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

ROZPOZNÁNÍ HRANIC JÍZDNÍHO PRUHU V ZÁBĚRECH PALUBNÍ KAMERY

RECOGNITION OF DRIVING LANE BORDERS IN VIDEO FROM ON BOARD CAMERA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LUKÁŠ LETOVANEC**
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: **Letovanec Lukáš**
Program: Informační technologie
Název: **Rozpoznání hranic jízdního pruhu v záběrech palubní kamery**
Recognition of Driving Lane Borders in Video from On Board Camera
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou počtíačového vidění pomocí hlubokých konvolučních neuronových sítí; zaměřte se na problematiku detekce v obraze a videu.
2. Vyhledejte a popište existující přístupy k rozpoznání hranic jízdního pruhu v obraze a videu.
3. Získejte a/nebo pořiďte vhodnou datovou sadu pro učení a vyhodnocování řešených algoritmů.
4. Vyberte vhodný algoritmus či algoritmy a implementujte je.
5. Experimentujte s vytvořeným řešením na datech, laďte algoritmus a získejte podrobné znalosti o jeho chování.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Aharon Bar Hillel, Ronen Lerner, Dan Levi, Guy Raz: Recent progress in road and lane detection: a survey, Machine Vision and Applications, 25/3, 2014
- Sibel Yenikaya, Gökhan Yenikaya, Ekrem Düven: Keeping the vehicle on the road: A survey on on-road lane detection systems, ACM Computing Surveys (CSUR), 46/1, 2013
- Jun Li, Xue Mei, Danil Prokhorov, Dacheng Tao: Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene, IEEE Tran. on Neural Networks and L.S., 28/3, 2017
- Brody Huval et al.: An Empirical Evaluation of Deep Learning on Highway Driving, arXiv:1504.01716

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. května 2019

Abstrakt

Táto práca sa zaoberá problematikou rozpoznávania hraníc jazdného pruhu v záberoch palubnej kamery. V práci je predstavená architektúra hlbokej konvolučnej neurónovej siete, pomocou ktorej sa rieši spomínaný problém. Sieť bola trénovaná na rozsiahlej dátovej sade pomocou algoritmu gradientového zostupu. Natrénovaný model preukázal schopnosť kvalitne rozpoznávať hranice jazdného pruhu v rôznych situáciách a podmienkach. Výsledok práce potvrdzuje, že hlboké konvolučné neurónové siete sú vhodným nástrojom pre rozpoznávanie hraníc jazdného pruhu.

Abstract

This thesis is dedicated to the issue of driving lane borders recognition in frames of an onboard camera. In this thesis, an architecture of a deep convolutional neural network is introduced, by means of which the said problem is dealt with. The net was trained on a large dataset using a gradient descent algorithm. The trained model has demonstrated the ability to recognize borders of a driving lane well in different situations and conditions. The result of the thesis confirms that deep convolutional neural networks are a suitable tool for driving lane borders recognition.

Kľúčové slová

hlboké konvolučné neurónové siete, gradientový zostup, rozpoznávanie hraníc jazdného pruhu, trénovanie pod dohľadom, počítačové videnie

Keywords

deep convolutional neural networks, gradient descent, driving lane borders recognition, supervised learning, computer vision

Citácia

LETOVANEC, Lukáš. *Rozpoznání hranic jízdního pruhu v záběrech palubní kamery.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Rozpoznání hranic jízdního pruhu v záběrech palubní kamery

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána prof. Ing. Adama Herouta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Lukáš Letovanec
15. mája 2019

Poďakovanie

Ďakujem prof. Ing. Adamovi Heroutovi, Ph.D. za usmerňovanie v priebehu výskumu a cenné rady, ktoré mi pomohli pri vypracovaní tejto bakalárskej práce. Rovnako ďakujem aj kamarátom, priateľke a rodine za pomoc a podporu, ktorú mi poskytovali počas jej písania.

Obsah

Kapitola 1

Úvod

V súčasnosti žijeme v hektickej dobe, v ktorej ľudia každý deň potrebujú cestovať veľké vzdialenosti či už za prácou, rodinou alebo priateľmi. Vývoj v oblasti technológií je touto potrebou ovplyvňovaný a sú vyvíjané stále nové systémy za účelom zvýšenia komfortu a bezpečnosti pri cestovaní. Pre automobily sú to napríklad parkovacie systémy, asistenti riadenia alebo dokonca celkový autopilot. Všetky tieto technológie nejakým spôsobom vnímajú prostredie, v ktorom sa vozidlo nachádza, vyhodnocujú ho a prípadne vykonajú určitú akciu. Rozpoznávanie hraníc jazdného pruhu vozidla v záberoch palubnej kamery sa čoraz častejšie stáva súčasťou práve takýchto moderných systémov.

K problému rozpoznávania hraníc jazdného pruhu existuje viacero prístupov, pričom všetky sú založené na rôznych technikách počítačového videnia. Vďaka náhlemu rozmachu hlbokých konvolučných neurónových sietí v posledných rokoch a ich využiteľnosti v oblasti rozpoznávania obrazu, každým dňom pribúdajú nové inovatívne algoritmy. Cieľom tejto práce je zoznámiť sa práve s hlbokými konvolučnými neurónovými sieťami a s ich prístupom k problémom v oblasti počítačového videnia, natrénovať takúto sieť na vhodných dátach pre rozpoznávanie hraníc jazdného pruhu a získať poznatky o jej chovaní v rôznych situáciách.

O úvode do problematiky hlbokých konvolučných neurónových sietí a o jednotlivých vrstvách, ktoré sa v nich používajú sa dočítate v kapitole 2. V kapitole 3 sú spomenuté algoritmy spojené s trénovaním takejto siete. Kapitola 4 obsahuje informácie o dátovej sade použitej pri trénovaní siete pre rozpoznávanie hraníc jazdného pruhu. V kapitole 5 je stručne opísaný jeden z existujúcich prístupov k riešeniu daného problému spolu s mojím návrhom riešenia a predstavením navrhnutej architektúry siete. Kapitola 6 opisuje nástroje použité pri implementácii riešenia ako aj samotnú implementáciu jednotlivých častí. Testovanie chovania natrénovanej siete nájdete v kapitole 7 a v závere budú zhodnotené výsledky spolu s návrhmi pre pokračovanie v projekte.

Kapitola 2

Hlboké konvolučné neurónové siete

V prvých odstavcoch tejto kapitoly sa nachádzajú všeobecné informácie o neurónových sieťach ako takých a ďalšie odstavce sú už venované konkrétne hlbokým konvolučným neurónovým sieťam. Spomenuté sú tu aj jednotlivé vrstvy, ktoré sa používajú v týchto sieťach a ich význam pri spracovaní obrazových dát.

2.1 Úvod do problematiky

Klasická neurónová sieť [29] pozostáva z množstva jednoduchých, poprepájaných jednotiek zvaných neuróny. Neurón je matematický model biologického neurónu, ktorý prijíma množinu vstupov a váh, a produkuje jeden výstup pomocou aktivačnej funkcie. Neurónová sieť ďalej môže spadať do kategórie rekurentných (cyklických) alebo acyklických (v angličtine často označovaných ako *feedforward*) neurónových sietí. Pre natrénovanie modelu som použil architektúru siete, ktorá spadá do kategórie acyklických neurónových sietí a preto sa rekurentnými sieťami v tejto práci zaoberať nebudem. V acyklických neurónových sieťach, ako už z ich názvu vyplýva, žiadne prepojenie medzi jej neurónmi nevytvára cyklus.

Zoberme si acyklickú sieť [8], ktorej úlohou je aproximovať funkciu f^* napríklad pre klasifikátor s predpisom $y = f^*(x)$. Tento klasifikátor mapuje vstup x na kategóriu y. Acyklická sieť definuje mapovanie $y = f(x; w)$ a pri trénovaní sa naučí hodnoty parametrov w tak, aby aproximácia funkcie bola čo najpresnejšia. Takáto sieť sa nazýva acyklickou preto, lebo informácie prechádzajú cez funkciu, ktorá bola vyhodnotená z x, cez medziľahlé výpočty, ktoré definujú funkciu f a nakoniec na výstup y. Nenachádzajú sa tu teda žiadne spätné prepojenia, v ktorých je výstup z modelu znova použitý ako vstup.

Neurónová sieť, v ktorej sa medzi vstupom a výstupom nachádza viac ako jedna skrytá vrstva, sa nazýva hlboká neurónová sieť [12]. Opakom tejto siete je sieť plytká (pozostáva z jednej skrytej vrstvy), ale tomuto typu siete sa v práci venovať nebudem.

Na vysvetlenie pojmu *skrytá vrstva* je možné využiť príklad z predošlého paragrafu. Pri trénovaní siete [8] má každá trénovacia vzorka x trénovacej dátovej sady k sebe priradený label[1] y kde $y \approx f^*(x)$. Trénovacie vzorky teda priamo špecifikujú, čo má byť výstupom siete – pre vstup x to má byť hodnota blízka hodnote y. Správanie ostatných vrstiev (všetkých medzi vstupnou vrstvou a výstupnou vrstvou) nie je priamo špecifikované trénovacími vzorkami. Algoritmus učenia teda musí rozhodnúť, ako tieto vrstvy použiť pre čo najlepšiu implementáciu aproximácie f^*. Tieto vrstvy sa nazývajú skryté, pretože trénovacie dáta neobsahujú očakávané výstupy pre žiadnu z medziľahlých vrstiev.

[1] Label – očakávaný výstup zo siete

Konvolučná neurónová sieť [8] (často označovaná ako **CNN**) je špecializovaným druhom neurónovej siete zameraným na spracovávanie dát, ktoré majú mriežkovanú topológiu. Ako príklady takýchto dát je možné uviesť časové rady (1-D), čiernobiele obrázky (2-D) alebo napríklad RGB obrázky (2-D s tromi kanálmi).

Už z názvu *konvolučná neurónová sieť* vyplýva, že sa v tomto type sietí využíva matematická operácia **konvolúcia**, o ktorej sa viacej dozviete v sekcii 2.2.1. Samotná konvolučná vrstva by pri spracovávaní vyššie spomenutých dát častokrát nestačila a preto sa architektúra siete zvykne doplňovať o vrstvy ako pooling (2.2.2), plne prepojené vrstvy či nelineárne (taktiež nazývané aktivačné) vrstvy (2.2.3).

Vhodným naskladaním týchto vrstiev za sebou teda vzniká hlboká konvolučná neurónová sieť. Keďže v tejto práci sa zaoberám problematikou detekcie objektov (konkrétne hraníc jazdného pruhu vozidla) v záberoch palubnej kamery, tento typ neurónovej siete vytvára vhodný nástroj pre realizáciu riešenia.

2.2 Vrstvy plne konvolučnej siete

Každú vrstvu plne konvolučnej neurónovej siete tvorí trojrozmerné pole veľkosti $h \times w \times d$, kde h a w sú priestorové rozmery (výška a šírka) a d určuje počet kanálov alebo čŕt. Napríklad v našom prípade bude vstupnou vrstvou siete RGB obrázok a teda $d = 3$. Takýto typ siete je postavený na tzv. prenosovej invariácii. To znamená, že vrstvy tejto siete operujú nad lokálnymi vstupnými regiónmi a závisia len na relatívnych priestorových súradniciach [20].

2.2.1 Konvolučné vrstvy

Konvolúcia

Konvolúcia je jednoduchá matematická operácia, ktorá je základom mnohých algoritmov na spracovanie obrazu. Na rozdiel od napríklad sčítania, ktoré berie ako vstup dve čísla a výstupom je ďalšie číslo, konvolúcia berie ako vstup dva signály a výstupom je ďalší signál. V našom prípade sú vstupnými signálmi časť obrázku a *filter* (používa sa aj označenie *kernel*), výstup sa často označuje ako *feature map* (voľne preložiteľné ako mapa čŕt). Viac o konvolúcii ako takej je uvedené napríklad v [8] v sekcii 9.1. V tejto práci sa jej budem venovať len z pohľadu využitia pri spracovaní obrazu.

Využitie konvolúcie pri spracovaní obrazu

Parametre konvolučnej vrstvy sa skladajú zo sady učenia sa schopných filtrov. Každý filter je malý z hľadiska šírky a výšky, ale rozpína sa cez všetky kanály vstupu. Napríklad filtre s rozmermi $5 \times 5 \times d$ alebo $3 \times 3 \times d$, kde d je počet kanálov vstupu, sú jednými z najrozšírenejších typov filtrov a používajú sa v mnohých architektúrach sietí, ktoré preukázali skvelé výsledky v rôznych úlohách spracovania obrazu. Hovoríme o architektúrach ako AlexNet [17], VGG16 [30], ResNet [11] a mnohých ďalších. V praxi nie je obvyklé používať filtre, ktorých priestorové rozmery majú iný ako štvorcovitý tvar.

Nasledujúce odstavce sú prevzaté z [1] a [15]. Počas procesu nazývaného *forward pass*[2] sa každý filter posúva naprieč šírkou a výškou vstupu a počíta skalárne súčiny medzi maticou

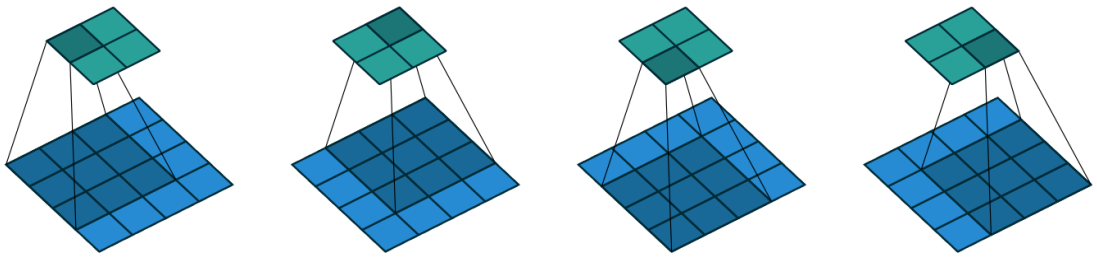
[2] Proces výpočtu výstupov vrstvy zo vstupov.

filtra a časťou vstupu, nad ktorou filter práve operuje, v každej možnej pozícii. Posúvanie filtra pri konvolúcii je znázornené na obrázku 2.1.

Ako sa filter posúva naprieč šírkou a výškou vstupu vrstvy, *konvolvuje*, produkuje sa dvojrozmerná aktivačná mapa, ktorá určuje reakciu na ten daný filter v každej priestorovej polohe. Sieť sa takto intuitívne naučí filtre, ktoré reagujú keď zaznamenajú nejaký typ vizuálnej črty, ako napríklad hranu určitej orientácie, škvrnu nejakej farby na prvej vrstve, alebo prípadne celé plástové vzory vo vyšších vrstvách siete a podobne. Keďže pre každý filter vznikne dvojdimenzionálna aktivačná mapa a filtrov v konvolučnej vrstve obvykle býva niekoľko, naskladaním týchto aktivačných máp za seba pozdĺž dimenzie hĺbky vznikne výstup vrstvy.

