Klasifikacija slika metodama dubokog učenja

Novoselnik, Filip

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:527518

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij

KLASIFIKACIJA SLIKA METODAMA DUBOKOG UČENJA

Diplomski rad

Filip Novoselnik

Osijek, 2017.

Sadržaj

1. UVOD 3
2. KRVNE STANICE 5
2.1. Tipovi krvnih stanica 5
2.2. Diferencijalna krvna slika 6
2.3. Uređaji za dobivanje diferencijalne krvne slike i automatsku analizu 7
3. UMJETNE NEURONSKE MREŽE 10
3.1. Konvolucijske neuronske mreže 11
3.1.1. Konvolucijski slojevi 12
3.1.2. Slojevi sažimanja 12
3.1.3 Aktivacijske funkcije 13
3.1.4. Metode regularizacije 14
3.1.5. Učenje konvolucijskih neuronskih mreža 15
4. ALGORITAM ZA DETEKCIJU I IDENTIFIKACIJU BIJELIH KRVNIH STANICA 17
4.1. Dostupne slike za detekciju i identifikaciju bijelih krvnih stanica 18
4.1.1. Način dobivanja slika 18
4.1.2. Web okruženje za označavanje slika priprema skupa podataka za učenje i testiranje 20
4.2. Segmentacija slika 22
4.2.1. Metode segmentacije slika 22
4.2.2. Algoritam segmentacije originalne ulazne slike 23
4.3. Izgradnja konvolucijske neuronske mreže za klasifikaciju bijelih krvnih stanica 24
4.3.1 Arhitektura konvolucijske neuronske mreže 24
4.2.2. Predobrada podataka 26
4.2.3. Korištene metode regularizacije 26
4.2.4. Funkcija gubitka i učenje 27
5. IMPLEMENTACIJA PREDLOŽENOG RJEŠENJA U PROGRAMSKOM JEZIKU PYTHON 28
5.1. Programski jezik Python 28

5.1.1. Python scikit-image biblioteka 28
5.1.2. Programska biblioteka *Keras* 28
5.2. Implementacija algoritma segmentacije slike 29
5.3. Implementacija konvolucijske neuronske mreže 30
5.3.1. Predložena arhitektura modela za klasifikaciju bijelih krvnih stanica 30
5.3.2. Proširenje skupa slika za učenje slučajnim transformacijama 33
5.3.3. Učenje i odabir optimalnog modela 34
5.3.4. Predikcija na temelju izgrađenog modela i prikaz rezultata 36
6. REZULTATI 39
6.1. Testiranje i vrednovanje algoritma segmentacije slika 39
6.2. Testiranje i vrednovanje izgrađenog modela konvolucijske neuronske mreže 43
6.3. Primjer rada predloženog algoritma 44
7. ZAKLJUČAK 46
LITERATURA 48
SAŽETAK 50
ABSTRACT 51
ŽIVOTOPIS 52
PRILOZI 53

1. UVOD

Diferencijalna krvna slika jedna je od osnovnih pretraga kroz koju se određuju omjeri količine pojedinih bijelih krvnih stanica te može ukazivati na mnoga patološka stanja organizma. Diferencijalna krvna slika je posebno zanimljiva sa stajališta zastupljenosti bijelih krvnih stanica u krvnom uzorku čime se dobiva uvid u stanje imunološkog sustava. Promjena u broju neke podvrste bijelih krvnih stanica jako dobro korelira s određenom bolešću (npr. u slučaju leukemije to je drastično povećan broj limfocita). Mnoge od danas korištenih metoda za dobivanje i analizu krvne slike još uvijek se svode na „ručno“ brojanje krvnih stanica u uzorku iili korištenje skupe opreme. Ručno brojanje je vremenski zahtjevno te ga mora raditi obučeni zdravstveni djelatnik. Cilj rada je primijeniti računalne algoritme koji spadaju u grupu strojnog učenja, kako bi se proces dobivanja diferencijalne krvne slike automatizirao, a što bi onda moglo u stvarnoj primjeni i smanjiti cijenu cjelokupnog procesa.

Cilj ovog rada je izrada programske podrške za automatsku detekciju i klasifikaciju bijelih krvnih stanica čime bi se dobila diferencijalna krvna slika. Postupak se načelno sastoji od dva koraka obrade slike krvnih uzoraka. U prvom koraku se pronalaze lokacije svih bijelih krvnih stanica (bazofili, eozinofili, limfociti, monociti, segmentirani neutrofili) na slici. Drugi korak je klasifikacija detektiranih bijelih krvnih stanica kojom se određuje točan tip stanice [1].

Klasični pristup problemu klasifikacije slika i raspoznavanju objekata u računalnom vidu sastoji se od uobičajenih koraka: izlučivanje značajki nekom od metoda (HOG, SIFT, itd.) te klasifikacija dobivenih značajki korištenjem nekog od klasifikatora (SVM, Bayesov naivni klasifikator, itd.). Jedan od problema kod takvog pristupa su vremenska zahtjevnost izrade takvog sustava te problem pronalaženja kvalitetnih značajki objekta.

U posljednjih nekoliko godina počinju se koristiti duboke neuronske mreže. Iako je koncept razvijen još 1980-ih godina, treniranje takvih mreža s dubokim arhitekturama je bilo previše kompleksno i predugo je trajalo. Razvojem hardvera, poglavito GPU-a (engl. *Graphical Processing Unit*) te dostupnošću sve većeg broja podataka duboke neuronske mreže ponovno doživljavaju uzlet. Kod takvog pristupa, i izlučivanje značajki i klasifikacija se potpuno automatski provodi na temelju dostupnog skupa podataka.

Identifikacija bijelih krvnih stanica se u ovom radu zasniva na modelima temeljenim na dubokim neuronskim mrežama. Dobiveni modeli primijeniti će se za klasifikaciju bijelih krvnih stanica na slikama dobivenim iz Kliničke bolnice Osijek.

Rad je strukturiran na sljedeći način. U drugom poglavlju prikazan je pregled tipova krvnih stanica i njihova važnost u medicinskoj dijagnostici. U trećem poglavlju predstavljen je koncept umjetnih neuronskih mreža, dok je u četvrtom poglavlju opisan predloženi algoritam za detekciju

i identifikaciju bijelih krvnih stanica. Implementacijski detalji predloženog algoritma prikazani su u petom poglavlju. Rezultati su predstavljeni u šestom poglavlju, te naposljetku, u sedmom poglavlju dani su zaključci.

2. KRVNE STANICE

Medicinsko laboratorijska dijagnostika jedan je od temelja postavljanja dijagnoze u medicini. Ako se u nalazu nalazi samo broj krvnih stanica onda je riječ o krvnoj slici. Jedna od najvažnijih pretraga je upravo diferencijalna krvna slika, tj. analiza broja bijelih krvnih stanica koje se dijele na eritrocite, limfocite, bazofile, eozinofile i segmentirane neutrofile. Krvnim pretragama otkrivaju se patološki procesi koji se u tijelu odvijaju. Iz tog je razloga hematologija grana medicine koja uvelike ovisi o laboratorijskoj dijagnostici. U kontekstu patoloških procesa važno je prepoznati različite upalne i maligne procese.

2.1. Tipovi krvnih stanica

Razlikuju se bijele i crvene krvne stanice te trombociti. Uloga crvenih krvnih stanica je prijenos kisika. To su jedine stanice u ljudskom organizmu koje nemaju jezgru. Crvene krvne stanice zovu se tako jer su ispunjene hemoglobinom, a to je bjelančevina koja je zaslužna za prijenos kisika. Crvene krvne stanice imaju bikonkavan oblik koji je bitan jer omogućuje eritrocitima veliki omjer površine prema volumenu, čime se eritrocitima olakšava proces izmjene plinova [1].

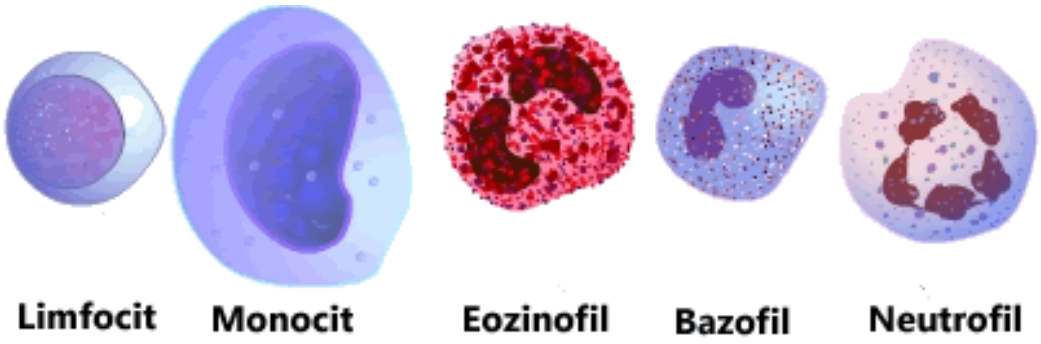
Funkcija bijelih krvnih stanica je prvenstveno imunološka - obrana organizma od bakterija, virusa, parazita i drugih stranih tvari, a neke od njih sudjeluju u reakcijama preosjetljivosti. Bijele krvne stanice imaju jezgru koja može biti raznolikog oblika, od okrugle do jezgre koja je podijeljena u režnjeve. Osim što cirkuliraju u krvi, mogu i izlaziti iz krvožilnog sustava i ulaziti u tkiva (dijapedeza). Bijele krvne stanice na temelju izgleda jezgre dijele se u dvije skupine, a to su: polimorfonuklearni i mononuklearni leukociti. Polimorfonuklearni leukociti imaju jezgru podijeljenu u režnjeve, a mononuklearni nemaju.

Polimorfonuklearni leukociti nazivaju se još i granulocitima zato jer imaju specifična zrnca koja će vezati bazične ili kisele boje. Mononuklearne leukocite zovemo još agranulocitima zato jer ne sadržavaju specifična zrnca. U granulocite (polimorfonuklearne leukocite) ubrajamo: eozinofilne, bazofilne i neutrofilne leukocite. U agranulocite ubrajamo limfocite i mastocite.

Eozinofili imaju crveno obojena zrnca u citoplazmi i vrlo se jasno razlikuju od drugih leukocita, a njihov povećan broj može biti rezultat alergijske reakcije. Neutrofilima je jezgra podijeljena u 2-5 režnjeva, a u svojoj citoplazmi sadržavaju zrnca koja se bojaju neutralno. Imaju sposobnost aktivno fagocitirati male mikroorganizme, pa ih stoga zovemo mikrofazima. Sljedeća skupina polimorfonuklearnih leukocita su bazofili. Bazofili imaju nepravilno podijeljenu jezgru, a u citoplazmi sadrže zrnca koja se boje bazično.

Bijele krvne stanice koje imaju okruglu jezgru i koja ispunjava gotovo cijelu citoplazmu zovemo limfocitima, koji također imaju svoje podskupine. Monociti su bijele krvne stanice koje

se nakon ulaska u tkivo diferenciraju u makrofage. Monociti imaju tipičan izgled „bubrežaste“ ili ovalne jezgre. Makrofazi koji nastaju diferencijacijom monocita služe kao fagocitne stanice, imaju ulogu obrane organizma kako od mikroorganizama tako i od drugih stranih tvari [1]. Na slici 2.1. prikazane su vrste bijelih krvnih stanica.



Sl. 2.1. Klasifikacija bijelih krvnih stanica [2]

2.2. Diferencijalna krvna slika

Diferencijalna krvna slika ili leukogram je krvna pretraga kojom se otkriva broj različitih vrsta bijelih krvnih stanica (leukocita) u krvi. Najčešće se izvodi kao dio kompletne krvne slike (KKS) kao opća procjena zdravlja ili kako bi se pronašao uzrok povišenih ili sniženih vrijednosti u bijeloj krvnoj slici. Pretraga pomaže u dijagnosticiranju i praćenju bolesti koje utječu na imunološki sustav, kao što su infekcije, upalna stanja, leukemija, poremećaji koštane srži i autoimune bolesti. Diferencijalna krvna slika radi se na bazi od 100 stanica. Za pretragu je potrebno pripremiti uzorak venske ili kapilarne krvi. Razmaz uzorka boji se posebnom bojom koja pomaže isticanju razlika između različitih vrsta bijelih krvnih stanica. U određenim slučajevima može se analizirati i druga vrsta tjelesnih tekućina, primjerice cerebrospinalna tekućina ako se sumnja na meningitis.

Pretraga otkriva postotak svake od 5 vrsta bijelih krvnih stanica u uzorku. Stanice uzorka broji i razvrstava zdravstveni djelatnik u laboratoriju, dok se u većim laboratorijima koriste automatizirane metode radi brže obrade uzoraka.

Diferencijalni ili razlikovni broj svakog od pet tipova leukocita utvrđuje jesu li oni prisutni u normalnom broju. U tablici 2.1. prikazane su očekivane vrijednosti udjela pojedinog tipa bijelih krvnih stanica u krvnom uzorku kod zdravog čovjeka.

Tab. 2.1. Očekivane vrijednosti pojedine vrste leukocita u krvnom uzorku [1]

Vrsta leukocita	**Postotak**
Neutrofili	40 – 70 %
Eozinofili	1 - 4 %
Bazofili	< 1 %
Limfociti	20 – 50 %
Monociti	1 – 8 %

Odstupanja najčešće uzrokuju različite infekcije i izloženost radijaciji, a mogu biti i znak vrsta raka. Neka od najčešćih pokazatelja povišenih ili sniženih vrijednosti pojedinih vrsta leukocita prikazani su u tablici 2.2.

Tab. 2.2. Bolesti i stanja uzrokovana smanjenim ili povećanim postotkom određene vrste leukocita [1]

Vrsta leukocita	**Povećani postotak**	**Smanjeni postotak**
Neutrofili	pušenje, gljivična ili bakterijska infekcija, giht, stres, eklampsija	alergijska reakcija na lijek, anemija, gripa, autoimune bolesti
Eozinofili	kronična kožna infekcija, astma, neke vrste raka, Addisonova bolest	teško ga je otkriti jer su vrijednosti i inače niske (najčešće stres)
Bazofili	mijeloproliferativna stanja, alergije, bolesti vezivnog tkiva, vodene kozice	stres ili trauma, alergijske reakcije
Limfociti	kronične bakterijske infekcije, hepatitis, mononukleoza, limfom	sepsa, lupus, zatajenje bubrega, virus HIV-a
Monociti	kronične upalne bolesti, leukemija, parazitske i virusne infekcije	HIV, poremećaj koštane srži, reumatoidni artritis, izloženost radijaciji

2.3. Uređaji za dobivanje diferencijalne krvne slike i automatsku analizu

Današnja tehnologija koja omogućava automatsku ili poluautomatsku detekciju i klasifikaciju krvnih stanica obično je vrlo skupa te mnoge bolnice još uvijek taj posao rade ručno, tj. liječnici pregledavaju preparate pomoću mikroskopa i rade analizu brojanjem odgovarajućih stanica u razmazu.

Međutim, postoje uređaji opremljeni adekvatnim softverom koji se koriste u većim centrima, na primjer, u Hrvatskoj možemo naći stroj pod nazivom „CellaVision“ kojeg trenutno koriste

Klinički bolnički centri Osijek i Split. Takvi uređaji i njihovi zaštićeni programi obično su vrlo skupi.

CellaVision DM1200 je model koji se nalazi u hitnom laboratoriju Kliničkog bolničkog centra Osijek. To je automatizirani sustav namijenjen *in vitro* dijagnostici. CellaVision automatski pregledava dio mikroskopskog razmaza koji odabere osoba koja upravlja uređajem. Automatski prikazuje smještaj i slike stanica na razmazima. Uređaj se može primijeniti u analizi periferne krvi i tjelesnih tekućina. Preparati se bojaju May-Grünwald Giemsa metodom bojanja. Bojanje preparata je automatizirano i odvija se u zasebnom stroju. Na slici 2.2. prikazan je model takvog uređaja.



Sl. 2.2. CellaVision DM1200

Kod pregleda periferne krvi uređaj može diferencijalno brojati bijele krvne stanice, karakterizirati morfologiju eritrocita i procijeniti broj trombocita u uzorku. Ako se koristi u analizi tjelesnih tekućina, tada je važan u diferencijalnom brojanju stanica s jezgrama. CellaVision sastoji se od optičke jedinice koju čini mikroskop i kamera te od računalnog sustava koji sadrži softver za prikupljanje i klasifikaciju. Stroj sadrži i čitač barkodova radi smanjenja greške pri klasificiranju uzoraka. Za stroj je potrebno kupovati imerzijsko ulje, a slikanje preparata odvija se pod povećanjem od 220 puta.

Uređaj CellaVision proizvodi se u Švedskoj. Potreba za ovakvim strojem opravdana je u većim zdravstvenim centrima. Broj preparata koji u takvim ustanovama dođu na obradu vrlo je

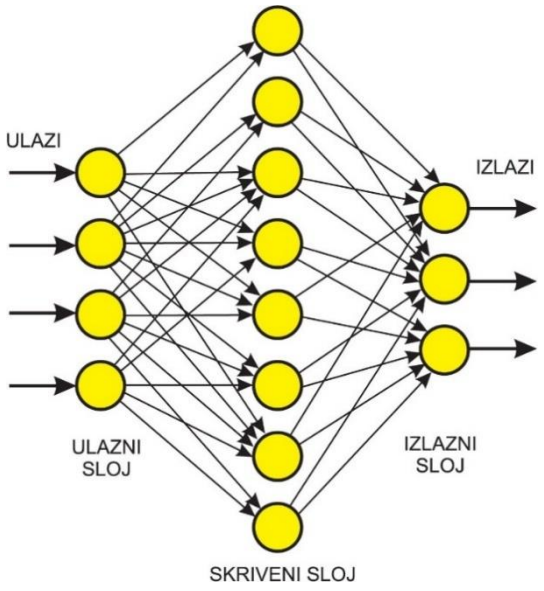
velik, stoga automatizirani mehanizam skraćuje posao i omogućuje brzo dobivanje diferencijalne krvne slike što je vrlo bitno ukoliko se radi o hitnom slučaju.

Cilj ovog rada je istražiti moguće načine izrade sličnog takvog sustava, uz primjenu dostupne opreme poput mikroskopa i kamere, koji bi bio približno efikasan u dobivanju diferencijalne krvne slike, ali znatno jeftiniji.

