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

HOLISTICKÉ ROZPOZNÁNÍ REGISTRAČNÍ ZNAČKY POMOCÍ KONVOLUČNÍCH NEURONOVÝCH SÍTÍ

HOLISTIC LICENSE PLATE RECOGNITION BASED ON CONVOLUTION NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DUŠAN MORBITZER**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAKUB ŠPAŇHEL**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Morbitzer Dušan**
Program: Informační technologie
Název: **Holistické rozpoznání registrační značky pomocí konvolučních neuronových sítí**
Holistic License Plate Recognition Based on Convolution Neural Networks
Kategorie: Zpracování obrazu

Zadání:

1. Zorientujte se v současných metodách rozpoznávání registračních značek (RZ) vozidel.
2. Prostudujte dostupné materiály na téma rozpoznávání pomocí konvolučních neuronových sítí.
3. Vyberte vhodnou metodu a navrhněte konvoluční neuronovou síť pro rozpoznání RZ vozidla holistickým způsobem.
4. Experimentujte s vaší implementací a návrh případně iterativně vylepšujte.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručný plakát a video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- ŠPAŇHEL Jakub, SOCHOR Jakub, JURÁNEK Roman, HEROUT Adam, MARŠÍK Lukáš a ZEMČÍK Pavel. Holistic Recognition of Low Quality License Plates by CNN using Track Annotated Data. In: *International Workshop on Traffic and Street Surveillance for Safety and Security (AVSS 2017)*. Lecce: IEEE Computer Society, 2017, s. 1-6. ISBN 978-1-5386-2939-0.
- ŠPAŇHEL Jakub, SOCHOR Jakub, JURÁNEK Roman a HEROUT Adam. Geometric Alignment by Deep Learning for Recognition of Challenging License Plates. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Lahaina, Maui: IEEE Intelligent Transportation Systems Society, 2018, s. 3524-3529. ISBN 978-1-72810-321-1. ISSN 2153-0017.
- Dále dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Špaňhel Jakub, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cílem této práce je vytvoření modelu neuronové sítě pro holistické rozpoznávání registračních značek se zaměřením na přesnost a zkrácení doby trénovacího procesu. Model byl implementován, jako spojení konvoluční neuronové sítě pro extrakci hlubokých rysů obrázku značky a Bidirectional LSTM s CTC. Natrénovaný model byl porovnán s jinou implementací, využívající holistického přístupu, která byla natrénována na stejném datasetu. Vlastní návrh sítě dosáhl lepších výsledků při rozpoznávání na datové sadě, odlišné od trénovací, s chybovostí $8,3\%$.

Abstract

The goal of this work is to create a model of neural network for holistic recognition of license plates, focused on accuracy and shortening of the learning process. The model was implemented as a union of convolutional neural network for extraction of deep features of a plate and Bidirectional LSTM with CTC. The trained model was compared to another implementation using a holistic approach, that was trained on the same dataset. My design of the network achieved better results in recognition on a dataset, which is different from the training one, with an error rate of 8.3%.

Klíčová slova

konvoluční neuronové sítě, Bidirectional LSTM, CTC, Python, Keras, TensorFlow, zpracování obrazu, rozpoznání registrační značky, deep learning

Keywords

convolutional neural networks, Bidirectional LSTM, CTC loss, Python, Keras, TensorFlow, image processing, license plate recognition, deep learning

Citace

MORBITZER, Dušan. *Holistické rozpoznání registrační značky pomocí konvolučních neuronových sítí*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jakub Špaňhel

Holistické rozpoznání registrační značky pomocí konvolučních neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Špaňhela. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Dušan Morbitzer
17. května 2022

Poděkování

Tímto chci poděkovat panu Ing. Jakubovi Špaňhelovi za odborné vedení, ochotu a čas při vytváření této práce.

Obsah

Kapitola 1

Úvod

Počítačové vidění zaznamenalo v posledním desetiletí obrovský rozvoj. Hlavními faktory jsou zlepšování výpočetní techniky podle Moorova zákonu a dostupnost většího množství kvalitnějších obrazových dat. Největší uplatnění zaznamenává počítačové vidění v oblastech dopravy, kde může sloužit například jako základ pro automatická vozidla, monitoring vozidel či chodců, detekce počtu volných parkovacích míst nebo k analýze toku dopravy. Ve zdravotnictví může počítačové vidění pomáhat při analýze snímků z rentgenu, CT, či dalších vyšetření. Dále nalézá vyskoké uplatnění ve výrobních procesech, v oblastech agrikultury i třeba u maloobchodů.

Počet dopravních prostředků stále roste a s tím i nemožnost manuálního zpracování dat, pocházejících z oblasti dopravy. Tato práce je zaměřena na rozpoznávání registračních značek vozidel. Rozpoznání registrační značky slouží k identifikaci daného vozidla a dá se tak využít například k automatickému účtování mýtného, nebo třeba identifikaci účastníků dopravní nehody. Nejpoužívanější technologií pro rozpoznání registračních značek jsou v současné době právě neuronové sítě. Jsou tak populární převážně pro jejich rychlost, přesnost, odolnost vůči okolním vlivům, jako počasí, špatné osvětlení, rozostření obrazu, nebo sklon značky, a schopnost poměrně rychlého natrénování i na menších datových sadách.

Cílem této práce je:

- Porovnat vliv hyperparametrů a augmentací na proces trénování sítě.
- Tyto poznatky aplikovat při trénování výsledného modelu.
- Výsledný model porovnat s jinou implementací holistického způsobu rozpoznávání registračních značek.

Neuronových sítí existuje více druhů a dají se kombinovat pro dosažení co nejlepšího výsledku pro daný problém. Konkrétním druhům neuronových sítí a metodám pro rozpoznávání registračních značek se věnuje kapitola č. 2. Kapitola č. 3 se věnuje výběru modelu sítě, implementaci navrženého řešení a popis a zdůvodnění změn, provedených oproti původnímu návrhu. Popis datových sad, použitých při trénování, a vyhodnocování modelů se nachází v kapitole č. 4. Nachází se zde také popis experimentů, prováděných v rámci této práce, a vyhodnocení jejich výsledků. V poslední kapitole č. 5 je závěrečné shrnutí a zhodnocení celé práce.

Kapitola 2

Umělá neuronová síť

Kapitola se věnuje základům neuronových sítí, nutných ke komplexnějšímu pochopení problému a kroků, učiněných při vytváření této práce. Oblast strojového učení a neuronových sítí je velice rozsáhlá, a proto se tato kapitola zaměřuje jen na oblasti, využité v této práci. V sekci 2.1 je obsažen popis umělého neuronu, jeho funkce a popis struktury umělé neuronové sítě. Sekce 2.2 je zaměřena na podrobný popis problematiky konvolučních neuronových sítí a její inspirace ze studia vidění u savců. Sekce 2.3 obsahuje popis rekurentních neuronových sítí a možnosti jejich využití. Vysvětlení funkce CTC Loss a její propojení s rekurentními neuronovými sítěmi se nachází v sekci 2.4. O technice trénování neuronových sítí pojednává sekce 2.5. Rozdíly mezi segmentačním a holistickým přístupem k rozpoznání obrazu popisuje sekce 2.6.

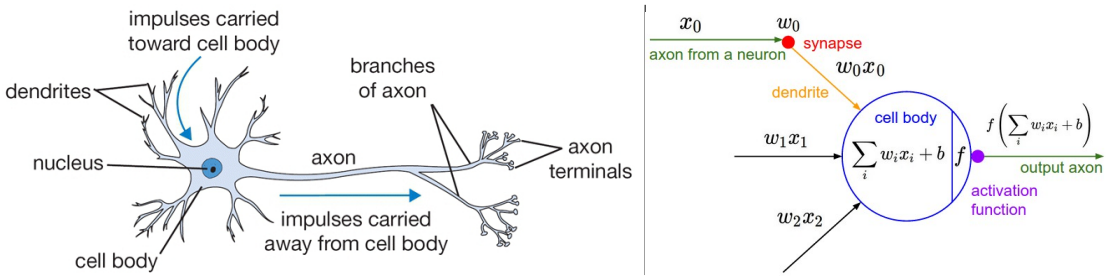
2.1 Základy neuronových sítí

Základní stavební kámen neuronových sítí je umělý neuron. Jeho podobnost s biologickým neuronem lze vidět na obrázku 2.1. Biologický neuron přijímá signály od okolních neuronů pomocí dendritů. Přijaté signály poté zpracuje podle funkce daného neuronu a zpracovaný signál předá pomocí axonů dalším neuronům. Podobnost biologického a umělého neuronu je ale jen na této základní úrovni, protože biologických neuronů je mnoho druhů různé komplexity, velikostí, tvarů a funkcí. Umělý neuron v neuronových sítích se dá představit jednoduše, jako objekt, který má svůj vnitřní stav (*váhy* a *bias*), vstupy a výstup. Jako biologický neuron, umělý neuron příjme signál (*vektor čísel*) na vstupu, ve svém těle jej zpracuje a výsledek předá na svůj výstup dalším neuronům, nebo k výsledné interpretaci. Jeho chování se dá popsat funkcí:

$$F(x) = f(\sum_i x_i w_i + b), \tag{2.1}$$

kde x značí vstupy neuronu, w je matice vah pro vstupy, b je bias a f aktivační funkce. Neurony se sdružují do jednotlivých vrstev, které se následně propojují s dalšími vrstvami tak, že výstup jedné vrstvy se předá na vstup vrstvy následující. Plně propojená vrstva je taková, kde vstupem každého neuronu aktuální vrstvy jsou vstupy všech neuronů vrstvy předcházející.

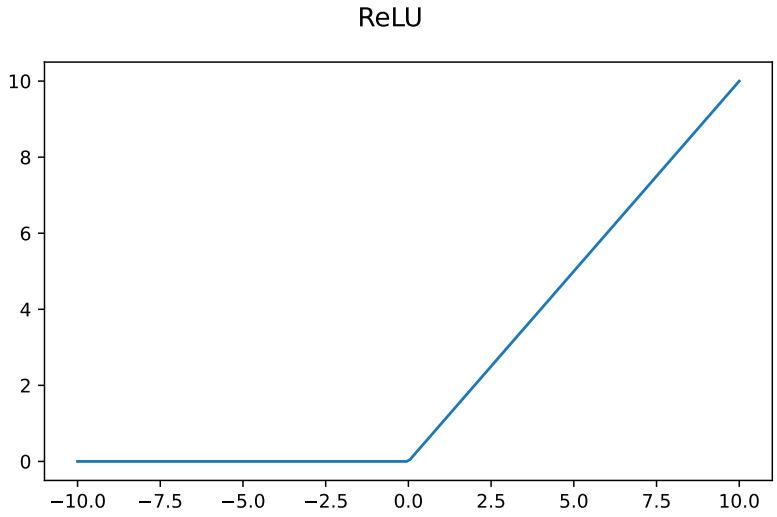
Mezi vrstvy se vkládají nelineární funkce, říká se jim aktivační funkce, které zajišťují, zda daný neuron bude aktivován nebo ne. To znamená, že aktivační funkce rozhoduje, zda je výstup neuronu potřebný k výpočtu výstupu sítě. Díky aktivačním funkcím neuronová síť



Obrázek 2.1: Demonstrace podobnosti biologického modelu (vlevo) a matematického modelu (vpravo) neuronu. Aktivační funkce umělého neuronu simuluje sílu vzruchu neuronu. Obrázky jsou přejaty z [11].

reguluje, do jaké míry se výstup každého neuronu využije. Nejhojněji využívanou aktivační funkcí je ReLU (obrázek 2.2). ReLU je funkce, která pro záporné hodnoty vrací hodnotu 0 a pro kladné hodnoty je identitou:

$$F(x) = max(0, x) \tag{2.2}$$



Obrázek 2.2: Aktivační funkce ReLU.

Zjednodušeně řečeno, neuronová síť je tedy soubor jednoduchých lineárních funkcí, sestavených hierarchicky, s nelinearitami mezi nimi, za účelem vytvoření komplexní nelineární funkce, schopné řešit námi požadované náročné úkony.

K predikci náležitosti vstupu k jednotlivým třídám se za poslední vrstvu neuronů plně propojené vrstvy vloží aktivační funkce *softmax*:

$$F(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.3}$$

Ta hodnoty predikující vrstvy konvertuje na pravděpodobnostní rozložení, kdy hodnota každého neuronu značí pravděpodobnost příslušnosti vstupu k dané kategorii. Počet kate-

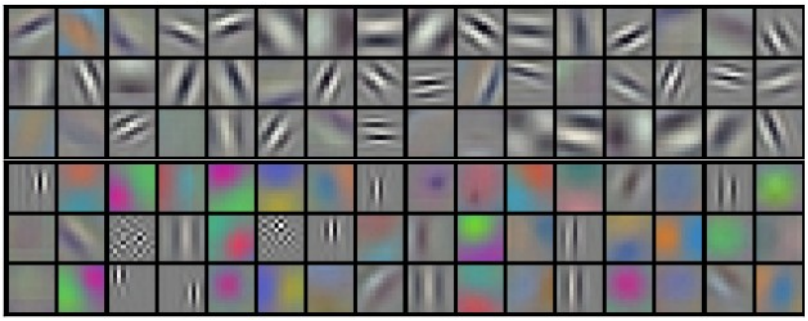
gorií, který je model schopen predikovat záleží na počtu neuronů v poslední predikující plně propojené vrstvě.

2.2 Konvoluční neuronové sítě

Průlom v oblasti studia vidění udělali v práci z roku 1959 [5] pánové Hubel a Wiesel za použití elektrofyziologie. Studovali, jak různé části kočičího mozku reagují na různé stimuly. Přišli na to, že v mozku existuje mnoho druhů neuronů, kde každý druh reaguje na jiný podnět. Nejdůležitějším objevem této studie pro počítačové vidění bylo objevení jednoduchého neuronu, který reaguje na hrany a jejich směr. Zjistili, že zpracovávání obrazu začíná jednoduchou strukturou vizuálního světa (*orientované hrany*), která se postupně sestavuje ve složitější informace až do bodu, kdy mozek rozezná komplexní objekty. Tento jednoduchý neuron a proces byl inspirací pro *konvoluční neuronové sítě*.

