TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Neuronové sítě pro automatickou detekci log v obraze

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Zbyněk Novák**
Vedoucí práce: Ing. Karel Paleček Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2018/2019**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení:	**Zbyněk Novák**
Název práce:	**Neuronové sítě pro automatickou detekci log v obraze**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**
Vedoucí práce:	**Ing. Karel Paleček Ph.D.**
Rozsah práce:	**30—40 stran**
Konzultant:	**Ing. Lukáš Matějů**

Zásady pro vypracování:

1. Seznamte se s problematikou neuronových sítí a hlubokého učení.
2. Vytvořte rešerži v oblasti aplikace neuronových sítí pro detekci objektů a log v obraze.
3. Otestujte a vyhodnoťte vybrané modely na více testovacích databázích.

Seznam odborné literatury:

[1] Goodfellow, I., Bengio, Y., Courville, A. Deep learning. MIT Press, 2016. ISBN: 978-0262035613

[2] Bishop, C. Pattern Recognition and Machine Learning. 2006. ISBN 13: 978-038731073

[3] Karpathy, A., Johnson, J., Li, F. Convolutional neural neworks for visual recognition, dostupné online: https://cs231n.stanford.edu

V Liberci dne .....

.....
Ing. Karel Paleček Ph.D.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

29. 4. 2019 Zbyněk Novák

Poděkování

Rád bych poděkoval Ing. Karlu Palečkovi, Ph.D. za vedení mé práce a také za pomoc při získávání potřebných informací a podkladů.

Abstrakt

Tato práce se zabývá problematikou hlubokého učení a neuronových sítí v rámci detekce log v obraze. Cílem je vytvoření rešerše v oblasti aplikace neuronových sítí pro detekci objektů a log v obraze a otestování vybraných modelů pro detekci log v obraze na vybraných testovacích databázích.

V rešeršní části je vysvětlen pojem hluboké učení a uvedeny příklady jeho konkrétního využití v praxi v rámci detekce objektů v obraze. Jsou popsány neuronové sítě, architektura dopředné neuronové sítě a typ učení s učitelem, které se využívají pro detekci objektů v obraze. Dále je popsána klasifikace objektů pomocí konvolučních neuronových sítí. Jsou představeny stávající systémy, používané pro detekci log v obraze, a trénovací databáze log.

Pro otestování byly vybrány dva modely, a to YOLOv3, implementovaný pomocí frameworku PyTorch, a Faster R-CNN, implementovaný pomocí frameworku Tensorflow. V rešeršní části jsou tyto modely a použité frameworky popsány a navzájem porovnány. Pro testování byly vybrány dva datasety, a to dataset FlickrLogos-32 a dataset TopLogo-10.

Detektor YOLOv3 byl testován ve dvou variantách – ve verzi YOLOv3-tiny, která je rychlejší a méně výpočetně náročná, a v plnohodnotné verzi YOLOv3. Tento detektor dosáhl ve verzi YOLOv3 nejvyšší přesnosti 45 % v čase 22 hodin na datasetu FlickrLogos-32 a přesnosti 59 % v čase 11 hodin na datasetu TopLogo-10. Verze YOLOv3-tiny byla čtyřikrát rychlejší, ale oproti plnohodnotné verzi měla na obou datasetech třetinovou přesnost.

Detektor Faster R-CNN dosáhl nejvyšší přesnosti 60 % na datasetu FlickrLogos-32 a 67 % na datasetu TopLogo-10. V obou případech trvalo trénování 7 hodin.

Z výsledků testování vyplývá, že ačkoli měl být detektor YOLOv3 rychlejší než detektor Faster R-CNN a měl dosahovat obdobné přesnosti, byl pomalejší a dosahoval menších přesností na obou datasetech. To mohlo být způsobeno im-

plementací detektoru YOLOv3, která obsahovala implementační chyby. Detektor Faster R-CNN je tak v tomto případě lepší volbou pro detekci log v obraze.

Klíčová slova:

COCO, Detekce objektů, detekce log, Faster R-CNN, FlickrLogos-32, hluboké učení, konvoluční neuronové sítě, neuronové sítě, openCV, Python, PyTorch, TopLogo-10, Tensorflow, YOLOv3,

Abstract

This thesis deals with the topic of deep learning and neural networks. The aim is to do research in the field of application of neural networks for object and logo detection and to test the chosen models for logo detection on chosen databases.

In the research part, the concept of deep learning is explained and examples of its practical use in object detection are given. Neural networks are described/explained, as well as Feed-Forward architecture and supervised learning, which are used for object detection. Next, object classification using convolutional neural networks is described. Existing systems, used for logo detection, and logo training databases are presented.

Two models were selected for testing , namely YOLOv3, implemented with PyTorch framework, and Faster R-CNN, implemented with Tensorflow framework. In the research part these models and used frameworks are described and compared. Two datasets, the FlickrLogos-32 dataset and the TopLogo-10 dataset, were selected for testing.

The YOLOv3 detector was tested in two versions - the YOLOv3-tiny version, which is faster and less computationally demanding, and in the full-featured version of YOLOv3. The YOLOv3 detector achieved the highest accuracy of 45 % at 22 hours on the FlickrLogos-32 dataset and accuracy of 59 % at 11 hours on the TopLogo-10 dataset. The YOLOv3-tiny version was four times faster, but compa-

red to the full-featured version, it had a one-third accuracy on both datasets.

The Faster R-CNN detector reached the highest accuracy of 60 % on the FlickrLogos-32 dataset and 67 % on the TopLogo-10. In both cases, model was trained for 7 hours.

The test results indicate that although the YOLOv3 detector supposed to be faster than Faster R-CNN detector and should achieve similar accuracy (according to the documentation), it was slower and less accurate on both datasets. This could be due to bad implementation of the YOLOv3 detector. In this case, the Faster R-CNN detector proved to be a better choice for logo detection.

Key words:

COCO, Object detection, logo detection, Faster R-CNN, FlickrLogos-32, deep learning, convolutional neural network, neural network, openCV, Python, PyTorch, TopLogo-10, Tensorflow, YOLOv3,

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

AP	Average Precision
CNN	Convolutonal neural network
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network library
CV	Computer vision
FPN	Feature Pyramid Network
IoU	Intersection over Union
JSON	JavaScript Object Notation
R-CNN	region - convolutional neural network
SSD	Single shot detector
YOLO	You only look once

1 Úvod

V současnosti je stále více využívána technologie hlubokého učení neuronových sítí, díky které lze automatizovat systémy či hledat řešení složitějších problémů. Jednou z oblastí, kterými se tato technologie zabývá, je detekce objektů v obraze. Díky ní lze vytvořit programy které můžou sloužit v mnoha oblastech od automobilismu, přes kontrolu produktů až po filtrování obsahu. Konkrétním příkladem detekce objektů v obraze je detekce log v obraze, kterou se tato práce zabývá.

Cílem této bakalářské práce je seznámit se s problematikou neuronových sítí a hlubokého učení, vytvořit rešerši v oblasti jejich využití v rámci detekce objektů a log v obraze a otestovat vybrané modely pro detekci log v obraze na vybraných trénovacích databázích.

V rešeršní části této práce bude popsána problematika hlubokého učení a jeho využití v praxi. Dále budou popsány neuronové sítě a jejich využití v rámci detekce objektů a detekce log. Budou popsány stávající systémy pro detekci log a testovací databáze log pro trénování neuronových sítí. Pro otestování budou vybrány dva modely pro detekci objektů v obraze a provedeno jejich testování na dvou testovacích databázích log. Tyto vybrané modely budou rovněž popsány v rešeršní části. Praktická část se bude věnovat popisu a rozdělení vybraných testovacích databází a následnému testování a porovnání výsledků vybraných modelů. Na závěr budou tyto modely porovnány i se stávajícími systémy pro detekci log v obraze.

2 Teoretická část

2.1 Strojové učení

Strojové učení je schopnost algoritmů získávat informace pomocí extrahování vzorů ze surových dat. Na rozdíl od klasického programování je strojové učení založené na reálných datech, ze kterých extrahuje vzory, na jejichž základě je schopné provádět samostatná rozhodnutí. Příkladem strojového učení může být logická regrese.

Účinnost algoritmů strojového učení je závislá na reprezentaci dat, která mají k dispozici. Využívají veškeré informace ze vstupních dat, s jejichž pomocí se učí vyhledávat skryté vzory nebo další informace obsažené ve vstupních datech. Některé informace však mohou být obtížněji detekovatelné, např. přízvuk člověka v případě rozpoznávání hlasu. Tento typ údajů vyžaduje sofistikovanější chápání v podstatě na lidské úrovni. K detekci těchto komplexních problémů slouží hluboké učení.

2.2 Hluboké učení

Hluboké učení je technika strojového učení, která učí hluboké neuronové sítě. Tyto sítě jsou označovány jako hluboké, protože mají velký počet vrstev. Počítačový model se učí provádět klasifikaci objektů z obrázků, textů, zvukových záznamů a dalších zdrojů dat. Modely hlubokého učení mohou v určitých případech dosáhnout přesnosti převyšující přesnost lidskou.

Za pomoci neuronových sítí, které obsahují mnoho vrstev, jsou modely

trénovány pomocí rozsáhlých datových setů. Je tak možné vytvářet komplexní úlohy a zpracovávat je pomocí jejich rozdělení na menší, jednodušší úkoly.

Podnětem pro vývoj hlubokého učení byl výzkum v oblasti biologického učení. Díky rozvoji v této oblasti bylo možné implementovat první modely neuronů a jejich následné trénování. Příkladem tohoto modelu neuronu je perceptron, což je nejjednodušší neuronová síť, obsahující pouze jeden neuron. V průběhu dalšího vývoje došlo ke spojení více neuronů do neuronových sítí a trénování na těchto sítích. Tyto pokročilejší sítě byly složené z více vrstev a umožňovaly i zpětnou vazbu.

Příčinou současného rozmachu neuronových sítí a hlubokého učení jsou rychlejší počítače s větší pamětí a lepším výpočetním výkonem. Dříve byla dostupná pouze data, avšak neexistoval způsob, jak tato objemná data ukládat a efektivně s nimi pracovat. V současné době jsou k dispozici výkonné procesory a grafické karty, které umožňují pracovat s větším objemem dat. Díky těmto technologickým pokrokům je možné testovat nové možnosti využití hlubokého učení a neuronových sítí. Postupným vývojem hlubokého učení byla zlepšena jeho schopnost poskytovat přesnější výsledky, a proto dnes přibývá stále více oblastí každodenního života, ve kterých je možné algoritmy hlubokého učení využít.

2.2.1 Hluboké učení v praxi

Aplikace využívající hluboké učení ve spojení s detekcí objektů v obraze jsou dnes používány v mnoha odvětvích. Lze se s nimi setkat např. v autonomních vozidlech, u počítačového vidění, filtrování sdíleného obsahu, při kontrole produktů a v mnoha dalších oblastech.

Detekce objektů v obraze

Detekce objektů v obraze může být využita například v odvětví automobilismu pro detekci dopravního značení, chodců, světelné signalizace nebo hlídání jízdních pruhů. Existují rovněž systémy, které dokáží sledovat řidiče a jeho reakce, na jejichž základě ho upozorní např. na nutnou přestávku.

Dalším příkladem použití může býtkontrola kvality výrobků, kterou se zabývá software PEKAT VISION. Tento software lze natrénovat na detekování vad či rozlišování produktů, pomocí kamer umístěných na výrobní lince. [1]

Detekce objektů je velkým přínosem také pro zdravotnictví. Výzkumný tým Kalifornské univerzity v Los Angeles vyvinul pokročilý mikroskop, který dokáže pomocí detekce objektů v obraze rozlišit rakovinové buňky od bílých krvinek, a to s přesností více než 95 %. Při použití mikroskopu je možné zkoumat vzorky bez jejich poškození, a navíc identifikovat různé vlastnosti buněk jako např. velikost, zrnitost či biomasu. [2]

Detekce log v obraze

Konkrétním příkladem využití detekce log v obraze v praxi je systém pro detekci log společností zabývajících se výrobou lihovin. V Thajsku je nelegální úmyslné zveřejňování obrázků a fotografií, na kterých se vyskytují loga výrobců lihovin. Pichitchai Pimkote a Thanapat Kangkachit [3] proto vytvořili model konvoluční sítě, který detekuje přítomnost loga na vstupním obraze a provádí případnou klasifikaci značky daného loga. Tento systém je tedy možné používat jako filtr, který upozorní na nevhodný obsah obrázku nebo zabrání jeho nahrání např. na sociální síť.