3. UMJETNE NEURONSKE MREŽE

U umjetnoj inteligenciji, neuronske mreže predstavljaju skupinu modela inspiriranih biološkim neuronima. Pokazale su se kao iznimno koristan alat za rješavanje klasifikacijskih i regresijskih problema kada je teško ili nemoguće pronaći rješenje klasičnim algoritmima koji su temeljeni na eksplicitnim pravilima. Umjetne neuronske mreže općenito su prikazane pomoću neurona koji su slojevito organizirani te primaju i prosljeđuju signale između sebe. Neuroni predstavljaju osnovnu jedinicu umjetne neuronske mreže te su međusobno povezani vezama kojima se dodjeljuju težine koje se podešavaju postupkom učenja. Učenje neuronske mreže je zapravo podešavanje težina sve dok se ne dobije zadovoljavajuća aproksimacija između ulaznih i izlaznih veličina.

Jedna od osnovnih struktura je unaprijedna neuronska mreža koja se sastoji od jednog ulaznog sloja, jednog ili više skrivenih slojeva te izlaznog sloja. Takva neuronska mreža najčešće se koristi za određeni problem klasifikacije uzorka ulaznih veličina u različite kategorije. Ulazni sloj se sastoji od onoliko neurona kolika je dimenzionalnost ulaznog prostora, a broj neurona na izlazu jednak je broju klasa. Broj skrivenih slojeva je proizvoljan, iako sama struktura skrivenog sloja utječe na performanse mreže te je jako bitno odrediti optimalnu strukturu mreže kako bi se dobili zadovoljavajući rezultati. Na slici 3.1. prikazana je struktura unaprijedne neuronske mreže s jednim ulaznim, jednim skrivenim te jednim izlaznim slojem. Žuti krug predstavlja neuron, dok je strelica između dva neurona veza koja ima određenu težinu. Ulaz sadrži četiri neurona odnosno četiri ulazne veličine, skriveni sloj sadrži osam neurona, dok izlaz sadrži tri neurona odnosno mreža klasificira ulaznu veličinu u jednu od tri klase.



Sl. 3.1. Unaprijedna neuronska mreža

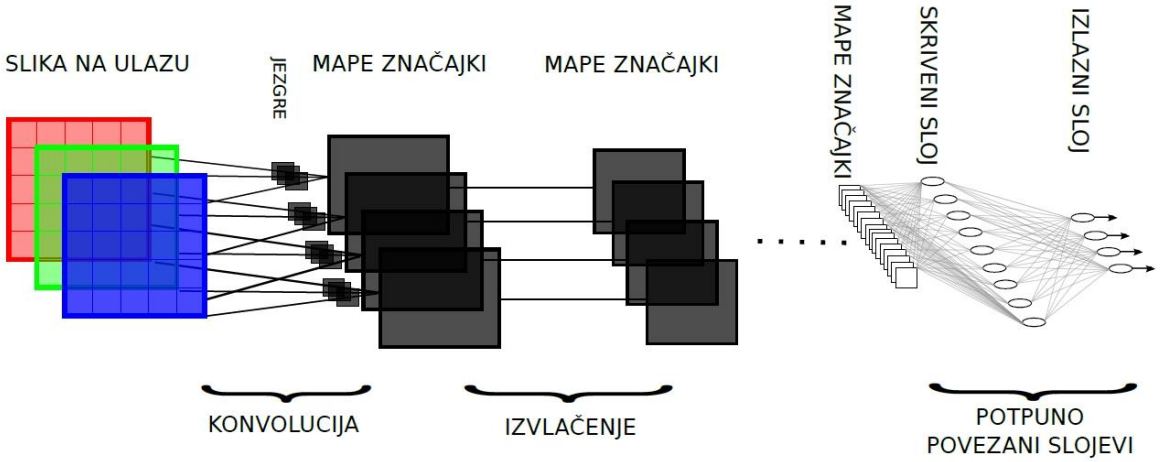
Zadnji sloj je inače potpuno povezani klasifikacijski sloj neurona s linearnim izlazom. Nakon njega slijedi *softmax* funkcija koja djeluje na izlaze zadnjeg sloja neurona. *Softmax* funkcija daje probabilističku vrijednost pripadnosti uzorka određenoj klasi.

Ukoliku bi uklonili zadnji, klasifikacijski sloj neuronske mreže, tada bi izlaz zadnjeg skrivenog sloja predstavljao deskriptor ulaznog podatka, tj. sažet prikaz ulaznog uzorka u kojem su pohranjene njegove bitne značajke. Na taj način moguće je izvući bitne značajke ulaznog uzorka te koristiti unaprijed istreniranu mrežu kao izlučivač značajki (engl. *feature extractor*). Tako dobivene značajke moguće je koristiti kao ulaz u neki od klasifikatora kao što su npr. SVM, Logistička regresija, itd. Navedena metoda naziva se još i učenje prijenosom (engl. *transfer learning*).

3.1. Konvolucijske neuronske mreže

U dosad opisanoj strukturi neurona izlaz iz svakog od njih je bio skalarna veličina. U konvolucijskim neuronskim mrežama uvodi se proširenje u obliku konvolucijskih slojeva, tj. izlazi iz takvih elemenata su dvodimenzionalni i nazivamo ih mapama značajki (engl. *feature maps*). Ulaz u takve neuronske mreže je dvodimenzionalan (slike), a umjesto težina koriste se jezgre (engl. *kernels*).

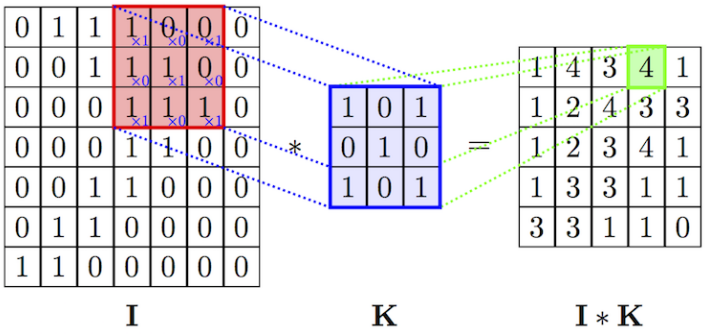
Na slici 3.2. prikazana je opća struktura konvolucijske neuronske mreže. Na ulaz se dovodi slika koja je u ovom slučaju RGB, tj. sastoji se od tri kanala. Osim višekanalnih, slike mogu biti i monokromatske. Unutarnja struktura mreže se sastoji od nekoliko naizmjeničnih konvolucijskih slojeva i slojeva sažimanja (engl. *pooling layers*). Na kraju se nalazi nekoliko potpuno povezanih slojeva koji su jednodimenzionalni.



Sl. 3.2. Struktura konvolucijske neuronske mreže

3.1.1. Konvolucijski slojevi

Konvolucija je matematički operator definiran nad dvije funkcije realne varijable. Određena ulazna reprezentacija podatka (originalna slika) konvolvira se s konvolucijskom jezgrom s određenim parametrima. Procesom učenja ovi parametri se podešavaju. U slučaju konvolucijskih neuronskih mreža kombiniraju se mapa značajki i konvolucijska jezgra koja se može interpretirati kao filter. Konvolucijska jezgra filtrira sliku, tj. mapu značajki kako bi izlučila neku korisnu informaciju kao što je recimo određeni oblik, boja ili rub. U prvih nekoliko slojeva izlučuju se osnovni oblici (npr. rubovi). Obično se koristi više konvolucijskih slojeva koji svojom kombinacijom mogu izlučiti apstraktnije razine značajki karakteristične za pojedini objekt (npr. ljudsko lice). Na slici 3.3. prikazana je operacija konvolucije gdje **I** predstavlja originalnu sliku, dok **K** predstavlja jezgru kojom je originalna slika konvolvirana. Ovisno o tipu jezgre moguće je detektirati razne značajke (rubove, točke). Vrijednosti parametara jezgre se podešavaju procesom učenja tj. konvolucijske neuronske mreže na taj način „nauče“ prepoznavati određene značajke. Operacijom konvolucije dobiva se transformirana slika **I*K**. Dobivena transformacija je lokalna tj. pikseli izlaza ovise o lokalnom susjedstvu piksela ulaza.

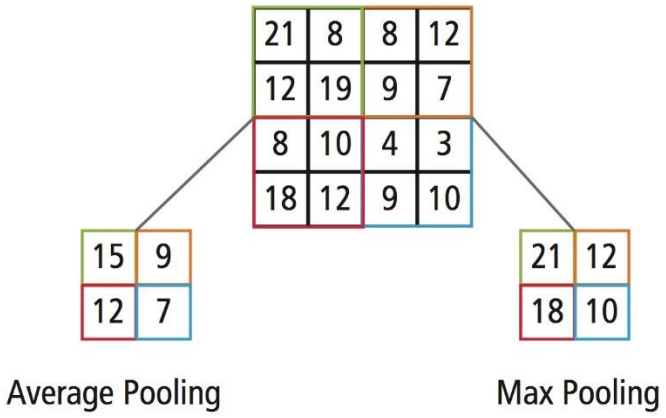


Sl 3.3. Operacije konvolucije originalne između slike (**I**) i jezgre (**K**) [3]

3.1.2. Slojevi sažimanja

Nakon konvolucijskih slojeva, obično dolaze slojevi sažimanja koji reduciraju mapu značajki na manju rezoluciju, tj. određeni dio mape značajki postavljaju na jednu vrijednost. U početku se radilo tako da se određeni dio mape značajki zamijeni aritmetičkom sredinom. Kasnije se pokazalo da se puno bolji rezultati dobivaju korištenjem maksimalne vrijednosti. Sažimanjem se postiže invarijantnost na blage rotacije i translacije. Na primjer, ako je objekt kojeg želimo prepoznati blago pomaknut mreža će svejedno uspjeti dobro prepoznati objekt jer se operacijom sažimanja dobila jednaka vrijednost u oba slučaja, tj. funkcija sažimanja mapira skup prostorno bliskih značajki na ulazu u jednu značajku na izlazu.. Osim toga, sažimanjem se smanjuju memorijski zahtjevi implementacije umjetne neuronske mreže. Ipak, treba biti oprezan, budući da se dio informacija može izgubiti ako je područje sažimanja preveliko. Na slici 3.4. prikazana

je operacija sažimanja korištenjem maksimalne vrijednosti unutar prozora (engl. *max pooling*) i srednje vrijednosti prozora (engl. *average pooling*).



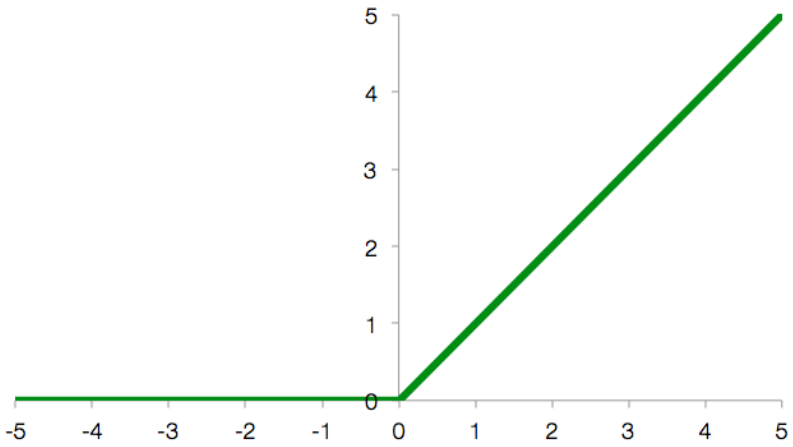
Sl. 3.4. Operacija sažimanja slike (engl. *pooling*) [4]

3.1.3 Aktivacijske funkcije

Aktivacijska funkcija propagira ili zaustavlja ulaznu vrijednost u neuron ovisno o svom obliku. Postoji veliki broj aktivacijskih funkcija od kojih su najpoznatije linearne, funkcije skoka, sigmoidalne itd. Jedna od najčešće korištenih aktivacijskih funkcija u konvolucijskim neuronskim mrežama je *ReLU* (engl. *Rectified Linear Unit*) aktivacijska funkcija. *ReLU* funkcija *f* je definirana na sljedeći način:

$$f(x) = \max(0, x) \tag{3-1}$$

gdje je *x* ulaz u neuron. Ova aktivacijska funkcija unosi nelinearnost u model te model najčešće znatno bolje opisuje stvarni objekt budući da je svaki kompleksni sustav u manjoj ili većoj mjeri nelinearan. Osim toga, pokazalo se da ova aktivacijska funkcija puno bolje opisuje stvarno ponašanje biološkog neurona. U usporedbi s ostalim aktivacijskim funkcijama kao što je sigmoidalna funkcija, *ReLU* omogućava brže i efikasnije učenje dubokih neuronskih mreža na velikim i kompleksnim skupovima podataka. Na slici 3.5. prikazana je ReLU aktivacijska funkcija.

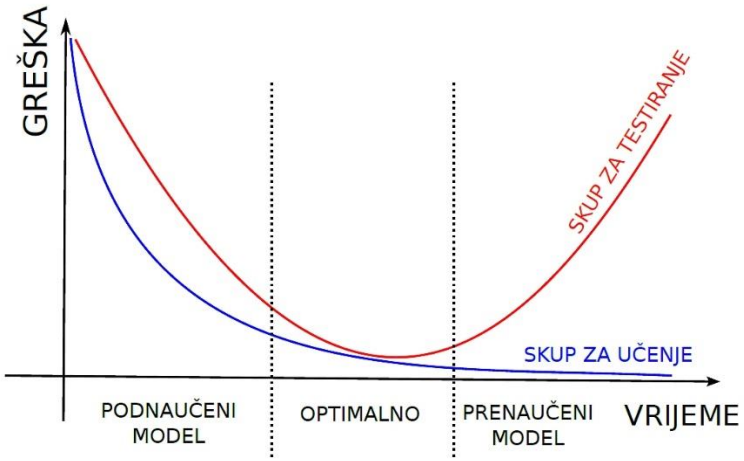


Sl. 3.5. ReLU aktivacijska funkcija

3.1.4. Metode regularizacije

Regularizacija je jedan od važnih koncepata u strojnom učenju te jedan od razloga zašto je moguće postići dobra generalizacijska svojstva i kada je na raspolaganju manji skup podataka za učenje modela. Pod pojmom regularizacije podrazumijeva se smanjenje greške generalizacije, tj. smanjenje greške na podacima koje model prvi puta vidi. Neke od metoda regularizacije su: rano zaustavljanje (engl. *early stopping*), regularizacija korištenjem L1 i L2 normi, dodavanje *dropout* sloja te proširenje skupa podataka za učenje slučajnim transformacijama [5]. U ovom radu korištene s upravo posljednje dvije metode.

Učenjem se klasifikator sve više prilagođava skupu za učenje te je moguće da se s vremenom previše prilagodi skupu za učenje. U tom slučaju model postaje prenaučen (engl. *overfitting*) te će loše klasificirati one uzorke koje nije vidio (loša generalizacija). Problem prenaučenosti i podnaučenosti modela ilustriran je na slici 3.6.



Sl. 3.6. Greške modela na skupu za treniranje i skupu za testiranje [6]

Dropout je zapravo tehnika regularizacije koja ima za cilj smanjiti kompleksnost modela te spriječiti prenaučenost. Korištenjem *dropouta* nasumično se „deaktiviraju“ pojedini neuroni u sloju s određenom vjerojatnosti, obično iz Bernoullijeve distribucije (najčešće 0.5). To kao posljedicu ima da se mreža neće moći „oslanjati“ na određene neurone nego će učiti što različitije reprezentacije uzoraka. Prednost uvođenja ove tehnike je i brže treniranje mreže budući da treba podesiti upola manje parametara. *Droput* se primjenjuje tijekom samog treniranja mreže. Osim toga, potrebno je reskalirati preostale vrijednosti neurona. Ako je 50 % neurona postavljeno na nulu, preostale, tj. one koji se aktiviraju potrebno je skalirati faktorom 2.

3.1.5. Učenje konvolucijskih neuronskih mreža

Duboke neuronske mreže obično sadrže veliki broj parametara (obično nekoliko milijuna) koje je potrebno podesiti kako bi mreža ispravno radila klasifikaciju. Postupak podešavanja parametara neuronske mreže naziva se učenje ili treniranje. Učenje je iterativni postupak u kojem se na ulaz neuronske mreže dovodi uzorak za kojeg je poznata vrijednost izlazne veličine, pri čemu dolazi do postupnog podešavanja težina neurona korištenjem nekog od algoritama matematičke optimizacije. Matematička optimizacija podrazumijeva pronalaženje minimuma ili maksimuma odgovarajuće kriterijske funkcije. Postupak se svodi na pretraživanje *n*-dimenzionalnog prostora parametara, gdje je *n* ukupan broj težina u mreži. Pogreška u takvom prostoru može se vizualizirati kao hiper-površina s više lokalnih minimuma.

Najčešće korišteni algoritam za učenje neuronskih mreža je *backpropagation* algoritam, tj. algoritam s povratnim prostiranjem pogreške. Algoritam se temelji na optimizacijskoj metodi gradijentnog spusta. Kroz praksu se pokazao kao računski vrlo isplativ algoritam. Vrlo ga je jednostavno proširiti i za učenje konvolucijske neuronske mreže uzimajući u obzir neke specifičnosti u konvolucijskim slojevima i slojevima sažimanja.

Težine konvolucijskih slojeva potrebno je inicijalizirati tako da su pomaci (engl. *bias*) svih neurona postavljene na nulu, a težina svakog neuona je broj iz normalne razdiobe čija je srednja vrijednost nula.

Najprije se definira kriterijska funkcija, a učenje neuronske mreže svodi se na minimizaciju kriterijske funkcije. Kod korištenja gradijentnih postupaka optimizacije potrebno je definirati i stopu učenja (engl. *learning rate*) koja zapravo određuje veličinu koraka kojim se pretražuje prostor parametara. U slučaju odabira premale vrijednosti stope učenja, sustav će sporo konvergirati, dok u slučaju odabira prevelike vrijednosti sustav može previše oscilirati i preskakati optimume. U prvom koraku poznati uzorak se dovodi na ulaz mreže te se gleda odziv na izlazu iz mreže. Zatim se računa greška, tj. razlika između odziva mreže i očekivanog izlaza za označeni uzorak. U drugom koraku greška se širi unazad te se mreža ažurira kako bi se greška

smanjila. Algoritam *backpropagation* računa grešku u svakom sloju te ažurira težine na temelju gradijenta smanjujući grešku neuronske mreže. Optimizacijski problem koji se rješava metodom gradijentnog spusta ne jamči da je pronađen globalni minimum kriterijske funkcije.

