VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020

Bc. Matej Boszorád

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SEGMENTACE OBRAZOVÝCH DAT POMOCÍ GRAFOVÝCH NEURONOVÝCH SÍTÍ
IMAGE SEGMENTATION USING GRAPH NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Matej Boszorád**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Vojtěch Myška**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Matej Boszorád ***ID:*** 186035
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Segmentace obrazových dat pomocí grafových neuronových sítí

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte existující techniky segmentace obrazových dat. Seznamte se s principem grafových neuronových sítí. Získané poznatky využijte při návrhu vlastní grafové neuronové sítě za účelem segmentace obrazových dat. Funkčnost prezentujte pomocí získaných výsledků.

DOPORUČENÁ LITERATURA:

[1] LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. p. 3431-3440.

[2] DeepMind, “Graphnets,” https://github.com/deepmind/graphnets,2018.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Vojtěch Myška

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Táto diplomová práca opisuje a implementuje návrh grafovej neurónovej siete použitej na 2D segmentáciu nervovej štruktúry. Prvá kapitola práce v krátkosti uvádza problém segmentácie. Nachádza sa v nej rozdelenie segmentačných techník podľa princípov metód, ktoré používajú. Každý typ techniky obsahuje podstatu tejto kategórie a taktiež popis jedného predstaviteľa. Druhá kapitola diplomovej práce sa zaoberá grafovými neurónovými sieťami (v skratke GNN). Tu práca rozdeľuje grafové neurónové siete celkovo a bližšie popisuje rekurentné grafové neurónové siete (v skratke RGNN) a grafové autoenkodéry, ktoré je možné použiť na segmentáciu obrazu. Konkrétne riešenie segmentácie obrazu je založené na metóde prenášania správ v RGNN, ktorou je možné nahradiť konvolučné masky v konvolučných neurónových sieťach. RGNN taktiež používajú jednoduchšiu topológiu viacvrstvového perceptronu. Druhým typom opísaných grafových neurónových sietí v práci sú grafové autoenkodéry, ktoré používajú rôzne metódy pre lepšie zakódovanie vrcholov grafu do euklidovského priestoru. Posledná časť diplomovej prace sa venuje rozboru problému, návrhu jeho konkrétneho riešenia a vyhodnotenie výsledkov. Úlohou praktickej časti práce je implementácia GNN na segmentáciu obrazových dát. Výhodou použitia neurónových sietí je možnosť riešiť rozličné typy segmentácie zmenou trénovacích dát. Na implementáciu GNN bola použitá RGNN s prenášaním správ a autoenkodér node2vec. Trénovanie RGNN bolo uskutočnené na grafických kartách poskytnutých školou a službou Google Colaboratory. Učenie RGNN s použitím node2vec bolo pamäťovo veľmi náročné, a preto bolo nutné trénovanie na procesore s operačnou pamäťou väčšou ako 12GB. V rámci optimalizácie RGNN bolo otestované učenie použitím rôznych stratových funkcií, zmenou topológie a učiacich parametrov. Pre použitie node2vec bola vytvorená metóda stromovej štruktúry na zlepšenie segmentácie, avšak výsledky nepotvrdili zlepšenie pre malý počet iterácií. Najlepšie výsledky praktickej implementácie boli vyhodnotené pomocou porovnania otestovaných dát s konvolučnou neurónovu sieťou U-Net. Je možné skonštatovať porovnateľné výsledky oproti sieti U-Net, avšak pre porovnanie týchto sietí je potrebné ďalšie testovanie. Výsledkom práce je použitie RGNN ako moderné riešenie problému segmentácie obrazu a poskytnutie základu pre ďalší výskum.

KĽÚČOVÉ SLOVÁ

Grafová neurónová sieť, segmentácia obrazu, prenášanie správ, node2vec, GNN, rekurentná grafová neurónová sieť, RGNN

ABSTRACT

This diploma thesis describes and implements the design of a graph neural network used for 2D segmentation of neural structure. The first chapter of the thesis briefly introduces the problem of segmentation. In this chapter, segmentation techniques are divided according to the principles of the methods they use. Each type of technique contains the essence of this category as well as a description of one representative. The second chapter of the diploma thesis explains graph neural networks (GNN for short). Here, the thesis divides graph neural networks in general and describes recurrent graph neural networks (RGNN for short) and graph autoencoders, that can be used for image segmentation, in more detail. The specific image segmentation solution is based on the message passing method in RGNN, which can replace convolution masks in convolutional neural networks. RGNN also provides a simpler multilayer perceptron topology. The second type of graph neural networks characterised in the thesis are graph autoencoders, which use various methods for better encoding of graph vertices into Euclidean space. The last part of the diploma thesis deals with the analysis of the problem, the proposal of its specific solution and the evaluation of results. The purpose of the practical part of the work was the implementation of GNN for image data segmentation. The advantage of using neural networks is the ability to solve different types of segmentation by changing training data. RGNN with messaging passing and node2vec were used as implementation GNN for segmentation problem. RGNN training was performed on graphics cards provided by the school and Google Colaboratory. Learning RGNN using node2vec was very memory intensive and therefore it was necessary to train on a processor with an operating memory larger than 12GB. As part of the RGNN optimization, learning was tested using various loss functions, changing topology and learning parameters. A tree structure method was developed to use node2vec to improve segmentation, but the results did not confirm an improvement for a small number of iterations. The best outcomes of the practical implementation were evaluated by comparing the tested data with the convolutional neural network U-Net. It is possible to state comparable results to the U-Net network, but further testing is needed to compare these neural networks. The result of the thesis is the use of RGNN as a modern solution to the problem of image segmentation and providing a foundation for further research.

KEYWORDS

Graph neural network, image segmentation, message passing, node2vec, GNN, recurrent graph neural network, RGNN

BOSZORÁD, Matej. *Segmentácia obrazových dat pomocou grafových neuronových sietí*. Brno, 2020, 60 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Vojtěch Myška,

VYHLÁSENIE

Vyhlasujem, že som svoju diplomovú prácu na tému „Segmentácia obrazových dat pomocou grafových neuronových sietí“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Vojtěchovi Myškovi, za odborné vedenie, konzultácie, trpezlivosť, výpočetné prostriedky a podnetné návrhy k práci.

Brno ....

podpis autora

Obsah

Zoznam obrázkov

Úvod

Táto práca sa zaoberá problémom segmentácie obrazu za použtia grafových neurónových sietí. Segmentácia obrazu má široké uplatnenie napríklad pri kontrole kvality v automatizácii, alebo pri autentifikácii pomocou odtlačkov prstov. Diplomová práca sa zameriava na využitie segmentácie v medicíne.

Prínosom práce je využitie potenciálu prenášania správ v rekurentných grafových neurónových sieťach pri segmentácii nervových štruktúr. Rieši sa v nej tiež rozdelenie grafových neurónových sietí a popis techník v grafových neurónových sieťach, ktoré boli použité v praktickej časti. Okrem toho sa v tejto práci nachádza rešerš techník segmentácie používaných v minulosti a v modernej dobe.

Práca je rozdelená na tri kapitoly.

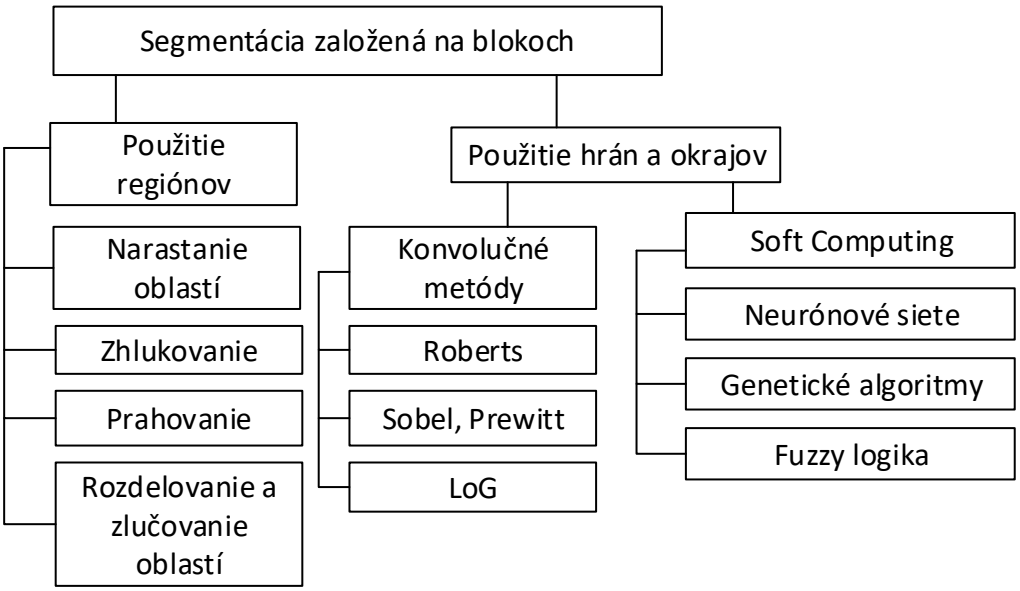
Kapitola Segmentácia obrazu obsahuje definíciu problému a rozdelenie možných prístupov k riešeniu segmentácie. Pri každom type prístupu k problému sa nachádzajú jeho špecifické vlastnosti a konkrétny príklad. Kapitola Grafové neurónové siete sa zaoberá popisom vstupnej časti, neurónovej siete a výstupu do výsledného grafu. V rámci pochopenia praktickej časti práce kapitola obsahuje popis metód v rekurentnej grafovej neurónovej sieti a taktiež metód používaných v grafových autoenkodéroch.

Praktická časť pozostáva z rozboru problému a návrhu riešenia s použitím grafových neurónových sietí. Táto časť tiež popisuje U-net ako moderné riešenie pre segmentáciu obrazu. Okrem použitia základného návrhu riešenia táto práca použila aj možnosti predspracovania obrazu alebo využitia grafovej štruktúry pre zlepšenie výsledkov segmentácie. V rámci praktickej časti sú výsledky spracované do grafov, tabuliek a segmentovaných obrazov.

Na záver boli zhodnotené výsledky praktickej časti práce, ktoré boli porovnané s výsledkami použitia neurónovej siete U-net.

1 Segmentácia obrazu

Segmentácia obrazu znamená proces rozdeľovania obrazu do logických častí pomocou vlastností obrazu [36] . Tento proces sa používa pri spracovaní obrazu, konkrétne pri klasifikácií obrazu. Pre správnu klasifikáciu je potrebné rozdeliť obraz na časti a následne podla vlastností objektu ohodnotiť, o aký objekt sa jedná. V tejto sekcii je popísaná segmentácia založená na blokoch podľa [45] , nakoľko vrstvová segmentácia prechádza do klasifikačnej časti.



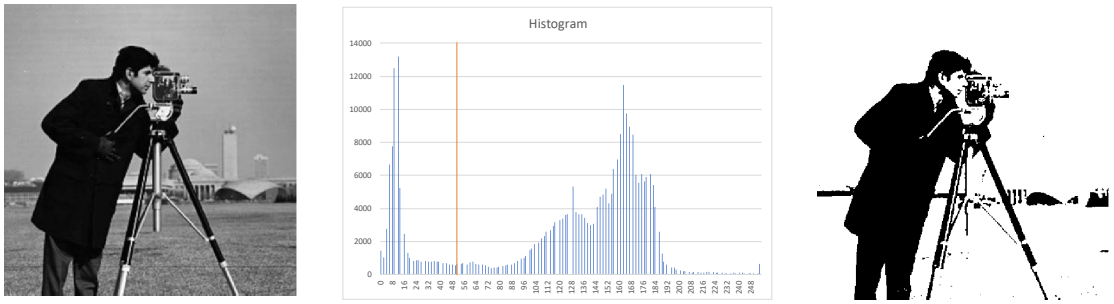
Obr. 1.1: Rozdelenie segmentačných metód

1.1 Použitie regiónov

Segmentácia obrazu pomocou použitia regiónov funguje na základe podobnosti pixelov v určitom regióne.

1.1.1 Prahovanie

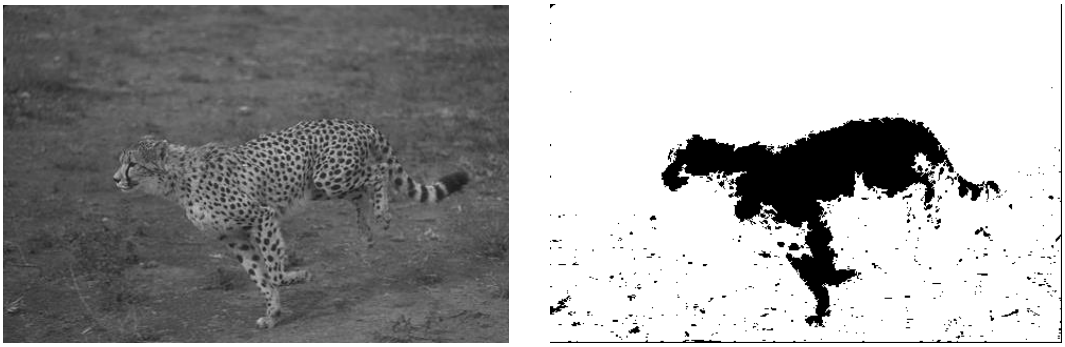
Prahovanie je metóda segmentácie obrazu, v ktorej sa používa histogram zvolenej vlastnosti. Tento histogram sa rozdelí na určitý počet segmentov rozdelených užívateľom alebo pomocou algoritmu. Populárne optimalizačné algoritmy pre adaptívne prahovanie sú napríklad Otsuov algoritmus [40] alebo Kril-Herdov optimalizačný algoritmus s viacvrstvovým prahovaním [4] .



Obr. 1.2: Segmentácia obrazu pomocou prahovania

1.1.2 Narastanie oblastí

Metóda, v ktorej sa prechádza okolím začiatočného bodu a podľa rozdielu od vybraného začiatočného bodu pripája dané body k oblasti. Ak je rozdiel vyšší, stáva sa táto časť novou oblasťou. Problém tejto metódy spočíva pri zašumenom obraze alebo zmene jasu. Tieto problémy sú však riešiteľné [3, 20] . Praktický príklad narastania oblastí je možné vidieť na obr 1.3 [39] .

