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

PROŘEZÁVÁNÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ PRO ROZPOZNÁVÁNÍ TEXTU

DEEP NEURAL NETWORK PRUNING FOR TEXT RECOGNITION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **SIMON PETRÁŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN KIŠŠ**
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: **Petráš Simon**
Program: Informační technologie
Název: **Prořezávání hlubokých neuronových sítí pro rozpoznávání textu**
Deep Neural Network Pruning for Text Recognition
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy konvolučních sítí a rozpoznávání textu.
2. Vytvořte si přehled o současných metodách rozpoznávání textu pomocí konvolučních sítí a prořezávání neuronových sítí (pruning).
3. Vyberte nebo navrhněte metodu na prořezávání sítí pro rozpoznávání textu.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad neuronovou sítí a datovou sadou.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- SHI, Baoguang; BAI, Xiang; YAO, Cong. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2016, 39.11: 2298-2304.
- KANG, Lei, J. Ignacio TOLEDO, Pau RIBA, Mauricio VILLEGAS, Alicia FORNÉS a Marçal RUSIOL. Convolve, Attend and Spell: An Attention-based Sequence-to-Sequence Model for Handwritten Word Recognition. In: *Pattern Recognition*. Stuttgart, Germany: Springer International Publishing, 2019, s. 459-472. ISBN 978-3-030-12938-5.
- SHENG, Fenfen; CHEN, Zhineng; XU, Bo. NRTR: A no-recurrence sequence-to-sequence model for scene text recognition. In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2019. p. 781-786.
- HUANG, Qiangui, et al. Learning to prune filters in convolutional neural networks. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018. p. 709-718.

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: **Kišš Martin, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 2. listopadu 2021

Abstrakt

Dokument predstavuje prácu na prerezávanie neurónovej siete slúžiacej na rozpoznávanie ručne písaného textu. Cieľom práce je vytvoriť program na prerezávanie danej siete. Prerezávali sme dva typy neurónových sieti a to konvolučné a rekurentné neurónové siete. Pri prerezávaní konvolučnej časti bolo experimentované s rôznymi kritériami výberu parametrov. Výsledkom práce je model, ktorý dosahuje 20% zrýchlenie pri znížení presnosti siete iba o 0.4%, ale aj množstvo iných modelov, ktoré sú rýchlejšie ale nadobúdajú aj vyššej nepresnosti.

Abstract

This document is a work on pruning neural network for handwriting recognition. The aim of the work is to create a program for pruning the network. We prune two types of neural networks, namely convolutional and recurrent neural networks. During the pruning of the convolution part, various criteria of parameter selection were experimented with. The result of the work is a model that achieves 20% acceleration while increasing the network inaccuracy by only 0.4%, but also a number of other models that are faster but also acquire higher inaccuracies.

Kľúčové slová

Neurónové siete, CNN, RNN, Pytorch, OCR, prerezávanie

Keywords

Neural networks, CNN, RNN, Pytorch, OCR, pruning

Citácia

PETRÁŠ, Simon. *Prořezávání hlubokých neuronových sítí pro rozpoznávání textu.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Kišš

Prořezávání hlubokých neuronových sítí pro rozpoznávání textu

Prehlásenie

Prehlasujem, že som tuto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Martina Kišša. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Simon Petráš
11. mája 2022

Poďakovanie

Rád by som sa poďakoval vedúcemu práce pánovi Ing. Martinovi Kiššovi za odborný prístup v podobe častých konzultácií, za trpezlivosť a veľa užitočných rád a pripomienok k práci.

Obsah

Kapitola 1

Úvod

Rozpoznávanie rukou písaného textu je téma, ktorou sa odborníci zaoberajú už veľa rokov. Transformácia ručne písaného textu do strojovo čitateľného formátu má veľké množstvo využití, ako napríklad spracovanie historických dokumentov, spracovanie v poštovej podateľni, administratívne dokumenty, atď. Veľké rozdiely v štýloch rukopisu medzi ľuďmi a nízka kvalita ručne písaného textu v porovnaní s tlačeným textom a množstvo rôznych jazykov a písiem predstavujú značné prekážky pri jeho prevode na strojovo čitateľný text. V tejto práci sa budeme zaoberať viacerými metódami na riešenie úloh rozpoznávania ručne písaného textu ako sú konvolučné rekuretné siete (CRNN)[11] s použitím Connectionist Temporal Classification (CTC)[4] alebo sequence-to-sequence modely s attention mechanizmom[13, 5].

Vysoký prediktívny výkon hlbokých neurónových sietí však často prichádza na úkor vysokých nákladov na ukladanie a výpočtové náklady, ktoré súvisia s energetickými výdajmi siete. Tieto hlboké architektúry sa skladajú z miliónov parametrov, ktoré sa majú trénovať, čo vedie k nadmernej parametrizácii (t. j. s väčším počtom parametrov ako trénovacie vzorky) modelu. Preparametrizácia je užitočná pre efektívne a úspešné trénovanie neurónových sietí, avšak akonáhle je trénovaná sieťová štruktúra vytvorená, prerezávanie môže pomôcť ku zníženiu parametrov pri zachovaní dobrej úspešnosti siete.

Zníženie požiadaviek na úložisko a výpočtové náklady modelu sa stáva rozhodujúcim pre širšiu použiteľnosť, napríklad vo vstavaných systémoch, autonómnych agentoch alebo mobilných zariadeniach. Pre prerezávanie siete je dôležité rozhodnúť, ako identifikovať irelevantné množstvo parametrov, ktoré neprispieva ku finálnej presnosti siete. V tejto práci sa budeme zaoberať dvomi typmi hlbokých neurónových sieti, ktoré budeme prerezávať a to konvolučnými a rekurentnými neurónovými sietami. Budeme skúmať rôzne kritéria výberu prerezaných parametrov a ich dopad na výslednú presnosť a rýchlosť siete.

V tejto práci sa používa konvolučná neurónová sieť poskytnutá vedúcim práce s pred trénovaným modelom, ktorý dosahuje presnosť 3.1% CER. Na to aby sme dosiahli zrýchlenie siete musíme odstrániť časť parametrov. Na správne fungovanie sme museli upraviť rôzne funkcie na vytváranie, testovanie a trénovanie poskytnutej siete, tak aby sme vedeli meniť štruktúru dodaného modelu.

Kapitola 2

Neurónové siete

Neurónové siete, ang. Neural networks, sú podmnožinou strojového učenia, ktoré napodobňujú spôsob fungovania ľudského mozgu. Neurónové siete sa skladajú z pospájaných neurónov, vyjadrených matematickou funkciou, ktoré zhromažďujú a klasifikujú vstupné informácie.

