UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Detectarea malariei folosind învățarea automată

propusă de

Corduneanu Florian Mihai

Sesiunea: februarie, 2020

Coordonator științific

Lect. dr. Ignat Anca

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Detectarea malariei folosind învățarea automată

Corduneanu Florian Mihai

Sesiunea: februarie, 2020

Coordonator științific

Lect. dr. Ignat Anca

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemntatul(a) ……..……

domiciliul în ……..

născut(ă) la data de ……..…....., identificat prin CNP ……..……..., absolvent(a) al(a) Universității „Alexandru Ioan Cuza" din Iași, Facultatea de ……. specializarea ……, promoția ……., declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul:

_____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, ……..…....….. Semnătură student …..……...……

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de accord ca Lucrarea de licență cu titlul „ *Detecția malariei prin analiza imaginilor*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de accord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent *Florian Mihai Corduneanu*

_____

Cuprins

Introducere

Motivaţie

Malaria este o boală gravă, care este răspândită pe scară largă în Africa sub-sahariană și provoacă aproximativ 400.000 de decese pe an, majoritatea fiind sugari și persoane în vârstă, iar depistarea ei este foarte costisitoare din punctul de vedere al timpului deorece trebuie analizată fiecare celulă de sange în parte, dar și a personalului deorece diagonosticul poate fi dat greșit în funcție de experiența acestora.

Problema principală este însă prescrierea medicamentului fără diagnostic clinic pozitiv din cauza lipsei de timp. Această abordare are ca rezultat rezistența parazitului împotriva medicamentelor anti-malariene.

Prin urmare, industria medicală are nevoie de o metodă care detectează automat procentul de celule roșii infectate dintr-un frotiu de sânge printr-un sistem care necesită mai puțin timp pentru a îndeplini această sarcină și este, de asemenea, precis, în ciuda experienței operatorului.

Obiective

Obiectivul acestui proiect a fost să producă un sistem care să poată segmenta fiecare globulă roșie dintr-un frotiu de sânge în celule individuale și folosind clasificatorul instruit pentru a clasifica cu exactitate ce celule sunt infectate și care nu și să calculeze procentul de celule infectate. pentru frotiul de sânge respectiv. Domeniul de aplicare al acestui proiect este de a ajuta medicii din țările în care malaria este larg răspândită pentru a da un diagnostic corect pentru această boală, prin urmare, pentru a reduce numărul de victime cât mai mult posibil.

Descrierea sumară a soluţiei

Acest proiect va consta într-o metodă de diagnostic a bolii Malariei care folosește tehnici automate precum învățarea automată, o aplicație de inteligență artificială care oferă sistemelor capacitatea de a învăța și îmbunătăți automat din experiență, de a crea un software care este capabil să detecteze dacă imaginea unei celule roșii, cum ar fi cea din figura 1 este fie infectată sau nu [8]. Programul creat va avea, de asemenea, opțiunea de a segmenta automat celulele individual de la o imagine a unui frotiu de sânge, folosind tehnici de procesare a imaginilor, astfel încât utilizatorul să nu fie nevoit să facă manual această sarcină, ceea ce ar necesita mai mult timp.

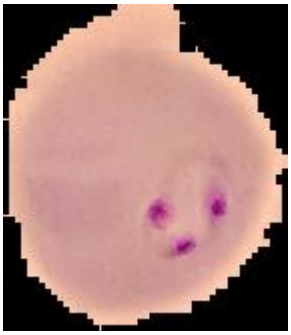

Fig. 1 Imagine a globulelor roșii infectate

Structura lucrării

Lucrarea am împarțit-o în patru capitole capitole:

- Malaria și necesitatea idetificării ei
- Implementarea
- Planificarea proiectului

Primul capitol ofer ă o descriere detaliata a malariei și probleme ridicate de aceasta.

Urmatorul capitol urmarește procesul de dezvoltare al aplicației și tehnicile folosite pentru obținerea unui rezultat cât mai satisfăcător cât si problemele întâlnite.

Ultimul capitol prezintă modul în care am abordat această problema și organizarea sarcinilor.

1. Contribuţii

Pentru realizarea acestei lucrări am învățat domeniul procesării de imagini și a învățării automate, domenii relativ noi pentru mine.

Am avut ocazia să invat diferite metode de segmentare, avantajele si dezavantajele lor cât și moduri diferite de a construi un clasificator.

În final, am adus toate funcționalitățile sub o singură interfață, construită cu ajutorul librăriei Tkinter.

2. Malaria și necesitatea idetificării ei

O boală este o tulburare de structură sau funcție într-un organism viu care produce simptome specifice sau care afectează o anumită locație și nu este rezultatul direct al unei leziuni fizice. Lumea caută în mod constant leacuri pentru tot felul de boli, dar multe dintre ele sunt încă necunoscute populației sau nu au încă niciun tratament. Pentru oameni, bolile pot provoca durere, suferință, disfuncție sau moarte persoanei afectate sau probleme similare celor care sunt în contact cu persoana respectivă. De asemenea, bolile pot fi clasificate ca autoimune, bacteriene, cancer, sânge, inimă, digestive, nervoase, tiroidiene sau cu transmitere sexuală. Datorită varietății lor și cât de ușor de răspândit, acestea sunt principalul motiv pentru creșterea mortalității mondiale. Tratamentele pentru majoritatea bolilor variază de la medicamente și dispozitive medicale până la operații sau până la îngrijirea de la sine. Unele boli pot fi vindecate pur și simplu în timp, în timp ce altele nu pot fi vindecate complet, dar simptomele acestora pot fi tratate folosind terapia de gestionare a durerii sau îngrijirea paliativă.

Mai multe boli pot fi transmise oamenilor de către insecte și sunt cauzate de agenți infecțioși, cum ar fi viruși, bacterii sau paraziți. Una dintre principalele boli care afectează aproximativ 450 de milioane de oameni este Malaria [1]. Este cauzată de microorganisme unicelulare, numite protozoare, ale parazitului Plasmodium și este răspândită cel mai frecvent de către un țânțar Anopheles feminin infectat. Parazitul este localizat în saliva țânțarului și infecția are loc atunci când mușcă și introduce parazitul în sângele persoanei. Apoi, paraziții călătoresc către ficat, unde se maturizează și se reproduc[10]. Există cinci tipuri de febră malară: P. knowlesi care rareori provoacă boală la om, P. vivax, P. ovale și P. malariae, care provoacă o formă mai ușoară de malarie și, în sfârșit, P. falciparum, care provoacă majoritatea deceselor [2]. Cele mai frecvente simptome care sunt asociate în mod obișnuit cu malaria includ febră, oboseală, vărsături și dureri de cap, în timp ce în cazuri grave poate provoca pielea galbenă, convulsii, comă sau moarte.

Toți paraziții malari au un ciclu de viață care poate fi împărțit în patru etape. Atunci când sunt transportate de țânțar, paraziții sunt într-o formă infecțioasă motilă, care se numește sporozoit. De îndată ce paraziții ajung la celulele hepatice, acesta începe să se reproducă în mod asexual, producând merozoite care este a doua etapă. Unele merozoite infectează noi globule roșii și încep un ciclu de înmulțire care produce între 8 și 24 de merozoite noi, în timp ce altele se transformă în gametocite imature, aceasta fiind a treia etapă. Atunci când țânțarul feminin mușcă o persoană infectată, unele gametocite sunt luate cu sângele și se maturizează în intestinul țânțarului unde se contopește și formează un zigot motil. În această

etapă finală, zigoții se transformă în noi sporozoiți care călătoresc către glandele salivare ale țânțarului, făcând țânțarul gata să infecteze din nou [3].

Malaria a fost descoperită pentru prima dată în 1880 de Alphonse Laveran [4]. De atunci, oamenii de știință au muncit din greu pentru a găsi modalități de diagnosticare a acestuia în etapele sale timpurii. Diagnosticul Malariei presupune două faze: determinarea procentului de celule sanguine infectate și identificarea speciilor și a etapelor ciclului lor de viață [5]. Cea mai comună abordare este să se preia un frotiu de sânge (figura 2) și urmărit la un microscop pentru a identifica anumite trăsături specifice ale celulelor sanguine infectate, ceea ce duce la un diagnostic precis. Dar această metodă consumă mult timp și necesită o muncă intensă pentru un operator uman, prin urmare, în țări precum Asia, America Latină sau Africa Sub-Sahariană, unde Malaria este foarte largă, medicii tind să prescrie medicamente anti-malariene fără a diagnostica cu exactitate o persoană [6]. Recent, au fost dezvoltate noi metode de diagnosticare cum ar fi detectarea rapidă a antigenului sau reacția în lanț a polimerazei, care se bazează pe detectarea secvențelor specifice de acid nucleic [7]. Dar abordarea principală rămâne diagnosticul microscopic, care prezintă avantajele că este cea mai eficientă și mai fiabilă metodă de diagnostic, foarte sensibilă, oferind posibilitatea diferențierii dintre specii și pot fi observate și etapele asexuale ale parazitului. În ciuda avantajelor sale, această abordare are de asemenea dezavantaje, inclusiv costul ridicat de achiziție și menținere a noii generații de microscopuri, precum și precizia acestei metode fiind puternic influențată de experiența tehnicianului.

