FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Car Counter
Student:	Richard Vacenovský
Vedoucí:	Ing. Alisa Benešová
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této práce je vytvořit systém, který na základě kamerových záznamů bude počítat projíždějící vozidla.
Analyzujte dostupné otevřené zdroje, na kterých lze počítat projíždějící vozidla. Data vhodně rozdělte na trénovací, testovací, validační.
Proveďte analýzu nástrojů pro rozpoznávání a klasifikaci předmětů. Navrhněte a aplikujte vhodné algoritmy pro řešení této úlohy.
Pomocí zvolených nástrojů vytvořte systém, který projíždějící vozidla spočítá a získané údaje uloží do databáze.
Výsledky učení volenou metodou/metodami pak prezentujte pomocí webových stránek.
Na stránkách bude uživateli umožněno nahrát video, na kterém algoritmus provede vyhodnocení a spočítá auta.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 8. dubna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Počítadlo aut

Richard Vacenovský

Department of Computer Science
Vedoucí práce: Ing. Alisa Benešová

13. května 2021

Poděkování

Rád bych poděkoval Ing. Alise Benešové a Ing. Markovi Sušickému za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce. Dále bych rád poděkoval Fakultě informačních technologií na ČVUT za poskytnutí počítačové učebny.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

Prague dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Vacenovský, Richard. *Počítadlo aut.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato bakalářská práce řeší problém sledování objektů z kamerových záznamů. Cílem je vytvořit systém, který na základě kamerových záznamů bude počítat projíždějící vozidla. Dále je cílem analyzovat dostupné otevřené zdroje, vhodně rozdělit data a provést analýzu nástrojů pro rozpoznávání a klasifikaci předmětů. Pro vyhodnocování byly použity metody YOLO a deep SORT. Výsledkem je funkční webová aplikace, která umožňuje uživateli nahrát video, kde je záběr na libovolný úsek silnice či dálnice. Výstupem aplikace je výsledný počet vozidel ve videu. Poskytnuty jsou i informace o počtu jednotlivých typů vozidel a také časová statistika (kolik vozidel projelo kterou minutu záznamu). Dataset kamerových záznamů byl poskytnut od AI CITY CHALLENGE. Hlavním přínosem práce je možnost nahrazení sčítacích komisařů. Práce splnila vše, co bylo jejím cílem, ale stále se jedná o jednoduchou základní verzi, která nabízí spoustu možností, jak ji rozšířit.

Klíčová slova počítání aut, detekce objektů, sledování objektů, YOLO, deep SORT

Abstract

This bachelor thesis solves the problem of monitoring objects from camera recordings. The aim is to create a system that will count passing vehicles based on camera recordings. Furthermore, the aim is to analyze the available open sources, appropriately divide the data and analyze the tools for recognition and classification of objects. Methods YOLO and deep SORT were used for evaluation. The result is a functional web application that allows the user to upload a video where there is a shot of any section of road or highway. The output of the application is the resulting number of vehicles in the video. Information on the number of individual types of vehicles is also provided, as well as time statistics (how many vehicles passed through which minute of recording). A camera dataset was provided by AI CITY CHALLENGE. The main benefit of the work is the possibility of replacing census commissioners. The work fulfilled everything that was its goal, but it is still a simple basic version, which offers a lot of possibilities to expand it.

Keywords car counting, object detection, object tracking, YOLO, deep SORT

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Je rok 2021 a technologický pokrok jde stále dopředu. Proto je zarážející, že se i v dnešní době můžeme setkat s takzvanými *sčítači*, kteří sedí u komunikace a ručně zaznamenávají, kolik projede vozidel. Přitom by jen stačilo na dané místo umístit kameru se systémem, který by odvedl stejnou práci. Informace o provozu by se tak daly zpracovávat prakticky nepřetržitě s mnohem menší námahou a větší přesností. Práce je určena pro kohokoliv, kdo potřebuje sbírat informace o provozu. Sčítání je možné na kterékoliv dopravní komunikaci, jako je např. silnice nebo dálnice. Teoretická část práce je zaměřena hlavně na popis použitých technologií a strukturu systému. Praktická část se zabývá nejprve použitím metod pro detekci a sledování objektů. Následně jsou výsledky ukládány do databáze a prezentovány ve webové aplikaci. Práce navazuje například na diplomovou práci studenta Jiřího Groha, který se ve své práci zabýval problematikou hledání volných parkovacích míst a používal při tom obdobné technologie.

Kapitola 2

Cíl práce

Hlavním cílem práce je vytvořit systém s přehledným a srozumitelným webovým rozhraním, který bude umět spočítat projíždějící vozidla na základě záznamu z kamery. Systém bude schopen vyhodnotit záběr z kamery, která je namířená na libovolnou dopravní komunikaci (silnice, dálnice). Dále je cílem analyzovat dostupné otevřené zdroje, na kterých lze počítat projíždějící vozidla a vhodně data rozdělit.

V teoretické části budou rozebrány technologie a metody použité v praktické části práce. Bude se jednat mimo jiné o průzkum dostupných otevřených zdrojů, ale hlavně o analýzu nástrojů pro rozpoznání, klasifikaci, detekci a sledování (anglicky tracking) objektů. Dále bude představeno i webové rozhraní a databázový systém.

Praktická část se bude zabývat použitím dostupných metod pro rozpoznání a sledování objektů (konkrétně vozidel) a jejich využití pro sčítání projíždějících vozidel na kamerovém záznamu. Získané údaje budou po vyhodnocení uloženy do databáze. V poslední fázi praktické části bude vytvořeno webové rozhraní. Hlavní funkcí webového rozhraní bude umožnit uživateli jak nahrát libovolné video, tak po zpracování zobrazit výsledky.

Kapitola 3

Analýza

3.1 Otevřené zdroje

Otevřené zdroje jsou *„volně přístupné, otevřeně licencované dokumenty a média, která jsou využívána pro výuku, učení a hodnocení, jakož i pro výzkumné účely“*.[19] V případě této práce je úkolem nalézt dataset, na kterém lze počítat projíždějící vozidla.

3.1.1 Záznamy z českých komunikací

Původní záměr této práce byl pracovat se záznamy z dopravních kamer, které byly pořízeny na území České republiky. Žádné takové záběry však nejsou volně přístupné.

3.1.2 Data z AI CITY CHALLENGE

Testovací dataset pro tuto práci poskytla **AI CITY CHALLENGE 2021**. Data pro tuto výzvu pocházejí z několika dopravních kamer z města ve Spojených státech amerických i ze státních dálnic v Iowě. Více informací o **AI CITY CHALLENGE 2021** se nachází na adrese: `https://www.aicitychallenge.org`. Informace týkající se licence jsou přístupné na adrese: `http://www.aicitychallenge.org/wp-content/uploads/2021/01/DataLicenseAgreement_AICityChallenge_2021.pdf`.

3.2 Struktura a fungování aplikace

Celá aplikace nejprve načte video od uživatele pomocí webové aplikace. Na dané video používá model pro detekci objektů (YOLOv4), který video zpracovává a předává je modelu pro sledování objektů (Deep SORT). Rozpoznaná vozidla jsou poté sčítána metodou, která je popsána v části **Im-**

plementace této práce. Výsledek je následně uložen do databáze a pomocí webového rozhraní zobrazen uživateli.

3.3 Definice pojmů

3.3.1 Klasifikace obrazu

Klasifikace obrazu určuje třídu jednoho objektu v obraze. Na rozdíl od lokalizace nebo detekce klasifikace neurčuje, kde se objekt v obraze nachází. Vstupem je tedy nějaký obraz s jedním objektem (např. fotografie) a výstupem je třída objektu, který se na fotografii nachází.[4]

3.3.2 Lokalizace objektů

Lokalizace objektů určuje polohu jednoho nebo více objektů v obraze. Objekty jsou vyhledány a ohraničeny obdélníkovým rámečkem. Vstupem je obraz s jedním nebo více objekty a výstupem je obraz s jedním nebo více ohraničujícími rámečky.[4]

3.3.3 Detekce objektů

Úloha detekce objektů spojuje dohromady obě již zmíněné disciplíny. Jedná se o kombinaci klasifikace a lokalizace. Na vstupu je obraz s jedním nebo více objekty a výstupem je obraz s jedním nebo více ohraničujícími rámečky s přidaným popisem příslušné třídy.[4]

3.3.4 Sledování objektů

Sledování objektů má za úkol pozorovat je a na základě informací, získaných ze snímků předchozích předpovídat, kde se budou ve snímku nacházet. Sledování objektů lze rozdělit na dvě kategorie:

- sledování jednoho objektu (SOT)
- sledování více objektů (MOT)

V této práci bude použita druhá varianta, tedy sledování více objektů.[25]

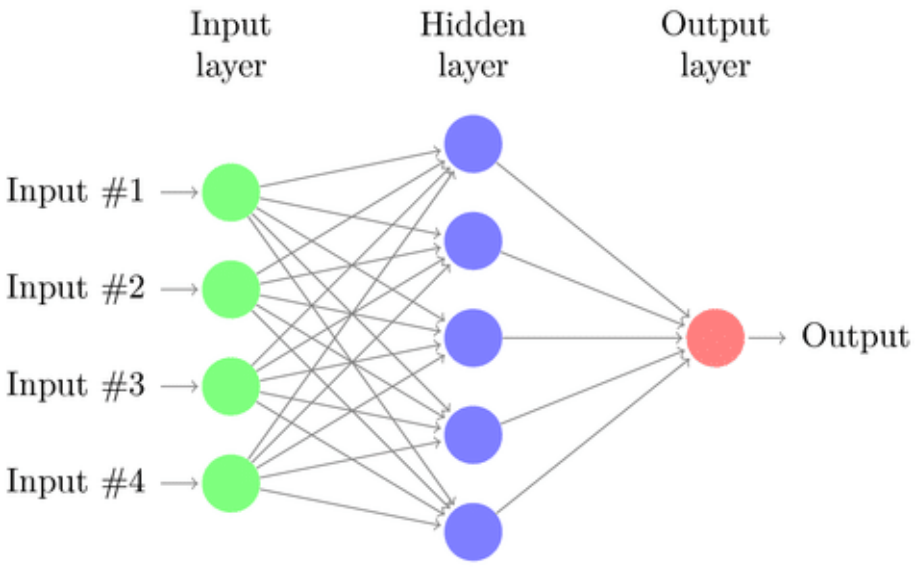
3.3.4.1 Multiple Object Tracking (MOT)

Multiple Object Tracking, jak již název napovídá, má za úkol identifikovat a sledovat více objektů ve videu. Objekty mohou patřit do jedné nebo více tříd, jako jsou například lidé, automobily nebo dopravní značky.[25]

3.3.5 Neuronové sítě

Neuronová síť (dále jen „NN“) se skládá z jednotlivých jednoduchých neuronů, kde jeden neuron je jednotka, která *„zpracovává váhované vstupní*

signály a generuje výstup“.[12] Celá NN je složena z několika vrstev. První vrstva se nazývá vstupní a každý neuron zde přijímá pouze jeden vstup. Následuje vrstva skrytá, ve které mohou mít neurony více vstupních i výstup--ních parametrů a zpravidla jsou mezi sebou neurony v sousedních vrstvách plně propojeny (je propojen každý neuron s každým). Na konci je vrstva výstupní, kde mají neurony pouze jeden výstup. NN se umí učit a přizpůsobovat vstupním datům pomocí změny vnitřních parametrů.[30]



Obrázek 3.1: Neuronová síť[5]

3.3.5.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě (dále již jen CNN) se využívají především na zpracování zvuku nebo obrazu. V případě této práce se jedná o obraz. CNN mají na rozdíl od klasických NN navíc konvoluční část. Ta se stará o zpracování obrazu/zvuku, který transformuje do vektoru, který následně putuje jako vstup do NN.[16]

