VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018 Bc. David Hlavoň

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE A KLASIFIKACE DOPRAVNÍCH PROSTŘEDKŮ V OBRAZE POMOCÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ

DETECTION AND CLASSIFICATION OF ROAD USERS IN AERIAL IMAGERY BASED ON DEEP NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. David Hlavoň

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Jaroslav Rozman, Ph.D.

BRNO 2018

Abstrakt

Práce se zaobírá problematikou neuronových sítí z pohledu úlohy detekce dopravních prostředků v obraze, který byl pořízen z dronu. Jelikož je cílem vytvořit prakticky použitelný detektor, práce jako první popisuje datovou sadu. Dále práce rozebírá několik architektur dopředných neuronových sítí, které byly následně použity při realizaci detektoru. Na architektury neuronových sítí navazují metody tvorby detektoru pomocí naivních metod a současně nejúspěšnějších meta architektur. V druhé části se práce zabývá praktickou realizací detektoru. Výsledkem práce je detektor postavený na meta architektuře Faster R-CNN a neuronové sítí PVA s úspěšností detekce přes 90 % a rychlostí 45 full HD snímků za sekundu.

Abstract

This master's thesis deals with a vehicle detector based on the convolutional neural network and scene captured by drone. Dataset is described at the beginning, because the main aim of this thesis is to create practicly usable detector. Architectures of the forward neural networks which detector was created from are described in the next chapter. Techniques for building a detector based on the naive methods and current the most successful meta architectures follow the neural network architectures. An implementation of the detector is described in the second part of this thesis. The final detector was built on meta architecture Faster R-CNN and PVA neural network on which the detector achieved score over 90 % and 45 full HD frames per seconds.

Klíčová slova

Neuronové sítě, Caffe, NVIDIA, TensorRT, detektor, doprava, faster R-CNN, auto, dopravní prostředek

Keywords

Neural nets, Caffe, NVIDIA, TensorRT, detector, traffic, faster R-CNN, car, vehicle

Citace

HLAVOŇ David. Detekce a klasifikace dopravních prostředků v obraze pomocí hlubokých neuronových sítí. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Detekce a klasifikace dopravních prostředků v obraze pomocí hlubokých neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Rozmana, Ph.D.
Další informace mi poskytla společnost RCE systems s.r.o.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
David Hlavoň
18. 5. 2018

Poděkování

V prvé řadě bych chtěl poděkovat společnosti RCE systems s.r.o. za možnost realizovat tuto práci s využitím jejich infrastruktury, zázemí a know-how, které mi umožnilo vytvořit prakticky použitelný produkt. Dále bych chtěl poděkovat vedoucímu mé práce Ing. Jaroslavu Rozmanovi, Ph.D., za podnětné připomínky v průběhu tvorby práce a celkový přístup k vedení práce.

Obsah

1 Úvod

Cílem této práce je vytvořit prakticky použitelný detektor dopravních prostředků v obraze, který byl pořízen z dronu. Prakticky použitelný znamená s úspešností detekce převyšující 90 %. Problémy s detekcemi objektů ve snímcích, respektive videích pořízených z dronů, vycházejí ze specifických vlastností zachycené scény (podrobněji je datová sada a její problémy rozebrány v kapitole 2). Pro celkový kontext je třeba poznamenat, že výsledný detektor bude funkční jako samostatný celek, avšak tento celek je součástí procesu trackování dopravních prostředků, čemuž odpovídají i datové sady.

V současné době jsou drony velice oblíbené díky jejich nízké ceně, mobilitě a obratnosti oproti helikoptéře, horkovzdušnému balónu nebo letadlu. V kombinaci s robustním detektorem, jakým je detektor postavený na neuronových sítích, lze získat mocný nástroj pro monitorování okolí.

Úlohu detekce objektů lze definovat jako lokalizaci a klasifikaci všech objektů ve scéně, které mají definovanou třídu. Ve scéně objekty zabírají malou oblast a zpravidla je jich mnoho. Obecně všechny objekty ve scéně, patřící do jedné z definovaných tříd, lze nazývat popředím. Disjunktní množinou k popředí je pozadí. Jestliže by pozadí a popředí obsahovalo všechny objekty vyskytující se ve scéně, pak lze hovořit o bezchybném detektoru, který má úspěšnost 100 %. Avšak v praxi se často stává, že objekt z množiny pozadí je zařazen detektorem do množiny popředí, kdy tento jev se nazývá falešně pozitivní detekce. Naopak při zařazení objektu z množiny popředí do množiny pozadí se tento jev nazývá falešně negativní detekce. V kontextu procesu trackování vozidel je větší chybou falešně pozitivní detekce oproti falešně negativní detekci, což je také kromě maximální celkové úspěšnosti cílem výsledného detektoru této práce.

Detektory postavené na neuronových sítích v drtivé většině případů vycházejí z klasifikátoru, který má za úkol zařadit daný objekt do jedné z definovaných tříd popředí. Při tvorbě klasifikátoru je možné zvolit určité architektury neuronových sítí, které jsou blíže popsány v kapitole 3. Tyto architektury, přesněji část, která extrahuje deskriptory, dokáží významně ovlivnit kvalitu výsledného detektoru a obecně platí, čím úspěšnější a robustnější je klasifikátor (extraktor deskriptorů) na daných datech, tím úspěšnější bude i výsledný detektor vycházející z tohoto klasifikátoru [1]. Z klasifikátoru je možné získat detektor několika přístupy. V této práci je jednak popsaný naivní přístup metody Sliding Window neboli metody posuvného okna, která na snímcích z dronu nevede k uspokojivým výsledkům, a dále metody založené na meta architekturách, které jsou výrazně úspěšnější a rychlejší. Přístupy tvorby detektoru jsou popsány v kapitole 4. Důležitou částí pro praktické použití je i rychlost detektoru. Některé architektury klasifikátorů je možné optimalizovat, a tím zvýšit jejich rychlost a snížit spotřebu paměti, jedná se především o architektury obsahující batch (dávka vzorků, které neuronová síť současně zpracovává) normalizace, plně propojené vrstvy a větší bloky typické pro danou architekturu. Existuje také knihovna TensorRT od společnosti NVIDIA, která dokáže provádět velmi dobré optimalizace inference neuronové sítě při zachování úspěšnosti. NVIDIA TensorRT je v této práci použit k dosažení maximální propustnosti detektoru. Více o optimalizacích v kapitole 6.3.

Zbylá část práce je praktického charakteru a popisuje tvorbu výsledného detektoru postaveného na meta architektuře Faster R-CNN. Kompletní detektor, jakožto neuronová síť, využívá framework Caffe. Meta architektura Faster R-CNN byla upravena, přičemž úpravy spočívaly v možnosti práce s vlastní datovou sadou, v částečné optimalizaci a v zavedení nových funkcí především v oblasti učení detektoru. Framework Caffe byl taktéž upraven, a to sadou optimalizačních úprav, které byly získány napříč

komunitou deep learning, a novými vrstvami, které rozšiřují funkčnost Caffe. Podrobnější informace o úpravách Faster R-CNN a Caffe jsou k dispozici v kapitole 5. Dále v kapitole 6.2 je podrobně popsán protokol, kterým byl získán výsledný detektor s úspěšností detekce okolo 90 % a rychlostí 45 full HD snímků za sekundu. V kapitole 6 experimenty s Faster R-CNN graficky zobrazují procentuální úspěšnosti, propustnost a spotřebu paměti detektorů postavených na architekturách VGG16 a PVA. Dále jsou graficky zobrazeny výsledky experimentů s inferenčním nástrojem NVIDIA TensorRT. Úspěšnost detektorů je prezentována jako video, v němž jsou znázorněny anotace, pozitivní a negativní detekce a celková procentuální úspěšnost na daném videu. Tato videa jsou k dispozici v elektronické příloze.

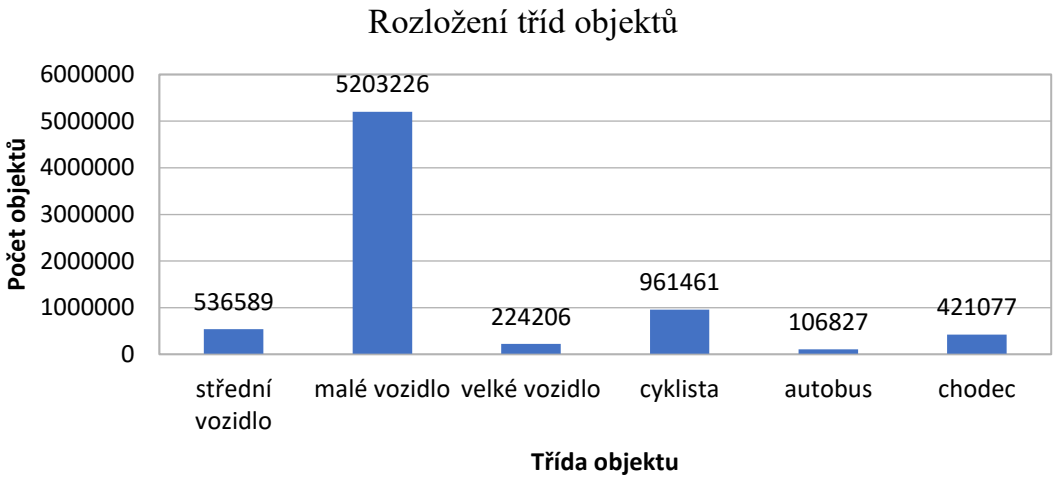
2 Datová sada

Úspěšnost detektoru je silně závislá na datech. Především se jedná o rozměry detekovaných objektů ve scéně, složitost scény a kvalitu obrazových dat. Složitá scéna zachycuje mnoho objektů, které se navzájem překrývají, nebo se ve scéně vyskytuje rozhraní mezi stínem a přímým sluncem, případně jiné povětrnostní podmínky, které zhoršují viditelnost. Proto pro dobrou úspěšnost detektoru je zapotřebí mít co nejlepší datovou sadu. Přístupy, které byly použity pro získání a zkvalitnění datové sady, jsou popsány v následujících podkapitolách.

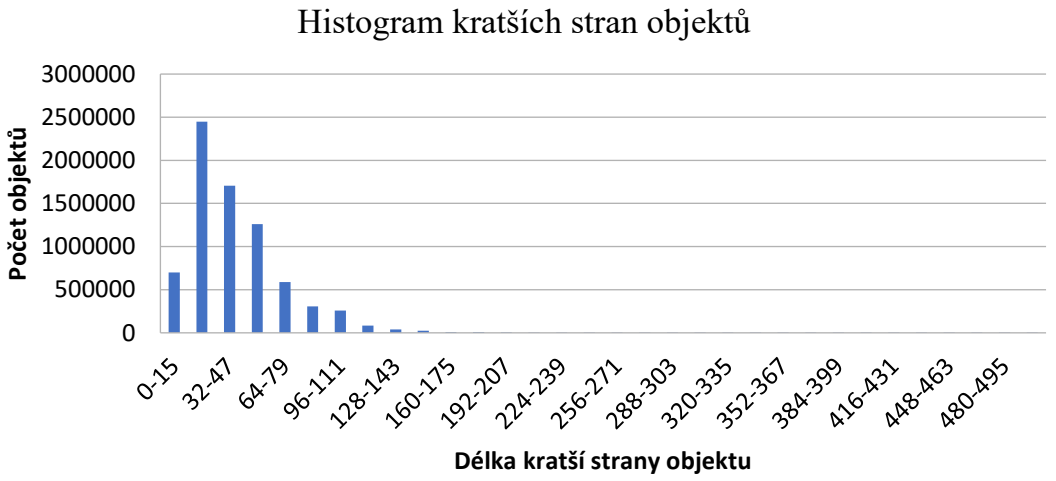
2.1 Popis datové sady

Datovou sadu tvoří snímky z videí pořízených převážně z dronů, výjimečně z balónu nebo statické kamery, kdy každé video bylo natáčeno v jiné lokalitě[1]. Pro natáčení byly použity různé kamery, ale převažují videa natáčená z kamer GoPro a proprietárních DJI kamer. Videa mají také různá rozlišení, kdy převažují videa s rozlišením 3840x2160 (4 K), avšak zastoupení mají také rozlišení 2704x1520 (2,5K), 1920x1080 (full HD) a výjimečně se objevují videa s nižším rozlišením než full HD nebo vyšším než 4 K, a to 4096x2048 (full 4 K). Dalšími vlastnostmi videí jsou, že byla natáčena vždy z jiné výšky a pod jiným úhlem kamery. Minimální výška, ze které byla videa natáčena, je asi 50 m a maximální přibližně 400 m. Interval úhlů kamery je od 30° (statická kamera) do 90° (balón). Příliš malý úhel kamery zvyšuje složitost scény. Při vyšších úhlech kamery s extrémně širokým objektivem (rybí oko) dochází k jevu, kdy objekty za kamerou se jeví otočené o 180° vertikálně, tedy vzhůru nohama, což datovou sadu obohacuje o další unikátní vzorky. Drtivá většina anotovaných objektů spadá do intervalu velikostí 5x5 pixelů až 128x128 pixelů, kdy průměrná velikost objektu je 43x43 pixelů, což lze považovat za malé objekty. Histogram počtu objektů dle jejich kratší strany zobrazuje Graf 2. Anotované objekty jsou zařazeny do jedné ze 6 tříd: malé vozidlo (osobní), střední vozidlo (tranzit), velké vozidlo (TIR), autobus, cyklista a chodec. Celkový počet anotovaných objektů je 7 453 402 na 313 417 snímcích. Histogram rozložení jednotlivých tříd v rámci datové sady zobrazuje Graf 1. Dalším důležitým údajem pro nastavení trénovacích hyper parametrů detektoru je počet objektů popředí v rámci jednotlivých snímků. Bylo zjištěno, že v rámci celé datové sady snímky v drtivé většině případů obsahují 0 až 19 objektů. Kompletní histogram počtu objektů v rámci jednoho snímku zobrazuje Graf 3. Kompletní analýzu datové sady lze najít v elektronické příloze. Znalosti získané z analýzy lze použít pro předpověď možností detektoru, kdy jako příklad lze uvést nemožnost detekce objektů větších než 256x256 pixelů v originálním rozlišení (nic nebrání tomu snímek zmenšit). Takto vzniklá datová sada má sama o sobě charakter více měřítkové datové sady (multi scale), což zvyšuje její kvalitu.

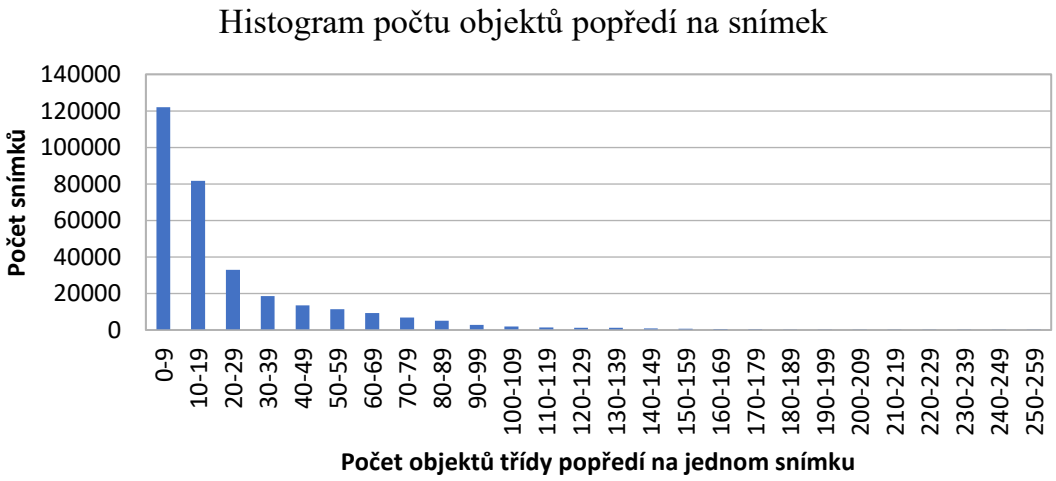
[1] Videa jsou pořízena z Ameriky a Evropy, převážně z Dánska, výjimečně jsou pořízena v jiných zemích.



Graf 1: Histogram zachycující rozložení tříd objektů v datové sadě.



Graf 2: Histogram počtu objektů v datové sadě dle jejich kratší strany. Znalost těchto dat se uplatní při konfiguraci detektoru.



Graf 3: Histogram počtu objektů popředí v rámci jednoho snímku. Znalost těchto dat se uplatní při konfiguraci detektoru.

Získaná datová sada je díky jejím vlastnostem popsaným výše unikátní oproti ostatním datovým sadám používaných k trénování neuronových sítí pro úlohu detekce. Mezi ostatní datové sady patří například Pascal Visual Object Classes (VOC)[2] nebo Common Objects in Context (COCO)[3]. Zásadní rozdíl mezi datovou sadou používanou v této práci a ostatními je právě v pozici kamery, a tedy ve velikosti objektů popředí. Nutno poznamenat, že datová sada z videí z dronů vznikla díky službě DataFromSky společnosti RCE systems s.r.o, a proto bude dále tato datová sada nazývána DataFromSky datová sada. Obrázek 1 zobrazuje ukázku z datové sady Pascal VOC a Obrázek 2 zobrazuje snímek z datové sady DataFromSky. Pokud tyto 2 obrázky porovnáme, je na první pohled vidět zásadní rozdíl ve velikostech objektů popředí. Rozdíl je také v rozlišení snímků, kdy ukázka z Pascal VOC má velikost pouze 500 x 375 pixelů, zatímco ukázka z DataFromSky má 3840 x 2160 pixelů, což i tak nedostatečně kompenzuje rozdíl velikostí objektů popředí. Ukázka z Pascal VOC má jednoduchou scénu (to neznamená, že by Pascal VOC neobsahoval i složité scény), zatímco ukázka z DataFromSky má středně složitou scénu, kde se vyskytuje hodně objektů a částečné překryvy.

Obrázek 1: Ukázka obrázku z datové sady Pascal VOC.

[2] http://host.robots.ox.ac.uk/pascal/VOC/
[3] http://cocodataset.org/

Obrázek 2: Ukázka z datové sady DataFromSky.

Ukázka nejvíce zastoupeného jevu zvyšujícího složitost scény je okluze objektů popředí a pozadí. Typické okluze v datové sadě DataFromSky zobrazuje Obrázek 3. Mimo okluze popředí s pozadím často také nastává okluze popředí s jiným popředím, konkrétně v datové sadě DataFromSky se jedná o okluze mezi malými a velkými vozidly, například osobní automobil a TIR.

Obrázek 3: Obrázek vlevo zobrazuje částečnou okluzi popředí a jiného popředí a také popředí a pozadí, kdy cisterna částečně zakrývá osobní automobil a současně cisternu zakrývá ještě pozadí v podobě keře. Obrázek vpravo zobrazuje silnou okluzi popředí a pozadí.

Celkově je datová sada DataFromSky výzvou pro realizaci úlohy detekce. Hlavní důvod zachycuje Obrázek 4, na kterém jsou zobrazeny detaily nejmenších anotovaných objektů v ukázkových snímcích z datových sad DataFromSky a Pascal VOC. Objekt datové sady DataFromSky náleží do třídy malé vozidlo, kdy anotace má rozměr 29 x 16 pixelů. Objekt datové sady Pascal VOC náleží do třídy osoba, kdy anotace má rozměr 22 x 64 pixelů. Je vidět, že objekt male vozidlo na rozdíl od objektu osoba nemá zřetelné téměř žádné význačné prvky, podle kterých by se dal tento objekt klasifikovat. Proto bude zapotřebí využít při detekci i kontextu objektu.

Obrázek 4: Porovnání nejmenších anotovaných objektů v detailu na ukázkových snímcích z datové sadě DataFromSky třídy malé vozidlo (vlevo) a Pascal VOC třídy osoba (vpravo).

2.2 Tvorba datové sady

Proces tvorby datové sady lze rozdělit do 3 částí: trackování objektů ve videu, export datové sady pro trénování detektoru z trajektorií a zpřesnění anotací. Pro trackování vozidel bylo využito technologií a prostředků, kterými disponuje společnost RCE systems s.r.o. Pro následné získání anotací z trajektorií a zkvalitnění datové sady bylo využito vlastních programů a skriptů.

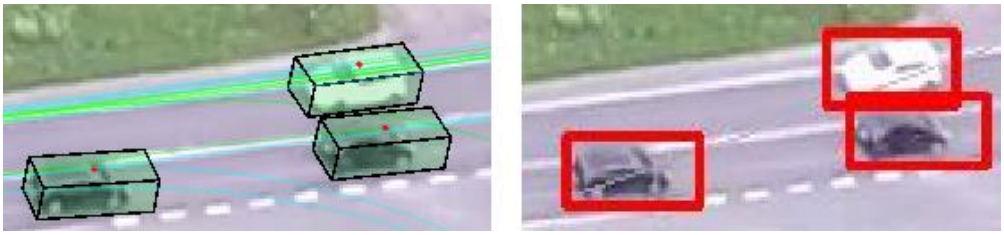
2.2.1 Trackování objektů ve videu a export

Proces anotace datové sady DataFromSky začíná u člověkem definovaných, co nevíce obtažených obdélnících okolo objektů (bounding box) ve snímcích daného videa, čímž vznikne inicializace trackování. Na základě této inicializace je spuštěno trackování těchto objektů. Pokud trackování objektů neprodukuje dostatečně přesné bouding boxy, například vlivem příliš složité scény, je nutné manuálně definovat polohu vozidel ve snímcích. Výsledkem jsou trajektorie jednotlivých objektů v rámci videa, což znamená, že je k dispozici pozice každého anotovaného objektu na každém snímku videa. V průběhu trackování se transformuje iniciační 2D bounding box na 3D bounding box objektu. Mimo jiné vlivem transformace iniciačního 2D bounding boxu na 3D bounding box odtrackovaného objektu a neúplné inicializace trackování vznikají problémy, které jsou blíže rozebrány v kapitole 2.2.3. Po získání trajektorií objektů je vygenerován základ datové sady z trajektorií, který transformuje 3D bounding box objektu zpět na 2D bounding box. Obrázek 5 zobrazuje tuto transformaci.

Generování základu datové sady je parametrizovaný proces, přičemž mezi hlavními parametry jsou:

- Číslo snímku videa, od kterého se data generují.
- Číslo snímku videa, po který se data generují.
- Velikost kroku generování, který značí, kolik snímků vynechat po předchozím generování.
- Maximální počet generovaných snímků v rámci definovaného intervalu.
- Definice tříd objektů, které se mají generovat.

Pro kvalitu výsledného základu datové sady má zásadní vliv parametr maximálního počtu generovaných snímků. Cílem je generovat snímky náhodně v rámci intervalu, avšak s minimálním krokem definovaným parametrem procesu generování. Výsledkem je snížení redundantních anotací, které jsou způsobeny neměnnou scénou, jako jsou stojící vozidla a pozadí scény samotné. Právě díky definici minimálního kroku a maximálního počtu generovaných snímků je možné vygenerovat náhodně snímky napříč celé video oproti statickému kroku generování. I přes toto opatření generování zavádí do základu datové sady redundantní anotace, částečné odstranění redundancí je popsáno v kapitole 2.2.2. Anotace objektů 2D bounding boxu obsahují objekt zarovnaný na střed bounding boxu, čímž se při učení neuronové sítě zajistí lepší skóre detekce na střed vozidla.



Obrázek 5: Zobrazení transformace 3D bounding boxu trackovaných objektů na 2D bounding boxy. Je vidět, že výsledné 2D bounding boxy jsou nepřesné.

