Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Použití metod strojového učení na automatickou segmentaci mikroskopických snímků

2018

Bc. Jan Nováček

Vedoucí práce: RNDr. Eduard Bartl, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Bc. Jan Nováček
Název práce:	Použití metod strojového učení na automatickou segmentaci mikroskopických snímků
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2018
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	RNDr. Eduard Bartl, Ph.D.
Počet stran:	39
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Bc. Jan Nováček
Title:	Microscope image segmentation by means of machine learning methods
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2018
Study field:	Computer Science, full-time form
Supervisor:	RNDr. Eduard Bartl, Ph.D.
Page count:	39
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Pro úlohu segmentace mikroskopických snímků jader buněk jsme navrhli a experimentálně ověřili novou metodu. Nejdříve jsme oddělili jádra buněk od pozadí pomocí prahování a následně jsme od sebe rozdělili jádra buněk, která tvořila klastry. Pro rozdělování klastrů jader buněk jsme využili konvoluční neuronové sítě, pomocí kterých se podařilo dosáhnout vysoké přesnosti. V teoretické části textu práce jsme popsali úlohu segmentace a vysvětlili jsme princip umělých neuronových sítí. Ve dvou závěrečných kapitolách jsme vysvětlili motivaci úlohy, princip nové metody a stručně jsme popsali implementaci ukázkového řešení.

Synopsis

For a microscope image segmentation task we developed and experimentally verified the new method. In a first step, we separated cell nuclei from the background by a thresholding and then we divided clusters of cell nuclei. For the cluster division we used convolutional neural networks by which the high accuracy was achieved. In the theoretical part, we described the segmentation task and also the core principle of artificial neural networks. In the two final chapters we explained the motivation of the task, principle of the new method a also implementation of sample solution was briefly described.

Klíčová slova: segmentace, strojové učení, umělá inteligence, umělé neuronové sítě, konvoluční neuronové sítě, jádra buněk, DAPI

Keywords: segmentation, machine learning, artificial intelligence, artificial neural networks, convolutional neural networks, cell nuclei, DAPI

Děkuji RNDr. Tomášovi Fürstovi Ph.D. a RNDr. Eduardu Bartlovi, Ph.D. za cenné rady, připomínky, ochotu a čas, který mi věnovali.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Cílem této práce bylo vyzkoušet použití nejnovějších metod umělé inteligence a strojového učení (machine learning) pro segmentaci mikroskopických snímků živých jader buněk, která byla obarvena fluorescenčním barvivem DAPI [1]. Vstupem řešeného problému jsou tedy mikroskopické snímky jader buněk. Cílem je ke každému jádru buňky získat seznam pixelů, které danému jádru ve snímku náleží. Hlavní problém, který při segmentaci jader buněk nastává je skutečnost, že jádra buněk se shlukují a tvoří tak klastry. Abychom byli schopni přiřadit ke každému jádru seznam jeho pixelů, je nutné klastry rozdělit, což se ukázalo jako klíčový problém celé úlohy.

Úlohu jsme se rozhodli řešit ve dvou fázích. V první fázi pomocí klasických metod zpracování obrazu (funkce threshold a find contours) najdeme klastry buněk, tedy černobílé masky snímků, kde bílé pixely označují jádra buněk a černé pixely pozadí. V druhé fázi pomocí metod umělé inteligence klastry rozdělíme na jednotlivá jádra buněk. Poznamenejme, že i první fázi bychom mohli řešit pomocí umělé inteligence, která by velmi pravděpodobně dosahovala výrazně vyšší přesnosti, ale z praktických důvodů (vytváření trénovacích dat je pro umělou inteligenci velmi časově náročné) jsme pro proof of concept této úlohy zvolili časově schůdnější variantu.

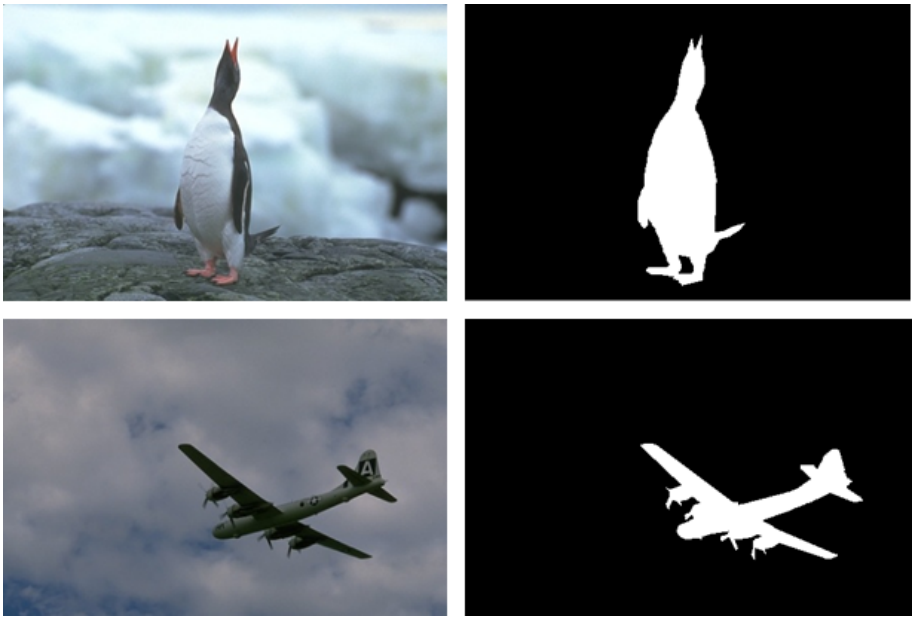
Dataset mikroskopických snímků živých jader buněk byl získán od vědecké skupiny Martina Mistrika z Ústavu molekulární a translační medicíny (UMTM), který patří pod lékařskou fakultu v Olomouci. Vědecká skupina z UMTM se zabývá genotoxicitou a nová metoda pro segmentaci jader buněk a rozdělování klastrů buněk může výrazně zvýšit efektivitu jejich práce.

Pro řešení úlohy rozdělování shluků buněk byly využity konvoluční neuronové sítě, což je jedna z nejmodernějších metod umělé inteligence pro klasifikaci a segmentaci obrazu. Konvoluční neuronové sítě jsou speciálním typem umělých neuronových sítí, které předpokládají, že jejich vstupem jsou obrazová data. Všechny ostatní vlastnosti konvolučních neuronových sítí jsou stejné jako u klasických umělých neuronových sítí [2].

Výstupem práce je nová metoda, jejíž efektivita byla ověřena na úloze segmentace jader buněk, které tvoří klastry. Dodejme, že nová metoda je obecná a lze použít pro rozdělování libovolných objektů[1], které se v obraze shlukují.

Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

[1] Přínos metody je výrazný především pro objekty stejného typu, pro objekty různých typů je úloha snadnější.



Obrázek 1: Vlevo je obrázek, který chceme segmentovat. Vpravo je maska, která je výsledkem segmentace. Zdroj: [3]

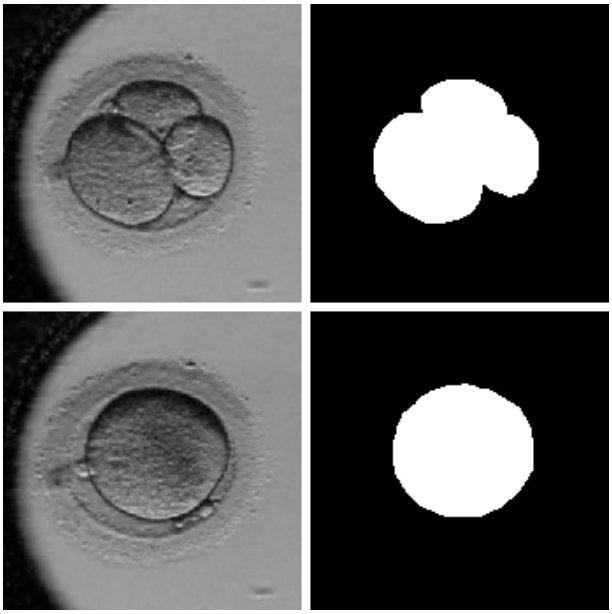
2 Segmentace obrazu

Cílem segmentace obrazu je rozpoznat hledané objekty a přesně určit, které pixely hledanému objektu patří, a které tvoří pozadí. Jinými slovy, pro každý pixel je třeba rozhodnout, zda patří hledanému objektu nebo nepatří. Pro jednoduchost se omezíme na případ, kdy v obraze hledáme pouze jeden typ objektu. V takovém případě může být výstupem segmentace černobílá maska původního obrázku, ve které černé pixely značí pozadí a bílé značí hledaný objekt. Vstup a výstup segmentace obrazu je znázorněn na obrázku 1.

Jak již bylo naznačeno, na úlohu segmentace obrazu se můžeme dívat i jako na klasifikační úlohu, ve které každý pixel obrázku reprezentuje jednu třídu, o které chceme binárně rozhodnout ANO/NE. Zda tedy daný pixel obsahuje hledaný objekt nebo neobsahuje. Dále samozřejmě platí, že se jednotlivé třídy vzájemně nevylučují, tedy více tříd může být klasifikováno jako ANO a více jako NE. Tento pohled na problém segmentace obrazu je velmi výhodný, především pokud chceme pro segmentaci využít umělé neuronové sítě. Jak přesně lze umělé neuronové sítě k segmentaci použít, bude popsáno v následujících kapitolách.

2.1 Aplikace segmentace obrazu

Segmentaci obrazu je možné využít v mnoha odvětvích. Pro představu uvedeme dva praktické příklady využití segmentace. První příklad bude z oblasti zdravotnictví a druhý z oblasti průmyslové výroby.



Obrázek 2: V levém sloupci jsou mikroskopické obrázky embryí a v pravém sloupci jejich segmentace. Zdroj: [5]

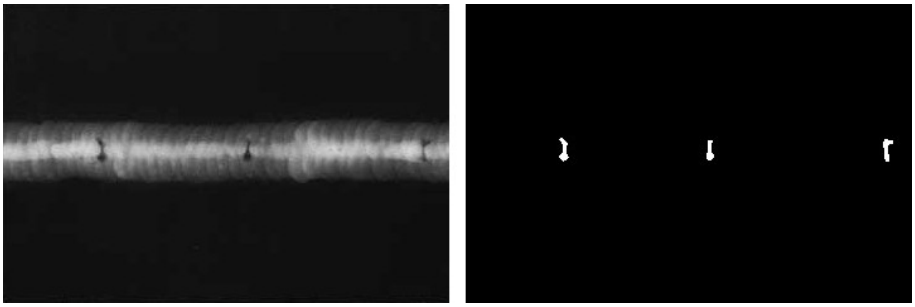
2.1.1 Zdravotnictví

Ve zdravotnictví se segmentace obrazu velmi často využívá pro preprocessing mikroskopických snímků. Příkladem takové aplikace segmentace je například námi řešená úloha, ale nyní popíšeme ještě jinou aplikaci. Jako příklad může sloužit úloha automatického hodnocení embryí, kde na základě velkého množství (tisíců) snímků embryí focených v čase je třeba rozhodnout, které embryo má největší pravděpodobnost úspěchu při umělém oplodnění. Prvním krokem analýzy mikroskopických snímků embryí je oddělení embrya od misky, ve které je embryo umístěno. Tento krok je velmi důležitý, protože výrazně usnadňuje následné vyhodnocování kvality embryí. Příklad mikroskopických snímků embryí spolu s jejich segmentacemi je znázorněn na obrázku 2.

2.1.2 Průmyslová výroba

V dnešní době je velká snaha o průmyslovou automatizaci. Jednou z oblastí průmyslové sériové výroby, kterou je možné velmi dobře automatizovat je nedestruktivní testování (NDT) [6]. Mezi nedestruktivní testy patří například radiografické testy, vizuální testy nebo magnetická prášková metoda. NDT metody produkují velké množství dat (většinou obrazových), které je nutné manuálně vyhodnocovat. Automatizace tedy spočívá v nahrazení lidské síly programem, který je schopen data vyhodnocovat automaticky.