Konvoluční neuronové sítě využívají k přetváření vstupů ve výstup konvoluce. Protože se předpokládá, že vstupem pro konvoluční neuronovou síť jsou obrazová data, tak se můžou využívat postupy, hodící se přímo pro zpracování obrazu. Konvoluční neuronová síť se typicky skládá z konvolučních vrstev následovaných aktivační funkcí, pooling vrstev a na konci plně propojených vrstev.

Konvoluční vrstva využívá maticové filtry (jádra) pro extrakci konkrétních rysů ze vstupního obrazu. Na obrázku 2.3 (převzato z [11]) lze vidět příklady těchto filtrů. Pro každý filtr se postupně projde celý vstup, kde v každém kroku se provede skalární součin receptivního pole s filtrem. Výsledek tohoto procesu se nazývá *aktivační mapa*. Výstupem konvoluční vrstvy jsou tedy aktivační mapy odpovídající počtu filtrů konvoluční vrstvy. Jinými slovy počet filtrů odpovídá výsledné dimenzi výstupu konvoluční vrstvy.

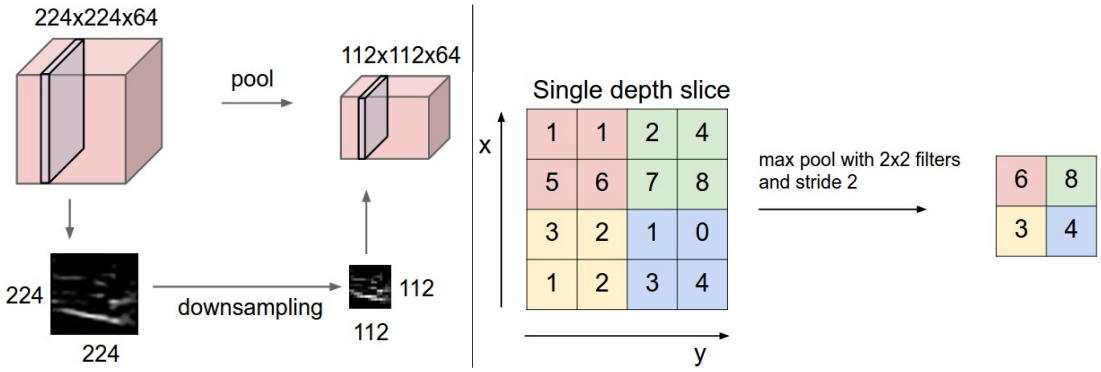


Obrázek 2.3: Příklad filtrů, naučených modelem Krizhevsky et al [7]. Filtry mají velikost $11 \times 11 \times 3$ a jsou sdíleny všemi neurony v jedné konvoluční vrstvě. Převzato z [11].

Filtry v prvních konvolučních vrstvách detekují základní/jednoduché (*low-level*) rysy, jako jsou např. hrany. Následující vrstvy už dokáží rozeznat i komplexnější (*mid-level*) rysy. Mohou mezi ně patřit rohy, skvrny, kapky a podobně. Poslední konvoluční vrstvy rozpoznají už komplexní (*high-level*) objekty, které už začínají připomínat spíše „koncepty“, než náhodné skvrny. Takový výstup je poté vstupem pro plně propojenou vrstvu, která na základě těchto rysů může predikovat kýžený výstup, typicky kategorie, do kterých vstupní obrázek spadá.

Pooling vrstvy se používají ke zmenšení rozměru (podvzorkování) vstupu, tím pádem ke zmenšení počtu parametrů sítě, ke snížení výpočetní náročnosti a napomáhají potlačit *over-*

fitting[1]. Overfitting lze řešit např. zvětšením trénovací datové sady, nebo augmentacemi[2]. Pro pooling vrstvy se typicky využívá *max pooling*, která každý region nahradí jednou maximální hodnotou z daného regionu (viz. obrázek 2.4).



Obrázek 2.4: Pooling vrstvy podvzorkují každou dimenzi vstupu nezávisle na ostatních dimenzích. Levý obrázek demonstruje podvzorkování vstupního tenzoru o počáteční velikosti $[224 \times 224 \times 64]$ pooling vrstvou s velikostí filtru 2×2 a stride 2 na výstupní tenzor o velikosti $[112 \times 112 \times 64]$. Stride určuje velikost kroku, s jakým se posouvá receptivní pole filtru nad vstupem. Obrázek napravo zobrazuje funkci nejčastěji používané *max pool* vrstvy, která pro podvzorkování využívá operaci *max*. V této ukázce je velikost filtru 2×2 a stride 2, což znamená, že na výstup se vybere maximum ze 4 hodnot každého vyznačeného pole. Převzato z [11].

2.3 Rekurentní neuronové sítě

Velikou výhodou rekurentních neuronových sítí je, že nabízí flexibilitu v návrhu architektury sítě. Obvykle s neuronovými sítěmi (obrázek 2.5) pracujeme se vstupním vektorem fixní velikosti, který je zpracován několika skrytými vrstvami a produktem je výstupní vektor také fixní velikosti (nejlevější schéma v obrázku 2.5). Zatím, co tento základní přístup neuronových sítí, kdy je pro jeden vstup produkován jeden výstup stačí pro mnoho případů, existují i úlohy, kdy je zapotřebí, aby model produkoval sekvenci výstupů, jak ukazuje schéma one-to-many v obrázku 2.5. Rekurentní neuronové sítě dovolují pracovat se sekvencemi vstupu, výstupu i obou zároveň.

- Příkladem *one-to-many* modelu je popis obrázku, kde je vstupem obrázek fixní délky a výstupem je sekvence slov, popisujících obsah vstupního obrázku (druhé schéma v obrázku 2.5). Tento přístup je využit i ve finálním návrhu této práce, kde vstupem je vektor rysů, zpracovaný konvoluční částí modelu a výstupem je popis obrázku pro predikci znaků pomocí CTC.
- Příkladem úlohy *many-to-one* je predikce založená na sekvenci snímků z videozáznamu místo z jednoho snímku (třetí schéma z leva v obrázku 2.5). Cílem této úlohy může být popis činnosti, která se ve videu vykovává. Dalším příkladem tohoto typu je určování

[1] Overfittingem nazveme stav při trénování, kdy pro trénovací data loss funkce stále klesá, ale pro validační, či testovací data tato hodnota stoupá. Síť tímto ztrácí schopnost generalizace.

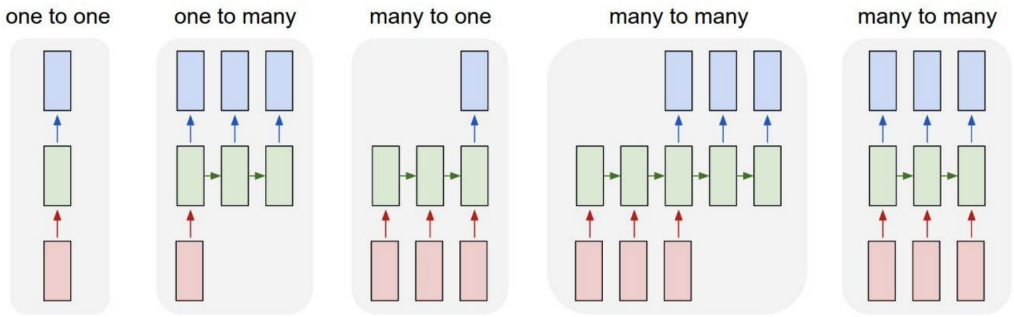
[2] Augmentace je umělá úprava vstupního obrázku (např. zoom, rotace, změna šířky/výšky, rozostření atp.).

nálady textu, kde vstupem je sekvence slov a cílem je predikce nálady sekvence (např. pozitivní nebo negativní).

- Příkladem *many-to-many* úlohy je popis videozáznamu, kde vstupem je sekvence snímků videozáznamu a výstupem je sekvence slov, popisujících děj videozáznamu (čtvrté schéma zleva v obrázku 2.5). Dalším příkladem tohoto typu je strojový překlad, kdy vstupem se sekvence slov například v angličtině a požadovaným výstupem je sekvence slov v jiném jazyce.

- Existuje i variace *many-to-many* úlohy, jak je ukázáno v posledním schématu obrázku 2.5, kde je výstup generován v každém časovém kroku. Příkladem této úlohy může být klasifikace videozáznamu na úrovni jednotlivých snímků, kde model klasifikuje každý snímek videa na základě nějakého počtu tříd. Tato predikce není založena na každém snímku zvlášť, ale i na základě všech snímků, které se objevily před snímkem aktuálním. Tento přístup je využit pro rozpoznání registrační značky v původním modelu pro tuto práci [15].

Obecně se dá tedy říci, že rekurentní neuronové sítě dovolují vytvoření architektury, kde predikce v každém časovém kroku je funkcí, která zahrnuje všechny časové kroky před aktuálním.

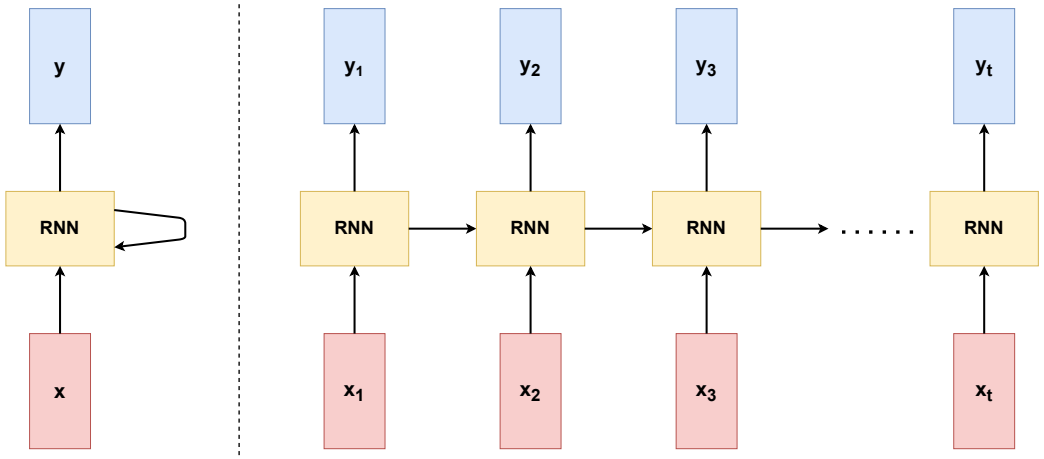


Obrázek 2.5: Schémata rozdílných typů rekurentních neuronových sítí. Červeně jsou vyznačeny vstupní vektory, zeleně jsou skryté (hidden) vrstvy a modře jsou vyznačeny výstupní vektory. Převzato z [11].

Popis rekurentní neuronové sítě (RNN)

Rekurentní neuronová síť je v podstatě černá skříňka (levá část obrázku 2.6) s vnitřním stavem, který se aktualizuje postupně při zpracovávání sekvence. V každém časovém kroku je do sítě vložen vstupní vektor, na jehož základě je modifikován vnitřní stav. V každém kroku je možnost produkovat výstup, který závisí na vnitřním stavu sítě (ilustruje obrázek 2.6).

Když model rozprostřeme v čase (pravá část obrázku 2.6), lze vidět, že síť v každém časovém kroku bere na vstupu dvě hodnoty - vstupní vektor x_i a vnitřní stav, reprezentující historii předešlých kroků - pro generování výstupu y_i a aktualizaci vnitřního stavu, který bude propagován dále. Všechny bloky v pravé části obrázku 2.6 jsou ten samý blok, sdílející stejné parametry, s jinými vstupy a historií v každém časovém kroku.



Obrázek 2.6: Vlevo se nachází zjednodušené schéma rekurentní neuronové sítě s rekurentní vazbou vnitřního stavu. V pravé části obrázku je pak toto zjednodušené schéma rozloženo v čase.

Přesněji lze rekurentní neuronovou síť reprezentovat funkcí f_W s parametry W:

$$h_t = f_W(h_{t-1}, x_t), \tag{2.4}$$

která v každém časovém kroku obdrží vektor předchozího stavu h_{t-1} předešlého kroku $t-1$ a vektor x_t k výpočtu aktuálního stavu h_t. Konstantní funkce f_W s váhami W je aplikována v každém časovém kroku nezávisle na délce vstupní či výstupní sekvence, což umožňuje uplatnění rekurentních neuronových sítí na sekvence libovolné délky.

V nejjednodušší podobě je rekurentní neuronová síť reprezentována pouze jedním skrytým stavem (dále *hidden state*) h, kde se využívá rekurentní funkce, která popisuje, jakým způsobem se modifikuje hidden state h_t na základě předchozího hidden state h_{t-1} a aktuálního vstupu x_t. Funkce se dá zapsat jako:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t), \tag{2.5}$$

kde W_{hh}, W_{hx} jsou transformační matice vah pro předchozí hidden state h_{t-1} a aktuální vstup x_t, tanh značí hyperbolický tangens a h_t označuje aktuální hidden state. Na základě aktuálního stavu h_t se můžou pomocí transformační matice vah W_{hy} vypočítat predikce pro aktuální časový krok y_t:

$$y_t = W_{hy}h_t \tag{2.6}$$

Long-Short Term Memory (LSTM)

Základní model rekurentní neuronové sítě se v praxi využívá jen zřídka, kvůli problému mizejícího gradientu (*vanishing gradient problem*). Běžně se používá rozšířený model LSTM.

