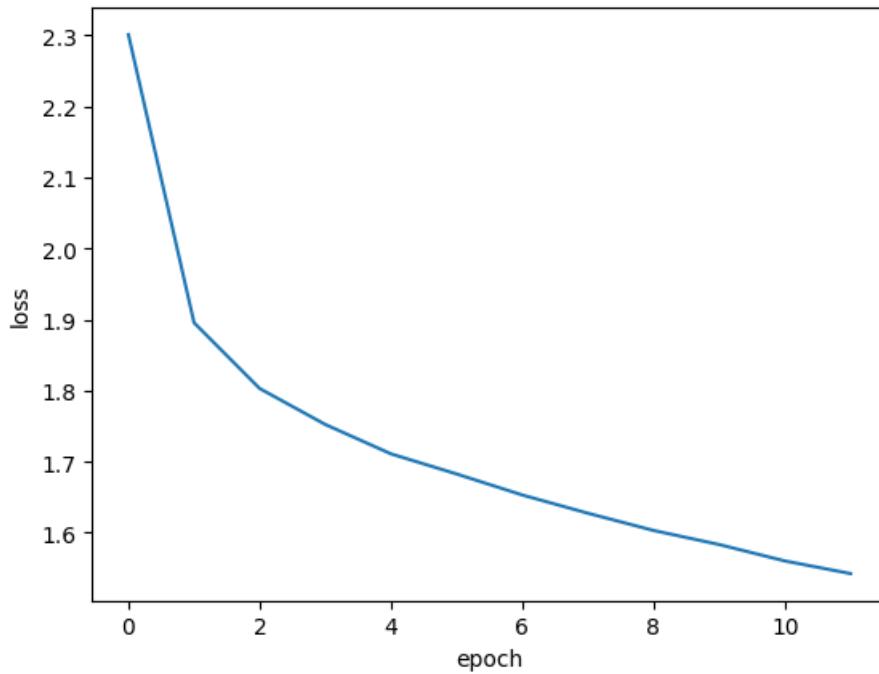


Figura 2.18: Loss-ul modelului ResNet-50 fine-tuned pe 12 epoci

2.5.2 Testarea pe exemple reale

Pentru a confirma acuratețea acestui clasificator și pe alte exemple, în afară de setul de testare din Food-101, s-au descărcat de pe internet aproximativ 100 de poze pe clasă, care au fost verificate ulterior și manual. Astfel, modelul antrenat anterior a fost testat și pe acest set de date format din imagini reale, conținând în jur de 10.100 de exemplare, fără duplicate.

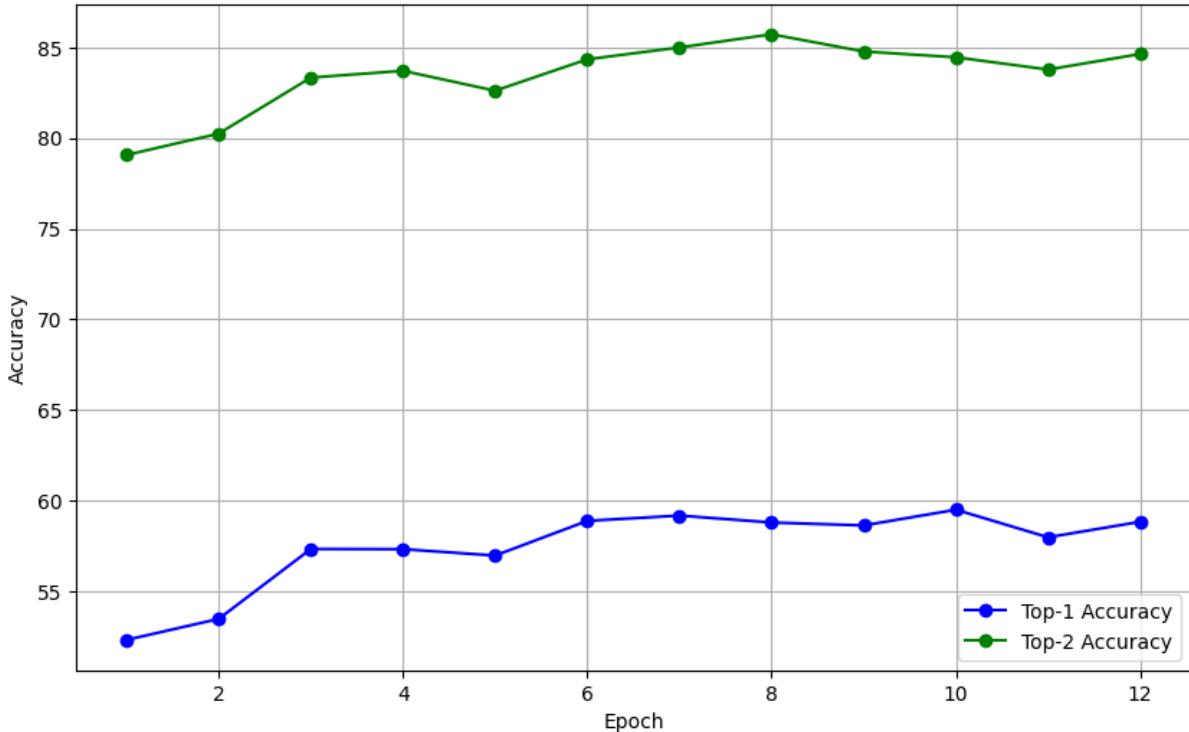


Figura 2.19: Acuratețea Top-1 și Top-2 pentru modelul ResNet-50 fine-tuned, testat pe imagini reale

Metrica Top-2 accuracy este relevantă, deoarece, în setul de date, sunt multe clase similare două câte două, precum spaghetti carbonara - spaghetti bolognese, salată caesar - salată grecească. Modelul are dificultăți în a le deosebi, deoarece o imagine bidimensională oferă o cantitate limitată de informație, neștiind ce se află în „adâncimea” pozei.

