Sintetičko generiranje mikroskopskih slika krvnih stanica za potrebe treniranja neuronske mreže

Petrik, Krunoslav

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

Sveučilišni studij

SINTETIČKO GENERIRANJE MIKROSKOPSKIH SLIKA KRVNIH STANICA ZA POTREBE TRENIRANJA NEURONSKE MREŽE

Diplomski rad

Krunoslav Petrik

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 13.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Krunoslav Petrik
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1238R, 08.10.2021.
OIB studenta:	02552582682
Mentor:	prof. dr. sc. Irena Galić
Sumentor:	Marin Benčević, mag. ing. comp.
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 1:	prof. dr. sc. Irena Galić
Član Povjerenstva 2:	dr. sc. Marija Habijan
Naslov diplomskog rada:	Sintetičko generiranje mikroskopskih slika krvnih stanica za potrebe treniranja neuronske mreže
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Istražiti i opisati mikroskopske slike krvnog brisa, vrste krvnih stanica i korist proučavanja krvnih stanica mikroskopom u biomedicini. Razviti i opisati algoritam koji metodama obrade slike generira realistične umjetne slike krvnih stanica. Trenirati jednostavnu neuronsku mrežu za prepoznavanje krvnih stanica na umjetnim slikama i testirati na stvarnim slikama. Usporediti generirane slike sa stvarnima. Usporediti rezultate mreže trenirane na umjetnim slikama s mrežama treniranim na stvarnim slikama. Sumentor: Marin Benčević. Za sve informacije u vezi rada javiti se na marin.benčević@ferit.hr. Rezervirano za: Krunoslav Petrik
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	13.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 03.10.2023.

Ime i prezime studenta:	Krunoslav Petrik
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1238R, 08.10.2021.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Sintetičko generiranje mikroskopskih slika krvnih stanica za potrebe treniranja neuronske mreže**

izrađen pod vodstvom mentora prof. dr. sc. Irena Galić

i sumentora Marin Benčević, mag. ing. comp.

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

U radu se istražuje i opisuje primjena sintetičkog generiranja mikroskopskih slika krvnih stanica u svrhu treniranja neuronske mreže. Prvo, analiziraju se mikroskopske slike krvnog brisa te se opisuju različite vrste krvnih stanica i korist proučavanja krvnih stanica mikroskopom u području biomedicine. Nakon toga, razvija se i opisuje algoritam koji koristi metode obrade slike kako bi generirao realistične i vjerodostojne umjetne slike krvnih stanica.

Glavni cilj rada je generirati što vjerodostojniji umjetni skup podataka alatima za obradu slika, analizirati moguća rješenja te trenirati vlastitu neuronsku mrežu na umjetnom skupu podataka i provesti validaciju iste.

Ovim radom se želi doprinijeti razumijevanju mogućnosti sintetičkog generiranja mikroskopskih slika krvnih stanica te pružiti uvid u potencijalnu primjenu takvih slika u biomedicinskim istraživanjima i dijagnostici.

2. PREGLED PODRUČJA TEME

2.1. Pregled skupa podataka

Skup podataka *A dataset for microscopic peripheral blood cell images for development of automatic recognition systems* [1] će se koristiti za potrebe ovog rada. Skup podataka sadrži ukupno 17.092 slike pojedinačnih normalnih stanica, koje su dobivene korištenjem analizatora CellaVision DM96 u Core laboratoriju klinike *Hospital Clinic* u Barceloni. Skup podataka je organiziran u osam skupina: neutrofili, eozinofili, bazofili, limfociti, monociti, nezreli granulociti (promijelociti, mijelociti i metamilociti), eritroblasti i trombociti. Veličina slika je 360 × 363 piksela, u *JPEG* formatu, a označene su od strane stručnih kliničkih patologa. Slike su snimljene kod osoba bez infekcije, hematoloških ili onkoloških bolesti i bez farmakološkog tretmana u trenutku uzorkovanja krvi.

Slika 2.1 Aparat CellaVision DM96 [1]

2.1.1. Podaci

Skup podataka normalnih perifernih krvnih stanica sadrži ukupno 17.092 slike pojedinačnih stanica, koje su dobivene korištenjem analizatora *CellaVision DM96*. Sve slike su dobivene u *RGB*

prostoru boja, format i veličina slika su *JPEG* i veličine 360 × 363 piksela, te su označene od strane kliničkih patologa *Hospital Clinic* u Barceloni.

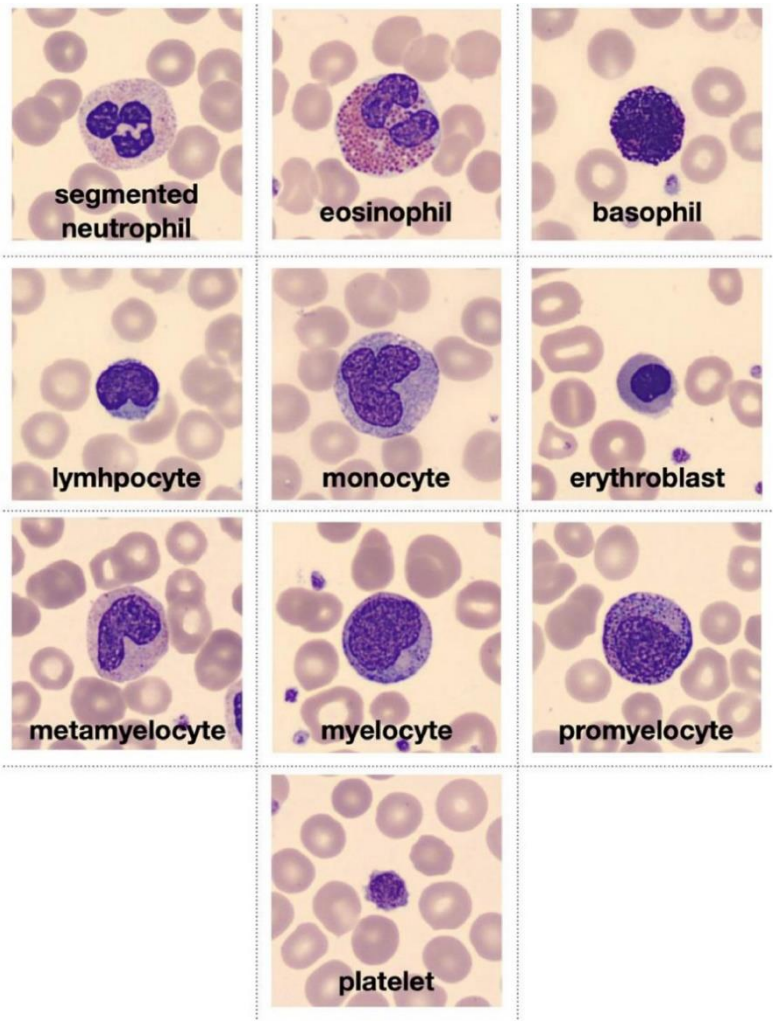
Skup podataka je podijeljen u osam grupa različitih tipova krvnih stanica vidljivo tablicom 2.1.

TIP STANICE	UKUPAN BROJ SLIKA PO TIPU	POSTOTNI UDIO
Neutrofili	3329	19.48 %
Eozinofili	3117	18.24 %
Bazofili	1218	7.13 %
Limfociti	1214	7.10 %
Monociti	1420	8.31 %
Nezreli granulociti (metamijelociti, mjelociti i promijelociti)	2895	16.94 %
Eritroblasti	1551	9.07 %
Plateli (trombociti)	2348	13.74 %
Ukupno	17,092	100 %

Tablica 2.1 Podjela skupa podataka po tipu stanice [1]

Unatoč tome što skup nezrelih granulocita uključuje mijelocite, metamijelocite i promijelocite, u ovom skupu podataka su grupirani iz dva glavna razloga: pojedinačna identifikacija specifičnih podgrupa nema poseban značaj za dijagnozu i morfološke razlike među ovim grupama su subjektivne čak i za kliničkog patologa.

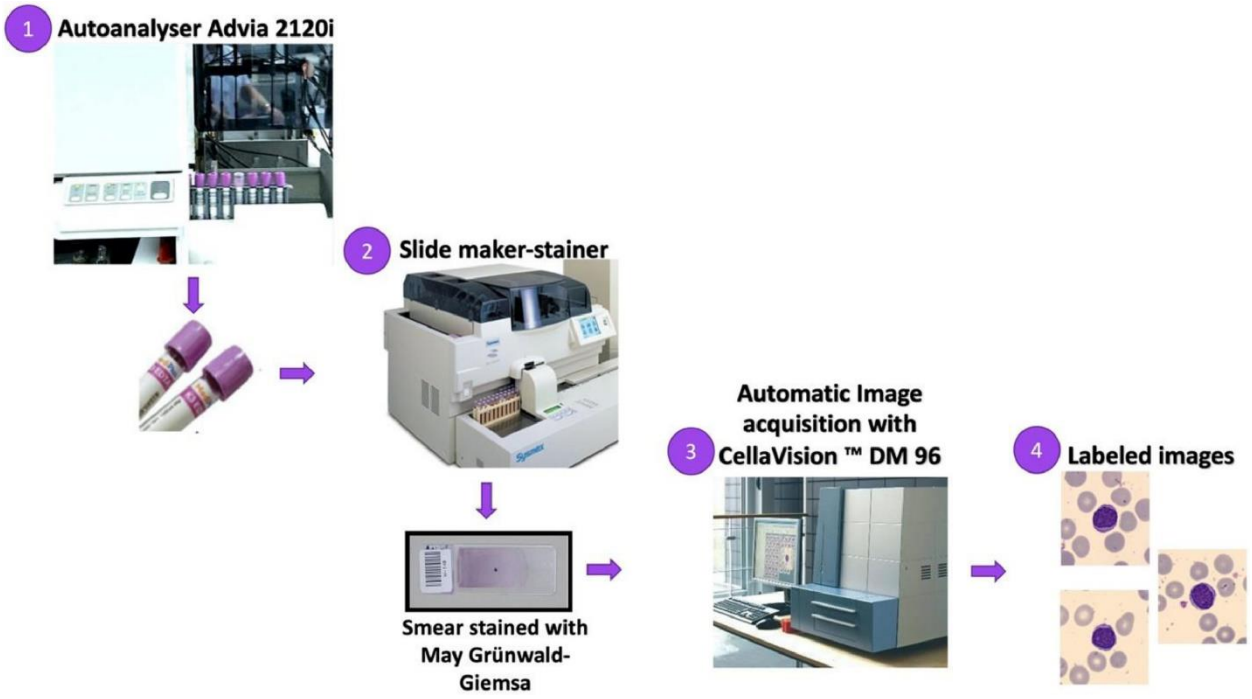
Slika 2.2 *Primjeri stanica iz skupa podataka* pokazuje primjere različitih vrsta perifernih krvnih stanica koji čine dio ovog skupa podataka organiziranog u osam različitih grupa, uključujući i one koji se češće pojavljuju za vrijeme infekcija.



Slika 2.2 Primjeri stanica iz skupa podataka

2.1.2. Metode prikupljanja podataka

Slike su dobivene tijekom razdoblja od 2015. do 2019. godine iz krvnih razmaza prikupljenih od pacijenata bez infekcija, hematoloških ili onkoloških bolesti te bez primjene bilo kakvog farmakološkog tretmana u trenutku vađenja krvi. Slikom 2.3 je prikazan postupak svakodnevnog radnog toka standardiziranog u *Core Laboratory* klinike *Hospital Clinic* u Barceloni.



Slika 2.3 Tok rada u laboratoriju [1]

Tijek rada započinje u automatskom analizatoru *Advia 2120*, gdje se krvni uzorci obrađuju u svrhu dobivanja općeg broja stanica. U drugom koraku, krvni razmazi se automatski boje korištenjem *May-Grünwald-Giemsa* postupka u automatskom bojaču *Sysmex SP1000i*. Ovaj automatizirani postupak osigurava jednako i stabilno bojanje bez obzira na specifičnog korisnika. Laboratorij ima standardizirani sustav kontrole kvalitete koji nadzire postupak.

Nakon toga, obojeni razmaz se šalje na aparat *CellaVision DM96* gdje se izvodi automatsko snimanje slika. Kao rezultat toga, dobivene su slike pojedinačnih normalnih stanica krvi, u *JPEG* formatu i veličini 360 × 363 piksela. Svaka slika stanice je označena od strane kliničkog patologa i spremljena s nasumičnim identifikacijskim brojem kako bi se uklonila povezanost i mogućnost praćenja do podataka pacijenta, rezultirajući anonimiziranim skupom podataka. Nije izvršena filtracija ni dodatna predobrada slika.

2.2. Medicinska pozadina

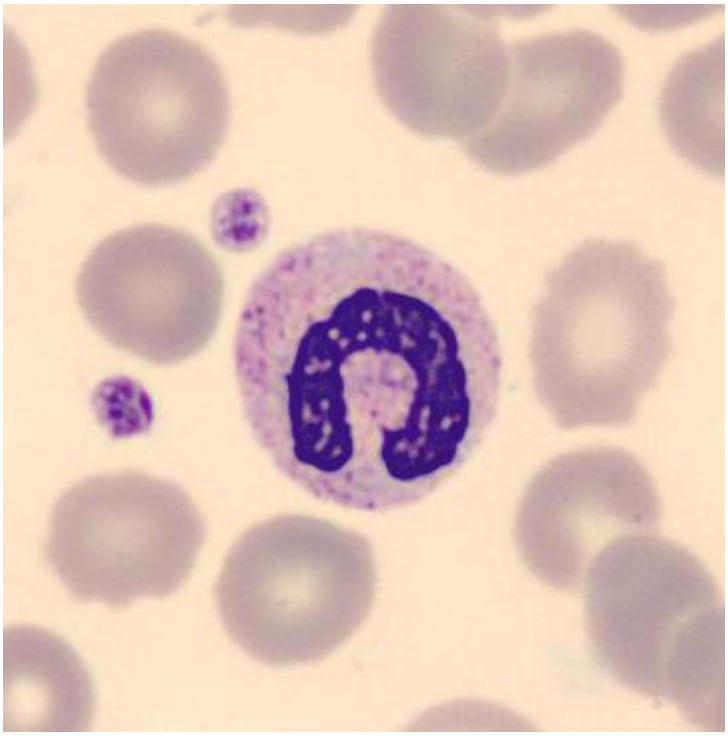
Prekomjerna ili nedovoljna količina bilo kojih perifernih krvnih stanica može upućivati na ozbiljne zdravstvene probleme. Sljedeća potpoglavlja će opisivati različite krvne stanice koje su subjekt obrade te njihovu medicinsku važnost.

2.2.1. Neutrofili

Neutrofili su povijesno definirani kao "vojnici našeg prirođenog imunološkog sustava". Oni su prva linija stanica koje se regrutiraju na mjestu infekcije, napadaju, unose i probavljaju

mikroorganizme proizvodnjom reaktivnih kisikovih vrsta. Također igraju važnu ulogu u akutnim i kroničnim upalnim stanjima te autoimunim poremećajima. Kod odraslih osoba, približan normalni raspon broja bijelih krvnih stanica je od 4000 do 11.000 stanica/mikroL, od čega je 60% do 70% zrelih neutrofila koji cirkuliraju u perifernoj krvi [2].

Apsolutni broj neutrofila (ANC), definiran kao postotak neutrofila u krvi kod odraslih osoba, obično se kreće između 2500 i 7000 neutrofila/mikroL. Povećanje broja bijelih krvnih stanica iznad 11.000 stanica/mikroL definira se kao leukocitoza. Neutrofilija je najčešći oblik leukocitoze. Definira se kao povećanje apsolutnog broja neutrofila od otprilike više od 7700 neutrofila/mikroL (11.000 stanica/mikroL x 70 posto), tj. dvije standardne devijacije iznad srednje vrijednosti. Slika 2.4 pokazuje primjer stanice neutrofila iz skupa podataka

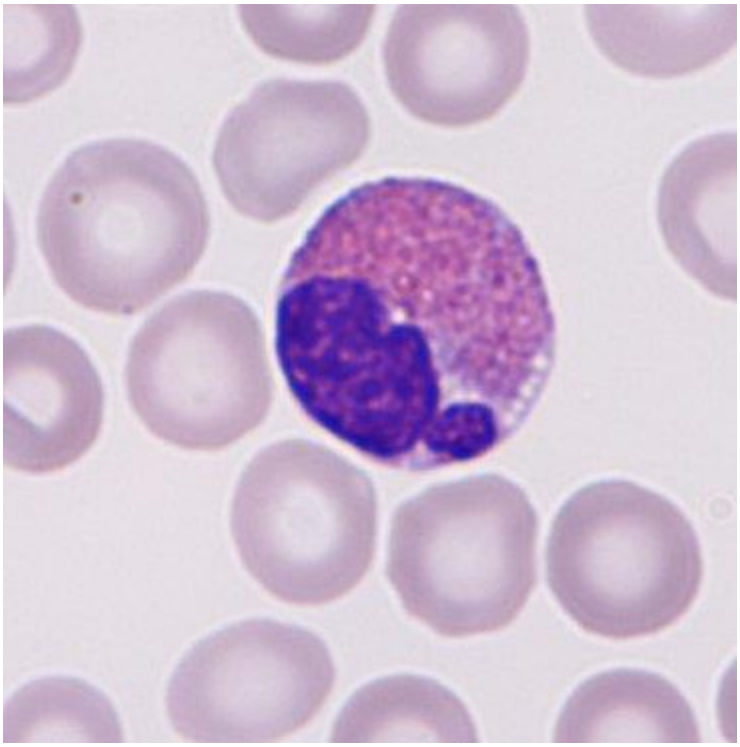


Slika 2.4 Neutrofil

2.2.2. Eozinofili

Eozinofili su vrsta granulocita u krvi koje izražavaju citoplazmatske granule koje sadrže osnovne proteine i vežu se za kisela bojila poput *eozina*. Potječu iz koštane srži, a njihovu proizvodnju potiču *IL-5, IL-3* i *GM-CSF*. Imaju poluživot u cirkulaciji od 4,5 do 8 sati. Mogu boraviti u tkivima, uglavnom u dišnim i probavnim sustavima, od 8 do 12 dana. Eozinofili čine manje od 5% cirkulirajućih leukocita. Eozinofilija se definira kao povećanje broja cirkulirajućih eozinofila iznad 500 po milimetru kubnom. Na temelju broja, eozinofilija se može podijeliti u različite kategorije: blagu (500 do 1500/mm^3), umjerenu (1500 do 5000/mm^3) i tešku (> 5000/mm^3) [3].

Hipereozinofilni sindrom definira se kao apsolutni broj eozinofila veći od 1500/mm^3 na najmanje dva odvojena mjerenja u razmaku od najmanje mjesec dana ili označenu eozinofiliju u tkivima. Na slici 2.5 je primjer stanice eozinofila uzeta iz skupa podataka za učenje.