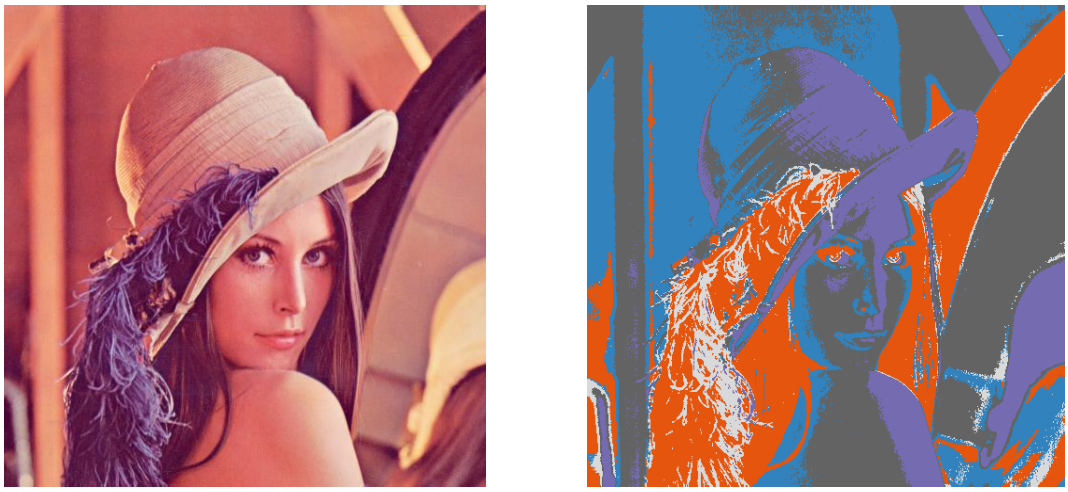


Obr. 1.3: Segmentácia obrazu pomocou Narastania oblastí: Originálny obraz (a), obraz po segmentácii (b)

1.1.3 Zhlukovanie

Zhlukovanie je algoritmus, ktorý zhlukuje najbližšie vlastnosti k segmentom. Pre pochopenie je ďalej popísané zhlukovanie ku k-priemerom (anglicky "k-mean clustering"). Tento algoritmus najprv vyberie k náhodných bodov v obraze. Následne prechádza všetkými pixelmi obrazu a pripája ich na základe vzdialenosti (napríklad euklidovej) k vybraným náhodným bodom podľa určenej vlastnosti (napríklad farby). V ďalšom kroku prechádza algoritmus vybranými bodmi a mení ich k priemeru k nim priradených bodov. Tento proces sa opakuje až pokiaľ nenastane zmena

v priradeniach pixelov ku k-priemerom. Táto metóda oproti iným zhlukovacím metódam konverguje rýchlejšie, avšak nezaručuje kontinuálne oblasti [45] . Taktiež nepoužíva priestorovú informáciu, čím metóda trpí pri zašumenom obraze.



a) b)

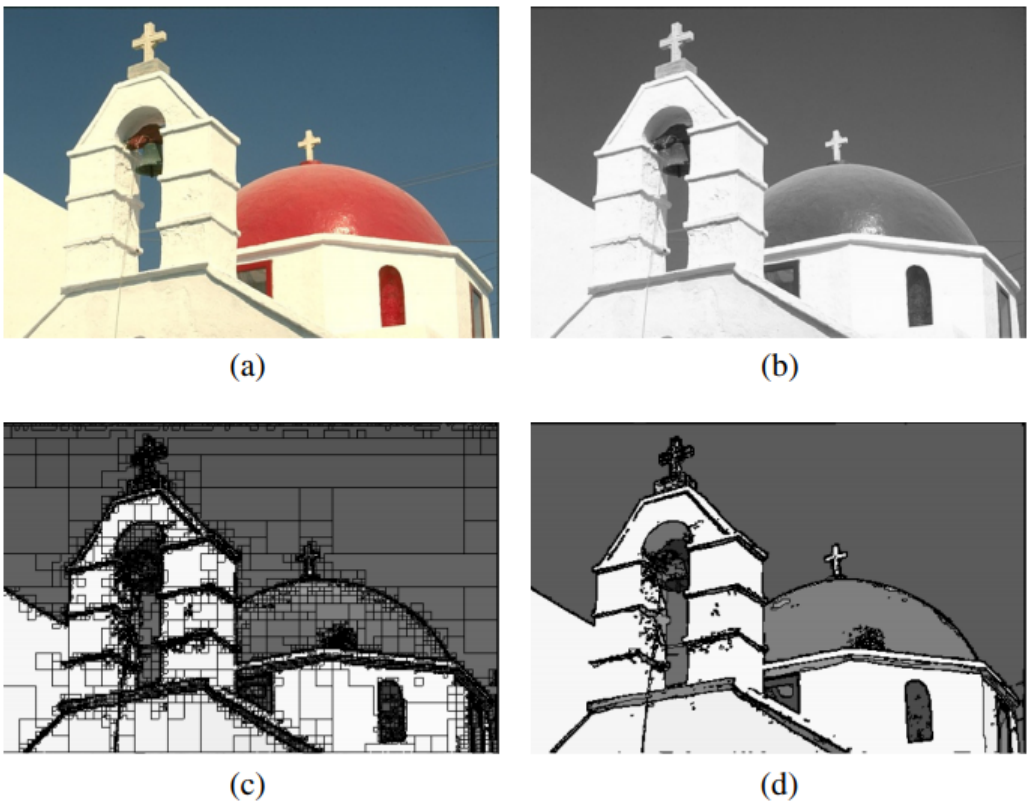
Obr. 1.4: Segmentácia obrazu pomocou k-priemerov: Originálny obrázok (a), obrázok po segmentácii (b)

1.1.4 Rozdeľovanie a zlučovanie oblastí

Spôsob segmentácie je v tomto prípade rozdelený do dvoch častí. V prvej sa rozdeľuje obraz do x rovnakých častí, pričom sa tento proces opakuje pre každú časť, kým nie je splnená podmienka prahu (napríklad rozdiel najmenšej a najväčšej jasovej zložky) [16] . Následne sú tieto časti zlučované tak, aby podmienka prahu bola splnená. Problém tejto metódy pri rozdeľovaní na rovnaké časti nastáva pri okrajoch, kde vznikajú bloky. Tento problém je možné riešiť napríklad použitím narastania oblastí [28] .

1.2 Použitie hrán a okrajov

V tomto type metódy segmentácie sa vyhľadávajú diskontinuity v obraze, čiže hrany a rohy, pomocou ktorých rozdeľujú obraz na jednotlivé časti. Pre vyhľadávanie hrán je možné použiť konvolučné metódy alebo prístup pomocou soft computing.



Obr. 1.5: Segmentácia pomocou Rozdeľovania a zlučovania oblastí: Originálny obraz (a), obraz prevedený do odtieňov šedej (b), obraz po rozdeľovaní (c), obraz po zlučovaní (d)

1.2.1 Konvolučné metódy

Fungujú na základe konvolúcie masky s obrazom, pri ktorej vzniká obraz so zvýraznenými hranami. Na základe prahu je následne možné určiť, či sa jedná o hranu, alebo nie. Prahovať sa môže podľa rôznych vlastností, napríklad priemeru všetkých pixelov, alebo podľa histogramu [2] .

Roberts

Vzniká pomocou konvolúcie dvoch masiek s obrazom:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Následne sa vytvoria dva obrazy gradientov, ktoré sa spájajú pomocou vzorca:

$$\nabla I(x,y) = G(x,y) = \sqrt{G_x^2 + G_y^2}$$

kde $I(x,y)$ je originálny obrázok a G_x/G_y sú obrazy gradientov.

Sobel, Prewitt

Tieto metódy fungujú na rovnakom princípe ako Robertsova metóda s tým rozdielom, že používajú na konvolúciu iné masky. Gradientné obrazy sú vytvorené:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -c & 0 & c \\ -1 & 0 & 1 \end{bmatrix} * A, G_y = \begin{bmatrix} -1 & c & 1 \\ 0 & 0 & 0 \\ 1 & c & 1 \end{bmatrix} * A$$

kde c je operátor, ktorý vytvára konvolučnú maticu. Pri Sobelovej metóde je $c = 2$ a pri Prewittovej $c = 1$. G je gradientný obraz a A je segmentovaný obraz [23] .

Rozširovanie a zlučovanie hrán

Je metóda spracovania obrazu, v ktorej sa dopĺňa použitie hrán a okrajov pri segmentácií obrazu. Používa sa najmä po konvolučných metódach segmentácie pre zníženie počtu nepotrebných hrán. Príkladom rozširovania a zlučovania hrán je v daľšom paragrafe popísaný Canny edge detector [5] .

Na začiatku sa používa jedna z derivačných typov konvolučných masiek . Najpoužívaniejšou konvolučnou maskou sa používajú Sobelove masky. Následne sa vyhľadajú hrany v obraze a hľadá sa podľa masky okolia hrany a smeru hrany lokáne maximum. Tento proces sa nazýva ztenšovanie hrán. Smer hray je vypočítaný podľa pomeru aktivácie Sobelovej masky pre smer x a smer y. Lokálnemu maximu priradí hranu najmenšej veľkosti. V druhom kroku sa používa hysterézne prahovanie, ktoré používa dva prahy. Dvoma prahmi je možné rozdeliť hrany do troch oblastí. Prvá s najvačšou intenzitou sa automaticky stáva hranou. Druhá s priemernou intenzitou hrany je určená podľa toho, či je hrana pripojená k hrane v prvej oblasti. Posledná oblasť určuje zahodené hrany.

Výhoda tejto metódy je jej jednoduchosť a efektívnosť. Problémom tejto metódy je použitie správnych prahov, masky a orientácie.

Obr. 1.6: Referenčný obraz

Obr. 1.7: Obraz po použití Canny detektoru hrán

1.2.2 Soft Computing

Je zlúčenie techník, ktoré simulujú ľudské myslenie, aby vyriešili komplexné úlohy. Oproti iným metódam pracuje soft computing s neistotou a aproximáciou pre riešenie širokého okruhu problémov [31] .

Fuzzy logika

Je podmnožinou matematickej logiky, ktorá používa namiesto tvrdého zaraďovania $[0, 1]$ interval $< 0, 1 >$. Pre príklad je popísané fuzzy zhlukovanie c-priemerov, ktoré je variantou zhlukovania k-priemerov. Na začiatku sa vyberie c (počet častí) a priradia sa centroidy častí k náhodným bodom. Vyberie sa podmienka chybovosti ϵ, fuzzy faktor $m > 1$ a inicializuje sa pravdepodobnosť $U = [u_j i]$. Následne sa prepočítavajú centroidy:

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m . x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

kde x je kvantitatívna vlastnosť pixelu (napríklad odtieň šedej) a u_{ij} je:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{||x_i - c_j||}{||x_i - c_k||} \right)^{\frac{2}{m-1}}}$$

Podmienka ukončenia algoritmu je $||\Delta U|| < \epsilon$ [19] . Tento algoritmus rieši prelínanie oblastí, avšak rovnako ako zhlukovanie k-primerov nepoužíva priestorovú informáciu. Tento problém je možné riešiť pomocou [13] .

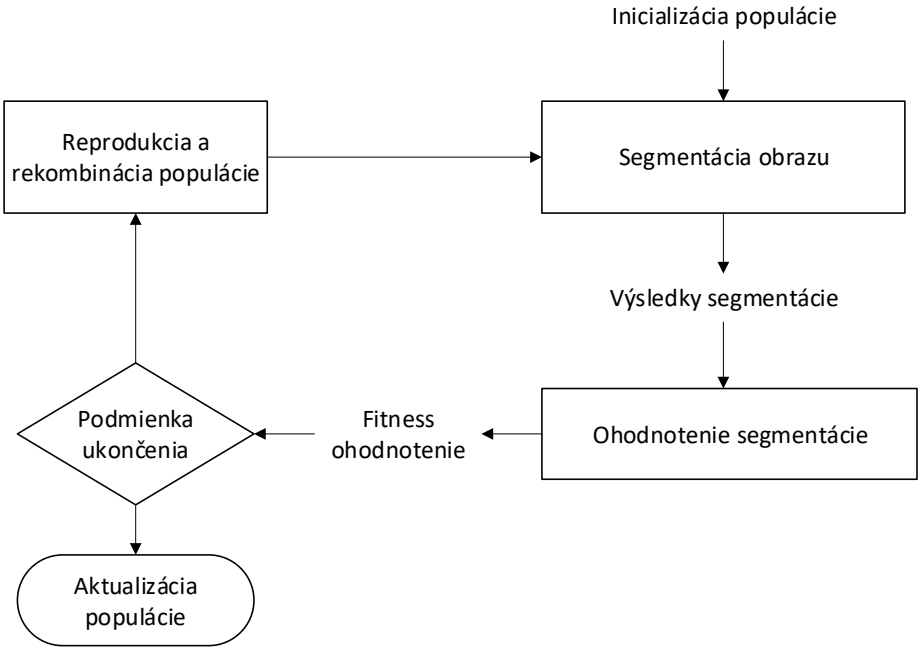
Genetické algoritmy

Sú algoritmy riešiace optimalizačné problémy pomocou evolučnej teórie. Fungujú na princípe prirodzeného výberu jedincov (segmentácií) podľa fitness funkcie, ktorá

ohodnotí, s akou presnosťou je obraz segmentovaný. Z prirodzeného výberu vytvorí algoritmus novú generáciu, ktorá vznikne rekombináciou génov (vlastností) ich rodičov s pridanými anomáliami. Celý cyklus sa opakuje, pokým nie je splnená určená úspešnosť [8].

Neurónové siete

Neurónová sieť je spojenie umelých neurónov pre vytvorenie systému, ktorý sa podobá ľudskému mozgu. Tento systém je robustný a invariantný voči šumu, avšak jeho nevýhodou je veľká výpočtová náročnosť. Pre segmentáciu obrazu sa v súčastnosi používajú konvolučné neurónové siete, ktoré dosahujú vysoké presnosti [33] . Táto práca používa na segmentáciu obrazu grafové neurónové siete, ktoré sú popísané v nasledujúcej kapitole.



Obr. 1.8: Bloková schéma genetických algoritmov

2 Grafové neurónové siete

Vo svete existuje veľké množstvo vzťahov medzi objektami, ktoré je možno reporezentovať pomocou grafov. Príkladom je pochopenie vzťahov medzi ľuďmi v sociálnej sieti alebo chápanie štruktúrneho vzorca chemických látok pre určenie návykovosti danej látky pre človeka [46] . V rámci algoritmov pre rozpoznávanie vzorov v grafoch a chápaniu ich významu sa zaoberala analýza siete a teória grafov. Pomocou grafových neurónových sietí je možné ľahšie pochopiť komplexnú štruktúru topológie, vybraných dát a smeru toku informácií v grafe [34] .

Grafové neurónové siete sú neurónové siete, ktorých vstupné a výstupné dáta sú grafy. Nakoľko grafom je možno popísať veľké množstvo informácií má grafová neurónová sieť široké využitie. Tatiež sa dá táto sieť použiť pre všetky možné typy grafov [35] . Nakoľko iné typy neurónových sietí použitých na grafy sú viazené na jeden typ grafov [29, 38] , majú grafové neurónové siete obrovskú výhodu.

2.1 Teória grafov

Graf G je usporiadaný pár disjunktných množín (V, E) tak že E je podmnožina V^2 neusporiadaných dvojíc V [7] . Množinu V nazývame vrcholy a množinu E hrany. Hlavné vlastnosti, ktoré je potrebné zohľadniť pri grafoch:

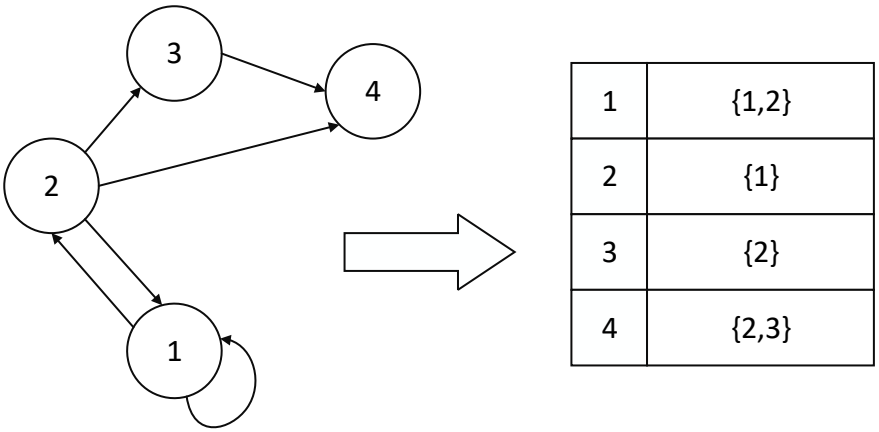
- Vážený/Nevážděný - podľa toho či sú hrany E ohodnotené váhou
- Orientovaný/neorientovaný - podľa toho či majú hrany E orientáciu
- Cyklický/acyklický - podľa vytvárania cyklov v orientovanom grafe

2.1.1 Graf a použitie v počítačovej vede

Pre použitie teórie grafov je potrebné transformovať grafy do abstraktej dátovej podoby spracovateľnej počítačom [15] . V praxi je možné ich rozdeliť na:

- List susednosti - list obsahujúci vrcholy, v ktorých sú uložené susediace vrcholy 2.1
- Matica susednosti - matica logických hodnôt, v ktorej každá pozícia (n, m) popisuje susednosť vrcholov n a m 2.3
- Matica incidencie - matica logických hodnôt, v ktorej každá pozícia (h, v) popisuje susednosť hrany h k vrcholu v

Pre vybranú dátovú štruktúru je potrebné zaistiť základné metódy pre vytváranie a manipuláciu z grafom [32] . Príkladom týchto operácií sú pridávanie a odstraňovanie hrán a vrcholov, nastavenie ich váh alebo výber susedných vrcholov.



Obr. 2.1: Vytvorenie listu susednosti

2.2 Rozdelenie grafových sietí

V rámci strojového učenia je možné rozdeliť grafové siete do podkategórií podľa novej taxonómie [43] :

- rekurentné grafové neurónové siete,
- konvolučné grafové neurónové siete,
- grafové autoenkodéry,
- časovo premenné grafové neurónové siete.

