VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Ondřej Doskočil

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV BIOMEDICÍNSKÉHO INŽENÝRSTVÍ
DEPARTMENT OF BIOMEDICAL ENGINEERING

DETEKCE BUNĚK POMOCÍ KONVOLUČNÍCH NEURONOVÝCH SÍTÍ
CELL DETECTION USING CONVOLUTIONAL NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Ondřej Doskočil

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Tomáš Vičar

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Biomedicínská technika a bioinformatika**

Ústav biomedicínského inženýrství

Student: Ondřej Doskočil ***ID:*** 203657

Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Detekce buněk pomocí konvolučních neuronových sítí

POKYNY PRO VYPRACOVÁNÍ:

1) Prostřednictvím odborné literatury se seznamte s metodami pro detekci objektů pomocí konvolučních neuronových sítí. 2) Proveďte literární rešerši se zaměřením na vhodné metody pro detekci buněk, které lze učit z anotací ve formě ohraničujícího rámečku nebo středu buňky. 3) Na základě literární rešerše vyberte nejvhodnější metodu/metody pro detekci buněk a ve zvoleném programovacím prostředí je realizujte. 4) Úspěšnost otestujte na dostupných datech a statisticky vyhodnoťte. 5) Proveďte diskusi dosažených výsledků, kde se zaměřte na srovnání s jinými metodami.

DOPORUČENÁ LITERATURA:

[1] HÖFENER, Henning, André HOMEYER, Nick WEISS, Jesper MOLIN, Claes F. LUNDSTRÖM a Horst K. HAHN. Deep learning nuclei detection: A simple approach can deliver state-of-the-art results. Computerized Medical Imaging and Graphics. 2018, 70, 43-52. DOI: 10.1016/j.compmedimag.2018.08.010. ISSN 08956111.

[2] LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU a Alexander C. BERG. SSD: Single Shot MultiBox Detector. In: Computer Vision – ECCV 2016. Cham: Springer International Publishing, 2016, 2016-09-17, s. 21-37. Lecture Notes in Computer Science. DOI: 10.1007/978--319-46448-0_2. ISBN 978-3-319-46447-3.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 5.6.2020

Vedoucí práce: Ing. Tomáš Vičar

prof. Ing. Ivo Provazník, Ph.D.
předseda oborové rady

ABSTRAKT

Tato bakalářská práce se zabývá využitím konvolučních neuronových sítí pro detekci buněk v obrazových datech. Teoretická část obsahuje popis fungování těchto sítí a jejich různých architektur. V praktické části byly tyto sítě implementovány a trénovány na dostupném datasetu. Každá z těchto sítí využívá však jiný přístup k detekci. Nakonec byly jednotlivé sítě statisticky vyhodnoceny a byl provedena diskuse.

KLÍČOVÁ SLOVA

Konvoluční neuronové sítě, detekce objektů, buňky, PyTorch

ABSTRACT

This bachelor thesis deals with the use of convolutional neural networks for cell detection in image data. The theoretical part contains a description of the functioning of these networks and their various architectures. In the practical part, these networks were implemented and trained on an available dataset. However, each of these networks uses a different approach to detection. Finally, the individual networks were statistically evaluated and a discussion was conducted.

KEYWORDS

Convolutional neural networks, object detection, cells, PyTorch

DOSKOČIL, Ondřej. *Detekce buněk pomocí konvolučních neuronových sítí*. Brno, Rok, 51 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav biomedicínského inženýrství. Vedoucí práce: Ing. Tomáš Vičar

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Detekce buněk pomocí konvolučních neuronových sítí“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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 bakalářské práce panu Ing.Tomáši Vičarovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato práce se věnuje detekci buněk pomocí konvolučních neuronových sítí. Automatická detekce buněk je zájmem široké oblasti lékařských a klinických aplikací. Detekce pomáhá urychlovat výzkum každé nemoci od rakoviny přes srdeční onemocnění až po různé vzácné nemoci. Identifikace buňky a jeho jádra je výchozím bodem většiny analýz, jelikož většina buněk obsahuje genetické informace, které jsou velmi důležité. Detekce tak umožňuje identifikovat každou jednotlivou buňku ve vzorku, a měřením toho jak daná buňka reaguje na různá ošetření je možné pochopit její biologické procesy.

V uplynulých letech byly provedeny různé výzkumy týkající se použití konvolučních neuronových sítí pro detekci buněk. Bylo dosaženo velkého pokroku a to zejména díky větší dostupnosti anotovaných datových sad a také rozvoji samotných konvolučních neuronových sítí.

Cílem této práce je pochopení principů detekce objektů v obrazech a jejich využití pro detekci buněk. První část práce se věnuje literární rešerši dané problematiky. Tedy nejprve základům jako je umělý neuron, jednoduché neuronové sítě, jejich učení a optimalizační algoritmy. Dále pak samotné konvoluční neuronové sítě. Zvláštní důraz byl kladen na různé způsoby jejich architektur a princip jejich fungování.

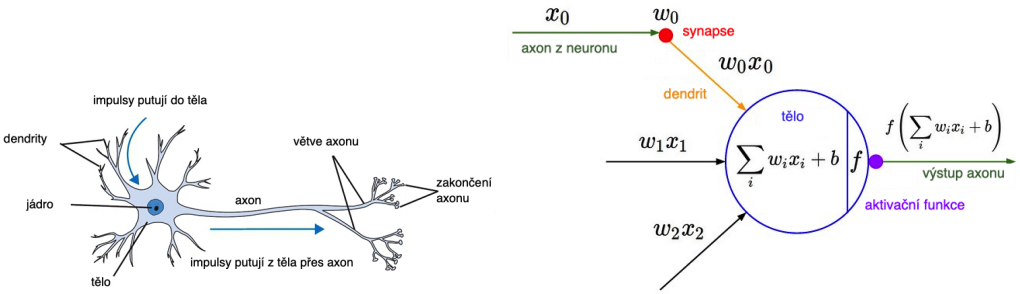
Dle získaných znalostí byly některé tyto architektury navrhnuty a implementovány pro detekci buněk. Architektury byly trénovány na dostupných datech a nakonec statisticky vyhodnoceny. Závěrem byla provedena diskuse.

1 Neuronové sítě

Oblast neuronových sítí byla původně inspirována cílem matematického modelování biologických nervových systémů, ale od té doby se rozcházela a stala se věcí inženýrství a dosahování dobrých výsledků v úlohách strojového učení.

1.1 Umělý model neuronu

Matematický model neuronu, také nazývaný perceptron, je inspirován anatomickou stavbou biologického nervového systému a jeho způsobem zpracování informací. Základní výpočetní jednotkou nervového systému je neuron. V lidském nervovém systému se nachází přibližně 86 miliard neuronů a jsou spojeny s přibližně 10 na 14 - 10 na 15 synapsemi. [26], [9] Na následujícím obrázku můžeme vidět stavbu biologického neuronu a jeho matematického modelu.



(a) Biologický neuron (b) Matematický model neuronu

Obr. 1.1: Porovnání biologického a matematického neuronu, převzato z [1]

Každý neuron přijímá vstupní signály ze svých dendritů a vytváří výstupní signály podél svého axonu. Axon nakonec odbočuje a spojuje se pomocí synapsí s dendrity jiných neuronů. Matematický model funguje obdobně. Vstupy x_0, ... x_n reprezentují informace, které jdou do biologické neuronu přes dendrity. Tyto vstupy jsou váhované vektorem vah w_0, ... w_n. Biologická analogie pro váhy je synapse (synaptická síla), která udává jak moc je dané spojení silné (významné). Všechny vstupy matematického modelu se váhují, a od jejich celkového součtu se odečte práh. Celkový součet nazývaný aktivace pak vstupuje jako argument do aktivační funkce f, která určuje zda-li bude daný neuron aktivovaný. [26] Dle toho můžeme model matematicky zapsat jako: [9]

$$y = f\left(\sum_{i=1}^{N} w_i x_i - b\right) \tag{1.1}$$

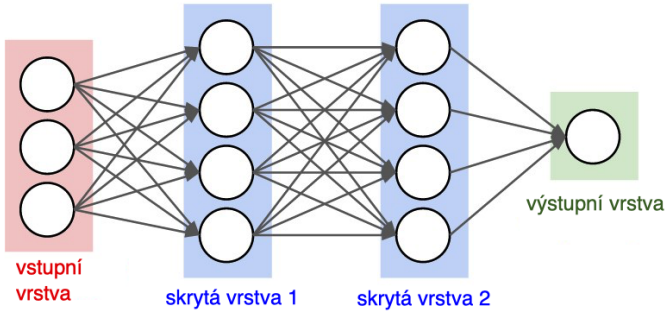
kde x je vstupní vektor neuronu a w je vektor obsahující jeho váhy, b je práh (často označovaný jako bias) a f je aktivační funkce. Charakteristiky nejpoužívanějších aktivačních funkcí můžeme vidět v tabulce 1.1.

Tab. 1.1: Přehled nejpoužívanějších aktivačních funkcí

Název aktivační funkce	Rovnice	Obor hodnot
Skoková	$f(x) = sign(x)$	{-1, 1}
Sigmoidální	$f(x) = \frac{1}{1+e^{-x}}$	(0,1)
Hyperbolický tangens	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1, 1)
ReLu	$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	$\langle 0, \infty)$
Softmax	$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}}$	(0,1)

1.2 Vícevrstvá neuronová síť

Vícevrstvá neuronová síť obsahuje jednu nebo více skrytých vrstev neuronů mezi vstupní a výstupní vrstvou. Počty neuronů v každé vrstvě závisí na typu a požadavcích úlohy, kterou daná síť řeší. Každý neuron z jedné vrstvy je spojen se všemi neurony vrstvy následující. Čím více skrytých vrstev použijeme, tím více bude síť schopna řešit komplexnější úlohy.



Obr. 1.2: Vícevrstvá neuronová síť, převzato z [1]

1.3 Učení neuronových sítí

Učení neuronových sítí můžeme rozdělit do dvou hlavních skupin. Učení s učitelem a učení bez učitele.

Učení s učitelem (supervised) je učení, při kterém máme k dispozici páry vstupů a jejich odpovídajících výstupů (anotací). Neuronová síť během učení přivádí vstupní vektory, které zpracovává a následně porovnává její výstup s danou anotací. Pokud se výstup od anotace liší, počítá se chybová funkce, podle které se následně upraví váhy.

Učení bez učitele (unsupervised) se liší v tom, nemáme k dispozici žádné anotace. Hodnoty vah jsou nastavené jen podle vstupních vektorů. Síť se snaží najít najít mezi vstupy podobnosti, podle kterých je shlukuje.

Zpětné šíření chyby

Abychom mohli upravovat jednotlivé parametry vícevrstvé neuronové sítě, musíme znát odchylku každého neuronu zvlášť. Tato odchylka se počítá pomocí algoritmu zpětného šíření chyby tzv. backpropagation. Tento algoritmus se dá rozdělit na tři hlavní části: dopředné šíření vstupního signálu, zpětné šíření chyby a aktualizace vah neuronů. Poté co projde vstupní vektor všemi vrstvami sítě, vypočítá se odchylka mezi výstupem sítě a očekávaným výstupem. Tato odchylka, nazývaná chybová funkce, udává, jak moc se výstup sítě liší od ideálního výstupu. Pro naučení sítě je nutné chybu minimalizovovat. Nejčastěji se jako chybová funkce používá střední kvadratická odchylka nebo křížová entropie. Po výpočtu chyby se následně chyba propaguje směrem k vstupní vrstvě, jelikož platí že chyba ve vrstvě n se rozdělí mezi neurony z vrstvy $n - 1$. Když máme chybu pro každý neuron, můžeme dle toho změnit jeho parametry. Změna těchto parametrů závisí na použitém optimalizačním algoritmu. Následně je přiveden znovu vstupní vektor a proces se opakuje dokud síť nenaučíme, tedy nemáme přijatelné hodnoty výstupu. [15]

1.4 Optimalizační algoritmy

Optimalizační algoritmy slouží k minimalizaci chybové funkce, která je závislá na nastavení vnitřních parametrů sítě - vah a prahu jednotlivých neuronů. V průběhu učení se snažíme tyto parametry vhodně upravovat tak, aby chybová funkce byla co nejmenší.

Gradientní sestup

Jeden z nejzákladnějších optimalizačních algoritmů je gradientní sestup. Je velmi používaný v řadě regresních a klasifikačních úloh. Princip je založený na výpočtu gradientu chybové funkce a následné úpravě parametrů tak, aby směr chybové funkce byl proti směru gradientu. Úprava parametrů se dá zapsat jako: [27]

$$\theta = \theta - \eta \times \nabla J(\theta) \tag{1.2}$$

kde η je rychlost učení a $\nabla J(\theta)$ je gradient chybové funkce. Mezi nevýhody patří to, že se počítá pouze jednou za epochu tedy pro celý dataset. Tím je algoritmus vypočetně náročný, pokud se síť trénuje na velkém množství dat. Důležité je aby vstupní data byli náhodně seřazeny, kvůli možnému zkreslení a kolísání gradientu.

Stochastický Gradientní sestup - SGD

Tento algoritmus je variantou zmíněného Gradientního sestupu. Rozdíl je ten, že SGD upravuje parametry po každém jednom vstupu. Tedy pokud dataset obsahuje 1000 vstupních dat, parametry budou aktualizovány 1000x za epochu. Úprava parametrů se dá zapsat jako: [27]

$$\theta = \theta - \eta \times \nabla J\left(\theta; x^i; y^i\right) \tag{1.3}$$

kde x^i je aktuální vstup a y^i je požadovaný výstup. Tím že upravuje parametry častěji, konverguje mnohem rychleji a je vypočetně méně náročný. Důležité u SGD je, aby vstupní data byly náhodně seřazené. Pokud by se stalo, že data budou seřazené podle nějakého vzoru, mohlo by to způsobit zkreslení a kolísání gradientu, což způsobí že nebude konvergovat nebo bude dosáhnutí konvergence trvat velmi dlouho.

