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 DOPRAVNÍCH ZNAČEK A SEMAFORŮ

DETECTION OF TRAFFIC SIGNS AND LIGHTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ CHOCHOLATÝ**
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: **Chocholatý Tomáš**
Program: Informační technologie
Název: **Detekce dopravních značek a semaforů**
Detection of Traffic Signs and Lights
Kategorie: Zpracování obrazu

Zadání:

1. Vyhledejte a prostudujte existující přístupy k detekci dopravních značek a semaforů.
2. Prostudujte moderní přístupy k detekci objektů v obraze - zaměřte se na moderní architektury konvolučních neuronových sítí.
3. Pořiďte/získejte datovou sadu vhodnou pro učení a vyhodnocování detektorů dopravních značek a semaforů.
4. Experimentujte s dostupnými architekturami konvolučních neuronových sítí; vylepšujte je a přizpůsobujte je pro řešenou úlohu.
5. Vyhodnocujte řešené algoritmy na datové sadě (datových sadách) a iterativně je vylepšujte. Komentujte vlastnosti vytvářeného řešení.
6. 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

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 až 5.

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: 1. listopadu 2018

Abstrakt

Práce se zabývá detekcí dopravních značek a semaforů v obraze s využitím konvolučních neuronových sítí. Cílem je vytvoření vhodného detektoru pro detekci a rozpoznání dopravního značení v reálném provozu. Za účelem trénování konvolučních neuronových sítí byly vytvořeny vhodné datové sady, které se skládají ze syntetické i reálné datové sady. Pro syntetickou datovou sadu byl vytvořen generátor, který simuluje různé deformace značek. Vyhodnocení kvality detekce je prováděno pomocí vlastního programu pro kvantitativní vyhodnocování. Podařilo se dosáhnout úspěšnosti 84% detekovaných značek nad vlastní testovací datovou sadou. Výsledky umožňují zjistit důležitost zastoupení reálné či syntetické datové sady v trénovací sadě a vliv jednotlivých deformací syntetické datové sady na konečnou kvalitu detekce.

Abstract

The thesis focuses on traffic sign detection and traffic lights detection in view with utilization convolution neural network. The goal is create suitable detector for detection and classification traffic sign in real traffic. For training of convolution neural network were created appropriate datasets, that contains synthetic and real dataset. For synthetic dataset was create generator, that can simulated different deformation of traffic signs. Evaluation is done by own program for quantitative evaluation. The detection rate successfully detected signs is 89% over own test dataset. The results allow to find out importance of representation real or synthetic dataset in training dataset and influence individual deformations synthetic dataset for final detection quality.

Klíčová slova

Detekce a klasifikace dopravní značek, konvoluční neuronové sítě, detekce objektů v obraze, YOLO, syntetická datová sada, generátor syntetické datové sady, kvantitativní vyhodnocování

Keywords

Traffic sign detection and classification, Convolution neural network, Object detecion, YOLO, Synthetic dataset, Generator for synthetic dataset, Quantitative evaluation

Citace

CHOCHOLATÝ, Tomáš. *Detekce dopravních značek a semaforů*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Adam Herout, Ph.D.

Detekce dopravních značek a semaforů

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 Chocholatý
13. května 2019

Poděkování

Rád bych poděkoval svému vedoucímu práce prof. Ing. Adamu Heroutovi, Ph.D za veškerou pomoc během práce, cenné rady a vstřícnost při konzultacích. Dále bych chtěl poděkovat za přístup k výpočetním zdrojům a úložným prostorům. 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 detekcí a rozpoznáváním dopravního značení v obraze. Detekce a rozpoznávání dopravního značení je jedno z hlavních témat spojených s rozvojem asistenčních systémů řidiče a autonomního řízení automobilů.

V současné době již existují systémy rozpoznávání dopravního značení (TSR – Traffic sign recognition), které spadají do pokročilých systémů asistence řidiče (ADAS). Tyto systémy mají však řadu omezení. Jsou schopny detekovat pouze značky omezující rychlost nebo značku zakazující předjíždění. Většina těchto detekcí používaných dnes v reálném provozu pracuje na způsobu detekce, který k detekci a klasifikaci nevyužívá konvoluční neuronové sítě. Tyto systémy pro zpracování obrazu jsou dnes v automobilovém průmyslu používány především díky své nízké náročnosti na výpočetní výkon a to i na úkor nižší spolehlivosti. S rychlostí neustálého vývoje ve výpočetní technice lze předpokládat, že bude možné brzy nasadit v tomto odvětví i modernější a spolehlivější systémy detekce pomocí konvolučních neuronových sítí, se kterými se dnes můžeme setkat při prototypech autonomních automobilů.

Cílem této práce bylo prostudovat moderní přístupy k detekci objektů v obraze se zaměřením na moderní architekturu konvolučních neuronových sítí, tuto architekturu vylepšovat a přizpůsobovat pro řešenou úlohu. Získat datovou sadu vhodnou pro učení a vyhodnocování detektorů dopravních značek a semaforů. Z důvodu neexistence vhodných volně dostupných datových sad, musely být tyto sady vytvořeny. Výsledná datová sada je složena z reálné a syntetické datové sady. Pro přípravu syntetické datové sady byl vytvořen generátor, který dokáže zohlednit nejen různou velikost dopravních značek a jejich umístění v obraze, ale i deformaci dopravního značení a imitaci zhoršených povětrnostních podmínek, se kterými se můžeme běžně setkat v reálném provozu. Reálná datová sada byla vytvořena anotací cca 8 000 sekvenčních snímků z palubní kamery automobilu pořízených v různých lokalitách na území České republiky. Práce obsahuje také vliv jednotlivých deformací dopravních značek, obsažených v syntetické datové sadě, na celkovou úspěšnost následné detekce.

V kapitole 2 jsou popsány vybrané tradiční přístupy k detekci objektů bez použití konvolučních neuronových sítí. Kapitola 3 obsahuje stručný úvod do problematiky neuronových sítí. V kapitole 3.2 – 3.5 jsou popsány a porovnány již existující nástroje pracující s konvolučními neuronovými sítěmi, které lze využít pro detekci. Kapitola 3.4 se zabývá popisem použitého nástroje pro tuto práci. S ohledem na nejlepší poměr rychlosti a kvality detekce byl nakonec vybrán nástroj YOLO (You Only Look Once). Tento nástroj má speciální požadavky na datovou sadu používanou při trénování modelu. Specifika a průběh vytváření datové sady zmiňuje kapitola 4.3. Kapitola 4.2 se věnuje průběhu vyhodnocování jednotli-

vých modelů a nástrojů, které byly k tomuto porovnání využity. V předposlední kapitole 5 lze nalézt výsledky detektoru a porovnání jednotlivých modelů, které byly natrénovány nad různými datovými sadami. V závěru této práce jsou shrnuty výsledky a je navržen případný budoucí vývoj.

Kapitola 2

Tradiční přístupy k detekci dopravních značek a semaforů

V této kapitole budou popsány způsoby detekce značek jinými metodami a přístupy, než pomocí konvolučních neuronových sítí. Každá metoda je popsána záměrně stručně. Jádro mé bakalářské práce tvořily konvoluční neuronové sítě, proto se jiným přístupům k detekci věnuji pouze okrajově.

I když se systémy pokročilé asistence řidiče ADAS[1] zdají být na první pohled výstřelkem až dnešní moderní doby [11], ve skutečnosti za sebou mají podstatnou část historie a vývoje. Již okolo roku 1986 začalo několik výrobců automobilů a výzkumných skupin vyvíjet iniciativu v této oblasti. Dnes už existuje více přístupů k problematice detekce objektů. Některé vybrané přístupy využívané před příchodem konvolučních neuronových sítí budou popsány v rámci této kapitoly.

Dopravní značky jsou navrhovány tak, aby měly vysoký kontrast vůči okolí a byly pro řidiče automobilu dobře zpozorovatelné. Jejich základní tvary v kombinaci s vysoce kontrastními barvami zaručují jejich dobrou viditelnost. U dopravních značek je také snaha odlišit od sebe jednotlivé tvary tak, aby nedošlo k jejich nechtěné záměně, což by mohlo vést ke vzniku dopravní nehody. Existují metody detekce, které těží z těchto specifických vlastností dopravního značení.

2.1 Detekce na základě segmentace barvy

Detekční techniky na základě barvy [21] používají barvu jako hlavní znak pro identifikaci oblasti obsahující dopravní značku. Segmentace barev eliminuje zbytečné objekty a zmenšuje tak oblast hledání. Originální obraz je převeden do tónů šedé a např. pro hledání červené barvy je pouze červený kanál extrahován z originálního obrazu. Obrázek ve stupních šedi je odečten z červené komponenty originálního obrazu a poté se odečtený obraz převede na binární hodnoty použitím prahového filtru. Pokud je hodnota RGB (Red – Green – Blue) komponenty větší jak hodnota prahu je převedena na hodnotu 255. V opačném případě je převedena na 0. Pomocí morfologických operací jsou odečteny šumy na pozadí. Na výstupu vznikne binární mapa.

V těchto metodách se používají různé barevné prostory. Nejčastěji se můžeme setkat s barevnými prostory RGB nebo HSI (Hue – Saturation – Intensity). Segmentace s použitím barevného prostoru RBG je méně časově náročná, není však tak robustní při změně osvě-

[1] http://www.adas.upol.cz/

tlení. U barevného prostoru HSI se oddělují achromatické a chromatické složky, detekce má lepší výsledky za vlivu náročnějších světelných podmínek. Obrázek RGB je v tomto případě převeden na HSI a segmentace se provádí na základně hodnot Hue a Saturation.

2.2 Detekce pomocí hran objektů

Dopravní značky, se kterými se dnes můžeme setkat, mají tvar trojúhelníku, obdélníku, čtverce, osmiúhelníku či kruhu. Pokud se při detekci zaměříme pouze na hrany, nevznikne problém s odlišnými barevnými provedeními značek v různých státech. Na těchto vlastnostech jsou založeny detektory využívající hran objektu [21].

Hrana je místo v obraze s výraznou změnou úrovně jasu. Před započetím tohoto způsobu detekce je většinou celý vstupní obraz převeden do stupňů šedi. Po tomto kroku nastává problém v oblasti šumu. Je zapotřebí provést prahování hran. V obraze zůstanou pouze výrazné hrany. Následně se aplikuje Gaussův filtr, který eliminuje zbývající šum.

Dalším krokem po segmentaci obrazu, ať již založené na segmentaci barev či tvarů, může být hledání určitých souvislostí mezi nalezenými body. K tomu lze využít například Houghovy transformace. Houghova transformace[2] je metoda pro nalezení parametrického popisu objektu v obraze. Tato metoda je používána pro detekci jednoduchých objektů v obraze jako jsou přímky, kružnice, elipsy, atd. Analýza všech kontur v obraze způsobí větší náklady na výpočetní výkon.

Nevýhodou metod založených na detekci hran obrazu je detekce v prostředí, kde se často vyskytují geometrické tvary podobné značkám. Většinou se jedná o sídliště, centra měst a další hustě obydlené oblasti. Výhodou, oproti detekci založené na segmentaci barev, je větší odolnost proti rozdílným světelným podmínkám nebo změnám v odstínech značek během jejich doby používání.

2.3 Detekce kombinující obě výše zmíněné metody

Při použití chromatických a tvarových informací odděleně se zvyšuje počet rušení. Rušení v tomto případě znamená situaci, kdy obraz obsahuje objekty stejné barvy nebo tvaru, ale nejedná se o dopravní značky. Chromatické informace s informací o tvaru jsou kombinovány právě proto, aby bylo minimalizováno toto rušení. Tento proces zahrnuje dvě fáze. První fází je segmentace barvy v libovolném barevném prostoru. Druhou fází je detekce tvaru značek tvarovou analýzou.

2.4 Viola-Jones detektor

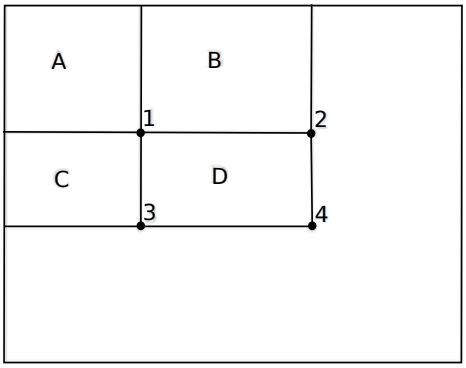
Detektor Viola-Jones [26, 12] byl poprvé představen P.Violou a M. Jonesem v roce 2001. Detektor pracuje s šedotónovými obrazy. Je složen ze tří částí. Konkrétně se jedná o výpočet intergrálního obrazu, filtrů založených na Haarových vlnkách a klasifikačního algoritmu AdaBoost. Tento detektor byl v praxi využíván především k detekci obličejů. Jeho výhodou je rychlost, spolehlivost a značná nezávislost na změně osvětlení.

Integrální obraz byl zaveden z důvodu snížení časové náročnosti. Je to způsob digitální reprezentace obrazu, kde každý bod (x, y) představuje sumu hodnot pixelů směrem doleva a nahoru od tohoto bodu. Každý bod (x, y) v obraze můžeme spočítat dle rovnice:

[2] https://patents.google.com/patent/US3069654A/en

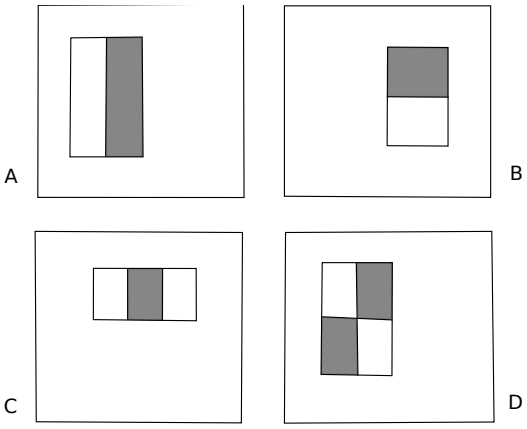
$$\begin{aligned} \mathrm{s(x,y)} &= \mathrm{s(x,y-1)} + \mathrm{i(x,y)} \\ \mathrm{ii(x,y)} &= \mathrm{ii(x-1,y)} + \mathrm{s(x,y)} \end{aligned} \tag{2.1}$$

kde $s(x, y)$ je kumulovaný součet hodnot v řádku a $i(x, y)$ jsou hodnoty původního vstupního obrazu. ii(x, y) je výsledný integralní obraz do pozice x, y.



Obrázek 2.1: Grafická reprezentace integrálního obrazu – Suma pixelů uvnitř obdélníku D může být vypočítána pomocí hodnot v jeho rozích. Hodnota integrálního obrazu 1 je suma všech pixelů v obdélníku A. Hodnota integrálního obrazu 2 je $A + B$, hodnota integrálního obrazu 3 je $A + C$ a hodnota integrálního obrazu 4 je $A + B + C + D$. Sumu pixelů uvnitř obdélníku D lze spočítat jako $4 + 1 - (2 + 3)$ [12].

Filtry založené na Haarových vlnkách jsou plošné detektory charakteristických rysů daného objektu. Základní sada Harrových filtrů je znázorněna na obr 2.2. Existuje celá řada i složitějších filtrů. Výsledkem Harrova filtru je číslo, které vznikne rozdílem jasových hodnot pod světlou a tmavou částí tohoto filtru. V subobraze je vypočteno mnoho odezev, na jejichž základě je následně rozhodnuto o přítomnosti objektů. Tento výpočet příznaků je velmi efektivní v kombinaci s integrálním obrazem.



