UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

COORDONATOR ȘTIINȚIFIC

Conf. dr. Radu Boriga

ABSOLVENT

Cristina Damov

BUCUREȘTI

Iunie, 2024

UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

SOLUȚIE DE CHECKOUT CU VEDERE ARTIFICIALĂ

COORDONATOR ȘTIINȚIFIC
Conf. dr. Radu Boriga

ABSOLVENT
Cristina Damov

BUCUREȘTI

Iunie, 2024

REZUMAT

Proiectul meu propune o soluție inovatoare pentru eficientizarea procesului de checkout în mediile comerciale, utilizând recunoaștere vizuală pentru identificarea rapidă și precisă a produselor. Prin integrarea unui model de învățare automată, ResNet50, acesta recunoaște diverse produse direct din imagini capturate în timp real. Soluția permite detectarea obiectelor de asemenea în cazul ambalajelor deteriorate sau codurilor de bare ilegibile, optimizând astfel fluxul de lucru la casele de marcat și reducând timpul de așteptare pentru clienți. Implementarea acestui sistem nu doar accelerează procesul de plată, ci îmbunătățește totodată experiența de a face cumpărături prin minimizarea erorilor de scanare și oferind o interfață prietenoasă și ușor de utilizat. Prin adoptarea acestei soluții, magazinele pot atinge o eficiență sporită și oferă satisfacție crescută a clienților.

ABSTRACT

My project proposes an innovative solution for streamlining the checkout process in commercial environments by using visual recognition to rapidly and accurately identify products. By integrating a machine learning model, ResNet50, this method recognizes various products directly from images captured in real time. This solution allows for the detection of objects even in case of damaged packaging or illegible barcodes, thus optimizing the workflow at checkout counters and reducing waiting times for customers. Implementing this system not only speeds up the payment process but also enhances the shopping experience by minimizing scanning errors and offering a friendly and easy-to-use interface. By adopting this solution, stores can achieve increased efficiency and enhance customer satisfaction.

Cuprins

INTRODUCERE

Implementarea tehnologiei a transformat în mod semnificativ industria de retail, de la operațiunile logistice, până la experiența consumatorilor. Un punct de contact major între o entitate de retail și client este procesul final de la casa de marcat. Introducerea codurilor de bare în anii '70 a revoluționat modul în care produsele sunt gestionate și vândute, fiind o modalitate simplă de a transforma informația obiectelor într-un format ce poate fi integrat cu tehnologia. În ciuda adoptării pe scară largă, sistemul codurilor de bare se bazează pe o tehnologie datând câteva decenii, nefiind modernizată în tandem cu statu-quo-ul tehnologic.

Această lucrare descrie dezvoltarea și implementarea unei soluții bazate pe vederea artificială, înlocuind codurile de bare pentru a facilita utilizarea caselor de marcat. Sistemul propus folosește algoritmi de învățare automată pentru a recunoaște și identifica produsele plasate pe o masă de plată. Astfel, proiectul își propune să îmbunătățească toate aspectele unui proces de plată, inclusiv experiența clienților și reducerea costurilor operaționale ale comercianților.

Clienții își formează percepția asupra calității sistemelor de auto-servire considerând cinci factori principale: viteză, ușurință în utilizare, control, fiabilitate și plăcere (Fernandes, 2017). Soluția propusă in această lucrare este concepută să satisfacă acești factori, integrând posibilitatea de a fi extinse și îmbunătățite în continuare.

Motivări

Numeroase aspecte au contribuit la inițiativa de a dezvolta o soluție pentru casele de marcat care nu se bazează pe coduri de bare:

1. Optimizarea experienței clienților: cumpărătorii moderni doresc un proces simplu si rapid. Experiența negativă poate apărea în urma unui proces îndelungat de scanare a articolelor, coada lungă sau alte întârzieri neprevăzute. De asemenea, o mare parte din clienții supermarketurilor preferă soluțiile de self-service, fără interacțiunea umană, iar fluidizarea procesului va crea o imagine favorabilă în ochii utilizatorilor. (Collier, Only If It Is Convenient: Understanding How Convenience Influences Self-Service Technology Evaluation, 2013).

2. Creșterea eficienței operaționale: Gestionarea liniilor lungi, rezolvarea problemelor legate de codurile de bare și reducerea erorilor umane sunt unele dintre provocările operaționale cu care se confruntă supermarketurile și unitățile de vânzare în amănunt. Scăzând nevoia de scanare umană, sistemul de casă fără coduri de bare poate simplifica procesele și poate crește productivitatea la casele de marcat.
3. Facilitarea supervizării clienților: În cadrul sistemelor de self-service, comercianții au devenit nevoiți să supravegheze procesul clienților pentru a diminua potențiale fraude sau erori. Simplificarea acțiunilor făcute de cumpărători poate simplifica la rândul lor și sarcinile cadrului de control.
4. Reducerea erorilor umane: Scanările omise, identificarea inexactă a produsului și discrepanțele de preț se numără printre greșelile care pot apărea atunci când codurile de bare sunt scanate manual. Prin identificarea corectă a produselor pe baza atributelor lor vizuale, o soluție bazată pe inteligență artificială reduce aceste erori și garantează o gestionare adecvată a prețurilor și a stocurilor.
5. Adaptarea la avansurile tehnologice: Implementarea unor soluții noi în domeniul comerțului poate crea imaginea unui supermarket inovativ și centrat pe client, sporind reputația în contextul unei piețe competitive. Soluția propusă ar fi totodată o modalitate de automatizare care poate produce o schimbare semnificativă în sectorul retail, la scară largă.
6. Timpul redus: Tehnologia permite identificarea rapidă și precisă a articolelor cumpărate, eliminând nevoia de scanare manuală a codurilor de bare și reducând astfel timpul petrecut la coadă de către clienți. Creșterea vitezei cu care este terminat procesul de plată este benefică atât pentru clienți, cât și pentru retaileri, care pot procesa tranzacții mult mai rapid.
7. Depășirea problemelor de scanare: Sistemele de scanare clasică pot întâmpina probleme cu codurile de bare nelizibile. Fie că este vorba de un cod de bare murdar, zgâriat sau deteriorat în alt mod, iluminare deficitară, o suprafață distorsionată, sau un material reflectorizant care interferează cu citirea, scanerul poate avea dificultăți de detectare. Identificarea produsului prin vederea artificială poate depăși aceste probleme, nedepinzând de codul de bare.

Implementare

Funcționarea soluției de checkout propusă implică câteva componente cheie:

- Componenta hardware: o cameră cu rezoluție standard este instalată deasupra mesei de la casa de marcat, capturând imagini ale produselor care urmează să fie identificate și achiziționate. Procesarea imaginilor este efectuată în timp real.
- Integrarea cu un card reader: proiectul curent cuprinde întreg procesul, de la selecționarea produselor până la realizarea coșului final virtual; în viitor este nevoie de coordonarea cu dispozitivul de plată cu cardul (POS).
- Conexiunea la internet: sistemul necesită o conexiune stabilă la internet pentru procesarea datelor în timp real și sincronizarea coerentă a informațiilor despre produse.

CAPITOLUL 1. IDENTIFICAREA PRODUSELOR

1.1. Setul de date

În prima fază am construit un set de date potrivit lucrării. În lipsa unuia compatibil cu forma proiectului, a fost creat de la zero. Am ales 14 produse existente în orice supermarket, astfel încât să ofere un ansamblu generic de produse pe care le-ar cumpăra un client într-o situație reală, dar și încât să testeze posibile cazuri care ar crea impedimente în performanța finală a soluției.

	id [PK] integer	name character varying (50)	price integer	description character varying (255)
1	1	alpro-milk	18	vegetal unsweetened milk from almonds
2	2	beans	6	[null]
3	3	chips	13	salt flavored
4	4	corn	6	[null]
5	5	gummies	7	haribo sour fizz
6	6	knoppers	3	[null]
7	7	lemon	2	[null]
8	8	lime	2	[null]
9	9	milk	7	cow milk 3%
10	10	milka-brownie	8	[null]
11	11	milka-hazelnut	8	[null]
12	12	orange	3	[null]
13	13	oreo	7	6-pack
14	14	water	4	[null]

Figure 1. Tabelul „Products” din baza de date

Pentru cele 14 clase de produse am realizat imagini individuale din diferite unghiuri pentru antrenarea clasificării, dar și imagini cu produse mixte pentru a simula posibile coșuri de cumpărături într-un magazin real. În total au fost introduse 472 de imagini cu produse individuale și 390 de imagini mixte.

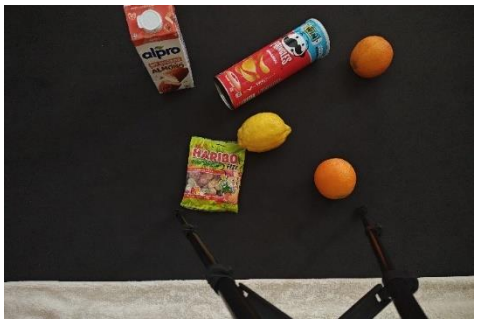


Figure 2. Exemplu imagine cu mai multe produse

1.2. Detectarea Obiectelor

Introducerea unui model eficient de detectare a obiectelor în timp real este esențială pentru succesul proiectului, având în vedere nevoia de a procesa și analiza rapid imagini pentru a identifica produsele diverse dintr-un supermarket. Astfel, utilizarea unui sistem de detectare a obiectelor în timp real nu este doar o opțiune, ci o necesitate. Modelul trebuie să fie capabil să opereze cu latență minimă și să ofere rezultate precise chiar și în condiții variabile de iluminare sau acoperire parțială a obiectelor. Modelul ales trebuie să fie robust și adaptabil, apt să se descurce cu scenarii dinamice și să învețe din date noi într-un mod eficient pentru a rămâne relevant și performant pe măsură ce condițiile de operare se schimbă. Pentru cerințele menționate anterior, modelul state-of-the-art aparține din seria YOLO (You Only Look Once).

Pentru dezvoltarea soluțiilor moderne de vizualizare computerizată, selecția unei platforme robuste pentru prelucrarea și antrenarea datelor este vitală. RoboFlow este o astfel de platformă avansată care facilitează preprocesarea, augmentarea și gestionarea seturilor de date pentru aplicații de învățare automată. RoboFlow se distinge prin capacitatea sa de a optimiza seturi de date pentru diferite modele de detectare a obiectelor, inclusiv pentru arhitectura YOLO, recunoscută pentru eficiența în detectarea rapidă și precisă a obiectelor în imagini. Utilizarea acestei platforme permite adaptarea flexibilă a datelor, creșterea acurateței modelului și accelerarea ciclului de dezvoltare, oferind o integrare adaptabilă la sistemele existente.

