Real-time detekce dopravy pomocí OpenCV

Jan Sáblík

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan Sáblík**
Osobní číslo:	**A19098**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Real-time detekce dopravy pomocí OpenCV**
Téma práce anglicky:	**Real-Time Traffic Detection Using Opencv**

Zásady pro vypracování

1. Proveďte průzkum moderních metod pro zpracování obrazu.
2. Popište metody pro detekci objektů.
3. Představte možnosti knihovny OpenCV.
4. Porovnejte knihovnu OpenCV s alternativami.
5. Navrhněte aplikační řešení s využitím jazyka Python.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. BRADSKI, Gary R. a Adrian KAEHLER. Learning OpenCV. Sebastopol: O'Reilly, 2008, xvii, 555 s. ISBN 9780596516130.
2. PETERS, James F. Foundations of computer vision: computational geometry, visual image structures and object shape detection. Cham, Switzerland: Springer, 2017, 1 online resource (xvii, 431 pages). Intelligent systems reference library. Dostupné z: doi:9783319524832
3. VILLAN, Alberto Fernandez. Mastering OpenCV 4 with Python. Packt Publishing, 2019, 517 s. ISBN 9781789344912.
4. SOLEM, Jan Erik. Programming Computer Vision with Python. O'Reilly Media, 2012, 261 s. ISBN 9781449316549.
5. DEY, Sandipan. _Hands-on image processing with Python: expert techniques for advanced image analysis and effective interpretation of image data_. Birmingham: Packt Publishing, Limited, 2018, 643 s. ISBN 978-178-9343-731.

Vedoucí bakalářské práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17. 5. 2022

Jan Sáblík, v. r.
podpis studenta

ABSTRAKT

Tato bakalářská práce pojednává o knihovně počítačového vidění OpenCV, moderních metodách zpracování obrazu a algoritmech pro detekci objektů. Na začátku teoretické části jsou představeny základy zpracování obrazu a příklady moderních metod. Druhá kapitola je zaměřena na druhy algoritmů pro detekci objektů a jejich princip. Dále jsou shrnuty funkce a algoritmy knihovny OpenCV, které se využívají při různých technikách detekce objektů. Poslední kapitola teoretické části představuje alternativní knihovny počítačového vidění a jejich porovnání s knihovnou OpenCV. V praktické části bakalářské práce je popisována vytvořená aplikace pro analýzu a získávání dat o dopravě v reálném čase za využití moderních algoritmů pro detekci a sledování objektů.

Klíčová slova: OpenCV, detekce objektů, počítačové vidění, zpracování obrazu

ABSTRACT

This bachelor thesis deals with the OpenCV library, modern image processing methods, and algorithms for object detection. At the beginning of the theoretical part, the basics of image processing and examples of modern technologies are introduced. The second chapter focuses on the types of algorithms for object detection and their principles. Furthermore, the functions and algorithms of the OpenCV library, which are used in various object detection techniques, are summarized. The last chapter of the theoretical part presents alternative computer vision libraries and their comparison with the OpenCV library. The practical part of the bachelor's thesis describes the application created for the analysis and acquisition of traffic data in real time using modern algorithms for object detection and tracking.

Keywords: OpenCV, object detection, computer vision, image processing

Na začátek bych chtěl poděkovat vedoucímu práce, panu Ing. Davidu Malaníkovi, Ph.D. za odbornou pomoc, cenné rady a připomínky při zpracovávání teoretické části a vytváření aplikace.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **10**
I TEORETICKÁ ČÁST **11**
1 ZPRACOVÁNÍ OBRAZU **12**
1.1 Úvod 12
1.2 Základní kroky zpracování obrazu 12
1.2.1 Získání obrazu 12
1.2.2 Předzpracování 12
1.2.3 Segmentace 13
1.2.4 Popis a klasifikace objektů 13
1.3 Moderní metody pro zpracování obrazu 14
1.3.1 Tesla Vision 14
1.3.1.1 Princip 14
1.3.2 Deep Learning Super Sampling (DLSS) 16
2 METODY DETEKCE OBJEKTŮ **18**
2.1 Rozdělení algoritmů 18
2.2 Tradiční algoritmy 19
2.2.1 Viola-Jones detektor 19
2.2.1.1 Získání Haarových příznaků 20
2.2.1.2 Vytvoření integrálního obrazu 20
2.2.1.3 Trénování s využitím AdaBoost algoritmu 21
2.2.1.4 Kaskádové klasifikátory 22
2.2.2 Porovnávání šablon 22
2.2.2.1 Naivní porovnávání šablon 22
2.2.2.2 Porovnávání šablon s využitím normované vzájemné korelace 23
2.2.3 Histogram orientovaných gradientů 24
2.2.3.1 Výpočet deskriptoru [16] [17] 24
2.2.4 Optický tok 25
2.2.4.1 Výpočet vektoru optického toku 25
2.3 Algoritmy využívající hluboké učení 27
2.3.1 Dvoufázové a jednofázové neuronové sítě 28
2.3.2 R-CNN 28
2.3.2.1 Fast R-CNN 29
2.3.2.2 Faster R-CNN 30
2.3.3 You Only Look Once (YOLO) 31
2.3.3.1 YOLOv2 32
2.3.3.2 YOLOv3 33
2.3.3.3 YOLOv4 33
3 KNIHOVNA OPENCV **34**
3.1 Vybrané metody a funkce 35
3.1.1 Základní operace 35
3.1.2 Funkce pro kreslení 36
3.1.3 Práce s videem 38
3.1.4 Zpracování obrazu 39
3.1.4.1 Vyhlazování obrazu 39
3.1.4.2 Morfologické operátory 39

3.1.4.3 Prahovací funkce ... 40
3.1.4.4 Detekce hran ... 41
3.1.5 Detekce příznaků a objektů ... 42
3.1.6 Strojové učení – DNN modul ... 42
4 ALTERNATIVY KNIHOVNY OPENCV ... 43
4.1 SIMPLECV ... 43
4.1.1 Výhody ... 43
4.1.2 Nevýhody ... 43
4.2 MATLAB ... 43
4.2.1 Computer Vision Toolkit ... 44
4.2.2 Výhody ... 44
4.2.3 Nevýhody ... 45
4.3 BOOFCV ... 45
4.3.1 Výhody ... 45
4.3.2 Nevýhody ... 46
4.4 SHRNUTÍ ... 46
II PRAKTICKÁ ČÁST ... 47
5 PŘEDSTAVENÍ APLIKACE ... 48
5.1 VYUŽITÉ TECHNOLOGIE ... 48
5.1.1 OpenCV 4.5.5-dev ... 48
5.1.1.1 GStreamer ... 49
5.1.1.2 CUDA ... 50
5.1.2 CamGear ... 50
5.1.3 YOLOv4 ... 51
5.1.4 Deep SORT ... 51
5.1.4.1 TensorFlow ... 52
5.1.5 Flask ... 52
5.1.6 Dash ... 52
5.1.7 Microsoft SQL Server ... 53
5.1.8 Pandas ... 53
5.2 POPIS FUNKCE ... 53
5.2.1 Získávání obrazu ... 55
5.2.2 Detekce vozidel ... 56
5.2.3 Sledování a počítání vozidel ... 58
5.2.4 Ukládání dat ... 61
5.3 UŽIVATELSKÉ ROZHRANÍ ... 63
5.3.1 Nastavení ... 65
5.3.2 Vytváření masky ... 65
5.3.3 Vytváření detekčních oblastí ... 66
ZÁVĚR ... 67
SEZNAM POUŽITÉ LITERATURY ... 68
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK ... 75
SEZNAM OBRÁZKŮ ... 77
SEZNAM ZDROJOVÝCH KÓDŮ ... 79
SEZNAM PŘÍLOH ... 80

PŘÍLOHA P I: OBSAH CD 81

ÚVOD

Metody z oblasti počítačového vidění v dnešní době umožňují automatizaci a usnadnění velkého množství úkolů. Díky rostoucímu výpočetnímu výkonu procesorů a grafických karet se využívá stále pokročilejších a výpočetně náročnějších algoritmů, které můžou sloužit například pro detekci objektů v reálném čase.

Vývoj algoritmů pro detekci objektů se dá rozdělit do dvou období. První období začíná téměř v počátcích počítačového vidění a sahá zhruba do roku 2014. Patří do něj především tradiční algoritmy jako Viola-Jones, nebo HOG. Období po roce 2014 se právě díky zvyšujícímu výpočetnímu výkonu nese ve znamení hlubokého učení a neuronových sítí, které přináší přesné detekce mnoha tříd objektů v rámci jednoho modelu a také vysokou snímkovou frekvenci umožňující nasazení v aplikacích pracujících v reálném čase.

OpenCV je v současnosti nejpoužívanější knihovna počítačového vidění, a to především díky své obsáhlosti. Je zde implementována většina existujících algoritmů a funkcí pro zpracování obrazu a počítačové vidění. Knihovna je dostupná v mnoha jazycích, například C++, Python, Java, nebo JavaScript.

Bakalářská práce je zaměřena na detekci dopravy v reálném čase za využití moderních algoritmů. V teoretické části jsou nejprve popsány pojmy z oblasti zpracování obrazu, pipeline a v závěru kapitoly jsou představeny dvě moderní metody pro zpracování obrazu. Konkrétně se jedná o Tesla Vision a NVIDIA DLSS. Druhá kapitola je zaměřena na algoritmy pro detekci objektů a jejich členění. Třetí kapitola popisuje knihovnu OpenCV a její části, které jsou využity v praktické části práce, nebo v oblasti detekce objektů obecně. Poslední kapitola teoretické části představuje alternativní knihovny počítačového vidění a porovnává je s knihovnou OpenCV. V praktické části práce popsána vytvořená aplikace pro analýzu a získávání dat o dopravě v reálném čase, která je určena jako nástroj pro získání přehledu o dopravní situaci v dané oblasti a poskytuje data pro další využití, například vytváření dopravních modelů, nebo plánování uzavírek.

I. TEORETICKÁ ČÁST

1 ZPRACOVÁNÍ OBRAZU

1.1 Úvod

Obraz můžeme vnímat jako 2D zobrazení 3D světa. Digitální obraz je reprezentován jako pole čísel, kde každé číslo je jeden pixel. Pokud se jedná o barevný obraz, pole je trojrozměrné (výška, šířka, barevné kanály), v případě černobílého je použito pole dvourozměrné.

Pojem zpracování obrazu odkazuje na manipulaci, automatické zpracování, analýzu a interpretaci obrazových dat s využitím jak pokročilých, tak i primitivních algoritmů. V základu se jedná o manipulaci s polem čísel, kdy například pomocí aritmetických operací můžeme v části pole změnit číselnou hodnotu pixelů a tím v obrazu vykreslit čtverec okolo detekovaného objektu. S rostoucí výpočetní kapacitou počítačů nalézá využití ve stále více odvětvích, například v medicíně, robotice, sociálních sítích, nebo automobilovém průmyslu.[1]

Průběh zpracování obrazu je rozdělen do několika základních kroků. Posloupnost těchto kroků není pevně daná, záleží na potřebách dané aplikace. Příkladem může být jednoduchá aplikace afinních transformací pro změnu velikosti, nebo rotace obrázku. V takovém případě bude potřeba minimum kroků pro vykonání daných operací. Při pokročilém zpracování obrazu jsou často nejdříve implementovány kroky, které obraz zpracují do požadovaného tvaru a formátu, přičemž dále následují kroky pro získání obrazových příznaků a jejich klasifikaci. [1] [2] [3]

1.2 Základní kroky zpracování obrazu

Níže jsou popsány základní kroky zpracování obrazu využívané u získávání příznaků a následné detekci objektů.

1.2.1 Získání obrazu

Prvním krokem je získání samotného obrazu. Podle druhu aplikace se také liší zdroj, ze kterého obraz získáváme. Při zpracovávání obrazu v reálném čase se většinou jedná o snímání pomocí kamery, ze které se data ukládají do datové struktury umožňující manipulaci s obrazem, například ndArray z knihovny NumPy.

1.2.2 Předzpracování

Předzpracování slouží především pro uvedení obrazu do potřebného tvaru a formátu pro následující kroky. Postup předzpracování se liší dle typu aplikace, ale téměř pokaždé je nutné

provést úpravy a vylepšení, aby se zabránilo zkreslení dat způsobené nepříznivými podmínkami při pořízení daných snímků. Pro odstranění těchto nedostatků se využívá mnoho metod, mezi které patří například odstranění šumu, převod barevného obrazu do stupňů šedi, nebo zaostření obrazu.[2]

1.2.3 Segmentace

Segmentace je klíčový krok v při zpracování obrazu, který slouží k jeho rozdělení do více sekcí, které odpovídají objektům, nebo oblastem reálného světa. Tyto sekce jsou tvořeny pixely s podobnými hodnotami, které tvoří viditelně jednotnou oblast.

Nejjednodušší metoda segmentace obrazu je se nazývá binární prahování. Zpravidla se využívá pro oddělení pozadí a popředí. Funguje na základě toho, že různé objekty v obraze budou mít různou hodnotu jasu. Při použití binární prahové funkce se proto nastaví práh, který určuje, zda se hodnota pixelu změní na minimální (0 - černá barva), anebo na maximální (255 - bílá barva) hodnotu.

$$g_{(x,y)} = \begin{cases} 255 & if\ f_{(x,y)} > T, \\ 0 & if\ f_{(x,y)} \leq T \end{cases} \quad (1)$$

Pokud jsou výsledkem segmentace oblasti odpovídajícím objektům v původním obraze, jedná se o tzv. kompletní segmentaci. Pokud oblasti neodpovídají přesně objektům, tak tuto segmentaci nazýváme částečnou. [1][2]

1.2.4 Popis a klasifikace objektů

Finální krok u většiny algoritmů je právě popis a klasifikace objektů ze získaných dat. V závislosti na použité metodě se také liší způsob klasifikace. Tradiční algoritmy, které nevyužívají hluboké učení a neuronové sítě často slouží pouze pro získání příznaků daného obrazu a samotnou klasifikaci má již na starosti jiný algoritmus. Metody jako Viola-Jones a HOG často využívají metodu podpůrných vektorů (SVM), která funguje na principu lineárního klasifikátoru.

1.3 Moderní metody pro zpracování obrazu

1.3.1 Tesla Vision

V dnešních moderních automobilech můžeme nalézt čím dál modernější technologie, které mají za cíl usnadnit řízení. Dříve se jednalo především o tempomat, který udržuje stálou rychlost vozidla, ale dnes jsou již automobily po svém obvodu často vybaveny kamerami a senzory na principu sonarů, které snímají okolí a usnadňují tak parkování, předcházejí srážkám, nebo udržují vozidlo ve stejném pruhu. Pro tyto úkony se využívá standartních metod počítačového vidění, pro detekci pruhů na silnici a výpočtu polohy vozidla mezi nimi.

Některé společnosti (například Waymo, Tesla nebo Comma AI) však usilují o to, aby automobily byly schopny kompletně autonomního řízení za využití mnoha senzorů, jako jsou kamery, radary a lidary. Ty mají za úkol shromažďovat data o svém okolí, mimo jiné detailní informace o okolní dopravě, chodcích, překážkách na vozovce atd. Tyto data jsou následně zpracována palubním počítačem, který vyhodnotí situaci a učiní potřebné akce.

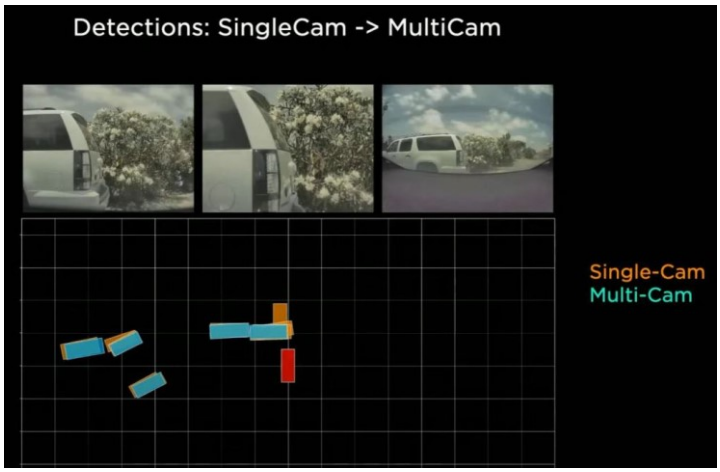
Lídrem v této oblasti je právě společnost Tesla, která vyvíjí technologii Autopilot. Tato technologie prošla mnoha iteracemi, kdy ještě donedávna byla pro sběr dat využívána kombinace kamer a radarů. Dnes již vozidla obsahují pouze soustavu 8 kamer po obvodu vozidla, které zajišťují 360° snímání okolí. Získaná data jsou zpracována palubním počítačem speciálně navrženým pro tyto účely, a právě technologií Tesla Vision.

Silnice, na kterých vozidla jezdí a jejich okolí byly navrženy pro člověka, respektive pro lidské oči a organickou neurální síť – mozek proto je Tesla přesvědčena, že správnou cestou pro umožnění plně autonomního řízení je získávání informací o okolí vozidla pouze pomocí kamer a následné zpracování dat pomocí pokročilých algoritmů a hlubokých neuronových sítí. Inovace technologie Tesla Vision tedy spočívá v tom, že pro schopnost autonomního řízení vozidla nejsou využívány radary a lidary, které mají nevýhodu v tom, že jsou relativně velké a spotřebují mnoho energie. Místo jsou využívány pouze kamery, z jejichž získaných dat je vytvořen takzvaný vektorový prostor, který obsahuje informace o prostoru okolo vozidla.

1.3.1.1 Princip

U předešlých iterací technologie Autopilot, byl každý snímek z každé kamery zpracováván zvlášť. Tento přístup vyžadoval velký výpočetní výkon a nebyl dostatečně robustní, protože zde chyběl kontext uceleného obrazu ze všech kamer.