3.4 Metody detekce objektů

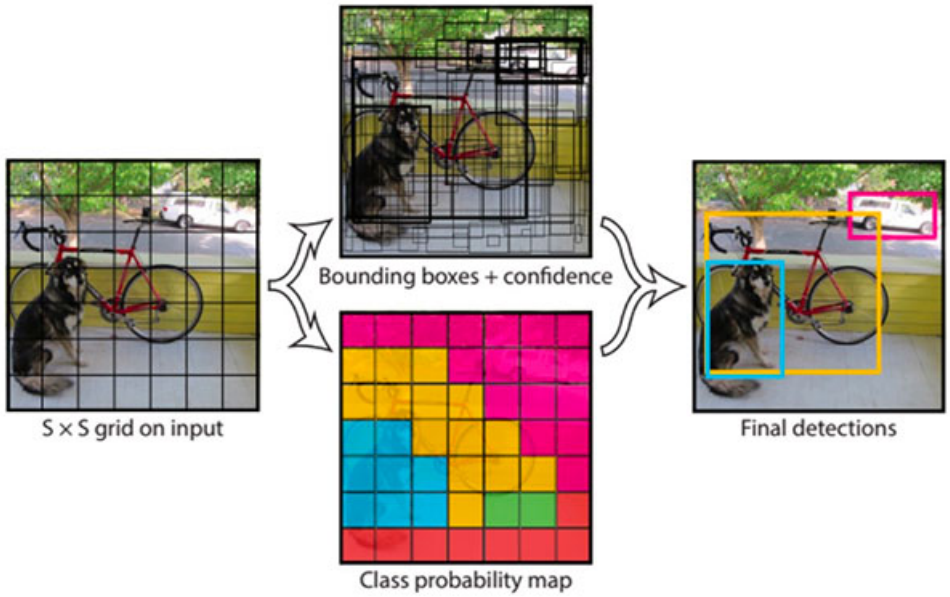
3.4.1 Jednostupňové detektory

Jednostupňové detektory řeší detekci jako *„jednoduchý regresní problém*“[20]. V jednom kroku určí souřadnice ohraničujícího rámečku a pravděpodobnost, že se jedná o konkrétní třídu. Tato metoda je výrazně rychlejší než dvoustupňová detekce, ale její nevýhodou je, že nedosahuje takové přesnosti. Používá

se hlavně v případech, kdy je potřeba rychlé vyhodnocování, například v živých vysíláních. Mezi nejznámější jednostupňové detektory patří například YOLO (You Only Look Once) a SSD (Single Shot MultiBox Detector).[20][28]

3.4.1.1 YOLO

You Only Look Once (YOLO) je jedna z nejpoužívanějších jednostupňových metod detekce objektů: *„Detekce objektů je přetvářena jako jediný regresní problém, přímo od obrazových pixelů po souřadnice ohraničujícího rámečku a pravděpodobnosti třídy. Jak je vidět na obrázku níže, YOLO je v podstatě jednoduché. Jediná konvoluční síť současně předpovídá více ohraničujících rámečků a pravděpodobnosti tříd pro tyto rámečky. YOLO je trénované na celých obrazech a přímo optimalizuje výkon detekce. Tento jednotný model má několik výhod oproti tradičním metodám detekce objektů. Jednou z výhod je, že je YOLO extrémně rychlé. Protože je detekce řešena jako regresní problém, nejsou tedy potřeba ‚složité série datových prvků' (volný překlad originálu ‚complex pipeline'). Jednoduše se spustí neuronová síť na novém obrázku v testovacím čase, aby předpovídala detekce. Základní síť běží rychlostí 45 snímků za sekundu bez dávkového zpracování na Titan X GPU a rychlá verze běží rychlostí více než 150 fps. To znamená, že je možné zpracovávat streamované video v reálném čase s latencí (prodlevou) méně než 25 milisekund. YOLO navíc dosahuje více než dvojnásobku průměrné přesnosti, než jiné systémy, které fungují v reálném čase.“*[22]



Obrázek 3.2: YOLO architektura[23]

Další výhodou této metody je vyhodnocování celého obrazu najednou. Jiné metody používají například metodu posuvného okna, kde se informace z obrazu sbírají postupně podle aktuální polohy posuvného okna, což je značně pomalejší.

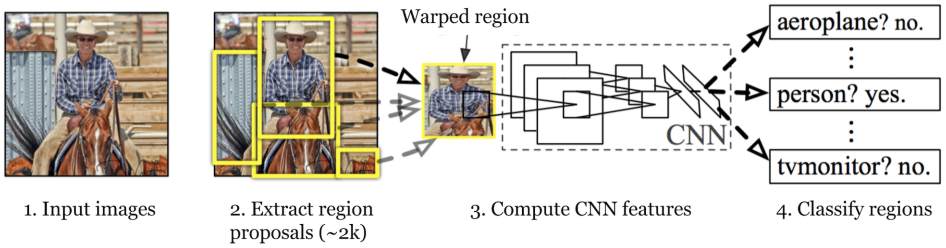
3.4.2 Dvoustupňové detektory

U dvoustupňových detektorů probíhá vyhodnocování ve dvou fázích. Nejprve jsou určeny oblasti, ve kterých bude vyhodnocování probíhat, a ve druhém kroku dochází ke klasifikaci pouze těch oblastí, které byly vybrány v prvním kroku. Tento postup je sice výrazně pomalejší než jednostupňová detekce, ale disponuje znatelně větší přesností. Nachází tak využití v případech, kdy není vyžadována rychlost, ale spíše přesnost. Mezi nejznámější dvoustupňové detektory patří například Faster R-CNN (Region-based Convolutional Neural Networks) nebo Mask R-CNN.[20][28]

3.4.2.1 R-CNN

Region-based Convolutional Neural Networks neboli R-CNN vychází z naivního řešení problému detekce objektů, kde jsou vybrány různé oblasti zájmu a ty vyhodnoceny pomocí CNN. Kvůli prostorovým umístěním objektů by ale těchto regionů mohlo být obrovské množství a problém by se tak stal výpočetně příliš náročným. Ross Girshick tento problém vyřešil pomocí selektivního vyhledávání, které z obrazu vybere pouze 2 000 regionů. Popis algoritmu selektivního vyhledávání lze popsat následovně:

1. *„Vytvoření počáteční subsegmentace, vygenerování mnoho kandidátských oblastí.“*
2. *„Pomocí hladového algoritmu rekurzivní spojování podobných oblastí do větších.“*
3. *„Použití vygenerovaných regionů k vytvoření konečných návrhů kandidátských regionů.“*[7]



Obrázek 3.3: R-CNN architektura[7]

Těchto 2 000 návrhů je poté zpracováno pomocí CNN a tím vznikají předpovědi přítomnosti objektu v návrzích oblastí. Metoda R-CNN má ale několik nevýhod:

1. *„Trénování sítě stále trvá obrovské množství času, protože by bylo potřeba klasifikovat 2 000 návrhů regionů na obrázek.“*

2. *„Nelze jej implementovat v reálném čase, protože u každého testovacího obrazu trvá přibližně 47 sekund.“*

3. *„Algoritmus selektivního vyhledávání je pevný algoritmus. V této fázi se tedy neděje žádné učení. To by mohlo vést k vytvoření návrhů špatných regionů.“*[7]

S řešením těchto problémů poté přicházejí optimalizované metody, jakými jsou Fast R-CNN a Faster R-CNN. Těm se tato práce ale již věnovat nebude. Jak už je zmíněno výše v kapitole **Jednostupňové detektory**, pro řešení problému sledování vozidel se více hodí detektory s jednostupňovou architekturou. Hlavním důvodem je rychlost vyhodnocení.

3.5 Metody sledování objektů

3.5.1 Tradiční metody

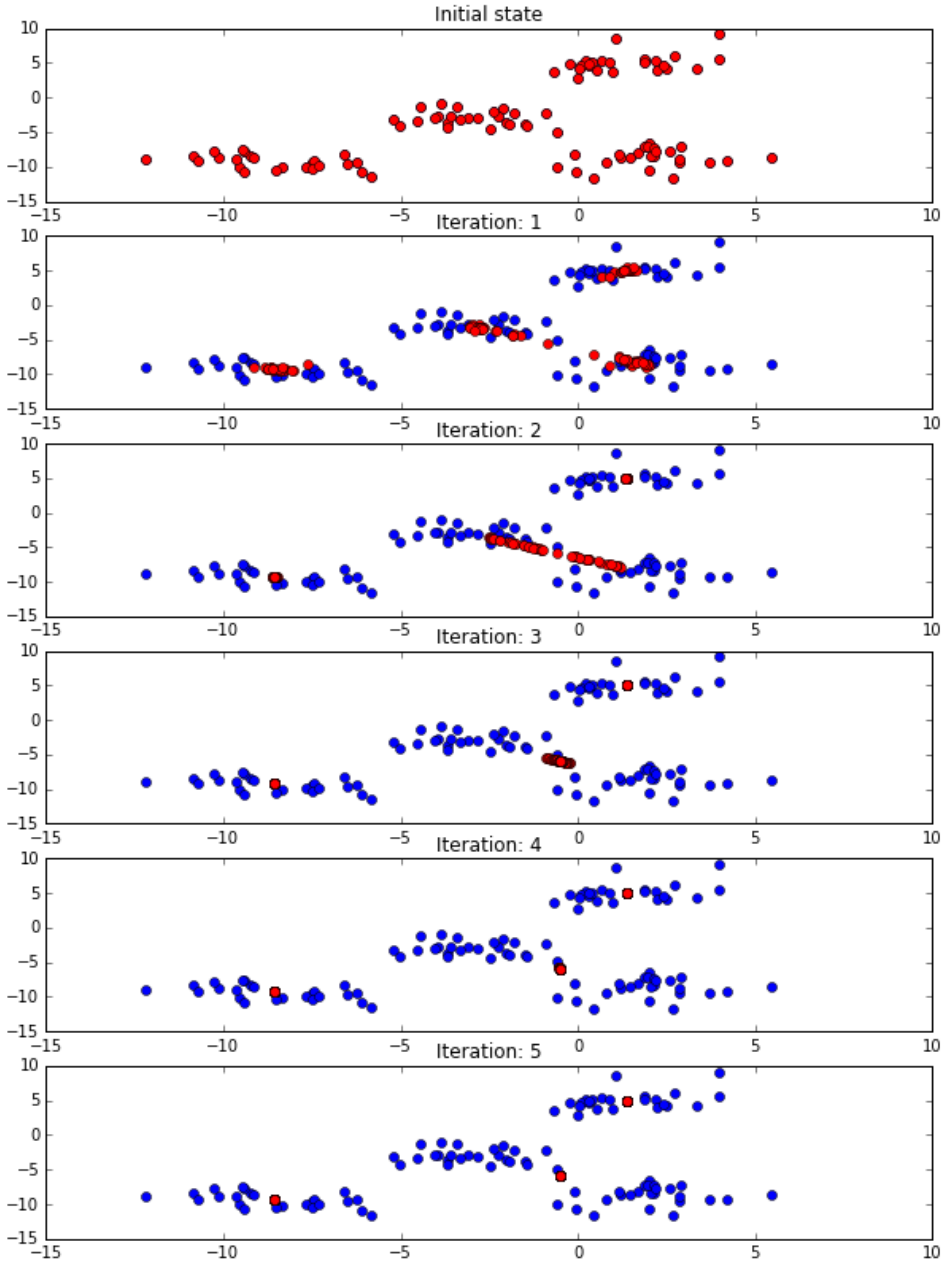
3.5.1.1 Sledování pomocí detekce objektů

V předchozí kapitole již byla rozebrána tematika detektorů objektů a jejich využití například i pro sledování objektů. V případě sledování objektů se ale pracuje s více snímky, které na sebe navazují. Detektory objektů umí celkem spolehlivě určit třídu daného objektu v jednom konkrétním obraze, ale neuchovávají si informace o objektech z předchozích snímků. Rozpoznají tedy, že se na dvou po sobě jdoucích snímcích jedná například o automobil, ale již neurčí, že se jedná o stejný automobil. V případě sledování jediného objektu by nešlo o tak velký problém, ale pro účely této práce je zapotřebí použít sofistikovanější metodu.

