Diplomová práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

3D sémantická segmentace RGBD dat pomocí hlubokých konvolučních neuronových sítí

Martin Hobza

Studijní program: Kybernetika a robotika.
Obor: Robotika.

Květen 2017
Vedoucí práce: Ing. Karel Zimmermann, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Martin H o b z a

Studijní program: Kybernetika a robotika

Obor: Robotika

Název tématu: 3D sémantická segmentace RGBD dat pomocí hlubokých konvolučních neuronových sítí

Pokyny pro vypracování:

Nastudujte metody 2D a 3D sémantické segmentace pomocí hlubokých konvolučních neuronových sítí jako je např. [1,2] a existující prostředí pro učení hlubokých sití (např. MatConvNet [3], Caffe či Tensorflow).
Navrhněte architekturu hluboké konvoluční sítě vhodnou pro 3D segmentaci z RGBD dat.
S použitím 3D anotací pro veřejně dostupný dataset Kitty [4], navrženou síť naučte a vyhodnoťte.

Seznam odborné literatury:

[1] Jonathan Long, Evan Shelhamer, Trevor Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR, 2015. (https://arxiv.org/abs/1411.4038)
[2] Rui Zhao, Wanli Ouyang, Hongsheng Li, Xiaogang Wang, Saliency Detection by Multi-Context Deep Learning, CVPR 2015 (http://www.ee.cuhk.edu.hk/~rzhao/project/deepsal_cvpr15/zhaoOLWcvpr15.pdf)
[3] http://www.vlfeat.org/matconvnet/
[4] http://www.cvlibs.net/datasets/kitti/

Vedoucí diplomové práce: Ing. Karel Zimmermann, Ph.D.

Platnost zadání: do konce letního semestru 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20. 2. 2017

Poděkování / Prohlášení

Rád bych hluboce poděkoval vedoucímu mé diplomové práce Ing. Karlu Zimmermannovi Ph.D. za jeho vedení, diskuzi, pomoc a trpělivost

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

V Praze dne

.....

Abstrakt / Abstract

Cílem této práce bylo nastudování problematiky konvolučních neuronových sítí. Jejich využití v oblasti 2D a 3D sémantické segmentace a seznámení se s možnostmi jejich implementace. Výsledkem práce je otestování implementace konvoluční neuronové sítě na anotovaných datech z veřejně přístupného datasetu KITTI a vyhodnocení jejích výsledků.

Klíčová slova: neuron, neuronové sítě, konvoluční neuronové sítě, KITTI, sémantická segmentace

Goal of this thesis was to study the issue of convolutional neural networks. Their use in 2D and 3D semantic segmentation and familiarize with possibilities of their implementation. The result of this thesis is to test an implementation of such convolutional network on anooted data from the publicly available KITTI dataset and evaluate its results.

Keywords: neuron, neural network, convolutional neural network, KITTI, semantic segmentation

Obsah

1 Úvod 1
1.1 Motivace 1
1.2 Přehled 1
2 Neuronové sítě 2
2.1 Neuron 2
2.2 Neuronová síť 4
2.3 Příklady modelů neuronových sítí 5
2.3.1 Perceptron 5
2.3.2 Vícevrstvá dopředná neuronová síť 6
3 Konvoluční neuronové sítě 8
3.1 Motivace 8
3.2 Konvoluční vrstva neuronové sítě 9
3.2.1 Příznaková mapa 9
3.2.2 Subsampling 9
3.3 Učení konvoluční neuronové sítě 10
3.4 Plně konvoluční síť 12
3.5 Existující prostředky pro učení konvolučních neuronových sítí 12
3.5.1 MatConvNet 12
3.5.2 Caffe 13
3.5.3 TensorFlow 13
3.6 Použití konvolučních neuronových sítí 13
4 KITTI Dataset 14
5 Praktická část 16
5.1 Automatická anotace 16
5.2 Učení FCN 19
5.3 Segmentace prostředí 24
6 závěr 26
Literatura 27
A DVD 29

Tabulky / Obrázky

5.1. výsledky přeučení sítě 22

2.1. Model neuronu 2
2.2. Skoková přenosová funkce 3
2.3. Lineární přenosová funkce 3
2.4. Sigmoidní funkce 3
2.5. Gaussova funkce 3
2.6. dopředná síť 4
2.7. zpětnovazební síť 4
2.8. lineárně separovatelná množina .. 6
2.9. Vícevrstvá neuronová síť 7
3.1. příklad identifikovaného znaku .. 8
3.2. příklad deformace znaku 8
3.3. příznaková mapa 9
3.4. redukční proces 10
4.1. KITTI platforma 14
4.2. KITTI uspořádání 15
5.1. automatická anotace 1 17
5.2. automatická anotace 2 17
5.3. automatická anotace 3 18
5.4. automatická anotace 4 19
5.5. Mísení skupin na okrajích objektů 19
5.6. srovnání automatické anotace . 20
5.7. srovnání manuální anotace 20
5.8. Architektura VGG16 21
5.9. Architektura FCN8s 21
5.10. průběh učení pomocí manuálně anotovaných dat 23
5.11. průběh učení pomocí malého množství automaticky anotovaných dat 23
5.12. výstup sítě naučené pomocí AA 24
5.13. zpětná projekce do prostoru ... 24

Kapitola 1
Úvod

1.1 Motivace

Rozpoznávání objektů na obrázku je dnes velmi rozšířené, ať už se jedná o rozpoznávání lidského obličeje, ručně psaného textu, nebo jiných objektů. Ke splnění těchto úkonů se v praxi stále častěji používá implementace pomocí různých druhů neuronových sítí. Neuronové sítě a ještě lépe konvoluční neuronové sítě jsou k těmto účelům vhodné díky jejich struktuře, učícím se vlastnostem a míře abstrakce.

V posledních letech slýcháme pojem konvoluční neuronové sítě čím dál častěji[1]. V oblasti umělé inteligence jsou průlomovým nástrojem. Dnes dosahují nejlepších výsledků nejen při rozeznávání obrazu, ale i řeči a v dalších klasifikačních úlohách. Navíc jsou díky mnoha veřejně přístupným knihovnám, z nichž několik je vyjmenováno v sekci3.5, daleko přístupnější veřejnosti. Navíc většina těchto knihoven nabízí výpočet na grafických procesorech, které jsou dnes stále silnější a levnější.

Dnes jsou sítě schopné naučit se mnoho z mála. Teoreticky může jedna síť učit druhou. Metodou *Weak Labeling*[7] lze apriorní klasifikační znalostí naučit segmentační síť. V této práci provedu něco podobného. Ale místo *Weak Labelů* použiji *Strong Labely* získané z apriorní sémantické znalosti problému, pomocí kterých budu učit segmentační síť.

1.2 Přehled

Na začátku této práce je čtenář obeznámen se základní problematikou, architekturou a principy fungování neuronů 2.1 a neuronových sítí. Také je zde představeno několik vybraných modelů

Další část se zabývá popisem konvolučních neuronových sítí3. Konvoluční neuronové sítě jsou v současnosti jedním z nejvhodnějších nástrojů pro identifikaci objektů na obrázku, protože na rozdíl od klasických neuronových sítí nejsou třeba žádné úpravy obrázku před vstupem do sítě. Zde je popsán princip jejich fungování a probrány jejich výhody oproti standardním neuronovým sítím. Následuje část přibližující problematiku sémantické segmentace a omezení, které klade na konvoluční neuronové sítě.

Před praktickou částí je zmíněno několik existujících struktur pro práci s konvolučními neuronovými sítěmi. Po té je představen dataset KITTI4. Následně je rozebrána praktická část, ve které je popsán záměr práce, implementace jednotlivých částí a prodiskutovány dosažené výsledky.

[1] trends.google.com, Google Trends `https://trends.google.com/trends/explore?date=all&q=convolutional%20neural%20network`

Kapitola 2
Neuronové sítě

Modely neuronových sítí vycházejí z neurofyzikální předlohy zpracovávání informací v nervových buňkách, tedy do určité míry kopírují přírodní způsob myšlení. Za počátek modelování neuronových sítí se považuje práce Warrena McCullocha a Waltera Pittsa z roku 1943 [1].

2.1 Neuron

Model umělého neuronu

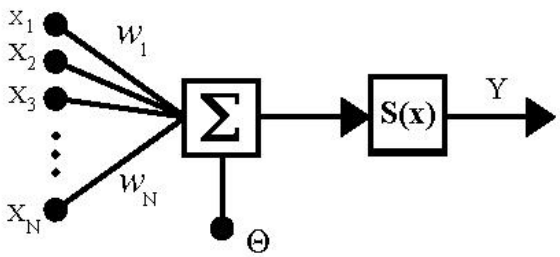
Umělý neuron je definován jako trojice $(S, \vec{\omega}, \Theta)$ kde:

- $S(x)$ je přenosová funkce $R \to R$
- $\vec{\omega} \in R^n$ je vektor vah
- $\Theta \in R$ je práh

Neuron má obecně n reálných vstupů tvořících vstupní vektor $\vec{x} = (x_1, ..., x_n)$. Tyto vstupy jsou ohodnoceny reálnými váhami $\vec{\omega} = (\omega_1, ..., \omega_n)$.

Nejčastěji používaný model umělého neuronu obsahuje také práh (bias) Θ. Takový neuron je zobrazen na obrázku 2.1 a jeho výstupní signál $y \in R$ je popsán vztahem

$$y = S(\omega x^\top - \Theta) \tag{1}$$



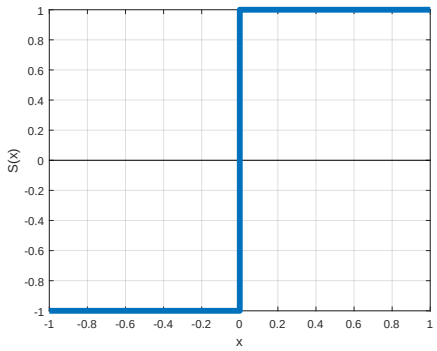
Obrázek 2.1. Model umělého neuronu [2]

Charakteristika výstupního signálu záleží na přenosové funkci $S(x)$. Vstup do přenosové funkce nazveme *potenciál* $\xi = \omega x^\top + \Theta$. Nejčastěji používané přenosové funkce jsou

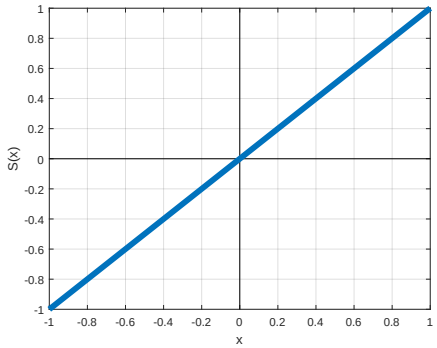
- $f(x) = sign(x)$ Silně nelineární, skoková přenosová funkce 2.2
- $f(x) = x$ Lineární přenosová funkce 2.3
- $f(x) = \frac{1}{1+e^{-x}}$ Sigmoidní přenosová funkce 2.4
- $f(x) = e^{-x^2}$ Gaussovská přenosová funkce 2.5

[1] www.chalmers.se, A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY `http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf`