2.6 Extinderea modelului

2.6.1 Clasificarea mâncărilor internaționale

Din dorința de a extinde acest model pentru a conține și mâncăruri tradiționale din România, au fost adăugate încă 4 feluri de mâncare: ciorbă de burtă, cozonac, papanasi și sarmale. Astfel, la cele 101 clase de mâncăruri internaționale din datasetul Food-101, se mai adaugă și cele 4 clase precizate anterior. Pentru fiecare dintre clasele noi, s-au

descărcat câte 750 de poze de pe internet, iar din clasele datasetului Food-101 s-au păstrat câte 750 de imagini din fiecare, cu scopul de a menține balanța reprezentării fiecărei clase. Astfel, „bias”-ul este menținut la un nivel minim. Așadar, pentru a clasifica 105 tipuri de mâncare s-a utilizat același model (ResNet-50 fine-tuned) și clasificator (2.3), schimbând doar ultimul strat liniar, pentru ca rețeaua neurală să aibă 105 clase de ieșire.

```
1 nn.Linear(128, 105)
```

Listing 2.7: Ultimul start al clasificatorului pentru 105 clase de mâncare

Modelul este antrenat pe un set de antrenament, alcătuit din câte 600 de imagini aleatorii din fiecare clasă, iar setul de testare constă în restul datelor (câte 150 din fiecare clasă).