2.2.2 Odstranění redundance

Redundance anotací je negativním dopadem automatického generování základu datové sady z trajektorií. Za redundanci lze považovat anotace stejného objektu, které jsou si vizuálně velice podobné, což znamená, že velikost objektu ani jeho natočení se téměř neliší, objekt je téměř na stejné pozici v rámci snímku a ve scéně nedošlo k žádné změně, jakou je např. změna povětrnostních podmínek ovlivňující viditelnost, případně okluze s jiným objektem. Obrázek 6 zobrazuje ukázku redundance anotovaných dat.

Obrázek 6: Ukázka redundance vyskytující se v datové sadě.

Důsledkem těchto redundancí je degradace kvality výsledné datové sady, jelikož právě redundance budou způsobovat větší náchylnost neuronové sítě k přetrénování (overfitting). Lze také říci, že redundance v datové sadě snižují unikátní počet vzorků datové sady.

Pro snížení redundance anotací byl použit postup, který provádí augmentaci, čili úpravu dat po rozpoznání redundance, k čemuž je využito dodatečných informací z trackeru, odkud data původně pocházejí. Před samotným hledáním redundancí se anotace seřadí podle čísla snímku, na kterém se nacházejí. Pro rozpoznání redundance jsou použity parametry plocha anotace a směr objektu, vyjádřený jako úhel vektoru rychlosti objektu. Jelikož byl základ datové sady generován s dostatečně malý časovým krokem, není potřeba uvažovat polohu anotací.

Uvažujme plochu H anotace objektu k na snímku i a plochu anotace stejného objektu na snímku $i + 1$.

$$H_{k,i} = ymax_{k,i} - ymin_{k,i}\ ;\ W_{k,i} = xmax_{k,i} - xmin_{k,i} \quad (1)$$

$$S_{k,i} = H_{k,i} * W_{k,i} \quad (2)$$

$$H_{k,i+1} = ymax_{k,i+1} - ymin_{k,i+1}\ , W_{k,i+1} = xmax_{k,i+1} - xmin_{k,i+1} \quad (3)$$

$$S_{k,i+1} = H_{k,i+1} * W_{k,i+1} \quad (4)$$

Aby byly 2 po sobě jdoucí anotace stejného objektu uznány jako redundantní, stačí, aby byl splněn jeden z predikátů:

- poměr plochy objektu na aktuálním snímku vůči ploše objektu na následujícím snímku je nižší než 10 % (5),
- změna úhlu vektoru rychlosti je menší než 0.2 radiánu (6)

$$P\left(S_{k,i}, S_{k,i+1}\right)\begin{cases}\frac{S_{k,i}}{S_{k,i+1}} < 0.1 & pro\ S_{k,i} < S_{k,i+1} \\ \frac{S_{k,i+1}}{S_{k,i}} < 0.1 & pro\ S_{k,i+1} < S_{k,i}\end{cases} \quad (5)$$

$$\Delta Angle\left(Angle_{k,i}, Angle_{k,i+1}\right) = \left|Angle_{k,i} - Angle_{k,i+1}\right| < 0.2 \quad (6)$$

Pro augmentaci byla náhodně použita jedna z následujících metod:

- Posuv světlosti (Value)
  - Tato metoda má za úkol simulovat den a zataženo, případně večer. Také simuluje nevhodné vyvážení bílé u kamery.
- Posuv odstínu (Hue)
  - Tato metoda je za určitých podmínek schopna kompletně změnit barvu objektu na jinou.
- Gaussovský šum (Gaussian noise)
  - Tato metoda byla použita pouze pro světlé denní snímky, jelikož při nočním osvětlení nebo příliš tmavých snímcích tato metoda generovala dosti výrazný šum.

- Posuv teploty (Temperature)
    - Tato metoda má za úkol simulovat slunečné teplé dny a chladné dny.
- Rozmazání (Blur)
    - Velikost jádra byla zvolena na 5x5 pixelů.
    - Tato metoda augmentace byla použita pouze v případě, že velikost objektu byla větší než 40x40 pixelů, aby nedošlo k příliš velkému potlačení charakteristických znaků objektu.
    - Tato metoda má za úkol simulovat vzdálené objekty a nedokonalosti kamery.

Aby bylo zamezeno ostrým hranám, na které jsou neuronové sítě citlivé, bylo využito gradientu alfa kanálu kolem augmentované oblasti. Obrázek 7 zobrazuje odstranění redundance s využitím jedné z metod augmentace.

Obrázek 7: Ukázka odstranění redundantní anotace s využitím augmentace. V tomto případě se uplatnil posuv světlosti.

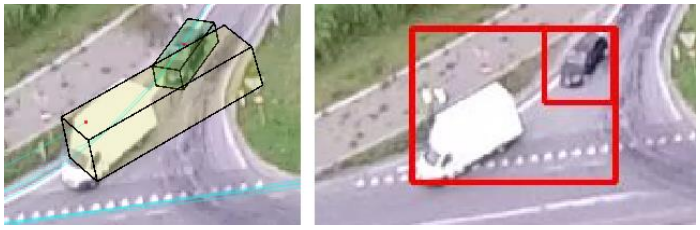
2.2.3 Problémy anotovaných dat

I když metoda odstraňující redundanci anotací tento problém značně snížila, tak stále přetrvává. Mimo redundanci dále jsou tu i problémy anotovaných dat, které není jednoduché vyřešit v kontextu velké datové sady z důvodu enormní časové náročnosti. Pokusy o opravu datové sady skončily ve většině případů neúspěchem z důvodů nedůslednosti člověka a zaváděním dalších chyb do datové sady. Nutno však podotknout, že za 1 měsíc zpřesňování vygenerovaných anotací bylo získáno 84 317 velice kvalitních anotací, které posloužily jako zpřesňující datová sada pro zlepšení úspěšnosti detektoru v závěru procesu trénování. Proto pokud nebude uvedeno jinak, byl problém ponechám bez řešení a bylo spoléháno na generalizační schopnosti neuronové sítě, což se ukázalo jako fungující řešení s uspokojivými výsledky. U datové sady se vyskytují následující problémy:

Problém chybné anotace třídy objektu

U některých objektů je složité určit jejich třídu, především se jedná o to, kdy zařadit objekt do třídy střední vozidlo a kdy do třídy velké vozidlo. Tento problém vychází již z trackeru, ze kterého data pocházejí. Ten totiž definuje 7 tříd a podle zvolené třídy generuje generický 3D bounding box. Problém nastane při generování 2D bounding boxů. Obrázek 8 zobrazuje důsledek problému anotace objektu

chybnou třídou. Tento problém je praktiky neřešitelný, jelikož by vyžadoval kompletní revizi datové sady člověkem.



Obrázek 8: Ukázka problému chybné anotace třídy a jejího důsledku u anotovaných dat. Střední vozidlo bylo označeno jako velké vozidlo, a proto byl vygenerován nesprávný bounding box.

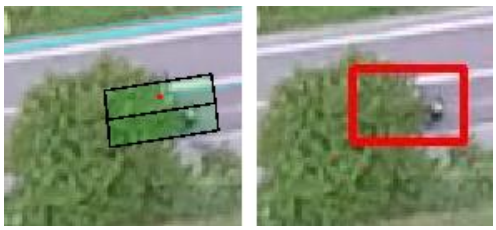
Problém chybné velikosti 2D bounding boxu

I v případě správně určené třídy objektu se u velkých vozidel často stává, že jejich generovaný bounding box není ideální. Ve většině případů je menší než samotný objekt, což v důsledku vede k chybným trénovacím vzorkům. Byly snahy tento problém vyřešit pomocí lidské síly, kdy ze snímků byly vyřezány objekty podle anotací a člověk měl za úkol ponechat pouze ty výřezy, na kterých byl celý objekt. Avšak tento přístup nevedl k uspokojivým výsledkům, naopak bylo zaváděno ještě více chyb.

Obrázek 9: Ukázka chybné velikosti 2D bounding boxu anotace, i když byla zvolena správná třída objektu.

Anotace okluze

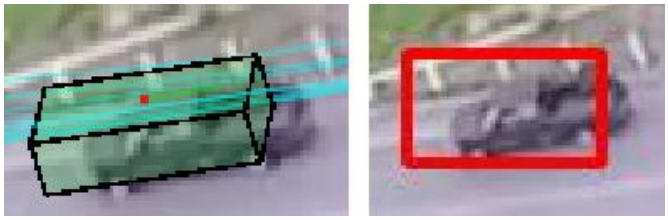
Tento problém vychází z automatického generování vzorků z trackeru, který uchovává trajektorie objektů. Trajektorie je definována i za okluzí, a proto jsou vygenerována data i těchto míst. Obrázek 10 zobrazuje chybu, kde anotací objektu je okluze. Tento problém se podařilo vyřešit zavedením regionů zájmů do trackeru, které definovaly oblasti, pouze ze kterých se mají anotace generovat.



Obrázek 10: Ukázka anotace okluze vzniklé z vygenerování dat podle trajektorie objektu, která je definována i za okluzí.

Nepřesnost anotovaného 2D bounding boxu při inicializaci trackování

Tato chyba se projevuje posunutím bounding boxu do takové pozice, kdy zabírá pouze část objektu. Je to způsobeno chybnou anotací objektu člověkem již při inicializaci trackování. Konkrétně u vozidel je třeba kliknout na střed střechy vozidla, čímž se vytvoří inicializace pro tracker. Obrázek 11 zobrazuje výsledek nepřesné inicializace trackování objektu. Tento problém je neřešitelný, aniž by se provedlo celé trackování znovu s přesnou počáteční inicializací nebo ruční opravou vzorků.



Obrázek 11: Ukázka nepřesné inicializace trackování, která se projeví v datové sadě posunutým bounding boxem. Červená tečka vyjadřuje klik člověka. Je vidět, že tečka není na středu střechy vozidla.

Neanotované objekty

Jelikož při inicializaci trackování není cílem trackovat úplně všechny objekty, tak při generování dat vznikají oblasti, ve kterých nejsou objekty anotovány, a tím jsou přiřazeny do třídy pozadí, i když objekt do třídy pozadí nepatří. V důsledku by to mělo vliv na učení neuronové sítě, která by měla nekonzistentní data, pokud by jednou vzorek byl přiřazen do třídy popředí a podruhé do třídy pozadí. Tento problém lze vyřešit s využitím regionů zájmů u trackeru, které definují oblasti, kde se vyskytují neanotované objekty popředí, jako jsou parkoviště, a tyto regiony při generování rozmaže, aby nebyly patrné charakteristické znaky objektů. Obrázek 12 zobrazuje aplikaci regionu zájmu a rozmazání oblasti, kde se vyskytují neanotované objekty popředí.

Obrázek 12: Ukázka řešení problémů neanotovaných dat popředí.

3 Architektury extraktorů konvolučních deskriptorů

V této kapitole jsou rozebrány různé architektury konvolučních[4] neuronových sítí, které lze využít pro stavbu klasifikátoru. V kontextu detektoru se jedná o extraktory deskriptorů, podle kterých se rozlišuje popředí, pozadí a následně třída popředí. Na internetu lze najít mnoho stránek, které měří úspešnosti neuronových sítí postavených na různých architekturách, například GitHub repozitář[5] uživatele soeaver, kde jsou pravidelně aktualizovány výsledky měření. Vizualizace všech sítí zmíněných v této kapitole lze najít v elektronické příloze.

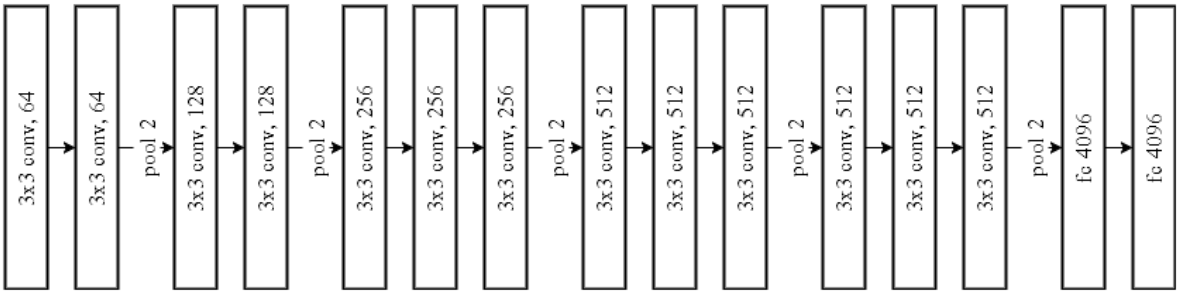
3.1 Lineární konvoluce

Architektura lineárních konvolucí je nejjednodušší a také nejstarší zde uvedenou architekturou. Její název vyjadřuje, že výstup předchozí vrstvy je vždy vstupem jedné bezprostředně následující vrstvy. Není to tedy tak, že by sítě postavené na této architektuře nedokázaly modelovat nelinearity, jak může název architektury klamat. Nejúspěšnější neuronová síť postavená na této architektuře je VGG16 [2] případně VGG19, které v soutěži ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2014 skončily 2. v kategorii klasifikace. Dle jejich názvu lze usoudit, že VGG16 je tvořena 16 vrstvami a VGG19 dokonce 19 vrstvami. Skutečnost, že bylo možné takto hluboké neuronové sítě trénovat, aniž by se výrazně přetrénovaly, bylo dosaženo použitím malých (3 x 3) konvolučních filtrů napříč celou sítí. Dále tyto sítě uplatňovaly princip zvyšování počtu kanálů vektorů deskriptorů na dvojnásobek, vždy po zvětšení receptivního pole sítě, čili oblast vstupních dat, kterou síť dokáže brát v potaz. Tyto sítě ukázaly, že hloubka neuronových sítí hraje významnou roli v jejich úspěšnosti. Zásadní nevýhodou těchto sítí je příliš vysoký počet učících se parametrů (144 miliónů), což vede k rychlému přetrénování. Rychlost těchto sítí je nízká a paměťová náročnost vysoká z důvodu opakovaného výpočtu všech vektorů deskriptorů stále dokola napříč sítí. I přes tyto nevýhody lze síť VGG16 úspěšně aplikovat. Důkazem je mnoho jiných prací (1457 citací), které základy VGG16 využívají nebo se touto sítí inspirují.

V rámci této práce bylo experimentováno se dvěma zástupci architektury lineárních konvolucí, a to VGG-CNN, což je redukovaná VGG16, a se samotnou VGG16. Nebylo však dosaženo uspokojivých výsledků z důvodu nízké schopnosti sítí reagovat na široký interval velikostí objektů a také z důvodu nedostatečné generalizace a celkové neschopnosti pojmout komplexnost scény datové

[4] Konvoluční neuronové sítě jsou dnes považovány za standardní a poměrně dobře známé, proto se jejich podstatou tato práce nezaobírá.
[5] https://github.com/soeaver/caffe-model

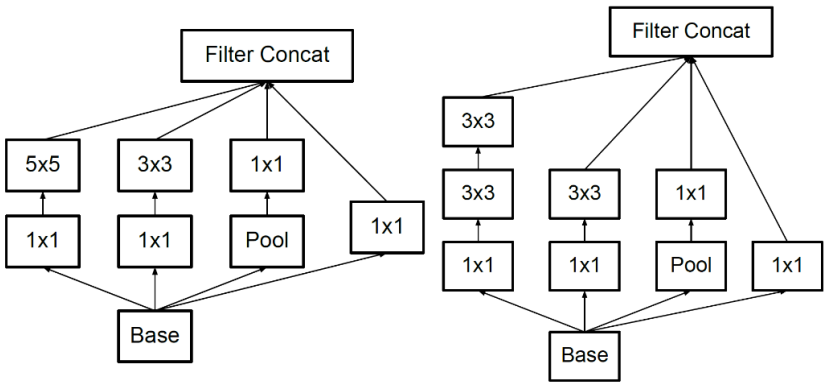
sady DataFromSky, kdy očekávaně VGG-CNN dávala horší výsledky než VGG16. Obrázek 13 zobrazuje schéma neuronové sítě VGG16.



Obrázek 13: Schéma sítě VGG16, jakožto zástupce architektury lineárních konvolucí.

3.2 Inception

Architektura Inception [3] byla navržena výzkumným týmem společnosti Google a jedná se v podstatě o neuronovou síť v neuronové síti. Hlavní motivací pro vytvoření této architektury bylo zvýšit úspešnost vzniklé neuronové sítě, avšak se zachováním rychlosti a bez nutnosti přidávat velké množství učících se parametrů, a tím se vyhnout přeučení. Neuronové sítě postavené na Inception architektuře jsou složeny ze stejnojmenných bloků. Obrázek 14 zobrazuje 2 Inception bloky: originální a vylepšený z hlediska výpočetní náročnosti.



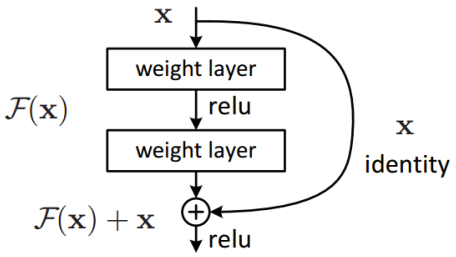
Obrázek 14: Inception blok [4], tak jak byl originálně navržen (vlevo). Vylepšený Inception blok, kde na místo konvoluce 5x5 jsou 2 konvoluce 3x3 (vpravo).

Hlavní výhodou této architektury je její schopnost paralelně pracovat s více různě velkými receptivními poli v rámci jednoho bloku, což zajištují paralelní konvoluční vrstvy s různou velikostí jádra konvoluce. Konvoluce 1x1 jsou umístěny v bloku pro redukci šířky vektoru deskriptorů. Možností práce s více receptivními poli je zlepšena kvalita extrakce konvolučních deskriptorů pro různě velké objekty. Další výhodou je to, že neuronová síť dokáže brát v potaz kontext objektu. Zástupcem této architektury

je neuronová síť pojmenovaná GoogLeNet, která má 22 vrstev. V soutěži ILSVRC 2014 prorazila VGG v úloze klasifikace, i když GoogLeNet má pouze 6,8 milionů učících se parametrů.

3.3 Reziduální

Residuální architektury [5] přinesly možnost tvorby velice hlubokých neuronových sítí, kdy experimentálně byla trénovaná i neuronová sít s 1001 vrstvami [6]. Návrh residuální architektury byl inspirován fenoménem degradace problému, který se objevoval u sítí, které zvyšovaly svoji hloubku za účelem zvýšit úspešnost sítě. Fenomén spočíval v menší úspešnosti hluboké sítě jak při trénování, tak při testování oproti sítím s nižším počtem vrstev. Dle předpokladu, který stanovily sítě VGG, se se zvyšujícím se počtem vrstev měla zlepšit i úspěšnost sítě. Samotný princip trénování reziduálních sítí je postaven na tom, aby celá síť nebo její části aproximovaly reziduální funkce na místo aproximace obecně neznámé funkce. Existuje ne zcela vyřešená hypotéza [7], která tvrdí, že pomocí několika nelineárních vrstev lze asymptoticky aproximovat složité funkce. Tedy jestliže $\Theta(x)$ je aproximací složité funkce pomocí několika nelineárních vrstev pro vstup x, pak ekvivalentním tvrzením k hypotéze je, že reziduální funkce $\Gamma(x) = \Theta(x) + x$ může být také aproximací složité funkce pomocí několika nelineárních vrstev neuronové sítě [5]. Tímto vzniklá předpoklad, že reziduální spojení v neuronové síti je funkčně ekvivalentní s tzv. lineární architekturou (posloupnost nelineárních vrstev). Obrázek 15 zobrazuje základní stavební blok reziduální sítě realizující reziduální funkci.



Obrázek 15: Základní stavební blok reziduální sítě [5].

Sítě postavené na reziduální architektuře jsou úspěšnější, než sítě postavené na lineární architektuře. Residuální sítě nejsou tolik náchylné k přetrénovaní, jelikož mají mnohem méně učících se parametrů oproti lineární architektuře. Například ResNet20, tedy 20 vrstvá síť, má pouze 0,27 milionů parametrů. Nízký počet učících se parametrů je možný díky residuálním spojením, která přenášejí data z předchozích vrstev. Lze říci, že výstupy z residuálního bloku jsou ovlivněny jeho vstupem. Tento fakt také zvyšuje rychlost sítě neboli výpočetní náročnost dopředného průchodu, jelikož se využívá již předpočítaných vektorů deskriptorů ve vyšších vrstvách sítě, a není proto potřeba definovat tolik konvolučních jader. Sítě postavené na této architektuře zvítězily v soutěži ILSVRC 2015 v kategorii

klasifikace. ResNet-152 dosáhla skóre 19,38 oproti tomu VGG16 skóre 24,4 pro TOP-1 chybu na testovací datové sadě ImageNet.

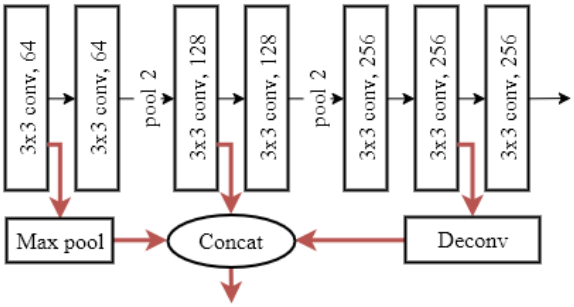
V rámci této práce bylo experimentováno s reziduálními sítěmi ResNet18, ResNet32 a ResNet50, kdy s přibývajícím počtem vrstev rostla i úspěšnost sítě, avšak neúměrně rostl i nárok na výpočetní výkon, a hlavně grafickou paměť výpočetního stroje.

3.4 Hybridní

Hybridní architektura kombinuje více architektur a přístupů. Zástupcem této architektury je PVA net [8], která je z pohledu této práce velice zajímavá díky tomu, že byla primárně navržena pro realizaci extraktoru deskriptorů pro detektor. V rámci této práce byla PVA net použita s velice dobrými výsledky. Tato neuronová síť je rozdělena na 2 části: reziduální a inception s vylepšenými bloky, viz Obrázek 14. I v rámci inception části jsou zavedeny reziduální spoje. Dále je v rámci této sítě využito několik technik:

Hyper Feature

Jelikož byla PVA net navržena primárně pro realizaci detektoru, využívá principu Hyper Feature [9]. Jedná se o techniku, kterou lze aplikovat na libovolnou neuronovou síť a která se používá především při potřebě zvýšit rozlišení konvolučních deskriptorů neuronové sítě. Princip spočívá ve vytvoření výstupního datového toku konkatenací vektorů konvolučních deskriptorů vzniklých v rámci různých receptivních polí neuronové sítě. Podmínkou je, aby velikosti konvolučních deskriptorů byly stejné v rámci všech konkatenovaných toků, což lze zajistit pomocí dodatečných vrstev umístěných před konkatenaci vektorů deskriptorů. Téměř vždy jsou spojovány 3 toky, kdy první tok je z místa sítě s nejmenším receptivním polem a před spojením s ostatními toky je na něj aplikován operátor Max pooling se správnou velikostí jádra, aby velikost konvolučních deskriptorů po aplikaci vrstvy byla stejná jako velikost konvolučních deskriptorů druhého toku. Druhý tok je zvolen z místa neuronové sítě, kde receptivní pole je větší než v místě, odkud pochází první tok. Druhý tok zůstává nezměněn. Třetí tok je volen z místa neuronové sítě, kde je receptivní pole větší než v místě, odkud pochází druhý tok. Na třetí tok je aplikována operace dekonvoluce se správnou velikostí jádra a ohraničení, aby velikost konvolučních deskriptorů po aplikaci dekonvoluce byla stejná, jako velikost konvolučních deskriptorů druhého toku. Výstupní tok po konkatenaci je složen z vektoru deskriptorů, které odpovídají různým měřítkům vstupního obrazu. Obrázek 16 znázorňuje princip Hyper Feature. Konktrétně toky pro Hyper Feature u PVA net jsou voleny z konce reziduální části sítě, z inception části, kde je proveden reziduální spoj a z konce inception části.



