Štetje objektov na slikah z uporabo genetskega algoritma

Gregor Babnik, Luka Šajn

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
E-pošta: gregor.babnik@student.uni-lj.si, luka.sajn@fri.uni-lj.si

Counting objects in images using a genetic algorithm

The work deals with the automatic counting of objects in images. A genetic algorithm is used as a learning method to find appropriate operations used to process the images. The success of an individual solution is measured as a difference between the number of counted objects and the real object count. ARes algorithm is used to adjust the resolution of input images. The image processing part is implemented using two libraries TensorFlow and OpenCV. The work is tested against various sets of images in different domains.

1 Uvod

Naše delo [1] obravnava implementacijo samodejnega štetja objektov na slikah. Obstaja veliko domen, ki imajo množico slik, te pa vsebujejo veliko količino določenih objektov. Primeri so lahko slike celic, insektov, rastlin, zrn in podobno. Ročno štetje vsega tega je lahko zelo zamudno. Cilj je bil implementirati program, ki se bo z uporabo manjše, učne množice slik iz neke specifične domene naučil oziroma poiskal rešitev za avtomatsko štetje objektov na vseh slikah v tej domeni. Iskanje rešitev na podlagi učne množice je potekalo z uporabo genetskega algoritma. Učna množica slik je bila procesirana z zaporednimi operacijami, te naj bi ustvarjal genetski algoritem in jih nato postopoma izboljševal. Program smo implementirali v programskem jeziku Python, z uporabo programskih knjižnic TensorFlow in OpenCV, ki sta bili uporabljeni za obdelavo in procesiranje slik. Program naj bi poiskal potrebne operacije in prav tako parametre teh operacij samodejno. Uspešnost našega programa smo testirali na slikah iz več domen po naraščajoči kompleksnosti.

2 Implementacija

Celotni program smo napisali v programskem jeziku Python. Za uporabo Pythona smo se odločili, ker ga hkrati podpirata TensorFlow in OpenCV.

Naš program se deli na tri glavne dele:

- prvi del opravlja predprocesiranje, torej pripravo vhodnih slik na procesiranje,
- drugi del programa izvaja genetski algoritem in razvija nove osebke rešitev,
- tretji del, ki sprejme novonastali osebek in po inštrukcijah, vsebovanih v osebku, obdela množico slik in tako ovrednoti podani osebek. Ovrednoteni osebek je nato ali vstavljen v populacijo ali zavržen, če je slabši od vseh, ki so trenutno v populaciji. Tretji del programa je torej evalvacija osebkov, ki se zgodi v treh fazah: procesiranje z operacijami TensorFlow, procesiranje z operacijami OpenCV in nato še štetje.

2.1 Predprocesiranje

Preden program lahko začne obdelavo slik, kot mu narekuje genetski algoritem, je treba te slike primerno pripraviti. Vhodne slike so lahko različnih oblik in velikosti. Prav tako so slike običajno barvne, naš algoritem pa obravnava slike v sivinah.

2.1.1 Pretvorba v sivine

Prvi korak je sprememba vseh slik v sive. Najprej se slike iz RGB-barvnega prostora konvertirajo v barvni prostor CIELAB [6]. Prednost CIELAB je namreč njegova komponenta L, ki se nanaša na lightness oz. osvetljenost. Konvertiranje prek vrednosti osvetlitve prinaša najboljše rezultate pri aplikacijah računalniškega vida [3].