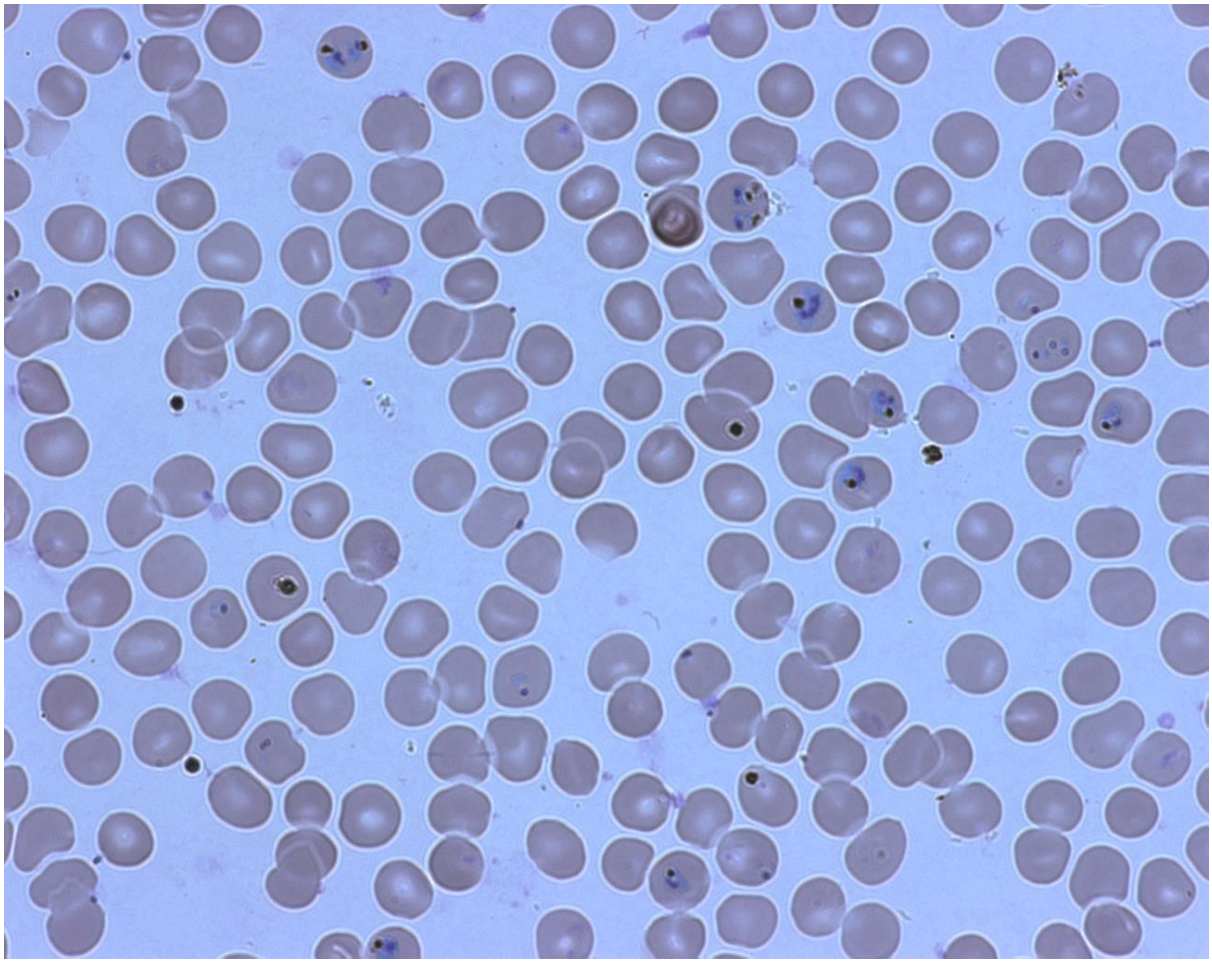

Fig. 2 Frotiu de sânge ce conține celule infectate

3. Implementare

3.1 Segmentatorul

Segmentarea imaginii reprezintă procesul de împărțire a unei imagini în grupuri de pixeli pe baza unor criterii specifice. Rezultatul segmentării imaginii este un set de segmente care acoperă colectiv întreaga imagine sau un set de contururi extrase din imagine. Ultimul rezultat este numit și detectarea marginilor. Detectarea marginilor este o tehnică de procesare a imaginilor pentru găsirea limitelor obiectelor din imagini. Funcționează prin detectarea discontinuităților în luminozitate.

Pentru prima fază a proiectului, scopul a fost să utilizeze tehnicile de procesare a imaginii și unul dintre cei mai comuni algoritmi de detecție a marginilor, pentru a crea un software care să primească ca intrare o imagine a unui micrograf care conține mai multe celule de sânge roșu celule și returneaza fiecare celulă din imaginea respectivă, segmentate individual, ca imagini. Această fază a proiectului a fost esențială pentru crearea unei aplicații viabile. Pentru dezvoltarea acestui software, mi s-au oferit un set de 30 de micrografii de un cercetător la University College London. Un exemplu de micrografie poate fi văzut în figura 2.

3.1.1 Utilizarea algoritmului watershed

O primă încercare de dezvoltare a acestei proceduri a fost făcută prin utilizarea mai multor algoritmi de thresholding, distance transform operator, împreună cu algoritmul watershed. Thresholding-ul este cea mai ușoară metodă de segmentare a imaginii și poate fi utilizat pentru a crea o imagine binară dintr-o imagine în tonuri de gri. Distance transform operator poate fi aplicat numai imaginilor binare. Algoritmul watershed este o transformare definită pe o imagine la scară gri. Tratează imaginea ca pe o hartă topografică, cu luminozitatea fiecărui punct reprezentând înălțimea ei și găsește liniile care se întind de-a lungul vârfurilor crestelor.

Aruncând o privire atentă asupra tuturor micrografiilor, am înțeles imediat că toate sunt de culoare și pentru a folosi un algoritm de thresholding pe ele a trebuit să le transform în imagini la scară gri. Acest lucru a fost realizat folosind una dintre cele mai comune funcții implementate în biblioteca OpenCV, care se numește cvtColor (input_image, flag) și transformă o imagine dintr-un spațiu color într-altul. Pentru o conversie de la culoare la imagine în tonuri de gri, trebuie utilizat indicatorul cv2.COLOR_BGR2GRAY.

După ce imaginea a fost convertită în scară de gri, următorul pas a fost utilizarea unui algoritm de thresholding pentru a găsi o estimare aproximativă a celulelor. Pentru aceasta am folosit algoritmul de binarizare al lui Otsu, o altă funcție implementată în biblioteca OpenCV, care calculează automat o valoare de thresholding din histograma imaginii pentru o imagine bimodală. Am ales să folosesc acest algoritm, deoarece pentru o segmentare bună trebuie să existe o valoare de thresholding bună. Funcția are forma ret2, th2 = cv2.threshold (img, 0,255, cv.THRESH_BINARY + cv.THRESH_OTSU), iar variabila th2 conține imaginea cu thresholding-ul aplicat. După aplicarea acestor funcții pe figura 2, imaginea rezultată poate fi văzută în figura 3.

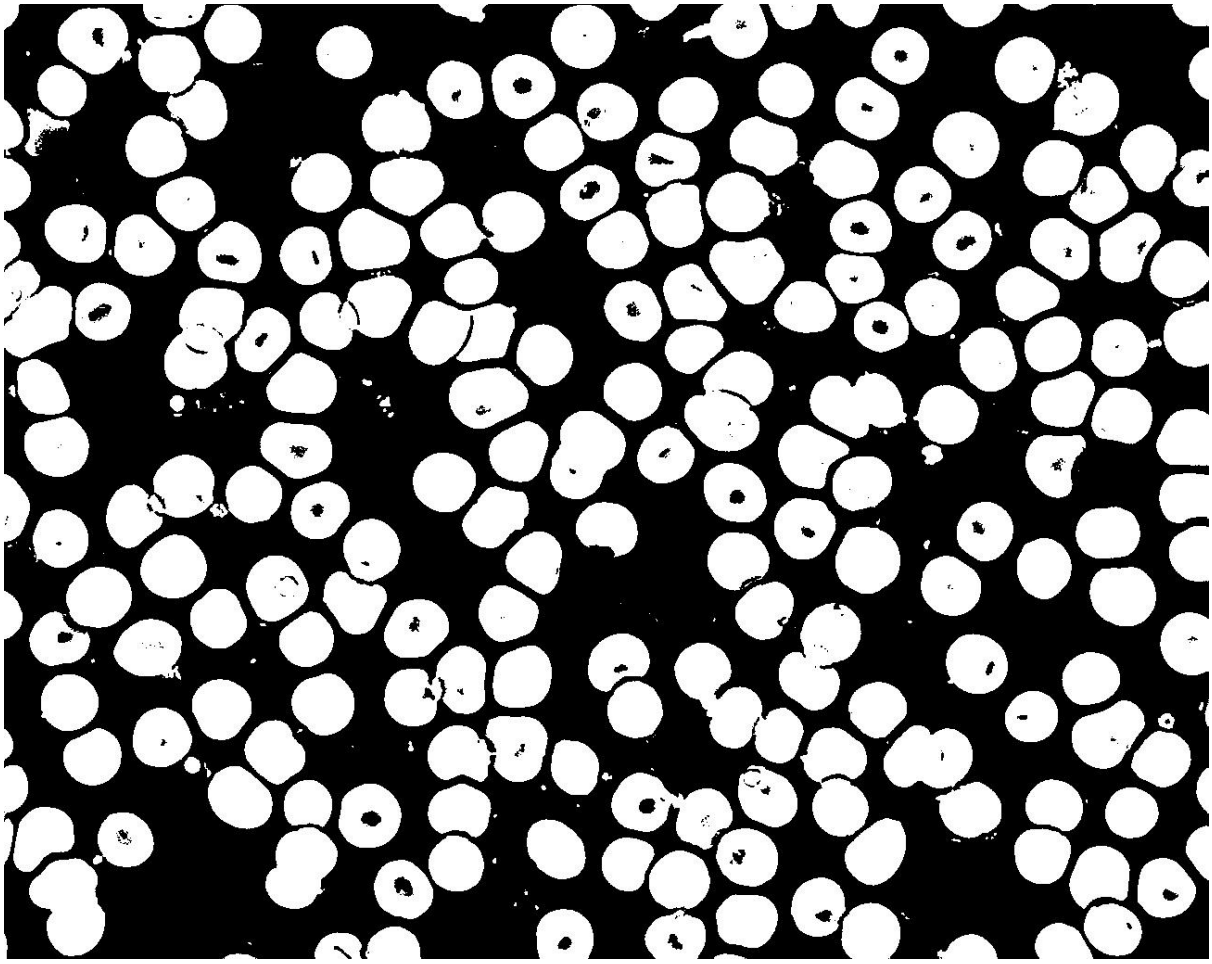

Fig. 3 Imaginea rezultată

După cum putem vedea din figura de mai sus, în fundal există câteva mici pete albe. Acestea se numesc zgomote și trebuie eliminate din fundal pentru a merge mai departe. O modalitate bună care este folosită în mod obișnuit în îndepărtarea zgomotelor de fundal reprezintă operația morfologică numită opening. Această operațiune este folosită ca argument al funcției cv2.morphologyEx (thresholded_image, type_of_operation, kernel) care este utilizată pentru a efectua transformarea morfologică. Un kernel este o matrice mică care este folosită pentru mai multe sarcini, precum estomparea, ascuțirea și chiar detectarea marginilor.

Operația de opening constă practic dintr-o altă operație care se numește erosion, care este apoi urmată de dilation. Operația de erosion erodează limitele obiectului prim-plan, făcând ca grosimea sau dimensiunea obiectului prim-plan să fie diminuate. Operația de dilation este opusă eroziunii.

De asemenea, fiecare gaură neagră din interiorul celulelor trebuie îndepărtată. O altă operație morfologică poate fi folosită pentru a face acest lucru și se numește closing. Closing--ul este inversul deschiderii; prin urmare, constă într-o dilatație urmată de o eroziune. Această funcție utilizează, de asemenea, un kernel care parcurge imaginea.

După aplicarea acestor două operații morfologice în ordinea în care au fost explicate în această lucrare, imaginea rezultată poate fi văzută în figura 4.

