VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2023 Samuel Bielik

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

DETEKCE SJÍZDNOSTI TERÉNU PRO MOBILNÍ ROBOT POMOCÍ RGB KAMERY

CAMERA-BASED TERRAIN TRAVERSABILITY ASSESSMENT FOR MOBILE ROBOTS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Samuel Bielik

AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Gábrlík, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Samuel Bielik ***ID:*** 230041

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

NÁZEV TÉMATU:

Detekce sjízdnosti terénu pro mobilní robot pomocí RGB kamery

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je analyzovat prostor před mobilním robotem ve vnějším prostředí za pomocí RGB kamery a algoritmu pro sémantickou segmentaci s cílem posoudit sjízdnost pro danou platformu.

1. Proveďte průzkum metod a softwarových nástrojů na bázi strojového učení sloužící ke klasifikaci prostředí pomocí obrazových dat z RGB kamery. Rovněž se seznamte s frameworkem ROS.
2. Seznamte se s principem funkce fotoaparátu, projekcí a strukturou obrazových dat; otestujte elementární operace, jako je konvoluce.
3. Zvolený nástroj pro segmentaci zprovozněte, naučte na dostupném datasetu pro automotive a otestujte na různých fotografiích.
4. Doučte algoritmus pro rozlišování tříd i mimo zpevněné komunikace za použití dostupného terénního datasetu či vlastních dat. Úspěšnost detekce ověřte na vlastním terénním datasetu.
5. Připravte softwarové řešení tak, aby segmentace probíhala na videu v reálném čase a proveďte testování dle pokynů vedoucího.

DOPORUČENÁ LITERATURA:

MURPHY, Robin R. Introduction to AI Robotics. Massachusetts: MIT Press, 2002. ISBN 9780262133838.

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

Vedoucí práce: Ing. Petr Gábrlík, Ph.D.

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

Abstrakt

Práca popisuje princíp fungovania kamery, počítačové videnie a konvolučné neurónové siete, sémantickú segmentáciu a DeepLab. Práca sa zaoberá tréningom modelu, zobrazovaním obrázkov, aplikáciou modelu na video a segmentovanie v reálnom čase

Kľúčové slová

Konvolučné neurónové siete, sémantická segmentácia, DeepLab, obrázok, počítačové videnie, kamera

Abstract

The thesis describes digital camera working principle, computer vision and convolutional neural networks, semantic segmentation and DeepLab. The thesis deal with model training, image visualisation, model's application on video and segmentation in real time.

Keywords

Convolutional neural networks, semantic segmentation, DeepLab, image, computer vision, camera

Bibliografická citácia

BIELIK, Samuel. Detekce sjízdnosti terénu pro mobilní robot pomocí RGB kamery. Brno, 2023. Dostupné tiež z: https://www.vut.cz/studenti/zav-prace/detail/151648. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce Petr Gábrlík.

Prehlásenie autora o pôvodnosti diela

Meno a priezvisko študenta:	*Samuel Bielik*
VUT ID študenta:	*230041*
Typ práce:	*Bakalárska práca*
Akademický rok:	*2022/23*
Téma záverečnej práce:	*Detekce sjízdnosti terénu pro mobilní robot pomocí RGB kamery*

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

Ako autor uvedenej záverečnej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušení ustanovení § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovení časti druhé, hlavy VI. dirl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brne dňa: 16. mája 2023 ----------

podpis autora

Poďakovanie

Ďakujem vedúcemu mojej bakalárskej práce Ing. Petrovi Gábrlíkovi Ph.D. za odborné rady, konzultácie a poskytnutie iného pohľadu na problémy pri trénovaní modelu a spracovaní tejto práce.

V Brne dňa: 16. mája 2023 ----------

podpis autora

Obsah

ZOZNAM OBRÁZKOV

Úvod

Predstavme si robotické vozidlo, ktoré sa pohybuje v teréne. Pre analýzu okolia používa lidar a kameru. Výstupom z lidaru je mračno bodov kde každý jeden bod predstavuje potenciálnu prekážku (pretože odniekiaľ sa lidarový lúč musel odraziť). Lidar však vidí aj podložku, po ktorej sa pohybuje, prípadne, ak je vozidlo v uzavretom priestore, vidí aj to, čo je nad ním. Podložky a stropy by bolo dobré z mračna odstrániť, pretože toto nepredstavuje pre vozidlo prekážku. Celkom dobre sa dajú filtrovať roviny, ako na príklad podlahy a stropy. Problém však nastáva vtedy, ak sa robot pohybuje na príklad po tráve, kde mračno bodov už nie je rovina. Cieľom práce je pomocou dát z kamery analyzovať povrch pred robotom. Dáta z kamery spracuje robot pomocou sémantickej segmentácie. To znamená, že každému bodu z obrázku priradí robot nejakú kategóriu do ktorej patrí. Pomocou projekcie vie potom robot zistiť, ktorý pixel z kamery prislúcha určitému bodu v mračne bodov. Ak by sa jednalo o nejaký štrk, trávu alebo poorané pole, cez ktoré by robot mal vedieť prejsť, tento bod v mračne odfiltruje a tým pádom ho nevníma ako prekážku.

Obrazové dáta musí robot nejako získať, takže sa práca zaoberá tým, čo je to kamera, ako funguje, základy projekcie a spôsoby ukladania obrazových dát. Získané dáta sa musia nejako spracovať, práca teda rozoberá počítačové videnie, konvolučné neurónové siete a sémantickú segmentáciu. V robotovi, pomocou ktorého sa získajú testovacie dáta, je operačný systém Linux s ROS2, preto popisuje ROS2.

V praktickej časti sa práca zaoberá modelom DeepLab a jeho tréningom. Po natrénovaní rozoberá ako zobraziť obrázky a prácu modelu v reálnom čase.

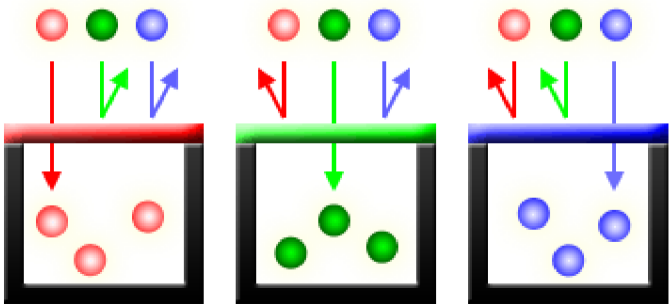
1. TEÓRIA

1.1 Digitálny obraz

Na obraz sa dá pozerať ako na dvojrozmernú funkciu z = f (x,y), kde x a y sú súradnice pixelov a z je amplitúda (čím je väčšia intenzita jasu v bode [x,y], tým väčšie číslo bude z). Jednotlivé pixely obrazu sú prvky matice. Najčastejšie sa amplitúda kóduje 8 bitovo (pixely nadobúdajú hodnôt 0-255, kde 0 je úplne čierna a 255 je úplne biela). Ak treba kódovať farebný obraz, tak jeden farebný pixel sa skladá z červenej, zelenej a modrej farby. Tieto farby sa spolu poskladajú. Intenzita každej farby môže tiež nadobúdať hodnoty 0-255. Pre uloženie farby jedného pixelu preto treba 24 bitov, čo sú 3 byty. Niekedy sa kóduje aj priehľadnosť daného pixelu. [1], [2]

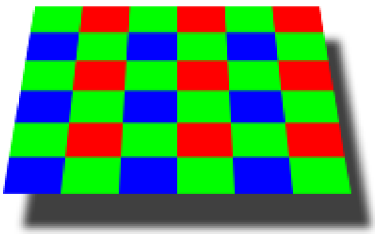
1.2 Kamera

Kamera zachytáva svetlo, ktoré prechádza šošovkami na obrazový snímač a signál ukladá do pamäte. Dôležitú časť kamery tvorí obrazový snímač. Obrazový snímač je elektronická súčiastka, ktorá mení optický signál na elektrický. Skladá sa z malých buniek, v ktorých sa zhromažďujú fotóny. Pracuje takýmto spôsobom: Na začiatku zachytávania snímky sa bunka otvorí, dopadajúce svetlo vo forme fotónov sa mení na elektróny. Po určitom čase sa bunky zatvoria a zmeria sa veľkosť elektrického náboja. Veľkosť elektrického náboja je úmerná intenzite svetla. Pred tým, ako fotóny dopadnú do bunky, prechádzajú filtrom (jeden filter na jednu bunku), čo ukazuje obrázok 1.1. Filtre majú za úlohu prepustiť iba určitú zložku svetla, konkrétne červenú, zelenú alebo modrú. Keby na bunkách filtre neboli, kamera by nevedela určiť farbu dopadajúceho svetla a preto by výsledný obraz bol čiernobiely. [3], [4]



Obrázok 1.1 Bunky s farebnými filtrami umiestnené na obrazovom snímači [3]

Filtre môžu byť na snímači rôzne rozložené. Jednou z možností je Bayerova maska, čo ukazuje obrázok 1.2. [3]



Obrázok 1.2 Bayerova maska [3]

Z obrázku 1.2 vidno, že v Bayerovej maske sa striedajú modro zelené a červeno zelené stĺpce. Z toho vyplýva, že zelených filtrov je na snímači dvakrát viac ako červených alebo modrých. Je to kvôli tomu, že ľudské oko je citlivejšie na zelenú farbu. Ak by bolo každého z filtrov rovnaký počet, výsledný obraz by sa zdal zašumený a menej detailný. [3] Ďalšie typy filtrov uvádza zdroj [5].

Základom snímača je kremíková fotodióda. Dopadajúce fotóny sa menia na elektróny vplyvom napájacieho napätia diódy. [4]

Najpoužívanejšie typy obrazových snímačov, sú CCD a CMOS. [6]

CCD snímače sú založené na poli pasívnych fotodiód (pixely), na ktorých sa zhromažďuje náboj počas expozície kamery (bunky sú otvorené). Potom elektronika umiestnená mimo fotodiód prepočítava veľkosť náboja na napätie. Premena svetla na napätie je veľmi účinná, čo sa hodí vtedy, keď sa fotografuje v tme. Elektronika vyhodnocujúca veľkosť napätia na fotodiódach je pre každý pixel rovnaká, a preto je dosiahnutá vysoká jednotnosť obrazu. Nevýhodou je to, že sú pomalé a celkom drahé. [6]

CMOS snímače sú založené na poli aktívnych pixelov – elektronika je na úrovni pixelov a výstupom zo snímača je náboj prevedený na veľkosť napätia. CMOS snímače sú rýchlejšie ako CCD, ale sú viac zašumené. [6]

1.2.1 Resekcia kamery

Resekcia kamery je proces odhadu parametrov dierkového modelu kamery (pinhole camera model) z fotografie. To znamená, že určuje smer svetelného lúču spojeného s konkrétnym pixelom obrázka. Zvyčajne sú parametre kamery reprezentované projekčnou maticou 3x4. Vonkajšie (extrinsic) parametre definujú pózu kamery a vnútorné (intrinsic) definujú formát obrazu (ohniskovú vzdialenosť, veľkosť pixela a pôvod obrazu). [7]

Projekčná matica je odvodená z vonkajších a vnútorných parametrov fotoaparátu a často je reprezentovaná sériou transformácii. Táto matica môže byť použitá na priradenie bodov z obrázku (2D) do priestoru okolo kamery (3D). [7]

Na popis pozície pixelu obrazu a pozície prislúchajúceho bodu v priestore sa používajú homogenizované vektory:

- Obraz: $[u\ v\ s]^T$

- 3D priestor: $[x_w\ y_w\ z_w\ s]^T$

kde s označuje mierku. To znamená, že označuje to, koľkokrát je vektor zväčšený.
Projekcia:

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R\ T] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = M \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (1.1)$$

- M je projekčná matica, a $M = K[R\ T]$
- u,v sú súradnice pixelu
- K je matica vnútorných parametrov
- R a T sú matice vonkajších parametrov (R je rotácia a T je translácia)
- x_w, y_w, z_w sú súradnice bodu v priestore
- z_c je z súradnica kamery v 3D priestore

Matica vnútorných parametrov je:

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (1.2)$$

Táto matica obsahuje 5 vnútorných parametrov modelu kamery. Jedná sa o ohniskovú vzdialenosť, formát snímača obrázka a hlavný bod kamery (principal camera point). Parametre αx = f·mx a αy = f·my hovoria o ohniskovej vzdialenosti v pixeloch (f je ohnisková vzdialenosť, m sú prevrátené hodnoty výšky a šírky pixela na snímači – veľkosť bunky). γ je parameter skreslenia medzi osami x a y, často je nulový. u0 a v0 reprezentujú hlavný bod kamery pričom ideálne by mali byť v strede obrázku. [7]

Matica vonkajších parametrov:

$$\begin{bmatrix} Rotácia_{3x3} & Translácia_{3x1} \\ Perspektíva_{1x3} & Mierka \end{bmatrix} \quad (1.3)$$

Vonkajšie parametre sú rotácia a translácia, vďaka ktorým sa dajú transformovať 3D súradnice bodu niekde vo svete do 3D súradníc kamery. Súradnice vektora translácie označujú svetové súradnice počiatku v súradnicovom systéme kamery. Súradnice kamery vo svetovom súradnicovom systéme sú:

$$C = -Rotácia^{-1} \cdot Translácia = -Rotácia^T \cdot Translácia \quad (1.4)$$

Je potrebné si uvedomiť, že pri odfotení nejakej scény vzniká prechod z 3D sveta do 2D roviny. Tento proces znižuje počet rozmerov z 3 na 2. Každý pixel na obrázku teda korešponduje s nejakým lúčom svetla z pôvodnej scény. [7]

1.3 Typy formátov obrazových súborov

Dáta z kamery je potrebné nejako uložiť. Na to sa používajú rôzne typy formátov.

1.3.1 JPEG (.jpg, .jpeg)

Jedná sa o stratový formát, kódovanie je 24 bitové (8 bitov pre každú zložku RGB), nepodporuje priehľadnosť. Kompresia zmenšuje veľkosť potrebnej pamäte, ale tým pádom aj kvalitu, ktorú však bežne človek nezaregistruje. JPEG nefunguje správne pri obrázkoch s ostrými hranami a veľkými jednofarebnými oblasťami. [8]

1.3.2 PNG (.png)

PNG pri kompresií nestráca žiadne dáta a preto je o mnoho väčší ako GIF alebo JPEG. PNG obrázky môžu byť priehľadné alebo polopriehľadné. Tak isto ako JPEG, kódovanie je 24 bitové. Formát nie je patentovaný. [9]

-

1.3.3 Raw súbory (.raw, .cr2, ...)

Súbor RAW obsahuje nekomprimované a nespracované údaje zachytené priamo z kamery. RAW je kvalitný a zaberá veľa pamäte. Súbory RAW sú typom rastrového formátu súborov, ale sami o sebe nie sú obrázkami. To znamená, že ich treba najskôr importovať do nejakého softvéru (Photoshop) a až potom sa dajú upraviť alebo exportovať ako iné rastrové obrázkové súbory. [10]

1.4 Neurónové siete v počítačovom videní

Počítačové videnie (Computer vision) využíva algoritmy umelej inteligencie (artificial intelligence) a hlbokého učenia (deep learning). Úlohou počítačového videnia je zanalyzovať obsah obrázka a získať z neho informácie. Počítačové videnie využíva konvolučné neurónové siete (Convolutional neural network). Pre spoľahlivé fungovanie modelu, ktorý využíva počítačové videnie, je potrebné obrovské množstvo dát. [11]

Hlavné úlohy počítačového videnia sú klasifikácia (classification), detekcia objektov (object detection), sémantická segmentácia (semantic segmentation) a inštančná segmentácia (instance segmentation). [12]

1.4.1 Konvolúcia

Konvolúcia (Convolution) je matematická operácia, ktorá sa aplikuje na 2 funkcie a jej výstupom je tretia funkcia, ktorá popisuje, ako je tvar jednej funkcie ovplyvnený druhou funkciou. [13]

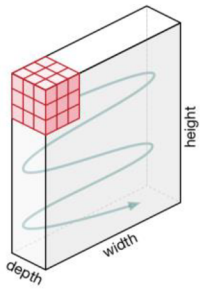
To vlastne popisuje, že jedna funkcia je signál vstupujúci do lineárneho systému, druhá funkcia popisuje tento lineárny systém a výstup zo systému sa rovná konvolúcií signálov. Jedná sa teda o matematickú operáciu pre určenie odozvy lineárneho systému. [14]