2.2.2 Limity hlubokého učení

V současnosti dochází k prudkému vývoji v oblasti hlubokého učení. Přestože je neustále zdokonalováno, nedosahuje dosud takové úrovně, aby mohlo plně zastoupit člověka ve všech jeho úkonech. Jedním z jeho zásadních omezení je absence přirozené lidské schopnosti orientovat se v nepředvídatelných situacích, vyskytujících se v realitě každodenního života. Algoritmy hlubokého učení nemohou vyvodit závěry týkající se jiného problému, než na který byly naprogramovány a naučeny. Chybí jim také pochopení myšlenkových operací jako je tzv. čtení mezi řádky; nedokáží dlouhodobě plánovat a postrádají kreativitu či

představivost. Většina algoritmů hlubokého učení se zaměřuje pouze na klasifikaci nebo redukci rozměrů.

2.3 Neuronové sítě

Neuronové sítě jsou jedním z výpočetních modelů, který je využíván hlubokým učením a umělou inteligencí. Jsou určeny pro distribuované paralelní zpracování dat.

Vzorem pro vytvoření těchto sítí bylo fungování lidského mozku. Lidský mozek se skládá z 86 miliard nervových buněk, které se nazývají neurony. Ty jsou mezi sebou navzájem propojeny pomocí neuronových výběžků – axonů a dendritů. Dendrity reagují na stimulace z vnějšího prostředí. Tyto „vstupy“ vytvářejí elektrické impulsy, které rychle procházejí neuronovou sítí. Neuron pomocí axonu může, nebo nemusí vyslat zmíněné impulsy k dalšímu neuronu. [4]

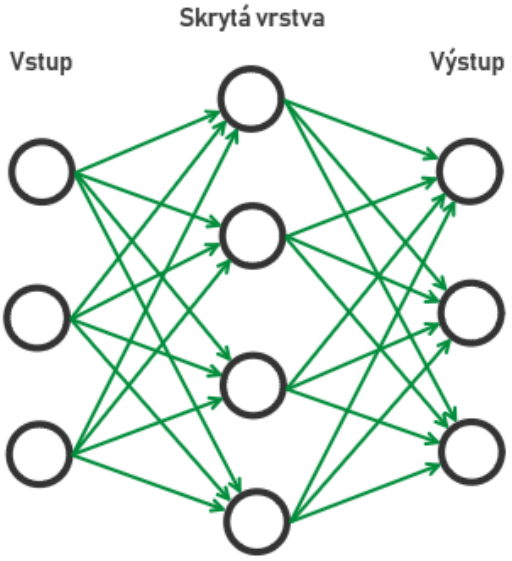
Umělé neuronové sítě fungují na stejném principu. Skládají se z umělých neuronů, jejichž vzorem je neuron biologický. Stejně jako má biologický neuron vždy jen jeden axon a jeden nebo více dendritů, tak i umělý neuron má pouze jeden výstup a libovolný počet vstupů. Signál se v umělé neuronové síti předává mezi těmito umělými neurony, jež pomocí přenosové funkce vyhodnotí, zda signál poslat, či neposlat dalšímu neuronu.

Existuje mnoho modelů uměle vytvořených neuronů. Některé využívají velmi jednoduché nespojité přenosové funkce; oproti tomu složitější modely popisují každý detail chování biologického neuronu. Jedním z nejpoužívanějších modelů umělého neuronu popsali v roce 1943 McCulloch a Pitts [5].

Vstupem neuronu může být výstup z jiného neuronu nebo informace z vnějšího prostředí. Tyto vstupní spoje mají udanou důležitost pomocí synaptických vah. Možností nastavení vah v síti takovým způsobem, aby odpovídaly výslednému řešení, je nekonečně mnoho. Pomocí funkce, vstupních dat a vah k jednotlivým vstupům může být vypočítána hodnota, která je porovnávána s prahem neuronu θ; výsledek tohoto porovnání následně rozhodne o výstupu.

2.3.1 Architektury sítě

Vícevrstvé sítě jsou tvořeny minimálně třemi vrstvami neuronů. Vrstvou vstupní, výstupní a minimálně jednou vrstvou vnitřní (skrytou). Mezi jednotlivými vrstvami sítě jsou všechny neurony propojeny tzv. úplným spojením – každý neuron z dané vrstvy je spojený s každým neuronem z vrstvy sousední, viz obr. 2.1.



Obrázek 2.1: Schema neuronové sítě s jednou skrytou vrstvou. (zdroj: autor)

Počet skrytých vrstev závisí na úkolu, k jehož řešení je daná neuronová síť určená. Neznamená proto, že čím více skrytých vrstev, tím lépe. Lepší výsledky nejsou přímo úměrné většímu počtu skrytých vrstev

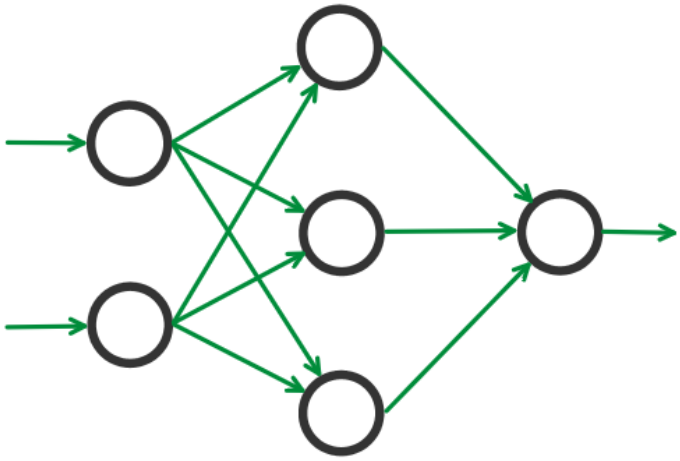
Existují dva typy topologií umělých neuronových sítí, a to síť dopředná a síť rekurentní. Pro detekci objektů v obraze se využívají dopředné sítě, zatímco sítě rekurentní se využívají pro aplikace, které pracují se sekvenčními daty, jako je např. text, zvuk nebo číselné řady. Vzhledem k zaměření práce budou dále popsány pouze sítě dopředné.

Dopředné sítě

V dopředné neuronové síti putují informace pouze jednosměrně. Jednotlivé neurony posílají informace dalším neuronům, od kterých však nedostávají žádnou

zpětnou vazbu. Síť neobsahuje žádné zpětné smyčky a neurony v této síti mají pevně dané vstupy a výstupy. Dopředné sítě se používají např. při generování nebo rozpoznávání vzorů.

Speciálním typem dopředné sítě je perceptron (viz kap. 2.2). Jeho využití je ovšem velmi omezené, jelikož je možné ho použít pouze na množiny, které jsou lineárně separovatelné.



Obrázek 2.2: Schéma dopředné neuronové sítě. (zdroj: autor)

2.3.2 Typy učení neuronových sítí

Neuronové sítě je možné kategorizovat na základě způsobu, kterým se učí. Jsou rozlišovány tři základní typy učení – s učitelem, bez učitele a posilované učení. Pro detekci objektů v obraze se používá typ učení s učitelem - je potřeba naučit síť klasifikovat objekty podle zadaných vzorů. Zbývající dva typy se používají v robotice, herních strategií nebo při vyhledávání skrytých vzorů v datech.

Učení s učitelem

Učení s učitelem, nazýváno také vedené učení, je strategie, která zahrnuje učitele, jenž je chytřejší než síť. Učitel dává síti data ke zpracování, k nimž sám zná výsledek. Síť poté provede své odhady, na něž učitel poskytuje odpovědi. V

případě, že byl původní odhad chybný, provede síť za účelem docílení správného výsledku na základě svých chyb úpravy vah a prahů předchozích vrstev. Tento algoritmus se nazývá algoritmus zpětného šíření chyby. Jsou rozlišovány dva typy šíření zpětné chyby, a to dávkové, kdy se váhy a prahy v síti mění až po skončení celého trénovacího cyklu, a sekvenční, kdy síť provádí úpravy po každém testovacím vzorku.

Trénování probíhá v trénovacích cyklech. Síť prochází všechna data z trénovací množiny; jejich pořadí má vliv na výsledek. Z tohoto důvodu je vhodné procházet data v náhodném pořadí.

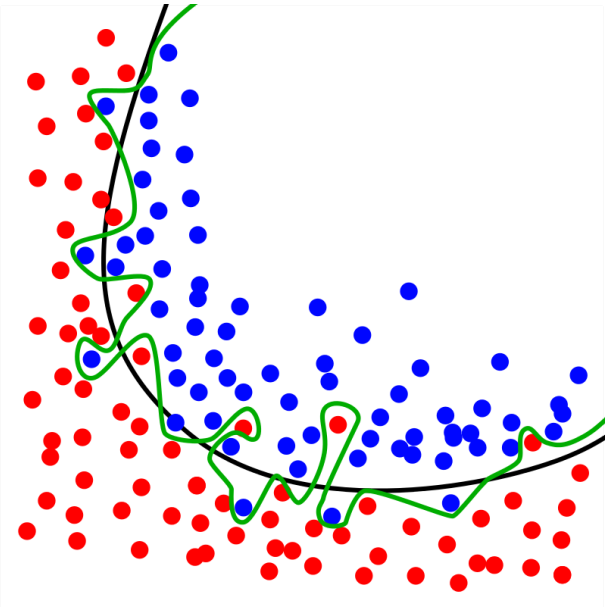
2.3.3 Přeučení

Při učení sítě může nastat jev, který se nazývá přeučení. Pokud má neuronová síť malý počet neuronů, má menší šanci na vystihnutí a popsání závislostí na trénovacích datech. Pokud ale síť obsahuje velký počet neuronů, je pravděpodobné, že snadno popíše závislosti na trénovaných datech, ale její schopnost vystihnout na nových datech správný výsledek bude horší. K tomuto jevu může dojít např. pokud je síti k dispozici velký počet vstupních parametrů, ale málo testovacích dat. Je potřeba nalézt kompromis mezi schopností sítě popsat závislost na trénovacích datech a závislost na datech nových.

Na obrázku 2.3 reprezentuje zelená křivka přeučení sítě a černá křivka ideálně natrénovaný model sítě.

2.4 Klasifikace objektů v obraze pomocí neuronových sítí

Pro detekci objektů v obraze nejsou neuronové sítě samy o sobě příliš vhodné. Nevýhodou využití neuronových sítí pro tento účel je velikost a struktura vstupu. Jako příklad lze uvést detekci v obraze o rozměru 224×224 pixelů se třemi barevnými kanály. Pokud by byla použita vícevrstvá perceptronová neuronová síť,



Obrázek 2.3: Přeučení sítě. (zdroj: [vid. 2019-20-03] dostupné z:https://commons.wikimedia.org)

obsahovala by přibližně 150 000 vah. Toto enormní množství může být velice výpočetně náročné a může vést k přeučení sítě (viz kap. 2.3.3). Pokud je navíc obraz do sítě vložen jako řada pixelů, dochází ke ztrátě prostorových informací.

Dalším problémem použití neuronových sítí jsou jejich odlišné reakce na různá umístění detekovaného objektu. V případě, že se lokace daného objektu ve druhém obraze liší od jeho lokace v obraze prvním, síť se snaží přizpůsobit novému umístění a předpokládá, že detekovaný objekt se bude dále nacházet vždy na této pozici.

Tyto problémy řeší za pomoci filtrů konvoluční neuronové sítě.

2.4.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě – CNN (convolutional neural network) jsou dopředné neuronové sítě, které mají pevnou strukturu propojení – konvoluce. Váhami těchto sítí jsou konvoluční jádra. Tyto sítě jsou většinou hluboké, tzn. mají velký počet skrytých vrstev. Jsou používány především ke klasifikaci a detekci objektů v obraze.

Konvoluční sítě jsou schopny zachytit prostorové závislosti v obraze pomocí filtrů. Jejich úkolem je redukce obrazu na formu, která je snadněji zpracovatelná, aniž by došlo ke ztrátě informací obsažených ve vstupním obraze.

Filtry snižují počet vah, které se musí neuronová síť naučit, a zároveň umožňují jejich opětovné použití. Na vstupní obraz je postupně přes všechny jeho body aplikován filtr, pomocí něhož jsou pro tyto body prostřednictvím konvoluce vypočítávány hodnoty. Díky filtrům lze z obrazu získat informace, jako např. kolikrát a s jakou pravděpodobností se objekt, který je daným filtrem zastupován, objevuje v obraze. Informace generované pomocí filtrů se nazývají mapy prvků. Tyto mapy jsou následně předávány dál ke klasifikaci.

Konvoluční neuronové sítě mají dva speciální typy vrstev. První z nich se nazývá konvoluční vrstva. Vrstvy tohoto typu jsou zodpovědné za zachycení funkcí v obraze. První konvoluční vrstva zachycuje nízkoúrovňové funkce, jako jsou hrany, barvy, gradient atd. S dalšími konvolučními vrstvami se síť přizpůsobuje složitějším funkcím, a je tak schopná lépe porozumět datům.

