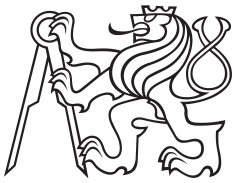
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídicí techniky

Detekce modelů autonomních aut F1/10 na soutěžním okruhu

Lukáš Beneda

Vedoucí: Ing. Joel Matějka
Studijní program: Kybernetika a Robotika
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Beneda** Jméno: **Lukáš** Osobní číslo: **483483**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Detekce modelů autonomních aut F1/10 na soutěžním okruhu

Název bakalářské práce anglicky:

F1/10 autonomous car model detection on a race track

Pokyny pro vypracování:

1. Seznamte se s projektem F1/10 (model autonomního automobilu).
2. Proveďte rešerši algoritmů použitelných pro detekci modelu F1/10 z kamery v reálném čase za využití neuronové sítě. Předpokládejte, že modely na soutěži mohou být vzhledově odlišné.
3. Sestavte potřebný trénovací a testovací dataset ze záznamů z autíčka, případně vizuálně podobných scén.
4. Několik vybraných řešení implementujte. Uvažujte výpočetní výkon dostupný na modelu. Porovnejte výsledky a nejlepší řešení demonstrujte na testovacím datasetu.
5. Vše pečlivě zdokumentujte.

Seznam doporučené literatury:

[1] HASTIE, Trevor, Robert TIBSHIRANI a J. H. FRIEDMAN. The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York: Springer, c2009. Springer series in statistics. ISBN 978-0-387-84857-0.
[2] Object Tracking Evaluation 2012. The KITTI Vision Benchmark Suite [online]. [cit. 2021-01-22]. Dostupné z: http://www.cvlibs.net/datasets/kitti/eval_tracking.php
[3] VAJGL, Marek a Petr HURTIK. A pipeline for detecting and classifying objects in images. In: 2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP) [online]. IEEE, 2020, 2020, s. 163-168 [cit. 2021-01-25]. ISBN 978-1-7281-3214-3. Dostupné z: doi:10.1109/DSMP47368.2020.9204121

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Joel Matějka, katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce:
do konce letního semestru 2021/2022

Ing. Joel Matějka	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Rád bych poděkoval Ing. Joelu Matějkovi za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze, 21. května 2021

Abstrakt

Tato bakalářská práce se zabývá problémem detekce soupeřícího modelu auta na soutěžní dráze F1/10 na snímku z kamery pomocí neuronových sítí. Cílem této práce bylo otestovat několik vybraných řešení a vyhodnotit je. Základem práce je příprava datasetu, návrh neuronových sítí na základě nabytých vědomostí, implementace vybraných řešení a ohodnocení výsledků. Dále práce popisuje postup prací a závěrem jsou porovnány jednotlivé výsledky a vyhodnoceno nejlepší řešení. Výsledkem práce jsou dva otestované koncepty, které jsou použitelné pro provoz na modelu auta F1/10 při nižších rychlostech. Zároveň jsou zde navrženy další postupy a vylepšení.

Klíčová slova: neuronová síť, autonomní model auta, F1/10

Vedoucí: Ing. Joel Matějka
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky – K13135
Karlovo náměstí 13
121 35 Praha 2

Abstract

This thesis deals with the issue of detection of F1/10 autonomous car model on a race track in the image from the camera using a neural network. The aim of this thesis was to test several solutions and evaluate them. In this thesis we explain basic principles and terms of neural networks, which are afterwards used to design, implement and evaluate the result. This thesis describes work progress and at the end the results are compared and the best solution is evaluated. As a result of this thesis are two tested concepts, that can be implemented in our F1/10 model driving lower speed. In the end there are proposed other improvements and procedures.

Keywords: neural network, autonomous car model, F1/10

Title translation: F1/10 autonomous car model detection on a race track

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Hromadný sběr dat a jejich automatizované zpracování je v dnešní době nezbytnou součástí v řadě průmyslových i vědeckých aplikací. Pro automatizované zpracování a vyhodnocení dat se běžně využívají různé algoritmy včetně neuronových sítí, které jsou základem této práce. Vstupní data těchto algoritmů můžou být dnes již různé povahy, ať se jedná o obraz, text nebo zcela odlišné formy.

Rozpoznávání aut pomocí kamer monitorujících provoz na silnicích je běžnou praxí. I v běžném životě se již můžeme setkat s prvními autonomními asistenčními systémy, které taktéž využívají důmyslné algoritmy a pokročilé neuronové sítě natrénované na obrovských datasetech.

V této práci se zabýváme podobnou problematikou, rozdíl je v tom, že jde o detekci modelu auta na soutěžní dráze F1/10 z obrazu kamery auta pro nově vzniklou kategorii závodů více modelů aut najednou, kdy autonomní model auta musí prokázat schopnost detekovat auto soupeře a aktivně upravit svoji trajektorii, tak aby do soupeře nenarazil. Cílem práce tedy je vytvořit základní prostředí pro implementaci a trénování sítě, implementovat vybraná řešení a otestovat je, případně se je pokusit vylepšit co se týče rychlosti a přesnosti. Z těchto důvodů se v práci věnujeme tvorbě datasetu a jeho správě, implementaci a testování dvou konceptů, u nichž jsme pracovali s několika kombinacemi.

Hlavní část práce na neuronových sítích v konkrétních případech, jako je tento, spočívá ve vytvoření kvalitního obsáhlého datasetu, který bude lehko rozšiřitelný a modifikovatelný. Proto jsme velkou část vývoje strávili právě vytvářením datasetu a dalších nástrojů k jeho správě. V další fázi jsme nemalý časový segment věnovali studiu funkčních principů neuronových sítí a navrhli jsme na základě získaných informací dva koncepty, které jsme následně implementovali. Jedním z přístupů je rozřezání vstupního obrázku na segmenty a následné určení pravděpodobnosti výskytu auta v daném segmentu, druhým navrženým přístupem je úprava a natrénování již v praxi využívané neuronové sítě určené k detekci.

Práce je rozčleněná do tří hlavních a jedné menší části, v první části je popsána soutěž F1/10 a základy neuronových sítí, ve druhé části jsou potom představeny jednotlivé koncepty, pomocí kterých jsme problém řešili. Třetí část obsahuje řešení jednotlivých dílčích úloh, kterými byly například tvorba

datasetu a nástrojů pro práci s ním, dále jsou zde popsány funkčnosti nástojů, na konci jsou pak shrnuty výsledky testů. V poslední části jsou shrnuty další poznatky, které se vyskytly během vyhodnocování výsledků a diskuze nad nimi.

Kapitola 2
Teorie

V této kapitole bude představena soutěž F1/10 a nezbytná teorie nutná ke korektnímu návrhu, implementaci a vyhodnocení spolehlivosti řešení.

2.1 Soutěž F1/10

F1/10 je mezinárodní soutěž, ve které soutěží studentské týmy s jejich autonomními modely aut. Soutěž je rozdělena do dvou soutěžních tříd (omezená a otevřená třída) a do dvou kategorií (závod na čas a závod mezi dvěma auty).

Omezená třída má jasně daná pravidla pro použité mechanické komponenty, rozměry modelu nebo výpočetní a senzorovou techniku. Do otevřené třídy spadají všechny ostatní projekty, které splňují omezení rozměrů modelu a typ pohonu. [25]

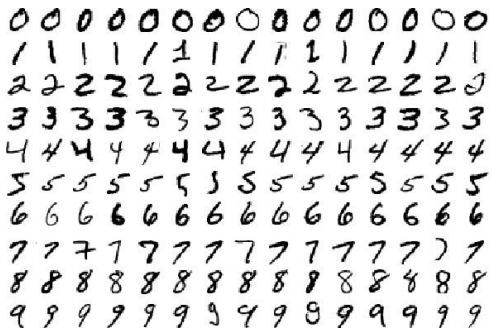
Při závodě na čas se hodnotí dvě kritéria: nejrychlejší kolo a počet ujetých kol bez nehody za omezený časový úsek, zpravidla 3-5 minut. Závodí se v několika kolech, ve kterých zůstává trať stejná. Mezi koly je možné upravit algoritmy, výsledek se počítá nejlepší ze všech kol.

Při závodu mezi dvěma auty musí plánovací algoritmy prokázat schopnost vyhýbat se kromě statických překážek také dynamickým a přitom nehavarovat. Samotný závod poté probíhá tak jak obvykle závody probíhají. Ze startu vyráží soupeři současně a je nutné projet stanovený počet kol. Kdo dojede do cíle první, vyhrává. [25]

Téma této práce je zaměřené na závod mezi dvěma auty, na podúlohu detekce soupeře za pomocí kamery a neuronové sítě.

2.2 Dataset

Definice 2.1. Dataset je definován jako sbírka dat, která je počítačem zpracovávána jako jeden celek. Dataset tedy obsahuje mnoho oddělených částí dat ale mohou být použity k trénování algoritmu s účelem hledání predikovatelných vzorů v datasetu jako v celku. [29, 9]



Obrázek 2.1: Ukázka datasetu MNIST [7]

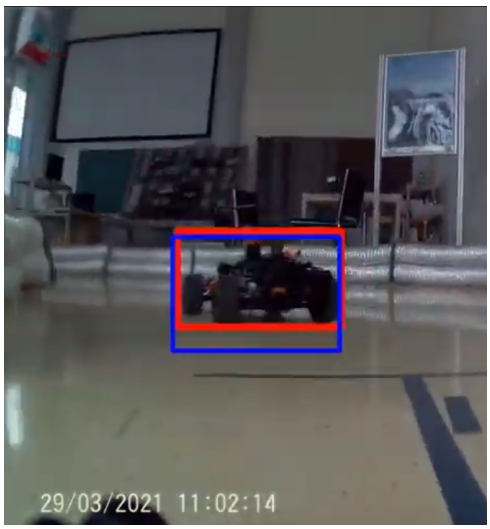
2.2.1 Základní typy anotací

Třída

Klasifikace vzorků do tříd je asi nejčastější úkol řešený pomocí neuronových sítí, v takovém případě každému vzorku odpovídá číslo nebo název jeho třídy. [9] Příkladem může být třeba rozlišování ručně psaných číslic na obrázku 2.1.

Bounding box

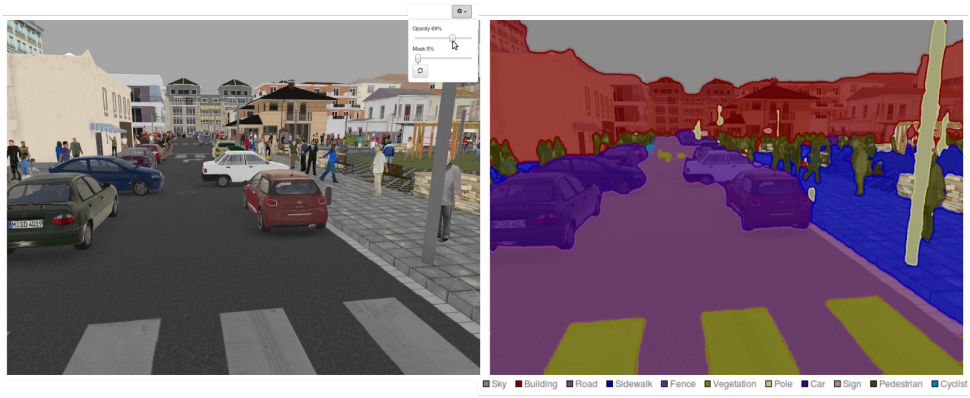
Definice 2.2. *"Axis-aligned bounding box"* ve dvourozměrném prostoru je nejmenší možný obdélník, který obsahuje všechny prvky požadované množiny a zároveň jsou jeho strany rovnoběžné s osami souřadnicového systému. [21]. Bounding box můžeme vidět na obrázku 2.2



Obrázek 2.2: Ukázka bounding boxu

Sémantická segmentace

Sémantická segmentace je typ úlohy pro neuronové sítě, kdy se jako požadovaný výstup sítě nebo jako označení požaduje obrázek stejné velikosti jako je vstupní obrázek, ale tento výstup nese informaci o třídě každého pixelu. Názorná ukázka je na obrázku 2.3. [12]



Obrázek 2.3: Ukázka sémantické segmentace [16]

Tedy vidíme, že jako označení můžeme mít téměř jakoukoli formu dat.

