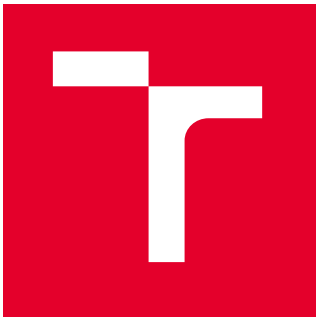


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ZAŘÍZENÍ PRO VYHODNOCENÍ DOPRAVNÍ SITUACE

DEVICE FOR THE TRAFFIC SITUATION EVALUATING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Matej Gábel**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Ilona Janáková, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Matej Gábel ***ID:*** 230065

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Zařízení pro vyhodnocení dopravní situace

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je navrhnout a realizovat asistenční systém pro záznam a vyhodnocení dopravní situace. Cílem práce je vytvoření embedded zařízení umístěného uvnitř vozidla sledující situaci před vozidlem a zpracovávajícího snímky v reálném čase. Předpokládá se zpracování obrazu pro rozpoznání dopravního značení.

1. Seznamte se s danou problematikou. Proveďte rešerši existujících přístupů.
2. Navrhněte vhodné hardwarové prostředky - snímací a vyhodnocovací část, případně část pro vhodnou prezentaci výsledků.
3. Navržené zařízení realizujte.
4. Pořiďte dostatečně rozsáhlou a pestrou databázi reálných snímků.
5. Navrhněte vhodné algoritmy pro zpracování snímků vzhledem k předpokládanému využití.
6. Navržené algoritmy implementujte a řádně otestujte.
7. Celý systém zhodnoťte. Definujte omezující podmínky.

DOPORUČENÁ LITERATURA:

BROGGI, Alberto, Pietro CERRI, Paolo MEDICI, Pier Paolo PORTA a Guido GHISIO. Real Time Road Signs Recognition. In: 2007 IEEE Intelligent Vehicles Symposium [online]. IEEE, 2007, 2007, s. 981-986 [cit. 2022-09-08]. ISBN 1-4244-1067-3. ISSN 1931-0587. Dostupné z: doi:10.1109/IVS.2007.4290244

SUN, Chang, Yibo AI, Sheng WANG a Weidong ZHANG. Dense-RefineDet for Traffic Sign Detection and Classification. Sensors [online]. 2020, 20(22) [cit. 2022-09-08]. ISSN 1424-8220. Dostupné z: doi:10.3390/s20226570

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 22.5.2023

Vedoucí práce: Ing. Ilona Janáková, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

Abstrakt

Bakalárska práca sa zaoberá vytvorením zariadenia pre vyhodnotenie dopravnej situácie konkrétne detekciou značiek. V tejto práci sú vyskúšané vybrané metódy rozpoznávania dopravných značiek, kde výsledná implementácia na hardware je vykonaná pomocou konvolučných neurónových sietí. Jedná sa presnejšie o architektúru YOLOv5, ktorá je vhodná pre rozpoznávanie dopravných značiek real time.

Klúčové slová

Rozpoznávanie, detekcia, dopravné značky, spracovanie obrazu, konvolučná neurónová sieť, YOLO, PyTorch, OpenCV, Raspberry Pi

Abstract

The bachelor's thesis deals with the implementation of a device for evaluating the traffic situation, specifically by traffic signs detection. In this work, I tried different methods of traffic sign recognition, where the resulting implementation on hardware is done using convolutional neural networks. More precisely, it is the YOLOv5 architecture, which is suitable for recognizing traffic signs in real time.

Keywords

Recognition, detection, traffic signs, image processing, convolutional neural network, YOLO, PyTorch, OpenCV, Raspberry Pi

Bibliografická citácia

GÁBEL, Matej. *Zařízení pro vyhodnocení dopravní situace* . Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/151674 Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Ilona Janáková

Prehlásenie autora o pôvodnosti diela

Meno a priezvisko študenta:	*Matej Gábel*
VUT ID študenta:	*230065*
Typ práce:	*Bakalárska práca*
Akademický rok:	*2022/23*
Téma záverečné práce:	*Zariadenie pre vyhodnotenie dopravnej situácie*

Prehlasujem, že svoju záverečnú prácu som vypracoval samostatne, pod vedením vedúcej/ceho záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia §11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky 40/2009 Sb.

V Brne dňa: 18. Mája 2023 ----------

podpis autora

Poďakovanie

Rád by som sa poďakoval vedúcej bakalárskej práce pani Ing. Ilone Janákovej, Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

V Brne dňa: 18.Mája 2023 ----------

podpis autora

Obsah

ZOZNAM SKRATIEK

Skratky:

FEKT	Fakulta elektrotechniky a komunikačních technologií
VUT	Vysoké učení technické v Brně
GPS	Global Positioning System
ROI	Oblasť záujmu
CNN	Konvolučná neuronová sieť
SVM	Support Vector Machine
SIFT	Scale-invariant feature transform
SURF	Speeded up robust features
MSER	Maximally stable extremal regions
HoG	Histogram of oriented gradients
FOV	Field of view
FPS	Frame per second
GTSRB	German Traffic Sign Recognition Benchmark
YOLO	You Only Look Once
px	Pixel

ZOZNAM OBRÁZKOV

Úvod

Dopravné značky majú veľký význam v doprave, regulujú, informujú a upozorňujú vodičov, udávajú im príkazy, zákazy a obmedzenia za účelom zvýšenia bezpečnosti v cestnej premávke. Sú vyhotovené v rôznych tvaroch a farbách, aby upútali pozornosť a upozornili na svoj význam. Zo štúdie vyplýva, že výrazné dopravné značky zvyšujú pozornosť vodičov. Taktiež má na vodičov rôzny vplyv aj vzdialenosť a doba viditeľnosti dopravnej značky. V dnešnej dobe majú význam asistenčné alebo podporné systémy riadenia, ktoré pomáhajú vodičovi a zvyšuje sa tak bezpečnosť na cestách. Jedným z jeho podsystémov je aj systém na rozpoznávanie dopravných značiek.

Tieto systémy sú v dnešnej dobe inštalované už takmer do všetkých automobilov a dokážu rozpoznávať všetky typy dopravných značiek. Princíp ako to funguje si vysvetlíme napríklad na značke najvyššej dovolenej rýchlosti. Ak vodič prekročí dovolenú rýchlosť v prípade, že bola zdetekovaná značka maximálnej dovolenej rýchlosti, systém ho na to upozorní buď zvukovým efektom alebo zvýraznením na palubnom počítači. Ďalším využitím rozpoznávania dopravných značiek je pre tvorcov máp a navigačných systémov.

Dnes v niektorých navigačných systémoch funguje upozornenie prekročenia rýchlosti tak, že systém zistí rýchlosť pomocou GPS a porovná ju s najvyššie dovolenou rýchlosťou ktorá je priradená danej ceste, po ktorej vodič aktuálne prechádza.

Avšak cieľom tejto práce je vytvoriť rozpoznávací systém, ktorý má za úlohu rozpoznávať dopravné značenie na cestách. V prvej kapitole 1 sa zaoberám históriou, rozdelením a rôznorodosťou dopravných značiek. Nasledujúca kapitola 2 sa zaoberá základným princípom rozpoznávania dopravných značiek a možnými metódami detekcie. Tretia kapitola 3 popisuje hardwarovú časť, kde vyberám jednotlivé hardwarové komponenty potrebné pre vyhotovenie zariadenia. Kapitola 4 popisuje softwarovú časť, kde som otestoval rôzne metódy rozpoznávania dopravných značiek a taktiež vytvorenie užívateľského rozhrania. Predposledná kapitola 5 sa zaoberá výsledným testovaním a implementáciou na hardware.

1. PREHĽAD PROBLEMATIKY

Jedným z najpodstatnejších prvkov cestnej infraštruktúry sú dopravné značky. Značky upozorňujú účastníkov cestnej premávky na nebezpečenstvo, udávajú smer a rýchlosť jazdy, informujú o vzdialenosti a smere cieľa a podobne. Pre vedenie motorového vozidla je potrebné vedieť význam všetkých dopravných značiek.

1.1 História dopravných značiek

Dopravné značky dnes patria k najdôležitejším elementom dopravnej infraštruktúry. Vyvíjali sa postupom času tak, ako rástlo množstvo dopravných prostriedkov na cestách. Najväčší posun vo vývoji dopravných značiek priniesol až rozvoj cyklistického a následne automobilového priemyslu od konca 19. storočia. V mnohých krajinách potom začali vznikať rôzne značky, čo bolo pre medzinárodnú dopravu nežiadúce.

Preto bolo nevyhnutné ich zjednotenie a štandardizovanie. Najdôležitejšia dohoda o dopravnom označení vznikla v roku 1949 v Ženeve. Táto zmluva tvorí základ súčasného európskeho systému. Dopravné značky definuje ako znakové symboly, kde tieto značky neobsahujú slová, aby boli pochopiteľné a jednoducho zrozumiteľné aj pre negramotných ľudí a cudzincov. Napriek tomu, nie všetky krajiny ju prijali a ratifikovali. Medzi krajiny, ktoré ju nepodpísali patrí napr. USA, Kanada, či Austrália.[1][3]

1.2 Rozdelenie dopravných značiek

Zo zákona o premávke na pozemných komunikáciách sa dopravné značky delia do dvoch hlavných kategórií, na zvislé a vodorovné. Táto práca sa zaoberá iba zvislým dopravným značením. Zvislé dopravné značky sa delia na:

- Výstražné značky
- Zákazové značky
- Príkazové značky
- Informatívne značky
- Dodatkové tabule

Z pohľadu spracovania obrazu ich delíme podľa ich farby a geometrického tvaru.

Rozdelenie na základe farby[3]:

- **Červené** - Jedná sa o dopravné značky výstražné, zákazové a značky daj prednosť v jazde. Taktiež sú jednými z najdôležitejších dopravných značiek . Červená farba je najlepšie rozoznateľná vo farebnom prostredí.

- **Oranžové a žlté** - sa používajú na informovanie vodičov o dočasných alebo trvalých nebezpečenstvách. Na upozornenie vodiča sa najčastejšie používa

čierny text (znak), ktorý je na žltom podklade, aby značka bola maximálne čitateľná a viditeľná.

- **Modré** - Väčšina príkazových a informatívnych dopravných značiek je modrej farby. Charakter týchto značiek je informatívny.

- **Zelené** - Patria sem informatívne smerové tabule. Tieto značky obsahujú informáciu ako napríklad výjazdové cesty na diaľnici.

- **Čierne a biele** – cieľom čiernych a bielych značiek je informovať vodiča bez rozptyľovania. Patria sem dodatkové tabule, ktoré obsahujú čierny text alebo znak na bielom podklade.

Rozdelenie na základe tvaru[3]:

- **Kruhové značky** - označujú zákaz alebo obmedzenie (zákazové, príkazové značky).

- **Trojuholníkové značky** - varujú pred nebezpečenstvom (výstražne značky).

- **Osemuholník** – iba značka STOJ.

- **Obdĺžnikové alebo štvorcové značky** - obsahujú doplnkové informácie (informatívne značky a dodatkové tabule).

- **Kosoštvorec** - značka označujúca hlavnú cestu alebo koniec hlavnej cesty

1.3 Rôznorodosť dopravných značiek

V súčasnosti existujú na svete dva systémy dopravných značiek, prvý je európsky na základe Viedenskej konvencie a americký. Avšak mnohé krajiny prijali vlastné zmeny a špecifikácie dopravných značiek. Hlavné rozdiely značiek používaných v rôznych krajinách sú v grafike, farbách, tvare, veľkosti, fonte písma, merných jednotkách, lokalizovaných textoch a význame.

Prehľad rôznorodosti dopravných značiek je ukázaný na obrázku č.1.1. V tomto obrázku si môžeme všimnúť, ako sa jednotlivé značky odlišujú hlavne v pozadí a tvaroch. Typickou ukážkou amerických značiek sú výstražné značky v tvare kosoštvorca so žltým pozadím v porovnaní s európskymi v klasickom tvare trojuholníka. Podotknúť treba aj to, že značný počet amerických značiek používa na vyjadrenie len textovú podobu namiesto piktogramov. [1]

Obrázok 1.1: Porovnanie dopravných značiek [1]

Ak by sme chceli vytvoriť systém na rozpoznávanie dopravných značiek, tak by sme museli počítať s rôznorodosťou dopravných značiek. Táto práca je zameraná na dopravné značky nachádzajúce sa na Slovensku. [1]

2. ROZPOZNÁVANIE DOPRAVNÝCH ZNAČIEK

Rozpoznávanie dopravných značiek v podstate slúži ako druhá sada očí vodiča tým, že zvyšuje povedomie o špecifických značkách na vozovke, čo pomáha vodičovi vykonávať lepšie a bezpečnejšie rozhodnutia počas jazdy. Je to teda bezpečnostný systém, ktorý rozpoznáva dopravné značky a prenáša informácie zobrazené na značke vodičovi cez združený prístroj, obrazovku infotainmentu alebo head-up displej. [2]

Tvorcovia týchto bezpečnostných systémov sa zameriavajú hlavne na podstatné značky, ktoré môžu byť životu nebezpečné. Jedná sa o značky napríklad zákazové, v ktorých sa tvorcovia zameriavajú hlavne na značky upravujúce maximálnu povolenú rýchlosť a zákaz predbiehania. Z ďalších skupín sa zameriavajú hlavne na značky výstražné a značky upravujúce prednosť v jazde, kde patrí napríklad dôležitá značka STOP.

GPS (Global Positioning System) sa využíva na navigáciu vodičov čo najefektívnejšou trasou, takže sa tieto systémy dajú použiť na ukladanie dopravných značiek do mapových súborov na základe GPS súradníc. To znamená, že v prípade, že na ceste sú dopravné značky, ktoré obmedzujú rýchlosti, práce na ceste a zápchy, tak GPS môže použiť tieto informácie na naplánovanie čo najlepšej trasy s čo najmenším obmedzením.

Metódy na detekciu dopravných značiek sú rozdelené do troch kategórií, rozpoznávanie na základe:

- farby,
- tvaru,
- strojového učenia.

Faktory ovplyvňujúce výber a výkonnosť jednotlivých metód sú napríklad:

- presnosť,
- rýchlosť,
- výpočtová náročnosť – ovplyvňuje výkon systému a vykonávanie požadovanej úlohy,
- flexibilita – metóda je špecializovaná len na jeden druh dopravných značiek alebo na niekoľko tried,
- okolité podmienky – použitie len počas dňa, v noci alebo počas nepriaznivého počasia,
- spracovanie – detekcia prebieha v reálnom čase alebo spracovanie v oneskorenom čase (offline processing).

Po určení požiadaviek sa na základe týchto faktorov vyberie metóda, ktorá týmto požiadavkám zodpovedá. V praxi to znamená, pokiaľ máme čiernobiely obrázok a chceme rozpoznávať rôzne druhy značiek, tak metóda detekcie na základe farby nebude

vhodná. Namiesto toho sa dá použiť detekcia na základe tvaru, kde každá značka má svoj špecifický tvar.

V opačnom prípade, ak máme farebný obraz, tak metóda detekcie na základe farby môže byť veľmi užitočná.

Metóda strojového učenia je vhodná, ak máme veľa nameraných dát , resp. v našom prípade databázu so značkami (dataset) . Ak dataset nemáme, metóda je prakticky nepoužiteľná.

2.1 Princíp rozpoznávania dopravných značiek

Vo všeobecnosti, postup pre rozpoznávanie dopravných značiek môžeme vyjadriť nasledujúcim postupom:

Obrázok 2.1: Postup rozpoznávania dopravných značiek[1]

Po získaní obrázka z nejakého zdroja (snímaním z kamery alebo z nahraného videa) nasleduje jeho predspracovanie, čo znamená použitie filtrov, odstránenie, resp. potlačenie šumu a pripravenie obrázka na ďalší krok, ktorým je detekcia. Tá zahŕňa segmentáciu, oddelenie popredia od pozadia a na základe toho identifikuje možné objekty, ktoré by mohli byť dopravnými značkami. Tieto objekty sa ďalej podrobujú klasifikácii do tried, čo predstavuje rozpoznanie konkrétnych dopravných značiek. Teda, výsledkom klasifikácie je určenie konkrétneho označenia typu značky, prípadne rozhodnutie, či to značka je alebo nie je.[1]

2.1.1 Snímanie obrazu

Vykonáva sa pomocou zariadení pre snímanie ako sú napríklad kamery. Tá sa skladá zo sústavy fotosenzorov a transformuje obrazovú informáciu na elektrický signál. Jedná sa o senzory CCD a CMOS. Tieto kamery sa v autách nachádzajú ako doplnková výbava, čiže sú zabudované priamo v aute a sú zobrazované na palubnom počítači, alebo je možné si kameru kúpiť separátne a obraz alebo video získať externe.

2.1.2 Predspracovanie obrázka

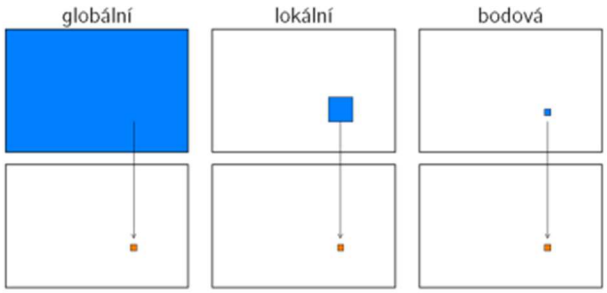
Jedná sa o základnú úpravu signálu, kde sa obrázok upravuje, aby pri segmentácií určil čo najpresnejšie príznaky z daného obrázka. Používajú sa rôzne operácie, ktoré pomáhajú odstrániť nedostatky, ktoré vznikli pri snímaní obrazu pomocou snímacieho zariadenia. Väčšinou sa jedná o potlačenie šumu, skreslenia. Najpoužívanejšie metódy predspracovania obrazu sú: [38][28]

- Jasové transformácie
  - Lokálne predspracovanie,
- Geometrické transformácie.

Jasové transformácie

Transformácia závisí od vlastností pixela samotného. Jasové transformácie sa ďalej delia na primárne podľa veľkosti okolia vyšetrovaného bodu na[26]:

- globálne – nová hodnota pixelu je vypočítaná z hodnôt celého obrazu,
- lokálne – nová hodnota pixelu je vypočítaná z hodnôt lokálneho okolia pixelu,
- bodová – nová hodnota pixelu je vypočítaná len z hodnoty toho istého pixelu.



Obrázok 2.2: Zobrazenie jasových transformácií [26]

Jasové transformácie sa rozdeľujú do dvoch skupín, kde prvá skupina je jasová korekcia. Táto korekcia modifikuje jas pixela, kde sa berie do úvahy jeho hodnota a aj poloha v obraze. Druhou skupinou sú šedotónové transformácie, ktoré menia jasovú úroveň bez ohľadu na pozíciu v obraze. Najčastejšie používané šedotónové transformácie sú roztiahnutie jasového intervalu alebo ekvalizácia histogramu (rovnako rozložené jasové úrovne v celej jasovej mierke).[38]

Lokálne predspracovanie

Najjednoduchšia metóda pre potlačenie šumu je filter založený na priemerovaní. Výsledok priemerovania síce potláča šum ale rozostruje hrany v obraze. Ďalšou metódou môže byť mediálny filter, kde sa hodnota pixelu nahrádza hodnotou mediánu z okolitých pixelov. Tento filter dokáže potláčať šum ale aj zvýrazňovať hrany, avšak porušuje ostré rohy a tenké čiary. Najpoužívanejšia je metóda pomocou Gaussovho filtra, ktorý rozostruje obrázok resp. vyhladzuje. Realizuje sa to pomocou masky, ktorú označujeme ako konvolučné jadro.

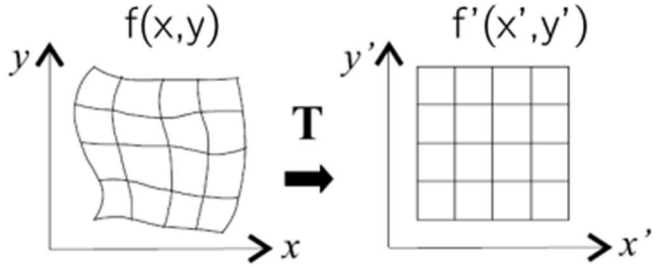
Detekcia hrán sa vykonáva pomocou gradientných operátorov, kde nás zaujíma veľkosť a smer gradientu. To znamená, že ak sa intenzity v danom okolí príliš nelíšia, pravdepodobne sa tam hrana nenachádza. Využívajú sa operátory ako Robertsov, Sobelov, Laplace a podobne. Nevýhodou týchto operátorov je ich závislosť na mierke a citlivosti na šum. [38]

Obrázok 2.3: Využitie Sobelovho operátora[26]

Geometrické transformácie

Transformácia súradníc obrazových bodov a s ňou spojená interpolácia jasových hodnôt. Požíva sa na potlačenie skreslenia obrazu, ktoré vzniklo pri jeho vytvorení alebo pri geometrických zmenách obrazu (veľkosť, rotácia). Máme dve fázy výpočtu transformovaného obrazu[27] :

- trasformácia súradníc obrazových bodov – mapovanie dvojíc (x,y) na dvojice (x‘,y‘),
- interpolácia jasových hodnôt – určenie novej jasovej hodnoty v mieste mimo ortogonálny raster.



Obrázok 2.4: Zobrazenie geometrickej transformácie[27]

2.1.3 Segmentácia

Dá sa chápať ako proces rozdelenia obrazu na viaceré časti, ktoré súvisia s predmetmi alebo oblasťami reálneho sveta. Je kľúčovým prvkom pri analýze obrazu a je prípravou nasledujúcich transformácií. Metódy segmentácie môžeme rozdeliť na základe farieb alebo tvaru (techniky založené na prahovaní a hranách).[28]

Prahovanie

Jedná sa o základnú a najrýchlejšiu techniku segmentácie, pomocou ktorej oddeľujeme oblasti záujmu od pozadia podľa zvolenej hodnoty prahu. Princíp prahovania spočíva v tom, že pokiaľ hodnoty prahu budú nižšie ako nastavená prahová hodnota, bude sa jednať o pozadie. V opačnom prípade sa bude jednať o oblasť záujmu. Prahovanie

môžeme vo všeobecnosti definovať ako transformáciu vstupného obrazu na výstupný binárny obraz nasledovne:

$$g(i,j) = 1 \qquad ak \;\; f(i,j) \geq T\,, \tag{2.1}$$

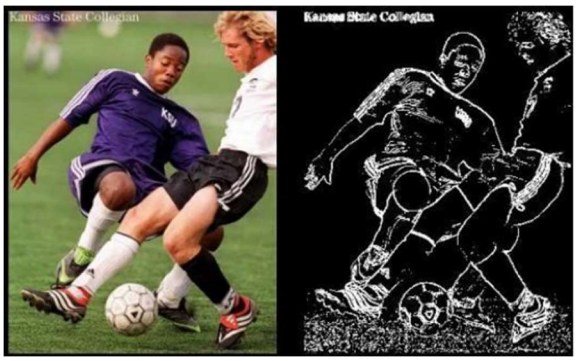
$$g(i,j) = 0 \qquad ak \;\; f(i,j) < T \tag{2.2}$$

kde: g(i,j) – predstavuje buď oblasť záujmu alebo pozadie
f(i,j) – predstavuje pozíciu pixela
T – predstavuje hodnotu prahu

Prahovanie poznáme globálne ale aj lokálne, kde pri globálnom používame rovnaký prah na celom obraze. V praxi sa nepoužíva z dôvodu zmeny jasu vplyvom osvetlenia. Pri lokálnom prahovaní je hodnota prahu určovaná na základe výpočtu v lokálnych častiach obrázka. Nevýhodou pri prahovaní je stanoviť hodnotu prahu. Taktiež je možné ju zistiť automaticky pomocou Otsuovej metódy, ktorá hľadá optimálny prah.[28][39]

Detekcia hrán

Jedná sa o segmentačnú metódu, ktorá funguje tak, že detekuje hrany v obraze. Hrany sú miesta, v ktorých dochádza k náhlym zmenám intenzity jasu a tieto hrany dokážeme detekovať pomocou hranových detektorov, ako napríklad Roberts,Laplace,Sobel a veľa ďalších. V dnešnej dobe je najpoužívanejším Cannyho hranový detektor. Výstupom tohto detektora je binárny obraz, ktorý zvýrazňuje hrany. Takýto výsledný obrázok sa používa v algoritmoch, ktoré sa používajú na detekciu tvarov. Hranové detektory môžu vytvárať pri segmentácií falošné alebo prerušované hranice.[39]



Obrázok 2.5: Segmentácia obrazu založená na detekcií hrán[28]

2.1.4 Detekcia dopravnej značky

V tejto časti sa po predspracovaní a segmentácií vyberajú oblasti, v ktorých by sa dopravná značka mohla nachádzať. Pri segmentácií došlo k zvýrazneniu hrán a na základe nich môžeme určiť tvary. Následne možno určiť, či sa jedná o kruhovú, trojuholníkovú, obdĺžnikovú značku.