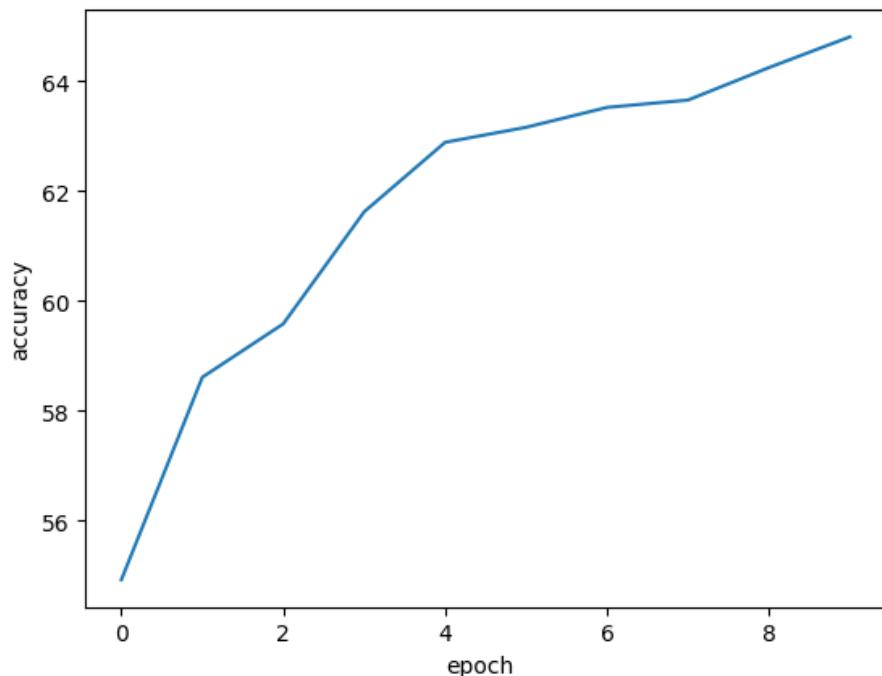


Figura 2.20: Acuratețea modelului pentru cele 105 clase

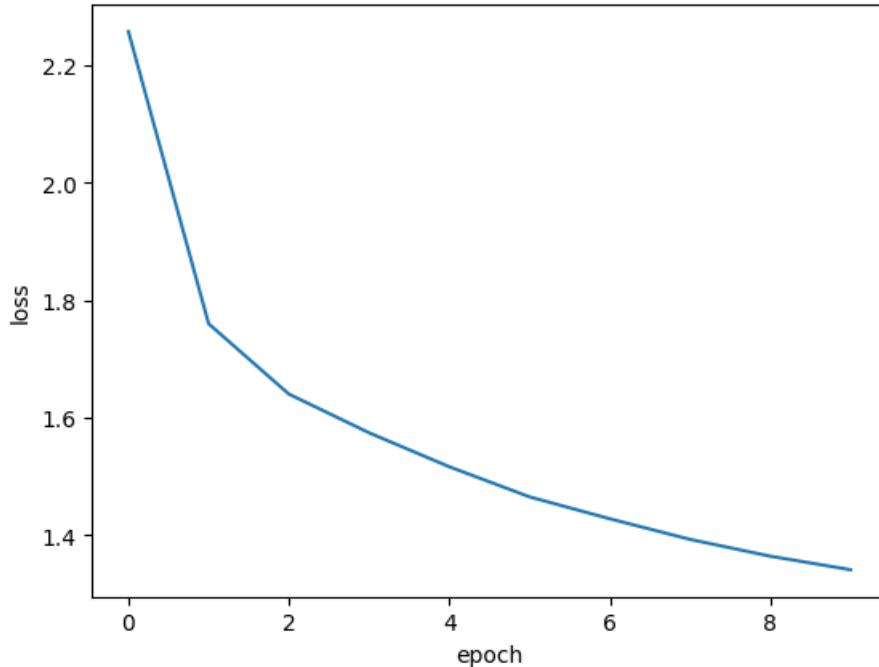


Figura 2.21: Loss-ul modelului la antrenarea pe cele 105 clase

2.6.2 Clasificarea mâncărilor românești

Cu motivația de a crea un clasificator doar pentru mâncăruri cu specific românesc, s-a antrenat un model care să clasifice 4 feluri de mâncare din România. Astfel, s-au folosit cele 3000 de imagini descărcate de pe internet (750 date x 4 clase), împărțite în același fel: 600 de date de antrenament și restul de 150 pentru testare, pe clasă. S-a utilizat clasificatorul creat pentru antrenarea pe Food-101, fiind schimbat doar ultimul strat.

```
1 nn.Linear(128, 4)
```

Listing 2.8: Ultimul start al clasificatorului pentru 4 clase de mancare

În urma testării, s-a obținut o acuratețe de aproximativ 97% pentru aceste 4 clase. Această acuratețe se datorează și numărului mic de clase, însă acest test a fost făcut în perspectiva extinderii setului de date pe mai multe clase de mâncare.

Capitolul 3

Aplicația Web

Aplicația Web are scopul de a permite oricărui utilizator să interacționeze cu modelul de clasificare într-un mod intuitiv. Aceasta are obiectivul de a permite oamenilor să își monitorizeze cantitatea de grăsimi, proteine și carbohidrați într-un mod simplu și rapid, fără a fi nevoiți să le adauge manual.

3.1 Front-End

Tehnologiile utilizate în acest proiect, pentru partea de front-end, sunt:

- **HTML (HyperText Markup Language)**: creează structura și conținutul paginilor web.
- **CSS (Cascading Style Sheets)**: controlează aspectul, dimensiunea și stilul unui document scris în HTML.
- **JavaScript**: adaugă un comportament dinamic și interactiv paginilor web.

Aplicația web conține o imagine de fundal în background, pentru un efect vizual plăcut. Tot textul este scris cu același font, însă mărimi și culori diferite în funcție de importanță.

3.1.1 Bara de navigare

Bara de navigare conține, în extremitatea stânga, logo-ul aplicației, care este și element clickable și conține un link către pagina home. În centru sunt amplasate cele trei butoane: home, scan și calendar, fiecare având scopul de a duce userul pe pagina respectivă, atunci când este apăsat. În dreapta, bara prezintă un buton, numit profile, cu obiectivul de a transporta userul în pagina contului personal sau pe pagina de logare și autentificare, dacă niciun cont nu este conectat. În plus, atunci când cursorul se află peste oricare dintre cele patru butoane (atributul hover), acesta își mărește dimensiunea de 1.1 ori și textul se face cărămiziu.

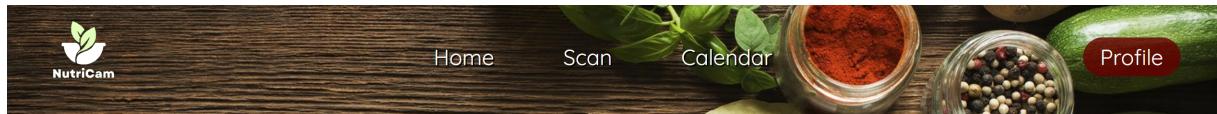


Figura 3.1: Bara de navigare

3.1.2 Pagina Home

Pagina home este pagina principală, de introducere a utilizatorului în contextul aplicației. Aceasta conține un slider, cu două diapositive, care se derulează automat, la fiecare 20 de secunde, sau instant, atunci când se apasă săgeata stânga sau dreapta, cu scopul navigării de pe un slide pe altul. Atunci când se plasează cursorul deasupra uneia dintre cele două butoane, se inversează culorile săgeții cu fundalului acesteia. Fiecare slide apare printr-o tranziție atunci când îi vine rândul, iar partea de jos a paginii se înnegrește gradual pentru a crea un efect vizual satisfăcător.

Pe primul slide este scrisă o scurtă descriere a obiectivului aplicației și beneficiile folosirii acestei platforme.

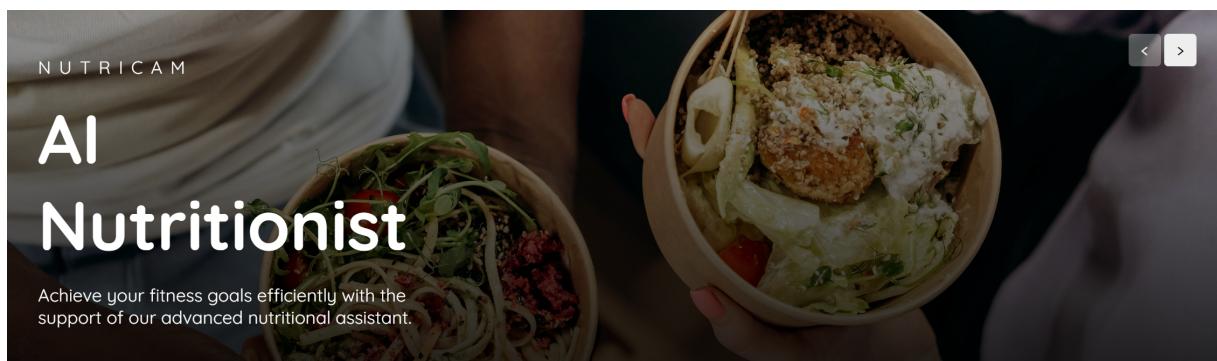


Figura 3.2: Primul slide de pe pagina home

Pe al doilea slide este descrisă principala funcționalitate a site-ului: monitorizarea numărului de calorii și macronutrienților consumați zilnic. În partea de jos a paginii se află un câmp, unde fiecare utilizator își poate seta target-ul zilnic de calorii.