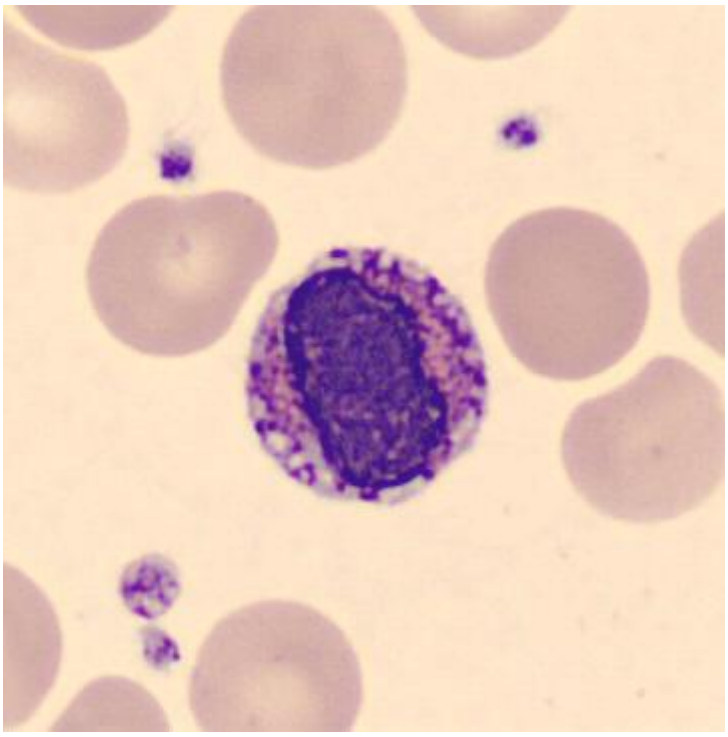


Slika 2.5 Eozinofil

2.2.3. Bazofili

Bazofili su obično najmanje brojne mijeloične stanice koje se mogu uočiti u perifernom razmazu krvi. Njihove brojne tamne azurofilne granule lako ih razlikuju od ostalih stanica. Basofilija nije uobičajeno stanje periferne krvi. Najčešće je reaktivni mehanizam često uočen u kombinaciji s eozinofilijom i apsolutnim brojem bazofila većim od 200 stanica/uL [4]. Različiti rasponi vrijednosti postavljaju se ovisno o laboratoriju i temelje se na lokalnoj populaciji. Ako se izvede, aspirati koštane srži mogu pokazati povećanje bazofila ili prekursora.

Povišene razine bazofila mogu ukazivati na prisutnost osnovnog neoplazma kao što su kronična mijeloična leukemija (*CML*), policitemija vera (*PV*), primarna mijelofibroza, esencijalna trombocitemija, akutna mijeloična leukemija ili rijetko, tumorske tvorbe. Češći uzroci uključuju alergijske reakcije ili kroničnu upalu povezanu s infekcijama (uključujući gripu i tuberkulozu), upalne bolesti crijeva i autoimune bolesti. Povezanost s unosom određenih lijekova i hrane također može korelirati sa simptomima i stupnjem bazofilije. Na slici 2.6 je prikazan primjer stanice bazofila preuzet iz skupa podataka.

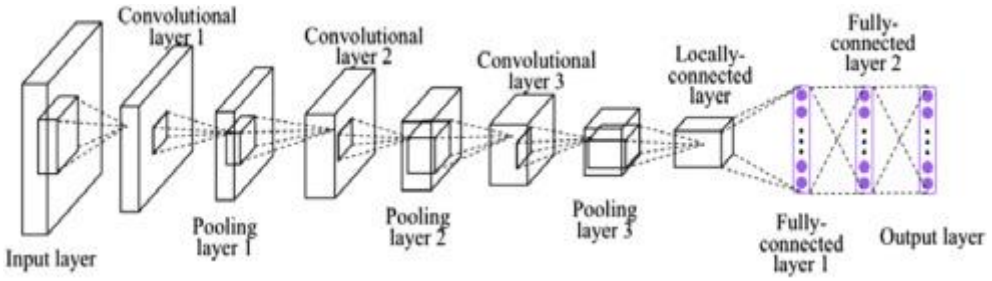


Slika 2.6 Bazofil

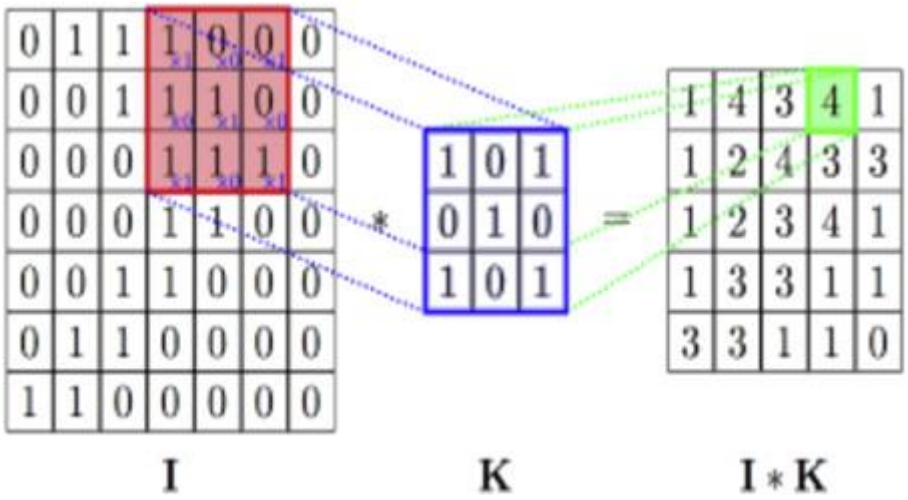
2.3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (*CNN*) su klasa dubokih, progresivnih umjetnih neuronskih mreža (*ANN*) koja se najčešće primjenjuje za analizu vizualnih slika [5]. Ova vrsta modela dubokog učenja postala je vrlo popularna nedavno zahvaljujući velikim poboljšanjima u algoritmima i računalnim mogućnostima.

CNN je duboka neuronska mreža (*DNN*) koji sadrži slojeve u kojima se izvode konvolucijske operacije. Uobičajeni dizajn *CNN*-a koristi tri osnovne vrste struktura prikazane na slikama 2.8 2.9 i 2.10.



Slika 2.7 Uobičajeni dizajn CNN-a [6]

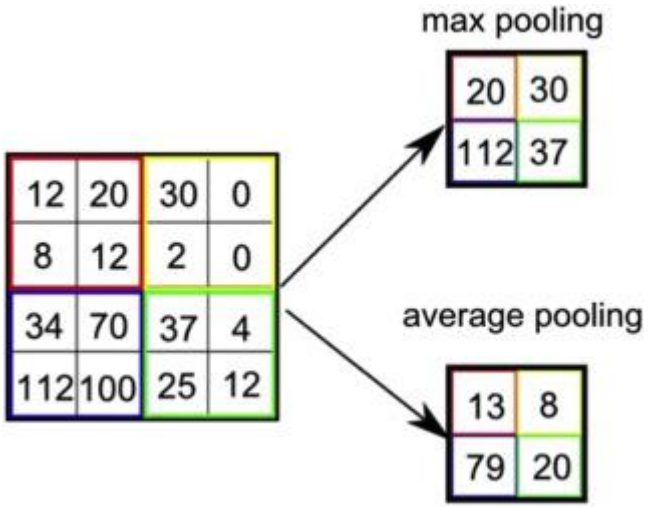


Slika 2.8 Operacija konvolucije [7]

Prva struktura naziva se *konvolucijski sloj* gdje se konvolucijske operacije primjenjuju na ulazne podatke, a zatim se izlazi šalju sljedećem sloju. Primjer konvolucijske operacije prikazan je na slici.

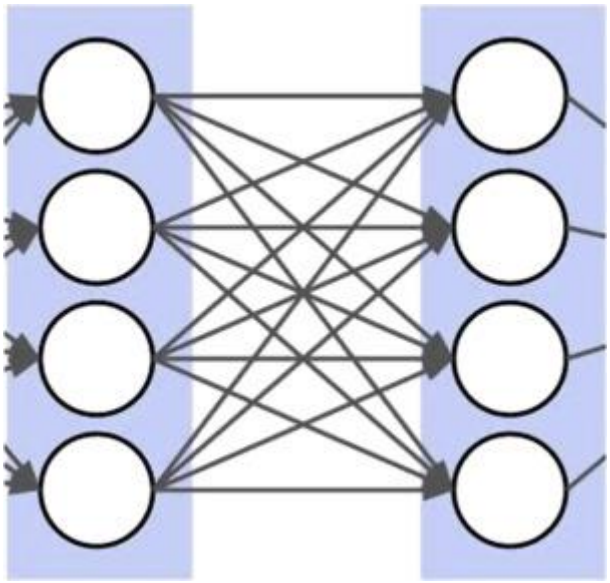
Svaki neuron konvolucijskog sloja prima samo mali dio izlaza prethodnog sloja nakon što se konvoluiraju s određenim *jezgrom*. Skup izlaznih vrijednosti koje neuron može vidjeti naziva se *receptivno polje* tog neurona.

Druga glavna struktura je *sloj grupiranja* (*pooling layer*). On kombinira svaku grupu izlaza prethodnog sloja u jedan neuron. Postoje dvije uobičajene varijacije grupirajućih operacija: prosječno grupiranje (*average pooling*) i maksimalno grupiranje (*max pooling*) (Slika 2.9). Sloj prosječnog grupiranja izračunava srednju vrijednost svojih ulaznih vrijednosti. S druge strane, maksimalno grupiranje uzima najveću vrijednost.



Slika 2.9 Pooling operacija [8]

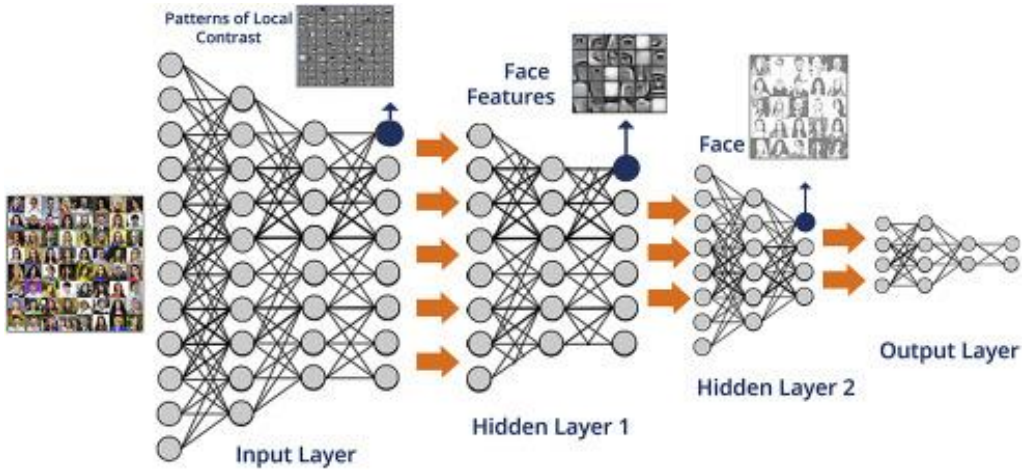
Treći tip sloja sličan je onom koji se koristi u tradicionalnim *ANN*-ovima gdje je svaki neuron u svakom sloju povezan sa svim neuronima u sljedećem sloju. Taj se sloj naziva *potpuno povezani* slojevi (*fully connected layers*) (Slika 2.10).



Slika 2.10 Potpuno povezani sloj [9]

CNN obično sadrži prethodne tri vrste slojeva. Kako je prikazano na slici , prvi dio *CNN*-a sastoji se od niza konvolucijskih slojeva i slojeva grupiranja, tako da su konvolucijski slojevi organizirani u grupe, a sloj grupiranja se stavlja nakon svake grupe konvolucijskih slojeva. Drugi dio započinje ravnanjem (*flattening*) posljednjeg sloja prvog dijela i završava s nizom potpuno povezanih slojeva.

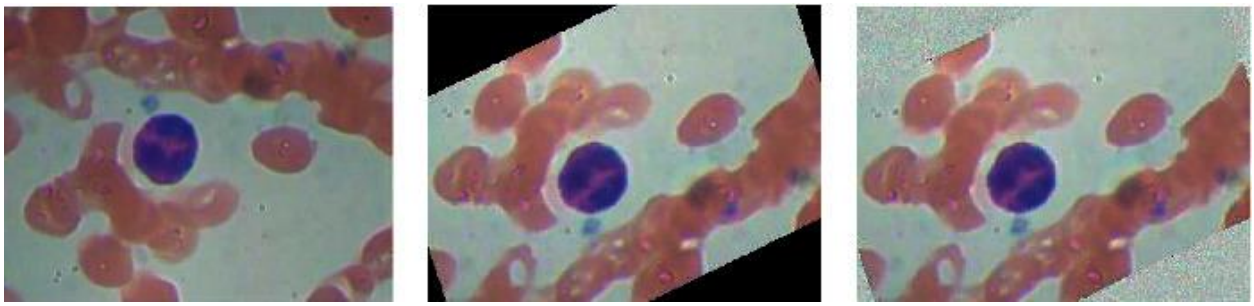
Glavna prednost *CNN*-a je da je sposoban zahvatiti složenost ulaznih podataka koristeći relativno mali broj parametara. U dobro obučenom *CNN*-u, apstraktne reprezentacije podataka se uče kako dublje ulazimo u *CNN*. To se može shvatiti ako postoji hijerarhija slojeva, pri čemu svaki dio te hijerarhije odgovara za zahvaćanje određene vrste informacija. Apstraktnije informacije zahvaćene su u dubljim dijelovima hijerarhije (Slika 2.11).



Slika 2.11 Dizajn konvolucijske neuronske mreže [9]

2.4. Pregled umjetnog generiranja podataka

Umjetno generiranje podataka je sveprisutno u treniranju dubokih neuronskih mreža jer se pomoću metoda obrada slika i matematičkih operacija može znatno proširit skup podataka za učenje i treniranje. Jedan od primjera je opisan u radu *A Method for Expanding the Training Set of White Blood Cell Images* [10] gdje se slika rotira i popunjavaju se praznine pomoću matematičkih operacija.



Slika 2.12 Rotiranje slike za proširivanje skupa podataka [10]

Svaki set podataka podliježe istom problemu, a to je da izrazito ovisi o načinu prikupljanja slika stanica: načinu bojanja krvi, korištenom mikroskopu, kameri i uvećanjem [11]. Metoda opisana u ovom radu razlikuje se od ostalih po tome što ne zahtjeva ulazni skup podataka za obradu, već se podaci generiraju od nule. Samim tim pristupom, moguće je mijenjati različite parametre koji utječu na kvalitetu skupa podataka kao što su: veličina stanice, veličina slike, rotacija, bojanje itd. Također, još jedna velika prednost je to što se podaci automatski označavaju za potrebe treniranja neuronske mreže te problem dupliciranih slika ne postoji [11].

2.5. Programski alati

2.5.1. Programski jezik Python

Python [12] je moćan i jednostavan objektno orijentirani programski jezik, sličan drugim jezicima kao što su *Perl, Ruby, Scheme* ili *Java*. Neke od ključnih značajki *Pythona* uključuju njegovu elegantnu sintaksu, koja poboljšava čitljivost koda, što ga čini idealnim za razne programerske zadatke, pogotovo u polju podatkovnih znanosti. Jednostavan je za korištenje, omogućuje razvojnim programerima brzu izradu i testiranje programa, što ga čini prikladnim za izradu prototipova bez ugrožavanja mogućnosti održavanja. *Python* nudi ogromnu standardnu biblioteku, pružajući podršku za brojne uobičajene programerske zadatke kao što je povezivanje web poslužitelja, pretraživanje teksta s regularnim izrazima i rukovanje datotekama. Jedna od prednosti *Pythona* leži u njegovoj proširivosti. Programeri mogu jednostavno dodati nove module implementirane u prevedenim jezicima poput *C* ili *C++*, proširujući njegovu funkcionalnost po potrebi. Dodatno, Python se može ugraditi u aplikacije kako bi se osiguralo programabilno sučelje. Olakšava objektno orijentirano programiranje s podrškom za klase i višestruko nasljeđivanje. Organiziranje koda u module i pakete jednostavno je u *Pythonu* i nudi robusnu obradu pogrešaka podizanjem i hvatanjem iznimaka. *Python* je snažno i dinamički tipiziran, otkrivajući nekompatibilne mješavine tipova podataka rano putem pokretanja iznimke.

2.5.2. Numpy biblioteka

NumPy je temeljna biblioteka za znanstvene proračune u *Python*-u [13]. To je *Python* biblioteka koja pruža objekt za višedimenzionalne nizove, razne izvedene objekte (kao što su maskirani nizovi i matrice), te raznolik asortiman rutine za brze operacije na nizovima, uključujući matematičke, logičke, manipulaciju oblikom, sortiranje, odabir, U/I, diskretne Fourierove transformacije, osnovnu linearnu algebru, osnovne statističke operacije, simulaciju slučajnih brojeva i mnogo više.