Používa sa napríklad Houghova transformácia, ktorá dokáže detekovať špeciálne tvary v obrázku. Najmä detekcia čiar, kružníc a elíps**.**

Ak zistíme, že by sa mohlo jednať o dopravnú značku, postúpime ju klasifikátoru, ktorý už rozhodne o tom, či sa skutočne jedná o dopravnú značku.

2.1.5 Rozpoznanie dopravnej značky(Klasifikácia)

Rozpoznávanie môžeme chápať ako postup resp. detekciu a klasifikáciu, ktoré vedú k rozpoznaniu objektov. Do časti rozpoznania sa dostanú v našom prípade väčšinou len dopravné značky, ktoré obsahujú len oblasti ktoré sú pre nás zaujímavé. Následne sú tieto oblasti triedené pomocou klasifikátora, ktorý dokáže rozlíšiť objekty na základe jeho vstupných vlastností ako napríklad tvar, veľkosť, svetlosť a podobne.

Klasifikácia je vykonávaná pomocou klasifikačných metód resp. nejakých algoritmov. Najviac používanými algoritmami pre klasifikáciu sú algoritmy strojového učenia, teda neurónové siete.

Taktiež je možné použiť mechanizmus podporných vektorov SVM, boosting mechanizmy (AdaBoost), ktoré sú popísané v časti klasifikátory. Samozrejme exituje oveľa viac mechanizmov pre klasifikáciu ako sú napríklad rozhodovacie stromy, porovnávanie so šablónou (template matching).[39]

Template matching je taktiež veľmi obľúbená a používaná metóda, ktorá funguje tak, že porovnáva jednotlivé časti obrazu s predom definovanou šablónou. Výhodou je, že sa porovnávajú obrazy so šablónou a nezávisí na ich otočení a jase. Má to však aj nevýhody, ako je veľká časová náročnosť z dôvodu, že v niektorých prípadoch porovnávame zložité obrazy.[3]

2.2 Detekcia na základe farby

Jedná sa o jednu z najbežnejších metód pri rozpoznávaní dopravných značiek. Značky boli navrhnuté tak, aby boli odlišné od okolitého prostredia. Problémom pri detekcií dopravných značiek je jej zakrytie okolitými predmetmi, poškodenie, dlhé vystavenie priamemu slnku.

Funguje to tak, že najskôr sa pomocou predspracovania a segmentácie nájdu požadované oblasti ROI (Region of intrest), v ktorých sa nachádza naša požadovaná farba. Na výstupe budú teda dáta, ktoré nás zaujímajú resp. požadované farby. Nevýhoda tohto spôsobu detekcie spočíva v tom, že závisí na počasí a tieňoch z okolitých predmetov.

Farba sa v počítačovom svete najčastejšie vyjadruje ako kombinácia troch farebných zložiek: červená, zelená a modrá. Používajú sa teda rôzne farebné modely ako RGB, HSV, L*a*b. Namiesto samostatných farebných modelov môžeme použiť konkrétny model a chroma subsampling.

Model RGB

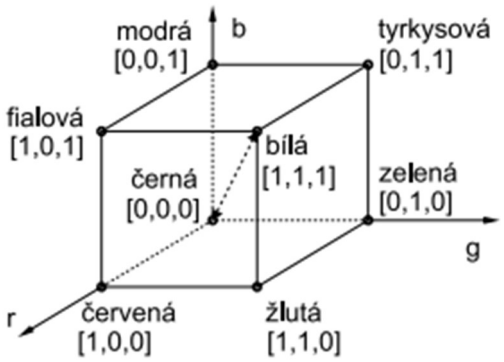
Farba vyjadrená vo farebnom priestore RGB, kde základné zložky pre daný model sú ako plynie z názvu R(červená), G(zelená), B(modrá).

Základnou vlastnosťou tohto modelu je súčtové, aditívne skladanie farieb, teda čím viac farieb sčítame, tým svetlejší bude výsledok. Farby je možné vyjadriť farebným vektorom, ktorého zložky nadobúdajú hodnoty z intervalu (0,1).

Taktiež môžu byť jednotlivé kanály kódované v príslušnej farbe v celočíselnom rozsahu 0 – 255. Používajú sa aj iné kódovania ako napríklad 12 alebo 16 bitov na farebný kanál. Hodnota 0 znamená, že príslušný kanál má čiernu farbu a pre maximálnu hodnotu to znamená, že kanál nadobúda najväčšiu intenzitu.

Napríklad pre kanál R to znamená, že budeme mať najjasnejšiu červenú. Farebný rozsah môže byť reprezentovaný v RGB priestore ako jednotková kocka(obrázok 2.6), kde počiatok súradníc odpovedá čiernej farbe, vo vrcholoch sa nachádzajú základné farby, kde červená [1,0,0], zelená [0,1,0], modrá [0,0,1].

Biela je reprezentovaná vo vrchole [1,1,1] a je zložená zo základných farieb. Farby odtieňa šedej zodpovedajú bodom na diagonále kocky, ktoré spojujú čierny a biely bod. Nevýhodou tohto modelu je, že jednotlivé kanály sú citlivé na intenzitu daného svetla, preto je náročné rozpoznávať inú farbu napríklad žltá sa skladá zo zelenej a červenej. [4]



Obrázok 2.6: Farebný priestor RGB reprezentovaný ako jednotková kocka [4]

Model HSV

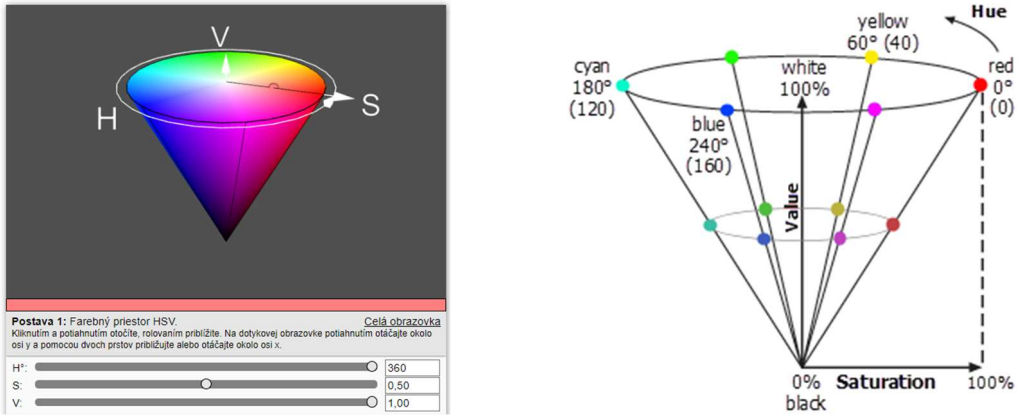
Farba je kódovaná v troch základných vlastnostiach, ktoré sú odtieň H (Hue), sýtosť S(Saturation), jas V (Value). Tento model si môžeme predstaviť ako kužeľ alebo valec so spektrom červenej a až modrej zľava doprava a od stredu k okraju sa intenzita farby zvyšuje. Zdola nahor sa zvyšuje jas.

Odtieň H hovorí o uhle pohľadu na valcový disk, kde odtieň predstavuje farbu a jej hodnota sa pohybuje od 0° do 360°. Obecne sa odtieň označuje názvom farby.

Sýtosť S hovorí o tom, aké množstvo príslušnej farby treba pridať, 100% sýtosť znamená, že sa pridá úplne čistá farba, zatiaľ čo sýtosť 0% nepridá žiadnu farbu a vedie to k odtieňom šedej. Napríklad červená farba s 50% sýtosťou bude farba ružová. Túto skutočnosť môžeme vidieť na obrázku č.2.7.

Jas V sa týka sýtosti S, kde hodnota 100 znamená, že farba bude veľmi jasná. Ak bude mať hodnotu 0 predstavuje to najmenej jasnú farbu, čiže čierna farba.

Nevýhodou tohto modelu je, že pri nízkych hodnotách jasu a sýtosti dochádza k nestabilnosti odtieňu. Taktiež dochádza k oneskoreniu, ktoré je spôsobené prevodom z RGB do HSV. [5]



Obrázok 2.7: Farebný priestor HSV[6][7]

2.3 Detekcia na základe tvaru

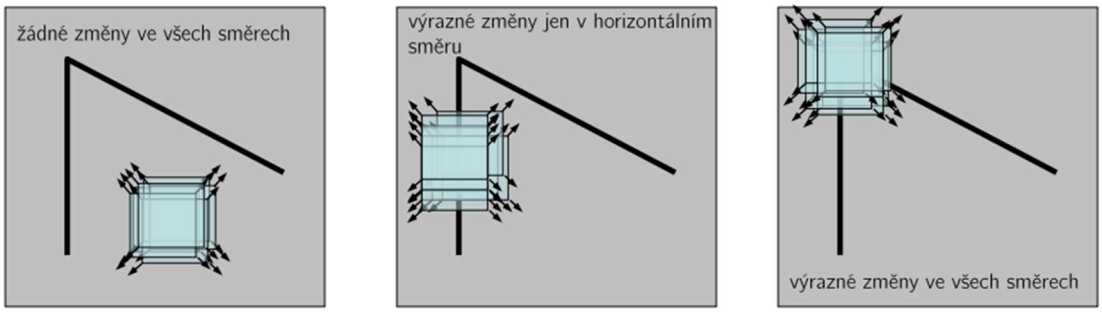
Pre detekciu tvarov sa často využíva Houghova detekčná metóda, ktorá detekuje špeciálne tvary v obrázku. Najmä detekcia čiar, kružníc a elíps. Avšak tieto metódy sú však príliš pomalé pri obrázkoch s vysokým rozlíšením.

Na základe Houghovej detekcie vznikla efektívnejšia metóda, ktorá dokáže rýchlo detekovať tvary a nazýva sa rýchla radiálna symetria.

Pre **detekciu kľúčových bodov** (keypoints) sa používajú hranové detektory, pomocou ktorých dokážeme vytvoriť obrysy tvarov, ktoré sa nachádzajú v obraze. Hrany sa nachádzajú tam, kde je veľký rozdiel v intenzite jasu. Medzi najpoužívanejšie hranové detektory patria Harrisov, Cannyho a veľa ďalších.[8][9]

Obrázok 2.8: Cannyho hranový detektor[9]

Harrisov rohový detektor- Posunutím lokálného okienka dôjde k výrazným jasovým zmenám vo všetkých smeroch v mieste významného bodu(rohu). Výhodou tohto detektoru je to, že je rotačne invariantný, teda natočenie nemá vplyv na vyhľadávanie významných bodov. Taktiež nezávisí na zmene intenzity. Výpočet gradientu je veľmi náchylný na šum, z tohto dôvodu sa pri tomto algoritme používa Gaussova funkcia na odstránenie šumu. [9] [10]



Obrázok 2.9: Harrisov rohový detektor [10]

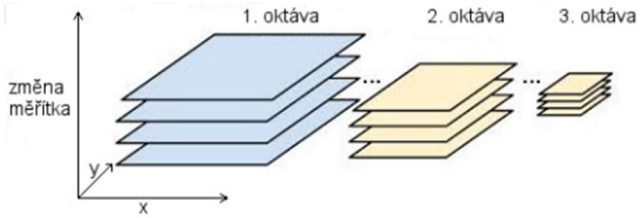
Pre detekciu kľúčových bodov a popis kandidátov resp. dopravných značiek sa používajú rôzne metódy ako napríklad *SIFT,SURF,MSER.*

2.3.1 SIFT

Z angličtiny Scale Invariant Feature Transform. Jedná sa vlastne o kombináciu deskriptoru a detektoru, ktorých úlohou je vyhľadať kľúčové (významné) oblasti s vysokou zmenou intenzity. Tento deskriptor môže pracovať so zmenami zobrazenia ako je rotácia, osvetlenie, škálovanie.

V prvej fáze sa vyhľadávajú potenciálne kľúčové miesta v priestore mierky, ktoré zostávajú stabilné aj počas zmeny mierky. Aby sme dokázali nezávislosť na mierke, musíme vyhľadávať cez viacej mierok. To zabezpečíme pomocou Gaussovskej funkcie, ktorú musíme viackrát aplikovať.

Postupne dostávame obrazy o menšej mierke a následným zostavením týchto obrazov získame jednu oktávu priestoru. [40]



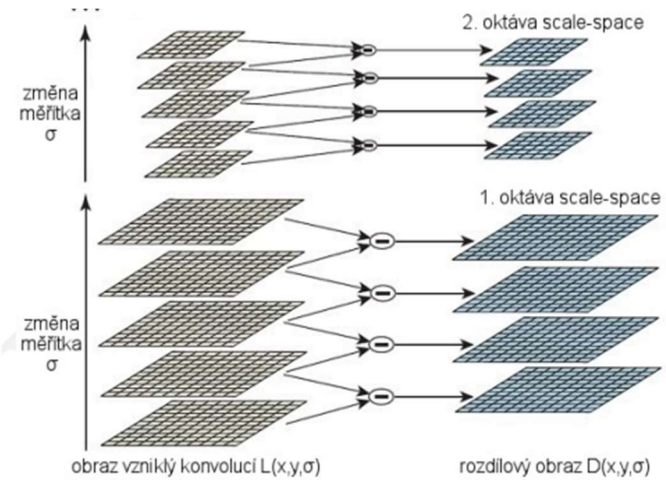
Obrázok 2.10: Zmena mierky[40]

Generuje sa niekoľko oktáv pôvodného obrazu a v každej oktáve je veľkosť obrázka polovičná v porovnaní s predchádzajúcou oktávou. Dochádza k vytvoreniu Gaussovej pyramídy, ktorá sa skladá z obrazov L(x,y,σ), ktoré vznikli konvolúciou Gaussovho jadra G(x,y,σ) o rôznej mierke jadra σ so vstupným obrazom I(x,y)

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.3}$$

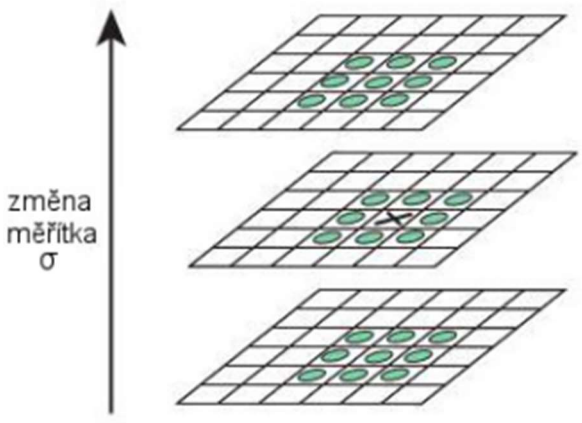
Obrazy L(x,y,σ) v oktávach sú následne využité k výpočtu rozdielu Gaussiánov(Difference of Gaussians-DoG), kde Gaussiánov rozdiel získame odčítaním dvoch obrázkov nasledujúcich za sebou. Je zrejmé, že obrazy v susedných oktávach majú rozdielne mierky k a preto sa rozdiel obrazov D(x,y,σ) (Gaussiánov) vypočíta ako:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.4}$$



Obrázok 2.11: Príklad výpočtu pomocou DoG[40]

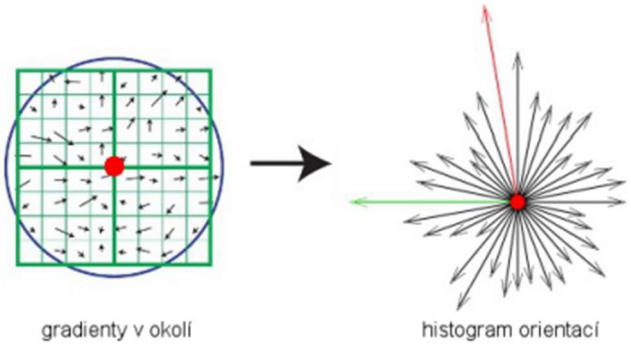
Následne sa určia lokálne maximá a minima v rámci oktávy. Tie získame z rozdielu Gaussianov a priľahlých dvoch úrovní. Porovnávame teda pixely s jeho susedmi v jednej vrstve a následne so susedmi z vrstvy nad a pod. Ak ma nejaký pixel najvyššiu alebo najnižšiu hodnotu došlo k zisteniu, že bod je lokálny extrém a jedná sa o významný bod.[40]



Obrázok 2.12:Detekcia lokálnych extrémov[40]

Takto však získame príliš veľa bodov, ktoré by sme mohli označiť za významné body. Preto prejdeme na druhú fázu resp. na lokalizáciu významných bodov. Teda je potrebné odstrániť tie body, ktoré majú nízky kontrast alebo ležia na hranách.

Ďalšou fázou je priradenie orientácie významným bodom. Zabezpečíme invariantnosť voči rotácií tak, že určíme dominantnú orientáciu. Proces priradenia orientácie prebieha tak, že skúmanému bodu je nájdený mierkovo najbližší obraz L(x,y,σ), ktorý sa nachádza v Gaussovej pyramíde. Týmto postupom zaistíme mierkovú nezávislosť a pre každý bod je potom spočítaná pomocou rozdielu jasových hodnôt pixelov veľkosť gradientu m (x,y) a orientácia gradientu θ (x,y). Následne je skonštruovaný histogram orientácií, ktorý je zostavený z gradientov spočítaných v okolí významného bodu. Histogram nadobúda hodnôt od 0° do 360°, kde tieto hodnoty predstavujú orientáciu gradientu v danom bode. K tejto orientácií sú pričítané hodnoty vážené veľkosťami gradientu a koeficientmi Gaussovho kruhového okna ktoré má stred v skúmanom bode. (obrázok 2.13)[40].



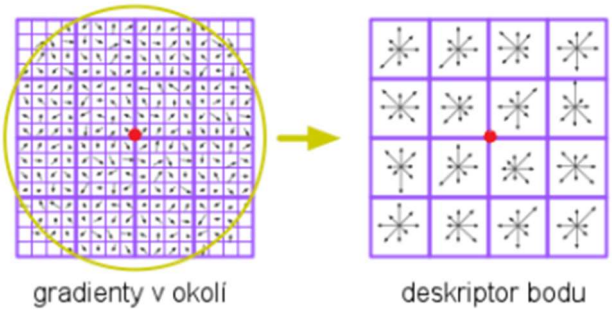
Obrázok 2.13:Gradienty spočítané v okolí významného bodu[40]

Dominantnú orientáciu určíme pomocou globálneho maxima v zostavenom histograme orientácií.

V tejto časti už máme nájdených kandidátov na významné body v obraze a taktiež poznáme ich polohu. Ďalším krokom je použitie deskriptora, ktorého úlohou je popísať

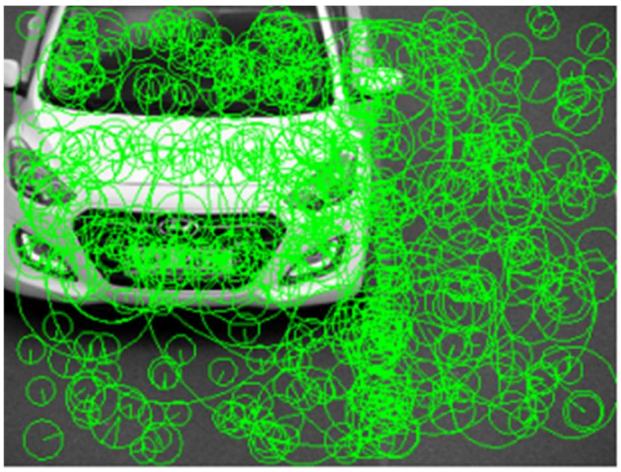
okolie významných bodov tak, aby bol popis nezávislý na geometrických a jasových transformáciach obrazu.

Okolie skúmaného bodu je rozdelené do 8 rovnako veľkých oblastí, kde pre každú oblasť je zostavený histogram orientácií. Následne sa určí dominantná orientácia významného bodu a na základe nej sú histogrami natočené.[40]



Obrázok 2.14: Deskriptor metódy SIFT[40]

Nezávislosť na transformáciách jasovej funkcie funguje tak, že pokiaľ dôjde k zmene osvetlenia, tak sa k jednotlivým pixelom pričíta konštantná hodnota, ktorá sa v gradiente neprejaví. Zmena kontrastu sa vykoná prenásobením hodnôt jasu v obraze určitou konštantou, dôjde len k zmene veľkosti gradientu. Tento vplyv môžeme vyriešiť normalizáciou vektora desktiptora. [11][12]



Obrázok 2.15: Výsledok algoritmu SIFT [14]

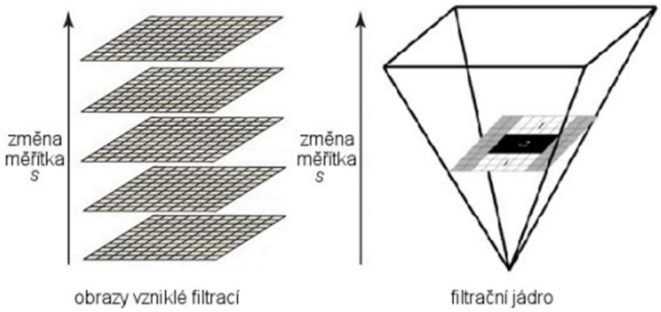
2.3.2 SURF

Z angličtiny Speed Up Robust Features, ktorej cieľom je rýchlejšie vyhľadávanie vlastností obrazu. Algoritmus je založený na rovnakom princípe ako SIFT s tým rozdielom, že je rýchlejší a postupne zväčšuje pôvodný obraz.

