VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Michal Polách

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 OBRAZU NEVYVÁŽENÝCH DAT POMOCÍ UMĚLÉ INTELIGENCE

IMAGE SEGMENTATION OF UNBALANCED DATA USING ARTIFICIAL INTELLIGENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Michal Polách
AUTHOR

VEDOUCÍ PRÁCE — Ing. Martin Kolařík
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Student: Bc. Michal Polách ***ID:*** 173729
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Segmentace obrazu nevyvážených dat pomocí umělé inteligence

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte současné metody segmentace obrazu, které používají tzv. hluboké učení. V rámci diplomové práce proveďte rešerši těchto metod a jejich vlastností se zaměřením na použití pro segmentaci nevyvážených dat. Dále ze zkoumaných postupů vyberte vhodnou metodu a aplikujte ji po dohodě s vedoucím na vybraný dataset. Výběr zdůvodněte a metodu otestujte pro použití na vybraná data. Výsledky testování vhodně prezentujte.

DOPORUČENÁ LITERATURA:

[1] LONG, Jonathan, Evan SHELHAMER a Trevor DARRELL, Fully convolutional networks for semantic segmentation [online]. [cit. 2017-09-13]. DOI: 10.1109/CVPR.2015.7298965. ISBN 10.1109/CVPR.2015.7298965. Dostupné z: http://ieeexplore.ieee.org/document/7298965/

[2] SCHMIDHUBER, Jürgen, Deep learning in neural networks: An overview [online]. [cit. 2017-09-13]. DOI: 10.1016/j.neunet.2014.09.003. ISBN 10.1016/j.neunet.2014.09.003. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0893608014002135

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: Ing. Martin Kolařík
Konzultant:

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

ABSTRAKT

Tato práce se zaměřuje na problematiku segmentace nevyvážených dat pomocí umělé inteligence. V práci jsou prozkoumány známé metody pro vypořádání se s nevyváženými daty, z nichž jsou vybrány vhodné metody, a ty jsou aplikovány na reálný problém, ve kterém je cílem segmentovat nevyvážená data s poměrem tříd větším než 6000:1.

KLÍČOVÁ SLOVA

Segmentace, Klasifikace, Metrika, Ztrátová funkce, Neuronová síť, Roztroušená skleróza, Nevyvážená data, Fokální ztráta, Diceův koeficient, Umělá inteligence

ABSTRACT

This thesis focuses on problematics of segmentation of unbalanced datasets by the use of artificial inteligence. Numerous existing methods for dealing with unbalanced datasets are examined, and some of them are then applied to real problem that consist of segmentation of dataset with class ratio of more than 6000:1.

KEYWORDS

Segmentation, Clasification, Metric, Loss function, Neural network, Multiple sclerosis, Unbalanced data, Focal loss, Dice coeficient, Artificial intelligence

POLÁCH, Michal. *Segmentace obrazu nevyvážených dat pomocí umělé inteligence*. Brno, 2019, 61 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Martin Kolařík

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Segmentace obrazu nevyvážených dat pomocí umělé inteligence“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Martinu Kolaříkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autora

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

Obsah

Seznam obrázků

Úvod

Tato studentská práce se věnuje problému segmentace obrazu pomocí umělé inteligence v případech, kdy poměr zastoupených tříd v obraze je velice nevyvážený.

V reálném prostředí je nevyváženost docela běžná věc, ale lidská mysl se dobře dokáže soustředit na konkrétní objekty a v podstatě ignorovat okolní prostředí. Bohužel umělé neuronové sítě takovouto vlastnost běžně nemají, takže ve většině případů mají problém si všimnout detailů.

Problém spočívá v tom, že běžně používané metriky pro hodnocení míry naučenosti selhávají v posouzení kvality naučené neuronové sítě. Neuronové sítě tak mají tendenci zanedbat vzácně se vyskytující třídu a klasifikovat celý obraz jako převažující třídu, čímž sice síť dosáhne vysoké přesnosti, ale z výsledných predikcí nezískáme jakoukoliv užitečnou informaci.

V rámci práce budou vysvětleny metody segmentace obrazu, princip fungování neuronových sítí a budou do hloubky prozkoumány metriky pro hodnocení sítě. Následně bude vytvořena umělá neuronová síť pomocí knihovny Keras v programovacím jazyce Python, na níž bude problém demonstrován na segmentaci roztroušené sklerózy ze snímků mozků. Poměr zastoupení pozitivní třídy, tj. pixelů, na kterých se ložiska roztroušené sklerózy nachází, vůči negativní třídě je menší než 1:6000, takže se jedná o silně nevyvážená data. Tento příklad bude sloužit jako reálná ukázka účinnosti zkoumaných metod v praxi.

Práce si dává za snahu objevit vhodné techniky pro boj s extrémně nevyváženými daty a na praktické ukázce je otestovat. Tyto techniky by v budoucnu mohly posloužit jako vodítko k řešení četných problémů segmentace obrazu nevyvážených dat.

1 Teoretická část studentské práce

1.1 Segmentace obrazu

Segmentace obrazu je technika zpracování obrazu, jejíž cílem je rozdělení obrazu do oblastí, které odpovídají různým, často reálným, objektům. Každý vzorek vstupních dat, v tomto případě obrazový bod, se opatří *značkou* (anglicky *label*), jenž určuje příslušnost k jedné či více klasifikačních tříd.

V dnešní době je segmentace obrazu fundamentální technikou pro spoustu aplikací využívajících počítačové vidění, zpracování nebo analýzu videa nebo obrazových dat.

1.1.1 Prahování

Prahování je nejjednodušší metoda segmentace obrazu. Je založená na rozdělení pixelů podle jasu – hodnoty s vyšším jasem než práh se klasifikují jako popředí, zbylé pixely jako pozadí. Obsahuje-li obraz různě jasné úseky, používá se metod adaptivního prahování, kdy se různým úsekům stanoví různé prahy.

Nevýhodou této metody je, že si nedokáže poradit se silně zašuměným obrazem.

1.1.2 Regionální metody

Regionální metody využívají pro segmentaci podobnosti sousedních pixelů. Vetšinou fungují na principu rozesetí oblastí po obraze, které se postupně rozrůstají do podobných sousedních pixelech.

1.1.3 Metody založené na detekci hran

Obraz se obvykle zpracuje gradientním operátorem, jehož výstupem je obraz se zvýrazněnými hranami. V tomto obraze se dále mohou nacházet artefakty, tak se obraz dále zpracovává např. prahováním.

Jako gradientní operátor se používá např. Cannyho hranový detektor nebo Sobelův filtr.

1.1.4 Segmentace obrazu pomocí neuronových sítí

K segmentaci obrazu se stále častěji používají umělé neuronové sítě, jejichž výhodou je, že nemusíme specifikovat kritéria, podle kterých se mají řídit, ale stačí jim předat dostatečné množství správně klasifikovaných dat, a neuronové sítě si tato kritéria najdou samy. To ovšem za předpokladu, že je tomu neuronová síť vhodně přizpůsobena.

1.1.5 Technika klouzavého okna

Někdy obraz obsahuje příliš mnoho zbytečných informací, které zbytečně komplikují proces učení a identifikaci hledaných objektů. Jako jedním z řešení tohoto problému vznikla technika klouzavého okna. Funguje tak, že se objekty hledají v menších oblastech obrazu, v nichž jsou objekty zřetelnější. Většinou se velikost těchto oblastí průběžně mění a částečně se překrývají, aby se našel i objekt, který by jinak mohl být roztroušen po několika segmentech.

1.1.6 Jednostupňové detektory

Jednostupňový detektor je takový, kterému je na vstup přiveden nezpracovaný obraz, a detektor přímo z něj detekuje objekty. Výhodou jednostupňových detektorů je, že jsou rychlé a celý proces detekce se odehrává v jedné fázi. Mezi nejúspěšnější jednostupňové detektory se v současnost řadí architektury YOLO [1], SSD [2], OverFeat [3] nebo RetinaNet [4].

1.1.7 Dvoustupňové detektory

Princip dvoustupňových detektorů spočívá v tom, že se obrazy nejprve předzpracují, hrubě se najdou nějaké objekty, a teprve potom se předzpracovaný obraz předává samotnému detektoru. Díky tomu dosahují dvoustupňové detektory lepších výsledků než jednostupňové.

První fáze může probíhat například tak, že se obraz vyfiltruje od většiny negativních pozic a ve druhé fázi se zbylé pixely klasifikují na popředí/pozadí. V druhé fázi se často využívají pro detekci objektů konvoluční neuronové sítě, čímž se rapidně zvýšila přesnost klasifikace. Ukázkou dvoustupňového detektoru je např. architektura navržená v [5].

1.2 Umělá neuronová síť

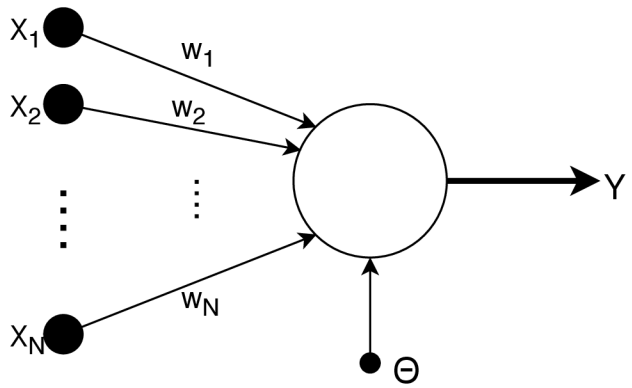
Neuronová síť je jeden z výpočetních modelů používaných při tvorbě umělé inteligence. Její funkce napodobuje funkci neuronů živých organismů. Umělá neuronová síť se může skládat z jednoho (viz **perceptron**) nebo více neuronů, které mohou být zapojené jak paralelně, tak sériově. Sériové spojení znamená, že výstup jednoho neuronu bude vstupem neuronu v další vrstvě. Takovéto zapojení pak nazýváme **vícevrstvý perceptron**.

Mezi jednotlivé vrstvy neuronů navíc můžeme přidávat i jiné matematické operace. V poslední době se například stalo velice populární používat konvoluci, která je vhodná pro hledání vzorů v datech (v obraze, zvuku, internetových útocích). Takovéto sítě se pak nazývají **konvoluční neuronové sítě**.

Všechny dosud zmíněné architektury počítají pouze s jedním směrem toku informací – od vstupu k výstupu, říkáme jim *dopředné sítě*, existují však i sítě, kde výstupy některých neuronů vstupují do neuronů předchozích vrstev, v tom případě se jedná o **zpětnovazebné (rekurentní) neuronové sítě**.

1.2.1 Model neuronu

Jelikož jsou reálné neurony dost komplexní, tak se pro tvorbu umělých neuronových sítě používá zjednodušený model (Obr. 1.1).



Obr. 1.1: Model neuronu

Výstup tohoto neuronu je vyjádřen funkcí

$$Y = S(\sum_{i=1}^{N}(w_i \cdot x_i) - \Theta) \tag{1.1}$$

kde w_i je váha vstupu i, x_i je hodnota vstupu i, S je přenosová funkce neuronu a Θ je práh citlivosti neuronu.

Sečtením vážených vstupů neuronů a odečtením prahu se získá tzv. potenciál neuronu. Ten se předává jako parametr aktivační funkce. Jako aktivační funkce se používají: skoková funkce, sigmoidální funkce, softmax, ReLU.

1.2.2 Předzpracování dat

Někdy je vhodné vstupní data předzpracovat, aby měla neuronová síť snadnější tato data interpretovat, například při práci s textem můžeme vstupní slova nahradit čísly, aby jsme síť zbytečně nezatěžovali skládáním slov z jednotlivých písmenek. Takto by jsme mohli ještě pokračovat třeba rozdělením slov podle kořenu slova, který by jsme mohli opět reprezentovat číslem a předávat jako další vstupní informaci. Takovéto úpravě vstupních dat se anglicky říká *preprocessing*.

Opačný proces zpracování dat se děje za výstupem, kdy ze směti čísel, jež byly výstupem sítě zpátky získáváme informaci snadno interpretovatelnou člověkem. V tomto příkladě bychom tedy přeložili čísla zpět na původní slova.

1.3 Učení neuronové sítě

Od naučené neuronové sítě očekáváme, že po vložení vstupních dat získáme na výstupu očekávaná data. Učením neuronové sítě se nazývá proces, jehož účelem je přizpůsobit parametry neuronů tak, aby reakce sítě co nejlépe odpovídala očekávané. Pro vyhodnocení toho, jak dobře je síť naučená, používáme **ztrátovou funkci**.

