Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Identifikace objektů v obraze se zaměřením na aplikace v dopravě

Bakalářská práce

Daniel Hejplík

Vedoucí práce: Ing. Jiří Jelínek, CSc.

České Budějovice 2019

Hejplík, D., 2019: Identifikace objektů v obraze se zaměřením na aplikace v dopravě. [Object identification in image with focus on application in traffic. Bc. Thesis, in Czech] – 57 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace:

Bakalářská práce se skládá ze dvou částí, teoretické a praktické.

Teoretická část práce se zabývá zkoumáním a porovnáním algoritmů a principů využívaných pro rozpoznávání objektů v digitálním obraze. Výsledkem je vyhodnocení jejich vhodnosti pro využití v dopravě. Hlavním kritériem hodnocení je rychlost jednotlivých metod a spolehlivost identifikace objektů.

Praktická část bude zaměřena na vytvoření konkrétního softwarového nástroje v jazyce Java, založeném na rozpoznávání objektů s využitím známých algoritmů, především knihovny OpenCV. Program bude obsahovat vlastní grafické rozhraní. Součástí výsledku budou dokumentace pro administrátory a koncové uživatele.

Annotation:

This bachelor´s thesis has two parts, theoretical and practical.

Theoretical part focuses on research and comparsion of algorithms and principles used for object recognition in digital images. The result is evaluation of their suitability for usage in traffic. Main evaluation criterion is speed of individual methods and reliability of object identification.

Practical part focuses on creating specific software tool in Java language, based on object recognition using known algorithms, primarily library OpenCV. The program will include its own graphical interface. Part of result will be documentation for administrators and end users.

Prohlášení:

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 10.4 2018 Daniel Hejplík

Poděkování:

Děkuji vedoucímu bakalářské práce, Ing. CSc. Jiřímu Jelínkovi, za cenná doporučení, trpělivost a poskytnutí odborné literatury k tématu.

Obsah

1. Úvod

Počítačové vidění je jednou z nejvíce se rozvíjejících technologií dnešní doby. Cílem počítačového vidění je rozpoznávání objektů, prostředí a jejich identifikace na základě vizuálních dat. Tento proces se pokouší napodobit vizuální schopnosti člověka a jeho způsob myšlení [1]. Kvůli tomu princip počítačového vidění ve velkém využívá principu neuronových sítí, jejichž účelem je nechat počítač napodobit postup myšlení a dojít k řešení sám, ne za pomoci přeprogramovaného postupu.

Počítačové vidění analyzuje vstupní data, video záznam či statický obraz a získává z nich informace, které je poté schopen použít k identifikaci jednotlivých částí obrazu [1], či sledování pohybujícího se objektu. Této analýze předchází předzpracování, které se skládá z detekce hran, segmentace a dalších algoritmů. Samotná analýza je poté většinou prováděna pomocí umělé neuronové sítě, *k*-NN klasifikátoru, nebo využitím SVM [10]. Neuronovou síť jsme schopni naučit rozpoznávat obsah na základě trénovacích dat. Nemusíme proto znát samotný algoritmus pro identifikaci určitého objektu, ale místo toho jí poskytneme dostatek vzorových obrazů a síť se „naučí“ jejich rozpoznávání.

1.1. Cíl práce

Hlavním cílem mé bakalářské práce na téma „Identifikace objektů v obraze se zaměřením na aplikace v dopravě“ je vytvoření komentované rešerše analýzy algoritmů a jejich částí využívaných při zpracování obrazu, jejich využití pro identifikaci objektů se zaměřením na rozpoznávací úlohy v dopravě. U algoritmů je testována přesnost a kvalita rozpoznávání a jejich rychlost z důvodu potřeby online zpracovaní videí u autonomních a asistovaných vozidel.

Na základě výsledků testování je navržena a implementována samostatná aplikace v jazyce Java.

2. Počítačové vidění

2.1. Přehled problematiky

Počítačové vidění je typ počítačového software, který je schopný získávat informace z poskytnutého obrazu. Může se jednat o zpracování jednoho obrazu, například fotografie, videa nebo obrazů z více zdrojů najednou [2]. Tato kapitola se věnuje rozboru jednotlivých částí algoritmů pro klasifikaci objektů a později ve 3. kapitole tyto znalosti budou využity pro pochopení komplexních postupů.

Nejčastější využití počítačového vidění je v oblastech:

- detekce objektů.
- detekce změn (například pohyb u bezpečnostních kamer),
- třídění a organizace obrazových dat na základě obsahu,
- interakce se člověkem (například mechanizmus Turningova testu – captcha),
- ovládání chování (pro umělou inteligenci),
- vyhledávání vzorů.

2.2. Historie

Počítačové vidění se začalo vyvíjet spolu s prvními pokusy o vývoj umělé inteligence, se záměrem naučit počítač napodobit lidský zrak [2]. V roce 1966 byl prvním takovým pokusem „The Summer Vision Project“, ve kterém se počítač s připojenou kamerou, pokusil technicky popsat, co vidí.

Základy tohoto projektu se začaly rozvíjet v 70. letech minulého století a daly vzniknout novým oborům, a to sice oboru digitálního zpracování obrazu a oboru dnes nazývanému Computer Vision. V této době vznikly základy mnoha algoritmů, které jsou běžně využívány pro identifikaci a detekci objektů.

Během 80. let byl největší pokrok ve vývoji počítačového vidění především v matematických analýzách, které byly nově založeny na matematických maticích. Pomocí matic mohl počítač získat číselná data schopná popsat prostorová měřítka jednotlivých objektů. Dokázal tak odvodit tvar těchto objektů, a to vše na základě různých podnětů jako je například stínování nebo obrysy modelů [3]. To vše mohl počítač snadno zjistit díky odlišnostem v barvách pixelů v různých částech objektů.

Velkým pokrokem pro počítačové vidění byl samozřejmě průlom v technologii kamer a rozvoj počítačové grafiky, ke které docházelo v průběhu 90. let. V té době také začaly výzkumy v 3D grafice, které dopomohly k lepšímu pochopení kalibrací prostoru, kde byla snaha o vytvoření trojrozměrné reprezentace prostoru z více „standardních“ obrazů. Dnes jsme již schopni sestavit 3D prostor z více 2D obrazů.

2.3. Klasifikace objektů

Postup na identifikaci objektů se nazývá Objekt classifier. Ten dostane na vstupu obraz či sérii obrazů a jako výstup poskytne klasifikační popisek, co na obrazu nalezl. Příklady jeho výsledku mohou být názvy objektů jako například „pes“, „stůl“, ale také jím může být popis prostředí jako „moře“ či „pole“.

Jak ale tyto algoritmy mohou rozpoznat obsah obrazů? Nejprve se algoritmus musí naučit nalézt rozdíly mezi jednotlivými třídami, které má klasifikovat. To znamená, že algoritmus umí rozpoznat pouze ty objekty, na které byl naučen. Samotné učení probíhá tak, že poskytneme algoritmu velké množství (tisíce) obrazů obsahujících objekt, který ho chceme naučit identifikovat a druhou sadu obrazů, které naopak tento objekt neobsahují. Tomuto principu se říká two-class nebo také binary klasifikátor. Z toho vychází, že takto vytvořený klasifikátor se naučí „rozpoznat“ pouze jeden objekt. Pokud je potřeba rozpoznávání většího množství tříd, lze využít většího množství takovýchto klasifikátorů. Komplexní klasifikátor se na pozadí tedy skládá z většího množství jednoduchých binary klasifikátorů a je tak schopen současně určit shodu testovaného objektu s větším množstvím tříd.

2.3.1. Předzpracovaní

Jako první krok se většinou vstupní obraz předzpracuje, aby s ním mohl poté počítač lépe pracovat. Kroky předzpracování jsou nejčastěji normalizace kontrastu a jasu, či odečtení střední intenzity obrazu. Občas se dá dosáhnout lepších výsledků pomocí změny barevného prostoru (RGB na LAB).

Tyto postupy nelze dělat automaticky, protože není předem zcela jasné, které kroky budou mít dobré výsledky. Většinou se proto postupuje principem pokus – omyl a zkouší se různé kombinace na stejném obraze a až poté se zvolí ten postup, který poskytne nejlepší výsledek.

Nedílnou součástí předběžného zpracovaní obrazu je také jeho oříznutí a změna velikosti. Velikost se nastavuje na pevně danou, jelikož následující krok, extrakce vlastností, se vždy provádí na obrazech s pevně danou velikostí.

2.3.1.1. Kernel

Nedílnou součástí předzpracování, i některých metod na extrakci vlastností, je kernel. Kernelu se také říká maska nebo konvoluční matice. Je to jednoduchá matice vždy čtvercového tvaru, která převážně mívá lichý počet prvků v řádcích a sloupcích. Nejčastěji vidíme kernely velikostí 3×3 nebo 5×5, větší matice bývají velmi neobvyklé [3].

Konvoluční matice jsou využity k cílené úpravě obrazu. Tento proces spočívá v přepočtení každého pixelu na základě hodnot jeho sousedů. Na základě matice 3×3 z následujícího obrázku můžeme spočítat jeho nové hodnoty tak, že se konvoluční matice nasadí středem na právě počítaný pixel (v našem příkladu je nad počítaným pixelem hodnota 5). Poté se vezmou všechny pixely, přes které je umístěna matice a jejich hodnoty se vynásobí odpovídající hodnotou z matice. Tyto hodnoty se poté sečtou jako nová hodnota počítaného (prostředního) pixelu.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} a1 & b1 & c1 & d1 & e1 \\ a2 & b2 & c2 & d2 & e2 \\ a3 & b3 & c3 & d3 & e3 \\ a4 & b4 & c4 & d4 & e4 \\ a5 & b5 & c5 & d5 & e5 \end{bmatrix}$$

Obrázek 1 - Příklad konvoluční matice a zdrojového obrazu (jehož pixely jsou reprezentovány proměnnými a1-e5)

Příklad ze zadaného obrázku:

$$\begin{aligned} c2 = {} & (1 * b1) + (2 * c1) + (3 * d1) + \\ & (4 * b2) + (5 * c2) + (6 * d2) + \\ & (7 * b3) + (8 * c3) + (9 * d3) \end{aligned}$$

Rovnice 1 - Výpočet pixelu na základě kernelu

2.3.1.2. Gaussův filtr

Jednou z nejvíce používanou součástí předzpracování může být odstranění šumu. K tomu se nejčastěji využívá Gaussova filtru, také nazývaného Gaussovo vyhlazení [4]. Tento postup přepočte hodnotu všech pixelů obrazu tak, aby byly odstraněny prudké změny v obraze. Tím se docílí ztráty šumu, ale současně i detailů obrazu.

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp\left(-\frac{\left(i-(k+1)\right)^2 + \left(j-(k+1)\right)^2}{2\sigma^2}\right); 1 \le i,j \le (2k+1)$$

Rovnice 2 - Gaussův filter

Na základě této rovnice můžeme vytvořit konvoluční matici, kterou vyhladíme obraz. Rovnice obsahuje dvě proměnné hodnoty, které jsou na sobě závislé. Jednou je velikost matice, která je dána hodnotou 2k+1, jelikož musí být vždy lichá a druhou hodnota σ.

Je důležité vědět, jaký vliv tyto hodnoty mají na obraz. Velikost filtru, nejčastěji 5×5, ovlivňuje sílu vyhlazení obrazu a tím i množství ztracených detailů. Hodnota σ silně ovlivňuje přesnost, kterou vyhlazení poskytne a tím může zhoršit detaily ve výsledném obraze. Je závislá na velikosti filtrovací matice a její hodnota se může lišit v závislosti na požadovaném výsledku Gaussova filtru. Většinou se udává, že $r = \sigma * 2$, kde radius je polovina velikosti matice $r = 2k + 1$. Pro σ = 1, by tedy matice měla být 4×4, ale je velice výhodné, aby měla matice liché hodnoty, proto tedy velikost 5×5. Při nastavování síly vyhlazení většina programů dává uživateli možnost ovládat hodnotu σ a z ní vypočítá velikost matice.

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A$$

Konvoluční matice 1 - Gaussův filtr s velikostí 5×5 a hodnotou σ=1.4

Obrázek 2 - Původní obraz

Obrázek 3 - Gaussian blur obraz

2.3.2. Extrakce vlastností

Vstupní obrazy většinou obsahují velké množství nepotřebných informací a proto je nutné informace v obraze zjednodušit, aby s nimi mohl algoritmus lepé pracovat. Volba a navržení těchto funkcí mají jeden z největších vlivů na výkon výsledného algoritmu.

Základní funkcí pro zjednodušení informace v obraze je vyhledání rozdílů v RGB jednotlivých pixelů nebo jejich seskupení. Tento princip je však velmi pomalý a náročný, proto byly vytvořeny lepší a rychlejší algoritmy pracující s detekcí hran objektů [9] (Oriented gradients).

Vezměme jako příklad rozpoznání knoflíků na oblečení. Pomoci detekce hran můžeme snadno identifikovat kruhový obrys knoflíku, ale bez dalších informací by poté algoritmus nebyl schopen najít rozdíl mezi knoflíkem a například kolem od auta. Proto jsou na extrakci vlastností využívány mnohem sofistikovanější funkce, kterých existuje velké množství a které mají různé výhody i nevýhody.

2.3.2.1. Canny edge detektor

Jak již bylo řečeno, jedním z nejjednodušších způsobů získávání informací z obrazu, je detekování hran. I když na detekci hran existuje větší množství postupů, všechny tyto postupy fungují na velmi podobných principech. Tím nejčastěji používaným je Canny edge detektor, který vymyslel John F. Canny v roce 1986 [5].

Hrany by měly být v obraze detekovány přesně, s co nejmenším množstvím chyb. Navíc by každá hrana měla být označena, nejlépe bodem nacházejícím se zhruba ve středu této hrany.

Na následujícím obrázku si předvedeme postup Canny detektoru.

Obrázek 4 - Výchozí obraz Canny edge detekce

Postup algoritmu Canny se dá rozdělit do 5 kroků:

1. Aplikace Gaussova filtru

   Jedním z největších problémů pro detekci hran je šum v obraze. Proto prvním krokem musí být jeho odstranění nebo alespoň co největší snížení. Proto zde využijeme Gaussova vyhlazení (2.3.1.2.).

Obrázek 5 - Vyhlazený obraz Canny edge detekce

2. Nalezení intenzity gradientů (první derivace)

   Po vyhlazení obrazu můžeme přejít k detekci hran, ke které se využívá filtrace přes dvě konvoluční matice. Tyto matice spočtou přibližnou hodnotu první derivace, jedna počítá horizontální změny, zatímco ta druhá, vertikální.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Konvoluční matice 2 - Sobel matice pro detekci hran
(G_x – vertikální | G_y – horionzální)

   Hodnota A v předchozích maticích je obraz, na kterém detekujeme hrany a G_x/G_y jsou nově vzniklé obrazy obsahující odhad vertikální či horizontální derivace. Z těchto matic lze poté zjistit velikost a směr nalezených hran.

   Pomoci Euklidovské vzdálenosti můžeme vypočítat hodnotu hrany

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Rovnice 3 - Vzdálenost gradientů

   a úhel, na který hrana směřuje, se spočítá pomocí rovnice

$$\theta = \text{atan2}(G_x, G_y)$$

Rovnice 4 - Úhel gradientů

   Výsledný úhel se poté zaokrouhluje na jeden ze čtyř směrů – vertikální (0°), horizontální (90°) a dvě diagonály (45° a 135°).