3.5.1.2 Mean-shift

Mean-shift je algoritmus, který funguje na bázi shlukování. Shlukování probíhá postupným přiřazováním datových objektů k těžišti clusteru. Ten je určen tím, kde je nejvíce bodů v jeho okolí. Algoritmus se zastaví ve chvíli, kdy má každý datový bod přiřazen svůj cluster. Svými vlastnostmi je Mean-shift velice podobný algoritmu K-means s tím rozdílem, že nevyžaduje předem počet shluků. V praxi to tedy znamená, že ve snímku je identifikován objekt a v dalším snímku se algoritmus snaží najít objekt, který nejvíce

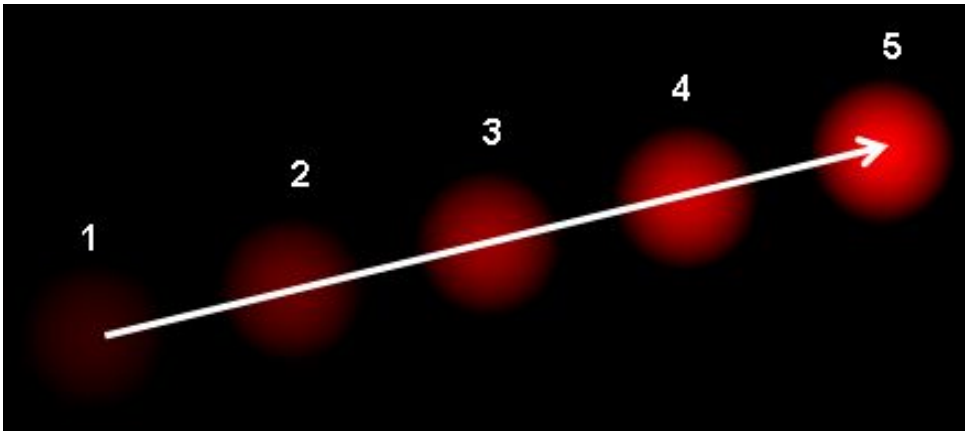
odpovídá tomu z předchozího snímku. Vyhledávání probíhá v blízkém okolí podle toho, kde se objekt nacházel v předchozím snímku.[15]



Obrázek 3.4: Mean-shift průběh algoritmu[15]

3.5.1.3 Optical flow

„Optical flow je vzor zdánlivého pohybu obrazových objektů mezi dvěma po sobě jdoucími snímky způsobený pohybem objektu nebo kamery. Jedná se o 2D vektorové pole, kde každý vektor je vektor posunutí ukazující pohyb bodů od prvního snímku k druhému. Algoritmus tedy umí pomocí těchto vektorů sledovat a dokonce předpovídat trajektorii objektu.



Obrázek 3.5: Optical flow příklad[18]

Na obrázku se nachází kulatý objekt v pěti navazujících snímcích. Šipka ukazuje vektor pohybu.

Je uvažován pixel $\boldsymbol{I(x,y,t)}$*, kde* $\boldsymbol{x}$ *a* $\boldsymbol{y}$ *jsou souřadnice a* $\boldsymbol{t}$ *časový údaj. Pohybuje se o vzdálenost* $\boldsymbol{(dx, dy)}$ *v dalším snímku po* $\boldsymbol{dt}$ *čase. Takže dokud jsou pixely stále stejné a intenzita se nemění, platí:* $\boldsymbol{I(x,y,t)=I(x+dx,y+dy, t+dt)}$*. Po aproximaci Taylorovy řady na pravé straně, odstranění běžných výrazů a vydělením* $\boldsymbol{dt}$*, vzniká následující rovnice:*

$$f_x u + f_y v + f_t = 0$$

kde:

$$f_x = \frac{\partial f}{\partial x} \; ; \; f_y = \frac{\partial f}{\partial y}$$

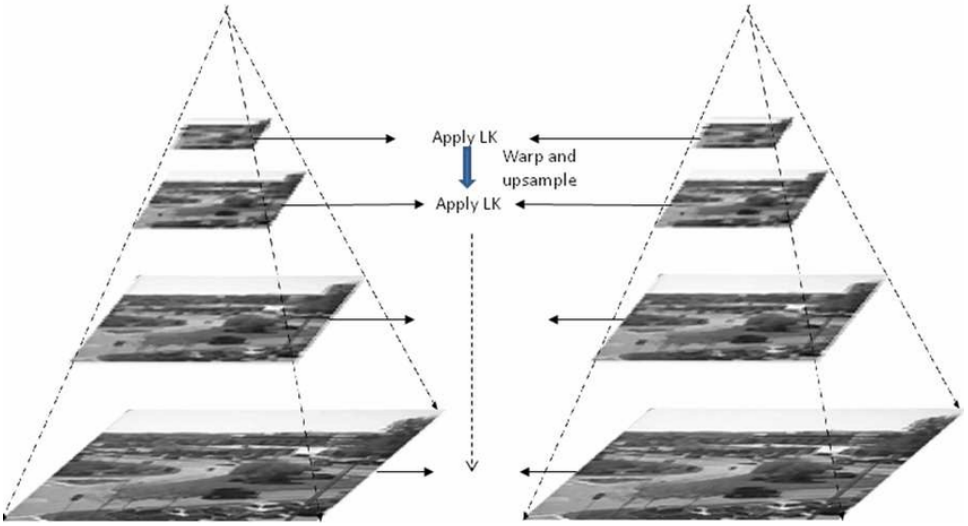
$$u = \frac{dx}{dt} \; ; \; v = \frac{dy}{dt}$$

Výše uvedená rovnice se nazývá rovnice optického toku. V něm můžeme najít f_x *a* f_y*, jsou to obrazové přechody. Podobně* f_t *je gradient v čase. Ale*

(u, v) *není známo. Tuto jednu rovnici nemůžeme vyřešit dvěma neznámými proměnnými. K řešení tohoto problému je poskytnuto několik metod a jednou z nich je Lucas-Kanade.*“[17]

3.5.1.4 Lucas Kanade

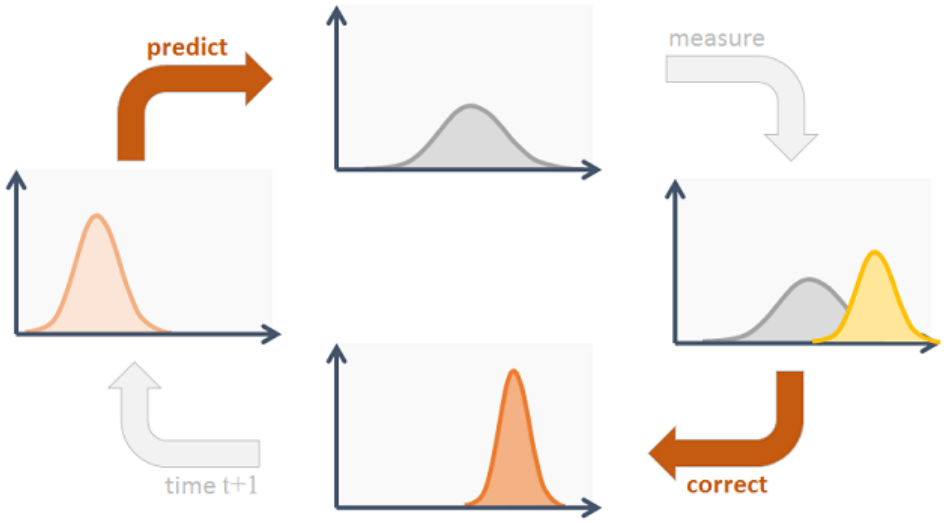
Algoritmus Lucas Kanade dostává body určené ke sledování a vrací vektor ve formátu Optical flow. Nastává zde ale problém, protože algoritmus se umí vypořádat pouze s malými pohyby. V případě většího pohybu dochází k selhání. K řešení tohoto problému se používá takzvaná pyramida. Čím výše se v pyramidě nacházíme, tak se z velkých pohybů stávají malé a malé pohyby mizejí úplně.[17]



Obrázek 3.6: Lucas Kanade pyramida[24]

3.5.1.5 Kalman Filter

Kalman Filter je algoritmus, který se snaží využít dostupných detekcí a předchozích predikcí k předpovědi, kde se bude objekt v následujícím snímku nacházet. Zároveň však uvažuje chyby, které v tomto procesu mohou nastat. V reálném světě se většinou objekty nepohybují konstantní rychlostí a v této práci tomu není jinak. Vozidla mění rychlost i v rámci krátkého snímku. Žádný detektor navíc není bezchybný. Proto přichází na řadu již zmiňované uvažování chyb. Tyto chyby zde budou nazývány jako takzvané složky hluku. První složkou je *procesní šum*, který zahrnuje fakt, že se vozidla nebudou nikdy pohybovat konstantní rychlostí. Druhou složkou je *šum měření*, zohledňuje, že žádný existující detektor není dokonalý.[14]



Obrázek 3.7: Kalman Filter[10]

3.5.1.6 Maďarský algoritmus

Maďarský algoritmus neboli Maďarská metoda je kombinatorický optimalizační algoritmus, který řeší problém s přiřazením v polynomiálním čase. Jako maďarský byl pojmenován kvůli dvěma významným matematikům (Dénes Kőnig a Jenő Egerváry) původem právě z této země.

3.5.1.7 SORT (Simple Online and Realtime Tracker)

SORT je podle[11] možné rozdělit na 4 komponenty, které společně tvoří výsledné sledování. Jsou to:

- detekce
- odhad
- asociace
- vytvoření a destrukce identity

3.5.1.7.1 Detekce

První komponentou metody SORT je **detekce**. Není možné ji využít přímo pro sledování objektů, ale její kvalita má značný vliv na přesnost sledování metody SORT.[11]

3.5.1.7.2 Odhad

Druhou komponentou je **odhad** polohy objektu v následujícím snímku. Odhad je proveden na základě modelu lineární konstantní rychlosti. V tomto bodě se zapojuje již zmíněný Kalman filter. Pokud detektor najde v následujícím snímku odpovídající objekt, dochází pouze k aktualizaci stavu cíle a složky rychlosti jsou vyřešeny pomocí Kalmanova filtru. Pokud detektor odpovídající objekt nenachází, dochází pouze k predikci bez další opravy pomocí modelu lineární rychlosti.[11]

3.5.1.7.3 Cílová asociace

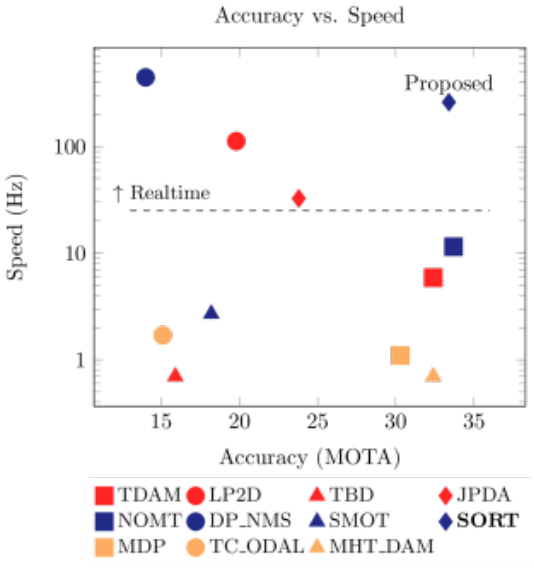
„*Při přiřazování detekcí stávajícím cílům se geometrie ohraničujícího rámečku každého cíle odhaduje předpovědí jeho nového umístění v nejnovějším snímku.* ***Assignment cost matrix*** *se vypočítá jako vzdálenost* ***intersection-over-union (IOU)*** *mezi každou detekcí a všemi předpokláda-nými ohraničujícími rámečky ze stávajících cílů.*

Problém je řešen optimálně pomocí maďarského algoritmu. To funguje obzvláště dobře, když jeden cíl zakrývá jiný.“[11]

3.5.1.7.4 Životní cyklus sledované identity

Poslední komponentou je **životní cyklus sledované identity**. Při zaznamenání nového objektu v obraze je potřeba vytvořit novou identitu, stejně tak je při opuštění obrazu příslušnou identitu potřeba zničit. Nový objekt ve snímku, který ještě není sledován, je rozpoznán ve chvíli, kdy je překrytí menší než minimální IOU. V tuto chvíli dochází k inicializaci, kde je rychlost objektu nastavena na nulu a složka nejistoty na vysokou hodnotu. Pro zabránění sledování špatně vyhodnocených objektů ještě probíhá tzv. *zkušební doba*, kdy se sbírají důkazy o validnosti tohoto sledování.

Konec sledování identity a její destrukce nastává, pokud detektor nenalezl objekt v n po sobě jdoucích snímcích. Hodnota n může být nastavena uživatelem. Pokud se identita objeví po její destrukci, vystupuje poté již jako nová identita.[11]



Obrázek 3.8: Simple Online and Realtime Tracker[3]

3.5.2 Přístupy založené na hlubokém učení

3.5.2.1 Sítě hlubokého učení (Deep regression networks)

„Sítě hlubokého učení jsou jednou z prvních metod, která využívala hluboké učení pro sledování jednoho objektu. Model je trénován na datové sadě skládající se z videí s označenými cílovými snímky. Cílem modelu je jednoduše sledovat daný objekt v dané části obrazu.

K dosažení tohoto cíle používají dvourámcovou architekturu CNN, která k přesné regresi k objektu používá aktuální i předchozí snímek.