K detekcií extrémov nepoužíva rozdiel Gaussiánov (DoG) ale využíva determinant Hessianovej matice, ktorého hodnota sa rýchlo odhadne pomocou box filtra, ktorý

umožňuje rapidné zrýchlenie výpočtu konvolúcie. Z výsledných odoziev na filtráciu je postupne spočítaná hodnota determinantu, čím získame obraz H_{DET} (x,y,σ), kde práve tieto obrazy predstavujú merítkovo nezávislú reprezentáciu vstupného obrazu.

Vyššie mierky sú generované postupným zväčšovaním konvolučnej masky(filtra).



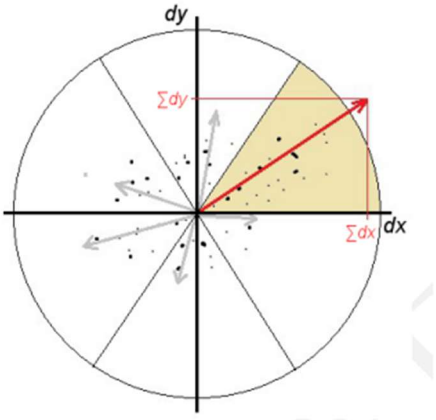
Obrázok 2.16: Znazornenie konštrukcie a aplikácie filtra[40]

Následne je znova potrebná lokalizácia významných bodov, ktorá je rovnaká ako pri SIFT-e to znamená, že pokiaľ má daný bod vo svojom okolí najvyššiu hodnotu, tak je považovaný za významný bod (porovnanie so susednými bodmi v priestore).

Ďalšia fáza je priradenie orientácie významným bodom, v ktorej je rozdiel oproti SIFT-e v tom, že sa berie len dominantný smer gradientu. Pre výpočet sa používa kruhová oblasť v okolí významného bodu.

Orientácia sa neurčuje pomocou histogramu ale pomocou odozvy Haarovej vlny v kruhovom okolí daného bodu.

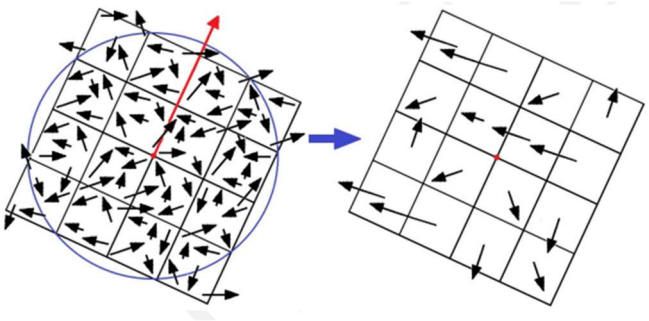
Pre výpočet orientácie sa používa kruhová oblasť v okolí detekovaného významného bodu o polomere 6s, kde s označuje mierku vrstvy. Pre dané body sa spočítajú odozvy na tzv. Haarove vlny v x a y smere. Hodnoty sú následne vážené Gaussovskou funkciou v rámci kruhu a dominantný smer je určený sumovaním cez všetky vypočítané smery a následným zaradením do jedného zo 6 smerov odpovedajúcich 60 stupňovému výseku. V týchto výsekoch vznikajú vektory $\sum d_x$ a $\sum d_y$ a výsek, v ktorom sa nachádza ten najväčší z nich, predstavuje dominantnú orientáciu významného bodu.[14][40]



Obrázok 2.17: Určenie dominantnej orientácie[40]

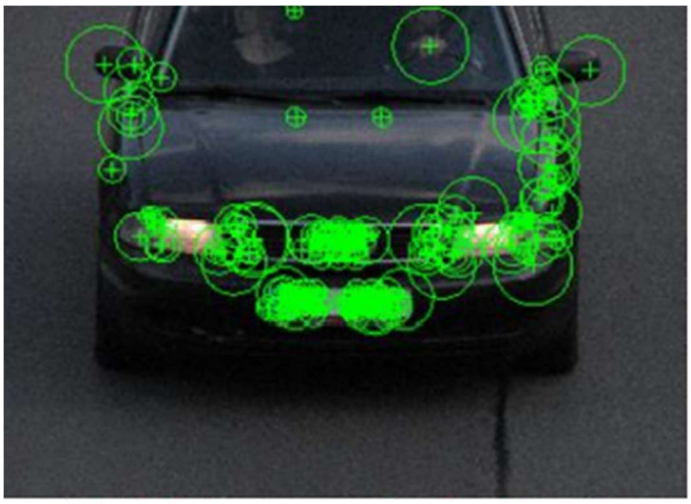
Okolo významného bodu vytvoríme okno (štvorcovú oblasť) o nejakej veľkosti, ktoré má rovnakú rotáciu, aká bola priradená významnému bodu.

Následne túto štvorcovú oblasť rozdelíme na 4x4 podoblastí a rovnakým spôsobom sa určí orientácia.



Obrázok 2.18: Deskriptor metódy SURF[40]

Deskriptor v konečnom dôsledku obsahuje 64 hodnôt pre jeden významný bod. [13][40]



Obrázok 2.19: Výsledok algoritmu SURF [14]

2.3.3 MSER

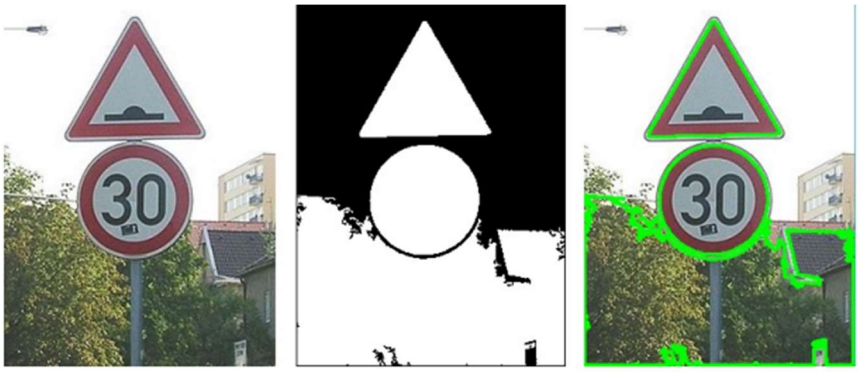
Taktiež nazývané ako aj maximálne stabilné extrémne oblasti. Táto metóda spočíva v detekcií nielen kľúčových bodov, ale celých obrazových štruktúr, resp. detekcia významných oblastí.

Je to teda algoritmus dynamického prahovania šedotónového obrazu, kde máme pre každú oblasť zvolený prah. To znamená, pokiaľ budú pixely pod danou prahovou úrovňou budú čierne, inak budú biele.

Postupným zvyšovaním prahovej hodnoty budú čierne oblasti zodpovedať minimám intenzity a v určitom okamžiku regióny s dvoma lokálnymi minimami sa zlúčia.

Maximálne stabilná extrémna oblasť sa zistí tak, pokiaľ je veľkosť jednej z týchto čiernych oblastí rovnaká (alebo takmer rovnaká) ako na predchádzajúcom obrázku.

Takýmto spôsobom sa oblasti budú zväčšovať a vytvárať oblasti, ktoré budú prepojené(regióny).



Obrázok 2.20: Výsledok algoritmu MSER[40]

Stabilne prepojené regióny predstavujú extrémy, teda všetky oblasti majú buď vyššiu alebo nižšiu intenzitu ako pixely za hranicou oblasti. Vybrané oblasti sú označené rôznym tvarom, napríklad elipsou a sú vyhlásené za významné oblasti MSER.

Jeho výhodou je jeho rýchlosť a stabilita výsledkov v prípade, že je vhodne implementovaný, umožňuje vyhľadanie podobných príznakov v odlišných obrázkoch.

Nevýhodou je, že vplyvom pohybu vzniká šum a v tomto prípade nedosahuje veľmi dobré výsledky. [14][15][40]

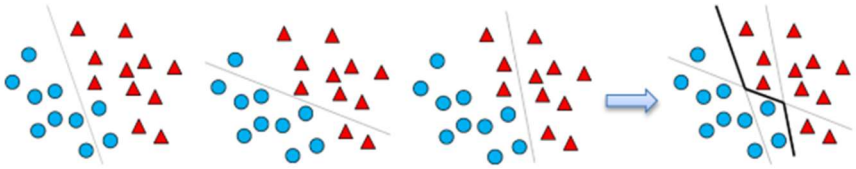
Obrázok 2.21:Výsledok algoritmu MSER [14]

2.4 Klasifikátory

2.4.1 Algoritmus AdaBoost

Jedná sa o jeden z najznámejších algoritmov. Základom tohto algoritmu je, že využíva viacero slabých klasifikátorov, ktoré nemajú príliš veľkú úspešnosť s cieľom vytvorenia jedného silného klasifikátora.

Výsledný klasifikátor sa stane silným kvôli lineárnej kombinácií slabých klasifikátorov. Túto skutočnosť môžeme vidieť na obrázku č.2.22.



Obrázok 2.22: Vytvorený výsledný silný klasifikátor vytvorený z troch slabých klasifikátorov[16]

Nevýhodou je, že učenie je príliš zdĺhavé a nie je ľahké tento natrénovaný algoritmus jednoducho pretrénovať. [16][39]

2.4.2 SVM

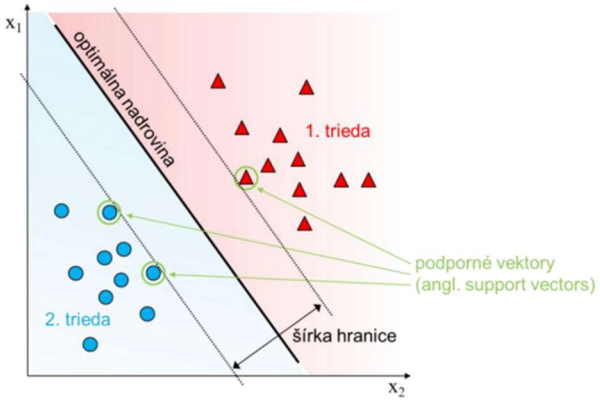
Jedná sa o mechanizmus podporných vektorov (Support Vector Machine). Je to metóda strojového učenia, ktorá je založená na učení sa s učiteľom. Vyvinutá ako binárny klasifikátor, to znamená, že rozdeľuje vstupy na dva výstupy.

Využíva HoG (Histogram of Oriented Gradients) príznaky pre detekciu objektu a následne algoritmus SVM klasifikuje objekt do tried.

HoG je vlastne detektor, ktorý obrázok nepopisuje ako súbor kľúčových bodov, ale ako obsah detekčného posuvného okna určitej veľkosti(výsek obrázka).

Princíp spočíva v tom, že na základe počtu vstupných vzoriek ktorých zaradenie poznáme sa postupne prerátava lineárna funkcia, vďaka ktorej je možné rovinu (optimálnu nadrovinu) rozdeliť na dve časti.

V 2D priestore je to priamka a v 3D rovina. Podstatou je teda nájdenie optimálnej nadroviny, ktorá dokáže maximalizovať šírku hranice medzi triedami. Pomocou podporných vektorov sa určí poloha a smer hranice. [17][39]



Obrázok 2.23: Princíp SVM[39]

2.5 Detekcia pomocou konvolučných neuronových sietí

2.5.1 CNN (Konvolučná neurónová sieť)

Keďže v tejto prací bude implementácia vykonaná pomocou konvolučných neuronových sietí, budem sa tejto časti venovať o niečo viac.

Použitie týchto sieti je vhodnejšie z toho dôvodu, že použitie obyčajných neurónových sietí nie je príliš vhodné pre spracovanie obrázkov a podobne. Výpočet je výpočtovo ale aj pamäťovo náročný, pretože ak máme obrázok o veľkosti 300x300x3, kde sa jedná o šírku, výšku a počet kanálov(RGB), tak pre tento obrázok by bola vstupná trénovacia množina (vstupný vektor) o veľkosti 270 000 x 1.

Každý neurón v prvej skrytej vrstve by musel počítať s 270 000 váh. Z tohto dôvodu sa používajú konvolučné neurónové siete, kde každý neurón je napojený len na časť vstupu (lokálna konektivita).

Taktiež využíva techniku zdieľania parametrov, čo si môžeme vysvetliť tak, že neuróny v jednej vrstve zdieľajú rovnaké hodnoty parametrov.

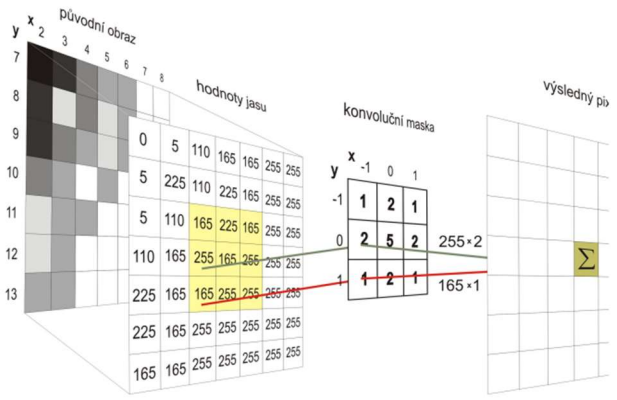
Počet parametrov, ktoré je treba uložiť, sa výrazne znižuje, čo vedie k zníženiu pamäťovej náročnosti. [18][19][41]

Vstupom do konvolučnej siete môže byť obraz, zvuk , text atď. Následne sa používajú nasledovné operácie:

1. Konvolúcia.
2. Nelineárna aktivačná funkcia.
3. Pooling.
4. Klasifikácia.

Konvolúcia

Základná a veľmi často používaná operácia vykonávaná nad obrazom. Konvolúciu si môžeme predstaviť ako priestorové posúvanie prevrátenej masky (konvolučného jadra). Pre každú vzájomnú polohu obrazu a masky je vypočítaný súčet hodnôt pixelov obrazu vážených príslušnými koeficientami masky. Tento súčet určuje výstupnú hodnotu obrazu v danom bode, kde princíp môžeme vidieť na obrázku č.2.24. [20]



Obrázok 2.24: Grafické znázornenie konvolúcie[21]

Pre dvojrozmerný obraz je možné konvolúciu vyjadriť vzťahom:

$$g(x,y) = f(x,y) * h(x,y) = \sum_{i=-S/2}^{S/2} \sum_{j=-R/2}^{R/2} f(x-i, y-j) \cdot h(i,j) \tag{2.5}$$

kde: g(x,y) – výstupný obraz
f(x,y) – vstupný obraz
h(x,y) – konvolučné jadro o rozmeroch RxS [20][21]

Konvolučná vrstva

Špeciálna vrstva, v ktorej nájdeme namiesto neurónu konvolučné jadrá. Jedna konvolučná vrstva obsahuje niekoľko konvolučných jadier z dôvodu, že každé jadro alebo filter detekuje iné príznaky.

Použitím niekoľkých filtrov dokážeme detekovať vo vstupnom obraze napríklad okraje a hrany v rôznych smeroch. Výstupom tejto vrstvy je aktivačná mapa, ktorej hĺbka sa rovná počtu jadier vo vrstve.[19]
Je potrebné vykonať nastavenie týchto parametrov:

- **Počet konvolučných jadier**- čím viac jadier tým je väčšia hĺbka aktivačnej mapy.
- **Veľkosť filtra** – môžu byť rôznych veľkostí ale najčastejšie používané sú 3x3, 5x5,7x7.
- **Krok konvolúcie** - o koľko je jadro posunuté.

Problém nastáva vtedy, pokiaľ filter presiahne hranice vstupného obrazu. K vyriešeniu tohto problému exitujú tri riešenia. Prvým riešením je priloženie filtra k vstupnému obrazu tak, aby pri posúvaní nedošlo k presiahnutiu hranice.

Z toho vyplýva, že rozmer výstupu bude menší. Druhým riešením je pridanie núl za okraje vstupného obrázka, čím umožníme, aby filter mohol presahovať za hranice vstupu. Týmto spôsobom nedôjde k zmenšeniu výstupnej aktivačnej mapy, avšak môže dôjsť k detekcií neexistujúcich príznakov. Tretím riešením je skopírovanie okrajových hodnôt vstupu a následne tieto hodnoty pridáme na okraje resp. rozšírime vstup.[19]

Nelineárna aktivačná funckia

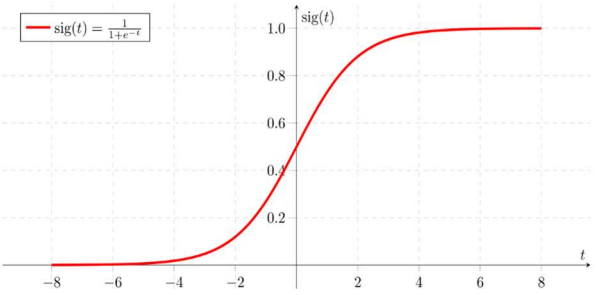
Na každú vytvorenú aktivačnú mapu je následne aplikovaná nelineárna aktivačná funkcia. Aktivačná funkcia rozhoduje o tom, či sa neurón aktivuje. V dnešnej dobe existuje množstvo aktivačných funkcií ako napríklad:

- **Sigmoida**

  Je to monotónne rastúca, hladká a ohraničená funkcia, ktorej tvar sa nachádza na obrázku č. 2.25.

  Na jej výstupe sa nachádza číslo v intervale <0,1>. Problém nastáva vtedy, keď dôjde k saturácií aktivácie neurónov, teda gradient môže dosahovať nulové hodnoty. Ďalším problémom je, že výstup nie je centrovaný v nule.

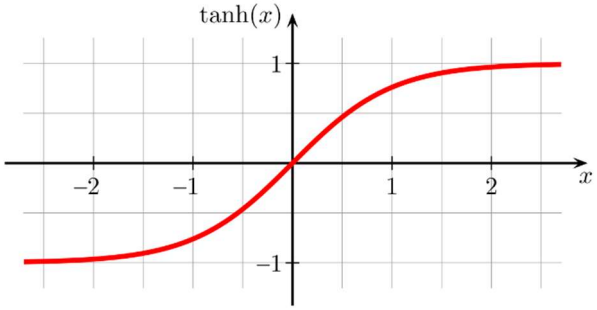
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$



Obrázok 2.25:Aktivačná funkcia sigmoid [22]

- **Tanh**

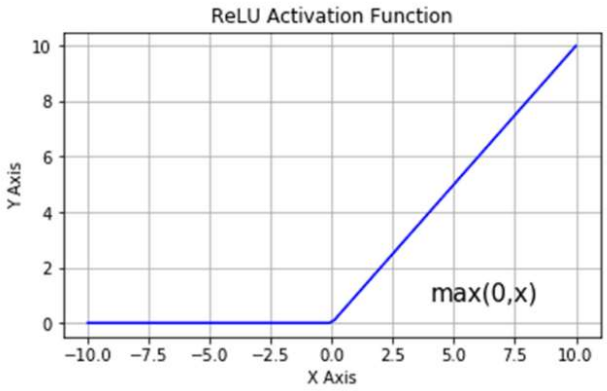
  Taktiež nazývaný aj ako hyperbolický tangens a jeho výhodou je, že je centrovaný v nule. Na jeho výstupe sa nachádza číslo z intervalu <-1,1>. Problém rovnaký ako pri sigmoide. Jeho predpis je tanh(x).



Obrázok 2.26: Aktivačná funkcia hyperbolický tangens [23]

- **ReLU**
  Jedná z najpoužívanejších aktivačných funkcií v dnešnej dobe. Oproti predchádzajúcim aktivačným funkciám je výpočtovo nenáročná.
  Na výstupe nie je obmedzená, takže nevzniká problém so saturáciou a gradient nedosahuje nulové hodnoty. Avšak môže nastať stav, kde sa neurón neaktivuje (dying ReLU).

$$f(x) = \max(0, x) \tag{2.7}$$

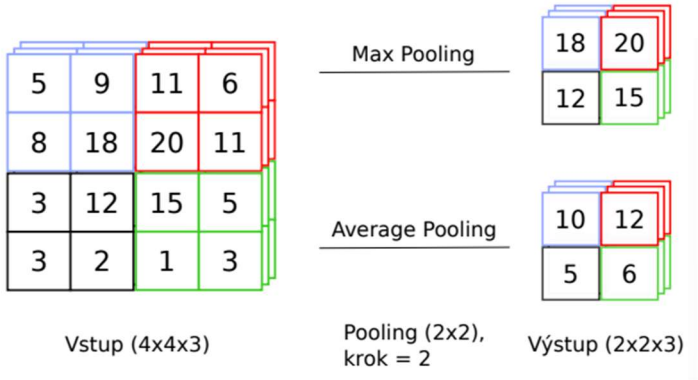


Obrázok 2.27: Aktivačná funkcia ReLU [24]

Poolingová vrstva

Je ďalšou špeciálnou vrstvou v konvolučných sieťach, ktorá zabezpečuje zmenšovanie rozmerov aktivačnej mapy s cieľom zmenšiť pamäťovú a výpočtovú náročnosť. Táto vrstva v porovnaní s konvolučnou pracuje nezávisle na každej aktivačnej mape samostatne a teda neovplyvňuje hĺbku, ale len šírku a výšku. Jadro sa posúva na základe rovnakého princípu ako v konvolučnej vrstve a nad každou časťou vykoná pooling.

K zmenšeniu rozmerov možno použiť dve metódy max pooling alebo average pooling. Max pooling funguje tak, že vyberie maximálne hodnoty z príslušnej oblasti. Average pooling(priemerovanie hodnôt) funguje tak, že vykoná priemer pixelov z danej časti. Niekedy sa parametre poolingu volia dynamicky a to vtedy, pokiaľ máme vstup rôznych veľkostí a požadujeme jednotné rozmery. V takomto prípade je umiestnené na začiatku architektúry siete.[19]



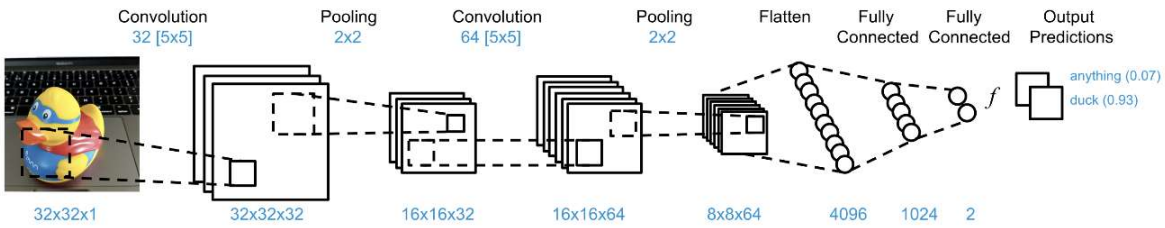
Obrázok 2.28: Príklad Poolingových operácií[19]

Plne prepojená vrstva

Jedná sa o vrstvu, v ktorej sú všetky neuróny prepojené s každým neurónom nasledujúcej vrstvy. Je to typická architektúra neurónových sietí a v klasifikačných konvolučných neurónových sieťach ju nájdeme tiež.

Z väčšej časti býva umiestnená ako výstupná vrstva siete. Týchto plne prepojených vrství môže byť niekoľko. V prvej z nich dôjde k spojeniu výstupných príznakových máp do jednej spoločnej vrstvy (jednorozmerný vektor).

