VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE VÝZNAČNÝCH BODŮ V OBRAZECH VOZIDEL

DETECTION OF LANDMARKS ON VEHICLE IMAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VOJTĚCH CHADIMA**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Chadima Vojtěch**
Program: Informační technologie
Název: **Detekce význačných bodů v obrazech vozidel**
Detection of Landmarks on Vehicle Images
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou sledování dopravy pomocí stacionárních kamer.
2. Prostudujte problematiku kalibrace kamery (vnitřní i vnější parametry).
3. Prostudujte problematiku detekce význačných bodů na obrazech vozidel (landmarks) pomocí konvolučních neuronových sítí.
4. Pořiďte a/nebo generujte vhodná data pro učení neuronových sítí pro detekci význačných bodů v obrazech vozidel.
5. Naučte neuronovou síť pro detekci význačných bodů v obrazech vozidel.
6. Zhodnoťte vlastnosti a úspěšnost natrénované neuronové sítě.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 9. dubna 2019

Abstrakt

Tato práce řeší automatickou detekci význačných bodů na obrázcích automobilu. Takto detekované význačné mohou dále sloužit k automatické kalibraci kamery, například pro dohled v dopravě, což je problém, po jehož vyřešení je možné kameru využít v aplikacích jako měření rychlosti vozidel či hustoty dopravy.

K detekci význačných bodů jsem použil konvoluční neuronovou síť typu Stacked Hourglass. Dále byl vytvořen generátor trénovacích dat v podobě obrázku a odpovídající anotace využívající API Blenderu, který umožňuje vytváření datasetů pro libovolné objekty. Detekované význačné body jsem analyzoval a seřadil dle přesnosti jejich detekce, přičemž platí, že čím přesněji je bod na snímku detekovatelný, tím je vhodnější pro použití při úlohách typu kalibrace kamery.

Podařilo se natrénovat modely neuronových sítí, které jsou schopny detekovat 1 021 význačných bodů, z nichž nejlepších 24 s průměrnou odchylkou menší než 3 pixely.

Výsledky této práce jsou základem pro kalibraci kamery na základě rozpoznání nejvhodnějších význačných bodů, případně mohou dále sloužit k vytváření vlastních trénovacích datasetů a trénování vlastních modelů neuronových sítí typu Stacked Hourglass.

Abstract

This thesis aims to introduce automatic detection of landmarks on vehicle images. Detected landmarks can be then used for automatic traffic surveillance camera calibration or other computer vision applications.

I solved the landmarks detection problem by using a novel type of convolutional neural network called Stacked Hourglass. Furthemore, I created an automatic trainig dataset (image + anotations) generator based on Blender API, which allows to create various datasets. Detected landmarks are analyzed and sorted in order to determine a set of superior landmarks that could be later used for camera calibration.

The best-performing models detect up to 1 021 landmarks, while the best of them have less than 3.0 pixels average error.

Finally, results can be further used in automatic camera calibration based on landmarks detection, to create custom datasets or to train Stacked Hourglass convolutional neural networks.

Klíčová slova

Stacked Hourglass, detekce význačných bodů, Blender, strojové učení, konvoluční neuronové sítě, kalibrace kamery

Keywords

Stacked Hourglass, detection of landmarks, Blender, machine learning, convolutional neural networks, camera calibration

Citace

CHADIMA, Vojtěch. *Detekce význačných bodů v obrazech vozidel.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Detekce význačných bodů v obrazech vozidel

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. Ing. Adama Herouta, PhD. Další informace mi poskytl Ing. Petr Dobeš. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Chadima
15. května 2019

Poděkování

Děkuji svému vedoucímu prof. Ing. Adamu Heroutovi, Ph.D., za odbornou i inspirativně lidskou stránku vedení práce. Dále děkuji Ing. Petru Dobešovi za cenné rady a Ing. Michalu Hradišovi, Ph.D. za úvod do problematiky konvolučních neuronových sítí. Poděkování patří v neposlední řadě i mým blízkým.

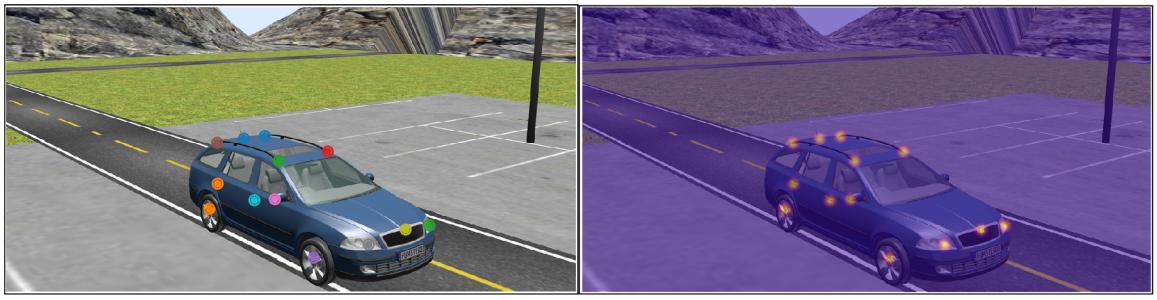
Obsah

Kapitola 1

Úvod

Cílem této práce je automaticky detekovat význačné body na obrázcích automobilu pomocí metod strojového učení. Takto detekované význačné body potom mohou sloužit ke kalibraci dopravní dohledové kamery pomocí nového postupu navrženého Bartlem [1]. V kapitole 6 tohoto článku je jako jedno z možných budoucích vylepšení uveden právě lepší výběr a detekce význačných bodů, což bylo mým hlavním záměrem.

Potenciálním význačným bodem použitelným ke kalibraci je kterýkoliv viditelný bod na povrchu automobilu. Takových bodů je tedy značné množství; vzájemně se ovšem odlišují přesností, s jakou je možno je detekovat. Pro co nejlepší výsledky kalibrace je tedy zapotřebí určit, které body jsou nejsnáze a nejpřesněji detekovatelné, a s nimi poté kalibraci provádět. Tato rozdílná míra přesnosti detekce je přirozená: kupříkladu střed zrcátka je pro lidské oko a s velkou pravděpodobností neuronovou síť snáze rozpoznatelný, než bod nacházející se uprostřed rozlehlé plochy kapoty. Je tedy nutné vybrat nejvhodnější význačné body z obrovské množiny bodů potenciálních. Podstata této práce je naznačena na obrázku 1.1. Dalšími produkty práce jsou skript pro generování trénovacích dat pro neuronovou síť typu Stacked Hourglass využívající Blender API, sady trénovacích dat s anotacemi, program umožňující trénovat neuronové sítě a také samotné natrénované modely sítí.



Obrázek 1.1: **Vlevo:** původní vyrenderovaný obrázek s několika znázorněnými význačnými body, **vpravo:** původní obrázek překrytý vizualizací výsledných pravděpodobnostních matic udávajících polohu jednotlivých význačných bodů. Tyto pravděpodobnostní matice jsou výstupem natrénovaného modelu neuronové sítě typu Stacked Hourglass, který detekuje polohy jednotlivých význačných bodů.

Kapitola 2

Konvoluční neuronové sítě a použité technologie

Jádrem práce je trénování neuronové sítě typu Stacked Hourglass pro detekci význačných bodů na obrazech automobilů, které mohou být použity ke kalibraci dopravní dohledové kamery. První část této kapitoly je tedy věnována kalibraci, druhá poté konvolučním neuronovým sítím s důrazem na síť typu Stacked Hourglass, která je dle Newella [7] vhodná právě pro detekci význačných bodů. Závěr kapitoly popisuje technologie a nástroje využité při řešení implementační části práce.

2.1 Kalibrace kamery

Vzhledem k tomu, že výsledek této práce má sloužit ke zlepšení kalibrace dopravní dohledové kamery, je tato kapitola věnována právě kalibraci. Kalibrace znamená určení vnějších a vnitřních parametrů kamery (viz následující odstavec), které jsou poté použity ke korigování zkreslení. Kalibrace samotná je rovněž základem pro analýzu a interpretaci dat z kamer, v případě dohledových dopravních kamer např. pro měření rychlosti jednotlivých automobilů, hustoty dopravy apod.

Kalibraci kamery popisuje následující vztah: pro bod v 3D prostoru $P = [X, Y, Z, 1]^T$ a jemu odpovídající bod $p = [u, v, 1]$ v 2D rovině platí

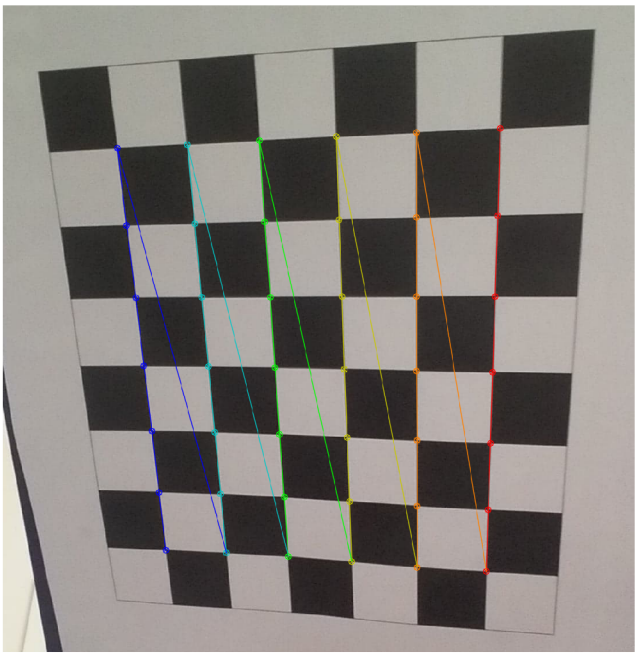
$$s\mathbf{p} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} P, \tag{2.1}$$

kde $\mathbf{R}$ a $\mathbf{t}$ jsou vnější parametry kamery, s je faktor změny měřítka (scale factor), $\mathbf{R}$ reprezentuje rotační matici kamery a $\mathbf{t}$ translační vektor. Tvar matice reprezentující vnitřní parametry kamery je

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

Člen γ představuje koeficient šikmosti (je nenulový v případě, že osy obrázku nejsou navzájem kolmé), u_0 a v_0 jsou souřadnice optického středu (principal point) a α a β reprezentují ohniskovou vzdálenost.

Klasickou kalibrační metodou, kterou představil Zhang [14], je užití určitého známého vzoru, jehož snímky jsou kamerou z různých úhlů pohledu pořizovány. Tuto metodu vystihuje obrázek 2.1, který jsem získal při svých prvotních experimentech s kalibrací webové



Obrázek 2.1: Klasická metoda kalibrace kamery pomocí užití a rozpoznání známého vzoru (v tomto případě šachovnice), kterou jsem si vyzkoušel pro počáteční seznámení s technikami kalibrace.

kamery. Tou jsem pořídil několik snímků šachovnice a následně kalibrační algoritmus na základě rozpoznané polohy význačných bodů (v tomto případě rohy jednotlivých políček) a jejich vzájemné polohy ve skutečnosti (kolmost apod.) provedl kalibraci.

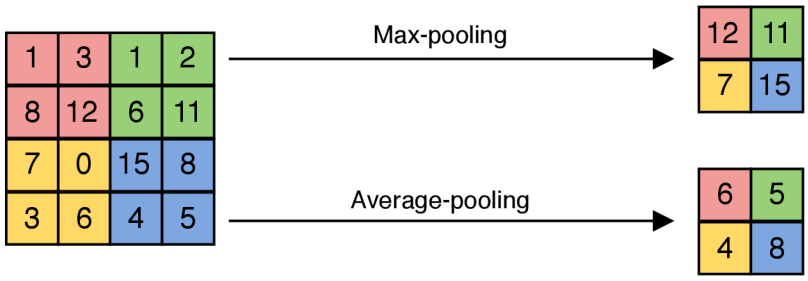
Tato klasická metoda ovšem není vhodná právě pro kalibraci dopravní kamery – v praxi by znamenala nutnost každé kameře ukazovat určitý vzor, což je vzhledem k množství a umístění kamer značně problematické.