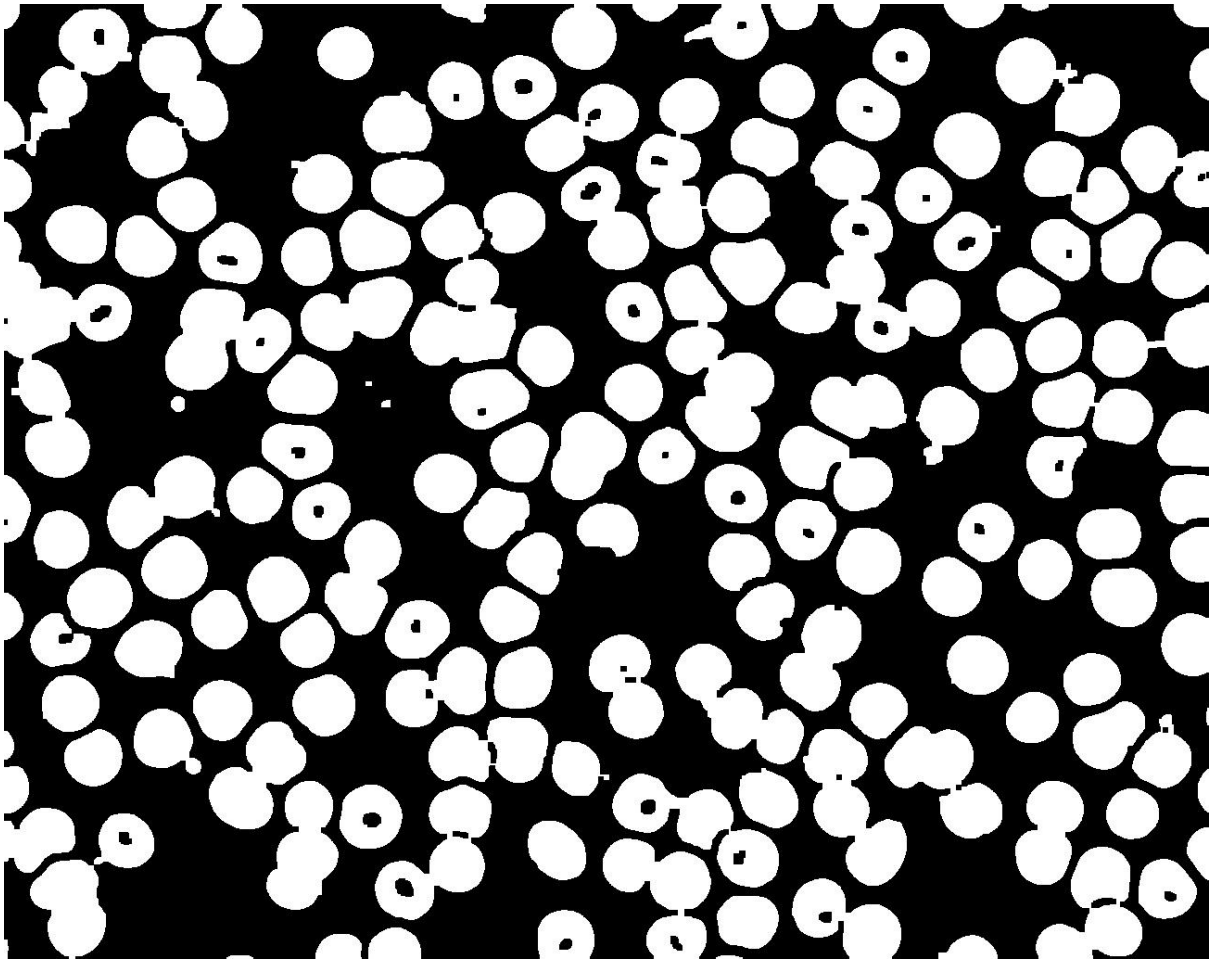

Fig. 4 Imaginea cu zgomotul redus

O altă problemă interesantă a fost aceea că, după cum putem vedea din figura 8, există celule care se ating între ele și, prin urmare, găsirea unei transformări la distanță și aplicarea unui threshold folosind-o pentru extragerea zonei prim-plan ar putea da rezultate greșite în acest moment. Pentru a scăpa pe moment de atingerea celulelor, am aflat că mai întâi pot dilata imaginea, ceea ce va face celulele mai mari și vor extrage cea mai mare parte a zonei de fundal și după ce am aplicat un threshold pentru extragerea zonei prim-planului. Operația de transformare la distanță a fost implementată în biblioteca OpenCV sub forma funcției cv2.distanceTransform (image, distance_type, mask size). Argumentul distance_type al acestei funcții poate fi cv2.DIST_L2 sau cv2.DIST_L1 sau cv2.DIST_C. Al treilea argument al acestei funcții reprezintă dimensiunea măștii de transformare la distanță și poate fi un număr sau cv2.DIST_MASK_PRECISE. Pentru acest proiect, am ales ca tipul de distanță să fie cv2.DIST_L2 și dimensiunea măștii egală cu 5. După aplicarea operației de dilatare și găsirea transformării distanței, puteam extrage zona prim-plan aplicând o altă operație de threshold având ca argument imaginea rezultată după operațiunea de transformare la distanță, o valoare de threshold bazată pe cea mai mare valoare de pixel a imaginii rezultate și un tip de threshold 0. Zona rămasă după scăderea zonei prim-planului din zona de fundal, rămâne încă necunoscută.

Zona necunoscută poate avea fie o parte din fundal, fie prim-plan. Acestea din urmă pot avea părți din celule separate sau din cele cu atingere. Următorul pas al procesului de segmentare este utilizarea unui marker pentru etichetarea diferită a fundalului, prim-planului

și a zonelor necunoscute. Pentru aceasta, am folosit o altă funcție implementată în biblioteca OpenCV numită cv2.connectedComponents (image) care returnează o valoare care constă din numărul total de etichete în care 0 reprezintă zona de fundal.

Această funcție este aplicată pe zona prim-plan pentru a eticheta zona de fundal cu 0 și restul obiectelor cu numere întregi diferite. Dar algoritmul watershed consideră zona marcată cu 0 ca zonă necunoscută, de aceea a trebuit să adaug 1 la toate etichetele pentru a mă asigura că întreaga suprafață de fundal devine marcată cu 1 și apoi pentru a marca regiunea necunoscută cu 0 astfel algoritmul watershed poate afla din ce zonă aparține. Algoritmul este implementat sub funcția cv2.watershed (image, markers) care are ca argument imaginea originală și markerii. După aplicarea acestei funcții, regiunea de margine a fiecărei celule trebuie marcată cu -1. Pentru o vizualizare mai bună am schimbat și culoarea regiunilor de delimitare. Figura 5 reprezintă rezultatul fazei de segmentare:

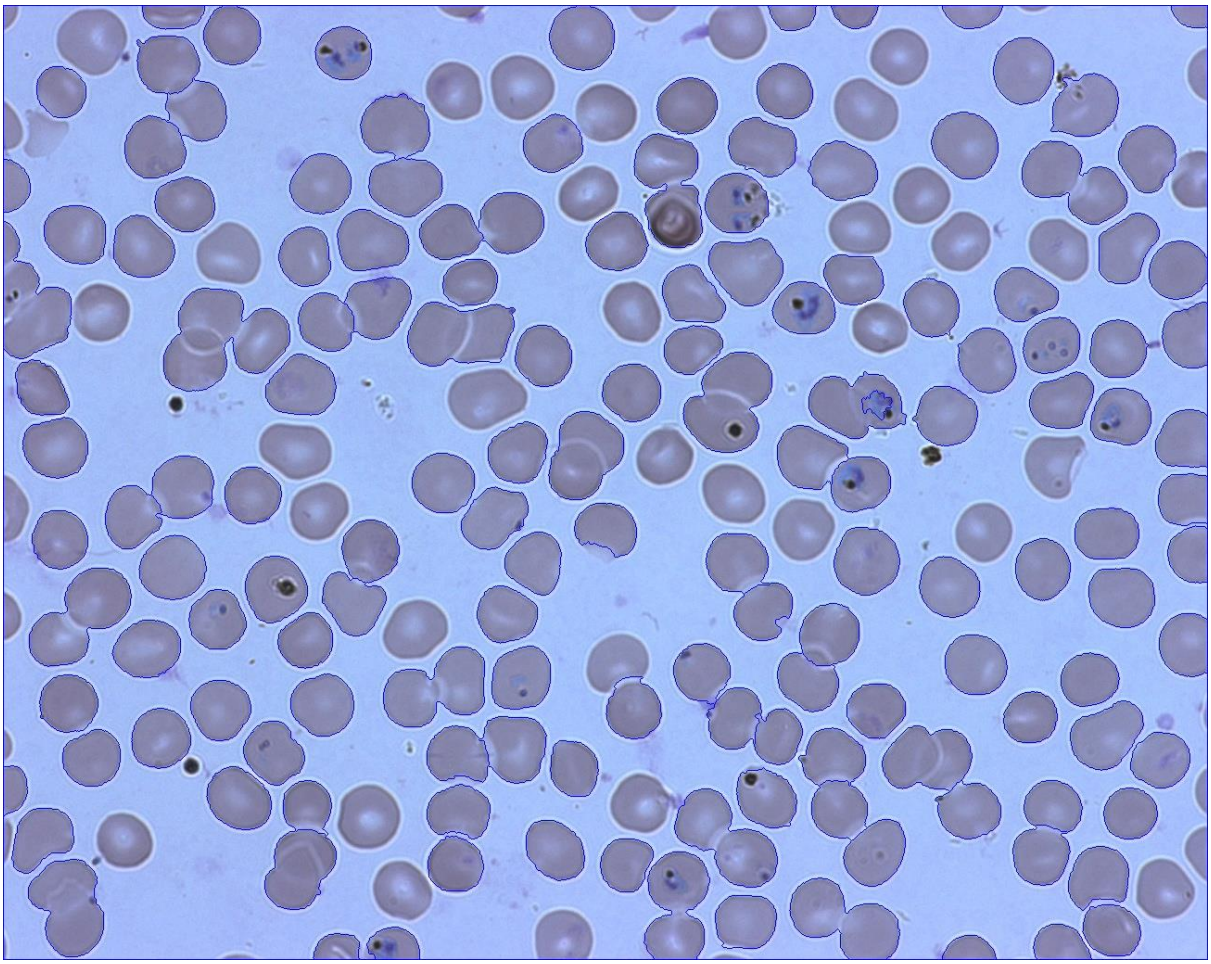

Fig. 5 Imaginea segmentata

Din imaginea rezultată am observat că pentru unele celule nu există deloc margini detectate, iar pentru unele altele, marginile detectate nu se suprapun cu marginile celulelor din imagine. Am rulat teste pe alte imagini, iar rezultatul a fost același. Din cauza acestui rezultat slab, am decis că această metodă nu este satisfăcătoare pentru proiect și am început să caut un alt mod de segmentare a celulelor mai precis.

3.1.2 Utilizarea algoritmului Canny

Cea de-a doua metodă pe care am implementat-o pentru această fază a proiectului se bazează pe detectorul de margini canny, operațiile morfologice utilizate și explicate în secțiunea 6.2.1 și funcția implementată în biblioteca OpenCV numită cv2.findContours (binary_image, contour_retrieval_mode, contur_approximation_mode) care preia contururile imaginii binare date ca argument. Argumentul contour_retrieval_mode poate fi fie cv2.RETR_EXTERNAL care recuperează doar contururile exterioare extreme, cv2.RETR_LIST care recuperează toate contururile fără a stabili relații ierarhice, cv2.RETR_CCOMP care recuperează toate contururile și le organizează într-o ierarhie la două niveluri, cv2.RETR_TREE care recuperează toate contururile și reconstruiește o ierarhie completă a contururilor. Argumentul contour_approximation_mode poate fi fie cv2.CHAIN_APPROX_NONE care stochează absolut toate punctele de contur sau cv2.CHAIN_APPROX_SIMPLE care comprima vecinii orizontali, verticali și diagonali.