2.5.3. OpenCV

OpenCV (*Open Source Computer Vision Library*) [14] je otvorena biblioteka koja uključuje nekoliko stotina algoritama za računalni vid. U ovom radu opisuje se takozvani *OpenCV 2.x API,* koji je primarno *C++* API, za razliku od starijeg *OpenCV 1.x API*-ja koji se bazirao na C-u (C API je zastario i nije testiran s "C" kompajlerom od izdanja *OpenCV* 2.4). *OpenCV* ima modularnu strukturu, što znači da paket uključuje nekoliko zajedničkih ili statičkih biblioteka. Dostupni su sljedeći moduli:

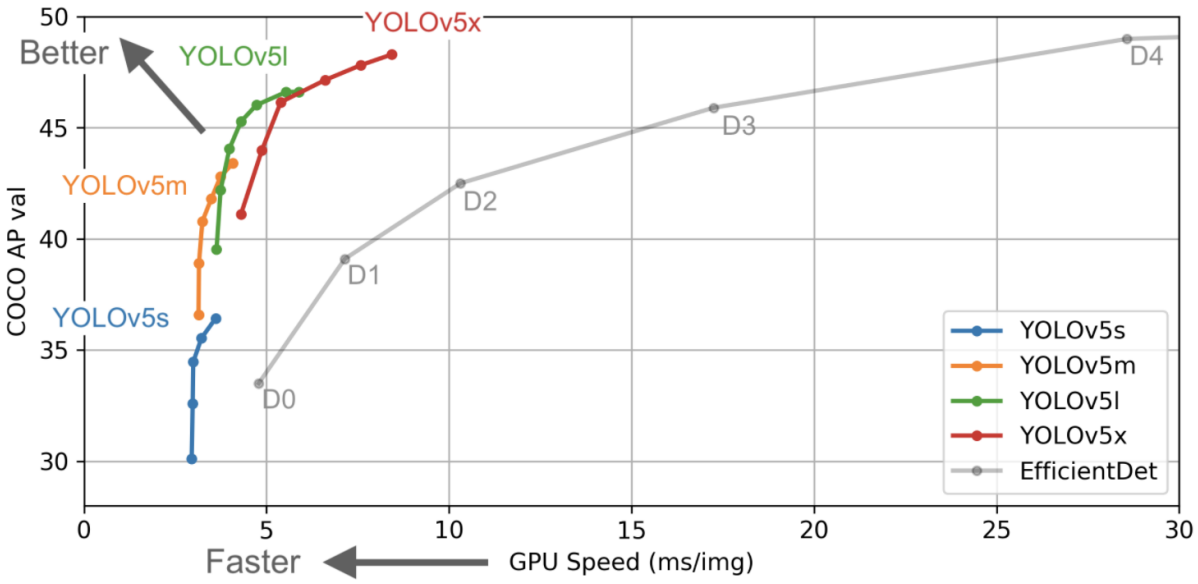
- Osnovne funkcionalnosti (*core*) - kompaktni modul koji definira osnovne strukture podataka, uključujući guste višedimenzionalne nizove *Mat* i osnovne funkcije koje koriste svi drugi moduli
- Obrada slika (*imgproc*) - modul za obradu slika koji uključuje linearno i nelinearno filtriranje slika, geometrijske transformacije slika (promjena veličine, afini i perspektivni *warping, generic table-based remapping*), pretvorbu prostora boja, histogram i drugo
- Analiza videozapisa (*video*) - modul za analizu videozapisa koji uključuje algoritme za procjenu pokreta, oduzimanje pozadine i praćenje objekata
- Kalibracija kamere i 3D rekonstrukcija (*calib3d*) - osnovni algoritmi za geometriju s više pogleda, kalibraciju pojedinačnih i stereo kamera, procjenu položaja objekta, algoritme za stereo korespondenciju i elemente 3D rekonstrukcije
- 2D okvir značajki (*features2d*) - detektori značajki, deskriptori i podudaranje deskriptora.
- Otkrivanje objekata (*objdetect*) - otkrivanje objekata i instanci predefiniranih klasa (na primjer, lica, očiju, šalica, ljudi, automobila itd.)
- Visokorazinski grafički korisnički sučelje (*highgui*) - jednostavno sučelje za osnovne mogućnosti korisničkog sučelja
- Video I/O (*videoio*) - jednostavno sučelje za snimanje videozapisa i video *kodeke*.
- ... neki drugi pomoćni moduli, poput *FLANN*-a i omotača za *Google testove*, *Python* veza i drugih

2.5.4. PyTorch

PyTorch je potpuno opremljen okvir za izgradnju modela dubokog učenja, što je vrsta strojnog učenja koja se obično koristi u aplikacijama kao što su prepoznavanje slika i obrada jezika [15]. Napisan u Pythonu, relativno je jednostavan za učenje i korištenje za većinu programera strojnog učenja. *PyTorch* je prepoznatljiv po svojoj izvrsnoj podršci za grafičke kartice i korištenju automatske diferencijacije obrnutog načina rada, što omogućuje izmjenu računskih grafikona u hodu. To ga čini popularnim izborom za brzo eksperimentiranje i izradu prototipova.

2.5.5. YOLOv5 algoritam

YOLOv5 je model iz *You Look Only Once (YOLO)* obitelji modela računalnog vida. *YOLOv5* se obično koristi za klasifikaciju objekata. *YOLOv5* dolazi u četiri glavne verzije: mala (s), srednja (m), velika (l) i ekstra velika (x), a svaka nudi progresivno veće stope točnosti. Svaka varijanta također zahtijeva različito vrijeme za treniranje, u pravilu što je veći model, duže je vrijeme treniranja.[16]



Slika 2.13 Graf ovisnosti performansi (y-os) o vremenu inferencije (x-os)

3. OPIS METODA

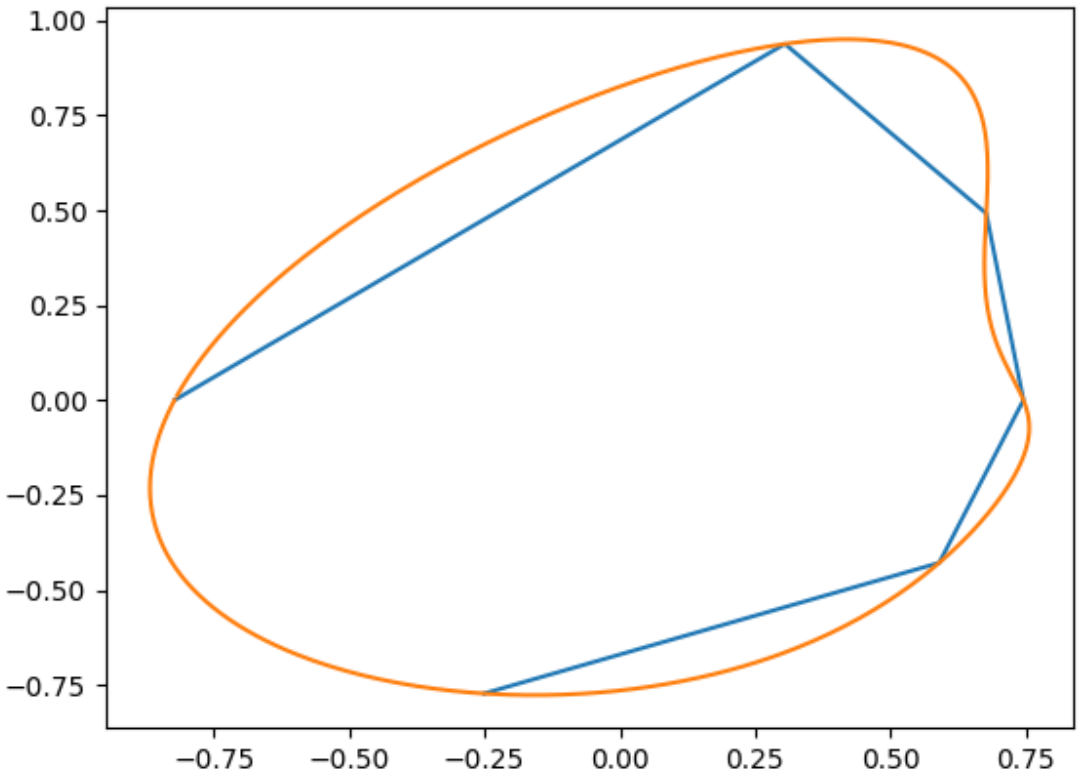
Ovim poglavljem će biti objašnjeno umjetno stvaranje stanica te ukupne slike zajedno sa stanicama.

3.1. Generiranje konture stanice

Od velikog značenja je generiranje konture stanice jer ono je temelj za stvaranje krvnih stanica, stanica monocita i neutrofila. Počinje se generiranjem sedam slučajnih kompleksnih brojeva na jediničnoj kružnici. Ovi brojevi su ravnomjerno raspoređeni oko kružnice. Zatim se računa konveksni omotač (*convex hull*) tih točaka kako bi se stvorio konveksni mnogokut. Konveksni omotač je najmanji konveksni oblik koji obuhvaća sve dane točke. Nakon toga, vrši se glatka interpolacija između točaka konveksnog omotača kako bi se generirao niz x i y koordinata koje definiraju granicu oblika nalik na kapljicu. Potrebno je stvoriti praznu sliku pomoću *numpy* niza određene veličine. Dobivene *x* i *y* koordinate normaliziraju se kako bi bile između 0 i 1, a zatim se skaliraju kako bi odgovarale veličini slike. Nakon toga, normalizirane i skalirane koordinate pretvaraju se u cijele brojeve i raspoređuju kao točke u formatu očekivanom od strane *OpenCV*-a za potrebe popunjavanja prostora unutar mnogokuta. Ako se koristi drugačiji pristup popunjavanja prostora, onda se može preskočiti taj korak. Rezultirajuće točke tvore zatvoreni mnogokut. Konačno, pomoću *OpenCV* bibilioteke se ispunjava poligon bijelom bojom (vrijednost piksela 255) na crnoj slici, stvarajući ispunjen oblik. Rezultat te funkcije prikazan je slikom Slika 4.1 i služi kao temelj za generiranje slika u ovom radu. Za računanje konveksnog plašta koristit će se postupak *Grahamovog skeniranja* [17]. Na početku se traži krajnja lijeva točka među ulaznim točkama. Ta točka će biti jedna od točaka na konveksnom omotaču. Zatim se izračunava polarni kut svih točaka u odnosu na krajnju lijevu točku. Matematička funkcija *atan2* koristi se za izračun kuta između pozitivnog dijela x-osi i zadane točke i te se pomoću te funkcije može sortirati lista točaka u ovisnosti o kutu između x-osi. Tim sortiranjem dobije se popis točaka redom po smjeru obrnuto od kazaljke na satu. Za potrebe interpolacije, koristit će se diskretna Fourierova transformacija. Diskretna Fourierova transformacija je matematička operacija koja se koristi za pretvaranje diskretnog niza podataka, često u vremenskoj domeni, u njegovu reprezentaciju u frekvencijskom domenu [18].

$$\tilde{F}(n) = \frac{1}{N}\sum_{k=0}^{N-1} f(k)e^{-j\frac{2\pi nk}{N}}. \qquad 3\text{-}1$$

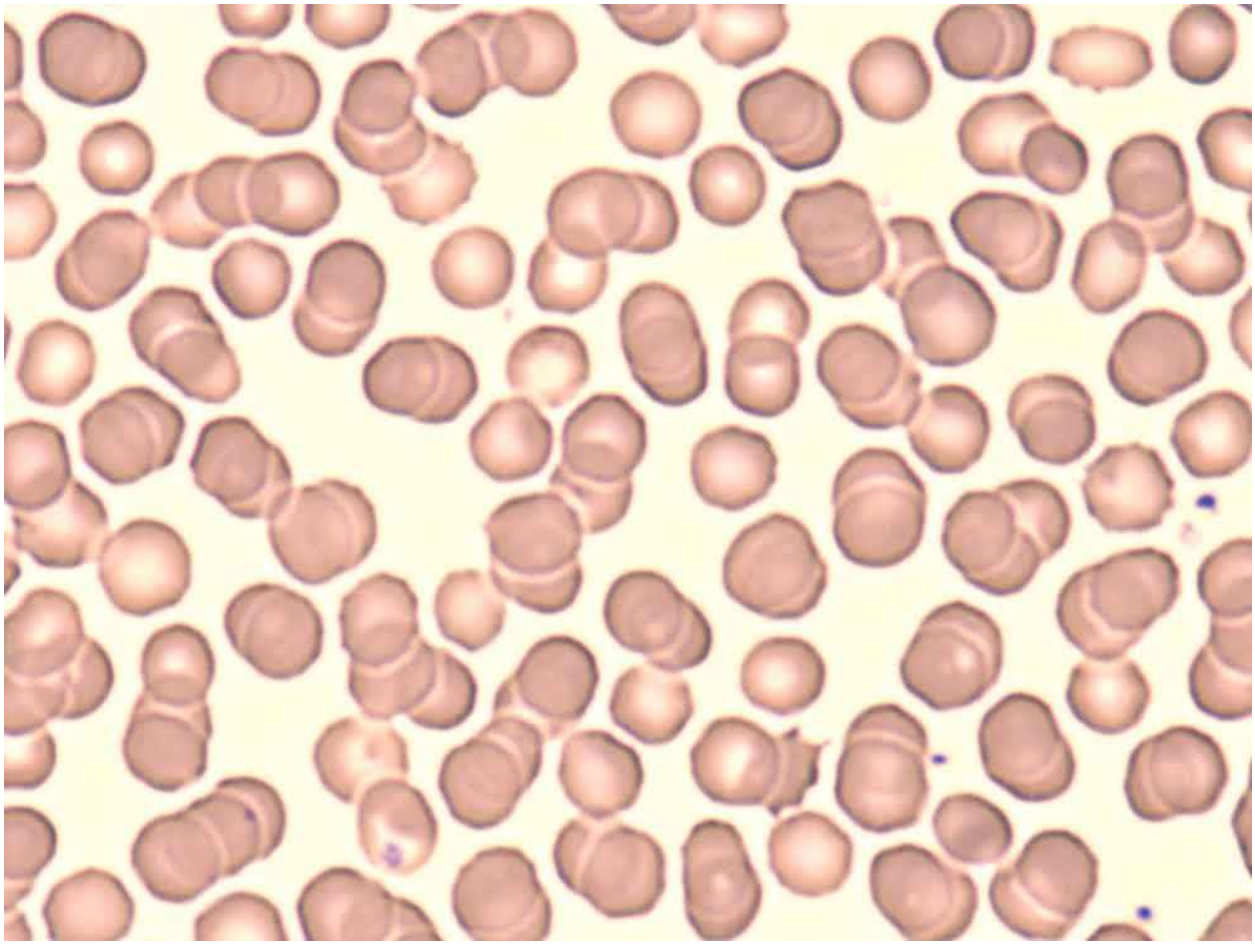
Jednadžbom 3-1 prikazana je formula za diskretnu Fourierovu transformaciju. Pomoću diskretne Fourierove transformacije, koordinate ulaznih točaka moguće je pretvoriti iz vremenske domene u frekvencijsku i obratno. Važna stavka diskretne Fourierove transformacije je to što je ona kontinuirana funkcija te je pomoću nje moguće generirati beskonačno mnogo dodatnih točaka između dvaju susjednih točaka. Glavna ideja je izabrati cjelobrojnu vrijednost koja će predstavljati broj točaka koje će se generirati interpolacijom takav da broj ne bude previše računalno zahtjevan, ali da bude dovoljno velik da preciznost prikaza ne bude unazadovana. Niže frekvencije u nizu ostavljamo kakve jesu, a visoke frekvencije zamjenjujemo nulama te se provodi operacija inverzne Fourierove transformacije kako bi se dobio rezultat u vremenskoj domeni. Jedan od mogućih prikaza se nalazi na slici 3.1, gdje plave linije predstavljaju mnogokut dobiven generiranjem slučajnih točaka, a narančastom linijom je prikazan rezultat interpolacije.



Slika 3.1 Rezultat interpolacije

3.2. Generiranje pozadine

Pozadinska slika sastoji se od nasumičnog broja crvenih krvnih stanica koje se mogu ali i ne moraju preklapati.

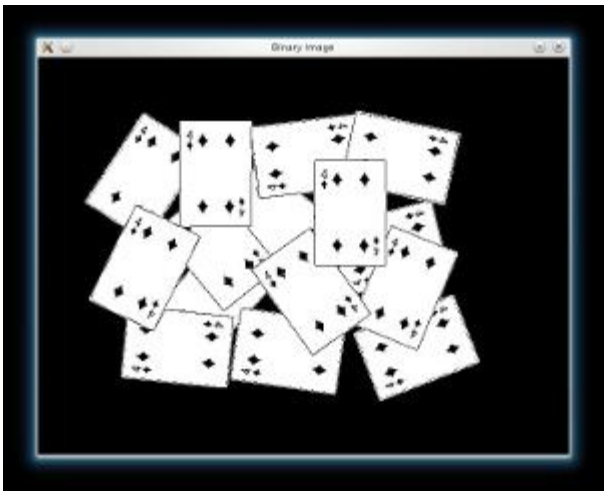


Slika 3.2 Eritrociti [19]

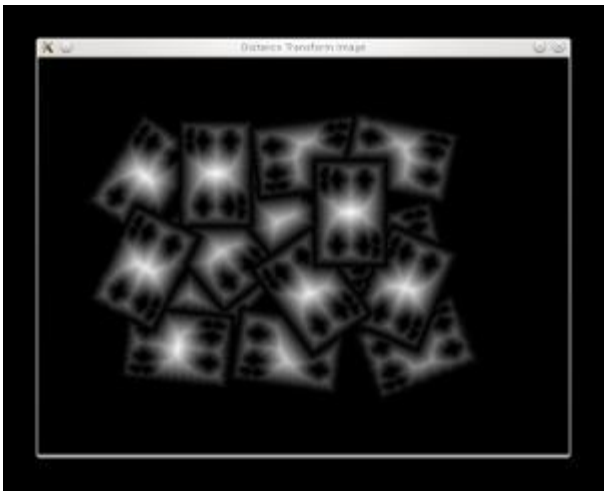
Slika 3.2 *Eritrociti* [19] prikazuje jednu od mogućih slika krvnih stanica dobivenih mikroskopom. Ostvarivanjem mogućnosti generiranja zasebnih krvnih zrnaca otvara se mogućnost generiranja ukupne pozadine jer je u pitanju samo postavljanje crvenih krvnih zrnaca na praznu sliku.

3.2.1. Generiranje i bojanje crvenih krvnih stanica

Za generiranje obruba stanica koristi će se prijašnje opisana funkcija u poglavlju *Generiranje obruba stanice*. Problem predstavlja bojanje stanice, jer eritrociti su u sredini svjetliji, a svjetlina opada što se više približavamo rubu stanice. Iz tog razloga, za bojanje je savršeno koristiti *transformaciju po udaljenosti* [20]. U srži, transformacija po udaljenosti iz binarne slike za svaku točku unutar oblika računa njezinu udaljenost od pozadine slike. Jedan od primjera transformacije po udaljenosti se nalazi na slici 3.3 i slici 3.4.



Slika 3.3 Binarna slika [21]



Slika 3.4 Transformacija po udaljenosti [21]

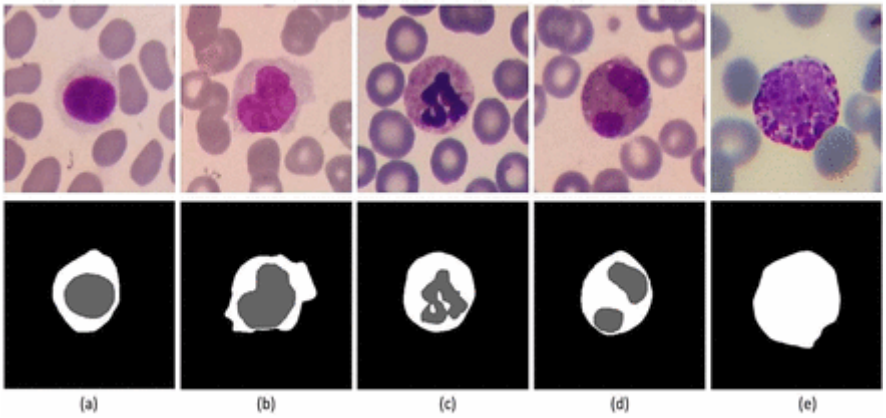
Dobivena je slika gdje je zrnce svjetlije što smo bliži sredini pa je takvo jedino još dovoljno obojati odgovarajućim bojama. Jedno od mogućih rješenja je korištenjem *alpha* kanala za modulaciju prozirnosti krvnog zrnca, no u ovom radu će se koristiti bojanje gradijentom boja koji obuhvaća *RGB* vrijednosti između najsvjetlije vrijednosti krvnog zrnca do najtamnije vrijednosti krvnog zrnca.