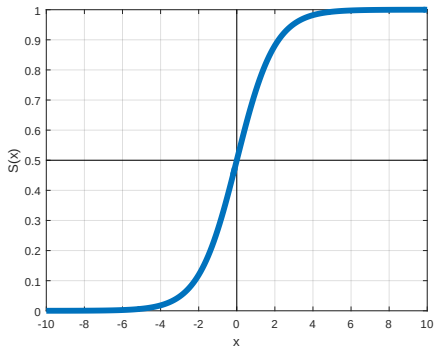
[2] wikipedia.org, Umělá neuronová síť `https://upload.wikimedia.org/wikipedia/commons/3/35/NeuronModel.jpg`



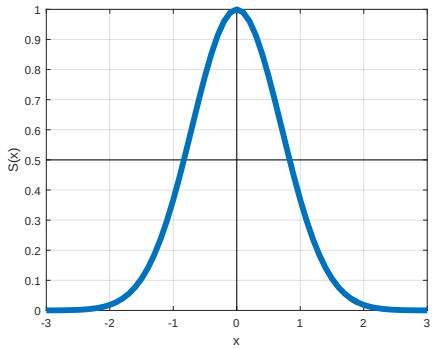
Obrázek 2.2. Skoková funkce



Obrázek 2.3. Lineární funkce



Obrázek 2.4. Sigmoidní funkce



Obrázek 2.5. Gaussova funkce

2.2 Neuronová síť

Neurony popsané v předchozí sekci můžeme mezi sebou propojit tím, že výstup y jednoho neuronu použijeme jako část vstupu $\vec{x}$ dalšího neuronu, případně dalších neuronů. Tímto se vytvoří neuronová síť. Počet neuronů a jejich způsob propojení určuje topologii neuronové sítě. Ta je určena orientovaným grafem představujícím propojení mezi neurony tak, že neurony představují vrcholy a jejich propojení představují hrany.

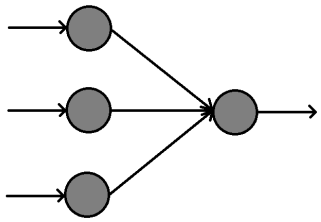
Matematicky se dá neuronová síť popsat takto:

- N je neprázdná množina neuronů
- $C \subset N \times N$ je množina orientovaných spojů mezi neurony
- $I \subset N$ je množina vstupních neuronů
- $O \subset N$ je množina výstupních neuronů
- $\omega : C \to R$ je váhová funkce
- $t : N \to R$ je prahová funkce

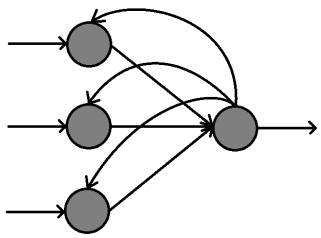
Neuronové sítě se odlišují několika vlastnostmi. Několik charakteristických vlastností sítě jsou například topologie sítě nebo způsob učení. U topologie rozlišujeme dva základní typy

- Neuronové sítě s dopředným šířením signálu
- Neuronové sítě se zpětnovazebním šířením signálu

V neuronových sítích s dopředným šířením signálu, tedy v sítích s acyklickou topologií, postupují všechny signály iterativně od vstupní vrstvy až k výstupní. V každé iteraci přijmou všechny neurony v dané vrstvě signál z předchozí vrstvy, spočítají svůj výstup a ten pošlou do následující vrstvy. U sítí se zpětnou vazbou, tedy cyklickou topologií, se u neuronů objevují vstupy závislé na výstupu sítě z předchozího cyklu. Příklady takových sítí jsou vidět na obrázcích 2.6 a 2.7.



Obrázek 2.6. Síť s dopředným šířením signálu



Obrázek 2.7. Síť se zpětnovazebným šířením signálu

V acyklických sítích nejsou žádné smyčky. Neurony tedy lze v síti disjunktně rozdělit do vrstev. Neuronová síť se skládá ze vstupná vrstvy, výstupní vrstvy a případně dalších

vrstev mezi nimi. Vrstvy mezi vstupní a výstupní vrstvou se nazývají skryté vrstvy. Při vhodném uspořádání skrytých vrstev pak vedou spoje pouze od nejnižších vrstev do vyšších vrstev

Neurony ve vstupní vrstvě mají jen jeden vstup a lineární přenosovou funkci. Tedy posílají svůj vstup přímo na výstup. Tato vrstva se stará o distribuci vstupního signálu do ostatních vrstev. Výstupní vrstva je naopak jediný výstup neuronově sítě. Vrstvy mezi vstupní a výstupní vrstvou nijak nezasahují do vnějšího okolí.

Další důležitá vlastnost sítí je schopnost učit se. Učení je algoritmus při kterém jsou nastaveny váhy ω tak aby síť co nejlépe zpracovala nová data. Učící algoritmy jsou rozděleny do dvou hlavních kategorií. Tedy učení s učitelem a učení bez učitele.

Učení s učitelem se vyznačuje tím, že známe požadovaný výstup sítě k danému vstupu. Tento algoritmus probíhá tak, že vstupní vektor je přiveden na vstup neuronové sítě. Síť tento vektor zpracuje se svými počátečními vahami. Výstupní vektor sítě je porovnán s požadovaným vektorem a pomocí jejich rozdílu je spočtena chybová funkce. Na jejím základě jsou pak upravovány váhy v síti tak, aby opětovný průchod vstupních dat snížil rozdíl mezi výstupem sítě a požadovaným výstupem.

Učení bez učitele naopak nemá požadavky na výstup. Síť dostává na vstup sadu vzorů, které si sama třídí. Váhy se nastavují tak aby výstup byl konzistentní, tedy aby síť poskytovala stejnou odezvu na podobné vstupní signály.

2.3 Příklady modelů neuronových sítí

2.3.1 Perceptron

Perceptron je základním stavebním prvkem neuronových sítí. Historický název vychází z matematického modelu biologického neuronu. Jeho autorem byl v roce 1957 Frank Rosenblatt [1].

Typickým perceptronem je jednoduchá síť s n vstupy a jedním neuronem. Přenosová funkce neuronu je buď binární (tj. má hodnotu 0 nebo 1), nebo bipolární (tj. hodnoty -1,0 nebo 1). Výstupem perceptronu je $y = S(\xi)$ a aktivační funkce

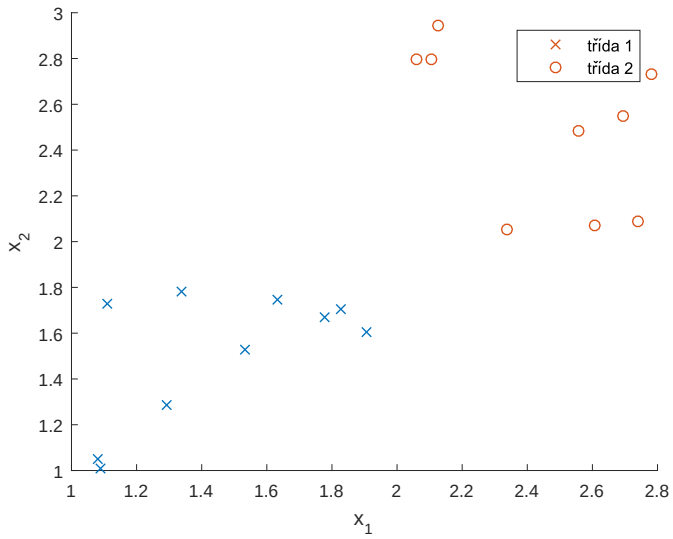
$$S(\xi) = \begin{cases} 1 & pokud\, \xi \geq 0 \\ 0 & pokud\, \xi < 0 \end{cases} \tag{2}$$

Perceptron se učí metodou učení s učitelem a jeho algoritmus je následovný:

- Nastavení vah $\omega_1, ..., \omega_n$ a prahu Θ
- Přivedení vstupního vektoru do Perceptronu
- Výpočet výstupu
- Porovnání výstupu z perceptronu s požadovaným výstupem
- Výpočet chybové funkce $e = S(\xi) - o$ kde o je očekávaný výstup a úprava vah
- Žádná chyba, žádná úprava vah, pokračuje se k dalšímu vstupnímu vektoru
- Při zjištění chyby se k váhám i k prahu přičte hodnota e

Perceptron se nejčastěji používá pro klasifikaci, například přiřazení objektu do konkrétní množiny. Učící algoritmus perceptronu je konečný jen pokud je učící množina lineárně separovatelná, jako například v 2.8. Složitější funkce jako například XOR nelze řešit pomocí perceptronu, ale lze je řešit rozšířením tohoto základního modelu do vícevrstvé neuronové sítě.

[1] csis.pace.edu, Rosenblatt contributions `http://csis.pace.edu/~ctappert/srd2011/rosenblatt-contributions.htm`



Obrázek 2.8. Lineárně separovatelná množina

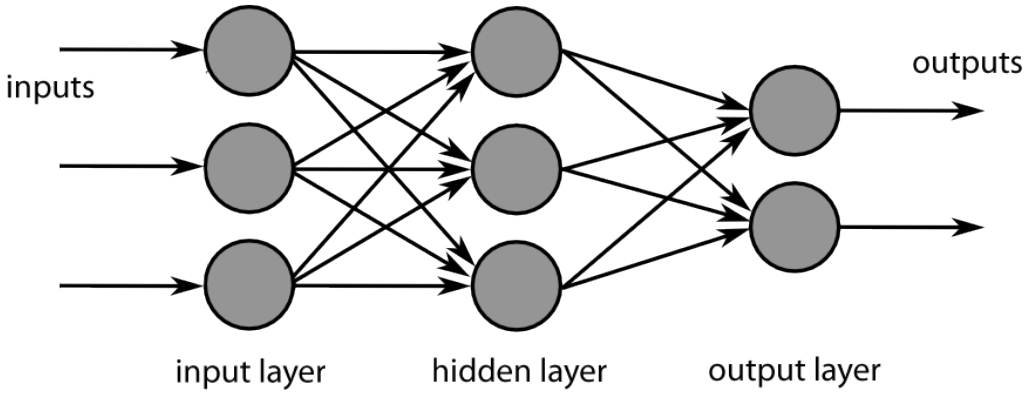
2.3.2 Vícevrstvá dopředná neuronová síť

Vícevrstvá dopředná neuronová síť byla uvedena v roce 1986 a jejími autory jsou G.E. Hinton, E. Rumelhart a R.J. Williams[1]. Jeho cílem bylo vyřešit omezené schopnosti perceptronu. Vícevrstvé sítě se skládají alespoň ze tří vrstev. Vstupní vrstvy, skryté vrstvy a výstupní vrstvy, jako na obr 2.9. Ačkoliv některá literatura nepovažuje vstupní vrstvu za součást sítě a v tom případě je popsaný model jen dvouvrstvý.

Neurony v takové síti jsou spojeny tak, že každý neuron v i-té vrstvě je spojený s každým neuronem v $i+1$-té vrstvě. Neexistují žádná spojení mezi neurony ve stejné vrstvě.

