Zaznava paradižnikov z računalniškim vidom

Janez Perš, Marija Ivanovska

15. januar 2024

Povzetek

V okviru te vaje študenti rešujejo problem avtomatizacije nadzora vzgoje paradižnikov v rastlinjakih. Ožja tema naloge je implementacija segmentacijske nevronske mreže z namenom zaznavanja paradižnikov na RGB slikah. Pri tem naj bi si pomagali s kodo, ki so jo razvili pri prejšnji vaji.

1 Podatkovna baza

Podatkovno bazo sestavljajo RGB slike iz baze LaboroTomato [1], zajete v rastlinjaku, kjer se gojijo paradižniki. Bazo, s katero boste delali, prevzamete na povezavi:

laboro_tomato.zip

Drevesna struktura direktorija *laboro_tomato* je naslednja:

- laboro_tomato
  - train
    - gt
    - images
    - masks
  - test
    - gt
    - images
    - masks
  - annotations.json

Vsi učni vzorci in pripadajoče oznake se torej nahajajo v mapi *train*, medtem ko so testni vzorci shranjeni v mapi *test*. Oba direktorija vsebujeta:

- mapo *images*, ki jo sestavljajo RGB slike v `.png` formatu. Za razliko od originalne baze *LaboroTomato* so te slike pomanjšane in včasih tudi obrnjene tako, da so vse enake velikosti in sicer 512×384 pikslov. Na slikah so predstavljeni paradižniki različnih vrst in različnih stopenj zrelosti. Baza ponuja 643 učnih slik in 161 testnih.
- mapo *masks*, ki jo sestavljajo črnobele `.png` slike, kjer so paradižniki označeni z belo barvo, okolica pa s črno. Tudi te slike so velikosti 512×384 pikslov.

- mapo *gt* (angl. ground truth), kjer za vsako RGB sliko najdete prikaz pripadajoče segmentacijske maske in prikaz očrtanih pravokotnikov.

Na Sliki 1 so prikazane izbrane slike iz baze in pripadajoče oznake.

Slika 1: Primeri slik iz podatkovne baze LaboroTomato [1] s pripadajočimi oznakami.

Datoteka *annotations.json* vsebuje koordinate točk potrebnih za izris očrtanih pravokotnikov ter oznako stopnje zrelosti vsakega paradižnika. Datoteka je skupna vsem vzorcem (učnim in testnim). V programskem jeziku `Python` lahko preberete vsebino te datoteke z modulom `json`:

```
import json

with open("annotations.json", "r") as json_file:
    data=json.load(json_file)
json_file.close()
```

Spremenljivka `data` je v tem primeru Pythonov slovar (angl. dictionary), ki za ključe (angl. keys) uporablja imena slik. Oznake slike z imenom *IMG_1022.png* na primer preberemo z `data['IMG_1022.png']`, pri čemer se izpiše:

```
[[[231.0528, 158.31552, 364.32383999999996, 298.14272], 1],
[[167.25119999999998, 257.8944, 278.1580799999999, 361.05216], 3],
[[153.2352, 162.78528, 249.53855999999996, 267.43296], 2],
[[104.639999999, 242.391039999, 185.43743999, 325.8726399], 3]]
```

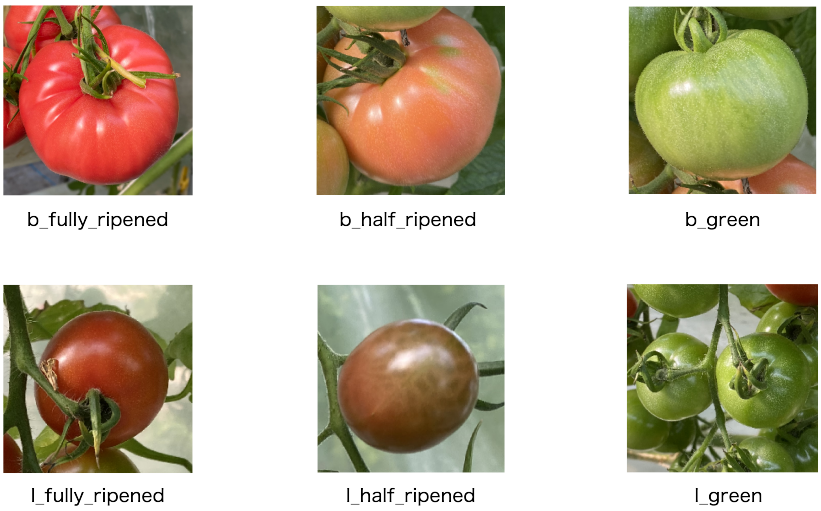
Na tej sliki so torej štirje paradižniki. Vsak paradižnik je označen z `[[x_1, y_1, x_2, y_2], c]`, kjer `(x_1, y_1)` predstavljata koordinati zgornjega levega ogljišča očrtanega pravokotnika, `(x_2, y_2)` predstavljata koordinati spodnjega desnega ogljišča, `c` pa je oznaka stopnje zrelosti ((Slika 2)).