Obrázek 2.2: Základní sada Haarových filtrů [12]

Klasifikace AdaBoost (název je zkratkou pro Adaptive Boosting) je klasifikační algoritmus, který vychází z metody strojového učení zvaného boosting. Cílem metody boosting

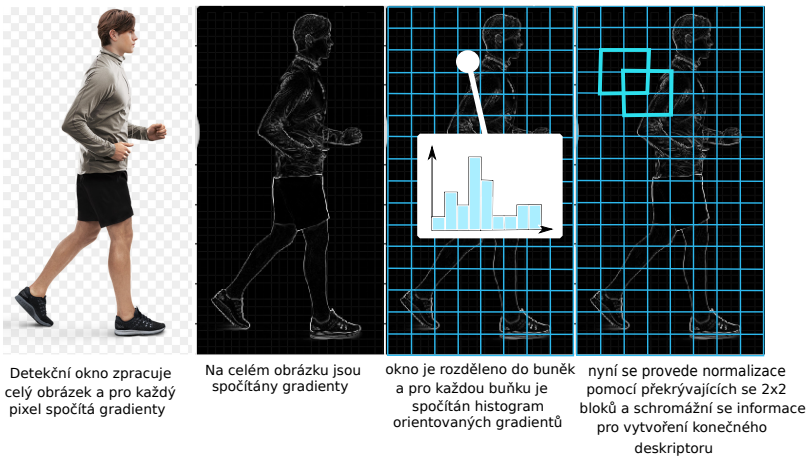
je zlepšení klasifikační přesnosti libovolného algoritmu strojového učení. Základem je existence několika klasifikátorů označovaných jako weak learners (slabý klasifikátor). První klasifikátor má úspěšnost jen o něco málo větší než přesnost odhadu. Postupně jsou však přidávány další klasifikátory, čímž je vygenerován soubor klasifikátorů označovaný jako strong learner (silný klasifikátor).

Detektor Viola Jones používá kaskádu silných klasifikátorů. Každý klasifikátor má obvykle rozdílné klasifikační schopnosti. Pokud první klasifikátor daný subobraz zamítne z důvodu, že se v obraze podle něj hledaný objekt nenachází, další klasifikátory tento subobraz již neprohledávají. Pravděpodobnost, že se v obraze daný objekt nachází, je po zamítnutí prvním klasifikátorem nízká. Hlavním přínosem kaskády klasifikátorů je úspora času. Na konci zůstanou pouze subobrazy označované jako hledaná oblast zájmu, kde se s největší pravděpodobností nachází hledaný objekt.

2.5 Histograms of Oriented Gradients – HOG

Metoda je založena na histogramech orientovaných gradientů [4]. Je pro ní charakteristická nízká citlivost na změnu rotace či osvětlení detekovaného objektu.

Hlavní myšlenkou je, že objekt lze dobře charakterizovat pomocí intenzity gradientů nebo směrem hran a to i bez předešlých znalostí odpovídajících gradientů či polohy hran. V praxi se obrazové okno rozdělí na malé prostorové oblasti a pro každou oblast se vypočítá 1-D histogram, který vznikne výpočtem ze všech pixelů dané oblasti. Pro výhodnější neměnnost osvětlení, stínování atd. je lépe obraz před výpočty normalizovat. Tento postup se provádí pomocí shromažďování informací do histogramu nejen z jedné oblasti, ale z většího prostoru okolních oblastí. Výsledkem je pak tzv. Histogram orientovaných gradientů.



Obrázek 2.3: Princip činnosti detektoru založeném na histogramu orientovaných gradientů

Nejdříve se provede výpočet gradientů. Pro určení hledaného vektoru je nejprve potřeba najít významné hrany v obraze. Detekce hran spočívá ve vyhledání lokálních změn v intenzitě sousedních pixelů v obraze. Jako nejlepší řešení se jeví použití dvou jednoduchých derivačních masek $[-1, 0, 1]$ a $[-1, 0, 1]^T$ na detekci horizontálních a vertikálních hran. Obraz je rozdělen na hustou mřížku buněk určité velikosti a pro každý pixel jsou spočteny gradienty. Pro každou buňku se následně určí histogram jejich gradientů. Každý pixel nese informaci o směru a velikosti pro výsledný histogram orientovaných gradientů. Kanály histogramu uvnitř každé buňky jsou určeny směrem gradientu. Počet kanálů je libovolný.

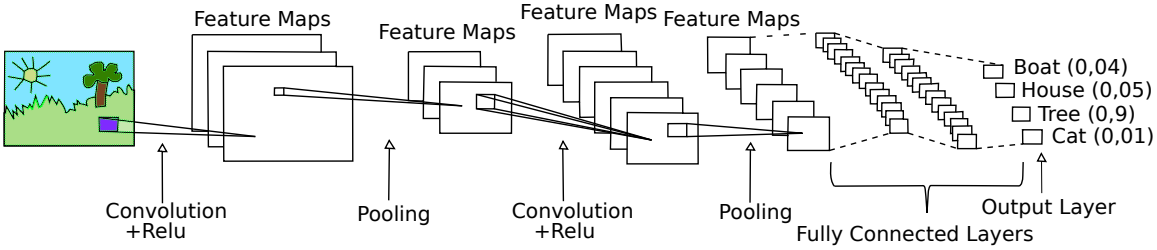
Buňka je tedy pro příklad rozdělena na 9 kanálů v rozsahu $0° - 180°$ nebo $0° - 360°$. Velikost kanálů histogramu je určena velikostí příslušných gradientů. Dále jsou prozkoumány všechny gradienty v buňce a podle orientace gradientu se zařadí do jednotlivých kanálů. Takto nám vznikne pro každou buňku histogram orientovaných gradientů. Pro lepší výsledky je následně každá buňka normalizovaná. Normalizace se provádí pomocí vzájemně se překrývajících bloků. HOG extrahuje příznakový vektor, ale neřeší klasifikaci. Klasifikace se řeší pomocí pokročilého klasifikátoru, kterým může být SVM (Support Vector Machine).

Kapitola 3

Konvoluční neuronové sítě a jejich použití pro detekci objektů

Konvoluční neuronové sítě (CNN) [1, 15] spadají do kategorie neuronových sítí. Mají efektivní využití především v obrazové detekci a klasifikaci.

Konvoluční neuronová síť, jak je znázorněno na obrázku 3.1, se skládá z jedné nebo více konvolučních a sdružovacích vrstev. Poté následuje jedna nebo více plně propojených vrstev jako ve standardní neuronové síti.



Obrázek 3.1: Princip konvoluční neuronové sítě – konvoluční vrstva provádí konvoluci, na výstupu této vrstvy probíhá sdružování. Následně se vytvoří několik map funkcí (závisí od počtu filtrů). Tento postup se několikrát opakuje. Dále je přidána plně propojená vrstva, která má za úkol klasifikaci objektu. Na výstupu se promítne míra důvěry pro každou třídu, která vyjadřuje pravděpodobnost, že se jedná o daný objekt [15].

3.1 Konvoluční neuronové sítě – vrstvy

Architektura CNN je navržena tak, aby využila 2D struktury vstupního obrazu nebo jiného 2D vstupu. Výhodou CNN je jednodušší trénování a menší množství parametrů, v porovnání s plně propojenými sítěmi se stejným počtem skrytých jednotek.

Konvoluční neuronové sítě se skládají ze čtyř hlavních operací:

1. Konvoluce
2. Zavedení nelinearity
3. Sdružování
4. Klasifikace

Tyto operace jsou základními stavebními bloky pro každou konvoluční neuronovou síť.

3.1.1 Konvoluční vrstva

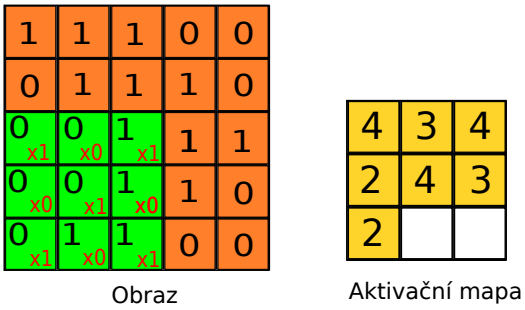
Hlavním úkolem konvoluční vrstvy je získání vlastností ze vstupního obrazu. Konvoluci můžeme vyjádřit dle vzorce:

$$\mathrm{f(x,y)\,h(x,y)} = \sum_{\mathrm{i=-k}}^{\mathrm{k}} \sum_{\mathrm{j=-k}}^{\mathrm{k}} \mathrm{f(x-i,y-j)\,h(i,j)} \tag{3.1}$$

V případě diskrétní konvoluce [5] lze jádro chápat jako tabulku (konvoluční filtr), která se položí na příslušné místo obrazu. Každý pixel překrytý tabulkou se vynásobí koeficientem v příslušné buňce a provede se součet všech těchto hodnot. Tím vznikne jeden nový pixel.

Počítače čtou obrazy jako pixely, které jsou vyjádřeny formou matice $(N \times N \times 3)$ (výška – šířka – hloubka). Snímky používají tři kanály (RGB), proto hloubka nabývá pevné hodnoty 3. Konvoluční vrstva [24] využívá sadu učitelných filtrů. Filtr se používá k detekci specifických funkcí nebo vzorů přítomných v původním obraze. Obvykle se vyjadřuje jako matice $(M \times M \times 3)$ s menší dimenzí, ale se stejnou hloubkou jako vstupní soubor. Na vstupním souboru se konvertují různé filtry, které detekují rozdílné vlastnosti a na výstupu je sada aktivačních map, která je předána další vrstvě CNN. V praxi se CNN během tréninku učí hodnoty těchto filtrů samostatně. Je ovšem nutné před tréninkem zadat parametry jako počet filtrů, velikost filtru, architekturu sítě atd. [25]. Velikost Aktivační mapy (Feature map) je řízena třemi parametry:

- Hloubka – odpovídá počtu filtrů, které používáme pro operaci konvoluce
- Krok – počet pixelů, o který posuneme filtrační matici
- Zero-padding – vložení původní matice s nulovými hodnotami kolem kraje, aby se dal použít filtr na hraniční prvky matice vstupního obrazu. Vstup i výstup pak mají stejnou výšku a šířku.



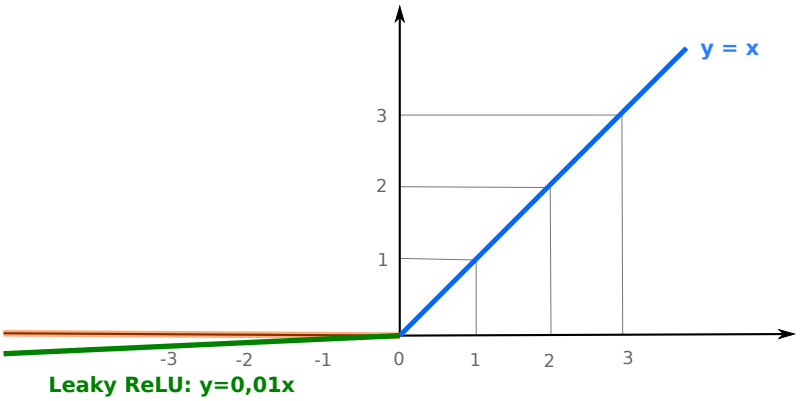
Obrázek 3.2: Princip konvoluce s použitím fitru o velikosti 3x3. Matice o rozměru 3×3 určuje takzvaný filtr. Filtr se posunuje přes původní obrázek např. o jeden pixel (krok). Pro každou pozici se vypočítá elementární násobení mezi dvěmi maticemi. Matice vytvořená posunutím filtru přes obraz se nazývá Aktivační mapa. [15]

3.1.2 Aktivační vrstva

Účelem aktivační funkce [24] je zavedení nelinearity. Aktivační funkce je nelineární transformace, která se provádí přes vstupní signál. Tento transformovaný výstup je poté poslán do další vrstvy jako vstup.

Máme různé typy aktivačních funkcí. Nejrozšířenější aktivační funkcí v neuronových sítích je funkce ReLU (Rectified Linear Units). Jedna z největších výhod ReLU oproti jiným aktivačním funkcím je, že neaktivuje všechny neurony najednou. Z obrázku 3.3 je zřejmé, že převádí všechny negativní vstupy na nulu a neuron se neaktivuje. V praxi konverguje funkce ReLU šestkrát rychleji než jiné aktivační funkce `sigmoid` nebo `tanh`.

Nevýhodou ReLU je, že je nasycený v záporné oblasti, což znamená, že gradient v této oblasti je nulový. S gradientem rovným nule během zpětného šíření chyby nebudou všechny váhy aktualizovány. Hrozí, že ReLU „umře“. To znamená, že bude vydávat stejnou hodnotu pro jakýkoliv vstup. Jakmile ReLU skončí v tomto stavu, je nepravděpodobné, že se zotaví. Jako řešení lze použít aktivační funkce Leaky ReLU s malým pozitivním gradientem pro záporné vstupy. Obě zmiňované funkce jsou vykresleny na obrázku 3.3.

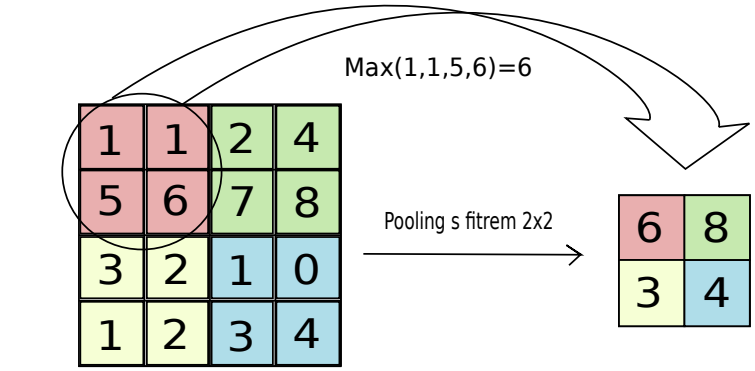


Obrázek 3.3: Porovnání funkce ReLU a Leaky ReLU. V prvním kvadrantu jsou obě funkce totožné. Ve třetím kvadrantu se funkce liší. Funkce ReLU je znázorněna oranžovou barvou, zatímco funkce Leaky ReLU je znázorněna zelenou barvou. Z grafu je patrné, že gradient funkce ReLU je ve třetím kvadrantu nulový na rozdíl od gradientu funkce Leaky ReLU.

3.1.3 Pooling vrstva

Jedná se o prostorové sdružování [2] také nazývané subsampling nebo downsampling. Snižuje rozměry každé mapové funkce, ale zachovává nejdůležitější informace. Vrstva sdružování může být viděna v architektuře CNN mezi vrstvami konvoluce. Tato vrstva snižuje množství parametrů a výpočtů v síti postupným snižováním prostorové velikosti sítě. Prostorové sdružování může být různého druhu: maximum, průměr, součet atd. V případě max pooling se definuje oblast (např. 2×2), jak lze vidět na obrázku 3.4, a vezme se největší prvek z dané oblasti mapové funkce. Namísto toho, aby byl vybrán největší prvek, se může vzít průměr nebo součet. Praxe ukazuje, že max pooling pracuje lépe.

Funkce pooling tudíž postupně snižuje prostorovou velikost vstupní reprezentace. Na rozdíl od konvoluční vrstvy, vrstva sdružování nemění hloubku sítě.