Antrenarea modelului pe dataset-ul propriu prin RoboFlow a contribuit semnificativ la îmbunătățirea capacității de recunoaștere și clasificare a proiectului. Modelul YOLO, renumit pentru viteza sa de procesare în timp real, este configurat să identifice diverse obiecte din imagini, utilizând un set de date etichetat și augmentat prin RoboFlow. Platforma oferă posibilitatea de a personaliza augmentarea, prin diverse opțiuni, cum ar fi rotirea, redimensionarea sau decuparea aleatorie, care ajută la crearea unui model performant. Integrarea RoboFlow în procesul de dezvoltare nu numai că optimizează antrenarea datelor, dar asigură și o potențial de scalare crescut, adaptând modelul la nevoile specifice ale proiectului.

După încărcarea setului de date pe RoboFlow, fiecare poză a fost adnotată manual pentru a crea casetele delimitatoare („bounding boxes”) corecte, iar setul final a fost împărțit folosind raportul standard de 70% training, 20% validation, 10% testing. Fiind compus din imagini create strict pentru acest proiect, nu a fost necesară o preprocesare majoră. Singura modificare necesară

a fost decuparea statică a imaginilor cu raportul de 0-80% pe axa verticală, pentru a elimina o porțiune a imaginii care nu aparține zonei cu produse. Fără preprocesare, modelul de identificare are o rată de eroare mai mare decât dorită, detectând într-o serie de imagini „obiecte fantomă” (în zona selectată nu există un produs) la limita dintre zona cu produse (fundal negru) și zona fără produse (fundal alb).

Pentru augmentare a fost folosită funcția de eliminare („cutout”) care introduce în imagini chenare negre amplasate aleatoriu, cu scopul de a crește acuratețea în situațiile în care un produs are o parte acoperită de alt produs. A fost testată și funcția de răsturnare („flip”) pe axa verticală și axa orizontală, însă nu a fost folosită pentru modelul final, cu speranța că modelul va reține suportul vizibil în cadrul camerei și nu îl va considera drept potențial obiect.

Figure 3. Metricile modelului de detectare a obiectelor

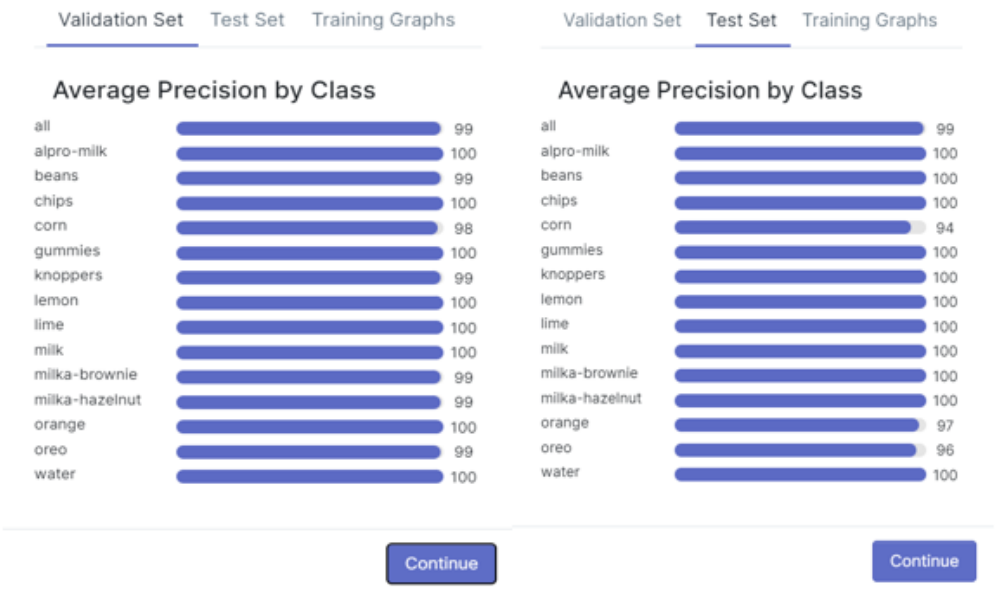


Figura 4. Precizia modelului de detectare a obiectelor în etapele de validare și testare

1.3. Algoritmul de clasificare

Din setul de date au fost selectate imaginile în care apar produsele individuale din diferite unghiuri, decupate strict pe produs și împărțite în funcție de clasă. În continuare, eșantionul a fost împărțit aleatoriu în 70% train, 20% validation și 10% test, iar batch size-ul pentru construirea claselor de DataLoader a fost 8 (ales în urma testării mai multor mărimi). Fiind un dataset propriu, construit specific pentru task-ul proiectului, nu au fost necesare alte preprocesări ale imaginilor.

Figura 5. Imagini de antrenare pentru clasa „corn” (porumb)

Diferența dintre numărul de imagini pentru fiecare clasa nu a fost una radicală, încât nu au apărut bias-uri în modelul final de identificare a produselor.

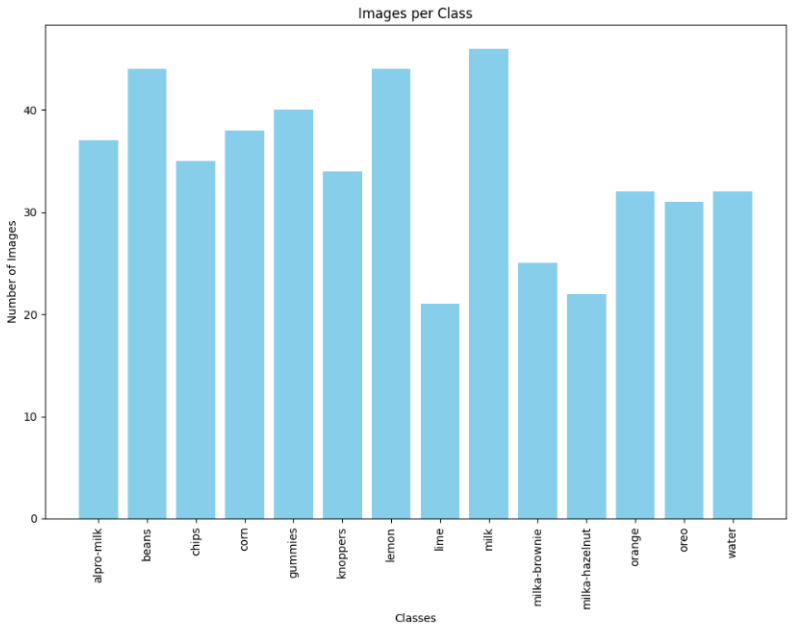


Figura 6. Raport imagini per clasă pentru antrenarea modelului de clasificare

Pentru alegerea algoritmului potrivit de clasificare a imaginilor, am consultat literatura de specialitate, orientându-mă către studii cu un subiect similar și care compară performanța a diferiți algoritmi eficienți în contextul recunoașterii vizuale. În urma evaluării a mai multor arhitecturi de rețele neuronale, printre care și „A Comparative Study of Different CNN Models and Transfer Learning Effect for Underwater Object Classification in Side-Scan Sonar Imagest" (Du, 2023), GoogleNet și ResNet s-au dovedit a fi superioare în termeni de acuratețe. Acestea se diferențiază față de celelalte modele prin acuratețea lor, însă diferența dintre rezultatele acestora nu era una semnificativă. Am ales să compar aceste două arhitecturi în contextul lucrării mele pentru a decide care este cel mai potrivit, nu doar din constatările literaturii de specialitate, ci și din observații empirice.

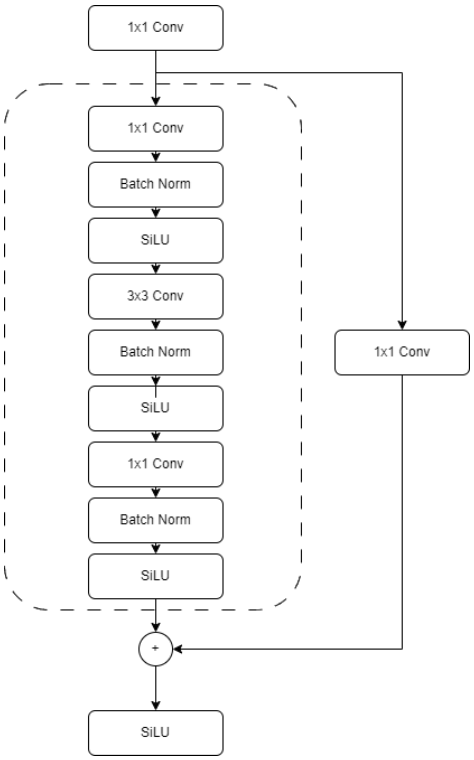
CNN Models	Training Accuracy (%)	Test Accuracy (%)
AlexNet	96.68	86.67
VGG-16	99.68	85.19
GoogleNet	100	94.81
ResNet101	100	90.37

Figura 7. Comparație a acurateței de predicție în diferite modele CNN (Du, 2023)

În cadrul arhitecturilor ResNet, sunt cunoscute mai multe arhitecturi în funcție de numărul de straturi (ex: ResNet-34, ResNet-50, ResNet 152). Numărul ridicat de straturi poate crea însă un procent ridicat de erori, din cauza mai multor motive, printre care se numără degradarea modelului, dispariția gradientului, probleme de optimizare, parametri diminuați în straturile suplimentare și overfitting (modelul retine datele de antrenament si ii scade capacitatea de a raspunde corect datelor noi). Pentru contextul proiectului, un model cu 101 straturi ar fi prea complex pentru dataset, așadar am optat pentru arhitectura cu 50 de straturi.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 8. Arhitecturi ResNet (He, 2015)




```
layer1 = add_block(64, 256)          x3
layer2 = add_block(128, 512)         x4
layer3 = add_block(256, 1024)        x6
layer4 = add_block(512, 2048)        x3
```

Figura 9. Structura blocului ResNet

În mod similar, arhitectura pentru GoogleNet a fost simplificată, eliminând unele straturi auxiliare și funcția de dropout, pentru a scădea complexitatea modelului la un nivel ajustat proiectului.