2.2.2 Rozdělení datasetu

Ve většině datasetů je běžné, že se celý dataset dělí na tři části, které mají svůj účel. Největší částí jsou trénovací data a pak dvě menší testovací a validační data. Trénovací data jsou ta, která jsou síti předkládána a síť se podle nich učí. Validační data slouží k otestování sítě, zda nedochází k přeučení. [9] Poslední částí jsou testovací data, na základě kterých později vyhodnocujeme výkon sítě.

2.3 Neuronové sítě

2.3.1 Co je neuronová síť

Definice 2.3. Umělá neuronová síť je výpočetní model. Jeho struktura je inspirována přírodou, konkrétně mozkem zvířat nebo lidí, který se stejně tak jako neuronová síť skládá z jednotlivých uzlů, které jsou navzájem propojené a nazývají se umělé neurony. Každý neuron má vstupní signály, které zpracuje a tím vznikne signál výstupní, který neuron vyšle k dalším neuronům. Signál je vždy reálné číslo, které vznikne ze vstupních signálů definovanou funkcí. Spoje mezi neurony jsou hrany. Každá hrana mezi dvěma neurony má typicky svojí váhu, která mění svoji hodnotu při učení neuronové sítě. [10, 15]

2.3.2 Historie neuronových sítí

V roce 1943 neurofyzik Warren McCulloch a matematik Walter Pitts publikovali v "Bulletin of Mathematical Biophysics" svůj článek "A Logical Calculus of the Ideas Immanent in Nervous Activity", kde představili první model neuronu jakožto základní logickou jednotku, ze kterých by mělo jít jejich konečným počtem sestavit tzv. Turingův stroj.[19] Jejich základní model měl ale několik nedokonalostí a tou hlavní byla schopnost pouze binárních operací nebo neschopnost se učit. O několik let později přináší přístup k modelování "nervových sítí" ke zpracování obrazového materiálu. V následujících letech pak pracoval McCulloch s jeho týmem na výzkumech kolem nervových sítí a tím dal za vznik neuronovým sítím.[19] Americký psycholog Frank Rosenblatt na popud nedokonalostí McCullochova modelu v roce 1958 představil "perceptron", který už netrpěl zmíněnými nedokonalostmi. [23, 11]

Další důležitý objev přišel v 80. letech minulého století, kdy byl představen algoritmus zpětné propagace (backpropagation), který umožňuje výrazně rychlejší učení sítí.

2.3.3 Struktura a základní prvky neuronových sítí

Když se zaměříme detailně na strukturu neuronových sítí, kromě vstupních a výstupních prvků zde nalezneme dva základní stavební prvky, které celou síť tvoří: jednotlivé neurony (perceptrony) a hrany mezi nimi. Vstupem sítě může být téměř cokoliv, co se dá nějakým způsobem reprezentovat reálnými čísly nebo tenzory. Výstupem je pak jiné reálné číslo nebo jiný tenzor.

Neurony a hrany

Neuron je základní stavební jednotka každé sítě. Perceptron je jeho základní model. Perceptron má několik vstupní signálů a jeden výstupní signál. Základní perceptron je funkčně definován jako $f(x)$, kde $x = [x_1, x_2, ..., x_n]$

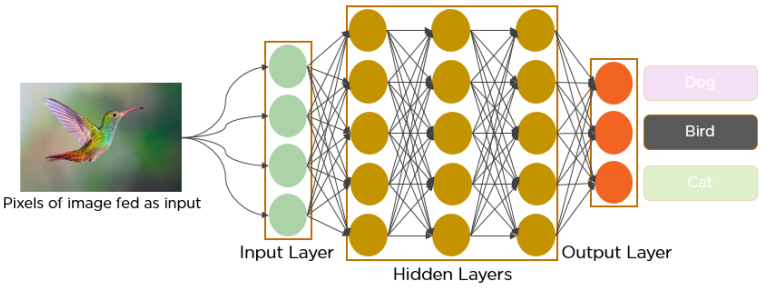
$$f(x) = \begin{cases} 1, & \text{pro } w \cdot x + b \geq 0 \\ 0, & \text{jinak} \end{cases} \tag{2.1}$$

kde w je váhový vektor a b je konstanta. Tedy se jedná o binární klasifikátor, který má skokovou přenosovou funkci. Pokročilejší neurony mají nejrůznější přenosové funkce. Perceptron je tedy jednovrstevná dopředná síť. [23, 9]

Spojováním více neuronů přes hrany vznikají další neuronové sítě. Vhodným propojením jednotlivých neuronů do definovaných, na sebe navazujících skupin vznikají vrstvy sítě. Každá taková vrstva má svoji funkci, definovanou uspořádáním neuronů a jejich vahami. Následným spojováním vrstev vznikají již celé sítě.

Vrstvy sítí

Jak už bylo naznačeno, sítě se skládají z vrstev, které mají všechny svoji funkci. Zde si představíme několik základních typů.



Obrázek 2.4: Ukázka jednoduché neuronové sítě [6]

Lineární/Plně propojená vrstva. Lineární nebo také plně propojená síť/-vrstva je základní vrstva, kde jsou všechny vstupy napojeny do každého neuronu této vrstvy, žádné spojení není vynecháno. Pokud bychom uvažovali dvouvrstevnou architekturu, tak každý výstup z vrstvy první je propojen do každého neuronu vrstvy druhé jako je ukázáno na obrázku 2.4.

Tedy každá taková vrstva může být matematicky reprezentována jako:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.2}$$

kde W je matice jednotlivých vah a b je váhový bias neboli zkreslení vzorků.

Často se ještě využívá logistické funkce sigmoidu, z důvodu reálného charakteru dat. [8]

Konvoluční vrstva. Konvoluční vrstva má na rozdíl od plně propojené vrstvy architekturu chudší na propojení neuronů se vstupními signály. Informaci ze vstupu zpracovává vždy neuron, který je ke vstupu blízký. [9]

Konvoluce v neuronových sítích může být reprezentována například jako:

$$\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} = conv(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}, \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}) \tag{2.3}$$

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22} \tag{2.4}$$

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23} \tag{2.5}$$

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32} \tag{2.6}$$

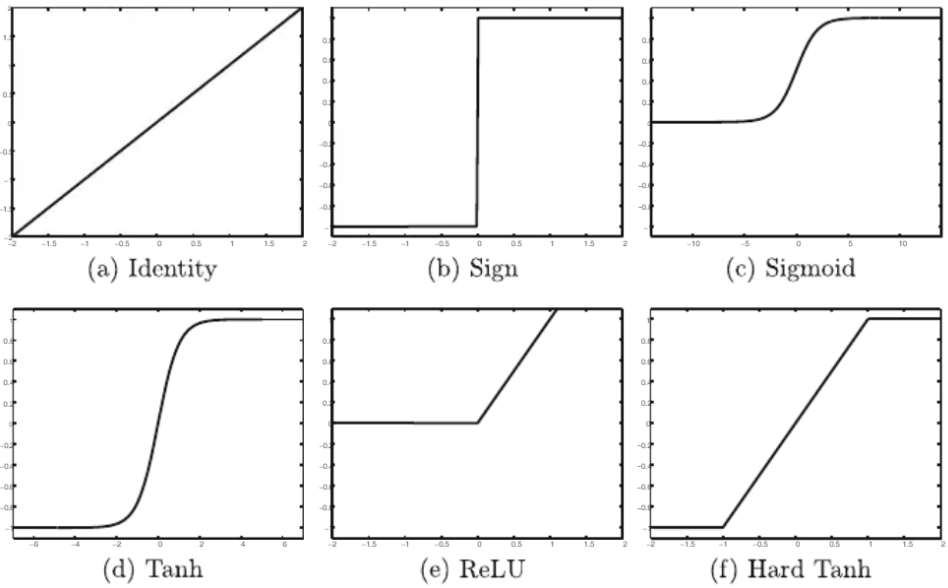
$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33} \tag{2.7}$$

kde W je konvoluční jádro, X jsou vstupní hodnoty a Y jsou výstupní hodnoty.

Každá konvoluční vrstva může mít další vlastnosti, těmi jsou například *stride* (posouvání jádra o počet pixelů, obyčejná konvoluce má $stide = 1$), *pad* (přidání okraje ke vstupu, u základní konvoluce je $pad = 0$) a *dilatation rate* (roztažení konvolučního jádra do stran, kdy uprostřed vzniká prázdný/nulový kříž).

Dále se hojně užívá vícekanálová konvoluce, například pro zpracování barevného obrazu, kdy je konvolučních jader více a každý kanál má své konvoluční jádro. Jádra také mohou mít různé rozměry, ale vždy se zachovává stejná velikost pro všechny kanály. [9]

Aktivační vrstva. Aktivační vrstva nebo také aktivační funkce má v neuronové síti úlohu prahování nebo úpravu míry odezvy na daný vstup. Mezi bohatě užívané patří funkce Sigmoid $\sigma(x)$, hyperbolický tangens $\tanh(x)$, ReLU $max(0, x)$, Leaky ReLU $max(0.1x, x)$, Maxout $max({w_1}^T x + b_1, {w_2}^T x + b_2)$ a ELU $y = x, x \geq 0; y = \alpha(e^x - 1), x < 0$. Některé jejich grafy jsou znázorněné na obrázku 2.5. [22, 9, 2]



Obrázek 2.5: Grafy aktivačních funkcí [2]

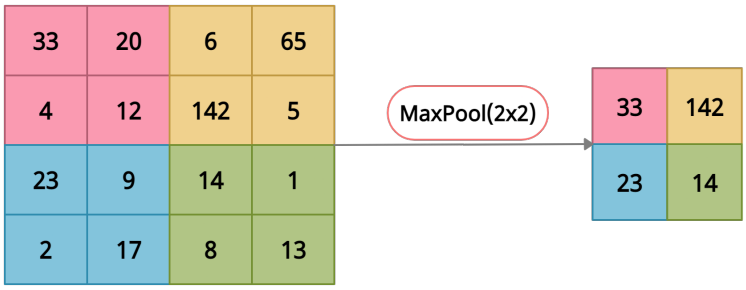
Normalizační vrstva. Každý *batch* (dávka nebo sada dat z datasetu) má své dimenze, u obrazových dat je to typicky čtyřrozměrný objekt, kde dva rozměry jsou rozměry jednotlivých snímků, třetí dimenze je počet barevných kanálů a poslední dimenze je počet jednotlivých obrázků v dané sadě. Normalizační vrstva má za úkol projít všechny kanály a data v nich normalizovat na základě střední hodnoty a standardní odchylky.[26, 14, 9]

Max-pooling. Max-pooling je technika, pomocí které se postupně přikládá jádro a ve výstupu se na příslušném místě objeví pouze ta hodnota, která je maximální ze všech, které jádro pokrývá. Příklad je na obrázku 2.6.[28]

Tvorba sítě

Pokud dnes chceme postavit neuronovou síť, většinou se uchylujeme k řazení jednotlivých vrstev za sebe. Někdy se pak užívá i tzv. "skip connection", tato metoda spočívá v propojení výstupu nějaké předchozí vrstvy do vrstvy, která se nachází hlouběji v síti. Takováto data potom poskytují určitou informaci navíc.

Běžně se také používá předtrénovaných sítí, kdy se vezme velká část již hotové sítě a napojíme na ni vlastní část. Lze samozřejmě využít i více sítí,



Obrázek 2.6: Max-pooling příklad [18]

jejichž výsledky na konci spojíme a vytvoříme tak požadovaný výstup.