Učení může probíhat několika způsoby:

- **Učení s učitelem**
  - učení probíhá na trénovací množině vstupů, u kterých víme výsledek
  - snaží co nejpřesněji simulovat vztah mezi vstupem a výsledky
  - běžně se používá pro problémy *klasifikace* a *regrese*
- **Učení bez učitele**
  - k dispozici jsou trénovací příklady, ale neznáme očekávaný výsledek
  - síť se snaží napodobit strukturu trénovacích příkladů
  - vhodné ke *shlukování* nebo pro hledání trendů v datech (*asociace*)
- **Zpětnovazební učení**
  - nemáme trénovací příklady, síť se musí učit sama
  - hraní počítačových her, robotika

1.3.1 Ztrátová funkce

Ztrátová funkce (angl. *loss function*) se používá jako metrika pro vyhodnocení, jak dobře je neuronová síť naučená.

Počítá se na základě reálného výstupu neuronové sítě a očekávaných výsledků. V případě učení bez učitele si síť nejprve musí vypočítat pravděpodobnost výsledku pro daný vstup a poté počítá s touto hodnotou jako se správným výsledkem.

Přestože se někdy síť zdá být dle hodnoty ztrátové funkce naučená dobře, nemusí to vždy odpovídat skutečnosti.

V našem případě silně nevyvážených dat máme krásnou ukázku takového případu – jelikož se třída RS vyskytuje v tak malém množství, podle většiny ztrátových funkcí je dost dobrý výsledek, klasifikujeme-li všechny pixely jako negativní. Tím dosáhneme přesnosti přes 99%, ale prakticky nedostaneme žádnou užitečnou informaci.

Abychom kvalitně naučili neuronovou síť, musíme zvolit správnou metriku, která bude dbát na to, aby se třída RS nezanedbala.

1.3.2 Zpětná propagace chyby

Zpětná propagace chyby (angl. backwards propagation of error, zkráceně *backpropagation*) je technika používaná pro učení neuronových sítí, tedy přizpůsobování vah nebo i prahů jednotlivých neuronů.

Obecně funguje tak, že se nejprve spočítá gradient ztrátové funkce, neboli závislost hodnoty ztrátové funkce na jednotlivých parametrech sítě, a potom se podle gradientu a aktuální hodnoty ztrátové funkce upravují parametry sítě tak, aby nová hodnota ztrátové funkce byla co nejmenší.

O tom, jaké parametry a jak se změní, rozhoduje použitý optimalizér. V Kerasu je v základu dostupných několik optimalizérů, z nichž jsem na základě studií jsem se rozhodl otestovat dva optimalizéry: Adam [6] a Nadam [7].

1.4 Architektury neuronových sítí

Hluboká neuronová síť je neuronová síť, která má mezi vstupní a výstupní vrstvou neuronů jednu či více skrytých vrstev.

V každé vrstvě se provádí nějaká matematická manipulace vstupních dat. Hluboké neuronové sítě většinou obsahují velké množství vrstev, díky čemuž dokážou modelovat komplexní nelineární vztahy mezi vstupními daty a výstupem.

V klasických hlubokých neuronových sítích proudí data jen jedním směrem, od vstupu k výstupu beze smyček. Speciálním případem jsou **rekurentní neuronové sítě**, kde můžou data proudit všemi směry. Další významnou podmnožinou hlubokých sítí jsou tzv. **konvoluční neuronové sítě**.

1.4.1 Perceptron

Nejjednodušší architekturou dopředné neuronové sítě je perceptron. Byl vynalezen roku 1957 americkým psychologem Frankem Rosenblattem a skládá se pouze z jednoho neuronu. Jeho použití je značně omezené, jelikož jej lze použít pouze na lineárně separovatelné množiny [8].

1.4.2 Vícevrstvý perceptron

Vícevrstvý perceptron vznikl jako rozšíření obyčejného perceptronu. Skládá se vždy nejméně ze tří vrstev perceptronů

1. Vstupní vrstva – jeden nebo více perceptronů, které se nachází na vstupu neuronové sítě
2. Skrytá vrstva – vrstva perceptronů mezi vstupní a výstupní vrstvou
3. Výstupní vrstva – konečné perceptrony, na kterých získáváme výstupní data

Pro lepší představu o tom, jak funguje MLP doporučuji vyzkoušet webovou aplikaci na stránce `https://playground.tensorflow.org/`, kde je možné si vytvořit jednoduchý MLP pro binární klasifikaci nebo regresi dat, nastavit počet jeho skrytých vrstev, počet neuronů nebo aktivační funkci a následně spustit učení na zvolených datech. V reálném čase lze pak vidět, jak se parametry přizpůsobují, podle jakých kritérií se momentálně klasifikují data a spoustu dalších věcí.

1.4.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě fungují na principu hledání rysů v obraze nebo v jakýchkoliv jiných datech, ve kterých je důležité pořadí dat. Neuronová síť se naučí několik rysů (angl. *features*), které se pak pomocí konvoluce hledají ve vstupních datech. Klasické konvoluční sítě se používají ke klasifikaci obrazových dat – dokáží říct, jaké objekty jsou na obraze, ale ne kde se nacházejí.

V hlubokých konvolučních sítích se používá hned několik po sobě jdoucích konvolucí, z nichž každá může obraz konvoluovat s jinými rysy.

1.4.4 Rekurentní neuronové sítě

Rekurentní neuronová síť (zpětnovazebná, RNN), je architektura sítě, v níž data neputují jen jedním směrem, ale mohou putovat i zpátky. Díky zpětné vazbě umožňují simulovat „paměť“ sítě, a tak jsou vhodné pro aplikace, kde je důležitý kontext – například generování textu, sekvenční predikce, ...

Problémem klasických RNN nastává, kdy kontext k aktuální informaci sahá daleko do minulosti – takto vzdálené závislosti se síť nedokáže naučit, jak vysvětluje

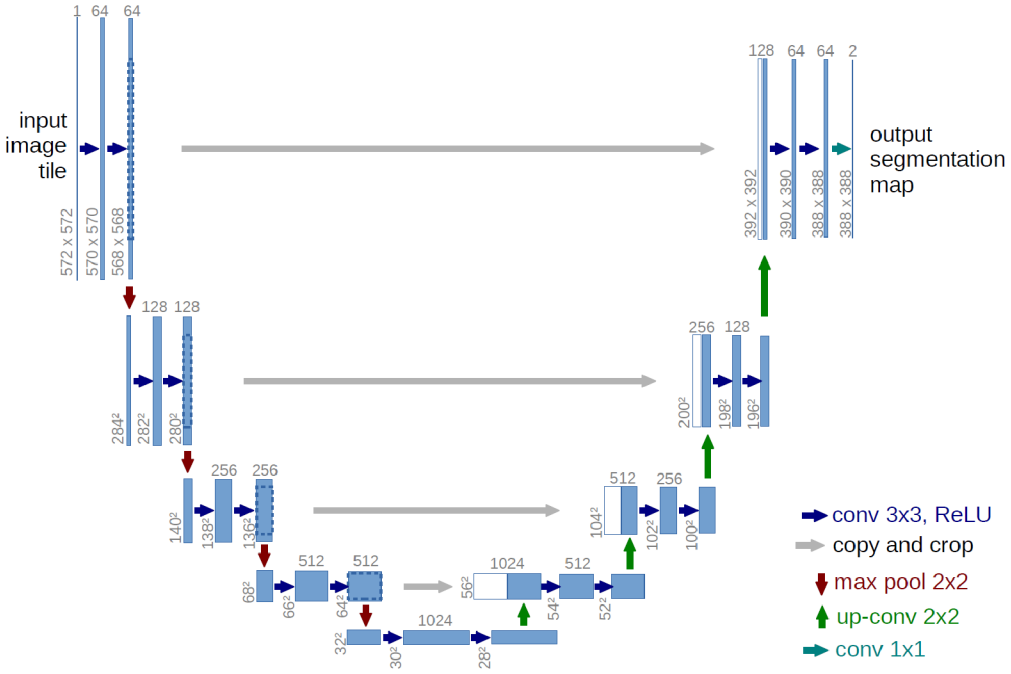
[9]. Tento problém řeší sítě typu **LSTM** [10]. LSTM na rozdíl od klasických RNN, kde se cyklí jednotlivé vrstvy, cyklí LSTM buňky. LSTM buňka si uchovává nějaký stav, který může a nemusí průběžně aktualizovat, přičemž rozhodování, kdy se stav aktualizuje se síť dokáže naučit.

1.4.5 Plně konvoluční sítě

Plně konvoluční sítě se od klasických konvolučních sítí liší v tom, že po provedených konvolucích provádí dekonvoluci pro zobrazení nalezených rysů na původním rozlišení obrazu. Tím se z klasifikační sítě stává segmentační síť – klasifikuje jednotlivé pixely namísto celého obrazu [11].

1.4.6 U-Net

Architektura neuronových sítí U-Net dostala název podle svého tvaru. Jedná se o modifikaci plně konvoluční neuronové sítě, která navíc přidává spoje mezi vrstvami na stejné úrovni (se stejným rozlišením), čímž usnadňuje propagaci již zjištěných rysů. V konečném důsledku je síť efektivnější, a může tak ušetřit spoje mezi po sobě jdoucími vrstvami [12].



Obr. 1.2: Původní architektura U-Net [12]

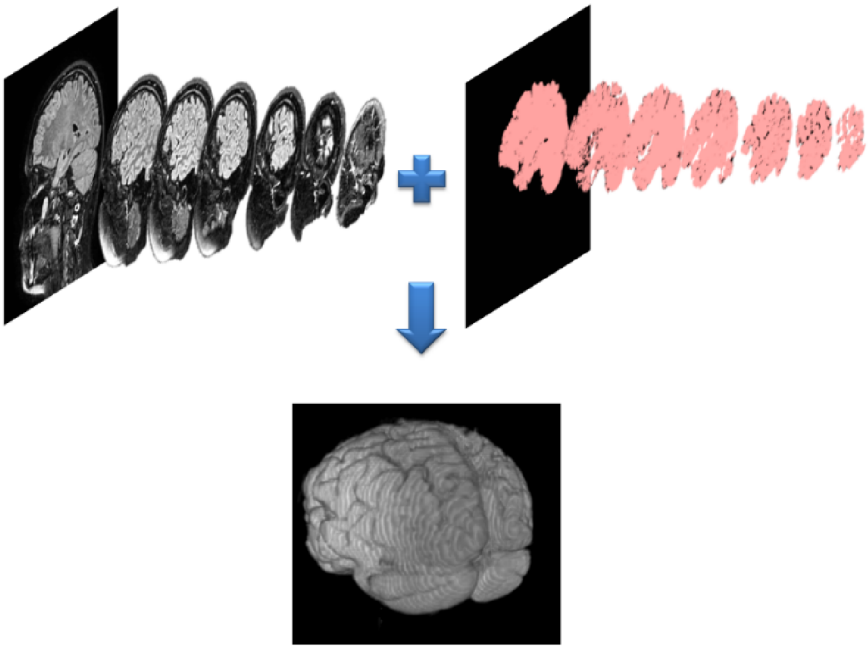
1.4.7 DenseNet

Studie [13] a [14] ukázaly, že spoje mezi vrstvami blízko vstupu a vrstvami blízko výstupu zlepšují celkovou účinnost a přesnost neuronových sítí. Na základě toho vznikla architektura DenseNet, ve které každá další vrstva bere výstupy všech předchozích vrstev jako své vstupy. Tím se zlepší propagace naučených vzorů mezi vrstvami.

1.4.8 3D segmentační sítě

Dalším odvětvím segmentačních neuronových sítí jsou 3D segmentační sítě, které na rozdíl od dosud zmiňovaných nepracují s 2D daty (obrázky), ale s třírozměrnými bloky voxelů. Hlavní myšlenkou 3D segmentace je, že síť bude mít nejen informace o voxelech v rovině, ale i v prostoru. Ty můžou nést užitečnou informaci pro potřebnou segmentaci, ale přichází to na úkor výpočetní náročnosti.

Tento přístup je vhodný zejména v případech, kdy dokážeme vytvořit třírozměrný obraz zkoumané scény. 3D segmentace je často používaná při zkoumání medicínských obrazů, kdy bývají k dispozici za sebou ležící 2D snímky.



Obr. 1.3: Tvorba 3D obrazu z 2D snímků [15]

2 Praktická část

2.1 Instalace prostředí

V této kapitole bude popsána instalace potřebných aplikací pro vytvoření umělé inteligence na systému Windows.

2.1.1 Anaconda

Anaconda je prostředí pro správu balíčků a virtuálních prostředí. Původně vzniklo pro zjednodušení práce s programovací jazykem Python, ale dokáže zabalit a distribuovat i jakýkoliv jiný software. V Pythonu umožňuje vytvořit několik separátních virtuálních prostředí, přičemž každé může mít nainstalované různé verze balíčků, a následně mezi nimi jednoduše přepínat. Jedná se o open-source software a je volně ke stažení na webové stránce `https://www.anaconda.com/download/`.

