Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klara Nosan

Inženiring programa za štetje objektov na slikah

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Šajn
Somentor: dr. Uroš Čibej

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preučite zmožnosti paralelizacije procesiranja v novi verziji programa ImageJ2. Prejšnjo verzijo ImageJ1 smo na fakulteti že uporabljali v programu Learn123, ki je implementiran v programskem jeziku Scala, za študijo štetja celic z genetskimi algoritmi s pomočjo zaporednega izvajanja. Poskusite dodelati program Learn123 tako, da bo lahko uporabil vzporedno procesiranje in rezultate testirajte na obstoječih in novih domenah.

Zahvaljujem se mentorju, doc. dr. Luki Šajnu, za priložnost sodelovanja pri ravoju programa Learn123 in pomoč pri iskanju novih primerov uporabe. Posebej se zahvaljujem tudi somentorju, dr. Urošu Čibeju, za vso pomoč pri posodabljanju programa Learn123 in nasvete pri pisanju diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	application programming interface	vmesnik za programiranje
CPU	central processing unit	centralno procesna enota
JVM	Java Virtual Machine	javanski navidezni stroj
UI	user interface	uporabniški vmesnik

Povzetek

Naslov: Inženiring programa za štetje objektov na slikah

Avtor: Klara Nosan

Štetje objektov je pomembno področje računalniškega vida, ki ima veliko aplikacij na različnih domenah. Na področju bioloških znanosti je posebej pomembno štetje celic na mikroskopskih slikah, saj gre za eno izmed temeljnih metod za analizo vzorcev. V diplomskem delu predstavimo pregled funkcionalnosti programske opreme ImageJ, ki se v bioloških znanostih uporablja za procesiranje mikroskopskih slik. Opišemo novosti in spremembe, ki jih prinaša najnovešja verzija ImageJ ter posebno pozornost namenimo arhitekturi programa in vmesniku za programiranje. Predstavimo program Learn123, namenjen avtomatizaciji štetja celic na mikroskopskih slikah z ImageJ, v katerem je učenje štetja celic implementirano z genetskim algoritmom. Opišemo postopek posodobitve programa Learn123, v kateri smo za avtomatsko štetje celic na slikah uporabili aktualno posodobitev ImageJ in paralelizirali postopek učenja štetja celic. Predstavimo rezultate testiranja paralelnega učenja v primerjavi z zaporednim. Opišemo tudi nov primer uporabe Learn123 in tipe slik, na katerih ne moremo šteti objektov samo z uporabo programa ImageJ.

Ključne besede: štetje objektov, procesiranje slik, ImageJ, genetski algoritem.

Abstract

Title: Engineering of an object-counting program

Author: Klara Nosan

Counting objects in images is an important problem in many fields, such as the field of life sciences, where counting cells in microscopic images is a fundamental analysis tool. In this Bachelor's Thesis we present ImageJ, the state of the art program for microscopic image processing in life sciences. We provide an overview of the newest version of ImageJ with an emphasis on the architecture and the API of the software. We present Learn123, a program designed to automize cell counting in microscopic images using ImageJ. We describe the process of updating Learn123 by using the most recent version of ImageJ and parallelizing the genetic algorithm used to learn cell counting. We then compare the parallel algorithm to its previous sequential version. Finally, we present a new use case of Learn123 and discuss which types of images are not suitable for object counting using ImageJ.

Keywords: object counting, image processing, ImageJ, genetic algorithm.

Poglavje 1

Uvod

Štetje objektov je zanimivo področje računalniškega vida, ki ima veliko aplikacij na različnih domenah. Pri analizi slik nas pogosto zanima, koliko določenih stvari je na njih. Za nadzor parkirišč je pomembno število parkiranih avtomobilov, pri nadzoru cest pa število vozil, ki so se peljala mimo izbrane točke. V tovarnah nas zanima število izdelkov na tekočem traku, v veliko situacijah pa število ljudi na slikah ali videoposnetkih.

Štetje objektov je pomembno tudi na področju biologije. Štetje živali v jatah in čredah se uporablja za spremljanje in napovedovanje sprememb v velikosti njihovih populacij. Štetje celic pa je ena izmed temeljnih metod za analizo vzorcev. Obema je skupno dejstvo, da je ročno anotiranje in štetje primerkov na slikah zamudno opravilo, pogosto podvrženo napakam. V zadnjih letih se zato pojavlja vedno več poskusov avtomatizacije teh procesov [2, 8, 37, 39].

Eden izmed programov namenjenih avtomatizaciji štetja na področju biologije je tudi Learn123 [23]. Gre za program za avtomatizacijo štetja celic na mikroskopskih slikah, ki je nastal na Fakulteti za računalništvo in informatiko. Učenje štetja celic je v Learn123 implementirano s pomočjo genetskega algoritma, za procesiranje slik pa je uporabljena programska oprema ImageJ, ki predstavlja eno izmed najbolj uporabljenih orodij za anotacijo in analizo mikroskopskih slik na področju biologije.

V okviru diplomskega dela smo pripravili pregled funkcionalnosti programske opreme ImageJ ter novosti in spremembe, ki jih prinaša njegova najnovešja verzija. Posebno pozornost smo namenili arhitekturi in vmesniku za programiranje ImageJ in opisali, kako deluje na primeru programa Learn123. Le-tega smo posodobili tako, da za avtomatsko štetje celic na slikah uporablja aktualno posodobitev ImageJ in učenje štetja celic paralelizirali. Posvetili smo se tudi iskanju novega primera uporabe programa Learn123.

V poglavju 2 predstavimo programsko opremo ImageJ, poglavje 3 pa opisuje delovanje programa Learn123. Proces posodobitve programa Learn123 opišemo v poglavju 4 in v poglavju 5 predstavimo postopek iskanja novega primera uporabe.

Poglavje 2

ImageJ

ImageJ je odprtokodni program za procesiranje slik, napisan v programskem jeziku Java. Namenjen je predvsem obdelavi slik v naravoslovnih znanostih, na primer mikroskopskih slik. Prva verzija ImageJ je nastala leta 1997 v ameriški instituciji National Institutes of Health (NIH) iz programa NIH Image. Gre za namizno aplikacijo, namenjeno prikazovanju, oblikovanju, procesiranju, analizi in shranjevanju slik.

Z večanjem števila uporabnikov in njihovih zahtev so aplikaciji postopoma dodajali funkcionalnosti. ImageJ se je tako razvil izven predvidenega obsega, vendar zaradi tehničnih omejitev in prvotne arhitekture ImageJ 1.x (v nadaljevanju ImageJ1) ni več dohajal trendov v naravoslovnih znanostih. Težavo je začel predstavljati tudi pomanjkljiv API. ImageJ 2.x (v nadaljevanju ImageJ2) predstavlja novo generacijo programa s popolnoma prenovljeno arhitekturo in dodatnimi funkcionalnostmi, ki sledi trenutnim trendom slik v naravoslovnih znanostih.

V nadaljevanju predstavimo funkcionalnosti implementirane v ImageJ1 ter spremembe in razširitve, ki jih prinaša ImageJ2. Posebno pozornost namenimo API-ju obeh verzij programske opreme, ki smo ga za detekcijo objektov na slikah uporabili v tem diplomskem delu. Predstavimo tudi razširjeno distribucijo programa ImageJ, FIJI.

2.1 ImageJ1

Osnovne funkcionalnosti programa ImageJ1 so prikazovanje, oblikovanje, procesiranje, analiza in shranjevanje slik, ki so lahko 8, 16 ali 32 bitne. ImageJ1 podpira standardne funkcije za procesiranje slik, kot so spreminjanje kontrasta, ostrenje, glajenje in filtriranje slik ter odkrivanje robov na slikah. Omogoča merjenje razdalj na sliki in računanje statističnih podatkov o označenih delih slike. Implementirano je tudi risanje diagramov intenzitete sivinskih slik, ki so posebej uporabni pri analizi bioloških vzorcev.

Funkcionalnosti programa ImageJ1 niso omejene le na osnovne operacije za procesiranje slik, saj ga je mogoče razširiti z uporabniškimi vtičniki. Ker pri procesiranju slik v znanosti pogosto želimo shraniti zaporedje operacij, ki smo jih uporabili za procesiranje slike, ali enake operacije uporabiti na več slikah, so ImageJ1 dodali tudi možnost skriptiranja [11].

2.1.1 Skriptiranje v ImageJ1 z jezikom ImageJ Macro

Za skriptiranje so v ImageJ1 razvili lasten skriptni jezik, *ImageJ Macro Language*. Posamezno skripto v jeziku ImageJ Macro imenujemo *Macro* (slovensko *makro*), shranimo pa jo kot tekstovno datoteko s končnico *.ijm*. Z makroji lahko avtomatiziramo ponovljive procese, dodamo bližnjice v orodno vrstico in določimo lastne bližnjice na tipkovnici.

Sintaksa je podobna programskemu jeziku Java, posamezni stavki se zaključijo s podpičjem, bloki kode pa so omejeni z zavitimi oklepaji. Spremenljivke v makrojih niso tipizirane, vendar je glede podatkovnih tipov Macro Language zelo omejen, saj so podprti samo numerični tipi in znakovni nizi. Pri pisanju makrojev je uporabnikom na voljo nabor funkcij za izvajanje ImageJ ukazov, obdelavo slik, datotek, znakovnih nizov, matematične operacije in manipulacijo z uporabniškim vmesnikom. Definiramo lahko tudi svoje funkcije, uporabljamo operatorje, ki so definirani v Javi, pogojne stavke, zanke in dodajamo komentarje.

Pisanja makrojev ni potrebno začeti z nič, saj lahko zaporedje ukazov

posnamemo z orodjem za snemanje makrojev, imenovanim *Macro Recorder*. Orodje med snemanjem shrani vse akcije uporabnika in zgenerira tekstovno datoteko, ki jo lahko uporabnik pregleda in spreminja v urejevalniku besedila, implementiranem v ImageJ1.

Jezik ImageJ Macro ima precej pomanjkljivosti, saj je zaradi arhitekture programa močno vezan na grafični uporabniški vmesnik. Zaporedje ukazov, ki ga opišemo z makrojem, je enako zaporedju akcij uporabnika, vključno z manipulacijo z okni odprtimi v uporabniškem vmesniku. To lahko ponazorimo s primerom: tako kot v uporabniškem vmesniku pred izvedbo operacije na sliki najprej izberemo okno, v katerem je prikazana slika, ki jo želimo obdelati, moramo ta ukaz dodati v makro:

```
selectWindow("img");
run("Sharpen");
```

Makroji so omejeni tudi z vidika podatkovnih tipov in nabora funkcij, ki so na voljo uporabnikom. Kasneje so zato v ImageJ1 dodali še možnost skriptiranja v jeziku JavaScript, vendar večina uporabnikov, predvsem tistih, ki niso vešči programiranja, zaradi preproste sintakse in orodja *Macro Recorder* uporablja jezik ImageJ Macro [17].

2.1.2 Vtičnki

Funkcionalnost ImageJ1 lahko razširimo tudi s pisanjem vtičnikov (ang. *plugin*) v programskem jeziku Java, s katerimi dodamo operacije v meni uporabniškega vmesnika ImageJ1.

V primerjavi z makroji je pisanje vtičnikov bolj zahtevno, vendar pa so vtičniki zmogljivejši. Ni omejitev podatkovnih tipov in nabora funkcij, saj so pri programiranju na voljo vse metode, operatorji in podatkovni tipi programskega jezika Java. Za izvedbo ukazov ImageJ1 uporabljamo ImageJ1 API.