Obrázek 6 - Obraz gradientů Canny edge detekce

3. Ztenčení hran

   Dalším krokem je technika na ztenčení hran. Princip non-maxima suppresion porovnává tloušťky hrany na pozici zvoleného pixelu s tloušťkou hrany v positivním a negativním směru hrany, ke které pixel patří. Pokud je momentální pixel největší v porovnání s ostatními, je jeho hodnota zachována, jinak je potlačena.

Obrázek 7 - Obraz po ztenčení hran Canny edge detekce

4. Aplikovaní prahu pro potencionální hrany

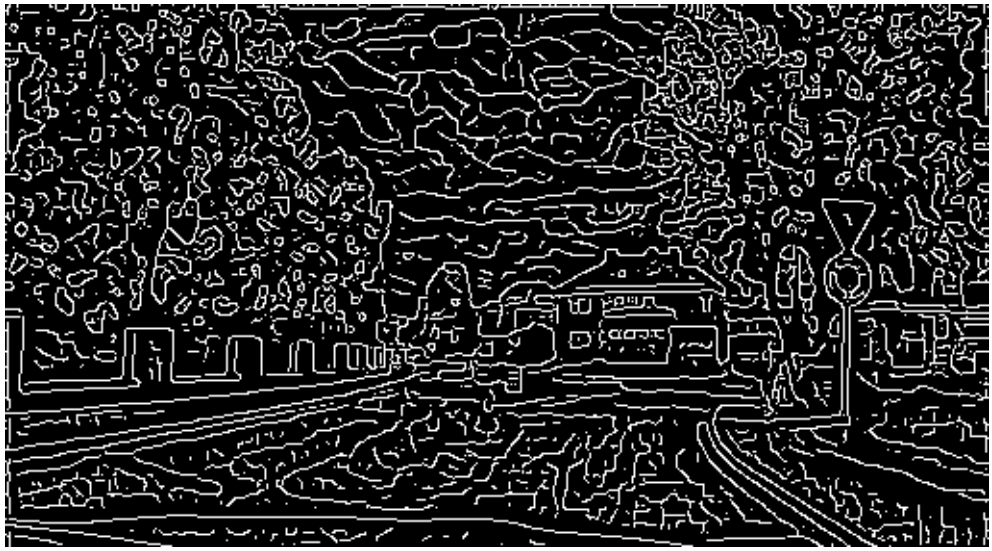
   Po provedení předchozího kroku by měl obraz obsahovat už jen reprezentaci reálných hran původního obrazu. Občas však zůstanou pixely, či celé hrany, které jsou způsobeny šumem, nebo rozdíly v barvách obrazu. Tyto „nechtěné“ hrany lze vyfiltrovat tak, že se zvolí limitní hodnoty – dolní a horní limit.

   Pokud je gradient pixelu větší než horní limit, je tato hrana označena jako silná. Je-li gradient menší než dolní limit, je hrana zcela potlačena. Hodnota nacházející se mezi horním a dolním limitem je označena jako slabá.

5. Finální detekce dostatečně silných hran

   Finálním krokem je detekování hran, které lze považovat za dostatečně silné. Nejslabší pixely byly označeny již v přechozím kroku, takže zbývá odstranění slabých bodů, jež nejsou důležitou součástí skutečných hran. Toho lze docílit jednoduchou kontrolou, zda bod označený za slabý má mezi 8 nejbližšími sousedními body takový, jež je označen jako silný. Pokud ano, je zachován, jinak je odstraněn. Na konci tohoto procesu máme nový obraz, který je velmi přesnou reprezentací hran obrazu původního.

Je důležité správně zvolit hodnoty dolního a horního limitu. Pokud by byly nevhodně zvolené, může dojít k vyhledání i extrémně slabých hran. Jako příklad lze uvést krajní extrém – dolní limit 0 %, horní limit 5 %.



Obrázek 8 - Obraz po aplikaci nevhodně zvolených limitů Canny edge detekce

Viditelně vhodnější výsledek poskytnou hodnoty parametrů – dolní limit 24 %, horní limit 15 %.

Obrázek 9 - Obraz po aplikaci kvalitně zvolených limitů Canny edge detekce

2.3.2.2. Vyhledání popředí obrazu

Vyhledání popředí obrazu se pokouší o nalezení změn popředí v sekvenci obrazů. Princip se také nazývá Background substraction, jelikož je založen na zaznamenání statických částí obrazu a jejich následném odečtení.

Nejprve se tedy zjistí pozadí obrazu, tj. stav, ve kterém jsou na obraze pouze věci, které nechceme detekovat, a od tohoto stavu se dále zjišťují rozdíly. Tím se získá popředí obrazu. Z principu, na kterém tento postup pracuje, je zcela zřejmý problém při použití videa z pohybující se kamery. Naproti tomu velmi dobré využití je ale u statických kamer, tj. např. u bezpečnostních nebo dopravních kamer [6].

Vyhledání popředí není ale v pohybujících se obrazech zcela nepoužitelné. Výsledek může sice obsahovat vyšší množství šumu, při správné volbě limitních hodnot lze však tato data velmi kvalitně použít na doplnění jiných algoritmů o informace o pohybech objektů.

Obrázek 10 - Zdrojový obraz pro Background substraction

Obrázek 11 - Výsledný obraz Background substraction

V předchozím obraze vidíme výsledná data detekce, kde bílé pixely obrazu reprezentují objekty, které byly vyhodnoceny jako měnící svou pozici, černé pixely byly vyhodnoceny tak, že jejichž změny nejsou dle nastavených parametrů důležité.

1. Nalezení pozadí

   Nejjednodušším způsobem k nalezení pozadí je pozadí algoritmu prostě ručně poskytnout. V prostorách, jako například uvnitř budov, se tím docílí velké jistoty obsahu pozadí. Ve venkovních prostorách však může nastat problém při rozdílu světelných podmínek v poskytnutém pozadí a reálném momentě v obraze nebo v pohybu drobných objektů jako jsou například listy.

   Dalším způsobem je využití dočasného pozadí, které se vypočítává za průběhu algoritmu. Toto dočasné pozadí se vytvoří tak, že se každý jeho pixel spočte jako medián pixelů na jeho pozici v posledních X obrazech, kde X určuje jakousi paměť, kterou pozadí má. Tato hodnota může pomoci snížit drobné pohyby, jakými může být vítr v trávě nebo dlouhotrvající pomalý pohyb, jako třeba pohyb stínů. Na začátku je třeba vzít v úvahu určitý čas tréninku, během kterého se počítají mediány prvního dočasného pozadí. Tato doba je dána množstvím obrazů, ze kterých se medián počítá.

2. Detekce popředí

   Detekce popředí dále tedy spočívá v nalezení rozdílů mezi pozadím a reálným obrazem. Odečtením pixelu z reálného obrazu v určitém čase $P[I(t)]$ od obrazu pozadí $P[B]$ se získá obraz popředí v onom čase $P[F(t)]$

$$P[F(t)] = P[I(t)] - P[B]$$

Rovnice 5 - Spočtení popředí obrazu (t=čas)

   Díky tomu, že pozadí je předpočítáno z několika předchozích obrazů, se získá velmi přesné popředí obrazu. Toto ale funguje jen v případech, kdy na pozadí neexistují pixely, které mění svoji pozici. Takovéto drobné pohyby by mohly způsobit velký šum v nalezeném popředí.

3. Limitní práh

   Ke snížení vlivu drobných pohybů se využije principu limitních hodnot. Jednoduše si určíme prahovou hodnotu, a pokud je rozdíl pixelů ve dvou po sobě jdoucích obrazech nižší než tento limit, je z výsledného popředí odstraněn

$$P[F(t)] - P[F(t+1)] > Limit$$

Rovnice 6 - Limitní práh

Obrázek 12 - Výsledek s limitním prahem 16

Obrázek 13 - Výsledek s limitním prahem 50

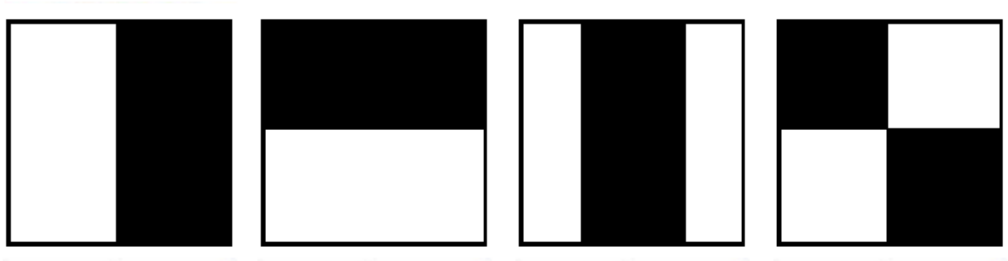
Obrázek 14 - Výsledek s limitním prahem 200

2.3.2.3. Haar-like features

Jedním z prvních komplexnějších algoritmů je Haar-like features algoritmus, představený v roce 2001 [8]. Tento algoritmus je schopen identifikovat jakýkoliv objekt, který se předem naučil a to i v reálném čase z videa. I přes to, že jej lze naučit rozpoznat jakékoliv visuální vlastnosti objektů, je nejvíce vhodný na detekci hran, čehož se dá vhodně využít při identifikaci obličejů, nebo pro tuto práci ve vhodnějším příkladu, identifikaci dopravního značení. Název Haar-like pochází z podobnosti s matematickými funkcemi Haar wavelet. Haar-like features je velmi podobné kernelu u neurálních sítí, kterým se bude více věnovat kapitola 2.3.3.3., rozdílem však je, že u Haar-like features není využito trénování, které se naučí tvar popisující objekt, ale jsou využity kombinace kernelů, jejichž tvar je předem určen.

Kernely jsou využity na vyhledání typických změn na pixelech obrazu a u Haar-like Features jsou předem dány. Popisují nejdůležitější tvary obecných objektů v obraze a jsou rozlišeny jako bílé a černé pixely. V reálném obraze samozřejmě nenacházíme pouze černé a bílé pixely, ale hodnoty, které se pohybují v určitém rozmezí. Haar-like vlastnosti však nehledají extrémní rozdíly černá a bílá, ale posun o určitou hodnotu v černo-bílé verzi obrazu.

Příklady kernelů vyhledávající hrany (1, 2), rovnou linii (3) a na posledním obrazci (4) je kernel pro nalezení diagonální linií [8]:



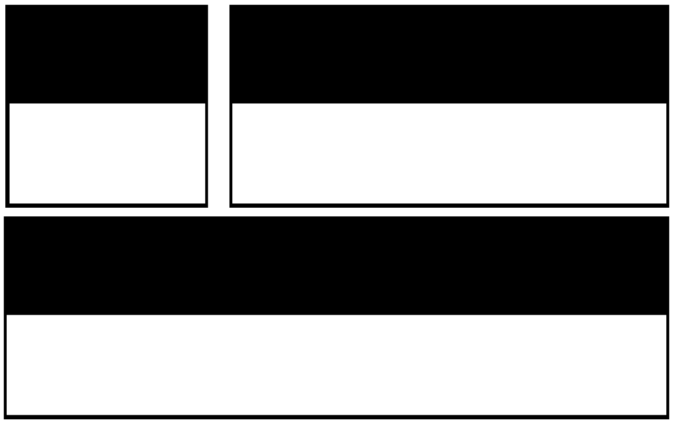
Obrázek 15 - Příklady Haar-like feature kernelů
(vyhledání vertikální hrany, horizontální hrany, rovné linie a diagonální linie)

Číselná reprezentace těchto kernelů by mohla vypadat takto:

$$\begin{bmatrix} -1 & -1 & 5 \\ -1 & -1 & 5 \\ -1 & -1 & 5 \end{bmatrix} \begin{bmatrix} 5 & 5 & 5 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} -1 & 5 & -1 \\ -1 & 5 & -1 \\ -1 & 5 & -1 \end{bmatrix} \begin{bmatrix} 5 & -1 & -1 \\ -1 & 5 & -1 \\ -1 & -1 & 5 \end{bmatrix}$$

Konvoluční matice 3 - Číselná reprezentace kernelů

Na předchozích příkladech vidíme kernely neboli konvoluční matice o velikosti 3×3, kterými po částech projdeme celý obraz. První kernel vyhledává vertikální hranu, která přechází ze světlé hrany (-1) do tmavších odstínů (5). Číselné hodnoty udávají sílu změny v barvách sousedících pixelů. Velikosti těchto vlastností a tím i jejich kernelů mohou být ale různé. Například všechny tři následující kernely jsou reprezentace stejné hrany, ale o jiné velikosti



Obrázek 16 - Haar-like feature o různých velikostech

Detektory založené na Haar-like features, mají určitou limitaci v tom, co dovedou rozlišit. Tímto omezením jsou pevně definované vlastnosti, které popisují jednotlivé Haar-like features. Problematickým příkladem může být částečné zakrytí objektu, pootočení úhlů, a to jak samotného objektu nebo i u kamery vytvářející obraz. Dá se tedy říci, že klasifikátory založené na Haar-like principu mají špatnou odolnost vůči nevhodným podmínkám ve srovnání s metodami založenými na trénování. Výhodou však je, že klasifikátor má velice malý soubor, dle kterého identifikuje objekty. Je potřeba pouze určit jaké jednotlivé Haar-like features mají být vyhledávány a jaká je jejich důležitost, tzn. musí se určit jejich váhy. Pro tyto zvolené Haar-like featuers poté určíme jejich relativní umístění mezi sebou navzájem. Je důležité vědět, že jednotlivé vlastnosti mohou být využity více než jednou.

Obrázek 17 - Příklad Haar-like freatures nalezených na značce „Stůj, dej přednost v jízdě!“

Zvolení vybraných vlastností, jejich rozměrů a vzdáleností lze automatizovat. Trénovací algoritmus pro návrh Haar-like features klasifikátoru je poměrně jednoduchý. Stačí mít pozitivní obrazy obsahující různé verze vyhledávaných objektů a obrazy negativní, které obsahují vše, co naopak vlastnosti popisovat nemají. Trénink poté probíhá v kolech, tzn. Stage, kde při každém kole se pokouší systém najít novou vlastnost, která má shodný vztah k již nalezeným vlastnostem ve všech positivních obrazech a zároveň nevytvoří falešný nález v negativních obrazech. Z tohoto postupu vyplývá, že prvotní vlastnosti jsou nalezeny velmi rychle, ale s každou přibývající doba nalezení nových stoupá.

2.3.2.4. Histogram of Oriented Gradients

Dalším principem pro nalezení informací v obraze je HOG descriptor, který je jedním z nejvhodnějších způsobů pro naši potřebu rychlé identifikace většího množství stejných objektů.

Popisovač HOG prvně představili Navneet Dalal a Bill Triggs v roce 2005 [9]. Tento postup je založen na možnosti velice efektivně popsat vzhled a tvar objektu v obraze pomocí distribuce (histogramu) detekovaných hran (oriented gradients). Tohoto je docíleno tak, že se obraz rozdělí na regiony nazývané buňky (cell), které obsahují jednorozměrný histogram směru hran v dané buňce. Kombinace těchto histogramů je poté popisem celého obrazu.

Na následujícím příkladu bude ukázáno, jak pro obraz 64×128 pixelů spočítat HOG popisovač (descriptor), ten bude reprezentován vektorem o velikosti 3780 (toto číslo bude později vysvětleno).