Po instalaci Anacondy můžeme přejít k vytvoření virtuálního prostředí a instalaci potřebných balíčků. Nejprve spustíme aplikaci Anaconda Prompt, která by se nyní měla objevit v nabídce aplikací. Otevře se okno nápadně připomínající klasický příkazový řádek, jen s tím rozdílem, že před aktuální cestou vidíme momentálně aktivní prostředí Anacondy – v tomto případě základní prostředí `base`. Nové virtuální prostředí (nazvěme jej třeba `deep-learning`) vytvoříme pomocí příkazu `conda create -n deep-learning` a následně se do něj přepneme pomocí `conda activate deep-learning`. Na začátku příkazové řádky by se nyní měl objevit název nového prostředí.

2.1.2 TensorFlow

TensorFlow [16] je open-source knihovna pro strojové učení. V současné době patří mezi nejpopulárnější knihovny v této oblasti.

Neuronové sítě mohou běžet buď na procesoru nebo na grafické kartě. Výrazně doporučuji používat grafickou kartu kdykoliv je to možné, protože učení hlubokých neuronových sítí je velice výpočetně náročné a na procesoru může doba učení takovéto sítě snadno přesáhnout několik dní. Díky tomu, že je proces dobře paralelizovatelný a grafické karty jsou na paralelní výpočty velmi dobře stavěné, dosahují v tomto ohledu mnohonásobně vyšší rychlosti.

Bohužel, TensorFlow vyžaduje pro práci s grafickou kartou techologii CUDA, takže je nutné mít grafickou kartu, která tuto technologii podporuje, nebo si nainstalovat upravenou verzi TensorFlow, která podporuje AMD GPU.

2.1.3 Keras

Keras [17] je vysokoúrovňové aplikační rozhraní pro práci s neuronovými sítěmi. Jako takový může pracovat nad různými frameworky jako jsou TensorFlow, Theano nebo Microsoft Cognitive Toolkit. V mém případě používám jako backend právě TensorFlow.

Naštěstí je Keras spolu s TensorFlow backendem součástí Anaconda balíčku `keras-gpu`. Pro jeho instalaci tedy stačí v příkazové řádce spustit příkaz `conda install keras-gpu`.

2.1.4 Scikit-image

Knihovna `scikit-image` [18] nabízí kolekci užitečných algoritmů pro zpracování obrazu. Jeho instalace se provede pomocí příkazu `conda install scikit-image`. V projektu jej bude potřeba pro úpravu dat a ukládání obrázků.

2.1.5 PyCharm

Pro psaní samotného kódu jsem využil vývojové prostředí PyCharm společnosti JetBrains. Jedná se o open-source aplikaci primárně určenou pro vývoj v programovacím jazyce Python. Jeho výhodami jsou:

- inteligentní zvýrazňování syntaxe,
- napovídání včetně varování před neznámými členy tříd,
- dokáže pracovat s virtuálními prostředími Anacondy
- umožňuje synchronizovat práci pomocí verzovacích systémů, např. Git.

Na webové stránce `https://www.jetbrains.com/pycharm/` je PyCharm volně ke stažení. Při instalaci se stačí řídit pokyny instalátoru.

2.2 Tvorba projektu

V této kapitole bude popsána struktura projektu a funkce jednotlivých komponent. Struktura adresáře projektu je zobrazena na obrázku 2.1.

2.2.1 Příprava dat

K dispozici mám snímky z magnetické rezonance mozků s roztroušenou sklerózou, rozdělené do osmi složek po 256 snímcích. Každá složka zobrazuje snímky jednoho mozku. Navíc mám ke každému mozku dostupnou černobílou masku, která vyznačuje ložiska roztroušené sklerózy na snímku.

Snímky jsem rozdělil do adresářů `train`, `masks` a `test`. Adresář `train` obsahuje testovací data, tj. sedm adresářů se snímky z magnetické rezonance mozku, a adresář `test` obsahuje testovací data – jeden adresář se snímky mozku. V adresáři `masks` se nachází správně segmentované masky pro všechny snímky identifikovány dle názvů adresářů.

```
rs_segmentation
├── masks ..... adresáře s maskami mozků
├── test ..... adresáře s testovacími snímky
├── train ..... adresáře s trénovacími snímky
├── custom_losses.py
├── predict.py
├── rs_data.py
├── train.py
└── nets.py
```

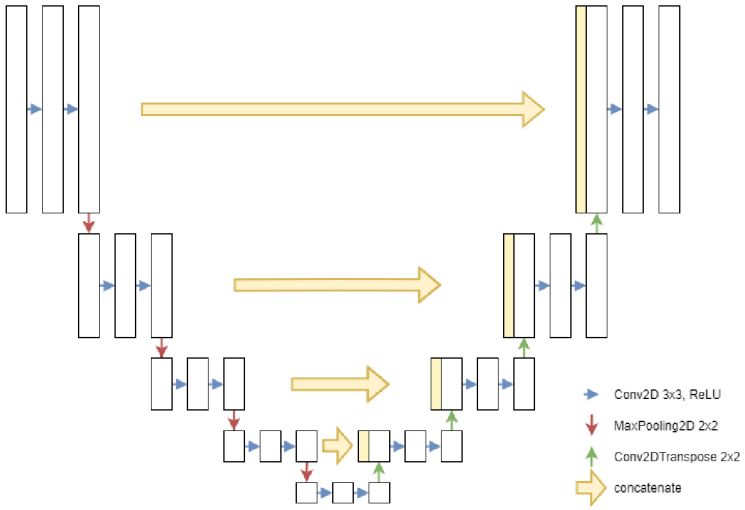
Obr. 2.1: Struktura adresáře projektu

Neuronová síť neumí pracovat přímo s obrazy, ale je potřeba snímky nejprve převést do NumPy [19] matic, se kterými si pak už poradí. Pro tento účel jsem připravil skript `rs_data.py`, který načte obrázky ze adresářů `train`, `masks` a `test` a uloží je jako NumPy matice `train_images.npy`, `train_masks.npy`, `test_images.npy` a `test_masks.npy`.

2.3 Tvorba neuronové sítě

Pro tvorbu neuronové sítě využívám knihovnu Keras. Díky tomu se nemusím zabývat jednotlivými neurony, ale mohu rovnou vytvářet celé vrstvy sítě, a ty pak jen skládat do potřebné architektury.

Rozhodl jsem se, že vytvořím síť architektury U-Net. Nechal jsem se inspirovat projektem uživatele jocicmarko na serveru GitHub. Byla zde použita velice podobná architektura jako u originální U-Net sítě.



Obr. 2.2: Architektura vytvořené sítě

2.3.1 Vstupní vrstva

Vrstva `keras.layers.Input` je vstupní vrstva sítě, která slouží k vytvoření tenzoru ze vstupní matice. Jako parametr přijímá tvar vstupních dat. V mém případě má vstupní vrstva tvar `(400, 400, 1)`, kde první dvě čísla značí rozlišení vstupujícího obrazu a poslední číslo značí počet barevných kanálů – pouze jedna jasová složka.

2.3.2 Konvoluční vrstva

2D konvoluční vrstva `keras.layers.Conv2D` je klíčová vrstva pro zpracovávání obrazu. Vnitřně si uchovává sadu rysů (angl. **feature maps**), což jsou v podstatě miniaturní obrázky, které se pak pomocí konvoluce hledají ve vstupním obraze. Výstupem této vrstvy je sada matic (pro každý rys jedna), jejichž hodnoty udávají pravděpodobnost výskytu daného rysu na příslušné pozici.

V důsledku konvoluce s rysem o velikosti větší než 1 pixel, má výstupní mapa menší rozměry než vstupní. Proto, aby výstup konvoluční vrstvy měl stejné rozlišení jako vstup, se používá technika zvaná *zero padding*. Ta nedělá nic jiného, než že vstupní obrázek na okrajích rozšíří o vhodný počet nulových pixelů tak, aby po konvoluci měl opět původní velikost.

2D Konvoluční vrstva v knihovně Keras umožňuje nastavit velké množství hyperparametrů (tj. parametrů, které se síť nenaučí sama, ale je třeba je zadat už při její tvorbě). Mezi nejdůležitější patří:

- `filters` – počet rysů, které se bude vrstva učit,
- `kernel_size` – velikost rysů a
- `activation` – typ aktivační funkce, který se aplikuje na výstup vrstvy.

2.3.3 Sdružovací vrstva

Sdružovací vrstva `keras.layers.MaxPooling2D` je zde použita ke koncentraci relevantní informace. Jejími hlavními parametry jsou:

- `pool_size` – velikost okna a
- `strides` – horizontální a vertikální krok.

Funguje tak, že postupně prochází vstupní obraz s krokem udaným střídou, v daném místě umístí okno a z něj následně vybere maximální hodnotu. Výsledkem je zmenšená mapa s vybranými lokálními maximy.

2.3.4 Transponovaná konvoluční vrstva

Transponovaná konvoluční vrstva `keras.layers.Conv2DTranspose`, někdy též nazývaná dekonvoluční. Jedná se o zpětnou operaci ke konvoluční vrstvě a v plně konvolučních sítích se používá ke zpětnému nadvzorkování konvoluovaného obrazu za pomocí transponovaného signálu. Podrobný popis, jak takové nadvzorkování funguje je vysvětelno v [20].

2.3.5 Spojovací vrstva

Vrstva `keras.layers.Concatenate` spojuje sadu vstupů o stejných rozměrech kromě osy, na které se vstupy spojují. V mém projektu ji používám pro připojení naučených rysů k výstupu dekonvoluční vrstvy.

2.4 3D segmentace

Upravil jsem stávající architekturu předešlé neuronové sítě tak, aby využívala místo 2D segmentace 3D segmentaci. Rozdíl oproti 2D segmentaci je takový, že se namísto 2D obrázků používá 3D obraz, a tak má síť u každého voxelu informaci o voxelech nejen vedle něj, ale i před a za ním.

2.4.1 Úprava vrstev

Neuronovou síť jsem upravil tak, aby si zachovala stejnou funkci jako původní verze s tím rozdílem, že namísto 2D operací se budou provádět 3D operace.

Vše, co bylo nutno udělat, bylo nahradit vrstvy `keras.layers.Conv2D`, `keras.layers.MaxPooling2D` a `keras.layers.Conv2DTranspose` jejich třírozměrné protějšky `keras.layers.Conv3D`, `keras.layers.MaxPooling3D` a `keras.layers.Conv3DTranspose` a uzpůsobit parametry všech vrstev pro 3D.

2.4.2 Nastavení velikosti bloků

Ideální případ 3D segmentace by bylo vytvořit 3D obraz celého mozku, který by se vcelku vložil na vstup neuronové sítě. Bohužel, z důvodu technického omezení, nemám možnost takovouto implementaci realizovat, jelikož by to vyžadovalo obrovské množství grafické paměti.

Možná východiska jsou: snížit rozlišení mozku, nebo použít menší bloky jako výřezy mozku. Obě tyto alternativy mají své klady i zápory.

Snížením rozlišením mozku se sice ztratí část informace kvůli interpolaci pixelů, ale výhodou je, že zůstane kompletní prostorová informace.

Rozkouskováním mozku na menší bloky jsme na druhou stranu schopni zachovat plné rozlišení právě za cenu informace, kde se blok v mozku nachází.

Zvolil jsem metodu rozkouskování, přičemž jsem velikost bloků nechal nastavitelnou parametrem `block_size`.

2.4.3 Zpracování vstupních dat a rekombinace bloků

Jelikož má nyní vstupní vrstva sítě třírozměrný tvar, bylo potřeba upravit předzpracování dat tak, aby výsledkem byla NumPy matice o rozměrech kompatibilní s vstupní vrstvou. Jestliže by měla vstupní vrstva tvar `(64, 64, 64, 1)`, musí mít matice vstupních dat tvar `(X, 64, 64, 64, 1)`, kde `X` je počet bloků.

V souboru `nets.py` jsem vytvořil metody `make_blocks(brains, block_shape, stride)` a `assemble_blocks(blocks, brain_shape, stride)`.

Parametry metody `make_blocks`:

- `brains`
  Matice vstupních dat o rozměrech (počet mozků, počet snímků, počet řádků, počet sloupců, počet kanálů),
- `block_shape`
  Požadovaná velikost bloků ve tvaru (hloubka, výška, šířka),
- `stride`
  Střída udává, v jaké vzájemné vzdálenosti od sebe se generují jednotlivé bloky. Pokud není střída specifikována, automaticky se nastaví na velikost bloků.

Parametry metody `assemble_blocks`:

- `blocks`
  Matice bloků s očetkávaným tvarem (počet bloků, hloubka, výška, šířka, počet kanálů)
- `brain_shape`
  Výsledná velikost matice mozku ve tvaru (hloubka, výška, šířka),
- `stride`
  Střída udává vzájemnou vzdálenost bloků od sebe. Pokud není střída specifikována, automaticky se nastaví na velikost bloků.

Algoritmus těchto metod umožňuje nastavit menší střídu, než je velikost bloku. V tom případě se bloky budou částečně překrývat.

Výhodou překrývání bloků je, že se díky tomu dá uměle zvýšit množství trénovacích dat.