Problém mizejícího gradientu

Blok RNN bere vstup x_t a předešlý hidden state h_{t-1}, učí se transformaci a poté je pomocí tanh produkována reprezentace hidden state h_t pro další časový krok a výstup y_t, jak je

ukázáno v rovnici 2.5 výše. Pro zpětnou propagaci se využívá parciálních derivací h_t podle h_{t-1}, zapsáno:

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{hx}x_t)W_{hh} \tag{2.7}$$

Váhy W_{hh} se aktualizují pomocí derivace loss v posledním časovém kroku L_t podle W_{hh}:

$$\begin{aligned} \frac{\partial L_t}{\partial W_{hh}} &= \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W_{hh}} \\ &= \frac{\partial L_t}{\partial h_t}\left(\prod_{t=2}^{T}\frac{\partial h_t}{\partial h_{t-1}}\right)\frac{\partial h_1}{\partial W_{hh}} \\ &= \frac{\partial L_t}{\partial h_t}\left(\prod_{t=2}^{T}\tanh'(W_{hh}h_{t-1} + W_{hx}x_t)W_{hh}^{T-1}\right)\frac{\partial h_1}{\partial W_{hh}} \end{aligned} \tag{2.8}$$

Jak lze vidět, $\tanh'(W_{hh}h_{t-1} + W_{hx}x_t)$ bude vždy menší, než 1, protože rozsah tanh je v intervalu $(-1,\ 1)$. To znamená, že čím větší t bude, tím více se bude gradient $(\frac{\partial L_t}{\partial W_{hh}})$ přibližovat nule. Tento jev se nazývá problém mizejícího gradientu, kde gradient pozdějších časových kroků má jen nepatrný dopad na první gradient prvních kroků. Problém nastává při delších sekvencích, protože aktualizace parametrů jsou extrémně pomalé.

Tento problém nevyřeší ani odstranění nelinearity tanh, protože pokud by hodnota W_{hh} byla větší než 1, při delších sekvencích by hrozila „exploze“ gradientu. Naopak pokud by hodnota W_{hh} byla menší, než 1, tak se nacházíme opět u problému mizejícího gradientu. Pro řešení tohoto problému bylo navrženo LSTM.

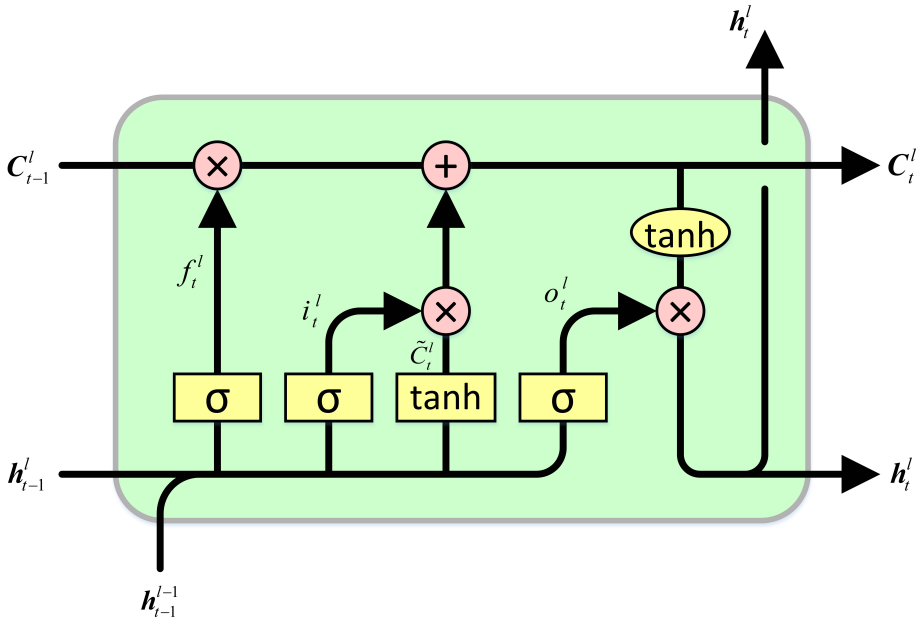
Popis LSTM

Buňka LSTM je rozšířením základního modelu RNN, která řeší problém mizejícího gradientu při zpětné propagaci novým vnitřním stavem buňky (dále *cell state*). Pro krok t existují hidden state h_t a cell state C_t, což jsou vektory o velikosti n. Do cell state C_t se ukládá dlouhodobá informace. LSTM může číst, zapisovat a mazat informace z tohoto stavu C_t. Tyto modifikace zajišťují speciální brány i, f, o, které značí „vstup“ (input), „mazání/zapomění“ (forget) a „výstup“ (output). Hodnoty těchto bran dosahují hodnot od uzavřené (0) po otevřenou (1). Všechny i, f, o brány jsou vektory o velikosti n. Schéma buňky LSTM je na obrázku 2.7.

V každém časovém kroku t LSTM vypočte nový hidden state h_t a cell state C_t na základě vstupního vektoru x_t, předchozího hidden state h_{t-1} a předchozího cell state C_{t-1} následovně:

$$\begin{aligned} f_t &= \sigma(W_{hf}h_{t-1} + W_{xf}x_t), \\ i_t &= \sigma(W_{hi}h_{t-1} + W_{xi}x_t), \\ o_t &= \sigma(W_{ho}h_{t-1} + W_{xo}x_t), \\ \tilde{C}_t &= \tanh(W_{h\tilde{C}}h_{t-1} + W_{x\tilde{C}}x_t), \\ \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \\ h_t &= o_t \odot \tanh(C_t), \end{aligned} \tag{2.9}$$

kde $\odot$ značí Hadamardův součin matic po složkách. $\tilde{C}_t$ slouží, jako výpočetní cache. Protože všechny hodnoty vektorů f, i, o jsou v intervalu $\langle 0,\ 1\rangle$, lze si povšimnout, že:



Obrázek 2.7: Schéma LSTM buňky. Lze si povšimnout, že stav C_t se mění jen na základě dvou lineárních operací (násobení a sčítání), čímž se při zpětné propagaci zamezuje problémům exploze či mizení gradientu. Obrázek převzat z příspěvku[4]ze stackoverflow.

- Brána f_t pro mazání informace v časovém kroku t kontroluje, jaké množství informace má být „vymazáno“ z předešlého cell state C_{t-1}. Tato brána se učí mazat skryté reprezentace z předchozích časových kroků, což je důvodem, proč má LSTM dvě skryté reprezentace h_t a C_t. Cell state C_t je propagován v čase a učí se, zda zapomenout předešlý cell state, nebo ne.
- Brána i_t pro vstup v časovém kroku t kontroluje, jaké množstní informace má být „přidáno“ k dalšímu cell state C_t z předchozího hidden state h_{t-1} a vstupního vektoru x_t. Tato brána rozhoduje, jaká hodnota, uložená v cache $\tilde{C}_t$, bude přidána do aktuálního cell state C_t.
- Brána o_t v časovém kroku t kontroluje, jaké množství informace se promítne do následujícího hidden state h_t.

Problém mizejícího gradientu je u LSTM řešen předáváním informace pomocí cell state C_t, kde jsou jen jednoduché lineární operace a při zpětné propagaci tak nedochází k postupné ztrátě informace.

2.4 CTC

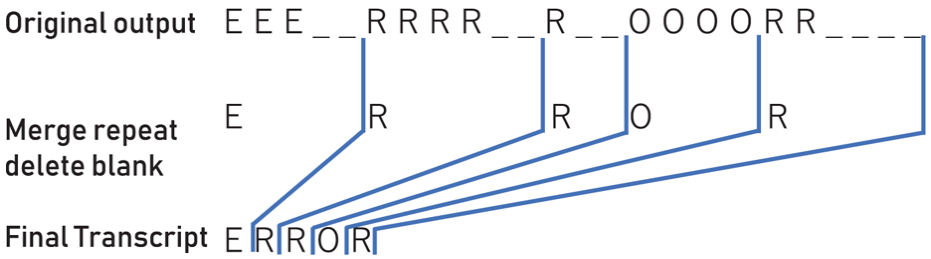
Pro transformaci výstupu z rekurentní neuronové sítě na výslednou sekvenci znaků registrační značky se používá CTC. Popis funkce CTC je převzat z práce [3]. Sítě založené na CTC se v podstatě skládají ze dvou částí - enkodér a CTC loss. Jako enkodér může sloužit jakýkoliv typ neuronové sítě, ale typicky se jedná o rekurentní neuronovou síť. CTC zastává

[4]Url schématu LSTM buňky: https://stackoverflow.com/questions/50488427/what-is-the-architecture-behind-the-keras-lstm-cell

funkci predikce (predikce znaku pro každou část vstupu) a transkripce (modifikace predikce na výslednou sekvenci).

Protože délka výsledné sekvence je kratší, než délka vstupní sekvence, metody založené na CTC zavádí prázdný symbol a tento prázdný symbol zahrnují do úvodní predikce znaků a dovolují opakování stejného symbolu v sekvenci za sebou.

Obrázek 2.8 demonstruje způsob transformace výstupu z CTC na finální řetězec znaků. CTC predikuje pro každý vzorek z enkodéru pravděpodobnost znaku (včetně prázdného znaku). Tento řetězec se následně dekóduje tak, že se sekvence po sobě opakujících stejných znaků oddělených prázdnými znaky sloučí do jednoho znaku a prázdné znaky se zahodí.



Obrázek 2.8: Obrázek demonstruje způsob transformace výstupu z CTC na finální řetězec znaků. Převzato z [3].

2.5 Trénování neuronové sítě

Klasická neuronová síť může mít miliony parametrů, které je nutné postupně vyladit tak, aby síť na svém výstupu předkládala kýžené hodnoty (přesněji *minimální odchylku* od této hodnoty). Cílem trénování neuronové sítě je nalezení optimálního nastavení všech trénovatelných parametrů sítě. Do procesu trénování zasahují také tzv. *hyperparametry*, tedy parametry, které ovlivňují průběh a vývoj trénování. Hyperparametry nastavuje sám vývojář podle toho, v jaké fázi trénování se daná síť právě nachází. Mezi hyperparametry patří například *batch size* - velikost dávky po které se přepočítávají parametry sítě.

Pro nalezení optima tak komplexní funkce, jako je neuronová síť potřebujeme znát:

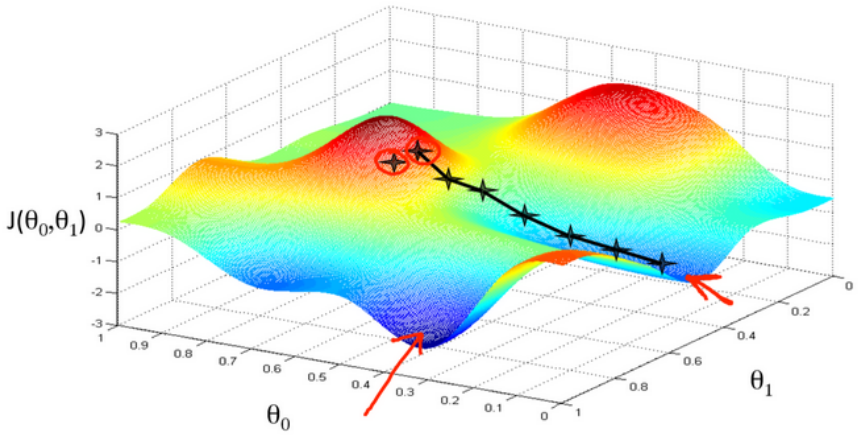
- Metriku vyhodnocení o kolik se výstup naší sítě liší od pravdy (*loss function*),
- vektor - směr, kterým se vydat (*gradient*),
- krok o kolik se daným směrem posunout (*learning rate*).

Tento proces hledání optima se nazývá *gradient-descent* (viz. obrázek 2.9).

Problematiku výpočtu odchylky výstupu od pravdy pomocí *loss function* pokrývá sekce 2.4.

Jako hledaný směr, který vede k nejprudšímu zvýšení hodnoty loss funkce se využívá vektor parciálních derivací loss funkce pro každý parametr, zvaný *gradient*. Negativní hodnota gradientu je tedy vektor nejprudšího poklesu loss funkce.

Learning rate, neboli velikost kroku, je hyperparametr, který udává rychlost učení sítě. Pokud je learning rate příliš vysoká hodnota, síť může mít problémy s nalezením optima, protože může velkým krokem optimum přeskočit. Naopak pokud je learning rate příliš malá hodnota, učení se tak mnohonásobně prodlouží. Ideální počáteční hodnota learning rate se



Obrázek 2.9: Obrázek demonstruje proces hledání optima pomocí metody gradient-descent. Můžeme vidět dvě optima této funkce, kde metoda z výchozí konfigurace parametrů postupně dospěla k lokálnímu optimu Θ_1, i když optimum Θ_0 má hodnotu loss funkce ještě nižší. Obrázek tak demonstruje, že při hledání optima záleží i na počátečním nastavení parametrů sítě. V reálu je však hodnota loss funkce závislá na milionech parametrů, takže i když tato metoda nalezne lokální optimum, tak se nijak významně neliší od optima globálního, které je s takovým počtem parametrů téměř nemožné nalézt. Obrázek převzat z článku [2].

pohybuje okolo 10^{-3}. Parametry sítě se aktualizují po proběhnutí dávky přičtením negativní hodnoty gradientu vynásobené s learning rate k stávajícím parametrům.

$$W_x = W_{x-1} - \text{gradient} \times \text{learningRate} \tag{2.10}$$

2.6 Rozdíl metod rozpoznání

Problematika rozpoznání registračních značek se řeší už dlouhou dobu. Vznikla řada rozdílných přístupů, avšak všechny tyto přístupy se dají rozdělit do dvou skupin na segmantační a holistické.

Segmentační metody spočívají v rozdělení vstupního obrazu na regiony zájmu, které se poté jeden po druhém pošlou klasifikátoru. Úkolem klasifikátoru je přiřadit vstupnímu obrazu příslušnost k nějaké kategorii. Pro rozpoznávání značek by tento postup znamenal detekování segmentů jednotlivých znaků značky a klasifikátor by poté přiřadil ke každému segmentu třídu (znak), které segment nejvíce náleží. Problémem segmentačních metod je, že pokud chceme špičkové rozpoznávací vlastnosti, tak je nutné, aby měl detektor segmentů i klasifikátor špičkovou kvalitu. Pokud by jedna z těchto složek nebyla dostatečně přesná, celý model pro rozpoznání nebude podávat kvalitní výsledky.