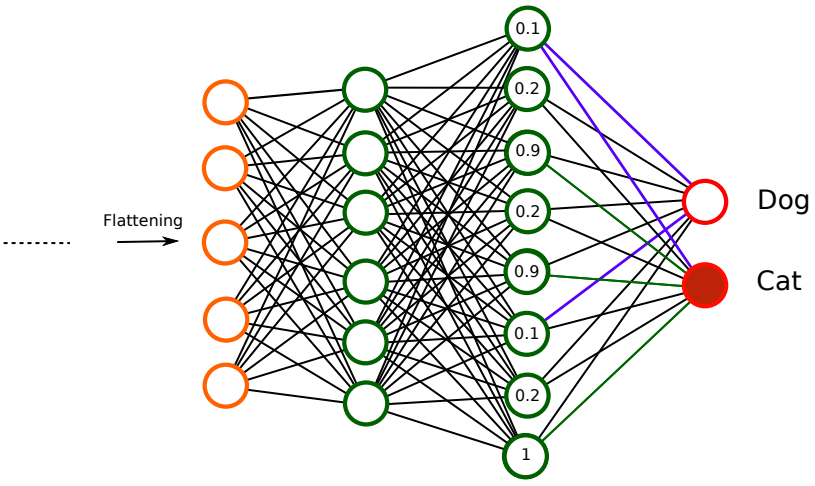


Obrázek 3.4: Na obrázku je znázorněno prostorové sdružování využívající funkci maximum s použitím filtru o velikosti 2×2 [15].

3.1.4 Plně propojená vrstva

Výše zmíněné vrstvy jsou základními stavebními kameny všech CNN. Po vrstvách konvoluce a sdružování se klasifikační část skládá z několika plně propojených vrstev [3]. Tyto propojené vrstvy však mohou přijímat pouze jednodimenzionální data. Vrstva, kterou nazýváme plně propojenou, má za úkol mimo jiné zplošťovat data do vektoru [15].

Neurony v plně propojené [22] vrstvě mají plné spojení se všemi aktivacemi v předchozí vrstvě. Tato část je v zásadě stejná jako běžná neuronová síť. Tato vrstva bere vstupní objem (bez ohledu na to jaký je výstup) a vydává n-dimenzionální vektor, kde n je počet tříd, které budou klasifikovány. Způsob, jakým tato plně propojená vrstva funguje, spočívá v tom, že se dívá na výstup předchozí vrstvy (které představují aktivační mapy na vysoké úrovni) a určuje, které funkce nejvíce korelují s určitou třídou.



Obrázek 3.5: Plně propojená vrstva – vstupní vrstva obsahuje vektor dat, který byl vytvořen v kroku zploštění. V tomto vektoru jsou zakódovány znaky, které vznikly v průběhu předchozích kroků. Úlohou umělé neuronové sítě je vzít tato data a kombinovat je do širší škály atributů, díky nimž je konvoluční síť schopna klasifikovat obrazy. [23]

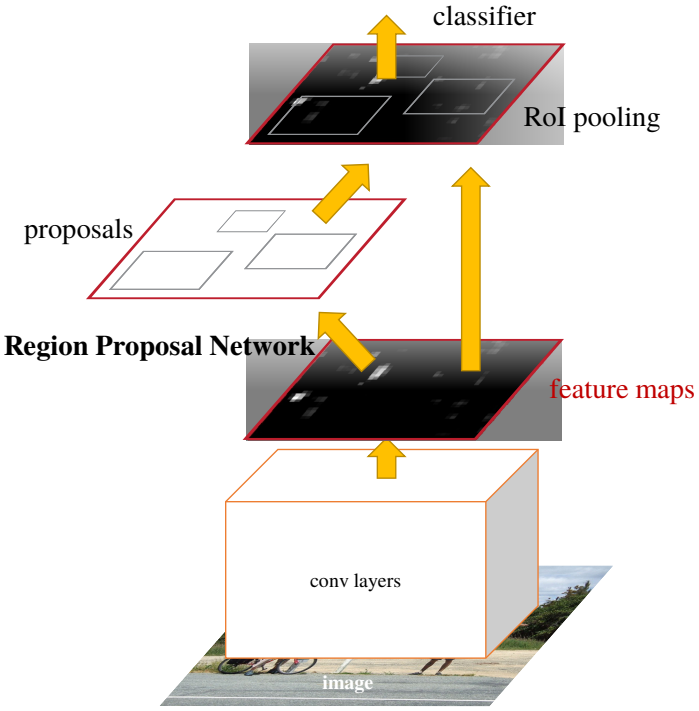
Typů konvolučních neuronových sítí již existuje dnes velké množství. V následujících kapitolách 3.2 – 3.4 jsou popsány konvoluční neuronové sítě, kterými jsem se v rámci bakalářské práce zabýval pro potřeby detektoru dopravních značek.

3.2 Faster R-CNN

Faster R-CNN [8] obsahuje dvě sítě. Síť pro oblast návrhu sítě (RPN – Region Proposal Network – pro generování regionálních návrhů) a síť využívající tyto návrhy k detekci objektů.

Hlavní rozdíl oproti předchozím verzím Fast R-CNN a R-CNN spočívá v tom, že Faster R-CNN používá později selektivní vyhledávání pro generování regionálních návrhů. Časové náklady na generování regionálních návrhů jsou mnohem menší v RPN než v selektivním vyhledávání. RPN sdílí většinu výpočtů se sítí detekce objektů, řadí oblastní boxy a navrhuje ty, které s největší pravděpodobností obsahují objekty.

Obraz je poskytován jako vstup do konvoluční sítě, které poskytuje konvoluční mapu znaků [7]. Namísto použití selektivního vyhledávacího algoritmu na mapě prvků pro identifikaci regionálních návrhů se používá samostatná síť pro předpovídání regionálních návrhů. Návrhy předpovězených oblastí jsou pak přetvořeny za použití sdružovací vrstvy, která je použita pro klasifikaci obrazu v navrhované oblasti a predikci hodnot posunu pro ohraničující rámečky.

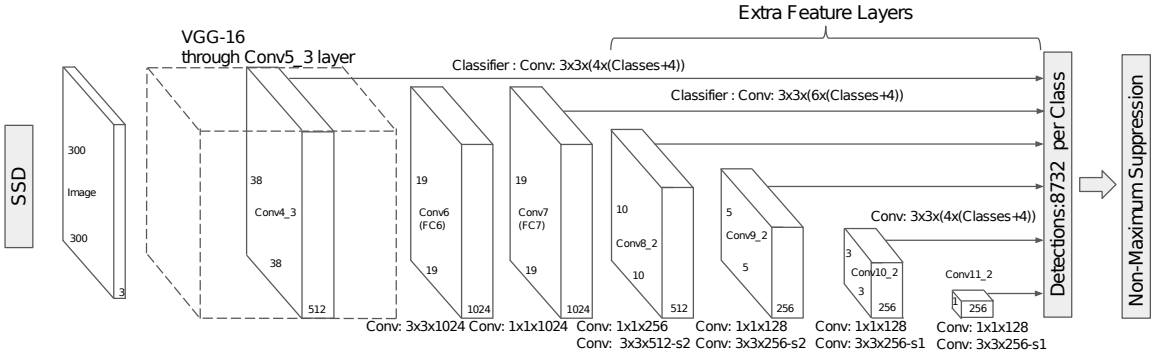


Obrázek 3.6: Faster R-CNN [7] – Vstupní obraz je zpracováván konvoluční sítí a získané příznaky jsou předány RPN. Nad vybranými návrhy je poté provedena klasifikace.

3.3 Single Shot Multibox Detector – SSD

Princip SSD (Single Shot Multibox Detector) [10] je založen na dopředné konvoluční síti, která produkuje sadu ohraničujících rámců s pevnou velikostí a skóre pro přítomnost tříd objektů v těchto rámcích. Následuje krok potlačení ne-maxim (non-maxima suppression) pro vyfiltrování detekcí a vytvoření konečné detekce.

První síťové vrstvy jsou založeny na standardní architektuře používané pro vysoce kvalitní klasifikaci obrazu. Základem se stala síť VGG16, která byla ořezána o plně propojené vrsty. Za tyto vrstvy jsou přidány další konvoluční vrstvy. Schéma sítě SSD je znázorněno na obrázku 3.7. Tyto vrstvy se postupně zmenšují. Predikce detekcí je prováděna na různých vrstvách. To pomáhá detekovat objekty s různou velikostí a přináší nepopíratelně lepší vliv na celkový počet nalezených objektů.



Obrázek 3.7: Schéma modelu SSD s velikostí vstupu 300×300 [10]. Na obrázku jsou znázorněny přidané konvoluční vrstvy, které jsou přímo propojeny s detekční vrstvou. Také si můžeme povšimnout, že se jednotlivé vrstvy postupně zmenšují, což napomáhá k detekci objektu různých velikostí.

3.4 You Only Look Once – YOLO

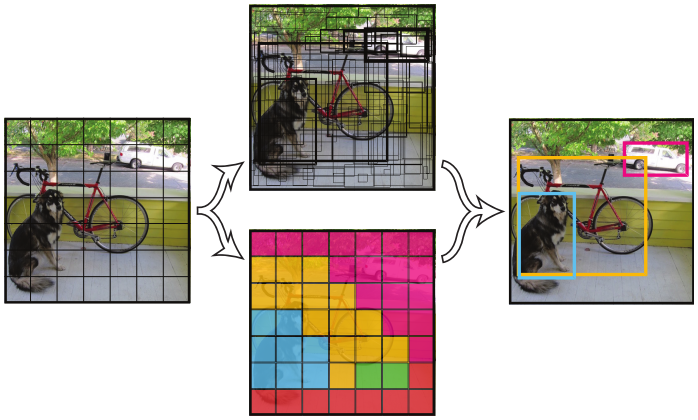
YOLO [17] na rozdíl od ostatních detektorů (faster R-CNN, SSD, atd.), které pracují při detekci ve dvou fázích (vyhledávání hraničních oblastí objektů a jejich následné rozpoznávání), řeší detekci objektu jako jediný regresní problém. Obsahuje pouze jednu konvoluční síť, která současně předpovídá ohraničující rámce a pravděpodobnost předpovědí pro tyto rámce.

Tato architektura sítě sleduje celý obraz v době trénování a testování, takže jeho předpovědi jsou ovlivněny globálním kontextem v obraze. Tento sjednocený model má několik výhod ve srovnání s ostatními detekčními systémy. Na druhou stranu vznikají specifické nároky na vhodnou datovou sadu. O této problematice bude pojednáno níže v sekci 4.3. Yolo na rozdíl od R-CNN využívá pouze jednu síť, která je zodpovědná jak za detekci, tak i za klasifikaci. Díky tomu dosahuje větší rychlosti než již zmiňované faster R-CNN.

Při detekci algoritmus YOLO rozdělí vstupní obraz na mřížku o velikosti $S \times S$. V případě, že se střed objektu vyskytuje v dané buňce, je následně tato buňka zodpovědná za detekci pro tento objekt. Každá buňka následně predikuje určitý počet rámců a skóre důvěry pro tyto rámce. Skóre důvěry je pravděpodobnost, s jakou si je detektor jistý, že se daný objekt v této oblasti vyskytuje.

Každý ohraničující rámec se skládá z pěti předpovědí. Jedná se o souřadnice (x, y, w, h, predikce spolehlivosti). Souřadnice x, y představují relativní střed pole vzhledem k hranicím

buňky mřížky. w, h je šířka a výška vzhledem k celému obrazu a poslední souřadnicí je skóre důvěry.



Obrázek 3.8: YOLO – detekce jako jeden regresní problém. Rozděluje obraz do mřížky $S \times S$ a pro každou z nich předpovídá ohraničující rámce B, spolehlivost těchto polí a pravděpodobnost třídy C. Tyto předpovědi jsou kódovány jako *tenzor* $S \times S \times (B \times 5 + C)$. Převzato z [17].

3.4.1 YOLOv2

YOLO [18] trpí různými nedostatky v porovnání s nejmodernějšími detekčními systémy. Analýza chyb YOLO ve srovnání s Fast R-CNN ukazuje, že YOLO vytváří významný počet lokalizačních chyb. SSD disponuje vyšší přesností a zpracováním v reálném čase. YOLOv2 je druhá verze nástroje YOLO. Zaměřuje se především na zlepšení lokalizace a rychlosti při zachování přesnosti klasifikace.

Aby bylo dosaženo potřebné rychlosti detekce, vyvarovali se autoři YOLOv2 použití hlubší sítě, která by sice přinesla lepší výsledky v množství detekovaných objektů, ale snížila by rychlost detekce. Namísto toho zvolili strategii zjednodušování, která má pozitivní vliv i na snazší učení.

Normalizace dávek trénovacích dat (batch normalization) vede k výraznému zlepšení konvergence a zároveň eliminuje potřebu další regularizace. Oproti původnímu modelu byly tedy odstraněny dropout metody zabraňující přetrénování. Přidáním dávky normalizace na všech konvolučních vrstvách došlo ke zlepšení mAP o více než 2%.

Všechny nejmodernější metody detekce používají klasifikátor učený na ImageNet[1]. Většina klasifikátorů pracuje na vstupních snímcích menších než 256×256. Předchozí verze YOLO trénovala na datech s rozměry 224×224. Systém YOLOv2 pracuje s rozlišením 448×448 pro 10 epoch v systému ImegeNet. To dodává síti čas, aby byly filtry efektivněji využity na vstupu s vyšším rozlišením. Tato změna přispívá k detekci menších objektů. Klasifikace s vysokým rozlišením navýšila mAP téměř o 4 %.

Byly odstraněny plně propojené vrstvy. K predikci ohraničení dochází pomocí *anchor boxes*, podobně jako u Faster R-CNN. S přidáním *anchor boxů* se změnilo rozlišení na 416×416. Jelikož tento model používá pouze konvoluční a sdružovací vrstvy, je možné velikost upravovat za běhu. Aby bylo YOLOv2 robustní na obrázcích s různou velikostí, je potřeba tyto skutečnosti zahrnout již při trénování. Namísto fixace velikosti vstupního obrazu, mění

[1] https://arxiv.org/pdf/1409.0575.pdf

Model	mAP [%]	FPS
R-FCNN	51,9	12
SSD500	46,5	19
YOLOv3 608x608	57,9	20
YOLOv2 608x608	48,1	40
SSD300	41,2	46
YOLOv3-tiny	33,1	220
Tiny-YOLO	23,7	244

Tabulka 3.1: Porovnání mAP a FPS jednotlivých nástrojů pro konvoluční neuronové sítě

síť tuto velikost každých několik iterací. Pro každých 10 dávek sítě se vybere náhodná velikost obrazu. Velikost obrazu musí být v násobcích 32. Tento režim nutí síť k dobré předpovědi napříč různými vstupními rozměry.

YOLOv2 předpovídá detekce na aktivační mapě 13×13. Lichý počet byl zvolen z důvodu předpokladu, že hledaný objekt se častěji vyskytuje ve středu obrazu. Je tedy efektivnější označit střed pouze jednou buňkou než pomocí čtyř.

3.4.2 YOLOv3

Hlavním rozdílem YOLOv3 [19] vůči předchozí verzi YOLOv2 je změna nástroje pro predikci tříd. Předchozí verze využívala pro predikci tříd funkci softmax. Omezením funkce softmax je vzájemná výlučnost tříd objektů, což v některých případech také odpovídá skutečnosti. Nelze totiž označit třeba objekt auto za auto i letadlo zároveň. U detekce více tříd se však můžeme setkat s tím, že jednotlivé třídy objektů se mohou překrývat. Můžeme například predikovat, že se jedná o objekt osoba a zároveň se může jednat o ženu.

Autoři pro tuto verzi nahradili funkci softmax funkcí cross-entropy. Tento způsob je vhodný u komplexnějších datových sad a zaručuje možné překrývání tříd. V tomto modelu je použita nová síť. Předchozí model obsahoval 19 konvolučních vrstev, nová verze jich obsahuje 53. Jedná se o kombinace konvoluční vrstvy 3×3 a 1×1. Jelikož síť obsahuje 53 konvolučních vrstev byla pojmenována Darknet-53. V porovnání se sítěmi ResNet-101 a Resnet-152 dosahuje síť Darknet53 srovnatelné přesnosti a zároveň dvakrát vyšší rychlosti.