Druhým typem speciální vrstvy, kterou konvoluční neuronová síť využívá, je tzv. sdružovací vrstva (pooling layer). Tato vrstva přijímá vstup z konvoluční vrstvy a stará se o zredukování rozlišení. Díky tomu je redukována velikost a úměrně k ní také potřebný výpočetní výkon a čas. Existují dva typy sdružování dat. První typ vrací maximální hodnotu ze všech hodnot v části obrazu pokryté jádrem. Druhý typ vrací průměrnou hodnotu z dané části. Typ shlukování využívající maximální hodnotu také funguje jako mechanismus pro redukci šumu.

Konvoluční vrstva spolu se sdružovací vrstvou tvoří jednu vrstvu konvoluční neuronové sítě. Zvýšením počtu těchto vrstev v síti je možné dosáhnout lepších výsledků, avšak za cenu vyšší výpočetní náročnosti. Výstup z těchto vrstev je nakonec předán do klasické neuronové sítě za účelem klasifikace.

K dispozici jsou různé architektury CNN jako například LeNet, AlexNet, VGGNet, GoogLeNet, ResNet apod.

2.5 Detekce log v obraze

Reklamní technologie, známé pod názvem Ad Tech, jsou používány výrobci, dodavateli a obchodními agenturami k analýze a získávání informací o potenciálních zákaznících. Pro tyto technologie se v současné době stalo hlavním nástrojem hluboké učení. Slouží k identifikaci např. produktů, značek a log na veřejně publikovaných obrázcích. Jelikož nejsnazší způsob, jak rozeznat značku produktu, je prostřednictvím právě jeho loga, jedná se především o detekci log v obraze.

Loga, která jsou zároveň ochrannou známkou, mají v oblasti marketingu důležitý význam – slouží k reprezentaci společností, spolků či produktů a k jejich rozeznatelnosti. Rozpoznávání log v obraze je klíčové pro mnoho aplikací, sloužících např. k zjišťování porušení autorských práv, detekci log vozidel pro inteligentní systémy řízení dopravy, kontextovému umisťování inzerce apod.

2.5.1 Systémy pro detekci log

Existují různé systémy, které využívají hluboké učení a neuronové sítě pro detekci log v obraze. Pro porovnání bylo vybráno několik odborných prací, které se touto problematikou zabývají.

Jedním z příkladů jsou systémy využívající metodu tzv. keypoint matching, ve které jsou porovnávány body popisující hledaný objekt na vstupním obraze. Tímto tématem se zabývají práce Scalable logo recognition in realworld images [6] a Bundle min-hashing for logo recognition [7]. Díky vylepšení reprezentace loga dosahují tyto systémy velmi vysokých přesností při detekci.

Některé práce, např. Deep learning logo detection with data expansion by synthesising context [8] nebo On the benefit of synthetic data for company logo detection [9], se zabývají systémy, které generují syntetické datasety nebo testují jejich využití v rámci detekce log, díky nimž je možné rozšířit stávající datasety, a tak zlepšit výsledky při trénování.

Dalšími příklady jsou systémy využívající metody jako CNN, FPN, Faster

R-CNN apod. Testují použití dalších možných vylepšení, jako jsou augmentace nebo jiné úpravy pro vylepšení přesností.

Autor	Název práce	Rok	Publ.	Metoda	Dataset	mAP
Stefan Romberg	Scalable Logo recognition in Real-World Images	2011	ICMR	keypoint matching	FlickrLogos-32	98.2 %
Stefan Romberg	Bundle Min-Hashing for Logo Recognition	2013	ICMR	keypoint matching	FlickrLogos-32	99.9 %
Hang Su	Deep Learning Logo Detection with Data Expansion by Synthesising Context	2017	WACV	Faster R-CNN	FlickrLogos-32	81.1 %
Christian Eggrt	On the Benefit of Synthetic Data for Company Logo Detection	2015	MM	R-CNN + SS + VGG16	FlickrLogos-32	99.6 %
Montserrat Daniel Mas	Training Object Detection And Recognition CNN Models Using Data Augmentation	2017	IMAWM	Faster R-CNN (ZF, VGG16)	FlickrLogos-32	85.4 %
Christian Eggert	Improving Small Object Proposals for Company Logo Detection	2017	ICMR	Faster R-CNN + FPN	FlickrLogos-32	67.1 %
Andras Tüzkö	Open Set Logo Detection and Retrieval	2017	VISAPP	Faster R-CNN + CNN	FlickrLogos-32	84.2 %

Tabulka 2.1: Srovnání vybraných systémů v oblasti detekce log v obraze. (zdroj: autor)

Existuje mnoho různých systémů, které úspěšně vylepšují funkčnosti jednotlivých částí procesu hlubokého učení a detekce log v obraze. Zmíněnými odbornými pracemi a jejich způsoby vylepšení detekce je možné se inspirovat pro úpravy stávajících algoritmů pro maximální možnou přesnost algoritmů vlastních.

2.5.2 Datasety pro detekci log

Pro trénování detekce objektů v obraze za pomocí hlubokých neuronových sítí je dostupné množství datasetů, příkladem mohou být COCO nebo Pascal VOC. Některé datasety disponují velkým počtem obrazů a tříd pro klasifikaci, jiné mohou být specifické pro jednu konkrétní třídu.

Existují datasety specializované přímo na trénování detekce log v obraze. Příkladem těchto datasetů jsou FlickrLogos, TopLogo, WebLogo nebo SynthLogo. Tyto datasety se od sebe odlišují množstvím obsažených obrázků, počtem rozlišovaných tříd nebo zdrojem, odkud obrázky pocházejí. Může se jednat například o vzorová loga, obrázky log z fotografií nebo synteticky vytvořené obrázky s logy, které zkoumá například práce „Logo detection and recognition with synthetic images“ [10].

V případě že jsou dostupné datasety svým obsahem nevyhovující, lze za použití specializovaných nástrojů na označování objektů v obraze a jejich uložením do požadovaného formátu vytvořit dataset nový.

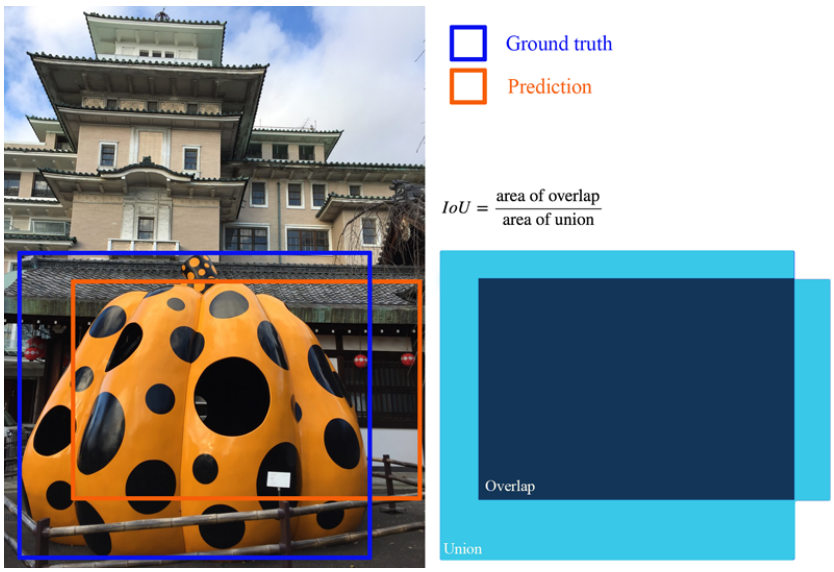
Většina nejpoužívanějších datasetů používá svůj vlastní formát zápisu dat. Mezi nejpoužívanější formáty patří COCO, pascal/VOC (Pascal VOC), imagenet (ImageNet) atd. Existuje rovněž formát TFRecords, který byl vyvinut a optimalizován pro framework Tensorflow. Data v tomto formátu jsou uložena binárně, tudíž zabírají méně místa a lze z nich číst mnohem efektivněji. To je nespornou výhodou při práci s rozsáhlým množstvím dat. Pro převod mezi jednotlivými formáty existuje mnoho algoritmů, závisí pouze na tom, jaký typ formátu umí daný detektor zpracovat.

2.6 Evaluace a přesnost detekce

Výsledky detekce lze rozdělit do čtyř skupin. „True positives“ – správně označené objekty, „True negatives“ – správně neoznačené objekty, „False positives“ – špatně označené objekty a „False negatives“ – špatně neoznačené objekty (neoznačené objekty, které však měly být označeny). Výsledná přesnost, označována zkratkou AP (Average precision), je vypočítávána z poměru počtu správně označených objektů (True positives) k celkovému počtu detekovaných objektů (True positives + False positives). Další mírou hodnocení výsledků je tzv „Recall“. Tato hodnota udává poměr správně označených objektů (True positives) k celkovému počtu objektů (True positives + False negatives). Jako příklad může

sloužit obrázek v němž jsou čtyři objekty stejné třídy. Detektor detekuje pouze dva objekty v obrázku a správně označí třídu které patří. Výsledná přesnost je tak 100%, ale recall pouze 50%.

$$AP = \frac{True\ \ positives}{True\ \ positives + False\ \ positives} \quad Recall = \frac{True\ \ positives}{True\ \ positives + False\ \ negatives}$$



Obrázek 2.4: Graficky znázorněný výpočet IoU (zdroj: [vid. 2019-20-03] dostupné z:`https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173`)

Přesnost (AP) může být doplněna dolním číselným indexem označujícím takzvanou IoU hodnotu (Intersection over Union), která udává přesnost umístění boxu v detekovaném objektu na škále od 0 do 1. Hodnota IoU je vypočítávána jako poměr průniku předpovídaného a správného boxu k jejich sjednocení, viz obr. 2.4. Přesnost (AP) bez dolního indexu udává přesnost průměrovanou z deseti prahových hodnot, a to od 0.5 až do 0.95, vždy s krokem 0.05. AP_{50} udává přesnost 0.5 a vyšší, AP_{75} 0.75 a vyšší. Indexy označené písmeny S, M, L udávají přesnost na různě velkých objektech. S je označení pro objekty menší než 32^2 pixelů, M pro objekty o velikosti od 32^2 pixelů do 96^2 pixelů a L pro objekty větší než 96^2 pixelů.

2.7 Detektory

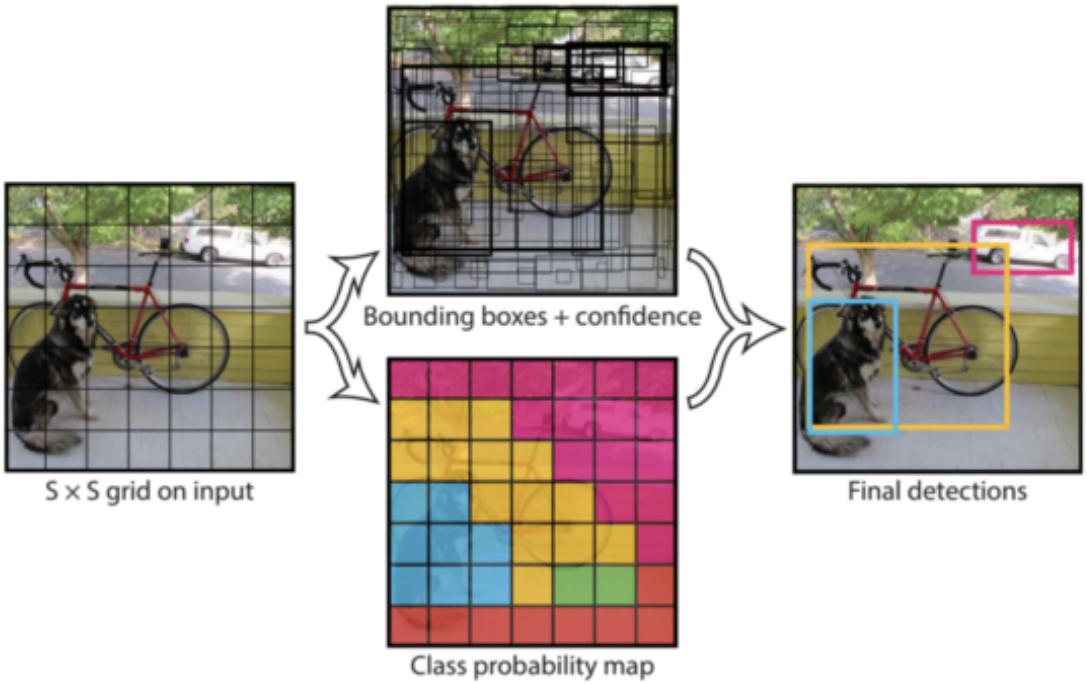
Pro detekci objektů v obrazu nemůže být použita standardní konvoluční síť, jelikož při detekci není délka výstupní vrstvy sítě konstantní. Tato nekonstantnost vychází z faktu, že není znám celkový počet výskytů hledaných objektů. Pokud by byl vstupní obraz rozdělen do mnoha regionů, bylo by možné v těchto regionech pomocí konvoluční neuronové sítě zjistit, zda se v tomto regionu objekt nachází a o jaký objekt se jedná. Objekty na vstupním obraze mohou mít různá prostorová umístění a také různé velikosti. Problém, jak správně rozdělit tyto regiony, a to co nejrychleji a s co nejmenší možnou výpočetní náročností, řeší detektory YOLOv3 a Faster R-CNN.