Holistické metody naopak mají za úkol z původní reprezentace vstupu vyvodit výsledek, bez členění vstupního obrázku na menší oblasti zájmu. Pro rozpoznávání registračních značek to znamená, že vstupem těchto metod je obrázek registrační značky, pro který po zpracování modelem pro rozpoznání, bude výstupem rovnou sekvence znaků na značce. Výhodou této metody je, že není nutná segmentace znaků zvlášť, tudíž se nemůže stát, že bude znak chybně oříznut.

Kapitola 3

Návrh a implementace řešení

Tato kapitola popisuje proces od výběru modelu z vědeckých publikací po implementační detaily zvoleného modelu. V sekci 3.1 je popsán rozbor článků, které přispěly k výběru modelu sítě a dořešení implementačních detailů a popis architektury vybraného modelu sítě. Sekce 3.2 obsahuje informace o změnách architektury modelu oproti vybranému modelu z práce [15]. Poslední sekce této kapitoly 3.3 obsahuje popis použitých nástrojů a knihoven, použitých pro řešení problému, popis načítání datasetů a prováděných augmentací a přiblížení problému rozpoznávání registračních značek různé délky a jeho řešení v této práci.

3.1 Výběr modelu

Pro rozpoznání registračních značek holistickým způsobem jsem hledal inspiraci v odborných pracích i na internetu. Dovolím si přiblížit prameny, které měly na výsledku této práce největší podíl.

První prací, která rozhodně stojí za zmínku je práce *Holistic Recognition of Low Quality License Plates by CNN using Track Annotated Data* [10]. V této práci je pro rozpoznávání registračních značek použita konvoluční neuronová síť, kde poslední vrstva pro predikci náležitosti ke třídě znaku je rozdělena na fixní počet plně propojených vrstev, kde každá vrstva rozpoznává znak na jedné pozici. Maximální počet rozpoznaných znaků odpovídá počtu predikujících plně propojených vrstev. Součástí této práce bylo i vytvoření datasetu ReId, který jsem využil pro trénování sítě v mojí práci. Také jsem zvolil výsledky, dosažené v této práci na datasetech ReId a HDR, pro porovnání s mojí implementací.

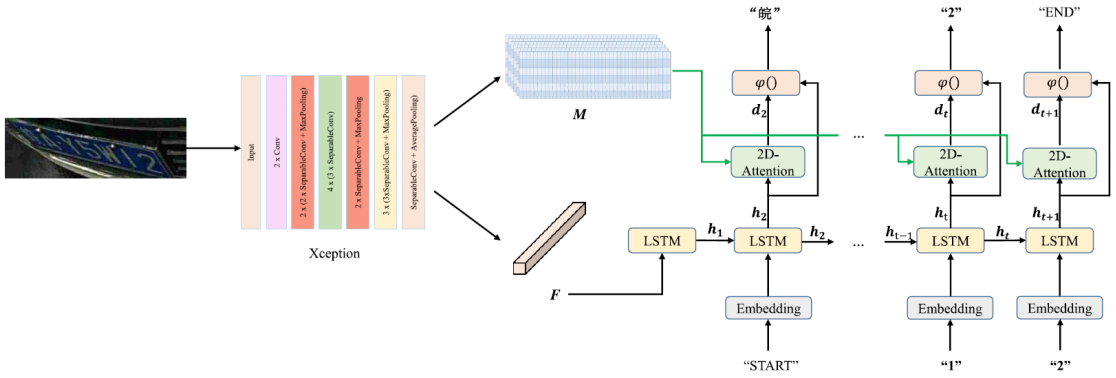
Práce [9] se zaměřuje na prolamování captcha kódů holistickým způsobem. Původně mi tato práce měla sloužit pouze pro vyzkoušení dvou holistických přístupů pro rozpoznávání sekvencí číslic k pohovoru na machine learning pozici. Avšak po neúspěchu při implementaci původně zvoleného modelu sítě jsem se rozhodl využít poznatků z mojí prvotní implementace této práce a zvolil jsem pro moji práci jako dekódovací část architektury spojení Bidirectional LSTM a CTC z této práce. Také jsem z této práce využil funkce pro vyhodnocování výsledků sítě. Více je přínos této práce popsán v sekci změn 3.2.

Hlavní článek pro tuto práci

Největší přínos pro tuto práci přinesla práce publikovaná v listopadu 2021 [15]. Za touto prací stojí tým Linjiang Zhang, Peng Wang, Hui Li, and Zhen Li, Chunhua Shen a Yanning Zhang. Jejich práce *A Robust Attentional Framework for License Plate Recognition in the Wild* se zaměřuje na holistické rozpoznání registračních značek v „divočině“, což znamená za

velmi ztížených podmínek. Ztíženými podmínkami je myšleno špatná viditelnost, extrémní zkosení registrační značky, překryv části znaků značky, extrémní temno i světlo, rozostření způsobené pohybem čí okolními podmínkami, jako například počasím. Pro vypořádání se s těmito podmínkami model využívá mechanizmus *2D-attention*, který je popsán níže v této sekci.

V článku tento model využívá k trénování data generovaná pomocí upraveného modelu *CycleGAN*. Tímto způsobem je možné se vyhnout namáhavé práci manuální anotace dat a obdržet obrovské množství trénovacích dat s vybalancovanou distribucí znaků na značce a za různých podmínek, což velice kladně napomáhá k posílení přesnosti rozpoznávání. Model, trénovaný tímto způsobem, dosáhl vynikajících výsledků na čtyřech veřejných datasetech, což demonstruje generalizaci a robustnost výsledného modelu. Na obrázku 3.1 lze vidět přehled navržené architektury pro rozpoznání registraček v extrémních podmínkách.



Obrázek 3.1: Přehled navržené architektury pro rozpoznání registraček v extrémních podmínkách. Obrázek registrační značky je vstupem pro 30-ti vrstvou konvoluční síť *Xception* za účelem obdržení globálního feature vektoru (označen písmenem F). LSTM model je adaptován pro dekódování tohoto vektoru na znaky z registrační značky. Z 12. vrstvy je extrahována feature mapa (označena písmenem M), která slouží jako lokální kontext pro rozpoznávání jednotlivých znaků. Převzato z [15].

Architektura modelu

Jak lze vidět na obrázku 3.1, model se skládá ze dvou částí: upravený model Xception pro extrakci rysů a LSTM založena na mechanismu 2D-attention pro dekódování jednotlivých znaků registrační značky.

Konvoluční část modelu slouží jako enkodér vstupní reprezentace obrázku. Skládá se ze 30-ti vrstvé Xception, která je upravená z originální Xception [4], jejíž detail popisuje obrázek 3.3. Všech 30 vrstev je strukturováno do 9 modulů s residuální vazbou, kromě prvního a posledního modulu. Termín „ResSeparableConv“ označuje blok tří separable konvolucí s identickou residuální vazbou.

Úkolem *entry flow* je snížení rozměrů z 160×48 na 40×6 a zvýšení počtu dimenzí ze 3 na 256 za použití separable konvolucí a max pooling vrstev. *Middle flow* se skládá ze 4 stejných ResSeparableConv bloků, za účelem extrakce hlubokých rysů, které obsahují více high-level reprezentace. V middle flow je rozměr i počet dimenzí neměnný. *Exit flow* má za úkol extrahovat feature mapu M („mapu rysů“) o velikosti $40 \times 6 \times 512$, která slouží

jako lokální kontext pro rozpoznání jednotlivých znaků pomocí 2D-attention mechanismu a výsledný feature vektor („vektor rysů“) F s velikostí 512 dimenzí.

Rekurentní část modelu slouží jako dekodér pro převedení výstupní reprezentace enkodéru na sekvenci znaků registrační značky. Rekurentní neuronové sítě (RNN) jsou hojně využívané pro překlady, popisování obrázků a rozpoznávání textu ve scéně. Zde je RNN využita pro rozpoznávání registračních značek. Díky integrovanému 2D-attention mechanismu není za potřebí upravovat nepravidelné registrační značky, nebo používat segmentaci znaků. Navržený model díky tomuto mechanismu zvládne rozpoznat registrační značky různých tvarů i extrémního zkosení.

V modelu se využívají 2 vrstvy LSTM s hidden state o velikosti 512. První vrstva se dá počítat ještě do enkodéru, protože ta má za úkol v časovém kroku 0 zpracovat feature vektor F a svůj hidden state a cell state předat, jako počáteční stav druhé vrstvy LSTM pro poskytnutí celkového přehledu o vstupním obrázku. Poté je druhé vrstvě LSTM v časovém kroku 1 vložen *START* token. Od kroku 2 se výstup předešlého časového kroku používá jako vstup pro následující krok, dokud není predikován token *END*. Pro vstup dekódovací vrstvy LSTM se využívá embedingu v podobě one-hot-encoding. Výpočet tohoto modelu LSTM při trénování lze vyjádřit vztahem:

$$h_{t+1} = f(h_t, \psi(x_t)), \qquad t = 1, ..., 8. \tag{3.1}$$

V rovnici h_t značí aktuální hidden state, $f()$ značí operaci LSTM v každém časovém kroku a $\psi(\cdot)$ je embedding operace. V části predikce znaku, $x_t = \text{softmax}(\phi(h_t, d_t))$ je aktuální výstup časového kroku, pokud se síť nachází ve stavu trénování, přímo predikovaný znak x_t. $\phi(\cdot)$ je lineární transformace a d_t je výstup 2D-attention modulu. Výpočet 2D-attention je následující:

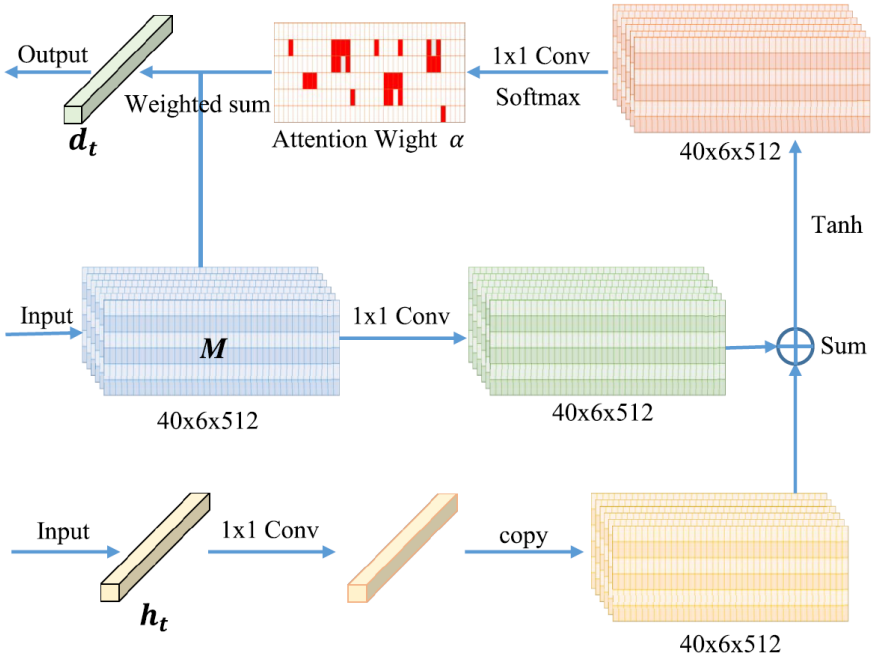
$$\begin{cases} g_{ij} = \tanh(W_m M_{ij} + W_h h_t), \\ \alpha_{ij} = \text{softmax}(W_g \cdot g_{ij}), \\ d_t = \sum_{i=1}^{H} \sum_{j=1}^{W} \alpha_{ij} M_{ij} \end{cases} \tag{3.2}$$

V rovnici 3.2 je M_{ij} feature vektor na pozici i, j feature mapy M a h_t je hidden state LSTM v časovém ktoku t. W_m, W_h, W_g jsou matice lineární transformace k naučení. α_{ij} je váha attention na pozici i, j. d_t je vážená suma feature mapy M pro lokální kontext v časovém kroku t. Schéma 2D-attention mechanismu ilustruje obrázek 3.2.

3.2 Změny oproti vybranému modelu

Zvolený model pro rozpoznávání registračních značek z práce [15] se mi podařilo úspěšně implementovat a sestavit, ale nepovedlo se jej natrénovat kvůli neustálým chybovým hláškám o špatném počtu dimenzí. Tyto chyby jsem se snažil opravit více než měsíc, ale bohužel bezúspěšně a to mě dovedlo k provedení změn v architektuře sítě. Jelikož jsem měl to štěstí a při jednom přijímacím pohovoru jsem vytvářel model pro rozpoznání tištěných čísel, kde jsem narazil na práci [9], kde se využívá spojení Bidirectional LSTM a CTC, zvolil jsem tento postup i pro řešení problému rozpoznání registračních značek.

Při vytváření modelu pro rozpoznání tištěných čísel jsem přišel na některé způsoby práce s daty, jako rozpoznání sekvencí různé délky a narazil na několik úskalí, jako neschopnost rozpoznání sekvencí s velkým sklonem, které jsem napravil při vypracovávání této práce. Problém rozpoznání sekvencí s větším sklonem bylo, že jsem měl nastavenou délku výstupu vrstvy pro predikci pomocí CTC na 20 znaků při maximální délce sekvence 10 číslic. Při



Obrázek 3.2: Schéma 2D-attention mechanismu. M je feature mapa obdržená z Xception (ukázáno v obrázku 3.1) a h_t je hidden state LSTM pro každý časový krok dekodovacího procesu. Převzato z [15].

větším sklonu se stávalo, že některá čísla byla úplně přeskočena. Proto jsem se rozhodl, že pro tuto práci s maximální délkou značky 10 znaků nastavím velikost vrstvy pro predicki pomocí CTC na dvojnásobek, čili 40. Kombinace Bidirectional LSTM a CTC prokázala skvělé vlastnosti při přesnosti rozpoznání a generalizace, a proto jsem jako dekodér v této práci zvolil právě tuto kombinaci.