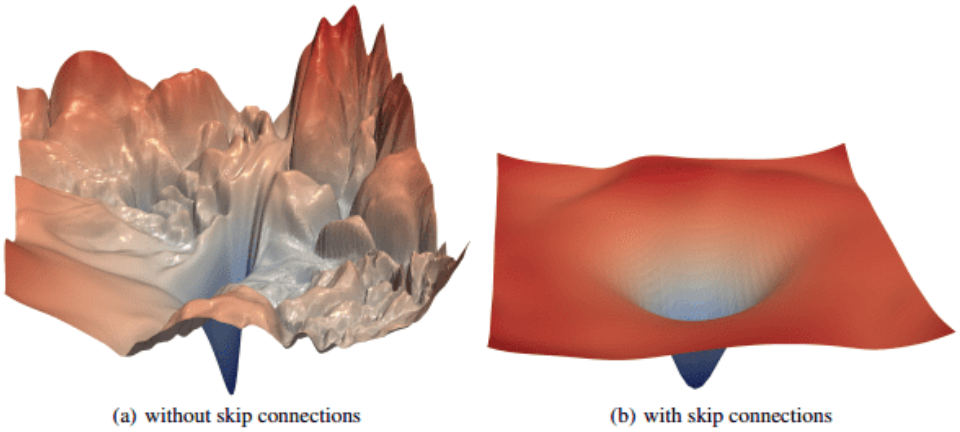
2.4 Trénování sítí

Při trénování neuronové sítě probíhají obecně dva kroky, *feed-forward* (postup sítí vpřed) a *backpropagation* (zpětná propagace gradientu), což je postup, díky kterému zjistíme, jak se daná váha neuronu promítne s daným vstupem na výstup sítě. Dále se na tento algoritmus navazují různé algoritmy využívající této zpětné propagace. Což je rychlé a celkem robustní řešení, ale má i své nevýhody. Neuronovou síť si můžeme představit jako n-dimenzionální funkci, což už samo o sobě může znít výpočetně složitě a náročně. V takové funkci hledáme minimum dle nějakého kritéria, které je schopné objektivně zhodnotit výstup naší sítě oproti požadovanému výstupu. Takové kritérium je také funkce, říkáme jí ztrátová funkce, často označovaná jako *loss function*, při trénování minimalizujeme právě tuto funkci. Příklad takové funkce může být na obrázku 2.7, kde vlevo vidíme vizualizaci neuronové sítě spojené se ztrátovou funkcí, která byla použita k sémantické segmentaci, vpravo poté jak se změnila přidáním *"skip connections"*. Tedy můžeme vidět, že na grafu vlevo je mnoho lokálních minim a na jednodušším grafu vpravo je jich o poznání méně a hledání globálního minima může být značně lehčí úlohou, tedy je jasné, že učení podle gradientu nemusí vždy dosáhnout optima, hlavně v případě, pokud máme takto složitou síť. [13, 1]

Dalším úskalím při trénování může být přeučení nebo přetrénování sítě, což je stav, kdy úspěšnost neuronové sítě na trénovacích datech roste a na validačních datech klesá. Tento jev se vyskytuje po mnoha průchodech skrz dataset, můžeme tomu zabránit včasným přerušením tréninku, pokud jsme schopni sledovat vývoj ztrátové funkce nebo průběžně kontrolovat úspěšnost sítě.

2.4.1 Trénovací algoritmy

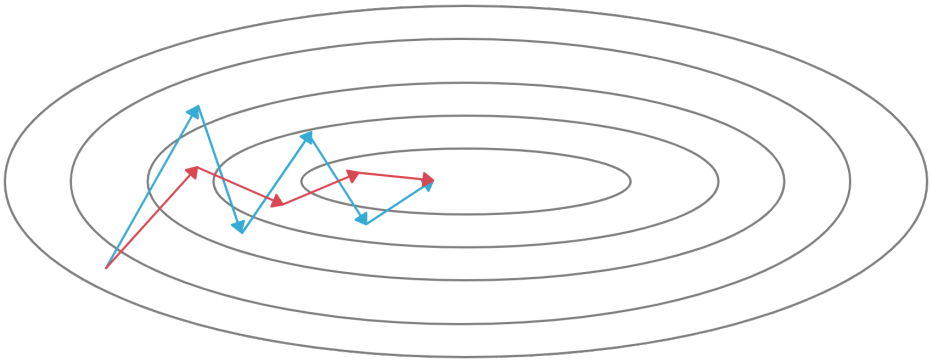
SGD. *Stochastic gradient descent* je iterativní algoritmus používaný pro optimalizaci. Je založený na úpravě hledané hodnoty na základě gradientu, konkrétní aplikace je v rovnici 2.8. Chování této metody je vyobrazeno na



Obrázek 2.7: Vizualizace ztrátové funkce [17]

snímku 2.8 modře. [3, 24]

$$w_k = w_{k-1} - \alpha \frac{\partial f^T(w)}{\partial w}\bigg|_{w=w_{k-1}} \tag{2.8}$$



Obrázek 2.8: Grafické znázornění algoritmu SGD [27]

Rozšířením algoritmu může být menší úprava a vznikne algoritmus *SGD + momentum* se vzorcem 2.9. Jeho průběh je na snímku 2.8 červeně. [24]

$$v_k = \beta v_{k-1} - \frac{\partial f^T(w)}{\partial w}\bigg|_{w=w_{k-1}} \tag{2.9}$$

$$w_k = w_{k-1} + \alpha v_k \tag{2.10}$$

Další úpravou algoritmu může být tzv. *Nesterov momentum*, které se liší jen málo od obyčejného momentu. Úpravu můžeme vidět ve vzorci 2.11. [24]

$$v_k = \beta v_{k-1} - \frac{\partial f^T(w)}{\partial w}\bigg|_{w=w_{k-1}+v_{k-1}} \tag{2.11}$$

$$w_k = w_{k-1} + \alpha v_k \tag{2.12}$$

Newtonova metoda. *Newtonova metoda* je iterační optimalizační metoda, která využívá Hessovu matici. Tato metoda je velmi rychlá a vždy účinná, ale bohužel kvůli Hessově matici je velmi výpočetně náročná, není tedy vhodná pro učení velkých sítí. [24] Její iterace je naznačena vzorcem 2.13.

$$w_k = w_{k-1} - \alpha H^{-1} \frac{\partial f^T(w)}{\partial w}\bigg|_{w=w_{k-1}} \tag{2.13}$$

Další algoritmy. Mezi další algoritmy patří například *RMSprop*, *Adam* nebo *AMSgrad*. [24] Tyto iterační algoritmy jsou již složitější a byly představeny v několika posledních letech.

2.4.2 Parametry trénování

Learning rate/Koeficient učení. U všech metod při učení volíme koeficient učení, tím určujeme, jak se bude gradient promítat do vah, které se snažíme optimalizovat. Volba koeficientu učení může významně ovlivnit proces učení, je proto dobré koeficient postupně během učení měnit.

Batch size/Velikost sady. Při učení je dalším důležitým parametrem velikost dávky dat, která je předložena síti ke zpracování. Čím větší je dávka dat, tím více se dat se v jednu chvíli dostane do sítě a na jejich základě se upraví váhy v neuronech, tedy úpravy jsou konzistentnější.

2.5 Hodnocení výkonu sítě

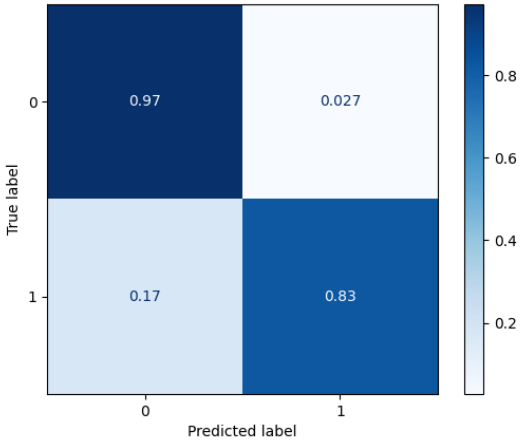
Když učíme síť, někdy nám může subjektivně přijít, že podává dobrý výkon, ale ve skutečnosti to tak být nemusí. Zavádí se proto objektivní kritéria hodnocení výsledků sítě, která si popíšeme níže. Zároveň se ale používají standardní statistická měřítka. Většinou se tyto hodnoty vypočítávají z testovacích dat, můžeme si je ale vypočítat i pro data trénovací, abychom byli informovaní o stavu sítě.

2.5.1 Confusion matrix/Matice záměn

Matice záměn je matice, která obsahuje údaje o úspěšnosti klasifikátoru. Ve sloupcích bývá predikované označení, v řádcích pak skutečné označení. Údaje v jednotlivých polích matice jsou počty nebo poměrná část z celku takto klasifikovaných objektů. V ideálním případě by měla matice nenulové hodnoty pouze na hlavní diagonále. Na obrázku 2.9 je příklad takové matice. [9, 30]

2.5.2 Křivka ROC

Receiver Operating Curve je křivka, která znázorňuje úspěšnost klasifikátoru v závislosti na nastavení jeho prahové hodnoty. S touto křivkou se vážou pojmy *TPR - true positive rate* a *FPR - false positive rate*, které vyjadřují pravděpodobnost správné detekce a pravděpodobnost falešného poplachu.

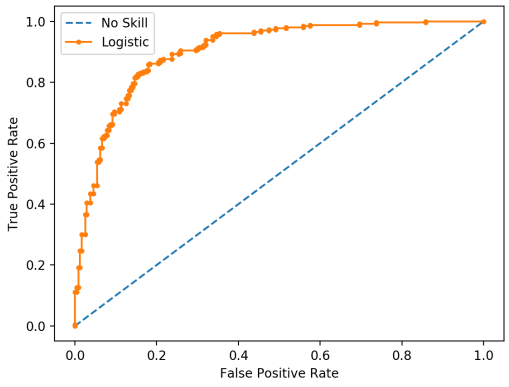


Obrázek 2.9: Matice záměn

Vzorce 2.14,2.15 vyjadřují jejich význam, kde TP je *true positive*, TN je *true negative*, FP je *false positive*, FN je *false negative* a obrázek 2.10 zobrazuje ROC křivky. [9]

$$TPR = \frac{TP}{TP + FN} \tag{2.14}$$

$$FPR = \frac{FP}{FP + TN} \tag{2.15}$$



Obrázek 2.10: ROC křivka [5]

2.5.3 Precision, Recall, Accuracy

Precision je další metrika založená na datech z matice záměn,*recall* je v tomto smyslu to samé jako *TPR*. *Precision* nám říká, jakou část z pozitivně

označených dat jsme určili správně, zatímco *recall* říká, jaká část byla určena správně ze všech možných dat.

$$precision = \frac{TP}{TP + FP} \tag{2.16}$$

Accuracy je metrika definovaná vzorcem 2.17, který ovšem může mít zkreslenou vypovídající hodnotu. Zavádí se proto *balanced accuracy* 2.18 nebo *predicted positive condition rate* 2.19. [9, 30]

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.17}$$

$$balanced\ accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.18}$$

$$predicted\ positive\ condition\ rate = \frac{TP + FP}{TP + TN + FP + FN} \tag{2.19}$$

2.5.4 F1 score

Tato metrika kombinuje *precision* a *recall* následovně. [30]

$$F1 = 2\frac{precision \cdot recall}{precision + recall} \tag{2.20}$$

Tento vztah vychází z obecného *F-score*, které využívá v tomto případě harmonického průměru a kombinuje tak pomocí stejných vah *precision* a *recall*. To nám zajišťuje vyváženost obou hodnot při jejich maximalizaci.

