VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE REGISTRAČNÍ ZNAČKY VOZIDLA VE VIDEU

DETECTION OF VEHICLE LICENSE PLATES IN VIDEO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ LÍBAL**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Líbal Tomáš**
Program: Informační technologie
Název: **Detekce registrační značky vozidla ve videu**
Detection of Vehicle License Plates in Video
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte problematiku detekce a rozpoznání registračních značek vozidel, zaměřte se na moderní přístupy založené na konvolučních neuronových sítích.
2. Vyhledejte dostupné datové sady registračních značek, podle potřeby je doplňte.
3. Experimentujte s relevantními algoritmy detekce registračních značek založenými na konvolučních neuronových sítích.
4. Provádějte experimenty na vhodných datových sadách a iterativně vylepšujte zkoumané algoritmy.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Jakub Špaňhel et al.: Holistic Recognition of Low Quality License Plates by CNN using Track Annotated Data, IWT4S-AVSS 2017
- Jakub Sochor et al.: BrnoCompSpeed: Review of Traffic Camera Calibration and A Comprehensive Dataset for Monocular Speed Measurement, arXiv:1702.06441

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 a 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 7. listopadu 2018

Abstrakt

Tato práce se zabývá přípravou trénovací datové sady a trénováním konvoluční neuronové sítě pro detekci registrační značky vozidla ve videu. Pro detekce byla použita technologie Darknet, konkrétně model neuronové sítě YOLOv3-tiny. Řešení bylo zaměřeno na co nejpřesnější detekce a na co nejmenší počet falešných detekcí na obrázek, a tím dosáhnout co nejmenší celkové chyby modelu. Datová sada byla připravena z již existujících volně dostupných datových sad, z datové sady poskytnuté výzkumnou skupinou GRAPH@FIT a z vlastnoručně anotovaných obrázků vytvořených ze stažených videí ze serveru YouTube. Tato datová sada byla dále také zpracována pomocí augmentace dat, čímž byla rozšířena na dvojnásobnou velikost. Pro vytvoření anotací byl použit nástroj YOLO Mark. Pro vizualizaci chybovosti modelu byla použita ROC křivka. Vytvořené řešení dosahuje minimální celkové chyby 10,849 %. Součástí řešení je i již zmiňovaná datová sada.

Abstract

This thesis deals with preparation of training dataset and training of convolutional neural network for licence plate detection in video. Darknet technology was used for detection, specifically the YOLOv3-tiny neural network model. The solution was focused on the most accurate detection and the smallest number of false positives per image, thus minimizing overall model error. Dataset was prepared from existing freely available datasets, from the dataset provided by the GRAPH@FIT research group, and from self-annotated images created from downloaded YouTube videos. Furthermore, this dataset has been processed using data augmentation, extending it to twice the size. The YOLO Mark tool was used to create annotations. An ROC curve was used to visualize the detection success. Created solution reaches minimum total error 10,849 %. Part of the solution is already mentioned dataset.

Klíčová slova

Konvoluční neuronová síť, CNN, Hluboké učení, Darknet, YOLO, YOLOv3, YOLOv3-tiny, detekce registračních značek, detekce objektů, ROC

Keywords

Convolutional neural network, CNN, Deep learning, Darknet, YOLO, YOLOv3, YOLOv3-tiny, licence plate detection, Object Detection, ROC

Citace

LÍBAL, Tomáš. *Detekce registrační značky vozidla ve videu.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Detekce registrační značky vozidla ve videu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Líbal
13. května 2019

Poděkování

Rád bych poděkoval svému vedoucímu prof. Ing. Adamu Heroutovi, Ph.D. za odborné vedení mé práce, jeho věcné připomínky a velkou trpělivost. Dále bych chtěl poděkovat virtuální organizaci MetaCentrum za poskytnutí přístupu k výpočetním zdrojům. Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá problematikou detekce registračních značek na silnicích. Cílem bylo najít vhodný nástroj, který bude schopný detekovat a poté s co největší přesností lokalizovat registrační značku vozidla ať už v obraze, či ve videu v reálném čase, a současně bude schopný vyvarovat se v obraze nebo videu co největšímu počtu falešných detekcí, čímž dosáhne co nejmenší celkové chyby.

Využití by takový nástroj našel na spoustě míst, ať už u výjezdů z parkovišť, kde proběhne kontrola, zda daný automobil nepřekročil povolenou dobu parkování, u vozidel policie, která kontrolují, zda dané vozidlo má dovoleno stát na parkovacím místě nebo zda není hledané, či u dopravních kamer monitorující například křižovatky, kde probíhá sledování automobilů jedoucích na červenou.

Pro tuto úlohu se vybízí počítačové vidění. V této oblasti pro detekci objektů v obraze existuje spousta různých algoritmů založených ať už na strojovém učení či na neuronových sítích z oblasti hlubokého učení (Deep-learning). Díky nárůstu výpočetního výkonu jsou dnes stále častěji používány konvoluční neuronové sítě, což je podtřída již zmiňovaných neuronových sítí, které jsou sice náročnější na výpočetní výkon, ale nabízejí o dost přesnější a rychlejší výsledky, než předchozí metody.

Aby bylo umožněno vytvořit co nejpřesnější nástroj pro detekci poznávacích značek založený na konvolučních neuronových sítích, bylo zapotřebí najít nebo vytvořit vhodný model neuronové sítě, najít nebo pořídit vhodné trénovací a testovací datové sady založené na reálných datech z provozu a natrénovat neuronovou síť, její přesnost a chybovost určit pomocí validace a získané výsledky poté vhodně interpretovat.

Z možných modelů konvolučních neuronových sítí byl zvolen model YOLOv3-tiny. Jako důvody lze uvést jeho rychlost při zachování poměrně vysoké přesnosti a nízká výpočetní náročnost oproti ostatním modelům.

Práce je rozdělena do několika kapitol. Po tomto úvodu následuje kapitola 2, ve které se nachází výčet vybraných metod pro detekci objektů a popis vzniku a vývoje neuronových sítí, včetně principů jejich fungování, a popis konvolučních neuronových sítí a jejich vrstev. V kapitole 3 se poté nacházejí stručné popisy různých modelů klasifikačních a detekčních neuronových sítí, přičemž zvláštní pozornost je věnována různým verzím modelu YOLO. Kapitola 4 se věnuje vytváření datové sady pro trénování a validaci. V kapitole 5 je uveden stručný návod, jak připravit model na trénování a jak trénování následně spustit. V kapitole 6 je natrénovaný model validován a jsou prezentovány dosažené výsledky. V závěrečné kapitole 7 se nachází celkové shrnutí práce, shrnutí dosažených výsledků a jsou navržena možná budoucí rozšíření práce.

Kapitola 2

Metody pro detekci objektů v obraze

Detekce objektů ve fotografii či videu je technika počítačového vidění, která se zabývá rozlišováním objektů v obraze. Na rozdíl od klasifikace obrazu, která pouze sdělí, co za objekt se v obraze nachází, detekce navíc i řekne, kde v obraze se objekty nachází (lokalizace objektů) a tuto oblast vyznačí tzv. bounding-boxem, což je zjednodušeně řečeno obdélníkový tvar označující hranice nalezeného objektu. Zvláštním případem detekce je segmentace objektu, která se používá, pokud je zapotřebí znát přesnou oblast výskytu objektu.

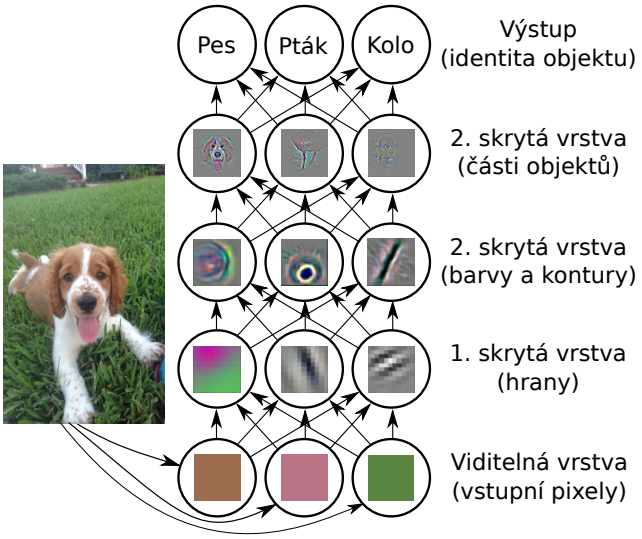
Podle článku [48], který hodnotí současné frameworky, se převážná většina metod detekce objektů skládá ze tří hlavních částí: výběru oblastí, extrakce příznaků (feature extraction) a již zmíněné klasifikace. Možné jsou i různé předzpracování obrazu po jeho pořízení, jako je změna jeho velikosti nebo potlačení šumu. Nasnímaný a předzpracovaný obraz je tedy rozdělen do určitých oblastí, ve kterých je pravděpodobný výskyt hledaných objektů a ve kterých poté probíhá vyhledávání a extrakce příznaků objektů, podle kterých poté klasifikátor rozhodne, o jaký objekt se jedná.

V průběhu historie se k detekci objektů přistupovalo nejrůznějšími metodami, kde ty nejúspěšnější fungovaly na bázi strojového učení či hlubokého učení (Deep Learning). Hluboké učení je jedna z podmnožin strojového učení. Dá se říci, že hluboké učení je způsob realizace strojového učení nebo-li jeho další evoluční krok.

V knize Goodfellowa a kol. [12] je psáno, že u strojového učení se na vstupu očekávají jasně strukturovaná data. Získání takových dat je ale často velmi pracné, protože vnějších faktorů, které je ovlivňují, mohou být spousty. Například při analýze snímku automobilu mohou být negativní vnější faktory třeba úhel záběru, světelné podmínky, či pozice automobilu v obraze. Naopak u hlubokého učení mohou na vstupu být data nestrukturovaná. Často je totiž problémem, že nelze přesně říci, jaké příznaky je zapotřebí extrahovat a hluboké učení tento problém řeší tak, že reprezentace komplexnějších objektů jsou vyjádřeny jinými, jednodušími reprezentacemi. Příklad lze vidět v obrázku 2.1, kde je složitější objekt, jako je pes, reprezentován jednodušími vzory, jako jsou například hrany, barvy či kontury.

2.1 Detekce objektů založené na strojovém učení

V případě detekce objektů za použití strojového učení jsou zapotřebí alespoň dvě věci – pomocí zvolené metody extrahovat sémantické znalosti o příznacích objektů, např. SIFT a HOG příznaky, a poté pomocí nich objekt klasifikovat. Pro **klasifikaci** se často používá



Obrázek 2.1: Ilustrace modelu hlubokého učení. Pro počítač je objekt, jako je třeba pes, velmi složitý, proto je rozdělen do série vzájemně propojených jednodušších vzorů, které jsou popsány v jednotlivých vrstvách. Pixely obrazu jsou načteny vstupní viditelnou vrstvou. Další skryté vrstvy extrahují jednotlivé abstraktní příznaky objektu, jako jsou hrany, barvy, či kontury. Poslední vrstvy už detekují celé části jednotlivých objektů. Na základě jednotlivých částí objektů je v poslední výstupní vrstvě provedena identifikace objektu. Text i schéma obrázku přejaty z knihy od Iana Goodfellowa a kol. [12]. Části obrázku přejaty z článku Zeilera a Ferguse [47].

metoda podpůrných vektorů (Support Vector Machine, SVM [3]). Popis jednotlivých metod v této podkapitole jsem držel velmi stručný, protože s nimi ve své práci dále nepracuji, uvádím je zde jen pro kontext.

2.1.1 Algoritmus Viola-Jones

Přestože je tento algoritmus, který jeho autoři, pánové Paul Viola a Michael Jones, poprvé představili ve svém článku [44], primárně určen k detekci obličejů, dá se použít i pro detekci jiných objektů. Například v diplomové práci [6] autor tento detektor používá k detekci dopravních značek. Tato podkapitola čerpá informace právě z těchto prací.

Metoda detekce objektů Viola-Jones byla poprvé představena v roce 2001 a od předchozích metod se odlišovala svou schopností rychle detekovat obličeje při zachování vysoké úspěšnosti. Na 700 MHz procesu Intel Pentium III toho byla schopná na snímcích o velikosti 384 na 288 pixelů rychlostí až 15 snímků za sekundu. Za tyto vlastnosti metoda vděčí třem klíčovým přispěním:

- Představení tzv. Integrálního obrazu, což je nový způsob reprezentace obrazu, a jeho použití v kombinaci s Haarovými vlnkovými transformacemi.
- Použití metody strojového učení AdaBoost[1]
- Použití kaskádového klasifikátoru, který nezajímavé oblasti ihned zahodí, aby nezabíraly výpočetní výkon zajímavějším oblastem

[1] Adaptive Boosting – podrobnosti o této metodě jsou v článku jejích autorů [9]

2.1.2 Algoritmus Scale Invariant Feature Transform

Tento algoritmus, zkráceně SIFT, představený v roce 1999 panem Davidem Lowem [22] slouží k nalezení korespondence mezi obrazy či scénami. Příznaky jsou neměnné při změně velikosti nebo rotaci a částečně neměnné při změně úhlu záběru, jasu či šumu. Díky tomu je algoritmus SIFT vhodný nejen pro detekci objektů, protože díky rozsáhlé kolekci příznakových vektorů je možné provést klasifikaci např. zmíněným SVM, ale například i pro tzv. Image stitching, což je metoda spojování vícera obrazů s překrývajícími se částmi do jednoho s vyšším rozlišením nebo pro vytváření panoramatických snímků.

Postup, kterým se algoritmus SIFT řídí:

- Detekce lokálních extrémů
- Přesná lokalizace klíčových bodů
- Výpočet deskriptoru
- Vyhledávání shodujících se klíčových bodů

2.1.3 Algoritmus Histogram of Oriented Gradients

Tento algoritmus, známý i pod zkratkou HOG, byl představen pány Navneetem Dalalanem a Billem Triggsem v roce 2005 v jejich článku [4], ve kterém se zaměřili na detekci postav v obraze. Algoritmus je založen na histogramu orientovaných gradientů (HOG).

Gradient je zjednodušeně řečeno změna hodnot intenzit pixelů na osách x a y v obraze. Matematicky vyjádřeno to je lineární suma derivací x a y v obraze a tento vztah se zapisuje rovnicí

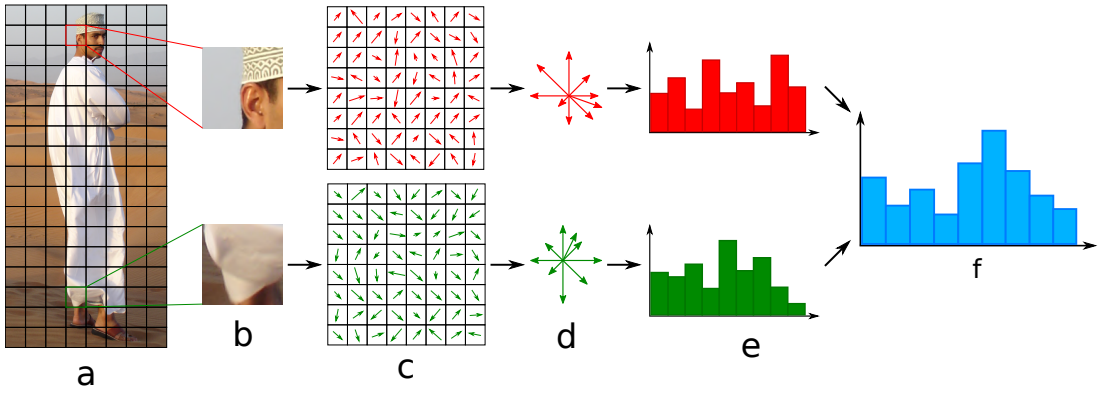
$$\Delta f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}. \tag{2.1}$$

Pro získání histogramu používá algoritmus pohyblivé okno o velikosti 64×128 pixelů. Algoritmus poté v polích o velikosti 8×8 pixelů vypočítá velikost a směr gradientů. Díky tomu pak získá 64 gradientních vektorů s úhly mezi 0–180 stupni. Každé toto pole vygeneruje histogram, ve kterém se na ose x nachází orientace gradientu rozdělené do 9 kategorií po 20 stupních a na ose y zase jejich velikost. Při tomto postupu, který je naznačen na obrázku 2.2, algoritmus dokáže zredukovat 64 vektorů do pouhých 9 hodnot (tzv. HOG příznaky). Pomocí již zmiňované metody pomocných vektorů [3] lze s těmito hodnotami provést klasifikaci objektu.

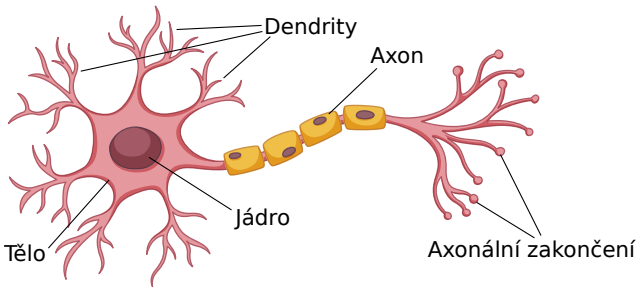
2.2 Detektory objektů založené na hlubokém učení – umělé neuronové sítě

Hluboké učení je, jak už bylo řečeno v úvodu této kapitoly, typ strojového učení a zaměřuje se na nalezení vícera úrovní distribuovaných reprezentací. Tato metoda má skvělé výsledky v oblasti reálného prostředí. I když u hlubokého učení existuje spousta jiných technologií, jako jsou kupříkladu hloubkově omezené jádrové algoritmy [40], umělé neuronové sítě mají v této oblasti jasnou převahu a tudíž se tato podkapitola věnuje právě jim.

Na otázku, proč jsou neuronové sítě stále populárnější, se dá odpovědět vyjmenováním několika důležitých a trvajících trendů. Jsou to například tyto [48]:



Obrázek 2.2: Proces extrakce HOG příznaků. a) Pohyblivé okno, b) Pole 8×8, c) výpočet gradientů, d) a e) vygenerování histogramu, f) Sloučení všech histogramů do konečného příznakového vektoru. Inspirováno obrázkem z článku [2].