Ako presne ale bude výstup z vrstvy vyzerať vzhľadom k rozmerom vstupu a hyperparametrom[3] konvolučnej vrstvy? Konkrétne tieto tri hyperparametre ovplyvňujú rozmery výstupu z vrstvy:

- **hĺbka** je hyperparametrom, ktorý určuje počet kanálov výstupu a zodpovedá počtu použitých filtrov v konvolučnej vrstve,
- **krok** v konvolučnej vrstve určuje, po koľkých pixeloch sa filter postupne posúva. Keď $krok = 1$, potom sa filter posunie vždy o jeden pixel ako možno vidieť na obrázku 2.1. Keby $krok = 2$, filter sa posunie vždy o dva pixely, čo bude mať za následok menšie priestorové rozmery výstupu z vrstvy. Znázornenie konvolúcie s krokom 2 môžete vidieť na obrázku 2.2. Väčšie hodnoty pre krok sa v praxi používajú iba veľmi ojedinele,
- **výplň nulami** (*padding*) je v istých prípadoch veľmi dôležitým hyperparametrom. Umožňuje vytvoriť okraje okolo dimenzii vstupov vyplnené nulovými hodnotami. Táto vlastnosť nám umožňuje kontrolovať priestorové rozmery výstupu z vrstvy. Na obrázku 2.3 môžete vidieť, ako pridanie paddingu ovplyvňuje rozmery výstupu.

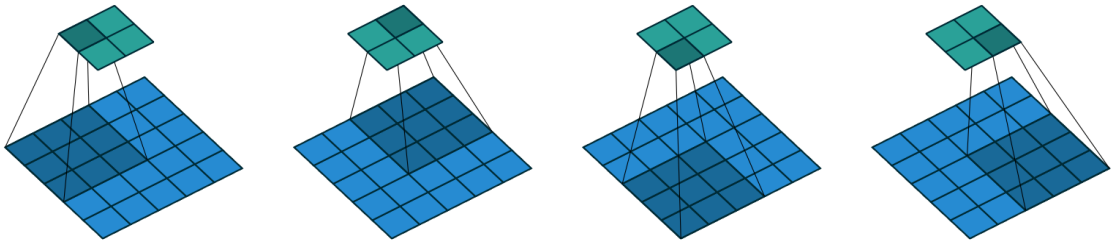


Obr. 2.1: Posúvanie 3×3 filtra cez 4×4 vstup. Modrou farbou je zobrazený vstup a tyrkysovou farbou výstup po konvolúcii. Obrázok je prevzatý z [7], strana 14.

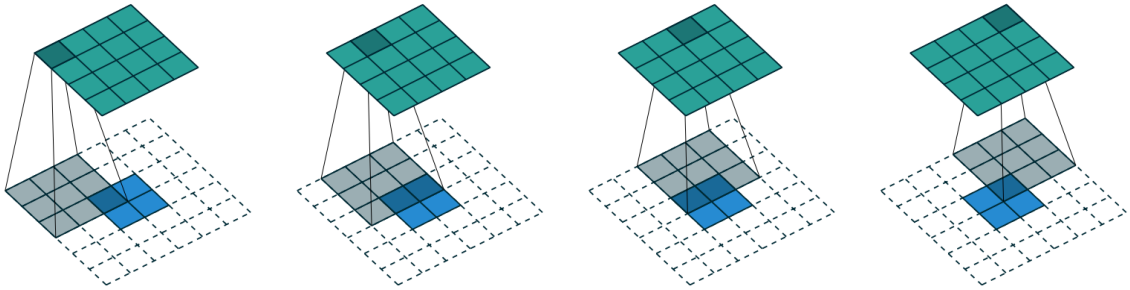
Pre vstup do vrstvy s rozmermi $\mathbf{W_1} \times \mathbf{H_1} \times \mathbf{D_1}$ vieme vypočítať výstup $\mathbf{W_2} \times \mathbf{H_2} \times \mathbf{D_2}$ rovnicami 2.1, 2.2 a 2.3

$$W_2 = (W_1 - F + P_{WL} + P_{WR})/S \ + 1 \tag{2.1}$$
$$H_2 = (H_1 - F + P_{HU} + P_{HD})/S \ + 1 \tag{2.2}$$
$$D_2 = K \tag{2.3}$$

[3]Hyperparameter – parameter, ktorého hodnota je nastavená pred začatím trénovania siete, a ktorého hodnota sa v priebehu trénovania nemení.



Obr. 2.2: Posúvanie 3×3 filtra cez 5×5 vstup s krokom 2. Modrou farbou je zobrazený vstup a tyrkysovou farbou výstup po konvolúcii. Obrázok je prevzatý z [7], strana 17.



Obr. 2.3: Prvé štyri kroky posúvania 3×3 filtra cez 2×2 vstup s krokom 1 a paddingom 2 vo všetkých smeroch. Modrou farbou je zobrazený vstup, tyrkysovou farbou výstup po konvolúcii a bielou farbou s prerušovanými hranami je označený padding. Obrázok je prevzatý z [7], strana 23.

kde F je veľkosť jedného z priestorových rozmerov filtra – počítame s tým, že priestorové rozmery filtra majú tvar štvorca a teda môžeme pre šírku aj výšku použiť jednu hodnotu F. S je veľkosť kroku pri posúvaní filtra[4], K určuje množstvo použitých filtrov a $P_{WL}, P_{WR}, P_{HU}, P_{HD}$ je počet vytvorených, nulami vyplnených okrajov v smere danom dolnými indexami týchto parametrov okolo priestorových rozmerov vstupu:

- P_{WL} určuje počet okrajov z ľavej strany,
- P_{WR} určuje počet okrajov z pravej strany,
- P_{HU} určuje počet okrajov zhora,
- P_{HD} určuje počet okrajov zdola.

Napríklad pre konvolúciu zobrazenú na obrázku 2.3 platí, že $P_{WL} = P_{WR} = P_{HU} = P_{HD} = 2$. [7]

Transponovaná konvolúcia a jej využitie

Existencia transponovaných konvolúcií vo všeobecnosti vyplýva z potreby mať nejaký mechanizmus, ktorý funguje v opačnom smere ako klasická konvolúcia. To znamená z niečoho, čo má tvar výstupu nejakej konvolúcie dostať tvar vstupu tejto konvolúcie. Napríklad v našom prípade používame transponovanú konvolúciu ako dekódovaciu vrstvu v dekóderovej

[4]Krok je v kontexte konvolučných vrstiev často označovaný ako *stride*.

časti modelu. Viac o architektúre modelu sa dozviete v sekcii 5.2.2. Vrstva transponovanej konvolúcie býva v moderných knižniciach pre strojové učenie implementovaná, takže sa s ňou viacej zaoberať nemusíme. Ak vás zaujíma, čo všetko stojí za transponovanou konvolúciou tak sa o tom môžete dočítať napríklad v [7], v sekcii 4.2.

2.2.2 Poolingové vrstvy

Nasledujúce odstavce sú prebraté z [7] a [36].

Pooling všeobecne

Popri konvolučných vrstvách, poolingové vrstvy (niekedy označované aj ako združovacie vrstvy) taktiež tvoria veľmi dôležitý stavebný blok v konvolučných neurónových sieťach. Poolingové vrstvy redukujú veľkosť mapy čŕt použitím určitej funkcie, ktorá sumarizuje jej podoblasti. Takýmito funkciami môžu byť napríklad priemer, maximum či suma. Z týchto funkcií sú odvodené aj názvy najviac rozšírených poolingových vrstiev – *average pooling*, *max pooling* a *sum pooling*.

Hlavným účelom poolingovej vrstvy [3] je transformovať reprezentáciu spoločnej črty do použiteľnejšej formy, pričom ostávajú zachované dôležité informácie a nepodstatné detaily sú zahodené. Použiteľnejšou formou myslíme mapu (alebo mapy) čŕt so zredukovanými rozmermi. Toto je docielené posúvaním okna (podobne ako pri konvolúcii a posúvaní filtra) cez vstup vrstvy a použitím už spomínanej *poolingovej funkcie* na obsah okna. V architektúre mojej siete bol použitý iba *max pooling* a preto sa budem v nasledujúcich odstavcoch venovať len tomuto typu.

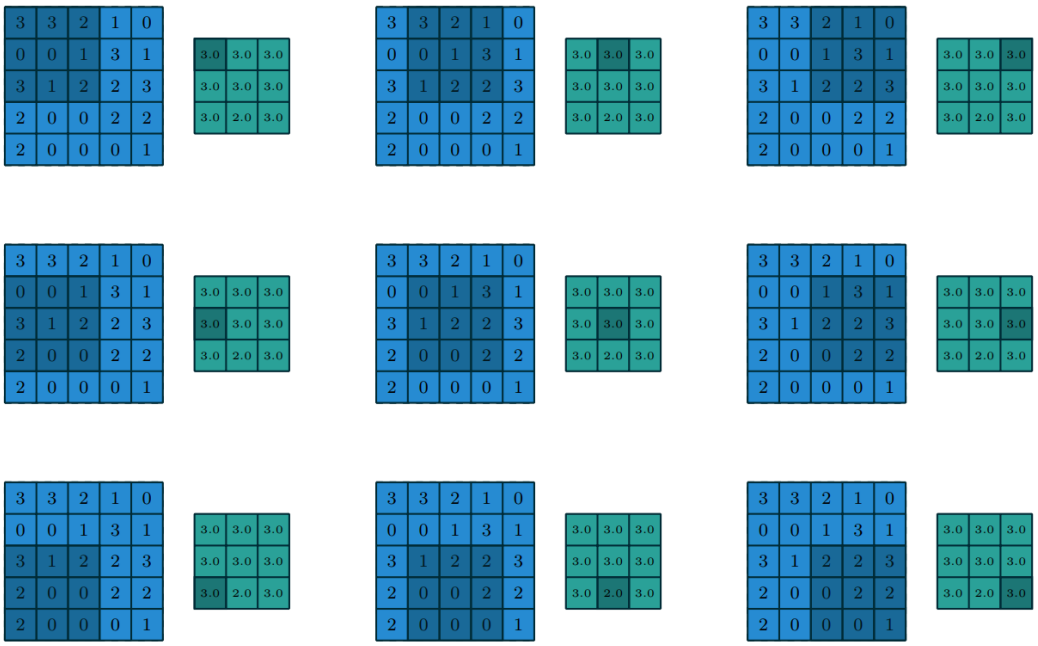
Max pooling

Max pooling je najbežnejším druhom poolingu, ktorý rozdeľuje vstup na menšie tzv. *regióny* alebo *okná* a vyberie z týchto regiónov prvok s najväčšou hodnotou. Tieto regióny sa nezvyknú prekrývať, ale nájde sa aj veľa prípadov s prekrývajúcimi sa regiónmi. Tento výber môže byť vyjadrený rovnicou 2.4 kde y_{kij} je výstup poolingového operátora spojeného s k-tou mapou čŕt, x_{kpq} je prvok na súradniciach (p, q) v rámci regiónu R_{ij}, ktorý reprezentuje okolie okolo pozície (i, j).

$$y_{kij} = \max_{(p,q)\in R_{ij}} x_{kpq} \tag{2.4}$$

Výstup z poolingovej vrstvy teda závisí od počtu máp čŕt na vstupe do vrstvy, rozmerov posuvného okna, nad ktorým prebieha pooling, od veľkosti kroku a od prípadného paddingu. Vo väčšine prípadov sa ale rozmery okna zadávajú tak, aby sa postupným posúvaním spracovali všetky prvky aj bez použitia paddingu. Pri použití nesprávnych rozmerov okna a nepoužití paddingu by sa tak pri tejto operácii mohli stratiť dôležité informácie, keďže okrajové prvky by z jednej strany neboli spracované. Proces spracovania jednej mapy čŕt pri max poolingu je vidieť vidieť na obrázku 2.4.

Z obrázkov je vidno, ako nápadne sa poolingová vrstva podobá na vrstvu konvolučnú. Celkové nahradenie max poolingu konvolučnými vrstvami s použitím väčšieho kroku bolo navrhnuté napríklad v [31]. Je teda možné, že v budúcich architektúrach konvolučných neurónových sietí sa s týmto prístupom budeme stretávať častejšie.



Obr. 2.4: Výpočet max poolingu s veľkosťou okna 3×3 nad 5×5 vstupom bez paddingu a s krokom 1. Modrou farbou je zobrazený vstup a tyrkysovou farbou výstup po max poolingu. Obrázok je prevzatý z [7], strana 11.

2.2.3 Aktivačné vrstvy

Aktivačná vrstva zvyčajne nasleduje hneď za vrstvou konvolučnou. Skladá sa z nelineárnej – *aktivačnej* mapovacej funkcie, ktorá aplikuje nelinearitu na každý prvok výstupu z konvolučnej vrstvy [3]. V našom prípade je týmto prvkom pixel. Keďže operácia prebieha nad jednotlivými prvkami, rozmery vstupu a výstupu tejto vrstvy sú identické. Prečo ale potrebujeme aplikovať nelinearitu? Bez nelinearity by bola jednoducho akokoľvek hlboká lineárna neurónová sieť po sčítaní jej vrstiev iba jednovrstvovou lineárnou sieťou [28]. Hoci môžu existovať funkcie, ktoré je možno dobre aproximovať i s využitím jednej vrstvy, v oblasti počítačového videnia nie sú takéto funkcie využitelné pre komplexnejšie problémy.