Nová metoda kalibrace navržená Bartlem [1] pro tyto účely umožňuje provést kalibraci kamery na základě rozpoznání význačných bodů na obrázku automobilu a znalosti jejich vzájemné prostorové polohy. Postup kalibrace dopravní dohledové kamery je potom následující:

- Rozpoznání konkrétní značky a modelu automobilu, pro který je kalibrační algoritmus nastaven.
- Rozpoznání několika (až desítek) význačných bodů na automobilu, které poslouží jako vstupy kalibračního algoritmu.
- Kalibrační algoritmus na základě polohy význačných bodů na snímku a jejich skutečné prostorové vzdálenosti (algoritmus má k dispozici přesný 3D model automobilu) provede kalibraci.

2.2 Konvoluční neuronové sítě

Tato kapitola čerpá především z knih Khana [6] a Bhardwaje [2]. Pro zpracování dat jako jsou obrázky jsou jedním z nejpoužívanějších typů neuronových sítí sítě konvoluční. Jsou



Obrázek 2.2: Operace max-pooling a average-pooling s krokem o velikosti 2 a velikostí oblasti 2×2. **Vlevo:** vstupní matice, **vpravo:** výsledné matice po provedení operace pooling.

charakteristické konvolučními filtry, kterými jsou vstupní data konvoluována (v kontextu této práce budou za vstupní data považovány obrázky). Architektura konvolučních neuronových sítí se skládá z vrstev různých typů, o nichž bude řeč v následujících kapitolách. První vrstvou je vstupní vrstva, které jsou poskytnuta výchozí data (např. RGB obrázek). Následující 3 vrstvy se zpravidla několikrát opakují; vznikají tak hluboké neuronové sítě. Ty jsou schopné vstupní obrázek postupně dekomponovat do podoby tzv. příznakového vektoru (feature vector), který uchovává informace o sledovaných vlastnostech obrázku.

2.2.1 Konvoluční vrstva

Konvoluční vrstva je charakteristickou a nejdůležitější vrstvou. Skládá se z jednoho či více filtrů (někdy také konvolučních jader nebo masek), kterými konvoluje svůj vstup. Filtr je matice (typicky o rozměrech 2×2, 3×3 či 5×5), jejíž koeficienty (jednotlivé hodnoty v buňkách matice) jsou na počátku trénování (učení) inicializovány náhodně a v jeho průběhu jsou postupně sítí samotnou upravovány tak, aby poskytovaly co nejpřesnější výsledky.

2.2.2 Pooling vrstva

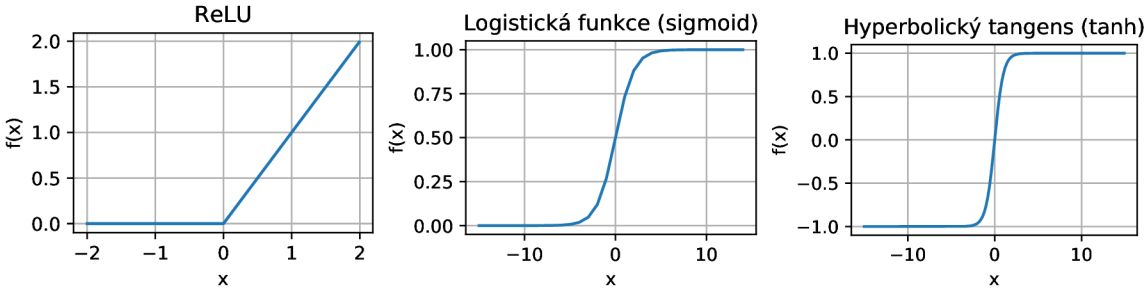
Vrstva typu pooling je využívána ke snížení rozměru vstupních dat (tzv. down-sampling, např. ze vstupního rozlišení 64×64 na výstupní 32×32). Charakteristickými parametry operace jsou velikost oblasti, nad níž je v jedné iteraci pooling prováděn, a krok (stride), který udává posun oblasti mezi jednotlivými iteracemi. Standardně používanými typy jsou max-pooling, který za výslednou hodnotu považuje maximum z oblasti, a average-pooling, jehož výsledkem je aritmetický průměr hodnot z oblasti. Podstata obou typů operace pooling je zachycena na obrázku 2.2.

Rozměry výsledné matice $h' \times w'$ pro vstupní matici o rozměrech $h \times w$ operace pooling s krokem k a rozměry oblasti $o \times o$ jsou určeny následujícím vztahem:

$$h' = [\frac{h - o + k}{k}], w' = [\frac{w - o + k}{k}]. \tag{2.3}$$

2.2.3 Nelineární aktivační funkce

Nelineární aktivační funkce mapuje své vstupy z oboru reálných čísel do určitého přesně vymezeného intervalu (často $\langle -1, 0\rangle$, nebo $\langle 0, 1\rangle$). Za nejjednodušší aktivační funkci splňující tyto požadavky lze považovat jednotkový skok (Heavisideova funkce), která všem záporným číslům přiřadí hodnotu 0 a všem nezáporným 1. Její výstup je tedy binární, což ovšem



Obrázek 2.3: Často používané nelineární aktivační funkce. **Vlevo:** ReLU, **uprostřed:** logistická funkce a **vpravo** hyperbolický tangens.

v kontextu strojového učení není optimální stav, neboť je obecně vhodné rozlišovat mezi více než 2 stavy. Řešením by mohla být lineární funkce. Použití lineární funkce nicméně vede k tzv. lineárnímu mapování, při kterém se ovšem ztrácí výhody použití více vrstev v síti (hluboké učení): takto navržená síť by potom byla nahraditelná sítí s jedinou vrstvou s lineární aktivační funkcí[1]. Z tohoto důvodu se tedy používají nelineární aktivační funkce, které splňují oba požadavky (transformace často neuspořádaných vstupů do hodnoty z přesně vymezeného rozsahu a zároveň nelineárnost).

Mezi nejčastěji využívané nelineární aktivační funkce se řadí **ReLU** s předpisem

$$R(x) = max(0, x), \tag{2.4}$$

dále speciální případ logistické funkce zvaný **sigmoid** definovaný jako:

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{2.5}$$

a **hyperbolický tangens** o předpisu

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.6}$$

Průběhy těchto funkcí jsou znázorněny na obrázku 2.3.

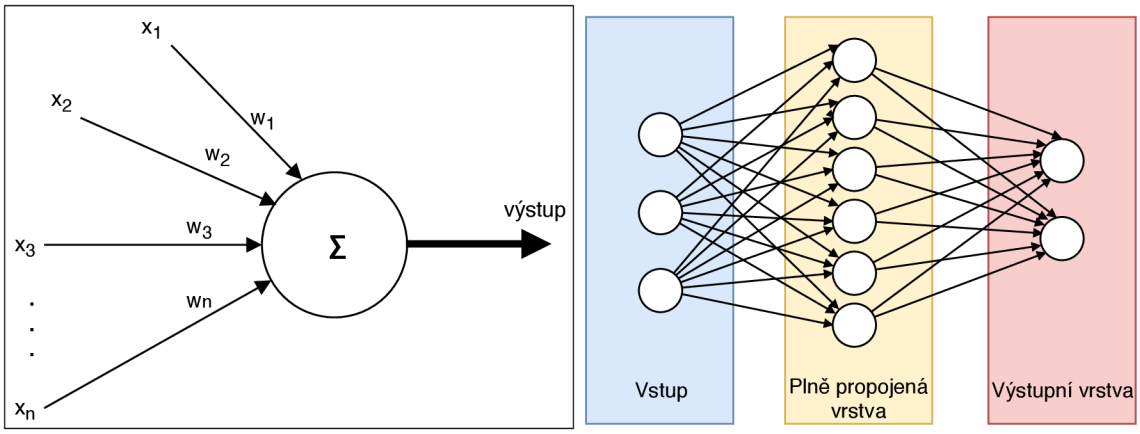
2.2.4 Plně propojená vrstva

Tato vrstva je nejčastěji umístěná na konci sítě. Jejím úkolem bývá poskytovat finální predikce (např. rozhodnutí o třídě, do které vstupní obrázek spadá). Každý tzv. perceptron plně propojené vrstvy je přímo napojen na každý z výstupů vrstvy předcházející. Na obrázku 2.4 je zobrazeno schéma perceptronu a plně propojené vrstvy.

2.2.5 Vrstva pro normalizaci dávky (batch normalisation)

Techniku normalizace dávky (batch normalisation) představil v roce 2015 Ioffe [12]. Její význam spočívá ve vylepšení celkového výkonu a rychlosti, se kterou je model sítě schopen se učit. Použití normalizace dávky rovněž zvyšuje schopnost sítě generalizovat (generalizace znamená schopnost sítě poskytovat korektní výsledky i pro jiná než trénovací data, tj. pro data, která neměla nikdy v minulosti k dispozici).

[1] Avinash Sharma V: Understanding Activation Functions in Neural Networks (2017). medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0



Obrázek 2.4: **Vlevo:** schéma perceptronu (obrázek vytvořen na základě nákresů z článku [13]). Symboly $x_1 - x_n$ značí vstupy perceptronu, koeficienty $w_1 - w_n$ váhy přisuzované jednotlivým vstupům. **Vpravo:** ilustrace plně propojené vrstvy. Každý jeden perceptron z této vrstvy je přímo propojen se všem vstupy i výstupy.

Algoritmus normalizace dávky je následující (zdroj: Ioffe [12]):

Pro dávku $\beta = \{x_1, x_2, ..., x_m\}$, kde $x_1, x_2, ..., x_m$ značí jednotlivé vstupy a m velikost dávky se nejprve vypočítá průměr:

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i, \tag{2.7}$$

a poté rozptyl:

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2. \tag{2.8}$$

Na základě získaného průměru a rozptylu je následně provedena vlastní normalizace podle vzorce

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \tag{2.9}$$

přičemž ϵ je konstanta přidávaná do výpočtu pro zajištění numerické stability. Posledním krokem je určení samotného výstupu vrstvy y_i:

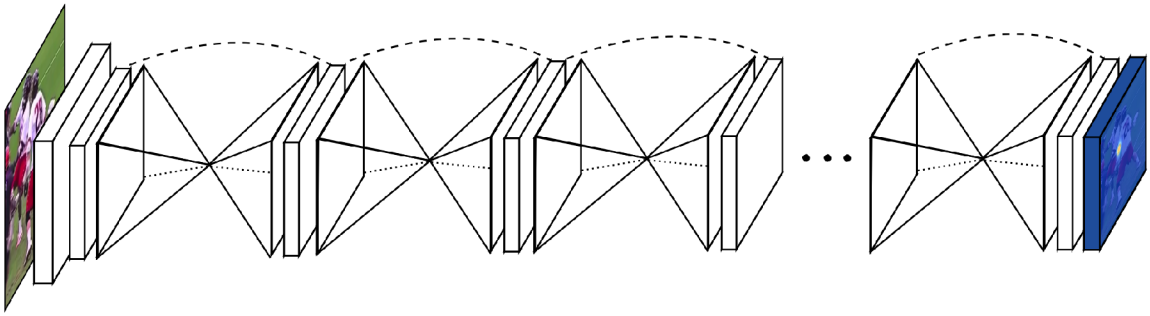
$$\widehat{y_i} = \gamma \widehat{x_i} + \beta. \tag{2.10}$$

Parametry γ – změna velikosti (*scale*) a β – posunutí (*shift*) jsou v průběhu učení trénovaným modelem samotným upravovány.

2.2.6 Konvoluční neuronová síť typu Stacked Hourglass

Newell ve svém článku [7] z roku 2016, ze kterého tato kapitola vychází, přišel s novým typem konvoluční neuronové sítě, který pojmenoval *Stacked Hourglass*. Po konzultaci s vedoucím práce jsem se rozhodl pro detekci význačných bodů použít právě tento typ sítě, protože dosahuje kvalitních výsledků v řešení úloh z oblasti detekce význačných bodů.