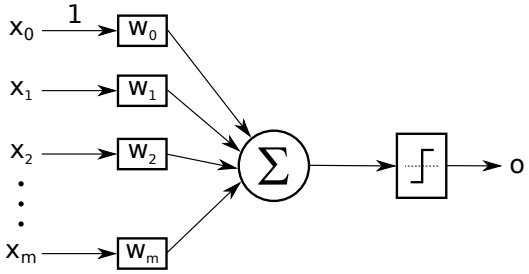
Obrázek 2.3: Biologický neuron. Krátké dostředivé výběžky se nazývají *dendrity*. Dlouhé odstředivé výběžky se nazývají *axony*. Literatura [24] popisuje princip fungování neuronu tak, že transmisní chemikálie v tekutině mozku mění elektrický potenciál uvnitř těla neuronu. Pokud tento potenciál dosáhne určité prahové hodnoty, neuron vyšle puls do axonu. Axony vytváří spojení s mnoha jinými neurony, které se připojují ke každému z těchto neuronů v oblasti nazývané synapse.

- Roste výpočetní výkon počítačů, obzvlášť grafických karet, které se ukázaly pro maticové operace jako velmi vhodné
- Vytváří se nové a rozšiřují se starší **anotované** datové sady pro trénování neuronových sítí
- Postupně se zdokonaluje design neuronových sítí, což vede k čím dál přesnějším a rychlejším výsledkům

2.2.1 Umělý neuron a Perceptron

Umělé neuronové sítě vznikly snahou napodobit funkčnost mozku při řešení složitých úloh. Mozek složitou úlohu postupně dekomponuje na čím dál menší podproblémy, které postupně vyřeší. Základní stavební jednotkou mozku je *biologický neuron*, jeho ukázka je na obrázku 2.3[2].

[2]obrázek byl převzat z https://www.freepik.com/free-vector/stem-cell-diagram-white-background_2480958.html



Obrázek 2.4: Model Perceptronu tvořený jediným neuronem. Vstupní signály x_i jsou vynásobeny váhami w_i. Perceptron následně vytvoří sumu těchto hodnot. Pokud je výsledná hodnota větší, než prahová hodnota daná aktivační funkcí signum, neuron vyšle signál na výstup. V opačném případě nikoliv. Vstupní signál x_0 se nazývá *práh (bias)* a v umělém neuronu slouží např. k tomu, aby v případě, kdy by všechny vstupní signály byly nulové, bylo možné dál kontrolovat výstup neuronu. Obrázek a text s úpravami přejaty z Marslandovy knihy [24].

Za počátek umělých neuronových sítí se dá podle Goodfellowa a kol. [12] považovat vytvoření modelu *umělého neuronu* pány Pittsem a McCullochem [26] ve čtyřicátých letech 20. století. Jejich model však dokázal ze vstupních signálů rozpoznat pouze dvě různé kategorie. Na jejich práci později v padesátých letech navázal Frank Rosenblatt, který jejich model umělého neuronu použil na implementaci *Perceptronu* a definoval pro něj učící metodu. Perceptron byl první model, který se dovedl naučit váhy, které definovaly kategorie vstupních signálů. Ukázka Perceptronu je společně s popisem fungování na obrázku 2.4. Perceptron může být tvořen jedním nebo i více neurony a množinou vstupů a vah pro urychlení vstupů do těchto neuronů.

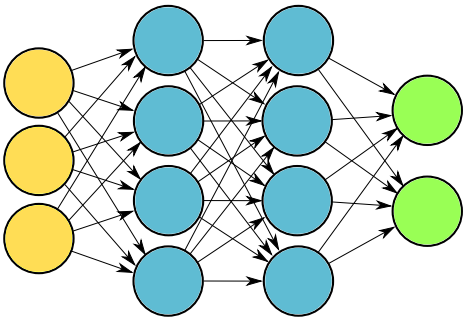
Perceptron byl namodelován jako soubor váhových vstupů w_i, sčítačky vstupních signálů a aktivační skokové funkce signum, která rozhoduje, zda pro dané vstupy vyšle nebo nevyšle signál na výstup. Aktivační funkcí mohou být ale například i lineární funkce, logistická funkce sigmoida, funkce hyperbolický tangens, saturační přenosová funkce nebo funkce ReLU. Některé z těchto funkcí budou ještě popsány později. Výběr této funkce má poté vliv na konvergenci výpočtu naučení neuronové sítě. Matematicky se dá Perceptron popsat zjednodušenou rovnicí z knihy Stephana Marslanda [24]

$$o = g\left(\sum_{i=0}^{m} w_i x_i\right) = \begin{cases} 1 & \text{pokud } w_i x_i > \text{prahová hodnota} \\ 0 & \text{pokud } w_i x_i \leq \text{prahová hodnota} \end{cases}, \tag{2.2}$$

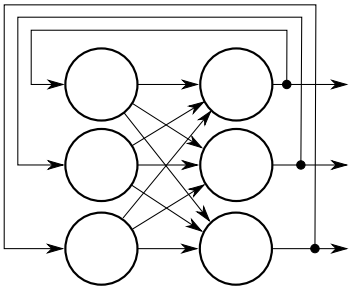
kde g je aktivační funkce a o je výsledek.

2.2.2 Struktura a popis funkce vícevrstvých neuronových sítí

Jelikož je samotný Perceptron lineární model, nedá se velmi často použít pro řešení komplexnějších úloh. Z tohoto důvodu se Perceptrony vzájemně propojují do neuronových sítí. Pro Perceptron se v tomto kontextu vžil obecný pojem neuron, proto dál bude uváděn s tímto názvem. Vícevrstvá neuronová síť (MLP, Multi Layer Perceptron) je složena z neuronů, které jsou mezi sebou vzájemně propojeny tak, že výstup neuronu z jedné vrstvy je vstupem jednoho nebo i více neuronů v další vrstvě. Z hlediska využití jsou v síti rozlišovány *vstupní*, *skryté*(pracovní, vnitřní) a *výstupní* vrstvy neuronů. Topologii těchto sítí se



Obrázek 2.5: Model vícevrstvé neuronové sítě. Žlutá je vrstva *vstupní*, modré jsou vrstvy *skryté* a zelená je vrstva *výstupní*. Jelikož se první vrstva nepočítá mezi produkční vrstvy, lze nazvat tuto síť třívrstvou. Skládá se tedy jen ze dvou skrytých vrstev a vrstvy výstupní.



Obrázek 2.6: Model rekurentní neuronové sítě. Neurony na vstup dostávají svůj vlastní výstup.

říká *Dopředné neuronové sítě (feedforward networks)*. Jako příklady dopředných sítí mohou sloužit obrázky 2.1 a 2.5.

Druhý typ topologie neuronových sítí jsou *Rekurentní neuronové sítě* (sítě se zpětnou vazbou). Základními vlastnostmi rekurentních sítí je, že jsou jednovrstvé a že počet vstupů se rovná počtu výstupů. Neurony tedy obdrží na vstupu vlastní výstup o jednu nebo i více iterací později. Příklad rekurentní neuronové sítě je možné vidět na obrázku 2.6.

Pro tuto práci je jedna z nejdůležitějších topologií neuronových sítí tzv. Konvoluční neuronová sít. Této síťové topologii je věnována samostatná podkapitola 2.2.4.

2.2.3 Učení (trénování) neuronových sítí

Podle Marslanda [24] je možné pojem „učení“ definovat jako „Stát se při řešení některé úlohy lepším na základě získávání zkušeností.“ Při učení neuronové sítě se tedy očekává, že síť po předložení menšího počtu řešených úkolů bude schopná úspěšně řešit větší počet podobných dalších úkolů. Tomuto se říká *generalizace*. Metod, jak dosáhnout toho, aby se neuronová sít naučila řešit nějaký problém, je hned několik.

Jedna z nejpoužívanějších metod pro učení neuronových sítí je metoda *učení s učitelem*. Tato metoda očekává na vstupu množinu trénovacích dat, která obsahuje dvojice vstupní data – požadovaná výstupní data. Učení probíhá na principu toho, že učící algoritmus po každém kroku kontroluje, jak moc se získaný výsledek liší od výsledku, který byl očekáván, a podle toho se rozhodne, jak upravit váhy propojení tak, aby v dalším kroku byl tento rozdíl menší.

Další používaná metoda je metoda *učení bez učitele*. Tato metoda, na rozdíl od metody učení s učitelem, na vstupu získá pouze vstupní data, ale už ne výstupní. Učící algoritmus se tedy snaží najít podobnosti mezi vstupními daty a podle nich dělí tato vstupní data do různých kategorií.

Jako další učící metody lze ještě jmenovat například *zpětnovazební učení* (Reinforcement learning), kde jsou stavy ohodnocovány na základě zpětné vazby, nebo *evoluční algoritmy*, které se často při učení inspirují chováním jevů v přírodě.

Algoritmus zpětného šíření chyby (Back-propagation)

Jak už bylo řečeno, nejpoužívanějším algoritmem pro učení neuronových sítí je *metoda učení s učitelem*, která říká, jak moc se výsledek liší od toho očekávaného. Tomuto rozdílu se také říká *velikost chyby*. Aby bylo možné tuto chybu minimalizovat, je zapotřebí upravit váhy jednotlivých propojení neuronů. Jelikož je ale neuronová síť většinou tvořena více vrstvami, není možné s jistotou říci, v jaké vrstvě a s jakou mírou je zapotřebí dané váhy upravit.

Řešení navrhli v roce 1986 pánové Rumelhart, Hinton a Williams [7]. Jejich řešení spočívalo v představení učícího algoritmu, který po dokončení dopředné fáze spočítá chybu neuronové sítě a tuto chybu zpětně rozšíří zpět. Tento postup iterativně opakuje, dokud není dosaženo minimální tolerované chyby. Rumelhartův, Hintonův a Williamsův [7] algoritmus zpětného šíření chyby funguje přibližně takto: Pro každý neuron ve výstupní vrstvě jsou vypočítané výsledky neuronu porovnány s očekávanými výsledky. Z porovnání těchto hodnot vzejde chyba neuronové sítě, pro kterou je vypočítán faktor, který odpovídá části chyby, která se z neuronu šíří zpětně do všech neuronů předchozí vrstvy, které mají s tímto neuronem společné spojení. Obdobně lze definovat i faktor pro předchozí vrstvu. Úprava vah ve vrstvě poté záleží na zmiňovaném faktoru a na výsledcích neuronů v předchozí vrstvě.

Aby bylo možné metodu zpětného šíření chyby použít, je nutné, aby aktivační funkce byla spojitá, diferencovatelná a monotónně neklesající. Nejčastěji jsou tedy používány logistická funkce sigmoida, jejíž definice je zapsána rovnicí

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{2.3}$$

a hyperbolická funkce tangens, jejíž definice je zapsána rovnicí

$$\tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}} \ . \tag{2.4}$$

Pro výpočet chybové funkce se používá funkce *Suma čtverců chyb* (MSE, Mean Squared Error). Umocnění se v této funkci používá z důvodu, že je potřeba, aby všechny vypočítané chyby byly stejného znaménka. Mohlo by se totiž stát, že sumarizací všech chyb by vznikla nula, což by bylo vyhodnoceno, jako že žádná chyba nenastala. Díky umocnění se ve výpočtu také více projeví větší rozdíly. Funkce MSE, kde k je počet trénovacích vzorků, se zapisuje v tomto tvaru:

$$E(\mathbf{t}, \mathbf{y}) = \frac{1}{2} \sum_{k=1}^{n} (t_k - y_k)^2. \tag{2.5}$$

Při hledání minimální chyby se používá tzv. *Gradientní sestup*, což je optimalizační algoritmus, který používá parciální derivace chybové funkce pro nalezení směru klesání chyby. Princip tohoto algoritmu se dá podle článku [27] přirovnat k sestupu slepého horolezce z vysokého kopce. Jeho cílem je dostat se z kopce co nejvíce dolů a to co nejkratší cestou,

tudíž jeho kroky jsou směřovány tím směrem, kde v danou chvíli pociťuje největší klesání. Matematicky lze toto zapsat jako

$$\Theta^1 = \Theta^0 - \alpha \nabla J(\Theta), \tag{2.6}$$

kde Θ^1 je další krok, Θ^0 je současná pozice, α je velikost kroku a $\nabla J(\Theta)$ je směr pohybu. Záporné znaménko je zde proto, že další krok jde v opačném směru, než směr růstu, či-li dolů. Tyto kroky se iterativně opakují, dokud se nepřestane chyba snižovat. Může se ale vyskytnout problém s uvíznutím v lokálním minimu. To znamená, že zde není jistota, zda se někde ještě nevyskytuje nižší hodnota (globální minimum). Toto se dá opět přirovnat na příkladu s horolezcem. Horolezec sice došel do nejnižšího bodu pod kopcem, neví však, jestli za dalším menším kopcem není možné sejít ještě níž a nemá už sílu na tento kopec vyjít.

Většina neuronových sítí dnes používá *Stochastický gradientní sestup*. V tomto případě se při každé iteraci nepoužívá každý prvek z celé trénovací sady, ale jen velmi malý počet náhodných prvků z ní, což má za následek značné zrychlení. Kroky této metody tedy sice nemíří přímo správným směrem, postupně ale k řešení aproximují. Jinými slovy, každý krok je značně méně náročný na výpočet, ale těchto kroků je potřeba vykonat pro nalezení řešení podstatně více [34].

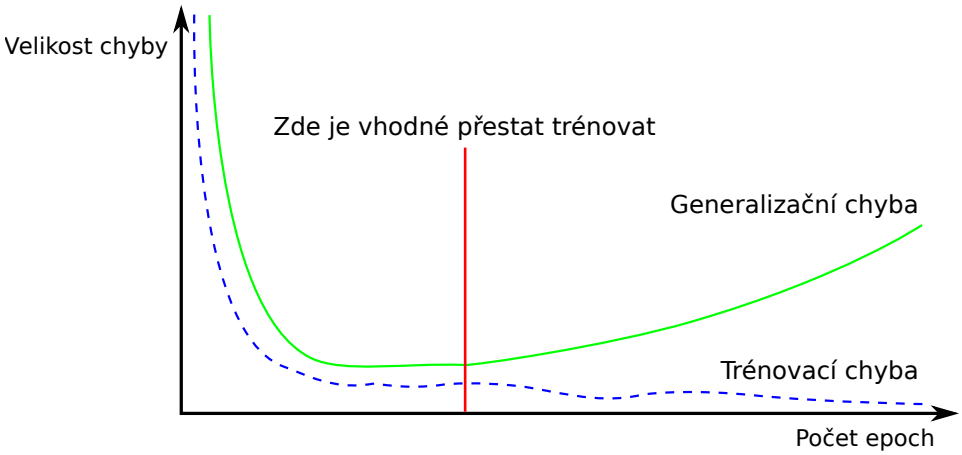
Pro samotný výpočet parciálních derivací (gradientů) algoritmus zpětného šíření chyby využívá tzv. *řetízkové pravidlo* (chain-rule) pro derivaci složených funkcí, které je definováno rovnicí

$$\frac{dF}{dx} = \frac{df}{dg}\frac{dg}{dx}, \quad \text{pokud platí} \quad F(x) = f(g(x)). \tag{2.7}$$

Pseudokód trénování (učení) neuronové sítě je naznačen v algoritmu 1, který byl s drobnými úpravami převzat z knihy Stephena Marslanda [24].

Algoritmus 1 Trénování neuronové sítě s algoritmem zpětného šíření chyby

Inicializace nastav všechny váhy na malé (kladné i záporné) náhodné hodnoty
while není konec trénování **do**
  for each vstupní vektor **do**
    ▷ **Dopředná fáze**
    $a_j = \frac{1}{1+\exp(-\beta \sum_i x_i v_{ij})}$ ▷ Vypočítej aktivační sigmoidní funkci každého neuronu j ve skrytých vrstvách (v jsou váhy první vrstvy)
    $y_k = \frac{1}{1+\exp(-\beta \sum_j a_j w_{jk})}$ ▷ Propracuj se celou sítí až k výstupní vrstvě, která má tuto aktivační sigmoidní funkci (w jsou váhy druhé vrstvy)
    ▷ **Zpáteční fáze**
    $\delta_{\text{výst}k} = (t_k - y_k) y_k (1 - y_k)$ ▷ Vypočítej chybu na výstupu
    $\delta_{\text{skryt}j} = a_j (1 - a_j) \sum_k w_{jk} \delta_{\text{výst}k}$ ▷ Vypočítej chybu skrytých vrstev
    $w_{jk} \leftarrow w_{jk} + \eta \delta_{\text{výst}k} a_j^{\text{skryté}}$ ▷ Aktualizuj váhy na výstupní vrstvě
    $v_{ij} \leftarrow v_{ij} + \eta \delta_{\text{skryt}j} x_i$ ▷ Aktualizuj váhy ve skrytých vrstvách
  end for
  Pro další iteraci zamíchej pořadí vstupních vektorů
end while



Obrázek 2.7: Graf velikosti trénovací a generalizační chyby v závislosti na počtu proběhlých epoch. Z obrázku je patrné, že je vhodné trénování ukončit ve chvíli, kdy se mezera mezi jednotlivými chybami začne rozšiřovat. Obrázek přejat od Goodfellowa a kol. [12].

Přeučení a nedoučení neuronové sítě

Jak už bylo psáno výše, při trénování neuronových sítí je možné díky anotované trénovací datové sadě pomocí výpočtů získat tzv. *trénovací chybu*. S touto chybou souvisí i testovací chyba tzv. *generalizační chyba*, která je definována jako očekávaná hodnota chyby na novém vstupu. [12]

S testováním souvisí nutnost rozdělit trénovací datovou sadu do dalších sad, typicky na testovací datovou sadu a validační datovou sadu. Doporučený poměr je podle literatury [24] asi 50:25:25 pro velkou datovou sadu, 60:20:20 pro menší datovou sadu. V případě, že je datová sada opravdu malá, dá se použít tzv. *leave-one-out křížová validace*. Tento způsob validace spočívá v tom, že datová sada je náhodně rozdělena do N dalších sad, kde jedna sada je použita pro validaci, zatímco síť je natrénována na zbylých. Jiná sada je poté použita pro validaci a síť se opět natrénuje na zbylých. Tento postup se opakuje pro všech N vytvořených sad. Pro otestování a použití je poté vybrána síť s nejmenší validační chybou. V podstatě se jedná o výměnu dat za výpočetní výkon, protože je zapotřebí natrénovat N různých sítí namísto jedné.