Typy aktivačných funkcií

Medzi najznámešie typy aktivačných funkcií v oblasti neurónových sietí patria *hyperbolický tangens*, *sigmoid* a *rektifikovaná lineárna jednotka* (ďalej len *ReLU*[5]). Je známe, že *sigmoid* a *hyperbolický tanges* stláčajú hodnoty celého definičného oboru do malého intervalu oboru hodnôt. Pri sigmoide je to interval $(0, 1)$ a pri hyperbolickom tangense $(-1, 1)$. Taktiež gradienty týchto funkcií sa pri veľkých vstupných hodnotách stávajú takmer nulovými. Tým pádom pri trénovacích metódach založených na gradientoch, viac v sekcii 3.2, sa aktualizácie váh pri trénovaní stávajú veľmi malými a vzniká problém nazývaný *problém miznúceho gradientu*, ktorý má za následok pomalé trénovanie siete. Viac o tomto probléme je uvedené

[5] ReLU – Rectified Linear Unit

napríklad v [9]. Riešiť tento problém pomáha *ReLU*, ktorá bola použitá aj v mojom modeli, a o ktorej je uvedené viac v nasledujúcich odstavcoch. [14]

ReLU

ReLU aktivačná funkcia je definovaná rovnicou 2.5 kde x je vstup do aktivačnej funkcie.

$$\text{ReLU}(x) = \begin{cases} 0 & \text{pre } x < 0 \\ x & \text{pre } x \geq 0 \end{cases} \tag{2.5}$$

Táto funkcia je ľahko optimatizovateľná, pretože je veľmi podobná lineárnej funkcii. Jediný rozdiel medzi lineárnou funkciou a ReLU je ten, že ReLU má na výstupe nulu na polovici jej definičného oboru. Gradienty v kladnej časti definičného oboru nadobúdajú konštantnú hodnotu 1, čo znamená, že smer gradientu je pre učenie siete oveľa užitočnejší ako pri aktivačných funkciách spomínaných v predošlom odstavci. Vyhýbame sa týmto aj problému miznúceho gradientu čím sa zvyšuje rýchlosť učenia hlbokých neurónových sietí.[29][14]

Existuje niekoľko modifikácií ReLU. Väčšina z týchto modifikácií má veľmi podobné výsledky ako klasická ReLU, ale príležitostne sú pri ich použití výsledky lepšie. Takýmito modifikáciami sú napríklad *leaky ReLU* [21], *parametrická ReLU* [10] alebo *exponenciálna LU* [5]. [29]

Kapitola 3

Algoritmy spojené s trénovaním modelu pod dohľadom

Nasledujúce odstavce vychádzajú z [19]. Najbežnejšou formou strojového učenia, či už hlbokého alebo nie, je tzv. *trénovanie pod dohľadom*[1]. Predstavme si, že chceme natrénovať model, ktorý bude schopný klasifikovať obrázky podľa toho, čo na nich je do určitých kategórii. Napríklad či sa na obrázku nachádza auto, človek, pes alebo dom. Najprv potrebujeme zhromaždiť veľkú dátovú sadu obrázkov áut, ľudí, psov a domov, každú anotovanú jej kategóriou (viac informácii o dátovej sade, ktorá bola použitá v tejto práci nájdete v sekcii 4). Pri trénovaní je modelu ukázaný obrázok a model vyprodukuje výstup vo forme vektoru pravdepodobností, jednu pravdepodobnosť pre každú kategóriu. Chceli by sme, aby mala správna kategória najväčšiu hodnotu pravdepodobnosti. To je ale bez trénovania modelu málo pravdepodobné. Pomocou objektívnej funkcie – *chybovej funkcie* (viac v sekcii 3.1.1), vypočítame chybu (alebo vzdialenosť) medzi výstupom zo siete a požadovaným výstupom. Model potom upraví svoje vnútorné regulovateľné parametre tak, aby sa táto chyba znížila. Tieto regulovateľné parametre sa často označujú aj ako *váhy*. V prípade konvolučných neurónových sietí sú tieto váhy jednotlivé prvky konvolučného filtra opísaného v 2.2.1. Viac o procese trénovania a o algoritmoch s ním spojenými sa dočítate v nasledujúcich podkapitolách.

3.1 Backpropagation

Backpropagation je momentálne najpopulárnejším algoritmom pre trénovanie pod dohľadom pri viacvrstvových *feedforward* neurónových sieťach. Ako už bolo spomenuté v úvode do kapitoly 3, pri trénovaní pod dohľadom sa snažíme neurónovú sieť natrénovať tak, aby sa jej výstup $\hat{Y}$ čo najviac priblížil k cieľovému výstupu Y pre trénovaciu sadu, ktorá obsahuje T vzoriek. Cieľom je prispôsobovať váhy v jednotlivých vrstvách siete tak, aby dobre fungovala aj pre vzorky mimo trénovacej sady. Táto vlastnosť modelu sa nazýva *generalizácia*. Problém, kedy má sieť dobré výsledky pre dáta z trénovacej sady, ale zlé pre dáta mimo tejto sady sa nazýva **pretrénovanie siete**[2]. Predísť tomuto problému pomáha napríklad *dropout*, o ktorom je viac uvedené v 3.4.[34]

Základnou myšlienkou algoritmu backpropagation pri procese prispôsobovania váh je opätovné uplatňovanie reťazového pravidla pre výpočet vplyvu každej váhy v neurónovej

[1] Z anglického *supervised learning*.
[2] Z anglického *overfitting*.

sieti vzhľadom na ľubovoľnú chybovú funkciu E. Vplyv váhy môžeme vyjadriť rovnicou 3.1 kde w_{ij} je váha z neurónu i do neurónu j, s_i je výstup a net_i je vážený súčet vstupov neurónu i.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} \tag{3.1}$$

Akonáhle je parciálna derivácia pre každú váhu známa, minimalizovanie chyby je docielené pomocou metódy *gradientového zostupu*[3], o ktorom je viac uvedené v sekcii 3.2.[26]

3.1.1 Chybová funkcia

Dôležitým aspektom pri trénovaní modelu je aj výber chybovej funkcie [29]. Používa sa pre ňu taktiež označenie *objektívna funkcia, loss funkcia* alebo *cost funkcia*. Táto funkcia je kritériom pre určovanie ako dobre model funguje pre daný problém. Pri algoritme backpropagation a gradientovom zostupe je potom cieľom hodnotu chyby vypočítanej touto funkciou minimalizovať. Výber jej typu záleží od konkrétneho problému, pri konvolučných neurónových sieťach to môže byť napríklad *cross-entropy loss* alebo *mean squared error loss*. Tieto populárne chybové funkcie bývajú v knižniciach pre strojové učenie implementované no pri zložitejších problémoch nemusia byť optimálnym riešením. Vytvárajú sa teda aj chybové funkcie špecifické pre daný problém ako napríklad v [23].

3.2 Gradientový zostup

Nasledujúce odstavce sú prevzaté z [26] a [27]. Gradientový zostup je zďaleka najpopulárnejším algoritmom pre optimalizáciu neurónových sietí. Účelom tohto algoritmu je minimalizovať chybu počítanú objektívnou funkciou $E(w)$ parametrizovanú váhami siete $w \in \mathbb{R}^d$ aktualizovaním váh v opačnom smere gradientu objektívnej funkcie $\nabla_w E(w)$ vzhľadom na váhy siete. Túto aktualizáciu možno vidieť v rovnici 3.2 kde η je *koeficient učenia*, t predstavuje buď *várku*, *minivárka* alebo *prvok* z trénovacej sady podľa toho, aký variant gradientového zostupu sa zvolí, w_{ij} je váha z neurónu i do j a $\frac{\partial E}{\partial w_{ij}}$ označuje parciálnu deriváciu objektívnej funkcie E podľa váhy w_{ij}. Viac o koeficiente učenia a prístupoch gradientového zostupu sa dozviete v 3.2.1 a 3.2.1.

$$w_{ij}(t+1) = w_{ij}(t) - \eta\frac{\partial E}{\partial w_{ij}}(t) \tag{3.2}$$

V súčasnosti každá moderná knižnica pre strojové učenie obsahuje implementácie rôznych algoritmov na optimalizáciu gradientového zostupu. Niektoré z nich sú popísané v sekcii 3.3.

3.2.1 Varianty gradientového zostupu

Existujú tri varianty gradientového zostupu, ktoré sa líšia podľa množstva dát použitého na výpočet gradientu objektívnej funkcie. Zvyčajne sa v závislosti od množstva dát robí kompromis medzi presnosťou aktualizácie váh a časom potrebným na aktualizovanie váh.

[3] Z anglického *gradient descent*.

Dávkový variant

Dávkový variant algoritmu gradientového zostupu[4] počíta gradient chybovej funkcie vzhľadom na parametre w pre celú trénovaciu datovú sadu a úprava váh modelu nastane na konci jednotlivých *epoch*[5] učenia siete. V rovnici 3.2 by teda t predstavovalo číslo epochy.

Keďže sa na jedno aktualizovanie váh musia vypočítať gradienty pre celú trénovaciu dátovú sadu, dávkový variant gradientového zostupu môže byť veľmi pomalý. Pri väčších dátových sadách je dokonca nepoužiteľný, keďže by sa všetky prvky tejto sady nemuseli vojsť do pamäte. Výhodou tohto prístupu ale je, že počítanie gradientu pre veľa prípadov zároveň využíva maticové násobenia, ktoré sú výpočtovo efektívne a to hlavne pri použití GPU[6].

Stochastický variant

Stochastický gradientový zostup (ďalej len *SGD*[7]) narozdiel od dávkového variantu aplikuje aktualizáciu váh vždy hneď po spracovaní jedného *prvku*. Tieto prvky sú z trénovacej sady vyberané v náhodnom poradí. V rovnici 3.2 by t tým pádom označovalo číslo prvku. SGD sa týmto zbavuje redundancie, ktorá nastáva pri použití dávkového variantu. Táto redundancia nastáva vtedy, keď dávkový variant pri použití veľkých dátových sád (ak sa vôbec zmestia do pamäte) vykonáva redundantné výpočty, keďže pred aktualizovaním parametrov siete vypočítava gradienty pre veľmi podobné vzorky.

Model sa pri využití SGD zvyčajne učí oveľa rýchlejšie. Zoberme si napríklad trénovaciu sadu, ktorá pozostáva z 500 prvkov. Počas jednej epochy SGD aktualizuje váhy 500krát, zatiaľ čo dávkový variant iba raz.

Mini-batch variant

Mini-batch variant kombinuje najlepšie vlastnosti z dvoch už spomínaných prístupov gradientového zostupu. Aktualizáciu váh aplikuje po spracovaní tzv. minivárky, ktorá pozostáva z n trénovacích vzoriek. V rovnici 3.2 v tomto prípade t označuje číslo minivárky. Počet aktualizácií parametrov za jednu epochu sa dá vypočítať ako $\frac{T}{n}$, kde T je počet trénovacích vzoriek a n veľkosť minivárky. V prípade, že T nie je deliteľné n bez zvyšku je počet aktualizácií zaokrúhlený nahor a posledná minivárka sa bude skladať z T mod n prvkov. Týmto sa zredukuje počet aktualizácií parametrov oproti SGD, čo môže viesť k stabilnejšej konvergencii[8] k minimu. Súčasne je možné pre výpočet gradientu vzhľadom na minivárku využiť vysoko optimalizované maticové násobenia implementované v moderných knižniciach pre strojové učenie rovnako ako pri dávkovom variante. Sieť sa tak bude učiť rýchlejšie oproti dávkovému prístupu a chybová funkcia nebude tak značne kolísať ako to zvyčajne býva pri SGD.

Rozdelenie prvkov do minivárok býva zväčša náhodné a ich veľkosť sa pohybuje v rozmedzí 8 do 512 prvkov. Toto však nie sú striktne dané čísla a pre niektoré prípady je vhodné použiť menšie, respektíve väčšie minivárky. Počet prvkov n v minivárke je hyperparametrom [33], ktorý ovplyvňuje:

[4] Z anglického *batch gradient descent.*

[5] Epocha – proces spracovania všetkých prvkov z trénovacej sady neurónovou sieťou

[6] GPU – *graphics processing unit*, viac na https://www.boston.co.uk/info/nvidia-kepler/what-is-gpu-computing.aspx

[7] SGD – *stochastic gradient descent*

[8] Konvergovať – približovať sa k niečomu

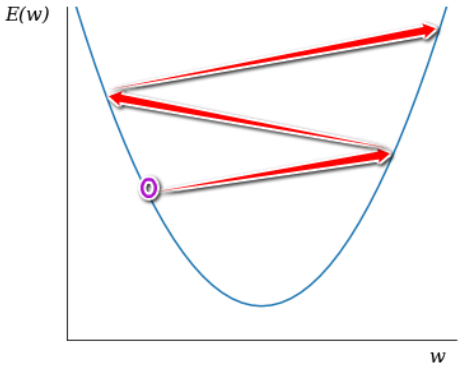
- **Počet epoch potrebných na na natrénovanie modelu** – čím menšie n, tým častejšie sú parametre modelu aktualizované. Tieto aktualizácie sú však uskutočnené na základe menšieho množstva trénovacích vzoriek. Preto sa gradienty pri minivárkach môžu značne odlišovať. V konečnom dôsledku ale menšie n konverguje rýchlejšie k bodu, v ktorom má gradientový zostup len veľmi malý vplyv na objektívnu funkciu a trénovanie sa týmto končí.
- **Trvanie trénovania modelu počas jednej epochy** – čím je n menšie, tým je doba trénovania jednej epochy dlhšia, pretože hardware nie je optimálne využitý.
- **Výslednú kvalitu modelu** – voľba hyperparametru n taktiež ovplyvňuje výslednú kvalitu trénovanej siete. Všeobecne sa predpokladá, že menšie minivárky predukujú ploché minimá a teda vedú k lepšej generalizácii [16].

Koeficient učenia

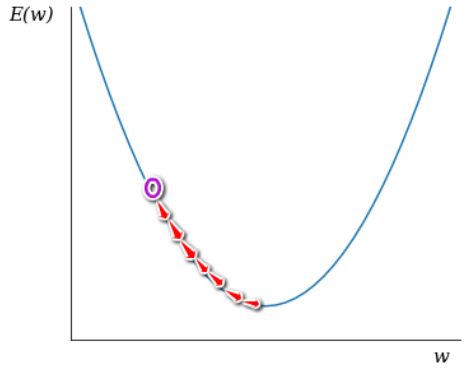
Koeficient učenia[9] je kladný skalár, ktorý určuje aký veľký krok vykonať v smere záporného gradientu pri algoritme gradientového zostupu. V rovnici gradientového zostupu 3.2 je označený gréckym písmenom η. Populárny prístup je nastaviť tento krok ako malú konštantu, zvyčajne z intervalu $\left(10^{-6}, 1\right)$. Počiatočný koeficient učenia je často **najdôležitejším** hyperparametrom a mali by sme sa vždy uistiť, že je dobre vyladený. Ak chyba po pár epochách neklesá, koeficient učenia je zrejme nastavený na príliš malú hodnotu. Naopak pri zvolení príliš vysokej hodnoty môže model z hľadiska objektívnej funkcie namiesto konvergovania divergovať[10]. Na obrázku 3.1 je možné vidieť postup minimalizovania chyby danej objektívnou funkciou algoritmom gradientného zostupu pre rôzne zvolené hodnoty koeficientu učenia. Pri hlbokých neurónových sieťach je vo väčšine prípadov nastavenie spomínaného hyperparametru na hodnotu 0.01 dobrým štartom. Netreba sa ale spoliehať, že pre konkrétny problém bude táto hodnota viesť k dobrým výsledkom. Nastaviť koeficient učenia správne sa pre mnohé problémy stáva viac umením ako vedou.[37][4]

Je síce nutné tento hyperparameter pred začatím učenia nastaviť vhodne, existujú algoritmy, ktoré nám pomáhajú dolaďovať jeho hodnotu pre jednotlivé váhy počas procesu učenia. Niektoré z týchto algoritmov sú uvedené v nasledujúcej sekcii.

