UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

realizată de

Tudose Adrian

Sesiunea: iulie 2020

Coordonator ştiinţific

Lect. Dr. Benchea Mihai Răzvan

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

Recunoașterea caracterelor în timp real

propusă de

Tudose Adrian

Sesiunea: iulie 2020

Coordonator ştiinţific

Lect. Dr. Benchea Mihai Răzvan

Cuprins

Introducere

Aplicația LiveCameraTranslator este o aplicație Android care are ca scop identificarea și traducerea în timp real a textelor în scene naturale utilizând camera frontală a dispozitivului. Selectând limba în care este scris textul ce se dorește tradus și limba în care se dorește traducerea, utilizatorii vor putea prin alinierea camerei dispozitivului cu diferite texte să obțină traducerile acestora pe măsură ce acestea apar în câmpul vizual al camerei dispozitivului.

Aplicația va fi accesibilă pentru dispozitivele mobile ce rulează sistemul de operare Android, versiunea 5.0(API level 21) sau versiunile mai recente, fiind compatibilă astfel cu marea majoritate a dispozitivelor actuale ce rulează Android.

Motivație

Am ales tema “ Recunoaștere caracterelor în timp real” deoarece consider că făcând parte din domeniul Computer Vision este o tema de actualitate în aria programării , folosind la baza pentru clasificarea caracterelor rețelele neuronale, a căror vaste aplicații în domeniu se pot observa în proiectele apărute recent, care variază de la procesarea limbajului natural până la clasificarea de imagini.

Aplicația vine în ajutor persoanelor care necesită traducerea unui text dintr-o scena naturală (plăcuțe de avertizare, etichete, etc.) într-o limbă familiară utilizatorului, pentru a ajuta utilizatorul să descifreze mesajul și să înțeleagă contextul textului în mediul de proveniență . Pentru ca o astfel de aplicație să fie cât mai facilă și să ofere un răspuns cât mai rapid în orice circumstanțe (chiar și în lipsa semnalului), ea ar trebui să fie accesibilă oriunde, cu ușurință, să își execute toate calculele local și să aibe o interfață ușor de folosit pe un număr mare de texte.

Accesibilitatea aplicației o vom asigura punând cerințele funcționale în contextul unei aplicații Android care va fi accesibilă posesorilor de dispozitive mobile, lucru care permite utilizarea acesteia de oriunde, permițând astfel o utilitate crescută în contextul vieții de zi cu zi, cum ar fi magazine, resturante, etc.

Execuția locală a codului poate facilita atât viteză de rulare cât și o utilizare mai vastă în situațiile în care accesul la internet este restricționat. Un bun exemplu este cazul în care utilizatorul calatorește într-o locație unde servicul de telecomunicații nu oferă acces la internet sau acesta este puternic monetizat.

Interfața aplicației și timpul de răspuns dictează experiența pe care utilizatorul o va avea, astfel am optat pentru o interfață simplă cu un singur buton ce permite selectarea limbilor din care se face traducerea , iar restul display-ului permite vizualizarea directă a traducerii textului, într-un context dinamic, astfel mai multe texte pot fi traduse doar prin mutarea camerei dispozitivului. Acest lucru ar putea fi util spre exemplu în traducerea rapidă a mai multor etichete a unor produse într -un magazin în care descrerile acestora nu sunt oferite în nici o limba cunoscută.

Context

OCR (Optical Character Recognition) sau recunoasterea optica a caracterelor reprezinta conversia in format electronic a imaginii unui text tipărit sau scris de mână. Este utilizat pe scară largă ca modalitate de extragere a datelor din documentele tiparite (facturi, extrase bancare, chitanțe, pașapoarte etc.), fotografiate sau scanate.

Pentru a extrage şi modifica datele din documente scanate, imagini provenite de la camera digitală sau fişiere PDF de tip „numai imagine”, este nevoie de un program OCR care să identifice literele din imagine, să le combine în cuvinte, iar apoi cuvintele în propoziţii. Astfel se poate accesa şi edita conţinutul documentului original.

Probabil cel mai cunoscut caz de utilizare pentru OCR este transformarea documentelor tipărite pe hârtie în documente text care pot fi citite de mașini. Odată ce un document scanat a trecut prin procesarea OCR, textul documentului poate fi editat cu procesoare de texte precum Microsoft Word sau Google Docs. Înainte de a fi disponibilă tehnologia OCR, singura opțiune de digitalizare a documentelor tipărite pe hârtie a fost re-tastarea manuală a textului. Nu numai că consuma foarte mult timp, dar a venit și cu inexactități și erori de dactilografiere.

OCR este adesea folosit că tehnologie „ascunsă”, alimentând multe sisteme și servicii cunoscute în viața noastră de zi cu zi. Mai puțin cunoscute, dar la fel de importante, cazurile de utilizare pentru tehnologia OCR includ automatizarea introducerii datelor, indexarea documentelor pentru motoarele de căutare, recunoașterea automată a plăcuțelor numerice, precum și asistarea persoanelor cu deficiențe de vedere sau nevăzătoare.

Tehnologia OCR s-a dovedit extrem de utilă în digitalizarea ziarelor și textelor istorice care acum au fost transformate în formate care pot fi căutate și au făcut accesarea textelor anterioare mai ușoară și mai rapidă.

Cea mai cunoscută aplicație, la momentul de fată, care realizează identificarea și traducerea de text în imagini este oferita chiar de compania Google, prin aplicația Google Translate, care printr-un update recent permite atât tranducera textului prin indroducere din tastatură cât și accesul la cameră, traducând textul în timp real, când acesta este identificat.

Cerințe Functionale

1. Identificarea textului într-o imagine – considerând un cadru al camerei video aplicația va trebui sa identifice toate zonele ce conțin text, împreună cu textul care este conținut în acea zonă
2. Afișarea pe ecran a textelor identificate – aplicația va trebui să poată afișa pentru un cadru video, textele identificate, suprapunând textul deja existent în imagine
3. Urmărirea textelor identificate – deoarece camera va rula continu, aplicația va urmări și modifica locațiile secțiunilor de text identificate, pentru că acestea să suprapună în continuare textele aferente
4. Posibilitatea de selecție a limbilor între care se încearcă traducerea – utilizatorul va putea selecta în orice moment limba de proveniență a textelor din imagine și limba în care se încearcă traducerea acestora
5. Traducerea textului – odată selectate două limbi, aplicația va trebui să înlocuiască textele identificate și afișate cu traducerile lor

Tehnologii folosite

Aplicația va fi realizată în două etape, implementarea unui cod python pentru testarea metodei și implementarea aplicației propriuzise Android.

Am ales implementarea inițială a unui cod python pentru testarea arhitecturii și identificarea parametrilor funcțiilor folosite în modulul identificării de text într-o imagine statică. Avantajele au constat în implementarea mai rapida a codului ce a permis testarea si verificarea parametrilor și diferitelor etape ale procesului de detecție.

Python este un limbaj de programare interpretat de nivel înalt, orientat pe obiecte, cu semantică dinamică. Nivelul său ridicat, construit în structuri de date, îl fac foarte atractiv pentru dezvoltarea rapidă a aplicațiilor precum și pentru utilizarea ca script sau limbaj intermediar pentru a conecta componentele existente.

Aplicația Android va fi apoi dezvoltată în limbajul de programare Java, iar mediul de dezvoltarea va fi Android Studio. SDK-ul Android joaca un rol crucial in dezvoltarea aplicatiei permitand prin librariile sale specifice accesul la camera dispozitivului.

Java este un limbaj de programare și o platformă de calcul lansată prima dată de Sun Microsystems în 1995. Există o mulțime de aplicații și site-uri web care nu vor funcționa decât cu Java instalat și mai multe sunt create în fiecare zi. Java este rapid, sigur și de încredere. De la laptopuri până la centrele de date, console de jocuri la supercomputere științifice, telefoane mobile la Internet, Java este peste tot!

Aplicațiile Android sunt dezvoltate folosind limbajul Java. În momentul de față aceasta fiind si singura opțiune. Android se bazează foarte mult elementele fundamentale Java. SDK Android include multe biblioteci Java standard (biblioteci cu structură de date, biblioteci matematice, biblioteci grafice, biblioteci de rețea ș.a.m.d.), precum și biblioteci speciale Android care ajută la dezvoltarea aplicațiilor Android.

În implementarea algoritmului de identificare a textului în imagini am utilizat librăriile:

- OpenCV – pentru preprocesarea imaginilor și propunerea zonelor de interes
- Keras – pentru realizarea și antrenarea unui clasificator de carctere

- TensorflowLite – pentru rularea clasificatorului în contextul aplicației Android

OpenCV(Open Source Computer Vision Library) este o bibliotecă software de viziune computerizată și software de învățare automată. OpenCV a fost construit pentru a furniza o infrastructură comună pentru aplicațiile de viziune computerizată și pentru a accelera utilizarea percepției mașinilor în produsele comerciale. Biblioteca are peste 2500 de algoritmi optimizați, care include un set cuprinzător de algoritmi de vizionare computerizată de ultimă generație și de învățare automată.

Keras este un API de învățare profundă scrisă în Python, care rulează pe baza platformei TensorFlow de învățare automată. Acesta a fost dezvoltată cu accent pe permiterea experimentării rapide. "A fi capabil să treci de la idee la rezultate cât mai rapid este esențial pentru a face o cercetare bună."

TensorFlow este o bibliotecă open source pentru calculare numerică și machine learning pe scară largă. TensorFlow îmbină o mulțime de modele și algoritmi de învățare automată și învățare profundă (numită rețea neuronală) și le face utile printr-o metaforă comună. Utilizează Python pentru a furniza un API pentru front-end pentru construirea de aplicații cu framework-ul, executând în același timp aplicațiile în C++ de înaltă performanță.

Pentru traducerea locală a textelor am ales utilizarea bibliotecii Firebase ML Kit, care ne permite descărcarea și utilizarea de modele capabile să traducă text.