Vezmeme oříznutí z předchozího snímku na základě předpovědi a definujeme ‚oblast hledání' v aktuálním rámci na základě dané části obrazu. Nyní je síť trénována na regresi pro objekt v této oblasti hledání. Síťová architektura je jednoduchá se CNN, následována plně spojenými vrstvami, které nám přímo dávají souřadnice ohraničujícího rámečku."[14]

3.5.2.2 ROLO - Recurrent YOLO

ROLO neboli Recurrent YOLO je, jak již název napovídá, upravenou verzí původní metody YOLO, která je popsána v sekci *Metody detekce objektů* této práce. Od metody YOLO se liší použitím LSTM (Long short-term memory) jednotky, která se *„od standardních dopředných neuronových sítí liší*

svým zpětnovazební připojením, díky kterému může zpracovávat nejen jednotlivé datové body, ale také celé sekvence dat“.[13]

„*Architektura je poměrně jednoduchá. Detekce z YOLO jsou zřetězeny feature vektorem založeny na CNN feature extractoru (Můžeme buď znovu použít back-end YOLO, nebo použít speciální extraktor funkcí). Nyní je tento zřetězený feauture vektor, který představuje většinu prostorových informací souvisejících s aktuálním objektem, spolu s informacemi o předchozím stavu předán do buňky LSTM.*

Výstup buňky nyní odpovídá prostorovým i časovým informacím. Tento jednoduchý trik používání CNN pro extrakci funkcí a LSTM pro předpo--vědi ohraničujících polí poskytl vysoké vylepšení při sledování objektů.“[14]

3.5.2.3 Deep SORT (Simple Online and Realtime Tracker)

Deep SORT je rozšířením již zmiňovaného SORTu a jedná se o jeden z *nejpopulárnějších a nejpoužívanějších frameworků pro sledování objektů.* Následně budou rozebrány jednotlivé složky tohoto frameworku.[14]

3.5.2.3.1 Kalman filter

Kalmanův filtr byl již popsán výše v této práci v souvislosti s metodou SORT v části **Metody sledování objektů**. V tomto případě Deep SORT pracuje s 8 proměnnými; „***(u, v, a, h, u, v, a, h)***, *kde* ***(u, v)*** *jsou středy ohraničujících rámečků,* ***a*** *je poměr stran a* ***h*** *výška obrazu. Ostatní proměnné jsou příslušné rychlosti proměnných.*“[14]

Protože Deep SORT pracuje pouze s jednoduchým modelem lineární rychlosti, mají proměnné pouze absolutní faktory rychlosti a polohy. Jak již bylo dříve zmíněno, pomáhá Kalmanův filtr eliminovat šum ve snímku a využívá detekci z předchozího snímku pro predikci, kde se bude objekt nacházet ve snímku následujícím. Na začátku bylo zmíněno 8 proměnných. Ty jsou uloženy a aktualizovány pro každou detekci a pomáhají tak sledovat daný objekt. Každý objekt uchovává také údaj o tom, kdy byl naposledy detekován, a v případě, že překročí určitý počet snímků, kdy se v obraze nenacházel, je jeho sledování ukončeno.[14]

„Rovněž pro eliminaci duplicitních stop existuje minimální prahová hodnota pro detekci prvních několika snímků.“[14]

3.5.2.3.2 Problém přiřazení

Nové detekce i nové předpovědi, které přicházejí z Kalmanova filtru, jsou zpracovávány nezávisle na sobě, což může být problém. Není totiž známo, jak tuto detekci s předpovědí spojit. Pro vyřešení tohoto problému je potřeba:

- metrika vzdálenosti
- algoritmus pro asociaci dat

3.5.2.3.3 Metrika vzdálenosti

„Pro metriku vzdálenosti je použita čtvercová Mahalanobisova vzdále--nost k začlenění nejistot z Kalmanova filtru. Stanovení prahové hodnoty této vzdálenosti nám může poskytnout velmi dobrou představu o skutečných asociacích. Tato metrika je přesnější než např. euklidovská vzdálenost, protože efektivně měříme vzdálenost mezi 2 distribucemi. Na jednotlivé distribuce je aplikován Kalman filtr.“[14]

3.5.2.3.4 Algoritmus pro asociaci dat

Pro asociaci dat je použit standardní maďarský algoritmus, protože je velmi efektivní a jednoduchý. Jeho popis je možné nalézt výše v této práci v kapitole **Metody sledování objektů**.[14]

3.5.2.3.5 Hluboké učení

Mohlo by se zdát, že hluboké učení již nebude potřeba, protože všechny problémy spojené se sledováním objektů byly vyřešeny. V reálném světě ale mohou nastat další problémy, jako jsou např. různé úhly pohledu, se kterými si bez hlubokého učení nelze poradit. Proto byla představena další metrika vzdálenosti, která je založena na *vzhledu* objektu.[14]

3.5.2.3.6 Vektor vzhledu (The appearance feature vector)

Nyní je tedy potřeba vytvořit vektor, který bude schopný popsat všechny vlastnosti daného obrazu. Nejprve vznikne klasifikátor, který bude trénován na datech do bodu, kdy dosáhne dostatečné přesnosti. Za předpokladu, že má klasifikátor klasickou architekturu, tak po odstranění finální klasifikační vrstvy zbude hustá vrstva, která vytváří jediný vektor funkcí.

*„Tento rysový vektor se stává naším **deskriptorem vzhledu** objektu. Po natrénování stačí předat všechny části detekovaného ohraničujícího rámečku*

z obrázku do této sítě a získat vektor dimenzionálního prvku ***128 X 1****. Aktualizovanou metriku vzdálenosti lze popsat vzorcem níže:*

$$D = Lambda * D_k + (1 - Lambda) * D_a$$

Kde D_k je vzdálenost Mahalanobis a D_a odpovídá kosinově vzdálenosti mezi vektory prvků vzhledu a Lambda je váhový faktor.

Význam D_a je tak vysoký, že autoři tvrdí, že dokázali dosáhnout sou-časného stavu techniky i pomocí: ***Lambda = 0*** *, tj. pouze za použití D_a. Jednoduchá metrika vzdálenosti v kombinaci s výkonnou technikou hlubokého učení je vše, co je zapotřebí k tomu, aby deep SORT byl elegantním a jedním z nejrozšířenějších sledovačů objektů.*“[14]

3.6 Použité technologie

3.6.1 Python

„Python je vysokoúrovňový skriptovací programovací jazyk, který v roce 1991 navrhl Guido van Rossum. Nabízí dynamickou kontrolu datových typů a podporuje různá programovací paradigmata, včetně objektově orientovaného, imperativního, procedurálního nebo funkcionálního. V roce 2018 vzrostla jeho popularita a zařadil se mezi nejoblíbenější jazyky.“[21]

Python je přístupný pro většinu platforem včetně Unixu, MS Windows, Mac OS a Android. V případě Unixu je až na výjimky dokonce součástí základní instalace. Je určen pro vývoj rozsáhlých aplikací, a to včetně grafického rozhraní. Jedná se o jazyk hybridní, což znamená, že je v něm současně možné využívat objektově orientované, procedurální i funkcionální paradigma. Jeho největší výhodou oproti ostatním programovacím jazykům je především jeho jednoduchost. Umožňuje uživateli napsat funkční program, který je výrazně kratší a díky tomu i přehlednější než v případě ostatních jazyků, například C++.[21]

3.6.2 Knihovny TensorFlow a PyTorch

V obou případech se jedná o open source knihovny, které se zaměřují hlavně na číselné výpočty dat. Obě knihovny jsou rovněž používány jak v akademických kruzích, tak i pro komerční účely.[9]

3.6.2.1 TensorFlow

TensorFlow je platforma, která zjednodušuje vývoj a práci s modely strojového učení. Přes svoje široké zaměření se nejvíce soustřeďuje na trénování

hlubokých neuronových sítí. Platforma byla původně vytvořena týmem Google Brain pro společnost Google v roce 2015.

TensorFlow využívá rozhraní Keras API, které umožňuje jednoduchý začátek práce se strojovým učením. Jednou z velkých výhod je možnost snadno trénovat modely bez ohledu na použitý jazyk či platformu. Je možné využít i různé moduly TensorFlow, například TensorFlow Lite pro mobilní zařízení, TensorFlow.js pro prostředí JavaScriptu nebo TensorFlow Extended pro plné využití všech funkcionalit. *„TensorFlow také podporuje ekosystém výkonných doplňkových knihoven a modelů, se kterými můžete experimentovat, včetně Ragged Tensors, TensorFlow Probability, Tensor2Tensor a BERT.“*[27] Platforma TensorFlow je dokonce využívána nejrůznějšími velkým firmami, mezi které se řadí již zmiňovaný Google a dále například CocaCola, Airbnb, Intel nebo sociální síť Twitter.[29]

3.6.2.2 PyTorch

PyTorch byl vyvinut společností Facebook v roce 2016. První veřejné vydání proběhlo tedy rok po vzniku první verze TensorFlow. Hlavním účelem bylo poskytnout podobné funkce, jako poskytuje TensorFlow. Zároveň se PyTorch snažil, aby práce s ním byla pro uživatele co nejjednodušší a tím i příjemnější. Tento přístup zafungoval a v roce 2016 popularita knihovny TensorFlow silně upadla. Práce s ní byla příliš složitá a PyTorch byl mnohem více *user friendly*. Programátoři, kteří pracovali s Pythonem, shledali práci s knihovnou PyTorch jako vyhovující a začali ji ve velké míře používat. To způsobilo, že naopak další verze TensorFlow kopírovala mnoho funkcionalit, kterými PyTorch disponoval. Po vydání verze TensorFlow 2.0 tak popularita knihovny opět vzrostla.[9]

3.6.3 OpenCV

OpenCV (Open Source Computer Vision Library) je open source knihovna se zaměřením na počítačové vidění a strojové učení. Hlavním cílem knihovny bylo vytvořit společnou infrastrukturu pro aplikace počítačového vidění a urychlit práci s technologiemi v komerční sféře.

„Knihovna má více než 2 500 optimalizovaných algoritmů, které zahrnují ucelenou sadu klasických i nejmodernějších algoritmů počítačového vidění a strojového učení. Tyto algoritmy lze použít k detekci a rozpoznávání tváří, identifikaci objektů, klasifikaci lidské činnosti ve videích, sledování pohybů kamer, sledování pohybujících se objektů. OpenCV má více než 47 tisíc uživatelů a odhadovaný počet stažení přesahuje 18 milionů. Knihovna je hojně využívána ve společnostech (Google, Yahoo, Microsoft, Intel), výzkumných sku-

pinách a vládními orgány.

Má rozhraní C ++, Python, Java a MATLAB a podporuje Windows, Linux, Android a Mac OS. OpenCV se opírá většinou o aplikace vidění v reálném čase a využívá instrukce MMX a SSE, jsou-li k dispozici. Právě teď se aktivně vyvíjí plně vybavená rozhraní CUDA a OpenCL. Existuje více než 500 algoritmů a asi 10krát tolik funkcí, které tyto algoritmy skládají nebo podporují. OpenCV je napsán nativně v C ++ a má templatované rozhraní, které bezproblémově funguje s kontejnery STL.“[1]

3.6.4 GitHub

GitHub je od roku 2019 bezplatná webová služba, která využívá verzovací nástroj git. Je vhodná pro ukládání rozsáhlejších programovacích projektů, jako jsou mimo jiné i bakalářské práce.

3.6.4.1 Git

Git je bezplatný a open source verzovací systém navržený tak, aby zvládl vše od malých až po velmi velké projekty s rychlostí a efektivitou.[8]

3.7 Požadavky

3.7.1 Funkční požadavky

- zpracování dopravního videozáznamu
- zobrazení výsledků ve webovém rozhraní
- uložení výsledků do databáze

3.7.2 Nefunkční požadavky

- jednoduchá práce s webovým rozhraním
- co nejvyšší přesnost sčítání
- co nejvyšší rychlost bez ztráty na přesnosti

Kapitola 4

Implementace

V této části je popsána implementace celé aplikace, která je celá napsána v programovacím jazyce Python. Ten je podrobněji popsán v kapitole *Použité technologie* této práce.

4.1 Repozitář kódu

Celý průběh vývoje aplikace je verzován v Git verzovacím systému. Repozitář byl vytvořen pod účtem OpenDataLab s GNU GPL (General Public License). Jeho stažení je volně dostupné na adrese: `https://github.com/opendatalabcz/carcounter`

4.2 YOLOv4 a Deep SORT

Pro účely této práce jsou použity již existující implementace **YOLOv4** a **Deep SORT**, protože jejich implementace je nad rámec této práce.
Použit byl projekt yolov4-deepsort jehož autorem je theAIGuysCode. Repozitář projektu je přístupný na stránkách githubu.com na adrese: `https://github.com/theAIGuysCode`

Konkrétně tento projekt byl vybrán, protože pro sledování objektů využívá již zmiňovaný Deep SORT s použitím YOLOv4 pro detekci objektů. Dle provedené analýzy se tedy jedná o jednu z nejlepších možností použitých metod.