Hlboké neurónové siete sú kategória sietí, ktoré obsahujú dve alebo viac vrstiev spracovania. Patria sem siete: viac-vrstvové perceptrony (MLP), konvolučné neurónové siete (CNN) a rekurentné neurónové siete (RNN). Ďalej sa budeme zaoberať už iba CNN a RNN lebo to sú dve architektúry s ktorými budeme pracovať.

2.1 Konvolučné neurónové siete

Konvolučné neurónové siete, ang. Convolutional Neural Network(CNN), sú najčastejšie používané na rozpoznávanie obrazu. Konvolučné neurónové siete extrahujú vlastnosti zo vstupu (napríklad z obrázku) za pomocí filtrov. Skladajú sa z troch typov vrstiev a to z konvolučných vrstiev, združovacích vrstiev a plne prepojených vrstiev.

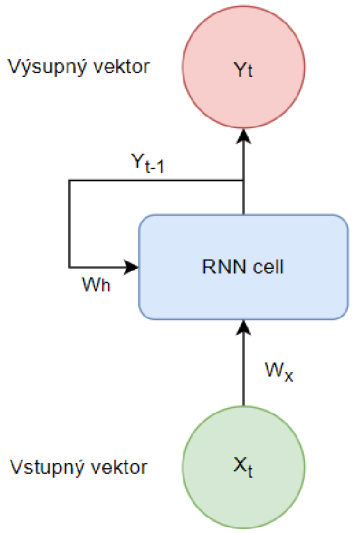
Konvolučné vrstvy

Táto vrstva je založená na naučiteľných filtroch[9]. Filtre sú matice pevných rozmerov, ktorých úlohou je analyzovať vlastnosti vstupu. Filter sa následne priloží na vstupný obrázok, vypočíta sa skalárny súčin medzi samotným filtrom a vstupom a následne sa filter posunie o veľkosť kroku (stride). Týmto posúvaním nám vznikne 2D mapa aktivácií so zachovaním priestorovej informácie.

Združovacie vrstvy

Združovacie vrstvy, ang. Pooling layers, sú vrstvy, ktorých úlohou je zmenšovanie rozmerov konvolučných vrstiev. Tieto vrstvy sa používajú na to, aby znížili výpočetnú náročnosť pri spracovaní dát. Okrem toho sú užitočné na extrakciu dominantných prvkov, ktoré sú rotačne a polohovo invariantné, čím sa udržiava proces efektívneho trénovania modelu. Poznáme dva typy združovacích vrstiev a to max pooling a average pooling vrstvy.

Pri max pooling vrstvách sa vyberie maximálna hodnota z danej oblasti vstupu, ktorou sa nahradia pôvodné hodnoty z danej oblasti. Max pooling funguje aj ako potláčač šumu. Pri average poolingu sa vypočíta priemerná hodnota pixelov v danej oblasti a rovnako ako pri max pooling sa tieto hodnoty nahradia novou priemernou hodnotou, čím nám vznikne



Obr. 2.1: Ukážka RNN bunky.

menšia matica. Daná oblasť sa následne posúva po vstupnej matici rovnako ako to bolo pri konvolučných filtroch. Takže ak máme oblasť veľkú 2x2 a použijeme ju na maticu s veľkosťou 4x4, tak po max alebo priemernom poolingu nám vznikne výstupná matica s veľkosťou 2x2. Združovacia vrstva sa môže ďalej použiť aj pre zmenšenie vstupu, vtedy sa volá na začiatku a veľkosť oblastí sa určuje dynamicky podľa veľkosti vstupu.

Plne prepojené vrstvy

Plne prepojená vrstva, ang. Fully-connected layer, je vrstva, v ktorej je každý neurón priamo prepojený ku dvom susedným vrstvám, bez toho aby boli vo vrstve prepojené medzi sebou[9]. Táto vrstva môže byť v architektúre použitá ako skrytá vrstva alebo ako posledná vrstva. Ak je použitá ako posledná tak slúži pri klasifikačných úlohách pre zaradenie vstupu do klasifikačných tried. Vtedy je počet výstupov rovný počtu klasifikačných tried a jej výstupy označujeme ako skóre pre jednotlivé triedy.

2.2 Rekurentné neurónové siete

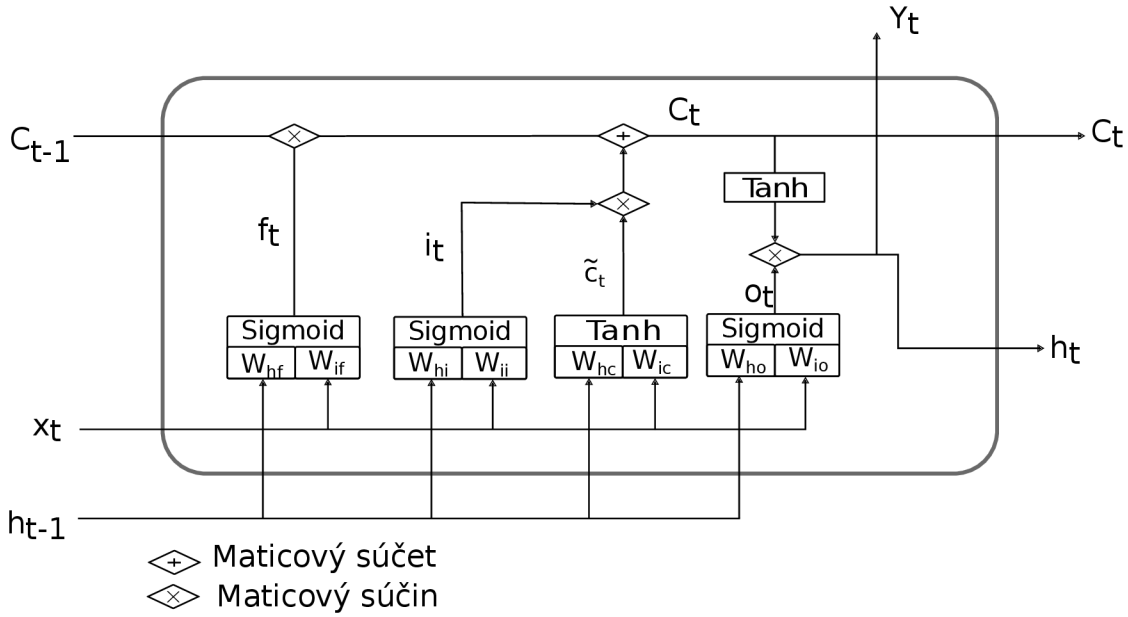
Rekurentné neurónové siete, ang. Recurrent Neural Networks(RNN), si zachovávajú informáciu o predchádzajúcom stave, čo znamená že výstup nieje ovplyvnený iba priamym vstupom, ale aj predchádzajúcimi vstupmi. RNN obsahuje cyklus, ktorý predáva každému výpočtu aj vstup predchádzajúcich dát. Výstup RNN je daný následujúcou rovnicou:

$$Y_t = f(W_x x_t + W_h Y_{t-1} + b) \tag{2.1}$$

Kde Y_t je výstup, Y_{t-1} je predchádzajúci výstup a x_t je vstup. W_x je matica váh ohodnocujúca vstup a W_h je matica váh ohodnocujúca predchádzajúce výstupy.