2.7.1 YOLOv3

YOLOv3 neboli „You only look once“ je systém pro detekci objektů v obraze. Ostatní detekční systémy aplikují model neuronové sítě několikanásobně na různé regiony o různé velikosti. Oblasti s nejvyšším skóre jsou poté sítí považovány za detekce. YOLOv3 k problému přistupuje zcela odlišným způsobem. Na jeden obraz je aplikována pouze jedna neuronová síť. Tato síť rozloží obraz do mnoha oblastí a následně detekuje objekty a jejich pravděpodobnosti pro jednotlivé třídy v každé oblasti zvlášť. Díky tomu, že v tomto systému prochází obrázek sítí jako celek a pouze jednou, disponuje YOLOv3 velkou výhodou v podobě rychlosti. Oproti konkurenčnímu detekčnímu systému R-CNN je až tisíckrát rychlejší; až stokrát rychlejší oproti jeho rychlejší variantě Fast R-CNN. [11]

Na vstupu se nachází obrázek, ve kterém má být provedena detekce, a dále třídy neboli kategorie obrázků, které má model detekovat. Tento obrázek je rozdělen na mřížku o velikosti $N \times N$ viz obr. 2.5, kde N může být zastoupeno libovolným celým přirozeným číslem. Na každé z buněk této mřížky je provedena detekce. Pokud se střed detekovaného objektu nachází v některém z čtverců mřížky, pak právě tento čtverec je za tuto predikci dále zodpovědný.

Jsou předpovídány ohraničující boxy (bounding boxes), které obklopují detekovaný objekt. Tyto boxy obsahují číselnou informaci, která udává míru pravděpodobnosti výskytu objektu v označené části a zároveň příslušnost k některé z tříd. Každý box se skládá z pěti hodnot: x, y, w, h a výše zmíněné číselné informace. Souřadnice x a y udávají střed boxu detekce vzhledem k okrajům buňky mřížky. Hodnoty w a h udávají šířku a výšku boxu vzhledem k celému obrázku.



Obrázek 2.5: Rozdělení vstupního obrázku pomocí mřížky a přiřazení jednotlivých buněk k určité třídě. (zdroj: [vid. 2019-20-03] dostupné z:`https://pjreddie.com`)

Každá z buněk mřížky také předpovídá pravděpodobnosti podmíněné třídy C,Pr(třída | objekt).Tyto pravděpodobnosti jsou podmíněny buňkou mřížky obsahující objekt. Je předpokládána pouze jedna množina pravděpodobností třídy na buňku mřížky bez ohledu na počet předpovězených boxů. Takto obsahuje každý box skóre, které je specifické pro jednotlivé třídy. Toto skóre předpovídá jak pravděpodobnost, že se třída objeví v poli, tak jak dobře předvídané pole odpovídá objektu [12].

Omezení YOLOv3

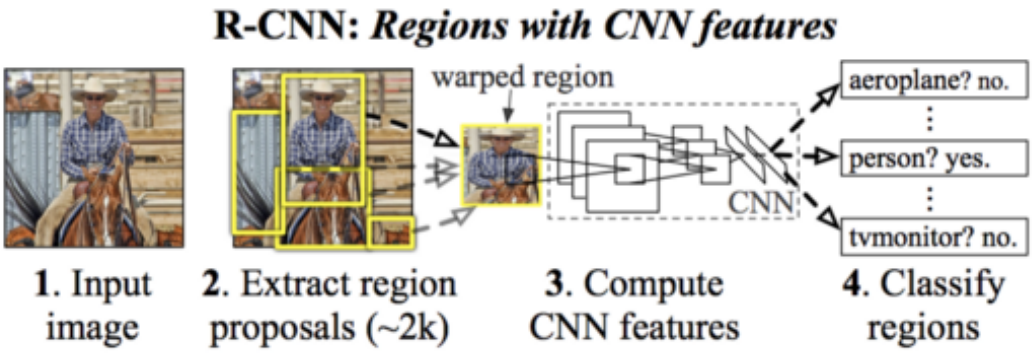
YOLOv3 má omezení týkající se ohraničujících boxů. Každá buňka v mřížce může mít pouze dva boxy a může patřit maximálně do jedné třídy, následkem čehož jsou omezeny počty blízkých objektů, které YOLOv3 dokáže předpovědět. Detektor se tak potýká s případy, kdy se objekty nachází v těsné blízkosti nebo kdy je soustředěno větší množství objektů na malém prostoru, např. davy lidí, ptačí hejna apod. Detektor se učí předvídat hranice podle dat, a tak se snaží zobecnit detekce i na objekty s neobvyklým poměrem stran.

Dalším omezením je používání ztrátových funkcí, pomocí nichž zpracovává chyby. Ztrátová funkce přistupuje rozdílně k chybám v malých a ve velkých boxech. Zatímco malá chyba ve velkém boxu nemá téměř žádný vliv, malá chyba v malém boxu má značný dopad na odhadovanou přesnost. Hlavním zdrojem chyb jsou tak nesprávné lokalizace. [11]

2.7.2 Faster R-CNN

R-CNN (Region-convolutional neural network) je dalším příkladem systému pro detekci objektů v obraze. Tato metoda využívá extrakci oblastí (regionů) z obrázku pomocí selektivního vyhledávání. Tento konkrétní detektor využívá rozdělení vstupního obrázku na dva tisíce regionů. Jednotlivé regiony prochází konvoluční neuronovou sítí, která detekuje objekty v daných regionech a zařazuje je do tříd. Konvoluční neuronová síť sama o sobě dokáže detekovat pouze to, co se na daném obrázku nachází, nikoli však už umístění daného objektu. Spojením regionů a konvolučních neuronových sítí tak dokáže model detekovat třídu i lokaci hledaného objektu.

Na vstupu se nachází obrázek, ve kterém má být provedena detekce. Tento obrázek je rozdělen pomocí selektivního vyhledávání do dvou tisíců regionů. Selektivní vyhledávání rozděluje obrázek na segmenty podle barvy, textury, velikosti a tvaru. Na začátku je tento obrázek algoritmem „přesegmentován“ podle intenzity pixelů. V následujícím průběhu obrázek prochází dalšími koly segmen-



Obrázek 2.6: Rozdělení vstupního obrázku na regiony a následná klasifikace regionů pomocí konvoluční neuronové sítě. (zdroj: [vid. 2019-20-03] dostupné z:`https://towardsdatascience.com`)

tace, ve kterých se propojují nejpodobnější segmenty do větších celků. Takto algoritmus pokračuje až dosáhne rozdělení na požadovaný počet segmentů čili hledaných dvou tisíců regionů. Rozdělené regiony jsou předány do konvoluční neuronové sítě, která produkuje vícerozměrný vektor. Tato konvoluční neuronová síť funguje jako extraktor rysů. Používá různé filtry a úpravy obrázku, které jsou spojeny ve výstupní vrstvě ze sítě. Tato výstupní data jsou poté předána do metody podpůrných vektorů za účelem klasifikace přítomnosti objektů v daném regionu.

Fast R-CNN

Omezení R-CNN spočívá v nutnosti procházení každého ze dvou tisíců regionů v každém obrázku, což vede k obrovskému množství dat a vysokému trénovacímu času připadajícímu na jeden obrázek, který se pohybuje kolem 47 sekund [13]. Detekce tak nemůže být implementována v reálném čase, jelikož je příliš pomalá. Tento problém však řeší upravená implementace, která se nazývá Fast R-CNN. Namísto plnění konvoluční neuronové sítě dvěma tisíci regiony je do ní posílán přímo vstupní obraz. Síť poté vytvoří konvoluční mapu, ze které je možné získat rozložení regionů. Po transformaci je možné z těchto dat pomocí softmax funkce předpovědět třídy a navrhované oblasti objektů.

Faster R-CNN

Přestože implementace Fast R-CNN dokázala zrychlit původní R-CNN z 47 sekund na přibližně 2 sekundy, je stále zpomalena časově náročným selektivním vyhledáváním [13]. Pokud jsou regiony známy předem, je síť výrazně rychlejší. Síť proto byla vylepšena, aby se naučila předpovídat regiony, a bylo tak možné vypustit vyhledávání regionů pomocí selektivního vyhledávání. Tato implementace se nazývá Faster R-CNN. Funguje na stejném principu jako Fast R-CNN, ale s tím rozdílem, že si síť sama dokáže navrhnout polohu regionů. Tím bylo možné zrychlit detekci obrazu na 0,2 sekundy, čímž se přiblížila konkurenčnímu YOLOv3 [13].

Další detektory

Existují i další detektory jako jsou například SSD (single shot detektor), RetinaNet a FPN (Feature Pyramid Network). Tyto detektory se souhrně nazívají „one stage" detektory. Na rozdíl od R-CNN, tzv. „two stage" detektoru, používají stejně jako YOLOv3 přístup, kdy vstupní obrázek prochází neuronovou sítí pouze jednou. Tyto detektory se liší v implementacích, ale všechny mají společný cíl, a to detekce objektů v obraze.

2.7.3 Porovnání

Tabulka 2.2 porovnává přesností jednotlivých detektorů na datasetu MS COCO.

V případě YOLOv3 byla oproti jeho předchůdci YOLOv2 zvýšena přesnost (která je nyní podobná jako u Faster R-CNN), aniž by došlo ke zpomalení. Z tabulky 2.2 lze vyčíst konkrétní údaje – oproti původní přesnosti 21.6 % u verze YOLOv3 byla u verze YOLOv3 zvýšena přesnost na 33 % . Faster R-CNN je stále pomalejší než YOLOv3, přestože byl výrazně snížen čas potřebný pro detekci. YOLOv3 však dokáže detekovat pouze dva objekty na jednu buňku mřížky, a proto má ve srovnání s Faster R-CNN, který takto omezen není, nevýhodu při detekci velkého počtu objektů na malém prostoru. Hlavním zdrojem chyb YO-

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Tabulka 2.2: Porovnání přesností detektorů na datasetu MS COCO. (zdroj: [vid. 2019-20-03] dostupné z:`https://medium.com`)

LOv3 je špatná lokalizace. To lze sledovat v tabulce 2.2, kde je pro YOLOv3 AP menší než u Faster R-CNN, ale pro AP_{50} je jeho přesnost už na stejné úrovni.

Přestože tyto chyby nejsou nějak velké, pro úlohy, u kterých závisí především na přesnosti a pro které je rychlost, kterou poskytuje Faster R-CNN, dostačující, je Faster R-CNN lepší volbou než YOLOv3. Pokud však lokalizace není důležitým faktorem a důraz je kladen především na rychlost, je lepší volbou YOLOv3. Nelze obecně určit, zda je jeden z modelů lepší než druhý, záleží na úloze, kterou má daný model vykonávat.

2.8 Frameworky Tensorflow a PyTorch

Všechny organizace se zaměřují na co největší možnou automatizaci a vyhýbají se jakémukoli druhu manuální závislosti na některém ze sektorů svého podnikání. Tomuto současnému trendu vyhovují oblasti umělé inteligence a hlubokého učení. Velké společnosti jako jsou např. Google a Facebook mají své vlastní implementace frameworků pro hluboké učení, z nichž je většina vytvořena pro jazyk Python. Mezi zástupce těchto frameworků patří Tensorflow a PyTorch.

2.8.1 Tensorflow

Tensorflow byl vyvinut společností Google Brain a je aktivně využíván společností Google pro potřeby výzkumu a výroby. Jeho předchůdcem, který však nebyl volně dostupný, byl DistBelif. Tensorflow je jedním nejpopulárnějších frameworků pro hluboké učení v současné době (viz kap. 2.8.3).

Tensorflow umožňuje vytvářet takzvané „grafy datového toku“, které popisují, jakým způsobem se data pohybují grafem (v případě hlubokého učení sítí), nebo pole zpracovávaných uzlů. Každý uzel představuje matematickou operaci a spojení mezi nimi jsou buď vícerozměrná pole, nebo tenzory. Uzly a tenzory, které používá Tensorflow, jsou v Pythonu reprezentovány pomocí objektů. Vlastní matematické operace však nejsou prováděny v Pythonu, ale jsou to binární soubory dostupné prostřednictvím Tensorflow, napsané v jazyce C++ a optimalizované pro maximální výkon. Tensorflow tedy poskytuje vysoký stupeň abstrakce pro práci s tenzory. [14]

Aplikace využívající Tensorflow je možné spouštět pomocí procesorů i grafických karet na většině platforem, jako jsou počítače, klastry, zařízení iOS i Android.