Firebase ML Kit este o bibliotecă care permite realizarea fără efort și, cu un cod minim, utilizarea unei varietăți de modele profunde extrem de precise, pre-antrenate în aplicațiile Android. Majoritatea modelelor pe care le oferă sunt disponibile atât la nivel local, cât și pe Google Cloud. În prezent modelele sunt limitate numai la sarcini legate de Compure Vision, precum recunoașterea optică a caracterelor, scanarea codurilor de bare și detectarea obiectelor.

Alte abordări cunoscute

Scrisul a fost una dintre cele mai importante invenții ale umanității, a permis atât comunicarea cât și stocarea ideilor,conceptelor și informațiilor la un anumit punct în timp. Pe parcursul evoluției umanitaii textele au avut un rol tot mai important în societăți , ajungând în ziua de azi să ne înconjoare în viața de zi cu zi. Putem spune astfel că cea mai mare parte a informației cunoscute nouă se află în format redactat, fie el scris de mâna sau tipărit . De aici importanța extragerii acestor texte pentru a fi utilizate sau stocate într-o manieră mai bună .

În momentul de față, dată fiind evoluția componentelor și software-urilor multă din informația care circulă prin internet este sub formă de imagini. Astfel extragerea informației textuale din imagini devine tot mai importantă și în consecință tot mai cercetată în ultimii ani, comunitatea din acest domeniu făcând eforturi majore, în ciuda unor provocări (zgomot, neclaritate, denaturare, ocluzie și variație).

Cu toate acestea localizarea și citirea textelor în scene naturale sunt sarcini extrem de dificile iar provocările majore din detectia și recunoașterea textului poat fi clasificate în trei tipuri:

- Diversitatea textelor in scene: Spre deosebire de caracterele din documente, care sunt de obicei cu font obișnuit, o singură culoare, dimensiunea constantă și aranjarea uniformă, textele din scene naturale pot avea fonturi complet diferite, culori, dimensiuni și orientări, chiar și în aceeași scenă.
- Complexitatea fundalului: fundalurile imaginilor și videoclipurilor scenelor naturale pot fi foarte complexe. Elemente precum semnele, gardurile, cărămizile și ierburile sunt practic nedistinguibile din textul adevărat și astfel provoacă confuzii și erori.
- Factorii de interferență: diverși factori de interferență, de exemplu zgomot, estompare, denaturare, rezoluție scăzută, iluminare neuniformă și ocluzie parțială creșterea eșecurilor în detectarea și recunoașterea textului scenelor.

Metodele utilizate pentru extragerea informațiilor textulale din imagini pot fi clasificate in 3 categorii detecție de text, recunoastere de text, recunoastere de text “end-to-end”.

Detecția de text se împarte și ea la randul ei in 3 mari categorii de metode utilizate: detectare bazată pe texturi, detectare bazată pe componente si hibridă.

Metodele bazate pe analiză texturii tratează textele din imagini ca pe o textură specială și se folosesc de caracteristicile lor, cum ar fi intensitatea locală, răspunsul la filtre, pentru a deosebi zonele de text de zonele ce nu conțin text. Din păcate aceste metode necesită multe calcule, necesitând scanarea tuturor locațiilor dintr-o imagine pentru toate mărimile de texte, astfel fiind și sensibile la rotația și diferența de dimensiuni a textelor. Un alt dejavantaj este că aceste metode tratează de obicei doar textele orizontale.

Metodele bazate pe componente extrag în primă fază componente candidat printr-o varietate de metode cum ar fi grupare după culoare sau extragere de regiuni extreme, apoi fitreaza componentele nontextuale utilizând metode bazate pe reguli manuale sau clasificatoare automate. În general, aceste metode sunt mult mai eficiente, deoarece numărul de componente care trebuie prelucrate este relativ mic. În plus, față de metodă precedentă, aceste nu sunt sensibile la rotație, scalare sau schimbare de font. În ultimii ani aceste metode au devin cele mai utilizate în domeniul detectării textului în scene naturale.

Metodele hibride sunt o combinație între cele două metode menționate precedent, bazate atât pe textura, cât și pe componente, folosindu-se de avantajele celor două. Una dintre metodele de acest gen propuse a fost publicată de Liu et . al ., în care pixelii ce alcătuiesc contururi pentru toate zonele de text posibile au fost extrași utilizând un algoritm elaborat de detecție a contururilor, iar proprietățile geometrice ale contururilor sunt verificate pentru a genera regiuni de text candidate, urmată de o procedură de analiză a texturii pentru a distinge regiunile text adevărate de regiunile nontextuale.

În ceea ce priveste recunoasterea de text in imagini, Campos et. al. au testat, comparat și analizat descriptorii de caracteristici actuali și algoritmii de clasificare în computer vision și recunoasterea de patternuri. În plus, au lansat un set de date cu imagini, numit Chars74K, pentru evaluarea algoritmului de recunoaștere a caracterelor. Chars74K a fost larg acceptat și utilizat în domeniul recunoașterii caracterelor în imagini naturale. Cu toate acestea, spre deosebire de metodele obisnuite de recunoaștere a caracterelor, care tratează cuvântul ca unitate de bază, metoda lui de Campos și colab. ia în considerare doar problema recunoașterii caracterelor individuale.

Cum în ultimii ani, algoritmii de recunoaștere a textului bazați pe părți au devenit foarte populari Shi et. al. a propus un model structurat ca arbore bazat pe părți pentru a recunoaște caracterele din imaginile decupate. Acest algoritm este robust față de zgomotul din imagine, blur, ocluzia parțială și variația fontului. Cu toate acestea, acest algoritm depinde de informațiile detaliate despre adnotare, inclusiv modele de caractere și adnotări de piese.

O altă lucrare publicată de Yao et. al. a propus o nouă reprezentare, numită Strokelets, care constă într-un set de elemente de nivel mediu pe mai multe scări. Strokelet-ele pot fi învățate automat prin etichetarea la nivel de caracter și sunt capabile să surprindă structural proprietățile caracterelor la diferite granularități. În plus, oferă o modalitate alternativă de identificare a caracterelor individuale și compune o caracteristică histogramă pentru a descrie eficient caractere. Totodată acest algoritm s-a dovedit a fi atât eficient cât și robust.

Metodele menționate anterior vizează doar un aspect al problemei de extragere a informațiilor textuale, fie detectarea textului, fie recunoașterea textului. Există o varietate de metode care încearcă să construiască un cadru unificat atât pentru detectarea textului, cât și pentru recunoaștere.

Recent, Neumann et. al. a prezentat un nou sistem pentru localizarea și recunoașterea textului în scene naturale, care combină avantajele bazate pe metodă “sliding-window” și bazate pe metodă bazată pe compnente. În acest sistem, părțile din caractere sunt modelate de filtre cu bare orientate. Aceste filtre de bare orientate sunt utilizat pentru a realiza atât detectarea caracterelor, cât și recunoașterea.

Yao et. al.. au realizat un algoritm de recunoaștere “end-to-end” care realizează simultan detectarea și recunoașterea textului în scene. Această este prima lucrare care poate localiza și citi texte de orientări arbitrare în imagini naturale.

În ultimii ani, detectarea textului și recunoașterea scenelor naturale au devenit subiecte de cercetare active. În consecință, a o mulțime de teorii, modele, algoritmi și sisteme relevante au fost propuși dezvoltării . Între timp , cercetătorii în domenii alăturate precum dezvoltarea de roboți și multimedia, adoptă aceste tehnologii în aplicații de navigare pentru roboți, căutare în imagini și recunoaștere de obiecte.

Contribuții

Prezenta lucrare va fi împărțită în două capitole în care vor fi prezentate pe rând cele două componenete care înglobează funcționarea aplicației, respectiv identificarea textului în imagini și reprezentarea rezultatelor în contextul dinamic al aplicației.

- Capitolul 1: Identificarea textului în imagini
- Capitolul 2: Afișarea textului tradus

Identificarea textului în imagini a fost realizată în două etape și anume testarea metodei (în Python) și implementarea în contextul aplicației (în Java). Cum codul Python deservește doar testarea și vizualizarea etapelor iar logica algoritmilor este aceeași, în primul capitol voi utiliza secvențele de cod specifice aplicației Android în combinație cu imagini rezultate din rularea codului Python care descriu diferiți pași ai codului.

Dezvoltarea identificatorului de text a avut că principal scop balansarea vitezei de rulare care este indispensabilă unei aplicații ce rulează în timp real, cu acuratețea necesară unei asemenea aplicații. Astfel am ajuns la implementarea unui pipe-line ce utilizează:

- Funcții optime pentru procesarea imaginilor și extragerea zonelor de interes
- O rețea neuronală convolutionala cu 10 straturi pentru clasificarea unei zone din imagine
- Un algoritm de clusterizare a literelor identificate bazat pe reguli de distanță

Pentru tranducerea textelor identificate am ales utilizarea modulului Firebase ML Kit, recent introdus de Google, deoarece permite descărcarea modelelor de traducere pe dispozitiv. Avantajul constă într-o viteză mai mare și independența de o rețea odată ce un model a fost descărcat.

Afișarea textelor traduse utilizează un algoritm clasic de urmărire a mișcării într-un context video și deservește îmbunătățirea experienței utilizatorului.

Soluția propusă

Aplicatia are ca scop identificarea textelor intr-un context video, astfel o prima componenta o reprezinta afisarea unei previzualizari a camerei dispozitivului. Acest lucru este realizat prin utilizarea unei bibliteteci oferite de Android care permite accesul la camerele dispozitivului. Previzualizarea va fi afisata in mod constant pe display in timpul rularii, astfel ca toate operatiunile posibile si rezultatele functionale ale aplicatiei vor fi reprezentate peste aceasta.