Pro učení vícevrstvé neuronově sítě se nejčastěji používá metoda zpětného šíření chyby (tzv. **backpropagation**). Nejedná se o typ zpětného šíření signálu, toto není rekurentní síť. Jejím autorem byl již zmiňovaný E. Rumelhart. Obdobně jako u perceptronu, i tato metoda je metoda učení s učitelem a je založena na úpravě vah spojů počítané na základě chyby. Také chyba se počítá jako rozdíl mezi výstupem neuronové sítě a požadovaným výstupem. Ale místo přičítání chyby přímo k vahám se používají gradientní metody. Váhy se upravují nejdříve od výstupní vrstvy a postupuje se ke skryté vrstvě, odtud pochází název zpětného šíření. Stejně jako u perceptronu se upravují pouze váhy a prahy, charakteristiky přenosových funkcí jednotlivých neuronů se nemění.

[1] cs.toronto.edu, Learning Internal Representations by Error Propagation `http://www.cs.toronto.edu/~fritz/absps/pdp8.pdf`



Obrázek 2.9. Vícevrstvá dopředná neuronová síť [1]

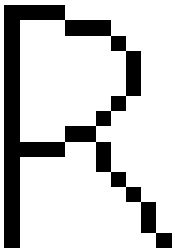
[1] technobium.com, Multilayer neural network `http://technobium.com/wordpress/wp-content/uploads/2015/04/MultiLayerNeuralNetwork.png`

Kapitola 3
Konvoluční neuronové sítě

V této kapitole se zaměříme na speciální typ neuronových sítí, kterým jsou konvoluční neuronové sítě. Budou zde popsány jejich výhody oproti normálním vícevrstvým neuronovým sítím a jejich použití

3.1 Motivace

Představme si, že chceme použít vícevrstvou neuronovou síť z předchozí kapitoly k rozpoznání znaku na obrázku. Například budeme chtít identifikovat písmeno na jednoduchém obrázku o rozměrech 20×20



Obrázek 3.1. příklad identifikovaného znaku

Obrázek bude pro síť interpretován jako vektor o velikosti 20×20 a bude nabývat hodnot 0 a 1. Nula bude odpovídat bílému pixelu, jednička černému. Síť s několika skrytými vrstvami se může naučit takovéto znaky rozpoznat. Problém ale nastane když budou nová data oproti těm naučeným posunuta, pootočena, nebo jinak lehce zdeformována.

Stačí jen malý posun a celý vstupní vektor se může přestat podobat jakémukoliv z naučených. Pro posun by se ještě dalo použít předzpracování obrázku, například vystředěním. Ale jakmile budeme po síti chtít rozpoznávání i ručně psaného textu, budeme čelit kombinaci mnoha různých deformací, které vůbec nepůjde snadno odstranit. Následující příklad nám situaci objasní

Obrázek 3.2. příklad deformace znaku

Na obrázku vidíme na první pohled podobné ručně psané písmena R. Z pohledu sítě se ale budou jejich vstupní vektory velmi lišit. Na obrázku je modrou barvou znázorněno jakou část mají tyto dva vzory společnou. Motivací konvolučních neuronových sítí je právě odolnost vůči deformacím jako je tato. Této odolnosti je dosaženo díky šikovnému sdílení vah mezi více neurony. Pokud se k takto naučené síti dostane podobně zdeformovaný obrázek, pak se sice nedostane stejný vstup na stejné neurony jako u učícího obrázku, ale může se dostat na neurony sdílející tu správnou váhu.

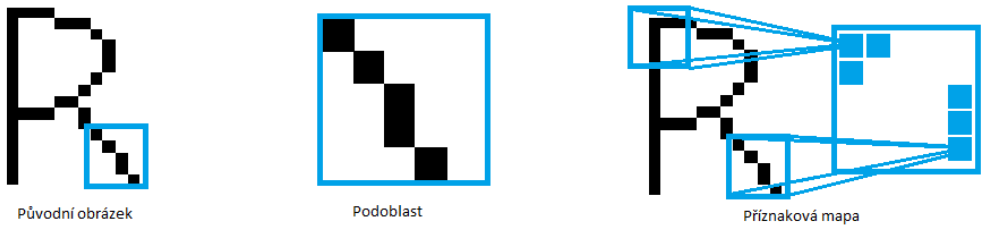
3.2 Konvoluční vrstva neuronové sítě

Základním stavebním kamenem konvoluční neuronové sítě jsou příznaková mapa a redukční (subsamplingová) vrstva. Konvoluční vrstva je složena z několika příznakových map a může navazovat jak na vstupní vrstvu, tak na subsamplingovou.

3.2.1 Příznaková mapa

Smyslem příznakové mapy je získání lokálních charakteristik obrázku. V rámci našeho příkladu s rozměry 20×20 bychom uvažovali podoblast o rozměrech například 5×5. Každý pixel takovéto podoblasti bude vstupem jednoho neuronu, který bude v dané oblasti rozpoznávat nějakou vlastnost. Například svislé nebo horizontální čáry, prázdnou oblast nebo nějakou křivku. Takovýto neuron bude mít 25 + práh, tedy 26 vstupů.

Zde přichází do hry zmiňované sdílení vah. Tento neuron rozkopírujeme nad celým obrázkem tak, že se jednotlivé podoblasti překrývají. Takovýchto podoblastí a jim příslušných neuronů bude potom 16×16 tedy 256. Tato operace je vlastně konvolucí nad obrazem, kde neuron zpracovávající pixely z podobrazu je konvolučním jádrem. Tím pádem všechny neurony v této příznakové mapě sdílejí váhy i práh. Celá mapa má tedy jen 26 vah, přestože obsahuje 256 neuronů. Důsledkem takovéhoto pokrytí obrázku příznakovou mapou je detekce lokálních charakteristik i při jejich posunu.



Obrázek 3.3. získání příznakové mapy

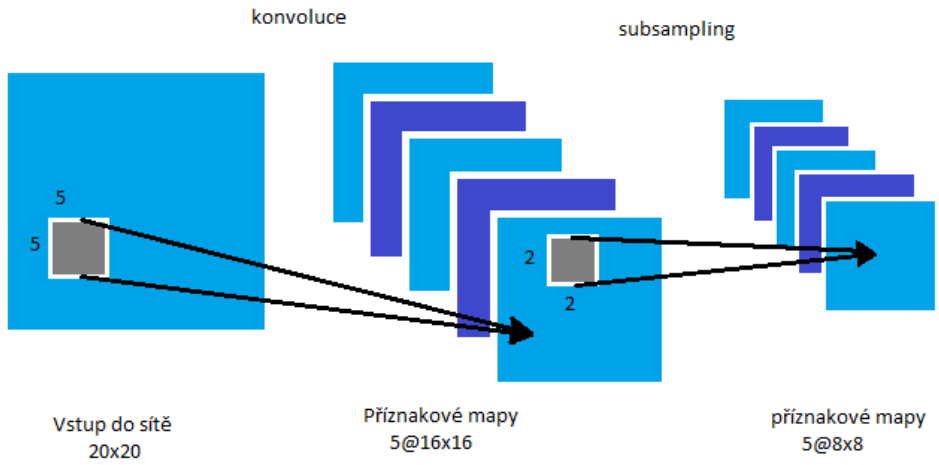
Ovšem jedna příznaková mapa zjišťuje v obrázku jen jednu charakteristiku, proto použijeme několik různých příznakových map. Každá mapa bude opět mít 256 neuronů a 26 vah. Tyto mapy mezi sebou však už nesdílejí žádné parametry. Kdybychom například chtěli detekovat vertikální čáry, horizontální čáry, obě diagonály a prázdný prostor, měli bychom 5 příznakových map. Potom by tato konvoluční vrstva obsahovala celkem 1280 neuronů (256×5) a 130 unikátních vah (26×5).

3.2.2 Subsampling

Subsamplingová vrstva se vždy pojí na konvoluční vrstvu. Jak její název napovídá, jejím účelem je redukce velikosti příznakových map. K tomu lze použít různé funkce jako například průměrování, lineární kombinace nebo výběr maxima (Max Pooling). Kdybychom si pro náš příklad zvolili výběr maxima (max-pooling), tak z předchozí

konvoluční vrstvy budeme vyřezávat podoblasti s rozměry $k \times k$ a z takové podoblasti vybereme maximum. Tedy když vstup do subsamplingové vrstvy bude mít rozměry $N \times N$, pak výstup bude mít rozměry $\frac{N}{k} \times \frac{N}{k}$.

Když tedy zvolíme $k = 2$ v našem konkrétním příkladě, pak bude vstupem do subsamplingové vrstvy 5 příznakových map o velikosti 16×16, a výstupem bude 5 příznakových map o rozměrech 8×8. Vizualizaci tohoto procesu můžeme vidět na obrázku 3.4



Obrázek 3.4. Subsampling konvoluční vrstvy

Po takovémto spojení několika konvolučních a subsamplingových vrstev je výstup poslední konvoluční nebo subsamplingové vrstvy transformován do jednorozměrného vektoru. Ten je přiveden jako vstup do již normálně fungující neuronové sítě, která vypočítá konečný výstup.

3.3 Učení konvoluční neuronové sítě

Tato sekce čerpá z [1]. Jak bylo popsáno, konvoluční neuronová síť se skládá z konvolučních a subsamplingových vrstev napojených na jednorozměrnou neuronovou síť. Taková síť se bude učit obdobou metody zpětného šíření chyby, avšak upravené pro jednotlivé vrstvy sítě.

Předpokládejme, že za konvoluční vrstvou l následuje subsamplingová vrstva $l + 1$. Z povahy subsamplingové vrstvy je několik jednotek z vrstvy l spojených do jedné ve vrstvě $l + 1$. Pro počítání chyb v konvoluční vrstvě l je tedy vhodné zvětšit mapu chyb ze subsamplingové vrstvy $l + 1$ na velikost mapy z vrstvy l. Následně bude zvětšená mapa ze subsamplingové vrstvy $l + 1$ vynásobena derivací přenosové funkce mapy z vrstvy l. Rovnice chyby j-té mapy v l-té vrstvě je tedy:

$$\delta_j^l = \beta_j^{l+1}(f'(u_j^l) \circ up(\delta_j^{l+1})) \tag{1}$$

Kde β_j^{l+1} jsou váhy j-té mapy v subsampligové vrstvě $l + 1$, $f'(u_j^l)$ je derivace přenosové funkce neuronu u_j^l. Funkce $up()$ je funkce zvětšení mapy chyb subsamplingové vrstvy. Taková funkce se dá realizovat například pomocí Kroneckerova součinu

$$up(X) = X \otimes 1_{n \times n} \tag{2}$$

Když máme vypočtené chyby pro každou mapu, můžeme počítat gradient prahu jako součet chyb map δ_j^l

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^l)_{u,v} \tag{3}$$

Podobně budeme počítat i gradient vah, jako součet přes všechny spoje používající tuto váhu.

$$\frac{\partial E}{\partial k_{ij}^l} = \sum_{u,v} (\delta_j^l)_{uv} (p_i^{l-1})_{uv} \tag{4}$$

Kde p_i^{l-1} představuje část v x_i^{l-1}, která byla vynásobena jádrem k_{ij}^l během konvoluce pro výpočet prvku na souřadnicích (u, v) na výstupu konvoluční mapy x_j^l.