Tento vektor je následne použitý ako vstup normálnej neurónovej siete. Kde jej úlohou je klasifikácia objektov do jednotlivých tried.[41]



Obrázok 2.29:Schéma konvolučnej neuronovej siete [25]

3. HARDWAROVÁ ČASŤ

Ako už z predchádzajúcich častí vieme, analýza obrazu je základným kameňom pri rozpoznávaní dopravných značiek.

3.1 Výpočtová časť

Pre programovanie algoritmov rozpoznávania a ich výpočty. Vyberali sme si z troch možností :

1. Raspberry Pi 4B.
2. STM32.
3. NVIDIA Jetson.

Pre moju aplikáciu som vybral Raspberry Pi 4B z dôvodu, že je to stredná cesta z uvedených možností. Taktiež postačuje z hľadiska výkonu a pamäte (plnohodnotný počítač). Nevýhodou je, že neobsahuje grafiku a teda všetky výpočty algoritmov budú vykonané len na CPU. STM32 som nevyberal z dôvodu, že je to v podstate mikroprocesor a bolo by potrebné pridať periferie a celé riadenie. NVIDIA Jetson by bola z môjho pohľadu lepšou možnosťou ako Raspberry z dôvodu, že je vyhotovená na detekciu a rozpoznávanie a taktiež obsahuje aj GPU, takže výpočty by sa vykonávali priamo na grafike a tým by sa zrýchlil celý proces rozponávania a detekcie značiek.

Rozdiel medzi Raspberry a NVIDIA Jetson je hlavne v rýchlosti vykonávania matematických operácií, kde NVIDIA dokáže robiť výpočty približne 27x rýchlejšie.

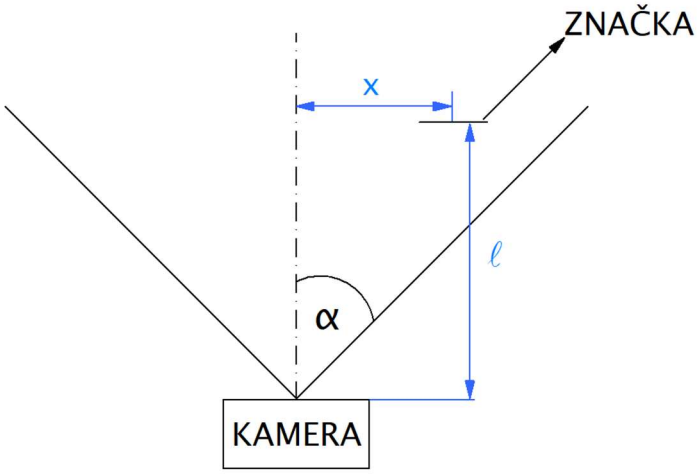
Avšak z dôvodu že je príliš drahá a Raspberry som už vlastnil som sa rozhodol túto možnosť tiež vylúčiť.

Obrázok 3.1: Raspberry PI 4B[42]

3.2 Snímacia časť

S pomocou kamery získame farebný snímok. Našim cieľom je získať najlepšie a najpresnejšie výsledky. Kamera musí byť teda dostatočne kvalitná, aby bolo možné rozpoznať dopravnú značku. Avšak vysoké rozlíšenie spomaľuje proces a z toho dôvodu som otestoval 2 kamery.

Kameru som volil na základe požadovaného FOV, ktoré bolo potrebné vypočítať na základe mojich predpokladov. Tieto predpoklady zahŕňali, na akú vzdialenosť chcem značku detekovať a ako blízko pred autom ešte požadujem rozpoznanie značky a či na určitom rozlíšení bude značka reprezentovaná dostatočným množstvom pixelov.
Výpočet pre výber kamery:



$$tg(\alpha) = \frac{x}{l} \quad (3.2)$$

$$\alpha = \text{atan}\left(\frac{x}{l}\right) \quad (3.3)$$

$$FOV_H = 2 \cdot \alpha \quad (3.1)$$

x – odsadenie značky
l – vzdialenosť do akej chcem značku detekovať

V mojom prípade:
x = 2,5 m
l = 2 m

$\alpha = \text{atan}\left(\frac{x}{l}\right) = \text{atan}\left(\frac{2{,}5}{2}\right) = 51{,}34°$
$FOV_H = 2 \cdot \alpha = 2 \cdot 51{,}34° = 102{,}68°$ => v horizontálním smere

Prvá kamera, ktorá bola testovaná, je kamera OV5647 s maximálnym rozlíšením 2592 x 1944, ktorú som mal len požičanú. Avšak ako sa dalo očakávať, táto kamera nie je až tak kvalitná. Jedná sa len o 5 Mpx kameru a na Full HD rozlíšení dokázala snímať na 30 FPS. Taktiež nevyhovovala na základe mojich výpočtov potrebných pre snímanie obrazu z dôvodu, že FOV je 54° v horizontálnom smere a požadovaných je 100°. Vo vertikálnom smere je FOV 41° a ohnisková vzdialenosť je fixná a odpovedá 3,6 mm.

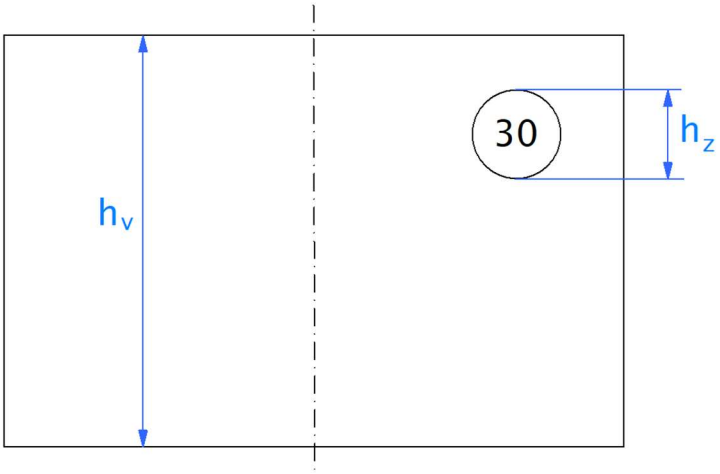
Na základe predchádzajúcich výpočtov som ako kameru zvolil:

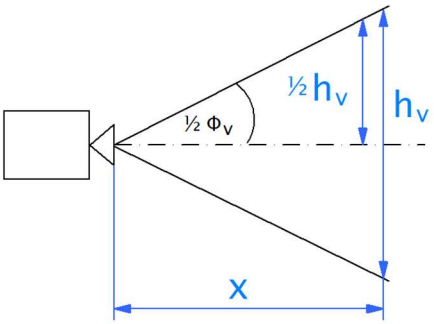
Arducam NOIR 8MP SONY IMX 219

FOV tejto kamery v horizontálnom smere je 100° a vo vertikálnom 68°, čo odpovedá požadovaným výpočtom. Ohnisková vzdialenosť je nastaviteľná, kamera sa dá doostriť. Táto kamera taktiež obsahuje odmontovateľný filter s nočným videním, rôzne nastavenia rozlíšenia až do 3280 x 2464. Ako z názvu vyplýva jedna sa senzor SONY IMX 219, ktorý má rozlíšenie 8 Mpx(Megapixel). Kamera pri maximálnom rozlíšení dokáže snímať do 15 FPS.

Obrázok 3.2: Kamera ARDUCAM NOIR[42]

Výpočet reprezentácie počtu pixelov:





$\Phi_H = 100°$ - z datasheetu vybranej kamery [42]
$\Phi_V = 68°$ - z datasheetu vybranej kamery [42]
x =15m - vzdialenosť kamery a značky
$h_V = ?$ - výška záberu kamery
$h_H = ?$ - šírka záberu kamery

$$tg\left(\frac{\Phi_v}{2}\right) = \frac{\frac{h_v}{2}}{x} \tag{3.4}$$

$$\frac{h_v}{2} = tg\left(\frac{\Phi_v}{2}\right) \cdot x \tag{3.5}$$

$$h_v = 2 \cdot tg\left(\frac{\Phi_v}{2}\right) \cdot x \tag{3.6}$$

$h_v = 20{,}23\ m$

Rozmer kruhovej značky
h_Z=0,6 m - zo stránky s rozmermi dopravných značiek na cestách [43]

Výpočet počtu pixelov značky na 15 m s maximálnym rozlíšením

$h_Z = 0{,}6\ m \quad . \quad . \quad . \quad . \quad . \quad h_{ZPIX} = ?$
$h_V = 20{,}23\ m \quad . \quad . \quad . \quad . \quad . \quad h_{VPIX} = 2464\ px$

$$\frac{2464}{x} = \frac{20{,}23}{0{,}6}$$
$$x = 73{,}01\ px$$

Značka bude mať na 15 metroch 73,01 px vo vertikálnom smere.

V horizontálnom smere je výpočet rovnaký s tým rozdielom, že počítame s Φ_H = 100°, potom je výpočet nasledovný:
$h_H = 35{,}75\ m$

$h_Z = 0{,}6\ m \quad . \quad . \quad . \quad . \quad . \quad h_{ZPIX} = ?$
$h_V = 35{,}75\ m \quad . \quad . \quad . \quad . \quad . \quad h_{VPIX} = 3280\ px$

$$\frac{3280}{x} = \frac{35{,}75}{0{,}6}$$
$$x = 55{,}05\ px$$

Môžeme si všimnúť, že kruhová značka je reprezentovaná rôznym počtom pixelov v horizontálnom a vertikálnom smere to je spôsobené tým, že nie sú v pomere zorné uhly (FOV) s rozlíšením.

3.3 Zobrazovacia časť

Ďalšou časťou hardwaru je displej, na ktorom sa budú rozpoznané dopravné značky zobrazovať. Samotný displej som vyberal už na základe toho, aby to nebolo až príliš rozmerné a mohlo to byť aplikované priamo v aute.
Displej som vybral:

WAVESHARE 4,3 LCD DISPLEJ 800x480

Jedná sa teda o podsvietený LCD displej s uhlopriečkou 4.3 palca ktorý je dotykový. Displej je priamo kompatibilný s rozhraním Raspberry. Samotné rozlíšenie displeja je

800x480, kde PPI vyšlo 217. Čo je v pomere na rozlíšenie displeja v celku kvalitné. PPI je jednotka používaná k určeniu rozlíšenia zobrazovacieho zariadenia.

Obrázok 3.3: Displej WAVESHARE[42]

3.4 Napájacia časť

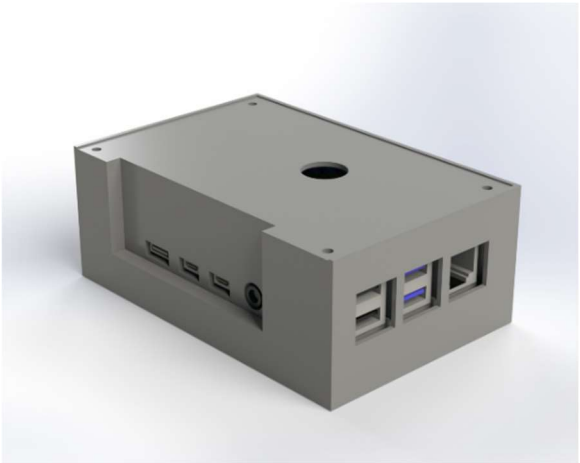
Aby Raspberry správne fungovalo, je potrebné vyriešiť jeho napájanie, ktoré je v mojom prípade zabezpečené pomocou USB-C konektora s maximálnym príkonom 15 W. V aute je táto skutočnosť zabezpečená pomocou nabíjačky do auto zásuvky, ktorý môže dodať maximálny výkon 12 W.

3.5 Návrh krabičky

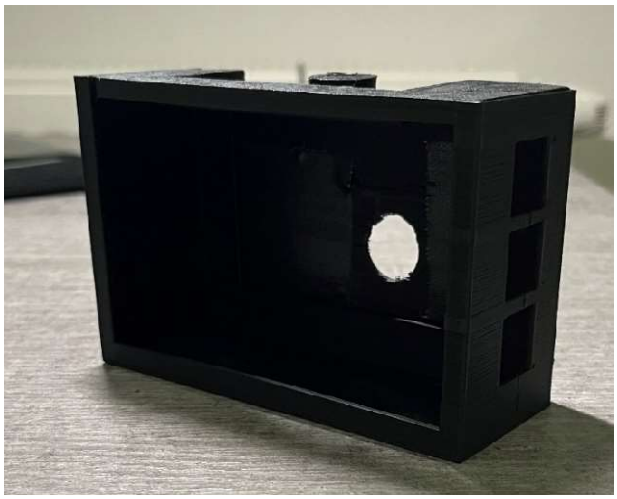
Po výbere jednotlivých komponentov v hardwarovej časti musíme prejsť k návrhu krabičky, v ktorej budú tieto komponenty uložené a následne mohli byť použité v aute.

Krabička

Bola navrhnutá tak, aby bolo možné komponenty uložené v krabičke jednoducho demontovať a následne znova zmontovať. Samotná krabička pozostáva z dvoch častí, ktoré sú zmontovateľné, aby bolo možná rýchla výmena komponentov v prípade nejakej poruchy. Časti krabičky sú zobrazené v prílohách v podobe výrobných výkresov.



Obrázok 3.4 Vizualizácia krabičky v programe SOLIDWORKS



Obrázok 3.5: Reálna fotografia krabičky

Držiak kamery

Samotný držiak kamery som nenavrhoval ale použil som držiak z auto kamery ktorý som mal doma. Tento držiak je vyhotovený z častí ako prísavka a nastaviteľný kĺb so ¼ colovým závitom na ktorom bude namontovaná krabička. Držiak je znázornený na obrázku č.3.6.

Obrázok 3.6: Držiak kamery

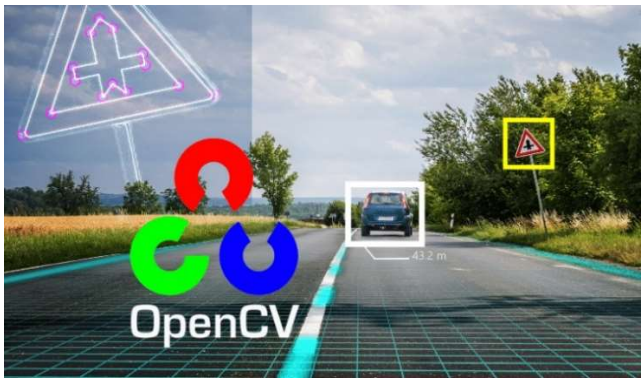
4. SOFTWAROVÁ ČASŤ

V tejto časti som prešiel k samotnému programovaniu, kde som sa snažil postupovať od nejakých základov rozpoznávania dopravných značiek až po rozpoznávanie pomocou spomínaných konvolučných neurónových sietí.

Programy budú implementované v programovacom jazyku Python, kde budem používať pomocné knižnice ako Open CV, PyTorch a veľa ďalších.

Knižnica OpenCV

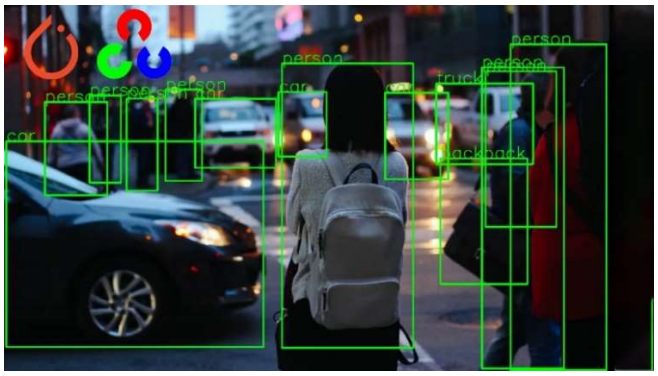
Jedná sa o open-source softwarová knižnica, ktorá obsahuje veľké množstvo programovacích funkcií, ktoré sú používané v počítačovom videní, strojovom učení a v reálnom čase. Knižnica teda obsahuje algoritmy, ktoré je možné použiť napríklad aj pre rozpoznávanie objektov a ich klasifikáciu.



Obrázok 4.1: Open CV[45]

Knižnica PyTorch

Taktiež je to open-source framework pre deep-learning, ktorý je v podstate jednoduchý na použitie. Používa sa na vytváranie modelov, napríklad konvolučných neurónových sietí, ktoré sa používajú v aplikáciách pre rozpoznávanie a klasifikáciu objektov. Obsahuje podporu GPU, ako napríklad na platforme CUDA pre grafické karty NVIDIA, kde je možné vykonávať výpočty spojené s rozpoznávaním a klasifikáciou priamo na grafike, čo je podstatne rýchlejšie ako na CPU.

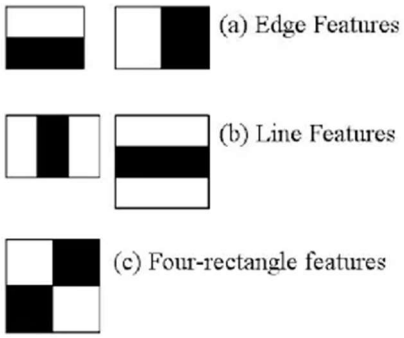


Obrázok 4.2: Využitie knižnice PyTorch[45]

4.1 Rozpoznávanie dopravných značiek pomocou Haar Cascade Classifiers

Haar Cascade je algoritmus, ktorý dokáže detekovať objekty v obrázkoch nezávisle na ich mierke a umiestení obrázku. Tento algoritmus bol navrhnutý v roku 2001 a bol použitý pre rozpoznávanie v reálnom čase, kde haar cascade detektor bol natrénovaný na detekciu rôznych objektov ako napríklad tvár, autá, budovy a podobne. [30]

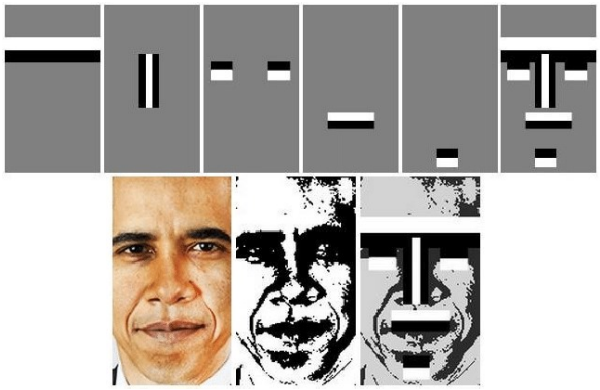
Pre detekciu objektov sa využívajú haarové príznaky, kde sa tieto príznaky získajú pomocou čiernych a bielych obdĺžnikových polí, ktoré fungujú ako konvolučné jadrá. Príznaky sú v podstate konkrétne hodnoty získané odčítaním súčtu pixelov pod bielym polom od súčtu pixelov pod čiernym polom. Haarove príznaky môžu mať rôzne tvary, pre detekciu hrán, čiar atď. Znázornenie haarových príznakov je na obrázku č.4.3.[29]



Obrázok 4.3: Typy haarových príznakov [29]

Avšak tieto príznaky by bolo zložité učiť pri veľkých rozmeroch obrázka, z toho dôvodu boli vyvinuté integrálne obrazy.

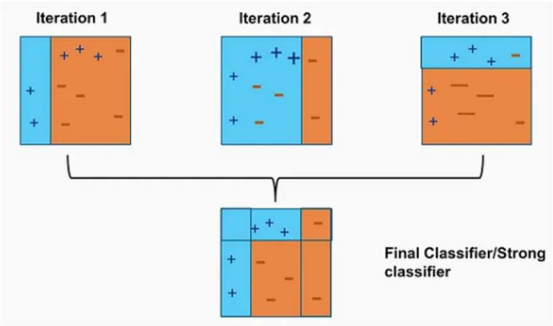
Integrálne obrazy zrýchľujú výpočet haarových príznakov, kde namiesto počítanie každého pixelu vytvára podobdĺžniky a vytvorí pole referencii pre každý z podobĺžnikov. Tieto obdĺžniky sú následne použité pre výpočet haarových príznakov, ktorých môže byť však príliš veľa.[29]



Obrázok 4.4: Ukážka haarových príznakov na obrázku[29]

Adaboost algoritmus zabezpečuje, že vyberie len tie najlepšie príznaky a následne tie klasifikátor použije. Jedná sa o klasifikátor, ktorý som popísal v kapitole 2.4.1.

Slabý klasifikátor sa vytvára presúvaním detekčného okna cez obrázok a výpočtom haarových príznakov pre každú pod sekciu obrázka. Vzniknutý rozdiel sa porovnáva s naučeným prahom, ktorým oddeľujeme objekty ktoré nás zaujímajú. Aby sme dosiahli väčšiu presnosť je potrebné veľké množstvo príznakov pre vytvorenie silného klasifikátora. [29]



Obrázok 4.5: Adaboost algoritmus[29]

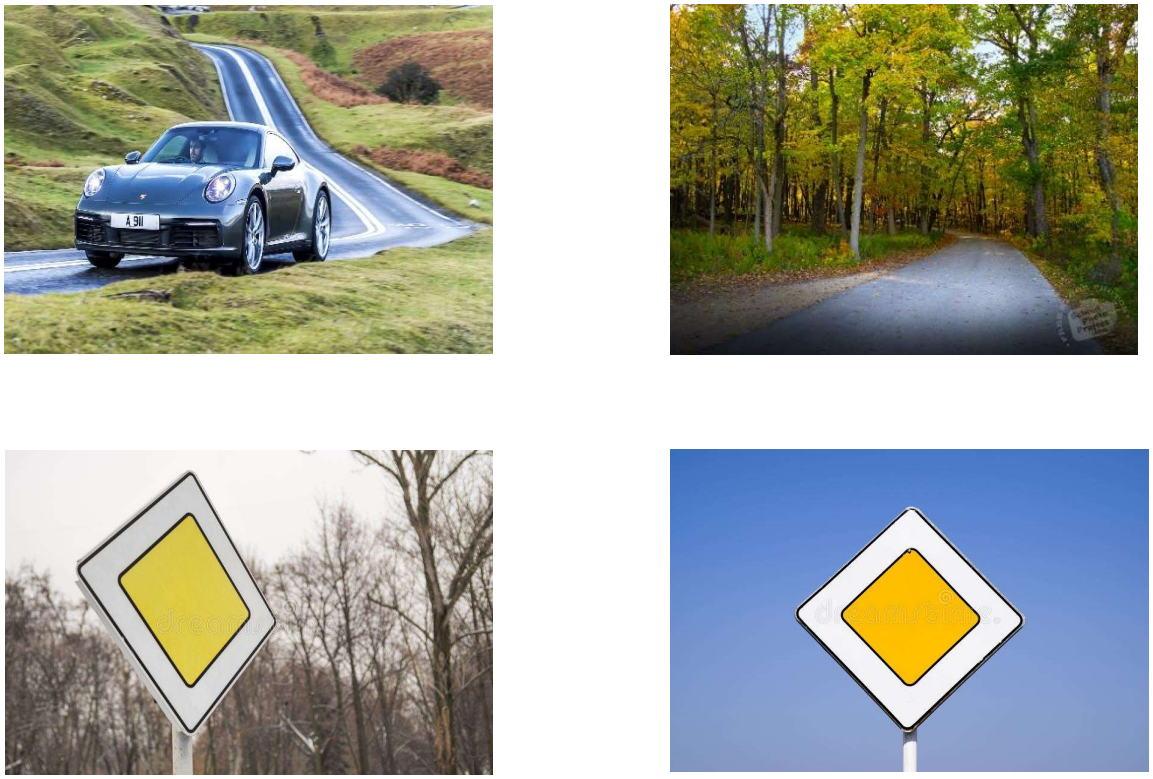
Kaskádový klasifikátor je vytvorený z určitého počtu stageov, kde každý stage je vytvorený ako súbor slabých klasifikátorov.