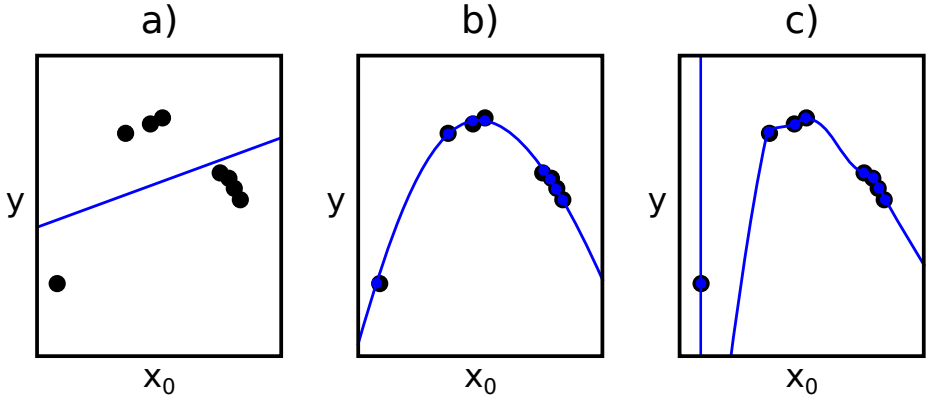
Při samotném trénování se hledí na to, aby trénovací chyba byla co nejmenší. Generalizační chyba bude typicky větší nebo stejná, jako trénovací chyba, tudíž se hledí na to, aby mezera mezi trénovací a generalizační chybou byla co nejmenší. Díky znalosti těchto chyb je možné odhadnout, kdy asi zastavit trénování. Ukázka je na obrázku 2.7.

Pokud je trénování zastaveno moc brzy nebo naopak příliš pozdě, zvyšuje se riziko vlivu podtrénování a přetrénování, s čímž je spojeno snižování schopnosti generalizace. Toto je naznačeno na obrázku 2.8.

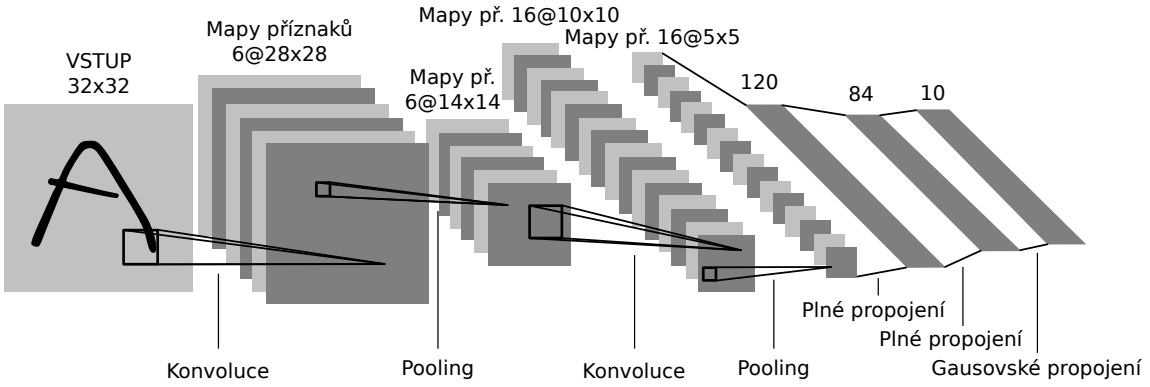
2.2.4 Konvoluční neuronové sítě

Konvoluční neuronové sítě (CNN) poprvé představili ve své práci Yann LeCun a kol. [20, 19]. Z těchto dvou prací je čerpána většina informací pro tuto podkapitolu.

Konvoluční neuronové sítě jsou vícevrstvé neuronové sítě speciálně uzpůsobené pro práci s daty ve formě několika polí. Typickým příkladem takových dat jsou obrázky či videa, které se skládají ze tří dvoudimenzionálních polí intenzit pixelů – každé pole pro jeden barevný



Obrázek 2.8: Vliv velikosti trénování. a) *Podtrénování* – Lineární funkce nedovedla zachytit zakřivení. b) *Správná generalizace* – Kvadratická funkce zachytila všechny vyznačené hodnoty a netrpí žádným velkým vlivem podtrénování nebo přetrénování. c) *Přetrénování* – Polynomiální funkce prochází perfektně vyznačenými hodnotami. Tímto je značně omezena možnost generalizace. Obrázek a text přejat od Goodfellowa a kol. [12].



Obrázek 2.9: Architektura konvoluční neuronové sítě LeNet-5 z LeCunova článku [20]. V obrázku je patrný standardní tvar klasifikačních konvolučních neuronových sítí nad obrazem – na začátku se pracuje s velkým rozlišením a malým počtem kanálů, ale postupně se rozlišení obrazu snižuje a počet kanálů zvyšuje.

kanál. Existují čtyři klíčové vlastnosti konvolučních sítí, které jim dávají výhodu oproti klasickým neuronovým sítím: lokální propojení, sdílené váhy, pooling a použití velkého počtu vrstev.

Typická architektura konvolučních neuronových sítí se tedy skládá z několika párů *konvolučních a pooling* (subsampling) *vrstev*. Poslední vrstvy se naopak skládají z několika *plně propojených vrstev* [1]. Příklad architektury konvoluční neuronové sítě je možné vidět na obrázku 2.9.

Konvoluční vrstva

Hlavním úkolem konvoluční vrstvy je detekovat lokální spojení příznaků z předchozí vrstvy. Důvod je pro to takový, že ve většině obrázků jsou lokální skupiny hodnot mezi sebou vysoce korelované, což vytváří lokální motivy, které se dají dobře detekovat. Tyto motivy se mohou objevit na jakémkoliv místě v obrázku.

Na rozdíl od ostatních neuronových sítí, konvoluční neuronové sítě používají namísto obyčejného násobení matic matematickou operaci jménem *konvoluce*. Pro aplikaci této operace na data se používá právě tato vrstva. Konvoluci je nutné uplatnit jednotlivě na všechny barevné kanály. Aby se neuronová síť mohla nazývat konvoluční, musí obsahovat minimálně jednu konvoluční vrstvu [12]. Protože síť pracuje s dvojrozměrnými obrazy, což jsou vlastně dvojrozměrné diskrétní hodnoty, používá se tzv. *diskrétní konvoluce*, která je definována jako

$$g(x,y) = f(x,y) * h(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j)h(i,j), \tag{2.8}$$

kde g je výstupní obraz, kterému se často říká *mapa příznaků*, f je vstupní obraz a h je konvoluční jádro (maska) o rozměrech k krát k (většinou 3×3 až 6×6 buněk).

Problém s operací konvoluce je v okrajových bodech obrázku, kde konvoluční jádro (maska) přesahuje hranici obrázku. Je několik způsobů, jak se tento problém dá řešit. Mezi nejpoužívanější patří:

1. *Valid convolution* – Konvoluce se počítá pouze v oblastech, do kterých se vejde celé konvoluční jádro. Tímto dojde ke zmenšení výsledného obrázku v závislosti na rozměrech jádra. V případě, kdy konvoluční jádro má šířku n, je šířka výsledného obrazu zmenšena o $n-1$ pixelů v každé vrstvě.

2. *Same convolution* – Originální obraz je za okraji zvětšen pouze o potřebné oblasti v závislosti na rozměrech. Hodnoty těchto oblastí se nastaví na 0. Rozměry výsledného obrázku poté zůstanou zachovány.

3. *Full convolution* – Hraniční oblasti mají na výstupní obraz menší vliv, než oblasti bližší středu. Pro každou k-krát navštívenou oblast je přidán dostatek nul. Výsledný obraz má šířku $m + k - 1$.

Součástí konvoluční vrstvy může být i aplikace aktivační funkce. Konvoluční neuronové sítě na rozdíl od klasických neuronových sítí používají jako aktivační funkci nelineární funkci ReLU (Rectified Linear Activation). Touto funkcí jsou zpracovány všechny lineární aktivace, které byly vytvořeny během konvolucí. Funkce je zapsána jako

$$f(x) = \max(0, x). \tag{2.9}$$

Jinými slovy pokud je x pozitivní hodnota, výsledek funkce bude x. Pokud je hodnota x záporná, výsledek funkce bude nula. Tato funkce je tedy méně výpočetně náročná než dříve zmiňované funkce signum a hyperbolický tangens. Funkce ReLU ale trpí problémem, který je popsán v článku [37] – jelikož pro všechny záporné hodnoty vrací funkce nulu, váhy se v této oblasti neaktualizují. To znamená, že neurony, které se dostanou do tohoto stavu, přestanou reagovat na další odchylky a část sítě proto „umře“. Tento problém se nazývá *Dying ReLU*. Řešením může být pro záporné hodnoty vracet lehce vyšší hodnoty – např. $y = 0.01x$ když $x < 0$. Hlavním cílem tohoto řešení tedy je pro záporné hodnoty nevracet nulu, aby se neurony postupně zotavovaly. Toto řešení má název *Leaky ReLU*.

Pooling vrstva

Poolingovou vrstvu je ve zvyku umisťovat hned za vrstvu konvoluční a tvořit s ní páry. Hlavním úkolem poolingové vrstvy je spojit dohromady sémanticky podobné příznaky. Tím dochází ke zmenšení obrázku a počtu dat, která by bylo potřeba později opět zpracovat.

Nejčastější operace, jakou je pooling realizován, je tzv. *max pooling*, který na určité pozici ve výstupním obraze nahradí hodnotu nalezeným maximem v obdélníkovém okolí buňky. Další častou operací je tzv. *average pooling*, který naopak hodnotu ve výstupním obraze nahradí průměrnou hodnotou obdélníkového okolí buňky.

Plně propojená vrstva

Jedny z posledních vrstev v každé konvoluční neuronové síti jsou plně propojené vrstvy. Neurony v této vrstvě jsou plně propojeny se všemi neurony v předchozí vrstvě. Plně propojená vrstva je chováním i vlastnostmi stejná, jako v klasické neuronové síti.

Hlavním úkolem, který má tato vrstva na starost, je transformovat mapu příznaků na příznakový vektor a pomocí něj provést klasifikaci.

Poslední vrstva konvoluční neuronové sítě obsahuje stejný počet neuronů, jako je počet klasifikačních tříd. Pokud jsou jsou klasifikovány více než dvě třídy objektů, používá se jako aktivační funkce tzv. funkce *softmax*.

Kapitola 3

Modely konvolučních neuronových sítí

V posledních letech vznikly spousty různých modelů konvolučních neuronových sítí. Důvod, proč se staly tak populárními v oblasti detekcí objektů či klasifikací obrázků, lze vidět například na výsledcích soutěže ImageNet Large Scale Visual Recognition Challenge (dále ILSVRC) [35], která se právě vyhodnocováním algoritmů v této oblasti zabývá – na předních příčkách v této soutěži různé modely konvolučních neuronových sítí dominují.

I když se tato práce zabývá detekcí objektů a nikoliv klasifikací obrázků nebo segmentací objektů, jsou zde vybrané modely, které se touto problematikou zabývají, pro kontext uvedeny také. Je to z důvodu, že spousty různých frameworků pro detekci nebo segmentaci objektů jsou z modelů pro klasifikaci objektů určitým způsobem odvozeny [13].

3.1 Vybrané modely pro klasifikaci obrázků

Většina informací pro tuto podkapitolu je čerpána z článků [13, 5] a z výsledků zmiňované soutěže ILSVRC [35].

Jak už bylo řečeno v předchozí kapitole 2.2.4, průkopníkem v oblasti konvolučních neuronových sítí byl pan LeCun se sítí *LeNet-5* pro klasifikaci ručně psaných číslic z roku 1998. Tato síť, jejíž architektura už byla naznačena na obrázku 2.9, se skládala ze 7 vrstev, které obsahovaly 60 tisíc parametrů[1] a na vstupu požadovala šedotónové obrázky o velikosti 32×32 pixelů.

V roce 2012 došlo v soutěži ILSVRC ke zlomu, když v té době největší model *AlexNet* [17] drtivě překonal všechny její ostatní účastníky tím, že dosavadní top-5 error[2] snížil z 26 % na 15,3 %. Tento model se skládal z 5 konvolučních a 3 plně propojených vrstev, které dohromady obsahovaly okolo 60 miliónů parametrů. Jedny z důvodů, proč byl AlexNet tak úspěšný, byly akcelerace výpočtů na GPU kartě nebo představení regularizační metody pro potlačování přetrénování „dropout“. Nevýhodou ale je, že na vstupu očekával pouze data pevné velikosti 224×224 pixelů.

Další velmi úspěšný model *VGGNet* [38] dosáhl o pár let později hodnoty top-5 error 7,32 %. Úspěchu v podobě tohoto výsledku dosáhl zkoumáním vlivu použití hlubších sítí s větším počtem konvolučních vrstev, které obsahují velmi malé konvoluční filtry velikosti

[1] Parametr je počet učitelných elementů pro všechny filtry ve všech vrstvách

[2] Top-5 error vyjadřuje hodnotu, v kolika procentech případů se očekávané označení vyskytuje v pětici označení, které mají v obraze podle modelu největší pravděpodobnost výskytu.

3×3 pixely. Architektura tohoto modelu zůstává jednoduchou, i když se skládá z 13-15 konvolučních a 3 plně propojených vrstev. Tento model má však ale nevýhodu v tom, že kvůli své hloubce obsahuje okolo 138 miliónů parametrů.

Jednomu z nejúspěšnějších současných modelů *GoogLeNet* [41] (známému také jako Inception v1) se v roce 2014 podařilo soutěž ILSVRC vyhrát s výsledkem top-5 error 6,67 %. Tento výsledek už se blížil výsledkům člověka, jehož top-5 error se nachází přibližně mezi 3,5–5 %. Hlavní přínosem tohoto modelu bylo, že podstatně zvýšil šířku a hloubku neuronové sítě, aniž by se nějak výrazně zvýšily výpočetní nároky. Tento model, který se skládá z 21 konvolučních vrstev a 1 plně propojené vrstvy, byl inspirovaný modelem LeNet a implementoval nový prvek, tzv. Inception modul, který je založen na několika velmi malých konvolucích. Díky těmto malým konvolucím a pouze jedné plně propojené vrstvě GoogLeNet obsahuje „pouze“ 4 milióny parametrů, což je při hloubce a šířce tohoto modelu v porovnání s například již zmiňovaným modelem AlexNet podstatně velký rozdíl.

Druhý z nejúspěšnějších současných modelů je *ResNet* [14] z výzkumné skupiny Microsoftu. Tento model v soutěži ILSVRC v roce 2015 překonal svými výsledky člověka a to s hodnotou top-5 error 3,57 %. Tato síť, která se skládala ze 152 vrstev, se dovedla takto natrénovat díky tzv. „zkratkám“ (skip connections), což byl nově představený přístup, který umožnil některé vrstvy přeskočit.

3.2 Vybrané modely pro detekci nebo sémantickou segmentaci objektů v obraze

V této podkapitole budou mezi sebou porovnány některé nejznámější modely pro detekci či sémantickou segmentaci objektů v obraze. Tyto modely lze obecně rozdělit do dvou kategorií [23]:

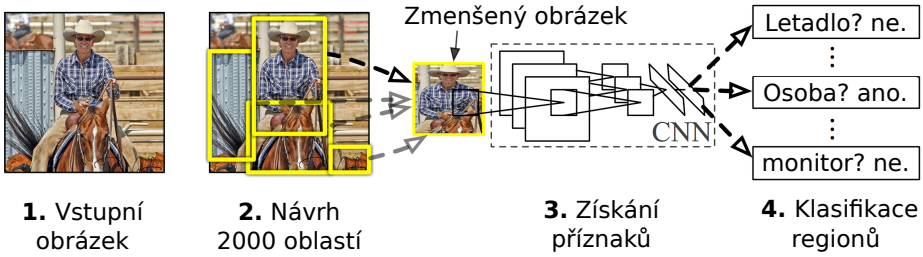
1. **Dvoufázové modely založené na klasifikaci** – Činnost těchto modelů je rozdělena do dvou fází. V první fázi probíhá tzv. navrhování oblastí. V těchto oblastech jsou poté pomocí modelu pro klasifikaci obrázků vypočteny predikce a podle jejich výsledků provedena klasifikace objektů. Tato metoda je relativně pomalá (řády sekund), protože se tento postup aplikuje na každou navrženou oblast. Mezi zástupce těchto modelů patří modely z rodiny R-CNN.

2. **Jednofázové modely založené na regresi** – Tyto modely provádí lokalizaci a klasifikaci objektů v jediném průchodu neuronovou sítí. Díky tomu jsou podstatně rychlejší – dovedou provádět detekce v reálném čase. Mezi typické zástupce patří modely You Only Look Once (YOLO) či Single Shot Detector (SSD).

V kontextu této podkapitoly se několikrát hovoří o datových sadách PASCAL VOC (dále už jen VOC) a COCO. Tyto datové sady obsahují standardizovaná anotovaná obrázková data pro klasifikaci a detekci objektů, rozdělená do kategorií pro trénování, testování a validaci.

3.2.1 Rodina modelů R-CNN (Region-based Convolutional Neural Network)

V době před představením prvního modelu R-CNN [11] v roce 2014 většina metod pro detekci objektů využívala převážně různé algoritmy pro extrakci příznaků a práci s nimi. Takovým algoritmem byl například dříve zmiňovaný algoritmus SIFT z kapitoly 2.1.2 nebo



Obrázek 3.1: Model R-CNN. Obrázek přejat z článku [11] a upraven.

algoritmus HOG z kapitoly 2.1.3. Model R-CNN však dosavadní metody překonal svou přesností, která byla v porovnání s nimi na datech z datové sady VOC 2012 až o 30 % vyšší.

Ze základního modelu R-CNN s postupem času vzniklo několik dalších modelů, které budou společně s ním popsány v této podkapitole. Modely z rodiny R-CNN se dají použít jak pro detekci objektů, tak i jejich sémantickou segmentaci. Pro potřeby této práce však sémantická segmentace není podstatná a je proto vynechána.

R-CNN

Tento model z roku 2013 byl úplně prvním z rodiny modelů R-CNN a položil základ všem dalším jejím modelům. Úspěchu na poli detekce objektů dosáhl představením detekčního systému, který se skládal ze 3 modulů.

V prvním modulu probíhalo generování návrhů oblastí s vysokou pravděpodobností výskytu objektu s použitím algoritmu selektivního vyhledávání. Algoritmus selektivního vyhledávání fungoval tak, že oblasti, které měly velmi podobné vlastnosti, rekurzivně slučoval. Tímto slučováním byl celkový počet navrhovaných oblastí snížen na přibližně 2000. Druhý modul byl tvořený modelem neuronové sítě AlexNet nebo VGG. Z toho důvodu bylo nutné velikost každé navržené oblasti transformovat na velikost 224×224 pixelů. Model poté nad každou navrženou oblastí spustil neuronovou síť a extrahoval příznakový vektor, který byl poté ve třetím modulu použit pro klasifikaci objektů pomocí metody pomocných vektorů (SVM). Schéma modelu R-CNN, včetně znázornění popisovaných kroků, je na obrázku 3.1.

