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 V OBRAZE A VIDEU

DETECTION OF TRAFFIC SIGNS IN IMAGE AND VIDEO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **FILIP KOČICA**
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: **Kočica Filip**
Program: Informační technologie
Název: **Detekce dopravních značek v obraze a videu**
Detection of Traffic Signs in Image and Video
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou počítačového vidění, zaměřte se na detekci objektů pomocí moderních přístupů, zejména konvolučních neuronových sítí.
2. Prostudujte problematiku detekce dopravních značek v obraze a videu, vyhledejte dostupné datové sady.
3. Experimentujte s různými podobami konvolučních neuronových sítí pro detekci dopravních značek.
4. Interpretujte závěry experimentů, iterativně vylepšujte proces trénování, používané datové sady i architektury detektorů s cílem dosáhnout co nejlepší úspěšnosti detekce.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Bradski G. R., Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, Inc. 2008.
- Ian Goodfellow, Yoshua Bengio, Aaron Courville: Deep Learning. MIT Press, 2016.
- Rafael C. Gonzalez, Richard E. Woods, Digital Image Processing, ISBN: 978-0131687288
- 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 a 4.

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Tato práce řeší problematiku detekce dopravního značení za pomoci moderních technik zpracování obrazu. K řešení byla použita speciální architektura hluboké konvoluční neuronové sítě zvaná YOLO, tedy You Only Look Once, která řeší detekci i klasifikaci objektů v jednom kroce, což celý proces značně urychluje. Práce pojednává také o porovnání úspěšnosti modelů trénovaných na reálných a syntetických datových sadách. Podařilo se dosáhnout úspěšnosti 63.4 % mAP při použití modelu trénovaného na reálných datech a úspěšnosti 82.3 % mAP při použití modelu trénovaného na datech syntetických. Vyhodnocení jednoho snímku trvá na průměrně výkonném grafickém čipu ~ 40.4 ms a na nadprůměrně výkonném čipu ~ 3.9 ms. Přínosem této práce je skutečnost, že model neuronové sítě trénovaný na syntetických datech může za určitých podmínek dosahovat podobných či lepších výsledků, než model trénovaný na reálných datech. To může usnadnit proces tvorby detektoru o nutnost anotovat velké množství obrázků.

Abstract

This thesis deals with the traffic sign detection problematics using modern techniques in image processing. Special architecture of deep convolutional neural network YOLO, i.e. You Only Look Once, which performs both detection and classification in one step, has been used. This architecture allows object detector to work on very high speeds. This thesis also deals with comparison of models trained on real and synthetic datasets. The best model trained on real dataset has reached 63.4 % mAP success rate and 82.3 % mAP when trained on synthetic dataset. Evaluation of one image takes about ~ 40.4 ms on average graphics processing unit and ~ 3.9 ms on higher than average graphics processing unit. The benefit of this thesis is that under certain conditions neural network model trained on synthetic data can achieve same or even better results than model trained on real data. This may simplify process of object detector development since it is not necessary to annotate large number of images.

Klíčová slova

konvoluční neuronová síť, YOLO, detekce, klasifikace, syntetická, reálná, datová sada, dopravní značka

Keywords

convolutional neural network, YOLO, detection, classification, synthetic, real, dataset, traffic sign

Citace

KOČICA, Filip. *Detekce dopravních značek v obraze a videu.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Detekce dopravních značek v obraze a videu

Prohlášení

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

.....
Filip Kočica
15. 5. 2019

Poděkování

Zde bych rád poděkoval svému vedoucímu práce prof. Ing. Adamu Heroutovi, Ph.D za cenné rady, podnětné připomínky a vstřícnost při konzultacích. Dále děkuji za přístup k výpočetním a úložným zařízením ve vlastnictví stran a projektů, přispívajících k Národní Infrastruktuře Metacentrum v rámci programu „Projects of Large Research, Development, and Innovations Infrastructures (CESNET LM2015042)“.

Obsah

Kapitola 1

Úvod

Asistenční systémy pro řidiče a autonomní vozidla se v posledních letech dočkaly velké pozornosti. Stojí zejména na správné interpretaci okolního prostředí, kam mj. spadá problematika detekce dopravního značení. Ta má stále prostor pro zlepšení. S nárůstem výpočetního výkonu velmi vzrostlo použití hlubokých neuronových sítí pro různé úlohy, zejména pak pro zpracování obrazu, kde se konvoluční sítě staly standardem pro velkou část úloh. Proto je vhodné pokusit se řešit problém detekce značek pomocí této metody.

Nevýhodou drtivé většiny metod používajících konvoluční sítě k detekci objektů a zároveň dosahujících dobrých výsledků jsou vysoké nároky na výpočetní výkon (což znesnadňuje jejich použití v přenosných či vestavěných zařízeních) a dlouhá doba zpracování jednoho snímku – oba zmíněné systémy však musí dokázat pracovat v reálném čase v kombinaci s velmi dobrou úspěšností detekce. I když se detekce značek může zdát jako jednoduchá, existuje velké množství faktorů, které tuto úlohu znesnadňují (různé světelné podmínky, částečné překrytí, malá velikost, nízká kvalita, atd.). Proto je potřeba dostatečně velká datová sada, obsahující co největší množství i těchto negativních faktorů. Proces anotace velkého počtu snímků je ovšem velmi náročný na čas (mj. může vzniknout chybná anotace). Ideálním řešením by tedy byl úspěšný systém, který by dokázal pracovat v reálném čase na zařízení s omezeným výpočetním výkonem a byl by schopen se naučit dostatek informací i ze syntetických dat.

Za účelem dosažení zpracování v reálném čase na běžně dostupných grafických čipech se zachováním dobré úspěšnosti byl pro práci vybrán systém Tiny YOLO třetí verze v kombinaci s neuronovou sítí Darknet. Generování datových sad pro trénování neuronových sítí je aktuálně velmi diskutovaným tématem a lze využít více způsobů v závislosti na typu generovaných dat. Z důvodu, že je systém YOLO trénovaný na plných snímcích a dopravní značky nabývají poměrně jednoduchých tvarů, byl využit způsob umisťování dopravních značek do plných snímků z prostředí městské zástavby.

Cílem tohoto dokumentu je jednak podat čtenáři krátký přehled o možnostech řešení problematiky detekce dopravního značení, dále pak popsat způsob řešení dané problematiky v rámci této práce a nakonec vyhodnotit dosažené výsledky. Struktura dokumentu je tedy následující. Kapitola 2 se zabývá popisem dopravního značení, metod detekce značek používaných v minulosti, metrik používaných k vyhodnocení úspěšnosti detekce objektů a nakonec porovnáním existujících řešení. Kapitola 3 je zaměřena na popis moderních technik detekce objektů založených na konvolučních neuronových sítích. V kapitole 4 je popsán návrh řešení, což zahrnuje výběr metody, datových sad a návrh generátoru syntetických dat. Implementací a testováním zvolené metody se zabývá kapitola 5 a nakonec dosažené výsledky a jejich zhodnocení lze najít v kapitole 6.

Kapitola 2

Detekce dopravního značení

Tato kapitola pojednává o obecném popisu dopravního značení, jehož automatizované rozpoznání pomocí počítače je cílem této práce. Dále jsou v této kapitole popsány běžně používané metody pro detekci a klasifikaci dopravního značení v současné době, a také metriky používané pro vyhodnocení úspěšnosti v oboru detekce objektů. Nakonec jsou zmíněny zajímavé existující práce z této oblasti, jimi použité metody a dosažené výsledky.

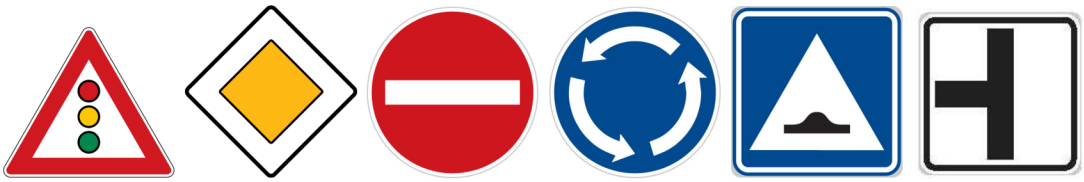
2.1 Dopravní značení

Dopravní značení slouží k účelu řízení a regulace dopravy na pozemních komunikacích a tedy k zajištění plynulosti a bezpečnosti dopravy. Svislé dopravní značení, kterému se tato práce věnuje, se běžně nachází na stranách silnice či na konstrukci nad silnicí [40, 38].

Dělení dopravního značení v Česku

Dělení svislého dopravního značení, které je nejčastějším způsobem dopravního značení, se skládá ze šesti tříd vizualizovaných na obrázku 2.1 a jejich krátký popis podle [39, 40, 38] je následovný:

- **Výstražné dopravní značky** upozorňují účastníky silničního provozu na možná nebezpečí a rizika. Jsou to zpravidla trojúhelníky s bílým pozadím, piktogramem uprostřed a ohraničené červenou barvou.
- **Značky upravující přednost** se používají na křižovatkách pro určení přednosti. Tyto značky mohou být trojúhelníkové, čtvercové i osmiúhelníkové a barevné provedení je buďto žluté, červené či modré.
- **Zákazové dopravní značky** ukládají účastníkům silničního provozu zákazy jistých činností.
- **Příkazové dopravní značky** slouží k odstranění možnosti vzniku nebezpečných situací. Tyto značky jsou vždy kruhového tvaru s modrým barevným provedením a bílým piktogramem či textem uprostřed.
- **Informativní dopravní značky** poskytují pro účastníky silničního provozu důležité informace, určující například orientaci, typ zóny či silnice.
- **Dodatkové tabulky** slouží k doplnění či omezení významu jiných dopravních značek.



Obrázek 2.1: Příklad značek jednotlivých tříd.[1]

Vzhled dopravních značek

Evropská norma definuje základní vlastnosti značek, ale ponechává státům jistou volnost. Značky v jednotlivých státech se mohou trochu lišit, ať už barvou, velikostí či tvarem. Pro člověka je tento rozdíl téměř nevnímatelný, ale počítači může při detekci působit potíže. Na obrázku 2.2 lze vidět variabilitu barev a způsobu zpracování značek *stop* pěti Evropských států (Rakouska, Česka, Řecka, Rumunska a Turecka) [40].

Obrázek 2.2: Rozdílnost značek stop některých Evropských zemí.[2]

Ačkoli se detekce dopravních značek může zdát jako jednoduchá úloha, tak stále není úplně vyřešena a stále přibývají kvalitní práce posouvající tuto problematiku kupředu. Velký problém při detekci značek sehrávají různé světelné podmínky, přírodní vlivy (např. vyblednutí barev, sníh, déšť), vandalismus, deformace či jiné poškození značek, částečné překrytí, malá velikost, apod., jak lze vidět na obrázku 2.3.

Obrázek 2.3: Problémové dopravní značky pocházející z použité datové sady.

[1] Převzato z [39].
[2] Převzato z https://en.wikipedia.org/wiki/Comparison_of_European_road_signs

2.2 Způsoby detekce dopravního značení

Při detekci těchto objektů lze vycházet z jejich specifického tvaru, barvy a také jejich dobré rozlišitelnosti od pozadí [2]. Vzhledem k množství již existujících prací na toto téma se nabízí celá řada způsobů, jak k řešení tohoto problému přistupovat, proto zde bude zmíněno jen několik nejdůležitějších. Problém lze řešit také pomocí konvolučních neuronových sítí, popsaných samostatně v kapitole 3. Na rozdíl od klasifikace objektů, jejich detekce vyžaduje lokalizaci (běžně více) objektů v rámci celého snímku [13].

Segmentace na základě barvy

Základní segmentace obrazu je běžně dosažena pomocí prahování na základě určité barvy. Podle práce [10] je intuitivní použití barevného modelu RGB (red, green, blue), u kterého je každý pixel definován pomocí třech hodnot: červené, zelené a modré. Prahování pomocí tohoto barevného modelu lze vyjádřit pomocí rovnic (2.1) a (2.2).

$$g(x,y) = k_1 \begin{cases} R_a \leq f_r(x,y) \leq R_b \\ G_a \leq f_g(x,y) \leq G_b \\ B_a \leq f_b(x,y) \leq B_b \end{cases} \tag{2.1}$$

$$g(x,y) = k_2 \text{ v jakémkoli jiném případě} \tag{2.2}$$

Funkce $f_*(x,y)$ udávají hodnotu jednotlivých barevných složek daného bodu obrázku. Výsledný pixel nabývá hodnoty k_1 pouze v případě, že všechny barevné kanály pixelu spadají do předem definovaných intervalů, jinak nabývá hodnoty k_2. Velkou nevýhodou tohoto barevného modelu ovšem je, že je velmi citlivý na změnu osvětlení. Z tohoto důvodu se již v 75. letech minulého století zavedly nové barevné modely HS* (HSI, HSV, HSL), které jsou mnohem bližší lidskému vnímání barev [2]. První složka je *hue*, tedy odstín. Druhá je *saturation*, která udává sytost a poslední např. *intensity* (I), udávající intenzitu barvy. Převod z barevného prostoru RGB, pomocí něhož jsou obrázky běžně reprezentovány, do jednoho ze zmíněných modelů byl v minulosti výpočetně dosti náročný, ale s dnešní výpočetní kapacitou to není problém, a proto se často využívá. Převod z barevného prostoru RGB do HSV podle [33] je následující:

$$V = max(R,G,B) \tag{2.3}$$

$$S = \begin{cases} \frac{V - min(R,G,B)}{V} & V \neq 0 \\ 0 & V = 0 \end{cases} \tag{2.4}$$

$$H = \begin{cases} \frac{60(G-B)}{V-min(R,G,B)} & V = R \\ \frac{120+60(B-R)}{V-min(R,G,B)} & V = G \\ \frac{240+60(R-G)}{V-min(R,G,B)} & V = B \end{cases} \tag{2.5}$$

Z tabulky porovnání detekčních metod v [2] vyplývá, že je používaný jak barevný model RGB, použit 5×, tak barevný model HS*, použit také 5×. Mimo ně byl v porovnání jedenkrát uveden model CIECAM97. Příklad použití prahování na základě červené barvy při použití modelu RGB lze vidět na obrázku 2.4.

Obrázek 2.4: Prahování na základě červené barvy pomocí barevného modelu RGB. Upraveno pomocí `OpenCV`.

Cannyho hranový detektor

Tento detektor byl vytvořen v roce 1986 Johnem Cannym, ale ještě dnes se v oblasti zpracování obrazu hojně využívá [3]. Canny definoval základní pravidla, která musí detektor splňovat, a ty jsou následovné:

1. **Vysoká úroveň detekce**. Detektor by měl mít nízkou pravděpodobnost selhání detekce existující hrany a zároveň nízkou pravděpodobnost detekce neexistující hrany.
2. **Dobrá lokalizace**. Body označené jako hraniční by měly být co nejblíže středu reálné hrany.
3. **Pouze jedna detekce jedné hrany**. Pokud je jedna hrana detekována vícekrát, pouze jedna z těchto detekcí by měla být označena jako správná a ostatní ignorovány.

Obrázek 2.5: Detekce hran pomocí Cannyho hranového detektoru. Upraveno pomocí `OpenCV`.

Algoritmus je koncipován pouze pro práci se signálem obsahující jeden kanál, proto nelze metodu přímo použít například na RGB obrázek, který má tři kanály. První je potřeba převést obrázek do stupňů šedi. Výsledek detekce hran dopravní značky pomocí Cannyho detektoru lze vidět na obrázku 2.5. Obecný popis zpracování obrazového signálu Cannyho detektorem je následovný [3, 33, 9].

1. Redukce šumu – provedení konvoluce obrazu f s gaussovým filtrem G, tedy $f * G$.

2. Nalezení velikosti a směru gradientu. n by měla být orientovaná normála ke směru detekované hrany a i přesto, že tento směr není znám, je možné udělat jeho poměrně přesný odhad na základě směru vyhlazeného gradientu:

$$n = \frac{\nabla(G * f)}{|\nabla(G * f)|} \tag{2.6}$$

3. Nalezení správné pozice hrany pomocí operace zvané *nonmaximum suppression*, tedy potlačení ne-maximálních hodnot. Ta slouží k odstranění hodnot, které nejsou ve směru gradientu lokální maxima.

4. Provést v obraze prahování s hysterezí pro odstranění bezvýznamných hran. Existují dva zadané prahy T_1 (minimální) a T_2 (maximální). Pokud je odezva v bodě vyšší než T_2, je bod označen za součást hrany. Pokud se odezva bodu nachází mezi T_1 a T_2 a zároveň je spojena s bodem, který již byl označen jako součást hrany, je tento bod také označen za součást hrany. Všechny ostatní body jsou označeny tak, že nejsou považovány za součást žádné hrany.

Velkou výhodou Cannyho detektoru hran je, že není citlivý na šum a dokáže tedy pracovat se značně zašuměným obrazem [42].

Detekce geometrických tvarů v obraze pomocí Houghovy transformace

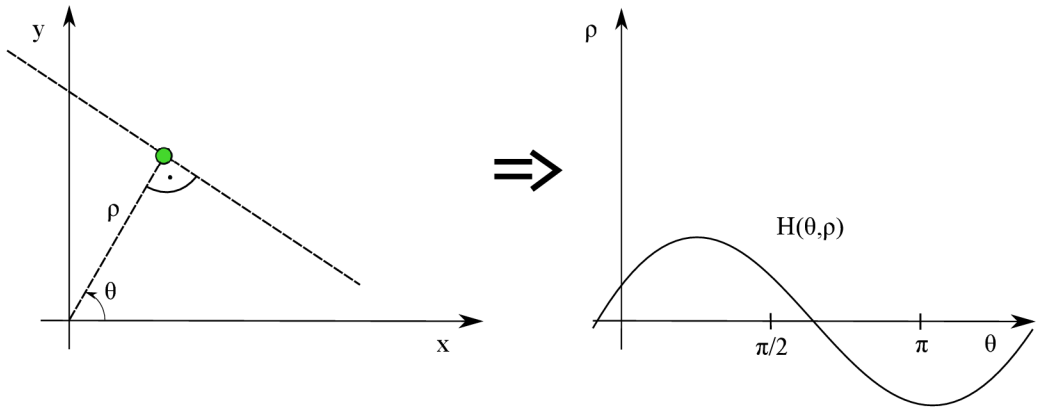
Houghova transformace je výpočetně poměrně náročná operace, a proto se na obraz běžně první aplikuje hranový detektor a až poté se provádí Houghova transformace. Detekce hran vede k poměrně značné redukci informace a šumu v obrazu a zachová informaci potřebnou pro nalezení objektu. Proto podstatně zrychluje celý proces a zpřesňuje výsledek. Pomocí Houghovy transformace se hledají tzv. ROI (*Regions of Interest*, tedy oblasti zájmu), ve kterých se může dopravní značka nacházet. Metoda byla nesčetněkrát modifikována a rozšířena [8, 34].