Obrázek 16: Diagram znázorňující princip Hyper Feature. Červené spojnice znázorňují první, druhý a třetí tok, které se konkatenují do výsledného toku.

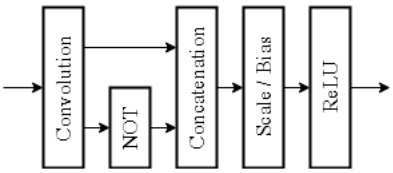
Concatenated ReLU

Další technikou použitou v rámci sítě PVA net je Concatenated Relu (CReLU) [10]. Na základě pozorování bylo zjištěno, že ve vyšších vrstvách konvolučních neuronových sítí jsou konvoluční deskriptory párovány tak, že pro jeden konvoluční deskriptor existuje jiný s opačnými hodnotami. Při použití aktivační funkce ReLU, která má tvar $f(x) = \max(0, x)$, a aplikace znalostí jevu párování konvolučních deskriptorů, je důsledkem ztráta užitečných informací, jelikož všechny záporné hodnoty se převedou na hodnotu 0, dále se sítí tyto informace budou propagovat s velice slabým vlivem, i když by původní data mohla pomoci k lepší úspešnosti sítě. CReLU tento problém řeší zavedením konkatenace původních a negovaných výstupů z konvoluční vrstvy sítě před aplikací ReLU. Předpis funkce CReLU je zapsán výrazy (7) a (8).

$$CReLU: \mathbb{R} \rightarrow \mathbb{R}^2 \qquad (7)$$

$$\forall x \in \mathbb{R}, CReLU(x) = (max(0, x), max(0, -x)) \qquad (8)$$

Tím dojde k odstranění párování deskriptorů a ke zdvojnásobení počtu kanálů vektoru konvolučních deskriptorů z výstupu konvoluční vrstvy, což je velice výhodné, protože stačí použít jednoduchou operaci změny znaménka a konkatenace vektoru oproti výpočtu konvoluce pro získání dvojnásobku užitečných dat. Obrázek 17 zobrazuje diagram operátorů neuronové sítě pro realizace CReLU. Není použit žádný nestandardní operátor, který by nebyl součástí jakéhokoliv frameworku pro realizaci neuronových sítí, což je další výhodou této techniky.



Obrázek 17: Diagram operátorů neuronové sítě pro realizaci CReLU. Ve frameworku Caffe je operátor NOT nahrazen operátorem Power s parametrem scale = -1.

4 Přístupy k tvorbě detektoru

Tato kapitola popisuje přístupy, kterými je možné dosáhnout úlohy detekce s využitím neuronových sítí. Jedná se o naivní přístupy a o využití frameworků, které jsou v této práci nazývány meta architekturami, jelikož frameworkem je v této práci míněn nástroj, který realizuje činnost samotné neuronové sítě, například Caffe.

4.1 Naivní přístupy

Naivními přístupy jsou označeny ty, které jsou přímočaré a není k jejich realizaci nutný žádný speciální balík programových nástrojů.

4.1.1 Sliding window

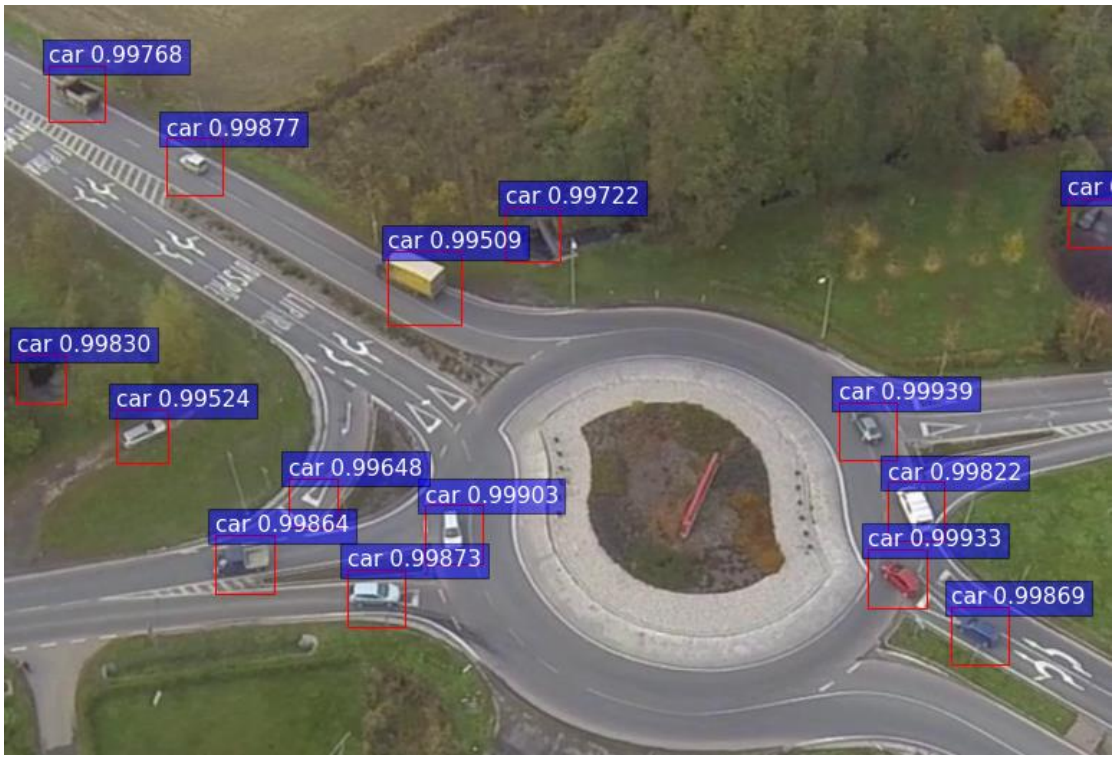
V rámci této práce byla tato metoda použita pro realizaci Prove of Concept (PoC), zda dokáže neuronová síť klasifikovat a lokalizovat objekty v datové sadě DataFromSky. Princip metody je velice jednoduchý, kdy okno statické velikosti přejíždí přes obraz s daným krokem, který definuje velikost posunu okna po obraze. V každém místě okna je pořízen výřez, který je definován pozicí a velikostí okna, a provede se nad ním operace klasifikace pomocí neuronové sítě. V případě, že neuronová síť zařadí výřez obrazu do třídy popředí, uchová se klasifikovaná třída, skóre klasifikace a pozice okna jako bounding box objektu, nebyla použita žádná metoda regrese bounding boxů. Pro možnost detekce různě velkých objektů se okno posunuje nad několika měřítky obrazu. Po klasifikaci všech výřezů se provede nad všemi bounding boxy operace Non Maximum Suppression (NMS) [11], která potlačí bounding boxy s nižším skóre než maximální skóre bounding boxu, který se s ostatními překrývá více než definovaný práh. NSM je standartní metoda, která je využívána téměř u každého přístupu tvorby detektoru.

Metoda sliding window potvrdila, že neuronová síť je schopna klasifikovat a lokalizovat objekty z datové DataFromSky, avšak úspešnost nebyla dostatečná, a proto je zde uvedena pouze jako slepá větev vývoje i s výsledky. Bylo využito neuronové sítě vlastního návrhu a dále GoogLeNet. V rámci realizace detektoru touto metodou bylo využito několik přístupů, které měly zvýšit přesnost a rychlost, která silně závisela na zvoleném kroku sliding window a pohybovala se mezi několika sekundami (velice nízká přesnost) až desítkami minut (vysoká přesnost) pro full HD snímek. V rámci všech přístupů byla použitá síť se vstupem 32 x 32 pixelů a velikost sliding window 64 x 64 pixelů.

Klasifikace

Tento přístup odpovídá popisu sliding window uvedenému výše. Obrázek 18 zobrazuje detekce získané tímto přístupem s velikostí kroku 1, kdy byly získány i falešně pozitivní detekce s vysokým skóre,

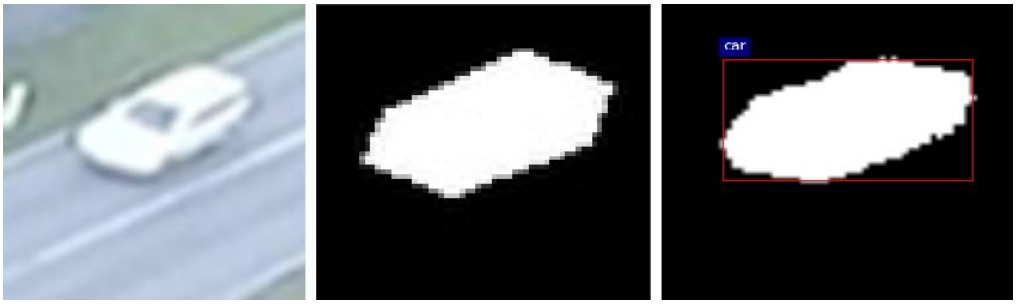
což je větší problém, než falešně negativní detekce, jak bylo definováno dříve. Krok 1 by měl zajistit maximální možnou přesnost, která i tak není dostačující. Čas detekce, který činil 44 minut na jeden snímek, nebyl únosný.



Obrázek 18: Ukázka detekcí objektů získaných pomocí metody sliding window s velikostí okna 64 x 64 pixelů a vlastní neuronové sítě. V modrém obdélníku nad objektem je uvedena přiřazená třída a skóre klasifikace.

Segmentace

Přístup využití segmentace objektu ve výřezu sliding window byl využit pro zavedení regrese bounding boxu objektu, aby bounding box byl co nejvíce obtažený kolem detekovaného objektu. Obrázek 19 zobrazuje výstup tohoto přístupu. Avšak ani tento přístup nepřinesl zlepšení v podobě vyššího počtu detekovaných objektů a menšího počtu falešně pozitivních detekcí, případně zrychlení detekce.



Obrázek 19: Ukázka segmentace využité pro zlepšení přesnosti bounding boxů. Vstupní data (vlevo), anotace pro segmentaci vycházející z 3D bounding boxů trackeru (střed), výsledek segmentace a nalezení výstupního bounding boxu objektu (vpravo).

4.2 Meta architektury

V této kapitole jsou popsány meta architektury, které obalují neuronovou síť a přetvářejí ji na detektor. Na místo framework je použito označení meta architektura z toho důvodu, že dané řešení určitým způsobem ovlivňuje samotnou architekturu neuronové sítě. Výsledný detektor této práce je postaven na meta architektuře Faster R-CNN, která bude v této kapitole podrobně rozebrána. Článek, zabývající se měřením výkonosti a úspešnosti meta architektur [1], popsaných v této kapitole, byl zásadním při výběru meta architektury pro realizaci práce.

4.2.1 R-CNN

Meta architektury Region-based Convolutional Network (R-CNN) pracují na základě klasifikace regionů zájmu a regrese bounding boxů namísto sliding window, kdy metoda získávání regionů závisí na verzi R-CNN. Doposud byly publikovány 4 verze R-CNN meta architektur: R-CNN, Fast R-CNN, Faster R-CNN a R-FCN, kdy nová verze byla vždy určitým vylepšením té předchozí a nahradila ji s výjimkou Faster R-CNN a R-FCN. Tyto poslední 2 verze lze považovat za rovnocenné, co se týče úspešnosti. Mimo tyto 4 hlavní přístupy bylo publikováno mnoho vylepšení těchto meta architektur.

R-CNN

R-CNN [12] je první verzí meta architektury z rodiny R-CNN, která byla publikována v roce 2013. Implementace[6] je volně dostupná. Průlomový byl přístup generování a následná klasifikace regionů zájmů na místo sliding window. Pro klasifikaci a regresi bounding boxů byla použita konvoluční neuronová síť v kombinaci se Support Vector Machine (SVM) realizující klasifikátor. Konvoluční neuronová síť byla použita pro extrakci konvolučních deskriptorů. Mohla být použita například libovolná neuronová síť zmíněná v kapitole 3. Trénování R-CNN probíhalo dle následujícího protokolu, který je zde uveden hlavně z důvodu možnosti porovnání přístupů s novějšími verzemi R-CNN a přístupem realizovaným v rámci této práce.

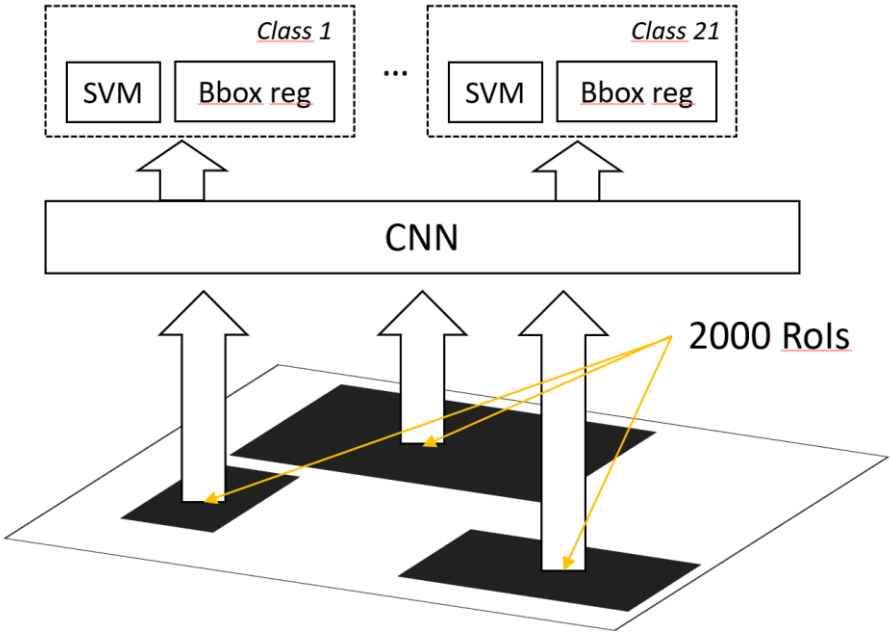
1. Natrénuje se obecný robustní klasifikátor (1000 tříd pro ImageNet), například s architekturou VGG16.
2. Provede se dotrénování na menší počet tříd (20 pro Pascal VOC), které se obohatí o třídu pozadí. Z klasifikátoru získaného v 1. kroku se odebere poslední plně propojená vrstva, nahradí se vrstvou pro definovaný počet tříd detektoru a váhy této vrstvy se inicializují from scratch, což značí že se použije určitá heuristika (náhodně, Xavier [13], msra [14] …).
3. Extrahují se regiony zájmů pomocí určitého algoritmu, kdy autoři R-CNN použili Selective Search [15]. Každý takto nalezený region byl vložen do neuronové sítě a z poslední konvoluční

[6] Implementace R-CNN je dostupná na https://github.com/rbgirshick/rcnn.

vrstvy byl získán vektor konvolučních deskriptorů, který se uložil na disk. Autoři uvádějí potřebu až 200 GB místa na disku pro datovou sadu Pascal VOC.

4. Pro každou třídu se natrénoval SVM pro klasifikaci a lineární regresor pro regresi bounding boxů z uložených vektorů konvolučních deskriptorů.

Výše popsaným postupem byl natrénován detektor, který dosahoval úspešnosti 63,5 mean average precision (mAP) na datové sadě Pascal VOC, což bylo zlepšení v průměru o 30 % oproti ostatním metodám. Mezi základní nevýhody R-CNN patří rychlost, kdy detekce objektů na snímku o velikosti 600x1000 trvala 50 sekund. Další nevýhodou je použití SVM, které se trénuje až po extrakci konvolučních deskriptorů a učení SVM se nijak neprojeví na vahách konvoluční neuronové sítě.



Obrázek 20: Schéma první verze R-CNN [16]. Černé obdélníky vyjadřují regiony zájmu (RoI). Ty vstupují do konvoluční neuronové sítě (CNN). Z každého regionu zájmu je zvlášť získán vektor konvolučních deskriptorů, který je klasifikován pro každou třídu zvlášť pomocí SVM.

Fast R-CNN

Další verzí R-CNN je Fast Region-based Convolutional Network (Fast R-CNN) [17], která přinesla zásadní vylepšení v podobě end-to-end přístupu učení a inference, kdy vektory konvolučních deskriptorů se již neukládaly na disk a byly využity ihned při vyhodnocování neuronové sítě pro realizaci klasifikace a lokalizace. Implementace[7] je volně dostupná. Činnost SVM jakožto klasifikátoru byla nahrazena vrstvou neuronové sítě realizující logistickou funkci Softmax, viz výraz (9).

[7] Implementace Fast R-CNN je dostupná na https://github.com/rbgirshick/fast-rcnn.

$$Softmax(X_i) = \frac{Exp(X_i)}{\sum_{j=0}^{k} Exp(X_j)} \; ; i = 0,1,2, \dots k \qquad (9)$$

Softmax v podstatě určuje pravděpodobnost dané třídy pro objekt, kdy součet pravděpodobností všech tříd je roven 1 a její ztrátová funkce pro klasifikaci je vyjádřena jako

$$L_{cls}(X, C) = \sum_{c=1}^{C} -log(Softmax(X_c), C_c) \qquad (10)$$

Symbol C ve ztrátové funkci pro klasifikaci představuje množinu tříd popředí včetně pozadí. Činnost lineárního regresoru realizující funkci L2 loss pro lokalizaci objektu byla nahrazena vrstvou realizující smooth L1 loss, která je stálejší a robustnější vůči změnám parametrů a nepotřebuje tudíž tak důsledné ladění hyper parametrů pro konvergenci. Ztrátová funkce pro lokalizaci je vyjádřena výrazy (11) a (12).

$$L_{loc}(T, v) = \sum_{c=1}^{C} \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i^c - v_i) \qquad (11)$$

$$smooth_{L1}(x) = \begin{cases} 0.5 * x^2 & |x| < 1 \\ |x| - 0.5 & jinak \end{cases} \qquad (12)$$

Ve výše uvedených výrazech $t^c = (t_x^c, t_y^c, t_w^c, t_h^c)$ značí opravu regionu zájmu, která byla získána jako výstup lokalizační větve neuronové sítě, pro danou třídu a anotaci bounding boxu $v = (v_x, v_y, v_w, v_h)$ objektu v obraze. Výsledná ztrátová funkce celé neuronové sítě byla definována jako součet ztrátových funkcí $L_{cls}(X, C)$ a $L_{loc}(t^c, v)$, kdy lokalizační funkce se do výsledné ztrátové funkce zapojuje pouze v případě, že třída není pozadí, jelikož pro pozadí není anotován žádný bounding box. Výsledná ztrátová funkce je vyjádřena výrazem (13).

$$L(X, C, T, v) = \sum_{c=1}^{C} (-log(Softmax(X_c, C_c)) + \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i^c - v_i)) \qquad (13)$$

Díky této výsledné ztrátové funkci, pomocí níž se neuronová síť trénuje, je možné oproti R-CNN používat mnohem hlubší neuronové sítě. Dalším důsledkem použití této ztrátové funkce, která při trénování neuronové sítě bere v potaz nejen třídu objektu, ale i jeho polohu, je přesnější lokalizace objektů a teoreticky schopnost učit se kontext objektů. Například pokud budou objekty jedné třídy v rámci datové sady vždy na stejném místě, pak se síť dokáže naučit, že v tomto místě nebude nic jiného. V závislosti na kontextu použití detektoru lze toto chování považovat za přetrénování nebo správné chování. Nahrazením SVM a lineárního regresoru bylo zajištěno trénování celé konvoluční neuronové sítě s ohledem na ztrátovou funkci klasifikace a lokalizace. Bounding boxy definující RoI

popředí bylo stále nutné dodávat do sítě externím algoritmem, avšak vzorky ze třídy pozadí si Fast R-CNN již generovalo samo, kdy autoři uvádějí ideální poměr popředí, pozadí 1:4.

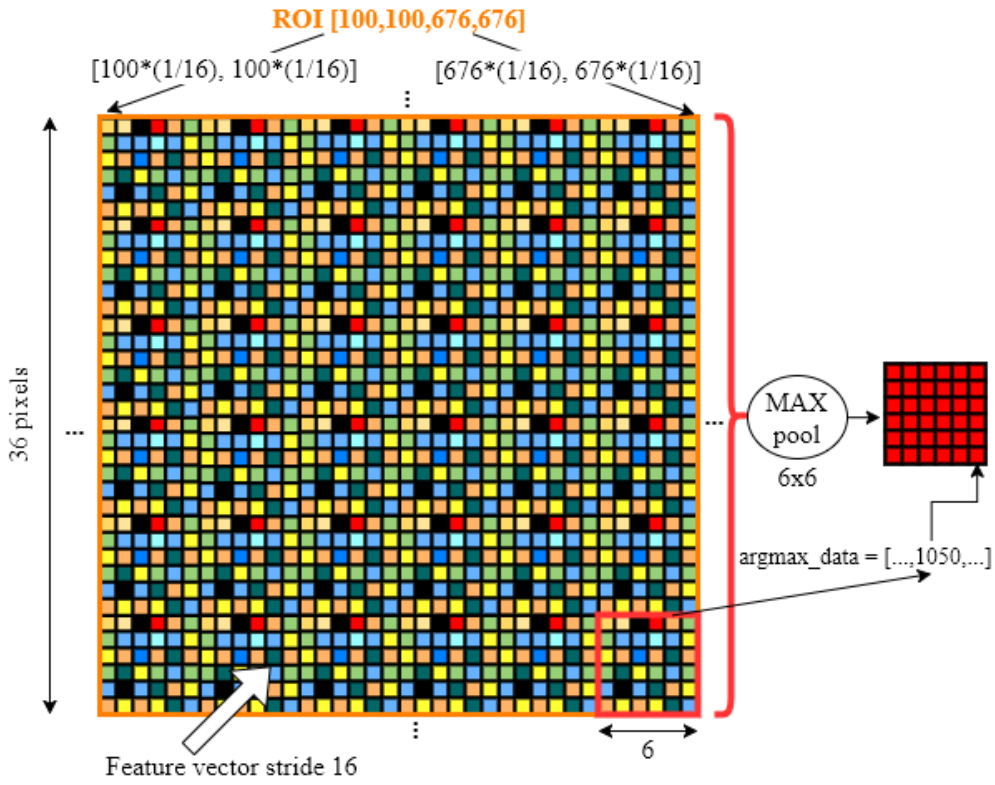
Dalším zásadním vylepšením byla extrakce vektoru konvolučních deskriptorů pro celý vstupní obraz najednou, tudíž nebylo nutné pro každý RoI provádět vyhodnocování skrze celou síť. Avšak aby tento mechanismus mohl být technicky realizován a aby bylo možné neuronovou síť vůbec učit end-to-end přístupem, bylo zapotřebí vytvořit speciální vrstvu, která bude diferencovatelná. Byla navržena vrstva RoI Pooling, která v dopředném kroku dokáže z vektoru konvolučních deskriptorů vyřezat data, která byla definována pomocí RoI v rámci originálního vstupního obrazu a vytvořit mini batch těchto RoI, které dále v rámci mini batch jsou klasifikovány a lokalizovány pomocí bounding box regrese. RoI Pooling je speciálním případem Spacial Pyramid Pooling (SPP) [18], avšak pouze s jednou úrovní pyramidy. V případě, že se RoI nepřekrývají, pak RoI pooling vrstva funguje jako vrstva Max Pooling s proměnnou velikostí jádra přes daný RoI. V případě překryvu se může uplatnit několik hodnot z vektoru konvolučních deskriptorů vícekrát a musí se uchovat index hodnoty, která byla vybrána jako výstup funkce Max Pool, v rámci vektoru konvolučních deskriptorů vstupujících do RoI Pooling, aby při zpětném šíření hodnoty ztrátové funkce bylo možné vypočítat gradient z hodnot, které vedly k dané hodnotě ztrátové funkce. Jelikož Max Pooling je diferencovatelná a současně jsou uchovány indexy hodnot, které se propagovaly z deskriptoru pomocí Max Pooling operace dále, lze RoI Pooling použít v end-to-end přístupu trénování neuronové sítě s využitím běžných optimalizačních algoritmů jako Stochastic Gradient Descent (SGD) a back propagation. Matematický zápis gradientu vycházejícího z RoI Pooling je zapsán výrazem (14).