Figura 3.3: Al doilea slide de pe pagina home

3.1.3 Pagina Scan

Pagina scan este dedicată încărcării unei poze cu un fel de mâncare și vizualizarea informațiilor despre acesta. La început, pagina conține doar un buton „Upload Image”, care, atunci când este apăsat, deschide o fereastră care permite selectarea unei imagini din dispozitiv.

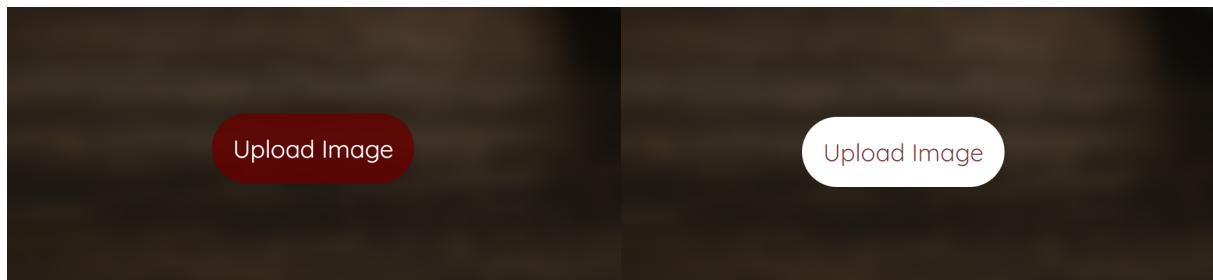


Figura 3.4: Stânga: Butonul de Upload. Dreapta: Butonul de Upload atunci când mouse-ul se află deasupra sa (hover)

Astfel, imaginea este transmisă serverului, care, ulterior, întoarce un răspuns, conținând clasa (mâncarea) și atributele acesteia (număr de calorii, proteine, grăsimi și carboidrați). Modelul procesează aproape instant imaginea, după care butonul de upload dispără. Apoi, pagina va fi împărțită în două, în partea stângă fiind afișată imaginea. În dreapta paginii vor apărea următoarele:

- Un titlu cu numele felului de mâncare.
- Un câmp unde se poate introduce orice număr de grame, de obicei greutatea porției de mâncare, și un buton de submit.
- Numărul de calorii pe 100 de grame pentru clasa detectată, precum și o diagramă a cantităților de macronutienți (exprimate în grame). Cantitatea exactă a fiecărui se

poate vizualiza atunci când utilizatorul plasează cursorul pe o porțiune din diagramă. Astfel, pe bucata roșie vor apărea proteinele, pe cea turcoaz grăsimile, iar pe fragmentul albastru se află carbohidrații. Toate valorile cantităților sunt rotunjite.

- Atunci când un utilizator apasă butonul de submit, va mai apărea încă o diagramă, în dreapta celeilalte, separată printr-o linie punctată, care va conține aceleași date, însă pentru cantitatea introdusă în câmpul de sub titlu. Butonul de submisie funcționează doar atunci când este introdusă, în câmp, o valoare mai mare ca zero, evitându-se, astfel, erorile.