Momentum

Momentum je metoda, která pomáhá urychlit SGD ve správném směru a tlumí oscilace. To se děje přidáním parametru γ, který násobí aktualizační vektor z minulého kroku a promítá se do aktuálního. Úprava parametru je zřejmá z jeho rovnice: [27]

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta) \tag{1.4}$$

úprava vah pak:

$$\theta = \theta - v_t \tag{1.5}$$

Velikost γ se nastavuje obvykle kolem hodnoty 0.9. Momentum se zvyšuje pro rozměry, jejichž gradienty směřují ve stejném směru. Tím je dosaženo ještě rychlejší konvergence a také snížení oscilací.

Adam

Adam nebo-li Adaptive Moment Estimation je optimalizační algoritmus speciálně určený pro trénovaní hlubokých neuronových sítí. Počítá adaptivní rychlost učení pro každý paramter. Adam používá pro úpravu parametrů informace o gradientu ale také o čtvercovém gradientu. Pracuje tedy s momentem prvního a druhého řádu. Odhad těchto momentů je podle těchto rovnic: [12]

$$\hat{v}_t = \frac{v_t}{1 - \beta_t^1} \tag{1.6}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_t^2} \tag{1.7}$$

kde v_t je první moment a m_t je druhý moment. β_t^1 a β_t^2 jsou parametry řídící exponenciální rychlost poklesu momentu.

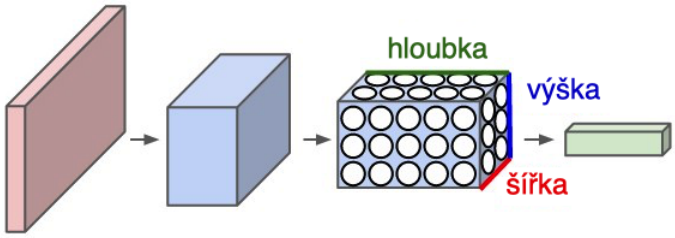
pro úpravu parametrů pak:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{m}_t} + \epsilon}\hat{v}_t \tag{1.8}$$

Adam dosahuje při učení neuronových sítí velmi dobré výsledky a patří k nejvíce používaným optimalizačním algoritmům po boku SGD s využitím momenta. Adam odstraňuje většinu problémů jiných optimalizačních algoritmů a to především rychlost konvergence a odchylky při aktualizaci parametrů.

2 Konvoluční neuronové sítě

Konvoluční neuronové sítě (CNN, ConvNets) jsou velmi podobné běžným neuronovým sítím. Obsahují vrstvy neuronů, které mají své váhy a aktivační funkce. Rozdíl u konvolučních neuronových sítí je ten, že předpokládají na vstup obraz, což nám umožňuje přidat určité vlastnosti do samotné architektury. Tyto vlastnosti pak zefektivňují dopřednou funkci při implementaci a velmi snižují množství parametrů v celé síti. [13]

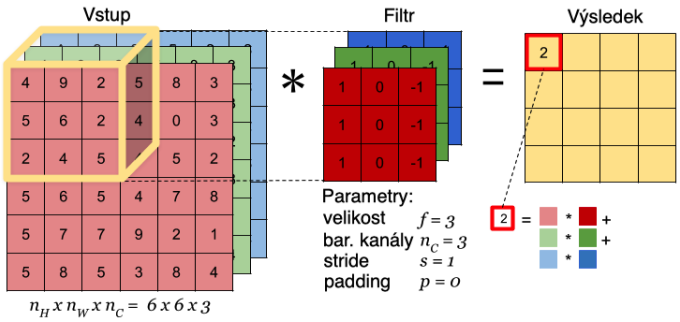


Obr. 2.1: Konvoluční neuronová sít, převzato z [1]

Tím, že předpokládáme na vstupu obraz, jsou jednotlivé neurony uspořádané do 3 rozměrů: šířka (width), výška (height) a hloubka (depth). Jak uvidíme později, neurony v jednotlivých vrstvách budou připojeny jen s určitou oblastí vrstvy předešlé, narozdíl od běžné neuronové sítě kde jsou všechny neurony plně propojené. Základní architektura konvolučních sítí se skládá ze vstupní vrstvy, konvoluční vrstvy, ReLu aktivační funkce, pooling vrstvy a plně propojené vrstvy.

2.1 Konvoluční vrstva

Konvoluční vsrstva je základním kamenem CNN. Hlavní smysl konvoluční vrstvy je extrakce jednotlivých příznaků z obrazu. Vrstva se skládá z několika filtrů (kernelů), které postupně procházejí celý vstupní obraz provádí s ním konvoluci. Protože vstupy jako obrazy mají velké rozměry, je velmi nepraktické (vypočetně náročné) spojovat každý neuron vstupní vrstvy se všemi neurony konvoluční vrstvy. Místo toho spojíme pouze určitý lokální region obrazu. Tento region se kterým filtr právě provádí konvoluci se nazývá receptive field. Každé toto spojení má nějakou váhu a nějaký práh a jelikož chceme jedním filtrem získat určitý příznak, tak se váhy a prahy v každé poloze pro ten samý filtr sdílí. Tím pádem si nepotřebujeme pamatovat tolik parametrů. Princip konvoluce můžeme vidět na následujícím obrázku. [13]



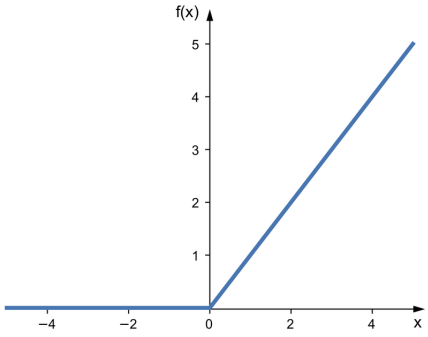
Obr. 2.2: Princip konvoluce

Jak je vidět na obrázku, filtr má vždy stejné rozměry jako region se kterým konvoluje.

Výstup konvoluční vrstvy se nazýva příznaková mapa a její rozměry určují 3 hyperparametry: hloubka neboli počet filtrů (depth), krok posouvání filtru (stride) a přidávání nul do vstupní matice (zero-padding). Každý filtr se učí hledat jiné příznaky například vertikální/horizontální hrany nebo barevné shluky. Krok určuje o kolik pixelů budeme posouvat filtr. Obvyklé nastavení kroku je na hodnotě 1 nebo 2. Větší krok způsobí menší výstup. Přidávání nul do vstupní matice umožňuje přímo kontrolovat velikost výstupu - většinou chceme aby výstupní matice byla stejná jako vstupní.

2.2 ReLU

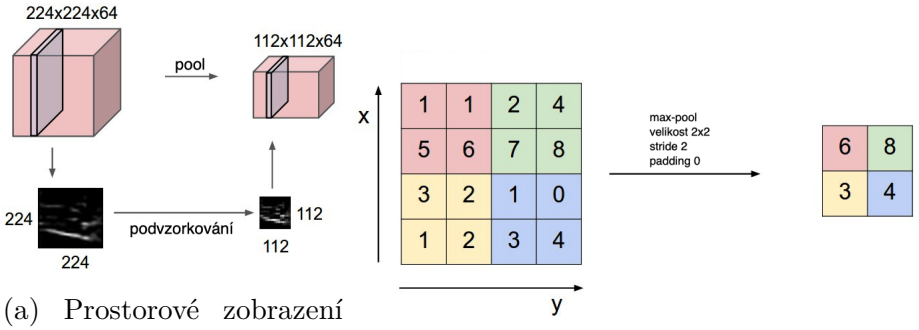
ReLU neboli Rectified Linear Unit je aktivační funkce, která následuje po každé konvoluční vrstvě. Díky ní přidáme do sítě nelinearitu. Kdybychom tuto funkci nepoužili, dostali bychom ze sítě pouze lineární regresní model. [19]



Obr. 2.3: Graf funkce ReLu

2.3 Pooling vrstva

Pooling vrstva se většinou vkládá mezi konvoluční vrstvy z důvodu zmenšení prostorové velikosti, kvůli snížení množství parametrů a tedy celkové výpočetní náročnosti. Nejčastěji se pro pooling používá operace max, ale můžou být použity také například funkce min nebo average (průměr). Princip max poolingu můžeme vidět na následujícím obrázku. [19]



(a) Prostorové zobrazení poolingu

(b) Max pooling

Obr. 2.4: Princip max poolingu, převzato z [1]

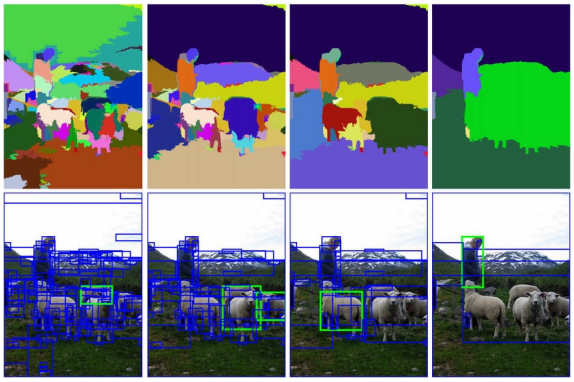
2.4 Plně propojená vrstva

Plně propojená (FC, fully-connected) vrstva je poslední vrstva, která představuje tradiční neuronovou síť, kde jsou všechny neurony propojeny s neurony předcházející vrstvy. Na výstupu už není příznaková mapa, jako je tomu u konvoluční a pooling vrstvy, ale pouze vektor pravděpodobností dané třídy v obrázku. Tento vektor je stejně dlouhý jako je počet rozlišovaných tříd.

3 Architektury konvolučních sítí

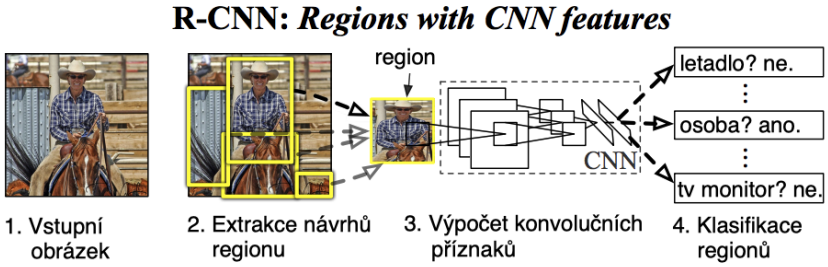
3.1 R-CNN

Jeden z prvních průlomů CNN v oblasti počítačové detekce objektů je síť R-CNN zvaná také Regions with CNN features jejíž autor je Ross Girshick. Tato síť je rozdělena na 3 fáze. V první fázi se pomocí selektivního vyhledávání extrahují z obrazu oblasti (region proposals), které by mohli patřit rozpoznávanému objektu v závislosti na barvě, struktuře, velikosti a tvaru. Podobné regiony se iterativně kombinují a tím získáme objekty (object proposals). Autoři používají algoritmus selektivního vyhledávání pro vytvoření 2000 kategoricky nezávislých regionů pro každý obrázek. [5], [30]



Obr. 3.1: Ukázka algoritmu selektivního vyhledávání převzatá z [30]

V další fázi se pomocí CNN extrahují příznaky ze všech nalezených regionů v podobě příznakového vektoru. Nakonec se vektory klasifikují s použitím individuálního lineárního klasifikátoru SVM (Support Vector Machine), který se učí pro každou rozpoznávanou třídu. Aby byla zvýšena přesnost lokalizace objektu, provádí se ještě regrese ohraničujícího rámečku (bounding boxu). [5].Architekturu můžeme vidět na Obr. 3.2.

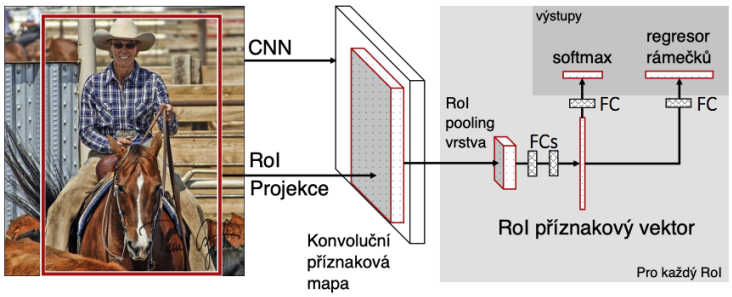


Obr. 3.2: Architektura R-CNN, převzato z [5]

Velký problém R-CNN je její velká časová náročnost jak na trénování tak na detekci. Z toho důvodu nemůže být používána pro real-time detekci. Dalším problémem je, že algoritmus selektivního vyhledávání je neměnný, tedy neprobíhá zde žádné učení. To by mohlo vést ke špatnému detekování oblastí. Zrychlení sítě přišlo od stejného autora v podobě Fast R-CNN.

3.2 Fast R-CNN

Rozdíl oproti běžné R-CNN je ten, že místo toho, abychom udělali konvoluci z každého nalezeného regionu, pošleme do CNN rovnou celý obraz. Ten projde sérií konvolucí a max poolingu a výstupem je příznaková mapa. Z příznakové mapy se pomocí roi-poolingu extrahuje pro každý objekt z navrhovaného regionu (RoI - Region of Interest) malá příznaková mapa s fixní velikostí $H \times W$. Ta vstupuje do sekvence plně propojených vrstev, ze kterých vznikne příznakový vektor jdoucí do dvou výstupních vrstev. Jedna produkuje softmaxové pravděpodobnosti detekovaných tříd a další produkuje reálné hodnoty ohraničujících rámečků. [4] Architekturu můžeme vidět na Obr. 3.3.

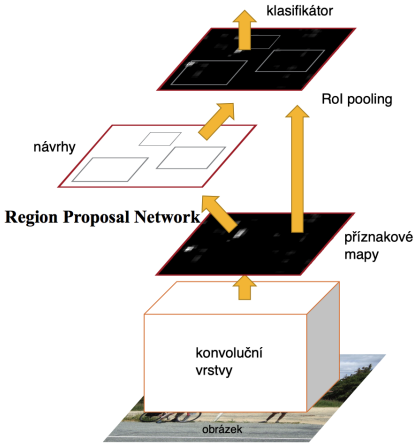


Obr. 3.3: Architektura Fast R-CNN, převzato z [4]

Fast R-CNN je oproti svému předchůdci mnohonásobně rychlejší, přesto ale zde pořád zůstává algoritmus selektivního vyhledávání, který je velmi pomalý. Tento algoritmus odstranil Shaoqing Ren se svou sítí Faster R-CNN. [24]

3.3 Faster R-CNN

Faster R-CNN se skládá ze dvou modulů. Podobně jako u Fast R-CNN obraz prvně vstupuje do CNN, ze které získáme příznakovou mapu. Místo použití algoritmu selektivního vyhledávání je zde samostatná síť, která oblasti navrhne tzv. Region Proposal Network (RPN). Tento modul říká detektoru kde hledat objekty. Jako detektor je použitá Fast R-CNN [24]. Architekturu můžeme vidět na Obr. 3.4.