GoogleNet și ResNet au atins ambele performanța state-of-the-art în problemele de clasificare a imaginilor; prin urmare, pentru a le compara trebuie luate în considerare mai multe puncte de vedere. Complexitatea modelului GoogleNet este redată de modulele Inception și straturile convoluționale paralele multiple. Asta face modelul mai eficient în general, însă poate fi mai greu de antrenat față de ResNet, în special pe seturile de date mici, unde poate avea loc problema de overfitting. Arhitectura simplă a ResNet-ului se datorează conexiunilor reziduale, facilitând antrenarea pe rețele adânci; așadar, se potrivește mai bine cu seturi de date reduse și în situații cu resurse computaționale reduse. ResNet este mai stabil de antrenat față de GoogleNet, care necesită un tuning al hiperparametrilor prudent. În consecință, deși în alte studii GoogleNet pare un model mai performant decât ResNet, am decis să îl includ în comparațiile făcute pentru alegerea modelului final datorită simplității, ușurința de a antrena cu resurse limitate și a acurateței ridicate pe seturile de date mai puțin complexe.

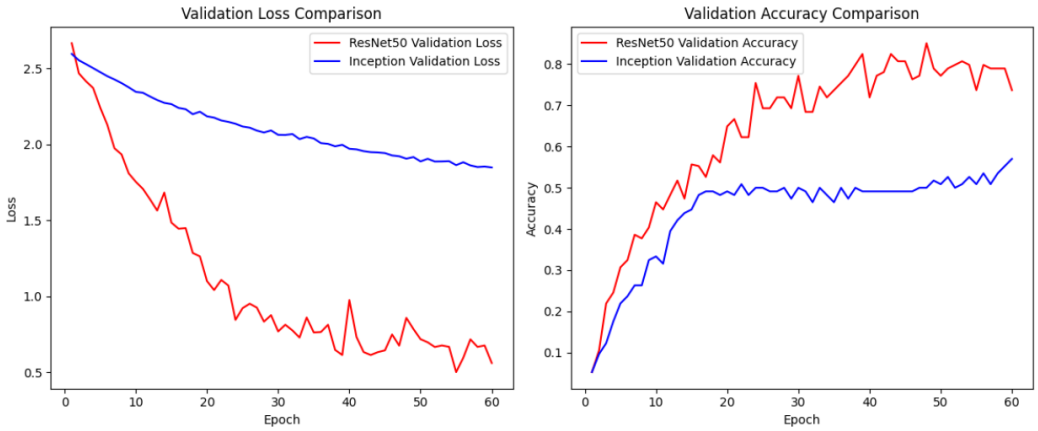


Figura 10. Compararea modelelor ResNet și Inception (funcția de pierdere și acuratețe)

În concluzie, modelul ResNet a avut o acuratețe mai bună decât GoogleNet pentru tema actuală.

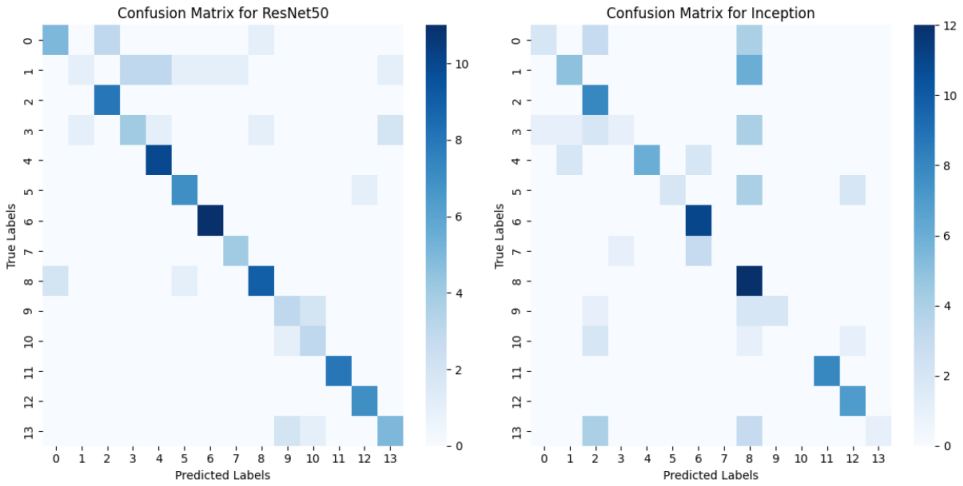


Figure 11. Matricea de confuzie a modelelor de clasificare comparate

O caracteristică extrem de benefică a rețelelor ResNet este capacitatea de învățare incrementală, care permite extinderea cu clase noi, fără a fi necesară reantrenarea modelului de la zero. Actualizările periodice ale inventarului de produse, prezente într-un sistem comercial obișnuit, pot fi optimizate prin această abordare, economisind timp și resurse computaționale semnificative.

1.4. Hiperparametrizare

Hiperparametrizarea se referă la parametrii stabiliți înainte de a începe antrenarea unui model. Selectarea parametrilor finali a fost făcută prin experimentarea cu diverse valori și alegerea rezultatelor cu o performanță superioară. Optimizarea a fost esențială preciziei modelului final, crescând acuratețea de la 65% la 94% (în etapa de validare).

Batch size-ul controlează numărul de exemple utilizate într-o iterație (forward/backward pass). Cu cât este mai ridicat numărul, cu atât va fi mai mult spațiu de memorie folosit. Alegerea parametrului influențează echilibrul viteza-acuratețe al antrenării modelului. În general, se folosesc numere care sunt puteri de 2, iar o alegere standard este 32. După încercarea mai multor mărimi, proiectul a dovedit acuratețe mai bună pentru batch size = 8.

Numărul de epoci se referă la o trecere completă a algoritmului de învățare prin întregul dataset de antrenare. Pe parcursul unei epoci, fiecare imagine din setul de training este introdusă în rețea și sunt ajustate weight-urile pentru îmbunătățirea clasificărilor. Pentru ca modelul să conveargă către soluția optimă sunt necesare mai multe epoci, însă un număr prea ridicat poate

cauza problema de overfitting. Pe parcursul antrenării modelului s-a putut observa un prag în jurul epocii 60, unde acuratețea nu mai creștea, așa că 60 a fost numărul de epoci ales pentru partea de antrenare.

Rata de învățare (learning rate) este unul dintre parametrii esențiali în antrenarea rețelelor neuronale și algoritmii de machine learning. Aceasta determină modul în care modelul actualizează weight-urile în procesul de backpropagation. O valoare mică poate însemna o convergență lentă, iar o valoare prea mare poate afecta rezultatul final prin oscilații și instabilitate în jurul minimului sau chiar divergența modelului. Experimentarea cu mai multe valori de learning rate a avut un rol important și în gestionarea minimurilor locale, fiind ales în final valoarea 10^-5.

O altă strategie de a gestiona rata de învățare și evitarea minimurilor locale a fost alegerea unui optimizator adaptiv, care adaptează rata de învățare pe baza istoricului gradientilor. Soluția folosește Adam, care nu necesită ajustări manuale frecvente ale ratei de învățare, spre deosebire de SGD sau alte metode. Alegerea a fost făcută și pe baza testării a mai multor metode de optimizare.

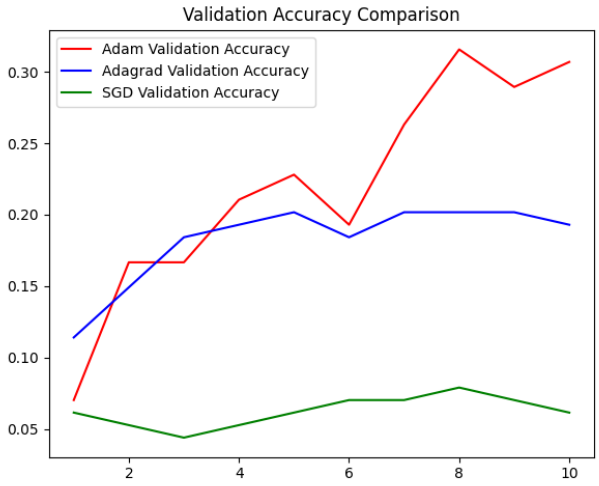


Figure 12. Compararea optimizatorilor

Funcția de activare determină modul în care datele de intrare sunt transformate de către rețele neuronale înainte de a fi transmise la următorul strat al rețelei. Aceasta este esențială în introducerea non-liniarității în model, ceea ce permite rețelei să învețe și să modeleze relații complexe și abstracte din date. Alegerea funcției de activare adecvate depinde de mai mulți factori, inclusiv de natura problemei, de arhitectura rețelei și de tipul datelor de intrare, având un

impact semnificativ asupra performanței acestuia. Selecția a fost făcută în urma testării majorității funcțiilor de activare valabile în biblioteca Torch, dar cu tendința inițială de a alege SiLU. Funcția de activare SiLU (Sigmoid-weighted Linear Unit), propusă ca o alternativă superioară la ReLU, combină proprietățile funcțiilor de activare sigmoid și linear, permițând modelelor să capteze atât non-linearități complexe cât și să mențină o eficiență computațională adecvată. ReLU este mai simplu din punct de vedere al operațiilor computaționale comparativ cu SiLU. Deși ReLU este mai simplu din punct de vedere computațional, SiLU poate oferi o performanță mai bună prin utilizarea mai eficientă a accelerării hardware și prin capacitatea sa de a învăța modele mai complexe. Contrar presupunerilor, performanța funcției ReLU a depășit așteptările și a oferit o acuratețe finală mai ridicată modelului decât SiLU.

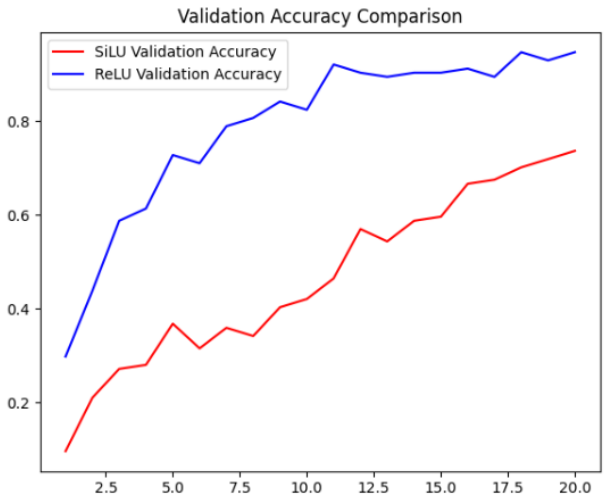


Figure 13. Compararea funcțiilor de activare SiLU si ReLU

Pentru funcția de pierdere a fost folosit Cross-Entropy Loss, fiind cea mai eficientă metodă pentru clasificarea seturilor de date cu mai multe clase. Deoarece penalizează predicțiile greșite cu probabilități mari, contribuie la stabilizarea rapidă a procesului de antrenare.