Klasická Houghova transformace

Houghova transformace pro detekci přímek vychází z rovnice přímky (2.7). Podle [8, 9] libovolná přímka může být reprezentována pomocí jediného bodu v prostoru. Parametr θ udává úhel, který svírá normála přímky s kladnou částí osy x. Parametr ρ udává vzdálenost přímky od středu souřadného systému.

$$x \cos\theta + y \sin\theta = \rho \tag{2.7}$$

Tyto parametry je pro detekci přímky potřeba nalézt a vytváří se pro tyto účely dvoudimenzionální Houghův prostor složený z parametrů θ a ρ. Postupně se prochází celý prohledávaný prostor a pro každý bod se určí všechny možné přímky procházející tímto bodem a umístí se do zmíněného Houghova prostoru. Pro daný bod tedy vznikne sinusoida, viz obrázek 2.6. Sinusoidy všech bodů, které leží na stejné přímce, se protnou ve stejném bodě.



Obrázek 2.6: Bod obrázku převedený na sinusoidu v Houghově prostoru.

Zobecněná Houghova transformace

Stejně jako metoda pro detekci přímek vycházela z rovnice přímky (2.7), vychází metoda pro detekci kružnic z rovnice kružnice (2.8), kde a a b udávají střed kružnice a parametr c udává poloměr kružnice.

$$(x-a)^2 + (y-b)^2 = c \tag{2.8}$$

Postupuje se stejně jako u detekce přímek, pouze jsou zde hledány tři parametry, a proto Houghův prostor bude mít v tomto případě tři dimenze.

Histogram orientovaných gradientů

Při studiu této problematiky jsem vycházel z práce [5]. Základní myšlenka je taková, že vzhled a tvar objektu může být reprezentován pomocí intenzity gradientů bez nutnosti znát jeho přesnou polohu. Metoda funguje na principu rozdělení vstupního obrázku na menší části pevně dané velikosti zvané *buňky* a získávání normalizovaných lokálních histogramů gradientů těchto buněk. Každá buňka produkuje jednorozměrný histogram směru gradientů či orientace hran na základě pixelů nacházejících se v buňce (každý pixel produkuje *hlas*, které jsou akumulovány do lokálních kontejnerů – zvýšení počtu těchto kontejnerů značně zlepšuje úspěšnost). Velikost gradientu závisí na lokálních změnách osvětlení, a proto pro lepší odolnost vůči změně osvětlení je dobré provést normalizaci kontrastu.

Normalizace kontrastu lze dosáhnout pomocí získání celkového množství „energie“ histogramu na větším prostoru zvaném *blok* a pomocí výsledku normalizovat veškeré buňky, které se v daném bloku nachází. Normalizace kontrastu se ukázala jako klíčová pro dosažení dobré úspěšnosti. Vektor těchto histogramů získaných z jednoho bloku je nazývaný deskriptor. V případě barevných obrázků jsou počítány gradienty pro každý kanál odděleně a výsledným vektorem gradientu daného bodu se stává ten s nejvyšší normalizovanou hodnotou.

2.3 Způsoby klasifikace dopravního značení

Po nalezení kandidátních oblastí (tzv. *Regions of Interest*), ve kterých se mohou nacházet dopravní značky, je třeba rozhodnout, zda se v tomto místě značka opravdu nachází, a také značku klasifikovat do správné třídy. K tomu lze využít následující metody.

Support Vector Machines – SVM

Support Vector Machines (dále jen zkratka SVM) vytváří jednu či více hyper-rovin ve vícerozměrném prostoru použitých pro klasifikaci. Při studiu této metody jsem vycházel ze [6, 25, 34]. SVM je v základní verzi binární klasifikátor, který dokáže klasifikovat data pouze dvou tříd, ale je možné jej použít také k regresi a a-třídní klasifikaci. Trénovací data se skládají z konečné sady N bodů x_i, kde každý spadá do jedné ze dvou tříd, které jsou reprezentovány pomocí y_i a nabývají hodnot 1 nebo -1. Taková datová sada potom může vypadat následovně:

$$x = \{(x_1, y_1), ..., (x_N, y_N)\} \tag{2.9}$$

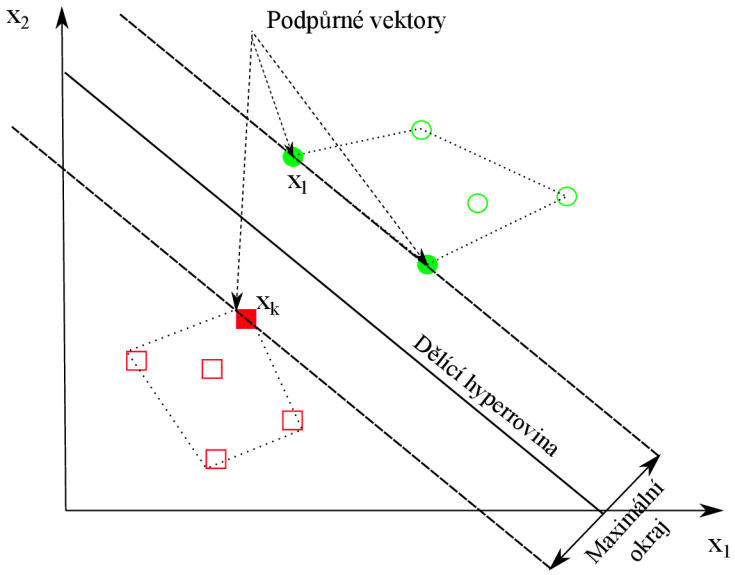
Pro lineárně odlišitelnou datovou sadu existují parametry w a b (kde w je vektor vah a b je práh reprezentující hyperrovinu) pro každý vstupní bod:

$$y_i(w \cdot x_i + b) > 0 \tag{2.10}$$

Takovýto výraz existuje pro každý z bodů vstupní datové sady a tato sada výrazů poté definuje omezení na volbu parametrů w a b. Tato omezení vyjadřují, že všechny vstupní prvky se záporným y_i by měly být na jedné straně hyper-roviny, zatímco prvky s kladným y_i na straně druhé. Při změně měřítek obou parametrů w a b nedochází k porušení podmínky vyjádřené pomocí (2.10), ale je možné zvolit parametry tak, aby platilo:

$$y_i(w \cdot x_i + b) \geq 1 \tag{2.11}$$

Nejběžnější přístup k vytvoření dělící hyper-roviny je takový, aby byla co nejdále od konvexních obalů shluků obou tříd. Dělící hyper-rovina je normála, která půlí přímku spojující dva nejbližší body jednotlivých konvexních obálek daných tříd, čímž dochází k maximalizaci minimální vzdálenosti obou shluků od dělící linie. Podpůrné vektory jsou potom body, které mají v rámci třídy co nejkratší vzdálenost k dělící hyper-rovině, jak ukazuje obrázek 2.7. Obecně se dá říct, že čím větší je tento okraj kolem hyper-roviny (angl. *margin*), tím nižší je chybovost.



Obrázek 2.7: Grafické znázornění oddělení vzorků dvou tříd pomocí hyperroviny. Vyplněné body, které tuto hyperrovinu definují, se nazývají podpůrné vektory.

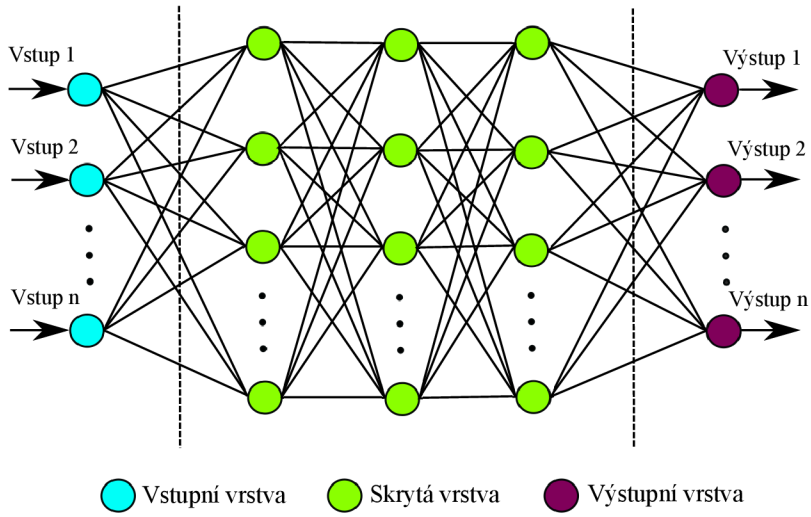
Výhody	Nevýhody
Unikátní řešení	Vysoká složitost algoritmu a s tím spojená náročnost na pochopení
Nemá problém tzv. *over-fitting* (česky přetrénování)	Poměrně pomalé trénování
Velmi účinné	Potíže s nalezením optimálních parametrů při použití nelineárně odlišitelných dat
Poměrně nízká výpočetní složitost	
Poskytuje flexibilitu v možnosti výběru formy prahu	

Tabulka 2.1: Výhody a nevýhody metody SVM [7, 25].

Umělé neuronové sítě

Umělá neuronová síť je matematický model inspirovaný biologickými neuronovými sítěmi. Umělá neuronová síť se skládá se ze sekvence vrstev, kde každá vrstva se skládá z propojené skupiny umělých neuronů a zpracovává informace za použití přístupu založeného na těchto vazbách. Všechny neurony každé vrstvy jsou váhově spojeny s neurony předchozí a následující vrstvy, jak lze vidět na obrázku 2.8. Umělé neuronové sítě slouží pro vytváření složitých vztahů mezi vstupními a výstupními daty. Úspěšnost těchto sítí závisí zejména na jejich struktuře a počtu vstupů [36, 25, 6].

Důležitá podmnožina umělých neuronových sítí pro obor zpracování obrazu jsou konvoluční neuronové sítě popsané samostatně v kapitole 3.



Obrázek 2.8: Architektura umělé neuronové sítě se třemi skrytými vrstvami. Všechny neurony jsou váhově spojeny se všemi neurony předchozí a následující vrstvy.

Učení těchto sítí je možné rozdělit do dvou hlavních tříd:

- **Metoda učení s učitelem** Funguje na principu anotovaných (oštítkovaných) vstupních dat. Anotace reprezentují správný výsledek pro daný vstup. Pro každý vzorek tedy existuje vektor vstupních dat a jeden či více očekávaných výsledků. Po provedení

klasifikace je výsledek porovnán s očekávaným výsledkem. Cílem této metody je snížit chybu klasifikace (pomocí ztrátové funkce).

- **Metoda učení bez učitele** se liší oproti metodě s učitelem v tom, že vstupní data nejsou vůbec anotovaná, tedy nejsou známy správné výstupy, pomocí kterých by se proces trénování řídil. Tyto sítě pracují na principu třídění vstupních dat na základě jejich podobnosti, bez možnosti ověření správnosti.

Většina úloh provádějící rozpoznání vzoru v oboru zpracování obrazu je řešena pomocí první zmíněné metody učení s učitelem [24].

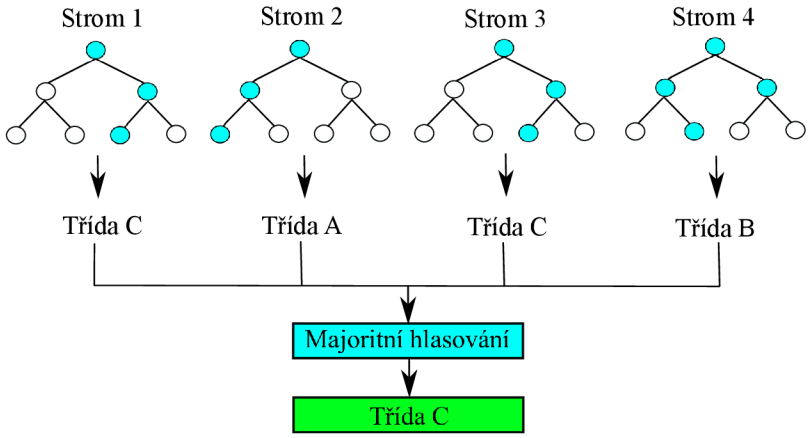
Výhody	Nevýhody
Velmi účinné pro velké datové sady	Vysoká náročnost na výpočetní zdroje a čas
Odolné vůči šumu v trénovací datové sadě	Problém tzv. *over-fitting* (česky přetrénování)
Bez-parametrický přístup. Tzn. nedělá předpoklady o mapovací funkci	Složitost výběru/nalezení správné architektury
Vysoká výpočetní rychlost	

Tabulka 2.2: Výhody a nevýhody umělých neuronových sítí [7, 25].

Rozhodovací stromy

Rozhodovací strom [7, 25] vytváří stromovou strukturu jednotlivých rozhodnutí. Rozhodnutí o členství v třídě je provedeno pomocí opakovaného rozdělování vstupní datové sady do jednotlivých podmnožin. Je to metoda učení s učitelem a oproti umělým neuronovým sítím je bezparametrická. Každá větev graficky znázorňuje rozhodnutí, které by mělo být vykonáno. Metoda umožňuje rozhodnutí o třídě (akceptaci či odmítnutí) v libovolném mezikroku a po provedení klasifikace při trénování rozhodovací stromy udávají sadu pravidel, která by měla být modelem zapamatována. Metoda se skládá ze třech částí. První je rozdělení jednotlivých uzlů, poté následuje nalezení listových uzlů a nakonec přidělení tříd jednotlivým listovým uzlům.

Metodu rozhodovacích stromů lze ještě dále rozšířit pomocí metody zvané *Random forest* (náhodný les), která během procesu trénování vytváří množinu několika rozhodovacích stromů, kde každý udává (hlasuje pro) třídu vstupního vzorku.



Obrázek 2.9: Náhodný les složený ze čtyř rozhodovacích stromů.

Výsledná třída je určena na základě výsledků jednotlivých rozhodovacích stromů a to tak, že je vybrána třída s nejvyšším počtem hlasů [36], znázorněno na obrázku 2.9.

Výhody	Nevýhody
Jednoduché na pochopení	Vysoká míra chybovosti klasifikace
Bez-parametrický přístup	Rozdělení je velmi citlivé vůči trénovací datové sadě
Nízká výpočetní náročnost	

Tabulka 2.3: Výhody a nevýhody rozhodovacích stromů [7, 25].

2.4 Vyhodnocení úspěšnosti detekce

Ve chvíli, kdy už existuje funkční model pro detekci objektů, je dobré vyhodnotit jeho úspěšnost. Úspěšnost detekce ve většině případů nestačí vyhodnotit pouhým pozorováním detekovaných obrázků (tedy kvalitativně), ale vyjadřuje se kvantitativně – proto je potřeba si zavést pojmy, které usnadní validaci detekcí a určení celkové kvality detektoru. Při studiu této problematiky jsem vycházel z [17].

Intersection over union – IoU

Intersection over union (dále jen mezi odborníky běžně používaná zkratka IoU), se počítá na základě dvou ohraničujících boxů – *ground-truth* (požadovaná výstupní hodnota, dále bude používán jen anglický tvar) – tedy pozice kde se objekt nachází a predikce vytvořené detektorem. IoU udává překrytí predikce a ground-truth normalizované na hodnotu náležící do intervalu $\langle 0, 1 \rangle$. Vizualizováno na obrázku 2.10.

Typy predikcí

Predikce detektoru jsou rozděleny do čtyř následujících tříd, dále budou požívány pouze jejich zkratky.

1. **True positive** – Udává správně detekovaný objekt (IoU je větší než hodnota prahu).

2. **False positive** – Udává špatně detekovaný objekt (IoU je menší než hodnota prahu).
3. **False negative** – Udává nedetekovaný objekt.
4. **True negative** – Místa kde není objekt jsou správně označena jako bez-objektová (používá se výjimečně).

Precision

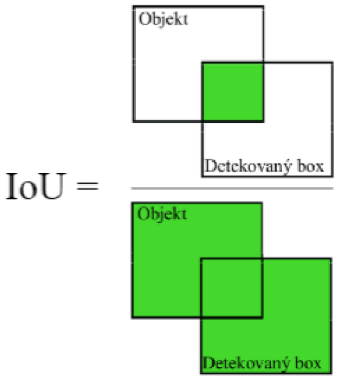
Precision (přesnost, nadále bude využívána jen anglická forma, protože to je zvykem i mezi odborníky v oboru), udává, jak přesné predikce jsou. Běžně se uvádí v procentech a vyjadřuje kolik z predikcí je pravdivých:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.12}$$

Recall

Recall (citlivost, nadále bude využívána jen anglická forma, protože to je zvykem i mezi odborníky v oboru), udává, kolik ze všech pozitivních případů bylo správně predikováno:

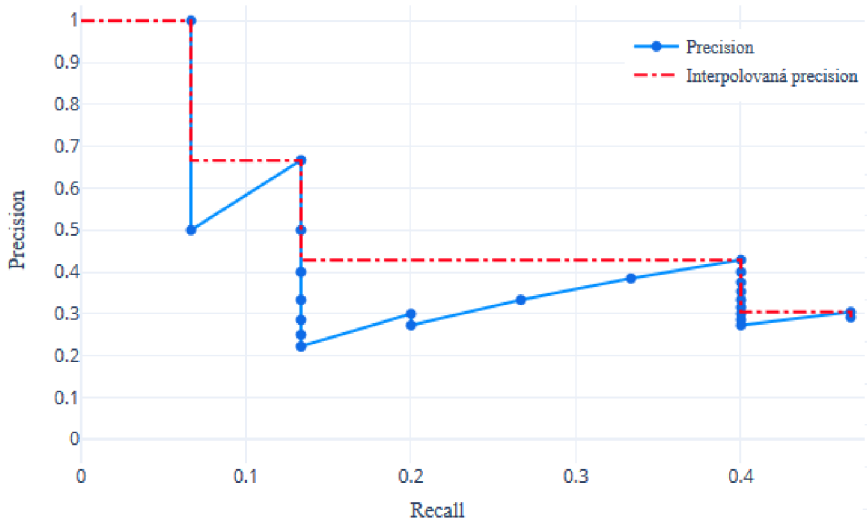
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.13}$$



Obrázek 2.10: Grafické znázornění IoU pomocí predikovaného a *ground-truth* boxu.

