VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

SÉMANTICKÁ SEGMENTACE OBRAZU Z KAMERY MOBILNÍHO ROBOTU

SEMANTIC SEGMENTATION OF MOBILE ROBOT CAMERA IMAGES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. STANISLAV DANIŠ**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. JIŘÍ KREJSA, Ph.D.**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ | FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Stanislav Daniš**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Jiří Krejsa, Ph.D.**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Sémantická segmentace obrazu z kamery mobilního robotu

Stručná charakteristika problematiky úkolu:

Úloha segmentace obrazu z palubní kamery mobilního robotu ve vnitřním prostředí může přinést podstatné zlepšení jak u navigace, tak především u řešení konkrétních úkolů mobilního robotu. Jednou z ověřených technik sémantické segmentace je použití konvolučních neuronových sítí. Podstatou této práce je implementace a ověření výkonu vybraných konvolučních neuronových sítí s využitím frameworku TensorFlow, který umožňuje použití již natrénovaných sítí i na méně výkonném HW mobilního robotu.

Cíle diplomové práce:

1/ seznamte se s metodami sémantické segmentace obrazu
2/ seznamte se s frameworkem TensorFlow/Keras
3/ vyberte vhodnou metodu s ohledem na výpočetní nároky
4/ metodu implementujte a ověřte na reálných datech

Seznam doporučené literatury:

SEWAK M. et. al., Practical Convolutional Neural Networks: Implement advanced deep learning models using Python, Packt Publishing Ltd, 2018

BALLAR W., Hands-On Deep Learning for Images with TensorFlow: Build intelligent computer vision applications using TensorFlow and Keras, Packt Publishing Ltd, 2018

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku.

V Brně, dne 22. 10. 2021

L. S.

prof. Ing. Jindřich Petruška, CSc.
ředitel ústavu

v z.

doc. Ing. Jaroslav Katolický, Ph.D.
děkan fakulty

Abstrakt
Diplomová práca bola zameraná na riešenie problému sémantickej segmentácie pre kameru mobilného robota, ktorý je postavený na menej výkonnom hardvéry. Výberom a implementáciou vhodnej konvolučnej siete bola docielená predikcia v reálnom čase aj na staršej grafickej karte akou je Nvidia GTX 1050Ti.

Summary
The thesis was focused on solving the problem of semantic segmentation for a mobile robot camera, which is built on less powerful hardware. By selecting and implementing a suitable convolution network, real-time prediction was achieved on an older graphics card such as the Nvidia GTX 1050Ti.

Klíčová slova
Tensorflow, ONNX Runtime, DDRNet 23 slim, sémantická segmentácia,

Keywords
Tensorflow, ONNX Runtime, DDRNet 23 slim, semantic segmentation

DANIŠ, S.*Sémantická segmentace obrazu z kamery mobilního robotu.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2022. 60 s. Vedoucí diplomové práce Ing. Jiří Krejsa, Ph.D.

Prehlasujem že, som túto prácu vypracoval sám za použitia odbornej literatúry, ktorú som uviedol na konci práce v časti Literatúra.

Bc. Stanislav Daniš

Ďakujem rodine, priateľom a známym ktorí ma podporovali a pomáhali mi v priebehu celého štúdia. Ďakujem doc. Ing. Jiří Krejsovi Ph. D. za odborné rady, nápady a pripomienky pri vypracovaní diplomovej práce.

Bc. Stanislav Daniš

Obsah

1. Úvod

Porozumenie toho ako svet okolo nás vyzerá v každodennej situácií, predstavuje pre každého jedinca základný nástroj na vykonávanie akejkoľvek činnosti. Sú to práve získané obrazové dáta, ktoré pre nás predstavujú najdôležitejšie informácie podľa ktorých sa v bežnom živote rozhodujeme.

Posunúť túto schopnosť spracovania obrazu počítačom tak, aby rovnako ako ľudia mohli ďalej využívať nájdené informácie pre konkrétne činnosti začalo už v 60. rokoch dvadsiateho storočia a s postupným rozvojom stále lacnejších a výkonnejších zariadení sa rozvinulo do veľmi širokej oblasti zaujmú. Základný predpoklad, ktorý mal byť splnený, však zostal stále rovnaký. Umožniť spracovať vizuálne dáta do podoby, ktorá je zrozumiteľnejšia a jednoduchšia na ďalšie vyhodnocovanie. Avšak zo zachovanou informačnou hodnotou.

Do oblasti spracovania obrazu patria rôzne podkategórie, ktoré sa líšia hlavne tým na akú úlohu sú zamerané. Segmentácia obrazu tvorí práve jednu z nich, kedy vo výslednom obraze sú celé objekty reprezentované iba jednou farbou (triedou). Čo umožňuje napríklad jednoduchšie nájdenie daného predmetu. Spôsobov akými je možné vykonať takúto operáciu existuje niekoľko, ale v posledných rokoch najväčšiu popularitu medzi obecnou ale aj odbornou komunitou rozhodne vyvolali konvolučné neurónové siete. Nárast záujmu o túto možnosť spracovania vstupných dát je možné priamo prepojiť s posunom v technológií grafických kariet, ktoré predstavujú ich dôležitú súčasť.

Konkrétnym uplatnením segmentácie pomocou konvolučných sieti predstavujú aj mobilné roboty. Predikciu v reálnom čase z kamery takéhoto zariadenia je možné využiť v množstve aplikácií, ktoré robot vykonáva. Akými je samotná navigácia či rozpoznávanie svojho okolia. Najväčším problémom implementácie tohto riešenia je to, že mobilný robot je zariadenie, ktoré je tvorené určitým menej výkonným hardvérom. To znamená, že konvolučný model, ktorý by sa používal, by musel byť dostatočné presný aby spustenie operácie malo vôbec zmysel a zároveň predikovať snímky z kamery v reálnom čase. Toto všetko za predpokladu nižšieho výkonu. Možným riešením je použitie siete z menším počtom parametrov a zvolením vhodného formátu na vykonávanie predikcie.

Úlohou diplomovej práce je preskúmať možnosti sémantickej segmentácie obrazu, s konkrétnejším zameraním na konvolučné neurónové siete. Následne s prihliadnutím na obmedzený hardvér zvoliť taký návrh modelu, ktorý dokáže úspešne spracovať a predikovať výsledky z kamery mobilného robota v reálnom čase.

Pre túto prácu je taktiež nutné vybrať a anotovaním vytvoriť vhodné vstupné dáta, ktoré obsahujú dostatočné veľkú množinu rôznych objektov. Tento celý proces implementovania, trénovania a testovania vytvoreného modelu na reálnych dátach bude spracovaný pomocou frameworku Tensorflow.

Posledným bodom je vytvorenie jednoduchej aplikácie na overenie funkčnosti natrénovaného riešenia na menej výkonnom hardvéry.

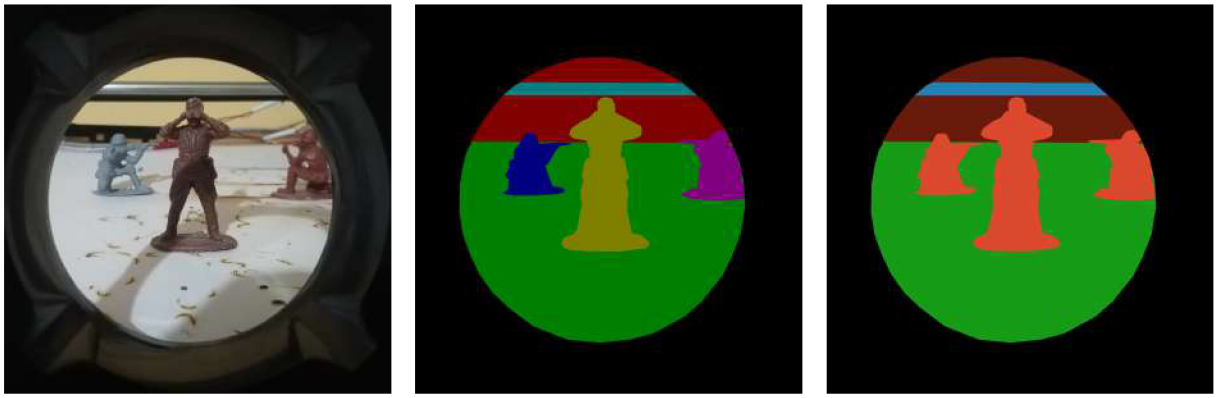
2. Rešerš

Kapitola rešerš sa zaoberá rozborom všetkých potrebných informácií a technologických postupov, ktoré plynú z cieľov zadania a boli nevyhnuté na dosiahnutie vypracovania diplomovej práce. Prvá časť kapitoly bola zameraná na zoznámenie sa s rôznymi možnosťami segmentácie obrazu. Zatiaľ čo druhá časť sa zaoberala potrebnými nástrojmi v rámci riešenia danej problematiky

2.1. Metódy segmentácie obrazu

Segmentácia sa zaraďuje ako jedna z oblastí spracovania obrazu, kde operáciou je proces separovania vstupných dát (obrazu) do jednotlivých segmentov (oblasti/ objektov/ tried). Účelom operácie je rozdeliť či zmeniť reprezentáciu obrazu na niečo, čo je jednoduchšie na ďalšie spracovanie. Ideálnym výstupom je obraz rovnakých rozmerov (zachovanie rozlíšenia), kde každému pixelu bola priradená trieda z dopredu určenej množiny. Výhodou takto spracovaného výstupu (zjednodušeného) je možnosť ľahšej analýzy a následného použitia v konkrétnych aplikáciach. Na obr. 2.1c je možné vidieť ukážku segmentovaného obrazu. [1, 2]

Nasledujúci obr.2.1 obsahuje ukážku základného rozdelenia segmentácia na sémantickú a instantnú. Pri sémantickej (obr. 2.1c) sú všetky objekty na obraze, ktoré patria do rovnakej triedy reprezentované jednou farbou. Narozdiel od instantnej (obr. 4.5b), u ktorej platí, že každý objekt na obraze je tvorený vlastnou farbou aj v prípade ak dva objekty patria do rovnakej triedy. [1, 2]



(a) Nespracovaný obraz (b) Instantná segmentácia (c) Sémantická segmentácia

Obr. 2.1: Rozdelenie segmentácie obrazu

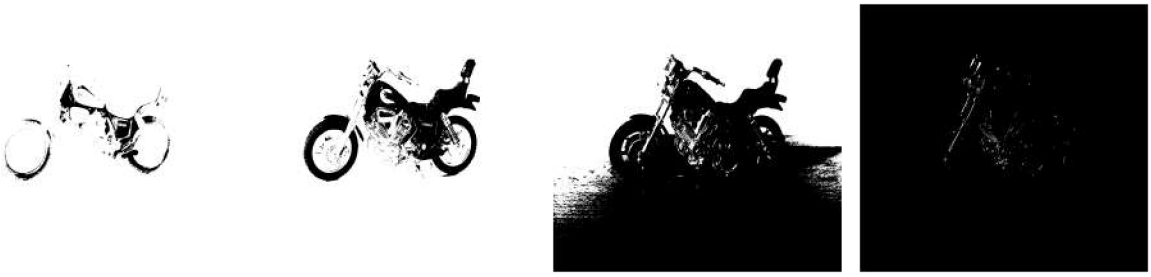
2.1.1. Prahovanie

Najzákladnejšia možnosť segmentovania je prahovanie (”thresholding”). Metóda využíva čiernobieleho obrazu a jeho histogramu na rozpoznanie hranice medzi objektom a pozadím (hodnoty pixelov). Práve vďaka histogramu je možné nájsť prah, ktorý tieto dve scény

oddeľuje a vytvoriť binárnu masku. Princíp prahovania popisuje rov. 2.1, kde y je výsledný pixel, x je vstupná hodnota pixelu a th je zvolená hodnota prahu.

$$y = \begin{cases} 1 & (th > x) \\ 0 & (th \leq x) \end{cases} \tag{2.1}$$

Z rov 2.1 a z popisu vyplýva, že voľba hodnoty prahu priamo ovplyvňuje výslednú segmentáciu. Efekt postupného zvyšovania prahu bol ukázaný na obr. 2.2. [3, 4, 5]



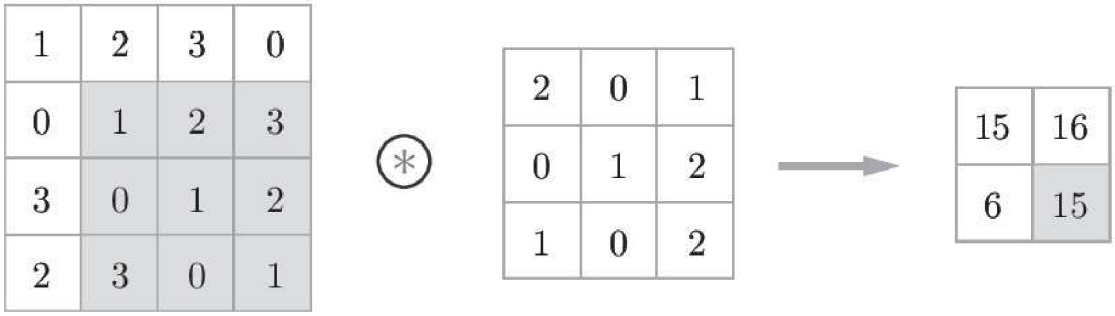
(a) Prah 0.1 (b) Prah 0.3 (c) Prah 0.6 (d) Prah 0.9

Obr. 2.2: Voľba prahu

Nevýhoda pozostáva v nutnosti ručne zvoliť ideálny prah, ktorý zvládne segmentovať celý obraz, čo prakticky nie je vždy možné. Rozširujúce metódy ako Otsu či adaptívne prahovanie čiastočne riešia túto nevýhodu, možnosťou algoritmický zvoliť prah či zvoliť viac ako jednu hodnotu prahu na rovnaký obraz. [3, 4, 5]

2.1.2. Detekcia hrán

Segmentovania pomocou nájdených hrán objektov funguje na základe detekovania nespojitosti v jase, ktoré sa vyskytuje práve medzi prechodom rozdielnych objektov. Aplikovaním operácie konvolúcia zo zvoleným filtrom na vstup (obr. 2.3) sú tieto hrany nájdené. [6, 5]



Obr. 2.3: Konvolúcia [7]

Nutnosť výberu či vytvorenia vhodného filtra predstavuje najväčšiu nevýhodou operácie, keďže filter je nutné definovať tak, aby dokázal určiť hrany zvoleného objektu na obraze (obr. 2.4- po aplikácií filtru na detekciu vertikálnych a horizontálnych hrán). [6, 5, 8]

(a) Nespracovaný obraz (b) Horizontále hrány (c) Vertikálne hrány

Obr. 2.4: Detekcia hrán

2.1.3. Detekcia oblasti

Metóda pre nájdenie oblastí na obraze predpokladá, že susediace pixely v tom istom susedstve zdieľajú navzájom rovnaké vlastnosti a črty. Základným postupom je definovať podmienky podobnosti (farba, intenzita) medzi dvoma pixelmy a iteratívnym spôsobom porovnávať a nachádzať tieto podobnosti. A na tomto základe vytvoriť segmentačnú masku. Samotná metóda sa rozdeľuje na dve kategórie: rastúca oblasť (Region Growing) a rozdeľovanie a spájanie oblasti (Region Splitting and Merging). [5, 8]

Rastúca oblasť (Region Growing)

Vstupom je obraz a počiatočné body, ktoré predstavujú súradnice segmentovaných oblasti. Následne sa kontrolujú kritéria podobnosti medzi pixelom z už priradenou oblasťou (v prvom kroku iba zvolené body) a susediacim. Ak tieto podmienky daný pixel splní, je priradený do rovnakého segmentu. Proces sa opakuje a daná trieda rastie, pokiaľ neobsahuje všetky pixely, ktoré tieto podmienky spĺňajú. [5, 8, 9]

(a) Nespracovaný obraz (b) Tolerancia podobnosti 0.07 (c) Tolerancia podobnosti 0.28

Obr. 2.5: Ukážka rastúca oblasť (region growing)

Nevýhodou metódy je nutnosť voľby polohy vstupných bodov, ktoré môžu byť ovplyvnená väčším/ menším šumom v danom mieste na obraze. [5, 8, 9]

Rozdeľovanie a spájanie oblasti (Region Splitting and Merging)

Algoritmus pracuje na základe postupného rozdeľovania a kontrolovania podmienok podobnosti na celom obraze. V prvom kroku tvorí celý obraz iba jednu oblasť (segment), tá

je následne celá skontrolovaná či obsahuje iba pixely, ktoré sú si navzájom podobnostne blízke a na základe výsledku je rozdelená na ďalšie pod oblasti. Proces zastaví delenie v momente, ak sú si všetky pixely v danej oblasti podobne.

Druhou možnosťou je oblasti spájať. Kedy v prvom kroku je každý pixel zobraný ako jedna samostatná oblasť a spojenie nastane ak spĺňajú zadané podmienky. Proces sa zástavy vo chvíli ak na obraze zostali už iba segmentované oblasti, ktoré zdieľajú rovnaké vlastnosti. [5, 8]

2.1.4. Zhlukovacie algoritmy (Clustering)

Zhlukovacie algoritmy na spracovanie obrazu patria do oblasti strojového učenia a ich cieľom je rozpoznať podobné vlastnosti (features) objektov a priradiť im odpovedajúci zhluk (cluster). Najpoužívanejšími algoritmami s tejto oblasti pre úlohu segmentácie sú K-means či Fuzzy C Means. [8]

K-means

V prvom kroku je pre algoritmus potrebné zvoliť počet zhlukov (hodnota K) do ktorých sa majú vstupné dáta roztriediť. Nasledujúci postup opisuje fungovanie K-means.[8, 10]

1. Voľba náhodných stredov pre každý zhluk.
2. Výpočtom získať všetky vzdialenosti medzi bodom a stredmi.
3. Priradenie bodu do najbližšieho zhluku
4. Upraviť stred na základe priemeru všetkých novo priradených bodov.
5. Opakovať postup od 2. bodu.

(a) K = 2 (b) K = 3 (c) K = 6

Obr. 2.6: Ukážka K-means algoritmus

Cely proces sa zastaví v momente, keď sa priradené hodnoty v zhlukoch prestanú meniť, alebo nebol dosiahnutý maximálny zadaný počet opakovaní. [8, 10]

Posledná z rozoberaných metód segmentácie obrazu je kategória **konvolučných neurónových sieti**. Táto metóda bola podrobnejšie rozobraná v podkap. 2.2 a 2.3.

2.2. Konvolučné neurónové siete

Konvolučné neurónové siete (konvolučné siete) predstavujú podskupinu klasických neurónových sieti s primárnym zameraním na úlohy spracovania obrazu (klasifikovanie, detekcia, atď.). K tomuto vyžívajú niekoľko vrstiev, ktoré umožňujú lepšiu reprezentáciu vstupných dát (obraz). Ich hlavnou výhodou je, že výstup z jednotlivých vrstiev nestráca informáciu o tom ako boli pixely na obraze na sebe priestorovo závislé. To tvorí kontrast s klasickou neurónovou sieťou, u ktorej by bola práve táto informácia čiastočne stratená, keďže ako vstup sa používa jeden stĺpec hodnôt. Tento nedostatok je riešený pridaním aspoň jednej konvolučnej vrstvy a tým vytvorením konvolučnej neurónovej siete. [11, 12, 7, 13]

2.2.1. Vrstvy konvolučnej siete

Nasledujúca podkapitola predstavuje fungovanie jednotlivých vrstiev, ktoré sa často vyskytujú práve v konvolučných modeloch.

Konvolučná vrstva

Princípom konvolučnej vrstvy je aplikovať konvolúciu (rov. 2.2) filtru (kernel) na vstupný obraz. Táto operácia je ukázaná na obr. 2.3.

$$G[i,j] = h * F = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v]F[i-u, j-v] \tag{2.2}$$

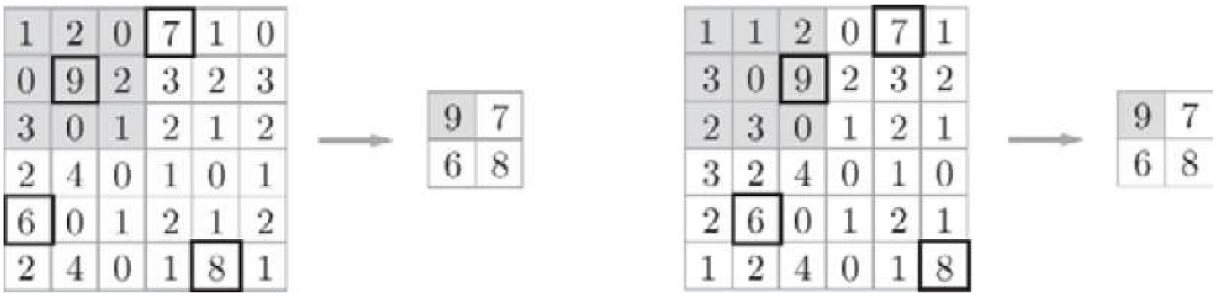
Výstupný obraz je získaný postupným posúvaním a aplikovaním filtru danej veľkosti ($sirka_f$ x $vyska_f$) na obraz ($sirka_o$ x $vyska_o$). Z princípu opisu operácie a z obr. 2.3 plynie, že hodnoty váh, ktoré kernel obsahuje rozhodujú o tom, aké hodnoty na výstupe vrstva produkuje. Proces fungovania sa podobá na metódu detekovania hrán, avšak s rozdielom, že parametre pre filter sú získané vďaka ich postupnému upravovaniu v procese trénovania konvolučnej siete a tým nájdenie najlepších parametrov pre daný problém. [11, 12, 7, 13]

Maxpool vrstva

Operácia maxpool má za úlohu zachovať výrazné črty objektov na obraze a dosiahnuť invarianciu voči posunutiu, ale zároveň znížiť výpočtovú náročnosť siete. Vrstva toto rieši zmenšením vstupného obrazu, čo zaisťuje zníženie požiadavok na hardvér.