Componenta de identificare a textului ruleaza in paralel cu restul functionalitatilor pentru a nu intrerupe vizualizarea rezultatelor obtinute. Componenta consta intr-un pipepline care primeste constant noile cadre incarcate in camera, iar in momentul in care este libera incepe procesarea unei imagini. Astfel in momentul cand un cadru nou este admis in procestul de procesare se ruleaza pe rand pasii care alcatuiesc detectia de text, iar nici un nou cadru nu este procesat in acel timp. Cand procesarea unui cadru se termina, datele obtinute sunt trimise pentru a fi afisate, iar urmatorul cadru primit va fi analizat in continuare. Pipeline-ul de detectie al textului consta in:

- Preprocesarea unei imagini, care pregateste imaginea pentru a fi folosita in continuare, aplicand:
    - Un filtru de gri, necesar pentru binarizare
    - Blur gaussian pentru a reduce complexitatea imaginii
    - Binarizare pentru a converti imaginea intr-un format binar, care poate fi utilizat pentru identificarea zonelor de interes
- Identificarea zonelor de interes utilizeaza imaginea preprocesata pentru a gasi sectiuni din imagine care ar putea contine caractere, aplicand:
    - Un algoritm care gaseste contururile imaginii binare
    - Un algoritm care selecteaza o parte dintre aceste contururi, care sunt cele mai probabile sa contina caractere si le transforma in casete care le incadreaza
- Clasificarea parcurge elementele identificate anterior, clasificandu-le pe rand in clasele potrivite de caractere, incercand si eliminarea casetelor care nu contin caractere. Realizeaza acest lucru taind din imaginea originala sectiunile incadrate in casete si trecandu-le printr-o retea convolutionala care returneaza clasa de apartenenta a elementului cuprins in imagine.

- Gruparea in texte urmareste sa grupeze caracterele care sunt cel mai probabil sa apartina aceleuias text, bazandu-se pe distanta dintre literele identificate si unghiul format de cetrele casetelor cu axa orizontala cu axa orizontala sau cu drepta care unea centrele precedente. Gruparea obtine un set de texte cu casetele care le cuprind.

Odata ce textele dintr-o imagine au fost identificate acestea sunt trecute printr-o clasa insarcinata cu traducerea acestora in functie de setarile selectate in acel moment. Daca utilizatorul nu a ales limbile intre care se realizeaza traducerea, textul este returnat asa cum este obtinut. Daca insa utilizatorul cere traducera textelor obtinute, se verifica daca modelul responsabil cu traducerea intre limbile selecatate este disponibil local, altfel acesta se descarca automat daca dispozitivul detine acces la o retea si apoi se va incepe automat traducerea. Utilizatorul are posibilatea de a selecta aceste limbi prin intermediul butonului “Translate” accesibil din interfata.

Cum in timpul in care componenta de identificare a textelor ruleaza afisarea nu este intrerupta, a fost nevoie de un modul ce coordoneaza locatia in care sunt afisate casetele de la un cadru la altul. Astfel am realizat un set de functii care urmaresc deplasarea textelor in imagini urmarind doua seturi de pixeli astfel:

- Un set rezultat din coordonatele casetelor zonelor de interes care este monitorizat in spate fara a fi afisat
- Un set rezultat din coordonatele casetelor finale care contin text

La prima afisare a unui set de texte se utilizeaza primul set de coordonate urmarite pentru a determina noua pozitie a textelor, in cazul in care acestea au fost deplasate din pozitiile de la inceputul procesarii imaginii. Acest lucru fiind realizat tianad cont de apartenenta casetelor cu caractere in cele cu text.

Dupa ce un set de texte a fost afisat pe ecran acesta foloseste cel de-al doilea set de puncte urmarite pentru a se deplasa in concordanta cu deplasarea curenta a textelor.

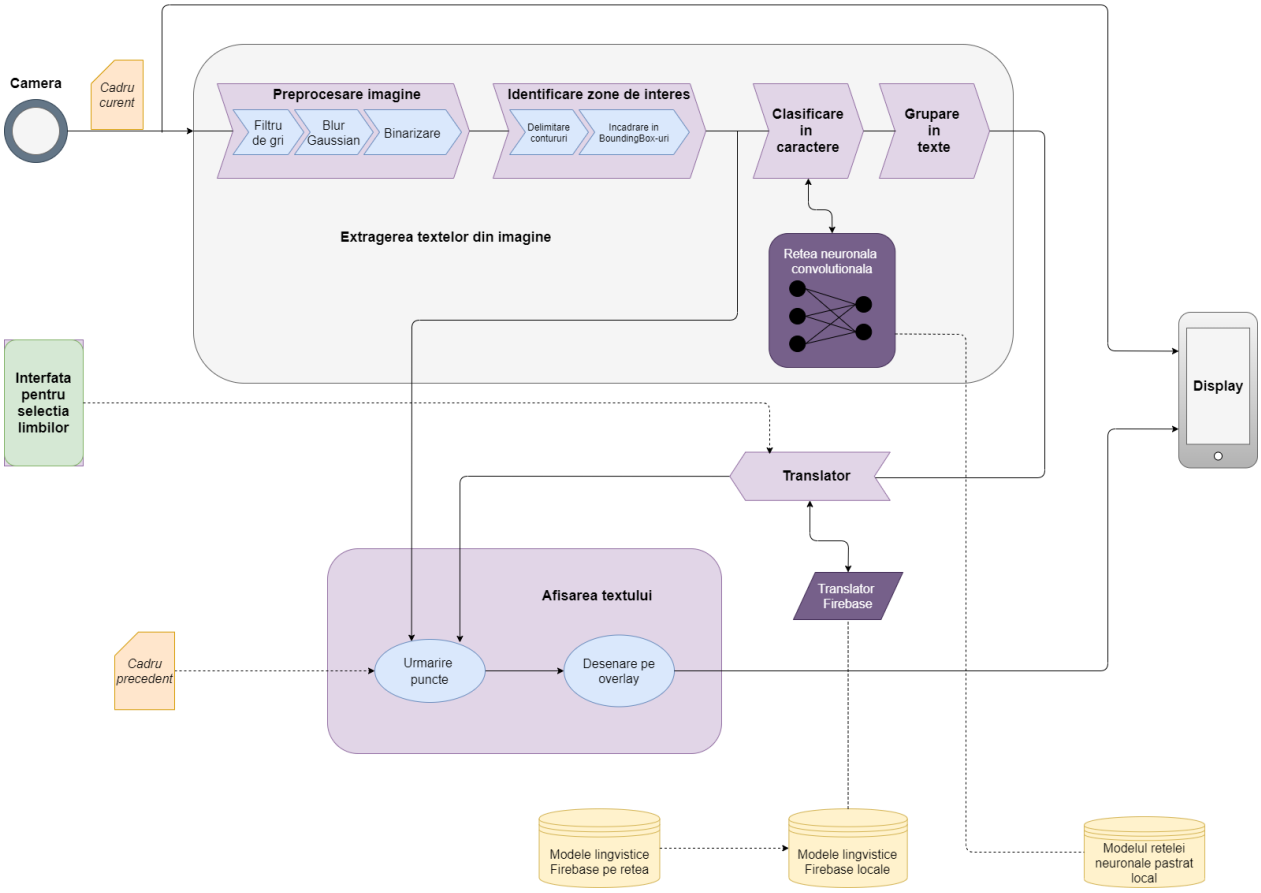

Figura 1 Arhitectura

Capitolul 1: Identificarea textului în imagini

Acest capitol va urmari implementarea unui pipeline care realizeaza identificarea textelor în imagini prin identificarea carcterelor din aceasta, iar apoi gruparea lor in secțiuni de text. Primul pas firesc a fost realizarea arhitecturii si a pașilor ce trebuie urmați(Figura 2). În continuare am ales implementarea unui cod python care să realizeze acești pași secvențiali și să îmi permită testarea cu usurință a rezultatelor intermediare, cât si determinarea parametrilor optimi pentru funcțiile utilizate. Acesta a fost utilizat si in scopul antrenării rețelei neuronale utilizate in proiect. Odata ce codul python a fost implementat cu succes, iar toți parametrii au fost aleși implementarea cu succes in aplicatia Android a fost realizată într-un timp mult mai scurt.

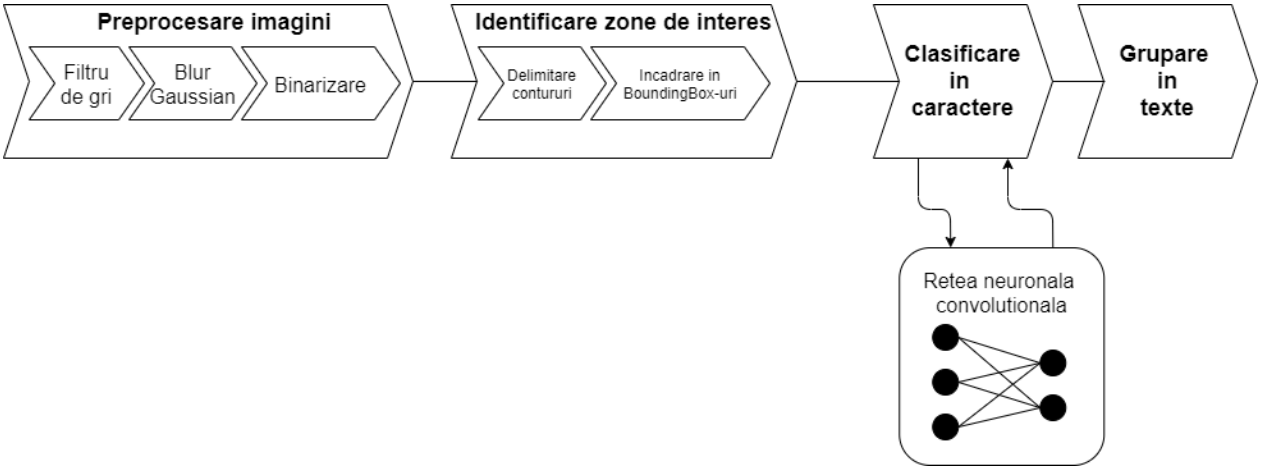

Figura 2 Pipeline-ul modulului de identificare a textului

Capitolul va urmari cele 4 elemente ale piepeline-ului in oridine, motivând totodata utilzarea lor pentru problema de față, acestea fiind:

- Preprocesarea unei imagini, cu scopul de a elimina elemente ce nu sunt de interes și de a pregăti imagine pentru un pas urmator de procesare
- Identificarea unor zone de interes sub forma unor casete ce vor fi clasificate in continuare
- Clasificarea casetelor în clasa de caractere aferentă dacă este posibil
- Grupara casetelor ce conțin caractere in texte