Average precision – AP

Average precision (dále jen mezi odborníky běžně používaná zkratka AP), je metrika používaná pro měření přesnosti detektoru objektů. Udává průměr maximálních hodnotu precision pro různé hodnoty recall. Princip AP je konceptuálně stejný jako počítání plochy pod precision-recall křivkou (AuC – *area under curve*). První je ale potřeba tzv. vyhladit křivku. Toho se dosáhne interpolací. Vytvoří se graf s hodnotou recall $0.0, 0.1, ...1.0$ a každá hodnota precision se nahradí maximální hodnotou precision pro recall $\geq \tilde{r}$. Graficky znázorněno na obrázku 2.11.



Obrázek 2.11: Precission-recall křivka a její interpolace.

AP je počítáno jako průměr maximálních hodnot precision na jedenácti hodnotách recall $(0.0, 0.1, ...1.0)$. Záměrem interpolace křivky na jedenácti bodech je snížit dopad „zákmitů“ křivky, způsobených malými odchylkami hodnocení vzorků. Výpočet je následující:

$$\text{AP} = \frac{1}{11} \sum_{r \in \{0.0,...,1.0\}} AP_r \tag{2.14}$$

$$= \frac{1}{11} \sum_{r \in \{0.0,...,1.0\}} p_i(r) \tag{2.15}$$

kde

$$p_i(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{2.16}$$

Hodnota p_i značí interpolovanou precision. AP se počítá pro každou třídu zvlášť, proto se u detektorů běžně používá mAP (*mean average precision*), což je průměr AP všech tříd.

2.5 Související práce

Detekce a klasifikace dopravních značek je již mnoho let řešená problematika, které spadá do dynamicky rozvíjejícího se oboru detekce objektů v obraze. Existuje velké množství přístupů k řešení této problematiky, ale i přesto detekce značek stále není považována za úplně vyřešenou a vznikají nové práce, jako je například tato, provádějící detekci za pomoci moderních technik ve snaze posunout tuto problematiku vpřed.

Porovnání výsledků jednotlivých prací není jednoduché, protože se často liší použitými metrikami pro vyhodnocení (hit rate, mAP (interpolovaná), mAP@0.5, mAP@0.75, ROC, F1, atd.). Dalším faktorem při vyhodnocení, který má poměrně velký vliv na celkovou úspěšnost, je použitá datová sada – datové sady se liší ať už konvencí značek, tak zejména jejich kvalitou a kvantitou. Jejich přehled je možné najít v sekci 4.1.

Jak plyne z porovnání detekčních metod dopravních značek [2], dříve byla tato úloha řešena pomocí segmentace obrazu na základě barvy v kombinaci s detekcí geometrických

útvarů či rohů [10]. V práci [22] se autoři zabývali rychlou metodou detekce tvarů založenou na Houghově transformaci přímo pro detekci dopravních značek. Výsledkem byla 95 % úspěšnost detekce a metoda je navíc invariantní vůči natočení značky a pracuje v reálném čase.

Metoda hojně využívaná pro detekci značek je extrakce příznaků z obrazu v kombinaci s klasifikátorem. Před příchodem konvolučních neuronových sítí byl přístup s použitím histogramu orientovaných gradientů (dále jen HOG) v kombinaci s SVM (*Support Vector Machines*) označován za tzv. *state of the art*[3] detekce objektů [4]. V práci [37] využívající zmíněnou kombinaci HOG a SVM se podařilo získat na Německé datové sadě výsledky detekce značek s hodnotami *precision* i *recall* blížícími se k jedné, ale vyhodnocení jednoho snímku trvalo déle než vteřinu. Další možností je použití metody využité v práci [1] – kombinaci Haarových příznaků s lineárním SVM pro klasifikaci (a konvolučními neuronovými sítěmi pro verifikaci), pomocí které se podařilo na Německé datové sadě dosáhnout taktéž výsledků s hodnotami *precision* i *recall* blížícím se k jedné a zároveň zpracování v reálném čase.

V posledních letech přitáhlo velkou pozornost hluboké učení a standardem pro detekci objektů se tak staly konvoluční sítě. Práce [13] prokázala, že jejich použití pro detekci objektů může vést ke dramatickému zvýšení úspěšnosti detekce. Výborných výsledků detekce objektů dosáhla metoda R-CNN, která kombinuje návrh kandidátních oblastí (selektivní hledání [35]) s konvolučními sítěmi a řadí se tak do třídy dvou-krokových metod. Upravená verze této metody, Faster R-CNN, byla použita v práci [41] pro detekci značek a dosáhla úspěšnosti 34.4 % mAP na Belgické datové sadě.

Pro rychlejší, ale stále přesnou detekci vznikla třída tzv. *single-shot* detektorů, kam spadá YOLO a SSD (*Single Shot Detector*). Ty provádí detekci i klasifikaci pomocí jedné konvoluční sítě. Upravený systém Tiny YOLOv1 byl pro detekci značek použit v práci [4], kde se autorům podařilo dosáhnout úspěšnosti 33.8 % mAP na Belgické datové sadě dopravních značek s rychlostí 100 snímků za sekundu. SSD byl použit v práci [20], kde se autoři snažili o přesný odhad okrajů dopravních značek a dosáhli úspěšnosti kolem 85 % mAP na Německé datové sadě při rychlosti 7 snímků za sekundu na zařízení s poměrně nízkým výpočetním výkonem.

Z dosažených výsledků zmíněných prací vyplývá, že pro úlohu detekce dopravních značek stále dosahují lepší úspěšnosti systémy využívající extrakci příznaků v kombinaci s klasifikátorem, ale konvoluční sítě je pomalu dohánějí jak z hlediska úspěšnosti, tak rychlosti.

[3] *State of the art* – stav techniky (tzn. aktuálně nejlepší způsob pro řešení dané problematiky).

Kapitola 3

Detekce pomocí konvolučních neuronových sítí

Konvoluční neuronové sítě existují již řadu let, ale standardem pro úlohy rozpoznání vzoru v oboru zpracování obrazu se staly až v posledních letech. Je to zapříčiněno zejména vyšším výpočetním výkonem počítačů a velkému množství obrázků a jimi tvořených datových sad, na kterých je možné tyto neuronové sítě trénovat.

3.1 Konvoluční neuronové sítě

Při studiu jsem vycházel z [24, 6]. Hlavní rozdíl mezi konvolučními a klasickým umělými neuronovými sítěmi je v tom, že konvoluční sítě jsou používány zejména pro účely rozpoznání vzorů v oboru zpracování obrazu. Největší nevýhodou umělých neuronových sítí pro výpočet obrazových dat je jejich problém s vysokou výpočetní složitostí – dokáží si poradit pouze s obrázky s jedním kanálem a malým rozlišením. Pro komplexnější obrazové vstupy by taková neuronová síť byla příliš velká pro efektivní učení.

Architektura

Konvoluční neuronové sítě se skládají ze třech typů vrstev. Konvolučních, pooling (sdružovacích) a plně propojených. První vrstva se nazývá vstupní a poslední výstupní, mezi nimi se nacházejí vrstvy skryté. Podle [24] je postup zpracování jednoho snímku následovný:

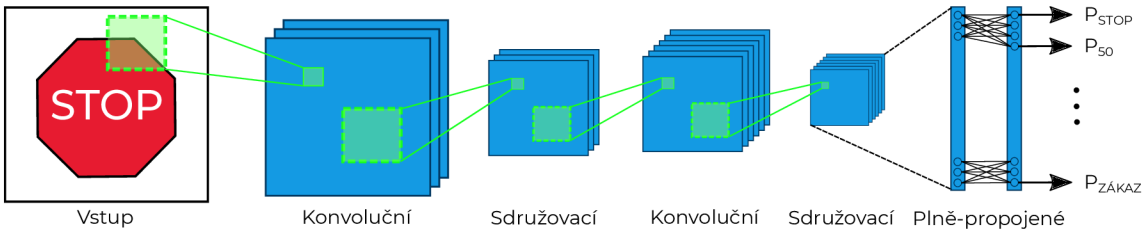
1. Hodnoty jednotlivých pixelů snímku se uloží do vstupní vrstvy
2. **Konvoluční vrstva** určí hodnoty jednotlivých neuronů. Ty jsou dány konvolucí jádra (vah) a lokální části vrstvy (zvané jako receptivní pole neuronu), se kterou je neuron spojen. Přenosová funkce umístěná za konvoluční vrstvou (např. ReLU[1]) slouží k provedení aktivací pro každý prvek z předchozí vrstvy.
3. **Sdružovací vrstva** poté provede tzv. *downsampling*, tedy podvzorkování (snížení počtu vzorků – informace), a to buď za pomoci maximalizace či průměrování okolí pixelu a výsledkem je pouze jedna hodnota. To slouží ke zmenšení velikosti dané vrstvy.

[1] ReLU – *Rectified Linear Unit* (rektifikovaná lineární jednotka).

4. **Konvoluční** a **sdružovací** vrstvy se mohou opakovat. Hluboké konvoluční neuronové sítě se zpravidla skládají z několika skrytých konvolučních vrstev.

5. Nakonec **plně-propojená vrstva** vyprodukuje skóre (pravděpodobnosti) jednotlivých tříd z aktivací, sloužící ke klasifikaci.

Proces zpracování konvoluční neuronovou sítí popsaný výše lze vidět na obrázku 3.1. Díky této jednoduché transformaci dokáže konvoluční neuronová síť za pomoci konvolučních a podvzorkovacích technik převést vstupní obrázek z matice pixelů na skóre jednotlivých tříd pro účely klasifikace či regrese.



Obrázek 3.1: Základní architektura konvoluční neuronové sítě.

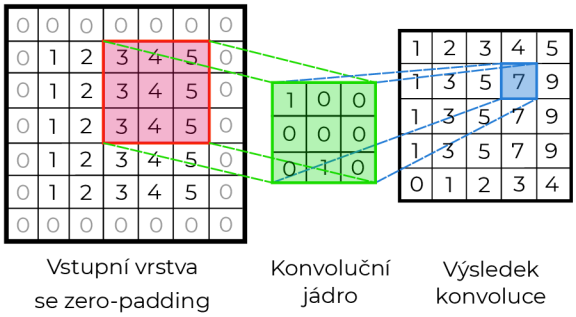
Konvoluční vrstva

Konvoluční vrstvy sehrávají ve fungování celé konvoluční neuronové sítě nezastupitelnou roli. Parametry jednotlivých vrstev jsou založeny na „učících“ se jádrech. Tato jádra jsou ve své podstatě matice, běžně o malém rozměru. Ve chvíli, kdy se data dostanou do konvoluční vrstvy, je postupně prováděna konvoluce všech jader se vstupními daty. Protože jsou vstupní data dvou-rozměrná, lze pomocí jednotlivých konvolučních jader „přejíždět“ přes vstupní data a produkovat výsledek této operace jako jednu hodnotu, jak lze vidět na obrázku 3.2. Výstupem je poté tzv. *heat map* (česky zvaná aktivační mapa), zobrazená na obrázku 3.3. Postupným učením se jádro zlepšuje a poté, když ve vstupu na jisté pozici najde určitou vlastnost, vydá podnět známý jako aktivace [24].

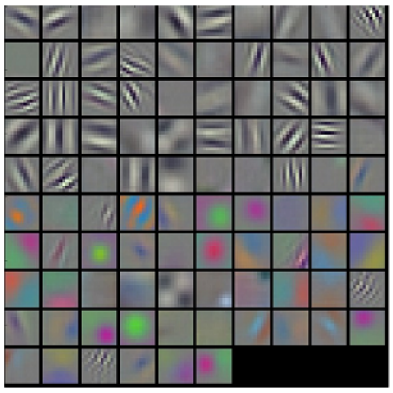
Pomocí optimalizace výstupu dokáže konvoluční vrstva také snížit celkovou složitost modelu. Tato optimalizace se provádí za pomoci třech hyper-parametrů, zvaných *depth* (hloubka), *stride* (krok), a *zero-padding* (nulové vyplňování). Hloubku lze nastavit pomocí počtu neuronů uvnitř vrstvy. Snížením tohoto hyper-parametru lze docílit podstatně nižšího počtu neuronů v síti, a tedy složitosti, ale zároveň se tím sníží schopnost rozpoznání vzorů. Krok určuje, o kolik se jádro posouvá při provádění konvolucí jader se vstupními daty. Při nastavení kroku na jedna by se receptivní pole jednotlivých neuronů příliš překrývala a výsledkem by byly příliš velké aktivace. Naopak při nastavení kroku na číslo velké by znamenalo velké mezery mezi receptivními poli neuronů a tedy příliš malé rozměry výstupu. Nulová výplň je technika vyplňování krajů vstupu a poskytuje vyšší kontrolu rozměrů výstupu. Pomocí těchto parametrů je tedy možné změnit dimenze výstupů a jejich výpočet je následující [24]:

$$D = \frac{(V - R) + 2Z}{S + 1} \tag{3.1}$$

Kde R udává velikost receptivního pole neuronu (stejné jako velikost jádra), S udává krok, V udává velikost vstupu (tedy výška $\times$ šířka $\times$ hloubka) a Z je velikost nulové výplně.



Obrázek 3.2: Konvoluce vstupu velikosti 5×5 (se *zero-padding* rovno jedné) s jádrem o velikosti 3×3.



Obrázek 3.3: Aktivační mapy pocházející z první konvoluční vrstvy.[2]

Vytváření architektury konvoluční neuronové sítě je složitý proces a existují tu jisté ověřené metody. Běžná architektura těchto sítí je založena na tom, že každá konvoluční vrstva je následována vrstvou sdružovací a to se opakuje až po plně propojenou vrstvu. Pro rozpoznání složitějších vlastností v obraze je doporučeným postupem umístit za sebe několik konvolučních vrstev. Dalším ověřeným postupem je rozdělení jedné velké konvoluční vrstvy na několik menších, což slouží pro snížení výpočetní náročnosti pro danou konvoluční vrstvu [24].

Overfitting

Overfitting, český zvané „přetrénování“ znamená, že neuronová síť se už nedokáže efektivně učit. Nastává ve chvíli, kdy neuronová síť dokáže správně predikovat pouze obrázky z trénovací datové sady, ale selhává u všech ostatních. Řešením je například snížení počtu parametrů, které je třeba se naučit. Čím méně parametrů je třeba se naučit, tím menší je pravděpodobnost výskytu přetrénování [24].

3.2 You Only Look Once – YOLO

Při studiu této problematiky jsem vycházel z [29, 30, 31, 19, 18, 4]. Systém YOLO přistupuje k detekci objektů jako k jednotné regresi, přímo od pixelů snímku k souřadnicím ohraničujících boxů a pravděpodobnostem jednotlivých tříd. Systém YOLO je poměrně jednoduchý, za pomoci jediné konvoluční neuronové sítě predikuje množinu ohraničujících boxů a zároveň pravděpodobnosti tříd. YOLO je speciální systém, který je trénovaný na tzv. „plných obrázcích“ (tedy celých snímcích, nikoli vyřezaných objektech jako klasifikátory) a při tom rovnou optimalizuje úspěšnost detekce. Tento systém má oproti běžným přístupům detekce objektů řadu výhod.

Zaprvé, YOLO přistupuje k detekci jako k regresnímu problému, což umožňuje odstranění složité *pipe-line* (posloupnost kroků). To umožňuje provádět detekce velmi rychle při zachování dobré úspěšnosti. Základní verze YOLO pracuje na rychlosti zpracování 45 snímků za sekundu a odlehčená verze dokonce na více než 150 snímcích za sekundu na grafickém čipu Titan X. YOLO také dosahuje více než dvojnásobek mAP než ostatní *real-time* detektory (detektory, které dokáží pracovat v reálném čase).

[2] Převzato z http://cs231n.github.io/understanding-cnn.

Zadruhé, YOLO na rozdíl od metod posuvného okna a návrhu oblastí (*region-proposal*) během trénování i testování „vidí“ celý snímek, což tomuto systému umožňuje se naučit kontextuální informace o třídách, stejně jako jejich vzhled. Metoda R-CNN popsaná v sekci 3.4 (jeden z nejúspěšnějších systémů pro detekci) zaměňuje velké množství částí pozadí za objekty, protože nevidí širší kontext, ve kterém se objekty nachází.

Zatřetí, YOLO se učí obecnou reprezentaci objektů. Například pokud je natrénováno na přirozených obrázcích a vyhodnoceno na umění, dosahuje mnohem lepších výsledků než ostatní detekční systémy. To znamená mnohem nižší pravděpodobnost selhání při neočekávaném či úplně jiném typu vstupu [29].

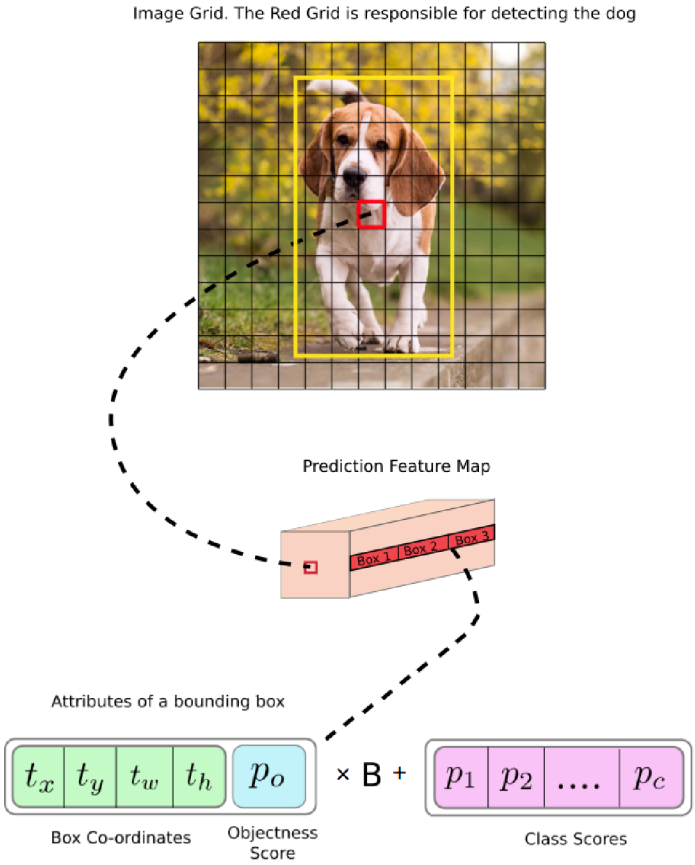
Princip fungování systému YOLO

Systém rozděluje vstupní snímek na mřížku o velikosti $S \times S$ buněk, jak lze vidět na obrázcích 3.4 a 3.5. Pokud se střed objektu nachází v buňce, pak je daná buňka zodpovědná za detekci tohoto objektu.