4.3 Příprava yolov4.weights

V této práci je použita knihovna TensorFlow. Proto je potřeba převést oficiální předtrénovaný soubor yolov4.weights do formátu, který TensorFlow

vyžaduje. K tomu slouží *script* **save_model.py** s parametry –**model yolov4**Výsledný soubor uloží v požadovaném formátu do složky **checkpoints**.

4.4 Načtení videa od uživatele

V první řadě je potřeba získat od uživatele video, které má být zpracováno. Video může být jakýkoliv záznam z kamery (ve formátu mp4 nebo avi), který zabírá určitou část dopravní komunikace (silnice, dálnice). K tomu je využita jedna z HTTP metod, POST. Jde o způsob pro posílání informací webovému serveru a používá se mimo jiné i pro uploadování souborů.

Upload je možný na hlavní straně webové aplikace. Video se po uploadu ukládá do složky **uploads** a při úspěšném uploadu je zobrazeno hlášení, že vše proběhlo v pořádku. V případě neúspěchu program uživateli neumožní spustit program a je nutné nahrát validní soubor.

4.5 Spuštění výpočtu

Po nahrání souboru se uživateli zobrazí odkaz, kterým má možnost spustit samotný výpočet. Pro výběr je v šabloně odkaz, který pomocí funkce *url_for()* odkazuje na příslušný výpočet. Nad odkazem pro výpočet je zobrazena odhadovaná doba výpočtu. Po kliknutí na odkaz je spuštěn samotný výpočet.

4.6 Detekce a sledování objektů

Hlavní výpočty jsou prováděny ve funkci **detector**, která se nachází v souboru **main.py**. V první fázi jsou nastaveny potřebné parametry, jako např. *max_cos_distance*. Poté se inicializuje *tracker* Deep SORT a dochází ke konfiguraci detektoru YOLOv4 Tiny.
Následuje **while cyklus**, ve kterém je čteno video snímek po snímku. Každý použitý snímek je nutné upravit na formát, vyžadovaný detektorem. Snímek je tedy upraven na velikost 416×416 a barvy převedeny z formátu BGR na RGB. Poté je upravený snímek předán detektoru (YOLOv4 Tiny). Po zpracování detektor vrací ohraničující rámečky s příslušnými třídami. Jsou omezeny pouze na třídy *car*, *truck* a *bus*, které jsou pro tuto práci potřebné. V další fázi jsou detekce daného snímku předány *trackeru* (Deep SORT).

4.7 Počítání vozidel

Navrácené sledované objekty jsou iterovány v cyklu. V této fázi přichází na řadu samotné sčítání neduplicitních vozidel.

Sčítání vozidel funguje na jednoduchém principu. Ve středu obrazu jsou vytvořeny dva úzké pruhy. Jeden je veden vertikálně po celé výšce obrazu a pro-chází středem obrazu. Druhý je veden horizontálně po celé šířce obrazu. Šířky pruhů jsou výška, resp. šířka dělená 15. Pokud se sledovaný objekt nachází v oblasti jednoho (nebo obou) z pruhů a jedná se o jeden z požadovaných objektů (třída *car*, *truck* nebo *bus*), je objekt vložen do příslušného kontejneru *list*. Využity jsou 4 kontejnery. Tři reprezentují každý jednu ze zmíněných tříd, ty jsou typu *list*, a čtvrtý uchovává celkový součet spolu s informacemi o časovém rozložení. Pro tento účel je použit kontejner *dictionary*, kde jednotlivé klíče jsou čísla příslušných minut a hodnoty jsou kontejnery typu *list* se všemi objekty, sledovanými v příslušné minutě. Z každého kontejneru jsou na konci vybrány pouze originální objekty, a tím vznikají i finální výsledky počtů vozidel.

4.8 Předání informací uživateli

Po ukončení vypočtu zná již program všechny potřebné informace. Ty se následně ukládají do databáze a pomocí webové aplikace zobrazují uživateli. Výsledky jsou předány šabloně pomocí funkce *render_template()* s parametry *název šablony* a *výsledné proměnné*. Pomocí šablony *result.html* jsou podrobné výsledky zobrazeny uživateli.

4.9 Databázový systém

4.9.1 SQLite

„SQLite je in-proces knihovna, která implementuje samostatný, bezserverový, transakční databázový stroj s nulovou konfigurací. Kód pro SQLite je public doména a je tedy zdarma pro jakékoli účely, komerční i soukromé. SQLite je nejrozšířenější databáze na světě s více aplikacemi, než můžeme počítat, včetně několika významných projektů.“[2]

SQLite se od ostatních databází SQL liší zejména svojí jednoduchostí. Nemá totiž samostatný proces serveru. Zápis i čtení probíhají přímo do běžných diskových souborů. Celá databáze tedy potřebuje pouze jeden soubor. Další výhodou je možnost libovolně kopírovat databázi mezi 32bitovými a 64bitovými systémy nebo mezi architekturami big-endian a little-endian.

Rychlost běhu databáze závisí na velikost paměti, kterou má k dispozici. Obecně ale dosahuje dobré rychlosti, i když nemá k dispozici velké množství paměti.

4.9.2 Použití

Pro vytvoření databáze slouží v této práci velice jednoduchý soubor s názvem *database.py*. Spuštěním je vytvořen soubor database.db, kam jsou následně ukládána data z běhů programu. Stejný soubor je poté využíván i pro výpis obsahu databáze. Ta má pouze jednu tabulku, která je zobrazena na následujícím obrázku:

Obrázek 4.1: Databázový diagram

4.10 Webová aplikace

4.10.1 Flask

„*Flask je mikro webový framework napsaný v programovacím jazyce Python. Je klasifikován jako mikro webový framework, protože nevyžaduje konkrétní nástroje ani další vnitřní knihovny. Nemá žádnou vrstvu abstrakce databáze, ověřování formulářů ani žádné jiné komponenty třetích stran poskytující běžné funkce.*

Flask však podporuje rozšíření, která mohou přidávat do aplikace další funkce, jako by byly implementovány v samotném Flasku. Existují rozšíření pro objektově-relační mapovače, ověřování formulářů, zpracování nahrávání, různé technologie otevřeného ověřování a několik dalších souvisejících nástrojů pro tvorbu webových aplikací.

Mezi aplikace využívající Flask patří například služba Pinterest a LinkedIn.

V základu je Flask založen na Pocoo projektech, knihovně nástrojů Werkzeug a šablonovacímu systému Jinja2.“[6]

4.10.2 Šablony

Šablony jsou soubory, které jsou uloženy ve složce **templates**. Obsahují jak statická data, tak i *placeholdery* pro dynamická data. K zobrazení dat ze šablon využívá Flask knihovnu Jinja. Aby nebylo nutné u všech šablon vytvářet stejné pozadí, rozložení atd., využívá se zde dědění. K tomu se používá základní šablona, která se většinou označuje jako **base.html**. Šablona obsahuje nastavení základního vzhledu a rozložení, které poté využívají všechny ostatní šablony, což práci s nimi výrazně usnadňuje.

Šablony jsou soubory s příponou *.html* (Hypertext Markup Language) a k formátování rozložení a vzhledu stránky jsou zde použity kaskádové styly (CSS). Ty mají na starost např. barvu, velikost a styl textu nebo pozadí celé webové stránky.

Pro pozadí této webové aplikace byla použita fotografie, dostupná na adrese: `https://www.pexels.com/cs-cz/`. Licenci je možné nalézt na adrese: `https://www.pexels.com/cs-cz/license/`.

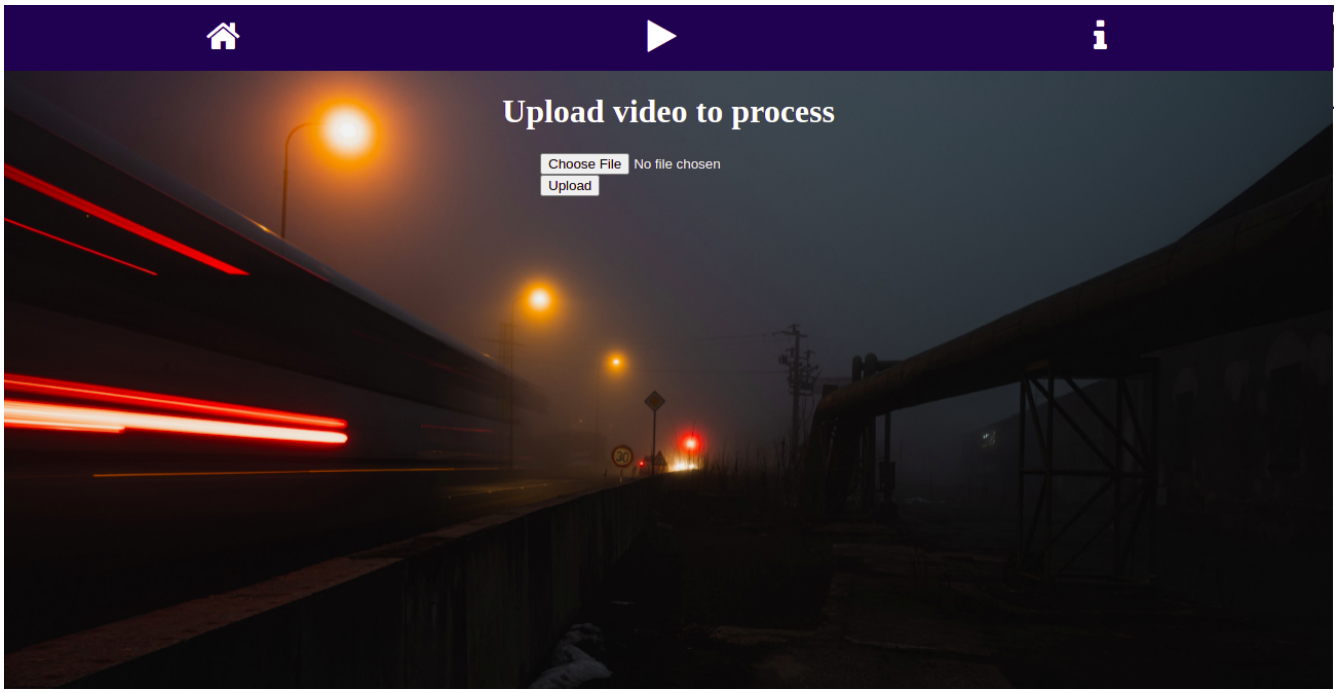
V této práci jsou využity následující šablony:

- base.html
- demo.html
- home_page.html
- info.html
- result.html

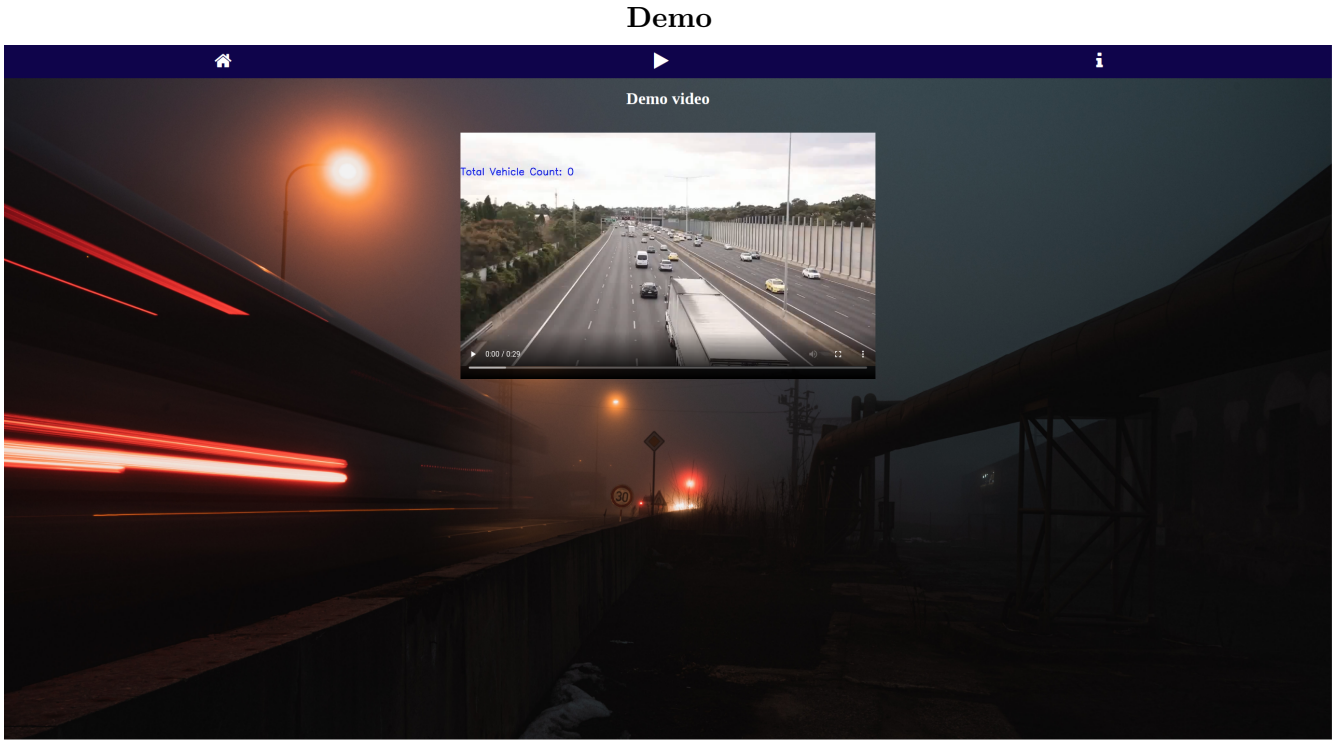
Jejich vzhled je možné vidět níže v sekci **Screenshoty**.