1.5. Analiza predicțiilor

La finalul dezvoltării aplicației, pe lângă metricele de acuratețe, au fost testate diverse scenarii care pot apărea în contextul implementării reale a soluției în industria de retail și care pot confunda procesul de predicție al modelului. Scopul a fost de a construi o soluție robustă și stabilă împotriva greșelilor care pot apărea, intenționat sau neintenționat, într-un context real al unui supermarket. Mai jos, voi descrie metoda de testare și rezultatul obținut.

1. Apropierea produselor

Pentru a testa capacitatea modelului de a distinge produsele aflate la distanțe mici, am redus spațiul dintre ele și am plasat inclusiv produse identice unele lângă altele. Inițial, rata de eroare a fost ridicată, însă a fost remediată prin includerea în dataset conținând mai multe imagini cu această situație.

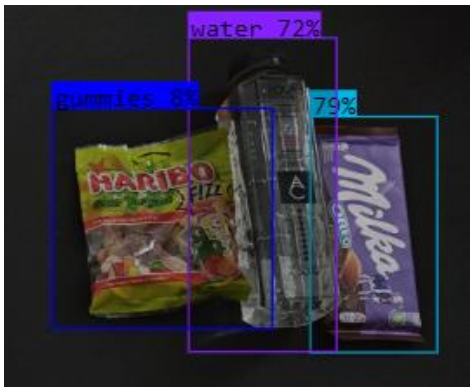


Figura 14. Exemplu imagine - produse apropiate

2. Compararea produselor asemănătoare

În contextul procesului de scanare poate apărea frecvent problema similarității, cauzată de ambalajul asemănător între produsele de același tip, dar cu arome diferite. Aceeași dificultate poate fi întâmpinată și la produse diferite, dar împachetate la fel, cum ar fi forma standard a unei conserve. Am selectat produse care se încadrează acestei descrieri și am urmărit bias-ul care poate apărea. Deși în urma creșterii acurateței modelului au fost eliminate multe erori de acest tip, încă există posibilitatea unui „unghi mort” în care nu pot fi identificate produsele din imagine nici de către o persoană. În cazurile respective în care nu sunt identificate caracteristici specifice unei clase, încă se poate observa un bias către un produs, însă este de neevitat.

Figura 15. Exemplu imagine - ambalaje asemănătoare

3. Suprapunerea produselor

Modelul a fost testat și în situațiile în care un obiect este ascuns parțial de alt obiect. Plasarea intenționată a unui produs peste altul într-un mod în care să blocheze vizibilitatea etichetei sau a trăsăturilor distincte a fost una dintre cele mai provocatoare teste pentru modelul de clasificare, care a fost antrenat în mod majoritar cu poze clare, în care se putea observa integral produsul. Rata de identificare a scăzut, însă problema principală a fost dacă modelul de detectare a obiectelor putea transmite o imagine decupată corect care să includă doar produsul respectiv.

Figura 16. Exemplu imagine - produse suprapuse

4. Deformarea produselor

Pentru a verifica răspunsul modelului în cazul produselor care suferă alterări fizice sau modificări ale formei obișnuite, am inclus în setul de test obiecte deformate intenționat, cum ar fi ambalaje strivite care nu își păstrează forma originală. Contrar așteptărilor, forma diferită nu a fost un impediment în identificarea produselor și nu au existat confuzii.

Figura 17. Exemplu imagine - produs deformat

În urma testării modelului, și a predicțiilor în diverse situații posibile într-un cadru real, acesta s-a dovedit a avea un grad ridicat de precizie și stabilitate. Prin introducerea a mai multor imagini de antrenare acesta a trecut peste provocările inițiale, însă unele situații încă se bazează pe cooperarea persoanelor care folosesc aplicația, fiind imposibil de rezolvat, chiar și cu intervenția umană, precum obstrucționarea parțială sau totală produsului într-un mod imposibilde recunoscut din perspectiva camerei.

CAPITOLUL 2. APLICAȚIA DE CHECKOUT

2.1. Tehnologii folosite

Putem clasifica tehnologiile folosite în 3 categorii: backend, frontend și modelul de predicție. Dependențele si librăriile specifice proiectului au fost incluse într-un environment pentru portabilitate, organizare, dar și izolarea față de alte proiecte (evitarea conflictelor de versiuni).

2.1.1. Backend

Pentru backend am folosit Flask, un framework popular scris în Python, folosit pentru dezvoltarea aplicațiilor web. Fiind minimalist și flexibil, Flask este recomandat proiectelor mici, oferind posibilitatea de modularizare și mentenanță ușoară. Acesta ne permite să cream rapid un server și să definim funcționalitățile aplicației.

Docker este o platformă de containerizare a aplicațiilor. Tehnologia este adesea folosită pentru dezvoltarea, testarea și implementarea soluțiilor, într-un mod eficient și scalabil. Un container este non-persistent, de fiecare dată când este rulat creează o nouă imagine a serviciului respectiv. Acesta este un format standard care conține toate dependențele, codurile, setările și tot ce mai este necesar unui software, eliminând problemele care apar odată cu rularea pe alte medii. Utilizarea tehnologiei Docker s-a demonstrat eficientă în crearea de medii izolate și stabile, remarcându-se valoarea nu numai pentru dezvoltare, cât și pentru scenariile de implementare. Aceasta asigură că aplicația este replicabilă, scalabilă și gata de rulare pe alte medii, cu o configurație minimă. Comenzile simple (docker build + docker run) au asigurat o construire și rulare rapidă a aplicației pe parcursul dezvoltării acesteia, fără proceduri complexe de configurare.

Baza de date foloseste PostgreSQL, un sistem de baze de date relaționale open-source, flexibil, dar și stabil, care poate fi integrat cu diverse limbaje de programare. Postgres nu a fost instalat local, ci a fost rulat cu ajutorul unei imagini Docker. Pentru accesarea bazei de date am introdus librăria SQLAlchemy, cu ajutorul căreia am creat conexiunea la baza de date.

```
project>docker pull postgres:latest
latest: Pulling from library/postgres
Digest: sha256:1bf73ccae25238fa555100080042f0b2f9be08eb757e200fe6afc1fc413a1b3c
Status: Image is up to date for postgres:latest
docker.io/library/postgres:latest
```

2.1.2. Frontend

Partea de frontend este standard, combinând HTML, CSS și Javascript pentru crearea unei interfețe intuitive utilizatorilor. Structura layoutului este definită de HTML, stilizată apoi de CSS pentru aspectul vizual plăcut. Prin Javascript cream posibilitatea de a comunica cu backend-ul și oferim funcționalitate interfeței.

2.1.3 Modelul de predicție

Algoritmul de identificare a produselor este o componentă esențială lucrării, fiind responsabil de procesarea imaginilor și clasificarea lor.

Pentru detectarea obiectelor dintr-o imagine este folosit un API (Application Programming Interface) Roboflow, unde a fost antrenat setul de date prin YOLOv8, state-of-the-art pentru real-time object detection. Pentru antrenarea modelului de clasificare, am apelat la resursele computaționale disponibile pe platforma Kaggle.

Modelul de clasificare a produselor găsite utilizează PyTorch, o librărie de machine learning pentru dezvoltarea și antrenarea modelelor de deep learning. Torchvision face parte din PyTorch, conceput pentru a susține soluțiile de computer vision, simplificând procesul de lucru prin instrumentele și utilitățile deținute. Numpy ajută prin abilitățile de procesare numerică, iar PIL (Pillow) facilitează procesarea imaginilor. Matplotlib a făcut mai ușoară compararea diverselor modele încercate și acuratețea lor, prin opțiunile de vizualizare a datelor.

2.2. Fluxul de lucru al aplicației

Capitolul acesta oferă o imagine completă asupra structurii sistemului dezvoltat, explicând modul în care componentele interacționează între ele pentru a îndeplini cerințele funcționale și non-funcționale ale aplicației și modul în care contribuie la realizarea obiectivelor proiectului. Aplicația adoptă arhitectura bazată pe microservicii, ceea ce implică descompunerea funcționalităților în servicii mai mici, independente, fiecare rulând separat și comunicând cu celelalte componente prin call-uri HTTP. În soluția curentă sunt trei containere: aplicația principală, modelul de clasificare și baza de date. Acest model de arhitectură oferă flexibilitate, facilitează mentenanța și susține scalarea, fără a afecta integral aplicația.