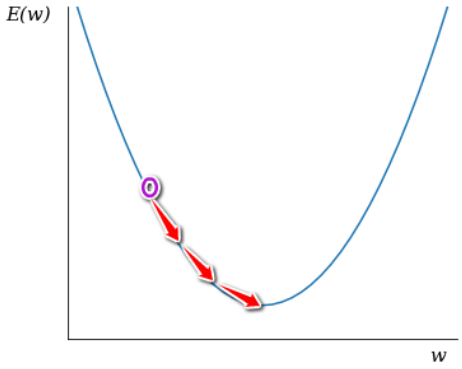
[9] Z anglického *learning rate*.
[10] Divergovať – odchyľovať sa



(a) Príliš veľká hodnota koeficientu.



(b) Príliš malá hodnota koeficientu.



(c) Vhodne zvolená hodnota koeficientu.

Obr. 3.1: Proces minimalizovania chyby pri gradientovom zostupe s rôznymi hodnotami koeficientu učenia. $E(w)$ označuje veľkosť chyby pre parameter w.

3.3 Optimalizátory gradientového zostupu

V nasledujúcich odstavcoch si predstavíme niektoré z algoritmov, ktoré sú v oblasti strojového učenia populárne a slúžia na optimalizovanie gradientového zostupu spracovaného v sekcii 3.2. Pomáhajú nám s problémami ako napríklad:

- **Upravovanie koeficientu učenia** – pri gradientovom zostupe sa tento koeficient v priebehu učenia neurónovej siete nemení, čo nie je vždy ideálne. Optimalizátory nám ponúkajú možnosť ho dynamicky upravovať tým, že sa jeho pôvodne nastavená hodnota násobí, respektíve delí hodnotami získanými zvyčajne nejakým pozorovaním chovania chybovej funkcie.
- **Veľkosť aktualizácie všetkých váh je rovnaká** – nie vždy je žiadúce aktualizovať všetky váhy modelu v rovnakom rozsahu. Napríklad pre zriedkavo sa vyskytujúce vlastnosti prvkov z trénovacej sady chceme upraviť váhy vo väčšom rozsahu ako to je pri frekventovanejších vlastnostiach.

Keďže nám pomáhajú riešiť problémy spomínané v predošlých bodoch označujú sa aj ako *algoritmy prispôsobivého učenia*[11].

3.3.1 Typy optimalizátorov

Do dnešnej doby bolo navrhnutých mnoho rôznych algoritmov, ktoré sa zaoberajú problémom primeraného prispôsobenia váh nejakou úpravou koeficientu učenia. Tieto adaptačné algoritmy môžu byť rozdelené do dvoch kategórií:

- **Globálne** – prispôsobujú koeficient učenia globálne a využívajú na to informácie o stave celej neurónovej siete (napr. smer predchádzajúceho kroku pri úprave váh).
- **Lokálne** – využívajú informácie špecifické pre jednotlivé váhy (napr. parciálnu deriváciu) na to aby prispôsobili keoficient učenia pre každú váhu zvlášť.

Lokálne adaptačné algoritmy majú bližšie ku konceptu neurónového učenia a preukázali svoju nadradenosť pri využiteľnosti oproti tým globálnym. Ako príklady adaptačných algoritmov môžeme uviesť napríklad algoritmy *Adam, Adadelta, Adagrad, AdaMax* či *RMSprop*. Algoritmu RMSprop venujeme sekciu 3.3.3 pretože bol použitý pri implementácii mojej práce. O ostatných spomenutých metódach je možné sa dozvedieť viac napríklad v [27][37].[26]

Taktiež existujú aj metódy, ktoré pomáhajú optimalizovať gradientový zostup bez toho aby nejakým spôsobom ovplyvňovali koeficient učenia. Tieto prístupy obvykle pri výpočte aktuálnej úpravy váhy zohľadňujú veľkosti jej predošlých úprav. Medzi spomínané metódy patrí napríklad *momentum* alebo *Nesterov zrýchlený gradient*. V praxi sa zvyknú kombinovať s algoritmami prispôsobovania koeficientu učenia.

3.3.2 Momentum

Jednou z metód urýchlenia trénovania siete pri gradientovom zostupe je *momentum*. Je to jedno z najjednoduchších rozšírení gradientového zostupu, ktoré je úspešne používané už desaťročia. Jeho myšlienkou je urýchliť pokrok pozdĺž dimenzií, v ktorých gradient konzistentne ukazuje rovnakým smerom a spomaliť pokrok pozdĺž dimenzií, v ktorých sa smer gradientu mení. Toto je docielené pridaním zlomku veľkosti predošlej úpravy do aktuálnej úpravy. Tento výpočet môžeme vidieť v rovnici 3.3, kde v_{ij} označuje veľkosť úpravy, γ určuje momentum a význam ostatných symbolov je rovnaký ako v rovnici 3.2, ktoré sú vysvetlené v sekcii 3.2. V rovnici 3.4 potom možno vidieť výpočet takto modifikovaného gradientového zostupu.

$$v_{ij}(t) = \gamma v_{ij}(t-1) + \eta \frac{\partial E}{\partial w_{ij}}(t) \tag{3.3}$$

$$w_{ij}(t+1) = w_{ij}(t) - v_{ij}(t) \tag{3.4}$$

Ak je pri trénovaní neurónovej siete použitá táto metóda, momentum γ sa stáva ďalším hyperparametrom a zvykne sa nastavovať na hodnotu 0.9.[27][37]

3.3.3 RMSprop

Algoritmus *RMSprop*[12] patrí medzi najpopulárnejšie algoritmy prispôsobivého učenia gradientového zostupu. O to zaujímavejšie je, že napriek jeho veľkému úspechu v praktickom

[11] Z anglického *adaptive learning algorithms*.
[12] RMSprop – Root Mean Square propagation

použití nebola publikovaná ešte žiadna rigorózna teoretická analýza RMSprop. Táto metóda bola navrhnutá Geoffom Hintonom v jednej z jeho prednášok [13] a je obzvlášť vhodná pre optimalizáciu mini-batch varianty gradientového zostupu.[22]

V RMSprop je koeficient učenia prispôsobovaný pre každú váhu samostatne, čo je často žiadúce. Tento proces je možné vidieť na rovniciach 3.5, 3.6 a 3.7. Rovnica 3.5 predstavuje substitúciu za účelom prehľadnosti. V nasledujúcich dvoch rovniciach [27] budeme teda gradient objektívnej funkcie vzhľadom na váhy $-\nabla_w E(w)$ nahrádzať znakom g a všetky výpočty budú vektorové. V rovnici 3.6 môžeme vidieť výpočet váženého priemeru gradientu umocneného na druhú, kde $P[g^2]$ je vážený priemer gradientu umocneného na druhú, β je konštanta označujúca váhu váženého priemeru z predošlej minivárky a t označuje číslo minivárky. Hinton navrhol nastavovať hodnotu β na 0.9 [13]. Ako má vážený priemer gradientu umocneného na druhú vplyv na koeficient učenia η pri algoritme RMSprop môžeme vidieť v rovnici 3.7. Hodnota ϵ sa zvykne nastavovať na veľmi malé číslo a jej účelom je zabrániť deleniu nulou.

$$g = \nabla_w E(w) \tag{3.5}$$

$$P[g^2](t) = \beta P[g^2](t-1) + (1-\beta)g(t)^2 \tag{3.6}$$

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{P[g^2](t) + \epsilon}} g(t) \tag{3.7}$$

Tento princíp optimalizácie gradientového zostupu vo všeobecnosti pri použití minivárok veľmi dobre funguje [22][18]. Implementácia RMSprop sa nachádza v rôznych populárnych knižniciach pre strojové učenie (PyTorch, Keras, Tensorflow, ...) aj napriek tomu, že nebola publikovaná.

3.4 Dropout

Nasledujúce odstavce vychádzajú z [32]. Hlboké neurónové siete obsahujú niekoľko nelineárnych skrytých vrstiev, čo z nich robí mohutné modely, ktoré sa dokážu naučiť komplikované vzťahy medzi ich vstupmi a výstupmi. Pri ich trénovaní však môže dojsť k stavu, ktorý poznáme pod názvom *pretrénovanie*, kedy má model dobré výsledky pre trénovacie dáta no nie je schopný generalizovať. V súčasnosti sú známe mnohé metódy, ktoré pomáhajú predchádzať tomuto stavu. Tieto metódy sa nazývajú regularizačné a *dropout* je jednou z tých najpoužívanejších.

Dropout pomáha riešiť problém pretrénovania modelu tzv. *vyhadzovaním* neurónov z neurónovej siete. Vyhadzovaním v tomto prípade myslíme ich dočasné odstránenie zo siete spolu s ich prichádzajúcimi a odchádzajúcimi spojeniami. Výber neurónov, ktoré budú dočasne odstránené je náhodný. V najjednoduchšom prípade je neurón zachovaný s fixnou pravdepodobnosťou p bez ohľadu na ostatné neuróny. Pri testovaní siete sú váhy prepojení z neurónu škálované podľa použitej pravdepodobnosti pre zachovanie neurónu. To znamená, že ak bol neurón zachovaný s pravdepodobnosťou p počas trénovania, tak pri testovaní sú odchádzajúce prepojenia násobené hodnotou p. Toto zaručuje, že pre každý neurón v skrytých vrstvách je očakávaný výstup (pod rozdelením použitým na vyhadzovanie neurónov počas trénovania) rovnaký ako skutočný výstup pri testovaní. Dropout má najväčší účinok pri použití plne prepojených vrstiev, avšak preukázal svoju využiteľnosť aj pri konvolučných neurónových sieťach [35].

Kapitola 4

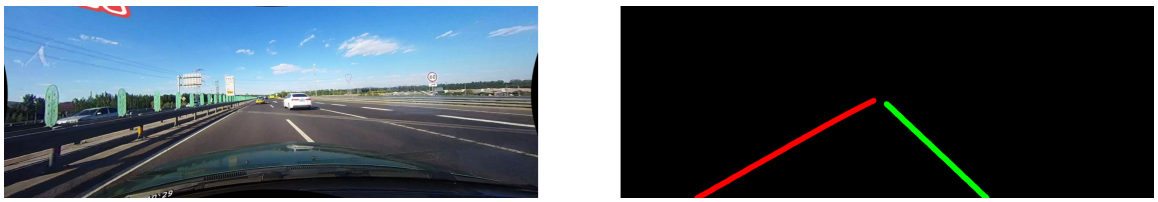
Dátová sada

Získanie vhodnej dátovej sady bezpochyby patrí medzi najdôležitejšie komponenty trénovania konvolučnej neurónovej siete pod dohľadom. Princíp trénovania pod dohľadom je načrtnutý v sekcii 3. Nasledujúce odstavce sa venujú práve dátovej sade, ktorú som použil pri trénovaní modelu pre rozpoznávanie hraníc jazdného pruhu.

4.1 Vzorky dátovej sady

Použitá trénovacia sada sa skladá z RGB obrázkov, ktorými sú snímky z palubnej kamery vozidla. Každý z týchto obrázkov má priradený očakávaný výstup zo siete tzv. *label*. Label pre trénovací obrázok je taktiež obrázok, kde intenzita zeleného kanálu určuje pravdepodobnosť, že daný pixel patrí pravému okraju jazdného pruhu a intenzita červeného kanálu zas určuje pravdepodobnosť, že daný pixel patrí ľavému okraju jazdného pruhu. Príklad trénovacej vzorky a jeho labelu môžete vidieť na obrázku 4.1.

Všetky trénovacie vzorky a k nim priradené labely sú z pôvodného rozlíšenia zmenšené na rozlíšenie 160×80 pixelov. Dôvodom zmenšovania rozlíšenia obrázkov je hlavne fakt, že výpočtová náročnosť konvolúcie rastie s rozmermi jej vstupu. Hranice jazdného pruhu sú pri tomto rozlíšení obrázkov ešte rozpoznateľné, a to modelu stačí na to aby sa dokázal na nich učiť.



Obr. 4.1: Vzorka dátovej sady (vľavo) a jej label (vpravo).

4.2 Vytvorenie vhodnej dátovej sady

Dátová sada použitá pri trénovaní pochádzala z dvoch zdrojov. Spôsob vytvárania vlastných dát a aj modifikácia prevzatých dát sú popísané v nasledujúcich odstavcoch.

4.2.1 Získanie vlastných dát

Po získaní viacerých videozáznamov z palubnej kamery s rôznych zdrojov som vytvoril *data pipeline*[1] systém, ktorý je určený na uľahčovanie procesu vytvárania vzoriek a labelov pre trénovanie modelu. Tento systém tvoria nasledujúce prvky:

1. **Extraktor snímok** – program určený na prehrávanie videa, extrahovanie a automatické uloženie snímku vybratého používateľom do priečinku. Meno extrahovaného snímku obsahuje jeho identifikačné poradové číslo.

2. **Nástroj pre označovanie hraníc pruhu** – nástroj pracuje s extrahovanými obrázkami a umožňuje v nich používateľovi označovať hranice jazdného pruhu. Používateľ si prepína medzi režimami nástroja podľa toho, ktorú hranicu jazdného pruhu ide označovať. Pozície bodov zakliknutých používateľom sú potom uložené vo formáte JSON[2] do súboru, ktorého názov obsahuje identifikačné číslo spracovávaného obrázku. Automaticky sú z priečinku vyberané iba tie obrázky, pre ktoré ešte neexsistuje súbor s pozíciami označených bodov.

3. **Generátor labelov** – pracuje so súbormi, ktoré obsahujú pozície bodov označujúcich hranice jazdného pruhu pre jednotlivé trénovacie obrázky uložené vo formáte JSON. S ich pomocou generuje labely definujúce očakávaný výstup z modelu. Požadovanú hrúbku generovaných čiar je možné v tomto nástroji jednoducho nastaviť.

Pri dbaní na kvalitu je tento proces vytvárania trénovacej sady zdĺhavý, aj napriek jeho uľahčeniu spomenutými nástrojmi. Trénovacie dáta vytvorené týmto systémom boli používané hlavne pri začiatočných experimentoch s trénovaním modelu. Záverom týchto pokusov o natrénovanie modelu bolo zistenie, že trénovacích dát budem potrebovať omnoho viac a preto som sa rozhodol zohnať a spracovať už existujúcu dátovú sadu, o ktorej sa zmieňujem v nasledujúcej sekcii. Po prečistení dát vytvorených systémom spomínaným v tejto sekcii od nevhodných vzoriek bolo vo finálnej dátovej sade použitých 410 z týchto vzoriek.