4.10.3 Screenshoty

Hlavní strana

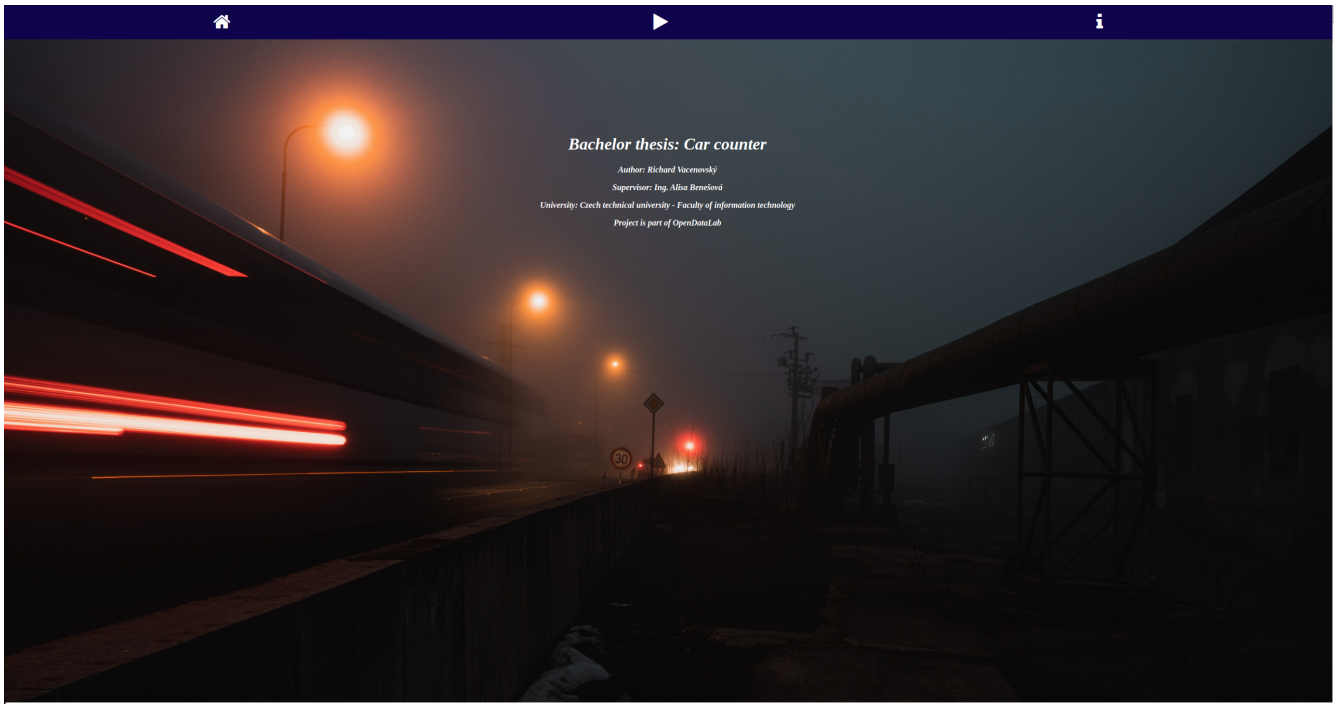


Obrázek 4.2: home_page



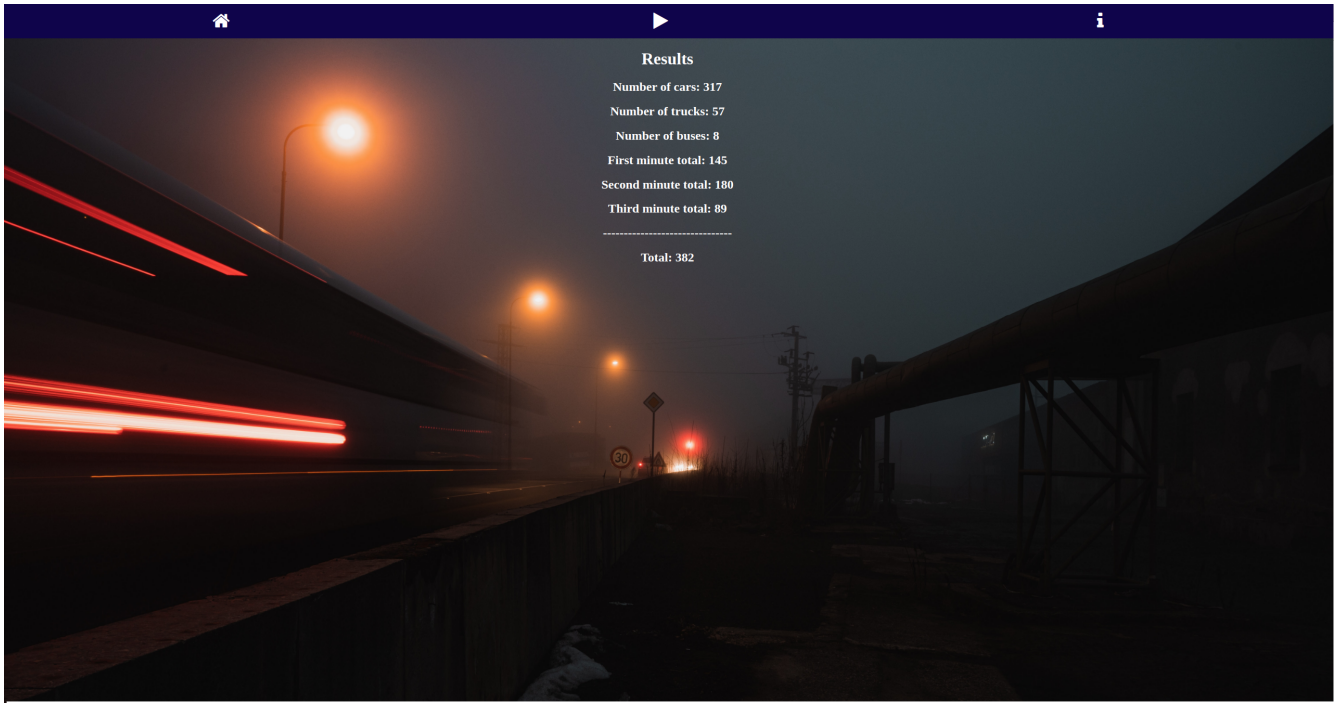
Obrázek 4.3: demo

Info



Obrázek 4.4: info

Výsledky



Obrázek 4.5: results

Kapitola 5

Testování

5.1 Hardware

Testování probíhalo na univerzitním počítači počítačové učebny v prostorech Fakulty Informačních Technologií Českého Vysokého Učení Technického v Praze. Počítač měl následující parametry:

- Paměť: 31.2 GiB
- Procesor: Intel® Xeon(R) CPU E3-1245 v6 @ 3.70GHz × 8
- Grafika: GeForce GTX 1060 3GB/PCIe/SSE2
- OS: openSUSE Leap 15.2
- OS typ: 64-bit
- Disk: 512.1 GB

5.2 Výsledky na MS COCO datasetu

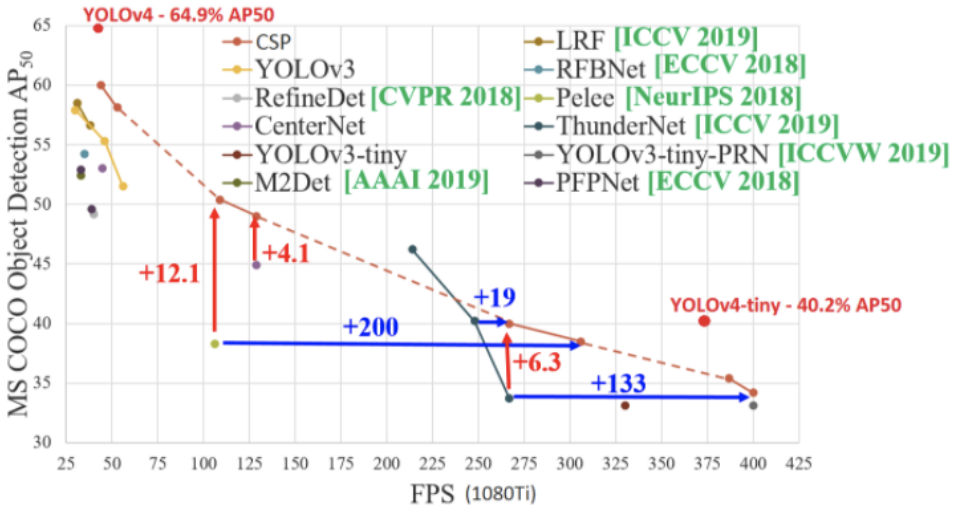
5.2.1 YOLOv4 vs. YOLOv4 Tiny

Při práci s metodou YOLO, v tomto případě konkrétně s verzí YOLOv4, jsou na výběr následující dva modely:

- YOLOv4
- YOLOv4 Tiny

Podle testů, provedených na MS COCO datasetu, což je velmi obsáhlý dataset od společnosti Microsoft, je model YOLOv4 Tiny přibližně 8× rychlejší

než YOLOv4. Na úkor rychlosti dosahuje pouze 66% přesnosti modelu YOLOv4. Použití YOLOv4 Tiny se tedy hodí hlavně v případech, kdy není k dispozici výkonná výpočetní technika a nižší přesnost je stále postačující.[26]



Obrázek 5.1: YOLOv4-tiny porovnání výkonnosti[26]

5.3 Testovací dataset

Testovací dataset pro tuto práci poskytla **AI CITY CHALLENGE 2021**. Videa z tohoto datasetu, která byla použita pro účely testování práce, mají vysoké rozlišení 1080p a 10fps. Je ale nutné zmínit, že 10 fps není u kamerových záznamů běžné. Obvykle bývají záznamy nahrávány na 30 fps. To znamená, že tato testovací videa odpovídají vyhodnocování každého třetího snímku z videa. Více informací o **AI CITY CHALLENGE 2021** se nachází na adrese: `https://www.aicitychallenge.org/`.

5.4 Vlastní testování - Verze 1

Vzhledem k tomu, že pro účely této práce využíváme pouze 3 z dostupných 80 tříd (car, truck, bus) a jedná se tedy o méně obecný problém. Bylo nutné provést testování obou modelů na vybraných datech, která snaží věrohodně simulovat očekávaný vstup. V první verzi je testována varianta, kde se započítávají vozidla, která jsou detektorem zachycena ve dvou úzkých pruzích, které vertikálně i horizontálně procházejí středem obrazu.

5.4.1 YOLOv4 vs. YOLOv4 Tiny

Z testovacího datasetu byla vybrána 3 videa. Dvě z nich jsou ze stejné kamery, která zabírá úsek dálnice. Třetí snímá křižovatku. Všechna videa jsou 30 sekund dlouhá a jejich frekvence je 10 fps. V tomto případě je vyhodnocován každý snímek (nedochází k přeskakování snímku pro zlepšení rychlosti). Na každém videu byl spuštěn nejprve model YOLOv4 a poté YOLOv4 Tiny. Každé video bylo na stejném modelu spuštěno dvakrát a následující výsledky jsou vždy průměrem z obou běhů.