Za pisanje vtičnikov ImageJ1 sta na voljo dve ogrodji v obliki Java interface [9], in sicer:

- `PlugInFilter` - ogrodje za pisanje vtičnikov, ki delujejo na eni sliki, torej implementirajo operacijo, s katero obdelamo eno sliko.

- `PlugIn` - ogrodje za pisanje vseh ostalih vtičnikov.

2.1.3 ImageJ1 API

ImageJ1 namizna aplikacija, ki je bila razvita z namenom, da uporabnik s pomočjo uporabniškega vmesnika naenkrat uporablja eno instanco programa, zato ImageJ1 API ni dodelan. Celoten vmesnik za programiranje predstavlja en sam javanski razred (`IJ`), v katerem so implementirane statične metode za izvajanje ukazov ImageJ1, makrojev in vtičnikov.

ImageJ1 API je primeren predvsem za pisanje vtičnikov, ki so namenjeni uporabi z grafičnim uporabniškim vmesnikom. Zaradi implementacije celotne programske opreme, ki je močno vezana na uporabniški vmesnik, ima veliko klicev metod *stranske učinke* povezane z uporabniškim vmesnikom, kot na primer odpiranje ali spreminjanje vsebine odprtih oken. Takšno obnašanje pri uporabi vtičnikov pričakujemo, pri rabi API-ja brez UI-ja v drugih programih pa ni zaželjeno. Predvsem pri takšnem načinu uporabe ImageJ1 sam javanski razred, ki naj bi bil celoten API, ne zadošča, saj je pogosto potrebno ročno klicati metode za zapiranje ali spreminjanje oken uporabniškega vmesnika, ki so implementirane v drugih javanskih razredih.

Ker so vse metode statične in v veliki meri uporabljajo iste vire, je v primeru sočasnih klicev metod možno tudi prepisovanje istih spremenljivk. ImageJ1 API je zato potrebno uporabljati previdno, saj lahko takšno prepisovanje podatkov vodi v napačne rezultate.

2.2 ImageJ2

ImageJ1 se je z večanjem števila uporabnikov in njihovih potreb s postopnim dodajanjem funkcionalnosti razvil izven predvidnega obsega. ImageJ2 je nova verzija programa ImageJ s prenovljeno arhitekturo, katere osrednji cilj

je bil razširiti funkcionalnost prve verzije programa s podporo procesiranja večdimenzionalnih slik, ki so trenutni trend v naravoslovnih znanostih.

Poleg podpore več tipom slik je v ImageJ2 podprtih tudi več skriptnih jezikov, boljša integracija z drugimi aplikacijami in možnost uporabe več instanc programa hkrati. Uporabniška izkušnja ostaja enaka, saj je uporabniški vmesnik nove verzije enak originalnemu, integrirane pa so tudi vse funkcionalnosti implementirane v ImageJ1. V uporabniškemu vmesniku so posodobili urejevalnik besedila (*ScriptEditor*) ter dodali orodje za avtomatsko posodabljanje programa (*ImageJ Updater*) in iskalno vrstico, ki omogoča iskanje ukazov ter iskanje po dokumentaciji ImageJWiki in forumu ImageJForum [29].

2.2.1 Arhitektura ImageJ2

Nova arhitekutra programa ImageJ je modularna. Podatkovni model in operacije za procesiranje slik so ločene od uporabniškega vmesnika. Tako ImageJ2 ni več samo namizna aplikacija, temveč obenem tudi zbirka knjižnic programske opreme, namenjenih za uporabo v drugih aplikacijah. Zgrajen je na ogrodju SciJava.

SciJava

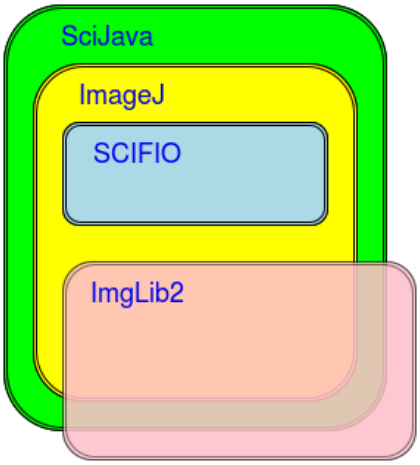
SciJava je zbirka programske opreme, ki združuje več projektov namenjenim uporabi v znanosti. Sestavljajo jo štiri osrednje komponente:

- **SciJava** - knjižnice namenjene pisanju razširljivih aplikacij. Gre za komponente za splošno rabo, ki niso specifične za delo s slikami. Mednje sodi tudi knjižnica SciJava Commons, ki predstavlja ogrodje aplikacije ImageJ2.
- **ImgLib2** - knjižnica, v kateri so implementirani podatkovni tipi namenjeni za predstavitev večdimenzionalnih znanstvenih slik [16].
- **SCientific Image File Input and Output (SCIFIO)** - knjižnica, v

kateri so implementirane operacije za branje, pisanje in pretvorbo med raznimi formati večdimenzionalnih slik [32].

- **ImageJ2** - knjižnice in aplikacija za procesiranje večdimenzionalnih slik.

Opisane komponente niso neodvisne, njihova hieararhična organizacija je predstavljena na sliki 2.1.



Slika 2.1: Hierarhična organizacija projekta SciJava [34]

ImageJ2

ImageJ2 sestavlja več programskih komponent:

- **ImageJ Common** - osnovni podatkovni model, ustvarjen z uporabo ImgLib2.
- **ImageJ Ops** - programsko ogrodje za implementacijo algoritmov za procesiranje slik.
- **ImageJ Updater** - mehanizem za posodabljanje posameznih vtičnikov in knjižnic znotraj ImageJ.
- **ImageJ Legacy** - skrbi za vzvratno kompatibilnost z ImageJ1.
- **SciJava Common** - ogrodje za pisanje aplikacij, vtičnikov, modulov.

2.2.2 Kompatibilnost z ImageJ1

ImageJ2 je popolnoma neodvisen od ImageJ1, vendar pa je zaradi obstoječih uporabnikov uporabniškega vmesnika in API-ja prve verzije programa nujna vzvratna kompatibilnost med prvo in drugo verzijo programa. Zanjo je poskrbljeno s komponento ImageJ Legacy, ki vključuje naslednje funkcionalnosti:

- Izvajanje ImageJ1 brez uporabniškega vmesnika (ang. *headless*).
- Ovijanje (ang. *wrapping*) uporabniškega vmesnika ImageJ1 v uporabniški vmesnik SciJava.
- Pretvorba med podatkovnimi strukturami ImageJ1 in ImageJ2.

ImageJ Legacy vključuje tudi ImageJ1 API [5].

2.2.3 ImageJ2 API

Arhitektura ImageJ2 je prilagojena uporabi posameznih knjižnic in metod v drugih aplikacijah. V primerjavi z ImageJ1 API-jem ni več težav z uporabniškim vmesnikom, saj je ta ločen od podatkovnih tipov, vhodno-izhodnih operacij in operacij za obdelavo slik.

Kljub izboljšani arhitekturi opazimo, da zaradi modularnosti ImageJ2 in zbirke SciJava, ki jo sestavljajo štirih komponente (SciJava, ImageJ2, ImgLib2 in SCIFIO), programiranje z ImageJ2 API-jem zahteva več programerskega znanja in razumevanja programskega jezika Java. Ukazi, ki jih z ImageJ1 API-jem lahko izvedemo s klicem ene metode, so v ImageJ2 pogosto kompleksnejši [15].

Šibka točka ImageJ2 API-ja je tudi dokumentacija, ki je sicer precej obsežna, vendar ni zbrana na enem mestu. Na voljo je avtomatsko generirana dokumentacija kode v Javi (ang. *Javadoc*) za posamezne komponente [33]. Na spletni strani projekta ImageJ so na voljo še opisi arhitekture, posameznih komponent in API-ja, ki jih je veliko, a se vsebina na različnih straneh ponavlja, izrazoslovje pa je v veliko primerih nekonsistentno. Primanjkuje predvsem konkretnih primerov uporabe API-ja.

Zaradi omenjenih negativnih lastnosti ImageJ2 API-ja za obstoječe uporabnike ImageJ1 API-ja ne predstavlja nujno tako velike izboljšave, kot se to zdi na prvi pogled. Primer uporabe ImageJ2 API-ja, potek prehoda z ImageJ1 API-ja na novo verzijo in težave, na katere smo naleteli pri tem, opišemo v poglavju 4.

2.3 FIJI

FIJI (**F**iji **I**s **J**ust **J**mageJ) je distribucija ImageJ2, razširjena z vtičniki, ki so nastali v uporabniški skupnosti programa ImageJ. FIJI je namenjem predvsem uporabi v bioloških razkiskavah, čemur je prilagojen tudi nabor vtičnikov, ki so vključeni v distribucijo.

Uporabniki lahko vtičnike prispevajo preko platforme *github*, zanje objavijo dokumentacijo in ustvarijo strani za posodabljanje (ang. *update site*) [38], kamor objavljajo posodobitve prispevanega vtičnika. Takšen način posodabljanja vtičnikov omogoča ostalim uporabnikom FIJI-ja, da z orodjem *ImageJ Updater* izberejo verzijo vtičnika, ki jo želijo uporabljati [31].

Poglavje 3

Program Learn123

Learn123 je program namenjen avtomatizaciji štetja celic na mikroskopskih slikah z uporabo orodja ImageJ. Ročno anotiranje in štetje celic je zelo zamuden proces, zato v biologiji zanj pogosto uporabijo program ImageJ. Z njim lahko sliko avtomatsko anotirajo in nato preštejejo označene objekte na sliki. Takšen način štetja je hitrejši od ročnega štetja, vendar pa je tudi izbira operacij in parametrov ukazov, s katerimi pred avtomatskim štetjem obdelajo sliko, zahtevna in dolgotrajna. Program Learn123 vključuje postopek učenja štetja celic, v katerem s poskušanjem določi parametre in tako popolnoma avtomatizira proces štetja celic.

Poleg popolnoma avtomatskega štetja celic na mikroskopskih slikah program uporabnikom omogoča tudi ročen pregled označenih slik in popravljanje oznak.

Learn123 uporablja ImageJ1 in je implementiran v programskem jeziku Scala. Učenje ocenjevanja števila celic na slikah poteka s pomočjo genetskega algoritma, ki je opisan v nadaljevanju. Podrobneje opišemo tudi postopek štetja celic z ImageJ1 in uporabo ImageJ1 API-ja.

3.1 Štetje celic z ImageJ1

Objekte na sliki v programu ImageJ1 štejemo s pomočjo ukaza *Analyze particles*. Ukazu kot parametre podamo velikost objektov, ki jih želimo upoštevati, njihovo približno obliko (sploščenost elipse, s katero lahko aproksimiramo obliko objekta) in katere podatke analize želimo prikazati oz. shraniti. Za avtomatsko analizo (štetje) objektov potrebujemo binarno sliko, na kateri se objekti ne prekrivajo. Zato moramo pred uporabo ukaza *Analyze particles* sliko segmentirati in nastaviti prage slike (ang. *thresholding*).

Pri nastavljanju pragov slike točke na sliki razdelimo v dva razreda: ospredje (ang. *foreground*) in ozadje (ang. *background*). Za nastavljanje pragov slike je v ImageJ1 implementiranih več metod za avtomatsko nastavljanje pragov slik, ki jih uporabimo z ukazom *setAutoThreshold*. Pred nastavljanjem pragov sliko navadno obdelamo tako, da se ospredje čimbolj loči od ozadja (na primer povečamo kontrast, izostrimo in podobno). Ko nastavimo prag slike, z ukazem *Convert to Mask* sliko pretvorimo v binarno masko. To lahko pred štetjem objektov dodatno obdelamo z naborom operacij, namenjenih obdelavi binarnih slik, s katerimi poskusimo objekte na sliki ločiti med seboj, da se ne prekrivajo.