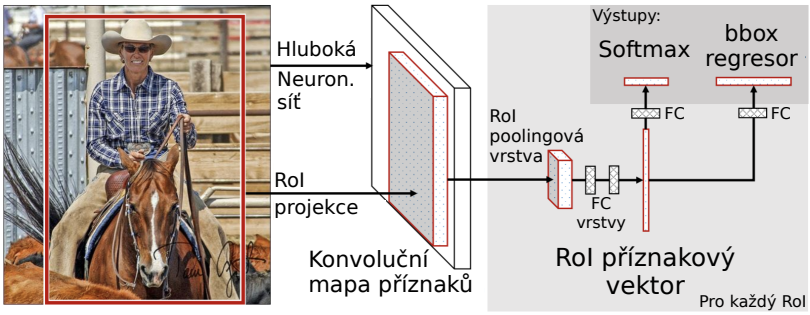
Model R-CNN dosahoval na datové sadě VOC 2007 přesnosti 58,5 % mAP[3], nicméně pro detekci v reálném čase byl nepoužitelný – zpracování jednoho obrázku a získání predikcí objektů trvalo až 50 sekund a čas trénování byl také příliš dlouhý.

Fast R-CNN

Model Fast R-CNN [10] vzešel v roce 2015 z předchozího modelu R-CNN. Hlavním rozdílem mezi těmito modely bylo předávání vstupního obrázku přímo konvoluční neuronové síti VGG, která se osvědčila lépe, než model AlexNet. Společně s obrázkem byly síti předány i návrhy oblastí zájmu a ta poté z obrázku extrahovala mapu příznaků. Schéma modelu Fast R-CNN a popis jeho funkce jsou na obrázku 3.2.

Výsledkem těchto změn tedy bylo, že díky extrakci příznaků ještě před návrhem oblastí odpadla nutnost spouštět postupně neuronovou síť nad každou z 2000 navrhovaných oblastí, jako tomu bylo u modelu R-CNN. Tím, že metoda podpůrných vektorů byla nahrazena softmax vrstvou, došlo ke zjednodušení celého modelu. Díky těmto změnám byla přesnost

[3]Mean Average Precision – průměrná přesnost



Obrázek 3.2: Model Fast R-CNN. Vstupní obrázek a oblasti zájmu jsou poslány na vstup konvoluční neuronové sítě VGG, která z obrázku vytvoří mapu příznaků. Pro každou oblast zájmu je pomocí RoI (Region of Interest) poolingových vrstev z příznakové mapy vytvořen příznakový vektor, jehož výstup ústí do dvou výstupních sesterských vrstev. Výstupem této sítě jsou pravděpodobnosti výskytu jednotlivých objektových tříd a hodnoty reprezentující polohu bounding-boxů. Obrázek a text přejaty z článku [10] a upraveny.

modelu Fast R-CNN zvýšena na hodnotu okolo 70 % mAP a model byl zároveň při trénování i testování mnohem rychlejší, než model R-CNN. Čas potřebný pro zpracování obrázku a získání predikcí objektů se snížil na přibližně 2 sekundy, což je téměř 25-ti násobné zrychlení oproti R-CNN.

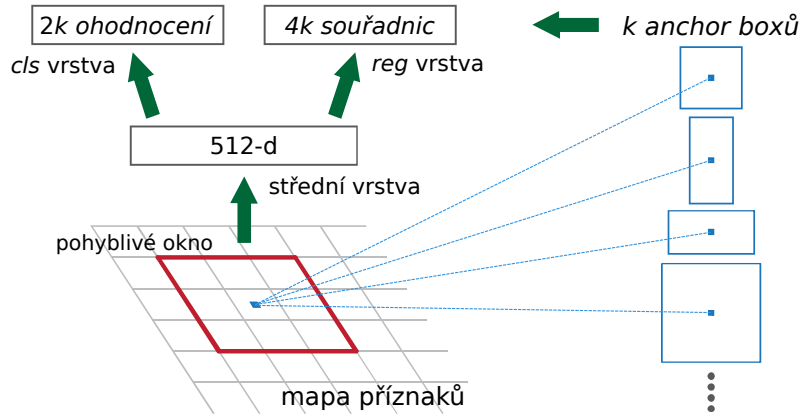
Faster R-CNN

Model Faster R-CNN [33] byl v polovině roku 2015 postaven na pozorování, že nejpomalejší částí detekce objektů v předchozích R-CNN modelech bylo samotné navrhování oblastí zájmu a jejich slučování pomocí selektivního vyhledávání. Výsledkem pozorování bylo vytvoření tzv. *sítě pro návrh oblastí* (RPN, Region Proposal Network).

Síť RPN sdílela konvoluční příznaky s detekční sítí, jakou byla například síť VGG. Za poslední sdílenou konvoluční vrstvu byla přidána malá síť, která prošla mapu příznaků $n \times n$ pohyblivým oknem a namapovala jí v případě sítě VGG do 512 rozměrného vektoru. Příznakový vektor byl poté předán dvěma sesterským, plně propojeným vrstvám – regresní a klasifikační. Pro každou pozici pohyblivého okna byla také vygenerována množina možných oblastí na základě tzv. *anchor boxů*, což jsou výchozí bounding-boxy pevně daných velikostí. Každý z těchto anchor boxů byl poté vyhodnocen, zda je možné, že se v něm bude objekt nacházet, či nikoliv. Každá takto vygenerovaná oblast poté obsahovala ohodnocení, do jaké třídy objektů asi patří a také souřadnice bounding-boxů. RPN síť je pro ilustraci uvedena na obrázku 3.3.

Obecný postup se dá tedy shrnout asi tak, že obrázek byl předán přímo síti RPN, která provedla extrakci příznaků a pomocí nich navrhla oblasti zájmu. Navržené oblasti zájmu byly poté předány RoI poolingové vrstvě a klasifikace objektů byla společně s predikcí bounding-boxů provedena prakticky stejným způsobem, jako u modelu Fast R-CNN.

Model Faster R-CNN odstraněním selektivního vyhledávání a dalšími změnami získal značné zrychlení oproti předchozímu modelu Fast R-CNN. Zpracování obrázku a získání predikcí objektů zvládl provést za méně než 0,2 sekundy. Díky této rychlosti bylo model Faster R-CNN teoreticky možné použít pro detekce v reálném čase, nicméně pro plynulejší získávání výsledků bylo zapotřebí tuto rychlost ještě zvýšit.



Obrázek 3.3: Síť pro návrh oblastí (RPN, Region Proposal Network). RPN posuvným oknem 3×3 prochází mapu příznaků a každou pozici namapuje do 512 rozměrného vektoru. Příznakový vektor je dále předán dvěma plně propojeným vrstvám cls a reg. k je maximální možný počet návrhů pro každou pozici. Cls vrstva obsahuje $2k$ pravděpodobnostních hodnot výskytů objektu v dané oblasti. Reg vrstva obsahuje $4k$ kódovaných souřadnic, které vyjadřují *posun oproti anchor boxům*. Obrázek a text přejat z článku [33].

3.2.2 Model Single Shot MultiBox Detector

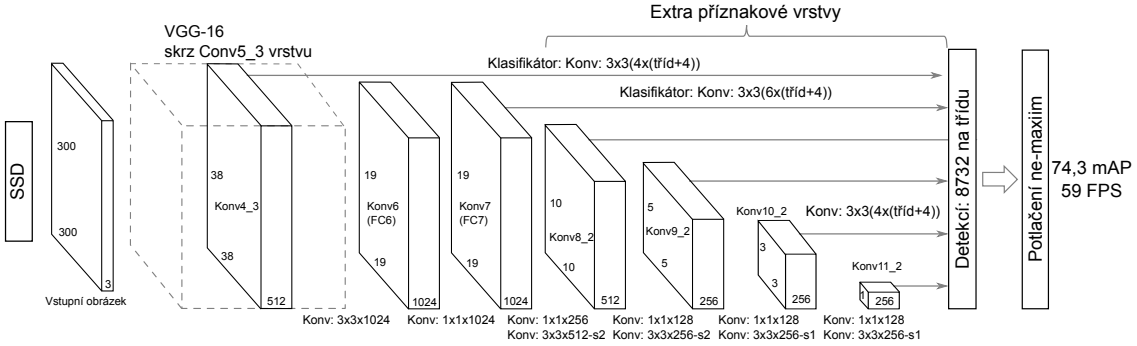
Model Single Shot Detector (zkráceně SSD) [21] z roku 2016 je díky své jednofázové architektuře, ve které jsou predikce bounding-boxů a klasifikace objektů provedeny v jediném průchodu neuronovou sítí, mnohem rychlejší, než Faster R-CNN.

Architektura modelu SSD je k vidění na obrázku 3.4. Joyce Xu [46] popisuje tuto architekturu tak, že její základ tvoří některý z existujících klasifikačních modelů, který extrahuje ze vstupního obrázku příznakové mapy a z kterého jsou odstraněny plně propojené vrstvy. V případě modelu SSD je to nejčastěji model VGG, Inception (GoogLeNet), RetinaNet či MobileNet. Za tímto klasifikačním modelem se poté nachází konvoluční vrstvy vytvářející příznakové mapy různých rozměrů. Rozměry těchto příznakových map postupně klesají z původní velikosti 34×34 až na nejmenší 1×1. Myšlenka vedoucí k takovému řešení je totiž taková, že pro malé objekty není nutné používat velká receptivní pole a naopak.

Pro každou pozici v každé mapě příznaků se vytvoří pomocí konvolučního filtru velikosti 3×3 bounding-boxy různých tvarů a velikostí, obdobně jako u modelu Faster R-CNN na obrázku 3.3. Tímto vznikne velké množství bounding-boxů, ve kterých se objekt vůbec nenachází, nebo se nachází jen částečně. Z tohoto důvodu se na konci neuronové sítě nachází závěrečný krok, kterému se říká *potlačení ne-maxim*. V tomto kroku jsou bounding-boxy podobných tvarů a velikostí porovnávány mezi sebou a výsledkem je, že až na výsledný bounding-box s nejvyšší shodou, jsou všechny ostatní zahozeny.

Při trénování se tento bounding-box s nejvyšší shodou porovnává s anotací (Ground Truth boxem) daného obrázku z trénovací sady. Pro toto porovnávání se používá technika známa jako Intersection Over Union (IoU). Tato technika bude ještě podrobněji popsána v kapitole 6.1.1.

Model SSD dosáhl díky svému přístupu při rozlišení vstupního obrázku 300×300 pixelů na datové sadě VOC2007 přesnosti 74,3 % mAP a rychlosti 59 FPS (snímků za sekundu) a při rozlišení 512×512 pixelů přesnosti 76,8 % mAP a rychlosti 22 FPS. Model SSD byl tedy podobně přesný, jako model Faster R-CNN, ale dosahoval mnohonásobně vyšších rychlostí, díky čemuž mohl být použit pro detekce objektů v reálném čase.



Obrázek 3.4: Architektura modelu SSD. Základ tvoří klasifikační neuronová síť, která provede extrakci příznaků a vytvoří příznakovou mapu 34×34. Za touto mapou se poté nachází série konvolučních vrstev, které vytváří postupně se zmenšující příznakové mapy, pomocí kterých se poté vytváří detekce objektů různých velikostí. Velké množství bounding-boxů získaných v každé příznakové mapě, je poté zpracováno v posledním kroku, kterému se říká *potlačení ne-maxim*. V tomto kroku se odstraní všechny bounding-boxy, až na jeden výsledný s nejvyšší shodou.

3.2.3 Model You Only Look Once

Model YOLO (You Only Look Once) [29] funguje na stejném principu, jako model SSD z předchozí podkapitoly 3.2.2 – celý obrázek je předán konvoluční neuronové síti a detekce objektů z něj získány v jediném průchodu touto sítí. Spadá tedy do stejné kategorie jednofázových regresních modelů. Během tohoto průchodu, ať už při trénování či při testování, YOLO vidí celý obrázek. Díky tomu je možné síť naučit i kontext, v jakém se objekt objevuje. Model YOLO byl představen v průběhu let v několika verzích, které zde budou podrobněji popsány.

Všechny verze modelu YOLO byly navrhnuty a trénovány open-source frameworkem pro neuronové sítě Darknet [28]. Tento framework byl napsaný v jazycích C a CUDA, což umožnilo provádět rychlé výpočty jak na CPU, tak hlavně na GPU.

Důležité je také říci, že YOLO používá vlastní formát anotací bounding-boxů [23]. Tento formát y je definován jako

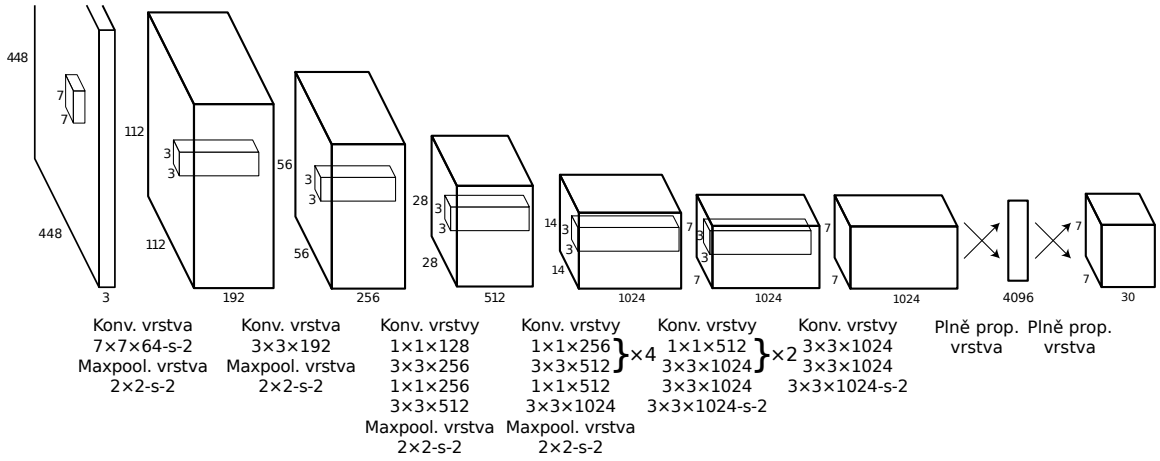
$$y = (p_c, b_x, b_y, b_h, b_w, c), \tag{3.1}$$

kde p_c je pravděpodobnost výskytu objektu v tomto bounding-boxu, (b_x, b_y) jsou souřadnice středu bounding-boxu v obrázku, (b_h, b_w) jsou výška a šířka bounding-boxu vzhledem k celkové velikosti obrázku a c je číslo, které reprezentuje třídu objektu. Souřadnice středu společně s výškou s šířkou bounding-boxu jsou normalizovány s výškou a šířkou obrázku, jejich hodnoty jsou tedy reálná čísla spadající do intervalu 0–1.

YOLO (v1)

První verze YOLO [30, 43, 15] byla představena v roce 2015 a dala základ všem dalším verzím, které na ní navazovaly.

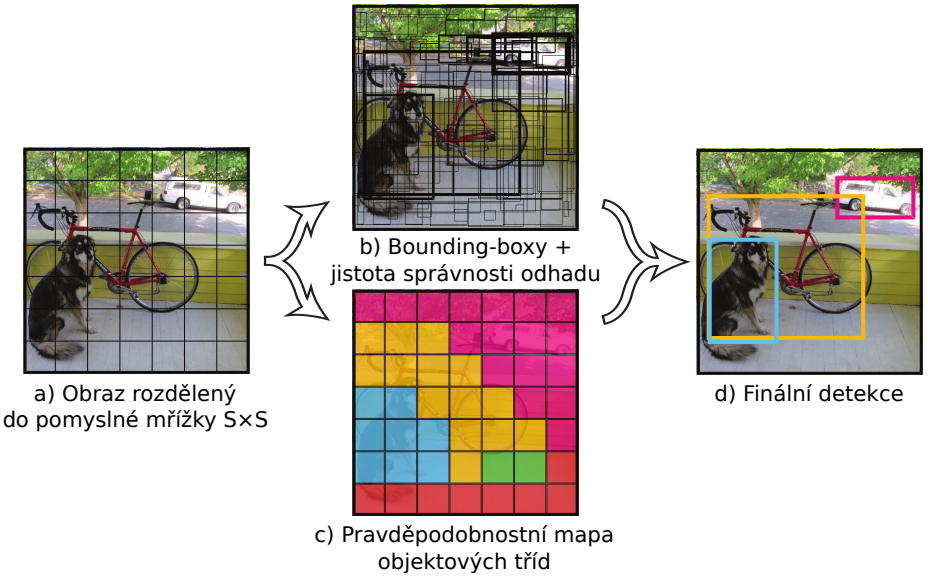
Celá detekce objektů probíhá v jediné neuronové síti, jejíž architektura je uvedena na obrázku 3.5. Tato síť je inspirovaná klasifikační sítí GoogLeNet a skládá se z 24 konvolučních a 2 plně propojených vrstev. Průběh získání predikcí objektů, který je naznačen na obrázku 3.6, se dá shrnout do několika částí:



Obrázek 3.5: Architektura modelu YOLO(v1). Neuronová síť je inspirovaná modelem GoogLeNet a skládá se z 24 konvolučních a 2 plně propojených vrstev. Konvoluční vrstvy extrahují příznaky, plně propojené vrstvy predikují pravděpodobnostní hodnoty a souřadnice výsledných bounding-boxů. Zdroj [30].