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r,j)] * \frac{\partial L}{\partial y_{rj}} \qquad (14)$$

Symbol $x_i \in \mathbb{R}$ označuje hodnotu v konvolučním deskriptoru, který vstupuje do RoI Pooling, s indexem i a symbol y_{rj} definuje j-tý výstup z RoI Pooling na základě zpracování r-tého vstupního RoI. Výraz $[\mathrm{i} = i * (r,j)]$ je roven 1, pokud hodnota konvolučního deskriptoru na indexu i byla pomocí Max Pooling operace předána dále z r-tého RoI jako j-tý výstup z RoI Pooling, jinak je roven 0, což jinými slovy značí, že gradient se započítá pouze a přesně pro ty hodnoty, které byly operací Max Pooling vybrány do výstupního deskriptoru.

Obrázek 21 zobrazuje demonstraci fungování RoI Pooling vrstvy. Barevná šachovnice představuje konvoluční deskriptor, který v daném místě neuronové sítě má stride 16, což značí že 1 pixel deskriptoru se mapuje na 16 pixelů v originálním obraze. Do RoI Pooling vstoupil RoI, který je v obrázku vyznačen oranžovou barvou, s definovanými souřadnicemi x_{min}, y_{min}, x_{max}, y_{max} v originálním obraze. Souřadnice RoI se namapují podle stride do konvolučního deskriptoru.

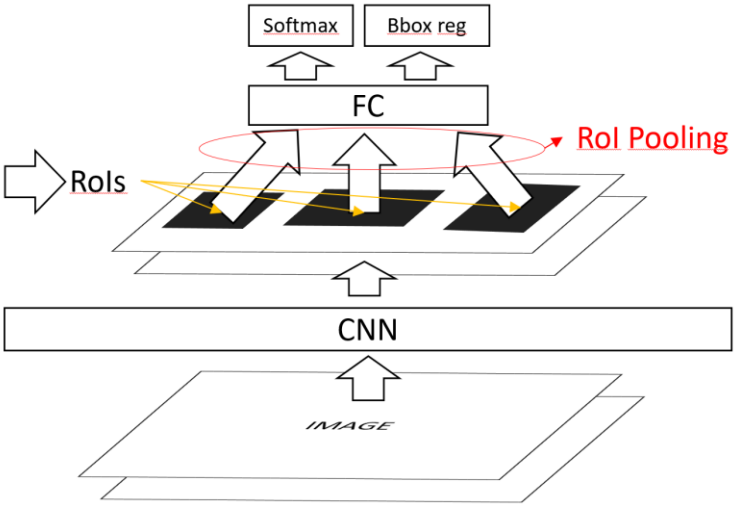
Dále se podle definované velikosti výstupního deskriptoru z RoI Pooling vypočítá velikost jádra pro Max Pooling operaci, kdy na schématu je vyznačeno, že RoI se mapuje do konvolučního deskriptoru na souřadnice [6, 6] a má velikost 36 pixelů, z čehož vyplývá, že při požadované velikosti výstupního deskriptoru má jádro operace Max Pooling velikost 6x6 pixelů. Na schématu vpravo dole je znázorněno uchování indexu hodnoty, která byla z daného regionu vybrána operací Max Pooling, pro výpočet gradientu při zpětném šíření chyby neuronové sítě.



Obrázek 21: Podrobné schéma fungování RoI Pooling vrstvy.

Také se musí přizpůsobit množství dat, které budou neuronovou sítí v jednom kroku procházet. V případě, že by se převzal stejný přístup jako u R-CNN, by jen pro vstupní vrstvu Fast R-CNN bylo zapotřebí $128 * 600 * 1000 * 3 * 4$ bytů, což je asi 875 MB a vedlo by to k 12x vyšším nárokům na výpočetní výkon a paměť oproti R-CNN. Proto bylo navrženo řešení v podobě hierarchického vzorkování dat pro sestavení mini batch, který byl tvořen z malého množství (cca 2) vstupních obrazů, avšak z každého obrazu bylo získáno mnoho RoI (cca 64), což vedlo na téměř shodné nároky na výpočetní výkon jako u R-CNN.

Všechna vylepšení, která byla zavedena u Fast R-CNN, v důsledku vedla k 25násobnému zrychlení detekce oproti R-CNN, což odpovídá 2 sekundám u obrázku 600x1000 pixelů, a zvýšení úspěšnosti o 3 % ze 63mAP na 66mAP u datové sady Pascal VOC. Obrázek 22 zobrazuje schéma meta frameworku Fast R-CNN.



Obrázek 22: Schéma Fast R-CNN [16]. Ze dvou celých vstupních obrazů jsou najednou extrahovány konvoluční deskriptory. Z externě dodaných RoI se pomocí RoI Pooling vytvoří mini batch, který dále pokračuje do plně propojených vrstev, kde se určí třída objektu (Softmax) a bounding box (Bbox reg).

Faster R-CNN

Vylepšením Fast R-CNN je Faster R-CNN [19], které spočívá hlavně ve zvýšení rychlosti využitím plnohodnotného end-to-end přístupu. Implementace[8] je volně dostupná. Jedná se v podstatě pouze o rozšíření Fast R-CNN o novou komponentu nazvanou Region Proposal Network (RPN), která nahrazuje externí algoritmus pro generování RoI. RPN je oproti externímu algoritmu mnohem rychlejší a také přesnější, jelikož se sama učí klasifikovat objekty popředí a pozadí. RPN je umístěna v neuronové síti za poslední konvoluční vrstvou, ze které se generují RoI a následně s vektorem konvolučních deskriptorů z poslední konvoluční vrstvy vstupují RoI do RoI Pooling, která byla popsána v meta architektuře Fast R-CNN.

Samotnou RPN lze popsat jako plně konvoluční neuronovou síť, která se učí klasifikovat vstupní vzorek jako popředí nebo pozadí a současně se učí bounding box regresy samotného RoI. Vstupním vzorkem u RPN je vektor konvolučních deskriptorů. RPN v tomto vektoru hledá RoI pomocí pravidelné mřížky bodů, které nanese na deskriptor. Body mají od sebe vzdálenost přesně tolik pixelů, kolik je stride neboli podvzorkování dané sítě v místě RPN. Bod definuje střed nabízeného regionu (anchor), který je vygenerován v několika měřítkách a s několika poměry stran. Po klasifikaci vygenerovaných anchor vzniknou RoI.

Během procesu učení sítě je RPN schopna generovat 20 000 RoI pro jeden vstupní obraz, a přesto je proces učení rychlejší, než tomu bylo u Fast R-CNN s pouze 2000 RoI. Při učení neuronové

[8] Implementace Faster R-CNN je dostupná na https://github.com/rbgirshick/py-faster-rcnn.

sítě RPN označí anchor jako pozitivní v případě, že Intersection over Union (IoU) tohoto anchor a anotovaného bounding boxu objektu je maximální nebo že IoU anchor a anotovaného objektu přesáhne definovaný práh. Za negativní anchor je považovaný ten, který má IoU s anotovaným objektem nižší než definovaný práh. Může se stát, že vlivem IoU nejsou některé anchor označeny ani pozitivní ani negativní, pak se tyto anchor ignorují a nezapojují se do procesu učení. Samotný proces učení Faster R-CNN s využitím přístupu end-to-end využívá 4 ztrátové funkce, kdy RPN využívá Softmax pro klasifikaci popředí/pozadí a smooth L1 Loss pro učení bounding box regrese. Stejné dvě ztrátové funkce využívá klasifikátor RoI a regresor bounding boxů pro RoI. Extraktor konvolučních deskriptorů se učí součtem gradientů přicházejících z RPN a klasifikátoru RoI. Tento přístup propagování gradientu je pouze aproximovaným učením neuronové sítě, a to vlivem toho, že oproti Fast R-CNN, kde RoI byly dodávané algoritmem, nejsou RoI z RPN konstantní, jelikož se v průběhu procesu učení celé sítě učí také RPN a produkuje v čase pro stejné vstupní vektory jiné RoI, a proto není RoI Pooling diferencovatelná podle RoI. Matematicky lze tento problém popsat následovně:

Pro Fast R-CNN platí, že RoI jsou konstantní.

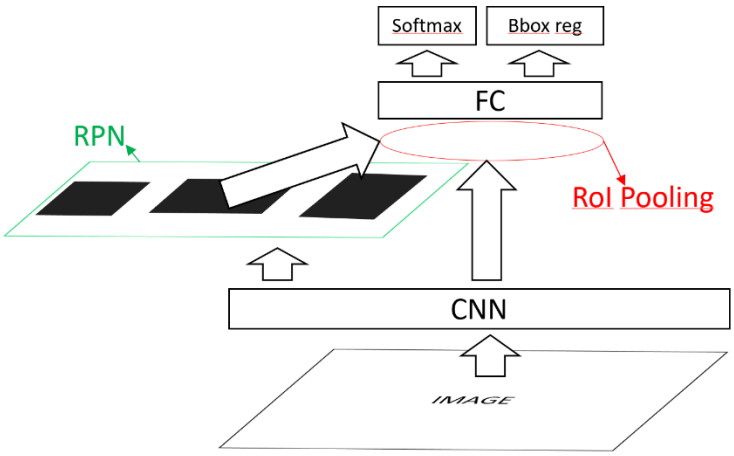
$$\frac{\partial L}{\partial RoI[i]} = 0 \, , i \in \{x_{min}, y_{min}, x_{max}, y_{max}\} \qquad (15)$$

Avšak pro Faster R-CNN obecně platí, že RoI nejsou konstanta

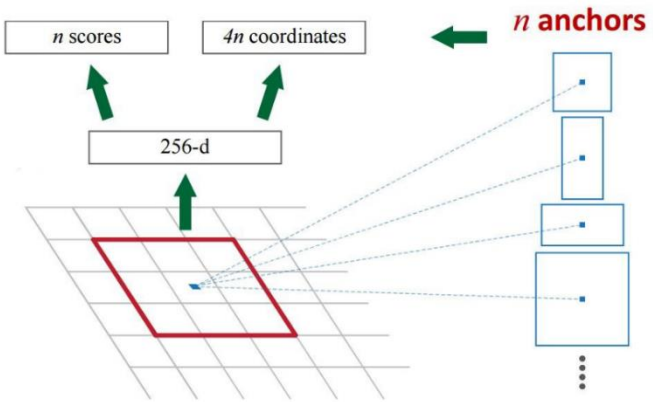
$$\frac{\partial L}{\partial RoI[i]} \neq 0 \, , i \in \{x_{min}, y_{min}, x_{max}, y_{max}\} \qquad (16)$$

ba dokonce $\frac{\partial L}{\partial RoI[i]} \, , i \in \{x_{min}, y_{min}, x_{max}, y_{max}\}$ neexistuje. Fakt, že neuronová síť není učena gradientem počítaným s ohledem na všechna data, která ovlivňují výstupní ztrátovou funkci, je ignorován a RoI produkované z RPN jsou považovány za konstantní. Dle experimentů autorů i přesto dosahuje trénování srovnatelných výsledků s neaproximovaným end-to-end přístupem s diferencovatelnou RoI Pooling podle RoI.

RPN u Faster R-CNN vede k 10násobnému zvýšení rychlosti oproti Fast R-CNN, což odpovídá 0,2 sekundy pro vstupní obraz 600x1000 pixelů. Vlivem přesnějších RoI se zvýšila také úspěšnost detektoru o 7 % na 73mAP na datové sadě Pascal VOC. V současnosti je Faster R-CNN state-of-the-art meta architektura pro realizaci detektoru a s využitím mocného extraktoru konvolučních deskriptorů bylo dosaženo úspešnosti 81,5mAP na datové sadě Pascal VOC. Obrázek 23 zobrazuje schéma meta architektury Faster R-CNN a Obrázek 24 zobrazuje princip generování RoI v RPN.



Obrázek 23: Schéma Faster R-CNN [16]. Ze vstupního obrazu je pomocí extraktoru konvolučních deskriptorů získán vektor deskriptorů, který je předán do RPN, kde dojde k vyhledání RoI pomocí klasifikace popředí/pozadí. Získaný vektor konvolučních deskriptorů spolu s RoI z RPN jsou pomocí RoI Pooling převedeny na mini batch, který je následně předán klasifikátoru tříd detektoru a regresosu bounding boxů.



Obrázek 24: Princip pokrytí vstupního konvolučního deskriptoru do RPN body, kde se kolem každého bodu generuje několik anchor s různými měřítky a poměry stran. Každý anchor je klasifikován dle třídy popředí/pozadí. Převzato z [16].

R-FCN

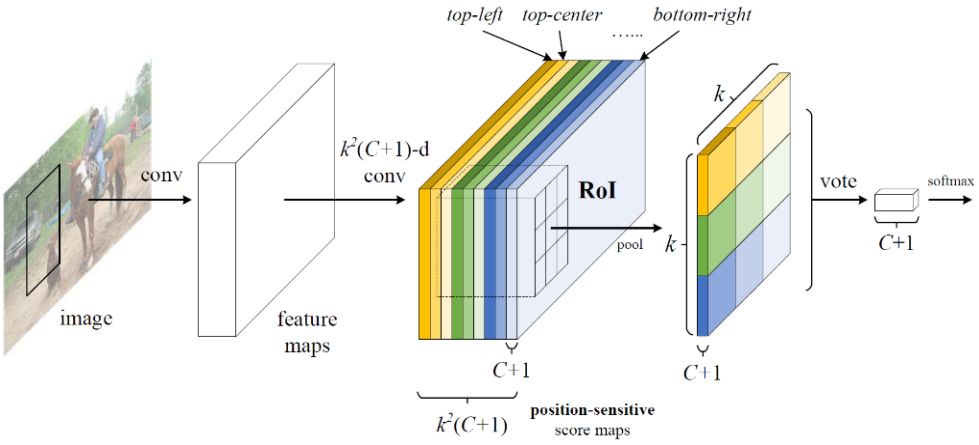
Jako poslední z rodiny meta architektur postavených na generování regionů zájmů byla zveřejněna plně konvoluční meta architektura s názvem Region-based Fully Convolutional Network (R-FCN) [20]. Tato meta architektura je silně inspirována Faster R-CNN a rovněž pro generování RoI využívá RPN, jejíž fungování je shodné s Faster R-CNN. Zásadním rozdílem oproti Faster R-CNN je využití plně konvolučního přístupu, z čehož vyplývá nutnost plně konvolučního extraktoru deskriptorů (klasifikátoru). Dalším rozdílem je využití maximálního možného sdílení předvypočítaných vektorů konvolučních deskriptorů.

Právě pro možnost sdílet předvypočítané vektory deskriptorů byla navržena Position Sensitive RoI Pooling (PS RoI Pooling), která má za úkol rozhodnout, zda se v daném RoI nachází objekt nebo ne na základě překryvu RoI a objektu. PS RoI Pooling pracuje v doméně konvolučních deskriptorů, což umožňuje tuto vrstvu neuronové sítě umístit až za všechny konvoluční vrstvy, a tak sdílet vektor konvolučních deskriptorů v rámci celého obrazu. Faster R-CNN využívalo sdílení konvolučních deskriptorů pouze po RPN a dále s využitím RoI Pooling se vytvořil nový batch, kdy na každý RoI v rámci batch byly aplikovány konvoluční operátory nezávisle.

Funkčnost PS RoI Pooling spočívá v rozdělení každého RoI vygenerovaného pomocí RPN pravidelnou mřížkou o $k\ x\ k$ buněk. PS RoI Pooling je navržena tak, že očekává z poslední konvoluční vrstvy extraktoru deskriptorů vektor o velikosti k^2 pro každou výstupní třídu, což dává vektor o velikosti $k^2 * (C + 1)$ konvolučních deskriptorů celého výstupního obrazu; nyní lze hovořit již o mapách skóre. PS RoI Pooling na rozdíl od RoI pooling provádí average pool operaci a to pouze nad jednou buňkou pravidelné $k\ x\ k$ mřížky, což lze vyjádřit vztahem (17).

$$r_c(i,j) = \sum_{(x,y)\in cell(i,h)} z_{i,j,c}(x + x_0, y + y_0)/n \qquad (17)$$

Označení $r_c(i,j)$ vyjadřuje výsledek average pooling operace nad buňkou (i,j) pravidelné mřížky vytvořené nad RoI pro třídu c, dále $z_{i,j,c}$ značí konkrétní mapu skóre z vektoru o velikosti $k^2 * (C + 1)$. Souřadnice levého horního rohu RoI v rámci celého vstupního vyjadřuje uspořádaná dvojice (x_0, y_0) a symbol n vyjadřuje počet pixelů v rámci jedné buňky pravidelné mřížky. Výsledkem PS RoI Pooling je $C + 1$ map skóre o velikosti $k\ x\ k$, kdy finální vektor aktivací pro klasifikaci objektu obsaženého v RoI je získán pomocí average pool operace. Obrázek 25 zobrazuje princip PS RoI Pooling.



Obrázek 25: Princip Position Sensitive RoI Pooling [20]. Vote vyjadřuje average pool operaci.

Stejně jako meta architektura Faster R-CNN zavádí R-FCN bounding box regresi, avšak R-FCN pracuje v režimu class-agnostic, což znamená, že se neprovádí bounding box regrese pro každou třídu, ale pouze pro popředí a pozadí, kdy třída výsledného bounding boxu popředí je dána klasifikační větví (Obrázek 25) detektoru.

PS RoI Pooling, plně konvoluční přístup a class-agnostic bouding box regrese má za následek 8násobné zvýšení rychlosti oproti Faster R-CNN a zvýšení úspěšnosti detekce na 79 mAP na datové sadě Pascal VOC, což je o 6 % lepší než Faster R-CNN.

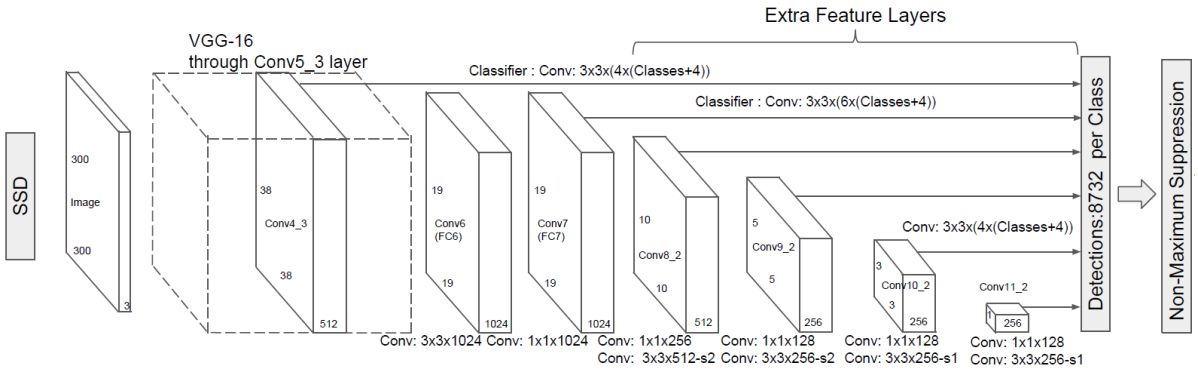
Přestože dle článku autorů dosahuje meta architektura R-FCN lepších výsledků než Faster R-CNN, byla pro realizaci této práce zvolena právě meta architektura Faster R-CNN, jelikož dosahuje dle měření [1] lepších výsledků při detekci malých objektů, což je s ohledem na velikosti objektů v datové sadě DataFromSky žádoucí.

4.2.2 SSD

Single Shot Multibox Detector (SSD) [21] je meta architektura, která realizuje detektor bez nutnosti generovat regiony zájmů, kdy pro takovéto meta architektury je používáno označení single shot. Implementace[9] je volně dostupná. Dominantní část výpočtů je realizována pomocí neuronové sítě. Kostra SSD je tvořena neuronovou sítí pro extrakci konvolučních deskriptorů, například některou z kapitoly 3, kdy jsou ke kostře dále přidány další konvoluční vrstvy, které mají zajistit extrakci konvolučních deskriptorů v několika meřítkách, tzv. feature layers. Výstupy z originálního extraktoru konvolučních deskriptorů a feature layers jsou propagovány každý do své speciální vrstvy, která zajistí vygenerování oken v několika měřítkách a poměrech stran přes celý vektor deskriptorů, které slouží k diskretizaci prostoru vstupního obrazu. Jelikož se okna generují pro každou feature layer, je dosaženo diskretizace vstupního obrazu v několika měřítkách, a tím možnost detekce různě rozměrných objektů. V rámci každé diskretizace existují dvě další větve, které slouží pro klasifikaci daného okna a regrese bounding boxu v rámci daného okna.

Existují dva modely SSD a to SSD-300 pracující s velikostí vstupního obrazu 300 x 300 pixelů a SSD-512 s velikostí vstupu 512 x 512 pixelů. Omezení na tyto velikostí vychází z konstrukce neuronové sítě, která využívá pro realizaci detektoru podvzorkování vstupního obrazu.

[9] Implementace SSD je dostupná na https://github.com/weiliu89/caffe/tree/ssd.



Obrázek 26: Schéma [21] SSD-300 meta architektury. Jako kostra je využita síť VGG-16 s velikostí vstupu 300 x 300 pixelů. Dodatečné feature layers pracují s měřítky obrazu 1/8, 1/16, 1/32, 1/64, 1/100 a 1/300.

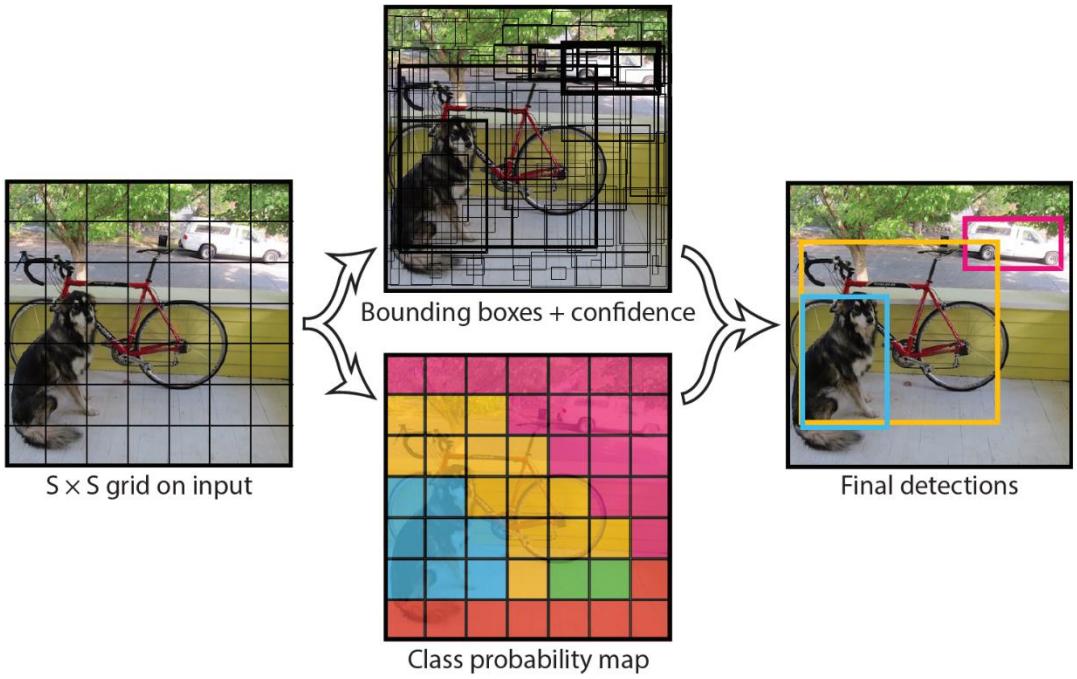
Výhodou SSD je jednoduchá struktura a implementace, která využívá standartní vrstvy neuronových sítí až na vrstvu, která diskretizuje vstupní obraz v prostoru konvolučních deskriptorů. Jako všechny detektory z rodiny single shot se vyznačuje SSD vysokou rychlostí detekce, která činí dle autorů 50 snímků velikosti 300 x 300 pixelů za sekundu. Na testovací datové sadě Pascal VOC dosahuje SSD skóre 80 mAP. Nicméně SSD má velice dobrou úspěšnost pouze v detekci velkých objektů, avšak špatně detekuje malé objekty [1], což není žádoucí vlastnost při realizaci detektoru nad datovou sadou DataFromSky.