Primer takšnega zaporedja ukazov lahko zapišemo z jezikom ImageJ Macro:

```
run("Enhance Contrast...", "Saturated pixels=0.5%");
setAutoThreshold("Otsu");
run("Convert to Mask");
run("Fill Holes");
run("Analyze Particles...", "size=100-Infinity
   circularity=0.50-1.00 show=Nothing");
```

Z zgornjim zaporedjem ukazov na sliki najprej povečamo kontrast, nato nastavimo prag slike z metodo Otsu in sliko pretvorimo v binarno masko. Na binarni sliki zatem zapolnimo luknje v objektih in izvedemo ukaz *Analyze Particles*. Rezultate analize objektov ImageJ1 prikaže v posebnem oknu, kjer

so navedeni podatki o posameznih objektih (na primer njihove koordinate na sliki, površina, in tako dalje).

V programu Learn123 celice na slikah štejemo tako, da s pomočjo ImageJ1 API-ja izvedemo makro z naborom ukazov za štetje objektov.

3.1.1 Implementacija z ImageJ1 API

V ImageJ1 API-ju je za izvajanje makrojev implementirana metoda `runMacro`, ki ji kot argument podamo zaporedje ukazov v jeziku ImageJ Macro v obliki znakovnega niza. Ker v programu Learn123 makroje izvajamo na posameznih slikah, bi potrebovali metodo, ki bi makro izvedla na izbrani sliki. Ta ne obstaja, zato se je potrebno pri programiranju z API-jem zgledovati po uporabniškem vmesniku, v katerem velja, da se makro izvede na sliki, ki je odprta v programu ImageJ1 in trenutno v fokusu. Makro na izbrani sliki z API-jem izvedemo tako, da dvojnik slike ročno nastavimo kot trenutno izbrano sliko v uporabniškem vmesniku in nato s klicem metode `runMacro` izvedemo makro:

```
WindowManager.setTempCurrentImage(img.duplicate)
IJ.runMacro(macro)
```

Slike so v ImageJ1 predstavljene kot instance razreda `ImagePlus`. Pri izvajanju makroja za detekcijo objektov sliko obdelamo in jo spremenimo v binarno sliko ter s tem spremenimo tudi objekt `ImagePlus`. Za trenutno izbrano sliko nastavimo dvojnik naše slike, če bi štetje želeli ponoviti ali na isti sliki kasneje izvesti drug makro. Dvojnik slike dobimo s klicem metode `duplicate`, ki je implementirana v razredu `ImagePlus`.

Podobno nedovršenost API-ja kot pri izvajanju makrojev opazimo tudi pri pridobivanju rezultatov štetja objektov, saj se moramo ponovno zgledovati po delovanju uporabniškega vmesnika. Ker je slednji implementiran tako, da je naenkrat odprto samo eno okno z rezultati oz. so rezultati shranjeni v eni globalni tabeli rezultatov, tudi te dobimo s klicem statične metode:

```
val results: ResultsTable = ResultsTable.getResultsTable
```

3.1.2 Težave pri štetju celic z ImageJ1

Kljub preprosti sintaksi jezika ImageJ Macro in relativno majhnemu številu ukazov, ki jih potrebujemo, da avtomatsko preštejemo celice na izbrani sliki, se lahko to v praksi izkaže za precej zamudno in zahtevno opravilo. Isti nabor ukazov ne deluje za vzorce z različnimi vrstami celic, drugačno povečavo mikroskopa ali osvetljenostjo vzorca pri zajemu slike. Nabolj zamuden del štetja je zato izbira operacij, s katerimi pred štetjem obdelamo sliko in določanje parametrov ukazov programa ImageJ1, ki jih uporabimo.

Ta proces je v programu Learn123 avtomatiziran z učenjem štetja, kjer gre v resnici za optimizacijo makroja z zaporedjem ukazov, ki jih uporabimo za štetje.

3.2 Učenje štetja

Pri učenju štetja celic z orodjem ImageJ gre v programu Learn123 za optimizacijo makroja, ki ga uporabimo za štetje celic. Proces optimizacije je implementiran s pomočjo genetskega algoritma.

Za začetek učenja potrebujemo množico slik z označenimi celicami in predlogo makroja z zaporedjem ukazov za štetje celic, ki jo bomo optimizirali.

3.2.1 Predloga za optimizacijo

Predloga (ang. *template*) za optimizacijo je napisana v jeziku ImageJ Macro s posebno sintakso, s katero posplošimo parametre makroja. Implementirane so naslednje posplošitve parametrov:

- **Interval vrednosti celoštevilskega parametra (integer interval)** - namesto celoštevilske vrednosti podamo meje intervala, v katerem mora biti dana vrednost. Primer: `$i{1,10}`.
- **Interval vrednosti realnega parametra (double interval)** - namesto realne vrednosti podamo meje intervala, v katerem mora biti dana vrednost. Primer: `$d{0.50,1.00}`.

- **Nabor možnih vrednosti v obliki znakovnih nizov (options for string parameters)** - namesto parametra v obliki znakovnega niza podamo nabor možnih parametrov. Primer: `$o{a,b,c,d}`.

Predloga je torej makro z zaporedjem ukazov za štetje, v katerem lahko namesto točnih vrednosti parametrov (argumentov) podamo samo interval, v katerem naj bi se vrednosti nahajale, ali pa nabor možnih vrednosti, v kolikor gre za nize znakov. Primer tako parametriziranega ukaza je sledeč:

```
run("Enhance Contrast...", "Saturated pixels=
    $i{0.1, 2.0}%");
```

Nabor možnih vrednosti lahko uporabimo tudi za opcijske operacije za procesiranje slike pred štetje objektov, za katere nismo prepričani, ali bi jih uporabili ali ne. V tem primeru imamo nabor dveh možnih vrednosti: ukaza ali ničesar (praznega niza):

```
$o{run("Fill Holes");,}
```

Del programa Learn123 je tudi vnaprej pripravljena predloga z osnovnim naborom ukazov in parametrov (priloga 1).

3.2.2 Genetski algoritem

Genetski algoritmi so vrsta optimizacijskih algoritmov, ki delujejo po vzoru naravne selekcije in za optimiziranje reštive uporabljajo biološke tehnike, kot so selekcija, križanje in mutacija [12].

Algoritem delovanje začne z množico rešitev problema (populacijo), ki jo predstavlja 30 kromosomov, sestavljenih iz genov. V programu Learn123 vsak kromosom predstavlja en makro, geni pa so parametri, ki jih optimiziramo. Začetna populacija je generirana tako, da makroje (kromosome) iz predloge ustvarimo tako, da naključno izberemo vrednosti parametrov za vsak kromosom.

V vsaki generaciji genetskega algoritma kromosome ocenimo, da lahko izberemo predstavnike, ki jih bomo križali med seboj (opravimo selekcijo).

Ocenjevanje kromosomov poteka tako, da z makrojem, ki ga predstavlja posamezen kromosom, avtomatsko preštejemo celice na vseh slikah iz učne množice. Rezultat avtomatskega štetja za vsako sliko nato primerjamo z ročnimi oznakami, ki so podane na začetku. V sodih iteracijah primerjamo število preštetih celic, v lihih pa razdalje med označenimi točkami. Končna ocena kromosoma je vsota razlik med številom preštetih celic oziroma vsota razdalj med označenimi točkami za vse slike iz učne množice.

Križamo pare kromosomov, ki jih izberemo glede na padajoče ocene. Najbolje ocenjena predstavnika izberemo kot prvi par, drugo- in tretje-najbolje ocenjenega kot drugi par in tako dalje. Izbrane pare križamo med seboj tako, da za vsak par najprej naključno izberemo točko križanja in glede na to razdelimo njune gene na dva dela. Prvega otroka sestavimo iz prvega dela genov prvega starša in drugega dela genov drugega starša, drugega otroka pa iz prvega dela genov drugega starša in drugega dela genov prvega starša.

Vsak otrok je z določeno verjetnostjo podvržen še mutaciji, ki poteka tako, da naključno izbranemu genu danega kromosoma zamenjamo vrednost.

Po selekciji, križanju in mutaciji vse tri skupine kromosomov (starše oz. predstavnike prejšnje generacije, otroke oz. podmladek in mutiran podmladek) ponovno ocenimo, razporedimo po padajočih ocenah in izberemo prvih 30 najboljših, ki preživijo do naslednje generacije.

Ta postopek ponavljamo, dokler ne pridemo do 30. generacije. Na koncu kromosome ponovno razporedimo po padajočih ocenah, tako da je prvi kromosom (makro) najboljši rezultat učenja. Postopek učenja lahko po potrebi ponovimo večkrat. Glede na učno množico in uspešnost učenja lahko spremenimo tudi velikost populacije, število iteracij, ki jih opravimo v enem sklopu učenja ali verjetnost mutacije podmladka, ki lahko močno vpliva na uspešnost algoritma.

Poglavje 4

Posodobitev programa Learn123

Eden izmed ciljev diplomskega dela je posodobiti program Learn123 tako, da namesto ImageJ1 uporabimo ImageJ2. V prvotni implementaciji Learn123 pri učenju štetja kromosome ocenjujemo zaporedno, saj lahko zaradi implementacije ImageJ1 API-ja naenkrat izvajamo samo en makro na eni sliki in z vsako izvedbo štetja objektov prepišemo rezultate prejšnje. Z uporabo ImageJ2 smo želeli učenje pospešiti, saj nova verzija omogoča, da naenkrat uporabljamo več instanc programa. To smo želeli izkoristiti za paralelizacijo ocenjevanja tako, da bi makroje, ki jih predstavljajo kromosomi, izvajali hkrati, in sicer vsakega v svoji instanci programa.

V nadaljevanju opišemo potek posodobitve programa, primerjamo različne načine implementacije ocenjevanja kromosomov z ImageJ2 in na primeru programa Learn123 komentiramo, kako poteka prehod z ImageJ1 na ImageJ2 za obstoječe uporabnike vmesnika za programiranje ImageJ1.

Izvorna koda posodobljenega programa je na voljo na spletnem naslovu `https://bitbucket.org/chibo17/learn123/`.

4.1 Ocenjevanje kromosomov z ImageJ2

Makroji so v ImageJ2 obravnavani kot skripte napisane v jeziku ImageJ Macro. Z ImageJ2 API-jem vse skripte izvajamo s pomočjo storitve za skriptiranje knjižnice SciJava Commons (`ScriptService`), ki jo ima vsaka instanca programa ImageJ. Skripto izvedemo s klicem metode `run`, implementirane v razredu `ScriptService`.

Makro tako izvedemo z naslednjim klicem, kjer je `ij` instanca programa ImageJ, `macro` pa znakovni niz ukazov v jeziku ImageJ Macro:

```
ij.script.run("macro.ijm", macro, false)
```

V ImageJ2 je možno skriptam podati vhodne in izhodne parametre [35], kar pomeni, da bi lahko naši skripti (makroju) kot vhodni parameter podali sliko (objekt `ImagePlus`), kot izhodni parameter pa bi skripta vrnila rezultate analize v obliki objekta `ResultsTable`. Ker je jezik ImageJ Macro omejen samo na numerične tipe in znakovne nize in so tako omejeni tudi parametri, takšna uporaba makrojev ni mogoča.

4.2 Prehod z jezika ImageJ Macro na Python

Zaradi omejitev jezika ImageJ Macro smo se odločili za menjavo skriptnega jezika. ImageJ2 za skriptiranje poleg makrojev podpira še programske jezike Groovy, Python, JavaScript, Ruby, Lisp in R [36]. Zaradi preproste sintakse in razširjenosti smo izbrali Python oziroma njegovo implementacijo Jython, ki jo podpira ImageJ2.