Obr. 3.4: Architektura Faster R-CNN, převzato z [24]

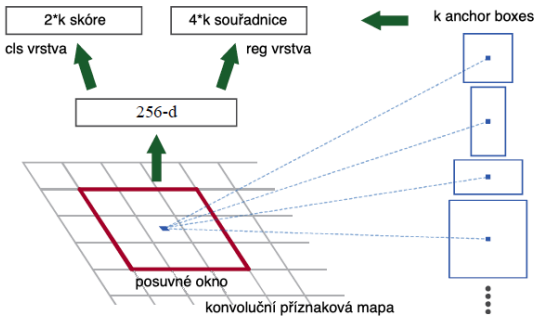
Region Proposal Network

Na vstup RPN je přiveden obraz jakékoliv velikosti, ze kterého síť vytvoří čtvercové návrhy objektů, kde každý má své objectness skóre[1]. Protože cílem je snížení vypočetní náročnosti, předpokládá se, že RPN má sdílené konvoluční vrstvy s Fast R-CNN detektorem. Tímto způsobem se sdílejí přímo výpočty a není tedy nutné znovu počítat příznakovou mapu.

Při vytváření regionů posouváme po příznakové mapě poslední sdílené konvoluční vrstvy malou síť, která z ní vždy vezme okno o velikosti $n \times n$. Toto okno je promítnuté do příznaku nižší dimenze. Tento příznak pak vstupuje do do dvou plně propojených vrstev. Jedna vrstva klasifikuje (cls) ohraničující rámeček pomocí

[1] objectness skóre vyjadřuje míru příslušnosti objektu do třídy oproti pozadí

softmax funkce. Druhá provádí regresi (reg) ohraničujícího rámečku. Malou sítě můžeme vidět na Obr. 3.5. Při každém posunu okna se predikuje zároveň několik oblastí návrhu (region proposal) jejichž maximální počet určuje k. Počet návrhů závisí na počtu k referenčních rámečcích, které autoři označují jako anchors. Každý anchor má střed uprostřed posuvného okna a je dán určitým měřítkem a poměrem stran. Anchors slouží pro definici tvaru objektů. Tím, že pro každou pozici posuvného okna máme několik anchors o různých rozměrech, jsou návrhy translačně invariantní. Tedy pokud se objekt posune, RPN navrhne stejný návrh regionu. Celkový počet anchors se dá vyjádřit jako $W \times H \times k$ kde $W \times H$ jsou rozměry příznakové mapy. [24]



Obr. 3.5: Region Proposal Network, na obrázku je vidět posuvné okno, které má v každé pozici několik anchors, které definují tvar navrhovaných objektů. Převzato z [24]

3.4 YOLO

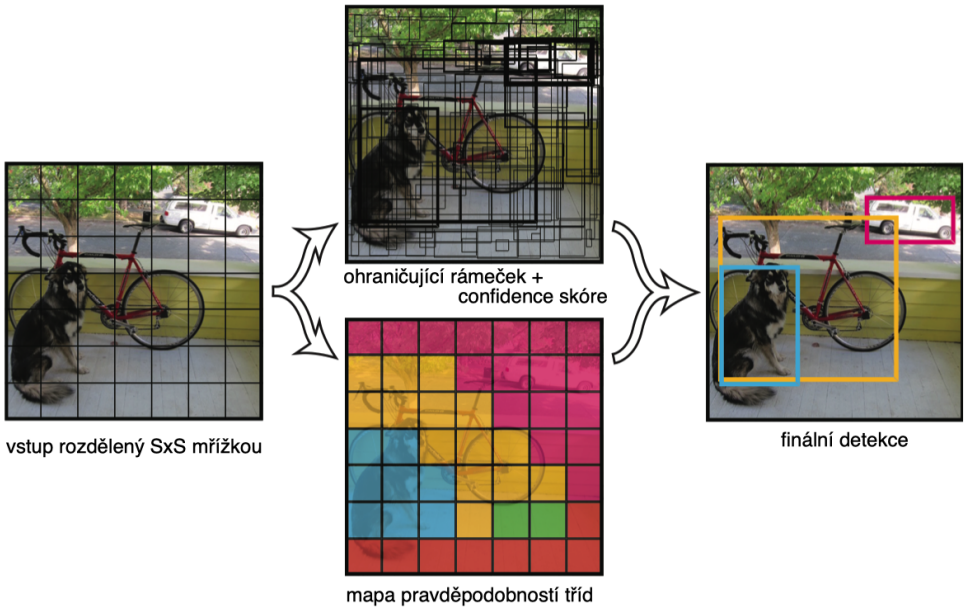
YOLO se skládá z jedné konvoluční sítě, která je schopná přímo z pixelů obrazu detekovat ohraničující rámečky a jejich pravděpodobnosti třídy. Trénink probíhá na celých obrázcích a přímo optimalizuje výkon detekce. Díky tomu že je systém jednotný, má mnoho výhod oproti tradičním metodám detekce jako je např. R-CNN. Největší výhodnou je mnohonásobná rychlost, kvůli které je možné YOLO použít pro real-time detekci. Na rozdíl od technik založených na návrhu regionů a posuvných oken, YOLO vidí při tréninku i detekci celý obraz, takže implicitně kóduje informace o třídách i jejich vzhledu. YOLO se při tréninku snaží co nejvíce generalizovat znázornění objektu. Tedy pokud například budeme trénovat na datech pocházejících z přírody a testovat na uměle vytvořených, YOLO zdaleka překoná systémy typu R-CNN. Díky tomu je tento systém velmi vhodný pro detekci objektů, které pocházejí z různých zobrazovacích modalit. [21]

Detekce

Pro detekci rámečků využívá síť příznaky celého obrázku. Stejně tak síť detekuje rámečky pro všechny třídy zároveň.

YOLO rozděluje vstupní obraz do $S \times S$ mřížky. Pokud se střed objektu nachází v buňce mřížky, je tato buňka zodpovědná za detekci tohoto objektu. Každá buňka mřížky detekuje B ohraničujících rámečků a jejich confidence score, které říká, jaká je pravděpodobnost, že rámeček obsahuje objekt a jak přesný je samotný rámeček. Confidence score obecně definujeme jako $Pr\,(Object) \times IOU_{pred}^{truth}$. Pokud v buňce nebude žádný objekt, confidence score rámečku bude nula. V ostatních případech je skóre rovno IoU[2] mezi predikovaným a pravdivým rámečkem.

Každý rámeček se skládá z pěti predikcí: $x, y, w, h, confidence$ kde x, y reprezentují střed rámečku, který je vztažený k dané buňce mřížky. w, h je šířka a výška vztažená k celému obrázku. Každá buňka také predikuje C podmíněné pravděpodobnosti tříd, které definujeme jako $Pr\,(Class_i|Object)$. To nám udává pravděpodobnost třídy pro objekt, který se nachází v dané buňce. Každá buňka může obsahovat pouze jednu třídu, přestože přes ni prochází více rámečků viz Obr. 3.6.



Obr. 3.6: Znázornění způsobu detekce sítě YOLO, převzato z [21]

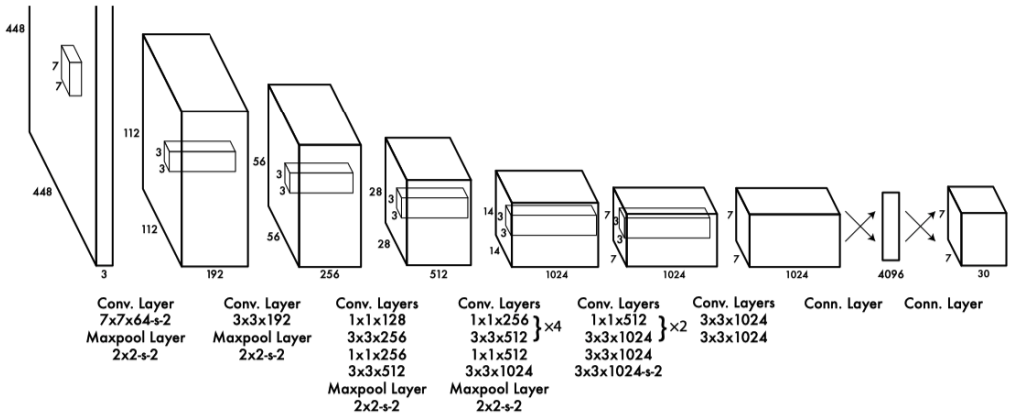
[2]Intersection over Union - hodnotící metrika, vyjadřuje podíl plochy průniku a plochy sjednocení mezi predikovaným a pravdivým (ground truth - anotace) rámečkem

Výsledné skóre je tedy dáno součinem pravděpodobnosti třídy a confidence skóre jednotlivých rámečků. Tím získáme přesnost rámečku a pravděpodobnost třídy.

$$Pr\left(Class_i|Object\right) \times Pr\left(Object\right) \times IOU_{pred}^{truth} = Pr\left(Class_i\right) \times IOU_{pred}^{truth} \quad (3.1)$$

Architektura

Architektura je inspirována sítí GoogLeNet [29], která se používá pro klasifikaci obrazů. YOLO je složené z 24 konvolučních vrstev následovaných 2 plně propojenými vrstvami. Používají se zde 1×1 redukční vrstvy pro redukci příznaků z vrstvy předešlé, následované 3×3 konvolučními vrstvami. Celá architektura je vidět na následujícím obrázku.



Obr. 3.7: Architektura sítě YOLO, převzato z [21]

3.5 YOLOv3

Velká nevýhoda původní realizace sítě YOLO, byl problém s detekcí malých objektů, které navíc byly blízko u sebe. Proto by bylo její použití pro detekci buněk nevhodné. Oba tyto problémy však řeší vylepšená architektura v podobě YOLOv3 od stejného autora. Podobně jako v případě Faster R-CNN, síť využívá pro detekci anchors (v článku nazývané priors). Síť predikuje 4 souřadnice pro každý ohraničující obrázek: t_x, t_y, t_w, t_h a $confidence$. Souřadnice ohraničujícího rámečku jsou pak dány následujícími rovnicemi:

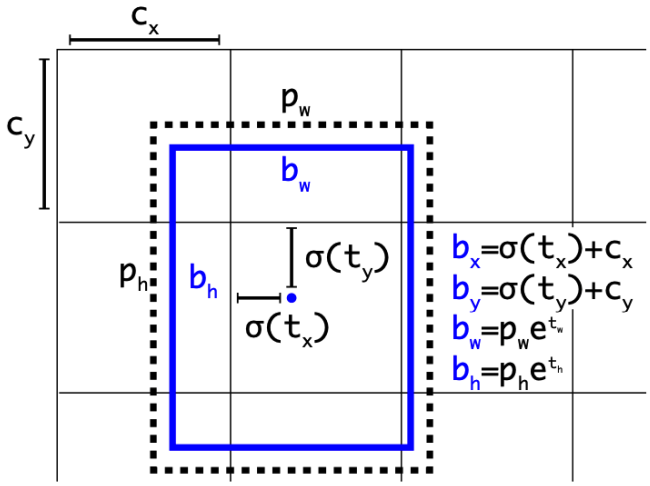
$$b_x = \sigma(t_x) + c_x \tag{3.2}$$

$$b_y = \sigma(t_y) + c_y \tag{3.3}$$

$$b_w = p_w e^{t^w} \tag{3.4}$$

$$b_h = p_h e^{t^h} \tag{3.5}$$

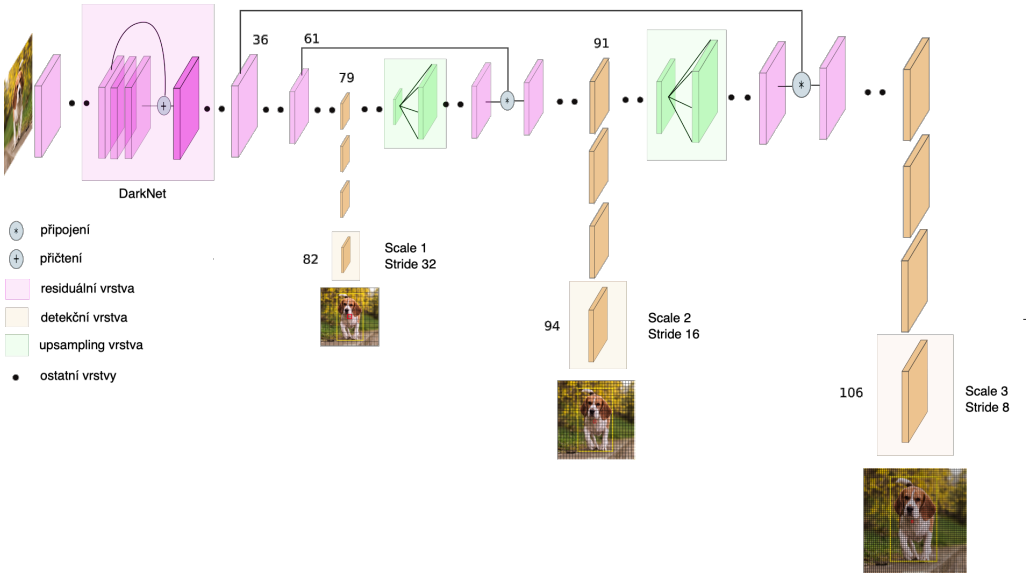
kde b_x, b_y, b_w, b_h jsou souřadnice ohraničujícího rámečku vztažené k celému obrázku, t_x, t_y, t_w, t_h jsou souřadnice rámečku, které predikuje síť vztažené k dané buňce, c_x, c_y je odsazení dané buňky od levého horního rohu obrázku a p_w, p_h je pak šířka a výška anchorboxu. Pochopení rovnic je zřejmé z obrázku 3.8. YOLOv3 také počítá pro každý ohraničující rámeček objectness skóre pomocí logistické regrese. Objectness se rovná 1 pokud anchor box má největší IoU s pravdivým rámečkem ze všech ostatních anchors. Pokud anchor není nejlepší, přestože má IoU nad určitý prah, tak se ignoruje. Síť tedy přiřazuje ke každému pravdivému rámečku pouze jeden anchor box. Tím je docíleno, že ostatní anchors nezpůsobují žádnou chybu. [23]