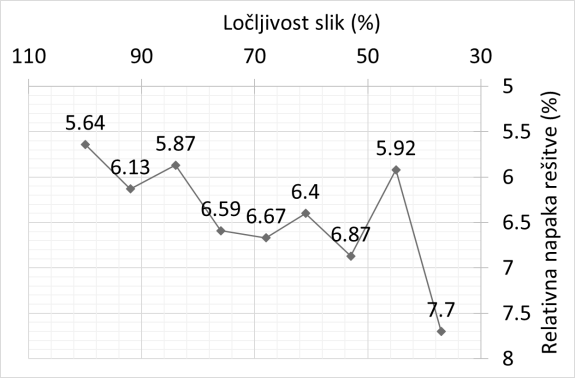
2.1.2 Izenačevanje slik

Ko so slike v sivinah, preidemo na drugi korak predprocesiranja. Zagotoviti je treba, da se uporabi celoten razpon sivin, ki so na voljo, oziroma izenačevanje histograma sivin posamezne slike. Ker običajno izenačevanje ne dosega dovolj dobrih rezultatov, uporabimo namenski algoritem CLAHE [7], ki močno izboljša kontrast in vidne podrobnosti na slikah. V primerjavi z običajnim izenačevanjem, ki deluje globalno, CLAHE izenačuje vrednosti sivin lokalno v obsegu 8×8 slikovnih točk in obenem omeji nastanek šumov, ki se utegnejo pojaviti ob tem.

2.1.3 Nastavitev ločljivosti - ARes

Tretji korak predprocesiranja je nastavitev vseh slik na skupno velikost in izbiranje primerne ločljivosti. Ta postopek opravimo z algoritmom *ARes* [8]. Algoritem *ARes* poišče primerne ločljivosti, ki so nižje od začetne. Pridobljena nižja resolucija nam omogoča hitrejše procesiranje slik. Prav tako nižje ločljivosti potrebujejo manj grafičnega pomnilnika, ki pa je omejen.

Primerna ločljivost se smatra za ločljivost, ki je manjša ali enaka začetni in pri tem ohrani oziroma izboljša kakovosti rešitve. Zmanjšana primerna resolucija lahko pripomore k izboljšanju rešitve (slika 1) tako, da odstrani nepotrebne in moteče podrobnosti slike.



Slika 1: Graf kakovosti rešitve v odvisnosti ločljivosti slik najdenih z algoritmom ARes. Primer štetja celic. [1]

2.2 Implementacija genetskega algoritma pri našem programu

Genetski algoritem deluje na populaciji osebkov, ki jo upravlja in nad njo izvaja selekcijo in križanje, ti pa pripeljeta do novega osebka. Novonastali osebek se nato še mutira.

2.2.1 Opis in zgradba osebka

Osebek vsebuje zapis oziroma navodila, katere operacije izvesti nad tenzorjem slik v našem programu. Osebek je sestavljen iz dveh delov, vsak del pa vsebuje zaporedje določenih operacij. V prvem delu so operacije, ki so implementirane v TensorFlowu. V drugem pa je zaporedje operacij, ki so implementirane v OpenCV.

2.2.2 Kriterijska funkcija

Kriterijska funkcija določa kvantitativni kriterij uspešnosti osebkov. Zmožnost pridobitve dobrih rezultatov pri genetskem algoritmu je zelo odvisna od kakovosti kriterijske funkcije. Kriterijska funkcija usmerja iskanje genetskega algoritma po prostoru in tako močno vpliva na konvergenco populacije. Pri uporabljeni kriterijski funkciji (enačba 1), x_i predstavlja resnično vrednost, y_i pa pridobljeno vrednost. Vsako posamezno odstopanje x_i od y_i se potencira, saj tako kaznujemo osebke, ki so dobri le v povprečju, obenem pa vsebujejo posamezne zelo nadpovprečno slabe rezultate.

$$f(X) = \frac{1}{n}\sum_{i=1}^{n}\frac{100 * (x_i - y_i)}{x_i}^{e} \quad (1)$$

2.2.3 Postopek križanja osebkov

Križanje osebkov deluje v dveh nivojih. Na prvem deluje na nivoju vseh operacij, na drugem pa na nivoju genov posamezne operacije.

Križanje na nivoju vseh operacij v osebku uporablja dve različni strategiji: enostavno enotno križanje in križanje z ujemanjem zaporedja.

Enostavno enotno križanje (uniform crossover) Pri enostavnem enotnem križanju se novi osebek sestavlja tako, da se istoležeče operacije staršev zaporedno križajo druge z drugo.