Jednou z největších výhod, ale zároveň i nevýhod Tensorflow je jeho abstrakce. Díky ní se programátor nemusí zabývat drobnými detaily, které za něj Tensorflow obstará na pozadí - lze se tak zaměřit na celkovou logiku a implementaci aplikace. Za běhu programu ovšem nelze upravovat nastavení a strukturu sítě a sledovat data, která jí prochází.

2.8.2 PyTorch

PyTorch vychází z frameworku Torch, založeném na jazyce Lua, který byl vyvinut a v současnosti je používán společností Facebook. Nejedná se však pouze o tzv. wrapper pro podporu jazyka Python. Celý framework byl přepsán a přizpůsoben pro jazyk Python tak, aby byl rychlý a více korespondoval s prostředím jazyka.

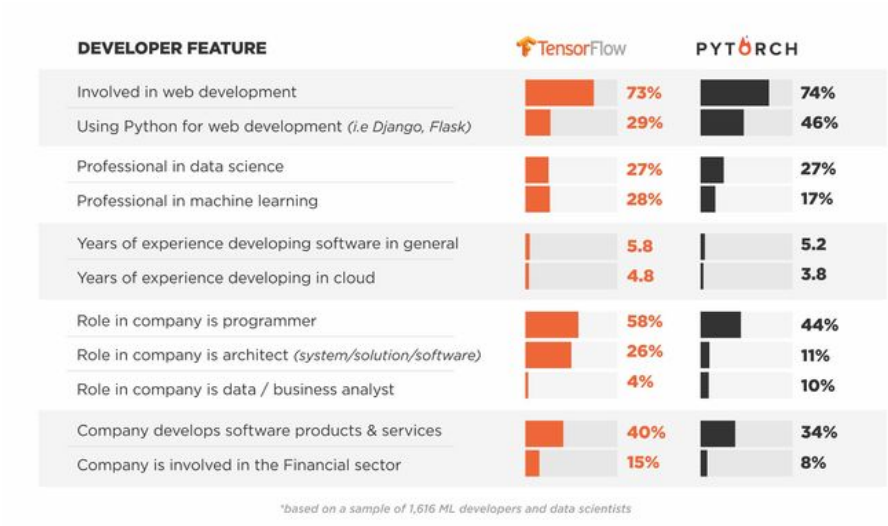
PyTorch poskytuje dvě hlavní funkce – výpočet tenzorů, který lze provádět i za pomoci akcelerace výkonných grafických karet, a budování hlubokých neuronových sítí. Díky tomu, že je PyTorch plně integrován do jazyka Python, může využívat všech jeho funkcí. Spolu s PyTorchem je možné použít i mnohé další balíčky pro Python, jako např. NumPy, SciPy atd., a s jejich pomocí rozšířit funkce, které PyTorch nabízí. Díky tomu je PyTorch flexibilní a lze ho snadno přizpůsobit konkrétním požadavkům. [15]

PyTorch nepoužívá statické grafy, které jsou běžně používané jinými frameworky, což umožňuje vývojářům měnit způsob chování neuronových sítí přímo za chodu aplikace. Při práci s PyTorchem je tak snadné pozorovat, co se v síti aktuálně děje, a podle toho přizpůsobit další kroky. Zejména díky tomu je PyTorch velice oblíbený u programátorů začínajících s tématikou hlubokého učení (viz kap. 2.8.3).

2.8.3 Porovnání

Z výsledků aktuálního průzkumu, zveřejněných na webové stránce developereconomics.com [17] vyplývá, že v současné době je z dvojice frameworků PyTorch-Tensorflow preferován druhý uvedený. Přibližně 43 % vývojářů, kteří se zabývají hlubokým učením, používá buď Tensorflow nebo PyTorch; z toho 86 % vývojářů používá Tensorflow jako hlavní framework. Ve srovnání s komunitou PyTorche je komunita Tensorflow složena z více profesionálních vývojářů a softwarových inženýrů. PyTorch je více využíván pro analýzu dat v obchodním prostředí a vývoj webových aplikací v jazyce Python, viz obr. 2.7. Díky své jednoduchosti je také více používán pro testování nápadů při vývoji. [16]

Oba porovnávané frameworky využívají tenzory a zobrazují jakýkoli model jako acyklický graf. Nicméně každý z nich přistupuje rozdílně k tomu, jak je tento graf definován. Tensorflow jej definuje staticky před spuštěním modelu. Veškerá komunikace s „vnějším světem“ je prováděna pomocí objektů tf.Session a tf.Placeholder., což jsou tenzory, které jsou při běhu programu nahrazeny externími daty.



Obrázek 2.7: Porovnání použití Tensorflow a PyTorch z průzkumu na webové stránce developereconomics.com. (zdroj: [vid. 2019-20-03] dostupné z:`https://www.developereconomics.com`)

PyTorch funguje mnohem dynamičtěji než Tensorflow. Je možné definovat, spouštět a měnit jeho různé součásti podle potřeby. Nejsou zde potřeba žádné relace ani zástupce pro předávání dat. Framework PyTorch je více integrován do jazyka Python a práce s ním je tak intuitivnější. Oproti tomu při práci s Tensorflow je framework oddělen od uživatele, který s ním pracuje pouze pomocí výše zmíněných relací a zástupců. Existuje také několik dynamických architektur neuronových sítí, které dokáží těžit z dynamického přístupu, který PyTorch nabízí.

Z porovnání frameworků Tensorflow a PyTorche vyplývá, že framework Tensorflow je populárnější především díky svým vizualizačním funkcím, zatímco novější PyTorch je populární díky dynamickému přístupu; jeho další výhodou je lepší integrovanost do jazyka Python, a především jednoduchost usnadňující orientaci.

K oběma frameworkům jsou na jejich oficiálních webových stránkách dostupné kvalitní dokumentace a také mnoho příkladů a projektů.

3 Praktická část

3.1 Data

Pro trénování detekce objektů v obraze jsou používány datasety, které je potřeba dále dělit. Ve většině případů se dataset rozděluje na dataset trénovací, který obsahuje 80 % dat, a testovací dataset, který obsahuje zbylých 20 %. Z trénovacího datasetu je následně odebráno 20 % dat pro vytvoření validačního datasetu. Výsledný poměr je tedy 60:20:20, kde 60 % patří právě trénovacímu datasetu. [18]

Trénovací datasety se používají pro vlastní trénování neuronové sítě. Tato data síť vidí a používá je k učení.

Validační datasety jsou používány pro ladění sítě. Síť tato data nepoužívá pří trénování. Na těchto datech probíhá pouze validace natrénované sítě, podle jejíchž výsledků je možné upravit parametry učení sítě. Na těchto datech lze rovněž pozorovat míru přeučení sítě.

Testovací datasety se používají pouze pro vyhodnocení úspěšnosti a výsledků neuronové sítě. Tyto datasety by se nikdy neměly jakýmkoli způsobem podílet na trénování dat, ať už na trénování samotném, nebo na úpravě parametrů sítě podle výsledků na daném datasetu.

3.1.1 Použité datasety

Pro účely této práce byly použity dva z dostupných datasetů pro detekci log v obraze. Jedná se o FlickrLogos-32 a TopLogo-10. Oba tyto datasety používají reálné fotografie, které byly staženy z komunitního webu pro sdílení obrázků a foto-

grafií Flickr.

Zatímco dataset FlickrLogos-32 se zaměřuje především na firemní loga společností z nejrůznějších oblastí, dataset TopLogo-10 obsahuje loga nejoblíbenějších módních značek oblečení, obuvi a doplňků.

FlickrLogos-32

Dataset flickrLogos existuje v několika variantách. Pro tuto práci byla použita varianta FlickrLogos-32. Tento dataset obsahuje přesně 8240 obrázků a 32 tříd. Každá třída zastupuje jedno logo. Dataset je rozdělen na testovací, validační a trénovací set. Testovací i validační set obsahují 3960 obrázků, z toho každé třídě náleží 30 obrázků, které obsahují alespoň jedno logo. Zbylých 3000 obrázků neobsahuje žádné logo reprezentované některou z tříd. Trénovací set je složen ze zbylých 320 obrázků, z nichž každé třídě náleží právě 10.

TopLogo-10

Dataset TopLogo-10 rozlišuje 10 tříd pro klasifikaci a obsahuje celkem 700 obrázků. Tento dataset obsahuje pouze testovací a trénovací set. Obrázky jsou rozděleny do dvou variant. První varianta obsahuje trénovací dataset 40 obrázků na jednu třídu (celkem tedy 400) a testovací set 30 obrázků na třídu (celkem 300). V druhé variantě je v trénovacím setu obsaženo 10 obrázků na třídu (celkem 100) a v testovacím setu 60 obrázků na třídu (celkem 600). Pro tuto práci byla použita první varianta datasetu.

Formát dat

Oba datasety používají formát COCO. Tento formát pracuje s tzv. anotacemi. Jedná se o JSON soubor, který obsahuje data k datasetu. Každý set (testovací, validační, trénovací) má vlastní JSON soubor. Tato anotace obsahuje obecná data jako např. název datasetu, rok, verzi, datum vytvoření apod. Dále obsahuje seznam obrázků k danému setu a ke každému obrázku jeho šířku, výšku, název, unikátní id a cestu. Rovněž obsahuje seznam „anotací“, které popisují výskyt

loga v některém z obrázků a jeho umístění na daném obrázku. Činí tak pomocí id obrázku, ve kterém se logo nachází, a tzv. bboxu, který se skládá ze souřadnic v obraze (X, Y) a šířky (W) a výšky (H) daného loga. Formát COCO obsahuje také seznam klasifikačních tříd, jejich id a název.

3.2 Hardware

Všechny části této práce byly testovány na stejném počítači. Na tomto počítači byl nainstalován operační systém Windows 10. Počítač byl osazen následující konfigurací: grafická karta NVIDIA GeForce GTX 960M s podporou CUDA, 8GB RAM a procesorem Intel Core i5-6300HQ.

Na počítači byl z důvodu ochrany hardwaru omezen maximální výpočetní výkon procesoru na 70 %. Trénování neuronové sítě probíhalo několik hodin denně, a tímto způsobem byl ochráněn hardware před vysokými teplotami a neustálým vytížením procesoru na 100 %, což by mohlo vést ke snížení jeho životnosti. Výpočetní výkon grafické karty omezen být nemusel. Teploty dosahované při trénování setrvávaly v bezpečných mezích.

3.3 Instalace a potřebné součásti

V této kapitole budou stručně shrnuty a popsány knihovny a balíčky, které byly potřebné pro funkčnost detektorů. Vzhledem k faktu, že obě implementace detektorů používají jazyk Python, bylo možné instalovat většinu komponentů pomocí distribuce Pythonu Anaconda.

3.3.1 Python

Algoritmy použité v této práci používají pro svůj běh jazyk Python. Python byl ve verzi 3.7 instalován prostřednictvím distribuce Anaconda.

Dále bylo nutné doplnit instalaci Pythonu o knihovnu OpenCV. Jedná se o knihovnu pro strojové vidění, pomocí níž lze pracovat se statickým obrazovým

či video vstupem. Tuto knihovnu bylo možné doplnit do nainstalované verze Pythonu prostřednictvím distribuce Anaconda.

3.3.2 CUDA, cuDNN

CUDA je architektura pro paralelní výpočty vyvinutá společností NVIDIA. Díky této technologii je možné využít mnoho výpočetních jader v grafickém procesoru k provádění výpočtů. Pro její využití je nezbytné použít grafickou kartu, která tuto technologii podporuje. Jednotlivé grafické karty mají různé výpočetní možnosti od méně výkonných podporovaných karet, které začínají na výpočetním indexu 2.0, až po moderní grafické karty jako například NVIDIA TITAN RTX, která dosahuje výpočetního indexu 7.5. Čím je výpočetní index vyšší, tím rychleji probíhá trénování a detekce sítě. Použití CUDA nemá však žádný dopad na výslednou přesnost a není nutnou součástí pro fungování detektorů, jedná se pouze o urychlení náročných výpočtů.

Pro tuto práci byla použita grafická karta s výpočetním indexem 5.0 a verze CUDA 10.1 a její knihovna pro práci s hlubokými neuronovými sítěmi cuDNN (CUDA Deep Neural Network library) verze 7.5 .

Nejprve bylo zapotřebí stáhnout a nainstalovat CUDA Toolkit a knihovnu cuDNN. Je důležité zvolit takovou verzi Cuda Toolkit, aby byla kompatibilní s verzí jazyka Python a s knihovnou cuDNN. Také je nutné nainstalovat aktuální ovladače grafické karty kompatibilní s verzí CUDA.