Obr. 3.8: Zobrazení odvození rovnic pro predikci rámečku z anchors sítě YOLOv3, modře je predikovaný rámeček a černě přerušovaně je anchor, který ho definuje. převzato z [22]

Architektura

Architektura je postavená na síti Darknet-53, obsahující 53 konvolučních vrstev, které extrahují z obrazu příznaky. YOLOv3 predikuje rámečky na třech různých místech sítě. Využívá se zde podobného principu jako u pyramidových [14] sítí. Po hlavní extrakci příznaků ze sítě Darknet-53 následuje několik konvolučních vrstev. Poslední z těchto vrstev, je první místo predikce rámečků. Následně se provede převzorkování na 2x větší velikost a připojí se příznaková mapa z předešlé části. Takovému spojení se říká skip connection. Toto spojení se používá, protože menší příznakové mapy poskytují lepší semantické informace a větší příznakové mapy poskytují detailnější informace o objektu [10]. Dále síť obsahuje další konvoluční vrstvy a provádí se druhá predikce. Tento postup se opakuje ještě jednou a nakonec následuje poslední predikce, která těží ze všech předchozích výpočtu stejně tak i z příznaků z počátku sítě. [23]



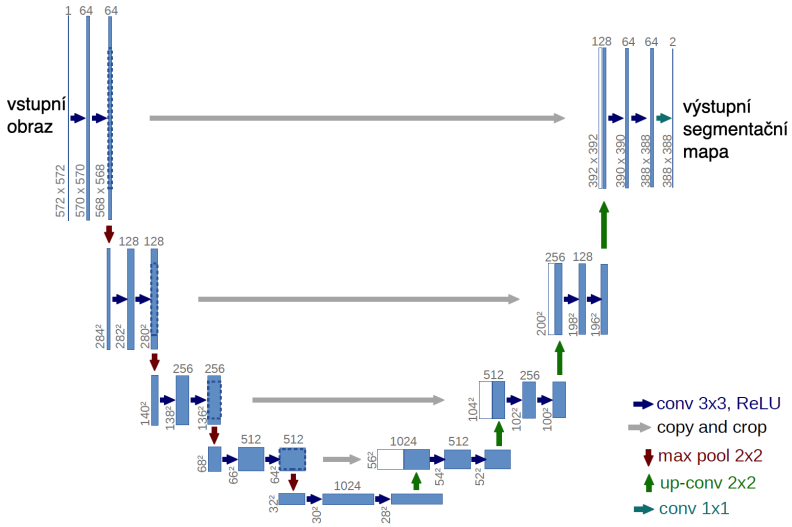
Obr. 3.9: Architektura sítě YOLOv3, převzato z [11]

3.6 U-Net

Ačkoliv se tato síť používá nejvíce při segmentaci najde své uplatnění i pro detekci. Segmentační metody se dají použít pro detekování objektu na základě generování tzv. heatmapy [3] [8]. Výstup sítě tedy není ohraničující rámeček jako v případě R-CNN nebo YOLO, ale obraz, ze kterého si vhodným zpracováním získáme polohu objektu. Toto zpracování je popsáno v kapitole 4.4

U-Net je postavená na plně konvoluční síti [28]. Skládá se z kontraktilní a expanzní dráhy, které jsou více-méně symetrické. Výsledkem je architektura ve tvaru písmene U jakou můžeme vidět na Obr. 3.10. Kontraktilní dráha (vlevo) je sestrojena z několika bloků, které mají typickou architekturu konvoluční neuronové sítě. Každý blok se skládá z opakované aplikace konvoluční (3x3) vrstvy s ReLU aktivační funkcí, následovanou (2x2) max-poolingem s krokem 2 pro podvzorkování (downsampling). Po každém bloku se počet příznakových map zdvojnásobí, ale naopak se zmenší její velikost. Tím je schopna síť najít i velmi komplexní příznaky.

Expanzivní dráha (vpravo) je postavena velmi podobně. Nejprve se provádí převzorkování (upsampling) pomocí transponované konvoluce, která zmenší počet příznakových map na polovinu, ale její velikost se zvětší. Následně připojíme (skip-connection) oříznutou příznakovou mapu z korespondující úrovně kontraktilní dráhy a provedeme dvojitou konvoluci s ReLU aktivační funkcí. Oříznutí je nutné kvůli ztrátě hraničních pixelů při každé konvoluci. Připojení způsobí, že příznaky naučené během kontraktilní dráhy budou použity pro rekonstrukci obrazu. V poslední vrstvě se provede (1x1) konvoluce pro namapování příznakového vektoru k danému počtu tříd. Není zde tedy žádná plně propojená vrstva [25].



Obr. 3.10: Architektura U-Net, převzato z [25]

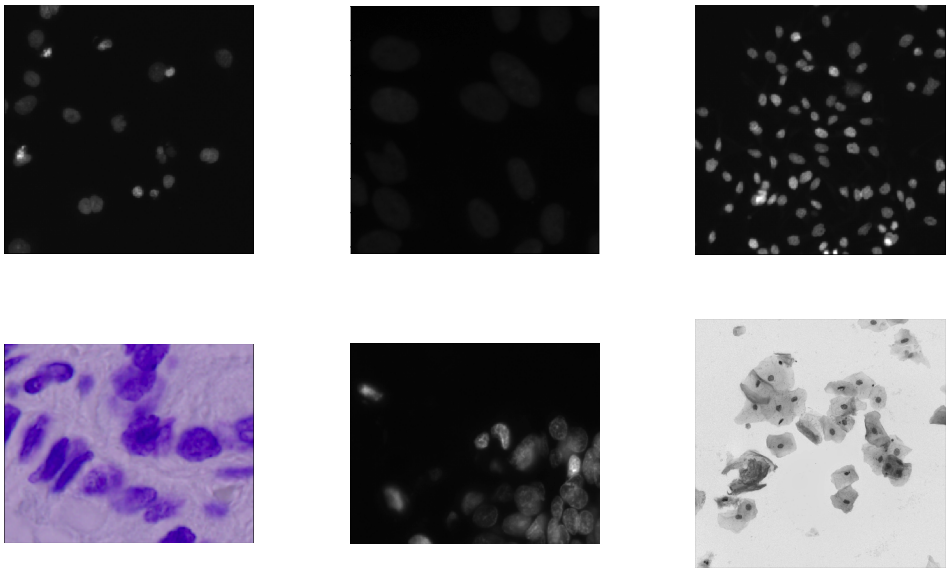
4 Návrh a implementace

Návrh a implementace probíhala v programovacím jazyce Python s deep learningovu knihovnou PyTorch [20]. Jako programovací prostředí byl použit PyCharm a interaktivní webové prostředí Jupyter. Veškeré výpočty probíhaly na vzdáleném linuxovém serveru s grafickou kartou Nvidia GeForce GTX 1070 s pamětí 8GB umístěném na UBMI.

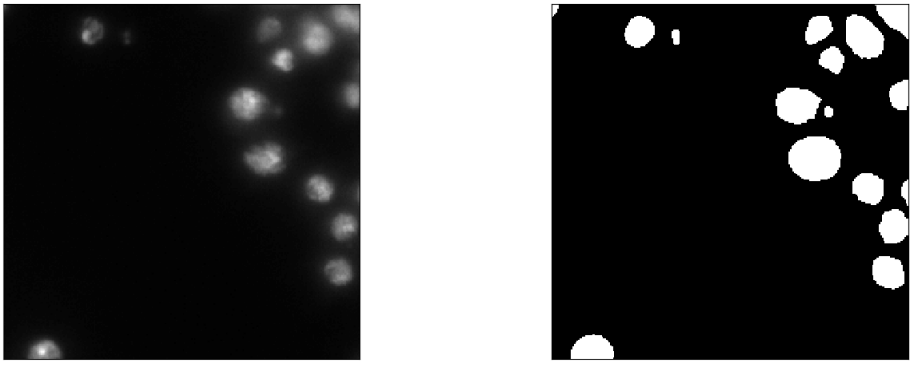
4.1 Dataset

Pro tuto práci byl použit dataset z Kaggle 2018 Data Science Bowl [6]. Dataset obsahuje velké množství obrázků, které byly získány za různých podmínek. Je zde několik typů buněk, které mají různé rozměry, zvětšení a také odlišnou zobrazovací modalitu (světlé pole vs. fluorescence), viz obrázek 4.1. Dataset je sestavený tak, aby prověřil schopnost použité sítě generalizovat své výstupy.

Dataset obsahuje 670 anotovaných obrázků (stage1-train), 65 testovacích (stage1-test), které jsou podobné trénovacím datům, a nakonec dalších 3019 testovacích obrázků (stage2-test), které jsou už velmi rozdílné a nacházejí se zde i jiné typy buňek. Trénovací část dat vždy obsahuje složku s ID v názvu, ve které je samotný obrázek a anotované masky, viz obrázek 4.2. Každá maska je stejně velká jako obrázek a obsahuje vždy jednu anotovanou buňku - tedy každý obrázek má několik masek v závislosti na počtu buněk. Masky se nepřekrývají, to znamená že žádný pixel nepatří do více masek.



Obr. 4.1: Ukázka variability obrázků z datasetu



(a) Vstupní obraz (b) Korespondující maska

Obr. 4.2: Část 4.2a zobrazuje vstupní obrázek a 4.2b zobrazuje jeho korespondující masky, ty byly pro přehlednost zkombinovány do jednoho obrazu

4.2 Implementace sítě Faster-RCNN

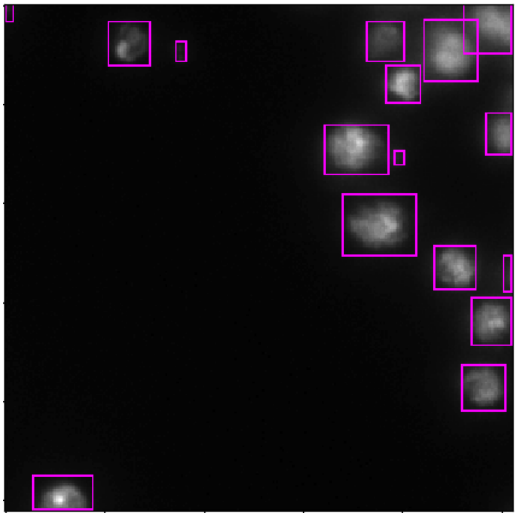
Návrh sítě Faster-RCNN se skládá z několika kroků. Prvním krokem je volba tzv. backbone. Backbone (páteř) je konvoluční neuronová síť, která z vstupního obrazu extrahuje příznaky. Takovýchto sítí existuje velké množství a každá má své výhody a nevýhody. Mezi nejpoužívanější patří různé variace VGG, ResNet, SqueezeNet nebo MobileNet. Je tedy nutné hledat nějaký kompromis mezi přesností, vypočetní náročností a rychlostí. Z porovnání jednotlivých sítí [2] jsem se rozhodl vybrat síť MobileNet-v2, jelikož je velmi rychlá a přesná.

Dalším krokem je nastavení RPN. Zde hraje velmi důležitou roli velikost anchors. V původní architektuře [24] používají anchors o velikostech $(32, 64, 128, 256, 512)$. Jelikož buňky jsou mnohem menší objekty je nutné tyto rozměry změnit. Rozměry buněk v použitém datasetu nepřesahují velikost velikost 50, rozhodl jsem se tedy použít anchors o velikostech $(4, 8, 16, 32, 64)$. Ostatní parametry jako počet generovaných návrhů nebo práh IoU pro odstranění redundantních návrhů jsem ponechal na původních hodnotách. Stejně tak nastavení Fast RCNN.

Faster RCNN se skládá ze dvou částí, kde každá má svou regresní a klasifikační část. Proto má každá čast svoji regresní a klasifikační chybu. Pro klasifikační chyby se používá křížová entropie a pro regresní chyby se používá L1 chybová funkce. Celková chyba je tedy dána součtem všech chyb dílčích částí.[24] Jednotlivé průběhy při učení můžeme vidět na obrázku 4.4.

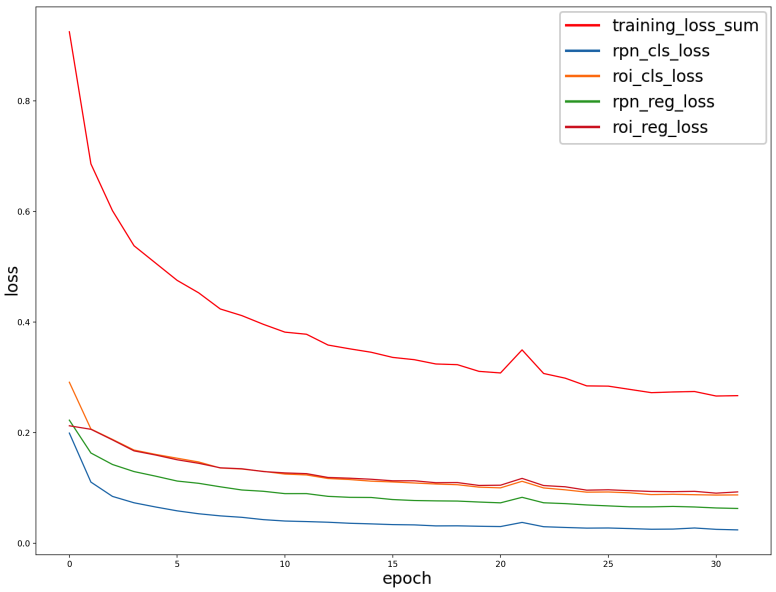
Faster RCNN predikuje přímo souřadnice ohraničujících rámečku. Aby bylo možné počítat jednotlivé chyby, musí být anotace ve stejném formátu. Jelikož každá

maska v datasetu obsahuje právě jednu buňku, bylo velmi snadné pomocí jednoduchých funkcí z knihovny NumPy [18] převést binární masku na ohraničující rámeček. Anotace tedy do sítě vstupovaly jako rámečky, které jsou definovány souřadnicemi levého horního a pravého dolního rohu rámečku. Každý rámeček má pak ještě zvlášť svoji třídu. Počet rozlišovaných tříd je 2. Faster RCNN totiž rozlišuje pozadí jako třídu, to znamená, že pro pozadí je to 0 a pro buňku 1. Obrázek s anotacemi můžeme vidět na obrázku 4.3.