Križanje z ujemanjem zaporedja Križanje z ujemanjem zaporedja (slika 2) v nasprotju z enostavnim enotnim križanjem upošteva tipe operacij in poskuša ohraniti obstoječe zaporedje, vsebovano v obeh starših, pri novem osebku. Križanje z ujemanjem omogoči prenos možne dobre podrešitve v novi osebek, kjer je podrešitev daljša od posamezne operacije in je pravzaprav zaporedje operacij. Cilj križanja z ujemanjem je hitrejše konvergiranje genetskega algoritma k rešitvi. Algoritem z ujemanjem zaporedja je predstavljen s psevdokodo (algoritem 1). Križanje parametrov operacij se zgodi le pri istoležnih operacijah istega tipa. Novi parameter se določi z naključnim izborom enega izmed parametrov staršev. Verjetnost, kateri parameter bo izbran, določimo z vhodnim parametrom programa *paramter_crossover_weight*. Če istoležni operaciji nista istega tipa, se parametri prepišejo le iz tistega starša, ki je istega tipa kot otrok. Če se tip otroka razlikuje od obeh staršev, se parametri naključno generirajo.

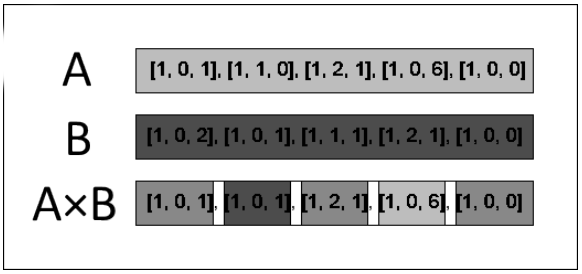
Algoritem 1 Najdi podobno zaporedje

```
indexA ← 0
indexB ← 0
for all operationA ∈ parentA do
  if indexB ≥ length(parentB) then
    indexB ← 0
  end if
  while indexB < length(parentB) do
    operationB ← parentB[indexB]
    if ¬sameType(operationA, operationB) then
      indexB ← indexB + 1
    end if
    append the pair {operationA, operationB} to
    the list commonOPs
  end while
  if operationA ∉ commonOPs then
    opeationB ← parentB[indexA]
    append the pair {operationA, operationB} to
    the list commonOPs
  end if
  indexA ← indexA + 1
end for
```

Zaporedne pare operacij nato križamo med seboj na nivoju genov.

2.2.4 Postopek mutacije osebkov

Mutacija osebkov se zgodi na dveh nivojih. Mutacija na prvi stopnji ima možnost spremembe enega ali več genov vsake operacije, kar spremeni operacijo v neko drugo



Slika 2: Primer križanja z ujemanjem zaporedja.

operacijo. Mutacija na drugi stopnji pa ima možnost spremeniti parametre vsake operacije v osebku. Mutacija parametrov se zgodi neodvisno od mutacije genov. Mutacija parametrov poteka tako, da se jih enostavno nadomesti z drugim naključno generiranim parametrom. Ali se zgodi mutacija, je odvisno od verjetnosti, ki jo določimo kot parameter programa *paramter_muatation_chance*. Ko je novi osebek ustvarjen, se uporabi pri procesiranju slik, pri čemer pridobi tudi oceno.

2.2.5 Opis operacij

V našem programu imamo dva tipa operacij. Prvi tip so operacije, ki so implementirane v TensorFlowu, drugi pa operacije, ki so implementirane v OpenCV. Prvi tip operacij zato imenujemo operacije TensorFlow, drugega pa operacije OpenCV. V program smo vključili pogosto uporabljene operacije procesiranja slik s področja avtomatskega štetja objektov [2]. Operacije Tensorflow (tabela 1) in operacije OpenCV (tabela 2) so podrobneje predstavljene v diplomskem delu [1].

	Koda	Kratek opis
1.	0 0 0	Ničelna operacija
2.	1 0 0	Inverzija
3.	1 0 1	Linearna točkovna op.
4.	1 0 2	Potenčna točkovna op.
5.	1 0 3	Logaritemska točkovna op.
6.	1 0 4	Polinomska točkovna op.
7.	1 0 5	Odstranitev intervala vrednosti
8.	1 0 6	Zamenjava vrednosti z drugo
9.	1 1 0	Linearna konvolucija
10.	1 1 1	Minimalni filter
11.	1 1 2	Maksimalni filter
12.	1 1 3	Povprečni filter
13.	1 1 4	Dvojna linearna konvolucija
14.	1 2 0	Sobelov operator
15.	1 2 1	Gaussova zameglitev
16.	1 2 2	Ostrenje