Detectorul de margini canny este un operator de detectare a marginilor care utilizează un algoritm în mai multe etape pentru a detecta o gamă largă de margini dintr-o imagine. Operatorul are forma de cv2.Canny (image, first_treshold, second_treshold). Prima valoare de threshold este utilizată pentru legarea muchiilor și a doua pentru a găsi segmente inițiale ale muchiilor puternice.

Primul pas în dezvoltarea acestei abordări a fost transformarea imaginii originale din RGB la scară de gri prin utilizarea funcției cv2.cvtColor (image, cv2.COLOR_BGR2GRAY). Pentru a obține o segmentare bună de la detectorul de margine la nivelul celulelor, trebuie să elimin mai întâi zgomotul din imagine. Pentru aceasta, am aplicat un filtru low-pass pe imaginea de gri, numită blur. Această funcție este de asemenea implementată în biblioteca OpenCV și are forma cv2.blur (target_image, kernel_size) și practic estompează imaginea țintă. După ce am efectuat câteva teste pe mai multe micrografii, am aflat că cea mai bună valoare pentru dimensiunea kernel-ului este 5 și Figura 6 reprezintă imaginea rezultată după ce a convertit-o în scară de gri și a aplicat estomparea.

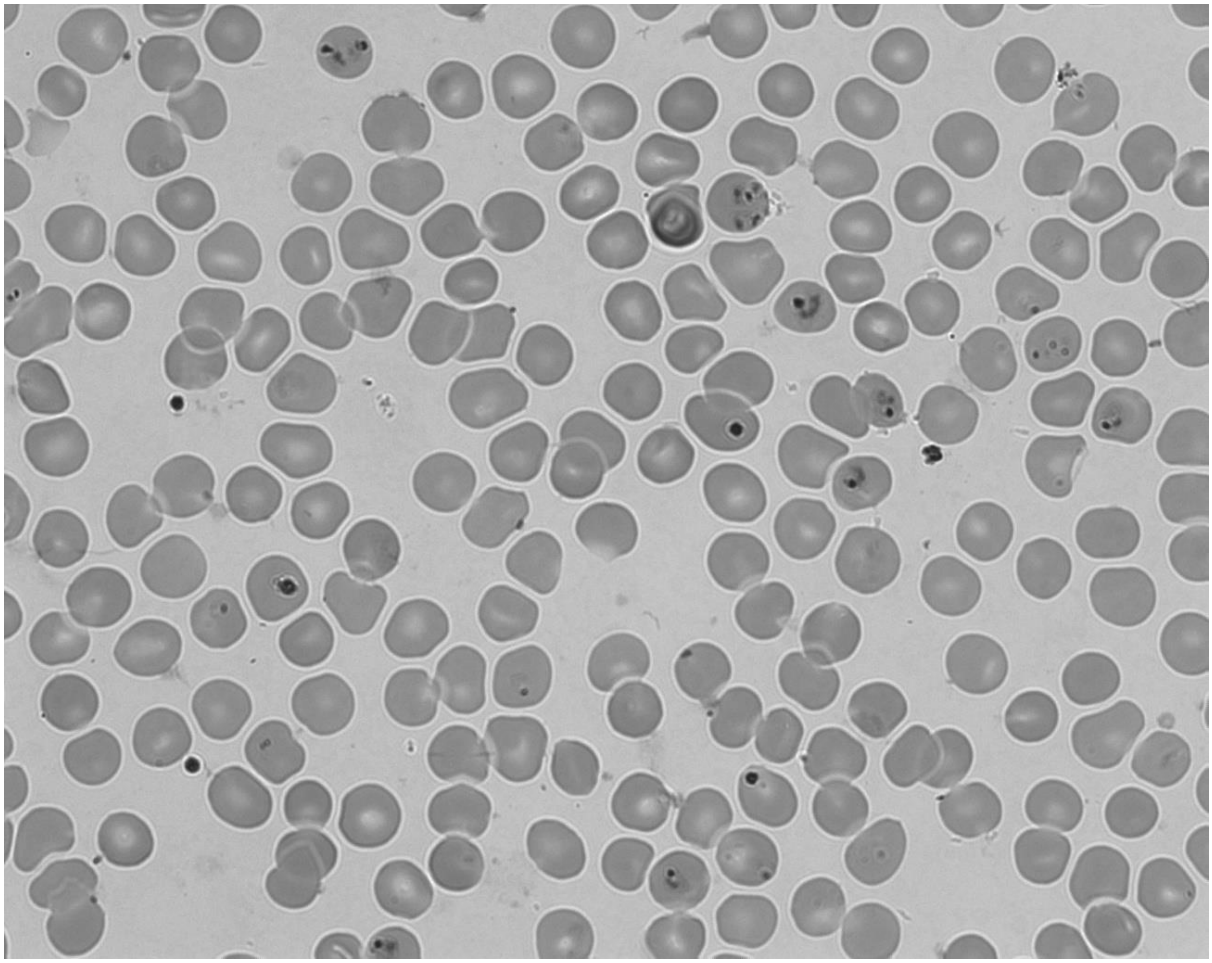

Fig. 6 Imaginea gri estompata

Stabilirea unor valori bune ale threshold-ului pentru detectorul de margini este importantă în găsirea tuturor marginilor celulelor, dar acest lucru depinde cu adevărat de nivelul luminii, de nuanțele de culoare și de zgomotul rămas. O bună abordare a furnizării în mod automat a valorilor de threshold adecvate este aplicarea unei operații de threshold folosind algoritmul de binarizare al Otsu, care ar întoarce a doua valoare de threshold a detectorului de margini și se va seta prima valoare a pragului ca jumătate din cel de-al doilea. După ce am implementat acești pași în cod, l-am testat și marginile rezultate ale micrografiei pot fi văzute în figura 7.

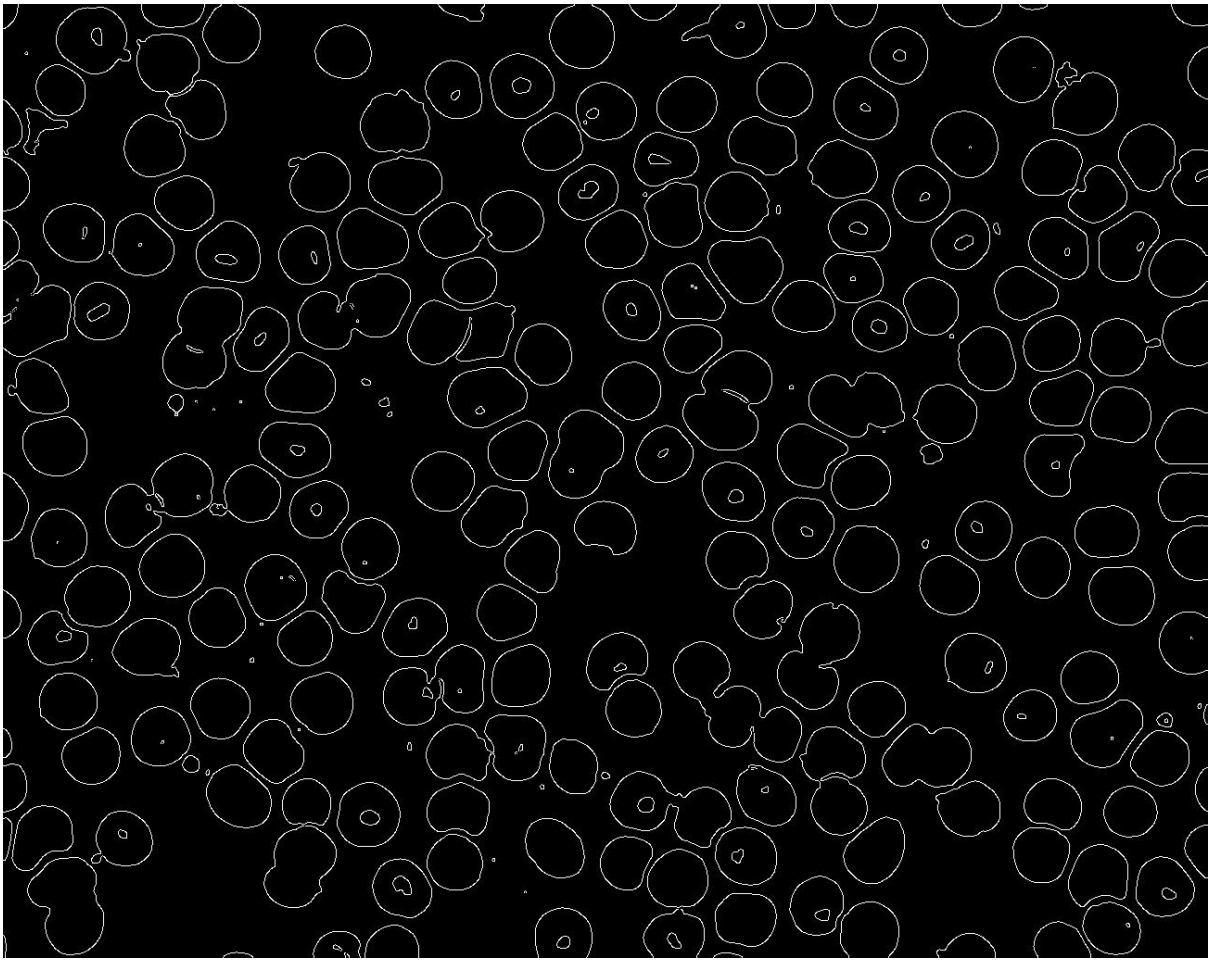

Fig. 7 Marginile detectate

După cum se poate observa din imaginea de mai sus, detectorul de margini Canny a făcut deja o recunoaștere a muchiei mai bună decât prima încercare care a folosit algoritmul watershed. Dar încă mai existau mici lacune în margini care trebuiau rezolvate pentru a trece la pasul următor. Acest lucru a fost realizat prin aplicarea operației morfologice de closing și folosirea unei dimensiuni de 9.

În acest moment, toate marginile au fost detectate, iar ultima etapă a fost aceea de a salva fiecare celulă ca imagine individuală. Un aspect important în legătură cu aceste micrografii este că imaginile originale au un fundal luminos, iar imaginile de antrenament care au fost utilizate pentru clasificator au un fundal negru. Prin urmare, a trebuit să găsesc o modalitate de a schimba culoarea de fundal a micrografiei în negru înainte de salvare pentru ca clasificatorul instruit să funcționeze așa cum era de așteptat la clasificarea imaginilor celulare individuale. În caz contrar, celulele segmentate nu ar avea aceleași caracteristici ca cele utilizate pentru antrenament și clasificatorul nu ar funcționa corect.