Nyní jsou však snímky z jednotlivých kamer spojeny do tzv. vektorového prostoru, což je 4D interpretace okolí vozidla (v prostoru a čase). Data z tohoto vektorového prostoru jsou tak využita pro detekci objektů, silničního značení, nebo například predikci dalších událostí. Právě díky návaznosti dat v čase a prostoru jsou tyto predikce velmi přesné a konzistentní. [4]



Obrázek 1. Porovnání predikcí vozidla [5]

Další inovací této technologie je metoda automatického značení objektů a struktur. Označené objekty ve snímcích videa jsou využity pro trénování neuronové sítě a tím zvýšení pokrytí různých méně častých scénářů, se kterými by si vozidlo nemuselo poradit. Značení bylo tradičně prováděno ručně na jednotlivých snímcích videa. Tento způsob je časově velmi náročný a neefektivní. Proto byla implementována metoda, kdy získaná videa z flotily vozidel určená pro trénování jsou předána hluboké neuronové síti, která pro každý bod (x, y) predikuje hodnotu z. Vnikají tak body v 3D prostoru, které jsou segmentovány dle tříd (povrch vozovky, obrubník, horizontální značení atd.) Tímto způsobem jsou také segmentovány statické objekty v okolí, například zdi.[5]

Obrázek 2. Získaný dataset pro trénování [5]

Vznikají tak obsáhlé datasety pro trénování hlubokých neuronových sítí, které díky možnosti získání specifických záznamů z flotily mohou být zaměřeny na problémové oblasti a situace. [4]

1.3.2 Deep Learning Super Sampling (DLSS)

DLSS je technologie vyvíjená společností NVIDIA pro zvýšení výkonu v počítačových hrách za využití real-time upscalingu, tedy převádění obrazu do vyššího rozlišení. Pro tuto činnost se využívají Tensor jádra na grafických kartách řady RTX. Z Tohoto důvodu je DLSS dostupná pouze na grafických kartách řady RTX společnosti NVIDIA.[6]

Jedná se o druh renderování videa, při kterém jsou jednotlivé snímky renderovány do nižší kvality, než je vyžadováno a ty jsou následně předány neuronové síti, která je převede do vyššího rozlišení. Díky této metodě je možno v počítačových hrách dosáhnout vyššího výkonu, avšak za cenu horší kvality obrazu než při nativním rozlišení.[7]

První verze této technologie není generalizovaná, a proto je nutné neuronovou síť natrénovat pro každou počítačovou hru zvlášť. Vstupními daty pro trénování jsou tisíce dvojic snímků ze hry, kdy na jeden snímek je vždy aplikován 64x supersampling anti-aliasing pro zvýšení kvality a rozlišení. Při trénování jsou tyto dvojice porovnávány a výsledkem je schopnost neuronové sítě upscalovat snímky do kvality, která se přibližuje 64x supersamplingovanému obrazu bez artefaktů, které se vyskytují při využití jiných metod. [8]

Obrázek 3. Ukázka výrazného zvýšení FPS a kvality při využití DLSS [7]

Kvůli výše uvedeným nedostatkům byla vydána druhá verze (DLSS 2.0), která je již generalizovaná tzn. není potřeba trénovat neuronovou síť pro každou hru zvlášť. Je zde využita také vylepšená neuronová síť využívající tzv. Temporal Feedback. Tato metoda využívá pohybové vektory pro zvýšení ostrosti obrazu a více konzistentních výsledků mezi jednotlivými snímky. Při trénování se opět porovnávají dvojice obrázků s vysokým a nízkým rozlišením, zde se však využívá snímků vyrenderovaných do 16K rozlišení místo 64x supersamplingovaného snímku. [7]

Obrázek 4. DLSS 2.0 [7]

2 METODY DETEKCE OBJEKTŮ

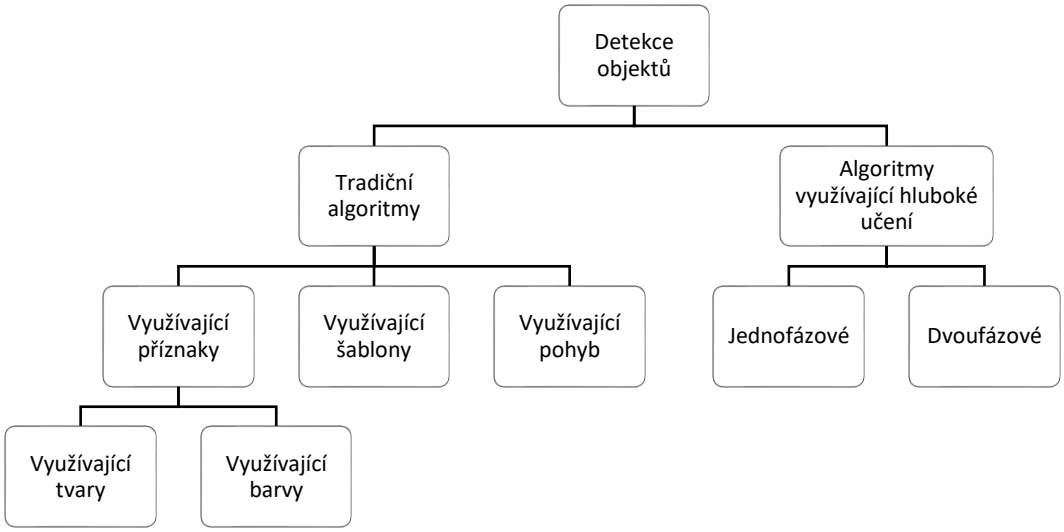
Detekce objektů je metoda zpracování obrazu, při které se provádí analýza snímku za účelem nalezení jeho částí, které odpovídají předmětům, nebo oblastem reálného světa. Tyto objekty jsou následně klasifikovány do předem definovaných tříd. Výsledkem detekce objektů jsou tedy pozice a třídy detekovaných objektů v daném snímku. Tato data se často vizualizují pomocí ohraničujících rámečků a názvem příslušné třídy.

Miniaturizace a šíření výkonných počítačů, dostupnost levných a kvalitních kamer, a především potřeba pro automatický sběr a analýzu dat vede k velkému zájmu o detekci a klasifikaci objektů. V těchto aplikacích je klíčová analýza videa a detekce pohybujících se objektů. Právě pro tyto účely se používají algoritmy porovnávající sekvenci snímků (např. algoritmus optického toku). S analýzou objektů ve videu také souvisí jejich sledování, které může zabraňovat získávání chybných dat při počítání objektů. Této skutečnosti je využito v praktické části práce, kde je implementován algoritmus Deep SORT pro sledování již detekovaných vozidel, a tedy zabránění jejich opětovnému započítání. [9]

Metod pro detekci objektů je velké množství a jejich použití závisí na požadované přesnosti a typu aplikace. Mezi nejjednodušší způsoby patří prahování, nebo přiřazování šablon. Dnes se nejvíce uplatňují algoritmy využívající hluboké učení a neuronové sítě, a to především v oblasti autonomních vozidel, medicíně, nebo prostorové orientaci. [10]

2.1 Rozdělení algoritmů

Metody detekce objektů se dají rozdělit do dvou hlavních kategorií. První kategorie obsahuje tradiční techniky, které nevyužívají hlubokého učení neuronových sítí. Tyto algoritmy se dále dělí dle způsobu detekce objektů (využívající příznaky, vzory, nebo pohyb) viz. Obrázek 5. Patří zde například algoritmus Viola-Jones, nebo HOG. Do druhé kategorie spadají algoritmy čistě využívající hluboké učení a neuronové sítě. Tyto algoritmy jsou relativně nové a v dnešní době také nejpoužívanější, především díky jejich rychlosti a přesnosti. Dále se dělí na jednofázové a dvoufázové.



Obrázek 5. Rozdělení druhů algoritmů pro detekci objektů

2.2 Tradiční algoritmy

Algoritmy v této kategorii slouží k získání příznakových dat z obrazu, které se dále předají klasifikačnímu algoritmu, který tato získaná data vyhodnotí a přidělí jednu z předem definovaných tříd. Příkladem algoritmu pro klasifikaci může být algoritmus SVM neboli metoda podpůrných vektorů.

Obecně fungují na principu hledání vlastností obrazu, které odpovídají hledaným objektům. Může se jednat o skupiny hran, bodů, nebo pixelů. Mezi zástupce této kategorie patří například algoritmus HOG, nebo Viola-Jones detektor využívající Haar charakteristiky.[9][10]

Níže jsou uvedeny příklady nejpoužívanějších metod z této kategorie.

2.2.1 Viola-Jones detektor

Motivací pro vytvoření Viola-Jones detektoru bylo vyřešení problému detekce obličeje. Jedná se o první algoritmus pro tuto problematiku pracující v reálném čase. Této rychlosti je dosaženo mimo jiné díky využití tzv. Haarových příznaků. Detektor je možné využít pro detekci různých objektů, je však potřeba předem vytvořit Haarovy kaskády pro příslušnou třídu objektů. Mezi největší výhody patří již zmíněné zpracovávání v reálném čase a vysoká přesnost detekcí.

Algoritmus se skládá ze čtyř hlavních kroků:

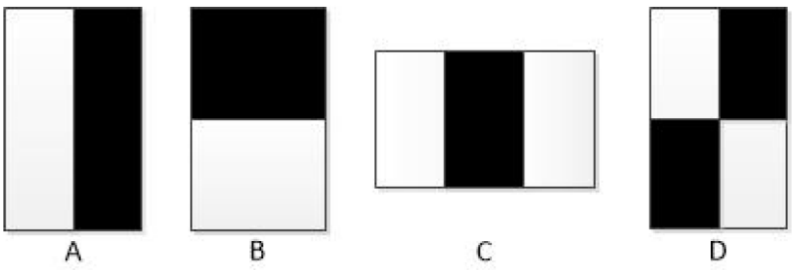
1. Získání Haarových příznaků
2. Vytvoření integrálního obrazu

3. Trénování s využitím AdaBoost algoritmu
4. Kaskádové klasifikátory

2.2.1.1 Získání Haarových příznaků

Viola-Jones detektor využívá několik druhů Haarových příznaků. Jedná se o příznaky se dvěma, třemi, nebo čtyřmi obdélníky. Hodnota těchto obdélníků odpovídá rozdílu v jasu světlých a tmavých oblastí daného příznaku.

Hodnota příznaku se dvěma obdélníky se vypočítá jako rozdíl mezi sumou dvou obdélníkových oblastí o stejné velikosti. Hodnota příznaku se třemi obdélníky se získá ze sumy dvou vnějších obdélníků odečtené od sumy hodnot v obdélníku prostředním. Pro získání hodnoty čtvrtého příznaku je nutné od sebe odečíst sumy párů diagonálních obdélníků. [11]



Obrázek 6. Základní druhy Haarových příznaků [3]

Pro natrénování klasifikátoru je potřeba velké množství pozitivních (s obličeji) a negativních (bez obličejů) snímků. Z těchto snímků se dále získají příznaky pomocí základních Haarových příznaků zobrazených na obrázku výše. Hodnota těchto příznaků se získá odečtením sumy pixelů pod bílým obdélníkem od sumy pixelů pod černým obdélníkem.

2.2.1.2 Vytvoření integrálního obrazu

Integrální obraz je způsob reprezentace obrazu, kdy hodnota každého bodu *x, y* představuje sumu všech předchozích bodů v integrálním obraze směrem nahoru a doleva:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x'y') \qquad (2)$$

kde $ii(x,y)$ je hodnota integrálního obrazu a $i(x,y)$ hodnota původního obrazu.

Pokud v integrálním obrazu označíme jakoukoliv oblast obdélníkem, můžeme jednoduše zjistit sumu bodů oblasti sečtením jeho hodnot v rozích. Této skutečnosti se využívá při získávání příznaků obrazu, kdy je potřeba odečítat sumy dvou, nebo více obdélníků od sebe.

Použití integrálního obrazu tedy umožňuje vysokou rychlost a tím i zpracování dat v reálném čase.

2.2.1.3 Trénování s využitím AdaBoost algoritmu

Kvůli velkému počtu možných příznaků v každém snímku a zrychlení procesu vyhodnocování se využívá algoritmus AdaBoost, který má za úkol vybrat nejvhodnější příznaky pro trénování a trénovat kaskádové klasifikátory, které tyto příznaky využívají.

Algoritmus AdaBoost [11]:

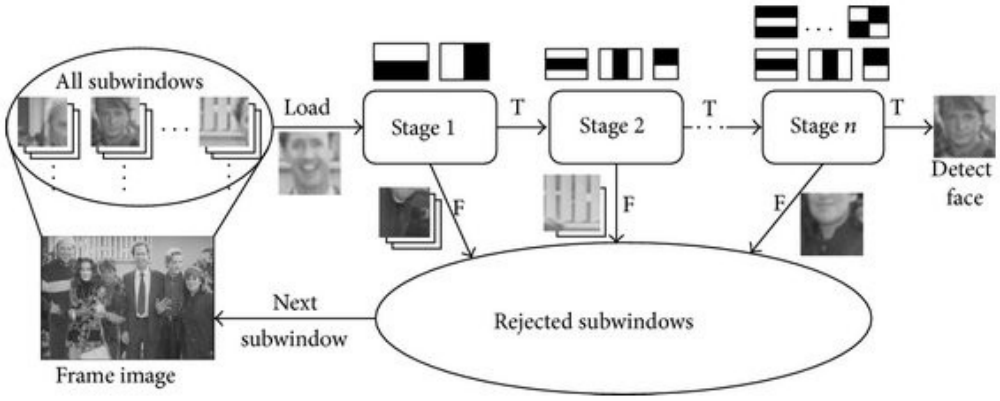
1. Snímky pro trénování $(x_1, y_1) \dots (x_n, y_n)$ jsou označeny $y_i = 0$ negativní vzorky a $y_i = 1$ vzorky pozitivní.
2. Inicializují se váhy s hodnotami $w_{1,i} = \frac{1}{2m}$ pro $y_i = 0$ a $w_{1,i} = \frac{1}{2l}$ pro $y_i = 1$, kde hodnoty *m* a *l* označují počet negativních a pozitivních vzorků.
3. Pro $t = 1, \dots, T$:
   a. Normalizace vah $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
   b. Pro každý příznak *j* se trénuje klasifikátor h_j, který může využívat pouze jeden příznak. Chyby jsou vyhodnoceny s ohledem na
   $$w_t, \epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right| \qquad (3)$$
   c. Vybrání klasifikátoru h_t s nejmenší chybou.
   d. Aktualizace hodnoty vah $w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$, kde $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ a $e_i = 0$ pokud vzorek x je správně klasifikován, jinak $e_i = 1$.
4. Konečný silný klasifikátor je definován jako:
   $$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & jinak \end{cases} \qquad (4)$$

   kde $\alpha_t = \log \frac{1}{\beta_t}$

2.2.1.4 Kaskádové klasifikátory

Obličej, který chceme detekovat často tvoří pouze malou část obrazu. Z toho to důvodu se využívají kaskádové klasifikátory, které mají za úkol zredukovat počet vzorků. Skládají se z více kroků obsahujících silný klasifikátor z algoritmu AdaBoost, které vyhodnocují části obrazu.

Kontrolují, zda daná část obrazu obsahuje náznak části obličeje, nebo ho neobsahuje vůbec. Pokud výběr obrazu zcela jistě část obličeje neobsahuje, je tento vzorek vyřazen. V opačném případě je vzorek předán dalšímu kroku pro zpracování. Vzorek obsahující oblast obličeje je takový, který úspěšně projde všemi kroky kaskádového klasifikátoru. [13]



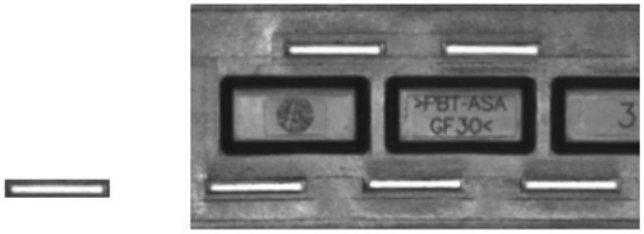
Obrázek 7. Proces funkce kaskádového klasifikátoru využívající Haarovy příznaky [12]

2.2.2 Porovnávání šablon

Detekci objektů založenou na porovnávání šablon je možno použít, pokud je k dispozici šablona (obraz) hledaného objektu. Tato šablona je využívána pro porovnání s aktuálně vybranou oblastí vstupního obrazu s cílem nalezení shody. [10]

2.2.2.1 Naivní porovnávání šablon

Jedná se o nejjednodušší verzi tohoto algoritmu. Spočívá v porovnání šablony, $t(x, y)$, kde (x, y) představují souřadnice pixelů šablony se všemi částmi vstupního obrazu $s(x, y)$. Při každém porovnání se vypočítá hodnota podobnosti mezi šablonou a aktuálně vybranou částí vstupního obrazu. Pixely s nejvyšší hodnotou podobnosti ve výsledném poli odpovídají části hledané šablony.

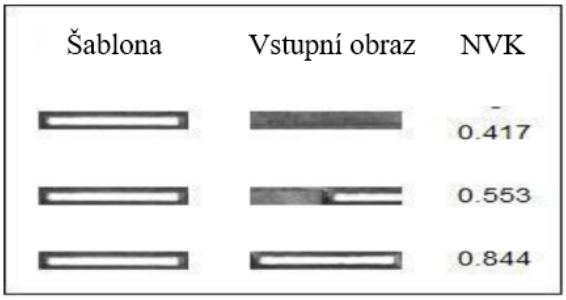


Obrázek 8. Příklad šablony (A) a vstupního obrazu (B) [14]

Tato metoda je vhodná pouze tehdy, pokud šablona přesně odpovídá hledaným objektům, tzn. objekty ve vstupním obraze mají stejnou velikost a úhel vůči kameře jako šablona. [15]

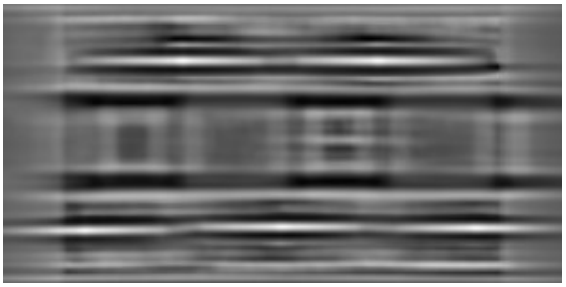
2.2.2.2 Porovnávání šablon s využitím normované vzájemné korelace

Normovaná vzájemná korelace (NVK) je v současnosti jednou z nejpoužívanějších metod pro získávání míry podobnosti při porovnávání šablon, a to především pro nezávislost výsledků na lineárních změnách jasu vstupního obrazu. Jedná se o sumu párových multiplikací odpovídajících pixelů šablony a obrazu. Výsledkem této metody je míra podobnosti v rozmezí od 0 do 1. To znamená, že hodnota dvou shodných obrazů bude 1. Pro získání přesnějších výsledků se často pracuje s obrazy, na kterých byl aplikován detektor hran, nebo byly převedeny do stupňů šedi.



Obrázek 9. Hodnoty NVK v závislosti na podobnosti [14]

Díky těmto vlastnostem má metoda mnohem širší využití a netrpí tolika nedostatky jako předchozí varianta. Stále však přetrvávají problémy s náchylností na změnu měřítka a rotaci, to znamená, že pokud bude hledaný objekt v obrazu pootočen, nebo mít odlišnou velikost, nemusí být detekován. [15]



Obrázek 10. Vizualizace hodnot NVK
– světlé pixely značí vyšší podobnost,
tmavé nižší [14]

2.2.3 Histogram orientovaných gradientů

HOG je deskriptor příznaků původně vytvořený pro detekci postav. Princip této techniky spočívá v tom, že vzhled a tvar objektu v rámci obrazu lze popsat směry hran, nebo pomocí rozložení gradientů intenzity pixelů. Velikost gradientů je největší právě v okolí hran a rohů, a to v důsledku náhlé změny intenzity pixelů. Histogramy směrů gradientů se proto používají jako příznaky deskriptoru. Získané deskriptory dále slouží k samotné detekci objektů za využití některého z algoritmů strojového učení. Společně s HOG se často pro detekci objektů využívá metoda podpůrných vektorů (SVM). [16]

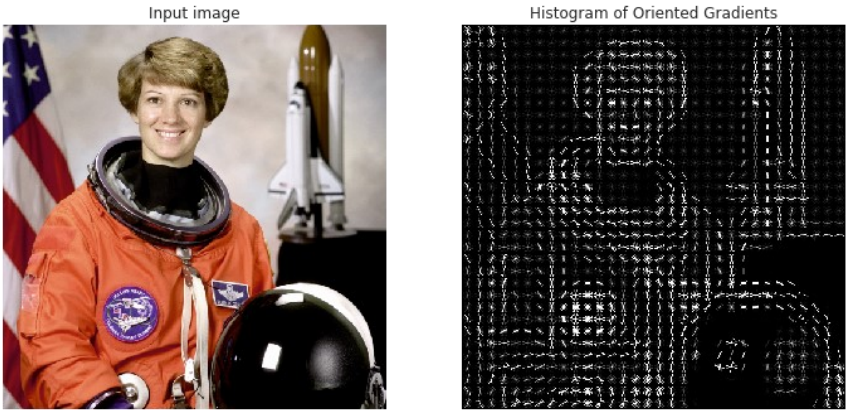
2.2.3.1 Výpočet deskriptoru [16] [17]

1. Normalizace snímku
2. Snímek se rozdělí do buněk (například 8x8) a pro každou se vypočítá směr a velikost gradientu.
3. Gradienty jednotlivých buněk se rozdělí a sečtou dle úhlů do 9 skupin v rozmezí 0–180°. Tím dojde ke zredukování hodnot z 64 na 9.
4. Histogram je normalizován pomocí gradientů buněk ve stejném bloku (skupina sousedních buněk). Pro normalizaci se využívají různé normy, například norma L_1:

$$v \leftarrow \frac{v}{(\|v\|_1 + e)} \qquad (5)$$

   kde e je kladná konstanta zabraňující dělení nulou u bloků neobsahující gradienty.