Tabela 1: Tabela operacij, implementiranih v Tensorflowu

2.3 Procesiranje slik in štetje

Procesiranje slik poteka z uporabo TensorFlowa in OpenCV. Prvi del procesiranja, ki je implementiran s TensorFlowom, predstavlja glavni del obdelave slik. Drugi del, implementiran z OpenCV, pa se osredotoča predvsem

	Koda	Kratek opis
1.	0	Ničelna operacija
2.	1	Inverzija
3.	2	Zameglitev z mediano
4.	3	Dvostranski filter
5.	4	Morfološka operacija
6.	5	Binarizacija
7.	6	Watershed algoritem

Tabela 2: Tabela operacij, implementiranih v OpenCV

na izboljšanje opravljenega dela prvega dela. Na koncu preštejemo skupke slikovnih točk, ki so se prikazali na sprocesiranih slikah in rezultate podamo kriterijski funkciji za končno oceno osebka.

2.3.1 Procesiranje s TensorFlowom

Na začetku izvajanja programa ustvarimo TensorFlow graf operacij. Graf vsebuje implementacije vseh operacij Tensorflow našega programa. Razlog za to je, da graf ustvarjamo samo enkrat na sejo, namesto ob vsakem procesiranju slik. Pri procesiranju pa izvajamo le tisti del grafa, ki izvede želeno operacijo. Pri vsakem ocenjevanju osebka naložimo tenzor neobdelanih slik v spremenljivko TensorFlow, nad katero nato izvršimo podane operacije.

2.3.2 Procesiranje z OpenCV

Ko je procesiranje s TensorFlowom opravljeno, pride na vrsto procesiranje z operacijami OpenCV. To je v nasprotju s TensorFlowom preprostejše, saj ne potrebuje vnaprej definiranega grafa operacij in ustvarjanja posebne seje. V primerjavi s TensorFlowom se slike obdelajo zaporedno.

3 Testiranje in rezultati

Za testne probleme smo izbrali množice slik, ki so vsebovale veliko določenih objektov. Množične slike smo razdelili na učno in testno množico. Program z uporabo genetskega algoritma izdela rešitev na podlagi učne množice in jo na koncu preizkusi na testni množici.

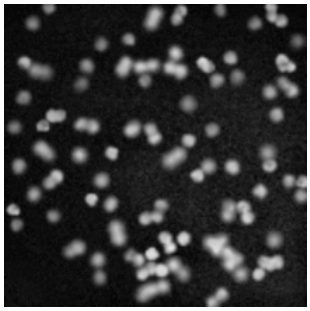
3.1 Test štetja celic

Test vsebuje umetno generirane realistične slike mikroskopskih celic bakterij [4]. Učna množica vsebuje 140 primerov slik, testna množica pa 60. Vhodne slike (slika 3) vsebujejo veliko število objektov celic, ki so neenakomerno razpršene po posamezni sliki in se tudi precej prekrivajo.

Rezultat testa ima poprečno relativno napako 5,88 %, standardna deviacija je 4,57 %, 95 % interval zaupanja je 5,88 % $\pm$ 1,16 %. Rezultat kvarijo predvsem slike, ki vsebujejo veliko prekrivajočih se celic. Problem prekrivajočih se celic program rešuje z uporabo algoritma Watershed, vendar ima tudi ta svoje omejitve in more pomagati, ko je prekrivanje previsoko.

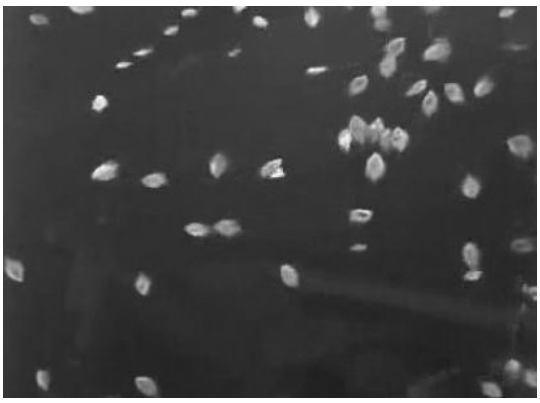
3.2 Test štetja rib

Test vsebuje slike rib [5]. Slike (slika 4) imajo globino in različno orientacijo posamezne ribe. Globina oziroma oddaljenost je sicer majhna, vendar je razlika v ostrini rib.