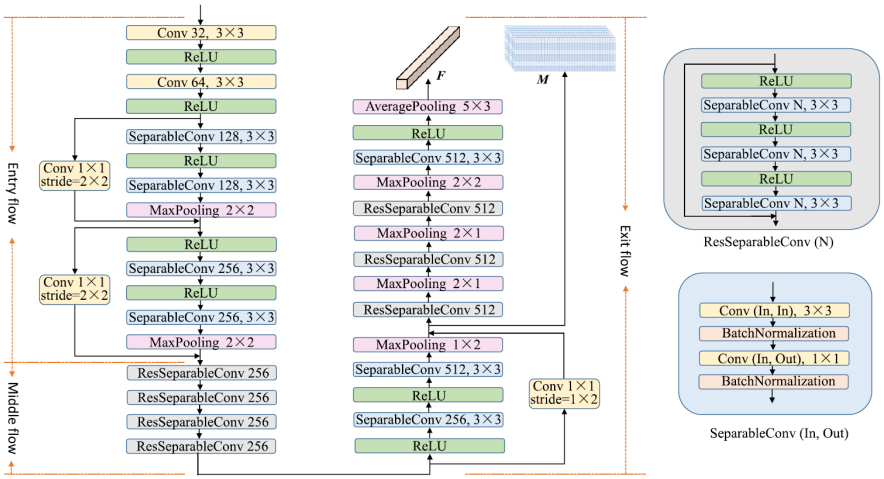
Konvoluční část modelu jsem chtěl ponechat s co nejmenšími změnami oproti zvolenému modelu. Obrázek 3.4 ilustruje konvoluční část mého modelu a obrázek 3.3 konvoluční část modelu zvoleného. Provedené změny se týkají pouze části *exit flow*. Z exit flow je vyjmuta část bloku, kde probíhá max pooling, protože v části, kde se extrahuje feature mapa M je velikost první dimenze tenzoru požadovaných 40. Pro zmenšení druhé dimenze z 6 na 2 jsem nastavil velikost poslední *AveragePooling* vrstvy na 1×3. K implementaci konvoluční části modelu jsem upravil originální kód implementace Xception pomocí knihovny Keras [8] podle architektury obrázku 3.4.

Dekódovací část modelu se skládá ze spojení Bidirectional LSTM a CTC. Pro lepší rozpoznávací schopnosti sítě jsem zvolil 2 vrstvy Bidirectional LSTM s velikostí hidden state 512, jako v původním návrhu modelu. Vstupem pro rekurentní část je výstupní reprezentace obrázku z konvoluční vrstvy o rozměrech $[40 \times 2 \times 512]$, která byla pomocí vrstvy `TimeDistributed` transformována na rozměr $[40 \times 1024]$, který je zpracovatelný pomocí LSTM. LSTM požaduje vstup ve formě [počet časových kroků × počet vzorků].

Výstup této rekurentní částu modelu slouží, jako vstup pro predikující plně propojenou vrstvu s aktivační funkcí softmax. Počet predikovaných kategorií je rozšířen o prázdný znak, potřebný pro zpracování predikce na požadovanou sekvenci znaků registrační značky pomocí CTC. Touto vrstvou končí základní model sítě, který slouží pro predikce výstupu. CTC predikuje i prázdné znaky, o které je popis každé registrační značky rozšířen. Model

pro trénování sítě je rozšířen o vstupy, potřebné k výpočtu CTC loss (label, velikost predikovaného vstupu, délka labelu). Vstupy pro CTC loss jsou vloženy do `Lambda` vrstvy, která k výpočtu hodnoty loss využívá implementaci `ctc_batch_cost` z backendu Kerasu.

Celá dekódovací část je upravena z původního kódu pro rozpoznání captcha kódů z práce [9]. Přejata byla také funkce pro výpočet přesnosti rozpoznání sítě a třída `Evaluate`, která je callbackem při trénování pro průběžný výpočet přesnosti modelu na validačních datech a uložení této hodnoty do historie trénování.



Obrázek 3.3: Schéma konvoluční části upravené Xception z článku [4]. „Conv“ značí konvoluční vrstvu s počtem výstupních kanálů a rozměry jádra (filtru), jak je vyznačeno. Stride a padding pro konvoluční vrstvy je nastaven na 1 a pro max-pool vrstvy se padding nepoužívá. Převzato z [15].

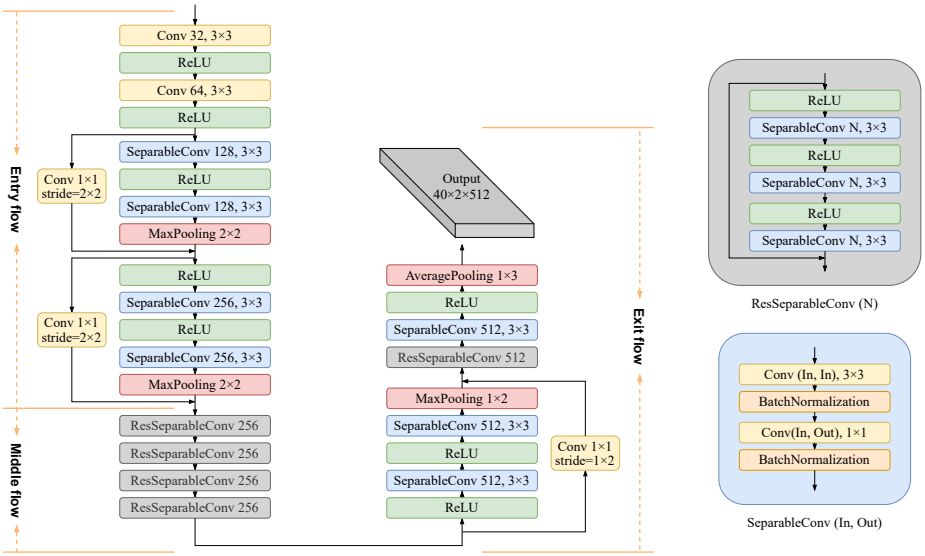
3.3 Implementace řešení

Pro implementaci vybrané sítě jsem zvolil jazyk *Python*, kvůli jeho podpoře pro strojové učení a množství knihoven jej podporující. Velikou výhodou Pythonu, alespoň pro mě, je možnost využití *Jupyter Notebooks* a s tím výhody interaktivního Pythonu. Členění kódu do sémanticky významných celků a možnost využití markdown buňky pro popis velice přispívá přehlednosti kódu a napomáhá rychlejšímu pochopení, hlavně pro učení nových témat. Největší předností Jupyter Notebooks je ale možnost spouštění jednotlivých bloků kódu samostatně a udržování vnitřního stavu proměnných. Tím pádem například pro vyzkoušení, zda nějaká malá úprava funguje správně, není za potřebí spouštět celý, potenciálně dlouho se vykonávající, kód, ale stačí si ji vyzkoušet v příslušném bloku.

Jako vývojové prostředí jsem zvolil *VSCode*, kvůli podpoře Jupyter Notebooks, pro jeho Python rozšíření k usnadnění práce při programování a protože mi vyhovuje propojení vývojového prostředí s terminálem. Také jsem měl už s vývojem ve VSCode předchozí zkušenosti a pracovalo se mi s tímto prostředím velice dobře.

Použité knihovny

Knihovny obecně dokáží značně ulehčit práci v jakémkoliv jazyce. Jedná se o soubor tříd, funkcí i konstant, které jsou ověřené a dají se využít v jakémkoliv vyvíjeném programu pou-



Obrázek 3.4: Schéma konvoluční části mojí implementace upravené Xception z originální implementace Xception pomocí knihovny Keras [8]. „Conv“ značí konvoluční vrstvu s počtem výstupních kanálů a rozměry jádra (filtru), jak je vyznačeno. Padding pro konvoluční vrstvy je nastaven na hodnotu „same“ a pro max-pool vrstvy se padding nepoužívá.

ným importováním. Pokud programátor chce vyvíjet program, který se věnuje určité oblasti pro kterou je už vytvořena některá knihovna, znamená to, že programátor nemusí začínat s vývojem programu či aplikace od nuly, ale může využít funkčnosti knihoven a zaměřit se více na svoji oblast problému. Například u této práce jsem nemusel vyvíjet neuronovou síť od základů a využitím knihoven jsem se mohl plně soustředit na problém rozpoznání registračních značek.

Keras

Díky propracovaným knihovnám Pythonu pro práci s neuronovými sítěmi lze svůj první model sestavit, natrénovat i otestovat už na pár řádcích kódu. Po prostudování deep learning frameworků *PyTorch* a *Tensorflow* jsem došel k závěru, že úvod do světa deep learningu mi více usnadní TensorFlow, přesněji Keras, který používá Tensorflow jako svůj backend, pro jeho uživatelsky přívětivý high-level přístup k programování neuronových sítí. Díky modularitě Kerasu se stávající architektura sítě snadno rozšiřuje, nebo rovnou předělává na řešení problému jiným způsobem, což jsem si osobně vyzkoušel a popis tohoto problému se nachází v secki 3.2.

V této práci využívám Keras pro generování vstupních dat z datasetu, sestavení modelu sítě a jeho trénování. Při trénování využívám funkce modulu `tensorflow.keras.callbacks` pro přidání dodatečné funkčnosti do trénovacího procesu. `EarlyStopping` zajišťuje, že se trénování ukončí, pokud hodnota loss funkce (implicitně nad validačními daty) nezaznamená pokles v posledních x epochách, nastavitelných parametrem `patience`. `CSVLogger` zaznamenává historii vývoje sledovaných metrik při tréninku sítě. `ModelCheckpoint` ukládá nastavení vah sítě do souboru po epochách. V této práci jsem využil přepínače této funkce pro ukládání pouze nastavení vah s nejnižší hodnotou loss funkce nad validačními daty.

`Callback` je abstraktní třída, která je využita pro vytvoření třídy `Evaluate`, která má za úkol vyhodnotit, vždy po skončení epochy, přesnost predikce sítě nad validačními daty.

TensorFlow

TensorFlow je end-to-end open source platforma pro strojové učení. Má rozsáhlou a flexibilní nástrojovou sadu, knihovny a komunitu, která napomáhá programátorům ulehčit sestavení, vývoj a nasazení aplikací strojového učení. TensorFlow byl vytvořen týmem Google Brain a slouží pro numerické výpočty. Pro uživatelsky přívětivý vývoj a sestavování aplikací, používá TensorFlow jako front-end Python, zatímco tyto aplikace spouští ve výkonném C++ [14]. TensorFlow pracuje na nižší úrovni, než Keras, takže má přístup přímo k datům, se kterými vrstvy v Kerasu pouze manipulují, a může tak přímo data sledovat, upravovat a provádět s daty základní numerické operace, ke kterým Keras jinak nemá přístup.

V této práci Tensorflow využívám ve třídě na předzpracování vstupních dat pro úpravu rozměrů obrázků na velikost, požadovanou strukturou sítě, protože chci síti předávat data o kterých vím, jak přesně vypadají a nenechávat to na vnitřní funkcionalitě `Resizing` vrstvy Kerasu.

NumPy

NumPy je knihovna Pythonu, používaná pro práci nad poli. Také obsahuje funkce pro práci s lineární algebrou, fourierovými transformacemi i maticovými operacemi. NumPy je open source projekt, založený roku 2005 Travisem Oliphantem. NumPy je zkratka pro *Numerical Python* [13]. NumPy vzniklo na popud toho, že práce s velkým množstvím dat v Pythonu byla pomalá a výpočetně náročná, protože Python zachází ve své vnitřní reprezentaci se všemi entitami aplikace (konstanty, proměnné, funkce..) jako s objekty. Tyto objekty mají mnoho výhod, ale za cenu větší režie a paměťové náročnosti. Pojem pole, jak je znám z jazyka C, kde jsou konstanty umístěny v paměti jedna za druhou, v Pythonu neexistoval. Python reprezentace pole je seznam, kde jsou uloženy odkazy na Python objekty, tím pádem data nejsou v souvislém bloku, ale přistupuje se do různých částí paměti skrze odkazy.

NumPy reprezentace pole `ndarray` má reprezentaci právě z jazyka C, tím pádem je mnohem paměťově úspornější a data jsou v paměti uloženy v jednom souvislém bloku a tím pádem lze k výpočtům využívat efektivnější výpočetní operace, které jsou optimalizovány pro CPU. NumPy je částečně implementováno v Pythonu, ale výpočetní operace jsou implementovány a optimalizovány v jazyce C/C++.

V této práci jsem NumPy využil při načítání datasetu po dávkách. Při práci s tak velkým množstvím dat, jako jsou datasety pro trénování a testování se musí používat optimalizované knihovny, jako NumPy a TensorFlow, protože s těmito strukturami počítají právě deep learning knihovny.

Pandas

Pandas je open source balíček Pythonu, který je využíván k data science, datové analýze a k úlohám pro strojové učení. Pandas je postaveno nad NumPy (zmíněno v sekci výše), které poskytuje podporu pro práci s multidimenzionálními poli [1]. Pandas je optimalizováno pro práci s daty, jako jsou tabulky. Pandas poskytuje obrovskou sadu operací na provádění statistik, analýz, agregaci, úpravu i vykreslování dat. Je to komplexní nástroj pro práci s daty s obrovskou komunitou a přehlednou dokumentací, takže téměř každou manipulaci s daty pomocí Pandas zvládne i nezkušený programátor.

V této práci využívám Pandas k analýze dat o datasetech a zpracování výsledků trénovacích procesů. `ImageDataGenerator` získá data pro načtení datasetů právě pomocí `pandas.DataFrame`. Pandas jsem využil ve spojení s modulem `matplotlib.pyplot` (pokryto v následující sekci) pro vizualizaci všech průběhů trénování a statistických analýz.

Matplotlib/Pyplot

Matplotlib je multiplatformní knihovna pro vizualizaci dat a grafů pro Python. Matplotlib vznikl původně, jako open source alternativa k MATLABu. Výhodou matplotlibu je možnost tvorby grafů mnoha typů pomocí několika málo řádků kódu. Dokáže generovat kvalitní výstup do nejběžnějších rastrových formátů (JPG, PNG) i do vektorových formátů (SVG, PS, PDF). Integruje interaktivní GUI pro základní inspekci dat a využívá NumPy operací pro zvýšení efektivity některých operací. Velkou výhodou je i nativní podpora v Jupyter Notebooks (IPython). Matplotlib podporuje jak procedurální přístup programování podobné MATLABu, tak objektově orientovaný přístup, který využívá OO návrhu knihovny Matplotlib. Tento přístup dovoluje vyšší míru kontroly nad výstupem.