Obrázek 18 - Zdrojový obraz HOG descriptoru

1. Předzpracování

   Jako první krok by mělo dojít k předzpracování, přičemž u postupu HOG není vhodné využívat vyhlazení obrazu. HOG pracuje na obrazech o fixním poměru stran, nejlépe velikosti násobků 2. Jako příklad lze uvést poměr 1:2 a rozměry 64×128. Předzpracováním se proto rozdělí obraz na několik částí, které by se pro větší přesnost měly částečně překrývat a poté až dojde k oříznutí a přizpůsobení velikosti obrazu.

2. Výpočet gradientů

   HOG popisovač je založen na spočtení gradientů obrazu. Gradient pixelu je hodnota popisující sílu změn ve vertikálním či horizontálním směru. Proto tedy spočteme hodnoty g_x a g_y z původního obrazu tím, že se přefiltruje původní obraz následujícími kernely.

-1	0	1

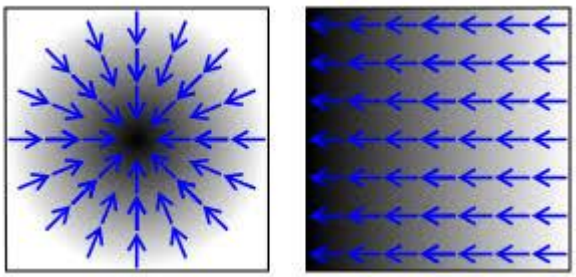
-1
0
1

Obrázek 19 - Kernely

Z těchto nově vzniklých obrazů (g_x a g_y) lze spočítat velikosti a orientaci gradientů za pomocí následujících rovnic, kde θ je úhel v rozmezí 0° až 180°:

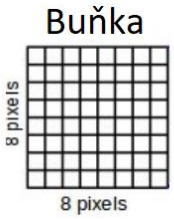
$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = arctan\frac{g_y}{g_x}$$



Obrázek 20 - Příklad gradientu

3. Rozdělení obrazu jednotlivé buňky o velikosti 8×8



Obrázek 21 - Buňka 8×8 pixelů

Každý pixel je reprezentován hodnotami tří barev. To znamená, že každou buňku lze reprezentovat $8 \times 8 \times 3 = 192$ hodnotami a gradientem, který se skládá ze dvou hodnot, velikosti a směru, tzn., obsahuje $8 \times 8 \times 2 = 128$ hodnot.

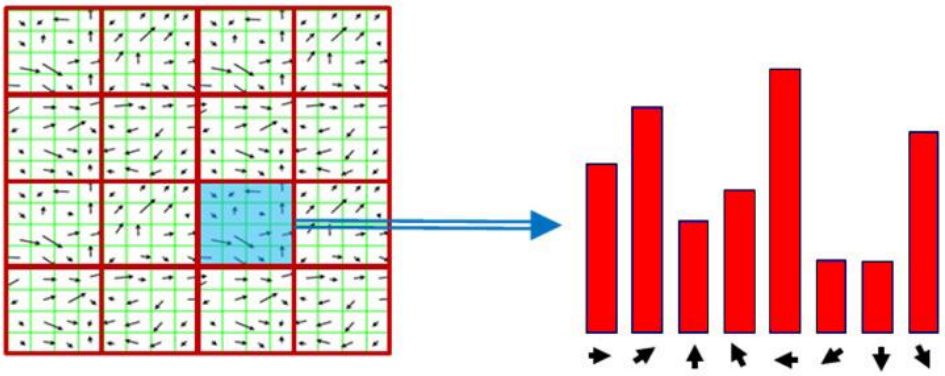
Obrázek 22 - Ukázka buňky HOG

4. Výpočet histogramu gradientů buněk

Hodnoty obsažené v buňce lze nyní přepočíst na data, která budou mnohem užitečnější na reprezentaci obrazu. Toho docílíme tak, že hodnoty překonvertujeme na histogram o 9 hodnotách. Histogram má 9 hodnot z toho důvodu, že gradient je úhel v rozmezí 0° až 180°a tyto hodnoty se rozdělí do skupin, tzn. binů, které reprezentují skupinu hodnot. Nejčastějším rozdělením bývá 20° a z toho vychází, že binů je 9 a odpovídají gradientům směru v úhlech 0°, 20°, 40°…180°.

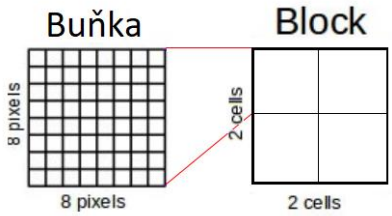
Každý pixel v buňce „hlasuje“ pro jeden či dva biny v histogramu. Pokud je velikost směru hrany na pixelu přesně 0°, 20°, …, je binu, reprezentujícímu tento úhel, zvýšena hodnota o velikosti gradientu právě hlasujícího pixelu. Pixely, jejichž hodnota není přesně ve stanovených úhlech, rozdělí svůj hlas mezi dva nejbližší biny a to na základě vzdáleností od těchto binů. Například: pixel, který má gradient o velikosti 2 a jeho úhel je 20°, hlasuje pro druhý bin (20°) hodnotou 2. Druhou možností je pixel také s gradientem o velikosti 2, ale jeho úhel je 30°. Tento pixel rozdělí svůj hlas dle

vzdálenosti od nejbližších binů (20° a 40°) a svůj hlas jim poměrově rozdělí. V našem případě tedy druhý bin (20°) a třetí bin (40°) dostanou hlas o hodnotě 1.



Obrázek 23 - Histogram gradientů

5. Normalizace bloků



Obrázek 24 - Vztah Buňky a Bloku

Histogram, který vznikl v předchozím kroku, není moc odolný proti změnám v osvětlení obrazu, to znamená, že jakékoliv změny v intenzitě obrazu změní i velikosti binů v histogramu. Pro odstranění těchto efektů se může histogram normalizovat. Pro normalizaci se každý element vydělí velikostí celého vektoru. Normalizace se nedělá nad jednotlivými buňkami, ale nad bloky 16×16 pixelů. To znamená, že na konci je vektor, který už nemá 9 elementů, ale má jich 36.

6. Vektor vlastností

Pro výpočet finálního vektoru, reprezentujícího celý obraz, se využije normalizovaný histogram tak, že jsou bloky 16×16 posouvány v krocích o velikosti 8, tj. polovina strany bloku. Výsledkem pak je 50% překrytí ve všech po sobě jdoucích krocích. Každým krokem je tak spočteno 36 čísel, které odpovídají 4 histogramům v bloku 16×16. Tato čísla postupně vytvářejí finální vektor. Velikost tohoto vektoru můžeme snadno odvodit. Postupujeme po krocích o velikosti 8 pixelů, proto můžeme na obrazu velikosti 64×128

udělat 7 kroků horizontálně a 15 kroků vertikálně. Celkem tedy uděláme 7×15=105 kroků, což odpovídá vektoru o velikosti 105×36=3780.

Na následujícím obrázku lze vidět mřížku buněk (zelená), kde krok č. 2 posouvání je označen červeně a krok č. 3 žlutě.

Obrázek 25 - Ukázka mřížky HOG vlastností

2.3.3. Klasifikace

Závěrečnou částí pří identifikaci objektů v obraze, občas nazývané rozpoznávání objektů v obraze, je samotná klasifikace. Hlavními problémy u klasifikace bývají nekompletní data v obraze. Těmi může být například špatný pozorovací úhel vůči objektu, částečné zakrytí objektu nebo velikost.

Nejzákladnějším, ale také nejvíce náchylným na nedostatky obrazu, bývá postup nazýván Edge matching, kde klasifikátor porovná předzpracovaná data z obrazu s předem zadanými vzorky dat a pokusí se najít co největší schodu. Tento postup má výhodu rychlosti při častém přidávání nových vzorků, ale jak již bylo zmíněno, není moc odolný proti chybám v datech.

Ať už se jedná o nejzákladnější klasifikaci nebo o sebesložitější postup, je vždy nutno předem „naučit“ software co a jak má klasifikovat. Historicky byly tyto klasifikátory naprogramovány ručně, to je však extrémně jednorázové a často náročné na programátora. Proto je dnes „ruční“ vývoj klasifikátoru prakticky nemožný.

2.3.3.1. *k*-Nearest Neigbours

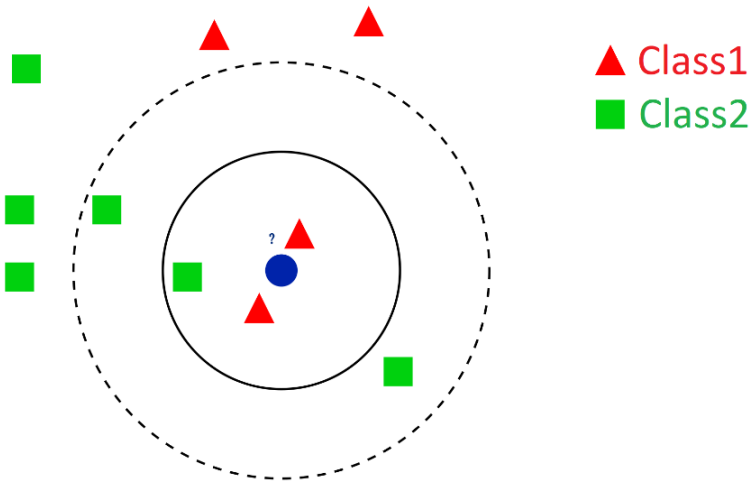
k-NN je algoritmus, který se využívá na klasifikaci či regresi vzorců (pattern). Algoritmus je založen na zařazení prvku do klasifikační kategorie na základně množství (*k*) blízkých vzorových příkladů.

Výstup se liší podle toho, zda je algoritmus využit na regresi či klasifikaci. Při regresi je výstupem přímá hodnota popisující objekt, kdy jako výsledek obdržíme průměrnou hodnotu *k* nebližších sousedů. Při využití algoritmu pro klasifikaci hlasují všichni z *k* nejbližších sousedů pro klasifikační skupinu, pod kterou spadají a výslednou hodnotou je skupina s největším počtem obdržených hlasů. Z těchto principů je pro tuto práci důležitá pouze klasifikační schopnost algoritmu *k*-NN.

Princip učení *k*-NN se dá nazvat jako instance learning nebo lazy learning, to znamená, že veškeré funkce jsou prováděny až při samotné klasifikaci. Ve skutečnosti se ani o učení nejedná. Trénovací fáze algoritmu je pouze vytvoření série vektorů, které reprezentují jednotlivé známé objekty a jejich přiřazení k odpovídajícím klasifikačním třídám. *k*-NN je tudíž velmi ovlivňován strukturou dat a díky tomu se může stát, že opakované testování stejného obrazu poskytne odlišné výsledky.

Klasifikační fáze algoritmu je založena na konstantní hodnotě *k,* která se musí dopředu ručně stanovit. Hodnota *k* určuje množství vektorů, které budou při klasifikaci brány v potaz. Poté se vytvoří nový neklasifikovaný vektor reprezentující identifikovaný objekt a *k*-NN mu přiřadí klasifikační třídu na základě nejvíce frekventované klasifikační třídy mezi *k* nejbližšími vektory. Vzdálenost vektorů se dá počítat několika způsoby. Nejjednodušší srovnávací hodnotou je Euklidovská vzdálenost, jejímž problémem ale je, že v obraze nelze snadno reprezentovat prostorovou vzdálenost mezi různými obrazy. Proto se využívá pro vyhledávání v obraze vhodnější způsob, Hammingova vzdálenost. Hammingova vzdálenost počítá počet kroků, které jsou potřeba, aby se bity reprezentující jeden datový objekt, v našem případě obraz, změnily na hodnotu druhého objektu. V obraze se to dá reprezentovat například vzdáleností barevných hodnot jednotlivých pixelů.

Hlavním optimalizačním problémem u algoritmu *k*-NN je volba parametru *k.* Základem je využití lichého čísla, aby se předešlo remíze mezi dvěma kategoriemi. Číslo samotné je často velmi ovlivněno velikostí a rozdílností trénovacích dat. Na následujícím příkladu je vidět rozdílný výsledek při různé velikosti *k.*



Obrázek 26 - Ukázka k-NN

Cílem je identifikovat nově přidávaný prvek, tj. modrý kruh. Pokud by hodnota *k* byla 1-3, výslednou hodnotou by byla identifikace jako Class1, ale pokud by se zvolila hodnota 5 a vyšší, výsledným identifikováním by se stala Class2. Proto je nutno zvolit hodnotu *k* cíleně na základě rozložení jednotlivých tříd v datech.

2.3.3.2. Support Vector Machine

Support Vector Machine, občas nazývané Support Vector Network, jsou učící se algoritmy založené na principu učení se s učitelem. Výsledkem je klasifikátor a regresní analýza, která rozdělí obsah do jedné ze dvou skupin, neboli binární klasifikátor. Originální SVM byl vytvořen ruským vývojářem Vladimir N. Vapnikem na začátku 90 let [11]. Původní využití bylo určeno pro rozpoznání textů či obličejů. Ve spojení s deskriptorem HOG (2.3.2.4.) lze tento princip využít k velmi rychlému nalezení a následnému sledování pohybujícího se objektu, jakým je například vozidlo.

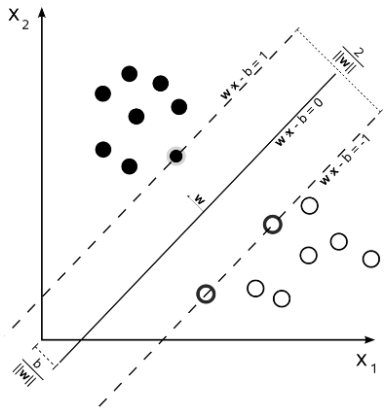
Na začátku se SVM předá množina trénovacích dat. Ty jsou reprezentovány jako dvojice $(x_{i,}y_i)$. Hodnoty x_i jsou vektory v multidimenzionálním prostoru. Dimenze tohoto prostoru jsou dány množstvím různých dat, které reprezentují jednotlivé objekty. Hodnota y_i je učitelem přiřazená hodnota 1 nebo -1, která reprezentuje, zda vektor do klasifikační skupiny patří či nikoliv. Cílem trénování SVM je poté nalezení hyperroviny, rozdělující prostor takovým způsobem, že jsou vektory z y_i = -1 a 1 jednoznačně odděleny.

Hyperroviny můžeme popsat následující rovnicí:

$$w * x + b = 0$$

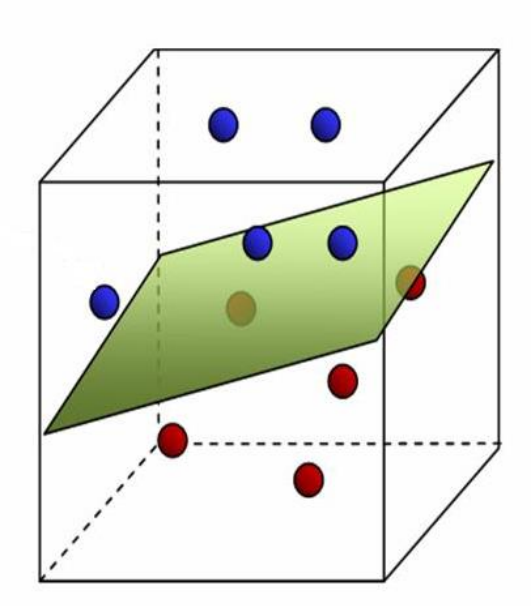
Rovnice 7 - Rovnice SVM hyper-roviny

Hodnota w je normálou hyperroviny, $||w||$ je její Euklidovská norma a $\frac{|b|}{||w||}$ je vzdálenost hyperroviny od středu (počátku) nejbližší souřadnice. Jinými slovy zvolená hyperrovina je taková, jejíž vzdálenost k nejbližšímu prvku jednotlivých tříd je co největší [10].