Architektura sítě Stacked Hourglass připomíná za sebe poskládané přesýpací hodiny, jak je patrné z obrázku 2.5. Důvod využití tvaru přesýpacích hodin vysvětluje Newell nutností



Obrázek 2.5: Obecné schéma sítě Stacked Hourglass. Vlevo je vstupní obrázek. Ten je následně zpracováván jednotlivými moduly (stacky), které připomínají přesýpací hodiny. Vpravo se nachází výsledná množina pravděpodobnostních matic, přičemž platí, že pro každý detekovaný význačný bod (v případě Newella pro každý detekovaný kloub) existuje právě jedna výsledná matice. Oba obrázky převzaty z [7]

zohledňovat informace (příznaky) obsažené v různých měřítkách (od lokálního úzkého okolí bodu až po celkové rozložení na obrázku). Tyto příznaky jsou sítí v jednotlivých vrstvách vyhodnocovány, kombinovány dohromady a nakonec využity k odhadu pixelových souřadnic daného význačného bodu.

První, zužující se polovina přesýpacích hodin (stacků), se skládá z konvolučních a max-poolingových vrstev, které postupně snižují rozlišení vstupu (tzv. down-sampling) až na rozlišení 4×4. Po dosažení tohoto minimálního rozlišení následuje postup opačný, kdy jednotlivé vrstvy postupně rozlišení zvyšují (up-sampling). Výstupem sítě je pravděpodobnostní matice, pro každou sledovanou entitu (např. konkrétní kloub, nebo význačný bod) jedna.

Tento typ sítě dále využívá i tzv. reziduální moduly, které představil ve svém článku Kaiming [4]. Jak uvádí Olafenwa [8], reziduální modul je charakteristický poskytováním svého výstupu nejen vrstvě bezprostředně následující, ale rovněž i dalším vrstvám v pořadí.

Označíme-li vrstvu sítě l a její vstup x, potom pro výstup y v klasické síti bude platit vztah

$$y = l(x), \tag{2.11}$$

v sítích využívající reziduální moduly je však tento vztah definován takto:

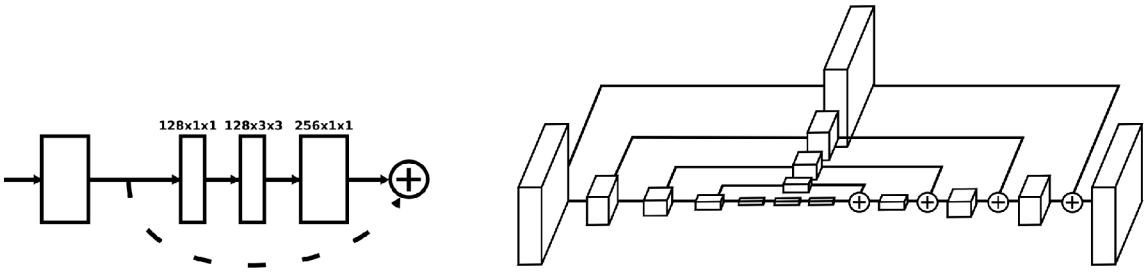
$$y = l(x) + x. \tag{2.12}$$

Konkrétní použití reziduálních modulů v síti Stacked Hourglass ilustruje obrázek 2.6.

2.3 PyTorch

Tato kapitola je založena především na informacích z knihy od Subramaniana [11]. PyTorch[2] je populární open-source platforma pro účely strojového učení v jazyce Python. Byl vytvořen v roce 2016 a je primárně vyvíjen firmou Facebook [9]. Umožňuje navrhovat a trénovat neuronové sítě pomocí konceptů známých z jazyka Python (na rozdíl od některých ostatních knihoven a frameworků pro strojové učení). PyTorch je určen primárně pro

[2] https://pytorch.org/



Obrázek 2.6: **Vlevo:** schéma jednoho reziduálního modulu. **Vpravo:** Schéma znázorňující vnitřní strukturu jednoho Hourglass modulu (stacku). Každý z bloků v tomto schématu představuje jeden reziduální modul. Převzato z [7].

výzkumné účely; není vhodný pro produkční nasazení v případech, kdy je kladen důraz na velmi rychlé poskytnutí výsledků.

Mezi nejdůležitější entity patří tensor, což je obdoba n-rozměrného pole (matice) používaného knihovnou *numpy*; výpočty s tensory je možné provádět na GPU, což značně urychluje celý proces trénování neuronových sítí. Tensor lze vytvořit z *numpy* pole (typicky obsahujícího trénovací data, např. z paměti načtený obrázek) jednoduchým způsobem:

```
image = np.array(Image.open("car.png"))
tensor_image = torch.from_numpy(image)
```

Další důležitou entitou je tzv. modul (*torch.nn.Module*), který reprezentuje základní třídu pro vytváření neuronových sítí. Každý modul by měl obsahovat konstruktor `__init__`, který inicializuje jednotlivé vrstvy sítě. Další důležitou metodou je metoda `__forward__`. V ní jsou předávana vstupní data vsrtvám definovaným v konstruktoru `__init__`; je tak zajištěn samotný výpočet. Následující fragment kódu ilustruje vytvoření jednoduché neuronové sítě se dvěma konvolučními vrstvami:

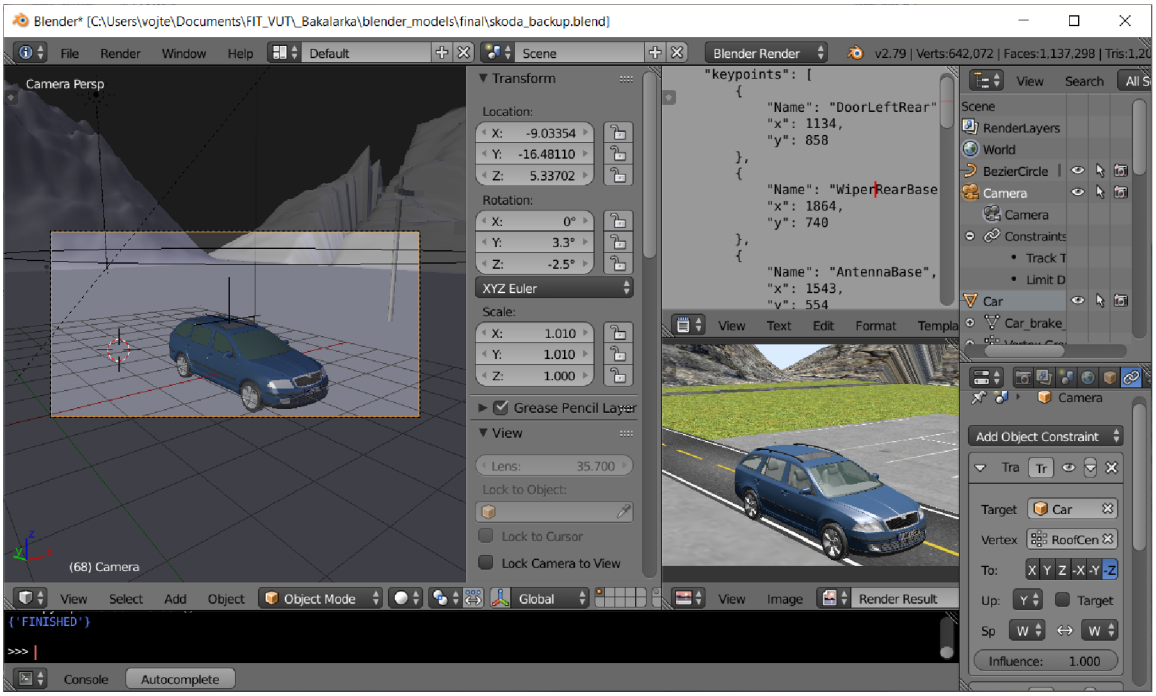
```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, 3)
        self.conv2 = nn.Conv2d(10, 10, 3)

    def forward(self, x):
        x = nn.ReLU(self.conv1(x))
        return nn.ReLU(self.conv2(x))
```

2.4 Blender

Blender je multiplatformní open-source softwarový nástroj sloužící k vytváření 3D grafiky a následné práci s ní, typicky zahrnující animace, renderování, simulace apod. Jak uvádí Kent [5], využití nachází kromě zábavního a videoherního průmyslu i ve vizualizacích z oblastí vědy a výzkumu. V této práci jsem ho využil k vytváření trénovacích datasetů.

Blender umožňuje kromě modelování prostorové scény (v případě této práce skládající se z 3D modelu automobilu a pozadí) i renderování 2D snímků. K tomuto účelu slouží objekt typu *Camera*, který je možno umístit do libovolné pozice ve scéně. Blender rovněž



Obrázek 2.7: Prostředí programu Blender 2.79. V levé části okna se nachází 3D pohled na objekt (model automobilu a scény na pozadí), v pravé spodní části vyrenderovaný snímek, v pravé horní části vygenerovaný soubor s anotacemi jednotlivých význačných bodů a nejvíce vpravo potom okna umožňující nastavování vlastností jednotlivých objektů scény. Ve spodní části okna je dále konzole pro skriptování v jazyce Python.

umožňuje definovat vztahy a omezení (*constraints*) mezi jednotlivými objekty, čehož jsem využil např. při změnách polohy kamery. Při přemístění kamery na jakékoliv souřadnice je tak kamera stále namířena na model automobilu (omezení typu *Track To*). Další vhodnou kombinací omezení lze poté docílit toho, že kamera se pohybuje po uživatelem definované trajektorii a nikdy se nevzdálí příliš daleko či nepřiblíží příliš blízko k určitému objektu (omezení *Limit Distance*). Anotace jednotlivých význačných bodů (vrcholů ve 3D modelu) lze definovat použitím struktury *Vertex Group* tak, že pro každý význačný bod je vytvořena jedna tato struktura obsahující odkaz pouze na tento jeden význačných bod (obecně však může *Vertex Group* obsahovat bodů více).

2.4.1 Blender Python API

Blender poskytuje přístup k všem funkcionalitám dostupným přes grafické uživatelské rozhraní i skriptům přes své Blender/Python API[3]. Vzhledem k potřebné velikosti mého trénovacího a validačního datasetu (desetitisíce obrázků s anotacemi) jsem ho využil právě pro generování trénovacích snímků s anotačními soubory. API umožňuje skriptům vykonávat veškeré úkony, které je možné provádět v grafickém rozhraní. Lze tak např. měnit polohu jednotlivých objektů, upravovat další jejich vlastnosti (jako např. geometrii, anotovat body na modelu), renderovat snímky atp. API je možné využít jak v interaktivním grafickém módu (viz obrázek 2.7), tak ze skriptů spouštěných pouze přes příkazovou řádku. Jeho konkrétní využití v rámci této práce blíže popisuje kapitola 4.1.

[3] docs.blender.org/api/2.79/

Kapitola 3

Návrh řešení

V této kapitole jsou popsány koncepty a postupy, kterými byly řešeny jednotlivé praktické části této práce. Konkrétní podobou jejich implementace se zabývá následující kapitola 4.

3.1 Architektura sítě Stacked Hourglass

Implementace sítě Stacked Hourglass, kterou jsem používal pro experimenty, používá aktivační funkci ReLU. V poolingových vrstvách je využíván max-pooling s velikostí kroku 2 a velikostí oblasti zpracovávané v jedné iteraci 2×2. Počet stacků i bloků je flexibilní; jedná se o klíčové parametry, s jejichž kombinacemi jsem experimentoval. Výchozí velikost příznakového vektoru je 128. Uvnitř sítě jsou použity konvoluční vrstvy s jádry o velikosti 1, 3 a 7. Je využita rovněž vrstva pro normalizaci dávky (batch normalisation).