Operácia zmenšuje vstup postupným pohybom definovaného okna ($sirka_max$ x $vyska_max$) po obraze a uložením vždy iba najväčšej hodnoty[1], ktorá sa v danom okne práve nachádza. [11, 12, 7, 13]

[1] môže byť zobraná aj priemerná hodnota vypočítaná z hodnôt v okne- averagepool



Obr. 2.7: Maxpool vrstva [7]

Týmto spôsobom sa na obraze zachovávajú dominantné črty objektov a zároveň je dosiahnutá invariancia voči posunutiu. Čo znamená, že výstup z maxpool sa nezmení, aj keď nastane malý posun objektu na obraze (obr. 2.7). Alternatívne je možné túto vrstvu nahradiť konvolúciou, avšak toto ma za následok určité zvýšenie výpočtovej náročnosti, keďže operácia maxpool neobsahuje žiadne parametre na učenie. [11, 12, 7, 13]

Normalizačná vrstva (Batch normalizácia)

Použitím normalizácie je zaistené rovnomerné rozloženie dát po výstupe z každej vrstvy. Normalizácia je vykonaná tak, aby priemerná hodnota výstupu bola nula a rozptyl jedna. Posledným krokom je využitie nasledujúcej rovnice.

$$y_i \leftarrow \gamma \hat{x}_i + \beta \tag{2.3}$$

Kde $\hat{x}$ je normalizovaný vstup, y_i je výstup a γ, β sú parametre, ktoré sa sieť učí počas trénovania. Tieto parametre tak dovoľujú aby normalizované dáta zmenili škálu a posunuli sa tak ako to danej vrstve najviac vyhovuje. Výhodou tejto operácie je získanie rýchlejšieho a stabilnejšieho procesu trénovania. [11, 12, 7, 13]

Softmax

Táto aktivačná funkcia ráta relatívnu pravdepodobnosť vstupu, takže výstupom je pravdepodobnosť pre každú možnú triedu (úloha klasifikovanie viacerých tried) v rozsahu <0, 1>, kedy ich súčet je rovný jednej.

$$\text{softmax}\left(z_i\right) = \frac{\exp\left(z_i\right)}{\sum_j \exp\left(z_j\right)} \tag{2.4}$$

Kde z je vstupná hodnota do funkcie. Táto vrstva sa najčastejšie používa ako posledná v architektúre modelu (výstup siete). [11, 12, 7]

ReLU

Usmernená lineárna aktivačná funkcia[2] (Rectified linear) vracia ako výstup y, hodnotu vstupu x, ak vstup je väčší ako nula, inak je výstupom nula. [11, 12, 7, 13]

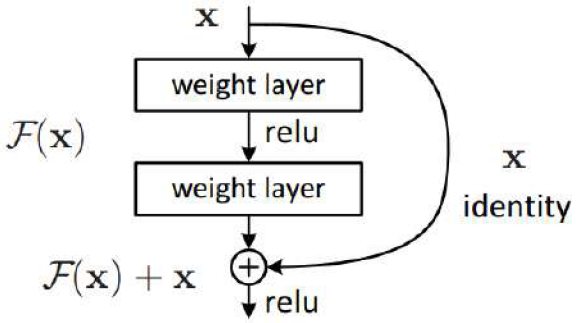
$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{2.5}$$

Výhodou funkcie je možnosť jednoduchého výpočtu (zníženie času trénovania) a zároveň linearitá v kladnom smere zaisťuje, že funkcia nebude saturovať vstupné hodnoty.

Nevýhodou je, že ak hodnoty na vstupe do funkcie začnú byť záporné, výstupom bude vždy nulu, čo znamená, že daná časť sieť sa zasekne na tejto hodnote (nebudú sa meniť váhy) a prestane byť užitočná (problém mrtvé ReLU- dead ReLU). Modifikované funkcie ako leaky ReLU a ELU tento problém čiastočne riešia. [11, 12, 7, 13]

2.2.2. Reziduálny blok

Neurónová sieť predstavuje univerzálny aproximátor, ktorý by mal byť schopný naučiť sa akúkoľvek funkciu. Avšak kvôli niekoľkým problémom, ktoré sa v sieťach s hlbokými vrstvami objavujú, tento predpoklad neplatí (degradačný problém). [14, 15, 16]



Obr. 2.8: Reziduálny blok [15]

Reziduálny blok rieši tento problém preskočením spojenia (skip connection- residual, obr. 2.8). Nasledujúce rovnice, kde R je zostatok (residual), x je vstup a H je skutočné rozdelenie popisujú ako tento blok funguje. [14, 15, 16]

$$\begin{aligned} R(x) &= H(x) - x \\ H(x) &= R(x) + x \end{aligned} \tag{2.6}$$

Reziduálny blok sa snaží naučiť skutočný výstup modelu, avšak z obr. 2.8 a rov. 2.6 plynie, že vrstvy v bloku sa snažia naučiť zostatok ($R(x)$). V porovnaní s klasickou sieťou, kde sa vrstvy snažia naučiť iba skutočný výstup $H(x)$. [14, 15, 16]

2.2.3. Trénovanie

Proces trénovania konvolučnej siete patrí do kategórie učenia s učiteľom. Úlohou modela počas trénovania je optimalizovať všetky parametre siete (váhy) tak, aby produkovali čo

[2] doslovný preklad

najlepšie výsledky pre danú úlohu. Toto je vykonané postupnou minimalizáciou zvolenej chybovej funkcie a zvyšovaním hodnotenia metriky.

Takéto učenie prebieha na zvolenom datasete dát (učenie s učiteľom), ktorý bol vytvorený a overený z hľadiska správnosti výsledkov na danú problematiku (napr. pre segmentáciu, vytvorenie masky). Takto definovaný dataset obsahuje nazbierané dáta a k nim pridané jediné správne riešenia, z ktorých by sa mala sieť naučiť generalizovať problém.

Na začiatku trénovania, sieť začína s náhodne vygenerovanými parametrami[3]. A postupným spracovaním (vykonanie všetkých operácií na daných dátach) načítanej dávky (batch) z datasetu je vypočítaná veličina, ktorá definuje ako blízko bola predikcia k skutočnému výsledku (forward propagation). Na základe tejto nájdenej chyby sieť mierne zmení parametre (backward propagation) tak, aby bolo možné pre ďalšie dávky dosiahnuť zlepšenie vo výsledkoch. To akým spôsobom sa parametre zmenia závislý na zvolenom optimalizačnom algoritme, ktorý bol použitý (SGD, Adam, atď.). Proces sa opakuje, pokiaľ model nespracuje celý dataset a tým ukončí jednu epochu trénovania. Nasledujúcim krokom je náhodne zamiešanie dát a začať s učením na prvej dávke z premiešaného dasetu. Koniec nastáva po určenom počte vykonaných epoch. [11, 12, 7, 13]

Hyperparametre siete

Parametre v sieti sa rozdeľujú do dvoch kategórií:

1. parametre, ktoré je možné meniť učením (napr. váhy konvolučného filtra) a
2. hyperparametre.

Hyperparametre siete nie je možné optimalizačne meniť počas trénovania a predstavujú nastavenie, ktoré je definované ešte pred samotným učením. Do týchto parametrov napríklad patria:

- veľkosť datasetu, veľkosť dávky (batch), počet epoch,
- voľba aktivačných funkcií,
- voľba optimalizačného algoritmu, voľba kroku učenia.

Voľba týchto parametrov, tak má priamo zásadný dosah na výsledok trénovania, ako aj na rýchlosť trénovania. Nájdenie ideálnych parametrov pre daný model, tak vyžaduje testovanie možných hodnôt a porovnávanie výsledkov modelu pri ich zmene. [11, 12, 7, 13]

2.2.4. Transfer learning

Metóda transfer learning v strojovom učení je spôsob akým je možné trénovať nový model za využitia už natrénovaných váh inej siete. Platí, že medzi úlohami na ktoré sú obidve konvolučné siete zamerané, by mala existovať podobnosť v ich štruktúre.

Ďalším predpokladom pre použitie je čiastočná symetria obidvoch modelov. To znamená, že novo zostavená sieť by mala mať priamo v sebe integrovanú časť architektúry z ktorej berie natrénované parametre. Celkový postup pre využitie tejto metódy je nasledujúci. [17, 18, 13]

[3] taktiež to môžu byť napr. iba nuly

1. Voľba natrénovaného modelu- M_s.
2. Použitie M_s a pridanie ďalších vrstiev, tým definovanie nového model- M_n.
3. Zmrazenie vrstiev M_s (počas trénovanie M_n sa tieto vrstvy nezmenia).
4. Trénovanie M_n.

Hlavnou výhodou tohto prístupu je využitie modelu, ktorý bol natrénoaných na veľkom množstve dát a na hardvéry, ktorý nemusí byť bežne dostupný. Takže je možné získať nový model, ktorý bol trénovaný na menšom datasete, kratšiu dobu, ale s kvalitnejším výsledkom ako keby bolo učenie vykonané klasickým spôsobom. [17, 18, 13]

2.2.5. Fine tunning

Metóda Fine tunning alebo doladenie, je možné aplikovať priamo po metóde transfer learning (podkap. 2.2.4). Princípom je zlepšiť výsledky doladením celej siete. Aplikovanie je nasledujúcim spôsobom.

1. Rozmrazenie vrstiev M_s v M_n.
2. Trénovanie M_n.

Týmto spôsobom je možné dosiahnuť, aby sa pôvodne vrstvy z M_s o čosi viac upravili tak, aby dávali lepšiu predikciu a tým zlepšili celkový výstup modela. Dôležitým predpokladom na použitie doladenia (fine tunning) je mať už celu sieť natrénovanú (M_n). V prípade ak by pridaná časť siete a vrstvy z M_s boli optimalizované spoločne, výhoda transfer learningu by úplne stratila zmysel.[19, 20]

2.3. Konvolučné siete pre úlohu segmentácie obrazu

Nasledujúca kapitola definuje parametre, ktoré sa používajú pri trénovaní ako aj vyhodnocovaní výsledkov segmentácie obrazu pomocou konvolučných neurónových sieti.

2.3.1. Chybová funkcia

Chybová funkcia v trénovaní modelu predstavuje vyjadrenie toho ako sa predikcia (výstup modela) líši od hodnoty, ktorú by mal v ideálnom prípade dosiahnuť (hodnota masky). [7, 13, 12]

Cross Entropy

$$E = -\sum_{k=1}^{k=n} Y_k \log p_k \tag{2.7}$$

Kde Y_k je maska (správna hodnota), p_k predstavuje výstup siete a log je prirodzený logaritmus zo základom 10. Cross entropy meria rozdiel medzi maskou a výstupom modela, kedy funkcie rastie, ak sa hodnota predikcie nezhoduje s požadovaným výstupom. Funkcia cross entropy sa rozdeľuje na niekoľko ďalších modifikácií, ktoré sa používajú na základe

toho, aká úloha je riešená a aký vstup do funkcie požadovaný. [7, 21]

Binary Cross Entropy: Funkcia ma uplatnenie v prípade ak segmentácia obsahuje na výstupe iba dve triedy klasifikácie (0- pozadie, 1- objekt) [7, 21, 22]

Categorical Cross Entropy: Rovnica sa zhoduje s Cross Entropy, avšak požadovaným vstupom je pravdepodobnosť toho ktorá trieda je správnym výsledkom. Toto je ovplyvnené aplikáciou aktivačnej funkcie softmax.

Vstupom do funkcie Cross Entropy ako aj Categorical Cross Entropy je tkz. 'one-hot encoder', ktorý definuje masku ako binárny vektor pre každý pixel o dĺžke n (n - počet tried klasifikovania). Obsah vektora tvoria všetko nulové hodnoty, okrem indexu správnej triedy (na tomto index je uložená jednotka). [7, 21, 22]

Sparse Categorical Cross-Entropy: Rozdiel oproti Categorical Cross Entropy pozostáva v tom, ako je vytvorená maska pre segmentáciu.

Vstupom je už priamo číselné pole tried (0, 1, 2). Tento prístup je vlastne aplikovanie funkciu 'argmax' na binárne pole, čím sa dostane index (trieda), kde mal vektor najväčšiu hodnotu (v prípade binárneho pola je táto hodnota vždy 1). Výhodou je reprezentovania každého pixelu iba jednou veličinou, namiesto poľa hodnôt (hlavne ak klasifikovaných tried je niekoľko). [7, 21, 22, 23]

Vyrovnaná funkcia Cross-Entropy: Táto modifikácia cross entropy funkcie rieši problém nevyrovnaného datasetu[4]. Riešenie pozostáva nahradením hodnoty Y_k, parametrom α. Ten pridáva váhu ku každej triede tak, aby sa znížilo nerovnomerné rozloženie tried (pridanie väčšej váhy malo reprezentovaným triedam). [7, 21, 22, 23]

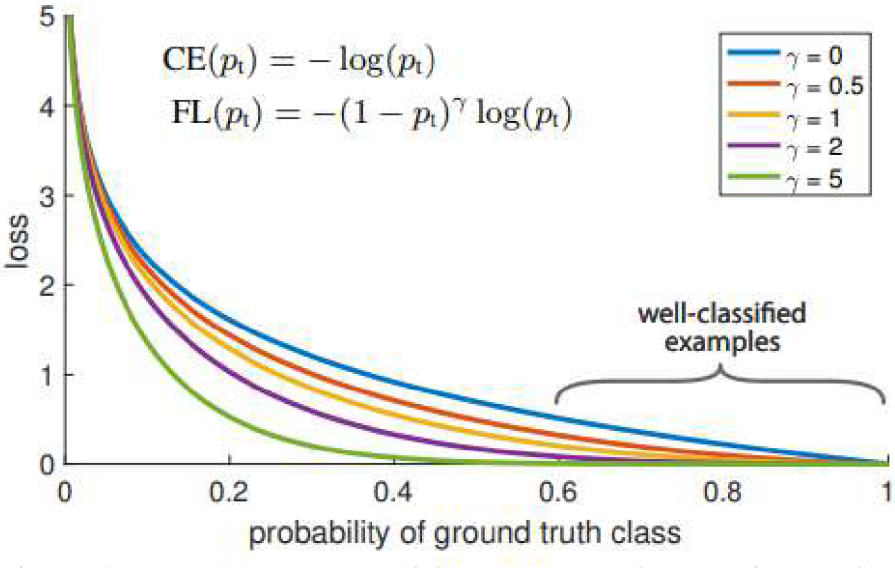
Focal loss

Rovnako ako predchádzajúca funkcia, aj focal loss má hlavné uplatnenie pre trénovanie ma nevyrovnanom datasete. Funkcia je popísaná vzorcom 2.8, ktorý sa skladá z rovnicu cross entropy s pridaným parametrom γ. Ten spôsobuje, že pri zvyšovaní hodnoty tohto parametru sa zmení tvar funkcie (obr. 2.9). [24, 25]

$$\mathrm{FL} = -\sum_{k=1}^{k=1} \alpha \left(k - p_k\right)^{\gamma} \log p_k \tag{2.8}$$

Pre hodnotu $\gamma = 0$ je focal loss definovaná ako rovnica cross entropy. Zmene tvaru funkcie má za následok možnosť väčšieho dôrazu na zle klasifikované triedy.

[4]jedna, alebo viac tried na obraze prevažujú počtom ostatné



Obr. 2.9: Chybová funkcia focal loss [25]

2.3.2. Metrika

Pre rozpoznanie správnosti výsledkov počas a po trénovaní je potrebné zvoliť vhodnú metriku (metriky), ktorá meria úspešnosť celkovej predikcie oproti ideálnemu výstupu (vytvorená maska). [13, 12]

Presnosť

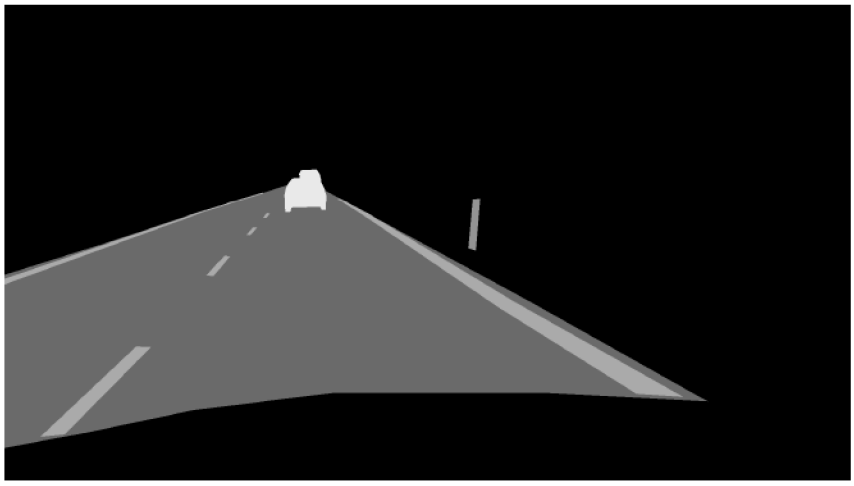
Základnou metrikou nie iba pre segmentáciu je presnosť (accuracy). Výsledok tejto metriky je percentuálna hodnota podľa vzorca 2.9.

$$P = \frac{Pre_s}{M} \cdot 100 \tag{2.9}$$

Kde P je výsledná presnosť, Pre_s je správna predikcia pixelov a M je hodnota pixelov masky. [13, 12]

Problém metriky presnosť

Používanie presnosti (accuracy) ako metriky pre úlohu segmentácia však môže vo výsledku skresľovať to ako dobre model predikuje. Toto nastane v prípade ak používaný dataset nemá rovnomerné reprezentované všetky triedy na obraze vo veľkosti plochy, ktorú zaberajú. Ako ukážka nevyrovnaného obrazu je obr.2.10.



Obr. 2.10: Ukážka z nevyrovnaného datasetu

Po analýze obr.2.10 bolo zistené, že obr. obsahuje nasledujúcich 5 tried:

	Rozlíšenie [V x Š]	**Počet [pixel]**	
	1080, 1920	2 073 600	
	Trieda	**Počet pixelov [-]**	**Podiel z celku [%]**
1.	Okolie	1553502	74,9
2.	Cesta	473826	22,9
3.	Cestný pätník	1934	0,09
4.	Cestný pruh	37448	1,8
5.	Vozidlá	6890	0,3

Tabuľka 2.1: Analýza obr. 2.10

Ako je vidieť z tab. 2.1 triedy okolie a cestu sú v obr. zastúpené v oveľa väčšom množstve ako ostatné kategórie. Dokopy tvoria 97,8 % celého obrazu, čo znamená, že ak by model segmentoval všetku cestu a okolie správne a k tomu všetky ostatne triedy označil ako okolie, výsledná presnosť by bola podľa rov. 2.9 práve 97.8 % (okolie a cesta sú správne priradené).

Intersection Over Union

Metrika Intersection Over Union (prienik nad zjednotením) tiež IoU či Jaccard Index je metrika, ktorá zaručuje lepšie ohodnotenie výstupného obrazu, keďže berie do výpočtu každú triedu samostatne a zohľadňuje ako bola označená a tým sa zamedzuje tomu, že by jedna dominantná triedy skreslila celý výsledok. Rov. 2.10 je použitá na výpočet metriky IoU pre jednu triedu. [26, 27]

$$IoU = \frac{A_p \cap A_m}{A_p \cup A_m} = \frac{A_{prie}}{A_s}. \tag{2.10}$$

Kde IoU je výsledná hodnota v percentách, A_p je predikcia, A_m je maska, A_{prie} je prienik a A_s je zjednotenie.

Metrika IoU sa vypočíta pre každú triedu na danom obraze. Čiže nájdená plocha medzi predikciou a maskou (danej triedy) je podelená celkovou spoločnou plochou danej triedy pre predikciu a masku.

Na získanie jedného ohodnotenia metriky pre daný obr. je vypočítaný priemer zo všetkých IoU (všetky triedy). Pre situácia z obr. 2.10, kedy predikcia správne označila okolie a cestu a všetko ostatné ako okolie, bola nasledujúcim výpočtom získaná výsledná hodnota mIoU (mean IoU) 39.00 %:

$$
\begin{aligned}
1.: A_{prie} = 74.90; A_s = (77,15 + 74,90) - 74,90 = 77,15; IoU1 = \frac{74,90}{77,15} = 0.97 \\
2.: A_{prie} = 22.90; A_s = (22,90 + 22,90) - 22,90 = 22,90; IoU2 = \frac{22,90}{22,90} = 1,00 \\
3.: A_{prie} = 0.00; A_s = (0,00 + 0,09) - 0,00 = 0,09; IoU3 = \frac{0,00}{0,09} = 0,00 \\
4.: A_{prie} = 0.00; A_s = (0,00 + 1,80) - 0,00 = 1,80; IoU4 = \frac{0,00}{1,80} = 0,00 \\
5.: A_{prie} = 0.00; A_s = (0,00 + 0,30) - 0,00 = 0,30; IoU5 = \frac{0,00}{0,30} = 0,00 \\
mIoU = \frac{IoU1 + IoU2 + IoU3 + IoU4 + IoU5}{5} = \\
\frac{0,97 + 1,00 + 0,00 + 0,00 + 0,00}{5,00} = 0,39 \cdot 100,00 = 39,00\ \%
\end{aligned}
\tag{2.11}
$$

F1

F1 má uplatnenie v prípade nevyrovnaného datasetu. Metrika je vypočítaná pomocou kombinácie dvoch ďalších metrík: precision a recall[5]. [28, 29]

Precision sa zameriava na to, koľko bolo správne predikovaných z celkového počtu predikcie z danej triedy. [28, 29]

$$P = \frac{TP}{TP + FP} \tag{2.12}$$

Recall počíta koľko bolo správne predikovaných danej triedy zo všetkých predikcií. [28, 29]

$$R = \frac{TP}{TP + FN} \tag{2.13}$$

Skratka	Názov	Význam
TP	True positive- skutočne pozitívne	Pozitívna predikcia, v skutočnosti pozitívna
FP	False positive- falošne pozitívne	Pozitívna predikcia, v skutočnosti negatívna
TN	True negative- skutočne negatívne	Negatívna predikcia, v skutočnosti negatívna
FN	False negaive- falošne negatívne	Negatívna predikcia, v skutočnosti pozitívna

Tabuľka 2.2: Význam skratiek

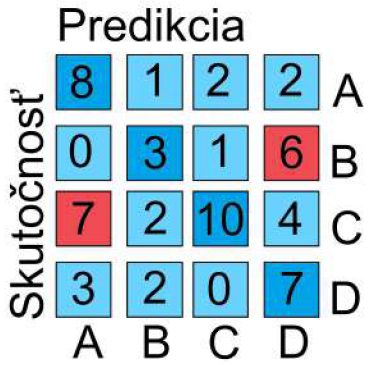
$$P = 2 \cdot \frac{P \cdot R}{P + R} \tag{2.14}$$

[5]v texte boli použité iba anglické názvy, keďže preklad je nejednoznačný

Metrika kombinuje dve spomenuté metriky a tým minimalizuje ich nepresnosť ak by boli použité samostatne.