5. Výsledný deskriptor je získán spojením všech normalizovaných histogramů do jednoho.



Obrázek 11. Vizualizace získaného deskriptoru [18]

2.2.4 Optický tok

Metoda optického toku detekuje pohybující se objekty pomocí zdánlivého pohybu pixelů v sérii snímků videa, za předpokladu, že sousedící pixely mají podobný pohyb a jejich intenzita mezi snímky je neměnná. Algoritmus vytvoří pole vektorů, kde každý vektor představuje pixel obrazu. Tyto vektory znázorňují rychlost a směr pohybu pixelů mezi jednotlivými snímky. [19]

Obrázek 12. Vizualizace vektorů optického toku a následná segmentace objektů [20]

2.2.4.1 Výpočet vektoru optického toku

Pixel $I(x, y, t)$ značí intenzitu pixelu na souřadnicích (x, y) v čase t. Pohyb tohoto pixelu mezi jednotlivými snímky je vyjádřen jako (dx, dy). Za předpokladu, že intenzita pixelů mezi snímky je konstantní, platí následující rovnice:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \quad (6)$$

Poté se pomocí aproximace pravé strany s využitím Taylorovy řady získá samotná rovnice optického toku:

$$f_x u + f_y v + f_t = 0 \qquad (7)$$

kde $u = \frac{dx}{dt}$; $v = \frac{dy}{dt}$. $f_x = \frac{\partial f}{\partial x}$ a $f_y = \frac{\partial f}{\partial y}$ jsou gradienty snímku a f_t je gradientem času. [19][21]

Výsledkem je rovnice (u, v), která kvůli dvěma neznámým veličinám nemá řešení. Pro získání vektoru toku (u, v) se využívají různé metody, např.:

- Lucas-Kanade
- Horn-Schunck

Jedna z nejpoužívanějších metod, Lucas-Kanade předpokládá, že pohyb pixelu mezi snímky je relativně malý a konstantní s jeho okolím a aplikuje rovnice optického toku pro všechny pixely v této oblasti. [21]

Po získání vektorů optického toku nastává tzv. post-processing, při kterém je z výsledného binárního obrazu odstraněn šum a aplikována segmentace.

Nevýhoda této metody je nutnost výpočtu vektorů v každém snímku videa a tím pádem velká výpočetní náročnost, na druhou stranu je ale relativně přesná a přináší informace o pohybu objektu a možnost jeho sledování. [9] [22]

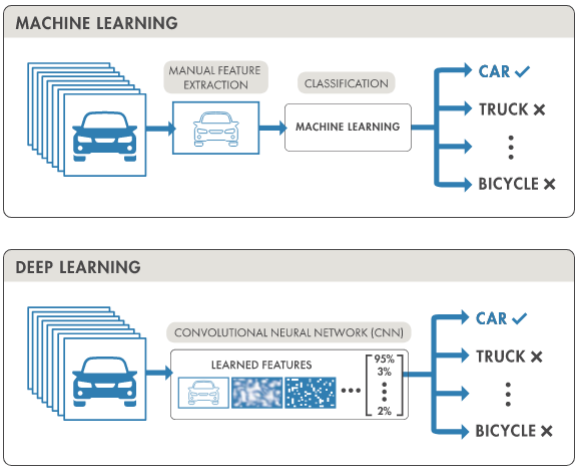
Obrázek 13. Segmentovaný binární obraz (a) Detekovaný objekt v původním snímku (b) [21]

2.3 Algoritmy využívající hluboké učení

Na rozdíl od předešlé kategorie jsou algoritmy využívající hluboké učení a neuronové sítě tzv. End-to-End. Jinými slovy zastávají funkci extrakce příznaků z obrazu a zároveň provádějí klasifikaci. Díky tomu jsou tyto metody dnes velmi populární a zástupce této kategorie je také použit v praktické části práce (YOLO algoritmus).

Algoritmy předávají vstupní obraz konvoluční neuronové síti, která na základě předtrénovaného modelu analyzuje příznaky a klasifikuje detekované objekty. Při použití těchto metod jsou dvě možnosti postupu. [23]

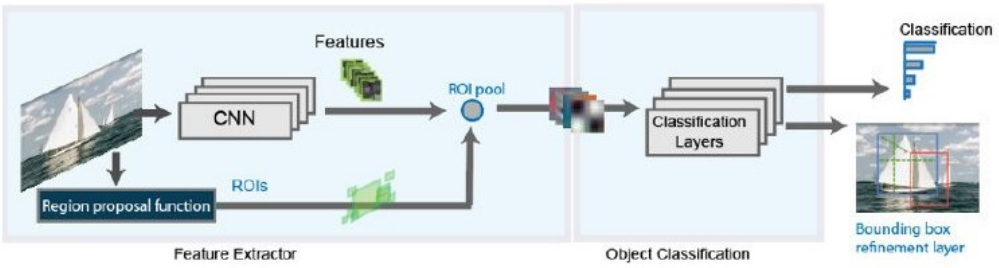
- První možností je využití modelu, který byl předtrénován na jednom z dostupných datasetů. Tento přístup výrazně urychluje vývoj, nelze ale použít ve všech případech. Datasety, na kterých jsou modely trénovány nemusí obsahovat třídy objektů, které jsou vyžadovány, a proto je nutné využít druhou možnost, a to natrénování vlastního modelu.
- Vytvoření vlastního modelu často také vyžaduje vytvoření vlastního datasetu vstupních obrazů, které jsou použity pro trénování. Tento proces může být zdlouhavý, pokud chceme dosáhnout dostatečně přesného a robustního modelu, který si umí poradit se zhoršenými podmínkami (například detekce dopravního značení při špatných světelných podmínkách).



Obrázek 14. Rozdíl mezi tradičními algoritmy a algoritmy využívající hluboké učení [23]

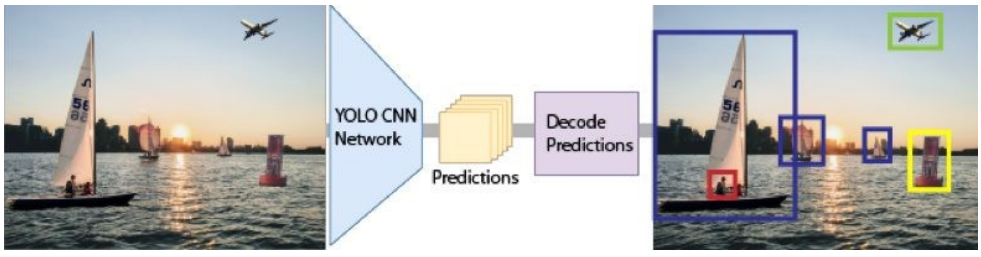
2.3.1 Dvoufázové a jednofázové neuronové sítě

Jak už název napovídá, dvoufázové neuronové sítě se skládají ze dvou hlavních kroků. V první fázi proběhne identifikace oblastí obrazu, ve kterých se může nacházet objekt a ve druhé fázi jsou objekty v navrhovaných oblastech klasifikovány. Příkladem takové neuronové sítě může být například R-CNN. Výhoda dvoufázového přístupu spočívá ve velmi přesných výsledcích, ale na druhou stranu jsou tyto sítě pomalejší a výpočetně náročnější než jednofázové.



Obrázek 15. Architektura dvoufázové sítě Fast R-CNN [24]

Zatímco dvoufázové sítě nejprve identifikují kandidáty a následně klasifikují objekty, v jednofázových sítích konvoluční neuronová sít vytváří předpovědi objektů pro oblasti napříč celým obrazem. Z těchto předpovědí jsou poté získány pozice a třídy detekovaných objektů. Zástupcem této kategorie je algoritmus YOLO. Hlavní výhoda jednofázových sítí je v rychlejším zpracování, a tedy menších nárocích na výpočetní výkon, může to být ale na úkor přesnosti detekce objektů. [24]

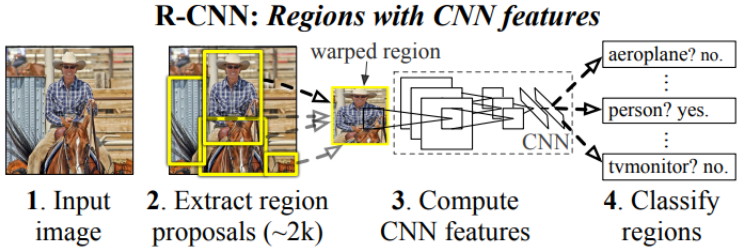


Obrázek 16. Architektura jednofázové sítě YOLO [24]

2.3.2 R-CNN

Region-based Convolutional Neural Networks byl poprvé představen v roce 2014 jako dvoufázový detektor. Jedná se o jednu z prvních úspěšných metod využívající hluboké učení a neuronové sítě a také první algoritmus z rodiny R-CNN.

V první fázi se pomocí algoritmu selektivního vyhledávání získá ze vstupního snímku množina až 2000 zájmových oblastí, u kterých je šance, že obsahují objekt. Ve druhé fázi jsou jednotlivé zájmové oblasti transformovány do definovaného tvaru a předány do konvoluční neuronové sítě, která vygeneruje příznakový vektor pro následnou klasifikaci pomocí SVM.



Obrázek 17. Workflow dvoufázové sítě R-CNN [26]

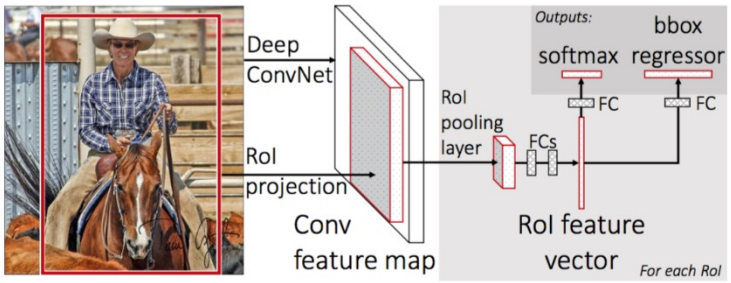
Pro zvýšení výkonu lokalizace se využívá regrese ohraničení objektu, kdy po zpracování vektoru pomocí SVM je vygenerováno nové ohraničení specifickým SVM pro předpovězenou třídu. Protože model často vytváří více ohraničení pro jeden objekt, využívá se metoda Non-max suppression pro nalezení lokálního maxima, čímž se vyfiltrují duplicitní ohraničení na základě důvěry (confidence).

Algoritmus R-CNN přinesl výrazné zvýšení přesnosti detekce objektů, kdy na datasetu PASCAL VOC 2010 dosáhl 53,7 % mAP. Pro představu, algoritmus využívající metodu bag-of-visual-words (na principu bag-of-words – jedná se o reprezentaci obrazu skupinou příznaků), který byl testován na stejném datasetu dosahoval 35,1 % mAP. Tento model má i své nedostatky, a to především velkou výpočetní náročnost z důvodu nutnosti extrakce příznaků v každé ze 2000 zájmových oblastí. Z tohoto důvodu tedy nelze R-CNN použít pro detekci objektů v reálném čase. Dalším problémem je fakt, že metoda se skládá ze tří oddělených fází, které se musí samostatně natrénovat a optimalizovat. [25][26]

2.3.2.1 Fast R-CNN

Metoda Fast R-CNN, představená v roce 2015 řeší výše uvedené nedostatky R-CNN. Pro usnadnění trénování a optimalizaci algoritmu je zde místo tří oddělených fází pouze jedna fáze. I v této metodě je aplikován algoritmus selektivního vyhledávání pro získání kandidátních oblastí, vektor příznaků již ale není extrahován z každé oblasti zvlášť. Místo toho jsou všechny kandidátní oblasti agregovány a předány konvoluční neuronové síti pro dopředný průchod celým snímkem najednou. Výsledkem je soubor příznaků celého snímku (Conv feature map), který je využit v následujícím kroku (RoI pooling layer), kdy jsou příznaky

transformovány do vektorů o potřebném tvaru pro plně propojené vrstvy, které následují. Pro klasifikaci je místo SVM využita Softmax metoda, jejíž výsledek je jedním z výstupů algoritmu. Druhý výstup je z metody regrese ohraničení objektu a jedná se o souřadnice rohů ohraničujícího rámečku.



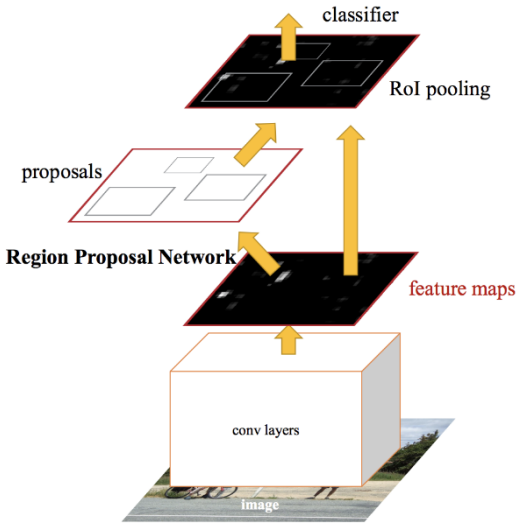
Obrázek 18. Architektura modelu Fast R-CNN [27]

Díky nahrazení tří modelů za jeden a předávání celého snímku konvoluční neuronové síti je tato metoda až 20x rychlejší než její předchůdce, R-CNN (viz. Obrázek 20 na konci kapitoly). Rychlost však stále není dostačující pro využití v aplikacích pracujících v reálném čase. I přes vyřešení problémů s opakujícími se výpočty při získávání příznaků a zefektivnění architektury modelu, je zde stále využívána metoda selektivního vyhledávání, která je časově a výpočetně velmi náročná.[25][27][28]

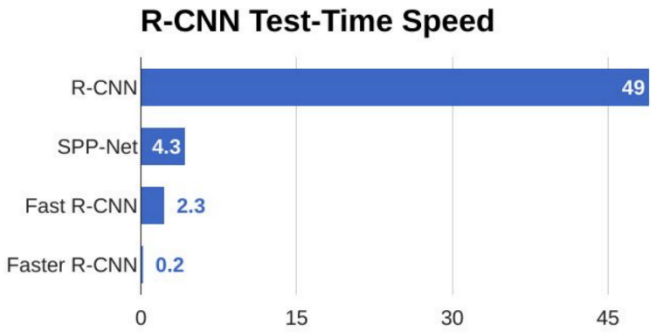
2.3.2.2 Faster R-CNN

Faster R-CNN z roku 2016 řeší poslední velký problém Fast R-CNN, a to sice využívání algoritmu selektivního vyhledávání pro získávání zájmových oblastí. Funkce této metody je nahrazena vrstvou RPN, která je obsažena v neuronové síti. Princip je stejný jako u Fast R-CNN, zde jsou ale zájmové oblasti získány pomocí vrstvy RPN, jejímž vstupem je obraz a výstupem je soubor zájmových oblastí. Tyto oblasti jsou dále předány do vrstvy RoI pooling pro transformaci do požadovaného tvaru a následnou klasifikaci a regresi ohraničení objektu.

Díky integraci získávání zájmových oblastí přímo do neuronové sítě je tento model mnohem rychlejší a výpočetně méně náročný než jeho předchůdci (viz. Obrázek 20.). Proto je také použitelný v aplikacích pracujících v reálném čase. [25][28]



Obrázek 19. Architektura Faster R-CNN [28]



Obrázek 20. Porovnání potřebného času (s) pro detekci objektu u jednotlivých algoritmů z rodiny R-CNN [29]

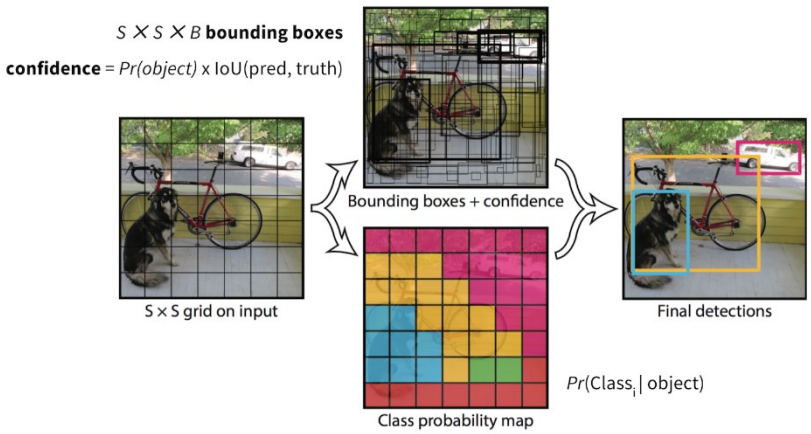
2.3.3 You Only Look Once (YOLO)

Algoritmus YOLO byl vytvořený za účelem detekce objektů v reálném čase a je jedním z prvních, který se řadí do kategorie jednofázových detektorů. Na rozdíl od jiných metod tedy využívá pouze jednu neuronovou síť pro celý proces. Liší se také ve způsobu detekce objektů, protože neuronová síť zpracovává obraz jako celek a získává tak globální kontext, který dále ovlivňuje předpovědi objektů.[30]

V tomto algoritmu je využito metriky IoU, která udává přesnost predikovaného ohraničení objektu oproti reálnému ohraničení. Hodnota IoU je rovna 1, pokud je predikované ohraničení totožné s reálným ohraničením. [31]

Vstupní obraz se nejprve rozdělí do mřížky SxS buněk, s tím, že buňka je odpovědná za případnou detekci, pokud se v ní nachází střed daného objektu. Každá buňka předpovídá B souřadnic ohraničení objektů, skóre důvěry (confidence), které se získá vynásobením pravděpodobnosti správné detekce objektu a hodnoty IoU. Pokud buňka obsahuje objekt, tak se také předpovídá C pravděpodobností tříd, které jsou nezávislé na počtu predikovaných ohraničení.

Jeden obraz tedy může obsahovat celkem $SxSxB$ ohraničení, která se překrývají a je potřeba je vyfiltrovat. Pro tento účel je využita metrika IoU, kdy je pro každý detekovaný objekt ponechána pouze predikce s nejvyšší hodnotou IoU.