Preprocesarea imaginii

Preprocesarea imaginii are că scop pregătirea unei imagini primite ca input, prin diferite transformări matematice, pentru a fi folosită în continuare în pipeline-ul algoritmului. În contextul aplicației de față motivul exclusiv al preprocesarii este pentru determinarea contururilor ce pot fi identificate în imagine ce constituie zone de interes prin aplicarea ulterioară a unei funcții ce îndeplinește această funcție. De asemenea pentru a reduce numarul acestor zone se aplica si un filtru care netezeste imaginea. Astfel pentru a obține zone cât mai relevante în continuare suntem nevoiți să aplicăm diferite filtre asupra imaginii inițiale.O imagine ce intră în procesul de preprocesare va fi supusă următoarelor transformări:

- Filtru de gri
- Blur Gaussian
- Binarizare

Aplicarea unui filtru de gri sau reducerea imaginii la nuanțe de gri este necesară pentru un pas următor și anume binarizarea imaginii. Acesta transformă imaginea dintr-un format RGB în care fiecare pixel are trei valori , la o imagine care este compusă doar din nuanțe de gri în care fiecare pixel reține doar o valoare întreagă care definește nuanța de gri.

```
Mat grayImage = new Mat (image.width(), image.height(), CvType.CV_8UC1);
Imgproc.cvtColor(image, grayImage, Imgproc.COLOR_BGR2GRAY);
```

Aplicarea unui filtru de blurare Gaussian poartă un rol important în reducerea de zgomot (“noise”) din imagine, lucru ce conduce la o mai bună delimitare a caracterelor și o reducere a numărului de contururi identificate, astfel și al vitezei generale al întregului program. Motivul pentru care am ales aceast algoritm și nu filtru bilateral care păstrează mai bine contururile este diferența de viteză substanțială dintre cei doi algoritmi în practică, în medie 0.05s pentru Gaussian Blur în comparație cu 1s în medie pentru Bilateral Filter.

```
Mat filteredImage = new Mat();
Size s = new Size(3,3);
Imgproc.GaussianBlur(image, filteredImage, s, 2);
```

Gaussian Blur este o metodă de netezire a imaginii (“Image Smoothing”) care folosește un kernel căruia îi aplică funcția Gaussiană pentru a reduce nivelul de zgomot, astfel că trecerile de la o culare la altă sunt mai fine, rezultând astfel și într-o trecere mai fluidă între elementele delimitate în imagine, contururile elementelor devenind astfel mai clare. Acesta este un obiectiv foarte important, deoareace pentru o imagine cu mult zgomot se vor delimita mai multe contururi care nu sunt relevante , astfel încetinind execuția . Există totuis și un factor de risc pentru care nu se poate folosi un blur prea puternic și anume momentul în care imaginea rezultată unește două elemente ce sunt disticte în mod natural datorită unui blur prea extins. De aici și parametrii aleși pentru funcția de blurare. Un bun exemplu se poate observa in Figura 3.

Figura 3 Blur Gaussian - examplu

Binarizarea unei imagini constă în reducerea imaginii doar la alb și negru, proces necesar pentru obținerea ulterioară a contururilor. Acest lucru îl realizăm utilizând metoda Image Tresholding, iar pentru proiectul de față am considerat că cel mai potrivit algoritm de tresholding este Adaptive Gaussian Tresholding. Această funcție , așa cum se poate bserva și în Figura 4 returnează dintr-o imagine în nuanțe de gri, o nouă imagine care utilizează doar alb și negru pentru a delimita elementele care o compun. Imaginea stă la baza detecției viitoare a contururilor, fiind un pas extrem de imporntant, cum elementele neconexe ce compun această imagine vor fi și elemetele ce vor fi clasificate pentru a se vedea dacă aparțin unei clase de caractere.

În Figura 4 se pot observa diferențele între rezultatele algoritmilor de thresholding de unde și decizia de a folosi Adaptive Gaussian Thresholding.

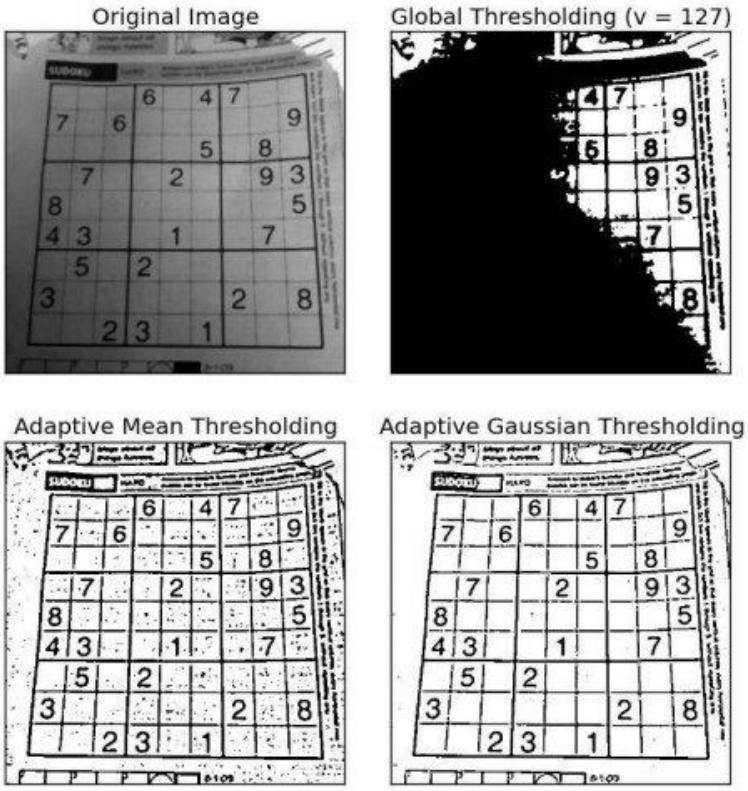

Figura 4 Binariazarea unei imagini - comparatie intre algoritmi

Adaptive Gaussian Tresholding folosește un treshold care se adaptează in funcție de pixelii din apropiere, lucru ce ajută pentru zonele ce ar putea avea diferite condiții de lumină. Avantajul acesta o plasează deasupra medodei

Global Thresholding, cum condițiile de lumina diferite sunt un factor care variază foarte mult în scenele naturale, iar această metodă după cum se observă nu satisface foarte bine acest criteriu avnd thresholdul fix. Din același motiv al diferitelor condiții de lumină am ales utilizarea acestei metode și asupra metodei preferate, anume binarizrea Otsu. Iar Adaptive Mean Thresholding deși se adaptează la condițiile de lumina returnează o imagine cu mult zgomot, lucru care după cum am menționat precedent am încercat să evit pentru a minimiza numărul de contururi obținute, care nu reprezintă caractere.

```
Mat binaryImage = new Mat();
Imgproc.adaptiveThreshold(filteredImage,binaryImage,255,
                          ADAPTIVE_THRESH_GAUSSIAN_C,THRESH_BINARY,11,2);
```

Obținerea casetelor propuse pentru verficare

În această secțiune urmărim căutarea unui set de BoundingBox-uri propuse pentru a fi clasificate mai departe în clasele corespunzătoare de caractere dacă acest lucru este posibil. Pentru aceasta utilizăm din nou două funcții ale bibliotecii OpenCV, una care caută contururile într-o imagine și una care obține din aceste contururi BoundingBox-urile care le includ, ce vor fi stocate în structuri și trimise în continuare pentru a le fi sau nu asignate un caracter.

Prin BoundingBox înțelegem o zona dreptunghiulară care poate fi identificată prin coordonatele vârfului din stânga sus, lungime și înălțime.

În prima faza aplicăm funcția findContours din biblioteca OpenCV peste imaginea preprocesată până în acest punct care va returna o listă în care fiecare element al listei este o listă de coordonate care descrie un contur identificat în imagine. Algoritmul folosește metoda descrisă în Topological structural analysis of digitized binary images by border following. Descrierea sumară a metodei preluată din abstractul lucrării:

"Doi algoritmi care urmăresc granița sunt propuși pentru analiza topologică a imaginilor binare digitalizate. Primul determină relațiile de "surroundness" dintre granițele unei imagini binare. Deoarece marginile exterioare și marginile găurilor au o corespondență unu la unu cu componentele conectate de 1 pixeli și, respectiv, cu orificiile, algoritmul propus obține o reprezentare a unei imagini binare, din care se poate extrage caracteristici fără a reconstrui imaginea. Al doilea algoritm, care este o versiune modificată a primului, urmărește doar marginile cele mai exterioare (adică, marginile exterioare care nu sunt înconjurate de găuri). Acești algoritmi pot fi utilizați în mod eficient în numărarea componentelor, micșorarea și analiză structurală topologică a imaginilor binare, atunci când este utilizat un computer digital secvențial."

```
List<MatOfPoint> contours = new ArrayList<>();
Mat hierarchy = new Mat();
Imgproc.findContours(binaryImage,contours,hierarchy,RETR_TREE,
                     CHAIN_APPROX_SIMPLE);
```

În Figura 5 sunt reprezentate cu verde contururile identificate de algoritm, care deși la o prima vedere în număr foarte mare, după o trecerea printr-o serie de condiții care le validează în funcție de arie și proporția dintre lungime și lățime devin mult mai reduse. Acest lucru se poate observă în Figura 6 unde sunt reprezentate cu verde BoundingBox-urile contururilor păstrate în urmă acestor eliminări.

Figura 6 Contururi identificate

Figura 5 BoundingBox-urile propuse

Odată obținută lista contururilor iteram prin această și folosim funcția contourArea, din OpenCV, pentru a ignora acele contururi ce au suprafața foarte mică și nu ar face decât să încetinească algoritmul și acelea care au aria de o dimensiune foarte mare (mai mare decât jumătate din aria imaginii). Asupra restului de contururi aplicăm funcția OpenCV, boundingRect pentru a obține un BoundingBox care încadrează perfect conturul și creem o lista de obiecte de tip PredictionBox(Figura 7) în care le stocăm.