Každá buňka predikuje B ohraničujících boxů a zároveň také jistotu (angl. *confidence*), s jakou dané boxy predikuje. Tyto jistoty udávají, jak si je model jistý, že daný box obsahuje objekt (anglicky tzv. *objectness* – dále jen objektovost), a také jak přesný box je. Tuto jistotu lze formálně definovat jako $\Pr(\text{Object}) \times \text{IoU}^{\text{truth}}_{\text{pred}}$, kde $\Pr(\text{Object})$ udává pravděpodobnost výskytu objektu v boxu a $\text{IoU}^{\text{truth}}_{\text{pred}}$ udává IoU mezi predikovaným a *ground-truth* boxem. Pokud se v dané buňce žádný objekt nenachází, mělo by skóre objektovosti být ideálně rovno nule, v opačném případě rovno IoU mezi boxy predikce a *ground-truth*.

Každý ohraničující box je složen z pěti predikcí. Jmenovitě to jsou x, y, w, h a jistota boxu. Souřadnice x,y reprezentují střed predikovaného boxu, w a h pak výšku a šířku daného boxu. Tyto hodnoty jsou relativní (normalizovány velikostí celého obrázku). Poslední predikce, jistota boxu, udává IoU mezi predikovaným a kterýmkoli *ground-truth* boxem.

Každá buňka také predikuje C podmíněných pravděpodobností jednotlivých tříd – $\Pr(\text{Class}_i|\text{Object})$. Tyto pravděpodobnosti jsou podmíněny tím, zda se v dané buňce nachází objekt. Predikováno je takové množství sad pravděpodobností všech tříd, jaký je počet buněk, nezávisle na počtu predikovaných ohraničujících boxů.



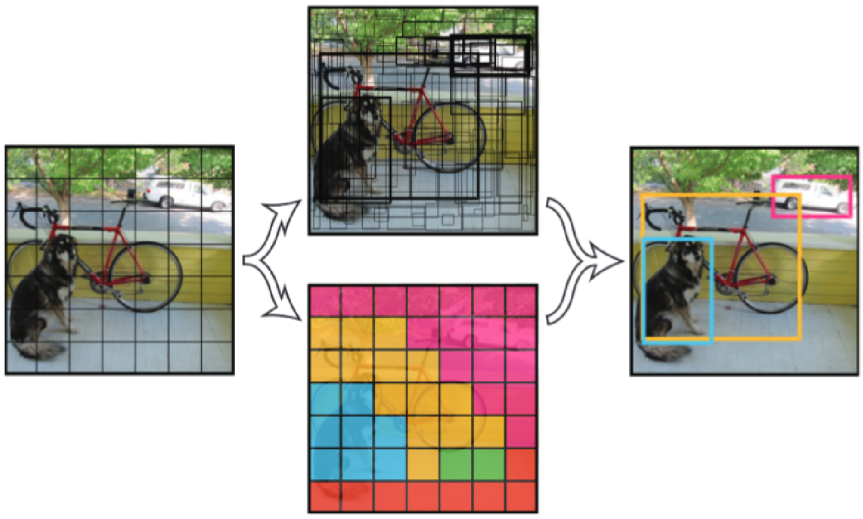
Obrázek 3.4: Princip vytváření predikcí pomocí systému YOLO. Výsledkem je tenzor o velikosti $S \times S \times (B \times 5 + C)$.[3]

Při testování se násobí podmíněné pravděpodobnosti tříd s jistotou predikce jednotlivých boxů:

$$\mathrm{PR}(\mathrm{Class}_i|\mathrm{Object}) \times \mathrm{Pr}(\mathrm{Object}) \times \mathrm{IoU}^{\mathrm{truth}}_{\mathrm{pred}} = \mathrm{PR}(\mathrm{Class}_i) \times \mathrm{IoU}^{\mathrm{truth}}_{\mathrm{pred}}, \tag{3.2}$$

což udává jistotu jednotlivých tříd pro každý ohraničující box. Tato skóre udávají jak pravděpodobnost, že se třída v daném boxu nachází, tak jak dobře ohraničuje predikovaný box daný objekt. Vizualizováno na obrázku 3.5.

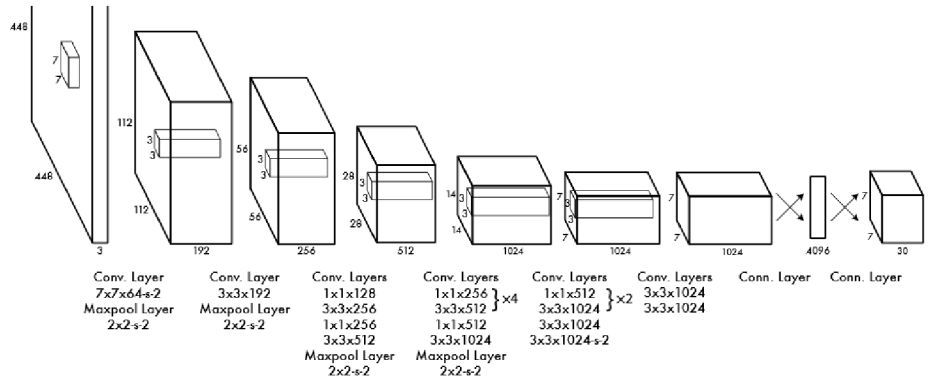
[3]Převzato z https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch a upraveno.



Obrázek 3.5: Vytvoření prediktivních ohraničujících boxů. Tloušťka hrany boxu reprezentuje jistotu, s jakou systém daný objekt predikuje – čím tlustější, tím si je systém jistější.[4]

Architektura

Architekturu základní verze systému YOLO je možné vidět na obrázku 3.6. Skládá se ze 24 konvolučních vrstev následovaných dvěma plně-propojenými vrstvami. Střídající se vrstvy o velikosti 1×1 slouží pro snížení množství vlastností z předcházejících vrstev. Počáteční vrstva slouží k extrakci vlastností ze vstupních snímků a koncové plně-propojené vrstvy slouží k predikci výstupních pravděpodobností jednotlivých tříd a souřadnic ohraničujících boxů. Výstupem je tenzor predikcí o velikosti $7 \times 7 \times 30$.



Obrázek 3.6: Architektura základní verze YOLO.[5]

Ztrátová funkce

Systém YOLO vytváří několik predikcí pro každou buňku mřížky, ale pouze jedna z nich je zodpovědná za predikci objektu, a to ta s nejvyšším IoU s *ground-truth*. Ztrátová funkce

[4] Převzato z [29].
[5] Převzato z [29].

slouží k ověření, jak dobře systém odpovídá trénovací sadě. Při trénování pomocí systému YOLO je optimalizována ztrátová funkce skládající se z několika následujících částí.

$$\text{Loss} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{3.3}$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \tag{3.4}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \tag{3.5}$$

$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{3.6}$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{3.7}$$

kde $\mathbb{1}_i^{obj} = 1$ pokud se v buňce i nachází objekt, jinak 0. Dále pak $\mathbb{1}_{ij}^{obj} = 1$ udává, že j-tý ohraničující box je zodpovědný za detekci v rámci buňky i. Systém YOLO používá tzv. *sum-squared error* mezi predikovanými a *ground-truth* boxy a třídami. Funkce se dá rozdělit na pět částí, penalizující systém za tři typy chyb.

- **Chyba klasifikace** – Prochází každou buňku mřížky a pokud se v ní nachází objekt, tato chyba je vypočítána jako *squared-error* podmíněné pravděpodobnosti třídy, a to pro všechny třídy pomocí rovnice (3.7). Pravděpodobnost výskytu třídy c v buňce i udává $\hat{p}_i(c)$.
- **Chyba lokalizace** – Tato chyba udává, jak dobře či špatně byl objekt lokalizován, tedy rozdíl souřadnic a velikosti predikovaného boxu vůči *ground-truth* pomocí rovnic (3.3) a (3.4). Ke zvýšení váhy chyby predikce souřadnic ohraničujícího boxu slouží λ_{coord}.
- **Chyba jistoty predikce** – Pokud je v boxu detekován objekt, tak se tato chyba počítá pomocí rovnice (3.5), jinak (3.6). Většina boxů neobsahuje objekt, což způsobuje nevyváženost, a proto λ_{noobj} snižuje váhu chyby při detekci pozadí.

Non-maxima suppression

Systém YOLO může vytvořit duplicitní predikce jednoho objektu, proto provádí *non-maxima suppression* (potlačení ne-maximálních hodnot) k odstranění predikcí s nižší jistotou, což může vést ke zvýšení mAP až o 3 % [18]. Běžná implementace tohoto potlačení je následovná:

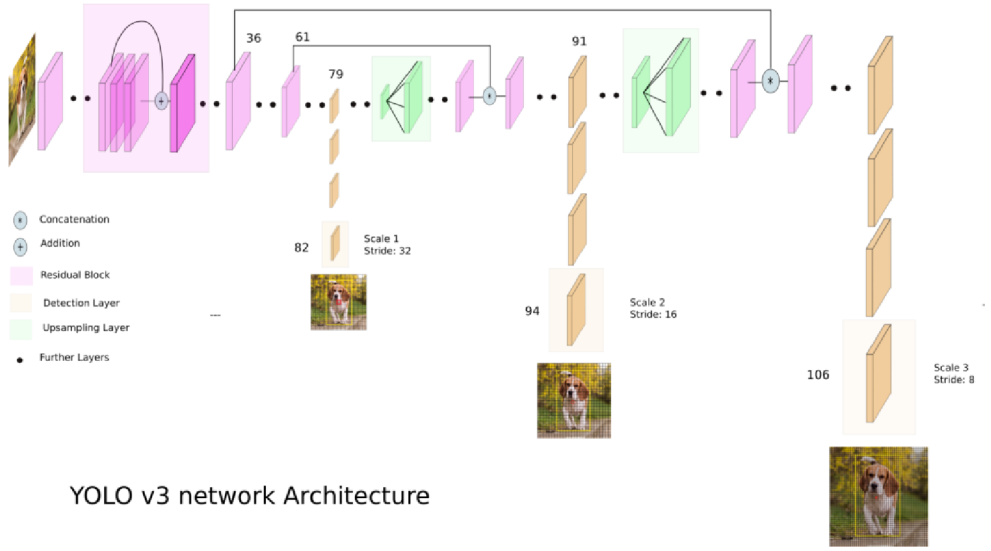
1. Seřadit predikce podle jistoty.
2. Začít procházet predikce od těch s nejvyšší jistotou a ignorovat všechny predikce, které mají stejnou třídu a IoU > 0.5 s aktuální predikcí.
3. Opakovat krok 2 dokud nejsou všechny predikce zkontrolovány.

Odlehčená verze – Tiny YOLO

Odlehčená verze, zvaná Tiny či Fast YOLO, se skládá z menšího počtu konvolučních vrstev (9 oproti 24) a také méně filtrů v těchto vrstvách než základní verze. Kromě zmíněné změny velikosti jsou všechny ostatní parametry jak při trénování, tak testování u odlehčené verze zachovány. Architektura odlehčené verze je navržena tak, aby posunula hranice rychlé detekce objektů. Jak již bylo zmíněno, dosahuje rychlosti zpracování více než 150 snímků za sekundu na GPU Titan X (oproti základní verzi, která zvládne zpracovat 45 snímků za sekundu) [29, 18].

Třetí generace architektury – YOLOv3

Třetí verze YOLO (dále jen YOLOv3) provádí predikce boxů na **třech různých rozměrech**, které jsou dány zmenšením dimenzí vstupního obrazu o 32, 16 a 8, jak lze vidět na obrázku 3.7. Systém získává z různým rozměrů snímku vlastnosti pomocí konceptu podobného tzv. *feature pyramid networks*. K extraktoru vlastností z původní verze YOLO je přidáno několik dalších konvolučních vrstev. Poslední z těchto vrstev predikuje 3D tenzor obsahující ohraničující box, objektovost (s jakou jistotou se v boxu nachází objekt) a pravděpodobnosti jednotlivých tříd. Základní verze YOLO měla problém s detekcí malých objektů. Za pomoci predikce na různých rozlišeních má nyní YOLOv3 relativně **dobrou úspěšnost detekce malých objektů**, ale na druhou stranu se u této verze snížila úspěšnost detekce středně velkých a velkých objektů. Třetí verze obecně dosahuje lepších výsledků detekce než předchozí verze, a to za cenu snížení rychlosti detekce. Změny se také dočkala ztrátová funkce, kde byla v posledních třech částech (rovnice (3.5), (3.6) a (3.7)) nahrazena chyba *squared-error* za *cross-entropy error*. Pravděpodobnosti tříd a objektovost jsou predikovány pomocí logistické regrese. Predikce se rovná jedné pokud se překrývá s *ground-truth* objektem více než všechny ostatní predikované ohraničující boxy (které pokud překrývají *ground-truth* o předem určený práh jsou následně ignorovány) [31, 19].



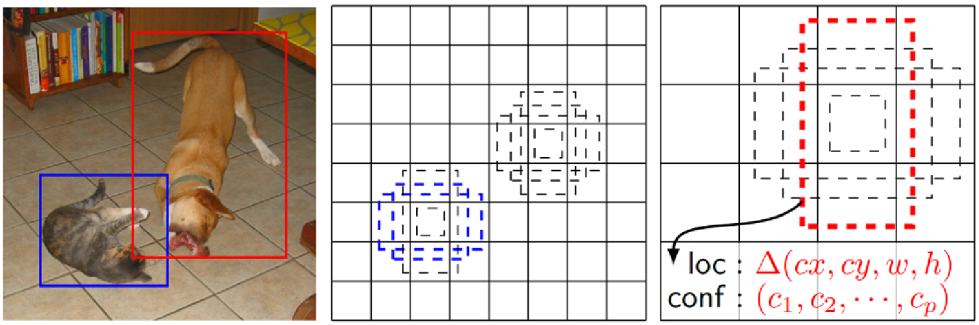
Obrázek 3.7: Architektura YOLOv3 se 106 vrstvami vytvářejícími predikce na třech různých rozlišeních (jmenovitě na vrstvách 82, 94 a 106).[6]

[6] Převzato z [19].

3.3 Single Shot Detector – SSD

Při studiu této problematiky jsem vycházel z [21]. Single shot detector (dále jen zkratka SSD), stejně jako YOLO, využívá pro detekci objektů jednu hlubokou neuronovou síť. Oba systémy SSD i YOLO spadají do kategorie tzv. *single-shot* detektorů. SSD je oproti ostatním metodám detekce objektů (R-CNN apod.) relativně jednoduché, protože odstraňuje krok návrhu kandidátních oblastí a fáze převzorkování vlastností a tedy dokáže zapouzdřit všechny výpočty do jediné neuronové sítě. Za pomoci detekce objektů množinou vrstev na **různých rozlišeních vstupního obrázku** lze dosáhnout relativně dobrých výsledků i při nízkém rozlišení vstupní vrstvy, což mimo jiné také zvyšuje rychlost detekce a snížení nároků na výpočetní zdroje. Systém SSD je založen na principu dopředné konvoluční neuronové sítě, která produkuje pevný počet ohraničujících boxů a zároveň také pravděpodobnosti výskytu jednotlivých tříd v daném boxu. Tento proces je následován tzv. *non-maxima suppression* (potlačením ne-maximálních hodnot) pro odstranění duplikátních detekcí a následným výstupem jednotlivých detekcí.

Vstupní vrstvy jsou založeny na běžném přístupu určeném pro klasifikaci objektů. Dále následují pomocné vrstvy sloužící pro vytvoření detekcí, skládající se z aktivačních map o různých rozměrech připojených za základní klasifikační část sítě. Tyto vrstvy slouží ke snížení rozměrů a vytvoření predikcí na různým rozměrech vstupu, jak lze vidět na obrázku 3.8. Každá z připojených aktivačních map může vytvářet pevný počet predikcí pomocí sady konvolučních filtrů a to o velikosti $3 \times 3 \times p$, kde p označuje počet kanálů, udávající parametry možných detekcí. Tyto parametry udávají relativní odsazení detekovaného boxu od původního boxu a pravděpodobnosti výskytu jednotlivých tříd. Pro každý z k boxů je predikováno c pravděpodobností tříd a 4 hodnoty udávající odsazení k původnímu boxu. Ve výsledku to pro obrázek velikosti $m \times n$ je $(c+4)mnk$ výstupních hodnot.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Obrázek 3.8: Princip vytváření predikcí systémem SSD na různých rozměrech vstupního obrázku s hodnotami odsazení vůči původnímu boxu a skóre pravděpodobností jednotlivých tříd.[7]

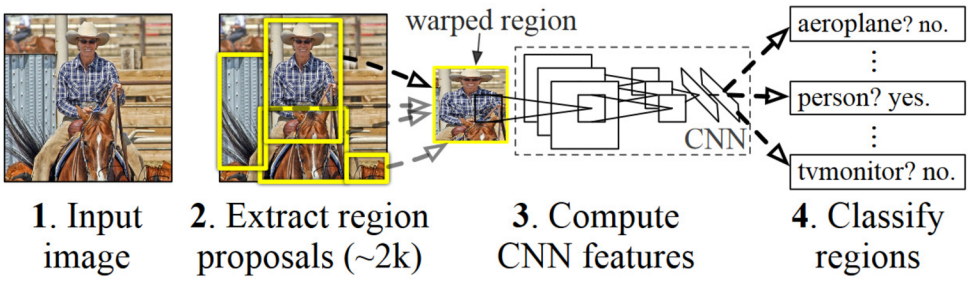
Pro výslednou detekci se aktivační mapy (různých rozměrů z nižších i vyšších vrstev) jedné konvoluční neuronové sítě kombinují dohromady pro dosažení vyšší jistoty (úspěšnosti) predikce.