Nevýhodou je složitější rekombinace bloků. Za běžných okolností se bloky jeden po druhém umisťují na správné souřadnice do matice celého mozku přepisujíce veškeré předchozí hodnoty. Řešením není ani zprůměrování původní a vkládané hodnoty, protože by se brala v potaz i hodnota, se kterou se matice inicializuje (v NumPy je to nula) a každá další vkládaná hodnota by měla větší váhu, než předchozí vkládané.

Tento problém řeším tak, že hodnoty všech bloků vkládaných do matice mozku sčítám a vytvořil jsem druhou matici o stejných rozměrech jako matice mozku, která slouží jako počítadlo zápisů příslušných prvků. Po vložení všech bloků do matice mozku se celá matice po složkách vydělí maticí „počítadla“, čímž se na všech pozicích získá průměrná hodnota všech překrývaných bloků.

2.4.4 Úprava kódu pro snadné přepínání sítí

Nutno poznamenat, že 2D a 3D segmentační síť vyžadují jiný způsob zpracování dat – u 2D sítě stačí případná změna velikosti obrázků, zatímco 3D síť vždy potřebuje rozdělit data do bloků dané velikosti.

Abych mohl různé kombinace sítí a ztrátových funkcí rychle otestovat, využil jsem konceptu objektově orientovaného programování. Vytvořil jsem abstraktní bázovou třídu `BaseNet`, která deklaruje metody:

- `get_model()`
  Vrátí zkompilovaný model neuronové sítě
- `prepare_images(images)`
  Připraví obrázky pro vstupní vrstvu sítě
- `prepare_masks(masks)`
  Připraví masky pro vstupní vrstvu sítě. Hodnoty pixelů masek se škálují do intervalu $\langle 0; 1 \rangle$.
- `postprocess_result(predicted_masks)`
  Upraví predikované masky do NumPy matice o rozměrech (počet mozků, počet plátků, počet řádků, počet sloupců, počet kanálů)

Jednotlivé architektury sítě jsem vytvářel jako potomky této bázové sítě, což umožnilo z pohledu skriptu `train.py` pracovat se všemi architekturami jednotně.

2.5 Implementace ztrátových funkcí

V této kapitole bude popsána funkce a implementace jednotlivých ztrátových funkcí.

Testované ztrátové funkce jsem volil s ohledem na to, aby byla pokud možno vhodná pro nevyvážená data. Většina těchto funkcí se v praxi běžně používá, a některé jsou dokonce implicitně připraveny v knihovně Keras. Ztrátová funkce použitá pro učení modelu neuronové sítě se specifikuje parametrem `loss` metody `model.compile` – funkci můžeme specifikovat přímo nebo i jen názvem, pokud je dostupná v Kerasu.

Keras umožňuje i vytvoření vlastních ztrátových funkcí a metrik – stačí vytvořit funkci s dvěma parametry po vzoru

```
def my_loss_function(y_true, y_pred):
```

a předat ji parametrem při kompilaci modelu. Tato funkce bude v době učení volána, přičemž první parametr se naplní maticí pravdivých dat a druhý parametr maticí predikovaných dat. Na základě těchto parametrů můžeme vypočítat vlastní hodnotu ztrátové funkce, kterou předáme jako návratovou hodnotu.

2.5.1 Matice záměn

Velké množství metrik pro hodnocení klasifikátorů se dá přehledně znázornit pomocí tzv. matice záměn (angl. *confusion matrix*).

Jedná se o kontigenční matici, jejíž sloupce udávají skutečnou hodnotu zkoumaných prvků a řádky udávají hodnotu předpovídanou klasifikátorem. U binárních klasifikátorů jsou jen dvě možnosti předpovědi – pozitivní a negativní. Matice tedy bude mít dva řádky a dva sloupce, pro jejichž buňky se zažila označení

- **TP** – skutečná pozitiva, *true positive*,
- **TN** – skutečná negativa, *true negative*,
- **FP** – falešná pozitiva, *false positive*,
- **FN** – falešná negativa, *false negative*.

		Skutečnost	
		Pozitivní	Negativní
Predikce	Pozitivní	TP	FP
	Negativní	FN	TN

Tab. 2.1: Vliv použité ztrátové funkce na ostatní metriky

Matice záměn pro binární klasifikaci předpokládá, že každý prvek bude jednoznačně predikován do jedné ze tříd. Výstupem vytvořené neuronové sítě však je

pravděpodobnost toho, že prvek náleží do pozitivní třídy. Aby jej bylo možné zařadit do matice záměn, zvolil jsem takový přístup, že část prvku poměrně odpovídající predikované hodnotě se bere jako pozitivní predikce, zatímco zbytek se do matice zařadí jako negativní predikce.

Na matici záměn je navázaná řada běžně používaných metrik pro vyhodnocování testů. Typickými příklady jsou např.

- **Přesnost**
  - zkratka **ACC**, angl. *Accuracy*
  - poměr správně klasifikovaných prvků vůči všem zkoumaným prvkům

$$\mathrm{ACC} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

- **Citlivost**
  - zkratka **TPR**, angl. *Sensitivity, Recall, True Positive Rate*
  - poměr správně klasifikovaných pozitivních prvků vůči počtu všech pozitivních prvků

$$\mathrm{TPR} = \frac{TP}{TP + FN} \tag{2.2}$$

- **Specifičnost**
  - zkratka **TNR**, angl. *Specificity, True Negative Rate*
  - poměr správně klasifikovaných negativních prvků vůči počtu všech negativních prvků

$$\mathrm{TNR} = \frac{TN}{TN + FP} \tag{2.3}$$

- **Preciznost**
  - zkratka **PPV**, angl. *Precision, Positive Predictive Value*
  - poměr správně klasifikovaných pozitivních prvků vůči všem prvkům klasifikovaných jako pozitivní

$$\mathrm{PPV} = \frac{TP}{TP + FP} \tag{2.4}$$

2.5.2 Binární cross-entropie

Cross-entropie je jednou z nejčastěji používaných ztrátových funkcí pro klasifikační neuronové sítě. Udává míru nepodobnosti dvou množin. Obecně je funkce cross-entropie definována vztahem

$$\mathrm{CE}(p, y) = -\sum_i y_i \log p_i, \tag{2.5}$$

kde y_i a p_i vyjadřují skutečnou a predikovanou pravděpodobnost, že zkoumaný prvek náleží do třídy i, přičemž

- hodnota y_i je předem dána a nabývá hodnot 0 nebo 1 a
- hodnota p_i je predikovaná sítí a platí, že $p_i \in \langle 0; 1 \rangle$.

Binární cross-entropie je speciální případ cross-entropie, ve které se počítá jen se dvěma klasifikačními třídami – pozitivní a negativní, přičemž jeden prvek nemůže být součástí obou tříd zároveň. Těchto vlastností se dá využít k tomu, aby nebylo třeba predikovat pravděpodobnost pro každou třídu zvlášť, ale postačí predikce do pozitivní třídy. Binární cross-entropie je vyjádřena vztahem

$$\mathrm{BCE}(p, y) = \begin{cases} -\log(p) & \text{pokud } y = 1 \\ -\log(1-p) & \text{jinak.} \end{cases} \tag{2.6}$$

Pro jednodušší zápis můžeme definovat p_t

$$p_\mathrm{t} = \begin{cases} p & \text{pokud } y = 1 \\ 1-p & \text{jinak} \end{cases} \tag{2.7}$$

a používat zápis

$$\mathrm{BCE}(p, y) = \mathrm{BCE}(p_\mathrm{t}) = -\log(p_\mathrm{t}). \tag{2.8}$$

Počítá s tím, že u každého vzorku se značkou $y \in \{0; 1\}$ je odhadnutá pravděpodobnost $p \in \langle 0; 1 \rangle$, se kterou tento vzorek náleží do pozitivní třídy. Pravděpodobnost toho, že tento vzorek patří do negativní třídy pak automaticky odpovídá hodnotě $(1 - p)$. To znamená, že každý pozitivní vzorek přidá do celkové funkce $\log(p)$ a každý negativní vzorek přidá $\log(1 - p)$. Celková hodnota ztrátové funkce je pak rovna průměru dílčích hodnot cross-entropie všech prvků (2.9).

$$\mathrm{BCE}_\mathrm{loss} = -\frac{1}{N} \cdot \sum^{N} \log(p_\mathrm{t}). \tag{2.9}$$

Binární cross-entropie je již implementovaná v Kerasu, takže ji stačí v modelu nastavit parametr `loss='binary_crossentropy'`.

2.5.3 Diceův koeficient podobnosti

Diceův koeficient podobnosti (Dice Similiarity Coefficient, DSC) je známý pod mnoha jmény, jmenovitě např. F1-skóre, Czekanowskiho binární index, nebo Zijdenbosův index podobnosti. Pro výpočet koeficientu se bere v úvahu *přesnost* a *citlivost* testu.

Přesnost testu udává poměr správně zvolených prvků ku celkovému počtu zvolených prvků, neboli udává poměr, kolik ze zvolených prvků je relevantních. Citlivost pro změnu udává, kolik procent ze všech relevantních prvků je zvoleno.

$$\mathrm{DSC}(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|} \tag{2.10}$$

Rovnice 2.10 vyjadřuje hodnotu DSC pomocí množin X a Y, kde množina X značí množinu skutečných značek prvků a množina Y značí predikované značky, přičemž v binární klasifikaci tyto značky nabývají pouze hodnot $[0, 1]$. Průnik množin $X \cap Y$ obsahuje hodnoty, ve kterých značky prvků obou množin nabývají hodnotu 1, a tak je možno zapsat

$$|X \cap Y| = TP. \tag{2.11}$$

Obdobně se dají vyjádřit i zbylé členy rovnice:

$$|X| = TP + FN, \tag{2.12}$$

$$|Y| = TP + FP, \tag{2.13}$$

a celá rovnice 2.10 se tedy dá přepsat jako

$$\mathrm{DSC} = \frac{2TP}{2TP + FN + FP}. \tag{2.14}$$

Lze si povšimnout, že funkce není definována pro případy, kdy všechny prvky zkoumané množiny jsou skutečnými negativy. Kvůli tomuto případu jsem použil nejmenší možnou konstantu ε, kterou jsem přičetl ke jmenovateli rovnice. Takto upravená rovnice (2.15) ve výše zmíněném příkladě nabývá hodnoty 0. Mírně se změna projeví i v ostatních případech, ale díky velice malé velikosti konstanty ε je změna zanedbatelná.

$$\mathrm{DSC} = \frac{2TP}{2TP + FN + FP + \varepsilon} \tag{2.15}$$

Na první pohled se může zdát, že by bylo moudřejší přičíst tuto konstantu i k čitateli, aby v případě správně označených negativ funkce nabývala hodnotu 1. Tuto variantu jsem také nejprve vyzkoušel, při testování jsem však došel k závěru, že se síť učí lépe s konstantou pouze ve jmenovateli. Důvodem je, že případ, kdy je ve zkoumané množině 100% zastoupení negativními prvky, nastává docela často (odhadem přibližně ve 20-40% případů). Keras vždy počítá průměr ztrátové funkce za

celou uběhlou epochu, a tudíž se průměrná hodnota DSC pohybuje relativně vysoko i v případě, kdy síť klasifikuje všechny prvky jako negativní a toto chování je nežádoucí.