3.4.3 YOLOv3-Tiny

Poslední popsanou verzí bude YOLOv3-Tiny. Nedosahuje takové přesnosti jako YOLOv3, poskytuje ale několikanásobně větší rychlost. Jeho síť na rozdíl od YOLOv3 obsahuje pouze 10 konvolučních vrstev.

Porovnání jednotlivých verzí YOLO s ostatními nástroji pro detekci je znázorněno v tabulce 3.1.

3.5 Výběr detekční sítě

V dopravě je důležitá nejen rychlost detekce, ale i kvalita detekce. Potřebujeme tedy detektor, na který se bude možné spolehnout a který je schopen vyhodnotit detekce v co nejkratším čase. Při výběru detekční sítě byl kladen hlavní důraz právě na tyto vlastnosti (porovnání jednotlivých sítí je vidět v tabulce 3.1). Pokud se zaměříme na kvalitu detekce, jako první možnost se jeví výběr YOLOv3. Dosahuje mAP 57.9, což je nejvyšší hodnota

z porovnávaných sítí. Pokud se ale zaměříme na rychlost a náročnost na výpočetní výkon, nebyl nástroj YOLOv3 shledán jako optimální pro tuto práci. Z důvodu preference rychlé detekce byl nakonec zvolen model YOLOv3-tiny, který sice nedosahuje takové přesnosti, konkrétně dosahuje hodnoty 33.1% mAP, oproti tomu však nabízí hned několikanásobně vyšší rychlost. To byl hlavní důvod, proč byla tato síť vybrána pro konečné trénování.

3.6 Hodnocení kvality a výkonnosti detektorů založených na CNN

Abychom zjistili kvalitu natrénovaného modelu, bylo nutné ho podrobit testu. Na zhodnocení výsledků detekce existují speciální nástroje. Testování modelu vetšinou probíhá nad testovací datovou sadou a výsledky predikované detekce jsou následně shrnuty prostřednictvím hodnoty nebo pomocí grafického zobrazení. Čím větší testovací datová sada, tím komplexnější budou výsledky celkové detekce. Výsledná kvalita kvantitativního vyhodnocování použitého modelu vychází z toho, jaká datová sada na testování je použita. V případě použití malé datové sady, která by obsahovala pouze 100 obrázků a detektor by chyboval pouze v jednom z nich, by byla míra úspěšnosti detekce 99%. A to z důvodu, že jeden obrázek tvoří v tomto případě pouze jedno procento celkové datové sady. V takovém případě nemají výsledky dostatečnou vypovídající schopnost o modelu. Ve 100 obrázcích nebudou zcela jistě zahrnuty všechny situace, se kterými se detektor může setkat v běžném provozu. Výsledná úspěšnost může být tudíž hodně hluboko pod touto hranicí. Je tedy nutné připravit dostatečně velkou a rozmanitou datovou sadu. Aby bylo možné modely vzájemně porovnat, je potřeba využít nástroje, které se používají ke kvantitativnímu vyhodnocení. Nástroje, které jsem použil pro tuto práci, budou popsány níže.

Při validaci se můžeme setkat s terminologií označující úspěšnost detekce [13].

- True Possitive (TP) – Skutečně pozitivní – zjištěné případy výskytu odpovídají skutečnosti. Neboli objekt byl detekován správně.
- True Negative (TN) – Skutečně negativní – zjištěné případy nepřítomnosti znaku odpovídají skutečnosti. Objekt, který neměl být detekován, skutečně detekován nebyl.
- False Possitive (FP) – Falešně pozitivní – odpovídá falešnému poplachu. Objekt, který byl detekován, byl detekován špatně nebo detekován být neměl.
- False Negative (FN) – Falešně Negativní – odpovídá přehlédnutému výskytu. Objekt, který měl být detekován, detekován nebyl.

3.6.1 IoU – Intersection over Union

Prvním z nástrojů, využívaných při kvantitativním vyhodnocování, je Intersection over Union (IoU). IoU [27] je metrika používaná ke zjištění přesnosti detektoru objektů. Pro výpočet IoU jsou potřebné dvě sady ohraničujících boxů:

1. originální rámeček, který přesně definuje, kde se hledaný objekt nachází
2. rámeček, který vznikl predikcí detektoru

Podle vzorce $\text{IoU}(A, B) = \frac{A \cap B}{A \cup B}$ se vypočítá hodnota IoU. Jako hraniční pro uznání detekce se většinou používá hodnota 0.5. Znamená to, že poměrné zastoupení plochy průsečíků obou rámců vůči celkové ploše obou rámců je větší jak 50%. V tomto případě považujeme detekci za dostatečně přesnou, abychom ji mohli uznat jako True Possitive. Cokoliv pod touto hranicí je bráno, že nebylo detekováno. Takový případ lze označit jako False Negative.

IoU dokáže vyhodnotit přesnost detekce, ale neposkytuje informace potřebné ke globálnímu zhodnocení kvality modelu nad testovací datovou sadou. K tomuto účelu jsem zvolil nástroj Receiver Operating Characteristic (neboli ROC křivku).

3.6.2 ROC křivka – Receiver Operating Characteristic

Receiver Operating Characteristic [9] neboli ROC křivka [13] [16] je nástroj pro hodnocení a optimalizaci klasifikačního systému, který vyjadřuje vztah mezi specificitou a senzitivitou daného testu nebo detektoru pro všechny přípustné hodnoty.

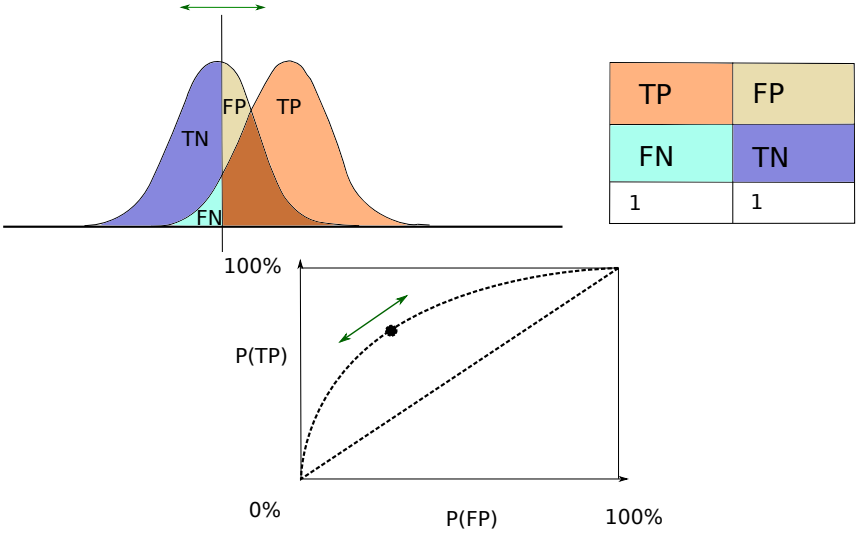
- Senzitivitu lze vyjádřit pomocí vzorce $P = TP/(TP + FN)$. Senzitivita je poměr správně pozitivních pozorování ku všem pozitivním případům, které byly detekovány.
- Specificita je poměr správně negativních pozorování ku všem negativním případům. Specificitu můžeme vyjádřit zlomkem $TN/(TN + FP)$.

Princip použití ROC křivky bude pro snadnější pochopení ilustrován na příkladu z lékařství. Mějme dvě skupiny obyvatel. Konkrétně zdravé a nemocné. Dejme tomu, že by se test, zda se jedná o nakaženou osobu, dal provést na základě měření teploty. Levá křivka na obrázku 3.9 by nám znázorňovala část populace, u které byla naměřená teplota nižší, jak určitá hraniční hodnota a jedná se tudíž o tu zdravou část populace. Křivka napravo by nám znázorňovala část populace, která je opravdu nakažena a její teplota se pohybuje podstatně výše nad zmíněnou hraniční hodnotou. V praxi to však většinou takto ideální není. Jak se dá předpokládat, teploty v obou skupinách se překrývají. Část populace tedy bude klasifikována špatně. Část pozorování bude falešně negativní nebo falešně pozitivní. Jejich podíl se bude měnit v závislosti na volbě hodnoty prahu. Pokud posuneme práh vlevo ve směru zelené šipky, jak je znázorněno v obrázku 3.9, počet FN se zmenší, ale oproti tomu bude mít počet FP rostoucí tendenci. Z tabulky na obrázku 3.9 lze vyčíst, že podíl TP a FN v součtu nabývá hodnoty 1 (plocha pod pravou křivkou). Stejně součet podílu FP a TN nabývá hodnoty 1 (plocha pod levou křivkou).

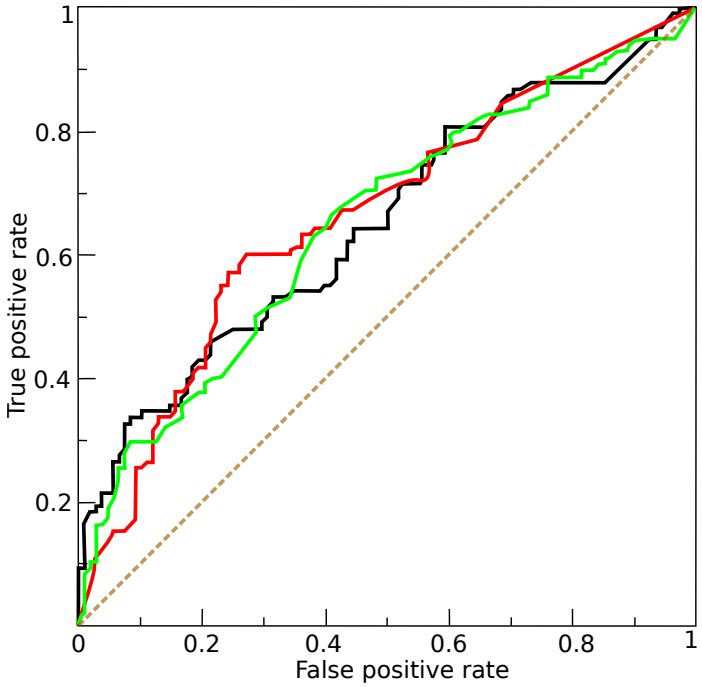
Na obrázku 3.10 je znázorněna ROC křivka. Jedná se o graf, kde na ose Y je vynesena hodnota TPR (True Possitive Rate). Jinými slovy, jedná se o relativní četnost skutečně pozitivních případů TP. Tedy pravděpodobnost, že jako správný bude vyhodnocen pozitivní případ. Na ose X je vynesen FPR (False Possitive Rate). Jedná se o relativní četnost falešně pozitivních případů FP, tedy pravděpodobnost, že jako správný bude vyhodnocen negativní případ. Jednotlivé body, vynesené na ROC křivce, odpovídají hodnotám jednotlivých prahů.

V grafu na obrázku 3.10 je možné vidět čárkovanou tenkou čáru přímé úměry. Jedná se o hranici s níž můžeme detekce porovnat. Tato čára přímé úměry znázorňuje výsledek náhodného výběru. Pokud bude naše výsledná křivka ležet v oblasti nad touto čarou, znamená to, že detektor pracuje lépe než náhodný výběr. Naopak pokud bude naše výsledná křivka ležet v oblasti pod touto čarou, jedná se o alarmující výsledek a znamená to, že náš detektor funguje hůře, než náhodný výběr.

Nastavováním různých prahových hodnot hledáme na ROC křivce ideální poměr mezi množstvím falešně pozitivních a falešně negativních pozorování. Nebo jinými slovy senzitivitou a specificitou. Zde záleží na konkrétním případu využití. Důležitost správné volby



Obrázek 3.9: Znázornění ROC křivky a závisloti volby prahu na grafu a tabulce. [16]



Obrázek 3.10: Ukázka ROC křivky [16]

prahu je možné znázornit na použití policejního radaru. V tomto případě je lepší, pokud se některé z aut jedoucí rychle přehlédne, než aby dostal pokutu řidič, který jel povolenou rychlostí, ale systém to chybně vyhodnotil. Další příklad lze uvést z lékařství, kde je lépe prověřit více pacientů, kteří hledanou chorobou netrpí, než aby bylo vyšetření některého z pacientů zanedbáno. To by mohlo mít pro pacienta fatální následky.

3.6.3 Mean Average Precision – mAP

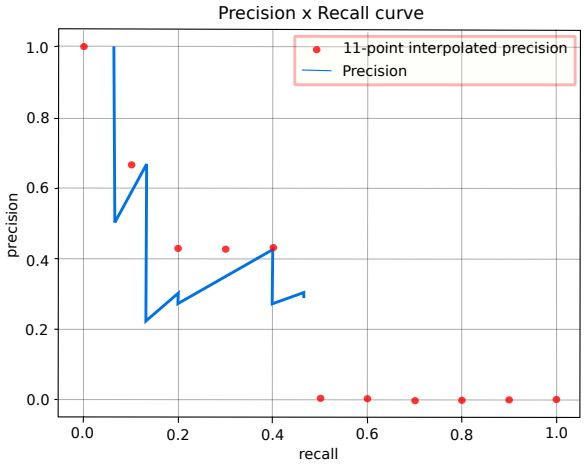
Aby bylo možné vypočítat střední průměrnou přesnost (mAP) [6][14] v detekci objektů, je nutné vypočítat nejprve průměrnou přesnost (Average Precision – AP) pro každou třídu a následně vypočítat průměr AP nad všemi třídami. Abychom mohli vykreslit graf Precision–Recall, ze kterého získáme AP, musíme spočítat nejprve hodnoty precision a recall pro všechny prahové hodnoty stejně jako v případě ROC křivky. Precision je schopnost modelu identifikovat pouze relevantní objekty. Recall je schopnost modelu najít všechny relevantní případy (všechny anotované objekty). Hodnoty precision a recall spočítáme následovně:

$$\begin{aligned} \text{Precision(P)} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{Recall(R)} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \end{aligned} \tag{3.2}$$

Existují dva různé přístupy vzorkování P-R křivky.

Výpočet 11-ti bodové interpolace

V případě 11-ti bodové interpolace se vybere maximální hodnota P získaná pro každý R' >= R, kde R jsou vzorky Recall pro různé prahové hodnoty (0,0.1,0.2,...,1). AP je pak průměrná přesnost všech prahových hodnot.



Obrázek 3.11: Výpočet AUC pomocí bodové interpolace [14]

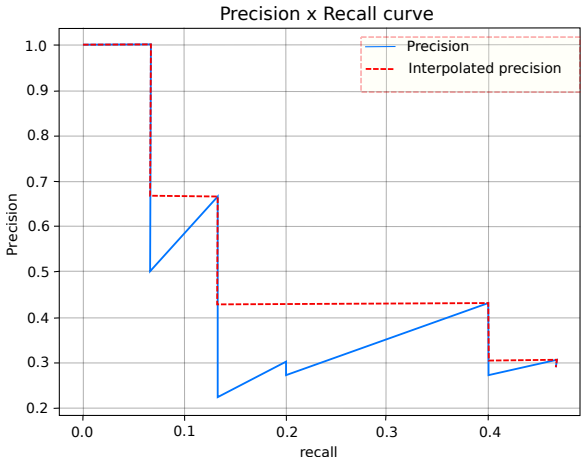
Použitím 11-ti bodové interpolace na příkladu z obrázku 3.11:

$$\text{AP} = \frac{1}{11} \sum_{\text{r}\epsilon\{0,0.1,0.2,\ldots,1\}} \rho\text{interp(r)}$$

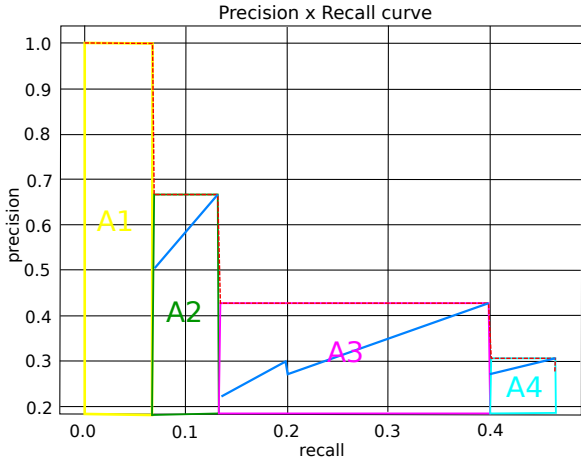
$$\text{AP} = \frac{1}{11}(1 + 0.666 + 0.4285 + 0.4285 + 0.4285 + 0 + 0 + 0 + 0 + 0 + 0)$$

Výpočet interpolace nad všemi body

Pro výpočet interpolace nad všemi body volíme maximální P pro libovolné R' >= R, zatímco R patří do všech hodnot recall. AP je pak velikost plochy pod křivou P-R. Záměrem je snížit dopad kroutící se křivky.