[7] Převzato z [21].

3.4 Region-based Convolutional neural network – R-CNN

Při studiu této problematiky jsem vycházel z [13, 12, 35]. *Region-based Convolutional neural network* (oblastní konvoluční neuronová síť, dále jen zkratka R-CNN) kombinuje návrh kandidátních oblastí, ve kterých se mohou nacházet objekty, s klasifikací těchto oblastí založené na extrakci vlastností pomocí konvoluční sítě a aplikování lineárního SVM. Obecný algoritmus zpracování, vizualizovaný na obrázku 3.9, je následující:

1. Načtení vstupního snímku.
2. Extrakce kolem 2000 oblastí ze vstupního obrázku pomocí selektivního vyhledávání.
3. Extrakce vektoru vlastností z jednotlivých oblastí pomocí komplexních konvolučních neuronových sítí.
4. Klasifikace všech regionů pomocí sady tzv. *class-specific* SVM.



Obrázek 3.9: Princip zpracování snímku pomocí R-CNN.[8]

R-CNN využívá algoritmus selektivního hledání kandidátních oblastí představený v práci [35]. Ta udává, že oblasti můžou obsahovat více informací než samotné pixely, proto by měly být při hledání upřednostňovány. Algoritmus je založen na nalezení několika počátečních oblastí a následném iterativním shlukování oblastí (založeném na počítání podobnosti oblastí). Shlukování nejpodobnějších oblastí probíhá až do doby, kdy se celý snímek stane samostatnou oblastí. Protože algoritmus není invariantní vůči rozlišení vstupního snímku a množství navržených oblastní závisí na jeho velikosti, musí se všem snímkům před provedením selektivního hledání změnit velikost na pevných 500 px.

Pro výpočet vstupního vektoru konvoluční sítě o pevné délce z navržené oblasti, nezávisle na jejím rozměru, se používá technika zvaná *affine image warping*[9]. Architektura konvoluční sítě má homogenní strukturu a skládá se ze 13 konvolučních vrstev s jádry o velikosti 3×3. Mezi konvolučními vrstvami se nachází 5 maximalizačních sdružovacích vrstev a architektura je zakončena třemi plně-propojenými vrstvami. Protože navržené oblasti mohou obsahovat jen část objektu, byl zvolen prah IoU 0.3 – pokud při validaci oblast překrývá *ground-truth* o méně než daný prah, je označena jako negativní. Při změně prahu na 0.5 dojde k poklesu mAP na datové sadě PASCAL VOC o 5 bodů [13].

Po získaní vlastností z oblastí pomocí konvoluční sítě je použito jedno lineární SVM na každou třídu, které ohodnotí pravděpodobnost, zda vektor vlastností spadá do dané třídy. Poté je aplikováno potlačení ne-maximálních hodnot samostatně pro každou třídu, které ignoruje oblast, pokud má s oblastí vyšší IoU, než je naučený prah.

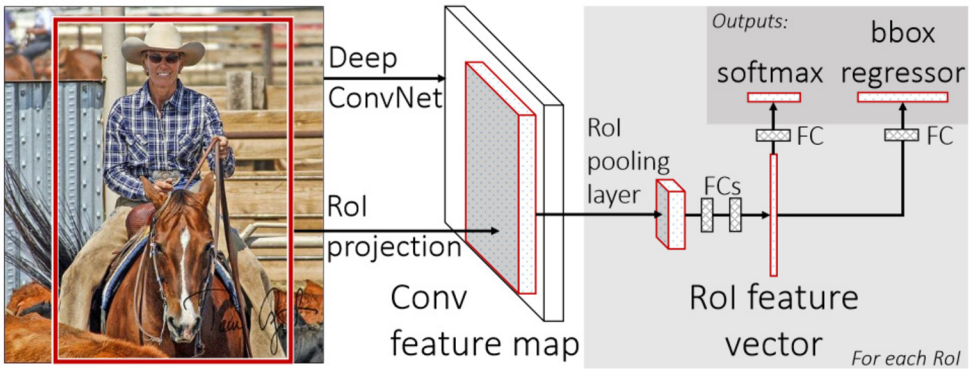
[8] Převzato z [13].

[9] *Affine image warping* – afinní obrazová deformace.

Rychlejší verze – Fast R-CNN

Ačkoli R-CNN dosahuje velmi dobré úspěšnosti detekce, doba vyhodnocení jednoho snímku trvá ~ 47 sekund. Z tohoto důvodu vznikly navazující verze Fast a Faster (rychlá a rychlejší) R-CNN. Zde je popsaná verze Fast R-CNN [12]. Tato verze se snaží odstranit nevýhody R-CNN a zároveň zvýšit rychlost a úspěšnost detekce. Oproti R-CNN jsou během trénování upravovány všechny vrstvy, trénování je jedno-stupňové za použití více-úlohové ztrátové funkce a extrahované vlastnosti už nejsou ukládány na disk.

Vstupem Fast R-CNN je vstupní snímek a sada navržených oblastí, které jsou poté zpracovány, jak lze vidět na obrázku 3.10. Jako první je zpracován vstupní snímek pomocí několika konvolučních a sdružovacích vrstev za účelem získání aktivační mapy. Poté jsou pro všechny navržené oblasti z aktivačních map získány vektory vlastností pevné délky, které jsou posléze načteny do sady plně-propojených vrstev. Ty se poté větví do dvou posledních výstupních vrstev. Jedna slouží pro určení pravděpodobností jednotlivých tříd pomocí aktivační funkce softmax. Těchto tříd je celkem K a jedna specifická třída, která slouží k odchycení veškerých detekcí pozadí. Druhá vrstva produkuje sadu 4 souřadnic predikovaných ohraničujících boxů pro každou z K tříd [12].



Obrázek 3.10: Architektura systému Fast R-CNN.[10]

Výsledkem těchto úprav je, že:

- Trénování Fast R-CNN je 9× rychlejší než trénování R-CNN.
- Detekce jednoho snímku provádí Fast R-CNN 213× rychleji.
- Na datové sadě PASCAL VOC 2012 dosahuje Fast R-CNN lepší úspěšnosti 66 % mAP (oproti 62 % R-CNN).

[10] Převzato z [12].

Kapitola 4

Návrh systému pro detekci dopravního značení

Tato kapitola se zabývá porovnáním jednotlivých technik pro detekci a klasifikaci značek popsaných v kapitolách 2 a 3, návrhem systému pro detekci dopravního značení, a také porovnáním datových sad a neuronových sítí, které lze použít k trénování a testování.

4.1 Datové sady dopravního značení

Datové sady jsou neodmyslitelnou součástí strojového učení. Dostatek datových sad je jedním z důvodů, proč se neuronové sítě staly v dnešní době tak populárními. Datových sad dopravních značek existuje spousta, avšak většina z nich obsahuje pouze vyřezané dopravní značky z celých snímků, na kterých se běžně trénují klasifikátory (tedy systémy, které slouží pouze k rozhodnutí o třídě značky, nikoli k detekci a lokalizaci v rámci celého snímku). Běžná velikost snímků v takovýchto sadách bývá velmi malá, například 64×64 pixelů. Jak je popsáno v sekci 3.2, systém YOLO použitý v této práci pro detekci se trénuje na tzv. celých (nebo také plných) snímcích. Dokáže se z nich naučit jak celkový kontext, tak i relativní velikost detekovaných značek. Proto bylo potřeba najít takovou datovou sadu, která bude obsahovat celé snímky.

Podařilo se najít čtyři takové datové sady dopravních značek: Německou (GTSD) [16], Belgickou (BTSD) [27], Ruskou (RTSD) [32] a Americkou, kde však používají naprosto rozdílnou normu dopravního značení, a proto byla v rámci této práce nepoužitelná. Počáteční písmeno těchto datových sad běžně slouží k jejich identifikaci (název státu, kde byly snímky pořízeny) a končí písmenem "D", které je zkratkou pro anglické *detection* a běžně značí, že se jedná o datovou sadu s celými snímky používanými pro detekci. Stejně tak "C" jako *classification*. V tabulce 4.1 lze najít velikost jednotlivých sad, dále pak počet tříd a značek v nich obsažených.

Datová sada	Počet snímků	Počet tříd	Počet značek
GTSD [16]	900	43	1213
BTSD [27]	25630	108	13444
RTSD [32]	179138	156	104358

Tabulka 4.1: Tabulka velikostí, počtu tříd a značek jednotlivých datových sad značek.

Na obrázku 4.1 lze vidět ukázku snímků z jednotlivých datových sad. Jak lze vidět v tabulce 4.1, velikost jednotlivých datových sad se liší. Stejně tak se liší kvalita, velikosti a anotace snímků.

Obrázek 4.1: Ukázka snímků datových sad (**zleva**) GTSD, BTSD a RTSD.

Nejvyšší kvalitu má Německá datová sada, tedy GTSD, avšak nevýhoda této sady je velmi malý počet značek na třídu – v průměru to je cca. 28, a to pro natrénování a zároveň vyhodnocení kvalitního detektoru nestačí. Ruská datová sada má naopak značek na třídu dost, ale kvalita těchto značek a snímků obecně je velmi špatná. Značky jsou malé a rozmazané. Jako dostatečná se tedy jeví Belgická sada, která má poměrně dost snímků i dobrou kvalitu, ale nevýhodou této sady je, že téměř dvě třetiny datové sady je pouze tzv. *negativní datová sada* (neobsahující žádné značky). I přes to ale byla tato sada vybrána jako hlavní pro trénování i testování.

4.2 Návrh generátoru syntetických dat

Z důvodu nedostatku datových sad dopravních značek s plnými snímky, a také za účelem porovnání detektoru trénovaného na reálných a syntetických datech jsem se rozhodl v rámci této práce vytvořit generátor syntetických datových sad. Cílem bylo iterativně zlepšovat generátor, aby výsledná data co nejvíce odpovídala reálným snímkům a zlepšování výsledků, aby se kvalita detektoru trénovaného na syntetických datech vyrovnala, či dokonce překonala kvalitu detektoru trénovaného na reálných datech. Testování obou detektorů probíhalo na reálných snímcích Belgické sady. Aby se syntetická data co nejvíce podobala reálným snímkům, je potřeba dosáhnout velké variability snímků simulující různé světelné podmínky, přírodní vlivy, natočení, vandalismus, apod. To následně umožňuje hluboké konvoluční síti se naučit dostatek vlastností a dokázat správně generalizovat na reálných snímcích. Z důvodu, že se zvolená metoda YOLO učí i celkový kontext ve kterém se objekty nachází, byly pro generování využity jako pozadí snímky zachycené z jedoucího vozidla (převážně v městské zástavbě), neobsahující žádné značky. Značky, které by nebyly anotovány a negativně by ovlivňovaly proces trénování.

Alternativním způsobem pro získání kvalitní dostatečně velké anotované datové sady, bez nutnosti anotace každého snímku jednotlivě, by bylo využití existujícího detektoru značek (pokud možno toho s nejvyšší úspěšností) a jeho aplikování na ne-anotovanou datovou sadu. Poté by stačilo výsledky projít a případně upravit nevalidní detekce, což by zabralo podstatně méně času, než anotovat celou datovou sadu ručně. Protože částečným cílem práce je i porovnání modelů trénovaných na reálných a syntetických datech, rozhodl jsem se tento způsob nevyužít a soustředit se na tvorbu syntetické datové sady.

4.3 Porovnání a výběr metod pro detekci a klasifikaci

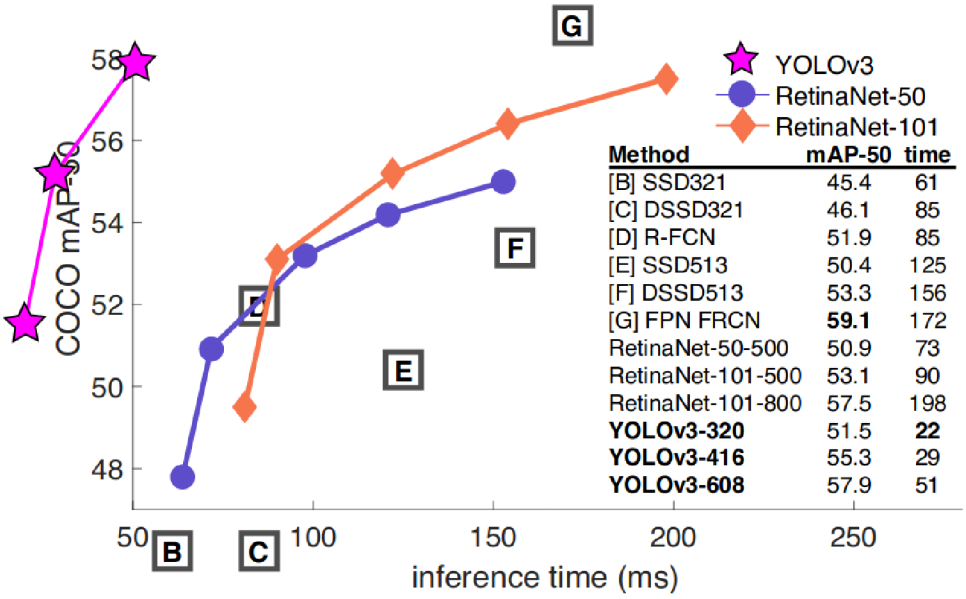
Tato práce si dává za cíl použití **moderních technik pro detekci**, a proto se do užšího výběru dostaly pouze metody jako SSD, YOLO, Fast(er) R-CNN a RetinaNet. V tabulce 4.2 lze vidět komplexní porovnání úspěšností těchto metod na datové sadě COCO. Tabulka zobrazuje úspěšnost AP v závislosti na různých prazích IoU a různých velikostech detekovaných objektů (*small*, *medium* a *large*, tzn. malé, střední a velké). Tabulka je rozdělena na dvě části podle počtu kroků vykonávaných systémem při detekci. Protože při detekci je důležitá také rychlost, je na obrázku 4.2 graficky znázorněno porovnání metod podle času potřebného pro vyhodnocení jednoho snímku v závislosti na úspěšnosti.

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

Tabulka 4.2: Porovnání detekčních metod na různých metrikách [31].

Jak lze z tabulky vyčíst, rozdíl mezi SSD a YOLOv3 v rámci úspěšnosti při použití moderní metriky AP, počítané na 10 různých prazích IoU, není velký a oba tyto systémy předčil model RetinaNet. Při použití tradiční metriky AP_{50}, počítané pouze na IoU = 0.5 má YOLOv3 velmi dobrou úspěšnost srovnatelnou s RetinaNet a podstatně lepší úspěšnost než obě verze SSD. Se zvyšujícím se prahem IoU značně klesá úspěšnost metod YOLOv3 a SSD, což indikuje nedokonalou lokalizaci objektů (predikované boxy neodpovídají přesně *ground-truth*, tedy skutečné pozici objektů).

Protože u SSD slouží k detekci pouze několik počátečních vrstev, je úspěšnost detekce malých objektů poměrně nízká. V minulosti mělo s detekcí malých objektů velké problémy i YOLO, ale třetí verze provádějící detekci na třech různých rozlišeních značně zlepšila jejich detekci, ale stále je podstatně nižší než u RetinaNet.



Obrázek 4.2: Porovnání detekčních systémů v poměru úspěšnosti a rychlosti vyhodnocení detekce na GPU Titan X.[1]

YOLOv3 je jednoznačně nejrychlejší metodou ze všech uvedených a i přesto dosahuje velmi dobré úspěšnosti detekce. Odlehčená verze dokonce dosahuje dokonce rychlosti až 150 snímků za sekundu výměnou za několik bodů mAP. Při použití běžné grafické karty (oproti Titan X, na níž jsou uvedené rychlosti získány) výrazně klesne rychlost trénování a zejména detekce, a proto jsem se rozhodl použít systém Tiny YOLOv3, což je již zmíněná odlehčená verze systému YOLOv3, se kterou by bylo možné dosáhnout zpracování v reálném čase i na běžné grafické kartě a přitom dosáhnout dobré úspěšnosti detekce.

4.4 Neuronová síť Darknet

Darknet [28] je *open-source* (česky otevřený software) aplikační rámec neuronové sítě napsané v programovacím jazyce `C` a `CUDA`[2]. Je možné s ním provádět výpočty jak na CPU (procesoru), tak na GPU (grafické kartě). *Darknet* je možné používat jak na operačním systému Linux, tak i upravenou verzi na systému Windows.

Backbone	Top-1	Top-5	Bn Ops	BFLOPS/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Tabulka 4.3: Porovnání neuronových sítí: Přesnost, počet bilionů operací bez i s plovoucí řádovou čárkou a počet snímků za sekundu [31].

Kvůli parametrům neuronové sítě Darknet-53 jsem se rozhodl ji použít v této práci.

[1] Převzato z [31].
[2] CUDA – *Compute Unified Device Architecture*.

Kapitola 5

Implementace navrženého detektoru

V této kapitole je popsána implementace generátoru syntetických datových sad vytvořeného v rámci této práce. Dále je zde popsán proces vytváření detektoru dopravních značek, jeho trénovaní, vyhodnocení a iterativního zlepšování úspěšnosti.

5.1 Generátor syntetických datových sad

Generování syntetických datových sad pro trénování hlubokých neuronových sítí se stává čím dál populárnější technikou, která usnadňuje proces trénovaní o nutnost anotovat datovou sadu (či rozšíření existující datové sady) a zároveň umožňuje dosažení dobré úspěšnosti detekce/klasifikace. Avšak problém je dokázat generovat datovou sadu tak, aby model na ní trénovaný dokázal správně generalizovat na reálných snímcích, což je čím dál častěji předmětem výzkumu a bude popsáno i v této práci.