Kapitola 3
Návrh

3.1 Návrh datasetu

3.1.1 Návrh obsahu datasetu

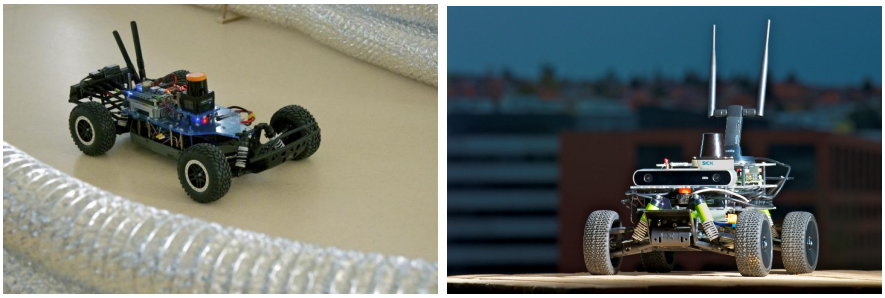
Na soutěži F1/10 se pohybují modely aut podobné autu na obrázku 3.1. Takové auto chceme detekovat v obraze z kamery, která je umístěná na autě vpředu. Jako efektivní způsob, jak vytvořit obsáhlý dataset, jsme navrhli natočení videí kamerou na autě, ke kterým jsme následně vytvořili anotace (*labely*). Abychom měli dataset vyhovující i po kvalitativní stránce, museli jsme se zajímat více i o reálný obsah videí a tedy při pořizování videí vytvořit rozmanité prostředí a různé scény. Stanovili jsme tedy, že v datasetu nesmějí chybět následující situace:

- Následování jiného auta
- Předjíždění jiného auta
- Jízda po prázdné dráze
- Jízda po dráze s překážkami
- Jízda po dráze s více auty
- Další kombinace již zmíněných situací

Zároveň pokud to bylo možné, snažili jsme se videa zachytit v různých světelných podmínkách.

3.1.2 Návrh struktury

Strukturu datasetu jsme navrhovali tak, aby byla lehko modifikovatelná, přehledná a případné úpravy obsahu nebo rozdělení datasetu nebyly náročné. Složku s datasetem jsme proto rozdělili na dvě části. Jednou částí jsou zdrojová videa a zdrojový soubor s *labely* a druhou extrahovaná data. K extrakci dat jsme proto navrhli program, který je schopný vygenerovat dataset ze zdrojových souborů, přidat úplně nové video nebo umožní vytvořit *labely*.



Obrázek 3.1: Model auta pro soutěž F1/10 [4, 20]

Složka s extrahovanými daty potom obsahuje soubor s informacemi o obsahu datasetu ve formátu *.json*.

Dataset jsme podle teorie rozdělili na 3 části, kterými jsou:

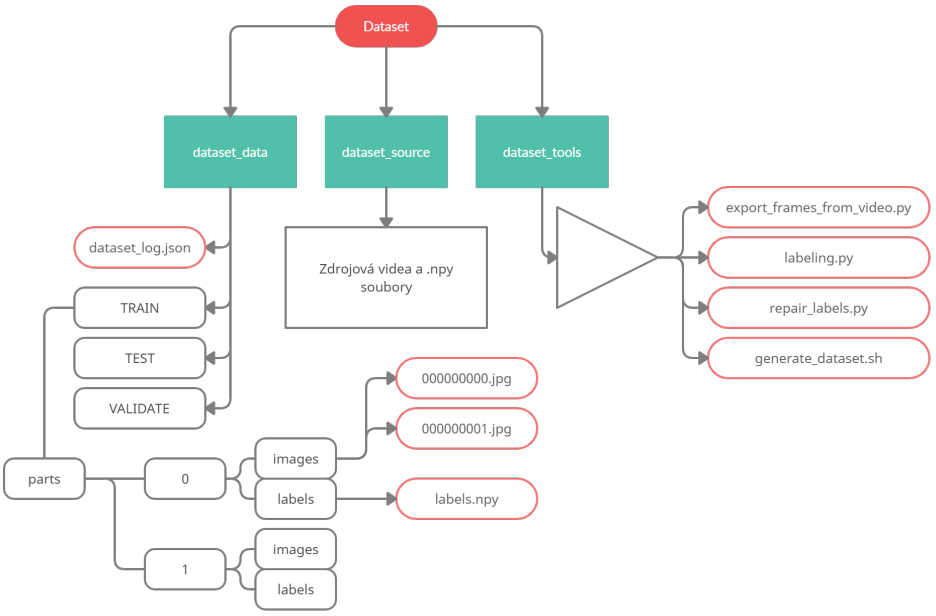
- trénovací data - *TRAIN*
- testovací data - *TEST*
- validační data - *VALIDATE*

Ty jsme umístili každé do vlastní složky, jejichž názvy jsme zvolili dle jejich účelu *TRAIN*, *TEST* a *VALIDATE*, aby už od počátku byly jednotlivé části oddělené. V každé této složce jsme navrhli složku, která obsahuje už složky s daty podle jednotlivých videí. Složku jsme si označili *parts*. Tuto složku jsme navrhli proto, aby zde vznikl volný prostor pro případné doplňující soubory, jako by mohly být například soubory s informacemi o jednotlivých videích nebo logovací soubory. Ve složce *parts* se vždy při přidání videa do datasetu vytvoří složka pojmenovaná podle identifikačního čísla videa, které mu je přiřazeno při zařazení do datasetu, ve které se vždy ještě data rozdělují na snímky a anotace do složek *images* a *labels*. Do složky *images* jsou potom extrahovány jednotlivé snímky z videa, které označujeme číslem podle pořadí v příslušícím videu. Ve složce *labels* potom ukládáme vytvořený soubor s labely ve formátu *.npy*. Struktura uložení dat je ilustrovaná na obrázku 3.2a.

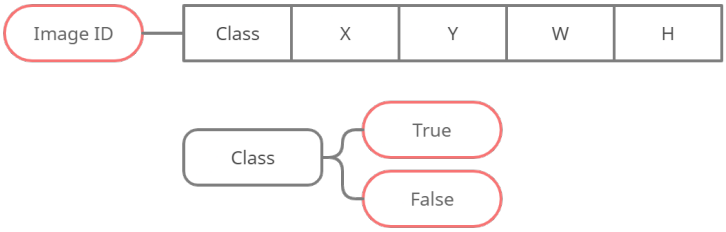
V souboru s anotacemi jsou obsaženy vždy anotace pro celé jedno video, prozatím jsme se spokojili s jedním *labelem* pro jeden snímek, popřípadě jsme si nechali otevřené dveře pro více anotací. Pro označení jsme využili standardní formu *bounding boxu* podle definice 2.2. Ten ukládáme ke každému snímku jako pětici čísel:

- *true/false* podle toho, zda auto na snímku je nebo není
- *x,y* souřadnice levého horního rohu bounding boxu
- *w,h* šířka a výška bounding boxu

Anotaci snímku ukládáme takto do pole pod indexem čísla snímku, případně *label* můžeme rozšířit o další bounding box, který k němu přidáme do dalšího pole. Formát uložení labelu si můžeme prohlédnout na obrázku 3.2b



(a) : Struktura datasetu



(b) : Struktura labelu

Obrázek 3.2: Ukázka struktury datasetu a labelu

3.1.3 Návrh nástrojů k datasetu

K datasetu jsme navrhli několik nástrojů. Jedním z nich je nástroj, který vygeneruje data ze zdrojových souborů a správně je poskládá do složek, dále je to například program, který umožňuje prohlížet jednotlivé části datasetu a manuálně upravovat jejich anotace.

Nástroj pro generování datasetu ze zdrojových souborů. Tento nástroj jsme navrhovali tak, aby uměl ze zdrojových souborů vygenerovat celý dataset úplně od počátku znovu. Jednou z jeho součástí je skript sloužící k přidání nového videa do datasetu. Ke každému videu, ke kterému nespecifikujeme soubor s labely, pak program automaticky vytvoří základní soubor s *labely*, které nesou informaci, že označení je neplatné, respektive mají význam takový, že jsme anotaci nespecifikovali, a že daný snímek nebyl stále ručně označen.

Nástroj pro prohlížení a upravování anotací. Tato aplikace je navržena tak, aby byla interaktivní a aby nezůstala pouze u výpisů na terminál, ale hlavně aby zobrazovala snímky a jejich anotace tak, jak je požadováno. Volně pak navazuje na předchozí nástroj, který pouze vygeneruje soubor s labely, kde se nenachází žádná určitá data. Až potom tento nástroj umožňuje tyto data jednoduše interaktivně vyplnit. Auto na snímku je označováno myší a zbylé vhodné příkazy vyplynuly a byly přidány dle potřeby a pohodlnosti až při implementaci, aby označování modelu auta probíhalo co nejrychleji a nejjednodušeji.

3.2 Návrh neuronové sítě

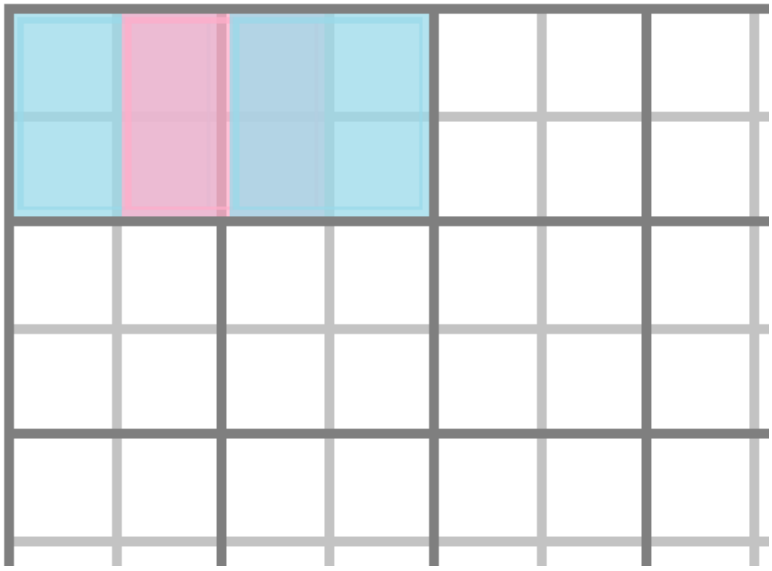
Pro určení polohy soupeřova auta je třeba vědět, kde se auto nachází v obraze z kamery. Pomocí těchto informací jsme potom schopni určit polohu soupeřova auta. Pro detekci polohy auta soupeře v obraze jsme se rozhodli otestovat několik různých přístupů a zjistit, který je pro naši aplikaci nejvýhodnější a zda jsme vůbec schopni, vzhledem k výkonu výpočetní jednotky na autě, síť na autě provozovat s dostatečnou rychlostí výpočtů. Vzhledem ke složitosti některých postupů jsme se rozhodli postupovat od jednodušších ke složitějším. Začali jsme tedy u experimentů s několika předtrénovanými sítěmi, na které jsme napojovali vlastní koncové vrstvy. Ve druhé fázi jsme využili již existující sítě určené k detekci objektů a upravili ji k použití na detekci soupeřova modelu F1/10.

V prvním případě jsme rozdělili snímek na části a síť měla za úkol určovat pravděpodobnost, s jakou se model auta nachází v této části obrázku. Ve druhém případě jsme využili sítě určené k detekci objektů *faster R-CNN*.

3.2.1 Rozdělení na segmenty

Při detekci objektu v obraze se využívá předdefinovaných boxů, které obraz rozdělí na části, ze kterých se následně vyberou ty s vyšším potenciálem, že obsahují hledaný objekt, a následně se zpracují. Na základě této znalosti jsme navrhli jednoduché rozdělení na menší části a u každého takového segmentu jsme určovali pravděpodobnost, s jakou se na něm auto nachází. O této úloze můžeme mluvit jako o binární klasifikaci jednotlivých segmentů. Mezi nejznámější sítě používané pro klasifikaci patří například *ResNet*, dále pro *low power* systémy *Squeezenet* a nebo *Mobilenet*. Pro náš případ jsme tyto tři využili a navázali na ně algoritmus nejbližšího souseda, který se většinou používá k otestování konceptu nebo pro nenáročné aplikace. Po otestování jsme algoritmus nejbližšího souseda nahradili vrstvami plně propojenými nebo konvolučními tak, abychom dosáhli co nejlepších výsledků. Předpokládali jsme, že toto řešení bude pomalejší, ale o to bude snadnější učení sítě a implementace.