Rekurentné neurónové siete majú viacero výhod oproti iným typom a to: môžu spracovávať vstupy ľubovolnej veľkosti, výpočet berie do úvahy aj predchádzajúce stavy, váhy sú zdielané naprieč celým časom.

Existujú dve hlavné prekážky, s ktorými sa RNN musí vysporiadať:



Obr. 2.2: Ukážka LSTM bunky

Exploding gradients

Explodujúce gradienty sú problémom, keď sa hromadia veľké chybové gradienty a výsledkom sú veľmi veľké aktualizácie váh modelu neurónovej siete počas tréningu. V prípade explodujúcich gradientov akumulácia veľkých derivácií vedie k tomu, že model je veľmi nestabilný a neschopný efektívneho učenia. Veľké zmeny váh modelov vytvárajú veľmi nestabilnú sieť, ktorá sa pri extrémnych hodnotách stáva taká veľká, že spôsobuje pretečenie, ktorého výsledkom sú hodnoty hmotnosti *NaN*, ktoré už nie je možné aktualizovať.

Vanishing gradients

Miznúce gradienty sa vyskytujú, keď sú hodnoty gradientu príliš malé a model sa prestane učiť alebo to trvá príliš dlho. Akumulácia malých gradientov vedie k modelu, ktorý nie je schopný naučiť sa zmysluplné poznatky, pretože váhy a predsudky počiatočných vrstiev, ktoré majú tendenciu učiť sa základné vlastnosti zo vstupných údajov, sa nebudú aktualizovať efektívne. V najhoršom prípade bude gradient 0, čo zase zastaví sieť a zastaví ďalšie trénovanie.[17]

Tieto problémy sa dajú vyriešiť viacerými spôsobmi a to znížením počtu vrstiev, gradient clipping[19] alebo použitím LSTM, GRU architektúry.

2.2.1 LSTM

Long short-term memory (LSTM) je druh rekurentnej neurónovej siete, ktorý umožňuje dlhodobé uchovávanie kontextu. LSTM bunka sa skladá z pamäťovej bunky a troch brán a to: *Forget gate*, *Input gate* a *Output gate*. Tieto brány regulujú množstvo informácií a učia sa, ktoré informácie nie sú potrebné a môže ich zabudnúť a ktoré informácie sa majú zachovať lebo sú nevyhnutné. LSTM obsahuje taktiež aj *cell state*, ktorý slúži ako pamäť siete a je zodpovedný za prenos informácií medzi bránami.

Forget gate

Forget gate určuje, ktoré údaje budú zachované alebo zahodené. Určuje tak na základe predchádzajúceho skrytého stavu h_{t-1} a aktuálnom vektore x_t na vstupe, a vypíše číslo medzi 0 a 1 o čo sa postará sigmoid funkcia, kde 0 znamená odstránenie a 1 ponechanie stavu.

$$f_t = \delta(W_f x_t + U_f h_{t-1} + b) \tag{2.2}$$

Input gate

Na aktualizáciu používame *input gate*, vstupom input gate je predchádzajúci skrytý stav h_{t-1} a aktuálny vstupný vektor x_t. Táto brána je rozdelená na dve časti a to na sigmoid vrstvu a tanh vrstvu. Sigmoid vrstva vyberá podstatné dáta, ktoré dostávame zo tanh vrstvy. Sigmoid funkcia obmedzí výstup tak aby bol od 0 do 1 zatiaľ čo tanh obmedzuje výstup na hodnoty od -1 do 1.

$$\begin{aligned} i_t &= \delta(W_i x_t + U_i h_{t-1} + b) \\ C_t &= tanh(W_c x_t + U_c h_{t-1} + b) \end{aligned} \tag{2.3}$$

Teraz máme dostatok dát nato aby sme mohli vypočítať stav bunky(c_t). Výsledok z forget gate sa sčíta s výsledkom input gate, čo aktualizuje hodnotu stavu bunky (c_t) na hodnotu ktorá je relevantná pre neurónovú sieť.

Output gate

Output gate udáva aký bude následujúci skrytý stav h_t. Najprv vložíme do sigmoidnej funkcie predchádzajúci stav h_{t-1} a aktuálny vstup x_t. Potom výsledok vynásobíme s výstupom tanh funkcie, ktorej vstup tvorí novo vzniknutý stav bunky c_t.

$$\begin{aligned} o_t &= \delta(W_o x_t + U_i h_{t-1} + b) \\ h_t &= tanh(c_t) * o_t \end{aligned} \tag{2.4}$$

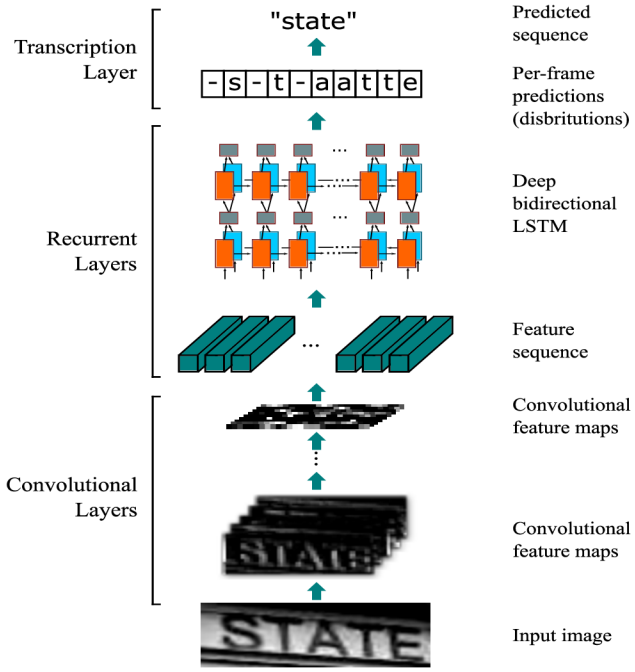
Premenné vystupujúce v LSTM bunke sú x_t, ktorý predstavuje vstupný vektor, h_t predstavuje skrytý stav, c_t je vektor stavu bunky, W, U predstavujú matice naučitelných váh a b predstavuje bias.