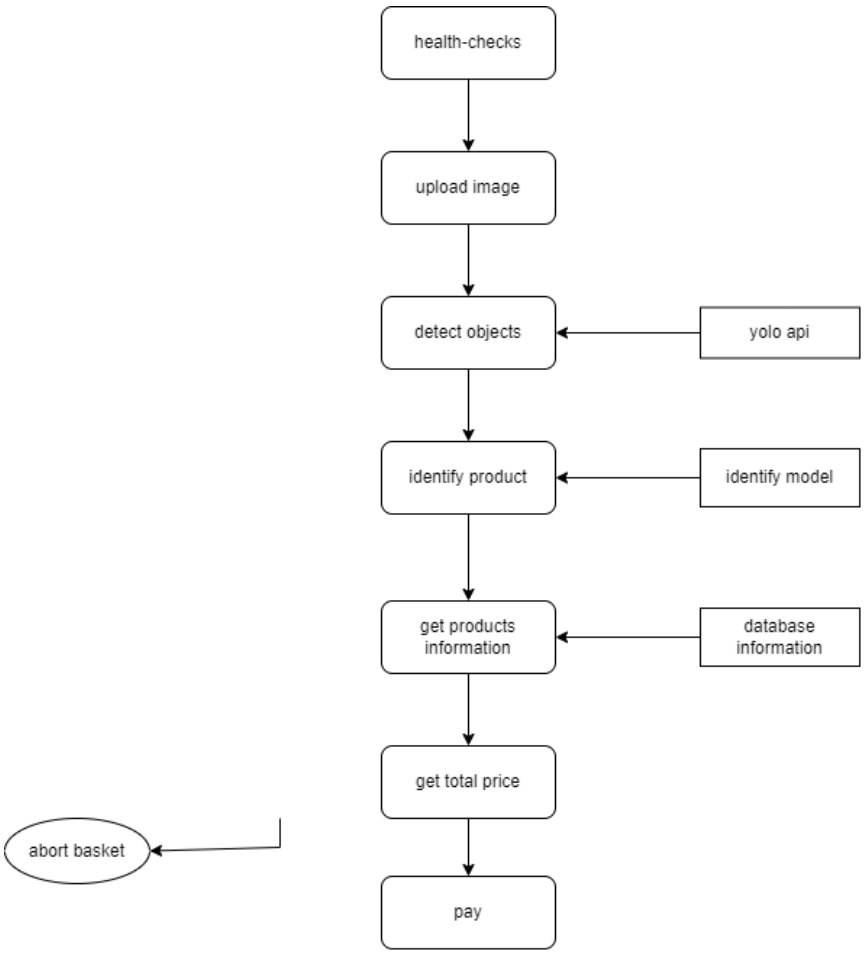


Figura 18. Diagrama cu flow-ul aplicației

Prima acțiune a aplicației este metoda de „health check”, un procedeu comun folosit în aplicațiile cu microservicii pentru a monitoriza și verifica starea și disponibilitatea componentelor sistemului. Health check-urile sunt esențiale pentru a asigura funcționarea corectă și disponibilitatea aplicației. Înainte de orice request al aplicației, fără a fi necesară o apelare explicită, va fi folosită metoda „check_service_health()”. Aceasta apelează fiecare endpoint HTTP expus al serviciilor utilizate („/health-check”) care returnează codul de stare HTTP 200, dacă conexiunea există și serviciul funcționează obișnuit, sau codul de eroare 500 în caz contrar. Mecanismul asigură că aplicația funcționează eficient, minimizând downtime-ul și ajutând la sesizarea problemelor într-un timp cât mai scurt.

```python
@app.before_request
def check_service_health():
    try:
        response = requests.get('http://localhost:5001/health-check')
        if response.status_code != 200:
            flash('Services are down', 'error')
            return render_template('service_down.html')
    except requests.ConnectionError:
        flash('Services are down', 'error')
        return render_template('service_down.html')
```

```python
@app.route('/health-check', methods=['GET'])
def health():
    return jsonify({'status': 'ok'}), 200
```

Un health check eșuat va afișa pe interfața o pagină de eroare, pentru a nu folosi soluția într-un stadiu nefuncțional.

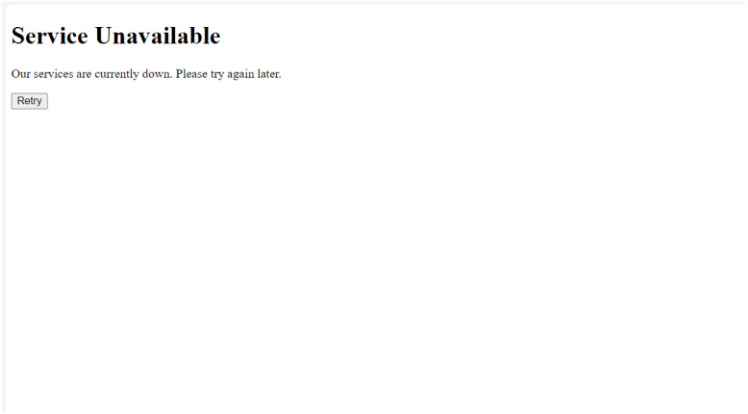


Figura 19. Pagina servicii nevalabile

În urma confirmării funcționalității tuturor serviciilor, aplicația pornește și primește cereri. Flask determină care funcție de view (controller) va gestiona cererea, pe baza URL-ului și a metodei HTTP (GET, POST). Odată ce funcția corespunzătoare este apelată aceasta poate efectua diverse operațiuni, cum ar fi interogarea bazei de date, procesarea datelor, apeluri către alte microservicii etc. Răspunsul generat poate fi o pagină HTML sau o redirecționare, în funcție de fluxul logic și secvențialitatea operațiunilor din cadrul aplicației. Primul pas în logica cronologică a aplicației este pagina principală („index.html”), unde poate fi încărcat un fișier – imaginea cu coșul de cumpărături. Ruta aplicației poate fi simplă „/” sau de upload, „/upload”, în funcție de existența unei imagini încărcate. Odată ce a fost aleasă imaginea cu coșul de cumpărături, aceasta va fi salvată în configurațiile sesiunii și vizibilă pe interfața utilizatorului. În utilizarea modelului de detectare a obiectelor sunt folosiți doi parametri: „confidence=40” și „overlap=30”, adică vor fi considerate valide predicțiile cu pragul minim de încredere 40% și gradul de suprapunere dintre casetele de delimitare 30%. Parametrii au fost selectați întrucât redundanțele să fie reduse și preciziile finale îmbunătățite.

```
detections = model.predict(img_array, confidence=40, overlap=30).json()
simple_detections = [{'x': det['x'], 'y': det['y'], 'width': det['width'],
  'height': det['height'], 'class': det['class'], 'class_id':det['class_id']}
  for det in detections['predictions']]
session['detections'] = simple_detections
```

Odată cu apăsarea butonului „Checkout”, utilizatorul va fi redirecționat întâi pe pagina de confirmare a produselor, unde, conform predicțiilor făcute anterior, vor fi decupate produsele din poză în propriile lor imagini separate. Imaginile sunt transmise pe rând modelului de clasificare care returnează clasa fiecărui produs sub forma numărului identificator („id”) al clasei. Apoi este apelată baza de date care returnează numele clasei si produsul este salvat in lista de obiecte identificate ale sesiunii. Un utilizator poate vedea pe interfață toate obiectele din coșul lui de cumpărături (tipul produsului si imaginea decupată).

```
product = Product.query.filter_by(id=predicted_class_id+1).first()
if product:
      identified_objects.append({
      'class': product.name,
      'image_url': cropped_filepath
      })
```

```
<h1>Confirmation</h1>
<h2>Final Products List</h2>
<ul>
    {% for obj in results %}
    <li>
        <p>{{ obj.class }}</p>
        <img src="{{ obj.image_url }}" alt="Detected Image"
         <button class="report-button" onclick="reportImage('{{ obj.image_url
}}', '{{ obj.class}}');">x</button>
    </li>
    {% endfor %}
</ul>
```

Tot pe pagina de „Confirmation”, poate fi folosită funcția de raportare a produselor identificate greșit prin folosirea butonului „X”. Informațiile vor fi salvate în baza de date în tabela „Issues” și imaginea produsului în fișierul „/reports” (sub forma [clasa_predicției]_[timpul_raportării].jpeg) pentru viitoare îmbunătățiri ale modelului. Obiectul va dispărea din lista de produse ale sesiunii.

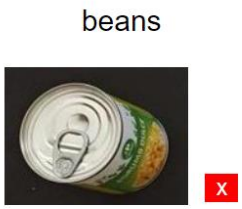


Figura 20. Produs cu buton de raportare

Figura 21. Mesaj raportare cu succes

```
# create new issue
new_issue = Issue(
    prediction = image_class,
    path = target_directory,
    checked = False
)

# add issue to database
try:
    db.session.add(new_issue)
    db.session.commit()
```

Altă funcționalitate posibilă este renunțarea la coșul de cumpărături prin apăsarea butonului „Abort basket” care va șterge configurațiile sesiunii curente și redirecționează utilizatorul la pagina principală unde procesul de introducere a unui nou coș poate începe. Metoda a fost adăugată ținând cont de scenarii posibile într-un magazin, în care o persoană poate dori să renunțe la cumpărături.

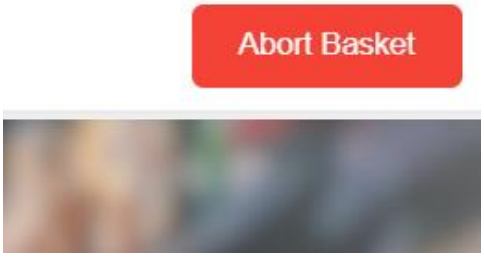


Figura 22. Buton renunțare tranzacție

```
@app.route('/abort_basket')
def abort_basket():
    session.clear()
    return redirect(url_for('index'))
```

Dacă este continuat procesul de cumpărături, următorul pas posibil este cel de checkout, unde se prezintă lista finală a produselor, inclusiv prețul total. Baza de date este apelată din nou, fiind extrase prețurile unitare din tabela Products.

La apăsarea butonului „Pay”, este afișată pentru 3 secunde o pagină care simulează acțiunea de plată cu cardul, ca într-un magazin real. La final apare mesajul final pentru clienți și posibilitatea de a începe următoareasesiune de checkout.

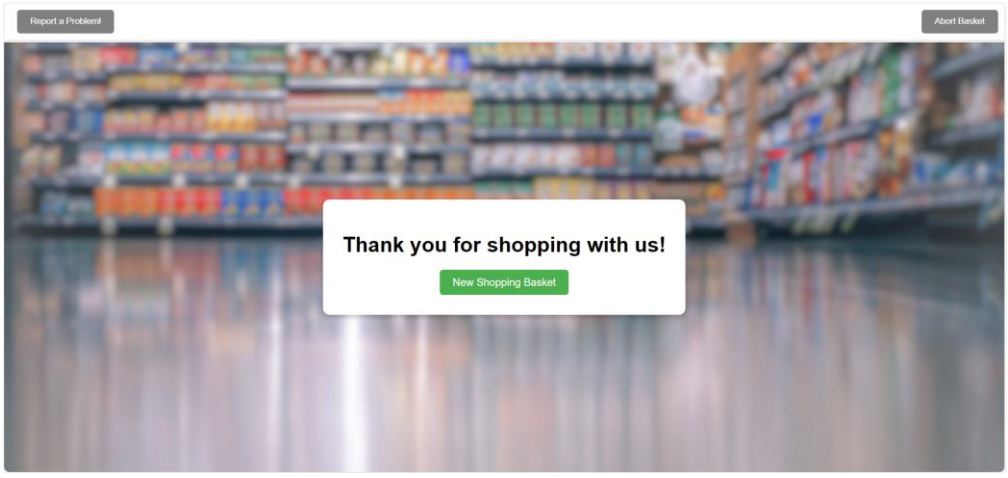


Figura 23. Pagina mesaj final

CAPITOLUL 3. GHID DE UTILIZARE AL APLICAȚIEI

Acest capitol prezintă modalitatea de folosire a aplicației de către utilizatori. Fiecare funcțiune va fi demonstrată cu ajutorul capturilor de ecran atașate pentru a face mai accesibilă utilizarea. Interfața este adaptată la scenariul unui magazin real, unde este valabil un ecran tactil (cu tehnologie touchscreen).