Obr. 4.3: Ukázka vstupního obrázku a jeho ohraničujících rámečků

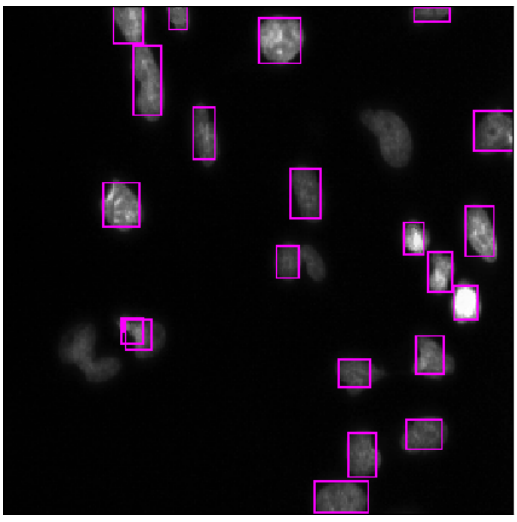
Trénování probíhalo pomocí optimalizačního algoritmu SGD s využitím momenta. Nejlepších výsledků bylo dosaženo při rychlosti učení $\eta = 0.005$. Pro trénování byl trénovací dataset (stage1-train) rozdělen na dvě části. Trénovací část obsahovala 600 obrázku a zbylých 70 obrázků bylo použito pro validaci. Každý obrázek byl před vstupem do sítě normalizován. Trénování bylo ukončeno po 30 epochách, jelikož celková chyba *training_loss_sum* začala vykazovat známky konvergence a zároveň jsem nechtěl způsobit, aby se síť přeučila.



Obr. 4.4: Průběh chybové funkce při tréninku.

Detekce

Díky komplexní architektuře Faster R-CNN není nutné na výsledné detekce aplikovat postproccessing. Všechny nežádoucí detekce jsou odstraněny uvnitř architektury pomocí algoritmu Non-maximum Supression, který je popsán v kapitole 4.3, kde se přímo používá.



Obr. 4.5: Ukázka detekce sítě Faster R-CNN

4.3 Implementace sítě YOLOv3

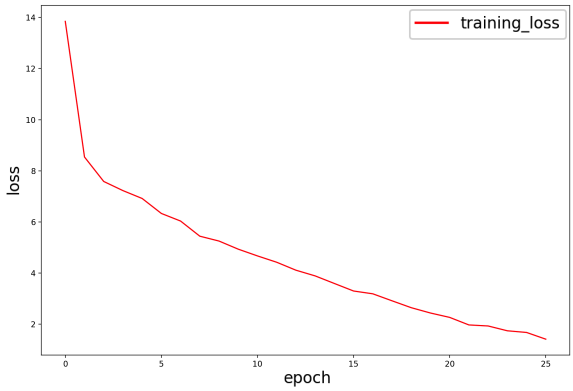
Návrh sítě YOLOv3 byl o něco jednodušší oproti Faster R-CNN, což je dáno už jen jednodušší konstrukcí. Veškerá konfigurace probíhá přes konfigurační soubor cfg, který byl vydán autorem pro snadné postavení sítě. Pro trénovaní na vlastním datasetu je zde nutné změnit 3 hlavní nastavení. Nejprve počet tříd, který je v mém případě 1, jelikož YOLO nerozeznává pozadí. Dalším krokem je nastavení počtu filtrů konvoluční vrstvy před každou detekční vrstvou. Tento počet je dán vzorcem $3 \times (počet\_tříd + 4 + 1)$, jelikož síť predikuje na 3 místech pro každou třídu 4 souřadnice a 1 $confidence$ skóre [23]. Počet filtrů je tedy 18. Posledním nastavením, podobně jako u Faster R-CNN, je nastavení anchors. To se děje pomocí k-means shlukovování.

K-means funguje zjednodušeně na následujícím principu. Nejprve je náhodně umístěn k počet centroidů. Následně se počítá vzdálenost každého objektu s každým centroidem a na základě toho se přiřadí objekt k nejbližšímu centroidu. Po přiřazení všech objektů se přepočítá poloha centroidu, tak aby centroid byl novým "průměrem (mean)"daného shluku, jedná se tedy o těžiště. Následně se proces opakuje dokud nedojde k ustálení. Výsledkem je k shluků, kde hodnoty každého centroidu reprezentují průměrné hodnoty objektů z jeho shluku.

Jako objekty jsou v našem případě pravdivé ohraničující rámečky, přesněji jejich šířka a výška. Počet anchors, které generuje každá buňka v mřížce je 9, proto počet shluků je k=9. Výsledkem je tedy 9 shluků s centroidy, kde každý reprezentuje výšku a šířku nejčastěji se vyskytujících velikostí buňek v datasetu. Tyto rozměry jsou ještě přepočítány na velikost obrázků, který vstupuje do sítě. YOLOv3 pracuje s obrázky o rozměru 412×412, tedy všechny vstupní obrázky byl převzorkovány na tuto velikost. Výsledné anchors mají tedy rozměry: $(6, 8), (12, 16), (22, 22), (22, 42), (32, 32), (32, 52), (46, 40), (62, 68), (116, 118)$

Dalším krokem je úprava anotací. Podobně jakou u Faster R-CNN je potřeba předělat masky na ohraničující rámečky. YOLO ale predikuje ve formátu x, y, w, h proto i anotace jdou do sítě v této formě. Jedná se tedy o stejné rámečky jako na obrázku 4.3, akorát jsou jinak definované. Dataset je použit stejný jako v předchozím případě.

Síť byla trénovaná pomocí algoritmu Adam s rychlosti učení $\eta = 0,001$. Průběh učení můžeme vidět na obrázku 4.6. Učení bylo zastaveno po 25 epoše, kvůli tomu že model začal vykazovat zvyšující se chybu na validačním datasetu, přestože celková trénovací chyba stále klesala.



Obr. 4.6: Průběh chybové funkce při tréninku.

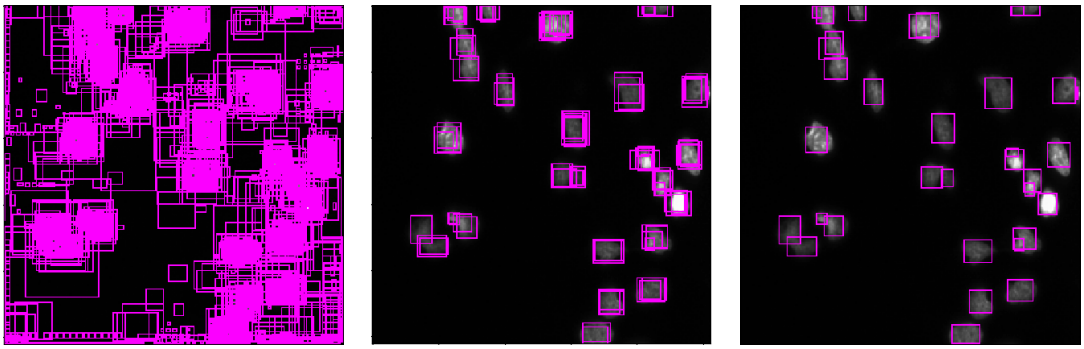
Detekce

Síť YOLOv3 na výstupu sítě generuje několik detekcí, které se ještě musí zpracovat, abychom je mohli považovat za pravdivé detekce. Nejprve je nutné odstranit všechny rámečky, jejichž $confidence$ skóre je pod určitý prah. Jako prah se obecně používá hodnota 0.5. Následně je nutné odstranit rámečky, které mají vysoké skóre, ale překrývají se. To se provádí pomocí algoritmu Non-maximum Suppression (NMS). Algoritmus vezme rámeček s nejvyšším $confidence$ skóre a přesune ho do výstupního seznamu rámečků. Následně se vypočítá IoU mezi tímto a všemi ostatními rámečky. Pokud je IoU nějakého rámečku větší než daný práh, je tento rámeček odstraněn. Jako práh, stejně jako pro $confidence$, se používá hodnota 0.5. Tento postup se opakuje dokud původní seznam rámečků není prázdný, tedy všechny rámečky byly přesunuty do výstupního seznamu nebo byly odstraněny. Odstranění nežádoucích detekcí je vidět na obrázku 4.7.

4.4 Implementace sítě U-Net

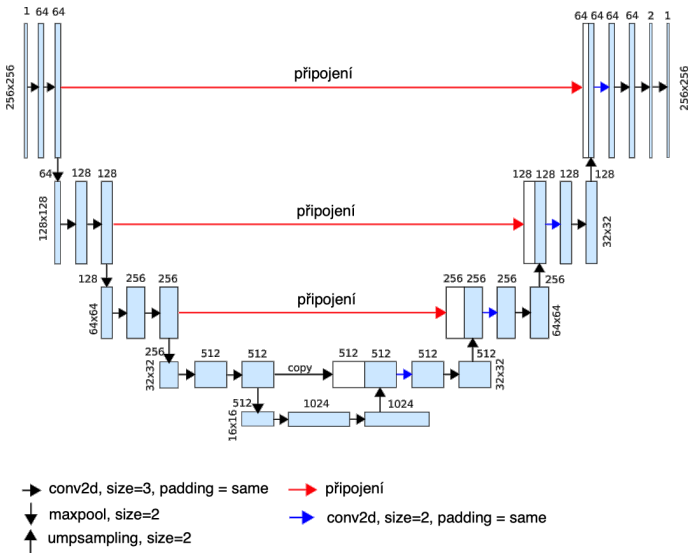
Implementace sítě U-Net byla se všech tří sítí nejjednodušší. V originální verzi sítě je velikost vstupního obrazu rozdílná s tím na výstupu. Proto jsem se rozhodl použít upravenou architekturu, u které je vstup a výstup rozměrově stejný. Zároveň zde očekává vstup o velikosti 256×256, což je pro můj dataset velmi výhodné. Architekturu můžeme vidět obrázku 4.8. Zachování stejných rozměrů je dosaženo pomocí zero paddingu 2.1 při konvoluci.

Následovala úprava anotací. U-Net má narozdíl od sítí Faster RCNN a YOLO na výstupu obraz. Proto i anotace je ve formě obrazu. Jednotlivé masky byly tedy



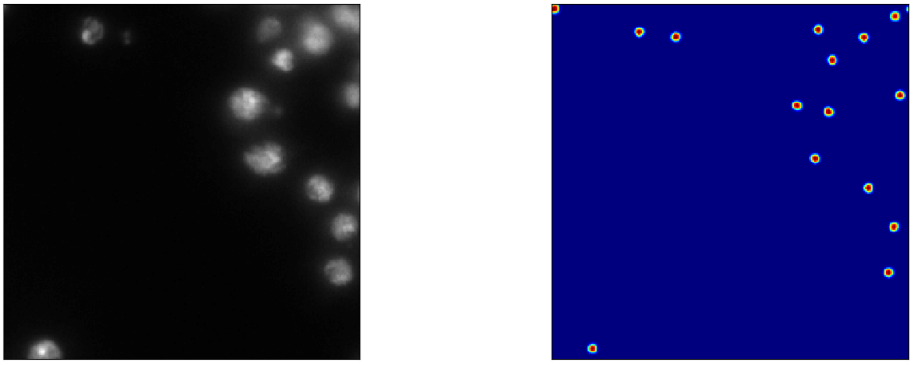
(a) $confidence > 1^{-5}$ (b) $confidence > 0.5$ (c) NMS, $IoU > 0.5$

Obr. 4.7: Prahování výstupů sítě YOLOv3. V části 4.7a je zobrazen výstup sítě, který je prahován s velmi nízkým prahem, pro představu kolik rámečku se úpravou odstraní, jelikož samotný výstup sítě obsahuje 10647 rámečků, z nichž většina má *confidence* skóre velmi blízké nule. Část 4.7b ukazuje skutečné prahování použité při detekci a část 4.7c pak výsledné detekce po uplatnění algoritmu Non-max supression.



Obr. 4.8: Použitá architektura sítě U-Net, převzato z [16]

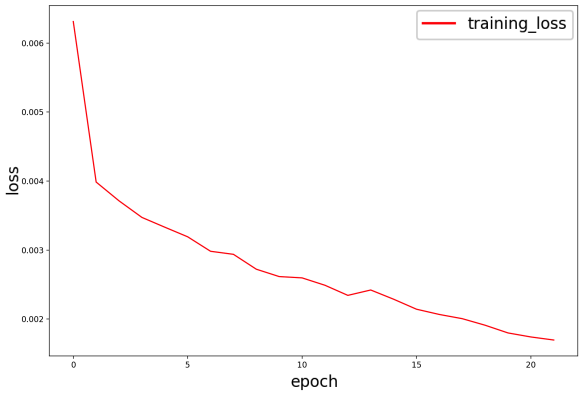
předělány tak, aby reprezentovaly střed buňky. Ukázku můžeme vidět na obrázku 4.9. U-Net tedy v tomto případě predikuje po vhodném postprocessingu pouze středy buněk a nikoli ohraničující rámeček. Dataset je použit stejný jako v předchozích případech.



(a) Vstupní obraz (b) Heat mapa

Obr. 4.9: Část 4.9a zobrazuje vstupní obrázek a 4.9b zobrazuje jeho korespondující heat mapu vstupující do sítě jako anotace

Trénování probíhalo pomocí optimalizačního algoritmu Adam s rychlostí učení $\eta = 0.001$. Jako chybová funkce byla použita střední kvadratická odchylka. Průběh učení v závislosti na chybové funkci můžeme vidět na Obr. 4.10. Trénování bylo ukončeno po 20. epoše.



Obr. 4.10: Průběh chybové funkce při tréninku.