Jython je implementacija programskega jezika Python, ki je prilagojena za izvajanje na JVM. V programih, napisanih v Jythonu, lahko uporabljamo vse razrede napisane v Javi. Uporabljamo lahko tudi skoraj vse module, ki so na voljo v Pythonu, z izjemo nekaterih CPython modulov, ki so napisani v programskem jeziku C [19].

Osnovno predlogo (priloga 1), napisano v jeziku ImageJ Macro, smo prepisali v Jython. Ohranili smo enako sintakso za posplošitev parametrov, le

da smo vejico nadomestili s podpičjem:

- `$i{n;m}` - interval vrednosti celoštevilskega parametra,
- `$d{x;y}` - interval vrednosti realnega parametra,
- `$o{a;b;c;d}` - nabor možnih znakovnih vrednosti parametra.

Novi predlogi (priloga 2) smo dodali tudi vhodne in izhodne parametre. Pri skriptiranju v ImageJ2 morajo biti le-ti deklarirani na začetku skripte. V vsaki vrstici je lahko samo ena deklaracija parametra, ki se začne z oznako `#@`.

Primer deklaracije vhodnega parametra, kjer je `Type` podatkovni tip parametra, `variableName` pa ime spremenljivke, ki ga predstavlja:

```
#@ Type variableName
```

Podobno deklariramo izhodne parametre, le da začetku deklaracije dodamo ključno besedo `output`:

```
#@output Type outputName
```

Nabor podatkovnih tipov, ki jih lahko uporabimo kot parametre pri skriptiranju, je omejen pri vseh skriptnih jezikih, ki so podprti v ImageJ2. Poleg numeričnih tipov, Boolovih vrednosti, znakov in znakovnih nizov, so podprti samo še tipi `Dataset`, `ImagePlus`, `ColorRGB`, `Date` in `File` [35]. Ker med njimi ni podatkovnega tipa `ResultsTable`, ki smo ga želeli uporabiti za izhodni parameter, smo rešitev implementirali tako, da skripta namesto rezultatov analize vrne obdelano sliko, ki ima tabelo z rezultati shranjeno med svojimi lastnostmi (ang. *properties*).

V predlogi smo vhodni parameter poimenovali `img`, izhodnega pa `output`:

```
#@ ImagePlus img
#@output ImagePlus output
```

Izhodni parameter izvedene skripte v programu Learn123 dobimo na naslednji način:

```
val resultImage: ImagePlus = ij.script.run("script.py",
   script, false, "img", img.duplicate).get.getOutput("
   output").asInstanceOf[ImagePlus]
```

Tabela rezultatov analize je med lastnostmi slike shranjena pod imenom (ključem) `results`:

```
val resultsTable: ResultsTable = resultImage.getProperty
   ("results").asInstanceOf[ResultsTable]
```

Kromosomi v posodobljeni verziji Learn123 ne predstavljajo več makrojev, temveč skripte v programskem jeziku Jython, zato v nadaljevanju namesto o izvajanju makrojev pišemo o izvajanju skript.

4.3 Primerjava različnih implementacij ocenjevanja kromosomov

Z uporabo Pythona za skriptiranje in uvedbo vhodnih in izhodnih parametrov skript lahko kromosome ocenjujemo z ImageJ2. Skripte izvajamo z instanco programa ImageJ oziroma s storitvijo za skriptiranje, ki ji pripada. Ocenjevanje lahko paraleliziramo tako, da kromosome ocenjujemo sočasno, vsakega s svojo instanco ImageJ, uporaba vhodnih in izhodnih parametrov pa nam omogoča tudi paralelno izvajanje skript v eni instanci ImageJ.

Odločili smo se preizkusiti, kateri izmed omenjenih načinov paralelizacije je bolj učinkovit in kako se primerjajo z zaporednim ocenjevanjem kromosomov z ImageJ2. Implementirali smo zaporedno ocenjevanje kromosomov in več različic paralelnega ocenjevanja kromosomov ter implementacije testirali z izbrano učno množico.

4.3.1 Implementacije ocenjevanja kromosomov

Ocenjevanje kromosomov smo implementirali na štiri različne načine, od tega gre pri treh za paralelno ocenjevanje. Pri vseh treh paralelizacijah ocenje-

vanja kromosomov smo uporabili paralelne zbirke (ang. *parallel collections*), ki so implementirane v programskem jeziku Scala. Iz seznama kromosomov (`chs`), smo s klicem metode `par` dobili njegovo pripadajočo paralelno zbirko. Na njej smo potem paralelno izvedli operacijo `map`, s katero smo zbirko kromosomov preslikali v zbirko ocen kromosomov:

```
val chromosomes: ParSeq[Chromosome] = chs.par
val results: List[Evaluation] = chromosomes.map(
   evalOne).toList
```

Velika prednost uporabe paralelnih zbirk v Scali je, da ima vsaka svojo zalogo niti (ang. *thread pool*), s katerimi lahko razpolaga in tako mednje učinkovito razporeja opravila. Ni se nam torej potrebno ukvarjati z ustvarjanjem novih niti ali določanjem velikosti zaloge niti, saj je to učinkovito implementirano v paralelnem ogrodju Scale [26, 28].

I Zaporedno ocenjevanje vseh kromosomov v eni instanci programa ImageJ

Zaporedno ocenjevanje vseh kromosomov v eni instanci programa ImageJ je enakovredno zaporednemu ocenjevanju makrojev z ImageJ1. Na začetku učenja ustvarimo eno instanco ImageJ:

```
val ij: ImageJ = new ImageJ
```

To med učenjem nato uporabljamo za zaporedno izvajanje skript.

II Paralelno ocenjevanje kromosomov v eni instanci programa ImageJ

Kromosome lahko zaradi uporabe vhodnih in izhodnih parametrov paralelno ocenjujemo v eni instanci programa ImageJ. Na začetku učenja ustvarimo eno instanco ImageJ, ki jo med učenjem nato uporabljamo tako, da s pomočjo paralelnih zbirk v Scali paralelno z njo izvajamo več skript (ocenjujemo več kromosomov).

III Paralelno ocenjevanje kromosomov v novih instancah programa ImageJ

ImageJ2 API je implementiran tako, da lahko hkrati uporabljamo več instanc programa. To je mogoče, ker ima vsaka aplikacija, napisana z uporabo knjižnice SciJava Commons, lasten kontekst (`Context`), ki inicializira in skrbi za seznam storitev SciJava te aplikacije [6].

Tako lahko implementiramo našo prvotno idejo paralelizacije učenja štetja. Kromosome ocenimo tako, da na začetku ocenjevanja posameznega kromosoma ustvarimo novo instanco programa ImageJ in z njo izvedemo skripto na vseh slikah, ki pripadajo učni množici. Ocenjevanje izvajamo paralelno.

Pri takšnem načinu ocenjevanja s konstantnim ustvarjanjem instanc ImageJ porabljamo veliko pomnilnika. Da nam slednjega ne bi zmanjkalo, po končanem ocenjevanju kontekst instance zavržemo in ga tako pripravimo za čiščenje pomnilnika (ang. *garbage collection*). To storimo s klicem metode `dispose`:

```
val ij: ImageJ = new ImageJ
...
ij.context.dispose
```

IV Paralelno ocenjevanje z novimi instancami storitve za skriptiranje SciJava

Za ocenjevanje kromosomov ne potrebujemo vseh funkcionalnosti programa ImageJ, temveč le dve storitvi SciJava: `ScriptService`, ki jo uporabljamo za izvajanje skript in `WidgetService`, ki omogoča uporabo vhodnih parametrov pri skriptiranju.

Ustvarjanje instanc ImageJ je časovno zahtevno. Ker se zavedamo, da vključuje kreiranje objektov, ki jih za ocenjevanje kromosomov ne potrebujemo, smo implementirali različico, v kateri za ocenjevanje posameznega kromosoma ustvarimo samo kontekst z nujno potrebnima storitvama:

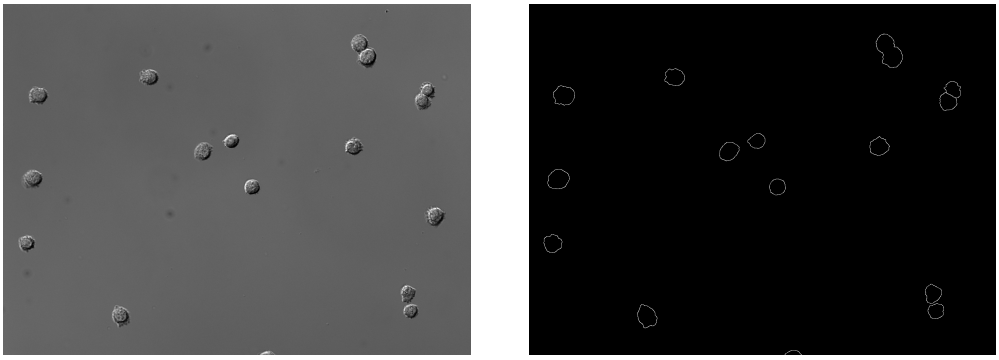
```
val context = new Context(classOf[ScriptService],
   classOf[WidgetService])
val scriptService = context.service(classOf[
   ScriptService]).asInstanceOf[ScriptService]
scriptService.addAlias(classOf[ImagePlus])
```

Skripto potem izvedemo s storitvijo za skriptiranje, ki jo inicializira kontekst (`scriptService` v odseku kode zgoraj). Enako kot pri prejšnjem načinu paralelizacije tudi v tej implementaciji po končanem ocenjevanju kontekst zavržemo.

4.3.2 Učna množica, uporabljena za testiranje implementacij

Implementacije smo testirali z izvajanjem učenja na različno velikih učnih množicah slik. Za testiranje smo uporabili mikroskopske slike celic CHO (ang. *Chinese Hamster Ovary cells*) [4] iz množice slik BBBC030 [3], ki je prostodostopna v okviru zbirke Broad Bioimage Benchmark Collection (BBBC) [22].

Množico BBBC030 sestavlja 60 mikroskopskih slik celic CHO. Na voljo so tudi referenčne slike z obrisi celic, s katerimi smo si pomagali pri označevanju slik. Primer mikroskopske slike celic in pripadajoče referenčne slike prikazuje slika 4.1.



Slika 4.1: Primer slike celic CHO iz množice BBBC030 in pripadajoče slike obrisov celic

Implementacije učenja smo testirali z učnimi množicami velikosti 10, 20 in 30 slik.

4.3.3 Kriteriji primerjave implementacij

Implementacije učenja smo primerjali glede na tri kriterije:

1. **Čas izvajanja** - zanimal nas je dejanski čas izvajanja učenja posameznih implementacij. Čas izvajanja je najpomembnejši kriterij, saj je glavni cilj paralelizacije pohitritev učenja.

2. **Procesorski čas** (ang. *CPU time*) - zanimalo nas je, koliko procesorskega časa za izvajanje potrebujejo različne implementacije. To je čas, ki ga za procesiranje ukazov programa porabi centralno procesna enota. Ta kriterij je posebej zanimiv pri paralelnem procesiranju, saj nam pomaga oceniti, kako zahteven je program za procesiranje. Pri paralelnem procesiranju naenkrat izkoriščamo več jeder našega procesorja (CPU), procesorski čas pa nam pove, koliko časa so vse centralno procesne enote skupaj porabile za izvajanje našega programa [7].

3. **Poraba pomnilniškega prostora** - opazovali smo, koliko pomnilniškega prostora pri učenju porabijo različne implementacije.