V této práci Matplotlib využívám pro generování všech grafů, a vizualizaci výstupů sítě, které lze vidět v této zprávě. Napomáhá mi tak k analýze dat i trénovacího procesu. Bez vizualizace by bylo mnohem obtížnější porozumět výstupním datům a snadno by se dalo přehlédnout některé souvislosti.

Načítání datasetů

Oba datasety obsahují mimo obrázků registračních značek také soubor `trainVal.csv`, obsahující metadata o všech obrázcích značek v datasetu. Soubor se skládá ze 4 sloupců, kde první dva sloupce `track_id` a `image_path` obsahují stejná data a to cestu k danému obrázku značky. Třetí sloupec `lp` obsahuje textový přepis znaků na značce. Poslední sloupec `train` je příznak, značící informaci, zda má být daná značka použita pro trénování, nebo testování sítě. Z CSV souboru se vytvoří `pandas.DataFrame`, který je následně rozdělen na 2 dataframy. Jeden obsahuje pouze trénovací data a druhý pouze data testovací. Pro převedení obrázků na vhodnou reprezentaci jsem zvolil `ImageDataGenerator` z balíčku `tensorflow.keras.preprocessing.image`. Generátor zajišťuje normalizaci vstupní reprezentace obrázků (převedení hodnoty pixelů z intervalu $\langle 0,\ 255\rangle$ do intervalu $\langle 0,\ 1\rangle$). U trénovacího datasetu zajišťuje navíc augmentace a rozdělení dat na trénovací a validační. Nastavení generátoru zobrazuje výpis kódu 3.1:

```
datagen = ImageDataGenerator(rescale=1./255.,
                             rotation_range=30,
                             zoom_range=0.2,
                             width_shift_range=0.15,
                             height_shift_range=0.15,
                             shear_range=0.15,
                             channel_shift_range=0.25,
                             fill_mode="nearest",
                             validation_split=0.1)
```

Výpis 3.1: Ukázka kódu pro vytvoření generátoru trénovacích a validačních dat z datasetu. Kód zde slouží pro demonstraci nastavení augmentací generátoru.

Rozpoznání značek různé velikosti

Model je implementován pro rozpoznávání registračních značek různé velikosti. U segmentačních způsobů rozeznávání variabilní délka registrační značky není problém, ale pro holistické způsoby tato úloha vyžaduje specifičtější řešení. Rozdíly segmentačních a holistických způsobů rozpoznávání jsou popsány v sekci 2.6. Protože predikující vrstva sítě má konstantní počet neuronů, ale počet predikovaných znaků je variabilní, bylo za potřebí variabilitu eliminovat. Toho jsem docílil přidáním tzv. „prázdného znaku“ do kategorií pro rozpoznávání a nastavením maximálního počtu rozpoznaných znaků. V této práci jsem zvolil maximální počet znaků na 10, protože větší počet znaků se na registračních značkách typicky neobjevuje (v Indii jsou zregistrační značky o 10 znacích běžné). Zástupným znakem pro prázdné místo se doplní popisky registračních značek na zadanou délku 10 znaků (např. 8B92022 - - -). Zástupný znak pro prázdné místo využívá k dekódování sekvence predikovaných znaků sítí i CTC Loss. Funkce CTC Loss je popsána v sekci 2.4.

CTC Loss vyžaduje pro výpočet hodnoty loss a pro dekódování predikovaného výstupu sítě další parametry, kromě výstupu ze sítě. Funkci CTC Loss je za potřebí ke každému vzorku z dávky dodat:

- Zakódovaný přepis textu registrační značky (v práci využit pro každý znak jeho index v seznamu použitých znaků).
- Délka predikovaného výstupu sítě (v práci 40, kvůli lepší predikci znaků u zkosených značek).
- Délka přepisu textu registrační značky (maximální počet predikovaných znaků sítě).

K doplnění těchto informací pro CTC Loss, ke změně velikosti obrázků na velikost požadovanou sítí a pro doplnění přepisů značky o zástupné znaky pro prázdné místo slouží třída `LPSequence`. Inspirace pro tuto třídu pochází z práce [9]. Třída dědí ze základního objektu pro obalení dat, jako jsou právě data z datasetu, `Sequence` modulu `tensorflow.keras.utils`. Každá `Sequence` musí implementovat metody `__getitem__` a `__len__`. Metoda `__getitem__(index)` vrací dávku dat na pozici `index`. `Sequence` zajišťuje, že se při multiprocessing trénování každý vzorek použije právě jedenkrát, což generátory nezajišťují.

Kapitola 4

Experimenty a vyhodnocení

Začátek této kapitoly, sekce 4.1, obsahuje popis vlastností a složení datových sad, využitých pro trénování a následné experimenty. Dále sekce 4.2 obsahuje 4 experimenty, ve kterých jsem se pokusil demonstrovat vliv augmentací a batch size[1] na průběh trénování a výslednou přesnost rozpoznávání. Poznatky z těchto experimentů a znalosti nabyté ze stanfordských přednášek [12] jsem aplikoval při trénování výsledného modelu pro tuto práci v sekci 4.3. Porovnání výsledného modelu s jiným holistickým modelem pro rozpoznání registračních značek [10] se nachází v sekci 4.4.

4.1 Datasety

Tato kapitola popisuje datové sady, které byly v této práci využity pro trénování a experimenty. Datová sada je soubor anotovaných dat, které slouží pro trénink a testování neuronových sítí. Typicky se pro lepší porozumění trénovacího procesu část trénovacích dat (v případě této práce 10 %) vybere jako validační data. Validační data se neúčastní trénovacího procesu, ale slouží, jako zpětná vazba o probíhajícím tréninku. Pomocí zpětné vazby z validačních dat se dá odhalit například špatné nastavení hyperparametrů, nebo schopnost generalizace sítě. Testovací data jsou plně oddělena od trénování a využívají se pro vyhodnocování experimentů. Datové sady by obecně měly obsahovat co největší počet unikátních vzorků s dostatečným počtem zástupců každé kategorie, kterou chceme, aby výsledná síť rozpoznávala. V případě rozpoznávání značek jsou tyto kategorie jednotlivé znaky registrační značky (znaky abecedy a číslice) a délka (počet znaků) registrační značky. Protože jsou sedmiznaké registrační značky u nás nejběžnější velikostí, tak i dále zmíněné datové sady mají většinové zastoupení značek této kategorie (řádově cca 500× více oproti kategorii s druhým největším zastoupením). Z tohoto důvodu má výsledná síť často problémy s rozpoznáváním značek jiných délek, než 7 znaků.

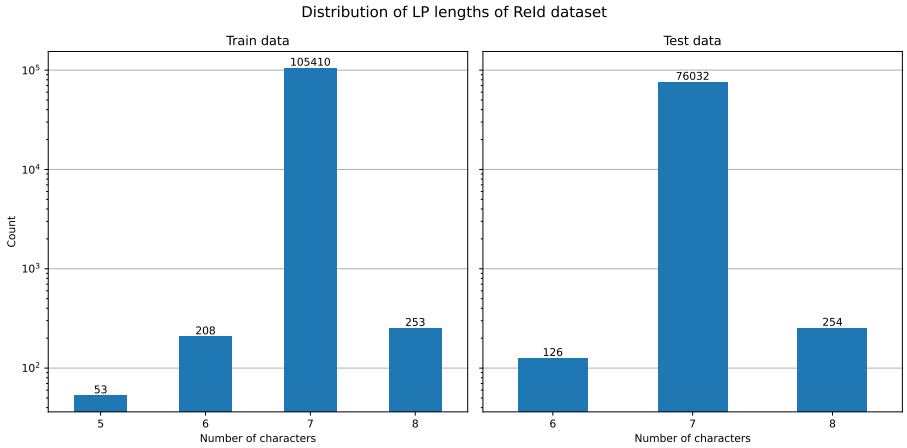
ReId

Datová sada ReId obsahuje 105 924 trénovacích a 76 412 testovacích barevných obrázků registračních značek. V obrázku 4.1 je vybrán vzorek registračních značek o všech čtyřech délkách, obsažených v datasetu. Tato sada obsahuje pro každou registrační značku vždy více smímků různé ostrosti. Sada obsahuje registrační značky různých délek. Distribuci délek registračních značek v datasetu zobrazuje graf 4.2. Protože dataset obsahuje více

[1] augmentace, batch size - Pojmy vysvětleny v sekci 2.5.

snímků stejné značky, tak množstní unikátních značek pro délku jinou než 7 znaků je ještě značně menší. Přesný počet unikátních značek různé délky zobrazuje tabulka 4.1.

Obrázek 4.1: Ukázka vzorku registračních značek z ReId datasetu. Do vzorku byly vybrány značky všech čtyř délek po 4 značkách. V levé polovině se nachází značky s 5 a 6 znaky, v pravé polovině pak značky o délce 7 a 8 znaků.



Obrázek 4.2: Graf zobrazuje statistiku o distribuci značek různé délky v ReId datasetu. Osa y grafu je v logaritmickém měřítku, kvůli značnému nepoměru sedmiznakých registračních značek vůči značkám jiné délky (sedmiznakých je více než 200× více, než značek jiné délky). Lze si povšimnout, že značky o pěti znacích se vyskytují pouze v trénovacím datasetu, ale v testovacích datech zcela chybí.

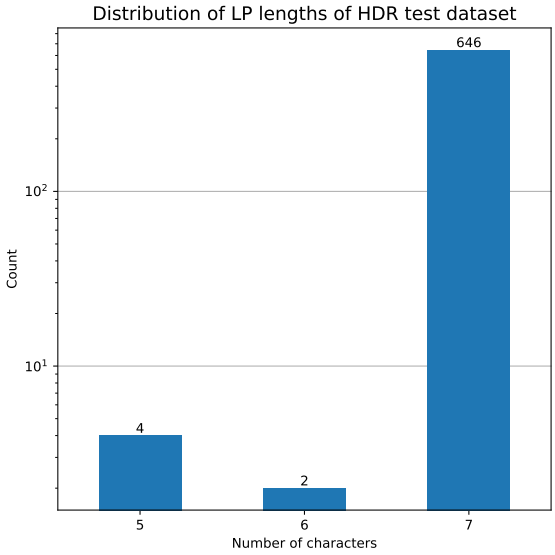
Počet znaků	Unikátních značek
5 znaků	6
6 znaků	28
7 znaků	8762
8 znaků	38

Tabulka 4.1: Počet unikátních registračních značek v ReId datasetu podle počtu znaků.

HDR

HDR dataset obsahuje pouze testovací data v podobě 652 obrázků. Registrační značky v tomto datasetu mají různou rotaci, sklon, a poměr stran, čímž se velmi liší od značek z ReId datasetu, použitých při trénování. Ukázku registračních značek z tohoto datasetu lze vidět v obrázku 4.3. Dataset obsahuje i značky délky 5 a 6 znaků, ale jsou opět ve velmi menšinovém zastoupení oproti sedmiznakým (viz. graf 4.4).

Obrázek 4.3: Ukázka vzorku registračních značek z HDR datasetu. Obrázek demonstruje odlišnost dat od ReId datasetu ve sklonu a poměru stran obrázků registračních značek. Do vzorku byly vybrány značky všech tří délek obsažených v datasetu, kde první dvě značky mají 5 znaků, třetí má 6 znaků a zbytek jsou značky sedmiznaké, protože je v datasetu nedostatek různých značek menší délky.



Obrázek 4.4: Graf zobrazuje statistiku o distribuci značek různé délky v HDR datasetu. Osa y grafu je v logaritmickém měřítku, kvůli značnému nepoměru sedmiznakých registračních značek vůči značkám jiné délky. HDR dataset obsahuje pouze testovací data.

4.2 Porovnání metod učení

Tato sekce se zaměřuje na oblast vlivu hyperparametrů na trénovací proces. V článku [6] jsem se dočetl o různých tipech pro trénování neuronových sítí. V rámci studie těchto tipů jsem provedl čtyři experimenty s cílem zjistit, jaký vliv má batch size a augmentace na průběh trénování a výslednou přesnost na testovacích datech při trénování na stejném množství dat.

Následující podsekce popisují průběh a vyhodnocení těchto experimentů. Pro tyto čtyři experimenty jsem využil optimalizátor Adam s learning rate 10^{-3}, trénování probíhalo 100 epoch. Při všech experimentech jsem využil callback funkce `ModelCheckpoint` pro ukládání nejlepšího nastavení parametrů sítě podle loss funkce na validačních datech a cílovou přesnost na testovacích datech ověřoval na tomto nejlepším nastavení parametrů. Nastavení augmentací využitých ve všech experimentech demonstruje ukázka kódu 3.1.

Experiment 1

Nastavení parametrů trénování pro experiment:

- Augmentace: Ne
- Batch size: 16

Průběh trénování demonstruje a popisuje obrázek 4.5. Lze si všimnout, že po prvních dvaceti epochách kolísání loss funkce na validačních datech téměř ustala a od 60. epochy se již téměř nezměnila. Protože při experimentu 1 neprobíhaly žádné augmentace, lze vidět, že při posledních cca osmi epochách se hodnota loss funkce na validačních datech nepatrně zvýšila, což je náznakem overfittingu. Celkově průběh tohoto trénování s batch size 16 je poměrně hladký bez nečekaných výkyvů. Výsledná přesnost tohoto experimentu na testovacích datech je $92,98\%$.