Detekce

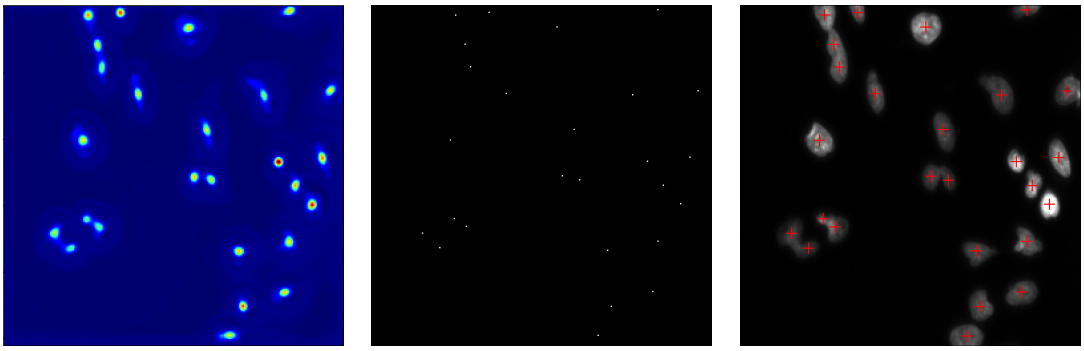
Ačkoliv návrh sítě byl z ostatních nejlehčí, zpracování výstupu sítě je zde nejsložitější. Jak už bylo několikrát zmiňováno, výstupem je obraz - heatmapa, jejíž nejvyšší hodnoty reprezentují střed detekované buňky. Je tedy nutné hledat v obrazu lokální maxima. Způsobů jak tyto maxima hledat je několik. Obraz je možné prahovat,

hledat maxima v určitém okolí nebo lze použít různé techniky matematické morfologie. Rozhodl jsem se jít sofistikovanější cestou a zvolit techniku zvanou H-maxima transformace.

H-maxima transformace je morfologická operace používaná pro filtrování lokálního maxima v obrazech. Lokální maximum je definováno jako spojení pixelů v daném okolí, jejichž jasová hodnota je větší než hodnota okolních pixelů. H-maxima trasformace potlačuje všechna maxima, jejichž jasové hodnoty jsou pod daný práh h. To se děje pomocí morfologické dilatace z f na $f - h$ popsané rovnicí:

$$HMAX_h\left(f\right) = R_f^\delta\left(f - h\right) \tag{4.1}$$

kde f je jasová hodnota obrázku a h je práh. Ukázku můžeme vidět na obrázku 4.11

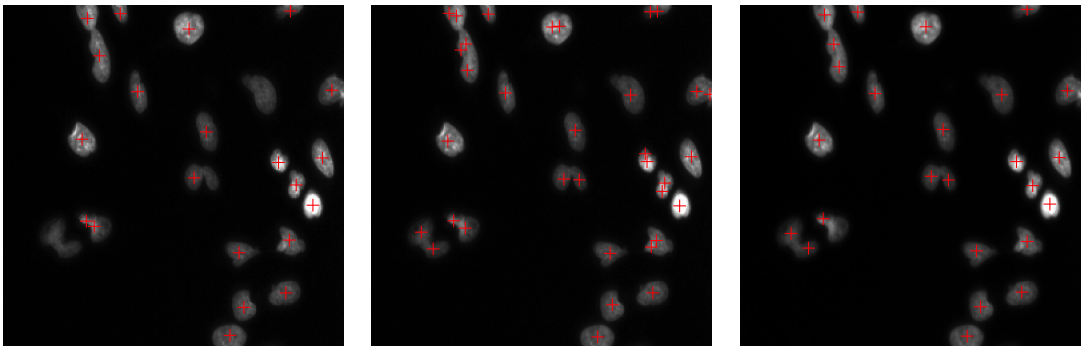


(a) výstup sítě (b) h-maxima, h=0.3 (c) výsledná detekce

Obr. 4.11: Zpracování výstupů sítě U-Net. V části 4.11a je zobrazen přímý výstup sítě. Část 4.11b zobrazuje aplikaci h-maxima transformace na výstup sítě. Po aplikaci této transformace zbyly na obraze pouze jednotlivé pixely, které přímo reprezentují střed maxima daného okolí. Část 4.11c pak zobrazuje detekce z h-maxima transformace na vstupním obrazu.

5 Výsledky a hodnocení

Aby bylo možné nějak objektivně porovnat všechny použité sítě, bylo nejprve nutné najít metriku, která se dá použít pro všechny. Obecně se detekce objektů hodnotí pomocí překryvu ohraničujících rámečků pomocí IoU. Síť U-Net však v tomto případě dokáže detekovat pouze středy buněk. Proto je nutné převést výstupy ostatních sítí, tak aby reprezentovaly také střed buňky. Pokud si vzpomeneme na kapitolu 3.5, tak síť YOLOv3 detekuje rámečky definované pomocí souřadnic středu rámečku a jeho šířky a výšky. Převést je tedy potřeba pouze rámečky sítě Faster R-CNN, které jsou definovány souřadnicemi levého horního a pravého dolního rohu rámečku.



(a) Faster R-CNN (b) YOLOv3 (c) U-Net

Obr. 5.1: Ukázka detekcí jednotlivých sítí jako souřadnice středu buňky

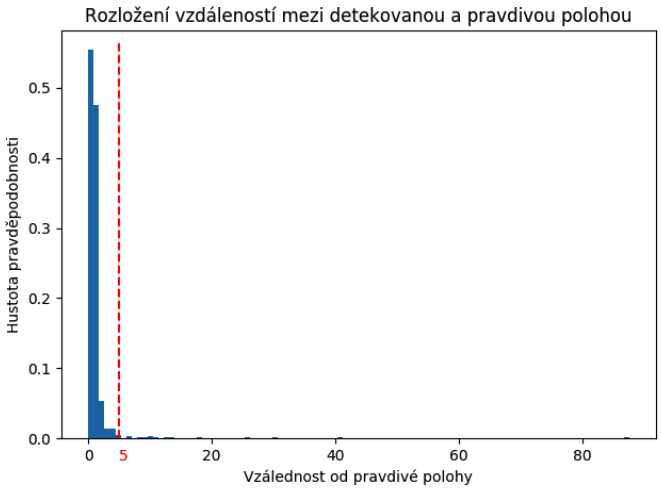
Stanovení pravdivé detekce

Za pravdivou detekci byla považovaná detekce, jejíž vzdálenost od pravdivé polohy buňky byla pod určitý práh. Pro výpočet vzdálenosti byla použita Euklidovská vzdálenost. Jak můžeme vidět z hustoty rozložení vzdáleností na obrázku 5.2, naprostá většina má vzdálenosti do hodnoty 5. Proto i práh byl zvolen na této hodnotě. Tento práh bych označil za optimální, když vezmeme v úvahu velikost obrázku $256x256$.

Jednotlivé hodnoty pravdivé pozitivity (TP), falešné pozitivity (FP) a falešné negativity (FN) pro každý obrázek byly dány následovně:

$$TP = vzdálenost < práh$$
$$FP = vzdálenost > práh$$
$$FN = počet\_buňek - TP$$

kde *počet_buněk* představuje počet anotací k danému obrázku.



Obr. 5.2: Rozložení vzdáleností mezi detekovanou a pravdivou polohou buňky, červeně je vyznačený práh. Jednotlivé hodnoty představují vzdálenosti ze všech obrázků validačního datasetu a všech použitých sítí.

Z těchto hodnot byly vypočítány tři základní statistické údaje - precision, recall a dice. Precision neboli pozitivní prediktivní hodnota (PPV), určuje pravděpodobnost výskytu pravdivě pozitivní detekce vůči všem detekcím (pravdivým i falešným), a je dána následujícím vzorcem:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

Recall neboli senzitivita (TPR), určuje pravděpodobnost pravdivě pozitivní detekce vůči všem buňkám (pravdivě pozitivní + falešně negativní), a je dána následujícím vzorcem:

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

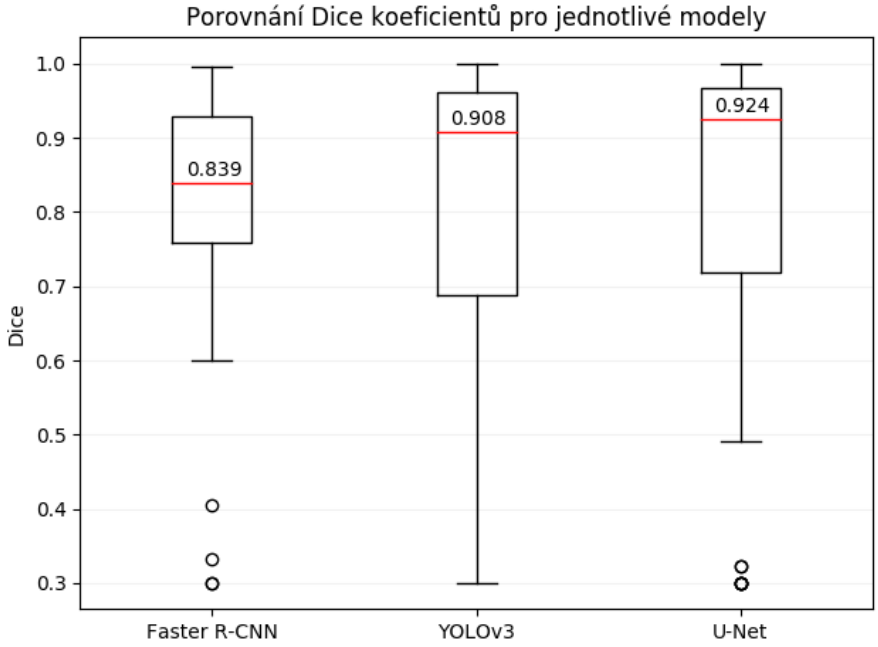
Dice je koeficient podobnosti. Jedná se o harmonický průměr výše zmíněných Precision a Recall. Koeficient udává míru shody dvou souborů údajů: anotace a detekce, a je dán následujícím vzorcem:

$$Dice = \frac{2TP}{2TP + FP + FN} \tag{5.3}$$

Všechny tyto údaje byl vypočítány na základě obrázků a jejich anotací z validačního datasetu, který vznikl, jak bylo popsáno v kapitole 4.2, rozdělením datasetu $stage1\_train$ na 600 trénovacích a 70 validačních. Průměrné výsledky jednotlivých koeficientů pro každou síť můžeme vidět v tabulce 5.1. Dice byl pak speciálně vynesen do krabicového grafu 5.3.

Tab. 5.1: Průměrné hodnoty Precision, Recall a Dice koeficientů pro jednotlivé sítě.

	Precision	Recall	Dice
Faster R-CNN	0,919	0,896	0,865
YOLOv3	0,823	**0,965**	0,882
U-Net	**0,951**	0,853	**0,899**



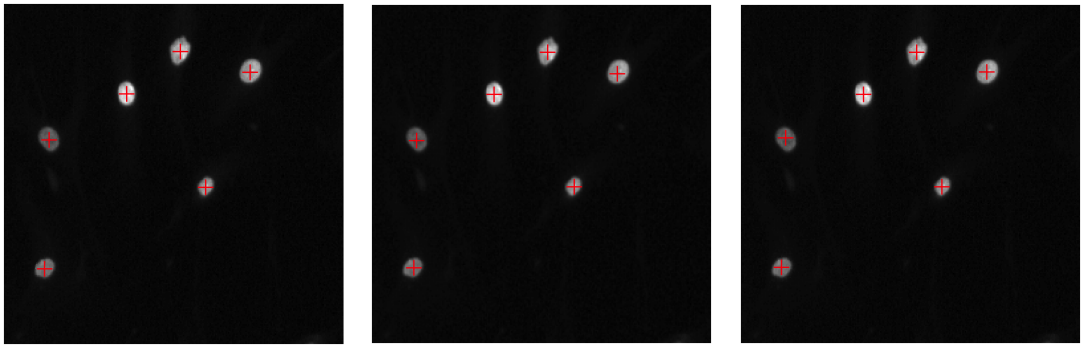
Obr. 5.3: Rozdělění Dice koeficientů pro jednotlivé sítě. Hodnoty v grafu reprezentují Dice koeficienty pro každý obrázek validačního datasetu. Červeně je pak vyznačen medián a jeho hodnota.

5.1 Diskuse dosažených výsledků

Jak můžeme vidět z tabulky 5.1, nejlepších výsledků dosáhla síť U-Net, která měla nejvyšší precision i dice ale naopak nejhorší recall. To znamená že síť predikovala velmi málo falešně pozitivních výsledků ale ne vždy se jí podařilo detekovat všechny buňky. Síť YOLOv3 dosáhla naopak nejlepšího výsledku recall ale nejhoršího precision. Často se tedy stávalo, že YOLOv3 predikovala hodně falešně pozitivních výsledků. Síť Faster R-CNN má hodnoty precision a recall nejvíce vyrovnané, celkově má ale dice nejnižší.

Pokud se podíváme na krabicový graf dice koeficientů 5.3, můžeme vidět rozdělní

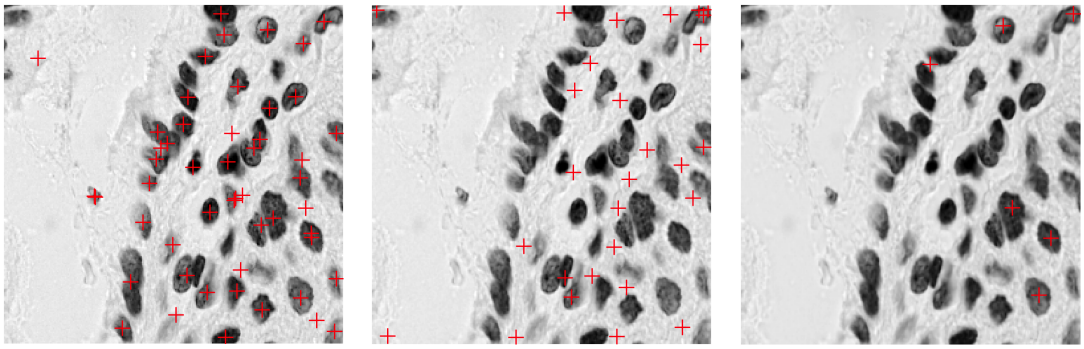
jednotlivých dice koeficientů pro jednotlivé obrázky validačního datasetu. Faster R-CNN má toto rozdělní nejvíce kolem mediánové hodnoty a má nejmenší rozptyl, obsahuje ale více odlehlých hodnot. YOLOv3 má největší rozptyl což je dáno především predikcí většího množství falešné pozitivních výsledků a tedy největšího rozdílu mezi precision a recall hodnotami. Medián má však vyšší než Faster R-CNN. Nejvyšší hodnotu mediánu podobně jako v tabulce dosáhla síť U-Net. Medián je zde dokonce 0.924, což značí že průměrné hodnoty jsou zatížené odlehlými hodnotami.