Obrázek 27 - Vzdálenosti 2D hyper-roviny

Jako příklad by šlo využít HOG deskriptoru, jehož výsledkem je vektor o 3780 položkách. Protože je ale pro člověka prostor o rozměru 3780 nepředstavitelný, ukázkou je o dost jednoduší 3D prostor.



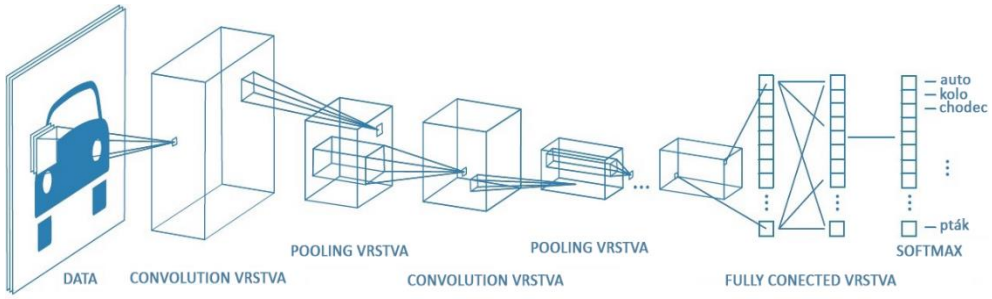
Obrázek 28 - 3D SVM Hyper-rovina

Je vidět, že rovin rozdělujících tato data, může být mnoho. Proto je vhodné mít dostatečné množství trénovacích dat, která by měla obsahovat všechny extrémní možnosti jednotivých tříd. Tak se docílí vytvoření co nejpřesnější hyperroviny.

2.3.3.3. CNN a Deep-Learning

Convolutional neural network je klasifikátor, který vezme zdrojová data a spojí je s určitou, již naučenou, kategorií [12].

CNN nechá data projít přes sérii vrstev jako postupně se měnící hodnoty. Tyto vrstvy se dají rozdělit do kategorií dle jejich funkcí. Kategoriemi jsou convolution vrstvy, pooling vrstvy a fully conected vrstvy, které se mohou několikrát opakovat. Jako poslední krok se využije Softmax funkce (normalizovaná exponenciální funkce). Tento postup poskytne jako výsledek pravděpodobnostní hodnoty shody s klasifikačními kategoriemi a to jako číslo v rozmezí 0 a 1, čímž se dá s určitou přesností identifikovat obsah obrazu.



Obrázek 29 - Vrstvy CNN

1. Convolution vrstva

   Tato vrstva extrahuje informace ze vstupních dat tak, aby zmenšila jejich velikost, ale aby ztráta informací byla co nejmenší. Docílí se toho matematickou operací, která využije dva vstupy – obrazovou matici a filtrovací matici neboli kernel.

   Jako příklad se využije obraz velikosti 5×5 a kernel velikosti 3×3:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Konvoluční matice 4 - Obrazová matice 5×5 a filtrovací matice 3×3

   Na následujícím obrázku je vidět vypočet nové matice (zcela vpravo), která vzniká postupným posouváním filtrační matice (uprostřed) po obraze a spočtením nových hodnot tak, že se vynásobí hodnota z filtrační matice s hodnotou odpovídajícího pixelu v původním obraze (vlevo). To znamená, že označené hodnoty na odpovídajících pozicích vynásobíme, aby vznikla jedna nová hodnota. Celkově tak vznikne nový, menší obraz.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 3 & 4 \\ 5 & 3 & 4 \\ 4 & 2 & 6 \end{bmatrix}$$

Konvoluční matice 5 - Obrazová matice 5×5, filtrovací 3×3 a výsledná matice

$$\begin{aligned} 4 = {} & (0 * 1) + (0 * 1) + (1 * 1) + \\ & (1 * 0) + (0 * 0) + (1 * 1) + \\ & (1 * 1) + (0 * 0) + (1 * 1) \end{aligned}$$

Rovnice 8 - Vypočtení hodnoty v Convolution vrstvě

Velikosti jednotlivých kroků posunu jsou určovány hodnotou stride, která určuje, o kolik pixelů se filtrovací matice posune.

Pokud nebyl vstupní obraz správně ošetřen na základě předzpracování (2.3.1.), může nastat problém s neperfektní velikostí obrazu. To může způsobit, že konvoluční vrstva není schopna vypočíst hodnotu, jelikož obraz neobsahuje data, která síť očekává. Tento problém se může řešit dvěma způsoby:

- doplnit obraz o hodnoty 0 (zero-padding),
- vynechat část obrazu, který je považován za nepotřebný.

2. Pooling vrstva

Tato vrstva může snížit množství parametrů u obrazů, které jsou moc velké. Při snižování počtu parametrů je velice důležité, co nejvíce předejít ztrátě informací. Těmto postupům, kdy proces vezme část obrazu (obvykle 2×2) a spočte novou menší matici, se říká subsampling nebo downsampling.

Existuje několik způsobů:

- Max Pooling – jako hodnotu nové matice využije největší hodnotu ze zdrojové části,
- Average Pooling – nová hodnota je spočtena jako průměr původních hodnot,
- Sum Pooling – jako nové hodnoty se využije součet původních hodnot, přičemž tento postup je nejvíce náchylný na ztrátu dat při extrémních rozdílech v nasvícení obrazu.

3. Fully conected vrstva

   Před tím, než můžeme matici předložit neurální síti, musíme ji změnit na jednoduchý vektor, tzn., že původní matice se změní na sérii hodnot (x1, x2, …). Tato vrstva poté využije nově vzniklý vektor pro porovnání modelu s naučenými vlastnostmi a pomocí funkce, jakou je například Softmax, vyhodnotí shodu s jednotlivými klasifikačními kategoriemi.

   Softmax funkce převede hodnoty, které získá na vstupu a přepočte je na pravděpodobnostní hodnoty, jejichž součet je roven 1.

vstupní hodnoty *Softmax* *pravděpodobnostní hodnoty*

$$y\begin{bmatrix}2.0 \rightarrow \\ 1.0 \rightarrow \\ 0.1 \rightarrow\end{bmatrix} \boxed{S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}} \begin{matrix}\rightarrow p = 0.7 \\ \rightarrow p = 0.2 \\ \rightarrow p = 0.1\end{matrix}$$

Rovnice 9 - Softmax

Problémem konvolučních neuronových sítí může být to, že Fully connected vrstva není konstantní, tzn., že má proměnlivou velikost. To je z důvodu, že počet nalezených objektů v obraze není nikdy stejný. Nejjednodušším přístupem by bylo ručně rozdělit obraz na regiony a pokusit se klasifikovat nalezený objekt v konkrétním regionu. Způsoby, jakým určit rozdělení na regiony, si detailněji probereme u jejich jednotlivých implementací v kapitole 3.

3. Srovnání algoritmů

Jak již bylo výše zmíněno, konvoluční sítě mají problém s vyhledáváním proměnlivého počtu objektů. Z tohoto důvodu vznikly komplexnější algoritmy na vyhledávání objektů a jejich následnou identifikaci.

3.1. Fast R-CNN

Jeden z přístupů, jak se vyhnout obrovskému množství prohledávaných oblastí, navrhl Ross Girshick [14]. Jeho návrh R-CNN spočíval v prohledání 2000 regionů, které pojmenoval Proposal regions, což je již poměrně malý počet k automatickému prohledání. Těchto 2000 regionů je vytvářeno pomocí algoritmu na selektivní vyhledávání. Algoritmus se může skládat z blokově orientovaných histogramů, jakým je i výše zmiňovaný HOG (2.3.2.4.), a následným zjištěním množství informací v nich obsažených [15]. Obsah těchto oblastí poté nechá vyhodnotit konvoluční sítí.

Prohledávání 2000 regionů je ale stále velice zdlouhavé. Navíc toto selektivní vyhledávání je fixní algoritmus, který vždy vyhledává stejné regiony, tudíž v procesu prohledávání nedochází ke zlepšení neboli k učení.

Problém s učením vyřešil autor původního návrhu Ross Girshick novou verzí detekčního algoritmu pojmenovaném Fast R-CNN [14]. V této vylepšené verzi se prohledávané oblasti nevyhledávají přímo z obrazu, ale místo toho z obrazu vytvoříme konvoluční mapu vlastností. Tato mapa se skládá ze sady kernelů, které propustí pouze oblasti splňující podmínky, například limitní množství informací v regionu. Přístup přes tuto vrstvu umožňuje, aby se pravidla pro vybrané oblasti s postupem času „učila“. Tyto vybrané oblasti poté prohledáme pomocí selektivního vyhledávání a dále se již pokračuje jako u standartní konvoluční sítě (2.3.3.3.). U přístupu přes mapu vlastností dosahujeme větší rychlosti, jelikož není potřeba, aby konvoluční síť testovala všech navrhnutých 2000 regionů.

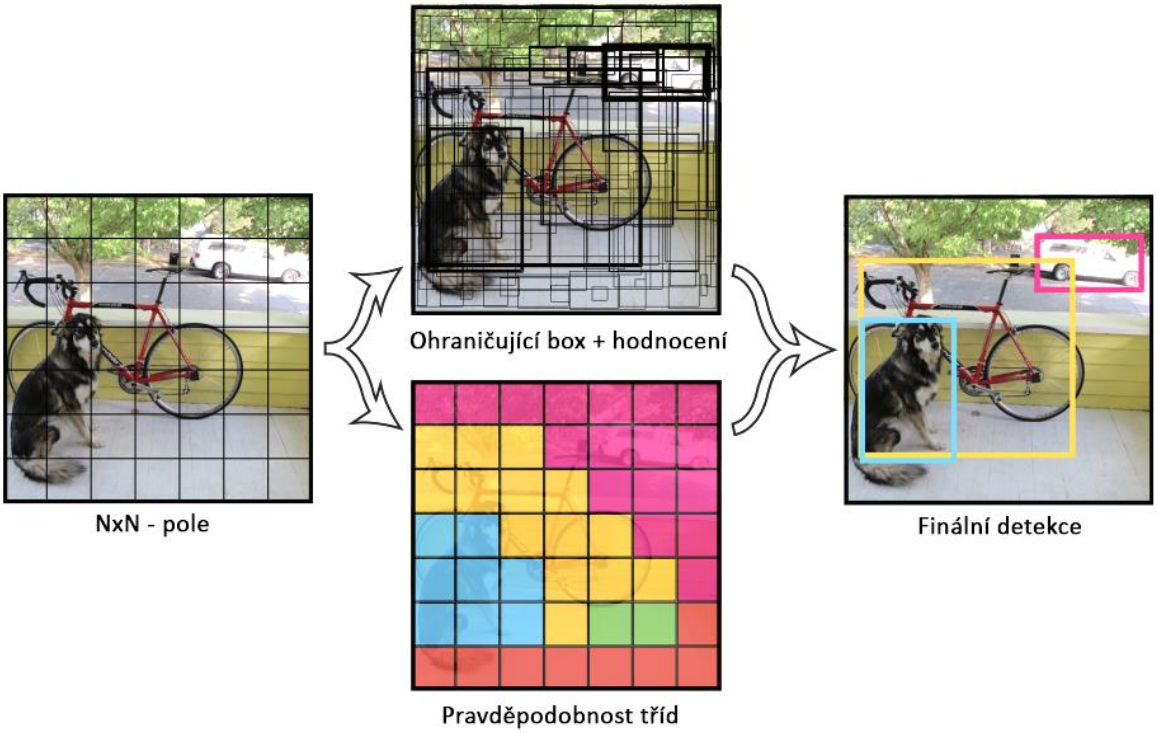
Jak metoda R-CNN, tak její rozšíření Fast R-CNN, jsou stále zpomalovány algoritmem na selektivní vyhledávání regionů. S řešením a novou verzí nazvanou Faster R-CNN přišel Shaoqing Ren [16]. Tato detekční metoda nechá neuronovou síť, aby se učila vyhledávat regiony. Shodně jako u Fast R-CNN předáme obraz konvoluční síti, aby vytvořila mapu vlastností. Místo abychom poté prohledaly nalezené vlastnosti pomocí selektivního vyhledávání, zde využijeme druhou neuronovou síť k predikování regionů.

Touto změnou dosáhneme několikanásobného zvýšení rychlosti oproti standardnímu selektivnímu prohledávání.

3.2. YOLO – You Only Look Once

Přístup algoritmu YOLO je zcela odlišný od všech variant výše zmíněného R-CNN, které využívají lokalizaci regionů v obraze a proto se u nich neuronová síť nikdy nepodívá na celý obraz.

YOLO využívá jedné konvoluční neuronové sítě k predikci ohraničujících boxů kolem objektů a současně hodnot definujících pravděpodobnost, že box obsahuje jednu z klasifikačních tříd, které neuronová síť hledá.



Obrázek 30 - YOLO

YOLO tedy vezme obraz a rozdělí jej na N×N polí. Každý nalezený objekt je přiřazen odpovídajícímu poli, které je za něho „zodpovědné“. Jako zodpovědné pole je zvoleno to, které obsahuje přibližný střed objektu. Každému poli poté vytvoříme M ohraničujících boxů, které obsahují odhad rozdělení, jaké objekty může pole obsahovat. Boxy mají určené souřadnice (x, y). Každý ohraničující box doplní neuronová síť o výstup obsahující pravděpodobnost klasifikačních tříd a velikost. Boxy, které mají nějakou pravděpodobnost nad limitní hodnotou, jsou vybrány a použity k lokalizaci objektu v poli.

Pravidlem je, že se v poli detekuje právě jeden objekt. Jako detekovaný objekt je zvolena klasifikační třída, která byla nejčastěji nalezena v ohraničujících boxech příslušného pole. Algoritmus poté penalizuje odhad boxů, které neodpovídají očekávanému objektu. Hodnota klasifikace je ovlivňována velikostí boxu. Výsledkem je jedna sada pravděpodobnostních hodnot pro každé pole a to nezávisle na M množství ohraničujících boxů. Pro další postup si každé pole vybere box, který ho nejvíce vystihuje, tzn. má největší pravděpodobnost obsahovat objekt, o kterém i ostatní boxy tvrdí, že by mohl být nalezeným objektem. Takto zvolené boxy se předají neuronové síti.

Zbylá struktura sítě je stejná jako standartní CNN (2.3.3.3.), obsahuje tedy pooling vrstvu, následovanou dvěma fully conected vrstvami.