- V první části je vstupní obraz pomyslně rozdělen do mřížky o velikosti $S \times S$ polí, přičemž každé pole této mřížky je zodpovědné za detekci objektu, jehož střed se v něm nachází.
- V druhé části se v každém poli pomyslné mřížky predikuje B bounding-boxů společně s pravděpodobnostmi výskytů objektů v těchto bounding-boxech a hodnotou toho, jak si je model jistý správností svého odhadu pozice detekovaného bounding-boxu oproti anotovanému bounding-boxu. Tyto dvě hodnoty se mezi sebou vynásobí, čímž je získána celková hodnota jistoty odhadu pozice daného bounding-boxu. Pokud některé z polí žádný objekt neobsahuje, predikuje jen pár bounding-boxů a to s velmi nízkou pravděpodobností výskytu objektu – ideálně nulovou. Výstupem této části je mapa objektů, ve které jsou jednotlivé bounding-boxy seřazené podle toho, jak si je model jistý svou predikcí. Díky tomu lze odhadnout, kde se objekty nachází, ale stále ještě není známé, co to je za objekty.
- V další části se tedy provádí klasifikace těchto objektů. V Každém poli je predikováno C pravděpodobnostních hodnot – jedna pro každou objektovou třídu. Pokud se tedy v daném poli vyskytuje nějaký objekt, bude to ten objekt, jehož třída byla predikována s nejvyšší pravděpodobnostní hodnotou. Tímto vznikne pravděpodobnostní mapa objektových tříd.
- V poslední části se v každém poli hodnota pravděpodobnosti predikované třídy objektu vynásobí s pravděpodobnostní hodnotou predikovaných bounding-boxů a vytvoří se cílová mapa pravděpodobnostních tříd všech detekovaných objektů. Poté už je jen zvolena vhodná prahová hodnota, pomocí které se odfiltrují všechny detekce s pravděpodobnostní hodnotou nižší, než je tato hodnota (potlačení ne-maxim). Tímto jsou získány všechny požadované finální detekce.

Finální predikce jsou kódovány jako tenzor $S \times S \times (B * 5 + C)$. Obraz je v případě YOLOv1 rozdělen do mřížky 7×7 a obvykle jsou predikovány 2 bounding-boxy pro každé pole,



Obrázek 3.6: Princip detekce objektů modelem YOLO. a) Vstupní obrázek je rozdělen do pomyslné mřížky. Každé pole mřížky má na starost objekt, jehož střed se v ní nachází. b) V každém poli mřížky, pokud podle YOLO obsahuje objekt, jsou predikovány bounding-boxy společně s celkovou hodnotou jistoty odhadu daného bounding-boxu. V obrázku je míra jistoty naznačena tloušťkou rámečku bounding-boxu. c) V každém poli je predikována nejpravděpodobnější třída objektu, který se v poli nachází. Různé nejpravděpodobnější třídy jsou naznačeny různými barvami. d) Součinem předchozích pravděpodobností jsou získány finální detekce, které jsou vyfiltrovány pomocí prahové hodnoty. Zdroj [30].

kde se nachází nějaký objekt. Počet tříd už závisí na konkrétní úloze, ale za předpokladu, že tříd objektů bude 20 (datová sada VOC), výsledná predikce bude tenzor $7 \times 7 \times 30$.

Model YOLO má několik omezení, jedno z hlavních je detekce většího počtu menších objektů poblíž sebe – například hejno ptáků na obloze. Tento problém spočívá v tom, že v každém boxu mohou být detekovány pouze dva objekty a to ještě stejné třídy. Další omezení, které způsobuje nesprávné lokalizace, vzniká kvůli tomu, že YOLO se chová k chybám ve velkých bounding-boxech stejně jako k chybám v malých. Zatímco ve velkém bounding-boxu se tato chyba relativně ztratí, v malém má podstatně větší dopad na přesnost detekce.

YOLO dosahuje na datové sadě VOC 2007 přesnosti 63,4 %mAP při rychlosti 45 FPS. Tímto výsledkem v době svého vzniku překonal všechny další existující modely. Autoři tohoto modelu se zároveň pokusili vytvořit i YOLO založené na klasifikační síti VGG. Takto upravená síť je sice přesnější, než síť GoogLeNet, ale pomalejší. Tomu odpovídal i výsledek – přesnost 66,4 % mAP a rychlost 21 FPS.

Současně s modelem YOLO(v1) byl vytvořen i v té době nejrychlejší detekční model Fast YOLO, který se od původního modelu liší jen v hloubce neuronové sítě. Místo 24 konvolučních vrstev jich používá pouze 9. V těchto vrstvách je zároveň méně filtrů. Díky tomu je tento model opravdu podstatně rychlejší, než původní model YOLO – na datové sadě VOC dosahuje rychlosti 155 FPS. Za tuto rychlost ale platí snížením přesnosti, která je 52,7 % mAP.

YOLOv2

Nová verze YOLOv2 [31, 15] z roku 2016 přinesla několik změn, které přispěly ke zvýšení přesnosti a rychlosti modelu. V této verzi se autoři zaměřili hlavně na detekce objektů, které se v předchozí verzi úspěšně detekovat nepodařilo, a také na zpřesnění lokalizací objektů.

První ze změn oproti standardnímu modelu YOLO je, že model YOLO je natrénován na obrázcích s nižším rozlišením z datové sady ImageNet a poté s vyšším rozlišením doladěn pro konkrétní úlohu. Model YOLOv2 mezi tyto kroky přidává jeden další – po natrénování sítě na datové sadě ImageNet je trénování spuštěno nad touto datovou sadou znovu, ale tentokrát s vyšším rozlišením. Teprve poté je provedeno doladění pro konkrétní úlohu.

U druhé změny se autoři zaměřili na to, že původní model predikuje bounding-boxy přímo. YOLOv2 se nechal inspirovat řešením pomocí anchor boxů jako u modelu SSD, tudíž byly z modelu odstraněny plně propojené vrstvy, které měly na starost predikce bounding-boxů.Hlavní nedostatek anchor boxů se týká toho, že jejich velikost je nastavována ručně, čímž se model připravuje o možnost lepšího startu při detekování. Proto autoři přicházejí s řešením v podobě *k-means clusteringu*, který prochází anotované bounding-boxy v trénovací datové sadě a vytváří soubor dimenzionálních clusterů, které jsou poté používány jako anchor boxy. Cílem je, aby středy těchto clusterů byly co nejblíže středům objektů a tím se minimalizoval počet posunů.

Třetí změna se týká použití *Batch normalizace* [16]. Ta je použita na každou konvoluční vrstvu a vede ke zlepšení regularizace neuronové sítě a ke zvýšení přesnosti, díky čemuž je možně z modelu odstranit dropout.

Jako další provedená změna se dá uvést ještě použití víceúrovňového trénování, které se týká trénování na různě velkých rozlišeních a vede opět k mírnému zvýšení přesnosti. Díky tomuto způsobu trénování je ale také možné provést změnu rozlišení pro detekci, čímž se dá docílit zvýšení rychlosti na úkor přesnosti a naopak. Jinými slovy, při nižším rozlišení vstupního obrázku bude model sice méně přesný, ale bude o to rychlejší.

Na zvýšení rychlosti YOLOv2 má také podíl změna jeho architektury, která je založena na mírně rychlejší a přesnější klasifikační síti **Darknet-19**, jejíž architektura je naznačena v tabulce 3.1.

Výsledkem všech těchto změn je, že model YOLOv2 na datové sadě VOC 2007 při rozlišení 288×288 pixelů dosáhl přesnosti 69,0 % mAP a rychlosti 91 FPS a při rozlišení 544×544 pixelů dosáhl zvýšení přesnosti na hodnotu 78,6 % mAP při zachování rychlosti 40 FPS.

YOLOv3

Poslední verzí modelu YOLO je v době psaní této práce YOLOv3 [32, 15] z roku 2018. Tato verze opět přinesla několik změn.

První změna se týká predikcí objektových tříd. Předcházející modely doposud pracovaly s tím, že jednotlivé objekty se vzájemně vylučují a tudíž objekt mohl patřit jen do jedné třídy. V této verzi už může objekt spadat do více tříd zároveň, například takový automobil může patřit do tříd „Dopravní prostředek“, „Automobil“ a „Audi“. Jelikož v tomto případě suma pravděpodobností může být větší než 1, Softmax funkce je odstraněna a je nahrazena logistickými klasifikátory, které pravděpodobnost spočítají pro každou objektovou třídu zvlášť.

Další změna se týká predikce bounding-boxů. Zde YOLOv3 počítá pravděpodobnostní hodnotu pro bounding-box pomocí logistické regrese. Pokud se anchor box překrývá s ano-

Typ vrstvy	Počet filtrů	Velikost/Krok	Výstup
Konvoluční	32	3×3	224×224
Maxpooling		$2 \times 2/2$	112×112
Konvoluční	64	3×3	112×112
Maxpooling		$2 \times 2/2$	56×56
Konvoluční	128	3×3	56×56
Konvoluční	64	1×1	56×56
Konvoluční	128	3×3	56×56
Maxpooling		$2 \times 2/2$	28×28
Konvoluční	256	3×3	28×28
Konvoluční	128	1×1	28×28
Konvoluční	256	3×3	28×28
Maxpooling		$2 \times 2/2$	14×14
Konvoluční	512	3×3	14×14
Konvoluční	256	1×1	14×14
Konvoluční	512	3×3	14×14
Konvoluční	256	1×1	14×14
Konvoluční	512	3×3	14×14
Maxpooling		$2 \times 2/2$	7×7
Konvoluční	1024	3×3	7×7
Konvoluční	512	1×1	7×7
Konvoluční	1024	3×3	7×7
Konvoluční	512	1×1	7×7
Konvoluční	1024	3×3	7×7
Konvoluční	1000	1×1	7×7
Avgpooling		Globální	1000
Softmax			

Tabulka 3.1: Architektura klasifikační sítě Darknet-19. Zdroj [31].

tovaným bounding-boxem z trénovací datové sady víc než ostatní, jeho hodnota by měla být 1. Každému anotovanému bounding-boxu je přiřazen pouze jeden anchor box.

Při predikcích napříč úrovněmi jsou příznaky extrahovány z tří úrovní. Do nového modelu **Darknet-53**, jehož architektura je uvedena v tabulce 3.2, je přidáno několik konvolučních vrstev, které mají na starost predikce bounding-boxů a jejich pravděpodobnostních hodnot společně s pravděpodobnostními hodnotami tříd objektů. Následně je pomocí této sítě extrahována příznaková mapa z první a z druhé vrstvy, která je poté $2\times$ zvětšena. Další získaná příznaková mapa z modelu je poté spojena s touto zvětšenou příznakovou mapou. Tímto způsobem je možné z dřívější mapy příznaků získat významnější a jemnější sémantické informace.

Díky těmto změnám a dalším drobným vylepšením dosahuje tato verze YOLOv3 na datové sadě VOC 2007 přesnosti více než 80 % mAP a rychlosti okolo 40 FPS. Přesnost a rychlost se dá opět ovlivnit změnou rozlišení, stejně jako u YOLOv2.

Díky tomu, že jsou jak framework Darknet, tak všechny modely YOLO zveřejněny jako open-source, vznikají ze strany uživatelů nejrůznější modifikace těchto modelů. Nejznámější z nich[4] se prezentuje několika vylepšeními oproti standardní verzi:

- Podpora pro Microsoft Windows

[4]Darknet YOLOv3 pro Windows a Linux https://github.com/AlexeyAB/darknet

	Typ vrstvy	Počet filtrů	Velikost	Výstup
	Konvoluční	32	3×3	256×256
	Konvoluční	64	$3 \times 3 \,/\, 2$	128×128
$1\times$	Konvoluční	32	1×1	
	Konvoluční	64	3×3	
	Zbytková			128×128
	Konvoluční	128	$3 \times 3 \,/\, 2$	64×64
$2\times$	Konvoluční	64	1×1	
	Konvoluční	128	3×3	
	Zbytková			64×64
	Konvoluční	256	$3 \times 3 \,/\, 2$	32×32
$8\times$	Konvoluční	128	1×1	
	Konvoluční	256	3×3	
	Zbytková			32×32
	Konvoluční	512	$3 \times 3 \,/\, 2$	16×16
$8\times$	Konvoluční	256	1×1	
	Konvoluční	512	3×3	
	Zbytková			16×16
	Konvoluční	1024	$3 \times 3 \,/\, 2$	8×8
$4\times$	Konvoluční	512	1×1	
	Konvoluční	1024	3×3	
	Zbytková			8×8
	Avgpooling		Globální	
	Plně propojená		1000	
	Softmax			

Tabulka 3.2: Architektura klasifikační sítě Darknet-53. Síť se skládá z 53 konvolučních vrstev, které mají filtry o velikostech 3×3 a 1×1 a využívají zkratek. Zdroj [32].

- Zvýšení výkonosti neuronové sítě o přibližně 7 %, díky sloučení konvoluční a batch normalizace do jedné vrstvy
- Zvýšení výkonu $1,5\times$ u HD a $2\times$ u 4K videí
- Spousty paměťových a výpočetních optimalizací
- Vytvoření spousty pomocných nástrojů na nejrůznější úkony

K modelu YOLOv3 je také vytvořena jeho zjednodušená verze **YOLOv3-tiny**, která se skládá z 13 konvolučních vrstev. Z toho důvodu sice zdaleka nedosahuje takové přesnosti, jako původní model, ale oproti tomu vyniká svou rychlostí sahající přes 200 FPS.

3.3 Zhodnocení modelů

Při porovnávání všech dříve jmenovaných detekčních modelů byla hlavní pozornost směřována na celkovou přesnost a rychlost těchto modelů. Jelikož novější modely už testují svou přesnost na novější a pro detektory složitější datové sadě COCO, je souhrnné zhodnocení všech těchto modelů rozděleno do tabulek 3.3 a 3.4.

Ve výsledných hodnotách lze jasně vidět, že jednofázové detektory svými výsledky dominují nad jinými dvoufázovými modely. Ze všech vyjmenovaných modelů detekčních neuronových sítí byl zvolen model **YOLOv3-tiny**. Tento model sice zdaleka není tak přesný, jako ostatní modely, ale jeho rychlost a nízká výpočetní náročnost je dostatečnou kompenzací.

Model	Trénovací sada	Přesnost (mAP)	Přibližná rychlost (FPS)
R-CNN (AlexNet)	07	58,5	0,02
R-CNN (VGG)	07	66,0	0,02
Fast R-CNN	07	66,9	0,5
Fast R-CNN	07+12	70,0	0,5
Faster R-CNN	07	69,9	5
Faster R-CNN	07+12	73,2	5
Faster R-CNN	07+12+COCO	78,8	5
SSD 300	07	68,0	59
SSD 300	07+12	74,3	59
SSD 300	07+12+COCO	79,6	59
SSD 512	07	71,6	22
SSD 512	07+12	76,8	22
SSD 512	07+12+COCO	81,6	22
YOLO(v1)	07+12	63,4	45
YOLO(v1) (VGG)	07+12	66,4	21
Fast YOLO	07+12	52,7	155
YOLOv2 288	07+12	69,0	91
YOLOv2 416	07+12	76,8	67
YOLOv2 544	07+12	78,6	40

Tabulka 3.3: Výsledky detekčních modelů na testovací datové sadě VOC 2007. Číslovka za názvem modelu značí vstupní rozlišení. Zkratky trénovací sady: „07“ – datová sada VOC 2007, „12“ – datová sada VOC 2012, „COCO“ – datová sada COCO. Zdroje dat pro tabulku čerpány z článků [11, 10, 33, 21, 30, 31].

Model	Trénovací sada	Přesnost (AP_{50})	Přibližná rychlost (FPS)
Fast R-CNN	train	39,9	–
Faster R-CNN	trainval	45,3	–
SSD 300	trainval35k	41,2	59
SSD 512	trainval35k	46,5	22
YOLOv2 416	trainval35k	44,0	91
YOLOv2 608	trainval	48,1	40
Tiny YOLO	trainval	23,7	244
YOLOv3 320	trainval	51,5	45
YOLOv3 416	trainval	55,3	35
YOLOv3 608	trainval	57,9	20
YOLOv3-tiny	trainval	33,1	220

Tabulka 3.4: Výsledky detekčních modelů na datové sadě COCO. Číslovka za názvem modelu značí vstupní rozlišení. Data pro tabulku jsou přejata z článků [32, 48].

Kapitola 4

Datové sady

Pro to, aby byl výslední detekční model co nejpřesnější, bylo zapotřebí připravit dostatečně velkou datovou sadu pro trénování konvoluční neuronové sítě, která by obsahovala co nejrůznorodější obrázky automobilů s registračními značkami. Zároveň s ní bylo třeba vytvořit i vhodnou datovou sadu pro otestování natrénované neuronové sítě. Jelikož tato práce cílí na detekce značek evropského formátu, byla snaha vytvořit datovou sadu právě z nich.

V této kapitole jsou popsány výsledné datové sady, jakým způsobem byly pořízeny a jak dále upraveny, aby byly připravené na následné využití. Výsledné zmiňované datové sady a použité skripty jsou k dispozici v příloze na paměťovém mediu.

4.1 Nevhodné datové sady

Při vyhledávání volně dostupných existujících datových sad evropských registračních značek jsem v raném stádiu práce zjistil, že takových datových sad doposud moc neexistuje. Většina nalezených datových sad buď nebyla vůbec anotována [39], nebo obsahovala poznávací značky automobilů ze států mimo Evropskou unii [18].

Jiné případy datových sad obsahovaly pouze ořezané [49] nebo velmi přiblížené registrační značky[1]. Jejich ukázka je na obrázku 4.1[2]. Tyto značky se pro použití v modelu YOLO ukázaly jako nevhodné, jelikož jak bylo psáno v kapitole 3.2.3: YOLO se při trénování dívá na celý obrázek, díky čemuž je možné síť naučit i kontext, v jakém se objekt na obrázku objevuje. Pokud se tedy síti při trénování předají jen výřezy nebo pouhé části automobilů s registračními značkami, nebude další značky později správně detekovat.

4.2 Přejaté datové sady

První datová sada použitá pro trénování sítě se skládala z obrázků datové sady Vlasty Srebriće[3] [39], kterou bylo zapotřebí ručně anotovat. Ruční anotování bude ještě podrobněji popsáno v kapitole 4.3. Tato sada se skládá z více než 400 fotografií ve vysokém rozlišení, na kterých se nacházejí zaparkované automobily a nákladní vozy. Ukázky z této datové sady jsou uvedeny v obrázku 4.2. Díky této datové sadě detektor dobře detekuje velké registrační značky v popředí obrázku, jako jsou například ty patřící automobilům jedoucím v koloně před vozidlem, z kterého byl pořizován záznam.

[1] https://dataturks.com/projects/devika.mishra/Indian_Number_plates

[2] Části obrázku přejaty z https://www.flickr.com/photos/vladimir-911/

[3] Dospupné ke stažení z http://www.zemris.fer.hr/projects/LicensePlates/english/results.shtml

a) b)

Obrázek 4.1: Příklady nevhodných obrázků v datové sadě. Model YOLO se nebude moci naučit v jakém kontextu se značky na obrázcích běžně objevují. a) Ořezané registrační značky [49]. b) Velmi přiblížené registrační značky

Obrázek 4.2: Typické obrázky z datové sady Vlasty Srebriće [39].