Slika 3: Celice: primer vhodne slike

Med ribami ni veliko prekrivanja. Ozadje je enobarvno in monotono. Odsev stekla akvarija nekoliko popači sliko. Velikost učne množice je 90 slik, velikost testne pa je 39 slik.



Slika 4: Primer slike rib.

Program ni imel večjih težav pri štetju posameznih rib. Napaka štetja je le pri ribah, ki se prekrivajo ali pa so nagnetene v večje gruče. Rezultat testa ima poprečno relativno napako 4,87 %, standardna deviacija je 3,3 %, 95 % interval zaupanja je 4,87 % $\pm$ 1,04 %.

3.3 Test štetja čebel

Zadnji test štetja vsebuje slike čebel [5]. Učna množica vsebuje 93 slik, testna množica pa 25.

Slike (slika 5) vsebujejo roj čebel na neenakomerni podlagi – travi in travnatih rastlinah. Na slikah skorajda ni prekrivanja med čebelami. Čebele so v zraku in tako različno orientirane, kar pomeni, da se njihovi obrisi razlikujejo po velikosti in obliki.

Povprečna relativna napaka štetja čebel je 8,29 % s standardno deviacijo 6,39 %. 95 % interval zaupanja je 8,29 % $\pm$ 2,50 %. To je bil najtežji izmed testov, ki smo jih dali v program. Del težavnosti izvira tudi iz tega, da imajo pri teh slikah barve pomembno vlogo - rumenkaste čebele med zelenkastim rastlinjem. Pri pretvorbi iz barvnih slik v sivini se je precej podatkov izgubilo.

4 Zaključek

Implementirali program, ki se nauči šteti objekte na slikah. Za učenje potrebuje množico slik, na kateri poteka

Slika 5: Primer slike čebel.

učenje. Učenje poteka tako, da program z uporabo genetskega algoritma išče uspešno zaporedje operacij, ki obdelajo slike. Zaporedje operacij je uspešno takrat, ko je razlika med preštetimi in dejanskimi objekti na slikah zmanjšana do zadovoljive mere. Ta razlika služi tudi kot vhodna spremenljivka v kriterijski funkciji. Pri določanju primernih ločljivosti slik smo si pomagali z algoritmom ARes. Za izvedbo operacij nad slikami smo uporabili programski knjižnici TensorFlow in OpenCV. Uporaba TensorFlowa nam je pospešila izvajanje programa s procesiranjem podatkov (obdelavo slik) na GPE. Delovanje programa smo testirali na različnih domenah slik, z različnimi težavnostmi za učenje. Prednost programa je samodejno iskanje rešitev.

Literatura

[1] Gregor Babnik. Štetje objektov na slikah z uporabo genetskega algoritma. Diplomska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2020.

[2] Jayme Garcia Arnal Barbedo. A review on methods for automatic counting of objects in digital images. *IEEE Latin America Transactions*, 10(5):2112–2124, 2012.

[3] Christopher Kanan and Garrison W Cottrell. Color-to-grayscale: does the method matter in image recognition? *PloS one*, 7(1):e29740, 2012.

[4] V. Lempitsky and A. Zisserman. Vgg cell dataset from learning to count objects in images. 2010.

[5] Zheng Ma, Lei Yu, and Antoni B Chan. Small instance detection by integer programming on object density maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3689–3697, 2015.

[6] K McLaren. Xiii—the development of the cie 1976 (l* a* b*) uniform colour space and colour-difference formula. *Journal of the Society of Dyers and Colourists*, 92(9):338–341, 1976.

[7] Stephen M Pizer, E Philip Amburn, John D Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39(3):355–368, 1987.

[8] Luka Šajn and Igor Kononenko. Multiresolution image parametrization for improving texture classification. *EURASIP Journal on Advances in Signal Processing*, 2008:137, 2008.