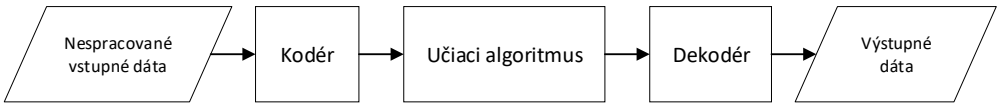
V rámci praktickej časti je použitý typ rekurentnej grafovej neurónovej siete a preto je teoretická časť učiaci algoritmus 2.3.3 venovaná práve týmto sieťam.

2.3 Model grafovej neurónovej siete

Algoritmus grafovej neurónovej siete pozostáva z načítania nespracovaných dát do podoby grafu 2.2. Následne je potrebné zakódovať tieto dáta do vektorovej alebo maticovej podoby tak aby bol schopný enkodér získať dôležité informácie pre spracovanie učiacim algoritmom. Problém zakódovania dát je možné poňať ako prenos z vektorovej alebo maticovej podoby grafu do viacrozmerného euklidovského priestoru so zachovaním štruktúry grafu [26, 6] . Následne je použitý učiaci algoritmus pre spracovanie dát. Poslednou časťou je dekodér, ktorého úlohou je spätná transformácia dát do vektorovej alebo maticovej podoby.

2.3.1 Kódovanie dát

Cieľom kódovania grafu je reprezentácia hrán E a vrcholov V do vektorového priestoru alebo matice tak, aby táto informácia obsahovala ako aj informácie o štruktúre grafu tak aj parametre jednotlivých vrcholov a hrán. V rámci grafových sietí je



Obr. 2.2: Bloková schéma Grafových neurónových sietí

možné sa stretnúť s kernelovým spracovaním grafov a spracovaním grafov pomocou zreťazenia. Problém pre kernelové predspracovanie dát vzniká pri vytváraní veľkých matíc párovej podobnosti a taktiež k tažkostiam pri zmene štruktúry grafu. Zreťazením grafov na základe reprezentácií je možné efektívnejšie klasifikovať dáta. Poslednou možnosťou je použitie grafových autoenkodérov, ktoré zakódujú vrcholy grafu do vkladacieho priestoru tak aby podobnosť vrcholov a hrán v tomto priestore aproximovala ich podobnosť v grafe [25]. V rámci praktickej časti je použitý grafový autoenkodér node2vec a preto je popísaný medzi metódami použitými v enkodéroch.

Pri vytváraní metódy kódovania je nutné riešiť ťažkosti:

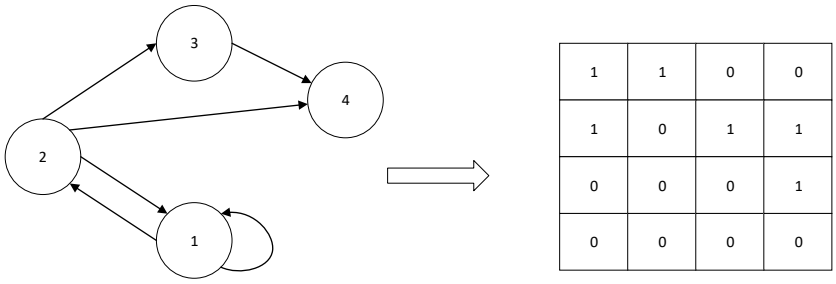
- Výber parametrov - je potrebný správny výber parametrov aby sa zachovala štruktúra grafu a aby vznikali rozdiely v kódovacom priestore. S týmto sa taktiež spája dimenzia vektoru, ktorá vznikne z parametrov. Je preto potreba vytvoriť kompromis medzi presnosťou a výpočetnou náročnosťou.
- Škálovateľnosť - je potrebné aby mohol systém spracovať veľké množstvo dát a preto sa využíva škálovateľnosť. Pre tento systém je následne potrebné vymyslieť zlučovací mechanizmus [37] tak, aby sa nezmenili globálne parametre.

2.3.2 Metódy použité v enkodéroch

V súčastnosti sa používajú tri typy metód alebo ich kombinácia pre zakódovanie grafu pomocou kodéru:

- Faktorizácia grafu
- Náhodné prechádzanie grafom
- Hlboké učenie

Faktorizácia grafu Fungujú na princípe faktorizácie matice podobnosti grafu. Matice podobnosti grafu môžu byť napríklad matica susednosti, Katzova podobnosť alebo Adamic-Adar podobnosť. Snahou faktorizácie matice je znižovať stratovú funkciu pomocou zmeny matice podobnosti. Pre lepšie vysvetlenie je popísané Mapovanie vyššieho stupňa podobnosti z angl."High-order proximity embedding". Tá vytvára maticu vyššieho stupňa podobnosti S, kde S_{ij} je podobnosť medzi vrcholom v_i a v_j. Taktiež vytvára mapovanie vlastností vrcholov $\mathbf{U} = [\mathbf{U}^s, \mathbf{U}^t]$, pričom $\mathbf{U}^s$ a $\mathbf{U}^t$ sú zdrojové a cieľové vrcholy. Počet riadkov a stĺpcov zodpovedá počtu vrcholom



Obr. 2.3: Vytvorenie matice susednosti

a ich dimenzií. Následne je potrebné vytvoriť maticu podobností ako:

$$\mathbf{S} = \mathbf{M}_g^{-1}.\mathbf{M}_l,$$

kde $\mathbf{M}_g^{-1}$ a $\mathbf{M}_l$ sú polynómy matíc podľa typu podobnosti. Pre Katz podobnosť sú tieto polynómy:

$$\mathbf{M}_g = \mathbf{I} - \beta.\mathbf{A}, \mathbf{M}_l = \beta.\mathbf{A}$$

$\mathbf{A}$ je matica susednotsti kde a_{ij} je počet spojení z i-teho do j-teho vrcholu 2.3 . β je parameter rozkladu, ktorý určuje rozklad váhy medzi vrcholmi podľa vzdialenosti vrcholov.

Po vytvorení matice podobnosti je potrebné aproximovať vektory vrcholov v $\mathbf{U}$. Pre to sa používa singulárny rozklad matice podobnosti:

$$\mathbf{M}_g^{-1}\mathbf{M}_l = \mathbf{V}^s\Sigma\mathbf{V}^{t\top},$$

kde $\mathbf{V}^s$ a $\mathbf{V}^t$ sú ortogonálne matice. Z tejto rovnice získame

$$\Sigma = diag(\sigma_1, \sigma_2, \ldots, \sigma_N),$$

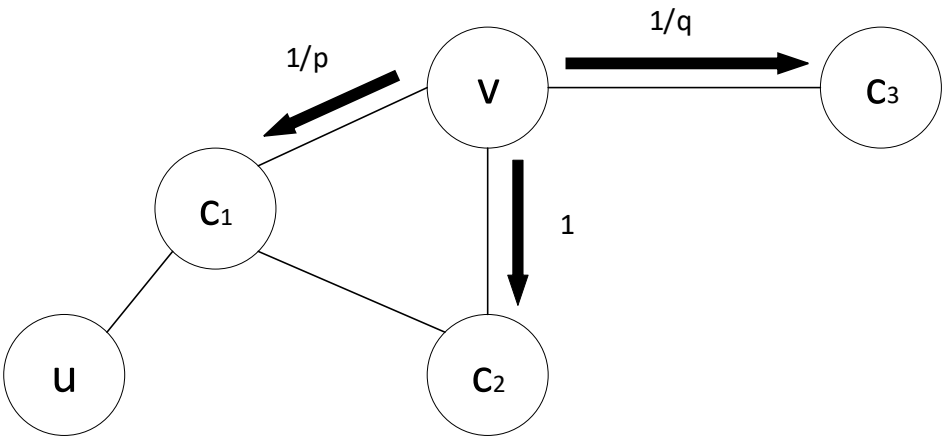
ktorými sú aproximované mapované vektory

$$\mathbf{U}^s = [\sqrt{\sigma_1}.\mathbf{V}_1^s, \ldots, \sqrt{\sigma_K}.\mathbf{V}_K^s], \mathbf{U}^t = [\sqrt{\sigma_1}.\mathbf{V}_1^t, \ldots, \sqrt{\sigma_K}.\mathbf{V}_K^t]$$

Matice $\mathbf{U}^s$ a $\mathbf{U}^t$ je možné použiť pre učiaci algoritmus. Tento algoritmus je použiteľný pre veľké grafy avšak má kvadratickú zložitosť pre parametre vrcholov. Taktiež jej komplexnosť závisí na počte iteráciách aproximácie. Problémom tohto riešenia je prepočet matíc pri zmene grafu.

Náhodné prechádzanie grafom Tento prístup k problému zakódovania grafu používa zmenu mapovania vrcholov tak, aby zvýšila šanca že podobné vrcholy budú mať spoločnú cestu. Ako príklad si je popísaný algoritmus Node2Vec [26] .

Prechádzanie grafom Pre správne vytváranie náhodných ciest z vrcholu u je potrebné použiť informácie z lokálneho aj globálneho celku. Preto bol v node2vec vytvorený systém náhodne zaujatého prechádzania. Tento systém používa váženú náhodosť podľa návratového parametru p a vstupnovýstupný parameter q. Zvyšovaním parametru p sa znižuje pravdepodobnosť, že náhodná cesta bude obsahovať rovnaké vrcholy. Vstupnovýstupný parameter udáva pravdepodobnosť globálneho-/lokálneho prechádzania grafom podľa jeho znižovania/zvyšovania 2.4.



Obr. 2.4: Náhodne zaujaté prechádzanie grafom

Optimalizácia Ako je už vyššie spomenuté je potrebné optimalizovať mapovanie tak aby mali podobné vrcholy väčšiu pravdepodobnosť výskytu náhodných ciest:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log P(v|\mathbf{z}_u)$$

, kde sa sumarizuje logaritmus pravdepodobnosti výskytu vrcholu v v ceste každého vrcholu u. Následne je potrebné parametrizovať pravdepodobnosť $P(v|\mathbf{z}_u)$. Na to je možné použiť softmax funkciu, avšak výpočet tejto funkcie by zaberal príliš mnoho výpočetného času takže sa autor rozhodol pre jej aproximáciu:

$$\log P(v|\mathbf{z}_u) = \log\left(\frac{\exp \mathbf{z}_u^\top \mathbf{z}_v}{\sum_{n \in V} \exp \mathbf{z}_u^\top \mathbf{z}_n}\right) \approx \log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - \sum_{i=1}^{k} \log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})\right), n_i \sim P_v$$

, kde σ je funkcia sigmoidy a n_i je i-ty vybraný vrchol z náhodnej distribúcie vrcholov P_v vybranej zo všetkých vrcholov. Parameter k popisuje počet náhodne vybraných vrcholov. Po vypočítaní pravdepodobnosti je použité stochastické klesanie gradientu tak aby sa táto pravdepodobnosť zvýšila.

Zhrnutie Výhoda tohto algoritmu prichádza so stochastickou definíciou podobnosti vrcholov čím sa získava lokálna aj globálna informácia. Taktiež nie je potrebné využívať všetky hrany nakoľko metóda používa náhodné prechádzanie.

Hlboké učenie Sú metódy, ktoré upravujú mapované dáta tak, aby sa získavali najdôležitejšie informácie. Používajú na to neurónové siete, pričom využívame rozpamätávanie tejto siete. Ako zástupcu použitia hlbokého učenia pre zakódovanie grafov je popísaný model učenia grafových reprezentácií pomocou hlbokých neurónových sietí alebo DNGR [10] .

DNGR Z angličtiny "Deep Neural networks for learning Graph representations"je model, ktorý používa stratégiu náhodného surfovania po vrcholoch podobná ako pri node2vec 2.3.2, avšak na optimalizáciu používa strátégiu potláčania šumu pomocou neurónovej siete. Taktiež využíva vrstviacu techniku pre redukciu dimenzie kódovaných dát.

Náhodné surfovanie vyberie náhodné zoradenie vrcholov a vytvorí maticu prechodov A. Taktiež vytvorí vektor

$$p_k = \alpha.p_{k-1}.A + (1-\alpha)p_0,$$

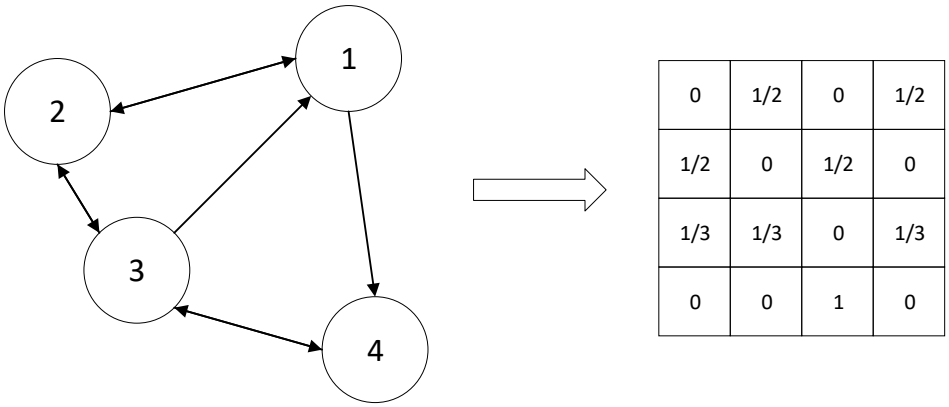
ktorý je pravdepodobnosť dosiahnutia j-teho vrcholu po k skokoch. Náhodné prechádzanie má pravdepodobnosť $1-\alpha$ že za vráti do začiatočného vrcholu. Reprezentácia vrcholu získavame pomocou náhodných prechádzaní grafom:

$$r = \sum_{k=1}^{K} w(k).p_k^*,$$

kde w(.) je klesajúca funkcia tak, aby s každým ďalším skokom táto funkcia klesala.

Vrstviaca technika s potláčaním šumu Z hodnôt pravdepodobností získame maticu pozitívnych bodovo vzájomných informácií [9] . Následne sú z týchto dát vytvorené vektory, ktoré sú použité do autoenkóderu s potláčaním šumu. Ten sa zkladá z zarušenia dát, ktoré náhodne zarušia náhodné dimenzie vektoru x na 0. Tieto dáta sa používajú pre viacvrtsvový perceptron, ktorý sa snaží minimalizovať kvadratickú chybu rekonštruovaného obrazu oproti korektnému obrazu. Procesom učenia tejto neurónovaj siete vzniknú filtre, váhy jednotlivých perceptronov, ktoré určujú dôležité vlastnosti matice a tým znižujú jej redundanciu.

Tento typ kódovania informácie dokáže prekonať iné moderné technológie, avšak je použitý len pre malé dimenzie vektorov. Taktiež je potrebné odborné naladenie hyperparametrov pre správnu funkciu.



Obr. 2.5: Vytvorenie matice prechodov

2.3.3 Učiaci algoritmus

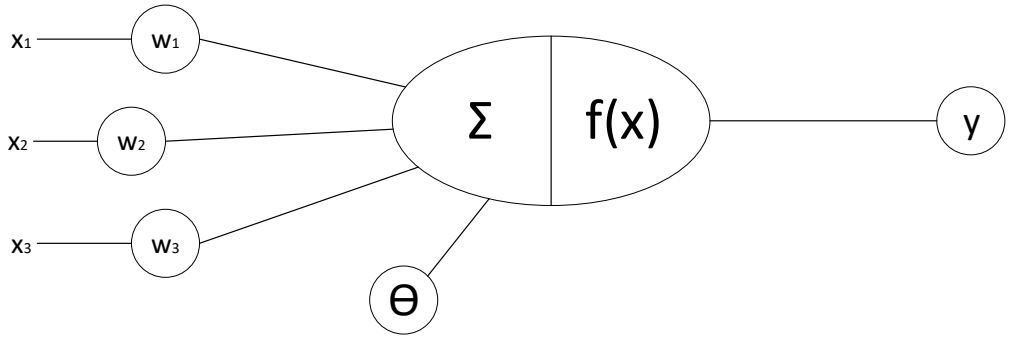
Vo všeobecnom prípade grafových sietí je možné použiť akýkoľvek typ neurónovej siete. Pre grafové vstupné dáta sa však používajú zvača konvolučné neurónové siete pracujúce s maticami. Problém pri maticovom zápise nastáva pri zmene grafu alebo pri veľkých veľkostiach matíc. Problém zmeny grafu riešia časovo premenné neurónové siete [44], ktoré sa používajú pre spracovanie videosekvencií pre klasifikáciu typu pohybu. Pre tento typ problému boli taktiež využité nelokálne grafové neurónové siete [42]. Problémy vychádzajúce z veľkosti vstupných matíc a potreby konvolúcie boli vyriešené pomocou neurónovej siete prenášajúcej správy [46] .