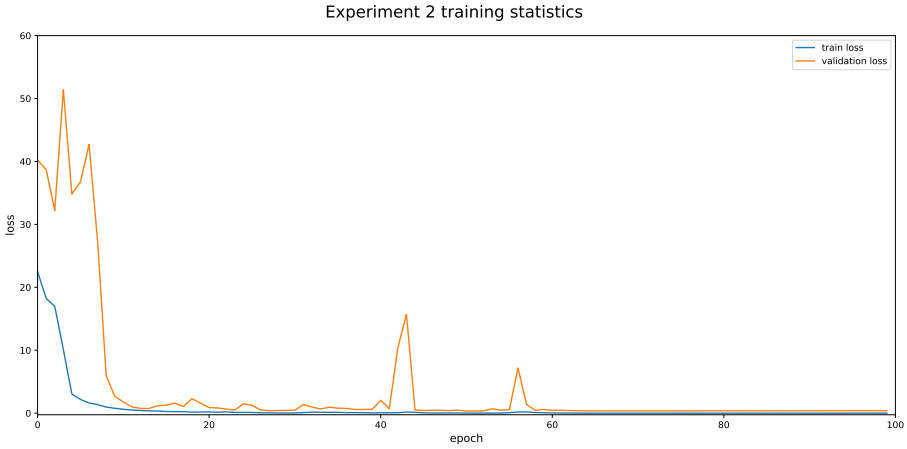
Obrázek 4.5: Graf vývoje loss funkce pro experiment 1. Lze si všimnout, že průběh tohoto trénování s batch size 16 je poměrně hladký bez nečekaných výkyvů.

Experiment 2

Nastavení parametrů trénování pro experiment:

- Augmentace: Ne
- Batch size: 64

Průběh trénování demonstruje a popisuje obrázek 4.6. Zvětšení batch size na 64 vzorků zapříčinilo, že se parametry sítě aktualizovaly méně často ($4\times$ méně) a tím v počátku trénování prudce kolísala hodnota loss funkce na validačních datech. Jakmile ale parametry dospěly do bodu konvergence, velice prudce tato hodnota klesla blízko optima. Od 60. epochy se hodnota loss funkce držela velice blízko optima podobně jako u prvního experimentu. Výkyvy kolem 40. epochy a před 60. epochou mohl zapříčinit například větší poměr značek jidé délky, než 7 znaků v dávce a tím posunutí parametrů více směrem, který neodpovídá obecnému popisu dat v datasetu. Nebyly zavedeny augmentace a tak by při delším trénování hrozil overfitting. Výsledná přesnost tohoto experimentu na testovacích datech je $94,75\%$.



Obrázek 4.6: Graf vývoje loss funkce pro experiment 2. Výkyvy kolem 40. epochy a před 60. epochou mohl zapříčinit například větší poměr značek jidé délky, než 7 znaků v dávce a tím posunutí parametrů více směrem, který neodpovídá obecnému popisu dat v datasetu.

Experiment 3

Nastavení parametrů trénování pro experiment:

- Augmentace: Ano
- Batch size: 16

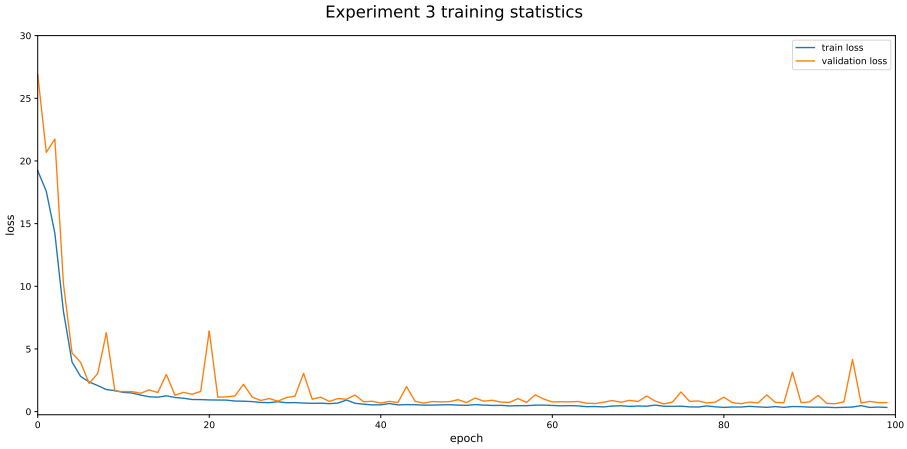
Průběh trénování demonstruje a popisuje obrázek 4.7. Zavedení augmentací očividně ztížilo proces trénování sítě, což lze vyčíst z neustálého kolísání loss funkce na validačních datech. Augmentace, které byly zavedeny modifikují vstup takovým způsobem, že nastávají případy, kdy na vstupním obrázku nejsou některé znaky vidět vůbec, což ale napomáhá generalizaci sítě. Síť se s tak velkými výkyvy v malých dávkách o velikosti 16 vypořádává hůře, takže by pro lepší výsledky bylo za potřebí mnohem delší trénování, než 100 epoch, za předpokladu, že by trénování s augmentacemi po tak malých dávkách konvergovalo. I přes tento náročnější proces trénování augmentace zajistily, že výsledná přesnost tohoto experimentu na testovacích datech je $95,17\%$.

Experiment 4

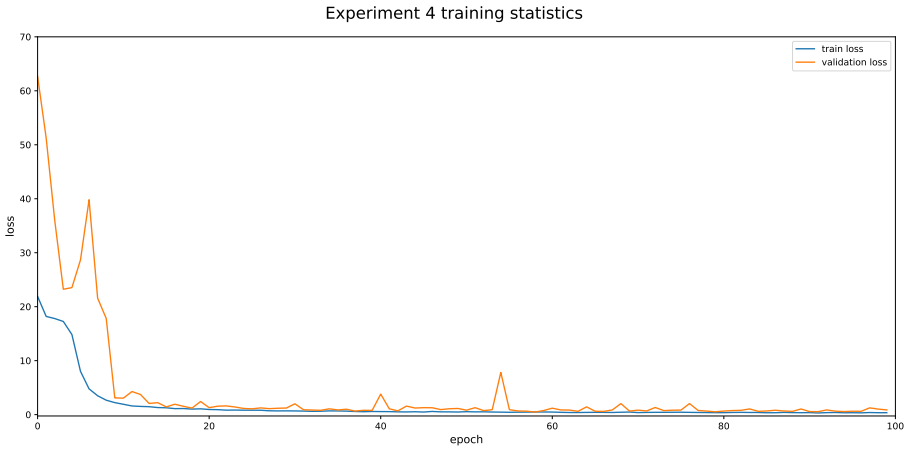
Nastavení parametrů trénování pro experiment:

- Augmentace: Ano
- Batch size: 64

Průběh trénování demonstruje a popisuje obrázek 4.8. Nastavení augmentací je stejné, jako u experimentu 3, ale zvětšením batch size na 64 se zmírnilo kolísání loss funkce na validačních datech. Z tohoto průběhu lze tedy odvodit, že velké výkyvy v několika vzorcích nemají tak negativní dopad na trénování při větších dávkách, jako při menších. Při zavedení augmentací větší batch size může tedy znamenat rychlejší konvergenci se zachováním generalizace zajištěné augmentacemi. Toto tvrzení potvrzuje i výsledná přesnost tohoto experimentu na testovacích datech $95,55\%$, což je nejlepší z těchto experimentů.



Obrázek 4.7: Graf vývoje loss funkce pro experiment 3. Zavedení augmentací očividně ztížilo proces trénování sítě, což lze vyčíst z neustálého kolísání loss funkce na validačních datech.

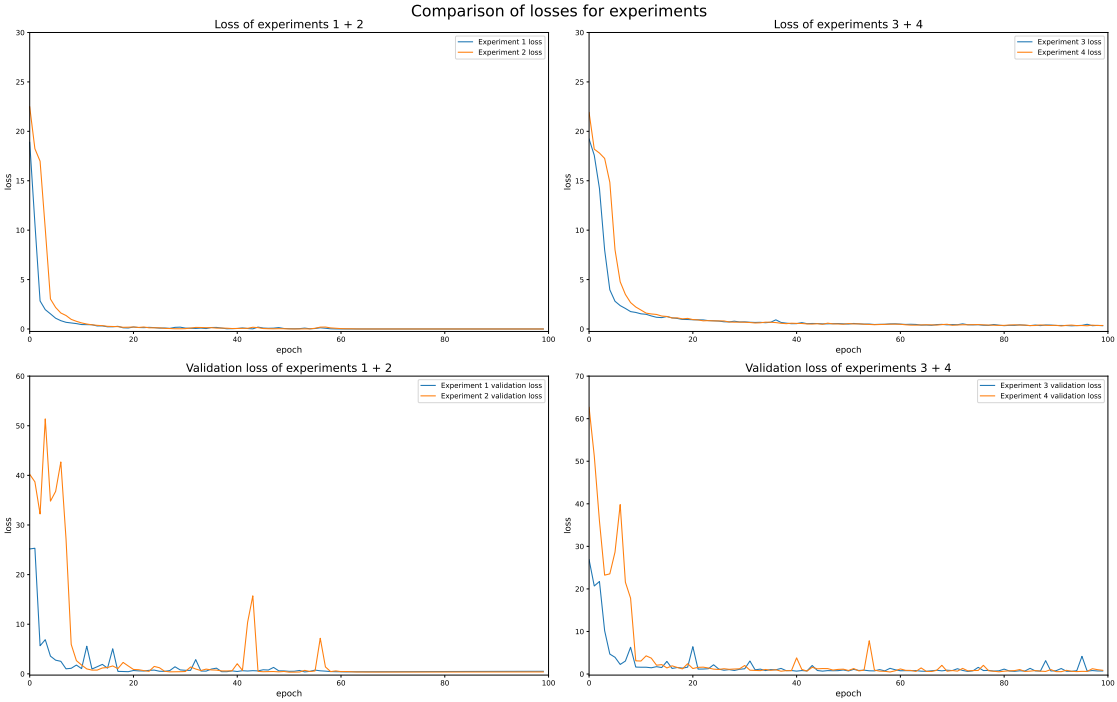


Obrázek 4.8: Graf vývoje loss funkce pro experiment 4. Nastavení augmentací je stejné, jako u experimentu 3, ale zvětšením batch size na 64 se zmírnilo kolísání loss funkce na validačních datech.

Shrnutí

Z těchto čtyř experimentů jsem se dozvěděl, jaký vliv má batch size a augmentace na průběh trénování a výslednou přesnost modelu. Při nastavení menší velikosti batch size hodnota loss funkce začne konvergovat mnohem dříve, než při větší velikosti batch size nezávisle na augmentacích. Nastavení větší velikosti batch size vedlo u obou experimentů s větší batch size ke zlepšení výsledné přesnosti sítě, než u experimentů s menší velikostí batch size (zvlášť pro nastavení s augmentacemi a bez). Experimenty bez augmentací měly hladší průběh trénování, kdy kolem 60. epochy přestala kolísat hodnota loss funkcí. Pokud by trénování pokračovalo ještě déle, tak by oba experimenty měly problém s overfittingem (první experiment u posledních 10 epoch začal jevit známky overfittingu). Zavedení augmentací pozitivně ovlivnilo výslednou přesnost modelu. Oba experimenty s augmentacemi měly větší přesnost na testovacích datech, než experimenty bez augmentací. Na druhou stranu trénování s augmentacemi potřebuje více času pro dosažení nejlepšího výsledku. Po-

rovnání průběhů trénování mezi experimenty bez augmentací a s augmentacemi popisuje obrázek 4.9.



Obrázek 4.9: Graf srovnání loss funkcí experimentů. Vlevo jsou experimenty bez augmentací a vpravo s augmentacemi. Experimenty vyznačeny modrou barvou mají batch size 16, oranžovou 64. Trénování s batch size 16 bez zavedení augmentací má rychlejší a hladší průběh, ale chybí generalizace sítě s augmentacemi spojená a hrozí rychlejší overfitting. Trénování s augmentacemi probíhá lépe při větší velikosti batch size. Celkově trénování s menší batch size začne rychleji konvergovat, ale při delším trénování zajistí lepší výsledky větší batch size.

4.3 Spojení poznatků

Cílem práce je natrénování sítě s co nejlepší přesností rozpoznávání registračních značek. Na základě poznatků z experimentů (sekce 4.4) a rad z příspěvku [6] jsem navrhl vlastní postup při trénování sítě. Cílem tohoto postupu bylo dosažení co nejlepšího výsledku za co nejkratší dobu trénování. Při trénování jsem využil oprimalizátor Adam.

Následuje popis trénovacího procesu tohoto postupu, kde nastavení každé jednotlivé části trénování i průběžná přesnost na testovacím datasetu je obsažena v tabulce 4.2. Graf 4.10 obsahuje informace o celém průběhu trénování. Každá barva v grafu značí jiné nastavení hyperparametrů. Tabulka 4.2 slouží jako legenda pro nastavení hyperparametrů každé fáze trénování. S nastavením se zelenou barvou ze zavedly augmentace.

Jak jsem zjistil z předešlých experimentů, nejrychleji se dosáhne konvergence loss funkce při nastavení menší batch size. Zavedení augmentací tento proces také prodloužilo, takže pro nasměrování nastavení parametrů sítě jsem zvolil batch size 16 bez augmentací. Protože tento postup měl parametry sítě navést jen správným směrem, využil jsem callback funkci

`EarlyStopping(patience=3)` pro zastavení trénování po třech epochách bez zlepšení loss funkce na validačních datech. Pro ještě větší přiblížení se tomuto „prozatimnímu“ optimu jsem pokračoval trénování se stejným nastavením, ale learning rate 10^{-4}.

Pro prohledání větší části stavového prostoru nastavení parametrů sítě jsem při zavedení augmentací vrátil hodnotu learning rate na 10^{-3}. Nastavení hyperparametrů úvodním postupem bez augmentací mělo požadovaný účinek, protože po zavedení augmentací hodnota loss funkce už příliš dramaticky nestoupla. Augmentace se nadále využívaly až do konce trénovacího procesu. 35 epoch mi přišlo jako dostačující počet pro toto nastavení, protože se parametry sítě už pohybovaly poblíž optima. Batch size jsem ponechal na velikosti 16, protože menší velikost batch size vede k prudším změnám nastavení parametrů, což odpovídá mému záměru prohledání větší části stavového prostoru.