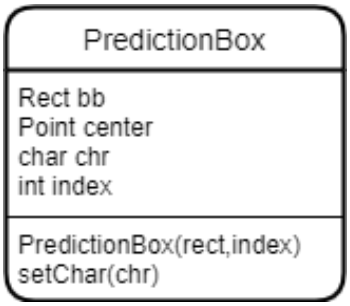

Figura 7 PredictionBox class

Clasificarea caracterelor

Rețeua Neuronală

Pentru a identifica dacă o imagine reprezintă un caracter sau o cifră utilizăm o rețea neuronală convoluțională. Această rețea este antrenată în modulul Python apoi exportată și importată în apliația Android care o va utiliza.

Rețeaua convoluțională este compusă din 10 nivele care pot fi observate în fig 11. Inputul este o matrice pătratică de dimensiune 32x32 care corespunde unei imagini alb-negru. Utilizarea convoluțiilor și pooling-ului au facilitat recunoașterea caracterelor în scene naturale. Outputul reprezintă un vector cu 63 de poziții, primele 10 reprezentând în ordine cifrele 0-9, următoarele 26 reprezentând pe rând literele mari ale alfabetului A-Z, următoarele 26 reprezentând pe rând literele mici ale alfabetului a-z, iar ultima poziție reprezentând neaparetenență la o clasă de caractere dintre cele precedente (astfel fiind eliminat în cele ce urmează).

Setul de antrenament constă în:

- Setul de date Chars74k conținând caracterele 0-9,A-Z,a-z în scene naturale, împarțite în:
    - imagini clare (Figura 10)
    - imagini neclare (Figura 9)
- Setul de date INRIA Holidays care conține scene naturale și din care selectez secțiuni aleatoare învațând rețeaua să clasifice orice nu e caracter într-o a 63-a clasă (Figura 8)

Dupa ce un model a fost antrenat si testat in contextul programului Python, acesta necesită converitirea în formatul tflite pentru a putea fi utilizat in aplicația Android.

```
model = tf.keras.models.load_model('keras_model.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("model.tflite", "wb").write(tflite_model)
```

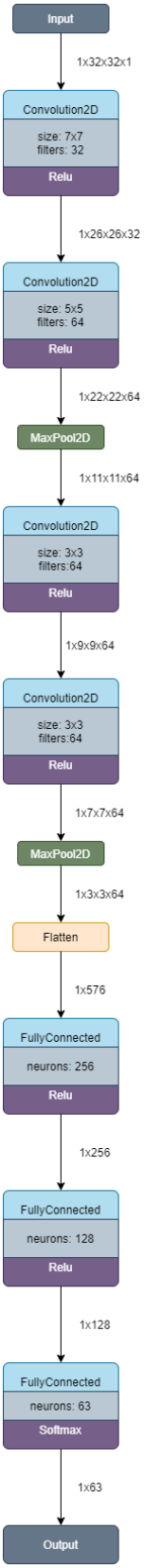

Figura 11 Arhitectura retelei neuronale convolutionale

Figura 10 Imagini clare cu caractere – Chars74K dataset

Figura 9 Imagine neclare cu caractere – Chars74K dataset

Figura 8 Scene naturale – INRIA Holidays dataset

Implementarea în Android

Clasificarea de caractere o vom realiza parcurgând fiecare element al listei de obiecte de tipul PredicționBox, obținute în secțiunea precedentă, și clasificând fiecare dintre acestea utilizând reteua neuronală antrenată în programul Python. Pentru aceasta avem nevoie de o metodă de a rula o rețea neuronală în Android, iar acest lucru este realizat prin instalarea bibliotecții TfLite, adăugând dependințele necesare în fișierul build.gradle. Acest lucru va instala bibliotecile necesare pentru lucrul cu rețele neuronale, iar în cazul nostru va permite importarea modelului deja antrenat din modulul python.

```
android {
   aaptOptions {
        noCompress "tflite"
    }
}

dependencies {
    implementation 'org.tensorflow:tensorflow-lite:2.2.0'
}
```

Inițializarea modelului citit din fișierul importat “model.tflite” se realizează o singură dată la prima pornire a aplicației și utilizează modelul comprimat al rețelei neuronale pentru a inițialaliza rețeaua care va deservi algoritmul de clasificare în cele ce urmează. Astfel se inițializeaza un file descriptor, căruia i se atribuie path-ul către fișierul ce conține modelul, pentru a fi accesat. Apoi fișierul este mapat direct în memorie, iar ByteBuffer-ul rezultat din acest procedeu este returnat unei noi instanțe de Interpreter, care va încarcă modelul și va acționa apoi pe post de clasificator, rulând rețeau neuronală convolutionala incăracată peste datale ce le vom trimite ca input și returnând outputul așteptat că în utilizarea în python.

Odată inițializat modelul este ținut în memorie pentru a fi utilizat pe toată durata în care aplicația este în execuție. Atunci când aplicația intră în repaus modelul este abandonat până când aplicația este din nou rulată, astfel asigurăm faptul că memoria nu este încărcată inutil cu acest model pe perioada când acesta nu este rulat.

```
private static final String MODEL_PATH = "model.tflite";

private MappedByteBuffer loadModelFile(Context context) throws IOException {
        AssetFileDescriptor fileDescriptor = context.getAssets().openFd(MODEL_PATH);
        FileInputStream inputStream =
                new FileInputStream(fileDescriptor.getFileDescriptor());
        FileChannel fileChannel = inputStream.getChannel();
        long startOffset = fileDescriptor.getStartOffset();
        long declaredLength = fileDescriptor.getDeclaredLength();
        return  fileChannel.map(FileChannel.MapMode.READ_ONLY,
                                startOffset,
                                declaredLength);
}
```

Pentru fiecare element din lista de obiecte de tipul PredicționBox, se decupează din imaginea originală (cu filtru de gri) secțiunea cuprinsă în BoundingBox-ul reținut în acesta și se redimensionează imaginea la dimensiunea de 32x32 pixeli. Această nouă imagine este apoi convertită într-un array 4-dimensional de tip float care va fi pasat clasificatorului inițializat mai sus rulând-ul prin comanda run.

```
float[][][][] input = convertBitmapToFloatArr(resizedImage);

float[][] output = new float[1][63];

clasifier.run(input,output);
```

Clasificatorul (rețeaua neuronală) returnează vectorul cu 63 de poziții, de unde se extrage caracterul aferent imaginii secționate, verificând care dintre cele 63 de clase are valoare cea mai mare de probabilite. Cum utilizăm funcția softmax pe ultimul nivel al rețelei, putem spune că valorile rezultate reprezintă confidența rețelei pentru a incadra imaginea primită în fiecare dintre cele 63 de clase. În cazul în care imaginea este clasificată în cea de a 63-a clasa (nu conține nici un caracter căutat) sau gradul de confidență a rețelei asupra carcterului este mai mic decât 60%, obiectul PredictionBox este eliminat, în caz contrar obiectului îi este atribuit caracterul determinat.

După acest pas avem o listă de elemente PredictionBox, care au atribuite caracterele care le descriu in imaginea inițială.

Gruparea caracterelor identificate în regiuni de text

Algoritmul de grupare al caracterelor are ca scop asocierea elementeor de tip PredictionBox identificate până în acest punct în pipe-line, în secțiuni de caractere care formează text. Grupările identificate vor fi memorate într-o nouă structură, WordBox (Figura 12), care va cuprinde atât textul propriuzis cât și un BoundingBox care îngloabează suprafața pe care acesta îl ocupă în imagine. Algoritmul va încerca pentru fiecare caracter să găsească recursiv o secvență de litere care îndeplinesc un set de condiții.

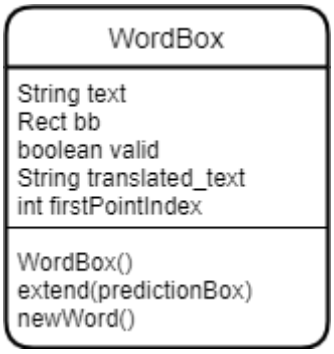

Figura 12 WordBox class

O primă etapă este sortarea elementelor PredictionBox după poziția centrului, pornind de la stânga la dreapta. Cum încercarea completării de cuvinte este făcută de la stânga la dreapta și nu se revine asupra unui cuvânt deja format acest pas asigură formarea de grupări maximale de caractere.

Pentru a grupa caracterele în text pornim cu ipoteza că acestea sunt pe aceeași linie, textul este de scris de la stânga la dreapta și distantele dintre literele unui cuvânt sunt mai mici decât distanțele dintre cuvinte. Astfel pentru a realiza gruparea, creeam o matrice de distanțe care va cuprinde toate distanțele între toate caracterele (centrele acestora). În același timp încercăm și eliminarea unor eventuale BoudingBox-uri rămase care nu reprezintă caractere (cum ar fi spațiul gol din interiorul literei “P” care ar putea fi clasificat în sine că litera “D” și ar diminua calitatea rezultatului) verificând suprapunerile și eliminând suprafața mai mică.

Ultimul pas va fi gruparea efectivă în cuvinte, care se realizează parcurgând lista sortată element cu element și căutând pentru fiecare cel mai apropiat element de acesta, apoi verificând dacă:

- unghiul format de dreapta ce unește centrele și dreapta ce unește centrul primului element cu centrul elementului precedent lui este mai mică decât deviația maximă acceptată (dreaptă inițială este verticală)
- distantă dintre cele două, fără a lua în considerare partea interioară a BoundingBox-ului este mai mică decât :
    - distantă maximă acceptată pentru caractere din același cuvânt
    - distantă maximă acceptată pentru caractere din același text (caz în care se aduga un spațiu în textul format)

Când un set de două caractere îndeplinesc regulile precedente, se creează un obiect de tipul WordBox și se adaugă cele două caractere la textul acestuia și se calcueaza un BoundingBox care înglobează ambele elemente, apoi cel de-al doiea caracter devine noul element de verificat, continuând recursiv și adăugând textul la elmentul WordBox creat până când nu se mai pot aplica regulile cu succes. Apoi se reia parcurgerea listei până la final. Textele care nu sunt formate din cel puțin două caractere nu sunt luate în considerare.

În Figura 13 se poate observa schematic modul în care lucrează algoritmul de grupare al caracterelor descris precedent.