3.2.2. Spajanje ukupne pozadine

Za dobivanje ukupne pozadine dovoljno je samo postaviti nasumičan broj krvnih zrnaca na nasumične točke slike kako bi se dobio krajnji rezultat. Jedan od problema je bio slučaj u skupu podataka da se dijelovi krvnog zrnca nalaze izvan rubova slike. Taj problem je jednostavno rješiv tako da se u početku generira nešto veća slika od željene, unutar nje se postavljaju krvna zrnca te nakon toga se slika odreže u željenu veličinu.

3.3. Generiranje monocita

Za generiranje oblika bijelih krvnih stanica treba uzeti u obzir njihov oblik.

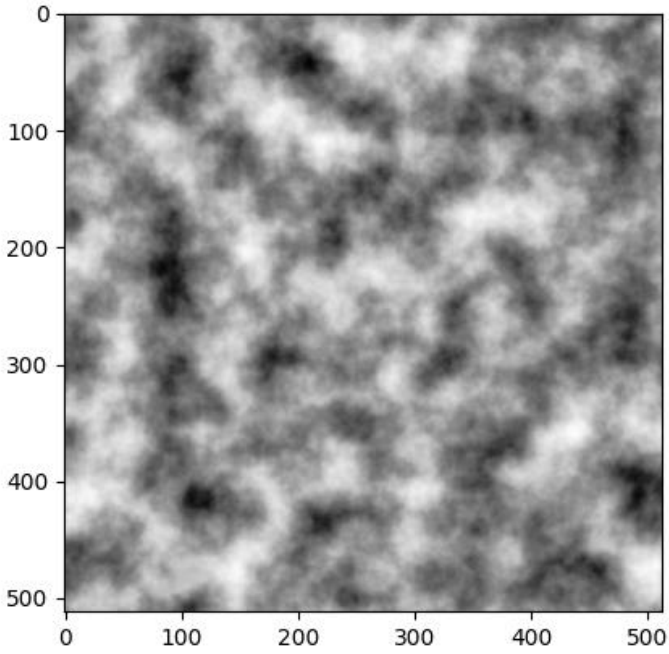


Slika 3.5 Tresholdane stanice (a) limfociti, (b) monociti, (c) neutrofili, (d) eozinofili, (e) bazofili [11]

Slika 3.5 pokazuje oblike stanica dobiveno pomoću posebne programske podrške *Easy-GT* [22]. Iz slike je vidljivo da jezgra stanice monocita nalikuje na oblike kakve dobijemo funkcijom za stvaranje kontura stanica. Citoplazma se može stvoriti tako da kopira kontura dobivena za jezgru monocita te se ona skalira na nešto veću vrijednost kako bi površinski bila veća od jezgre stanice. Nakon toga, za bojanje stanice koristi se *papar šum* tako da je šum intenzivniji za sliku citoplazme nego za sliku jezgre stanice. Bojanje se izvodi opet korištenjem gradijenta boja, posebno za jezgru, posebno za citoplazmu te se nakon toga slike spajaju tako da se slika jezgre monocita postavlja u sliku citoplazme monocita tako da su im geometrijski centri jednaki.

3.4. Generiranje neutrofila

Neutrofili imaju nešto složeniju građu jezgre stanice (Slika 3.5), stoga je i njihovo generiranje složenije. Za početak, generiranje citoplazme je zamalo pa identično kao kod monocita, jedino se razlikuje bojanje citoplazme, potrebno je koristiti druge vrijednosti gradijenta boja kako bi se dobila zornija slika. Jezgra neutrofila se uglavnom sastoji od više povezanih segmenata te je problem generiranja istih riješen tako da se generira nasumičan broj popunjenih konturi pomoću funkcije za generiranje konture stanice. Stvara se veća slika koja će sadržavati sve te konture, te se ta slika dijeli na četiri kvadranta. U svakom kvadrantu postoji nasumična šansa hoće li biti popunjen konturom ili neće. Nakon što se generira slika, donju polovicu slike pomiče se prema gore, a desnu polovicu pomiče se ulijevo za određen broj točaka tako da se konture preklapaju. Bojanje jezgre stanice izvedeno je pomoću fraktalnog šuma [23] jer svojim izgledom podsjeća na izgled jezgre neutrofila (Slika 3.6).



Slika 3.6 Fraktalni šum

Dobivenu sliku se boja pomoću gradijenata boja te se slika jezgre neutrofila postavlja u sliku citoplazme neutrofila.

3.5. Generiranje ukupne slike

Ukupna slika se generira tako da se prvo generira pozadina s crvenim krvnim stanicama. Nakon toga se, ovisno o tome generira li se slika za neutrofil ili monocit, u nasumične koordinate slike postavlja odgovarajuća generirana slika bijele krvne stanice, uzimajući u obzir veličinu same slike bijele krvne stanice. Potrebno je još vanjski zapisati veličine i lokacije stanice na slikama za potrebe treniranja neuronske mreže.

4. PROGRAMSKO RJEŠENJE ZADATKA

Ovim poglavljem će biti opisano umjetno stvaranje slike stanica te stvaranje i treniranje umjetne neuronske mreže za klasifikaciju dobivenih slika.

4.1. Generiranje konture stanice

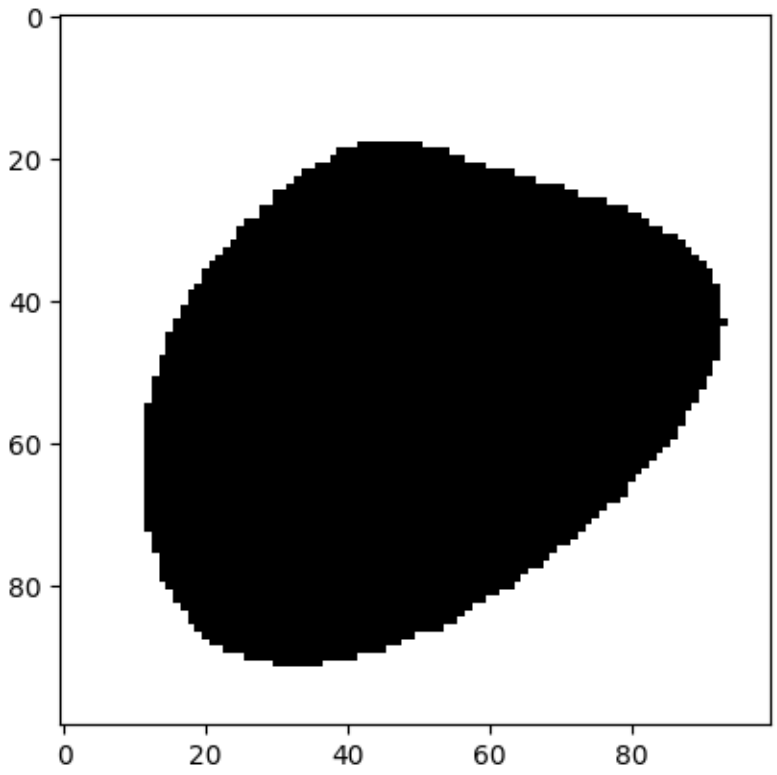
Za stvaranje slike stanice veliku važnost nam predstavlja mogućnost generiranja konture koja predstavlja obrub stranice. Programski kod 4.1 predstavlja kod za stvaranje iste.

```
def generate_blob_image(size=100):
    pts = [(random() + 0.8) * cmath.exp(2j*cmath.pi*i/7) for i in range(7)]
    pts = convexHull([(pt.real, pt.imag) for pt in pts])
    xs, ys = [interpolateSmoothly(zs, 30) for zs in zip(*pts)]

    # prazna slika size x size
    img = np.zeros((size, size))

    # prebacivanje u koordinate
    points = np.array(list(zip(xs, ys)))

    # normaliziranje da bude izmedju 0 i 1
    points += 2
    points /= 2 + 2

    # skaliranje na velicinu slike
    points *= size

    # openCV format
    points = points.astype(np.int32)
    points = points.reshape((-1, 1, 2))

    cv.fillPoly(img, [points], color=255)
    return img
```

Programski kod 4.1 funkcija za generiranje obruba

Funkcija započinje generiranjem sedam slučajnih kompleksnih brojeva (*pts*) na jediničnom krugu. Ovi brojevi su ravnomjerno raspoređeni oko kruga. Zatim se računa konveksni omotač (*convex hull*) tih točaka kako bi se stvorio konveksni poligon. Nakon toga, vrši se glatka interpolacija (*interpolateSmoothly*) između točaka konveksnog omotača kako bi se generirao niz x i y koordinata (*xs* i *ys*) koje definiraju granicu oblika nalik na kapljicu. Potrebno je stvoriti praznu

sliku pomoću *numpy* niza, veličine zadane parametrom funkcije (*size)*. Dobivene *x* i *y* koordinate normaliziraju se kako bi bile između 0 i 1, a zatim se skaliraju kako bi odgovarale veličini slike. Nakon toga, normalizirane i skalirane koordinate pretvaraju se u cijele brojeve i raspoređuju kao točke u formatu očekivanom od strane *OpenCV*-a (*cv.fillPoly*). Rezultirajuće točke tvore zatvoreni poligon. Konačno, funkcija *cv.fillPoly* ispunjava poligon bijelom bojom (vrijednost piksela 255) na crnoj slici, stvarajući ispunjen oblik. Rezultat te funkcije prikazan je slikom 4.1 i služi kao temelj za generiranje slika u ovom radu.



Slika 4.1 Izlaz funkcije generate blob

Programski kod 4.2 prikazuje definiciju funkcije *convexHull* koja za parametar prima listu točaka. Funkcija za početak prima popis točaka *pts* koje će biti korištene za stvaranje konveksnog omotača pomoću korištenja metode *Grahamovog skeniranja* [17]. Na početku se traži krajnja lijeva točka među ulaznim točkama (*xleftmost, yleftmost*). Ta točka će biti jedna od točaka na konveksnom omotaču. Zatim izračunava polarni kut svih točaka u odnosu na krajnju lijevu točku. Funkcija *atan2* koristi se za izračun kuta i stvara listu *by_theta* koja sadrži uređene parove s kutom, x-koordinatom i y-koordinatom svake točke.

```
def convexHull(pts):    #Graham's scan.
    xleftmost, yleftmost = min(pts)
    by_theta = [(atan2(x-xleftmost, y-yleftmost), x, y) for x, y in pts]
    by_theta.sort()
    as_complex = [complex(x, y) for _, x, y in by_theta]
    chull = as_complex[:2]
    for pt in as_complex[2:]:
        #Perp product.
        while ((pt - chull[-1]).conjugate() * (chull[-1] -
chull[-2])).imag < 0:
            chull.pop()
        chull.append(pt)
    return [(pt.real, pt.imag) for pt in chull]
```

Programski kod 4.2 convexHull funkcija

Također, za interpolaciju između točaka koristit će se diskretna Fourierova transformacija prikazana na programskom kodu 4.3.

```
def dft(xs):
    return [sum(x * cmath.exp(2j*pi*i*k/len(xs))
                for i, x in enumerate(xs))
            for k in range(len(xs))]
```

Programski kod 4.3 Diskretna Fourierova transformacija

Jednadžbom 3-1 prikazana je formula za diskretnu Fourierovu transformaciju koja je prevedena u programski kod. Na početku je definirana Python funkcija nazvana *dft* koja prima listu *xs* kao ulaz, što predstavlja diskretni niz koji treba transformirati. Funkcija vrši obradu liste točaka *xs* tako da računa diskretnu Fourierovu transformaciju svake od točaka liste te vraća listu kompleksnih brojeva koji predstavljaju ulazne točke u frekvencijskoj domeni. Linija programskog koda:

*„sum(x * cmath.exp(2j*pi*i*k/len(xs)) for i, x in enumerate(xs))“* predstavlja računanje pojedinog elementa DFT-a. Iterira kroz elemente ulaznog niza *xs*, i za svaki element *x* na indeksu *i* računa produkt *x* i kompleksnog eksponencijalnog člana *cmath.exp(2j*pi*i*k/len(xs))*. Taj član predstavlja sinusoidni val pri određenoj frekvenciji *k* u DFT-u. Vanjska iteracija liste prolazi kroz sve moguće frekvencije *k* od 0 do količine ulaznih podataka minus jedan, računajući DFT koeficijente za svaku frekvenciju.

Za mekanu interpolaciju među točkama korištena je funkcija prikazana na programskom kodu 4.4 *interpolateSmoothly* funkcija.

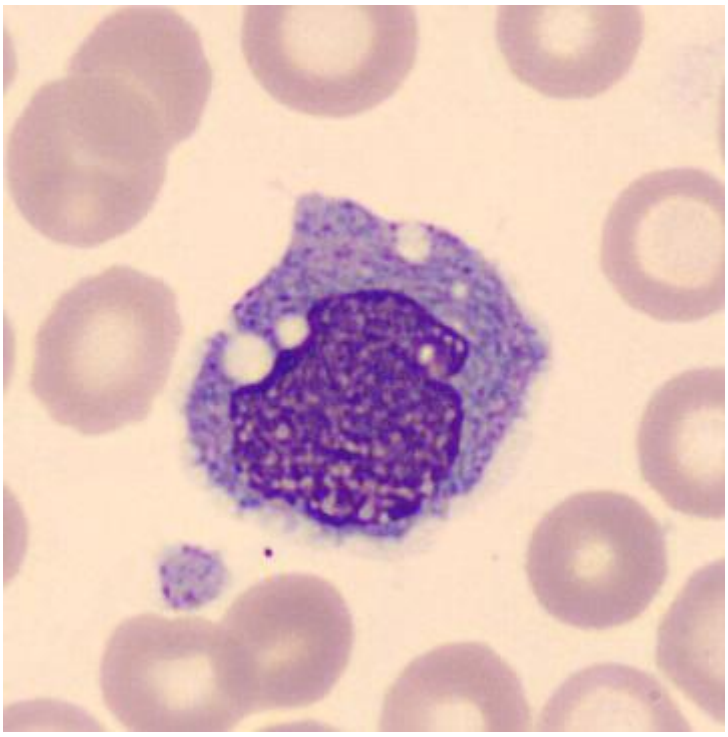
```
def interpolateSmoothly(xs, N):
    """For each point, add N points."""
    fs = dft(xs)
    half = (len(xs) + 1) // 2
    fs2 = fs[:half] + [0]*(len(fs)*N) + fs[half:]
    return [x.real / len(xs) for x in dft(fs2)[::-1]]
```

Programski kod 4.4 interpolateSmoothly funkcija

Programski kod 4.4 predstavlja implementaciju funkcije *interpolateSmoothly* koja za parametre prima listu točaka *xs* te cjelobrojnu vrijednost *N*. Svrha te funkcije je interpolacija između danih točaka stvarajući više točaka između dvaju susjednih točaka. Za početak, funkcija pretvara ulazne vrijednosti *xs* u frekvencijsku domenu pomoću diskretne Fourierove transformacije. Nadalje, računa se indeks vrijednosti koja predstavlja polovinu ulazne liste točaka xs, što će se koristiti za podjelu rezultata diskretne Fourierove transformacije na dva dijela. Nakon toga, stvara se nova lista *fs2* koja proširuje rezultat DFT-a *fs* s dodanim nulama. U srži zapravo udvostručuje veličinu DFT-a dodavajući nule u sredini liste. Taj postupak je potrebno izvršiti radi potreba interpolacije. Nadalje, funkcijom *dft(fs2)* računa se inverzna diskretna Fourierova transformacija sada prilagođene sekvence točaka *fs2*. Inverzna DFT vraća podatke iz frekvencijske domene natrag u vremensku domenu. Dijeljenjem realnog dijela kompleksne točke s količinom točaka liste omogućava se normaliziranje vrijednosti. Na kraju, se obrće redoslijed elemenata liste kako bi se omogućila mekana interpolacija točaka.

4.2. Generiranje pozadine

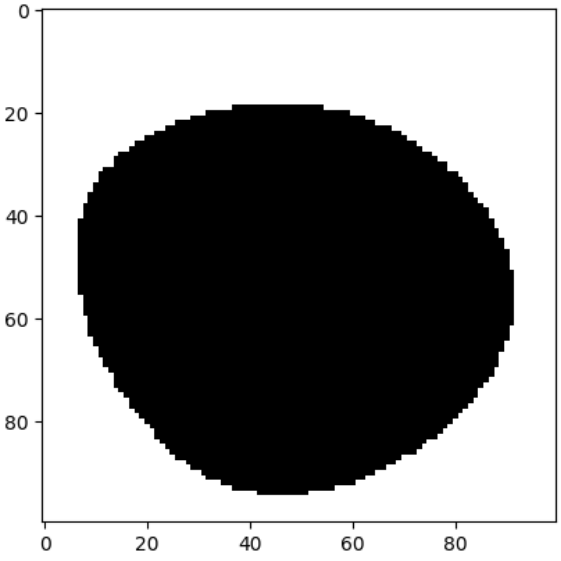
Na pozadinu se odnosi sve osim bijelih krvnih stanica, što u ovom slučaju predstavlja crvene krvne stanice i jednobojno pozadinsko osvijetljene. Dobar primjer pozadine koja će biti umjetno oponašana u ovom poglavlju je prikazan na slici 4.2 *stanica monocita*.



Slika 4.2 stanica monocita

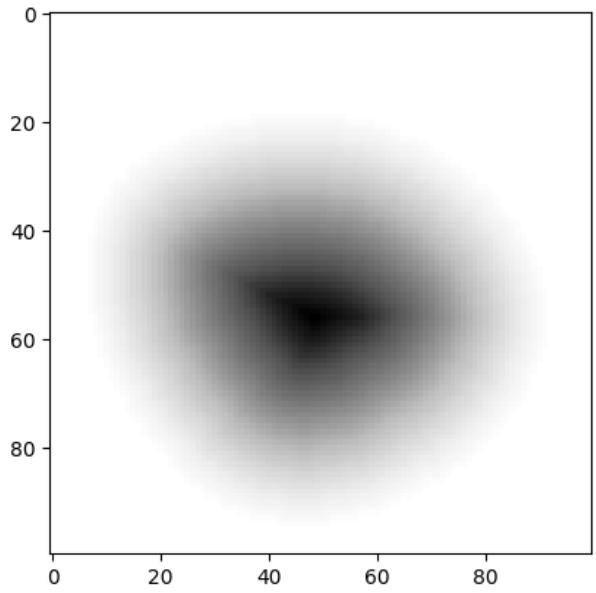
4.2.1. Generiranje crvenih krvnih stanica

Crvene krvne stanice pod mikroskopom jednostavnog su oblika i izgleda stoga njihovo generiranje nije zahtjevan programski poduhvat. Specifične su po svom eliptičnom obliku te po tome što su svjetlije u sredini. Za generiranje oblika stanice koristit ćemo *Programski kod 4.1 funkcija za generiranje obruba*.



