

UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE



PROIECT DE DIPLOMĂ

Detectia datelor de expirare folosind retelele neurale adânci

Vlad-Alexandru Florea

Coordonator științific:
Conf. dr. ing. Traian Rebedea

BUCUREŞTI

2019

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT



DIPLOMA PROJECT

Expiry date recognition using deep neural networks

Vlad-Alexandru Florea

Thesis advisor:
Conf. dr. ing. Traian Rebedea

BUCHAREST

2019

CUPRINS

Sinopsis	3
Abstract	3
Mulțumiri.....	4
1 Introducere.....	5
1.1 Context.....	5
1.2 Problema.....	5
1.3 Obiective	6
1.4 Structura lucrării	6
2 Analiza și specificarea cerințelor	8
3 Abordări existente	9
4 Soluția propusă.....	12
4.1 Extragerea regiunilor de interes	13
4.1.1 Arhitectura TextBoxes++	13
4.2 Extragerea textului.....	19
4.2.1 Arhitectura CRNN	20
4.3 Construirea unui set de antrenare pentru detecția datelor de expirare	21
4.4 Filtrare și obținerea rezultatului final	23
4.5 Aplicație pentru reducerea risipei alimentare.....	23
4.5.1 Frontend	24
4.5.2 Backend	25
5 Detalii de implementare.....	26
5.1 TextBoxes++	26
5.1.1 Utilizarea tf.data pentru eficiență maximă la antrenare	26
5.1.2 Crearea fisierelor TFRecords	27
5.2 CRNN	28
5.3 Construirea unui set de antrenare pentru detecția datelor de expirare	29
5.3.1 Imaginele reale	29
5.3.2 Imaginele generate sintetic	30
5.3.3 Augmentarea imaginilor.....	32
5.4 Filtrare și obținerea rezultatului final	32

5.5	Aplicație pentru reducerea risipei alimentare.....	33
5.5.1	Frontend	33
5.5.2	Backend	33
6	Evaluarea rezultatelor	35
7	Concluzii.....	37
7.1	Dezvoltări ulterioare	37
8	Bibliografie.....	38
9	Anexe	43
9.1	Wireframe-uri folosite pentru validarea interfeței grafice	43
9.2	Capturi de ecran din aplicația finală	44
9.3	Diagrama de clase pentru aplicația Android	45
9.4	Comparație între arhitecturile SSD [26] și YOLO [27].....	46

SINOPSIS

Risipa alimentară este una dintre cele mai grave probleme ce afectează planeta noastră la momentul actual, având atât un puternic impact social cât și consecințe grave asupra mediului înconjurător și economiei.

Această lucrare propune un algoritm ce folosește rețele neurale adânci pentru recunoașterea optică a caracterelor, specializat în detecția datelor de expirare, care poate fi folosit pentru a ține evidență datelor imprimate pe produse, ajutând astfel la reducerea risipei alimentare.

Algoritmul poate fi util atât pentru companii cât și pentru persoane fizice, folosind un dispozitiv mobil ce dispune de o cameră foto (exemplu: telefon mobil) pentru a scana data de expirare a diferitelor produse într-un timp foarte scurt sau poate fi atașat unui dispozitiv fix precum un frigider sau o bandă rulantă folosind o placă de dezvoltare ce poate capta imagini (exemplu: RaspberryPi).

De asemenea, acest algoritm poate fi combinat cu un motor de sinteză vocală pentru a ajuta persoanele cu deficiențe de vedere să descopere data de expirare a unui produs.

ABSTRACT

Food waste is one of the worst problems currently affecting our planet as it comes with huge environmental, social and economic issues.

This paper proposes a deep learning based algorithm for optical character recognition, specifically tuned to detect expiration dates, that can be used to track the expiration date printed on the packaging of food items and thus can help reduce wasted food.

The algorithm can be useful for both businesses and consumers, by using a camera-enabled mobile device (e.g. smartphone) to scan the expiration date of various items in a very short time or it can be embedded in a fixed device like a fridge or a conveyor belt by using a camera-enabled single board computer (e.g. RaspberryPi).

Furthermore, this algorithm can be combined with a speech synthesis engine to help those who are vision impaired find out the expiration date of a certain item.

MULTUMIRI

Mulțumesc domnului profesor Traian Rebedea pentru sfaturile acordate 24/7. Fără ajutorul său acest proiect nu ar fi ajuns niciodată la final în timp util.

Mulțumesc colegilor mei Cosmin Dobrița, Cosmin Sterian, David Ivan, Florin Iordache și Valeriu Alexandru pentru imaginile cu date de expirare pe care le-au donat în faza de testare și pentru participarea lor în procesul de validare a interfeței grafice.

1 INTRODUCERE

O aplicație ce permite utilizatorului să scaneze data de expirare a unui produs folosind imagini surprinse de camera foto a unui dispozitiv mobil sau fix ar putea economisi foarte mult timp în anumite procese, ar putea încuraja folosirea unor aplicații ce monitorizează aceste date pentru a reduce risipa alimentară și i-ar putea ajuta pe cei cu deficiențe de vedere să cunoască această informație fără ajutorul unei alte persoane.

Deși există lucrări de cercetare în domeniul detecției datelor de expirare în imagini [1, 2], după căutări amănunțite nu am reușit să găsesc pe piață o aplicație care să folosească o asemenea soluție, în ciuda faptului că există destul de multe aplicații pentru monitorizarea datelor de expirare [3, 4, 5, 6], ce au un număr considerabil de utilizatori și folosesc metode de introducere manuală a datei precum: „date picker”, câmpuri text, etc.

1.1 Context

Risipa alimentară este una dintre cele mai grave probleme ce afectează planeta noastră la momentul actual. Conform Organizației Națiunilor Unite, la nivel global, o treime din alimentele produse se pierd în fiecare an [7].

Risipa de alimente reprezintă o problemă socială importantă. Conform statisticilor 508.2 milioane de locuitori ai celor 28 de țări membre ale Uniunii Europene au dificultăți majore în a-și procura hrana zilnică [8], iar 88 milioane de tone de alimente sunt aruncate anual la nivelul Uniunii Europene, dintre care 53% se arunca la nivel casnic, restul fiind pierdute pe parcursul lanțului de producție și aprovisionare [9]. De asemenea, se poate observa și o problemă economică majoră deoarece prețul anual estimat pentru această cantitate de deșeuri este de 143 de miliarde de euro [9].

Statisticile la nivelul țării noastre sunt de asemenea îngrijorătoare. Conform Asociației Mai Mult Verde în jur de 2.5 milioane de tone de alimente se aruncă anual în România. Această cantitate corespunde încărcăturii a 127 500 de tiruri alinate în coloană din București până în München [10].

Risipa de alimente afectează puternic mediul înconjurător atât prin epuizarea inutilă a resurselor naturale [11] cât și prin amprenta sa de carbon. Comparativ, dacă risipa alimentară ar fi o țară, aceasta ar fi cea de-a treia ca emisii de gaze cu efect de seră din lume [12].

1.2 Problema

La momentul actual există soluții pentru monitorizarea datelor de expirare adresate atât companiilor cât și persoanelor fizice [3, 4, 5, 6], ce au scopul de a trimite notificări utilizatorului atunci când produsele sale ajung aproape de data lor de expirare. Din păcate, toate aceste aplicații necesită introducerea manuală a datei de expirare pentru fiecare produs în parte, ceea ce duce la un consum foarte mare de timp.

Unele companii încearcă introducerea unor coduri de bare unice pentru fiecare produs în parte, ce pot conține atât identitatea unică a produsului cât și informații despre data de

expirare [13]. Adoptarea acestora va fi însă anevoieasă deoarece necesită o schimbare majoră a tuturor sistemelor atât pentru magazine, depozite, etc. cât și pentru producători.

O soluție ce asigură introducerea automată a datei de expirare ar putea crește rata de adoptie a sistemelor de monitorizare, deoarece nu sunt necesare schimbări majore ale infrastructurii pentru a putea fi folosită, iar astfel poate fi redusă risipa alimentară cauzată de produse uitate. În cadrul magazinelor din România obligația de a urmări data de expirare a produselor de pe raft intră de obicei în atribuțiile şefilor de raion sau plasatorilor („promoteri”) contractați de către producători și în general aceștia folosesc liste scrise manual. De asemenea problema produselor uitate există și în mediul casnic, deoarece conform unui studiu online, realizat în Statele Unite ale Americii de către KRC Research pentru compania Peapod, 2 din 5 locuitori ai New York-ului admit că uită produse expirate în frigider [14].

O altă problemă este că legile în vigoare nu obligă producătorii să includă data de expirare în format Braille pe ambalajele produselor lor, deci persoanele cu deficiențe de vedere nu pot avea acces la această informație importantă. Singura soluție existentă este o aplicație mobilă ce determină poziția datei de expirare în funcție de poziția codului de bare imprimat pe produs, însă această metodă necesită înregistrarea manuală a poziției datei pentru fiecare produs în parte într-o bază de date, iar pentru unele produse poziția datei de expirare poate varia între mai multe loturi sau uneori chiar între produse ce fac parte din același lot, fiind imposibil de citit cu o aplicație ce utilizează această tehnică [2].

1.3 Obiective

Proiectul are ca scop principal dezvoltarea unui algoritm pentru recunoașterea automată a unei date de expirare prezente într-o imagine.

De asemenea va fi dezvoltată o aplicație ce rulează pe un telefon mobil sau pe o placă de dezvoltare și integrează acest algoritm pentru a demonstra modul în care soluția poate fi folosită în lumea reală, cu scopul de a reduce risipa alimentară.

1.4 Structura lucrării

Pentru a construi soluția propusă, mai întâi am stabilit cazurile de utilizare cu cea mai mare utilitate și funcționalitățile necesare ce vor trebui studiate pentru a le putea acoperi cu succes în practică (Capitolul 2). De asemenea am studiat metode existente, ce ar putea fi adaptate pentru problema de față și am testat precizia lor, pentru a dezvolta algoritmul propus pornind de la soluția cea mai bună la momentul actual (Capitolul 3). Am pornit de la o rețea neurală adâncă, cu o arhitectură ușor modificată (Capitolul 4), care va extrage zone de interes ce conțin text similar unei date de expirare (Secțiunea 4.1). Regiunile de interes extrase din imagine, vor fi folosite ca *input* pentru o altă rețea neurală adâncă ce va extrage textul conținut în acestea (Secțiunea 4.2). Pentru cele două modele se vor încărca ponderile obținute pe setul de date inițial apoi rețelele vor fi re-antrenate pe un set de date construit special pentru problema detecției datelor de expirare (Secțiunea 4.3). La final, textul va fi filtrat pentru a obține data de expirare (Secțiunea 4.4). În jurul algoritmului propus am

construit și o aplicație bazată pe o arhitectură client-server, formată dintr-un REST API pe partea de *backend* (4.5.2) și două aplicații client, pentru telefoane mobile și single board computere, pe partea de *frontend* (4.5.1), pentru a demonstra cum poate fi folosită soluția în lumea reală cu scopul de a reduce risipa alimentară (Secțiunea 4.5). În cadrul implementării propriu-zise (Capitolul 5) pentru arhitectura de extragere a regiunilor de interes am scris de la zero un model în Tensorflow cu scopul de a înțelege cât mai bine procesul de funcționare, astfel încât să pot face ajustări care să îmbunătățească performanța (Secțiunea 5.1). Arhitectura de extragere a textului din regiunile de interes nu a necesitat decât un pas de reantrenare aşa că am folosit un model open source existent (Secțiunea 5.2). În capitolul 5 sunt descrise procedeele necesare pentru construirea setului de date (Secțiunea 5.3) și filtrarea textului (Secțiunea 5.4). Secțiunea 5.5 detaliază modul în care am dezvoltat aplicația finală. În capitolul 6 am evaluat rezultatele finale obținute, iar tot efortul necesar dezvoltării soluției propuse s-a dovedit a fi un succes. Toate concluziile trase după finalizarea acestui proiect se găsesc în capitolul 7.

2 ANALIZA ȘI SPECIFICAREA CERINȚELOR

După cum se va observa și în capitolul următor (capitolul 3) problema detecției datelor de exprirare în imagini poate fi comparată ca dificultate cu problema „Scene Text Detection” [15], foarte discutată în literatura „state of the art” deoarece prezintă un grad de dificultate ridicat și poate fi rezolvată cu rezultate bune folosind rețele neurale adânci. Sistemele existente pentru recunoaștere optică a caracterelor au o precizie foarte slabă când vine vorba despre detecția datelor de exprirare (Tabelul 1, Capitolul 3).

Precizia foarte slabă a sistemelor existente pentru recunoaștere optică a caracterelor este cel mai probabil cauzată de faptul că seturile de date pe care au fost antrenate nu includeau text cu font de tipul „dot matrix” din care sunt formate majoritatea datelor de exprirare. De asemenea sistemele prezentate în capitolul 3 încearcă să obțină un grad cât mai mare de generalitate pentru orice tip de text, dimensiune, densitate de text în imagine, etc. deci nu se pot supra-specializa în detecția datelor de exprirare.

Precizia ridicată este esențială atât pentru persoanele cu deficiențe de vedere cât și pentru ceilalți utilizatori și companii deoarece influențează direct utilitatea aplicației finale și deci rata de adoptie a acesteia. Problema risipei alimentare descrisă în secțiunea 1.1 nu va putea fi redusă decât dacă va crește rata de adoptie a aplicațiilor de acest tip, deci o precizie cât mai bună este esențială.

Din acest motiv în această lucrare vor fi studiate metode prin care se poate obține un sistem specializat în recunoașterea optică a datelor de exprirare cu scopul de a obține o precizie mai bună. În comparație cu soluțiile existente, ce folosesc tehnici clasice de vedere computațională pentru detecția datelor de exprirare, în această lucrare am dezvoltat un algoritm bazat pe rețele neurale adânci, construite folosind tehnici de ultimă generație (precum modelul TextBoxes++ [16] publicat cu numai un an în urmă la momentul redactării acestei lucrări) cu scopul de a obține performanțe cât mai bune.

O altă problemă este că la momentul actual nu există un set de date public pentru problema detecției datelor de exprirare. Pentru a dezvolta tehniciile din această lucrare va fi necesară construirea unui set de date de dimensiuni considerabile. Prin publicarea ulterioară a setului de date construit și instrumentelor utilizate alți dezvoltatori vor putea încerca metode proprii pentru detecție și vor putea contribui la rezolvarea problemei în cadrul comunității open source. Îmbunătățirea tehniciilor prezentate în capitolele următoare ar putea aduce beneficii majore în viitorul apropiat, iar adoptarea sistemului prezentat de către utilizatori ar putea reduce problema risipei alimentare, o problemă foarte gravă conform statisticilor prezentate în secțiunea 1.1.