2.2.2 GRU

Gate Recurrent Unit je novšia generácia rekurentnúch sietí, ktorá sa architektúrou podobá na LSTM. Rovnako ako LSTM tak aj GRU je implementovaná tak, aby pomáhala s problémom miznúceho a vybuchujúceho gradientu. Na vyriešenie tohto problému používa dve brány a to *update gate* a *reset gate*. Namiesto *cell state*, ktorý slúži ako pamäť siete, GRU používa skrytý stav. GRU sú oproti LSTM jednoduchšie a majú menej operácií a tým môžu byť aj o trochu rýchlejšie.

Update gate

Funguje na rovnakom princípe ako input a forget gate v LSTM architektúre, rozhoduje, aké informácie budú zahodené a aké informácie budú pridané do skrytého stavu.



Obr. 2.3: Grafické zobrazenie architektúry CRNN[11].

Reset gate

Určuje aké množstvo minulých informácií zo skrytého stavu má byť zahodené alebo ponechané.

2.3 Konvolučne rekurentné neuronové siete

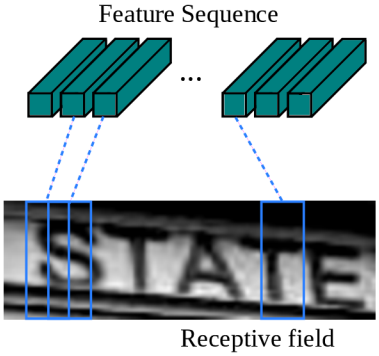
Konvolučné neurónové siete nám dovoľujú extrahovať relevantné črty z obrázka, zatiaľ čo *rekurentné neurónové siete* pomáhajú neurónovým sieťam brať do úvahy informácie z minulosti s cieľom predpovedať alebo analyzovať. Na rozpoznávananie textu potrebujeme tieto typy hlbokých neurónových sietí spojiť do jednej a to tak, že najprv získame črty textu za pomocí konvolučnej vrstvy a následne ich pošleme rekurentnej vrstve, ktorá ich spracuje pomocou CTC funkcie, ktorá vytvorí finálny výstup. Takýto typ neurónovej siete sa nazýva **konvolučná rekurentná neurónová sieť (CRNN)**.

Architektúra

V tejto práci sa budeme zaoberať architektúrou podľa článku[11], ktorá sa skladá z konvolučných, rekurentných a z transkripčnej vrstvy, ktoré sú zobrazené na Obrázku 2.3.

Konvolučné vrstvy

Pred samotným spracovaním vstupného obrázku je potrebné upraviť jeho výšku na definovanú veľkosť. Tento krok môžeme spraviť napríklad pomocou združovacích vrstiev. Po zmenšení/zväčšení obrázka použijeme konvolučnú vrstvu, ktorá vychádza zo štandardného modelu CNN, z ktorej získame dôležité črty. V CRNN sa nepoužívajú plne prepojené vrstvy na konci CNN, namiesto toho sa výstup z konvolučných vrstiev (mapy prvkov) transfor-



Obr. 2.4: Každý vektor v extrahovanom prvku sekvencie je spojený s receptívnym poľom na vstupnom obrázku a možno ho považovať za charakteristický vektor tohto poľa[11].

muje na sekvenciu vektorov prvkov, ktoré sú následne vstupom pre *rekurentné vrstvy* viď. Obrázok 2.4.

Rekurentné vrstvy

Rekurentné vrstvy analyzujú informácie získané z konvolučných vrstiev. Vstupom pre každú sekvenciu môžu byť vlastnosti (črty) získané z konvolučných vrstiev ako je zobrazené na Obrázku 2.4. Vďaka rekurentným vrstvám vieme rozpoznávať obrázky ľubovoľnej šírky. Ďalšou výhodou je rozpoznávanie podobných znakov ako je ”i” a ”l” na základe výšky predchádzajúcich znakov v kontexte. Pri implementácii rekurentnej vrstvy sa najčastejšie používajú LSTM alebo GRU architektúry, kvôli problému miznúceho a výbušného gradientu.

Transkripčné vrstvy

Transkripcia je proces konverzie čiastočných predpovedí (sekvencií), vykonaných pomocou rekurentných vrstiev na výsledný výstup. Transkripčná vrstva vytvára finálnu predikciu celého slova alebo riadku. Transkripcia je matematicky vyjadrená ako proces hľadania sekvencie s najväčšou pravdepodobnosťou podmienenou predikciami jednotlivých sekvencií[11].

Používajú sa dva typy transkripcie a to transkripcia založená na *slovníku* a transkripcia *bez slovníka*. Pri transkripcii so slovníkom je každá predikcia vybraná z tohto slovníka na základe najväčšej pravdepodobnosti zhody sekvencie získanej z neurónovej siete zo vstupu nad daným slovom. Problém nastáva pri použití väčší slovníkov, keďže nato aby sme dostali odpoveď musíme prejsť všetkými slovami v slovníku, táto operácia môže zabrať dlhú dobu. Na vyriešenie tohto problému sa používa bez slovníková transkripcia, ktorá vie celkom presne predpovedať dané slovo a následne môžme limitovať hľadanie v slovníku iba na slová, ktoré sú kandidáti najbližšieho suseda (nearest-neighbor candidates), ktorých odlišnosť musí byť menšia ako predom daná vzdialenosť[11]. Z tohto dôvodu používa CRNN architektúra kombináciu oboch transkripcií.

CTC

Keď sa zaoberáme výstupom CRNN modelu, ktorý je opísaný vyššie, tak na vyhodnotenie výsledku nemôžme použiť cross-entropy funkciu ako pri klasických neurónových sietiach. Výstup CRNN produkuje pre jeden vstup viacero sekvencií pravdepodobnosti. Jeden znak vo slove môže obsahovať viac ako jednu výstupnú sekvenciu a výstupom by v takomto

prípade boli duplicitné znaky, ktoré sa reálne vo vstupe nenachádzajú. Tento problém rieši *Connectionist Temporal Classification* (CTC) funkcia[4]. Ako už bolo povedané nechceme anotovať výsledky pre každú sekvenciu samostatne.

Trénovanie neurónového modelu bude vedené CTC funkciou straty. Do CTC funkcie privádzame výstupné sekvencie CRNN modelu a zodpovedajúci kľúč (textovú reprezentáciu) ku danému vstupnému obrázku. Výsledkom je matica ohodnotení každej jednej sekvencií. Matica ohodnotení má následné dva typy využitia. Používa sa pri trénovaní na výpočet straty (loss) a pri výslednom dekódovaní, ktoré obsahuje výsledný text zo vstupného obrázku. CTC funguje na základe troch konceptov a to *kódovanie textu*, *výpočet straty (loss)* a *dekódovanie*.