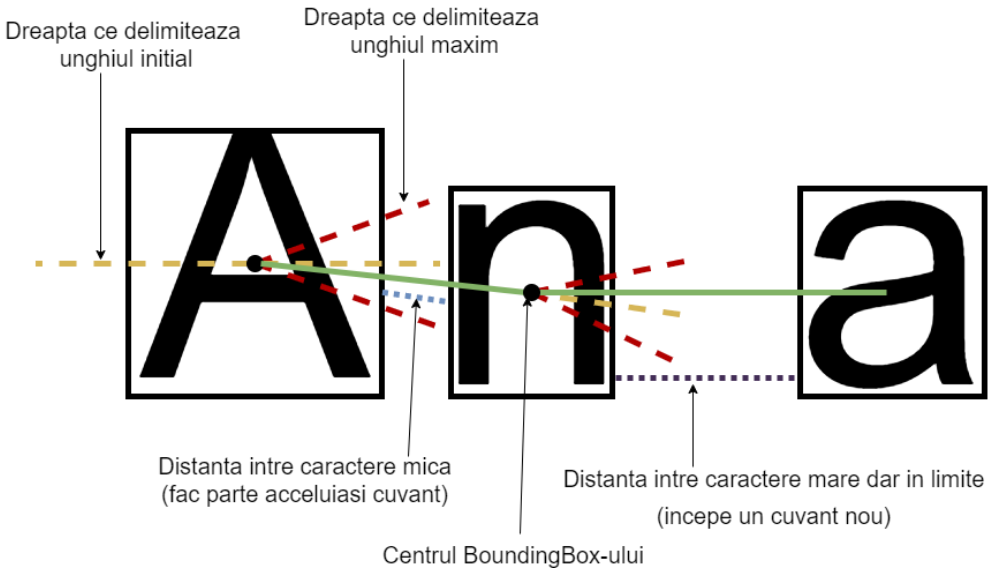

Figura 13 Gruparea caracterelor

```
for(int i = 0;i < n;i++) {
  int j = i;
  double angle = 0;
  WordBox group = new WordBox();
  groups.add(group);
  while (visited[j] == false) {
    group.extend(predictionBoxes.get(j));
    visited[j] = true;
    int k = argMin(distances[j]);
    if(k!=-1) dist = distances[j][k];
    else dist = Float.POSITIVE_INFINITY;
    while (k!=-1 && dist<word_distance_treshold * predictionBoxes.get(k).bb.height){
      y = predictionBoxes.get(k).center.y - predictionBoxes.get(j).center.y;
      x = predictionBoxes.get(k).center.x - predictionBoxes.get(j).center.x;
      if (x == 0) x = 0.00000000000000001;
      check_angle = toDegrees(atan(y / x)) - angle;
      if (visited[k] == false &&
          min_angle <= check_angle && check_angle <= max_angle &&
          predictionBoxes.get(j).bb.height/2 <= predictionBoxes.get(k).bb.height &&
          predictionBoxes.get(k).bb.height <= predictionBoxes.get(j).bb.height*2){

          if (dist >= letters_distance_treshold * predictionBoxes.get(k).bb.height)
            group.newWord();

          distances[j][k] = distances[k][j] = Float.POSITIVE_INFINITY;
          angle = check_angle;
          j = k;

          break;
       } else {
          distances[j][k] = distances[k][j] = Float.POSITIVE_INFINITY;
          k = argMin(distances[j]);
          if(k!=-1) dist = distances[j][k];
          else dist = Float.POSITIVE_INFINITY;
       }
     }
     if(k == -1 || dist >= word_distance_treshold*predictionBoxes.get(k).bb.height)
       break;
  }
}
```

Capitolulu 2: Afișarea textului tradus

Spre deosebire de capitolul precedent care urmărea pipeline-ul de detecție a textelor în imagine, capitolul acesta urmărește utilizarea informațiilor extrase din imagine. Aceste informații în combinație cu contextul dinamic al aplicației conferă vizualizarea textelor extrase din cadrele video vizualizate imbunanațind experiența utilizatorului. Traducerea textelor conferă aplicabilitatea practică a aplicației în contextul utilizării de zi cu zi. Astfel se urmăresc trei etape de procesare a acestor informații:

- Traducerea textelor identificate
- Urmărirea unor coordonate detectate în decursul flow-ului video
- Afișarea textelor traduse sub formă unor casete care urmăresc textele de care aparțin

Pașii urmăriți sunt inițial o traducera a tuturor textelor care au fost identificate pentru ultimul cadru, după ce acesta este procesat, timp în care se face urmărirea coordonatelor BoundingBox-urilor determinate în procesul de identificare a zonelor de interes și urmărirea și afișarea casetelor cu text prezente în momentul de față pe ecran, de la ultima detecție de text finalizată până la acel moment în timp. Aceste lucruri se realizează independent de pipeline-ul de detecție, astfel fiind neîntrerupt de procesarea unei imagini. Modulul renunta la punctele ce sunt urmărite și le înlocuiește cu altele noi de fiecare dată când detectorul de text ii trimite un nou set de puncte. De asemenea modulul primește constant de la camera video cadrul cel mai recent, folosindu-se de acesta și de cel precedent pentru a urmări constant punctele necesare în contextul video.

Traducerea textelor, realizată utilizând modele de traducere salvate local conferă aplicabilitatea practică a aplicației , oferind posibitatea de utilizarea a detecției de texte pentru o problemă reală .

Afișarea prin urmărire a casetelor are că scop o mai bună percepție asupra rezultatelor obținute, dat fiind faptul că într -o aplicare practică camera dispozitivului va fi constant deplasată sau chiar textele în sine , iar acestea își vor schimbă pozițiile în concordanță, astfel urmărirea lor devine o necesitate pentru cazul real de utilizare.

Traducerea textelor

Pentru a traduce textele identificate am dorit utilizarea unui modul local de traducere pentru a evita depedența de rețea pe tot parcursul utilizării aplicației, cât și de a mări viteza în care aceasta se realizează, cum nu mai necesită trimiterea textului pentru a fi analizat până la un server. Astfel am ales utilizarea Firebasse ML Kit care oferă posibilitatea de a descarca modele responsabile pentru traducere doar în momentul în carea acestea nu au fost descărcate la un momement precedent de la instalarea aplicației, restricționând astfel necesitatea accesului constant la o rețea pentru a traduce textele.

Pentru a utiliza modulul de translație oferit de Firebase trebuie adăugate în primă instanță referințele nesare pentru bibliotecă. Apoi trebuie creeat un cont ce va fi sincronizat cu proiectul și va permite utilizarea serviciilor de pe cloud, cum ar fi în cazul de față descarcarea modelelor ce realizează traducerea textelor.

```
android {
    aaptOptions {
        noCompress "tflite"
    }
}

dependencies {
    implementation 'org.tensorflow:tensorflow-lite:2.2.0'
    implementation 'com.google.firebase:firebase-ml-natural-language:22.0.0'
    implementation 'com.google.firebase:firebase-ml-natural-language-
translate-model:20.0.8'
}
```

Fiecare text, înainte de a fi afișat pe ecran este trecut prin translator. Dacă nu sunt setate limbile din care se face translația sau utilizatorul alege că textul să nu mai fie tradus, textul inițial va fi returnat nealterat. Odată ce utilizatorul specifică limba de proveniență a textului și limba în care dorește ca acesta să fie tradus, translatorul va fi inițializat, iar orice nou text care va fi trecut prin acesta va fi tradus ca atare dacă este posibil, în caz contrar textul va fi returnat nemodificat.

Translatorul Firebase ML Kit utilizează limba engleză ca limbă intermediară pentru a realiza traducerile, dacă nici una din limbile selectate nu este engleză, astfel calitatea traduceri poate fi puțin afectată in funcție de limbă.

Modulul de translație permite descărcare modelului de traducere dacă acesta nu a fost descărcat până în acel punct. O dată ce un model a fost descărcat acesta este stocat în fișierele aplicației, permițând utilizarea lui fără a necesita accesul la rețea de fiecare dată cand un text se dorește tradus. Dacă modelul nu a fost descărcat cu succes aplicația va returna un mesaj de eroare: “Could not download language model!”.

```
public boolean setLanguages(Integer languageFrom, Integer languageTo) {
  FirebaseTranslatorOptions options = new FirebaseTranslatorOptions.Builder()
                        .setSourceLanguage(languageFrom)
                        .setTargetLanguage(languageTo)
                        .build();
  translator = FirebaseNaturalLanguage.getInstance().getTranslator(options);

  FirebaseModelDownloadConditions conditions = new
      FirebaseModelDownloadConditions.Builder()
                .build();

  translator.downloadModelIfNeeded(conditions)
             .addOnSuccessListener(new OnSuccessListener<Void>() {
                  @Override
                  public void onSuccess(Void aVoid) {
                      returnSameText = false;
                  }
              })
              .addOnFailureListener(
                      new OnFailureListener() {
                          @Override
                          public void onFailure(@NonNull Exception e) {
                              returnSameText = true;
                          }
                      });
  return !returnSameText;
}
```

Urmărirea textului in contextul dinamic

Deoarece aplicația utilizează un context dinamic în detectarea și afișarea textelor este nevoie de un modul care să urmărească si să repoziționeze obiectele detectate în concordanță cu deplasarea lor în reprezentarea video oferită de camera dispozitivului. Astfel aplicația monitorizează, utilizând metoda Lucas and Kanade, două seturi de coordonate:

- coordonatele punctelor din coltul stânga sus al BoundingBox-urilor identificate ca zone de interes
- coordonatele punctelor din coltul stânga sus al BoundingBox-urilor ce descriu o secțiune de text identificată

Algoritmul Lucas-Kanade face câteva ipoteze implicite:

- Cele două imagini sunt separate printr-un interval de timp mic, in așa fel incat încât obiectele nu s-au deplasat semnificativ (adică algoritmul funcționează cel mai bine cu obiecte cu mișcare lentă).
- Imaginile prezintă o scenă naturală care conține obiecte texturate care prezintă nuanțe de gri (diferite niveluri de intensitate) care se modifică fără probleme.