4.2.2 Prevzatá dátová sada

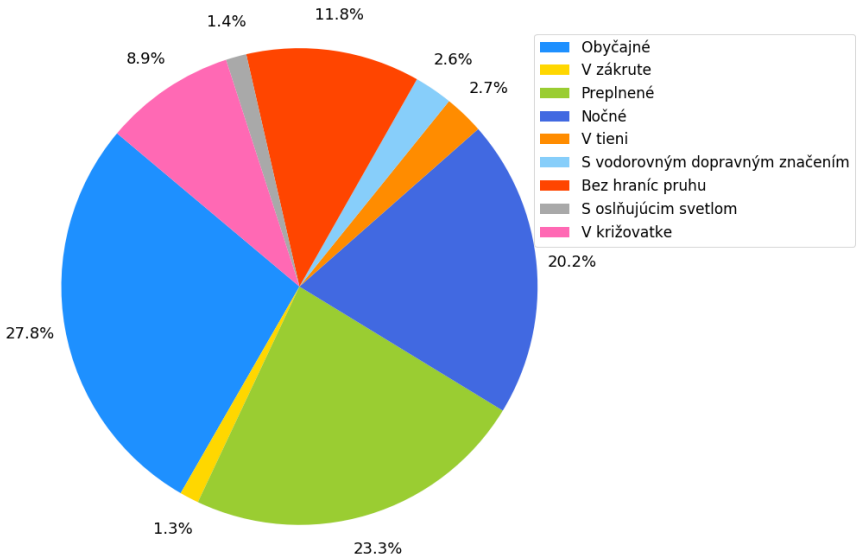
Pre zvýšenie počtu trénovacích vzoriek som sa rozhodol spracovať dátovú sadu *CULane* [24]. Všetky vzorky v tejto sade boli získané z videozáznamov jazdy po Pekingu šiestimi rôznymi autami. Jedná sa o pomerne rozsiahlu a novú sadu dát z roku 2018. Zo všetkých voľne dostupných dátových sád sa zdala byť najvhodnejšou na trénovanie môjho modelu. Jej súčasťou je aj data pipeline systém na vygenerovanie labelov pre jednotlivé vzorky. Pôvodne totižto sada obsahuje iba snímky z palubnej videokamery a textové súbory obsahujúce pozície bodov označujúcich hranice pruhov. Autori okrem hraníc jazdného pruhu vozidla označovali taktiež hranice susedného ľavého a pravého pruhu (ak sú prítomné). Preto som si nimi vytvorený data pipeline systém prispôsobil tak, aby vygenerované labely zodpovedali labelom, ktoré boli vytvorené systémom opísaným v predchádzajúcej sekcii a takto sa moja dátová sada rapídne rozrástla.

[1] Data pipeline – séria prvkov, ktoré nejakým spôsobom spracovávajú dáta, kde výstup z jedného prvku je vstupom toho nasledujúceho

[2] JSON – JavaScript Object Notation, viac na https://www.json.org/

4.2.3 Finálna dátová sada

Po prispôsobení prevzatej CULane dátovej sady a jej spojení s tou mojou vznikla finálna sada, ktorá bola neskôr použitá na trénovanie modelu. Keďže CULane je veľmi rozsiahla, finálna sada pozostáva zo 133 645 vzoriek, ktoré sú rozdelené na trénovaciu časť, validačnú časť a testovaciu časť. Trénovacia a validačná časť spolu obsahujú 97 555 vzoriek, ktoré boli vybraté tvorcami CULane sady, plus mnou vytvorené dáta. O rozdelení týchto dát počas trénovania modelu sa zmieňujem v sekcii 6.2.2. Natrénovaný model bol okrem vzoriek z testovacej časti testovaný aj na snímkach získavaných priebežne aj po jeho natrénovaní, čo uvádzam v sekcii 7. Približné rozdelenie sady podľa vlastností, ktoré charakterizujú jednotlivé snímky sú uvedené na grafe 4.2. Správnosť údajov pre CULane sadu nebola overovaná ale vychádza z [24].



Obr. 4.2: Graf zobrazujúci rozdelenie dátovej sady podľa vlastností charakteristických pre vzorky.

Kapitola 5

Návrh riešenia

Dôkladným návrhom riešenia sa dá predísť nepríjemným komplikáciám pri jeho neskoršej implementácii. Preto odstavce tejto kapitoly budú venované práve návrhu riešenia problému rozpoznávania hraníc jazdného pruhu v záberoch palubnej kamery. Bude tu spomenutý existujúci prístup k tomuto problému, ktorý neurónové siete nevyužíva. Vysvetlená teda bude aj motivácia pre použitie konvolučných neurónových sietí namiesto spomínaného prístupu. Dozviete sa o architektúre modelu hlbokej konvolučnej neurónovej siete, ktorý som navrhol ako riešenie a niečo málo aj o architektúrach, ktorými je tento model inšpirovaný. Taktiež sú tu definované očakávané výsledky implementácie.

5.1 Rozopoznávanie pomocou techník počítačového videnia

Existujú rôzne kombinácie algoritmov počítačového videnia, ktorých výsledkom je možné detegovať hranice jazdného pruhu. Keďže sú si častokrát tieto kombinácie podobné, a v ich krokoch sa zvyknú jednotlivé používané algoritmy opakovať, predstavíme si iba jedno z možných riešení.

Pri tomto prístupe sa predpokladá, že väčšina ciest smeruje relatívne priamo, a že zákruty sú len pod malým uhlom za účelom zachovania plynulosti premávky. Preto táto implementácia funguje na princípe detekcie výrazných rovných čiar v záberoch palubnej kamery za pomoci techník detekcie okrajov. Proces získania detegovaných pruhov môžeme rozdeliť do nasledovných bodov:

1. **Prevedenie obrázku do odtieňov šedej** – týmto z pôvodného trojkanálového obrázku (za predpokladu, že snímok z kamery je RGB) dostaneme obrázok s jedným kanálom. Dôvodom tejto operácie je hlavne urýchlenie celého procesu, keďže po jej aplikovaní každý pixel obsahuje menej informácií, s ktorými nasledujúce kroky musia pracovať.

2. **Aplikovanie Cannyho detektoru hrán** – jedná sa o viacstupňový algoritmus, ktorý sa skladá z:

   (a) redukcie šumu

   (b) nájdenia intenzity a smeru gradientu pre každý pixel obrázku

   (c) odstránenia pixelov, ktoré nepredstavujú hranu

   (d) hysterézneho prahovania

Viac informácií o jednotlivých stupňoch algoritmu je uvedených napríklad v [6]. Tieto časti algoritmu sú v moderných knižniciach pre počítačové videnie (napr. OpenCV) zapúzdrené pod názvom Cannyho detektor hrán. Jeho výstupom je maska obsahujúca body nájdených hrán.

3. **Aplikovanie masky oblasti záujmu** – týmto sa odstránia prebytočné hrany, ktoré boli nájdené a mohli by nepriaznivo ovplyvniť celkový výstup. Oblasť záujmu označuje časť obrázku, v ktorej očakávame prítomnosť hraníc pruhu.

4. **Houghova transformácia** – výstup z Cannyho detektoru po aplikovaní masky oblasti záujmu predstavuje sériu bielych bodov, pre ktoré je potrebné nájsť rovnice nimi prechádzajúcich priamok. Všetky priamky prechádzajúce určitým bodom v karteziánskej súradnicovej sústave predstavujú jednu priamku v *Houghovom priestore*. Týmto spôsobom môžeme identifikovať, ktoré body ležia na jednej priamke a nájdením priesečníku priamiek v Houghovom priestore pre každý bod z masky je získaný jej predpis. Správnym nastavením parametrov, ako je napríklad minimálna dĺžka úsečky, ktorú body tvoria alebo maximálna povolená medzera medzi bodmi tej istej čiary, dostaneme predpis dvoch čiar, ktoré sú vykreslené do výstupu.

Použitím opísanej kombinácie algoritmov sme schopní detegovať čiary okrajov jazdného pruhu v snímke zachytávajúcej rovnú cestu za vhodných podmienok. Hlavným problémom tejto metódy je, že už pri mierne ostrejších zákrutách nie je spoľahlivá. To isté platí pri snímkoch zachytávajúcich zložitejšie podmienky, akými je napríklad jazda v noci za slabého osvetlenia alebo neprítomnosť krajnice cesty. Dokonca aj umiestnenie palubnej kamery zohráva úlohu, keďže maska oblasti záujmu sa pre rôzne umiestnenia môže líšiť a môže mať za následok nedetegovanie hranice pruhu, poprípade detegovanie nesprávnej hranice / hraníc. Rozlíšiť pravú a ľavú hranicu je možné z predpisov nájdených čiar, avšak pri spomenutých problémoch sa môže aj toto stať problematickým. Tu sa dostávajú na scénu hlboké konvolučné neurónové siete, ktoré prisľubujú mohutnejšie a stabilnejšie riešenie pre problém rozpoznávania hraníc jazdného pruhu.

5.2 Navrhnutý model

Riešenie spomenuté v predošlej sekcii funguje dobre, no len v ideálnych podmienkach a má teda veľmi malý okruh využitia. Za použitia hlbokých konvolučných neurónových sietí som sa rozhodol vytvoriť model, ktorý by pomohol rozšíriť tento okruh a predchádzať tak vyššie spomínaným problémom.

5.2.1 Prístup konvolučných neurónových sietí k problému

Hlboké konvolučné neurónové siete preukázali široké uplatnenie pri riešení rôznych problémov týkajúcich sa spracovania obrazu. Jednou z ich najsilnejších stránok je to, že pre riešenie daného problému nie je potrebné žiadne predspracovanie v podobe extrakcie nejakých čŕt obrázku. Pri učení sa takáto sieť naučí črty extrahovať pomocou využitia konvolúcie, ktoré bolo opísané v sekcii 2.2.1. Mapy čŕt vyprodukované použitím konvolúcie sú ďalej posielané do nasledujúcej vrstvy. Tu sú vstupné mapy čŕt konvolvované s ďalšími filtrami a takto vznikajú stále abstraktnejšie mapy čŕt. Počas učenia sa teda filtre prispôsobia tak, aby boli schopné rozpoznávať črty podstatné pre vyprodukovanie výstupu, ktorý je sieti predložený ako očakávaný. Týmto sa úplne eliminuje nutnosť manuálneho vybratia čŕt, ktoré

chceme použiť v procese získavania očakávaného výstupu – správne navrhnutá sieť sa ich naučí sama.

5.2.2 Architektúra navrhnutej hlbokej konvolučnej siete

Správne zvolená architektúra neurónovej siete rozhodne spadá medzi najdôležitejšie faktory pre získanie dobre natrénovaného modelu pre riešenie daného problému. V mojom prípade som potreboval zvoliť architektúru, ktorá pomocou jej skrytých vrstiev dokáže extrahovať potrebné črty pre určenie pravdepodobnosti výskytu ľavej a pravej hranice jazdného pruhu a vyprodukovať výstup s rovnakými rozmermi ako má vstup do siete. Zároveň by natrénovaný model so zvolenou architektúrou mal byť schopný spracovávať snímky takou rýchlosťou, aby bolo možné ho uplatniť v reálnych situáciách.

Inšpirácia

Pri rozhodovaní o počte a usporiadaní jednotlivých vrstiev, spomenutých v sekcii 2.2, a ich hyperparametrov v mojej sieti som sa inšpiroval niekoľkými architektúrami, ktoré preukázali kvalitné výsledky pri problémoch spracovania obrazu. Môj návrh architektúry má najbližšie k architektúre SegNet [2], ktorá je určená pre segmentáciu obrazu. Využíva vzor kóder-dekóder, ktorý je vhodný aj pre riešenie problému detekcie hraníc jazdného pruhu. V kódovacej časti sú informácie (črty) zakódované do matíc malých rozmerov a v dekódovacej časti sú dekódované pomocou transponovaných konvolúcií na výstup s rozmermi identickými tým vstupným. Z dôvodu odľahčenia modelu som sa rozhodol moju architektúru urobiť plytšou oproti SegNetu, čo má za následok zníženie počtu trénovateľných parametrov a v konečnom dôsledku aj rýchlejšie spracovanie snímky.

Konečná verzia architektúry

Model, ktorý som navrhol, neskôr implementoval a natrénoval, sa skladá z dvoch sémanticky odlišných častí – *kóderu* a *dekóderu*. Úlohou kóderu je namapovať vstupný RGB obrázok na mapy čŕt, zatiaľ čo dekóder má na vstupe mapy čŕt vytvorené kóderom a spracováva ich za účelom vytvorenia výstupu. Každá z týchto častí sa skladá zo sedemnástich vrstiev, vrátane vrstiev aktivačných. Ich usporiadanie je uvedené na obrázku 5.1.

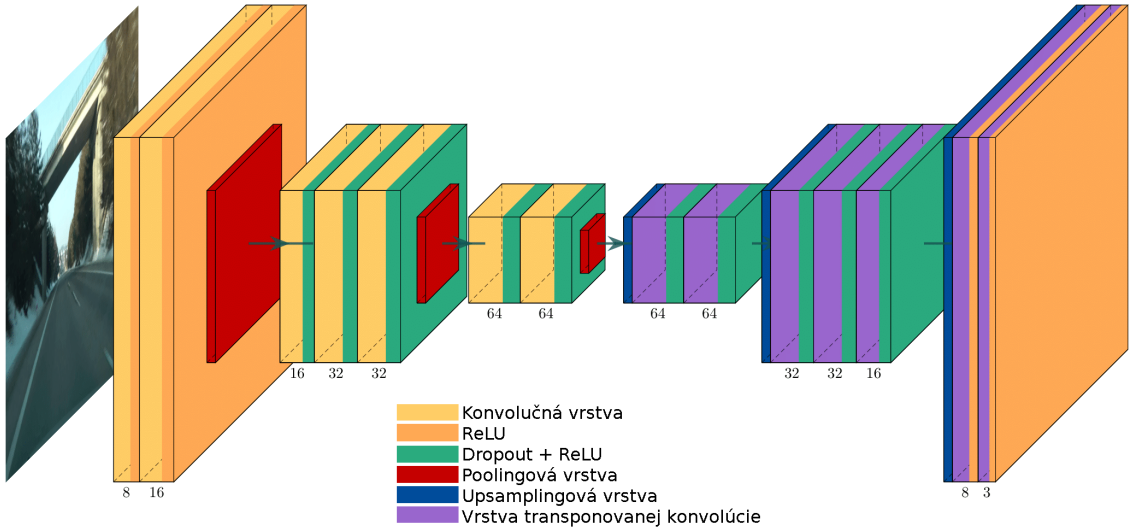
Kóder obsahuje 2D konvolučné vrstvy, poolingové vrstvy a aktivačné vrstvy. Za každou konvolučnou vrstvou nasleduje vrstva aktivačná, ktorá využíva funkciu ReLU (sekcia 2.2.3) na aplikovanie nelinearity. ReLU je najpoužívanejšou aktivačnou funkciou a rozhodol som sa ju použiť z dôvodu jej skvelých výsledkov konkrétne v oblasti konvolučných neurónových sietí. Ani po preštudovaní článkov o iných alternatívach som nedospel k záveru, že by som jej zámenou za inú docielil lepšie výsledky. Vo všetkých konvolučných vrstvách je použitý 3×3 konvolučný filter, ktorý sa posledné roky stáva stále populárnejším a jeho využitie preukázalo lepšie výsledky, ako pri použití filtrov väčších rozmerov[1]. Rovnako v každej konvolučnej vrstve má veľkosť kroku pre posúvanie filtra hodnotu 1 a nie je použitý žiadny padding. Počet použitých filtrov sa postupne zväčšuje s narastajúcou hĺbkou kóderu čo je dokumentované na obrázku 5.1. Na posledných 5 z týchto vrstiev kóderu je aplikovaný dropout s pravdepodobnosťou $p = 0.2$ pre dočasné vyhodenie neurónu pri trénovaní. Posuvné okno, nad ktorým prebieha max pooling v poolingových vrstvách má rozmery 2×2. Krok tohto okna pri posúvaní je 2, čo znamená, že jednotlivé regióny, na ktoré je aplikovaný max

[1] Odvodené z architektúr úspešných v súťaži ILSVRC – Image Large Scale Visual Recognition Challenge. Viac na http://www.image-net.org/challenges/LSVRC/.

pooling sa neprekrývajú. Pri použití týchto hyperparametrov budú teda rozmery jednotlivých máp čŕt na výstupe z poolingovej vrstvy dva krát menšie, a to značne urýchli výpočty v nasledujúcej vrstve.