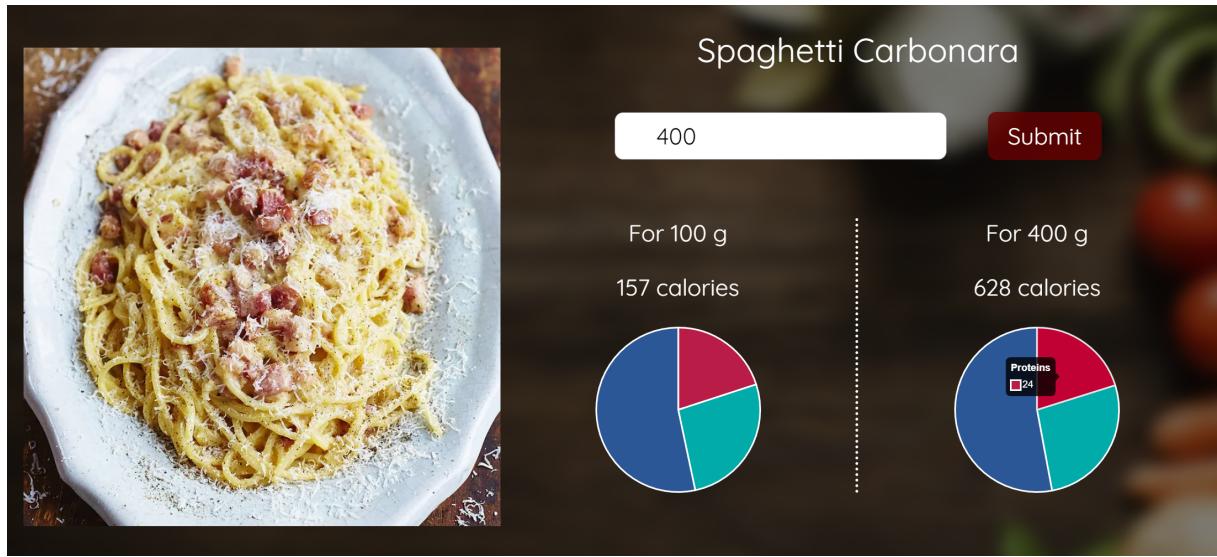


Figura 3.5: Pagina Scan după ce se încarcă o imagine

3.1.4 Responsive Design

Designul responsive asigură faptul că aplicația web arată și funcționează bine pe o varietate de dispozitive și dimensiuni de ecran. Acest lucru se realizează folosind HTML, CSS și JavaScript pentru a crea un layout flexibil și scalabil. Pentru ca tot site-ul să funcționeze optim este necesar ca fiecare pagină și element al acesteia să se adapteze în funcție de dimensiunile ecranului.

Bara de navigare

Bara de navigare dispare atunci când lățimea ecranului este mai mică de 1000 de pixeli. În schimb, va apărea o iconiță cu 3 linii, care reprezintă un meniu de tip dropdown. Când aceasta este apăsată, se va deschide meniul, unde vor apărea numele paginilor, unul sub celălalt. Iconița specifică „burger” meniului va fi înlocuită acum de simbolul „X”, care va închide meniul când este apăsat. Pentru un plus de vizibilitate, fundalul căsuței meniului este blurat și va fi afișat peste orice element din pagină, având atributul z-index foarte

mare. Pe de altă parte, când se deschide, meniul va avea o tranziție cu efect de cortină timp de o secundă.

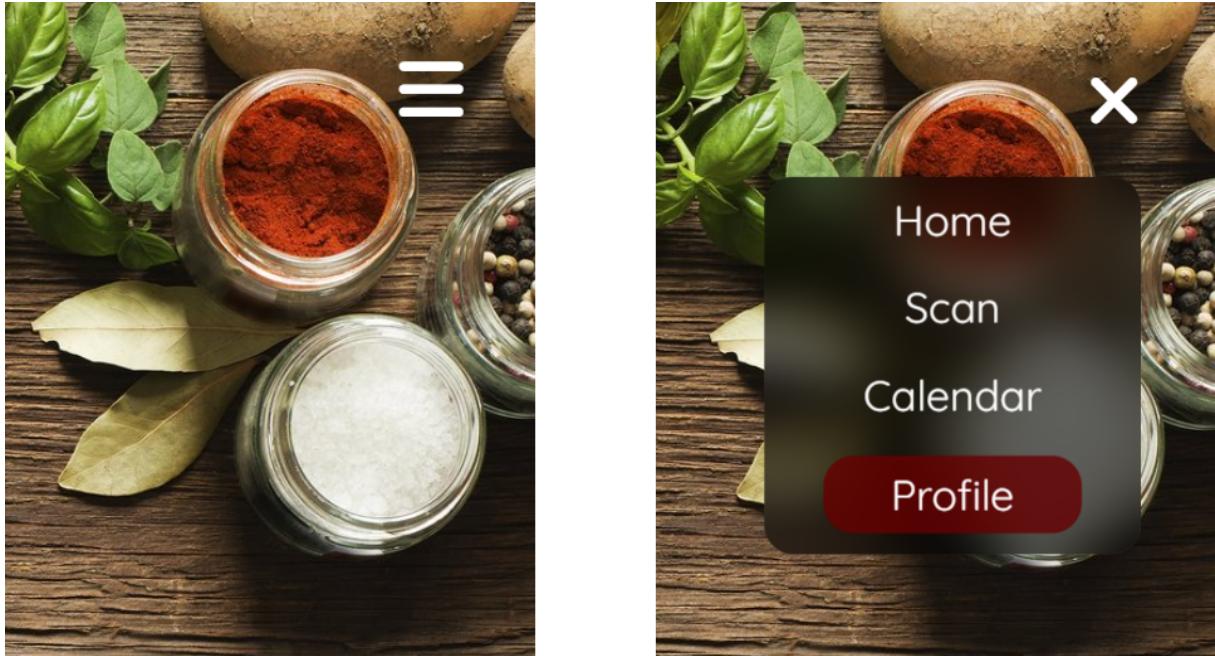


Figura 3.6: Meniul dropdown, când lățimea paginii este mai mică de 1000 de pixeli

Pagina Home

Atunci când lățimea paginii home este sub 700 de pixeli, săgețile își schimbă poziția, astfel vor apărea în stânga paginii, sub logo. În formatul normal al paginii, acestea se situează în dreapta, sub butonul „Profile” (Figura 3.2). În plus, textul se micșorează pentru a se încadra mai bine în pagină.

Pagina Scan

Atunci când pagina scan are valoarea lățimii între 800 și 1300 de pixeli, imaginea va fi repozitionată deasupra titlului clasei de mâncare, în loc de a fi așezată în stânga acestuia. Când lățimea are sub 800 de pixeli, mărimea textului scade, iar căsuța unde se introduce greutatea porției și butonul de submit se micșorează. Diagramele și distanța dintre acestea scad, pentru a putea încăpea în regiunea destinată. Spațiul blurat este scrollable, astfel se va derula doar informația de pe pagină, în timp ce poza de fundal va rămâne nemîscată.