Home page

Pagina principală a aplicației este primul lucru pe care îl va vedea un client când dorește să acceseze serviciul de checkout. Aici apare opțiunea de a încărca poza cu coșul de cumpărături cu ajutorul butonului de „Upload”. Meniul din antetul paginii conține două butoane, și anume „Report a problem” și „Abort basket”.

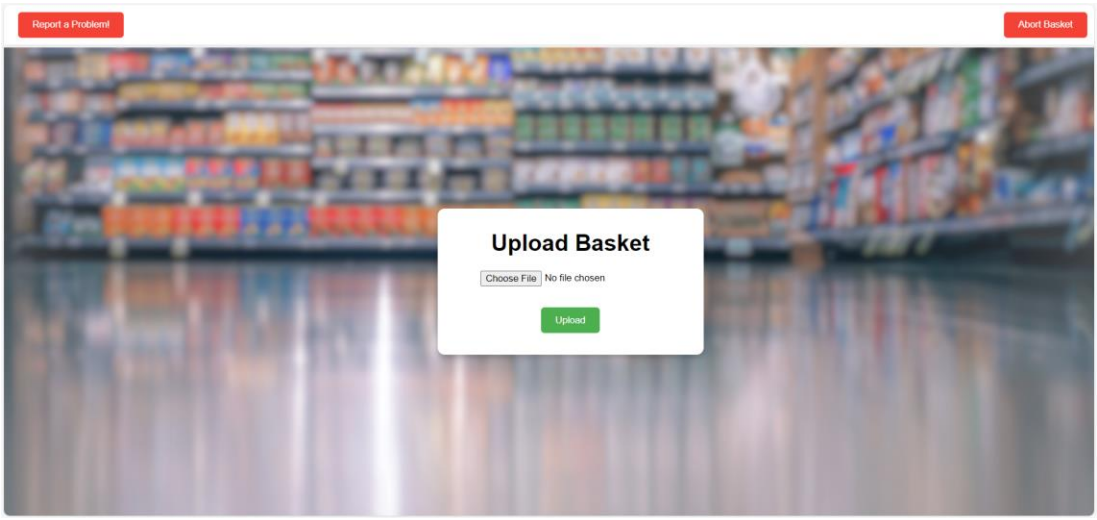


Figura 24. Pagina principală

Upload page

După încărcarea pozei cu coșul de cumpărături, aceasta va fi afișată pe ecran pentru a se asigura corectitudinea și vizibilitatea produselor. Dacă există probleme cu imaginea, aceasta poate fi încărcată iar prin același proces de upload. Dacă totul este în ordine, se apasă butonul „Checkout” pentru următorul pas.

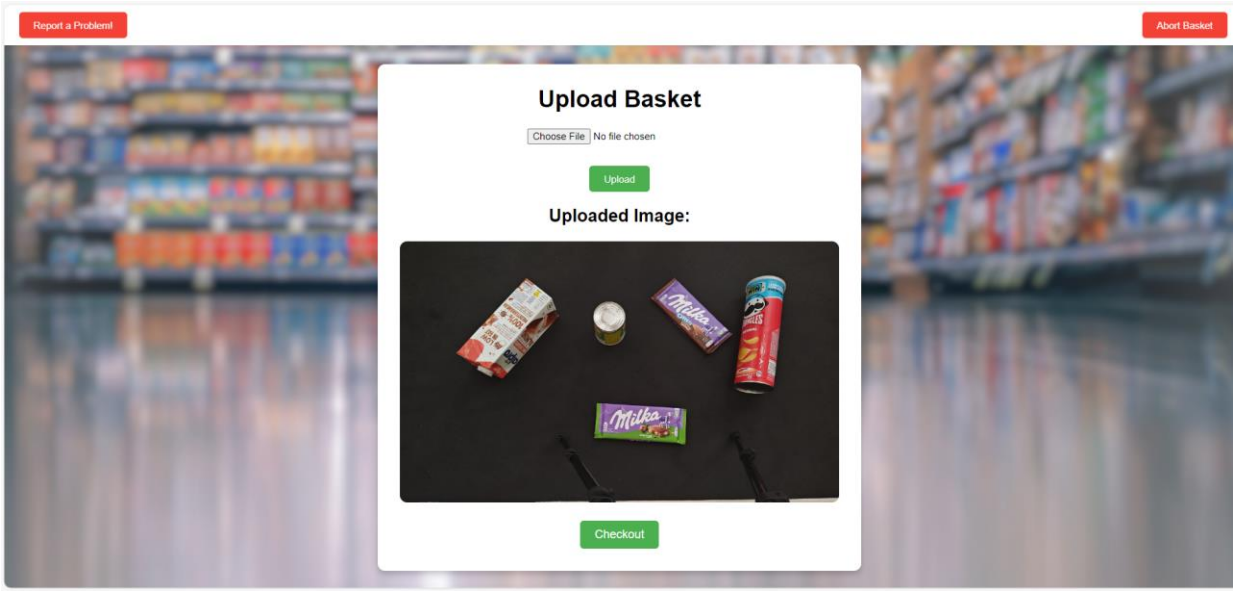


Figura 25. Pagina încărcare imagine

Confirmation page

Pagina actuală este menită confirmării produselor identificate. Sunt afișate imagini cu produsele individuale și tipul identificat. În dreptul fiecărui produs este un buton „X” prin care pot fi raportate imaginile în cazul în care este sesizată o eroare. Imaginile raportate vor fi șterse din coșul de cumpărături. Dacă sunt multe produse va trebui să se dea scroll până la finalul paginii unde se află butonul „Confirm”, prin care se validează produsele și se trece la pasul următor.

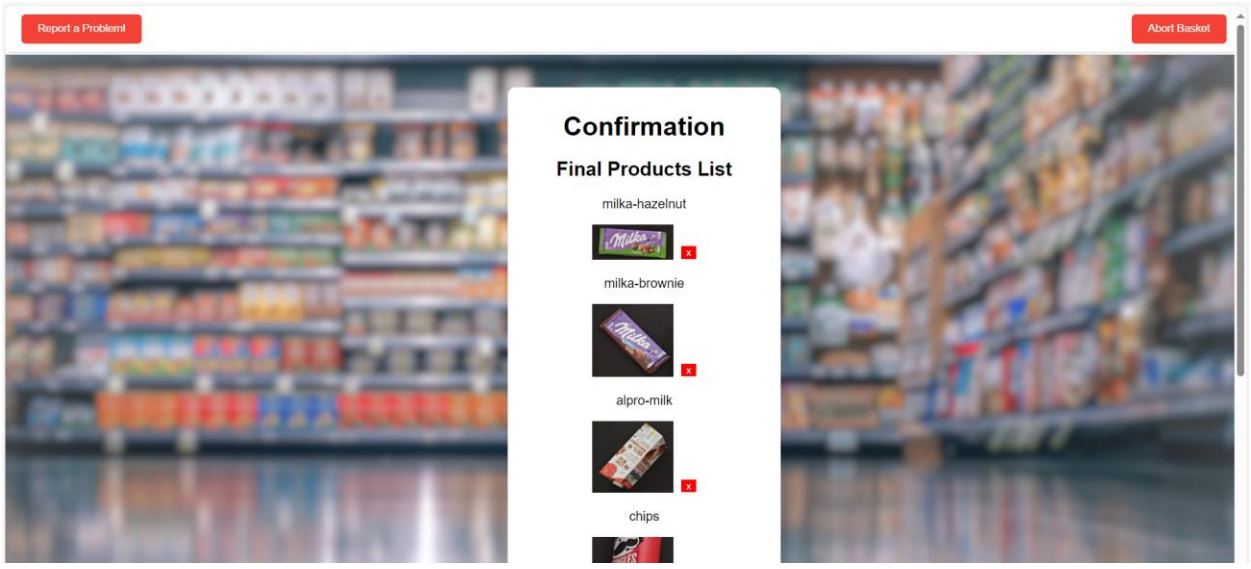


Figura 26. Pagina confirmare produse

Checkout page

Aici este valabilă lista de produse introduse, prețul acestora și totalul de plată. La final apare butonul de “Pay” pentru a continua cu achitarea produselor.

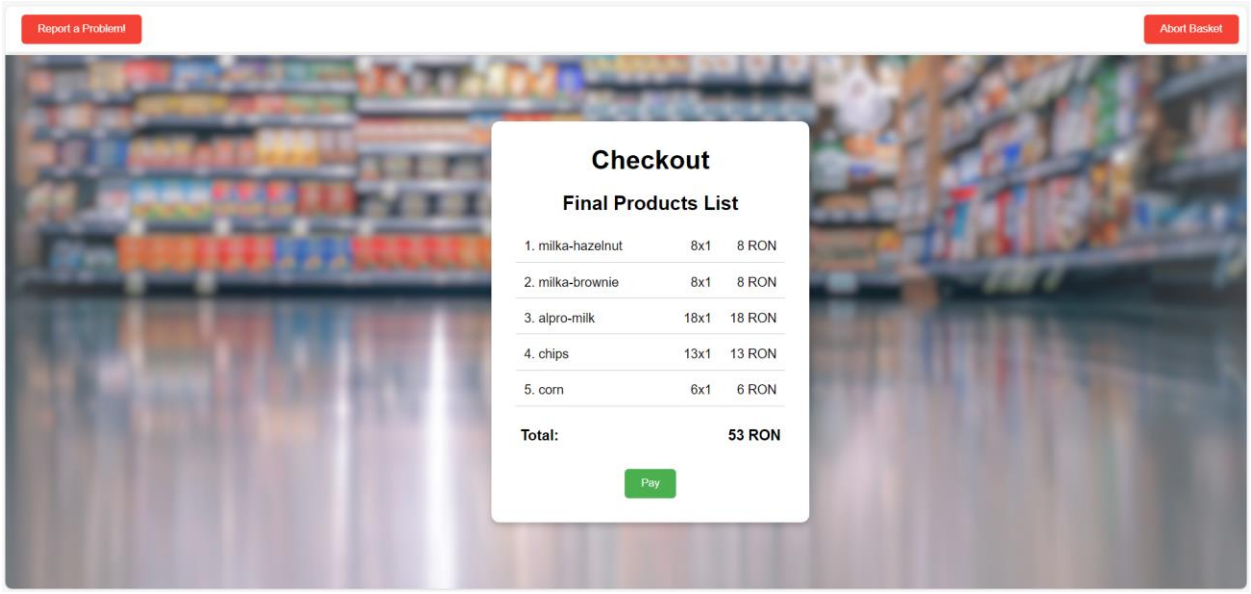


Figura 27. Pagina lista finală

Pay page

Pagina de plată apare 3 secunde pe ecran la finalizarea procesului de cumpărături, pentru a simula plata cu cardul.