3 ABORDĂRI EXISTENTE

Sistemele existente de recunoaștere optică a caracterelor au o performanță foarte slabă atunci când vine vorba despre detecția datelor de expirare. Pentru a putea vizualiza performanțele acestora am utilizat propriul meu set de date de validare (descriși în secțiunea 4.3) peste care am rulat soluțiile existente, iar apoi pentru consistență am trecut textul extras prin algoritmul de filtrare menționat la pasul 4.4, ce folosește expresii regulate și câteva alte tehnici pentru a extrage date de expirare.

În secțiunea următoare vor fi prezentate câteva dintre cele mai populare sisteme existente și va fi analizată aplicabilitatea lor pentru această problemă folosind metoda de mai sus:

- a) **Tesseract OCR** este un motor open source pentru recunoaștere optică a caracterelor, oferit gratuit sub licență Apache 2 [17]. Pentru a testa performanțele acestuia am folosit pachetul „PyTesseract” [18], un „wrapper” cu ajutorul căruia se pot accesa facilitățile Tesseract OCR [17] direct din limbajul Python. Rezultatele obținute (Tabelul 1) arată că acest sistem nu poate fi aplicat problemei detecției datelor de expirare, fiind mai mult specializat în recunoaștere optică a caracterelor pentru documente scanate, în timp ce problema detecției datelor de expirare este similară ca dificultate cu problema „Scene Text Detection” [15] din literatura „state of the art”. Încercările mele de a folosi acest sistem pentru problema curentă au eşuat. La majoritatea imaginilor *output*-ul metodei *image_to_text()* este vid, chiar în cazul în care data este scrisă cu font obișnuit și nu de tip „dot matrix”. De asemenea nici în cazul în care se taie din imagine numai portiunea ce conține data de expirare nu se obține un rezultat bun, din cauza nivelului mare de zgomot din imagine (fundal, suprapunere cu alte elemente, etc.) și fontului de tip „dot matrix”.
- b) Având în vedere eşecul Tesseract OCR [17] este clar că pentru această problemă va fi necesar un sistem mai robust. **Modele de tipul „scene text detector”** din literatura „state of the art” par să fie o soluție mai bună, fiind specializate în detecție a textului „in the wild” precum numere poștale, panouri publicitare, semne de circulație etc. Unele dintre cele mai răspândite sunt **SegLink** [19] și **TextBoxes++** [16]. Aceste modele sunt de obicei combinate cu modelul CRNN [20] (explicit în secțiunea 4.2.1), pentru a extrage textul din regiunile în care a fost detectat. Pentru cele trei modele am utilizat implementări open source disponibile pe GitHub (SegLink, TextBoxes++ și CRNN [21]) folosind ponderile antrenate de către autor (*default*). Rezultatele obținute se găsesc în Tabelul 1.
- c) **Google Cloud Vision API** [22] este un serviciu ce le oferă dezvoltatorilor de aplicații acces la modele pentru computer vision gata făcute, ce rulează în cloud și pot fi accesate prin intermediul unui REST API. Folosirea sistemului este gratuită numai pentru primele 1000 de imagini. Performanțele „out of the box” ale acestui sistem sunt cele mai bune dintre toate modele pre-antrenate studiate în această secțiune și pot fi observate în tabelul Tabelul 1. De asemenea în Figura 1 poate fi văzut un exemplu în care detecția este corectă și un exemplu în care acest model eșuează.

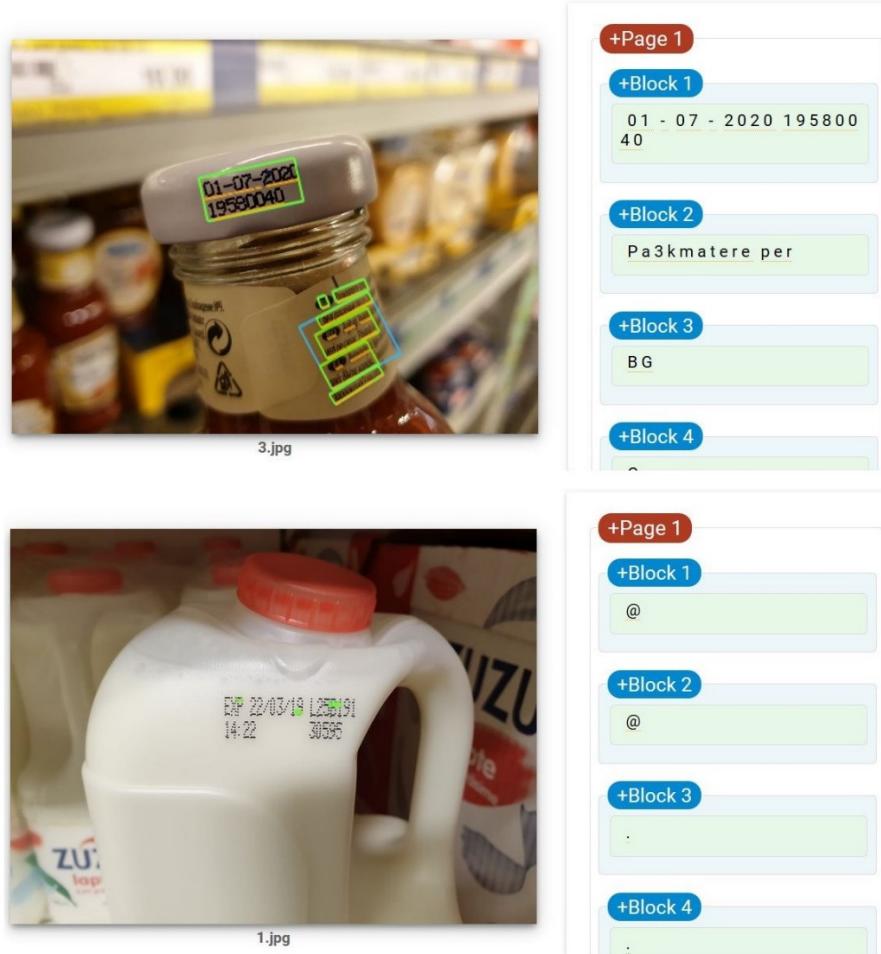


Figura 1 Exemplu detecție corectă (sus) și detecție eșuată (jos) folosind Google Cloud Vision API

Tabelul 1 Precizia obținută de sisteme existente pe setul de validare

Sistem	Precizie
Tesseract OCR	10.6 %
TextBoxes++ (default) + CRNN (default)	13.6 %
SegLink (default) + CRNN (default)	16.6 %
Google Cloud Vision API	50.0 %

Deoarece sistemele prezentate mai sus nu oferă o performanță foarte bună, experții în domeniu au rezolvat de-a lungul timpului problema detecției datelor de expirare ocolind o parte din pașii necesari unui sistem de recunoaștere optică a caracterelor „end to end”:

- d) În cadrul lucrării „Product Barcode and Expiry Date Detection for the Visually Impaired Using a Smartphone” [2], autorii au folosit relația dintre poziția codului de bare al unui produs și poziția de pe ambalaj a datei de expirare. În felul acesta este eliminat practic pasul de extragere a regiunilor de interes (detaliat în secțiunea 4.1) fiind necesară numai

extragerea textului din regiunea deja setată. Dezavantajul major este că poziția datei de expirare poate să nu fie neapărat corelată cu poziția codului de bare, în special pentru produse unde aceasta este imprimată manual și nu poate avea o poziție fixă. De asemenea este necesară construirea unei baze de date ce stochează relația dintre cele două pentru fiecare produs în parte.

- e) Lucrarea „A Vision Approach for Expiry Date Recognition using Stretched Gabor Features” [1] propune o metodă de extragere a regiunilor de interes folosind tehnici clasice de vedere computațională: inițial se aplică o serie de filtre peste imagine (binarizare, subțiere, etc.) apoi se generează un set de trăsături pentru fiecare regiune folosind operații matematice complexe, trăsăturile urmând a fi clasificate folosind o rețea neurală cu un singur strat ascuns sau un SVM pentru a separa regiunile ce conțin date de expirare de restul imaginii. Algoritmul propus în această lucrare elimină nevoie de a preciza manual trăsături, deoarece rețeaua adâncă de convoluție din secțiunea 4.1 va extrage propriile trăsături din imaginile folosite pentru antrenare.
- f) Deși informațiile disponibile online sunt limitate, probabil pentru a păstra secretele din industrie, fabricile folosesc soluții bazate pe camere video pentru a verifica corectitudinea datei de expirare la ieșirea de pe linia de producție (exemplu: [23]). Avantajul în acest caz este că se cunoaște poziția exactă a datei de expirare pe produs și fontul folosit la imprimare. Algoritmul prezentat în secțiunea 4 s-ar putea dovedi util și în industrie datorită gradului mare de generalitate, în special în cazul unui depozit ce primește produse cu formate variate ale datei de expirare.

4 SOLUȚIA PROPUȘĂ

Similar arhitecturilor pentru recunoașterea optică a caracterelor din literatura „state of the art” prezентate mai sus, algoritmul propus reprezintă o soluție „end to end” pentru detecția datelor de expirare în imagini (cu precizie foarte mare) și este format din trei pași importanți de procesare, a căror ordine poate fi observată mai jos în Figura 2.

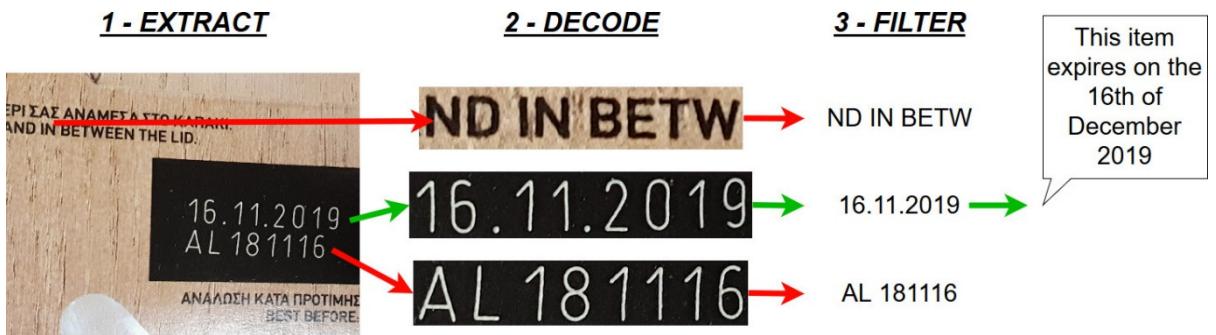


Figura 2 Pașii principali de procesare ai algoritmului propus

Primul pas constă în extragerea regiunilor de interes ce ar putea conține date de expirare folosind o rețea neurală adâncă „state of the art” din categoria „Scene Text Detector” (Secțiunea 4.1). Spre deosebire de tehniciile clasice pentru recunoaștere optică a caracterelor din documente, rețelele neurale de acest tip sunt specializate în detecție a textului „in the wild” precum imagini ce conțin firme ale magazinelor, numere poștale, panouri publicitare, semne de circulație etc. (exemplu: setul de date ICDAR [24]). Astfel pot detecta text în poziții variate, diverse condiții de iluminare, fonturi, etc. Din acest motiv sunt cele mai potrivite pentru detecția datelor de expirare, o problemă ce prezintă dificultăți similare.

În cadrul celui de-al doilea pas, patrulaterele ce delimitizează regiunile de interes obținute la pasul anterior vor fi „tăiate” din imagine și folosite ca input pentru o rețea neurală adâncă de tipul „Scene Text Detector”, specializată în extragerea secvențelor din imagini. Astfel, pentru fiecare regiune de interes extrasă la primul pas, vom obține conținutul acesteia în format text ASCII (Secțiunea 4.2).

Deoarece nu există seturi de date publice pentru problema detecției datelor de expirare, la antrenarea celor două rețele neurale va fi necesară construirea unui set de date specializat pe această problemă, format atât din imagini reale cât și din imagini generate artificial folosind un motor grafic 3D. Acest set de date este detaliat în cadrul secțiunii 4.3.

Pentru ultimul pas (Secțiunea 4.4), textul extras în cadrul etapei anterioare va fi filtrat folosind o serie de expresii regulate și criterii logice, apoi se va obține data conținută folosind o bibliotecă ce poate detecta date calendaristice în mai multe formate.

Opțional, la final, data extrasă poate fi trimisă ca input către un motor text-to-speech pentru a fi accesibilă persoanelor cu deficiențe de vedere (Secțiunea 4.5.1).

4.1 Extragerea regiunilor de interes

Prima parte a sistemului constă în extragerea regiunilor de interes din imagine ce prezintă trăsături similare cu cele ale unei date de expirare.

În cadrul acestui proiect am ales arhitectura TextBoxes++ [16] pe baza performanțelor ridicate pentru detecție și localizare de text în cadrul „challenge”-ului ICDAR (International Conference on Document Analysis and Recognition), pentru viteza mare de rulare în comparație cu alte modele și pentru existența unei implementări oficiale publicate de către autori [25], ce demonstrează modul de funcționare și justifică rezultatele obținute. De asemenea similaritatea dintre acest model și modelul Single Shot Detector (SSD) [26] face arhitectura TextBoxes++ [16] ușor de înțeles, deoarece SSD [26] este un model de object detection foarte utilizat la ora actuală (folosit pentru mașini autonome, camere de supraveghere inteligente, etc). În consecință, există o mulțime de materiale disponibile online unde este explicitat modul de funcționare al arhitecturii SSD [26].

4.1.1 Arhitectura TextBoxes++

Modelul TextBoxes++ [16] tratează problema detecției textului în imagini sub forma unei probleme de detecție a obiectelor. Pentru a înțelege modul de funcționare al acestei arhitecturi este necesară mai întâi înțelegerea modelelor de recunoaștere a obiectelor de tip „Single Shot” pe care este bazată.

În literatura „state of the art” unul dintre cele mai performante și populare modele pentru detecția obiectelor este modelul „You Only Look Once” (YOLO) [27]. Spre deosebire de alte abordări, arhitectura YOLO [27] poate rezolva această problemă folosind numai o singură rețea de conoluție „end to end”, inputul fiind reprezentat de imagini iar *output*-ul de către coordonatele dreptunghiurilor ce încadrează obiectele detectate și clasa asociată fiecărui. Astfel se poate obține o viteză foarte mare de rulare, iar performanțele nu sunt afectate în comparație cu alte modele.

Ideea de bază a acestei arhitecturi constă în împărțirea regiunilor din imagine sub forma unei grile. Fiecare pătrat din grilă va avea asociată o „fâșie” din stratul final de conoluție al rețelei, sub forma unui tensor 1D. Acest strat final de conoluție poartă denumirea de „feature map” în lucrarea YOLO [27].

Prima parte a fiecărui tensor 1D asociat conține valorile *offset* prezise pentru a ajusta poziția pătratului în spațiul 2D și factorii de scalare ai acestui pătrat pe axe x și y, pentru a acoperi întreaga suprafață a obiectului detectat. Aceste valori sunt denumite în lucrarea originală *bx*, *by*, respectiv *bw* și *bh* (Figura 3).