Diceův koeficient nabývá v ideálním případě hodnotu 1, v nejhorším případě hodnotu 0. Pro použití tohoto koeficientu jako ztrátové funkce je tedy zapotřebí použít jeho komplementární hodnotu.

$$\mathrm{DL}(X, Y) = 1 - \mathrm{DSC}(X, Y). \tag{2.16}$$

Výpis 2.1: Implementace Diceova koeficientu a jeho ztrátové funkce

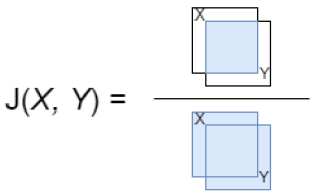
```
def dice_coef(y_true, y_pred):
    eps = K.epsilon()
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + eps) / \
          (K.sum(y_true_f) + K.sum(y_pred_f) + eps)

def dice_coef_loss(y_true, y_pred):
    return 1-dice_coef(y_true, y_pred)
```

2.5.4 Jaccardův index

Taktéž znám jako průnik nad sjednocením (*Intersection over Union*, *IoU*). Tato metrika se používá pro určení podobnosti a rozdílnosti dvou množin vzorků. Je definován jako poměr správně určené množiny ku celkovému počtu prvků, které se nachází alespoň v jedné z množin. Matematicky je Jaccardův index vyjádřen

$$\mathrm{J}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}. \tag{2.17}$$



Obr. 2.3: Grafické znázornění výpočtu Jaccardova indexu

Podobně jako Diceův koeficient, i Jaccardův index se dá přepsat s použitím prvků z matice záměn:

$$\mathrm{J} = \frac{TP}{TP + FN + TP + FP - TP} = \frac{TP}{TP + FN + FP}. \tag{2.18}$$

Jaccardův index je velice podobný metrice SMC (angl. Simple Matching Coefficient, 2.19), ale na rozdíl od ní nebere v potaz míru správně určených negativních vzorků. Dosahuje tak mnohem více vypovídajících výsledků v případech, kdy značná část vzorků náleží právě do negativní třídy.

$$\mathrm{SMC} = \frac{TP + TF}{TP + FN + FP + TF} \tag{2.19}$$

Stejně jako Diceův koeficient, i Jaccardův index není definovaný v případě $TP = FN = FP = 0$, a tak opět přičítám konstantu ε ke jmenovateli.

$$\mathrm{J} = \frac{TP}{TP + FN + FP + \varepsilon} \tag{2.20}$$

Další podobností s Diceovým koeficientem je, že Jaccardův index dosahuje v ideálním případě hodnoty 1. I tady je tedy potřeba použít jako ztrátovou funkci jeho komplementární hodnotu – Jaccardovu vzdálenost:

$$\mathrm{d_J}(X, Y) = 1 - \mathrm{J}(X, Y). \tag{2.21}$$

Výpis 2.2: Implementace Jaccardova indexu a vzdálenosti

```
def jaccard_index(y_true, y_pred):
    eps = K.epsilon()
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    union = K.sum(y_true_f) + K.sum(y_pred_f) - intersection
    return intersection / (union + eps)

def jaccard_distance(y_true, y_pred):
    return 1 - jaccard_index(y_true, y_pred)
```

2.5.5 Fokální ztráta

Fokální ztráta (angl. *Focal Loss*) byla navrhnuta v práci [4] přímo za účelem vypořádání se s problémem nevyvážených dat u jednofázových detektorů.

Fokální ztráta vychází z binární cross-entropie, ale přidává faktory α a γ. Balanční faktor α byl přebrán z balancované cross-entropie, soustřeďovací faktor γ byl přidán proto, aby klasifikace časté třídy neměla takový vliv.

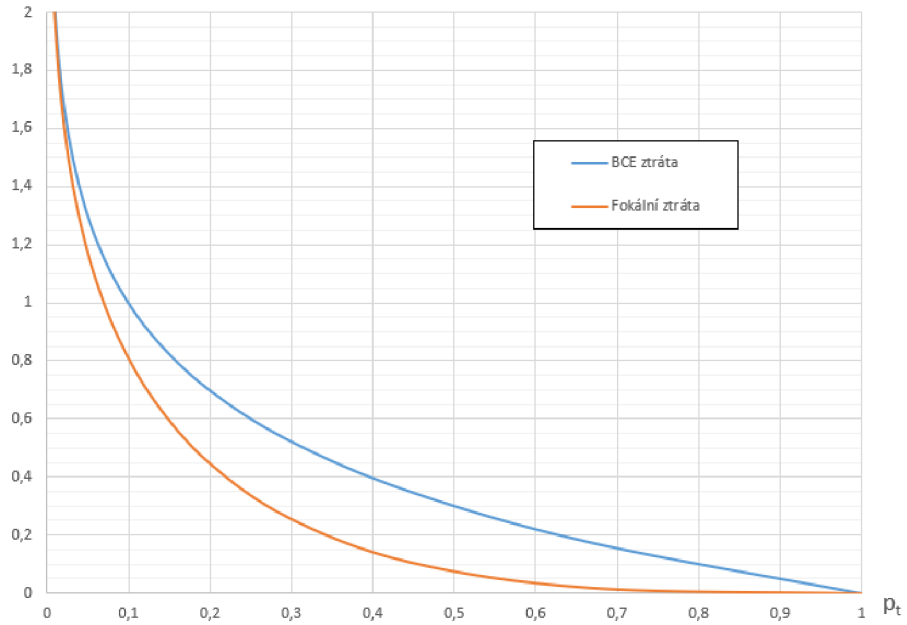
Fokální ztráta každého prvku vychází ze vztahu 2.8 pro binární cross-entropii. Opět, jako v předchozím případě, pro přehlednost definuji predikovanou pravděpodobnost náležitosti prvku y do správné třídy

$$p_\mathrm{t} = f(p, y) = \begin{cases} p & \text{pokud } y = 1 \\ 1 - p & \text{jinak} \end{cases} \tag{2.22}$$

a funkci fokální ztráty vyjádřím jako

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t). \tag{2.23}$$

Lze si povšimnout, že čím větší bude pravděpodobnost výskytu třídy p_t, tím menší bude vliv tohoto prvku na celkovou hodnotu ztrátové funkce. Tuto vlastnost sice má už binární cross-entropie, ale fokální ztráta poskytuje ještě větší penalizaci dobře klasifikovaných prvků. Experimentálně se tvůrcům této ztrátové funkce osvědčilo používat hodnoty faktorů $\alpha = 0,25$ a $\gamma = 2$. Srovnání fokální ztráty a binární cross-entropie jde vidět na grafu 2.4 (bez použití balančního faktoru α).



Obr. 2.4: Srovnání fokální ztráty a binární cross-entropie v závislosti na p_t

Zprvu jsem používal implementaci uživatele MKocabas, kterou jsem naleznul na serveru GitHub [21]. Později jsem funkci přepsal podle sebe a ověřil výsledky podle původní implementace. Přišel jsem totiž na dva problémy dané implementace fokální ztráty:

1. Hodnota ztrátové funkce je kumulativní, a je tedy závislá na počtu zkoumaných prvků. Kvůli tomu bude fokální ztráta nabývat větších hodnot při nasazení na větších blocích dat, a tudíž se hodnoty mezi sebou budou špatně porovnávat.
2. Není ošetřen případ, kdy se v celé zkoumané množině vzorků nenachází ani jeden pozitivní vzorek – v tom případě bude platit $p_1 = 0$ a bude se počítat logaritmus nuly, jehož výsledkem bude $-\infty$.

První problém jsem vyřešil přepsáním funkce tak, aby vracela průměrnou hodnotu namísto součtu.

Druhý problém jsem ošetřil přičtením nejmenšího možného čísla k počítanému logaritmu, aby nedocházelo k počítání log(0). Výsledná funkce sice bude nabývat nepatrně nižších hodnot, ale závislost na správných odhadech zůstává zachována.

Výsledný vztah pro použitou fokální ztrátu je tedy

$$\mathrm{FL}_{\mathrm{norm}} = -\frac{1}{N} \cdot \sum^{N} \alpha_{\mathrm{t}}(1 - p_{\mathrm{t}})^{\gamma} \log(p_{\mathrm{t}} + \varepsilon). \tag{2.24}$$

Výpis 2.3: Implementace fokální ztráty

```
def focal_loss(y_true, y_pred, alpha=0.25, gamma=2.):
    eps = K.epsilon()
    a_t = tf.where(tf.equal(y_true, 1),
          tf.zeros_like(y_true) + alpha,
          tf.ones_like(y_true) - alpha)
    p_t = tf.where(tf.equal(y_true, 1),
          y_pred, tf.ones_like(y_pred) - y_pred)
    return -K.mean(a_t * K.pow(1. - p_t, gamma) * K.log(p_t + eps))
```

2.5.6 Kombinace DSC a fokální ztráty

Rozhodl jsem se také otestovat kombinaci Dice koeficientu a fokální ztráty. Tato kombinace byla navržena v práci [22] a je vyjádřena vztahem 2.25. Jejich experimenty nasvědčují tomu, že tato kombinace dosahuje lepších výsledků, než jednotlivé funkce samotné.

$$\mathrm{DFL} = \mathrm{DL} + \lambda \mathrm{FL} \tag{2.25}$$

Jedná se o jednoduchý součet Dice koeficientu a fokální ztráty v určitém poměru, který je nastavitelný parametrem λ.

Této ztrátové funkce se dotýká stejný problém jako fokální ztráty, a to ten, že při větším množství prvků se bude fokální ztráta zvětšovat, zatímco Dice koeficient bude stále stejný. To by způsobovalo různé výsledky na různě velkých blocích dat. Z tohoto důvodu jsem vztah 2.25 upravil tak, aby používal normalizovanou fokální ztrátu. Navíc jsem upravil použití parametru λ tak, že v případě nastavení $\lambda = 0$ bude ztrátová funkce rovna funkci DL a při $\lambda = 1$ bude rovna funkci $\mathrm{FL}(X, Y)$.

Výsledná funkce vypadá následovně:

$$\mathrm{DFL}(X, Y) = (1 - \lambda) \cdot \mathrm{DL}(X, Y) + \lambda \mathrm{FL}(X, Y). \tag{2.26}$$

2.5.7 Střední teseraktická chyba

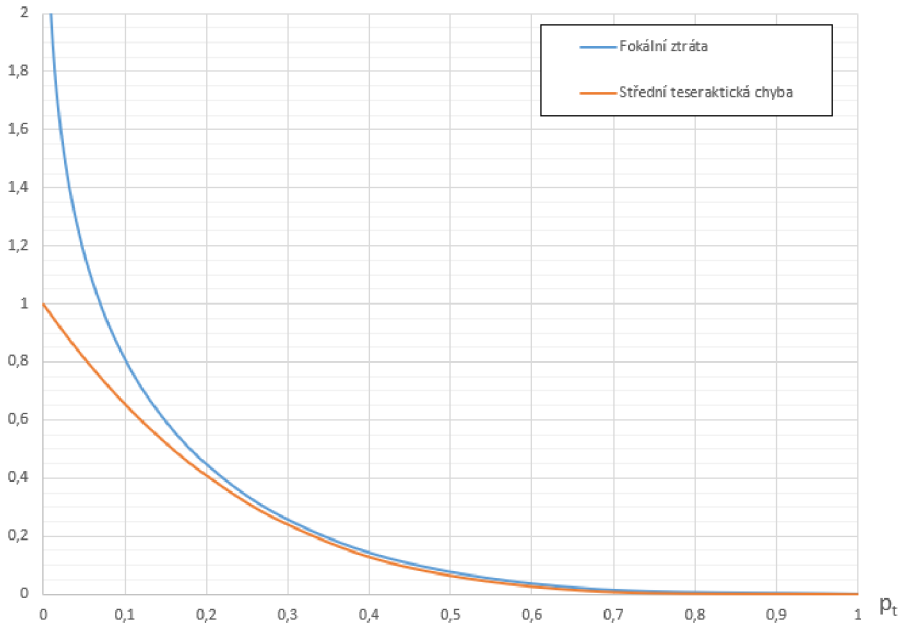
U fokální ztrátové funkce si můžeme povšimnout, že se jedná o součin kvadratické chyby a logaritmické ztráty. Zkusil jsem navrhnout funkci, která bude mít podobný charakter jako fokální ztráta, ale nebude mít nešvar v podobě

$$\lim_{pt \to 0} \mathrm{FL}(p_t) = \infty. \tag{2.27}$$

Upravil jsem funkci střední kvadratické chyby tak, že namísto chyby na druhou, počítá s chybu na čtvrtou. Pojmenoval jsem ji analogicky podle teseraktu, což je čtyřrozměrná analogie čtverce. Navržená funkce má tvar

$$\mathrm{MTE}(p_t) = \frac{1}{N} \cdot \sum^{N} (1 - p_t)^4. \tag{2.28}$$

a její graf je zobrazen na obr. 2.5. Pro interval $p_t \in \langle 0{,}1; 1 \rangle$ téměř kopíruje fokální ztrátu a hlavní rozdíl nastává až v případech, kdy je daný prvek velmi špatně klasifikován. Fokální ztráta se v tomto případě asymptoticky blíží nekonečnu, zatímco teseraktická chyba konverguje k hodnotě 1.



Obr. 2.5: Srovnání fokální ztráty a střední teseraktické chyby

2.6 Testování

Implementovné metody jsem vyzkoušel na dostupných datech.

2.6.1 Trénování sítě

Pro učení neuronové sítě jsem připravil skript `train.py`. Dobu trénování sítě jsem nastavil na 20 epoch, přičemž jedna epocha značí jednu iteraci přes všechna trénovací data. Před začátkem učení se trénovací data promíchají, a následně se vyčlení 20% dat jako validační množina, která slouží k prevenci proti přeučení sítě. V době trvání každé epochy se použijí k učení všechna trénovací data kromě validačních a na konci každé epochy se testuje síť na validační množině.

Učení jsem spouštěl na serverovém počítači s grafickou kartou nVidia GeForce 1080 Ti, přičemž proces běžel přibližně 30 minut. Ve složce se v průběhu učení vytváří soubor `weights.h5`, který uchovává váhy nejlepšího dosaženého modelu, tedy toho, který dosahoval nejlepších výsledků ztrátové funkce na validační množině.