V některých aplikacích je potřeba kromě rozhodnutí, zda testovaný výrobek obsahuje defekt či nikoli, také případný defekt na snímku najít a dále analyzovat.



Obrázek 3: Snímek sváru spolu se segmentacemi defektů. Zdroj: [7]

Jedná se například o změření defektu a porovnání jeho velikosti s předepsanou normou. Tím můžeme zhodnotit závažnost defektu a rozhodnout, zda je potřeba výrobek vyřadit. Abychom byli schopni zjistit, jak velký defekt je, musíme jej nejdříve segmentovat a následně je už snadné jej změřit. Jako příklad uvádíme rentgenový snímek sváru 3 s vyznačenými segmentacemi defektů.

2.2 Metody segmentace

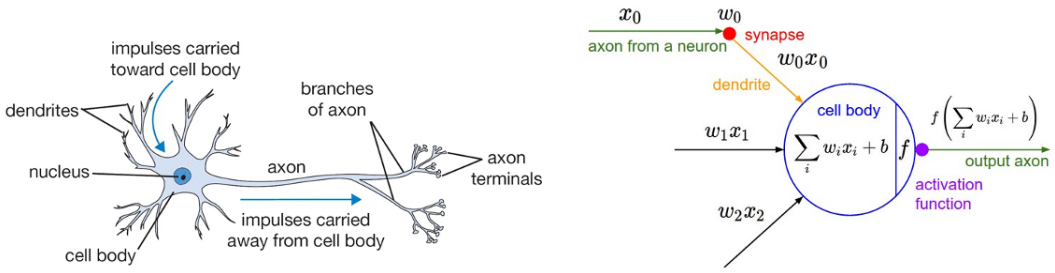
V současné době existuje velké množství metod, které je možné použít pro segmentaci obrazu. Metody lze rozdělit do dvou skupin. První skupinu tvoří klasické algoritmy zpracování obrazu, jako je jednoduché prahování nebo sofistikovanější Wathershed algoritmus. Nejpoužívanější algoritmy z této skupiny jsou většinou implementovány v knihovnách programovacích jazyků. Například pro programovací jazyk Python můžeme použít knihovny OpenCV ve které nalezneme jak funkci pro prahování (`cv2.threshold()`) tak pro Wathershed algoritmus (`cv2.watershed()`). Je samozřejmě možné si vybrané algoritmy naprogramovat ručně, nicméně kvůli rychlosti je většinou výrazně výhodnější využít předchystané implementace. Princip některých algoritmů spadajících do této skupiny lze nalézt například v [4].

Do druhé skupiny patří metody založené na umělé inteligenci. Nejpopulárnější metodou jsou umělé neuronové sítě, které ve zpracování obrazu slaví velký úspěch a pro komplexní úlohy dosahují výrazně vyšší přesnosti než klasické přístupy. Umělé neuronové sítě podrobně popíšeme v následující kapitole.

3 Umělé neuronové sítě

V této kapitole stručně popíšeme koncept umělých neuronových sítí, následně navážeme konvolučními neuronovými sítěmi a nakonec se zaměříme na použití konvolučních neuronových sítí pro zpracování obrazu, konkrétně na klasifikaci a segmentaci obrazu.

Umělé neuronové sítě poprvé zažily velký rozmach v 80. a 90. letech [8], ale doba pro ně ještě nebyla zralá. V dnešní době jsou umělé neuronové sítě



Obrázek 4: Model biologického neuronu (vlevo) a jeho matematická reprezentace (vpravo). Zdroj: [9]

jedním z nejpopulárnějších algoritmů umělé inteligence a to ze dvou důvodů. Prvním důvodem je zvýšení výpočetního výkonu, který potřebujeme, abychom byli schopni natrénovat neuronovou síť v rozumném čase. Druhým důvodem je fakt, že dnes především díky internetu máme k dispozici velké objemy dat, které potřebujeme pro učení neuronových sítí. Kombinací velkého výpočetního výkonu a velkého množství dat jsme schopni neuronové sítě velmi kvalitně naučit a dosáhnout tak vysoké přesnosti. Vysoká přesnost neuronových sítí je hlavním argumentem pro jejich použití namísto jiných přístupů.

Nevýhodou neuronových sítí je jejich neprůhlednost. Nejsme totiž schopni zdůvodnit, proč pro daný vstup dostaneme daný výsledek. Jinými slovy nejsme schopni přesně určit, co se neuronová síť naučila. Tento rys neuronových sítí je problémem například v oblasti bankovnictví, kde je typicky požadavek na zpětné zdůvodnění rozhodnutí prediktivního modelu. Proto se v bankovnictví častěji používají jiné metody, jako jsou například rozhodovací stromy [2].

3.1 Architektura neuronových sítí

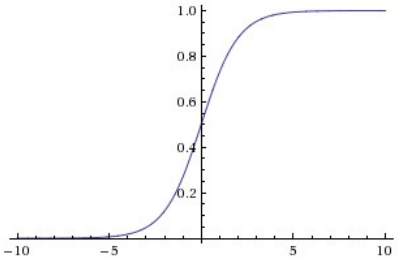
Umělé neuronové sítě reprezentují nelineární výpočetní funkci s velkým počtem parametrů. Neformálně se na funkci můžeme dívat jako na černou skříňku, která přijímá vstup a produkuje výstup. Tento pohled na neuronové sítě je důležitý, abychom si uvědomili, že neuronové sítě nejsou nic jiného než funkce ve speciálním tvaru.

3.2 Model neuronu

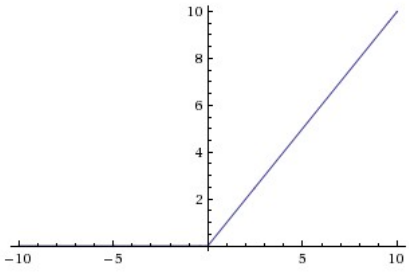
Neuronová síť se skládá z jednotlivých neuronů, což jsou nejjednodušší výpočetní jednotky neuronových sítí. Architektura umělých neuronů je inspirována biologickými neurony. Vztah mezi biologickým a umělým neuronem je znázorněn na obrázku 4.

Neuron je reprezentován číselnými parametry (váhami) w_i a parametrem b, který se nazývá bias a jeho funkce je podobná jako parametr b v lineární funkci

[2]Informace byla získána od člena machine learning týmu nejmenované banky.



Obrázek 5: Graf aktivační funkce sigmoid. Zdroj: [9]



Obrázek 6: Graf aktivační funkce ReLU. Zdroj: [9]

$y = ax+b$. Výpočet samostatného neuronu probíhá jako skalární součin vstupu x a parametrů w plus bias b. Na výsledek skalárního součinu je nakonec aplikována aktivační funkce f. Celkový výpočet pro jeden neuron tedy můžeme zapsat jako

$$S(x) = f\left(\sum_i x_i \cdot w_i + b\right).$$

Jako aktivační funkci f můžeme použít například funkci sigmoid σ, která vrací hodnoty z intervalu 0 a 1 a je definována jako

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

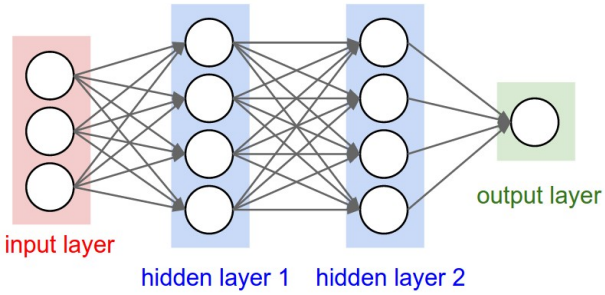
Graf funkce sigmoid je znázorněn na obrázku 5. Další, v dnešní době často používanou aktivační funkcí, je ReLU (Rectified Linear Unit), která je definovaná jako

$$relu(x) = max(0, x)$$

a její graf je znázorněn na obrázku 6.

Pro názornost uvedeme příklad výpočtu jednoho neuronu. Neuron bude reprezentován váhovými parametry $w = [1, 3, 2, 1]$ a bias parametrem $b = 1$. Jako aktivační funkci použijeme sigmoid a vstupní vektor bude $x = [0.5, 1, 0.7, 0.1]$. Výpočet neuronu tedy bude vypadat následovně:

$$S(x) = f\left(\sum_i x_i \cdot w_i + b\right) = \sigma\left((1 \cdot 0.5 + 3 \cdot 1 + 2 \cdot 0.7 + 1 \cdot 0.1) + 1\right) = 0.99753.$$



Obrázek 7: Schéma fully-connected neuronové sítě. Zdroj: [9]

3.3 Neuronová síť

Umělé neuronové sítě [9] jsou sestaveny z neuronů, které jsou organizovány do vrstev a tvoří tak acyklický graf. Cykly v grafu nejsou povoleny, protože by vedly k nekonečnému výpočtu. Nejčastější typ vrstvy neuronové sítě je fully-connected vrstva, ve které jsou neurony ve dvou následujících vrstvách propojeny každý s každým, ale neurony uvnitř jedné vrstvy mezi sebou nemají žádné spoje. Výstup jednoho neuronu je vstupem pro všechny neurony v následující vrstvě.

Příklad neuronové sítě se třemi vstupními neurony, čtyřmi neurony ve dvou skrytých vrstvách a jedním výstupním neuronem uvádíme na obrázku 7. Dodejme, že zobrazení architektury neuronové sítě do grafu slouží pouze pro názornost, v žádném případě se tato grafová struktura nereprezentuje v paměti počítače. Samozřejmě by to bylo teoreticky možné, ale výpočet by pak byl velmi neefektivní. V praxi se neuronové sítě obvykle reprezentují pomocí matic. Hlavní výhodou této reprezentace je možnost vektorizace výpočtu neuronové sítě. Tedy celý výpočet neuronové sítě je možné realizovat pomocí maticových operací, které je možné velmi efektivně implementovat. Maticová reprezentace neuronových sítí je hlavní důvod, proč se v dnešní době pro učení neuronových sítí používají grafické karty. Vektorizovaný výpočet je na grafické kartě výrazně rychlejší než na procesoru (např. 50×).

Nyní popíšeme, jak by vypadala reprezentace a výpočet třívrstvé neuronové sítě (vstupní vrstva se nepočítá) z obrázku 7. Vstup je tvořen vektorem x o rozměru $[3 \times 1]$. Váhy odpovídající spojům ze vstupu do první skryté vrstvy jsou uloženy do matice W_1 o rozměru $[4 \times 3]$ a bias parametry jsou uloženy do vektoru b_1 o rozměru $[4 \times 1]$. Podobně matice W_2 o rozměru $[4 \times 4]$ obsahuje váhy spojů pro druhou skrytou vrstvu a matice W_3 o rozměru $[1 \times 4]$ obsahuje váhy spojů pro výstup. Pro druhou skrytou vrstvu máme bias parametry b_2 o rozměru $[4 \times 1]$ a pro výstup b_3 o rozměru $[1 \times 1]$.