3.2 Generátor trénovacích dat využívající API Blenderu

Základním předpokladem pro trénování neuronových sítí jsou trénovací data. Pro tuto práci mají podobu snímků automobilu pořízených z různých úhlů pohledu spolu s informací, kde přesně se na obrázku jednotlivé význačné body nacházejí (v podobě pixelových souřadnic; viz obrázek 3.1). Trénovací data tohoto typu nejsou volně dostupná; vytvořil jsem tedy nástroj, který je schopen je pomocí programu Blender generovat. Nástroj využívá Blender/Python API a pracuje s 3D modelem, který má označené význačné body (v Blenderu je tak každý význačný bod reprezentován jednou strukturou *Vertex Group*, obsahující vždy právě jeden vrchol z trojrozměrného modelu). Automaticky mění pro každý snímek polohu kamery, která je ovšem vždy zaměřena na klíčový 3D model (například automobilu) pomocí tzv. omezení (*constraints)*. Ke každému vyrenderovanému snímku je následně vygenerován soubor obsahující 2D souřadnice jednotlivých význačných bodů na výsledném snímku. Pokud je bod z daného zorného úhlu kamery neviditelný (viditelnost je ověřována metodou vrhání paprsku, viz kapitolu 3.2.1), není ve výsledném souboru obsažen.

Algoritmus renderování snímku a generování patřičného souboru s anotacemi má následující podobu:

1. Nastavení rozlišení výsledného snímku a polohy kamery v prostoru.
2. Aktualizování scény.
3. Renderování snímku a jeho uložení.

Obrázek 3.1: Při generování trénovacích dat jsou renderovány snímky automobilu z různých úhlů pohledu. Barevné tečky na obrázcích zde reprezentují polohu několika vybraných význačných bodů (jejich souřadnice jsou uloženy v souboru s anotacemi).

4. Výběr význačných bodů, které jsou přítomny na výsledném snímku a zároveň jsou „viditelné“.

5. Výpočet 2D souřadnic viditelných význačných bodů na základě jejich 3D souřadnic v modelu a aktuální polohy kamery .

6. Uložení seznamu viditelných význačných bodů s údaji o jejich poloze na vyrenderovaném snímku do souboru.

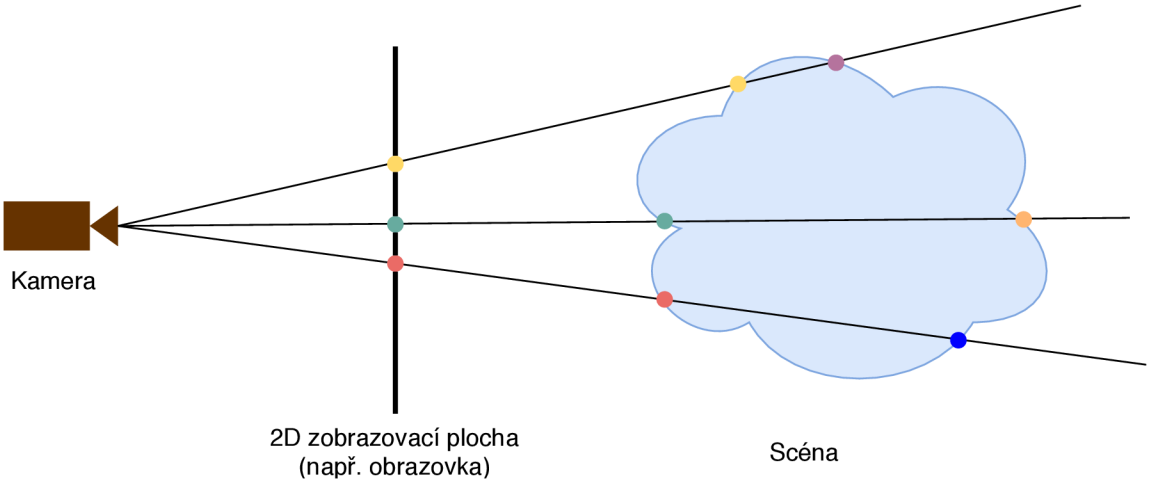
3.2.1 Vrhání paprsku pro zjištění viditelnosti vrcholů

Blender sám o sobě neobsahuje metodu jak určit, zdali jsou jednotlivé vrcholy ve struktuře *Vertex Group* z určité pozice kamery viditelné, či nikoliv. Tento údaj je však pro účely této práce klíčový: význačné body, které nejsou z pozice kamery viditelné, by samozřejmě neměly být sítí detekovány a tím pádem souřadnice těchto bodů nesmí být ani obsaženy v souboru s anotacemi. K vyřešení tohoto problému jsem využil metodu vrhání paprsku (*ray casting*). Její podstatou je vyslání paprsků z místa, kde se nachází kamera, různými směry do scény. Pro každý paprsek je poté zaznamenáváno, které body a v jakém pořadí zasáhl. V kontextu mé práce je podstatný vždy pouze první bod, který paprsek zasáhne. Ten je považován za viditelný a je dále využíván. Ostatní body zasažené paprskem v pozdějším pořadí nejsou použity a neobjevují se tak ani ve výsledném anotačním souboru. Podstatu metody vrhání paprsku vystihuje obrázek 3.2.

3.3 Augmentace trénovacích dat

Augmentace trénovacích dat znamená v kontextu strojového učení vytváření nových dat modifikacemi dat stávajících (Perez [10]). V případě obrázků jsou klasickými augmentačními postupy např. posunutí, rotace, zvětšení, zmenšení, změna jasu či přidání šumu. Tyto typy augmentací jsou znázorněny na obrázku 3.3. Jak uvádí Bhardwaj [2], k augmentaci dat je přistupováno ze dvou hlavních důvodů.

Prvním z nich je obtížné zajištění dostatečného (zpravidla značného) množství trénovacích dat. V mém případě byla např. doba potřebná k vygenerování jednoho trénovacího snímku s anotacemi v řádech desítek sekund (blíže popsáno v kapitole 4.1), přičemž pro kvalitní trénování jsou zapotřebí řádově desetitisíce snímků. Existují rovněž i aplikace neuronových sítí, pro které je získávání trénovacích dat ještě obtížnější, jako například sítě pro



Obrázek 3.2: Podstata metody vrhání paprsku. **Vlevo:** kamera, ze které jsou vysílány paprsky různými směry. **Uprostřed:** zobrazovací plocha, na které se zobrazí vždy první bod zasažený paprskem. **Vpravo:** prostorová scéna.

klasifikaci objektů. V takovém případě je totiž často nezbytné vytvářet trénovací datasety manuálně (např. označováním jednotlivých objektů a určování jejich typu).

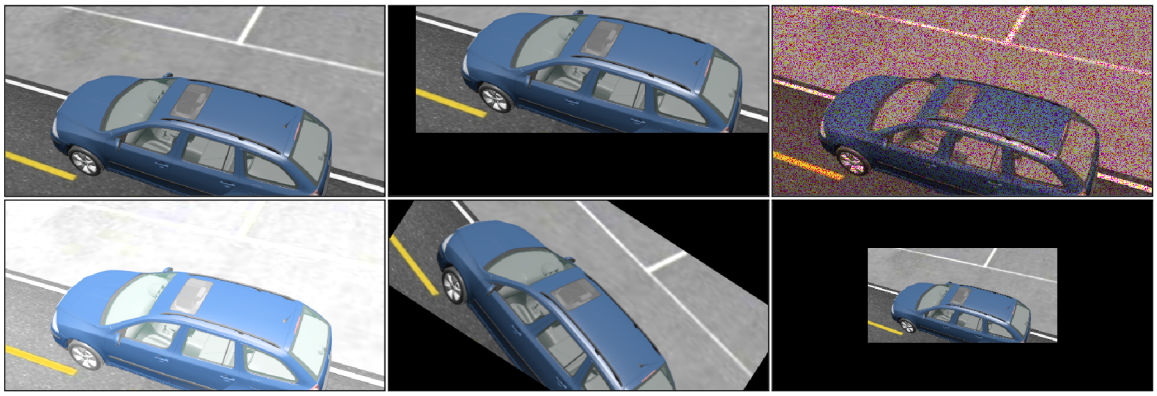
Druhým hlavním důvodem, proč je standardně přistupováno k augmentaci dat, je snaha o prevenci přeučení sítě (*overfitting*). Přeučení je nežádoucí jev, při kterém je síť naučena na příliš specifickém či úzkém datasetu, přičemž poskytuje relativně kvalitní výsledky pro data trénovací, pro data podobná ovšem selhává. Pokud bude síť naučena rozpoznávat automobily a trénovací dataset bude obsahovat pouze snímky modrých automobilů, je pravděpodobné, že síť nerozpozná automobil červený, neboť byla naučena na příliš úzkém datasetu a její schopnost generalizovat je tak do značné míry omezena. Tím, že síti poskytneme data augmentovaná, bude množina trénovacích dat různorodější a k přeučení nedojde tak snadno.

Při tvorbě vlastního datasetu jsem přistoupil k augmentaci posunutím, změnou měřítka, rotací, přidáním šumu typu *Salt and pepper* a náhodnými změnami jasu výchozích snímků. Mou hlavní motivací pro augmentaci dat při tvorbě této práce byla právě prevence přeučení, neboť vzhledem k povaze úlohy byl můj dataset tvořen snímky stejného automobilu pořízenými pouze z různých úhlů a vzdáleností. Augmentaci dat jsem realizoval s použitím knihovny OpenCV (vycházel jsem z postupů popsaných Bradskim [3]). Pozn.: Model pozadí scény pochází ze stránek `free3d.com`[1].

3.4 Vyhodnocování úspěšnosti detekce význačných bodů

Prvním krokem vyhodnocování úspěšnosti detekce význačných bodů na snímku byla vizuální kontrola. Vzhledem ke značnému množství detekovaných význačných bodů a velikosti validačního datasetu bylo ovšem nutné vytvořit metodu automatického vyhodnocování úspěšnosti detekce. Pro každý význačný bod generuje model sítě právě jednu výslednou pravděpodobnostní matici; souřadnice detekovaného bodu pak odpovídá souřadnici prvku s nejvyšší hodnotou. Je tedy zapotřebí určit prahovou hodnotu, sloužící pro rozhodnutí,

[1] `https://free3d.com/3d-model/dodge-challeger-and-python-script-for-driving-with-small-map-402706.html`



Obrázek 3.3: Typy augmentací, které jsem použil při vytváření trénovacího datasetu. **Vlevo nahoře:** původní vyrenderovaný obrázek, **nahoře uprostřed:** augmentace posunutím, **vpravo nahoře: augmentace přidáním šumu typu *Salt and pepper*** augmentace přidáním šumu typu *Salt and pepper*, **vlevo dole:** augmentace změnou jasu (složka *Value* v modelu HSV), **dole uprostřed:** augmentace rotací snímku, **vpravo dole:** augmentace změnou měřítka (*scale*). Všechny obrázky jsou v rozlišení 512×256 pixelů.

zdali síť daný význačný bod detekovala, či nikoliv. Tuto prahovou hodnotu jsem určil na základě experimentů (více v kapitole 4.4). Nejdůležitějším kritériem při vyhodnocování je vzdálenost detekovaného význačného bodu od jeho skutečné polohy určené souborem s anotacemi. Pro výpočet vzdálenosti d mezi body $A_1[x_1, y_1]$ a $A_2[x_2, y_2]$ v rovině je použit následující vztah:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \tag{3.1}$$

přičemž platí, že čím je vzdálenost d menší, tím je daný význačný bod detekován přesněji.

Dalším kritériem je počet falešných detekcí (*false positive rate*) význačných bodů, tedy situací, kdy natrénovaný model neuronové sítě chybně detekuje význačný bod na snímku, na kterém ovšem tento bod ve skutečnosti není vůbec viditelný. Model sítě by se falešných detekcí měl dopouštět skutečně minimálně, neboť obecně platí, že falešná detekce je závažnější než situace, kdy síť nedetekuje bod, který detekován být měl (na obrázku byl viditelný).

Dalším parametrem braným v potaz při vyhodnocování úspěšnosti je konkrétní hodnota ve výsledné pravděpodobnostní matici. Platí, že čím vyšší je tato hodnota, tím si je model sítě „jistější“ pozicí daného bodu; hodnota by měla být v rozsahu $\langle 0, 1 \rangle$.

Kapitola 4

Implementace a experimenty s trénováním modelů sítě Stacked Hourglass

Tato kapitola popisuje v první části implementaci jednotlivých částí této práce. Druhá část kapitoly je věnována popisu a vyhodnocení jednotlivých experimentů s trénováním modelů, přičemž pořadí dílčích kapitol odráží postup tvorby práce.