Kódovanie

Kódovanie textu slúži na riešenie problému, kedy je jeden charakter obsiahnutý vo viacerých sekvenciach. CTC tento problém rieši zoskupeným opakujúcich sa charakterov do jedného charakteru a to napr. ak by sme mali slovo 'hey', kde 'h' sa bude skladať z troch sekvencií, 'e' a 'y' z jednej tak by náš výsledok bez CTC predstavoval 'hhhey' ale s kódovaním by výsledok dosiahol požadovaného tvaru 'hey'. Tu ale nastáva problém pri slovách ktoré majú opakujúce sa znaky. Pri týchto slovách sa medzi opakujúce sa znaky vloží pseudo-znak nazývaný blank, ktorý môže byť označený ako '-'. Zoberme si slovo 'too', kde možné kódovanie by mohlo byť 'too-oo', ale zlé kódovanie by vyzeralo 't-ooo', čo by pri dekódovaní malo výsledok 'to'. CRNN je trénovaná aby predpovedala zakódovaný text.

Výpočet straty

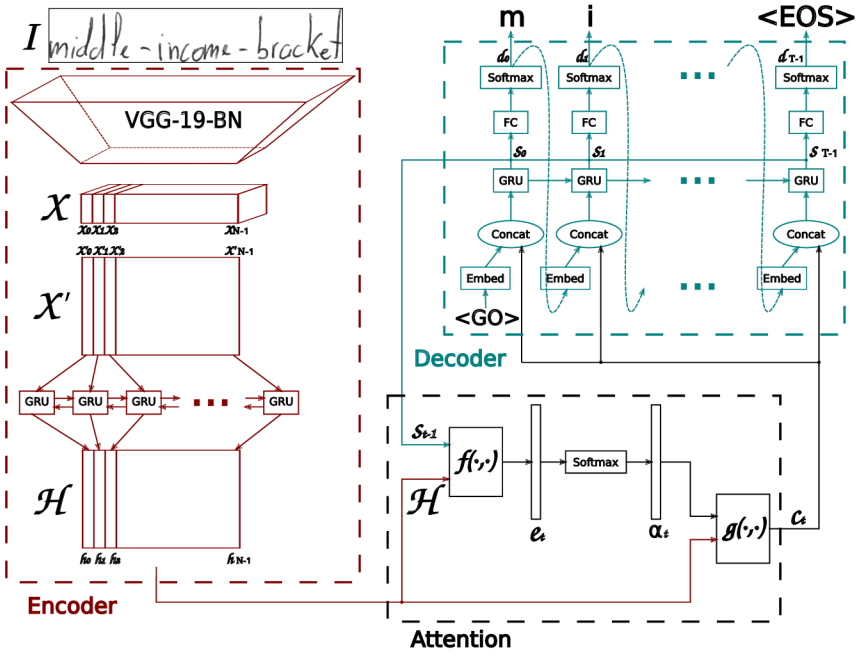
Výpočet straty, ang. loss, sa používa pri trénovaní neurónovej siete vzhľadom na vstupný obrázok a jeho textovú reprezentáciu (kľúč). Ako už bolo spomenuté výstupom CRNN je matica obsahujúca skóre pre každý znak v každej sekvencií. Na výpočet straty sa spočítajú všetky možné skóre z kľúčov, ktorých suma výsledných skóre pre každú sekvenciu sa rovná 1, týmto spôsobom nezáleží na akej pozícii v obrázku sa daný znak nachádza.

Dekódovanie

Po natrénovaní neurónovej siete ju chceme použiť na rozpoznávanie textu, ktorý ešte neurónová sieť nevidela, to znamená, že chceme vypočítať najpravdepodobnejší text vzhľadom na výstupnú maticu CRNN. Jednou z metód ktorú môžme použiť je preskúmavanie každého potenciálneho textového vstupu. Táto metóda nie je z výpočtového hľadiska veľmi praktická. Pre to sa používa metóda algoritmu najlepšej cesty. Algoritmus najlepšej cesty sa skladá z dvoch krokov a to výpočet najlepšej cesty na základe najväčšej pravdepodobností znaku v každej sekvencií a odstraňovanie *blank* a duplicitných znakov. Jeho výsledkom je požadovaný výsledný text.

2.4 Sequence to sequence

Napriek svojej flexibilite a sile sa hlboké neurónové siete dajú použiť iba na problémy, ktorých vstupy a ciele možno rozumne zakódovať vektormi s pevnou veľkosťou. Ide o výrazné obmedzenie, keďže mnohé dôležité problémy sú najlepšie vyjadrené sekvenciami, ktorých dĺžky nie sú predom známe ako napríklad reč, písaná veta, atď. Práve preto sa tento model používa na riešenie jazykových problémov akými sú strojové preklady. Táto architektúra



Obr. 2.5: Architektúra seq2seq modelu s použitím attention mechanizmu[6].

našla využitie ja pri riešení problému prepisu ručne písaného textu ako uviedli Kang a spol v publikácii z roku 2019[6], ktorá je znázorněna bližšie na Obrázku 2.5.

Architektúra sa skladá z troch hlavných častí a to *kódovač*, *dekódovač* a *attention mechanizmus*, rovnako ako sa používa pri bežných seq2seq modeloch.

Kódovač

Vstupom tejto siete je konvolučná neurónová sieť, v tomto prípade konkrétne VGG-19-BN[12], ktorej úlohou je získavanie čŕt zo vstupného textu I. Výstupom nám vznikne dvojrozmerná mapa čŕt X', ktorá môže byť označovaná ako sekvencia vektorov čŕt stĺpca. Po konvolučnej sieti nasleduje viacvrstvové obojsmerné GRU, ktorého vstup tvoria jednotlivé extrahované črty z konvolučnej siete, ktorá bude zahŕňať vzájomné informácie a ďalšie polohové informácie pre každý stĺpec a bude kódovať sekvenčnú povahu ručne písaného textu. Výstupom kódovaču je opäť dvojrozmerné pole vektorov H s rovnakou šírkou ako bolo X', ktoré bude slúžiť na výpočet *attetion* pri každej sekvencií.

Attention

Attention mechanizmus sa delí na dve hlavné mechanizmy a to *Content-based Attention* a *Location-based attention.*

Content-based attention je založený na hľadaní podobností medzi aktuálnym skrytým stavom a mapou čŕt výstupného obrázku, čo znamená, že vieme nájsť najviac korelované vektory čŕt v mape čŕt kódovača, ktoré slúžia na predpovedanie znaku v aktuálnom kroku[6].

$$\alpha_t = \text{Softmax}(e_t) \tag{2.5}$$

$$e_{t,i} = f(h_i, s_{t-1}) = w^T \tanh(Wh_i + Vs_{t-1} + b) \tag{2.6}$$

Kde α_t predstavuje masku vektora attencie v časovom kroku t a w, W, V a b sú natrénovatelné parametre, h_i je skrytý stav kodéra v časovom kroku $i \in \{0, 1, ...N-1\}$, s_t je

skrytý stav dekódera v časovom kroku $t \in \{0, 1, ...T-1\}$ a T predstavuje maximálnu dĺžku dekódovaných znakov. Po získaní *attention masky* je následne vypočítaný vektor najväčšej závislosti a to:

$$c_t = g(\alpha_t, H) = \sum_{i=0}^{N-1} \alpha_{ti} h_i \tag{2.7}$$

Hlavnou nevýhodou tejto metódy je, že očakáva, že pozičné informácie budú zakódované v extrahovaných črtách. Na prekonanie tohto problému môžme použiť Location-based attention.