Pre vysvetlenie práca popisuje neurónovú sieť prenášajúcu správy nakoľko je tento typ siete možné použiť pre segmentáciu obrazu.

Neurónové siete prenášajúe správy

Tieto neurónové siete sa zkladajú z časti prenášania správ pomocou váh vo vnútri neurónovej sieťe.

Umelá neurónová sieť Je počítačový systém, ktorý je inšpirovaný neurónovými sieťami v mozgu. Táto sieť sa zkladá z umelých neurónov, ktoré sú pospájané do navrhnutej architektúry. Taktiež je potrebné nastaviť zmenu systém zmeny váh pre učenie neurónovej siete.



Obr. 2.6: Umelý neurón

Umelý neurón sa zkladá zo vstupných informácií X, ktoré sa následne násobia váhami W. Tieto informácie sa sumarizujú spolu s prahom Θ a použijú sa pre aktivačnú funkciu $f(x)$. Výsledok funkcie je výstup y.

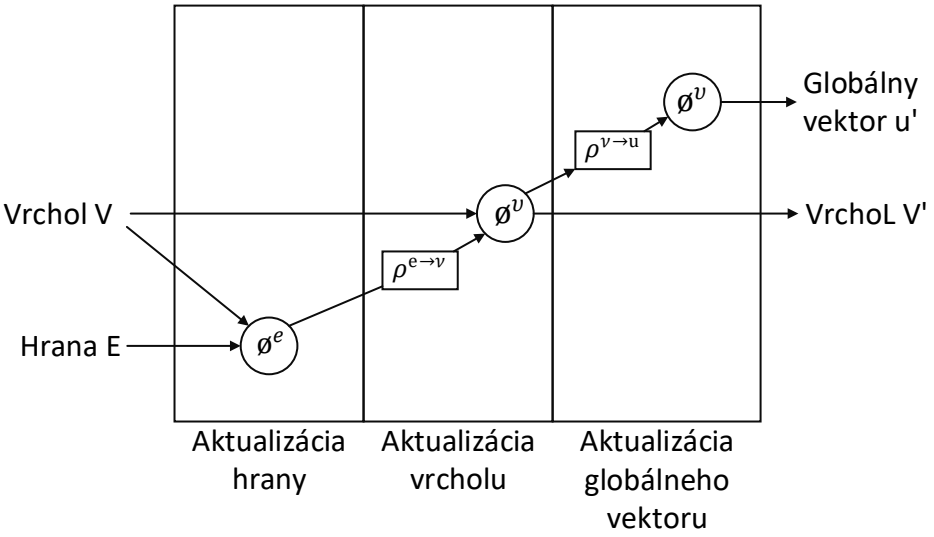
Prenášanie správ Je iteratívny proces prenášania vlastností vrcholov a hrán v grafovej neurónovej sieti [6]. Počet iterácií je hyperparametrom grafovej neurónovej siete, ktorý označuje veľkosť okolia, z ktorých neuróny získavajú informácie. Prenášanie zpráv sa zkladá z výpočtu jednotlivých správ a z aktualizačnej funkcie pre jednotlivé neuróny. Najprv sa získavajú jednotlivé vlastnosti okolia v-teho vrcholu:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}),$$

kde t je číslo iterácie a M je funkcia správy, akou sa zlučujú tieto dáta [21] . Vo vnútri tejto funkcie sa nachádzajú vlastnosti vrcholu h_v^t, okolitých vrcholov h_w^t a vlastnosti hrany spájajúceho tieto vrcholy e_{vw}. Následne sa vrcholy v aktualizujú funkciou:

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

Nakoniec sa tieto vlastnosti zakódujú do vektorov vlastností grafu. Po prenášaní správ sa posielajú vektory do umelej neurónovej siete, ktorá je trénovaná na špecifickú operáciu.



Obr. 2.7: Schéma prenášania správ

2.3.4 Dekodér

Dekodér má v pri grafových neurónových sieťach dvojaký zmysel. V prvom rade sa používa pre optimalizáciu mapovania do euklidovského priestoru. V druhom zmysle sa tento pojem používa pre dekódovanie latentnej reprezentácie grafu z výstupu učiaceho algoritmu.

Dekódovanie autoenkodérov pre učenie

Pre grafové autoenkodéry je potrebné určenie podobnosti jednotlivých vrcholov medzi sebou pre zaistenie správneho mapovania vstupnej grafovej štruktúry.

Pre dekódovanie v rámci podobnosti sa zvačša používa párový dekodér. Cieľom dekódovania podobnosti je znížiť stratovú funkciu voči podobným párom:

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{D}} l(DEC(\mathbf{z}_i,\mathbf{z}_j), s_{\mathcal{G}}(v_i,v_j)),$$

kde DEC je dekódovacia funkcia párov vrcholov vo vektorovej forme a $s_{\mathcal{G}}$ je podobnosť vrcholov definovaná užívateľom. l je stratová funkcia definovaná užívateľom. Príklady definície týchto funkcií je možné vidieť v tabuľke [30] :

Metóda	Dekodér	Stratová funkcia
Laplaceove Eigenmapy	$\|\|\mathbf{z}_i - \mathbf{z}_j\|\|_2^2$	$DEC(\mathbf{z}_i,\mathbf{z}_j).s_{\mathcal{G}}(v_i,v_j)$
HOPE	$\mathbf{z}_i^\top \mathbf{z}_j$	$\|\|DEC(\mathbf{z}_i,\mathbf{z}_j) - s_{\mathcal{G}}(v_i,v_j)\|\|_2^2$
node2vec	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k\in V} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$-s_{\mathcal{G}}(v_i,v_j)\log(DEC(\mathbf{z}_i,\mathbf{z}_j))$

Dekódovanie reprezentácie grafu

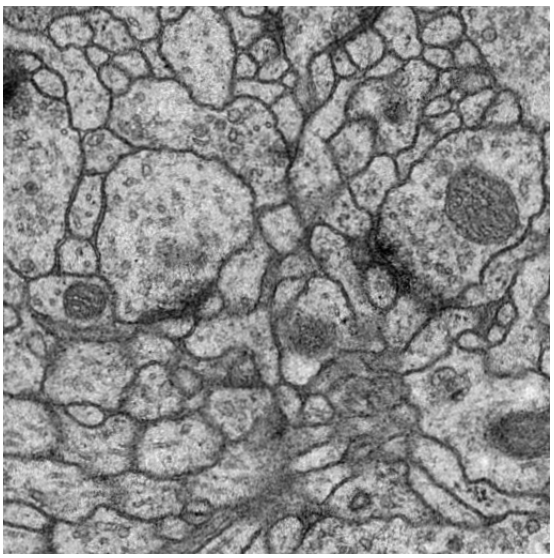
Je dekódovanie výstupných dát z učiaceho algoritmu do vektorovej podoby pre jednotlivé vrcholy a dátovej štruktúry grafu. Príkladom je použitie neurónovej sieťe, ktorá funguje ako spätná tranformácia vstupnej neurónovej siete kodéru [6] . V tejto práci je vytvorený dekodér ako jednovrstvový perceptron pre hrany, vrcholy a globálne parametre, ktorého očakávaným výstupom sú učené vektory grafu.

3 Praktická časť semestrálnej práce

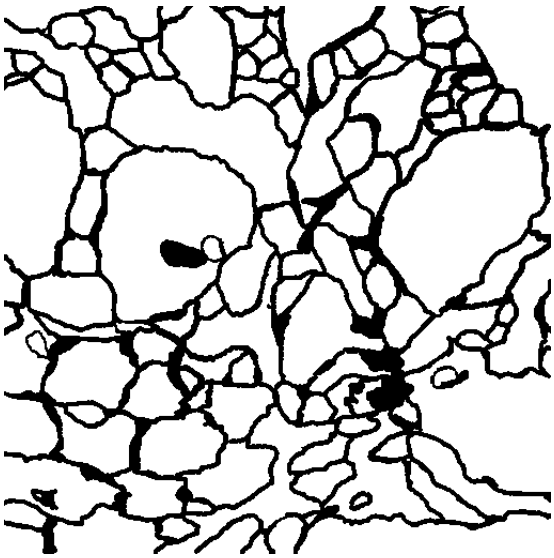
Nasledujúca kapitola popisuje parktický návrh grafovej neurónovej siete pre segmentáciu medicínskych dát. Objasňuje konkrétny problém, návrh a popis riešenia práce a dosiahnuté výsledky. V analýze problému sú charakterizované vstupné a referenčné dáta. V návrhu sú popísané použité knižnice, hardvérové prostriedky a riešenie problému. Taktiež sú v časti testovania dosiahnutých výsledkov opísané metriky testovania segmentácie medicínskych dát.

3.1 Rozbor problému

Cieľom praktickej časti diplomovej práce bolo sprevádzkovať grafovú neurónovú sieť a použiť ju pre automatickú segmentáciu medicínskych dát. Pre trénovanie a testovanie segmentácie boli použité biomedicínske dáta získané elektrónového mikroskopu, ktorý sledoval časť centrálnej nervovej sústavy drozofilov [11] . Tento dataset obsahoval tridsať snímkov z elektrónového mikroskopu a ich referečné segmentácie. Snímky majú veľkosť 512×512 pixelov pričom jeden pixel má reálnu veľkosť $4 \times 4\,\text{nm}$.



Obr. 3.1: Obraz z medicínskych dát

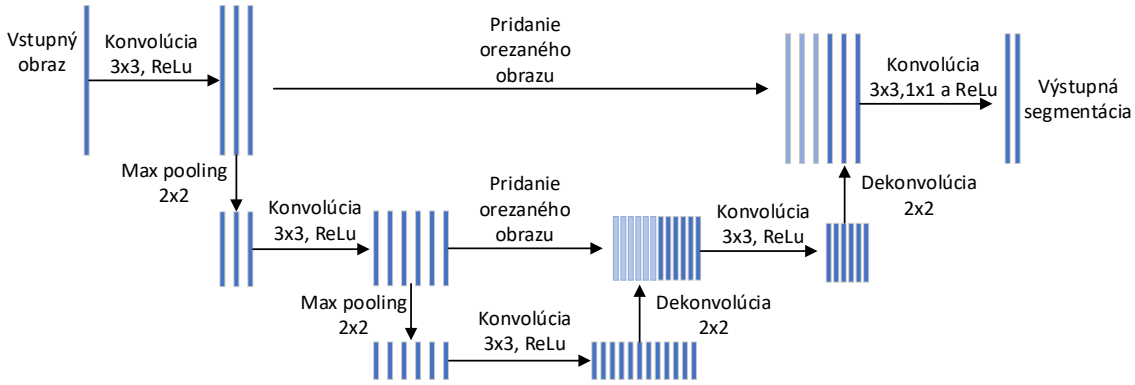


Obr. 3.2: Referenčná segmentácia obrazu

3.2 Moderné techniky segmentácie obrazu

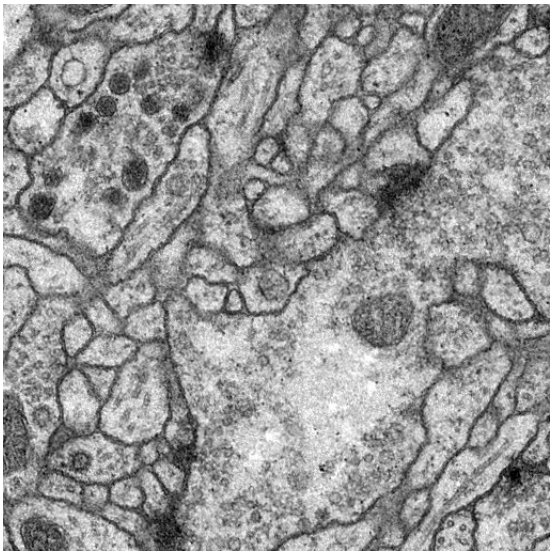
Existuje viacero možností ako pristupovať k problému segmentácie [1] avšak v posledných rokoch sa dostalo do popredia používanie konvolučných neurónových sietí. Konkrétne v roku 2015 vyšla publikácia na zlepše konvolučných neurónových sietí U-Net [33] . Táto neurónová sieť sa zkladá z častí konvolúcie, združovania maxima

(max pooling) a kopírovania dát podľa obrázku 3.3. Tento systém je navrhnutý z dôvodu zníženia vstupných dát do neurónovej siete a taktiež pre získanie lokálnych a globálnych informácií z obrazu pomocou konvolúcie.

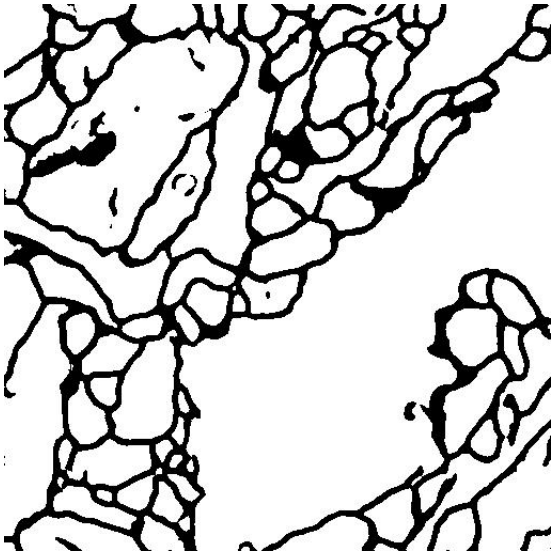


Obr. 3.3: Schéma U-Net

Táto sieť bola taktiež použitá pre segmentáciu medicínskych dát, ktoré boli použité v praktickej časti tejto práce. Pre segmentáciu týchto dát dosiahla pixelovú presnosť 94%. Výsledky segmentácie U-Net znázornené pomocou výstupnej segmentácie [33]:



Obr. 3.4: Obraz z medicínskych dát



Obr. 3.5: Segmentácia obrazu použitím U-Net

3.3 Navrhované riešenie

Táto sekcia sa zaoberá praktickým návrhom grafovej neurónovej siete v práci.

Prvá časť „Štruktúra projektov“ sa zaoberá adresárovou štruktúrou praktickej časti respektíve popisu významu jej zložiek a súborov.

Model grafovej neurónovej siete popisuje výber modelu grafových sietí pre segmentáciu. Zaoberá vrstvami a popisom neurónovej siete použitej v praxi.

Subsekcia „Vstupné dáta“ opisuje proces predspracovania dát do neurónovej siete pre najlepší výsledok. Taktiež je v tejto časti komentár ku predspracovaniu dát z hľadiska segmentácie a chápaniu grafov.

Trénovanie siete popisuje samotný učiaci algoritmus. Výber učenia, propagácie chyby a optimalizácie sú zahrnuté v tejto subekcii.

Posledná časť Inicializačné parametre sa venuje súboru `Initalizaion.py`, ktorý zoskupuje väčšinu hyperparametrov pre rýchle nastavenie neurónovej siete.

3.3.1 Štruktúra projektu

Praktická časť obsahuje viaceré projekty, pretože bolo potrebné vyskúšať viacero typov grafových neurónových sietí, avšak každý typ má v celku podobnú štruktúru so zmenou v knižnici neurónovej siete. Pre praktickú časť bola použitá grafová neurónová sieť s prenášaním správ `graph_nets`, grafový autoenkodér `node2vec` a grafová konvolučná sieť `keras_deep_graph_learning`. Kompletná štruktúra projektu sa nachádza v prílohe A .