1.4.2 Konvolučná neurónová sieť

Konvolučné neurónové siete sa primárne používajú v oblasti rozpoznávania vzorov na

obrázkoch. Jedným z najväčších obmedzení obyčajných umelých neurónových sietí je to, že zápasia s obrovskou výpočtovou zložitosťou potrebnou na získanie dát z obrázku. Ak je potrebné naučiť model rozpoznávať obrázok na príklad z databázy MINST (obsahuje ručne napísané číslice, je to jeden zo základných datasetov), tak takýto obrázok má veľkosť 28x28 pixelov. Každý neurón v prvej skrytej vrstve neurónovej siete preto obsahuje 282, vstupov a teda aj 784 váh. Ak je ale treba pracovať s obrázkom 64x64, už má jeden neurón v prvej skrytej vrstve 12 288 váh, čo je príliš veľa, pretože k dispozícii nie je neobmedzené množstvo výpočtového výkonu, času a pamäte. Ďalším problémom je preučenie (overfitting). Nastáva vtedy, keď je model až príliš naučený na tréningovú množinu a nevie zovšeobecňovať. To znamená, že ak bude na vstupe nový obrázok, ktorý model nikdy predtým nevidel, nevedel by ho spracovať. [15]

Konvolučná neurónová sieť pozostáva z 3 typov vrstiev: konvolučná, poolovacia (pooling) a plne prepojená vrstva. (fully connected layer, dense layer). [15]

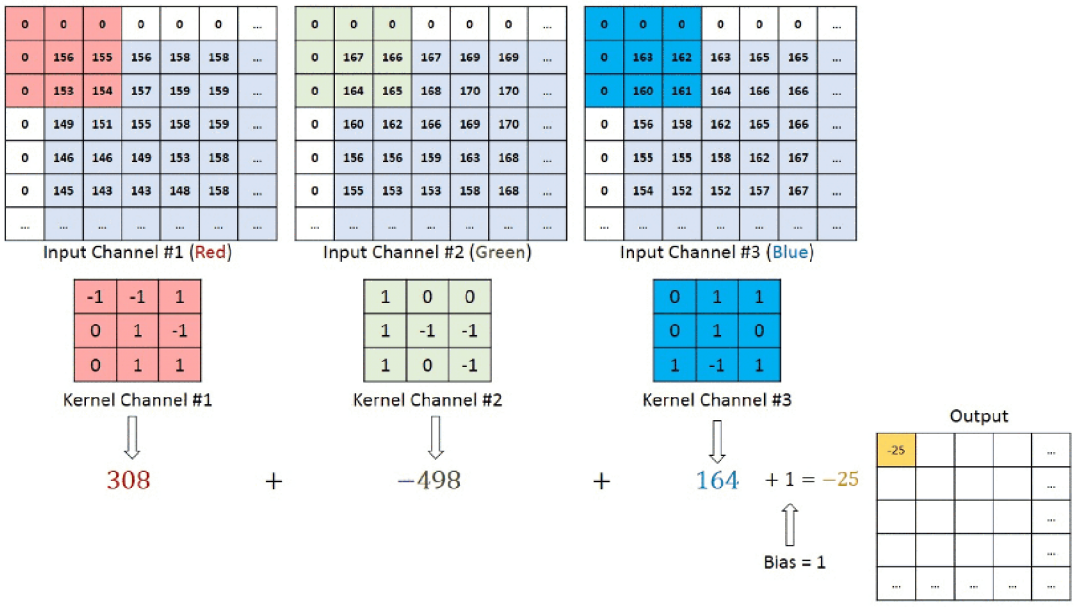


Obrázok 1.3 Pohyb konvolučného jadra po obrázku [15]

Na obrázku 1.3 vidno biely kváder, ktorý predstavuje vstupný obrázok (dĺžka, šírka, hĺbka – pre kódovanie RGB), červená kocka sa nazýva kernel, konvolučné jadro alebo taktiež aj filter. Filter sa posúva cez celý obrázok o krok doprava, postupne sa elementárne násobí so vstupným obrázkom a súčty súčinov si zapisuje. Keď je v jednom smere na konci, posunie sa na začiatok, o krok nižšie a robí to isté dookola až kým nedôjde na koniec obrázka (obrázok 1.4). To, o koľko pixelov sa filter posúva (či sa posunie o 1,2,3,...,x pixelov) závisí od parametru krok (stride). Cieľom konvolučnej neurónovej siete je získať z obrázku jeho vlastnosti. V prvej konvolučnej vrstve sa získavajú jednoduché vlastnosti, ako sú hrany objektov a farba. V ďalších sa už získavajú zložitejšie vlastnosti. Na obrázku 1.4 predstavuje šedo modrá časť matice vstupné dáta. Okolo matice vstupných dát pribudli nuly. O týchto nulách hovorí parameter obálka (Padding) – podľa toho, aké číslo mu prislúcha, toľko rádov núl bude okolo. Namiesto núl môžu tvoriť okolie vstupu krajné hodnoty pôvodnej matice. Obálka zabezpečuje to, že výstup z konvolučnej vrstvy má taký istý rozmer, ako vstup. [15]

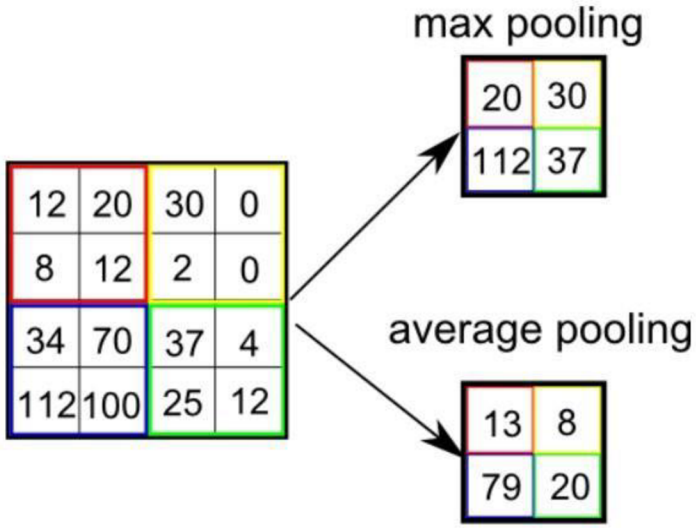
Výsledok z konvolučnej vrstvy môže vstúpiť do aktivačnej funkcie. Tým sa do siete pridáva nelinearita a tak sa môže sieť naučiť riešiť komplexnejšie problémy. Jedným

z príkladov je ReLU (rectified linear unit), kde sa zo záporných hodnôt spravia nuly a ostatné sa nechajú bez zmeny. Ďalšie typy aktivačných funkcií popisuje zdroj: [16]



Obrázok 1.4 Priebeh konvolúcie [15]

Výstupu z konvolučnej vrstvy sa hovorí mapa vlastností (feature map). Ďalej nasleduje poolovacia vrstva. Jej úlohou je zmenšiť obrázok z dôvodu obmedzenia potrebného výpočtového výkonu a celkového zrýchlenia. Okrem toho je dobrá na získanie dominantných prvkov, ktoré sú rotačne a pozične nezávislé. Je viacero typov poolovania. Obrázok 1.5 ukazuje max, kde sa z oblasti pokrytej kernelom vyberá najväčšie číslo a average, kde sa počíta priemer.



Obrázok 1.5 Poolovanie [15]

Konvolučné a poolovacie vrstvy sa za seba niekoľkokrát spoja. Na koniec nasleduje plne prepojená vrstva. To je obyčajná umelá neurónová sieť, ktorá sa dokáže učiť. Vstupom do tejto siete sú vlastnosti obrázka získané z predchádzajúcich vrstiev a výstupom je nenormalizovaná pravdepodobnosť s akou obrázok patrí do určitej skupiny. [15]

Výstup má tvar vektora, ktorého veľkosť je taká istá ako počet neurónov v poslednej vrstve. Na tento vektor sa ešte môže aplikovať funkcia Softmax.

1.4.3 Softmax

Softmax je aktivačná funkcia, ktorej funkčný predpis je:

$$Softmax(\hat{y}_i) = \frac{e^{\hat{y}_i}}{\sum e^{\hat{y}_j}} \tag{1.5}$$

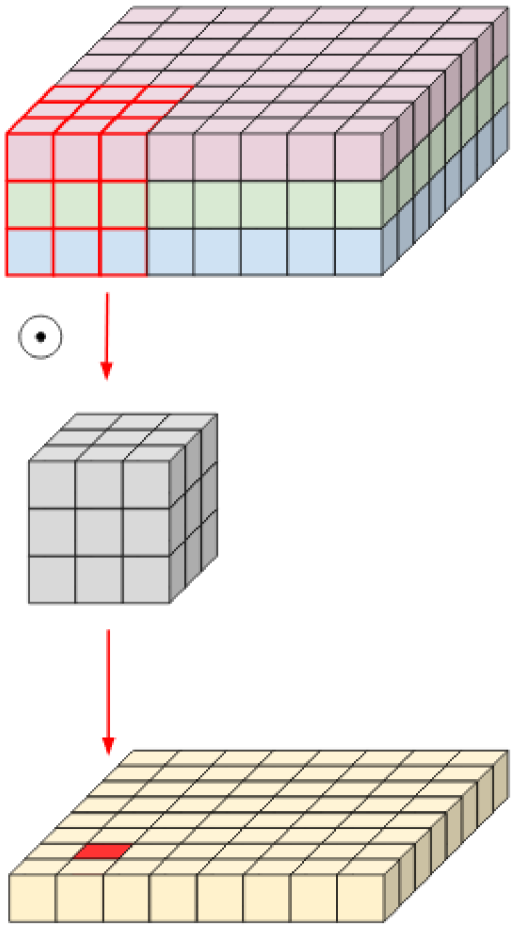
Softmax aplikujem na každú súradnicu vektora. Čitateľ funkcie tvorí exponenciálna funkcia, ktorej základ je eulerovo číslo a exponentom je hodnota aktuálnej súradnice vektora. Menovateľ funkcie tvorí suma takýchto exponenciálnych funkcí, kde sa na mieste exponenta vystriedajú všetky hodnoty vo vektore. Význam funkcie softmax je v tom, že výsledné hodnoty posúva viac od seba oproti obyčajnej normalizácii (výsledok podelím súčtom výsledkov). [17]

1.4.4 Práca s farebnými obrázkami a do hĺbky separovateľná konvolúcia

Farebný obrázok sa skladá väčšinou z 3 kanálov (jeden pre každú zložky R,G,B). Z Filtra v prvej konvolučnej vrstve sa preto stáva tenzor veľkosti (šírka) × (výška) × hĺbka=3, čo ukazuje obrázok 1.6. Ak sú veľkosti filtra 5×5×3 tak jeden filter má 75 parametrov. Ak sa na konvolúciu využije iba tento jeden filter, tak výstupný obrázok má veľkosť (šírka obrázka) × (výška obrázka) × 1. Filtrov však býva viac, tak, na príklad, ak sa použije v jednej vrstve 20 takýchto filtrov, tak sa používa 1500 parametrov a veľkosť výstupu je (šírka obrázka) × (výška obrázka) × 20. To zaberá veľké množstvo pamäte a na konvolúciu sa spotrebúva veľa času. Z dôvodu veľkého množstva parametrov, v sieti taktiež môže prísť k preučeniu. [18]

Tento problém rieši do hĺbky separovateľná konvolúcia (depthwise separable convolution). Tá sa skladá z konvolúcie do hĺbky (depthwise convolution) nasledovanej bodovou konvolúciou (pointwise convolution). Konvolúcia do hĺbky pracuje s každým kanálom (na vstupe do siete to sú 3 farebné zložky) samostatne (výstupom sú tenzory veľkosti (šírka vstupu) × (výška vstupu) × 1) a ich výsledky následne spojí (výstup je tenzor veľkosti (šírka vstupu) × (výška vstupu) × (počet vstupných kanálov)). Ukazuje to obrázok 1.7. Na tento tenzor sa ďalej uplatní bodová konvolúcia, čo ukazuje obrázok 1.8. Jej filter má veľkosť 1×1×(počet kanálov). Bodová konvolúcia sa spraví toľkokrát, koľko filtrov sieť v danej vrstve požaduje vyžaduje. Obrázok 1.8 ukazuje požiadavku na jeden filter. Takže ak sieť potrebuje 20 filtrov 5x5 ako v prípade vyššie, tak používam $3 \cdot 5 \cdot 5 + 3 \cdot 20 = 135$ parametrov. Aj keď obyčajná konvolúcia a do hĺbky separovateľná

konvolúcia zaberajú podobné množstvo času, tak obyčajná konvolúcia prebieha toľkokrát, koľko je potrebných filtrov a do hĺbky separovateľná konvolúcia prebehne len raz. Takže okrem pamäte sa šetrí aj čas. [18], [19]



Obrázok 1.6 Konvolúcia s viackanálovým vstupom [18]

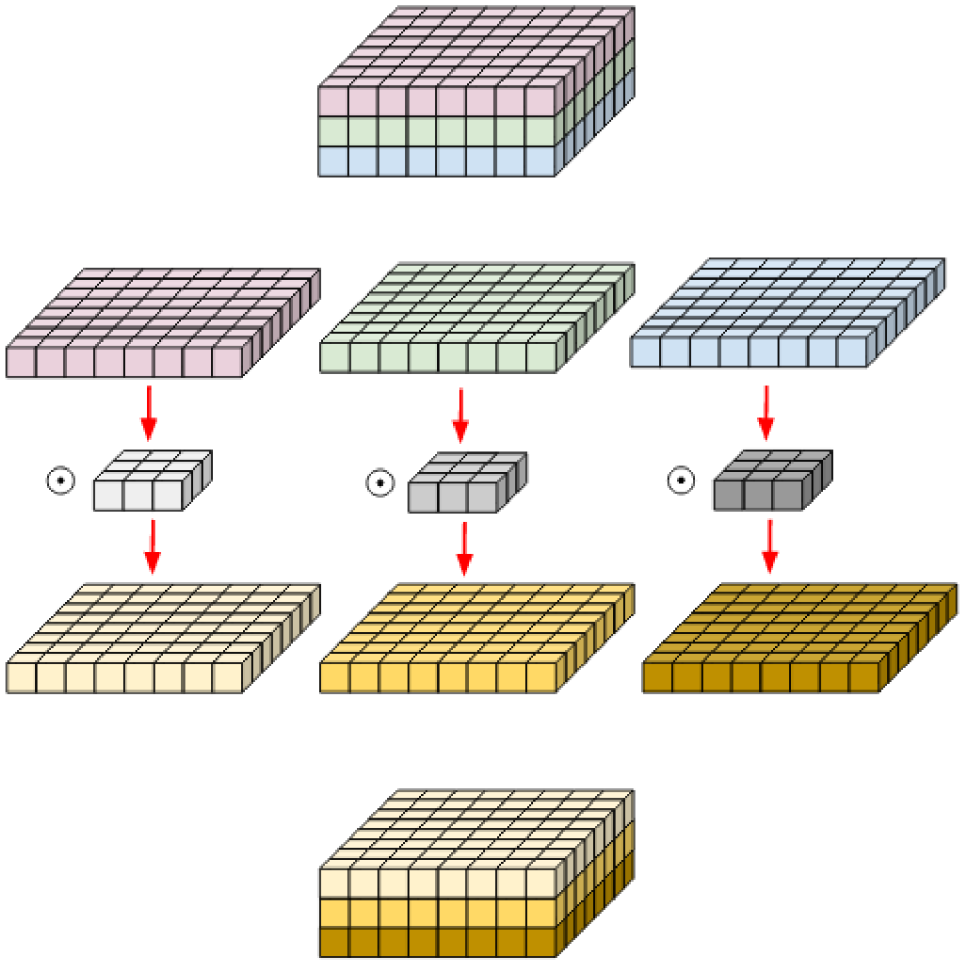
1.4.5 Tréning konvolučnej neurónovej siete

V konvolučnej neurónovej sieti sa dá pozerať na hodnoty v konvolučnom jadre ako na váhy, ktoré sa násobia so vstupným signálom, teda časťou obrazu. Tieto súčiny sa následne sčítajú a v jednom cykle vznikne jedna hodnota. Presne takto sa ale správa neurón. Preto sa dajú na konvolučnú neurónovú sieť použiť algoritmy, ktoré sa používajú v umelých neurónových sieťach, aj keď konvolučná neurónová sieť nemusí obsahovať ani jeden neurón. Počas tréningu sa menia váhy a teda hodnoty v konvolučnom jadre.