4.1 Generování trénovacích dat pomocí API Blenderu

Nástroj pro generování trénovacích dat jsem implementoval v jazyce Python 3.6 (API Blenderu je podporuje právě tento jazyk). Soubor obsahující 2D souřadnice viditelných význačných bodů je ve formátu JSON a má následující podobu:

```
{"keypoints":
    [{"Name": "MirrorLeftFront",
      "x": 108,
      "y": 68},

     {"Name": "RandomKP_775",
      "x": 107,
      "y": 69}
]}
```

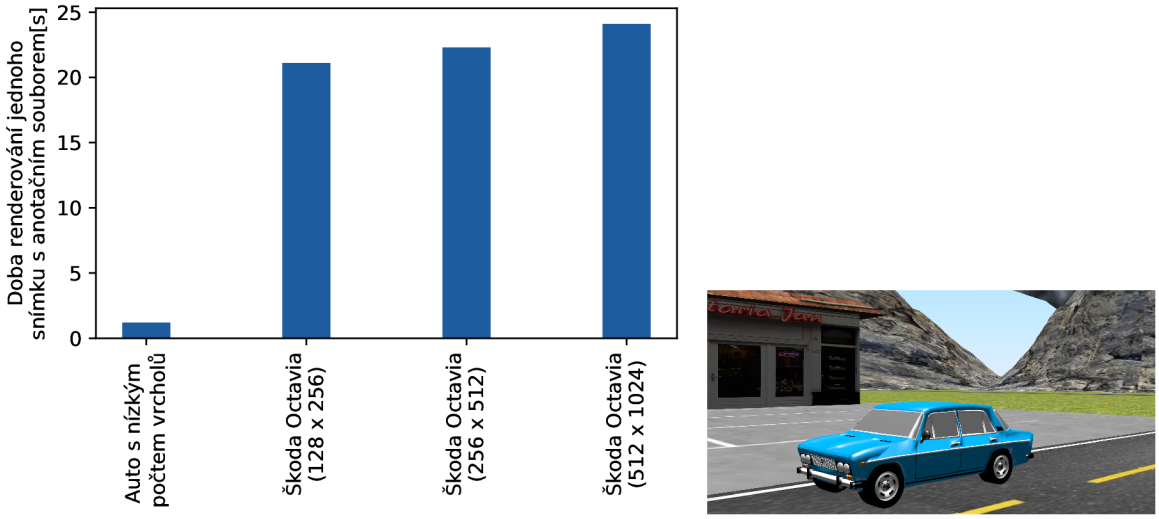
Následuje fragment kódu skriptu pro generování trénovacích dat. Za povšimnutí stojí zejména poslední řádek: po nastavení nové polohy kamery je nutné provést volání funkce `bpy.context.scene.update()` zajišťující obnovení scény s novou polohou kamery, v opačném případě je pro další výpočty použita původní poloha kamery. V grafickém uživatelském prostředí programu Blender stačí ovšem nastavit pouze polohu kamery a volání je provedeno na pozadí; rovněž v oficiální dokumentaci API Blenderu i dalších zdrojích není nutnost volání funkce zmíněna. Můj skript tak dlouho vykazoval chyby, než jsem odhalil tuto jejich příčinu.

```
cam = bpy.data.objects['Camera']
scene = bpy.context.scene
```

```
render = scene.render
render.resolution_x = res_x
render.resolution_y = res_y
# set camera position in space
cam.location.x = random.uniform(-5,14)
cam.location.y = random.uniform(-8,8)
cam.location.z = random.uniform(1,4)

bpy.context.scene.update()  # extremely important!
```

Doba renderování jednotlivých snímků je porovnána na obrázku 4.1, z něhož vyplývá, že klíčovým faktorem dobu ovliňujícím je složitost 3D modelu s význačnými body (nikoliv tedy počet vrcholů celé scény včetně pozadí, jak jsem na počátku předpokládal).



Obrázek 4.1: **Vlevo:** Srovnání doby renderování jednoho snímku pro různé modely a rozlišení. Pozadí scény obsahovalo 642 072 vrcholů, model auta s nízkým počtem vrcholů 10 179 a model Škody Octavia II combi 576 624. Z grafu je patrné, že nejpodstatnějším faktorem ovlivňujícím dobu renderování snímku je složitost použitého 3D modelu, nikoliv rozlišení. **Vpravo:** model automobilu s nízkým počtem vrcholů, který jsem používal při ladění generátoru dat.

Nejprve jsem tedy používal jednoduchý model automobilu[1], který díky nízkému počtu vrcholů umožňoval rychlé generování dat. Pro použitelnost v reálných aplikacích bylo ovšem nutné zajistit přesný model skutečného automobilu, který se ideálně vyskytuje na českých silnicích často. Pro tyto účely mi vedoucí práce poskytl detailní model automobilu Škoda Octavia II Combi.

4.1.1 Skript pro automatické generování náhodných význačných na modelu v Blenderu

Význačnými body potenciálně použitelnými ke kalibraci jsou kromě bodů „přirozených“ (např. logo výrobce, střed zpětného zrcátka či klika u dveří) i všechny další body na po-

[1] https://free3d.com/3d-model/vaz-2106-low-poly-963123.html

vrchu automobilu. Tento fakt jsem se rozhodl zohlednit tak, že kromě zmíněných bodů „přirozených“ vyberu z povrchu auta i další náhodné význačné body. Vzhledem k velkému množství a časové náročnosti manuálního výběru bodů jsem vytvořil skript, který dokáže pomocí API Blenderu náhodné body vybrat. Pro každý takto náhodně vybraný význačný bod pak vytvoří vlastní strukturu *Vertex Group*, která dále slouží pro přepočet prostorových souřadnic vrcholu do rovinných pixelových souřadnic na výsledném snímku.

Jádro skriptu má tuto podobu:

```
selected_verts = [v.index for v in obj.data.vertices if v.select]
for i in range (0, num_new_keypoints):
    # create new vertex group
    bpy.ops.object.vertex_group_add()
    v_groups = bpy.context.active_object.vertex_groups
    # set vertex group name
    v_groups[i].name = "RandomKP_" + str(i)
    idx = random.randint(0,len(selected_verts)-1)
    # add random vertex to vertex group
    v_groups[i].add(index=[selected_verts[idx]], weight=1, type='ADD')
    # vertex can be in max. 1 vertex group (no duplicate VG allowed)
    selected_verts.remove(selected_verts[idx])
```

4.2 Implementace programu pro trénování neuronové sítě

Pro implementaci neuronové sítě typu Stacked Hourglass pro rozpoznávání význačných bodů na obrázcích automobilu jsem na základě doporučení svého vedoucího zvolil jazyk Python 3.6 a knihovnu PyTorch (verze 1.1.0).

Základem implementace, ze které jsem vycházel, byla implementace sítě pro rozpoznávání kloubů za účelem detekce celkové pozice lidského těla, kterou publikoval Wei Yang[2], Ph.D., pod licencí GNU GPL 3.0. Yang vycházel z díle Newella [7]; jeho implementace sloužila k rozpoznávání pozice lidského těla pomocí rozpoznávání poloh jednotlivých kloubů.

Největší výzvou byla příprava dat do správného formátu, očekávaného sítí. Yangova implementace totiž pracovala se standarními datasety (COCO[3], MPII Human Pose Dataset[4]; můj dataset byl ovšem odlišný. Po sérii experimentů jsem zvolil rozlišení vstupního snímku 256×256, respektive 256×128 čemuž odpovídá výstupní pravděpodobnostní matice o rozměrech 64×64, resp. 64×32.

4.2.1 Načítání dat

Nedílnou součástí implementace trénování modelu neuronové sítě je způsob, jak mu poskytnout trénovací data. Vzhledem k tomu, že jsem vytvořil vlastní dataset, bylo nutné implementovat i tzv. „loader“. Loader je entita (v mé implementaci třída) poskytující síti jednotlivé dávky trénovacích dat (batch). Dávka o velikosti N potom obsahuje vždy X trénovacích obrázků; ke každému z nich dále sadu N pravděpodobnostních matic (znázorněno na obrázku 4.2). Sada obsahuje vždy právě tolik matic, kolik význačných bodů bude naučená síť detekovat (pro každý význačný bod právě jednu matici). Matice samotná je poté

[2] github.com/bearpaw/pytorch-pose
[3] cocodataset.org
[4] http://human-pose.mpi-inf.mpg.de/

Obrázek 4.2: Výstup loaderu je vždy jedna dávka, v tomto případě o velikosti 2. Ke každému obrázku z dávky poté přísluší tolik pravděpodobnostních matic, kolik význačných bodů bude síť detekovat, zde 4.

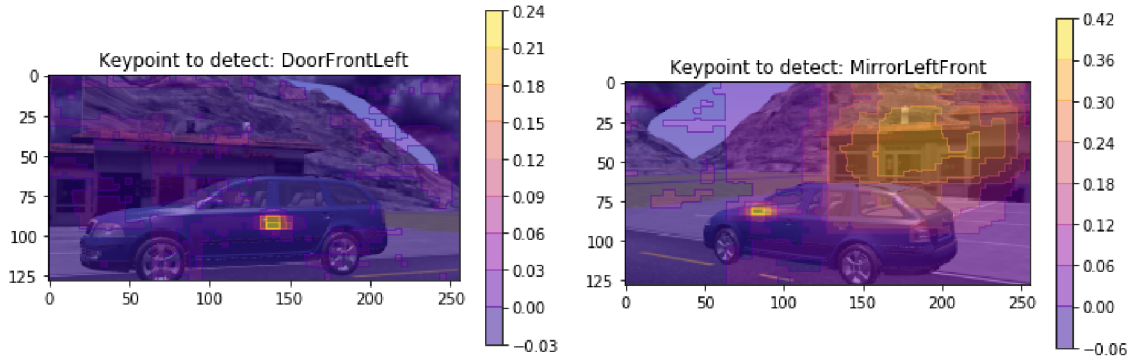
dvourozměrné pole, které obsahuje hodnoty v rozsahu $\langle 0, 1 \rangle$. Tyto hodnoty reprezentují pravděpodobnost, že se na dané souřadnici matice nachází konkrétní význačný bod.

Nejvyšší hodnoty se nacházejí v těsné blízkosti pixelových souřadnic význačného bodu. Tyto souřadnice jsou načteny z JSON souboru s anotacemi. V případě, že význačný bod není z daného úhlu pohledu viditelný (JSON soubor pro něj neobsahuje žádný záznam), má výsledná matice na všech pozicích hodnotu 0. V případě, že pro význačný bod anotace existuje, je nutné odrazit informaci o jeho poloze na obrázku ve výsledné matici. Triviálním řešením je na tuto danou souřadnici význačného bodu do výsledné matice uložit hodnotu 1 (maximální pravděpodobnost) a na ostatní pozice nuly. Toto řešení se ovšem již po prvotních experimentech s jednoduchou sítí detekující červené body na zeleném pozadí (blíže popsáno v kapitole 4.4) ukázalo být nevyhovujícím, neboť přechod byl příliš ostrý. Síť tak nemohla být tak dobře „odměněna“ za situaci, kdy označila za nalezený význačný bod bod v jeho těsné blízkosti (byla za něj oceněna stejně, jako kdyby za nalezený význačný bod označila bod velice vzdálený skutečným souřadnicím). Optimálním řešením se tedy ukázalo být generování dvourozměrné Gaussovy funkce se středem právě v souřadnicích daného význačného bodu a s hodnotou σ 0,7. K této konkrétní hodnotě jsem dospěl po sérii experimentů, kdy nižší hodnoty tvořily příliš ostrý přechod a naopak vyšší hodnoty σ postihovaly příliš mnoho z okolí význačného bodu, což vedlo k nepřijatelné nepřesnosti.