Dopředný průchod neuronové sítě s výše popsanou architekturou a aktivační funkcí sigmoid σ by vypadal následovně:

$$S(x) = \sigma(W_3\, \sigma(W_2\, \sigma(W_1\, x + b_1) + b_2) + b_3).$$

Pro lepší názornost přikládáme krátký program 1 napsaný v programovacím jazyce Python, který dopředný průchod třívrstvé neuronové sítě realizuje. Pro efektivní maticové násobení byla použita funkce `np.dot` implementována v knihovně Numpy.

```
1 import numpy as np
2
3 f = lambda x: 1.0/(1.0 + np.exp(-x))
4 x = np.random.randn(3, 1)
5 h1 = f(np.dot(W1, x) + b1)
6 h2 = f(np.dot(W2, h1) + b2)
7 out = np.dot(W3, h2) + b3
```

Zdrojový kód 1: Dopředný průchod třívrstvé neuronové sítě, implementován pomocí jazyka Python a knihovny Numpy. Zdroj: [9]

Ve výše popsaném příkladu jsme uvažovali vstupní vektor o rozměru $[3 \times 1]$, který reprezentuje jeden samostatný příklad (v angličtině input example). Například pro dataset MNIST bychom tedy měli vstup o rozměru $[784 \times 1]$ reprezentující jeden obrázek. Otázkou je, jakým způsobem bychom mohli realizovat výpočet pro více příkladů naráz. Naivní přístup by byl procházet data postupně například ve for-cyklu, aplikovat výpočet pro jednotlivé příklady a výsledky následně ukládat do seznamu. Toto řešení by jistě vedlo ke správnému výsledku, ale je velmi neefektivní. Díky maticové reprezentaci neuronové sítě můžeme jako vstup použít matici x o rozměru $[3 \times n]$ kde n je počet příkladů, které budou vstupem do neuronové sítě. Jako výstup poté dostaneme vektor o rozměru $[1 \times n]$.

Celá neuronová síť z výše uvedeného příkladu je reprezentována parametry uloženými v $W_1, W_2, W_3, b_1, b_2, b_3$, které určují jakým způsobem se výpočet pro určitý vstup bude chovat. Naším cílem je najít takové číselné parametry, aby síť pro danou úlohu dělala to, co chceme. Tedy například aby správně klasifikovala obrázky, zda se jedná o snímek kočky nebo psa. Hlavní otázkou tedy je, jakým způsobem můžeme vhodné parametry najít. Proces hledání vhodných parametrů se nazývá učení neuronové sítě (nebo také trénování neuronové sítě), který popíšeme v následující kapitole.

3.4 Učení neuronových sítí

V této kapitole se zaměříme na problém učení neuronových sítí, tedy hledání vhodných parametrů sítě, kterými je reprezentována. Na začátek zdůrazníme, že chování neuronové sítě dané architektury je určeno pouze hodnotami jejich parametrů. Neuronová síť tedy neobsahuje žádná předdefinovaná pravidla typu if-then, podle kterých by se rozhodovala.

Chování neuronové sítě je indukováno z dat, na kterých ji učíme. Abychom tedy mohli neuronovou síť použít k řešení nějakého problému, je nutné mít k dispozici trénovací dataset. V dalším popisu se omezíme na úlohu klasifikace.

Pro názornost budeme mluvit o jednoduchém školském příkladu rozpoznávání obrazu, kde vstupem do neuronové sítě je obrázek zvířete a úkolem neuronové sítě je určit, zda je na obrázku pes nebo kočka. Trénovací dataset bude pro tuto úlohu složen z jednotlivých snímků psů a koček a ke každému snímků budeme mít k dispozici značku, určující co obrázek obsahuje. Značka nám tedy říká, zda je na obrázku kočka či pes.

Učení neuronové sítě probíhá následujícím způsobem. Postupně neuronové síti ukazujeme jednotlivé obrázky z trénovacího datasetu a pro každý obrázek zjistíme, co si o něm neuronová síť myslí. Tedy například neuronové síti ukážeme obrázek kočky a následně zjistíme, že neuronová síť si na 80% myslí, že jde o psa a na 20%, že jde o kočku. Takový výsledek evidentně není správný a proto parametry neuronové sítě upravíme takovým způsobem, aby neuronová síť příště odpověděla lépe. Tímto způsobem neuronové síti postupně ukazujeme další a další obrázky z trénovacího datasetu a to až do momentu, kdy neuronová síť správně klasifikuje obrázky, které dostane na vstupu.

Popis učení neuronové sítě, který jsme právě uvedli je velmi zjednodušený a slouží pouze pro hrubou představu o tom, jak učení sítě probíhá. Podívejme se nyní na učení neuronových sítí detailněji a přesněji. Tím, že se neuronová síť podívá na obrázek myslíme aplikaci neuronové sítě na daný obrázek a to co si o obrázku neuronová síť myslí je výsledek této aplikace. Dále jsme uvedli, že zjistíme, že výsledek aplikace neuronové sítě na vstupní obrázek není správný. Otázkou je, jakým způsobem to můžeme zjistit. K tomuto účelu se používá chybová funkce (anglický termín je cost nebo loss), která změří jak dobrou predikci nám neuronová síť poskytla. Chybových funkcí, které můžeme pro učení neuronových sítí použít existuje mnoho. Jako příklad uvedeme funkci MSE (Mean squared error), která je definovaná jako

$$J_{MSE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h_\theta(x_i))^2,$$

kde n je velikost trénovacího datasetu, $h_\theta(x_i)$ označuje výsledek aplikace neuronové sítě na obrázek x_i a y_i je správné řešení, které chceme aby neuronová síť určila (značka, kterou máme k dispozici ke každému obrázku z trénovacího datasetu). Symbol θ značí parametry sítě (v předchozím textu značeno symbolem w). MSE tedy pro každý obrázek z trénovacího datasetu spočítá odchylku neuronové sítě od správného řešení a ze všech druhých mocnin odchylek spočítá průměr. Hodnota MSE kterou pro daný dataset a neuronovou síť dostaneme nám říká, jak je neuronová síť dobrá. Platí, že čím menší hodnotu MSE dostaneme, tím je síť lepší.

Nakonec zbývá dořešit, jakým způsobem budeme upravovat parametry neuronové sítě, abychom chybu sítě co nejvíce redukovali. K tomuto účelů se používá algoritmus Gradient Descent.

3.4.1 Gradient Descent algoritmus

Cílem Gradient Descent algoritmu je najít takové parametry θ, aby hodnota chybové funkce $J(\theta)$ byla co nejmenší. V následujícím popisu algoritmu budeme pro jednoduchost uvažovat pouze dva číselné parametry θ_0 a θ_1. Nejdříve je potřeba inicializovat parametry θ_0 a θ_1, např. $\theta_0 = 0.7$, $\theta_1 = 0.5$. Dále budeme postupně upravovat parametry θ_0 a θ_1 tak, abychom našli minimum chybové funkce $J(\theta_0, \theta_1)$. Způsob úpravy parametrů je popsán v algoritmu 1.

Algorithm 1 Algoritmus Gradient Descent

1: **repeat**
2: $\quad \theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
3: $\quad$ (for $j = 0$ and $j = 1$)
4: **until** convergence

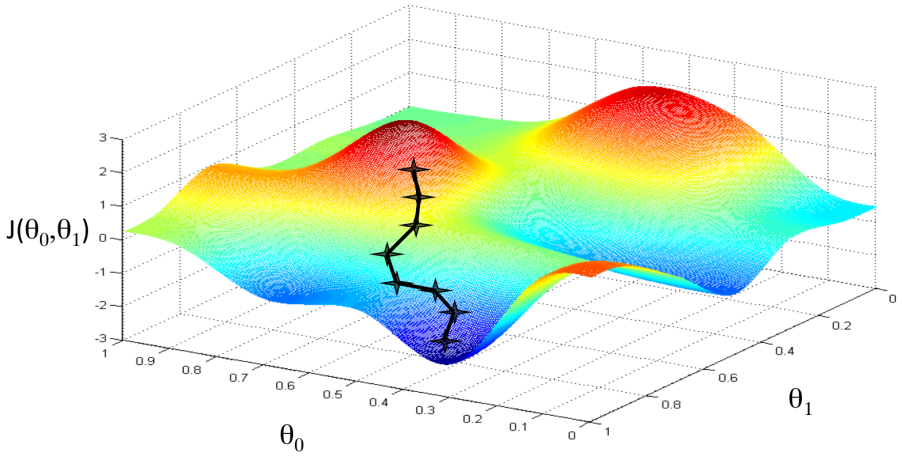
Proces hledání parametrů θ_0 a θ_1 které minimalizuji chybovou funkci, je možné znázornit do 3D grafu. Příklad povrchu chybové funkce s vyznačenými kroky hledání minima funkce uvádíme na obrázku 8. Na horizontálních osách jsou vyneseny hodnoty parametrů θ_0 a θ_1 a na vertikální ose hodnoty chybové funkce $J(\theta_0, \theta_1)$. Při inicializaci parametrů začneme na náhodném místě povrchu a následně postupnými kroky hledáme minimum chybové funkce. Každý krok po povrchu chybové funkce je určen směrem největšího spádu funkce a velikostí tohoto kroku. Směr určíme pomocí gradientu chybové funkce $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ a velikost kroku je určena hyperparametrem α, který se nazývá rychlost učení (z anglického termínu learning rate) a který si musíme vhodně zvolit.

Pokud jako chybovou funkci zvolíme MSE a provedeme parciální derivace podle parametrů θ_0 a θ_1, můžeme algoritmus 1 rozepsat tak, jak je znázorněno v algoritmu 2. V takovém tvaru lze algoritmus Gradient Descent přímo implementovat.

Algorithm 2 Algoritmus Gradient Descent s derivovanou chybovou funkcí MSE

1: **repeat**
2: $\quad \theta_0 \leftarrow \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$
3: $\quad \theta_1 \leftarrow \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$
4: **until** convergence

V předcházejícím textu jsme popisovali hledání vhodných parametrů na příkladě, kdy máme pouze dva parametry θ_0 a θ_1. Zobecnění algoritmu Gradient Descent pro libovolný počet parametrů je velmi snadné. Obecná forma algoritmu Gradient Descent je uvedena v algoritmu 3.



Obrázek 8: Grafické znázornění chybové funkce $J(\theta_0, \theta_1)$. Zdroj: [10]

Algorithm 3 Algoritmus Gradient Descent v obecném tvaru

1: **repeat**
2: $\quad \theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$
3: $\quad$ (for every $j = 0, \ldots, n$)
4: **until** convergence

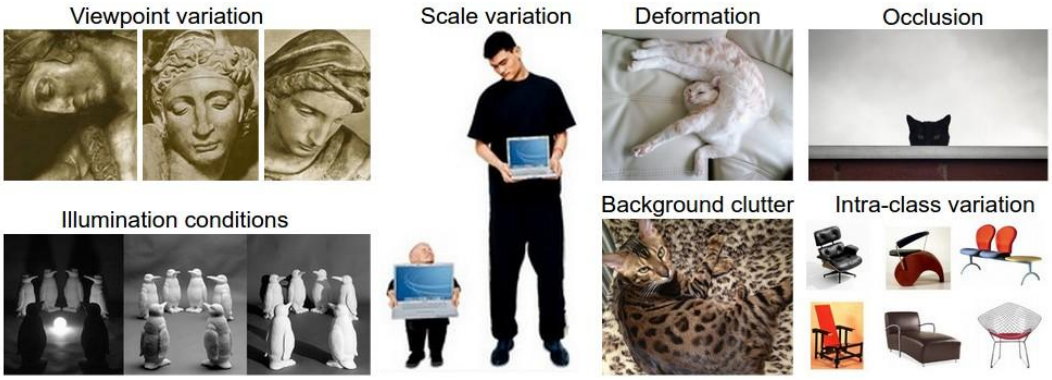
Hlavním krokem při použití Gradient Descent algoritmu je spočítání gradientu chybové funkce. Pro jednoduché modely, jako je například lineární regrese může derivace vypadat tak, jak jsme uvedli v algoritmu 2. Komplikovanější situace nastává v případě použití neuronových sítí. Pro výpočet gradientu a následné hledání vhodných parametrů neuronové sítě byl vyvinut algoritmus Backpropagation, jehož popis je poměrně technický a proto zájemce odkazujeme na [11].