5.4.1.1 Výsledky

Video	Model	Čas (s)	Počet	Správný počet
test_1.mp4	YOLOv4	160,56	19	25
	YOLOv4 Tiny	46,00	19	
test_2.mp4	YOLOv4	163,78	46	56
	YOLOv4 Tiny	49,21	40	
test_3.mp4	YOLOv4	159,93	15	19
	YOLOv4 Tiny	52,04	15	

Tabulka 5.1: YOLOv4 vs. YOLOv4 Tiny - verze 1

5.4.1.2 Diskuse

Na výsledcích je jasně vidět, že YOLOv4 Tiny je opravdu značně rychlejší než model YOLOv4. Porovnání běhů YOLOv4 Tiny oproti YOLOv4 je následující. Na prvním videu byl model 3,5× rychlejší, na druhém 3,3× rychlejší a na třetím 3,1× rychlejší. To znamená, že model YOLOv4 Tiny byl v průměru 3,3× rychlejší, než YOLOv4. Z toho vyplývá, že výsledky rychlostí modelů sice neodpovídají výsledkům na MS COCO datasetu, kde byla rychlost přibližně osminásobná, ale stále dosahuje více než trojnásobné rychlosti. Snížení rychlosti může být způsoben například vysokým rozlišením testovacích videí.
Druhým bodem je přesnost. Tady model YOLOv4 dosáhl 76%, 82% a 79% přesnosti na prvním, druhém a třetím testovacím videu. Druhý model YOLOv4 Tiny dosáhl kromě druhého testovacího videa stejných výsledků. U dru-hého videa měl úspěšnost jen 71 %, protože detekoval o 6 vozidel méně. Z porovnání obou modelů jasně vychází mnohem lépe menší model YOLOv4 Tiny, který byl v průměru 3,3× rychlejší a v průměru jen o 4 % méně přesný. Kdyby byla zanedbána doba inicializace detektoru a trackeru, dosahuje rychlost menšího z modelů sledování v reálném čase.

5.4.2 Přeskakování snímků

Pro zrychlení výpočtů se ještě nabízí jedna možnost, a to přeskakování snímků. Pro optimalizaci rychlosti, na úkor malého snížení přesnosti, je možné při výpočtu vyhodnocovat pouze každý i-tý snímek ve videu. V tomto případě jsou ale pro testování použita videa s 10 fps, což odpovídá použití každého třetího snímku. Následující testování se pokusí najít ideální i pro zvýšení rychlosti s přijatelnou ztrátou přesnosti detekce. Nejspíše ale zaznamená velký pokles přesnosti.

5.4.2.1 Výsledky

Video	Každý x snímek	Čas (s)	Počet	Správný počet
test_1.mp4	1	160,56	19	25
	2	105,44	5	
	4	81,05	1	
test_2.mp4	1	163,78	46	56
	2	110,14	18	
	4	82,50	2	
test_3.mp4	1	159,93	15	19
	2	108,32	7	
	4	83,24	3	

Tabulka 5.2: Přeskakování snímků YOLOv4 - verze 1

Video	Každý x snímek	Čas (s)	Počet	Správný počet
test_1.mp4	1	46,00	19	25
	2	37,57	3	
	4	33,47	0	
test_2.mp4	1	49,21	40	56
	2	39,41	5	
	4	34,86	0	
test_3.mp4	1	52,04	15	19
	2	41,54	5	
	4	35,01	2	

Tabulka 5.3: Přeskakování snímků YOLOv4 Tiny - verze 1

5.4.2.2 Diskuse

Jak je možné pozorovat z výsledků, došlo k očekávanému snížení přesnosti. Při použití každého druhého (u videa s 30 fps šestého) snímku již úspěšnost signifikantně klesla u obou modelů o více než 50 %. Při použití každého čtvrtého (u videa s 30 fps dvanáctého) už byla úspěšnost téměř nulová. Čas

obou modelů se přitom nedostal ani na polovinu původní hodnoty (průměrně 58 sekund oproti původním průměrným 104 sekundám).

5.4.3 Vlastní testování - Verze 2

Druhá verze se snaží vyřešit nedostatky verze první. V této variantě jsou vozidla započítávána na celé ploše obrazu videa, nikoliv pouze na dvou úzkých pruzích. Myšlenka této varianty spočívá v tom, že takto detektoru nemělo uniknout téměř žádné vozidlo. Nastává zde však riziko vzniku velkého množství duplicit. Ty jsou způsobeny nedokonalostí trackeru. Když tracker ztratí vozidlo na delší časový úsek a poté je opět objeví, považuje vozidlo za nové a započítá je tedy znovu. Testování probíhá na stejných třech videích jako při testování první verze, tj. třech videích s délkou 30 sekund a 10 fps.

5.4.4 YOLOv4 vs. YOLOv4 Tiny

5.4.4.1 Výsledky

Video	Model	Čas (s)	Počet	Správný počet
test_1.mp4	YOLOv4	157,03	28	25
	YOLOv4 Tiny	46,33	27	
test_2.mp4	YOLOv4	163,38	78	56
	YOLOv4 Tiny	48,62	58	
test_3.mp4	YOLOv4	159,22	25	19
	YOLOv4 Tiny	52,21	23	

Tabulka 5.4: YOLOv4 vs. YOLOv4 Tiny - verze 2

5.4.4.2 Diskuse

Při porovnání získaných výsledků, s výsledky správnými, je na první pohled jasné, že zde vznikají duplicity. Časy jsou totožné s časy první verze (průměrně 104 sekund). Rozhodující roli zde tedy bude hrát přesnost. Při pohledu na výsledky většího modelu, který by měl být i přesnější, je vidět, že zaznamenal v prvním, druhém a třetím videu o 3, 22 a 6 vozidel více, než by měl. To je v průměru o 27 % více než správný počet vozidel. Lze očekávat narůstající počet duplicit u videí, kde je větší hustota provozu. Výsledky menšího modelu vypadají na první pohled značně lépe, v průměru se liší jen o 11 %. To není způsobeno přesností modelu, ale právě naopak. Duplicity zde vznikají v podobném množství jako u většího modelu YOLOv4, ale kvůli nepřesnosti menšího z modelů nejsou některá vozidla zaznamenána vůbec. Výsledek je poté paradoxně blíže ke správné hodnotě, ale pouze zaviněním dvou nepřesností, které se navzájem vyruší. Když se v matematickém příkladu dvakrát špatně změní znaménko, tak také vyjde správný výsledek, postup je

však nekorektní. Tady se navíc nelze spoléhat na to, že by se u podstatně delšího videa nepřesnosti vyrušovaly i dále s takovou přesností.

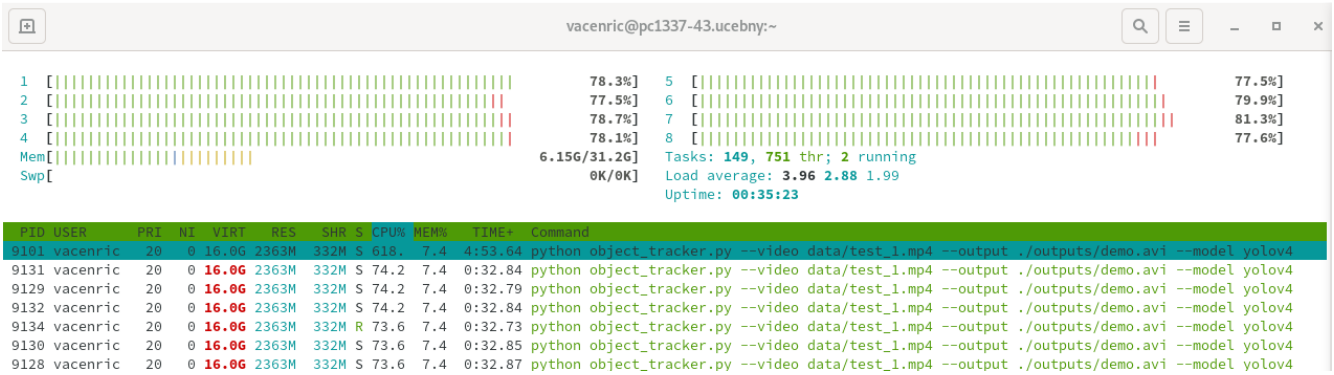
5.4.5 Přeskakování snímků

Přeskakování snímku v tomto případě nedává smysl. Problémem totiž není rychlost, ale přesnost. Ta se touto metodou nezlepší, a proto ani nebude testována.

5.5 Vytíženost CPU

5.5.1 Běh modelu YOLOv4

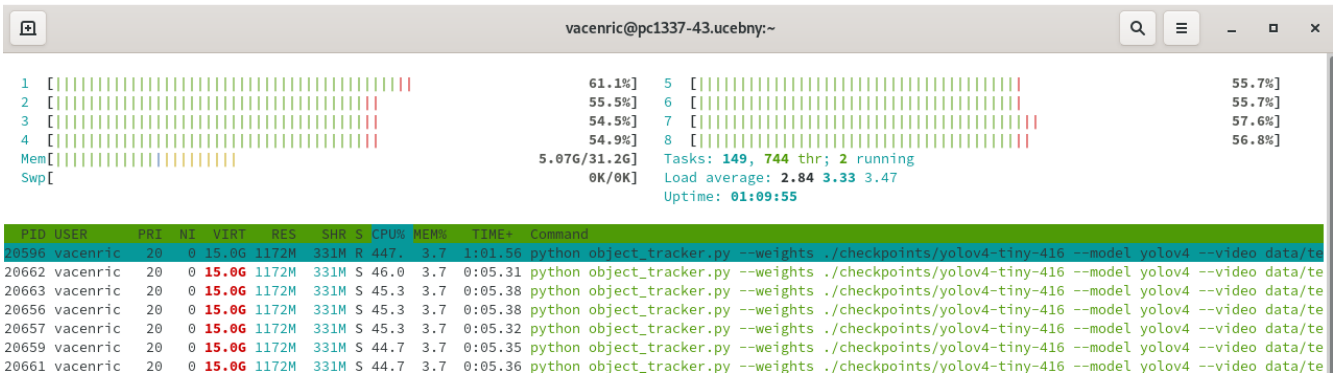
U běhu modelu YOLOv4 se vytížení CPU pohybovalo mezi hodnotami 600 % až 700 %. Vytíženost CPU dosahuje hodnot vyšších, než 100 % v pří-padech, kdy se jedná o výpočetně náročnou operaci a počítač disponuje více, než jedním jádrem. Počítač, na kterém testování probíhalo má jader 8, proto je možné dosahovat takto vysokých hodnot. Na následujícím obrázku je možné vidět snímek obrazovky, který zachycuje běh programu se spuštěným příkazem **htop**.



Obrázek 5.2: Vytížení CPU YOLOv4

5.5.2 Běh modelu YOLOv4 Tiny

U běhu modelu YOLOv4 Tiny se vytížení CPU pohybovalo mezi hodnotami 400 % až 500 %. Je tedy možné pozorovat, že běh menšího modelu není tak výpočetně náročný, jako běh modelu YOLOv4. Na následujícím obrázku je možné vidět snímek obrazovky, který zachycuje běh programu se spuštěným příkazem **htop**.



Obrázek 5.3: Vytížení CPU YOLOv4 Tiny

5.6 Porovnání výsledků obou verzí

5.6.1 První verze

První verze programu používá ke sčítání pouze velmi malou oblast, ve které jsou vozidla započítávána. Jedná se o dva úzké pruhy, procházející vertikálně a horizontálně středem obrazu.

5.6.1.1 Výhody

Díky této metodě tak dochází k minimálnímu vzniku duplicit. Aby byla totiž duplicita započítána, muselo by dojít ke změně identity vozidla přesně v tom úzkém pruhu, který slouží ke sčítání.

5.6.1.2 Nevýhody

Výhoda této metody sčítání je zároveň její nevýhodou. Vzhledem k tomu, že oba pruhy procházejí středem obrazu (vertikálně a horizontálně), tak musí vozidlo, aby bylo započítáno, minimálně jednu z těchto hranic *překročit*. Problém tedy nastává v případě, že vozidlo projíždí jen rohem obrazu. Započí-
-tána nejsou ani vozidla, která stojí v obrazu nehybně a nenacházejí se přímo v oblasti jednoho z detekčních pruhů. Tento fakt ale tolik nevadí, protože úkolem práce je sčítat projíždějící vozidla.

5.6.2 Druhá verze

Druhá verze programu využívá ke sčítání celou plochu obrazu snímku. Snaží se tak detekovat všechna vozidla v obraze a optimalizovat tak první verzi.

5.6.2.1 Výhody

Výhodou této metody je, že opravdu detekuje veškerá vozidla, která se v obraze nacházejí. Sčítá vozidla bez ohledu na to, kde se během svého pohybu v daném snímku nachází. Dokonce jsou započítávána i vozidla, která jsou nehybná.