Confusion matrix

Confusion matrix (matica zmätenia) je technika (metrika), ktorá jednoduchou vizualizáciou vytvára predstavu o tom, ako dobre model vykonal predikciu a poprípade ktoré triedy sa navzájom zle klasifikujú. Výsledkom je matica (obraz), ktorá má počet stĺpcov a riadkov rovnajúci sa počtu predikovaných tried. [13, 30]



Obr. 2.11: Confusion matrix

Jednotlivé predikované hodnoty sa rozložia do matice podľa ich stavu predikcie, kedy riadok predstavuje skutočnú hodnotu a stĺpec predstavuje predikovanú hodnotu. Výhodou takto zobrazeného výsledku, je nájdenie informácií o zlej predikcií, ktorá by inak nemusel byť priamo viditeľná.

2.3.3. Možnosti architektúry siete

V oblasti konvolučných neurónových sieti pre sémantickú segmentáciu obrazu existujú rôzne spôsoby akým je možné definovať sieť. Tieto architektúry sa líšia ako v počte použitých vrstiev, v kvalite dosiahnutých výsledkov tak aj v potrebe výpočtového výkonu (rýchlosť predikcie). Nasledujúca podkapitola zhrnula niektoré z najviac používaných prístupov pre návrh siete určenej na segmentáciu.

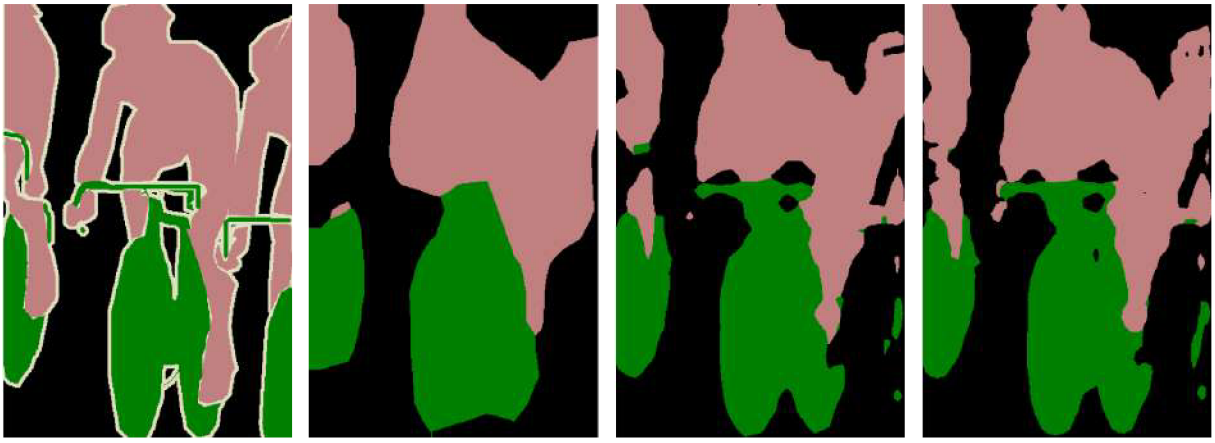
Základnou možnosťou vytvorenia architektúry je umiestniť niekoľkých za sebou radených konvolučných vrstiev s rovnako definovanou veľkosťou výstupu, ktorá sa rovná veľkosti vstupného obrazu. Takáto architektúra nestráca priestorovú informáciu, keďže stále udržuje pôvodnú veľkosť na všetkých vrstvách siete.

Nevýhodou však je vysoká nároky na výpočet, práve kvôli zachovaniu rovnakej veľkosti obrazu. [31]

Fully Convolutional Networks for Semantic Segmentation

Zásadným modelom v oblasti sémantickej segmentácie je všeobecne považovaná architektúra popísaná v článku Fully Convolutional Networks for Semantic Segmentation[6] (FCN) z roku 2014 od autorov Evan Shelhamer, Jonathan Long a Trevor Darrell [32].

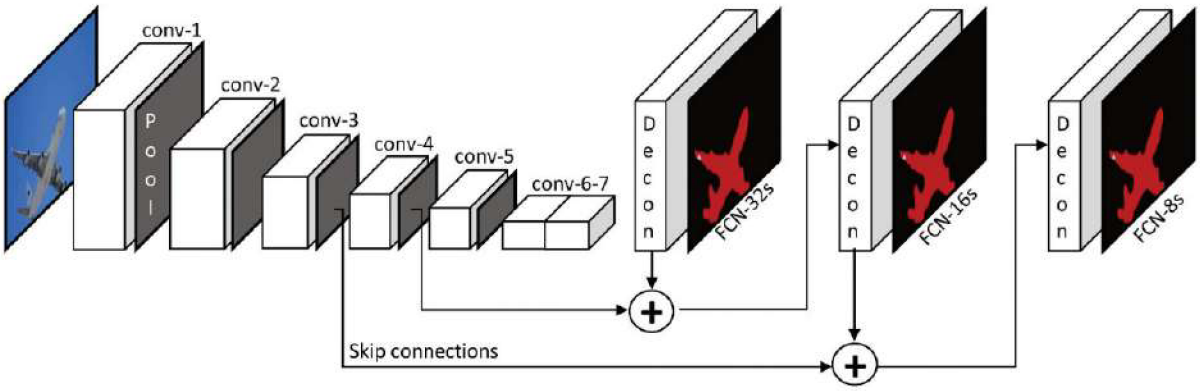
Hlavná časť siete sa skladá z enkóder modulu, ktorý je vytvorený z už natrénovanej siete na úlohu klasifikácie obrazu (AlexNet, VGG alebo GoogLeNet). Ten má za úlohu postupné zmenšovanie obrazu medzi konvolučnými vrstvami a tým získavanie informácie o objektoch na obraze. Na záver bol k výstupu enkóder časti pripojený dekóder, ktorý sa skladá z transponovanej konvolučnej vrstvy, ktorá slúži na celkové zväčšenie obrazu (tak, aby sa veľkosť výstupu rovnala veľkosti vstupu). Obr. 2.12b obsahuje predikciu, ktorá vznikla pomocou siete FCN. [32]



(a) Orig. maska [32] (b) FCN 32s [32] (c) FCN 16s [32] (d) FCN 8s [32]

Obr. 2.12: Výsledky FCN

Nedokonalosť v predikcií (obr. 2.12b) autori riešili modifikovaním siete na možnosť vynechania spojenia (skip connection). Prínos tejto modifikácie je vidieť na obr. 2.12c a 2.12d. [32]



Obr. 2.13: Architektúra FCN [32]

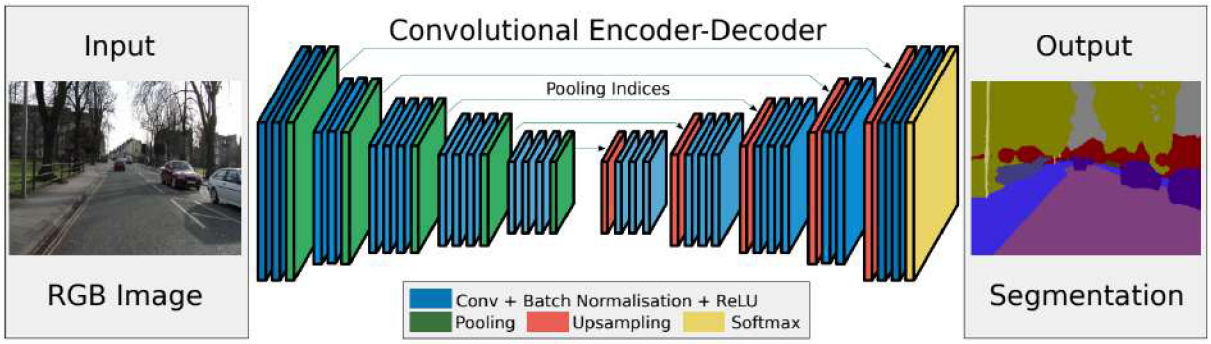
[6]plno konvolučná sieť

Enkóder-Dekóder

Architektúra enkóder- dekodér, ktorá bola použitá na vytvorenie siete FCN je často používaná architektúra pre segmentáciu obrazu. Avšak rozšírená do viac typického usporiadania bola až pomocou siete U-Net. [32, 33]

U-Net architektúra bola zameraná na zväčšenie funkčnosti časti dekodér, ktorá bola v FCN. Modifikácia pozostáva vo vytvorení symetrickej veľkosti dekodéru k časti enkóder (obr. 2.14). To malo zabezpečiť lepšie zväčšovanie obrazu, keďže rekonštrukcia prebiehala postupne a s prenášaním informácií z enkoderu. [34, 33]

Ďalšie možnosti rozšírenia enkóder- dekodér architektúry predstavuje napr. sieť **SegNet**, ktorá rieši zväčšovanie obrazu z výstupu enkóderu, dodatočným prenášaním hodnôt indexov po vykonaní operácie maxpool do symetrickej vrstvy v dekodéri a na základe tejto informácie postupne vytvára zväčšený obraz. [34]



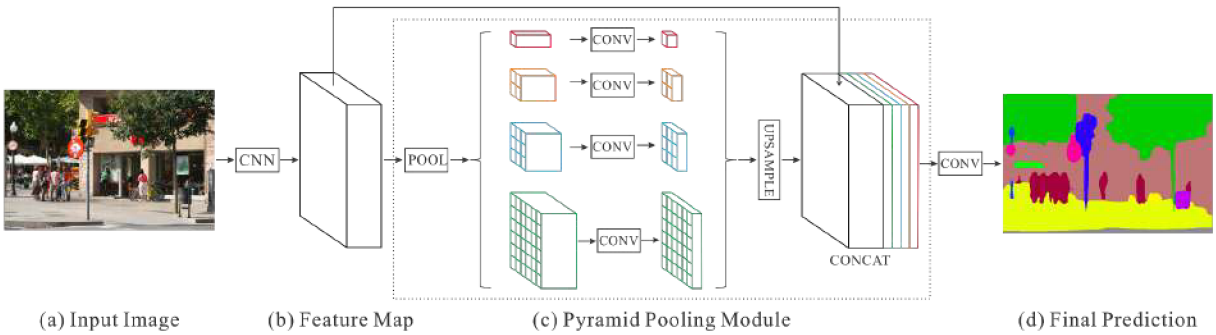
Obr. 2.14: Architektúra SegNet [34]

Pyramid pooling module

Pyramid pooling module (PPM) sa priamo nezaraduje ako architektúra siete, ale skôr metóda, ktorá sa môže využiť v konvolučných sieťach pre sémantickú segmentáciu na dosiahnutie lepších výsledkov.

Modul bol vyvinutá na zachytenie všetkých informácií z obrazu v globálnej mierke, čo nemôže byť docielené, ak je použitá architektúra ako FCN či SegNet. Keďže tieto siete, obraz postupne zmenšuje a tým čiastočne strácajú priestorovú informáciu.

PSPNet architektúra vyžíva k preskúmaniu globálnych súvislosti na obraze práve PPM. Obr. 2.15 zobrazuje túto siet, ako aj PPM. [35]

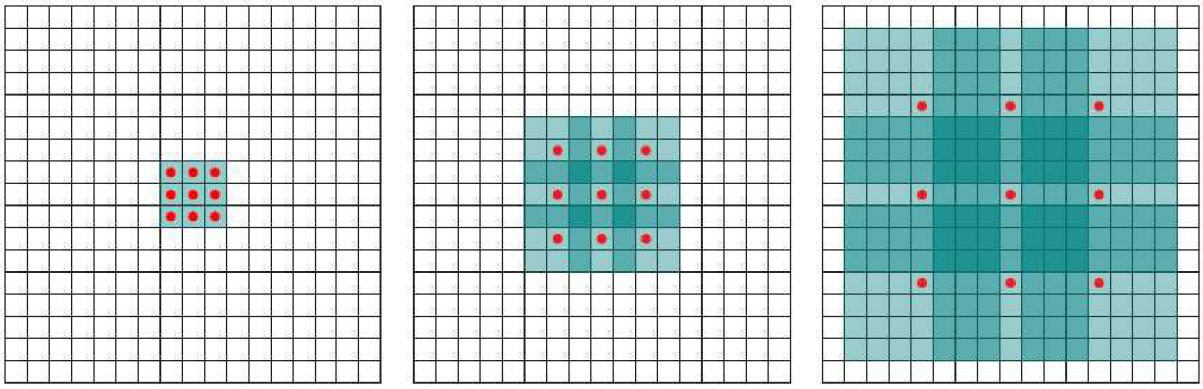


Obr. 2.15: Architektúra PSPNet [35]

PPM obsahuje 4 samostatné konvolučné vrstvy z rozdielnou veľkosťou filtra. Každá z nich vykonáva operáciu na rovnakom vstupnom obraze a následne sú ich výstupy upravené na rovnakú veľkosť a spojené (operácia concat) do jedného. Výhodou modulu je aplikovanie štyroch rôzne veľkých konvolučných filtrov na vstup a tým možnosť získať lokálnu aj globálnu informáciu. [35]

Rozšírená (Dilated) konvolúcia

Rozšírená konvolúcia pracuje podľa obr. 2.16, kedy s rastúcou medzerou (dilation rate), rastie aj rozsah pola na ktorom konvolúcia pracuje, ale pri zachovaní rovnakého počtu parametrov na učenie. [36]



Obr. 2.16: Rozšírená konvolúcia [36]

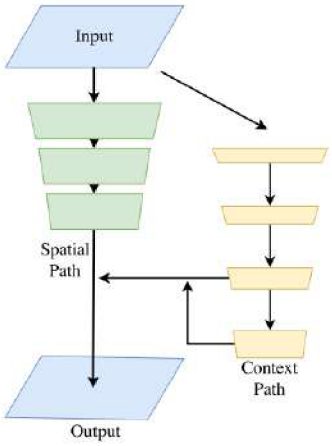
Architektúra, ktorá obsahuje rozšírenú konvolúciu dosahuje dobrých výsledkov, avšak nevýhodou sú vyššie nároky na GPU[7].

Dvoj-vetvová architektúra (two-pathway)

Pre úlohu segmentácie sú od výsledku siete požadované dve hlavné podmienky. Získať informácie o objektoch na obraze (vytvoriť masku) a zachovať pôvodné rozlíšenie vstupného obrazu. Dvoj-vetvová architektúra rieši tieto dva problémy, ich rozdelením na samostatné časti (vetvy). [37]

BiSeNet sieť je vytvorená práve ako dvoj-vetvová architektúra. Jedna cesta zmenšuje obraz a získava informácie o objektoch, zatiaľ čo druhá cesta obsahuje menej vrstiev, ale s vyšším rozlíšením. Vetvy tak vykonávajú operácie nezávisle na sebe a až pred koncovým krokom sú spojené, čím je vytvorený jeden výsledný obraz segmentácie. [37]

[7] grafická karta

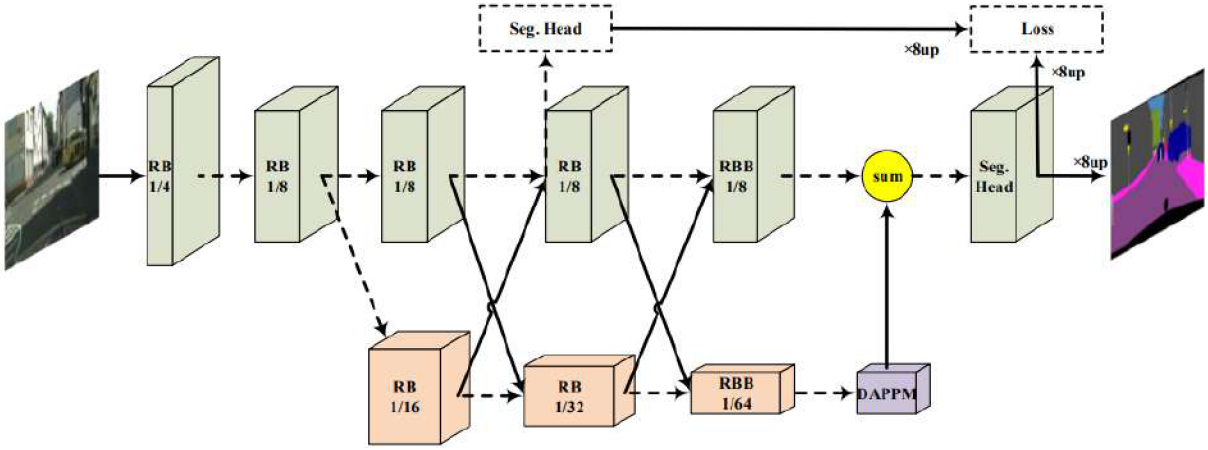


Obr. 2.17: Architektúra BiSeNet [37]

Kombinovaná architektúra

Výskum v oblasti konvolučných sieti určených na segmentáciu sa stal veľmi populárny, s veľkým množstvom navrhovaným modelov. Avšak nie všetky siete môžu byť jednoznačné zaradené, keďže kombinujú viaceré prvky.

DDRNet architektúra čiastočne využíva dvoj-vetvovej architektúry na získanie masky s požadovaným rozlíšením. Avšak ako obr. 2.18 ukazuje, rozdelenie na dve vetvy nastáva až po prvých spoločných operáciach. Navyše samotné cesty sa spolu navzájom niekoľko krát bilaterálne spoja, aby postupne vytvárali lepšie spojenie. [38]



Obr. 2.18: Architektúra DDRNet [38]

Na záver siete je ešte pridaný Deep Aggregation Pyramid Pooling Module (DAPPM), ktorý sa snaží docieliť lepšieho spojenie viacúrovňového kontextu z výstupov siete, ktoré majú malé rozlíšenie. Celkovo DDRNet architektúra dosahuje dobrých výsledkov pri rýchlom procese spracovania obrazu. [38]

2.4. Tensorflow

Tensorflow je framework s otvoreným prístupom od firmy Google zameraný na aplikáciu strojového učenie a hlbokého učenie. Vďaka integrovanému high-level frameworku Keras, ktorý dovoľuje zjednodušenú prácu s celým frameworkom tensorflow je získaná vysoká úroveň flexibility. Kedy pomocou Kerasu, môže byť definovaná architektúra siete rôznymi spôsobmi či práca z vrstvami a tensorflow zaisťuje ich optimalizované výpočty. [39, 40]

2.4.1. Základy tensorflow

Tensorflow obsahuje už veľké množstvo vrstiev a funkcií, ktoré sa bežne používajú v strojovom učení a sú tak priamo pripravené na aplikovanie. Nasledujúci python kód demonštruje základnú definíciu niektoré z týchto operácií.[41]

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

input_shape = (4, 28, 28, 3)
x = tf.random.normal(input_shape)           #TensorShape([4, 28, 28, 3])

# definovanie konvolucie
conv = layers.Conv2D(filters=1,
                     kernel_size=3)(x)      #TensorShape([4, 26, 26, 2])
# definovanie maxpool
maxpool = layers.MaxPooling2D ()(x)         #TensorShape([4, 14, 14, 3])
# definovanie batchnorm
norm = layers.BatchNormalization()(x)       #TensorShape([4, 28, 28, 3])
```

2.4.2. Možnosti implementácie siete

Vytvorenie samotného modelu siete pomocou Tensorflow/Keras je možné tromi spôsobmi:

- Sekvenčný model [42],
- The Functional API [43],
- Model Subclassing [44].

Sekvenčný model je najjednoduchšia metóda pre vytvorenie jednoduchého modelu (jeden vstup, jeden výstup). V tomto modelu sa jednotlivé vrstvy púšťajú za sebou a navzájom si predávajú vstupný obraz (ako funkcia vo funkcií vo funkcií atď.). Nevýhodou tohto modelu je nutnosť jedného vstupu a výstupu pre všetky vrstvy, či nemožnosť vytvoriť viac vetvový model. Nasledujúci python skript ukazuje definovania sekvenčného modelu. [42]

```
model = keras.Sequential(
    [
        layers.Conv2D(filters=1, kernel_size=3),
```

```
        layers.BatchNormalization(),
        layers.ReLU(),
    ]
)
y = model(x)                                    #TensorShape([4, 26, 26, 1])
```

The Functional API je najpopulárnejšia tvorba modelu, keďže obsahuje všetky výhody, ktoré má sekvenčný model, ale k tomu podporuje aj možnosť viacerých vstupov a výstupov z a do vrstiev, či možnosť vytvoriť viac vetvový model atď. Model je viac flexibilný ako predchádzajúca možnosť. Takto definovaný model je v nasledujúcom kóde. [43]

```
def layer_block(x):
    conv = layers.Conv2D(filters=1, kernel_size=3)(x)
    norm = layers.BatchNormalization()(conv)
    relu = layers.ReLU()(norm)

    return [conv, norm, relu]

input_shape = keras.Input(shape=(28, 28, 3))
block1 = layer_block(input_shape)
model_fun = keras.Model(input_shape, block1)
y = model_fun(x)                                    #len(y) --> 3
```

Model Subclassing je najkomplexnejšia metóda tvorby modelu. V ďaka ktorej je možné dosiahnuť úplnú kontrolu nad celou architektúrou. Vytvárať nové vrstvy, modely či definovať proces trénovania. Nevýhodou, však je nutnosť hlbšieho porozumenia fungovania frameworku tensorflow. Takýto model bol ukázaný na nasledujúcom skripte. [44]

```
class ConvNormRelu(keras.layers.Layer):
    def __init__(self):
        super(ConvNormRelu, self).__init__()
        self.conv = Conv2D(filters=1, kernel_size=3)
        self.norm = BatchNormalization()
        self.relu = ReLU()

    def call(self, inputs):
        x0 = self.conv(inputs)
        x1 = self.norm(x0)
        x2 = self.relu(x1)
        return [x0, x1, x2]

block2 = ConvNormRelu()(input_shape)
model_sub = keras.Model(input_shape, block2)
y = model_sub(x)                                    #len(y) --> 3
```

2.4.3. Definovanie datasetu

Tensorflow obsahuje dva hlavné spôsoby ako je možné nedefinovať dataset, ktorý postupne načítava a posiela dáta do modelu počas trénovania.

1. sekvenčný dataset [42],
2. pomocou API tf.data [45].

Sekvenčný dataset je možné definovať napríklad ako triedu. Takýto typ datasetu vždy načíta jednú dávku obr. na základe vyžiadania od modelu (pri trénovaní) a následne aplikuje transformácie (napr.: zmena veľkosti, normalizácia, augmentácia).

API tf.data je novší spôsob vytvorenia datasetu pre trénovanie a validovanie modelu. Na vytvorenie datasetu sa používa nasledujúci definovaný zápis.