4 Zpracování obrazu pomocí neuronových sítí

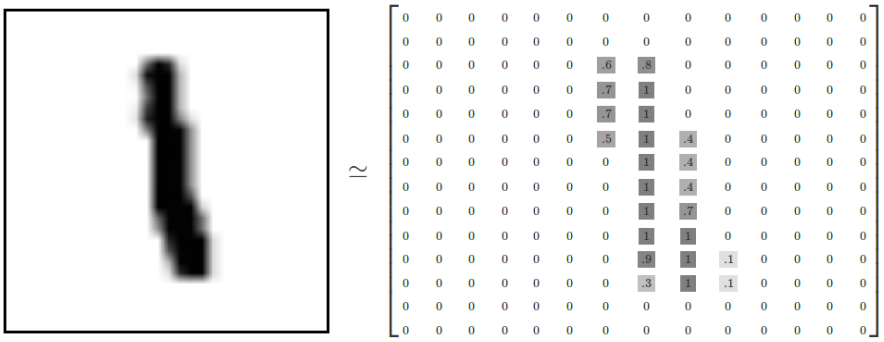
V předcházejících kapitolách jsme zmínili dvě základní úlohy z oblasti zpracování obrazu, kterými jsou klasifikace a segmentace. Pro řešení obou úloh je možné využít umělé neuronové sítě, které dosahují v porovnání s ostatními metodami vysoké přesnosti.

Úlohy zpracování obrazu patří k velmi náročným informatickým úlohám. Vraťme se k ukázkové úloze klasifikace psů a koček z předchozí kapitoly. Klasifikátor, který bychom chtěli pro řešení této úlohy vytvořit, musí být invariantní například k rotacím, posunutí, rozmazání, překrytí a dalším variacím obrazu. Příklady variací vstupů klasifikátoru jsou uvedeny na obrázku 9. Tato velká variabilita v možnostech vstupů do klasifikátoru je důvodem obtížnosti úlohy.

Prvním předpokladem, který musí být splněn abychom o využití neuronových sítí pro danou úlohu mohli uvažovat, je možnost reprezentovat vstup neuronové



Obrázek 9: Příklady variací obrazu. Zdroj: [13]



Obrázek 10: Reprezentace obrazu pomocí matice čísel. Zdroj: [12]

sítě pomocí čísel. Toto omezení na vstup vyplývá ze způsobu, jakým neuronová síť počítá. Jelikož obraz je v počítači reprezentován pomoci číselných matic, jak je znázorněno na obrázku 10, tento předpoklad je splněn a neuronové sítě ke zpracování obrazu můžeme snadno použít.

Nyní je otázka, v jakém tvaru můžeme obraz neuronové síti předat. Nejdříve se zaměříme na fully-connected neuronové sítě, které jsme popisovali v předchozích kapitolách. Vstupem pro tento typ neuronových sítí je vektor čísel, ale obraz je reprezentován pomocí matice a tedy abychom mohli obraz předat neuronové síti, musíme jej nejdříve linearizovat do vektoru. Linearizaci matice můžeme realizovat například po řádcích tak, jak je uvedeno v následující příkladě.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

Protože architektura sítě je fully-connected, tedy z každého vstupu vedou spoje do každého neuronu první vrstvy neuronové sítě, linearizaci můžeme použít bez ztráty informace o prostorovém uspořádání pixelů obrazu. Je zde ale jeden

problém. I pro relativně malé obrázky, dostáváme po linearizaci velké vektory. Například pokud bychom měli dataset obsahující barevné obrázky o velikosti 400×400 px, vstupní vektor neuronové sítě bude mít $(400^2) \cdot 3 = 480000$ px. Pro neuronovou síť obsahující v první vrstvě pouze jeden neuron budeme v tomto případě potřebovat 480000 vah. Pravděpodobně ale budeme chtít použít více než jeden neuron a tedy počet vah rychle poroste. Takto velké množství vah může vést jednak k problémům s nedostatkem operační paměti a hlavně k přetrénování (overfitting) neuronové sítě [2]. Problémy s velkým počtem parametrů řeší speciální typ neuronových sítí. Jedná se konvoluční neuronové sítě, které popíšeme v následující kapitole.

4.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě [2] jsou velmi podobné klasickým neuronovým sítím, které byly popsány v předchozím textu. Hlavním rozdílem je předpoklad, že vstupem konvolučních neuronových sítí je obraz. Tento předpoklad nám umožní efektivněji realizovat dopředný průchod obrázku neuronovou sítí (výpočet) a také redukovat množství parametrů neuronové sítě. Před tím, než se pustíme do technického popisu architektury konvolučních neuronových sítí, uvedeme krátkou motivaci.

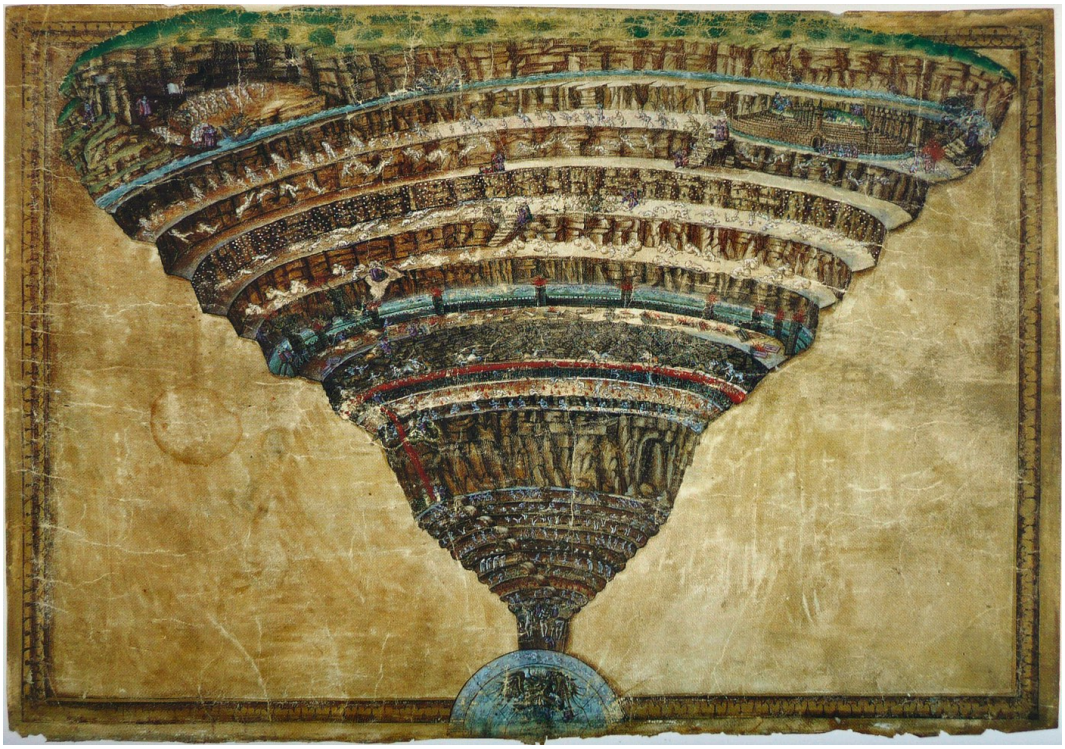
Představme si, že stojíme například před obrazem The Abyss of Hell 11 a zamysleme se, jakým způsobem si tento komplikovaný obraz budeme prohlížet. Uvažujme situaci, kdy nás zajímá, jak vypadají jednotlivé kruhy pekla, tedy budeme chtít vědět, co obraz obsahuje. Pravděpodobně se postupně zaměříme na jednotlivé části obrazu a postupně budeme zkoumat jednotlivé kruhy pekla. Tedy postupně budeme posouvat oči po obrazu a budeme se snažit pochopit co obraz obsahuje. Tento způsob analýzy obrazu přesně kopírují konvoluční neuronové sítě. Použití klasických fully-connected neuronových sítí by odpovídalo jednomu pohledu na obraz jako celek. Konvoluční neuronové sítě tedy kopírují způsob jakým analyzuje obraz člověk.

Na rozdíl od klasických neuronových sítí, které mají váhy ve vrstvách uspořádané do 2D matic, konvoluční neuronové sítě mají váhy ve vrstvách uspořádané ve třech dimenzích: šířka, výška a hloubka. Takové uspořádání vah v jednotlivých vrstvách koresponduje s reprezentací obrázků. Například RGB obrázek je reprezentován trojrozměrnou maticí $width \times height \times 3$.

Konvoluční neuronová síť je sekvence 3D vrstev, kde každá vrstva transformuje výsledek aplikace předchozí vrstvy přes aktivační funkci (např. sigmoid). Konvoluční neuronové sítě mohou být složeny ze tří typů vrstev: konvoluční vrstva, pooling vrstva a fully-connected vrstva. Vhodným poskládáním těchto vrstev vytvoříme výslednou architekturu konvoluční neuronové sítě.

4.1.1 Konvoluční vrstva (CONV)

CONV vrstva je základní vrstvou konvolučních neuronových sítí a realizuje většinu výpočtu sítě. Parametry jsou uspořádané do 3D filtrů jejichž výška a šířka

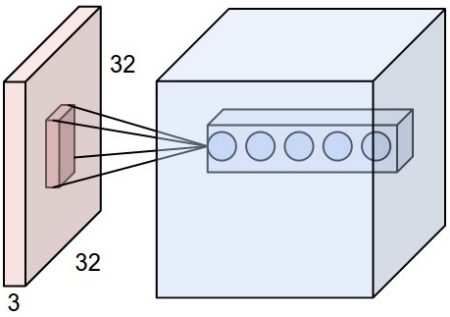


Obrázek 11: Obraz The Abyss of Hell od malíře Sandro Botticelliho. Zdroj: [14]

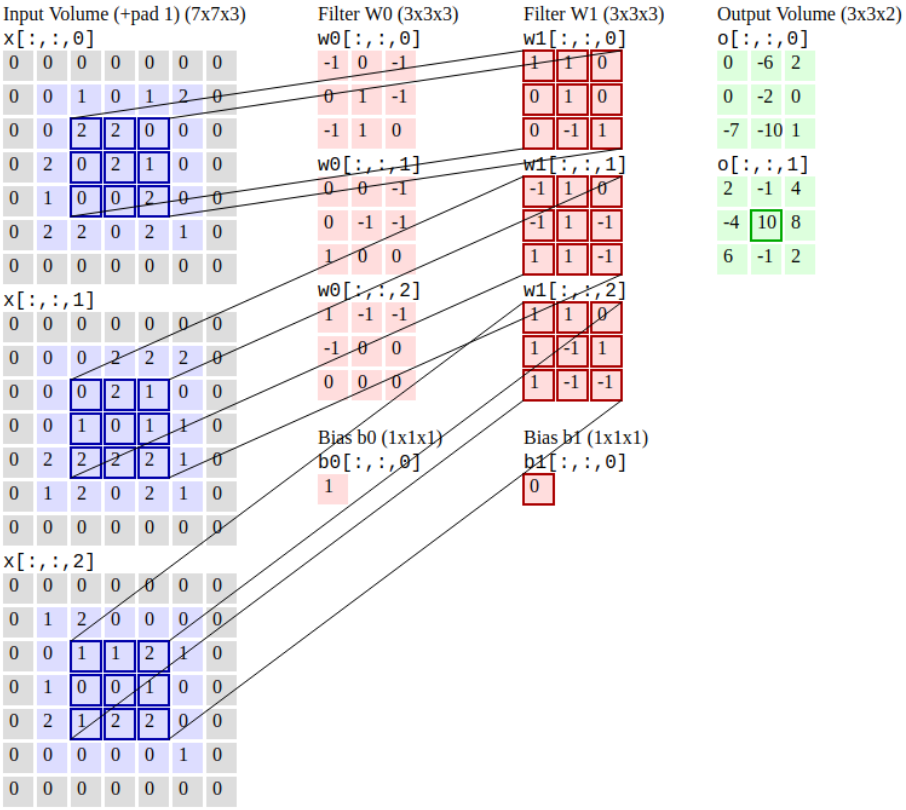
jsou vzhledem k velikosti vstupu malé a vhodně je volíme (jeden z hyperparametrů). Hloubka filtru je vždy stejná jako hloubka vstupu do této vrstvy. Například pokud budeme mít na vstupu RGB obrázek o rozměru $(32 \times 32 \times 3)$, tak filtr může mít rozměr například $5 \times 5 \times 3$, kde rozměr strany filtru 5 jsme si zvolili a hloubka 3 je dána hloubkou vstupu (pro první vrstvu počtem barevných kanálů obrázku). Schéma konvoluční vrstvy uvádíme na obrázku 12.