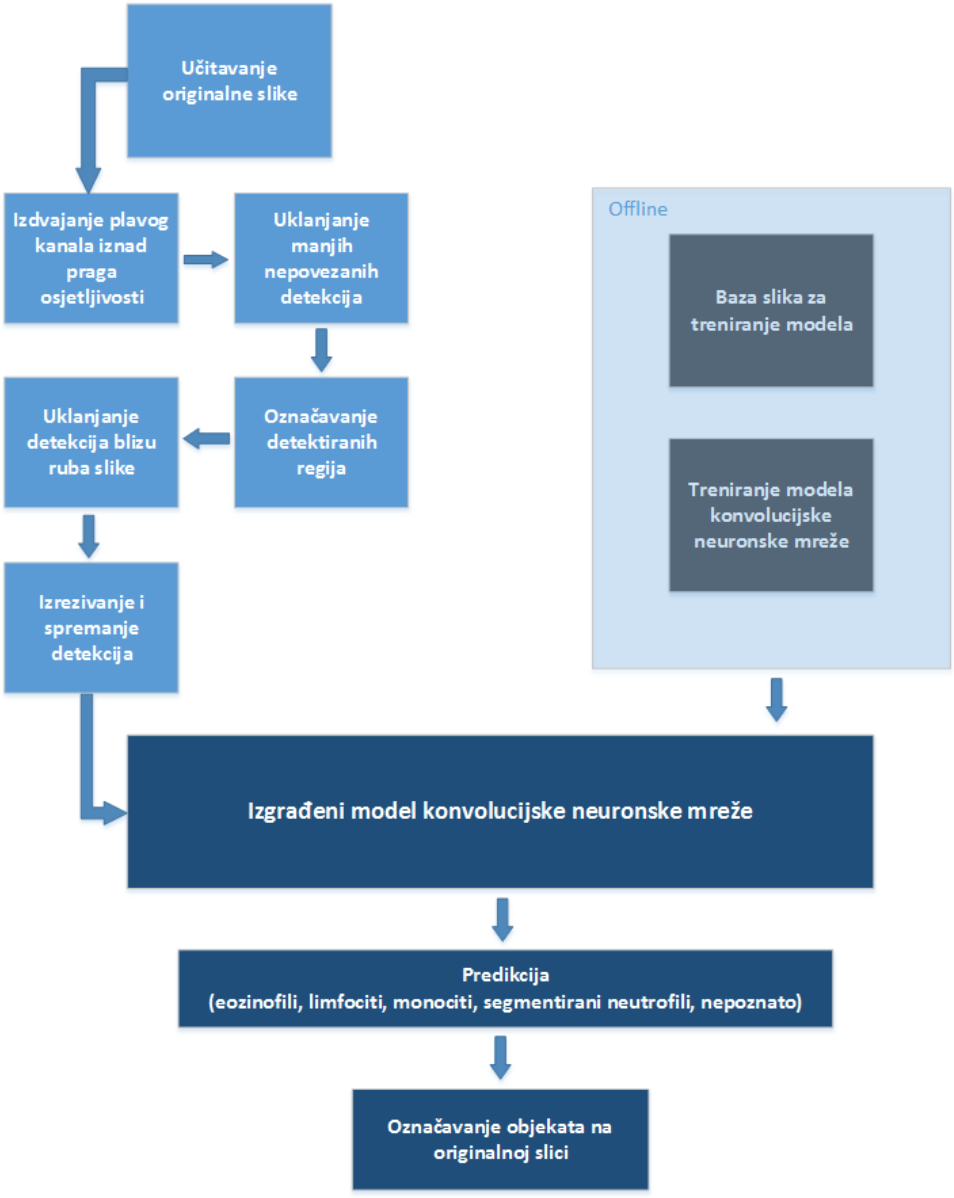
Kao postupak optimizacije u ovom radu korištena je *RMSprop* metoda. Ova metoda predložena je od strane Geoffa Hintona u popularnom *online* kolegiju na platformi za učenje *Coursera* [7]. Metoda *RMSprop* predstavlja jednu od verzija stohastičkog gradijentnog spusta te koristi adaptivnu stopu učenja. Ova metoda pokušava riješiti problem jako malih vrijednosti gradijenata na način da trenutni gradijent dijeli s eksponencijalno isčezavajućim prosjekom kvadrata gradijenata.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \varepsilon}} g_t \tag{3-2}$$

θ_t predstavlja vrijednost težine $E\left[g^2\right]$ prosjek gradijenata unutar pomičnog prozora, ε je šum koji se nadodaje kako bi se uvela nesigurnost. Hinton predlaže početnu vrijednost $\eta = 0.001$ [17].

4. ALGORITAM ZA DETEKCIJU I IDENTIFIKACIJU BIJELIH KRVNIH STANICA

U ovom radu predstavljen je algoritam za detekciju i identifikaciju bijelih krvnih stanica prikazan na slici 4.1. Originalna ideja je bila na temelju dostupne baze slika bijelih krvnih stanica izgraditi konvolucijsku neuronsku mrežu koja klasificira bijele krvne stanice u jedan od pet tipova. Izgradnja modela napravljena je *offline*, tj. napravljena je prije same analize slika novih krvnih uzoraka. Slike krvnih uzoraka se segmentiraju kako bi se izdvojile sve bijele krvne stanice. Segmentirane slike bijelih krvnih stanica zatim se dovode na ulaz klasifikatora koji je unaprijed istreniran te radi predikciju u jednu od pet klasa (eozinofili, segmentirani neutrofili, monociti, limfociti ili nepoznato). Na samom kraju su stanice označene na originalnoj slici zajedno s pripadnom klasom.



Sl. 4.1. Arhitektura predloženog rješenja za detekciju i identifikaciju bijelih krvnih stanica

4.1. Dostupne slike za detekciju i identifikaciju bijelih krvnih stanica

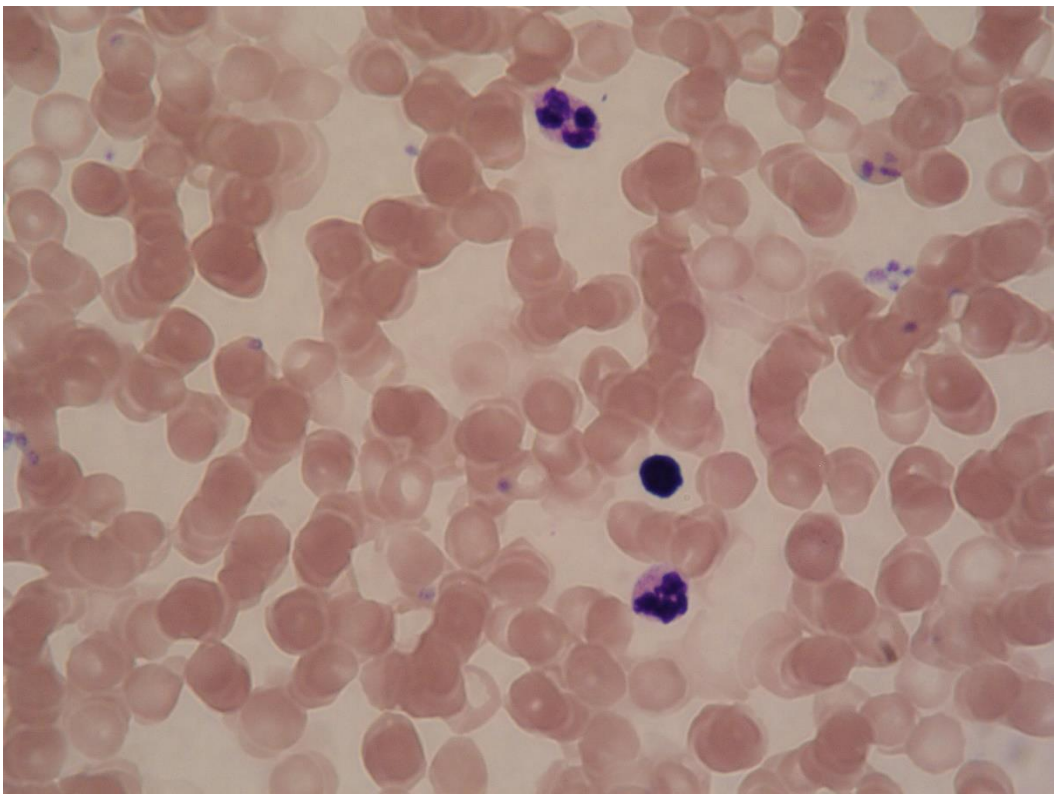
4.1.1. Način dobivanja slika

Slike preparata krvnih razmaza su dobivene od Medicinskog fakulteta u Osijeku. Dobiveno je ukupno 412 slika veličine 2560 x 1920 piksela. Slike su fotografirane s digitalnim fotoaparatom marke Olympus® (model C-5050) spojenim na mikroskop marke Olympus®, model BX-50 (slika 4.2.) uz pomoć računalnog programa *QuickPHOTO Pro*. Preparati su također vlasništvo fakulteta. Mikroskop je priključen na fotoaparat, a cijeli sustav na računalo. Korišteni mikroskop prikazan je na slici 4.2.

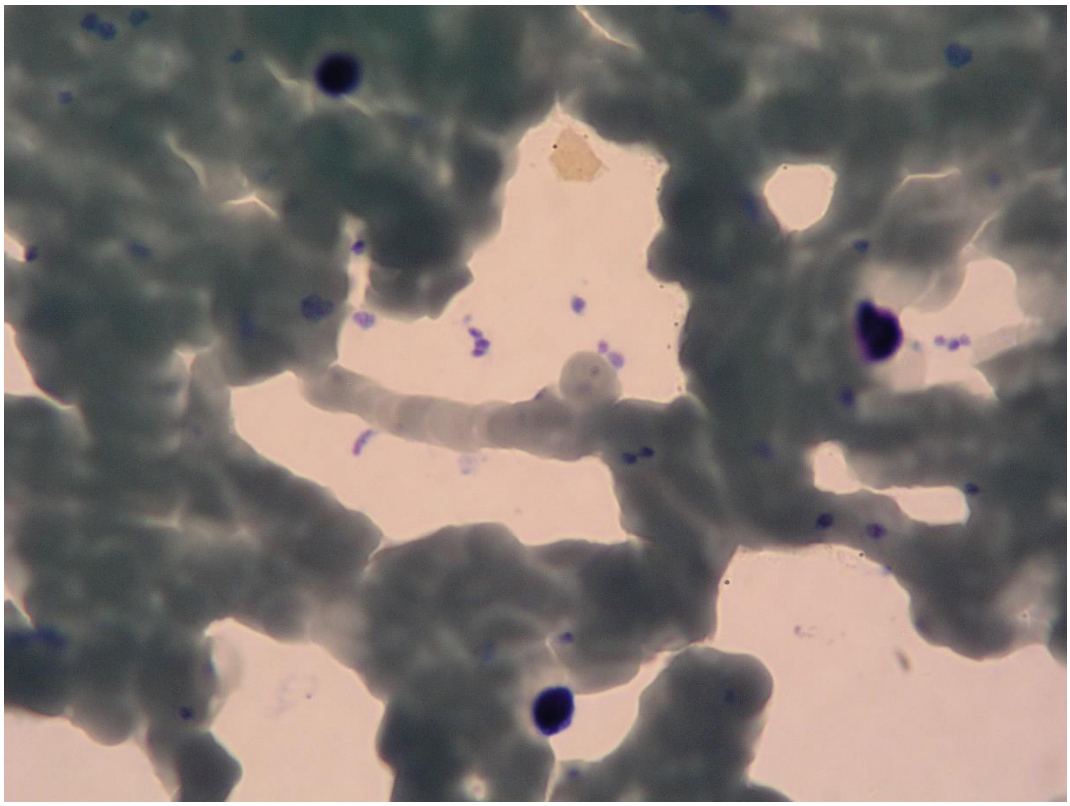


Sl. 4.2. Mikroskop Olympus® BX-50 korišten za dobivanje slika [8]

Preparati su razmazi kapilarne krvi koji su obojani metodom hemalaun eozin. Ova se boja standardno koristi u pripremi histoloških preparata, kako za studentsku izobrazbu tako i za izradu patoloških preparata. Slike variraju u kvaliteti što prvenstveno ovisi o kvaliteti bojenja te razmaza. Na slici 4.3. prikazana je dobro obojana i razmazana slika, dok je na slici 4.4. prikazana loše obojana i razmazana slika gdje su neke stanice uništene.



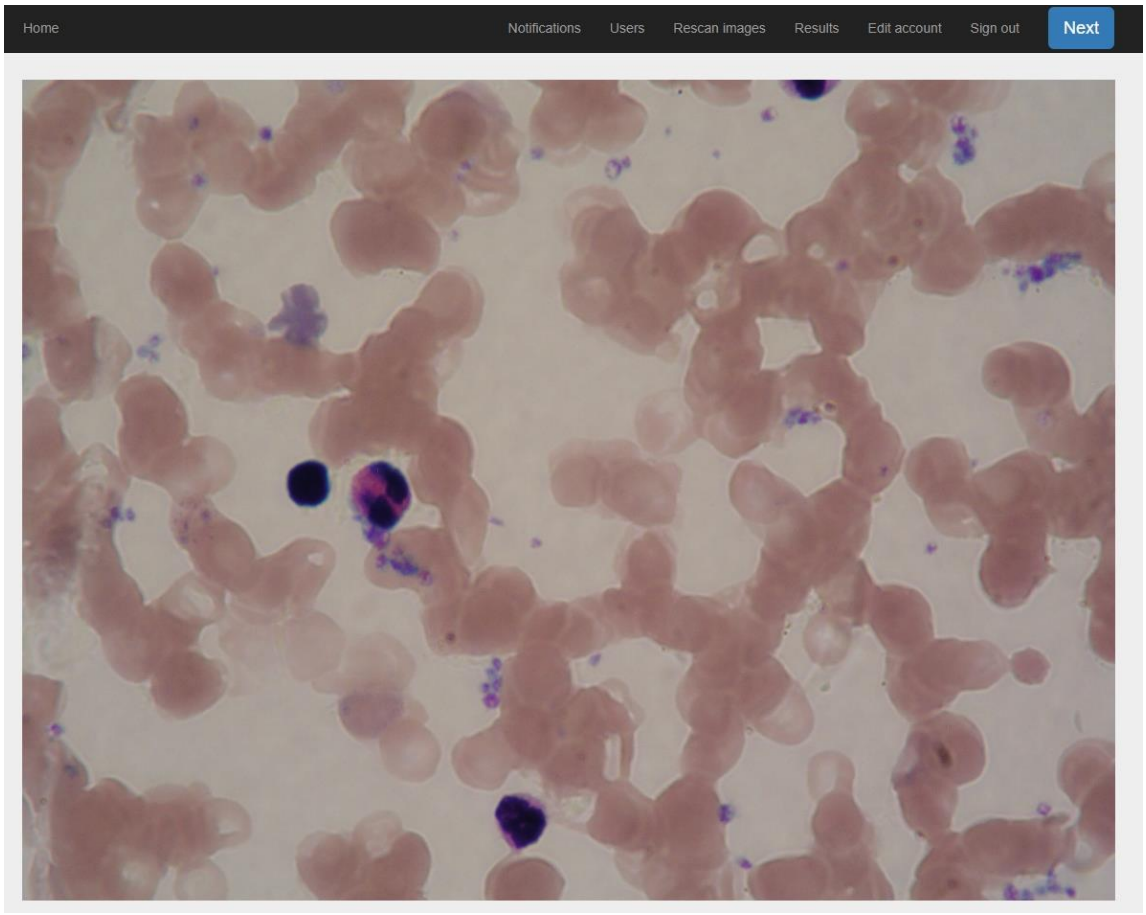
Sl. 4.3. Originalna slika krvnog razmaza (dobra kvaliteta)



Sl. 4.4. Originalna slika krvnog razmaza (loša kvaliteta)

4.1.2. Web okruženje za označavanje slika priprema skupa podataka za učenje i testiranje

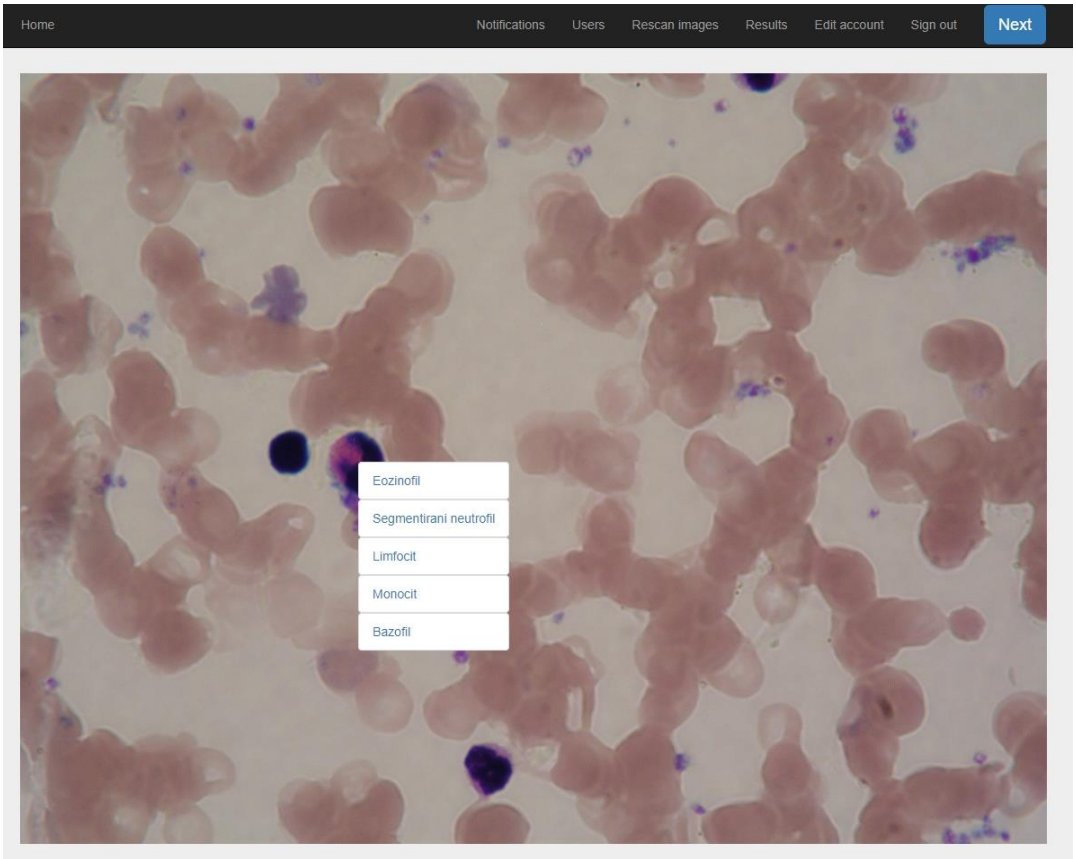
Pri korištenju algoritama nadgledanog učenja (engl. *supervised learning*) jako je važno imati dobro označene podatke za treniranje modela, tj. za svaki mjerni uzorak ulaznih veličina postoji odgovarajuća vrijednost izlazne veličine. U slučaju klasifikacije stanica ulazni podatak je slika bijele krvne stanice, a izlaz je jedna od 5 klasa (eozinofil, bazofil, monocit, limfocit, segmentirani neutrofil). U sklopu suradnje između Medicinskog fakulteta Osijek i Fakulteta elektrotehnike, računarstva i informacijskih tehnologija izrađena je web stranica na kojoj su postavljene sve dobivene slike krvnih uzorak. Stranica nije dio ovog diplomskog rada, izrađena je zasebno. Na slici 4.5. prikazano je web sučelje za označavanje takvih slika.