Rozdelenie objektov do daných tried zaisťuje kaskádový klasifikátor, ktorý je trénovaný na veľkom množstve pozitívnych obrázkov (požadovaný objekt) a negatívnych obrázkov (objekty, ktoré nás nezaujímajú), v ktorom sa využívajú rôzne haarové príznaky.[29]

V mojom prípade sa jednalo o prvotný pokus o implementáciu rozpoznávania dopravných značiek, kde som sa snažil o rozpoznávanie troch typov značiek (Stopka, Daj prednosť v jazde, hlavná cesta).

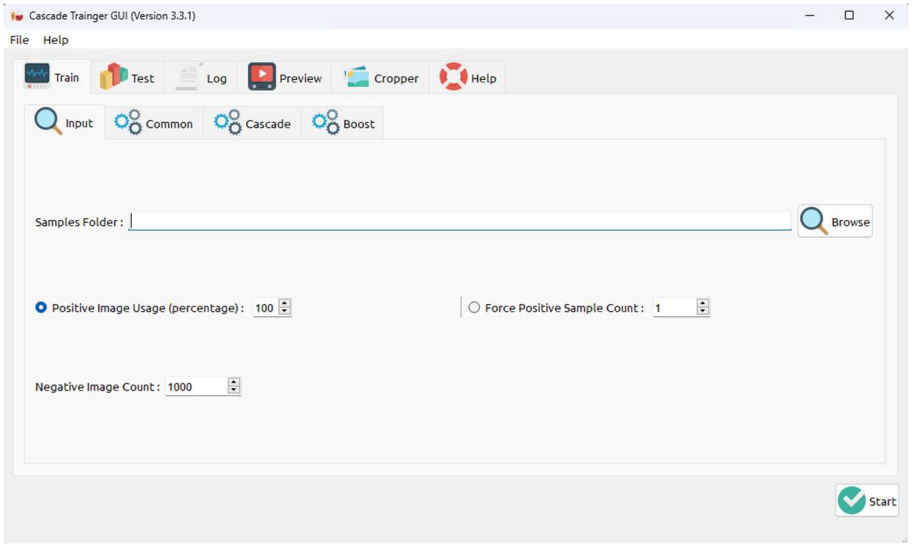
Začal som natrénovaním klasifikátorov objektov, ktoré som ukladal do XML súborov. Najskôr bolo potrebné získať veľké množstvo fotiek príslušných značiek (pozitívne obrázky) a taktiež obrázkov, ktoré sa značkám nepodobajú (negatívne obrázky). Pre trénovanie som používal už vytvorenú aplikáciu Cascade-Trainer-GUI. Dataset pozitivných a negatívnych fotiek som vytváral pomocou google obrázkov, google máp, kde som používal iCrawler, ktorému som zadal počet obrázkov, ktoré má nájsť a ich názvy.

Napríklad pre trénovanie značky hlavná cesta bolo použitých 967 negatívnych obrázkov, ktoré pozostávali z budov, áut, ľudí, bicyklov, stromov a z 3700 pozitívnych obrázkov.



Obrázok 4.6: Negatívne a pozitívne obrázky pre trénovanie hlavnej cesty

V tejto aplikácií stačilo zadať cesty k pozitívnym a negatívnym obrázkom značiek a nastaviť počet stageov a či sa majú použiť všetky typy haarových príznakov. Ďalej bolo potrebné nastaviť parametre Adaboost algoritmu, ktoré som nechal v podstate predvolené z dôvodu, že sa jednalo o odporúčané nastavenie.



Obrázok 4.7: Aplikácia trénovania Haarových príznakov

Po dotrénovaní sa vytvoril XML súbor, ktorý som následne načítaval v programe. Ďalším krokom bolo vytvorenie rozpoznávania real-time. Použil som knižnicu OpenCV, ktorá prevádza obraz z kamery, ktorý pozostáva z troch kanálov RGB na kanály BGR, avšak na účel rozpoznávania je v tomto prípade výhodnejšie konvertovanie do odtieňa šedej z dôvodu jednoduchšieho spracovania a výpočtov, pretože používame prakticky len jeden kanál.

```
import cv2

stop_classifier=cv2.CascadeClassifier('cascade_stop_sign.xml')

##Real time
vid=cv2.VideoCapture(0)

while(ret == True):
      ret,frame=vid.read()

#Uloženie aj originálnych snímkov RGB

      img_rgb = frame

#prevedenie na odtieň šedej

      img_gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
```

Po konvertovaní som sa pokúsil nájsť príznaky real time v obrázkoch s dopravnými značkami, s použitím načítaného objektu (stopky), ktorý obsahuje natrénované príznaky a funkciou detectMultiScale.

Táto funkcia mi pomáha nájsť príznaky resp. oblasti, v ktorých sa dopravná značka nachádza. Funkcia ma niekoľko nastaviteľných parametrov

- scaleFactor – tento parameter určuje, o koľko sa zmenší veľkosť obrázka (pokiaľ vstupný obraz bude iných rozmerov ako ten trénovaný).
  Môj model má počas tréningu pevne dané rozmery obrázka, to znamená, že takáto veľkosť značky sa bude detekovať v reálnych snímkach. Avšak zmenou mierky vstupného obrazu môže dôjsť, že sa veľkosť značky zmenší. Z toho dôvodu je vhodné voliť tento parameter do hodnoty 1.05, síce algoritmus pracuje podstatne pomalšie, ale je väčšia šanca, že nájde značku aj pri iných rozmeroch vstupného obrázka,
- minNeighbors – tento parameter určuje, koľko by mal mať každý kandidát susedov. Čím vyššia hodnota spôsobí menej detekcí, ale vyššiu kvalitu detekcie,
- minSize – minimálna veľkosť detekovaných objektov (objekty menšie ako hodnota sú ignorované),
- maxSize – maximálna veľkosť detekovaných objektovb (objekty väčšie ako hodnota sú ignorované).

```
  found = stop_data.detectMultiScale(img_gray,1.05,5)
```

Táto funkcia vracia štyri hodnoty, x-ovú súradnicu, y-ovú súradnicu, šírku(w) a výšku(h) detekovaných príznakov značky. Následne na základe týchto štyroch hodnôt som nakreslil obdĺžnik okolo dopravnej značky.

```
  # z dôvodu aby som nevykonaval nič pokiaľ nenájde žiadnu značku
  sum_found=len(found)

  If(sum_found !=0):

  #vytvorenie obdĺžnika okolo zdetekovanej značky

    for (x,y,width,height) in found:

        cv2.rectangle(img_rgb,(x,y),(x+height),(y+width),
                      (0,255,0),2)
        cv2.putText(img_rgb,'STOP',(x,y-10),
                    cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,255,0),1)

  #Zobrazenie v real-time
  cv2.imshow('frame',frame)

  if cv2.weitKey(1) & 0xFF == ord('q'):
        break

vid.release()
cv2.destroyAllWindows()
```

Výsledok detekcie sa nachádza na obrázku č.4.9. Avšak ako som spomínal už na začiatku, jednalo sa o prvotný pokus rozpoznávania dopravných značiek, keďže mojím cieľom je vyskúšanie rozpoznávania dopravných značiek pomocou rôznych metód. Túto metódu by som zhodnotil, že v mojom prípade táto metóda nefungovala príliš spoľahlivo z dôvodu, že v real-time som detekoval niektoré značky aj keď by som nemal (obrázok č.4.8). Myslím si, že to bolo spôsobené nedostatočným počtom pozitívnych a negatívnych obrázkov jednotlivých značiek.

Obrázok 4.8: Nesprávne zdetekovaná značka

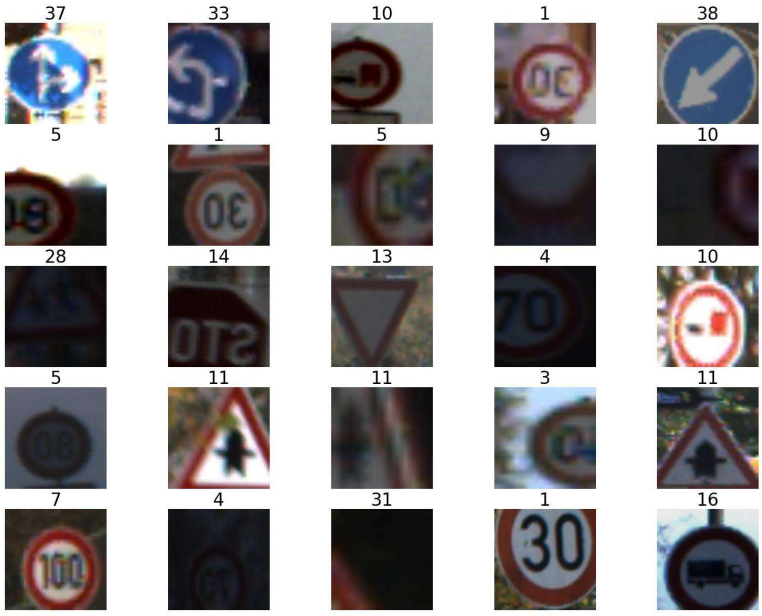
Obrázok 4.9: Výsledok rozpoznania STOPKY v real-time a na fotke

4.2 Vytvorenie vlastnej konvolučnej neuronovej siete

V tejto časti som využíval knižnicu PyTorch a jej funkcie. Postup vytvorenia konvolučnej siete bol vytvorený podľa príkladu, ktorý sa nachádza priamo v oficiálnej dokumentácií.

Prvým krokom bolo stiahnutie datasetu, na čo som použil knižnicu *torchvision,* ktorá obsahuje niekoľko vytvorených datasetov a pomocných funkcií.

Dataset, ktorý budem používať je GTSRB (German Traffic Sign Recognition Benchmark). Tento dataset obahuje 43 tried, kde každá trieda predstavuje iný druh dopravného značenia. V každej triede sa nachádza niekoľko príslušných obrázkov, celkovo má dataset cez 50 000 obrázkov. Tento dataset obsahuje len výrezy značiek.



Obrázok 4.10: Ukážka datasetu GTSRB a prislušných tried pre jednotlivé značky

Po importovaní knižníc som použil funkciu *transform.Compose* ktorá mi umožní preformátovanie obrázkov v datasete, aby mali rovnaký rozmer.

Následne som prešiel k stiahnutiu datasetu a ten som rozdelil na trénovaciu časť a testovaciu časť.

```
import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose(
    [transforms.Resize((32, 32)),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


#trenovací dataset
trainset = torchvision.datasets.GTSRB(root='./data', split='train',
                                      download=True, transform=transform)
#testovací dataset
testset = torchvision.datasets.GTSRB(root='./data', split='test',
                                      download=True, transform=transform)
```

Trénovacia časť datasetu obsahuje 26 640 obrázkov. Avšak v tomto prípade by sme načítavali celý dataset naraz čo nie je veľmi praktické a mohlo by to spomaliť výrazne PC. Preto sa používa *dataloader*, pretože počas trénovania je praktickejšie predávať obrázky po menších častiach (batchoch), taktiež umožňuje tieto obrázky pomiešať v každej epoche, čo vedie teda ku kvalitnejšiemu a rýchlejšiemu trénovaniu.

Vytvoril som dva dataloadre pre trénovanie a testovanie, v ktorých som nastavil batch_size, shuflle na True čo znamená, že obrázok z každej triedy je zahrnutý v batchy. Ďalším parametrom je *num_workers,* ktorý hovorí o tom, koľko podprocesov sa má použiť na načítanie obrázkov. Prakticky to znamená, pokiaľ by *num_workers* bol rovný nule, tak GPU musí čakať kým CPU nenačíta obrázky. Teoreticky platí čím väčšie číslo tým GPU menej čaká.

```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                          shuffle=True, num_workers=2)

testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                         shuffle=False, num_workers=2)
```

Ďalším bodom je vytvorenie už konvolučnej neurónovej siete, kde som začal vytvorením novej triedy, ktorá rozširuje triedu *nn.Module*.

Následne bolo potrebné definovať vrstvy siete, čo som vykonal v metóde *__init__*. Funguje to tak, že inicializujem jednotlivé vrstvy a potom ich priradím k príslušnej vrstve ku ktorej chcem, buď ku konvolučnej vrstve, pooling vrstve alebo plne prepojenej vrstve.

Ako posledný bod bolo potrebné definovať metódu forward, ktorej účelom je, v akom poradí budú dáta prechádzať sieťou resp. cez jednotlivé vrstvy.

Aby som urýchlil operácie tejto siete, je možné, vykonávať jednotlivé operácie na GPU resp. na CUDA jadrách.

```
import torch.nn as nn
import torch.nn.functional as F

device = (
    "cuda"
    if torch.cuda.is_available()
    else "cpu"
)

print(f"Využíva {device}")


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 43)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net().to(device)
```

Použité metódy:

- nn.Conv2d – definovanie konvolučnej vrstvy , kde definujem počet vstupných kanálov, výstupných kanálov a veľkosť konvolučného filtra. V mojom prípade som začal s 3, pretože vstupné obrázky sú RGB,
- nn.MaxPool2d – aplikovanie 2D pooling vrstvy a vyžaduje veľkosť konvolučného filtra, ktorý je 2 x 2 a krok filtra (stride),
- nn.Linear – jedná sa o plne prepojenú vrstvu,
- F.relu – aktivačná funkcia ReLU.

Po vytvorení konvolučnej siete som mohol prejsť k nastaveniu hyperparametrov dôležitých pre trénovanie. Pre natrénovanie modelu je potrebná *loss function* a *optimizer*.

Loss function alebo chyba siete je funkcia, ktorá porovnáva výstupné hodnoty a predpokladané hodnoty, resp. meria, ako dobre sa sieť učí na príslušnom datasete. Platí, pokiaľ sa hodnota tejto funkcie znižuje, tak naša sieť sa učí správne.

Optimizers sú algoritmy používané pre zmenu atribútov siete, ako napríklad rýchlosť učenia, váhy siete z dôvodu zníženia strát (chýb siete). Taktiež zlepšujú výkonnosť siete, čo ovplyvňuje jej presnosť a rýchlosť učenia.

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Ďalší bod je už samotné trénovanie siete, kde v jednej trénovacej slučke resp. v epoche sieť vykoná predikciu na trénovacom datasete (batchy) a spätné šíri chybu predikcie aby následne upravil parametre siete.

```
for epoch in range(100):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 200 == 199:    # print every 200 mini-batches
           print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss
                    / 200))
           running_loss = 0.0
```

Táto časť kódu v podstate robí to, že začneme iterovať cez každú epochu resp. cez tréningové dáta v každom batchy. Pri prechode sieťou vypočítavame chybu na základe predikcie a našich labelov. Následne spätné šíri chybu predikcie a aktualizuje váhy, aby sme zlepšili sieť pomocou funkcie *optimizer.step()*.

Výstupom tejto časti je, že nám každých 200 mini batchov vypíše aká je chyba siete. V ďalšej časti budem zisťovať presnosť siete.

```
[Epoch : 1,   200] loss: 0.673
[Epoch : 1,   400] loss: 0.606
[Epoch : 2,   200] loss: 0.549
[Epoch : 2,   400] loss: 0.522
[Epoch : 3,   200] loss: 0.472
```

Obrázok 4.11: Výpis chyby siete

V tejto časti predikujem každý batch na nepoznaných dátach s využitím siete a vypočítam, koľko z nich predikuje správne.

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the test images: %d %%' % (100 *
       correct / total))
```

Výsledná získaná presnosť siete vyšla po 100 epochách učenia 83 %, jedná sa o úspešnosť klasifikácie do 43 tried.

```
Accuracy of the network on the test images: 83 %
```

Obrázok 4.12: Výpis presnosti siete

Po natrénovaní siete som si model uložil, pokiaľ by som chcel tento model prípadne niekde použiť. K tomu som použil príkaz *torch.save*.

```
PATH = './model.pt'
torch.save(net.state_dict(), PATH)
```

Poslednou časťou bolo otestovať, či skutočne mnou naučená konvolučná neurónová sieť rozpoznáva dopravné značky na jednotlivých obrázkoch. K tomu som použil testovací dataset, presnejšie obrázky z neho. Testovanie je možné vykonať manuálne, kde v tejto časti kódu zisťujem len predikovanú triedu alebo som vytvoril kód, ktorý zisťuje automaticky predikovanú aj aktuálnu triedu. Tento kód som tu avšak nepopisoval, ale nachádza sa v skripte *CNN_own.ipynb*.

Pomocou knižnice PIL som načítal obrázok z testovacieho datasetu a následne pomocou funkcie *transform* preformátoval na obrázok, ktorý sieť dokáže spracovať.

```
from PIL import Image
import torchvision.transforms as transforms

image_path = 'path to the image from test dataset'
image = Image.open(image_path)
```

```
image = transform(image).unsqueeze(0)
```

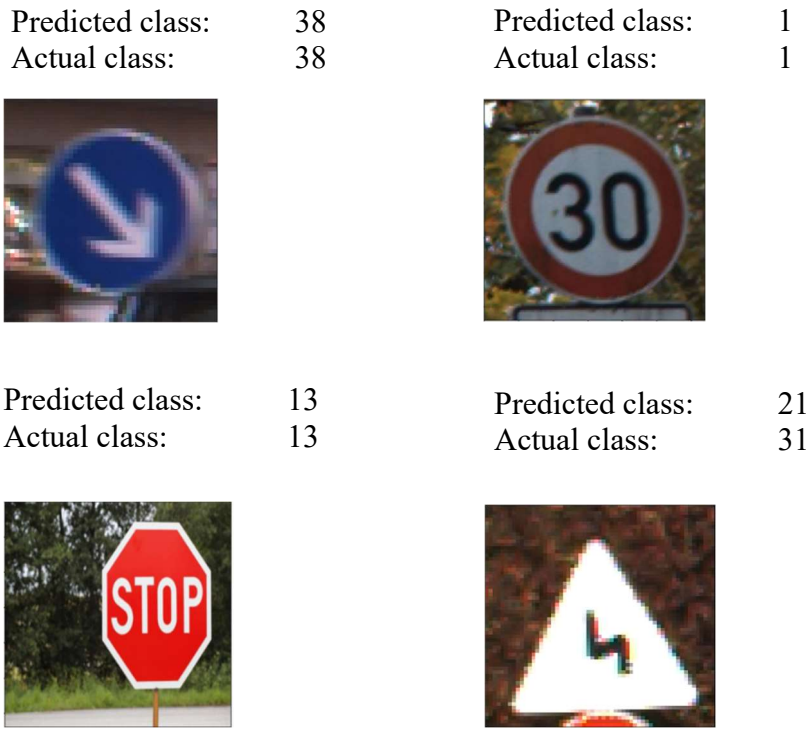
Následne som použil natrénovaný model, aby vykonal predikciu triedy predloženého obrázka

```
outputs = net(images)
_, predicted = torch.max(outputs, 1)
predicted=predicted.numpy()
predicted_class = predicted.item()
print(f'Predicted class: ',predicted)
```