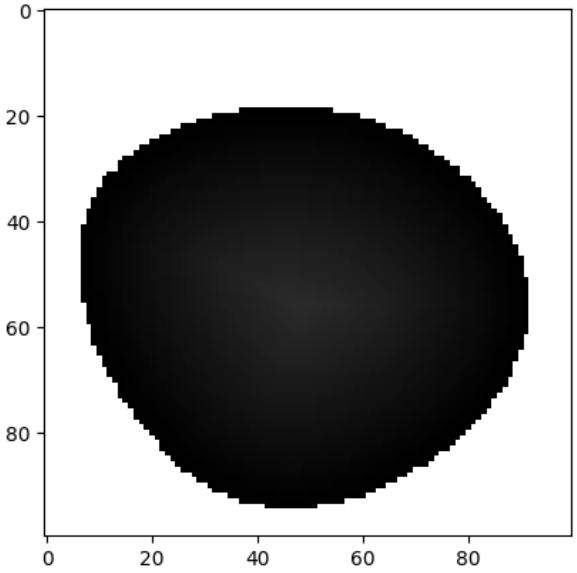
Slika 4.3 stanica

Izlaz funkcije je prikazan slici 4.3. Da bi se dobio oblik u kojem je sredina stanice svjetlija od ruba, koristit će se transformacija po udaljenosti (eng. *distance transform*). Izlaz te funkcije na ulaznom skupu nalazi se na slici 4.4.



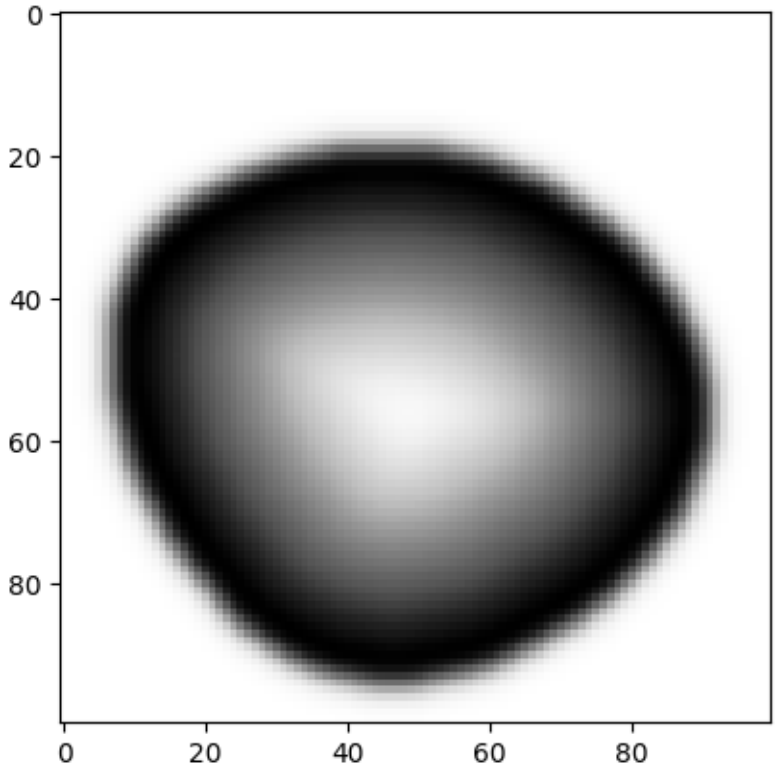
Slika 4.4 distance transform stanice

Sada je samo potrebno oduzeti sliku dobivenu *distance transforomom* od naše ulazne slike da bi se dobio željeni oblik slike (Slika 4.5).



Slika 4.5 rezultat oduzimanja DT-a od slike stanice

Nakon dodatne obrade normaliziranja podataka na vrijednosti između nule i jedinice, te primjene Gaussovog šuma, dobiva se sljedeći izlaz (Slika 4.6).



Slika 4.6 obrađena slika stanice

Programski kod 4.5 *obrada slike stanice* prikazuje upravo taj postupak u programskom rješenju.

```
def transform_blob(img):
    img2 = distance_transform_edt(img)
    img = img - img2
    img[img != 0] = (img[img != 0] - np.min(img[img != 0])) /
(np.max(img[img != 0]) - np.min(img[img != 0]))
    img = gaussian_filter(img, sigma=2)
    return img
```

Programski kod 4.5 obrada slike stanice

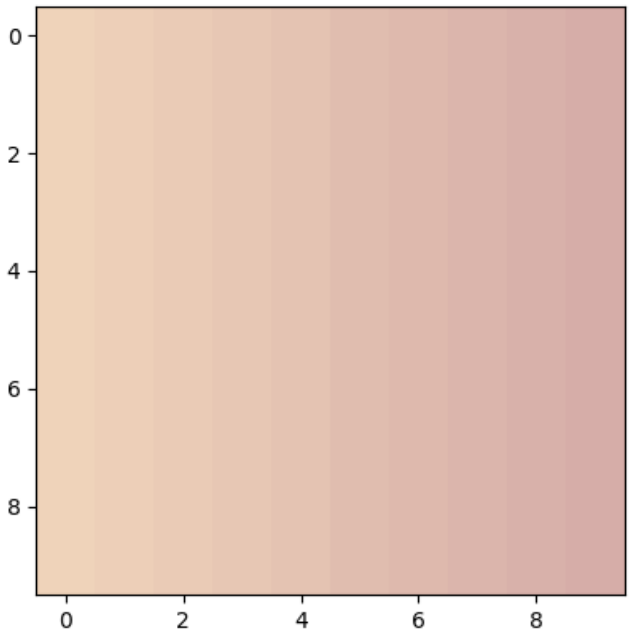
Nadalje, da bi postupak generiranja crvenih krvnih stanica bio dovršen, potrebno je stanice dobivene sa slike 4.6 *obrađena slika stanice* obojati prikladnom bojom. Za realizaciju bojanja, korišten je alat biblioteke *matplotlib* imena *colormap*. Alat omogućava generiranje vlastitih gradijenata boja te korištenje istih za bojanje crno-bijelih slika. Programski kod 4.6 *Generiranje palete boja* pruža programsko rješenje problema stvaranja vlastite palete boja. Boje su

predstavljene u RGB obliku u formatu tipa podatka s pomičnim zarezom s vrijednostima koje se kreću od nule do jedinice.

```
colors = [(0.94,0.83,0.73), (0.84,0.68,0.66)]
cellcmap = LinearSegmentedColormap.from_list(
        "Custom", colors, N=20)
```

Programski kod 4.6 Generiranje palete boja

Kada se taj gradijent boja prikaže pomoću funkcije *plt.imshow()* izgleda kao na slici 4.7 *gradijent boja krvi*.

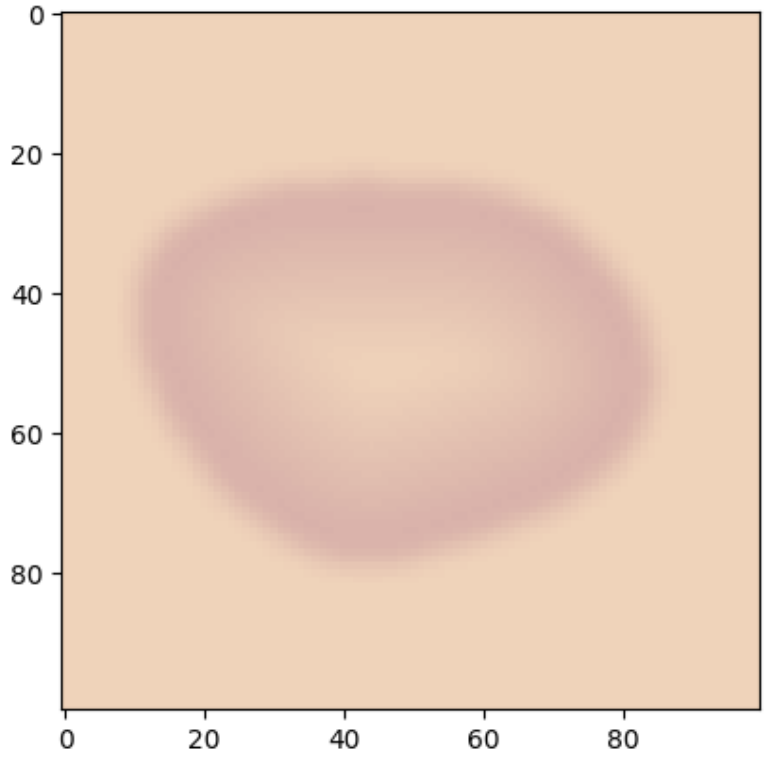


Slika 4.7 gradijent boja krvi

Pokretanje programskog koda 4.7 na oblik dobiven slikom 4.6 *obrađena slika stanice* dobija se izlaz prikazan slikom 4.8 *obojana krvna stanica*.

```
def color_blob(img):
    #grayscale_blob = img.copy()
    img = transform_blob(img)
    img = cellcmap(img)
    img = img[:,:,:3] #cut alpha channel
    # Modify img where blob has a value of 0
    img[np.all(img == (0.94,0.83,0.73), axis=-1)] = [0, 0, 0]  # Setting
background to zero
    return img
```

Programski kod 4.7 Bojanje krvne stanice



Slika 4.8 obojana krvna stanica

Potrebno je još postaviti sve pozadinske vrijednosti na nulu da bi se krvna stanica mogla lakše postaviti na pozadinu s ostalim krvnim stanicama.

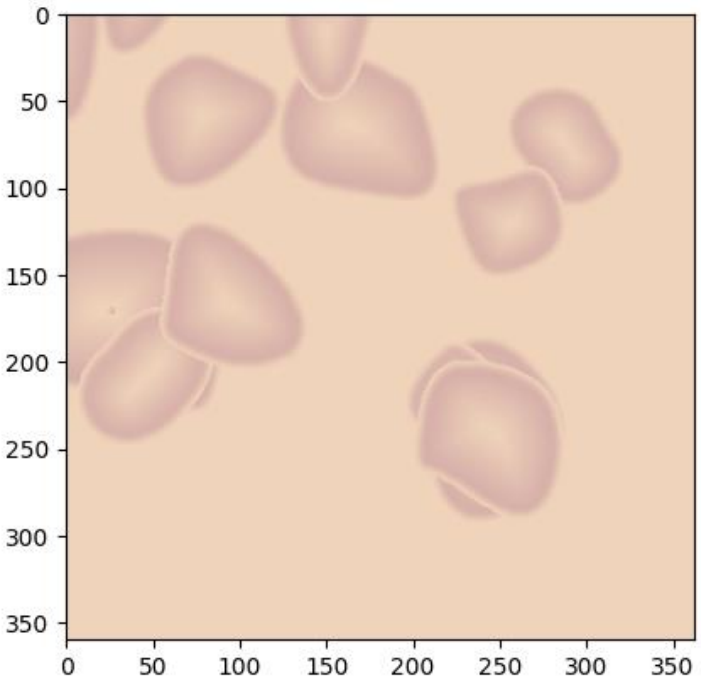
4.2.2. Generiranje ukupne pozadine

Za stvaranje ukupne pozadine, generirat će se nasumični broj krvnih stanica te će se one postavljati na nasumične koordinate pozadinske slike. Programski kod 4.8 generiranje pozadinske slike pokazuje upravo programsko rješenje problema generiranja ukupne pozadine. Funkcija *generate_background_image()* ne prima parametre, već su predefinirane maksimalne i minimalne vrijednosti veličine krvne stanice. Stvara se slika veličine 360 x 363 s dodatnim prostorom u obje dimenzije jednak vrijednosti maksimalne veličine krvne stanice, popunjena vrijednostima boje pozadine, u ovom slučaju, svijetlo narančasta. Dodatni prostor omogućava da se dijelovi krvnih stanica nalaze izvan granica krajnje slike, što dodatno pospješuje vjerodostojnost same slike. Pomoću biblioteke *random* generira se slučajna cjelobrojna vrijednost koja određuje broj krvnih stanica na slici. Nadalje, petlja se izvršava *N*-puta, gdje *N* predstavlja broj krvnih stanica. U svakoj iteraciji petlje generira se slučajna vrijednost veličine krvne stanice, njene koordinate na slici te se pomoću tih vrijednosti generiranju slike krvnih stanica i postavljaju na sliku, pazeći pritom da

zanemarimo pozadinu slike svake stanice zasebno korištenjem maski. Na samom kraju, obrezuje se dodatni prostor i vraća se slika 4.9 RGB formata veličine 360 x 363.

```
def generate_background_image():
    # prazna slika size x size
    max_blob_size = 140
    min_blob_size = 60
    img = np.full((360 + max_blob_size, 363 + max_blob_size,
3),(0.94,0.83,0.73))
    num_of_blobs = randint(10, 20)
    for i in range(0, num_of_blobs):
        size = randint(min_blob_size, max_blob_size)
        centre_x = randint(0, 360 + max_blob_size - size)
        centre_y = randint(0, 363 + max_blob_size - size)
        blob = generate_blob_image(size)
        blob = color_blob(blob)

        # Binary mask to ignore zeros in the blob image
        mask = np.all(blob != [0,0,0], axis=-1)

        img[centre_x:centre_x+size, centre_y:centre_y+size][mask] =
blob[mask]

    return img[max_blob_size::,max_blob_size::,::]
```

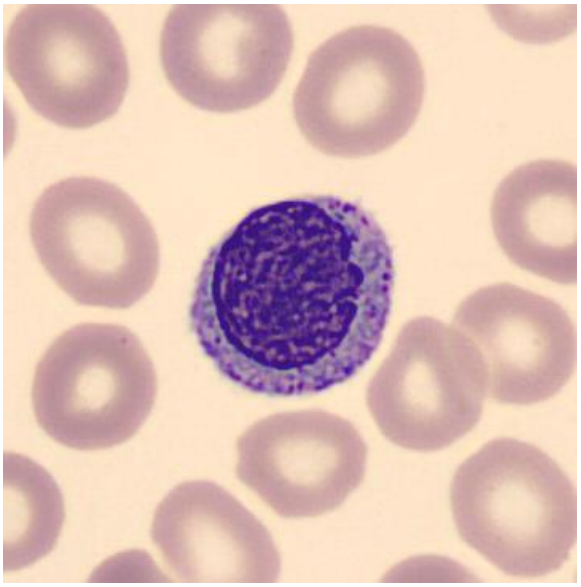
Programski kod 4.8 generiranje pozadinske slike



Slika 4.9 Pozadina

4.3. Generiranje monocita

Generiranje stanice monocita bit će opisano ovim potpoglavljem.



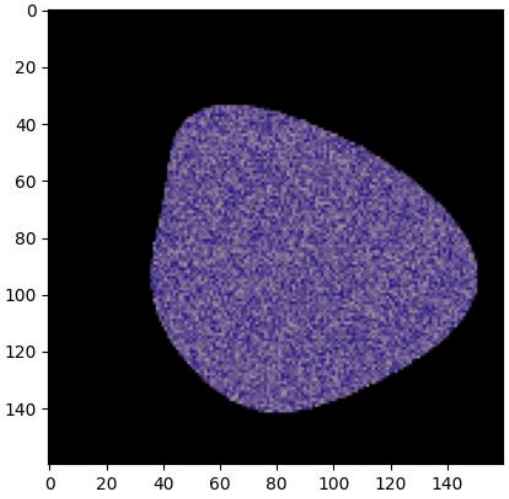
Slika 4.10 monocit

Programsko rješenje stvaranja umjetne slike stanice monocita prikazano je kodom 4.9.

```
def generate_monocyte(size = 512):
    img = generate_blob_image(size)
    blob = img.copy()
    underlayer = rescale(img, 1.25, anti_aliasing=False)
    underlayerblob = underlayer.copy()
    underlayer = random_noise(underlayer, 'pepper', amount=0.5)
    underlayer = monocytecmap(underlayer)
    underlayer = underlayer[:,:,:3]
    underlayer[underlayerblob==0] = [0,0,0]
    underlayer = rescale(underlayer, 0.25, channel_axis=2,
anti_aliasing=False)
    img = random_noise(img, 'pepper', amount=0.2)
    img = monocytecmap(img)
    img = img[:,:,:3]
    img[blob==0] = [0,0,0]
    img = rescale(img, 0.25, channel_axis=2, anti_aliasing=False)
    difference = underlayer.shape[0]-img.shape[0]
    difference = int(difference/2)
    mask = np.all(img != [0,0,0], axis=-1)
    underlayer[difference:img.shape[0]+difference,difference:img.shape[0]+
difference][mask] = img[mask]
    underlayer = gaussian_filter(underlayer, sigma=0.5)
    return underlayer
```

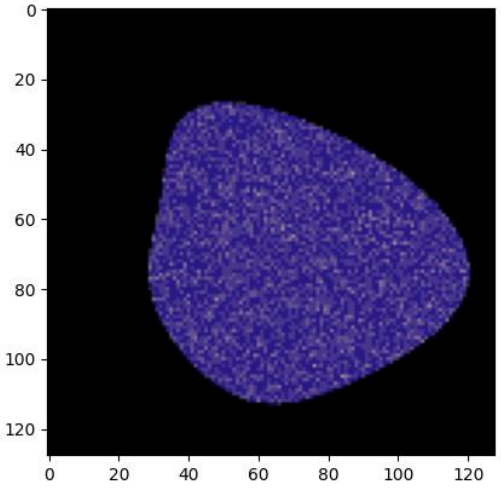
Programski kod 4.9 generiranje slike monocita

Prvo se generira kontura koja će predstavljati temelj naše stanice, odnosno obrub jezgre stanice. Varijabla *underlayer* predstavlja citoplazmu stanice i u ovom slučaju će biti slična jezgri stanice, ali za četvrtinu veća. Na sliku citoplazme ćemo primijeniti *papar šum* u omjeru jedan naprema dva. Pomoću prije generirane palete boja za monocit, boja se slika *underlayer*, odnosno citoplazma te se dobije sljedeća slika 4.11 *citpolazma*.



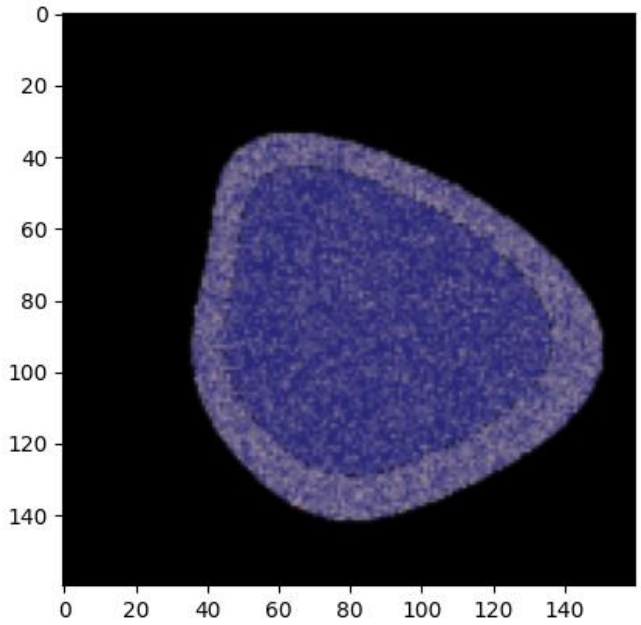
Slika 4.11 citpolazma

Nakon toga, vrijednosti izvan citoplazme se postavljaju na nulu i briše se *alpha* kanal, a slika se skalira na veličinu prikladnu za izlaz. Slična je procedura za generiranje jezgre stanice monocita, ali razlika je u tome što je količina papar šuma u ovom slučaju manja, jedna petina slike će biti izmijenjena šumom.