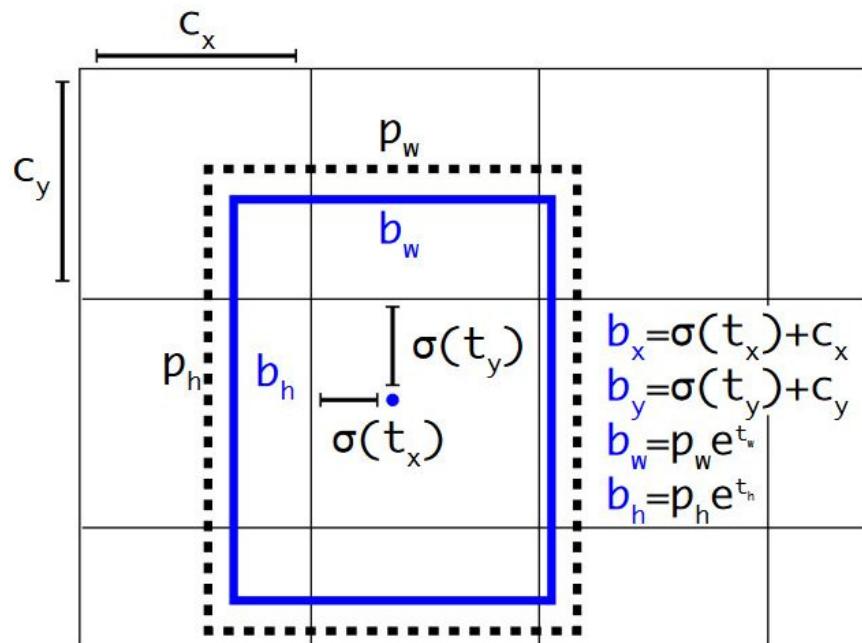


Figura 3 Parametrii *offset* asociați unui pătrat din grilă
(imagine preluată din [27])

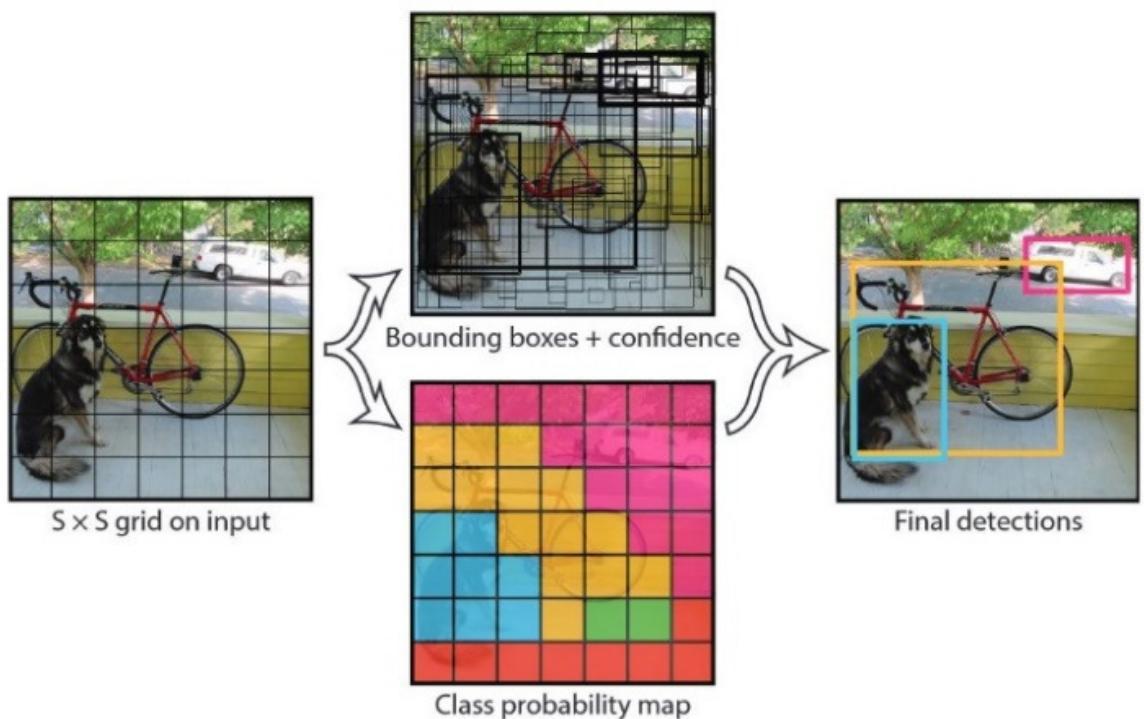


Figura 4 Grila și formarea predicției (imagine preluată din [28])

Cea de-a doua secțiune este formată dintr-un vector ce conține o distribuție de probabilități softmax, care va prezice clasa obiectului detectat de către pătratul curent, exemplificată în Figura 4 de mai sus pentru întreaga grilă sub denumirea „Class Probability Map”.

Arhitectura „Single Shot Multibox Detector” (SSD) [26] îmbunătățește precizia modelului prin introducerea conceptului de „default boxes”. Spre deosebire de arhitectura YOLO [27], ce folosește o singură grilă formată din pătrate de dimensiune fixă, SSD introduce mai multe grile ce conțin dreptunghiuri de dimensiuni variate. Astfel, *offset*-urile de scalare și poziționare se vor găsi întotdeauna într-un interval mai restrâns, deoarece sunt șanse mari să existe un dreptunghi în „default boxes” ce are dimensiuni apropriate de dimensiunile obiectului detectat în imagine. Acest fapt este exemplificat în Figura 5 de mai jos unde rețea a ales „default box”-ul colorat cu roșu, ce avea dimensiunile cele mai apropriate pentru detecția câinelui.

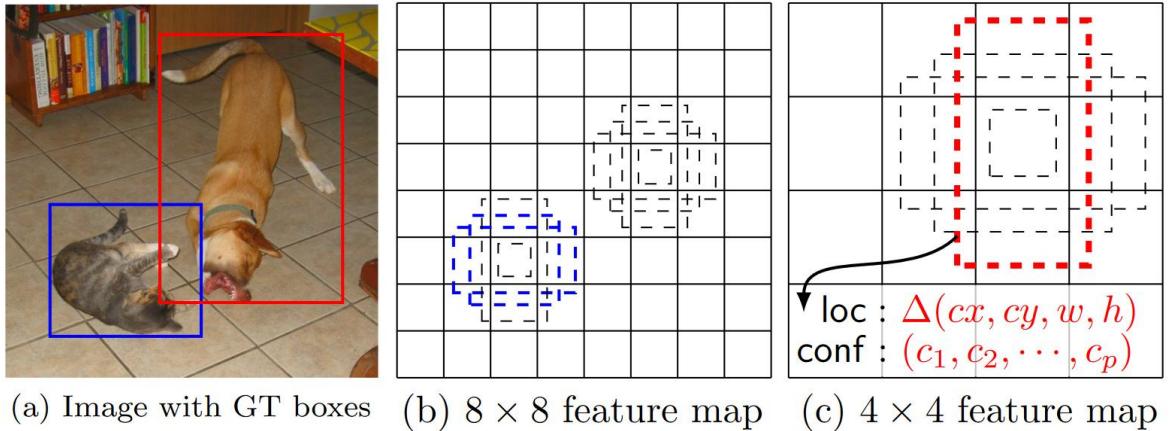


Figura 5 Exemplu de activare a „default box”-urilor preluat din [26]

Pentru a genera *offset*-urile necesare acestui număr mare de dreptunghiuri adăugate, rețea neurală va extrage atrbute din mai multe straturi de pe parcurs, adăugând bifurcații (exemplificate în Figura 6): straturi de conveție asupra cărora nu este aplicată o funcție de activare, ce pleacă din straturile aflate la mijlocul rețelei și din care nu mai pornesc alte conveții. Aceste straturi vor reprezenta *output*-ul rețelei și vor fi concatenate pentru a genera rezultatul final.

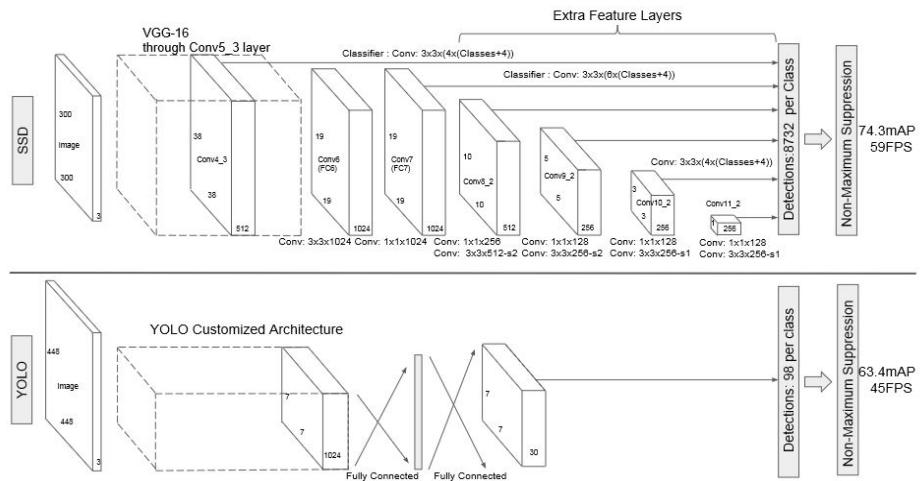


Figura 6 Comparație între arhitecturile SSD [26] și YOLO [27]

(imagine preluată din [26], detalii în anexa 9.4)

Dimensiunea straturilor din *output* corespunde dimensiunilor fiecărei grile. Pentru a exemplifica putem presupune că avem prima grilă de dimensiune 12×12 și presupunem că fiecare dreptunghi are asociată o distribuție de probabilitate softmax ce reprezintă 5 clase de obiecte (4 clase + clasa fundal) și parametrii *offset* asociați *cx*, *cy*, *w*, *h* (exemplifică în Figura 5). Dacă presupunem că fiecare celulă conține 4 dreptunghiuri de aspecte diferite, conform figurii de mai sus (Figura 5), vom avea un strat de convoluție pentru *output* de dimensiuni $12 \times 12 \times (4 \times (5 + 4))$. Dacă rețeaua mai conține și straturile pentru output ce corespund unor grile 6×6 , 3×3 și 1×1 , vom avea un strat final de concatenare ce are dimensiunea $(12 \times 12 + 6 \times 6 + 3 \times 3 + 1 \times 1) \times 36$.

Spre deosebire de exemplul de mai sus în practică se utilizează grile de dimensiuni mai mari și mai multe rate de aspect, deci numărul final al dreptunghiurilor prezise de către rețea va fi foarte mare. Pentru a forma rezultatul final afișat utilizatorului, format numai din cele mai bune dreptunghiuri prezise, va fi folosit un algoritm de tipul „Non Max Supression” (abreviat NMS) ce va separa aceste predicții în funcție de valoarea maximă din distribuția de probabilitate asociată, clasa prezisă și poziția relativă a acestora (dintre dreptunghiurile suprapuse se va extrage numai cel cu probabilitatea maximă). O implementare eficientă a acestui algoritm de post procesare este esențială dacă se dorește o viteză ridicată de detecție, în special în cazul rulării acestei arhitecturi pe un stream video.

Toate arhitecturile menționate implică rețele neurale adânci cu un număr foarte mare de straturi și parametri antrenabili. Pentru a antrena cu succes aceste arhitecturi este esențial ca primele straturi să fie inițializate folosind ponderile primelor straturi din rețele neurale adânci pentru clasificare de imagini precum VGG [29] (în cazul SSD [26]). Ponderile acestor straturi nu vor fi actualizate pe parcursul antrenării, scopul lor este de a extrage atrbute de nivel scăzut ce au fost învățate folosind seturi de date cu dimensiuni foarte mari (14 milioane de imagini pentru setul de date ImageNet [30] în cazul VGG [29]) și care apoi vor putea fi folosite de către noile straturi pentru a compune obiectele detectate.

Modelul TextBoxes [31] pleacă de la arhitectura SSD [26] adăugând câteva modificări importante pentru a permite recunoașterea regiunilor ce conțin text, folosind aceleași principii de bază ca și în cazul modelelor pentru detecție a obiectelor descrise mai sus.

Conceptul de „default box” este păstrat, însă „default box”-urile sunt formate numai din pătrate sau dreptunghiuri ce au lățimea mai mare decât înălțimea, mai specific acestea au 6 rate de aspect diferite 1, 2, 3, 5, 7 și 10. Însă numărul de dreptunghiuri asociate unei celule din grilă este de fapt 18, deoarece sunt generate alte 6 dreptunghiuri de aceeași dimensiune deasupra și dedesubtul primelor 6, pe axa O_x , folosind un deplasament egal cu jumătate din înălțimea unei celule din grilă. Acest deplasament este folosit cu scopul de a acoperi întreaga suprafață a imaginii, dat fiind faptul că se folosesc dreptunghiuri foarte late deci se pierde din densitatea „default box”-urilor pe verticală.

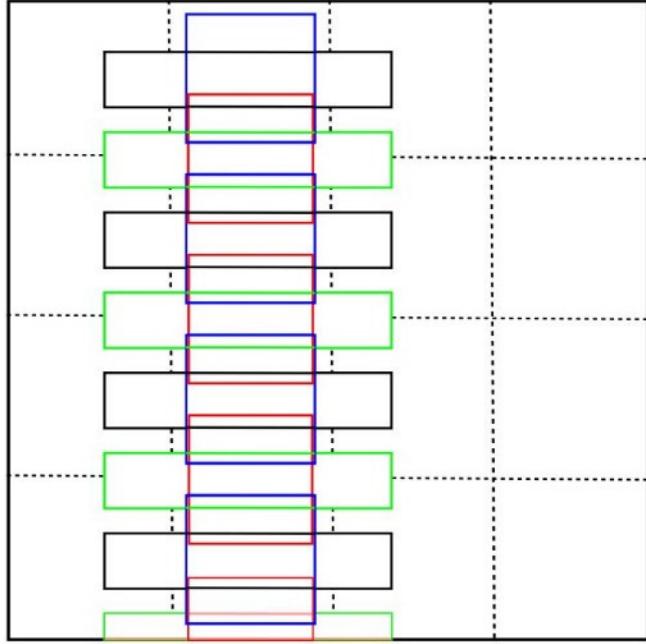


Figura 7 Extinderea „default box”-urilor pe întreaga suprafață a imaginii
în lucrarea TextBoxes [31]

În Figura 7 de mai sus autorii lucrării TextBoxes [31] au exemplificat algoritmul de construire a „default box”-urilor pentru ratele de aspect 1 și 5. Dreptunghiurile de culoare albastră și neagră reprezintă pozițiile originale, fără *offset*, pentru ratele de aspect 1 respectiv 5. Dreptunghiurile de culoare roșie și verde reprezintă cel de-al doilea set de „default box”-uri aplicând *offset*-ul negativ pe axa O_x .