Obrázek 21. Princip modelu YOLO [30]

Největší výhoda algoritmu YOLO spočívá v jeho rychlosti a zároveň přesnosti detekce. Na datasetu PASCAL VOC 2007 dosahoval přesnosti 63,4 % mAP při 45 snímcích za sekundu, což je více než dostačující rychlost pro potřeby aplikací pracujících v reálném čase. Pro porovnání, metoda Faster R-CNN ZF má na stejném datasetu podobnou přesnost, 62,1 % mAP, ale rychlost pouze 18 snímků za sekundu. [30][32][33]

2.3.3.1 YOLOv2

YOLOv2 vylepšuje původní implementaci algoritmu YOLO pro zvýšení přesnosti detekce objektů a snížení množství lokalizačních chyb.

Mezi hlavní vylepšení patří zvýšení rozlišení klasifikátoru z 224x224 na 448x448 pixelů, což vedlo ke zvýšení výkonu u vstupních obrazů s vyšším rozlišením. Dále byly plně propojené vrstvy nahrazené tzv. anchor boxy, které slouží pro předpovídání ohraničení objektů, podobně jako v algoritmu Faster R-CNN. Při trénování se nyní využívají snímky o rozlišení

416x416, pro získání lichého počtu buněk a tím i středové buňky. Tato buňka umožňuje rychlejší detekci velkých objektů ve středu obrazu, které se zde často nacházejí.[33][34]

2.3.3.2 YOLOv3

Tento model z roku 2018 aplikuje další vylepšení a poznatky na předchozí verzi algoritmu, YOLOv2.

Jednou z hlavních změn je nahrazení funkce Softmax pro predikci tříd za metodu Binary cross-entropy loss, která umožňuje efektivní využití u datasetů, které obsahují třídy nadřazené jiným třídám (například muž a osoba). Při aplikaci tohoto modelu jsou tak získána detailnější data o objektech, které se ve vstupním obraze nacházejí.

Je zde také aplikována vícestupňová predikce, kdy jsou ohraničení objektů postupně predikována ve třech různých rozlišeních. Díky tomu je k dispozici více kandidátů ohraničení objektu.[33][35]

2.3.3.3 YOLOv4

Jedná se o modifikovanou verzi modelu YOLOv3, která implementuje nové techniky a dále optimalizuje algoritmus pro zvýšení výkonu. Příkladem může být vylepšená augmentace dat při trénování modelu, která zajistí větší robustnost a lepší výsledky při detekci objektů. Tato verze dále umožňuje relativně snadné a rychlé natrénování vlastního modelu na konzumních grafických kartách jako je NVIDIA RTX 2080Ti.

Díky těmto vylepšením dosahuje model až dvakrát vyššího výkonu než algoritmus EfficientDet. Je také ideální pro detekci objektů v reálném čase, díky vysoké snímkové frekvenci, která se pohybuje okolo 65 FPS na grafické kartě NVIDIA Tesla V100. [36]

3 KNIHOVNA OPENCV

OpenCV je knihovna s otevřeným zdrojovým kódem pro počítačové vidění, strojové učení a zpracování obrazu. Byla vytvořena především pro usnadnění implementace metod počítačového vidění v komerčních produktech a také za účelem zpracovávání dat v reálném čase, což je v dnešních systémech velmi důležité. To je umožněno mimo jiné díky podpoře hardwarové akcelerace pomocí technologií IPP pro procesory společnosti Intel a CUDA pro grafické karty společnosti NVIDIA.

Jedná se o velmi rozsáhlou knihovnu, která v současnosti zahrnuje přes 2500 optimalizovaných algoritmů počítačového vidění a strojového učení. Obsahuje jak tradiční algoritmy, které se používají například pro vyhlazování obrazu, či odstraňování šumu, tak i nejmodernější algoritmy využívající neuronové sítě například pro detekci a sledování objektů.

Vývoj knihovny OpenCV započal ve společnosti Intel v roce 1999 za účelem zvýšení výkonu aplikací pracujících s obrazovými daty. Na začátku byly stanoveny cíle, které udávaly směr dalšího vývoje knihovny. Jednalo se především o vytvoření knihovny s otevřeným zdrojovým kódem, která by byla dostatečně robustní a optimalizovaná pro základní techniky počítačového vidění a umožňovala tak efektivní výzkum v této oblasti. Další cíle kladly důraz na usnadnění vytváření čitelného a přenositelného kódu. V roce 2006 byla po více než šesti letech vývoje vydána první oficiální verze 1.0. [37]

Knihovna byla původně vytvořena pro použití v jazyce C++, nyní existují oficiální rozhraní také v jazycích Python, Java a MATLAB. Dále existují knihovny a wrappery, které umožňují použití OpenCV v dalších jazycích, jedná se například o OpenCV.js [38] pro JavaScript, nebo EmguCV [39] pro jazyk C#. Oficiální rozhraní jsou dostupná na většině současných operačních systémů, jako jsou například Windows, Linux, nebo macOS. Knihovna je také využitelná na mobilních zařízeních s operačními systémy Android a iOS.

Díky velkému množství algoritmů a funkcí pokrývajících techniky počítačového vidění a zpracování obrazu, otevřenému zdrojovému kódu a podpoře mnoha programovacích jazyků a platforem je dnes OpenCV nejpoužívanější knihovnou počítačového vidění. Alternativ s podobně rozsáhlým obsahem a rozšířeností není velké množství a každá má své výhody a nevýhody oproti OpenCV, vybrané knihovny jsou představeny v kapitole 4. [37][40]

OpenCV má modulární strukturu a skládá se z 8 hlavních modulů: [40]

- **Jádro (core)** – základní modul, definuje struktury a funkce využívané v dalších modulech.
- **Zpracování obrazu (imgproc)** – modul pro zpracování obrazu, který obsahuje metody pro filtrování a transformaci obrazu, změny barevného spektra.
- **Analýza videa (video)** – obsahuje funkce pro analýzu videa, jako jsou například algoritmy pro sledování objektů, předpovídání pohybu, nebo rozdílu pozadí.
- **Kalibrace kamery a 3D rekonstrukce (calib3d)** – modul se skládá z funkcí pro různé úrovně kalibrace a nastavení kamery, 3D rekonstrukce a dalších.
- **2D příznakový framework (features2d)** – zde se nachází především detektory a deskriptory příznaků, například SIFT, nebo FAST.
- **Detekce objektů (objdetect)** – obsahuje algoritmy pro detekci objektů a QR kódů, patří zde například HOG a kaskádový klasifikátor Viola-Jones.
- **HighGUI (highgui)** – modul pro vytváření jednoduchého uživatelského rozhraní, například při zobrazování videa či obrázku.
- **Video I/O (videoio)** – obsahuje funkce pro práci s videem a různými kodeky.

3.1 Vybrané metody a funkce

Níže jsou představeny vybrané metody a funkce využívané mimo jiné v oblasti detekce objektů v reálném čase.

3.1.1 Základní operace

Mezi základní operace lze zařadit práci se soubory, především načítání a ukládání obrazových dat. Pro tyto účely knihovna obsahuje funkce *imread()* a *imwrite()*. Je nutno podotknout, že OpenCV načítá obrázky v barevném prostoru BGR místo dnes tradičního RGB, proto je nutné provést převod pro správné zobrazení. Načtené obrázky následně lze zobrazit díky modulu HighGUI pomocí *imshow()*. Při zobrazování obrázků je také nutné využít funkce *waitKey()*, díky které zůstane okno s obrázkem otevřené, dokud jej uživatel nezavře.

Po načtení obrazových dat je často nutné provést transformaci do požadovaného tvaru pro další zpracování. K tomu slouží funkce *resize()* se vstupními parametry pro zdrojový obrázek a požadovanou šířku a výšku v pixelech.

Zdrojový kód 1. Základní operace

```
img = cv2.imread('image.png')
img = cv2.resize(img, (150,150))
cv2.imshow('Image', img)
cv2.waitKey()
```

Jak bylo výše uvedeno, pro správné zobrazení načtených obrázků je nutný převod barevného prostoru. Pro tyto účely slouží funkce *cvtColor()* se vstupními parametry pro zdrojový obrázek a druhem převodu. Pokud je potřeba obrázek převést do prostoru v odstínech šedi, který se často využívá právě při detekci příznaků a podobných operacích, lze provést konverzi přímo při načítání obrázku, a to přidáním volitelného parametru *0*. Knihovna ovšem nabízí i další barevné prostory jako jsou například BGR, nebo HSV.

Zdrojový kód 2. Změna barevného prostoru

```
img = cv2.imread('image.png', 0)
img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
cv2.imshow('Image', img)
cv2.waitKey()
```

V praktické části bakalářské práce jsou tyto funkce využity pro načítání uloženého snímku z kamery jako pozadí v prvku uživatelského rozhraní a také pro změnu velikosti snímků živého přenosu.

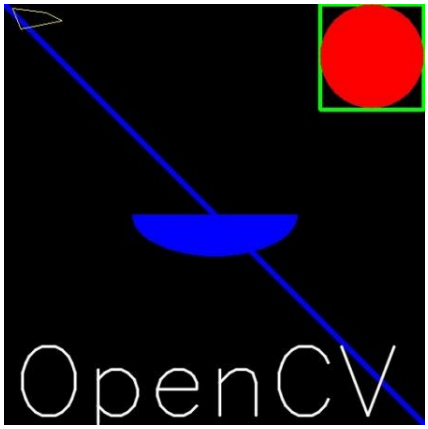
3.1.2 Funkce pro kreslení

Získaná data, například ohraničení objektů je často potřeba vizualizovat přímo ve snímcích. Pro tyto účely je implementováno mnoho funkcí, které umožňují kreslení úseček, polygonů, základních geometrických tvarů a také přidávání textu. Tyto funkce mají zpravidla vstupní parametry pro obraz, souřadnice bodů, tloušťku čáry, barvu a v případě textu také font a jeho velikost. Pokud má parametr pro tloušťku čáry hodnotu -1, bude výsledný útvar vyplněn barvou. U složitějších tvarů, jako je například elipsa, nebo polygon je nutné zadat více parametrů definující tvar výsledného obrazce.

Vzhledem k tomu, že obrázek je vícerozměrné pole hodnot, je také možné editovat přímo toto pole za využití indexace. Tento způsob editace obrazu se hodí například při aplikaci masky na konkrétní část obrazu, nebo oříznutí. [41]

Zdrojový kód 3. Funkce pro kreslení [41]

```
# Vytvoreni prazdneho obrazu
img = np.zeros((512,512,3), np.uint8)
# Diagonalni modra cara s tloustkou 5 px
cv2.line(img,(0,0),(511,511),(255,0,0),5)
# Zeleny ctverec umisteny v levem hornim rohu
cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)
# Vyplneny cerveny kruh s polomerem 63 px
cv2.circle(img,(447,63), 63, (0,0,255), -1)
# Elipsa
cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
# Polygon definovany pomoci pole bodu
pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)
pts = pts.reshape((-1,1,2))
cv2.polylines(img,[pts],True,(0,255,255))
# Text s definovanym fontem a typem cary
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,500), font, 4,(255,255,255),2,cv2.LINE_AA)
```



Obrázek 22. Výsledný obrázek po použití funkcí pro kreslení [41]

V praktické části je pomocí kreslení aplikována maska, která zabraňuje nechtěným detekcím vozidel ve zvolených částech obrazu. Pomocí kreslících funkcí jsou taky v obrazu vizualizovány detekční oblasti a trasy jednotlivých vozidel.

3.1.3 Práce s videem

Pro práci s videem jsou dostupné dvě hlavní funkce, *VideoCapture()* a *VideoWriter()*. První z uvedených funkcí slouží pro otevírání video souborů z uvedené cesty a také umožňuje přístup k webkameře. Protože video je vlastně sekvence jednotlivých snímků, zpracovávání videa je vesměs totožné se zpracováváním obrazu. Práce s videem tedy probíhá uvnitř smyčky, ve které se z načteného video souboru zpracovávají jednotlivé snímky. Z videa je také možné získat informace o počtu snímků za sekundu, nebo výšce a šířce jednotlivých snímků a to pomocí funkce *get()*.

Druhá funkce, *VideoWriter()* slouží pro opětovné vytvoření videa z modifikovaných snímků. Pro tuto funkci je také nutno využít *VideoWriter_fourcc()*, jenž slouží pro specifikaci čtyř znakového kódu určující použitý kodek ve vytvářeném videu. Tento kód může být například *DIVX*, nebo *XVID.* Dalšími parametry funkce *VideoWriter()* je název a koncovka vzniklého souboru, počet snímků za sekundu a rozlišení snímků. Stejně jako u čtení videa, tak i jeho vytváření probíhá ve smyčce. Je tedy možné načítat video z webkamery, provádět na jednotlivých snímcích různé operace a rovnou tyto snímky ukládat do souboru. [42]

Zdrojový kód 4. Zobrazení a ukládání videa z webkamery

```
cap = cv2.VideoCapture(0)
writer = cv2.VideoWriter('video.mp4', cv2.VideoWriter_fourcc(*'DIVX'),
                         30, (1280, 720))
while True:
    ret, frame = cap.read()
    if ret:
        writer.write(frame)
        cv2.imshow('Webcam', frame)
        # Q pro ukonceni
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
cap.release()
writer.release()
cv2.destroyAllWindows()
```

Funkce *VideoWriter()* je na pozadí využívána knihovnou CamGear pro práci s video streamem daného živého přenosu.

3.1.4 Zpracování obrazu

3.1.4.1 Vyhlazování obrazu

Vyhlazování obrazu se často využívá pro snížení šumu, nebo odstranění artefaktů, které jsou způsobeny kamerou. Tuto techniku je také vhodné využít při detekci hran, protože díky odstranění šumu se sníží počet detekovaných rohů a hran, které jsou většinou nedůležité.

Existuje mnoho způsobů vyhlazení obrazu, v knihovně OpenCV je na výběr hned z několika nejpoužívanějších metod, například *blur()*, *GaussianBlur()*, *boxFilter()*, nebo *medianBlur().*Tyto funkce se liší především způsobem výpočtu výsledného pixelu. Zatímco u nejjednodušší metody *blur()* je výsledný pixel průměrem okolních pixelů, u dalších funkcí se využívá složitějších metod výpočtu, příkladem může být *GaussianBlur()*, kde se používá Gaussova funkce. Vstupem funkcí je zpravidla obraz a velikost kernelu, tedy oblasti využívané pro výpočet vyhlazeného pixelu. [43]

3.1.4.2 Morfologické operátory

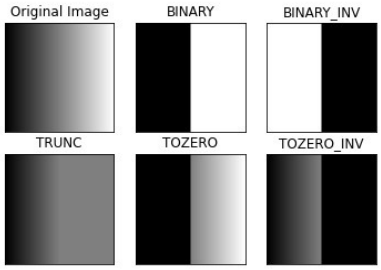
Morfologické operátory pracují s tvarem obrazu a často se používají po aplikaci prahování. Využívají se zde dvě hlavní funkce, *erosion()* a *dilatation()*. První funkce, jak název napovídá napodobuje erozi, která se odehrává v přírodě. Vstupními parametry funkce jsou obrázek a velikost pracovního okna. Obraz, který již prošel prahováním obsahuje pixely s hodnotami 0, nebo 1. Při posouvání pracovního okna se mění hodnota pixelu z 1 na 0, pokud má některý pixel v okně hodnotu 0. Ve výsledku tak jsou všechny pixely s hodnotou 1, které se nachází na hraně objektu změněny na hodnotu 0 a tím dojde k jeho ztenčení. [44]

Druhá funkce, je přesným opakem eroze. Hodnoty pixelů se mění z 0 na 1 pokud se v pracovním okně nachází pixel s hodnotou 1. Tato metoda se většinou používá po erozi, protože ta sice odstraní nežádoucí šum, ale také dojde ke ztenčení objektu v popředí. Proto se využije dilatace, aby se objekt zase zvětšil.

Obrázek 23. Obrázek před a po aplikaci eroze [44]

3.1.4.3 Prahovací funkce

Prahovací funkce slouží pro vyfiltrování pixelů na základě jejich hodnoty. Tímto způsobem lze například oddělit pozadí a popředí obrazu. Základní princip prahování je porovnávání hodnoty pixelu se stanoveným pravidlem. Pokud je hodnota pixelu menší než práh, jeho hodnota je změněna na minimum – 0. V opačném případě je hodnota změněna na maximální hodnotu – 255. Výsledkem tohoto procesu je binární obraz. V knihovně OpenCV je ve funkci *threshold()* implementováno několik základních metod pro prahování, které se liší především pravidlem pro nahrazení pixelu. Nejjednodušší varianta je binární prahování, která změní hodnotu pixelu buď na hodnotu 0 – černou barvu, nebo na hodnotu 255 – bílou barvu. Příkladem jiného způsobu prahování může být metoda *THRESH_TRUNC()*, kdy jsou pixely s hodnotou vyšší než nastavenou změněny na hodnotu prahu. [45]



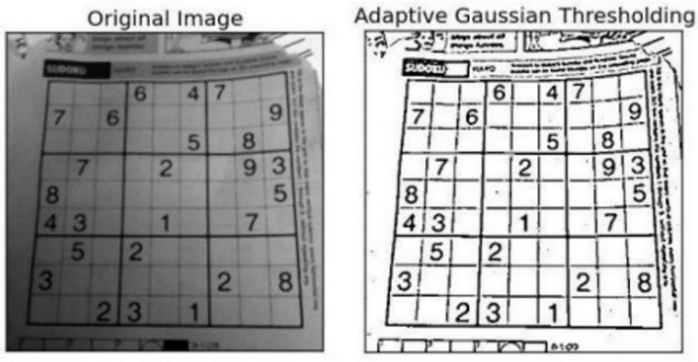
Obrázek 24. Základní funkce pro prahování [45]

Je zde také implementováno adaptivní prahování, které nemá pevně stanovený práh. Místo toho je tato hodnota dynamicky upravována dle aktuálního pixelu a jeho okolí. Práh může být vypočítán dvěma způsoby, buď odečtením konstanty C od průměru hodnot pixelů v okolí, anebo odečtením konstanty C od průměru hodnot vypočítaného s využitím Gaussovy funkce.

Funkce pro adaptivní prahování *adaptiveThreshold()* přijímá jako parametry obrázek, hodnotu prahu, zvolenou metodu adaptivního prahování, prahovací funkci, velikost bloku a nakonec hodnotu konstanty C.