Sl. 4.5. Web sučelje za prikaz i označavanje slika

Na stranici se učitava slika po slika te je omogućeno označavanje određenog tipa stanice na način da se desnim klikom približno označi središte stanice. Nakon klika dobije se padajući izbornik gdje se odabire tip stanice (bazofili, eozinofili, segmentirani neutrofili, limfociti, monociti). Primjer rada prikazan je na slici 4.6. Koordinate označene stanice na slici zajedno s odabranim tipom spremaju se u bazu (slika 4.7.). Sve slike označene su od strane medicinskog djelatnika. Osim navedenih klasa dodana je još i klasa *Nepoznat*, tj. za slučaj da slika sadrži niti jednu od 5 navedenih bijelih krvnih stanica nego je rezultat pogrešno segmentiranog objekta.

Slike sadržane u klasi *Nepoznat* dobivene su nasumičnim izrezivanjem originalne slike. Iz dobivenog skupa slika izbačene su one slike koje sadrže neku od bijelih krvnih stanica.



Sl. 4.6. Proces označavanja slika

```
1003,211 | /cellimages/upload20170906/leukociti_slike_20170906_87.jpg | 2| Segmentirani neutrofil
1012,1210 | /cellimages/upload20170906/leukociti_slike_20170906_34.jpg | 3| Limfocit
1012,412 | /cellimages/upload20170830/leukociti_slike_20170823_119.jpg | 2| Segmentirani neutrofil
1014,1349 | /cellimages/upload20170906/leukociti_slike_20170906_6.jpg | 3| Limfocit
1014,22 | /cellimages/upload20170906/leukociti_slike_20170906_87.jpg | 3| Limfocit
1017,1414 | /cellimages/upload20170906/leukociti_slike_20170906_21.jpg | 2| Segmentirani neutrofil
1021,1514 | /cellimages/upload20170906/leukociti_slike_20170906_170.jpg | 2| Segmentirani neutrofil
1023,1111 | /cellimages/upload20170906/leukociti_slike_20170906_60.jpg | 3| Limfocit
1026,1445 | /cellimages/upload20170906/leukociti_slike_20170906_54.jpg | 2| Segmentirani neutrofil
1028,901 | /cellimages/upload20170830/leukociti_slike_20170823_79.jpg | 2| Segmentirani neutrofil
```

Sl. 4.7. Nakon obrade spremaju se koordinate i tip označene stanice u tekstualnu (*.txt*) datoteku

Nakon što su sve slike označene i klasificirane dobivene su vrijednosti prikazane u tablici 4.8.:

Tab. 4.8. Broj i vrsta stanica u skupu podataka za učenje

Vrsta leukocita:	**Broj slika:**
Bazofili	3
Eozinofili	37
Limfociti	200
Monociti	129
Segmentirani neutrofili	369
Nepoznato	200

Nakon što su sve stanice označene, korištenjem Python skripte izrezuju se slike detektiranih stanica željene veličine s tipom stanice sadržanim u imenu slike. Ovako dobivene označene slike koriste se kao skup za učenje u postupku izgradnje klasifiktora temeljenog na konvolucijskoj neuronskoj mreži.

4.2. Segmentacija slika

4.2.1. Metode segmentacije slika

U računalnom vidu, segmentacija slike podrazumijeva dekompoziciju slike na njezine sastavne dijelove (regije) prema određenom kriteriju. Najčešće se koristi za lokalizaciju objekta i granica objekta (linije, krivulje, itd.) Pri tome segmentirana slika ima sljedeće željene karakteristike:

- Regije su uniformne s obzirom na neko svojstvo (npr. vrijednost točke ili tekstura)
- Granice regija moraju biti jednostavne
- Regije ne smiju imati male otvore
- Susjedne regije se moraju značajno razlikovati

Segmentacija slike nalazi svoju primjenu u mnogim područjima: analizi medicinskih slika (detekcija tumora, mjerenje volumena tkiva), detekciji objekata (detekcija pješaka, lica, prometnih znakova), obradi satelitskih snimaka (šume, ceste, usjevi), video nadzoru, kontroli prometa te u mnogim drugim područjima. Na Sl. 4.8. prikazan je primjer segmentacije slike raščlanjen u nekoliko koraka. Algoritam kombinira detekciju rubova s detekcijom regija kako bi izdvojio objekte na slici.

Metode segmentacije mogu se podijeliti u tri skupine:

1. Metoda praga (engl. *thresholding*) – izdvajanje cijelog objekta od pozadine
2. Detekcija rubova (engl. *edge-based*) – detekcija rubova koji odvajaju objekt od okoline
3. Detekcija regija (engl. *region-based*) – detekcija regija sličnih karakteristika

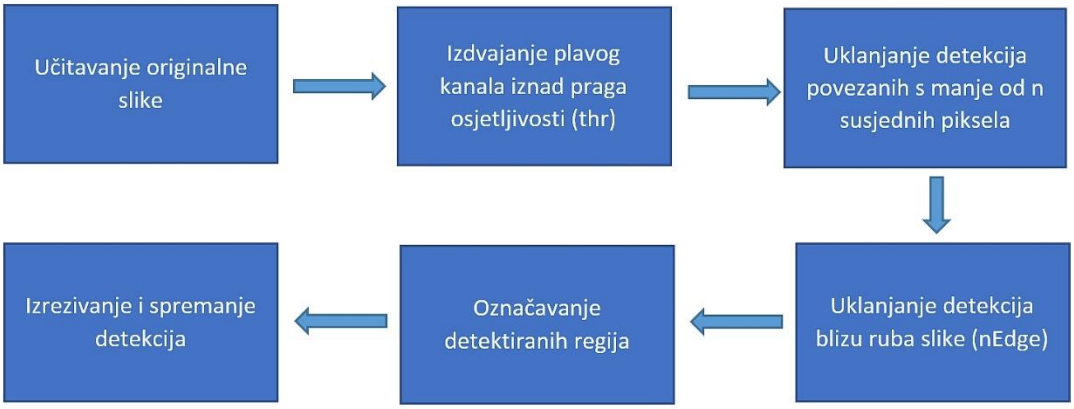
U ovom radu korištena je kombinacija metode praga i detekcije regija za segmentaciju bijelih krvnih stanica.



Sl. 4.8. Primjer segmentacije slike [9]

4.2.2. Algoritam segmentacije originalne ulazne slike

Prvi korak algoritma za detekciju i identifikaciju bijelih krvnih stanica je izdvajanje, tj. segmentiranje pojedinačnih bijelih krvnih stanica sa originalne slike kako bi jednom kada se klasifikatora zasnovan na konvolucijskoj neuronskoj mreži istrenira mogla napraviti identifikacija bijele krvne stanice. Procesom segmentacije se izdvajaju bijele krvne stanice te se svaki izdvojeni objekt u nastavku naziva detekcija. Osmišljen je algoritam koji se sastoji od nekoliko koraka obrade koji su prikazani na slici 4.9. U prvom koraku učitava se originalna slika krvnog uzorka. Budući da su bijele krvne stanice najbliže plavoj boji RGB slike, izdvaja se samo plavi kanal kako bi bile što više istaknute. Zatim se izdvajaju samo nakupine piksela s vrijednošću intenziteta većom od određene predefinirane vrijednosti praga. Na taj način na slici ostaju samo objekti istaknuti u plavom kanalu. Ipak, zbog nesavršenosti bojenja krvnih uzoraka i kvalitete same slike ponekad su detektirane manje nakupine piksela koje zapravo ne pripadaju bijelim krvnim stanicama. Zbog tog razloga odbacuju se nakupine piksela s manje od *n* povezanih susjeda. Osim toga, detekcije s ruba slike potencijalno mogu napraviti problem pri identifikaciji te se stoga i sve detekcije koje se nalaze na udaljenosti *nEdge* od ruba slike odbacuju. Preostale detektirane nakupine piksela se zatim označavaju te im se računa centroid, tj. koordinate centra detekcije. Na kraju se slike izrezuju na izračunatim koordinatama i spremaju na računalo. Veličina izrezanih slika je postavljena na 300 x 300 piksela što je određeno veličinom samih stanica, tj. povećanjem koje se koristi na mikroskopu. Ova veličina slika se pokazala kao optimalnom kako bi bili obuhvaćeni svi mogući tipovi i veličine stanica.



Sl. 4.9. Prikaz algoritma segmentacije slike

Algoritam se sastoji od sljedećih koraka:

1. Učitavanje originalne slike
   - učitavanje RGB slike u .png formatu
2. Izdvajanje plavog kanala iznad praga osjetljivosti (*thr*)
   - prag osjetljivosti određuje što će se smatrati detekcijom, a što ne
3. Uklanjanje manjih nepoveznih detekcija
   - uklanjanje komponenti koje su povezane s manje od *n* susjednih piksela
4. Uklanjanje detekcija blizu ruba slike
   - uklanjanje detekcija koje su samo djelomično vidljive (nalaze se na udaljenosti od 100 piksela od ruba slike)
5. Označavanje detektiranih regija
   - označavanje regija dobivenih u prethodnom koraku obrade
6. Izrezivanje i spremanje detekcija
   - izrezivanje slika veličine 300 x 300 piksela s detektiranim stanicama i spremanje na računalo

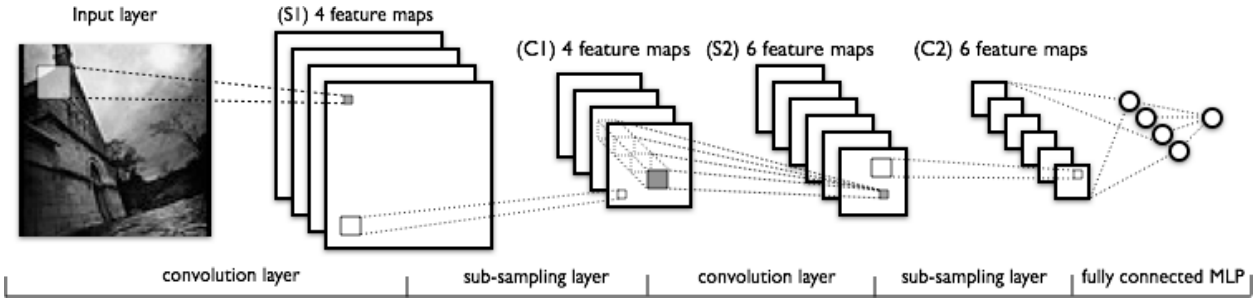
4.3. Izgradnja konvolucijske neuronske mreže za klasifikaciju bijelih krvnih stanica

4.3.1 Arhitektura konvolucijske neuronske mreže

Duboke neuronske mreže su mreže s mnogo skrivenih slojeva između ulaznog i izlaznog sloja. U ovom radu korištena je duboka neuronska mreža koja se temelji na konvoluciji (engl. *convolution*) i sažimanju (engl. *pooling*) slike. Na kraju mreže nalazi se potpuno povezani sloj (engl. *fully connected layer*) koji je zapravo perceptron. Mreže ovakvog tipa omogućuju izdvajanje značajki za klasifikaciju svakog pojedinog piksela. Ovakva arhitektura se pokazala u mnogim manje ili više složenim primjenama kao zadovoljavajuće rješenje te je stoga odabrana.

Jedna od prvih dubokih neuronskih mreža, koja je zapravo i pokrenula razvoj područja dubokog učenja bila je *LeNet* arhitektura. Temeljena je na pionirskom radu Yanna LeCuna koji je započeo rad na dubokim neuronskim mrežama 1988. godine. Nakon nekoliko pokušaja

tijekom godina, 1998. godine objavljen je rad u kojem je opisana arhitektura *LeNet5* duboke neuronske mreže [10]. U to vrijeme koristila se uglavnom za prepoznavanje znakova, poštanskih brojeva i znamenki. Iako su u međuvremenu razvijene neke poboljšane verzije dubokih mrežnih arhitektura, sve one se uglavnom temelje na konceptima koje koristi *LeNet*. Na slici 4.10. prikazana je arhitektura *LeNet* duboke neuronske mreže.



Sl. 4.10. *LeNet* duboka mrežna arhitektura [11]

Četiri su glavne operacije u gore opisanoj mreži:

1. Konvolucijski sloj
2. Nelinearnost (ReLU)
3. Sloj sažimanja
4. Klasifikacija (potpuno povezani sloj)

Na ulazu u mrežu može biti monokromatska slika ili višekanalna slika u boji. Nakon toga slijede slojevi konvolucije (detaljnije objašnjeno u točki 3.1.1.) i slojevi sažimanja (detaljnije objašnjeno u točki 3.1.2.). Na samom kraju nalazi se nekoliko potpuno povezanih slojeva (klasični perceptron) koji su dvodimenzionalni, uključujući i izlazni sloj (slika 3.2.). Duboka neuronska mreža obično ima desetak slojeva. Konolucijski slojevi i slojevi sažimanja imaju dvodimenzionalne „neurone“ koji se nazivaju mapama značajki (engl. *feature maps*). Mape značajki u svakom sljedećem sloju postaju sve manjih i manjih dimenzija. Konvolucijski slojevi su povezani s prethodnim slojevima te učenjem podešavaju težine veza. Slojevi sažimanja ne uče nikakve težine nego su konstantni. ReLU (engl. *Rectified Linear Unit*) uvodi nelinearnost u model (detaljnije objašnjeno u 3.1.3.). Neka istraživanja su pokazala da korištenje *ReLU* aktivacijske funkcije znatno ubrzava treniranje mreže na jako velikim skupovima podataka [12]. Ovakav tip neuronskih mreža je lakše trenirati od ostalih dubokih mrežnih arhitektura te pronaći više značajki koje se propuštaju kroz filtere, tj. slojeve duboke konvolucijske neuronske mreže. Također, ovakav tip dubokih arhitektura zahtjeva relativno malo predobrade podataka. U nekim slučajevima se pokazalo da čak rade bolje i od čovjeka. No, još uvijek je relativno nepoznato što se zapravo događa unutar skrivenih slojeva mreže kao što je predstavljeno u točki 3.1.

4.2.2. Predobrada podataka

Jedan od glavnih ciljeva ovoga rada je implementacija konvolucijske neuronske mreže za klasifikaciju bijelih krvnih stanica. Kako je već navedeno u potpoglavlju 4.1.2. prikupljeno je ukupno 738 slika (3 bazofila, 37 eozinofila, 129 monocita, 200 limfocita i 369 segmentiranih neutrofila, 200 nepoznatih – objekti koji ne pripadaju niti jednoj od prethodno navedenih klasa). Budući da je bazofila samo 3 te čine manje od 0.01 % oni nisu uzeti u obzir pri implementaciji i treniranju mreže. Osim toga, izbačene su slike sa samog ruba, tj. one na kojim je bijela krvna stanica samo djelomično vidljiva. Kako je vidljivo u Tablici 4.1. dobiven je manji broj eozinofila i monocita te nekoliko stotina limfocita i segmentiranih neutrofila. Kod takvih nebalansiranih skupova podataka javlja se problem pristranosti modela (engl. *bias*), tj. događa se da model puno veću „težinu“ daje određenom objektu pri predikciji. Za treniranje je korišteno ukupno 168 slika (28 eozinofila, 34 limfocita, 34 monocita, 36 neutrofila, 36 objekata koji ne pripadaju niti jednoj klasi - nepoznato) upravo da bi izbjegli problem pristranosti. Dio slika za učenje modela prikazan je na slici 4.11.



Sl. 4.11. Dio baze slika za učenje modela

4.2.3. Korištene metode regularizacije

Prenaučenost se može spriječiti na nekoliko načina, a jedan od najčešće korištenih je povećanje varijacije uzorka za učenje. Skupljanje dodatnih uzoraka često je vrlo skupo, kako vremenski tako i novčano. Kako bi se riješio ovaj problem moguće je koristiti slučajne transformacije kao što su rotacija, translacija i skaliranje nad postojećim skupom podataka. U ovom radu korištene su sljedeće transformacije:

- **rotacija** – rotiranje slike za određeni raspon (u stupnjevima)
- **slučajni horizontalni pomak** – pomak za frakciju širine slike
- **slučajni vertikalni pomak** – pomak za frakciju visine slike
- **horizontalno okretanje** – slučajno okretanje slike

Osim proširenje originalnog skupa slika slučajnim transformacijama, u ovom radu je korišten i tzv. *Dropout* sloj koji je dodan u konvolucijsku neuronsku mrežu nakon slojeva konvolucije i sažimanja. Korišten je *Dropout* sloj koji nasumično „isključuje“ 50 % neurona tijekom postupka učenja.