Abordarea pe care am găsit-o folosește funcția cv2.findContours având ca argumente marginile găsite de detectorul canny ca imagine țintă, cv2.RETR_EXTERNAL ca mod de regăsire a conturului și cv2.CHAIN_APPROX_SIMPLE ca metodă de aproximare a conturului. Apoi am creat o serie de zerouri având forma micrografului inițial, care reprezintă o imagine neagră de aceeași dimensiune cu cea a micrografiei. După aceasta, am folosit

funcția cv2.fillPoly (black_background, contours, white_colour) care desenează contururile cu culoare albă pe imaginea neagră. În cele din urmă, am folosit funcția cv2.bitwise_and (original_image, contoured_black_background) care umple contururile cu imaginea celulelor. Pentru o mai bună înțelegere a modului în care funcționează această abordare, Figura 8 arată imaginea rezultată după ce au fost aplicate operațiunile de mai sus.

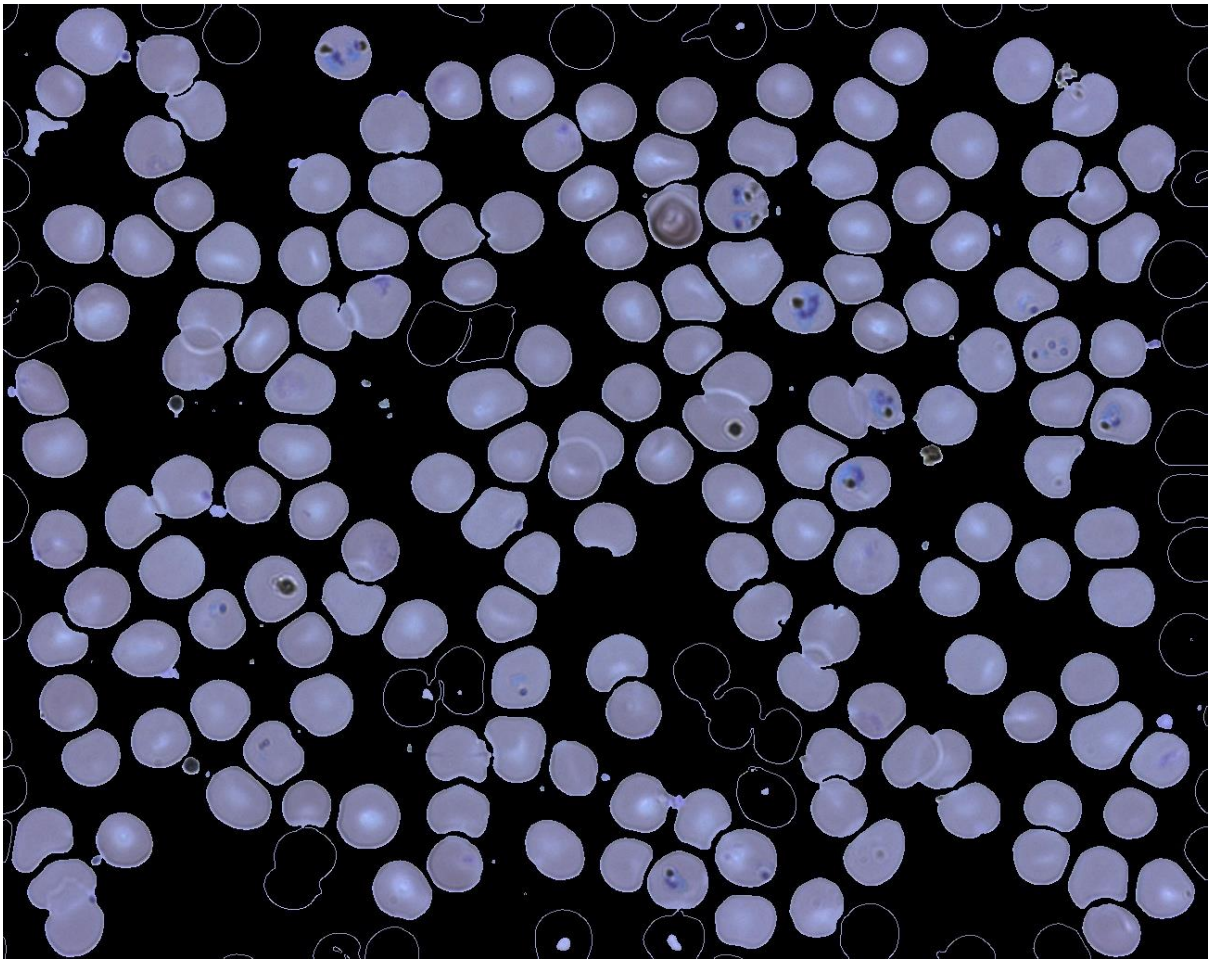

Fig. 8 Imaginea micrografică cu fundalul schimbată din alb la negru

În acest moment, ultima etapă a fost aceea de a salva fiecare celulă ca imagine individuală pentru finalizarea celei de-a doua faze a proiectului. Pentru acest lucru, am folosit funcția cv2.boundingRect (points). Folosind o buclă for care iterează prin toate contururile găsite, am stocat coordonatele x și y și lățimea și înălțimea dreptunghiului de delimitare pentru un contur și prin setarea unei valori de threshold de 50 de pixeli, astfel încât zgomotul rămas să nu să fie prins în imagini. Am creat o nouă imagine, preluând din imaginea rezultată din funcție bitwise_and numai din pixelii care erau în interiorul dreptunghiului de delimitare respectiv. În cele din urmă, segmentatorul creează, de asemenea, un director cu numele micrografului inițial, dacă nu există deja unul creat și salvează fiecare celulă ca imagine individuală în folder. Extensia imaginilor este .png și numele lor încep cu 0 pentru prima celulă și se termină cu numărul total de celule din micrograf.

3.1.3 Analizarea rezultatelor

Procentul de celule segmentate corect cu algoritmul Canny este de 80%, dar rămân celule a căror contur a fost detectat corect, însă nu au fost umplute cu imaginea celulei, unul dintre motive ar putea fi întreruperi în contur.

3.2 Clasificatorul de imagini

3.2.1 Antrenare și testare

Primul pas în dezvoltarea acestui proiect a constat în crearea și formarea unui clasificator de imagini prin utilizarea setului de date al imaginilor furnizate și a bibliotecii de învățare scikit-learn. Setul de date a fost produs de Institutul Național al Sănătății din SUA și constă din 27558 de probe de globule roșii individuale parazitate și neinfectate care au fost separate în două dosare diferite, prin urmare, jumătate din aceste probe aparțin unei clase și cealaltă jumătate a cealaltă clasă.

Citind documentația de pe site-ul scikit-learn, am descoperit că această bibliotecă folosește învățarea automată pentru a crea clasificatoare, iar clasificatorul pentru acest proiect trebuie instruit inițial folosind o parte din imaginile furnizate, care se numește set de instruire. Fiecare eșantion din setul de antrenament are mai multe caracteristici, iar clasificatorul trebuie să învețe acestea pentru a putea prezice cu exactitate din ce clasă aparține o imagine necunoscută.

În această bibliotecă, un estimator pentru clasificare este un obiect Python care implementează metoda fit (x, y), care este utilizată pentru procesul de instruire și prezice (T) care este utilizat pentru procesul de testare. Un exemplu de estimator este clasa sklearn.svm.SVC, care implementează support vector classification. SVM reprezintă mașini de suport-vector și sunt modele de învățare supravegheate care analizează datele utilizate pentru clasificare. O tehnică de învățare supravegheată este posibilă numai atunci când datele de instruire sunt etichetate. Un model SVM este o reprezentare a datelor de instruire ca puncte în spațiu, mapate astfel încât eșantioanele din categoriile separate să fie împărțite printr-un decal clar care să fie cât mai larg posibil. Un exemplu de mașină vector-suport poate fi văzut în figura 9.

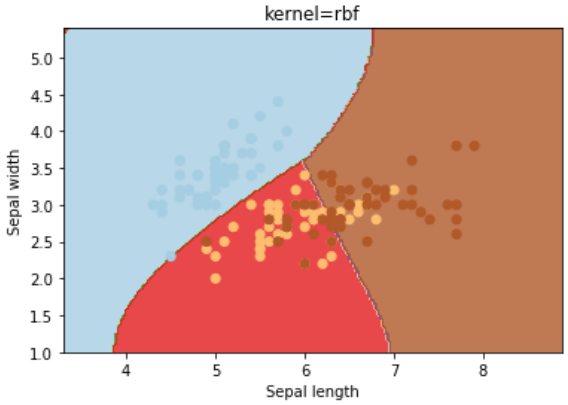

Fig. 9 Exemplu de SVC

Prin urmare, pentru a crea un prim clasificator pentru acest proiect, a trebuit să import inițial din biblioteca sklearn clasa svm, apoi să dau acestui clasificator un nume și să apelez constructorul estimatorului, având toate argumentele setate implicit. Pentru antrenament, a trebuit să apelez metoda fit (x, y), x fiind probele de instruire și y fiind etichetele lor. Următorul pas a constat în stocarea tuturor imaginilor și a etichetelor lor în doi vectori separați, astfel încât fazele de pregătire și testare să poată fi posibile.

Timpul de antrenare al unui clasificator depinde de cantitatea de date de instruire. Am ales pentru setul de antrenament 18000 de imagini, iar pentru testarea 9990. Figurile 10 și 11 arată un exemplu de celule infectate și neinfectate.