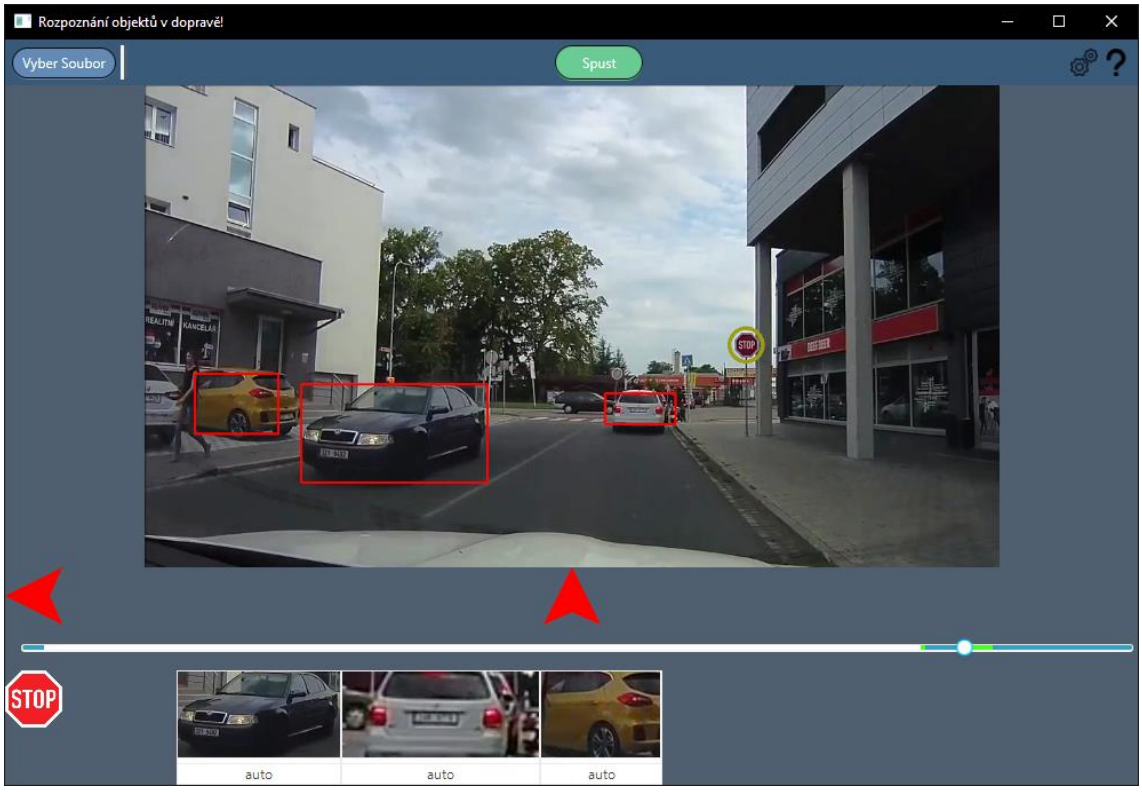
Výhodou YOLO je jeho neuvěřitelná rychlost, naopak nevýhodou je problém s obtížným vyhledáváním malých objektů či se splynutím většího množství překrývajících se objektů v jeden.

4. Popis praktické části

Praktická část práce se zabývá detekcí dopravního značení a složitých objektů nacházejících se v dopravě. Těmito objekty jsou například vozidla, chodci či cyklisté.

Program je napsán v jazyce Java. K jeho spuštění je potřeba JRE (Java Runtime Environment) ve verzi 1.8. a vyšší a JavaFX. Aplikace je určena pro otestování a grafickou ukázku metod, které mohou být využity při zpracovávání videa z dopravního prostředí. I když je aplikace schopna pracovat s jakýmkoliv videoobsahem, je silně doporučeno použít pouze videa obsahující data dopravních situací z kamer z vozidla.

Nalezené objekty jsou v obraze vyznačeny. Čtverec označuje dopravní objekty a kruh ohraničuje nalezené dopravní značky. Dalé jsou tyto nálezy vyznačeny v dolní liště. Při nalezení objektů se také zobrazuje šipka určující směr nalezeného objektu.



Obrázek 31 - GUI programu

Identifikace je rozdělena na dva postupy. Je to z důvodu, že dopravní značení je dáno zákonem a díky tomu máme jistotu tvaru a barev. Je také vždy natočeno směrem, ke kterému je platné, proto není potřeba použít postup, který je odolný na tyto problémy.

4.1. Identifikace dopravních objektů

Identifikace dopravních objektů je založena na procházení obrazu založeném na principech YOLO a na neuronové sítí, jejíž váhy předurčují vyhledávané objekty a kvalitu vyhledávání. Výkon postupu je proto silně závislý na datových modelech, které jsou využity k vytrénování vah sítě. Data, která byla v programu využita, byla založena na COCO datasetu, který poskytuje data pro identifikaci většiny objektů nalézaných v dopravě. Na základě tohoto datasetu existuje množství předtrénovaných modelů, jako příklad využiji dva z nich a data, která o nich poskytují oficiální stránky projektu YOLO.

Název modelu	YOLOv2 608x608	Tiny YOLO	Můj model
mAP - (mean Average Precision)	48.1	23.7	28.5
FPS (na Pascal Titan X)	40	244	~60-70 (odhad)
FPS (pouze CPU – i7-9700k)	6	32	10

Dle oficiálních testů poskytují tyto modely rychlost 40, respektive 244 detekovaných snímků o velikosti 416×416 pixelů za vteřinu, osobní test jsem provedl pouze s CPU, kde výsledná rychlost byla 6 a 32 FPS. Můj model jsem mohl testovat pouze na CPU, kde dosáhl průměru 10 FPS.

V předchozí tabulce se také setkáváme s pojmem mAP, to je průměrná hodnota přesnosti nalezení objektů. K této hodnotě se dostaneme spočtením průměru z hodnot, které vznikají při testování tohoto modelu.

TP = True positive (všechny správě positivně identifikované objekty jedné třídy)

TN = True negative (všechny správě negativně identifikované objekty jedné třídy)

FP = False positive (všechny špatně positivně identifikované objekty jedné třídy)

FN = False negative (všechny špatně negativně identifikované objekty jedné třídy)

Precision = $\frac{TP}{TP+FP}$ (určuje důvěryhodnost positivně identifikovaných objektů)

Recall = $\frac{TP}{TP+FN}$ (určuje množství nenalezených objektů)

AP = průměrný Presicion

mAP = průměrná hodnota AP přes všechny třídy (čím vyšší tím lepší)

Na základě tabulky můžeme vidět, že model Tiny YOLO je více než 6× rychlejší, ale dosahuje méně než poloviční kvality identifikace objektů. Výsledkem mého testování je zjištění, že modely mají větší problém s přesností pozice nalezeného objektu, a ne už tolik

s jeho samotným nalezením. Jelikož mnou vytvořený model neposkytoval dostatečně lepší detekci v poměru za skoro ¼ rychlost, rozhodl jsem se využít oficiální model Tiny YOLO.

4.2. Identifikace dopravního značení

Druhým postupem je identifikace dopravních značek nalezených v obraze. Tohoto postupu využívají například vozidla s asistentem, kde si mohou řidiči zobrazit na obrazovce příslušné informace nebo autonomní vozidla. U nich je pak tato schopnost identifikace značek nedílnou součástí jejich schopnosti rozhodovat se, jak řídit.

I když je tato úloha velmi specifická, stále se skládá se stejných postupů, jakou je kterákoliv jiná identifikace:

1. Zlepšení obrazu za pomoci předzpracování
   Zvolen byl postup odstranění šumu z obrazu za pomoci Gaussova filtru.

2. Získání potřebných informací
   V případě značek je nejvíce vhodný Haar-like classifier, jelikož jednotlivé značky se dají snadno rozlišit jednoduchými vlastnostmi, které dovedou Haar-like vlastnosti dobře popsat.

3. Klasifikace
   Samotná klasifikace je již jen jednoduché srovnání získaných informací se souborem obsahujícím předcvičený popis určité značky.

Obrázek 32 - Identifikace značek

Ke klasifikaci značek byl využit kaskádový klasifikátor využívající Haar-like features (2.3.2.3.), který pracuje na principu postupného srovnávání jednotlivých vlastností

s vlastnostmi nalezenými v obraze. Díky postupnému kontrolování, většinou od jednodušších vlastností po složitější, dochází k ušetření velkého množství opakovaných kontrol dat. To díky tomu, že další úrovně kaskádového srovnávání probíhá již jen na oblastech obrazu, které prošly kontrolou předchozích kroků.

Modely Haar-like features definující určitý vyhledávaný objekt se převážně uchovávají ve formátu `xml`. Tento soubor se skládá z jednotlivých trénovacích vrstev (stage), které postupně vylepšují složitost vyhledávaného objektu.

Na internetu jsem nalezl pár volně dostupných Haar-like klasifikačních modelů. Velká většina z nich však nevyhovovala kvalitou či typem vyhledávaných značek. Proto jsem došel k závěru vytvořit vlastní klasifikátory.

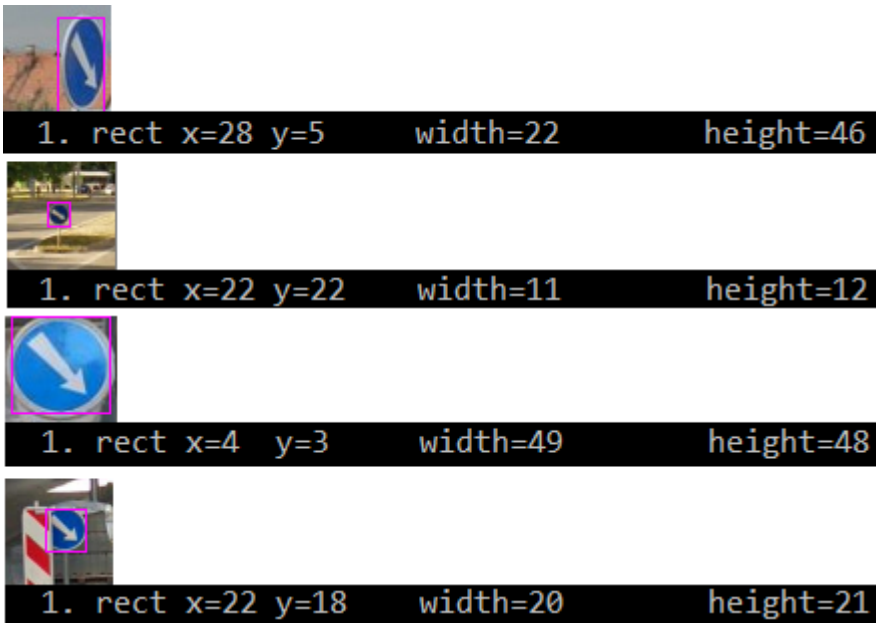
Prvním krokem učení klasifikátoru je nalezení vhodných dat pro trénování. Nalezl jsem několik volně dostupných datasetů pro identifikaci značek, z nichž jako nejlepší mi připadal German Traffic Sign Benchmark. Tento soubor dat vypadal velmi kvalitně. Skládá se ze skoro 40 000 obrazů rozdělených do 43 klasifikačních kategorií, přičemž kategorie se skládají převážně z obrazů různých verzí značky omezení rychlosti a značek převážně specifických pro dálnice. Jeho obsahu jsem proto využil na vytvoření pouze jediného modelu a to sice značky „Stůj, dej přednost v jízdě!“. Pro ostatní modely jsem využil vlastních dat vytvořených ze záběrů Street View z Google Maps. Celkovým výsledkem práce jsou soubory formátu `xml` definující vyhledávání následujících 6 značek:

Obrázek 33 - Vyhledávané značky
(„Stůj, dej přednost v jízdě!“, „Kruhový objezd“, „Dej přednost v jízdě!“, „Přechod pro chodce“, „Přikázaný směr objíždění vpravo“, „Zákaz zastavení“)

Data pro klasifikátor se skládají ze dvou částí

1. Postivies

   Pozitivní data, která by měla obsahovat vyhledávaný objekt s drobným okolím. U těchto dat je nutné, aby všechny soubory měly stejný aspekt-ratio, u mého projektu byl využit 1:1 čtverec. Pozitivní data nemusí být přesně oříznutý objekt. Naopak bych doporučil ponechat pozadí okolo objektu. To dopomůže naučit systém, jak od objektu oddělit pozadí.



Obrázek 34 - Příklad positives dat

Pokud obraz ale obsahuje i něco jiného než objekt samotný, je nutno určit oblast, kde přesně se objekt nachází. K tomu slouží popisný soubor, jehož struktura je následující:

```
1  positive/1.jpg 1 2 9 49 38
2  positive/10.jpg 1 7 14 42 39
3  positive/11.jpg 1 5 2 46 51
4  positive/12.jpg 1 6 5 40 47
5  positive/13.jpg 1 5 1 46 47
6  positive/14.jpg 1 5 6 45 44
7  positive/15.jpg 1 4 0 45 51
8  positive/16.jpg 1 5 7 36 35
9  positive/17.jpg 1 11 10 36 31
```

Obrázek 35 - Struktura positivních dat

a každý řádek definuje jeden obraz:

- adresa souboru (positive/1.jpg)
- počet nalezených objektů (1), ale lze i více
- souřadnice ohraničující objekt – x, y levého horního rohu; šířka; výška (2 9 49 38)

Poté se z obrazu a souboru, který ho popisuje, vytvoří datový vektor. Ten má pevně danou velikost obrazu, na kterou se všechny positivní obrazce přetransformují. Z tohoto důvodu je nutné dodržet aspekt-ratio.

2. Negatives

   Negativní data obsahují obrazy objektů, které mohou tvarem připomínat vyhledávaný objekt a obrazy obsahující objekty, které se mohou nacházet jako pozadí u pozitiv (stromy, budovy, silnice, vozidla, …). Negativa nesmí nikdy obsahovat ani malou část objektu, který by měl být pozitivně identifikován.

   Obrazy mohou být libovolně velké. Pokud se však z velké části opakují či obsahují pro cíl trénování zbytečné objekty, způsobí zpomalení tréninku.

Pozitivních dat jsem měl pro každou značku mezi 200–300. Negativní obrazy jsem využil shodné pro všechny značky a bylo jich 5266.

Trénink byl prováděn za pomoci `opencv_traincascade` programu [13]. Pro rychlost tréninku je velmi důležité množství dostupné operační paměti a velikost, na kterou byly positivní obrazy transformovány.

Porovnání doby tréninku mezi velikostí pozitiva 28×28 a 56×56:

```
PARAMETERS:
numPos: 327
numNeg: 5266
sampleWidth: 28
sampleHeight: 28
Number of unique features given windowSize [28,28] : 486772

===== TRAINING 5-stage =====
<BEGIN
POS count : consumed   327 : 327
NEG count : acceptanceRatio    5266 : 9.21771e-05
Precalculation time: 133.684
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|  0.41596|
+----+---------+---------+
END>
Training until now has taken 0 days 2 hours 40 minutes 33 seconds.
```

```
PARAMETERS:
numPos: 327
numNeg: 5266
sampleWidth: 56
sampleHeight: 56
Number of unique features given windowSize [56,56] : 7888521

===== TRAINING 5-stage =====
<BEGIN
POS count : consumed   327 : 327
NEG count : acceptanceRatio    5266 : 4.31746e-05
Precalculation time: 195.756
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4|        1| 0.404482|
+----+---------+---------+
END>
Training until now has taken 1 days 23 hours 15 minutes 11 seconds.
```

Obrázek 36 - Porovnání doby 5-stage mezi 28×28 a 56×56

Průměrný rozdíl všech časů učení byl zhruba 1:16, což odpovídá i rozdílu unikátních vlastností, které jsou 486 772:7 888 521, tedy v poměru 1:16,2.