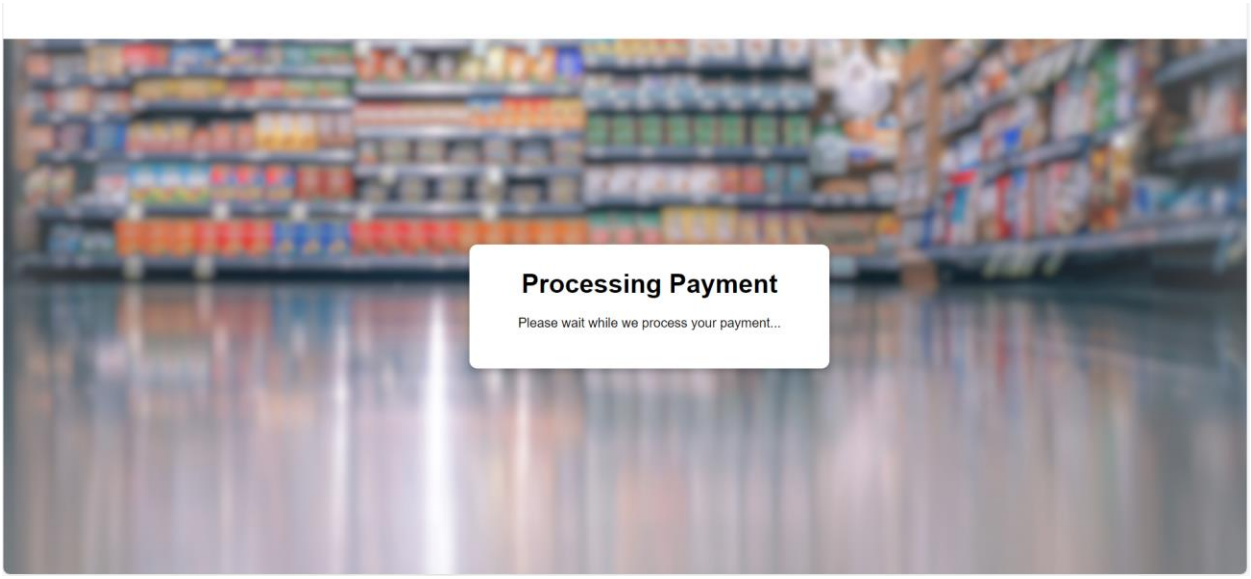


Figura 28. Pagina procesare plata

Final page

După terminarea cumpărăturilor, pe ecran apare mesajul final „Thank you for shopping with us” („Mulțumim că ați cumpărat de la noi”) și opțiunea de a începe scanarea unui nou coș de cumpărături.

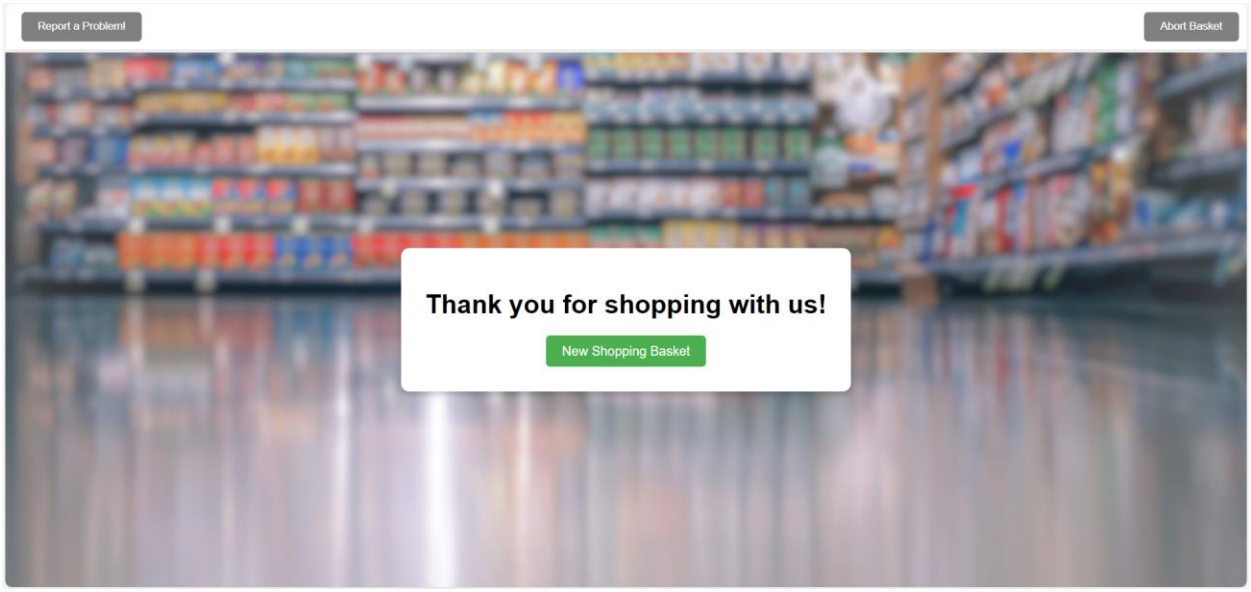


Figura 29. Pagina mesaj final

Butonul „Report a problem”

Acesta este mereu valabil în partea de sus a paginii, cerând ajutorul unui personal din magazin să ajute. Va apărea pe ecran un mesaj de așteptare a unui angajat care poate fi închis prin butonul „X” sau prin apăsarea pe ecran în orice punct al paginii.

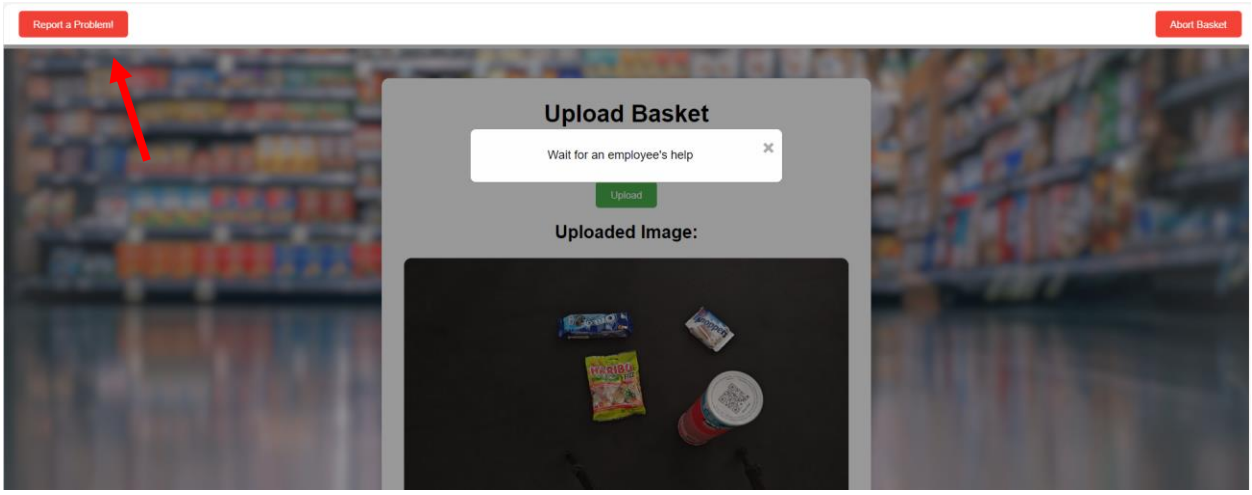


Figura 30. Butonul report problem

Butonul „Abort basket”

Poate fi apăsat în orice punct al procesului de cumpărare, acesta va șterge datele coșului de cumpărături actual și va redirecționa clientul către pagina principală, unde poate fi luată de la capăt scanarea unui nou coș.

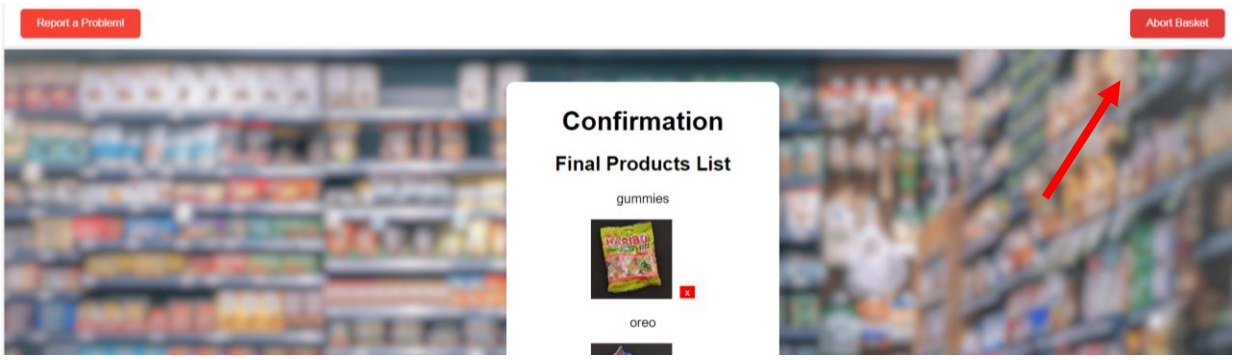


Figura 31. Butonul renunțare tranzacție

CAPITOLUL 4. DIRECȚII DE DEZVOLTARE

Aplicația, deși funcțională și eficientă, poate beneficia de o serie de îmbunătățiri care să sporească performanța, scalabilitatea, utilitățile, dar și experiența utilizatorilor. Mai departe vor fi explorate câteva posibilități de optimizare a soluției, propunând modificări care ar putea fi implementate pentru un rezultat final mai bun.