Slika 4.12 jezgra monocita

Nakon bojanja i skaliranja, sve vrijednosti slike jezgre stanice se postavljaju u središte slike citoplazme te se dobije krajnji rezultat, slika 4.13.



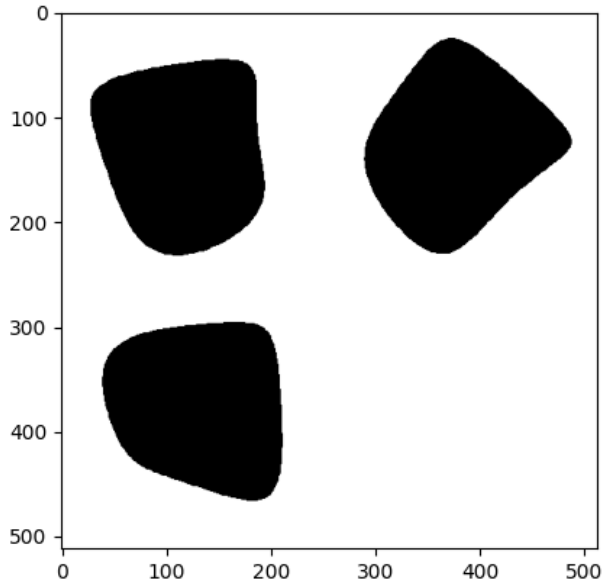
Slika 4.13 monocit

4.4. Generiranje neutrofila

Za stvaranje umjetne slike neutrofila koristit će se funkcija za generiranje obruba stanice te fraktalni šum.

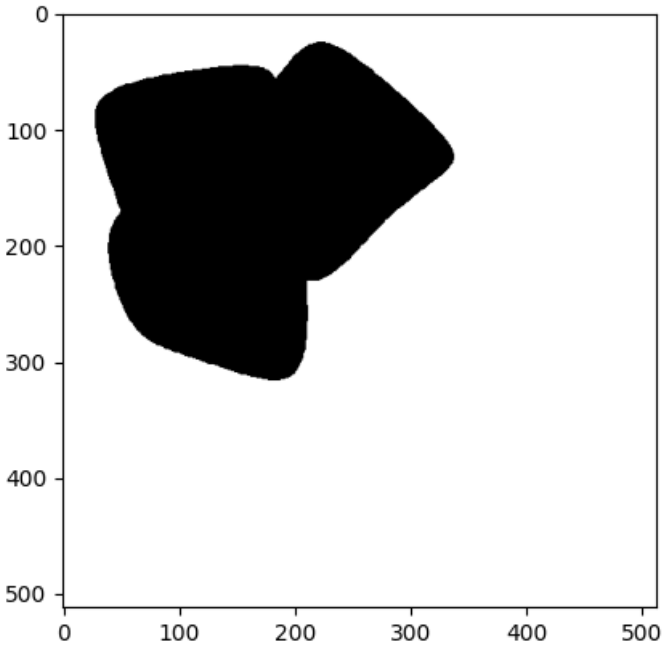
4.4.1. Generiranje jezgre

Neutrofili imaju specifičnost po tome što im je jezgra često segmentirana, nerijetko u obliku slova *S* ili *C*. Kreiranje jezgre je ostvareno programskim kodom 4.10 *Stvaranje jezgre neutrofila*. Glavna ideja je napraviti praznu sliku veličine 512 x 512, te tu sliku podijeliti na četiri jednaka kvadranta. Nakon toga, nasumično se biraju cjelobrojne vrijednosti *i* i *j* koje označavaju kvadrante u kojima će se crtati generirana kontura. Moguće je da se nasumične vrijednosti preklapaju više puta u različitim iteracija petlje, što će za posljedicu imati da će količina nacrtanih kontura varirati između jedne, dvije, tri ili četiri konture. Nakon što se petlja izvrši, moguć izgled dobivene slike prikazan je na slici 4.14 slika nakon izvršavanja petlje



Slika 4.14 slika nakon izvršavanja petlje

Da bi se slike djelomično preklapale, jedno od mogućih rješenja je pomicanje desne strane slike ulijevo, te donje strane slike gore, ili obrnuto. U slučaju kao u Programski kod 4.10 Stvaranje jezgre neutrofila, vrši se pomicanje za 150 točaka desne strane ulijevo te 150 točaka donje strane prema gore. Dobiveni rezultat je prikazan slikom *4.15 slika nakon* spajanja.



Slika 4.15 slika nakon spajanja

```
def generate_neutrophil_nucleus():
    padding = 0
    img = np.full((512 + padding, 512 + padding), 0)
    blob_size = int(256)
    num_blobs = 4
    for _ in range(num_blobs):
        i = np.random.randint(0, 2)
        j = np.random.randint(0, 2)
        blob = generate_blob_image(blob_size)
        img[blob_size * i:blob_size * (i + 1), blob_size * j:blob_size *
(j + 1)] = blob
    mid_row = int(img.shape[0] / 2)
    mid_column = int(img.shape[1] / 2)
    non_zero_indices = np.where(img != 0)
    non_zero_values = img[non_zero_indices]
    # Get the bottom half of the image
    bottom_half_indices = (non_zero_indices[0] >= mid_row)
    bottom_half_values = non_zero_values[bottom_half_indices]
    # Shift the bottom half non-zero values up by 150 pixels
    shifted_indices = (non_zero_indices[0][bottom_half_indices] - 150,
non_zero_indices[1][bottom_half_indices])
    img[non_zero_indices[0][bottom_half_indices],
non_zero_indices[1][bottom_half_indices]] = 0  # Set non-zero values to
zero
    img[shifted_indices] = bottom_half_values  # Place non-zero values in
the shifted positions
    non_zero_indices = np.where(img != 0)
    non_zero_values = img[non_zero_indices]
    # Get the right half of the image (you can adjust 'mid_column'
accordingly)
    right_half_indices = (non_zero_indices[1] >= mid_column)
    right_half_values = non_zero_values[right_half_indices]

    # Shift the right half non-zero values left by 150 pixels
    shifted_indices = (non_zero_indices[0][right_half_indices],
non_zero_indices[1][right_half_indices] - 150)
    img[non_zero_indices[0][right_half_indices],
non_zero_indices[1][right_half_indices]] = 0  # Set non-zero values to
zero
    img[shifted_indices] = right_half_values  # Place non-zero values in
the shifted positions
    zero_rows = np.all(img == 0, axis=1)
    zero_cols = np.all(img == 0, axis=0)
    # Remove zero rows and columns
    img = img[~zero_rows][:, ~zero_cols]
    return img
```

Programski kod 4.10 Stvaranje jezgre neutrofila

Na samom kraju, sa slike se brišu svi redovi i stupci koji sadržavaju samo vrijednosti jednake nuli, te funkcija vraća vrijednosti te slike. Bojanje jezgre implementirano je putem programskog koda 4.11 *Bojanje jezgre neutrofila*

```
def color_neutrophil_nucleus(img):
    blob = img.copy()
    noise = generate_fractal_noise_2d((512, 512), (8, 8), 5)
    # Crop the noise to match the size of img
    noise = noise[:img.shape[0], :img.shape[1]]
    noise = (noise-np.min(noise))/(np.max(noise)-np.min(noise)) #normalize
the noise from 0 to 1

    img = noise
    img = cm(img)
    img = img[:,:,:3] #cut alpha channel
    # Modify img where blob has a value of 0
    img[blob == 0] = [0, 0, 0]  # Setting background to zero
    #img = gaussian_filter(img, sigma=0.3)
    return img
```

Programski kod 4.11 Bojanje jezgre neutrofila

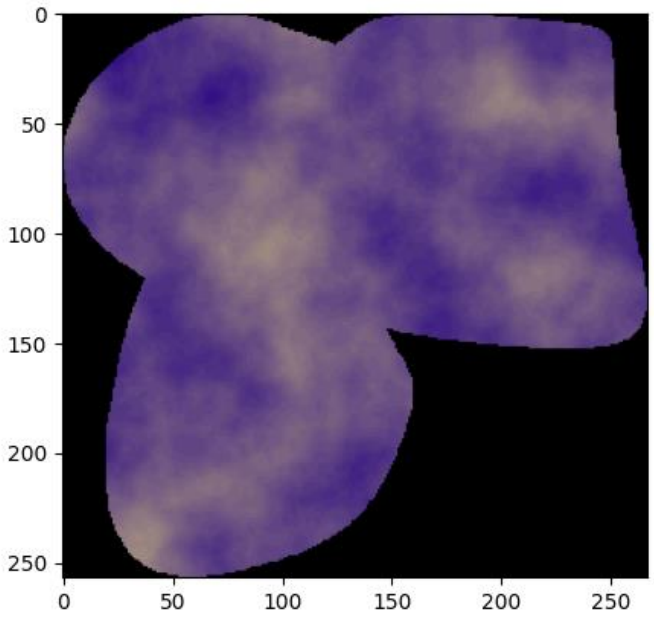
Primjerom koda 4.12 prikazano je programsko rješenje bojanja jezgre neutrofila.

```
def color_neutrophil_nucleus(img):
    blob = img.copy()
    noise = generate_fractal_noise_2d((512, 512), (8, 8), 5)
    plt.imshow(noise, cmap="Greys")
    plt.show()
    # Crop the noise to match the size of img
    noise = noise[:img.shape[0], :img.shape[1]]
    noise = (noise-np.min(noise))/(np.max(noise)-np.min(noise)) #normalize
the noise from 0 to 1

    img = noise
    img = cm(img)
    img = img[:,:,:3] #cut alpha channel
    # Modify img where blob has a value of 0
    img[blob == 0] = [0, 0, 0]  # Setting background to zero
    #img = gaussian_filter(img, sigma=0.3)
    return img
```

Programski kod 4.12 Bojanje jezgre neutrofila

Generira se jedinstvena slika fraktalnog šuma veličine 512 x 512 točaka tako da se po svakoj osi generira po osam perioda šuma s pet oktava. Nakon toga, slika šuma se obrezuje na veličinu tražene slike te se ista normalizira na vrijednosti između nule i jedinice. Slika šuma se boja koristeći predefiniran gradijent boja te se, koristeći vrijednosti prije generirane konture jezgre

stanice, sve vrijednosti izvan konture postavljaju na nule. Funkcija vraća moguć izlaz vidljiv slikom 4.16.



Slika 4.16 Jezgra neutrofila

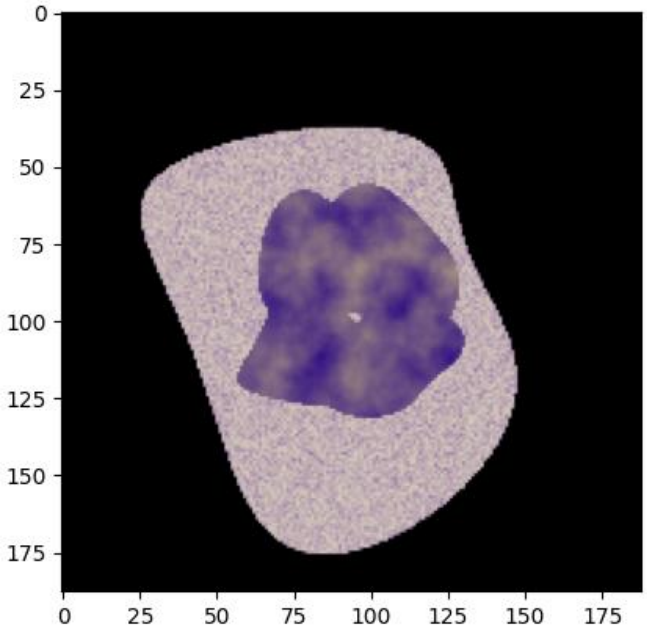
4.4.2. Stvaranje ukupne slike stanice neutrofila

Stvaranje ukupne slike stanice pokazuje programski kod 4.13.

```
def generate_neutrophil():
    neutrophil = generate_neutrophil_nucleus()
    neutrophil = color_neutrophil_nucleus(neutrophil)
    cell = generate_cell(750)
    non_zero_indices = np.where(neutrophil != 0)

    # Calculate the coordinates to place neutrophil in the middle of cell
    cell_height, cell_width = cell.shape[:2]
    neutrophil_height, neutrophil_width = neutrophil.shape[:2]

    y_start = (cell_height - neutrophil_height) // 2
    y_end = y_start + neutrophil_height
    x_start = (cell_width - neutrophil_width) // 2
    x_end = x_start + neutrophil_width
    # Copy neutrophil into the cell at the calculated coordinates
    cell[y_start:y_end, x_start:x_end][non_zero_indices] =
neutrophil[non_zero_indices]
    cell = rescale(cell, 0.25, channel_axis=2, anti_aliasing=False)
    return cell
```

Programski kod 4.13 Stvaranje slike stanice neutrofila

Najprije se generira i boja jezgra stanice neutrofila i sprema se u varijablu *neutrophil*. Nakon toga, stvara se slika citoplazme, veličine 750 x 750 točaka. Pohranjuju se sve vrijednosti slike *neutrophil* koje nisu jednake nuli u varijablu *non_zero_indices*. Nadalje, računaju se koordinate točaka za postavljanje slike jezgre neutrofila u citoplazmu neutrofila tako da obje slike imaju jednak geometrijski centar. Pomoću maske *non_zero_indices* omogućuje se da se sa slike jezgre na sliku citoplazme postavljaju samo obojane vrijednosti, čime izbjegavamo slučaj da crna pozadina slike narušava vjerodostojnost slike. Na samom kraju, cijela slika se skalira na četvrtinu svoje prvobitne veličine da bi je bilo moguće postaviti na krajnju sliku. Slika 4.17 *Umjetno generirani neutrofil* predstavlja izlaz funkcije *generate_neutrophil()* prikazan pomoću *matplotlib* biblioteke.



Slika 4.17 Umjetno generirani neutrofil

4.5. Generiranje ukupne slike s pozadinom

Stvaranje ukupne slike stanica neutrofila i monocita, zajedno s pozadinskom slikom krvnih zrnaca, bit će pojašnjeno u ovom potpoglavlju.

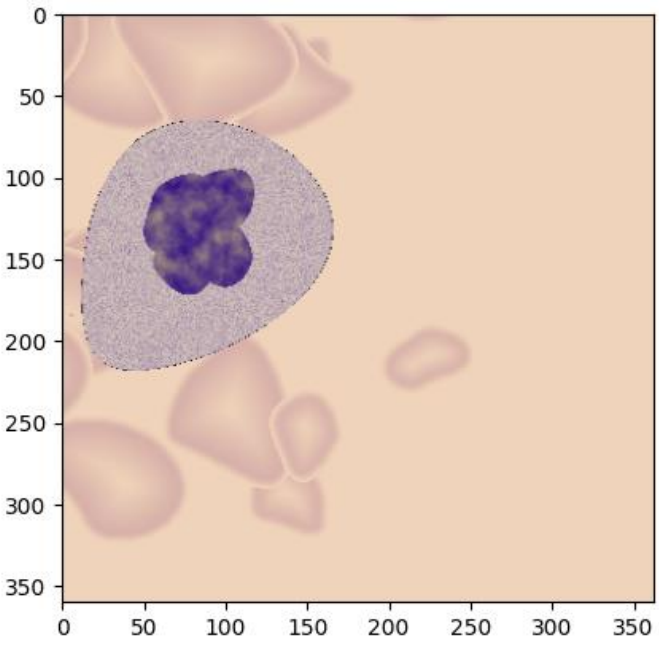
4.5.1. Ukupna slika monocita

Programski kod 4.14 *Generiranje ukupne slike monocita* prikazuje programsko rješenje upravo ovog problema. Za početak se stvara slika monocita te se od nje režu svi rubni stupci i redci koji samo sadržavaju vrijednosti jednake nuli. Nakon toga, generira se nasumičan cijeli broj koji predstavlja *x* i *y* koordinate gornje lijeve točke stanice monocita na pozadinskoj slici. Vrijednosti

su u rasponu od margine do veličine pozadinske slike umanjena za veličinu slike monocita. Margine su potrebne iz razloga da slika monocita ne bude postavljena baš uz sam rub slike, ponajviše zbog preglednosti. Nakon toga, stvara se pozadinska slika s krvnim stanicama, te se iz slike stanice monocita uzima maska svih vrijednosti koje nisu tamno-crna boja. Na kraju se slika monocita postavlja na prethodno generirane koordinate, a vraćaju se određene vrijednosti potrebne za treniranje neuronske mreže varijablom *yolobbox*, značenje istih bit će objašnjeno u sljedećim poglavljima.

```
def generate_monocyte_image():
    cell = generate_monocyte()
    cell = remove_null_rows_and_columns_rgb_image(cell)

    x = randint(10, 360-cell.shape[0]-10)
    y = randint(10, 363-cell.shape[1]-10)

    background = generate_background_image()

    mask = np.all(cell >= [0.05,0.05,0.05], axis=-1)

    background[x:x+cell.shape[0],y:y+cell.shape[1]][mask]=cell[mask]
    x = (x + cell.shape[0]/2) / background.shape[1]
    y = (y + cell.shape[1]/2) / background.shape[0]
    h = cell.shape[0]/background.shape[0]
    w = cell.shape[1]/background.shape[1]

    yolobbox=(y, x, w, h)
    return background, yolobbox
```

Programski kod 4.14 Generiranje ukupne slike monocita



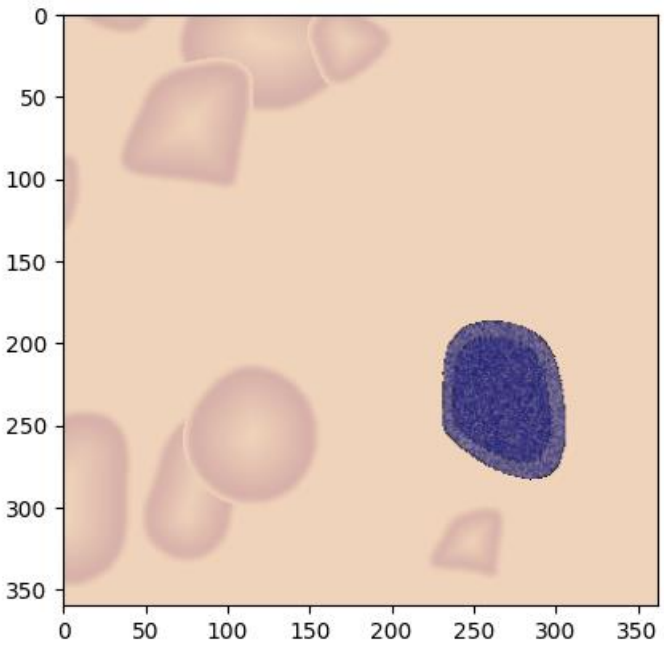
Slika 4.18 ukupna slika neutrofila

4.5.2. Ukupna slika neutrofila

Generiranje ukupne slike neutrofila je slično generiranju ukupne slike monocita što se može vidjeti programskim kodom 4.15 *Generiranje ukupne slike neutrofila*.

```
def generate_neutrophil_image():
    cell = generate_neutrophil()
    cell = remove_null_rows_and_columns_rgb_image(cell)
    x = randint(10, 360-cell.shape[0]-10)
    y = randint(10, 363-cell.shape[1]-10)
    print("x, y, w, h", x+cell.shape[0]/2, y+cell.shape[1]/2,
cell.shape[0], cell.shape[1])
    background = generate_background_image()
    mask = np.all(cell >= [0.05,0.05,0.05], axis=-1)
    background[x:x+cell.shape[0],y:y+cell.shape[1]][mask]=cell[mask]
    #background = gaussian_filter(background, sigma=0.5)
    #print(background.shape)
    x = (x + cell.shape[0]/2) / background.shape[1]
    y = (y + cell.shape[1]/2) / background.shape[0]
    h = cell.shape[0]/background.shape[0]
    w = cell.shape[1]/background.shape[1]

    yolobbox=(y, x, w, h)
    return background, yolobbox
```

Programski kod 4.15 Generiranje ukupne slike neutrofila



Slika 4.19 Ukupna slika monocita

4.6. TRENIRANJE NEURONSKE MREŽE

Ovim poglavljem bit će pojašnjena sama priprema podataka, postavljanje i treniranje neuronske mreže te analiza i komentar rezultata dobivenih istom. Koristit će se *YOLOv5* algoritam za prepoznavanje i klasifikaciju objekata u stvarnom vremenu u *PyTorch* okruženu. Za logiranje podataka i prikaz, koristit će se *CometML*.