Slika 2: Prikaz očrtanih pravokotnikov slike *IMG_1022.png* in oznak podanih koordinat enega pravokotnika.

Oznaka `c` lahko zavzame naslednje vrednosti:

- 1 (*b_fully_ripened*): zrel paradižnik normalne velikosti,
- 2 (*b_half_ripened*): delno zrel paradižnik normalne velikosti,
- 3 (*b_green*): zelen paradižnik normalne velikosti,
- 4 (*l_fully_ripened*): zrel češnjev paradižnik,
- 5 (*l_half_ripened*): delno zrel češnjev paradižnik,
- 6 (*l_green*): zelen češnjev paradižnik.

Različne stopnje zrelosti so prikazane na Sliki 3



Slika 3: Prikaz različnih stopenj zrelosti paradižnikov in njihove oznake: 1 = *b_fully_ripened*, 2 = *b_half_ripened*, 3 = *b_green*, 4 = *l_fully_ripened*, 5 = *l_half_ripened*, 6 = *l_green*

2 Segmentacija paradižnikov z nevronoskim omrežjem U-Net

Za segmentacijo paradižnikov iz zbirke LaboroTomato uporabite omrežje U-Net [2]. Članek o tem omrežju je na voljo na povezavi https://arxiv.org/pdf/1505.04597.pdf. U-Net je konvolucijska nevronska mreža, ki je bila prvotno predlagana za semantično segmentacijo mikroskopskih slik. Sestavljata jo kodirnik in dekodirnik, ki skupaj oblikujeta črko U, od koder prihaja tudi ime omrežja (Slika 4):

- kodirnik (angl. encoder), ki podobno kot klasična razvrščevalna konvolucijska omrežja lušči značilke, karakteristične za bazo, ki jo analiziramo. Kodirnik vsebuje konvolucijske sloje in sloje, ki v določenih korakih izvajajo podvzorčenje izluščenih značilk z operacijo združevanja z maksimizacijo (angl. maxpooling),
- dekodirnik (angl. decoder) iz izluščenih značilk ustvari segmentacijsko matriko (masko), v kateri so različna semantična področja označena z različnimi oznakami. Poleg standardne

konvolucije dekodirnik v določenih korakih izvaja tudi tako imenovano transponirano konvolucijo, ki poveča velikost matrik (značilk) izračunanih s filtri prejšnjega sloja.

Segmentacijsko omrežje U-Net implementirajte z uporabo programskega jezika `Python` in knjižnjice `PyTorch` [3].

2.1 Bralnik podatkov

Za učenje omrežja U-Net boste potrebovali tako RGB slike, kot tudi pripadajoče maske, kjer so paradižniki označeni z belo barvo, okolica pa s črno. Bralnik podatkov (angl. dataloader) mora zato sproti brati oboje. Za evalvacijo naučenega omrežja pa je koristen tudi podatek o imenu vzorca (*path*), ki ga privzeti bralnik ne vrne samodejno. Iz teh razlogov boste definicijo klase `DatasetFolder` prilagodili svojim potrebam in sicer tako, da si k sebi prenesete izvirno kodo omenjene klase in ustrezno preuredite definicijo funkcije `__getitem__`. Se pravi kodo, ki jo najdete na povezavi:

https://pytorch.org/vision/main/_modules/torchvision/datasets/folder.html#DatasetFolder

shranite lokalno (na primer kot datoteko *tomato_dataset.py*) in preden začnete z prilagajanjem te skripte, v 7. vrstici popravite `from .vision import VisionDataset` v `from torchvision.datasets import VisionDataset`. V skripti, kjer definirate instance klase `DatasetFolder` pa namesto `from torchvision.datasets import DatasetFolder`, `DatasetFolder` uvozite iz skripte, ki jo boste nadgradili sami (na primer `from tomato_dataset import DatasetFolder`).