V dekóderi sú konvolučné vrstvy nahradené vrstvami transponovaných 2D konvolúcií a poolingové vrstvy vrstvami upsamplingovými. Aktivačná vrstva s funkciou ReLU je tu zakomponovaná rovnako ako v kóderi. Upsamplingová vrstva je náprotivkom vrstvy poolingovej – jej účelom je zväčšiť výšku a šírku vstupu podľa nejakého faktoru. V každej tejto vrstve je použitý faktor veľkosti 2. V poolingových vrstvách kóderu sa teda rozmery vstupu dvojnásobne zmenšujú a pri upsamplingu v dekóderovej časti sa dvojnásobne zväčšujú. Keďže pri zväčšovaní sa z jedného prvku stáva prvkov viac, logicky nie je možné použiť operáciu rovnakú ako v max poolingu. Preto som zvolil algoritmus *najbližšieho suseda*, ktorý patrí medzi najjednoduchšie a výpočtovo najúspornejšie z možných variantov. Hodnoty hyperparametrov vo vrstvách transponovaných konvolúcií a dropoutu v dekóderi sú rovnaké ako pri konvolučných vrstvách a dropoute v kóderi pričom dropout je aplikovaný na prvých 5 vrstiev transponovaných konvolúcií.

Takto postavený model obsahuje 180 595 trénovateľných parametrov a tým sú nároky na výpočtovú silu nižšie oproti väčším architektúram konvolučných sietí, ktoré často obsahujú aj milióny parametrov.



Obr. 5.1: Architektúra navrhnutého modelu. Čísla pod jednotlivými vrstvami označujú počet použitých filtrov.

5.3 Ciele

Cieľom práce je implementovať spomínanú architektúru, natrénovať ju na dátovej sade a otestovať jej schopnosti rozpoznávať hranice jazdného pruhu v rôznych situáciách. Pre trénovanie zvoliť vhodné hyperparametre, chybovú funkciu a optimalizačný algoritmus. Pomocou sledovania vývinu hodnôt chybovej funkcie pre trénovacie a validačné dáta tieto hyperparametre ladiť za účelom získania čo najlepších výsledkov. Finálnu verziu natrénovaného modelu potom otestovať a získať znalosti o jeho chovaní v rôznych situáciách, ktoré sa môžu odohrávať na snímkach z palubnej kamery. Rovnako vyskúšať jeho schop-

nosť detekcie pri použití viacerých druhov kamier a umiestnení týchto kamier pod rôznymi uhlami v prednej časti vozidla. Na koniec zistiť rýchlosť spracovania snímok a zhodnotiť využiteľnosť tohto modelu ako súčasť nejakého systému.

Kapitola 6

Implementácia a trénovanie

V tejto kapitole sa dozviete o nástrojoch a prostrediach, ktoré boli použité pri implementácii, ako aj o samotnej implementácii riešenia. Spomenuté tu bude aj trénovanie modelu a jeho priebeh. Výsledkom implementácie a trénovania je model pre rozpoznávanie hraníc jazdného pruhu, ktorý je možné importovať do rôznych aplikácií za predpokladu, že podporujú technológiu pomocou ktorej bol vytvorený. Existuje aj možnosť tento model migrovať do iných knižníc a teda ho využiť aj v rozličných iných systémoch.

6.1 Použité nástroje

Všetky vytvorené skripty použité pri implementácii sú napísané v jazyku *Python* verzie 3.6. Okrem neho bol použitý ešte jazyk *C++*, v ktorom bola napísaná časť prevzatého data pipeline spomínaného v sekcii 4.2.2, ktorú bolo potrebné pre moje využitie modifikovať. Jazyk Python bol zvolený kvôli jeho schopnosti pohodlne pracovať s dátami, širokej komunite užívateľov a pretože najmodernejšie knižnice pre strojové učenie poskytujú prívetivé API[1] práve v jazyku Python.

V skriptoch pre získavanie dátovej sady som použil knižnicu NumPy spolu s OpenCV 3+ a pre prácu s modelom a prípravu dát na trénovanie knižnicu PyTorch verzie 0.4.1. Túto knižnicu som zvolil hlavne kvôli možnosti pracovať s modelom a jeho trénovaním na nižšej úrovni na rozdiel od iných variantov, akým je napríklad knižnica Keras. Tento prístup umožňuje písať si vlastné trénovacie a validačné funkcie, mať lepšiu kontrolu nad výpočtom a tým celkovo nad trénovaním modelu. Skripty pre testovanie priebežných výsledkov rovnako ako aj trénovací skrypt som neskôr prepísal z Pythonu do Jupyteru, využívajúceho IPython – Interaktívny Python [25]. O dôvodoch tohto rozhodnutia sa zmieňujem v sekcii 6.3.

6.2 Implementácia jednotlivých častí

Všetky časti implementácie, od získavania a manipulácie dátovej sady až po vizualizovanie výsledkov natrénovaného modelu, sú rozdelené do viacerých skriptov. Týmto sa zachováva modularita a možnosť jednoducho upravovať jednotlivé kroky spracovania dát podľa potreby. Vytvorené pomocné skripty pre manipuláciu obrázkov a vizualizovanie výsledkov nie sú v tejto sekcii spomenuté, lebo nie sú svojou implementáciou obzvlášť zaujímavé.

[1] API – Application Programming Interface, predstavuje súbor funkcií a procedúr umožňujúcich prístup k funkciám alebo dátam operačného systému, aplikácie alebo inej služby.

6.2.1 Data pipeline pre získanie časti dátovej sady

Keďže mojím pôvodným plánom bolo použiť iba mnou vytvorené dáta, táto časť implementácie bola dôležitá. V konečnej verzií bola použitá aj rozsiahla dátová sada spomínaná v sekcii 4, no napriek tomu si tento data pipeline zaslúži pár odstavcov.

Extraktor snímok

Najjednoduchšou časťou, ktorá tvorí data pipeline pre vytvorenie dátovej sady je jednoduchý videoprehrávač. Pre načítanie videa a iterovanie jeho snímkami je tu použitá knižnica OpenCV. Jeho jediným účelom je poskytnúť jednoduchý a pohodlný spôsob extrahovania zvolených snímok z videa a ich automatické ukladanie. Ovládanie je úplne minimalistické – po stlačení *medzerníku* sa uloží aktuálne zobrazená snímka do priečinku definovaného v skripte a po stlačení klávesy q sa prehrávanie videa spolu so skriptom ukončí. Meno automaticky uloženej snímky obsahuje jej poradové číslo. Poradové číslo budúcej snímky je ukladané do textového súboru, aby nenastala kolízia v názvoch snímok po opätovnom spustení skriptu.

Nástroj pre označovanie hraníc pruhu

Úlohou tohto nástroja ja urýchliť proces označovania hraníc jazdného pruhu. Pre tento účel je taktiež využitá knižnica OpenCV, pomocou ktorej sú používateľovi skriptu postupne zobrazované snímky z priečinku definovaného v skripte a to iba tie, pre ktoré sa vo výstupnom priečinku ešte nenachádza súbor so súradnicami bodov označených bodov.

Ovládanie tohto skriptu je taktiež jednoduché. Pomocou kláves l a p sa vyberá režim označovania – označovanie ľavej hranice alebo označovanie pravej hranice. Pre istotu je aktívny režim zobrazený v hornom ľavom rohu okna. Klikaním ľavým tlačítkom myši sú označované body okrajov a tlačítkom n sa uložia označené body do automaticky vygenerovaného súboru a prejde sa na ďalšiu snímku. Súradnice jednotlivých bodov sú v súbore uložené vo formáte JSON nasledovne:

```
{
    "right_border": [
        [
            1687.1489361702127,
            934.7727272727273
        ],
        [
            1282.723404255319,
            695.4545454545454
        ]
    ],
    "left_border": [
        [
            537.1914893617021,
            1028.8636363636363
        ],
        [
            988.595744680851,
            664.7727272727273
```

```
        ]
    ]
}
```

Pri pomýlení sa je možné zresetovať označené body klávesou *Esc* a klávesou *q* ukončiť skript. Pred uložením označených súradníc je ešte vygenerovaný obrázok, ktorý obsahuje čiary vykreslené podľa zadaných súradníc do pôvodného obrázku. Tento obrázok slúži iba pre kontrolu – je ľahšie všimnúť si prípadnú chybu na ňom, ako v súbore so súradnicami.

Generátor labelov

Poslednou časťou procesu vytvárania labelov je nástroj pre vygenerovanie obrázkového labelu zo súradníc bodov označujúcich hranice pruhu. Dôvodom, prečo sa tento proces nedeje hneď pri označovaní bodov je ten, že ak by sme potrebovali zmeniť hrúbky alebo farby čiar museli by sme nanovo označovať všetky obrázky. V tomto skripte jednoduchou zmenou troch parametrov vygenerujeme zmenené labely.

6.2.2 Trénovací skript

Ako už bolo spomenuté v sekcii 6.1, architektúra modelu a jeho trénovanie boli implementované pomocou knižnice PyTorch. Táto implementácia sa nachádza v Jupyter notebooku *define_and_train_model.ipynb*.

Trieda modelu

Trieda **CNN** reprezentuje architektúru modelu navrhnutú v sekcii 5.2.2 a dedí z triedy **Module**, ktorá je základnou triedou všetkých neurónových sietí v spomínanej knižnici. Metóda ***forward*** triedy CNN definuje výpočet vykonaný pri každom volaní objektu tejto triedy. Z pohľadu neurónových sietí je to teda proces spracovania vstupu všetkými vrstvami siete v definovanom poradí a vyprodukovanie výstupu.

Dátová sada a DataLoadery

Dátová sada je v tomto skripte reprezentovaná triedou **MyDataset**, ktorá je podtriedou abstraktnej triedy **Dataset**. V tejto triede sú definované metódy pre inicializáciu dátovej sady, získanie veľkosti dátovej sady a vybratie obrázku a jeho labelu z dátovej sady na základe indexu. V metóde ***__getitem__*** pre vybratie obrázku a jeho labelu z dátovej sady sa zároveň tieto dáta normalizujú z rozsahu pixlov $[0, 255]$ na rozsah $[0, 1]$ a konvertujú sa na **Tensor** objekty, s ktorými PyTorch pracuje. Pre zvýšenie efektivity pri trénovaní sa tu tieto objekty presúvajú na GPU, ak je k dispozícii.

Pred vytvorením trénovacej a validačnej dátovej sady sú dáta načítané z komprimovanej podoby a následne náhodne rozdelené v pomere $80 : 20$, kde väčšia časť pripadá trénovacej sade. Po vytvorení objektov MyDataset pre obidve sady sú tieto objekty použité na vytvorenie **DataLoaderov**, ktoré slúžia na efektívne náhodné generovanie minivárok z trénovacej a validačnej sady.

Trénovacie a validačné funkcie

Funkcie pre trénovanie boli implementované podľa algoritmu opísaného v sekcii 3.2. Konkrétne bol použitý mini-batch variant implementovaný pomocou vyššie spomenutých DataLoaderov a gradientový zostup je optimalizovaný pomocou optimalizačného algoritmu.

O konkrétnych hodnotách hyperparametrov trénovania a zvolených algoritmoch sa zmieňujem v sekcii 6.3.1. Po každej epoche sú váhy parametrov modelu ukladané a tým pádom aj keď sa model začne pretrénovávať stále máme k dispozícii posledný najkvalitnejšie natrénovaný model. Veľkosť chyby pre trénovaciu aj validačnú sadu je ukladaná a vypisovaná, čo umožňuje sledovať vývoj trénovania a podľa potreby ho zrušiť a dolaďovať hyperparametre.

6.3 Trénovanie modelu

Trénovanie implementovaného modelu prebiehalo v prostredí Google Colaboratory. Spomínané prostredie je nástroj pre vzdelávanie a výskum v oblasti strojového učenia od spoločnosti Google. Je založené na prostredí Jupyter notebook a to bolo dôvodom prepísania trénovacieho skriptu práve do IPythonu. Toto prostredie umožňuje využívať výpočtové prostriedky prideleného serveru vrátane GPU.

6.3.1 Použité hyperparametre a algoritmy

Dôležitou úlohou pri trénovaní bolo správne zvoliť hyperparametre pre trénovanie, ktorými sú veľkosť minivárky, koeficient učenia a parametre optimalizačného algoritmu. Rovnako dôležité bolo aj vybrať vhodnú chybovú funkciu. Za optimalizačný algoritmus gradientového zostupu som zvolil algoritmus RMSprop popísaný v sekcii 3.3.3 a ako chybovú funkciu funkciu strednej kvadratickej chyby. Táto funkcia počíta strednú kvadratickú chybu medzi každým prvkom vstupu x a výstupu y. Zvolené hodnoty hyperparametrov som postupne ladil až som dospel k nasledujúcim hodnotám, pri ktorých sa model dobre trénoval:

- Počiatočný koeficient učenia $\eta = 0.0001$,
- Veľkosť minivárky pri mini-batch gradientovom zostupe $n = 256$,
- Konštanta označujúca váhu váženého priemeru z predošlej minivárky v RMSprop $\beta = 0.99$,
- Konštanta zabraňujúca deleniu nulou v RMSprop $\epsilon = 10^{-8}$,
- Momentum vo finálnej verzii pri použití RMSprop nebolo využité a teda bolo nastavené na $\gamma = 0$.