Na začiatku tréningu sa sieti predloží dataset plný obrázkov, na ktorých sa sieť bude učiť. Ďalej sa kernel inicializuje na náhodné hodnoty. Tréning prebieha v niekoľkých cykloch nazývaných epochy. V každej epoche sú sieti predložené obrázky a anotované

obrázky k nim prislúchajúce. Pre tréning sa počíta chybová funkcia Loss, a pomocou nej a algoritmu Backpropagation (spätné šírenie chyby) sa upravujú hodnoty v kerneli (adaptácia váh) dovtedy, kým hodnota chybovej funkcie nebude dostatočne nízka. Chybová funkcia sa môže počítať buď vždy po predložení jedného obrázku, po predložení niekoľkých obrázkov alebo po predložení všetkých obrázkov. O tom hovorí parameter veľkosť dávky (batch size). [13], [20]

Viac informácií a konkrétnu implementáciu algoritmu Backpropagation popisuje zdroj [20].



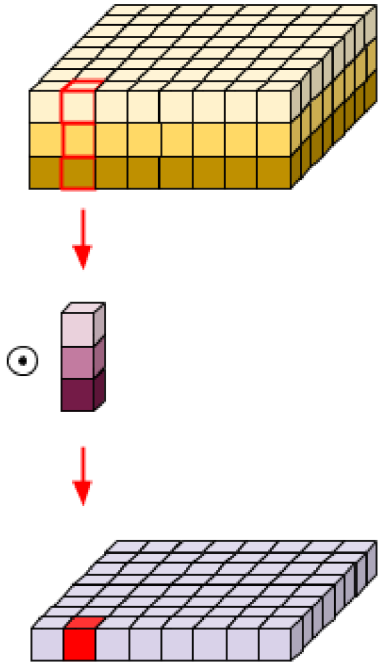
Obrázok 1.7 Do hĺbky separovateľná konvolúcia [18]

V jednoduchosti funguje algoritmus Backpropagation tak, že si vezme chybovú funkciu, parciálne ju zderivuje podľa aktuálne menenej váhy, výsledok vynásobí parametrom koeficient učenia (learning rate) a toto odčíta od aktuálnej váhy:

$$w_x(k+1) = w_x(k) - \alpha \cdot \frac{\partial L}{\partial w_x} \tag{1.6}$$

Problém je v tom, že chybová funkcia sa nepočíta priamo z váh a preto vyššie uvedená derivácia nie je známa. Avšak toto obmedzenie sa dá obísť tak, že sa najskôr

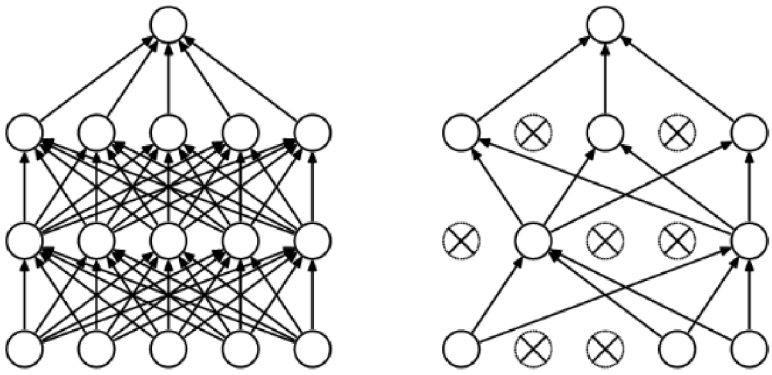
zderivuje chybová funkcia podľa výstupu zo siete (napr. z funkcie softmax), následne sa výstup zo siete zderivuje podľa vstupu do poslednej vrstvy a tak ďalej. Proste vždy sa derivuje výstup nejakej operácie podľa vstupu do nej, až kým sa nedostane k požadovaným váham. Tieto derivácie sa následne vynásobia a tým sa získa výsledok derivácie. [21]



Obrázok 1.8 Bodová konvolúcia [18]

1.4.6 Dropout

Dropout je metóda, ktorá v umelých neurónových sieťach rieši problém preučenia. Nastáva vtedy, keď sa používa veľmi hlboká sieť a tá potom nevie zovšeobecňovať. Ak je sieť preučená, vie veľmi dobre pracovať s datasetom, na ktorom sa trénuje, ale je úplne zlá, pokiaľ má pracovať s novým, ešte nepoznaným obrázkom. Cieľom tréningu je, aby bola chybová funkcia čo najnižšia, respektíve nulová. Veľkosť chybovej funkcie ovplyvňujú všetky neuróny. Preto sa počas preučenia môžu váhy adaptovať takým spôsobom, že chyba jedného neurónu sa kompenzuje iným neurónom. Ak sieť používa dropout, tak výstupy náhodných neurónov v konkrétnej vrstve vynuluje – takže tieto neuróny odstráni. To znamená, že sa počas tréningu mení architektúra siete, čo ukazuje obrázok 1.9. Takýmto spôsobom sa neuróny nesnažia kompenzovať chybu iných neurónov. V konvolučných neurónových sieťach funguje dropout tak, že vynuluje náhodné mapy vlastností. [22]

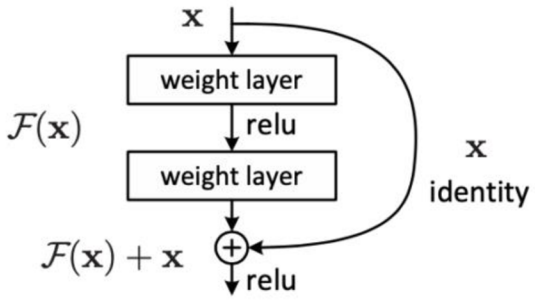


Obrázok 1.9 Princíp dropout-u [22]

1.4.7 Reziduálne bloky

Ak sa konvolučná neurónová sieť skladá z veľa vrstiev, môže sa stať, že derivácie chybovej funkcie popísané v kapitole o tréningu budú veľmi malé (vanishing), alebo naopak, obrovské (exploding), čo v najhorších prípadoch môže viesť až k tomu, že sa sieť už nebude ďalej učiť. [23]

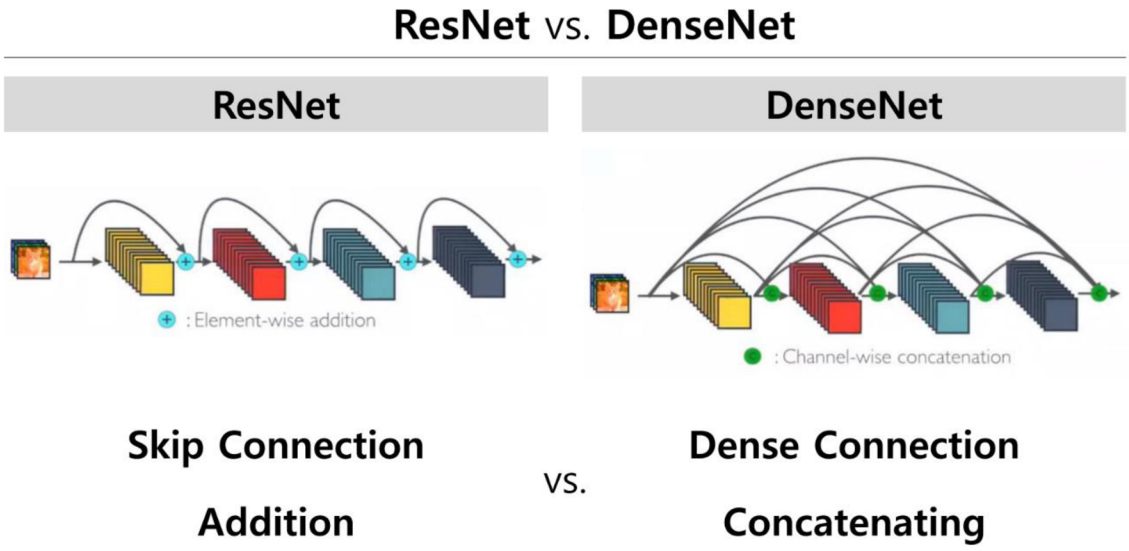
Pre vyriešenie tohto problému sa používa metóda preskakovania spojení (skip connections). Niekoľko konvolučných vrstiev sa spojí dokopy a vytvoria reziduálny blok, čo ukazuje obrázok 1.10. Na začiatku tréningu nerobia tieto vrstvy nič, preskočia sa, a ďalej sa počíta s výsledkami z predchádzajúcej vrstvy. Toto preskočenie zrýchľuje počiatočný tréning, pretože celý model sa tvári tak, že má menej vrstiev. Neskôr, keď je potrebné model pretrénovať, už sa využívajú všetky vrstvy modelu. Táto technika zjemňuje tok gradientu chybovej funkcie a tým je možné mať siete s viac vrstvami. [24]



Obrázok 1.10 Reziduálny blok. Do aktivačnej funkcie (v tomto prípade relu) ide výsledok z poslednej konvolučnej vrstvy (F(x)) sčítaný so vstupom do bloku (x) [25]

Príkladom siete, ktorá využíva reziduálne bloky je ResNet a DenseNet. Do bloku siete ResNet vstupuje predchádzajúci výsledok, naproti tomu DenseNet používa pre vstup do nasledujúceho bloku všetky predchádzajúce výsledky. Tieto bloky sú potom

prepojené prechodovou (transition) vrstvou, ktorú tvorí konvolúcia a pooling. Porovnanie ukazuje obrázok 1.11. [26]



Obrázok 1.11 Vľavo architektúra ResNet, vpravo DenseNet. Architektúra štandardnej CNN by nemala šípky nad jednotlivými vrstvami (farebné štvorčeky za sebou) [27]

1.4.8 Dávková normalizácia

Na to aby konvolučná neurónová sieť fungovala správne, potrebuje mať na vstupe normalizované dáta. Ak sú však k dispozícii iba nenormalizované, tak sa môže použiť dávková normalizácia (Batch normalization), ktorá prebieha medzi vrstvami v sieti. Táto normalizácia sa vykonáva v malých dávkach, nie na celom datasete. Rovnica dávkovej normalizácie: [28]

$$z_{norm} = \left(\frac{z - m_z}{s_z}\right)\gamma + \beta \qquad (1.7)$$

- z je výstup neurónu predtým, ako je naň aplikovaná aktivačná funkcia
- m_z je priemer týchto výstupov
- s_z je štandardná odchýlka výstupov (o nej viac na príklad tu: [29]
- γ (štandardná odchýlka) a β (priemer) sú parametre dávkovej normalizácie, ktoré sa menia počas tréningu. Pre konvolučnú neurónovú sieť sú tieto parametre rovnaké pre všetky vstupy. [28]

Použitím dávkovej normalizácie je učenie modelu rýchlejšie, výsledky majú menšiu chybovosť a teda model je presnejší. Správna hodnota koeficientu učenia leží vo väčšom intervale (je prijateľných viac hodnôt). Taktiež môže byť koeficient učenia rapídne väčší a tým sa môže model pri učení vyhnúť lokálnym minimám chybovej funkcie. [30]

1.4.9 Pred trénované modely

Ak sa niekedy niekto zaoberal podobným problémom ako je potrebné teraz vyriešiť, je možné použiť jeho pred trénovaný model (pretrained model respektíve transfer learning), namiesto toho, aby sa vymýšľala nová architektúra siete a aby sa míňali dlhé hodiny tréningom, ktorý nikam nemusí viesť. Pred trénovaný model sa nemusí dokonalo hodiť na požadovanú aplikáciu, dá sa upraviť rôznymi spôsobmi:

- Odstráni sa iba posledná vrstva, ktorej výstupom sú pravdepodobnosti. Inak sa necháva model bez zmeny (využívajú sa aj váhy)
- Využije sa architektúra modelu, váhy sa inicializujú na náhodné čísla a model sa pretrénuje na nový dataset
- Váhy niektorých vrstiev sa pretrénujú (hlbšie vrstvy), iné sa nechajú bez zmeny (vrstvy na začiatku siete). [31]

1.4.10 Cross-entropy Loss

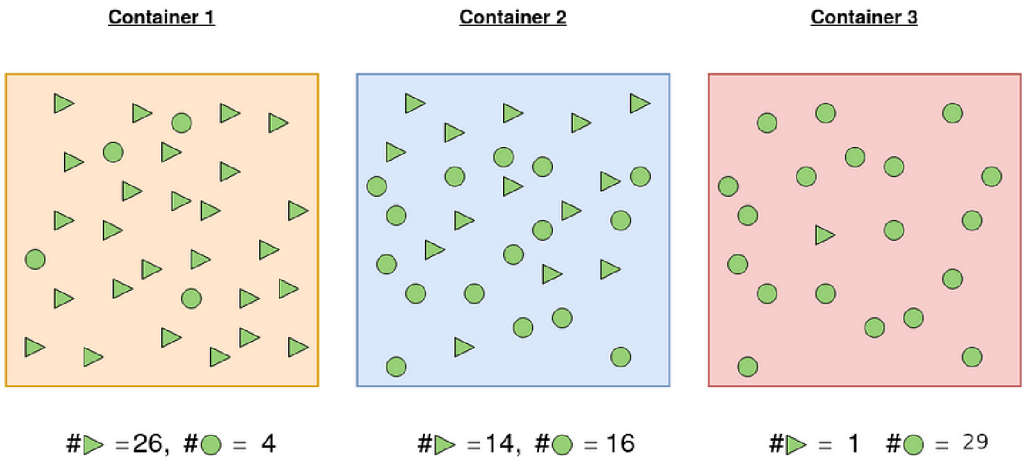
Cross-entropy loss je jedna z najdôležitejších chybových funkcií. V ideálnom prípade by výstupom modelu bol vektor, ktorý obsahuje jednotku (kategória do ktorej patrí obrázok) a nuly (zvyšné kategórie). Výstupom modelu je však vektor, ktorý obsahuje rôzne čísla v intervale medzi 0 a 1. Cieľom chybových funkcií je zistiť, ako ďaleko je model od pravdy pri zaradení objektu do konkrétnej triedy. [32]

Entropia popisuje úroveň neurčitosti. Diskrétna entropia je popísaná vzorcom:

$$H(x) = -\sum p(x) \cdot \log_2(p(x)) \qquad (1.8)$$

- H je vypočítaná entropia
- p(x) je pravdepodobnosť toho, že nastane prípad x

Čím väčšia je hodnota entropie, tým si je model menej istý svojou predikciou. Príklad ukazuje obrázok 1.12. [32]



Obrázok 1.12 Kontajnery s rôznymi útvarmi a ich počet [32]

Na obrázku 1.12 majú všetky kontajnery rovnaký súčet rôznych útvarov. V prvom kontajneri je 26 trojuholníkov a 4 kruhy, a teda pravdepodobnosť toho, že vybraný útvar je trojuholník je 26/30 a toho, že bude vybraný kruh je 4/30. V druhom kontajneri je 14 trojuholníkov a 16 kruhov, a teda pravdepodobnosť toho, že bude vybraný trojuholník je 14/30 a toho, že bude vybraný kruh je 16/30. V treťom kontajneri je iba 1 trojuholník a 29 kruhov, preto pravdepodobnosť toho, že bude vybraný trojuholník je 1/30 a toho, že bude vybraný kruh je 29/30. Z toho všetkého vyplýva, že najvyššia neurčitosť výberu konkrétneho tvaru z kontajnera je pri kontejneri 2 a najnižšia pri kontajneri 3. Entropia pre prvý kontajner bude:

$$H(x) = -\sum p(x) \cdot \log_2(p(x)) = -\left(\frac{26}{30} \cdot \log_2\left(\frac{26}{30}\right) + \frac{4}{30} \cdot \log_2\left(\frac{4}{30}\right)\right) = 0{,}5665 \tag{1.9}$$

Podobne sa dá vypočítať entropia pre zvyšné dva prípady. Pre druhý kontajner je entropia 0,9968 a pre tretí je 0,2108. [32]

Funkčný predpis Cross-entropy loss je:

$$L_{CE} = -\sum_{i=1}^{n} t_i \log_2(p_i) \tag{1.10}$$

- LCE je hodnota chybovej funkcie
- t_i je požadovaný výstup (0 alebo 1)
- p_i je výstup z modelu (na príklad výstup z funkcie Softmax)

Keďže t_i môže nadobúdať iba hodnoty 0 a 1, funkčný predpis sa pre prípady zaradenia objektov len do jednej kategórie dá zjednodušiť a výsledkom bude logaritmus hodnoty pravdepodobnosti zaradenia do správnej kategórie. [32]

1.4.11 Klasifikácia obrazu

Klasifikácia obrazu je úloha, kedy model priradí obrazu jeden alebo viac štítkov, ktoré označujú čo je na obrázku, teda do ktorej triedy obrázok patrí. Výstupom z klasifikačnej siete je vektor, ktorého veľkosť je taká istá, ako počet tried. Hodnoty jednotlivých súradníc výstupného vektora označujú pravdepodobnosť, s akou obrázok patrí do danej triedy. Výsledný vektor je vstupom do aktivačnej funkcie Softmax.