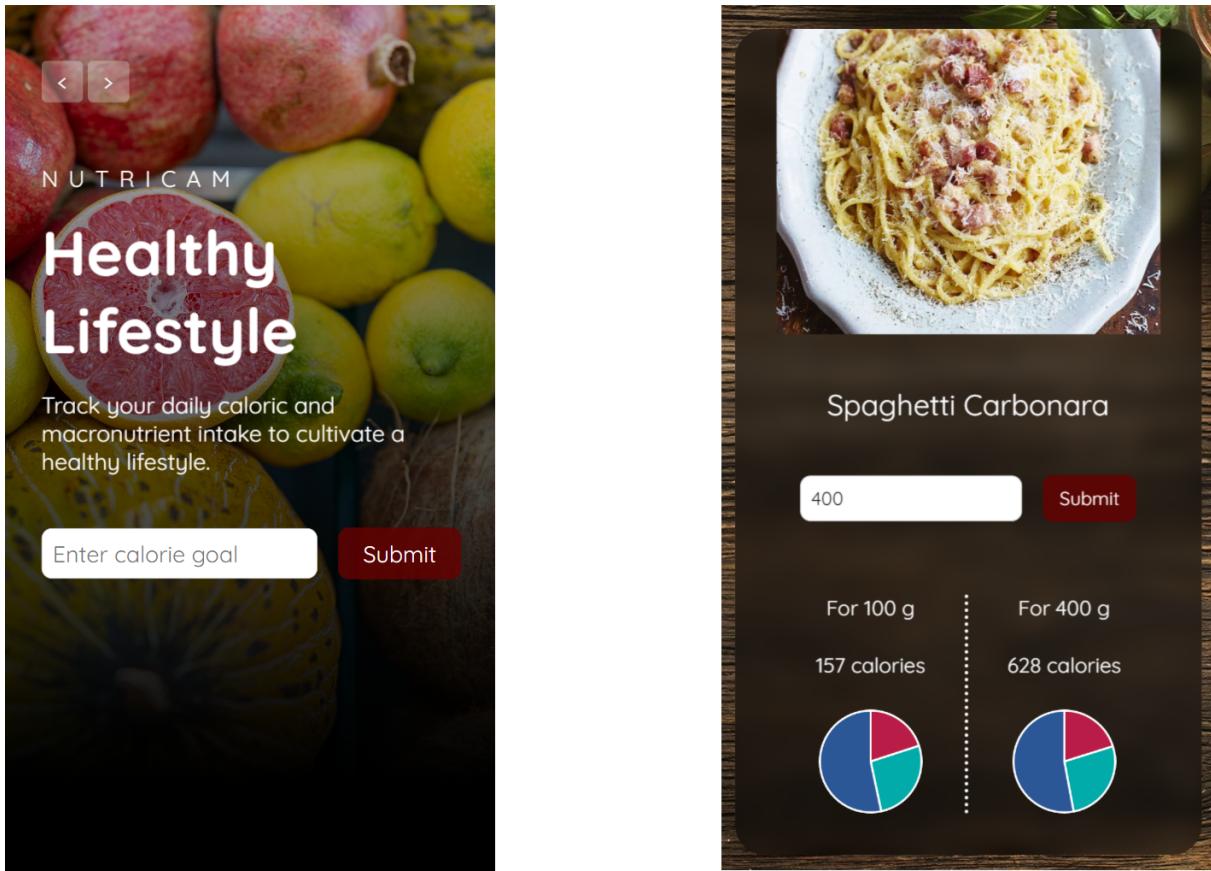


Figura 3.7: Paginile Home, respectiv Scan, atunci când mărimea ecranului este de 500 pixeli x 1000 pixeli

3.2 Back-End

Pentru crearea serverului din partea de back-end a aplicației am folosit Flask (framework scris în Python). Serverul conține, de asemenea, și câteva fișiere auxiliare pentru interacționarea cu modelul de învățare automată:

- Un fișier cu extensia „.pkl” (pickle), care conține un dicționar cu stările rețelei neurale adânci, antrenată pe tot setul de date.
- Un fișier Python cu o clasă, care definește structura modelului fine-tuned, folosit pentru clasificare.
- Un fișier Python care conține o clasă de tip „Enum”, unde sunt salvate toate felurile de mâncare, împreună cu atributele acestora (nume, număr de calorii, cantitatea de macronutrienți). Aceasta acționează ca o bază de date (Listing 3.1).
- O clasă „ModelManager” în Python, care conține următoarele trei funcții:
 1. `save_image_from_json`: decodifică o imagine din formatul base64 și o salvează la un path dat ca parametru.

2. preprocess_image: primește ca parametru calea corespunzătoare unei imagini care va fi: transformată în color ("RGB") dacă nu este deja, redimensionată în 512x512, transformată în tensor și normalizată.
3. classify_image: primește informațiile unei imagini în format json, o salvează într-un anumit folder, folosind funcția save_image_from_json, apoi preprocesează imaginea salvată (funcția preprocess_image), încarcă rețea neurală antrenată din fișierul pickle și evaluează fotografia folosind modelul. Aceasta va întoarce clasa detectată, iar, în final, funcția va returna obiectul din clasa „Foods” corespunzător mâncării detectate.

```

1 class Foods(Enum):
2     APPLE_PIE = {'name': 'apple_pie', 'calories': 237,
3                   'proteins': 2.4, 'fats': 11, 'carbs': 34.6}