4.6.1. Priprema podataka

Za početak je potrebno generirane slike i anotacije pretvoriti u oblik koji *YOLOv5 PyTorch* prihvaća.

Slika 4.20 Anotacije YOLOv5 podataka [24]

Slika 4.20 [24] predstavlja vizualnu reprezentaciju anotacije slika u YOLOv5 formatu. Za svaki objekt potrebno je upisati geometrijski centar u obje dimenzije te visinu i širinu pravokutnika koji definira veličinu objekta. Te vrijednosti je potrebno skalirati od nule do jedinice u odnosu na ukupnu veličine slike, čime se omogućava da, ako se provodi daljnje rastezanje ili slična obrada slike, ne dolazi do narušavanja anotacija. Programski kod 4.14 *Generiranje ukupne slike monocita* i Programski kod 4.15 *Generiranje ukupne slike neutrofila* varijablom *yolobbox* upravo postiže traženi rezultat, spremajući tražene vrijednosti u listu. Za svaku generiranu sliku treba postojati istoimena tekstualna datoteka koja sadrži podatke o anotacijama na način prikazan kodom 4.16 .

```
<redni broj klase> <x koordinata> <y koordinata> <širina pravokutnika> <dužina pravokutnika>
```

Programski kod 4.16 Format anotacija

Za *PyTorch* verziju *YOLOv5* algoritma potrebno je dodatno kreirati *.yaml* datoteku koja u sebi sadrži putanje do podataka za treniranje, testiranje i validaciju. Programski kod 4.17 data.yaml datoteka je primjer jedne od mogućih konfiguracija.

```
train: ../datasets/cells/train/images
val: ../datasets/cells/valid/images
test: ../datasets/cells/test/images

nc: 2
names: ['monocyte', 'neutrophil']
```

Programski kod 4.17 data.yaml datoteka

4.6.2. Konfiguracija i treniranje neuronske mreže

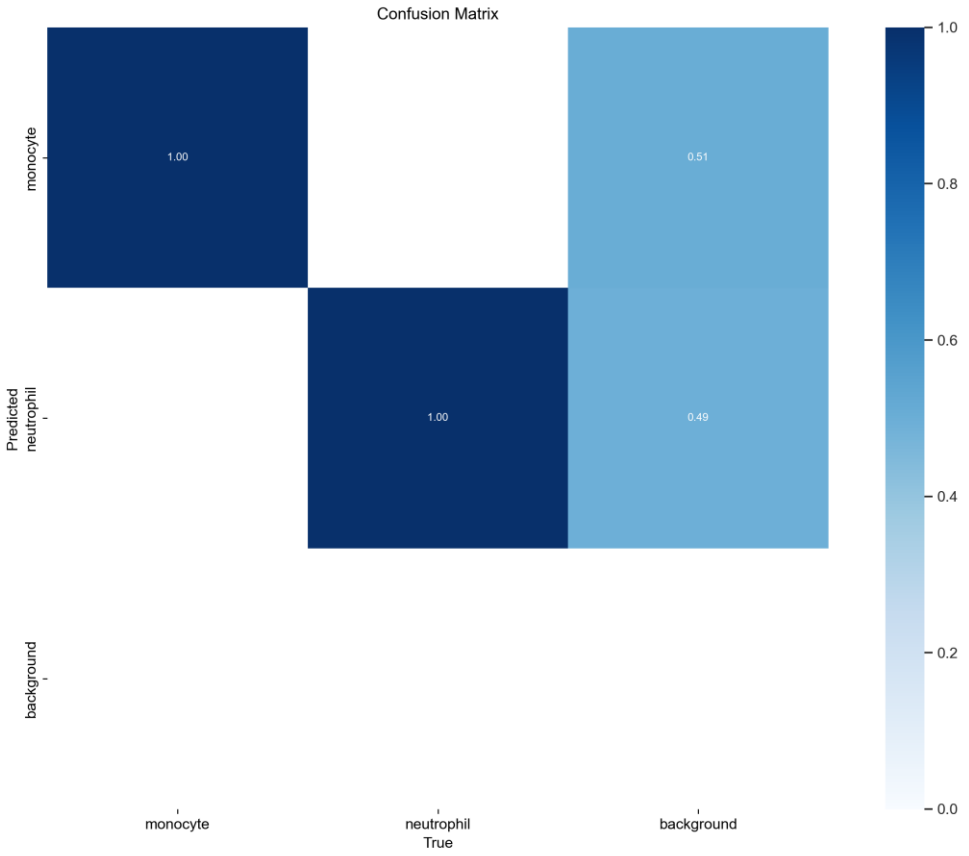
Od hiperparametara, bitno je postaviti *batch-size* na veličinu ne veću od 25 jer u protivnom se riskira prekoračenje kvote od šest gigabajta radne memorije grafičke kartice te će samim time doći do prestanka i rušenja treniranja neuronske mreže. Moguće je koristiti trenirane težine kako bi se ubrzalo vrijeme treniranje neuronske mreže i pospješila njena preciznost. U ovom slučaju koristit će se *YOLOv5 small* težine. Također, prvo će se trenirati model s tri epohe, a nakon toga model sa šest epoha te će se uspoređivati rezultati istih. Za početak treniranja neuronske mreže, pokreće se programski kod 4.18 *Skripta za pokretanje treniranja*.

```
python train.py --img 640 --epochs 3 --data ..\datasets\cells\data.yaml --weights yolov5s.pt --batch-size -25
```

Programski kod 4.18 Skripta za pokretanje treniranja

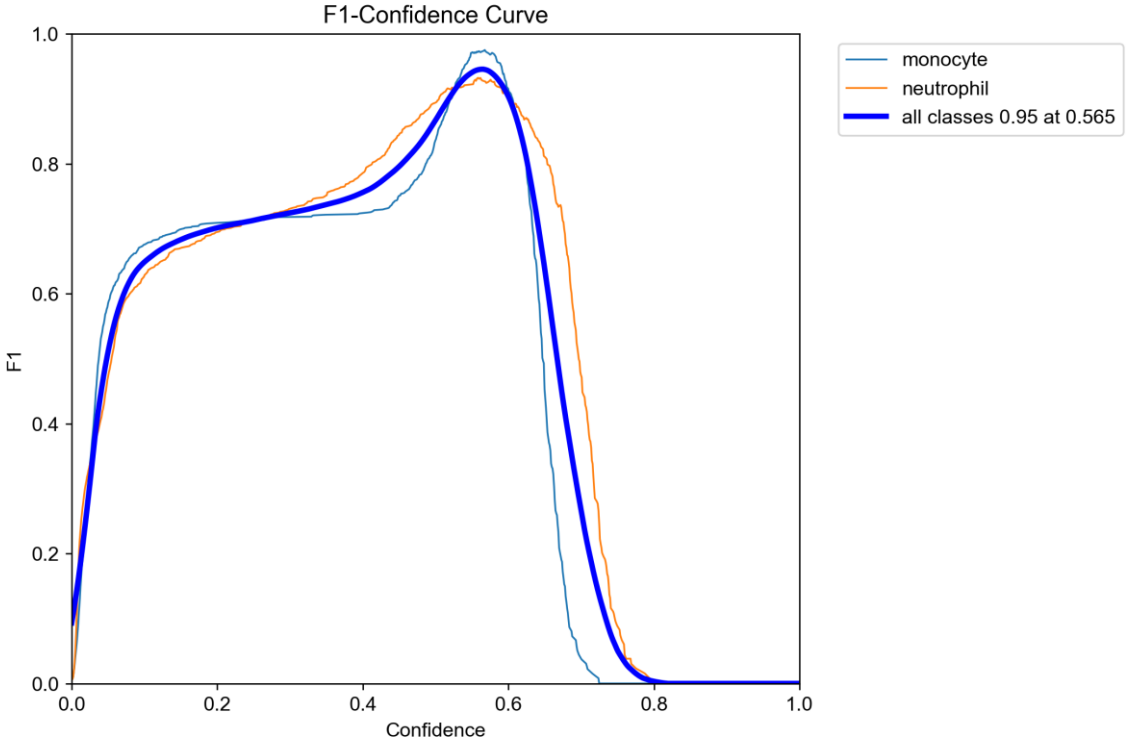
4.6.3. Rezultati treniranja neuronske mreže

Za validaciju modela moguće je koristit *val.py* skriptu *Ultralytics* [24] repozitorija. Izborom modela, kreira se direktorij imena *exp* i rednog broja te se u njega pohranjuju rezultati evaluacije. Prvo će se promatrati rezultati modela treniranog tri epohe.

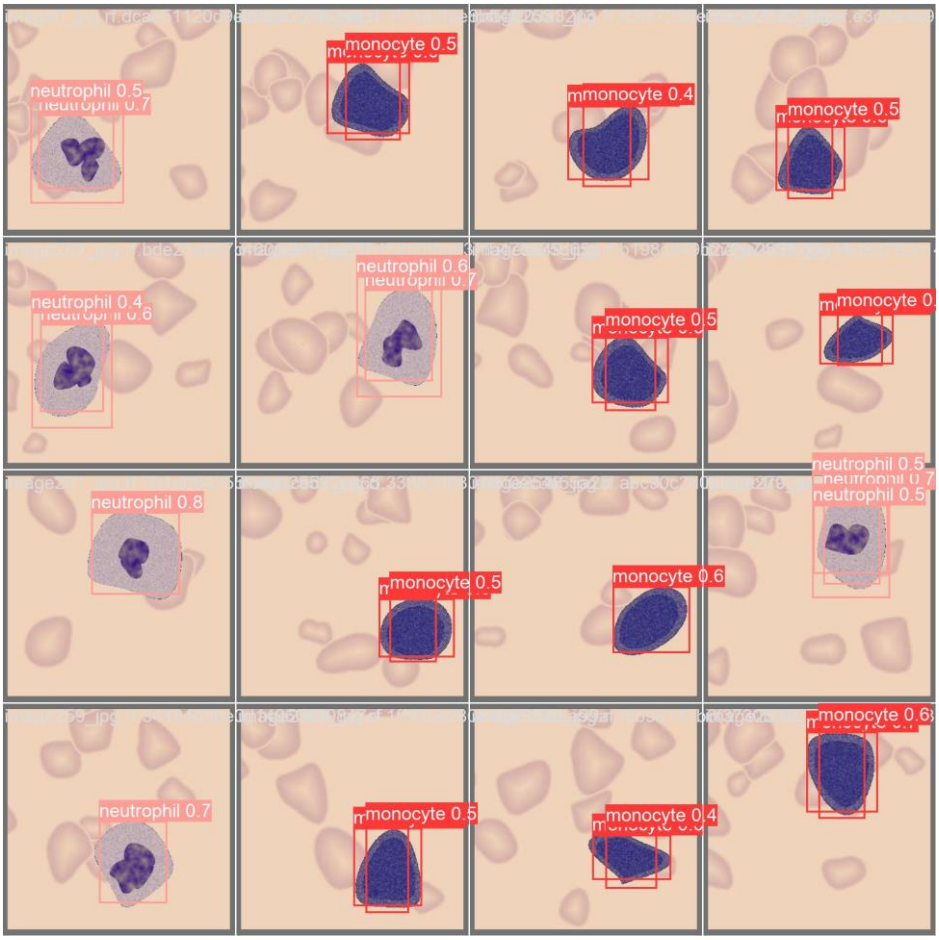


Slika 4.21 Matrica zabune za model 3 epohe

Slika 4.21 *Matrica zabune za model 3 epohe* pokazuje da model jako dobro razlikuje neutrofile od monocita. No, trenutno postoji velik broj lažno pozitivnih detekcija na slikama. Slika 4.23 Detekcija modela, 3 epohe pokazuje nekoliko primjera detekcije modela, a vidljivo je da model ima tendenciju detektiranja istog objekta više puta, ali s manjom pouzdanošću. Graf ovisnosti vrijednosti F1 o pouzdanosti detekcije (Slika 4.22) također ukazuje na problem da postoji malen broj detekcija iznad pouzdanosti od 0.7, a ni za jednu detekciju model nije pouzdan više od 0.8. Također, pri pouzdanosti od 0.6, model bolje prepoznaje monocite od neutrofila.

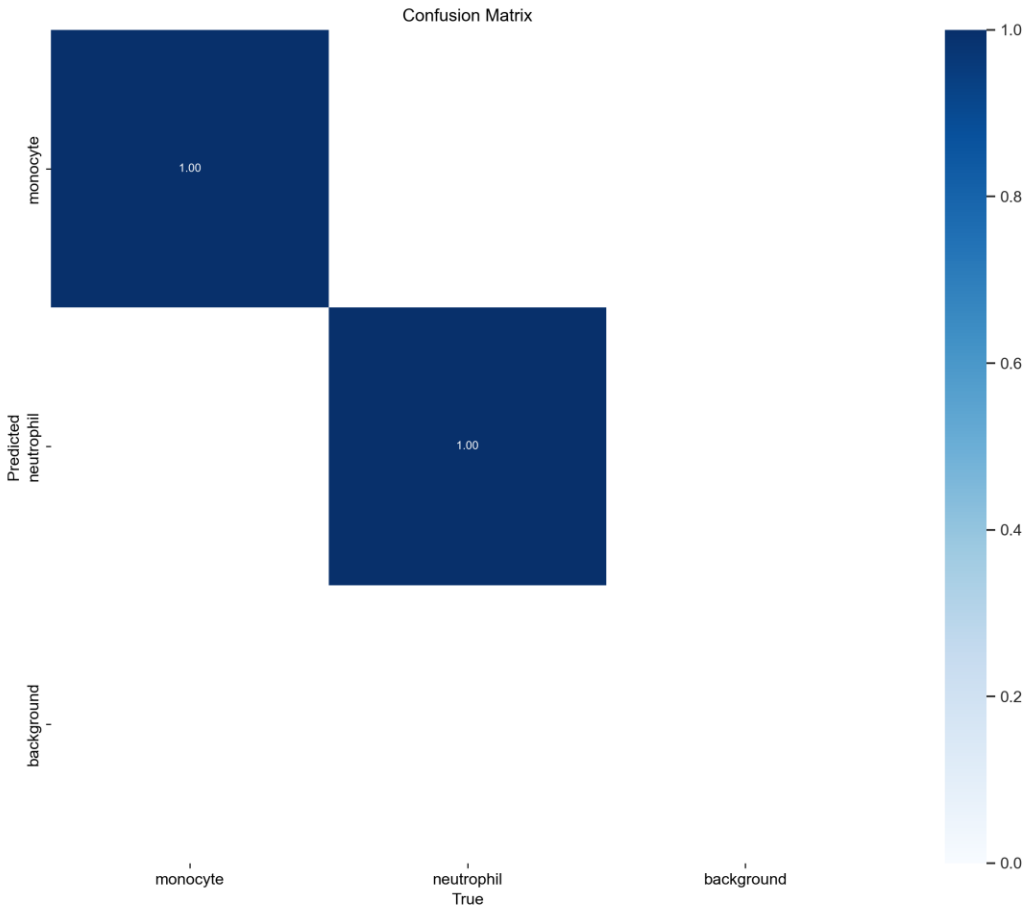


Slika 4.22 Graf F1 vrijednosti u ovisnosti pouzdanosti

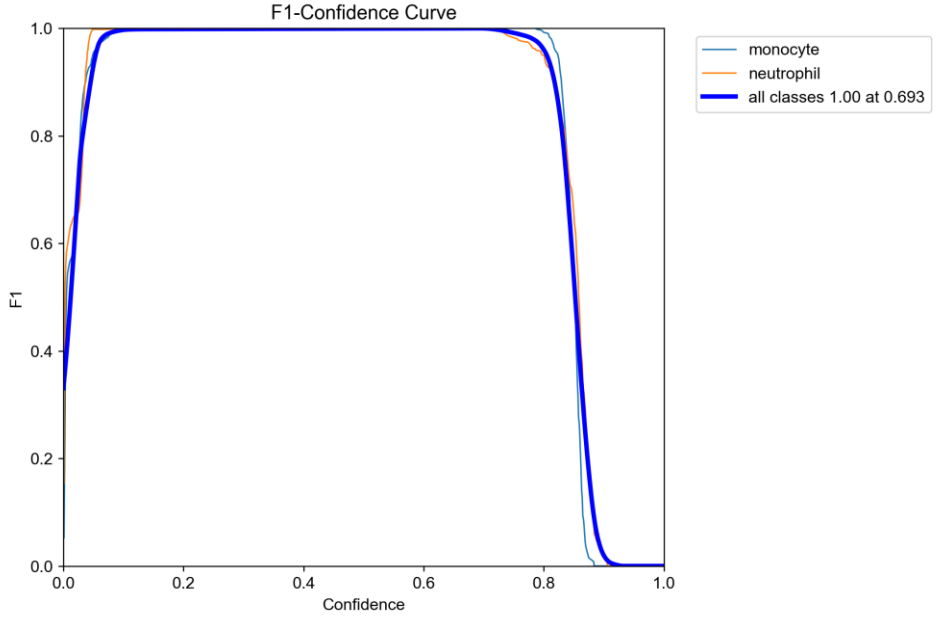


Slika 4.23 Detekcija modela, 3 epohe

Kod modela treniranog sa šest epohi situacija je već znatno drugačija. Matrica zabune (Slika 4.24) već pokazuje da trenirani model ne griješi kod klasifikacije objekata i da ne postoje lažne detekcije.