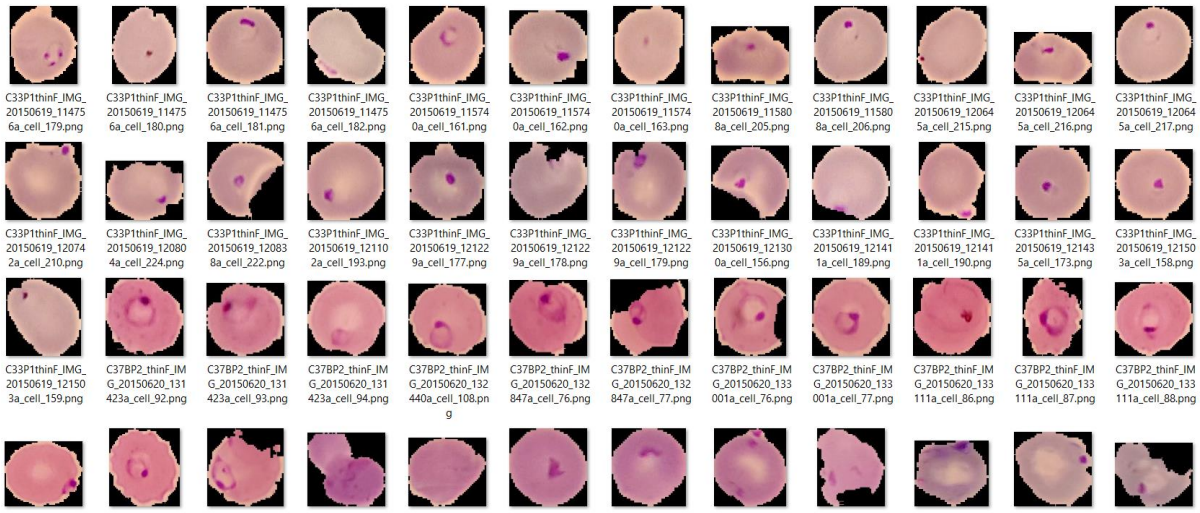

Fig. 10 Exemplu de celulă infectată folosită pentru antrenament

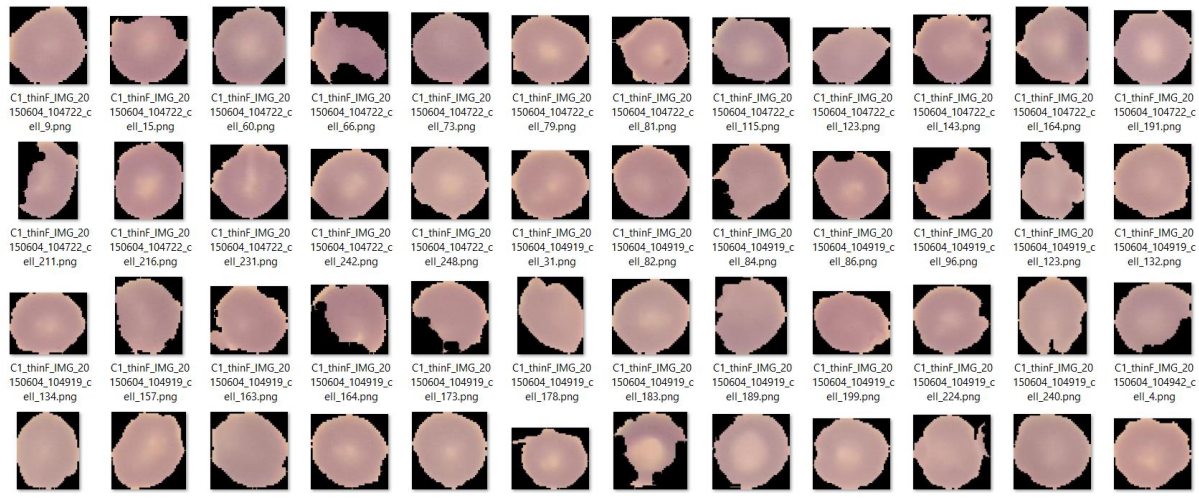

Fig. 11 Exemplu de celule neinfectate utilizate pentru antrenament

O problemă a imaginilor a fost aceea că au dimensiuni diferite. Prin urmare, în procesul de stocare a imaginilor a trebuit să redimensionez fiecare imagine la o dimensiune universală, pe care am ales-o să fie 100x100 pixeli. Pentru aceasta, am folosit metoda resize (imagine, imagine_shape) implementată în biblioteca OpenCV.

O altă problemă care ar fi consumat timp este aceea că, de fiecare dată când un nou clasificator este instruit, aș fi trebuit să citesc toate imaginile. Biblioteca NumPy a implementat o funcție numită save (nume de fișier, vector) care salvează un vector într-un fișier binar în format .npy, așa că am folosit această funcție pentru a salva datele de formare și testare, astfel încât atunci când am antrenat un clasificator, am putut încărca doar fișierele este necesară utilizarea unei alte funcții implementate în această bibliotecă numită load (nume de fișier).

Fiecare imagine de antrenament a fost stocată inițial într-un tablou cu patru dimensiuni, dar acest tablou a trebuit să fie redimensionat într-o dimensiune bidimensională pentru a putea fi folosit ca prim argument al funcției fit (). Prin urmare, am folosit o altă funcție implementată în biblioteca NumPy numită reshape (array, new_shape) care dă o nouă formă unui tablou fără a-și schimba datele. Am setat al doilea argument al acestei funcții la -1, deoarece acest lucru ar multiplica a doua, a treia și a patra dimensiune și a întoarce un tablou bidimensional.

3.2.2 Găsirea celui mai bun clasificator

În acest moment al dezvoltării proiectului am putut încărca mostrele de instruire și etichetele lor prin două fișiere diferite, iar următorul pas a fost să instruiesc un clasificator și apoi să îl testez pentru a obține exactitatea acestuia. Precizia unui clasificator este dată de o funcție implementată în biblioteca sklearn numită score (x_test, y_test) unde x_test reprezintă probele de testare, respectiv y_test etichetele lor.

Pentru a obține un clasificator precis, parametrii funcției svm.SVC () trebuie reglați în funcție de problema de învățare a acestui proiect. Făcând unele cercetări despre acești parametri, am descoperit că cei mai influenți sunt C, gamma și kernel-ul folosit. Parametrul C controlează balanța între limita de decizie și clasificarea corectă a punctelor de antrenament. Practic, într-un SVM căutăm un hiperplan cu cea mai mare marjă minimă și un hiperplan care separă corect cât mai multe puncte de antrenament. O valoare mică de C va determina optimizatorul să caute un hiperplan care separă marja mai mare, în timp ce o valoare mare de C va determina optimizatorul să aleagă un hiperplan cu marjă mai mică care să obțină toate punctele de antrenament clasificate corect. Parametrul gamma definește cât de departe ajunge influența unui singur punct de antrenament. Pentru valori scăzute ale gamma, fiecare punct are o influență îndepărtată și, în schimb, pentru valori mari ale gamma, fiecare punct are o influență apropiată. Acest lucru înseamnă că, de exemplu, pentru o valoare ridicată a gama, detaliile exacte ale graniței deciziei vor depinde doar de punctele de antrenament cele mai apropiate către aceasta și exact opusul pentru o valoare scăzută a gama.

Modelul trebuie salvat într-un fișier pentru a putea fi utilizat în clasificarea celulelor segmentate sau pentru calcularea exactității acestuia. Pentru a face acest lucru, am folosit biblioteca pickle care a implementat o funcție numită dump (obiect, fișier) care scrie o reprezentare a obiectului în fișier. Pentru încărcarea modelului instruit, am folosit o altă

funcție implementată în această bibliotecă care se numește load (fișier), care citește un șir din și îl interpretează ca un flux de date, reconstruind ierarhia obiectului original.

Așadar, pentru a găsi cel mai bun clasificator, am început să antrenez diferite modele prin utilizarea unor parametri diferiți și apoi să-l testez pe datele de testare pentru a calcula exactitatea acestora. Pentru fiecare proces de instruire și testare am calculat, de asemenea, timpul petrecut prin utilizarea funcției time () din biblioteca time și am păstrat înregistrările rezultatelor.

Primul clasificator pe care l-am instruit a avut toate argumentele funcției svm.SVC setate în mod implicit, deoarece în acel moment nu aveam prea multă experiență în alegerea parametrilor potriviți pentru problema de învățare a acestui proiect. În conformitate cu output-ul codului de instruire, acest proces a durat aproximativ o oră și treizeci de minute pentru a finaliza și după salvarea modelului într-un fișier am început procesul de testare pentru acesta, care a durat o oră și treisprezece minute și am dat un scor de precizie pentru acest model de 50%.

Deoarece acuratețea acestui prim clasificator nu a fost satisfăcătoare, am procedat la antrenarea altuia, dar prin setarea parametrilor C, gamma și kernel la 10, 1 și, respectiv, „rbf”. Procesul de instruire pentru acest clasificator a durat o oră și douăsprezece minute, iar procesul de testare a durat o oră și treisprezece minute și a dus la un scor de precizie de 53.3%.

Chiar dacă scorul de precizie al celui de-al doilea clasificator a fost puțin mai mare decât primul, totuși a fost nesatisfăcător pentru un clasificator care ar putea fi utilizat în industria medicală pentru a da un diagnostic precis. Acest lucru a însemnat că parametrii modelului pot fi reglați în continuare, așa că am instruit un al treilea clasificator cu parametrii C, gamma și kernel reglați de data aceasta la 1, 0.01 și, respectiv, „rbf”. Procesul de pregătire a durat o oră și cincisprezece minute, iar testarea a durat o oră și patruzeci și șase de secunde. Rezultatul procesului de testare a fost un scor de precizie de 51.3% pentru acest clasificator.

După cum se poate observa din rezultatele de până acum, nu a existat nicio îmbunătățire a preciziei, chiar dacă fiecare clasificator a avut valori diferite ale parametrilor. Aceasta m-a dus la ideea de a crește valoarea parametrului C, astfel încât optimizatorul să poată alege un hiperplan cu marjă mai mică și, prin urmare, să clasifice corect mai multe puncte de antrenament. Pentru a face acest lucru, am antrenat un al patrulea clasificator cu parametrii C, gamma și kernel stabiliți la 100, 0,01 și respectiv „rbf” și după finalizarea procesului de antrenament care a durat o oră și patruzeci și două de minute, am testat acest clasificator sperând că o sa obține o precizie mai bună. Dar rezultatul procesului de testare a arătat că acest clasificator are exact aceeași precizie ca și cei care au fost instruiți înainte și după ce am făcut unele cercetări, am aflat că există două posibile motive pentru această problemă: fie modelele suferă de overfitting sau parametrii clasificatorului nu sunt suficient de reglați.

3.2.3 Cross-validation

Cross-validation este o măsură preventivă puternică împotriva overfitting-ului. Ideea din spatele acesteia este de a utiliza datele de instruire pentru a genera mai multe mini-divize de testare și de a utiliza aceste diviziuni pentru a regla modelul [9].

Termenul overfitting se referă la un model care modelează prea bine datele de instruire. Overfitting-ul se întâmplă atunci când un model învață detaliile și zgomotul din datele de instruire, în măsura în care aceasta are un impact negativ asupra performanței modelului asupra datelor noi.

În acest proiect, am folosit Cross-validation pentru a verifica dacă modelele care au fost instruite până acum și au o precizie de cel mult 50% sunt overfitted. Pentru aceasta, am folosit o funcție implementată în biblioteca sklearn numită cross_val_score (model, x, y, splits). Această funcție ia ca argumente estimatorul la care trebuie să se desfășoare procesul de validare încrucișată, datele de instruire, etichetele asociate și numărul de diviziuni ale acestor date. Această funcție returnează o serie de scoruri ale estimatorului pentru fiecare execuție de validare încrucișată. Figura 6 ilustrează vizual modul în care datele sunt împărțite pentru validare încrucișată pe baza numărului de divizări date ca argument.

Fig. 1 Exemplu cross-validation folosind k diviziuni

Primul proces de cross-validation a fost realizat pe primul model instruit care a avut toți parametrii prestabili și o precizie de 50%. Caracteristicile și etichetele lor care au fost utilizate pentru acest proces sunt aceleași cu cele utilizate pentru antrenament, iar numărul de împărțiri a fost stabilit la 5. Întregul proces a durat o oră și douăzeci și două de minute, iar tabloul rezultat a fost [0,5. 0,498 0,5 0,5 0,498]. Un al doilea cross-validation a fost realizat pe al doilea model care a avut o precizie puțin mai bună (53%) prin utilizarea acelorași argumente de mai sus. Gama rezultată de scoruri de precizie a fost [0,5 0,5 0,498 0,5 0,498].