4.2.3 YOLO

Meta architektura You Only Look Once (YOLO) [22] stejně jako SSD negeneruje regiony zájmů, ale k realizaci detektoru využívá vlastností konvolučních deskriptorů vzniklých aplikací jednoduché konvoluční neuronové sítě na celý vstupní obraz. YOLO nahlíží na problém detekce z hlediska neuronové sítě pouze jako na problém klasifikace, kdy výsledné bounding boxy a pravděpodobnosti třídy objektu jsou získány algoritmicky.

Princip YOLO spočívá v diskretizaci vstupního obrazu pomocí mřížky o rozměru $S * S$ buněk. Jestliže je objekt ve středu této buňky, je tato buňka zodpovědná za jeho detekci. V rámci každé buňky je definováno B bounding boxů a pro každý bounding box jeho skóre, které definuje, zda bounding box obsahuje daný objekt a současně jak moc přesný je daný bounding box kolem objektu. Lze říci, že skóre bounding boxu je rovno 0, pokud v dané buňce není žádný objekt popředí, jinak je skóre dané podílem plochy průniku bounding boxu buňky, anotace a sjednocením plochy bounding boxu buňky, anotace – neboli Intersection over Union (IoU). Dále je nad buňkou pravidelné mřížky definována pravděpodobnost třídy objektu obsaženého v buňce, avšak toto skóre existuje pouze pokud buňka opravdu objekt obsahuje. Pro každou buňku je definován pouze jeden vektor pravděpodobností třídy objektu, i když nad buňkou je definováno B bounding boxů. Pro získání skóre třídy objektu pro každý

bounding box se pouze vynásobí pravděpodobnosti třídy objektu a skóre bounding boxu nad danou buňkou mřížky. V rámci neuronové sítě jsou tyto pravděpodobnosti kódovány jedním vektorem o rozměru $S * S * (B * 5 + C)$. Obrázek 27 zobrazuje princip YOLO detektoru.



Obrázek 27: Princip YOLO [22]. Mřížka byla zvolena jako 7 x 7 a u každé buňky mřížky byly nezávisle na sobě vygenerovány 2 bounding boxy a skóre každé třídy, kdy obrázek zobrazuje buňku obarvenou barvou třídy s maximálním skóre. Výsledné bounding boxy a pravděpodobnost třídy jsou získány aplikací NMS a vynásobením pravděpodobnosti třídy objektu a skóre bounding boxu.

Přístup k realizaci detekce, který zavádí YOLO, vede k velice výkonnému řešení, které dokáže pracovat s rychlostí 45 snímků za sekundu a s přesností 64 mAP. Autoři také uvádějí, že YOLO pracuje dobře pouze při detekci velkých objektů a pro malé objekty selhává, proto nebylo YOLO zvoleno pro realizace této práce.

5 Úpravy Faster R-CNN a Caffe

V rámci této práce byla upravena meta architektura Faster R-CNN a framework Caffe. Jedná se především o přidání dodatečné funkcionality ovlivňující rychlost a celkový výsledek učení.

5.1 Caffe

Napříč komunitou bylo posbíráno mnoho drobných úprav frameworku Caffe, které vedly k optimalizacím. Dále byly provedeny následující úpravy:

5.1.1 Integrace knihovny cuDNN 7

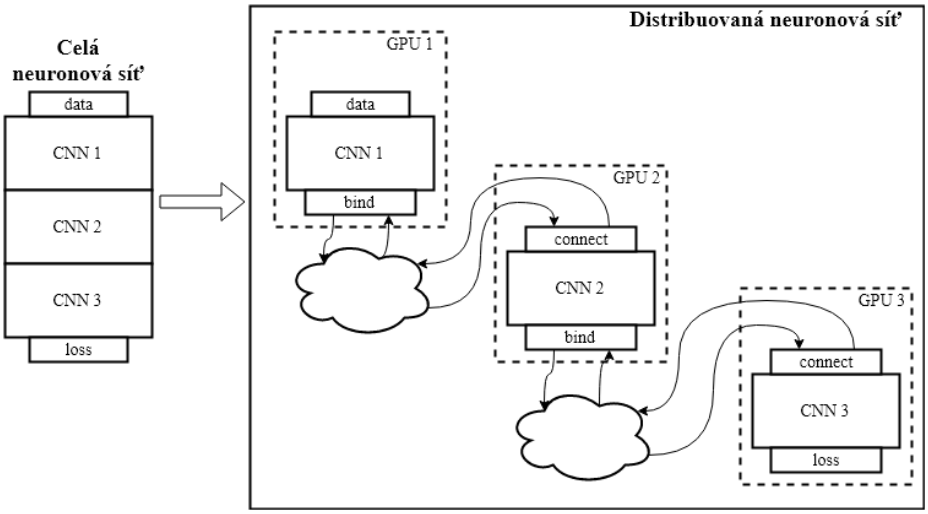
Meta architektura Faster R-CNN využívá několik speciálních operátorů, které bylo nutné do frameworku Caffe přidat jako vlastní vrstvy, což zajistili autoři Faster R-CNN. Framework Caffe se však od doby vydání Faster R-CNN dále vyvíjí, a proto bylo nutné do verze Caffe pro Faster R-CNN integrovat nová vylepšení. Jedno z nich je knihovna NVIDIA cuDNN 7 [24], která umožňuje velice efektivní výpočty konvoluce a jiných operátorů používaných v oblasti neuronových sítí na grafické kartě. Bylo tedy nutné přemigrovat všechny změny z verze Caffe pro Faster R-CNN do nejnovější verze Caffe 1.0.

5.1.2 Distribuované učení

Framework Caffe umožňuje trénovat neuronové sítě pouze duplikací stejné neuronové sítě na více grafických karet s využitím knihovny NVIDIA NCCL, což v praxi přináší pouze možnost zvýšit velikost efektivního batche a rychlejší učení, ale neumožňuje to trénovat obří sítě, které vyžadují víc grafické paměti, než je k dispozici na jedné grafické kartě. Další nevýhodou je, že při trénování na více grafických kartách je Caffe omezeno nejmenší velikostí grafické paměti, kterou disponuje jedna z grafických karet zapojených do učení.

Rozšíření v podobě distribuovaného učení přináší možnost rozložit jednu neuronovou síť na více grafických karet, a to i na grafické karty na jiných uzlech systému (na jiném počítači). Implementace vyžadovala vytvoření speciálních Caffe operátorů, které fungují jako sloty zprostředkovávající dopředné a zpětné šíření dat v síti. Sloty jsou implementovány pomocí knihovny zeroMQ [23] pro soketovou komunikaci, z čehož vyplývá možnost distribuce neuronové sítě na více uzlů výpočetního systému. Distribuovaná síť je synchronizována díky blokujícím čekáním soketů. Díky použití nativního prostředí frameworku Caffe při tvorbě slotů je možné na každém uzlu využít učení na více grafických kartách, jak bylo popsané v předchozím odstavci. Při praktickém použití byla trénována neuronová síť na 4 grafických kartách NVIDIA GTX 1080ti s plně obsazenou grafickou

pamětí 44 GB, což umožnilo trénovat neuronovou síť s obřím efektivním batch pro rychlejší konvergenci sítě. Rychlost učení díky knihovně zeroMQ nebyla nijak omezena. Obrázek 28 zobrazuje ukázku distribuce neuronové sítě. Každá část ukládá své váhy lokálně, váhy jsou po ukončení učení spojeny do jedné originální sítě prostřednictvím implementovaných skriptů.



Obrázek 28: Schéma distribuce neuronové sítě na části CNN1, CNN2 a CNN3 s využitím soketové komunikace.

5.2 Faster R-CNN

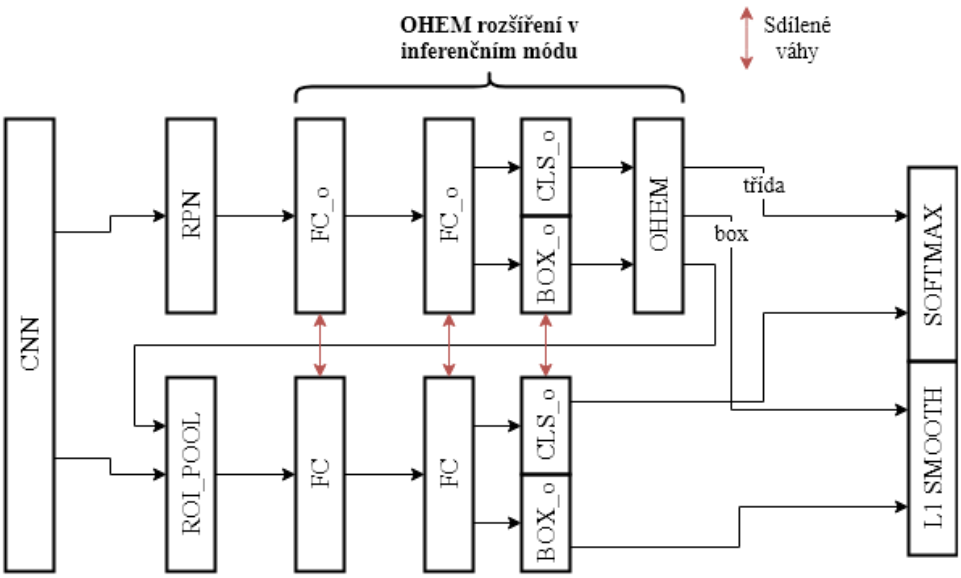
U meta architektury Faster R-CNN byly přidány následující funkcionality:

5.2.1 Online Hard Example Mining

Hard Example Mining je technika učení neuronové sítě, kdy na základě aktuálního stavu natrénování sítě jsou vybírány trénovací vzorky pro síření gradientu učení dle odezvy v podobě ztrátové funkce neuronové sítě na daný vzorek. Vybírá se *n* vzorků s nejvyšší hodnotou ztrátové funkce, tedy vzorky, které jsou pro neuronovou síť složité. Tímto lze docílit, že se do procesu učení nebudou zapojovat vzorky, které již nepřináší žádné zlepšení, a naopak se primárně vybírají vzorky, které zajistí gradient vedoucí k výraznější změně učících se parametrů sítě.

V případě meta architektury Faster R-CNN je tato metoda zavedena přímo do end-to-end procesu učení, a proto je nazývána Online Hard Example Mining (OHEM). Pro realizaci OHEM bylo nutné vytvořit dodatečné operátory a do architektury neuronové sítě přidat větev, která zajistí výběr složitých vzorků. Obrázek 29 zobrazuje rozšířený model neuronové sítě o OHEM. Nutno poznamenat, že OHEM se uplatní pouze při trénování sítě.

Pro zavedení OHEM byla využita technika sdílených vah mezi operátory, jakož tomu bývá u architektur siamských neuronových sítí. Konkrétně u frameworku Caffe zavedení OHEM se sdílenými váhami operátorů zapříčiní nemožnost trénovat na více GPU přes nativní rozhraní, jelikož Caffe nepodporuje multi GPU trénování operátorů se sdílenými váhami. Trénování s OHEM technikou s sebou nese zvýšené nároky na výpočetní výkon a grafickou paměť. Tato technika byla využita v závěrečné fázi trénování neuronové sítě pro maximální zlepšení úspěšnosti.



Obrázek 29: Schéma operátorů rozšířeného modelu neuronové sítě o OHEM.

5.2.2 Online výřezy

Aby bylo možné trénovat neuronovou síť na datové sadě DataFromSky, která je tvořena převážně snímky s vysokým rozlišením (4K), bylo nutné zavést techniku výřezů snímků.

Originální řešení tohoto problému v meta architektuře Faster R-CNN je v podobě zmenšení vstupního snímku. Toto řešení je však pro tuto práci nepřijatelné, jelikož objekty na snímcích z datové sady DataFromSky jsou již v originálním rozlišení dosti malé a dodatečným zmenšením snímku by se velikost objektů dostala na technologickou hranici návrhu detektoru, která činí 16 pixelů díky 16násobnému podvzorkování extraktoru konvolučních deskriptorů. Jakákoliv anotace se stranou menší než 16 pixelů je vyřazena z trénování, jelikož by RPN klasifikovala deskriptor o velikosti 1x1, případně žádný.

Pro zavedení tohoto rozšíření bylo nutné upravit stávající operátory načítající data pro neuronovou síť. Výřezy jsou tvořeny z originálního snímku o definované velikosti s definovaným překryvem, aby nedošlo k půlení objektů popředí. Dle výřezu se musejí příslušně upravit anotace a ponechat pouze

ty, které svým obsahem zasahují do výřezu alespoň s definovanou částí (70 % plochy). Celý tento proces se provádí v rámci end-to-end učení detektoru v odděleném vlákně a s přednačítáním snímků.

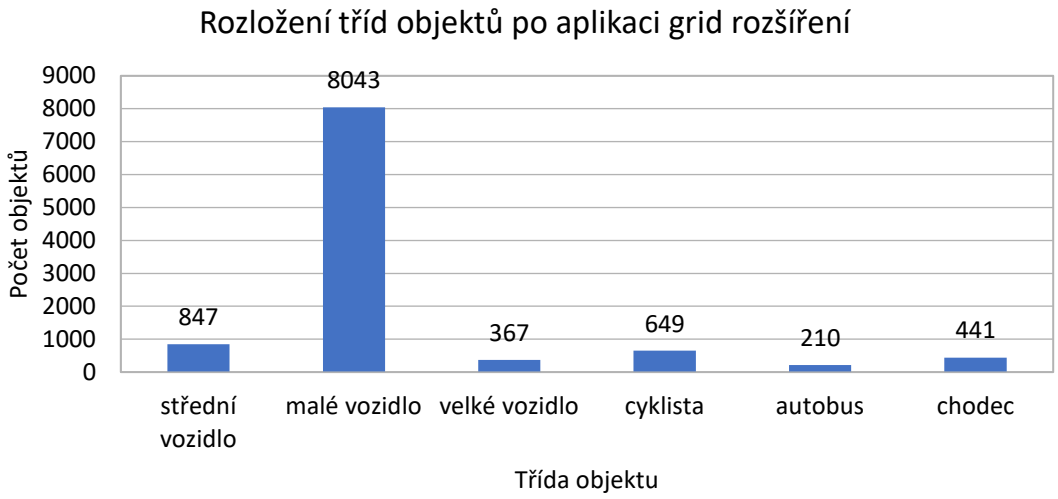
5.2.3 Online grid

Zásadním problémem originálního řešení meta architektury Faster R-CNN je staticky definovaný poměr mezi pozitivními a negativními vzorky. V kapitole popisu datové sady Graf 3 zobrazuje histogram počtu objektů na jednotlivých snímcích. Je vidět, že u datové sady DataFromSky není možné téměř v žádném batch dosáhnout efektivního poměru mezi vzorky popředí a pozadí při velikosti batch 128 a poměru popředí a pozadí 1:4. Pak by se neuronová síť učila primárně rozpoznávat pozadí, avšak cílem je rozpoznávat popředí. Proto je zavedení tohoto rozšíření kritické pro dosažení lepší úspěšnosti detektoru.

Rozšíření spočívá v tvorbě nového snímku pro trénování složeného z mnoha výřezů objektů popředí napříč několika snímky. Takto vzniklé snímky jsou nazvány gridem a výřezy jsou označeny jako dlaždice. Velikosti dlaždice musí být dělitelem velikosti gridu a tyto velikosti jsou voleny staticky. Generují se 2 typy gridů: pozitivní grid a negativní grid. Při generování pozitivního gridu je volena menší velikost dlaždice, čímž je dosaženo vyššího počtu objektů popředí na trénovacím snímku. Negativní grid je naopak generován s většími dlaždicemi, aby bylo v rámci trénování zajištěno dostatečné množství vzorků spadající do třídy pozadí. V rámci každé dlaždice jsou anotovány objekty, které jsou uvnitř dlaždice alespoň 70 % svého obsahu. Graf 4 zobrazuje histogram počtu objektů po aplikaci grid rozšíření na 350 gridech.

Graf 4: Histogram počtu objektů na snímku po aplikaci grid rozšíření. V porovnání s Graf 3 je vidět znatelné zvýšení počtu objektů popředí.

Počty objektů popředí se po aplikaci grid rozšíření zlepšily, avšak rozložení tříd stále kopíruje původní rozložení tříd datové sady DataFromSky, což vede k nekvalitním detekcím málo zastoupených tříd. Tento problém zobrazuje Graf 5.

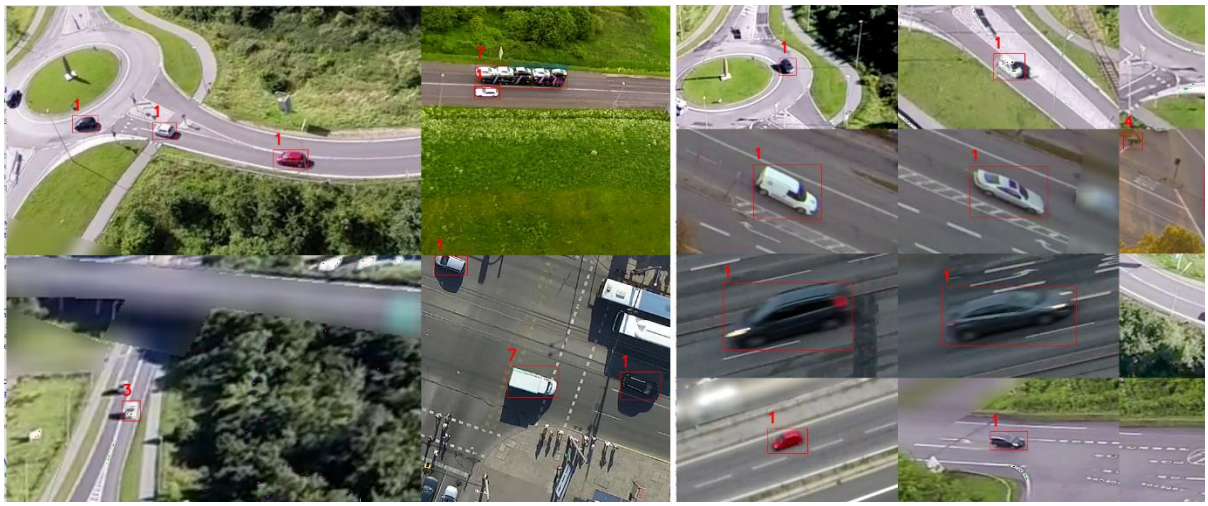


Graf 5: Rozložení tříd objektů po aplikaci grid rozšíření.

Pro celý tento proces byla naimplementována obsluha, která zajistí generování gridů v průběhu end-to-end procesu trénování detektoru. Pozitivní grid se generuje každou sudou iteraci a negativní grid každou lichou.

Grid rozšíření je kombinováno s rozšířením tvorby výřezů, které je popsáno v kapitole 5.2.2, pro dosažení lepší distribuce datové sady do batch. Pak je velikost gridu nastavena staticky na 8064x4320 a velikost pozitivní dlaždice na 384x216 a velikost negativní dlaždice na 1008x540, z čehož vyplývá, že na pozitivním gridu je nejméně 420 objektů popředí a na negativním gridu je nejméně 64 objektů popředí z více náhodně vybraných snímků. Velikost výřezu je volena tak, aby výsledný snímek pro trénování odpovídal výřezu 3x4 dlaždic z pozitivního gridu a jelikož jsou dlaždice do gridu zařazovány po řádcích, tak je vysoká pravděpodobnost (dle Graf 3), že se v rámci výsledného výřezu (3x4 dlaždice) budou nacházet objekty napříč 3 různými snímky. Nutno poznamenat, že dlaždice jsou vyřezávány ze snímků/gridů v originálním měřítku a poměrech stran pro zachování skutečných rozměrů a tvarů objektů.

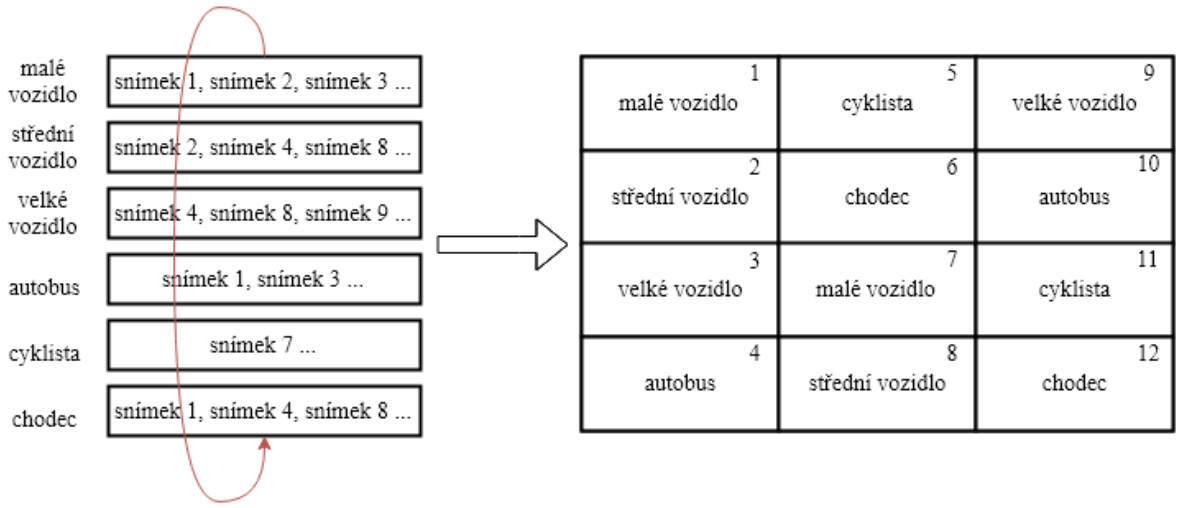
Obrázek 30 zobrazuje ukázku pozitivního a negativního gridu generovaného pro trénování. Bylo ověřeno, že ostré hrany mezi jednotlivými dlaždicemi nezpůsobují žádné negativní chování detektoru.



Obrázek 30: Ukázka negativního gridu (vlevo) a ukázka pozitivního gridu (vpravo) s vyznačenými anotacemi a třídou objektu.