4.3.4 Sistem, uporabljen za testiranje implementacij

Učenje smo z različnimi implementacijami izvajali na prenosnem računalniku Apple MacBook Pro z naslednjimi specifikacijami:

- CPU: Intel Core i7 @ 2,5 GHz
- RAM: 16 GB
- GPU: AMD Radeon R9 M370X
- OS: MacOS Mojave 10.14.3

Programsko kodo, napisano v Scali, smo izvajali z orodjem *sbt*, ki smo mu za izvajanje dodelili 2 GB pomnilniškega prostora [30].

4.3.5 Rezultati testiranja implementacij

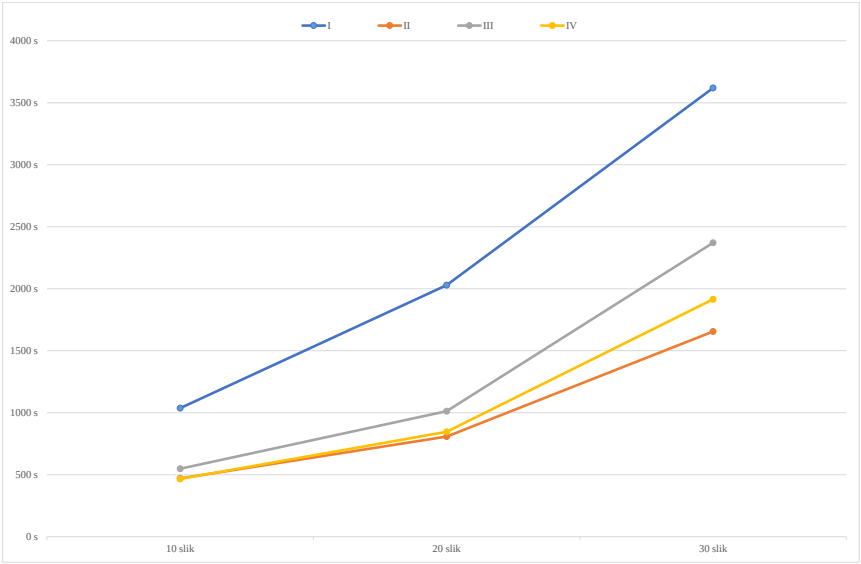
Pred testiranjem različnih implementacij ocenjevanja kromosomov moramo pomisliti tudi na implementacijo samega genetskega algoritma, s pomočjo katerega poteka učenje štetja celic. Na koncu vsake iteracije algoritma novo generacijo kromosomov sestavimo iz najboljših predstavnikov prejšnje generacije in njihovega podmladka. V izogib večkratnemu ocenjevanju kromosomov, ki preživijo več generacij, ocene kromosomov med učenjem shranjujemo. Zato se moramo zavedati, da pri vsakem izvajanju učenja ne ocenimo nujno istega števila kromosomov, saj je število ocenjevanj odvisno od sestava generacij. Pri testiranju smo zato za vse velikosti učnih množic slik z vsako implementacijo učenje izvedli trikrat in za primerjavo uporabili povprečne vrednosti meritev posameznih kriterijev.

Za lažjo predstavitev in komentiranje rezultatov testiranja različne implementacije označimo kot v razdelku 4.3.1. Rezultate meritev testiranja prikazuje tabela 4.1.

		I	II	II	IV
10 slik	Čas izvajanja	1037 s	470 s	548 s	466 s
	Procesorski čas	1170 s	1843 s	2573 s	1828 s
	Pomnilnik	1,47 GB	2,52 GB	2,50 GB	2,50 GB
20 slik	Čas izvajanja	2029 s	808 s	1012 s	846 s
	Procesorski čas	2183 s	2916 s	4084 s	3009 s
	Pomnilnik	1,67 GB	2,49 GB	2,51 GB	2,51 GB
30 slik	Čas izvajanja	3619 s	1655 s	2371 s	1915 s
	Procesorski čas	3809 s	5170 s	9108 s	6045 s
	Pomnilnik	1,89 GB	2,5 GB	2,53 GB	2,52 GB

Tabela 4.1: Rezultati testiranja učenja z različnimi implementacijami ocenjevanja kromosomov pri različnih velikostih učnih množic

Primerjava implementacij glede na čas izvajanja

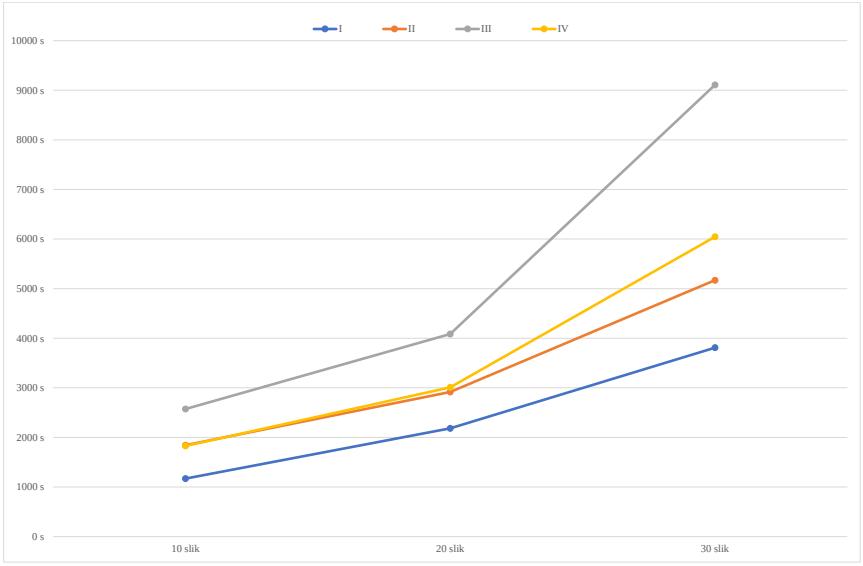


Slika 4.2: Rezultati testiranja časa izvajanja

Slika 4.2 prikazuje rezultate testiranja časa učenja z vsemi štirih implementacijami ocenjevanja kromosomov. Vidimo lahko, da je čas učenja pri vseh treh paralelnih implementacijah (II, III in IV) manjši od časa učenja pri implementaciji I, kjer kromosome ocenjujemo zaporedno.

Čas izvajanja učenja je, če kromosome ocenjujemo paralelno, približno 50 % manjši kot pri zaporednem ocenjevanju kromosomov. Razlika med časi izvajanja paralelnih implementacij se z večanjem učne množice povečuje. Za najhitrejše se izkaže paralelno ocenjevanje kromosomov v eni instanci programa ImageJ (III). To pri učni množici s 30 slikami učenje pohitri kar za 33 minut.

Primerjava implementacij glede na procesorski čas



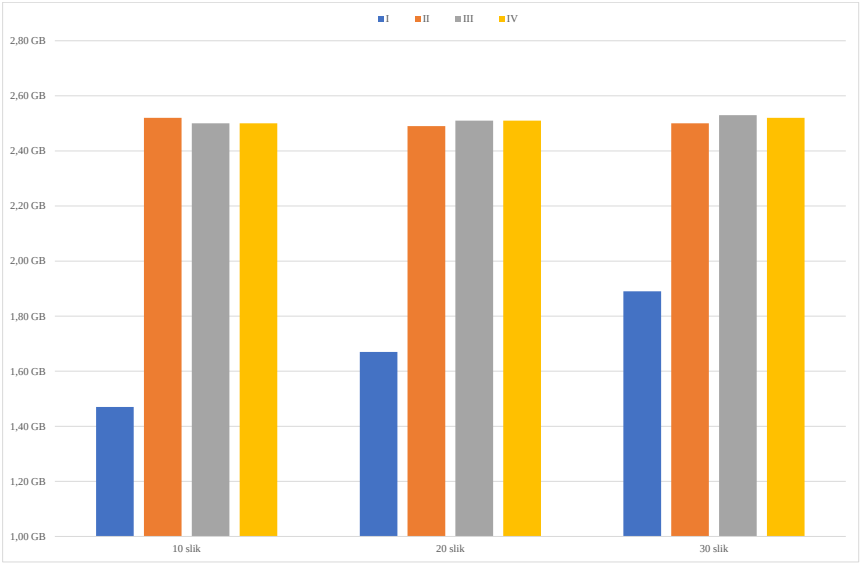
Slika 4.3: Rezultati testiranja procesorskega časa

Pri rezultatih merjenja procesorskega časa, ki ga posamezne implementacije porabijo pri učenju, opazimo ravno obraten vzorec kot pri času izvajanja.

Na sliki 4.3 lahko vidimo, da je za učenje s paralelnim ocenjevanjem kromosomov (II, III in IV) potrebno več procesorskega časa kot za učenje, kjer je ocenjevanje implementirano zaporedno (I).

Za procesor je najbolj zahtevno paralelno ocenjevanje kromosomov v novih instancah programa ImageJ (III), kar je smiselno, saj je ustvarjanje novih instanc programa zahtevna operacija. Paralelno ocenjevanje kromosomov z novimi instancami storitve za skriptiranje SciJava (IV) je, kot smo pričakovali, hitrejše od ocenjevanja z novimi instancami ImageJ, vendar pa je razlika v procesorskem času bolj opazna. Zagotovo lahko torej potrdimo predvidevanje, da je implementacija IV boljša od implementacije III. Enako kot pri kriteriju času izvajanja se je med paralelnimi implementacijami najboljše izkazalo paralelno ocenjevanje kromosomov v eni instanci programa ImageJ (II).

Primerjava implementacij glede na porabo pomnilniškega prostora



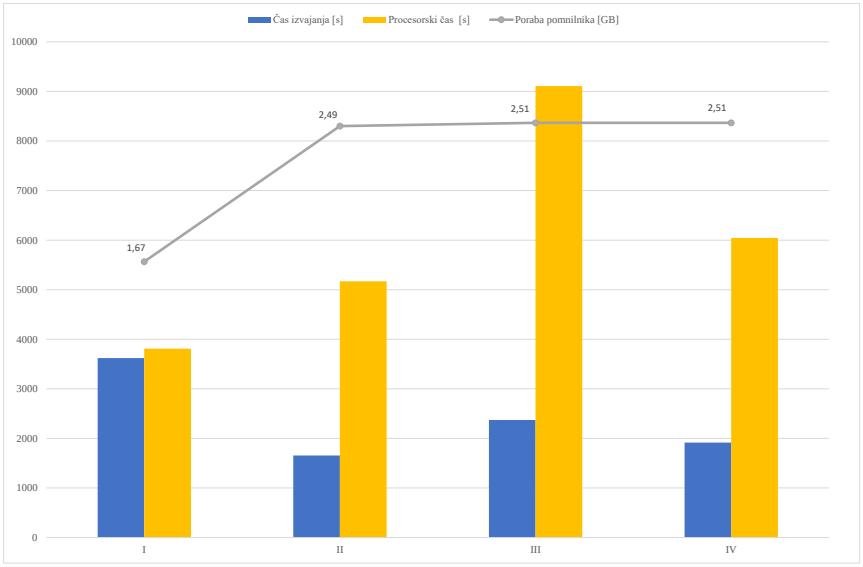
Slika 4.4: Rezultati testiranja porabe pomnilnika

Na sliki 4.4, ki prikazuje količino porabe pomnilniškega prostora pri učenju štetja z različnimi implementacijami ocenjevanja kromosmov vidimo, da za zaporedno ocenjevanje potrebujemo manj pomnilniškega prostora. Poraba je pri paralelnih implementacijah (II, III in IV) skoraj enaka.

Povzetek rezultatov

Pri vseh velikostih učnih množic, uporabljenih za testiranje, lahko vidimo enak trend rezultatov meritev. Slika 4.5 prikazuje rezultate meritev pri učenju z učno množico z 20 slikami.