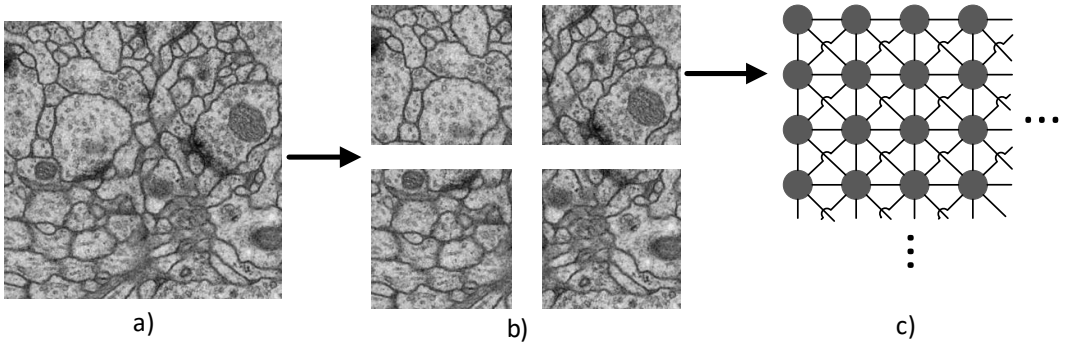
3.3.2 Model grafovej neurónovej siete

Hlavný model `graph_nets` použitý pre projekt sa zkladá z kodéru, jadra a dekodéru `EncodeProcessDecode` [6] . Vrstva enkodéru a obsahue viacvrstvový perceptron pre zakódovanie vektorov vrcholov a hrán grafu do stavového euklidovského priestoru. Jadro grafovej neurónovej siete tvoria viacvrstové perceptrony, ktorých počet zodpovedá počtu prenosov správ. Tu sú upravené hrany, ktorými sa aktualizujú vrcholy. Poslednou časťou je dekodér pre dekódovanie aktualizovaných vrcholov a globálnych parametrov. Globálne parametre použité v `graph_nets` sa v tejto práci nepoužívajú.

3.3.3 Vstupné dáta

Vstupné dáta sú v podobe obrázku a preto bolo potrebné použiť metódu pre zakódovanie obrázku do grafu. Vrcholy grafu sú preto určené pixelmi s obsahujúcou informáciou odtieňa sivej. Hrany grafu sú určené parametrom `region` a označujú vzdialenosť pixelov, ku ktorým sú pripojené. Váhy hrán sú určené euklidvou vzdialenosťou pripojených pixelov. Všetky tieto informácie sú normalizované pre trénovanie neurónovej sieti. Obrázky boli rozdelené do častí 128×128 pre získavanie globálnej informácie pri prenášaní správ. Taktiež bolo použité podvzorkovanie obrazu jedna ku

dvom, ktoré bolo následne pripojené k danému grafu. O vytváranie grafov z obrázku sa stará metóda `generate_graphs` B .



Obr. 3.6: Metóda generate graphs:Vstupné dáta (a), Rozdelenie obrazov (b), Vytvorenie grafov (c)

3.3.4 Trénovanie

Pre správne natrénovanie neurónovej siete je potrebné zadefinovať správne funkcie a hyperparametry. Aktivačná funkcia neurónov bola zvolená rektifikovaná lineárna jednotka ReLu, nakoľko poskytovala najlepšie výsledky. Náhodný kľúč pre pseudonáhodný generátor váh neurónov predvolene nastavený na desať avšak pre iné aktivačné funkcie bol zmenený. Práca používa stratovú funkciu softmax s optimalizáciou odhadom adaptívneho momentu (AdamOptimizer) avšak boli použité aj iné pre porovnanie.

3.3.5 Parametre siete

V rámci každého projektu je vytvorený inicializačný súbor `initialization.py`, ktorý obsahuje všetky globálne parametre na správu neurónovej siete:

- `seed` - jadro pre výpočet náhodných počiatočných váh neurónovej siete.
- `num_processing_steps_X` - Počet prechádzaní správ pre trénovaciu a testovaciu množinu.
- `num_training_iterations` - Počet epoch
- `learning_rate` - rýchlosť učenia optimalizátora
- `beta1,beta2,epsilon,use_locking` - parametre adam optimalizátora
- `X_start_image,X_end_image` - začiatok a koniec obrazov trénovacej a testovacej množiny
- `log_every_seconds` - čas logovania dice koeficientu množín
- `log_every_outputs` - logovanie výstupu testovacích množín

Ostatné parametre v inicializačnom súbore neboli použité pre neurónovú sieť.

3.3.6 Použitý hardvér a cloudové služby

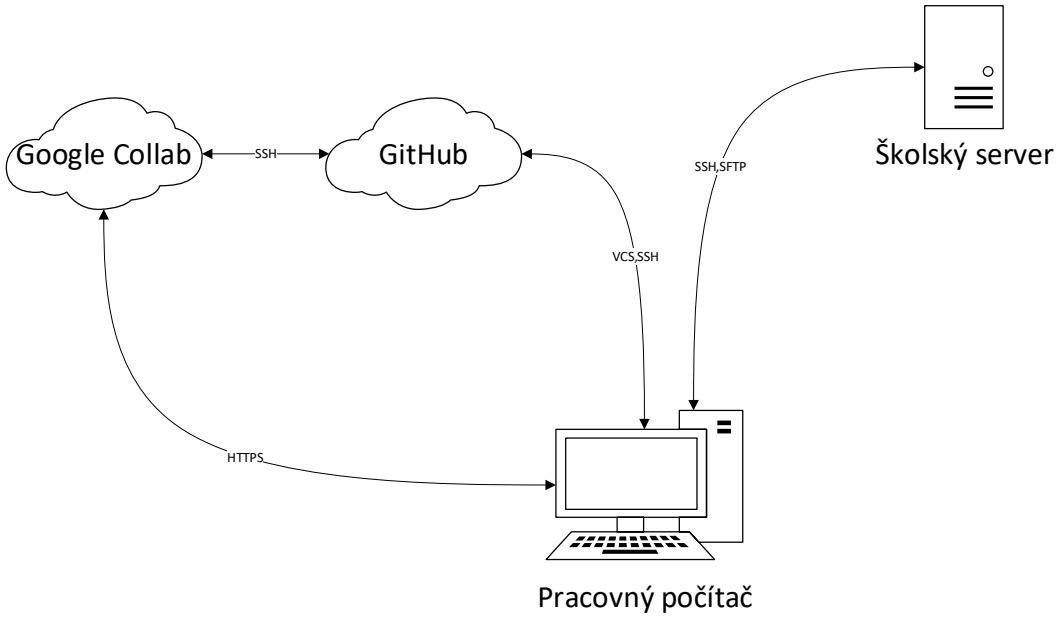
Pre učenie grafovej neurónovej siete bolo potrebné použiť rýchle výpočetné prostriedky. Táto práca používa dva typy grafických procesorov a to Nvidia T4 a Nvidia Titan Xp. Nvidia T4 bola použitá v cloudovej službe Google Collaboratory. Táto služba ponúka vývojové prostredie Jupyter, cloudové úložisko 300Gb a výpočetné prostriedky v podobe grafických procesorov. Výpočetné prostriedky Nvidia Titan Xp boli zapožičané cez školské servery.

Grafická karta	Nvidia Titan Xp	Nvidia T4
Architektúra	Pascal	Turing
Max.frekvencia[Mhz]	1582	1590
CUDA jadrá[-]	3840	2560
Pamäť[Gb]	12	16
Rýchlosť pamäte[Gb/s]	547.7	320.0
Príkon[W]	250	70

Tab. 3.1: Špecifikácia použitých grafických kariet

Prepojenie ku Google Collabu bolo vytvorené cez službu GitHub klonovaním vytvoreného repozitáru [22, 24] .

GitHub poskytuje hostovanie cloudového úložiska pre vývoj softvéru pomocou systému kontroly verzie VCS. Kontrola verzie bola použitá pri prepojení s pracovným počítačom. Prepojenie školského serveru s pracovným počítačom bolo vytvorené cez reláciu SSH a SFTP protokol.



Obr. 3.7: Vývojový model

3.3.7 Vývojové prostredie a knižnice

Praktická časť práce je naprogramovaná v jazyku python verzie 3.6.8 a 3.7.5. Python je objektovo-orientovaný, interpretovaný jazyk na vysokej úrovni [41] . Tento jazyk bol použitý pre jeho jednoduchosť a rýchlosť a hlavne veľkej podpore pre umelú inteligenciu. Taktiež bola použitá široká škála knižníc dostupných pre tento programovací jazyk.

Graph nets

Je knižnica na vytváranie grafových neurónových sietí pre verejnosť, ktorá spolupracuje s knižnicou Tensorflow a Sonnet. Táto knižnica bola publikovaná 14.júna 2018 za pomoci týmu Deepmind, Google Brain, MIT a University of Edinburgh [17] . Pre praktickú časť bola použitá verziu graph nets 1.0.5.

Tensorflow

Je otvorená platforma pre strojové učenie, ktorá bola vydaná 11. februára 2017 firmou Google brain. Táto platforma dokáže paralelizovať výpočty a tak pracovať na viacerých procesoroch alebo grafických kartách (napríklad rozšírením CUDA). Táto platforma pracuje s viacdimenzionálnymi poliami, ktoré sa hodia pre prácu s neurónovými sieťami [1]. Praktické riešienie používa tensorflow pre prácu s grafickými kartami tensorflow-gpu 1.15.0 a rozšírenie CUDA Toolkit 10.0.

Sonnet

Sonnet je nadstavba pre Tensorflow, ktorá vytvára neurónové siete pre rôzne typy použitia. Sonnet sa zakladá na princípe modulov, častí alebo preddefinovaných neurónových sietí [18]. Prvá verzia Sonnetu bola vydaná 12. júna 2017. Táto práca používa verziu Sonnet 1.35.

Node2Vec

Node2Vec je framework pre reprezentatívne učenie v grafoch. Akýkoľvek graf sa môže nepretržite učiť reprezentácie vrcholov v grafe do n-dimenzionálneho priestoru, ktoré je možné následne použiť no rôzne úlohy strojového učenia [27, 14] .

Keras

Keras je vysokoúrovňové rozhranie pre programovanie aplikácií zaoberajúca sa neurónovými sieťami, ktorá dokaže pracovať nad platformou Tensorflow. Toto rozhranie bolo vytvorené predovšetkým pre rýchlu expreimentáciu s neurónovými sieťami [12] .

3.4 Vyhodnotenie výsledkov

Výstupom učenia grafovej neurónovej siete je jeho naučený model a jednotlivé testované výsledky jednotlivých grafov uložené do textového súboru. Pre urýchlenie bolo potrebné zredukovať veľkosti obrazov 512×512 na časti, grafy o veľkosti 128×128 vrcholov. Tieto výsledky boli následne zrekonštruované pomocou metódy `textToImage` v skriptovacom jazyku Python C .

Pre lepšie popísanie výsledkov je však potrebné použitie exaktných metód pre testovanie segmentácie.

3.4.1 Možnosti testovania segmentácie

Existuje viacero možností ako pristupovať k problému ohodnotenia segmentácie. Táto práca používa volumetrický typ ohodnotenia segmentácie, nakoľko iné typy ohodnotenia segmentácie sa vzťahujú na iné typy segmentačných techník alebo nedostatočne reprezentujú dosiahnuté výsledky (napr. pixelová presnosť). Taktiež hodnotenie rýchlosti je zanedbateľné vzhľadom na to, že sa nejedná o segmentáciu v reálnom čase.

Volumetrické metódy ohodnotenia segmentácie

Nakoľko sú pri testovaní použité boolovské dáta (logická nula alebo jedna), je možné použiť pre ohodnotenie množiny:

- N_{TP} - Počet výsledných a referenčných pixelov na rovnakom mieste, ktoré majú hodnotu logickej jednotky
- N_{TN} - Počet výsledných a referančných pixelov na rovnakom mieste, ktoré majú hodnotu logicej nuly
- N_{FP} - Počet výsledných pixelov, ktoré sú ohodnotené logickou jednotkou ale na referenčnom obraze majú na rovnakom mieste logickú nulu
- N_{FN} - Počet výsledných pixelov, ktoré sú ohodnotené logickou nulou avšak na referenčnom obraze majú na rovnakom mieste logickú jednotku.

Pre spracovanie výsledkov v tejto práci bola použitá podobnosť DICE a Jaccardov index a preto sú popísané iba tieto metódy.

Dice koeficient možno vypočítať ako:

$$DC = \frac{2.N_{TP}}{2.N_{TP} + N_{FP} + N_{FN}}$$

Jaccardov index je možné počítať ako:

$$J = \frac{N_{TP}}{N_{FP} + N_{FN} + N_{TP}}$$

Implementácia testovania a ukladania výstupných dát

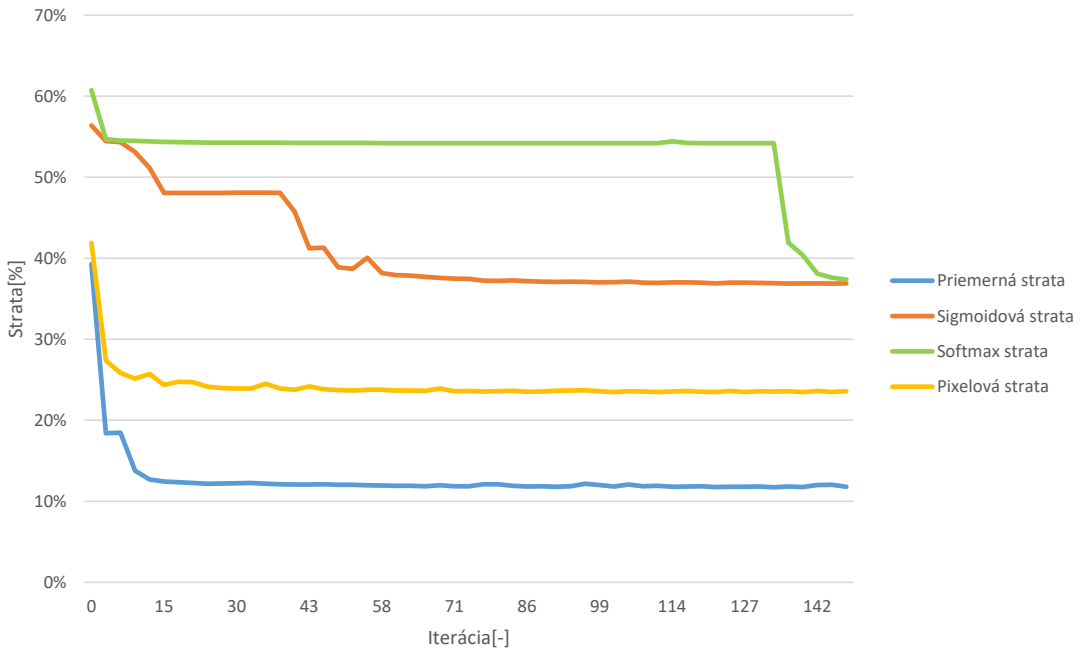
V praktickej časti práci v súbore `accuracy.py` sú implementované metódy pre výpočet koeficientov:

- `compute_dice_accuracy` - Výpočet dice koeficientu
- `compute_accuracy` - Výpočeť pixelovej presnosti
- `compute_jaccard_accuracy` - Výpočet Jaccardovho indexu.

Taktiež sú v tomto súbore metódy `compute_X_accuracyTest`, ktoré vyhodnocujú koeficienty pre testované obrazy a taktiež ukladajú výstupné vrcholy do textových súborov `iterácia_index.txt`. Implementácia výpočtu `compute_dice_accuracyTest` D .