```
Predicted class:  [38]
```

Obrázok 4.13: Predikcia natrénovaného modelu

Ako môžeme vidieť, predikovaná a aj aktuálna trieda, do ktorej patrí dopravná značka sú rovnaké, čo znamená, že mnou navrhnutá sieť dokáže rozpoznať a klasifikovať dopravnú značku do svojej triedy.



Obrázok 4.14: Výsledok klasifikácie na niektorých obrázkoch z testovacieho datasetu

Konvolučná neurónová sieť ktorú som vytvoril dokáže rozpoznávať dopravné značky s úspešnosťou klasifikácie do 43 tried na 83% avšak táto sieť nie je moc vhodná z dôvodu, že nemám dostatočné skúsenosti aby som ju dokázal optimalizovať a mohol použiť v reálnom čase. Z tohoto dôvodu som sa rozhodol použiť už vytvorenú architektúru konvolučnej neurónovej siete s názvom YOLOv5 ktorá sa nachádza v nasledujúcej sekcií.

4.3 Použitie vytvorenej konvolučnej neurónovej siete YOLOv5

Vo všeobecnosti YOLO znamená You Only Look Once a jedná sa o jeden z modelov konvolučných neurónových sietí v počítačovom videní. Tento model siete je najčastejšie používaný pre detekciu a klasifikáciu objektov v reálnom čase.

Model siete YOLOv5 je vylepšením predchádzajúcich modelov sietí YOLO. Tento konkrétny model bol vyvinutý spoločnosťou Ultralytics napísaný v jazyku Python a s využitím frameworku PyTorch. [32]

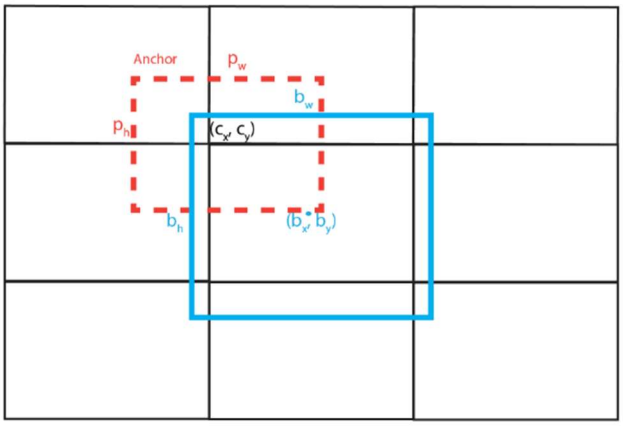
Na spracovanie celého obrázku využíva jedinú neurónovú sieť nazývanú ako Single-shot.

Vstupný obrázok je rozdelený do mriežky o SxS veľkosti a každá oblasť z mriežky je predávaná ako samostatný obrázok, v ktorej sa môže nachádzať detekovaný objekt. Každá bunka z mriežky predikuje na základe anchor boxov určitý počet bounding boxov, ktoré reprezentujú umiestnenie a veľkosť detekovaného objektu (lokalizácia objektu). Anchor boxy sú predom definované a majú pevne dané rozmery. Každý bounding box je reprezentovaný vo formáte:

$$y = [p_c, b_x, b_y, b_h, b_w, c] \tag{4.1}$$

kde: p_c – pravdepodobnosť, či sa objekt nachádza v bounding boxe
b_x , b_y – súradnice stredu bounding boxu
b_h , b_w – výška a šírka boudning boxu
c – predstavuje triedu objektu

Všetky súradnice boudning boxu sú normalizované s výškou a šírkou obrázka a hodnoty sa pohybujú od 0 do 1.

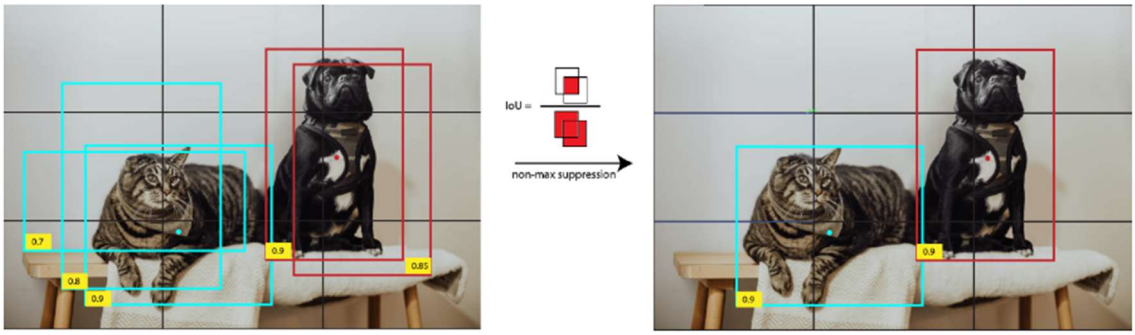


Obrázok 4.15: Predikcia bounding boxu založená na anchor boxe[31]

Pre každý bounding box sa predikuje pravdepodobnosť, že sa v danej bunke nachádza objekt a zároveň sa predikuje trieda objektu.

Na základe týchto predikcií sa určí finálna množina ohraničujúcich bounding boxov a tried objektov. To dosiahneme tak, že pre každú bunku mriežky sa vyberie bounding box s najvyššou pravdepodobnosťou detekovaného objektu a pridá sa k finálnej množine. Využíva sa metóda Non-Max suppresion, ktorá rieši problém, keď algoritmus detekuje niekoľko bounding boxov pre jednu triedu ako môžeme vidieť na obrázku č.4.16.

Cieľom je minimalizovať chybu predikcií so skutočnými anotáciami a tým dosiahnuť lepšiu presnosť detekcie objektov. Chyba predikcie sa vypočíta ako rozdiel medzi predikovanými bounding boxami a skutočnými anotáciami.



Obrázok 4.16: Využitie Non-Max suppresion [31]

Tento proces sa následne opakuje pre každý obrázok v tréningovej sade a algoritmus sa učí zlepšovať svoje predikcie na základe vypočítanej chyby. Proces trénovania znova prebieha v niekoľkých epochách.

Táto architektúra je používaná hlavne z dôvodu jej rýchlosti a presnosti a používa sa v rôznych aplikáciách, ako je napríklad zdravotníctvo a autonómne vozidlá.

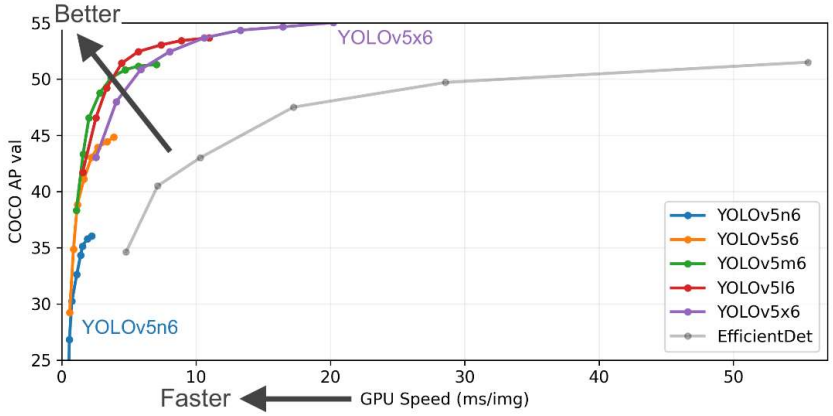
Samotná presnosť tohto algoritmu závisí na veľa faktoroch, ako je počet epoch trénovania, správne zvolené parametre trénovania, kvalitný dataset a pod. Kvalitný dataset znamená, že objekty, na ktoré má byť model naučený, musia byť najskôr oanotonované, resp. musia byť vytvorené bounding boxy okolo týchto objektov.

Algoritmus YOLOv5 obsahuje niekoľko modelov:

- **YOLOv5n:** je to model typu nano a je určený pre IoT zariadenia,
- **YOLOv5s:** jedná sa o malý model s 7,2 miliónmi parametrov a je určený pre vykonávanie výpočtov na CPU,
- **YOLOv5m:** jedná sa o stredný model s 21,2 miliónmi parametrov a je asi najvýhodnejší model pretože poskytuje rovnováhu medzi rýchlosťou a presnosťou,
- **YOLOv5l:** jedná sa o väčší model so 46,5 miliónmi parametrov a je ideálny pre datasety, kde je potrebné detekovať menšie objekty,

- **YOLOv5x:** najväčší model z modelov YOLOv5 s 86,7 miliónmi parametorov je pomalší ale poskytuje najvyššiu presnosť. [32]

Na tomto obrázku sa nachádza porovnanie modelov YOLOv5, ktoré boli trénované na datasete COCO.



Obrázok 4.17: Porovanie modelov YOLOv5[33]

Po zistení, ako funguje algoritmus YOLO, som prešiel k zisteniu, ako funguje **trénovanie** nejakého z modelov. Postup trénovania prebieha v niekoľkých krokoch:

- **Výber datasetu:** ako som spomínal, dataset už bol v mojom prípade oanotonovaný, to znamená, že bol vytvorený bounding box, kde sa skutočne nachádza objekt. Avšak ak takýto dataset mať nebudeme, je potrebné obrázky v našom vlastnom dataset ohraničiť a tak vytvoriť bounding box kde sa hľadaný objekt skutočne nachádza.
- **Vytvorenie konfiguračného súboru:** Vytvorí sa konfiguračný súbor, ktorý obsahuje rôzne nastavenia modelu, ako napríklad počet tried objektov, veľkosť vstupného obrazu, a iné hyperparametre.
- **Trénovanie modelu:** Model sa trénuje pomocou trénovacej sady obrázkov. Počas trénovania sa postupne upravujú váhy modelu tak, aby sa minimalizovala chyba medzi predikciami modelu a skutočnými anotáciami. Váhy sa upravujú pomocou algoritmu optimalizácie, ako napríklad Stochastic Gradient Descent (SGD), ktorý prispôsobuje váhy modelu tak, aby minimalizoval chybu.
- **Validácia modelu:** Po každej epoche sa model testuje na validačnej sade obrázkov, kde táto sada obrázkov nemôže byť rovnaká ako trénovacia sada. Validácia umožňuje určiť, ako dobre model generalizuje na nové obrázky, na ktoré nebol trénovaný. Ak sa zistí, že model sa zhoršuje v predikovaní objektov, môže sa zastaviť trénovanie alebo upraviť niektoré nastavenia.

- **Testovanie modelu:** Po dokončení trénovania sa model testuje na sade obrázkov na ktorých nebol trenovaný ani validovaný, aby sa zistilo, ako dobre dokáže detekovať objekty v nových obrázkoch.

Augmentácia dát

S každým trénovacím batchom (predloženie niekoľkých rôznych obrázkov vstupu), tak YOLOv5 tieto trénovacie dáta pošle cez dataloader, ktorý augmentuje data online. Data loader vykonáva tri druhy augmentácií:

- úprava rozmerov,
- úprava farebného priestoru,
- mosaic augmentácia dát.

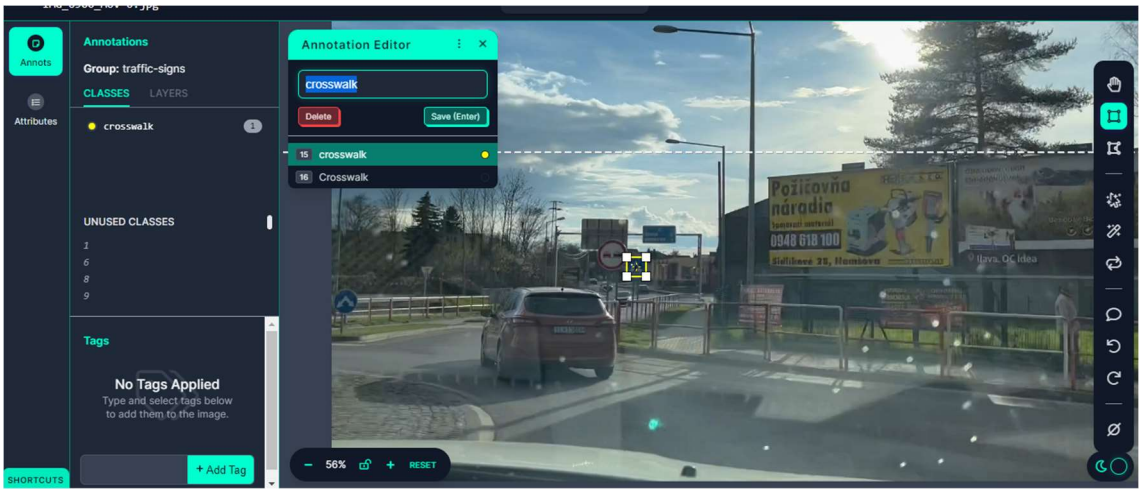
Mosaic augmentácia dát kombinuje štyri obrázky do štyroch buniek mriežky a tvoria jeden obrázok. V mojom prípade je použitý už aj batch, takže na obrázku č.4.18 je ich o niečo viac. [35]

Obrázok 4.18: Mosaic augmentacím dát

Získal som znalosti potrebné pre to, aby som mohol trénovať vlastný dataset a mohol som prejsť k samotnej implementácií s použitím modelu YOLOv5.

Z dôvodu, že algoritmus YOLO potrebuje mať všetky obrázky oanotonované, rozhodol som sa použiť na anotáciu obrázkov stránku Roboflow a vytvoril si vlastný dataset. Môj dataset obsahuje niečo cez 24 000 obrázkov a sú rozdelené do 6 tried (Prechod pre chodcov, daj prednosť v jazde, kruhový objazd, stopka a hlavná cesta). Dataset som ešte rozdelil na trénovaciu, testovaciu a validačnú časť. Trénovacia časť obsahuje 21 595 obrázkov, testovacia 829 obrázkov a validačná 1754 obrázkov. Tento dataset bol vytvorený už z existujúcich datasetov a taktiež z niekoľkých mojich fotiek, ktoré bolo treba ručne oanotovať.

Na stránke Roboflow to funguje jednoducho, buď nahrám video, alebo obrázky, a následne už len vytváram bounding boxy okolo objektov, ktorým následne priradzujem triedy, do ktorých ma byť objekt zaradený.



Obrázok 4.19: Použitie Roboflow pre anotáciu objektov

Prvý krokom bolo rozhodnúť sa, ktorý model použiť. Najskôr som sa rozhodol pre trénovanie použiť model YOLOv5s, kde samotný proces učenia už bol jednoduchý.

Po stiahnutí datasetu zo stránky Roboflow som potreboval naklonovať repozitár zo stránky ultralytics s modelmi yolov5.

Ďalším krokom bolo vytvorenie konfiguračného súboru, ktorý je vo formáte YAML, kde v mojom prípade sa v ňom nachádzajú cesty k trénovacím, testovacím, validačným dátam, počtu tried a samotné názvy tried.

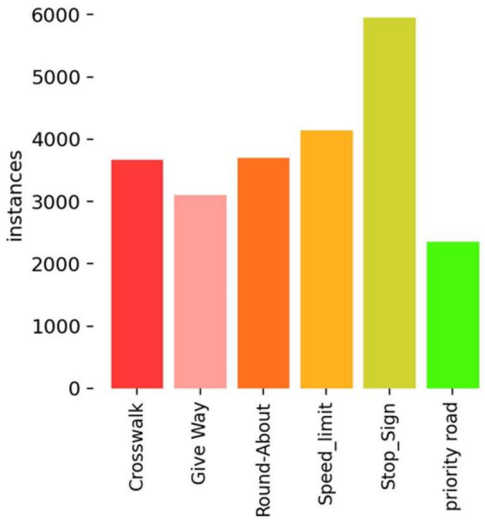
```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 6
names: ['Crosswalk',
        'Give Way',
        'Round-About',
        'Speed_limit',
        'Stop_sign',
        'Priority road']
```

Na všetko ostatné stačil príkazový riadok, z ktorého som spúšťal aj trénovanie pomocou príkazu, kde môžeme vidieť veľkosť vstupných obrázkov , batch size, koľko epóch sa má vykonať, cesta ku konfiguračnému súboru a počiatočné váhy. Voľba týchto parametrov je na našom uvážení, kde obecne platí, *--img* volíme na základe veľkosti obrázkov v datasete , *--batch* sa snažíme mať čo najväčší z dôvodu zlepšenia kvality učenia, počet epóch je z dokumentácie odporúčané na 300 epóch avšak v mojom prípade však dochádzalo už k preučeniu, *--weights* sú pred učené váhy modelu ktorý chceme použiť.

```
python train.py --img 640 --batch 16 --epochs 150 --data PATH TO CONFIG
FILE  --weights yolov5s.pt --workers 2
```

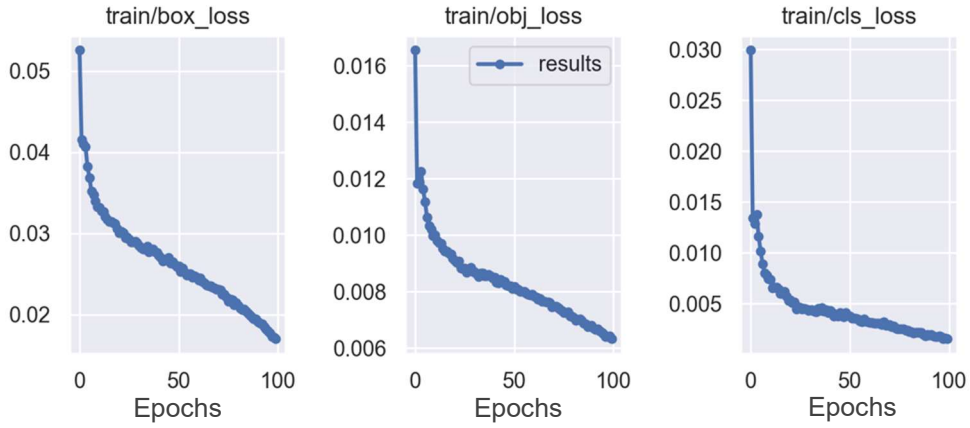
Po spustení príkazu som sledoval trénovanie pomocou aplikácie TensorBoard, ktorej stačilo zadať cestu, kde sa postupne ukladajú trénované checkpointy, resp. sa tam nachádzajú všetky potrebné parametre metrík, strát a validačné predikcie. Taktiež v ňom nájdeme aj náhľad celého modelu siete, kde je možne jednotlivé vrstvy rozkliknúť a pozrieť sa čo sa v nich nachádza. Model bude priložený v prílohách z dôvodu veľkosti obrázka.

Taktiež som si mohol prezrieť rozloženie datasetu, ktoré taktiež nie je príliš kvalitne rozdelené. Ideálnym prípadom by bolo, pokiaľ by bol dataset rovnomerne rozložený, z hľadiska počtu obrázkov v jednotlivých triedach.



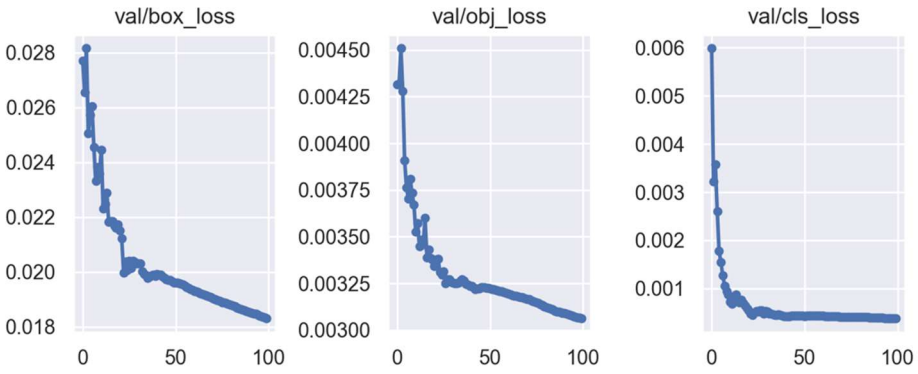
Obrázok 4.20: Rozloženie datasetu

Po skončení trénovania sme si mohli pozrieť úspešnosť našej natrénovanej siete. Výsledok sa nachádza na obrázkoch č.4.21, č.4.22 a č.4.23.



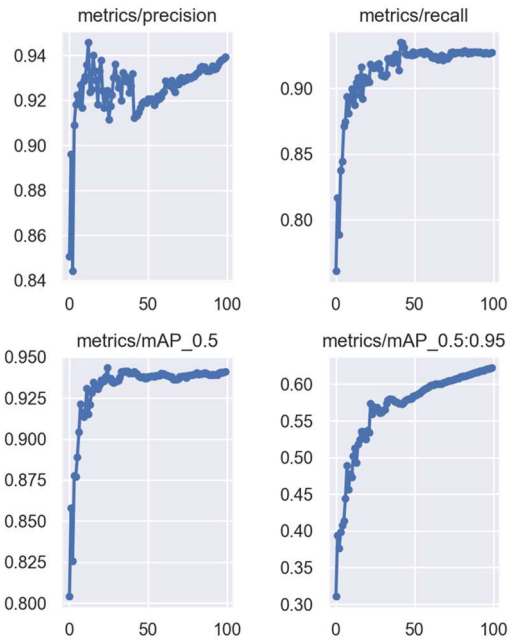
Obrázok 4.21: Výsledok trénovania siete

Box_ loss – hovorí o tom, ako dobre dokáže algoritmus lokalizovať stred objektu a ako dobre predikovaný bounding box prekrýva objekt,
Obj_loss – objektivita predstavuje mieru pravdepodobnosti, či objekt existuje v navrhovanej oblasti (bounding boxe). Ak je objektivita vysoká, znamená to , že bounding box obsahuje objekt,
Cls_loss – hovorí o tom, ako dobre dokáže algoritmus predpovedať správnu triedu a daný objekt.



Obrázok 4.22: Výsledok validácie natrénovanej siete

V tomto prípade sa validácia vykonáva počas trénovania na validačnom datasete. Na obrázkoch č.4.21 a č.4.22 si môžeme všimnúť, že jednotlivé chyby klesajú, čo značí, že sa model učí. Taktiež si môžeme všimnúť na obrázku č.4.22, že hodnoty u validácie sú nižšie ako trénovacie. Na základe odpovede jedného z vývojárovej tejto siete je to spôsobené tým, že trénovacie dáta sú augmentované a validačne nie sú. [34]



Obrázok 4.23: Výsledok metrík natrénovanej siete na validačnom datasete

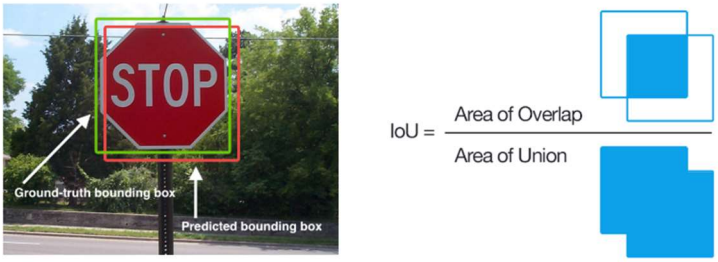
Precision – hovorí o tom, koľko predikovaných bounding boxov bolo správnych

$$precision = \frac{True\ positive}{True\ positive + False\ postive} \tag{4.2}$$

Recall – hovorí o tom, koľko bolo skutočne predikovaných bounding boxov resp. či model predikoval vždy keď mal

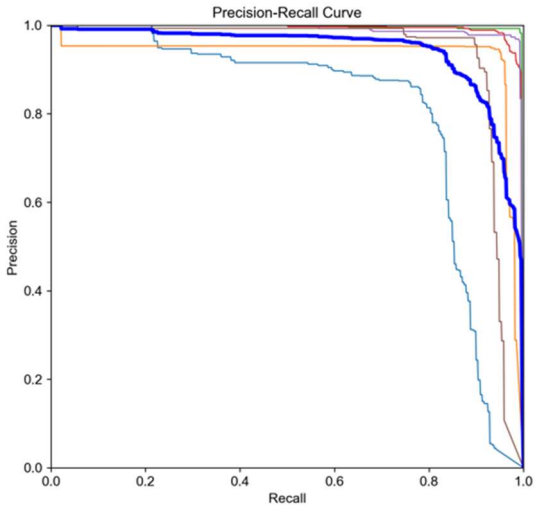
$$recall = \frac{True\ positive}{True\ positive + False\ negative} \tag{4.3}$$

IoU(Intersection over Union) – jedná sa o prekrytie predikovaných bounding boxov s ground truth boxami (naše oanotonované obrázky). Ak chceme, aby predikovaný bounding box odpovedal čo najlepšie ground truth boxu, tak to zabezpečíme tým, že IoU bude väčšie číslo ktoré môže nadobúdať hodnoty od 0 do 1.



Obrázok 4.24: Intersection over Union[36]

Precision-Recall curve (PR curve) – predstavuje kompromis medzi presnosťou a recall pre rôzne prahové hodnoty. Táto krivka pomáha vybrať najlepší práh, ktorý umožní maximalizáciu obidvoch metrík (precision a recall).[37]



Obrázok 4.25: PR curve

Average Precision (AP) - jedná sa o oblasť pod krivkou PR, kde táto priemerná presnosť je počítaná pre každú triedu. Taktiež predstavuje spôsob ako spojiť tieto dve metriky do jednej a je to v podstate vážený súčet presností pre každú prahovú hodnotu tejto PR krivky. [37]

mAP – používa sa k vyhodnoteniu, aká kvalitná je naša sieť, pre jej výpočet využíva niekoľko metrík ako je Average Precision (AP), ktorý sa počíta pomocou metrík, ako je napríklad Confusion Matrix, Intersection over Union (IoU), Recall ,Precision, Precision-Recall curve ktoré som popísal vyššie. Používa sa napríklad pre porovnanie dvoch rôznych modelov, ktoré majú rovnakú úlohu.

Prakticky to funguje tak, že pre každú triedu vypočíta AP s rôzne nastavenými prahmi IoU a následne spravím priemer všetkých hodnôt AP.[37]

$$mAP = \frac{1}{n}\sum_{i=1}^{n} AP_i \tag{4.4}$$

kde: n – počet tried

AP_i – priemerná presnosť triedy n

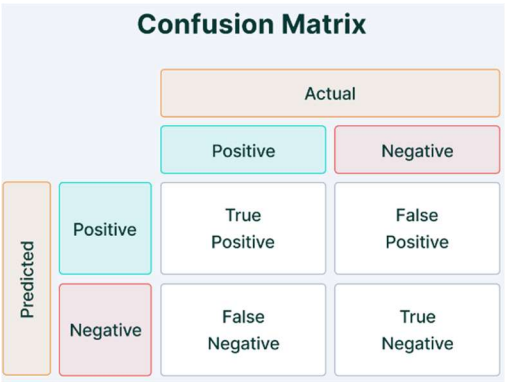
Výsledný mAP je však vypočítaný na validačných obrázkoch a vypočíta sa ako priemer všetkých hodnôt mAP na triedu.[37] V mojom prípade bol mAP vypočítaný pre IoU =0.5 a mAP pre IoU = 0.5 do 0.95 s krokom 0.05. Túto skutočnosť si môžeme všimnúť na obrázku č.4.23.

Testovanie natrénovanej siete som následne robil pomocou príkazu