Povzamemo lahko, da je učenje s paralelnim ocenjevanjem kromosomov hitrejše od tistega z zaporednim, obenem pa zanj potrebujemo več procesorskega časa in pomnilniškega prostora. Najboljša implementacija, ki je obenem najhitrejša in procesorsko najmanj zahtevna, pa je učenje z ocenjevanjem kromosomov v eni instanci programa ImageJ.



Slika 4.5: Rezultati testiranja implementacij učenja z učno množico velikosti 20 slik

4.4 Težave pri prehodu z ImageJ1 na ImageJ2

Med posodabljanjem programa Learn123 smo dodobra spoznali razlike med ImageJ1 in ImageJ2, njuno arhitekturo in vmesnikoma za programiranje. ImageJ2 na prvi pogled prinaša veliko izboljšav, vendar smo pri prehodu z ImageJ1 API-ja na ImageJ2 API naleteli na več težav. Opazili smo tudi marsikatere slabosti novega API-ja.

4.4.1 Dokumentacija

Prva težava, s katero smo se soočili pri prehodu z ImageJ1 na ImageJ2, je pomanjkanje, ponavljanje in nedoslednost dokumentacije. Na portalu ImageJWiki je objavljenih veliko strani o ImageJ2, vendar gre predvsem za opise arhitekture, informacije na njih se v veliki meri ponavljajo, besedišče pa ni vedno konsistentno.

Za razvijalce, ki se prvič lotevajo programiranja z ImageJ2 API-jem, razumevanje arhitekture in delovanja API-ja traja veliko dalj časa, kot bi to pričakovali po branju o preprostosti integracije ImageJ2 z drugimi programi. Zaradi obsežne arhitekture in veliko nivojev abstrakcije, je tudi iskanje po dokumentaciji Java (ang. *Javadocs*) zelo zamudno. Pogosto je težko najti v sklopu katere komponente ImageJ2 je implementirana željena funkcija.

4.4.2 Pisanje skript v Pythonu

Pri pripravljanju nove predloge skripte v Pythonu (priloga 2) smo sledili dokumentaciji o skriptiranju v Jythonu [20] in objavljenim vzorcem skript. Po vzoru dokumentacije smo pri pisanju skripte uporabili ImageJ1 API. Kljub popolnoma novi arhitekturi ImageJ2, zaradi katere naj bi bil API bolj intuitiven in lažji za uporabo, je vsaj po dokumentaciji sodeč ImageJ1 API še vedno ključen za skriptiranje.

Predvidevamo, da razlog za to leži v dejstvu, da je uporaba ImageJ1

API-ja veliko bolj preprosta. Ukaze ImageJ tako recimo izvajamo s klicem metode `IJ.run`, ki ji podamo ime ukaza in njegove parametre:

```
IJ.run("Normalisation", "");
```

Implementirana je tudi metoda, s katero ukaz izvedemo na izbrani sliki (objektu `ImagePlus`):

```
IJ.run(img, "Normalisation", "");
```

Za izvedbo istega ukaza z ImageJ2 API-jem je potrebno veliko več programerskega znanja in razumevanja arhitekture programa [15]:

```
ij.command().run(ImageNormalizerIJ2Plugin.class, true, "
   input", img, "ij", ij);
```

V tem vidimo slabost novega API-ja in predvidevamo, da bo zaradi njegove kompleksnosti večina uporabnikov ostala pri ImageJ1 API-ju, v kolikor ta zadošča njihovim potrebam.

4.4.3 Izvajanje skript z ImageJ2 API-jem

Pri izvajanju skript z ImageJ2 smo pri testiranju vsakič, ko smo ustvarili novo instanco programa ali storitve za skriptiranje, dobili naslednjo napako:

```
[ERROR] Cannot create plugin: class='org.scijava.plugins
   .scripting.javascript.JavaScriptScriptLanguage', name
   ='JavaScript', priority=0.0, enabled=true, pluginType
   =ScriptLanguage
java.lang.IllegalArgumentException: No such script
   engine: javascript
```

Napaka nas je presenetila predvsem zato, ker pri skriptiranju nismo uporabljali programskega jezika JavaScript. Izkaže se, da se napaka pri uporabi ImageJ z Javo 8 na operacijskem sistemu OS X pojavi ne glede na uporabljen jezik skriptiranja [14, 24]. Kljub napaki pa ImageJ potem deluje brez težav.

Čeprav ne vpliva na delovanje ImageJ2, je napaka pri uporabi ImageJ2 API-ja za skriptiranje lahko moteča, predvsem če orodno vrstico sicer uporabljamo za izpisovanje podatkov. Zato jo tukaj izpostavljamo kot primer nedodelanosti ImageJ2 API-ja.

4.4.4 ImageJ Legacy

Pri uporabi ImageJ1 API-ja se je pri izvajanju makrojev ob vsaki izvedbi ukaza *Analyze Particles* odprlo ali posodobilo okno, ki je prikazovalo rezultate detekcije objektov. V ImageJ2 je nadzor nad odprtimi okni in uporabniškim vmesnikom programa implementiran s storitvama SciJava za upravljanje z okni (`WindowService`) in za upravljanje z uporabniškim vmesnikom (`UIService`). Glede na dokumentacijo lahko z njuno pomočjo ImageJ2 API uporabljamo brez uporabniškega vmesnika oz. nadziramo (in ročno zapiramo) vsa odprta okna.

Pričakovali smo, da bomo z uporabo ImageJ2 rešili to težavo, ki se je pojavljala tudi, ko smo menjali jezik skriptiranja, saj smo pri pisanju skript po vzoru dokumentacije uporabili ImageJ1 API. Izkazalo se je, da je v ImageJ Legacy implementiran samo nadzor nad glavnim oknom uporabniškega vmesnika in okni, ki prikazujejo slike [1]. Na odpiranje vseh ostalih oken nimamo vpliva, tako da se tudi pri izvajanju skript z novim API-jem ne moremo izogniti odpiranju okna z rezultati.

Na primeru programa Learn123 vidimo, da ImageJ Legacy v resnici samo pokrpa razlike v arhitekturi med ImageJ1 in ImageJ2 tako, da lahko metode, implementirane v ImageJ1, uporabljamo tudi v ImageJ2. Uporaba kombinacije API-jev ImageJ1 in ImageJ2 je sicer zaradi ImageJ Legacy precej preprosta, vendar pa z njeno uporabo ne moremo nujno izkoristiti novosti in storitev, ki jih ponuja ImageJ2.

Edina možnost, da popolnoma izkoristimo ImageJ2 je, da celoten program prepišemo tako, da ImageJ1 in ImageJ Legacy popolnoma nadomestimo z novim API-jem. To je zelo zamudno delo, zato se moramo vprašati, ali je smiselno.

4.4.5 Premalo pomnilniškega prostora pri testiranju implementacij

Pri testiranju različnih implementacij ocenjevanja kromosomov smo sprva imeli težave zaradi premajhne količine pomnilniškega prostora, s katerim je razpolagalo orodje sbt. Med izvajanjem se je pojavilo okno uporabniškega vmesnika ImageJ1 z obvestilom o napaki, ki ga prikazuje slika 4.6, medtem ko v orodni vrstici, kjer smo izvajali implementacije, obvestila o napaki nismo dobili.

Sledili smo navodilom v obvestilu o napaki in v predlogi skripte poskusili spremeniti količino pomnilniškega prostora, ki ga lahko uporablja program ImageJ. Pri tem smo dobili napako, da za spreminjanje količine pomnilniškega prostora nimamo dovoljenja. Iskali smo način, kako bi to storili z ImageJ2 API-jem, vendar ga nismo našli.

Slika 4.6: Obvestilo o napaki

Zaradi opisane težave prvi rezultati testiranja niso bili pravilni in učenje s paralelnim ocenjevanjem kromosomov ni bilo skorajda nič hitrejše od implementacije, v kateri smo kromosome ocenjevali zaporedno. Težavo smo rešili šele kasneje, ko smo orodju sbt iz drugih razlogov za izvajanje namesto 1 GB dodelili 2 GB spomina.

Poglavje 5

Nov primer uporabe programa Learn123

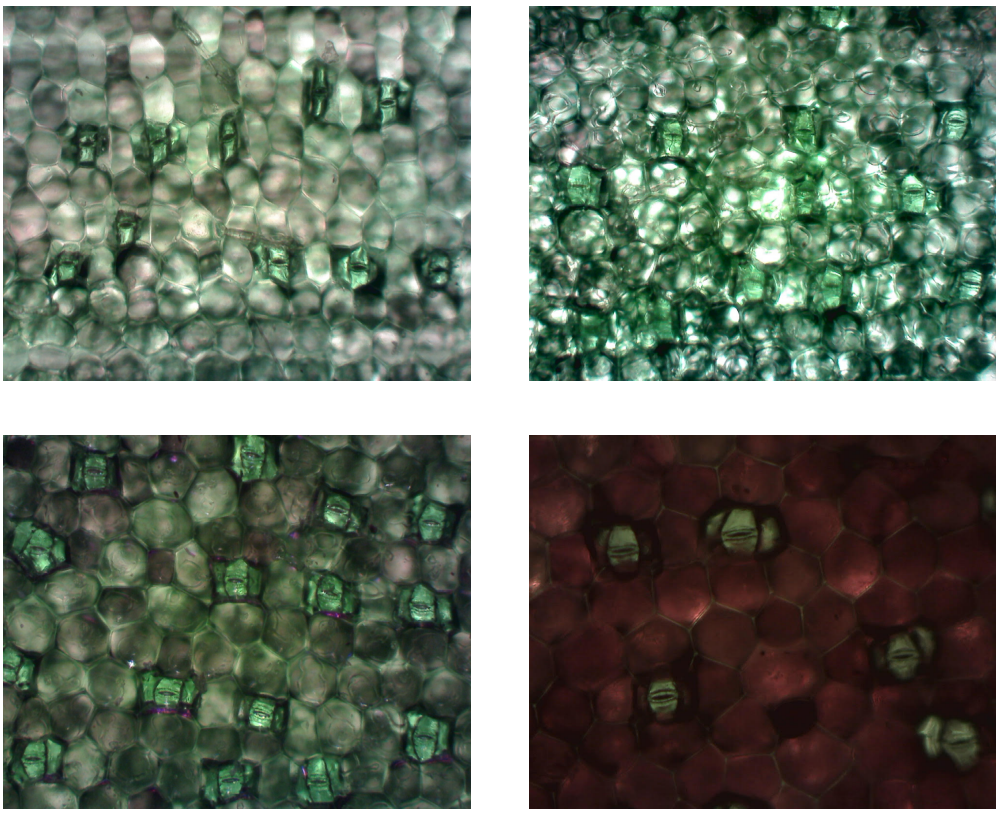
Poleg posodobitve programa Learn123 smo v sklopu diplomskega dela želeli najti novo domeno, na kateri bi lahko uporabili posodobljen program Learn123. Področje uporabe smo s štetja celic relativno preproste okrogle oblike želeli razširiti na štetje zahtevnejših objektov na slikah.

Z Learn123 smo z uporabo ImageJ poskusili avtomatsko šteti objekte na treh različnih bioloških domenah, ki jih predstavimo v nadaljevanju. Ob opisu posameznih poskusov izpostavimo tudi omejitve programa ImageJ, s katerimi smo se srečali in utemeljimo sklep, da je ImageJ najbolj primeren predvsem za štetje preprostih okroglih celic.

5.1 Štetje listnih rež

Prva domena, na kateri smo poskusili uporabiti program Learn123, so mikroskopske slike listov rastline zebrasta tradeskancija (lat. *Tradescantia zebrina*). Na slikah smo želeli šteti oziroma detektirati listne reže, ki so za raziskovalce na področju biologije pomembne, ker lahko s spremljanjem odpiranja in zapiranja listnih rež določamo stanje rastline. Glede na velikost rež, ko so odprte, lahko na primer ugotovimo, koliko vlage potrebuje rastlina.