Při mých experimentech jsem nejvíce pracoval s dávkou o velikosti 32 snímků. Tato hodnota se ukázala být optimální z hlediska využití paměti RAM v kombinaci s rychlostí trénování. Můj prvotní předpoklad byl, že čím větší dávka bude, tím rychleji se bude síť učit. Ukázalo se ovšem, že použití dávky o velikosti 64, či 128 snímků vedlo k prodloužení celkové doby trénování (v řádu vyšších jednotek procent).

4.3 Automatické vyhodnocování přesnosti rozpoznávaných význačných bodů

Po provedení řádově desítek experimentů, při kterých jsem vizuálně kontroloval úspěšnost detekce jednotlivých význačných bodů (včetně kontroly numerických hodnot maxima na výsledné pravděpodobnostní matici – viz obrázek 4.3), jsem došel k závěru, že prahovou hodnotu, od které je možno považovat bod za rozpoznaný, je 0,1. Při automatickém vyhodnocování jsou tedy jako rozpoznané význačné body brány ty, pro které je maximální hodnota ve výsledné matici větší než tato prahová hodnota.

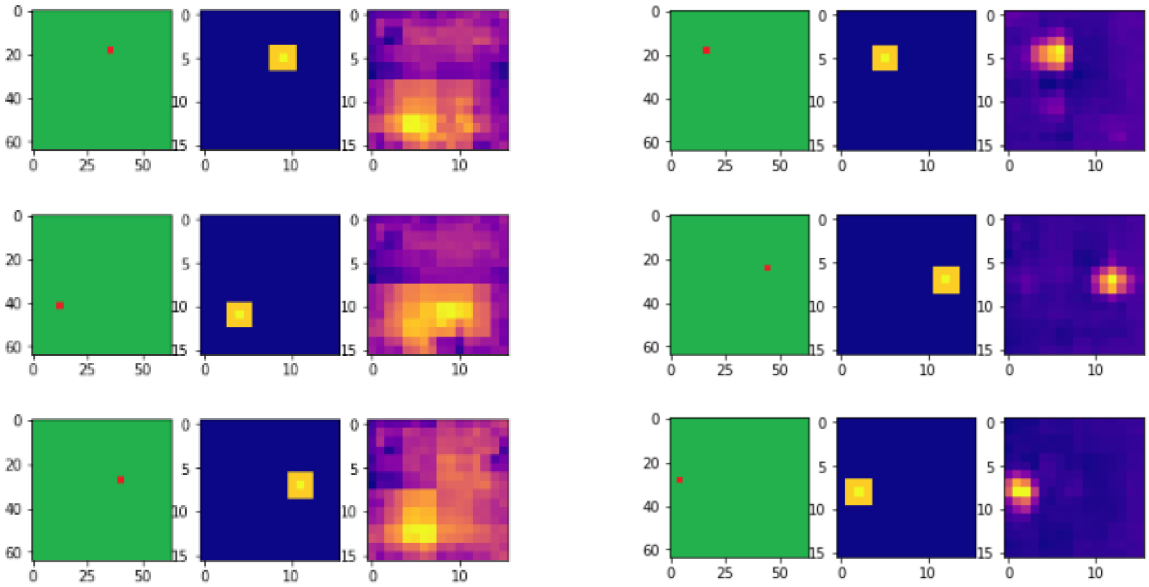


Obrázek 4.3: Vizuální kontrola přesnosti detekce význačných bodů. **Vlevo** na každém z obrázků je vstupní snímek automobilu překrytý vizualizací výsledné pravděpodobnostní matice. **Vpravo** se nachází škála, z níž lze určit numerickou hodnotu pro daný detekovaný význačný bod. Na obrázku vpravo je patrné, že tento konkrétní model sítě neposkytuje ještě ideální výsledky – vysoké hodnoty jsou kromě okolí levého předního zrcátka (což je správné) ještě v rozlehlé oblasti v pravé horní čtvrtině obrázku, což je typickým znakem toho, že model musí být dále trénován.

Pro vyhodnocování jsem používal validační dataset obsahující 9 600 snímků, přičemž úspěšnost detekce určitého význačného bodu je dána vektorovou vzdáleností pozice detekované sítí na snímku a pozice skutečné (dané anotačním souborem). V potaz bylo nutné brát i počet detekcí daného bodu při validaci. Některé význačné body totiž vykazovaly velkou míru přesnosti detekce (nízkou odchylku), ovšem při bližším zkoumání vyšlo najevo, že tyto body jsou detekovány zřídka (v jednotkách až desetinách procent). Tyto body tedy do množiny bodů nejlepších, které byly dále analyzovány, zařazeny nebyly. Za dostatečný počet detekcí jsem zvolil 33 % (aby měl tedy význačný bod šanci stát se jedním z nejlepších bodů, musel být detekován alespoň na jedné třetině snímků z validačního datasetu). Nejpřesněji detekovatelné význačné body poté dosahují průměrné odchylky menší než 3 pixely (podrobněji v kapitole 4.4).

4.4 Výsledky trénování s různě nastavenými parametry sítě

Tato kapitola obsahuje popis a vyhodnocení vybraných experimentů s neuronovou sítí. Zachycuje tak i chronologický vývoj práce, kdy jsem nejprve celou architekturu zprovoznil na relativně jednoduchém příkladu detekce bodu na pozadí kontrastní barvy. Poté jsem postupně přidával na složitosti za neustálého ladění parametrů experimentů, z nichž nejdůležitějšími se ukázalo být *learning rate*, rozlišení vstupních obrázků a počet detekovaných



Obrázek 4.4: Detekce červených bodů na zeleném pozadí. Síť Stacked Hourglass (1 stack, 2 bloky) natrénována na 10 000 obrázcích o rozměru 64 × 64, výsledná matice 16 × 16. První sloupec obsahuje vždy vstupní obrázek, pro který bude prováděna detekce, druhý ideální výstup sítě (společně tvoří trénovací dataset). Třetí sloupec obsahuje výsledek detekce (pravděpodobnostní matici s odhadem pozice bodu).

význačných bodů spolu s počtem stacků a bloků sítě. Ukázalo se, že příliš rychlý postup (např. čtyřnásobný počet detekovaných keypointů při zachování všech ostatních parametrů) nevedl k cíli, proto bylo nutné provést řádově desítky experimentů (za experiment považuji jedno natrénování modelu sítě s určitými parametry), z nichž nejvýznamnější jsou blíže popsány v následujících podkapitolách.

4.4.1 Síť detekující body na kontrastním pozadí – Proof of Concept

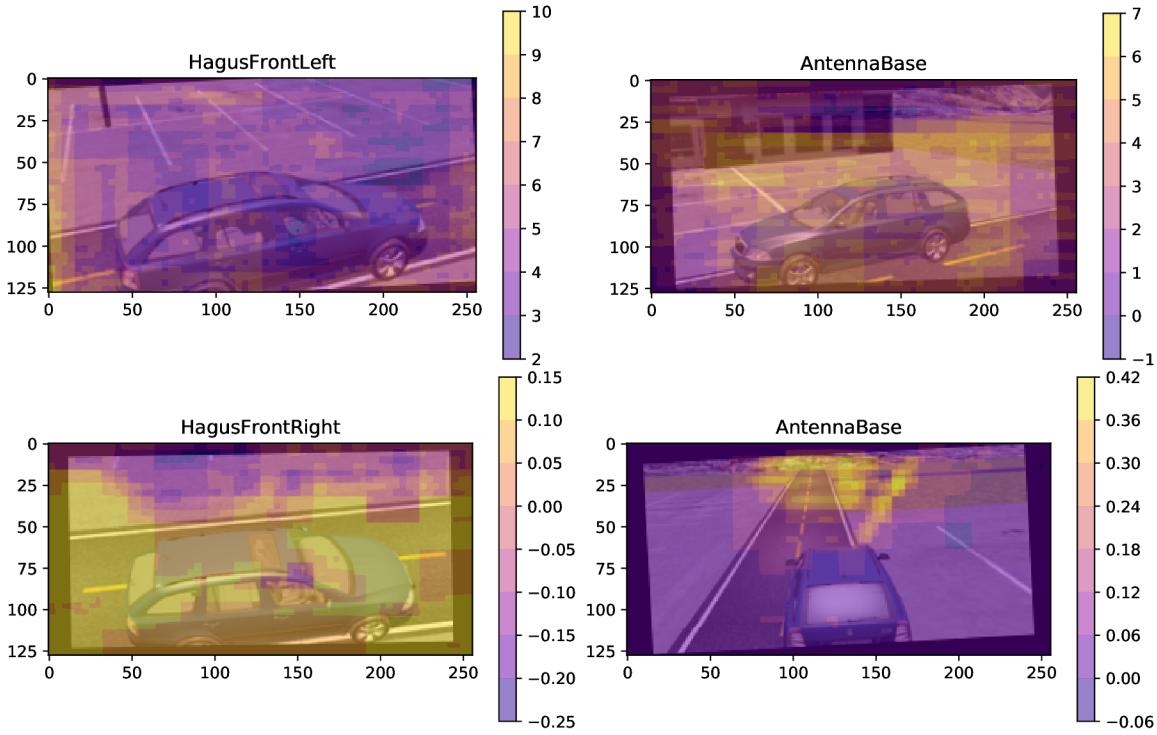
Pro ověření funkčnosti celé architektury jsem nejprve experimentoval se sítí, jejímž úkolem bylo detekovat na zeleném obrázku pozici červené tečky. Tato relativně triviální úloha měla především za cíl vyladit implementaci sítě do použitelného stavu a dokázat, že jednotlivé komponenty (trénovací data, DataLoader a samotná síť) jsou vzájemně kompatibilní natolik, aby byly schopné poskytnout výsledek této základní úlohy z oblasti detekce.

Po sérii ladících experimentů se mi podařilo natrénovat síť schopnou relativně přesně detekovat pozici bodů na kontrastním pozadí, jak je patrné z obrázku 4.4.

4.4.2 Ladící experimenty

Prvním krokem při provádění experimentů s detekcí význačných bodů na obrázcích automobilu byla změna formátu datasetu, kdy jsem z původního čtvercového formátu (nejběžnější formátu obrázku při trénování konvolučních sítí) trénovacích snímků přešel k formátu obdélníkovému. Motivací tohoto kroku byla snaha více přizpůsobit formát tvaru automobilu tak, aby bylo na snímku obsaženo více plochy automobilu a méně pozadí.

Následovaly experimenty, při nichž jsem učil síť detekovat postupně 10, 24, 128, 256 a 512 význačných bodů. Hlavními parametry, které od sebe jednotlivé experimenty odli-



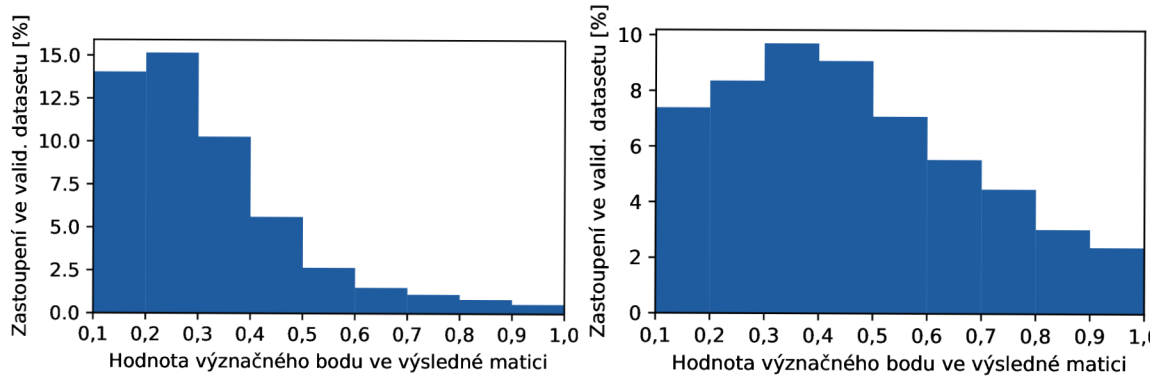
Obrázek 4.5: **Vlevo a vpravo nahoře:** výstup modelu sítě, který nebyl trénován. Jeho vnitřní parametry jsou tak náhodná čísla, což lze vypozorovat mj. na hodnotách ve výsledné pravděpodobnostní matici (ty se pohybují ve velmi širokém rozsahu, značně vzdáleném od kýženého intervalu $\langle 0, 1 \rangle$). **Vlevo a vpravo dole:** výstupy jednoho z mnoha ladících experimentů, který nevedl k použitelnému výsledku. Parametry modelu: 1 stack, 1 blok, 512 detekovaných význačných bodů, learning rate 0,1.