```

Listing 3.1: Primul element din clasa de mâncăruri

Scriptul principal, care lansează serverul web, folosind Flask, este „app.py” (3.2). Inițial importăm bibliotecile și clasele necesare și creăm o instanță a clasei Flask, reprezentând aplicația web. Apoi, definim ruta, care acceptă request-uri de tip „POST” și funcția get_food, care gestionează solicitările. Aceasta primește un obiect json conținând imaginea, și, cu ajutorul funcției classify_image, din clasa ModelManager definită anterior, clasifică imaginea, returnând caracteristicile felului de mâncare detectat.

```

1 @app.route("/get_food", methods=['POST'])
2 def get_food():
3     data = request.json
4     mm = ModelManager()
5     food = mm.classify_image(data)
6     data = {
7         'message': 'OK',
8         'food': food
9     }
10    response = jsonify(data)
11    return response

```

Listing 3.2: Setarea rutei în Flask (fisierul app.py)

Pentru ca imaginea să poată fi prelucrată de către server, clientul trimite, pe ruta ”/get_food” a aplicației, poza încărcată de utilizator în pagina scan și, ulterior, va primi răspunsul, returnat de funcția get_food.

```
1 fetch('http://127.0.0.1:5000/get_food', {
2     method: 'POST',
3     headers: {
4         'Content-Type': 'application/json'
5     },
6     body: JSON.stringify(data)
7 })
```

Listing 3.3: Trimiterea datelor imaginii spre server în JavaScript

Capitolul 4

Concluzii și perspective

4.1 Limitări și probleme întâmpinate

Prinț-o poză 2d se poate extrage doar o cantitate limitată de informații. Pentru a putea deosebi, cu acuratețe mai mare, între două feluri de mâncare similare, ar trebui obținute mai multe imagini din unghiuri diferite asupra farfuriei/bolului cu mâncare sau o vizualizare 360 a mâncării (chiar și așa ar exista o limitare, deoarece nu se poate accesa informația din „adâncimea” imaginii).

Rețeaua neurală adâncă poate recunoaște doar cele 101 (sau 105 pentru modelul extins) clase de mâncare pe care a fost antrenată. Astfel, dacă un utilizator va încărca o imagine cu un fel de mâncare care nu este conținut în lista claselor, modelul va returna un răspuns eronat.

Fiecare fel de mâncare are un număr prestabilit de calorii și macronutrienți pe sută de grame. În realitate, pot exista variații ale unui dish, ceea ce poate produce modificări acestor valori.

O problemă întâmpinată a fost crearea setului de date cu mâncăruri românești. Este dificil de creat un set de date cu multe imagini de calitate, fără a fi generat artificial sau fără a conține poze nerealiste.

4.2 Îmbunătățiri și idei viitoare

O primă îmbunătățire ar fi extinderea setului de date și a modelului, pentru a conține aproape toate felurile de mâncare populare. În plus, un model mai complex ar fi benefic pentru a crește acuratețea clasificării.

O idee de îmbunătățire a paginii scan, din aplicația web, ar fi posibilitatea utilizatorului de a schimba numărul de calorii și macronutrienți pentru felul de mâncare detectat, în caz că acesta este inexact.

Un alt viitor feature al acestei platforme este crearea unei pagini de autentificare și

gestionare a contului, iar, în pagina de profil, posibilitatea de a introduce sau modifica datele personale. Însă, un utilizator ar trebui să poată accesa paginile home și scan fără a fi logat (modul guest). Pentru o persoană autentificată, va apărea un nou buton pe pagina de scan, numit „Add to Calendar”. Acesta permite utilizatorului să salveze cantitatea de calorii consumată la acea masă, după ce a submis gramajul. Caloriile vor fi adăugate la consumul zilnic, care va fi afișat în pagina calendar. Aceasta va expune un calendar dinamic, fiecare zi fiind hașurată cu o culoare, în funcție de cât de aproape este utilizatorul de target-ul setat pe home page. Va fi necesară o bază de date pentru a putea salva toate informațiile descrise anterior.

În concluzie, această lucrare îmbină un model de deep learning, având la bază o rețea neurală adâncă antrenată, și o aplicație web robustă și funcțională pe orice tip de dispozitiv. Astfel, prinde contur un asistent intelligent în nutriție, care ar putea îmbunătăți sănătatea fizică și mintală, precum și stilul de viață al oricărui utilizator.

Listă de figuri

2.1	Exemple de imagini din setul de date	7
2.2	Comparație între o arhitectură CNN cu 20 de straturi și una cu 56 de straturi	9
2.3	Bloc Rezidual (Conexiune skip)	10
2.4	Diferite versiuni ale arhitecturii ResNet	10
2.5	Arhitecturile VGG-19, „34-layer plain”, respectiv ResNet-34	11
2.6	Arhitectura modelului ResNet-50	11
2.7	Stânga: un Bloc Standard Rezidual pentru ResNet-34. Dreapta: un Bloc „Bottleneck” Rezidual pentru ResNet-50/101/150	12
2.8	Funcția de activare ReLU	13
2.9	Vizualizarea hărților de caracteristici (feature maps)	14
2.10	Acuratețea modelului pentru diferite funcții de activare	18
2.11	Loss-ul modelului pentru diferite funcții de activare	18
2.12	Vizualizarea overfitting-ului și underfitting-ului la antrenarea unui clasificator	19
2.13	K-Fold Cross Validation	20
2.14	Stânga: SGD fără Momentum. Dreapta: SGD cu Momentum	21
2.15	Acuratețea modelului ResNet-18 fine-tuned pe 15 epoci	26
2.16	Loss-ul modelului ResNet-18 fine-tuned pe 15 epoci	26
2.17	Acuratețea modelului ResNet-50 fine-tuned pe 12 epoci	27
2.18	Loss-ul modelului ResNet-50 fine-tuned pe 12 epoci	27
2.19	Acuratețea Top-1 și Top-2 pentru modelul ResNet-50 fine-tuned, testat pe imagini reale	28
2.20	Acuratețea modelului pentru cele 105 clase	29
2.21	Loss-ul modelului la antrenarea pe cele 105 clase	30
3.1	Bara de navigare	32
3.2	Primul slide de pe pagina home	32
3.3	Al doilea slide de pe pagina home	33
3.4	Stânga: Butonul de Upload. Dreapta: Butonul de Upload atunci când mouse-ul se află deasupra sa (hover)	33
3.5	Pagina Scan după ce se încarcă o imagine	34
3.6	Meniul dropdown, când lățimea paginii este mai mică de 1000 de pixeli . .	35