Obrázek 3.12: Výpočet AUC pomocí interpolace nad všemi body [14]



Obrázek 3.13: Výpočet AUC pomocí interpolace nad všemi body – rozložení výpočtu [14]

Výpočet AP znázorněný v příkladu na obrázcích 3.12 a 3.13:

$$AP = A1 + A2 + A3 + A4$$

$$AP = ((0.0666-0)\times 1)+((0.1333-0.0666)\times 0.6666)+((0.4-0.13333)\times 0.4285)+((0.4666-0.4)\times 0.3043)$$

$$AP = 24.56\%$$

Kapitola 4

Implementace a trénování

Seznam všech vytvořených nebo mnou upravovaných souborů je v souboru README.md, které je také vloženo na konci této práce.

4.1 Konfigurace souborů pro trénování modelu

Pro vlastní detektor byl vybrán nástroj YOLO. Tento nástroj je implementován pomocí několika frameworků, které lze volně stáhnout a upravovat. Pro vlastní implementaci jsem vybral framework Darknet. Darknet je open source framework neuronové sítě napsaný v jazyce C s využitím *CUDA*, který podporuje výpočet CPU i GPU. Pro plné využití tohoto frameworku na grafické kartě je nutné nainstalovat několik nástrojů. Prvním z nich je *NVIDIA CUDA Toolkit*. NVIDIA CUDA Toolkit je sada nástrojů, které obsahují knihovny s akcelerací GPU a nástroje pro ladění a optimalizaci. Další modul, který bylo nutno nainstalovat byla knihovna *NVIDIA CUDA Deep Neural Network* (cuDNN). Jde o GPU akcelerovanou knihovnu pro hluboké neuronové sítě, která poskytuje vysoce vyladěné implementace pro standardní rutiny, jako jsou dopředné a zpětné konvoluce, sdružování, normalizace a aktivační vrstvy. Posledním z potřebných nástrojů je knihovna *OpenCV*. Jedná se o svobodnou a otevřenou multiplatformní knihovnu pro manipulaci s obrazem, která je zaměřená především na počítačové vidění a zpracování obrazu v reálném čase. Podporuje náhled detekce v obraze nebo videu.

Samotný framework Darknet k úspěšnému trénování modelu detektoru nestačí. Nástroj YOLO potřebuje několik specifických souborů, aby věděl, jak a co trénovat. Konkrétně se jedná o soubory s příponou .data, .names ,.cfg a soubory pro trénování a testování modelu.

Pro detektor jsem zvolil síť YOLOv3-Tiny z důvodu, které již byly uvedeny výše v kapitole 3.5. Bylo tedy nutné stáhnout soubor *.cfg* a vložit jej do složky *cfg/*. Stažení tohoto souboru je možné z oficiálních stránek frameworku Darknet. Soubor *.cfg* obsahuje popis architektury sítě a řadu konfiguračních parametrů využívaných při trénování modelu. Tento soubor je potřeba modifikovat na základě vlastností detektoru.

V tomto souboru lze případně modifikovat hodnoty batch a subvision. Hodnota batch nám udává kolik snímků v jedné iteraci bude zpracováno. Hodnota subvision nám rozděluje sadu snímku z hodnoty batch na tzv. mini-batch (menší části) a ty posílá do grafické karty na zpracování. Tyto hodnoty také mimo jiné nastavují využití operační paměti GPU. Pokud jsou tyto hodnoty zvoleny nevhodně a paměť na grafické kartě nemá dostatečnou kapacitu, můžeme se setkat s chybovou hláškou CUDA Error: out of memory. V mém případě jsem pro trénování nastavil batch na hodnotu 24 a subvision na hodnotu 8. V tomto souboru

je nutné zohlednit také počet tříd, které budou detekovány. Tomu se přizpůsobuje počet filtrů v poslední konvoluční vrstvě. V tomto konfiguračním souboru bylo nutné modifikovat konkrétně řádky 127 a 171 na hodnotu udávající onen zmíněný počet filtrů. Tato hodnota byla vypočtena dle vzorce $filters = (classes + 5) * 3$. Na řádcích 135 a 177 je nutné přepsat hodnotu na počet tříd, který bude detekován. Čísla řádků platí pouze pro verzi sítě YOLOv3-tiny. Dalším parametrem, který lze v tomto souboru modifikovat, je maximální počet iterací. Jakmile je dosaženo nastaveného počtu iterací, jsou uloženy finální váhy a trénování končí. Maximální počet iterací se nastavuje pomocí parametru max_batches. Tento parametr byl inkrementován na hodnotu 2 500 000. V tomto souboru lze také zvolit velikost vstupního obrazu. Nastavuje se pomocí parametrů weight(šířky) a height(výšky). Hodnoty těchto dvou parametrů musí být dělitelné 32. Výchozí hodnota je 416×416. Pro lepší a jemnější detekci jsem velikost vstupního souboru nastavil na 1280×1280.

Druhý soubor, který je nutné vytvořit je soubor .names. Tento soubor obsahuje názvy jednotlivých tříd v takovém formátu, kde každé jméno třídy je vždy na samostatném řádku.

Třetím souborem potřebným pro trénování je soubor s příponou .data. Tento soubor musí mít následující strukturu:

```
classes = [number of clases]
train = train.txt
valid = test.txt
names = [file].names
backup = [backup saving folder]
```

Na prvním řádku tohoto souboru nastavíme počet tříd, který bude detekován. Další parametry *train* a *valid* nám určují textové soubory, které budou využity pro trénovaní nebo validaci a obsahují seznam obrázků pro tyto úkony. O vytváření těchto souborů bude pojednáno níže v rámci této kapitoly. Jako parametr *names* zadáme jméno souboru .names (cesta z kořenového adresáře Darknet/[file_name].names), jehož popis byl uveden výše a který obsahuje jména pro jednotlivé třídy. V posledním parametru *backup* nastavíme cestu, kam se budou ukládat již vypočtené váhy. Váhy se ukládají po určitých iteracích. Periodu ukládání iterací lze nastavit v souboru *src/detector.c*.

Dalšími soubory, které jsou potřebné k trénování a testování, jsou soubory *train.txt* a *test.txt*. Soubor *train.txt* obsahuje všechny názvy obrazků s relativní či absolutní cestou, které se využívají na trénování modelu. Soubor *test.txt* lze využít pro následné testování modelu a je stejné struktury jako soubor *train.txt*. Na vytváření těchto souborů byl napsán pomocný skript (*train-test-maker.py*) v jazyce Python, který tyto soubory vytvoří a postupně do nich zapíše všechny názvy obrázků v zadané složce i s cestou. Lze si zvolit poměr, v jakém budou obrázky rozděleny do souborů *train.txt* a *test.txt*.

4.2 Průběh validace

Pro zhodnocení modelu jsem vytvořil program kvantitativního vyhodnocování v jazyce Python. Vyhodnocování detektoru probíhá nad vlastní testovací datovou sadou, která obsahuje reálné snímky z dopravy. Snímky jsou pořízeny nafocením značek v ulicích nebo záznamem z palubní kamery automobilu. Malá část datové sady byla pořízena pomocí aplikace Google StreetView. Pro testovací datovou sadu jsem pořídil 899 fotek, které obsahují přibližně 1 600 anotovaných značek.

Vytvořený program určuje kvalitu detekcí a na základě získaných údajů vykreslí ROC křivku a grafy, které vypovídají o úspěšnosti modelu. Dále program vypočte hodnoty re-

flektující úspěšnost modelu v číslech. Validaci jsem rozdělil na dvě části. První program se věnuje detailnímu testování určitého modelu a primárním úkolem druhého programu je vzájemné porovnání většího množství modelů mezi sebou.

4.2.1 Varianta 1 – detailní testování modelu detektoru

Validace probíhá následujícím způsobem. Nejdříve je spuštěn framework Darknet s příznakem *valid*, který vygeneruje pro každou třídu soubor do složky results/. Každý soubor obsahuje názvy obrazů z validační datové sady, kde byla detekována tato třída, míru se kterou si je detektor jistý, že se jedná o danou třídu a souřadnice predikovaného rámce. Pro kvantitativní vyhodnocování jsou využívány programy *validation.py* a *RoC.py*. V prvním kroku je volána funkce, která zajišťuje výpočet hodnot pro následné vykreslení ROC křivky. Výpočet ROC křivky probíhá na základě vstupních požadavků, kdy se provede zvolený počet iterací nad vygenerovanými soubory a v každé další iteraci se zvýší hodnota prahu (tzv. thresh), jejíž velikost je závislá na počtu iterací. Vygenerované soubory jsou dočasně duplikovány, aby mohly být modifikovány, aniž by to ovlivnilo další kroky. Program prochází soubory s ručně anotovanými značkami z testovací datové sady a postupně se snaží ve vygenerovaných detekčních souborech najít odpovídající detekce, které mají větší práh, než je aktuálně nastavený pro daný krok. Pokud byla detekce nalezena a výsledek IoU originálního a predikovaného rámce je větší jak 0.5, jedná se o true positive (TP), hodnota správně nalezených detekcí je inkrementována a záznam z vygenerovaných souborů je odstraněn. Před výpočet IoU musí být souřadnice ve formátu Darknet převedeny do formátu VOC. Princip převodu a formát jednotlivých anotací bude popsán v kapitole 4.3.2. V případě, že nebyla detekce ve vygenerovaných souborech nalezena, i když se v ručně anotovaných záznamech nachází, jedná se o false negative(FN). Pokud je soubor k obrázku z testovací datové sady prázdný, znamená to, že se v tomto obraze nevyskytují žádné detekovatelné značky a program by neměl najít výskyt tohoto obrazu ani ve vygenerovaných souborech. V tomto případě se jedná o true negative (TN). Na konci se soubory vygenerované frameworkem Darknet projdou a vše co zbylo spadá do kategorie false positive (FP). Tento proces probíhá pro každou hodnotu prahu mezi 0 a 1. Počet prahových hodnot, pro který je tento proces prováděn, závisí na vstupních parametrech validačního programu. Výstupem je graf, který znázorňuje ROC křivku. Na ose X se vykresluje true positive rate a na ose Y je vynesen počet false positive pro celou testovací datovou sadu. Podle hodnoty přesnosti ACC (celková správnost predikce klasifikátoru) je nakonec vybrán nejlepší práh a ten je následně použit pro konečnou validaci, kdy obrazy s predikovanými i originálními rámci jsou uloženy do složky IoU_results/. Pro lepší znázornění nakonec program uloží také dva výsečové grafy, které nám reprezentují poměr TP, FP, FN, TN. Nakonec je vygenerován textový soubor, který obsahuje hodnoty vypovídající o celkové úspěšnosti detektoru. Tento soubor konkrétně obsahuje:

- počet TP, FN, FP, TN
- TPR (True Positive Rate) – míra skutečně pozitivních výskytů
- TNR (True Negative Rate) – míra skutečně negativních výskytů
- FPR (False Positive Rate) – míra falešně pozitivních výskytů
- FNR (False Negative Rate) – míra falešně negativních výskytů

- PPV – prediktivní hodnota pozitivního testu. Určuje pravděpodobnost, že jev je skutečně pozitivní, když test vyšel pozitivně
- ACC – odráží celkovou správnost predikce klasifikátoru, tj. pravděpodobnost správné predikce rovnou poměru počtu správných rozhodnutí k celkovému počtu rozhodnutí
- F1 – skóre F1 lze interpretovat jako vážený průměr precision (podíl správně pozitivních výsledků ku všem pozitivním) a recall (podíl mezi počtem správných pozitivních výsledků a počtem pozitivních výsledků, jež měly být vráceny). Skóre F1 dosahuje své nejlepší hodnoty při hodnotě 1 a nejhorší při 0.
- mAP - hodnota reflektující míru a kvalitu detekce nad testovací datovou sadou. mAP bylo podrobněji popsáno v kapitole 3.6.3.

4.2.2 Varianta 2 – porovnání většího množství modelů detektorů

Verze kvantitativního vyhodnocování, která se zaměřuje na porovnání více detektorů mezi sebou, pracuje na podobném principu jako verze první. Z důvodu časové a výpočetní náročnosti testování modelu nad testovací datovou sadou byl program rozdělen na dvě oddělené části. První část se věnuje samotnému testování a všem výpočtům přesnosti a úspěšnosti modelu. Druhá část programu má za úkol vypočtená data zobrazit. Je tedy možné nechat spočítat složité výpočty validace na serveru, který má vyšší výkon a samotné zobrazení provést na lokální stanici. Důvody, proč jsou tyto dvě klíčové části takto rozděleny, jsou z hlediska výkonu a úspory času a z důvodu, že na serverech, kde se validace obvykle provádí chybí potřebné knihovny, které by byly schopny data přímo vykreslit.

Při spuštění je první části programu (all-validation.py) předán název složky obsahující váhové modely, které mezi sebou chceme porovnat. Následně tento program spustí pro každou váhu framework Darknet s příznakem *valid*. Dojde k vygenerování souborů obsahujících všechny detekce, jak již bylo zmíněno ve variantě 1 v kapitole 4.2.1. Následně na stejném principu probíhá výpočet hodnot ROC křivek a to pouze s tím rozdílem, že data nejsou zobrazena, ale jsou ukládána do výstupního souboru typu *pickle* (Soubor typu pickle umožňuje ukládat objekty, které je možné později znovu načíst). Tímto způsobem je vygenerován soubor pro každý váhový model, který obsahuje data pro vykreslení ROC křivky. Druhá část programu (draw-graphs.py) slouží pro vykreslení grafu. Po spuštění programu dojde k načtení hodnot ROC křivek ze souborů a ty jsou následně vyneseny do jednoho grafu. Křivce je pro lepší vizualizaci přiřazena barva, jenž se v grafu dosud nenachází. Všechny křivky jsou vykresleny v jednom grafu pro snazší porovnání. K tvorbě grafů byla použita knihovna *bokeh*.

4.2.3 Implementace exportu videa

Aby mohlo být anotované video exportováno a uloženo, bylo potřeba tuto funkci doimplementovat do frameworku Darknet. Byl přidán přepínač *-out* který povolí uložení videa. Pro implementaci byl použit programovací jazyk C++. Ve frameworku Darknet bylo ukládání videa doimplementováno do souboru *image_opencv.cpp* do funkce *show_image_cv*. Všechny volání této funkce musely být modifikovány pro potřeby předávání parametrů potřebných k exportu videa. Pro export videa je nutné vytvořit objekt *VideoWriter*. Je nezbytné specifikovat název výstupního souboru, kód FourCC (4-bytový kód, který slouží k určení video kodeku), počet snímků FPS a velikost rámce. Protože většina videí, na kterých jsem detektor testoval, byla pořízena v 25 FPS, zvolil jsem pro exportované video

pevnou hodnotu 25 FPS. Pro export videa jsem vybral kodek MJPG. Výsledné video je exportováno ve formátu *.avi* s rozlišením 1920×1080.