```
dataset = tf.data.Dataset.from_tensor_slices(dataset_data)
```

Následne je možné aplikovať transforácie či augmentáciu. Hlavným rozdielom medzi prístupmi je v rýchlosti načítavania nových dát do modelu počas trénovania, kedy API tf.data dosahuje vyššej rýchlosti vďaka optimalizácií priamo od tensorflow na túto úlohu. Táto skutočnosť tak celkovo znižuje čas potrebný na trénovanie. Nevýhodou je nutnosť rozumieť tvorbe takto definovaného datasetu. Kedy sekvenčný dataset je ľahšie pochopiteľný priamo z definície jeho tvorby. [45, 42]

2.5. Možnosti optimalizácie natrénovaného modelu

Optimalizáciou modelu po trénovaní je možné dosiahnuť lepšieho času predikcie modelu (inference time), kedy zrýchlením predikcie sa model stáva lepšie orientovaný na použitie v aplikáciach. Nasledujú podkapitola rozoberá nástroje a ich možnosti optimalizácie natrénovaného modelu.

2.5.1. ONNX Runtime

ONNX runtime je knižnica zameraná na optimalizáciu a zrýchlenia modelov strojového učenia z rôznych frameworkov (Tensorflow, Pytorch, atď). Výhodou je možnosť trénovania modelu v jednom frameworku a následné konvertovanie do ONNX runtime, ktorý podporuje množstvo operačných systémov na ktorý je možné následne model spustiť, ako aj akcelerátory hardwaru. [46]

ONNX ponúka 3 druhy optimalizácie modelu na zlepšenie výsledkov modelu a zrýchlenie predikcie.

Ladenie výkonnosti: Do tejto kategórie patrí výber hardware akcelerátoru ako je CUDA (GPU), CPU, TensorRT (potrebná Nvidia GPU), OpenVino (potrebný Intel procesor) a ďalšie. Voľba najlepšej možnosti akcelerácie hardwaru je obmedzené softwarom a hardwarom na ktorom model je spustený.

Optimalizácia grafu: ONNX vytvorí model ako graf, ktorý tvoria prepojené jednotlivé vrstvy a operácie. Optimalizácia grafu zahŕňa zjednodušenie či spojenie viacerých

operácií do jednej a tým zlepšiť ako rýchlo a presne model predikuje výsledky. Optimalizácia je však opäť závislá na parametroch softwaru a hardwaru na ktorom model vykonáva predikciu.

Kvantizácia modelu: Táto operácia znižuje náročnosť predikcie modelu priamou zmenou presnosti výpočtu, ktoré sú vykonané. Takouto zmenou je možné dosiahnuť nižšie výpočtové nároky a tým zrýchliť model, avšak hrozí, že sa môže mierne zmeniť presnosť s akou bude sieť po kvantizácií predikovať. [46]

2.5.2. Tensorflow lite

Tensorflow lite patrí pod Tensorflow a predstavuje optimalizáciu modelu prevedením ho do formátu .tflite, ktorý bol vytvorený pre zariadenia s obmedzeným výkonom (android, Linux, atď.).

Prvá optimalizácia veľkosti modelu prebieha už iba prevedením siete do formátu .tflite. Ďalšie, ako je použitie hardware akcelerátoru a kvantizácia modelu sú rovnaké ako pre ONNX.

Nepriamou nevýhodou tensorflowlite zostáva, že aj keď je možne prevedený model spustiť napr. na operačnom systéme Windows s Intel procesorom, model môže dosahovať horšieho času predikcie, keďže nie je na tento symstem postavený. [47]

2.6. Anotačný nástroj

Anotačný nástroj slúži v prípade segmentácie, na vytvorenie pravdivostného obrazu, ktorý sa využíva pri trénovaní na optimalizáciu parametrov a neskôr pri validácia testovaní na evaluáciu. Voľno dostupných nástrojov, ktoré podporujú úlohu segmentácie je na výber viacero. Rozhodujúcimi faktormi sú možnosti, ktoré program podporuje.

Anotačný nástroj **CVAT**[8] vyvinutý od firmy intel poskytuje veľké množstvo funkcionalít, ktoré teoreticky dokážu zrýchliť proces vytvárania masiek. Hlavne v prípade ak je potrebné anotovať väčšie množstvo dát. Nástroj podporuje široký vyber vstupných ako aj výstupných formátov. Výhodou je taktiež možnosť používať už pripravený program v internetovom prehliadači bez nutnosti inštalácie. [48]

Label Studio ponúka o niečo menej možností použitia ako predchádzajúci nástroj CVAT. Tieto nedostatky sú ako v možnosti používať program bez inštalácie, tak v počte podporovaných formátov. Výhodu však predstavuje jednoduchá inštalácia na vlastné zariadenie, ktorá taktiež podporuje čiastočné upravovanie grafického rozhrania. [49]

Nejjednoduší z trojice možností je **MakeSense**, tento voľne dostupný program funguje priamo v okne prehliadača. Avšak možnosti jeho použitia na anotovanie väčšieho datasetu dát sú znateľne obmedzené, keďže neobsahuje žiadne pokročilejšie nastavenia ako napr. CVAT či Label Studio. Nástroj je teda skôr určený na menej náročné úlohy, kedy vzhľadom k svojej jednoduchosti funguje v celom procese rýchlejšie (bez nutnosti prípravy či nahrávania dát ako pre CVAT/ Label Studio). [50]

[8] computer vision anotation tool

3. Formulácia problému

V kapitole 2 (Rešerš) boli spracované všetky potrebné teoretické základy a informácie potrené k tomu aby bola téma diplomovej práce spracovaná v plnom rozsahu. Nasledujúca kapitola obsahuje podrobnejšie rozpísané problémy a opis postupu riešenia.

3.1. Definovanie problému

Najväčším problémom sémantickej segmentácie z kamery mobilného robota je menej výkonný hardvér, na ktorom je potrebné vykonať predikciu jedného snímku z kamery v reálnom čase. A keďže termín v reálnom čase nie je úplne jasne definovaný a skôr záleží na type úlohy, bude v tejto práci pojem odkazovať na hodnotu 30 FPS (frames per second-snímky za sekundu).

Problém bol v práci rozdelený na tri hlavné bloky. Implementovanie vhodnej metódy na sémantickú segmentáciu s ohľadom na menej výkonný hardvér, trénovanie a overenie na reálnych dátach a otestovanie modelu na simulovanej aplikácií, ktorá overí rýchlosť a zaťaženie siete na zvolenom hardvéry.

3.1.1. Vyber metódy na implementovanie

V podkap.2.3.3 boli spomenuté možnosti architektúry siete pre segmentáciu. Zo spomenutých architektúr bola na implementáciu vybraná sieť DDRNet 23 slim [38] (obr. 2.18), ktorá dosiahla 77.4 % mIoU a 102 FPS na jednej GPU Nvidia 2080Ti v pôvodnom práci [38]. A pri počte 5.7 M parametrov siete sa jedná o relatívne malú sieť. K výberu tak prispela veľkosť siete, ako aj dosiahnutie hodnoty FPS, ktoré naznačovali, že model bude schopný dosiahnuť dostatočne rýchlu predikciu v reálnom čase aj na menej výkonnej grafickej karte.

Posledný dôvod výberu danej architektúry bola možnosť využitia metódy transfer learning, keďže autori poskytli natrénované parametre pre rovnakú architektúru použitú na úlohu klasifikácie obrazu.

3.1.2. Overenie na reálnych dátach

Tento cieľ práce bol kvôli väčšej prehľadnosti rozdelený na niekoľko nasledujúcich úloh.

Dataset: Vytvorenie datasetu bolo podmienené požiadavkami na dostatočné množstvo dát, pre dosiahnutie lepšej generalizácie modelu počas trénovania.

Trénovanie: Pre proces učenia siete, boli zvolené štyri samostatné modely vytvorené z rovnakej architektúry DDRNet 23 slim, ktoré budú natrénované. Ich rozdielom je iba veľkosť vstupných dát. Trénovanie bolo vykonané z ohľadom na najmenšiu chybovú funkciu, u ktorej nastáva rýchla reakcia na nechcené zmeny počas trénovania a tým vie zabrániť nežiadúcim výsledkom.

Overenie na dátach: Po skontrolovaní trénovania a vyhodnotení výsledkov (priebehy učenia), predstavovalo testovania poslednú fázu tohto bloku. Kedy bol použitý testovací dataset na evaluáciu natrénovaných sieti a vytvorené celkové zhrnutie výsledkov.

3.1.3. Test na obmedzenom výkone

Posledná časť prace predstavoval experiment v ktorom bolo definované testovanie na vybranom hardvéry, za účelom získania závislosť veľkosti vstupných dát na rýchlosti predikcie. Pre splnenie tejto úlohy bol každý z modelov prevedený na formát ONNX Runtime, ktorý predstavuje optimalizovanejšiu verziu siete. A pre simuláciu aplikácie modelu bol vytvorený evaluačný nastroj, ktorý meria rýchlosť segmentácie jedného snímku videa.

4. Dataset

Nasledujúca kapitola popisuje voľbu dát a vytvorenie datasetu určeného na trénovanie, validáciu a testovanie konvolučnej siete. Prvá časť za zaoberala výberom a definovaním vstupných dát a objektov na segmentovanie, následne bola vykonaná dátová analýza, ktorej cieľom bolo rozobrať štruktúru anotovaných snímkov a zistiť rozloženie tried. Výstup z kapitoly tvorí definovaný dataset pomocou frameworku tensorflow, ktorý je použiteľný počas procesu učenia modelu.

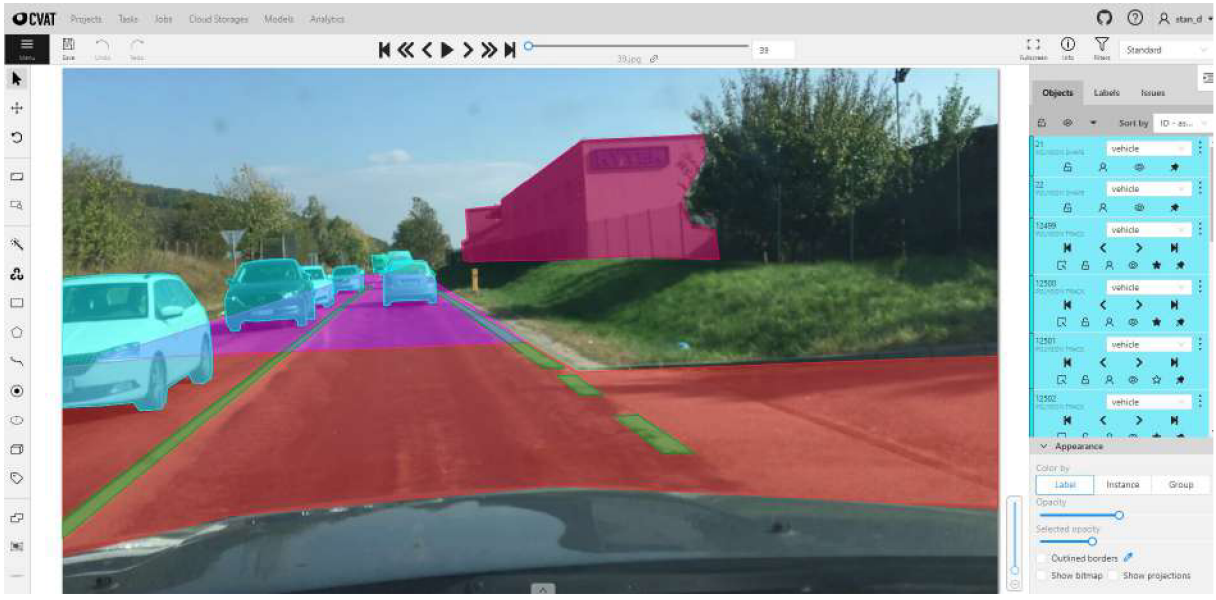
4.1. Definovanie zvolených dát

Ako zdroj pre dataset bolo zvolené video, ktoré zachytilo prejazd auta Juhomoravskou oblasťou Českej republiky. Tieto dáta boli vybraný na základe predpokladu, že obsahuje dostatočné množstvo často sa meniacich tried na segmentovanie. Tab 4.1 obsahuje bližšie informácie o zdrojovom videu. Zo záznamu videa boli následne extrahované jednotlivé snímky, ktoré vytvorili originálne dáta použité na anotovanie.

Rozlíšenie	1929, 1080
Dĺžka videa	194 s
Frekvencia snímkov	30
Celkový počet snímkov	5824

Tabuľka 4.1: Informácie o zdrojovom videu

Triedy boli zvolené ako reprezentácia najdôležitejších objektov na videu tak, aby sa po segmentácií zachovali všetky podstatné črty zdrojových dáta a zároveň poskytovali dodatočné informácie (poloha auto/ človeka, výskyt križovatky), ktoré je možné použiť pre ďalšie správanie. Popis vybraných tried, ich farbu (pre RGB formát) a číselnú reprezentáciu (číslo, ktoré tvorilo výstup zo segmentácie) boli zhrnuté do tab. 4.2.



Obr. 4.1: Ukážka nástroja cvat

Trieda	pozadie	budova	križovatka	motorka	človek	cesta
Číslo triedy	0	1	2	3	4	5
Farba RGB	(0, 0, 0)	(163, 0, 89)	(255, 74, 70)	(245, 147, 49)	(122, 73, 0)	(255, 52, 255)

Trieda	zvodidlá	cestny pätnik	cestný pruh	semafor	značka	vozidlo
Číslo triedy	6	7	8	9	10	11
Farba RGB	(250, 51, 2)	(183, 151, 98)	(0, 137, 65)	(0, 0, 166)	(99, 255, 172)	(28, 230, 255)

Tabuľka 4.2: Zvolené triedy

Pre anotovanie bol zvolený voľno dostupný nástroj CVAT[1], ktorý poskytuje najlepšie možnosti pre zrýchlenie celého procesu vytvárania masky vrámci svojej kategórie. Obr. 4.1 zobrazuje ukážku z používania tohto nástroja v procese anotovania nazhromaždených dát.

4.2. Dátová analýza

Porozumenie obsahu informácií, ktoré tvoria dáta (po vykonaní anotovania), predstavuje dôležitú súčasť prípravy datasetu na trénovanie. Hlavnou úlohou analýzy bolo zistiť a vizualizovať konkréte zloženie nazbieraných snímkov s vytvorenými maskami a odhaliť prípadne problémy, ktoré môžu neskôr kvôli datasetu nastať vo fáze trénovania.

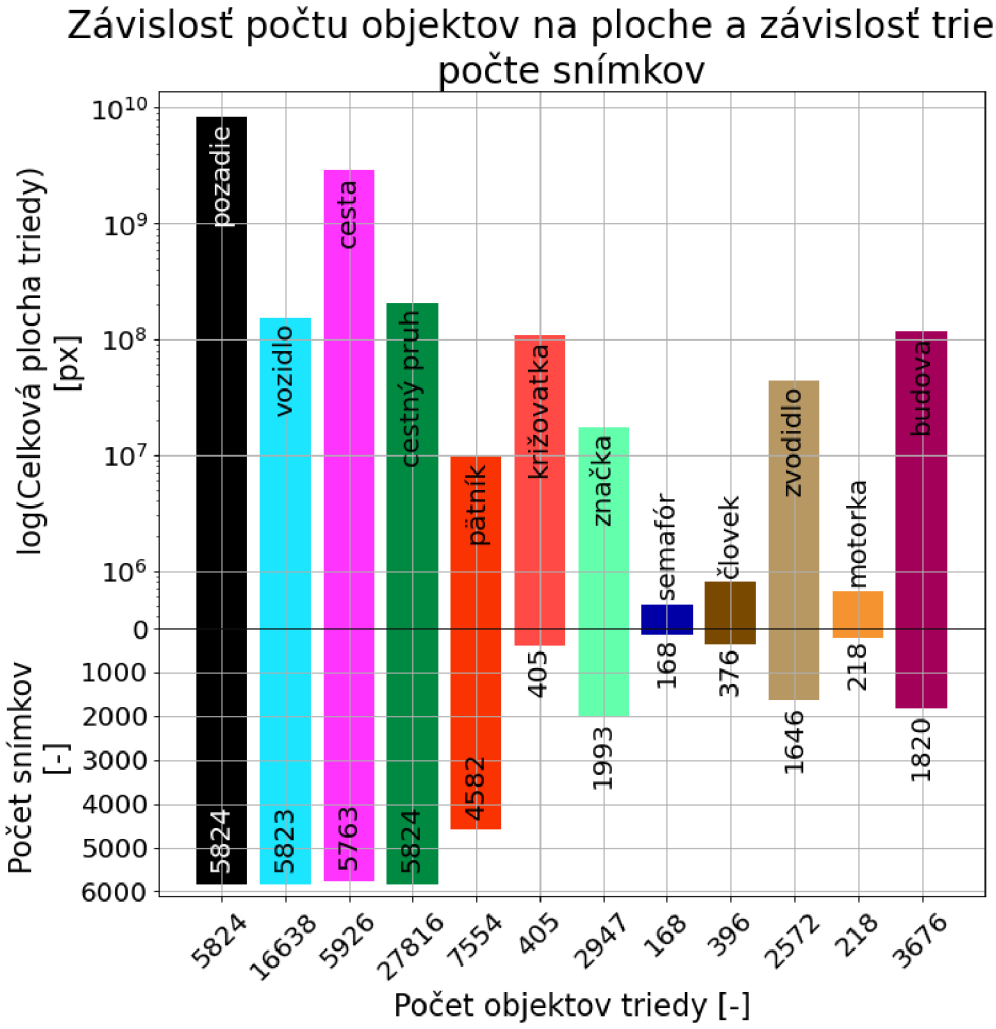
Obr.4.2 zobrazuje komplexnú analýzu nazhromaždených dát. Na ose x ma graf vynesenú informáciu o počte objektov pre jednotlivé triedy, kedy objekt je definovaný ako jeden uzavretý polygón na obraze. Osa y pre horný graf zobrazuje celkovú plochu, ktorú daná trieda zaberá z datasetu v logaritmickej mierke.

Následne, graf pod nulou má na ose y vynesenú informáciu o počte výskytu tried na všetkých snímkoch, kedy viac objekt tej istej triedy na rovnakom obraze je chápaný iba ako jeden prírastok, takže maximálne bolo možné dosiahnuť hodnotu 5824 snímkov (cele video).

Z dátovej analýzy a zo skutočnosti, že kvôli veľkému rozdielom v počte celkovej plochy, ktorú triedy zaberajú bolo nutné pre hornú polovicu osi y použiť logaritmickú mierku vychádza, že dataset obsahuje nerovnomerne rozložené triedy. Ako je možné vidieť (graf. 4.2) "pozadie"dominuje medzi zábranou plochou na obraze, na druhú stranu "semafor", "človek"či "motorka"tvoria najmenej zastúpené triedy ako vo veľkosti plochy ktorú zaberajú, tak a aj v počte snímkov na ktorých sa nachádzajú. Ďalšou zaujímavosťou sú triedy "križovatkaä "budova", ktoré síce relatívne zaberajú väčšiu plochu, ale z celého datasetu sa nachádzajú iba na minimálnom množstve snímkov.

Z graf bolo taktiež možné nájsť informáciu o tom, ktorá trieda je na obraze obsiahnutá vždy iba jeden krát pre obraz (počet celkových snímkov- dolná osa y sa zhoduje s počtom objektov- osa x).

[1]computer vision annotation tooll



Obr. 4.2: Dátová analýza

Nevyrovnaný dataset

Problémom nerovnomerne rozloženého datasetu označuje také data, v ktorých niekoľko dominantných tried zaberá väčšinovú plochu z obrazu (celého datasetu, obr. 4.2). Toto predstavuje závažný problém pri trénovaní konvolučných sieti (s učiteľom), keďže model dostáva dostatočné množstvo ukážok väčšinových tried a rastie mu schopnosť ich ľahšie rozpoznať, zatiaľ čo menej reprezentované triedy mu môžu unikať (zlá predikcia). Na tento problém ukazuje aj príklad z podk. 2.3.2.

Na vyriešenie tohto problému existujú viaceré prístupy. Základným je nazbierať dodatočné vzorky objektov, ktorých je príliš malé množstvo. Alebo zredukovať počet dominantných tried či vytvoriť modifikované verzie existujúcich prvkov (menej reprezentovaného) tak, aby sa triedy dorovnali.

Pokročilejšími možnosťami je do procesu trénovania vložiť chybovú funkcia, ktorá dokáže túto nevyrovnanosť čiastočne kompenzovať či zvoliť váhy, ktorý pridávajú vyšší dôraz na menšie triedy počas učenia. Keďže spomenuté základné možnosti sa ťažko aplikujú na obrázové dáta, bol tento problém riešený zvolením funkcie focal loss, ktorá sa používa na nevyrovnané datasety.

4.2.1. Segmentačná maska

Obr. 4.3 obsahuje ukážku vytvorenej masky z pôvodného snímku (obr. 4.3a). Maska bol vo formáte RGB, čo znamená, že každý pixel je reprezentovaný tromi hodnotami. Keďže však model očakáva masku ako maticu, kde jeden pixel odpovedá jednej triede, nebolo možné takto vytvorenú pôvodnú masku použiť pre trénovanie.

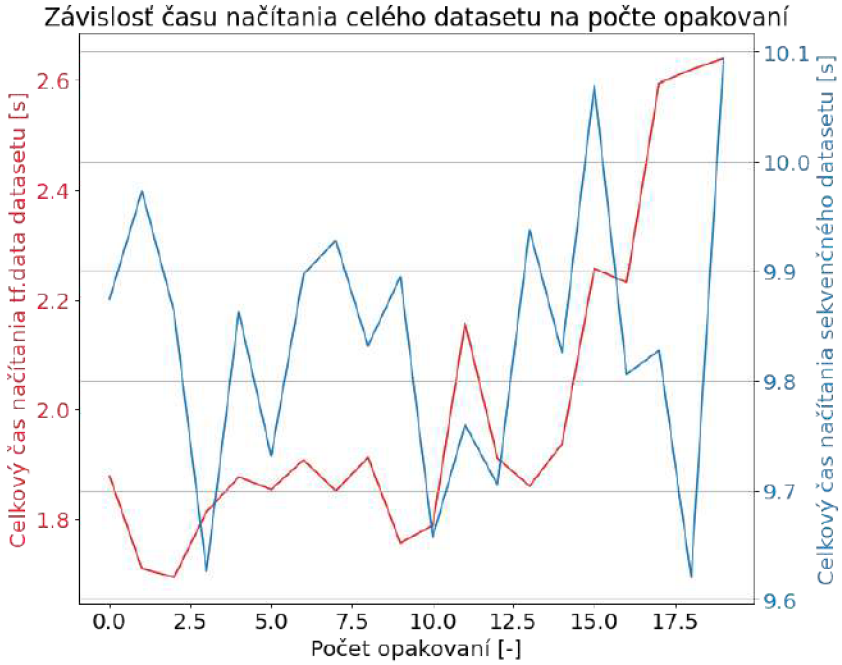
(a) Pôvodný obraz (b) Maska v RGB (c) Upravená maska

Obr. 4.3: Ukážka segmentačnej masky

Maska bola preto premapovaná do podoby ako zobrazuje obr. 4.3c. Kde bol jeden pixel reprezentovaný iba jednou hodnotou (tieto hodnoty tried odpovedajú číslu tiedy v tab. 4.2). Upravená maska, pôsobí že obsahuje iba dve farby (biela a čierna), čo bolo však spôsobené tým, že pixely sú reprezentované príliš blízkymi hodnotami (0 - 11).

4.3. Tensorflow dataset

V podkap. 2.4.3 boli rozobrané možnosti vytvorenia datasetu pre trénovanie tensorflow modelu. Z dvoch predstavených spôsobov, bola pre model vybraný prístup, ktorý využíva API tf.data. Rozhodujúcou výhodou použitia bola rýchlosť s akou tf.data operuje oproti sekvenčnej možnosti a tým dokáže urýchliť celý proces trénovania ako ukazuje obr. 4.4.



Obr. 4.4: Porovnanie rýchlosti datasetu

Obr. 4.4 vznikol ako výsledok jednoduchého testu, ktorý pozostával vo vytvorení datasetov dvoma spomenutými metódami. Každý z nich obsahoval 100 snímkov a funkcie na čítanie a úpravu boli v obidvoch spôsoboch identické. Test bežal 2 epochy a načítal všetky dáta z datasetu a bol opakovaný 20 krát pre každú metódu.

Priemerný čas načítania celého datasete pre 2 epochy bol pre sekvenčný spôsob 9.8 sekúnd a pre tf.data 2.0 sekúndy. Tf.data navyše ponúka ďalšie optimalizácie, na možné zvýšenie rýchlosti.

4.3.1. Augmentácia dát

Ako bolo ukázané v tab.4.1 dataset bol vytvorený celkovo z 5824 snímkov. Takto relatívne malému datasetu s viacerými triedami na segmentovanie, ktoré nie sú rovnomerne rozložené by však mohlo počas trénovania hroziť, že u modelu nastane preučenie.

Preučenie označuje situáciu, keď model veľmi dobre predikuje snímky z datasetu na ktorých bol trénovaný, ale úspešnosť predikcie iných dát nedosahuje rovnakej kvality. Navyše pri použití menšieho datasetu, pravdepodobnosť výskytu tohto probléme ešte viac rastie, keďže málo dát znamená, že je nutné posielať rovnaké obrazy do modelu viackrát.

Možná stratégia ako tento problém minimalizovať, či sa mu čiastočne vyhnúť je použiť augmentáciu dát. To znamená mierne upravovanie vstupných dát náhodnými operáciami, takže aj keď dataset neobsahuje dostatočne množstvo dát, vstupné dáta nikdy nie sú úplne rovnaké, aj keď pochádzajú z rovnakého zdroja.

Augmentácia dat, môže byť rozdelená na offline a online. Kedy **offline** prebieha vytvorením pozmenených dát a následne ich uložením. To znamená, že pri vytváraní datasetu, sú už tieto dáta iba načítané rovnakým spôsobom ako zdrojové snímky. Výhodou/ nevýhodou u takejto augmentácie je nutnosť udržiavať súbory stále uložené. **Online** forma prebieha v reálnom čase na zadanom datasete, čítaním iba pôvodného zdroja dát a vykonanie modifikácie v rámci preprocesovania obrazu, keď sa vytvorená dávka posiela do modelu na trénovanie. U tejto možnosti nie je nutné súbory mať uložené, avšak keďže augmentácia je väčšinou séria náhodných úprav, ten istý trénovaný model (nezávisle na sebe) dostane vždy iné dáta, narozdiel od offline, kedy sa vstupne dáta pozmenili jedenkrát a už sa iba používajú. Rovnako ako u offline metódy, tieto vlastnosti môžu predstavovať výhodu aj nevýhodu.

Pre dataset tejto práci bola vybraná možnosť online augmentácie a s ohľadom na obsah dát nasledujúce úpravy:

1. náhodné horizontálne prevrátenie obrazu,
2. náhodné priblíženie v rozsahu ±5 až 30 %[2]
3. náhodne orezanie obrazu na zadanú veľkosť,
4. náhodná zmena kontrastu obrazu.

Tensorflow podporuje dva hlavné spôsoby definovania náhodnej modifikácie na dataset. Prvým je použitie vrstiev, ktoré obsahujú operáciu a druhým použitie funkcií. Keďže využitie vrstiev je z praktického hľadiska jednoduchšie, bola zvolená táto možnosť. Možný spôsob vytvorenia augmentácie je v nasledujúcom ukážkovom skripte.

[2]nutné použitie interpoláciu najbližších susedov (nearest neighbor)