Při dopředném průchodu obrázku sítí posouváme (konvolujeme) filtr postupně po celém vstupu (pro první vrstvu je vstupem obrázek) a v každé pozici spočítáme skalární součin mezi filtrem a vstupem. Postupným posouváním filtru o zvolený krok (stride) a počítáním skalárních součinů v každé pozici vstupu dostaneme 2D matici, na kterou aplikujeme aktivační funkci (pro každý prvek matice). Každá CONV vrstva je složena z více filtrů (například 12) a každý filtr produkuje 2D matici aktivací. Poskládáním těchto jednotlivých 2D aktivací za sebe získáme 3D výstup CONV vrstvy. Například pokud máme 12 filtrů a každý produkuje 2D matici aktivací o rozměru 50×50, výstupem je 3D matice o rozměru $50 \times 50 \times 12$.

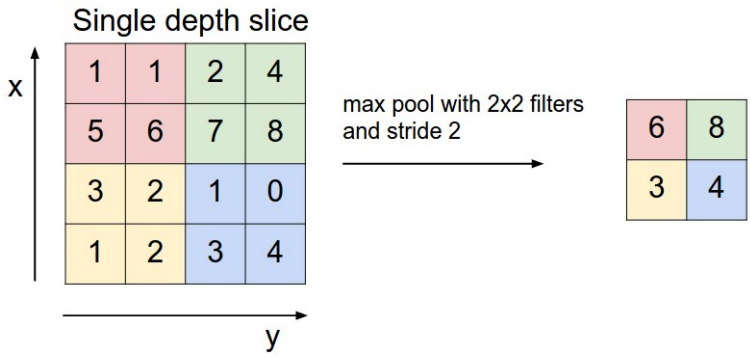
Na obrázku 13 je zachycen výpočet CONV vrstvy složené ze dvou filtrů o rozměru $3 \times 3 \times 3$ pro vstup o rozměru $7 \times 7 \times 3$. Animaci zachyceného výpočtu je možné najít na [2].



Obrázek 12: Grafické znázornění CONV vrstvy, která má na vstupu obrázek o rozměru $32 \times 32 \times 3$ a je složena z pěti filtrů. Zdroj: [2]



Obrázek 13: Znázornění výpočtu konvoluce. Zdroj: [2]



Obrázek 14: Znázornění poolingu. Zdroj: [2]

4.1.2 Pooling vrstva (POOL)

POOL vrstva se typicky vkládá za CONV vrstvu. Jejím hlavním účelem je redukovat rozměr vstupu (výšku a šířku) a tím omezit overfitting. POOL vrstva konvoluje stejně jako CONV vrstva plovoucí okno po vstupu, přičemž operuje nezávisle v každé hloubce vstupu. Na každou část vstupu, kde se aktuálně plovoucí okno nachází, aplikuje *max* operaci. Výsledkem aplikace *max* operace je tedy maximální číslo, které se nachází na aktuální pozici plovoucího okna. Na rozdíl od CONV vrstvy se plovoucí okno většinou posouvá o takový počet pixelů, aby nedošlo k překrývání. Místo operace *max* je možné použít i jiné funkce jako je například funkce *average*, která jako výstup vrací průměr ze všech pixelů v dané hloubce na pozici plovoucího okna. Na obrázku 14 je znázorněn max-pooling s plovoucím oknem o velikosti 2×2 a krokem 2, který aplikujeme na vstup o rozměru $4 \times 4 \times 1$. Na závěr zdůrazněme, že POOL vrstva pouze zmenšuje rozlišení vstupu a neobsahuje žádné váhy.

4.1.3 Fully-connected vrstva (FC)

Fully-connected vrstva se v konvolučních sítích typicky používá jako poslední (např. pro klasifikační úlohu) a funguje úplně stejně jako v klasických neuronových sítích. Z každého vstupu vedou spoje do všech neuronů této vrstvy. Pokud bychom například měli klasifikovat do dvou tříd, povedou všechny spoje ze vstupu do jednoho neuronu.

4.1.4 Architektura pro klasifikaci a segmentaci

Jedním ze zásadních problémů při návrhu architektury neuronové sítě (ať už fully-connected nebo konvoluční) je skutečnost, že pro aktuálně řešenou úlohu dopředu nevíme, s jakou architekturou dosáhneme nejvyšší přesnosti. V současné době vychází velké množství vědeckých článků, které popisují, jakých výsledků bylo dosaženo s novými typy architektur neuronových sítí, nicméně vždy je nová architektura vyzkoušena na jedné úloze a tedy nemáme žádnou záruku, že pro

naši, i když podobnou úlohu bude tato architektura fungovat stejně dobře. Existují také články (např. [15]), ve kterých se autoři snaží vymyslet obecný návod pro výběr vhodné architektury. Tyto přístupy ale bývají většinou časově náročné a často se spíše vyplatí vyzkoušet více různých architektur a vybrat tu nejvýkonnější. I když dopředu nevíme, zda se zvolenou architekturou dosáhneme uspokojivých výsledků, přesto s nějakou architekturou začít musíme. Uvedeme nyní příklady často používaných architektur pro úlohy klasifikace a segmentace. Nejčastěji používaná architektura konvolučních sítí pro klasifikaci má následující strukturu

$$INPUT \to [[CONV \to ReLU]{*}N \to POOL?]{*}M \to [FC \to ReLU]{*}K \to FC,$$

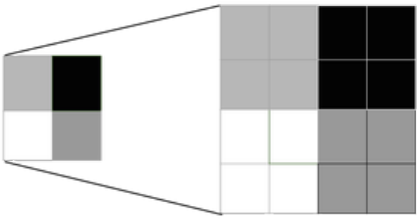
kde $*$ označuje opakování části sítě a $POOL?$ označuje volitelnou pooling vrstvu. Počty opakování jsou obvykle $N \geq 0, M \geq 0, K \geq 0$. Příklady často používaných architektur konvolučních sítí, vytvořené dle uvedeného vzoru vypadají následovně [2]

- $INPUT \to FC$ reprezentuje lineární klasifikátor, kde $N = M = K = 0$.
- $INPUT \to CONV \to ReLU \to FC$, kde $N = M = 1, K = 0$.
- $INPUT \to [CONV \to ReLU \to POOL] * 2 \to FC \to ReLU \to FC$, kde $N = 1, M = 2, k = 1$.
- $INPUT \to [CONV \to ReLU \to CONV \to ReLU \to POOL] * 3 \to [FC \to ReLU] * 2 \to FC$, kde $N = 2, M = 3, K = 2$.

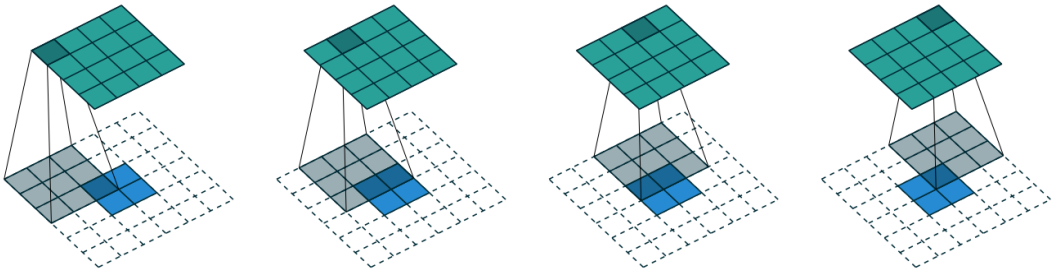
Některé architektury neuronových sítí mají své jméno, jedná se většinou o architektury, pomocí kterých se podařilo kvalitně vyřešit veřejně známý dataset. Jednou z nejznámějších architektur konvolučních neuronových sítí je AlexNet [16] se kterou Alex Krizhevsky v roce 2012 vyhrál ImageNet ILSVRC challenge [17]. Další úspěšné architektury jsou například GoogLeNet [18], VGGNet [19] nebo ResNet [20].

Hlavním rozdílem neuronových sítí, které jsou určeny pro segmentaci oproti sítím, které jsou určeny pro klasifikaci, je poslední vrstva sítě. Zatímco u klasifikačních sítí je poslední vrstva typicky fully-connected a jednotlivé neurony poslední vrstvy reprezentují třídy klasifikace, poslední vrstvou u segmentačních sítí je většinou konvoluční vrstva a neurony této vrstvy reprezentují třídu pro každý pixel obrázku. Pro každý pixel obrázku tedy dostaneme informaci o tom, zda pixel patří segmentovanému objektu či nikoli. Jinými slovy, poslední vrstva reprezentuje černobílou masku segmentovaného obrazu (bíle pixely vyznačují segmentovaný objekt a černé pozadí). V principu lze říci, že jediný požadavek na architekturu segmentační sítě je možnost získat z výstupu sítě masku segmentovaného obrazu.

Často používanou strategií pro učení segmentačních sítí je nejdříve snížit rozlišení vstupního obrazu pomocí CONV a POOL vrstev. Motivace pro toto snižování rozlišení je snaha ignorovat nevýznamné části vstupního obrazu a učit pouze



Obrázek 15: Znázornění unpoolingu. Zdroj: [21]



Obrázek 16: Znázornění dekonvoluce. Zdroj: [22]

významné rysy obrazu, jako jsou tvary nebo hrany segmentovaných objektů. Po dosažení dostatečně malého rozlišení vstupního obrazu následuje opačná fáze, ve které pomocí unpoolingu nebo dekonvoluce rekonstruujeme obraz o původní velikosti reprezentující masku. Tento typ segmentačních neuronových sítí se nazývá encoder-decoder.

Unpooling funguje velmi jednoduše. Postupně procházíme obraz po jednom pixelu a z každého vytvoříme matici (o velikosti plovoucího okna poolingu) obsahující hodnoty pouze tohoto pixelu. Pokud bychom tedy měli vstup o rozměru 2×2 a z každého pixelu bychom vytvořili matici 2×2, výsledkem unpoolingu by byla matice 4×4 tak jak je znázorněno na obrázku 15.

Alternativou k použití unpoolingu je dekonvoluce. Princip dekonvoluce popíšeme na konkrétním příkladu. Detailní vysvětlení a obecný princip dekonvoluce lze najít v [22]. Uvažujme příklad, ve kterém konvolujeme filtr 3×3 po vstupu 4×4 s krokem 1. Výstup této konvoluce má rozměr 2×2. Dekonvoluce (nebo také transponovaná konvoluce) bude pro tento případ vypadat následovně. Filtrem 3×3 budeme konvolovat po vstupu 2×2, ke kterému je přidán okraj z nulových pixelů o velikosti 2 (zero-padding je 2). Princip dekonvoluce pro tento případ je znázorněn na obrázku 16. Při konvoluci filtrem 3×3 po vstupu 4×4 jsme získali výstup 2×2 a při dekonvoluci filtrem 3×3 po vstupu 2×2 opadovaného okrajem o tloušťce 2 jsme získali zpět výstup 4×4.