šovaly, byl počet stacků a bloků. Implementace sítě (vytvořená pro odhad postoje osob na snímcích) počítá s výchozím počtem 2 stacky a 4 bloky. Tato konfigurace se ukázala pro moji práci jako nevyhovující; s rostoucím počtem detekovaných bodů poskytovala stále horší výsledky. Počet stacků a bloků jsem tedy postupně snižoval a kombinoval s různými výchozími hodnotami learning rate. Na obrázku 4.5 jsou ilustrovány případy, kdy je model buď zcela nenatrénovaný, či natrénovaný, ale špatně. Zajímavým pozorování plynoucím z tohoto experimentu byly hodnoty objevující se ve výsledné matici u špatně natrénovaného modelu. V porovnání s hodnotami u správně natrénovaných modelů (viz následující kapitolu) jsou tyto hodnoty vysoké, což indikuje, že si je tento model sítě relativně „jistý“ svými odhady, které jsou nicméně naprosto mylné.

Nejlepší konfigurací se ukázala být kombinace 1 stacku a 2 bloků spolu s velikostí příznakového vektoru 128. Na obrázku 4.6 jsou zobrazeny výsledné histogramy hodnot ve výsledné matici správně natrénovaných sítí pro detekci 128 a 512 význačných bodů.

4.4.3 Modely pro detekci 1 021 význačných bodů

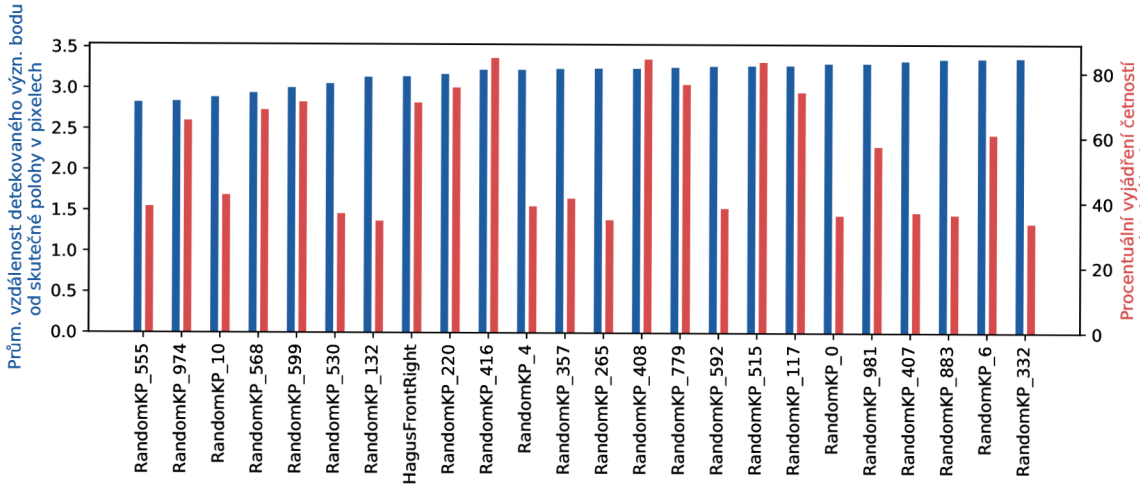
Cílem experimentů popsaných v této kapitole bylo natrénovat model sítě pro detekování 1 021 význačných bodů. Tento počet jsem po dohodě s vedoucím určil jako dostačující a dále ho nenavyšoval, protože detekce více bodů by již narážela na limity rozlišení trénovacích dat (256×128 pixelů). Po předchozích experimentech se ukázala být nejefektivnější konfigurace



Obrázek 4.6: Na obou histogramech je zachycena míra odezvy („jistoty“) sítě při detekci 24 nejlepších význačných bodů. **Vlevo:** síť natrénovaná pro detekci 128 význ. bodů, **vpravo** pro 512 význ. bodů. Obě sítě byly v konfiguraci 1 stack, 2 bloky, s learning rate $2,5e-4$, celkem 210 000 trénovacích obrázků.

sítě 1 stack a 2 bloky, velikost příznakového vektoru potom 128. Trénovací dataset tvořilo 70 000 snímků (včetně snímků augmentovaných), přičemž obě sítě byly natrénovány celkem na 210 000 obrázcích (každý snímek z unikátního datasetu tedy měla síť k dispozici během celého trénování třikrát). Trénování sítě v této konfiguraci trvalo zpravidla 31 hodin (na stroji s CPU Intel Core i5 7200U a 16 GB RAM).

Z výsledné tabulky 4.1 vyplývá, že trénování s learning rate $1{,}25e-4$ je ve všech sledovaných ohledech nejúspěšnější. Z grafu na obrázku 4.7 je patrné, že mezi 24 nejlepších význačných bodů se řadí 1 „nenáhodný“, mnou vybraný (*HagusFrontRight*). Lepší představu o výkonu sítě poskytují histogramy na obrázku 4.10, na nichž lze pozorovat rozložení odezvy sítě na vybrané význačné body. Vzhledem k poměru sémantických a náhodných význačných bodů (21 ku 999) se tento výsledek nezdá být překvapivý. Očekával jsem ovšem, že sémantické význačné body budou mezi nejlepšími obsaženy v hojnějším počtu, právě



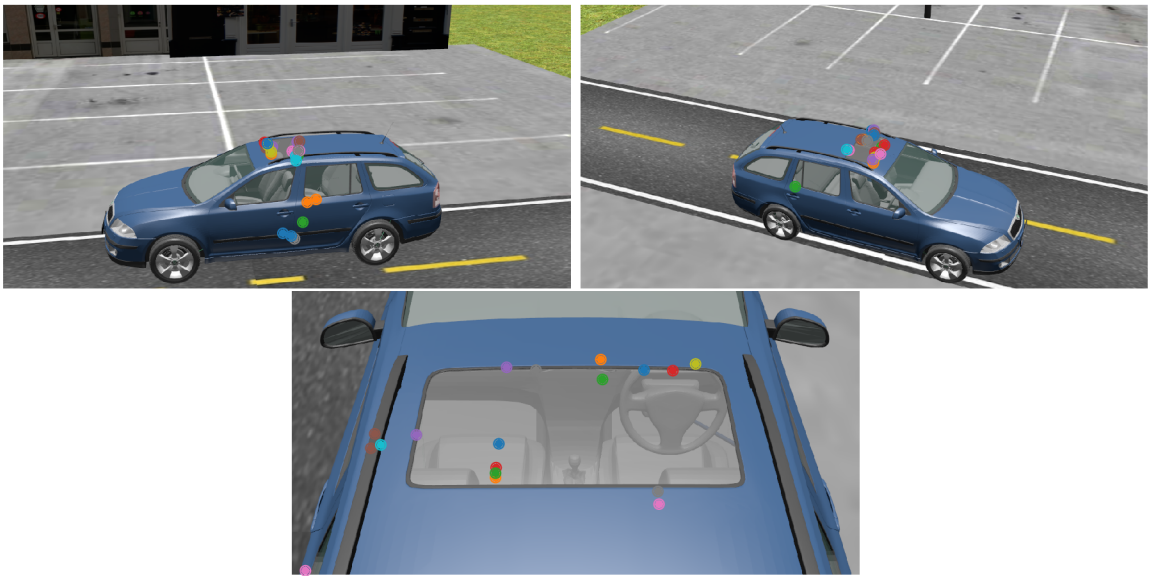
Obrázek 4.7: Porovnání četnosti detekce a průměrné odchylky nejlepších 24 význačných bodů. Hodnoty odchylek jsou velice podobné, četnosti detekce se však mezi jednotlivými význačnými body výrazně liší. Parametry tohoto modelu sítě jsou 1 stack, 2 bloky, learning rate 5e-4.

z důvodů své „přirozenosti“ (lidské oko snadněji rozpozná zrcátko než bod uprostřed střechy). Že se mezi nejlepšími sémantickými body objevuje právě *HagusFrontRight* připisuji faktu, že je díky své poloze velmi často na snímku viditelný a detekovaný – na validačním datasetu je detekován na 72 % snímků.

Histogramy zobrazené na obrázku 4.10, které vyjadřují rozložení odezvy daného modelu neuronové sítě na jednotlivé detekované význačné body (čím vyšší hodnota, tím si je síť detekcí „jistější“), jsou si vzájemně do značné míry podobné. Nejvíce hodnot se na všech nachází v intervalu $\langle 0,1; 0,3\rangle$. Všechny histogramy mají relativně plynulé přechody, z čehož se dá usuzovat, že síť poskytuje stabilní výsledky. Lze předpokládat, že při dalším rozšiřování trénovacího datasetu či zvýšení počtu trénovacích epoch by se střed histogramu posouval dále doprava.

K falešným detekcím prakticky nedocházelo. Z toho usuzuji, že hodnotu prahu, od kterého je význačný bod považován za detekovaný, činící $(0,1)$, jsem zvolil správně; zpětně viděno možná až příliš striktně. Rozhodl jsem se ji nicméně neměnit, neboť cílem práce bylo vybrat několik (desítek) nejlepších význačných bodů, a správně natrénované sítě jsou schopny bodů přesahujících tuto hodnotu prahu detekovat řádově stovky (typicky 200 až 300 – viz výsledky v tabulce 4.1), tj. množství více než dostačující. Automatická validace výsledků pro tuto síť trvala 56 minut (validační dataset o velikosti 9 600 snímků, hardwarová specifikace zmíněna na začátku této kapitoly).

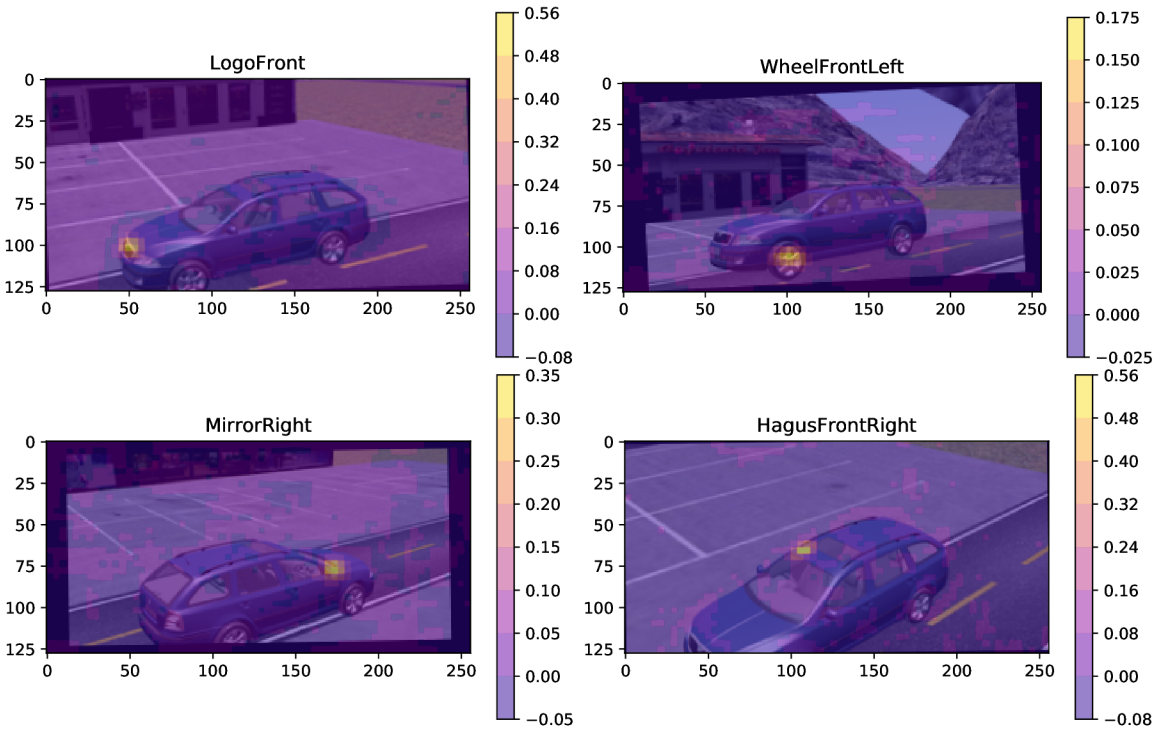
Celkově hodnotím průměrnou odchylku 24 nejlepších detekovaných bodů pro oba dva modely jako uspokojivou. Na obrázku 4.8 je zachyceno oněch 24 nejlepších význačných bodů detekovaných modelem poskytujícím nejlepší výsledky (tabulka 4.1). Pozoruhodné je jejich rozložení na modelu automobilu: přestože byly význačné body po jeho povrchu rozmístěny rovnoměrně, nachází se většina nejlepších bodů na střeše, konkrétně kolem střešního okna. Důvodem tohoto jevu může být fakt, že tyto body jsou viditelné z většiny úhlů pohledu (na rozdíl od bodů nacházejících se kupř. na levém zrcátku či pravém kole). Obecně je ovšem možné tvrdit, že dobře detekovatelné body se nacházejí v blízkosti hran či barevných změn, což koresponduje s prvotním předpokladem uvedeným v kapitole 1.