4.3 Trénování a tvorba datové sady

Pro trénování modelu je potřeba nashromáždit vhodnou datovou sadu. Převážně se jedná o obrázky obsahující objekty, které mají být detekovány. Pro trénování pomocí nástroje YOLO musí být obrázky ve formátu *.jpg*. Ke každému obrázku musí existovat anotační soubor *.txt*. Tento soubor obsahuje popis všech objektů na obrázku, které by měl detektor umět detekovat a klasifikovat.

Textový soubor s anotacemi musí mít následující formát: [číslo třídy] [souřadnice X středu objektu] [souřadnice Y středu objektu] [výška objektu] [šířka objektu]. Převod a podrobnosti tohoto formátu budou popsány v kapitole 4.3.2.

4.3.1 Existující datové sady

První pokusy trénování modelu probíhaly na již existujících datových sadách. Většina datových sad, které jsou dnes volně k dispozici, není však příliš vyhovující. Byly použity datové sady German Traffic Sign Recognition Benchmark (GTSRB)[1] a BelgiumTS Dataset[2]. Datová sada GTSRB obsahuje trénovací sadu čítající celkem 39 209 obrázků obsahujících 43 tříd značek a testovací sadu, která obsahuje 12 630 obrázků. Datová sada BelgiumTS obsahuje 4 575 obrázků v trénovací sadě a 2 520 obrázků v testovací sadě. Ukázku z těchto datových sad je možné vidět na obrázku 4.1. Je zřejmé, že tyto datové sady obsahují ve velkém množství pouze ořezané značky. První pokusy trénování detekčního modelu na těchto datových sadách nedopadly úspěšně. YOLO se při tréninku učí z celého obrázku a zahrnuje také velikost objektu, jeho umístění v obraze, okolí a další aspekty. V této datové sadě, jejíž ukázku vidíme na obrázku 4.1, tyto aspekty však chybí. Výsledkem bylo, že detektor identifikoval pouze ty značky, které stejně jako na trénovací datové sadě zabíraly více jak 80% plochy vstupního obrazu a zbytek značek nebyl detekován.

Obrázek 4.1: Ukázka volně dostupných datových sad z externích zdrojů[1][2]. Tyto datové sady však nejsou příliš vyhovující pro trénování detektoru s použitím nástroje YOLO.

[1] http://benchmark.ini.rub.de/
[2] https://btsd.ethz.ch/shareddata/

4.3.2 Generátor syntetické datové sady a tvorba vlastního reálné datové sady

Z nedostatku volně dostupných vhodných reálných datových sad jsem vytvořil svojí vlastní syntetickou datovou sadu.

V programovacím jazyce Python jsem vytvořil program na generování datových sad. Tento program na svém vstupu vyžaduje dvě složky. První z nich obsahuje podsložky, ve kterých se nachází vzorky dopravních značek dané třídy, které budou vkládány do obrazu. Z důvodu napodobení reálných případů jsem vybral ukázky značek přímo z provozu. Druhá složka obsahuje sekvenci snímků, které byly pořízeny z palubní kamery jedoucího vozidla. Pro generování obrazů z videa jsem vytvořil v programovacím jazyce Python pomocný skript, který podle zadaných parametrů uloží pouze n-tý snímek ze zdrojového videa. Pro tyto účely byla použita hodnota 25, která se jevila jako nejvýhodnější. Při snímací frekvenci použitých kamer 25 FPS byla ukládána pouze každá sekunda ze vstupního videa. Na výstupu generátoru datové sady jsou pak vytvořené obrazy obsahující značky. Společně s každým obrazem je vytvořen také anotační soubor. Při generování souboru je potřeba zjistit o jakou třídu značky se jedná a vygenerovat souřadnice umístění značky ve formátu Darknet.

Program převezme několik vzorků reálných značek a postupně je vkládá do připravené sekvence snímků. Cílem bylo napodobit běžné umístění značek v provozu. Značky z reálného provozu byly před spuštěním generátoru upraveny tak, že okolí značek je obarveno černou barvou, která je následně při vkládání vyklíčována. Okraje značek jsou ve výsledném obrázku ostré a nedochází k prolínání vrstev. Za účelem dosažení co nejreálnějšího výsledku byly značky vkládány na náhodná místa s náhodnou velikostí.

Původní plán byl generovat velikost značky v závislosti na umístění v obraze. Čím by byla značka více uprostřed, tím by byla menší. Více ke krajům by se velikost značky zvětšovala, což by simulovalo přibližování a míjení značky z palubní kamery auta. Ve skutečném provozu se však ukázalo, že tuto teorii nelze do takové míry předpokládat, protože plně neodpovídá všem umístěním dopravních značek. Po použití takto vygenerované vlastní datové sady však nebyly výsledky příliš uspokojivé. Oproti první verzi datové sady, která obsahovala pouze ořezané značky, se v tomto případě zlepšila schopnost detekce. Pokrok nastal ve schopnosti detekovat značky různých velikostí. Většina značek, které neměly dobrý kontrast a světelné podmínky, stále však nebyla detekována. Při testování tohoto modelu ve videu z běžného provozu nedocházelo téměř k žádným detekcím.

Při porovnání vygenerované datové sady se záběry z reálného provozu jsem došel k závěru, že na detekci má velký vliv osvětlení a odstín barvy značek. I přesto, že datová sada obsahovala značky, které byly pořízeny v ulicích, nebyly v datové sadě zahrnuty zdaleka všechny možnosti osvětlení. Zvláště u videa bylo patrné, že značky nejsou tak sytých barev a nemají dobrý kontrast. Z těchto důvodů byla v další verzi generátoru syntetické datové sady měněna značkám hodnota jasu. Některé značky byly tmavší a jiné světlejší, což mělo simulovat různé světelné podmínky během dne. Tento pokus prokázal, že se jedná o krok správným směrem. Množství detekovaných značek se nad testovací datovou sadou zvýšilo. Největší pokrok nastal ve schopnosti detekce značek ve videu. Oproti předchozí verzi, kdy nedocházelo skoro k žádné detekci, byly nyní již některé značky detekovány. Detektor však stále nedosahoval takové kvality, aby se na případné detekce bylo možné spolehnout.

Z těchto důvodů byla v dalším kroku do generátoru přidána simulace vnějších vlivů ve větší míře. Vkládaným značkám byly upravovány gamma korekce a kontrast. Tyto úpravy měly za úkol simulovat situace, kdy se dopravní značka nachází ve stínu, nebo je detekce

prováděna v šeru či za tmy. Naopak zvýšením hodnoty gamma korekce se simulovalo dopadající sluneční světlo na značku, ale také různé odchylky v odstínech barev způsobené stářím značek. Dále došlo také k přidání šumu a rozmazání pomocí blur filtru za účelem detekce rozmazaných značek, které mohou vzniknout při snímání obrazu kamerou při vyšších rychlostech nebo rychlejších změnách směru.

V provozu nejsou podmínky k detekci vždy ideální. Často se můžeme setkat s tím, že je nějaká značka pod jiným úhlem, nebo je poškozena a její pozice tudíž není vždy vertikální a kolmo směrem k pozici pozorování. Z toho důvodu byly na vkládaných značkách provedeny deformace prostřednictvím otáčení značky kolem vertikální a horizontální osy. Po trénování modelu s touto verzí datové sady se výsledky výrazně zlepšily. Nadpoloviční většina značek z testovací datové sady již byla detekována. Stále však míra detekce nebyla uspokojivá. Porovnání jednotlivých verzí datových sad a vliv jednotlivých transformací na kvalitu a míru detekce budou podrobněji popsány a znázorněny níže v sekci 5.

Ukázku generování konečné verze datové sady lze vidět na obrázku 4.2. Konečná verze generátoru funguje následovně. Podle vstupních parametrů je vybrán zdroj značek a pozadí a určí se, které deformace se mají při generování použít. Míra všech deformací je vybírána náhodně v pevně stanoveném rozmezí. Vygeneruje se náhodná velikost a pozice značky. Před vložením značky do obrazu jsou nad ní provedeny všechny vybrané deformace. Okolí vkládané značky je vyklíčováno a značka je následně vsazena do výsledného obrazu. Zároveň dojde k vytvoření anotačního souboru .txt, který disponuje stejným názvem jako generovaný obraz. Zde musíme převést pozici vkládané značky do formátu darknet. Způsob převodu souřadnic VOC do formátu darknet lze vidět na následující ukázce kódu.

```
1  def convert(size, box):
2      dw = 1.0/size[0]
3      dh = 1.0/size[1]
4      x = (box[0] + box[1])/2.0
5      y = (box[2] + box[3])/2.0
6      w = box[1] - box[0]
7      h = box[3] - box[2]
8      x = x*dw
9      w = w*dw
10     y = y*dh
11     h = h*dh
12     return (x,y,w,h)
```

Výpis 4.1: Ukázka převodu souřadnic VOC do souřadnic Darknet v jazyce Python

Na řádcích 2-3 je výpočet koeficientů používaných k přepočtu souřadnic VOC do formátu Darknet. Vyjadřují hodnoty *1/šířka celého obrazu* a *1/výška celého obrazu*. Na řádku 4 a 5 probíhá výpočet souřadnic x a y, které v tomto případě vyjadřují střed označovaného objektu. Na řádcích 6 a 7 probíhá výpočet šířky a výšky označovaného objektu. Tyto souřadnice jsou stále v hodnotách počtu pixelů. Dále, jak můžeme vidět na řádcích 8–12, jsou tyto hodnoty vynásobeny koeficienty, které byly spočítány na řádcích 2 a 3. Nyní dostáváme souřadnice pro objekt ve formátu Darknet. Následně tyto hodnoty musíme seřadit do správného pořadí a přidat třídu objektu, o který se v daném případě jedná. Konečný formát pro anotační soubor Darknet je následující:

[číslo třídy] [souřadnice X středu objektu] [souřadnice Y středu objektu] [výška objektu] [šířka objektu].

Ukázka formátu anotačního souboru vypadá následovně:

```
5 0.652104 0.324713 0.106796 0.178161
```

Mezi velkou výhodu syntetické datové sady patří její široká použitelnost. V každé zemi se dopravní značky liší. V některých zemích jsou rozdíly markantní. Není tedy možné zaručit správné fungování detektoru natrénovaného na jedné konkrétní datové sadě pro všechny země. V případě potřeby natrénovat detektor pro jinou zemi by to s použitím běžné datové sady znamenalo datovou sadu znovu pořídit v dané zemi a ručně anotovat. Jedná se však o velmi časově náročný proces. S použitím tohoto generátoru syntetické datové sady tato potřeba odpadá. V generátoru stačí pouze vyměnit malý vzorek značek pro konkrétní zemi a vygenerovat datovou sadu znova.

4.3.3 Vlastní reálná datová sada

Další trénování modelu nad syntetickou datovou sadou lepší výsledky již nepřineslo a to ani po rozšíření datové sady dalším generováním. Bylo tedy nutné datovou sadu nějakým způsobem inovovat. Aby byla míra detekce uspokojivější, rozšířil jsem syntetickou datovou sadu o anotované snímky z reálné datové sady. Malá část datové sady byla převzata od Ing. Tomáše Svobody [20], většina reálné datové sady byla pořízena ze záznamů palubní kamery automobilu a nafocením dopravních značek v ulicích. Záznamy byly pořízeny v různých lokalitách, aby obsahovaly větší rozmanitost prostředí. Datová sada obsahuje záznamy z Jihlavy a okolí, Prahy, Brna a Zlína. Ty byly následně ručně anotovány. K anotování značek byl použit nástroj Yolo_mark-maser od AlexeyAB[3]. Ovládání tohoto programu bylo následně lehce modifikováno pro snadnější použitelnost. Celkem bylo anotováno cca 8 000 snímků, s tím, že každý snímek obsahuje průměrně 2-3 značky. Právě přidání reálné datové sady přineslo tížené výsledky. Počet úspěšně detekovaných značek se razantně zvýšil, zatímco počet false negativů a false positivů byl výrazně nižší. Porovnání ROC křivek před a po přidání reálné datové sady do syntetické bude podrobněji znázorněno a popsáno níže v sekci 5.

4.3.4 Trénování modelu YOLOv3-Tiny

Trénování modelu vycházelo již z předtrénovaných vah darknet53.conv.74. Přetrénovaní již existujících vah na vlastní problém detekce je rychlejší a efektivnější, než školit celý model úplně od začátku. Jelikož je trénování modelu náročný proces z hlediska výpočetního výkonu, bylo k trénování využito virtuální organizace MetaCentrum VO. Tato organizace nabízí studentům a akademickým pracovníkům možnost bezplatně využít jejich výpočetních zdrojů. K trénování bylo zapotřebí vytvořit skripty obsahující příkazy pro spuštění dané úlohy. Tyto skripty byly následně zařazeny do fronty (v tomto případě konkrétně gpu fronty), kdy po uvolnění zdrojů byly dané úlohy automaticky spuštěny. Pro potřeby trénování s využitím frameworku Darknet bylo nutné importovat před trénováním moduly cuda-9.0 a cudnn-7.0. Trénování probíhalo na clustreru (výpočetním uzlu) *konos*. K trénování bylo využito 6 procesorů cores Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, grafická karta GPU NVIDIA GeForce GTX 1080 Ti, 16-32 GB operační paměť a 10 GB dočasného uložiště. Doba spuštění skriptu pro trénování je v této frontě *gpu* omezena na 24 hodin. Poté bylo nutné skript modifikovat a spustit pokračování trénování. V případě velkého vytížení zdrojů byla využita také fronta gpu_long. Úlohy v této frontě jsou spouštěny na jiném typu clusteru, který disponuje grafickými kartami nVidia Tesla K20 5GB (Kepler). V praxi bylo toto trénování modelu pomalejší. Výhoda této fronty však spočívala v kratší čekací době na přidělení zdrojů a v možnosti nechat běžet trénování až 165 hodin. Průměrný čistý čas

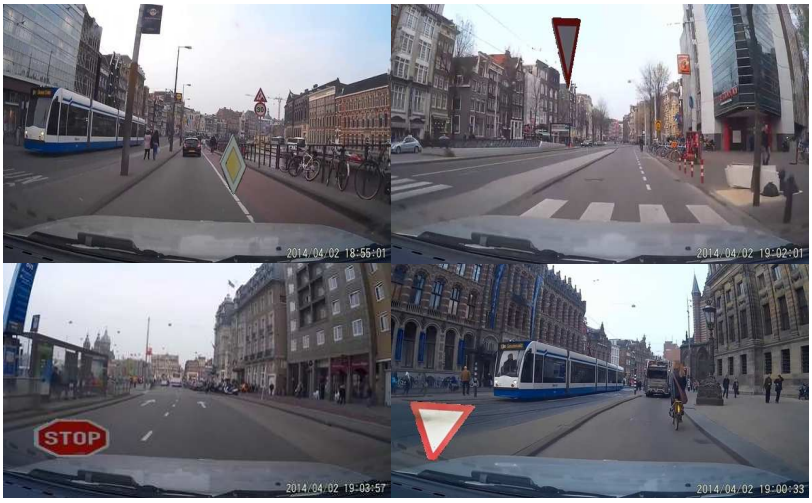
[3] https://github.com/AlexeyAB/Yolo_mark

trénování jednoho modelu detektoru, bez zahrnutí čekací doby na přidělení zdrojů, čítal přibližně 170 – 200 hodin.