Na vajah ste prebrane RGB slike transformirali z uporabo množice transformacijskih funkcij `tf`transforms.Compose([...])=, ki je spremenila velikost slik, izvedla normalizacijo podatkov in matrike pretvorila v PyTorch tenzorje. Pri učenju segmentacijske mreže boste mogli na podoben način pretvoriti vse prebrane matrike (RGB slike in maske) v tenzorje, vendar normalizicije ne smete izvajati na maskah. Tranformacije, ki se nanašajo samo na RGB slike zato definirajte ločeno.

Pri globokem učenju je za preprečevanje preprileganja omrežja (angl. overfitting) pomembno, da je učna množica dovolj velika in raznovrstna. Kadar nimamo dovolj učnih vzorcev, se pogosto poslužimo tehnike bogatenja podatkov (angl. data augmentation). Pri bogatenju podatkov umetno ustvarimo nove učne vzorce z uporabo obstoječih. Obstoječe vzorce lahko na primer prezrcalimo, rotiramo, spremenimo kontrast na slikah itd. V ta namen lahko uporabimo PyTorchov modul `torchvision` ali kakšno drugo knjižnjico, kot je na primer *Albumentations*:

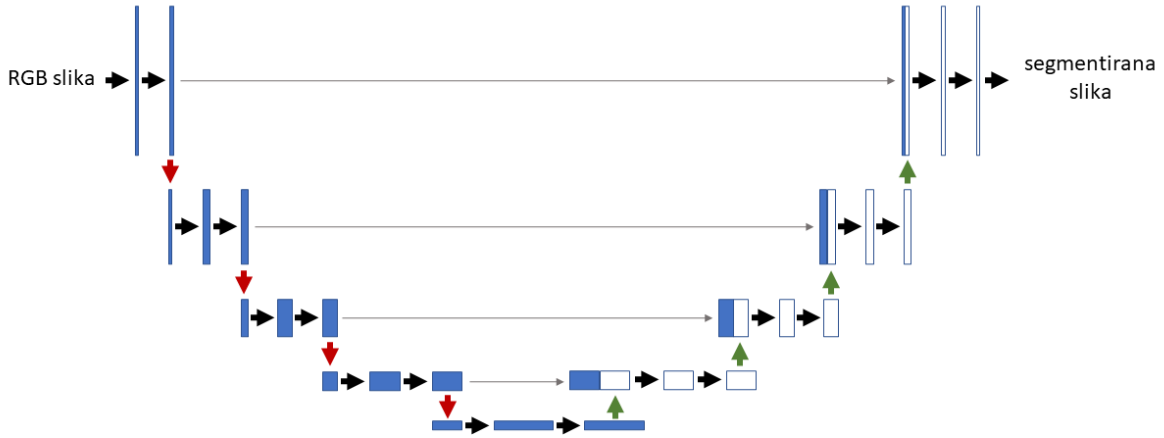
https://albumentations.ai/docs/getting_started/image_augmentation/

Iz te knjižnjice si izberite funkcije, za implementacijo augmentacije učnih podatkov, ki je smiselna za problem, ki ga rešujete (segmentacija paradižnikov). Pri tem ne pozabite na augmentacijo mask, tam kjer je to smiselno. Če boste na primer originalno sliko prezrcalili, morate prezrcaliti tudi masko. Če boste v okviru augmentacije spreminjali kontrast na originalnih slikah, te transformacije ne uporabljajte na maskah. Pri razvoju kode si lahko pomagate z informacijami na povezavi:

https://albumentations.ai/docs/getting_started/mask_augmentation/

2.2 Arhitektura omrežja

Arhitekturo segmentacijskega omrežja U-Net boste definirali sami, z uporabo knjižnice `PyTorch`, ki ste jo spoznali na vajah. Pri tem si pomagajte s shemo in tabelo iz Slike 4. Za implentacijo konvolucijskih slojev *Conv3-x* uporabite klaso `torch.nn.Conv2d()` tako, da se konvolucija izvaja s filtri velikosti 3×3, pri čemer je število filtrov oz. število izhodnih kanalov enako x. Na podoben način definirajte tudi transponirano konvolucijo (`torch.nn.ConvTranspose2d()`), tokrat s filtri velikosti 2×2. Za vsakim konvolucijskim slojem (razen za zadnjim *Conv1-1*, ki ustvari izhodno masko) s klaso `torch.nn.ReLU()` implementirajte nelinearno preslikavo z aktivacijsko funkcijo ReLU (angl. Rectified Linear Unit). Za implementacijo združevanja značilk z maksimizacijo uporabite klaso `torch.nn.MaxPool2d()`. Bodite pozorni na način izvajanja vseh teh operacij. Privzeti parametri namreč premikajo filtre s korakom (angl. stride) $s = 1$ in brez dodajanja ničel (angl. padding) ob robovih $p = 0$. V vašem primeru, vrednosti teh parametrov prilagodite v skladu s tabelo na Sliki 4. Oznaka *p1* vam pove, da je vrednost parametra `padding` enaka 1, oznaka *s2* pa vam pove, da je vrednost parametra `stride` enaka 2. Tam kjer ni teh oznak, so vrednosti parametrov enake privzetim.



Encoder
RGB image 512x384x3
Conv3-64/p1 Conv3-64/p1
Maxpool2/s2 Conv3-128/p1 Conv3-128/p1
Maxpool2/s2 Conv3-256/p1 Conv3-256/p1
Maxpool2/s2 Conv3-512/p1 Conv3-512/p1
Maxpools/s2 Conv3-1024/p1 Conv3-1024/p1

Decoder
ConvT2-512/s2 Conv3-512/p1 Conv3-512/p1
ConvT2-256/s2 Conv3-256/p1 Conv3-256/p1
ConvT2-128/s2 Conv3-128/p1 Conv3-128/p1
ConvT2-64/s2 Conv3-64/p1 Conv3-64/p1
Conv1-1
Mask 512x384x1

Slika 4: Arhitektura konvolucijske nevronske mreže U-Net za segmentacijo paradižnikov.

2.3 Učenje omrežja

Za optimizacijo parametrov uporabite metodo Adam `torch.optim.Adam()`. Stopnjo učenja (angl. learning rate) lahko za začetek nastavite na 0.0001. Ker je vaša segmentacija binarna, za kriterijsko funkcijo izberite binarno navkrižno entropijo (angl. binary cross entropy), ki jo implementirate s klaso `torch.nn.BCEWithLogitsLoss()`. Ta PyTorchova funkcija bo najprej transformirala izhodno matriko omrežja (napovedano masko) s Sigmoidno preslikavo, nato bo primerjala ta rezultat z zlatim standardom (angl. ground truth) oz. z masko iz baze. Napaka odstopanja se izračuna po pikslih in sicer po formuli:

$$\frac{1}{N}\sum_{i=1}^{N} -(y_i * log(p_i) + (1 - y_i) * log(1 - p_i))$$

kjer je y_i pravilna oznaka piksla z indeksom i, prebrana iz maske (vrednost je enaka 1, ko je to piksel nekega paradižnika ali 0, ko je to piksel okolice), p_i pa je verjetnost, da ta piksel pripada objektu, ki ga želimo segmentirati (v tem primeru paradižniku). Naloga optimizatorja je, da z vzvratnim razširjanjem (angl. backpropagation) in posodobitvijo parametrov filtrov, zmanjša napako odstopanja izračunane maske od prave. Zaradi kompleksnosti omrežja in variance v učni množici, bo omrežje potrebovalo nekaj časa, da skonvergira. Omrežje učite najmanj 100 epoh. V eni epohi obdelamo vse učne vzorce iz baze. Posodobitev parametrov omrežja pa izvajamo sproti, po vsakem posredovanem paketu vzorcev. Velikost enega paketa oz. število vzorcev v enem paketu določimo tako, da je to čim večje (kolikor nam spomin na grafični kartici dopušča). Vse operacije, ki so povezane z omrežjem (izračun konvolucij in vzvratno razširjanje) ponavadi izvajamo na grafični kartici, zaradi pohitritve procesa učenja. Iz istega razloga inferenco (angl. inference) oz. izračun značilk z naučenimi filtri v testni fazi ravno tako izvajamo na grafični kartici.