Jak už bylo nastíněno, snímek jsme rozdělili na předdefinovanou obdélníkovou síť a každý takový obdélník byl vstupními daty sítě. Tento obdélníkový obrázek jsme nechali klasifikovat jako 1, pokud tam se auto v obdélníku

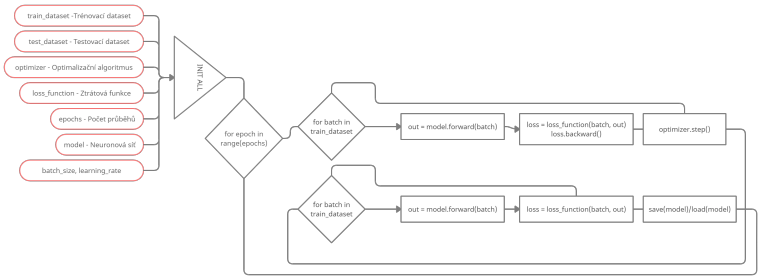


Obrázek 3.3: Obdélníkové rozdělení

nachází, nebo 0, pokud se auta v obdélníku nenachází. Výsledným složením výstupů do původní obdélníkové sítě získáme mapu pravděpodobnosti, kde se auto nachází. Takovou mapu potom můžeme použít různými způsoby nebo na ni navázat mnoho jiných algoritmů. Další možností se ukázalo být učení sítě podle překryvu obdélníku s bounding boxem auta. Tedy vzhledem k velikosti bounding boxu a velikosti obdélníku jsme předpokládali, že budeme schopni určit přesněji, kde se auto nachází. Obdélníkové rozdělení jsme později vylepšili o poloviční překryv, tedy základní obdélník posouváme vždy o polovinu obdélníku tak, aby se jednotlivé segmenty překrývaly jako na obrázku 3.3, tím jsme dostali data pro jednoduchý průměrovací algoritmus a předpokládali jsme zlepšení ve výsledcích i větší rozlišení mapy pravděpodobnosti. Dále jsme uvažovali o vylepšení, které by implementovalo zpracovávání všech segmentů najednou v jedné síti, díky čemuž by nedocházelo k duplikování výpočtů na velké části dat.

3.2.2 Architektura *faster R-CNN*

V případě detekční architektury jsme využili praxí ověřenou síť *faster R-CNN*, která využívá předtrénovaných klasifikačních sítí jako svého základu/páteře (*backbone*). Rychlost této páteře je jedním z důležitých faktorů, jak ovlivnit celkovou rychlost výpočtu. Za prvé jsme síť upravili, aby detekovala pouze jeden typ objektu a to auto soupeře na dráze. Dále jsme vyzkoušeli různé sítě (*ResNet*, *Mobilenet*, *Squeezenet*, *VGG*,...) jako páteř pro *faster R-CNN*.



Obrázek 3.4: Blokové schéma trénovacích funkcí

3.3 Návrh trénovacích funkcí

3.3.1 Trénovací funkce pro klasifikaci jednotlivých segmentů

Při trénování neuronové sítě se využívá iteračních algoritmů k úpravám vah neuronů sítě. K experimentům jsme vybrali Adam a SGD. Ztrátovou funkci jsme volili podle typu požadovaného výstupu. Vyzkoušeli jsme jich hned několik, například L1Loss, BCELoss nebo MSELoss a další podobné.

Před začátkem jsme si vždy stanovili počet průběhů skrz trénovací data, v hlavní smyčce, která zajišťuje průběh skrz dataset, síti předkládáme data po menších sadách a porovnáváme výstup sítě s požadovaným výstupem pomocí ztrátové funkce, na jejíž hodnotu jsme vždy aplikovali algoritmus zpětné propagace gradientu a iterační algoritmus pro úpravu vah pak zajistil jejich úpravu. Po projití celého trénovacího datasetu vypočítáme průměrnou hodnotu ztrátové funkce a dále pak použijeme stejným způsobem (mimo úpravu vah v neuronech) validační data k orientačnímu ohodnocení sítě, abychom zjistili, zda se síť nepřeučuje. Pokud by se průměrná hodnota ztrátové funkce na validačních datech zhoršovala, vrátíme se s hodnotami vah zpět před poslední průchod a změníme koeficient učení. To by nám mělo alespoň částečně zaručit, že se síť nepřeučí. Tento postup můžeme pořád dokola opakovat, dokud nenalezneme nějaké vyhovující řešení. Přibližná struktura průběhu je ilustrována na obrázku 3.4.

3.3.2 Trénovací funkce pro *faster R-CNN*

Stejně jako u klasifikace segmentů jsme si zvolili algoritmus pro optimalizaci vah. Tentokrát jsme ale používali pouze SGD. Jelikož jsme předpokládali dlouhé učení sítě, než dosáhneme uspokojujících výsledků, zvolili jsme vyšší počet průběhů. Po konstrukční stránce je struktura funkce podobná jako u předchozí varianty s tím rozdílem, že počítáme několik různých hodnot ztrátových funkcí, jelikož síť nemá pouze jeden výstup. Ty následně sečteme, provedeme zpětnou propagaci gradientu a upravíme váhy. Podobný postup aplikujeme i u validačních dat a ověříme tím stav sítě.

Kapitola 4
Realizace

Pro implementaci jsme vybrali programovací jazyk *python*. K implementaci neuronových sítí je využito frameworku *PyTorch* společně s knihovnou *OpenCV*. *PyTorch* je open-source framework sloužící k implementaci strojového učení. Dále je pak minimálně využit *shell script* k zautomatizování některých procesů.

4.1 Tvorba datasetu

Při pořizování videí do datasetu jsme v laboratoři na dráze natočili videa s rozlišením *1920x1080* pixelů se snímkovou frekvencí 30 nebo 60 fps. Během dvou dní jsme pořídili 11 různých videí a později jsme je doplnili o 5 starších videí z internetu.

Z každého videa jsme vyexportovali jednotlivé snímky do příslušné složky pomocí nástroje pro generování datasetu. Potom jsme postupně vytvořili pro každý snímek vlastní anotaci dle návrhu a spojili je do jednoho souboru pro každé video dle pořadí snímků ve videu. Vizualizace takového snímku s anotací je vidět na obrázku 4.1, kde modrý obdélník je ručně označené auto a červený obdélník je již predikce naší sítě.

4.1.1 Parametry datasetu pro F1/10

Pro vypsání statistiky datasetu jsme vytvořili jednoduchý skript v jazyce *python*, nalezneme ho ve složce *dataset/dataset_tools* pod názvem *dataset_stats.py*, ten nám po spuštění vypíše do příkazové řádky číselně obsah jednotlivých částí datasetu. Příklad takového výpisu je na obrázku 4.2. Skript se spouští bez jakýchkoli příznaků jako běžný *python* program.

Aktuálně celý dataset čítá na 53942 snímků, z toho 16352 s pozitivní anotací a 37589 s anotací negativní. Mezi další vlastnosti můžeme zařadit i vysoké rozlišení snímků, které je *1920x1080* pixelů.

4.1.2 Nástroj pro generování datasetu z videa

Nástroj pro generování datasetu z videa je implementován ve skriptovacím jazyce *shell script*. Jeho funkcí je vygenerovat dataset ze zdrojových souborů

Obrázek 4.1: Snímek z datasetu

```
(BP) ~/.../dataset/dataset_tools >>> python dataset_stats.py
Section ../dataset_data/TRAIN/ contains (positive:negative - not anotated):   15285:37143 - 0
Section ../dataset_data/TEST/ contains (positive:negative - not anotated):   698:352 - 0
Section ../dataset_data/VALIDATE/ contains (positive:negative - not anotated):   370:94 - 0
(BP) ~/.../dataset/dataset_tools >>>
```

Obrázek 4.2: Příklad výpisu statistiky datasetu

úplně od začátku. Skript je ve složce projektu *dataset/dataset_tools* pojmenován *generate_dataset.sh*. Aplikace při spuštění vytvoří složku *dataset/dataset_data*. A pomocí programu *extract_frames_from_video.py* vyexportuje snímky z jednotlivých videí a přiřadí soubor s labely do správné složky. Po přidání videa do datasetu je důležité tento skript rozšířit, aby se zachovala úpnost datasetu i po jeho přegenerování. Podobně je k dispozici i skript *dataset_reset.sh*, který dataset vymaže. Program *extract_frames_from_video.py* přijímá při spuštění několik parametrů, ty jsou popsány v tabulce 4.1. Program se zároveň používá jako nástroj pro přidání nového videa do datasetu. Je vhodné zdrojové video umístit do složky *dataset_source* a zároveň s ním i soubor s *labely*, pokud ho máme.

Příznak	Význam hodnoty
-f	cesta ke zdrojovému videu
-lf	cesta k souboru s anotacemi
-dp	část datasetu (TRAIN, TEST, VALIDATE)

Tabulka 4.1: Tabulka parametrů nástroje pro extrahování snímků z videa

4.1.3 Nástroj pro ruční označení jednotlivých snímků v datasetu

Nástroj pro ruční označování snímků je k dispozici opět ve složce projektu *dataset/dataset_tools*. Aplikace je stejně jako většina ostatních programů psána

v jazyce *python*, ve složce jí najdeme pod názvem *labeling.py*. Program přijímá při spuštění několik parametrů, ty jsou popsány v tabulce 4.2. Tento program umožňuje postupně projít jednotlivé části datasetu a detailně prohlédnout jednotlivé anotace snímků a případně je upravit. Ovládání je následující:

- Šipka vlevo - zobrazí předchozí snímek
- Šipka vpravo - zobrazí následující snímek
- Pravé tlačítko myši - kliknutí vymaže aktuální anotaci
- Levé tlačítko myši - stisknutím začne kreslit nový obdélník, uvolněním kreslit přestane
- Mezerník - potvrdí aktuálně nastavenou anotaci a uloží ji do souboru s anotacemi
- Šipka nahoru - potvrdí aktuálně nastavenou anotaci pro 10 snímků (aktuální a následujících 9); použití je převážně pro urychlení anotování prázdných snímků
- Q - uloží a korektně ukončí aplikaci

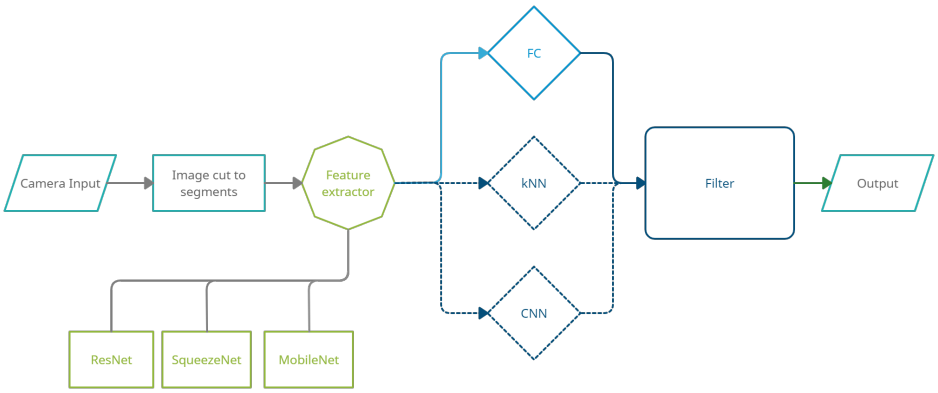
Příznak	**Význam hodnoty**
-if, -part_folder	cesta ke složce části datasetu, např: *"…/parts/0"*
-lf, -labels_file	soubor s anotacemi, automaticky podle části datasetu
-is, -image_size	rozlišení obrázků, automaticky (1920,1080)

Tabulka 4.2: Tabulka parametrů nástroje pro ruční označování

4.1.4 Další užitečné nástroje

V průběhu práce vznikly další užitečné nástroje, které usnadnily například hledání chyb nebo nebo prohlížení logovacích souborů. Mezi ně patří například nástroj, který umožňuje rychlou prohlídku sekce datasetu, nebo nástroj k rychlé opravě neplatných anotací. Další důležité skripty slouží k prohlížení logovacích souborů, do kterých se ukládá postupně hodnota ztrátové funkce, tyto skripty z těchto souborů vygenerují a zobrazí graf, kde si potom můžeme prohlédnout průběh učení sítě.