Funcția de cost folosită este identică cu cea din modelul SSD [26]. Aceasta este formată din 2 elemente „confidence loss” și „location loss”:

$$L = \frac{1}{N} (L_{Conf} + \alpha L_{Loc}) \quad (1)$$

a) L_{Conf} („Confidence loss”) este funcția de eroare co-entropie („cross entropy loss”), aplicată peste vectorul softmax de probabilități asociat fiecărui dreptunghi în parte ce indică clasa prezisă de acesta. Singura diferență aici față de modelul SSD este că nu există decât 2 clase: text și fundal.

$$L_{Conf} = - \sum_i y_i \log(f(x)_i) \quad (2)$$

b) L_{Loc} („Location loss”) este calculat folosind funcția de eroare Huber („Smooth L1 Loss”), între indicii de translație și scalare preziși pentru un dreptunghi și cei reali, calculați pe baza adnotărilor din setul de date. Se calculează exclusiv pentru „default box”-urile ce au prezis clasa text, cele care au prezis clasa background sunt ignorate. Se folosește funcția Huber în loc de funcția quadratică („Mean Squared Error”) pentru rezistență mai mare la valori

extreme. Constanta α are rolul de a ajusta ponderea „location loss”-ului pentru ecuația finală, însă în lucrarea originală valoarea sa a fost setată la 1, deci poate fi ignorată.

$$L_{Loc} = \begin{cases} \frac{1}{2}(y - f(x))^2, & |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & |y - f(x)| > \delta \end{cases} \quad (3)$$

Întreaga ecuație se împarte la N, ce reprezintă numărul total de dreptunghiuri din „ground truth” care trebuiau să prezică clasa text (adnotările acestora au valoarea „confidence” egală cu 1).

Initial în procesul de antrenare rețeaua va genera foarte multe predicții negative (dreptunghiuri poziționate în zone ce nu conțin text). Din această cauză va fi necesară (la fel ca și în cazul modelului SSD [26]) folosirea noțiunii de **Hard Negative Mining**. La fiecare pas, pentru calcularea funcției de eroare descrise mai sus vor fi folosite numai o treime din exemplele negative, luate în ordine descrescătoare după magnitudinea erorii între distribuția de probabilitate prezisă și cea reală (vor avea prioritate dreptunghiurile care au prezis clasa incorectă cu un „confidence” cat mai mare).

Arhitectura TextBoxes++ „A Single-Shot Oriented Scene TextDetector” [16] revine asupra modelului TextBoxes [31] inițial și adaugă modificări pentru a putea detecta și extrage cu precizie text orientat în diferite unghiuri față de planul imaginii/camerei. Structura rețelei neurale rămâne aproape neschimbată, după cum se poate observa în Figura 8, diferențele principale fiind în cadrul secțiunilor de pre și post procesare.

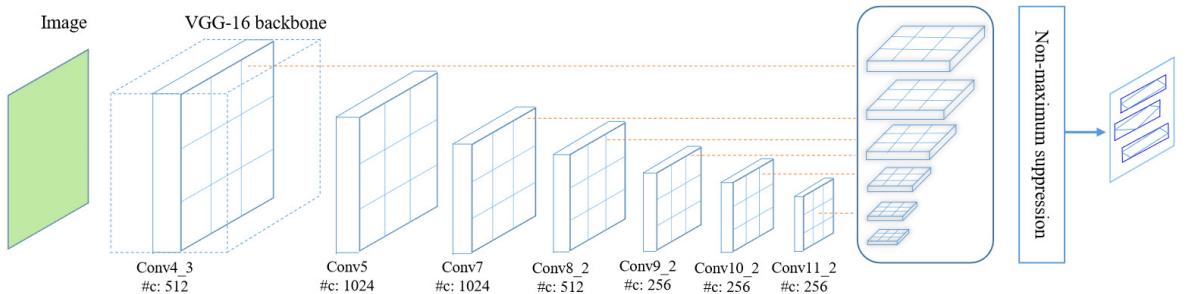


Figura 8 Structura rețelei neurale, extrasă din lucrarea TextBoxes++ [16]

În timp ce modelul TextBoxes [31] putea extrage numai dreptunghiuri, TextBoxes++ [16] introduce conceptul de „Rotated Bounding Box” înlocuind dreptunghiurile detectate (spre exemplu cel de culoare verde din Figura 9) cu paralelograme (spre exemplu cel de culoare galbenă din Figura 9). Astfel este extras foarte puțin fundal din imaginile ce conțin text rotit în unghiuri arbitrară, un fapt deosebit de util în problema detecției datelor de expirare.

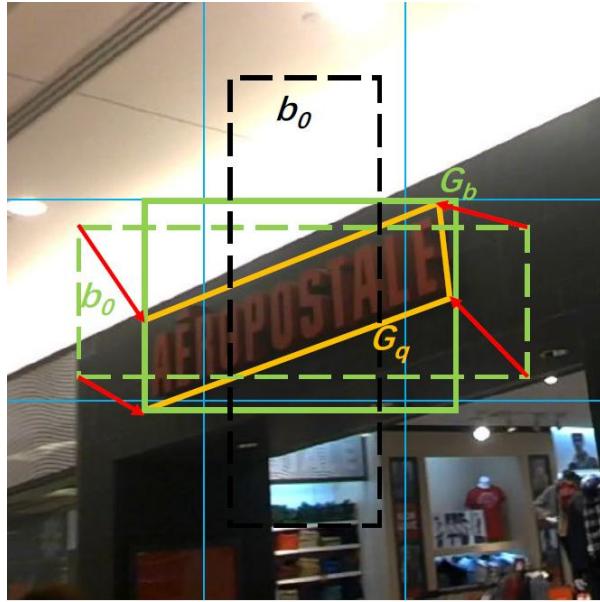


Figura 9 „Rotated Bounding Box” [16]

Două etape foarte importante atât pentru TextBoxes++ [16] cât și pentru celelalte modele sunt **pre procesarea și post procesarea adnotărilor**. În cazul TextBoxes [31], SSD [26], YOLO [27] trebuiau calculate cele 4 coordonate de poziționare și scalare prezentate mai sus (Figura 3). Pentru paralelogramele din arhitectura TextBoxes++ [16] se vor calcula *offset*-urile dintre colțurile stânga și dreapta sus ale paralelogramului și cele ale dreptunghiului din „default boxes” plus înălțimea finală a paralelogramului raportat la înălțimea dreptunghiului. În cadrul pasului de preprocessare este necesară mai întâi generarea tuturor dreptunghiurilor din „default boxes”. Apoi, pentru fiecare imagine în parte, pentru adnotările fiecărei zone de text (în cadrul setului de date ICDAR [24] formate din cele 4 perechi de coordonate ale colțurilor unui patrulater) vor trebui calculate *offset*-urile necesare formării paralelogramelor, ce reprezintă *output*-ul dorit al rețelei. Un „default box” va fi asociat unei regiuni adnotate numai dacă indicele „Intersection Over Union” (aria zonei de intersecție între dreptunghiul ce va trebui prezis și cel din grila „default boxes” supra aria reunii acestora) dintre acestea depășește o anumită valoare *threshold*, altfel probabilitatea „confidence” a „default box-ului” va lua valoarea 0. După antrenare, pentru a putea utiliza rețea, *offset*-urile precise trebuie transformate înapoi în patrulatere, care apoi vor trece prin algoritmul de „Non Max Supression” prezentat mai sus.

Se pot observa câteva posibile modificări pentru a adapta modelul TextBoxes++ [16] la problema detecției datelor de expirare. Putem folosi numai *output*-ul „default box”-urilor ce au o rată de aspect similară cu datele de expirare și putem ajusta hiperparametri precum *threshold*-ul „Intersection Over Union” prezentat mai sus, *threshold*-urile folosite în algoritmul „Non Max Supression”, etc. pentru a maximiza precizia pe acest set de date.

4.2 Extragerea textului

Regiunile de interes extrase la pasul anterior sunt introduse ca *input* către o rețea neurală recurrentă, specializată în detecția secvențelor, pentru a obține textul conținut.

4.2.1 Arhitectura CRNN

În cadrul acestui proiect am folosit cu precădere arhitectura CRNN [20] deoarece aceasta este folositoare foarte des în practică în combinație cu modele precum TextBoxes++ [16], East Text Detector [32], SegLink [19], etc. Chiar autorii acestora folosesc CRNN [20] pentru a demonstra rezultatele modelelor lor pe problema recunoașterii textului „end to end”.

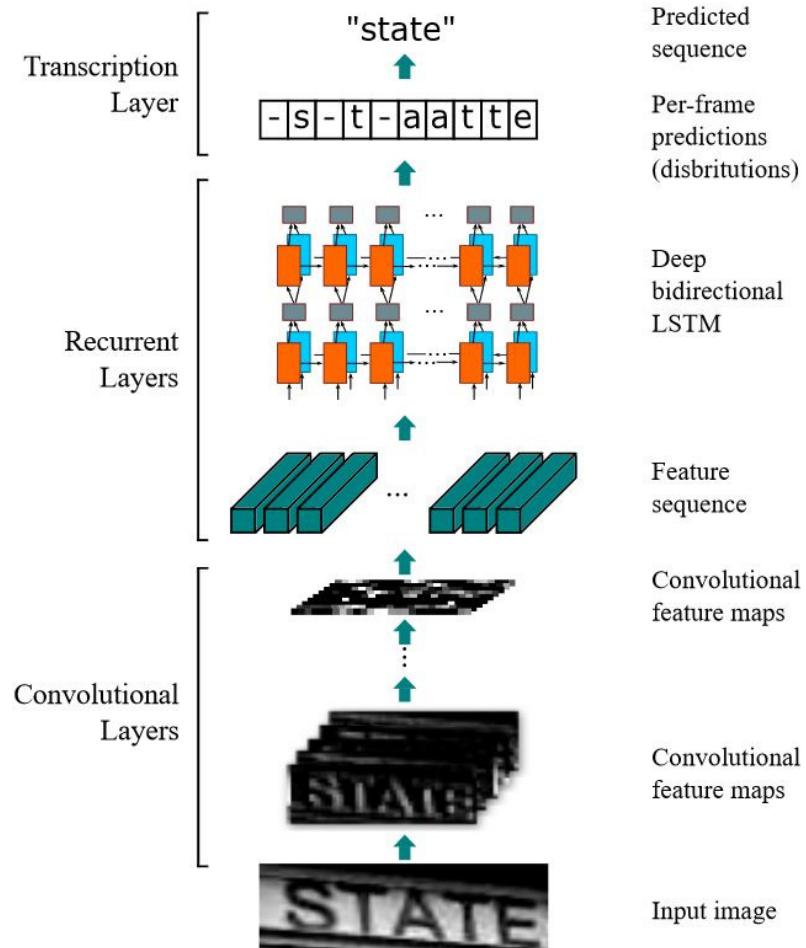


Figura 10 Arhitectura rețelei neurale, extrasă din lucrarea CRNN [20]

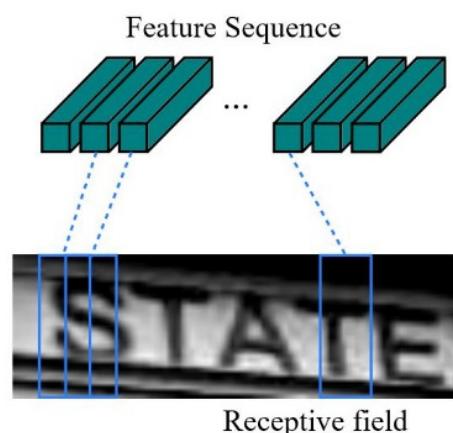


Figura 11 Coloanele ce reprezintă *input*-ul blocurilor de memorie [20]

Arhitectura modelului poate fi observată în Figura 10. Prima parte a CRNN este formată din straturi de conoluție, pentru a extrage atribute de nivel scăzut din imagine. Output-ul acestora este apoi împărțit în coloane, iar fiecare coloană este introdusă într-un bloc format din celule de tip „Long Short-Term Memory” (LSTM) [33] pentru a extrage caracterul conținut (Figura 11). Toate detecțiile vor fi apoi filtrate pentru a elibera eventuale duplicate sau spații libere în exces.

Modelul CRNN [20] este foarte flexibil, astfel încât autorii acestuia au putut să îl utilizeze și pentru detecția notelor în partituri muzicale, deci adaptarea pentru detecția caracterelor din datele de expirare nu ar trebui să reprezinte o problemă. Pentru a extrage textul datelor de expirare folosind această arhitectură este necesară numai reantrenarea folosind imagini ce conțin font de tip „dot matrix” (vor fi extrase patrulaterele adnotate din imaginile setului de date explicit în secțiunile 4.3 și 5.3).

4.3 Construirea unui set de antrenare pentru detecția datelor de expirare

Pe măsură ce rețelele neurale adânci devin din ce în ce mai populare și sunt necesare din ce în ce mai multe date pentru antrenarea lor, au început să apară și foarte multe metode de generare sintetică a datelor de antrenare. Un bun exemplu ar putea fi considerat simulatorul AirSim [34] oferit gratuit de către Microsoft, ce simulează folosind un motor grafic 3D seturi de date de antrenare pentru drone sau mașini autonome. Acesta captează imagini folosind camere virtuale, situate în spațiul 3D, ai căror parametri au fost ajustați pentru a simula corespondentul lor hardware din viață reală (poziție în relație cu vehiculul, unghi de deschidere, etc.).

Pentru antrenarea rețelelor menționate la pașii 4.1 și 4.2, am construit un set de date format atât din imagini reale cu date de expirare de pe produse cât și imagini generate sintetic folosind o aplicație dezvoltată într-un motor grafic 3D, pentru a simula cât mai bine caracteristicile observate în imagini reale privind iluminarea, forma materialelor pe care sunt acestea imprimate etc.

Pentru generarea imaginilor sintetice va fi necesară realizarea unei scene 3D în care vor fi folosite diferite surse de lumină, diferite poziții ale camerei virtuale, diferite materiale și obiecte 3D precum conserve, cutii, sticle, etc. pentru a simula o situație reală. Datele de expirare vor fi generate sub forma unor texturi folosind fonturi specifice acestora (exemplu cele cu format „dot matrix”) și pentru mai mult realism vor fi aplicate obiectelor sub forma unor „decal”-uri cu volum de proiecție, pentru a simula modul de funcționare al imprimantelor care aplică acest text pe ambalajele produselor în viață reală.

La antrenare imaginile din setul de date vor fi augmentate, pentru a preveni fenomenul de *overfitting*, utilizând transformări precum rotație aleatoare, translație aleatoare, adăugare de zgomot, etc.