3.4.2 Dosiahnuté výsledky

Pre dosiahnutie najlepších výsledkov je potrebné správne udanie hyperparametrov. Hyperparametre grafovej neurónovej siete sa nachádzali v samostatnom súbore `initialization.py`. Táto práca testovala rozličné metódy stratovej funkcie pre správny výber nejlepšej z nich:



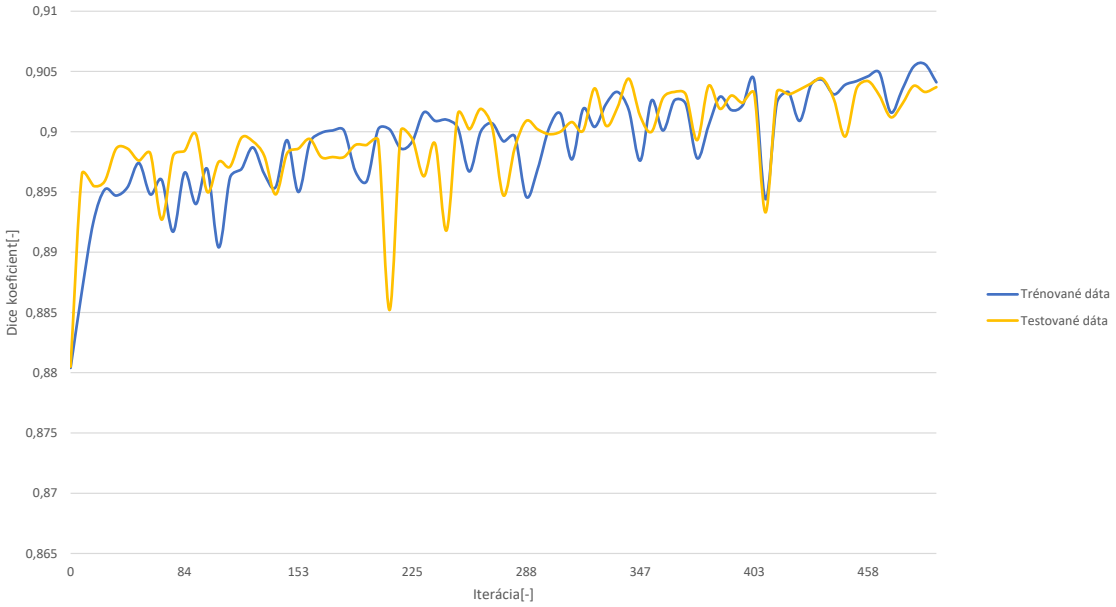
Obr. 3.8: Graf závislosti straty od iterácie pre rôzne stratové funkcie

Pre optimálne stratové funkcie boli považované sigmoidová a softmax stratová funkcia nakoľko boli používané aj v predchádzajúcich prácach [26] . Ďalší hyperparameter, na ktorý sa práca zameriavala bol počet krokov prenášania správ a topológia siete. Počet krokov prenášania správ udáva veľkosť okolia, od ktorých vrchol získava informácie. Ide o podobný proces konvolúcie, avšak je možné použiť variabilnú veľkosť okolia a taktiež váhu týchto informácií. Topológia viacvrstvového perceptronu bola testovaná pre zlepšenie presnosti sieťe a vzhľadom na dlhý výpočetný čas sa nastavil počet iterácii na 150.

Topológia/Počet prenášaní správ	4	6	8
3x32	89,65	89,59	89,83
3x64	89,24	89,85	88,28
3x128	88,94	87,55	-

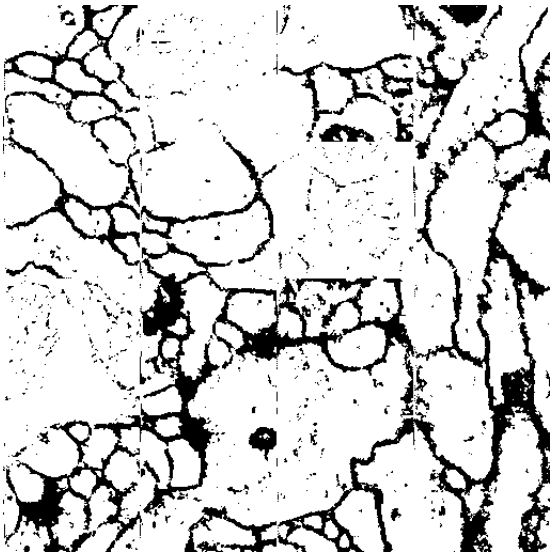
Tab. 3.2: Úspešnosť dice koeficientu v percentách podľa výberu topológie a počtu prenášaní správ

Kvôli obmedzenej videopamäti grafickej karty sa nepodarilo otestovať úspešnosť topológie 128×3 pri 8-násobnom prenášaní správ. Pre správne ohodnotenie výsledkov je potrebné použiť niektorú z možností testovania segmentácie. Pre ohodnotenie trénovaných a testovaných bol použitý graf závislosti DICE koeficientu od iterácie.

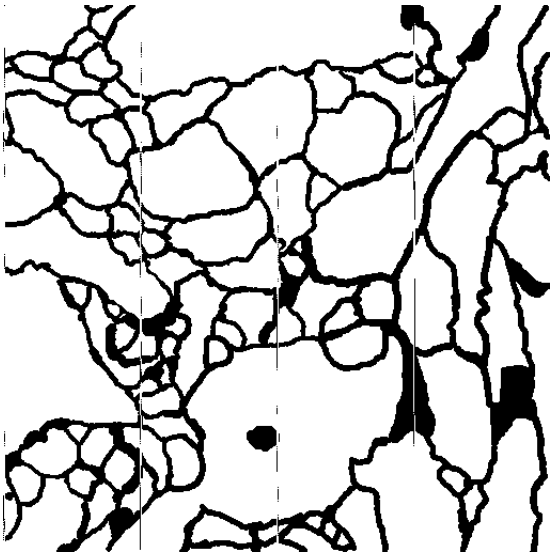


Obr. 3.9: Graf závislosti Dice koeficientu na iterácii trénovania

Tento výsledok bol spätne zrekonštruovaný z grafov na výsledný obraz segmentácie



Obr. 3.10: Výsledná segmentácia obrazu



Obr. 3.11: Referenčná segmentácia obrazu

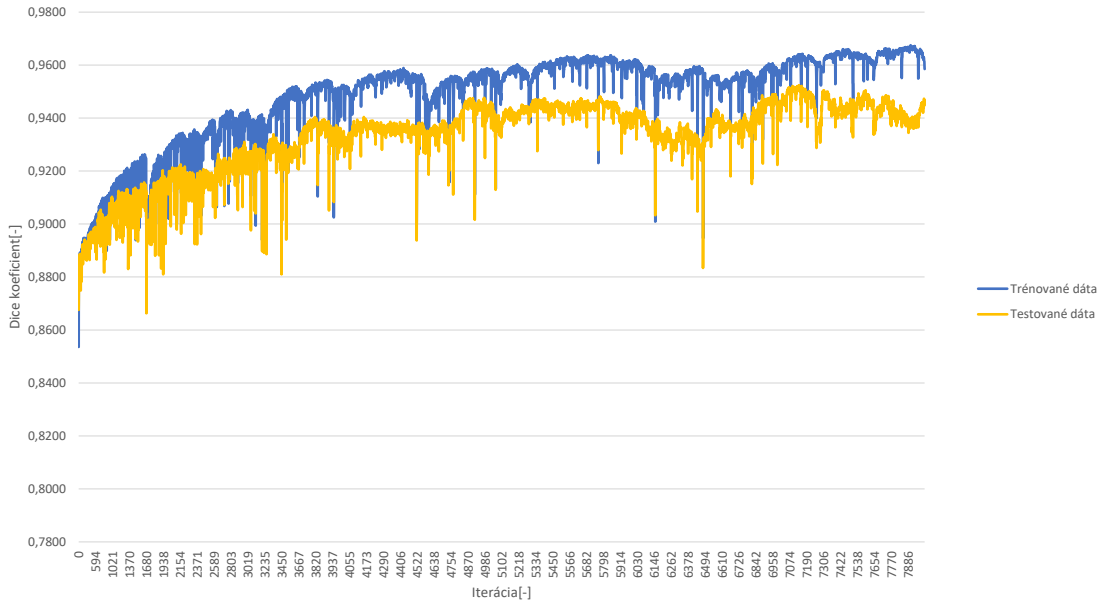
Zvýšenie počtu iterácií

Pre zlepšenie výsledkov neurónová sieť pokračovala v učení po 8000 iterácií za dobu dvoch týždňov. Najlepší výsledok tejto siete bol 96, 5% pre trénovaciu množinu a 95%

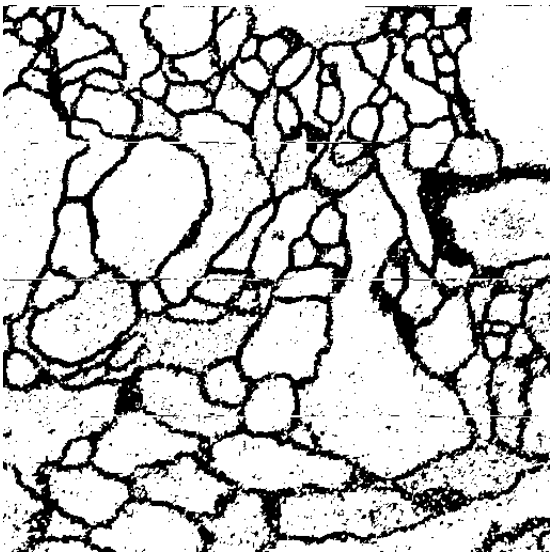
pre testovaciu. Neurónová sieť bola nastavená tak aby existoval najvačší potenciál pre zlepšie:

- počet prenášaní správ: 6,
- rýchlosť učenia: 5×10^{-3},
- epsilon(spomaľovanie učenia): 2×10^{-10}.

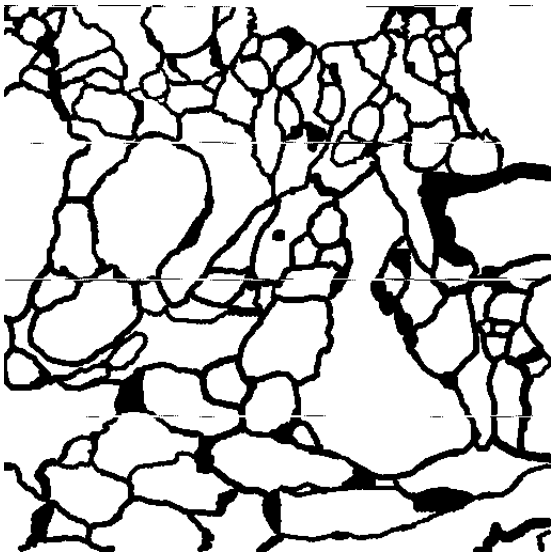
Výsledky boli zrekonštruované do grafu a obrazu jedného z testovacích dát:



Obr. 3.12: Graf závislosti Dice koeficientu na iterácii trénovania pre 8000 iterácií



Obr. 3.13: Výsledná segmentácia obrazu



Obr. 3.14: Referenčná segmentácia obrazu

Na základe obrázkov 3.13 a 3.12 je možné vyhodnotiť že rekurentná grafová neurónová sieť s prenášaním správ má porovnateľné výsledky oproti neurónovej sieti U-Net [33]. Pre zlepšenie tejto siete práca doporučuje použiť ako postspracovanie štatistické filtre ako napríklad mediánový filter pre zamedzenie šumu, ktorý sa nachádza v obraze. Taktiež prichádza na radu otázka preučenia siete pri daľšej iterácií trénovania.

Keras

Použitie grafovej konvolučnej siete poskytuje spoľahlivé filtre pre grafové vstupné dáta. Nakoľko táto sieť spadá pod grafové konvolučné siete založené na spektre nie je možné meniť veľkosť sieťe. Druhým problémom tejto sieťe je veľkosť vstupných dát nakoľko matica susednosti dosahuje kvadratickú veľkosť v závislosti od počtu vrcholov. V rámci praktickej časti bola vytvorená implementácia grafovej konvolučnej neurónovej sieti E.

Nakoľko pri vytváraní vstupných dát pre učenie bola videopamäť procesora zaplnená, nebola táto implementácia úspešná.

Použitie autoenkodéru node2vec na predspracovanie

Na problém segmentácie sa je možné pozerať rozličnými spôsobmi. Pre predspracovanie vstupných dát pomocou autoenkodéru node2vec boli použité dva pohľady na problém.

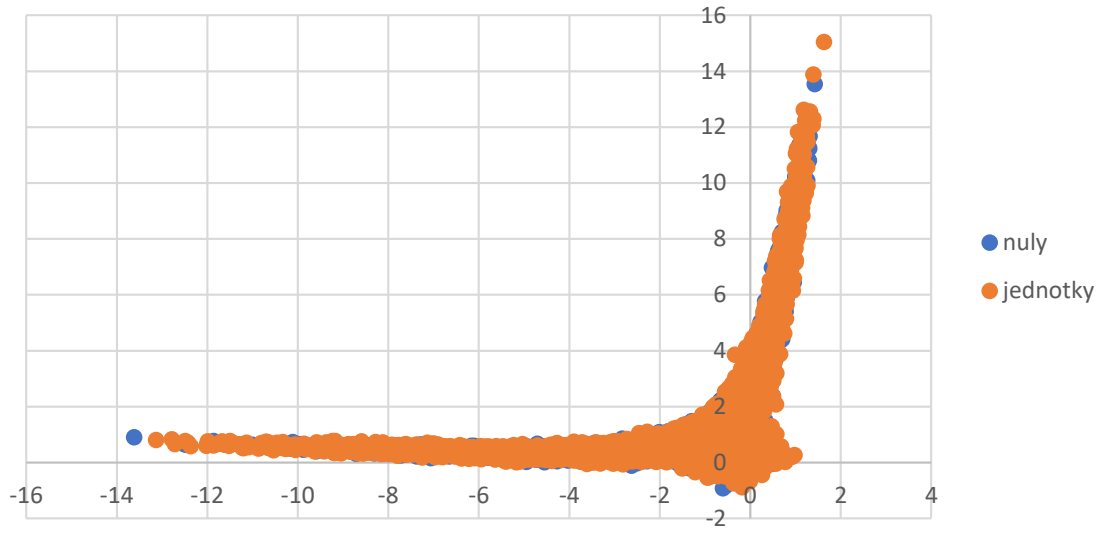
Prenos do N-rozmerného priestoru Node2vec autoenkodér umožňuje premiestniť vrcholy grafu do N-rozmerného priestoru, v ktorom je možno lepšie použiť strojové učenie. V implementácií mali byť súradnice N-dimentzionálneho priestoru použité ako parametre vrcholov pre zlepšenie segmentácie.

V rámci praktickej časti bol vyskúšaný 2, 16 a 128 dimenzionálny priestor. V rámci odskúšania tejto realizácie bol použitý script `segmentation.py` na ukladanie jednotlivých indexov do textového súboru:

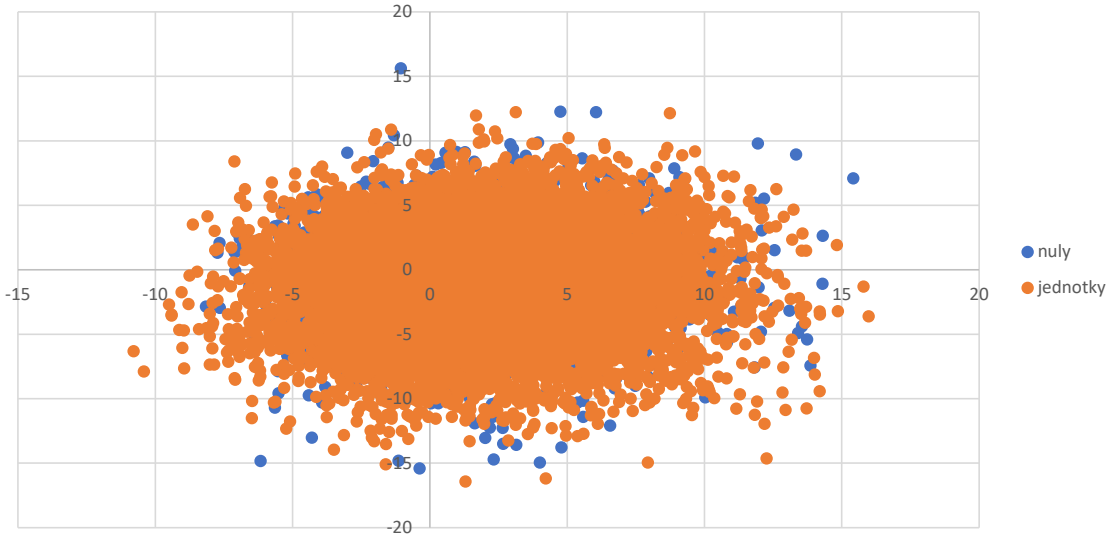
```
input_graphs, target_graphs  = generate_networkx_graphs(rand, batch_size_tr,
num_nodes_min_max_tr, theta, True)
node2vecs = []
models = []
#Počet dimenzií priestoru = 16
#Dĺžka prechádzania grafom z~vrcholu = 50
#Počet prechádzaní grafom z~vrcholu = 20
#Veľkosť okna pre jeden vrchol = 100
for index in range(len(input_graphs)):
    node2vecs.append(Node2Vec(input_graphs[index], dimensions=16,
    walk_length=50,num_walks=20, workers=4))
```

```
models.append(node2vecs[index].fit(window=100,
min_count=1, batch_words=4))
models[index].wv.save_word2vec_format("./results/"+
str(index)+"embeddings.emb")
```