Pre klasifikáciu obrazu potrebujeme nazbierať veľké množstvo dát. Obyčajne sa dataset skladá z obrazov a názvov tried s tým, že každý obrázok môže byť priradený do jednej alebo viacerých tried. Veľmi dôležité je to, aby obrázky boli vyvážené (napr. aby boli fotené za každého počasia, aby sa sieť neučila podľa toho, či fotografia zachytáva slnečný deň alebo deň, počas ktorého prší). [17]

1.4.12 Kroky potrebné pre klasifikáciu obrazu

- Predspracovanie obrazu: Cieľom tohto kroku je upraviť obraz tak, že potlačíme nevhodné vlastnosti obrazu (šum) a zvýrazníme niektoré dôležité

vlastnosti obrazu. Predspracovanie pozostáva z načítania obrazu, zmeny veľkosti obrazu a rôznych operácií s obrazom, ako na príklad translácia a rotácia.

- Detekcia objektu: pre lokalizáciu objektu treba obrázok segmentovať a nájsť pozíciu objektu záujmu.
- Extrakcia vlastností a tréning: hľadajú sa vzory obrazu alebo vlastnosti, ktoré má len určitá trieda. Toto neskôr pomôže modelu rozhodnúť, do ktorej triedy obrázok patrí.
- Klasifikácia objektu: Kategorizuje detegované objekty do preddefinovaných tried. [33]

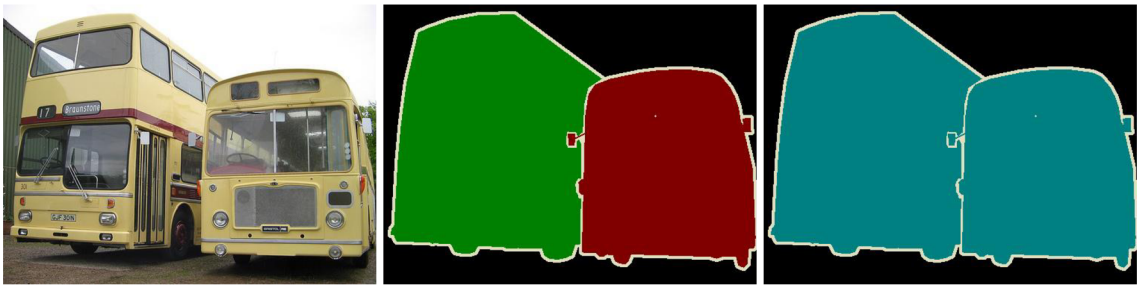
Príklad klasifikácie obrazu ukazuje obrázok 1.13.

Obrázok 1.13 Príklad klasifikácie obrázkov (toto je s jedným štítkom)

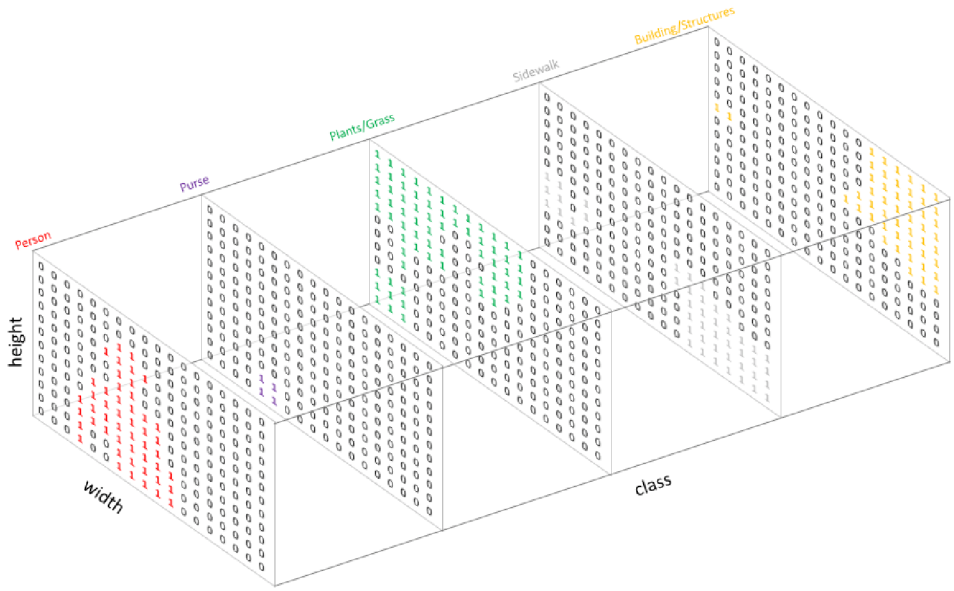
1.5 Sémantická segmentácia

Sémantická segmentácia je úloha kedy model priradí každý pixel obrázku nejakej triede, čo ukazuje obrázok 1.14. Pomocou nej vie počítač zistiť čo je na obrázku, a kde na obrázku sa to nachádza. Cieľom je zmeniť vstupný obrázok tak, aby farby pixelov výstupného obrázku prislúchali konkrétnej triede, do ktorej patria objekty na obrázku. Výstupom modelu pre klasifikáciu obrazu je vektor, ktorého zložky hovoria o tom, s akou pravdepodobnosť patrí objekt na obrázku konkrétnej triede. Pri sémantickej

segmentácií je však tento vektor potrebný pre každý jeden pixel. To znamená, že výstupom nie je matica (teda 2D tenzor), ale 3D tenzor, v ktorom následne model hľadá maximá v každej súradnici obrazu. [34] Sémantická segmentácia prebieha tak, že najskôr sa vyznačia na obrázku všetky objekty patriace do jednej kategórie. Toto sa spraví toľkokrát, koľko rôznych tried model rozpoznáva. Následne sa výsledky poskladajú do jedného obrazu. (obrázok 1.15) Aktuálne sa pre sémantickú segmentáciu využíva hlboké učenie. [35] [36]



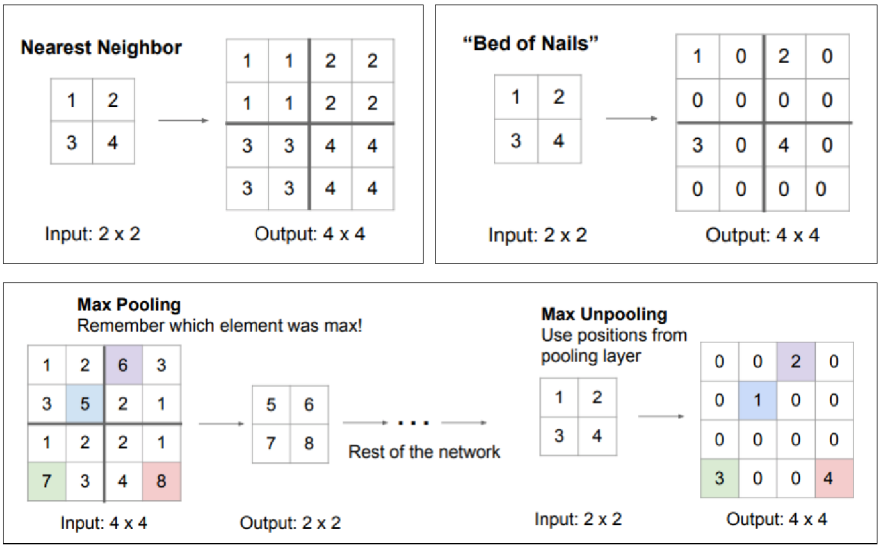
Obrázok 1.14 Príklad inštančnej (v strede) a sémantickej (vpravo) segmentácie [34]



Obrázok 1.15 Sémantická segmentácia viacerých tried a ich skladanie do jednoho obrazu [35]

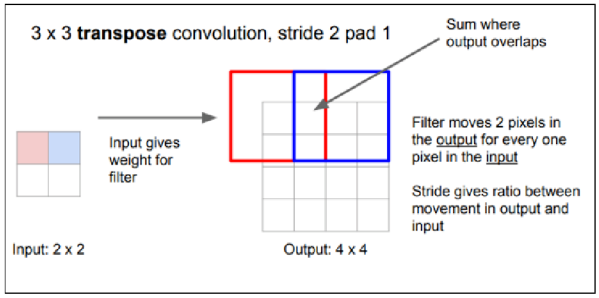
Architektúra modelu môže byť taká, že sa spojí niekoľko konvolučných vrstiev za seba, avšak problémom je veľká výpočtová náročnosť. Pri klasifikácií, čím hlbšia je konvolučná sieť, tým viac špecifické vlastnosti objektov skúma. Problém pri sémantickej segmentácií je ten, že čím hlbšia je konvolučná sieť, tým menší je obrázok kvôli operácii pooling. Toto pri obyčajnej klasifikácií nevadí, pretože je podstatné iba to, čo je na obrázku a nie to, kde sa to nachádza. Preto je potrebné na koniec obrázok

zasa zväčšiť a na to sa používa opačné poolovanie: V ľavej hornej časti obrázka 1.16 vidno prvý typ – z matice 2x2 sa spraví matica 4x4 takým spôsobom, že vo výslednej matici budú po rozdelení na 4 rovnaké štvorce v celom štvorci rovnaké čísla – také ako v pôvodnej matici 2x2. Ďalší typ je vidno vpravo hore, kde číslo z pôvodnej matice sa umiestni do ľavého horného rohu štvorca novej matice (nová matica je rozdelená na 4 štvorce tak isto ako v prvom type). Tretí typ si pamätá ako bol obrázok zmenšený, konkrétne súradnicu, odkiaľ sa vzala hodnota do zmenšenej matice. Do tejto súradnice vloží hodnotu pri opačnom poolovaní. [35]



Obrázok 1.16 Opačné poolovanie [35]

Najpopulárnejší je však prístup pomocou transponovanej konvolúcie (nazývanej aj dekonvolúcia, ale to nie je matematicky presné, pretože dekonvolúcia označuje niečo iné), čo ukazuje obrázok 1.17: Model si vezmem hodnotu prvého elementu vstupnej matice a vloží ho do prvých 4 políčok výstupnej matice. Toto isté spraví s druhým elementom. Tam kde sa hodnoty prekrývajú, tam ich sčíta. [35]



Obrázok 1.17 Transponovaná konvolúcia [35]

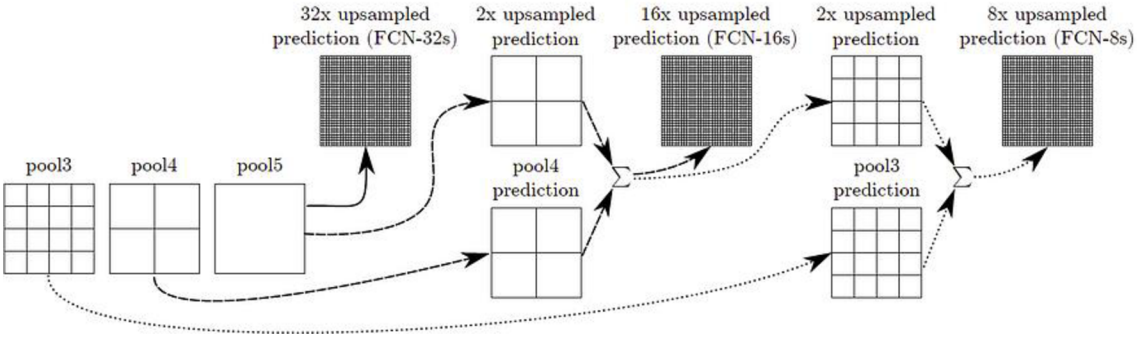
1.5.1 Plná konvolučná sieť

Plná konvolučná sieť (Fully convolutional network) sa od konvolučnej neurónovej siete

líši tým, že v architektúre na konci vynecháva neurónovú sieť a namiesto nej je tam konvolúcia 1x1, ktorá prejde celý región obrazu. To má výhodu v tom, že model môže pracovať s akoukoľvek veľkosťou obrázka. Ďalej, výstupom siete je mapa vlastností a nie vektor tried, z ktorého sa následne zisťuje do ktorej triedy patrí objekt na obrázku. Na tejto mape vlastností vidno pozíciu požadovaného objektu. [37]

Z dôvodu poolovania je výsledná mapa vlastností zmenšená oproti obrázku na vstupu a preto treba výsledok opäť zväčšiť. Na to je možné použiť bilineárnu interpoláciu (interpolácia funkcie dvoch premenných), alebo, ako autori siete odporúčajú, naučené zväčšenie obrazu pomocou transponovanej konvolúcie. [37]

Časť modelu, ktorá zmenšuje obrázok, sa volá enkóder a tá, ktorá ho zväčšuje sa volá dekóder. Enkóder zmenšuje obraz 32 násobne. Následne je teda potrebné obraz 32 násobne zväčšiť. Ak sa zväčšuje iba pomocou transponovanej konvolúcie, jedná sa o FCN-32s. Výsledky tejto siete sú hrubé, nepresné, strácajú sa jemné detaily. Ak sa však sčíta zväčšený výsledok z poslednej poolovacej vrstvy (pomocou opačného poolovania) s výsledkom z predposlednej poolovacej vrstvy už model pracuje iba so 16-krát zmenšeným obrázkom (FCN-16s). Podobne sa dá získať obrázok 8-krát zmenšený, vtedy ide o model FCN-8s (Obrázok 1.18). Na tieto výsledné obrázky sa následne aplikuje transponovaná konvolúcia. [37]



Obrázok 1.18 Zväčšovanie obrázka pomocou opačného poolovania [37]

1.5.2 U-net

Zväčšovanie výsledku v U-nete funguje tak, že každá vrstva dekóderu získava informácie z korešpondujúcej vrstvy enkóderu. Preto má U-net jemnejšie výsledky s menšími výpočtovými nárokmi. Architektúru U-netu ukazuje obrázok 1.19. [36]

1.5.3 Hodnotenie výsledkov

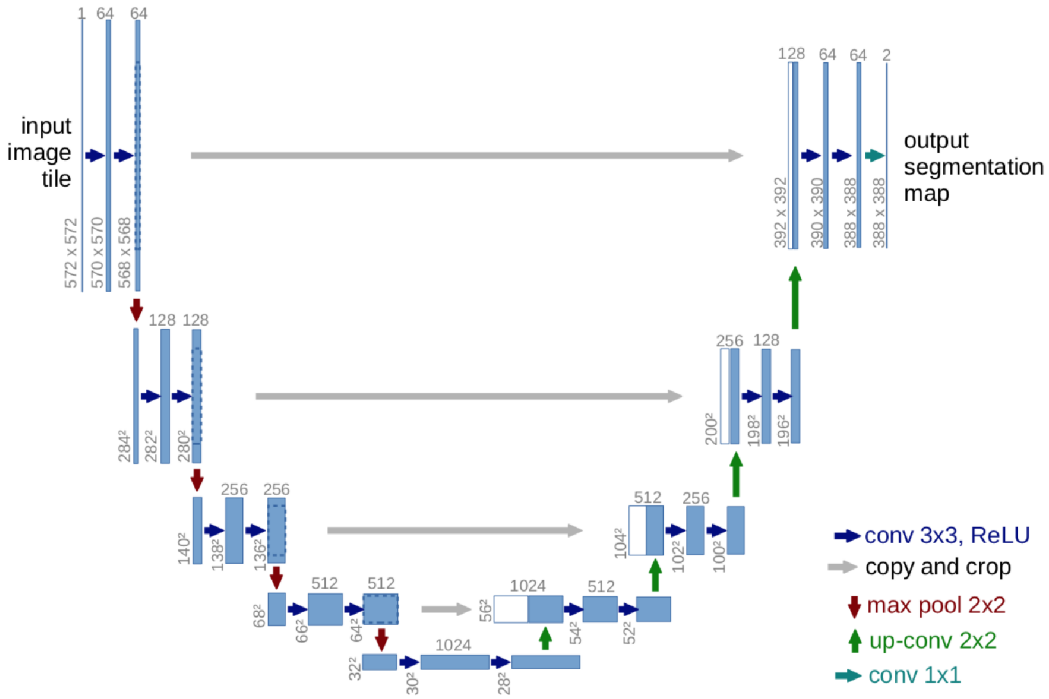
Presnosť pixelov

Výstupom z tejto funkcie je percento pixelov, ktorým model správne určil kategóriu. Počíta sa pre každú triedu samostatne ale aj globálne pre všetky triedy. Na výstup z naučeného modelu sa dá pozerať ako na maticu, ktorej prvky tvoria jednotky a nuly. Jednotky prislúchajú správne určeným pixelom a nuly nesprávne určeným pixelom. Pre