```
def augment_using_layers(image, mask):

    def aug_sek_mod(height, width):
        # vytvorenie a definovanie vrstie augmentácie
        aug_layers = ...
        aug_model = tf.keras.Sequential(aug_layers)
        return aug_model

    # vytvorenie augmentačného modelu
    aug_model = aug_sek_mod()
    # výstupom sekvenčného modelu je upravený vstup
    return aug_model([image, mask])
```

Funkcia pracuje na princípe vytvorenia sekvenčného modelu, ktorý má definované vrstvy na úpravu obrazu. Keďže sa jedná o takto postavený model, jednotlivé operácie si vstup navzájom posúvajú.

Výhodou operácií (normalizácia, zmena mierky) ako vrstiev je v možnosti priameho zapojenie na začiatok architektúry siete (pred vstupom), čím je možné zaistiť, že akýkoľvek vstupné dáta boli rovnako spracované.

Najvýraznejšou výhodou tohto celého procesu je získanie prídavných dát pre dataset, bez nutnosti reálneho zväčšovania a tým obmedziť vzniknutý problém možného preučenia. Výsledný počet dát mohol byť vypočítaný ako $5824(pvodnvekso) \cdot e$, kedy e predstavuje počet epoch trénovania. Keďže náhodné modifikácie prebiehajú počas trénovania na každej vzorke a za epochu je snímka poslaná do siete iba jeden krát.

Základný dataset pomocou API tf.data bol definovaný nasledujúcim spôsobom (bez použitia optimalizácií):

```
# vytvorenie datasetu ako list ciest k obr.
dataset_train = tf.data.Dataset.from_tensor_slices(path_to_data)
# vytvorenie datasetu na trěnovanie
dataset_train = (dataset_train
                 # zamiešanie datasetu
                 .shuffle(shuffle_number)
                 # mapovanie datasetu ciest --> dataset obr. pomocou funkcie
                 .map(load_dataset)
                 # voľba veľkosti dávky
                 .batch(batch_size)
                 # augmentácia dát
                 .map(lambda x, y: augment_using_layers(x, y))
             )
```

Overenie definovaného datasetu

Krok pozostáva v overení celkovej funkčnosti datasetu, či tak ako bol navrhnutý (načítanie dát, normalizácia, augmentácia) aj skutočne operuje počas trénovania. Preto bol vytvorený jednoduchý dataset, ktorý obsahoval iba 2 snímky, veľkosť dávky bola 2 a možnosti upravovania (augmentácie) boli rovnako zadané ako v podkap. 4.3.1. Simulácia prebiehala 2 epochy a výsledok bol spracovaný do obr. 4.5.

(a) Vstup do modelu pre 1. epochu (b) Vstup do modelu pre 2. epochu

Obr. 4.5: Vstup do modelu po dvoch epochách

Každá epocha dostala 2 vstupy (1 dávka) v náhodnom poradí a s náhodnou augmentáciou. Výsledok tak odpovedá predpokladu fungovania.

5. Implementácia

Pôvodné vytvorená sieť DDRNet bola vytvorené a natrénované vo frameworku pytorch[1], ktorý nemá identický spôsob tvorby a ukladania modelu. Nasledujúca kapitola preto popisuje spôsob implementácie vybranej architektúry DDRNet pomocou tensorflow, ako aj použitie predtrénovaných parametrov pre možnosť využitia metódy transfer learning. Na záver kapitoly bol vykonaný test na porovnanie výstupov z vrstiev týchto dvoch rozdielne vytvorených modelov (tensorfow/ pytorch). Účelom bolo overiť správnosť implementácie.

5.1. Implementácia siete DDRNet

Popis architektúry na ktorej je postavená sieť DDRNet bol opísaný v podkap. 2.18. Celkovo existujú 4 rôzne prevedenia, ktoré sú rozdielne vo veľkosti siete a tým aj v počte parametrov, ktoré je potrebné získať trénovaním.

Konkrétne vybraná sieť DDRNet 23 slim je zo všetkých možností najmenšia s 5.7 miliónom parametrov, čo predstavuje relatívne malý model. Výhodou menšej architektúry sú výrazne nižšie nároky na výpočet predikcie a tým spĺňa predpoklad na implementáciu na obmedzený hardvér mobilného robota.

Sieť bola z hľadiská implementácie rozdelená podľa obr. 2.18 na niekoľko blokov vrstiev, ktoré boli na seba v kóde navzájom naviazané.

1. Reziduálny blok,
2. bilaterálne spojenie,
3. DAPPM.

Z možností implementácie siete v tensorflow (podkap. 2.4.2) bola zvolená možnosť Functional API, v ktorej je možné vytvoriť kompletnú architektúru bez žiadnych obmedzení. Celkový model tak bol vytvorený s nasledujúcou štruktúrou.

```
# bloky modelu a model boli definované ako funkcie
def residual_block():
    ...
    return output

def DAPPM():
    ...
    return output
```

```
    def bilate_fusion():
        ...
        return output

    def DDRNet_model():
        x = residual_block()
        x2 = bilate_fusion()
        x3 = DAPPM()
        ...
        return output
```

Autori pôvodnej práce taktiež poskytli už optimalizované váhy pre model DDRNet 23 slim určený na klasifikáciu obrazu (sieť neobsahuje DAPPM a niektoré posledné vrstvy), ktorý

[1] framework zameraný na riešenie a aplikáciu strojového učenia, rovnako ako Tensorflow

bol trénovaný na štyroch grafických kartách RTX 2080Ti. Tieto parametre boli dostupné vo forme voľne sťiahnuteľného súboru z webovej stránky github.

Keďže však formát súboru bol určený pre model, ktorý operuje pod pytorch, nebolo možné ho priamo vložiť do vytvorenej siete v tensorflow. Aj keď existujú nástroje na konvertovanie celých architektúr medzi frameworkom, ukázali sa mať obmedzenia, kvôli ktorým ich nebolo možné použiť.

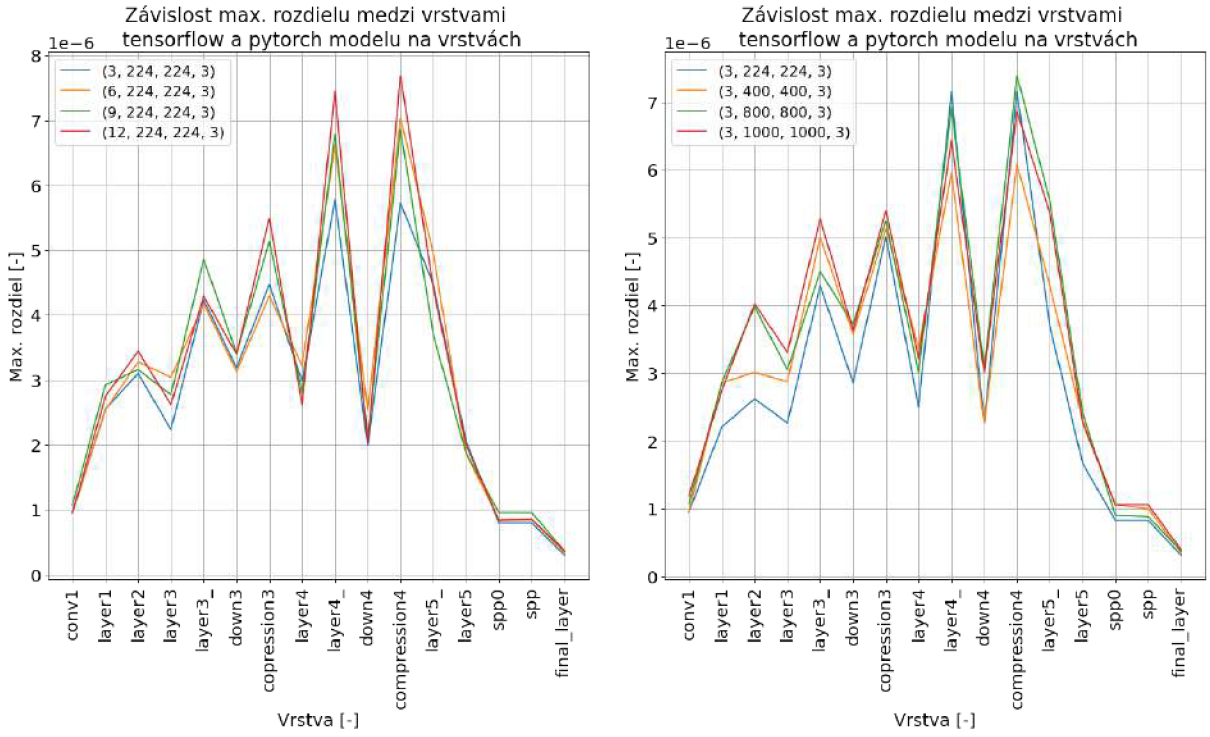
Preto bolo nutné upraviť vytvorenú štruktúru tensorflow modelu tak, aby jednotlivé názvy vrstiev boli identické s pytorch implementáciou. Týmto spôsobom bolo možné načítané váhy v pytorch formáte (dátový typ dict v jazyku python) namapovať na tensorflow model s rovnako označenými vrstvami.

Overenie vytvoreného modelu

Táto skúška slúžila na overenie správnosti vytvoreného modelu s vloženými parametrami. Celý proces pozostával z automatizovaného testu, ktorý najprv vytvoril sieť, zadal pripravené hodnoty váh a následne porovnával výstupy medzi tensorflow a originálnym modelom.

Kvôli porovnávaniu bolo nutné upraviť obidve architektúry tak, aby po jednom vložení vygenerovaných dát bolo možné na výstupe pristúpiť k výsledku každého bloku vrstiev.

Konečný experiment bol vykonaný postupným generovaním náhodných vstupov s meniacou sa veľkosťou dávky a rozmermi matice (2 samostatne testy). A výstupom porovnania je získanie maximálnej hodnoty rozdielov zhodných vrstiev.



(a) Zmena veľkosti dávky (batch size)

(b) Zmena veľkosti vstupného tensoru

Obr. 5.1: Rozdiel medzi vrstvami tensorflow a pytorch modelu

Pre lepšie porovnanie funkčnosti, bol popísaný test opakovaný štyri krát pre rôzne dávky a veľkosti vstupu (celkovo osem rôznych vstupov). Tieto zvolené hodnoty ako aj výsledky boli zobrazené na grafe 5.1.

Graf 5.1 zobrazuje získanú závislosť maximálnej hodnoty rozdielu (osa y) medzi vrstvami (osa x) modelu pytorch a tensorflow, ktoré zdieľajú identické pomenovania. Prvý graf 5.1a obsahu výsledky pri zmene veľkosti dávky (batch) a druhý (graf 5.1b) pri zmene veľkosti rozmerov vstupu (pre obraz šírka x výška). Najvyššia dosiahnutá rozdielna hodnota z testovania vyšla 7.6^{-6} pre názov vrstvy "compression4", čo predstavuje iba minimálny rozdiel vo výpočtoch. Na základe ukázaných výsledkoch bolo možné siete prehlásiť za identické a proces namapovania parametrov za úspešný.

6. Trénovanie siete

Pre úspešné začatie trénovania bolo potrebe vytvoriť dataset a implementovať vybranú architektúru. Tieto úlohy boli detalnejšie popísané v kap. 4 a 5. Nasledujúci text sa zameriava na využitie spomenutých riešení práce v procese trénovania konvolučnej siete. Ktorý zahŕňal popis voľby vhodných hyperparametrov a vyhodnotenie dosiahnutých výsledkov.

6.1. Voľba hyperparametrov

Hyperparametre predstavujú dôležitú súčasťou konvolučných (neurónových) sieti a ich voľba zásadne ovplyvňuje ako bude trénovanie prebiehať. Ich význam a konkrétne kategórie parametrov, ktoré do tohto označenia patria boli bližšie popísané v podkap. 2.2.3. Keďže neexistujú žiadne priamočiare či v zásade jednoduché pravidlá pre výber najlepších hodnôt, tento proces bol vykonaný postupným testovaním a ladením. Väčšina zvolených parametrov bola teda získaná ako výsledok testovania. Okrem niekoľkých, ktoré boli zobrané ako zaužívaný štandard.

Dataset, ktorý bol vytvorený v predchádzajúcej kap., bol pre účely trénovania rozdeliť na 3 diely. Trénovacie dáta predstavovali jediné správne vzorky z ktorých sa model učil. Validačné dáta, ktoré slúžili na evaluáciu siete počas trénovania a na nájdenie niektorých hyperparametrov experimentálnym postupom. A na testovacie dáta, ktoré boli použité iba jeden krát a to až potom ako bola sieť natrénovaná. Výsledok predikcie na tomto posledom datasete slúži na záverečné ohodnotenie vytvoreného modelu

Pomer rozdelenia (60/20/20) % bol zvolený na základe toho, aby všetky tri diely obsahovali primerané množstvo jednotlivých tried v takom počte, aby sa pri evaluácií minimalizoval výskyt skreslených výsledok kvôli nerovnomernému rozloženiu dát.

	Trénovací dataset	**Validačný dataset**	**Testovací dataset**	
Trieda	**Počet objektov** [-] [%]	**Počet objektov** [-] [%]	**Počet objektov** [-] [%]	**Celkovo** [-]
vozidlo	9899 (59.5)	3348 (20.12)	3391 (20.38)	16638
cesta	3549 (59.89)	1192 (20.11)	1185 (20.0)	5926
cestný pruh	16762 (60.26)	5567 (20.01)	5487 (19.73)	27816
pätník	4510 (59.7)	1542 (20.41)	1502 (19.88)	7554
križovatka	230 (56.79)	88 (21.73)	87 (21.48)	405
značka	1755 (59.55)	639 (21.68)	553 (18.76)	2947
semafór	95 (56.55)	35 (20.83)	38 (22.62)	168
človek	241 (60.86)	82 (20.71)	73 (18.43)	396
zvodidlo	1546 (60.11)	524 (20.37)	502 (19.52)	2572
motorka	135 (61.93)	46 (21.1)	37 (16.97)	218
budova	2203 (59.93)	773 (21.03)	700 (19.04)	3676
Priemer	59.55 %	20.74 %	19.71 %	

Tabuľka 6.1: Rozloženie tried v datasetoch

Pre overenie požadovaného rozloženia boli vytvorené časti pôvodného datasetu vyčíslené v počte objektov ktoré obsahujú. V tab. 6.1 bolo možné vidieť splnenie zadaného pomeru.

Tab. 6.2 obsahuje niekoľko najpodstatnejších hyperparametrov, ktoré boli použité. Pre samotný proces učenia bol zvolený optimalizačný algoritmus Adam. Tento algoritmus ponúka adaptívnu zmenu kroku učenia pre rôzne parametre, ktoré optimalizuje

Kroku učenia (maximálne hranica kroku učenie pre parametre u algoritmu Adam) bola priradená počiatočná hodnota 0.01. Taktiež bolo pre krok vytvorené pravidlo podľa ktorého sa tento parameter každých 5 epoch exponenciálne zmenšoval.

Optimalizačný algoritmus	Adam
Krok učenia	0.01
Chybová funkcia	Focal loss
Metrika	mIoU
	Sparse cate. accuracy
Veľkosť dávky	15
Základný počet epoch	150

Inicializácia váh	he normal
GPU Trénovanie	Nvidia RTX 3060
Rozlíšenie	1. 336x600
	2. 400x720
	3. 720x1280
	4. 1080x1920

Tabuľka 6.2: Zvolené hyperparametre

Keďže vytvorený dataset obsahuje nerovnomerne rozložené dáta, bola zvolená chybová funkcia focal loss, ktorej použitie je odporúčané v prípade takéhoto problému (kap. 2). A pre meranie úspešnosti klasifikácie správnych tried bola zvolená mIoU, ktorá je všeobecne zaužívaná pre sémantickú segmentáciu a slúži tak ako dobrý spôsob akým porovnať výsledky medzi rôznymi prácami.

Posledný parameter z tab. 6.2 predstavujú vybrané rozlíšenia modelov. Následkom čoho boli celkovo vytvorené 4 siete DDRNet 23 slim s rôznymi rozlíšeniami (veľkosťami vstupov). Vďaka čomu bolo neskôr v práci možné vykonať experiment na získanie údajov o tom ako veľkosť vstupných dát závisí na rýchlosti predikcie.

V nasledujúcom texte práce bolo zavedené označenie pre modely z rôznym rozlíšením ako DDRNet23sX, kde X predstavuje číslo od 1 do 4, podľa zoradenia rozlíšení v tab. 6.2.

6.2. Definovanie trénovania

Pred samotným spustením učenia konvolučnej siete boli vytvorené dva tensorflow objekty triedy callback. Tieto objekty slúžia na vykonávanie zadanej funkcie počas optimalizačného procesu (napr. po každej epoche, atď). Prvý callback, slúžil na exponenciálnu zmenu kroku trénovania (tab. 6.2) a druhý sledoval **najnižšiu hodnotu validačnej chybovej funkcie** a uložil parametre modelu akonáhle bola nájdená nová nižšia chyba. Pomocou tohto objektu bolo zaistené, že ak by trénovanie neočakávane skončilo, najlepšia nájdená verzia modelu do toho okamžiku by bola stále uložená.

Všetky zvolené konvolučné siete (DDRNet23s[1-4]) boli natrénované na grafickej karte Nvidia RTX 3060. Využitie grafickej karty na tento proces vyžadoval mať stiahnutý frame-

work *tensorflow-gpu* a nainštalovanú a správne nastavenú knižnicu CUDA (pre použitie Nvidia GPU), ktorá slúži na paralelné výpočty. Proces stiahnutia a inštalácie týchto nástrojov nebolo v tejto práci rozoberané. Alternatívnou možnosťou pre trénovanie bolo použitie procesoru namiesto grafickej karty. Avšak rozdiel v čase, ktorý by bol potrebný na celý proces optimalizácie by bol nezanedbateľný.

Akonáhle bol pripravený dataset a nastavené všetky potrebné parametre, bolo možné definovať siete a s využitím metódy transfer learning začať celé trénovanie. Ukážkový skrip obsahuje definovanie modelu, zmrazenie vrstiev (transfer learning) a vypísanie informácií o zložení siete.