2.6.2 Vyhodnocení výsledků

Keras nabízí možnost jednoduše vypočítat metriky naučeného modelu pomocí funkce `keras.models.evaluate`. Tato metoda má však stejné omezení, jako má trénování modelu – nedokáže do paměti načíst celou testovanou množinu dat najednou, ale bere ji po částech a spočítané metriky průměruje. To by způsobovalo zkreslení velké části vyhodnocovaných metrik, zejména Diceova koeficientu a Jaccardova indexu.

Proto jsem vytvořil skript `predict.py`, jenž načte váhy `weights.h5` uložené při trénování sítě, predikuje výsledky a na jejich základě vytvoří sérii snímků do složky `preds`. V případě 3D segmentace může nastávat, že se jeden pixel odhaduje hned v několika blocích – v tomto případě se jako směrodatný bere aritmetický průměr predikovaných hodnot všech překrývajících se bloků.

K vyhodnocení metrik jsem využil program `EvaluateSegmentation` [23], který dokáže vyhodnotit až 22 různých metrik. Program očekává parametrem předané dva snímky: masku a predikci. Využil jsem toho, že formát tiff dokáže pojmout více snímků zároveň a program s ním umí pracovat – z masek jsem vytvořil jediný soubor ve formátu tiff, a to samé jsem vždy provedl s predikovanými snímky.

Program navíc umožňuje snímky před měřením hodnot prahovat – jako práh jsem zvolil střední hodnotu. Vytvořené tiff soubory jsem předal programu a nechal jsem vyhodnotit hodnoty Diceova koeficientu a Jaccardova indexu – nejprve bez nastaveného prahu, poté s ním.

Rozhodl jsem se otestovat ztrátové funkce

- **BCE** – binární cross-entropie,
- **DL** – doplňková hodnota DSC,
- **FL** – Fokální ztráta s parametry $\alpha = 0{,}25$ a $\gamma = 2$,
- **DFL01** – Kombinace DL a FL s parametrem $\lambda = 0,1$,
- **DFL05** – Kombinace DL a FL s parametrem $\lambda = 0,5$,
- **DFL09** – Kombinace DL a FL s parametrem $\lambda = 0,9$,
- **DFL09** – Kombinace DL a FL s parametrem $\lambda = 0,99$ a
- **MTE** – střední teseraktická chyba.

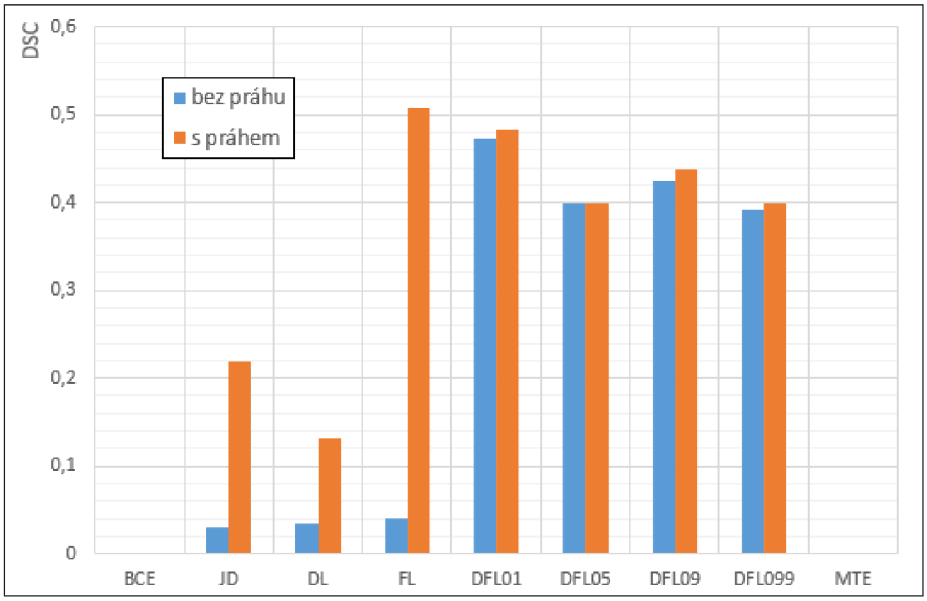
Testování metrik jsem prováděl na 2D segmentační síti s velikostí dávky `batch_size=16`. Výsledky jsou vypsány v tabulce 2.2 a graficky znázorněny na grafu 2.6.

Tab. 2.2: Výsledky ztrátových funkcí při použití 2D segmentace

	Před prahováním		Po prahování	
Ztrátová funkce	DSC	J	DSC	J
BCE	0	0	0	0
DL	0,0342	0,0174	0,1316	0,0704
d_J	0,0304	0,0154	0,2193	0,1232
FL	0,0410	0,0209	0,5070	0,3400
DFL01	0,4740	0,3107	0,4820	0,3175
DFL05	0,3993	0,2494	0,3989	0,2492
DFL09	0,4250	0,2699	0,4374	0,2799
DFL099	0,3915	0,2434	0,3998	0,2499
MTE	0	0	0	0

Z výsledků jde vidět, že běžně používaná ztrátová funkce binární cross-entropie BCE nedokázala správně označit žádná ložiska roztroušené sklerózy, a tudíž dosahovala hodnot DSC i Jaccardova indexu rovno 0. Tahle situace nastává z toho důvodu, že celková hodnota ztrátové funkce pochází převážně z dobře klasifikovatelných vzorků. BCE používá logaritmickou ztrátu, která sice dává větší váhu špatně klasifikovaným vzorkům, ale při takto velkém nepoměru tříd „snadná“ třída stále dominuje.

V prvních testech vycházely výsledky při použití DL i d_J stejně jako BCE. To se napravilo upravením ztrátových funkcí podle rovnic 2.15 a 2.20. Při učení sítí s těmito ztrátovými funkcemi jsem u obou zpozoroval podobný problém – většinu doby se ztráta viditelně nezmenšovala, dokud nedosáhla bodu, ve kterém začala rychle klesat. V tomto svém „vrcholu“ se však ani v jednom případě neudržela dost dlouho, a po chvíli se síť zase navrátila do stavu s maximální hodnotu ztrátové funkce. Díky vahám uložených v těchto světlých chvilkách se síť dokázala naučit

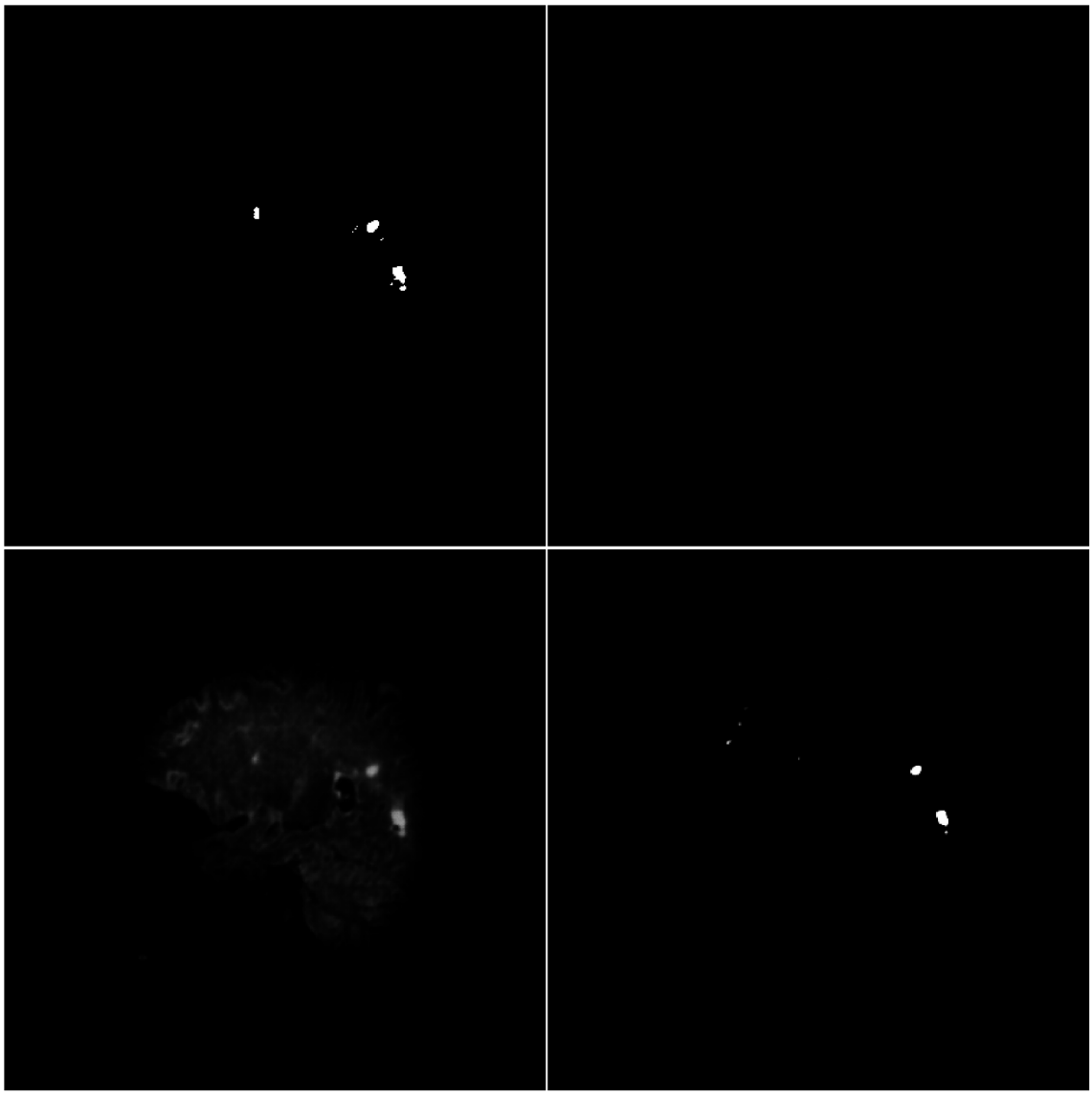


Obr. 2.6: Graf výsledků sítě při použití různých ztrátových funkcí

segmentovat některá ložiska sklerózy, ale stále nedosahuje tak dobrých výsledků jako při použití funkcí s fokální ztrátou.

Problém popsaný u BCE se snaží řešit fokální ztráta FL. Díky soustřeďovacímu faktoru dává dobře klasifikovaným prvkům ještě menší váhu, než samotná BCE. V grafu 2.6 si můžeme všimnout, že při použití FL dosahuje síť jen o něco málo lepších výsledků, než DL a d_J, ale to jen do chvíle, než se výsledky prahují. Toto chování vychází z principu fokální ztráty a z toho, jakým způsobem funguje počítání metrik na obrázcích ve stupních šedi programu EvaluateSegmentation, jež bere jakýkoliv nenulový bod jako pozitivum. Fokální ztráta je dělaná tak, aby se nesoustředila na dobře klasifikované pixely, a tak nemá tendenci negativa značit nulou, ale jen nějakou nízkou hodnotou, přičemž další zlepšování už pro má velmi malou důležitost.

Výše popsané chování fokální ztráty napravuje přičtení Diceova koeficientu ke ztrátové funkci. Při trénování jsem si nechal vypisovat jednotlivé složky (FL a DSC), na nichž bylo krásně vidět, že pro síť je jednodušší snižovat hodnotu fokální ztráty než hodnotu DL – vždy se nejprve snižovala pouze hodnota FL, a až byla síť dostatečně naučená rozpoznávat ložiska, teprve tehdy se začala zvyšovat hodnota DSC. Výsledné predikované masky jsou mnohem kontrastnější než při použití samotné funkce FL a i v tabulce 2.2 jde vidět, že prahování masek získaných pomocí DFL už mělo jen velmi malý přínos (u DFL05 dokonce lépe vycházely neprahované masky).



Obr. 2.7: Srovnání predikovaných masek – zleva po řádcích: skutečná maska, maska predikovaná pomocí BCE, pomocí FL a pomocí DFL09

2.6.3 Prahování masek

Pro masky predikované pomocí fokální ztráty jsem testoval různé nastavení práhu. K prahování masek jsem využil program Aliza [24]. Ten dokáže načíst sérii snímků a zobrazit je jako 3D model a umožňuje na ně aplikovat různé operace. Využil jsem funkci binárního prahování s manuálním nastavením hodnot. Hodnota prahu udává, od jaké hodnoty pixelu se pixel začíná klasifikovat jako pozitivní (nastaví se na maximální hodnotu, ostatní se nastaví na nulu). Práh jsem nastavoval na po 16 v intervalu hodnot 16-240, přičemž jsem pokaždé nechal vypočítat metriky výsledných masek. Výsledky jsem zpracoval do tabulky 2.3 a důležité hodnoty vynesl do grafu 2.9.

Obr. 2.8: Srovnání výsledků prahování – nalevo skutečná maska, pak predikové masky s prahy 48, 80 a 112

Lze vidět, že se zvyšujícím se prahem se zvyšuje preciznost, ale snižuje citlivost masek. Například při použití prahu 32 maska správně označovala 85,9% pixelů, na kterých se roztroušená skleróza skutečně nacházela, ale na druhou stranu jen 11,6% z celkového počtu bílých pixelů skutečně značilo roztroušenou sklerózu.