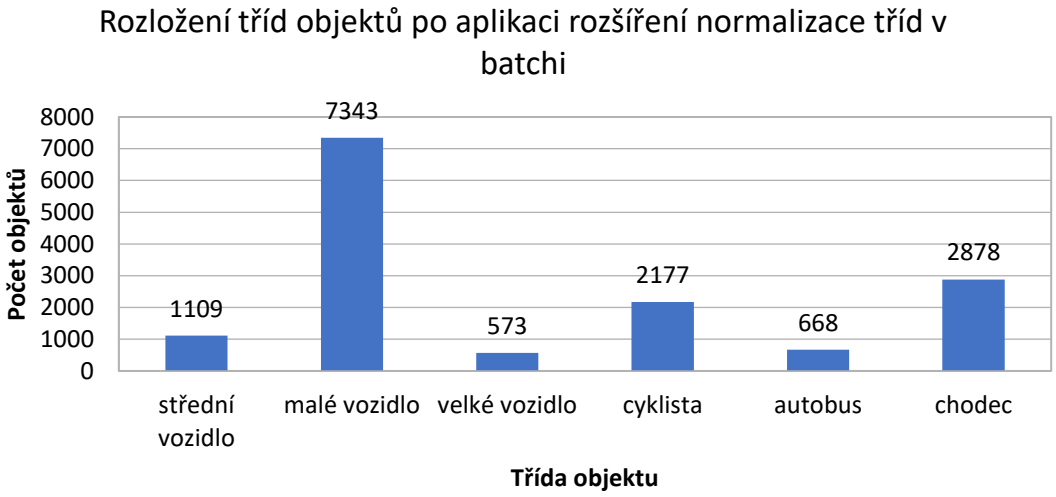
5.2.4 Normalizace tříd v batch

Jelikož při trénování detektoru není ideální poměr v zastoupení jednotlivých tříd popředí v batch, dochází k nekvalitní výsledné detekci málo zastoupených tříd. Tento problém se snaží řešit úprava nazvaná normalizace tříd v batch, kdy cílem je vyrovnat zastoupení jednotlivých tříd popředí a přiblížit se k uniformnímu rozložení. Princip spočívá ve vytvoření seznamů snímků dle výskytu jednotlivých tříd popředí. Nutno podotknout, že toto rozšíření lze uplatnit pouze v kombinaci s grid rozšířením, kdy při tvorbě jednotlivých dlaždic se postupně vybírají objekty ze seznamů snímků, na kterých se právě zařazovaná třída vyskytuje. Z daného snímku se vybere pouze jeden objekt, a to objekt právě zařazované třídy. Seznamy jsou zamíchány nezávisle na sobě a výběr jednotlivých snímků z daného seznamu je také nezávislý. Obrázek 31 zobrazuje princip rozšíření normalizace tříd v batch.



Obrázek 31: Schéma ilustrující rozšíření normalizace tříd v batch při tvorbě gridu. Je vidět, že v rámci gridu 3x4 je zastoupena každá třída popředí alespoň jednou.

Při učení detektoru je toto rozšíření kombinováno s rozšířením tvorby výřezů, kdy velikost výřezu je příhodně definována, aby rozložení dlaždic gridu v daném výřezu pro trénování bylo přesně 3 x 4 dlaždice, jak zobrazuje Obrázek 31. Tím je dosaženo, že v každém batch je každá třída popředí zastoupená alespoň dvakrát, a tím pádem se váhy neuronové sítě upravují s ohledem na všechny třídy.



Graf 6: Rozložení tříd objektů po aplikaci rozšíření normalizace tříd.

Graf 6 zobrazuje rozložení tříd objektů popředí po aplikaci rozšíření normalizace tříd. Rozložení tříd je optimálnější, než ukazuje Graf 1 a Graf 5, což potvrzují i výsledky učení detektoru.

5.2.5 Augmentace

Aby výsledná neuronová síť dokázala co nejlépe generalizovat a současně se zamezilo náchylnosti k přetrénování, bylo zavedeno rozšíření, které během end-to-end procesu učení aplikuje na trénovací vzorky úpravy:

- vizuální změny:
  - posuv světlosti,
  - posuv kontrastu,
  - aditivní šum – nastavena nižší pravděpodobnost aplikace,
  - posuv teploty,
  - rozostření – nastavena nižší pravděpodobnost aplikace,
  - posuv gammy,
  - zvýšení nebo snížení ostrosti,
- geometrické změny:

- zmenšení nebo zvětšení snímku, aby kratší strana nejmenšího anotovaného objektu na snímku byla dlouhá 32 pixelů. Tato úprava se aplikuje pouze v případě, že kratší strana nejmenšího objektu na snímku je větší než 10 pixelů a kratší strana největšího objektu na snímku je menší než 128.

Parametry posuvů všech vizuálních změn jsou definovány gaussovou křivkou, tedy středem a směrodatnou odchylkou, kdy střed je volen v nulové hodnotě transformace (například pro teplotu je to okolo 6500 kelvinů) a směrodatná odchylka dle pravidla 3 sigma, aby nejvzdálenější vygenerovaná hodnota reprezentovala 50% změnu daného parametru. Mimo proces učení se aplikuje geometrická změna v podobě zrcadlení obrazu kolem vertikální osy, kdy díky této úpravě je možné zdvojnásobit počet vzorků v datové sadě okamžitě. Při zavedení rozšíření augmentace lze hovořit o efektivní velikosti datové sady DataFromSky jako o 10krát větší datové sadě, což činí asi 70 milionů objektů na 3 000 000 snímcích.

5.2.6 Distribuované učení

Pro natrénování detektoru bylo zapotřebí velké množství grafické paměti, především při aktivaci OHEM rozšíření a zvýšení velikosti efektivního batch. Jako řešení bylo implementováno rozšíření v podobě distribuované meta architektury Faster R-CNN nad rozšířením frameworku Caffe popsaným v kapitole 5.1.2. Implementace spočívá v zavedení synchronizačních mechanismů a datových toků realizovaných pomocí soketové komunikace s využitím knihovny zeroMQ. Synchronizovat je nutné především ukládání výsledků, zahájení a ukončení učení. Dále je nutné přenášet parametry datové sady (mean, bounding box mean a směrodatná odchylka), které se počítají při spuštění učení, jelikož při distribuovaném režimu načítá datovou sadu pouze jeden proces.

Dále byla do původní meta architektury Faster R-CNN přidána podpora učení neuronových sítí na více grafických kartách tak, jak tomu je u frameworku Caffe. Nebylo možné využít nativního multi GPU řešení, protože meta architektura má mnoho operátorů implementovaných prostřednictvím Python rozhraní a framework Caffe není schopen zapojit tyto operátory do procesu učení na více grafických kartách z důvodu odlišného chování multi procesního zpracování programů v jazyce Python a C++. Nicméně knihovna NVIDIA NCCL disponuje i Python rozhraním, prostřednictvím kterého bylo multi GPU učení v meta architektuře Faster R-CNN doimplementováno.

5.2.7 Soft NMS

V drtivé většině případů je NMS algoritmus aplikován pro extrakci detekcí pro filtraci překrývajících se detekcí (zpravidla jednoho objektu). Překryv je stanoven na základě poměru plochy průniku detekcí a jejich plochy po sjednocení (IoU). Po aplikaci NMS jsou všechny detekce překrývající se více

než je daný práh smazány, pokud jejich skóre není maximální, z čehož vyplývá, že při aplikaci klasického NMS existuje ostrá hrana mezi ponechanými detekcemi a vymazanými, jinými slovy řečeno detekcím, které nejsou ponechány je přiřazeno skóre 0 dle vztahu (18), kde s_i je skóre detekce *i*, T je práh míry překryvu, b_i je bouding box detekce a M jsou bouding boxy s maximálním skóre. U překrývajících se objektů je problematické určit, zda se jedná o detekci toho samého objektu a tím pádem bude vymazána, anebo o detekci jiného objektu a pak by měla zůstat zachována.

$$s_i = \begin{cases} s_i, & IoU(M, b_i) < T \\ 0, & IoU(M, b_i) \geq T \end{cases} \quad (18)$$

Tento problém se snaží řešit metoda Soft NMS [25], která zavádí namísto přiřazení skóre 0 detekcím, které nejsou ponechány, přiřazení skóre dle funkce překryvu. Tato funkce snižuje skóre detekcím se zvětšujícím se překryvem. Autoři Soft NMS navrhují 2 funkce pro přiřazení nového skóre detekcím na základě IoU. Funkce (19) lineárně snižuje skóre se zvětšujícím se překryvem detekcí, nicméně tato funkce není spojitá a změna skóre je uplatněna pouze v případě dosažení prahu překryvu.

$$s_i = \begin{cases} s_i, & IoU(M, b_i) < T \\ s_i * (1 - IoU(M, b_i), & IoU(M, b_i) \geq T \end{cases} \quad (19)$$

Funkce (20) je druhou funkcí pro výpočet nového skóre překrývajících se detekcí. Tato funkce je modelována Gaussovou funkcí a je spojitá a její vlastností je, že změna skóre při změně překryvu je rychlejší. Funkce je aplikována na všechny bouding boxy, které ještě nebyly vybrány jako finální.

$$s_i = s_i e^{-\frac{IoU(M,b_i)^2}{\sigma}}, \forall b_i \notin Zachované\ detekce \quad (20)$$

Složitost Soft NMS je $O(N^2)$, což je stejná složitost, jako u klasického NMS, a nedojde tedy k žádné navýšení výpočetní náročnosti. Úspěšnost detektorů postavených na meta architektuře Faster R-CNN byla dle měření autorů díky aplikaci Soft NMS navýšena v průměru o 1 mAP, nicméně v případě této práce bylo navýšení úspěšnosti zanedbatelné a raději byla využita klasická verze NMS s GPU akcelerací.

5.2.8 Další úpravy

V rámci této práce prošla meta architektura Faster R-CNN revizí. Jelikož zdrojové kódy Faster R-CNN byly postaveny na zdrojových kódech meta architektury Fast R-CNN, bylo v obslužných rutinách mnoho kódu, který neměl pro Faster R-CNN žádný význam případně byl neoptimálně implementován

(například aplikace transformace zvětšení, i když k žádnému nedojde), a proto byl odstraněn a mnoho částí bylo zjednodušeno.

Pro potřeby datové sady DataFromSky byla integrována obsluha načítání této datové sady. Tato obsluha je volána pouze jednou na začátku procesu učení, a proto bylo mnoho podmínek a filtrací nevalidních anotací, které se v originální verzi prováděly napříč zdrojovými kódy, přesunuto do této obsluhy. Mezi hlavní úpravy, které zrychlily proces učení, patří načítání veškerých dostupných informací o snímcích (velikost …) z anotačního souboru snímků, jelikož v původní verzi se velikost snímku zjistila načtením obrázku z disku do paměti, což je poměrně časově náročná operace.

Dále bylo po revizi RPN implementace zjištěno, že v originální verzi se neuplatní při výběru trénovacích vzorků anotované objekty vždy, ale jsou vybírány náhodně i ze sady RoIs, které odpovídají kritériu, že IoU anotace a RoI je větší než 75 %. RPN byla upravena, aby vždy přednostně vybírala anotace a až poté doplňovala vzorky náhodně do velikosti batch dle stanoveného poměru popředí a pozadí.

Pro rychlejší inferenci Faster R-CNN byla nahrazena vrstva generující RoI implementovaná v jazyce Python jakožto část RPN za implementaci v C++ a CUDA, což se provilo zrychlením detektoru o 20 %.

Pro pohodlnější obsluhu Faster R-CNN byla všechna nastavení parametrů a rozšíření přesunuta do globálních konfiguračních souborů, které lze přetížit lokálním konfiguračním souborem vstupujícím jako parametr do procesu učení a inferenčních skriptů.

Pro dosažení maximální propustnosti detektoru byl implementován distribuovaný systém jakožto zřetězená linka s vyrovnávací pamětí mezi jednotlivými bloky využívající framework Caffe a meziprocesní komunikaci v jazyce Python. Linka byla složena z bloků:

- načítání snímků z videa a tvorba výřezů u snímků s vysokým rozlišením,
- inference sítě,
- sběr a skládání detekcí z jednotlivých výřezů,
- export detekcí případně vizualizace.

Jelikož obsluha v jazyce Python nebyla schopna dosáhnout požadované rychlosti, byla tato zřetězená linka přepsána do jazyka C++, avšak stále využívající framework Caffe, který také nebyl schopen dosáhnout dostatečné rychlosti, proto byla vytvořena zcela nová optimalizovaná implementace distribuovaného systému v jazyce C++ využívající inferenční knihovnu NVIDIA TensorRT, více v kapitole 6.3.

6 Experimenty

V této kapitole jsou prezentovány úspěšnosti detektorů postavených na architekturách VGG16 a PVA, dále jsou prezentovány výsledky s využitím navrhovaných změn tvorby datové sady pro učení meta architektury Faster R-CNN na architektuře PVA a také zvýšení efektivity detektoru z hlediska rychlosti a úspory paměti při aplikaci optimalizací.

6.1 Sestava a definice experimentů

Pro učení a další experimenty byla využita infrastruktura společnosti RCE systems s.r.o. Pro učení byl použit stroj s následující konfigurací:

- OS Ubuntu 16.04 LTS
- 4x NVIDIA GTX 1080ti 11 GB
- AMD Ryzen 7 1700
- 32 GB DDR4
- Samsung Evo 850 500 GB SSD

Učení neuronových sítí probíhalo s využitím framework Caffe a pro tvorbu datové sady pro klasifikátor ve formátu LMDB byl použit nástroj NVIDIA Digits. V rámci této kapitoly je popsán protokol učení, kterým bylo dosaženo maximální úspěšnosti detektoru.

Pro testování natrénovaných neuronových sítí byla využita stejná infrastruktura, avšak postačily pouze 2 grafické karty NVIDIA GTX 1080ti. Úspěšnost detektoru byla měřena na testovací datové sadě, která je složena z 14 anotovaných videí, které byly vyčleněny z trénovací množiny. Dále bylo z každého testovacího videa vygenerováno 750 snímků pro vizualizaci výsledků. Počet snímků odpovídá 30 sekundám při vzorkovací frekvenci 25 snímků/s. Na testovacích snímcích byly rozmazány regiony, kde se nacházely neanotované objekty, aby negativně neovlivňovaly validaci. Pro měření úspešnosti a vizualizaci výsledků byly naimplementovány příslušné obsluhy neuronové sítě v jazyce Python a C++ v podobě distribuovaného systému s využitím knihovny zeroMQ. Pro porovnání úspešností natrénovaných detektorů byla zvolena metrika $F_{0,5}$ a F_1, kdy finální detektor byl vybrán dle maximální průměrné hodnoty metriky $F_{0,5}$, jelikož bylo stanoveno, že falešně pozitivní detekce snižují skóre více než falešně negativní detekce. Vztahy (21) a (22) definují metriky F_1 a $F_{0,5}$.

$$F_1 = 2 * \frac{prec*rec}{prec+rec} \quad (21)$$

$$F_{0,5} = 1{,}25 * \frac{prec*rec}{(0{,}25*prec)+rec} \quad (22)$$

Jako pozitivní detekce jsou považovány ty, které mají IoU s anotovaným bouding boxem alespoň 0,75. Detekce se skórem nižším než 0,5 se filtrují ještě před procesem validace.

V rámci experimentů byl detektor optimalizován aplikací singulárního rozkladu (SVD), technikou odstranění batch normalizace a s využitím knihovny NVIDIA TensorRT, což přineslo výrazné zvýšení rychlosti a úspory grafické paměti při zachování úspešnosti.

6.2 Protokol učení

V této kapitole je popsán protokol učení neuronové sítě, který byl použit pro dosažení prezentované úspěšnosti výsledného řešení detektoru dopravních prostředků. Jako vstupní předpoklad tohoto protokolu je mít již vytvořenou datovou sadu, jak bylo popsáno v kapitole 2.

6.2.1 Klasifikátor

Při trénování detektoru postaveného na meta architektuře Faster R-CNN je téměř nutné mít smysluplně naučené váhy pro počáteční inicializaci extraktoru konvolučních deskriptorů, aby RPN již od první iterace učení konvergovala. Byla snaha, aby byla neuronová síť již od počátku trénována na datech stejného charakteru, se kterými bude následně detektor pracovat, a proto byla vytvořena datová sada pro klasifikaci objektů z datové sady DataFromSky. Klasifikační datová sada byla generována tak, aby splňovala podmínky neměnnosti poměru stran objektů, avšak rozměry objektů byly náhodně změněny, aby delší strana objektu byla v rozsahu 92 až 192 pixelů. Takto vzniklé objekty byly umístěny do středu černého čtverce o velikosti 192x192. Výsledná datová sada pro klasifikaci byla tvořena 2 392 081 vzorky. Obrázek 32 zobrazuje ukázku vzorků z klasifikační datové sady.

Jelikož byla využita neuronová síť PVA, kterou autoři trénovali na datové sadě ImageNet, která je tvořena 1000 třídami, bylo možné inicializovat váhy trénovaného klasifikátoru právě z naučených vah na ImageNet a dále dotrénovat na vzniklé klasifikační datové sadě. Klasifikátor byl trénován s využitím Stochastic Gradient Descent (SGD) a distribuovaným učením popsaným v kapitole 5.1.2 při počátečním koeficientu učení 0,01 snižovaném vždy, když se hodnota ztrátové funkce na určitý počet iterací příliš nezměnila, po 25 epoch s efektivním batch 256. Výsledkem byla neuronová síť s úspěšností 92,33 % TOP-1 skóre a 99,3 % TOP-3 skóre.

Obrázek 32: Ukázka vzorků z datové sady pro trénování klasifikátoru.

6.2.2 Negativní učení

Proces trénování samotného detektoru byl rozdělen na tzv. negativní učení, kdy byla všechna rozšíření popsána v kapitole 5.2 deaktivována až na distribuované učení a bylo využito pouze originální funkčnosti meta architektury Faster R-CNN. Negativním učením je tato fáze nazvána, jelikož zastoupení pozitivních vzorků (popředí) na jednotlivých snímcích trénovací datové sady není dostatečný, aby naplnil podmínky poměru 1:4 popředí:pozadí, a tedy se batch doplňuje vzorky pozadí. Neuronová síť se spíše učí, co je pozadí a RPN se snaží učit spíše to, že daný vzorek je pozadí, než že je tento vzorek popředím. Obrázek 33 zobrazuje ukázku celého snímku s anotacemi objektů využitého při negativním učení.

Obrázek 33: Ukázka snímku z datové sady využité při negativním trénování. Červeným obdélníkem jsou označeny anotace.

Detektor byl trénován s využitím SGD a distribuovaným učením Faster R-CNN s velikostí snímku 1088x1920, jelikož architektura PVA vyžaduje velikost vstupního vzorku dělitelnou 32. Koeficient učení byl nastaven na 0,005 a snižován 10násobně vždy po 22 epochách, kdy bylo trénováno po 70 epoch. Výsledkem byl detektor, který má RPN dobře naučenou, avšak klasifikátor díky malému zastoupení tříd popředí nebyl příliš dobře natrénovaný a mnoho objektů klasifikoval jako pozadí, případně velice často klasifikoval objekty pouze jako malé vozidlo.

6.2.3 Pozitivní učení

V této fázi učení detektoru byla aktivována téměř všechna vylepšení popsána v kapitole 5.2 bez distribuovaného učení a normalizace tříd v batch. Jako počáteční inicializace byly použity váhy získané z negativního učení. Detektor byl trénován se stejným nastavením hyper parametrů, jako tomu bylo u negativního učení, avšak nyní bylo trénování ukončeno po 25 epochách. Výsledkem byl detektor s $F_{0,5}$ skóre 0,73, kdy skóre bylo poníženo hlavně vyšším množstvím falešně pozitivních detekcí. Tento detektor je prezentován v kapitole 6.4 s výsledky pod názvem PVA.

6.2.4 Zpřesňující učení

Během zpřesňujícího učení byly využity jen a pouze ručně opravené a zpřesněné anotace. V rámci negativního ani pozitivního učení nebyly použité anotace, které byly ručně opraveny a zpřesněny. Pro počáteční inicializaci vah byl použit výstup z pozitivního učení. Byl změněn poměr mezi pozitivními a negativními vzorky na 1:3 a celkově navýšen batch ze 64 na 128 vzorků. Dále u všech batch normalizací bylo zastaveno učení. Finální dotrénování proběhlo s využitím SGD a s počátečním koeficientem učení 0,001 na 1 epochu, což činilo asi 25 000 iterací, kdy koeficient učení byl snižován 10x každých 5 000 iterací. Výsledkem byl detektor s $F_{0,5}$ skóre 0,89. Tento detektor je prezentován v kapitole 6.4 s výsledky pod názvem PVA dotrénováno.

6.3 Optimalizace

Na natrénovanou neuronovou síť byly v rámci experimentů aplikovány následující optimalizace:

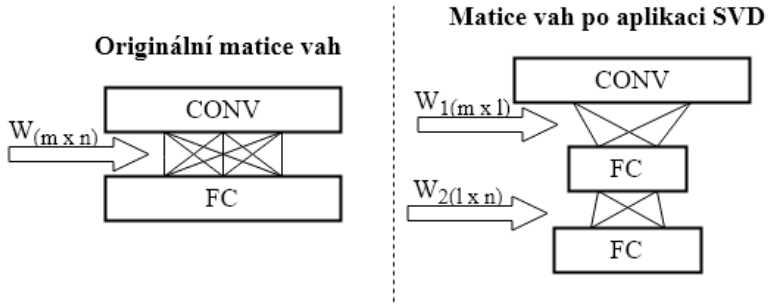
Singulární rozklad

Singulární rozklad (SVD) je operace z oblasti pokročilé lineární algebry a jedná se o faktorizaci matice reálných či komplexních čísel. V teorii neuronových sítí lze SVD využít pro kompresi matice vah plně propojených vrstev. Detailně o aplikaci SVD při optimalizaci neuronových sítí pojednává článek autora Jian Xue [31] a obecně o aplikaci SVD článek autora Virginia C. Klema [32].