Cílem subsamplingové vrstvy je zmenšení vstupních map, jak již bylo zmíněno v předchozí sekci. Formálněji zapsáno:

$$x_j^l = f(\beta_j^l down(x_j^{l-1}) + b_j^l) \tag{5}$$

Kde $down()$ representuje subsamplingovou funkci. Každá výstupní mapa má vlastní multiplikativní práh β a aditivní práh b, tyto dvě hodnoty můžeme upravovat a aktualizovat u subsamplingové vrstvy.

Pokud se za subsamplingovou vrstvou nachází jednorozměrná vrstva, můžeme chybovou funkci počítat metodou zpětného šíření. Pokud se za subsamplingovou vrstvou nachází další konvoluční vrstva, musíme zjistit jaké části této vrstvy odpovídají pixelům následující vrstvy. Váhy násobící spojení mezi prostředím na vstupu a pixelem na výstupu odpovídají váhám otočeného konvolučního jádra.

$$\delta_j^l = f'(u_j^l) \circ conv2(\delta_j^{l+1}, rot180(k_j^{l+1}), 'full') \tag{6}$$

V tomto zápise byla použita syntaxe MATLABu, kde se díky parametru 'full' vypořádáme s hraničními případy tak, že případy kde konvoluční jádro postrádá vstupy jsou chybějící vstupy doplněny nulami.

Nyní můžeme vypočítat gradient pro aditivní a multiplikativní práh. Aditivní práh je opět jen součet přes chybovou mapu.

$$\frac{\partial E}{\partial b_j} = sum_{u,v} (\delta_j^l)_{uv} \tag{7}$$

Pro multiplikativní práh β je třeba použít originální mapu vytvořenou při dopředném průchodu sítí.

$$d_j^l = down(x_j^{l-1}) \tag{8}$$

Potom je gradient pro β určen vztahem

$$\frac{\partial E}{\partial \beta_j} = \sum_{u,v} (\delta_j^l \circ d_j^l)_{uv} \tag{9}$$

3.4 Plně konvoluční síť

Plně konvoluční síť se odlišuje od předchozí sítě tím, že všechny její vrstvy jsou konvoluční filtry nebo subsamplingové. Tedy taková síť neobsahuje žádné plně propojené vrstvy a i její konečné rozhodovací vrstvy jsou jen filtry.

Plně konvoluční síť se učí a rozhoduje na základě lokálních charakteristik. Připojení plně propojených vrstev umožní síti se učit i díky globální informaci. Nevýhodou ale je, že plně propojené vrstvy očekávají vstup určitých rozměrů. Díky absenci těchto vrstev je plně konvoluční síť schopná zpracovat obrázek téměř libovolné velikosti

Další výhodou plně konvoluční sítě je zachování informace o umístění. Plně propojené vrstvy u Konvolučních neuronových sítí tuto informaci většinou ztrácejí, protože jsou „plně propojené“. Přesto mohou plně propojené vrstvy dělat segmentaci, pokud se omezí jen na malé množství kategorií a omezené vizuální variace natolik, že aktivace propojené vrstvy poslouží jako statistika pro tyto obrazy[15][16].

3.5 Existující prostředky pro učení konvolučních neuronových sítí

Existujících implementací neuronových sítí je dnes velké množství. Často se jedná o toolboxy a knihovny, které vybaví programátora řadou základních nástrojů. V této kapitole se na některé z nich podíváme.

Navíc je dnes možné pro tyto implementace i využití grafického procesoru (GPU). V současnosti je možné použít dva způsoby programování s GPU, které jsou *OpenCL*[1] a *CUDA*[2].

OpenCL (Open Computing Language) je první otevřený standard pro multiplatformové paralelní programování. To znamená, že program napsaný v *OpenCL* může být spuštěn jak na procesorech (CPU), grafických jednotkách (GPU), tak i na programovatelných hradlech (FPGA) nebo jiném hardwaru. *OpenCL* je dostupný pro produkty od společností *AMD*, *Intel*, *nVidia*, *IBM* nebo *Apple*. *OpenCL* sestává z knihoven implementovaných v C/C++ a překladačů umožnujících spuštění na jednotlivých platformách. Pro použití v jiných programovacích jazycích existují různé API knihovny dodané třetí stranou.

CUDA je platforma vytvořena společností *nVidia*. Tato platforma je implementována pomocí grafických jednotek a umožňuje přímý přístup k virtuálním instrukcím nebo paměti v *CUDA* grafických jednotkách. *CUDA* je také implementována v C/C++. Pro použití v jazyce Java jsou k dispozici knihovny jako například *JCUDA*. Nevýhodou této platformy je, že funguje jen s grafickými jednotkami společnosti *nVidia*.

3.5.1 MatConvNet

MatConvNet [3] je toolbox pro MATLAB. Za jeho vznik se zaslujuje Oxford Visual Geometry Group. Narozdíl od jiných neoficiálních toolboxů pro matlab je všechen jeho kód i psán v matlabu a nic neschovává jako kompilovaný kód. V kombinaci s Parallel Computing toolboxem a platformou CUDA lze jeho používání urychlit využitím výpočetního výkonu grafické karty. Tento Framework jsem zvolil i pro svojí práci, protože preferuji Matlab.

[1] khronos.org, OpenCL `https://www.khronos.org/opencl/`
[2] nvidia.com, CUDA toolkit `https://developer.nvidia.com/accelerated-computing-toolkit`
[3] vlfeat.org, MatConvNet `http://www.vlfeat.org/matconvnet/`

3.5.2 Caffe

Caffe[1] je framework vyvíjen laboratoří Berkley AI Research[2].Je psaný v C++ a je spustitelný jak v pythonu, tak v MATLABu. Stejně jako MatConvNet podporuje výpočet na GPU pomocí CUDA.

3.5.3 TensorFlow

TensorFlow[3] je softwarová knihovna pro numerické výpočty pomocí metody data flow graph vyvinutá společností Google. Je psán v C++ a Pythonu. Podporuje výpočet na GPU pomocí CUDA

3.6 Použití konvolučních neuronových sítí

Konvoluční neuronové sítě byly primárně navrženy pro rozpoznávání obrazových vzorů přímo z pixelů obrazu s minimálním předzpraováním. Jejich vznik lze datovat do roku 1980 kdy Kunihiko Fukushima vytvořil síť *Neocognitron*[2]. Poté pomalu probíhal jejich vývoj. Další zlom nastal v roce 2012, kdy Alex Krizhevsky vytvořil s kolegy síť, která předčila všechny ostatní v soutěži ILSVRC na datasetu *ImageNet*[3]. Od té doby se konvoluční sítě těší daleko větší popularitě a mnoho sítí staví na principech Alexovi sítě *AlexNet*.

Několik principů představených při tvorbě sítě *AlexNet*

- ReLU (Rectified Linear Unit) Model neuronu s aktivační funkcí

$$f(x) = max(0, x)$$

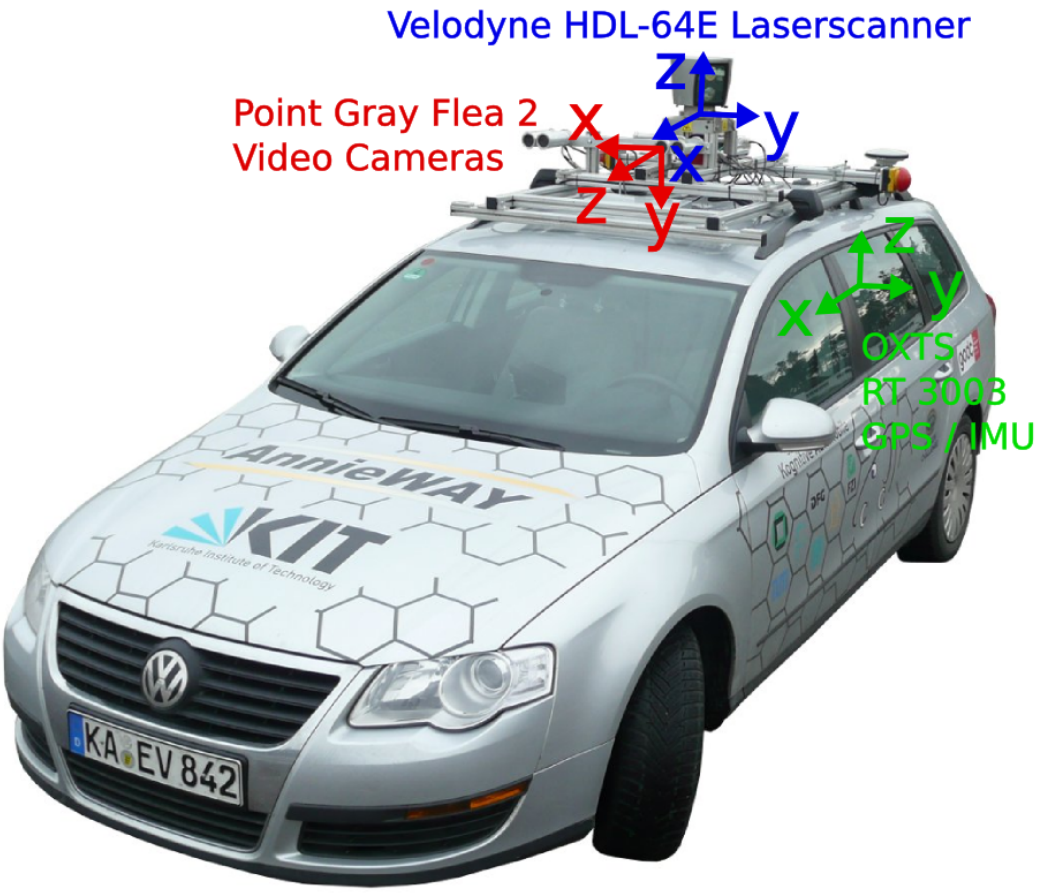
  Pro kladné hodnoty součtu váhovaných vstupů má lineární průběh, pro záporné hodnoty dává nulu.
- Data augmentation Jedná se o umělé navýšení velikosti datasetu vyříznutím menších obrazů ze vstupních obrazů, jejich nenáročným transformováním a prezentováním je síti jako nová trénovací data.
- *Dropout* Kombinace výsledků několika modelů sítě je efektivním způsobem jak zmenšit chybu, ale narůstá při ní výpočetní náročnost. Technika Dropout spočívá v nastavení nulového výstupu skrytých neuronů s pravděpodobností 0,5. Takové neurony se nepodílejí na učení sítě.

Dnes se konvoluční neuronové sítě používají nejen k rozpoznání vzoru na obrazu, detekování v rámci bounding boxu [9] [5] nebo segmentaci na úrovni jednotlivých pixelů [4] , ale rozšiřují se i na detekci vzorů a segmentaci ve 3D prostoru [6] [8].