Pro generování se nabízí hned několik způsobů. Aplikační rámce jako např. PyTorch či TensorFlow obsahují zabudované „generátory“, které lze využít pro rozšíření již existující datové sady. Ty fungují na principu modifikace jednotlivých snímků datové sady (např. otočení snímku kolem osy y, změna světelných podmínek, apod.) a jejich následné připojení k datové sadě. Další způsob využívaný zejména pro komplexní objekty, jako je například tvář člověka, je vymodelování objektu (např. v programu *Blender*) a poté pomocí skriptu „nafocení“ tohoto objektu z různých uhlů a za různých světelných podmínek. Pro generování plných snímků, které jsou pro trénování neuronové sítě YOLO potřeba, lze využít techniku umisťování těchto objektů do snímků pozadí a modifikaci jejich vzhledu za účelem dosažení větší variability vzniklé datové sady. Poslední zmíněnou technikou je GAN[1]. Tato technika, jak již jméno napovídá, je založena na „zápasení“ dvou neuronových sítí a slouží mimo jiné pro generování syntetických snímků. Je to ovšem metoda učení bez učitele, a proto se nedá použít pro generování objektů do plných snímků.

Systém YOLO je trénovaný na plných snímcích, a proto jak bylo navrženo, byl v této práci využit způsob umisťování dopravních značek do plných snímků pozadí. Dále byla získána sada snímků z palubní kamery a veškeré snímky obsahující dopravní značky byly odstraněny, protože by negativně ovlivňovaly proces trénovaní.

[1] GAN – *Generative adversarial network.*

Efekty aplikované na značky

Aby bylo dosaženo větší variability datové sady a konvoluční neuronová síť se byla schopna naučit co nejvíce vlastností dopravních značek, bylo před samotným vložením značky do pozadí provedeno několik operací upravující vzhled značky. Seznam operací, které se s určitou pravděpodobností aplikují na vyřezané dopravní značky, je následující:

- **Gamma korekce** – Slouží k simulaci různých světelných podmínek. Pravděpodobnost aplikace tohoto efektu je 15 %.
- **Úprava jasu** – Slouží k simulaci různých světelných podmínek. Pravděpodobnost aplikace tohoto efektu je 15 %.
- **Úprava jasu podle pozadí** – Slouží k získání průměrného osvětlení v místě, kam bude dopravní značka umístěna a následné úpravě jasu značky podle získané hodnoty. Lze nahradit za předchozí efekt.
- ***Salt and pepper* šum** – Slouží k simulaci poškození značky. Pravděpodobnost aplikace tohoto efektu je 5 %.
- **Gaussovský šum** – Slouží k simulaci poškození značky. Pravděpodobnost aplikace tohoto efektu je 5 %.
- **Překrytí části značky** – Simuluje běžně se vyskytující překrytí značky. Pravděpodobnost aplikace tohoto efektu je 5 %.
- **Gradient osvětlení** – Simuluje reálné osvětlení značek. Pravděpodobnost aplikace tohoto efektu je 15 %.
- **Rotace v ose Z** – Simuluje záběr z různých úhlů. Pravděpodobnost aplikace tohoto efektu je 25 %.
- **Rotace v ose Y** – Simuluje záběr z různých úhlů. Pravděpodobnost aplikace tohoto efektu je 25 %.
- **Rozmanání značky** – Simuluje vyšší rychlost vozidla či nižší kvalitu kamery. Pravděpodobnost aplikace tohoto efektu je 5 %.
- **Úprava barvy (HSV)** – Simuluje značky různých Evropských norem (viz obrázek 2.2). Pravděpodobnost aplikace tohoto efektu je 10 %.
- **Změna velikosti značky** – Simuluje různé vzdálenosti značky od kamery. Pravděpodobnost aplikace tohoto efektu je 90 %.

Naopak techniky, které úspěšnost nezvýšily, či dokonce snížily jsou následující:

- Rotace značky kolem osy y
- Přidání luminiscenčního podkladu značek
- Přidání ocelové konstrukce značek
- Výběr pozic, kam značku umístit a následná úprava velikosti podle umístění

Z důvodu, že se dopravní značky nachází většinou po stranách silnice, bylo provedeno několik pokusů s umisťováním značek pouze na okraje, a to tak že čím blíže byly středu, tím menší velikosti nabývaly (simulace reálného prostředí). Tyto pokusy však neprokázaly zvýšení úspěšnosti. Luminiscenční podklad i ocelová konstrukce měly za úkol přidat přirozený kontext, ve kterém se značky nachází. Tyto přidané objekty ovšem výrazně snižovaly úspěšnost detekce. Jedna z vlastností mnoha značek je, že jsou osově souměrné, a proto byla vyzkoušena také rotace značek kolem osy y (kromě značek obsahující šipky vlevo/vpravo apod.), ale ke zlepšení úspěšnosti to nevedlo.

Způsob generování s transparentními značkami

Tato technika je založena na principu umisťování transparentních značek (obsahující alfa-kanál) do snímků pozadí. Výhodou tohoto přístupu je, že kolem značky nevzniká výrazná přestupná hrana, jako je tomu u druhého způsobu používající vyřezané značky.

Obrázek 5.1: Příklad vygenerovaných dat za pomoci transparentních značek. Značkám bylo před vložením do snímku pozadí upraveno osvětlení, velikost a natočení.

Naopak nevýhodou tohoto přístupu je, že aby bylo dosaženo jisté variability takto vygenerované datové sady, bylo by potřeba velké množství značek vyřezaných z reálných snímků. Tento problém se dá řešit pomocí již zmíněných efektů aplikovaných na dopravní značky popsaných v sekci 5.1 a vizualizovaných na obrázku 5.2.

Obrázek 5.2: Příklad aplikace všech efektů modifikujících vzhled dopravních značek na jedné značce typu *stop*. Originál vlevo nahoře.

Pravděpodobnost aplikace efektu je dána pravděpodobností s rovnoměrných rozložením, takže se běžně stává, že se aplikuje více efektů na jednu značku (například natočení, změna jasu i přidání šumu). Každý efekt používá jiné hodnoty, například natočení značky se udává ve stupních. Z daného, předem definovaného, rozsahu aplikovaných hodnot se pseudonáhodně vybere hodnota při každé aplikaci efektu. Kdyby se při každé aplikaci efektu používaly stejné hodnoty, vzhled značek by se začal brzy opakovat.

Pravděpodobnost aplikace efektu i rozsah aplikovaných hodnot byly pečlivě vybírány a upravovány jak pomocí experimentů, tak inspirací vzhledu značek z reálného světa. Například vizuálně poškozené značky se příliš často nevyskytují, proto byla pro přidání šumu zvolena poměrně nízká pravděpodobnost. Naopak různé světelné podmínky či natočení značek jsou běžně se vyskytující jevy, proto jim byla přiřazena poměrně vysoká pravděpodobnost aplikace.

Při implementaci jednotlivých efektů byla vždy prováděna kontrola realističnosti vzhledu vygenerovaných značek. Tzn. zda vygenerované značky vypadají jako by pocházely z reálného světa. Dosažení opravdu reálného vzhledu je velmi problematické a vyžadovalo by složitou 3D analýzu jak značky, tak pozadí, kam se značka umisťuje. Ani při dosažení realistického vzhledu (pro člověka) není jisté, zda se dokáže neuronová síť naučit potřebné vlastnosti a dokáže poté správně generalizovat na reálných snímcích. Lidské oko je totiž více citlivé na jiné faktory, než počítač.

Způsob generování s ořezanými značkami

Tento způsob je založen na použití reálných snímků vyřezaných dopravních značek určených pro trénování klasifikátorů. Těchto značek je velké množství, ale nevýhodou je, že po vložení do snímku pozadí kolem těchto značek vzniká výrazná přestupná hrana, jak lze vidět na obrázku 5.3 (vlevo), která negativně ovlivňuje proces trénování. Řešením je použití tzv. *Poisson blending*, které kombinuje gradienty a intenzity barev obou obrázků a ve výsledku to vypadá tak, že objekt do daného kontextu patří. Nevýhodou je, že algoritmus *Poisson blending* začíná počítat pouze od hraničních bodů, kam se objekt umisťuje, a proto pokud jsou tyto hraniční body příliš světlé, sníží se i intenzita barev vkládané značky a ve výsledku není téměř vidět. Podobně tomu je pokud jsou hraniční body příliš tmavé, až černé [26]. Příklad výsledku vložení pomocí techniky *Poisson blending* lze vidět na obrázku 5.3 (vpravo). Použité vyřezané značky pochází z Německé datové sady, protože použití značek

vyřezaných z Belgické datové sady, na které se později provádělo vyhodnocení by mohlo zkreslit výsledky.

Obrázek 5.3: Porovnání vložení ořezané dopravní značky do pozadí bez použití (**vlevo**) a s použitím (**vpravo**) techniky *Poisson blending*.

Množství generovaných dat

Pro trénování na syntetických datech bylo potřeba zjistit optimální počet generovaných značek na třídu. V tabulce 5.1 lze vidět závislost počtu syntetických snímků na výsledné úspěšnosti. Jak lze z tabulky vyčíst, mezi 200, 500 a 1000 snímky se vyskytují velké skoky v úspěšnosti, zatímco mezi 1000 a 2000 se úspěšnost již téměř nezvýšila, zato se razantně prodloužila doba trénování. Jako optimální počet pro budoucí trénování na syntetických datech tedy vyšlo 1000 značek na třídu.

Tabulka 5.1: Závislost úspěšnosti YOLOv3-608 na množství syntetických snímků na třídu.

Počet snímků	Úspěšnost v mAP
200	63.2 %
500	77.9 %
1000	86.3 %
2000	**87.6** %

5.2 Konfigurace neuronové sítě při trénovaní a testování

Pro konfiguraci neuronové sítě se používá konfigurační soubor, `yolo-train.cfg`, při jehož studiu a popisu jsem vycházel z [11, 14]. Nejdůležitější parametry, které lze v rámci konfiguračního souboru nastavit a se kterými bylo v rámci této práce experimentováno jsou následující:

- **anchors** – Výstupní hodnoty neuronové sítě YOLO neudávají přímo výšku a šířku ohraničujícího boxu, ale udávají pouze *offset* vůči před-definovaným „kotvám“ (anglicky ***anchors***) s nejvhodnější velikostí. Tyto kotvy se vyskytují ve vrstvách `[yolo]`

a jsou definovány následovně: `anchors = 4,7, 7,15, 13,25, 25,42, 41,67, 75,94, 91,162, 158,205, 250,332`.

- **mask** – Každá `[yolo]` vrstva potřebuje vědět o všech kotvách, ale jen některé z nich může použít. Proto se do těchto vrstev zavedl seznam *mask*, ve kterém se udají indexy kotev, které smí daná vrstva použít. Pro použití pouze prvních třech kotev lze použít `mask = 0,1,2`.
- **filters** – Udává počet konvolučních jader v dané vrstvě. Nejvhodnější počet filtrů je podle autorů `filters = (num_classes + 5) * 3`. Hodnota 5 označuje počet predikovaných hodnot pro jeden ohraničující box (x, y, výška, šířka a skóre *objectness*) a 3 v tomto případě označuje počet kotev.
- **activation** – Určuje aktivační funkci vrstvy. Může být např. *ReLu*, *sigmoid*, *tanh*, *linear*, *leaky*, apod.
- **random** – Pokud se rovná jedné, daná vrstva mění velikost vstupního snímku vždy po několika dávkách na různé velikosti.
- **angle** – Otočení vstupního snímku o uvedený počet stupňů.
- **batch** – Dávka, tedy množství snímků k dopředné propagaci skrze neuronovou síť sloužící k výpočtu gradientu, který se poté používá k úpravě vah pomocí zpětné propagace.
- **subdivisions** – Dávka je rozdělena do tohoto množství bloků. Autoři neuronové sítě Darknet uvádí, že jednotlivé bloky *subdivisions* jsou počítány na GPU paralelně, ovšem trénování při `subdivisions=16` bylo pomalejší, než když byl tento parametr roven čtyřem či osmi.
- **flip** – Pokud je nastaven na 1, jsou snímky po několika dávkách otočeny. Podle autorů zvyšuje použití této funkce úspěšnost, ale protože je potřeba rozlišit značky s šipkami vlevo/vpravo, nebylo možné jej v rámci této práce použít.
- **step** – Určuje po kolika dávkách se má upravit učící faktor.

Dále je potřeba nastavit v souboru `obj.data` počet tříd, cesty k trénovací a validační datové sadě, soubor se jmény jednotlivých tříd (`obj.names`) a cestu, kam se budou ukládat váhy modelu.

5.3 Trénování neuronové sítě

Trénování neuronové sítě je zpravidla proces velmi náročný na výpočetní prostředky (záleží na konfiguraci parametrů sítě) a není jej tedy vhodné (mnohdy ani možné) provádět na osobním počítači. Tato podkapitola se zabývá všemi aspekty, které bylo potřeba splnit k úspěšnému natrénování sítě včetně experimentů.

Sdílený cluster

Proces trénování hluboké konvoluční neuronové sítě je velmi náročný na výpočetní prostředky. To je jeden z důvodů, proč nelze komplexní neuronové sítě s velkým počtem vstupů

a vrstev trénovat na běžných osobních počítačích a bylo tedy potřeba najít výkonný sdílený cluster, na kterém bude možné výpočty provádět. Tyto služby, kterých jsem po značnou část práce využíval, poskytuje pro akademickou obec zdarma virtuální organizace MetaCentrum [23].

MetaCentrum funguje na principu vytváření úloh pomocí plánovacího systému PBS[2]. Na začátku je nutné vytvořit dávkový skript, který obsahuje posloupnost instrukcí, jež se mají na sdíleném clusteru v rámci úlohy vykonat. Poté je potřeba úlohu naplánovat, tedy zařadit do fronty, kde bude čekat na spuštění (až úloha přijde na řadu a bude dostatek požadovaných zdrojů). V rámci plánování úlohy je možné specifikovat, jaké zdroje úloha potřebuje. Doba strávená ve frontě je úměrná množství požadovaných výpočetních prostředků.

Obecné parametry, které se běžně specifikují, jsou například počet procesorů, grafických čipů, množství paměti apod. Dále je možné specifikovat detailnější parametry, jako například verzi CUDA, specifický cluster, zařazení do speciální fronty, apod. Příklad spuštění úlohy, který byl využíván pro trénování na Belgické datové sadě je následující:

```
qsub -q gpu -l ncpus=8:ngpus=1:mem=25gb:gpu_cap=cuda35:local=5gb BTSD.sh
```

Od počátku se vyskytoval závažný problém s trénováním neuronové sítě *Darknet*. I přes přiřazení poměrně velkého množství výpočetních zdrojů úloha ihned od spuštění alokovala mnohem více zdrojů, než měla přiřazeno a plánovací systém byl tedy nucen tuto úlohu ukončit. Problém byl z počátku řešen pomocí umístění úlohy do speciální fronty zvané `exclhost`, kde úloha smí použít neomezené množství výpočetních zdrojů. Toto privilegium s sebou ovšem přináší další problém, a to velmi dlouhou dobu čekání na spuštění úlohy. Po jisté době procházení zdrojových textů neuronové sítě bylo zjištěno to, co bylo od počátku jasné. A to, že tento program nepočítá se spuštěním na sdíleném clusteru a alokuje si co nejvíce zdrojů dostupných na daném stroji. Problém byl vyřešen jednoduchou modifikací neuronové sítě (nastavení maximálního počtu alokovaných jader procesoru) a jejího opětovného překladu. Poté už úloha používala pouze přiřazené množství zdrojů a trénování probíhalo v pořádku.

Spuštění trénování

Na začátku je potřeba provést překlad neuronové sítě *darknet*. Je možné využít jak Linuxovou verzi, tak verzi pro Windows. Během celé práce byla využívána pouze verze pro Linux. Pro překlad na Linuxovém operačním systému slouží soubor `Makefile`, kde je možné nastavit několik konstant dle potřeby. Nastavované parametry v průběhu tvorby této práce jsou následující:

- `GPU=1` – Nejdůležitější parametr při trénování i testování. Zajišťuje akceleraci výpočtů na GPU za pomoci CUDA API.
- `CUDNN=1` – Pužití knihovny cuDNN umožňující tvorbu neuronových sítí za pomoci CUDA.
- `CUDNN_HALF=1` – Sloužící pro zrychlení detekce a trénování na určitých grafických čipech.
- `OPENCV=1` – Umožňuje použití knihovny OpenCV. V *darknet* mj. používané při zpracování videa.

[2]PBS – *Portable batch system.*

Dále je možné nastavit méně důležité parametry, jako například akcelerace výpočtů na CPU, které ovšem nebyly v této práci využívány – veškeré trénování i vyhodnocení bylo prováděno na GPU. Po nastavení kompilačních konstant (a při použití CUDA/cuDNN instalace potřebných ovladačů grafického čipu NVIDIA) je možné provést samotný překlad. Pokud je překlad úspěšný, je v kořenovém adresáři vytvořen spustitelný binární soubor `darknet`.