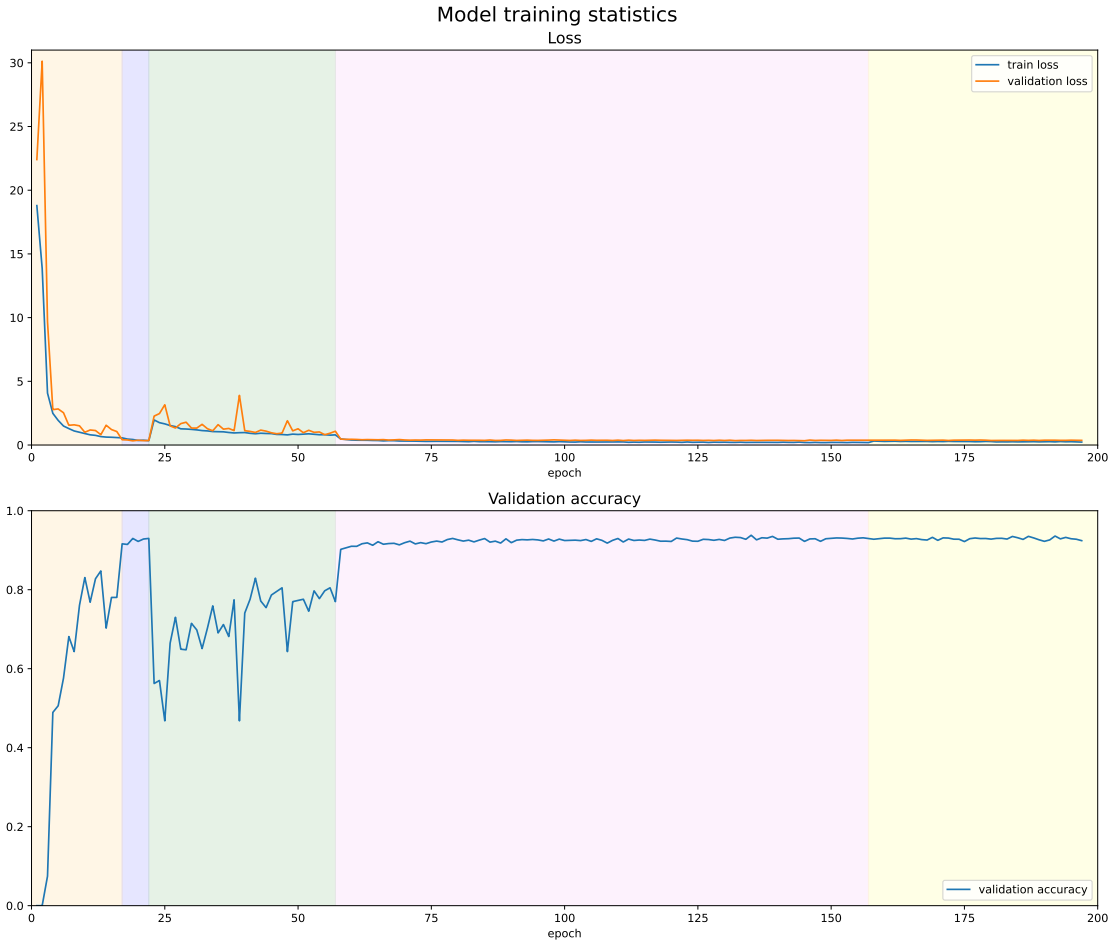
Ve čtvrté a nejdelší části tohoto procesu trénování jsem čerpal z poznatku z předešlých experimentů, že pokud jsou zavedeny augmentace, tak má nastavení batch size na větší hodnotu pozitivní účinek na proces trénování i výslednou přesnost modelu za ceny delšího trénování. Proto jsem nastavil batch size na 64. Protože se nastevaní parametrů sítě už velice blížilo hledanému optimu, nechtěl jsem, aby se prudkou změnou nastavení parametrů od tohoto optima vzdálilo k jinému optimu. Z tohoto důvodu jsem snížil learning rate na 10^{-4}.

Na závěr, pro dosažení nejlepšího výsledku a co největšímu přiblížení se optimu, jsem ponechal stejné nastavení a snížil learning rate na 10^{-5}. Trénování s tímto nastavením jsem ponechal po 40 epoch. Ke konci si lze povšimnout začínajícího poklesu přesnosti modelu, což naznačuje, že začíná docházet k overfittingu a tím pádem už lepších výsledků z tohoto trénovacího procesu nelze dosáhnout.

Závěrem této sekce si dovolím připomenout větu z úvodu předešlé sekce 4.4: „Při všech experimentech jsem využil callback funkce `ModelCheckpoint` pro ukládání nejlepšího nastavení parametrů sítě podle loss funkce na validačních datech a cílovou přesnost na testovacích datech ověřoval na tomto nejlepším nastavení parametrů.“ Každá část trénování s jiným nastavením hyperparametrů začínala tedy s tímto nejlepším nastavením parametrů sítě.

	Barva v grafu 4.10				
	Oranžová	**Modrá**	**Zelená**	**Růžová**	**Žlutá**
Počet epoch	17	5	35	100	40
Learning rate	1e-3	1e-4	1e-3	1e-4	1e-5
Batch size	16	16	16	64	64
Testovací přesnost	92, 39%	96, 22%	96, 67%	97, 27%	97, 85%

Tabulka 4.2: Tabulka nastavení hyperparametrů a průběžných přesností na testovacím datasetu pro každou část trénovacího procesu hlavního experimentu. Tabulka slouží pro přehled o tomto trénování a jako legenda pro graf 4.10.



Obrázek 4.10: Graf průběhu trénování a přesnosti pro hlavní experiment. V horním grafu je zobrazen vývoj loss funkce na trénovacích i validačních datech. V dolním grafu je zobrazen vývoj přesnosti rozpoznání celých značek na validačních datech. Každá barva v grafu značí jené nastavení hyperparametrů, které je popsáno v tabulce 4.2. Výkyvem při nastavení se zelenou barvou bylo přidání augmentací dat. Lze si povšimnout, že ke konci trénování začíná klesat přesnost, což může značit overfitting.

4.4 Porovnání modelů

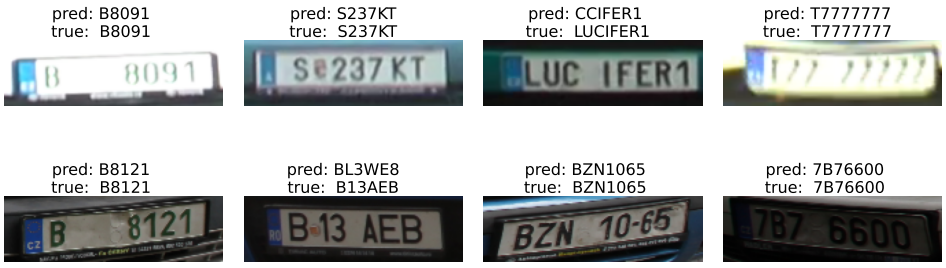
Jako metriku pro určení, zda se hlavní experiment vyvedl, či nikoliv jsem zvolil porovnání mého modelu s jiným modelem s holistickým přístupem rozeznávání registračních značek. Aby bylo porovnání relevantní, je vhodné vybrat model natrénovaný na stejném datasetu ReId. Proto jsem pro porovnání zvolil model z práce [10], za kterým stojí tým Jakub Špaňhel, Jakub Sochor, Roman Juránek, Adam Herout, Lukáš Maršík a Pavel Zemčík. Porovnání proběhlo na testovacích datech datasetů ReId a HDR. Výsledky porovnání jsou obsaženy v tabulce 4.3.

Jak lze v tabulce vidět, výsledek této práce má lehce větší chybovost v rozeznávání celých značek na datasetu ReId, než vybraný model pro porovnání. Zvýšení chybovosti na HDR datasetu je důsledkem rozlišnosti HDR datasetu od trénovacího ReId datasetu (porovnání datasetů lze vidět v sekci 4.1). Můj model překonal porovnávaný model na

HDR datasetu, na kterém dosáhl o poznání menší chybovosti. Důvodem úspěchu na HDR datasetu je využití augmentací při trénování, které vnáší do trénovacích dat velký rozdíl ve sklonu a zařazuje i změnu poměru stran. Také je důležitým faktorem naddimenzovaná velikost výstupu enkóderu sítě právě kvůli lepšímu rozpoznávání značek s větším sklonem pomocí CTC (zmíněno v sekci 3.3). Obrázek 4.11 je ukázkou rozpoznaných registračních značek mým modelem.

error rate [%] (znak/celá značka)		
model	ReId	HDR
práce [10]	**0,4**/**1,4**	3,5/9,7
práce [10] (small)	**0,4**/1,7	4,5/12,1
můj	**0,4**/2,2	**2,2**/**8,3**

Tabulka 4.3: Tabulka porovnání chybovosti mého modelu a vybraného modelu [10] v rozpoznávání registračních značek z datasetů ReId a HDR. Data vybraného modelu jsou přejata z téže práce a vyhodnocování chybovosti proběhlo na základě dvou metrik. Hodnota nalevo udává chybovost v rozpoznávání jednotlivých znaků a hodnota napravo udává chybovost v rozpoznávání celých značek.



Obrázek 4.11: Ukázka registračních značek, rozpoznaných sítí z této práce. Horní 4 vzorky jsou z ReId datasetu a dolní 4 vzorky z HDR datasetu. Lze si všimnout, že značky s jiným počtem znaků, než 7 jsou pro síť větší problém, kvůli nepoměru kategorií dat v datasetu (více popsáno v sekci 4.1).

Kapitola 5

Závěr

Hlavním cílem této práce bylo vytvoření modelu neuronové sítě pro rozpoznávání registračních značek s co nejlepší přesností rozpoznání za co nejkratší dobu trénování. Proto bylo vedlejším cílem této práce nejdříve porovnat vliv hyperparametrů a augmentací na proces trénování sítě a výslednou přesnost modelu. Pro porovnání vlivu hyperparametrů a augmentací jsem provedl čtyři experimenty, dva bez augmentací a dva s augmentacemi. Výsledkem těchto experimentů bylo zjištění, že při nastavení menší velikosti dávky při trénování začne loss funkce konvergovat mnohem rychleji, než při větších dávkách. Dalším zjištěním bylo, že při zavedení augmentací mají hladší průběh trénování naopak dávky větší velikosti.

Poznatky z experimentů jsem následně aplikoval při trénování výsledného modelu této práce. Přesnost rozpoznání pro jednotlivé znaky/celou značku byla na datové sadě **ReId** $99,55\%/97,85\%$, na datové sadě **HDR** byla přesnost $97,83\%/91,72\%$. Výsledky jsem následně porovnal s implementací z práce [10], kde moje implementace dosáhla na ReId datasetu o málo horší přesnosti, ale na datasetu HDR moje implementace dosáhla o poznání lepší přesnosti, než srovnávaná implementace.

Pro ještě lepší výsledek by bylo dobré trénovat model na více druzích datových sad. Dosáhlo by se tak lepší generalizace a tím větší přesnosti na neznámých datech. Také by mohlo k výsledku přispět různé nastavení augmentací, místo jednoho konstantního napříč celým trénovacím procesem. Potenciálně velkého zlepšení by se dalo dosáhnout implementací rekurentní části modelu s 2D-attention mechanismem, jako v původním návrhu modelu, čímž by se zamezilo ručně zvolenému počtu znaků pro predikci pomocí CTC.

Literatura

[1] ActiveState. *What Is Pandas In Python? Everything You Need To Know* [online]. [cit. 2022-05-09]. Dostupné z: https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/.

[2] Anwar, F. *Gradient Descent — Intro and Implementation in python* [online]. Analytics Vidhya, 2019 [cit. 2022-05-06]. Dostupné z: https://medium.com/analytics-vidhya/gradient-descent-intro-and-implementation-in-python-8b6ab0557b7c.

[3] Chen, J., Nishimura, R. a Kitaoka, N. End-to-end recognition of streaming Japanese speech using CTC and local attention. *APSIPA Transactions on Signal and Information Processing*. Cambridge University Press. 2020, sv. 9, s. e25. DOI: 10.1017/ATSIP.2020.23.

[4] Chollet, F. Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, s. 1251–1258.

[5] HUBEL, D. a WIESEL, T. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*. October 1959, sv. 148, s. 574—591. DOI: 10.1113/jphysiol.1959.sp006308. ISSN 0022-3751. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/pmid/14403679/?tool=EBI.

[6] Janetzky, P. *Tips and tricks for Neural Networks* [online]. [cit. 2022-05-12]. Dostupné z: https://towardsdatascience.com/tips-and-tricks-for-neural-networks-63876e3aad1a.

[7] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C., Bottou, L. a Weinberger, K., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, sv. 25. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[8] Lee, T., Chollet, F. a fuzzythecat. *Xception V1 model for Keras* [online]. [cit. 2022-05-13]. Dostupné z: https://github.com/keras-team/keras-applications/blob/master/keras_applications/xception.py.

[9] Peiwen, Y. *Captcha Break* [online]. [cit. 2022-05-09]. Dostupné z: https://githubhelp.com/ypwhs/captcha_break.

[10] Špaňhel, J., Sochor, J., Juránek, R., Herout, A., Maršík, L. et al. Holistic recognition of low quality license plates by CNN using track annotated data. In:

IEEE. *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2017, s. 1–6 [cit. 2022-05-08].

[11] Stanford, U. *CS231n Convolutional Neural Networks for Visual Recognition* [online]. [cit. 2022-05-06]. Dostupné z: https://cs231n.github.io/.

[12] Stanford, U. *Lecture Collection | Convolutional Neural Networks for Visual Recognition (Spring 2017)* [online]. [cit. 2022-05-08]. Dostupné z: https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv.

[13] W3Schools. *NumPy Introduction* [online]. [cit. 2022-05-09]. Dostupné z: https://www.w3schools.com/python/numpy/numpy_intro.asp.

[14] Yegulalp, S. *What is TensorFlow? The machine learning library explained* [online]. [cit. 2022-05-09]. Dostupné z: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html.

[15] Zhang, L., Wang, P., Li, H., Li, Z., Shen, C. et al. A robust attentional framework for license plate recognition in the wild. *IEEE Transactions on Intelligent Transportation Systems*. IEEE. 2020, sv. 22, č. 11, s. 6967–6976.