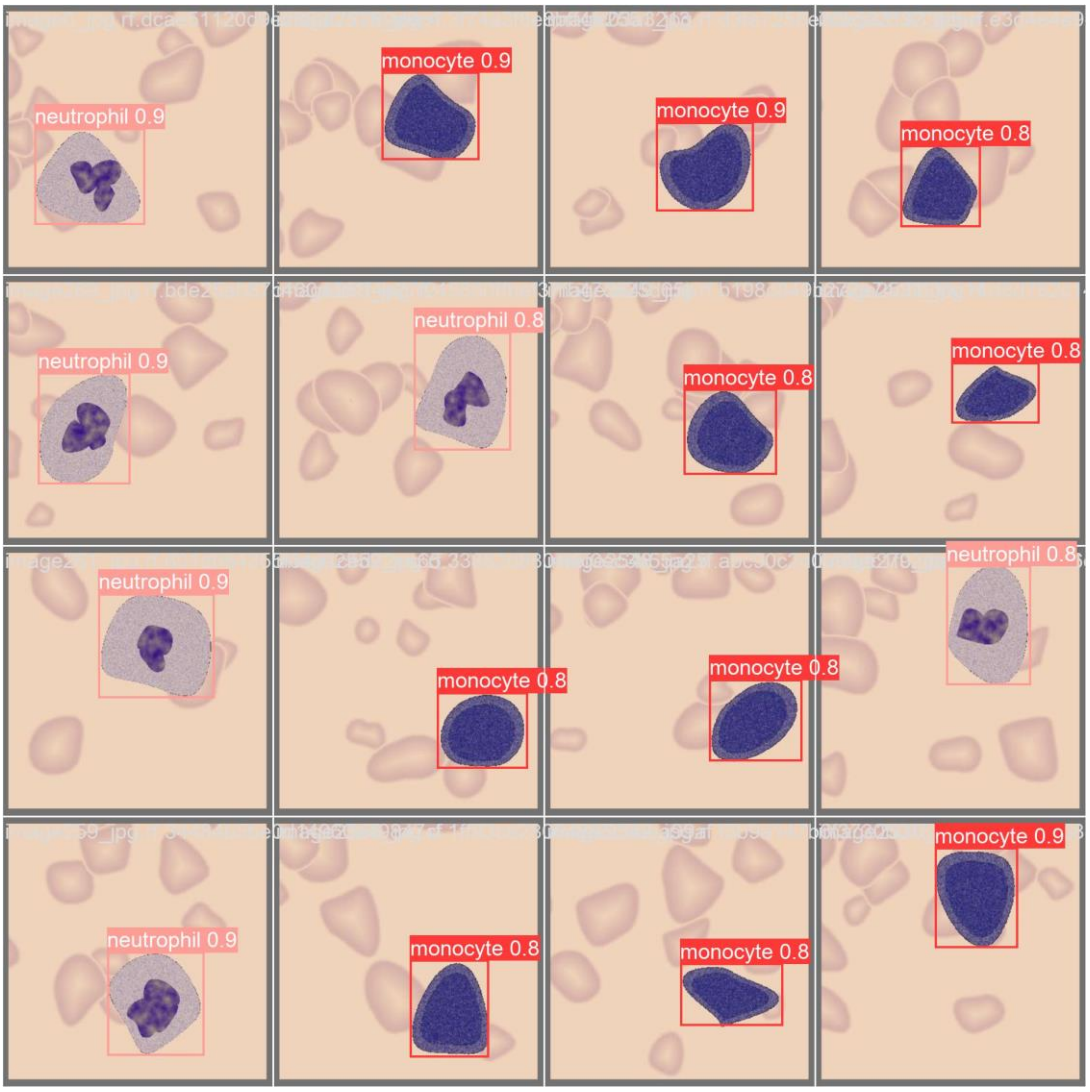


Slika 4.24 Matrica zabune modela s 6 epohi



Slika 4.25 F1-pouzdanost graf modela s 6 epohi

Graf F1 u ovisnosti o pouzdanosti (Slika 4.25) nalikuje na graf idealnog modela za klasifikaciju više klasa. Linije za klase neutrofila i monocita se zamalo preklapaju, no model i dalje malo bolje radi za stanice monocita.



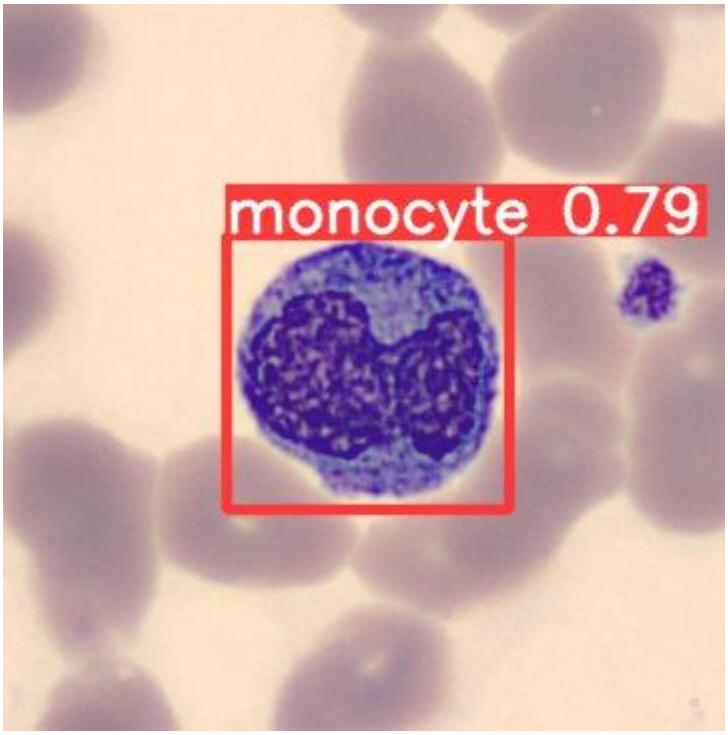
Slika 4.26 Detekcije modela treniranog 6 epohi

Mozaik detekcija (Slika 4.26) vizualno predočuje rad i detekcije modela. Za razliku od modela treniranog tri epohe, kao veliku razliku ovdje se primjećuje nepostojanje duplih detekcija te visoka razina pouzdanosti detekcija. Takvi rezultati mogu upućivati na pojavu pretjeranog usklađivanja modela na podatke, no postoji mogućnost da je skup podataka jednostavan u smislu da postoje jasne razlike između klasi koje model brzo detektira.

4.6.4. Validacija na stvarnom skupu podataka

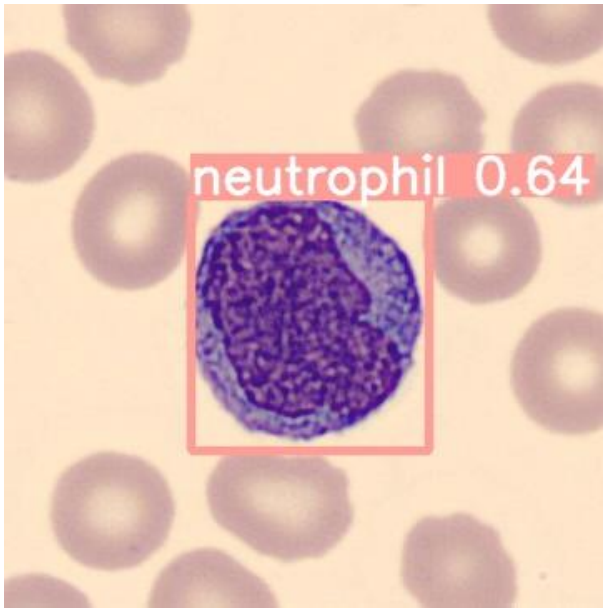
Kako bi se procijenila korisnost modela u stvarnom svijetu, vrši se validacija na stvarnom skupu podataka [1]. Za traženi skup podataka ne postoje anotacije, te iz tog razloga nije moguće provesti

detaljnu analizu rada modela, već će se oslanjati na uzimanje slučajnih uzoraka. Na velikoj većini slika model ne prepoznaje ništa iznad pouzdanosti od 0.25, no prokomentirat će se rezultati kada je neki objekt prepoznat.



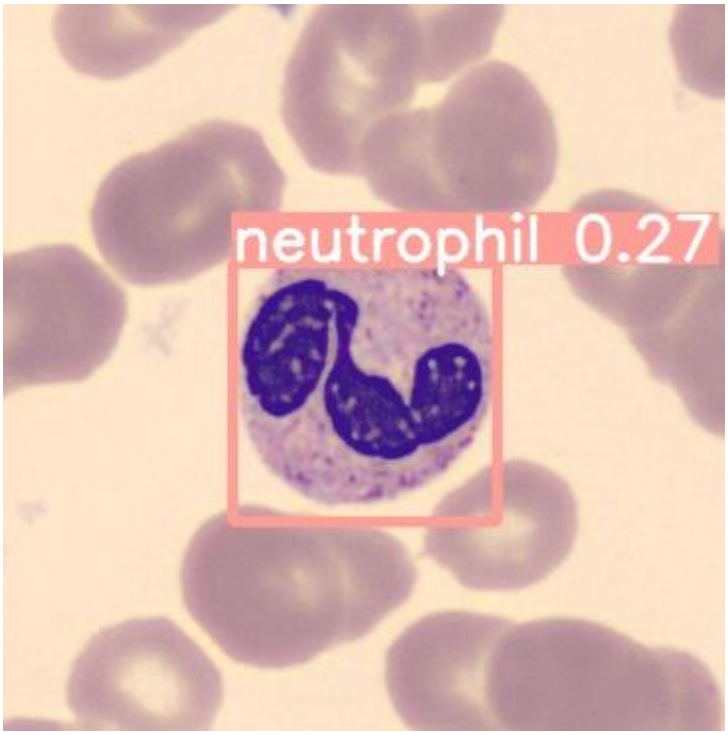
Slika 4.27 Prepoznat monocit

Na slici 4.27, model sa visokom pouzdanošću ispravno prepoznaje stanicu monocita. Jezgra na toj slici i općenito ton boje je dovoljno sličan generiranim podacima da model uspješno prepozna stanicu.



Slika 4.28 Prepoznat neutrofil

Na slici 4.28 je krivo prepoznata stanica neutrofila kada je riječ o stanici monocita. Moguć razlog je to što je stanica monocita svjetlija nego inače te njena jezgra ima sličnost fraktalnom šumu kakav je korišten za generiranje stanice neutrofila.



Slika 4.29 Uspješno prepoznat neutrofil

Na slici 4.29 uspješno je prepoznata stanica neutrofila, doduše sa slabom pouzadnošću. Jedna od važnih stavki za istaknuti jest da model ima tendenciju da češće prepoznaje neutrofile, čak i ako ih na slici nema, dok za monocite model rijetko prepoznaje objekt na slici, no ako prepozna objekt, ne griješi. S dodatnom prilagodbom algoritma za generiranje stanica te treniranja neuronske mreže, mogući su bolji rezultati.

5. ZAKLJUČAK

Oskudnost kvalitetnih skupova podataka krvnih stanica predstavlja problem u svijetu medicine, pogotovo s rastom popularnosti umjetne inteligencije. Jedan on načina generiranja umjetnih podataka jest korištenje metoda obrada slike, što je bio zadatak ovog rada. Predloženo je rješenje generiranje slika za stanice neutrofila i za stanice monocita. Jedno od velikih prednosti umjetnih podataka je što se automatski anotiraju slike te se izbjegava potreba za ručnim označavanjem slika što iziskuje mnogo truda i sredstava. Ovim rješenjem moguće je stvoriti beskonačno mnogo velik skup podataka, slobodan za korištenje i obradu. Predstavljeno rješenje nije dovoljno precizno da bi bilo korišteno u ozbiljnije svrhe kao što su medicinske, no uz savjete medicinskih stručnjaka i dodatno dorađivanje rješenja, autor smatra da je moguće predstaviti rješenje koje kvalitetno stvara umjetni skup podataka za različite bijele krvne stanice.

LITERATURA

[1] A. Acevedo, A. Merino, S. Alférez, Á. Molina, L. Boldú, i J. Rodellar, „A dataset of microscopic peripheral blood cell images for development of automatic recognition systems“, *Data Brief*, sv. 30, str. 105474, lip. 2020, doi: 10.1016/j.dib.2020.105474.

[2] N. Tahir i F. Zahra, „Neutrophilia“, u *StatPearls*, Treasure Island (FL): StatPearls Publishing, 2023. Pristupljeno: 08. rujan 2023. [Na internetu]. Dostupno na: http://www.ncbi.nlm.nih.gov/books/NBK570571/

[3] S. Kanuru i A. Sapra, „Eosinophilia“, u *StatPearls*, Treasure Island (FL): StatPearls Publishing, 2023. Pristupljeno: 08. rujan 2023. [Na internetu]. Dostupno na: http://www.ncbi.nlm.nih.gov/books/NBK560929/

[4] K. L. Sticco, N. K. Pandya, i D. T. Lynch, „Basophilia“, u *StatPearls*, Treasure Island (FL): StatPearls Publishing, 2023. Pristupljeno: 08. rujan 2023. [Na internetu]. Dostupno na: http://www.ncbi.nlm.nih.gov/books/NBK535365/

[5] J. Raitoharju, „Convolutional neural networks“, u *Deep Learning for Robot Perception and Cognition*, Elsevier, 2022, str. 35–69. doi: 10.1016/B978-0-32-385787-1.00008-7.

[6] „YOLO v3 algorithm flow chart A convolutional neural network usually consists of an input“, *ResearchGate*. https://www.researchgate.net/figure/YOLO-v3-algorithm-flow-chart-A-convolutional-neural-network-usually-consists-of-an-input_fig2_337451395 (pristupljeno 08. rujan 2023.).

[7] C. Rodríguez - Pardo, „Personalised aesthetics assessment in photography using deep learning“, PhD Thesis, 2018.

[8] A. Almryad i H. Kutucu, „Automatic identification for field butterflies by convolutional neural networks“, *Eng. Sci. Technol. Int. J.*, sv. 23, velj. 2020, doi: 10.1016/j.jestch.2020.01.006.

[9] Q. Sellat, S. K. Bisoy, i R. Priyadarshini, „Semantic segmentation for self-driving cars using deep learning“, u *Cognitive Big Data Intelligence with a Metaheuristic Approach*, Elsevier, 2022, str. 211–238. doi: 10.1016/B978-0-323-85117-6.00002-9.

[10] Y. Su, Y. Zang, Q. Su, i L. Peng, „A Method for Expanding the Training Set of White Blood Cell Images“, *J. Healthc. Eng.*, sv. 2022, str. 1267080, stu. 2022, doi: 10.1155/2022/1267080.

[11] Z. M. Kouzehkanan *i ostali*, „Raabin-WBC: a large free access dataset of white blood cells from normal peripheral blood“, Bioengineering, preprint, svi. 2021. doi: 10.1101/2021.05.02.442287.

[12] G. Van Rossum i F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[13] C. R. Harris *i ostali*, „Array programming with NumPy“, *Nature*, sv. 585, izd. 7825, str. 357–362, ruj. 2020, doi: 10.1038/s41586-020-2649-2.

[14] G. Bradski, „The OpenCV Library“, *Dr Dobbs J. Softw. Tools*, 2000.

[15] A. Paszke *i ostali*, „PyTorch: An Imperative Style, High-Performance Deep Learning Library“, u *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, str. 8024–8035. [Na internetu]. Dostupno na: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[16] J. S. JUN 29 i 2020 13 Min Read, „What is YOLOv5? A Guide for Beginners.“, *Roboflow Blog*, 29. lipanj 2020. https://blog.roboflow.com/yolov5-improvements-and-evaluation/ (pristupljeno 08. rujan 2023.).

[17] P. T. An, P. T. T. Huyen, i N. T. Le, „A modified Graham’s convex hull algorithm for finding the connected orthogonal convex hull of a finite planar point set“, *Appl. Math. Comput.*, sv. 397, str. 125889, svi. 2021, doi: 10.1016/j.amc.2020.125889.

[18] H. Babić, „Signali i Sustavi“.

[19] „Erythrocytes“. https://medcell.org/histology/blood_bone_marrow_lab/erythrocytes.php (pristupljeno 12. rujan 2023.).
[20] T. Strutz, „The Distance Transform and its Computation“. arXiv, 24. veljača 2023. Pristupljeno: 07. rujan 2023. [Na internetu]. Dostupno na: http://arxiv.org/abs/2106.03503
[21] „OpenCV: Image Segmentation with Distance Transform and Watershed Algorithm“. https://docs.opencv.org/3.4/d2/dbd/tutorial_distance_transform.html (pristupljeno 12. rujan 2023.).
[22] Z. M. Kouzehkanan, S. Tavakoli, i A. Alipanah, „Easy-GT: Open-Source Software to Facilitate Making the Ground Truth for White Blood Cells Nucleus“, 2021, doi: 10.48550/ARXIV.2101.11654.
[23] pierre, „Perlin Noise With Numpy“, *pvigier's blog*, 13. lipanj 2018. https://pvigier.github.io/2018/06/13/perlin-noise-numpy.html (pristupljeno 12. rujan 2023.).
[24] G. Jocher *i ostali*, „ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation“. Zenodo, 22. studeni 2022. doi: 10.5281/ZENODO.3908559.

ŽIVOTOPIS

Bacc. ing. Krunoslav Petrik je prvostupnik računarstva Fakulteta Elektrotehnike, Računarstva i Informacijskih Tehnologija u Osijeku. Trenutno pohađa diplomski studij informacijskih i podatkovnih tehnologija.

_____

Potpis autora

SAŽETAK

Generiranje umjetnog skupa podataka bijelih krvnih stanica u ovom diplomskom radu riješeno je za stanice neutrofila i monocita. Glavni motiv ovog zadatka jest ukazati na mogućnost i prednosti stvaranja umjetnog skupa podataka te dokazati da je moguće stvoriti umjetni skup podataka iz nule koristeći metode obrade slika i matematičke operacije. Opisane su sve linije programskog koda korištene za ostvarenje zadatka te je za korištene metode pružena teorijska podloga. Korištena je umjetna neuronska mreža trenirana na generiranom skupu podataka kako bi se prikazala stvarna upotreba umjetno generiranog skupa podataka te su analizirani rezultati treniranja.

Ključne riječi: računalni vid, strojno učenje, obrada slike, generiranje podataka

ABSTRACT

The generation of an artificial dataset of white blood cells in this thesis is accomplished for neutrophils and monocytes. The main motive of this task is to point out the possibilities and advantages of creating an artificial data set and to prove that it is possible to create an artificial data set from scratch using image processing methods and mathematical operations. All the used programming code is thoroughly described, and a theoretical basis is also provided for the methods used. An artificial neural network was trained on the generated data set to represent the real use of the artificially generated data set and the training results were analyzed.

Key words: computer vision, machine learning, image processing, data generation