Oproti tomu detekcím registračních značek menších rozměrů ve středních vzdálenostech velmi pomohla interní školní datová sada, která byla vytvořena výzkumnou skupinou GRAPH@FIT a která byla pro tuto práci poskytnuta Lukášem Teuerem a kol. [42]. Tato anotovaná datová sada se skládá z více než 20 000 snímků běžného provozu na rychlostní komunikaci. Ukázky jsou na obrázku 4.3. Jelikož tato datová sada pro zápis anotací používá formát `pickle`[4], bylo zapotřebí anotace extrahovat a transformovat je do formátu YOLO (viz jeho zápis 3.1 na straně 20). K tomuto účelu posloužil skript `parse_pkl.py`. Jelikož některé anotace z této datové sady nebyly vždy úplně přesné, musel jsem minimálně ty z testovací datové sady projít a nejhorší ručně upravit.

4.3 Anotované datové sady

Jelikož model stále nedokázal detekovat vzdálené malé registrační značky a další vhodné datové sady nebyly k nalezení, přistoupil jsem k ručnímu anotování nových datových sad.

Pro tento úkol byl ze začátku používán nástroj `YOLO-Annotation-Tool`[5]. Jelikož ale režie spojená s organizací dat pro tento nástroj byla příliš obtěžující, byl brzy nahrazen nástrojem `Yolo_mark`[6]. Tomuto nástroji stačí pouze předat složku se vstupními obrázky, soubor, kam se budou ukládat cesty k těmto obrázkům, a soubor se jmény objektových tříd. Poté už jen stačí klikáním myší anotovat a standardními klávesami procházet jednotlivé obrázky. Ukládání vytvořených anotací je prováděno automaticky.

[4] https://docs.python.org/3/library/pickle.html
[5] https://github.com/ManivannanMurugavel/YOLO-Annotation-Tool
[6] https://github.com/AlexeyAB/Yolo_mark

Obrázek 4.3: Obrázky z interní školní datové sady výzkumné skupiny GRAPH@FIT [42]

a) b) c)

Obrázek 4.4: Ukázky obrázků z ručně anotovaných datových sad. a) Část datové sady Laboratoře multimediálních technologií Národní technické univerzity v Athénách. b) Datová sada Matta Hilla. c) Datová sada vytvořená z videa ze serveru YouTube.

Pro potřeby této práce jsem ručně anotoval 3 přejaté datové sady – již zmíněnou datovou sadu Vlasty Srebriće [39], evropské registrační značky z datové sady Matta Hilla[7] a užitečné části datové sady Laboratoře multimediálních technologií Národní technické univerzity v Athénách[8], které jsou tvořeny opět převážně obrázky zaparkovaných automobilů. Dalších devět datových sad jsem pomocí skriptu `videoToFrames.py` vytvořil z různých volně dostupných videozáznamů z palubních kamer ze serveru YouTube. Tento skript z videozáznamu na vstupu vybere na základě zadaného čísla každý n-tý snímek a uloží ho do vybraného adresáře. Takto vytvořené datové sady jsem opět poté ručně anotoval.

Celkově jsem ručně anotoval přes 4 500 obrázků, které obsahují přes 8 000 registračních značek. O tyto anotované datové sady projevil zájem i další student, který je využil pro potřeby své diplomové práce. Ukázky ze zmiňovaných anotovaných datových sad jsou uvedeny na obrázku 4.4.

4.4 Augmentace dat

Pro potřeby experimentů z kapitoly 6.2 bylo zapotřebí vytvořit i augmentované datové sady. Augmentace v kontextu této práce je technika, která aplikuje různé transformace na obrázky z **trénovací** datové sady, za účelem jejího rozšíření. Typicky je využívána v případech, kdy je trénovací datová sada příliš malá a další data už nejsou k dispozici. Augmentace se aplikuje pouze na obrázky z trénovací datové sady, protože samotné testování modelu už je zapotřebí provádět na nezkreslených datech.

[7] https://github.com/openalpr/benchmarks/tree/master/endtoend/
[8] http://www.medialab.ntua.gr/research/LPRdatabase.html

a) b) c)

Obrázek 4.5: Vliv velikosti augmentace na podobu výsledného obrázku: a) Slabě augmentovaný obrázek, b) Středně augmentovaný obrázek, c) Silně augmentovaný obrázek.

Pro augmentaci dat se původně zdála jako vhodná Pythonovská knihovna `Augmentor`[9]. Při samotné augmentaci se objevilo několik problémů s aplikací výsledku augmentace na anotované bounding-boxy, kde v některých případech poloha i tvar bounding-boxu zůstaly nezměněny a v některých dalších případech augmentovaný bounding-box nepasoval na objekt v augmentovaném obrázku. Tyto problémy se naštěstí nevyskytovaly u jiné knihovny jménem `imgaug`[10], která byla nakonec pro augmentaci dat použita ve skriptu `image_augmentation.py`.

Jelikož jsem cílil na co možná nejrůznorodější augmentace obrázků, snažil jsem se provést všechny jejich transformace co nejvíce náhodně, proto tento skript postupně prochází obrázky z trénovací datové sady a na každý z nich aplikuje náhodný počet transformací v náhodném pořadí, přičemž každá transformace je provedena náhodnou silou a jen v určitém procentu případů. V rámci augmentace dat jsem použil tyto transformace obrázků:

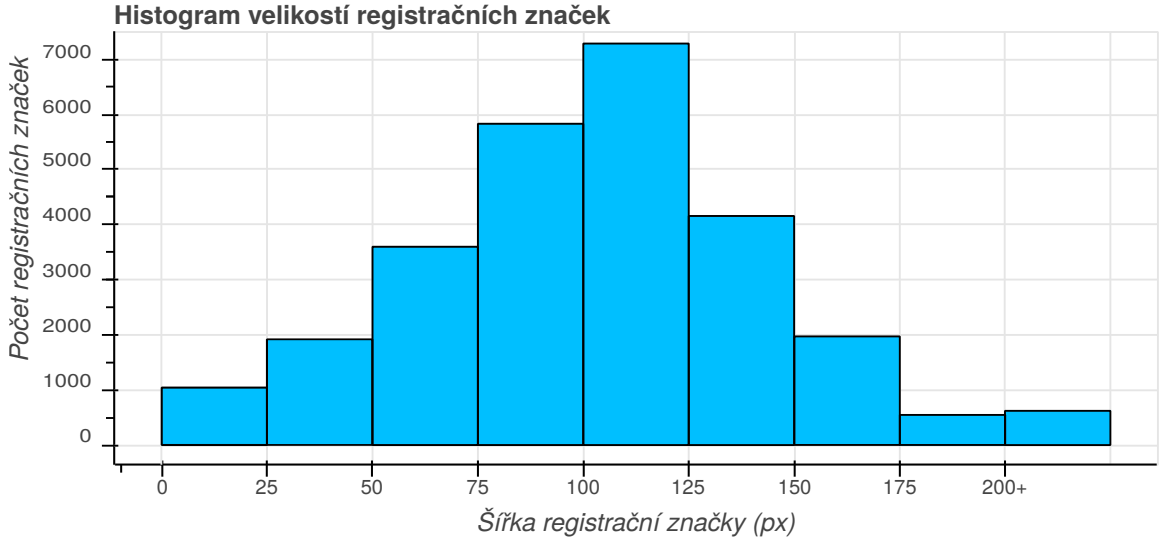
- Afinní transformace – změna měřítka, pootočení, posun a zkosení
- Zrcadlové otočení
- Gaussovské rozostření
- Zesvětlení nebo ztmavení
- Zesílení nebo zeslabení kontrastu
- Gaussovské rozostření jednotlivých barevných kanálů
- Zesvětlení nebo ztmavení jednotlivých barevných kanálů

Pro potřeby experimentu jsem vytvořil dohromady 3 augmentované datové sady: lehce, středně a silně augmentovanou. Ukázka jednoho z obrázku, který se vyskytuje ve všech třech augmentovaných sadách, je na obrázku 4.5.

4.5 Výsledná datová sada

Výsledná datová sada se skládá téměř z přibližně 46 000 obrázků a disponuje širokým rozsahem velikostí registračních značek. Ukázka histogramu jejich velikostí před augmentací dat je na obrázku 4.6. Tuto datovou sadu jsem původně oproti doporučení z kapitoly 2.2.3 z důvodu nedostatku dat rozdělil na trénovací, testovací a validační datovou sadu poměrem

[9] https://github.com/mdbloice/Augmentor
[10] https://imgaug.readthedocs.io/en/latest/



Obrázek 4.6: Histogram velikostí (šířky) registračních značek v trénovací datové sadě před augmentací obrázků. V datové sadě se převážně vyskytují středně velké registrační značky o šířce okolo 100 pixelů.

80:10:10. Později jsem ale zjistil, že model validační datovou sadu na nic nevyužívá, a proto jsem ji s úmyslem zlepšit výsledky trénování sloučil s trénovací datovou sadou. Výsledná trénovací datová sada se tedy skládá z přibližně 43 500 obrázků obsahujících přes 54 000 registračních značek a testovací datová sada, skládající se z téměř 2 500 obrázků, obsahuje přibližně 3 000 registračních značek.

Kapitola 5

Trénování vlastního detekčního modelu

V této kapitole je popsána příprava, spuštění a průběh trénování neuronové sítě, zároveň je zde uvedena i ukázka výsledku trénování společně s popisem několika způsobů, jak může být s detekcemi objektů naloženo. Ke trénování neuronové sítě je využitý dříve zmiňovaný framework `Darknet` [28] z kapitoly 3.2.3 s argumenty `detector train`, napsaný v jazycích C a CUDA.

5.1 Příprava a nastavení trénování neuronové sítě

Aby bylo možné aplikaci Darknet plně využívat, je zapotřebí mít nainstalovanou knihovnu OpenCV pro podporu práce s obrázky a vykreslování výsledků. Dále je zapotřebí mít nainstalovanou architekturu CUDA, která umožňuje spouštění aplikací napsaných v jazyce C/C++ nebo Python na vybraných grafických kartách společnosti NVIDIA. Tato společnost zároveň nabízí ke stažení knihovnu cuDNN, která je speciálně vytvořena pro akcelerace výpočtů spojených s hlubokými neuronovými sítěmi na jejích grafických kartách. Na mém operačním sytému Ubuntu 16.04 se ukázala jako funkční kombinace OpenCV 3.4.2[1], CUDA 9.0[2] a cuDNN Runtime+Developer pro tyto verze[3].

Po instalaci těchto nástrojů je zapotřebí ještě upravit Makefile soubor pro překlad aplikace. Na prvních třech řádcích je potřeba povolit přepsáním hodnoty 0 na 1 trénování na grafické kartě, akceleraci výpočtů pomocí knihovny cuDNN a využívání knihovny OpenCV pro zpracování obrázků a vykreslení výsledků. Zároveň bylo nutné v Makefile souboru přidat vepsáním řádku „`-gencode arch=compute_61,code=[sm_61,compute_61]`“ do seznamu podporovaných architektur podporu novějších grafických karet NVIDIA GeForce GTX řady 10.

Architektury neuronových sítí jsou implementovány pomocí konfiguračních souborů, kde se také nachází nastavení těchto sítí. V případě modelu YOLOv3-tiny se jedná o konfigurační soubor `spz_train_1440.cfg`. V tomto souboru jsem upravil několik řádků:

- Na řádcích 6 a 7 je změněna velikost dávky(batch) a rozdělení (subdivision) na hodnoty 24 a 8. Tyto hodnoty se ukázují jako vhodné pro rychlejší trénování, jsou ale

[1] https://github.com/opencv/opencv/releases/tag/3.4.2
[2] https://developer.nvidia.com/cuda-90-download-archive
[3] https://developer.nvidia.com/rdp/cudnn-archive

více paměťově náročné. Další snížení velikosti dávky už vede k pádům trénování. U testování byly hodnoty dávky a rozdělení ponechány na 1 a 1.

- Na řádcích 8 a 9 je zvýšeno vstupní rozlišení obrázku na rozlišení 1440×1440, čímž je dosaženo přesnějších detekcí. Toto zvýšení rozlišení má ale za následek snížení počtu zpracovaných snímků za sekundu. Toto odpovídá i výsledkům z tabulky 3.4, kde zvýšení vstupního rozlišení většinou vede ke zvýšení přesnosti modelu na úkor rychlosti.

- Na řádcích 127 a 171 je zapotřebí upravit počet filtrů v konvolučních vrstvách. Počet těchto filtrů se odvozuje ze vztahu

$$\text{počet filtrů} = (\text{počet objektových tříd} + 5) * 3. \tag{5.1}$$

  Jelikož objektová třída, kterou síť bude detekovat, je v této práci pouze jedna (registrační značka), jsou počty filtrů nastaveny na hodnotu 18.

- Na řádcích 135 a 177 je zapotřebí také nastavit počet objektových tříd, což už je, jak bylo řečeno, 18.

Před spuštěním trénování je ještě důležité vytvořit datový soubor, který obsahuje další pro síť potřebné informace. Abych nemusel po každé změně datové sady vytvářet tento soubor znovu, upravil jsem soubor `detector.c` tak, aby cestu k trénovací a testovací datové sadě přijímal přes argumenty příkazové řádky, stejně jako cestu k adresáři, kam budou ukládány natrénované váhy. V datovém souboru se tedy očekávají pouze informace o počtu objektových tříd, které síť bude detekovat, cestu k souboru `spz.names` se jmény těchto objektových tříd a v jakém formátu budou zpracovávána validační data. V případě této práce se jedná o soubor `universal.data`, kde jsou všechny tyto informace vyplněny a není potřeba už ho dále měnit.

5.2 Spuštění trénování

Samotné trénování neuronové sítě je časově dlouhodobá a na výpočetní výkon náročná úloha, pro kterou je vhodné mít co nejvýkonnější grafickou kartu, aby se čas trénování co nejvíce snížil. Jelikož ale takovou kartou nedisponuji, pro trénování jsem využil nabízených služeb virtuální organizace MetaCentrum[4]. Tato organizace nabízí poskytnutí výpočetní a úložné kapacity pro všechny uživatele z akademického prostředí České republiky, díky čemuž jsem obdržel přístup k výpočtům na výkonných grafických kartách NVIDIA Tesla K20Xm 6GB, NVIDIA GeForce GTX 1080 Ti nebo nVidia Tesla K20 5GB, což umožnilo značné urychlení celého průběhu trénování. Trénování všech modelů zabralo na výpočetním centru MetaCentrum přes 454 dnů procesorového času.

5.2.1 Požadavek na poskytnutí zdrojů a vytvoření úlohy v MetaCentru

Aby bylo možné spustit trénování na některém ze strojů MetaCentra, je nejdříve zapotřebí vytvořit požadavek na rezervaci výpočetních a úložných zdrojů [45], který je poté zaslán plánovači úloh. Tento plánovač požadavek vyhodnotí, nalezne nejvhodnější výpočetní clustery a úlohu umístí do některé z front, které tyto clustery obsluhují. Následně se po

[4] https://metavo.metacentrum.cz/

Argument	Požadavek
-l select=(počet)	přiřazení určitého počtu „chunků“[5]
-l ncpus=(počet)	přiřazení určitého počtu procesorů
-l ngpus=(počet)	přiřazení určitého počtu grafických karet
-l mem=(kapacita)	přiřazení operační paměti.
-l scratch_local=(kapacita)	přiřazení fyzické paměti na výpočetním uzlu
-l gpu_cap=cuda35	přidělení uzlů, který disponuje CUDA verze 35
-l walltime=((hh:)mm:)ss	maximální délka trvání úlohy[6]
-l cluster=(název clusteru)	přidělení výpočetního uzlu z konkrétního clusteru
-q (název fronty)	zařazení do konkrétní fronty úloh
-I	práce v interaktivním režimu
cesta k dávkovému souboru	provedení úlohy

Tabulka 5.1: Používané argumenty plánovače úloh MetaCentra. Argumenty přepínače -l je možné sloučit do jednoho, v tom případě je ale zapotřebí oddělit je pomocí dvojtečky – např. `qsub -l select=1:ncpus=3:ngpus=1... skript.sh`. U přepínače `-q` se pro potřeby této práce využívá buď fronta gpu, nebo fronta gpu_long, které zajišťují obsluhu clusterů s grafickými kartami. Fronta gpu_long se od druhé liší tím, že umožňuje provádět výpočet úlohy až 168 hodin namísto 24.

příchodu na řadu automaticky postará o spuštění úlohy na uvolněném výpočetním uzlu, její běh i vrácení vyprodukovaných výsledků.

Podle návodu [45], jak vytvořit výpočetní úlohu, je ze všeho nejdříve zapotřebí na některém z čelních uzlů vytvořit dávkový skript. Tento skript se skládá z několika příkazů, kde mezi nejdůležitější patří načtení potřebných aplikačních modulů – pro trénovací úlohu jsou to moduly `cuda-9.0` a `cudnn-7.0`. Dále se zde nachází příkaz pro překopírování potřebných dat přímo do paměti výpočetního uzlu, tzv. `SCRATCH` úložiště, aby nedocházelo ke zdržování výpočtů a zatěžování sítě při práci s daty. Nejdůležitější příkaz je samotné spuštění trénování neuronovou sítí aplikací `darknet` s argumenty `detector train`.

Po vytvoření dávkového skriptu už je možné požádat o rezervaci zdrojů. O tyto zdroje se žádá pomocí příkazu `qsub` a jeho přepínačů. Ty, které byly použity při řešení práce jsou uvedeny v tabulce 5.1.