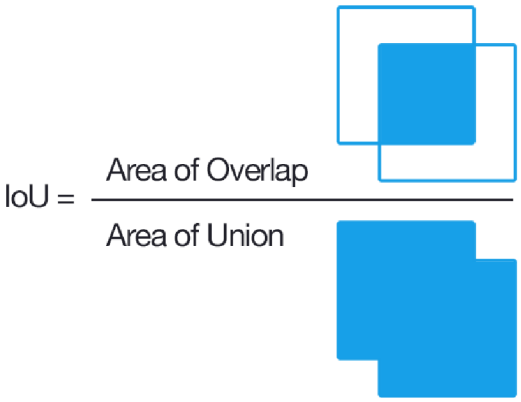
jednu kategóriu vyzerá predpis funkcie takto:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.11}$$

Kde T znamená správne, F nesprávne, P pozitívne (teda v jednotke), N negatívne (teda v nule). Výsledná hodnota je teda rovná podielu počtu správne zaradených pixelov (počtu jednotiek) a počtu všetkých pixelov (počtu jednotiek aj núl spolu). Problémom tejto metódy je to, že ak by na vstupe modelu bol obrázok, kde 90% tvorí pozadie a zvyšných 10% nejaká iná kategória a výstupom by bol jednofarebný obrázok s farbou pozadia, model by mal stále 90% presnosť. [36]



Obrázok 1.19 Architektúra U-net [36]



Obrázok 1.20 Architektúra U-net [36]

IOU (Intersection over union)

Jedná sa o pomer medzi plochou, kde sa oštítkovaný obraz zhoduje s predikovaným obrazom z modelu a celkovou plochou (obrázok 1.20). Pri segmentácií do viac kategórií sa IOU vypočíta pre všetky kategórie a následne sa vypočíta ich priemer. Pre porovnanie s príkladom vyššie, IOU vychádza 45%. [36]

$$IoU = \frac{Plocha\ prekrytia}{Plocha\ bunky} \tag{1.12}$$

Vylepšením IOU je IOU s váženým výskytom, kde sa pri výpočte berie do úvahy to, aký častý je výskyt kategórie v obrázkoch: čím častejší, tým má pri výpočte menšiu váhu. [36]

1.5.4 Funkcie popisujúce chybu siete (Loss)

Pixel-wise cross entropy loss

Vezme si jeden pixel a skúma jeho hodnoty vo všetkých segmentačných triedach. Toto spraví so všetkými pixelmi. [36]

$$L = -\sum y_{true} \cdot \log\left(y_{pred}\right) \tag{1.13}$$

Nedostatkom funkcie je to, že s každým pixelom na obrázku počíta rovnako. To môže spôsobovať problémy, ak máme nerovnaký počet vzorov v jednotlivých triedach. Možné riešenie je také, že pre pixely, ktoré zobrazujú hranice objektov priradíme väčšiu váhu pri určovaní chyby (aby pri určovaní chyby robilo väčší problém to, že nesedia hrany obrázku ako, to že model zle predikuje pixel v strede obrazu). [36]

Diceho funkcia

Diceho funkcia počíta chybu siete pomocou toho, ako veľmi sa výstup siete podobá pravdivému výsledku. [38]

$$Dice = \frac{2 \cdot |A \cap B|}{|A| + |B|} \tag{1.14}$$

Prienik v čitateli sa počíta tak, že sa vezme matica výstupu siete A a oštítkovaná matica B, elementárne sa vynásobia a prvky sa sčítajú. V menovateli sa sčítajú všetky prvky matice A a matice B a výsledky sa sčítajú. Model je natrénovaný dokonale ak sa Dice rovná jednej. [38]

$$|A \cap B| = \underbrace{\begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix}}_{\text{prediction}} * \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{target}} \xrightarrow{\text{element-wise multiply}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

Obrázok 1.21 Diceho Loss funkcia [38]

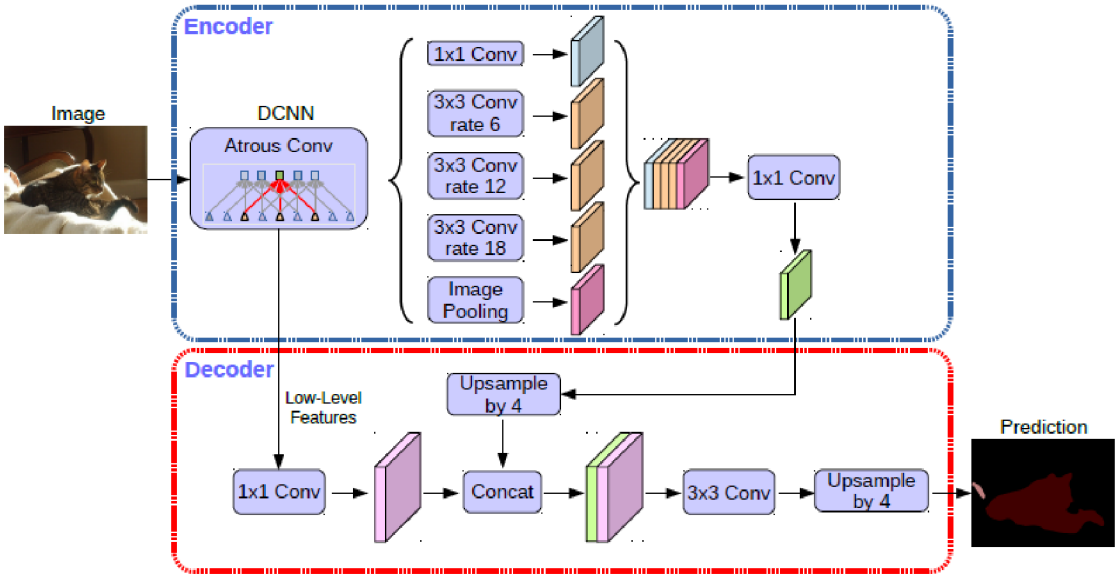
1.6 DeepLab

Deeplab je model plne konvolučnej neurónovej siete, ktorý vymysleli vo firme Google. Rieši sémantickú segmentáciu a má niekoľko verzií. Výhody DeepLabu spočívajú v tom, že je rýchly, presný, má jednoduchú architektúru a dá sa použiť na akúkoľvek úlohu. Z dôvodu poolovania a konvolúcií, pri ktorých sa kernel po obrázku posúva o viac ako jeden pixel, klesá do hĺbky siete veľkosť obrázkov. Ďalším problémom je to, že na obrázku sa nachádzajú objekty v rôznych veľkostiach. Väčšie objekty sa stávajú dominantné a menšie sú potláčané. Na zmiernenie týchto problémov sa používa:

1. Dilatačná konvolúcia
2. Dilatačné priestorové pyramídové poolovanie (pre verzie DeepLabv3 a viac)

Verzie DeepLabv1 a v2 používali namiesto dilatačného priestorového pyramídového poolovania podmienkové náhodné oblasti (Conditional Random Fields). [39]

Súčasťou modelu Deeplabu je časť nazývaná kostra (backbone). Úlohou tejto časti je vyextrahovať z obrázku vlastnosti, s ktorými ďalej Deeplab pracuje. Ako kostra sa používa MobileNetv2, Xception, ResNet-v1, PNASNet. Architektúru modelu ukazuje obrázok 1.22. [40]

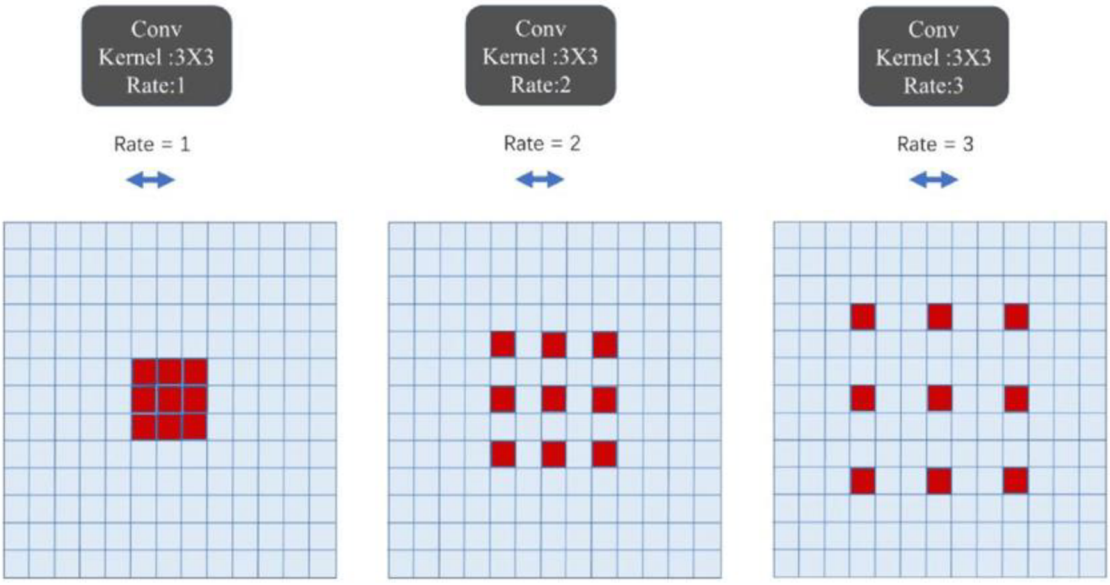


Obrázok 1.22 Architektúra modelu DeepLabv3+ [40]

1.6.1 Dilatačná konvolúcia

Dilatačná konvolúcia (Atrous/dilated/hole convolution) je špeciálna v tom, že do kernelu sú medzi jednotlivé hodnoty vkladané medzery, čiže nuly. Trik je v tom, že čokoľvek sa vynásobí nulou bude 0, takže to počítať netreba a tým sa šetrí výpočtový výkon a čas. To, koľko núl sa vloží medzi hodnoty určuje parameter dilatation rate. Na príklad kernel 3x3 s dilatation rate = 2 má takú oblasť videnia (field of view) ako

kernel 5x5, ale má len 9 parametrov. [41] Vidno to na obrázku 1.23, kde svetlomodrá matica je vstup do siete, červené políčka označujú kernel. Štandardná konvolúcia je teda špeciálny typ dilatačnej konvolúcie, kedy parameter rate = 1. [34]



Obrázok 1.23 Dilatačná konvolúcia [42]

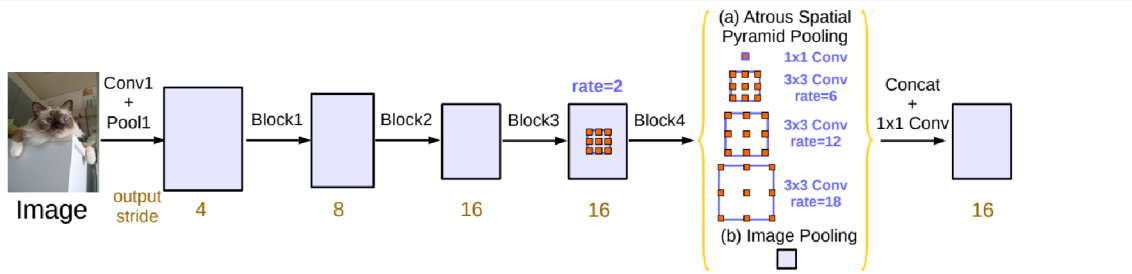
Parameter rate treba voliť s rozumom. Ak sa používa kernel veľkosti 3x3 a rate sa zvolí (v extrémnom prípade) na veľkosť obrázka, tak už zrazu má model kernel veľkosti 1x1, pretože oblasť videnia je násobne väčšia ako vstupný obrázok a krajné hodnoty kernelu sa násobia s obálkou obrázku, ktorá je väčšinou 0. [34]

Architektúra DeepLab vynecháva v kostre posledné poolovacie vrstvy a preto sa vstupný obrázok zmenší iba 8 alebo 16-krát. Následne sa používa dilatačná konvolúcia, čo má podobný efekt ako keby sa použila poolovacia a za ňou konvolučná vrstva (obrázok 1.24). Pre porovnanie výstupu modelu s oštítkovaným obrazom sa pri tréningu oštítkovaný obrázok tiež zmenší 8 alebo 16 násobne. Pre zväčšenie výstupu sa používa bilineárna interpolácia. [43]

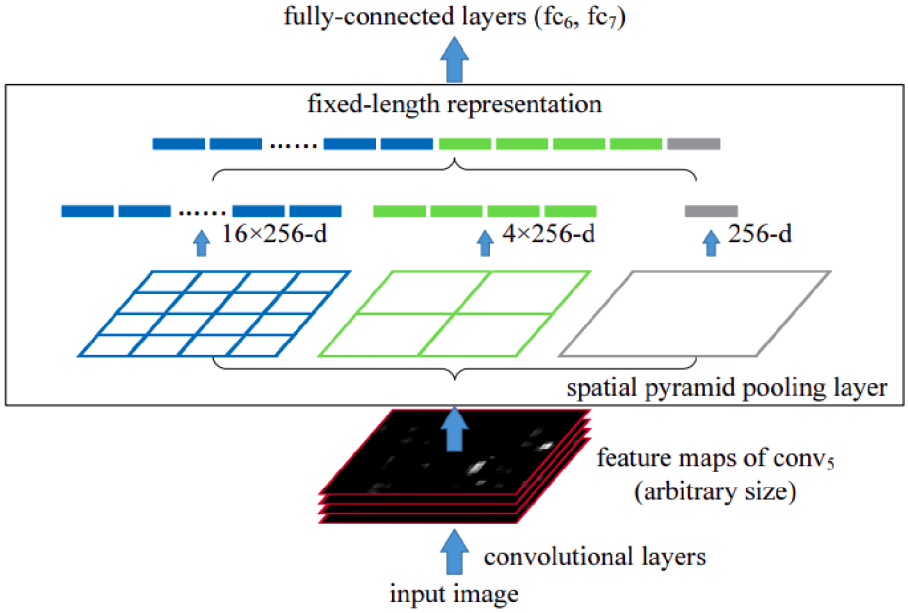
1.6.2 (Dilatačné) Priestorové pyramídové poolovanie

Konvolučná neurónová sieť obsahuje na konci plne prepojenú vrstvu, ktorá požaduje, aby na jej vstupe bol vždy rovnaký počet parametrov. Pri klasickej konvolučnej časti siete to znamená, že je potrebné mať vždy rovnaké rozmery obrázkov. Priestorové pyramídové poolovanie sa ukladá na miesto poslednej poolovacej časti (obrázok 1.24) a jej výstup má vždy rovnaký počet hodnôt (vždy je to vektor s rovnakými rozmermi). Na výstupe konvolučnej časti siete je niekoľko máp vlastností, obrázok 1.25 ukazuje na príklad 256. Priestorové pyramídové poolovanie funguje tak, že každá priestorová mapa je poolovaná do 1 hodnoty, do 4 hodnôt a do 16 hodnôt.