Za učenje štetja celic smo uporabili nabor mikroskopskih slik, ki je nastal pri umetniškem projektu *Nociceptor: Branje z ustnic* intermedijske umetnice Špele Petrič [27].



Slika 5.1: Primeri mikroskopskih slik listov rastline zebrasta tradeskancija

Na sliki 5.1, ki prikazuje primere mikroskopskih slik listov, lahko vidimo, da se glede na osvetlitev in nastavitev fotoaparata, uporabljenega za zajem slik, le-te močno razlikujejo. Celice, v katerih se nahajajo reže, se človeškemu očesu na vseh slikah sicer zdijo izstopajoče, vendar z uporabo enega zporedja ukazov ImageJ za štetje objektov z istimi parametri ne moremo pravilno prešteti in označiti celic na vseh. Spomnimo se, da pri štetju objektov z ImageJ sliko z operacijami za procesiranje slik, ki so na voljo v programu, obdelamo tako, da iz nje dobimo binarno sliko, na kateri so samo objekti, ki nas zanimajo in te potem preštejemo. Zaradi različnih kontrastov, svetlosti

in barv slik, z enakimi ukazi za procesiranje slik na vseh slikah celic, v katerih so listne reže, ne moremo ločiti od ozadja.

Ob pregledu obstoječih metod za avtomatsko detekcijo listnih rež rastlin še dodatno potrdimo našo trditev, saj opazimo, da so za detekcijo rež v večini primerov uporabljene metode, ki so kompleksnejše od metode štetja objektov z ImageJ. Med bolj uveljavljene metode za detekcijo listnih rež sodijo na primer primerjanje predlog (ang. *template matching*) [21] ali pa kaskadna detekcija objektov (ang. *cascade object detection*) [18].

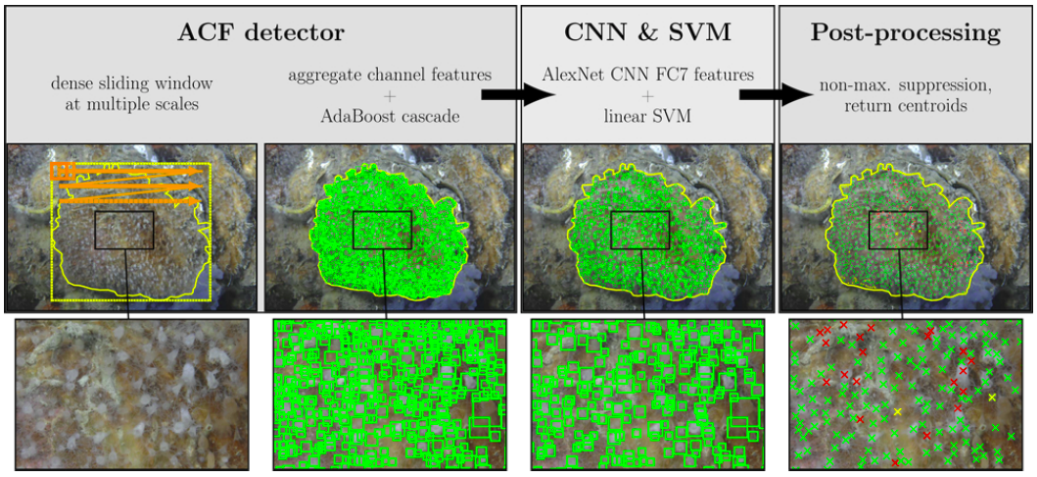
Omenjeni metodi lahko v programu ImageJ sicer uporabimo s pomočjo obstoječih uporabniških vtičnikov, vendar smo pri avtomatskem štetju objektov s programom Learn123 želeli ostati pri uporabi osnovnih funkcionalnosti programa ImageJ. Če bi uporabnikom Learn123 v skriptah dovoljevali uporabo pluginov, bi morali v Learn123 vključiti tudi orodje ImageJ Updater, s katerim bi vtičnike prenašali s spleta. Uporaba programa Learn123 bi v tem primeru zahtevala tudi internetno povezavo.

5.2 Štetje polipov meduz

Druga domena, ki smo se ji posvetili, je štetje polipov meduz na ostrigah. To na področju morske biologije pomembno za spremljanje spreminjanje velikosti populacij meduz v morju.

S štetjem polipov meduz so se na Fakulteti za računalništvo in informatiko ukvarjali v Laboratoriju za umetne vizualne spoznavne sisteme, kjer so v ta namen razvili aplikacijo PoCo [39]. Pregledali smo slike ostrig poraščenih s polipi meduz, ki so jih uporabili za učenje štetja. Na sliki 5.2 lahko vidimo, da se polipi med seboj v veliki meri prekrivajo, in da niso okrogle oblike, kot celice, ki smo jih s programom Learn123 avtomatsko šteli do sedaj. Za deljenje prekrivajočih se objektov na slikah je v ImageJ implementiran ukaz *Watershed*, ki pri deljenju objektov predvideva, da so le-ti okrogle oziroma elipsaste oblike. Ta je torej zadoščal pri štetju prekrivajočih se celic okrogle oblike, v primeru polipov pa zaradi njihovih manj pravilnih oblik ne bi prišel

v poštev. Za slike polipov podobno kot za slike listnih rež velja tudi, da se razlikujejo v osvetljenosti, kontrastu in ostrini.



Slika 5.2: Primer slike ostrige poraščene s polipi meduz in predstavitev poteka algoritma PoCov1

Slika 5.2 prikazuje tudi potek delovanja algoritma PoCov1, ki je enako kot algoritmi za detekcijo listnih rež očitno zahtevnejši od metode štetja objektov z ImageJ. V drugi verziji programa PoCov2 so algoritem za štetje polipov še izboljšali in dosegli zelo visoko pravilnost detekcije polipov, ki je uporabo z Learn123 in metodo štetja objektov z ImageJ ne moremo preseči.

5.3 Štetje ptic selivk v jatah

Za tretjo domeno smo izbrali določanje števila ptic na slikah jat ptic selivk. Učno množico slik smo v tem primeru sestavili sami. Izbirali smo 10 slik, za katere smo vedeli, da bomo število ptic na sliki lahko določili z uporabo programa ImageJ. Enako kot velja pri štetju polipov, z ukazi implementiranimi v ImageJ ne moremo šteti prekrivajočih se ptic, saj so ptice med letom na slikah različnih oblik glede na kot slike in zamahe kril. Podobno štetje objektov z ImageJ ni mogoče, če se objekti, ki nas zanimajo, od ozadja ne

ločujejo po barvi ali kontrastu. Izbirali smo torej slike z enostavnim (enobarvnim) ozadjem, na katerih se ptice ne (ali pa vsaj čimmanj) prekrivajo, kot na primer na sliki 5.3.

Slika 5.3: Primer slike jate ptic selivk [10]

Osnovna predloga za učenje štetja z genetskim algoritmom (priloga 2) za štetje ptic na barvnih slikah ne zadošča, zato smo jo prilagodili za izbrano množico slik. Na začetku predloge smo dodali ukaz, s katerim smo barvne slike spremenili v 8 bitne sivinske slike. Dodali smo tudi več opcijskih ukazov za obdelavo slike pred določanjem pragov slik in opcijskih ukazov za obdelavo binarnih slik.

Optimizirana skripta, ki je rezultat učenja štetja, dobro deluje za našo množico slik, kot lahko vidimo v tabeli 5.1. Odstopanja se pojavljajo predvsem pri slikah, na katerih se nekaj parov ptic prekriva. Pri štetju ptic na prvi sliki, predstavljeni med rezultati, pa vidimo primer, pri katerem je število ptic, ki ga je določil program Learn123 večje od ročno določenega števila. To se je zgodilo zaradi ptice z raztegnjenimi krili, ki smo jo pri obdelavi slike pred štetjem razdelili na več delov (trup in krila) in jih šteli kot ločene objekte. Optimizirano skripto kljub odstopanjem ocenjujemo kot dobro avtomatizacijo procesa štetja, vendar pa je ne moremo uporabiti za štetje ptic v jatah na poljubnih slikah. Težave se pojavijo takoj, ko se ptice na slikah prekrivajo ali se med seboj preveč razlikujejo.

Slika	Št. ptic (Learn123)	Št. ptic (ročno)
1	23	21
2	11	11
3	6	6
4	6	6
5	10	10
6	11	11
7	30	33
8	15	16
9	18	19
10	9	10

Tabela 5.1: Rezultati štetja ptic v jatah s programom Learn123 v primerjavi s številom ročno preštetih ptic na posameznih slikah

5.4 Uporaba programa ImageJ za štetje poljubnih objektov

Vsi trije poskusi uporabe Learn123 na novih domenah so nam pokazali, da ImageJ ni posebej prilagojen za štetje objektov na slikah. To je navadno implementirano z metodami, ki upoštevajo specifične lastnosti objektov, kot na primer obliko. ImageJ je namenjen predvsem procesiranju slik, medtem ko je za štetje objektov na voljo samo ena metoda, s katero lahko določimo število črnih objektov na binarnih slikah.

Povzamemo lahko, da uporaba programa Learn123 za štetje objektov na poljubnih domenah ni mogoča, saj sama programska oprema ImageJ ne ponuja dovolj splošnih metod, namenjenih štetju objektov. Zato je primeren predvsem za štetje celic preproste oblike na mikroskopskih slikah. Za te navadno velja, da vzorci, v katerih želimo šteti celice, ne vsebujejo velikega števila različnih tipov celic, ozadje slik vzorcev pa je enostavno.

Poglavje 6

Zaključek

V okviru diplomskega dela smo spoznali programsko opremo ImageJ, namenjeno procesiranju mikroskopskih slik. Seznanili smo se z osnovnimi operacijami za procesiranje slik, implementiranimi v ImageJ, in postopkom predprocesiranja in segmentiranja slik, ki je potrebno za štetje objektov na slikah z izbranim programom. Spoznali smo tudi omejitve programa ImageJ, namenjenega predvsem procesiranju slik, in težave, s katerimi se soočimo, če ga želimo uporabiti za štetje objektov na poljubnih vrstah slik.

Ker je bila prva verzija programa ImageJ1 razvita kot namizna aplikacija, je njena arhitektura močno vezana na uporabniški vmesnik, API pa pomanjkljiv. Pripravili smo pregled nove verzije programa ImageJ2 z novo modularno arhitekturo, ki naj bi omogočala tudi lažjo integracijo ImageJ z drugimi programi, torej vključevala boljši API.

Posodobili smo program Learn123, namenjen avtomatskemu štetju celic na mikroskopskih slikah z uporabo ImageJ. V novi verziji Learn123 za štetje celic uporabljamo ImageJ2, zamenjali pa smo tudi skriptni jezik in z jezika ImageJ Macro prešli na Python. Med procesom posodobitve smo identificirali pomanjkljivosti API-ja nove verzije ImageJ. Ugotovili smo, da je glavni vir težav pri programiranju z ImageJ2 API-jem komponenta ImageJ Legacy, ki skrbi za vzratno kompatibilnost med staro in novo verzijo programa. Ta je nujna zaradi obstoječe uporabniške skupnosti programa

ImageJ1, vendar pa poskrbi le za nujno premostitev razlik med verzijama programa za to, da lahko ukaze in programski vmesnik ImageJ1 uporabljamo tudi v ImageJ2. V primeru uporabe kombinacije ImageJ1 in ImageJ2 tako ne moremo izkoristiti vseh novosti ImageJ2, saj te niso nujno implementirane za komponente iz ImageJ1. Zaradi kompleksne arhitekture ImageJ2 in poslednično obsežnejšega API-ja ter dokumentacije polne primerov uporabe ImageJ1 API-ja, pa predvidevamo, da novi API še dolgo časa ne bo popolnoma nadomestil starega.