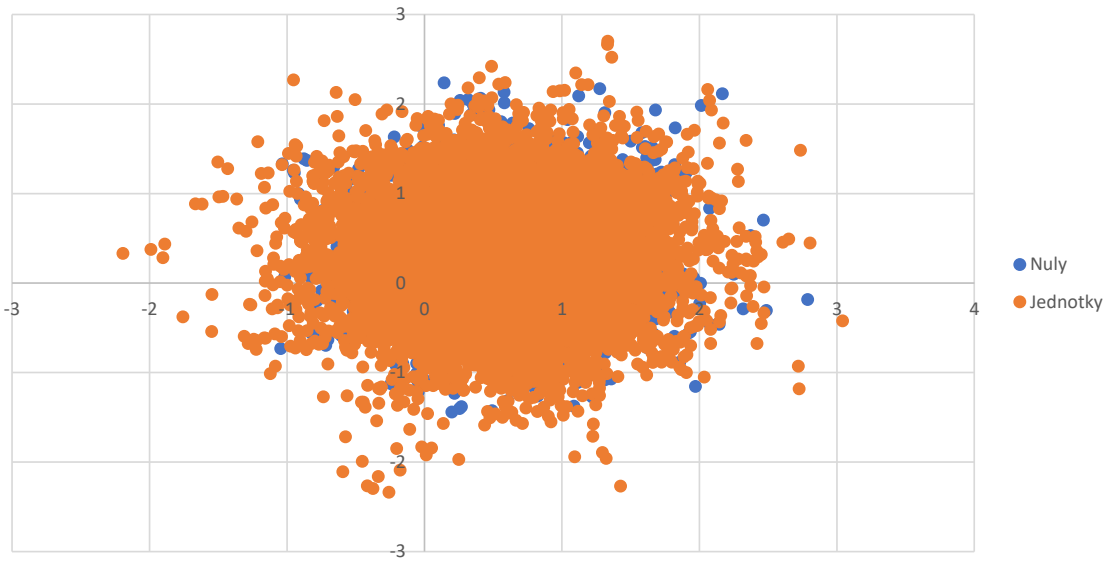
Následne boli vybrané dimenzie s najvačšou varianciou v dvojrozmernom priestore, ktoré boli vykreslené do grafu:



Obr. 3.15: node2vec v dvojrozmernom priestore



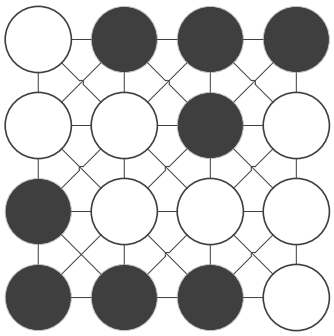
Obr. 3.16: node2vec v 16-rozmernom priestore



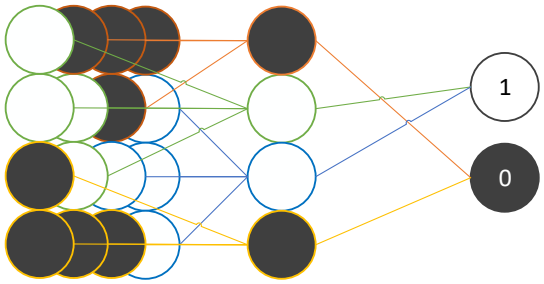
Obr. 3.17: node2vec v 128-rozmernom priestore

Na základe výsledkov grafov 3.15,3.16 a 3.17 táto implementácia nebola použitá.

Stromová štruktúra Problém segmentácie obrazu prenesnenej do grafovej štruktúry je možné popísať ako problém určenia kostry grafu s n segmentami. Určením kostry vzniká stromová štruktúra:



Obr. 3.18: Príklad štruktúry grafu



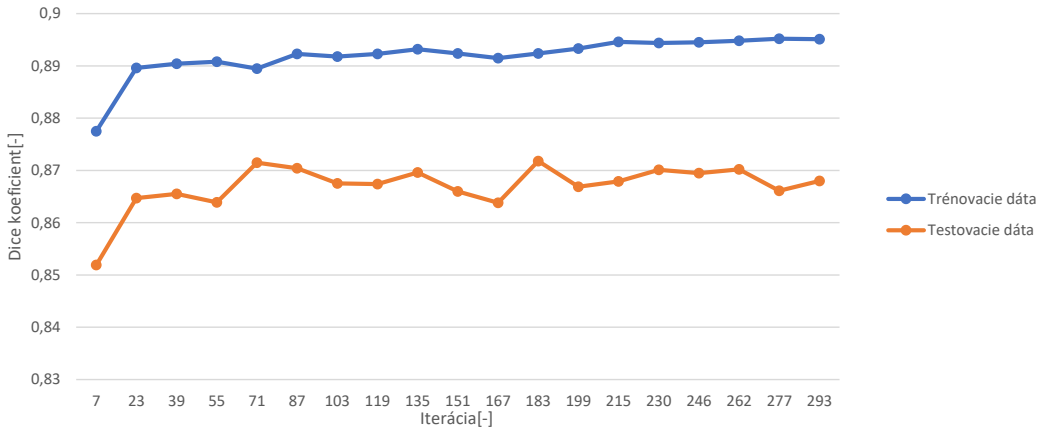
Obr. 3.19: Stromová štruktúra z grafu

Vytvorenie stromovej štruktúry náráža na rôzne problémy. V prvom rade je potrebné vedieť, akým spôsobom budú určené vrcholy v daľších stupňoch grafu. Nutnosťou týchto vrcholov je mať jednoznačné určenie riešenia. Druhým problémom je správne pridelenie hrán k daným vrcholom.

Vysledkom tejto štruktúry je jednoduchšie prenášanie správ medzi podobnými vrcholmi a tým zaručenie presnejšej segmentácie.

V rámci praktickej časti boli vytvorené dve metódy vytvárania stromovej štruktúry.

Stromová štruktúra s podvzorkovaním Prvou je jednoduché podvzorkovanie grafu s prepojením s pôvodným grafom. Výsledky boli zhodnotené pomocou grafu:



Obr. 3.20: Graf závislosti Dice koeficientu na iterácii trénovania pre jednoduchú stromovú štruktúru

Tento typ riešenia nezaručuje správne pridelenie hrán medzi vrcholmi nakoľko podvzorkované vrcholy sú priradené k vrcholom okolia. Druhou nevýhodou tohto riešenia prináša veľký počet podvzorkovaných vrcholov a tým rýchlejšie zaplnenie videopamäťe.

Stromová štruktúra s node2vec Nakoľko podvzorkovanie s prepojením čelilo pamäťovým problémom a neuvažovalo topológiu siete bolo v práci vyvinuté použitie autoenkodéru node2vec. Tento systém vyberie určený počet náhodných vrcholov podľa parametru `number_of_node2vec`. Následne sa zakóduje vstupný graf do node2vec priestoru. Náhodne vybrané vrcholy sú určené do ďalšieho stupňa grafu. Pomocou node2vec sú vybrané najpodobnejšie vrcholy k vrcholom vstupného grafu, ktoré sú následne spojené hranou. Hrany sú nastavené na kvadratickú differenciu pixelových hodnôt:

$$W = 1 - (p_i - p_j)^2$$

kde p_i a p_j sú hodnoty pichelov na indexoch i a j. Tento systém je oproti podvzorkovaniu grafu pamäťovo menej náročný, nakoľko nie je potrebný veľký počet nových vrcholov. Problémom metódy je pomalé zakódovanie grafu avšak rýchlosť je možno upraviť pomocou nastavení autoenkodéru:

- `number_of_node2vec` - počet náhodne vybraných vrcholov
- `similarity_threshold` - prah podobnosti vrcholov
- `node2vec_dimensions` - počet dimenzí node2vec priestoru
- `node2vec_walk_length` - dĺžka prechádzania grafom

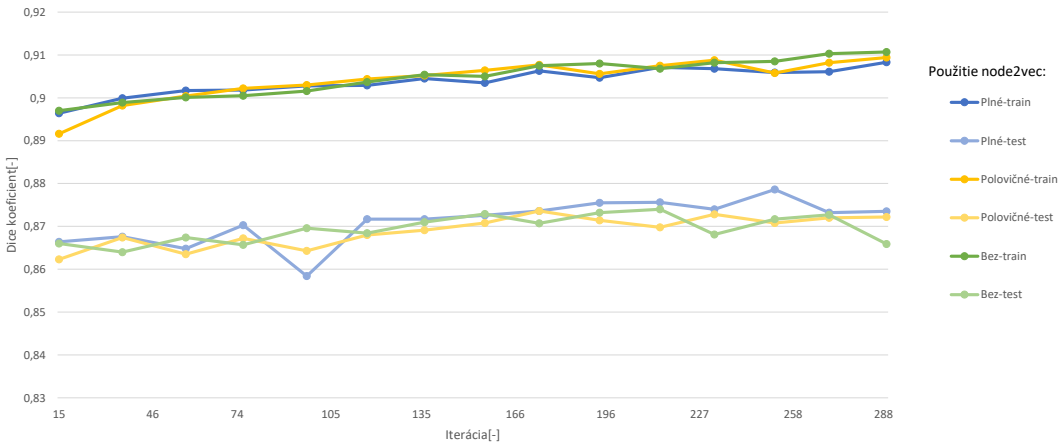
- `node2vec_num_walks` - počet prechádzaní grafom z každého vrcholu

Podľa tohto modelu bol v praktickej časti vytvorený súbor `definitionsNode2Vec` s upravenou metódou `generate_graphs` F .

V rámci používania node2vec vznikla otázka či je správne mapovať testovacie dáta do node2vec euklidovského priestoru ako predspracovanie vstupných dát. Používanie naučenej neurónovej siete je preto pomalšie, nakoľko je potrebné pri každom obraze mapovať vstupné dáta. Pre porovnanie boli vytvorené tri typy generovaní grafu:

- Bez použitia node2vec
- S použitím node2vec iba pre trénovacie dáta
- S použitím node2vec pre trénovacie a testovacie dáta

Topológia neurónovej siete je 3×32 viacvrstvových perceptronov. Táto topológia bola zvolená pre porovnanie osemsmerového prepájania grafu so štvorsmerovým, použitým v 3.2. Ostatné nastavenia zostali rovnaké. Počet náhodne vybraných vrcholov použitých pre vytvorenie stromovej štruktúry node2vec je 2000. Výsledky učenia sú sumarizované v grafe:



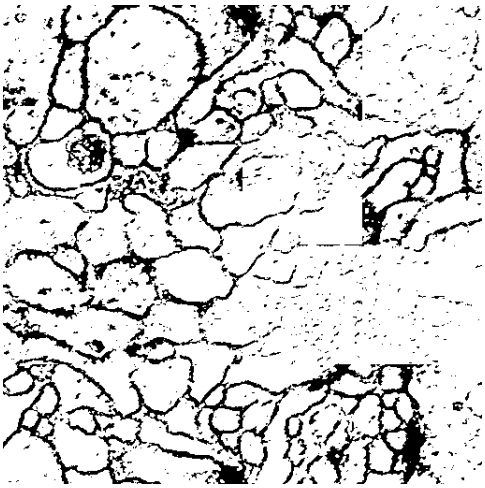
Obr. 3.21: Graf závislosti Dice koeficientu na iterácii trénovania pre node2vec stromovú štruktúru

Prvým typom generovania grafu je plné použitie stromovej štruktúry node2vec pre trénovacie a testovacie dáta. Druhým typom je polovičné použitie node2vec, ktoré používa túto štruktúru iba pre trénovacie dáta. Posledným typom je použitie grafovej štruktúry bez použitia node2vec. Pre každý typ je zobrazený vysledok dice koeficientu pre trénovacie a testovacie dáta.

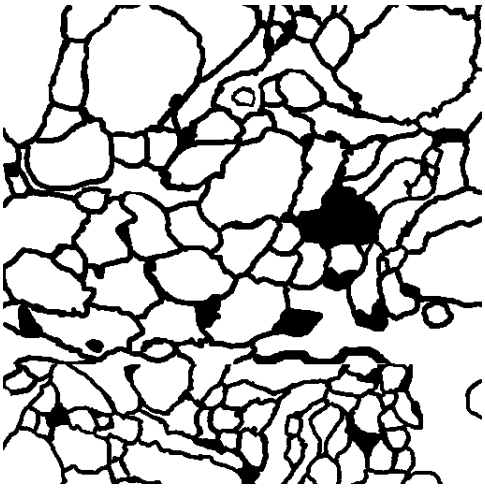
Z výsledkov je možné zhodnotiť, že nedošlo k zlepšeniu učenia použitím node2vec štruktúry. Použitie node2vec závisí hlavne na štruktúre grafu, ktorá sa v prípade segmentácií obrazov drasticky nemení.

Oproti štvorsmerovému prepájaniu vrcholov 3.2 je možné pozorovať už pri 150 iteráciách zlepšenie segmentácie o jedno percento. To má však za cenu vačšiu výpočtovú zložitosť a vačšie nároky na pamäť. Celkovo však dochádza k zlepšeniu a preto je výhodné použiť osemsmerové prepájanie vrcholov.

Okrem grafovej podoby boli výsledky za plného použitia node2vec reprezentované rekonštrukciou do obrazu:



Obr. 3.22: Výsledná segmentácia za použitia node2vec



Obr. 3.23: Referenčná segmentácia obrazu

Tu je možné vydieť zmenšovanie zašumenia vo výslednej segmentácií avšak za cenu zhlukovania chybných pixelov. Oproti šumovým chybám je táto chyba závažnejšia z hľadiska opravy. Pre zlepšenie grafových neurónových sietí používajúcich grafové autoenkodéry je preto potrebné daľší výskum.

4 Záver

Táto semestrálna práca sa zaoberala problematikou segmentácie nervovej štruktúry za použitia grafových neurónových sietí. Ich popisu sa bližšie venovala druhá kapitola. Nakoľko je typ rekurentných grafových neurónových sietí pod prebiehajúcim výskumom univerzít, môžeme v budúcnosti očakávať výrazné zlepšenie ich výkonu a využitia.

V rámci prakticej časti bola vytvorená grafová neurónová sieť s možnosťou spracovania obrazových dát pre segmentáciu. Táto sieť bola náročná na videopamäť grafických kariet, avšak podľa rýchlosti učenia sa zaraďovala medzi rýchlejšie metódy. To bolo zapríčinené jednoduchou topológiou siete a taktiež použitím novej technológie Tesla jadier na trénovanie. Výsledky natrénovanej neurónovej siete na EM segmentačnej výzve [11] dosahovali Dice koeficient pre testovacie dáta $0,95$, pri zvýšení počtu iterácií na 8000.

Z nameraných výsledkov práce vyplýva, že vytvorená grafová neurónová sieť dosahuje porovnateľné výsledky oproti U-net sieti, ktorá mala pixelovú presnosť 0,9389.

Okrem zvýšenia počtu iterácií bolo otestované aj použitie predspracovania vstupných grafov pomocou grafového autoenkodéru node2vec alebo podvzorkovaním vstupných dát. Výsledky pri testovaní dát vyvrátili zlepšenie oproti jednoduchému modelu grafu bez predspracovania.

Hlavným prínosom tejto práce je vytvorenie rozhrania v jazyku Python pre segmentáciu obrazu pomocou grafových neurónových sietí. Toto prostredie je ľahko nadefinovateľné a používa voľne dostupnú grafovú neurónovú sieť `graph_nets`. Práca tiež opisuje rôzne možnosti zakódovania a dekódovania grafov pre spracovanie neurónovou sieťou. Tieto možnosti predspracovania sú implementované v projekte a výsledky sú vyhodnotené v praktickej časti. Práca navyše objasňuje metódu prenášania správ pri trénovaní, ktorá je typická pre grafové neurónové siete. Vedľajším prínosom práce je všeobecný prehľad riešenia problému segmentácie.

Pre zlepšenie tejto práce je potrebné použiť zložitejší typ neurónovej siete, napríklad konvolučnú grafovú neurónovú sieť. V rámci zlepšenia bola implementovaná grafová neurónová siet convNet. Rovnako je možné použitie zlučovania grafov do stromovej štruktúry podľa podobnosti, čím vznikne model podobný U-Netu, vďaka ktorému sa zníži potrebná videopamäť a zrýchli trénovanie siete.

Literatúra

[1] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org.
URL `https://www.tensorflow.org/`

[2] Al-Amri, S. S.; Kalyankar, N. V.; aj.: Image segmentation by using threshold techniques. *arXiv preprint arXiv:1005.4020*, 2010.