Zdrojový kód 5. Adaptivní prahování s využitím Gaussovy funkce

```
img = cv2.imread('sudoku.png',0)
img = cv2.medianBlur(img,5)

th = cv2.adaptiveThreshold(img,255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                           cv2.THRESH_BINARY,11,2)
```



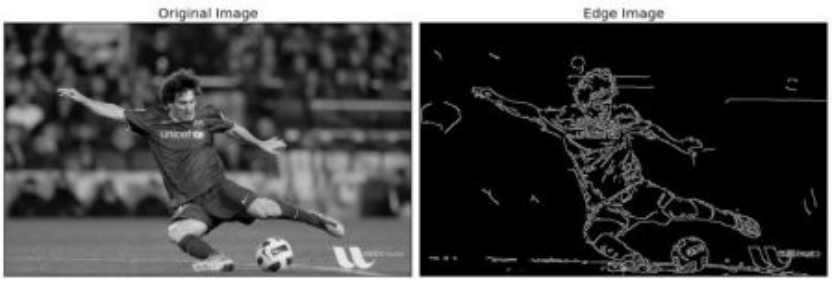
Obrázek 25. Adaptivní prahování s využitím Gaussovy funkce [45]

3.1.4.4 Detekce hran

Pro detekci hran je v knihovně OpenCV k dispozici několik metod. Jedná se například o *canny()*, nebo *sobel()*. První funkce je implementací Cannyho hranového detektoru. Jedná se o více stupňový algoritmus, který nejprve pomocí Gaussovy funkce odstraní z obrazu šum. Jedná se o důležitý krok, který zajistí, že nebudou detekovány falešné hrany. V dalším kroku je pomocí Sobelova operátoru vypočítána velikost a směr hranového gradientu. Následně jsou nalezena lokální maxima hodnot pixelů, jejichž okolí je ve směru gradientu. Tato lokální maxima jsou považována za potencionální hrany, proto jsou pixely, které se v maximech nenachází odstraněny a výsledkem je obraz se zvýrazněnými hranami. Jelikož jsou ve vzniklém obrazu zvýrazněny i ty nejmenší hrany, v posledním kroku je využita prahovací funkce, která má jako vstupní parametry minimální a maximální hodnotu prahu. Všechny hrany, které jsou mimo rozsah těchto hodnot jsou odstraněny a tím je vytvořen finální obraz. [46]

Zdrojový kód 6. Detekce hran pomocí Cannyho detektoru

```
img = cv2.imread('img.jpg', 0)
edges = cv2.Canny(img, 100, 125)
```



Obrázek 26. Detekované hrany v obraze [46]

3.1.5 Detekce příznaků a objektů

Jak bylo popsáno v kapitole 2, způsobů detekce příznaků je velké množství. Z těch nejpoužívanějších zde můžeme nalézt například metodu Shi-Tomasi pro detekci rohů implementovanou ve funkci *goodFeaturesToTrack()*, SIFT, nebo FAST. Pro detekci objektů je zde ve funkci *CascadeClassifier()* implementována metoda Viola-Jones využívající kaskádové klasifikátory a Haarovy příznaky. Knihovna OpenCV také poskytuje nástroje pro sledování detekovaných objektů, a to mimo jiné pomocí *meanShift()*, nebo algoritmu optického toku ve funkci *calcOpticalFlowPyrLK()*.

3.1.6 Strojové učení – DNN modul

Tento volitelný modul umožňuje využití moderních metod pro detekci objektů a zpracování obrazu, které využívají hluboké učení a neuronové sítě. Je podporováno několik frameworků pro strojové učení jejichž předtrénované modely je možno použít. Mimo jiné se jedná o PyTorch, Darknet, TensorFlow a další. Tento modul tedy umožňuje snadné použití předtrénovaných modelů bez potřebných znalostí některého z výše uvedených frameworků. Další výhodou je možnost využití hardwarové akcelerace při výpočtech. Pro akceleraci na grafických kartách je k dispozici technologie CUDA od společnosti NVIDIA a také OpenCL pro procesory a grafické karty od Intelu a AMD. [47]

Pro načtení modelu knihovnou OpenCV je potřeba konfigurační soubor samotného modelu a soubor s předtrénovanými váhami. Tyto dva soubory jsou načteny pomocí funkce odpovídající frameworku, ve kterém byl model natrénován, například pro Darknet: *dnn.readNetFromDarknet()*. Dále může být povolena hardwarová akcelerace a nastaven požadovaný formát desetinného čísla s plovoucí čárkou pro optimalizaci výkonu. Nakonec je model předán funkci pro detekční modely a jsou nastaveny parametry vstupního obrazu.

Zdrojový kód 7. Načtení Darknet modelu a povolení hardwarové akcelerace

```
net = cv2.dnn.readNetFromDarknet('yolov4.cfg', 'yolov4.weights')

# Use GPU if available
if cv2.cuda.getCudaEnabledDeviceCount() > 0:
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA_FP16)

model = cv2.dnn_DetectionModel(net)
model.setInputParams(size=(704, 704), scale=1/255)
```

4 ALTERNATIVY KNIHOVNY OPENCV

V této kapitole jsou představeny nejpoužívanější knihovny a frameworky zaměřené na počítačové vidění, které můžou sloužit jako náhrada za knihovnu OpenCV.

4.1 SimpleCV

SimpleCV je framework počítačového vidění s otevřeným zdrojovým kódem vytvořený pro jazyk Python. Skládá se z různých knihoven a algoritmů, které se v této oblasti využívají (například OpenCV, numpy, scipy). Byl vyvinut za účelem usnadnění práce při vývoji aplikací, které tyto knihovny používají, a to díky funkcím s jednodušší syntaxí.

Tento framework je tedy vhodný především pro začátečníky, vývojáře bez znalostí z oblasti počítačového vidění, nebo pro vytváření rychlých prototypů. K dispozici je také oficiální kniha Practical Computer Vision with SimpleCV [48], která popisuje funkce knihovny na užitečných příkladech.

Vývoj SimpleCV byl již ukončen a poslední verze je z roku 2012. Framework tak sice neobsahuje nejnovější techniky a algoritmy, které se dnes využívají v počítačovém vidění, ale i přesto je vhodný pro širokou škálu aplikací, které pracují s technikami počítačového vidění. [49]

4.1.1 Výhody

- Zjednodušená syntaxe oproti OpenCV
- Oficiální kniha s návody
- Rychlé prototypování

4.1.2 Nevýhody

- Vývoj byl ukončen
- Pouze pro Python 2
- Neobsahuje nejnovější funkce a algoritmy

4.2 MATLAB

MATLAB je programovací jazyk a také vývojové prostředí pro vědeckotechnické výpočty od společnosti MathWorks. Využívá se v mnoha technických oblastech pro výpočty, modelování, analýzu dat, zpracování signálů, či měření. Jak už název napovídá, klíčovou

strukturou jsou zde matice, pomocí kterých probíhá většina výpočtů. Prvky těchto matic mohou být nejen čísla, ale i složitější objekty. Jelikož se jakýkoliv obraz dá reprezentovat jako vícerozměrná matice čísel, tak je MATLAB ideální nástroj pro práci s obrazovými daty. Pro účely počítačového vidění je zde obsažena samostatná knihovna s názvem Computer Vision Toolkit.

Na rozdíl od ostatních knihoven a frameworků počítačového vidění je program MATLAB licencovaný. K dispozici jsou licence standartní, akademické, studentské a pro domácí použití. Cena standartní licence aktuálně začíná na 800 eurech. I díky vysoké ceně je program využívaný především mezi akademiky, nebo inženýry. [50]

4.2.1 Computer Vision Toolkit

Jedná se o jednu rozšiřujících knihoven programu MATLAB zaměřenou na počítačové vidění. Obsahuje funkce a algoritmy pro zpracování obrazu i videa využívané v této oblasti, jako jsou například:

- Detekce objektů
- Sledování objektů
- Segmentace obrazu
- Detekce a klasifikace příznaků
- Kalibrace kamery

K dispozici jsou také nástroje pro trénování vlastních detekčních modelů, které využívají hluboké učení a neuronové sítě, jako například YOLOv2, nebo SSD. Tyto modely je také možné spouštět na grafických procesorech společnosti NVIDIA s využitím technologie CUDA a tím tak výrazně snížit výpočetní čas. [51]

4.2.2 Výhody

- Rychlé prototypování
- Velké množství funkcí a nástrojů pro různé výpočty
- Možnost trénování vlastních detekčních modelů (v OpenCV nelze)
- Integrované nástroje pro vizualizaci dat (OpenCV obsahuje pouze nástroje pro vytváření histogramů)

4.2.3 Nevýhody

- Nutno zakoupit licenci
- Nepřenositelný kód
- Menší výkon v porovnání s OpenCV

4.3 BoofCV

BoofCV je knihovna počítačového vidění s otevřeným zdrojovým kódem vyvíjena za účelem zpracovávání obrazu v reálném čase. První verze byla vydána v roce 2011 a je napsána v programovacím jazyce Java.

Jedná se o poměrně obsáhlou knihovnu zahrnující základní techniky zpracování obrazu, kalibraci kamery, detekci a sledování příznaků a další funkce běžné v oblasti počítačového vidění. BoofCV je dále rozdělena do následujících balíčků:

- **Zpracování obrazu** obsahuje převážně funkce a algoritmy pracující s jednotlivými pixely obrazu. Jedná se o konvoluce, prahovací funkce, morfologické operátory či rozmazání obrazu.
- **Balíček s příznaky** zahrnuje algoritmy pro získání a sledování příznaků a segmentaci. Nachází se například algoritmy optického toku, nebo SIFT a SURF detektory.
- **Geometrické vidění** slouží pro zpracování získaných obrazových příznaků s využitím 2D a 3D geometrie.
- **Kalibrační balíček** se skládá z nástrojů pro zjištění vlastností kamery a její správnou kalibraci.
- **Rozpoznávací balíček** je určen pro klasifikaci a sledování objektů.
- **Vizualizační balíček** obsahuje funkce pro vizualizaci dat.
- **IO Balíček**

I když byla knihovna původně vytvořena pro programovací jazyk Java, její části byly autorem předělány do jazyka C++, Python a nově také Kotlin. Právě díky implementaci v Javě a Kotlinu je možné knihovnu využít také na zařízeních se systémem Android. [52]

4.3.1 Výhody

- Robustní knihovna s velkým množstvím funkcí
- Dostupnost na platformě Android
- Některé implementované algoritmy mají vyšší výkon než v OpenCV (SURF) [53]

4.3.2 Nevýhody

- Knihovna na rozdíl od OpenCV neobsahuje nástroje pro práci s algoritmy využívající hluboké učení a neuronové sítě

4.4 Shrnutí

V oblasti počítačového vidění je již dlouhou dobu nejpoužívanější knihovnou OpenCV. A to především díky široké škále funkcí a algoritmů, které nabízí a velkému množství programovacích jazyků, ve kterých je dostupná. Samozřejmě ne ve všech situacích je tato knihovna nejlepší volbou a je lepší zvolit alternativu. Vzhledem k popularitě OpenCV je alternativních knihoven dosahujících podobných kvalit a funkcionality poměrně malé množství. Každá z představených knihoven má své výhody, nevýhody a také okolnosti použití.

Pro studenty či akademiky se zde jako nejlepší alternativa nabízí knihovna Computer Vision Toolkit programu MATLAB, protože licence je často poskytována v rámci školy, nebo univerzity zdarma. Knihovna disponuje jak klasickými metodami počítačového vidění, tak i nástroji a algoritmy využívající hluboké učení a neuronové sítě, které se dnes často využívají při detekci objektů v reálném čase. Další velká výhoda je možnost využití grafické karty pro několikanásobné urychlení výpočtů a mnoha dalších funkcí, které MATLAB nabízí.

Pro vývojáře je zde jako nejlepší alternativa knihovna BoofCV z důvodu její obsáhlosti funkcí a algoritmů počítačového vidění, dostupnosti v mnoha programovacích jazycích a vyšší rychlosti oproti knihovně SimpleCV.

II. PRAKTICKÁ ČÁST

5 PŘEDSTAVENÍ APLIKACE

V praktické části bakalářské práce byla vytvořena webová aplikace pro analýzu a získávání dat o dopravě v reálném čase z webkamery za využití moderních algoritmů pro detekci a sledování objektů. Aplikace je napsána v programovacím jazyce Python za využití několika knihoven, které jsou popsány níže.

Tato aplikace je určena především pro soukromé společnosti, výzkumné ústavy nebo státní orgány pro získání přehledu o dopravní situaci v dané oblasti. Jsou zde využity moderní algoritmy pro detekci a následné sledování dopravy, díky kterým jsou získávána data o počtu, různorodosti a směru příjezdu/ odjezdu dopravy. Tato data jsou následně vizualizována a ukládána v databázi pro další možné zpracování a vytváření modelů pro zlepšení dopravní situace, nebo plánování uzavírek.

Aplikace tak může nahradit zaměstnance, nebo brigádníky, kteří jsou často využíváni pro počítání dopravy ve městech a jejich okolí. Příkladem může být akce Celostátní sčítání dopravy [54] z Centra dopravního výzkumu, kdy jsou získaná data jsou využívána pro plánování oprav pozemních komunikací, nebo zpracovávání dokumentace dopravních služeb.

5.1 Využité technologie

Vytvořená aplikace využívá OpenCV DNN modul pro detekci vozidel za využití modelu YOLOv4. Tyto detekce jsou dále předány Deep SORT algoritmu pro jejich sledování mezi jednotlivými snímky a následné počítání vozidel. Získávaná data jsou vizualizována v uživatelském rozhraní, které zprostředkovává framework Dash společně s knihovnou Plotly. Dále jsou data ukládána lokálně, nebo pomocí knihovny Pyodbc, která umožňuje připojení na MS SQL Server.

5.1.1 OpenCV 4.5.5-dev

Pro využití hardwarové akcelerace na grafické kartě a alternativní pipeline pro zpracování videa – GStreamer je nutné vytvořit vlastní build knihovny OpenCV. Prvním krokem je stažení všech potřebných knihoven pro framework GStreamer a ovladačů pro technologii CUDA. Následně jsou naklonovány repozitáře knihovny OpenCV, a to hlavní repozitář *opencv-python* a doplňující repozitář *opencv-contrib-python*. Tento repozitář kromě dodatečných modulů, které jsou stále ve vývoji, nebo nejsou dostatečně otestovány pro přidání

do hlavního repozitáře obsahuje algoritmy odladěné pro hardwarovou akceleraci pomocí technologie CUDA.

Po stažení všech potřebných knihoven a ovladačů proběhne konfigurace buildu knihovny pomocí programu CMake. Důležitými atributy jsou *-DWITH_GSTREAMER=ON* pro přidání frameworku GStreamer, *-DWITH_CUDA=ON a -DOPENCV_DNN_CUDA=ON* pro podporu CUDA backendu a *dnn.cuda* modulu. Nakonec proběhne samotný build knihovny pomocí příkazu *make -j8 install*. [55]

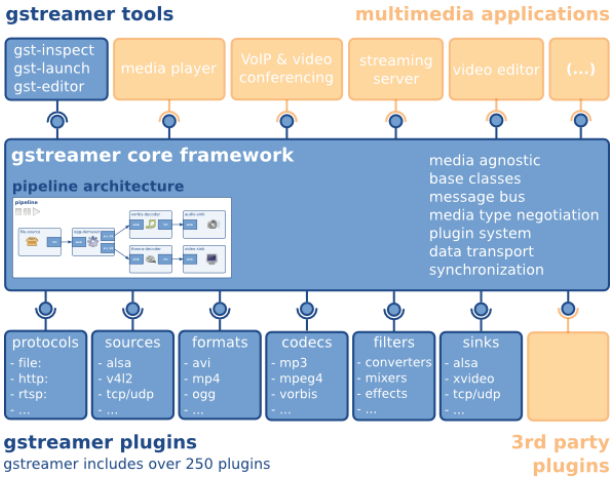
Zdrojový kód 8. Build knihovny OpenCV [55]

```
sudo apt install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev

git clone https://github.com/opencv/opencv
git clone https://github.com/opencv/opencv_contrib
mkdir /content/build
cd /content/build

cmake -DOPENCV_EXTRA_MODULES_PATH=/content/opencv_contrib/modules
-DCMAKE_BUILD_TYPE=RELEASE
-DINSTALL_C_EXAMPLES=OFF -DPYTHON_EXECUTABLE=$(which python3)
-D BUILD_opencv_python2=OFF -DPYTHON3_EXECUTABLE=$(which python3)
-DBUILD_EXAMPLES=ON -DBUILD_SHARED_LIBS=OFF
-DWITH_GSTREAMER=ON -DBUILD_TESTS=OFF
-DBUILD_PERF_TESTS=OFF -DBUILD_EXAMPLES=OFF
-DWITH_OPENEXR=OFF -DWITH_CUDA=ON -DWITH_CUBLAS=ON
-DOPENCV_DNN_CUDA=ON /content/opencv

make -j8 install
```

5.1.1.1 GStreamer

Jedná se open source framework pro práci s multimédii, který umožňuje vytvářet aplikace pracující s audiem a videem. GStreamer obsahuje velké množství kodeků a systémů pro zpracování audia a videa, které je možno využít v jedné pipeline.

Tento framework je využit jako alternativní pipeline v knihovně OpenCV pro zpracování živého přenosu. Důvody pro výběr této knihovny jsou popsány v kapitole 5.1.2. [56]



Obrázek 27. Architektura frameworku GStreamer [57]

5.1.1.2 CUDA

CUDA je platforma pro paralelní výpočty vyvinutá společností NVIDIA. Umožňuje využití grafických karet ve výpočetně náročných aplikacích a tím výrazné zvýšení výkonu, kterého je dosaženo díky paralelnímu využití tisíců GPU jader pro výpočty.

Hlavní využití se nachází ve výpočetně náročných aplikacích, které mohou benefitovat z paralelních výpočtů. Jedná se především o oblasti strojového učení, bioinformatiky, nebo vytváření modelů počasí. [58]

V aplikaci je tato platforma obsažena v knihovně OpenCV a je využita především pro zvýšení výkonu detekčního modelu YOLOv4.

5.1.2 CamGear

CamGear je jeden z modulů knihovny VidGear, která je určená pro práci s videem a jeho vysíláním. Jedná se o nástavbu funkce *VideoCapture()* z OpenCV. Díky robustnímu API umožňuje práci s mnoha typy video přenosů od IP kamer umístěných v lokální síti po živé přenosy vysílané na platformy jako je YouTube, nebo Twitch. Pro lepší výkon a synchronizaci knihovna využívá více vláknovou frontu, do které se dočasně ukládají stažená data. [59]

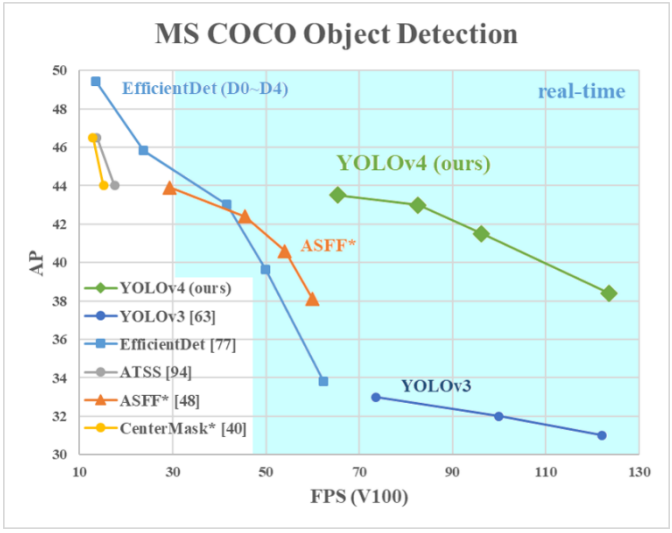
V aplikaci je tato knihovna využita pro stahování živého přenosu z platformy YouTube a byla vybrána kvůli podpoře alternativní knihovny pro dekódování videa, konkrétně knihovně GStreamer. Nahrazuje tak framework FFmpeg, který je využívaný ve většině knihoven, včetně OpenCV. Alternativní knihovna je potřeba z důvodu stálého stahování živého přenosu, protože se vývoji aplikace ukázalo, že jiné knihovny pro tyto účely využívající

FFmpeg, například Youtube-dl [60], nebo Streamlink [61], které byly při vyzkoušeny nezvládají kontinuálně zpracovávat HLS/ m3u8 živé přenosy a po pár minutách dochází k zastavení stahování dat (viz. komentář u Github Issue [62]).

5.1.3 YOLOv4

Tento model pro detekci objektů byl popsán v kapitole 2.3.3.3.