Setul de date prezentat este folosit numai pentru „fine-tuning”, fiind mai întâi necesar un pas de pre-antrenare datorită adâncimii mari și numărului foarte mare de parametri ai rețelelor utilizate. Pentru pre-antrenarea modelelor din această lucrare am utilizat două seturi de date foarte populare în domeniul recunoașterii de caractere ce sunt și recomandate în lucrările TextBoxes++ [16], SegLink [19], etc. :

- a) **Setul de date ICDAR** [24, 35] este denumit după una dintre mai importante conferințe în domeniul recunoașterii optice a caracterelor („International Conference on Document Analysis and Recognition”). În cadrul acestei conferințe sunt lansate competiții, însotite și de câte un set de date, pentru a susține dezvoltarea de noi modele în domeniul recunoașterii optice a caracterelor. Cele mai relevante seturi de date pentru proiectul de față și arhitecturile prezentate în secțiunile 4.1 și 4.2 sunt cele din categoria „Scene Text Detection”, în special versiunile:
- **2013-2015 „Focused Scene Text”**, ce conține imagini cu firme de magazine, afișe stradale, semne de circulație, etc. captate intenționat de către oameni folosind o cameră foto
 - **2015 Incidental Scene Text**, conține imagini captate folosind ochelari Google Glass în timpul mersului prin oraș, în mijloace de transport în comun, etc. fără a fi introduse în vederea camerei intenționat de către purtător.
 - **2017 Multilingual Scene Text**, este similar cu versiunea 2013 „Focused Scene Text” de mai sus, însă de data aceasta au fost adăugate mai multe imagini, preluate de la utilizatori din mai multe țări, pentru a avea text mai în mai multe limbi.

La pre-antrenarea rețelei TextBoxes++ [16] am combinat toate aceste 3 seturi de date, obținând în final un set de aproximativ 6000 de imagini. Câteva exemple pot fi observate în Figura 12.



Figura 12 Imagini din setul de date ICDAR [24, 35]

Pentru mai multă consistență, la adnotarea setului meu de antrenare ce conține date de expirare am folosit un fișier tip CSV, similar cu cel folosit în setul ICDAR [24].

- b) **Setul de date SynthText** este generat după procedeul descris în lucrarea „Synthetic data for text localisation in natural images” [36] și conține 800 000 de imagini generate sintetic prin suprapunerea de text în diferite formate peste o varietate de imagini comune. Pentru pre-antrenarea rețelei TextBoxes++ [16], cu scopul de a reduce timpul necesar

preprocesării și antrenării, am utilizat numai 40% din acest set de date, adică 320 000 imagini alese aleatoriu. Câteva exemple din setul SynthText [36] pot fi observate în Figura 13.



Figura 13 Exemple din setul de date SynthText [36]

4.4 Filtrare și obținerea rezultatului final

Pentru filtrare va fi utilizat un set de expresii regulate ce extrag combinații între cifre („\d”) și cele mai frecvente caractere folosite pentru delimitare (spațiu, punct, linie, etc.) cu scopul de a extrage toate porțiunile de text ce seamănă ca format cu o dată.

Toate porțiunile de text extrase vor fi apoi procesate cu ajutorul unei biblioteci ce poate detecta date calendaristice în mai multe formate (în cazul de față dateutil.parser [37] în Python3) și filtrate logic pentru a separa data de expirare de alte date, precum cea de producție (care va fi o dată anteroară acesteia), numere de lot (care se vor găsi într-un interval complet diferit spre exemplu 4167/45/55), etc.

4.5 Aplicație pentru reducerea risipei alimentare

Pentru a demonstra modul în care algoritmul prezentat în lucrarea de față poate fi utilizat în lumea reală, cu scopul de a combate risipa alimentară, am ales să construiesc o aplicație ce rulează pe un telefon mobil sau pe o placă de dezvoltare.

Aplicația îi va permite utilizatorului să scaneze rapid data de expirare a unui produs folosind camera foto a dispozitivului în corelație cu algoritmul prezentat în această lucrare. Utilizatorul va putea consulta oricând lista cu produsele sale și va primi notificări atunci când acestea ajung aproape de data lor de expirare.

Aplicația va fi formată din *backend*, un REST API ce va rula în cloud, și două aplicații client: prima va rula pe un telefon mobil (în cadrul acestei lucrări am dezvoltat partea de *frontend* pentru sistemul de operare Android), iar cea de-a doua va rula pe o placă de dezvoltare (în implementarea mea am folosit o placă RaspberryPi). Placa de dezvoltare va fi folosită pentru a simula un frigider inteligent, care ajută la scanarea datelor de expirare în momentul în care se introduc noi produse, deci o metodă alternativă pentru a putea capta date în sistem.

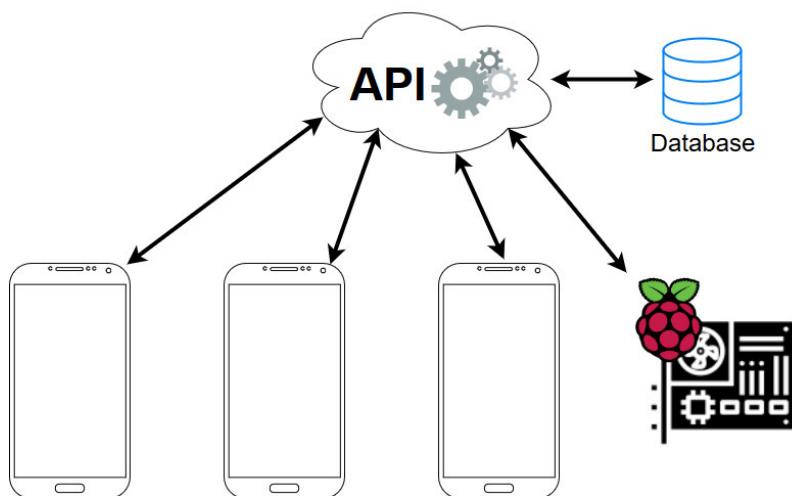


Figura 14 Modul în care vor fi conectate aplicațiile client (*frontend*) și serverul (*backend*)

4.5.1 Frontend

Pentru partea de frontend vor fi necesare următoarele funcționalități principale:

- login / logout
- adăugare / inspectare / ștergere / editare produs
- scanare dată de expirare (aplicația va capta o imagine și o va trimite către serverul din *backend* pentru inferență)

Acste metode vor fi implementate sub forma unor *endpoint*-uri în *backend* și apoi vor fi accesate din codul aplicațiilor client.

De asemenea am adăugat și posibilitatea de a scana codul de bare al produsului (pasul de scanare va fi executat local pe dispozitiv). Astfel, dacă acesta există deja în baza de date *OpenFoodFacts* [38], aplicația va putea completa automat numele produsului și în plus alte informații opționale (imagine, descriere, etc.). Introducerea unui nou produs în aplicație este foarte rapidă, fiind necesară numai preluarea a două imagini folosind camera foto a dispozitivului: scanarea codului de bare și scanarea datei de expirare (procesul este exemplificat în Figura 15).

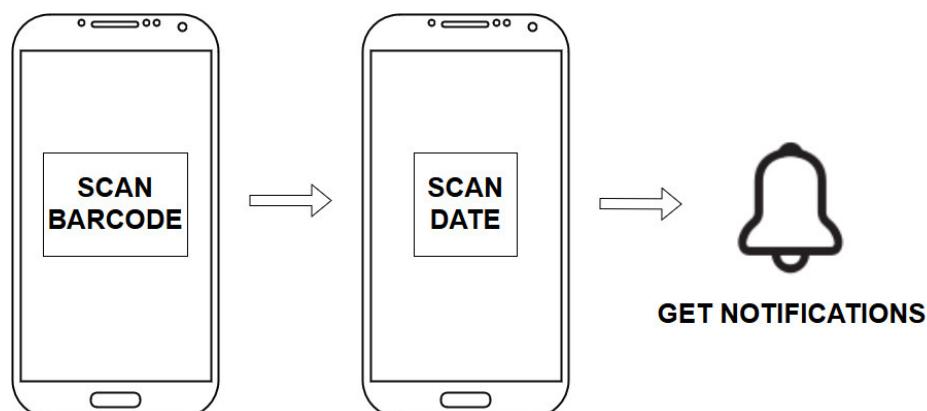


Figura 15 Introducerea unui nou produs folosind codul de bare

Aplicația va putea accesa motorul *text-to-speech* al dispozitivului și asistentul vocal (în cazul Android „Google Assistant”) pentru a livra date de expirare în format audio persoanelor cu deficiențe de vedere.

4.5.2 Backend

Partea de *backend* a aplicației va trebui să permită existența mai multor utilizatori și posibilitatea de a stoca lista de produse a fiecăruiu într-o bază de date. De asemenea vor trebui implementate *endpoint*-urile precizate mai sus pentru facilitățile de *login/logout*, managementul produselor din lista utilizatorului și scanarea datelor de expirare.

Pentru a susține funcționalitățile de bază precizate, *backend*-ul aplicației va folosi o bază de date relațională, a cărei structură este detaliată în Figura 16.

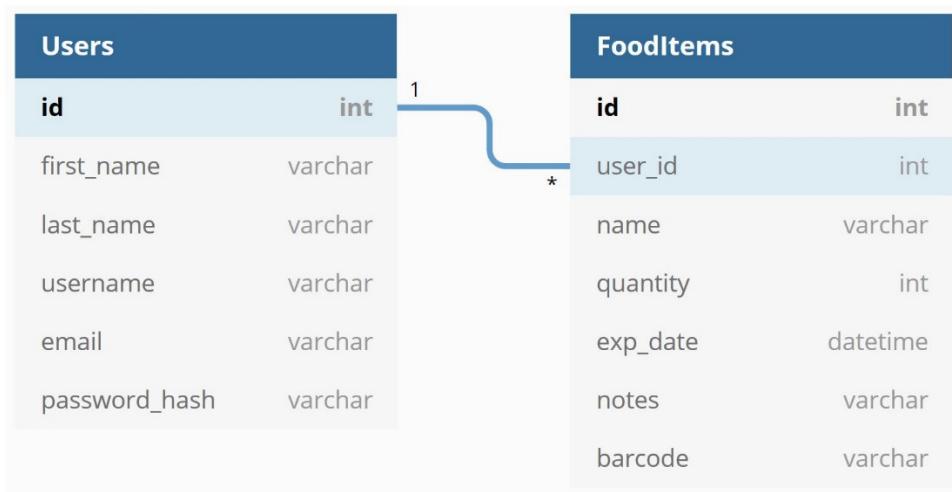


Figura 16 Diagrama entitate-asociere a bazei de date

5 DETALII DE IMPLEMENTARE

5.1 TextBoxes++

Pentru a avea o înțelegere cât mai bună a arhitecturii TextBoxes++ [16], care este folosită pentru extragerea zonelor cu text și probabil este cea mai importantă parte a proiectului, am decis să implementez de la zero acest model folosind Tensorflow.

Faptul că nu am preluat o implementare existentă mi-a oferit oportunitatea de a folosi cele mai noi tehnologii în domeniu pentru a putea maximiza viteza de antrenare a modelului. Această eficiență a fost absolut esențială, având în vedere că autorii lucrării au folosit un GPU Nvidia GTX Titan XP și antrenarea modelului a durat 2 zile [16].

5.1.1 Utilizarea tf.data pentru eficiență maximă la antrenare

Plăcile grafice actuale pot procesa un volum foarte mare de exemple în paralel în cadrul procesului de antrenare, astfel antrenarea unui singur *batch* durează foarte puțin. Asigurarea acestui flux masiv de date pentru a nu avea „*timpi morți*” în care acceleratorul grafic să aștepte date noi este esențială pentru a obține performanță de antrenare maximă.

În cazul versiunilor mai vechi ale Tensorflow bucla principală de antrenare urmează următorul tipar:

```
for i in range(max_iterations):
    crt_batch, crt_batch_labels = next(training_data_generator)
    sess.run(optimizer, feed_dict={data: crt_batch, labels: crt_batch_labels})
```

Utilizarea „*feed_dict*” pentru date de dimensiune mare este descurajată în versiunile recente Tensorflow, deoarece acest cod funcționează după un procedeu pas cu pas, ilustrat în Figura 17: în cadrul generatorului sunt extrase exemplele de pe disc, se preprocesează aceste exemple, apoi folosind „*feed_dict*” se încarcă în memoria plăcii grafice exemplele de antrenare pentru *batch*-ul curent, se rulează modelul, apoi se așteaptă rezultatul, se generează următorul *batch*, se încarcă din nou în memoria GPU-ului, se așteaptă rezultatul și.a.m.d.

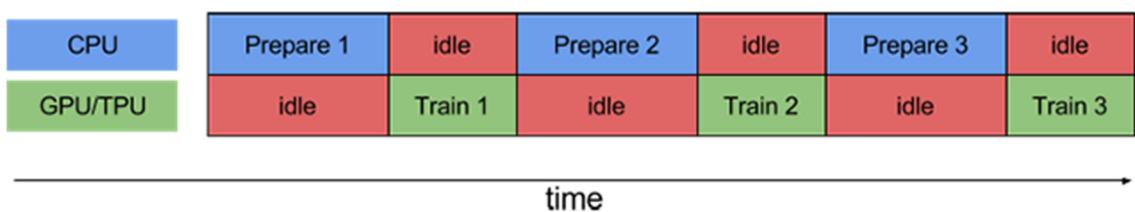


Figura 17 Funcționarea unui *pipeline* obișnuit ce utilizează „*feed_dict*” [39]

Această abordare este ineficientă din două puncte de vedere:

- nu există un mecanism bine stabilit pentru *caching*, deci la fiecare pas placa grafică așteaptă încărcarea datelor de pe disc și preprocesarea acestora, în plus față de

copierea propriu zisă din RAM către VRAM, costul din punct de vedere al timpului de rulare fiind foarte mare

- b) Plăcile grafice actuale, precum cele produse de către Nvidia ce suportă tehnologie CUDA versiunea 8 sau mai mare, pot încărca date din memoria RAM în memoria plăcii grafice (VRAM) în mod asincron folosind operația „cudaMemcpyAsync()” [40], în timp ce rulează alte operații pe procesorul grafic, și de asemenea suportă mecanisme foarte complexe de *caching*, *prefetch*, etc. (acestea sunt detaliate în articolul [40])

API-ul **tf.data** [39] propune o nouă structură, pe care am implementat-o în acest proiect, în care se utilizează conceptul de *pipelining*. Astfel CPU poate face *prefetch* datelor necesare pentru pașii următori, poate menține un buffer de exemple de antrenare în memoria RAM pentru a asigura viteza de rulare în ciuda vitezei reduse de accesare a datelor de pe disc și poate deja să copieze în memoria plăcii grafice datele necesare în timp ce aceasta rulează pașii anteriori. Procesul este ilustrat în Figura 18.