4.3.5 Ukázka syntetické a reálné datové sady

V následujících obrázcích jsou zobrazeny ukázky datových sad použitých při trénování. Na obrázku 4.2 se nachází ukázka generované syntetické datové sady zahrnující kombinaci všech deformací. V každém obrázku je použita deformace otáčením kolem osy, změna hodnot jasu, úprava gamma korekcí, přidání rozmazání a šumu v náhodné míře. Na obrázcích 4.3 – 4.7 pak můžeme vidět detailněji jednotlivé deformace dopravních značek. Vlivy jednotlivých deformací na výsledky detekce budou podrobněji popsány v kapitole 5. Ukázku reálné datové sady lze vidět na obrázku 4.8

Obrázek 4.2: Syntetická datová sada vytvořená vlastním generátorem. Celá tato ukázková datová sada byla vygenerována s použitím všech výše zmíněných deformací.



Obrázek 4.3: Na tomto obrázku je znázorněna deformace dopravního značení otáčením kolem horizontální a vertikální osy. Tato deformace má za úkol simulovat situace, kdy je dopravní značka snímána pod úhlem nebo je dopravní značka poškozena nebo její pozice není kolmo ke směru pozorování.

Obrázek 4.4: Na tomto obrázku je znázorněna deformace dopravního značení s použitím tzv. blur filtru. V podstatě jde o rozmazání značky, které má simulovat detekci za vysoké rychlosti či při prudkých změnách směru. Rozmazání ve snímaném obraze vzniká technickými omezeními kamery ve vozidle.

Obrázek 4.5: Na tomto obrázku je znázorněna deformace pomocí změny hodnot gamma korekcí. Tato deformace má za úkol simulovat situace, kdy se značka nachází ve stínu nebo naopak na slunečním svitu. Tato deformace má také za úkol simulovat vybledlé barvy značek, které jsou způsobeny různým stářím dopravních značek. Cílem této deformace je simulovat i povětrnostní podmínky. Pokud je slunečný den, kontrast značek na záznamu z kamery je daleko větší, než například pokud je zataženo, nebo prší. V této situaci značky více splývají s podkladem.

Obrázek 4.6: Na tomto obrázku je znázorněna deformace pomocí změny úrovně jasu. Tato deformace má za úkol, podobně jako změna gamma korekcí, simulovat situace, kdy se značka nachází na přímém slunečním svitu a dopravní značky s vybledlými barvami, které jsou způsobeny různým stářím dopravních značek.

Obrázek 4.7: Na tomto obrázku je znázorněna deformace pomocí přidání šumu. Tato deformace má za úkol simulovat situace, kdy kamera ne zcela kvalitně zaznamená detekovanou značku. Tím ztěžuje podmínky detektoru již v průběhu trénování, aby měl možnost se na tuto situaci předem adaptovat.

Obrázek 4.8: Ukázka reálné datové sady čítající cca 8 000 anotovaných snímků. Datová sada byla pořízena z kamerového záznamu z automobilu, nafocením dopravních značek v ulicích a prostřednictvím aplikace Google StreetView. V datové sadě je zahrnuto více lokací z území ČR. Konkrétně byly záznamy pořízeny v Jihlavě a okolí, Brně, Praze a Zlíně. V datové sadě jsou zahrnuty obrázky z centra měst, z dálnic nebo okresních silnic. V této datové sadě jsou zahrnuty různé světelné a povětrnostní podmínky, déšť, mlha, šero, tma a další.

Kapitola 5

Experimentální výsledky a zhodnocení vlastností detektoru

5.1 Porovnání syntetické a reálné datové sady a jejich vliv na detekci

První pokusy trénování detektoru nad vlastní datovou sadou byly provedeny pomocí syntetické datové sady. Pro trénování byla použita datová sada se všemi deformacemi. Ta obsahovala deformace otáčením značek kolem vertikální a horizontální osy, úpravy úrovně jasu, změny gamma korekcí, rozmazání a přidání šumu. Výsledky jsou znázorněny pomocí ROC křivky v grafu na obrázku 5.1. Konkrétně se jedná o křivku fialové barvy. Model dosahoval při použití této datové sady úspěšnosti $58,43\,\%$ mAP. Další podrobnější výsledky je možné vidět v tabulce 5.1. V této datové sadě bylo pro trénování zahrnuto celkem 192 226 snímků.

Druhý pokus vytrénovat model detektoru byl pro změnu proveden pouze na reálné datové sadě. Tato datová sada obsahovala 7 862 snímků. Ty byly pořízeny za různých světelných podmínek. Jedná se o obrázky pořízené v ulicích či záznamem z kamery jedoucího auta. Výsledek tohoto modelu můžeme vidět na ROC křivce na obrázku 5.1 a jsou znázorněny zelenou křivkou. Tento model dosáhl nad testovací datovou sadou úspěšnosti $51,96\,\%$ mAP. Zajímavým výsledkem bylo, že samotná syntetická datová sada dosáhla o něco lepší úspěšnosti, než pouze reálná datová sada a to i přes fakt, že reálná datová sada obsahuje snímky podobné těm, které jsou použité k testování modelu. Z těchto výsledků můžeme dojít k závěru, že syntetická datová sada s využitím všech deformací věrně napodobuje situace, které detektor musí následně řešit v reálném provozu. Další výsledky pro porovnání jsou rovněž uvedeny v tabulce 5.1.

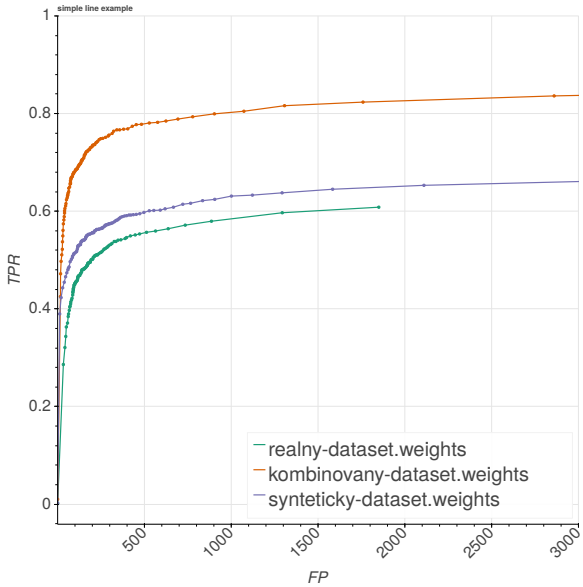
Výsledky detekce se syntetickou nebo reálnou datovou sadou však nedosahovaly velké úspěšnosti. Pro další trénování modelu detektoru jsem obě tyto datové sady (syntetickou a reálnou) spojil do jedné. Datová sada obsahovala celkem 201 933 obrázků. Zastoupení reálné a syntetické datové sady bylo v poměru 1:20. Spojení obou datových sad přineslo poměrně velký pokrok v úspěšnosti detekce. Nad testovací datovou sadou bylo dosaženo hodnoty $79,9\,\%$ mAP. Grafické znázornění úspěšnosti tohoto modelu je vyneseno v grafu na obrázku 5.1 oranžovou křivkou. Jak je patrné v tabulce 5.1, spojením obou datových sad bylo docíleno většího množství detekovaných značek. Počet správně detekovaných značek se zvýšil a zároveň se snížila hodnota falešné detekce o $25\,\%$.

Model	mAP[%]	TPR	TNE	PPV	ACC	F1	TP	FN	FP	TN
S	58,43	0.56	0,014	0,56	0,51	0,666	811	650	141	2
R	51,96	0,574	0,013	0,574	0,497	0,666	835	620	226	3
K	79,9	0,79	0,013	0,79	0,718	0,67	1172	310	151	2

Tabulka 5.1: Porovnání výsledků syntetick0é (S), reálné (R) a kombinované (K) datové sady.

Model	mAP[%]	TPR	TNE	PPV	ACC	F1	TP	FN	FP	TN
S	58,43	0,62	0.014	0,62	0,557	0,666	810	503	141	2
R	51,96	0,58	0,013	0,576	0,49	0,665	758	559	226	3
K	79,9	0,838	0,013	0,838	0,754	0,666	1123	217	151	2

Tabulka 5.2: Porovnání výsledků syntetické (S), reálné (R) a kombinované (K) datové sady bez semaforů.

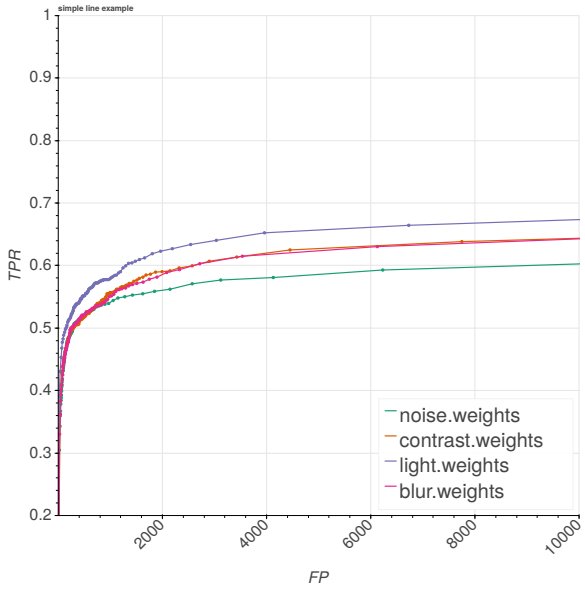


Obrázek 5.1: Porovnání ROC křivky modelů vytrénovaných na syntetické, reálné a kombinované datové sadě. Syntetická datová sada zde dosahuje o něco lepší úspěšnosti než v případě použití pouze reálné datové sady. Nejlepších výsledků však bylo dosaženo kombinací těchto dvou datových sad. Tato kombinovaná datová sada je zde znázorněna pomocí oranžové křivky.

5.2 Porovnání deformací syntetické datové sady a jejich vliv na úspěšnost detekce

Do syntetické datové sady byly přidávány různé druhy deformací. V této části bude podrobněji popsáno porovnání výsledků jednotlivých deformací a jejich vliv na detekci. Byly vytrénovány různé modely nad syntetickou datovou sadou, které obsahovaly vždy pouze jeden druh deformace. Porovnání jednotlivých modelů je zřejmé z grafu 5.2. Nejmenším přínosem pro větší míru detekce bylo přidání šumu. Nejvíce přispěla k větší míře detekovaných objektů změna hodnot úrovně jasu vkládaných dopravních značek. Syntetická datová

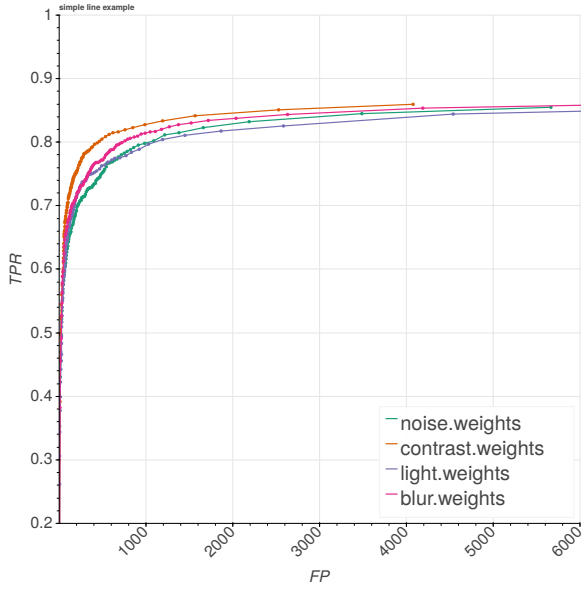
sada má omezený počet značek, který používá při generování. To znamená, že v konečné datové sadě je například 5 vzorků jedné dopravní značky, nad kterými jsou následně prováděny deformace. Podle mého názoru přispěla k detekci nejvíce deformace hodnoty úrovně jasu z toho důvodu, že i v reálném provozu dochází k častému přechodu světelných podmínek, což není možné v takové míře reflektovat pouze na několika málo reálných vzorcích značek použitých pro generování datové sady.



Obrázek 5.2: Porovnání ROC křivek modelů vytrénovaných nad syntetickou datovou sadou při použití pouze jednoho druhu deformace dopravních značek

5.3 Porovnání deformací kombinované syntetické a reálné datové sady

Další experiment byl proveden s přidáním reálné datové sady. Byly natrénovány jednotlivé modely, které obsahovaly syntetickou datovou sadu pouze s jedním druhem deformace, podobně jako v minulém experimentu. Ke každé datové sadě byla však přidána reálná datová sada obsahující záznamy ze skutečného provozu. Jak je patrné z grafu 5.3, situace se oproti předešlému experimentu výrazně změnila. Při použití pouze syntetické datové sady přinesla největší zlepšení změna hodnoty úrovně jasu. S použitím reálné datové sady již ale není přínos této deformace pro detekci tak markantní. Dovolím si tvrdit, že důvodem je přítomnost velkého množství značek s rozdílnými podmínkami osvětlení obsažených v reálné datové sadě. Proto se velká část jasových změn už v datové sadě promítla. Při použití kombinace s reálnou datovou sadou měla největší přínos deformace pomocí změn kontrastu. Změna kontrastu vkládané dopravní značky vizuálně nejen zesvětluje, ale také ztmavuje. V datové sadě jsou následně obsaženy značky, které daleko více splývají s prostředím, jejich tvary a charakteristické rysy nejsou tak výrazné a jsou pro detektor náročnější na detekci. Dle mého názoru právě tato deformace přispívá tím, že ztěžuje podmínky detektoru již v průběhu trénování, aby měl možnost se na tuto situaci předem adaptovat.



Obrázek 5.3: Porovnání ROC křivek modelů vytrénovaných nad syntetickou datovou sadou v kombinaci s reálnou datovou sadou. Syntetická datová sada obsahovala pro každý model pouze jeden druh deformace dopravních značek.

5.4 Zhodnocení natrénovaného detektoru a jeho úspěšnosti

Nejlepší výsledky pro detektor byly dosaženy při použití kombinace obou výše uvedených datových sad. Při nejlepší použité prahové hodnotě bylo detekováno 87 % dopravních značek z testovací datové sady. Bylo dosaženo hodnoty 79, 9 % mAP. Detektor v běžných situacích nemá problém dopravní značku detekovat. Detektor je plně použitelný pro detekci dopravních značek a byl testován i v reálném provozu, kde dosahoval rychlost zpracování 18 FPS. Na obrázku 5.4 je vidět, že detektor zvládá i poměrně náročné detekce. Jsou detekovány i velmi malé značky na značnou vzdálenost. Například semafor se na obrázku opravdu nachází, objekt je ale tak malý, že ho rámeček detekce celý překryl. Detektor rozeznává dopravní značení i za zhoršených povětrnostních podmínek. Také většina značek, které byly hodně pod úhlem ke směru pozorování, byla bez problému detekována. Zde z velké části přispívá deformace otáčení značek kolem horizontální a vertikální osy. I přesto, že ve většině případů byl detektor úspěšný, nastaly případy, ve kterých selhal. Jelikož trénovací datová sada zahrnovala pouze 34 tříd vybraných dopravních značek, dochází občas k záměně velmi podobných dopravních značek. Tento efekt je patrný na sérii obrázků 5.5 v levém horním rohu. Zde detektor chybně vyhodnotil informační tabuli jako parkoviště kvůli velkému zastoupení modré barvy. Také jsou problémové značky různých speciálních druhů zákazů vjezdu. Ty byly trénovány jako jedna třída a občas dojde k záměně za značku maximální povolené rychlosti v důsledku přítomnosti číslice. Většina dopravních značek je ale detekována. Největší problémy činí detektoru rozpoznání semaforů. Semafor má velké zastoupení černé barvy a proto není pro detektor tak výrazný jako barevně odlišené dopravní značky. Také se často setkáváme s případy, kdy je několik semaforů vedle sebe. Netvoří tedy dokonalý obrys semaforu, ale spíše tvoří dohromady jeden velký černý objekt. Z těchto důvodů mohou být pro detektor hůře rozpoznatelné.