Location-based attention explicitne berie do úvahy lokáciu informácií. To znamená že rozšírime Content-based attention na Location-based attention a to tak, že zohľadníme pozíciu na základe zarovnania vytvoreného pri predchádzajúcom kroku, ktorý je implementovaný následovne:

$$l_t = F * \alpha_{t-1} \tag{2.8}$$

a následne vymeníme rovnicu 3.2 za:

$$e_{t,i} = f'(h_i, s_{t-1}, l_t) = w^T \tanh(Wh_i + Vs_{t-1} + Ul_{t,i} + b) \tag{2.9}$$

Na dodatočné vylepšenie môžme nahradiť funkciu Softmax v 2.5 za sigmoid funkciu σ. Táto funkcia má za následok o trochu širšie zachytávanie okolia okolo písmena na vstupnom obrázku. Výsledok to nemení, keďže model dokáže vykonávať správne predikcie aj s úzkou oblasťou. Z pohľadu ľudského vnímania by bolo prospešné keby bola oblasť pokrytia širšia.

$$\alpha_{t,i} = \frac{\sigma(e_{t,i})}{\sum_{i=0}^{N} \sigma(e_{t,i})} \tag{2.10}$$

Dekóder

Dekodér je taktiež implementovaný za pomoci RNN, konkrétne sa skladá z jednosmerných viacvrstvových GRU, ktorý preberá vstupný vektor z kódovača a vytvára výslednú sekvenciu prepisu textu zo vstupného obrázku. Vstupom dekodéra je príslušný vektor v časovom kroku t, ktorý vznikol zreťazením embedding vektora predchádzajúceho časového kroku $\tilde{y}_{t-1}$ a vektora obsahujúceho kontextové informácie c_t. Predikcia každého kroku je vyjadrená následovne:

$$y_t = \arg\max(\omega(s_t)) \tag{2.11}$$

$\omega(\cdot)$ predstavuje lineárnu vrstvu. Následne použijeme príslušný embedding vector $\tilde{y}_t$ z *look-up* tabuľky, ktorá sa počas trénovania aktualizuje.

$$\tilde{y}_t = \text{Embedding}(y_t) \tag{2.12}$$

Dekodér vždy začína štartovacím signálom <GO> a končí dekódovanie keď narazí na koncový signál <EOS>, alebo prečerpá maximálny počet krokov T. Predchádzajúci embeding vektor a vektor kontextov sú zreťazené, a tak dostávame skrytý stav dekódera v konkrétnom čase s_t. Vďaka tomu GRU dostáva informácie o predchádzajúcom znaku a potencionálne najrelevantnejšej vizuálnej črty, čo vedie ku korektným predikciami.

$$s_t = \text{Decoder}([\tilde{y}_{t-1}, c_t], s_{t-1}) \tag{2.13}$$

$[\cdot,\cdot]$ predstavuje konkatenáciu dvoch vektorov.

Existujú dve techniky vďaka ktorým môžme vylepšiť proces dekódovania a to *multi-nominálne* dekódovanie a *label smoothing*. *Multi-nominálne* dekódovanie je inšpirované na základe článku[3]. Počas trénovania môžme namiesto výberu najväčšej pravdepodobnosti zo Softmax výstupu d_t v časovom kroku t, vybrať viacnásobné indexy, ktoré budú vybrané na základe multi-nominálneho rozdelia pravdepodobností zo Softmax výstupu d_t. Z dôvodu udržateľnosti jednoduchého modelu, vyberieme iba jeden náhodný index. Náhodný výber je založený na multi-nominálnej distribúcie pravdepodobnosti a tento index korešponduje ku jednému konkrétnemu znaku.

Label smoothing je mechanizmom regulácie, ktorá zabraňuje modelu aby vytváral prehnané predpovede. Model robí prispôsobivejší a zlepšuje generalizáciu, ktorú dosahuje tak, že podporuje model, aby mal pri predikcii vyššiu entropiu. Upravujeme kľúče tak aby čisté nuly a jednotky klasifikačných cieľov zamenil za následujúce ciele $\frac{\epsilon}{k}$ a $1 - \frac{k-1}{k}\epsilon$, kde ϵ je konštanta.

Kapitola 3

Prerezávanie

Prerezávanie, ang. pruning, je metóda, ktorou sa snažíme zmenšiť veľkosť a znížiť výpočetný čas neurónových sietí. Základným princípom prerezávania neurónových sietí je odstraňovanie nepotrebných častí neurónových sieti, ktoré ovplyvnia čo najmenej úspešnosť danej neurónovej siete.

3.1 Prerezávacie štruktúry

Neštrukturálne prerezávanie

Pri neštrukturálnom prerezávaní, sa zameriavame na odstránení jednotlivých váh medzi vrstvami. Jednotlivé váhy sa odstránia tým, že sa ich hodnota nastaví na nulu.

Toto riešenie má veľa výhod. Je jednoduché na implementáciu, kedže iba nastavujeme prerezávané váhy z pôvodnej hodnoty na nulu, má priamu podporu vo frameworku Pytorch[1] ale aj v iných frameworkoch. Najväčšou výhodou je že touto metódou sa prerezávajú najmenšie a zároveň najzákladnejšie prvky neurónovej siete, čo znamená, že máme väčšiu kontrolu nad tým ako ovplyvní prerezávanie vyslednú úspešnosť neurónovej siete.

Nevýhody, väčšina frameworkov nevie urýchliť sparse matice (to sú matice, ktoré sa skladajú zväčša z núl), čo znamená že vo výsledku sa neprejavuje zmena na rýchlosť neurónovej siete. Ďalším problém je ten, že veľkosť modelu je zhodná s veľkosťou pôvodného modelu. Jedine ako vieme zmenšiť veľkosť je s použitím kompresie prerezaného modelu. To znamená že táto metóda nemá žiadne výhody ak nepoužijeme špeciálny program alebo hardware, ktorý vie akcelerovať sprace matice ako napríklad *Neural Magic*[2].