V programu Learn123 je učenje štetja celic implementirano z genetskim algoritmom. Z uporabo ImageJ2 API-ja smo pripravili štiri različne implementacije ocenjevanja kromosomov v genetskem algoritmu, od tega tri paralelne. Implementacije učenja z različnimi načini ocenjevanja kromosomov smo testirali na treh različno velikih učnih množicah in izbrali najboljšo. Pri testiranju se je najbolje izkazala implementacija, pri kateri kromosome paralelno ocenjujemo v eni instanci programa ImageJ. Z njo smo učenje v primerjavi z učenjem z zaporednim ocenjevanjem kromosomov pohitrili za približno 50 %.

Predstavili smo tudi nov primer uporabe programa Learn123, in sicer določanje števila ptic selivk v jatah. Learn123 oziroma ImageJ smo preizkusili tudi za štetje listnih rež rastline zebrasta tradeskancija in štetje polipov meduz na ostrigah ter opisali težave pri teh domenah. Sklenili smo, da je ImageJ glede na nabor implementiranih funkcionalnosti najbolj primeren za procesiranje mikroskopskih slik.

Ob pisanju o omejitvah programske opreme ImageJ je potrebno poudariti, da je genetski algoritem, implementiran v programu Learn123, zelo dober pri izbiri oziroma optimizaciji parametrov operacij, implementiranih v ImageJ. Learn123 bi zato v prihodnosti lahko razširili z uporabo katere druge knjižnice za procesiranje slik, ki vključuje več metod za štetje objektov. Dobra izbira bi bila knjižnica OpenCV (Open Source Computer Vision) [25]. OpenCV bi lahko z uporabo knjižnice IJ-OpenCV [13], namenjene komunikaciji med programsko opremo ImageJ in knjižnico OpenCV, integrirali v

program Learn123 in tako za predprocesiranje slik ohranili uporabo ImageJ, za štetje objektov pa uporabili metode implementirane v knjižnici OpenCV.

Literatura

[1] A problem running ImageJ in headless mode - Image.sc Forum. Dosegljivo: https://forum.image.sc/t/a-problem-running-imagej-in-headless-mode/3909/14. [Dostopano 20. 2. 2019].

[2] Amr Abd-Elrahman, Leonard Pearlstine, and F Percival. Development of pattern recognition algorithm for automatic bird detection from unmanned aerial vehicle imagery. *Surveying and Land Information Science*, 65:37–45, 03 2005.

[3] BBBC030: Chinese Hamster Ovary Cells. Dosegljivo: https://data.broadinstitute.org/bbbc/BBBC030/. [Dostopano 21. 2. 2019].

[4] Celice CHO. Dosegljivo: https://sl.wikipedia.org/wiki/Celice_CHO. [Dostopano 21. 2. 2019].

[5] Compatibility - ImageJ. Dosegljivo: https://imagej.net/Compatibility. [Dostopano 18. 2. 2019].

[6] Context (SciJava Javadocs 1.0.0-SNAPSHOT API). Dosegljivo: https://javadoc.scijava.org/SciJava/org/scijava/Context.html. [Dostopano 19. 2. 2019].

[7] CPU time - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/CPU_time. [Dostopano 28. 2. 2019].

[8] Sebastian De Boodt, Ahmad Poursaberi, Jan Schrooten, Daniel Berckmans, and Jean-Marie Aerts. A semi-automatic cell counting tool for

quantitative imaging of tissue engineering scaffolds. *Tissue engineering. Part C, Methods*, 19, 01 2013.

[9] Developing Plugins for ImageJ 1.x - ImageJ. Dosegljivo: `https://imagej.net/Developing_Plugins_for_ImageJ_1.x`. [Dostopano 18. 2. 2019].

[10] Eurasian Cranes migrating to Meyghan Salt Lake, Markazi Province of Iran. Dosegljivo: `https://en.wikipedia.org/wiki/V_formation#/media/File:Eurasian_Cranes_migrating_to_Meyghan_Salt_Lake.jpg`. [Dostopano 22. 2. 2019].

[11] Tiago Ferreria and Wayne Rasband. ImageJ User Guide IJ1.46r. Dosegljivo: `https://imagej.nih.gov/ij/docs/guide/user-guide.pdf`. [Dostopano 16. 2. 2019].

[12] Genetski algoritem. Dosegljivo: `http://wiki.fmf.uni-lj.si/wiki/Genetski_algoritem`. [Dostopano 28. 1. 2019].

[13] IJ-OpenCV. Dosegljivo: `https://github.com/joheras/IJ-OpenCV`. [Dostopano 23. 2. 2019].

[14] ImageJ cannot load the scripts Issue #12 imagej/imagej. Dosegljivo: `https://github.com/imagej/imagej/issues/116`. [Dostopano 20. 2. 2019].

[15] ImageJ1-ImageJ2 cheat sheet - ImageJ. Dosegljivo: `https://imagej.net/ImageJ1-ImageJ2_cheat_sheet`. [Dostopano 18. 2. 2019].

[16] ImgLib2. Dosegljivo: `https://imagej.net/ImgLib2`. [Dostopano 7. 2. 2019].

[17] Introduction to Macro Programming - ImageJ. Dosegljivo: `https://imagej.net/Introduction_into_Macro_Programming`. [Dostopano 16. 2. 2019].

[18] Hiranya Jayakody, Scarlett Liu, Mark Whitty, and Paul Petrie. Microscope image based fully automated stomata detection and pore measurement method for grapevines. *Plant Methods*, 13(1):94, Nov 2017.

[19] Jython. Dosegljivo: `https://wiki.python.org/jython/JythonFaq/GeneralInfo`. [Dostopano 7. 2. 2019].

[20] Jython Scripting - ImageJ. Dosegljivo: `https://imagej.net/Jython_Scripting`. [Dostopano 7. 2. 2019].

[21] H. Laga, F. Shahinnia, and D. Fleury. Image-based plant stornata phenotyping. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 217–222, Dec 2014.

[22] Vebjorn Ljosa, Katherine Sokolnicki, and Anne Carpenter. Annotated high-throughput microscopy image sets for validation. *Nature methods*, 9:637, 06 2012.

[23] Jasna Lojk, Uroš Čibej, David Karlaš, Luka Šajn, and Mojca Pavlin. Comparison of two automatic cell-counting solutions for fluorescent microscopic images. *Journal of microscopy*, 260(1):107–116, 2015.

[24] Make JavaScript work in Java 8 Issue #116 imagej/imagej. Dosegljivo: `https://github.com/imagej/imagej/issues/116`. [Dostopano 20. 2. 2019].

[25] OpenCV. Dosegljivo: `https://opencv.org/`. [Dostopano 23. 2. 2019].

[26] Parallel Collections Overview — Scala Documentation. Dosegljivo: `https://docs.scala-lang.org/overviews/parallel-collections/overview.html`. [Dostopano 19. 2. 2019].

[27] Pogovor z rastlino — Dnevnik. Dosegljivo: `https://www.dnevnik.si/1042861401`. [Dostopano 28. 2. 2019].

[28] Aleksandar Prokopec, Tiark Rompf, Phil Bagwell, and Martin Odersky. On a generic parallel collection framework. page 26, 2011.

[29] Curtis Rueden, Johannes Schindelin, Mark Hiner, Barry Dezonia, Alison E. Walter, and Kevin Eliceiri. Imagej2: Imagej for the next generation of scientific image data. *BMC Bioinformatics*, 18, 01 2017.

[30] sbt - The interactive build tool. Dosegljivo: `https://www.scala-sbt.org/index.html`. [Dostopano 24. 2. 2019].

[31] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: An open-source platform for biological-image analysis. *Nature methods*, 9:676–82, 06 2012.

[32] SCIFIO. Dosegljivo: `https://imagej.net/SCIFIO`. [Dostopano 7. 2. 2019].

[33] SciJava Javadoc. Dosegljivo: `https://scijava.org/javadoc.scijava.org/`. [Dostopano 18. 2. 2019].

[34] SciJava Organization Hierarchy - ImageJ. Dosegljivo: `https://imagej.net/File:SciJava_Organization_Hierarchy.svg`. [Dostopano 19. 2. 2019].

[35] Script parameters - ImageJ. Dosegljivo: `https://imagej.net/Script_Parameters`. [Dostopano 7. 2. 2019].

[36] Scripting - ImageJ. Dosegljivo: `https://imagej.net/Scripting`. [Dostopano 7. 2. 2019].

[37] C. Spampinato, Y.-H. Chen-Burger, G. Nadarajan, and R. Fisher. Detecting, tracking and counting fish in low quality unconstrained underwater videos. In *Proc. 3rd Int. Conf. on Computer Vision Theory and Applications (VISAPP)*, volume 2, pages 514–519, 2008.

[38] Update Sites - ImageJ. Dosegljivo: `https://imagej.net/Update_Sites`. [Dostopano 15. 2. 2019].

[39] Martin Vodopivec, Rok Mandeljc, Tihomir Makovec, Alenka Malej, and Matej Kristan. Towards automated scyphistoma census in underwater imagery: a useful research and monitoring tool. 2018.

Priloge

Priloga 1

```
1 run("Enhance Contrast...", "Saturated pixels=$d{0,0.9}
    normalize");
2 setAutoThreshold("$o{Otsu,RenyiEntropy,Huang,IsoData,
    Intermodes,MaxEntropy,Mean,Moments,Shanbhag,Triangle,Yen}
    $o{dark,}");
3 run("Convert to Mask");
4 $o{run("Watershed");,}
5 $o{run("Fill Holes");,}
6 run("Analyze Particles...", "size=$1{0,10000}-Infinity
    circularity=$d{0,0.1}-1.00 show=Nothing summarize");
```

Priloga 1: Predloga v jeziku ImageJ Macro z obogateno sintakso (template.ijm)

Priloga 2

```
#@ ImagePlus img
#@OUTPUT ImagePlus output

from java.lang import Double
from ij import ImagePlus, IJ
from ij.measure import ResultsTable, Measurements
from ij.plugin.filter import ParticleAnalyzer

# IMAGE PREPROCESSING
IJ.run(img, "Enhance Contrast...", "Saturated pixels=
    $d{0;0.9}% normalize")
IJ.setAutoThreshold(img, "$o{Otsu;RenyiEntropy;Huang;
    IsoData;Intermodes;MaxEntropy;Mean;Moments;Shanbhag;
    Triangle;Yen} $o{dark; }")
IJ.run(img, "Convert to Mask", "")
$o{IJ.run(img, "Watershed", );}
$o{IJ.run(img, "Fill Holes", );}

# PARTICLE ANALYSIS
measurements = Measurements.RECT + Measurements.MEAN
options = ParticleAnalyzer.SHOW_NONE
# particle size
minSize = $1{0;10000} # minimum particle size
maxSize = Double.POSITIVE_INFINITY # max particle size
# particle circularity
minCirc = $d{0;0.1} # minimum circularity
maxCirc = 1.0 # max circularity

results = ResultsTable()
pa = ParticleAnalyzer(measurements, options, results,
   minSize, maxSize, minCirc, maxCirc)
```

```
28 pa.analyze(img)
29
30 # OUTPUT
31 # output is the input ImagePlus object with the
      ResultsTable saved as property "results"
32 output = img
33 output.setProperty("results", results)
```

Priloga 2: Predloga v Pythonu z obogateno sintakso (template.py)