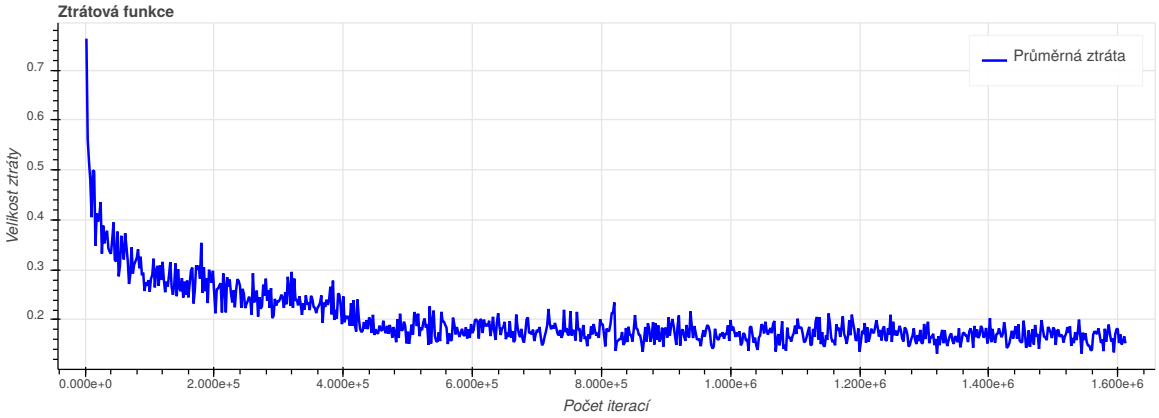
5.3 Průběh a výsledek trénování

Aby bylo možné sledovat stav trénování, rozšířil jsem trénovací funkci v souboru `detector.c` o výpis aktuálních a průměrných hodnot ztrátové funkce do souboru. Tento soubor je poté zpracován pythonovským skriptem `loss_graph.py`, který data zpracuje a vykreslí graf ztrátové funkce. Příklad jedné z nich je uveden na obrázku 5.1.

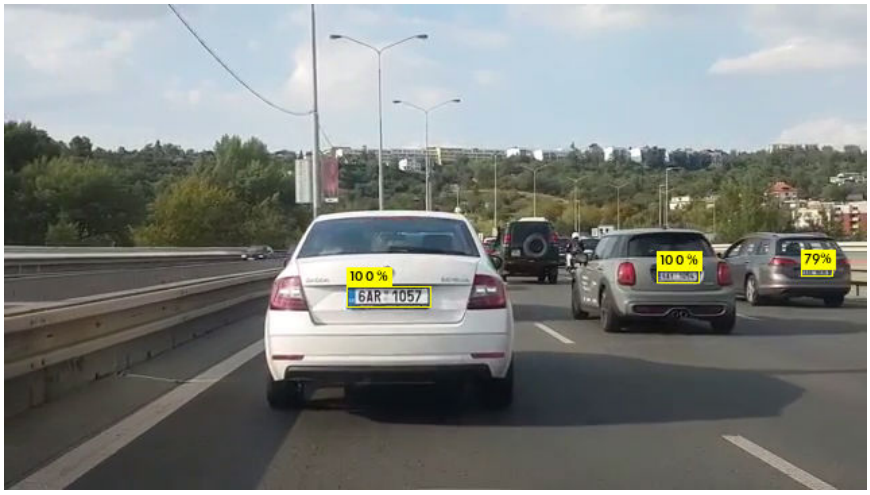
Výstupem trénování jsou váhové soubory, kde je každý uložen po určitém počtu iterací do zadaného adresáře. Vybraný váhový soubor je poté možné společně buď se vstupním

[5] „Chunk“ se dá popsat i jako další stroj. Tímto je možné provádět různé paralelní výpočty na více strojích zároveň.

[6] Po vypršení tohoto času bude běžící úloha přerušena.



Obrázek 5.1: Křivka průměrných výsledků chybové funkce v každých dvou tisících iterací. Z obrázku je patrný typický tvar ztrátové funkce – ze začátku hodnoty chyb prudce klesají a poté postupně konvergují ke svému minimu. Z obrázku je také patrné, že nedošlo k přetrénování sítě, tudíž je zde možnost síť ještě nějakou dobu trénovat.



Obrázek 5.2: Příklad úspěšných detekcí registračních značek modelem YOLOv3-tiny.

obrázkem předat aplikaci `darknet` s argumenty `detector test`, nebo s videozáznamem s argumenty `detector demo`. Aplikace poté provede v obrázku nebo jednotlivých snímcích videozáznamu detekce objektů a pomocí knihovny OpenCV je v obrázku označí. Příklad výstupního obrázku s označenými detekcemi registračních značek je uvedeny na obrázku 5.2.

Aplikaci `darknet` jsem dále úpravou souborů `demo.c`, `image.c` a `image_opencv.cpp` rozšířil o argumenty `detector save_detections`. Aplikace při spuštění s těmito argumenty provádí detekce objektů stejně jako s argumenty `detector demo`, jen s tím rozdílem, že v případě, kdy je ve snímku detekována registrační značka, je snímek z videozáznamu extrahován a uložen. Zároveň se vytvoří i soubor s anotovaným bounding-boxem. Je možné si pomocí přepínače zvolit, zda pomocí OpenCV knihovny bude bounding-box do obrázku vykreslen, či nikoliv. Stejně tak je možné si dalším argumentem nastavit, kolik snímku se má přeskakovat, aby se poté do výsledků neukládaly snímky s jednou a tou samou registrační značkou. Detekční model po této úpravě může být využit například pro tvorbu

nových anotovaných datových sad nebo pro ukládání detekovaných registračních značek na pozdější zpracování.

Další rozšíření o argumenty `detector export`, které jsem vytvořil, umožnilo exportovat ze vstupního videozáznamu jednotlivé snímky s označenými detekcemi nebo z těchto snímků vytvořit opět videozáznam. Toto rozšíření jsem použil například pro vytvoření prezentačního videa v příloze.

Kapitola 6

Validace natrénovaného modelu

Aby bylo možné zjistit, jaká prahová hodnota vede k nejlepším výsledkům či jaká je výsledná přesnost a chybovost natrénovaného modelu, je zapotřebí provést jeho validaci nad obrázky z testovací datové sady. Tato datová sada, jak už bylo zmíněno v kapitole 4.4, by neměla obsahovat žádná augmentovaná data. Z toho důvodu jsem vytvořil testovací datovou sadu, která obsahuje co nejvíce obrázků ze situací, do kterých bude model později skutečně nasazen. Pro účely validace jsem vytvořil pythonovský skript `roc_validate.py`, který implementuje veškerá měření a experimenty, které jsou v této kapitole zmiňovány.

6.1 Metodiky hodnocení

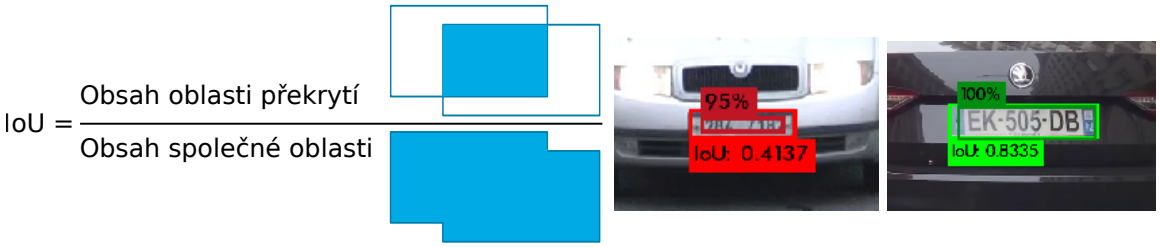
Pro validaci používám několik druhů metodik, které mají za úkol zjistit a vizualizovat, jak dobře model provádí detekce s různými prahovými hodnotami či jak jsou výsledné detekce přesné. Použité metodiky jsou popsány v následujících podkapitolách a informace k nim jsou čerpány z článku [8] a z encyklopedie [36].

6.1.1 Intersection Over Union

Tato metodika, jež se zkráceně označuje jako IoU, je použita pro určení přesnosti detekce. Bounding-boxy detekovaného a anotovaného objektu jsou mezi sebou pomocí rovnice v obrázku 6.1 porovnány, čímž je vypočítáno, jak moc se jejich pozice od sebe liší. Cílem je samozřejmě to, aby se detekovaný bounding-box co nejvíce překrýval s anotovaným bounding-boxem, tj. aby se hodnota IoU blížila co nejvíce k hodnotě 1.

Pokud nastane takový případ detekce, že hodnota IoU bude nižší, než je daná prahová hodnota, bude tato detekce prohlášena jako neúspěšná a bude s ní zacházeno jako s chybou. V rámci této práce jsem jako prahovou hodnotu IoU zvolil hodnotu 0.5, tj. anotovaný a detekovaný bounding-box se musí alespoň z poloviny překrývat, aby byla detekce považována za úspěšnou.

Vizualizaci úspěšnosti a neúspěšnosti jsem vytvořil úpravou několika souborů aplikace `Darknet`, z největší části úpravou testovací funkce v souboru `detector.c`. Pro detekovaný bounding-box je ze všeho nejdřív nalezen odpovídající anotovaný bounding-box. Ze souřadnic těchto bounding-boxů je spočítána hodnota IOU. Anotovaný i detekovaný bounding-box jsou následně pomocí knihovny OpenCV zakresleny do obrázku, včetně hodnoty IOU. Pokud je IOU větší než 0.5, jsou bounding-boxy zakresleny zeleně, v opačném případě červeně (viz obrázek 6.1) a s detekcí je zacházeno jako s chybnou.



Obrázek 6.1: Rovnice výpočtu Intersection over Union s ukázkami obrázků. Jasnějšími barvami jsou označeny anotované bounding-boxy, tmavšími barvami jsou označeny bounding-boxy detekovaných objektů. Čím více se anotovaný a detekovaný bounding-box překrývají, tím je hodnota IoU vyšší. Inspirace převzata z článku [25].

	Anotovaný: +	**Anotovaný: -**
Detekovaný: +	TP	FP
Detekovaný: -	FN	TN

Tabulka 6.1: Příklad matice záměn na anotovaných a detekovaných bounding-boxech. Ve sloupcích jsou uvedeny informace o tom, jaké mají být výsledky, a v řádcích jsou naopak informace o tom, jakých výsledků dosáhl model.

6.1.2 Matice záměn

Základní metodikou pro vizualizaci správnosti detekcí je matice záměn (známá také jako chybová matice), která je zde naznačena tabulkou 6.1. Tato 2×2 matice v sobě obsahuje počty správných a chybných detekcí, které jsou rozděleny do následujících tříd:

Skutečně pozitivní detekce neboli True Positive (TP) je první typ správné detekce – v obrázku se nachází registrační značka a byla úspěšně nalezena.

Skutečně negativní detekce neboli True Necgative (TN) je druhý typ správné detekce – v obrázku se nenachází žádná registrační značka a model žádnou nenašel

Falešně pozitivní detekce neboli False Positive (FP) je tzv. Chyba typu I – model označil jako registrační značku něco, co registrační značkou není (např. nápis na automobilu).

Falešně negativní detekce neboli False Negative (FN) je tzv. Chyba typu II – v obrázku se nachází registrační značka, ale model ji nedokázal detekovat.

Cílem je, aby se v tabulce nacházel co nejvyšší počet TP a TN tříd a co nejnižší počet FP a FN tříd, v takovém případě model dosahuje nejvyšší přesnosti. Pokusy snížit počet chyb typu I vedou obvykle k navýšení počtu chyb typu II a naopak. Z tohoto důvodu je nutné si jednu z těchto chybových tříd vybrat na úkor druhé. Při rozhodování dost záleží na povaze úkolu. Jelikož pro detekce registračních značek je rozhodně přijatelnější občas nějakou značku nedetekovat, než detekovat jako značku něco, co značkou není, zaměřil jsem se na co největší snížení počtů chyb typu I tj. na co nejmenší počet falešných detekcí.

Pomocí výsledných hodnot tříd se dá poté vypočítat celá řada statistických údajů, jako je mimo jiné **senzitivita** (True Positive Rate, TPR)

$$\text{senzitivita} = \frac{TP}{TP + FN}, \tag{6.1}$$

která říká, s jakou pravděpodobností model správně detekuje registrační značku, pokud se nějaká na obrázku nachází, **specificita** (True Negative Rate, TNR)

$$\text{specificita} = \frac{TN}{TN + FP}, \tag{6.2}$$

která říká, s jakou pravděpodobností model správně registrační nedetekuje, pokud se na obrázku žádná registrační značka nenachází, **celková přesnost** (Accuracy, Acc)

$$\text{celková přesnost} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{6.3}$$

která udává, kolikrát ze všech detekcí provede model detekci správně, nebo **celková chyba** (Error, Err)

$$\text{celková chyba} = \frac{FP + FN}{TP + TN + FP + FN} = 1 - \text{celková přesnost}, \tag{6.4}$$

která udává, kolikrát ze všech detekcí provede model detekci špatně.

6.1.3 ROC křivka

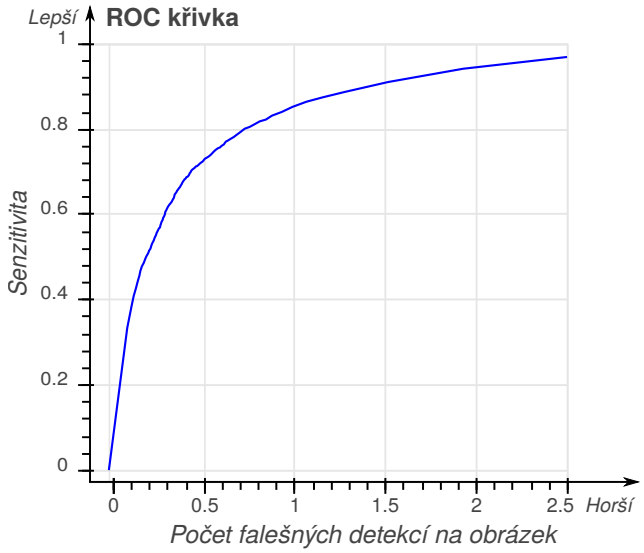
V této práci používám lehce pozměněnou křivku ROC, jejíž příklad je na obrázku 6.2. ROC křivka podle Marslanda [24] slouží pro vizualizaci vztahu mezi počtem falešných detekcí na obrázek na ose x a senzitivitou na ose y. Jelikož provedení validace modelu vytvoří pouze jedinou hodnotu (souřadnici), je pro získání více hodnot nutné provést validaci vícekrát a to s různými prahovými hodnotami. Výsledné hodnoty jsou poté zaneseny do grafu a je vykreslena křivka. Na této křivce je poté možné sledovat kompromisy mezi přesností a falešnými detekcemi. Čím více bodů se nachází „v levém horním rohu“ tj. poblíž souřadnic $[0; 1]$, tím lépe. Zároveň se tyto ROC křivky dají porovnávat mezi sebou, díky čemuž je možné lehce porovnávat úspěšnosti jednotlivých modelů.

ROC křivky v rámce této práce získávám tak, že skript `validate_roc.py` s argumentem `validate` pomocí upravené validační funkce v souboru `detector.c` provede validaci modelu. Výsledky validace jsou pomocí dalších upravených nebo přidaných funkcí uloženy do souboru `.json`. Tento soubor je poté skriptem s argumentem `roc` zpracován a jsou získána potřebná data pro ROC křivku, která je poté vykreslena pomocí pythonovské knihovny Bokeh[1]. Tyto data je možné také uložit do `.pkl` souboru pro pozdější vykreslení ROC křivky argumentem `parse_pkl`.

6.2 Experimenty s natrénovaným modelem

V této podkapitole jsou uvedeny různé experimenty s natrénovanými modely na testovací datové sadě a jejich ROC křivky, reprezentující výsledky. Tyto experimenty jsem provedl s cílem najít různé způsoby, jak zvýšit senzitivitu modelu a snížit počet falešných detekcí na obrázek, a tím co nejvíce **snížit celkovou chybu** modelu.

[1] https://bokeh.pydata.org/

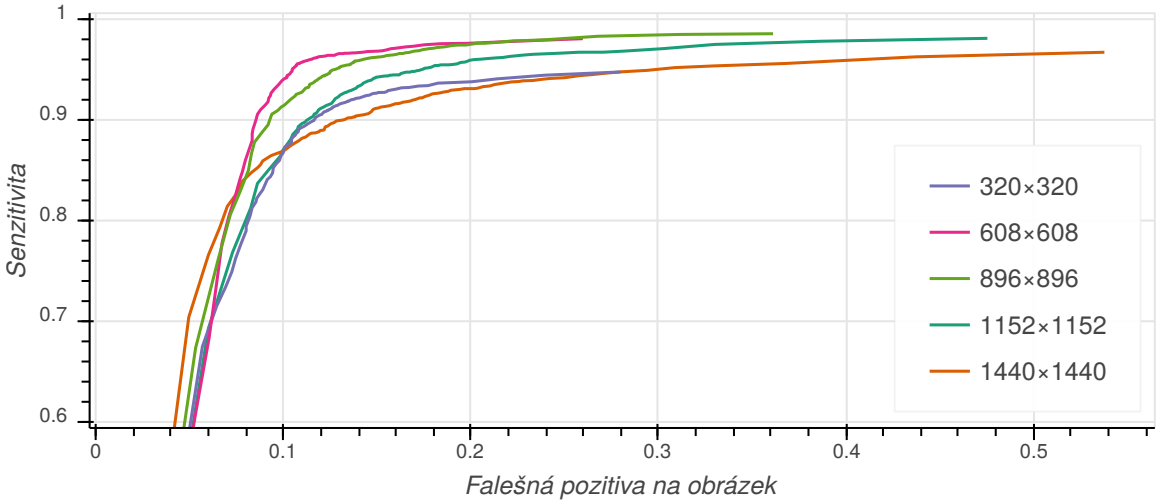


Obrázek 6.2: Vztah mezi senzitivitou a počtem falešných detekcí na obrázek znázorněný ROC křivkou.

6.2.1 Vliv velikosti rozlišení na chybovost a rychlost modelu

První experiment se zabývá nalezením vhodného vstupního rozlišení pro konvoluční neuronovou síť natrénovaného modelu. V obrázku 6.3 je možné vidět ROC křivky reprezentující výsledky validace s různými vstupními rozlišeními z rozsahu 320×320 pixelů až 1440×1440 pixelů. V tabulce 6.2 jsou tyto výsledky podrobněji popsány.

Na základě výsledků byly všechny další experimenty prováděny se vstupním rozlišením 608×608.