Obrázek 4.8: Vizualizace pozic 24 nejlepších bodů, detekovaných modelem sítě poskytujícím nejlepší výsledky (learning rate $1,25e-4$, 1 stack, 2 bloky).

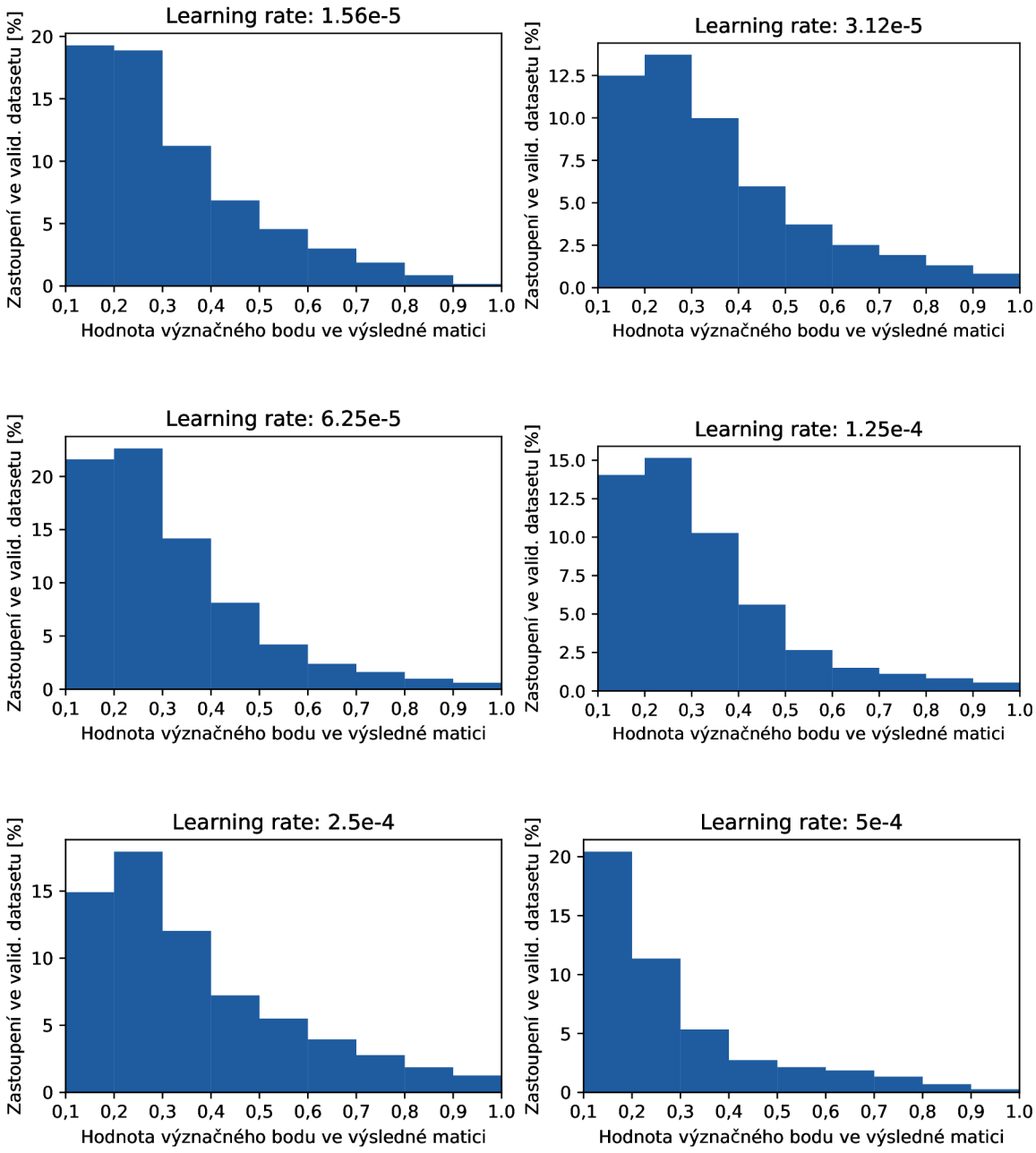
Tabulka 4.1: Výsledky trénování modelů sítě detekující 1 021 význačných bodů s různými hodnotami parametru *learning rate*. Za nejúspěšnější hodnotu learning rate lze prohlásit $1,25e-4$.

Body detekované na 33 % a více validačních snímcích					
				24 nejlepších	
Learning rate	poměr detekovaných význ. bodů	průměrná odchylka od skutečné polohy [pixel]	průměrná hodnota odezvy sítě na bod (z int. $\langle 0, 1 \rangle$)	průměrná odchylka od skutečné polohy [pixel]	průměrná hodnota odezvy sítě na bod (z int. $\langle 0, 1 \rangle$)
5,0e-4	21,35 %	8,58	0,23	4,16	0,24
2,5e-4	30,46 %	5,35	0,26	4,03	0,33
1,25e-4	37,12 %	5,15	0,26	3,73	0,33
6,25e-5	26,25 %	5,65	0,22	4,27	0,30
3,12e-5	23,12 %	13,40	0,25	4,09	0,33
1,56e-5	25,07 %	5,66	0,24	4,14	0,29



Obrázek 4.9: Několik detekovaných význačných bodů. Každý výchozí obrázek automobilu je překryt vizualizací pravděpodobnostní matice pro jednotlivé význačné body, škála vpravo poté udává míru pravděpodobnosti, že pixel je detekovaným význačným bodem.

Pravděpodobnostní matice vybraných detekovaných bodů zobrazuje obrázek 4.9; obrázek 4.11 obsahuje 6 bodů, které byly natrénovanými modely z tabulky 4.1 detekovány celkově nejlépe.



Obrázek 4.10: Porovnání výsledků natrénovaných modelů sítí s odlišnými hodnotami *learning rate*. Histogram je sestaven z výsledků detekce 24 nejpřesněji detekovaných význačných bodů. Pozn.: histogramy neobsahují interval $\langle 0,0;0,1)$, nebo význačný bod je považován za detekovaný až při hodnotách větších než 0,1.

Obrázek 4.11: Vizualizace bodů, které v každém z experimentů popsaných v tabulce 4.1 byly ve skupině 24 nejlepších. Vlevo vždy vstupní obrázek s černě vyznačenou skutečnou polohou bodu a červeně s polohou detekovanou modelem sítě. Vpravo poté vstupní obrázek překrytý vrstvou obsahující pravděpodobnostní matici.

Kapitola 5

Závěr

Stěžejním výsledkem práce jsou natrénované modely konvolučních neuronových sítí typu Stacked Hourglass detekující význačné body na snímcích automobilu. Tyto modely byly trénovány pro detekci až 1 021 význačných bodů. Body, které dané modely detekují s největší přesností, dosahují průměrné odchylky menší než 3,0 pixelu (na snímcích v rozlišení 256×128 pixelů).

Podařilo se rovněž implementovat generátor trénovacích dat vhodných pro trénování konvolučních neuronových sítí detekujících význačné body na obrázcích. Generátor využívá API programu Blender a 3D modelů s Blenderem kompatibilních. Dále jsem vytvořil trénovací dataset čítající 70 000 trénovacích snímků automobilu Škoda Octavia II combi s anotacemi 1 021 význačných bodů a validační dataset obsahující 9 600 snímků.

Výstupem práce je dále snadno parametrizovatelný trénovací program implementovaný v jazyce Python, který obsahuje všechny komponenty potřebné k experimentování s trénováním modelů sítí typu Stacked Hourglass (načítání dat, vlastní trénování a následná prezentace a automatické vyhodnocování výsledků).

Práci je možno rozšířit vytvářením dalších trénovacích datasetů se snímky stejného modelu automobilu pořízené na různých pozadích, případně vytvořením datasetů založených na jiných 3D modelech automobilů, než modelu mnou použitém.

Celkové výsledky této práce budou primárně sloužit výzkumné skupině Graph@FIT, především v oblasti automatické kalibrace dopravních dohledových kamer. Dále je možné výsledky využít k obdobným úlohám z oblasti strojového učení, jejichž podstatou je detekce význačných bodů na obrázcích.

Literatura

[1] Bartl, V.; Špaňhel, J.; Dobeš, P.; aj.: Automatic Camera Calibration by Landmarks on Rigid Objects. *[under review]Computer Vision and Image Understanding*, 2018.

[2] Bhardwaj, A.; Di, W.; Wei, J.: *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*. Packt Publishing, 2018.

[3] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[4] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *CoRR*, ročník abs/1512.03385, 2015.

[5] Kent, B.: *3D Scientific Visualization with Blender*. IOP Concise Physics, Morgan & Claypool Publishers, 2014, ISBN 9781627056120.

[6] Khan, S.; Rahmani, H.; Shah, S.; aj.: *A Guide to Convolutional Neural Networks for Computer Vision*. Synthesis Lectures on Computer Vision, Morgan & Claypool Publishers, 2018.

[7] Newell, A.; Yang, K.; Deng, J.: Stacked Hourglass Networks for Human Pose Estimation. *CoRR*, ročník abs/1603.06937, 2016.

[8] Olafenwa, J.: Understanding Residual Networks. *Towards Data Science*, Duben 2018.

[9] Patel, M.: When two trends fuse: PyTorch and recommender systems. Prosinec 2017, [Online; navštíveno 1.5.2019].
URL www.oreilly.com/ideas/when-two-trends-fuse-pytorch-and-recommender-systems

[10] Perez, L.; Wang, J.: The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *CoRR*, ročník abs/1712.04621, 2017.

[11] Subramanian, V.: *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*. Packt Publishing, 2018.

[12] Szegedy, S. I. C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, ročník abs/1502.03167, 2015.

[13] Widrow, B.; Lehr, M. A.: 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, ročník 78, č. 9, 1990: s. 1415–1442.

[14] Zhang, Z.: A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 22, Prosinec 2000: s. 1330–1334.

Příloha A

Obsah přiložené SD karty

- **datasets** – adresář obsahující trénovací a validační datasety
- **src** – adresář obsahující zdrojové kódy jednotlivých komponent
  - **Readme.txt** – manuál pro spouštění jednotlivých programů a skriptů
  - **dataset-generator** – skripty pro generování trénovacích dat
  - **train** – hlavní trénovací program
  - **eval** – program k vyhodnocování mnatrénovaných modelů
- **trained-models** – natrénované modely neuronových sítí
- **xchadi05-BP-2019.pdf** – text bakalářské práce
- **poster-xchadi05.pdf** – plakát s výsledky práce
- **teaser-video-xchadi05.mp4** – minutové video shrnující celou práci
- **LaTeX** – zdrojové kódy práce v LaTeXu