3.7 Paginile Home, respectiv Scan, atunci când mărimea ecranului este de 500 pixeli x 1000 pixeli	36
---	----

Bibliografie

- [1] Abien Fred Agarap, *Deep Learning using Rectified Linear Units (ReLU)*, 2019, arXiv: [1803.08375 \[cs.NE\]](https://arxiv.org/abs/1803.08375).
- [2] Abid Ali Awan, *A Complete Guide to Data Augmentation*, 2022, URL: <https://www.datacamp.com/tutorial/complete-guide-data-augmentation>.
- [3] Lukas Bossard, Matthieu Guillaumin și Luc Van Gool, „Food-101 – Mining Discriminative Components with Random Forests”, în *European Conference on Computer Vision*, 2014, URL: https://data.vision.ee.ethz.ch/cvl/datasets_extra/food-101/static/bossard_eccv14_food-101.pdf.
- [4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li și Li Fei-Fei, „ImageNet: A large-scale hierarchical image database”, în *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255, DOI: [10.1109/CVPR.2009.5206848](https://doi.org/10.1109/CVPR.2009.5206848).
- [5] Pytorch documentation, *Tensor Normalization with Torchvision Transforms*, URL: <https://pytorch.org/vision/main/generated/torchvision.transforms.Normalize.html>.
- [6] Shiv Ram Dubey, Satish Kumar Singh și Bidyut Baran Chaudhuri, *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*, 2022, arXiv: [2109.14545 \[cs.LG\]](https://arxiv.org/abs/2109.14545).
- [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren și Jian Sun, *Deep Residual Learning for Image Recognition*, 2015, arXiv: [1512.03385 \[cs.CV\]](https://arxiv.org/abs/1512.03385).
- [8] Sergey Ioffe și Christian Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015, arXiv: [1502.03167 \[cs.LG\]](https://arxiv.org/abs/1502.03167).
- [9] Diederik P. Kingma și Jimmy Ba, *Adam: A Method for Stochastic Optimization*, 2017, arXiv: [1412.6980 \[cs.LG\]](https://arxiv.org/abs/1412.6980).
- [10] Alex Krizhevsky, Ilya Sutskever și Geoffrey E Hinton, „ImageNet Classification with Deep Convolutional Neural Networks”, în *Advances in Neural Information Processing Systems*, ed. de F. Pereira, C.J. Burges, L. Bottou și K.Q. Weinberger, vol. 25, Curran Associates, Inc., 2012, URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

- [11] Ilya Loshchilov și Frank Hutter, *Decoupled Weight Decay Regularization*, 2019, arXiv: [1711.05101 \[cs.LG\]](https://arxiv.org/abs/1711.05101).
- [12] *Normalization (image processing)* — Wikipedia, The Free Encyclopedia, 2024, URL: [https://en.wikipedia.org/wiki/Normalization_\(image_processing\)](https://en.wikipedia.org/wiki/Normalization_(image_processing)) #References.
- [13] Keiron O'Shea și Ryan Nash, *An Introduction to Convolutional Neural Networks*, 2015, arXiv: [1511.08458 \[cs.NE\]](https://arxiv.org/abs/1511.08458).
- [14] *Optimizers in Deep Learning*, 2021, URL: <https://musstafa0804.medium.com/optimizers-in-deep-learning-7bf81fed78a0>.
- [15] Petru Potrimba, *What is ResNet-50?*, 2024, URL: <https://blog.roboflow.com/what-is-resnet-50/#:~:text=ResNet%2D50%20is%20CNN%20architecture,efficiency%20in%20image%20classification%20tasks..>
- [16] *ResNet-50 Pretrained Model Syntax in PyTorch*, URL: <https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html>.
- [17] Karen Simonyan și Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2015, arXiv: [1409.1556 \[cs.CV\]](https://arxiv.org/abs/1409.1556).
- [18] *Stochastic Gradient Descent (SGD)*, 2024, URL: <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>.
- [19] Juan Terven, Diana-Margarita Cordova-Esparza, Alfonzo Ramirez-Pedraza și Edgar Arturo Chávez Uriola, *Loss Functions and Metrics in Deep Learning. A Review*, 2023, URL: https://www.researchgate.net/publication/372163006_Loss_Functions_and_Metrics_in_Deep_Learning_A_Review.
- [20] *Transfer learning*, 2024, URL: https://en.wikipedia.org/w/index.php?title=Transfer_learning&oldid=1227031258.