Obrázek 6.3: Porovnání chybovostí natrénovaného modelu validovaném na různě velkých vstupních rozlišeních. Nejlepších výsledků dosahuje model se vstupním rozlišením 608×608

Vstupní rozlišení (px)	Velikost minimální celkové chyby	FPS
320 × 320	17,504 %	122
608 × 608	**12,292 %**	**43**
896 × 896	14,023 %	24
1152 × 1152	16,212 %	15
1440 × 1440	18,603 %	10

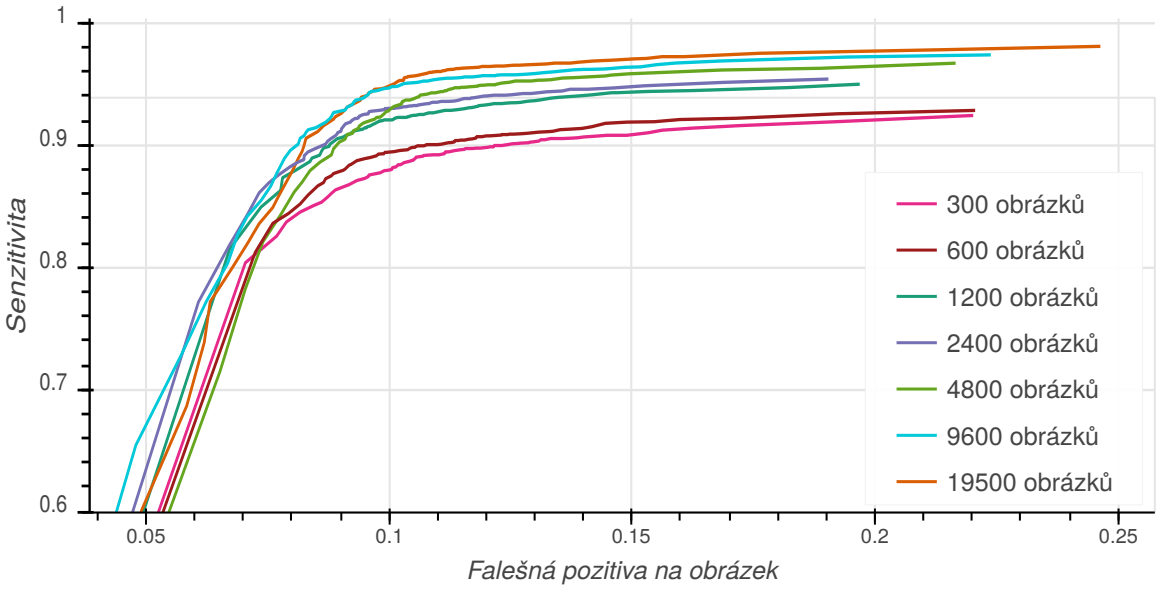
Tabulka 6.2: Tabulka výsledků experimentu 6.2.1. Z výsledků je možné vypozorovat, že chybovost modelu klesá se zvyšujícím se rozlišením až do rozlišení 608 × 608, poté už chybovost spíše opět roste.

6.2.2 Vliv velikosti datové sady na chybovost modelu

Při vytváření dalších anotovaných datových sad jsem dospěl k otázce, kolik dalších jich ještě bude zapotřebí vytvořit. Proto jsem provedl experiment, při kterém jsem dosavadní trénovací datovou sadu o velikosti přibližně 19 500 obrázků postupně náhodným výběrem dělil na menší datové sady o výsledných velikostech 300, 600, 1 200, 2 400, 4 800 a 9 600 obrázků, abych mohl sledovat, zda minimální celková chyba modelu nekonverguje k některé hodnotě, nebo naopak, zda se minimální celková chyba modelu dále nesnižuje nebo nezvyšuje.

S těmito vytvořenými datovými sadami jsem provedl nová trénování modelů a následně jejich validaci. Výsledek experimentu je zobrazen pomocí ROC křivek v obrázku 6.4 a v tabulce se získanými hodnotami 6.3.

Experiment potvrdil tvrzení, že s velikostí datové sady klesá i minimální celková chyba modelu. Z tohoto důvodu jsem se rozhodl provést augmentaci trénovací datové sady, aby byla rozšířena o další obrázky.



Obrázek 6.4: Porovnání chybovostí modelů natrénovaných na různě velkých trénovacích datových sadách.

Počet obrázků v datové sadě	Velikost minimální celkové chyby
300	18,076 %
600	17,305 %
1 200	14,924 %
2 400	14,06 %
4 800	13,572 %
9 600	12,48 %
19 500	**12,055 %**

Tabulka 6.3: Tabulka výsledků experimentu 6.2.2. S rostoucí velikostí datové sady klesá minimální celková chyba. Nejmenší celkové chyby dosahuje model natrénovaný na datové sadě o velikosti 19 500 obrázků.

6.2.3 Vliv síly augmentace dat na chybovost modelu

Poté, co jsem se rozhodl využít možnosti augmentace dat, pokusil jsem se pomocí dalšího experimentu zjistit, jaká síla augmentace (viz obrázek 4.5) vede ke snížení celkové chyby modelu. Z toho důvodu byly vytvořené, v kapitole 4.4 dříve zmiňované, augmentované datové sady, které byly použity k trénování, přičemž každá z těchto datových sad byla vytvořena jinak silnou augmentací.

Porovnání výsledků validace těchto datových sad a výsledku datové sady bez augmentace je zobrazeno pomocí ROC křivek v obrázku 6.5. Ve výsledcích experimentu lze vidět, že využití augmentace má pozitivní vliv na snižování velikosti celkové chyby. Minimální celkové chyby modelů natrénovaných na augmentovaných datech jsou si velmi podobné – 11,226 % u slabé augmentace, 11,122 % u středně silné augmentace a 10,849 % u silné augmentace.

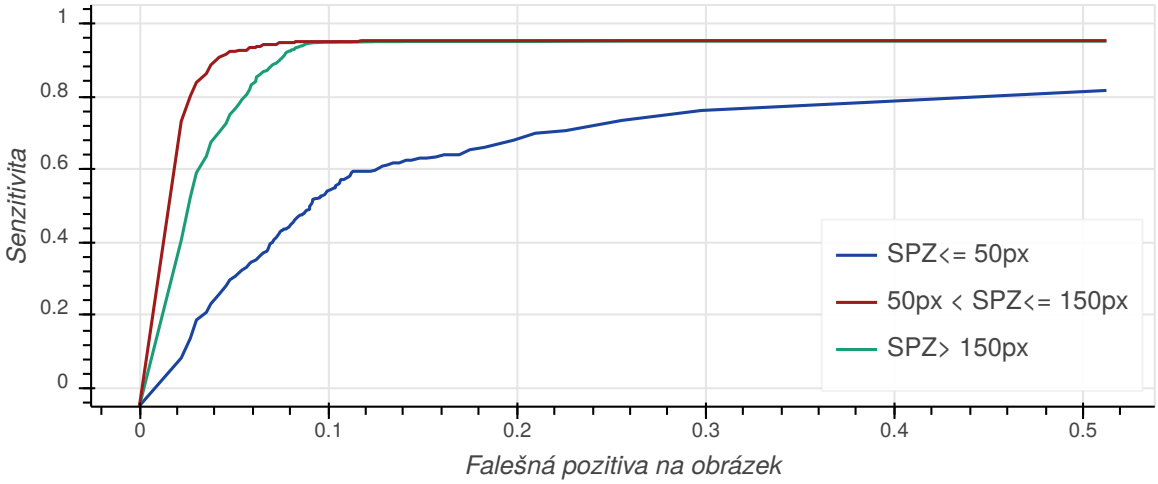
Pro další experimenty byl zvolen model, který byl natrénován na silně augmentované datové sadě, jelikož dosahuje nepatrně nižší minimální celkové chyby, než ostatní modely.

Obrázek 6.5: Porovnání chybovostí natrénovaného modelu validovaném na datových sadách, které jsou vytvořeny různou silou augmentace. Na datové sadě se silnou augmentací model dosahuje až 0,52, falešných detekcí na obrázek.

6.2.4 Chybovost modelu při detekci registračních značek rozdělených do kategorií podle velikostí

V tomto experimentu jsem se pokusil zjistit, jak model chybuje v jednotlivých velikostních kategoriích registračních značek. Na základě histogramu velikostí registračních značek v trénovací datové sadě na obrázku 4.6 jsem trénovací datovou sadu rozdělil do tří kategorií – malé značky o šířce do 50 pixelů, středně velké značky o velikosti 50–150 pixelů a velké značky o velikosti větší než 150 pixelů.

Výsledky experimentu jsou uvedeny na obrázku 6.6. V kategorii značek menších než 50 px model dosahuje minimální celkové chyby 60,401 %, v kategorii značek 50-150 px dosahuje minimální celkové chyby 9,314 % a v kategorii značek větších než 150 px dosahuje minimální celkové chyby 21.48 %.



Obrázek 6.6: Úspěšnost detekcí registračních značek v různých velikostních kategoriích. Z grafu lze vyčíst, že model má velmi často problém správně detekovat malé registrační značky o velikosti $\leq$ 50 px, naopak velmi dobře zvládá detekovat středně velké a velké značky.

Kapitola 7

Závěr

V rámci řešení jsem nastudoval problematiku detekce registračních značek vozidel různými metodami, které jsou založeny ať už na strojovém učení, tak i na hlubokém učení. V rámci studia hlubokého učení jsem se dále seznámil se vznikem a vývojem umělých neuronových sítí, principem jejich fungování i učení, a s jejich různými typy, převážně konvolučními neuronovými sítěmi. Dále jsem prozkoumal různé architektury konvolučních neuronových sítí, u kterých jsem se seznámil s jejich různými klasifikačními a detekčními modely. Dále jsem porovnal přednosti různých detekčních modelů, jejich nevýhody i výsledky, a na základě těchto porovnání si vybral výsledný model YOLOv3-tiny, a to pro jeho rychlost při zachování poměrně malé celkové chyby.

V rámci řešení jsem posbíral z volně dostupných zdrojů datovou sadu, kterou jsem dále rozšířil o datovou sadu vytvořenou výzkumnou skupinou GRAPH@FIT z mé fakulty a o mnou vytvořené, ručně anotované, datové sady. Tuto mou datovou sadu dávám veřejně k dispozici, aby mohla být dále použita pro další projekty.

Pro trénování konvolučních sítí jsem využil existujícího frameworku pro konvoluční neuronové sítě Darknet a pro své potřeby jsem upravil a rozšířil jeho testovací, trénovací i validační funkce. Pro různé potřeby během trénování i validace jsem vytvořil i několik skriptů, které sloužily ke zpracování různých datových sad a k vykreslování grafů. S natrénovaným modelem jsem provedl několik experimentů, díky kterým se mi povedlo zjistit, kdy model dosahuje nejmenší celkové chyby a dosahuje nejlepších výsledků.

Výsledný model dosahuje minimální celkové chyby 10,849 %. Této hodnoty dosahuje po natrénování na normální a silně augmentované datové sadě při vstupním rozlišení 608×608 px, a to při prahové hodnotě 47 %. Model nejlépe detekuje značky o šířce v rozmezí 50–150 px.

Celková chyba modelu by tudíž ještě šla pravděpodobně snížit přidáním dalších malých a velkých registračních značek do datové sady, tuto datovou sadu celkově rozšířit o křivé, rozmazané a různě deformované registrační značky a model na těchto datech dotrénovat. Jelikož testovací datová sada se skládá z obrázků značek, které jsou všechny pořízeny za dne, jsou ostré a ve vysokém rozlišení, bylo by vhodné vytvořit další testovací datové sady, které by obsahovaly obrázky různě poškozených značek, značek pořízených za šera nebo v noci, pořízených ve špatném počasí apod., a model otestovat i na nich. Stejně tak by bylo vhodné otestovat výsledný model na datových sadách značek jiných státu, které používají jiné tvary, velikosti či barvy.

Literatura

[1] Bishop, C. M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, ISBN 0387310738.

[2] Carcagnì, P.; Del Coco, M.; Leo, M.; aj.: Facial expression recognition and histograms of oriented gradients: a comprehensive study. *SpringerPlus*, ročník 4, listopad 2015: str. 645.

[3] Cortes, C.; Vapnik, V.: Support-Vector Networks. *Machine Learning*, ročník 20, č. 3, září 1995: s. 273–297.

[4] Dalal, N.; Triggs, B.: Histograms of Oriented Gradients for Human Detection. In *International Conference on Computer Vision & Pattern Recognition (CVPR '05)*, ročník 1, editace C. Schmid; S. Soatto; C. Tomasi, San Diego, United States: IEEE Computer Society, červen 2005, s. 886–893.

[5] Das, S.: CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more.... *Medium.com*, listopad 2017.

[6] Dvořák, M.: *Detekce a rozpoznání dopravního značení*. Diplomová práce, Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav automatizace a měřicí techniky, Brno, 2015.

[7] E. Rumelhart, D.; E. Hinton, G.; J. Williams, R.: Learning Representations by Back Propagating Errors. *Nature*, ročník 323, říjen 1986: s. 533–536.

[8] Fawcett, T.: An introduction to ROC analysis. *Pattern Recognition Letters*, ročník 27, č. 8, 2006: s. 861–874, ISSN 0167-8655, rOC Analysis in Pattern Recognition.

[9] Freund, Y.; Schapire, R. E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, ročník 55, č. 1, 1997: s. 119 – 139, ISSN 0022-0000.

[10] Girshick, R.: Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, prosinec 2015, s. 1440–1448.

[11] Girshick, R.; Donahue, J.; Darrell, T.; aj.: Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, červen 2014: s. 580–587.

[12] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, ISBN 9780262035613.

[13] Guo, Y.; Liu, Y.; Oerlemans, A.; aj.: Deep learning for visual understanding: A review. *Neurocomputing*, ročník 187, 2016: s. 27 – 48, ISSN 0925-2312, recent Developments on Deep Big Vision.

[14] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: s. 770–778.

[15] Hui, J.: Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3. *Medium.com*, březen 2018.

[16] Ioffe, S.; Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, editace F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, s. 1097–1105.

[18] Laroca, R.; Severo, E.; Zanlorensi, L. A.; aj.: A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector. In *2018 International Joint Conference on Neural Networks (IJCNN)*, červenec 2018, ISSN 2161-4407, s. 1–10.

[19] LeCun, Y.; Bengio, Y.; Hinton, G.: Deep Learning. *Nature*, ročník 521, květen 2015: s. 436–444.

[20] LeCun, Y.; Bottou, L.; Bengio, Y.; aj.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, ročník 86, č. 11, listopad 1998: s. 2278–2324.

[21] Liu, W.; Anguelov, D.; Erhan, D.; aj.: SSD: Single Shot MultiBox Detector. In *European conference on computer vision*, 2016, s. 21–37.

[22] Lowe, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, ročník 60, č. 2, listopad 2004: s. 91–110, ISSN 1573-1405.

[23] Maj, M.: Object Detection and Image Classification with YOLO. srpen 2018, [Online; navštíveno 27.04.2019].
URL https://appsilon.com/object-detection-yolo-algorithm/

[24] Marsland, S.: *Machine Learning: An Algorithmic Perspective*. New Jersey, USA: CRC Press, druhé vydání, 2014, ISBN 978-1466583283.

[25] Parsad, N. M.: Deep Learning in Medical Imaging V. *Medium.com*, červen 2018.

[26] Pitts, W.; McCulloch, W. S.: A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, ročník 5, č. 4, prosinec 1943: s. 115–133, ISSN 1522-9602.

[27] Prada, J. C. B.: Hello, Gradient Descent. *Medium.com*, únor 2017.

[28] Redmon, J.: Darknet: Open Source Neural Networks in C. 2013–2016, [Online; navštíveno 29.04.2019].
URL http://pjreddie.com/darknet/

[29] Redmon, J.: YOLO: Real-Time Object Detection. březen 2018, [Online; navštíveno 27.04.2019].
URL https://pjreddie.com/darknet/yolo/

[30] Redmon, J.; Divvala, S.; Girshick, R.; aj.: You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, červen 2016, s. 779–788.

[31] Redmon, J.; Farhadi, A.: YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, s. 7263–7271.

[32] Redmon, J.; Farhadi, A.: YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[33] Ren, S.; He, K.; Girshick, R.; aj.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 39, č. 6, červen 2017: s. 1137–1149.

[34] Ruder, S.: An overview of gradient descent optimization algorithms. *arXiv*, červen 2017.

[35] Russakovsky, O.; Deng, J.; Su, H.; aj.: ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, ročník 115, č. 3, 2015: s. 211–252.

[36] Sammut, C.; Webb, G. I.: *Encyclopedia of Machine Learning*. Springer Publishing Company, Incorporated, první vydání, 2011, ISBN 0387307680, 9780387307688.

[37] Sharma, A.: Understanding Activation Functions in Neural Networks. *Medium.com*, březen 2017.

[38] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*, září 2014.

[39] Srebrić, V.: *Postupci za poboljšanje kontrasta sivih slika*. Diplomski zadatak, Sveučilište u Zagrebu. Fakultet elektrotehnike i računarstva. Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave., Zagreb, 2003.

[40] Suykens, J.: Deep Restricted Kernel Machines Using Conjugate Feature Duality. *Neural Computation*, ročník 29, květen 2017: s. 1–41.

[41] Szegedy, C.; Liu, W.; Jia, Y.; aj.: Going deeper with convolutions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, červen 2015: s. 1–9.

[42] Teuer, L. a kol.: Dataset ZoomLPDataset_sorted, výzkumná skupina GRAPH@FIT.

[43] Tsang, S.-H.: Review: YOLOv1 – You Only Look Once (Object Detection). říjen 2018, [Online; navštíveno 29.04.2019].
URL https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89

[44] Viola, P.; Jones, M.; aj.: Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, ročník 1, 2001: s. 511–518.

[45] Virtuální organizace Metacentrum: Jak počítat/Rychlý start. 2019, [Online; navštíveno 30.04.2019].
URL https://wiki.metacentrum.cz/wiki/Jak_počítat/Rychlý_start

[46] Xu, J.: Deep Learning for Object Detection: A Comprehensive Review. září 2017, [Online; navštíveno 28.04.2019].
URL https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9

[47] Zeiler, M. D.; Fergus, R.: Visualizing and Understanding Convolutional Networks. In *European conference on computer vision*, Springer, 2014, s. 818–833.

[48] Zhao, Z.; Zheng, P.; Xu, S.; aj.: Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 2019: s. 1–21, ISSN 2162-237X.

[49] Špaňhel, J.; Sochor, J.; Juránek, R.; aj.: Holistic recognition of low quality license plates by CNN using track annotated data. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, Srpen 2017, s. 1–6.

Příloha A

Obsah přiloženého paměťového media

darknet	Přejaté a upravené zdrojové soubory aplikace Darknet
datasets	Veškeré datové sady, které byly v práci použity
examples	Ukázky obrázků a videí se značkami, které byly úspěšně i neúspěšně detekovány vytvořeným modelem
scripts	Vytvořené skripty pro zpracování různých dat a vykreslení grafů
thesis	Zdrojové soubory pro elektronickou verzi této práce
weights	Natrénované váhové soubory
README.md	Podrobný popis souborů na přiloženém paměťovém médiu

Příloha B

Plakát

Obrázek B.1: Plakát reprezentující hlavní body této práce. Vytištěný plakát ve formátu A2 je k práci samostatně přiložen.