```
python val.py --weights  PATH TO TRAIN WEIGHTS --data PATH TO CONF OF
DATASET --img 640 --task test
```

a výsledkom je matica zámien alebo confusion matrix.

Confusion matrix (matica zámien) – jedná sa v podstate o tabuľku, ktorá sa používa pre vyhodnotenie presnosti klasifikačných algoritmov. Predikcia v mojom prípade závisí na prahu, IoU, názvoch tried a predikovaných bounding boxoch.



Obrázok 4.26: Všeobecne matica zámien[36]

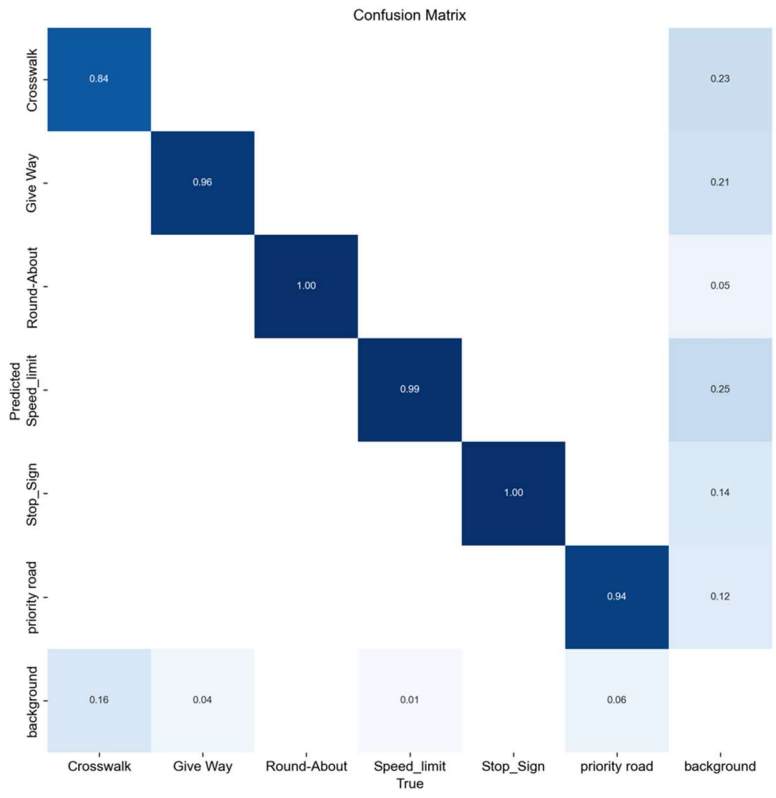
V tejto tabuľke sa nachádzajú:

- True Positive (TP) – model predikuje, že bounding box existuje a to je správne.
    - Príklad: Predikujem, že sa značka nachádza v obrázku a skutočne tam je.
- True Negative (TN) – model nepredikuje žiadny bounding box a to je správne.
    - Príklad: Predikujem, že sa značka nenachádza v obrázku a skutočne tam nie je.
- False Positive (FP) – model predikuje, že bounding box existuje, ale je to nesprávne.
    - Príklad: Predikujem, že sa značka nachádza v obrázku, ale skutočne tam nie je.
- False Negative (FN) – model nepredikuje žiadny bounding box a to je nesprávne.
    - Príklad: Predikujem, že sa značka nenachádza v obrázku, ale skutočne tam je.

Z tejto matice zámien vychádzajú v podstate niektoré metriky, o ktorých som písal o niečo vyššie. [37]

V mojom prípade, ako môžeme vidieť na obrázku č.4.27, mám maticu zámien s klasifikáciou do viacerých tried, ktoré sú ešte normalizované.

Z obrázka si môžeme všimnúť, že napríklad pre triedu Crosswalk si je model istý na 84 %, že predikuje správne túto značku na testovacom datasete. Taktiež si z tejto matice môžeme všimnúť, že nedochádza k zámenám medzi jednotlivými triedami.



Obrázok 4.27:Matica zámien

Po vykonaní validácie resp. testovania na testovacom datasete som prešiel k samotnej detekcií, kde som použil príkaz

```
python detect.py --source PATH TO THE IMAGE/VIDEO --weights
runs/train/exp/weights/best.pt --conf 0.25
```

Alebo je možné použiť môj skript *traffic.ipynb*

Obrázok 4.28: Detekcia na reálnych snímkach

Následne som prešiel k vytvoreniu môjho skriptu *real_time.py,* v ktorom vykonávam detekciu dopravných značiek real-time na testovacom videu.

Obrázok 4.29: Detekcia na testovacom videu

4.4 Vytvorenie uživateľského rozhrania(GUI)

Mojim cieľom bolo taktiež vytvoriť užívateľské rozhranie, v ktorom bude detekcia prebiehať real-time na displeji Rasspberry Pi . V tomto užívateľskom rozhraní sa budú nachádzať základné informácie o zariadení a možnosť otvoriť kameru, kde sa následne zobrazí okno v ktorom prebieha detekcia. Taktiež bude možné detekciu dopravných značiek vypnúť pomocou tlačidla zatvoriť kameru. Taktiež je v tomto rozhraní implementovaná možnosť prepnutia dvoch režimov dark a light.

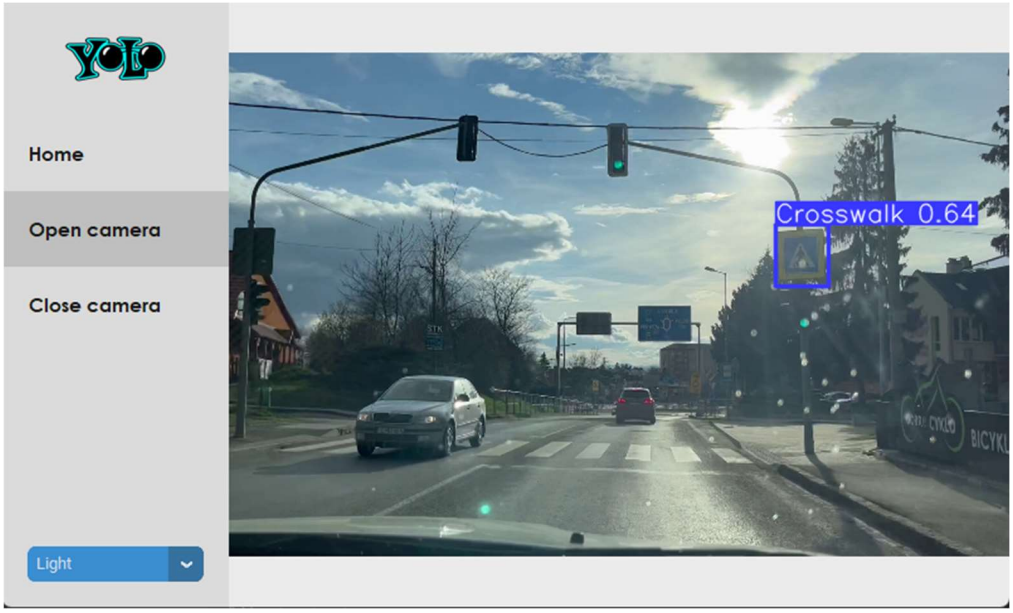
Detekcia v tomto GUI prebieha rovnako ako pri obyčajnom spustení skriptu, jediný rozdiel je len v obohatení o vizuálnu stránku a interaktívne ovládanie mojej aplikácie pomocou vizuálnych prvkov ako sú tlačidla, textové polia a pod.

K vytvoreniu tohto užívateľského rozhrania som využil pythonovskú knižnicu CustomTkinter založenú na Tkinter, ktorá poskytuje nové a moderné widgety a taktiež poskytuje tmavý a svetlý režim, ktorý môže byť nastavený manuálne alebo na základe nastaveného vzhľadu systému.

Skript k tomuto GUI ktorého názov je *GUI.py* bude priložený do príloh z dôvodu, že je príliš dlhý a zdokumentovaný bude priamo v skripte.

Obrázok 4.30: Náhlad uživateľského rozhrania

Po otvorení kamery v uživateľskom rozhraní výstup resp. režim detekcie a rozpoznávania dopravnej značky funguje tak, že vidíme stream z kamery a pri detekcií značky sa vykreslí bounding box a trieda príslušnej značky. Túto skutočnosť môžeme vidieť na obr. č.4.31.



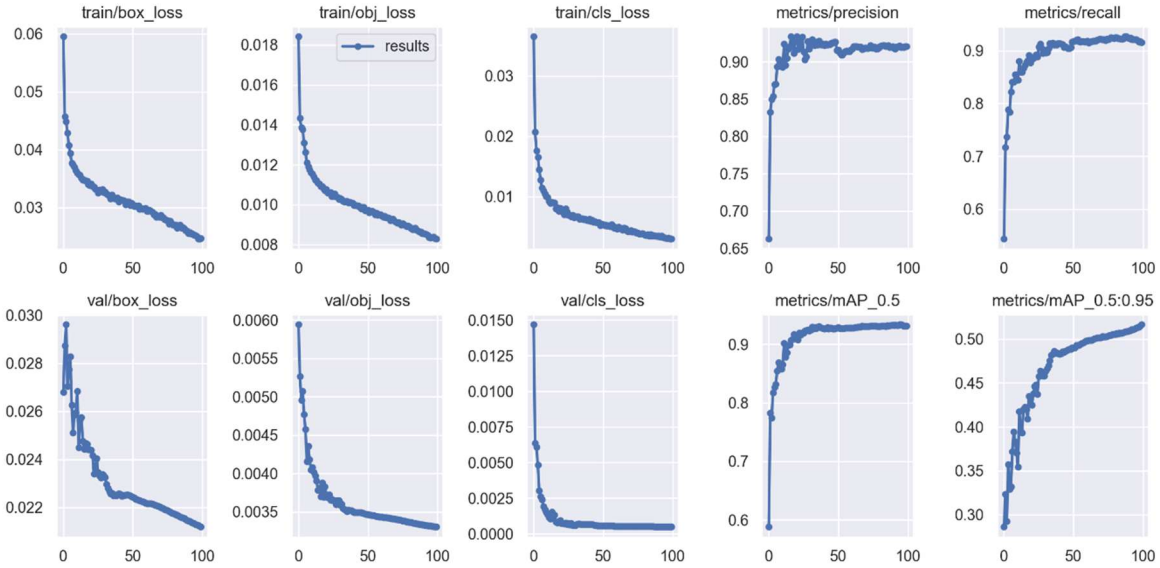
Obrázok 4.31: Reálny režim po otvorení kamery

5. VÝSLEDNÉ TESTOVANIE A IMPLEMENTÁCIA NA HARDWARE

V tejto časti sa budem zaoberať výsledným testovaním mojej naučenej siete na testovacích videách, ktoré boli vytvorené počas dňa ale aj noci. V týchto videách sa nachádzajú všetky dopravné značky, na ktoré sa konvolučná neurónova sieť učila. Jednalo sa teda o šesť dopravných značiek ako Stop, Daj prednosť v jazde, Maximálna povolená rýchlosť, Hlavná cesta, Prechod pre chodcov a Kruhový objazd. Taktiež sa budem zaoberať implementáciou na hardware.

Implementácia na hardware

Pri implementácii na hardware som nepoužíval model yolov5s, čiže jeden z menších modelov, ale použil som model nano, ktorý ma o polovicu menej parametrov ako model small, a teda dôjde k zvýšeniu výkonu, avšak za cenu zhoršenia detekcie značiek.



Obrázok 5.1: Výsledok trénovania modelu nano

Pri porovnaní výsledkov trénovania si môžeme všimnúť, že chyby pri modely nano sú väčšie oproti modelu small, čo vedie k spomínanému zhoršeniu detekcie. Taktiež si môžeme všimnúť, že metriky nedosahujú rovnakých výsledkov ako pri modely small.

Spojenie Raspberry Pi, kamera a displej bolo v podstate jednoduché z dôvodu, že komponenty boli kompatibilné s Raspberry. Displej po pripojení fungoval hneď, problém nastal až s kamerou, ktorú som sa snažil obsluhovať pomocou knižnice OpenCV. Kameru sa mi pomocou tejto knižnice podarilo otvoriť, avšak neposielala žiadne snímky na obrazovku. Tento problém sa mi nakoniec nepodarilo vyriešiť a použil som pre obsluhu

kamery knižnicu picamera2, kde bolo následne potrebné upraviť skript *GUI.py* ktorý je implementovaný na hardwari. Upravený kód bude priložený v prílohách pod názvom *GUI_raspberry.py*

Po implementácií na samotnom hardvéri detekcia funguje na 2,6 FPS, čo nie je ideálne pre plynulú detekciu a rozpoznávanie, avšak je to maximum, čo dokážem s príslušným hardwarom dosiahnuť. Pri rýchlosti 50 km/hod odpovedá 2,6 FPS približne 5,3 m na snímku, čo by pre detekciu mohlo stačiť, ale ako zariadenie do auta to nie je ideálne. Možné zlepšenie výkonu je možne prípadným dokúpením Coral USB akcelerátora, na ktorom by mohli byť vykonané všetky operácie súvisiace s konvolučnou neurónovou sieťou. Ďalšou možnosťou je použitie výkonnejšieho hardwaru, ako je napríklad NVIDIA Jetson Nano, ktorý bol vytvorený pre aplikácie ako je detekcia objektov, klasifikácia a pod. Avšak to som rozhodol nepoužiť z dôvodu, že Raspberry Pi 4B som mal fyzicky dostupné. Taktiež je možné zvýšiť výkon hardwaru prípadným zmenšením rozlíšenia alebo výrezom len pravá polka kde väčšinou sa v reálnom svete nachádzajú dopravné značky na pravej strane vozovky.

Obrázok 5.2: Fotka reálneho zariadenia v aute

Testovanie

V nasledujúcich tabuľkách bude vyhodnotené výsledné testovanie na testovacom videu počas dňa ale aj noci. V týchto tabuľkách si pod pojmom správne klasifikovaná značka predstavíme správne zdetekovanú, lokalizovanú a klasifikovanú značku do správnej triedy.

Detekcia značky vodičovi pomáha z hľadiska bezpečnosti na ceste, kde na displeji sa zobrazia všetky zdetekované značky vo forme orámovania a príslušnej triedy. Z hľadiska bezpečnosti by bolo najlepšie aby bol vodič po zdetekovaní značky upozornení zvukovo a prípadne zobrazením zväčšenej značky na obrazovku.

Deň:

Počet kandidátov	Správne klasifikované značky	Nesprávne klasifikované značky	Neklasifikované značky
52	42	10	0
	81%	19%	0%

Tabuľka 1: Úspešnosť klasifikácie modelu počas dňa

Noc:

Počet kandidátov	Správne klasifikované značky	Nesprávne klasifikované značky	Neklasifikované značky
52	31	21	0
	59%	41%	0%

Tabuľka 2: Úspešnosť klasifikácie modelu počas noci

Môžeme si všimnúť, že vo videu sa nachádza 52 značiek rôznych druhov ktoré boli detekované. Správne klasifikovalo počas dňa 42 značiek čo znamená, že na ceste sa nachádzalo 42 značiek na ktoré bol model naučený avšak nastal aj prípad kedy model nesprávne klasifikoval, to znamená, že napríklad značku prikázaný smer vpravo klasifikoval ako kruhový objazd.

Taktiež vidíme, že úspešnosť klasifikácie dopravných značiek cez deň bola o 22 % vyššia ako v noci. Hlavným dôvodom môže byť to, že video kde prebiehala klasifikácia v noci bolo natočené za nepriaznivých podmienok (pršalo).

6. ZÁVER

V tejto bakalárskej práci som sa zaoberal detekciou a rozpoznávaním dopravných značiek, kde som sa najskôr zameral na teoretickú časť, v ktorej som sa zoznámil s určitými metódami detekcie a rozpoznávania dopravných značiek, vo všeobecnosti objektov.

V úvode tejto práce sa venujem histórií a problematike, ktorá súvisí s detekciou a rozpoznaním dopravnej značky. Popísal som detekciu na základe farby, tvarov a strojového učenia.

Časť strojového učenia sa detailnejšie zaoberá konvolučnými neurónovým sieťami z dôvodu, že som detekciu a rozpoznávanie implementoval práve pomocou týchto sietí.

V hardwarovej časti som sa venoval výberu jednotlivých komponentov pomocou ktorých bol samotný hardware zostrojení a taktiež návrhu, a vytvoreniu krabičky v ktorej budú jednotlivé komponenty uložené. Návrh krabičky sa nachádza v časti návrh krabičky, kde k tomuto návrhu sú vytvorené aj výrobné výkresy, ktoré sa nachádzajú v prílohách.

Taktiež som si v softwarovej časti vyskúšal rozpoznávanie dopravných značiek pomocou Haarových príznakov, čo bol prvotný pokus o rozpoznanie nejakej dopravnej značky. Následne som si vyskúšal aj vytvorenie svojej konvolučnej neurónovej siete, ktorá dokázala rozpoznať a klasifikovať dopravnú značku do rovnakej triedy ako bola aktuálna trieda a jej úspešnosť klasifikácie do 43 tried bola 83%.

Ako poslednú možnosť som si vyskúšal použiť už vytvorenú architektúru konvolučnej neurónovej siete. Presnejšie sa jedná o konvolučnú neurónovú sieť YOLOv5, ktorá je vhodná pre real-time rozpoznávanie a detekciu objektov. Dôkladnejší postup je popísaný v časti použitie konvolučnej neurónovej siete YOLOv5, kde som používal model YOLOv5s (small).

Následne som prešiel k implementácií na hardware, presnejšie na Raspberry Pi 4B, kde som naučený model, avšak model nano použil na detekciu a klasifikáciu priamo z kamery. Pri implementácií na tento hardware vznikol dosť závažný problém, ktorý som dlho nedokázal vyriešiť. Tento problém sa zaoberal kamerou, ktorú sa mi cez knižnicu OpenCV podarilo otvoriť, avšak nedokázal som získavať žiadne reálne snímky. Po konzultáciách s rôznymi vyučujúcimi sme nedokázali prísť na príčinu tohto problému. Nakoniec som použil na obsluhu kamery knižnicu picamera2. Detekcia na hardwari prebiehala len 2,6 FPS, čo bolo maximum, čo som dokázal dosiahnuť s príslušným hardwarom. Niektoré možnosti ako zvýšiť výkon hardwaru som popísal v časti výsledné testovanie a implementácia na hardware.

Pri tejto konvolučnej sieti je potrebné vytvoriť rozsiahly dataset ktorý som vytvoril pomocou stránky Roboflow. Najhorší problém, ktorý vznikol, bol práve s datasetom, pretože konvolučné neurónové siete potrebujú, aby sa mohli kvalitne naučiť veľké množstvo trénovacích obrázkov. Od tohto problému závisí aj kvalita neurónovej siete a jej detekcia na reálnych snímkach, kde som na testovacom videu ktoré obsahuje 52

značiek správne klasifikoval počas dňa len 81% a v noci len 59% značiek. Môj model má za úlohu detekovať a klasifikovať značky do šiestich tried.

Cieľ do budúcna vidím hlavne vo výmene alebo vytvorení vlastného hardwaru z dôvodu, že v momentálnom stave je toto zariadenie na plynulé rozpoznávanie nevhodné. Taktiež by sa dala presnosť modelu získať zväčšením datasetu. Avšak túto prácu považujem za úspešnú z dôvodu, že s témou o konvolučných neurónových sieťach som sa v doterajšom štúdiu moc nestretol a táto práca mi umožnila sa priblížiť viac do problematiky.

LITERATÚRA

[1] TOTH, Štefan. *ROZPOZNÁVANIE DOPRAVNÝCH ZNAČIEK A ICH POUŽITIE V MAPOVÝCH APLIKÁCIÁCH*. In: . GIS Ostrava, 2011, s. 16.

[2] CHOKSEY, Jessica. WHAT IS TRAFFIC-SIGN RECOGNITION. *J.D.POWER* [online]. 29.6.2022 [cit. 2022-12-28]. Dostupné z: https://www.jdpower.com/cars/shopping-guides/what-is-traffic-sign-recognition

[3] SZABÓOVÁ, Anikó. *Detekcia a rozpoznavanie dopravných značiek* [online]. Bratislava, 2017 [cit. 2023-05-10]. Dostupné z: http://www.dcs.fmph.uniba.sk/bakalarky/obhajene/getfile.php/Bakal%E1rska+pr%E1ca.pdf?id=338&fid=653&type=application%2Fpdf. Bakalárska práca. Univerzita Komenského v Bratislave, Fakulta matematiky,fyziky a informatiky. Vedoucí práce RNDr. Zuzana Černeková, PhD.

[4] *ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. Moderní počítačová grafika. Druhé. ČVUT v Prahe: Computer Press, 2004. ISBN 8025104540.*

[5] HSV Color Model in Computer Graphics. *GreeksforGeeks* [online]. 2009,24.8.2022 [cit. 2022-12-28]. Dostupné z: https://www.geeksforgeeks.org/hsv-color-model-in-computer-graphics/

[6] *HSV: Ako funguje HSV* [online]. [cit. 2022-12-28]. Dostupné z: https://web.cs.uni-paderborn.de/cgvb/colormaster/web/color-systems/hsv.html

[7] *Color Model Conversion function* [online]. [cit. 2022-12-28]. Dostupné z: https://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/color-model-conversion-function.htm

[8] LIU, Chunsheng, Shuang LI, Faliang CHANG a Yinhai WANG. *Machine Vision Based Traffic Sign Detection Methods: Review, Analyses and Perspectives* [online]. 26.6.2019 [cit. 2022-12-28]. Dostupné z: https://ieeexplore.ieee.org/document/8746141

[9] HLAVÁČ, Václav. *Hledání hran a hranových bodů* [online]. ČVUT v Prahe [cit. 2022-12-29]. Dostupné z: http://people.ciirc.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/22EdgeDetectionCz.pdf

[10] HORÁK, Karel. *Lokální příznaky a korespondence* [online]. VUT v Brne [cit. 2022-12-29]. Dostupné z: http://vision.uamt.feec.vutbr.cz/ROZ/lectures/02_Lokalni_priznaky_a_korespondence.pdf

[11] TRÁVNÍČEK, Vojtěch. *POROVNÁVÁNÍ VÝZNAMNÝCH BODŮ PRO DETEKCI OBJEKTŮ V OBRAZE*. Brno, 2013. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce Ing. Vratislav Harabiš.

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91–110, 2004. Dostupné z: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[13] H. Bay, T. Tuytelaars, and L. van Gool. SURF: Speeded up robust features. In Proceedings of European Conference on Computer Vision, 2006. Dostupné z: http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

[14] ŽILKA, F. *Detektory a deskriptory oblastí v obrazu* [online]. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. 2016.

[15] MATAS. J., CHUM. O., URBAN. M., PAJDLA. T., Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. Proceedings of the British Macine Vision Conference. 2002. Dostupné z: https://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/matas_bmvc2002.pdf

[16] FREUND, Yoav a Robert SCHAPIR. *Experiments with a new boosting algorithm* [online]. 22.1.1996 [cit. 2022-12-29]. Dostupné z: http://machine-learning.martinsewell.com/ensembles/boosting/

[17] ČERŇANSKÝ, Michal. *Support Vector Machines* [online]. STU v Bratislave [cit. 2022-12-29]. Dostupné z: http://www2.fiit.stuba.sk/~cernans/nn/nn_texts/neuronove_siete_priesvitky_svm.pdf

[18] KARPATHY, Andrej. *Konvolučné neurónové siete- Architektúra* [online]. [cit. 2022-12-29]. Dostupné z: https://cs231n.github.io/convolutional-networks/

[19] PANÍČEK, Andrej. *DETEKCE OBJEKTŮ POMOCÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ* [online]. Brno, 2019 [cit. 2023-05-10]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=198195. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačných technologií. Vedoucí práce Ing. LUKÁŠ TEUER.

[20] MURÁŇ, Juraj. *Úvod do konvolučných neurónových sietí* [online]. 28.11.2019 [cit. 2022-12-29]. Dostupné z: https://umelainteligencia.sk/uvod-do-konvolucnych-neuronovych-sieti/

[21] JIRSÍK, Václav. *Cvičenie č.5- Konvolúcia*. 2022. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií.

[22] *Sigmoid-function* [online]. [cit. 2022-12-29]. Dostupné z: https://de.wikipedia.org/wiki/Datei:Sigmoid-function-2.svg

[23] *Hyperbolický tangens* [online]. [cit. 2022-12-29]. Dostupné z: https://cs.wikipedia.org/wiki/Hyperbolick%C3%BD_tangensvg

[24] *What is ReLU and Sigmoid activation function?* [online]. 20.4.2022 [cit. 2022-12-29]. Dostupné z: https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/

[25] ŠTUPÁK, Branislav. *TRÉNUJEME AI: TENSORFLOW A KONVOLUČNÉ NEURÓNOVÉ SIETE V PRAXI A APPKE* [online]. 20.4.2022 [cit. 2022-12-29]. Dostupné z: https://www.eman.cz/blog/trenujeme-ai-tensorflow-konvolucne-neuronove-siete-praxi-appke/

[26] HORÁK, Karel. *Jasové transformácie* [online]. 2022 [cit. 2022-12-29]. Dostupné z: http://vision.uamt.feec.vutbr.cz/ZVS/lectures/05_Jasove_transformace.pdf. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií.

[27] HORÁK, Karel. *Geometrické transformácie* [online]. 2022 [cit. 2022-12-29]. Dostupné z: http://vision.uamt.feec.vutbr.cz/ZVS/lectures/06_Geometricke_transformace.pdf. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií.

[28] Spring Conference on Computer Graphics [online]. 2014 [cit. 2022-12-30]. Dostupné z: http://www.sccg.sk/~ftacnik/IP-5.pdf

[29] ADITYA, Mittal. Haar Cascades Explained. *Medium* [online]. 21.10.2020 [cit. 2023-05-02]. Dostupné z: https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d

[30] ABHISHEK, Jaiswal. Object Detection Using Haar Cascade: OpenCV. *Analytics Vidhya* [online]. 1.4.2022 [cit. 2023-05-02]. Dostupné z: https://www.analyticsvidhya.com/blog/2022/04/object-detection-using-haar-cascade-opencv/

[31] ZVORNICANIN, Enes. What is YOLO Algorithm?. *Baeldung* [online]. 4.11.2022 [cit. 2023-05-02]. Dostupné z: https://www.baeldung.com/cs/yolo-algorithm

[32] SOVIT, Rath. YOLOv5 – Custom Object Detection Training. *LearnOpenCV* [online]. 19.4.2022 [cit. 2023-05-02]. Dostupné z: https://learnopencv.com/custom-object-detection-training-using-yolov5/#What-is-YOLOv5?

[33] JOCHER, Glenn. Yolov5. *GitHub* [online]. [cit. 2023-05-02]. Dostupné z: https://github.com/ultralytics/yolov5

[34] JOCHER, Glenn. *Val loss is smaller then Train loss* [online]. 17.3.2022 [cit. 2023-05-10]. Dostupné z: https://github.com/ultralytics/yolov5/issues/7017

[35] SOLAWETZ, Jacob. What is YOLOv5? A Guide for Beginners. *Roboflow* [online]. 29.6.2020 [cit. 2023-05-02]. Dostupné z: https://blog.roboflow.com/yolov5-improvements-and-evaluation/#what-is-yolov5

[36] SHAH, Deval. Mean Average Precision (mAP) Explained: Everything You Need to Know. *V7* [online]. 7.3.2022 [cit. 2023-05-02]. Dostupné z: https://www.v7labs.com/blog/mean-average-precision

[37] ANWAR, Aqeel. What is Average Precision in Object Detection & Localization Algorithms and how to calculate it?. *Medium* [online]. 13.5.2022 [cit. 2023-05-02]. Dostupné z: https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b

[38] Spring Conference on Computer Graphics [online]. 2014 [cit. 2023-05-10]. Dostupné z: http://sccg.sk/~ftacnik/IP-4.pdf

[39] TOTH, Štefan. *SPRACOVANIE OBRAZU S VYUŽITÍM DOPYTOV V PROSTREDÍ VANET* [online]. Žilina, 2013 [cit. 2023-05-10]. Dostupné z: https://www.fri.uniza.sk/uploads/phd/13f-Toth_praca.pdf. Dizertačná práca. Žilinská univerzita v Žiline. Vedoucí práce Doc. Ing. Emil Kršák, PhD.

[40] HRÚZ, Marek. *SIFT,SURF,MSER* [online]. Plzeň, 2015 [cit. 2023-05-10]. Dostupné z: https://www.kky.zcu.cz/uploads/courses/mpv/04/materialy04.pdf. Západočeská univerzita v Plzni, Katedra Kybernetiky.

[41] LIETAVCOVÁ, Zuzana. *KONVOLUČNÍ NEURONOVÉ SÍTĚ* [online]. Brno, 2018 [cit. 2023-05-10]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=180844. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačných technologií. Vedoucí práce Doc. Ing. FRANTIŠEK V. ZBOŘIL, CSc.

[42] *Uctronics* [online]. Čína, 2022 [cit. 2023-05-11]. Dostupné z: https://www.uctronics.com/

[43] Nové usporiadanie a rozmery zvislých dopravných značiek. *Dopravné značenie s.r.o* [online]. Trnava, 2019, 28.5.2020 [cit. 2023-05-11]. Dostupné z: https://www.dopravneznaceniesro.sk/hlavne-informacie-znacky/

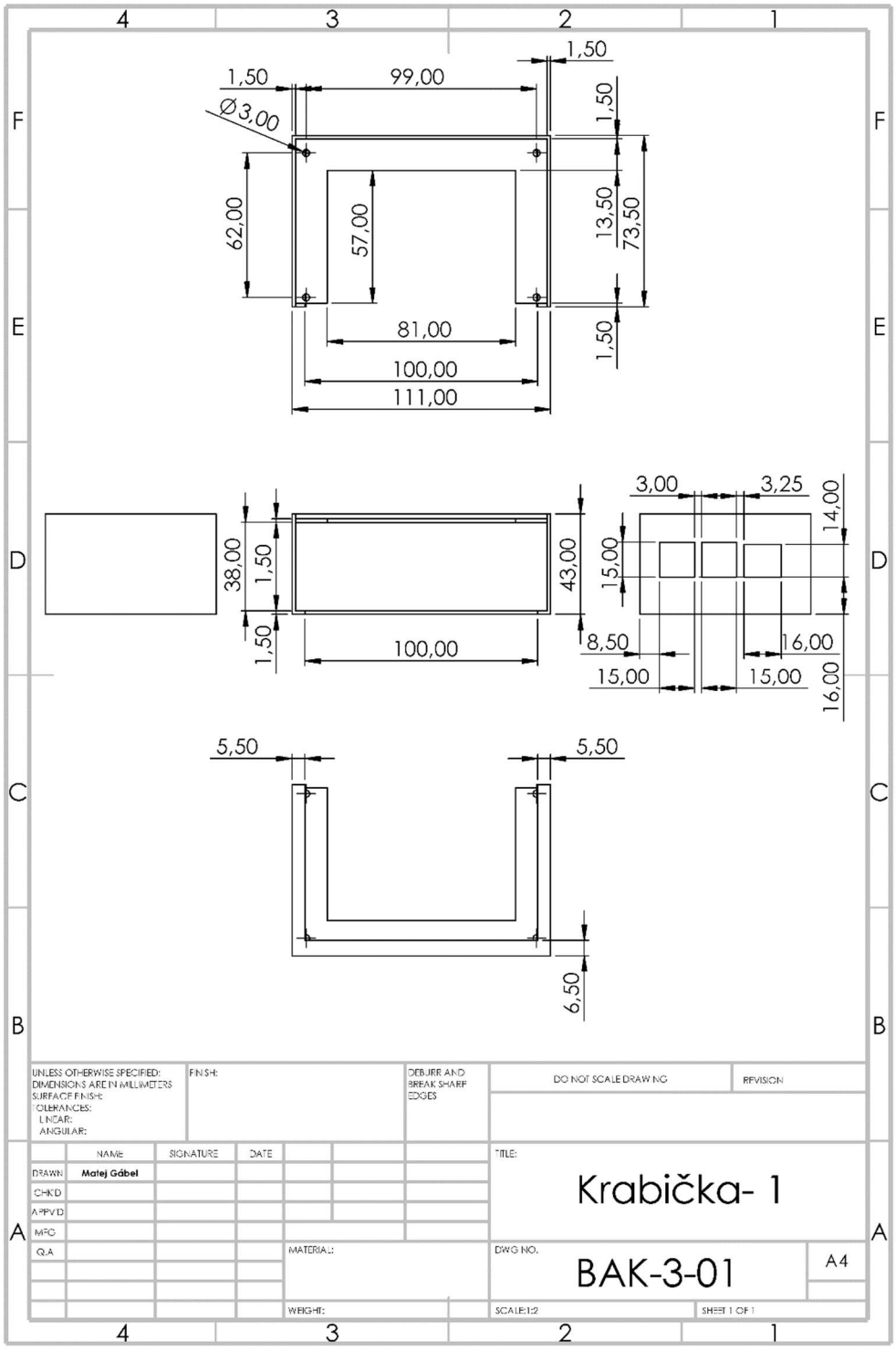
[44] Computer Vision with Python OpenCV. In: *Batoi* [online]. 2010, 5.3.2021 [cit. 2023-05-11]. Dostupné z: https://www.batoi.com/blogs/developers/computer-vision-python-opencv-604280ea054f8

[45] Implementing Real-time Object Detection System using PyTorch and OpenCV. In: *Morioh* [online]. 2022 [cit. 2023-05-11]. Dostupné z: https://www.google.com/search?q=opencv+and+pytorch&rlz=1C1GCEA_enCZ1024CZ1024&sxsrf=APwXEdep9CTo6VyyktPURbEmARzc0lE00A:1683802387853&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjnrczZjO3-AhXtS_EDHTk-D00Q_AUoAnoECAEQBA&biw=2133&bih=1032&dpr=0.9#imgrc=F2qwCxiekkQsuM

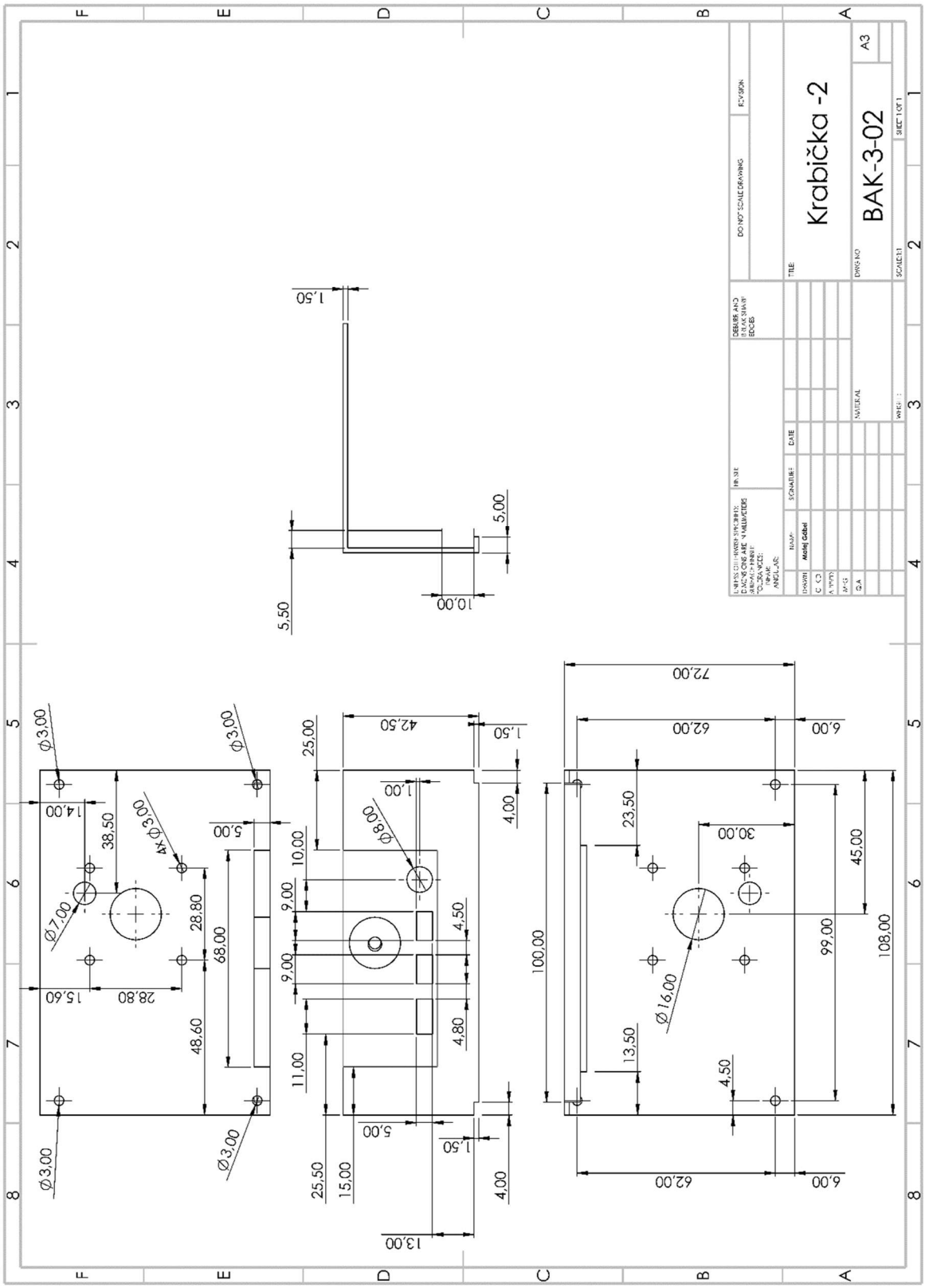
ZOZNAM PRÍLOH

Příloha A - Výrobné výkresy krabičky

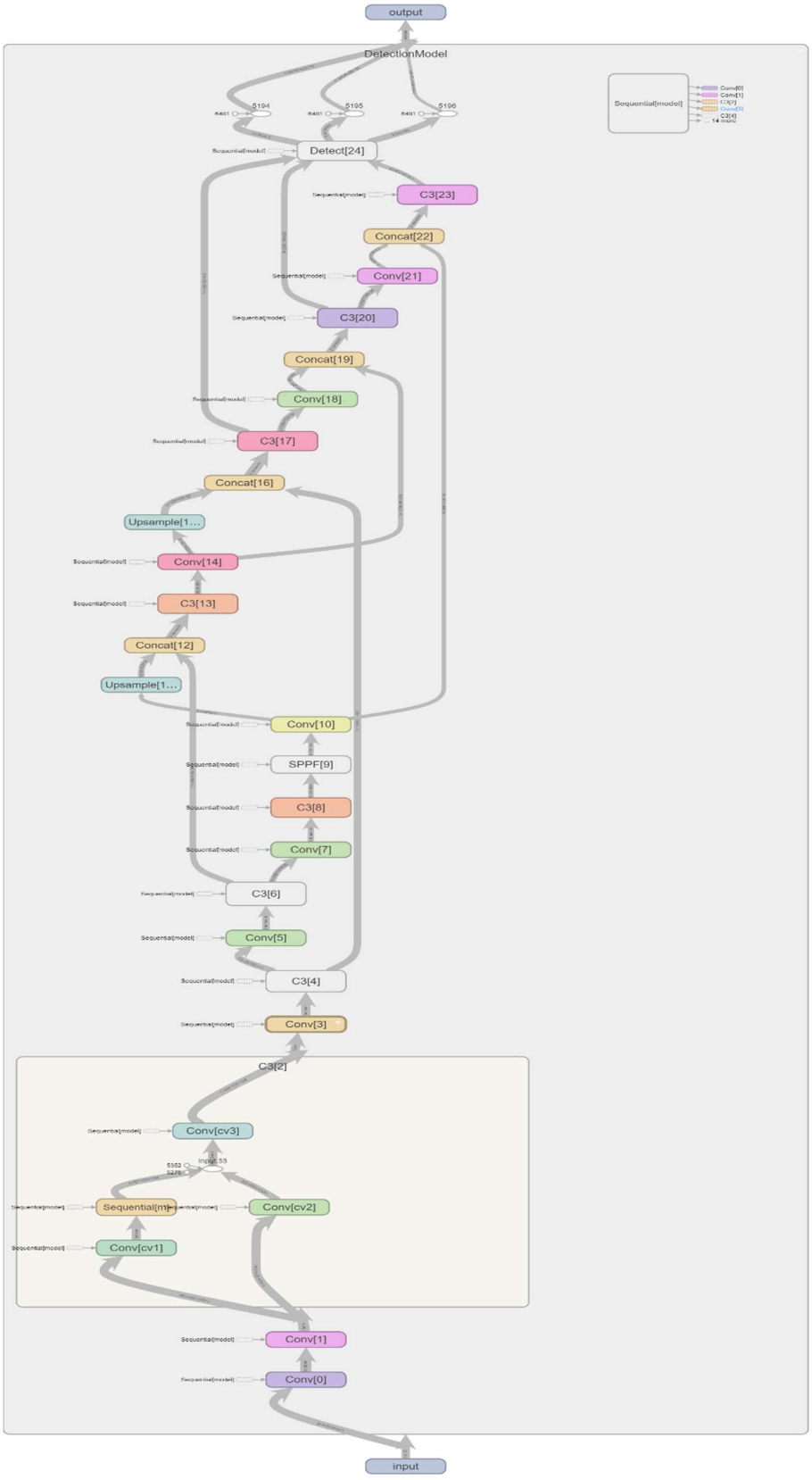
A.1 Krabička- 1. časť



A.2 Krabička- 2. časť



Příloha B - Náhlad modelu



Príloha C - Obsah elektronickej prílohy

Jednotlivé zložky obsahujú algoritmy detekcie a rozpoznávania značiek