Obrázek 5.4: Ukázky detekcí náročných případů

5.5 Budoucí vývoj práce

Do budoucna by bylo možné míru detekce ještě vylepšit. Bylo by vhodné zaměřit se především na slabá místa detektoru. Jedním z nich je zlepšení simulace osvětlení generátoru syntetické datové sady. Ta v současném stavu umožňuje určitou simulaci vnějšího osvětlení, ale i přes tento fakt dochází k situacím, kdy je dopravní značka v jedné části dne detekována bez problému a v jiné přehlédnuta. Doporučoval bych pro detekci využít generování globálního osvětlení, které by lépe simulovalo podmínky skutečného osvětlení nebo provést analýzu pozadí obrazu na místě vložení značky a dle toho upravit světelné podmínky. Detektor detekoval dopravní značky s poměrně velkou úspěšností, největší problém falešně negativních případů tvořily semafory. Pro další vývoj by bylo vhodné rozšířit datovou sadu, případně upravit architekturu sítě pro snadnější detekování svislých obdélníkových předmětů. Většina detektorů používaných v reálném provozu má problémy především na dálnici,

Obrázek 5.5: Ukázky chybně detekovaných případů

kde dochází k záměně informačních štítků kamionů za značku nejvyšší povolené rychlosti. Díky použití nástroje YOLO, který zahrnuje i globální kontext obrazu, by bylo možné toto selhání odstranit. K tomu je nutné připravit dostatečně velkou datovou sadu, která by obsahovala snímky podobné těmto situacím, čímž by mohly být tyto chybné detekce vyloučeny. Stejným způsobem lze řešit i chybné detekce velkých modrých informačních značek, které jsou v některých případech zaměněny za značku jednosměrné ulice či parkoviště.

Kapitola 6

Závěr

Cílem této práce bylo použít moderní architekturu konvolučních neuronových sítí pro vytvoření detektoru dopravních značek a semaforů . V práci je popsán teoretický základ potřebný k pochopení základního principu a fungování konvolučních neuronových sítí. Pro výsledný detektor byl vybrán nástroj YOLO z důvodu rychlosti a kvality detekce a z důvodu, že do detekce zahrnuje i globální kontext okolí detekovaného objektu. Pro detektor byla použita architektura konvoluční sítě *Yolov3-tiny*, která dosahuje nejlepšího poměru rychlosti a míry detekce. Detektor umožňuje provádět detekci jak ve statickém obraze, tak i ve videu. Detektor dokáže zpracovat video v reálném čase.

V této práci byl detektor trénován na vlastních datových sadách. Vytvořil jsem generátor syntetické datové sady zahrnující různé deformace značek a povětrnostní podmínky způsobující zhoršenou viditelnost. Připravil jsem vlastní reálnou datovou sadu čítající 8 000 ručně anotovaných snímků. Na těchto datových sadách jsem prováděl experimenty. V této práci byl testován vliv zastoupení syntetické nebo reálné datové sady v trénovací datové sadě. Také byl proveden experiment s kombinací obou datových sad. Byly provedeny experimenty zahrnující vždy pouze jeden druh deformace, přičemž byl sledován jejich vliv na celkovou úspěšnost a míru detekce.

Pro hodnocení modelu detektoru byla vytvořena vlastní testovací datová sada čítající přibližně 900 obrázků obsahujících 1 600 anotovaných dopravních značek. Pro kvantitativní vyhodnocování byly vytvořeny programy v jazyce Python, které hodnotí nejen kvalitu detekce daného modelu nad testovací datovou sadou, ale zvládnou zároveň porovnat více modelů mezi sebou. A v neposlední řadě umožňují exportovat příklady detekce ve formě obrázku nebo videa.

Bylo dosaženo 79,9% mAP nad vlastní testovací datovou sadou. Úspěšně bylo detekováno a klasifikováno 84% procent případů z testovací datové sady.

Tato práce byla vybrána a prezentována na studentské konferenci inovací, technologií a vědy v IT Excel@FIT, kde získala ocenění partnera této akce, kterým byla firma CAMEA[1].

[1] http://excel.fit.vutbr.cz/submissions/2019/010/10.pdf

Literatura

[1] Unsupervised Feature Learning and Deep Learning Tutorial. [Online; navštíveno 30.3.2019].
URL http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[2] *CS231n Convolutional Neural Networks for Visual Recognition.* [Online; navštíveno 31.3.2019].
URL http://cs231n.github.io/convolutional-networks/

[3] Cornelisse, D.: An intuitive guide to Convolutional Neural Networks. *MEDIUM*, 2018, [navštíveno 31.3.2019].

[4] Dalal, N.; Triggs, B.: Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, ročník 1, June 2005, ISSN 1063-6919, s. 886–893 vol. 1, doi:10.1109/CVPR.2005.177.

[5] *Počítačová grafika: Diskrétní dvourozměrná konvoluce.* [Online; navštíveno 30.3.2019].
URL http://cgtucna.blogspot.com/2013/11/diskretni-dvourozmerna-konvoluce.html

[6] Everingham, M.; Van Gool, L.; Williams, C. K. I.; aj.: The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, ročník 88, č. 2, Jun 2010: s. 303–338, ISSN 1573-1405, doi:10.1007/s11263-009-0275-4.

[7] Gandhi, R.: R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms. [Online; navštíveno 2.4.2019].
URL https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[8] Gao, H.: Faster R-CNN Explained. *MEDIUM*, 2018, [navštíveno 2.4.2019].

[9] Karimollah Hajian-Tilaki, P.: Receiver Operating Characteristic (ROC) Curve Analysis for Medical Diagnostic Test Evaluation. *Caspian J Intern Med*, 2013, [navštíveno 31.3.2019].

[10] Liu, W.; Anguelov, D.; Erhan, D.; aj.: SSD: Single Shot MultiBox Detector. *arXiv e-prints*, Dec 2015: str. arXiv:1512.02325.

[11] M Swathi, K. V. S.: Behavioural impacts of Advanced Driver Assistance Systems-an overview. *European Journal of Transport and Infrastructure Research*, ročník 1, 01 2001.

[12] Malach, T.; Bambuch, P.; Malach, J.: Detekce obličeje v obraze s využitím prostředí MATLAB. In *Mezinárodní konference Technical Computing Prague 2011*, EBIS, spol. s r.o., 2011, s. 1–6.

[13] Marsland, S.: *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, první vydání, 2009, ISBN 1420067184, 9781420067187.

[14] Padilla, R.: Metrics for object detection. https://github.com/rafaelpadilla/Object-Detection-Metrics.

[15] Prabhu: Understanding of Convolutional Neural Network (CNN)—Deep Learning. *MEDIUM*, 2018, [navštíveno 30.3.2019].

[16] *ROC křivka – WikiSkripta*. [Online; navštíveno 4.4.2019].
URL https://www.wikiskripta.eu/w/ROC_k%C5%99ivka

[17] Redmon, J.; Divvala, S.; Girshick, R.; aj.: You Only Look Once: Unified, Real-Time Object Detection. *arXiv e-prints*, Jun 2015: str. arXiv:1506.02640.

[18] Redmon, J.; Farhadi, A.: YOLO9000: Better, Faster, Stronger. *arXiv e-prints*, Dec 2016: str. arXiv:1612.08242.

[19] Redmon, J.; Farhadi, A.: YOLOv3: An Incremental Improvement. *arXiv e-prints*, Apr 2018: str. arXiv:1804.02767.

[20] Svoboda, B. T.: *Detekce, lokalizace a rozpoznání dopravních značek*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2011.

[21] Swathi, M.; Suresh, K. V.: Automatic traffic sign detection and recognition: A review. In *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*, Feb 2017, s. 1–6, doi:10.1109/ICAMMAET.2017.8186650.

[22] Team, S.: Convolutional Neural Networks (CNN): Step 4 - Full Connection. [Online; navštíveno 31.3.2019].
URL https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection

[23] Team, S.: Convolutional Neural Networks (CNN): Step 4 - Full Connection. [Online; navštíveno 28.4.2019].
URL https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection/

[24] Udofia, U.: Basic Overview of Convolutional Neural Network (CNN). *MEDIUM*, 2018, [navštíveno 31.3.2019].

[25] ujjwalkarn: *An Intuitive Explanation of Convolutional Neural Networks*. [Online; navštíveno 30.3.2019].
URL https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[26] Viola, P.; Jones, M.: Robust Real-time Object Detection. In *International Journal of Computer Vision*, 2001, s. 1–9.

[27] Zitnick, C. L.; Dollár, P.: Edge Boxes: Locating Object Proposals from Edges. In *Computer Vision – ECCV 2014*, editace D. Fleet; T. Pajdla; B. Schiele; T. Tuytelaars, Cham: Springer International Publishing, 2014, ISBN 978-3-319-10602-1, s. 391–405.

Příloha A

Návod na instalaci a spuštění

Yolo-v3 - darknet

Darknet

Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.

For more information see the Darknet project website.

For questions or issues please use the Google Group.

neural network for traffic sign and traffic light detection

Author: Tomáš Chocholatý

- Requirements

0. Improvements
1. Created and edited files
2. How to compile on Linux
3. How to train
4. How to start detect
5. How to valid one model
6. How to valid more model
7. How to use generator for syntetic dataset

Requirements

- Linux
- **CMake** for modern CUDA support: https://cmake.org/download/
- **CUDA 10.0**: https://developer.nvidia.com/cuda-toolkit-archive

- **OpenCV 3.4.1** https://opencv.org/
- **cuDNN >= 9.1 for CUDA 10.0**

Compiling on **Linux** by using command `make`
Examples of results https://www.youtube.com/watch?v=aGYro84K9pg&feature=youtu.be
Excel@FIT paper: http://excel.fit.vutbr.cz/submissions/2019/010/10.pdf

Improvements

- create program in Python for detail testing concrete model
- create program in Python for compare more models success rate
- create program in Python for draw graphs with bokeh library
- create auxiliary program in Pyhton for create files test.txt and train.txt for training
- create and edit files for training (all-signs-generateII.cfg, all-signs-generateII.name, all-signs-generateII.data)
- implement video export to darknet files
- create program in Python for generate synthetic dataset with simulation of degraded conditions and deformations
- create auxiliary program in Python for generate frame from video file

Created and edited files

Create:

- SD/datasets/syntetic-dataset-generator/generator.py
- SD/datasets/frame-video/frameRate.py
- SD/darknet-all/validation.py
- SD/darknet-all/RoC.py
- SD/darknet-all/all-validate-RoC.py
- SD/darknet-all/draw-graphs.py
- SD/darknet-all/all-signs-generateII.names
- SD/darknet-all/all-signs-generateII.data
- SD/darknet-all/test.txt
- SD/darknet-all/train.txt
- SD/dataset/test-dataset/* (annoted test dataset, contains cca 900 images)
- SD/datasets/real-dataset/* (annoted real dataset for training, contains 8000 images)

Edit:

- SD/darknet-all/src/image_opencv.cpp (edit function show_image_cv)
- SD/darknet-all/all-signs-generateII.cfg

How to compile on Linux

Just do make in the darknet directory. Before make, you can set such options in the Makefile

- `GPU=1` to build with CUDA to accelerate by using GPU
- `CUDNN=1` to build with cuDNN to accelerate training by using GPU
- `OPENCV=1` to build with OpenCV - allows to detect on video files and video streams from network cameras or web-cams
- `DEBUG=0` to build debug version of Yolo
- `OPENMP=0` to build with OpenMP support to accelerate Yolo by using multi-core CPU

To run Darknet on Linux use examples from this article, just use `./darknet`, i.e. use this command: ./darknet detector test ./cfg/coco.data ./cfg/yolov3.cfg ./yolov3.weights!

How to train

- you need a dataset with yolo annoted format, you can use this GUI-software for create it: https://github.com/AlexeyAB/Yolo_mark
- create files train.txt and test.txt containts image from dataset
- use pretrained weights
- edit file .cgf

```
change line batch to [‘batch=24‘]
change line subdivisions to [‘subdivisions=8‘]

change lines 135 177: [class number]
change lines 127 171: $filters = (classes + 5) * 3
‘‘‘
filters=117

classes=34
```

Create file `obj.names` in the directory of darknet, with objects names - each in new line

Create file `obj.data` in the directory of darknet, containing (where classes = number of objects):

```
classes= 2
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = backup/
```

start training:

```
./darknet detector train cfg/*.data cfg/*.cfg started-weights
```

How to start detect

```
example: ./darknet detector test all-signs-generateII.data cfg/all-signs-
    ↪ generateII.cfg backup/14.03.19-all-signs-add-real/all-signs-
    ↪ generateII\_1430000.weights validation/real-image-3303.jpg
```

How to valid one model (files: validation.py, RoC.py)

1) Run Darknet with flag valid

```
    example: ./darknet detector valid all-signs-generateII.data cfg/all-
        ↪ signs-generateII.cfg backup/\\14.03.19-all-signs-add-real/all-
        ↪ signs-generateII_1430000.weights
```

2) run validation.py

```
    example: python validation.py -f all-signs-generateII.names -i 100 -n
        ↪ syntetic -r -s -g
```

```
    usage: validation.py -f FILE -i N -n NAME [-t BESTTRASHPARAMETER] [-r]
        ↪ [-g]
                [-s] [-h]

    optional arguments:
      -f FILE, --name FILE name of file contain all class name
      -i N file for statistic
      -n NAME name for graphs
      -t BESTTRASHPARAMETER
                          set manual best trash (default - best trash from
                              ↪ Roc or 0.5)
      -r compute and export RoC + compute mAP
      -g export graph
      -s save annotated images
      -h Show this help message and exit.
```

How to valid more model (files: all-validate-RoC.py, draw-graphs.py)

Program all-validate-RoC.py make valid for all weights in foler over test dataset and create .bin file with result to folder IoU_results_graphs/data-for-graph/[original path of backup file]. Then run program draw-graphs.py, that draw all results from all-validate-RoC.py. Program have flag save or show options and edit axis properties.

```
example: python all-validate-RoC.py backup/bak-exp/bak-exp3-deformation-
    ↪ compare2/ all-signs-generateII.data cfg/all-signs-generateII.cfg
```

```
usage: all-validate-RoC.py [path to folder with weights for compare] [path
    ↪ to file .data] [path to file .cfg]
```

```
example: python draw-graphs.py --input IoU_results_graphs/data-for-graph/
    ↪ bak-exp/bak-exp3-deformation-compare2/ --out plot --save --startx
    ↪ 0.001 --endx 3000
```

```
usage: draw-graphs.py --input INPUT_FILE --out GRAPH_NAME [--save] [--show]
                [--startx STARTX] [--endx ENDX] [--starty STARTY]
                [--endy ENDY] [-h]

optional arguments:
  --input INPUT_FILE path to folder, whitch contain data for gaph from all-
                     validation.py
  --out GRAPH_NAME name for graph export
  --save save graph as .svg
  --show show graph in web browser
  --startx STARTX set first value on x axis
  --endx ENDX set last value on x axis
  --starty STARTY set first value on y axis
  --endy ENDY set last value on y axis
  -h Show this help message and exit.
```

How to use generator for synthetic dataset (generator.py)

This program generate synthetic dataset. You have to choose folder that contain folder with cropped sign (warning - name of folder is use for dataset items name) and you have to choose folder with background picture. You can choose type of deformation. All files is save to folder in param –out.

```
example: python generator.py --signs all-signs-generateII.names --out out -
    ↪ n -l -b -c
```

```
usage: generator.py --signs CLASS_FILE --out OUT [-c] [-l] [-b] [-n] [-h]

optional arguments:
  --signs CLASS_FILE file of signs .names
  --out OUT file of signs .names
  -c random change contrast
  -l random change lightness
  -b random add blur filter
  -n random add noise
  -h Show this help message and exit.
```

How to make frames from video (frameRate.py)

```
example: python frameRate.py {[}name of video in source\_video folder{]}
    ↪ {[}frame
rate{]}
```