5 Experiment

V mnoha oblastech medicínského výzkumu se setkáváme se situací, kdy metoda, kterou výzkumní pracovníci používají, je založena na získání a následném zpracování velkého objemu dat. Příkladem je výzkum v oblasti genotoxicity, kterou se zabývá skupina z Ústavu molekulární a translační medicíny (UMTM). Výzkumní pracovníci v určitém časovém intervalu snímají jádra buněk v Petriho misce a nasbíraná data následně předají softwaru který je dodávaný k použitému mikroskopu. Software má za úkol najít všechna jádra buněk, která jsou na jednotlivých snímcích a následně vyhodnotit jejich charakteristiky, které jsou pro vědecké pracovníky podstatné. Nejnáročnější část zpracování snímků je nalezení jednotlivých jader buněk v obraze, tedy jejich segmentace. V rámci jednoho snímku mají buňky často různé odstíny, což například při použití metod prahování může způsobovat problémy. S použitím adaptivního prahování lze dosáhnout poměrně uspokojivých výsledků, nicméně i tato metoda má své limity. Buněčná jádra s různými odstíny jsou v obrázku 17 vyznačeny zelenou barvou. Výrazně větším problémem je to, že buňky se shlukují do klastrů tak, jak je vyznačeno v obrázku 17 modrou barvou. Abychom jádra buněk mohli dále korektně analyzovat, je potřeba jednotlivá jádra buněk v klastrech od sebe oddělit, což je pro software dodávaný k mikroskopu zásadní problém. Cílem experimentu i diplomové práce bylo navrhnout metodu, která tento problém vyřeší.

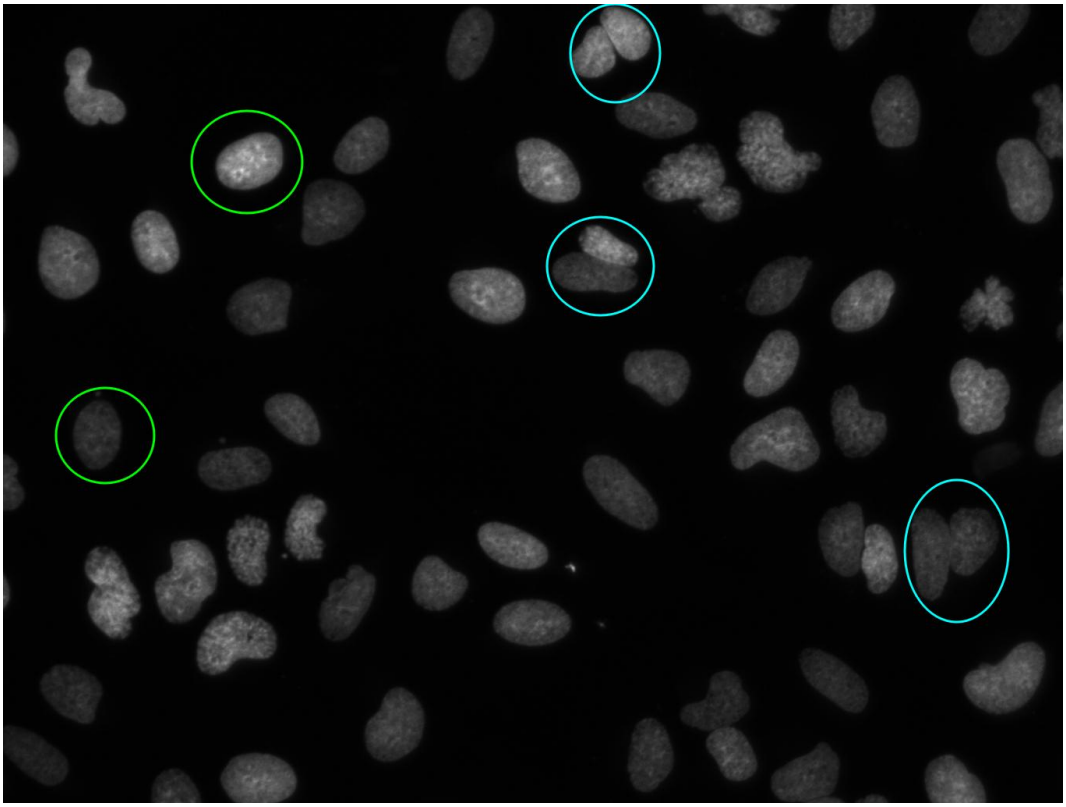
5.1 Zadání úlohy

Jak již bylo nastíněno v úvodu této kapitoly, vstupem pro řešenou úlohu jsou mikroskopické snímky jader buněk a cílem je na snímku segmentovat všechna jádra buněk. Datová sada obsahuje 780 mikroskopických snímků a byla získána od UMTM.

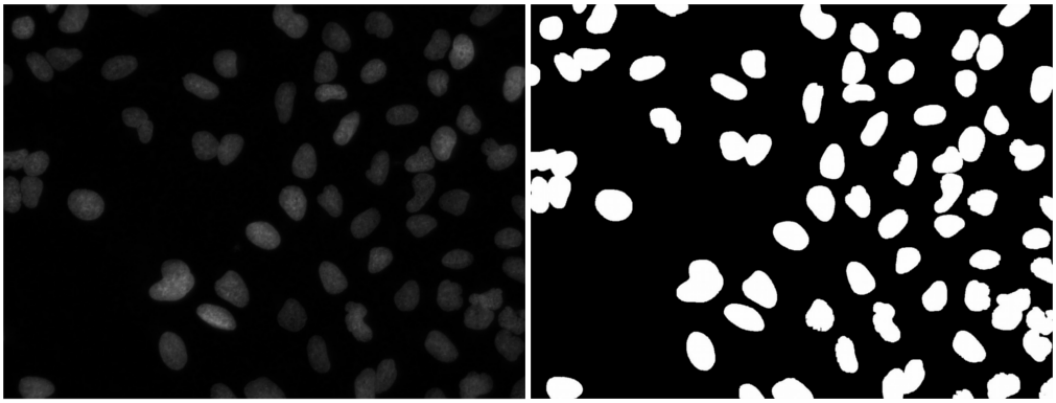
Řešení problému bylo rozděleno na dvě části. Cílem první části bylo ze snímků získat segmentaci všech jader buněk, tedy masku obsahující jak jednotlivé buňky tak i klastry. V druhé části jsme od sebe oddělovali jednotlivá jádra buněk, která tvořila klastry.

5.2 Segmentace jader buněk

Aby bylo možné použít umělou neuronovou síť pro segmentaci jader buněk, potřebovali bychom trénovací data. Dataset by musel být složen z dvojic: snímek jader buněk (vstup neuronové sítě) a maska (správné řešení). Příklad dvojice snímek-maska je na obrázku 18. Pro každý snímek datasetu bychom tedy potřebovali ručně vytvořit masku (například v programu GIMP). Přibližná doba tvorby masky pro jeden snímek je 15 minut, tedy pro celý dataset obsahující 780 snímků potřebujeme minimálně 195 hodin. Takový čas nebylo možné tvorbě datasetu obětovat a proto jsme pro tento experiment zvolili metodu prahování, konkrétně funkci `cv2.threshold()`.



Obrázek 17: Mikroskopický snímek jader buněk, která jsou obarvená fluorescenčním barvivem DAPI [1]. Zelenou barvou jsou vyznačeny různé odstíny jader buněk a modrou barvou shluky jader buněk



Obrázek 18: Snímek jader buněk s maskou

Na první pohled se může zdát, že vytvoření datasetu je tak časově náročné, že není jasné zda se to vyplatí. Výkon neuronových sítí pro segmentaci je však výrazně vyšší než například zmíněného prahování. V praxi se proto do vytvoření kvalitního datasetu investují poměrně velké zdroje.

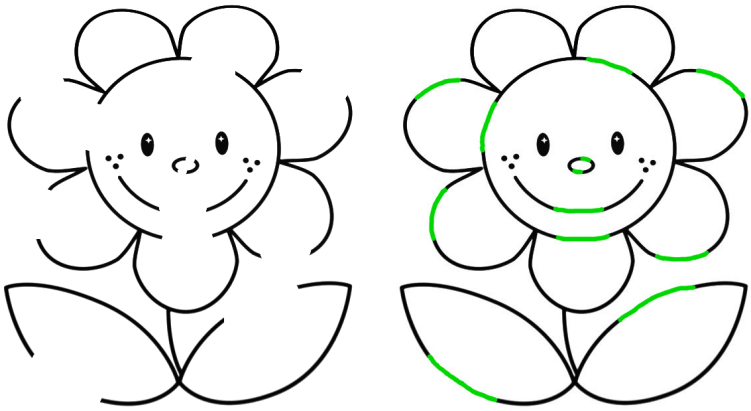
Hlavním navrhovaným vylepšením řešení této úlohy by bylo zvýšit výkon segmentace snímků. Tedy vytvořit kvalitní trénovací dataset pro segmentační neuronovou síť a získat tak přesnější masky, než jaké jsme schopni získat pomocí prahování.

5.3 Rozdělování klastrů

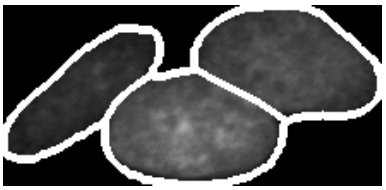
Myšlenka nové metody pro rozdělování klastrů jader buněk je velmi jednoduchá a byla inspirována úlohou, kterou řeší děti v předškolním věku[3]. Protože kreslení celých obrázků je pro malé děti většinou příliš náročné, často se jim předkládá jednoduchý obrázek s vynechanými částmi. Úlohou je dokreslit chybějící části obrázku tak, aby dával smysl. Úloha s řešením je znázorněna na obrázku 19.

Nejjednodušší způsob, jak rozdělit klastr jader buněk, je ohraničit každé jádro buňky v klastru okrajem, tak jak je znázorněno na obrázku 20. K vyřešení tohoto problému se nabízí využít konvoluční neuronovou síť, která jako vstup vezme výřez klastru a jako výstup vyprodukuje masku okraje. Vstup a výstup sítě je znázorněn na obrázku 21. Při využití tohoto přístupu se podařilo síť naučit velmi dobře generovat vnější okraj, ale měla velký problém se spoji jader buněk, které jsou pro rozdělení klastrů zásadní. Na základě mnoha experimentů které jsme provedli s tímto přístupem jsme usoudili, že síť se učila přechod barev mezi pozadím a jádrem buněk, který je znatelný na vnějším okraji klastru. Spoje mezi jádry často nejsou barevně odlišeny (buňky mají často stejný odstín), a proto síť měla problém tyto spoje vyznačit. Pro odstranění tohoto problému potřebujeme přimět neuronovou síť, aby se učila obecnější vlastnost než přechod barev.

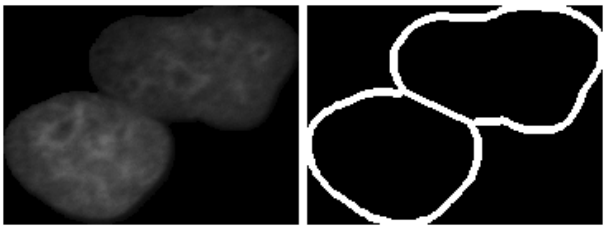
[3]Autorkou myšlenky nové metody je Helena Nováčková.



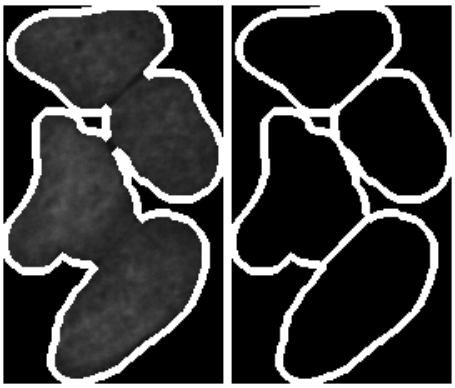
Obrázek 19: Dokreslovací obrázek kytky pro děti s řešením



Obrázek 20: Rozdělený klastr jader buněk



Obrázek 21: Vstup a výstup neuronové sítě pro neúspěšný pokus generování okraje



Obrázek 22: Vstup a výstup neuronové sítě pro úspěšný pokus generování okraje

Abychom zabránili neuronové síti učit se přechod barev, rozhodli jsme se spojit vnější okraj klastru se snímkem klastru a takto vytvořený vstup předat neuronové síti. Vnější okraj klastrů máme díky segmentaci z první fáze řešení této úlohy k dispozici. Můžeme například použít funkci `cv2.drawContours()`, která pro segmentovaný klastr vytvoří vnější okraj tohoto klastru. Požadovaným výstupem je stejně jako v prvním přístupu celkový okraj všech jader buněk. Vstup pro neuronovou síť a požadovaný výstup jsou znázorněny na obrázku 22.