4.2.4. Funkcija gubitka i učenje

Kao kriterijska funkcija korištena je *categorical_crossentropy* (Kategorička unakrsna entropija) koja je pogodna za višeklasnu klasifikaciju na temelju vjerojatnosti dobivenih *softmax* aktivacijskom funkcijom. Kategorička unakrsna entropija smatra se boljim izborom od srednje kvadratne pogreške budući da srednja kvadratna pogreška daje preveliku težinu neispravnim ulazima. Osim toga, kako se povećava vjerojatnost ispravne klasifikacije određenog uzorka, to je korekcijski faktor koji podešava težine manji te vrijeme treniranja može potrajati. Za minimizaciju kriterijske funkcije korištena je *RMSprop* metoda koja slično kao i stohastički gradijentni spust koristi male lokalne pomake i adaptivnu stopu učenja da bi podesila težine.

5. IMPLEMENTACIJA PREDLOŽENOG RJEŠENJA U PROGRAMSKOM JEZIKU PYTHON

5.1. Programski jezik Python

Algoritam je implementiran u programskom jeziku Python [13]. Python je programski jezik opće namjene, visoke razine koji podržava objektno-orijentirano programiranje. Stvorio ga je Guido von Rossum 1990. godine. Zbog svoje jednostavnosti, ali i velikog broja biblioteka sve je popularniji među programerima. Pogodan je za početnike, ali i za rješavanje vrlo složenih problema (koriste ga Google, NASA). Njegov dizajn naglašava čitljivost koda, a sintaksa omogućava programerima da izraze koncepte u puno manje linija koda nego što bi to bilo moguće u drugim jezicima kao što su C++ ili Java. Jedan je od najpopularnijih jezika u brzorastućem području strojnog učenja i obrade velikih količina podataka za što je razvijen veliki broj biblioteka. Dostupnost velikog broja biblioteka i gotovih rješenja omogućava izradu složenijih aplikacija u vrlo kratkom vremenu. Stoga je odabran upravo Python za implementaciju svih algoritama ovog rada.

5.1.1. Python scikit-image biblioteka

Za obradu slike korištena je *scikit-skimage* [14] biblioteka u kojoj su implementirani algoritmi obrade slike [15]. Biblioteka je izdana pod modificiranom BSD licencom otvorenog koda, zajedno s vrlo dobrom dokumentacijom. Razvijena je od strane međunarodnog tima programera s ciljem da biblioteka bude pogodna za korištenje kako u znanstveno-istraživačkim projektima tako i u industriji. *Scikit-image* manipulira slikama u obliku standardnog Python NumPy polja. Neke od funcija i algoritama implementiranih u ovoj biblioteci su:

- funkcije za ulaz i izlaz podataka
- transformacija prostora boja
- crtanje
- detekcija i izdvajanje značajki
- filtriranje slika
- morfološke operacije i segmentiranje

5.1.2. Programska biblioteka *Keras*

Keras je programska biblioteka otvorenog koda napisana u programskom jeziku Python. Visoke je razine i namijenjena je dizajniranju i treniranju dubokih neuronskih mreža. Kao pozadinski kod može koristiti *TensorFlow, CNTK (Microsoft Cognitive Toolkit) ili Theano*. Razvijena je kao dio projekta ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot*

Operating System), a glavni autor joj je Googleov inženjer Francois Chollet. 2017. godine Googleov *TensorFlow* tim odlučio je dodati podršku za *Keras* unutar *TensorFlow* jezgre. Glavne značajke Kerasa su: jednostavno korištenje, modularnost, proširivost, rad s Python programskim jezikom [16].

5.2. Implementacija algoritma segmentacije slike

Algoritam je implementiran u programskom jeziku Python korištenjem *scikit-image* biblioteke za obradu slike [16], dok je za prikaz dobivenih rezultata korištena *matplotlib* biblioteka [17]. U prvom koraku definirani su parametri algoritma za segmentaciju slike koje je predstavljen u potpoglavlju 4.2.2.. Parametar *thr* definira vrijednost praga svjetline piksela iznad kojeg detekcije smatramo valjanima te ih uzimamo u obzir u sljedećem koraku obrade, *nMin* definira minimalan broj piksela unutar nakupine piksela koje se prihvaćaju kao detekcija, *nNeigh* definira minimalan broj piksela s kojima je svaka detekcija povezana da bi se smatrala valjanom te *nEdge* definira udaljenost od ruba slike u pikselima koji se ne uzima u obzir prilikom daljnje obrade.

```
# algorithm parameters

# threshold
thr = 30
# minimum number of detections in one region
nMin = 1500
# number of neighbours of one region
nNeigh = 10
# number of pixels close to the edge
nEdge = 100
```

Glavni dio algoritma segmentacije podijeljen je u četiri koraka. Budući da su objekti od interesa najbliži plavom kanalu, u prvom koraku se izdvaja samo plavi kanal od učitane slike te se dobiva slika *bPlane*. U drugom koraku algoritam odabire nakupine piksela čija je vrijednost iznad predefiniranog praga *thr* kako bi ostavili samo objekte koji se jako ističu u plavom kanalu na slici. U trećem koraku koristi se funkcija *morphology.remove_small_objects()* kako bi se uklonile detekcije koje sadrže manje od *nMin* detekcija te koje su povezane s manje od *nNeigh* susjeda. U četvrtom koraku uklanjaju se detekcije koje su jako blizu ruba, tj. uklanjaju se sve one koje su na manje od *nEdge* piksela od ruba. Ovaj korak je nužan zbog toga što takve nepotpune detekcije mogu stvarati problem prilikom treniranja modela, kao i kod dimenzioniranja slike budući da će nedostajati jedan dio. Kao krajnji rezultat dobiva se binarna slika na kojoj su s 1 označeni segmentirani objekti, tj. bijele krvne stanice.

```
# extract blue channel
bPlane = img_rgb[:,:,2]  - 0.5*(img_rgb[:,:,0]) - 0.5*(img_rgb[:,:,1]);

```

```
# detect everything above threshold
BW = bPlane > thr;

# remove small objects
cleanSmall = morphology.remove_small_objects(BW, min_size=nMin, connectivity=nNeigh)

# remove detections close to the edge
cleanEdge = clear_border(cleanSmall, nEdge)
```

Nakon što je dobivena segmentirana binarna slika, treba pronaći koordinate segmentiranih dijelova, izdvojiti ih te zasebno snimiti izrezane slike detektiranih stanica. Za označavanje detektiranih regija koristi se *skimage.measeure.label* funkcija. Označena slika se zatim predaje funkciji *skimage.measure.regionprops* koja vraća izmjerena svojstva slike kao što su broj piksela određene regije, koordinate centra detekcije, minimalnu i maksimalnu vrijednost piksela itd. Za određivanje koordinata detekcije koristi se svojstvo *centroid*. Na dobivenim koordinatama izrezuje se slika dimenzija 300 x 300 piksela te spremamo svaku od njih. Osim toga spremamo i koordinate detekcija u tekstualnu datoteku *coordinates.txt*.

```
# mark regions on image
label_img = label(cleanEdge)
props = regionprops(label_img)

coords = open("coordinates.txt", "w")

# cut and save detections
for region in props:
    r0, c0, r1, c1 = region.bbox
    x,y = region.centroid
    img_crop = img_rgb[r0-200:r1+200, c0-200:c1+200]
    img_resize = resize(img_crop, (300, 300))
    plt.imsave(".../crop" + str(i) + ".png", img_resize)
    coords.write(str(x) + " " + str(y) +  "\n")
    i+=1
coords.close()
```

5.3. Implementacija konvolucijske neuronske mreže

5.3.1. Predložena arhitektura modela za klasifikaciju bijelih krvnih stanica

U svrhu klasifikacije bijelih krvnih stanica korištena je blago modificirana verzija *LeNet* duboke mrežne arhitekture opisana u potpoglavlju 4.3.1. Prije definiranja arhitekture mreže potrebno je definirate broj klasa (*nClasses*) koji iznosi 5 budući da se slike klasificiraju u 5 klasa (limfociti, monociti, eozinofili, segmentirani neutrofili, nepoznato). Veličina slike je postavljena na 300 x 300 piksela, a kako se radi o RGB slikama broj kanala je postavljen na 3. Definirani model je sekvencijalan, što znači da su slojevi postavljeni redom jedan iza drugoga. Definirane su tri operacije konvolucije i tri operacije sažimanja slike. Konvolucija koristi jezgru veličine 3 x 3. Sloj sažimanja koristi jezgru veličine 2 x 2, uzimajući maksimalnu vrijednost unutar jezgre. U svakom od slojeva kao aktivacijska funkcija se koristi *ReLU*. Svojstva i uloga *ReLU* aktivacijske funkcije objašnjena je u poglavlju 2.1.3. Bitno je napomenuti da *ReLU* ulaznu vrijednost manju

od nule postavlja na nulu, dok vrijednost veću od nule propušta. Nakon operacije konvolucije i sažimanja slijedi transformacija 3D slike značajki u 1D vektor značajki pomoću funkcije *Flatten()*. Nakon nje slijedi sloj u kojem je svaki neuron spajamo sa svakim neuronom u sljedećem sloju korištenjem funkcije *Dense()*.

Kako bi se izbjegao problem prenaučenosti modela dodajemo tzv. *Dropout* sloj opisan u prethodnom poglavlju. Sve operacije vezane za *Dropout* biblioteka *Keras* odrađuje u pozadini, dovoljno je u model dodati *Droput()* s argumentom kojim definiramo postotak neurona koji će biti deaktivirani. Korištena je vrijednost 0.5, tj. polovica svih neurona nasumično se deaktiviraju tijekom procesa učenja.

U posljednjem sloju koristi se *Dense(nClasses)* funkcija koja sve neurone iz prethodnog sloja spaja s 5 izlaznih klasa objekata koje je potrebno klasificirati. Kao aktivacijska funkcija u ovom sloju koristi se *softmax*. Ova aktivacijska funkcija je zapravo poopćenje logističke funkcije za slučaj kada postoji više od dvije klase ($K > 2$). Funkcija preslikava ulaznu vrijednost u raspon [0, 1]. Time se zapravo dobiva vjerojatnost pripadnosti određene ulazne slike određenoj klasi što je ujedno i izlaz iz mreže.

```
# broj klasa
nClasses = 5
# visina i sirina slile
imgWidth = 300
imgHeight = 300
# broj kanala slike - RGB
nChannels = 3
```

```
def getModel():
    model = Sequential()
    model.add(Lambda(lambda x: x/127.5 - 1.,input_shape=(imgWidth,imgHeight,
    nChannels), output_shape=(imgWidth, imgHeight, nChannels)))

    # Konvolucija i sažimanje #1
    model.add(Conv2D(32, (3, 3),input_shape=(imgWidth,imgHeight,nChannels)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Konvolucija i sažimanje #2
    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Konvolucija i sažimanje #3
    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

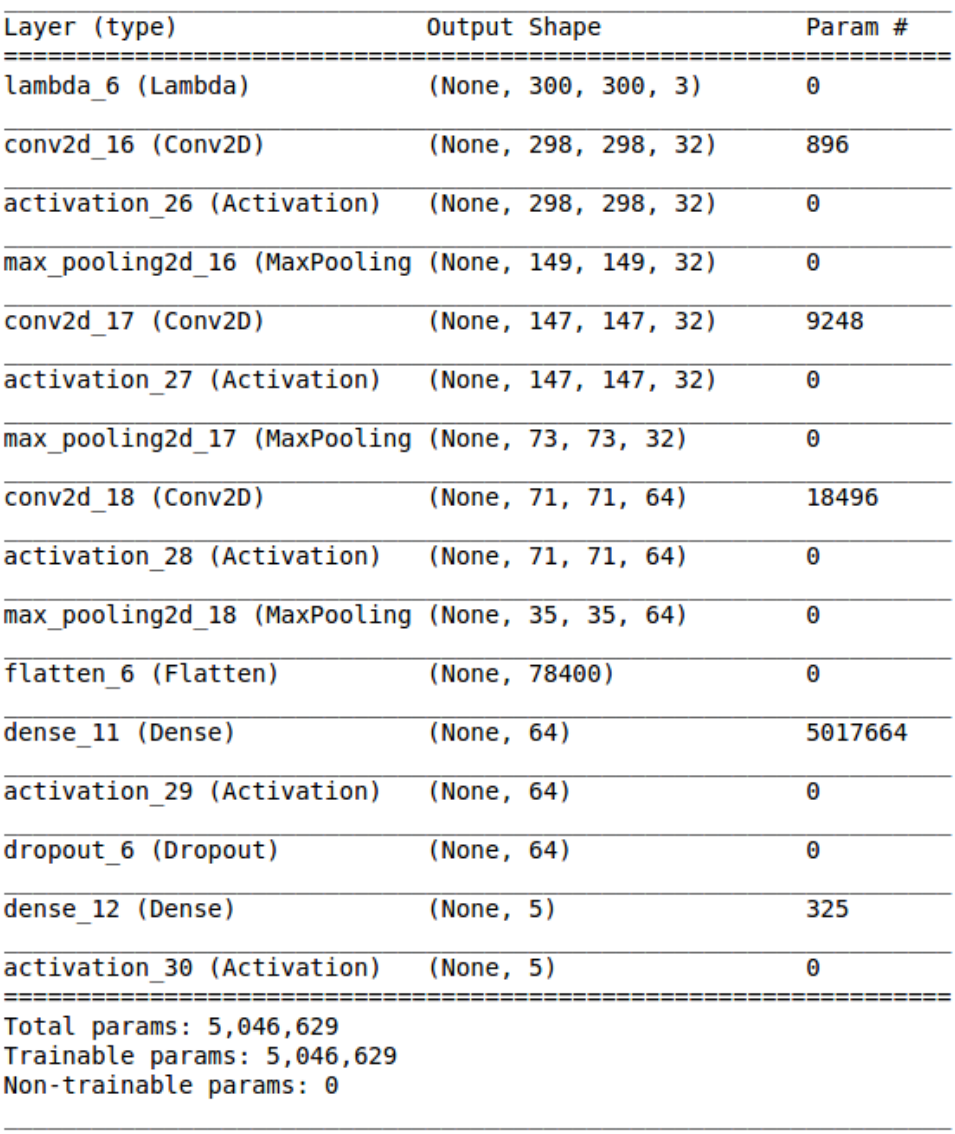
    # Transformacija 3D slike značajki u 1D vektor značajki
    model.add(Flatten())
    model.add(Dense(64))
    model.add(Activation('relu'))

    # Dodavanje dropouta - broj ulaza koji se odbacuje
```

```
27.     model.add(Dropout(0.5))
28.     model.add(Dense(nClasses))
29.     model.add(Activation('softmax'))
30.
31.     # Konfiguriranje modela
32.     model.compile(loss='categorical_crossentropy',
33.                   optimizer='rmsprop',
34.                   metrics=['accuracy'])
35.
36.     return model
```

Prije nego se pokrene proces treniranja mreže potrebno je konfigurirati model, tj. definirati kriterijsku funkciju (*loss*), algoritam optimizacije (*optimizer*) te metodu vrednovanja dobivenih rezultata (*metrics*). Metoda vrednovanja služi da bi se ocjenila performanse izgrađenog modela pomoću neke od kriterijskih funkcija. Kao kriterijska funkcija korištena je metoda *categorical_crossentropy* koja je optimizirana *rmsprop* postpupkom optimizacje (potpoglavlje 4.2.4.).

Nakon što je model kofiguriran, funkcija *summary()* vraća arhitekturu modela, tj. broj i vrstu slojeva (*Layer (type)*), izlaz iz svakog sloja (*Output Shape*) te broj parametara koje je potrebno podesiti u svakom sloju (*Param #*). Ukupan broj parametara mreže koje treba podesiti je 5 046 629.



Layer (type)	Output Shape	Param #
lambda_6 (Lambda)	(None, 300, 300, 3)	0
conv2d_16 (Conv2D)	(None, 298, 298, 32)	896
activation_26 (Activation)	(None, 298, 298, 32)	0
max_pooling2d_16 (MaxPooling	(None, 149, 149, 32)	0
conv2d_17 (Conv2D)	(None, 147, 147, 32)	9248
activation_27 (Activation)	(None, 147, 147, 32)	0
max_pooling2d_17 (MaxPooling	(None, 73, 73, 32)	0
conv2d_18 (Conv2D)	(None, 71, 71, 64)	18496
activation_28 (Activation)	(None, 71, 71, 64)	0
max_pooling2d_18 (MaxPooling	(None, 35, 35, 64)	0
flatten_6 (Flatten)	(None, 78400)	0
dense_11 (Dense)	(None, 64)	5017664
activation_29 (Activation)	(None, 64)	0
dropout_6 (Dropout)	(None, 64)	0
dense_12 (Dense)	(None, 5)	325
activation_30 (Activation)	(None, 5)	0

Total params: 5,046,629
Trainable params: 5,046,629
Non-trainable params: 0

Sl. 5.1. Arhitektura konvolucijske neuronske mreže

5.3.2. Proširenje skupa slika za učenje slučajnim transformacijama

Transformacije originalnog skupa podataka implementirane su u programskom jeziku *Python* korištenjem već spomenute biblioteke *Keras* [16]. Keras sadrži funkciju *ImageDataGenerator* koja u sebi ima implementirano nekoliko navedenih transformacija. Također je moguće generirati slike (engl. *batch*) u stvarnom vremenu tijekom treniranja konvolucijske neuronske mreže. Generiranje slika traje sve dok traje i treniranje.