[1] caffe.berkeleyvision.org, Caffe `http://caffe.berkeleyvision.org/`
[2] bair.berkley.edu, BAIR `http://bair.berkeley.edu/`
[3] tensorflow.org, TensorFlow `https://www.tensorflow.org/`

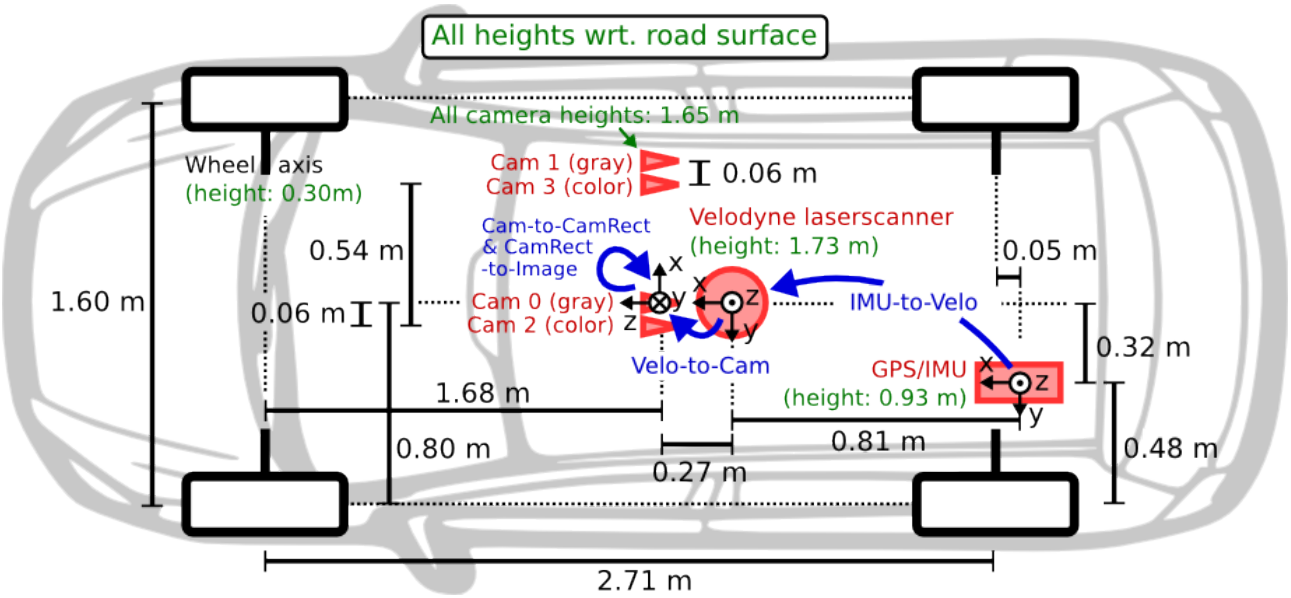
Kapitola 4
KITTI Dataset

Veřejně přístupný dataset KITTI [1] je soubor dat získaných z autonomní platformy Annieway [2]. Jedná se o RGB a Greyscale obrázky pořízených čtveřicí kamer, jim příslušného 3D skenu okolí auta a GPS souřadnice. Uspořádání jednotlivých senzorů na voze je vidět na obrázcích 4.1 a 4.2



Obrázek 4.1. Autonomní platforma Annieway[3]

[1] cvlibs.net, KITTI dataset `http://www.cvlibs.net/datasets/kitti/index.php`
[2] mrt.kit.edu, Annieway platform `http://www.mrt.kit.edu/annieway/`
[3] cvlibs.net, KITTI setup `http://www.cvlibs.net/datasets/kitti/setup.php`



Obrázek 4.2. Uspořádání senzorů na platformě[1]

Konkrétně se jedná o následující senzory:

- 1 navigační systém OXTS RT 3003
- 1 Laserscanner Velodyne HDL-64E
- 2 černobílé kamery, 1.4MPx Point Grey Flea 2 (FL2-14S3M-C)
- 2 barevné kamery, 1.4MPx Point Grey Flea 2 (FL2-14S3-C)

Dataset je koncipován formou soutěží mezi algoritmy vytvořenými uživateli v několika kategoriích, jako odometrie, identifikace objektů nebo sledování objektů. Je zde i kategorie sémantické segmentace, ale pro tu nejsou žádná srovnávací trénovací data poskytována týmem stojícím za KITTI a tím pádem ani žádné oficiální vyhodnocení a srovnáni algoritmů uživatelů. Je zde jen malé množství ručně anotovaných příspěvků od uživatelů.

[1] cvlibs.net, KITTI setup `http://www.cvlibs.net/datasets/kitti/setup.php`

Kapitola 5
Praktická část

Jak již bylo zmíněno v sekci 3.6. Existuje velké množství sítí sloužících k nejrůznějším úlohám. Od detekce obrazových vzorů pomocí bounding boxů[5], detekce vzorů v reálném čase[9], přes 3D detekci[12], až po rekonstrukci 3D scény z 2D obrazu [13].

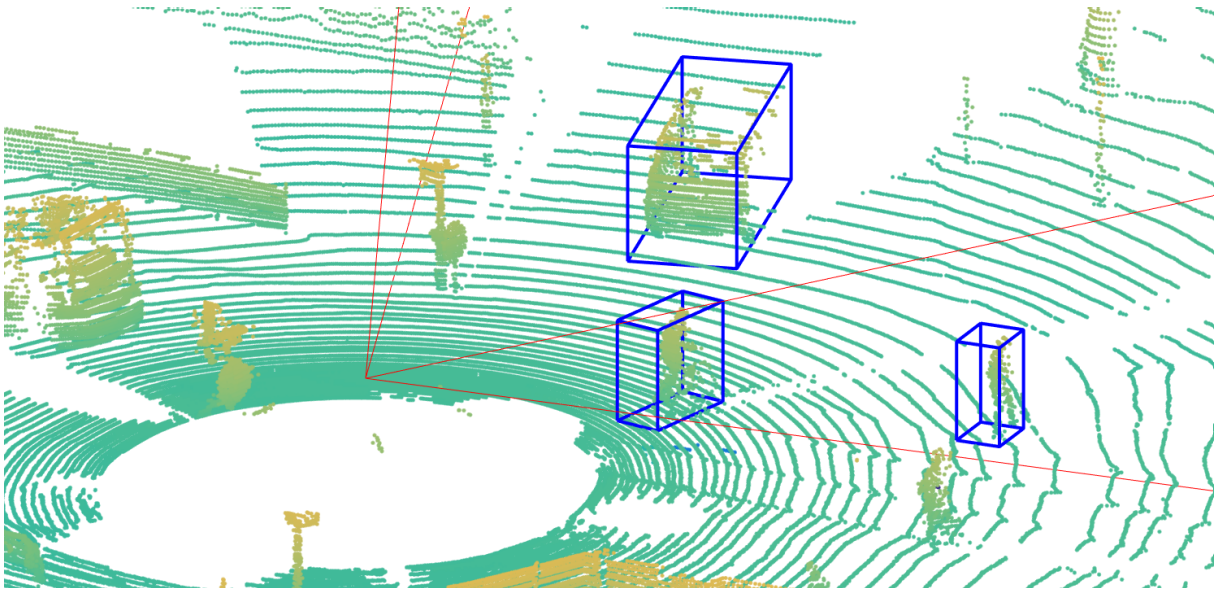
Právě množství sítí schopných vysoce kvalitní detekce bylo motivem k jejich využití k učení sítě schopné segmentace. Jelikož trénovací data pro sémantickou segmentaci je pracné vytvářet ručně, rozhodli jsme se zjistit zda se dají smysluplně nahradit uměle vytvořenými daty.

K tomu jsem vytvořil algoritmus vytvářející 2D segmentovaná data z množiny 3D bodů a bounding boxů. Tyto data jsem použil k přeučení veřejně dostupné plně konvoluční sítě a srovnal s výsledky získanými tou samou sítí naučenou pomocí ručně vytvořených dat. Nakonec jsem vzal segmentovaný obraz získaný naučenou sítí a zpětnou projekcí s ním označil 3D body ve scéně.

5.1 Automatická anotace

Jak již bylo zmíněné v předchozí kapitole, dataset KITTI sice obsahuje velké množství kvalitních, veřejně přístupných dat. Ale obsahuje jen málo anotovaných obrázků, které museli uživatelé anotovat ručně. Proto jsme se rozhodli vytvořit algoritmus pro tvorbu takovýchto segmentovaných obrázků.

Myšlenkou automatické anotace je vzít značky z jiných kategorií KITTI, kde jsou některé objekty označeny pomocí 3D bounding boxů, jako je vidět na obrázku 5.1. Vybral jsem si dataset původně určený pro sledování objektů, kde je bounding boxy vyznačeno 5 různých typů objektů (vozidlo, chodec, cyklista, tramvaj, ostatní), což mi včetně pozadí dává 6 tříd pro segmentaci.



Obrázek 5.1. 3D scéna s bounding boxy s vyznačením zorného pole kamery

Prvním krokem je z bounding boxů získat label pro každý bod ve 3D scéně. Proto je potřeba u každého bodu rozhodnout zda leží v nějakém bounding boxu, případně v jakém. K tomuto účelu jsem implementoval funkci *label2truth.m*. Matlab sice disponuje funkcí *inpolygon()*, která vrací logické hodnoty podle toho zda zadaný 2D bod leží v zadaném 2D regionu, ale nemá žádný ekvivalent této funkce pro 3D body a 3D objekty. Toto se ale dá obejít pomocí kombinace funkcí *delaunay()* a *tsearchn()*. Funkce *delaunay()* vytvoří Delaunayho triangulaci[1] z bodů tvořících rohy bounding boxu. Funkce *tsearchn()* pak pro každý boc vrací index simplexu který ho obklopuje. Nás sice nezajímá který konkrétní simplex z původního bounding boxu daný bod obklopuje, ale zajímá nás, že funkce vrací číslo pokud se bod nachází uvnitř, nebo vrací hodnotu *NaN* pokud ho žádný simplex neobklopuje. Určování zda bod náleží určitému bounding boxu se pak vyřeší následujícími dvěma řádky

```
tess=delaunay(corners2);
inside=~isnan(tsearchn(corners2,tess,points));
```

kde *corners2* je matice 8×3 s rohy bounding boxu, *points* je matice $N \times 3$ obsahující 3D body ve scéně a *inside* je logický vektor $N \times 1$, který u každého bodu říká, zda je v daném bounding boxu, či nikoliv.

Následně je vytvořen vektor *truth* o stejné velikosti jako *inside* který obsahuje číslo skupiny kterou bounding box obsahoval. Celá funkce běží iterativně pro každý bounding box, takže vektor stačí inicializovat při načtení scény.

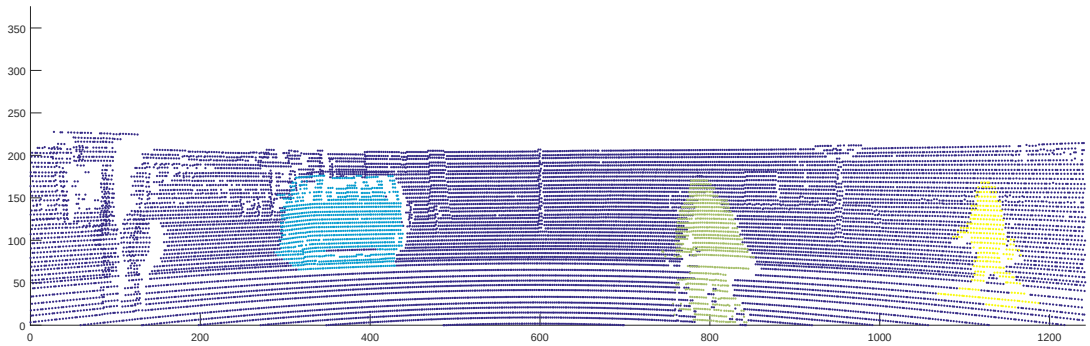
```
truth=zeros(length(points),1);
```

A následně updatovat pro každý bounding box pomocí jednoduchých podmínek jako:

```
if (strcmp(tracklets{i}(j).type ,'Pedestrian'))
truth(inside)=3;
end
```

kde *tracklets(i)(j)* je *j*-tý bounding box v *i*-té scéně.