Figura 18 Funcționarea unui *pipeline* eficient ce utilizează tf.data [39]

Recent Google a adăugat în platforma sa *cloud* un accelerator bazat pe un *ASIC* („Application Specific Integrated Circuit”) denumit *TPU* („Tensor Processing Unit”), proiectat din punct de vedere electronic special pentru a maximiza viteza operațiilor folosite la antrenarea rețelelor neurale adânci. Viteza acestora este net superioară plăcilor grafice curente, spre exemplu este posibilă antrenarea întregului model ResNet50 în numai 2.2 minute [41]. Pentru a putea utiliza acest accelerator este obligatoriu să se folosească API-ul **tf.data** pentru a putea livra toate datele necesare eficient și în timp util.

Conform analizei din utilitarul Tensorboard, 98 % din operațiile prezente în implementarea mea a modelului TextBoxes++ [16] pot rula pe *TPU*, deci întregul model poate beneficia de o viteză mult mai mare în cazul în care este rulat pe un accelerator de acest tip.

5.1.2 Crearea fisierelor TFRrecords

Deși API-ul **tf.data** prezentat mai sus poate prelua date în aproape orice format, folosind vectori *numpy* și metoda *Dataset.from_tensor_slices()*, se preferă în practică (și este recomandată și în documentația oficială) folosirea unui fișier binar special *.tfrecords* pentru a obține eficiență maximă. Putem aplica încă de la scrierea acestui fișier preprocesări asupra datelor pentru a reduce din *overhead* la rulare, în special în cazul în care vom face multe rulări ale modelului pentru a ajusta hiperparametrii, iar setul de date rămâne neschimbat. De asemenea datele vor fi deja serializate în acest fișier, gata să fie citite și introduse ca input pentru model.

Fișierele .tfrecords suportă și tehnici de compresie, precum gzip, zlib, etc. pentru a reduce spațiul ocupat pe disc. În cadrul acestui proiect compresia a fost esențială deoarece *tensorii* generați pentru „default boxes” [16] au dimensiuni foarte mari și conțin foarte multe elemente care se repetă: spre exemplu *tensor-ul „confidence”* conține în general 0-uri și numai câteva valori 1 pe pozițiile în care se găsește text în imagine.

Cel mai important pas în implementarea modelului și cel care a consumat cel mai mult timp a fost fără îndoială pasul de preprocesare pentru generarea „default box”-urilor [16], acesta având scopul de a calcula *offset*-ul dintre „default box”-uri și poligoanele din „ground truth” ce indică poziția textului, aceste *offset*-uri reprezentând chiar *output*-ul dorit al rețelei neurale. Prin urmare, la final este necesar și un pas de decodare, în care sunt calculate poligoanele prezise de către rețea pe baza valorilor *offset* obținute la output.

Pentru operațiile de codare/decodare (detaliate în secțiunea 4.1.1) am folosit biblioteca *numpy* cu scopul de a vectoriza pe cât posibil numărul mare de calcule necesare.

Pentru a genera fișierul .tfrecords folosind un singur *thread*, timpul estimat pentru 30% din imaginile setului de date SynthText [36] era inițial 3 zile. Pentru a genera mai rapid acest fișier am considerat folosirea tuturor *thread*-urilor disponibile ale procesorului (8 pe mașina de calcul utilizată pentru acest experiment).

Prima implementare naivă a fost să folosesc mai multe fire de execuție, unde fiecare fir va procesa câte o imagine și apoi va folosi un *lock* peste obiectul *tf.python_io.TFRecordWriter* pentru a scrie în fișier. Această abordare s-a dovedit a fi un eșec, diferența de viteză nefiind semnificativă. Problema principală este că operațiile din procesul de codare sunt atât de rapide datorită vectorizării, încât timpul de calcul este mai mic decât timpul pe care îl așteaptă un fir de execuție ca să poată obține acces de scriere în fișier.

Pentru a rezolva această problemă m-am folosit de faptul că *tf.records.Dataset* poate citi simultan din mai multe fișiere .tfrecord pentru a obține un singur set de date folosind metoda „list_files()”. Am generat câte un fișier nou pentru fiecare fir de execuție, eliminând problema sincronizării (aceasta a trecut în sarcina sistemului de operare, iar datorită vitezei relativ mari de scriere pe disc este neglijabilă ca timp), iar în acest mod am putut genera fișierele .tfrecords pentru 40% din setul de date SynthText [36] (400 000 de imagini) în numai 3 ore.

5.2 CRNN

Pentru extragerea textului din zonele de interes am utilizat arhitectura CRNN [20] detaliată în secțiunea 4.2. Deoarece nu erau necesare foarte multe modificări am folosit o implementare open source disponibilă pe GitHub [21]. Acest model vine deja și cu un set de ponderi pre-antrenate pe setul de date SynthText [36]. Am încărcat aceste ponderi și am reantrenat modelul folosind setul meu propriu de date (detaliat la pasul 4.3).

5.3 Construirea unui set de antrenare pentru detecția datelor de expirare

5.3.1 Imaginele reale

Pentru setul de date ce conține imagini reale am folosit aproximativ 660 de fotografii făcute cu ajutorul camerei telefonului mobil, ce conțin date de expirare ale unei game variate de produse alimentare. Pentru a adnota aceste imagini am utilizat platforma online labelbox.com [42] unde am încărcat imaginile și apoi am definit o interfață minimală pentru adnotarea acestora. Toate imaginile au fost adnotate manual utilizând interfața din Figura 19.

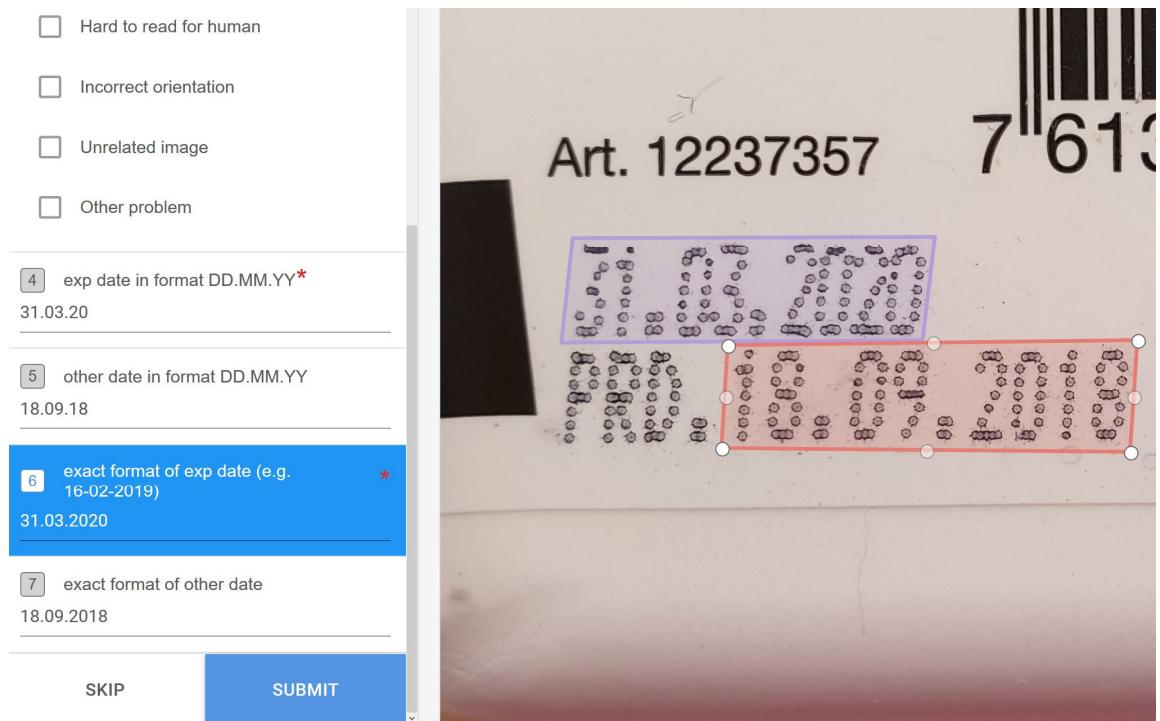


Figura 19 Interfață creată folosind Labelbox [42]

După precizarea tipurilor de obiecte necesare, aplicația Labelbox [42] permite definirea poligoanelor de segmentare peste imagine cu ajutorul mouse-ului. De asemenea pot fi adăugate câmpuri extra, ce pot fi marcate ca obligatorii sau nu. Pentru acest proiect am ales să definesc următoarele câmpuri: data exactă într-un format fix, pentru a putea testa la final eficiența pasului de filtrare (Secțiunea 4.4) și textul exact prezent în imagine ce definește această dată pentru a putea antrena pasul de recunoaștere optică a caracterelor propriu zis (exemplu rețeaua CRNN [20] secțiunea 4.2). Am dublat câmpurile prezentate pentru cazul în care imaginea conține și data fabricației unui produs, ce este identică cu cea de expirare ca trăsături și deci utilă pentru antrenarea pasului de segmentare, aceste date putând fi eliminate apoi în cadrul pasului de filtrare. Am adăugat și câteva *checkbox*-uri în interfață pentru a putea semnala imagini cu probleme care ulterior au fost reintroduse sau eliminate complet din setul de date.

Conținutul câmpurilor și coordonatele poligoanelor de segmentare pot fi descărcate la final de pe platformă sub forma unui fișier JSON. Am scris un script Python pentru a transforma

fișierul JSON generat în fișiere CSV cu aceeași formatare ca cele din setul de date ICDAR [36] pentru a nu aduce modificări *pipeline*-ului de intrare și generare a fișierelor .tfrecords.

Am utilizat 10% din imaginile obținute la final (66 de imagini reale) ca set de validare. Acestea nu au fost introduse în procesul de antrenare și au fost utilizate numai pentru evaluarea performanțelor din capitolele 3 și 6.

5.3.2 Imaginile generate sintetic

Pentru generarea setului de date sintetic am pornit prin a descărca câteva fonturi freeware de pe internet ce conțin caractere formate din puncte sau caractere modelate după imprimante termice și mașini de scris (exemple: [43], [44], [45]). Am importat aceste fonturi într-un script Python și am folosit biblioteca *date* și biblioteca *PIL* pentru a genera imagini .png cu fundal transparent care conțin date în diferite formate. Câteva exemple pot fi observate în Figura 20.

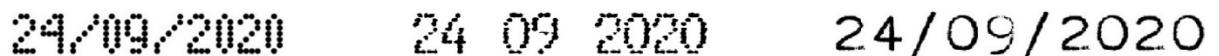


Figura 20 Exemple text generat sintetic folosind fonturile [43], [44], [45]

În continuare am importat imaginile generate în motorul grafic Unity3D sub formă de texturi transparente.

Am descărcat de pe site-ul free3d.com câteva modele .obj precum doze de băuturi, cutii, sticle, conserve, etc. cu ajutorul cărora am creat o scenă nouă, în care am poziționat aceste obiecte și câteva lumini direcționale, apoi am scris un script C# care randomizează intensitatea și poziția luminilor.

Am adăugat pe suprafața fiecărui obiect o “ancoră” (un GameObject ce conține numai un element Transform). Poziția acestei ancore va deveni ținta unui obiect DecalProjector ce folosește un volum de proiecție (Figura 21) pentru a simula imprimarea datei pe suprafața neregulată a obiectului, proiectând pe rând texturile ce conțin date, generate în Python la pasul anterior (Figura 20), peste suprafața modelelor 3D.

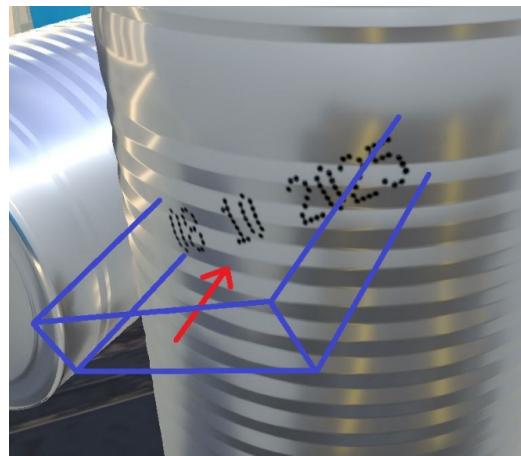


Figura 21 Proiectorul și volumul de proiecție asociat

Am pus în scenă o cameră ai cărei parametri (tipul de proiecție, unghiul de deschidere etc.) au fost ajustați pe cât posibil pentru a simula camera unui telefon mobil actual.

Am scris apoi un script C# atașat la camera virtuală, pentru a aduce automat în câmpul vizual date de expirare a fiecărui obiect folosind ancorele definite mai devreme pe suprafața acestora. Odată ce zona este adusă în câmpul vizual al camerei, este apelată o metodă pentru a schimba textura proiectată pe obiect și se salvează imaginea curentă a camerei pe disc. Se calculează apoi coordonatele texturii proiectate și se transformă în sistemul de coordonate al camerei, înmulțind cu matricea de transformare corespunzătoare, pentru a obține coordonatele poligonului ce delimită data de expirare în imaginea finală. Textul din imagine este aflat pe baza numelui fișierului texturii respective (exemplu: 24s09s2020.png unde „s” semnifică caracterul „/”). Coordonatele și textul din imagine sunt la final salvate ca fișier CSV în același format cu setul de date ICDAR [24], pentru a păstra consistența cu setul de imagini reale. Astfel a fost generat un exemplu de antrenare complet sintetic exemplificat în Figura 22.

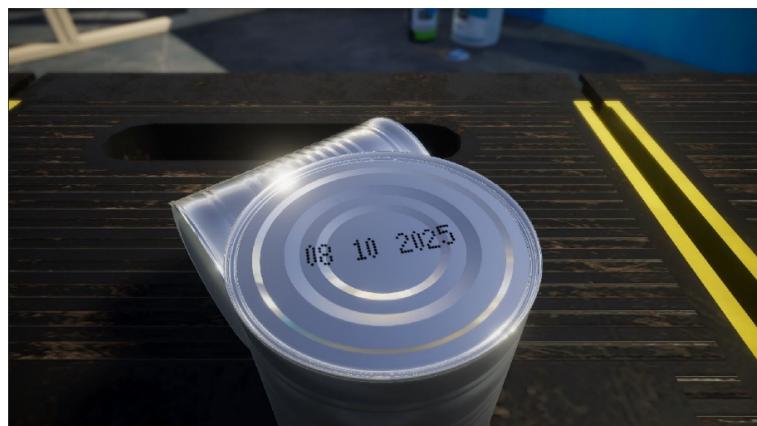


Figura 22 Imagine generată sintetic în spațiul virtual 3D

Pentru a obține rezultate cât mai bune cu privire la iluminare, reflexii, transparența obiectelor, etc. și pentru a simula o cameră reală din punct de vedere al planului de focalizare am utilizat

biblioteca oficială „HD Render Pipeline” din Unity, ce folosește tehnici de ultimă generație în domeniu grafic și se găsește numai în versiune beta la momentul redactării.