Trénink probíhá v kolech, tzv. stage, kdy každý následující stage trvá déle než předchozí:

```
PARAMETERS:
numPos: 327
numNeg: 5266
sampleWidth: 28
sampleHeight: 28
Number of unique features given windowSize [28,28] : 486772
===== TRAINING 0-stage =====
<BEGIN
POS count : consumed   327: 327
NEG count : acceptanceRatio    5266: 1
Precalculation time: 130.101
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4|        1| 0.188705|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 4 minutes 42 seconds.

.
.
.

===== TRAINING 4-stage =====
<BEGIN
POS count : consumed   327:327
NEG count : acceptanceRatio    5266 : 0.000222959
Precalculation time: 129.942
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4|        1| 0.361604|
+----+---------+---------+
END>
Training until now has taken 0 days 1 hours 9 minutes 59 seconds.
```

```
===== TRAINING 5-stage =====
<BEGIN
POS count : consumed   327 : 327
NEG count : acceptanceRatio    5266 : 9.21771e-05
Precalculation time: 133.684
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|  0.41596|
+----+---------+---------+
END>
Training until now has taken 0 days 2 hours 40 minutes 33 seconds.

.
.
.

===== TRAINING 13-stage =====
<BEGIN
POS count : consumed   327 : 327
NEG count : acceptanceRatio    5266 : 9.2443e-07
Precalculation time: 117.544
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4|        1|        1|
+----+---------+---------+
|   5|        1|        1|
+----+---------+---------+
|   6|        1|  0.65359|
+----+---------+---------+
|   7|        1| 0.406208|
+----+---------+---------+
END>
Training until now has taken 3 days 18 hours 22 minutes 11 seconds.
```

Obrázek 37 - Porovnání doby mezi jednotlivými stage

Zde je vidět, že první kolo trvá necelých 5 minut. Všechna kola až do 4-stage dohromady něco málo přes hodinu. Ale výpočet samotné 5-stage již trvá skoro 2 hodiny. Celkový čas tréninku tohoto modelu byl 3 dny a 18 hodin, a i když cílený stage byl zadán 20-stage, trénovací program vyhodnotil, že akceptační poměr ($9.2443e^{-7}$) je již dostatečné a po 13-stage odmítl trénovat dál. Stage, ve kterém trénink skončí, je dán složitostí vyhledávaného objektu a kvalitou trénovacích dat.

	čas stage	čas celkem	akceptační poměr
0-stage	4 m 42 s	4 m 42 s	1
1-stage	6 m 53 s	11 m 35 s	0.150884
2-stage	16 m 22 s	27 m 57 s	0.0276642
3-stage	12 m 36 s	40 m 33 s	0.00320006
4-stage	29 m 26 s	1 h 9 m 59 s	0.000868653
5-stage	1 h 30 m 34 s	2 h 40 m 33 s	0.000222959
6-stage	2 h 14 m 38 s	3 h 55m 11s	9.21771e-05
7-stage	3 h 39 m 11 s	8 h 34 m 22 s	3.4454e-05
8-stage	4 h 55 m 48 s	13 h 30 m 10 s	1.20852e-05
9-stage	5 h 2 m 30 s	18 h 32m 40s	9.8443e-06
10-stage	9 h 45 m 4 s	1 d 4h 17 m 44 s	7.4454e-06
11-stage	15 h 26 m 2 s	1 d 19 h 43m 46	3.6454e-06
12-stage	20 h 31 m 32 s	2 d 16 h 15 m 18	1.3443e-06
13-stage	1 d 2 h 6 m 53 s	3 d 18 h 22 m 11	9.2443e-07

Hodnota akceptačního poměru je vytvářena z negativních obrazů a ukazuje hodnotu, kolik bylo potřeba udělat průměrně vyhledání na obrazu, než byla nalezena chyba. Například u 3-stage hodnota ~0.003 znamená, že na každém ze zadaných 5266 obrazů bylo vyhledáno zhruba 333 pokusů, než byl obraz akceptován. U posledního stage to znamená, že bylo celkem 5.69 bilionu různých pokusů.

$$5.69 \times 10^{9} = \frac{5266}{9.2443e^{-7}}$$

Rovnice 10 - Počet vyhledávání v 13-stage

Všechny výše uvedené testy byly provedeny na počítači s procesorem 4core 3.7 GHz a 24 GB RAM.

Ve dvou rozdílných velikostech pozitiva jsem vytvořil pouze první model, obě verze byly dotrénovány do 9-stage, kde skončily. Jejich výsledné chování jsem porovnal. Hlavním rozdílem byla nejmenší velikost, ve které je objekt nalezen, tzn., že větší model nalezl objekt o několik snímků videa dříve. Na základě těchto výsledků jsem ostatní modely trénoval v rozlišení 28×28.

4.3. Otestování praktické části

Pro testování navrženého programu jsem využil video z palubní kamery vozidla. Video je v rozlišení 1126×634 o rychlosti 25 snímků za sekundu, dlouhé 76 sekund a celkem obsahuje 1912 snímků. Zobrazuje jak potřebné dopravní značky, tak pohyb vozidel v dopravě.

Testoval jsem vliv nastavení paramertů na přesnost a rychlost vyhledávání.

Obrázek 38 - Nastavení programu

- Rychlost přehrávání je v programu pouze pro uživatelskou přívětivost, na testy nemá žádný vliv.
- Doba zobrazování nalezených objektů určuje, jak dlouho se má vyhledaný objekt po nelezení zobrazovat. Neovlivňuje tedy skutečné nálezy, jen prodlouží dobu zobrazení nalezeného objektu a tím zamezí občasnému blikání nálezů při krátkodobých ztrátách objektu v obraze.
- Hodnota parametrů „Vyhledávat dopravu/značky každý X snímek“ určuje frekvenci vyhledávání a tím ovlivňuje, jak dlouho bude trvat prohledávání celého videa. Například hodnota 3 znamená, že se po nalezení objektu v daném snímku ignorují 2 následující a uměle se do nich nález zapíše. Tím se docílí velkého zrychlení vyhledávání za cenu menší přesnosti.
- Možnost deaktivovat vyhledávání složitých objektů a nechat program vyhledávat pouze dopravní značení velmi silně ovlivňuje rychlost programu
- Velikost bufferu určuje, kolik snímků se předem načítá a uchovává v paměti.

Testy byly prováděny na počítači s procesorem Intel Core i5-6600K o 4 jádrech a frekvenci 3.5GHz. Výsledky jsou rozděleny do dvou částí, a to se zapnutým a vypnutým vyhledáváním složitých objektů. Tabulky obsahují celkový čas vyhledávání a hodnotu kolik bylo nalezeno snímků obsahujících danou značku. Hodnoty jsem získal z počtu objektů jednotlivých tříd, které program nalezl při dané frekvenci. Falešné nálezy jsem doplnil ručním nalezením. Všechny testy jsem opakoval 10× a výsledky jsou zprůměrovány.

Výsledky při vypnutém vyhledávání složitých objektů:

Frekvence vyhledávání	Celkový čas vyhledávání	Počet nalezených (počet falešných nálezů)						Falešné nálezy celkem
								
1	243 s	12 (0)	7 (2)	18 (3)	13 (0)	50 (2)	530 (15)	22
2	117 s	6 (0)	4 (2)	9 (3)	5 (0)	26 (1)	268 (13)	19
3	78 s	4 (0)	5 (2)	4 (2)	3 (0)	16 (1)	176 (11)	16
4	58 s	3 (0)	3 (1)	3 (2)	2 (0)	12 (1)	140 (11)	15
5	44 s	2 (0)	0,4 (0)	4 (2)	2 (0)	9 (1,5)	108 (11)	14

Z výsledku je podle očekávání zřejmý vliv frekvence na počty nálezů. Hodnoty pro dvojnásobnou frekvenci jsou poloviční, pro frekvenci 3krát vyšší je nálezů 3krát méně. Oproti tomu je vidět, že dochází k velmi shodným počtům falešných nálezů. Domnívám se, že nálezy jsou stejné a poukazují na drobné chyby v některých vyhledávácích modelech, především v modelu „Přikázaný směr objíždění vpravo“.

Obrázek 39 - Falešný nález „Přikázaný směr objíždění vpravo“

Dále jsem zjistil, že značka, která se na videu objevuje jen krátce, nemusí být při vysoké frekvenci vůbec nalezena. Toto je vidět u značky „Přechod pro chodce“, která neměla v 6 z 10 testů o frekvenci 5 žádný zaznamenaný nález.

Výsledky vyhledávání složitých objektů nejsou nijak ovlivněny vyhledáváním dopravních značek, proto jsem se rozhodl tyto testy dělat s nastaveným parametrem „Vyhledávat značky každý 4 snímek“. Jelikož testy ukázaly pouze na jeden opakující se falešný nález, počet falešných nálezů jsem v následující tabulce vynechal. Tímto jediným falešným nálezem byla identifikace série popelnic jako vozidlo.

Obrázek 40 - Falešný nález vozidla

Výsledky při zapnutém vyhledávání složitých objektů:

Frekvence vyhledávání	Celkový čas vyhledávání	Počet nalezených (počet nálezů vynásobených frekvencí)		
		vozidla	chodci	motorky
1	1286 s (21 m 8 s)	2556 (2556)	28 (0)	6 (6)
2	613 s (10 m 13 s)	1273 (2546)	14 (28)	3 (6)
3	430 s (7 m 10 s)	850 (2550)	9 (9)	2 (6)
4	320 s (5 m 20 s)	628 (2512)	7 (28)	2 (8)
5	268 s (4 m 28 s)	504 (2520)	5 (25)	2 (10)
6	218 s (3 m 38 s)	419 (2514)	4 (24)	2 (12)
7	195 s (3 m 15 s)	358 (2506)	3 (21)	1,2 (8,4)
8	175 s (2 m 55 s)	313 (2504)	3 (24)	1 (8)
9	158 s (2 m x 38 s)	285 (2565)	3 (27)	0 (0)

Nálezy při vyšších frekvencích vypadaly podle očekávání skokově. Snížila se sice přesnost nalezené pozice objektů, ale objekty, které se vyskytovaly na více snímcích, než byla hodnota frekvence, byly vždy nalezeny. Z tohoto důvodu se s vyšší frekvencí snížil počet nalezených chodců a jednostopých vozidel (motorek a kol). Stejně jako u vyhledávání značek nemusí ani zde dojít k nalezení objektů, které jsou na méně snímcích, než je frekvence. Jako výchozí hodnotu jsem proto zvolil nastavení „Vyhledávat dopravu každý 6 snímek“. Tato frekvence byla nejvyšší, při které byly nalezeny všechny složité objekty.

Poslední testovanou součástí byl buffer:

Počet snímků v bufferu	Velikost operační paměti
samotný program	520 MB
100 snímků	1200 MB
500 snímků (výchozí hodnota)	2700 MB
1000 snímků	4700 MB
1912 snímků (celé video)	8560 MB

Po testech vlivu velikosti bufferu na obsazení operační paměti programem, jsem do okna Nastavení přidal textovou poznámku „zabere ~X MB“, kde hodnota X zobrazuje horní odhad MB spočtený na základě zvolené velikosti bufferu.

5. Závěr a doporučení

Na detekce vozidel a jiných složitých objektů je dnes již poměrně propracovaný postup využívající logického a neopakujícího se procházení obrazu. Při vývoji programu jsem se pokoušel o alternaci principů YOLO a vytvoření vlastního postupu procházení obrazu. Mé pokusy skončily vždy buď o poznání pomalejší, nebo jejich nálezy byly velmi špatné. Z toho důvodu jsem vytvořil postup silně založený na YOLO postupu a pokusil se rychlost vyhledávání zlepšit na úrovni modelu neuronové sítě.

Jak již bylo řečeno, rychlost těchto postupů je silně ovlivňována modely, které jsou využity pro trénování neuronové sítě využité k nacházení objektů. Proto je mým základním a hlavním doporučením vyhledávání pouze objektů, které jsou skutečně důležité a odstranění všech nepotřebných částí modelů. Jako příklad lze říci, že plná verze COCO datasetu obsahuje i identifikační třídy, jako jsou tučňák či různé druhy ovoce. Proto byl pro vyhledávání vybrán model, který je sice založen na COCO datasetu, ale obsahuje pouze objekty nacházející se v dopravě.

Hlavní činností, kterou jsem strávil nejvíce času, bylo testování různých verzí Haar-like features modelů na detekování značek. Haar-like features je nejčastěji využíváno pro vyhledávání obličejů, ale po testech jsem dospěl k závěru, že pro dopravní značení jsou velmi vhodné. Na základě testů bych dále doporučil způsob, jakým volit data pro trénování Haar klasifikátoru. Základem je přizpůsobení dat potřebám, které na klasifikátor máme a prostředkům, které jsou k dispozici k jeho natrénování. Těmi je hlavně čas a výpočetní výkon. Velké množství pozitivních dat se těžko shání a poté nastává další problém, kdy některá z nich lze považovat za zbytečná, jelikož dataset začne obsahovat obrazy, které si jsou velmi silně podobné. Proto doporučuji menší množství dat, kde je nejdůležitější zahrnout především extrémní verze hledaného objektu. Za extrémy lze považovat velmi malý a vzdálený objekt, špatné světelné podmínky nebo částečné zakrytí hledaného objektu. Negativní data mají výhodu, že mohou a i by měla obsahovat vše, co není hledaný objekt. Proto se negativní data dají sdílet mezi více modely. Pro modely projektu jsem využil v průměru 250 pozitiv na 5266 negativ.

Poslední radou je velikost těchto dat. Je zřejmé, že čím větší data, tím přesnější model. Po překročení určité hranice však již rozdíly nebudou znatelné. Velikost modelu doporučuji odhadnout na základě dat, které vyhledáváte, aneb vezměte nejdetailnější verzi objektu, jakou můžete nalézt a tím si určíte extrémní rozlišení hledaného objektu. U mne

touto velikostí bylo 56×56. Poté můžete i od oka testovat, jaké množství detailů se ztratí zmenšením rozlišení obrázku. Vyzkoušel jsem, že není moc znatelný rozdíl do rozlišení 28×28, což je přesně polovina původního. Poté jen stačí zvolit vybranou velikost a nechat systém učit. Pokud je však potřeba model s extrémní přesností nebo je dostatečný čas a hardwarový výkon, doporučuji využít obrazů s co největší velikostí.

Pokud se tady u programu vhodně přizpůsobí nastavení na využívaný hardware, je program schopen vyhledávat v reálném čase i když za cenu nízké přesnosti na slabších strojích. Díky tomu by mohl být využit jako asistenční software ve vozidlech či autokamerách. Program je připraven na snadné rozšíření Haar-like klasifikátoru o další značky. Při rozšíření programu o velké množství značek by bylo vhodné vytvořit strukturované vyhledávání, tj. nejprve vyhledají obecné tvary značek (kruh, trojúhelník) a až tyto tvary testovat na identifikaci.

6. Seznam použité literatury

[1] H. BALLARD, Dana a Christopher M. BROWN. Computer Vision. 1. New Jersey: PrenticeHall, 1982. ISBN 0131653164 AACR.

[2] SZELISKI, Richard. Computer Vision: Algorithms and Applications. London: Springer, 2010. Texts in computer science. ISBN 978-1-84882-935-0.

[3] SONKA, Milan, Vaclav HLAVAC a Roger BOYLE. Image processing, analysis, and machine vision. 2nd ed. Pacific Grove, CA: PWS Pub., c1999. ISBN 978-0-534-95393-5.