Algoritmul nu folosește informațiile despre culoare în mod explicit. Aceasta nu scanează a doua imagine căutând o potrivire pentru un pixel dat. Funcționează încercând să ghicească în ce direcție s-a mișcat un obiect, astfel încât schimbările locale în intensitate pot fi explicate.

Deoarece din momentul în care un frame a fost preluat din camera pentru a fi prelucrat de algoritmul de detecție până la momentul de afișare al rezultatelor poate fi substanțial, iar camera dispozitivului își va schimba cel mai probabil poziția în acest interval sau obiectul ce contine text va fi mutat de la pozitia lui de start, am ales să urmăresc deplasarea punctelor provenite din modulul de identificare a zonelor de interes, deoarece acestea sunt obținute int-un timp scurt, iar aceste casete vor compune in final noile casete care descriu textul. Odată ce se obțin zonele de text de interes pentru afișare se poate calcula poziția la care acestea vor trebui afișate, în concordanță cu deplasarea primului element WordBox(care a fost urmărit) care face parte din

acesta. Prin această parte asigurăm că BoundingBox-urile ce conțin text sunt afișate pentru prima dată în pozițiile în care se află respectivul text în cadrul curent.

Pentru a păstră textele afișate în pozițiile în care acestea sunt în imagine urmărim coordonatele casetelor ce conțin text și le schimbăm poziția de afișare de fiecare dată când detectăm o modificare a acestora.

De fiecare dată când un nou set de coordonate este pasat algoritmului de urmarire se renunță la setul precedent de coordonate de același tip. Functionarea este reprezentată schematic in Figura 14, unde se poate observa urmarirea punctelor pentru doua cadre consecutive trimise in procesul de extragere a textului. Astfel la scurt timp dupa ce primul cadru intră în procesul de detecție și recunoastere al textului se trimite lista de elemente PredictionBox către modulul responsabil cu urmărirea, care incep din acel moment sa fie urmărite. Când predicția a fost finalizată se utilizează coordonatele urmărite pentru a calcula noile poziții ce vor începe să fie urmărite, reprezentând coordonatele casetelor cu texte. După aceasta la foarte scurt timp un nou set de puncte de tip coordonate de caractere este primit și se renunta la punctele precedente. La fel la finalizarea predicției pentru al doilea set se renunta la setul de coordonate urmărite pentru texte și se înlocuiesc cu noul set. În mod constant se realizează afișarea casetelor text la noile poziții calculate.

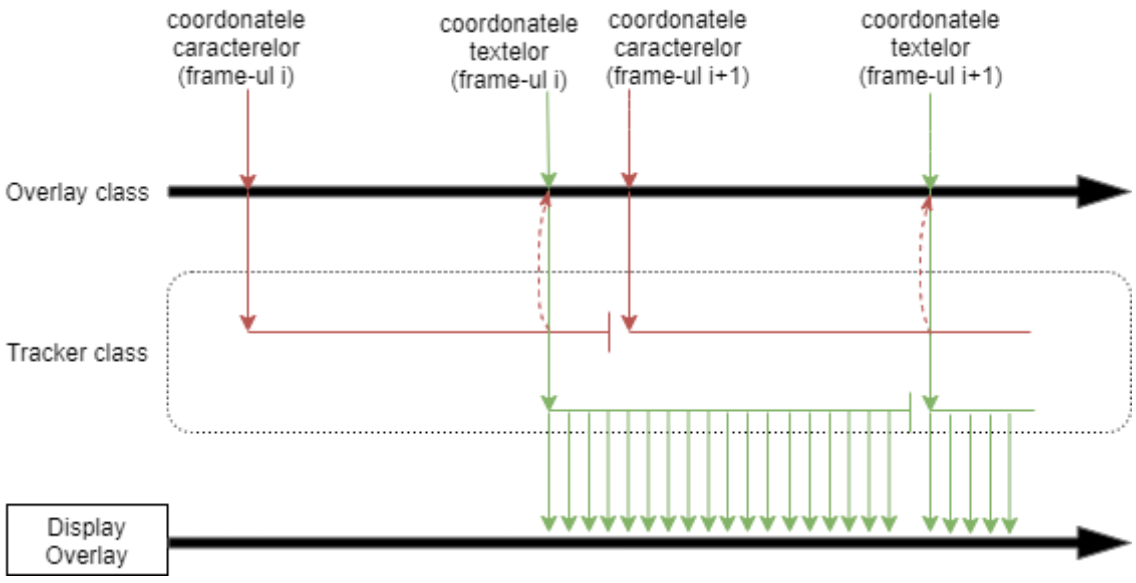

Figura 14 Urmarirea punctelor de interes - reprezentare schematică

Afișarea propriuzisă a rezultatelor obținute de algoritm

Afișarea constantă a predicțiilor obținute până în acest punct se realizează pe un obiect TextureView poziționat deasupra previzualizarii camerei. Cum obiectul este transparent se poate vizualiza afișajul camerei video prin acesta, iar când utilizăm o funcție de desenare putem desena casete cu text care să suprapună textele identificate având locațiile acestora în imagine.

Înainte de a afișa casetele verificăm deplasarea lor față de cadrul precedent capturat de cameră și se recalculează pozițiile acestora in conformitate cu deplasarea casetelor cu caracterele ce le compun, care au fost urmarite până la acel punct.

Pentru afișarea constantă a casetelor la fiecare finalizare a unei desenări pe canvas-ul view-ului invalidăm afișarea, ceea ce forțează reafișarea, astfel și refacerea pașilor, creeând un ciclu constant de afișare ce este întrerupt doar atunci când aplicația este pusă în repaus si reintră în buclă odată ce aplicația este din nou pornită.

```
for(WordBox box : predictions) {
  Paint letterPaint = new Paint(textPaint);
  letterPaint.setTextSize((int)(box.bb.height*0.8));
  letterPaint.setStrokeWidth(3);
  Paint boxPaint = new Paint(backgroundPaint);
  boxPaint.setStrokeWidth(3);
  canvas.drawRect(box.bb.x*REDUCE_IMAGE_BY_FACTOR,
                  box.bb.y*REDUCE_IMAGE_BY_FACTOR,
                  box.bb.x*REDUCE_IMAGE_BY_FACTOR+box.bb.width,
                  box.bb.y*REDUCE_IMAGE_BY_FACTOR+box.bb.height,
                  boxPaint);
  canvas.drawText(String.valueOf(box.translated_text),
                  box.bb.x*REDUCE_IMAGE_BY_FACTOR,
                  box.bb.y*REDUCE_IMAGE_BY_FACTOR+box.bb.height*0.9f,
                  letterPaint);
}
if(currentFrame!= null)
  oldFrame = currentFrame.clone();
postInvalidate();
```

Experimente

În realizarea lucrării am încercat multiple idei și variante de rețele neuronale, ajungând într-un final la varianta de față. Variațiile în arhitectură sunt reprezentate de metode diferite de obținere a caracterelor din imagine.

O arhitectură precedentă utiliza o rețea neuronală convolutională antrenată asemănător cu cea actuală, având tot 63 de clase din cară ultima pentru imagini fără caratere, dar spre deosebire de cea actuală această era în întregime convolutionala. Acest fapt a permis utilizarea metodei "sliding-window", copiind valorile de pe layer-ele intermediare într-o altă rețea care avea aceleași layer-e cu exepția primului, care era setat de dimensiunea întregii imagini primite ca input. Rețeaua obținută fiind astfel în întregime convolutională returna o matrice în care fiecare element reprezenta o secțiune din imagine, iar fiecare element era format dintr-un vector cu 63 de poziții, care defineau caracterele asemeni celei din rețeaua curentă. Ultimul layer al rețelei utiliza funcția softamx pe a treia dimensiune a outputului, returnând pentru cele 63 de clase certitudinea rețelei pentru fiecare caracter în casetă respectivă . Modul în care ar fi urmat să fie implementată identificare cacterelor era trecearea aceleiași imagini redimensionata la mai multe dimensiuni pentru a extrage caracterele de mărimi diferite. Apoi pentru obținerea textelor din fiecare output al rețelei ar fi trebuit selectate acele elemente din matrice ce aveau valoarea corespunzătoare unui caracter cât mai mare și gruparea acestora după poziția în matrice în cuvinte și texte.

Avantajele unei astfel de metode constau într-un timp constant al detecției textului pentru imagini, posibil chiar mai scurt decât cel obținut cu metodă folosită în final și identificarea poziției caracterelor cât și a claselor din care fac parte în aceași etapă.

Dezavantuajele care au dus la schimbarea metodei au fost o rețea greu de antrenat deoarece casetele având o dimensiune fixă cuprideau uneori alte date de fundal sau caracterul incomplet ce ducea la o predicție incorectă și rularea pentru un număr mare de mărimi posibile al textului ar fi dus la un timp mai îndelungat de execuție. Un alt inconvenient a constat în gruparea caracterelor deoarece mai multe casete ajungeau să prezică același caracter, deplasarea casetelor fiind mică, lucru care făcea destul de complicată identificarea cu precizie a caracterului și gruparea lui în cuvinte.

Dezavantajele creeate de această metodă au dus la utilizarea metodei curente care rezolva aceste probleme, dar nu excelează în timpul de execuție, fiind dependență de complexitatea imaginii.

De asemenea în implementarea metodei curente am incerat multiple rețele neuronale, atât nonconvolutionale cât și mixte până la antrenarea celei curente care a oferit cele mai bune rezultate în practică . În aceași măsură am încercat și utilizarea a diferite funcții pentru preprocesarea imaginii pentru a obține rezultate cât mai bune. Iar pentru gruparea caracterelor am considerat de asemenea utilizarea clusterizarii hierarhice, dar am ales în final utilizarea unei clusterizari bazate pe distanțele între elemente și un set de reguli de aliniere pentru rezultate mai bune.