Velká snímková frekvence s relativně vysokou přesností detekce oproti alternativním modelům (viz. Obrázek. 28), jako například SSD, nebo Faster-RNN umožňuje efektivní využití pro detekci objektů v reálném čase. V aplikaci je využitý model předtrénovaný na datasetu MS COCO, který obsahuje 80 tříd objektů, zvířat a lidí. Konkrétně jsou z těchto tříd využity dopravní prostředky (třídy *car*, *bus*, *motorcycle*, *bicycle*, *truck*).



Obrázek 28. Porovnání výkonu a přesnosti YOLOv4 s ostatními modely [36]

5.1.4 Deep SORT

Deep SORT je algoritmus pro sledování více objektů najednou využívající hluboké učení a neuronové sítě. Jedná se o rozšíření původního modelu SORT pro sledování objektů v obraze o informace o vzhledu objektů.

Sledování mezi jednotlivými snímky probíhá na základě získaných dat z modelu pro detekci objektů (zde YOLOv4). Konkrétně jsou využívány ohraničující rámečky objektů. Sledování objektů tedy probíhá na základě vzdálenosti ohraničujících rámečků mezi jednotlivými snímky a informací o vzhledu objektu v daném rámečku, které jsou vypočítány. Právě díky

informacím o vzhledu je tento model mnohem úspěšnější při situacích, kdy se objekty překrývají, nebo jsou příliš blízko kameře. [63][64]

Tento způsob sledování objektů je v aplikaci využit z důvodu přesnosti sledování více objektů najednou a vysokého výkonu.

5.1.4.1 TensorFlow

Algoritmus Deep SORT pro svou funkci využívá knihovnu TensorFlow. Jedná se o platformu pro strojové učení od společnosti Google, která poskytuje nástroje a další knihovny pro vytváření, trénování a aplikaci neuronových sítí. Knihovna je dostupná ve více programovacích jazycích, konkrétně se jedná o Python, Javu a C++. [65]

5.1.5 Flask

Flask je framework pro vytváření webových stránek v Pythonu. Je klasifikován jako mikro framework, protože nevyžaduje žádné další nástroje a knihovny pro vytváření obsahu webové stránky. Využívá šablonovací systém Jinja2, který nahrazuje tradiční HTML kód. V porovnání s alternativními knihovnami, jako je například Django, je Flask více přehledný a vhodný spíše pro menší až střední aplikace. [66]

V aplikaci je knihovna využita především pro mapování cesty zpracovaných snímků videa pomocí *@app.route()* na adresu, ze které jsou následně načítány do elementu na úvodní stránce. Dále je framework využit jako základ pro Dash, což je nástavba frameworku Flask obsahující nástroje pro vizualizaci dat.

5.1.6 Dash

Jedná se o framework pro vytváření webových stránek v Pythonu zaměřený na analytické aplikace. Jak již bylo zmíněno výše, v základu se jedná o nástavbu frameworku Flask, který je doplněn o nástroje pro vizualizaci a také prvky z frameworku React určené pro front-end. Aplikace využívající Dash se skládá ze dvou hlavních částí. První části je layout, který obsahuje veškerý obsah, moduly uživatelského rozhraní a vizualizace dat. Druhou částí jsou tzv. callbacky, které slouží k propojení aplikační logiky a prvků v layoutu. K vizualizaci dat slouží knihovna Plotly [67] vytvořená stejnými autory, která umožňuje zobrazení jak tradičních grafů, tak i složitějších vizualizací, jako jsou například 3D modely objektů, nebo zobrazení dat v prostoru. [68]

Kromě Plotly lze doinstalovat i další knihovny, které přidávají moduly uživatelského rozhraní. V aplikaci je využita doplňková knihovna Dash_daq [69], která přidává moduly pro uživatelské vstupy, například pro výběr barvy, nebo posuvníky. Druhá využitá knihovna je Dash_bootstrap_components [70] která přidává komponenty z knihovny Bootstrap.

V aplikaci tento framework poskytuje veškeré uživatelské rozhraní a zobrazuje data.

5.1.7 Microsoft SQL Server

Microsoft SQL Server, také MS SQL je relační systém řízení báze dat (DBMS). Jedná se o systém ukládání dat, ve kterém se data nachází v tabulkách relačního schématu. Stejně jako ostatní DBMS používá programovací jazyk SQL pro svou funkci. Pro připojení na MS SQL Server je využita Python knihovna Pyodbc [71], která umožňuje připojení na relační databáze díky využití ODBC API.

V aplikaci je využit školní MS SQL server pro ukládání získaných dat o vozidlech do dvou tabulek, kdy v první tabulce se nachází data o vozidle, například typ vozidla, směr příjezdu, směr odjezdu, nebo čas. Ve druhé tabulce se nachází souřadnice jednotlivých vozidel v průběhu jejich jízdy. [72]

5.1.8 Pandas

Jedná se o open source knihovnu pro manipulaci a analýzu dat. Staví na knihovně Numpy, která je také určena pro manipulaci s daty a podporuje vícerozměrná pole. Pandas implementuje vlastní verzi vícerozměrných polí, nazývanou DataFrame. V rámci tohoto objektu lze provádět velké množství operací s daty, jako například porovnávání, třídění a rozdělování dat a další operace. Je zde také umožněno importování dat z různých zdrojů, může se jednat o soubory jako JSON, nebo CSV ale i data z SQL serverů. [73]

V aplikaci je tato knihovna využita především pro práci se získanými daty a jejich úpravu pro následnou vizualizaci.

5.2 Popis funkce

Aplikace je rozdělena do několika souborů dle odpovídající funkce. Hlavním souborem je *app.py*, ve kterém se nachází jádro aplikace a callbacky pro elementy uživatelského prostředí. Ve složce projektu je také obsažena složka *deep_sort*, která byla stažena z Github repozitáře [63] a nachází se v ní soubory potřebné pro fungování algoritmu Deep SORT.

Hlavní smyčka, ve které probíhá zpracování obrazu se nachází ve funkci *run(stream)*, která jako vstupní parametr přijímá objekt třídy *VideoStream()*. Na začátku smyčky se pomocí funkce *getFrame()* získá aktuální snímek z video streamu. Následně je tento snímek spolu se seznamem požadovaných tříd pro detekci předán do funkce *detectVehicles(frame, ALLOWED_CLASSES)*. Výstupem z této funkce jsou seznamy detekovaných tříd, důvěry (confidence) a souřadnice bodů ohraničujících rámečků objektů. Tyto tři seznamy jsou dále předány do funkce *trackVehicles(frame, data, points, classes, confidences, boxes)*, která jako vstupní parametry dále přijímá snímek z video streamu a dva dataframy pro ukládání získaných dat. Výstupem z této funkce jsou již zmíněné dataframy doplněné o získaná data a také snímek, ve kterém dle zvoleného nastavení v uživatelském rozhraní můžou být znázorněny trasy vozidel, nebo detekční polygony.

V této smyčce také dochází k ukládání dat, a to v nastaveném intervalu dle počtu zpracovaných snímků. Podle zvolené metody ukládání jsou data zpracována buď funkcí *insert(data, points)*, kdy proběhne uložení do databáze, anebo pomocí funkce *saveData(data, points)*, kdy jsou data uložena lokálně ve formátu CSV. Následně jsou z dataframu pozičních bodů vozidel odstraněna uložená data. Data vozidel nejsou smazána, protože jsou využívána pro vizualizaci.

Po případném uložení dat je snímek pomocí funkce *imencode()* enkódován do formátu vhodného pro přenos, aby jej bylo možné odeslat na definovanou adresu */video_stream* pro následné zobrazení v uživatelském rozhraní. Po dokončení těchto operací je na základě času z předchozího snímku a aktuálního času vypočítána snímková frekvence, která udává rychlost běhu programu. Nakonec je enkódovaný snímek převeden z pole na řetězec bytů a odeslán na definovanou adresu pomocí *yield*. Využití klíčového slova *yield*, které vrací generátor místo tradičního *return* umožňuje opakované odesílání snímků a tím plynulé zobrazování zpracovaného živého přenosu v uživatelském rozhraní.

Zdrojový kód 9. Hlavní smyčka

```
def run(stream):
    global data, points, frame, fps
    prevTime = 0
    frameCount = 0
    while True:
        frame = camera.getFrame()
        frameCount += 1


        classes, confidences, boxes = detectVehicles(frame, ALLOWED_CLASSES)
        frame, data, points = trackVehicles(frame, data, points,
                                            classes, confidences, boxes)


        if frameCount % 4500 == 0:
            if USE_DB:
                insert(data, points)
            else:
                saveData(data, points)
            points = points[0:0]


        _, jpeg = cv2.imencode('.jpg', frame)
        currTime = time.time()
        fps = 1 / (currTime - prevTime)
        prevTime = currTime


        yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' +
               jpeg.tobytes() + b'\r\n\r\n')
```

5.2.1 Získávání obrazu

Obraz z webkamery je získáván pomocí třídy *VideoStream*, která obsahuje atribut *self.stream* a metodu *get_frame()*. Při vytvoření objektu třídy je v konstruktoru inicializováno spojení s živým přenosem pomocí třídy *CamGear*, která jako parametry přijímá URL zdroje přenosu (může se jednat o odkaz na Youtube livestream, ale také adresu přenosu v místní síti), druhý parametr povoluje stream mód potřebný pro stahování živého přenosu. Poslední parametr je slovník s nastavením videa.

Metoda *get_frame()* slouží pro získávání jednotlivých snímků z daného živého přenosu. Po stažení je snímek transformován do požadované velikosti za pomoci funkce *resize()*, která jako parametry přijímá obraz, dvojici obsahující šířku a výšku videa a nakonec způsob interpolace.

Při vývoji byla využita kamera umístěná na Zlínském divadle, která má veřejný živý přenos na Youtube. [74]

Zdrojový kód 10. Třída VideoStream

```
class VideoStream():

    def __init__(self):
        self.stream = CamGear(source=os.getenv('CAM_URL'),
                              stream_mode=True,
                              **options).start()

    def __del__(self):
        self.stream.stop()


    def get_frame(self):
        img = self.stream.read()
        img = cv2.resize(img, (WIDTH, HEIGHT), interpolation=cv2.INTER_AREA)
        return img
```

5.2.2 Detekce vozidel

Pro detekci vozidel je využit model YOLOv4, který je inicializován ve funkci *initYOLO()*. Prvním krokem je načtení tříd datasetu MS COCO, na kterém je tento model natrénován. Dále jsou načteny soubory pro tento model, konkrétně se jedná o konfigurační soubor a soubor s váhami. Tyto soubory byly staženy z oficiálního repozitáře modelu YOLOv4. [75]

V dalším kroku je pomocí funkce *cuda.getCudaEnabledDeviceCount()* zkontrolována dostupnost GPU v systému. V případě splnění podmínky je povolena hardwarová akcelerace na dané grafické kartě. Jedná se o důležitý krok, který umožňuje plynulý běh aplikace, protože i když byl zvolen jeden z nejvýkonnějších modelů pro detekci objektů pomocí hlubokých neuronových sítí, jedná se o stále velmi výpočetně náročnou operaci a při využití pouze procesoru není dosaženo požadované rychlosti.

Zdrojový kód 11. Funkce pro inicializaci modelu YOLOv4

```
def initYOLO():
    with open('./application/resources/coco.names', 'rt') as f:
        class_names = f.read().rstrip('\n').split('\n')


    net=cv2.dnn.readNetFromDarknet('./application/resources/yolov4.cfg',
                                   './application/resources/yolov4.weights')


    if cv2.cuda.getCudaEnabledDeviceCount() > 0:
        net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
        net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)


    model = cv2.dnn_DetectionModel(net)
    model.setInputParams(size=(WIDTH, HEIGHT), scale=1/255)


    return model, class_names
```

Detekce vozidel ve snímku videa probíhá pomocí funkce *detectVehicles(frame, ALLOWED_CLASSES)*, která má jako první vstupní parametr aktuální snímek živého přenosu. Druhý parametr je seznam požadovaných tříd vozidel (například auto, autobus, motorka atd). Nejprve je na snímek aplikována maska, která je definována uživatelem. Tato maska má za úkol zabránit nechtěným detekcím, například pokud se v zorném poli kamery nachází parkoviště. V dalším kroku je do inicializovaného detekčního modelu předán upravený snímek spolu s nastavenými prahy. Výstupem z modelu jsou tři seznamy. První obsahuje počty vozidel každé třídy, které byly detekovány, ve druhém seznamu se nachází míry důvěry (confidence) a v posledním souřadnice ohraničujícího rámečku každého vozidla. Následně jsou získaná data vyfiltrována dle požadovaných tříd, aby v následujícím kroku probíhalo sledování pouze žádaných tříd vozidel.

Zdrojový kód 12. Funkce pro detekci vozidel v obraze

```
def detectVehicles(frame, ALLOWED_CLASSES):
    global model, class_names, tracker, encoder, vID


    frame_masked = frame.copy()
    if len(mask) > 0:
         for name, poly in mask.items():
            cv2.fillPoly(frame_masked,
                         [np.array(poly, np.int32)],
                         (0, 0, 0))
    classes, confidences, boxes = model.detect(frame_masked,
                                               confThreshold=float(
                                               os.getenv(
                                                'CONFIDENCE_THRESHOLD')),
                                               nmsThreshold=0.4)


    # Boxes filtering - Passsing only specific classes to tracker
    filtered_boxes = []
    filtered_classes = []
    filtered_confidences = []
    for i in range(len(boxes)):
        if class_names[int(classes[i])] in ALLOWED_CLASSES:
            filtered_boxes.append(boxes[i])
            filtered_classes.append(class_names[int(classes[i])])
            filtered_confidences.append(confidences[i])
    filtered_classes = np.array(filtered_classes)
    filtered_boxes = np.array(filtered_boxes)
    filtered_confidences = np.array(filtered_confidences)


    return filtered_classes, filtered_confidences, filtered_boxes
```

5.2.3 Sledování a počítání vozidel

Pro sledování vozidel je využit algoritmus Deep SORT, který je inicializován ve funkci *initDeepSort()*. Prvním krokem je načtení modelu ze souboru a z něj následné vytvoření enkodéru, který slouží pro zpracování částí obrazu s detekovanými objekty. Následně je

vytvořena instance třídy *Tracker*, která slouží pro samotné sledování. Vstupním parametrem je zvolená metoda výpočtu vzdálenosti objektů mezi jednotlivými snímky. Model byl spolu s potřebnými soubory pro funkci algoritmu stažen z oficiálního Github repozitáře. [63]

Zdrojový kód 13. Funkce pro inicializaci modelu Deep SORT [63]

```
def initDeepSort():
    model = './application/resources/mars-small128.pb'
    encoder = gdet.create_box_encoder(model, batch_size=1)
    metric = nn_matching.NearestNeighborDistanceMetric(
                                            "cosine", matching_threshold=0.5)
    tracker = Tracker(metric)
    return tracker, encoder
```

Samotné sledování detekovaných vozidel probíhá ve funkci *trackVehicles(frame, data, points, classes, confidences, boxes)*, která jako vstupní parametry přijímá aktuální snímek, dataframy pro uložení získaných dat a 3 seznamy získané z funkce *detectVehicles()*.

Nejprve je pomocí funkce *encoder()* a získaných ohraničujících rámečků vytvořen seznam ROI detekovaných objektů. Dále jsou z tohoto seznamu a dalších vstupních parametrů (*boxes, confidences, classes*) vytvořeny objekty třídy *Detection* a nakonec pomocí třídy *Tracker* zjištěny pohyby vozidel mezi aktuálním a předchozím snímkem. Výstupem je seznam sledovaných objektů třídy *Track*, do které bylo pro potřeby aplikace přidáno několik atributů. Konkrétně se jedná o *self.origin* a *self.exit*, které slouží pro uložení informací o směru/ ulici ze které vozidlo přijelo a odjelo. Další přidaný atribut je *self.counted*, který udává, zda bylo dané vozidlo již spočítáno a jeho data uložena. Poslední atribut je *self.points* a jedná se o seznam, do kterého se ukládají poziční body vozidla.

Zdrojový kód 14. Část funkce pro sledování vozidel

```
def trackVehicles(frame, data, points, classes, confidences, boxes):
    features = encoder(frame, boxes)

    detections = [Detection(bbox, score, class_name, feature)
                  for bbox, score, class_name, feature in
                  zip(boxes, confidences, classes, features)]


    tracker.predict()
    tracker.update(detections)
```

Po získání seznamu sledovaných objektů s aktuálními pozicemi se jednotlivá vozidla prochází ve smyčce. Ze souřadnic ohraničovacího rámečku je vypočten střed objektu, který je následně vizualizován pomocí kreslících funkcí OpenCV a uložen do atributu *self.points*, ze kterého je následně možno vizualizovat trasy projíždějících vozidel. Dalším krokem je počítání a ukládání dat daného vozidla. Aplikace umožňuje buď počítání všech vozidel s tím, že jsou zaznamenána pouze data o třídě vozidla a času kdy vozidlo projíždělo, anebo pomocí tzv. detekčních oblastí, které jsou podobně jako maska vytvořeny uživatelem. Každá z těchto detekčních oblastí by měla být jednou z možností, kam může vozidlo odbočit. Pro zjištění, zda se vozidlo nachází v jedné z detekčních oblastí je využita OpenCV funkce *pointPolygonTest()*, která jako vstupní parametry přijímá seznam bodů detekční oblasti a střed vozidla, který byl vypočítán na začátku smyčky. Tato funkce může vrátit 3 různé hodnoty – 1, pokud se střed vozidla nachází uvnitř dané detekční oblasti, -1, pokud se nachází vně oblasti a 0, pokud je přímo na hranici. Dle výstupu z této funkce je tedy vozidlu buď zaznamenán příjezd z dané oblasti (pokud má atribut příjezdu defaultní hodnotu *None*), nebo odjezd ve směru této oblasti (pokud má atribut příjezdu již uloženou hodnotu nějaké jiné detekční oblasti a zároveň má atribut pro odjezd defaultní hodnotu *None*). Jakmile jsou získány informace o příjezdu a odjezdu vozidla jsou tato data společně se třídou a aktuálním časem uložena do dataframu. To samé platí pro poziční body, které jsou uloženy do příslušného dataframu. Posledním krokem je označení daného vozidla za spočítané pomocí atributu *self.counted*, aby nedocházelo ke zdvojení informací.