5.3.3 Augmentarea imaginilor

Pentru augmentarea imaginilor am utilizat biblioteca *imgaug* [46] ce oferă o mulțime de transformări gata făcute, accesibile prin simpla apelare a unei metode, precum adăugare de zgomot, varierea luminozității și multe altele exemplificate pe pagina oficială. De asemenea prezintă două avantaje importante:

- Posibilitatea de a combina mai multe transformări fie prin înlănțuire fie prin opțiunea „one of” folosind distribuții de probabilitate atât pentru a decide frecvența de apariție cât și pentru a varia aleator parametrii fiecărei transformări în intervalele dorite.
- Calculul automat al noilor poziții pentru punctele de interes din imagine după secvența de transformări utilizată. Astfel putem afla punctele cheie ale poligoanelor ce delimitizează data de expirare în imaginea augmentată.

Deși calculul punctelor cheie se face automat, acestea se pot găsi în exteriorul zonei imaginii, patrulaterele generate pot avea dimensiuni prea mici, unghiul de rotație al patrulaterului poate fi prea mare în cazul în care imaginea nu era dreaptă la început, și.a.m.d. De aceea am scris un script care verifică toate cazarile limită și rulează din nou procesul de augmentare pentru o imagine în cazul în care există criterii ce nu se respectă.

Un exemplu pentru augmentarea unei imagini poate fi observat în Figura 23.



Figura 23 Imaginea originală (stânga) și 4 exemple de augmentare (dreapta)

5.4 Filtrare și obținerea rezultatului final

Pentru filtrare am folosit următorul set de expresii regulate:

```
(?:\d{2}(?:\d{2})?(?: *)[/:\-,_.]?(?: *))?\d{2}(?: *)[/:\-,_.]?(?: *)\d{2}(?:\d{2})?
```

Astfel am putut extrage text în format an/lună/zi, zi/lună/an, lună/an, etc. Rezultatele sunt apoi introduse în *dateutil.parser* [37] folosind opțiunea „fuzzy” pentru a îi permite *parser*-ului să aleagă automat formatul datei introduse. Rezultatele vor fi obiecte de tipul *datetime*, ce vor fi filtrate folosind un interval de 30 de ani față de data curentă pentru a elimina din detecțiile eronate. Datele rămase vor fi sortate și va fi extrasă data cea mai târzie. La final va fi obținut un obiect de tip *datetime*, ce poate fi livrat direct către *frontend*-ul aplicației sau

poate fi folosit pentru a calcula precizia obținută prin compararea datei cu cea din setul de date din secțiunea 4.3.

5.5 Aplicație pentru reducerea risipei alimentare

Aplicația este formată din partea de *frontend* pentru telefon mobil / RaspberryPi și partea de *backend*, arhitectura ei fiind exemplificată în Figura 14 și descrisă în secțiunea 4.5.

5.5.1 Frontend

Pentru partea de *frontend* pe telefonul mobil, am scris o aplicație Java, nativă Android, folosind suita oficială Android Studio. Am compus *request*-urile către *backend* folosind biblioteca Volley [47]. Scanarea codurilor de bare se face local pe dispozitiv, cu ajutorul bibliotecii Google Vision API din cadrul Google Play Services.

Am construit lista ce conține alimentele utilizatorului (ecranul principal al aplicației) folosind *Recycler View* [48], o bibliotecă relativ nouă în sistemul Android, utilizată în cadrul aplicațiilor ce conțin liste infinite (exemplu: aplicații pentru mesaje, galerii de imagini, etc.). Avantajul major al acestei tehnologii față de *ListView* este modul eficient în care încarcă elementele din listă în memorie numai la momentul afișării lor pe ecran, pentru a reduce consumul de resurse și pentru a oferi o experiență cât mai plăcută utilizatorului, prin rapiditatea cu care se poate naviga între elemente. Pentru a putea utiliza această tehnologie am construit un *cache* temporar folosind o clasă de tip *singleton* ce conține un vector de obiecte. La intrarea în aplicație vectorul este populat făcând o interogare către server. *RecyclerView*-ul va folosi elementele din acest vector pentru a afișa lista pe ecran.

Atunci când se apasă butonul pentru scanarea unei date de expirare este construit un *Intent*, ce lansează aplicația de cameră foto a sistemului pentru a prelua o imagine, iar apoi aceasta este trimisă către server pentru inferență.

Diagrama de clase completă ce descrie structura finală a aplicației este atașată în anexa 9.3.

Interfața aplicației grafice a fost construită pe baza *wireframe*-urilor atașate în anexa 9.1, ce au fost validate și îmbunătățite pe parcursul dezvoltării proiectului folosind *feedback*-ul obținut de la câțiva colegi ale căror nume sunt menționate în secțiunea „Mulțumiri”. Capturi de ecran ce detaliază modul de funcționare al aplicației finale pot fi găsite în anexa 9.2.

Pentru *frontend*-ul care rulează pe RaspberryPi am scris o aplicație în Python3, ce folosește bibliotecile *Requests* și *PyCamera* pentru a trimite imagini către server. Aceasta este activată la apăsarea unui buton. Toate produsele ce au fost introduse vor putea fi apoi vizualizate în aplicația pentru telefon.

5.5.2 Backend

Partea de *backend* este formată dintr-un *REST API* scris în Python, folosind biblioteca Flask [49], ce rulează în *cloud* pe platforma Heroku [50]. Pe acest server sunt executate operațiunile de login / logout, adăugare produs în listă, etc. și aici se găsește și baza de date PostgreSQL [51] folosită. Cererile pentru inferență (extragerea datelor de expirare din imagini) sunt trimise către o mașină virtuală ce beneficiază de accelerare GPU și rulează pe Google Cloud Platform [52].

Pentru a ușura procesul de dezvoltare și pentru a avea portabilitate cât mai mare a codului am utilizat bibliotecile „Flask-SQLAlchemy” [53] și „Flask-Migrate” [54]. SQLAlchemy permite definirea modelelor din baza de date folosind o structură universală, apoi aceste modele pot fi create automat în baza de date, indiferent de sintaxa folosită. Această funcționalitate a fost foarte utilă deoarece la început am dezvoltat aplicația pe mașina locală, folosind o bază de date SQLite, iar la final am putut porta codul pe Heroku [50], unde am folosit o bază de date PostgreSQL [51], fără a fi necesare modificări. „Flask-Migrate” este un sistem de versionare a modelelor bazei de date, aplicat peste SQLAlchemy, utilizat pentru a putea salva și restaura structura bazei de date la starea în care aceasta se găsea la un anumit moment în timp.

Am creat conturile pentru utilizatori folosind biblioteca „Flask-Login” [55]. Aceasta va genera un *cookie* de autentificare pentru fiecare utilizator atunci când sunt introduse datele de autentificare, care apoi va fi inclus în următoarele cereri pentru a putea accesa *endpoint*-uri care nu sunt publice.

Am implementat un set de metode ce pot fi apelate din consola Flask, prin comanda „heroku run flask shell”. Acestea au rolul de a automatiza operații precum adăugarea/ștergerea utilizatorilor, investigarea stării sistemului (numărul de cereri pentru inferență, dimensiunea ocupată de baza de date, etc).

6 EVALUAREA REZULTATELOR

În final proiectul s-a dovedit a fi un succes, arhitectura propusă obținând o precizie mult mai bună decât sistemele existente (testate în secțiunea 3) pentru detecția datelor de expirare, aşa cum se poate observa în Tabelul 2.

Tabelul 2 Comparație cu soluțiile existente

Sistem	Precizie
Tesseract OCR	10.6%
SegLink (default) + CRNN (default)	16.6%
TextBoxes++ (default) + CRNN (default)	13.6%
Google Cloud Vision API	50.0 %
Modelul propus în lucrarea curentă	72.7 %

În Tabelul 3 am comparat performanțele obținute folosind pentru antrenare fiecare dintre seturile de date prezentate în secțiunea 4.3 (setul de date generat sintetic și setul de date reale) pentru fiecare dintre cele două rețele neurale utilizate (TextBoxes++ [16] și CRNN [20]). Cele mai bune rezultate au fost obținute prin combinarea setului de date generat sintetic folosind Unity3D cu setul de imagini adnotate manual. Se poate observa că reantrenarea modelului CRNN [20] aduce în general un beneficiu mai mare decât reantrenarea modelului TextBoxes++ [16], iar reantrenarea ambelor modele este cea care dă cele mai bune rezultate. Precizia obținută arată clar diferența între datele sintetice și cele reale, deoarece oricât de mult am încercat să fac imaginile sintetice să pară cât mai realiste, modelele antrenate cu date reale au avut performanțe mai bune de fiecare dată. Rezultatul final cu precizia cea mai bună a fost însă modelul antrenat folosind ambele seturi de date, deci construirea setului de date sintetic și-a dovedit la final utilitatea.

Tabelul 3 Comparații metode reantrenare

Reantrenare TextBoxes++	Reantrenare CRNN	Set de date	Precizie
NU	NU	-	13.6 %
DA	NU	Imagini sintetice	19.6 %
NU	DA	Imagini sintetice	28.7 %
DA	DA	Imagini sintetice	40.9 %
DA	NU	Imagini reale	24.2 %
NU	DA	Imagini reale	34.8 %
DA	DA	Imagini reale	63.3 %
DA	DA	Imagini reale + sintetice	72.7 %

De asemenea aplicația pentru reducerea risipei ce încorporează metoda prezentată funcționează perfect, timpul necesar pentru extragerea datei fiind de aproximativ 4 secunde. Timpul necesar în total pentru introducerea unui nou produs de către un utilizator experimentat este de aproximativ 15 secunde.

Am testat aplicația pentru reducerea risipei alimentare cu ajutorul a 5 colegi (menționați în secțiunea „Mulțumiri”), cărora le-am trimis kitul de instalare al aplicației (APK-ul). Totul a mers bine, singura problemă apărută este un *bug* ce apare ocazional la lansarea camerei video pe telefoanele Sony Xperia. Probabil problema apare din cauza aplicației de cameră foto din sistemul de operare, ce a fost modificată de către fabricant față de versiunea oficială Android.

Un alt mic inconvenient a fost faptul că utilizatorii raportau dificultăți la conectarea în aplicație deși eu nu îmi aduceam aminte să fi făcut modificări asupra codului. Problema apărărea deoarece serviciul gratuit de *hosting* folosit pentru API-ul din *backend* (Heroku [50]) pune mașina virtuală în modul *sleep*, pentru a economisi resurse atunci când nu au mai fost detectate conexiuni pe o anumită perioadă de timp. Problema poate fi ușor rezolvată așteptând 15-20 de secunde după prima tentativă de *login* pentru restaurarea mașinii virtuale sau prin achiziționarea unui cont plătit.

7 CONCLUZII

În concluzie, proiectul acesta dovedește că este posibilă construirea unei soluții „end-to-end”, bazată aproape în întregime pe rețele neurale adânci, ce poate detecta în imagini date de expirare cu o precizie mult mai mare decât cea a celor mai bune sisteme pentru recunoaștere optică a caracterelor disponibile pe piață. De asemenea am construit cu succes o aplicație ce utilizează algoritmul propus în scopul reducerii risipei alimentare, o problemă deosebit de gravă având în vedere statisticile din secțiunea 1.1. Dacă această aplicație va ajunge să fie folosită la scară largă pe viitor în mod sigur va avea un impact pozitiv asupra planetei.

Pe parcursul lucrării am învățat cât de dificilă este construirea unei soluții de învățare automată atunci când nu există deja un set de date de dimensiuni suficiente pentru problema studiată. Din fericire pentru detecția datelor de expirare am putut genera un set de date sintetice și am putut aduna un număr considerabil de imagini reale (aproximativ 660), fiind vorba despre produse alimentare, care sunt foarte variate și ușor disponibile.

Pentru elaborarea proiectului am combinat foarte multe cunoștințe studiate pe parcursul celor 4 ani de facultate, fapt ce m-a impresionat puternic și mi-a demonstrat din nou importanța fiecărei discipline.

În primul rând am utilizat noțiunile din cadrul materiei „Învățare Automată”, din anul 4, pentru a construi arhitectura celor două rețele neurale prezentate în secțiunile 4.1 și 4.2. Pentru a construi setul de date din secțiunea 4.3, în special partea generată sintetic, am utilizat multe cunoștințe din domeniul graficii computerizate, studiate la materiile „Elemente de Grafică pe Calculator” și „Sisteme de Prelucrare Grafică”.

În cazul aplicației de reducere a risipei alimentare am validat interfața grafică folosind procedeele învățate în cadrul materiei „Interacțiunea Om-Calculator” apoi am dezvoltat aplicația folosind tehnici de programare Android deprinse pe parcursul materiei „Elemente de Informatică Mobilă”. Pentru funcționarea corectă a aplicației au fost necesare cunoștințe de „Rețele Locale”, „Baze de Date”, „Sisteme de Operare” și desigur „Programare Orientată Obiect”.

Am publicat rezultatele finale, seturile de date folosite și tot codul dezvoltat pe parcursul realizării proiectului pe GitHub [56]. Tot aici poate fi găsit și un clip video captat de pe ecranul telefonului mobil ce ilustrează funcționarea aplicației.

7.1 Dezvoltări ulterioare

Pentru pasul de detecție a datelor arhitectura „Fused Text Segmentation Networks” [57] pare a fi o îmbunătățire majoră pe viitor, datorită faptului că este mult mai robustă la rotație. Momentan însă nu există o implementare disponibilă public prin care să poată fi reproduse rezultatele menționate de către autori.

Desigur în viitorul apropiat pot fi adăugate noi funcționalități în aplicația de reducere a risipei alimentare, precum sugerarea unor rețete pe baza produselor aflate aproape de data expirării

sau opțiunea de a contacta asociații caritabile pentru a le putea dona mai ușor către oamenii nevoiași.

8 BIBLIOGRAFIE

- [1] A. Zaafouri, M. Sayadi și F. Fnaiech, „A Vision Approach for Expiry Date Recognition using Stretched Gabor Features,” *The International Arab Journal of Information Technology*, vol. 12, pp. 448-455, 2015.
- [2] E. Peng, P. Peursum și L. Li, „Product Barcode and Expiry Date Detection for the Visually Impaired Using a Smartphone,” în *2012 International Conference on Digital Image Computing Techniques and Applications (DICTA)*, 2012.
- [3] „Batch and Expiry Tracking,” tradegecko.com, [Interactiv]. Available: <https://www.tradegecko.com/product-tour/batch-expiry-tracking>. [Accesat 19 06 2019].
- [4] A. Walts, „Sort Inventory Based on Expiration Dates with Lots, FIFO, and FEFO Features,” skuvault.com, 31 08 2018. [Interactiv]. Available: <https://www.skuvault.com/blog/skuvaultlotsfifofofeatures/>. [Accesat 19 06 2019].
- [5] „Stop expired products,” datecheckpro.com, [Interactiv]. Available: <https://datecheckpro.com/>. [Accesat 19 06 2019].
- [6] M. Lumibao, „Expiry Date Tracker,” [Interactiv]. Available: <https://play.google.com/store/apps/details?id=com.lumibao.expirydatetracker>. [Accesat 19 06 2019].
- [7] Food And Agriculture Organization of the United Nations, „Global food losses and food waste – Extent, causes and prevention,” în *Interpack*, Rome, 2011.
- [8] Food And Agriculture Organization of the United Nations, „Regional Overview of Food Security and Nutrition in Europe and Central Asia,” Budapest, 2018.
- [9] Å. Stenmarck, C. Jensen, T. Quested și G. Moates, „Estimates of European food waste levels,” Stockholm, 2016.
- [10] Asociația Mai Mult Verde, „România fără risipă,” 2017. [Interactiv]. Available: <http://foodwaste.ro/wp-content/uploads/2018/10/FoodWasteRO-Anexa21-CatalogONG.pdf>. [Accesat 19 June 2019].