U nastavku je prikazan dio *Python* koda gdje su definirane slučajne transformacije:

```
1.  datagen = ImageDataGenerator(
2.      rotation_range=20,
3.      width_shift_range=0.2,
4.      height_shift_range=0.2,
5.      horizontal_flip=True)
```

Na slici 5.2. prikazane su korištene slučajne transformacije na primjeru jedne slike. Korišteni su sljedeći argumenti funkcije:

rotation_range – rotiranje slike za 20°
width_shift_range – slučajni horizontalni pomak za 20 % ukupne širine slike
height_shift_range – slučajni vertikalni pomak za 20 % ukupne visine slike
hortizontal_flip – slučajno okretanje slike u horizontalnom smjeru



Sl. 5.2. Slučajne transformacije slike

5.3.3. Učenje i odabir optimalnog modela

Uobičajena praksa u području strojnog učenja je raspodjela podataka na skup za učenje (trening skup), skup za validaciju i skup za testiranje. Trening skup se koristi za određivanje optimalnih parametara mreže, dok se na validacijskom skupu provjeravaju generalizacijska svojstva mreže. Učenje se zaustavlja kada se pogreška prestane znatno mijenjati i na skupu za učenje i na validacijskom skupu što znači da model više ne može apsorbirati nove informacije. U slučaju kada se pogreška na skupu za učenje smanjuje, a na validacijskom skupu počinje rasti model se počinje previše prilagođavati skupu za učenje, tj. dolazi do prenaučenosti modela.

```
1.  # podjela podataka na trening i test skup
2.  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1
    0)
```

Prije samog postupka učenja neuronske mreže potrebno je definirati neke od parametara postupka učenja. Parametar *epochs* (broj epoha) određuje koliko će se puta svi podaci za učenje propagirati kroz mrežu. Unutar jedne epohe svi trening podaci prolaze od ulaza do izlaza mreže (engl. *forward pass*) te od izlaza do ulaza (engl. *backward* pass). Parametar *batchSize* određuje koliko će se trening uzoraka propagirati u jednoj epohi.

Programska biblioteka *Keras* omogućava spremanje težina modela nakon svake epohe korištenjem tzv. kontrolnih točaka (engl. *checkpoints*). Nakon svake epohe težine se spremaju s podatkom o pogreški na validacijskom skupu. Težine se spremaju u *.h5* formatu korištenjem *h5py* biblioteke [18]. Ovaj format omogućava spremanje velikih količina numeričkih podataka te jednostavno manipuliranje njima iz *Python NumPy* biblioteke. Unutar samo jedne ovakve

datoteke moguće je spremiti nekoliko stotina označenih skupova podataka te ih vrlo jednostavno ponovno koristiti.

```
# izgradnja modela i spremanje težina nakon svake epohe
model = getModel()
checkpoint = ModelCheckpoint(filepath='./checkpoints/checkpoint-{epoch:02d}-
{val_loss:.2f}.h5')
```

Ne postoje metode koje mogu točno odrediti da je jedna arhitektura bolja od druge, ali generalno postoje određene smjernice. Dobro je koristiti što više konvolucijskih slojeva kako bi se mogle detektirati što apstraktnije značajke. Veličina jezgre ne smije biti ni prevelika niti premala za dani problem.Veličina jezgre u sloju sažimanja ne smije biti prevelika kako se ne bi izgubilo previše informacija. Isto tako ne smije biti ni premala jer onda nema nikakvog efekta, tj. nema efikasnog izvlačenja značajki. Obično se sve ove vrijednosti određuju eksperimentalno, tj. isprobavanjem različitih arhitektura i njihovih parametara.

Nakon što je definirana arhitektura mreže i svi njeni parametri, moguće je pokrenuti sam proces treniranja mreže. Funkcija *flow()* u beskonačnoj petlji dohvaća slike od objekta *datagen* koji generira slike na koje su primjenjene slučajne transformacije. U programskoj biblioteci *Keras* postoji funkcije *fit_generator* kojem se kao argumenti prosljeđuju podaci za učenje, podaci za validaciju, broj koraka po epohi i pozivi funkciji (engl. *callbacks*) tijekom procesa učenja kako bi se spremile kontrolne točke s težinama nakon završetka svake epohe. U nastavku je prikazan programski kod koji pokreće proces učenja mreže te na kraju sprema model i njegove težine. Na slici 5.3. prikazan je ispis za prvih 5 epoha s pripadnom točnošću i vrijednošću funkcije gubitka.

```
epochs = 20
batchSize = 32

train_generator = datagen.flow(
        X_train,
        y_train,
        batch_size=batchSize)

validation_generator = datagen.flow(
        X_test,
        y_test,
        batch_size=batchSize)


# učenje mreze sa slučajnim transformacijama u stvarnom vremenu
model.fit_generator(
    train_generator,
    steps_per_epoch=len(X_train),
    validation_data=validation_generator,
    validation_steps=len(X_test),
    epochs=epochs,
    callbacks=[checkpoint])

# spremanje modela i spremanje tezina modela
model.save('trained_model.h5')
```

```
26. model.save_weights('trained_weights.h5')
```

```
Epoch 1/20
105/105 [==============================] - 348s - loss: 0.7424 - acc: 0.7236 - val_loss: 1.4220 - val_acc: 0.4938
Epoch 2/20
105/105 [==============================] - 325s - loss: 0.5359 - acc: 0.8075 - val_loss: 1.2070 - val_acc: 0.7064
Epoch 3/20
105/105 [==============================] - 325s - loss: 0.3971 - acc: 0.8720 - val_loss: 1.5078 - val_acc: 0.6914
Epoch 4/20
105/105 [==============================] - 328s - loss: 0.2864 - acc: 0.9020 - val_loss: 1.1358 - val_acc: 0.6886
Epoch 5/20
105/105 [==============================] - 324s - loss: 0.2638 - acc: 0.9005 - val_loss: 1.2854 - val_acc: 0.6790
```

Sl. 5.3. Učenje konvolucijske neuronske mreže za prvih 5 epoha

5.3.4. Predikcija na temelju izgrađenog modela i prikaz rezultata

Nakon završetka procesa učenja i odabira optimalnog modela potrebno je testirati i vrednovati dobiveni model na novim podacima. *Keras* omogućuje jednostavno učitavanje težina modela sljedećom naredbom:

```
1. # Izgradnja modela i učitavanje težina
2. model = getModel()
3. model.load_weights('trained_weights.h5')
```

Nakon što su učitane težine, pomoću *cv2.imread* se učitava slika *img_fname*. Slika se zatim konvertira u *NumPy* polje te se kao takva predaje modelu. Funkcija *model.predict()* provlači sliku kroz mrežu te na izlazu daje vjerojatnost pripadnosti jednoj od 5 klasa. Funkcijom *rint()* zaokružujemo vrijednosti na najbliži cijeli broj te dobivamo izlaz kao na slici 5.4.

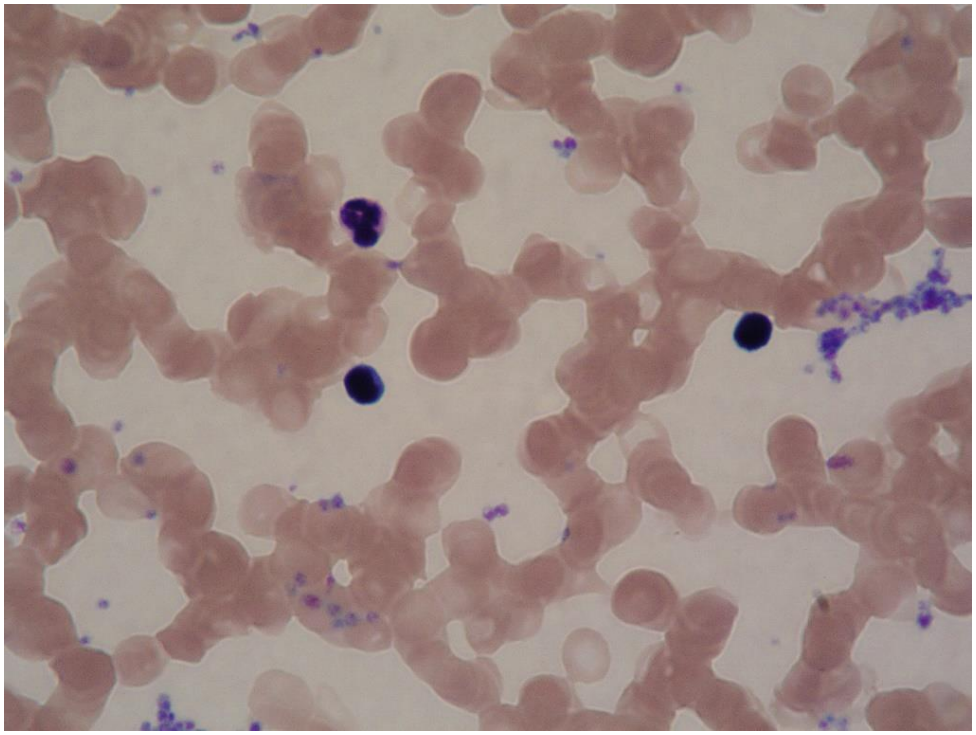
```
1. # Make predictions on new (segmented) image
2.
3. X = []
4. img_file = cv2.imread(img_fname)
5. img_arr = np.asarray(img_file)
6. X.append(img_arr)
7. X = np.asarray(X)
8. y_pred = np.rint(model.predict(X))
9.
10. if (y_pred[0][0] == 1):
11.     print 'Eozinofil'
12. if (y_pred[0][1] == 1):
13.     print 'Limfocit'
14. if (y_pred[0][2] == 1):
15.     print 'Monocit'
16. if (y_pred[0][3] == 1):
17.     print 'Segmentirani neutrofil'
18. if (y_pred[0][4] == 1):
19.     print 'Nepoznato'
```

```
CNN predictions:

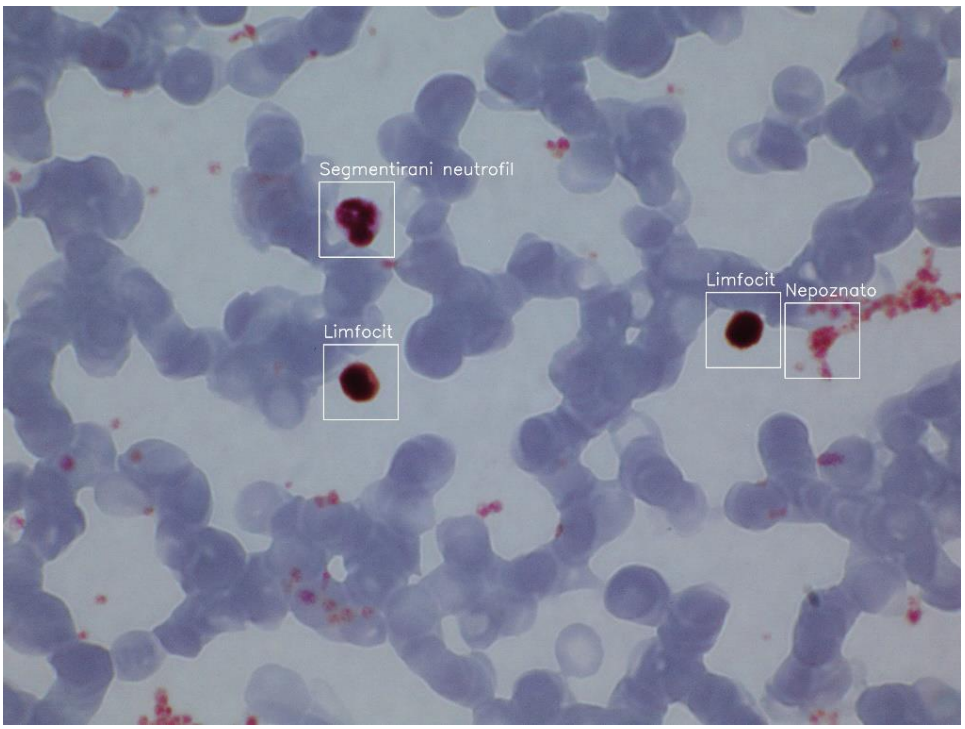
[[ 1.  0.  0.  0.  0.]]
Eozinofil
[[ 1.  0.  0.  0.  0.]]
Eozinofil
[[ 0.  0.  0.  1.  0.]]
Segmentirani neutrofil
[[ 0.  1.  0.  0.  0.]]
Limfocit
```

Sl. 5.4. Primjer izlaza mreže za četiri ulazne slike

Posljednji korak obrade je označavanje detektiranih i identificiranih stranica na originalnoj slici dodavanjem okvira (engl. *bounding box*) i naziva stanice. Okviri su dodani funkcijom *cv2.rectangle()*, dok je tekst dodan funkcijom *cv2.putText()*. Na slici 5.5. prikazana je originalna ulazna slika te na slici 5.6. krajnji rezultat obrade s detektiranim i identificiranim stanicama.



Sl. 5.5. Originalna ulazna slika



Sl. 5.6. Krajnji rezultat obrade s detektiranim i identificiranim bijelim krvnim stanicama

6. REZULTATI

6.1. Testiranje i vrednovanje algoritma segmentacije slika

Zbog različitih uvjeta koji utječu na segmentaciju slike, kao što su osvjetljenje slike te kvaliteta bojenja krvnog razmaza, potrebno je eksperimentalno odrediti optimalne parametre algoritma segmentacije. Implementirani algoritam prvo je testiran na 10 slika kako bi se odredili optimalni parametri algoritma te je zatim testiran na novih 20 slika u svrhu evaluacije algoritma. U tablici 6.1. prikazane su vrijednosti korištenih parametara.

Tab. 6.1. Eksperimentalno određene vrijednosti parametra algoritma segmentacija slike

Parametar	**Vrijednost**
thr	30
nMin	1500
nNeigh	10
nEdge	100

Algoritam je zatim testiran na 10 slika te su rezultati predstavljeni matricom zabune (engl. *confusion matrix*). Općenito, algoritmi segmentacije se obično evaluiraju tako da se odredi koliko piksela na slici je ispravno segmentirano, tj. koliko dobro su određene granice objekta na slici. Međutim, budući da predloženi algoritam segmentacije radi s konvolucijskom neuronskom mrežom, nije toliko važno da su granice jako precizno određene nego je važno da se objekt i dio njegove okoline nalazi na slici. Iz tog razloga izabrana je matrica zabune za evaluaciju.

Matrica zabune predstavlja jednu od standardnih mjera evaluacije rezultata klasifikacije. Prikazna je u obliku tablice gdje redak predstavlja uzorak predviđene klase, dok stupac predstavlja stvarnu klasu. Pozitivna klasa uzorka označena je s 1, dok je negativna klasa uzorka primjera označena s 0 [19]. Na slici 6.1. prikazana je matrica zabune.



Sl. 6.1. Matrica zabune

Ako je predviđna klasa uzorka 1 i stvarna klasa uzorka je 1, tada je taj par *true positive (TP).*
Ako je predviđna klasa uzorka 1, a stvarna klasa uzorka je 0, tada je taj par *false positive (FP).*
Ako je predviđena klasa uzorka 0, a stvarna klasa uzorka je 1, tada je taj par *false negative (FN).*
Ako je predvđena klasa uzorka 0 i stvarna klasa uzorka je 0, tada je taj par *true negative (TN).*

Iz matrice zabune izvode se mjere koje su standard pri evaluacija algoritama u strojnom učenju:
Preciznost (engl. *precision*):

$$P = \frac{TP}{TP + FP} \tag{6-1}$$

Odziv (engl. *recallI):*

$$R = \frac{TP}{TP + FN} \tag{6-2}$$

Točnost (engl. *accuracy)*:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{6-3}$$

F1 mjera (engl. *F1 score*):

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{6-4}$$

Nakon testiranja algoritma segmentacije dobivena je sljedeća matrica zabune dana u tablici 6.2. Na 20 slika korištenih za testiranje algoritma segmentacije ukupno je označeno 30 bijelih krvnih stanice „ručnim“ pregledavanjem slika od strane osobe obučene za pregledavanje slika.

Tab. 6.2. Matrica zabune algoritma segmentacije slike

	Stvarna klasa	
Predviđena klasa	Objekt na slici	Objekt nije na slici
Objekt detektiran	29	3
Objekt nije detektiran	1	0

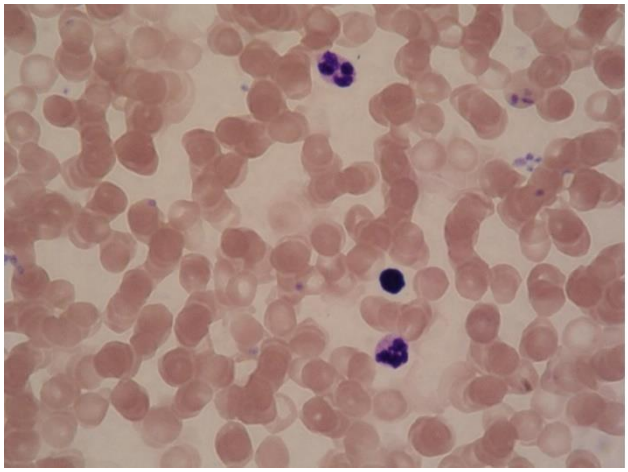
U tablici 6.3. prikazane su izračunate mjere evaluacije izvedene iz matrice zabune prema izrazima ((6-1) do (6-4)). Od ukupno 30 bijelih krvnih stanica, algoritmom segmentacije slike detektirano je njih 29. Nije detekirana 1 stanica, dok je detektirano i 3 objekta koji ne pripadau krvnim stanicama. Vrlo je važno da *False negative* uzorak bude što manji jer je vrlo bitno detektirati sve stanice na slici. Takvi uzorci koji su detektirani, a zapravo ne pripadaju niti jednoj klasi obrađivat će se i na sljedećoj razini gdje konvolucijska neuronska mreža klasificira i takve

uzorke i može im dodijeliti klasu *nepoznato*. Zbog zahtjeva da *False negative* bude što manji, pojavljivat će se više *False positive* detekcija. Međutim, nije dobro da ni ovakvih detekcija bude previše jer mreža u tom slučaju mora klasificirati znatno veći broj ulaznih slika.