[3] Alvarez, J. M.; Lopez, A.; Baldrich, R.: Illuminant-invariant model-based road segmentation. In *2008 IEEE Intelligent Vehicles Symposium*, IEEE, 2008, s. 1175–1180.

[4] Baby Resma, K.; Nair, M.: Multilevel thresholding for image segmentation using Krill Herd Optimization algorithm. *Journal of King Saud University - Computer and Information Sciences*, 2018: str. lt;xocs:firstpage xmlns:xocs=quot;quot;/gt;, ISSN 13191578.

[5] Bao, P.; Zhang, L.; Wu, X.: Canny edge detection enhancement by scale multiplication. *IEEE transactions on pattern analysis and machine intelligence*, ročník 27, č. 9, 2005: s. 1485–1490.

[6] Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; aj.: Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[7] Bender, E. A.; Williamson, S. G.: *Lists, Decisions and Graphs*. S. Gill Williamson, 2010.

[8] Bhanu, B.; Lee, S.; Ming, J.: Adaptive image segmentation using a genetic algorithm. *IEEE Transactions on systems, man, and cybernetics*, ročník 25, č. 12, 1995: s. 1543–1567.

[9] Bouma, G.: Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, 2009: s. 31–40.

[10] Cao, S.; Lu, W.; Xu, Q.: Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, s. 1145–1152.

[11] Cardona, A.; Saalfeld, S.; Preibisch, S.; aj.: An Integrated Micro- and Macroarchitectural Analysis of the Drosophila Brain by Computer-Assisted Serial Section Electron Microscopy. *PLOS Biology*, ročník 8, č. 10, 10 2010: s. 1–17, doi:10.1371/journal.pbio.1000502.
URL `https://doi.org/10.1371/journal.pbio.1000502`

[12] Chollet, F.: Keras: The Python Deep Learning library.
URL https://keras.io/

[13] Chuang, K.-S.; Tzeng, H.-L.; Chen, S.; aj.: Fuzzy c-means clustering with spatial information for image segmentation. *computerized medical imaging and graphics*, ročník 30, č. 1, 2006: s. 9–15.

[14] Cohen, E.: eliorc/node2vec. Feb 2020.
URL https://github.com/eliorc/node2vec

[15] Cormen, T.; Leiserson, C.; Rivest, R.; aj.: Introduction to algorithms (ISBN: 0-262-03293-7, 9780262032933). 2001.

[16] Correa-Tome, F. E.; Sanchez-Yanez, R. E.: Integral split-and-merge methodology for real-time image segmentation. *Journal of Electronic Imaging*, ročník 24, č. 1, 2015: s. 013007–013007, ISSN 1017-9909.

[17] DeepMind: Graph Nets library. Nov 2019.
URL https://deepmind.com/research/open-source/graph-nets-library

[18] DeepMind: Sonnet. Nov 2019.
URL https://deepmind.com/research/open-source/deepmind-sonnet

[19] Dunn, J. C.: A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, ročník 3, 1973: s. 32–57.

[20] Espindola, G.; Câmara, G.; Reis, I.; aj.: Parameter selection for region-growing image segmentation algorithms using spatial autocorrelation. *International Journal of Remote Sensing*, ročník 27, č. 14, 2006: s. 3035–3040.

[21] Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; aj.: Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, s. 1263–1272.

[22] GitHub, I.: The world's leading software development platform. https://github.com/, accessed: 2020-04-10.

[23] Gonzalez, R. C.; Woods, R. E.: Digital Image Processing Addison-Wesley. *Reading, Ma*, ročník 2, 1992.

[24] Google, I.: Google Collab. https://colab.research.google.com/notebooks/welcome.ipynb, accessed: 2020-04-10.

[25] Goyal, P.; Ferrara, E.: Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, ročník 151, 2018: s. 78–94.

[26] Grover, A.; Leskovec, J.: node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2016, s. 855–864.

[27] Grover, A.; Leskovec, J.: Scalable Feature Learning for Networks. 2016.
URL `https://snap.stanford.edu/node2vec/`

[28] Gutzeit, E.; Scheel, C.; Dolereit, T.; aj.: Contour based Split and Merge segmentation and pre-classification of zooplankton in very large images. In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, ročník 1, SCITEPRESS, 2014, ISBN 9789897581335, s. 417–424.

[29] Hagenbuchner, M.; Sperduti, A.; Tsoi, A. C.: A self-organizing map for adaptive processing of structured data. *IEEE transactions on Neural Networks*, ročník 14, č. 3, 2003: s. 491–505.

[30] Hamilton, W. L.; Ying, R.; Leskovec, J.: Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[31] Ibrahim, D.: An overview of soft computing. *Procedia Computer Science*, ročník 102, 2016: s. 34–38.

[32] Mehlhorn, K.; Naher, S.; Näher, S.: *LEDA: a platform for combinatorial and geometric computing*. Cambridge university press, 1999.

[33] Ronneberger, O.; Fischer, P.; Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, s. 234–241.

[34] Rusek, K.; Suárez-Varela, J.; Mestres, A.; aj.: Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN. In *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019, s. 140–151.

[35] Scarselli, F.; Gori, M.; Tsoi, A. C.; aj.: The graph neural network model. *IEEE Transactions on Neural Networks*, ročník 20, č. 1, 2008: s. 61–80.

[36] Shapiro Linda, G.: *Computer vision*. New Jersey: Prentice-Hall, vyd. 1 vydání, 2001, ISBN 0-13-030796-3.

[37] Singh, D. K.; Patgiri, R.: Big graph: tools, techniques, issues, challenges and future directions. In *Sixth International Conference on Advances in Computing and Information Technology (ACITY 2016)*, 2016, s. 119–128.

[38] Sperduti, A.; Starita, A.: Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, ročník 8, č. 3, 1997: s. 714–735.

[39] Suhas, N.: Image Segmentation using Region-Growing. 2017.
URL `github.com/suhas-nithyanand/Image-Segmentation-using-Region-Growing`

[40] Vala, H. J.; Baxi, A.: A review on Otsu image segmentation algorithm. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, ročník 2, č. 2, 2013: s. 387–389.

[41] Van Rossum, G.; Drake, F. L.: *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN 1441412697.

[42] Wang, X.; Girshick, R.; Gupta, A.; aj.: Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, s. 7794–7803.

[43] Wu, Z.; Pan, S.; Chen, F.; aj.: A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[44] Yan, S.; Xiong, Y.; Lin, D.: Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[45] Zaitoun, N. M.; Aqel, M. J.: Survey on Image Segmentation Techniques. *Procedia Computer Science*, ročník 65, 2015: s. 797–806, ISSN 1877-0509.

[46] Zhou, J.; Cui, G.; Zhang, Z.; aj.: Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Zoznam symbolov, veličín a skratiek

GNN	Graph neural network
RGNN	Recurrent graph neural network
LoG	Laplacean of gaussian
DNGR	Deep neural networks for learning graph representations
HOPE	High order proximity embedding
MPNN	Message passing neural networks
ReLu	Rectified Linear unit
VCS	Version Control system
SSH	Secure shell protocol
SFTP	Secure file transfer protocol
TP	True positive
TN	True negative
FP	False positive
FN	False negative

Zoznam príloh

A Štruktúra projektu

- /
  - dataset
    - images — Obrazové snímky
      - dir
        - 0.png
        - 1.png
        - ...
    - labels — Referenčné snímky
      - dir
        - 0.png
        - 1.png
        - ...
  - encodings — Mapovanie node2vec
    - 0-0.emb
    - 0-0.model
    - 0-1.emb
    - 0-1.model
    - ...
  - results
    - textToImage.py — Skript na vytvorenie obrázku z textového výstupu
  - graph_nets — Knižnica graph_nets
  - accuracy.py — Metódy pre výpočet presnosti segmentácie
  - definitionsV4.py — Definície pre vytváranie grafov
  - definitionsNode2Vec.py — Definície pre vytváranie grafov s použitím node2vec
  - graphPlotter.py — Trieda na vykresľovanie grafov
  - initialization.py — Súbor s inicializačnými parametrami
  - segmentation.py — Štartovací script

B Metóda generate_graphs

```
def generate_graphs(img_path, solution_path,region, j, placeholder=False):
#Čítanie dát zo zložky
img = imageio.imread(img_path)
img_solution = imageio.imread(solution_path)

#Rozdelenie do polí o blokoch 128x128 a 256x256
img = np.array(img)
img_solution = np.array(img_solution)
img_subsample = img[::2, ::2]
img_subsample = slice_blocks(img_subsample, 128, 128)
img_solution_subsample = img_solution[::2, ::2]
img_solution_subsample = slice_blocks(img_solution_subsample
, 128, 128)
images = slice_blocks(img, 256, 256)
images_solutions = slice_blocks(img_solution, 256, 256)

graphs = [];
#Iterácia cez polia
for i in range(len(images)):
    index = 0 * 4096 - 1
    graph = nx.Graph()
    for row in range(0, 256):
        for column in range(0, 256):
            index += 1
            node_solution = 0
            value = images[i][row][column] / 255
            if images_solutions[i][row][column] == 255:
                node_solution = 1
            # pridanie uzlu
            graph.add_node(index, value=value, solution=node_solution)
    index = -1
    for row_e in range(0, 256):
        for col_e in range(0, 256):
            # pridanie hrán
            index += 1
            add_edges_to_graph(index, graph,images[i], row_e, col_e,
            region, images_solutions[i])
    #Prechádzanie podvzorkovanými poľami
    for row in range(0, 128):
        for column in range(0, 128):
            index += 1
            value = img_subsample[i][row][column] / 255
            solution = 0
            if img_solution_subsample[i][row][column] == 255:
                solution = 1
```

```
            #pridanie uzlu
            graph.add_node(index, value=value, solution=solution)
            #pridanie hrán
            for x in range(0, 2):
                for y in range(0, 2):
                    edge_index = (row + x) * 2 + column + y
                    solution = 0
                    if images_solutions[i][row + x][column + y] ==
                    img_solution_subsample[i][row][column]:
                        solution = 1
                    graph.add_edge(index, edge_index, weight=1,
                    solution=solution)
    graphs.append(graph)
return graphs
```

C Metóda text_to_image

```
def textToImage(iteration):
image1 = np.zeros((512, 512), dtype="uint8")
image2 = np.zeros((512, 512), dtype="uint8")
imageNumber=4;
for x in range(4):
    for y in range(4):
        target = open("target_" + str(iteration) + "_" +str(imageNumber) + ".txt")
        result = open(str(iteration) + "_" + str(imageNumber) + ".txt")
        imageNumber=imageNumber+1
        for i in range(128):
            for j in range(128):
                color=target.readline()
                if color == "1\n":
                    image1[x * 128 + i][y * 128 + j] = 255
                else:
                    image1[x * 128 + i][y * 128 + j] = 0
                color=result.readline()
                if color == "1\n":
                    image2[x * 128 + i][y * 128 + j] = 255
                else:
                    image2[x * 128 + i][y * 128 + j] = 0
im1=Image.fromarray(image1)
im1.save('target.png')
im2=Image.fromarray(image2)
im2.save('result.png')
```

D Metóda compute_dice_accuracyTest

```
def compute_dice_accuracyTest(target, output, iteration, index, test_values
, last_iteration,iteration_count, use_nodes=True, use_edges=False):
    if not use_nodes and not use_edges:
    raise ValueError("Nodes or edges (or both) must be used")
#Vytvára štrukturované dáta z obrazových dát do dvojíc vrcholov a hrán
tdds = utils_np.graphs_tuple_to_data_dicts(target)
odds = utils_np.graphs_tuple_to_data_dicts(output)
cs = []
ss = []
#Prechádzanie dátami
for td, od in zip(tdds, odds):
    xn = np.argmax(td["nodes"], axis=-1)
    yn = np.argmax(od["nodes"], axis=-1)
    xe = np.argmax(td["edges"], axis=-1)
    ye = np.argmax(od["edges"], axis=-1)
    c = []
    solved_pixels = []
    #ukladanie dát do zložky results
    if iteration_count<int(iteration/log_every_outputs):
        for i in range(len(test_values)):
            with open('results/'+str(last_iteration) +
            '_' + str(index) + '.txt', 'w+') as f:
                for value in yn:
                    f.write("%s\n" % value)
                f.close()
            with open('results/'+'target_' + str(last_iteration) +
            '_' + str(index) + '.txt', 'w+') as x:
                for value in xn:
                    x.write("%s\n" % value)

#Ukladanie jednotlivých koeficientov a výpočet priemernej hodnoty
    if use_nodes:
        c.append(float(1 - dice(xn, yn)))
        solved_pixels.append(xn == yn)
    if use_edges:
        c.append(float(1 - dice(xe, ye)))
        solved_pixels.append(xe == ye)
    s = np.all(solved_pixels)
    cs.append(c)
    ss.append(s)
mean_koeficient = np.mean(cs)
solved = np.mean(np.stack(ss))
return mean_koeficient, solved
```

E Implementácia grafovej konvolučnej siete

```
    #Vytvorenie grafov
input_graphs, target_graphs  = generate_networkx_graphs(rand, batch_size_tr,
num_nodes_min_max_tr, theta, True)
#Vytvorenie matíc susednosti a hlavnej matice
A=networkx.convert_matrix.to_numpy_array(input_graphs);
Y= networkx.convert_matrix.to_numpy_array(target_graphs);
X= networkx.attr_matrix(input_graphs,"weight","value");
print(A)
print(Y)

#Preprocessing
SYM_NORM = True
A_norm = preprocess_adj_numpy(A, SYM_NORM)
num_filters = 2
graph_conv_filters = np.concatenate([A_norm, np.matmul(A_norm, A_norm)], axis=0)
graph_conv_filters = K.constant(graph_conv_filters)

#Sekvenčný model Grafovej konvolučnej siete
model = Sequential()
model.add(GraphCNN(16, num_filters, graph_conv_filters,
input_shape=(X.shape[1],), activation='elu', kernel_regularizer=l2(5e-4)))
model.add(Dropout(0.2))
model.add(GraphCNN(Y.shape[1], num_filters, graph_conv_filters,
activation='elu', kernel_regularizer=l2(5e-4)))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.01),
metrics=['acc'])
```

F Implementácia pre node2vec

```
#Výber náhodných indexov
rand_indexes = random.sample(range(image_size*image_size-1), number_of_node2vec)
#Vytvorenie odkazov pre načítanie dát
my_emb = Path("encodings/"+str(image_index)+"-"+str(i)+".emb")
my_model = Path("encodings/" + str(image_index)+ "-" + str(i) + ".model")
if my_emb.is_file() and my_model.is_file():
    #Načítanie mapovaných dát
    model = Word2Vec.load("./encodings/"+str(image_index)+"-"+str(i)
    +".model")
    model.wv.load_word2vec_format("./encodings/"+str(image_index)+"-"
    +str(i)+".emb")
else:
    #Mapovanie node2vec a ukladanie do adresára
    node2vec = Node2Vec(graph, dimensions=node2vec_dimensions,
    walk_length=node2vec_walk_length,
                        num_walks=node2vec_num_walks, workers=4)
    model = node2vec.fit(window=64, min_count=1, batch_words=4)
    model.wv.save_word2vec_format("./encodings/"+str(image_index)
    +"-"+str(i)+".emb")
    model.save("./encodings/"+str(image_index)+"-"+str(i)+".model")

for rand_index in rand_indexes:
    #Výpočet vrcholov
    index += 1

    edge_indexes = model.wv.most_similar(str(rand_index))

    row = int(rand_index / image_size)
    column = int(rand_index) - row*image_size

    value = images[i][row][column] / 255

    node_solution = 0
    value = images[i][row][column] / 255
    if images_solutions[i][row][column] == 255:
        node_solution = 1
    #Pridanie vrcholu do grafu
    graph.add_node(index, value=value, solution=node_solution)
    graph.add_edge(index, rand_index, weight=1, solution=1)
    for x, y in edge_indexes:
        edge_index=int(x)
        #Prahovanie hrán podľa podobnosti vrcholov
        if y < similarity_threshold:
            continue
        #Výpočet parametrov hrany
```

```
row_edge = int(edge_index/image_size);
column_edge = edge_index-row_edge*image_size;

solution = 0
if images_solutions[i][row][column] == images_solutions[i][row_edge][column_edge]:
    solution = 1
weight = pow(abs(images[i][row][column]-images[i][row_edge][column_edge])/255,2)
#Pridanie hrany do grafu
graph.add_edge(index, edge_index, weight=weight, solution=solution)
```