În aplicația actuală, ansamblul hardware limitează posibilitățile aplicației într-o formă minimală. Printre problemele principale cu care se confruntă modelul de identificare momentan, se numără posibilitatea de obstrucție vizuală a produselor din unghiul camerei, chiar dacă este amplasată deasupra mesei produselor. O metodă de rezolvare ar fi introducerea a mai multor camere din mai multe unghiuri, sincronizate pentru a acapara toate produsele integral. În imposibilitatea amplasării mai multor puncte de fotografiat produsele, ar fi un avantaj înlocuirea camerei actuale, care este susținută cu ajutorul unui trepied. Pentru captarea integrală a mediului de lucru, o parte din trepied era vizibilă în cadrul camerei. Chiar dacă în urma antrenării, nu a mai fost o problemă majoră și nu era detectat ca obiect, performanța modelului ar crește considerabil dacă nu ar mai exista interferența fizică a suportului de cameră.

Pe lângă soluțiile de creștere a preciziei modelului, echipamentul fizic mai poate fi dezvoltat prin înlocuirea cadrului static cu unul dinamic. În locul unei poze făcute mesei pe care se află produsele de cumpărat, poate fi implementată o masă cu bandă rulantă, cum există deja în majoritatea magazinelor de tip supermarket. Algoritmul poate fi modificat să implementeze o cameră care detectează produsele din filmare real-time. Conceptul ar spori experiența clienților și viteza cu care scanează, nemaifiind necesar să aranjeze obiectele într-un mod avantajos algoritmului pentru rezultate mai bune.

Partea de plată este simulată în aplicația actuală, însă cu integrarea unui terminal POS (Point of Service), tranzacțiile monetare ar deveni mai realiste și ar simula mai bine scenariul unei plăți la magazin.

Sistemul evidențiază viziunea unui proces de checkout fără existența codurilor de bare. Așadar multe funcționalități din cadrul unui sistem real de checkout nu au fost implementate, deoarece nu erau esențiale. Dacă ar fi pus în funcțiune într-un context real, aplicația ar trebui extinsă să includă reduceri, înregistrări cu istoricul tuturor tranzacțiilor, stocurile produselor, data de expirare etc. Fiind construit în direcția unui software cu microservicii, nu a unei aplicații

monolit, procesul de dezvoltare ar fi mai ușor, împărțind fiecare funcționalitate în propriul serviciu independent.

Tot în ideea necesităților dintr-un cadru real, pe lângă operațiuni extra, un punct foarte important de luat în considerare este securitatea aplicației. Din fericire nu există date cu caracter sensibil salvate în baza de date, însă trebuie gestionată valabilitatea microserviciilor pentru a nu permite conexiunea oricărei persoane neautorizate. Utilizatorii ar trebui să aibă acces doar la informațiile și funcțiile necesare rolului lor. Aceasta poate fi reglementată prin implementarea autentificării pentru accesul la endpoint-uri, sau prin stabilirea unui VPN care să asigure o conexiune securizată dispozitivelor pe care rulează aplicația. Aplicația actuală este o formă minimală a unui sistem de acest tip, însă în posibilitatea extinderii, ar trebui creat un protocol de răspuns la incidente care include notificarea părților afectate într-un timp cât mai scurt și măsuri de remediere care să limiteze daunele și să restabilească operațiunile normale.

În contextul actual, utilizarea containerelor Docker în dezvoltarea aplicației prezintă multiple avantaje, precum portabilitatea și eficiența în utilizarea resurselor. Totuși, pentru a răspunde nevoilor de scalabilitate într-un mediu de producție, este esențială implementarea unei soluții robuste de orchestrare a containerelor care să facă față unui număr mare de cereri fără a impacta performanța sau valabilitatea serviciilor. În acest sens, Kubernetes oferă o platformă de gestionare avansată, care permite scalarea automată a serviciilor în funcție de cerințele de trafic și resurse, oferind o posibilă soluție de extindere a aplicației.

Modelul de detecție a obiectelor implementat în cadrul aplicației demonstrează o funcționalitate eficientă, însă înregistrează o rată de eroare care poate fi îmbunătățită. Unul dintre factorii critici care influențează acuratețea modelului este volumul și diversitatea setului de date utilizat în procesul de antrenare. În prezent, modelul a fost antrenat utilizând un set de 700-800 de imagini, care, deși adecvat pentru scenarii de bază, se dovedește a fi insuficient pentru acoperirea tuturor varietăților de situații cu care aplicația ar putea să se confrunte într-un mediu real. Pentru a aborda această provocare și a îmbunătăți performanța modelului, este preferabilă extinderea setului de date prin includerea unui număr mai mare de imagini, reflectând o gamă mai largă de scenarii, inclusiv situații considerate nefavorabile sau atipice. Acestea ar putea include imagini cu iluminare variabilă, pentru a îmbunătăți precizia în diferite medii de iluminare, sau mai multe obiecte suprapuse, cu variații de orientare sau condiții fizice.

Modelul de clasificare a fost ales în urma comparării mai multor algoritmi de clasificare după experimentarea cu diverși hiperparametri, însă nu cu mai multe arhitecturi. Deși ResNet50 a fost arhitectura cu rezultate optime, o propunere ar fi testarea mai multor arhitecturi ResNet, cu numere diferite de straturi.

CONCLUZII

Proiectul prezentat aduce o contribuție semnificativă la optimizarea procesului de checkout, adresându-se nevoilor dinamice ale mediilor comerciale contemporane. Prin implementarea modelului ResNet50 pentru recunoașterea vizuală a produselor in real-time, sunt deschise noi posibilitati de fluidizare a fluxului de lucru intr-un spatiu retail.

Structura modulară a soluției permite o implementare flexibilă în diverse medii de retail, fie că vorbim de magazine mici sau lanțuri comerciale mari. Acest aspect este crucial pentru adoptarea la scară largă, oferind posibilitatea de a ajusta sau extinde funcționalitățile în funcție de cerințele specifice ale fiecărui comerciant. De asemenea, sistemul este proiectat pentru a facilita integrarea ușoară cu alte tehnologii existente în punctele de vânzare, cum ar fi sistemele POS sau platformele de management al inventarului.

Scalabilitatea este un alt avantaj remarcabil al acestei soluții, permițând comercianților să gestioneze eficient creșterea volumului de clienți sau expansiunea stocurilor de produse fără compromisuri în ceea ce privește performanța. Modelul de învățare automată poate fi antrenat continuu pentru a recunoaște noi produse, facilitând astfel adaptarea rapidă la schimbările din oferta comercială. Această capacitate de învățare incrementală este vitală pentru menținerea unui nivel înalt de relevanță și eficiență în recunoașterea produselor.

Implementarea acestei soluții oferă, de asemenea, oportunitatea de a colecta date valoroase în timp real despre preferințele și comportamentul de cumpărare ale clienților, deschizând calea către analize detaliate care pot informa strategii de marketing și optimizări operaționale. În esență, proiectul nu doar că simplifică procesele logistice ale magazinelor, dar transformă și modul în care interacțiunile cu clienții sunt gestionate, conducând la o experiență de cumpărare îmbunătățită și la un proces de checkout rapid și fără erori.

Prin urmare, soluția propusă este mai mult decât o simplă îmbunătățire tehnologică; ea reprezintă un pas înainte în evoluția digitală a comerțului, adaptându-se nevoilor emergente ale sectorului retail și anticipând tendințele viitoare. Aceasta oferă un exemplu clar de cum inovațiile tehnologice pot fi aplicate în mod eficient pentru a îmbunătăți operațiunile zilnice și pentru a maximiza satisfacția clienților.

Bibliografie

Bangar, S. (2022). *Resnet Architecture Explained*. Retrieved from medium.com: https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d

Bayer, M. (2012). SQLAlchemy. In *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks.*

Collier, J. (2012). Situational Variables and Attitudes toward Self-Service. *Society for Marketing Advances*.

Collier, J. (2013). Only If It Is Convenient: Understanding How Convenience Influences Self-Service Technology Evaluation. *Journal of Service Research*. doi:10.1177/1094670512458454

Collier, J. (2014). Self-service delight: Exploring the hedonic aspects of self-service. *Journal of Business Research*. doi:doi.org/10.1016/j.jbusres.2014.09.036

dev0928. (2020). *Build RESTful APIs using Python / Flask.* Retrieved from dev.to.

Du, X. (2023). A Comparative Study of Different CNN Models and Transfer Learning Effect for Underwater Object Classification in Side-Scan Sonar Images. doi:doi.org/10.3390/rs15030593

Dwyer, B. (2024). Roboflow (Version 1.0). Retrieved from https://roboflow.com

Fernandes, T. (2017). The effect of self-checkout quality on customer satisfaction and repatronage in a retail context. *Service Business*. doi:10.1007/s11628-016-0302-9

Grinberg, M. (2018). Flask web development: developing web applications with python.

Harris, C. (2020). Array programming with NumPy. *Nature*, 357-362.

He, K. (2015). Deep Residual Learning for Image Recognition. doi:doi.org/10.48550/arXiv.1512.03385

Koonce, B. (2021). ResNet 50. In *Convolutional Neural Networks with Swift for Tensorflow.* doi:https://doi.org/10.1007/978-1-4842-6168-2_6

(n.d.). Retrieved from Kaggle: kaggle.com

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*.

Nyuytiymbiy, K. (2020). *Parameters and Hyperparameters in Machine Learning and Deep Learning*. Retrieved from towardsdatascience.com.

Orel, D. (2013). Supermarket self-checkout service quality, customer satisfaction, and loyalty: Empirical evidence from an emerging market. *Journal of Retailing and Consumer Services*. doi:https://doi.org/10.1016/j.jretconser.2013.07.002

Page, D. (2016). pgAdmin 4.

Sharma, P. (2020). Self-service technology in supermarkets – Do frontline staff still matter? *Journal of Retailing and Consumer Services*. doi:doi.org/10.1016/j.jretconser.2020.102356

Szegedy, C. (2014). Going Deeper with Convolutions.

The pandas development team. (2020). Pandas.

The PostgreSQL Global Development Group. (1996). PostgreSQL.

Van Rossum, G. (1995). Python reference manual. Centrum voor Wiskunde en Informatica Amsterdam.