```
# vytvorenie modelu s premapovanými natrénovanými parametrami
ddrnet_model, pred_model = DDRNet(zvoleny_rozmer_vstupu, pocet_tried)
# definovaný dataset s augmentáciou
dataset_train = tf.data.Dataset(...)
#zmrazenie vrstiev, ktoré boli premapované
pred_model.trainable = False
# vypísanie vrstiev modelu a parametrov
ddrnet_model.summary()          #    Total params: 5,715,372
                                #    Trainable params: 1,148,300
                                #    Non-trainable params: 4,567,072
```

Ako ukázali výsledky, sieť dokopy obsahuje 5.7 milióna parametrov. Z toho však v proces učenia tejto konvolučnej siete bolo optimalizovaných iba 1.2 milióna. Takéto zníženie náročnosti na výpočet bolo spôsobené práve zmrazením premapovaných vrstiev. Posledný krok kódu už predstavoval iba kompiláciu modelu (kompilovaním sa zmrazí to akým spôsobom sieť operuje) a spustenie celého procesu.

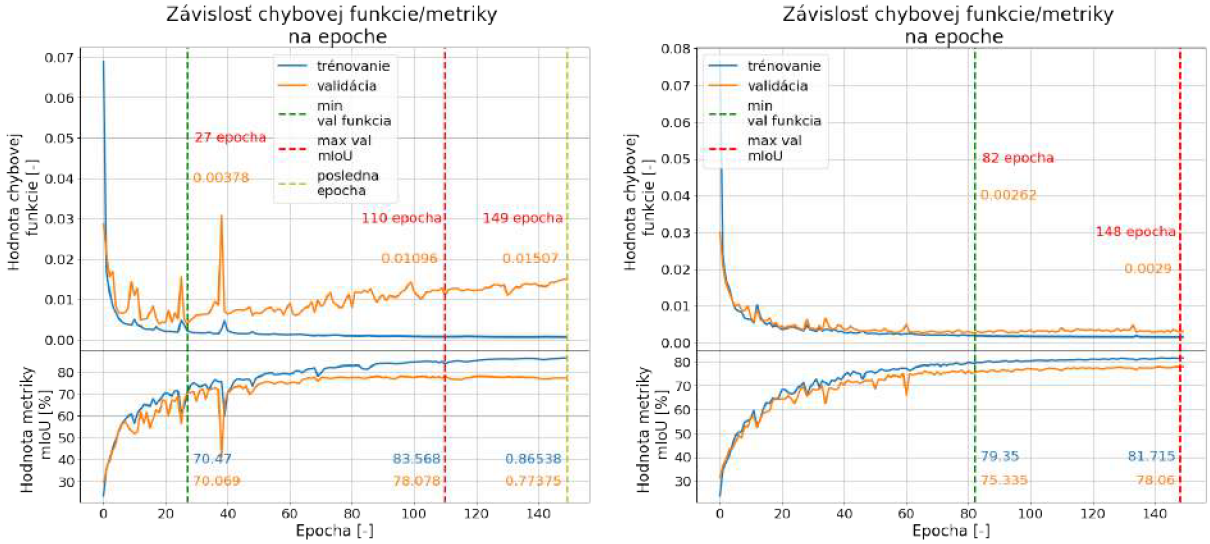
```
# definovaný callback pre ukladanie siete počas učenia
najlepsi_model = tf.keras.callbacks.ModelCheckpoint(...)
# kompilovanie modela
ddrnet_model.compile(optimizer= zvoleny_opt_algo,
                     loss= zvolena_chybova_funkcia,
                     metrics= list_zvolenych_metrik,
                     )
# spustenie trénovania
ddrnet_model.fit(dataset_train,
                 epochs=pocet_epoch,
                 dataset_val)
```

6.3. Vyhodnotenie trénovania

Všetky vybrané modely predstavujú rovnakú architektúru konvolučnej siete DDRNet 23 slim a rozdiel bol iba v rozlíšení vstupných dát, to spôsobilo, že priebehy sa navzájom podobali. Preto bol detailnejšie rozobraný celý proces iba pre model DDRNet23s1. Všetky ostane nájdené výsledky boli zhrnuté vo forme tabuľky.

Prvé získané výsledky zobrazené na obr. 6.1a, reprezentujú procesu učenia modelu DDRNet23s1 iba na neupravenom (bez augmentácie) datasete. Takýto tréning bol spustený aby sa potvrdil predpoklad z kap. 4 o hroziacom preučení siete.

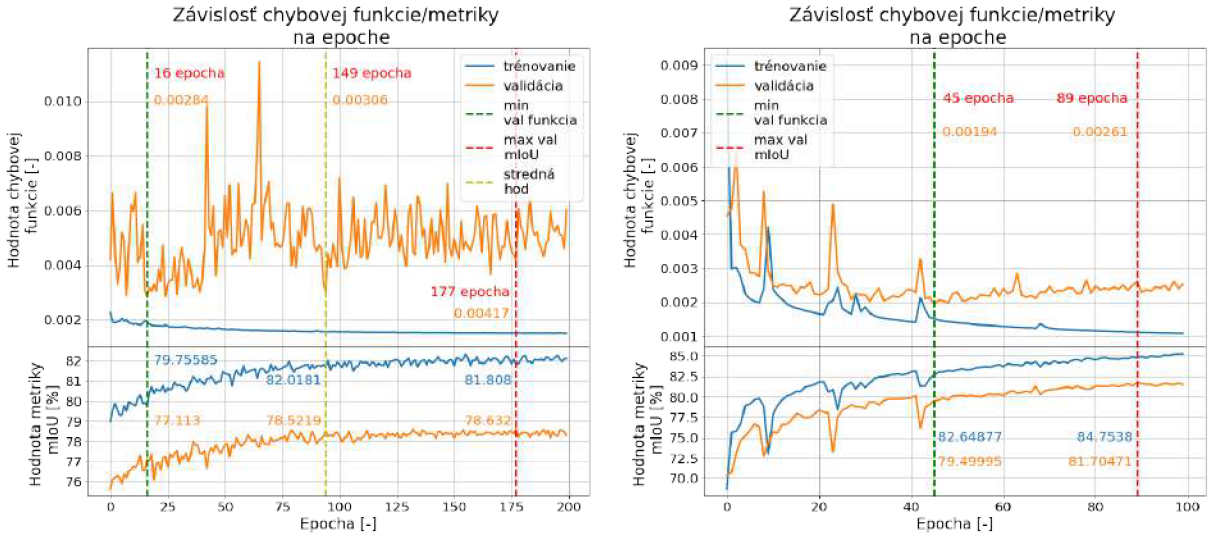
Na hornej časti grafu je vidieť, že približne okolo 27 epochy prestala validačná hodnota chybovej funkcie klesať a opäť začala stúpať, zatiaľ čo hodnota chybovej funkcie trénovania stále klesala. Metrika pre validačné dáta bola po 60 epoche saturovaná približne na 77 % mIoU, keďže však metrika je odolnejšia voči zmenám (nutnosť zhoršenia predikcie tak, že prekročí prah medzi správnou a nesprávnou triedou), nemusí odklon v krivke nastať okamžite.



(a) DDRNet23s1 preučenie (overfitting) (b) DDRNet23s1

Obr. 6.1: Výsledky pre model DDRNet23s1

Výsledky na obr. 6.1a, ktoré boli prezentované vyššie potvrdzujú predpoklad z kap. 4 o preučení ak nebola aplikovaná modifikácia vstupných dát. Sieť sa v procese učenia zlepšovala v predikcií na trénovacom datasete, ale na odlišných vzorkách nedokázala dosiahnuť rovnako dobrú predikcie z hľadiska chybovej funkcie.



(a) DDRNet1 učenie ďalších 200 epoch (b) DDRNet1 fine tunning

Obr. 6.2: Výsledky pre model DDRNet1

Graf 6.1b zobrazuje rovnakú situáciu, ale s tým rozdielom, že dataset obsahoval aj mierne upravené dáta (tak ako boli definované v kap. 4). Výsledkom bolo zabránenie preučenia DDRNet23s1 a dosiahnutie výslednej najnižšej hodnoty validačné chybovej funkcie 0.00262 a 75.335 % mIoU v 82 epoche. Ako je vidieť z grafu priebeh validačnej mIoU stále stúpal, ale hodnota funkcie už nižšie nepoklesla.

Ak by bola zobraná sieť na základe najlepšieho mIoU (graf 6.1b), model by dosiahol hodnoty 0.0029 validačnej chybovej funkcie a 78.06 % mIoU. Chyba by sa zvýšila o 10 % (zhoršila) a metrika zlepšila o 3 %. Avšak skutočnosť, že hodnota funkcie pre validačné dáta ďalej neklesala, ale dokonca začala mierne rásť, mohol naznačovať začiatok preučenia.

Pre overenie tohto predpokladu, bola sieť ďalej trénovaná. Keďže model bol uložený po najlepšiu hodnotu validačnej chybovej funkcie, učenie začalo s váhami, ktoré boli nájdené v epoche 82 (graf 6.1b). Učenie prebiehalo ďalších 200 epoch s počiatočným krokom 0.002, ktorý sa opäť každých 5 epoch exponenciálne zmenšoval. Tento krát bol callback nastavený na uloženie všetkých váh modelu po každej epoche.

Výsledky na grafe 6.2a ukázali, že validačná chyba sa opäť zvýšila oproti najlepšej nájdenej o dvojnásobok (do rozsahu medzi 0.004 až 0.006). Hodnota validačnej metriky sa zasaturovala okolo 78.5 % mIoU s najlepšou nájdenou hodnotou 78.632 % mIoU a s chybou 0.00417.

Podľa priebehu z dodatočného trénovania na grafe 6.2a) sa javilo, že sieť sa začínala príliš sústrediť na trénovacie dáta a postupne prestala generalizovať. Malý krok učenia a nerovnomerné rozloženie tried zapríčinilo, že sa predikcia nezhoršila na toľko aby prekročila prah medzi správnou a nesprávnou triedou a tým ovplyvnila aj krivku metriky.

Metrika modelu teda rastie (pravdepodobne spôsobené nerovnomerným rozložením, kedy rozpoznanie veľkých tried je stále viac jednoduchšie), avšak model si je stále menej istý celkovou predikciou všetkých tried (nárast chybovej funkcie). Najbezpečnejšia verzia siete na použitie (predpoklad najlepšej generalizácie) je teda tá, ktorá dosiahla najnižšiu validačnú chybovú funkciu (graf. 6.1b, epocha 82). Alternatívne by bolo možné zobrať aj parametre pre sieť na grafe 6.2a označené ako stredná hodnota, kde metrika dosiahla 78.52 % pri chybe 0.003. To by predstavovalo zlepšenie metriky o 3.2 % a zhoršenie chyby o 14.5 % oproti prechádzajúcemu modelu. Stredná pozície bola vybraná na základe získania saturovanej val mIoU (78.5 %) s najnižšou chybou.

Na záver bola na vybranú sieť s najnižšou validačnou chybou použitá metóda fine tunning na získanie o niečo lepších výsledkov. Trénovanie bolo vykonané na 100 epochách so začiatočným krokom 0.0002. Callback bol nastavený na uloženie parametrov po každej epoche.

Výsledný priebeh doladenia zobrazuje graf 6.2b. Pre validačnú krivku ukazuje začiatok preučenia, kedy chybová funkcia prestala klesať a opäť začala rásť okolo epochy 45. A práve v tejto epoche sa aj nachádza najnižšia dosiahnutá chyba s veľkosťou 0.00194 a 79.499 % mIoU. Ďalšia vyznačená epocha v grafe 6.2b nebola ďalej braná do úvahy, kvôli spomenutému vzniku preučenia, ktoré je v epoche 89 už znateľné.

6.3.1. Testovanie na reálnych dátach

Testovací dataset, ktorý bol definovaný na začiatku kapitoly predstavuje posledný krok v celom procese a to použitím na evaluáciu natrénovaných konvolučných sieti. Všetky zistenia boli zanesené do nasledujúcej tab.

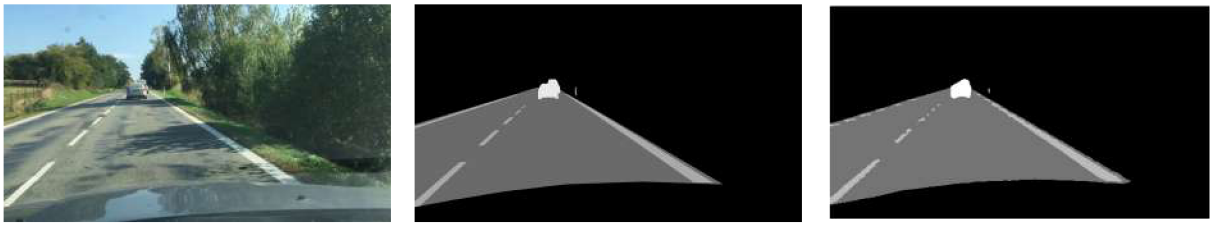
Názov modelu	DDRNet23s1	DDRNet23s2	DDRNet23s3	DDRNet23s4
Rozlíšenie [px]	336 x 600	400 x 720	720 x 1280	1080 x 1920
Najmenšia validačná chybová funkcia (trénovanie) [-]	0.00262	0.00275	0.00244	0.00234
Validačná mIoU (trénovanie) [%]	75.34	78.707	82.170	80.509
Najmenšia validačná chybová funkcia (dolaďovanie) [-]	0.00194	0.00227	0.00239	0.00204
Validačná mIoU (dolaďovanie) [%]	79.50	82.04	82.77	84.92
Najmenšia validačná chybová funkcia (testovanie) [-]	0.00190	0.00206	0.00259	0.00212
Validačná mIoU (testovanie) [%]	79.90	81.29	82.45	84.74

Tabuľka 6.3: Zhrnutie výsledkov

Tab. 6.3 obsahuje zhrnuté výsledky najnižších validačných chybových funkcií a ich súhlasných metrík, ktoré k nim v danú epochu boli vypočítané. Ako je vidieť niektoré hodnoty získané evauláciou testovacieho datasetu boli mierne odlišné. Keďže sa však jednalo iba o malé rozdiely, boli tieto nepresnosti zanedbané. Z tab. bolo možné vidieť jasný vzor, kedy z rastúcim rozlíšením vstupu rastie aj celková metrika mIoU. Najvyššej ohodnotené skoré metriky tak bolo pre model DDRNet23s4. Postupne zvyšovanie mIoU pre siete z vyšším rozlíšením bolo pravdepodobne zapríčinené tým, že pôvodný obraz v datasete má rozlíšenie 1080 x 1920 a pre siete DDRNet(1-3) museli byť vstupné dáta zmenšené a tým sa mohli niektoré detaily na obraze čiastočne skresliť či úplne stratiť.

Ukážka na reálnych dátach

Nasledujúce zobrazené obr. boli zaradené do testovacieho datasetu, takže neboli počas trénovania modelov vôbec používané.

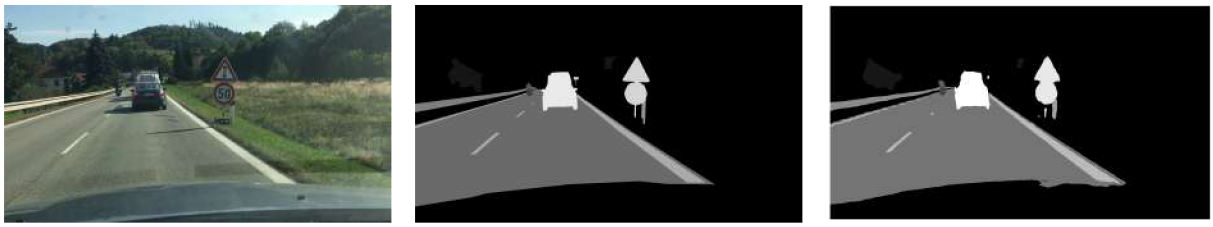


(a) Pôvodný obraz (b) Vytvorená maska (c) Predikovaná maska

Obr. 6.3: Výstup z modelu DDRNet23s1

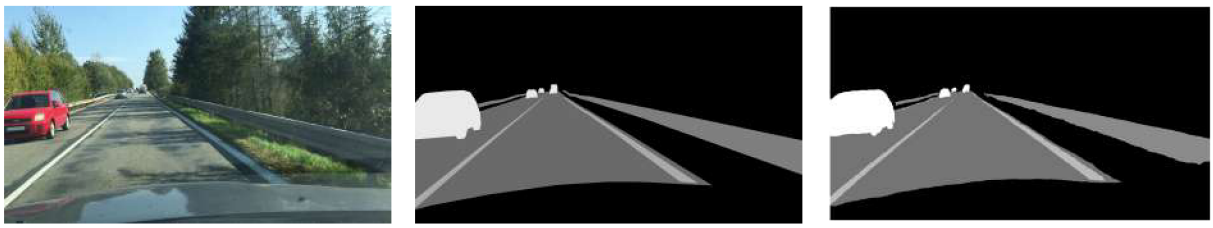
(a) Pôvodný obraz (b) Vytvorená maska (c) Predikovaná maska

Obr. 6.4: Výstup z modelu DDRNet23s2



(a) Pôvodný obraz (b) Vytvorená maska (c) Predikovaná maska

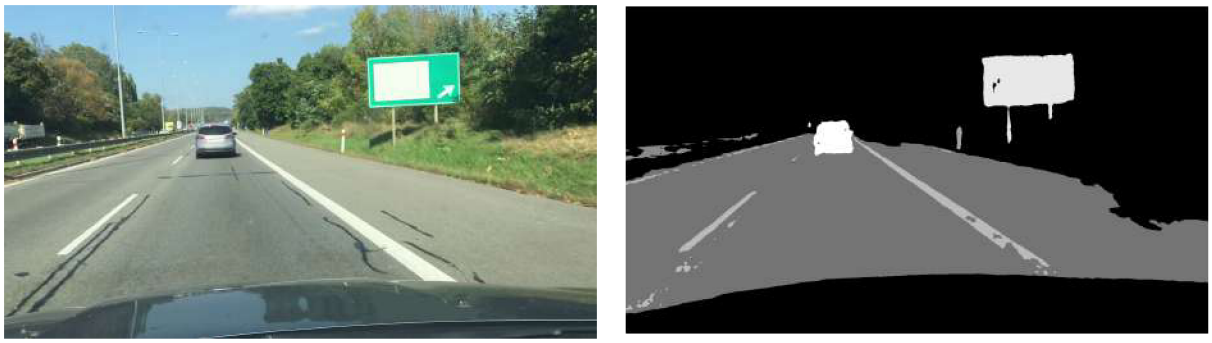
Obr. 6.5: Výstup z modelu DDRNet23s3



(a) Pôvodný obraz (b) Vytvorená maska (c) Predikovaná maska

Obr. 6.6: Výstup z modelu DDRNet23s4

Posledný obr. zobrazujú predikciu na vstupné dáta, ktoré nepochádzajú zo žiadneho z definovaných datasetov v tejto práci, čo znamená že k nemu nebola vytvorená maska a úspešnosť predikcie tak nebolo možné vyjadriť číselne. Ukážka však bola pridaná, aby bola ukázaná aj situácia z bežného nasadenie siete do aplikácie, kedy masky správnej segmentácie taktiež neexistujú.



(a) Pôvodný obraz (b) Predikovaná maska

Obr. 6.7: Ukážka obrazu mimo dataset (model DDRNet4)

7. Test na obmedzenom výkone

Zatiaľ čo predchádzajú kapitoly boli zamerané na rozbor a riešenie problému sémantickej segmentácie. Obsahom tejto kapitoly je použitie natrénovaných sietí DDRNet(1 - 4) na testovanie predikcie na videu v reálnom čase, za účelom zistenia praktického použitia v aplikáciách pre mobilné roboty, ktoré sú limitované menej výkonným hardverom.

Náplň nasledujúcej kapitoly bola rozdelený do niekoľkých nasledujúcich úloh.

1. Optimalizácia natrénovaných modelov.
2. Definovanie parametrov pre testovanie.
3. Vytvorenie nástroja (simulácie) na evaluáciu modelu.
4. Vyhodnotenie získaných výsledkov.

7.1. Optimalizácia modelu

Rýchlosť predikcie modelu na jednom snímku záleží na viacerých parametroch, ako je veľkosť modelu (počet parametrov), použitý formát uloženého modelu či hardvér na ktorom model operuje. Cieľom optimalizácie je čo najviac znížiť závislosť na týchto parametroch. A tým dosiahnuť také zrýchlenie siete, aby bolo možné efektívne spustiť sieť napríklad aj na menej výkonom stroji.

Framework tensorflow predstavuje veľmi dobrý a flexibilný nástroj na vytvorenie a trénovanie architektúry konvolučnej siete, avšak formát v akom operuje nie je vždy dostačujúci na to, aby model bežal požadovanou rýchlosťou. Jednou z možností dosiahnutia vyššej rýchlosti predikcie je prevod do iného formátu.

7.1.1. ONNX Runtime

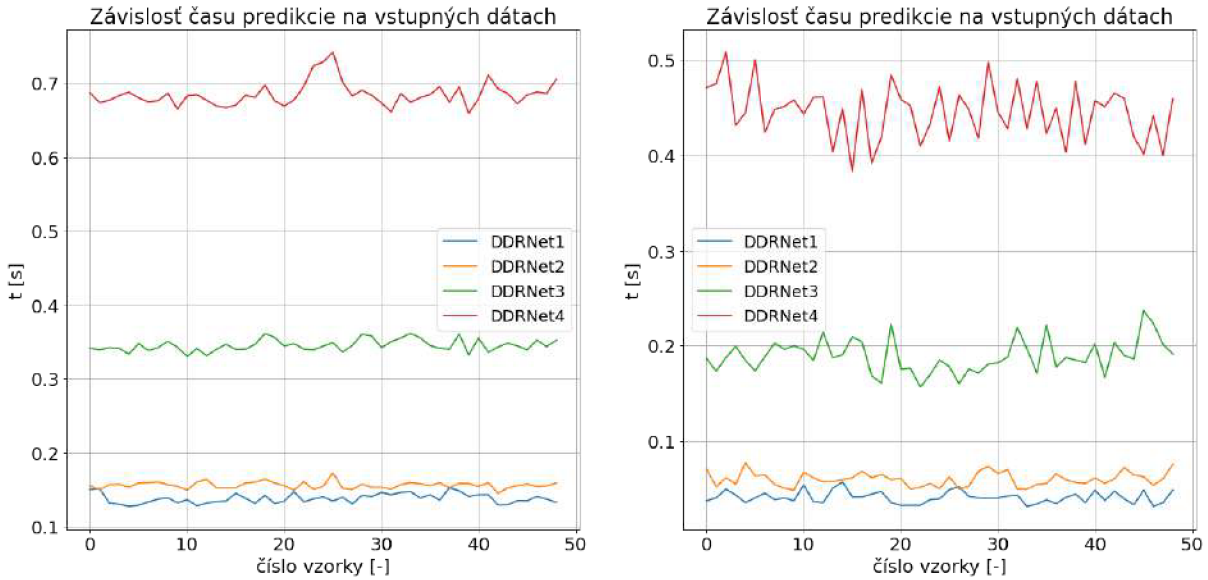
Popis a obsah knižnica ONNX Runtime bol rozobraný v kap. 2. Z praktického hľadiska predstavuje tento nástroj univerzálnu platformu vďaka ktorej je možné model z rôznych frameworkov previesť na ONNX formát a ďalej na ňom pracovať. Výhodnou vlastnosťou knižnice predstavujú možnosti optimalizácie siete, ktoré zaisťujú zrýchlenie modelu.

Prevod vytvoreného modelu z tensorflow do ONNX Runtime je zaistné nasledujúcim kódom.