Nyní se vrátíme k původní inspiraci této metody. Stejně jako se malé děti nejdříve učí propojovat nespojené čáry obrázku, snažili jsme se neuronovou síť naučit doplňovat okraj v místě, kde se buňky dotýkají.

5.4 Trénovací data

Dataset obsahuje 597 příkladů a byl rozdělen na trénovací (555 snímků) a testovací část (42 snímků). Datová sada byla vytvořena ve dvou krocích, nejdříve jsme pomocí masek klastrů získali vnější okraj, který jsme přidali ke snímku klastru. Následně jsme pomocí programu GIMP manuálně doplňovali spoje buněk do masek okrajů. Protože manuální vytváření masek bylo časově náročné, dataset je poměrně malý. Abychom tedy poskytli k učení co největší počet příkladů, vynechali jsme validační část. Cílem experimentu bylo provést proof of concept tohoto přístupu, tedy zda je neuronová síť vůbec schopna naučit se něco takového, jako je doplňování okrajů a proto se domníváme, že vynechání validační části je přípustné.

5.5 Augmentace

Abychom částečně vyřešili problém malého počtu dat, použili jsme při trénování neuronové sítě augmentace. Cílem augmentací je rozšířit dataset o nové snímky, které získáme pomocí transformací snímků datasetu. Jednou z augmentací, kterou jsme při trénování neuronové sítě použili je rotace. Jádra buněk se

mohou vyskytovat na snímku různě otočená, a proto bychom chtěli, aby naše síť byla vůči těmto otočením invariantní. Tedy aby síť uměla řešit daný klastr bez ohledu na to, jak je jádro buňky natočené. Další augmentace, které je možné použít jsou například rozmazání, přidání šumu, škálování a další. Při trénování sítě jsou augmentace většinou použity na jednotlivé příklady náhodně i s náhodnými parametry transformace. Například můžeme zvolit, že rotace příkladu (vstupu i výstupu) se provede s pravděpodobností 0.9 a úhel rotace se náhodně vybere z intervalu 0 až 360 stupňů s krokem dva stupně.

Augmentace jednak zvyšují přesnost sítě, ale také zabraňují jejímu přetrénování a hrají tedy významnou roli při řešení úloh pomocí umělé inteligence.

5.6 Architektura sítě a parametry učení

Při řešení této úlohy bylo vyzkoušeno mnoho architektur neuronových sítí. Nejlepšího výsledku dosáhla konvoluční neuronová síť s architekturou

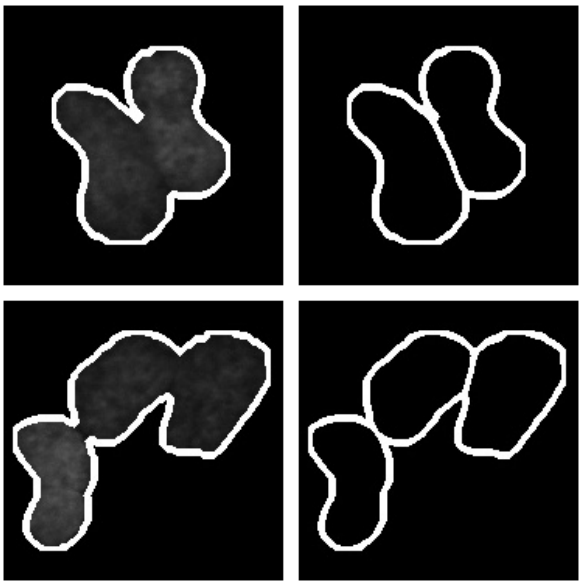
$$(CONV \to SIGMOID \to POOL) * 4 \to (DECONV \to SIGMOID) * 4.$$

Konvoluční neuronová síť tedy měla 4 konvoluční a 4 pooling vrstvy v části encoder a 4 dekonvoluční vrstvy v části decoder. Dále jsme použili aktivační funkci *sigmoid* a chybovou funkci MSE. Počet filtrů první konvoluční vrstvy byl 32 a s každou další vrstvou se zdvojnásobil, tedy další vrstvy měly 64, 128 a 256 filtrů. Pro plovoucí okno (receptive field) konvoluce byl zvolen rozměr 5×5. Rozměr pooling plovoucího okna byl zvolen 2×2. Počet kanálů výstupu dekonvoluční vrstvy byl v každé následující vrstvě poloviční, než ve vrstvě předcházející a výsledek poslední vrstvy měl kanál jeden (což odpovídá požadovanému rozměru masky). Optimalizační algoritmus byl zvolen Adam [23], což je vylepšený Gradient Descent algoritmus. Rychlost učení byla zvolena 0.0005.

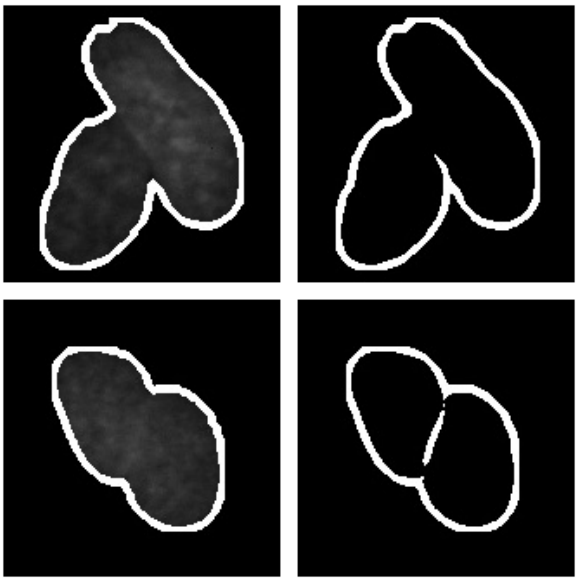
5.7 Výsledky

Na testovací části dat jsme dosáhli 99.98 % přesnost a hodnota chybové funkce byla 0.00145. Dodejme, že přesnost je počítána vzhledem k správně klasifikovaným pixelům. Uvedená přesnost tedy znamená, že průměrně 99.98 % pixelů v testovacím obrázku bylo klasifikováno správně. Z celkového počtu 42 testovacích obrázků bylo správně 39 a pouze u třech klastrů nebyl okraj úplně doplněn. Příklady klastrů, kterým neuronová síť doplnila okraj správně jsou znázorněny na obrázku 23 a klastry se kterými měla síť problémy jsou znázorněny na obrázku 24.

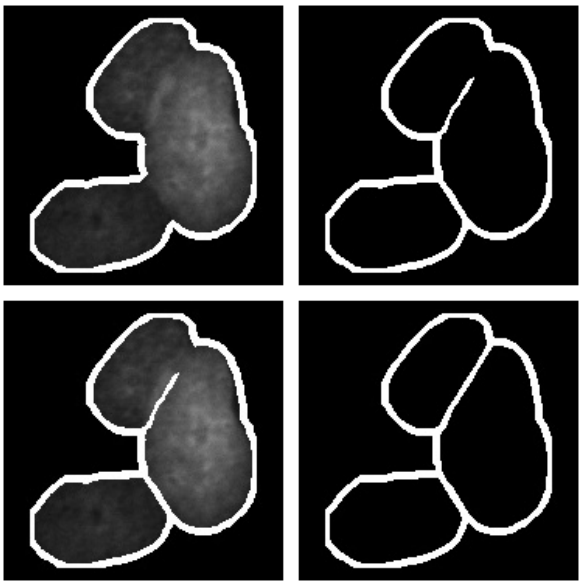
Na maskách, které se neuronové síti nepodařilo dobře propojit je vidět, že síť se pokusila okraj z části vytvořit, ale nepovedlo se jí ho dokončit. Otázkou je, zda by tento začátek okraje síti nepomohl při dalším pokusu okraj dokončit. Tento přístup jsme vyzkoušeli a ukázalo se, že funguje velmi dobře. Pokud tedy spojíme nedokončenou masku okraje s původním vstupním obrázkem a předáme



Obrázek 23: Příklady klastrů, kterým neuronová síť správně doplnila okraj



Obrázek 24: Příklady klastrů, se kterými měla neuronová síť problémy



Obrázek 25: Příklad použití předchozího výsledku sítě pro přesnější tvorbu masky okraje

takto vytvořený obrázek neuronové sítě znovu, ve spoustě případů se jí okraj již podaří dokončit úplně, jak je znázorněno na obrázku 25. Pokud by se síti nepodařilo okraj dokončit při druhém pokusu, můžeme stejný postup aplikovat dále, dokud okraj není úplně doplněn. Počet těchto iterací můžeme například odvodit z testovací části dat, na které vyzkoušíme, kolik je minimálně nutné provést iterací, aby většina problematických klastrů byla rozdělena.

Nakonec dodejme, že kvalita řešení je omezena velikostí trénovacího datasetu. Pokud bychom měli dataset například stokrát větší, obsahující více možných případů klastrů, pak by výsledné řešení bylo s velkou pravděpodobností ještě přesnější.

6 Implementace řešení

Při řešení úloh pomocí neuronových sítí je nutné provádět velké množství experimentů, abychom zjistili, jak síť učit, jakou zvolit architekturu, hyperparametry apod. Z toho důvodu je důležité mít k dispozici framework, který nám toto rychlé provádění experimentů umožní. Při tvorbě této práce se podařilo takový framework vytvořit, ale je nutné podotknout, že jeho možnosti jsou značně omezené, nicméně pro účel této práci byl dostačující.

6.1 Požadavky na framework

Framework pro řešení machine learning úloh musí poskytovat dvě základní funkce, trénink neuronové sítě a predikci neuronové sítě. Jak trénink tak predikci je nutné konfigurovat, tedy například zvolit hodnoty hyperparametrů, zvolit dataset, který chceme pro učení využít a podobně. Vhodně navržená konfigurace nám velmi usnadní řešení úlohy a i když se to na první pohled nezdá, je to velmi důležitá součást frameworku. Dále je užitečné mít možnost obecným způsobem definovat jednotlivé typy neuronových sítí tak, abychom si následně mohli pouze zvolit, kterou síť pro daný experiment použijeme. Stejná situace je s definicí datasetu. Framework by měl také poskytovat možnost použít augmentace, tedy vhodným způsobem je začlenit mezi načítání dat (o které se stará dataset) a učení sítě. Poslední důležitou funkcí je průběžné zobrazování a ukládání výsledků učení i predikce neuronové sítě.

6.2 Použité nástroje a vybavení

Pro implementaci frameworku i definici neuronových sítí byl použit programovací jazyk Python, který je v současné době spolu s jazykem C++ nejpoužívanějším programovacím jazykem k řešení machine learning úloh. Pro definici neuronových sítí byl použit framework TensorFlow, který vytvořila firma Google. TensorFlow slouží k definici machine learning modelů, tedy například umělých neuronových sítí. Uživatel (programátor) tedy pouze nadefinuje architekturu sítě, TensorFlow tento model zkompiluje do výpočetního grafu a spustí jej na CPU nebo GPU. Hlavní výhodou frameworků jako je TensorFlow (alternativy jsou například Theano, PyTorch, Caffe a další) je, že jsou velmi rychlé, automaticky spočítají derivaci funkce, kterou síť reprezentuje a použijí algoritmus Backpropagation pro učení sítě. Uživatel se tedy touto technicky náročnou záležitostí nemusí zabývat a může se plně soustředit na řešení úlohy. Pro představu jak může vypadat neuronová síť vytvořená s použitím TensorFlow, uvádíme implementaci neuronové sítě 2 dle obrázků 7. Konfigurace tréninku a predikce byla řešena pomocí knihovny argparse. Pro provádění experimentů a trénink výsledné neuronové sítě bylo využito MetaCentrum, kde bylo možné spustit trénink sítě na grafické kartě.