[4] FISHER, R, S PERKINS, A WALKER a E WOLFART. Gaussian smoothing [online]. 2003 [cit. 2019-03-14]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[5] CANNY, John. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence [online]. 1986, PAMI-8(6), 679-698 [cit. 2019-03-14]. DOI: 10.1109/TPAMI.1986.4767851. ISSN 0162-8828. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4767851

[6] CHEUNG, Sen-ching S., Sethuraman PANCHANATHAN, Bhaskaran VASUDEV a Chandrika KAMATH. Robust techniques for background subtraction in urban traffic video [online]. In: . 2004-1-7, s. 881- [cit. 2019-03-14]. DOI: 10.1117/12.526886. Dostupné z: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.526886,

[7] Convolutional Neural Networks for Visual Recognition [online]. [cit. 2019-03-17]. Dostupné z: http://cs231n.github.io/convolutional-networks/#overview

[8] VIOLA, P a M JONES. Rapid Object Detection Using a Boosted Cascade of Simple Features [online]. 2004 [cit. 2018-10-29]. Dostupné z: http://www.merl.com/publications/docs/TR2004-043.pdf

[9] DALAL, N. a B. TRIGGS. Histograms of Oriented Gradients for Human Detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) [online]. IEEE, 2005, s. 886-893 [cit. 2018-11-28]. DOI: 10.1109/CVPR.2005.177. ISBN 0-7695-2372-2. Dostupné z: http://ieeexplore.ieee.org/document/1467360/

[10] ŽIŽKA, J. Support vector machines (SVM) [online]. 2005 [cit. 2018-11-17]. Dostupné z: https://is.muni.cz/el/1433/podzim2005/PA034/09_SVM.pdf

[11] CORTES, Corinna a Vladimir VAPNIK. Support-vector networks. Machine Learning [online]. 1995, 20(3), 273-297 [cit. 2018-11-28]. DOI: 10.1007/BF00994018. ISSN 0885-6125. Dostupné z: http://link.springer.com/10.1007/BF00994018

[12] Nicu Sebe; I. Cohen; Ashutosh Garg; Thomas S. Huang (2005). Machine Learning in Computer Vision. Springer Science & Business Media. ISBN 978-1-4020-3274-5

[13] OpenCV. OpenCV [online]. [cit. 2018-01-26]. Dostupné z: https://opencv.org/

[14] GIRSHICK, Ross. Rich feature hierarchies for accurate object detection and semantic segmentation [online]. [cit. 2019-02-04]. Dostupné z: https://arxiv.org/pdf/1311.2524.pdf

[15] UIJLINGS, J.R.R. a K.E.A. VAN DE SANDE. Selective Search for Object Recognition [online]. [cit. 2019-02-04]. Dostupné z: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf

[16] REN, Shaoqing a Ross GIRSHICK. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [online]. [cit. 2019-02-04]. Dostupné z: https://arxiv.org/pdf/1506.01497.pdf

7. Přílohy

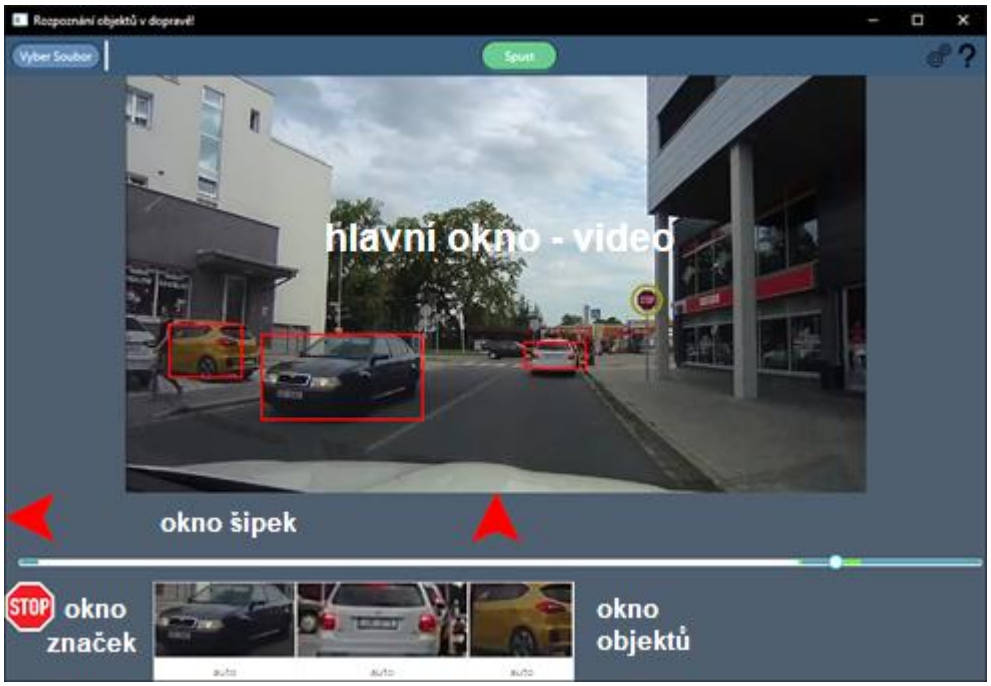
I. Seznam obrázků

II. Seznam rovnic

III. Seznam konvolučních matic

IV. Uživatelská dokumentace

Program na rozpoznání objektů v dopravě.



Obrázek 41 - GUI programu

Grafické rozhraní obsahuje:

1. `Tlačítko` ***Vybrat Soubor*** – zvolení testovaného souboru.
2. `Okno` ***Hlavní okno*** – prostor pro zobrazení videa.
3. `Tlačítko` ***Spusť*** – zapnutí/vypnutí přehrávání videa (tlačítko mění svůj popisek dle jeho momentální činnosti).
4. `Tlačítko` ***Pomoc*** **?** – zobrazí okno popisující jednotlivé funkce programu
5. `Posuvník` ***Timeline*** – ukazuje délku videa, aktuální pozici a načtená data z videa, video lze posouvat.
6. `Okno` ***značek*** – vykresluje ikony značek nalezených v obraze.
7. `Okno` ***objektů*** – vykresluje obrázky nalezených objektů z obrazu (například: auta, chodci a nákladní vozy).
8. `Okno` ***šipek*** – zobrazují směr, ve kterém se nacházejí nalezené objekty.

9. `Tlačítko` ***Nastavení*** 

Obrázek 42 - GUI nastavení

Nastavení obsahuje:

10. `Přepínač` ***Rychlost přehrávání*** – volba rychlosti přehrávání videa.
11. `Textové pole` ***Zobrazovat nalezené objekty*** – určuje, kolik snímků se bude nalezený objekt zobrazovat i když v následujícím snímku již nebude nalezen.
12. `Textové pole` ***Vyhledávat dopravu*** – určuje počet snímků, které se vynechají při vyhledávání složitých objektů (vozidla, chodci, …).
    1 znamená každý snímek, 2 znamená ob snímek, atd…
    Oblasti nalezených objektů budou vyznačeny v následujících x snímcích, nezávisle na jejich skutečném obsahu. Zvýšení této hodnoty zvyšuje rychlost programu, ale snižuje přesnost nalezených objektů.
13. `Textové pole` ***Vyhledávat značky*** – stejná funkcionalita jako předchozí textové pole, ale týká se vyhledávání značek.
14. `Tlačítko` ***Vyhledávat i složité objekt*** – Zapne/Vypne vyhledávání objektů dopravy (vozidla, chodci, …).
    Vypnutí silně zvýší rychlost programu, ale vyhledávají se poté pouze dopravní značení.
15. `Textové pole` ***Velikost bufferu*** – určuje počet snímků, které se předem načítají a ukládají do operační paměti.

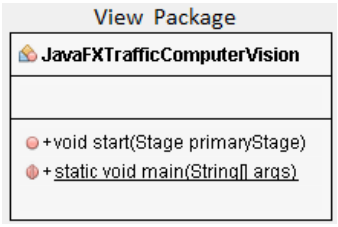
V. Technická dokumentace

Program na rozpoznání objektů v dopravě.

Projekt je složen z balíčku: *view*, *controller*, *videoProcessing a models*. Balíček *view* obsahuje spouštěcí třídu a fxml grafický soubor. Balíček *controller* obsahuje třídy komunikující a ovládající grafické rozhraní a třídu obsahující data pro jednoduché rozšíření programu. Hlavním balíčkem je *videoProcessing*, který obsahuje jednotlivé třídy reprezentující vlákna pro zpracování videa. Balíček *models* obsahuje asistující datové třídy, které nezpracovávají video.

Další částí projektu jsou externí knihovny OpenCV (verze 4.0.0) a tensorflow (verze 1.5.0), které jsou umístěny v Libraries. Tyto knihovny poskytují implementaci velkého množství metod Computer Vision.

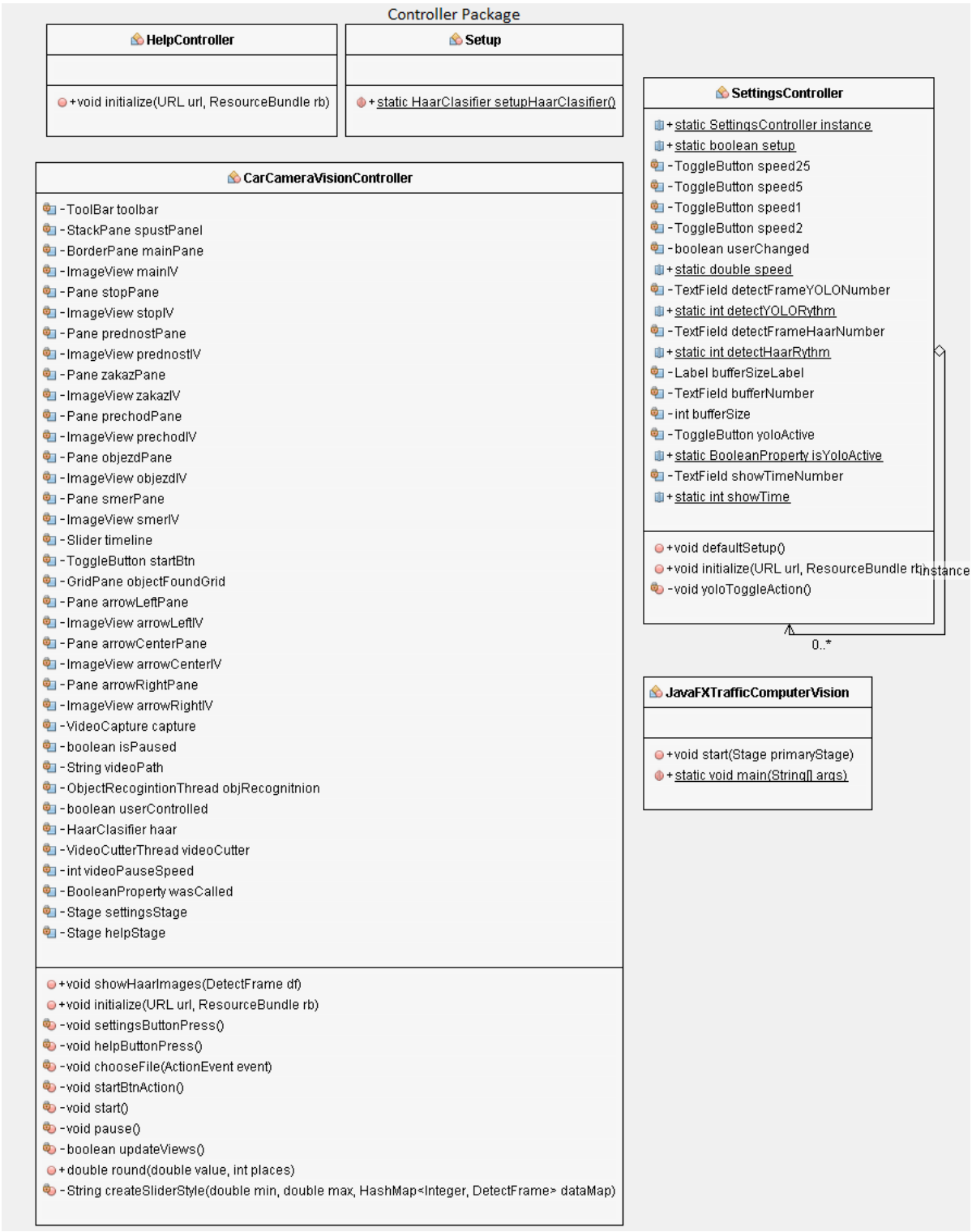
Balíček *view:*



Obrázek 43 - View Package

- `JavaFXTrafficComputerVision` – Hlavní třída spouštějící projekt, která je rozšířením JavaFX třídy Application.
- `CarCameraVision.fxml` – JavaFX `xml` soubor definující hlavní okno programu.
- `Settings.fxml` – JavaFX soubor obsahující `xml` definici GUI nastavení.
- `Help.fxml` – JavaFX `xml` soubor definující okno s nápovědou.

Balíček *controller:*

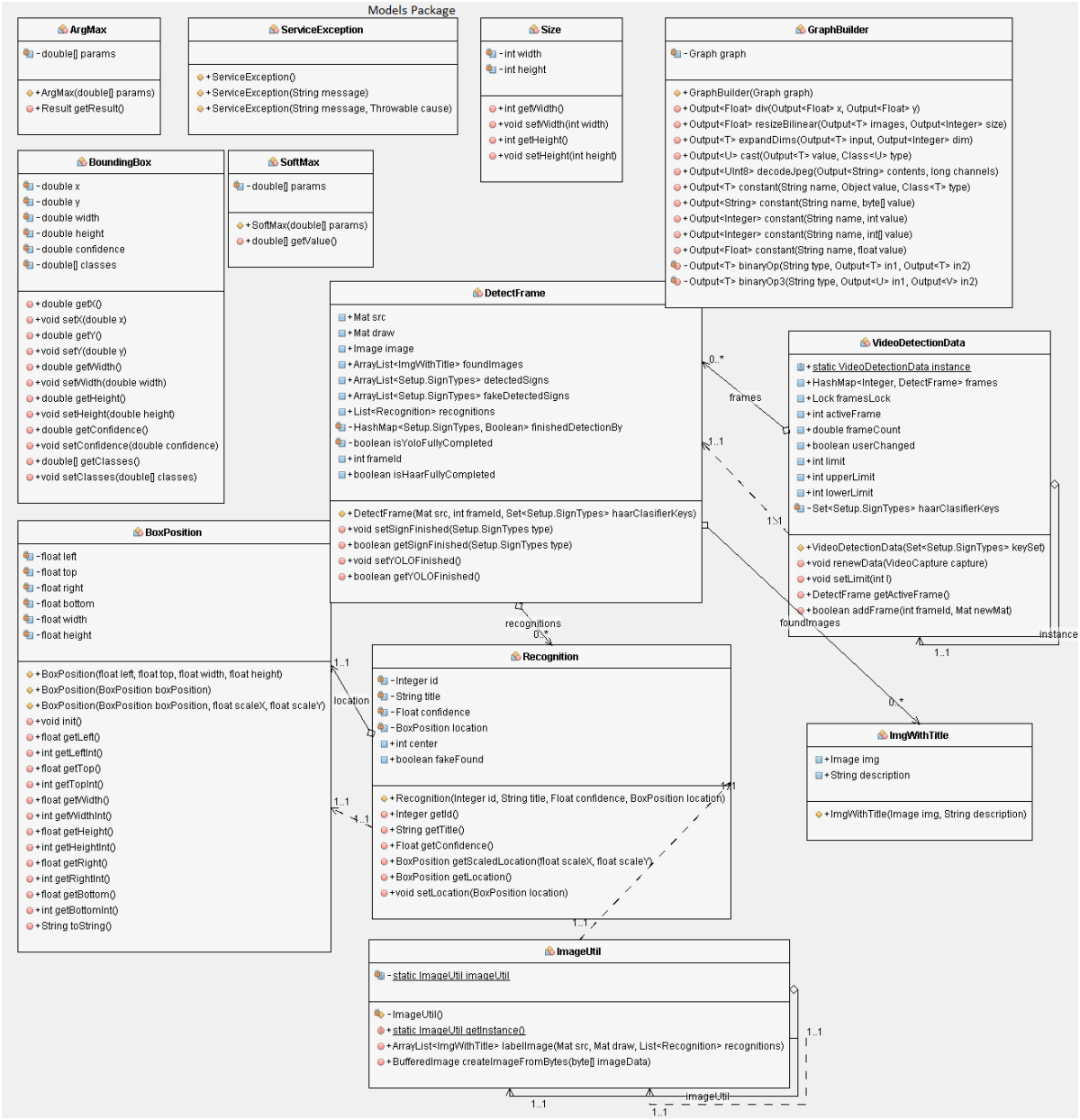


Obrázek 44 - Controllers Package

- `CarCameraController` – Je kontrolní třídou, která obsluhuje grafický FXML soubor hlavního okna. Využívá dat poskytnutých z tříd z balíčku videoProcessing. Implementuje rozhraní Initializable.