Příkaz pro spuštění trénování neuronové sítě *darknet* je následovný:

```
./darknet detector train obj.data yolo-train.cfg conv.weights
```

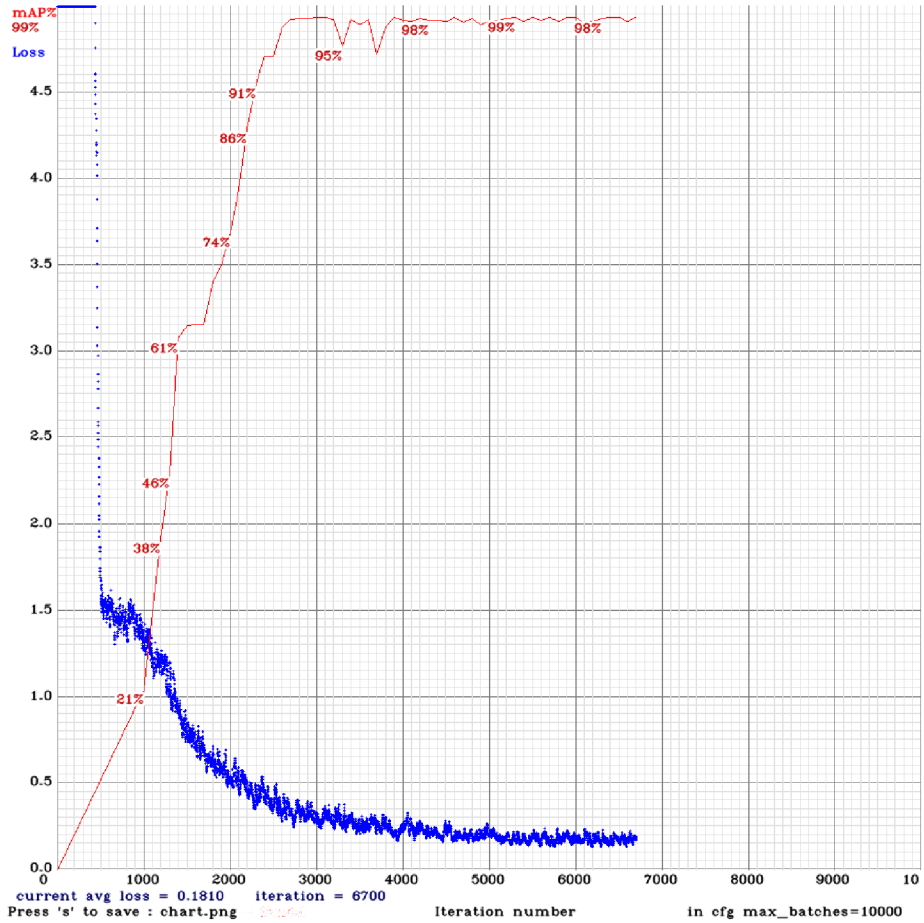
Soubory `obj.data` a `yolo-train.cfg` již byly popsány v sekci konfigurace neuronové sítě. Další, tentokrát volitelný parametr, je před-trénovaný model (někdy taky zvané váhy). Autoři systému poskytují několik před-trénovaných vah pro různé modely (s různými počty vrstev). Tyto modely byly před-trénovány na datové sadě *ImageNet*, a proto se názory na použití před-trénovaných vah pro detekci vlastních objektů (například v tomto případě dopravních značek) liší. Pravda však je, že vstupní vrstvy před-trénovaného modelu jsou již naučeny detekovat hrany, tvary a další obecné vzory, které jsou použitelné pro jakékoli třídy a detekce určité třídy nastává až v plně propojené vrstvě. Neuronová síť musí dokázat detekci hran a tvarů tak jako tak, a proto je lepší využít již před-trénované hodnoty. Mimo jiné vlastnosti vstupních vrstev naučené z rozsáhlé datové sady *ImageNet* budou podstatně lepší, než vlastnosti naučené z omezené datové sady dopravních značek a kvalitní před-trénovaný model bude trénování vést správným směrem. Při použití před-trénovaného modelu dosáhl výsledný model o cca. 2 % lepší mAP a při trénování potřeboval o 4000 iterací méně, než při trénování modelu úplně od nuly.

Průběh a ukončení trénování

Program *darknet* byl doplněn o jednoduchou vizualizaci hodnot konvolučních vrstev. Několik z takto vizualizovaných filtrů lze vidět na obrázku 5.4.

Obrázek 5.4: Vizualizace třech filtrů z jedné z konvolučních vrstev modelu.

V průběhu trénování neuronová síť za pomoci knihovny `OpenCV` (pouze při `OPENCV=1`) vizualizuje hodnoty ztrátové funkce a mAP na trénovací sadě, jak lze vidět na obrázku 5.5. Při trénování na sdíleném clusteru ovšem není možné využít GUI pro zobrazení, a proto *darknet* poskytuje možnost streamování výsledků pomocí `mjpeg` serveru na určitém portu při zadání přepínače `-mjpeg_port <port>`. Přístup k výsledkům je potom možný pomocí prohlížeče `http://<ip-address>:<port>`.



Obrázek 5.5: Výstupní obrázek `chart.png` vizualizující stav trénování neuronové sítě s počtem iterací na ose x a hodnotou ztrátové funkce na ose y. Zvyšující se červená křivka reprezentuje hodnotu mAP na validační datové sadě a modré vzorky hodnotu ztrátové funkce při určité iteraci.

Tyto hodnoty byly získány při trénování na syntetické datové sadě skládající se ze třech tříd, každá po 1000 trénovacích snímcích. Jak lze v grafu vidět, od 6000 iterací se mAP již nezvyšuje a hodnota ztrátové funkce se zde ustálila. Ideální hodnota pro zastavení trénování je podle autorů dosažení počtu iterací 2000-krát počet tříd. To je v tomto případě třech tříd přesně 6000 iterací. Poté začne u modelu docházet k jevu zvanému přetrénování, což způsobuje že model dokáže správně detekovat pouze objekty z trénovací datové sady.

5.4 Experimenty

Poté co bylo dosaženo dobré úspěšnosti a nedařilo se ji zvýšit za pomoci standardních metod, nastal čas provést řadu experimentů, které by mohly zajistit lepší parametry výsledného modelu.

Smíchání reálné se syntetickou datovou sadou

Při trénování pouze na syntetické datové sadě se může konvoluční síť naučit vlastnosti vyskytující se pouze u syntetických snímků a nedokázat poté správně generalizovat na reál-

ných snímcích. Proto bylo provedeno několik pokusů s přimícháním malého počtu reálných snímku k syntetické datové sadě. Tato malá sada reálných snímků by měla „usměrňovat“ trénování správným směrem a výsledný model by poté měl dosáhnout lepší úspěšnosti na reálných snímcích. Při přidání cca. 30 reálných snímků k 1000 syntetickým snímkům na třídu došlo ke zvýšení mAP o 2 %.

Hard negative mining

Hard negative mining je technika postupného doplňování trénovací datové sady snímky, ve kterých bylo chybně interpretováno pozadí jako některý z detekovaných objektů. YOLO je trénováno na plných snímcích, kde detekované objekty tvoří velmi malou část z celého snímku a zbytek snímku se dá tedy považovat za tzv. negativní datovou sadu, proto se může doplňující negativní datová sada zdát nadbytečná. Po provedení několika testů se ukázalo, že ačkoli došlo ke snížení počtu falešně pozitivních detekcí, což bylo cílem, snížil se i počet správných detekcí a úspěšnost klesla o 3.5 % mAP z důvodů přetrénování.

Binární model

Model XNOR-net pracuje jak se vstupem, tak s vahami ve formě binárních čísel. Tento model se používá pro zvýšení rychlosti na CPU a snížení výpočetních nároků, například pro použití v přenosných zařízeních s omezenými zdroji. Při testování s uvedeným modelem došlo k poklesu mAP o více než 20 % oproti běžnému modelu pracujícími s reálnými čísly a nevýraznému zvýšení rychlosti detekce.

Přidání dalších vrstev do architektury

V modelu Tiny YOLOv3 se vyskytují dvě vrstvy `[yolo]` s kotvami. Protože značky mohou nabývat různých velikostí, pokusem bylo přidání třetí takovéto vrstvy s `mask = 6,7,8` (používající největší *kotvy*) do modelu Tiny. Výsledkem byla úspěšnost obdobná modelu se dvěma vrstvami. Důvodem pravděpodobně bylo, že se v Belgické datové sadě nevyskytuje mnoho velkých značek, které by tato vrstva detekovala.

Přepočítání kotev

Recalculate anchors, tedy přepočítání „kotev“, slouží k výpočtu nových hodnot tak, aby co nejlépe odpovídaly velikosti vstupních snímků. Tato technika má zvýšit průměrné IoU, tzn. zajistit lepší úspěšnost detekce. Při přepočítání kotev na velikost vstupních snímků došlo ke zlepšení o 0.5 % mAP.

Počáteční hodnoty modelu

Autoři systému poskytují několik před-trénovaných vah pro různé modely. Tyto váhy byly před-trénovány na datové sadě *ImageNet* a protože konvoluční síť musí detekci hran a tvarů umět, je lepší využít již před-trénované hodnoty. Při použití před-trénovaných vah dosáhl výsledný model o cca. 2 % lepší mAP a při trénování potřeboval o 2000 iterací méně, než při trénování modelu od „nuly“. Z tohoto důvodu byly při každém dalším trénování použity před-trénované váhy.

5.5 Podpůrné prostředky pro trénování a vyhodnocení

Konvoluční neuronová síť YOLO pracuje s obrovským množstvím snímků doplněných o anotace v samostatných textových souborech, což vyžaduje jejich automatické zpracování. Stejně tak výstupem této sítě je nesčetné množství souborů s predikcemi v jistých formátech, které je taktéž třeba automaticky zpracovat. Z těchto důvodů bylo vytvořeno několik skriptů pro automatické zpracování a stejně tak použito několik již existujících. V této podkapitole budou zmíněny nejdůležitější z nich.

Trénování

Anotace datových sad jsou běžně udávány v různých formátech. YOLO striktně vyžaduje jistý formát, do kterého bylo potřeba tyto anotace převést. Formát anotace vyžadovaných systémem YOLO je následující:

```
<class> <x> <y> <width> <height>
```

Hodnota *class* je celé číslo, které spadá do intervalu $\langle 0, classes - 1 \rangle$. Parametry x, y, *width* a *height* udávají relativní souřadnice (střed) a proporce daného objektu vůči výšce a šířce celého snímku. Náleží tedy do intervalu $\langle 0, 1 \rangle$. Do tohoto formátu bylo potřeba převést anotace všech použitých datových sad (GTSD, BTSD a RTSD).

Vyhodnocení

Pro predikované boxy je formát podobný, pouze je doplněn o jistotu (confidence), s jakou je objekt predikován a souřadnice i proporce objektu jsou celá čísla:

```
<class> <confidence> <x> <y> <width> <height>
```

Jistota spadá do intervalu $\langle 0, 1 \rangle$. Pro následné vyhodnocení těchto výstupních souborů s predikcemi vůči souborům s *ground-truth* byly v práci využity dva repozitáře, mAP[3] a Object-Detection-Metrics[4], implementující výpočet *precission-recall* křivek, AP jednotlivých tříd a výsledného mAP. Oba tyto repozitáře počítají s IoU alespoň 0.5 (tzn. aby byla predikce považována za správnou, musí se predikovaný box překrývat s *ground-truth* alespoň z půlky, jinak je považována za falešnou), jenž je často používanou hodnotou IoU při vyhodnocení úspěšnosti detektorů.

[3] https://github.com/Cartucho/mAP
[4] https://github.com/rafaelpadilla/Object-Detection-Metrics

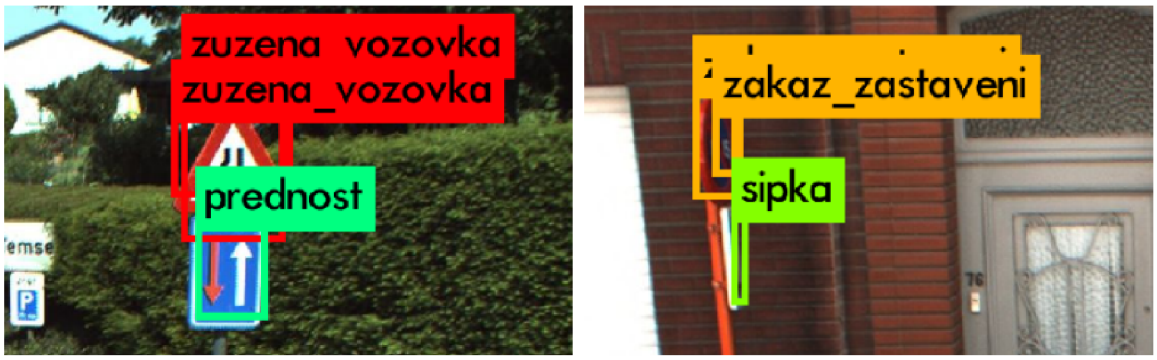
Kapitola 6

Vyhodnocení úspěšnosti vytvořeného detektoru

Vyhodnocení je rozděleno na dvě části – a to sice na první, která se zabývá kvalitativním vyhodnocením detektoru (tedy jak kvalitní detekce systém provádí) a druhou, která udává celkovou úspěšnost systému (kolik z testovaných značek dokáže systém správně detekovat a s jakou chybovostí systém pracuje).

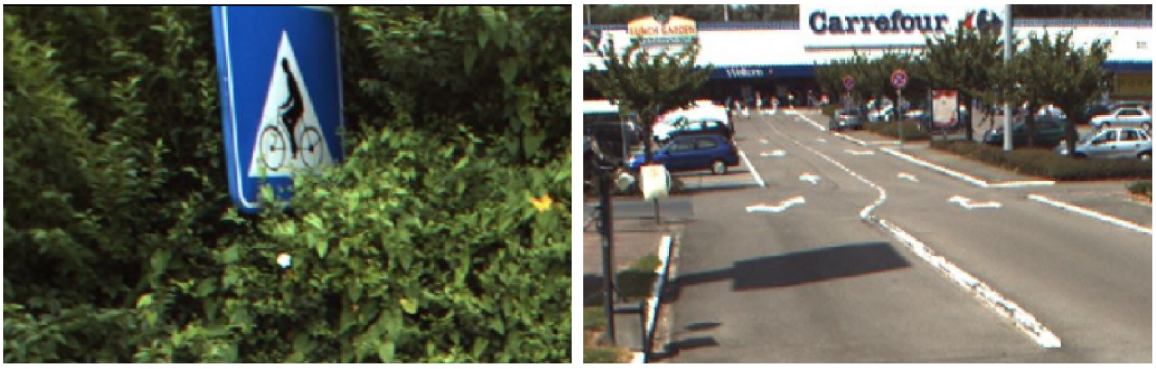
6.1 Kvalitativní vyhodnocení

Pokud je značka dobře viditelná a bez znatelného poškození, model ji ve většině případů správně detekuje s 90-100 % jistotou. Problém modelu dělají značky malé, pod velkým úhlem, poničené či vybledlé, a také občasná záměna pozadí za objekt (pokud má objekt podobný tvar a barvu), jak lze vidět na obrázcích 6.1, 6.2, 6.3 a 6.4. Selhání potlačení nemaximálních hodnot, tzn. některá ze značek je detekována vícekrát, je nejméně často se vyskytující chybou.

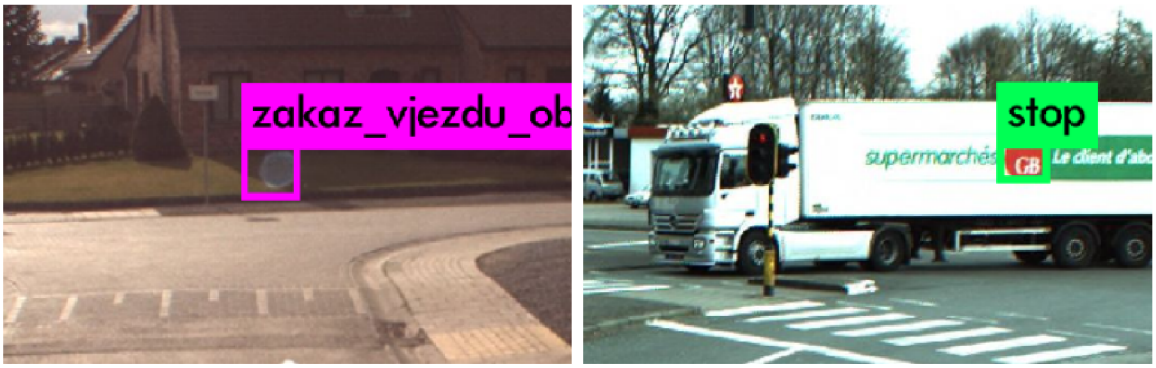


Obrázek 6.1: Značky jsou správně detekovány i klasifikovány, ovšem některé více-násobně. To značí selhání potlačení ne-maximálních hodnot.

Obrázek 6.2: Nesprávně klasifikované dopravní značky.



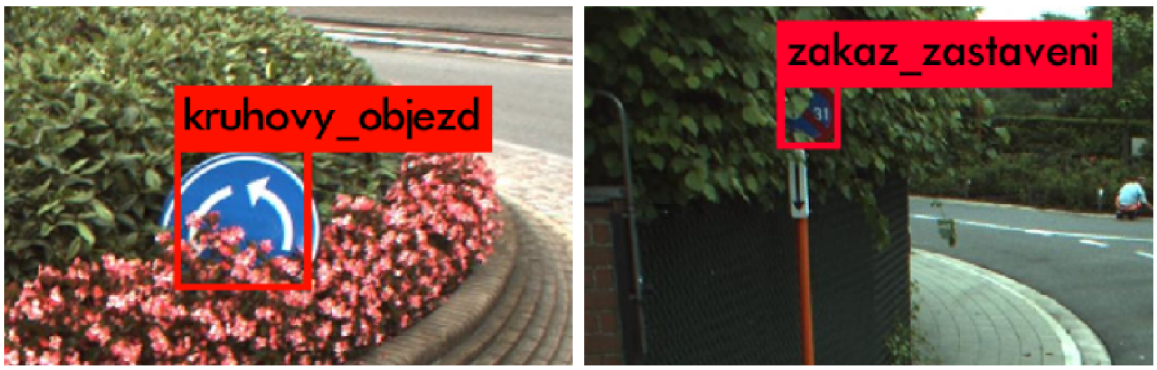
Obrázek 6.3: Značky, které nebyly detekovány.



Obrázek 6.4: Záměna pozadí za detekované objekty.

I přes to, že má YOLO problémy s dokonalou lokalizací objektů, predikované boxy v mnoha případech ohraničují celou značku a to poměrně přesně. Příklad několika správných detekcí za zhoršených podmínek lze vidět na obrázcích 6.5, 6.6, 6.7 a 6.8.

Obrázek 6.5: Správné detekce i za zhoršených světelných podmínek.



Obrázek 6.6: Správné detekce i při částečném překrytí.

Obrázek 6.7: Správné detekce i přes velmi malou velikost dopravních značek.

Obrázek 6.8: Správné detekce i přes nízkou kvalitu či poškození dopravních značek.

6.2 Kvantitativní vyhodnocení

Ke kvantitativnímu vyhodnocení úspěšnosti byl využit zejména repozitář mAP[1], který byl v rámci této práce doplněn o vykreslení ROC (*Receiver operating characteristic*) křivky a analýzu chyb.