```
# definovanie rozmerov vstupu
vstup = [tf.TensorSpec([3, 3], tf.float32, name='vstup')]
# konvertovanie
onnx_model, _ = tf2onnx.convert.from_keras(model, vstup)
onnx.save(onnx_model, "/onnx_model.onnx")
```

Pre porovnanie získanej rýchlosti, tensorflow a jeho konvertované verzie predikovali 50 obrazov za použitia rovnakého procesoru. Pôvodná sieť DDRNet1 dosiahla priemerného času 0.1386 sekundy, zatiaľ čo ONNX formát mal priemerný čas 0.0416 sekundy. Toto predstavuje zrýchlenie predikcie siete na jednom vstupe o 70 % oproti tensorflow verzií. Všetky ostatné číselné porovnania boli zapísané do tab. 7.1

Nasledujúci graf obsahuje priebehy času potrebného na predikciu jedného obr. z už načítaného pola pomocou tensorflow a onnx formátu.



(a) Tensorflow model- čas predikcie

(b) ONNX model- čas predikcie

Obr. 7.1: Porovnanie čas predkcie

	DDRNet1	DDRNet2	DDRNet3	DDRNet4
Tensorflow t [s]	0.1386	0.1562	0.3448	0.7051
ONNX t [s]	0.0416	0.0606	0.1900	0.4455
Rozdiel [%]	70.00	61.20	44.90	36.82

Tabuľka 7.1: Porovnanie priemerných časov

Ďalšími možnými optimalizáciami bola kvantizácia. Proces pri ktorom sa zmení presnosť s akou model počíta presnosť predikcie. Avšak na využitie tejto metódy bolo potrebné mať hardvér, ktorý podporuje operácia ako Tensor Core int8 pre GPU či procesor architektúry x86-64 s podporov VNNI[1]. Keďže tieto funkcie väčšinou obsahuje novší hardvér s vyšším výkonom, optimalizácia sieti kvantizáca by nemala žiadny zmysel.

7.2. Definovanie parametrov pre testovanie

Výsledky dosiahnuté v grafe. 7.1 a v tab. 7.1 však predstavovali iba konkrétne časy potrebné na predikciu. Neobsahovali prvky, ktoré by musela mať aplikácia v sebe zahrnuté aby mohla byť úspešne nasadená. Operácie ako čítanie, úprava, predikcia a následne zobrazenie, by pri nesprávne vytvorenej aplikácií pridávali ďalší čas medzi zobrazením dvoch snímkov.

Hlavnou podmienkou použitia segmentácie na mobilnom robotovi je nutnosť zaistiť aby prebiehala relatívne rovnako rýchlo (rovnakou snímkovou frekvenciou) ako kamera, ktorou je vstupný obraz snímaný. Toto je však obmedzené náročnosťou výpočtu danej konvolučnej siete a použitého hardvéru.

[1]Vector Neural Network Instructions

Pre nájdenie závislostí vstupných dát na rýchlosti predikcie a zaťaženiu hardvéru bolo nutné vytvoriť aplikáciu, ktorá simulovala získavanie snímkov s kamery (videa) a vykonávala predikciu. S tým, že rýchlosť aplikácie (zobrazovanie segmentácie) bola obmedzená iba rýchlosť použitého modelu na danom stroji a nie frekvenciou nahraného videa, či zle vytvoreným programom.

Na testovanie boli zvolené dve grafické karty a dva procesory, ktoré sa líšili parametrami, možným dosiahnuteľným výkonom a aj rokom výroby. Vybrané karty a procesy reprezentujú v experimente obmedzený nižší výkon, ktorým disponujú mobilné roboty.

	Názov	Pamäť [GB]	Rok výroby	Zariadenie
GPU	Nvidia GTX 750 Ti	2	2014	PC
	Nvidia GTX 1050 Ti	4	2017	Notebook
CPU	Intel i5-4590	8	2014	PC
	Intel i7-7700HQ	16	2017	Notebook

Tabuľka 7.2: Zvolený obmedzený hardvér

7.3. Nástoj na evaluáciu modela

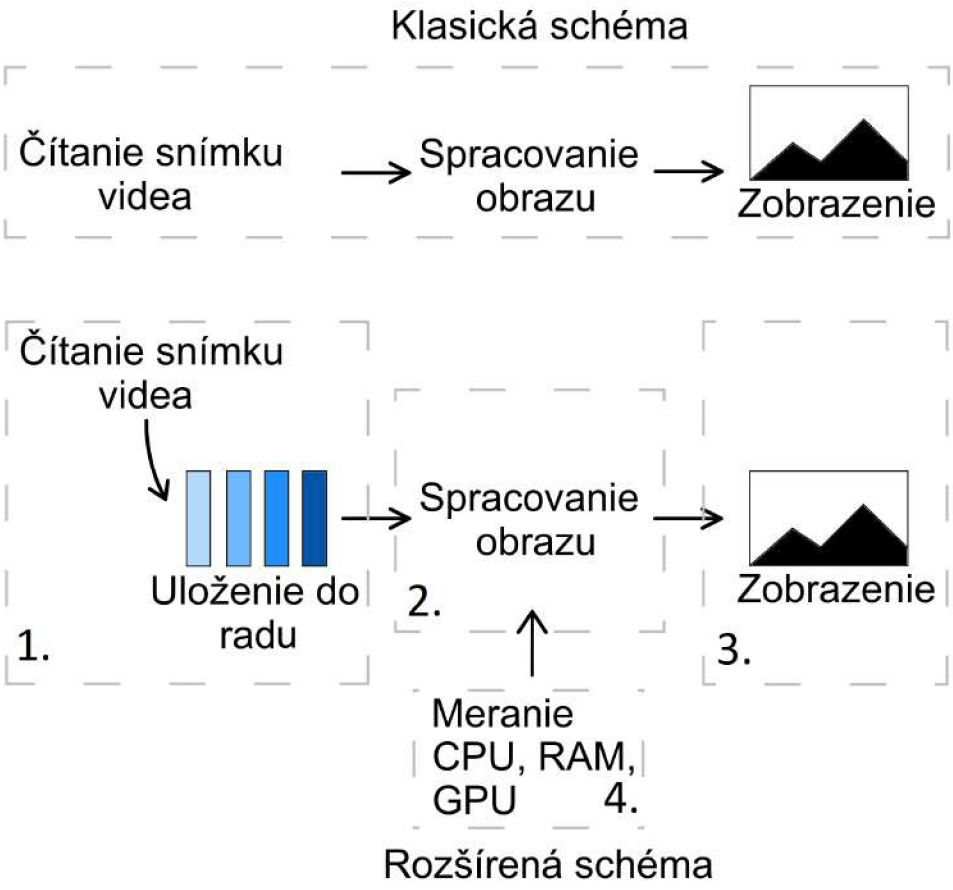
Evaluačný nástroj predstavuje jednoduchú aplikáciu, ktorá simulujú používanie natrénovaného modelu v optimálnom stave. Takýto stav bol definovaný ako situácia, kedy rýchlosť odozvy celého systému (aplikácia) sa rovná odozve modelu (rýchlosť predikcie jedného snímku).

Vytvorenie tohto nástroja vychádzalo z dvoch hlavných parametrov. Neobmedzená rýchlosť simulácie čítania a segmentovania na snímku z videa. A simultánne ukladanie dát rýchlosti predikcie či zaťaženia hardvéru počas tohto procesu.

Postupnosť podľa, ktorej by mal nástroj na testovanie modelu na vybranom videu fungovať môže byť spísaný do nasledujúcich bodov:

1. čítanie prvého/ ďalšieho snímku z videa,
2. vykonanie predikcie na obraze,
3. spracovanie na vytvorenie masky,
4. zobrazenie výsledku.

Vizuálna ukážka tohto procesu sa nachádza na obr. 7.2- prvý z vrchu. Problém tejto schémy pozostáva v nedostatočnej rýchlosti načítavania snímkov z videa. Kedy v takto vytvorenej aplikácií by dochádzalo k nesplneniu podmienky, že odozva systému sa má rovnať rýchlosti modelu. Operácie sa navzájom brzdia, keďže pracujú tak povediac v sériovom zapojení. Ďalšia časť programu nezačne, kým predchádzajúca neskončí. Toto je spôsobené tým, že aplikácia (obr. 7.2- prvý z vrchu) bežala iba na jednom vlákne procesoru (označené ako prerušovaná čiara).



Obr. 7.2: Schéma nástroja na evaluáciu (šedé prerušované čiary značia vlákno procesora)

Preto bola navrhnutá druha rozšírená schéma, zobrazená na obr. 7.2- druhý z vrchu, ktorá stále pracuje podľa rovnako popísanej postupnosti ako predtým, ale s lepšou organizáciou operácií. Systém bol rozdelený z jedného spoločného vlákna na štyri nezávislé bloky (vlákna procesoru) a do schémy bol dodatočne zaradený abstraktný dátový typ "rada"(queue), ktorý funguje na princípe FIFO (first in, first out) zásobníku.

Prvý blok predstavuje dekódovanie snímku z videa (získanie obrazu) pomocou knižnice openCV[2] a následné vloženia do radu (možné zvoliť počet ukladaných snímkov). Pridanie tohto kroku (uloženia snímku do radu) bolo nutné, keďže čítanie videa prebiehalo na samostatnom vlákne a nebolo ničím brzdené. Vytvorený proces bol rádovo rýchlejší ako spracovanie obrazu od modelu a tým pádom ak by bola zo schémy odstránená premenná rad, sieť by dostavala každý n-tý obraz (podľa toho, kedy by model chcel ďalší snímok). Načítanie videa by tak mnohonásobne predbehlo predikciu.

Druhý blok je hlavné vlákno systému na ktorom beží predikcia a ukladanie nameraných hodnôt. Blok beží a vykonáva predikciu pokiaľ existujú dáta, ktoré je schopný vyberať z radu.

Tretí blok zakončuje celú slučku zobrazením výstupu z 2. bloku. A keďže operácia opäť pracovala na vlastnom vlákne, bola brzdená iba rýchlosťou dodávanej predikcie.

[2]opencv- knižnica na spracovanie obrazu

Posledný- štvrtý blok slúžil na meranie a zaznamenanie hodnôt CPU, RAM a GPU, ktoré sa v hlavnej slučky uložia do premennej. Po skončení testu sú pripravené na analyzovanie a následné vyhodnotenie celého experimentu na danom výkone.

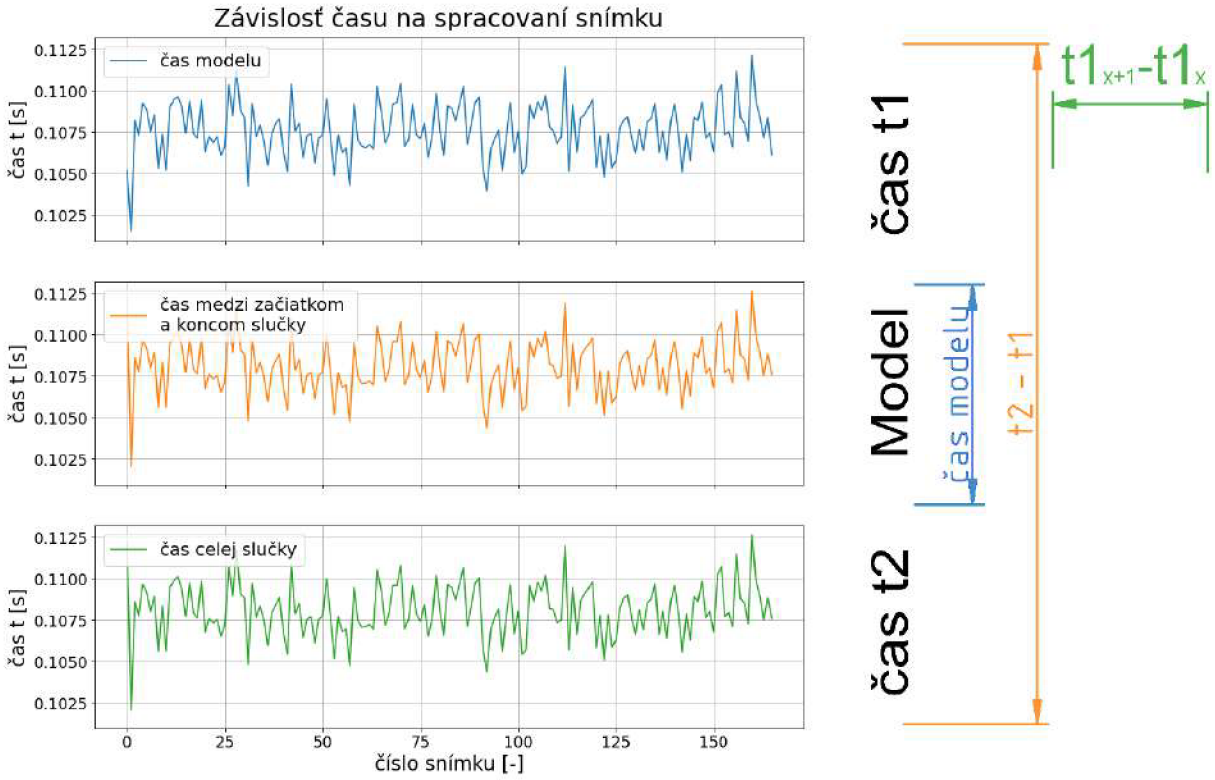
Celý popis fungovania aplikácie bol spracovaný vo forme triedy v jazyku python. Objekt vytvorený pomocou tejto triedy je schopná otestovať každý model, ktorý je definovaný ako funkcia s jedným vstupom (snímka na predikovanie) na ktoromkoľvek zvolenom videu. A po skončení evaluácie je možné z objektu pristúpiť k všetkým nazbieraným hodnotám.

7.3.1. Overenie nástroja na evaluácie modelu

Tento krok predstavuje overenie klúčového parametru navrhnutej aplikácie, ktorým bola možnosť prehrávanie videa rýchlosťou, ktorá nezávisí na tom s akou frekvenciou bolo video vytvorené, ale na tom ako rýchlo dokáže model spracovať jednu snímku.

Pre vytvorenie experimentu bol zvolený nasledujúci postup.

1. Výber akéhokoľvek videa.
2. Natrénovaná konvolučná sieť bola nahradená zdržaním kódu v dĺžke 0.1 sekundy.
3. Zobrazenie načítaného snímku z videa.
4. Uloženie všetkým parametrov.



Obr. 7.3: Overenie aplikácie

Čas modelu vyjadruje priame meranie ako rýchlo model (nahradený ako zdržanie) vykonal predikciu. To znamená, že prvý (modrý) priebeh predstavuje ideálnu krivku, na ktorú by sa mali časy odmerané kdekoľvek v ďalších častiach kódu podobať.

Stredný graf bol získaný meraním času od začiatku po koniec slučky a posledný graf zobrazuje čas, ktorý bol získaný odčítaním dvoch časov nameraných v tom istom mieste. Z výsledných priebehov bolo možné rozoznať, že všetky tri grafy kopírujú krivku ktorú udávalo zvolené zdržanie. Test teda dokázal potvrdiť, že bola splnená hlavná požiadavka, ktorá bola definovaná v podkap. 7.2.

7.4. Simulácia aplikácie modelu

Kapitola obsahuje popis použitia vytvoreného nástroju na evauláciu pre získanie závislostí vstupný veľkostí dát na rýchlostí predikcie na danom hardvéry a jeho zaťaženie počas tohto procesu. Experiment vychádzal z podkap. 7.2, kde bolo určené, že pozitívnym výsledkom je dosiahnutie rovnakej (alebo vyššej) frekvencie snímkovania akou bolo video nahrané.

Inicializácia

Na vytvorenie triedy bolo potrebné definovať 3 hlavné parametre. Záznam videa, ktorý bol použitý na experiment, akýkoľvek model to forme funkcie ktorá dokáže zobrať iba jeden argument, ktorý obsahuje aktuálne vstupné dáta na predikciu. A výstupom funkcie bola už vytvorená segmentovaná maska zo siete. Posledným parametrom bola funkcia, ktorá mala za úlohu spracovať obraz na formát, ktorý model očakáva (zmenšenie, normalizácia, atď.).