Rychlé prohlížení datasetu. Tento velmi užitečný nástroj je pojmenován *watch_dataset_section.py* a nachází se ve složce *dataset/dataset_tools*. Jakožto příznak přijímá pouze *-s [SECTION]*, kde volbou může být pouze *TEST, TRAIN* nebo *VALIDTE*. Po spuštění začne program postupně zobrazovat celou sekci datasetu s anotacemi jako video.



Obrázek 4.3: Struktura sítí při prvních pokusech

Oprava neplatných anotací. Skript nazvaný *repair_labels.py* v adresáři *dataset/dataset_tools* má za úkol projít všechny anotace v daném souboru a zkontrolovat jejich platnost. Soubor skriptu specifikujeme přes příznak *-lf*, ke kterému přidáme cestu k souboru s labely. Nástroj opraví anotace takové, kdy je například třída uvedená jako *True* a zároveň je některý z údajů o bounding boxu číslo záporné nebo když jsou 3 některé údaje o bounding boxu nulové. Další případy lze jednoduše implementovat.

Nástroje pro prohlížení logů. Tato kategorie nástrojů momentálně čítá dva různé skripty, které jsou ve složce *graph_tools*. Soubor s odpovídajícím názvem *graph_generating.py* s volbou *-f [FILE]* zobrazí graf hodnot z daného souboru. Druhým je *detailed_graph_generating.py*, který umožňuje s volbou *-f [FILE] -e [EPOCH]*, kde *FILE* je jméno souboru bez zakončení *_e_5.log*, které nese informaci o čísle průběhu datasetem (zde například je to číslo 5, tedy šestý průběh, jelikož indexujeme od nuly), a kde *EPOCH* je číslo průběhu, kde graf bude končit, vygenerovat detailnější graf průběhu učení.

4.2 Neuronová síť a její trénování

Jak už bylo zmíněno v předešlé kapitole, vyzkoušeli jsme dva různé přístupy k problému. Jedním je rozřezání na segmenty a druhým je upravení architektury *faster R-CNN* určené k detekci objektů.

4.2.1 Rozdělení na segmenty

Prvním přístupem je rozřezání snímků na segmenty a následné přiřazování třídy celému segmentu. K experimentu jsme využili tři předtrénované sítě a napojili na ně naše dvě vlastní vrstvy a algoritmus nejbližšího souseda. Jako předtrénované sítě jsme použili sítě: *Resnet18, Mobilenet v2 a Squeezenet 1.0*, které jsme označovali jako *feature extractor*. Napojili jsme na ně plně propojenou nebo konvoluční vrstvu. Tato struktura je znázorněná na obrázku 4.3.

Při prvních pokusech jsme snímek zmenšili a zkoušeli ho rozřezat obdélníkovou sítí *4x2* segmenty plus posun o polovinu. Tedy jsme celkově měli 21 segmentů. Každý takový segment po vstupu do sítě vrátil jedno výstupní číslo, které bylo mezi 0 a 1. Spojením těchto čísel do vektoru jsme dostali jakousi mapu, kterou jsme vložili do filtru, který pracoval na principu návaznosti jednotlivých snímků videa, čímž se značně zlepšil celý výstup systému. Bohužel ale takovýto systém byl velmi pomalý a nebyl ani zdaleka tak přesný, jako jsme si představovali. Výstup takového systému byl například takový jako na obrázku 4.4a, ale i takový jako na obrázku 4.4b.

(a) : Uspokojivý výstup **(b) :** Špatný výstup

Obrázek 4.4: Vizualizace výstupu při prvních pokusech

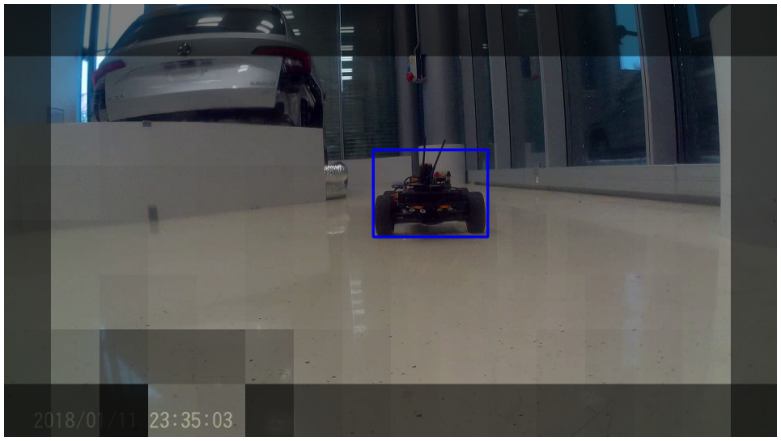
Pro všechny varianty a kombinace vypadaly výstupy velmi podobně a nebyli jsme s nimi spokojeni, rozhodli jsme se proto uchýlit ke zvětšení počtu segmentů. Zvolili jsme proto rozdělení na *8x4* segmenty, což je v celku 105, a snažili jsme se o lepší výsledky. Zde už jsme používali pouze *Resnet18* a plně propojenou vrstvu, jelikož tato kombinace měla lepší výsledky než ostatní. Dále jsme změnili strategii a síť byla učena podle překryvu segmentu s bounding boxem auta, tuto hodnotu jsme ale měnili skokově, tedy jsme síť učili podle čísel 0, 0.3, 0.6 a 1. Tato změna se ale ve výsledném výstupu příliš neprojevila. Dalším pokusem o lepší výsledek byla změna aktivačních funkcí nebo třeba vynechání některé normalizační vrstvy.

Bohužel se ale tento přístup neosvědčil a síť měla výstup horší, než bylo očekáváno. Příklad takového výstupu je vidět na obrázku 4.5, který není ani trochu vyhovující (modrý obdélník je opět označen člověkem, výstup sítě je vizualizován ztmavením části obrazu). Dále pak na obrázku 4.6 je vidět v grafu průběh hodnoty ztrátové funkce při učení takovéto sítě.

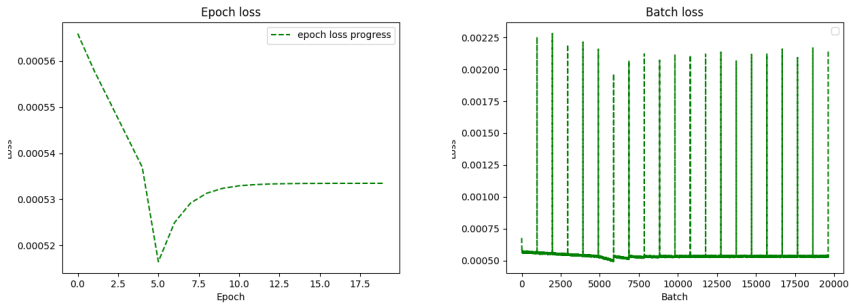
Všechny potřebné skripty pro replikaci výsledků a trénování této sítě najdeme ve složce *neural_net/segments*. Například skriptem *train.py* se spustí trénování sítě nebo skriptem *run.py* se spustí vizualizace výstupu sítě. Adresář dále obsahuje například skript *dataset.py*, ve kterém jsou nadefinované různé formy datasetu určené pro různorodé použití.

4.2.2 *faster R-CNN*

Po ne příliš úspěšných prvních pokusech jsme se rozhodli začít práce na druhém přístupu, kterým je úprava detekční architektury. K úpravě jsme zvolili síť *faster R-CNN*, jelikož implementace této sítě v *PyTorch* není příliš obtížná.



Obrázek 4.5: Výstup sítě s rozřezáním na *8x4* segmenty



(a) : Průměrná hodnota ztrátové funkce **(b) :** Detailní hodnoty ztrátové funkce

Obrázek 4.6: Průběh hodnoty ztrátové funkce pro rozřezáním na *8x4* segmenty

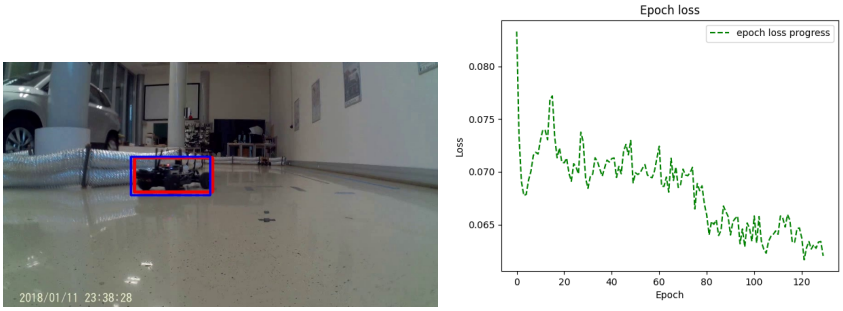
Zdrojové soubory k naší úpravě sítě jsou v adresáři *neural_net/detection*, kde základní architektuře dáváme jako parametr počet tříd objektů, v našem případě dva, jelikož síť rozlišuje i pozadí. Z počátku jsme se potýkali se vstupními rozměry snímku, jelikož síť je možno nastavit maximální a minimální velikost rozměrů, v našem případě jsme k této možnosti nepřistupovali. Místo toho jsme přeškálovali vstupní snímek na poloviční velikost, která už byla modelem sítě akceptována bez obtíží. Vstupní obrázek má tedy velikost *960x540* pixelů.

U této sítě jsme experimentovali s páteří (backbone) sítě, kterou jsme různě zaměňovali za různé předučené sítě a porovnávali jejich úspěšnost a rychlost.

faster R-CNN + Resnet50

Tato kombinace je ve frameworku *PyTorch* jednou ze základních verzí sítě faster R-CNN. Začali jsme tedy síť přeučovat na detekci pouze našeho modelu auta. Po přibližně 12 průchodech skrz dataset jsme měli první výsledky, které byly vcelku uspokojivé, ale stále občas síť označovala i různé prvky z pozadí. Po 26 průchodech byly výsledky daleko lepší a vypadaly jako na obrázku 4.7a (modře označení člověkem, červeně sítí). Síť jsme dále nechali učit do 130

průchodů, ale výrazné změny jsme nezaznamenali. Graf ztrátového kritéria je vidět na obrázku 4.7b. Detailnější grafy z učení, kde jsou vidět jednotlivá kritéria zvlášť, jsou na obrázku 4.11 na konci sekce.

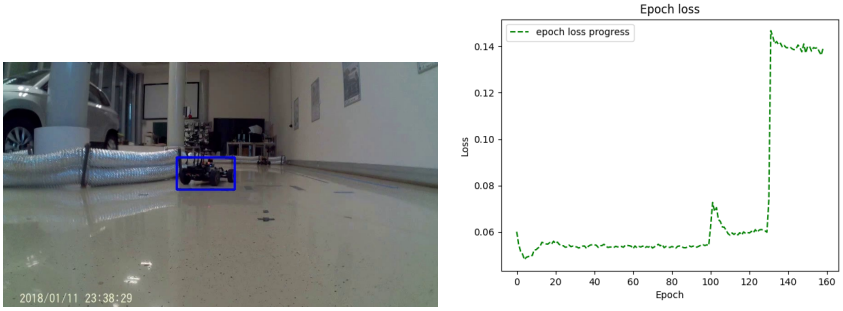


(a) : Vizualizace výstupu sítě **(b)** : Vývoj hodnoty ztrátové funkce

Obrázek 4.7: *faster R-CNN + Resnet50*

faster R-CNN + Mobilenet v3

Jelikož se předchozí řešení jevilo jako pomalé, vyzkoušeli jsme změnu předučené páteře sítě za *Mobilenet v3*, který je designován pro nízkovýkonové aplikace. Jak bylo očekáváno, síť počítala rychleji, bohužel se u ní ale vyskytuje mnoho falešných detekcí, auto nezaregistruje vůbec nebo je bounding box kolem auta nepřesný jako na obrázku 4.8a. Síť jsme stejně jako předchozí variantu učili nejprve 15 epizod a výsledky byly na počátek učení podle očekávání. Po dalších průchodech se ale už výsledky moc nezlepšily a proto jsme učení po 159 epizodách zastavili. Průběh učení je vidět v obrázku 4.8b na grafu. Detailnější grafy z učení, kde jsou vidět jednotlivá kritéria zvlášť, jsou na obrázku 4.12 na konci sekce.