După cum putem observa din rezultatele celor două procese, nu există aproape nici o diferență de precizie între testarea modelelor față de întregul set de testare și cross-validation. Această variație scăzută indică faptul că modelele sunt destul de consistente și, prin urmare, nu sunt overfitted.

3.2.4 Reglarea parametrilor folosind GridSearchCV

După ce am fost sigur că modelele instruite până acum nu au fost overfitted, am început să caut o modalitate de a ajusta automat parametrii funcției SVC (). GridSearchCV este o funcție implementată și în biblioteca sklearn, care generează exhaustiv parametrii dintr-o grilă de valori specificate și prin potrivirea și evaluarea tuturor combinațiilor posibile de parametri, o păstrează pe cea mai bună. De exemplu, dacă grila conține doi parametri și pentru fiecare parametru sunt date două valori, atunci funcția va evalua un estimator de patru ori, de fiecare dată cu o combinație diferită de valori ale parametrilor din grilă și va păstra funcția SVC care conține cele mai bun scor.

O abordare bună este utilizarea GridSearchCV cu StratifiedKFold. Această funcție oferă indicii de antrenament și test pentru a împărți datele în seturi de antrenament și test. Singura problemă a fost că, din cauza memoriei reduse, nu am putut folosi toate datele pe care le-am folosit pentru antrenament, prin urmare, a trebuit să scad numărul de probe până la 2000 și să salvez alte două fișiere .npy care conțin aceste caracteristici și etichete folosind codul python prezentat în figura 6. După acest proces, am construit metodologia de căutare a celor mai bune valori ale parametrilor C și gamma prin căutare.

Deoarece parametrii care au cea mai mare influență în scorul de precizie al unui clasificator sunt C și gamma, am început să caut cea mai bună combinație dintre aceștia. În primul rând, am modificat în cod intervalul de valori C pentru a fi de la 0,00001 la 100 inclusiv (C_range = 10,0 ** np.arange (-5, 2)), iar intervalul de valori gama va fi de la 0,000000001 la 100 inclusiv (gamma_range = 10,0 ** np.arange (-9,2)). Am ales aceste valori pentru că o valoare scăzută de C ar determina optimizatorul să caute un hiperplan cu marjă mai mare, iar o valoare scăzută de gamma ar face ca limita de decizie să depindă doar de cele mai îndepărtate puncte. De asemenea, am ales ca kernel-ul estimatorului să fie „rbf”, deoarece acest nucleu este cel mai frecvent utilizat în clasificarea SVC. După ce am schimbat intervalele, am rulat codul, iar cel mai bun clasificator este: SVC(C=10.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=1e-09, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False) dupa o durată de rulare: 24 ore și 11 minute

Apoi am antrenat un clasificator care are acești parametri și acest proces a durat încă două ore și opt minute, ceea ce este ceva mai lung decât timpul obișnuit petrecut pentru antrenament. Pentru a vedea dacă clasificatorul instruit folosind parametrii reglați găsiți este de fapt mai exact, a trebuit să îl testez. După treizeci și patru de minute am obținut scorul de precizie pentru acesta, care este de 69,28%. Acesta a fost un pas uriaș în procesul de găsire a celui mai bun clasificator pentru acest proiect, deoarece acest clasificator este mult mai precis decât toate celelalte instruite înainte.

În speranța că aș putea obține un scor de precizie chiar mai bun, am efectuat o altă căutare, însă de data aceasta, setând intervalele de valori ale C și gamma de la 1e-9 la 1000, respectiv de la 1e-12 la 100.

Deși aveam așteptări mari, dupa 36 de ore, a doua desfășurare a căutării a găsit aceeași combinație de valori C și gamma.

3.2.5 Analizarea rezultatelor

Rezultatul testării pentru ultimul clasificator instruit a încheiat faza inițială a acestui proiect care a constat în găsirea unui set bun de parametri și apoi să antreneze un model bazat pe aceștia care să aibă o rată de precizie suficient de mare pentru a prezice dacă o globulă roșie este sau nu infectată cu parazitul Plasmodium. Clasificatorul ales pentru a fi cel care va fi păstrat pentru faza finală este cel care are cea mai mare precizie. Întrucât dimensiunea fiecărui clasificator este de aproximativ 3 GB.

3.3 Evaluarea celulelor segmentate

A treia și ultima fază a constat i crearea unui fișier python care să utilizeze cel mai bun clasificator găsit și să clasific imaginile celulare segmentate ale unui micrograf, calculând astfel procentul de celule infectate. Am vrut ca software-ul final să fie interactiv cu utilizatorul, de aceea am folosit o intrare în care utilizatorul trebuie să specifice numele directorului în care se află imaginile celulelor segmentate. Dacă utilizatorul specifică un nume care nu este un director, software-ul ar imprima un mesaj informativ de eroare. În caz contrar, software-ul ar citi toate celulele segmentate, le-ar stoca într-un tablou tridimensional, ar putea schimba noua matrice, astfel încât să poată fi transmis ca argument la funcția de previziune, să încarce clasificatorul, să lase clasificatorul să facă predicțiile mai întâi și în final, parcurge toată predicția și numără numărul de celule infectate și neinfectate preconizate și tipărește numărul total de celule, numărul de celule infectate și neinfectate preconizate de clasificator.

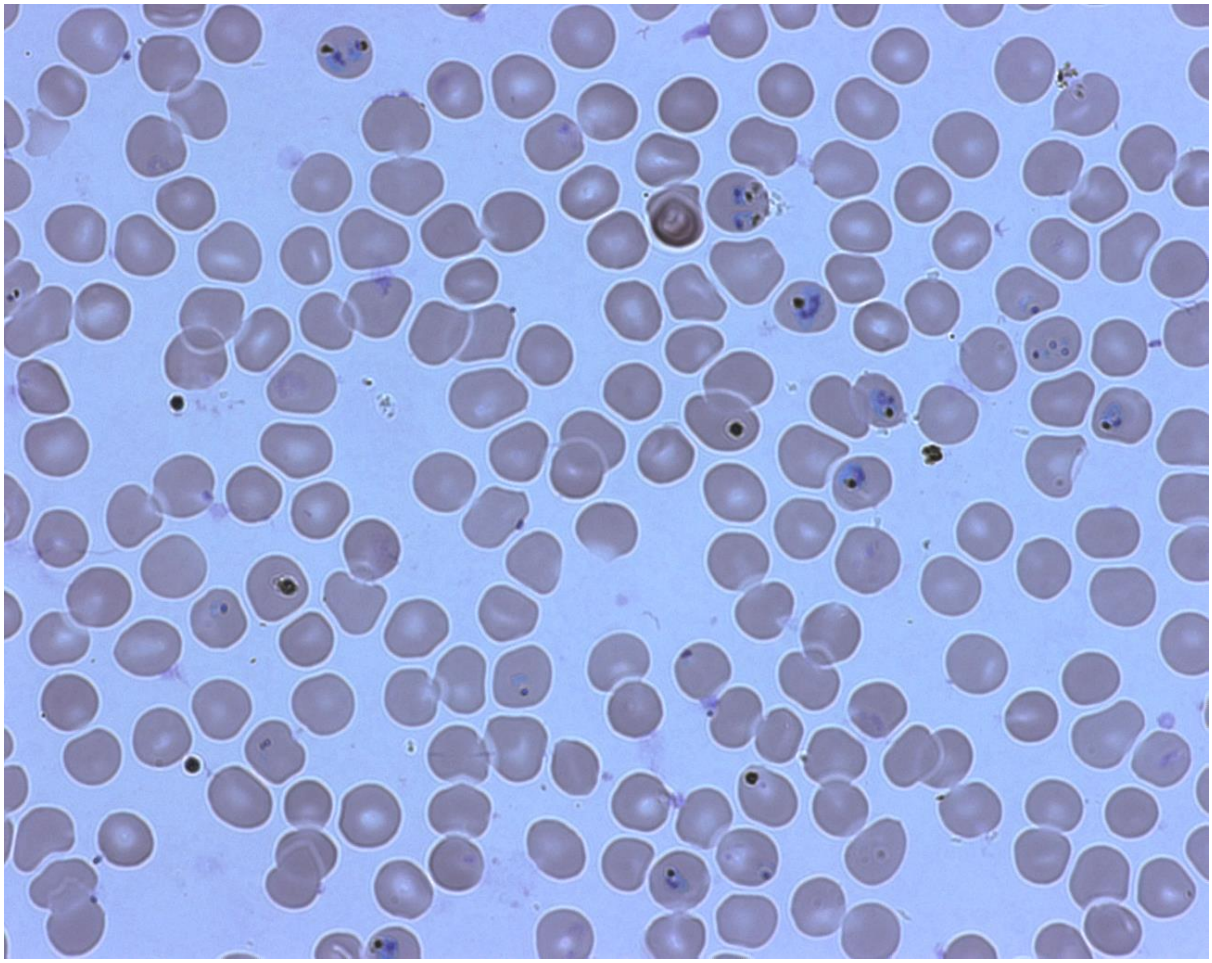

Fig. 13 Micrografie folosită pentru testare

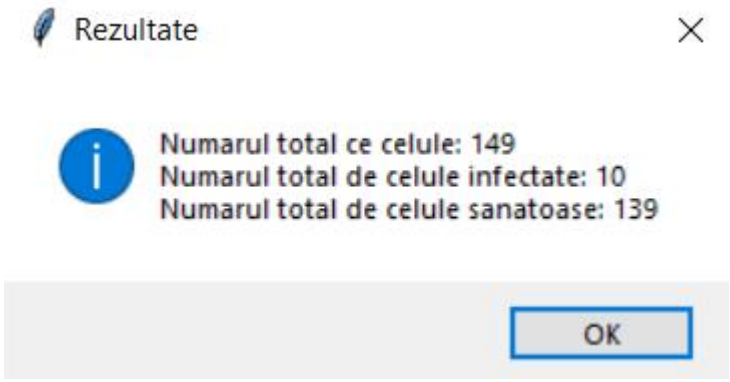

Figure 14 Rezultatele clasificării

Pentru a testa cât de performant este cel mai bun clasificator pe celulele segmentate, am rulat segmentatorul pe micrografie din figura 13 pentru a obține celulele sale ca imagini individuale și apoi am executat fișierul Python creat pentru această fază. Acest micrograf conține 175 celule, din care 13 sunt infectate rezultatele din figura 14 arată că clasificatorul a clasificat greșit 3 celule ca fiind neinfectate.