- [11] Food And Agriculture Organization of the United Nations, „Food wastage footprint, Impacts on natural resources,” 2013. [Interactiv]. Available: <http://www.fao.org/3/i3347e/i3347e.pdf>. [Accesat 19 06 2019].
- [12] Food And Agriculture Organization of the United Nations, „Food wastage footprint & Climate Change,” [Interactiv]. Available: <http://www.fao.org/3/a-bb144e.pdf>. [Accesat 19 June 2019].
- [13] „GS1 DataBar barcodes,” gs1.org, [Interactiv]. Available: <https://www.gs1.org/standards/barcodes/databar>. [Accesat 19 06 2019].
- [14] „Fill Your Fridge New York,” peapod.com, [Interactiv]. Available: <http://www.fromthepod.com/fill-your-fridge-nyc>. [Accesat 19 06 2019].
- [15] S. Long, X. He și C. Ya, „Scene Text Detection and Recognition: The Deep Learning Era,” *arXiv preprint arXiv:1811.04256*, 2018.
- [16] L. Minghui, S. Baoguang și B. Xiang, „Textboxes++: A single-shot oriented scene text detector,” *IEEE transactions on image processing*, vol. 27, nr. 8, pp. 3676-3690, 2018.
- [17] „Tesseract OCR,” [Interactiv]. Available: <https://github.com/tesseract-ocr/tesseract>. [Accesat 20 06 2019].
- [18] „Python Tesseract,” [Interactiv]. Available: <https://github.com/madmaze/pytesseract>. [Accesat 20 06 2019].
- [19] B. Shi, X. Bai și S. Belongie, „Detecting oriented text in natural images by linking segments,” în *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [20] B. Shi, X. Bai și C. Yao, „An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, nr. 11, pp. 2298-2304, 2016.
- [21] M. Völk, „SSD-based object and text detection with Keras,” [Interactiv]. Available: https://github.com/mvoelk/ssd_detectors. [Accesat 20 06 2019].
- [22] „Vision AI,” google.com, [Interactiv]. Available: <https://cloud.google.com/vision/>. [Accesat 19 06 2019].
- [23] „Advanced OCR Tool,” datalogic.com, [Interactiv]. Available: <https://www.datalogic.com/eng/manufacturing/vision-systems/advanced-ocr-tool-pd-828.html>. [Accesat 21 06 2019].

- [24] A. Shahab, F. Shafait și A. Dengel, „ICDAR 2011 Robust Reading Competition Challenge 2: Reading Text in Scene Images,” în *International Conference on Document Analysis and Recognition*, 2011.
- [25] L. Minghui, „TextBoxes++: A Single-Shot Oriented Scene Text Detector,” [Interactiv]. Available: https://github.com/MhLiao/TextBoxes_plusplus. [Accesat 19 06 2019].
- [26] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu și A. C. Berg, „SSD: Single shot multibox detector,” în *European conference on computer vision*, 2016.
- [27] J. Redmon, S. Divvala, R. Girshick și A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection,” în *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [28] J. Redmon și A. Farhadi, „Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.
- [29] K. Simonyan și A. Zisserman, „Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li și L. Fei-Fei, „Imagenet: A large-scale hierarchical image database,” în *IEEE conference on computer vision and pattern recognition*, 2009.
- [31] M. Liao, B. Shi, X. Bai, X. Wang și W. Liu, „TextBoxes: A fast text detector with a single deep neural network,” în *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [32] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He și J. Liang, „EAST: an efficient and accurate scene text detector,” în *IEEE conference on Computer Vision and Pattern Recognition*, 2017.
- [33] S. Hochreiter și J. Schmidhuber, „Long short-term memory,” *Neural computation*, vol. 9, nr. 8, pp. 1735-1780, 1997.
- [34] „Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research,” Microsoft, [Interactiv]. Available: <https://github.com/microsoft/AirSim>. [Accesat 19 06 2019].
- [35] „ICDAR2017 Competition on Multi-lingual scene text detection and script identification,” uab.es, 2017. [Interactiv]. Available: <https://rrc.cvc.uab.es/?ch=8&com=introduction>. [Accesat 19 06 2019].
- [36] A. Gupta, A. Vedaldi și A. Zisserman, „Synthetic data for text localisation in natural images,” în *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

- [37] „Useful extensions to the standard Python datetime features,” [Interactiv]. Available: <https://github.com/dateutil/dateutil/>. [Accesat 21 06 2019].
- [38] „Open Food Facts - The free food products database,” openfoodfacts.org, [Interactiv]. Available: <https://world.openfoodfacts.org/discover>. [Accesat 20 06 2019].
- [39] „Data Input Pipeline Performance,” tensorflow.org, [Interactiv]. Available: <https://www.tensorflow.org/guide/performance/datasets>. [Accesat 20 06 2019].
- [40] N. Sakharnykh, „Beyond GPU Memory Limits with Unified Memory on Pascal,” nvidia.com, 14 12 2016. [Interactiv]. Available: <https://devblogs.nvidia.com/beyond-gpu-memory-limits-unified-memory-pascal/>. [Accesat 20 06 2019].
- [41] C. Ying, S. Kumar, D. Chen, T. Wang și Y. Cheng, „Image Classification at Supercomputer Scale,” *arXiv preprint arXiv:1811.06992*, 2018.
- [42] „Building industrial grade training data software for machine learning teams,” labelbox.com, [Interactiv]. Available: <https://labelbox.com/about>. [Accesat 20 06 2019].
- [43] S. K. Gunnarson, „Dot Matrix Font,” dafont.com, [Interactiv]. Available: <https://www.dafont.com/dot-matrix.font>. [Accesat 20 06 2019].
- [44] J. Zamarripa, „Club Dia Font,” fontsplace.com, [Interactiv]. Available: <https://www.fontspace.com/dibujado/club-dia>. [Accesat 20 06 2019].
- [45] K. King, „Kingthings Trypewriter Font,” [Interactiv]. Available: <https://www.dafont.com/kingthings-trypewriter.font>. [Accesat 20 06 2019].
- [46] A. Jung, „Image augmentation for machine learning experiments,” [Interactiv]. Available: <https://github.com/aleju/imgaug>. [Accesat 20 06 2019].
- [47] „Volley Overview,” [Interactiv]. Available: <https://developer.android.com/training/volley>. [Accesat 21 06 2019].
- [48] „Create a List with RecyclerView,” [Interactiv]. Available: <https://developer.android.com/guide/topics/ui/layout/recyclerview>. [Accesat 24 06 2019].
- [49] „Flask (A Python Microframework),” [Interactiv]. Available: <http://flask.pocoo.org/>. [Accesat 24 06 2019].
- [50] „Cloud Application Platform,” [Interactiv]. Available: <https://www.heroku.com/>. [Accesat 24 06 2019].

- [51] „PostgreSQL: The World's Most Advanced Open Source Relational Database,” [Interactiv]. Available: <https://www.postgresql.org/>. [Accesat 24 06 2019].
- [52] „COMPUTE ENGINE Scalable, High-Performance Virtual Machines,” google.com, [Interactiv]. Available: <https://cloud.google.com/compute/>. [Accesat 24 06 2019].
- [53] „Flask-SQLAlchemy,” [Interactiv]. Available: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/>. [Accesat 24 06 2019].
- [54] „Flask-Migrate,” [Interactiv]. Available: <https://github.com/miguelgrinberg/Flask-Migrate>. [Accesat 24 06 2019].
- [55] „Flask-Login,” [Interactiv]. Available: <https://github.com/maxcountryman/flask-login>. [Accesat 24 06 2019].
- [56] V. Florea, „Expiry Date Recognition Using Deep Neural Networks,” 24 06 2019. [Interactiv]. Available: https://github.com/vladalexgit/expiry_date_recognition. [Accesat 24 06 2019].
- [57] Y. Dai, Z. Huang, Y. Gao, Y. Xu, K. Chen, J. Guo și W. Qiu, „Fused text segmentation networks for multi-oriented scene text detection,” în *International Conference on Pattern Recognition (ICPR)*, 2018.

9 ANEXE

9.1 Wireframe-uri folosite pentru validarea interfeței grafice

The wireframe illustrates a mobile application interface for managing food items in a refrigerator. The top section shows a notification bar with the date (10 December), time (16:40), and a message: "An item in your fridge is about to expire! WasteWarrior has noticed Cheddar Cheese expires in 2 days." Below this is a header with the text "Add your new item". To the right are three rounded rectangular buttons: "Scan Barcode", "Scan Expiry Date", and "SAVE". The main content area displays a list of items with checkboxes:

<input type="checkbox"/>	Cheddar Cheese
<input type="checkbox"/>	Ice Cream
<input type="checkbox"/>	Smoked Fish
<input type="checkbox"/>	Sour Cream
<input type="checkbox"/>	Chocolate Sauce

Below the list are four input fields labeled "Name:", "Barcode", "ExpDate", and "ADD NEW ITEM". At the bottom left are "USERNAME" and "PASSWORD" fields, and a "LOGIN" button.

9.2 Capturi de ecran din aplicația finală

The image displays three screenshots of the WasteWarrior mobile application interface:

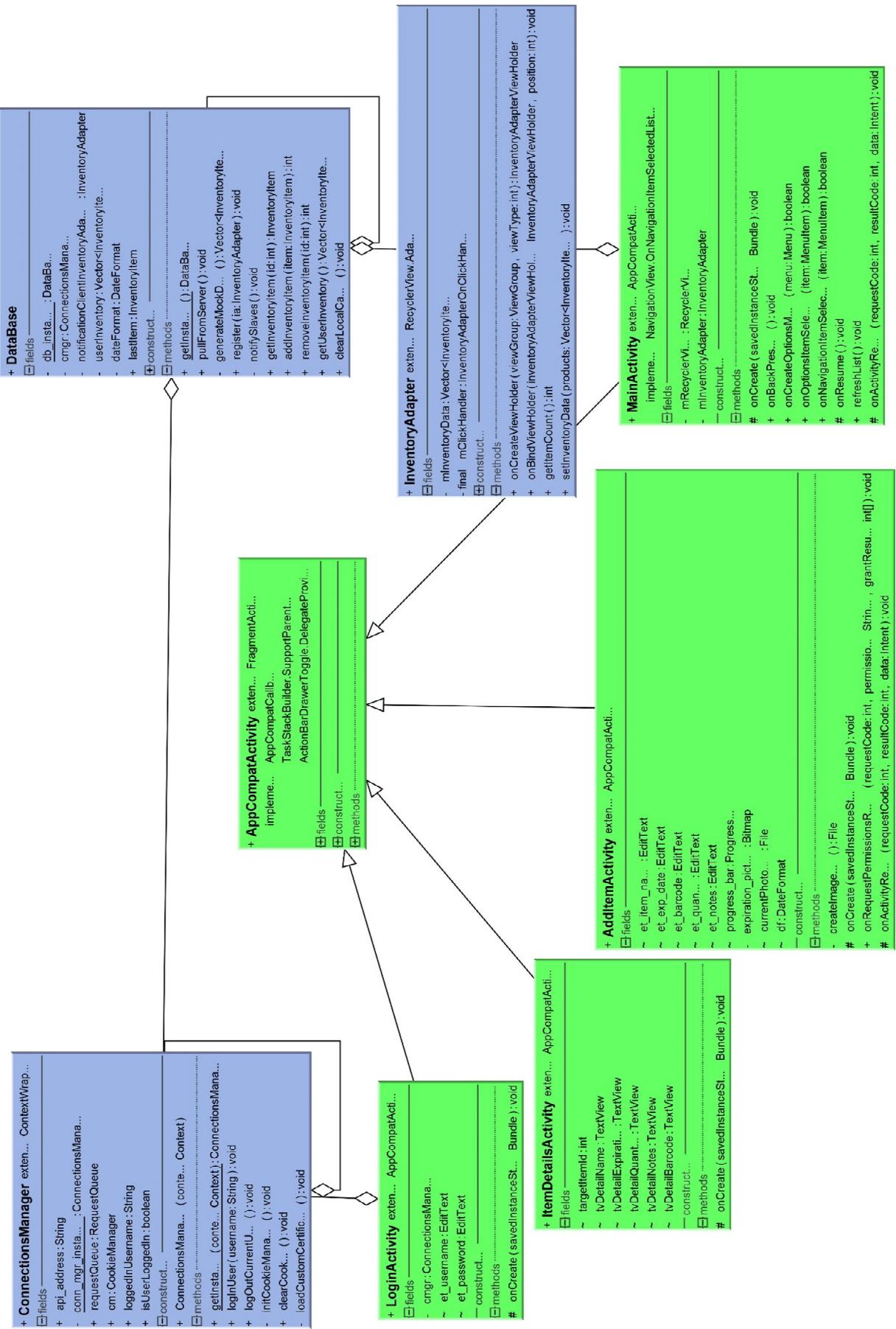
- Screenshot 1: Add a new item screen**

This screen shows a blue header with the text "Add a new item". Below the header are five input fields: "Barcode" (with a red underline), "Name", "Expiration Date", "Quantity", and "Notes". Each field has a "SCAN" button to its right. A red circle with a white plus sign is located in the top right corner of the header area.
- Screenshot 2: Home screen**

This screen shows the WasteWarrior logo at the top left. It lists two items: "Penne rigate" (1 years left) and "Sauce tomate aux olives" (1 years left). Each item has a circular profile picture of a cartoon character and a green "edit" icon. A red circle with a white plus sign is located in the top right corner of the header area.
- Screenshot 3: Login screen**

This screen shows a blue header with the WasteWarrior logo. Below the header are two input fields: "Username" (with a red underline) and "Password" (with a red underline). A "LOG IN" button is located below the password field. A red circle with a white plus sign is located in the top right corner of the header area.

9.3 Diagrama de clase pentru aplicația Android



9.4 Comparație între arhitecturile SSD [26] și YOLO [27]

(imagine preluată din [26])