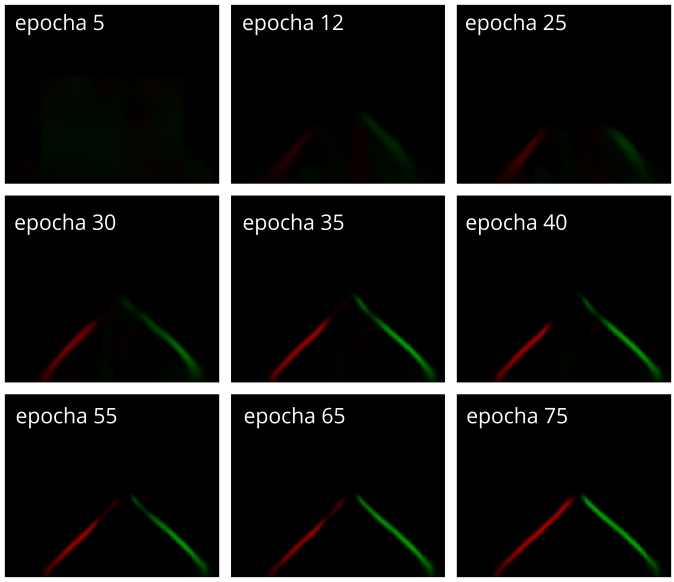
Pri použití týchto hodnôt a zvolených algoritmov chyba v priebehu trénovania klesala a výsledky boli stále lepšie, o čom sa dočítate v nasledujúcej sekcii.

6.3.2 Priebeh trénovania

Trénovanie prebiehalo v spomínanom prostredí Google Colaboratory na grafickej karte NVIDIA Tesla T4 s pamäťou veľkosti 16 GB. Model sa trénoval približne 7 hodín a 12 minút kým dospel do štádia, kedy sa začal zhoršovať pri generalizovaní a trénovanie bolo teda po pár ďalších epochách ukončené.

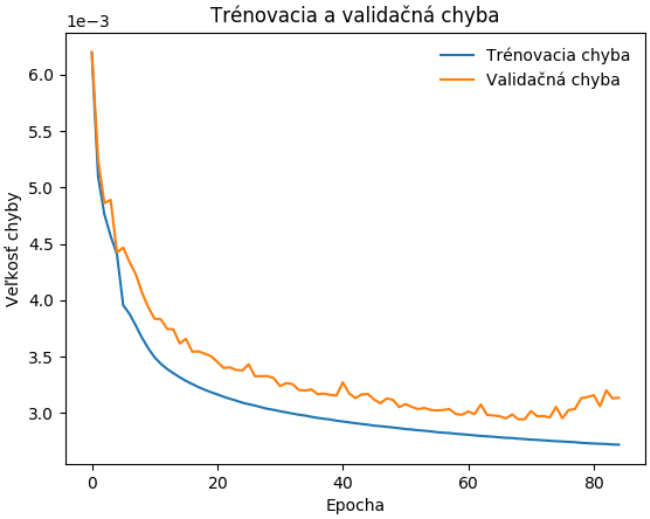
Trénovaciu funkciu som implementoval tak, aby ukladala stav trénovaných parametrov modelu po každej epoche. To mi umožňovalo sledovať vývoj správania modelu už počas jeho trénovania pomocou vizualizácie jeho predikcií. Tento vývoj v niektorých zvolených kontrolných bodoch je uvedený na obrázku 6.1. Predikcie sú pre obrázok z testovacej sady.

Najlepšie výsledky pri generalizovaní preukázal model po epoche číslo 75. Veľkosť chyby pre validačnú sadu sa v nasledujúcich epochách už prestala zmenšovať, práve naopak začala



Obr. 6.1: Vizualizácia predikcií modelu pri jeho trénovaní.

sa zväčšovať. Toto môžeme vidieť aj na grafe 6.2 kde sú zobrazené hodnoty chyby počas jednotlivých epoch pri trénovaní modelu.



Obr. 6.2: Graf zobrazujúci veľkosť chyby v jednotlivých epochách počas trénovania modelu.

Kapitola 7

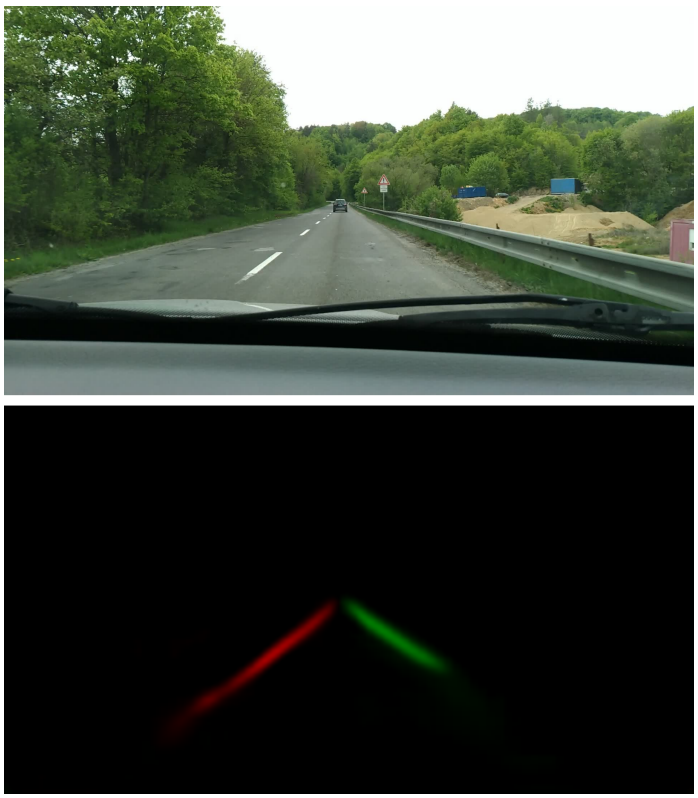
Testovanie natrénovaného modelu

V tejto kapitole sú zhrnuté výsledky testovania natrénovaného modelu. Okrem počiatočného testovania na snímkach z testovacej sady som ďalšie testy robil s využitím videozáznamov z rôznych palubných kamier a dokonca aj z mobilu, získavaných postupne počas semestra. Z týchto videí som potom vybral nie len prípady, v ktorých model funguje, ale aj prípady, ktoré poukazujú na jeho nedostatky a vytvárajú tak priestor na jeho ďalšie zdokonaľovanie. Všetky snímky, ktoré som použil pri testovaní boli zo slovenských alebo českých ciest. Schopnosť modelu rozpoznávať hranice bola hodnotená pomocou vizualizovania jeho predikcií v rôznych situáciách.

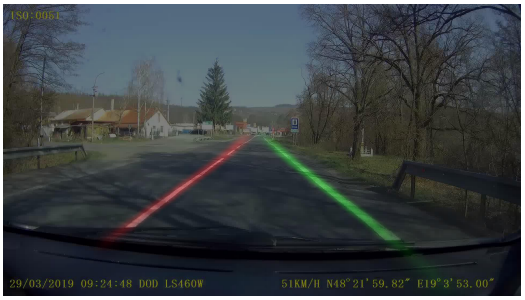
7.1 Chovanie modelu na rovných cestách

Testovanie chovania modelu na cestách s rovnými okrajmi pruhu patrilo k najzákladnejšiemu overeniu jeho funkčnosti a odhaľovaniu nedostatkov. Sledované boli prípady pruhov s plnou čiarou, prerušovanou čiarou, krajnicou, prerušovanou krajnicou a bez krajnice. Drvivá väčšina trénovacích dát pochádzala z prostredia mesta Peking, za prítomnosti výrazných krajníc a teda sa dali očakávať nedostatky pri ich neprítomnosti. Model však preukázal schopnosť detegovať okraj pruhu aj bez ich prítomnosti, avšak s menšou istotou, čo je zobrazené na obrázku 7.1. Podobne sa model správa aj pri obrubníkoch, či iných ohraničeniach jazdného pruhu.

V prítomnosti dobre vyznačených hraníc pruhu model preukázal kvalitné výsledky ich detekcie, čo je zrejmé z obrázku 7.2a. Počas testovania som ale zistil to, že umiestnenie kamery zohráva veľkú rolu pri kvalite rozpoznávania. Pri umiestnení kamery takmer zarovno s palubnou doskou je uhol medzi kamerou a vozovkou malý, čoho následkom je, že pruh sa na zábere javí výrazne širší a detekcia hlavne ľavej hranice nie je kvalitná. V trénovacej sade boli totiž zábery z kamier pod väčším uhlom, a teda v nich pozícia hranice **vedľajšieho ľavého** pruhu približne zodpovedá hranici jazdného pruhu pri použití kamery položenej na palubnej doske a smerujúcej zarovno s vozovkou. Hranica vedľajšieho pruhu bola pri trénovaní ignorovaná a to malo za následok nízku kvalitu rozpoznávania hraníc pri umiestnení kamery spomínaným spôsobom, čo je dokumentované napríklad na obrázku 7.2b.



Obr. 7.1: Záber cesty bez krajnice a predikcia modelu pre tento záber.

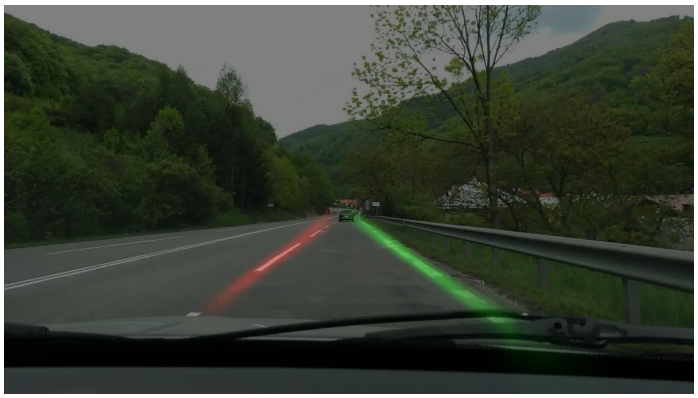


(a) Kamera umiestnená pod vhodným uhlom.

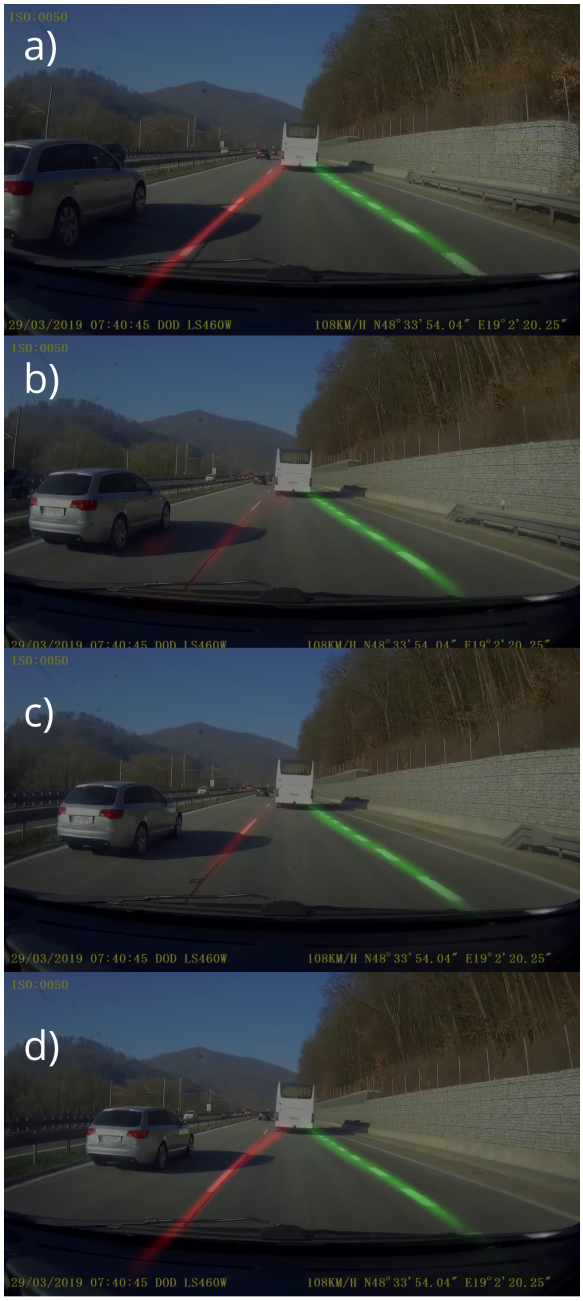
(b) Kamera umiestnená na palubnej doske pod malým uhlom k vozovke.

Obr. 7.2: Vizualizácia predikcie modelu pre snímky z rovnej cesty pri rôznych pozíciách kamery.

Predikcie modelu pre snímky zachytávajúce pruh s rôznymi prerušovanými čiarami sú zobrazené na obrázkoch 7.3 a 7.4. Je na nich vidieť, že model preukázal schopnosť poradiť si aj s prerušovanými čiarami. Zároveň však treba podotknúť, že v prípade, keď sú medzi prerušovanými čiarami na záberoch veľké medzery a auto práve prešlo cez jednu takúto čiaru, kvalita predikcie ľavej hranice výrazne klesá až kým sa nepriblíži ďalšia časť prerušovanej čiary. Spomenutý prípad je najlepšie vidieť vo videu, no obrázok 7.4 ho dobre vystihuje. V tomto obrázku je zároveň vidieť, že prerušovaný pravý okraj je dobre detegovaný v každom zábere, keďže medzery medzi jednotlivými čiarami sú menšie.



Obr. 7.3: Vizualizácia predikcie modelu na snímke s prerušovanou ľavou hranicou pruhu.



Obr. 7.4: Vizualizácia predikcie modelu na štyroch snímkach, zachytávajúcich jazdu v pruhu s prerušovanými hranicami.

7.2 Chovanie modelu v zákrutách

Zákruty sú samozrejmosťou každej cesty, respektíve komunikácie a preto som sledoval správanie modelu aj v rôznych zákrutách. Pri miernych až stredných zákrutách model dokázal detegovať hranice dobre a to aj z dôvodu, že takéto zákruty sa líšia len v malom uhle od rovnej cesty. Takúto zákrutu spolu s predikciou modelu dokumentujem na obrázku 7.5a. V záberoch s ostrými zákrutami model zlyhával pri detekcii hranice, ktorá bola na vonkajšej strane zákruty. Rovnaké chovanie bolo zaznamenané u ľavotočivých aj pravotočivých zákrut. Ukážku dokumentujem na obrázku 7.5b. Pri ostrých zákrutách vonkajšia hranica nápadne pripomína prípad z obrázku 7.2b a nastáva teda rovnaký problém. Dátová sada obsahovala iba približne 1.3% snímok, na ktorých sa nachádzali zákruty a aj to zväčša miernych a nie ostrých, čo má za následok spomínané chovanie modelu.

(a) Mierna pravotočivá zákruta. (b) Ostrá ľavotočivá zákruta.

Obr. 7.5: Vizualizácia predikcie modelu v zákrutách.