Když jsou všechny body označeny, můžeme si prohlédnout 3D scénu z pohledu kamery 5.2. Protože známe kalibrační matici kamery a polohu kamery vůči Velodyne sensoru, můžeme nastavit pohled přesně odpovídající tomu, co vidí skutečná kamera 5.3



Obrázek 5.2. 3D scéna z pohledu levé kamery

[1] mathworks.com, Delaunayho triangulace `https://www.mathworks.com/help/matlab/math/delaunay-triangulation.html`

Obrázek 5.3. Původní obrázek kamery

Nyní stačí tento pohled přetvořit v jednovrstvý PNG obrázek, kde bude hodnota každého pixelu odpovídat jeho třídě. K tomu jsem implementoval funkci *velo2segment.m* jejímiž vstupy jsou cesta k 3D scénám z velodyne senzoru a cesta k jim příslušným MAT souborům s vektory *truth* získanými předchozí funkcí. Tato funkce odstraní všechny body nacházející se za kamerou a následně zbylé body promítne pomocí kalibrační matice $K(3 \times 4)$.

```
proj=(K*T*points')';
proj = proj(:,1:2)./repmat(proj(:,3),1,2);
```

kde *points* je matice $N \times 4$ s homogenizovanými 3D body scény, a T je transformační matice 4×4 posouvající body ze souřadného systému Velodyne senzoru do souřadného systému kamery. *proj* jsou 2D souřadnice bodů v zorném poli kamery. Z nich jsou odstraněny všechny body sahající mimo rozlišení kamery

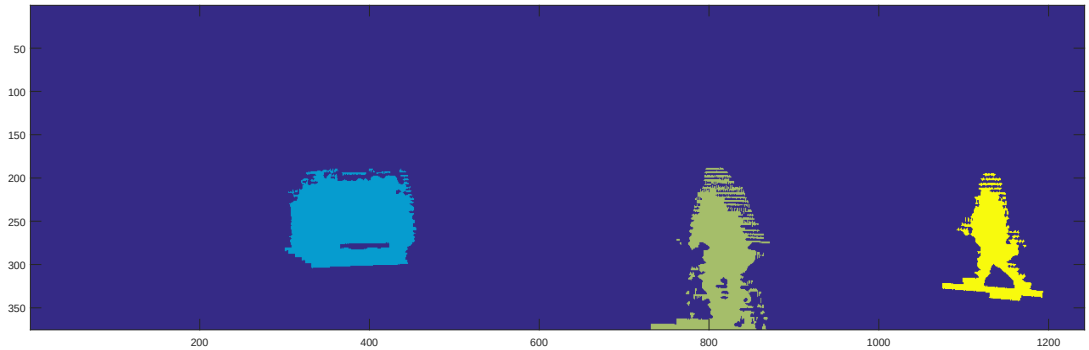
```
inside = proj(:,1) < imageSize(2);
inside = and(inside, proj(:,2) < imageSize(1));
inside = and(inside, proj(:,1) >= 0);
inside = and(inside, proj(:,2) >= 0);

proj=[proj(inside,:),truth(inside,:)];
```

tyto body jsou nakonec interpolovány pomocí matlabovské funkce *griddata()* s parametrem *'nearest'* která pokryje matici reprezentující segmentovaný obrázek pixely se stejnou hodnotou jako jim nejbližší bod z *proj*

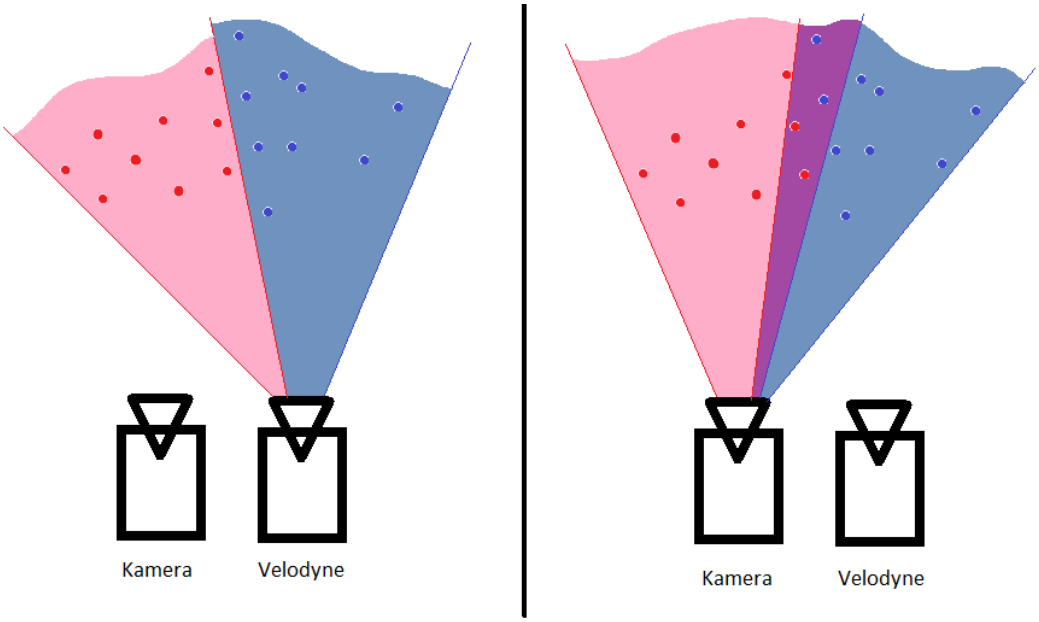
```
[Xq,Yq]=meshgrid(1:imageSize(2),1:imageSize(1));
Vq=griddata(proj(:,1),proj(:,2),proj(:,3),Xq,Yq,'nearest');
```

Příklad vzniklého segmentovaného obrázku je vidět na5.4



Obrázek 5.4. Automaticky anotovaný obrázek

Okraje objektů se zdají být rozostřené, protože se zde mísí body různých skupin. Pro senzor Velodyne žádné překrývající se body neexistují, ale pokud se na ty samé body dívá kamera, která je vůči Velodyne senzoru posunuta, může na okrajích objektů vidět body, které jsou až dál za objektem. Situaci zjednodušené ilustruje obrázek5.5, kde je fialově znázorněna oblast mísících se bodů.

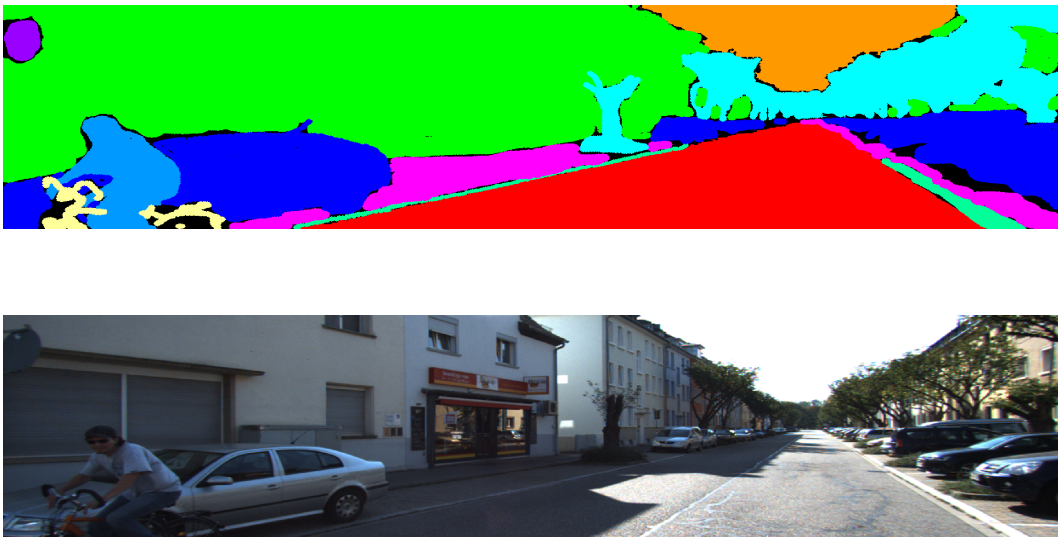


Obrázek 5.5. Ilustrace mísení skupin pro kameru posunutou vůči velodyne senzoru

5.2 Učení FCN

Cílem automatické anotace bylo získání většího množství trénovacích dat. Tato data jsou přirozeně mnohem méně kvalitní než data vytvořená ručně, jak je vidět na obrázcích 5.6 5.7.