3.3.3 PyTorch

Použitá implementace detektoru YOLOv3 funguje prostřednictvím frameworku PyTorch. PyTorch lze nainstalovat ve dvou verzích. První verzi lze využít pouze pro výpočty prováděné pomocí procesoru. Druhou verzi je možné použít pro výpočty prováděné pomocí grafického procesoru (pokud daná grafická karta podporuje technologii CUDA). PyTorch bylo možné doplnit do nainstalované verze Pythonu prostřednictvím distribuce Anaconda. Dále byl nainstalován

balíček Torchvision, který obsahuje modelové architektury a algoritmy pro transformaci obrazu pro počítačové vidění.

3.3.4 Tensorflow, Tensorboard

Pro implementaci Faster R-CNN byl použit framework Tensorflow. Tensorflow byl do nainstalované verze Pythonu nainstalován prostřednictvím distribuce Anaconda. Spolu s Tensorflow byl nainstalován i doplněk Tensorboard. Jedná se o vizualizační nástroj pro práci s Tensorflow. Díky tomuto nástroji lze sledovat průběh tréninku prostřednictvím grafů. V grafickém prostředí Tensorboard lze vidět čas trénování, počet kroků, průběh i odchylka od validačního datasetu. Lze tak přehledně určit dobu pro skončení trénování. I tento balíček je dostupný prostřednictvím distribuce Anaconda.

3.4 PyTorch a YOLOv3

Jako zástupce „one stage“ detektorů byl vybrán detektor YOLOv3, implementovaný pomocí frameworku PyTorch. Bylo otestováno několik různých dostupných verzí této kombinace. Bylo však nutné nalézt takovou kombinaci, aby obsahovala i trénování nových dat. Většina dostupných implementací řeší pouze detekování již natrénovaných datasetů pomocí již natrénovaných vah, ale neimplementují trénování nových dat, nebo je pevně zabudováno do kódu pro trénování jednoho určitého datasetu.

Pro test byl vybrán a zprovozněn repositář „PyTorch 0.4 yolov3“, dostupný na stránce `https://github.com/andy-yun/pytorch-0.4-yolov3`. Tato implementace umožňuje jak detekci, tak i trénování nových dat. Nebyla však nalezena taková verze, jež by zároveň implementovala i evaluaci natrénované sítě pro otestování výsledné přesnosti, např. pomocí nástroje cocoeval.

3.4.1 Konfigurační soubory a data

Před samotným trénováním bylo nutné převést data do formátu, který je vyžadován implementací detektoru, a také nastavit konfigurační soubory pro trénování a běh sítě. Zároveň bylo nutné ze stránek repositáře stáhnout počáteční váhy sítě, na kterých probíhalo trénování.

Použitá implementace detektoru YOLOv3 pomocí frameworku PyTorch vyžaduje specifický formát dat. Data, která byla k dispozici, byla ve formátu COCO. Tato data musela být převedena na takzvaný YOLOv3 formát, který tato implementace používá. Jedná se o textové soubory, ve kterých jsou popsány objekty vyskytující se na daném obrázku. Dále byl vygenerován další textový soubor, který obsahuje seznam obrázků v daném setu, tzn. pro trénovací, validační i testovací dataset zvlášť. V tomto textovém souboru jsou uloženy názvy obrázků. Tento soubor je v kódu používán jako seznam obrázků a labelů.

Dalším potřebným souborem byl textový soubor, ve kterém se nachází cesta k trénovacímu a validačnímu setu, počet detekovatelných tříd a název složky pro výstup. Jednotlivé třídy určené k detekci jsou popsány zvlášť v textovém souboru.

Poslední ze souborů potřebných pro běh sítě byl samotný konfigurační soubor pro nastavení sítě. První nutnou úpravou tohoto souboru pro daný dataset bylo přepsání řádku „classes“, který obsahuje počet detekovatelných tříd. Dalším řádkem, který bylo nutné upravit, byl řádek „filters“. Hodnota tohoto řádku byla doplněna podle vzorce „(počet tříd + 5)*3“. Další hodnoty bylo možné upravovat bez závislosti na daném datasetu, viz tab. 3.1. Po nastavení těchto hodnot bylo zahájeno trénování sítě.

Použitý detektor YOLOv3, implementovaný pomocí frameworku PyTorch, i původní implementace YOLOv3 používají pro práci s daty svůj vlastní formát. Tento formát je však velmi nepřehledný, obsahuje řadu implementačních chyb a na některých místech se vyskytují redundance; tyto skutečnosti vedou k neefektivitě kódu.

Hyperparametr	YOLOv3	YOLOv3-tiny
batch size	4	16
subdivisions	1	2
width	416	416
height	416	416
momentum	0.9	0.9
decay	0.0005	0.0005
burn in	1000	1000
learning rate	0.001	0.001
anchors	9	6
conv. layers	75	13

Tabulka 3.1: Přehled nastavených hyperparametrů pro trénování detektoru YOLOv3. (zdroj: autor)

3.4.2 Vlastní trénování

Jelikož byly testovány dva datasety, musely být všechny konfigurační soubory vytvořeny dvakrát – každý z nich v úpravě pro jeden dataset. Pro započetí trénování sítě na nových datech byly použity výše zmíněné stažené základní váhy. Tyto přednastavené váhy byly dostupné prostřednictvím repositáře. Trénování probíhalo ve dvou variantách. Obě tyto varianty mají své vlastní základní váhy pro trénování. Trénována byla nejprve odlehčená a rychlejší verze YOLOv3-tiny, která byla méně náročná na paměť počítače i na grafickou paměť, a následně plnohodnotná varianta YOLOv3. Varianta YOLOv3-tiny sloužila také ke zjištění vhodných parametrů pro trénování sítě a seznámení se s jeho průběhem. Při použití grafické karty s pamětí 4 GB tak bylo možné simultánně trénovat až šestnáct obrázků načtených do paměti, což mělo za následek kratší čas trénování.

Nejprve byla trénována verze YOLOv3-tiny na datasetu FlickrLogos-32 a následně i na datasetu TopLogo-10. V této implementaci byl přítomen kód, který po každé desáté epoše testoval přesnost sítě na validačních datech. Jednalo se však o jednoduchou vnitřní implementaci, výsledek byl tedy pouze orientační a nebyl vhodný pro přesné posuzování přesnosti sítě. Pro spuštění trénování sítě bylo nutné zadat jako parametr cestu k textovému souboru popisující cestu k tes-

tovaným datům (popsán výše), dále cestu ke konfiguračnímu souboru a cestu k vahám sítě.

Při trénování sítě byly vypisovány informace o dění v síti, jako např. data ze ztrátových funkcí, popisující odchylku boxů detekovaného objektu, odchylku pro třídy a celkovou odchylku, kolikátá epocha byla aktuálně trénována, počet již prošlých obrázků a počet obrázků za sekundu. Samotné trénování probíhalo až do doby ustálení celkové odchylky na nejnižší možné hranici. Pokud by došlo k překročení této hranice, odchylka by se začala opět zvětšovat a docházelo by k přeučení sítě.

Trénování verze YOLOv3-tiny probíhalo rychlostí cca 100 epoch za 2 hodiny. Trénování datasetu FlickrLogos-32 pokračovalo až do epochy 1160. Po přesažení této hranice už byl pokles celkové odchylky minimální a nemělo smyl dále pokračovat. Při trénování datasetu TopLogo-10 bylo dosaženo epochy 720, po jejímž přesažení opět došlo k ustálení celkové odchylky.

Vzhledem k paměťové náročnosti na grafickou kartu bylo u trénování sítě s verzí YOLOv3 nutné upravit hodnoty konfiguračního souboru tak, že do paměti mohly být načítány pouze čtyři obrázky. Z tohoto důvodu byla snížena rychlost trénování, která však byla kompenzována rychlejším nalezením optimálního řešení. Při trénování datasetu FlickrLogos-32 pomocí verze YOLOv3 trvalo 100 epoch přibližně 7 hodin. Při trénování se však hodnota celkové odchylky ustálila už při trénování epochy 320. Při trénování datasetu TopLogo-10 bylo ustálené hodnoty celkové odchylky dosaženo již při trénování epochy 160.

Při trénování byly nové váhy ukládány vždy po pěti epochách. Pokud však došlo k problémům např. s nedostatečnou pamětí, program byl ukončen s chybovou hláškou a bylo nutné pokračovat od poslední uložené hodnoty. I v trénovacím algoritmu se vyskytovaly různé části, ve kterých se s daty pracovalo neefektivně a docházelo k redundanci dat i kódu. Zároveň zde byly obsaženy i některé části kódu, které postrádaly jakýkoli význam a pouze zabíraly řádky.

3.4.3 Testování

Výsledkem trénování byly natrénované váhy sítě pro jednotlivé datasety a verze YOLOv3 (celkem čtyři). Použitá implementace detektoru obsahovala kód na detekování objektů v obraze pomocí natrénovaných vah, avšak pouze pro jeden obrázek. Pro vizuální kontrolu detekce byla vytvořena vlastní varianta detekce, která procházela obrázky z testovacího setu a prováděla na nich detekci. Tato část byla vytvořena pomocí zacyklení a upravení detekce pro jeden obrázek, který tato implementace detektoru obsahovala. Implementace detekce však stejně jako další části této implementace YOLOv3 opět vykazovala určité segmenty špatné či neefektivní práce s daty.

Obrázek 3.1: Příklad detekce pomocí detekrotu YOLOv3 na jednom z obrázků datasetu FlickrLogos-32. (zdroj: autor)

Po vizuální stránce byly detekce jak pro dataset FlickrLogos-32, tak pro dataset TopLogo-10 uspokojivé. V některých případech však detektor na jednom obrázku objekt našel, ale obdobný objekt na jiném obrázku, přestože měl stejnou velikost a natočení, nenašel. Jedná se pouze o subjektivní pocit z detekce. Pro evaluaci přesnosti výsledných vah sítě bylo potřeba použít speciální nástroj.

3.4.4 Evaluace a přesnost

Implementace detektoru YOLOv3, ani žádná jiná, používající framework PyTorch, neobsahovala evaluaci, proto bylo nutné tuto část doimplementovat. Pro evaluaci dat byla použita knihovna cocoeval. Byl vytvořen kód, který byl složen z cyklu a částí detekce této implementace detektoru, a data z detektoru ukládal do JSON souboru. Tento soubor obsahoval informace o tom, zda a v jakém obrázku se detekované objekty nachází, o pozici detekovaných objektů a o odhadované přesnosti. Tento soubor byl použit spolu s testovacími daty a váhami sítě jako vstup pro knihovnu cocoeval, která následně vypočítala přesnost natrénovaných vah na testovacím setu.

Verze	Dataset	Čas det.	Trénováno	$AP_{0.5}$	AP_S	AP_M	AP_L	recall
YOLOv3-tiny	FlickrLogos-32	0.16 s	23 hod.	15.3 %	0 %	0.7 %	9.5 %	7.6 %
YOLOv3-tiny	TopLogo-10	0.16 s	14 hod.	19.3 %	4 %	7.3 %	7.4 %	8.6 %
YOLOv3	FlickrLogos-32	0.7 s	22 hod.	45 %	0 %	3.9 %	31.5 %	26 %
YOLOv3	TopLogo-10	0.7 s	11 hod.	59 %	14 %	34 %	26 %	35 %

Tabulka 3.2: Porovnání přesností detektoru YOLOv3. (zdroj: autor)

Takto byly evaluovány natrénované váhy sítě pro jednotlivé verze detektoru a datasetů. Tabulka 3.2 obsahuje přehled hodnot dosažených jednotlivými natrénovanými váhami. Nejvyšší přesnost, která byla na datasetu FlickrLogos-32 dosažena, byla 45 %, a to v čase 22 hodin. Na datasetu TopLogo-10 byla nejvyšší dosažená přesnost 59 % v čase 11 hodin. Avšak i při dosažené přesnosti byl recall téměř poloviční, a to konkrétně pro dataset FlickrLogos-32 26 % a pro dataset TopLogo-10 35 %. Z tabulky 3.2 je také patrné, že verze YOLOv3-tiny nedosahuje takové přesnosti jako jako verze YOLOv3, což kompenzuje rychlostí trénování a detekce. Recall u verze YOLOv3-tiny nedosahuje u obou testovaných datasetů ani 10%. Zároveň je z tabulky 3.2 patrné, že za nižší přesnost u datasetu FlickrLogos-32 může přesnost detekce malých a středních objektů, která je oproti datasetu TopLogo-10 téměř nulová.

3.5 Tensorflow a Faster R-CNN

Jako zástupce „two stage“ detektorů byl vybrán detektor R-CNN, a to konkrétně jeho nejrychlejší varianta Faster R-CNN, implementovaná pomocí frameworku Tensorflow. Tato verze obsahuje jak trénování na vlastních datech, tak evaluaci natrénované sítě pomocí knihovny cocoeval.