(a) Faster R-CNN (b) YOLOv3 (c) U-Net

Obr. 5.4: Ukázka detekcí jednotlivých sítí na snímek pořízený pomocí fluorescenčního mikroskopu.

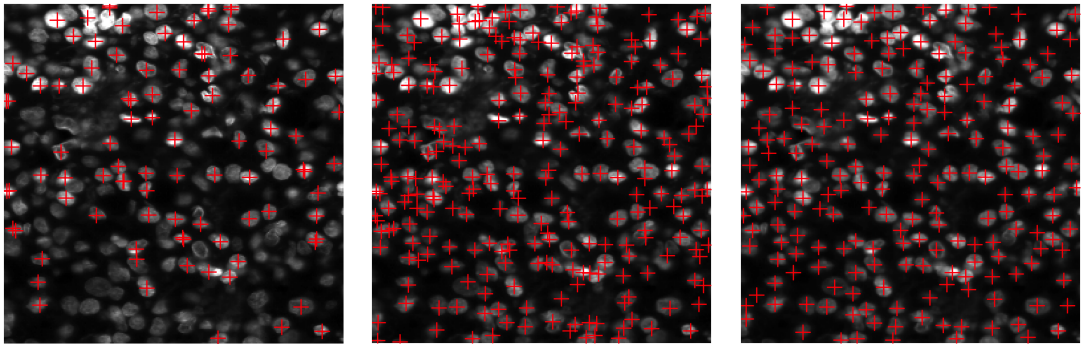
Nyní se ale podíváme jak jednotlivé sítě dokázali detekovat buňky přímo na obrázcích. Na obrázku 5.4 můžeme vidět snímek z fluorescenčního mikroskopu, kterých je v celém datasetu většina. Buňky jsou zde ostré, nejsou příliš u sebe a mají velmi jasně definované hrany. Všechny tři sítě neměli, s podobným typem obrázku a rozložením buněk, žádný problém a vždy detekovali všechny buňky na obrázku.



(a) Faster R-CNN (b) YOLOv3 (c) U-Net

Obr. 5.5: Ukázka detekcí jednotlivých sítí na snímek pořízený pomocí světelného mikroskopu založeném na pozorování ve světlém poli.

Jinou situaci můžeme vidět na obrázku 5.5 pořízeného pomocí světelného mikroskopu, jednotlivé sítě se ve výsledcích velmi liší. Obstojně si vedla akorát síť Faster R-CNN, která dokázala rozeznat buňku i na světlém pozadí. Síť YOLOv3 zde neuspěla a detekovala pouze velmi mnoho falešně pozitivních výsledků. U-Net na tom není o moc lépe, sice zde není takový počet falešně pozitivních výsledků ale je zde také velké množství falešně negativních. Je tedy vidět že na tento typ snímků se naučila pouze síť Faster R-CNN.

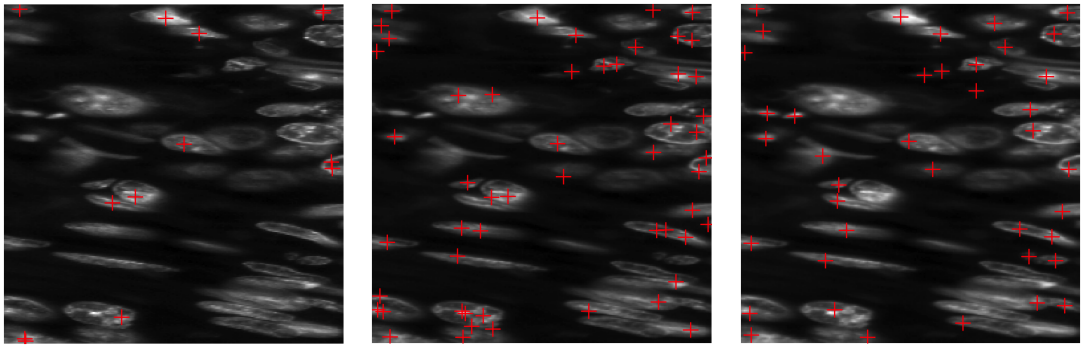


(a) Faster R-CNN (b) YOLOv3 (c) U-Net

Obr. 5.6: Ukázka detekcí jednotlivých sítí.

Naopak Faster R-CNN má oproti YOLOv3 a U-Net problémy s detekcí buněk, které jsou velmi blízko u sebe. Ukázku můžeme vidět na obrázku 5.6. Jednotlivé buňky navíc nejsou moc dobře zobrazeny a nemají tak ostré hranice. YOLOv3 je zde mnohem lepší, avšak někdy se stane, že stejnou buňku detekuje vícekrát. Překryv těchto detekcí však není dostatečný a tak nebyl odstraněn pomocí Non-max supression. U-Net zde velmi jasně podává nejlepší výsledek s minimem falešně negativních výsledků.

Všechny sítě pak mají problém s detekcí příliš velkých a „dlouhých“ buněk. To můžeme vidět na obrázku 5.7. Buňky jsou zde detekovány velmi nepřesně a někdy dokonce i víckrát. Obzvláště Faster R-CNN má s velkými buňkami, které mají nestandartní tvar, velký problém, to je dáno především rozměry použitých anchors.



(a) Faster R-CNN (b) YOLOv3 (c) U-Net

Obr. 5.7: Ukázka detekcí jednotlivých sítí.

Pokud ale vezmeme v úvahu jak vysoká variabilita použitého dataset byla a podíváme se na další detekce v přílohách, tak všechny sítě dosahují úctyhodných výsledků. Pokud bych však teď, v době psaní této práce, začínal pracovat od znovu, jsou nejméně dvě věcí, které bych udělal jinak. Osobě jsem totiž očekával, že dnešní „state of art“ detekční systémy v podobě Faster R-CNN a YOLOv3 zdaleka překonají segmentační metodu U-Net, která však byla speciálně vytvořená pro biomedicínské aplikace. Proto bych se více zaměřil na trénování jednotlivých sítí, které by samotné mohlo být předmětem několika dalších bakalářských prací. Strojové učení je totiž velmi široká oblast, kde je několik způsobů jak daný problém řešit. Druhá věc je volba použitého datasetu. Tím že je velmi různorodý tak každý typ buněk obsahuje různý počet trénovacích snímků, a proto bych zkusil natrénovat sítě na jiných datech a nebo využil různé metody augumentace.

Závěr

Tématem této práce byla detekce buněk pomocí konvolučních neuronových sítí. Cílem bylo seznámení se s metodami detekce objektů a jejich použití a implementace pro detekci buněk. Nejprve byla tedy vytvořena literární rešerše zabývající se základními neuronovými sítěmi a jejich principy učení a optimalizace. Dále byly rozebrány jednotlivé architektury konvolučních neuronových sítí a popis jejich principu fungování. Z těchto znalostí byly následně tyto architektury implmentovány v jazyce Python s využitím knihovny PyTorch. Byly zde využity 3 různé přístupy k detekci. První, Faster R-CNN, využívá k detekci objektů návrhy regionů, ze kterých se následně provádí klasifikace a regrese ohraničujících rámečků. Druhý, YOLOv3, bere detekci objektu jako jednoduchý regresní problém a učí se získat rovnou z vstupního obrazu pravděpodobnosti třídy a souřadnice ohraničujících rámečků. Posledním typem je síť U-Net, která využívá metod segmentace k vytvoření heatmapy, ze které se získávají souřadnice pomocí hledání lokálního maxima.

Jednotlivé sítě se pak v poslední části statisticky vyhodnocují. Hodnocení probíhalo na základě měření vzdálenosti detekované polohy od její polohy pravdivé. Z těchto dat byly vypočítány tři základní statistické koeficienty precison, recall a dice. Nejvyšší hodnoty dice koeficientu dosáhla síť U-Net a to 0,899, dále pak YOLOv3 s hodnotou 0,882 a nakonec síť Faster R-CNN s hodnotou 0,865. Tyto výsledky a jednotlivé detekce přímo na obrázcích jsou v závěru práce diskutovány i s možnostmi vylepšení, které by mohly vést k lepším výsledkům.

Literatura

[1] Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/, citovano dne 24.12.2019.

[2] Bianco, S.; Cadène, R.; Celona, L.; aj.: Benchmark Analysis of Representative Deep Neural Network Architectures. *CoRR*, ročník abs/1810.00736, 2018, 1810.00736.
URL http://arxiv.org/abs/1810.00736

[3] Dubost, F.; Bortsova, G.; Adams, H.; aj.: GP-Unet: Lesion Detection from Weak Labels with a 3D Regression Network. 05 2017, doi:10.1007/978-3-319-66179-7_25.

[4] Girshick, R. B.: Fast R-CNN. *CoRR*, ročník abs/1504.08083, 2015, 1504.08083.
URL http://arxiv.org/abs/1504.08083

[5] Girshick, R. B.; Donahue, J.; Darrell, T.; aj.: Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, ročník abs/1311.2524, 2013, 1311.2524.
URL http://arxiv.org/abs/1311.2524

[6] Hamilton, B. A.: 2018 Data Science Bowl: Find the nuclei in divergent images to advance medical discovery. 2018.
URL https://www.kaggle.com/c/data-science-bowl-2018

[7] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *CoRR*, ročník abs/1512.03385, 2015, 1512.03385.
URL http://arxiv.org/abs/1512.03385

[8] Höfener, H.; Homeyer, A.; Weiss, N.; aj.: Deep learning nuclei detection: A simple approach can deliver state-of-the-art results. *Computerized Medical Imaging and Graphics*, ročník 70, 2018: s. 43 – 52, ISSN 0895-6111.
URL http://www.sciencedirect.com/science/article/pii/S0895611118300806

[9] Jan, J.: *Medical image processing, reconstruction, and restoration*. Boca Raton, FL: Taylor & Francis, 2006, ISBN 978-0-8247-5849-3.

[10] Ju; Yun, L.; Wang; aj.: The Application of Improved YOLO V3 in Multi-Scale Target Detection. *Applied Sciences*, ročník 9, 09 2019: str. 3775, doi:10.3390/app9183775.

[11] Kathuria, A.: Whats new in YOLO v3? 2019, citovano dne 20.5.2020.
URL https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[12] Kingma, D. P.; Ba, J.: Adam: A Method for Stochastic Optimization. 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
URL http://arxiv.org/abs/1412.6980

[13] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: Imagenet classification with deep convolutional neural networks. 2012.

[14] Lin, T.; Dollár, P.; Girshick, R. B.; aj.: Feature Pyramid Networks for Object Detection. *CoRR*, ročník abs/1612.03144, 2016, 1612.03144.
URL http://arxiv.org/abs/1612.03144

[15] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Elements of Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1996, ISBN 0262133288.

[16] Melnikov, S.: Semantic segmentation to detect Nuclei using U-net. 2019, citovano dne 20.12.2019.
URL https://medium.com/analytics-vidhya/semantic-segmentation-using-u-net

[17] Nielsen, M. A.: *Neural Networks and Deep Learning*. Determination Press, 2015.

[18] Oliphant, T.: A guide to NumPy. USA: Trelgol Publishing, 2006.
URL http://www.numpy.org/

[19] O'Shea, K.; Nash, R.: An Introduction to Convolutional Neural Networks. *ArXiv e-prints*, 11 2015.

[20] Paszke, A.; Gross, S.; Chintala, S.; aj.: Automatic differentiation in PyTorch. 2017.
URL https://pytorch.org

[21] Redmon, J.; Divvala, S. K.; Girshick, R. B.; aj.: You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, 2015, 1506.02640.
URL http://arxiv.org/abs/1506.02640

[22] Redmon, J.; Farhadi, A.: YOLO9000: Better, Faster, Stronger. *CoRR*, ročník abs/1612.08242, 2016, 1612.08242.
URL http://arxiv.org/abs/1612.08242

[23] Redmon, J.; Farhadi, A.: YOLOv3: An Incremental Improvement. *CoRR*, ročník abs/1804.02767, 2018, 1804.02767.
URL http://arxiv.org/abs/1804.02767

[24] Ren, S.; He, K.; Girshick, R. B.; aj.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, ročník abs/1506.01497, 2015, 1506.01497.
URL http://arxiv.org/abs/1506.01497

[25] Ronneberger, O.; Fischer, P.; Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR*, ročník abs/1505.04597, 2015, 1505.04597.
URL http://arxiv.org/abs/1505.04597

[26] Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, ročník 65, č. 6, 1958: str. 386.