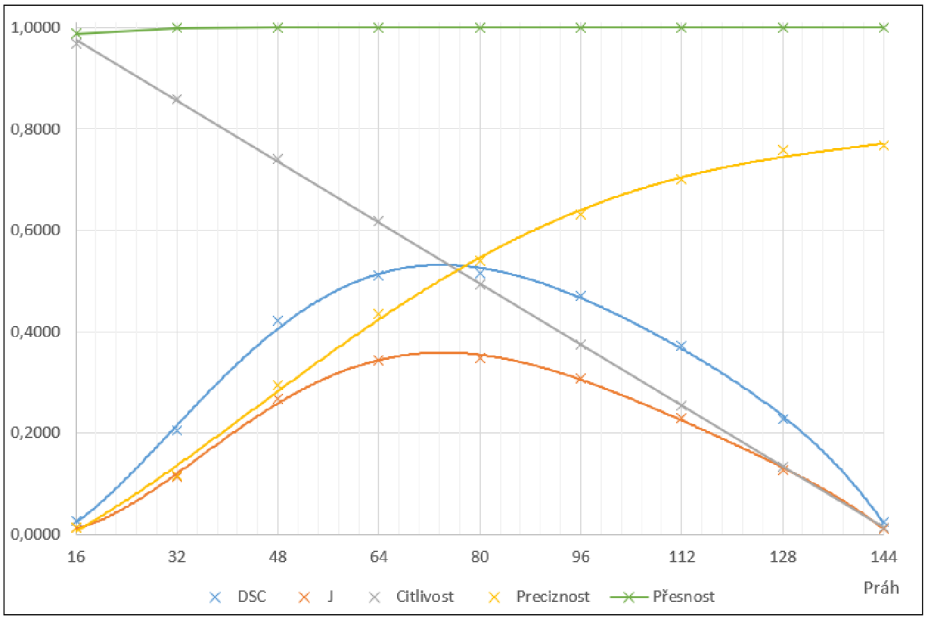
Můžeme si povšimnout, že citlivost klesala lineárně s použitým prahem, ale rychlost stoupání preciznosti se postupně snižovala. Při použití prahu vyššího než 160 už byly veškeré pixely klasifikovány jako negativní třída, a tudíž se preciznost již nedala spočítat a citlivost vycházela nulová.

Diceův koeficient nebo Jaccardův index se jeví jako vhodné kompromisy mezi precizností a citlivostí – svého maxima dosahují při prahu nastaveném mezi hodnotami 64 a 80 v místě, kde se protínají křivky preciznosti a citlivosti.

Naopak, je možno z tabulky vyčíst, že přesnost se jako metrika pro posuzování podobnosti obrazů moc nehodí, neboť ačkoliv bylo ve všech případech správně klasifikováno víc jak 99,9% pixelů, ze segmentované masky nezískáme žádnou informaci.

Tab. 2.3: Dopad nastavení práhu na metriky

Práh	DSC	J	Citlivost	Preciznost	Přesnost
16	0,0254	0,0129	0,9675	0,0129	0,9876
32	0,2049	0,1142	0,8590	0,1163	0,9989
48	0,4211	0,2667	0,7403	0,2942	0,9997
64	0,5107	0,3429	0,6170	0,4357	0,9998
80	0,5159	0,3476	0,4937	0,5403	0,9998
96	0,4708	0,3078	0,3751	0,6319	0,9998
112	0,3733	0,2294	0,2545	0,7000	0,9999
128	0,2264	0,1277	0,1331	0,7579	0,9999
144	0,0237	0,0120	0,0122	0,7664	0,9998
160	0,0006	0,0003	0,0003	0,5000	0,9998
176	0,0000	0,0000	0,0000	—	0,9998
192	0,0000	0,0000	0,0000	—	0,9998
208	0,0000	0,0000	0,0000	—	0,9998
224	0,0000	0,0000	0,0000	—	0,9998
240	0,0000	0,0000	0,0000	—	0,9998



Obr. 2.9: Graf vlivu prahu na metriky

2.6.4 Testování 3D segmentace

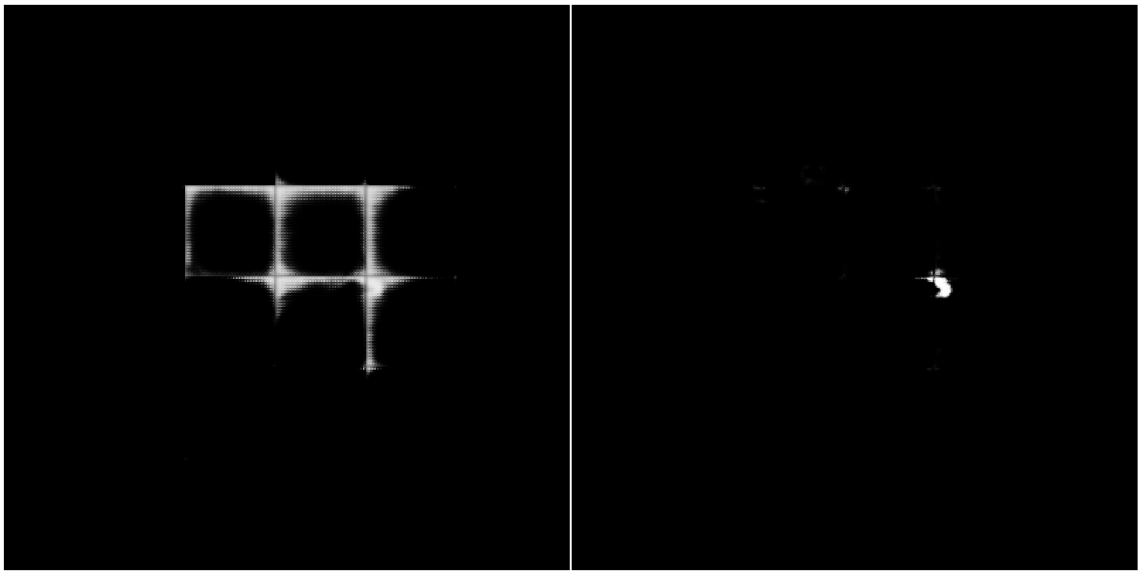
Pro porovnání výsledků 2D a 3D segmentace jsem využil data z předchozího testování 2D sítě a pro testování 3D segmentace jsem vytvořil dvě různé konfigurace sítě:

- 3D pláty o velikosti $400 \times 400 \times 16$ pixelů a
- 3D bloky o velikosti $64 \times 64 \times 64$ pixelů.

Tab. 2.4: Výsledky při použití 3D segmentace ($64 \times 64 \times 64$)

	Před prahováním		Po prahování	
Ztrátová funkce	DSC	J	DSC	J
DL	0,2323	0,1314	0,2654	0,1530
FL	0,0056	0,0028	0,1808	0,0994
DFL01	0,0120	0,0060	0,0146	0,0074
DFL05	0,0365	0,0186	0,0536	0,0276
DFL09	0,2329	0,1318	0,2696	0,1558

Krychlové bloky Na predikovaných maskách pří použití 3D bloků jde vidět „švy“ mezi jednotlivými bloky, což je způsobeno právě nedostatečným naučením sítě – čím lépe byla síť naučená, tím méně šly švy vidět (viz 2.10). Dalším nedostatkem tohoto přístupu je, že síť neví, ze které části skenu dané bloky pochází, a tak se občas stává, že síť značí ložiska roztroušené sklerózy i mimo mozek (např. v dásních).

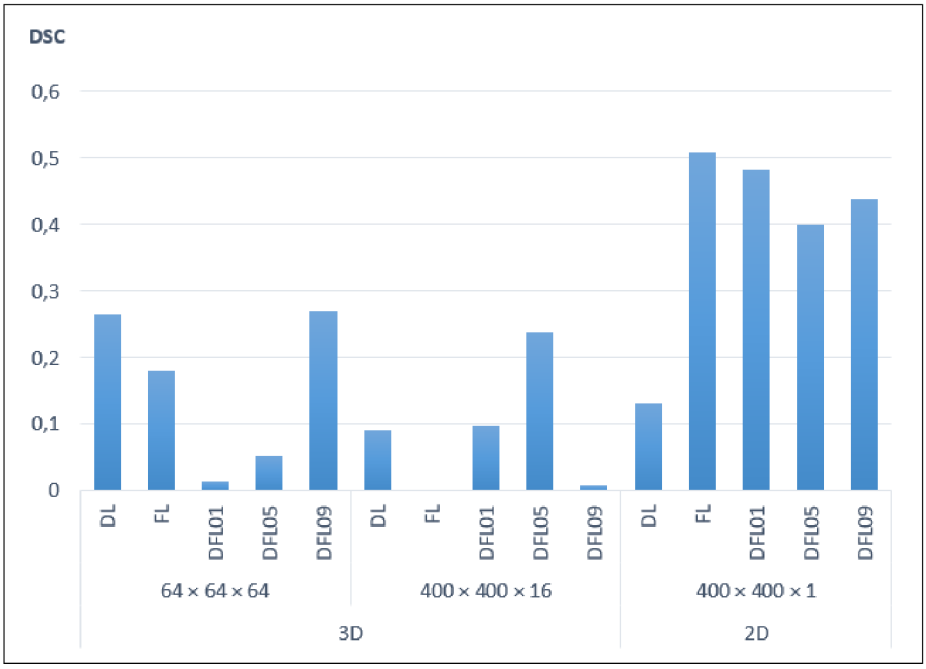


Obr. 2.10: Švy mezi bloky při použití DFL01 a DFL09

3D pláty Při použití 3D plátů o rozměrech $400 \times 400 \times 16$ jsem musel snížit dávku na jeden blok (`batch_size=1`) – pro vyšší množství už grafická karta neměla dostatek paměti.

Tab. 2.5: Výsledky při použití 3D segmentace ($400 \times 400 \times 16$)

	Před prahováním		Po prahování	
Ztrátová funkce	DSC	J	DSC	J
DL	0,0295	0,0150	0,0920	0,0482
FL	0,0022	0,0011	0,0000	0,0000
DFL01	0,0470	0,0223	0,0973	0,0511
DFL05	0,0625	0,0323	0,2388	0,1356
DFL09	0,0043	0,0021	0,0079	0,0040



Obr. 2.11: Srovnání výsledků ztrátových funkcí na různých sítích

Na obrázku 2.11 lze vidět, že výsledky 2D segmentace dopadaly konzistentně mnohem lépe, než výsledky 3D segmentace (s výjimkou u DL, jehož učení bylo nestabilní i v ostatních případech), přičemž výsledky dopadaly o něco lépe při použití krychlových bloků oproti plátům. Způsobeno je to tím, že použité pláty jsou zhruba desetkrát větší než bloky, takže je jich v konečném důsledku mnohem méně a síť tedy neměla dostatečné množství dat ke svému naučení. To je způsobeno tím, že 3D segmentační síť má přibližně třikrát více naučitelných parametrů, a tudíž je náročnější ji naučit.

2.6.5 Překrývání bloků

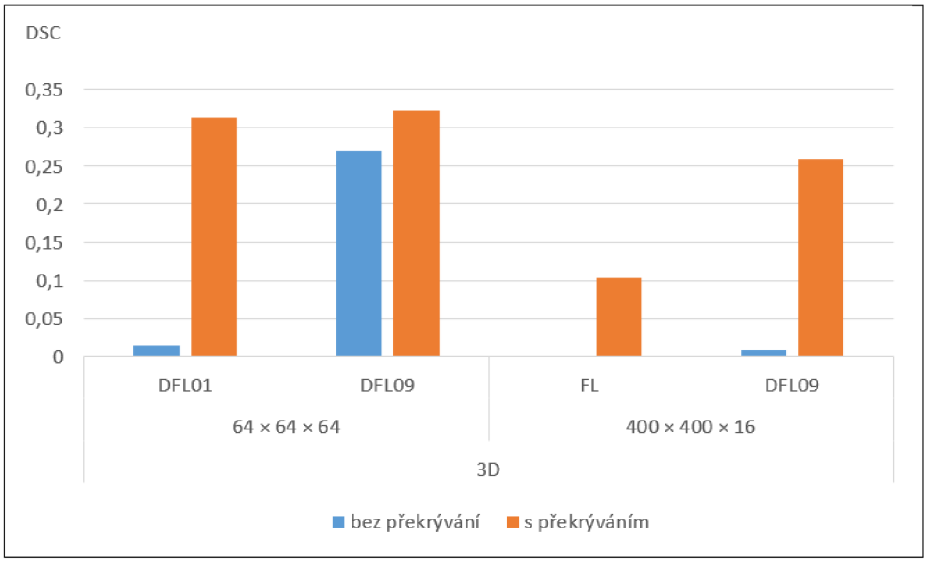
Zlepšení výsledků 3D bylo dosaženo použitím menší střídy bloků – tím se navýšil počet trénovacích bloků. U 3D bloků jsem volil použití střídy $32 \times 32 \times 32$, což navýšilo celkový počet trénovacích bloků téměř na osminásobek. U 3D plátů jsem zkusil použít střídu $400 \times 400 \times 8$, čímž se zvýšil celkový počet 3D plátů téměř na dvojnásobek, přičemž se každý plát vždy z poloviny překrýval s předchozím.