- `SettingController` – Je kontrolní třídou, která obsluhuje grafický FXML soubor okna nastavení. Jiné třídy využívají data z této třídy pro nastavení svého chování. Implementuje rozhraní Initializable.
- `Setup` – V této třídě se dá jednoduše upravit a přidat nastavení vyhledávaných značek.

Balíček *models:*

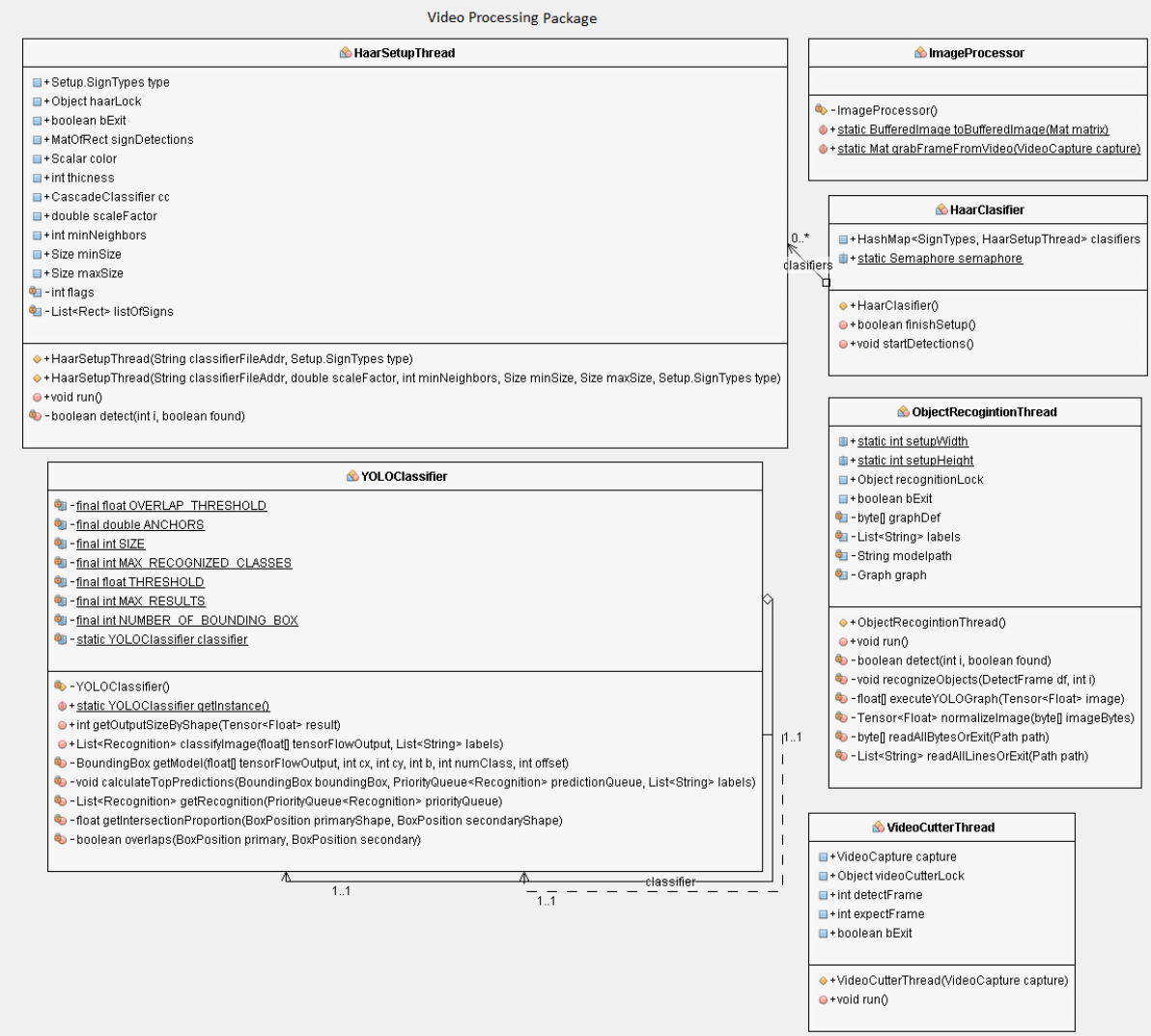


Obrázek 45 - Models Package

- `ImgWithTitle` – Model obsahující propojení nalezeného objektu a jeho popisku.
- `Recognition` – Model obsahující pozice a jistotu nalezeného objektu.

- `DetectFrame` – Model obsahující definici jednoho snímku videa a informace o všech dokončených/nedokončených nálezech na něm.
- `VideoDetectionData` – Model spojuje sérii `DetectFrame` a tím definuje celé video, využívá buffer limity a kontroluje, zda lze přidat snímek a jaký.
- `Ostatní modely` – Ostatní modely jsou nedůležité obalové prvky definující malé součástky ostatních modelů.

Balíček *videoProcessing:*



Obrázek 46 - Video Processing Package

- `ImageProcessor` – Statická třída převádějící a zpracovávající video soubory tak, aby s nimi mohly pracovat ostatní třídy nebo knihovna OpenCV.
- `VideoCutterThread` – Třída reprezentující vlákno, které vytváří objekty z videa a připojuje je do modelu `VideoDetectionData`.

- `YOLOClassifier` – Třída definující postup průchodu obrazem. Založeno na principu YOLO.
- `ObjectRecognitionThread` – Třída reprezentující vlákno, které prochází nalezené obrazy a vyhledává v nich objekty (Vozidla, Chodci, …). Využívá třídu `YOLOClassifier` k průchodu obrazu a cizí knihovny TensorFlow k nalezení objektů.
- `HaarClasifier` – Třída spojující jednotlivá vlákna `HaarSetupThread` pro jednodušší ovládání všech Haar klasifikátorů.
- `HaarSetupThread` – Třída reprezentuje vlákno jednoho kaskádového klasifikátoru. K identifikaci objektu využívá knihovny OpenCV a vlastního klasifikačního modelu, dodaného ve formátu `xml` .

Program obsahuje modely na vyhledáváni šesti značek. Pro snadnou rozšiřitelnost lze jednoduše přidat vlastní model definovaný v souboru ve formátu `xml`. V třídě Setup se nachází metoda `setupHaarClasifier()` definující, které cascádové haar klasifikátory se budou v programu používat. Pro přidání vlastního klasifikátoru stačí vytvoření nového `HaarSetupThread`, který přijímá parametry:

- `String classifierFileAddr` – adresu, kde se nachází `.xml` soubor definující klasifikátor,
- `SignTypes` type – popisný typ značky, využit pro jeho ostatní zpracování. SighnTypes je definován v Enumu, který se nachází na začátku souboru Setup.

```
HaarSetupThread KruhovyObjezd = new HaarSetupThread("data/cascades/KruhovyObjezd-s14.xml", SignTypes.KruhovyObjezd);
setup.clasifiers.put(KruhovyObjezd.type, KruhovyObjezd);
```

Obrázek 47 - Přidání nového Haar modelu

Současně lze nastavit barvu, kterou je značka označována a upřesnit některé parametry, které mohou zvýšit či snížit přísnost vyhledávání. Větší volnost parametrů může zvýšit dobu detekce. Parametry jsou:

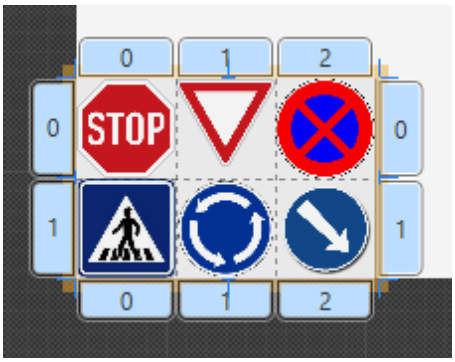
- `ScaleFactor` – poměr možných velikostí hledaného objektu vůči naučenému modelu. Hodnota v rozsahu 1-2. Nízká čísla (např. 1.01) způsobují zpomalení vyhledávání.
- `MinNeighbors` – udává minimální množství nalezených rozměrů a rotací značky než je vyhodnocena jako skutečná. Větší číslo sníží falešné nálezy, ale může způsobit zhoršení skutečných nálezů.

- `MinSize` – minimální velikost hledaného objektu v pixelech.
- `MaxSize` – maximální velikost hledaného objektu v pixelech.

```
KruhovyObjezd.color = new Scalar(255, 20, 128);
KruhovyObjezd.scaleFactor = 1.15;
KruhovyObjezd.minNeighbors = 4;
KruhovyObjezd.minSize = new Size(15, 15);
KruhovyObjezd.maxSize = new Size(50, 50);
```

Obrázek 48 - Nastavení Haar modelu

Poté jen stačí přidat značku do `setup.classifiers` a identifikace je funkční. Pro zobrazování značky v GUI je potřeba přidat vlastní `ImageView`, definující vaší novou značku.

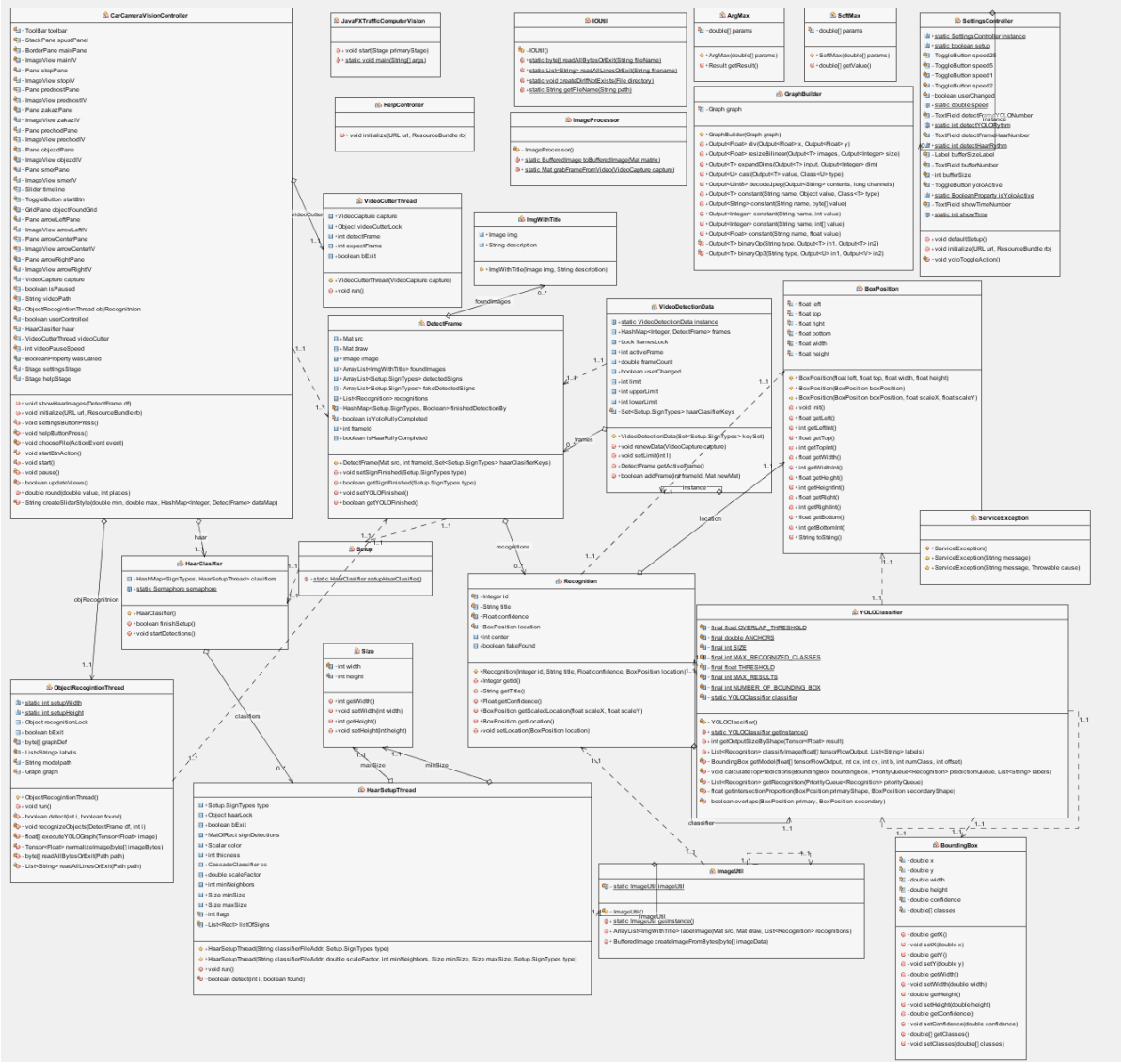


Obrázek 49 - Přidání ImageView do GUI

Pro zobrazení je nutné propojit požadované `ImageView` s odpovídajícím klasifikátorem přidáním nového řádku do metody `showHaarImages()`, která se nachází na začátku třídy `CarCamereVisionController`. Jako příklad propojení 6 značek, které se v programu již nacházejí.

```
public class CarCameraVisionController implements Initializable {

    public void showHaarImages(DetectFrame df) {

        stopIV.setVisible(df.detectedSigns.contains(SignTypes.Stop) || df.fakeDetectedSigns.contains(SignTypes.Stop));

        zakazIV.setVisible(df.detectedSigns.contains(SignTypes.ZakazZastaveni) || df.fakeDetectedSigns.contains(SignTypes.ZakazZastaveni));

        prednostIV.setVisible(df.detectedSigns.contains(SignTypes.PrednostVJizde) || df.fakeDetectedSigns.contains(SignTypes.PrednostVJizde));

        prechodIV.setVisible(df.detectedSigns.contains(SignTypes.Prechod) || df.fakeDetectedSigns.contains(SignTypes.Prechod));

        objezdIV.setVisible(df.detectedSigns.contains(SignTypes.KruhovyObjezd) || df.fakeDetectedSigns.contains(SignTypes.KruhovyObjezd));

        smerIV.setVisible(df.detectedSigns.contains(SignTypes.PrikazanySmerVprave) || df.fakeDetectedSigns.contains(SignTypes.PrikazanySmerVprave));
    }
```

Obrázek 50 - Propojení GUI a modelu



Obrázek 51 - Kompletní UML class diagram