Pro testování byl zvolen repositář „TensorFlow Object Detection API Tutorial Train Multiple Objects Windows 10“, dostupný na stránce `https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10`. Tento repositář je aktivně opravován, vylepšován a poskytuje kvalitní dokumentaci a rady pro zprovoznění.

3.5.1 Konfigurační soubory a data

I v případě detektoru Faster R-CNN bylo nutné upravit datasety na formát, který tento detektor podporuje. Díky použití Tensorflow bylo možné využít formát TFRecords, který má díky své binární podobě výhodu v rychlosti a je pro tento framework optimalizován. Samotný repositář obsahuje nástroje na převod různých formátů na formát TFRecords.

Byl vytvořen JSON soubor, popisující detekovatelné třídy. Obsahem tohoto souboru byl vždy název a index dané třídy.

V konfiguračním souboru pro běh sítě byl upraven počet tříd pro detekci, cesta k souboru, který ukládá průběžná data z trénování, cesty k souborům datasetů a cesta k výše zmíněnému JSON souboru, který obsahuje seznam detekovatelných tříd. Zbylé parametry, nezávislé na daném datasetu, byly nastaveny podle tabulky 3.3.

Práce s těmito daty byla jednoduchá a intuitivní, nebylo potřeba zdlouhavé nastavování a provazování více souborů, jako tomu bylo v případě detektoru YOLOv3. Data jsou v přehledném formátu a neobsahují zbytečné redundance.

Hyperparametr	Faster R-CNN
min dimension	600
max dimension	1024
initial learning rate	0.0002
learning rate for step >900000	0.00002
learning rate for step >1200000	0.000002
momentum optimizer value	0.9

Tabulka 3.3: Přehled nastavených hyperparametrů pro trénování detektoru Faster R-CNN. (zdroj: autor)

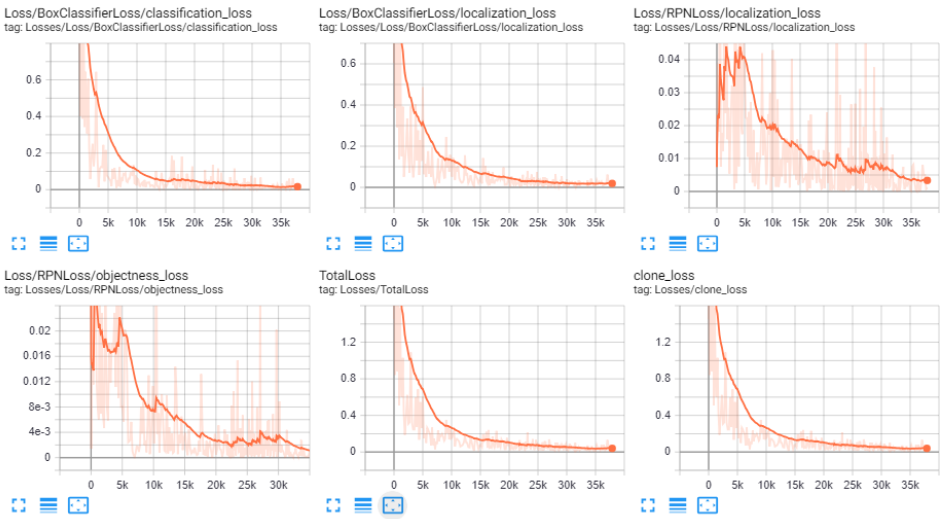
3.5.2 Vlastní trénování

Stejně jako u detektoru YOLOv3, i u detektoru Faster R-CNN byly konfigurační soubory vytvořeny dvakrát, pro každý dataset zvlášť. Všechny další potřebné soubory byly staženy spolu s repositářem.

Pro započetí trénování bylo nutné vyplnit parametry, kterými jsou výstupní soubor, logující průběh trénování, cesta ke složce, do které jsou ukládána průběžná data z trénování, a cesta ke konfiguračnímu souboru. Pomocí souborů generovaných v průběhu trénování je možné sledovat jeho průběh pomocí grafického prostředí Tensorboard. Lze tedy ve vhodný čas trénování zastavit, aby nedošlo k přetrénování. V průběhu trénování jsou vypisována data popisující o jaký krok trénování se jedná, čas nutný pro trénování jednoho kroku a výstup ze ztrátové funkce, který udává odchylku trénovaných dat od nejlepšího možného výsledku.

Prvním trénovaným byl dataset FlickrLogos-32. Trénování tohoto datasetu probíhalo rychlostí přibližně dva kroky za sekundu. Při dosažení kroku 39 415 se celková odchylka ustálila a trénování bylo přerušeno, a to v čase 7 hodin. Trénování datasetu TopLogo-10 probíhalo stejnou rychlostí jako v případě prvního datasetu. Ustálení celkové odchylky bylo dosaženo v kroku 38 105 v totožném čase.

Pokud při trénování došlo k situaci nedostatku grafické paměti, detektor tento stav dokázal obejít opakovaným procházením daného kroku až do doby jeho úspěšného natrénování. Díky tomu bylo možné nechat síť při trénování bez do-



Obrázek 3.2: Průběh trénování datasetu FlickrLogos-32 v grafickém prostředí Tensorboard. (zdroj: autor)

zoru. Průběh trénování pak bylo možné sledovat v grafickém prostředí Tensorboard, kde bylo možné zjistit, v jakém bodě byla síť nejblíže optimálnímu výsledku.

3.5.3 Testování

V průběhu trénování byly ukládány tzv. checkpointy. Z těchto checkpointů byl podle grafu z grafického prostředí Tensorboard vybrán takový výsledek, který se nejvíce blížil optimálnímu řešení. Tento checkpoint byl převeden pomocí speciálního nástroje, obsaženého v repositáři, na graf, který zastupoval nastavení sítě pro detekci. Poté bylo možné provádět detekci pomocí připravených nástrojů.

Součástí repositáře byly algoritmy pro detekci objektů v jednom obrázku, ve videu a v kamerovém vstupu. Stejně jako v případě detektoru YOLOv3, i pro Faster R-CNN bylo doimplementováno řešení pro detekci více obrázků po sobě. To bylo vytvořeno pomocí cyklu a zmíněného algoritmu pro detekci v jednom obrázku. Výsledné řešení procházelo všechny obrázky v testovacím setu a na nich provádělo detekci. Při vizuální kontrole obrázků, na kterých byla provedena detekce, vypadaly výsledky obdobně jako v případě detektoru YOLOv3.

Obrázek 3.3: Příklad detekce pomocí detekrotu Faster R-CNN na jednom z obrázků datasetu FlickrLogos-32. (zdroj: autor)

3.5.4 Evaluace a přesnost

Použitá implementace detektoru Faster R-CNN obsahovala i algoritmus pro evaluaci výsledků pomocí knihovny cocoeval. Stejně jako v případě trénování, i zde bylo nutné jako parametr do funkce pro evaluaci předat cestu k souboru se zalogovanými parametry z trénování, cestu ke konfiguračnímu souboru a cestu ke složce pro trénink a pro výstup.

Verze	Dataset	Čas det.	Trénováno	$AP_{0.5}$	AP_S	AP_M	AP_L	recall
Faster R-CNN	FlickrLogos-32	0.4 s	7 hod.	60 %	8.3 %	12.9 %	57.4 %	55 %
Faster R-CNN	TopLogo-10	0.4 s	7 hod.	67 %	11.5 %	44.6 %	44.3 %	55.7 %

Tabulka 3.4: Porovnání přesností detektoru Faster R-CNN. (zdroj: autor)

Po skončení evaluace pro oba datasety byla výsledná nejlepší přesnost pro dataset FlickrLogos-32 60 % – síť byla natrénována v čase 7 hodin. Pro dataset TopLogo-10 byla nejvyšší přesnost 67 % dosažena v čase přibližně 7 hodin. Hodnota recall se u obou trénovaných datasetů blížila 55 %. Z tabulky 3.4 je také patrné, že detektor měl potíže detekovat menší (v případě datasetu FlickrLogos-32 i středně velké) objekty, u nichž byla přesnost detekcí velmi malá.

3.6 Vyhodnocení

3.6.1 Porovnání přesnosti a výsledků

V porovnání přesnosti detektoru YOLOv3 a detektoru Faster R-CNN dosahoval detektor Faster R-CNN na obou testovaných datasetech větších přesností než obě verze detektoru YOLOv3, konkrétně 60 % na datasetu FlickrLogos-32 a 67 % na datasetu TopLogo-10. Přesnost detektoru YOLOv3 v plnohodnotné verzi byla u datasetu TopLogo-10 o 8 % a u datasetu FlickrLogos-32 o 15 % menší než u detektoru Faster R-CNN. Zároveň také detektor Faster R-CNN dosahoval nejvyšších hodnot recall, a to přibližně 55 % u obou datasetů. Hodnoty recall byly u detekoru YOLOv3 na datasetu FLickrlogo 26 % a na datasetu TopLogo-10 35 %. Detektor YOLOv3 tak dostahoval skoro poloviční hodnot recall, tudíž i když přesnost byla jen o max. 15% menší než u detektoru Faster R-CNN, povedlo se mu detekovat pouze poloviční množství objektů. Přesnost detektoru YOLOv3 ve verzi tiny byla nejmenší z testované sady a oproti jeho plnohodnotné verzi pouze třetinová (15.3 % na datasetu FlickrLogos-32 a 19.3 % na datasetu TopLogo-10) a recall byl u obou datasetů nejmenší z celé testované sady. Tato verze ale dosahovala nejvyšších rychlostí při detekci, a to konkrétně jeden obrázek za 0.16 sekund, oproti detektoru Faster R-CNN, kde detekce jednoho obrázku trvala 0.4 sekund a u detektoru YOLOv3, dokonce 0.7 sekund. Přestože by měl detektor YOLOv3 podle dokumentace být rychlejší než detektor Faster R-CNN, trénování na datasetu TopLogo-10 probíhalo o 4 hodiny déle; na datasetu FlickrLogos-32 trvalo trénování až o 15 hodin déle. Při detekci dosahoval detektor Faster R-CNN skoro poloviční časů potřebných pro detekci objektů v obraze, než detektor YOLOv3. Detekce jednoho obrazu byla u detektoru Faster R-CNN o 0.3 sekundy rychlejší než u detektoru YOLOv3.

Detektor Faster R-CNN, implementovaný pomocí frameworku Tensorflow, byl na testovaných datasetech a na použitém hardwaru přesnější a rychlejší než detektor YOLOv3, implementovaný pomocí frameworku PyTorch.

3.6.2 YOLOv3 vs Faster R-CNN

Mezi použitými detektory je patrný rozdíl v implementaci. Detektor YOLOv3 obsahuje mnoho chyb v oblasti práce s daty a s kódem obecně. Oproti tomu detektor Faster R-CNN je naimplementován mnohem profesionálněji. Konkrétní implementace YOLOv3, pomocí frameworku PyTorch, která byla použita pro tuto práci, je pouze upravenou verzí originální implementace YOLOv3, tudíž chyby a problémy, které se týkají použité implementace, jsou totožné jako u originální verze. Díky těmto chybám a nedostatkům je obtížné se v kódu zorientovat, což ztěžuje práci při ladění či upravování kódu. Práce s daty v detektoru Faster R-CNN je mnohem efektivnější, přehlednější a pohodlnější než u jeho konkurenta.

Oproti tomu velkou výhodou detektoru YOLOv3 je díky použití frameworku PyTorch a jeho integrovanosti do jazyka Python právě transparentnost. Je tedy snadné pracovat s kódem a při ladění sledovat, co se při jeho běhu děje. Po zorientování v kódu a jeho mírném upravení je práce s tímto detektorem velice snadná. Přestože je zpočátku orientace v systému fungování detektoru Faster R-CNN obtížnější, ve výsledku je práce s tímto detektorem pohodlnější. Největší vliv na uživatelskou přívětivost má grafické prostředí Tensorboard, jehož prostřednictvím je možné sledovat kompletní průběh trénování.

Práce byla znepříjemňována občasnou nestabilitou knihovny cuDNN, která se však projevovala pouze při práci s detektorem YOLOv3 a frameworkem PyTorch. Za účelem opětovného zprovoznění knihovny bylo nutné občasné restartování počítače. Pro tuto chybu nebyla dosud vydána žádná oprava. Jestli za neprofesionální řešení implementace YOLOv3 může to, že se nachází stále ve vývoji, nebo nedbalost autora, není známo. V implementaci detektoru ovšem existují části, které by mohly být upraveny pro lepší přesnost, příp. i rychlost tohoto detektoru.