Manualul de utilizare

În momentul lansării aplicației utilizatorul este întâmpinat de o fereastra care cuprinde întreg ecranul prezentând doar o vizualizare a camerei video a dispozitivului și un buton centrat în partea de jos. (Figura 15)

Figura 15 Interfaa la prima vedere

Algoritmul de detectare a textului pornește automat, implicit fără a traduce textele, iar după câteva secunde îndreptând camera dispozitivului spre o zonă cu text și aliniind textul vertical se vor afișa casete care acoperă textele identificate și în care este scris textul care a fost identificat. (Figura 16)

Figura 16 Afișarea textului detectat

În partea de jos a ecranului se poate observa un buton cu textul “Translate” prin apăsarea căruia se va deschide un meniu nou.Meniul cuprinde:

- în stângă o listă de limbi din care se poate alege doar una, iar cea selectată va fi marcată cu albastru, reprezentând limba de proveniență a textului care va fi tradus
- în dreapta o listă de limbi din care se poate alege doar una, asemenea cea selectată va fi marcată cu albastru, reprezentând limba de proveniență a textului în care se dorește traducerea textelor identificate

- trei butoane:
    - GO – devine interacționabil doar dacă sunt selectate atât un element din prima lista cât și un element din a doua, iar prin apăsarea acestuia se setează limbile între care se va realiza traducerea, aceasta începând imediat
    - CANCEL – anulează modificarea selecției limbilor de translație realizate în acea sesiune a ferestrei
    - NONE – setează translatorul pentru a nu traduce textele, deselectand limbile selectate în acel moment

Primă dată când o limba este folosită pentru traducere este necasar accesul la internet pentru a fi descărcat modelul responsabil cu aceasta, apoi el va fi stocat în dispozitiv fără a mai fi nevoie de acces la rețea pentru viitoare utilizări. Dacă însă accesul la internet nu este oferit în acel moment aplicația va oferi un mesaj de eroare cu textul “Could not download language model!”.

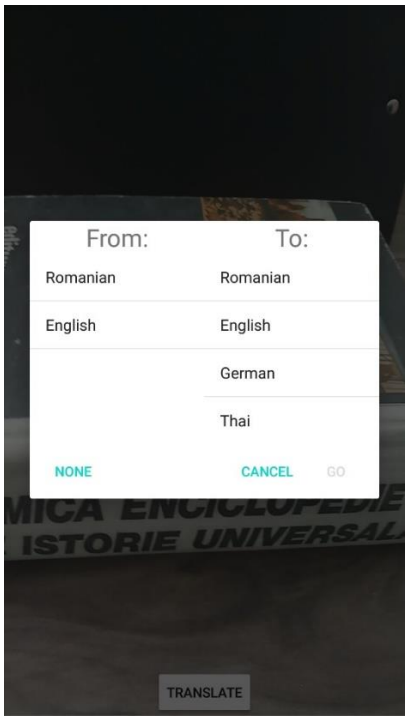

Figura 17 Interfața de selecție a limbilor

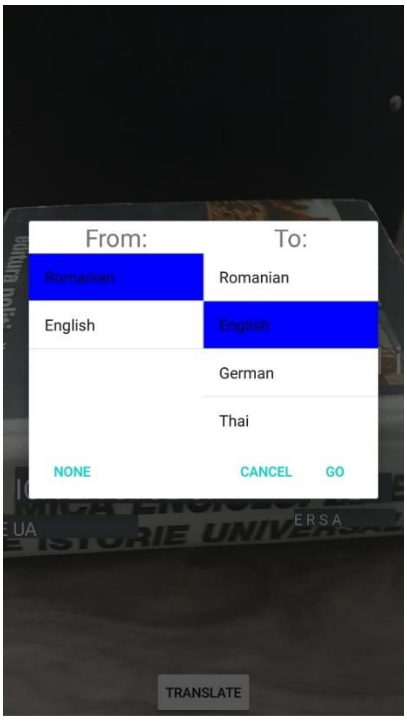

Figura 18 Selecția limbilor

Când există o pereche de limbi selectate pentru traducere toate textele din casete vor fi afișate direct in forma tradusă, dacă traducerea a avut succes, altfel în forma inițială.

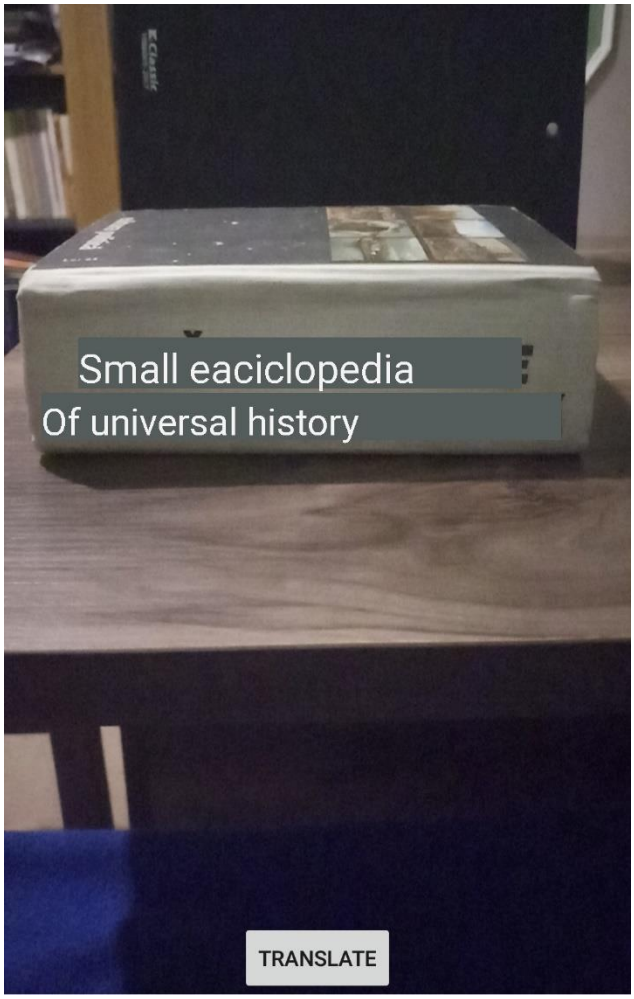

Figura 19 Traducerea textului

Concluzii

Aplicația concepută în această lucrare are ca scop aducerea într-un context ușor de utilzat și foarte accesibil a unei tehnologii care poate avea multe aplicații practice în viața de zi cu zi. Simplitatea interfeței și a accesibilitatea ei pe dispozitive mobile Android o recomandă pentru un cadru larg de aplicații practice,în special într-o societate care este continu în mișcare interacționând constant cu diferite culturi cu limbi necunoscute oricărui utilizator.

În implementarea aplicației am întâmpinat o gama variată de probleme variind de la proble de optimizare al timpului de rulare, până la probleme de implementare al codului. Cum aplicația rulează în contextul dinamic al reprezentării pe video în timp real unul din conceptele ce a trebuit luat în considerare pe întreaga durată a realizării acesteia a fost balansarea vitezei de rulare cu acuratețea de identificare a textelor. Pentru aceasta am ales diferite funcții în preprocesarea imaginii și am încercat realizarea unei rețele neuronale care să ofere rezultate în timp cât mai scurt, păstrând totuși o acuratețe cât mai bună.

Un motiv important pentru alegerea temei și metodei alese de rezolvare a problemei a fost însușirea tehnologiilor folosite și o mai bună înțelegere a domeniului intelegenței artificiale aplicate în practică. Consider astfel că am învățat o multitudine de lucruri prin realizarea aplicației ce mă vor ajuta pe viitor în cariera de programator.

Bibliografie

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., . . . Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Retrieved from https://www.tensorflow.org/

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Chollet, F., & others. (2015). Keras. *Keras*.

De Campos, T. E., Babu, B. R., Varma, M., & others. (2009). Character recognition in natural images. *VISAPP (2), 7*.

de Campos, T. E., Bodla, R. B., & Varma, M. (2009). The chars74k dataset. *The chars74k dataset*.

Jegou, H., Douze, M., & Schmid, C. (2008). INRIA Holidays dataset. *INRIA Holidays dataset*. Oct.

Liu, Y., Goto, S., & Ikenaga, T. (2006). A contour-based robust algorithm for text detection in color images. *IEICE transactions on information and systems, 89*, 1221–1230.

Malakhova, E., & Shelepin, Y. (2016). Modeling Convolutional neural networks for text detection task. *Video and Audio Signal Processing in the Context of Neurotechnologies. IEEE*, 16.

Neumann, L., & Matas, J. (2013). Scene text localization and recognition with oriented stroke detection. *Proceedings of the ieee international conference on computer vision*, (pp. 97–104).

Shi, C., Wang, C., Xiao, B., Zhang, Y., Gao, S., & Zhang, Z. (2013). Scene text recognition using part-based tree-structured character detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 2961–2968).

Suzuki, S., & others. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing, 30*, 32–46.

Yao, C., Bai, X., & Liu, W. (2014). A unified framework for multioriented text detection and recognition. *IEEE Transactions on Image Processing, 23*, 4737–4749.

Yao, C., Bai, X., Shi, B., & Liu, W. (2014). Strokelets: A learned multi-scale representation for scene text recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 4042–4049).

Zhu, Y., Yao, C., & Bai, X. (2016). Scene text detection and recognition: Recent advances and future trends. *Frontiers of Computer Science, 10*, 19–36.

Link-uri

Zipper: https://ezipper.ro/optical-character-recognition/

Docparser: https://docparser.com/blog/what-is-ocr/

Python: https://www.python.org/doc/essays/blurb/

Java:https://java.com/en/download/faq/whatis_java.xml#:~:text=Java%20is%20a%20programming%20language,fast%2C%20secure%2C%20and%20reliable.

Tutsplus: https://code.tutsplus.com/tutorials/learn-java-for-android-development-introduction-to-java--mobile-2604#:~:text=Android%20applications%20are%20developed%20using,(now%20owned%20by%20Oracle).&text=These%20libraries%20exist%20to%20help%20developers%20build%20applications.

OpenCV: https://opencv.org/about/#:~:text=About,perception%20in%20the%20commercial%20products.

Keras: https://keras.io/about/

Infoworld: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

Tutsplus: https://code.tutsplus.com/tutorials/getting-started-with-firebase-ml-kit-for-android--cms-31305

OpenCV: https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html

Sciencedirect: https://www.sciencedirect.com/topics/engineering/gaussian-blur

OpenCV: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html