```
# trieda pre onnx model
class Onnx_model:
    def __init__(self, model=None, names=None):
        ...
    def predict(self, frame):
        if frame is not None:
            return np.argmax(self.sess.run(self.names, {'input': frame})[0])

# metóda modelu na predikciu
funkcia_modelu = Onnx_model(model, output_names).predict
# úprava obrazu
def transform_func(img=None):
    if img is not None:
        frame = normalizacia(img)
        ...
        return frame
    else:
        return None
# objek simulacie aplikácie
eval_his = Evaluate(zvolene_video, funkcia_modelu,
                    transform_func)
# spustenie
eval_his.start_evaluate()
```

Na experiment bolo použité video, ktoré sa skladalo z 200 snímkov a obsahovalo záznam prejazdu auta Juhomoravským krajom Českej republiky. Nahrané bolo pri 30 snímkoch za sekundu a s rozlíšením 1080 x 1920 pixelov. Každá sieť (tensorflow aj onnx verzia DDRNet23s[1 - 4]) bola otestovaná pre toto video pomocou CPU a GPU (vždy iba jeden z hardvérov). A pre onnx formát boli povolené všetky optimalizácie grafu, o ktoré sa stará sám.

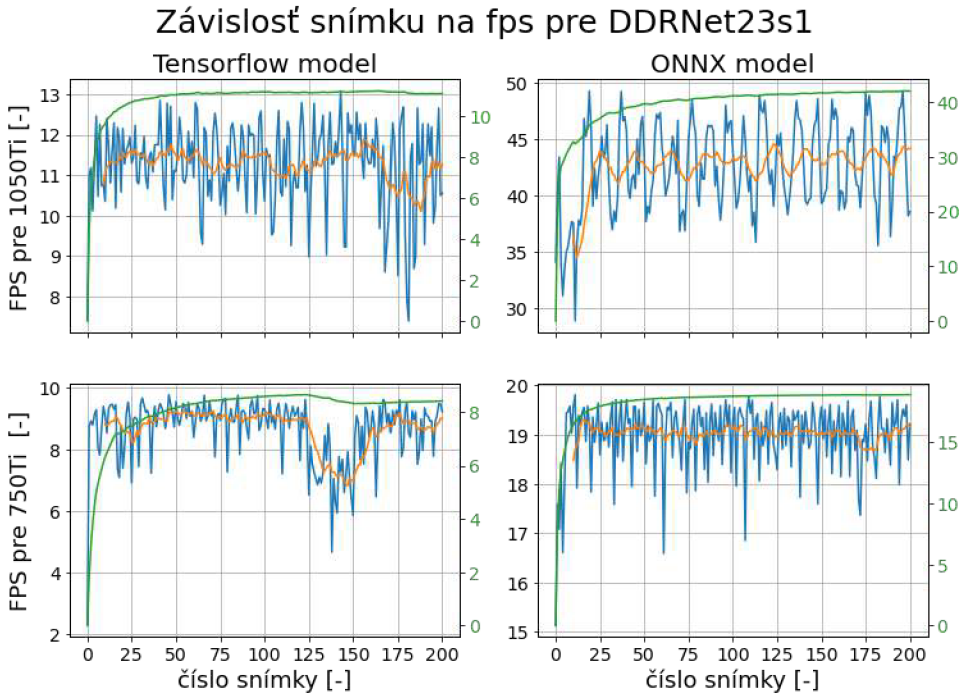
Po vykonaní všetkých kombinácií testov, boli dáta uložené vo formáte .csv súborov, ktoré boli obsiahnuté v rámci prílohy.

Výsledky pre rýchlosť modelu

Pre spracovanie výsledkov bola snímková frekvencia vypočítaná dvoma spôsobmi podľa nasledujúcich vzorcov.

$$FPS1 = \frac{1}{t_s}$$
$$FPS2 = \frac{n}{t_{s2}} \tag{7.1}$$

Kde t_s je čas potrebný na spracovanie jedného snímku, t_{s2} je celkový čas od začiatku segmentovania a n predstavuje narastajúci počet snímkov. FPS1 (na grafe modra krivka) tak vytvára predstavu o tom, ako sa menil čas medi jednotlivými predikciami, zatiaľ čo FPS2 (na grafe reprezentovaná zelenou), odpovedá na otázku ako rýchlo sa model dostane na priemernú hodnotu FPS.

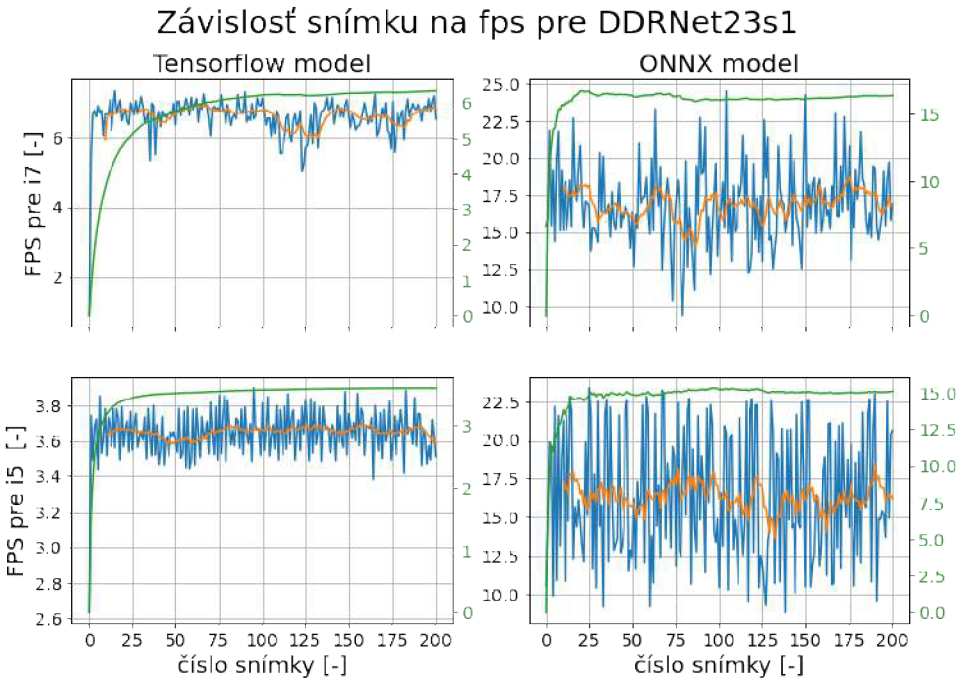


Obr. 7.4: FPS pre DDRNet23s1 na GPU

Ako zobrazuje Graf 7.4, hodnoty FPS2 rastu postupne až sa dosiahnu na priemer. Toto je spôsobene tým, že počiatočný nápočet je obecne pomalší ako tie ktoré nasledujú. Tento rozdiel teda spôsobil, že zelená krivka nezačína na úrovni priemeru modrej. Tretia

krivka na grafe (oranžová) tvorí pohybujúci sa priemer, ktorý bol vypočítaný z FPS1 s oknom zahrňujúcim 10 prvkov.

Najlepšie nájdené hodnoty pre DDRNet23s1 na GPU boli dosiahnuté pre formát onnx s kartou GTX 1050Ti s priemerom 42.55 fps. Táto hodnota zároveň znamená, že model na tomto hardvéry úspešne dokázal dosiahnuť hranicu 30 fps, definovanú v tejto práci ako spracovanie obrazu v reálnom čase.



Obr. 7.5: FPS pre DDRNet23s1 na CPU

U procesorov, dosiahli obidva zvolené hardvéry pre formát onnx podobných hodnôt, i5 16.37 fps a i7 hodnotu 16.88 fps. Čo predstavuje zaujímave zistenie, keďže procesory mali medzi sebou 3 ročný rozdiel od ich vyrobenia. Avšak tieto výsledky nikdy neboli čisto z predikcie, keďže procesor vždy zabezpečoval spustenie a organizáciu celej aplikácie na svojich vláknach.

			DDRNet23s1 [fps]	DDRNet23s2 [fps]	DDRNet23s3 [fps]	DDRNet23s4 [fps]
Tf	GPU	1050Ti	11.25	10.49	4.79	3.01
		750Ti	8.65	7.81	2.94	1.69
	CPU	i7	6.59	5.54	1.93	1.04
		i5	3.64	2.87	1.12	0.57
ONNX	GPU	1050Ti	42.55	32.19	9.44	4.41
		750Ti	18.99	10.98	4.07	2.37
	CPU	i7	16.88	12.53	2.73	1.29
		i5	16.37	11.49	2.85	1.17

Tabuľka 7.3: Závislosť medzi veľkosťou vstupu, použitým hardverom, formátom modelu a hodnoty fps

Keďže priebehy z používania simulovanej aplikácie boli viac menej podobné, ostatné výsledky boli zahrnuté do tab. 7.3, ktorá predstavuje závislosť medzi veľkosťou vstupných dát, použitým hardvérom a dosiahnutej hodnoty fps.

V kompletnej tab. 7.3 boli vyznačené dve hodnoty, ktoré predstavujú dosiahnutie definovanej hranice, určujúcej spracovanie v reálnom čase.

Výsledky zaťaženia hardvéru

Namerané hodnoty, ktoré určujú zaťaženie hardvéru pre dané konvolučné siete, počas predikcie na videu boli spracované vo forme tab., ktorých štruktúra bola symetrická k tab. 7.3. To vytvára ľahšiu orientáciu a umožňuje jednoducho nájsť zaťaženie pre dané testovanie. Vytvorené tab. slúžili ako referencia v prípade, že by boli podobné testy opakované rovnakom hardvéry.

			DDRNet23s1 [%]	DDRNet23s2 [%]	DDRNet23s3 [%]	DDRNet23s4 [%]
Tf	GPU	1050Ti	21.0	86.0	93.0	94.0
		750Ti	48.0	55.0	92.0	100.0
	CPU	i7	8.0	6.0	25.0	22.0
		i5	5.0	4.0	61.0	47.0
ONNX	GPU	1050Ti	44.0	44.0	56.99	61.0
		750Ti	79.0	89.0	100.0	100.0
	CPU	i7	5.0	2.0	22.0	24.0
		i5	5.0	6.0	48.0	43.0

Tabuľka 7.4: Maximálne zaťaženie GPU

			DDRNet23s1 [%]	DDRNet23s2 [%]	DDRNet23s3 [%]	DDRNet23s4 [%]
Tf	GPU	1050Ti	27.38	28.45	66.25	35.88
		750Ti	45.49	41.73	68.66	48.39
	CPU	i7	52.06	55.35	88.87	86.68
		i5	57.12	60.858	98.33	88.28
ONNX	GPU	1050Ti	36.67	34.54	44.37	54.76
		750Ti	42.30	38.42	52.73	58.13
	CPU	i7	68.22	59.57	89.35	80.06
		i5	99.17	98.69	90.05	89.37

Tabuľka 7.5: Priemerné zaťaženie CPU

			DDRNet23s1 [%]	DDRNet23s2 [%]	DDRNet23s3 [%]	DDRNet23s4 [%]
Tf	GPU	1050Ti	62.73	71.85	72.34	74.31
		750Ti	77.39	82.25	87.71	83.35
	CPU	i7	63.77	67.95	68.00	74.89
		i5	64.96	65.11	67.78	75.28
ONNX	GPU	1050Ti	72.60	76.55	74.15	77.93
		750Ti	56.98	56.94	60.20	62.19
	CPU	i7	78.41	77.04	78.33	56.78
		i5	48.42	48.86	53.53	61.27

Tabuľka 7.6: Priemerné využitie pamäte RAM

8. Záver

Diplomová práca sa zaoberala použitím konvolučnej siete na získanie segmentovaného obrazu z kamery mobilného robota. Hlavnou problematikou u tohto zadanie tvorí skutočnosť, že takéto zariadenie je vybavené obmedzeným hardvérom a tak je nutné zvoliť metódu, ktorá dokáže pracovať v takto definovanom prostredí dostatočne efektívne.

Po naštudovaní teoretických základov v kap.2 bola na implemnetáciu vybraná sieť architektúry DDRNet 23 slim, kvôli svojej relatívne malej veľkosti, možnosti použiť k trénovaniu už optimalizované parametre časti modelu a pôvodné výsledky prezentované v článku [38], ktoré ukázali, že sieť bola schopná dosiahnuť cez 100 fps s vysokým rozlíšením vstupu na výkonnej grafickej karte. Na základe týchto výsledkov bol zvolený predpoklad, že sieť by mala byť schopná rýchlosti predikcie aspoň 30 fps (brané ako spracovanie v reálnom čase) s nižším rozlíšením vstupu na menej výkonnej karte.

Keďže pôvodná implementácia existuje iba pre Pytorch, v rámci kapitoly 5 bolo architektúra vytvorená pomocou frameworku tensorflow s dopredu zadanými názvami vrstiev tak, aby odpovedali pôvodnému modelu. Následne bolo možné tensorflow sieť namapovať získanými parametrami pre použitie metódy transfer learning vrámci trénovania. Pre kontrolu implementácie bol vytvorený jednoduchý test, kedy boli do obidvoch modelov vložené náhodne generované dáta a ako výstup získaný maximálny rozdiel medzi vrstvami s rovnakým menom. Celková najvyššia hodnota bolo 7.6^{-6}, čím bola overená správna funkčnosť nového modelu.

V rámci vypracovania diplomovej práce bol taktiež definovaný nový dataset, ktorý obsahoval 5824 vzoriek s 12 možnými triedami segmentácie.

Vzniknutá sieť a dataset boli použité pre trénovanie štyroch samostatných modelov DDRNet 23 slim s rozdielnou veľkosťou vstupu. Dosiahnuté výsledky boli zhrnuté do tab. 6.3. Najlepší model v rámci najnižšej chybovej funkcie vyšiel DDRNet23s1 (rozlíšenie 336 x 600 px) s hodnotami dosiahnutými na testovacom datasete 0.00190 a s metrikou mIoU 79.9 %. Pre prípad najvyššej metriky to bol model DDRNet23s4 (rozlíšenie 1080 x 1920 px) s hodnotami 0.00212 a 84.79 % mIoU.

Na záver bola vytvorená jednoduchá aplikácia, ktorá zabezpečovala, že frekvencia zobrazenia jedného snímku sa rovná rýchlosti s akou sieť dokáže na danom hardvéry pracovať. Na základe čoho bolo možné zistiť hodnotu fps pre každý model na každom zo zadaných hardvérov (tab. 7.2). DDRNet23s1 a DDRNet23s2 na grafickej karte GTX 1050Ti boli ako jediné schopné dosiahnuť spracovanie v reálnom čase (minimálne 30 fps) s hodnotami 42.55 fps a 32.19 fps. Všetky ostatné namerané veličiny boli zanesené do tab. 7.3. Opisom poslednej úlohy, tak boli splnené všetky zadané ciele práce.

Literatúra

[1] Image segmentation. *Wikipedia.org* [online]. 2022-04-28 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Image_segmentation

[2] DADHICH Abhinav. *Practical Computer Vision*. Packt Publishing, 2018 [cit. 2022-05-20]. ISBN 9781788297684.

[3] Image processing. *Wikipedia.org* [online]. 2022-04-04 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Thresholding_(image_processing)

[4] Image processing. *Homepages.inf.ed.ac.uk* [online]. c2003 [cit. 2022-05-20]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/threshld.htm

[5] DEY Sandipan. *Hands On Image Processing with Python*. Packt Publishing, 2018 [cit. 2022-05-20]. ISBN 9781789343731.

[6] Edge detection. *Wikipedia.org* [online]. 2022-05-04 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Edge_detection

[7] SAITOH, Koki. *Deep Learning from the Basics*. Packt Publishing, 2021 [cit. 2022-05-05]. ISBN 9781800206137.

[8] GLASBEY C. A. a G. W. HORGAN. *Image Analysis for the Biological Sciences*. Wiley, 1995 [cit. 2022-05-20]. ISBN 978-0471937265.

[9] Region growing. *Wikipedia.org* [online]. 2021-10-16 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Region_growing

[10] K-means clustering. *Wikipedia.org* [online]. 2021-10-16 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/K-means_clustering

[11] Convolutional neural network. *Wikipedia.org* [online]. 2022-04-28 [cit. 2022-05-20]. Dostupné z: https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24

[12] SEWAK Mohit, Md. Rezaul KARIM a Pradeep PUJARI. *Practical Convolutional Neural Networks*. Packt Publishing, 2018 [cit. 2022-05-20]. ISBN 9781788392303.

[13] ZAFAR Iffat, Giounona TZANIDOI, Richard BURTON, Nimesh PATEL a Leonardo ARAUJO. *Hands-On Convolutional Neural Networks with TensorFlow*. Packt Publishing, 2018 [cit. 2022-05-20]. ISBN 9781789130331.

[14] Residual neural network. *Wikipedia.org* [online]. 2022-04-19 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Residual_neural_network

[15] KAIMING, He, Xiangyu ZHANG, Shaoqing REN a Jian SUN. *Deep Residual Learning for Image Recognition*. In: arXiv. 2015. DOI: https://doi.org/10.48550/arXiv.1512.03385. [cit. 2022-05-05].

[16] Residual blockk. *Paperswithcode.com* [online]. 2022-04-19 [cit. 2022-05-20]. Dostupné z: https://paperswithcode.com/method/residual-block

[17] Transfer learning. *Wikipedia.org* [online]. 2022-04-03 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Transfer_learning

[18] Transfer learning. *Cs231n.github.io* [online]. 2022-04-03 [cit. 2022-05-20]. Dostupné z: https://cs231n.github.io/transfer-learning/

[19] Fine tuning. *Pyimagesearch.com* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/

[20] Fine tuning. *Tensorflow.org* [online]. 2022-05-20 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/tutorials/images/transfer_learning

[21] Cross entropy. *Wikipedia.org* [online]. 2022-05-04 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Cross_entropy

[22] Cross entropy. *Machinelearningmastery.com* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://machinelearningmastery.com/cross-entropy-for-machine-learning/

[23] Sparse Categorical Crossentropy. *Tensorflow.org* [online]. 2022-05-18 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

[24] Focal loss. *Paperswithcode.com* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://paperswithcode.com/method/focal-loss

[25] LIN Tsung-Yi, Priya GOYAL, Ross GIRSHICK, Kaiming HE a Piotr DOLLÁR. *Focal Loss for Dense Object Detection*. In: arXiv. 2017. DOI: https://doi.org/10.48550/arXiv.1708.02002. [cit. 2022-05-05].

[26] IoU. *Pyimagesearch.com* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[27] Jaccard index. *Wikipedia.org* [online]. 2022-04-05 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Jaccard_index

[28] F score. *Wikipedia.org* [online]. 2022-04-09 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/F-score

[29] Classification accuracy. *Machinelearningmastery.com* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

[30] Confusion matrix. *Wikipedia.org* [online]. 2022-04-19 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Confusion_matrix

[31] Prezentácia Detection and Segmentation. *Stanford.edu* [online]. c2017 [cit. 2022-05-20]. Dostupné z: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[32] LONG Jonathan, Evan SHELHAMER a Trevor DARRELL. *Fully Convolutional Networks for Semantic Segmentation*. In: arXiv. 2014. DOI: https://doi.org/10.48550/arXiv.1411.4038. [cit. 2022-05-20].

[33] RONNEBERGER Olaf, Phillip FISCHER a Thomas BROX. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. In: arXiv. 2015. DOI: https://doi.org/10.48550/arXiv.1505.04597. [cit. 2022-05-20].

[34] BADRINARAYANAN Vijay, Alex KENDALL a Roberto CIPOLLA. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. In: arXiv. 2015. DOI: https://doi.org/10.48550/arXiv.1511.00561. [cit. 2022-05-20].

[35] ZHAO Hengshuang, Jianping SHI, Xiaojuan QI, Xiaogang WANG a Jiaya JIA. *Pyramid Scene Parsing Network*. In: arXiv. 2016. DOI: https://doi.org/10.48550/arXiv.1612.01105. [cit. 2022-05-20].

[36] YU Fisher a Vladlen KOLTUN. *Multi-Scale Context Aggregation by Dilated Convolutions*. In: arXiv. 2015. DOI: https://doi.org/10.48550/arXiv.1511.07122. [cit. 2022-05-20].

[37] YU Changqian, Jingbo WANG, Chao PENG, Changxin GAO, Gang YU a Nong SANG. *BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation*. In: arXiv. 2018. DOI: https://doi.org/10.48550/arXiv.1808.00897. [cit. 2022-05-20].

[38] HONG Yuanduo, Huihui PAN, Weichao SUN a Yisong JIA. *Deep Dual-resolution Networks for Real-time and Accurate Semantic Segmentation of Road Scenes*. In: arXiv. 2021. DOI: https://doi.org/10.48550/arXiv.2101.06085. [cit. 2022-05-20].

[39] Keras. *Wikipedia.org* [online]. 2022-04-05 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Keras

[40] TensorFlow. *Wikipedia.org* [online]. 2022-04-01 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/TensorFlow

[41] TensorFlow základy. *Tensorflow.org* [online]. 2022-02-16 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/tutorials

[42] Sequential model. *Tensorflow.org* [online]. 2022-05-18 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

[43] Functional API. *Tensorflow.org* [online]. 2022-01-10 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/guide/keras/functional

[44] Subclass model. *Tensorflow.org* [online]. 2022-01-10 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/guide/keras/custom_layers_and_models

[45] tf data API. *Tensorflow.org* [online]. 2022-04-19 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/guide/data

[46] ONNX Runtime. *Onnxruntime.ai* [online]. c2020 [cit. 2022-05-20]. Dostupné z: https://onnxruntime.ai/about.html

[47] Tensorflow lite. *Tensorflow.org* [online]. 2022-03-31 [cit. 2022-05-20]. Dostupné z: https://www.tensorflow.org/lite/guide

[48] Cvat. *Cvat.org* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://cvat.org/

[49] Label Studio. *Labelstud.io* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://labelstud.io/

[50] Makesense. *Makesense.ai* [online]. c2022 [cit. 2022-05-20]. Dostupné z: https://www.makesense.ai/

9. Zoznam príloh

- Všetky použité skripty boli uložené na stranke https://github.com/StanDanis/DDRNet-Tensorflow