6.3 Struktura frameworku

Framework obsahuje dva základní python skripty, `train.py` a `predict.py`. Trénovací skript obsahuje implementaci tréninkového cyklu (training loop), ve kterém se neuronové síti postupně předávají trénovací a testovací data. Průběžné výsledky (např. přesnost sítě) jsou zapisovány na standardní výstup a zároveň ukládány do log souboru. Skript pro predikci pouze předá neuronové síti data k vyhodnocení a výsledky uloží. Oba skripty je možné konfigurovat separátním konfiguračním souborem, který na každém řádku obsahuje jeden argument ve tvaru `--key=value`. Uvádíme příklad konfiguračního souboru 3 pro trénink.

```
import tensorflow as tf

x = tf.placeholder(tf.float32, [None, 3])
W1 = tf.Variable(tf.truncated_normal([3, 4], stddev=0.1))
b1 = tf.Variable(tf.constant(0.1, shape=[4]))
W2 = tf.Variable(tf.truncated_normal([4, 4], stddev=0.1))
b2 = tf.Variable(tf.constant(0.1, shape=[4]))
W3 = tf.Variable(tf.truncated_normal([4, 1], stddev=0.1))
b3 = tf.Variable(tf.constant(0.1, shape=[1]))

h1 = tf.nn.sigmoid(tf.matmul(x, W1) + b1)
h2 = tf.nn.sigmoid(tf.matmul(h1, W2) + b2)
y = tf.nn.sigmoid(tf.matmul(h2, W3) + b3)
```

Zdrojový kód 2: Implementace neuronové sítě znázorněné na obrázku 7 ve frameworku TensorFlow

```
--dataset=DataSet
--net=EncoderDecoderNet
--test_root=../data/old/added_border_data/test/
--train_root=../data/old/added_border_data/train/
--model_dir=./segment/
--solution_dir=solutions/
--batch_size=5
--learning_rate=0.001
```

Zdrojový kód 3: Konfigurační soubor pro trénink neuronové sítě

Neuronové sítě, dataset i augmentaci reprezentujeme pomocí tříd. Framework obsahuje třídu `Net`, která implementuje základní funkcionalitu pro neuronové sítě vytvořené pomocí TensorFlow. Slouží tedy jako třída, ze které ostatní sítě dědí. Dataset umožňuje načítat data pro trénink i predikci a dále také data před předáním neuronové síti upravovat (padování, škálování apod.). Ve třídě `Dataset` je použit objekt `AugmentManager`, který se stará o augmentaci trénovacích dat.

Závěr

Výstupem práce je nová metoda pro segmentaci mikroskopických snímků jader buněk. Úloha byla řešena ve dvou krocích. V prvním kroku jsme oddělovali buňky od pozadí a v druhém kroku jsme rozdělovali klastry buněk. Pro oddělení jader buněk od pozadí jsme použili metodu prahování namísto umělých neuronových sítí a to z důvodu velké časové náročnosti přípravy trénovacího datasetu, který je pro učení sítě nezbytný. Pro rozdělování klastrů buněk jsme využili konvoluční neuronové sítě, které jsme naučili doplňovat chybějící části masky okraje. Pro experimentování s neuronovým sítěmi jsme vytvořili minimalistický framework, který přikládáme v příloze této práce. Dále v příloze také přikládáme natrénovaný model konvoluční neuronové sítě, který je možné pomocí frameworku použít pro rozdělování klastrů buněk. Celkový program přiložený k této práci je pouze ukázkový a jako hlavním účelem bylo ověřit proof of concept nové metody. Funkčnost nové metody pro rozdělování klastrů buněk jsme experimentálně ověřili a domníváme se, že je vhodná pro využití v praxi. Všechny požadavky definované zadáním práce se tedy podařilo naplnit.

V úvodní teoretické části práce jsme nejdříve definovali úlohu segmentace obrazu a uvedli jsme dva praktické příklady jejího využití. Dále jsme vysvětlili základní princip umělých neuronových sítí, tedy především k čemu umělé neuronové sítě slouží, jak jsou definované a jak je můžeme použít. V poslední části teoretického úvodu jsme se zaměřili na konvoluční neuronové sítě, které jsou určeny pro zpracování obrazu. Poslední dvě kapitoly se věnují samotnému experimentu. Nejdříve jsme popsali motivaci úlohy a hlavní myšlenky nové metody a v poslední kapitole jsme stručně popsali implementaci celkového řešení.

Přestože se podařilo dosáhnout poměrně vysoké přesnosti, řešení je možné dále vylepšit. Největší slabinou této úlohy jsou data, které je nutné manuálně připravit, což zabere mnoho času. Domníváme se, že pokud bychom měli k dispozici větší dataset, dosáhli bychom výrazně lepších výsledků. Další prostor ke zlepšení je bezpochyby v oddělování jader buněk od pozadí (první část úlohy). Pokud bychom namísto prahování využili neuronové sítě (pro které ale potřebujeme kvalitní trénovací dataset), získali bychom kvalitnější masky a rozdělení klastrů by bylo přesnější.

Conclusions

The result of the thesis is the new method for microscope image segmentation of cell nuceli. We solved the task in the two steps. In a first step, we separated cell nuceli from the background and in the second step we divided the clusters of cell nuclei. For the cell nuclei separation from the background we used thresholding instead of artificial neural networks because the preparation of training dataset is very time expensive. For the division of cell nuclei clusters we used convolutional neural networks, which was trained to fill missing part of border mask. For the experiments with nerual networks we developed a new minimalist framework which is attached to this thesis. In the attachment of this thesis we also provide trained model of convolutional neural netowrks, which is possible to run for division of cell nuclei clusters using the framework. The final software attached to this thesis serves just as a demonstation and its main purpose was to verify a proof of concept of the new method. We experimentally verified a functionallity of the new method and we assume that suitable for use in practice. All requirements defined in assigment of this thesis were accomplished.

First of all, in the theoretical part we definied the image segmentation task and we described two practical examples of its usage. We also explained the core principle of artificial neural networks, its purpose, how they work and how we can use them. In the last part of the theoretical introduction we focused on convolutional neural networks, which are used for computer vision. In the last two chapters we explained the experiment and the new method. We descibed the motiviation of the task, main ideas of the new method and in the last chapter we briefly described the implementation of the final solution.

Even though with the new method we achieved high accuracy, there are several ways how the method can be improved. The main weakness of the task are data, which is necessary to prepare manually and it is very time expensive. We assume, that if we have the bigger dataset, then we could achieve much better results. There is also more room for improvement in separation of the cell nuclei from the background. If we use neural networks (for which we need good training dataset) instead of the thresholding, we can get better masks and the division would be more accurate.

A Obsah přiloženého CD/DVD

doc/
: Diplomová práce ve formátu PDF včetně všech příloh a souborů potřebných k vygenerování této práce.

nucleus/
: Zdrojové kódy frameworku.

train.py
: Skript pro trénink neuronových sítí.

predict.py
: Skript pro predikci neuronových sítí.

settings/
: Konfigurační soubory pro trénink a predikci.

README
: Stručný popis frameworku, instrukce pro instalaci a příklad použití.

Navíc CD/DVD obsahuje:

data/
: Data připravená pro trénink a predikci.

model_border/results_45/
: Výsledky predikce neuronové sítě.

model_border/model-45.data-00000-of-00001
: Natrénovaný model konvoluční neuronové sítě.

Literatura

[1] DAPI. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-19].
Dostupné z: https://cs.wikipedia.org/wiki/DAPI

[2] KARPATHY, Andrej. Stanford CS class CS231n: Convolutional Neural Networks [online]. [cit. 2018-02-27].
Dostupné z: http://cs231n.github.io/convolutional-networks/

[3] LI, Hui, Jianfei CAI, Thi Nhat Anh NGUYEN a Jianmin ZHENG. A benchmark for semantic image segmentation [online]. [cit. 2018-03-06].

[4] GONZALEZ, Rafael C. a Richard E. WOODS. Digital image processing. 3rd ed. Upper Saddle River, N.J.: Prentice Hall, c2008. ISBN 01-316-8728-X.

[5] FARAMARZI, Azita, Mohammad Ali KHALILI, Azam AGHA-RAHIMI a Marjan OMIDI. Is there any correlation between oocyte polarization microscopy findings with embryo time lapse monitoring in ICSI program?. Archives of Gynecology and Obstetrics. 2017, 295(6), 1515-1522. DOI: 10.1007/s00404-017-4387-8. ISSN 0932-0067.
Dostupné z: http://link.springer.com/10.1007/s00404-017-4387-8

[6] Nondestructive testing. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-19].
Dostupné z: https://en.wikipedia.org/wiki/Nondestructive_testing

[7] Radiograph Interpretation - Welds - NDT Resource Center [online]. [cit. 2018-03-07].

[8] NG, Andrew. Coursera Machine Learning: Neural Networks: Representation [online]. [cit. 2018-04-19].
Dostupné z: https://www.coursera.org/learn/machine-learning/home/week/4

[9] KARPATHY, Andrej. Stanford CS class CS231n: Neural Networks Part 1: Setting up the Architecture [online]. [cit. 2018-03-11].
Dostupné z: http://cs231n.github.io/neural-networks-1/

[10] NG, Andrew. Coursera: Machine Learning [online]. [cit. 2018-03-14].
Dostupné z: https://www.coursera.org/learn/machine-learning/

[11] KARPATHY, Andrej. Stanford CS class CS231n: Backpropagation, Intuitions [online]. [cit. 2018-03-14].
Dostupné z: http://cs231n.github.io/optimization-2/

[12] MNIST For ML Beginners [online]. [cit. 2018-03-14].
Dostupné z:
https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners

[13] KARPATHY, Andrej. Stanford CS class CS231n: Image Classification [online]. [cit. 2018-03-14].
Dostupné z: http://cs231n.github.io/classification/

[14] Art and the Bible: The Abyss of Hell [online]. [cit. 2018-04-19].
Dostupné z: https://www.artbible.info/art/large/819.html

[15] SHANKAR, Sukrit, Duncan ROBERTSON, Yani IOANNOU, Antonio CRIMINISI a Roberto CIPOLLA. Refining Architectures of Deep Convolutional Neural Networks [online]. 22. května 2016, 9 [cit. 2018-03-18].
Dostupné z: https://arxiv.org/abs/1604.06832

[16] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON. ImageNet Classification with Deep Convolutional Neural Networks [online]. 2012 [cit. 2018-05-01]. Dostupné z: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[17] ImageNet: Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) [online]. [cit. 2018-05-01].
Dostupné z: http://image-net.org/challenges/LSVRC/2012/index

[18] SZEGEDY, Christian, Wei LIU, Yangqing JIA, et al. Going Deeper with Convolutions [online]. 2015 [cit. 2018-05-01].
Dostupné z: https://arxiv.org/pdf/1409.4842.pdf

[19] SIMONYAN, Karen a Andrew ZISSERMAN. Very Deep Convolutional Networks for Large-Scale Image Recognition [online]. 2014 [cit. 2018-05-01].
Dostupné z: https://arxiv.org/pdf/1409.1556.pdf

[20] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Deep Residual Learning for Image Recognition [online]. 2015 [cit. 2018-05-01].
Dostupné z: https://arxiv.org/pdf/1512.03385.pdf

[21] SWARBRICK JONES, Mike. Convolutional autoencoders in python/theano/lasagne [online]. 29. května 2015 [cit. 2018-03-18].
Dostupné z: https://swarbrickjones.wordpress.com/2015/04/29/convolutional-autoencoders-in-pythontheanolasagne/

[22] DUMOULIN, Vincent a Francesco VISIN. A guide to convolution arithmetic for deep learning [online]. 26. března 2016 [cit. 2018-03-18].
Dostupné z: https://arxiv.org/abs/1603.07285

[23] P. KINGMA, Diederik a Jimmy LEI BA. Adam: A Method for Stochastic Optimization [online]. 30. ledna 2017, 15 [cit. 2018-03-21].
Dostupné z: https://arxiv.org/pdf/1412.6980.pdf