Zdrojový kód 15. Počítání vozidel pomocí detekčních oblastí – získání středu vozidla

```
for track in tracker.tracks:
        if not track.is_confirmed() or track.time_since_update > 1 or
        track.counted:

            continue

        box = track.to_tlbr().astype('int32')
        left, top, width, height = box


        cx = int((left + width) / 2)
        cy = int((top + height) / 2)
        cv2.circle(frame, (cx, cy), 5, color, -1)
```

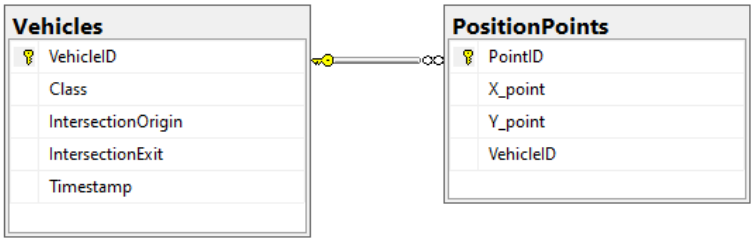
Zdrojový kód 16. Počítání vozidel pomocí detekčních oblastí

```
if not track.counted:
   track.points.append((cx,cy))

    if len(routes) > 0:
        for name, poly in routes.items():

            result = cv2.pointPolygonTest(np.array(poly, np.int32),
                                          (int(cx), int(cy)), False)

            if result >= 0 and track.origin is None:
                track.origin = name
                break

            if result >= 0 and track.origin is not None and
               name != track.origin:

                track.exit = name

                data = data.append({'VehicleID': vID,
                                    'Class': track.get_class(),
                                    'IntersectionOrigin': track.origin,
                                    'IntersectionExit': track.exit,
                                    'Timestamp': datetime.now().strftime(
                                       "%Y-%m-%d %H:%M:%S")},
                                    ignore_index=True)

                pointsDF = pd.DataFrame(track.points,
                           columns=['X_point', 'Y_point'])
                pointsDF['VehicleID'] = vID
                points = pd.concat([points, pointsDF], ignore_index=True)

                track.counted = True
                break
```

5.2.4 Ukládání dat

Získaná data jsou v pravidelných intervalech ukládána do uživatelem zvoleného místa. Primární způsob ukládání je databáze, do které jsou data odesílána. Pro demonstrativní účely byla vytvořena databáze na školním MS SQL serveru, která obsahuje 2 tabulky. První tabulka s názvem *Vehicles* slouží pro zaznamenávání dat o vozidlech. Konkrétně se zde nachází informace o třídě vozidla, směru, ze kterého vozidlo přijelo a kam odjelo a v neposlední řadě datum a čas. Druhá tabulka, *PositionPoints*, slouží pro zaznamenávání

jednotlivých bodů v obraze, kudy dané vozidlo projíždělo. V každém záznamu jsou tedy souřadnice daného bodu a ID vozidla, ke kterému tento bod patří.



Obrázek 29. Diagram vytvořené databáze

VehicleID	Class	IntersectionOrigin	IntersectionExit	Timestamp
1	Car	TB_Nemocnice	Dlouha	2022-03-05 16:57:46.000
2	Car	TB_Nemocnice	TB_Malenovice	2022-03-05 16:58:26.000
3	Car	TB_Nemocnice	TB_Malenovice	2022-03-05 16:58:35.000
4	Car	Osvoboditelu	TB_Nemocnice	2022-03-05 17:00:38.000
5	Car	Dlouha	TB_Malenovice	2022-03-05 17:01:58.000

PointID	X_point	Y_point	VehicleID
1	1219	401	1
2	1195	401	1
3	1123	397	1
4	1047	392	1
5	963	387	1

Obrázek 30. Ukázka získaných dat

Funkce pro uložení dat *insert(data, points)* má první vstupní parametr dataframe s detekovanými vozidly a druhý vstupní parametr dataframe s pozičními body těchto vozidel. Po připojení k databázi jsou vybrána pouze ta data vozidel, která se v databázi nenachází (tento dataframe se je také využit pro vizualizaci dat, proto obsahuje všechna vozidla z daného dne) a následně jsou po jednotlivých vozidlech nahrána do databáze. Poziční body se do databáze nahrávají všechny, protože nejsou vizualizovány v grafech a jejich dataframe je po každém uložení vyčištěn. V případě nedostupnosti databáze jsou data uložena do CSV souboru pomocí funkce *saveData(data, points)*.

Zdrojový kód 17. Funkce pro ukládání dat na MS SQL server

```
def insert(data, points):
    try:
        lastID = getLastID()
        data = data.loc[(data['VehicleID'] > lastID)]


        for v in data.iterrows():
            cursor.execute("INSERT INTO Vehicles(VehicleID, Class,
                           IntersectionOrigin, IntersectionExit, Timestamp)
                           values(?,?,?,?,?)", v['VehicleID'], v['Class'],
                           v['IntersectionOrigin'], v['IntersectionExit'],
                           v['Timestamp'])

        for point in points.iterrows():
            cursor.execute("INSERT INTO PositionPoints(X_point, Y_point,
                           VehicleID) values (?,?,?)", point['X_point'],
                           point['Y_point'], point['VehicleID'])

        conn.commit()
        cursor.close()

    except:
        saveData(data, points)
```

V případě potřeby je možné data z databáze stáhnout přímo v aplikaci. V uživatelském rozhraní uživatel vybere období, ze kterého chce data získat a jejich typ (data o vozidlech – tabulka *Vehicles*, nebo poziční body z tabulky *PositionPoints*). Data jsou následně stažena ve formátu CSV.

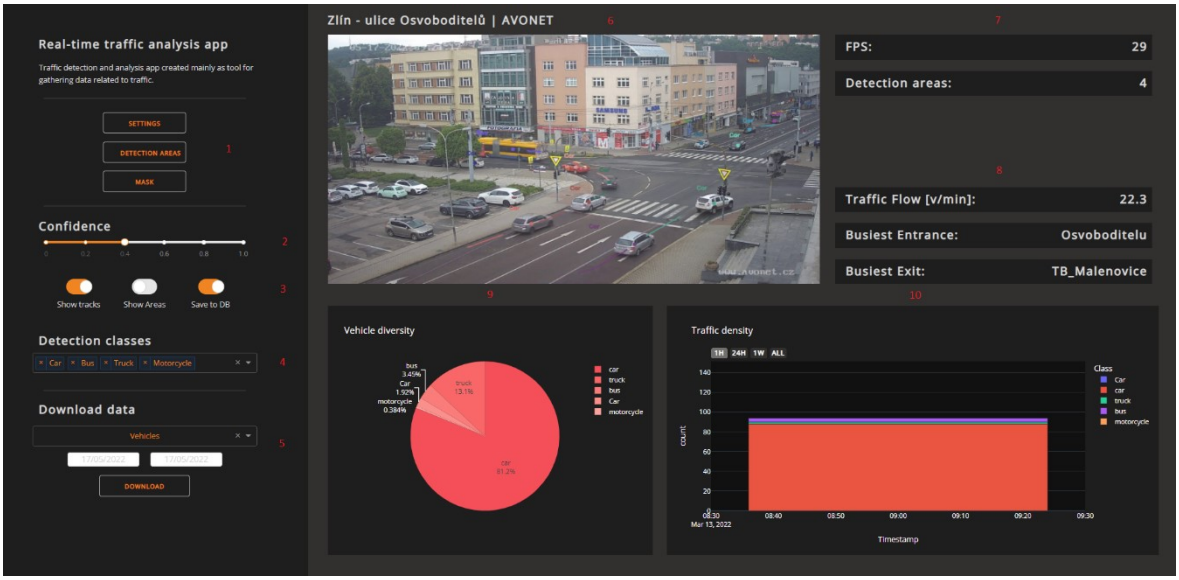
Druhá možnost pro ukládání souborů je do lokálního úložiště ve formátu CSV. Pokud uživatel nevyžaduje využití databáze, může v uživatelském rozhraní změnit nastavení ukládání dat. Také v případě, že se nepodaří připojit k databázi pro uložení dat, jsou tato data uložena lokálně, aby nedošlo k jejich ztrátě.

5.3 Uživatelské rozhraní

Jak bylo uvedeno v kapitole 5.1.6, uživatelské rozhraní je vytvořeno pomocí knihovny Dash. Úvodní stránka aplikace se dá rozdělit na dvě hlavní části. První částí je boční panel, který obsahuje nastavení aplikace a možnost stažení získaných dat z databáze. Druhou část tvoří

zbytek úvodní stránky a obsahuje zpracovaný živý přenos, informace o snímkové frekvenci, statistiky z poslední hodiny a základní vizualizace dat z posledních 24 hodin.

Základ CSS stylů pro knihovnu Plotly.js byl stažen z následující webové stránky [76].



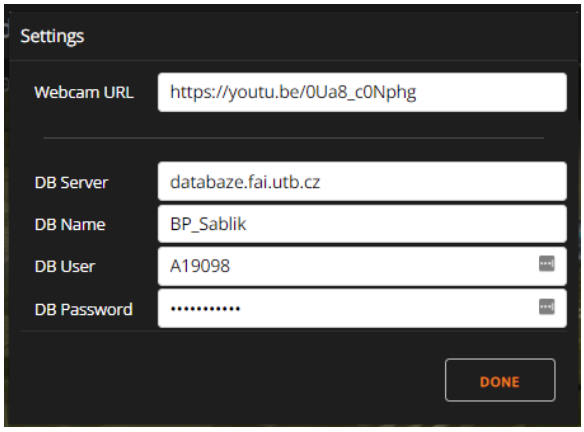
Obrázek 31. Uživatelské rozhraní

V bočním panelu se nachází tlačítka pro zobrazení jednotlivých modalů (1), které obsahují nastavení a nástroje pro vytváření masek a detekčních oblastí. Pod tlačítky je umístěn posuvník (2), kterým lze ovládat práh důvěry pro detekční model. Tuto hodnotu může být potřeba upravit, aby nedocházelo k falešným detekcím na základě používané webové kamery a světelných podmínek. Dále se zde nachází ovládací prvky (3) pro vizualizaci trasy jednotlivých vozidel, zobrazení detekčních oblastí a povolení ukládání do databáze. Pod ovládacími prvky je rozbalovací seznam (4), ve kterém jsou vybrány požadované třídy pro detekci. Poslední částí bočního panelu je stažení dat z databáze (5), která obsahuje kalendář pro výběr časového období, rozbalovací seznam s výběrem druhu dat, a nakonec tlačítko pro stažení.

Ve druhé části se nachází název aktuálního živého přenosu a zobrazení zpracovaných snímků (6). Na pravé straně jsou uvedeny informace o aktuální snímkové frekvenci a počtu definovaných detekčních oblastí (7). Níže se nachází statistiky z dat získaných za poslední hodnu. Konkrétně je zde zobrazen tok dopravy, který udává počet projíždějících vozidel za minutu, a nejrušnější příjezdová a odjezdová cesta (8). Pod zobrazením živého přenosu se nachází koláčový graf (9) zobrazující skladbu tříd vozidel za posledních 24 hodin. Poslední částí je sloupcový graf (10), který vizualizuje míru dopravy v průběhu dne.

5.3.1 Nastavení

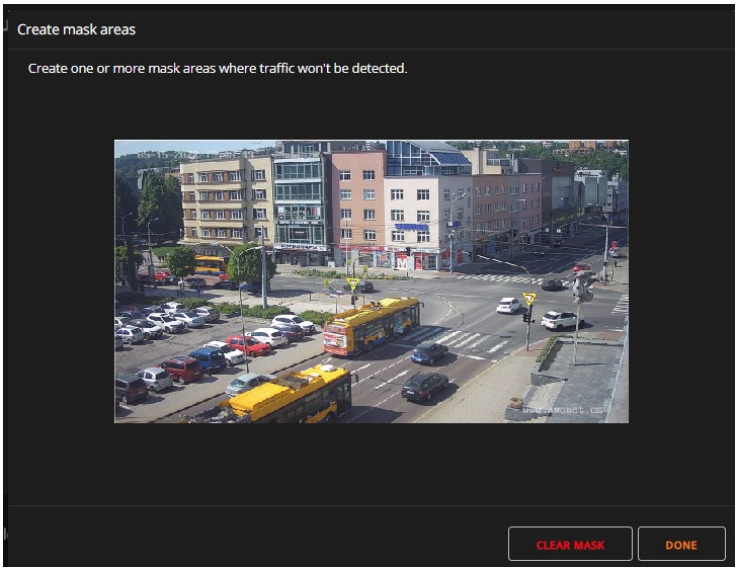
V tomto modalu (vyskakovacím okně) se nachází základní nastavení aplikace. Jedná se především o URL, ze které se má získávat živý přenos. Může se jednat o adresu YouTube videa, jako v obrázku 32. níže, ale lze využít i rstp, rtmp a další síťové adresy. Dále se zde nachází pole pro adresu serveru databáze, název databáze a přihlašovací údaje uživatele. Tato data jsou ukládána do .env souboru. Pro práci s .env soubory je využita knihovna python-dotenv. [77]



Obrázek 32. Modal s nastavením aplikace

5.3.2 Vytváření masky

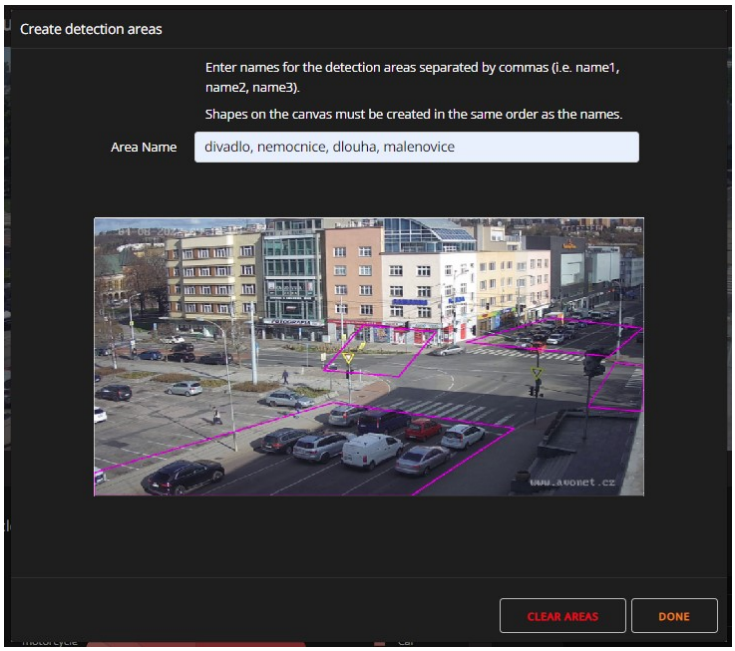
V případě, že se v zorném poli kamery nachází oblast, kde je nežádoucí detekovat dopravu, lze pomocí tohoto modalu vytvořit masku, která dané oblasti zakryje. Pro tyto účely je využito nástrojů pro anotace obrázků z knihovny Plotly. Na pozadí se vždy nachází snímek z aktuálně používaného živého přenosu a uživatel na základě tohoto pozadí označí oblast pro vytvoření masky. Následně jsou body vytvořeného polygonu jsou uloženy do JSON souboru a dále používány při detekci vozidel.



Obrázek 33. Modal pro vytvoření masky

5.3.3 Vytváření detekčních oblastí

Modal pro vytváření detekčních oblastí funguje na stejném principu jako vytváření masky, avšak zde jsou vytvářené polygony pojmenovány uživatelem. Tato pojmenování jsou později využívána při ukládání dat. Body polygonů jsou opět uloženy v JSON souboru a následně při sledování vozidel využívány pro kontrolu, zda se střed vozidla v daném polygonu nachází.



Obrázek 34. Modal s vyznačenými detekčními oblastmi

ZÁVĚR

Cílem této bakalářské práce bylo v teoretické části seznámit čtenáře s oblastí počítačového vidění, detekcí objektů, zpracování obrazu a také knihovnou OpenCV. V praktické části byla popsána vytvořená aplikace pro analýzu a získávání dat o dopravě v reálném čase, její použití a fungování.

V první kapitole, která je zaměřena na zpracování obrazu byly nejprve popsány základy zpracování obrazu a pipeline. Ve druhé části kapitoly byly představeny dvě moderní metody zpracování obrazu. První vybraná metoda je Tesla Vision. Jedná se o systém autonomního řízení, který využívá tisíců 360° videí z flotily vozidel Tesla, ze kterých je vytvořen vektorový prostor určený pro trénování neuronových sítí. Druhá metoda je NVIDIA DLSS, která je využívána především v oblasti počítačových her pro upscaling obrazu do vyššího rozlišení. Toho je dosaženo díky neuronové síti trénované porovnáváním dvojic snímků, kdy jeden snímek je v normálním a druhý ve vysokém rozlišení.

Druhá kapitola popisuje metody pro detekci objektů, jejich rozdělení a princip funkce. Byly zde představeny dvě hlavní skupiny algoritmů. V první skupině se nachází tradiční algoritmy, které nevyužívají hluboké učení a neuronové sítě. Konkrétně jsou zde popsány metody Viola-Jones, HOG, porovnávání šablon a optický tok. Do druhé skupiny patří moderní algoritmy využívající hluboké učení a neuronové sítě. Je zde popsáno jejich další dělení na jednofázové a dvoufázové a konkrétní zástupci těchto podskupin, R-CNN a YOLO.

Třetí kapitola je věnována knihovně OpenCV, jejímu vzniku a funkci. Dále jsou zde popsány moduly, které knihovna obsahuje a konkrétní funkce využívané v praktické části práce, nebo při detekci objektů obecně.

Čtvrtá kapitola představuje alternativní knihovny počítačového vidění a jejich výhody či nevýhody oproti OpenCV. Konkrétně se jedná o SimpleCV, BoofCV a Computer Vision Toolkit z programu MATLAB.

V praktické části bakalářské práce byla vytvořena aplikace pro analýzu a získávání dat o dopravě v reálném čase za využití moderních algoritmů pro detekci a sledování objektů. Využité této aplikace se nachází především ve výzkumných ústavech, úřadech, nebo soukromých společnostech pro získání přehledu o dopravní situaci, vytváření dopravních modelů, nebo plánování uzavírek. Aplikace umožňuje nahrazení brigádníků, kteří jsou často využíváni pro ruční počítání dopravy a tím i zautomatizování celého procesu.

SEZNAM POUŽITÉ LITERATURY

[1] DEY, Sandipan. *Hands-on image processing with Python: Expert techniques for advanced image analysis and effective interpretation of image data* [online]. Birmingham: Packt Publishing, Limited, 2018 [cit. 2022-01-21]. ISBN 978-178-9343-731. Dostupné z: https://www.packtpub.com/product/hands-on-image-processing-with-python/9781789343731

[2] FIŘT, Jaroslav a Radek HOLOTA. *Digitalizace a zpracování obrazu* [online]. Nové technologie–výzkumné centrum, Plzeň, 2002, [cit. 2022-01-23]. Dostupné z: http://home.zcu.cz/~holota5/publ/DigZprO.pdf

[3] VILLÁN, Alberto Fernández. *Mastering OpenCV 4 with Python: A practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7* [online]. Birmingham: Packt Publishing, 2019 [cit. 2022-01-23]. ISBN 978-1-78934-491-2. Dostupné z: https://www.packtpub.com/product/mastering-opencv-4-with-python/9781789344912

[4] Autopilot. Tesla [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://www.tesla.com/autopilot

[5] Tesla Autonomy Day. Youtube [online]. 22. 4.2019 [cit. 2022-05-08]. Dostupné z: https://youtu.be/Ucp0TTmvqOE

[6] Technologie Deep Learning Super Sampling (DLSS). NVIDIA [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://www.nvidia.com/cs-cz/geforce/technologies/dlss/

[7] BURNES, Andrew. NVIDIA DLSS 2.0: A Big Leap In AI Rendering. NVIDIA [online]. 23.3.2020 [cit. 2022-05-08]. Dostupné z: https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/

[8] HARDING, Scharon. What Is Nvidia DLSS? A Basic Definition. Tom's Hardware [online]. New York: Future US, 23.6.2021 [cit. 2022-05-08]. Dostupné z: https://www.tomshardware.com/reference/what-is-nvidia-dlss

[9] PANCHAL, Payal, Gaurav PRAJAPATI a Savan PATEL. A Review on Object Detection and Tracking Methods. International Journal for Research in Emerging Science and Technology [online]. 2015, (Volume-2, Issue-1), 6 [cit. 2022-05-08]. e-ISSN: 2349-7610. Dostupné z: https://www.ijrest.net/downloads/volume-2/issue-1/pid-21201506.pdf

[10] SHANTAIYA, Sanjivani, Keshri VERMA a Kamal MEHTA. A Survey on Approaches of Object Detection. International Journal of Computer Applications [online]. IJCA Journal, 18.3.2013, 65(18) [cit. 2022-05-08]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.303.826&rep=rep1&type=pdf

[11] VIOLA, P. a M. JONES. Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001 [online]. IEEE Comput. Soc, 2001, I-511-I-518 [cit. 2022-03-20]. ISBN 0-7695-1272-0. Dostupné z: doi:10.1109/CVPR.2001.990517

[12] KIM, Mooseop, Deokgyu LEE a Ki-Young KIM. System Architecture for Real-Time Face Detection on Analog Video Camera. International Journal of Distributed Sensor Networks [online]. 2015, 11(5) [cit. 2022-05-08]. ISSN 1550-1477. Dostupné z: doi:10.1155/2015/251386

[13] CEN, Kaiqi. Study of Viola-Jones Real Time Face Detector [online]. 2016 [cit. 2022-05-08]. Dostupné z: https://web.stanford.edu/class/cs231a/prev_projects_2016/cs231a_final_report.pdf. Stanford University.