5.6.2.2 Nevýhody

Nevýhodou této metody jsou však duplicity. Protože je monitorována celá plocha obrazu, veškeré duplicity, které vzniknou, jsou okamžitě přičteny k celkovému počtu vozidel. Takových duplicit bohužel vzniká celkem velké množství, a to i u většího a přesnějšího z modelů (YOLOv4). Další nevýhodou druhé verze je přičítání nehybných vozidel, což není cílem této práce.

5.7 Závěr testování

Po porovnání všech testovaných variant vyšla výrazně nejlépe první verze sčítání, která využívá model YOLOv4 Tiny a vyhodnocuje každý snímek v testovacím videu. To ve skutečnosti znamená vyhodnocení každého třetího snímku. Videa trvající 30 sekund s 10 fps model vyhodnotil v průměru za 49 sekund a zachytil 74 % vozidel. Hodnota 100 % vyjadřuje všechna vozidla včetně nehybných. Nehybná vozidla zde ale tvoří pouze zanedbatelnou část. Ve všech testovacích videích se celkem nachází pouze 3.

Kapitola 6

Závěr

Cílem práce bylo vytvořit systém s přehledným a srozumitelným webovým rozhraním, který bude schopen spočítat projíždějící vozidla na základě záznamu z kamery. Systém má za úkol vyhodnotit záběr z kamery, která je namířena na libovolnou dopravní komunikaci (silnice, dálnice). Dále bylo cílem analyzovat dostupné otevřené zdroje, na kterých lze počítat projíždějící vozidla a data vhodně rozdělit. Záznamy z kamer na českých silnicích bohužel nejsou veřejně dostupné. Místo toho byl použit dataset z AI CITY CHALLENGE. Všechny ostatní body se podařilo v této práci splnit. Výsledkem je tedy funkční webová aplikace, kde se nachází video s názornou ukázkou, jak celé vyhodnocování probíhá, ale hlavně stránka, kam může uživatel nahrát své vlastní video a nechat je systémem vyhodnotit a spočítat jednotlivá vozidla. Uživatelské rozhraní bylo testováno dvěma lidmi nezávisle na sobě. Jedním z nich je spolužák z Fakulty Informačních technologií na ČVUT. Druhým je rodinný příslušník, který je z odlišného prostředí a nemá povědomí o informačních technologiích. Oba rozhraní označili jako jednoduché a přehledné.

Práce dále rozebrala jednotlivé metody a technologie, které v ní byly použity, což jsou hlavně nástroje pro rozpoznání, klasifikaci, detekci a sledování (anglicky tracking) objektů. Nechybí ani popis následného vývoje aplikace a poté testování. Testování ukázalo, že ze všech testovaných variant vyšla výrazně nejlépe první verze sčítání. Jedná se o verzi sčítání, která využívá pouze dva úzké pruhy, procházející středem obrazu vertikálně a horizontálně. Nejlepších výsledků bylo dosaženo za použití modelu YOLOv4 Tiny s vyhodnocováním každého snímku na testovacím videu. To v reálném případě znamená vyhodnocení každého třetího snímku. Videa trvající 30 sekund s 10 fps model vyhodnotil v průměru za 49 sekund a zachytil 74% vozidel.

Repozitář této práce je přístupný na adrese: `https://github.com/vacenovskyrichard/carcounter`, kde je možné pozorovat postupný průběh a vývoj aplikace.

V tuto chvíli je aplikace nastavena na vyhodnocování rovných úseků silnic či dálnic a počítá vozidla v plném rozsahu obrazu videa. Je zde tedy určitě mnoho možností, jak aplikaci dále rozvíjet a vylepšovat. Jednou možností by bylo zlepšit metodu sčítání tak, aby aplikace uměla započítat vozidla, bez ohledu na trajektorii jejich pohybu, a to bez zbytečných duplicit. Příkladem takového záznamu je např. složitější křižovatka. Další variantou by mohla být možnost vytvoření tzv. masky, což by umožnilo, např. V případě dálnice, omezit sčítání pouze na jeden směr provozu.

Bibliografie

[1] *About [online]*. web page. [Citováno 2021-4-24]. URL: https://opencv.org/about/.

[2] *About SQLite [online]*. web page. [Citováno 2021-5-5]. URL: https://www.sqlite.org/about.html.

[3] Alex Bewley et al. *Simple Online and Realtime Tracking [online]*. web page. [Citováno 2021-4-23]. URL: https://www.arxiv-vanity.com/papers/1602.00763/.

[4] Jason Brownle. *A Gentle Introduction to Object Recognition With Deep Learning [online]*. web page. [Citováno 2021-4-5]. Břez. 2019. URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/.

[5] *Dark side of neural networks explaine [online]*. web page. [Citováno 2021-4-5]. Led. 2021. URL: https://research.aimultiple.com/how-neural-networks-work/.

[6] *Flask [online]*. web page. [Citováno 2021-5-5]. Led. 2021. URL: https://cs.wikipedia.org/wiki/Flask.

[7] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [online]*. web page. [Citováno 2021-4-6]. Čvc 2018. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[8] *git [online]*. web page. [Citováno 2021-5-7]. URL: https://git-scm.com.

[9] Ray Johns. *PyTorch vs TensorFlow for Your Python Deep Learning Project [online]*. web page. [Citováno 2021-5-7]. Zář. 2020. URL: https://realpython.com/pytorch-vs-tensorflow/.

[10] Darko Jurić. *Object Tracking: Kalman Filter with Ease [online]*. web page. [Citováno 2021-4-26]. Led. 2015. URL: https://www.codeproject.com/articles/865935/object-tracking-kalman-filter-with-ease.

[11] Ritesh Kanjee. *DeepSORT — Deep Learning applied to Object Tracking [online]*. web page. [Citováno 2021-4-26]. Srp. 2020. URL: https://augmentedstartups.medium.com/deepsort-deep-learning-applied-to-object-tracking-924f59f99104.

[12] Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity. *Koncept umělé neuronové sítě [online]*. web page. [Citováno 2021-4-6]. URL: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--uvod-do-neuronovych-siti--koncept-umele-neuronove-site.

[13] *Long short-term memory [online]*. web page. [Citováno 2021-5-4]. Břez. 2021. URL: https://en.wikipedia.org/wiki/Long_short-term_memory.

[14] Shishira R Maiya. *DeepSORT: Deep Learning to Track Custom Objects in a Video [online]*. web page. [Citováno 2021-4-26]. Dub. 2019. URL: https://nanonets.com/blog/object-tracking-deepsort/#deep-sort.

[15] *Meanshift Algorithm for the Rest of Us (Python) [online]*. web page. [Citováno 2021-4-5]. Květ. 2016. URL: http://www.chioka.in/meanshift-algorithm-for-the-rest-of-us-python/.

[16] Michal Mitrenga. *Konvoluční neuronová síť pro segmentaci obrazu [online]*. web page. [Citováno 2021-4-7]. 2018. URL: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=177588.

[17] *Optical Flow [online]*. web page. [Citováno 2021-4-26]. Dub. 2021. URL: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html.

[18] *Optical flow [online]*. web page. [Citováno 2021-4-26]. Dub. 2021. URL: https://en.wikipedia.org/wiki/Optical_flow.

[19] *Otevřené vzdělávací zdroje [online]*. web page. [Citováno 2021-4-20]. Lis. 2020. URL: https://cs.wikipedia.org/wiki/Otev%C5%99en%C3%A9_vzd%C4%9Bl%C3%A1vac%C3%AD_zdroje.

[20] Radu Tudor Ionescu Petru Soviany. *Optimizing the Trade-off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction [online]*. web page. [Citováno 2021-4-6]. Srp. 2018. URL: https://arxiv.org/pdf/1803.08707.pdf.

[21] *Python [online]*. web page. [Citováno 2021-4-22]. Břez. 2021. URL: https://cs.wikipedia.org/wiki/Python.

[22] Joseph Redmon et al. *You Only Look Once: Unified (Volný překlad RV), Real-Time Object Detection [online]*. web page. [Citováno 2021-4-6]. Květ. 2016. URL: https://arxiv.org/pdf/1506.02640v5.pdf.

[23] Adrian Rosebrock. *YOLO object detection with OpenCV [online]*. web page. [Citováno 2021-4-6]. Lis. 2018. URL: https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/.

[24] Amisha J. Shah. *OPTICAL FLOW FOR MOTION DETECTION WITH MOVING BACKGORUND [online]*. web page. [Citováno 2021-4-5]. Říj. 2015. URL: https://www.researchgate.net/figure/Fig-5-Lucas-Kanade-with-pyramid_fig1_282913643.

[25] Deval Shah. *The Surveillance phenomenon you must know about : Multi Object Tracking. [online]*. web page. [Citováno 2021-4-5]. Dub. 2020. URL: https://cv-tricks.com/object-tracking/quick-guide-mdnet-goturn-rolo/.

[26] Jacob Solawetz a Samrat Sahoo. *Train YOLOv4-tiny on Custom Data - Lightning Fast Object Detection [online]*. web page. [Citováno 2021-4-30]. Čvc 2020. URL: https://blog.roboflow.com/train-yolov4-tiny-on-custom-data-lighting-fast-detection/.

[27] *TensorFlow [online]*. web page. [Citováno 2021-4-23]. Dub. 2021. URL: https://en.wikipedia.org/wiki/TensorFlow.

[28] Lilian Weng. *Object Detection Part 4: Fast Detection Models [online]*. web page. [Citováno 2021-4-6]. Pros. 2018. URL: https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html#two-stage-vs-one-stage-detectors.

[29] *Why TensorFlow [online]*. web page. [Citováno 2021-4-23]. URL: https://www.tensorflow.org/about.

[30] Jiří Zacha. *Konvoluční neuronové sítě pro klasifikaci objektů z LiDARových dat [online]*. web page. [Citováno 2021-4-6]. Květ. 2019. URL: https://dspace.cvut.cz/bitstream/handle/10467/82351/F3-BP-2019-Zacha-Jiri-Konvolucni_neuronove_site_pro_klasifikaci_objektu_z_LiDARovych_dat.pdf.

Instalační příručka

Prerekvizity

Jedinou prerekvizitou pro instalaci programu je nainstalovaná **conda**. Nejjednodušší způsob, jak ji získat, je instalace **Minicondy**, což je minimální verze **Anacondy**, která obsahuje pouze příkaz **conda** a příslušné závislosti. Instalace pro kteroukoliv distribuci (Windows, Linux, Mac OS) je dostupná na adrese: `https://docs.conda.io/en/latest/miniconda.html` .

Prerekvizity pro instalaci Minicondy

- 32-bit nebo 64-bit počítač
- 400 MB volného místa na disku
- jedna ze tří distribucí Windows, Linux nebo Mac OS

Naklonování repozitáře

Nejprve je nutné naklonovat repozitář se zdrojovým kódem a ostatními potřebnými soubory. Repozitář je dostupný na adrese: `https://github.com/vacenovskyrichard/carcounter`. Repozitář je možné stáhnout jako zip nebo naklonovat pomocí příkazu *git clone*.

Stažení oficiálního YOLOv4 Tiny weights souboru

Dále je potřeba samostatně stáhnout oficiální předtrénovaný soubor YOLOv4 Tiny weights. Ten je dostupný na následující adrese: `https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.weights`. Soubor je nutné uložit do složky **data**.

Vytvoření a spuštění virtuálního prostředí

Pomocí příkazu conda je nutné vytvořit virtuální prostředí, které obsahuje všechny potřebné knihovny a balíčky. Virtuální prostředí vytvoříme pomocí následujícího příkazu:

```
$ conda env create -f conda-carcounter.yml
```

Aktivace prostředí:

```
$ conda activate car_counter
```

Zformátování YOLOv4 Tiny weights souboru do TensorFlow formátu

Pro zformátování slouží následující příkaz:

```
$ python save_model.py --weights ./data/yolov4-tiny.weights
  --output ./checkpoints/yolov4-tiny-416 --model yolov4 --tiny
```

Spuštění

V tuto chvíli by již mělo být vše připraveno. Pro spuštění celé aplikace slouží příkaz:

```
$ python main.py
```

Příloha A

Zkratky

AI Artificial Intelligence

CNN Convolutional Neural Network

CPU Central Processing Unit

fps frames per second

GUI Graphical User Interface

MOT Multiple object tracking

NN Neural Network

SORT Simple Online and Realtime Tracking

SOT Single Object Tracking

XML Extensible Markup Language

YOLO You Only Look Once