```
|  README.txt
|
+---CNN_own
|      CNN_own.ipynb
|      model.pt
|
+---GUI
|  |  GUI.py
|  |  yolov5pic.png
|  |  yolo_logo.png
|  |
|  \---yolov5
|      +---.github
|      +---classify
|      +---data
|      +---models
|      +---runs
|      |  \---train
|      |      +---exp
|      |      |  |  confusion_matrix.png
|      |      |  |  F1_curve.png
|      |      |  |  labels.jpg
|      |      |  |  labels_correlogram.jpg
|      |      |  |  PR_curve.png
|      |      |  |  P_curve.png
|      |      |  |  results.csv
|      |      |  |  results.png
|      |      |  |  R_curve.png
|      |      |  |  train_batch0.jpg
|      |      |  |  train_batch1.jpg
|      |      |  |  train_batch2.jpg
|      |      |  |  val_batch0_labels.jpg
|      |      |  |  val_batch0_pred.jpg
|      |      |  |  val_batch1_labels.jpg
|      |      |  |  val_batch1_pred.jpg
|      |      |  |  val_batch2_labels.jpg
|      |      |  |  val_batch2_pred.jpg
|      |      |  |
|      |      |  \---weights
|      |      |          best.pt
|      |      |          last.pt
|      |      |
|      |      \---exp2
|      |          |  confusion_matrix.png
|      |          |  F1_curve.png
|      |          |  labels.jpg
|      |          |  labels_correlogram.jpg
|      |          |  PR_curve.png
|      |          |  P_curve.png
|      |          |  results.csv
```

```
|       |           |  results.png
|       |           |  R_curve.png
|       |           |  train_batch0.jpg
|       |           |  train_batch1.jpg
|       |           |  train_batch2.jpg
|       |           |  val_batch0_labels.jpg
|       |           |  val_batch0_pred.jpg
|       |           |  val_batch1_labels.jpg
|       |           |  val_batch1_pred.jpg
|       |           |  val_batch2_labels.jpg
|       |           |  val_batch2_pred.jpg
|       |           |
|       |           \---weights
|       |                   best.pt
|       |                   last.pt
|       |
|       +---segment
|       +---utils
|       \---__pycache__
+---Haar Cascade
|       cascade_stop_sign.xml
|       Haar.py
|       main_road.xml
|       yieldsign12Stages.xml
|
+---YOLOv5_own_script
|   |   IMG_6900_640.mp4
|   |   real_time.py
|   |   Traffic_sign_dataset.zip
|   |
|   \---yolov5
|       +---.github
|       +---classify
|       +---data
|       +---models
|       +---runs
|       |   \---train
|       |       \---exp
|       |           |  confusion_matrix.png
|       |           |  F1_curve.png
|       |           |  labels.jpg
|       |           |  labels_correlogram.jpg
|       |           |  PR_curve.png
|       |           |  P_curve.png
|       |           |  results.csv
|       |           |  results.png
|       |           |  R_curve.png
|       |           |  train_batch0.jpg
|       |           |  train_batch1.jpg
|       |           |  train_batch2.jpg
|       |           |  val_batch0_labels.jpg
|       |           |  val_batch0_pred.jpg
|       |           |  val_batch1_labels.jpg
|       |           |  val_batch1_pred.jpg
```

```
|       |       |   val_batch2_labels.jpg
|       |       |   val_batch2_pred.jpg
|       |       |
|       |       \---weights
|       |               best.pt
|       |               last.pt
|       |
|       +---segment
|       +---utils
|       \---__pycache__
\---YOLOv5_Training_Detection_script
    |   traffic.ipynb
    |   Traffic_sign_dataset.zip
    |
    \---yolov5
        +---.github
        +---classify
        +---data
        +---models
        +---runs
        |   \---exp
        |       |   confusion_matrix.png
        |       |   F1_curve.png
        |       |   labels.jpg
        |       |   labels_correlogram.jpg
        |       |   PR_curve.png
        |       |   P_curve.png
        |       |   results.csv
        |       |   results.png
        |       |   R_curve.png
        |       |   train_batch0.jpg
        |       |   train_batch1.jpg
        |       |   train_batch2.jpg
        |       |   val_batch0_labels.jpg
        |       |   val_batch0_pred.jpg
        |       |   val_batch1_labels.jpg
        |       |   val_batch1_pred.jpg
        |       |   val_batch2_labels.jpg
        |       |   val_batch2_pred.jpg
        |       |
        |       \---weights
        |               best.pt
        |               last.pt
        |
        +---segment
        +---utils
        \---__pycache__
```