Přímoúměrně k počtu trénovacích bloků však narůstala i doba trvání epochy a celková doba učení. Navíc, jelikož data pochází pořád ze stejných mozků, roste s menší střídou taky hrozba přeučení.

Výsledky dopadly o něco lépe, ale i přesto síť nedosahovala tak dobrých výsledků jako při použití 2D segmentace.

Tab. 2.6: Výsledky 3D segmentace při překrývajících se blocích

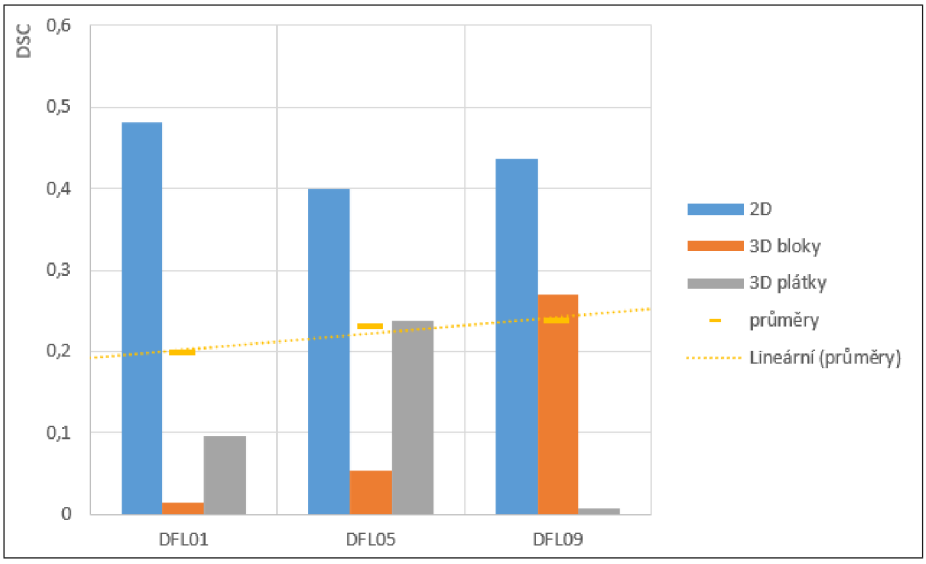
Velikost bloků	Ztrátová funkce	DSC	J
$64 \times 64 \times 64$	DFL01	0,3141	0,1863
	DFL09	0,3223	0,1921
$400 \times 400 \times 16$	FL	0,1029	0,0542
	DFL09	0,2587	0,1486



Obr. 2.12: Výsledky před a po snížení střídy 3D bloků

2.6.6 Nastavování parametru λ funkce DFL

Pokusil jsem se porovnat hodnoty ztrátové funkce DFL (2.26) v závislosti na parametru λ funkce. Ze změřených hodnot v předchozích testech jsem pro tento účel vytvořil graf 2.13, ale jak jde vidět, rozptyl hodnot Dice koeficientu pro různá nastavení a různé architektury sítě je obrovský, a tak i přesto, že průměrná hodnota výsledného Diceova koeficientu mírně stoupá s vyšším nastavením parametru λ, jsou výsledky statisticky nevýznamné.



Obr. 2.13: Výsledky hybridní ztrátové funkce v závislosti na parametru λ

Závěr

V práci byl prezentován problém klasifikace nevyvážených dat, byla prozkoumaná existující řešení, a v návaznosti na to byla navržena a implementována umělá neuronová síť, která je schopná ze snímků z magnetické rezonance mozku rozpoznat a vyznačit ložiska roztroušené sklerózy.

Trénování sítě probíhalo na silně nevyváženém setu dat, ve kterém je poměr pozitivní ku negativní třídě více než 1:6000, navíc se trénovací množina skládala pouze ze snímků sedmi mozků po 256 snímcích, což dále znesnadňuje učení sítě.

Při trénování sítě za použití standardní ztrátové funkce binární cross-entropie, se neuronová síť naučila veškeré pixely klasifikovat jako negativní třídu, čímž dosahovala zdánlivě dobrých výsledků, avšak užitečná informace byla nulová.

Trénování pomocí inverzních hodnot Diceova koeficientu a Jaccardova indexu se ukázalo být nestabilní, což způsobovalo velmi nekonzistentní výsledky – někdy vycházely dobře, jindy nikoliv.

Hlavního pokroku bylo dosaženo až díky použití fokální ztrátové funkce popsané v [4], která předchází největšímu problému silně nevyvážených dat, a to posouzení kvality predikce sítě. Po provedení vhodného prahování výsledků dosahují predikované výsledky Diceova koeficientu až 0,516, což je s ohlédnutím na náročnost problému a malé množství trénovacích dat velice dobrý výsledek.

Nevýhodou učení pomocí fokální ztráty je nutnost prahovat predikované snímky, přičemž volba vhodného prahu může být obtížná. Tento problém se podařilo značně potlačit zkombinováním fokální ztráty s Diceovým koeficientem (resp. jeho komplementární hodnotou). Výsledné predikované masky jsou mnohem kontrastnější a prahování již není potřeba.

V práci byla také vyzkoušena architektura sítě pro 3D segmentaci obrazu. Ukázalo se ale, že takovouto síť je mnohem obtížnější naučit než 2D segmentační a na její plné naučení by bylo potřeba více času nebo více trénovacích dat.

Výsledek této práce má přínos nejen pro segmentaci roztroušené sklerózy, ale obecně pro jakýkoliv klasifikační problém, kdy jednotlivé třídy nejsou zcela rovnoměrně zastoupeny – fokální ztráta přináší zlepšení nejen na silně nevyvážených datech, ale i na množinách dat, kde nepoměr tříd není tak markantní.

Literatura

[1] Redmon, J., Divvala, S. K., Girshick, R. B. et al. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, abs/1506.02640. Dostupné na: <http://arxiv.org/abs/1506.02640>.

[2] Liu, W., Anguelov, D., Erhan, D. et al. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, abs/1512.02325. Dostupné na: <http://arxiv.org/abs/1512.02325>.

[3] Sermanet, P., Eigen, D., Zhang, X. et al. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *CoRR*. 2013, abs/1312.6229. Dostupné na: <http://arxiv.org/abs/1312.6229>.

[4] Lin, T., Goyal, P., Girshick, R. B. et al. Focal Loss for Dense Object Detection. *CoRR*. 2017, abs/1708.02002. Dostupné na: <http://arxiv.org/abs/1708.02002>.

[5] Girshick, R. B., Donahue, J., Darrell, T. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, abs/1311.2524. Dostupné na: <http://arxiv.org/abs/1311.2524>.

[6] Kingma, D. P. a Ba, J. Adam: A Method for Stochastic Optimization. *CoRR*. 2014, abs/1412.6980. Dostupné na: <http://arxiv.org/abs/1412.6980>.

[7] Dozat, T. Incorporating Nesterov Momentum into Adam. In. 2016.

[8] Minsky, M. L. a Papert, S. Perceptrons: an introduction to computational geometry. 1969.

[9] Dreuw, P. *Diplomarbeit im Fach Informatik Improved Modeling in Handwriting Recognition*. 2009.

[10] Hochreiter, S. a Schmidhuber, J. Long short-term memory. *Neural computation*. 1997, roè. 9, è. 8. S. 1735–1780.

[11] Long, J., Shelhamer, E. a Darrell, T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015. S. 3431–3440.

[12] Ronneberger, O., Fischer, P. a Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR*. 2015, abs/1505.04597. Dostupné na: <http://arxiv.org/abs/1505.04597>.

[13] Huang, G., Liu, Z. a Weinberger, K. Q. Densely Connected Convolutional Networks. *CoRR*. 2016, abs/1608.06993. Dostupné na: <http://arxiv.org/abs/1608.06993>.

[14] Jégou, S., Drozdzal, M., Vázquez, D. et al. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. *CoRR*. 2016, abs/1611.09326. Dostupné na: <http://arxiv.org/abs/1611.09326>.

[15] *How 3D object is created from 2D slices and mask*. Dostupné na: <http://splab.cz/wp-content/gallery/3d-images/2dto3d.png>.

[16] Abadi, M., Agarwal, A., Barham, P. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Dostupné na: <http://tensorflow.org/>.

[17] Chollet, F. et al. *Keras* [https://keras.io]. 2015.

[18] Walt, S. van der, Schönberger, J. L., Nunez-Iglesias, J. et al. Scikit-image: image processing in Python. *PeerJ*. èerven 2014, roè. 2. S. e453. Dostupné na: <http://dx.doi.org/10.7717/peerj.453>. ISSN 2167-8359.

[19] Oliphant, T. *NumPy: A guide to NumPy* [USA: Trelgol Publishing]. 2006–. [Online; accessed <today>]. Dostupné na: <http://www.numpy.org/>.

[20] Shibuya, N. *Up-sampling with Transposed Convolution*. 2017. Dostupné na: <https://bit.ly/2UHMu5O>.

[21] Kocabas, M. *Focal Loss implementation in Keras*. 2018. Dostupné na: <https://github.com/mkocabas/focal-loss-keras>.

[22] Zhu, W., Huang, Y., Zeng, L. et al. AnatomyNet: Deep learning for fast and fully automated whole-volume segmentation of head and neck anatomy. *Medical physics*. 2019, roè. 46, è. 2. S. 576–589.

[23] Taha, A. A. a Hanbury, A. Metrics for Evaluating 3D Medical Image Segmentation: analysis, selection, and tool. *BMC Medical Imaging*. August 2015, roè. 15. S. 29.

[24] Imaging, A. M. *Aliza Medical Imaging & DICOM Viewer Application*. 2018. Dostupné na: <https://www.aliza-dicom-viewer.com/>.

Seznam symbolů, veličin a zkratek

ACC	Accuracy, přesnost testu
BCE	Binární cross-entropie
CNN	Konvoluční neuronové sítě
CUDA	Compute Unified Device Architecture
DFL	Dice-Focal Loss, hybridní ztrátová funkce
DL	Dice Loss, Diceova ztráta
DSC	Dice-Sørensenův koeficient
FL	Focal Loss, fokální ztráta
FN	False Negative, falešné negativum
FP	False Positive, falešné pozitivum
GPU	Grafický procesor
J	Jaccardův index
LSTM	Long Short Term Memory
MLP	Vícevrstvý perceptron
PPV	Positive Predictive Value, hodnota pozitivní předpovědi
R-CNN	Regions with CNN features, architektura dvoustupňového detektoru
ReLU	Rectified linear unit, typ aktivační funkce
RNN	Rekurentní (zpětnovazebné) neuronové sítě
RS	Roztroušená skleróza
SMC	Simple Matching Coefficient
SSD	Single Shot Detector, architektura jednostupňového detektoru
TN	True Negative, skutečné negativum
TNR	True Negative Rate, míra skutečných negativ
TP	True Positive, skutečné pozitivum
TPR	True Positive Rate, míra skutečných pozitiv
YOLO	You Only Look Once, architektura jednostupňového detektoru

Seznam příloh

A Přiložené DVD

Na přiloženém DVD se nachází kompletní zdrojový kód nutný pro naučení umělé inteligence a elektronickou verzi této dokumentace.

Ke spouštění skriptů je potřeba mít nainstalované prostředí Pythonu s potřebnými knihovnami, viz kapitolu Instalace prostředí. Skripty se pak dají spouštět skrz PyCharm nebo z příkazové řádky pomocí příkazu `python <název>.py`.

Nejprve je nutné zpracovat data pomocí skriptu `rs_data.py`, poté je možné spustit učení sítě pomocí skriptu `train.py`, a pak je teprve možné spustit predikci testovacích snímků pomocí skriptu `predict.py`.

```
/ ..... kořenový adresář DVD
├── masks
│   └── ... ..... adresáře s maskami mozků
├── results
│   └── ... ..... adresáře s predikovanými snímky
├── test
│   └── ..... adresáře s testovacími snímky
├── train
│   └── ... ..... adresáře s trénovacími snímky
├── dokumentace.pdf ..... pdf dokumentace diplomové práce
├── custom_losses.py ..... vlastní implementace ztrátových funkcí
├── nets.py ..... implementace neuronových sítí
├── predict.py ..... skript pro predikci snímků
├── rs_data.py ..... skript pro zpracování a načítání dat
└── train.py ..... skript pro spuštění učení sítě
```

Obr. A.1: Struktura obsahu DVD