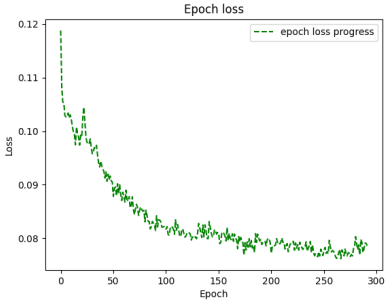
(a) : Vizualizace výstupu sítě **(b)** : Vývoj hodnoty ztrátové funkce

Obrázek 4.8: *faster R-CNN + Mobilenet v3*

faster R-CNN + Squeezenet 1.0

K další kombinaci jsme přistoupili z důvodů nespokojenosti s rychlostí kombinace se sítí *Resnet* a nepřesností kombinace se sítí *Mobilenet v3*. Další síť designovaná pro zařízení s nižším výkonem je *Squeezenet 1.0*, použili jsme tedy této sítě. Síť již od počátku byla výrazně rychlejší než varianta s*Resnetem* a dosahovala podobné přesnosti. První výsledky byly hned po 26 epizodách a nakonec jsme trénink zastavili po 292 průchodech. Výstup je vidět na obrázku 4.9a a graf z průběhu učení na obrázku 4.9b. Detailnější grafy z učení, kde jsou vidět jednotlivá kritéria zvlášť, jsou na obrázku 4.13 na konci sekce.

(a) : Vizualizace výstupu sítě



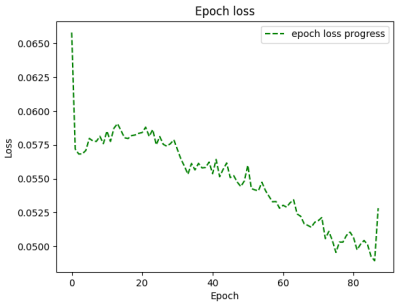
(b) : Vývoj hodnoty ztrátové funkce

Obrázek 4.9: *faster R-CNN + Squeezenet 1.0*

faster R-CNN + VGG-16

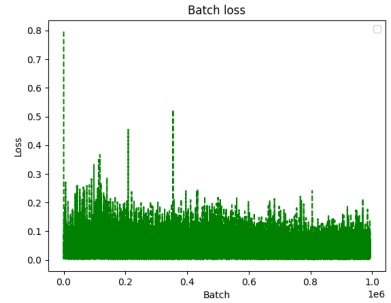
Jelikož ani předchozí verze se sítí *Squeezenet 1.0* nebyla dostatečně rychlá k použití na real-time řízení modelu. Vyzkoušeli jsme proto jako základ sítě ještě i síť VGG-16. U té ale rychlost zklamala podobně jako u sítě *Resnet*, stejně tak to bylo i s přesností predikce, ta byla uspokojivá jako u sítě s *Resnetem* nebo *Squeezenetem*. Kvůli neuspokojivé rychlosti jsme učení přerušili po 88 průbězích. Graf učení je na obrázku 4.10b a výstup sítě na obrázku 4.10a. Detailnější grafy z učení, kde jsou vidět jednotlivá kritéria zvlášť, jsou na obrázku 4.14 na konci sekce.

(a) : Vizualizace výstupu sítě

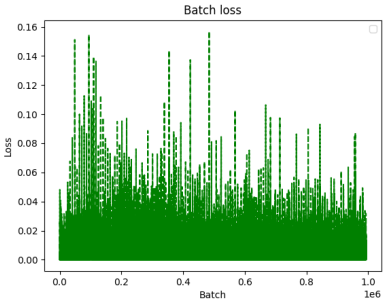


(b) : Vývoj hodnoty ztrátové funkce

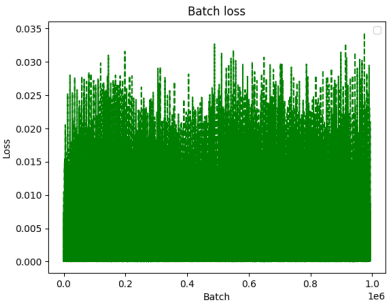
Obrázek 4.10: *faster R-CNN + VGG-16*



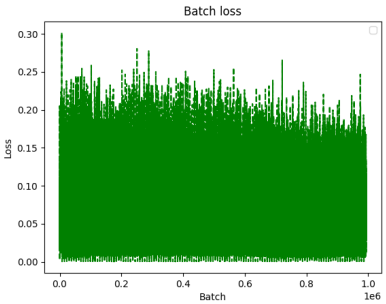
(a) : Classification



(b) : Objectness

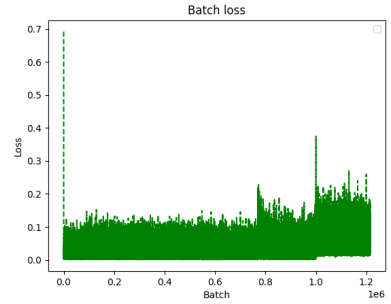


(c) : RPN box regresion

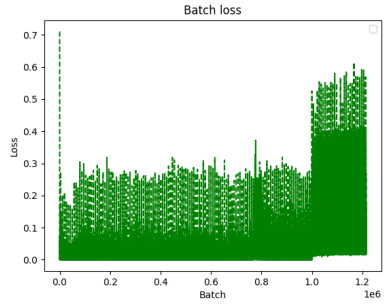


(d) : Bounding box regresion

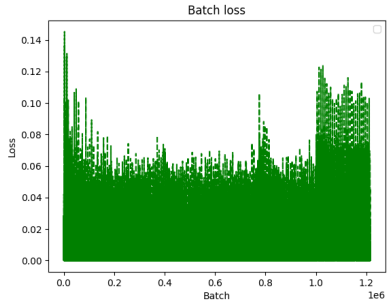
Obrázek 4.11: *faster R-CNN + Resnet50*



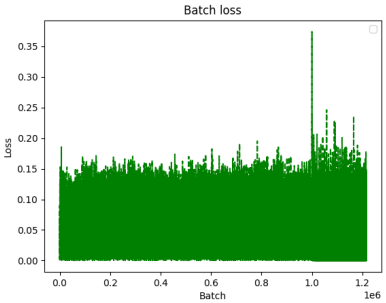
(a) : Classification



(b) : Objectness

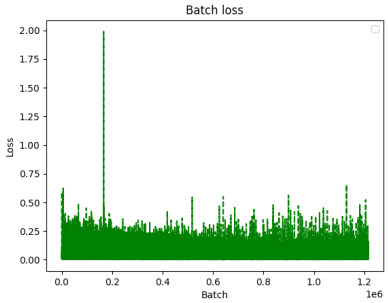


(c) : RPN box regresion

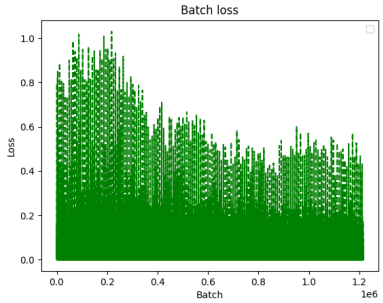


(d) : Bounding box regresion

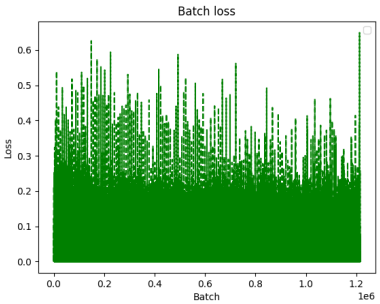
Obrázek 4.12: *faster R-CNN + Mobilenet v3*



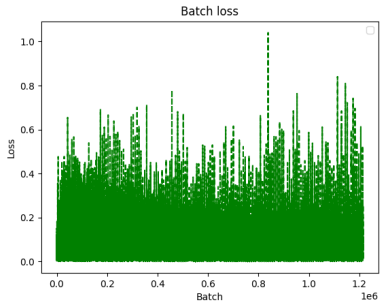
(a) : Classification



(b) : Objectness

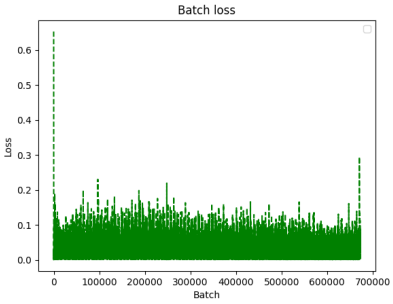


(c) : RPN box regresion

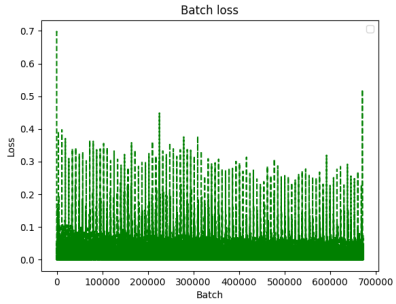


(d) : Bounding box regresion

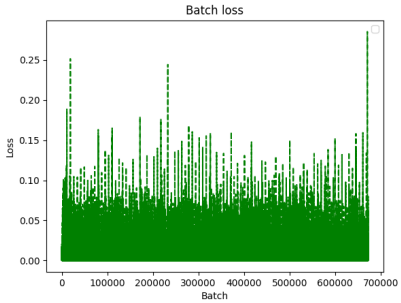
Obrázek 4.13: *faster R-CNN + Squeezenet 1.0*



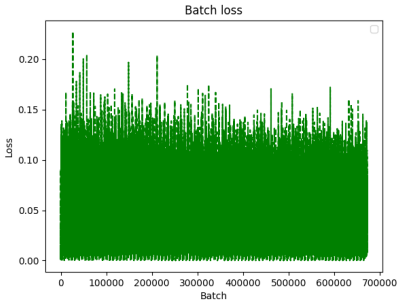
(a) : Classification



(b) : Objectness



(c) : RPN box regresion



(d) : Bounding box regresion

Obrázek 4.14: *faster R-CNN + VGG-16*

4.3 Výsledky vybraných řešení

Výsledky jednotlivých variant byly testovány na stejném zařízení s GPU pracujícím na frekvenci 1.5 *GHz* a při užití stejných validačních dat.

4.3.1 Výsledky rozdělení na segmenty

V tabulkách 4.3 a 4.4 najdeme přehledně výsledky jednotlivých kombinací. Zde nás zajímá převážně rychlost, jelikož přesnost byla velmi podobná u všech kombinací.

FPS $[\frac{1}{s}]$	Nejbližší soused	Plně propojená	Konvoluční
Resnet	0.5	4	4
Mobilenet	-	3	6
Squeezenet	-	6	3

Tabulka 4.3: Tabulka výsledků při rozdělení na segmenty

Precision [%]	Nejbližší soused	Plně propojená	Konvoluční
Resnet	50	55	60
Mobilenet	-	45	51
Squeezenet	-	58	51

Tabulka 4.4: Tabulka výsledků při rozdělení na segmenty

4.3.2 Výsledky s architekturou *faster R-CNN*

Přesnost a rychlost zkoumaných kombinací je shrnutá v tabulkách 4.5, 4.6. Detailně jsou potom výsledky na obrázcích 4.15, 4.16, 4.17 a 4.18

Varianta	FPS $[\frac{1}{s}]$
Resnet50	1.3
Mobilenet v3	4.1
Squeezenet 1.0	5.5
VGG-16	1.2

Tabulka 4.5: Tabulka výsledků s architekturou *faster R-CNN* pro rychlost

Varianta	Precision [%]
Resnet50	95.8
Mobilenet v3	4.9
Squeezenet 1.0	89.5
VGG-16	96.8

Tabulka 4.6: Tabulka výsledků s architekturou *faster R-CNN* pro přesnost

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.502
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.958
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.476
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.505
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.518
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.584
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.604
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.604
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.605
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.602
```

Obrázek 4.15: Celkové výsledky *faster R-CNN + Resnet50*

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.023
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.049
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.014
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.037
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.010
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.024
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.034
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.034
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.051
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.012
```

Obrázek 4.16: Celkové výsledky *faster R-CNN + Mobilenet v3*

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.450
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.895
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.335
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.460
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.495
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.524
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.561
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.561
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.564
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.557
```

Obrázek 4.17: Celkové výsledky *faster R-CNN + Squeezenet 1.0*

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.577
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.968
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.612
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.535
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.628
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.630
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.632
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.632
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.596
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.679
```

Obrázek 4.18: Celkové výsledky *faster R-CNN + VGG-16*

Kapitola 5

Návrhy na budoucí vylepšení

Během psaní této zprávy se vyskytly nové myšlenky a nápady k vylepšení řešení, které nebyly zatím otestovány, ale jistě je uvedeme do provozu a otestujeme v budoucnu.