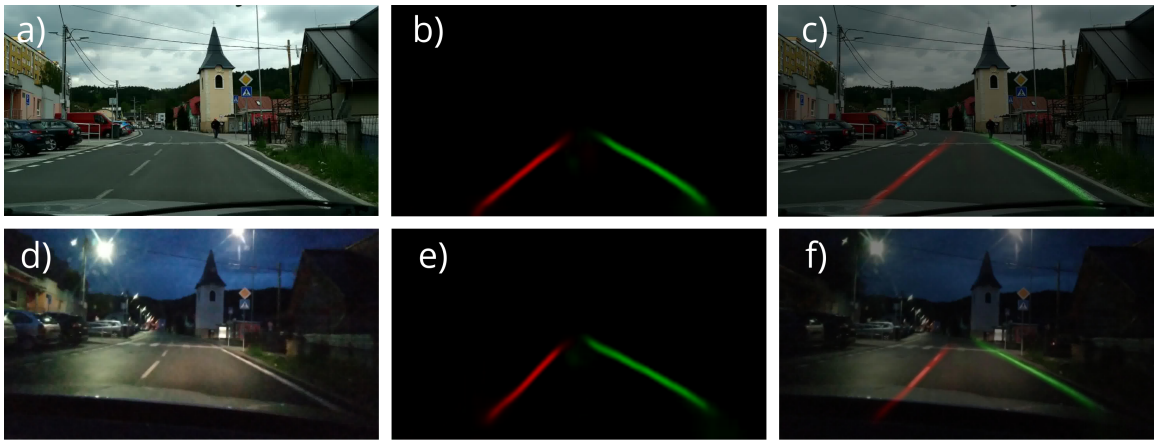
7.3 Zmena jazdného pruhu

Dôležitým kritériom pri rozpoznávaní hraníc jazdného pruhu je aj chovanie modelu pri prechode medzi jednotlivými pruhmi a jeho schopnosť detegovať správne hranice pri výskyte viacerých pruhov. Mnou natrénovaný model kvalitne udržiava jazdný pruh aj pri výskyte viacerých pruhov, ktoré sa vyskytujú napríklad pri výjazdoch alebo viacprúdovej dialnici. Keď je však reálna hranica pruhu prítomná v časti snímky, v ktorej je obvykle hranica susedného pruhu, tak nie je rozpoznaná, čo nie je ideálne. Takýto stav nastáva napríklad pri predbiehaní, kedy auto prechádza medzi jazdnými pruhmi. V tomto prípade ako postupuje smerom do stredu vozovky, pravdepodobnosti výskytu hraníc predpovedané modelom klesajú a keď je auto v strede cesty často sú až nulové. Ako auto postupuje viac do pruhu, cez ktorý ide predbiehať, pravdepodobnosti sa začínajú zvyšovať a pruh je zas detegovaný. Model v takýchto prípadoch v skutočnosti nezlyhá, akurát nedokáže určiť, v ktorom pruhu sa auto nachádza keďže je rovno na ich hranici.

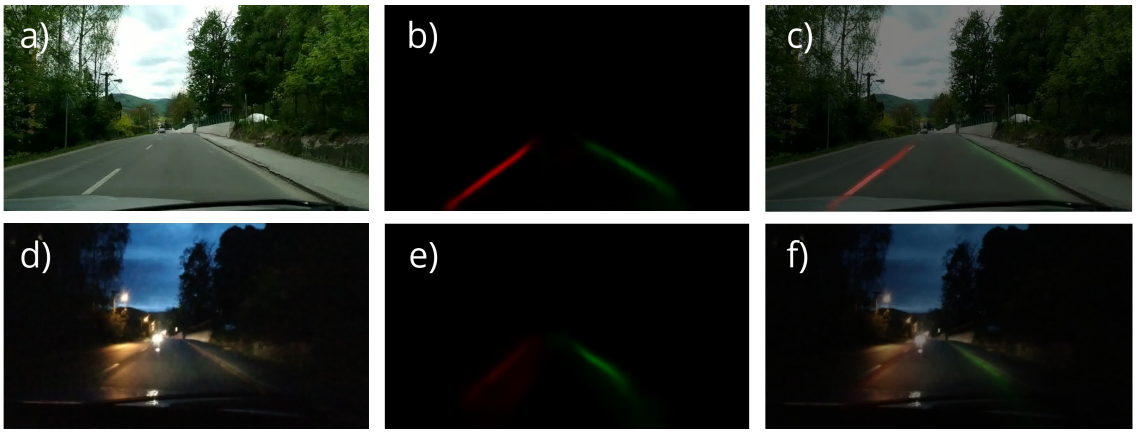
7.4 Denné a nočné zábery

Trénovacia sada obsahovala aj značné množstvo záberov nočnej scény – a preto bol model testovaný aj pri nočných snímkach. Rozhodol som sa získať záznam z palubnej kamery pri prejazde toho istého úseku za denného svetla a v noci a následne ich porovnať. Pri nočnom prejazde sa ako bonus pridal ešte aj dážď, ktorý však v týchto prípadoch nemal vplyv na rozpoznávanie.

Na obrázku 7.6 môžeme vidieť porovnanie chovania modelu pri prejazde tým istým miestom cez deň a v neskorý večer. Aj napriek zníženej viditeľnosti a odrazu svetiel lámp od mokrej vozovky mala detekcia hraníc pruhu kvalitný výsledok. Ďalší prípad môžeme vidieť na obrázku 7.7, kde je rovnako zachytený ten istý úsek cesty v priebehu dňa a v noci. Osvetlenie je tu značne slabšie oproti predošlému prípadu, čo sa podpísalo aj na predikcii modelu. Môžeme teda usúdiť, že dostatočné osvetlenie vozovky je podstatným faktorom pre kvalitnú detekciu.



Obr. 7.6: Porovnanie predikcie modelu na rovnakom úseku cesty počas dňa a cez noc. Významy jednotlivých častí sú nasledovné: a) snímka počas dňa, b) predikcia modelu pre snímku zachytenú počas dňa, c) vizualizácia predikcie na pôvodnej snímke, d) snímka v noci, e) predikcia modelu pre snímku z noci, f) vizualizácia predikcie na pôvodnej snímke z noci.



Obr. 7.7: Porovnanie predikcie modelu na rovnakom úseku cesty počas dňa a cez noc za výrazne slabšieho osvetlenia. Významy jednotlivých častí sú rovnaké ako v obrázku 7.6.

7.5 Rýchlosť spracovania a využiteľnosť

Čistá rýchlosť spracovania snímok modelom je približne 37 snímok za sekundu. Ak zarátame do procesu aj zmenšovanie snímok z originálneho *Full HD* zdroja, rýchlosť spracovania bude

približne 13.5 snímok za sekundu. Táto rýchlosť bola nameraná na procesore Intel Core i5-4210M s frekvenciou 2.6 GHz.

Model preukázal jeho schopnosť rozpoznávať hranice jazdného pruhu vozidla v rôznych situáciách. Má však aj nedostatky, ktoré už boli spomenuté v predošlých sekciách. Jeho využiteľnosť v komplexnejších systémoch je teda kvôli nim otázna, keďže isté prípady, v ktorých model nevie rozpoznať aktuálny jazdný pruh vozidla, by museli byť riešené inou časťou systému. Dokázal však, že má potenciál a vhodným doplnením dát pre dotrénovanie by sa jeho vyhodnocovanie v problémových prípadoch mohlo vylepšiť a tým aj zvýšiť jeho šanca na nasadenie do reálneho systému.

Kapitola 8

Záver

Cieľom tejto práce bolo nadobudnúť poznatky o hlbokých konvolučných neurónových sieťach a využiť ich na navrhnutie a natrénovanie modelu pre rozpoznávanie hraníc jazdného pruhu vozidla v snímkach z palubnej kamery na vhodných trénovacích dátach. Trénovacie dáta boli v rámci práce získané z viacerých zdrojov a podarilo sa natrénovať model, ktorý pri testovaní preukázal schopnosť rozpoznávania hraníc v rozličných podmienkach, čím bol cieľ práce naplnený. Pre získavanie trénovacích dát boli vytvorené nástroje, ktoré je po malých úpravách v budúcnosti možné použiť aj na vytváranie dátových sád, pre riešenie iných problémov v oblasti počítačového videnia. Trénovanie navrhnutej architektúry siete bolo úspešne implementované pomocou gradientového zostupu a jednotlivé hyperparametre jeho optimalizátora boli pri neúspešných pokusoch o natrénovanie postupne doladené. Pri riešení práce bolo potrebné naštudovať veľa informácií o problematike hlbokých konvolučných neurónových sietí a ich využití v oblasti počítačového videnia. Keďže sa jedná o pomerne rýchlo sa rozvíjajúcu oblasť, som presvedčený, že nadobudnuté vedomosti využijem aj v budúcnosti pri iných projektoch, čo beriem ako veľké pozitívum tejto práce.

Po spracovaní videozáznamov zachytávajúcich rôzne situácie v premávke sa podarilo odhaliť aj nedostatky natrénovaného modelu, ako napríklad zlú detekciu vonkajšej hranice pruhu pri ostrých zákrutách, čo vytvára priestor pre pokračovanie v projekte a jeho ďalšie zdokonaľovanie v budúcnosti. Za pokus by stálo vyskúšať jeho dotrénovanie na nových dátach, ktoré mali malé zastúpenie v pôvodnej trénovacej sade, čo by mohlo viesť k minimalizovaniu týchto nedostatkov. Ďalej by bolo možné napríklad vytvoriť jednoduchý systém, ktorý bude využívať hodnoty pravdepodobností pre výskyt hraníc pruhu produkované týmto modelom, analyzovať ich a poprípade upozorní vodiča, že prechádza cez hranicu pruhu. Systém by mohol byť postavený tak, aby pri analyzovaní pravdepodobností bral vhodne do úvahy aj predikcie z predošlých snímok, čo by ho robilo robustnejším. K tomuto systému by sa potom postupne mohli pripájať ďalšie modely, až by vznikol komplexnejší systém reálne využiteľný v praxi.

Literatúra

[1] Albawi, S.; Bayat, O.; Al-Azawi, S.; aj.: Social Touch Gesture Recognition Using Convolutional Neural Network. *Computational Intelligence And Neuroscience*, ročník 2018, 2018: str. 6973103, ISSN 1687-5273.

[2] Badrinarayanan, V.; Kendall, A.; Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, ročník 39, č. 12, 2017: s. 2481–2495.

[3] Bayar, B.; Stamm, M. C.: A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, ACM, 2016, s. 5–10.

[4] Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, Springer, 2012, s. 437–478.

[5] Clevert, D.-A.; Unterthiner, T.; Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[6] Ding, L.; Goshtasby, A.: On the Canny edge detector. *Pattern Recognition*, ročník 34, č. 3, 2001: s. 721–725.

[7] Dumoulin, V.; Visin, F.: A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[8] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[9] Hanin, B.: Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients? In *Advances in Neural Information Processing Systems*, 2018, s. 582–591.

[10] He, K.; Zhang, X.; Ren, S.; aj.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 2015, s. 1026–1034.

[11] He, K.; Zhang, X.; Ren, S.; aj.: Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, s. 770–778.

[12] Hinton, G.; Deng, L.; Yu, D.; aj.: Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, ročník 29, 2012.

[13] Hinton, G.; Srivastava, N.; Swersky, K.: Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, ročník 14, 2012.

[14] Ide, H.; Kurita, T.: Improvement of learning for CNN with ReLU activation by sparse regularization. In *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, s. 2684–2691.

[15] Karpathy, A.: CS231n Convolutional Neural Networks for Visual Recognition. https://cs231n.github.io/convolutional-networks/, navštívené 20.04.2019.

[16] Keskar, N. S.; Mudigere, D.; Nocedal, J.; aj.: On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[17] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, editácia F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, s. 1097–1105.
URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[18] Kurbiel, T.; Khaleghian, S.: Training of Deep Neural Networks based on Distance Measures using RMSProp. *arXiv preprint arXiv:1708.01911*, 2017.

[19] LeCun, Y.; Bengio, Y.; Hinton, G.: Deep learning. *nature*, ročník 521, č. 7553, 2015: str. 436.

[20] Long, J.; Shelhamer, E.; Darrell, T.: Fully Convolutional Networks for Semantic Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[21] Maas, A. L.; Hannun, A. Y.; Ng, A. Y.: Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, ročník 30, 2013, str. 3.

[22] Mukkamala, M. C.; Hein, M.: Variants of rmsprop and adagrad with logarithmic regret bounds. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, s. 2545–2553.

[23] Neven, D.; De Brabandere, B.; Georgoulis, S.; aj.: Towards end-to-end lane detection: an instance segmentation approach. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, s. 286–291.

[24] Pan, X.; Shi, J.; Luo, P.; aj.: Spatial as deep: Spatial cnn for traffic scene understanding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[25] Pérez, F.; Granger, B. E.: IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*, ročník 9, č. 3, Máj 2007: s. 21–29, ISSN 1521-9615, doi:10.1109/MCSE.2007.53.
URL https://ipython.org

[26] Riedmiller, M.; Braun, H.: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE international conference on neural networks*, ročník 1993, San Francisco, 1993, s. 586–591.

[27] Ruder, S.: An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[28] Sanger, T. D.: Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks*, ročník 2, č. 6, 1989: s. 459–473.

[29] Schmidhuber, J.: Deep learning in neural networks: An overview. *Neural networks*, ročník 61, 2015: s. 85–117.

[30] Simonyan, K.; Zisserman, A.: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; aj.: Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[32] Srivastava, N.; Hinton, G.; Krizhevsky, A.; aj.: Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, ročník 15, č. 1, 2014: s. 1929–1958.

[33] Thoma, M.: *Analysis and Optimization of Convolutional Neural Network Architectures*. Masters's thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, Jún 2017.
URL https://martin-thoma.com/msthesis/

[34] Werbos, P. J.; aj.: Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, ročník 78, č. 10, 1990: s. 1550–1560.

[35] Wu, H.; Gu, X.: Towards dropout training for convolutional neural networks. *Neural Networks*, ročník 71, 2015: s. 1–10.

[36] Yu, D.; Wang, H.; Chen, P.; aj.: Mixed pooling for convolutional neural networks. In *International Conference on Rough Sets and Knowledge Technology*, Springer, 2014, s. 364–375.

[37] Zeiler, M. D.: ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Príloha A

Obsah priloženého DVD

Súčasťou pamäťového média DVD priloženého k práci sú:

- Zdrojové súbory tejto bakalárskej práce v LaTeXe a táto práca preložená do formátu **pdf**,
- Zdrojové súbory skriptov použitých na získavanie dátovej sady, manipuláciu s dátami, trénovanie modelu a vyzualizáciu výsledkov modelu,
- **pth** súbor natrénovaného modelu,
- Obrazové dáta, na ktorých je možné vyskúšať jednotlitvé nástroje,
- Súbor README obsahujúci ďalšie informácie a pokyny,
- Krátke prezentačné video,
- Plagát.