3.6.3 Shrnutí

Přestože detektor YOLOv3 po teoretické stránce vypadá přesnější a rychlejší než detektor Faster R-CNN, výsledek testování těchto datasetů prokázal opak. Detektor Faster R-CNN byl po otestování přesnější i rychlejší. Z těchto dvou testovaných detektorů by tak byl Faster R-CNN vhodnější variantou pro jeho použití v rámci detekce log v obraze. Tento výsledek je pravděpodobně zapříčiněn implementací detektoru YOLOv3, která obsahuje implementační chyby, a to především v práci s daty. Při vhodné úpravě implementace a optimalizaci tohoto detektoru by mohlo dojít k zlepšení jeho přesnosti a rychlosti. Oba testované detektory dosahovaly na použitých datasetech menších přesností než některé stávající systémy pro detekci log v obraze, zmíněné v této práci.

4 Závěr

Cíle této práce, kterými bylo seznámení se s problematikou neuronových sítí a hlubokého učení, vytvoření rešerše, pojednávající o využití neuronových sítí pro detekci objektů a log v obraze, a otestování a vyhodnocení vybraných modelů pro detekci log v obraze na vybraných testovacích databází, byly naplněny.

Po teoretickém nastínění problematiky neuronových sítí v oblasti detekce log v obraze a vytvoření rešerše k tomuto tématu byly načerpané poznatky využity pro porozumění modelům zabývajících se detekcí log a pro jejich následné otestování. Byly vybrány dva modely detektorů, detektor YOLOv3, implementovaný pomocí frameworku PyTorch, a detektor Faster R-CNN, implementovaný pomocí frameworku Tensorflow. Tyto vybrané modely byly otestovány na dvou vybraných datasetech, a to konkrétně na datasetu FlickrLogos-32 a datasetu TopLogo-10.

Výsledkem tohoto testování bylo zjištění, že detektor Faster R-CNN je díky vyšší přesnosti a také rychlosti pro použití v rámci detekování log v obraze lepší volbou než detektor YOLOv3. Horší přesnost a především rychlost detektoru YOLOv3 může být zapříčiněna implementačními chybami, které se v něm nacházejí. Po vhodné úpravě by však mohly být tyto nedostatky opraveny. Bylo také zjištěno, že existují odborné publikace a systémy, které úspěšně testují vylepšení přesnosti modelů pro detekci log v obraze. Poznatky z těchto prací by se daly využít pro vylepšení testovaných modelů. Tato skutečnost tak otevírá prostor pro případné budoucí bádání.

V dnešní době jsou technologie využívající neuronové sítě pro detekci log v obraze a také detekci objektů v obraze obecně stále více používány v různých

oblastech každodenního života. Je tak nezbytné tyto technologie zkoumat a přibližovat se požadavkům větší přesnosti a rychlosti, které jsou na tyto systémy v současnosti kladeny. Tato práce se pokusila přispět k poznání v této oblasti a k dalšímu možnému posunutí hranic těchto technologií.

Použité zdroje

[1] ŠMÍD, Petr. Jak na vizuální kontrolu pomocí umělé inteligence a hlubokého učení. *Volty* [online]. [cit. 2019-04-25]. Dostupné z: https://www.volty.cz/2018/04/14/vizualni-kontrolu-pomoci-umele-inteligence-hlubokeho-uceni/

[2] MASON, Shaun. Microscope uses artificial intelligence to find cancer cells more efficiently. *UCLA* [online]. [cit. 2019-04-25]. Dostupné z: http://newsroom.ucla.edu/releases/microscope-uses-artificial-intelligence-to-find-cancer-cells-more-efficiently

[3] PIMKOTE, Pichitchai a Thanapat KANGKACHIT. Classification of alcohol brand logos using convolutional neural networks. In: *2018 International Conference on Digital Arts, Media and Technology (ICDAMT)* [online]. IEEE, 2018 [cit. 2019-04-25]. DOI: 10.1109/ICDAMT.2018.8376510. ISBN 978-1-5386-0573-8. Dostupné z: https://ieeexplore.ieee.org/document/8376510/

[4] Neuron. *Wikipedia, The Free Encyclopedia.* [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Neuron

[5] LAGANDULA, Akshay Chandra. McCulloch-Pitts Neuron-Mankind-s First Mathematical Model Of A Biological Neuron. *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

[6] ROMBERG, Stefan, Lluis Garcia PUEYO, Rainer LIENHART a Roelof VAN ZWOL. Scalable logo recognition in real-world images. In: *Proceedings of the 1st ACM International Conference on Multimedia Retrieval - ICMR '11* [online]. New York, New York, USA: ACM Press, 2011 [cit. 2019-04-25]. DOI: 10.1145/1991996.1992021. ISBN 9781450303361. Dostupné z: http://portal.acm.org/citation.cfm?doid=1991996.1992021

[7] ROMBERG, Stefan a Rainer LIENHART. Bundle min-hashing for logo recognition. In: *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval - ICMR '13* [online]. New York, New York,

USA: ACM Press, 2013 [cit. 2019-04-25]. DOI: 10.1145/2461466.2461486. ISBN 9781450320337. Dostupné z: http://dl.acm.org/citation.cfm?doid=2461466.2461486

[8] SU, Hang, Xiatian ZHU a Shaogang GONG. Deep Learning Logo Detection with Data Expansion by Synthesising Context. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)* [online]. IEEE, 2017 [cit. 2019-04-25]. DOI: 10.1109/WACV.2017.65. ISBN 978-1-5090-4822-9. Dostupn\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{e\global\mathchardef\accent@spacefactor\spacefactor}\accent19e\egroup\spacefactor\accent@spacefactorz:http://ieeexplore.ieee.org/document/7926648/

[9] EGGERT, Christian, Anton WINSCHEL a Rainer LIENHART. On the Benefit of Synthetic Data for Company Logo Detection. In: *Proceedings of the 23rd ACM international conference on Multimedia - MM '15* [online]. New York, New York, USA: ACM Press, 2015 [cit. 2019-04-25]. DOI: 10.1145/2733373.2806407. ISBN 9781450334594. Dostupné z: http://dl.acm.org/citation.cfm?doid=2733373.2806407

[10] MONTSERRAT, Daniel Mas, Qian LIN, Jan ALLEBACH a Edward J. DELP. Logo detection and recognition with synthetic images. *Electronic Imaging* [online]. 2018 [cit. 2019-04-25]. DOI: 10.2352/ISSN.2470-1173.2018.10.IMAWM-337. ISSN 2470-1173. Dostupné z: http://www.ingentaconnect.com/content/10.2352/ISSN.2470-1173.2018.10.IMAWM-337

[11] REDMON, Joseph a Ali FARHADI. YOLO: Real-Time Object Detection. *Pjreddie* [online]. [cit. 2019-04-25]. Dostupné z: https://pjreddie.com/darknet/yolo/

[12] CHABLANI, Manish. YOLO—You only look once, real time object detection explained. *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006

[13] GANDHI, Rohith. R-CNN, Fast R-CNN, Faster R-CNN, YOLO—Object Detection Algorithms. *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[14] YEGULALP, Serdar. What is TensorFlow? The machine learning library explained. *Infoworld* [online]. [cit. 2019-04-25]. Dostupné z: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

[15] SHETTY, Sunith. What is PyTorch and how does it work?. *Packt Hub* [online]. [cit. 2019-04-25]. Dostupné z: https://hub.packtpub.com/what-is-pytorch-and-how-does-it-work/

[16] Tensorflow vs Pytorch. *Educba* [online]. [cit. 2019-04-25]. Dostupné z: https://www.educba.com/tensorflow-vs-pytorch/

[17] ROSENZVAIG, Eitan. The battle: Tensorflow vs Pytorch. *Developereconomics* [online]. [cit. 2019-04-25]. Dostupné z: https://www.developereconomics.com/tensorflow-vs-pytorch

[18] SHAH, Tarang. About Train, Validation and Test Sets in Machine Learning. *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[19] Convolutional neural network. *Wikipedia, The Free Encyclopedia*. [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Convolutional_neural_network

[20] Goodfellow, I., Bengio, Y., Courville, A. *Deep learning.*. MIT Press, 2016. ISBN: 978-0262035613

[21] Bishop, C. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 978-038731073

[22] RUSSELL, John. Google-s AlphaGo AI wins three-match series against the world-s best Go player. *TechCrunch* [online]. [cit. 2019-04-25]. Dostupné z: https://techcrunch.com/2017/05/24/alphago-beats-planets-best-human-go-player-ke-jie

[23] Convolutional Neural Networks. *STANFORD* [online]. [cit. 2019-04-25]. Dostupné z: http://cs231n.github.io/convolutional-networks/

[24] VONDRÁK, Ivo. Neuronové sítě. *Vysoká škola báňská, Technická univerzita Ostrava* [online]. [cit. 2019-04-25]. Dostupné z: http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf

[25] SHIFFMAN, Daniel. Chapter 10. Neural Networks. *Nature of code* [online]. [cit. 2019-04-25]. Dostupné z: https://natureofcode.com/book/chapter-10-neural-networks/

[26] Artificial neural network. *Wikipedia, The Free Encyclopedia* [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Artificial_neural_network

[27] Perceptron. *Wikipedia, The Free Encyclopedia* [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Perceptron

[28] RADOVÁ, Vlasta. Typy umělých neuronových sítí. *Katedra kybernetiky, Západočeská univerzita v Plzni* [online]. [cit. 2019-04-25]. Dostupné z: http://www.kky.zcu.cz/uploads/courses/nses/2_Typy_umelych_neuronovych_siti_a_faze_jejich_cinnosti.pdf

[29] Artificial Intelligence - Neural Networks. *Tutorialspoint* [online]. [cit. 2019-04-25]. Dostupné z: https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm

[30] Operant conditioning. *Wikipedia, The Free Encyclopedia.* [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Operant_conditioning

[31] END-TO-END DEEP LEARNING PLATFORM. *Pytorch* [online]. [cit. 2019-04-25]. Dostupné z: https://pytorch.org/features

[32] Introduction to TensorFlow. *Tensorflow* [online]. [cit. 2019-04-25]. Dostupné z: https://www.tensorflow.org/guide/low_level_intro

[33] TSANG, Sik-Ho. YOLOv2 & YOLO9000—You Only Look Once (Object Detection). *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65

[34] IRIONDO, Roberto. Limitations of Deep Learning in AI Research. *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/limitations-of-deep-learning-in-ai-research-5eed166a4205

[35] BONNER, Anne. Intro to Deep Learning. *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/intro-to-deep-learning-c025efd92535

[36] JAIN, Akshat. A Gentle Introduction to Deep Learning. *Towards Data Science* [online]. [cit. 2019-04-25]. Dostupné z: https://towardsdatascience.com/a-gentle-introduction-to-deep-learning-part-1-introduction-43eb199b0b9

[37] RAY, Amit. 7 Limitations of Deep Learning Algorithms of AI. *Amitray* [online]. [cit. 2019-04-25]. Dostupné z: https://amitray.com/7-limitations-of-deep-learning-algorithms-of-ai/

[38] MARIE DE FRANCE, WARNKE, Karl, ed. *Vier Lais der Marie de France: nach der Handschrift des Mus. Brit. Harl. 978 mit Einleitung und Glossar*. Halle: Max Niemeyer, 1925. Sammlung romanischer Übungstexte, 2.

[39] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge, Massachusetts: The MIT Press, [2016]. ISBN 978-0262035613.

[40] TÜZKÖ, Andras, Christian HERRMANN, Daniel MANGER a Jürgen BEYERER. Open Set Logo Detection and Retrieval. In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* [online]. SCITEPRESS - Science and Technology Publications, 2018 [cit. 2019-04-25]. DOI: 10.5220/0006614602840292. ISBN 978-989-758-290-5. Dostupné z: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006614602840292

[41] EGGERT, Christian, Dan ZECHA, Stephan BREHM a Rainer LIENHART. Improving Small Object Proposals for Company Logo Detection. In: *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval - ICMR '17* [online]. New York, New York, USA: ACM Press, 2017 [cit. 2019-04-25]. DOI: 10.1145/3078971.3078990. ISBN 9781450347013. Dostupné z: http://dl.acm.org/citation.cfm?doid=3078971.3078990

[42] MONTSERRAT, Daniel Mas, Qian LIN, Jan ALLEBACH a EdwardJ. DELP. Training Object Detection And Recognition CNN Models Using Data Augmentation. *Electronic Imaging* [online]. 2017 [cit. 2019-04-25]. DOI: 10.2352/ISSN.2470-1173.2017.10.IMAWM-163. ISSN 2470-1173. Dostupné z: http://www.ingentaconnect.com/content/10.2352/ISSN.2470-1173.2017.10.IMAWM-163

A Obsah přiloženého CD

- Text bakalářské práce
  - `bakalarska_prace_2019_Zbynek_Novak.pdf`
  - `kopie_zadani_bakalarske_prace_2019_Zbynek_Novak.pdf`
- Obrázky
- Zdrojové kódy