Obrázek 5.6. Automaticky anotovaný trénovací obrázek

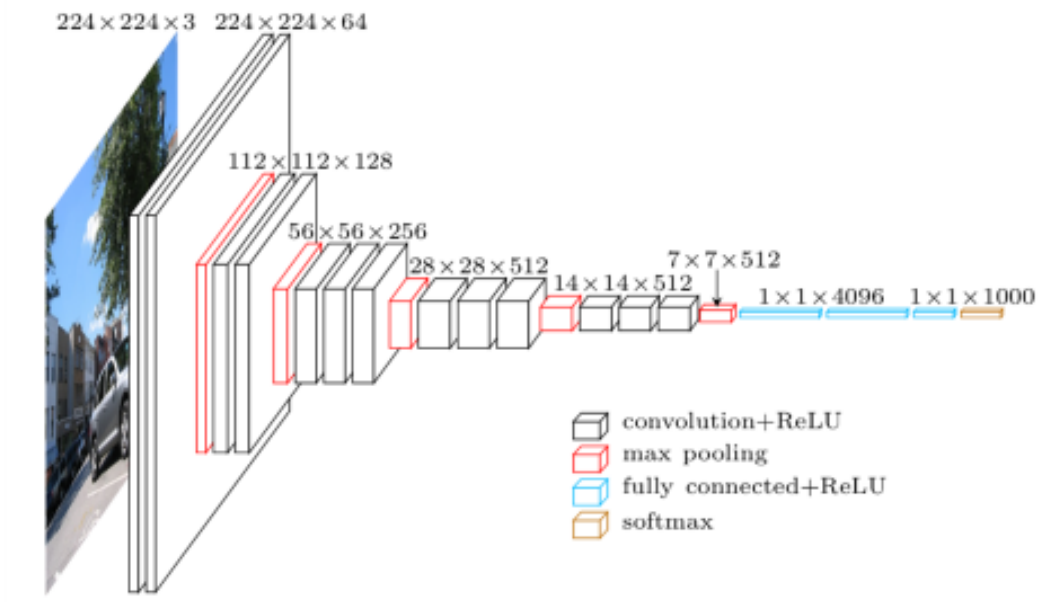


Obrázek 5.7. Manuálně anotovaný trénovací obrázek

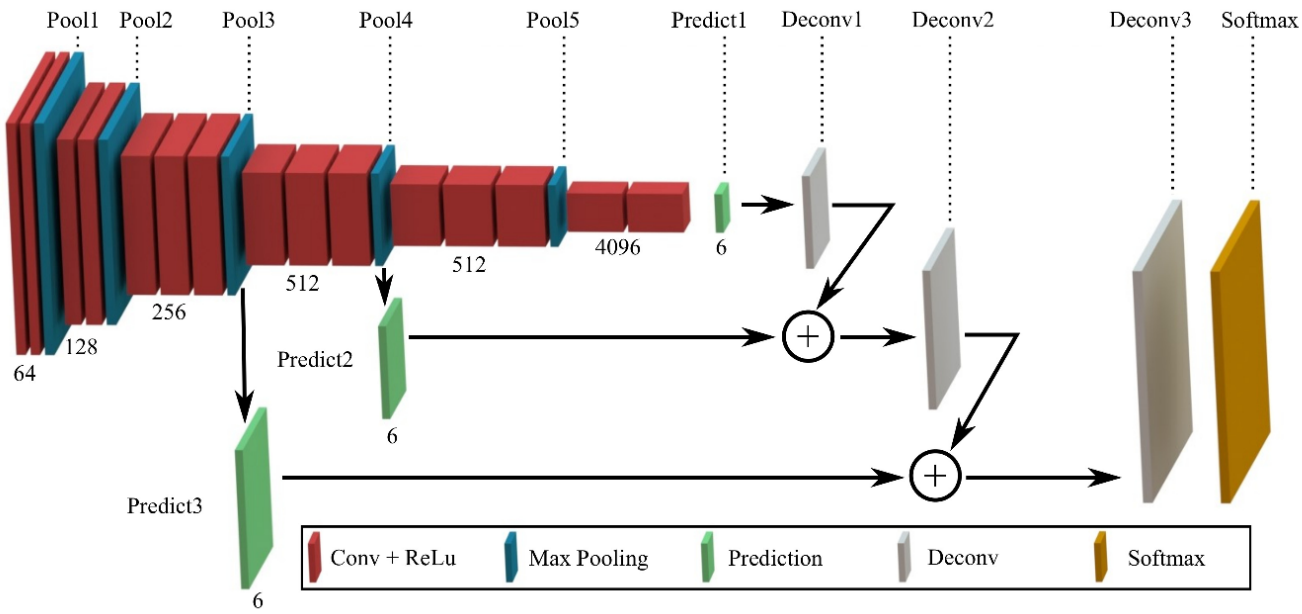
Takovýchto anotací je ale jenom kolem dvou set, zatímco díky automatické anotaci jsem z datasetu *Object Tracking*[1] získal přes sedm tisíc segmentovaných obrázků. S nadsázkou by se dalo říct že některé anotace nemají daleko do metod *Weak labeling*[7], ve kterých se síť neučí pod silným dohledem labelingu na úrovni pixelů, ale pod slabším , ale mnohem snadněji získatelným dohledem „slabých značek“. Ty neurčují konkrétní pixely, ale pouze říkají, že se na obrázku nachází určitý objekt.

[1] cvlibs.net, Object Tracking Evaluation `http://www.cvlibs.net/datasets/kitti/eval_tracking.php`

Síť použitá k učení na získaných datech je založena na plně konvoluční architektuře použité J. Longem[4]. Která čerpá z architektur AlexNet[3], VGG[10] a také GoogLeNet[11]. J.Long ze zmíněných sítí odřízl plně propojené vrstvy a nahradil je konvolučními, aby tyto sítě byly schopné sémantické segmentace. Navíc ještě vylepšil rozlišovací schopnosti sítě propojením hrubších hlubokých vrstev nesoucích sémantickou informaci s mělčími vrstvami s vyšším rozlišením, nesoucích informaci o vzhledu.



Obrázek 5.8. Architektura sítě VGG16[10][1]



Obrázek 5.9. Architektura použité sítě[2]

[1] cs.torronto.edu, VGG in TensorFlow `https://www.cs.toronto.edu/~frossard/post/vgg16/vgg16.png`

[2] semanticscholar.org, PCA-aided Fully Convolutional Networks for Semantic Segmentation of Multi-channel fMRIhttps://www.semanticscholar.org/paper/PCA-aided-Fully-Convolutional-Networks-for-Semanti-Tai-Ye/05d20ad124a8696f387e6c9632dec0b31251df64

Na obrázku 5.8 je vidět architektura původní sítě VGG16 a na obrázku 5.9 je vidět architektura sítě použité v této úloze.

Tuto síť jsem nejdříve naučil na manuálně anotovaném souboru dat (*cca. 200 obrázků*). Pak na automaticky anotovaném souboru dat obdobné velikosti (*cca. 250 obrázků*) a nakonec na řádově větším souboru automaticky anotovaných dat (*cca. 7000 obrázků*). Za účelem zjistit zda mohou být emanuálně anotovaná data efektivně nahrazena dostatečným množstvím automaticky anotovaných dat.

Data	Mean accuracy	Pixel accuracy	Mean IoU
MA200	0.3070	0.6948	0.2358
AA250	0.2629	0.4522	0.1705
AA7000	0.3011	0.9492	0.2600

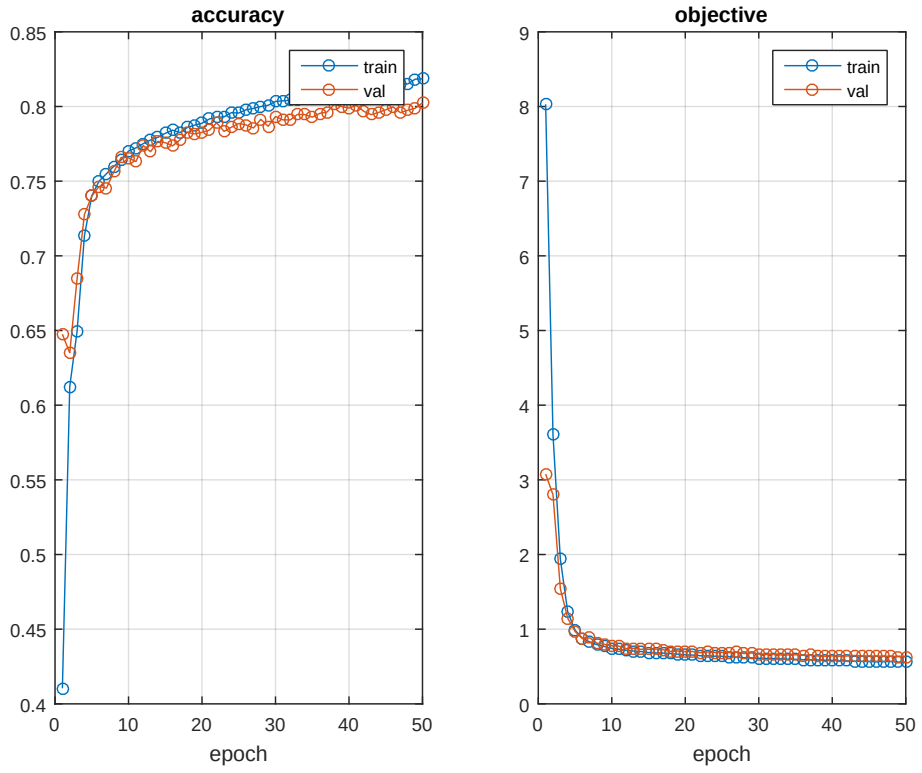
Tabulka 5.1. výsledky přeučení sítě

Tabulka 5.1 ukazuje srovnání výsledků učení sítě na manuálně anotovaných datech (*MA200*), malém množství automaticky anotovaných dat (*AA250*) a větším množství automaticky anotovaných dat (*AA7000*). Před učením sítě byly tyto datasety rozděleny na trénovací a testovací podmnožiny v poměru 90 : 10. Srovnávané veličiny jsou průměrná dosažená přesnost segmentace, celková přesnost pixelů a průměrný „průnik nad sjednocením“ (*angl. Intersection over Union*[1]).

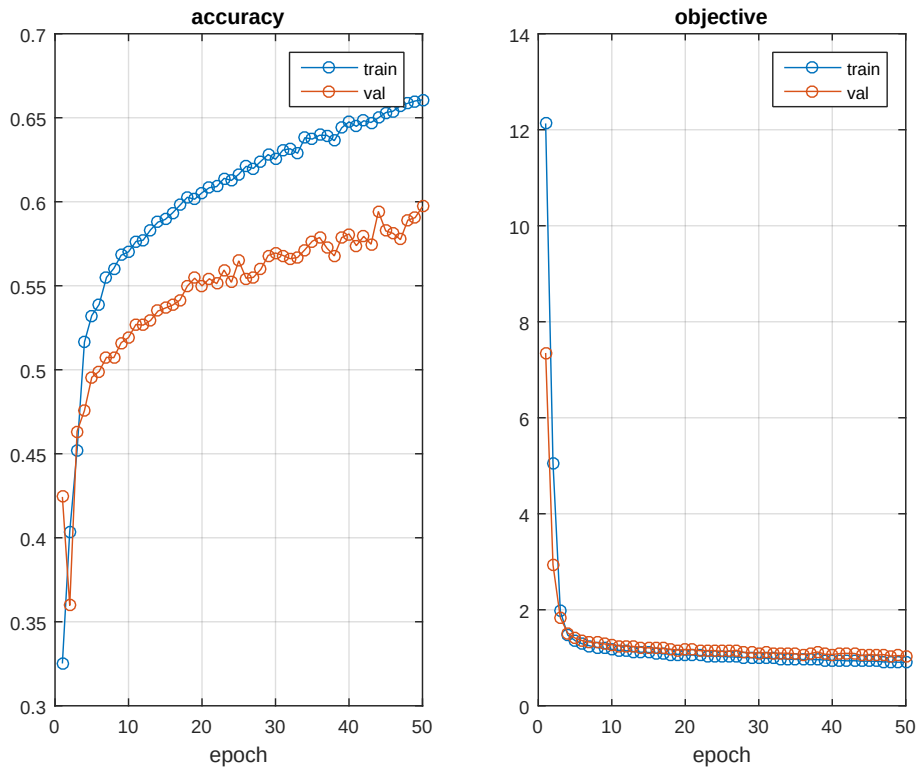
Podle očekávání dosáhla síť lepších výsledků s manuálně získanými daty. Ale výše dosažené přesnost během učení dat sahající k 0.80 u manuální anotace5.10, respektive 0.60 u automatické anotace5.11 naznačuje, že může docházet k přeučení (*overfitting*). K přeučování obecně dochází, když mám relativně velké množství parametrů na malé množství dat. Tento případ by mohl nastat při učení sítě o velikosti okolo 0.9GB na datech o celkovém objemu okolo 160MB.