5.1 Nástroj pro poloautomatické generování anotací

V průběhu práce vyšlo najevo, že by bylo dobré pro budoucí vývoj datasetu připravit nástroj, který urychlí práci při anotování, jelikož vytvořit anotace pro aktuální dataset trvalo celkově přibližně 20 hodin práce, což není zrovna málo času, který by mohl být věnován jiné odbornější práci. Tento nástroj by měl zjednodušit ruční označování auta na snímku, pomocí nějakého sledovacího nebo detekčního algoritmu. Při vyhodnocování výsledků a konzultacích se vyskytly dvě možné varianty. První variantou je vyžití neuronové sítě z této práce k vytvoření anotace, která bude následně pouze potvrzena člověkem, že je správná nebo bude případně opravena. Druhou možností je využití vylepšeného *KCF trackeru*, který je v projektu našeho auta F1/10 již připraven. Označování by probíhalo takovým způsobem, že by se na začátku označil model auta, který by byl následně sledován v obraze algoritmem *KCF trackeru*, poté co by se auto ztratilo či by obdélník kolem auta nebyl přesný, algoritmus by se manuálně zastavil, došlo by k ruční opravě a k opětovnému spuštění algoritmu.

Tento nástroj se tedy zatím nepodařilo připravit, jelikož to nebylo vyloženě nutné, do budoucna toto vylepšení bude přínosné.

5.2 Nápady pro vylepšení neuronové sítě

Během vyhodnocení vyšlo najevo, že otestované varianty jsou pomalé pro použití pro real-time řízení. Začali jsme proto uvažovat o budoucí implementaci některé ze sítí *Yolo* nebo *DPM*, ke kterým jsme nepřistoupili z důvodů složitější implementace a pravděpodobného nedostatku výkonu a paměti na našem modelu.

Další možnou variantou využití neuronové sítě k detekci auta soupeře by mohla být, pokud upustíme od využití kamery, detekce z mračna bodů za využití *LIDARu*, případně se nabízí i varianta kombinace obojího.

Kapitola 6

Závěr

V této práci jsme se zaměřili na nově vzniklou kategorii v soutěži F1/10, kterou je kategorie závodu mezi dvěma auty, kde musí auta jednotlivě prokázat schopnost detekovat protivníka a aktivně se mu vyhnout. Tato práce se zabývá pouze úlohou detekce soupeřova auta v obraze z kamery za pomoci neuronové sítě.

Velká část práce spočívala ve vytvoření kvalitního datasetu, který musel obsahovat různorodý obsah. Při jeho kompletování se vyskytly různé menší chyby, kvůli kterým vznikly další podpůrné nástroje mimo ty základní, jako je například nástroj pro vizualizaci nebo generování datasetu.

Z počátku jsme se snažili využít princip rozdělení snímku z kamery na části, který je využíván většinou sítí určených k detekci. Oproti těmto sítím jsme zvolili jiný systém rozdělení. Bohužel se ale tento navržený koncept v praxi neukázal dostatečně přesný. Jako pomyslnou druhou fázi považujeme přechod k využití přeučené detekční architektury *faster R-CNN*, u které jsme měnili její části a zkoumali vliv na rychlost a přesnost sítě.

Při testech jsme narazili na ne příliš vysokou přesnost při použití prvního konceptu, která se pohybovala kolem 55%. Naopak u architektury *faster R-CNN* se nám povedlo dosáhnout vyhovující přesnosti kolem 90% a při záměně základu sítě i celkem přijatelné rychlosti pro budoucí testování přímo na autě. Jako nejlepší výslednou kombinaci, kterou jsme testovali, bychom vyhodnotili architekturu *faster R-CNN* se základem ze sítě *Squeezenet 1.0*, který dosahoval přesnosti 89% při snímkové frekvenci 5.5 FPS.

Bibliografie

[1] Nikolas Adaloglou. *Intuitive Explanation of Skip Connections in Deep Learning*. 2020. URL: https://theaisummer.com/skip-connections/ (cit. 24. 04. 2021).

[2] Charu C. Aggarwal. *Neural networks and deep learning. a textbook*. Cham: Springer, [2018]. ISBN: isbn978-3-319-94462-3.

[3] Shun ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4-5 (1993), s. 185–196. ISSN: 09252312. DOI: 10.1016/0925-2312(93)90006-O. URL: https://linkinghub.elsevier.com/retrieve/pii/092523129390006O (cit. 13. 05. 2021).

[4] *Bronz ze soutěže F1/10 v New Yorku*. 2019. URL: https://fel.cvut.cz/cz/aktuality/2019/foto-cvut-ciirc-soutez-3.jpg (cit. 10. 05. 2021).

[5] Jason Brownlee. *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*. 2018. URL: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ (cit. 10. 05. 2021).

[6] *Convolutional Neural Network to identify the image of a bird*. 2009-2021. URL: https://www.simplilearn.com/ice9/free_resources_article_thumb/Convolutional_Neural_Network_to_identify_the_image_of_a_bird.png (cit. 19. 05. 2021).

[7] *Examples of MNIST digits*. 2008-2021. URL: https://www.researchgate.net/profile/Ulrich-Aivodji/publication/336084237/figure/fig3/AS:807584806211585@1569554630386/Examples-of-MNIST-digits.jpg (cit. 10. 05. 2021).

[8] *Fully connected neural network*. 2005–2021. URL: https://radiopaedia.org/articles/fully-connected-neural-network (cit. 23. 04. 2021).

[9] Trevor Hastie, Robert Tibshirani a Jerome Friedman. *The Elements of Statistical Learning. Data mining, Inference, and Prediction*. 2nd ed. New York: Springer, 2009. ISBN: isbn978-0-387-84857-0.

[10] Simon S. Haykin. *Neural networks. a comprehensive foundation*. 2nd ed. New York: Maxwell Macmillan International, 2008. ISBN: isbn0023527617.

[11] *Historie neuronových počítačů a sítí*. URL: https://www.fi.muni.cz/usr/jkucera/pv109/2000/xneudert.html (cit. 20. 04. 2021).

[12] *How to do Semantic Segmentation using Deep learning*. URL: https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef (cit. 23. 04. 2021).

[13] *How to do Semantic Segmentation using Deep learning*. URL: https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef (cit. 23. 04. 2021).

[14] Sergey Ioffe a Christian Szegedy. "Batch Normalization. Accelerating Deep Network Training b y Reducing Internal Covariate Shift". In: (2015), s. 1–11. URL: https://arxiv.org/pdf/1502.03167.pdf (cit. 23. 04. 2021).

[15] Helena KUČEROVÁ. *Neuronová síť*. 2003. URL: https://aleph.nkp.cz/F/?func=direct\&doc_number=000000120\&local_base=KTD (cit. 30. 03. 2021).

[16] James Le. *An example of semantic segmentation*. 2018. URL: https://miro.medium.com/max/3200/1*MQCvfEbbA44fiZk5GoDvhA.png (cit. 10. 05. 2021).

[17] Hao Li et al. "Visualizing the Loss Landscape of Neural Nets". In: (), s. 1. URL: https://arxiv.org/pdf/1712.09913.pdf (cit. 10. 05. 2021).

[18] *Max-pooling / Pooling*. 2001-2021. URL: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling (cit. 19. 05. 2021).

[19] Warren S. McCulloch a Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), 115–133.

[20] *MÁME BRONZ ZE SOUTĚŽE AUTONOMNÍCH FORMULÍ V NEW YORKU*. 2019. URL: https://aktualne.cvut.cz/sites/aktualne/files/styles/large/public/content/23e04b71-520d-405b-be61-843138c8b735/8a4dce0f-fbef-40f4-a3e7-51fd18177c4d.jpg?itok=EO2Kmpdt (cit. 10. 05. 2021).

[21] Joseph O'Rourke. "Finding minimal enclosing boxes". In: 14.3 (1985), s. 183–199. ISSN: 0091-7036. DOI: 10.1007/BF00991005. URL: http://link.springer.com/10.1007/BF00991005.

[22] Prajit Ramachandran, Barret Zoph a Quoc V. Le. "SEARCHING FOR ACTIVATION FUNCTIONS". In: (). URL: https://arxiv.org/pdf/1710.05941.pdf (cit. 13. 05. 2021).

[23] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychological review* 65.6 (1958), s. 386.

[24] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (2017). URL: https://arxiv.org/pdf/1609.04747.pdf (cit. 13. 05. 2021).

[25] *Rules for the F1/10 Autonomous Racing Competitions*. 2019. URL: https://f1tenth.org/race/rules-v2.pdf (cit. 20. 04. 2021).

[26] Shibani Santurkar et al. "How Does Batch Normalization Help Optimization?" In: (2019), s. 1–26. URL: https://arxiv.org/pdf/1805.11604.pdf (cit. 23. 04. 2021).

[27] *"SGD" vs "SGD + momentum" in 2D*. URL: https://cw.fel.cvut.cz/b201/_media/courses/b3b33vir/lectures/training.pdf (cit. 19. 05. 2021).

[28] Jost Tobias Springenberg et al. "STRIVING FOR SIMPLICITY. THE ALL CONVOLUTIONAL NET". In: (2015), s. 1–14. URL: https://arxiv.org/pdf/1412.6806.pdf (cit. 23. 04. 2021).

[29] *What Is a Dataset in Machine Learning: Sources, Features, Analysis*. 2021. URL: https://labelyourdata.com/articles/what-is-dataset-in-machine-learning/ (cit. 19. 05. 2021).

[30] Thomas Zeugmann et al. "Precision and Recall". In: *Encyclopedia of Machine Learning* (2010), s. 781–781. DOI: 10.1007/978-0-387-30164-8_652. URL: http://link.springer.com/10.1007/978-0-387-30164-8_652 (cit. 13. 05. 2021).

Příloha A

Obsah příloh

K práci je přiloženo DVD, na kterém jsou všechny zdrojové soubory. Podobně nalezneme zdrojové kódy na tomto odkaze `https://gitlab.com/feniix.kotm/ctu-f1-10-detection_dataset` a zdrojové soubory datasetu na odkaze `https://drive.google.com/drive/folders/1lLdGZhFHdruUmDHxT2PjKrXvPXAJbW93?usp=sharing`.

V kořenové složce nalezneme 2 dvě hlavní složky: *dataset* a *neural_net*.

Ve složce *dataset* nalezneme složku *dataset_tools*, ve které jsou nástroje pro generování a správu datasetu, které byly popsány v kapitole 4, a složku *dataset_source*, ve které jsou uložené zdrojové soubory datasetu.

Ve složce *neural_net* jsou ve dvou složkách zdrojové soubory jednotlivých konceptů, kdy soubor se jménem *train.py* slouží ke spuštění učení sítě a soubor se jménem *run.py* spustí vizualizaci nebo vyhodnocení výsledků sítě. Obě dvě složky obsahují složku *model* s jednotlivými natrénovanými sítěmi.

Pro replikaci výsledků postačí využít natrénovanou síť nebo je možno trénovat znovu.