[14] Template Matching. Adaptive Vision [online]. 2017 [cit. 2022-05-08]. Dostupné z: https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html

[15] HASHEMI, Nazanin Sadat, Roya Babaie AGHDAM a Atieh Sadat Bayat GHIASI. Template Matching Advances and Applications in Image Analysis [online]. 23.10.2016 [cit. 2022-05-08]. Dostupné z: doi:10.48550/arXiv.1610.07231

[16] DALAL, N. a B. TRIGGS. Histograms of Oriented Gradients for Human Detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) [online]. IEEE, 2005, 886-893 [cit. 2022-05-08]. ISBN 0-7695-2372-2. Dostupné z: doi:10.1109/CVPR.2005.177

[17] Histogram of Oriented Gradients (HOG) Descriptor. Intel [online]. 4.11.2022 [cit. 2022-05-08]. Dostupné z: https://www.intel.com/content/www/us/en/develop/documentation/ipp-dev-reference/top/volume-2-image-processing/computer-vision/feature-detection-functions/histogram-of-oriented-gradients-hog-descriptor.html

[18] CHOPRA, Eklavya. Using Histogram of Oriented Gradients (HOG) for Object Detection [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://iq.opengenus.org/object-detection-with-histogram-of-oriented-gradients-hog/

[19] Optical Flow. OpenCV [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html

[20] PATAIT, Abhijit. An Introduction to the NVIDIA Optical Flow SDK. NVIDIA Developer [online]. 13.2.2019 [cit. 2022-05-08]. Dostupné z: https://developer.nvidia.com/blog/an-introduction-to-the-nvidia-optical-flow-sdk/

[21] AGARWAL, Anshuman, Shivam GUPTA a Dushyant Kumar SINGH. Review of optical flow technique for moving object detection. 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I) [online]. IEEE, 2016, 409-413 [cit. 2022-05-08]. ISBN 978-1-5090-5256-1. Dostupné z: doi:10.1109/IC3I.2016.7917999

[22] SASTRASINH, Paul. Dense Realtime Optical Flow on the GPU. Brown University [online]. 2011 [cit. 2022-05-08]. Dostupné z: http://cs.brown.edu/courses/csci1290/2011/results/final/psastras/

[23] GOLLAPUDI, Sumila. Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs [online]. Berkeley, CA: Apress, 2019 [cit. 2022-05-08]. ISBN 978-1-4842-4261-2. Dostupné z: https://doi.org/10.1007/978-1-4842-4261-2

[24] What Is Object Detection?: 3 things you need to know. MathWorks [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://www.mathworks.com/discovery/object-detection.html

[25] WENG, Lilian. Object Detection for Dummies Part 3: R-CNN Family. Lil'Log [online]. 31.12.2017 [cit. 2022-05-08]. Dostupné z: https://lilianweng.github.io/posts/2017-12-31-object-recognition-part-3/

[26] GIRSHICK, Ross, Jeff DONAHUE a Trevor DARRELL. Rich feature hierarchies for accurate object detection and semantic segmentation: Tech report (v5) [online]. 2014 [cit. 2022-05-08]. Dostupné z: https://doi.org/10.48550/arXiv.1311.2524. UC Berkeley.

[27] GIRSHICK, Ross. Fast R-CNN [online]. Microsoft Research, 27.9.2015 [cit. 2022-05-08]. Dostupné z: doi:10.48550/arXiv.1504.08083

[28] REN, Shaoqing, Kaiming HE a Ross GIRSHICK. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [online]. 6.1.2016 [cit. 2022-05-08]. Dostupné z: doi:10.48550/arXiv.1506.01497

[29] GANDHI, Rohith. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms: Understanding object detection algorithms. Towards Data Science [online]. Medium, 9.7.2018 [cit. 2022-05-08]. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[30] REDMON, Joseph, Santosh DIVVALA a Ross GIRSHICK. You Only Look Once: Unified, Real-Time Object Detection [online]. University of Washington, 9.5.2016 [cit. 2022-05-08]. Dostupné z: doi:10.48550/arXiv.1506.02640

[31] REZATOFIGHI, Hamid, Nathan TSOI a Jun Young GWAK. Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [online]. Stanford University, 6. 2019 [cit. 2022-05-08]. Dostupné z: https://giou.stanford.edu/

[32] REDMON, Joseph Chet. YOLO: Real-Time Object Detection [online]. 2016 [cit. 2022-05-08]. Dostupné z: https://pjreddie.com/darknet/yolov1/

[33] WENG, Lilian. Object Detection Part 4: Fast Detection Models. Lil'Log [online]. 27.12.2018 [cit. 2022-05-08]. Dostupné z: https://lilianweng.github.io/posts/2018-12-27-object-recognition-part-4/

[34] REDMON, Joseph a Ali FARHADI. YOLO9000: Better, Faster, Stronger [online]. University of Washington, 25.12.2016 [cit. 2022-05-08]. Dostupné z: doi:10.48550/arXiv.1612.08242

[35] REDMON, Joseph a Ali FARHADI. YOLOv3: An Incremental Improvement [online]. University of Washington, 8.4.2018 [cit. 2022-05-08]. Dostupné z: doi:10.48550/arXiv.1804.02767

[36] BOCHKOVSKIY, Alexey, Chien-Yao WANG a Hong-Yuan Mark LIAO. YOLOv4: Optimal Speed and Accuracy of Object Detection [online]. 23.4.2020 [cit. 2022-05-08]. Dostupné z: doi:10.48550/arXiv.2004.10934

[37] About. OpenCV [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://opencv.org/about/

[38] TAHERI, Sajjad, Illie VALENTIN a Dan KAPLUN. OpenCV.js. Github [online]. 2016 [cit. 2022-05-08]. Dostupné z: https://github.com/ucisysarch/opencvjs

[39] EmguCV [online]. [cit. 2022-05-08]. Dostupné z: https://www.emgu.com/wiki/index.php/Main_Page

[40] Introduction. OpenCV [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/4.x/d1/dfb/intro.html

[41] Drawing Functions in OpenCV. OpenCV [online]. [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html

[42] Cv::VideoCapture Class Reference. OpenCV [online]. [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html

[43] Smoothing Images. OpenCV [online]. [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

[44] Morphological Transformations. OpenCV [online]. [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[45] Image Thresholding. OpenCV [online]. [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[46] Canny Edge Detection. OpenCV [online]. [cit. 2022-05-08]. Dostupné z: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

[47] Deep Learning with OpenCV DNN Module: A Definitive Guide. LearnOpenCV [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://learnopencv.com/deep-learning-with-opencvs-dnn-module-a-definitive-guide/

[48] DEMAAGD, Kurt, Anthony OLIVER a Nathan OOSTENDORP, 2012. Practical Computer Vision with SimpleCV. O'Reilly Media. ISBN 9781449343842.

[49] SimpleCV [online]. [cit. 2022-05-08]. Dostupné z: http://simplecv.org/

[50] MATLAB: Math. Graphics. Programming., c1994-2022. MathWorks [online]. [cit. 2022-05-09]. Dostupné z: https://www.mathworks.com/products/matlab.html

[51] Computer Vision Toolbox: Design and test computer vision, 3D vision, and video processing systems, c1994-2022. MathWorks [online]. [cit. 2022-05-09]. Dostupné z: https://www.mathworks.com/products/computer-vision.html

[52] BoofCV [online], 2022. [cit. 2022-05-09]. Dostupné z: http://boofcv.org/

[53] Performance: SURF. BoofCV [online]. 2.2.2012 [cit. 2022-05-09]. Dostupné z: http://boofcv.org/index.php?title=Performance:SURF

[54] Brigáda v rámci Celostátního sčítání dopravy 2020. Centrum dopravního výzkumu [online]. 2020 [cit. 2022-05-09]. Dostupné z: https://www.cdv.cz/novinky/brigada-v-ramci-celostatniho-scitani-dopravy-2020/

[55] OpenCV configuration options reference. OpenCV [online]. [cit. 2022-05-09]. Dostupné z: https://docs.opencv.org/4.x/db/d05/tutorial_config_reference.html

[56] GStreamer: Open source multimedia framework [online]. [cit. 2022-05-09]. Dostupné z: https://gstreamer.freedesktop.org/

[57] What is GStreamer?. GStreamer: Open source multimedia framework [online]. [cit. 2022-05-09]. Dostupné z: https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html?gi-language=c

[58] CUDA Zone, c2022. NVIDIA Developer [online]. [cit. 2022-05-09]. Dostupné z: https://developer.nvidia.com/cuda-zone

[59] Overview. VidGear [online]. 4.12.2021 [cit. 2022-05-09]. Dostupné z: https://abhitronix.github.io/vidgear/v0.2.5-stable/gears/camgear/overview/

[60] Youtube-dl. Github [online]. [cit. 2022-05-10]. Dostupné z: https://github.com/ytdl-org/youtube-dl

[61] Streamlink. Github [online]. [cit. 2022-05-10]. Dostupné z: https://github.com/streamlink/streamlink

[62] Live stream from youtube | Issue. Github [online]. [cit. 2022-05-10]. Dostupné z: https://github.com/abhiTronix/vidgear/issues/133#issuecomment-638263225

[63] Deep SORT. Github [online]. [cit. 2022-05-10]. Dostupné z: https://github.com/nwojke/deep_sort

[64] DWIVEDI, Priya. People Tracking using Deep Learning: Doing cool things with data!. Towards Data Science [online]. Medium, 7.2.2019 [cit. 2022-05-10]. Dostupné z: https://towardsdatascience.com/people-tracking-using-deep-learning-5c90d43774be

[65] TensorFlow [online]. [cit. 2022-05-10]. Dostupné z: https://www.tensorflow.org/

[66] Flask: Web development, one drop at a time [online], c2010. [cit. 2022-05-10]. Dostupné z: https://flask.palletsprojects.com/en/2.1.x/

[67] Plotly: The front end for ML and data science models [online], c2022. [cit. 2022-05-10]. Dostupné z: https://plotly.com/

[68] Dash overview, c2022. Plotly [online]. [cit. 2022-05-10]. Dostupné z: https://plotly.com/dash/

[69] Dash DAQ, c2021. Plotly [online]. [cit. 2022-05-10]. Dostupné z: https://dash.plotly.com/dash-daq

[70] Dash Bootstrap Components [online]. [cit. 2022-05-10]. Dostupné z: https://dash-bootstrap-components.opensource.faculty.ai/

[71] Pyodbc. Github [online]. [cit. 2022-05-10]. Dostupné z: https://github.com/mkleehammer/pyodbc

[72] Databases: SQL Server. Microsoft Docs [online]. 4.1.2022 [cit. 2022-05-10]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/databases/databases?view=sql-server-2017

[73] About pandas. Pandas [online]. [cit. 2022-05-10]. Dostupné z: https://pandas.pydata.org/about/

[74] Zlín - ulice Osvoboditelů | AVONET. Youtube [online]. 13. 3. 2019 [cit. 2022-05-16]. Dostupné z: https://youtu.be/0Ua8_c0Nphg

[75] Yolo v4, v3 and v2 for Windows and Linux. Github [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://github.com/AlexeyAB/darknet

[76] Plotly.js CSS. CodePen [online]. [cit. 2022-05-17]. Dostupné z: https://codepen.io/chriddyp/pen/bWLwgP.css

[77] Python-dotenv. Github [online]. [cit. 2022-05-17]. Dostupné z: https://github.com/theskumar/python-dotenv

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

2D	Dvourozměrný
3D	Trojrozměrný
4D	Čtyřrozměrný
AI	Artificial Intelligence
AMD	Advanced Micro Devices
API	Application Programming Interface
BGR	Blue Green Red
CSV	Comma Separated Values
CUDA	Computer Unified Device Architecture
DBMS	Database Management Systems
DIVX	Digital Video Express
DL	Deep Learning
DLSS	Deep Learning Super Sampling
DNN	Deep Neural Network
FAST	Features from Accelerated Segment Test
GPU	Graphics Processing Unit
HLS	HTTP Live Streaming
HOG	Histogram of Oriented Values
HSV	Hue Saturation Value
HTML	Hypertext Markup Language
I/O	Input / Output
IoU	Intersection over Union
IPP	Integrated Performance Primitives
JSON	JavaScript Object Notation

mAP	Mean Average Precision
MATLAB	Matrix Laboratory
MS COCO	Microsoft Common Objects in Context
NVK	Normovaná vzájemná korelace
ODBC	Open Database Connectivity
OpenCL	Open Computing Language
OpenCV	Open Computer Vision
QR	Quick Response
R-CNN	Region Based Convolutional Neural Networks
RGB	Red Green Blue
RoI	Region of Interest
RPN	Region Proposal Network
RTX	Ray Tracing Texel eXtreme
SIFT	Scale-invariant feature transform
SORT	Simple Real time Tracker
SQL	Structured Query Language
SSD	Single Shot Detector
SURF	A Simple, Universal, Robust, Fast Distribution
SVM	Support Vector Machine
VOC	Visual Object Classes
YOLO	You Only Look Once

SEZNAM OBRÁZKŮ

Obrázek 1. Porovnání predikcí vozidla [5] 15
Obrázek 2. Získaný dataset pro trénování [5] 15
Obrázek 3. Ukázka výrazného zvýšení FPS a kvality při využití DLSS [7] 16
Obrázek 4. DLSS 2.0 [7] 17
Obrázek 5. Rozdělení druhů algoritmů pro detekci objektů 19
Obrázek 6. Základní druhy Haarových příznaků [3] 20
Obrázek 7. Proces funkce kaskádového klasifikátoru využívající Haarovy příznaky [12] 22
Obrázek 8. Příklad šablony (A) a vstupního obrazu (B) [14] 23
Obrázek 9. Hodnoty NVK v závislosti na podobnosti [14] 23
Obrázek 10. Vizualizace hodnot NVK – světlé pixely značí vyšší podobnost, tmavé nižší [14] 24
Obrázek 11. Vizualizace získaného deskriptoru [18] 25
Obrázek 12. Vizualizace vektorů optického toku a následná segmentace objektů [20] 25
Obrázek 13. Segmentovaný binární obraz (a) Detekovaný objekt v původním snímku (b) [21] 26
Obrázek 14. Rozdíl mezi tradičními algoritmy a algoritmy využívající hluboké učení [23] 27
Obrázek 15. Architektura dvoufázové sítě Fast R-CNN [24] 28
Obrázek 16. Architektura jednofázové sítě YOLO [24] 28
Obrázek 17. Workflow dvoufázové sítě R-CNN [26] 29
Obrázek 18. Architektura modelu Fast R-CNN [27] 30
Obrázek 19. Architektura Faster R-CNN [28] 31
Obrázek 20. Porovnání potřebného času (s) pro detekci objektu u jednotlivých algoritmů z rodiny R-CNN [29] 31
Obrázek 21. Princip modelu YOLO [30] 32
Obrázek 22. Výsledný obrázek po použití funkcí pro kreslení [41] 37
Obrázek 23. Obrázek před a po aplikaci eroze [44] 39
Obrázek 24. Základní funkce pro prahování [45] 40
Obrázek 25. Adaptivní prahování s využitím Gaussovy funkce [45] 41
Obrázek 26. Detekované hrany v obraze [46] 41

Obrázek 27. Architektura frameworku GStreamer [57] 50
Obrázek 28. Porovnání výkonu a přesnosti YOLOv4 s ostatními modely [36] 51
Obrázek 29. Diagram vytvořené databáze 62
Obrázek 30. Ukázka získaných dat 62
Obrázek 31. Uživatelské rozhraní 64
Obrázek 32. Modal s nastavením aplikace 65
Obrázek 33. Modal pro vytvoření masky 66
Obrázek 34. Modal s vyznačenými detekčními oblastmi 66

SEZNAM ZDROJOVÝCH KÓDŮ

Zdrojový kód 1. Základní operace 36
Zdrojový kód 2. Změna barevného prostoru 36
Zdrojový kód 3. Funkce pro kreslení [41] 37
Zdrojový kód 4. Zobrazení a ukládání videa z webkamery 38
Zdrojový kód 5. Adaptivní prahování s využitím Gaussovy funkce 40
Zdrojový kód 6. Detekce hran pomocí Cannyho detektoru 41
Zdrojový kód 7. Načtení Darknet modelu a povolení hardwarové akcelerace 42
Zdrojový kód 8. Build knihovny OpenCV [55] 49
Zdrojový kód 9. Hlavní smyčka 55
Zdrojový kód 10. Třída VideoStream 56
Zdrojový kód 11. Funkce pro inicializaci modelu YOLOv4 57
Zdrojový kód 12. Funkce pro detekci vozidel v obraze 58
Zdrojový kód 13. Funkce pro inicializaci modelu Deep SORT [63] 59
Zdrojový kód 14. Část funkce pro sledování vozidel 59
Zdrojový kód 15. Počítání vozidel pomocí detekčních oblastí – získání středu vozidla 60
Zdrojový kód 16. Počítání vozidel pomocí detekčních oblastí 61
Zdrojový kód 17. Funkce pro ukládání dat na MS SQL server 63

SEZNAM PŘÍLOH

Příloha P I: Obsah CD

PŘÍLOHA P I: OBSAH CD

fulltext.pdf – textová část bakalářské práce

Složka **realtime-traffic-detection** obsahuje:

- složka **application**
  - složka **assets** – obsahuje CSS styly
  - složka **data** – obsahuje lokálně uložená data ve formátu CSV
  - složka **deep_sort** – obsahuje soubory nutné pro funkci algoritmu Deep SORT
  - složka **resources** – obsahuje konfiguraci a soubory s váhami pro detekční a sledovací modely
- soubor **.env** – obsahuje proměnné prostředí
- soubor **app.py** – hlavní soubor aplikace
- soubor **database.py** – obsahuje funkce pro práci s databází
- soubor **detection.py** – obsahuje funkci pro detekci a sledování vozidel
- soubor **layout.py** – obsahuje rozložení stránek aplikace
- soubor **polygons.json** – obsahuje vytvořené masky a detekční oblasti
- soubor **utils.py** – obsahuje pomocné funkce
- soubor **requirements.txt** – obsahuje seznam použitých knihoven
- soubor **cv2.cpython-37m-x86_64-linux-gnu.so** – build knihovny OpenCV pro python 3.7 a systém linux obsahující potřebné moduly pro spuštění aplikace
- soubor **readme.txt** – obsahuje instrukce k vytvoření databáze pro aplikaci a další informace