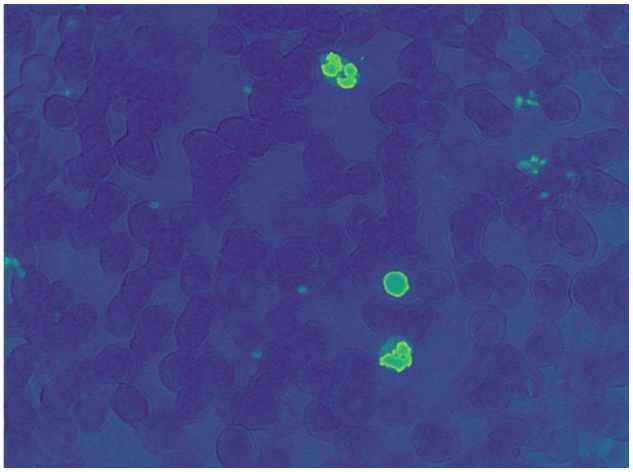
Tab. 6.3. Mjere evaluacija algoritma segmentacije slike

Preciznost	90.62 %
Odziv	96.66 %
Točnost	87.87 %
F1 mjera	93.55 %

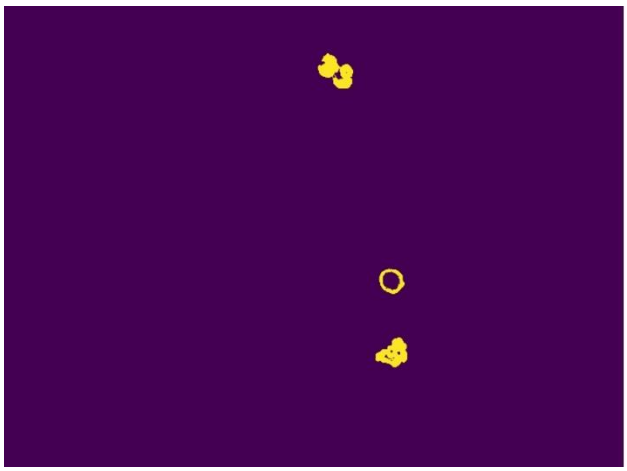
U nastavku je prikazan primjer ispravne segmentacije zajedno sa svim dobivenim slikama u procesu obrade. Na slici 6.1. a) prikazana je originalna ulazna slika, slika b) prikazuje izdvojeni plavi kanal na kojem su bijele krvne stanice puno istaknutije. Na slici 6.1. c) prikazan je rezultat uklanjanja manjih nakupina piksela te objekata jako blizu ruba slike (ako ih ima). Krajnji rezultat obrade je prikazan na slici d), dok su na slici e) prikazane izrezane detekcije koje se proslijeđuju konvolucijskoj neuronskoj mreži.



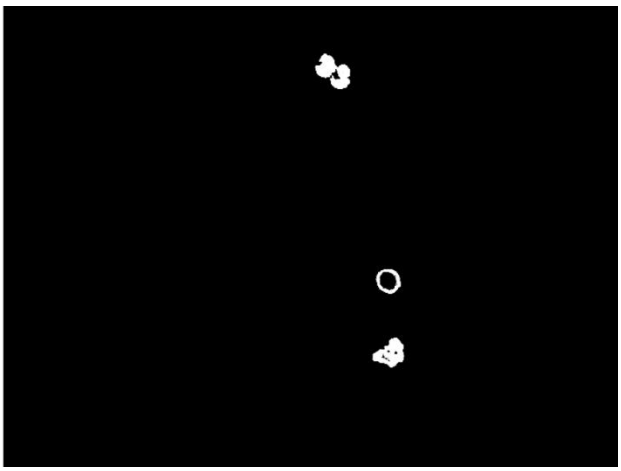
a) Originalna slika



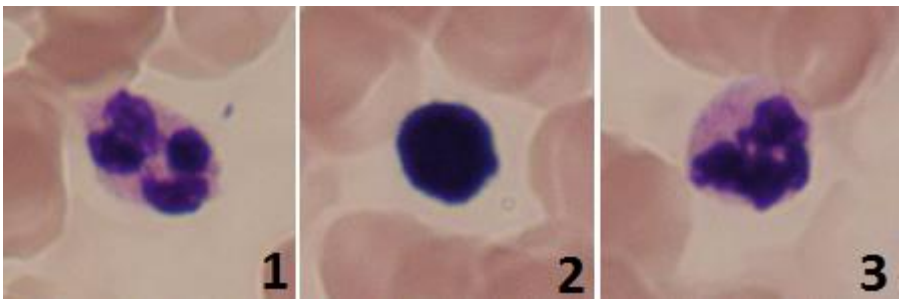
b) Izdvojeni plavi kanal



c) Uklanjanje manjih nakupina piksela i objekata s ruba slika



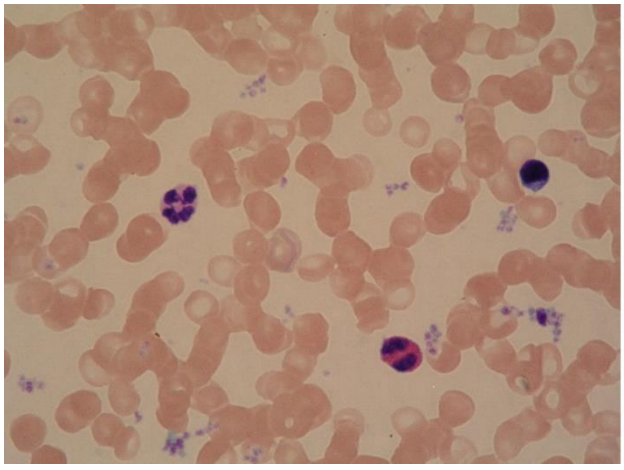
d) Segmentirana slika



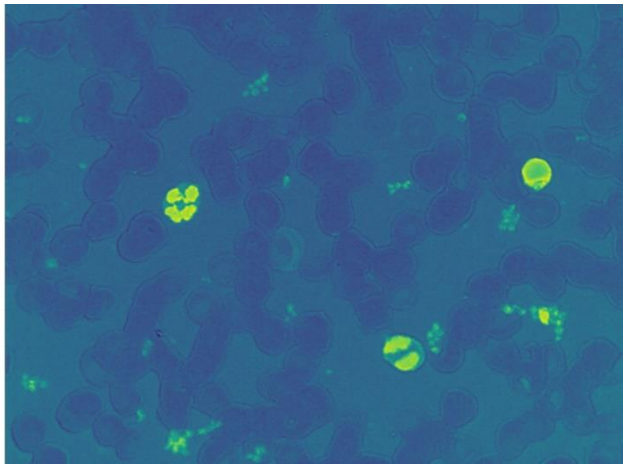
e) Detektirane bijele krvne stanice

Sl. 6.2. Primjer rezultata algoritma segmentacije

U nastavku je prikazan jedan primjer neispravne segmentacija slike prikazan je na slici 6.2. U ovom slučaju detektirana je i jedna nakupina piksela koja ne pripada niti jednoj bijeloj krvnoj stanici. Neispravna detekcija prikazana je na slici 6.2. e) broj 3.



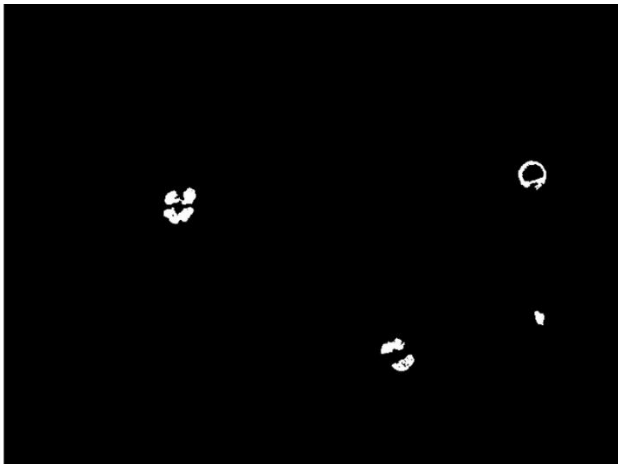
a) Originalna slika



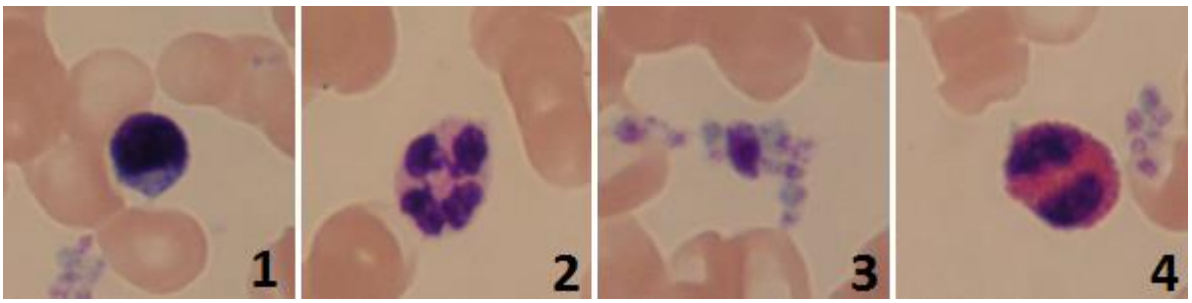
b) Izdvojeni plavi kanal



c) Uklanjanje manjih nakupina piksela i objekata s ruba slika



d) Segmentirana slika



e) Detektirane bijele krvne stanice

Sl. 6.3. Primjer rezultata algoritma segmentacije

6.2. Testiranje i vrednovanje izgrađenog modela konvolucijske neuronske mreže

Rad izgrađenog modela konvolucijske neuronske mreže za klasifikaciju bijelih krvnih stanica testiran je na 85 slika koje su ranije odvojene te koje nisu korištene za učenje modela. Zbog jako malog broja slika eozinofila, za testiranje su korištene samo 3 slike koje su bile dostupne. Ostale klase su sadržavale 22 limfocita, 18 monocita, 22 segmentirana neutrofila i 20 slika na kojima se ne nalazi niti jedna od navedenih klasa, tj. slučaj kada je slika krivo segmentirana. U tablici 6.4. prikazana je matrica zabune klasifikatora bijelih krvnih stanica.

Tab. 6.4. Matrica zabune klasifikatora.

	Stvarna klasa				
Predviđena klasa	Eozinofil	Limfocit	Monocit	Segmentirani neutrofil	Nepoznato
Eozinofil	3	3	0	9	0
Limfocit	0	17	1	0	0
Monocit	0	2	17	1	0
Segmentirani neutrofil	0	0	0	12	0
Nepoznato	0	0	0	0	20

Slično kao i za slučaj segmentacije slike, dobivene su sljedeće mjere evaluacije klasifikatora prikazane u Tablici 6.4. Točnost i preciznost su iznad 80 % što je generalno govoreći vrlo dobar rezultat za višeklasnu klasifikaciju. Međutim, u ovoj primjeni je na granici prihvatljivosti budući da je potrebno dobiti omjere količine bijelih krvnih stanica koji upućuju na neku bolest. Kako bi potencijal konvolucijskih neuronskih mreža bio u potpunosti iskorišten, ključno je imati dovoljan broj uzoraka za učenje modela, tj. dovoljan broj slika bijelih krvnih stanica. U ovom slučaju korišten je relativno mali broj slika za treniranje modela što se i odrazilo na preciznost samog klasifikatora. Osim toga preciznost bi se možda mogla povećati ako bi se povećala rezolucija ulazne slike čime bi se dobilo više detalja, ali i višestruko produžilo vrijeme treniranja modela. Kako je vidljivo u tablici 6.4. klasifikator na testnom skupu slika zamijeni eozinofile i

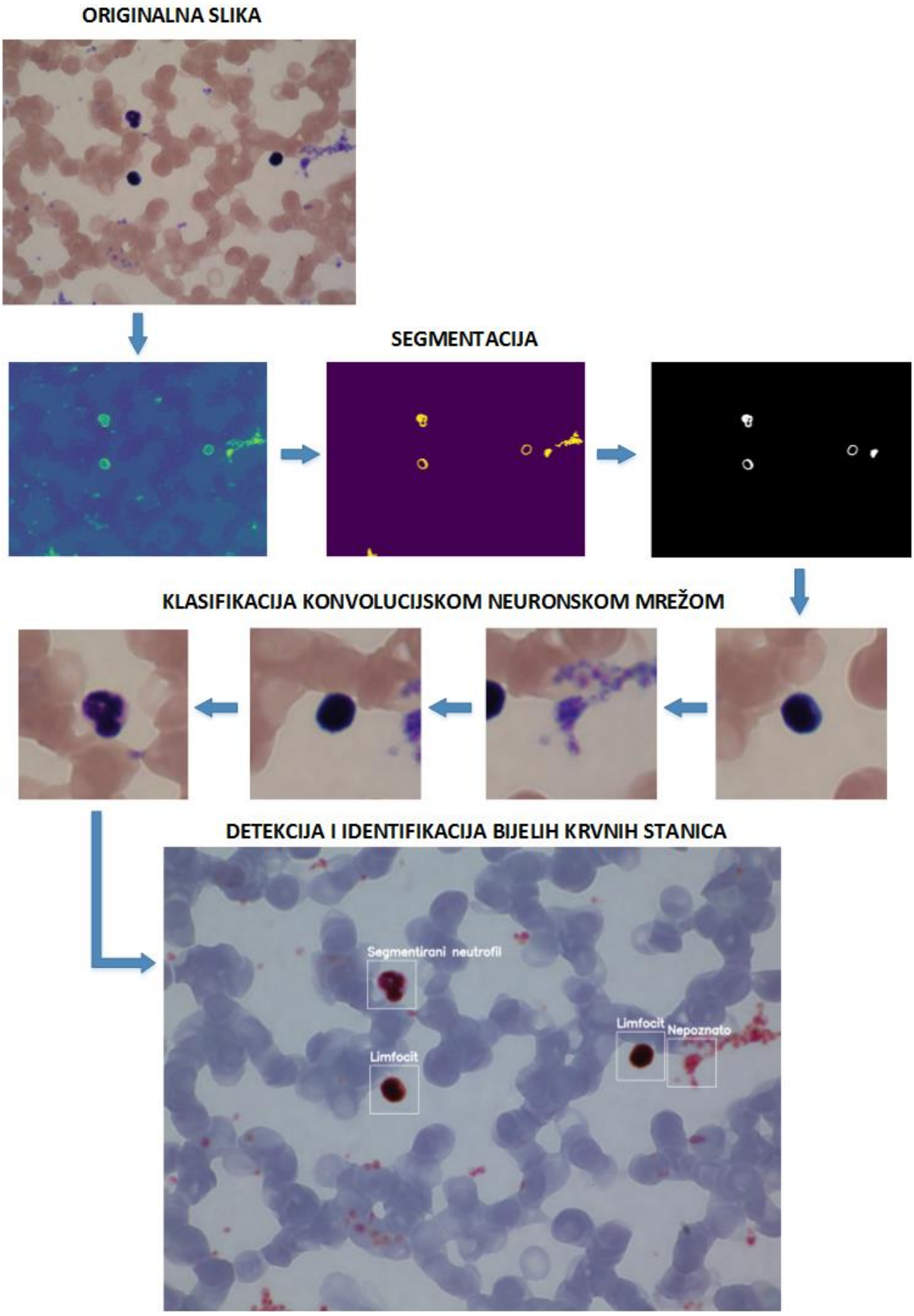
segmentirane neutrofile. To je djelomično očekivano budući da ova dva tipa za razliku od monocita i limfocita imaju segmentiranu jezgru.

Tab. 6.5. Matrica zabune klasifikatora

Preciznost	81.11 %
Odziv	100.00 %
Točnost	81.11 %
F1 mjera	89.61 %

6.3. Primjer rada predloženog algoritma

Na slici 6.4. prikazan je rad predloženog algoritma. U prvom koraku se originalna slika segmentira kroz 3 stadija kako bi se izdvojile sve bijele krvne stanice. Dobiveni rezultati segmentacije služe kao ulaz u istrenirani model konvolucijske neuronske mreže koja klasificira bijele krvne stanice u jednu od 5 kategorija i označava ih na slici.



Sl. 6.4. Prikaz rada predloženog algoritma za detekciju i klasifikaciju bijelih krvnih stanica

7. ZAKLJUČAK

U okviru ovog rada predložen je algoritam za detekciju i identifikaciju bijelih krvnih stanica koji može poslužiti za automatizaciju dobivanja diferencijalne krvne slike, tj. omjera pojedinog tipa bijelih krvnih stanica u krvnom uzorku. Trenutno korištene metode diferencijalne krvne slike temelje se na vizualnoj inspekciji krvnih preparata ili na korištenju skupe opreme koja nije dostupna svakom bolničkom centru. Upravo zato uočena je mogućnost da se ručno prebrojavanje bijelih krvnih stanica ubrza i pojeftini računalnom obradom slike. Predložena je metoda detekcije stanica temeljena na segmentaciji slike te identifikaciji stanica temeljenoj na dubokoj konvolucijskoj neuronskoj mreži. Klinička bolnica Osijek ustupila je 410 označenih slika krvnih razmaza. Slike bijelih krvnih stanica su klasificirane u 5 klasa (eozinofili, bazofili, monociti, limfociti, segmentirani neutrofili) od strane ovlaštenog stručnjaka.

Predloženi algoritam za detekciju i identifikaciju bijelih krvnih stnica sastoji se od segmentacije slike krvnog razmaza te klasifikacije dobivenih detekcija konvolucijskom neuronskom mrežom. Faza segmentacije uključuje izdvajanje slika bijelih krvnih stanica. Algoritam se temelji na segmentaciji po boji te veličini i povezanosti detektiranih segmenata. Parametri algoritma podešeni su eksperimentalno, tj. vizualnim prebrojavanjem rezultata segmentacije na nekoliko slika. Točnost segmentacije je blizu 90 %, dok je odziv gotovo 97 %. Vrlo je bitno da je odziv jako visok, tj. da svi potencijalni objekti budu detektirani budući da se problem lažnih detekcija rješava još i pomoću klasifikacije konvolucijskom neuronskom mrežom.

Na temelju 168 označenih slika implementirana je i trenirana duboka neuronska mreža koja klasificira objekte u pet klasa: eozinofili, limfociti, monociti, segmentirani neutrofili i nepoznato, tj. objekti koji su krivo detektirani u fazi segmentacije, a zapravo ne pripadaju niti jednoj klasi. Ovakvim pristupom su prilično uspješno uklonjene lažne detekcije iz faze segmentacije. Točnost klasifikacije na testnom skupu je približno 81 % što je vrlo dobar rezultat. Glavni problem je nedostatak slika za učenje mreže koji je djelomično ublažen korištenjem slučajnih transformacija slike. Osim toga, treba uzeti u obzir da ni stručnjaci koji su pregledavali slike ponekad nisu bili sigurni kojoj vrsti stanice objekt pripada. Cijeli sustav implementiran je u programskom jeziku Python korištenjem biblioteka *scikit-image* i *Keras* koji su se pokazali kao vrlo dobar odabir budući da se Python nametnuo kao jedan od efkasnih i najbolje dokumentiranih programskih jezika za područje strojnog i dubokog učenja.