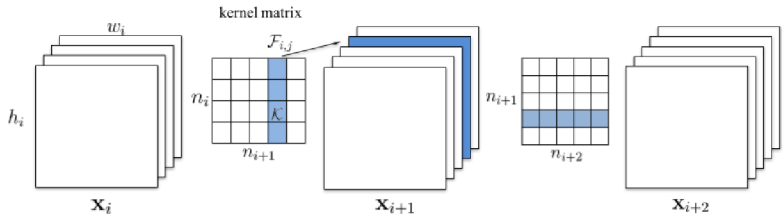
Štrukturálne prerezávanie

Štrukturálne prerezávanie, je typ prerezávania kde sa odstraňujú celé časti alebo filtre z neurónovej siete. Štrukturálne prerezávanie má teda vplyv na zmenu vstupných a výstupných tvarov vrstiev a váh matíc.

Výhodou tohto prístupu je, že výsledná sieť je menšia, kvôli znižovaniu počtu parametrov, ale vyžaduje aj menej systémových zdrojoch na výpočet.

Základom štrukturovaného prerezávania je prerezávanie filtrov konvolučných neurónových sietí[16]. Pri prerezávaní musíme dávať pozor na nasledujúce problémy. Pri prerezávaní celých filtrov vzniká problém, že v skutočnosti po prerezaní celého filtra sa neprereže len

[1] https://pytorch.org/docs/stable/nn.html
[2] https://github.com/neuralmagic



Obr. 3.1: Prerezávanie filtra má za následok odstránenie jeho zodpovedajúcej mapy prvkov a súvisiacich jadier v ďalšej vrstve[7].

samotný filter ale aj výsledok konvolúcie(konvolučná mapa), čo znamená, že pri prerezaní celých filtrov je možné v skutočnosti prerezať dvojnásobné množstvo parametrov aké sme pôvodne predpokladali.

Ďalším problémom je, keď sa odstráni celá vrstva tak výstupy predchádzajúcej vrstvy sú teraz nepripojené čo znamená že prerezanie jednej vrstvy môže zapríčiniť odstránenie všetkých predošlých vrstiev, ktoré nie sú inde prepojené, čo môže viesť k väčšej miere prerezania akú sme chceli. Preto pri prerezávaní filtrov by sme mali zvážiť presný počet skutočne prerezaných parametrov.

3.2 Kritéria prerezávania

Veľkosť váhy

Jedným z kritérií je prerezávanie na základe hodnoty váhy. Prerezávajú sa hodnoty, ktorých absolútna hodnota je najmenšia. Pri menších aktivačných váhach sa očakáva že ich dopad na výsledok bude menší, preto ich môžeme z neurónovej siete odstrániť.

Pri implementácii štruktúrovaného prerezávania sa na vyhodnocovanie filtrov používajú L1 alebo L2 normy. Pri L1 norme pre každý filter vypočítame sumu absolútnych váh filtra, následne tieto filtre zoradíme podľa veľkosti výsledných súm a prerežeme n najmenších filtrov a ich korešpondujúce mapy čŕt [7].

Gradient veľkosti

Ďalšie kritérium, ktoré sa používa je veľkosť gradientu. Už v 80. rokoch práca teoretizovala prostredníctvom Tayloroho rozkladu vplyv odstránenia parametra na stratu, že niektoré metriky odvodené zo spätného šírenia gradientu môžu poskytnúť spôsob, ako určiť ktoré parametre je možné prerezať bez následku na neurónovú sieť[16].

Modernejšia implementácia tohto kritéria prerezáva váhy s najnižšou absolútnou hodnotou (váha × gradient), vyhodnotené na dávke vstupov (mini batch).

Globálne a lokálne prerezávanie

Posledným kritériom ktorým sa budeme zaoberať je rozdiel medzi lokálnym a globálnym prerezávaním.

Globálne prerezávanie je typ prerezávania kde sa odstráni požadované množstvo n parametrov zo všetkých vrstiev naraz. Pri tomto type môže dôjsť ku kolapsu vrstvy[14].

Lokálne prerezávanie je prerezávanie, kde prerezávame parametre neurónovej siete pre každú vrstvu samostatne. Rieši problém kolapsu vrstiev, lebo každá vrstva má prerezaná pevný počet n parametrov.

3.3 Prerezávanie konvolučných filtrov pre zrýchlenie siete

Ak chceme dosiahnuť rýchlejšiu neurónovú sieť musíme použiť štrukturálne prerezávanie. Štrukturálne prerezávanie konvolučných neurónových sietí je založené na princípe odstraňovania celých filtrov a ich aktivačných máp. Po odstránení filtrov dostávame menšiu neurónovú sieť, čo má za následok menej operácií s pohyblivou rádovou čiarkou a tým pádom dochádza ku zrýchleniu siete. Táto metóda zasahuje priamo do štruktúry modelu a pre to výsledná presnosť siete je obvykle horšia ako pri neštrukturálnom prerezávanie kde prerezávame iba samostatné neuróny a nie celé štruktúry.

Filter norm pruning

Problémom tejto metódy je správny výber takých filtrov, ktoré majú čo najmenší vplyv na presnosť siete. Jedným z riešením predstavuje Hao Li a spol. vo svojej práci[7]. V svojej práci ohodnocujú filtre na základe ich L1/L2 noriem. Postup pri ich navrhovanej metóde je nasledovný. Najprv ohodnotia filtr každej vrstvy podľa ich L1 noriem, následne odstránia n počet najmenších filtrov, vytvoria nový model do ktorého nahrajú neprerezané váhy a na koniec model do trénujú na pôvodnú presnosť.

Particle filter pruning

Ďalšia práca od Sajid Anwar a spol[1] sa zdá byť podobná, ale hodnotenie je oveľa zložitejšie. Uchovávajú súbor N časticových filtrov, ktoré predstavujú N konvolučných filtrov, ktoré sa majú prerezať. Každej častici sa priradí skóre na základe presnosti siete na validačnej množine, keď filter reprezentovaný časticou nebol vymaskovaný. Potom sa na základe nového skóre vyberú nové prerezávacie masky. Keďže tento proces je výpočetne náročný, na meranie skóre častíc sa použila iba malá validačná množina.

Oracle pruning

Ideálna metóda ohodnocovania filtrov by bola metóda hrubej sily, čo znamená že by sme prerezávali každý filter samostatne a vyhodnocovali by sme na trénovacej množine jeho dopad na sieť. Túto metódu navrhla spoločnosť Nvidie, ktorá má prístup ku veľkému množstvu grafických kariet. Táto metóda sa nazýva *Oracle ranking*[8]. V tomto článku sa autori zaoberajú nájdením najlepšej metódy na odstraňovanie filtrov na základe zmeny výstupnej presnosti siete. Na zmeranie účinnosti iných metód použili výpočet *Spearmanov koeficient kolerácie* s oracle metódou. Spearmanov koeficient kolerácie je neparametrický test, ktorý sa používa na meranie miery asociácie medzi dvoma premennými $p = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$, kde d_i je rozdiel medzi dvoma hodnotami každého merania, n je počet meraní.