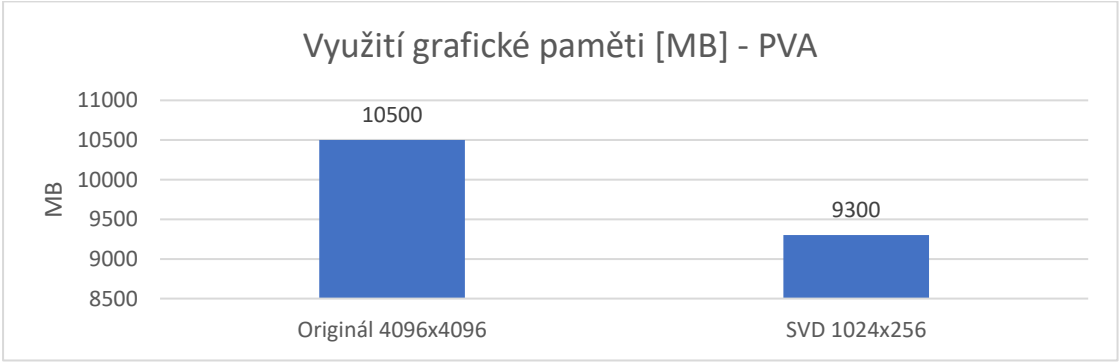
V jednoduchosti lze říci, že SVD rozloží matici vah $W_{(m*n)}$ o rozměrech $m * n$ na tři matice $U_{(m*m)}$, $\sum_{(m*n)}$ a $V^T_{(n*n)}$, kdy matice $W_{(m*n)}$ reprezentuje váhy dané plně propojené vrstvy, kdy m značí velikost vstupního vektoru dat a n značí počet neuronů dané plně propojené vrstvy. Například pro plně propojenou vrstvu, jejíž vstupní data jsou výstupem konvolučního operátoru se m rovná $c * w * h$, kde c značí počet konvolučních jader operátoru, w šířku a h výšku výstupních dat operátoru. Vztah (23) vyjadřuje aproximaci původní matice vah $W_{(m*n)}$, kdy ignorováním velice malých singulárních hodnot lze docílit redukce rozměrů matice, kde l je počet ponechaných singulárních hodnot a $W_{1(m*l)}$ a $W_{2(l*n)}$ jsou nové matice vah.

$$\begin{aligned} W_{(m*n)} &= U_{(m*n)} * \sum_{(m*n)} * V^T_{(n*n)} \\ &\approx U_{(m*l)} * (\sum_{(l*l)} * V^T_{(l*n)}) \\ &= W_{1(m*l)} * W_{2(l*n)} \end{aligned} \qquad (23)$$

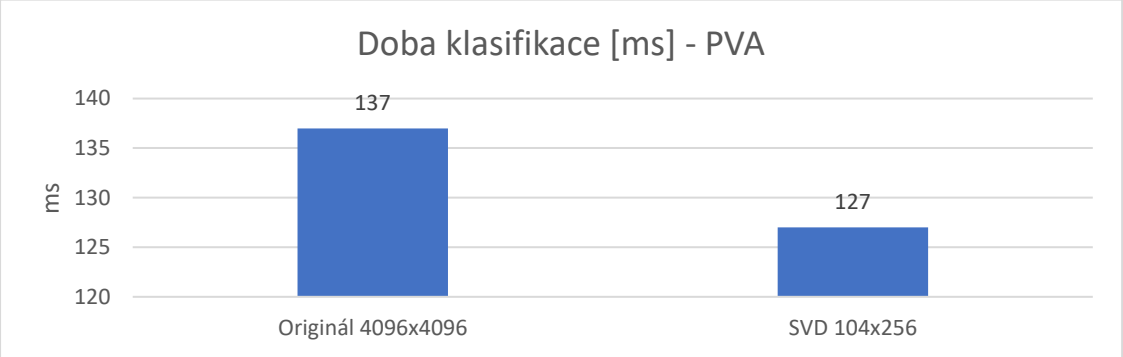
Obrázek 34 zobrazuje transformaci neuronové sítě dle získaných matic vah po aplikaci SVD a Graf 7, Graf 8 a Graf 9 zobrazují postupně využití grafické paměti, dobu klasifikace a úspěšnost neuronové sítě PVA při úloze klasifikace objektů datové sady DataFromSky před a po aplikaci singulárního rozkladu.



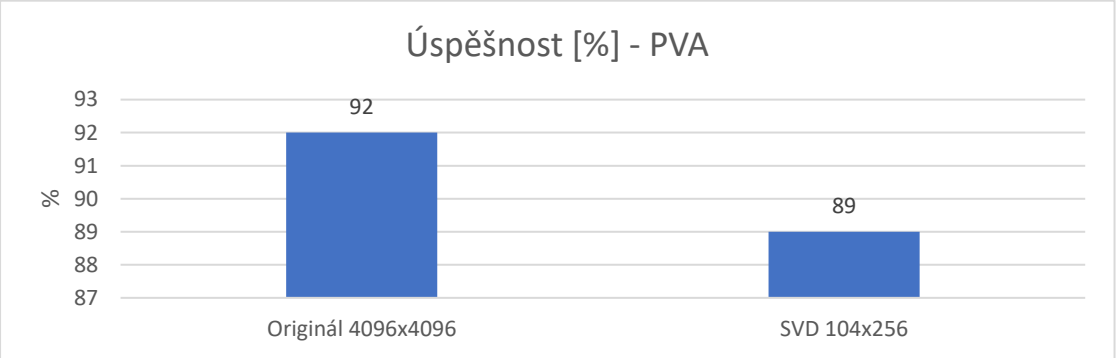
Obrázek 34: Schéma zavedení komprese matice vah s využitím SVD do neuronové sítě.



Graf 7: Využití grafické paměti v MB neuronové sítě PVA před a po aplikaci SVD.



Graf 8: Doba klasifikace neuronové sítě v milisekundách PVA před a po aplikaci SVD.



Graf 9: Úspešnost neuronové sítě PVA při klasifikaci objektů datové sady DataFromSky před a po aplikaci SVD.

Odstranění batch normalizace

Batch normalizace [33] je technika, která řeší fenomén vnitřního kovariantního posuvu vyskytujícího se při trénování neuronových sítí. Tento fenomén je způsobován změnou distribuce dat během učení neuronové sítě vlivem změny učících se parametrů vrstev, což vyžaduje trénování neuronových sítí s velice nízkým koeficientem učení, pečlivou počáteční inicializací učících se parametrů a vhodnou volbu aktivačních funkcí. Operátor batch normalizace zavádí normalizaci dat v rámci neuronové sítě a batch, čímž posunuje distribuci dat do definovaného prostoru a umožňuje trénovat neuronovou síť s mnohem větším koeficientem učení, zpomaluje přetrénování a zvyšuje rychlost konvergence sítě.

V případě normalizace vrstvy, jejíž vstup má d dimenzí $x = (x^1 \dots x^d)$, dle vztahu (24), kdy průměrná hodnota E a směrodatná odchylka Var je spočtena pro celou datovou sadu, je zjištěno, že dojde k urychlení konvergence neuronové sítě při učení [34].

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \qquad (24)$$

Je nutné však zajistit, aby nedošlo ke změně reprezentace dané vrstvy a transformace umožňovala modelovat identitu. Proto byly zavedeny pro každou aktivaci $x^{(k)}$dva parametry $\gamma^{(k)}$ a $\beta^{(k)}$, které násobí a posunují normalizovanou hodnotu aktivace $\hat{x}^{(k)}$ dle vztahu (25).

$$y^{(k)} = \gamma^{(k)} * \hat{x}^{(k)} + \beta^{(k)} \quad (25)$$

V případě nastavení parametrů $\gamma^{(k)} = \sqrt{Var[x^{(k)}]}$ a $\beta^{(k)} = E[x^{(k)}]$ je podmínka identity zajištěna. Nicméně toto řešení předpokládá trénování neuronové sítě metodou Gradient Descent, který pracuje s celou datovou sadou. V praxi není možné Gradient Descent použít, a proto se používá aproximace v podobě Stochastického Gradient Descent. Je nutné tedy přeformulovat výše uvedené vztahy (24) a (25) při použití batch. Jelikož je normalizace aplikována nezávisle na každou aktivaci, nebude dále uváděn index (k) z důvodu zjednodušení zápisu.

Předpokládejme batch B = $\{x_1 \dots x_m\}$ obsahující m aktivací. Dále mějme normalizované aktivace $\hat{x}_1 \dots \hat{x}_m$ a jejich lineární transformace $y_1 \dots y_m$, pak vztahy (26), (27), (28) a (29) definují postupně průměrnou hodnotu aktivací v rámci batch, rozptyl aktivací v rámci batch, normalizované aktivace v rámci batch s konstantou ϵ pro zamezení dělení 0 a lineární transformaci normalizovaných aktivací pomocí parametrů γ a β, které se učí v rámci procesu trénování neuronové sítě. Dohromady tyto vztahy představují batch normalizaci, kdy celá batch normalizace je diferencovatelná, a je tedy možné využít tuto transformaci v rámci neuronové sítě učené klasickou metodou back propagation. Výsledkem učení parametrů γ a β je afinní transformace y, která modeluje identitu, jak tomu bylo u vztahu (25) realizující afinní transformaci díky nastavení parametrů γ a β dle aktivací v rámci celé datové sady.

$$\mu_B = \frac{1}{m} * \sum_{i=1}^{m} x_i \quad (26)$$

$$\sigma_B^2 = \frac{1}{m} * \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad (27)$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (28)$$

$$y_i = \gamma * \hat{x}_i + \beta \quad (29)$$

V rámci neuronové sítě je batch normalizace aplikována před nelineární aktivační funkcí g jako ReLU nebo Sigmoida dle vztahu (30).

$$z = g(BN(W * u)) \quad (30)$$

Nutno poznamenat, že průměrná hodnota μ_B je využita při výpočtu klouzavého průměru μ a rozptyl σ_B^2 při výpočtu celkového rozptylu σ^2 celé datové sady v průběhu učení, aby bylo dosaženo ekvivalentního významu vztahu (24).

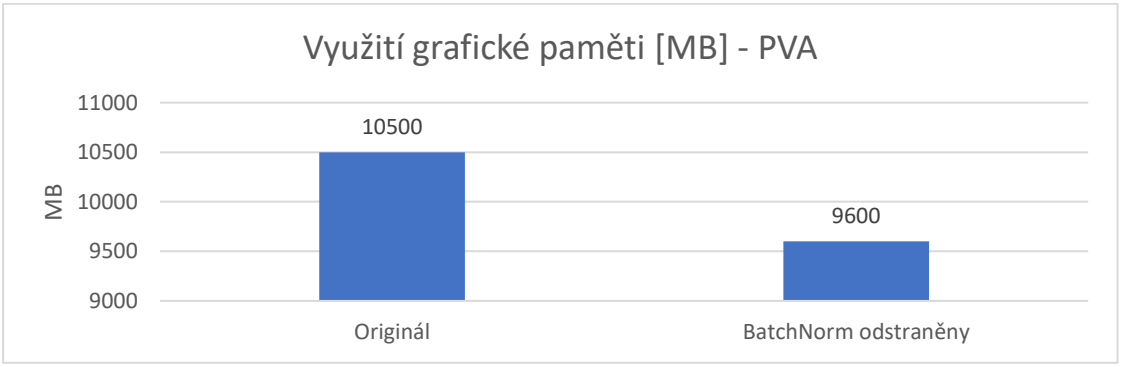
Ve chvíli, kdy je neuronová síť natrénována a připravena k inferenci, je možné dobu inference a spotřebu paměti neuronové sítě snížit odstraněním operátorů batch normalizace provedením této normalizace přímo na naučených váhách předcházejícího operátoru, který podléhá normalizaci. Vztahy (32) a (33) vyjadřují normalizaci každé váhy $w^{(k)}$ a bias $b^{(n)}$ každého neuronu operátoru podléhající batch normalizaci.

$$\alpha^{(k)} = \frac{\gamma^{(k)}}{\sqrt{\sigma^2+\epsilon}} \quad (31)$$

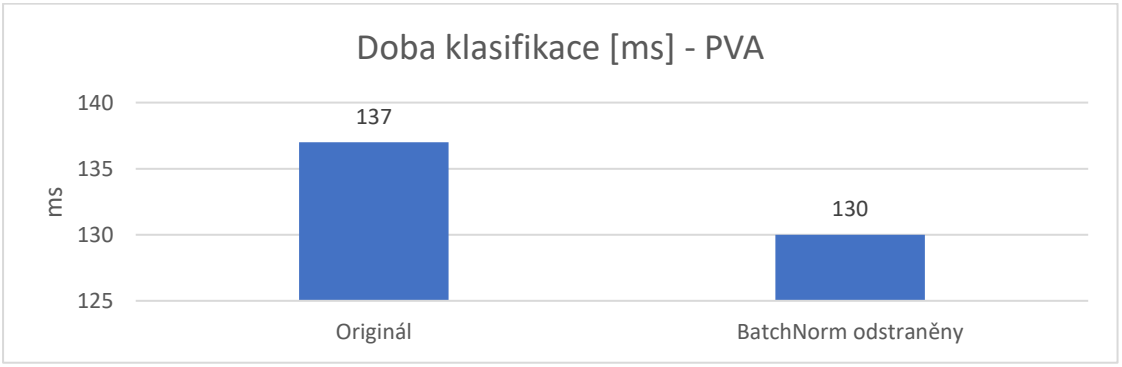
$$b^{(n)} = b^{(n)} * \alpha^{(k)} + (\beta^{(k)} - \mu * \alpha^{(k)}) \quad (32)$$

$$w^{(k)} = w^{(k)} * \alpha^{(k)} \quad (33)$$

Úspěšnost neuronové sítě PVA při klasifikaci objektů datové sady DataFromSky nebyla odstraněním batch normalizací nijak ovlivněna. Graf 10 a Graf 11 zobrazují postupně využití grafické paměti a doby inference neuronové sítě PVA při úloze klasifikace objektů datové sady DataFromSky před a po odstranění batch normalizací.



Graf 10: Využití grafické paměti v MB neuronové sítě PVA před a po aplikaci odstranění batch normalizací.

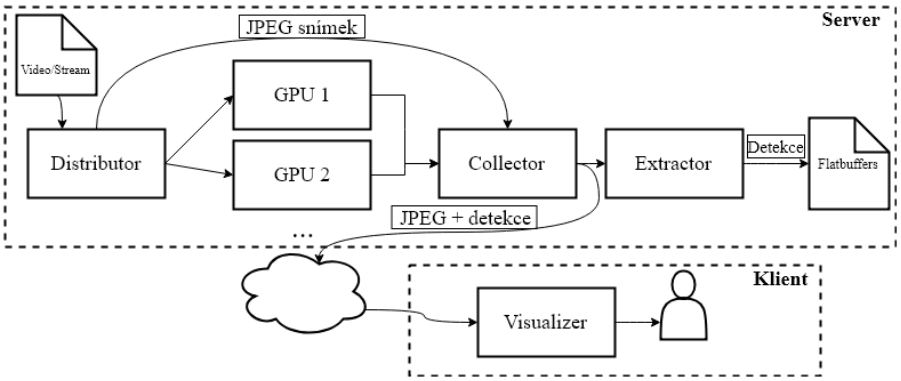


Graf 11: Doba klasifikace v milisekundách neuronové sítě PVA před a po aplikaci odstranění batch normalizací.

Implementace distribuovaného systému inference s využitím NVIDIA TensorRT

Pro dosažení maximální propustnosti a celkové rychlosti detekce na jeden snímek byl implementován v jazyce C++ distribuovaný systém v podobě zřetězené linky s využitím soketové komunikace prostřednictvím knihovny zeroMQ. Každý uzel byl optimalizován z hlediska výpočetní náročnosti. Byla snaha, aby uzel realizující detektor vykonával pouze nejnutnější operace, jelikož se jedná o úzké hrdlo systému. Kompletní dokumentace k distribuovanému systému se nachází v elektronické příloze. Obrázek 35 zobrazuje schéma implementovaného distribuovaného systému, který je složen z následujících uzlů:

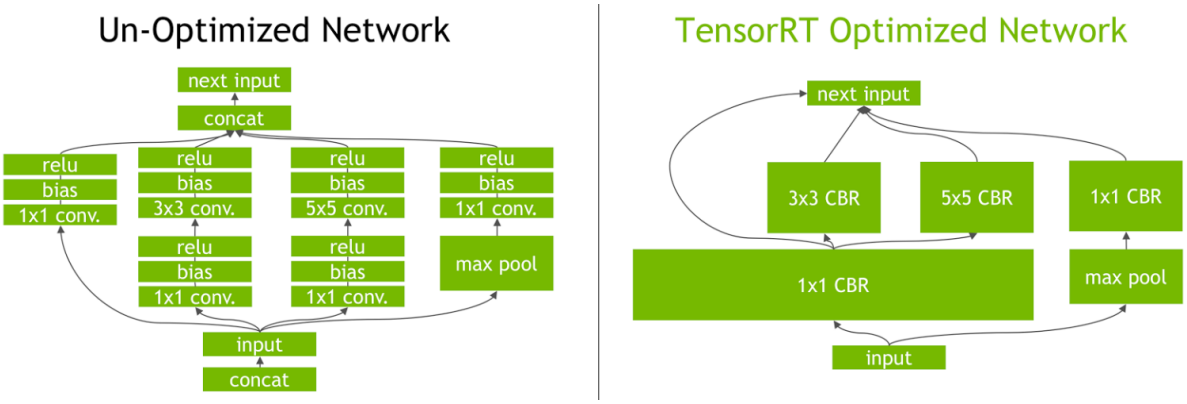
- Distributor
  - Jeho úkolem je načíst data z videa nebo RSTP streamu, rozdělit snímek na části podle počtu uzlů GPU a odeslat každému uzlu jeho část snímku. Současně na celý snímek aplikuje silnou JPEG kompresi, zmenší rozměry snímku a odešle jej na uzel Collector.
- Gpu
  - Uzel realizující detektor přijme část obrázku a transformuje ho pro potřeby neuronové sítě, případně snímek rozdělí na menší. Provede detekci a surová data vykopírovaná z paměti odešle na uzel Collector.
- Collector
  - Uzel sbírá data od uzlu Distributor a všech uzlů GPU, transformuje surová data z detektoru do patřičných datových struktur a přiřazuje detekce k odpovídajícím snímkům. Ve chvíli, kdy byly ke snímku přiřazeny detekce ze všech jeho částí, je komprimovaný snímek i se všemi detekcemi publikován prostřednictvím počítačové sítě.
- Extractor
  - Uzel se připojí na publikační port uzlu Collector a přijaté detekce filtruje dle skóre a velikosti případně dle oblastí zájmu, aplikuje NMS a výsledné detekce bez komprimovaných snímků serializuje do FlatBuffers [36] pro budoucí použití.
- Visualizer
  - Klientská část distribuovaného systému, která má za úkol se připojit na publikační port uzlu Collector a zobrazovat uživateli výsledné detekce. Uživatel má možnost volit parametry NMS, filtrace dle skóre a velikosti. Visualizer uživateli umožňuje nahrávat právě vizualizované detekce včetně komprimovaných JPEG snímků do FlatBuffers, který má stejnou strukturu jako FlatBuffers uzlu Extractor. Visualizer umožňuje takto zaznamenané sekvence přehrávat.



Obrázek 35: Schéma distribuovaného systému pro maximální propustnost detektoru.

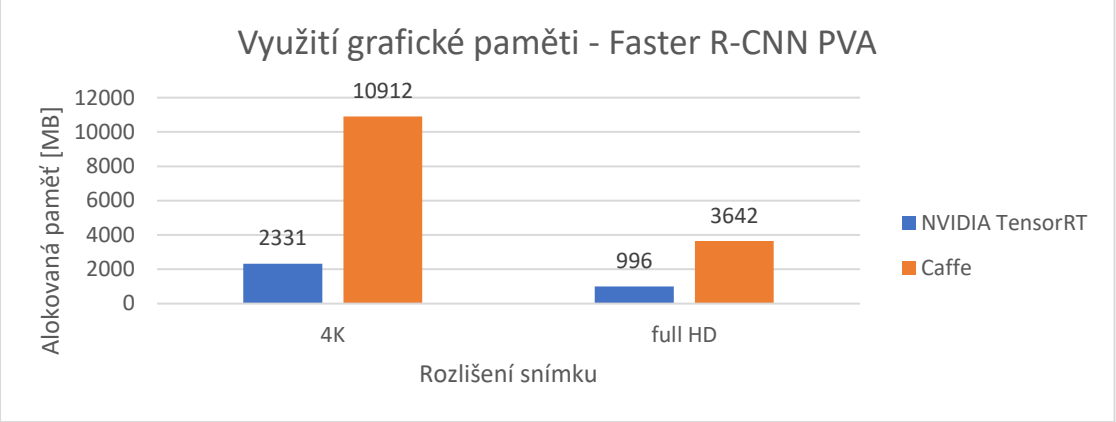
V rámci implementace distribuovaného systému byla pro akceleraci uzlu realizujícího detektor využita knihovna pro optimalizaci a inferenci neuronových sítí NVIDIA TensorRT. Tato knihovna je volně dostupná, nicméně nemá volně dostupné zdrojové kódy a dokumentace ke knihovně není příliš detailní, a proto jsou zde uvedeny pouze základní informace o knihovně.

Nvidia TensorRT podporuje modely z mnoho běžných frameworků jako Caffe, TensorFlow, Keras atd. Hlavní úlohou knihovny je optimalizace neuronové sítě aplikací různých metod, jako např. slučování vrstev (Obrázek 36), kdy lze sloučením dosáhnout až třetinového počtu vrstev sítě, což vede k vyšší propustnosti a úspoře paměti. Lze se domnívat, že knihovna aplikuje i dvě výše uvedené optimalizace v podobě odstranění batch normalizací a singulárního rozkladu matic vah. Knihovna dále umožňuje inferenci neuronových sítí v 32 bitových reálných číslech, 16 bitových reálných číslech, a dokonce v 8 bitových celých číslech, nicméně v rámci této práce nebylo optimalizace datových typů využito z důvodu nedostupnosti potřebného hardwaru. Knihovna také zajistí efektivní využití grafické karty, jelikož dokáže provádět inferenci několika batch současně díky asynchronnímu zpracování a bufferování. Knihovna je primárně vyvíjena pro podporu tenzor jader profesionálních výpočetních grafických karet řady Nvidia Tesla V, na kterých lze dosáhnout až 18x vyšší rychlosti. Zásadní nevýhodou této knihovny je omezení velikosti vektoru deskriptorů na 2^{31}-1 bytů.

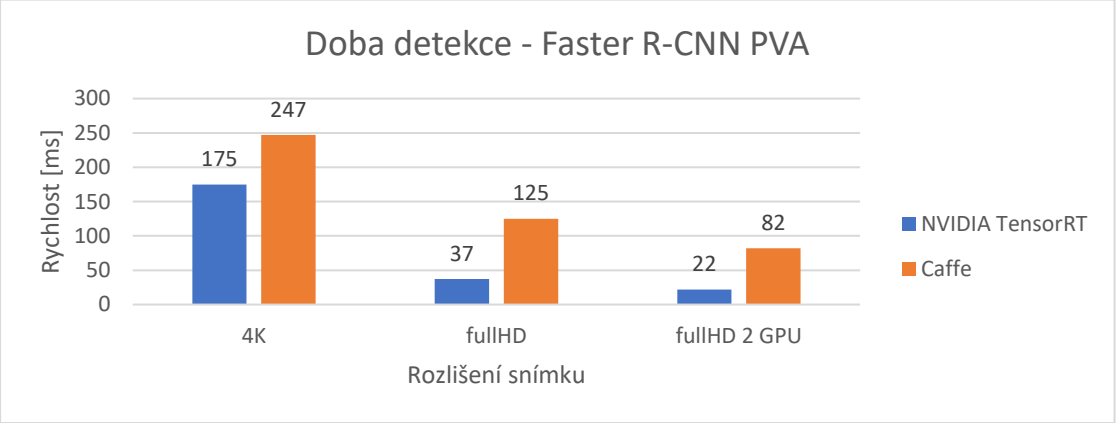


Obrázek 36: Schéma znázorňující slučování vrstev neuronové sítě jako hlavní optimalizaci knihovny NVIDIA TensorRT. Převzato z dokumentace NIVIDA TensorRT [34].

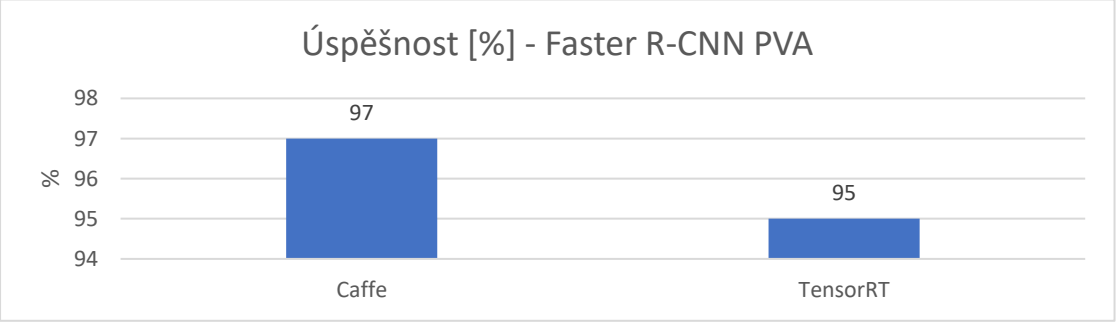
Graf 12 a Graf 13 porovnává framework Caffe a knihovnu NVIDIA TensorRT z hlediska využití grafické paměti a rychlosti detekce. Při použití knihovny NVIDIA TensorRT bylo dosaženo 3násobné rychlosti oproti frameworku Caffe a téměř 4násobné úspoře grafické paměti. Při zapojení 2 uzlů realizujících detektor bylo dosaženo rychlosti 45 full HD snímků za sekundu.