Výsledky při použití reálné datové sady

Při použití reálné datové sady pro trénování je možné postupovat tak, že se datová sada rozdělí na dvě části. První část jsou data, na kterých bude detektor/klasifikátor trénován a skládá se z trénovací a validační sady. Druhou částí jsou pak data, na kterých se vyhodnotí úspěšnost natrénovaného modelu a tato data jsou běžně označována jako testovací. Pro vyhodnocení není potřeba tolik dat, jako pro trénování a validaci, proto byla tato část menší, a to přesně v poměru 2:1. Belgická datová sada se skládá z cca. 9000 snímků (nepočítaje negativní datovou sadu), takže pro trénování bylo využíváno $\sim$ 6000 a pro testování $\sim$ 3000 snímků. Dosažené výsledky se pohybovaly kolem $\sim$ 60 % mAP, což je téměř dvakrát více, než se podařilo dosáhnout v práci [4] používající YOLO první verze. K nárůstu úspěšnosti přispěla zejména modifikace třetí verze YOLO, razantně zlepšující detekci malých objektů.

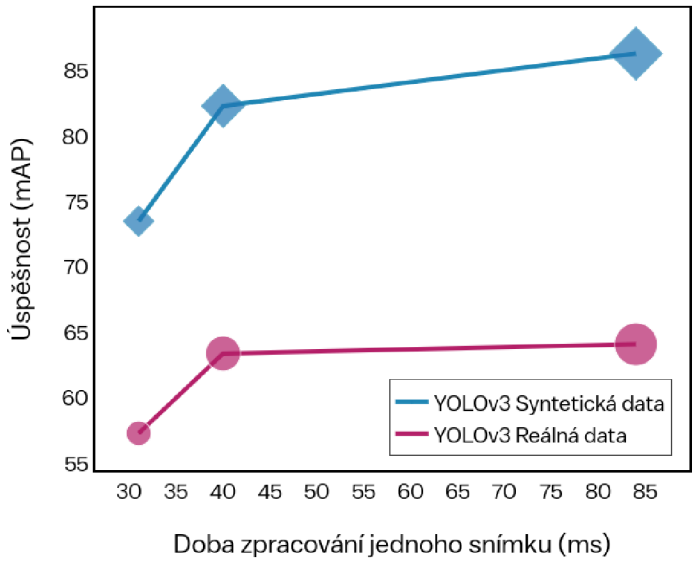
Výsledky při použití syntetické datové sady

Při trénování na syntetických datech bylo možné využít veškeré snímky z Belgické datové sady k testování. Jako nejoptimálnější počet generovaných dat podle tabulky 5.1 vyplývá 1000 snímků na třídu. Výsledky se pohybovaly kolem $\sim$ 80 % mAP. Lepší úspěšnosti dosahoval způsob generování s transparentními značkami (o několik bodů mAP lepší oproti generování s ořezanými značkami). Zlepšení o dva body mAP bylo dosaženo pomocí přimíchání malého množství reálných snímků do syntetické datové sady.

Porovnání výsledků

Porovnání výsledků modelů trénovaných na reálných a syntetických datech při použití různých velikostí vstupní vrstvy lze vidět na obrázku 6.9.

[1] https://github.com/Cartucho/mAP



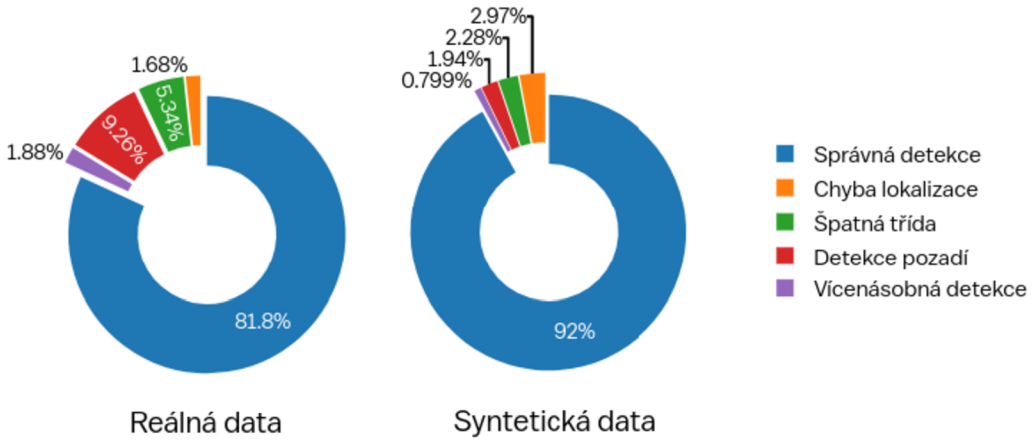
Obrázek 6.9: Porovnání závislosti úspěšnosti na době zpracování jednoho snímku třech modelů (zleva) YOLOv3-320, YOLOv3-416 a YOLOv3-608 trénovaných na syntetických a reálných datech. Zvětšující se velikost bodů reprezentuje velikosti vstupní vrstvy modelů. Z grafu lze vidět, že čím větší vstupní vrstva se použije, tím lepších výsledků detekce je dosaženo, ale také se tím prodlouží doba detekce. Za nejvhodnější lze tedy považovat prostřední model YOLOv3-416, který poskytuje nejlepší kompromis (tzv. *tradeoff*) mezi úspěšností a časem. Výsledky byly získány na GeForce 840M.

6.3 Analýza chyb

Výsledné modely měly poměrně vysoký počet falešně pozitivních predikcí, proto bylo vhodné zjistit, co dělá systému problém. Poté bylo možné navrhnout řešení a dosahovat tak lepších výsledků. Použita byla metodika analýzy chyb popsaná v práci [15]. Predikce je buď to správná, nebo je zařazena do jedné z následujících tříd [29].

- **Správná predikce** – správná třída, IoU > 0.5.
- **Chyba lokalizace** – správná třída, IoU $\in (0.1, 0.5)$.
- **Podobné** – podobná třída, IoU > 0.1.
- **Ostatní** – nesprávná třída, IoU > 0.1.
- **Detekce pozadí** – jakákoli detekce s IoU < 0.1.

Více-násobná detekce jednoho objektu se dá také považovat za chybu (selhání potlačení nemaximálních hodnot), a proto bylo vyhodnocení doplněno o třídu vícenásobné detekce. Naopak kvůli nedostatku času nejsou detekce nesprávných tříd rozděleny do kategorií *podobné* a *ostatní*. Vizualizovanou distribuci chyb obou modelů lze vidět na obrázku 6.10.



Obrázek 6.10: Analýza chyb dvou modelů, trénovaných na reálných a syntetických datech. Model trénovaný na reálných datech (**vlevo**) má větší problém se záměnou pozadí s objekty a také nesprávné určování tříd značek. To je zapříčiněno nedostatečným množstvím informací o některých třídách (malou velikostí datové sady). Naopak má nižší chybovost lokalizace, což znamená že dokáže predikované boxy lépe zarovnat na detekovanou značku. Pokles v úspěšnosti mAP zapříčiňují také falešně negativní vzorky, které do grafu nejsou zahrnuty.

Kapitola 7

Závěr

Práce měla za úkol použít moderní architekturu konvoluční neuronové sítě pro detekci dopravních značek za účelem získání dobrých výsledků detekce v kombinaci s co možná nejkratší dobou zpracování. Dále bylo cílem vytvořit generátor syntetických datových sad a provést porovnání modelů trénovaných na reálných a syntetických datech a zjistit, zda jsou syntetická data pro podobné účely vhodná.

Nejlepším výsledkem získaným při trénování modelu na reálných datech bylo **63.4** % mAP. Použití syntetických dat přineslo podstatně lepší úspěšnost detekce **82.3** % mAP. Oběma modelům trvá vyhodnocení jednoho snímků v průměru **40.4** ms na průměrně výkonném grafickém čipu Nvidia GeForce 840M. Při testování na sdíleném clusteru s nadprůměrně výkonným grafickým čipem Nvidia GTX 1080 Ti byla výsledná doba v průměru **3.9** ms a na CPU Intel i5 bylo dosaženo průměrné rychlosti **1263** ms.

Přínosem této práce je skutečnost, že ačkoli syntetická data nejsou pro trénování konvolučních sítí lepší než reálná, lze pomocí nich dosáhnout kvalitních výsledků bez nutnosti anotace tisíce snímků. Dále bylo potvrzeno, že YOLO nedosahuje nejlepších výsledků detekce (zejména má problém s malými objekty), ale zato dosahuje velmi vysoké rychlosti zpracování snímků, což ze systému dělá zaslouženého konkurenta *state of the art* metod.

Zúčastnil jsem se s touto prací 5. ročníku studentské konference Excel@FIT. Práce byla oceněna odbornou komisí za „inovativní práci s daty a pečlivou analýzu vedoucí ke zlepšení detektoru značek“.

V budoucnu by bylo možné rozšířit generátor datových sad o 3D analýzu pozadí a dopravních značek. Dále doplnění dalších efektů za účelem dosažení vyšší realističnosti syntetické datové sady a hlubší analýzu trénování konvolučních sítí na syntetická data.

Literatura

[1] Abdi, L.; Abdi, L.: Deep learning traffic sign detection, recognition and augmentation. In *Proceedings of the Symposium on Applied Computing*, 2017, s. 131–136, doi:10.1145/3019612.3019643.

[2] Brkic, K.: An overview of traffic sign detection methods. *Department of Electronics, Microelectronics, Computer and Intelligent Systems Faculty of Electrical Engineering and Computing Unska*, ročník 3, 2010: str. 10000.

[3] Canny, J.: A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 8, č. 6, Nov 1986: s. 679–698, ISSN 0162-8828.

[4] Chung Yu Wang, R. C.-Y.: Traffic sign detection using You Only Look Once framework. *Convolutional Neural Networks for Visual Recognition CS 231n*, 2015.

[5] 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.

[6] David A. Forsyth, J. P.: *Computer Vision: A modern approach*. Prentice Hall, 2002, ISBN 0-130-85198-1, 601–626 s.

[7] Dhaware, C.; Wanjale, K. H.: Survey On Image Classification Methods In Image Processing. Jun 2016, ISSN 2347-8578, s. 246–248.

[8] Duda, R. O.; Hart, P. E.: Use of the Hough transformation to detect lines and curves in pictures. Technická zpráva, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1971.

[9] Dvořák, M.: *Aplikace pro rozpoznání dopravních značek pro Windows Phone 7*. Bakalářská práce, FIT VUT v Brně, 2012.

[10] de la Escalera, A.; Moreno, L. E.; Salichs, M. A.; aj.: Road traffic sign detection and classification. *IEEE Transactions on Industrial Electronics*, ročník 44, č. 6, Dec 1997: s. 848–859, ISSN 0278-0046.

[11] FelEnd: Understanding darknet's yolo.cfg config files. [online], Naposledy navštíveno 14. 4. 2019.
URL https://stackoverflow.com/revisions/50696918/5

[12] Girshick, R. B.: Fast R-CNN. *CoRR*, ročník abs/1504.08083, 2015.

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

[14] Github: Can someone clarify the anchor box concept used in Yolo? [online], Naposledy navštíveno 14. 4. 2019.
URL https://github.com/pjreddie/darknet/issues/568

[15] Hoiem, D.; Chodpathumwan, Y.; Dai, Q.: Diagnosing error in object detectors. *Computer Vision–ECCV 2012*, ročník 7574, 2012: s. 340–353.

[16] Houben, S.; Stallkamp, J.; Salmen, J.; aj.: Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, 1288, 2013.

[17] Hui, J.: mAP (mean Average Precision) for Object Detection. [online], Naposledy navštíveno 24. 4. 2019.
URL https://medium.com/@jonathan_hui/45c121a31173

[18] Hui, J.: Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3. [online], Naposledy navštíveno 24. 4. 2019.
URL https://medium.com/@jonathan_hui/28b1b93e2088

[19] Kathuria, A.: What's new in YOLO v3? [online], Naposledy navštíveno 24. 4. 2019.
URL https://towardsdatascience.com/53fb7d3bfe6b

[20] Lee, H. S.; Kim, K.: Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network. *IEEE Transactions on Intelligent Transportation Systems*, ročník 19, č. 5, May 2018: s. 1652–1663, ISSN 1524-9050, doi:10.1109/TITS.2018.2801560.

[21] Liu, W.; Anguelov, D.; Erhan, D.; aj.: SSD: Single Shot MultiBox Detector. *CoRR*, ročník abs/1512.02325, 2015.

[22] Loy, G.; Barnes, N.: Fast shape-based road sign detection for a driver assistance system. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, ročník 1, Sep 2004, s. 70–75 vol.1.

[23] MetaCentrum: MetaCentrum VO – virtuální organizace pro celou akademickou obec. https://metavo.metacentrum.cz/cs.

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

[25] P. Kamavisdar, S. A., S. Saluja: A Survey on Image Classification Approaches and Techniques. Jan 2013, ISSN 2278-1021, s. 1005–1009.

[26] Patrick Pérez, A. B., Michel Gangnet: Poisson image editing. *ACM Transactions on Graphics (TOG)*, ročník 22, č. 3, July 2003: s. 313–318, doi:10.1145/882262.882269.

[27] Radu Timofte, K. Z.; van Gool, L.: Multi-view traffic sign detection, recognition, and 3D localisation. In *Journal of Machine Vision and Applications (MVA 2011)*, Dec 2011.

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

[29] Redmon, J.; Divvala, S.; Girshick, R.; aj.: You Only Look Once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, s. 779–788.

[30] Redmon, J.; Farhadi, A.: YOLO9000: Better, Faster, Stronger. *CoRR*, ročník abs/1612.08242, 2016.

[31] Redmon, J.; Farhadi, A.: YOLOv3: An Incremental Improvement. *CoRR*, ročník abs/1804.02767, 2018.

[32] Shakhuro, V.; Konushin, A.: Russian traffic sign images dataset. *Computer Optics*, ročník 40, č. 2, 2016: s. 294–300, ISSN 0134-2452, doi:10.18287/2412-6179-2016-40-2-294-300.

[33] Sochor, J.: *Rychlá detekce dopravních značek v obraze*. Bakalářská práce, FIT VUT v Brně, 2011.

[34] Svoboda, T.: *Detekce, lokalizace a rozpoznání dopravních značek*. Diplomová práce, FIT VUT v Brně, 2010.

[35] Uijlings, J. R. R.; van de Sande, K. E. A.; Gevers, T.; aj.: Selective Search for Object Recognition. *International Journal of Computer Vision*, ročník 104, č. 2, 2013: s. 154–171.

[36] Wahyono; Jo, K.: A comparative study of classification methods for traffic signs recognition. In *2014 IEEE International Conference on Industrial Technology (ICIT)*, Feb 2014, s. 614–619, doi:10.1109/ICIT.2014.6895001.

[37] Wang, G.; Ren, G.; Wu, Z.; aj.: A robust, coarse-to-fine traffic sign detection method. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Aug 2013, ISSN 2161-4407, s. 1–5, doi:10.1109/IJCNN.2013.6706812.

[38] Wikipedia: *Dopravní značení v Česku*. https://cs.wikipedia.org/wiki/Dopravn%C3%AD_zna%C4%8Den%C3%AD_v_%C4%8Cesku, 2019, [Online; Naposledy navštíveno 02.02.2019].

[39] Wikipedia: *Seznam dopravních značek v Česku*. https://cs.wikipedia.org/wiki/Seznam_dopravn%C3%ADch_zna%C4%8Dek_v_%C4%8Cesku, 2019, [Online; Naposledy navštíveno 02.02.2019].

[40] Wikipedia: *Traffic sign*. https://en.wikipedia.org/wiki/Traffic_sign, 2019, [Online; Naposledy navštíveno 02.02.2019].

[41] Zuo, Z.; Yu, K.; Zhou, Q.; aj.: Traffic Signs Detection Based on Faster R-CNN. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2017, ISSN 2332-5666, s. 286–288, doi:10.1109/ICDCSW.2017.34.

[42] Ťapuška, T.: *Detekce dopravních značek*. Bakalářská práce, FIT VUT v Brně, 2007.

Příloha A

Obsah přiloženého paměťového média

- `install` – Obsahuje instalační skript knihovny `OpenCV 3.4.2`.
- `src` – Zdrojové kódy generátoru datových sad (`Datagen`) a použité neuronové sítě (`Darknet`).
- `utils` – Skripty a programy (vytvořené či volně dostupné) používané v průběhu práce, zejména ke konverzi formátů anotací a k vyhodnocení.
- `doc` – Programová dokumentace a plakát použitý k prezentaci této práce.
- `README` – Více obsáhlá příručka v anglickém jazyce (ve formátu `.md` – *markdown*, i s přiloženými obrázky).
- `videos` – Video prezentující dosažené výsledky.
- `report` – Adresář obsahující zdrojové kódy tohoto dokumentu ve formátu LaTeX, a také jejich přeloženou verzi ve formátu PDF.

Příloha B

Manuál

Sestavení a použití generátoru datových sad `Datagen` na linuxovém OS:

1. Instalace `OpenCV` pomocí přiloženého instalačního skriptu příkazem `bash install/install-opencv.sh` v kořenovém adresáři projektu.
2. Umístit obrázky pozadí do adresáře `data/backgrounds` (již obsahuje několik příkladů).
3. Umístit snímky značek do adresářů `data/cropped` a `data/transparent`. Každá třída značek má svůj vlastní pod-adresář a v ní soubor `imgClass` obsahující pouze název třídy (již obsahuje několik příkladů).
4. Nastavení konstant v `Makefile` (výběr efektů a způsobu generování).
5. Spuštění kompilace pomocí napsání příkazu `make` v kořenovém adresáři projektu.
6. Po úspěšné kompilaci – spuštění generování pomocí již vytvořených cílů v `Makefile`:
   - V případě ořezaných značek: `make runc`.
   - V případě transparentních značek: `make runt`.
7. Vytvořená datová sada by se měla nacházet v adresáři `out`.

Sestavení a použití neuronové sítě `Darknet` na linuxovém OS:

1. Nastavení kompilačních konstant v `Makefile`. Před-nastaveno tak, jak bylo používáno v této práci.
2. Kompilace pomocí příkazu `make` v kořenovém adresáři projektu.
3. Nastavení konfiguračního souboru `cfg/yolo-train.cfg` a souborů `obj.names` a `cfg/obj.data` (opět před-nastaveno).
4. Vyhodnocení několika snímků s využitím před-trénovaných vah pomocí příkazů:
   - V případě modelu trénovaného na reálná data: `./run_real.sh`.
   - V případě modelu trénovaného na syntetická data: `./run_synt.sh`.
5. Posun na další obrázek (dohromady 6) pomocí libovolné klávesy.

Příloha C

Plakát