Takýmto spôsobom vie model spracovať objekty, ktoré sú na obrázku v rôznych veľkostiach. [44]



Obrázok 1.24 Nahradenie poslednej poolovacej vrstvy [43]



Obrázok 1.25 Priestorové pyramídové poolovanie [45]

Dilatačná konvolúcia prebieha paralelne s rôznymi parametrami rate. Dilatačné priestorové pyramídové poolovanie zabezpečuje to, aby sa rôzne výsledky z jednotlivých konvolúcií zlúčili dohromady.

1.6.3 Dekóder DeepLabu

Enkóder zmenší obrázok 8 alebo 16-krát. Tieto mapy vlastností ďalej pokračujú do dilatačných konvolúcií s dilatačným priestorovým pyramídovým poolovaním a do dekódera. Za poolovaním sa na mapy vlastností aplikuje konvolúcia 1x1 a štvornásobne sa zväčšia. V dekóderi sa na tie isté mapy vlastností aplikuje taktiež konvolúcia 1x1 a následne sa tieto mapy spoja. Ďalej pokračuje konvolúcia 3x3 a bilineárne zväčšenie obrazu na pôvodnú veľkosť. Ukazuje to obrázok 1.22. [39]

1.7 ROS 2

ROS 2 je súhrn softvérových knižníc a nástrojov na robotické aplikácie. Úlohou ROS 2 je prepojiť jednoduché programy do väčšieho celku. ROS sa skladá z mnoho častí, niektoré z nich sú popísané v nasledujúcom texte. [46]

1.7.1 Node

Node je základný prvok ROS 2. Každý node by mal mať jednu jednoduchú úlohu, napr. riadenie motora. Nody medzi sebou komunikujú. Nody majú konfiguračné hodnoty, nazývané parametre. Aktuálne hodnoty parametrov dokážeme z nodov zistiť, alebo ich zmeniť. Ak máme na príklad node, ktorý ovláda motor, parametrom môže byť rýchlosť otáčania motora. [46]

1.7.2 Topic

Nody musia medzi sebou nejako komunikovať. Jedným zo spôsobov je využite topicov. Node môže posielať dáta do množstva topicov a zároveň z topicov prijímať správy. [46]

1.7.3 Service

Ďalším zo spôsobov komunikácie je service. Service je založený na tom, že jeden node pošle požiadavku (request) a druhý node na túto požiadavku odpovie. Service nie je vhodné použiť na aplikáciu, ktorá potrebuje stály tok dát. [46]

1.7.4 Action

Action je ďalší typ komunikácie v ROS 2 a je vhodný pre dlhšie komunikácie. Využíva model klient-server. Pozostáva z troch častí: goal, feedback a result. Na rozdiel od servicov, môžu byť zrušené. Klientsky node pošle goal nodu, ktorý plní funkciu servera. Server sa oboznámi s goalom a klientovi vráti tok feedbacku a result. [46]

1.7.5 Bag

Bag je nástroj na nahrávanie dát publikovaných v topicu. Neskôr je možné si dáta prehrať a robiť s nimi experimenty alebo sa môžu tieto dáta niekomu poskytnúť aby s nimi experimentoval niekto iný. [46]

2. TESTOVANIE SÉMANTICKEJ SEGMENTÁCIE

Pre experimentovanie sú potrebné najskôr dáta, na ktorých môžem odskúšať naučený model. Na to potrebujem mať dataset.

2.1 Vytvorenie datasetu

Dataset sme vytvorili spolu s vedúcim práce pomocou robotickej platformy vyvíjanej na Ústave automatizace a měřící techniky (Obrázok 2.1). Na robotovi bola pripevnená kamera Intel Realsense a dva lidary (dáta z lidarov nie sú pre túto prácu potrebné). Všetky 3 snímače ukladali dáta do rosbag-u.

Obrázok 2.1 Robotická platforma pomocou ktorej sme s vedúcim vytvorili dataset

2.1.1 Parametre kamery

Použitá kamera Intel RealSense D435 má ideálny rozsah od 30 cm do 3 m. Zachytáva RGB obraz aj hĺbkový. RGB časť má rozlíšenie do 1920x1080 (2 MP), rýchlosť má do 30 fps. [47]

2.1.2 Tvorba datasetu

Dataset sme spolu s vedúcim práce vytvorili dňa 12.12.2022 o 9:52 s dĺžkou asi 5 minút. Počasie bolo ideálne, polooblačné, bez snehu. Natočili sme cestu, chodník zo zámkovej dlažby, štrkový chodník, skaly, trávu, listy, kopec, stromy, ľudí, oblohu, budovy, lampy.

2.1.3 Operácie s datasetom

Ako prvé je treba nejako dostať potrebné dáta z rosbag-u. To som spravil použitím skriptu [48] Výstupom je 7325 png obrázkov s rozlíšením 640x480 pixelov. Jeden z obrázkov ukazuje obrázok 2.2.

Obrázok 2.2 Obrázok z vytvoreného datasetu

2.1.4 Konvolúcia obrázkov z datasetu

Na obrázky som vyskúšal aplikovať konvolúciu s konvolučným jadrom:

$$h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (2.1)$$

Tieto hodnoty sú rovnaké pre všetky tri farebné zložky. Jedná sa o Laplacov filter (druhá derivácia funkcie jasu), ktorý kombinuje potlačenie šumu s detekciou hrán. Konvolúcia je v konvolučných neurónových sieťach kľúčová funkcia, pretože slúži na

extrahovanie vlastností z obrazu. Obrázok z datasetu s jeho konvolúciou ukazuje obrázok 2.3.

Obrázok 2.3 Príklad obrázku a jeho konvolúcie (výsledok konvolúcie som invertoval)

2.2 Experimentovanie na semestrálnu prácu

Hlavnou úlohou semestrálnej práce, na ktorú nadväzuje táto práca, bolo vyskúšať rozbehnúť model sémantickej segmentácie prakticky s hocijakým datasetom. S vedúcim práce sme sa zhodli na tom, že najlepší model pre túto úlohu bude DeepLab. Ako odrazový mostík som si stiahol skripty zo zdroja [49] a posťahoval som všetky potrebné knižnice a ich verzie (viac v nasledujúcej kapitole). Najväčší problém predstavoval čas potrebný na tréning siete v kombinácii s mojou neskúsenosťou v počítačovom videní. Prvý pokus tréningu som rozbehal vo VMware na Ubuntu 22.04. Jedna epocha trvala približne 11 hodín. Druhý pokus som skúšal na samotnom Windows 10 so 16 GB RAM a 6 jadrami, čo trvalo asi 8 hodín na cyklus. Ďalej som skúšal Google Colab, ale tam môžem byť pripojený iba 12 hodín, potom ma odpoja (a to v prípade, ak mám zaplatený Google Colab pro). Navyše, jedna epocha trvala aj tak asi 14 hodín. V platforme Google Cloud trval jeden cyklus 7 hodín. Z uvedených časov vyplýva, že treba prejsť na trénovanie pomocou grafickej karty. Napriek všetkému sa nakoniec model podarilo natrénovať. Segmentovaný výsledok ukazuje obrázok 2.4.

Neskôr, keď už sa mi podarilo rozbehnúť tréning na GPU (viac v ďalších kapitolách) len v rámci pokusov, jedna epocha trvala do 25 minút.

Obrázok 2.4 Vstup/výstup do/z naučeného modelu na semestrálnu prácu

2.3 DeepLabv3+

S modelom DeepLabu som sa rozhodol pokračovať aj v tejto práci.

2.3.1 Knižnice, ich verzie a príprava diskového priestoru pre prácu s DeepLab

Podľa návodu [50] treba mať nainštalované tieto knižnice a ich verzie:

- Python 3.7. – programovací jazyk, je dôležité, aby to bola táto verzia a nie novšia kvôli kompatibilite s knižnicou Tensorflow
- Tensorflow 1.15.2 – je to open source platforma pre strojové učenie. Bol vyvinutý tímom v Googli. Môže sa používať s pythonom a C++ [51]
- python-pil (knižnica Pillow) – intepreteru pythonu pridáva schopnosti spracovania obrazu [52]
- python-numpy – Numpy je knižnica pre prácu s poľami v pythone. Obsahuje funkcie pre prácu v oblasti lineárnej algebry, maticami a fourierovej transformácie. Numpy polia sú asi 50-krát rýchlejšie ako originálne zoznamy v pythone. [53]
- tf_slim – knižnica, ktorá sa používa na definovanie, trénovanie a ohodnotenie komplexných modelov. [54]
- Protobuf 3.20
- Jupyter
- Matplotlib – knižnica na kreslenie grafov v pythone

Z oficiálneho github repozitáru tensorflow preberiem celý priečinok models :

```
git clone https://github.com/tensorflow/models
```

Taktiež treba pridať priečinok research a slim do pythonpath:

```
#From tensorflow/models/research/
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

Po naklonovaní celého priečinka by mal mať pracovný adresár nasledujúcu štruktúru:

- deeplab/datasets
  - rugd
    - VOC2012
      - JPEGImages
      - Segmentation
      - ImageSets
    - tfrecord
    - exp
      - train_on_trainval_set
        - train
        - eval
        - vis
        - export
    - init_models

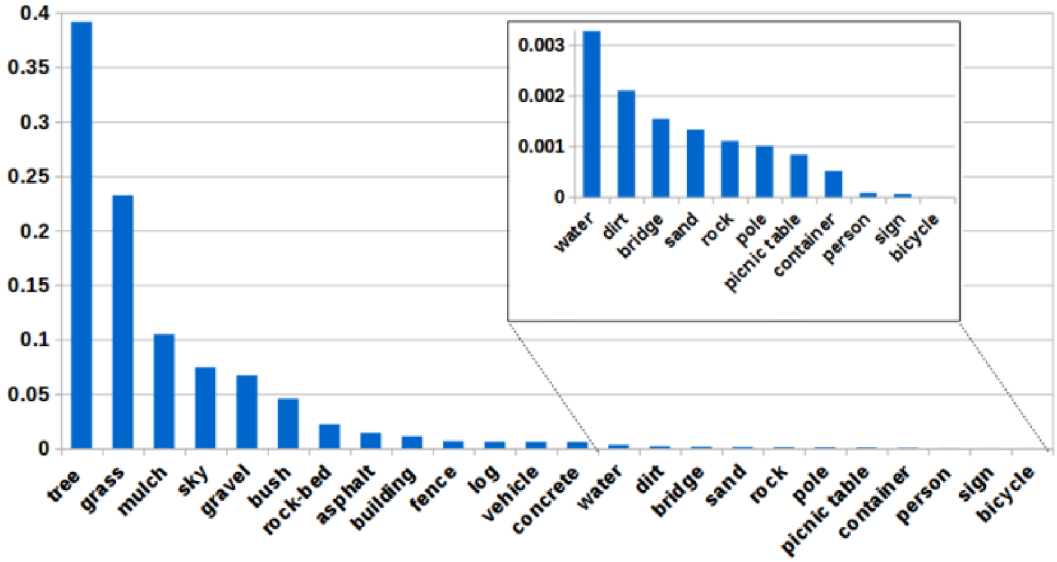
Preto treba do priečinku datasets vytvoriť ďalšie priečinky podľa schémy vyššie. Do priečinku JPEGImages patria originálne, neupravené obrázky. Do Segmentation treba vložiť anotované obrázky (obrázky požadovaných výstupov zo siete). Do ImageSets treba vložiť 3 textové súbory: train.txt, val.txt a trainval.txt. Do súboru train.txt treba vložiť názvy obrázkov bez prípon na ktorých sa bude model učiť (90% všetkých obrázkov). Súbor val.txt obsahuje názvy zvyšných obrázkov a trainval.txt obsahuje názvy všetkých obrázkov bez prípon. Model treba pomocou časti datasetu učiť a pomocou druhej časti validovať aby sa zabránilo preučeniu (treba overiť, či model vie zovšeobecňovať).

2.3.2 Dataset a jeho úprava

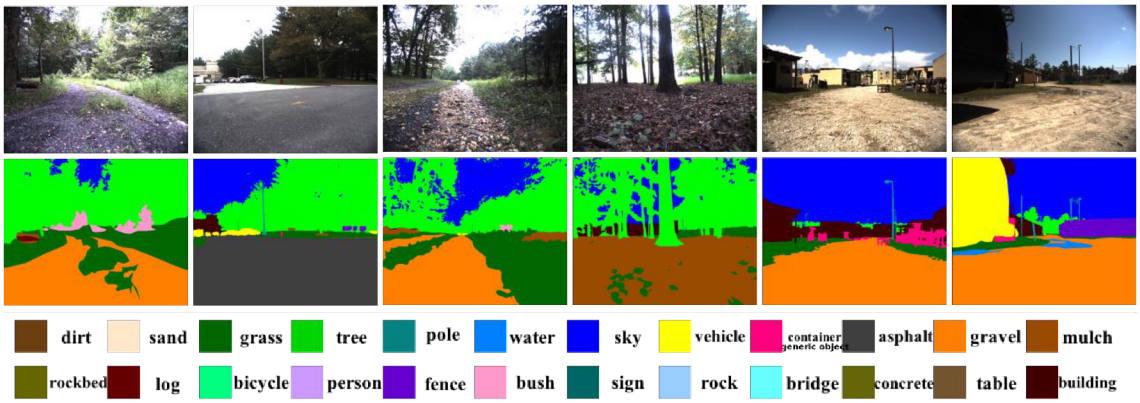
Cieľom práce je to, aby robot, na ktorom bude spustený natrénovaný model, rozpoznával terénne povrchy. Na to treba mať vhodný dataset. Na internete som našiel dataset, ktorý sa hodí priamo na túto úlohu a je ním RUGD: [55] Výskyt kategórií v datasete RUGD ukazuje obrázok 2.5 a ukážku obrázkov ukazuje obrázok 2.6.

Anotované obrázky RUGD-u majú hĺbku 24 bitov (8 bitov pre každú zložku RGB). Preto som ich previedol na 8 bitový farebný formát. Tu si treba dať veľký pozor na to, aby sa preniesla aj paleta farieb (colormap). Najskôr som previedol obrázky pomocou programu v matlabe, ktorý som si napísal, ale paleta sa nepreniesla (obrázky 2.7 a 2.8). Tým pádom sa model netrénoval správne (dôvody popisujem ďalej). Po tom, čo som si túto chybu uvedomil som v programe IrfanView otvoril náhodný anotovaný 8-bitový obrázok a začal upravovať paletu (na hornej lište Obrázok – Paleta – Upraviť paletu).

Výslednú paletu ukazuje obrázok 2.9. Paletu treba následne exportovať a importovať ju do všetkých obrázkov. Na to som použil skript: [56]



Obrázok 2.5 Pravdepodobnostné zastúpenie tried v datasete RUGD [55]

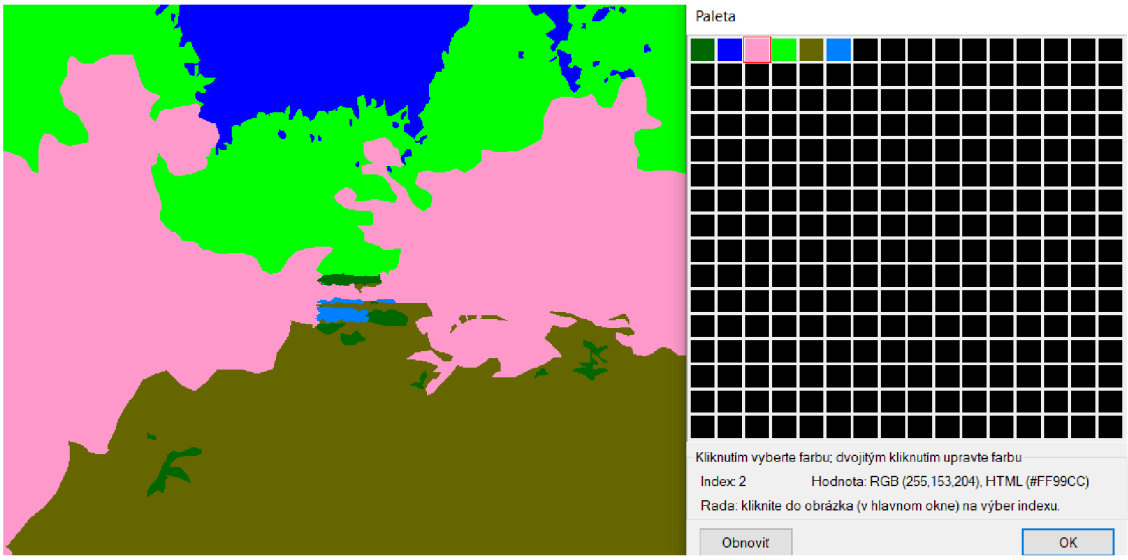


Obrázok 2.6 Pravdepodobnostné zastúpenie tried v datasete RUGD [55]

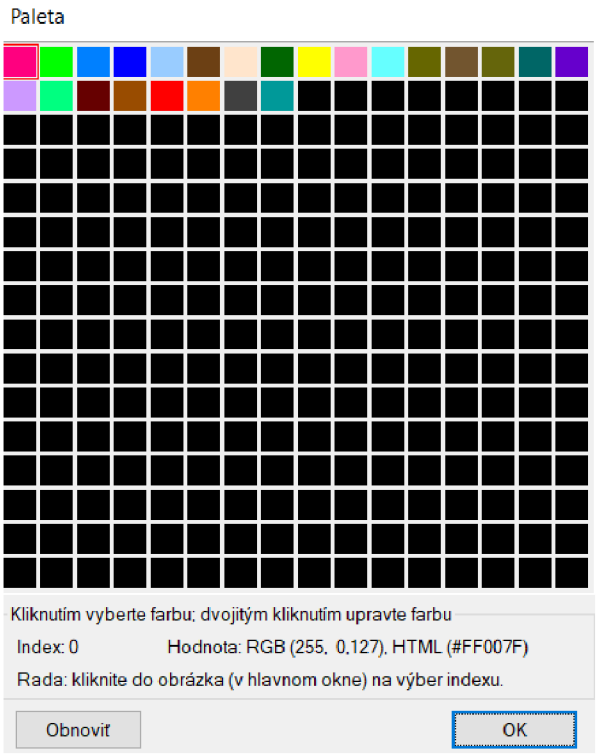
Anotované 8-bitové obrázky treba ďalej previesť na monochromatické obrázky, pričom farba pixelu (0-255) označuje kategóriu, do ktorej pixel patrí. Skript, ktorý som využil, používa na prevod paletu farieb, ktorá prislúcha k danému obrázku. Tým pádom, ak nemá každý obrázok rovnakú paletu, dôjde k tomu, že rovnaké farby na rôznych obrázkoch majú rôzny index. To znamená, že ružová farba, ktorá na všetkých anotovaných obrázkoch predstavuje kríky, má rôzne indexy v rôznych obrázkoch. A to znamená, že na obrázku síce vidím ružovú o ktorej viem, že patrí kríkom, ale program vlastne mieša všetky kategórie dokopy, keď kríkom priradí raz index 2 a inokedy 11, pričom číslo indexu znamená konkrétnu kategóriu. Keďže dataset RUGD delí objekty na obrázkoch do 24 kategórií, histogram monochromatických obrázkov bude mať

rozsah 0 až 23 plus pozadie. Na vytvorenie monochromatických obrázkov som využil [57].

Obrázok 2.7 Príklad anotovaného obrázku v 8-bitovom formáte (ružová má index 11)



Obrázok 2.8 Príklad iného anotovaného obrázku v 8-bitovom formáte (ružová má index 2)



Obrázok 2.9 Paleta, ktorá obsahuje všetky možné farby anotovaných obrázkov

2.3.1 Tréning modelu a jeho parametre

Monochromatické obrázky je treba následne previesť na formát tfrecord. Formát tfrecord konvertuje všetky obrázky do jedného veľkého súboru, kde sú obrázky sekvenčne uložené. Obrázky sú pri práci naraz načítané, čo šetrí čas. (formát popisujú zdroje: [58], [59]). Konverziu do tfrecord spraví skript build_voc2012_data.py. Ďalej treba doplniť informácie (počet obrázkov na trénovanie, validáciu a ich súčet) o RUGD datasete do súboru data_generator.py:

```
RUGD_info = DatasetDescriptor(
    splits_to_sizes={
        'train': 3149,
        'trainval':3498,
        'val': 349,
    },
    num_classes=25,
    ignore_label=255,
)

_DATASETS_INFORMATION = {
    'cityscapes': _CITYSCAPES_INFORMATION,
    'pascal_voc_seg': _PASCAL_VOC_SEG_INFORMATION,
    'ade20k': _ADE20K_INFORMATION,
    'RUGD': RUGD_info,
}
```

Následne treba stiahnuť pred trénovaný model a umiestniť ho do priečinka init_models. Teraz môže začať tréning (skript train.py). Trénoval som s týmito parametrami:

```
python "%WORK_DIR%\train.py" ^
--logtostderr ^
--train_split="train" ^
--model_variant="xception_65" ^
--atrous_rates=6 ^
--atrous_rates=12 ^
--atrous_rates=18 ^
--output_stride=16 ^
--decoder_output_stride=4 ^
--train_crop_size="300,300" ^
--train_batch_size=2 ^
--training_number_of_steps=75000 ^
--initialize_last_layer=False ^
--last_layers_contain_logits_only=True ^
--fine_tune_batch_norm=False ^
--train_logdir="%TRAIN_LOGDIR%" ^
--dataset=RUGD ^
--dataset_dir="%RUGD%" ^
--tf_initial_checkpoint=
"%INIT_FOLDER%\deeplabv3_pascal_train_aug\model.ckpt"^
```

Jednotlivé parametre znamenajú:

- train_logdir – cesta k natrénovanému modelu
- dataset_dir – cesta k obrázkom vo formáte tfrecord
- tf_initial_checkpoint – cesta k pred trénovanému modelu
- model_variant – kostra, na ktorej sa bude model učiť
- atrous_rates – parametre rate dilatačnej konvolúcie, ktoré priamo súvisia s:
- output_stride – koľkokrát má byť vstup zmenšený enkóderom
- train_crop_size – veľkosť orezaného obrázku počas tréningu
- train_batch_size – počet obrázkov, s ktorou model pracuje (výpočet chybovej funkcie a dávkovej normalizácie)
- training_number_of_steps – počet epoch
- train_split – na tréning sa použijú obrázky v súbore train.txt
- initialize_last_layer – rozhoduje o tom, či model využije všetky natrénované váhy pred trénovaného modelu
- last_layer_contain_logits_only – rozhoduje o tom, či sa bude v poslednej vrstve používať aktivačná funkcia
- fine_tune_batch_norm – rozhoduje o tom, či sa budú počas tréningu meniť parametre dávkovej normalizácie

Ďalšie parametre (aké časté majú byť výpisy na obrazovku, ako často sa má model ukladať, ...) som nenastavoval ale sú napísané v súbore train.py a dajú sa nastaviť. Pri nastavovaní train_crop_size a train_batch_size si treba dať pozor, aby sa alokovaná pamäť potrebná na tréning zmestila do operačnej pamäte grafickej karty. Kostra modelu môže byť Xception, MobileNetv2, ResNet-v1, PNASNet a Auto-DeepLab.

Po tom čo skript skončí je možné ho opäť spustiť s rovnakými parametrami a väčším číslom pri parametri training_number_of_steps. Model bude pokračovať v učení, nepôjde opäť od začiatku. Výhodou je, že skript si ukladá medzivýsledky, takže ak by sa udialo niečo, kvôli čomu sa nemôže skript dokončiť (na príklad výpadok elektriny), tak sa dá pokračovať z medzivýsledku.

2.3.2 Zobrazenie výsledku a validácia

Pre správne priradenie farieb jednotlivým kategóriám treba upraviť súbor vis.py, ďalej súbory v zložke utils save_annotation.py a get_dataset_colormap.py. Toto nebolo problematické, treba to iba zapísať rovnako ako sú zapísané farebné mapy pre iné datasety. Pre zobrazenie výsledkov treba pustiť skript vis.py s týmito parametrami:

```
python "%WORK_DIR%\vis.py" ^
--logtostderr ^
--vis_split="val" ^
--model_variant="xception_65" ^
--atrous_rates=6 ^
--atrous_rates=12 ^
--atrous_rates=18 ^
--output_stride=16 ^
--decoder_output_stride=4 ^
--vis_crop_size="300,300" ^
--checkpoint_dir="%TRAIN_LOGDIR%" ^
--vis_logdir="%VIS_LOGDIR%" ^
--dataset=RUGD ^
--dataset_dir="%RUGD%" ^
```

Parametre znamenajú to isté, čo aj v skripte train.py. Podobne sa spúšťa aj eval.py.

Keď som bol spokojný (po 75000 epochách) s výsledkami aké vyšli z eval.py a vis.py, naučený model som exportoval pomocou skriptu export_model.py. Tento vyexportovaný model som následne používal v upravenom súbore deeplab_demo.ipynb, v ktorom sa dá na vstup modelu poslať hocijaký obrázok, ktorý mi model segmentuje. Hodnoty miou naučeného modelu uvádzam v tabuľke 2.1.

Tabuľka 2.1 Výsledné hodnoty miou naučeného modelu

Kategória	miou
Hlina	0,0028
Piesok	0,3566
Tráva	0,8412
Strom	0,5037
Stĺp	0,0102
Voda	0,6567
Obloha	0,8
Vozidlo	0,6983
Kontajner	0
Asfalt	0,8452
Štrk	0,8078
Budova	0,7763

Kategória	miou
Kôra/kompost	0,8104
Väčšie kamene	0,8258
Kmeň/poleno	0,3683
Bicykel	0
Človek	0
Plot	0,5321
Krík	0,6089
Značka	0
Skala	0,2539
Most	0,6504
Betón	0,8241
Piknikový stôl	0,5085

2.3.3 Porovnanie tréningu na rôznych platformách a GPU

Ako prvé som skúšal trénovať model na mojom notebooku Dell Latitude 7400 s Windows 10, s procesorom Intel i5, integrovanou grafikou a 16 GB RAM. Výsledok bol taký, že po približne 14 hodinách, počas ktorých sa nedalo na notebooku robiť nič iné, pretože bol tak vyťažený, prebehlo asi 80 epoch s približne 3500 obrázkami (V tomto prípade sa počet natrénovaných epoch nedá určiť presne, pretože tréning neskončil tým, že by sa ukončil skript, ale tým, že počítač skončil v modrej obrazovke).

Ďalej som prešiel na platformu od Googlu a to Google colab. Problém s touto platformou je ten, že aj pri platenom účte Google colab pro sa každých 12 hodín ukončí bežiaci program. Tréning na colabe sa mi nepodarilo rozbehnúť s GPU. Podľa výstupov trvalo 50 epoch približne 18 minút, čo je za 24 hodín 4000 epoch. Colab pracoval tiež s 3500 obrázkami. 4000 epoch za 24 hodín je stále príliš málo.

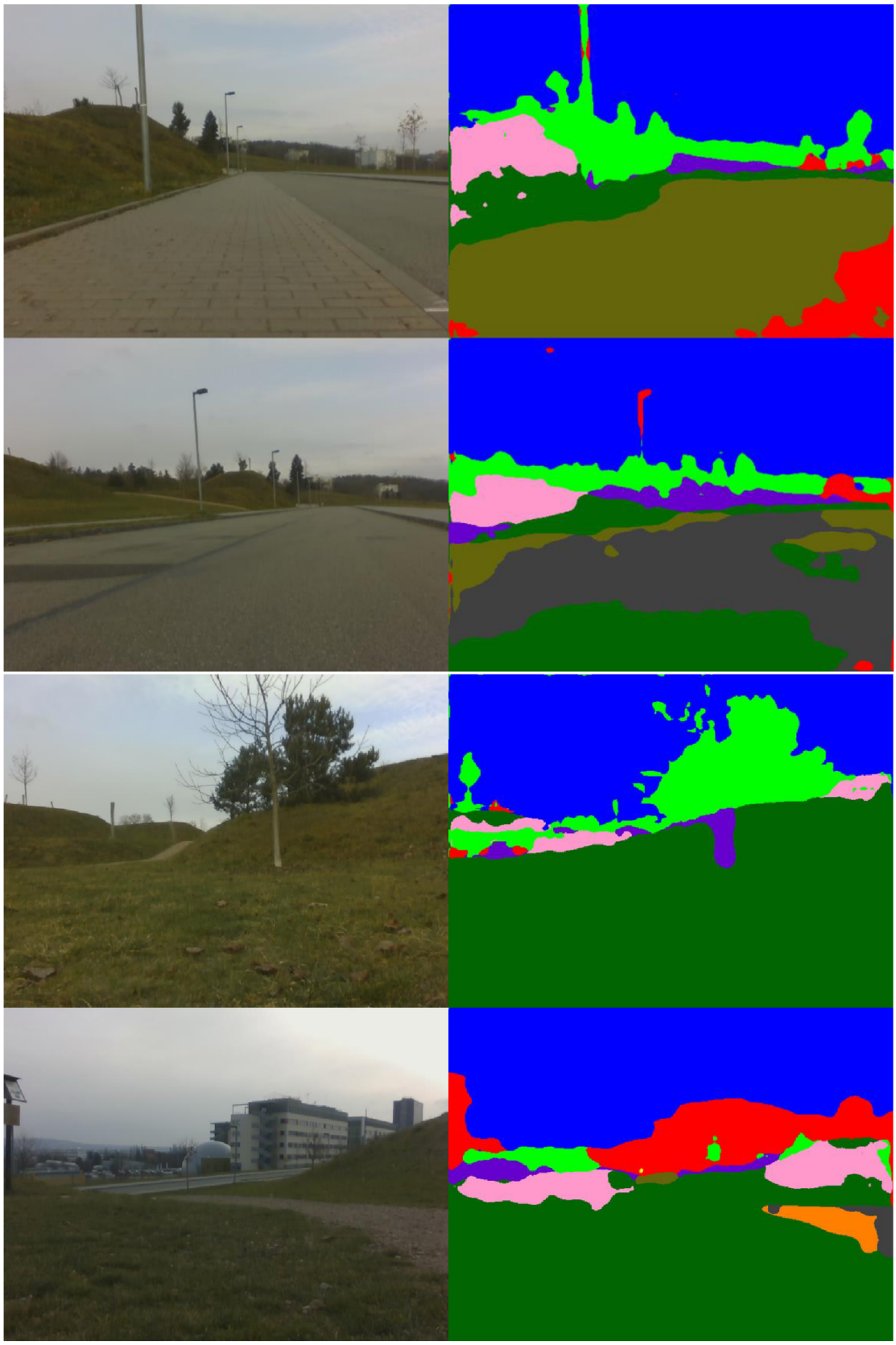
Preto som prešiel na domáci stolový počítač. Na ňom je Windows 10, grafická karta Nvidia GTX 1060 s 3GB RAM. S ohľadom na toto všetko je potrebné mať správne verzie všetkého: python 3.7, tensorflow 1.15, cuda 10.2., cuDNN 7.4 [60]. Na to, aby sa tréning rozbehol na GPU sa musí zmestiť do GPU RAM. Preto som znížil číslo v parametri train_batch_size a train_crop_size som nastavil na 300,300. Potom sa tréning mohol začať. Tréning prebiehal rýchlosťou 10000 epoch za hodinu.

2.3.4 Sémantická segmentácia videa a segmentácia v reálnom čase

Natrénovaný model treba vyskúšať na videu. Na to pôjdem podľa návodu: [61] Ďalej je potrebné odskúšať naučený model na webkamere notebooku: [62]

2.4 Výsledky segmentácie na vlastnom datasete

Naučený model dokáže celkom dobre spracovať aj obrázky z môjho vlastného datasetu, no nie je to dokonalé. Úspešnosť modelu na vytvorenom datasete nedokážem číselne vyjadriť, pretože tento dataset nie je anotovaný. Model vie na datasete celkom dobre rozpoznať trávu, ale na pomerne prudkom kopci si ju mýli s kríkmi. Na určitých záberoch v datasete vidí trávu na miesto asfalt. Dobre segmentuje stromy. Budovy vidí aj tam, kde nie sú. Mýli si asfalt a betón, čo ale nevadí, pretože z hľadiska prejazdnosti majú betón a asfalt rovnaké vlastnosti. Oblohu rozpoznáva dobre. Príklad segmentovaných obrázkov ukazuje obrázok 2.10. Ďalšie obrázky spolu s videom sú mojom GitHube [63].



Obrázok 2.10 Segmentácia na vytvorenom datasete

3. ZÁVER

Cieľ práce sa podarilo splniť. Pre sémantickú segmentáciu bol vybraný model Deeplab, ktorý sa podarilo naučiť s datasetom RUGD. Model sa celkom dobre naučil na tento dataset ale pri podstrčení dát z datasetu vytvoreným spolu s vedúcim práce už model nie je taký dobrý a robí viac chýb. Naučený model sa podarilo rozbehnúť na videu, aj na počítačovej kamere, kde pracuje v reálnom čase. Na mojom školskom notebooku je však segmentácia pomalá z dôvodu, že má iba internú GPU. Dosahuje rýchlosť 0,35 FPS. Na domácom stolovom počítači s GPU však pracuje s rýchlosťou asi 6 FPS. Pre ďalšie zvýšenie rýchlosti by bolo dobré buď mať výkonnejšiu GPU, alebo si zvoliť kostru MobileNetv2, ktorá je vhodná pre mobilné zariadenia (smartfóny, tablety apod.), pretože obsahuje menej parametrov. Vzorové výsledky, segmentáciu videa a kódy zverejňujem na môj GitHub. Pre lepšie výsledky by možno stálo za zváženie vytvoriť dataset s inou kamerou, alebo rozšíriť tréningový dataset o ďalšie obrázky. [63]

LITERATURA

[1] KUNDU, Rohit. Image Processing: Techniques, Types, & Applications [2022]. In: *V7 Labs* [online]. 2022 [cit. 2022-12-28]. Dostupné z: https://www.v7labs.com/blog/image-processing-guide

[2] NISHANT_KUMAR. Digital Image Processing Basics. In: *GeeksforGeeks* [online]. 2022 [cit. 2022-12-27]. Dostupné z: https://www.geeksforgeeks.org/digital-image-processing-basics/

[3] MCHUGH, Sean. DIGITAL CAMERA SENSORS. In: *CAMBRIDGE IN COLOUR* [online]. [cit. 2022-12-21]. Dostupné z: https://www.cambridgeincolour.com/tutorials/camera-sensors.htm

[4] Obrazový snímač. In: NEZNÁMY. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2013, 2022-06-06 [cit. 2022-12-29]. Dostupné z: https://cs.wikipedia.org/wiki/Obrazov%C3%BD_sn%C3%ADma%C4%8D

[5] NEZNÁMY. Color filter array. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017, 2022-12-12 [cit. 2022-12-26]. Dostupné z: https://en.wikipedia.org/wiki/Color_filter_array

[6] NEZNÁMY. Cameras. In: *Opto Engineering* [online]. [cit. 2023-01-09]. Dostupné z: https://www.opto-e.com/en/basics/cameras

[7] LIBERIO. Camera resectioning. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2006, 2022-10-30 [cit. 2023-02-14]. Dostupné z: https://en.wikipedia.org/wiki/Camera_resectioning

[8] ZLIECHOVSKÝ, Richard. JPEG a JPG: Je medzi týmito obrazovými formátmi vôbec nejaký rozdiel?. In: *Vo svete IT* [online]. Zoznam.sk, 2021 [cit. 2023-02-02]. Dostupné z: https://vosveteit.zoznam.sk/jpeg-a-jpg-je-medzi-tymito-obrazovymi-formatmi-vobec-nejaky-rozdiel/

[9] NEZNÁMY. Súbory PNG. In: *Adobe* [online]. [cit. 2023-02-02]. Dostupné z: https://www.adobe.com/sk/creativecloud/file-types/image/raster/png-file.html

[10] NEZNÁMY. Súbory RAW. In: *Adobe* [online]. [cit. 2023-02-02]. Dostupné z: https://www.adobe.com/sk/creativecloud/file-types/image/raw.html

[11] BANDYOPADHYAY, Hmrishav. What Is Computer Vision? [Basic Tasks & Techniques]. In: *V7 Labs* [online]. 2022 [cit. 2022-11-13]. Dostupné z: https://www.v7labs.com/blog/what-is-computer-vision

[12] CACCIOTTI, Niccolò. Computer vision: the ultimate guide on the 4 main tasks. In: *Smart-I* [online]. 2022 [cit. 2023-04-05]. Dostupné z: http://www.smart-interaction.com/2022/07/14/computer-vision-the-ultimate-guide-on-the-4-main-tasks/

[13] SHARMA, Pranshu. Basic Introduction to Convolutional Neural Network in Deep Learning. In: *Analytics Vidhya* [online]. 2022 [cit. 2023-04-27]. Dostupné z: https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/

[14] RICHTER, Miloslav, Karel HORÁK, Ilona JANÁKOVÁ a Petr PETYOVSKÝ. Zpracování vícerozměrných signálů. In: *Machine vision group* [online]. [cit. 2023-04-27]. Dostupné z: http://vision.uamt.feec.vutbr.cz/?course=ZVS

[15] SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. In: *Towards Data Science* [online]. Medium.com, 2018 [cit. 2023-04-09]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[16] BAHETI, Pragati. Activation Functions in Neural Networks [12 Types & Use Cases]. In: *V7 Labs* [online]. 2021 [cit. 2023-04-20]. Dostupné z: https://www.v7labs.com/blog/neural-networks-activation-functions

[17] BANDYOPADHYAY, Hmrishav. Image Classification Explained [+V7 Tutorial]. In: *V7 Labs* [online]. 2021 [cit. 2023-05-09]. Dostupné z: https://www.v7labs.com/blog/image-classification-guide

[18] PANDEY, Atul. Depth-wise Convolution and Depth-wise Separable Convolution. In: *Medium* [online]. 2018 [cit. 2023-04-19]. Dostupné z: https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

[19] MAYANK2498. Depth wise Separable Convolutional Neural Networks. In: *GeeksforGeeks* [online]. 2022 [cit. 2023-04-22]. Dostupné z: https://www.geeksforgeeks.org/depth-wise-separable-convolutional-neural-networks/#article-meta-div

[20] ZHOU, Victor. Training a Convolutional Neural Network from scratch. In: *Towards Data Science* [online]. 2019 [cit. 2023-04-10]. Dostupné z: https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754

[21] JIRSÍK, Václav. *Vícevrstvá neuronová síť - algoritmus učení backpropagation*. 2022.

[22] YADAV, Harsh. Dropout in Neural Networks. In: *Towards Data Science*

[online]. 2022 [cit. 2023-05-10]. Dostupné z: https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9

[23] JEBLAD. Residual neural network. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2023-04-13]. Dostupné z: https://en.wikipedia.org/wiki/Residual_neural_network

[24] NEZNÁMY. ResNet: The Basics and 3 ResNet Extensions. In: *Datagen* [online]. [cit. 2023-04-18]. Dostupné z: https://datagen.tech/guides/computer-vision/resnet/

[25] KAIMING, He. *Deep Residual Learning for Image Recognition* [online]. 2015, 12 [cit. 2023-04-15]. ISSN arXiv:1512.03385v1. Dostupné z: https://arxiv.org/pdf/1512.03385.pdf

[26] BALDHA, Shivam. Introduction to DenseNets (Dense CNN). In: *Analytics Vidhya* [online]. 2022 [cit. 2023-04-15]. Dostupné z: https://www.analyticsvidhya.com/blog/2022/03/introduction-to-densenets-dense-cnn/?fbclid=IwAR1-AbPJOdZYZbUqzBQO1TWos4uDIqjiwALvUZdsPL6lCICECMSOYHTHPGE

[27] JINSOL, KIM. *DenseNet (Densely connected convolution networks)* [online]. In: . 2023 [cit. 2023-04-18]. Dostupné z: https://gaussian37.github.io/dl-concept-densenet/

[28] RIVA, Martin. Batch Normalization in Convolutional Neural Networks. In: *Baeldung* [online]. 2023 [cit. 2023-05-1]. Dostupné z: https://www.baeldung.com/cs/batch-normalization-cnn?fbclid=IwAR3wZiklEW5BBLomXpOUO0AZdfLtEtfhU3Th0gigXQlevKFysWQULrBjbAo

[29] HARGRAVE, MARSHALL. Standard Deviation Formula and Uses vs. Variance. In: *Investopedia* [online]. 2023 [cit. 2023-05-01]. Dostupné z: https://www.investopedia.com/terms/s/standarddeviation.asp

[30] HUBER, Johann. Batch normalization in 3 levels of understanding. In: *Towards Data Science* [online]. 2020 [cit. 2023-05-05]. Dostupné z: https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338

[31] DISHASHREE26 GUPTA. Transfer Learning and the Art of Using Pre-trained Models in Deep Learning. In: *Analytics Vidhya* [online]. 2017 [cit. 2023-05-09]. Dostupné z: https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/

[32] KOECH, Kiprono Elijah. Cross-Entropy Loss Function. In: *Towards Data*

Science [online]. 2020 [cit. 2023-05-10]. Dostupné z: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

[33] SANGHVI, Kavish. Image Classification Techniques. In: *Medium* [online]. 2020 [cit. 2023-05-10]. Dostupné z: https://medium.com/analytics-vidhya/image-classification-techniques-83fd87011cac

[34] THALLES, Silva. Deeplab Image Semantic Segmentation Network. In: *Thalles' blog* [online]. 2018 [cit. 2023-03-10]. Dostupné z: https://sthalles.github.io/deep_segmentation_network/

[35] JORDAN, Jeremy. An overview of semantic image segmentation. In: *Jeremy Jordan* [online]. 2018 [cit. 2022-12-24]. Dostupné z: https://www.jeremyjordan.me/semantic-segmentation/

[36] MATCHA, Anil Chandra Naidu. A 2021 guide to Semantic Segmentation. In: *Nanonets* [online]. 2020 [cit. 2023-02-28]. Dostupné z: https://nanonets.com/blog/semantic-image-segmentation-2020/

[37] TSANG, Sik-Ho. Review: FCN — Fully Convolutional Network (Semantic Segmentation). In: *Towards Data Science* [online]. 2018 [cit. 2023-04-22]. Dostupné z: https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1

[38] BARLA, Nilesh. The Beginner's Guide to Semantic Segmentation. In: *V7 Labs* [online]. 2021 [cit. 2022-24-10]. Dostupné z: https://www.v7labs.com/blog/semantic-segmentation-guide

[39] SINGH, Vaibhav. The Ultimate Guide to DeepLabv3 – With PyTorch Inference. In: *LearnOpenCV* [online]. 2022 [cit. 2023-02-19]. Dostupné z: https://learnopencv.com/deeplabv3-ultimate-guide/?fbclid=IwAR1y6-4rVWwOaRy9aihtwYooOsGvWtPoBk4zf9F0SmAKlVvSHH0N3_eFm1w

[40] CHEN, Liang-Chieh, *Yukun ZHU, George PAPANDREOU, Florian SCHROFF a Hartwig* ADAM. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation [online]. 18 [cit. 2023-05-10]. Dostupné z: https://arxiv.org/pdf/1802.02611.pdf https://github.com/tensorflow/models/tree/master/research/deeplab

[41] PRÖVE, Paul-Louis. An Introduction to different Types of Convolutions in Deep Learning. In: *Towards Data Science* [online]. 2017 [cit. 2023-04-13]. Dostupné z: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

[42] JIAO, Chong, Kehua SU, Weiguo XIE a Ziqing YE. *Burn image segmentation based on Mask Regions with Convolutional Neural Network deep learning framework: more accurate and more convenient* [online]. 2019, **7,** 41038-018-0137-9 [cit. 2023-04-10]. ISSN 2321-3876. Dostupné z:

doi:10.1186/s41038-018-0137-9

[43] CHEN, Liang-Chieh, George PAPANDREOU, Florian SCHROFF a Hartwig ADAM. *Rethinking Atrous Convolution for Semantic Image Segmentation* [online]. In: . 2017, s. 14 [cit. 2023-05-11]. Dostupné z: https://arxiv.org/abs/1706.05587

[44] RAIKOTE, Pranav. Object Detection – Part 4: Spatial Pyramid Pooling in Deep Convolution Networks (SPPnet). In: *Applied Singularity* [online]. [cit. 2023-05-01]. Dostupné z: https://appliedsingularity.com/2021/06/01/spatial-pyramid-pooling-in-deep-convolution-networks-sppnet/

[45] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition* [online]. 2015, 14 [cit. 2023-04-06]. Dostupné z: https://arxiv.org/pdf/1406.4729.pdf

[46] *ROS2 Tutorials* [online]. [cit. 2022-12-25]. Dostupné z: https://docs.ros.org/en/humble/Tutorials.html

[47] NEZNÁMY. Depth Camera D435. In: *Intel Realsense* [online]. [cit. 2023-01-01]. Dostupné z: https://www.intelrealsense.com/depth-camera-d435/

[48] LAIACKER, Maximilian. Rosbag2video. In: *GitHub* [online]. 2020 [cit. 2023-02-11]. Dostupné z: https://github.com/mlaiacker/rosbag2video/tree/foxy

[49] MAO, Lei a Shengjie LIN. DeepLab V3. In: *GitHub* [online]. 2020 [cit. 2022-12-18]. Dostupné z: https://github.com/leimao/DeepLab-V3

[50] PARAJULI, Sanjay. How to train Deeplab on Custom Dataset. In: *Medium* [online]. 2021 [cit. 2023-01-20]. Dostupné z: https://sanjayparajuli27.medium.com/how-to-train-deeplab-on-custom-dataset-a40c41c4c6a3

[51] TensorFlow. In: *GitHub* [online]. [cit. 2023-05-09]. Dostupné z: https://github.com/tensorflow/tensorflow

[52] CLARK, Jeffrey. *Pillow* [online]. In: . [cit. 2023-05-09]. Dostupné z: https://pillow.readthedocs.io/en/stable/

[53] NumPy Introduction. In: *W3Schools* [online]. [cit. 2023-05-09]. Dostupné z: https://www.w3schools.com/python/numpy/numpy_intro.asp

[54] Tf-slim. In: *Pypi* [online]. 2020 [cit. 2023-05-09]. Dostupné z: https://pypi.org/project/tf-slim/

[55] WIGNESS, Maggie, Sungmin EUM, John ROGERS, David HAN a Heesung KWON. *A RUGD Dataset for Autonomous Navigation and Visual Perception in Unstructured Outdoor Environments* [online]. In: . 2019 [cit. 2023-01-11]. Dostupné z: http://rugd.vision/

[56] INFECTEDBYSMILE, a RICARDO BOHNER. I need to change the palette (.pal file) of multiple images at once. How can I do this?. In: StackExchange [online]. 2021 [cit. 2023-03-28]. Dostupné z: https://superuser.com/questions/1684777/i-need-to-change-the-palette-pal-file-of-multiple-images-at-once-how-can-i-d

[57] LOTTE1990. Remove_gt_colormap.py. In: *GitHub* [online]. [cit. 2023-02-13]. Dostupné z: https://github.com/xanjay/models/blob/master/research/deeplab/datasets/remove_gt_colormap.py

[58] JANETZKY, Pascal. A hands-on guide to TFRecords. In: *Towards Data Science* [online]. 2021 [cit. 2023-05-09]. Dostupné z: https://towardsdatascience.com/a-practical-guide-to-tfrecords-584536bc786c

[59] GAMAUF, Thomas. Tensorflow Records? What they are and how to use them. In: *Medium* [online]. 2018 [cit. 2023-05-09]. Dostupné z: https://medium.com/mostly-ai/tensorflow-records-what-they-are-and-how-to-use-them-c46bc4bbb564

[60] Install TensorFlow. In: TensorFlow [online]. [cit. 2023-03-09]. Dostupné z: https://www.tensorflow.org/install/source_windows

[61] TORONTO, PRIYA. Semantic Segmentation at 30 FPS using DeepLab v3. In: Deep Learning Analytics*:* Leading the way [online]. 2019 [cit. 2023-03-29]. Dostupné z: https://deeplearninganalytics.org/semantic-segmentation/

[62] VERDONE, Antonio. Real-time semantic image segmentation with DeepLab in Tensorflow. In: Antonio Verdone blog [online]. 2018 [cit. 2023-03-25]. Dostupné z: https://averdones.github.io/real-time-semantic-image-segmentation-with-deeplab-in-tensorflow/

[63] Moja práca na GitHube [online]. In: . Dostupné z: https://github.*com*/Masuell/Deeplab/tree/master/research