Graf 12: Využití grafické paměti detektoru postaveného na Faster R-CNN a PVA při použití frameworku Caffe a knihovny NVIDIA TensorRT.



Graf 13: Doba detekce detektoru postaveného na Faster R-CNN a PVA při použití frameworku Caffe a knihovny NVIDIA TensorRT.



Graf 14: Úspěšnost detektoru postaveného na Faster R-CNN a PVA při použití frameworku Caffe a knihovny NVIDIA TensorRT.

6.4 Výsledky

Nejvyššího průměrného $F_{0,5}$ skóre 0,89 bylo dosaženo pomocí neuronové sítě, která byla dotrénována na ručně opravených datech. Úspěšnosti neuronových sítí VGG16, PVA při trénování pouze na generovaných datech a PVA dotrénované na ručně opravených anotacích shrnuje Tabulka 1.

Pro porovnání neexistuje příliš mnoho prací řešících stejnou úlohu na podobných datech, možná právě z důvodu nedostupnosti datových sad. Nicméně několik prací lze nalézt. Jedna z nich se zabývala detekcí objektů z dronu pomocí embedded zařízení přímo na dronu při co nejvyšší rychlosti, výsledkem práce je neuronová síť pojmenovaná jako DroNet [37], která využívá meta architektury YOLO. Prezentované výsledky uvádějí úspěšnost okolo 95 %. Druhou prací [38] podobného charakteru je detekce lodí a člunů na moři pomocí dronu. Práce využívá sliding windows přístupu a generování oblastí na způsob regionů zájmů. Dle výsledků prezentovaných v práci bylo dosaženo úspěšnosti 99,4 %, nicméně doba zpracování jednoho full HD snímku trvala 1,8 až 10,5 sekundy.

Výstupy této práce v podobě videí se zobrazenou úspešností lze najít v elektronické příloze. Názvy videí vycházejí z interního pojmenování.

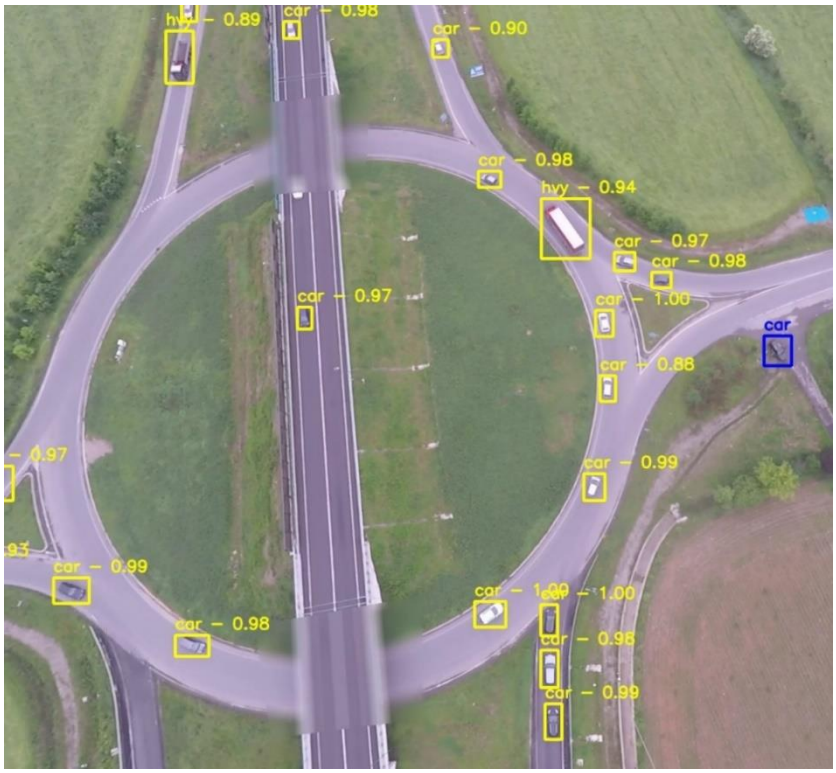
Síť	**VGG16**				**PVA**				**PVA dotrénováno**			
Video	prec	rec	F_1	$F_{0,5}$	prec	rec	F_1	$F_{0,5}$	prec	rec	F_1	$F_{0,5}$
cowild40	0,46	0,52	0,49	0,47	0,64	0,64	0,64	0,64	0,73	0,54	0,62	**0,68**
cowild46	0,73	0,72	0,72	0,73	0,57	0,74	0,64	0,6	0,91	0,80	0,85	**0,89**
cowild51	0,92	0,54	0,68	0,81	0,86	0,61	0,71	0,86	0,97	0,65	0,78	**0,88**
cowild54	0,78	0,72	0,75	0,77	0,85	0,89	0,87	0,86	0,95	0,87	0,91	**0,93**
cowild56-1	0,78	0,96	0,86	0,81	0,56	0,94	0,7	0,61	0,89	0,94	0,91	**0,90**
cowild56-2	0,91	0,77	0,83	0,88	0,35	0,9	0,51	0,4	0,98	0,92	0,95	**0,97**
cowild57-1	0,53	0,78	0,63	0,57	0,7	0,4	0,51	0,61	0,76	0,45	0,57	**0,67**
cowild60	0,89	0,79	0,83	0,86	0,72	0,87	0,79	0,75	0,96	0,91	0,93	**0,95**
marcold1	0,95	0,84	0,89	0,93	0,88	0,95	0,92	0,89	0,97	0,96	0,97	**0,97**
marcold4	0,94	0,87	0,90	0,92	0,95	0,94	0,94	0,95	0,98	0,96	0,97	**0,98**
marcold6-1	0,86	0,76	0,81	0,84	0,59	0,90	0,72	0,64	0,94	0,89	0,91	**0,93**
marcold7-1	0,88	0,90	0,89	0,88	0,84	0,89	0,86	0,85	0,92	0,90	0,91	**0,91**
marcold7-2	0,96	0,89	0,92	0,94	0,90	0,88	0,89	0,90	0,99	0,92	0,95	**0,97**
marcold8	0,52	0,42	0,46	0,50	0,80	0,78	0,79	0,80	0,89	0,73	0,80	**0,85**

Tabulka 1: Úspěšnosti detektorů VGG16, PVA (bez dotrénování), PVA dotrénováno (na ručně opravených anotacích)

Ukázky výsledných detekcí zobrazuje Obrázek 37 a Obrázek 38, které byly pořízeny jako výstřižky z videí prezentující úspešnost detektoru. Žluté obdélníky značí úspěšnou detekci, modré obdélníky

anotované ale nedetekované objekty a červené obdélníky označují falešně pozitivní detekce. Nutno poznamenat, že detekované objekty (auta, cyklisti …), které nebyly anotovány, jsou považovány při validaci za falešně pozitivní a jsou označeny červeným obdélníkem, což negativně ovlivňuje výsledné skóre detektoru. U každé detekce je připsána třída objetu a skóre detekce.

Obrázek 37: Ukázka detekcí z videa marcoId4.



Obrázek 38: Ukázka detekcí z videa MarcoId7-2.

7 Závěr

V rámci této práce byla natrénována neuronová síť realizující detektor dopravních prostředků s využitím frameworku Caffe, meta architektury Faster R-CNN a extraktoru konvolučních deskriptorů PVA, jelikož tato kombinace meta frameworku a extraktoru vyplynula z analýzy jako nejvhodnější. Samotné analýze předcházel PoC, který hned na začátku práce potvrdil, že neuronová síť je schopna úspěšně řešit problém detekce dopravních prostředků z dronu pomocí sliding window. Dále byly v rámci analýzy prostudovány principy a možnosti několika dalších meta architektur mimo Faster R-CNN: R-CNN, Fast R-CNN, RFCN, SSD a YOLO. Správné výsledky analýzy meta frameworků potvrzují i naměřené výsledky při použití Faster R-CNN s velice uspokojivým průměrným $F_{0,5}$ skóre 0,89.

Celá neuronová síť byla trénována na datové sadě DataFromSky, která byla v rámci této práce vytvořena. Jelikož byla datová sada automaticky generována z dat trackeru, bylo zapotřebí řešit mnoho problémů. Některé se podařilo vyřešit a některé byly ponechány na generalizačních schopnostech neuronové sítě, což se ukázalo z větší míry jako fungující řešení. Přesto byla část datové sady oddělena a ručně opravena pro finální dotrénování detektoru. Celá datová sada DataFromSky obsahuje 7 453 402 objektů rozdělených do 6 tříd, kdy ručně opravených objektů bylo 84 317, což díky zavedeným úpravám do Faster R-CNN stačilo k vylepšení výsledné úspěšnosti detektoru.

Framework Caffe byl obohacen o distribuované učení neuronových sítí, což umožnilo efektivně využít obří datovou sadu a trénovat neuronovou síť na velkém počtu vzorků současně, kdy dostupná infrastruktura umožnila používat 4 grafické karty a celkem 44 GB grafické paměti pro jednu neuronovou síť, což bylo využito především při trénování klasifikátoru objektů datové sady DataFromSky pro počáteční inicializaci detektoru. Mnoho dalších úprav bylo provedeno u meta architektury Faster R-CNN, které v drtivé většině případů vylepšovaly předzpracování a využití datové sady. Jednalo se o zavedení OHEM učení, které umožnilo během procesu učení vybírat trénovací vzorky, u kterých vykazovala neuronová síť chybné detekce. Dále byla zavedena úprava v podobě rozřezávání vstupních snímků na menší z důvodu úspory paměti a jistoty ekvivalence výsledného natrénování na výřezech. Významnou úpravou bylo zavedení tvorby tzv. grid trénovacích snímků, které umožnily efektivně trénovat neuronovou síť z hlediska počtu pozitivních objektů. Grid rozšíření dále vylepšuje z hlediska distribuce tříd objektů normalizace zastoupení tříd. I když získaná datová sada obsahuje velké množství objektů, byla dále implementována dodatečná augmentace vstupních snímků simulující různé povětrnostní podmínky, což umožnilo rozšířit datovou sadu ~10x. Do meta architektury Faster R-CNN bylo také zavedeno distribuované učení využívající již implementované distribuované učení u frameworku Caffe, což bylo využito při základním trénování

detektoru. Při trénování detektoru v rámci této práce byla všechna tato rozšíření postupně kombinována a byl nalezen protokol, který vedl k úspěšnému natrénování detektoru.

V rámci této práce byla cílem nejen vysoká úspěšnost, ale i rychlost detekce v reálném čase, čehož bylo díky optimalizacím s využitím NVIDIA TensorRT a implementaci distribuovaného detektoru využívajícího 2 grafické karty dosaženo. Pro full HD snímky byla dosažena dokonce rychlost 45 snímků/s.

Nicméně výsledný detektor trpí určitými nedostatky v podobě horší detekce cyklistů a chodců. Důvodem je příliš malé rozlišení konvolučních deskriptorů pro RPN a pro klasifikátor RoI, proto vyžaduje aktuální řešení několik úprav, které pomohou lépe detekovat opravdu malé objekty v podobě cyklistů, chodců a motorek. Tyto úpravy by mohly spočívat v nahrazení reziduálních bloků reziduálními bloky verze 2 [27], které umožňují lepšího využití batch normalizací, dále je zapotřebí odstranit Hyper Feature, jelikož využitá dekonvoluce je problematická při učení a jelikož nedovoluje využít à trous triku s využitím dilated konvolucí [28], které umožní zvětšení rozlišení konvolučních deskriptorů. Nicméně zde existuje další možnost vylepšení, které se nazývá deformovatelné konvoluce [29]. Dalším vylepšením bude kompletní revize PVA, odstranění nepotřebných operátorů a přidání SE bloků [30] pro lepší extrakci konvolučních deskriptorů. Jako vylepšení Faster R-CNN je navrhováno integrovat Atrous RPN [26]. Jako poslední změna při novém trénování detektoru by byla změna podkladové barvy klasifikační datové sady pro získání iniciačních vah na hodnotu průměrné hodnoty jednotlivých barevných složek obrázků datové sady.

Literatura

[1] HUANG, Jonathan, Vivek RATHOD, Chen SUN, et al. Speed/accuracy trade-offs for modern convolutional object detectors [online]. 2016 [cit. 2017-10-29]. Dostupné z: http://arxiv.org/abs/1611.10012

[2] SOMONYAN, Karen a Andreq ZISSERMAN. Very Deep Convolutional Networks for Large-Scale Image Recognition [online]. 2014 [cit. 2017-11-01]. Dostupné z: http://arxiv.org/abs/1409.1556

[3] SZEGEDY, Christian, Wei LIU, Yangqing JIA, et al. Going Deeper with Convolutions [online]. 2014 [cit. 2017-11-05]. Dostupné z: http://arxiv.org/abs/1409.4842

[4] SZEGEDY, Christian, Vincent VANHOUCKE, Sergey IOFFE, Jonathon SHLENS a Zbigniew WOJNA. Rethinking the Inception Architecture for Computer Vision [online]. 2015 [cit. 2017-11-05]. Dostupné z: http://arxiv.org/abs/1512.00567

[5] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Deep Residual Learning for Image Recognition [online]. 2015 [cit. 2017-11-10]. Dostupné z: http://arxiv.org/abs/1512.03385

[6] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Identity Mappings in Deep Residual Networks [online]. 2016 [cit. 2017-11-15]. Dostupné z: http://arxiv.org/abs/1603.05027

[7] MONTÚFAR, Guido, Razvan PASCANU, Kynghyun CHO a Yoshua BENGIO. On the Number of Linear Regions of Deep Neural Networks [online]. 2014 [cit. 2017-11-15]. Dostupné z: https://arxiv.org/abs/1402.1869

[8] KIM, Kye-Hyeon, Yeongjae CHEON, Sanghoon HONG, Byung-Seok ROK a Minje PARK. PVANet: Lightweight Deep Neural Networks for Real-time Object Detection [online]. 2016 [cit. 2017-11-18]. Dostupné z: http://arxiv.org/abs/1608.08021

[9] KONG, Tao, Anbang YAO, Yurong CHEN a Fuchun SUN. HyperNet: Towards Accurate Region Proposal Generation and Joint Object [online]. 2016 [cit. 2017-11-19]. Dostupné z: http://arxiv.org/abs/1604.00600

[10] SHANG, Wenling, Kihyuk SOHN, Diogo ALMEIDA a Honglak LEE. Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units [online]. 2016 [cit. 2017-11-29]. Dostupné z: http://arxiv.org/abs/1603.05201

[11] NEUBECK, A. a L. VAN GOOL. Efficient Non-Maximum Suppression. In: 18th International Conference on Pattern Recognition (ICPR'06) [online]. IEEE, 2006, 2006, s. 850-855 [cit. 2017-12-01]. DOI: 10.1109/ICPR.2006.479. ISBN 0-7695-2521-0. Dostupné z: http://ieeexplore.ieee.org/document/1699659/

[12] GIRSHICK, Ross B., Jeff DONAHUE, Trevor DARRELL a Jitendra MALIK. Rich feature hierarchies for accurate object detection and semantic segmentation [online]. 2013 [cit. 2017-12-05]. Dostupné z: http://arxiv.org/abs/1311.2524

[13] GLOROT, Xavier a Yoshua BENGIO. Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics [online]. 2010, 9(9), 249-256 [cit. 2017-12-13]. Dostupné z: http://proceedings.mlr.press/v9/glorot10a.html

[14] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification [online]. 2015 [cit. 2017-12-15]. Dostupné z: http://arxiv.org/abs/1502.01852

[15] UIJLINGS, J. R. R., K. E. A. VAN DE SANDE, T. GEVERS a A. W. M. SMEULDERS. Selective Search for Object Recognition. International Journal of Computer Vision [online]. 2013, 104(2), 154-171 [cit. 2017-12-15]. DOI: 10.1007/s11263-013-0620-5. ISSN 0920-5691. Dostupné z: http://link.springer.com/10.1007/s11263-013-0620-5

[16] ERDOĞAN, Göksu. Faster R-CNN [online]. [cit. 2017-12-15]. Dostupné z: https://web.cs.hacettepe.edu.tr/~aykut/classes/spring2016/bil722/slides/w05-FasterR-CNN.pdf

[17] GIRSHICK, Ross B. Fast R-CNN [online]. 2015 [cit. 2017-12-15]. Dostupné z: http://arxiv.org/abs/1504.08083

[18] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition [online]. 2014 [cit. 2017-12-16]. Dostupné z: http://arxiv.org/abs/1406.4729

[19] REN, Shaoqing, Kaiming HE, Ross B. GIRSHICK a Jian SUN. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [online]. 2015 [cit. 2017-12-26]. Dostupné z: http://arxiv.org/abs/1506.01497

[20] DAI, Jifeng, Yi LI, Kaiming HE a Jian SUN. R-FCN: Object Detection via Region-based Fully Convolutional Networks [online]. 2016 [cit. 2017-12-26]. Dostupné z: http://arxiv.org/abs/1605.06409

[21] LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott E. REED, Cheng-Ynag FU a Alexander C. BERG. SSD: Single Shot MultiBox Detector [online]. 2015 [cit. 2017-12-26]. Dostupné z: http://arxiv.org/abs/1512.02325

[22] REDMON, Joseph, Santosh K. DIVVALA, Ross B. GIRSHICK a Ali FARHADI. You Only Look Once: Unified, Real-Time Object Detection [online]. 2015 [cit. 2017-12-26]. Dostupné z: http://arxiv.org/abs/1506.02640

[23] Zero MQ [online]. 2007 [cit. 2018-05-01]. Dostupné z: http://zeromq.org/

[24] NVIDIA cuDNN [online]. 2016 [cit. 2018-05-01]. Dostupné z: https://developer.nvidia.com/cudnn

[25] BODLA, Navaneeth, Bharat SINGH, Rama CHELLAPPA a Larry S. DAVIS. Improving Object Detection With One Line of Code [online]. 8.8.2017 [cit. 2018-05-01]. Dostupné z: https://arxiv.org/pdf/1704.04503.pdf

[26] Atrous Faster R-CNN for Small Scale Object Detection. 2017 2nd International Conference on Multimedia and Image Processing (ICMIP) [online]. IEEE, 2017, 2017, 2017(2), 16-21 [cit. 2018-05-03]. DOI: 10.1109/ICMIP.2017.37. ISBN 978-1-5090-5954-6. Dostupné z: http://ieeexplore.ieee.org/document/8221063/

[27] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Identity Mappings in Deep Residual Networks [online]. 25.7.2016 [cit. 2018-05-03]. Dostupné z: https://arxiv.org/pdf/1603.05027.pdf

[28] YU, Fisher a Vladlen KOLTUN. MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS [online]. 30.8.2016 [cit. 2018-05-03]. Dostupné z: https://arxiv.org/pdf/1511.07122v3.pdf

[29] DAI, Jifeng, Haozhi QI, Yuwen XIONG, Yi LI, Guodong ZHANG, Han HU a Yichen WEI. Deformable Convolutional Networks [online]. 5.6.2017 [cit. 2018-05-03]. Dostupné z: https://arxiv.org/pdf/1703.06211.pdf

[30] HU, Jie, Li SHEN a Gang SUN. Squeeze-and-Excitation Networks [online]. 5.8.2018 [cit. 2018-05-03]. Dostupné z: https://arxiv.org/pdf/1709.01507.pdf

[31] XUE, Jian, Jinyu LI a Yifan GONG. Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition [online]. 2013 [cit. 2018-05-04]. Dostupné z: https://pdfs.semanticscholar.org/0eb3/29b3291944ee358fcbad6d26a2e111addd6b.pdf

[32] KLEMA, V. a A. LAUB. The singular value decomposition: Its computation and some applications. IEEE Transactions on Automatic Control [online]. 1980, 25(2), 164-176 [cit. 2018-05-04]. DOI: 10.1109/TAC.1980.1102314. ISSN 0018-9286. Dostupné z: http://ieeexplore.ieee.org/document/1102314/

[33] IOFFE, Sergey a Christian SZEGEDY. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [online]. 2.3.2015 [cit. 2018-05-04]. Dostupné z: https://arxiv.org/pdf/1502.03167.pdf

[34] Neural Networks: Tricks of the Trade [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012 [cit. 2018-05-04]. Lecture Notes in Computer Science. ISBN 978-3-642-35288-1. Dostupné z: http://link.springer.com/10.1007/978-3-642-35289-8_3

[35] Figure 4. In: NVIDIA Developer Blog [online]. [cit. 2018-05-05]. Dostupné z: https://devblogs.nvidia.com/tensorrt-3-faster-tensorflow-inference/

[36] FlatBuffers [online]. [cit. 2018-05-05]. Dostupné z: https://google.github.io/flatbuffers/

[37] KYRKOU, Christos, George PLASTIRAS a Theocharis THEOCHARIDES. DroNet: Efficient Convolutional Neural Network Detector for Real-Time UAV Applications [online]. [cit. 2018-05-06]. Dostupné z: http://cas.ee.ic.ac.uk/people/sv1310/papers/[2018]_date_dronet_efficient_cnn_detector_for_rt_uav_applications.pdf

[38] Aerial Detection in Maritime Scenarios Using Convolutional Neural Networks. Advanced Concepts for Intelligent Vision Systems [online]. Cham: Springer International Publishing, 2016, 2016-10-21, , 373-384 [cit. 2018-05-06]. Lecture Notes in Computer Science. DOI: 10.1007/978-3-319-48680-2_33. ISBN 978-3-319-48679-6. Dostupné z: http://link.springer.com/10.1007/978-3-319-48680-2_33

Seznam grafů

Seznam obrázků

Seznam příloh

Příloha 1. DVD

Na DVD lze najít následující soubory:

- **DVD**
  - **frameworky**
    - **caffe_xhlavo01**
      - *zdrojové kódy frameworku Caffe včetně zavedených úprav*
    - **fater-R-CNN_xhlavo01**
      - *zdrojové kódy meta architektury Faster R-CNN včetně zavedených úprav*
    - **tensorRT_detektor_xhlavo01** (*distribuovaný systém*)
      - **collector**
        - *zdrojové kódy uzlu collector*
      - **distributor**
        - *zdrojové kódy uzlu distributor*
      - **gpu_node**
        - *zdrojové kódy uzlu gpu*
      - **visualizer** (*zdrojové kódy vizualizéru*)
        - **release** (*spustitelná aplikace pro Windows*)
      - *manual.pdf* (*dokumentace k distribuovanému systému*)
  - **neuronové_sítě**
    - *předpisy použitých neuronových sítí*
  - **vizualizace**
    - **architektury**
      - *vizualizace v SVG jednotlivých neuronových sítí a meta architektur*
    - **datové_sady**
      - *README.txt* (*význam jednotlivých obrázků*)
  - **výsledky** (*videa zachycující detekce na testovací sadě včetně úspěšností*)
    - *validace_skóre.xlsx* (*přehled prec, rec a skóre jednotlivých sítí na testovacích videích*)
  - *analýza_datové_sady.xlsx*
  - *optimalizace.xlsx*
  - *xhlavo01_dip.pdf* (*tato práce v elektronické podobě*)