Síť naučená na větším množství automaticky anotovaných dat už si v testech vede srovnatelně se sítí z manuálních dat, což ukazuje, že těžko získatelná kvalitní učící data se dají nahradit dostatečným množstvím méně kvalitních, avšak snáze získatelných dat. Na obrázku 5.12 je vidět příklad segmentace takto naučenou sítí. Kde síť správně označí většinu chodců, ale milně označuje část cyklisty a zábradlí v pravé části obrazu za auto.

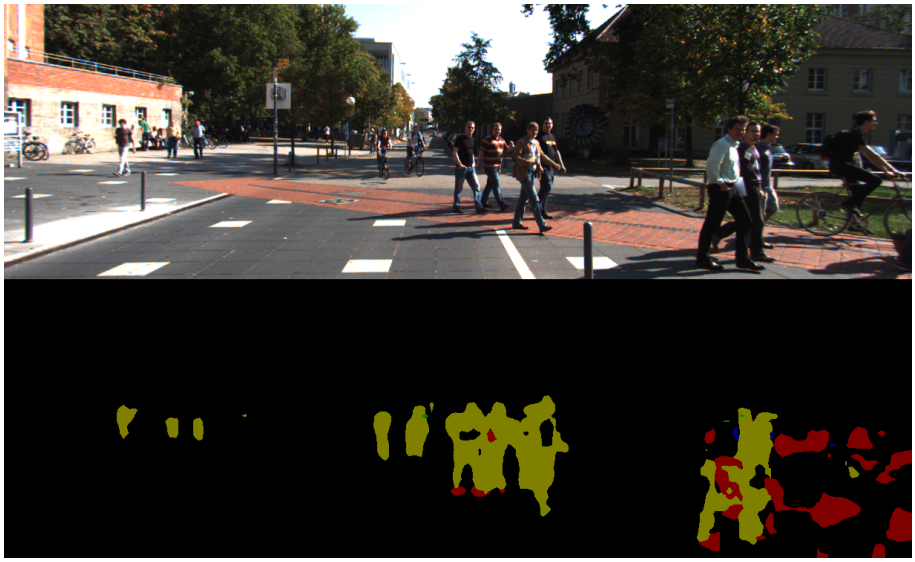
[1] pyimagesearch.com, Intersection over union `http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`



Obrázek 5.10. vývoj přesnosti a ztrátové funkce během učení pomocí manuálně anotovaných dat



Obrázek 5.11. vývoj přesnosti a ztrátové funkce během učení pomocí malého množství automaticky anotovaných dat



Obrázek 5.12. Výstup sítě naučené pomocí automaticky anotovaných dat

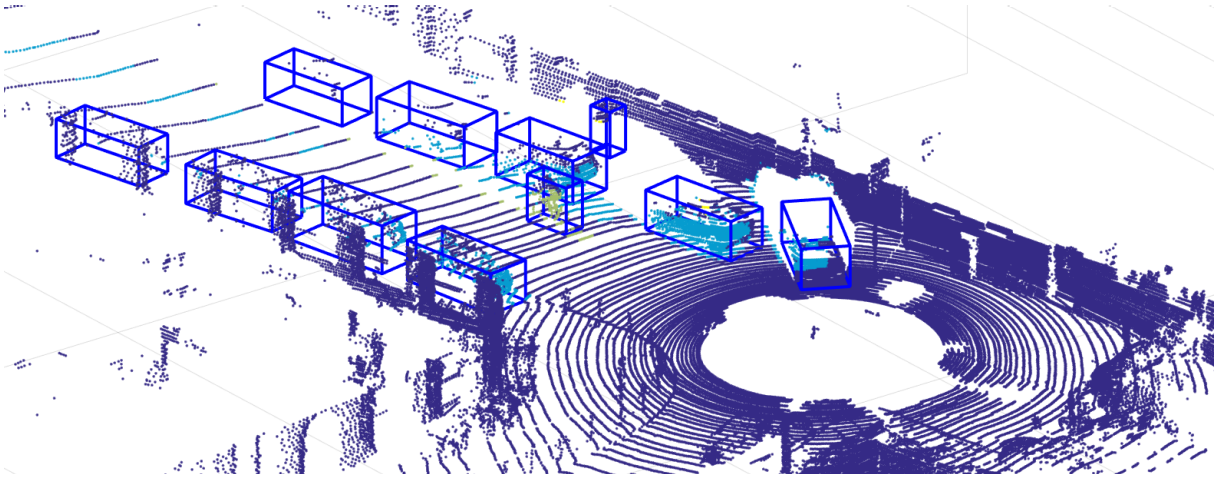
5.3 Segmentace prostředí

S naučenou sítí teď můžeme segmentovat pohled z kamery a segmentovaný obraz promítat zpět na 3D prostředí. Původní záměr byl vytvořit označenou mapu využívající kumulativní informaci z po sobě následujících snímků z vozu. To by bylo možné právě se značkami z manuálních anotací, které obsahují i třídy jako silnice, chodník, dům, vegetace, dopravní značky a jiné statické objekty, které má smysl anotovat ve 3D mapě. Bohužel dataset ze kterého mohla čerpat moje automatická anotace obsahovala pouze třídy vozidla, cyklisty a chodce.

Postup je obdobný jako u automatické anotace. Body ve scéně se promítnou do 2D. Ale tentokrát se jejich 2D souřadnice použijí jako indexy matice reprezentující segmentovaný výstup ze sítě. Tím se získá jejich sítí určená třída.

```
label(i)=anno(ceil(proj(i,2)),ceil(proj(i,1)));
```

kde *proj* je opět matice promítnutých bodů, *anno* je sítí anotovaný obraz a i je pořadí bodu ve scéně



Obrázek 5.13. Projekce anotovaného obrazu do scény

Problém s překryvem značek popsaný v sekci Automatické anotace se projevuje i při zpětné projekci, když je za vozidlo či cyklistu označován i prostor za nimi. Tento problém, i ekvivalentní problém v automatické anotaci nechávají prostor pro vylepšení. Také zmiňované rozšíření o hloubkový kanál obrazu a využití jeho informace při segmentaci dává námět k další práci.

Kapitola 6
závěr

Tato práce se věnovala problematice konvolučních neuronových sítí. Na začátku byl popsán jejich původ. Pak byla objasněna jejich struktura, princip fungování a jednotlivé vlastnosti včetně výhod a nevýhod oproti standardním neuronovým sítím. Následně bylo představeno několik způsobů jejich využití.

Před praktickou částí je ještě vyjmenováno několik frameworků pro implementaci těchto sítí a byla zmíněna možnost výpočtu pomocí grafického procesoru, která se uplatňuje čím dál častěji vzhledem k výpočetnímu výkonu a dostupnosti dnešních grafických karet.

V následující části práce je popsán dataset KITTI a implementace algoritmu automatické anotace. Jsou zde zmíněny i problémy spojené s jeho činností. Dále byla představena síť použitá pro 2D sémantickou segmentaci a naučena jak na automaticky, tak na manuálně anotovaných datech. Byly zde prezentovány a prodiskutovány její výsledky. Následně byl popsán způsob segmentace okolí pomocí zpětné projekce.

Původní záměr použití sítě pro segmentaci RGBD dat nebyl úplně splněn. Dataset KITTI nabízí jen RGB obraz a jemu příslušné mračno bodů. Hloubkový kanál by bylo třeba získat uměle projekcí 3D bodů na obraz a následnou šikovnou interpolací jejich hloubkových hodnot. Tedy obdobným způsobem jakým byla vytvořena trénovací data. Místo toho je segmentace 3D scény docílena opačnou projekcí 2D segmentovaného obrazu do prostoru. Přínos této práce je tedy spíše využití 3D bounding boxů k učení konvoluční sítě pro 2D sémantické segmentaci. J. Dai, K. He a J. Sun[14] prezentovali podobnou myšlenku s využitím 2D bounding boxů a iterativní úpravy trénovacího obrazu. Moje metoda vytváří přímo „Ground Truth“ trénovací obraz. Provedený test dále ukazuje, že automaticky anotovanými daty v dostatečném množství lze dosáhnout stejných výsledků jako s manuálně anotovanými. Rozsáhlejší testování bohužel nebylo možné vzhledem ke zdroji motivace pro tuto práci, nedostatku manuálně anotovaných dat.

Literatura

[1] BOUVRIE J. Notes on Convolutional Neural Networks [online] 2006, [cit. 22. 11. 2014].http://cogprints.org/5869/1/cnn_tutorial.pdf

[2] FUKUSHIMA, Kunihiko. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position [online]. [cit.2015-05-17]. http://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf

[3] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey HINTON. ImageNet Classification with Deep Convolutional Neural Networks [online]. 2012 [cit.2014-12-15] http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

[4] Jonathan Long, Evan Shelhamer, Trevor Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR, 2015. https://arxiv.org/abs/1411.4038

[5] J. Ren, X. Chen, J. Liu, W. Sun, J. Pang, Q. Yan, Y. Tai and L. Xu: Accurate Single Stage Detector Using Recurrent Rolling Convolution. CVPR 2017. https://arxiv.org/pdf/1704.05776.pdf

[6] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teuliere and T. Chateau: Deep MANTA: A Coarse-to-fine Many-Task Network for joint 2D and 3D vehicle analysis from monocular image. CVPR 2017. https://arxiv.org/pdf/1703.07570.pdf

[7] Pathak et al. Constrained Convolutional Neural Networks for Weakly Supervised Segmentation, ICCV, 2016 https://arxiv.org/abs/1506.03648

[8] Camille Couprie et al. Indoor Semantic Segmentation using depth information. [2013-3-14] https://arxiv.org/pdf/1301.3572.pdf

[9] Redmon, Joseph and Farhadi, Ali, YOLO: Real-Time Object Detection, 2016 https://pjreddie.com/darknet/yolo/

[10] K.Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR,abs/1409.1556, 2014 https://arxiv.org/abs/1409.1556

[11] C.Szegedy, W.Liu, Y.Jia, P.Sermanet, S.Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842,2014.

[12] Z. Wu, S. Song, A. Khostla, F. Yu, L. Zhang, X. Tang, J. Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes, [15-4-2015] https://arxiv.org/pdf/1406.5670.pdf

[13] C. B. Choy, D. Xu, J. Gwak, K. Chen, S. Savarese. 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction, Stanford University [2-4-2015] https://arxiv.org/pdf/1604.00449.pdf

[14] J. Dai, K. He a J. Sun. BoxSup: Exploiting Bounding Boxes to Supervise convolutional Networks for Semantic Segmentation,[18-5-2015] https://arxiv.org/pdf/1503.01640.pdf

[15] T. D. Kulkarni, W. F Whitney, P. Kohli, J. B. Tenenbaum. Deep Convolutional Inverse Graphics Network.http://papers.nips.cc/paper/5851-deep-convolutional-inverse-graphics-network.pdf

[16] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, T. Brox. Learning to Generate Chairs, Tables and Cars with Convolutional Networks. https://pdfs.semanticscholar.org/e47e/988e6d96b876bcab8ca8e2275a6d73a3f7e8.pdf

Příloha A
DVD

Součástí přiloženého DVD jsou následující položky:

- Text této práce `Dp_2017_hobza_martin.pdf`.
- Zdrojové kódy pro matlab v archivu `matconvnet_essentials.zip`.
- net-epoch-50.mat plně konvoluční síť naučenou na velkém množství automaticky anotovaných dat