Nedostatak algoritma segmentacija je što se u slučaju nekvalitetnog razmaza javljaju lažne detekcije. Jedan od problema je određivanje parametara algoritma koji su procijenjeni na nekoliko slika. Podešavanje parametara algoritma moglo bi se unaprijediti korištenjem nekog od

optimizacijskih postupaka kao što su genetski algoritmi. Arhitektura mreže korištena u ovom radu napravljena je po uzoru na *LeNet* [10] arhitekturu. U budućnosti, mogla bi se isprobati neka druga arhitektura mreže ili prilagoditi pojedine slojeve s ciljem dobivanja boljih rezultata klasifikacije. Osim toga, moglo bi se probati i s nekim drugim metodama pretprocesiranja slika kao što su maskiranje određenih područja. U ovom radu demonstriran je potencijal brzorastućeg područja konvolucijskih neuronskih mreža koji razvojem novih arhitektura, ali i sklopovlja kao što su specijalizirane grafičke kartice daje sve bolje rezultate.

LITERATURA

[1] LC . Junqueira, J. Carneiro, *Osnove histologije, udžbenik i atlas prema 10. američkom izdanju*, Školska knjiga, Zagreb, 2005.

[2] Bijele krvne stanice: http://www.biologija.rs/leukociti.html (29.11.2017.)

[3] Konvolucijski sloj neuronske mreže: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

[4] Sloj sažimanja neuronske mreže: https://www.embedded-vision.com/platinum-members/cadence/embedded-vision training/documents/pages/neuralnetworksimagerecognition

[5] F.Stamkenkovič, M. Čupić: *Duboke neuronske mreže*, FER, Zagreb, ožujak, 2015.

[6] Vedran Vukotić: *Raspoznavanje objekata dubokim neuronskim mrež*ama, Diplomski rad br. 696, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2014.

[7] RMSprop algoritam: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[8] Mikroskop Olympus BX-50: https://www.olympus-ims.com/de/microscope/bx50m/

[9] Image segmentation: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/ (9.11.2017.)

[10] Yann LeCun, et. al.: *Gradient-based learning applied to document recognition*, Proceedings of IEEE, 1998.

[11] http://deeplearning.net/tutorial/lenet.html (13.11.2017.)

[12] A. Krizhevsky, Ilya Sutskever, G. E. Hinton.: ImageNet Classification with Deep Convolutional Neural Networks, Communications of the ACM, Vol. 60 No. 6, Pages 84-90

[13] Python programski jezik: https://www.python.org/

[14] scikit-skimage biblioteka: http://scikit-image.org/ (9.11.2017.)

[15] van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, Gouillart E, Yu T, the scikit-image contributors, *scikit-image: image processing in Python*, PeerJ, 2014.

[16] Python scikit-image: http://scikit-image.org/docs/dev/api/skimage.html (8.11.2017.)

[17] Dokumentacija programske biblioteke Keras: https://keras.io/ (14.11.2017.)

[18] Dokumentacija programske biblioteke Matplotlib: https://matplotlib.org/ (17.11.2017.)

[19] Dokumentacija programske biblioteke h5py: http://www.h5py.org/ (15.11.2017.)

[20] I. Borko: Semantička segmentacija prirodnih scena dubokim neuronskim mrežama, Diplomski rad br. 1147, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2015.

[21] Li Fei-Fei, Rob Fergus, i Antonio Torralba. *Recognizing and learning object categories*, CVPR Short Course, 2, 2007.

[22] Bojana Dalbelo Bašić, Marko Čupić, Jan Šnajder. Umjetne neuronske mreže. Fakultet elektrotehnike i računarstva, 2008.

SAŽETAK

Naslov: Klasifikacija slika metodama dubokog učenja

Sažetak: Područje dubokih neuronskih mreža doživljava nagli uzlet u posljednjih nekoliko godina te postiže izvrsne rezultate u području obrade slike i računalnog vida. Diferencijalna krvna slika, tj. analiza bijelih krvnih stanica je jedno od zanimljivih područja primjene ovakvih metoda budući da se trenutno radi vizualnom inspekcijom krvnih preparata što je vremenski zahtjevno ili vrlo skupom i sofisticiranom opremom. U sklopu ovog rada predložen je algoritam za detekciju i identifikaciju bijelih krvnih stanica na krvnim razmazima koji se dobivaju kamerom i mikroskopom. Prva faza obrade je detekcija bijelih krvnih stanica segmentacijom slike kojom se nastoje izdvojiti sve bijele krvne stanice. U drugoj fazi je treniran model konvolucijske neuronske mreže na temelju 168 označenih slika dobivenih iz Kliničkog Bolničkog Centra Osijek. Implementirani model klasificira stanice u 5 klasa: eozinofili, limfociti, monociti, segmentirani neutrofili i nepoznato, tj. slučaj kada je došlo do pogreške segmentacije. Točnost segmentacije je blizu 90 %, dok je točnost klasifikacije oko 81 %. Predloženi algoritam dao je vrlo dobre rezultate te demonstrirao potencijal korištenih metoda koji bi u potpunosti bio iskorišten kada bi model bio treniran na puno većem broju slika.

Ključne riječi: duboko učenje, konvolucijske neuronske mreže, segmentacija slika, klasifikacija bijelih krvnih stanica, računalni vid

ABSTRACT

Title: Image classification using deep learning methods

Abstract: During the last few years, area of deep learning is growing and advancing rapidly, achieving very good results in image processing and computer vision. Differential blood count (white blood cells count) is one of the areas where such methods could be applied. Currently, it is done by visual inspection which is very time consuming or by using very expansive and sophisticated equipment. In this paper, a new system for white blood cells detection and identification is described and implemented. The first step is image segmentation, which extracts single cells out of original image. The second step is deep convolutional neural network training based on 168 labeled images from Clinical Hospital Center Osijek. Implemented model classifies white blood cells into five classes: eosinophiles, lymphocytes, monocytes, neutrphiles or unknown object (handles the case when segmentation has failed). Accuracy of segmentation on test samples is around 90 %, while accuracy of classification on test samples is around 81 %. Taking into account relatively small number of training samples implemented system showed very good results. It has demonstrated the potential of convolutional neural networks that would be fullfilled on much larger image dataset.

Keywords: deep learning, convolutional neural networks, image segmentation, white blood cells classification, computer vision

ŽIVOTOPIS

Filip Novoselnik rođen je 21.01.1992. godine u Našicama, a živi u Belišću. Nakon završene osnovne škole 2007. godine upisuje opću gimnaziju u Srednjoj školi Valpovo. Po završetku srednje škole upisuje Elektrotehnički fakultet u Osijeku, sveučilišni preddiplomski studij računarstva te 2014. godine stječe titulu sveučilišnog prvostupnika/inženjera računarstva (univ.bacc.ing.comp). Trenutno je student druge godine diplomskog studija procesnog računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Uz redovno obrazovanje, aktivno se bavi astronomijom u sklopu Astrnomskog društva „Anonymus“ Valpovo te kroz rad Hrvatske Meteorske Mreže u sklopu koje je u koautorstvu objavio nekoliko desetaka radova u znanstvenim časopisima i u zbornicima s konferencija. Jedan je od sudionika projekta *La Sagra Sky Survey* u sklopu kojeg je otkrivena 21 supernova potvrđena od strane Međunarodne astronomske unije. Dobitnik je Rektorove nagrade za 2016. godinu za rad „Detekcija crvenih krvnih stanica Houghovom transformacijom“. Dugogodišnji je sudionik i mentor Višnjanske škole astronomije. Trenutno se bavi područjima robotike i umjetne inteligencije u sklopu jedne privatne tvrtke.

PRILOZI

Prilog P.12.1. segment.py

```
from skimage import io
from matplotlib import pyplot as plt
from skimage import morphology
from skimage.morphology import label
from skimage.measure import regionprops
from skimage.segmentation import clear_border
import matplotlib.cm as cm
from skimage.transform import resize
import os

## Image segmentation

path = './crop/'
coords = open("coordinates.txt", "w")
img_fname = 'raw/test.png'

# algorithm parameters
nMin = 1500
nNeigh = 10
nEdge = 100

#  load image
img_rgb = io.imread(img_fname)

# blue channel
bPlane = img_rgb[:,:,2]  - 0.5*(img_rgb[:,:,0]) - 0.5*(img_rgb[:,:,1]);

# detect everything above threshold
BW = bPlane > 30;

# remove small objects
cleanSmall = morphology.remove_small_objects(BW, min_size=nMin, connectivity=nNeigh)

# remove detections close to the edge - 100 pixels
cleanEdge = clear_border(cleanSmall, nEdge)

# show original image

plt.figure(1)
plt.imshow(img_rgb)
plt.show()
plt.imsave("segmentation/original.png", img_rgb)

# show blue channel
plt.figure(2)
plt.imshow(bPlane)
plt.show()
plt.imsave("segmentation/blue_channel.png", bPlane)

# show cleaned image
plt.figure(3)
plt.imshow(cleanSmall)
plt.show()
plt.imsave("segmentation/cleaned.png", cleanSmall)

# show segmented image
plt.figure(4)
plt.imshow(cleanEdge, cmap=cm.gray)
plt.show()
plt.imsave("segmentation/segmented.png",  cleanEdge, cmap=cm.gray)

```

```
# mark regions on image
label_img = label(cleanEdge)
props = regionprops(label_img)

x_coord = []
y_coord = []

# delete old files from folder
files = [f for f in os.listdir(path)]

for f in files:
    os.remove(path + f)

# cut and save detections
i = 0
print 'Coordinates of detections (x, y)'
for region in props:
    r0, c0, r1, c1 = region.bbox
    x,y = region.centroid
    print x,y
    x_coord.append(x)
    y_coord.append(y)
    img_crop = img_rgb[r0-170:r1+170, c0-170:c1+170]
    img_resize = resize(img_crop, (300, 300))
    plt.imsave("crop/crop" + str(i) + ".png", img_resize)
    #plt.imsave("test_segment/crop" + str(i) + ".png", img_crop)
    coords.write(str(x) + " " + str(y) +  "\n")

    i+=1

coords.close()


## Remove multiple detections

coords = open("coordinates.txt", "w")

import os

      x_diff = []
      y_diff = []
      for x1, x2 in zip(x_coord, x_coord[1:]):
          diffX = x2-x1
          x_diff.append(diffX)


      for y1, y2 in zip(y_coord, y_coord[1:]):
          diffY = y2-y1
          y_diff.append(diffY)


      l = 0
      for z,k in zip(x_diff,y_diff):
          if ((abs(z) and abs(k)) < 100):
              os.remove('test/crop' + str(l) + '.png')
          l = l + 1
```

Prilog P.12.2. train.py

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Lambda
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

```
from keras.utils import np_utils
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import train_test_split
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score
from sklearn.metrics import roc_curve, auc
import csv
import cv2
import scipy
import os
import h5py

# Define model parameters

batch_size = 32
nClasses = 5
epochs = 20
imgWidth = 300
imgHeight = 300
nChannels = 3


# Define model architecture

def getModel():
    model = Sequential()
    model.add(Lambda(lambda x: x/127.5 - 1., input_shape=(imgWidth, imgHeight, nChannels), output_shape=(imgWidth, imgHeight, nChannels)))

    model.add(Conv2D(32, (3, 3), input_shape=(imgWidth, imgHeight, nChannels)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(0.5)) # mozda bi trebalo izbaciti dropout
    model.add(Dense(nClasses))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])

    return model


model = getModel()
path = './original_dataset'

X = []
y = []

for root, dirs, files in os.walk(path):
```

```
    for filename in files:
        #print file
        #print os.path.abspath(os.path.join(root, filename))

        if filename.endswith('.png'):
            parts = filename.split('.')
            parts[0] = parts[0].split('_')
            label = parts[0][0]
            img_fname = os.path.abspath(os.path.join(root, filename))
            img_file = cv2.imread(img_fname)
            img_arr = np.asarray(img_file)
            X.append(img_arr)
            y.append(label)


X = np.asarray(X)
y = np.asarray(y)

encoder = LabelEncoder()
encoder.fit(y)
encoded_y = encoder.transform(y)

y = np_utils.to_categorical(encoded_y)


# Train and test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)


      # Data augumentation

      datagen = ImageDataGenerator(
          rotation_range=20,
          width_shift_range=0.2,
          height_shift_range=0.2,
          horizontal_flip=True)


      train_generator = datagen.flow(
              X_train,
              y_train,
              batch_size=batch_size)

      validation_generator = datagen.flow(
              X_test,
              y_test,
              batch_size=batch_size)



      # save training checkpoints

      model = getModel()
      checkpoint = ModelCheckpoint(filepath='./checkpoint-{epoch:02d}-{val_loss:.2f}.h5')


      # # Train
      # fits the model on batches with real-time data augmentation:
      model_training = model.fit_generator(
          train_generator,
          steps_per_epoch=len(X_train),
          validation_data=validation_generator,
          validation_steps=len(X_test),
          epochs=epochs,
          callbacks=[checkpoint])
```

```
138.        model.save('trained_model.h5')
139.        model.save_weights('trained_weights.h5')
140.
141.
142.        # Load model
143.        model = getModel()
144.        model.load_weights('trained_weights.h5')
145.
146.        ## Learning curve
147.        import matplotlib.pyplot as plt
148.
149.        def plot_learning_curve(model_training):
150.            plt.clf()
151.            plt.figure(figsize=(12,8))
152.            plt.plot(model_training.history['acc'])
153.            plt.plot(model_training.history['val_acc'])
154.            plt.title('Tocnost modela', fontsize=20)
155.            plt.ylabel('Tocnost', fontsize=18)
156.            plt.xlabel('Epoha', fontsize=18)
157.            plt.legend(['Trening skup', 'Validacijski skup'], loc='lower right', prop={'
    size': 18})
158.            plt.savefig('./metrics/accuracy_curve.png')
159.            plt.clf()
160.            plt.plot(model_training.history['loss'])
161.            plt.plot(model_training.history['val_loss'])
162.            plt.title('Funkcija gubitka', fontsize=20)
163.            plt.ylabel('Gubitak', fontsize=18)
164.            plt.xlabel('Epoha', fontsize=18)
165.            plt.legend(['Trening skup', 'Validacijski skup'], loc='upper left', prop={'s
    ize': 18})
166.            plt.savefig('./metrics/loss_curve.png')
167.
168.        plot_learning_curve(model_training)
169.
170.
171.        # Metrics
172.        print('Predicting on test data')
173.        y_pred = np.rint(model.predict(X_test))
174.
175.        print "Accuracy score: "
176.        print(accuracy_score(y_test, y_pred))
177.
178.        print "Precision score: "
179.        precision = precision_score(y_test, y_pred, average = None)
180.        print precision
181.
182.        print "Recall score: "
183.        recall = recall_score(y_test, y_pred, average = None)
184.        print recall
185.
186.        print "f1 score: "
187.        f1 = 2 * precision * recall / (precision + recall)
188.        print f1
189.
190.        y_pred_unencoded = np.argmax(y_pred, axis=1)
191.        y_test_unencoded = np.argmax(y_test, axis=1)
192.
193.        print "Confusion matrix: "
194.        cm = confusion_matrix(y_test_unencoded, y_pred_unencoded)
195.        print cm
```

Prilog P.12.3. predict.py

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Lambda
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import train_test_split
from sklearn.metrics import roc_curve, auc
import csv
import cv2
import scipy
import os
import h5py
from keras.callbacks import ModelCheckpoint
from skimage import io

# Model parameters
batch_size = 32
num_classes = 5
epochs = 10

# Model architecture
def get_model():
    model = Sequential()
    model.add(Lambda(lambda x: x/127.5 -
 1., input_shape=(300, 300, 3), output_shape=(300, 300, 3)))
    model.add(Conv2D(32, (3, 3), input_shape=(300, 300, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(0.5)) # mozda bi trebalo izbaciti dropout
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])

    return model
model = get_model()
model.load_weights('trained_weights_5_classes_1.h5')


## Make predictions on new (segmented) image

labels = open('labels.txt', 'w')

path = './crop/'
files = [f for f in os.listdir(path) if f.endswith('.png')]
files.sort(key=lambda f: int(filter(str.isdigit, f)))

```

```
print 'CNN predictions: \n'

p = 0
for e in files:
    img_fname = path + files[p]
    X = []
    img_file = cv2.imread(img_fname)
    img_arr = np.asarray(img_file)
    X.append(img_arr)
    X = np.asarray(X)
    # Predict
    y_pred = np.rint(model.predict(X))
    y_pred_test = model.predict(X)
    print y_pred

    if (y_pred[0][0] == 1):
        print 'Eozinofil'
        labels.write('Eozinofil' + '\n')
    if (y_pred[0][1] == 1):
        print 'Limfocit'
        labels.write('Limfocit' + '\n')
    if (y_pred[0][2] == 1):
        print 'Monocit'
        labels.write('Monocit' + '\n')
    if (y_pred[0][3] == 1):
        print 'Segmentirani neutrofil'
        labels.write('Segmentirani neutrofil' + '\n')
    if (y_pred[0][4] == 1):
        print 'Nepoznato'
        labels.write('Nepoznato' + '\n')
    p = p + 1

       labels.close()


       ## Add annotations

       import cv2
       import numpy as np

       img_rgb = io.imread('raw/test.png')

       w = 100
       h = 100
       font = cv2.FONT_HERSHEY_SIMPLEX

       fopen_labels = open("labels.txt", "r")
       fopen_coords = open ("coordinates.txt", "r")

       labels = [y.strip() for y in fopen_labels]
       coordinates = fopen_coords.readlines()

       i = 0
       for line in coordinates:
           line = line.strip()
           line = line.split(" ")
           line[0] = int(float(line[0]))
           line[1] = int(float(line[1]))
           cv2.rectangle(img_rgb, (line[1]-w, line[0]-
   h), (line[1]+w, line[0]+h), (255, 255, 255), 2)
           cv2.putText(img_rgb, labels[i], (line[1] - 100, line[0] -
    120), font, 1.6, (255, 255, 255), 6, cv2.LINE_AA)
           i = i + 1

       cv2.imwrite("results/detections.png", img_rgb)
```