[27] Ruder, S.: An overview of gradient descent optimization algorithms. *CoRR*, ročník abs/1609.04747, 2016, 1609.04747.
URL http://arxiv.org/abs/1609.04747

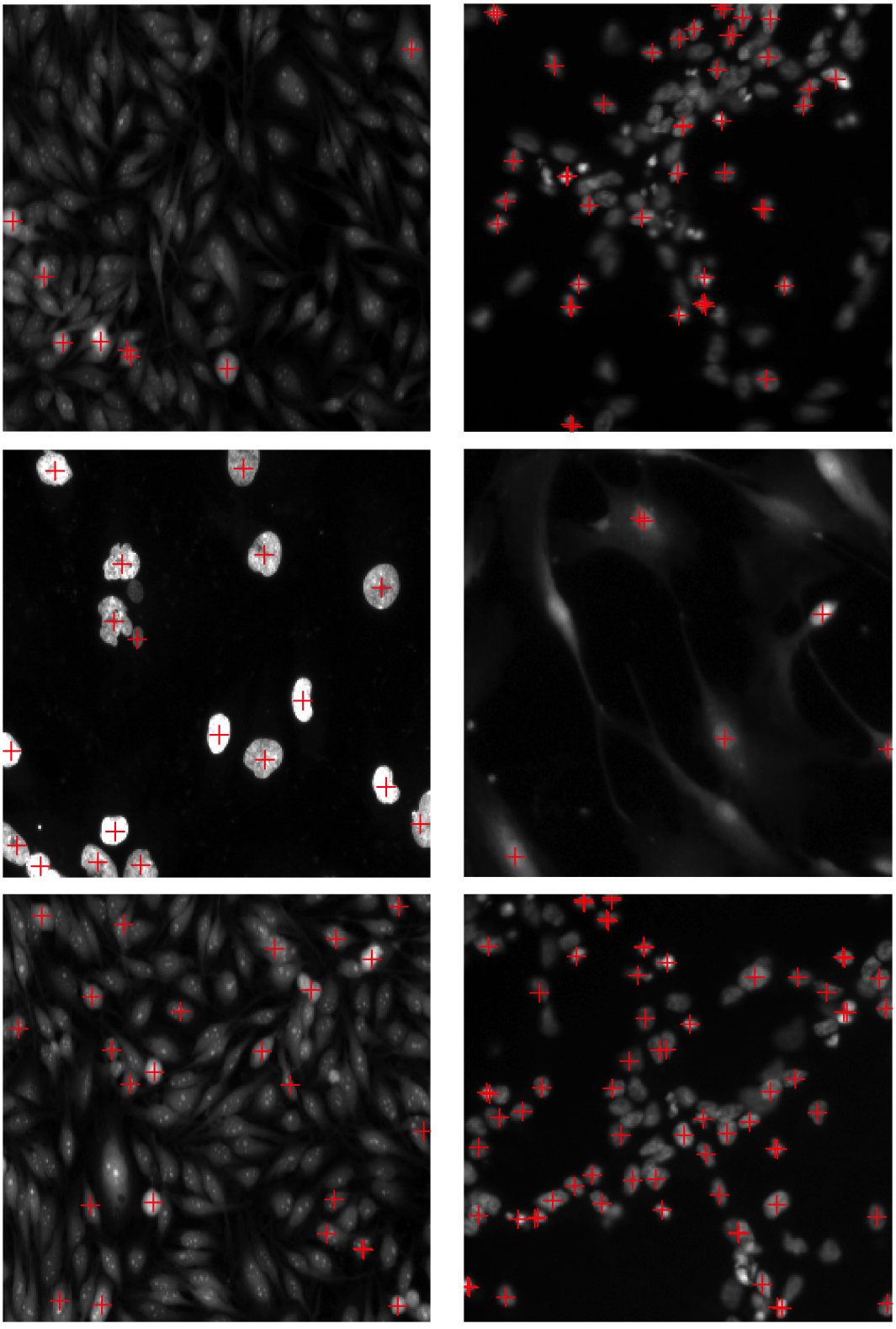
[28] Shelhamer, E.; Long, J.; Darrell, T.: Fully Convolutional Networks for Semantic Segmentation. *CoRR*, ročník abs/1605.06211, 2016, 1605.06211.
URL http://arxiv.org/abs/1605.06211

[29] Szegedy, C.; Liu, W.; Jia, Y.; aj.: Going Deeper with Convolutions. *CoRR*, 2014, 1409.4842.
URL http://arxiv.org/abs/1409.4842

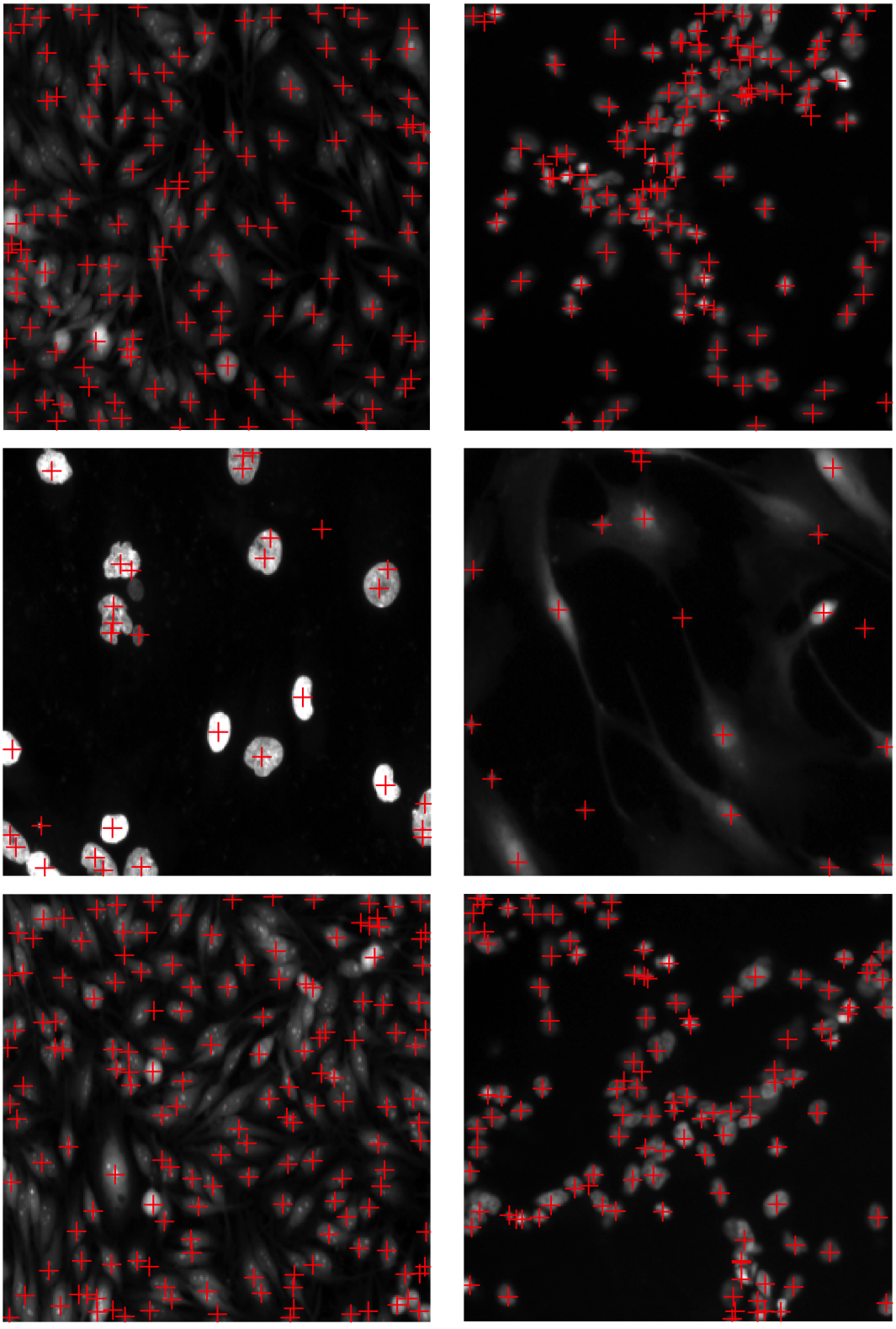
[30] Uijlings, J. R.; Van De Sande, K. E.; Gevers, T.; aj.: Selective search for object recognition. *International journal of computer vision*, ročník 104, č. 2, 2013: s. 154–171.

Seznam příloh

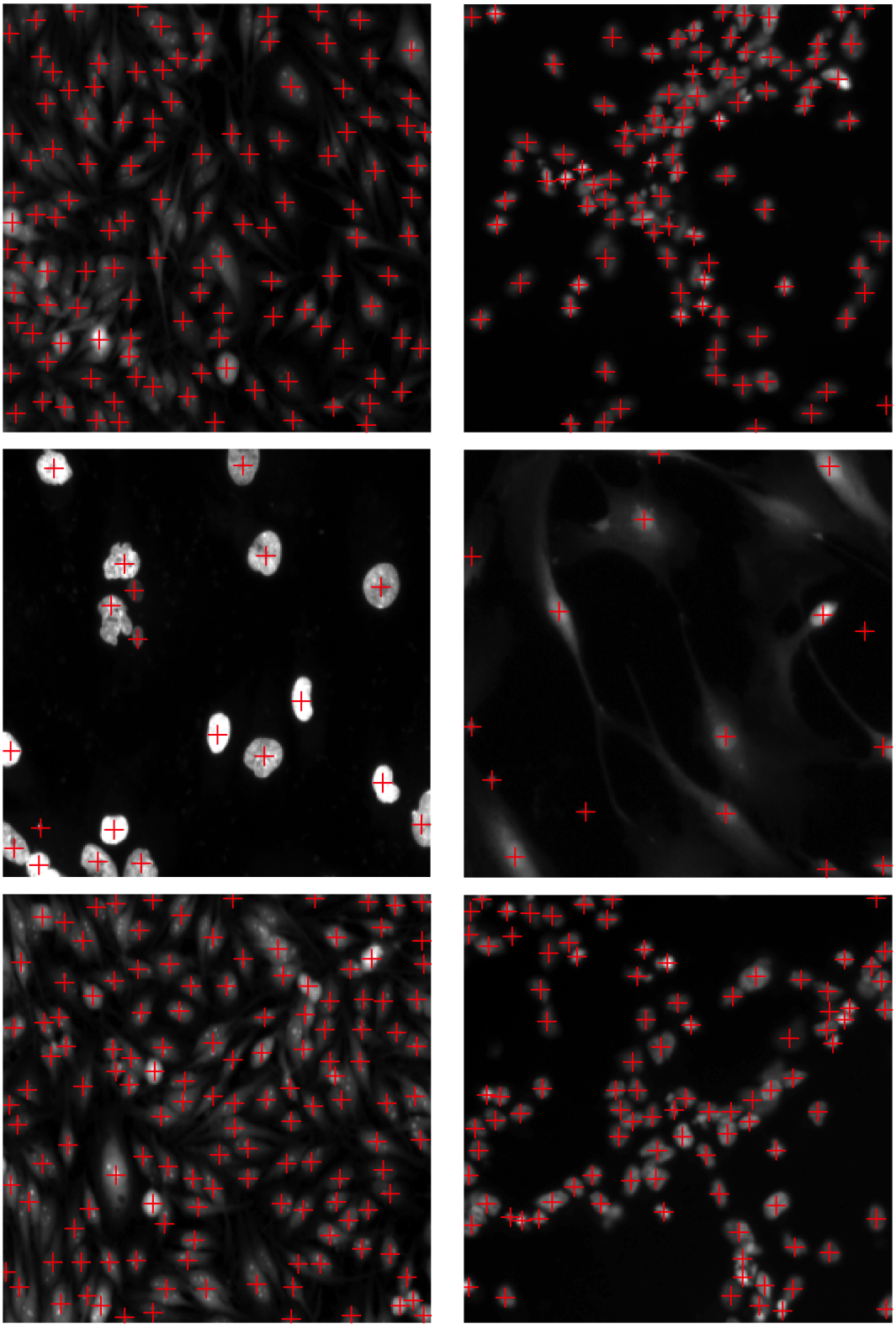
A Ukázky detekce sítě Faster R-CNN



B Ukázky detekce sítě YOLOv3



C Ukázky detekce sítě U-Net



D Obsah přiložených souborů

- `/cnn-cells` — kořenový adresář přiloženého CD
  - `conf` — konfigurační složka
    - `settings.py`
  - `data_utils` — složka funkcemi pro práci s daty
    - `dataset.py` — skript s sloužící k nahrávání dat do sítí
    - `transforms.py` — skript s sloužící k úpravě vstupních dat
  - `demo_images` — složka s demo obrázky
    - `0a5a390c955100df18bc11ba62721756487573a47ea5a5ea3f3ebc01d9789794`
    - `0c0cbeb6b808ecfda8b275f7053da56bdaf077a1dd5880877a91a13d673916bc`
    - `0fcf47e7a3c608678acfa7ebcd740b61e0b81c5382ce25d7200b91d89f5dc191`
    - `3d8260f94e55af7d298e8aab8bdb583c95b13bd633272acadd1fc1f9495fb8c6`
    - `3ec76685ce8ae25f3cbea4ea8e4d3a195fdb31d31d2937a7d0d99e2de5c58d8d`
  - `faster_rcnn` — implementace sítě Faster R-CNN
    - `eval.py` — skript sloužící k vyhodnocení sítě
    - `models.py` — skript s definicí architektury sítě
    - `predict.py` — skript s sloužící k detekci buněk sítě
    - `train.py` — skript s sloužící k trénování buněk sítě
  - `unet` — implementace sítě U-Net
    - `eval.py` — skript sloužící k vyhodnocení sítě
    - `models.py` — skript s definicí architektury sítě
    - `predict.py` — skript s sloužící k detekci buněk sítě
    - `train.py` — skript s sloužící k trénování buněk sítě
    - `unet_parts.py` — skript s definicí jednotlivých částí architektury
  - `utils` — složka s pomocnými funkcemi
    - `utils.py`
  - `yolo_v3` — implementace sítě YOLOv3
    - `config` — složka s konfiguracemi
      - `yolov3.cfg`
      - `yolov3-custom.cfg`
      - `yolov3-tiny.cfg`
    - `weights` — složka se skriptem pro stažení vah
      - `download_weights.sh`
    - `anchors_kmeans.py` — skript sloužící k navrhnutí rozměru anchors
    - `eval.py` — skript sloužící k vyhodnocení sítě
    - `models.py` — skript s definicí architektury sítě
    - `predict.py` — skript s sloužící k detekci buněk sítě
    - `train.py` — skript s sloužící k trénování buněk sítě
  - `demo.py` — demo skript pro ověření funkčnosti
  - `eval.py` — skript sloužící k vyhodnocení všech sítí zároveň
  - `predict.py` — skript s sloužící k detekci buněk všech sítí zároveň
  - `requirements.txt` — seznam závislostí
  - `requirements_cuda.txt` — zjednodušený seznam závislostí pro spuštění na CUDA
  - `requirements_tiny.txt` — zjednodušený seznam závislostí
- `readme.txt` — návod na instalaci a spuštění