2.4 Evaluacija omrežja

Med učenjem omrežij si ponavadi želimo sproti preverjati koliko dobro se omrežje uči in ali se kdaj začne preprilegati vzorcem iz učne množice. V ta namen že med učenjem izvajamo periodično evaluacijo omrežja na vzorcih iz validacijske baze. Ko zaključimo z učenjem, omrežje evaluiramo še na vzorcih iz testne baze. Baza LaboroTomato ne vsebuje ločene validacijske množice, zato boste evaluacijo izvajali kar na testni bazi.

Pri validaciji segmentacijskega omrežja izračunamo napako odstopanja napovedanih mask za vzorce iz testne množice. Pri tem pa je ključno, da v ta namen uporabljamo ustrezno metriko. Predstavljajte si, da omrežje ne najde paradižnikov (napovedana maska je črna), ki skupaj zavzamejo manj kot 10% površine slike (Slika 5):

Če za ta primer izračunamo odstotek pravilno razvrščenih pikslov, bo ta čez 90%, ker so vsi piksli iz okolice pravilno razvščeni. Čeprav je ta odstotek visok, omrežje očitno ne deluje tako, kot si želimo.

Pri evaluaciji segmentacijskih omrežij ponavadi uporabljamo metriko IoU (angl. Intersection of Union), znano tudi kot Jaccardov indeks (angl. Jaccard Index). Ta metrika izračuna razmerje med presekom napovedanega območja in območja iz zlatega standarda in njihovo unijo:

$$IoU = \frac{\mathbf{G} \cap \mathbf{P}}{\mathbf{G} \cup \mathbf{P}} = \frac{TP}{TP + FP + FN}$$

Slika 5: Primer slike, kjer paradižniki, ki jih želimo segmentirati zavzamejo manj kot 10% skupne površine slike. Omrežje, ki ne najde teh paradižnikov bo napovedalo črno masko. Ker so piksli okolice pravilno detektirani, bo v tem primeru odstotek pravilno razvrščenih pikslov presegel 90%, čeprav omrežje ne deluje tako, kot si želimo.

kjer je z **G** označeno območje iz zlatega standarda, s **P** pa napovedano območje. TP je število pravilno razvrščenih pikslov, FP je število pikslov, ki so bili napovedani kot paradižnikovi, so pa dejansko iz okolice in FN so paradižnikovi piksli, ki so bili napačno pripisani okolici. Ko boste testirali naučeno omrežje U-Net, lahko pri izračunu metrike IoU uporabite knjižnico, kjer je metoda že implementirana.

Pomembno: Ne pozabite na Sigmoidno transformacijo, ki se v učni fazi izvede v okviru funkcije, ki izračuna napako odstopanja. V testni fazi morate izhodne matrike omrežja (napovedane maske) najprej preslikati s Sigmoidno funkcijo. V ta namen lahko uporabite `torch.sigmoid()`.

Pri testiranju dovolj dobro naučenega omrežja v obliki slik shranite ustvarjene maske, da boste lahko naredili še kvalitativno analizo segmentiranih vzorcev. Optimizirane parametre omrežja, ki da najmanjšo napako vedno shranimo za kasnejšo uporabo na novih vzorcih. V PyTorchu to dosežemo s funkcijo `torch.save()`. Primer kode, ki shrani uteži omrežja, definirano z imenom `unet`:

```
from torch import save
import os

.....

if not os.path.exists('./checkpoints'):
    os.makedirs('./checkpoints')
save({'model_state_dict': unet.state_dict()}, './checkpoints/checkpoint.pt')
```

Predhodno shranjene uteži pa naložite z uporabo funkcije `torch.load()` (primer na povezavi: https://pytorch.org/tutorials/beginner/saving_loading_models.html).

Literatura

[1] Laboro.AI. LaboroTomato. `https://github.com/laboroai/LaboroTomato`. Accessed: 2023-01-06.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, pages 234–241, Cham, 2015. Springer International Publishing.

[3] PyTorch documentation. `https://pytorch.org/docs/stable/index.html`.