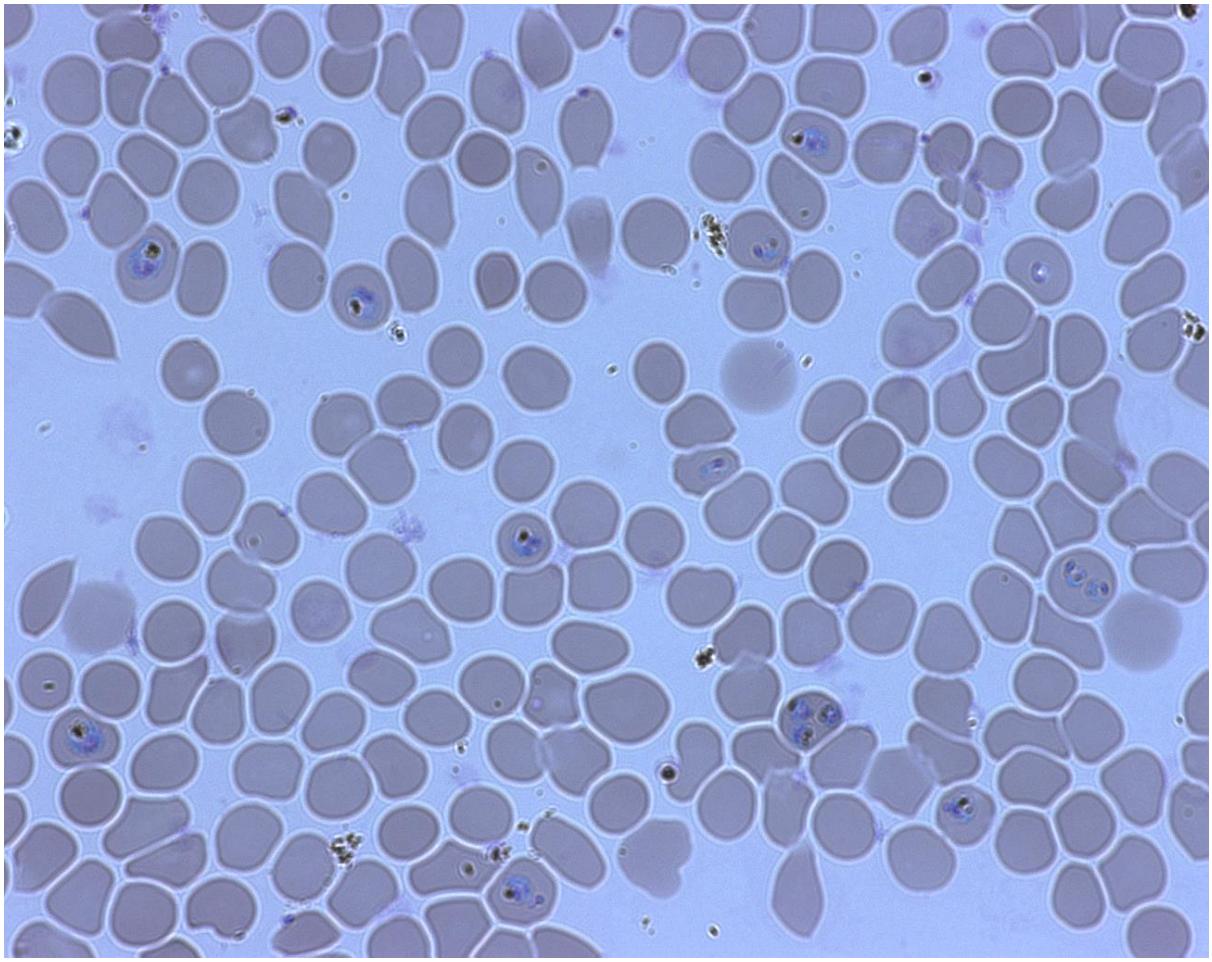

Fig. 15 A doua probă

Am efectuat un alt test pe un micrograf diferit care conține 168 de celule, dintre care 11 sunt infectate și rezultatele clasificatorului sunt prezentate în figura 16. Din aceste rezultate putem vedea că pentru acest al doilea test clasificatorul a clasificat greșit o singură celulă infectată.

În concluzie, clasificatorul clasifică greșit unele celule din cauza lipsei de acuratețe, dar rezultatele sunt satisfăcătoare. Rezultatul predicțiilor clasificatorului depinde și de nivelul de luminozitate al micrografiei și a celulelor.

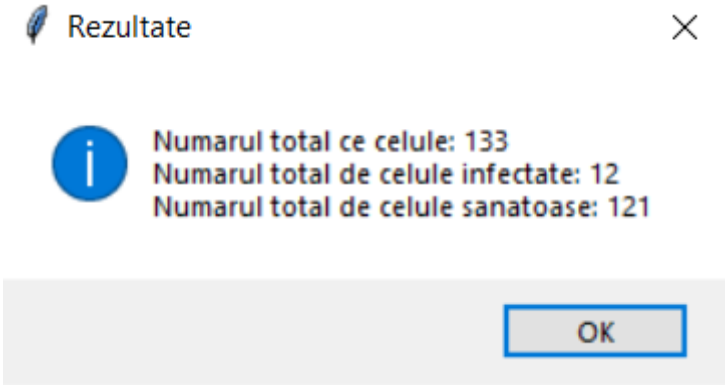

Figure 15 Rezultatele clasificării pe o a doua micrografie

3.4 Interfață

Pentru a combina toate funcționalitățile dezvoltate în acest proiect, am creat într-un fișier Python o interfață care oferă utilizatorului opțiunile de a utiliza segmentatorul sau de a utiliza clasificatorul pentru a calcula procentul din globule roșii infectate dintr-un director care conține toate celulele segmentate ale unui micrograf sau pentru evaluarea unei singure celule. Figura 17 arată output-ul codului.

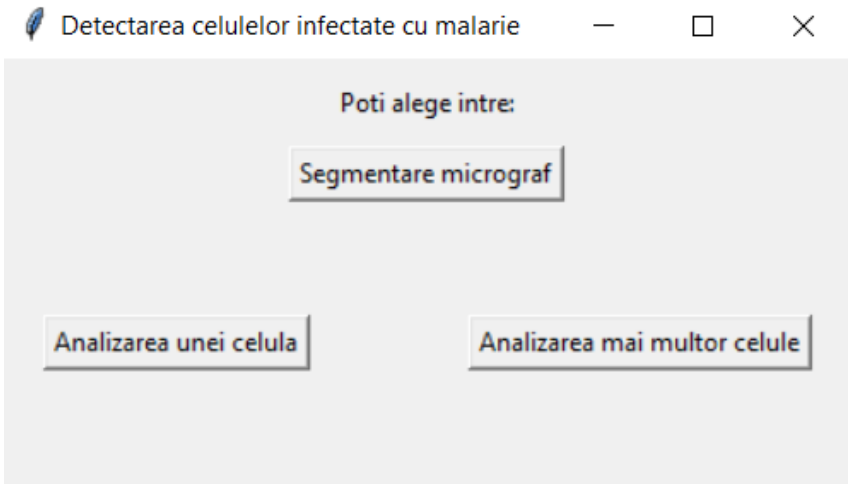

Figure 17 Meniul principal

3.4.1 Instrucţiuni pentru utilizare

Pentru a începe utilizarea software-ului, utilizatorii trebuie să execute fișierul Python numit main.py. Apoi, meniul interfața apare pe ecran, iar utilizatorul are trei opțiuni:

- Butonul “Segmentare micrograf” dacă vrea să segmenteze micrografii
- Butonul “Analizarea unei celule” dacă vrea să evalueze o singura celula
- Butonul “Analizarea mai multor celule” dacă vrea să evalueze un folder care contine mai multe celule

Dacă utilizatorul alege să apese “Segmentare micrograf”, atunci pe ecran va apărea un navigator de fișiere ce îl va ajuta să specifice micrografia ce o dorește a fi segmentată. În cazul în care utilizatorul specifică un fișier greșit, atunci va fi afișat un mesaj de eroare.

În cazul în care utilizatorul dorește sa analizeze celule, acesta are două opțiuni, să analizeze o singură celulă, caz în care un navigator de fișiere va fi deschis pentru a specifica imaginea cu celula ce se dorește a fi analizată, dacă utilizatorul alege altceva decât o imagine, acesta va primi un mesaj de eroare. A doua opțiune este aceea de a analiza un set întreg de celule, caz în car din nou un navigator de fișiere se va deschide iar de data asta utilizatorul va trebui sa selecteze un folder ce va conține imaginile cu celule.

4. Concluzii

Obiectivul primei faze a fost acela de a dezvolta un segmentator care să segmenteze toate globulele roșii dintr-un micrograf și să le producă ca imagini individuale într-un director. Am încercat două moduri de obținere a acestui rezultat, prima cale s-a dovedit a fi inadecvată pentru natura imaginilor pe care am vrut să le segmentez, iar cea de-a doua a putut segmenta, în medie, 80% din celulele unui micrograf. Segmentatorul are, de asemenea, funcționalitatea salvării fiecărei celule sanguine într-o imagine individuală într-un director numit după numele micrografiei.

Pentru următoarea fază a proiectului am creat un clasificator care, după mai multe antrenamente și căutări prin utilizarea unor metode specifice, precum GridSearchCV, am reușit să instruiesc un clasificator care are un scor de precizie de 69%.

A treia fază a constat în scrierea codului și rularea testelor pentru a verifica exactitatea clasificatorului pe datele segmentate. Am rulat mai multe teste, iar rezultatul acestuia a fost un fișier python care îndeplinește funcționalitatea prevăzută, dar întrucât clasificatorul are o precizie mai mică, clasifică greșit unele celule. Este totuși un rezultat satisfăcător al acestei faze.

În sfârșit, mi-am propus să conectez fiecare funcționalitate într-o singură bucată de software și am realizat acest lucru prin crearea unui meniu având ca opțiuni pentru utilizator toate funcționalitățile implementate. Acest software final a fost peste așteptările mele inițiale. Întregul software este interactiv cu utilizatorul și este foarte ușor de utilizat.

5. Bibliografie

[1] Organizație, W. (2012). *World Malaria Report 2012*. Geneva: World Health Organization.

[2] European Centre for Disease Prevention and Control: (2020). *Factsheet about malaria* [online] https://www.ecdc.europa.eu/en/malaria/facts/factsheet

[3] Centers for Disease Control and Prevention (2020). *Malaria* [online] https://www.cdc.gov/malaria/about/biology/index.html

[4] EG Cox, F. (2020). *History of the discovery of the malaria parasites and their vectors*. [online] https://parasitesandvectors.biomedcentral.com/articles/10.1186/1756-3305-3-5

[5] Who.int. (2020). *Fact sheet about Malaria*. [online] https://www.who.int/en/news-room/fact-sheets/detail/malaria

[6] Haditsch, M. (2004). *Quality and reliability of current malaria diagnostic methods*.

[7] Thung, F. și Suwardi, I. (2011). *Blood parasite identification using feature based recognition*. Electrical Engineering and Informatics (ICEEI), 2011 International Conference [online] https://www.researchgate.net/publication/221013718_Blood_parasite_identification_using_feature_based_recognition

[8] Scikit learn (2020). *Machine Learning in Python*. [online] https://scikit-learn.org/stable/

[9] EliteDataScience (2020). *Overfitting in Machine Learning: What It Is and How to Prevent It*. [online] https://elitedatascience.com/overfitting-in-machine-learning#overfitting-vs-underfitting

[10] ScienceDirect (2020). *Image analysis and machine learning for detecting malaria* [online] https://elitedatascience.com/overfitting-in-machine-learning#overfitting-vs-underfitting