Prichádzajú s novou metódou hodnotenia neurónov založenou na Taylorovom rozšírení funkcie sieťových nákladov prvého rádu. Filtre h sa prerezávajú tak, že sa ich hodnoty nastavia na nulu. Na ohodnotenie jedného filtra h sa používa následujúca rovnica:

$$|\Delta C(h_i)| = |C(D, h_i = 0) - C(D, h_i)| \tag{3.1}$$

Kde $C(D, hi = 0)$ predstavuje veľkosť výstupu ak sú filtre h_i prerezané, a $C(D, hi)$ je veľkosť výstupu neprerezanej siete. Na aproximáciu $|\Delta C(h_i)|$ použijeme prvý stupeň Taylorového polynómu.

$$\Theta_{TE}(h_i) = |\Delta C(h_i)| = |C(D, h_i) - \frac{\delta C}{\delta h_i} h_i - C(D, h_i)| = |\frac{\delta C}{\delta h_i} h_i|. \tag{3.2}$$

Toto kritérium orezáva parametre, ktoré majú takmer plochý gradient nákladovej funkcie v závislosti od mapy príznakov h_i. Tento prístup vyžaduje akumuláciu súčinu aktivácie a gradientu nákladovej funkcie vzhľadom na aktiváciu, ktorý sa ľahko vypočíta z rovnakých výpočtov pre spätnú propagáciu. θ_{TE} sa vypočíta pre viac variantný výstup, ako je mapa príznakov, následovne:

$$\Theta_{TE}(z_l^{(k)}) = \Big|\frac{1}{M} \sum \frac{\delta C}{\delta z_{l,m}^{(k)}} z_{l,m}^{(k)}\Big|, \tag{3.3}$$

kde M je dĺžka vektorizovanej mapy prvkov. Pre minibatch s T > 1 príkladov je kritérium vypočítané pre každý príklad osobitne a spriemeruje sa pre T. Ohodnotenia každej vrstvy sa následne normalizujú pomocou L2 normy.

3.4 Prerezávanie rekurentných sietí

Výpočet redundancie

Za redundanciu bunky v LSTM vrstve sa považuje neprítomnosť odchýlok vo výstupoch dátach, ktoré sú produkované bunkami LSTM. Tento predpoklad je konkrétne podporený na základe prác od autorov Cartera a spol.[2], ktorá dáva podklad na to, aby sa predpokladalo, že určité LSTM jednotky sa aktivujú a produkujú určité tenzorové výstupy, keď sú v údajoch, ktoré spracúvajú, prítomné určité črty[10]. Dôsledná absencia zmien v odchýlkach výstupov, ktoré by jednotky mohli produkovať s rôznymi vstupmi, naznačujú, že všetky vzory výstupov jednotky sú si navzájom podobné, čo by mohlo ukázať, že sa nenaučili žiadne konkrétne črty, a preto by sme ich mohli prerezať.

Na nájdene zmeny vo výstupoch jednotlivých bunkách bol zvolený následujúci prístup:

$$UO_i = \sigma(h_i) \tag{3.4}$$

kde UO_i predstavuje jednu jednotku štandardnej odchýlky, σ predstavuje štandardnú odchýlku a h_i výstupný vektor LSTM jednotky. Vychádza sa z predpokladu, že ak sa konkrétnu črtu LSTM jednotka naučí, potom počas fázy predikcie sa fluktuácie vo výstupných tenzorových vektoroch, ktoré vytvorí, budú výrazne líšiť, ak vstup obsahuje uvedenú črtu, v porovnaní s tým, keď táto črta chýba.

Aby boli výstupy jednotiek zmysluplné, musia tiež vykazovať značné a nie nepatrné odchýlky vo výstupoch vektoroch. Ak by niektoré jednotky konzistentne prispievali odlišnými výstupnými údajmi, ktoré sú vzhľadom na iné jednotky veľmi malé, potom by sa dalo navrhnúť, že takéto jednotky by mohli fungovať v tandeme s inými, ale samy osebe významne neprispievajú k predpovediam siete.

Veľkosť výstupnej odchýlky jednotlivej jednotky na jeden sekvenčný vstup možno vypočítať vo vzťahu k ostatným jednoduchým vydelením jej UO súčtom všetkých UO:

$$UOM_x = \frac{UO_x}{\sum_{i=1}^{n} UO_i} \tag{3.5}$$

kde UOM je veľkosť jednej jednotky štandardnej odchýlky, n je počet jednotiek v LSTM vrstve, x a i sú špecifické jednotky LSTM bunky.

V ďalšom kroku je potreba zistiť, či sa vo výstupoch LSTM jednotky vyskytujú zmysluplné odchýlky. Na to sa najprv počas testovacej fázy vypočíta priemerná hodnota štandardnej odchýlky všetkých výstupov LSTM jednotky, aby sa zistilo, či v priemere dochádza k zmenám na rôznych výstupoch. Problémom je, že samotný priemer zachytí iba priemernú povahu veľkosti výstupných odchýlok, pričom by mal byť známy aj ich rozsah, aby sa zohľadnilo, ako sa rozkladajú odchýlky rôznych výstupov (t. j. na vytvorené predpoklady, čím sú vyššie - tým je menej pravdepodobné, že jednotka je nadbytočná, pretože vykazuje pri niektorých príležitostiach výrazne odlišné vzory činnosti a naopak). Dá sa to dosiahnuť zistením rozdielu medzi uvedenými priemermi a najvyššími a najnižšími hodnotami rozsahu, z ktorého boli vypočítané. Po výpočte nám vznikne dvoj rozmerný vektor, v ktorom možno znázorniť odchýlky vektorov LSTM jednotiek a ich rozptyl.

V následujúcej rovnici je zobrazené ako sa spomínaný vektor vypočíta pre i-tu jednotku vo vrstve LTSM a ako sa ukladá do jedného dátového bodu x_i.

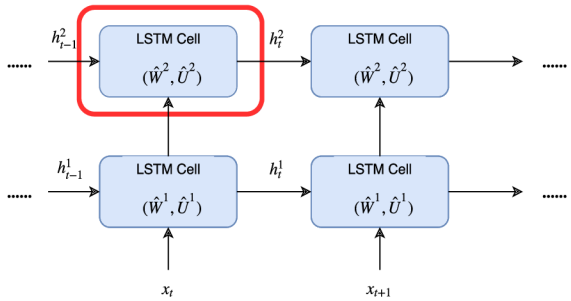
$$x_i = [mean(UOM_i) - min(UOM_i); max(UOM_i) - mean(UOM_i)] \tag{3.6}$$

kde $mean()$ je funkcia na výpočet priemeru, $min()$ a $max()$ sú funkcie na nájdenie najväčšej/najmenšej hodnoty.

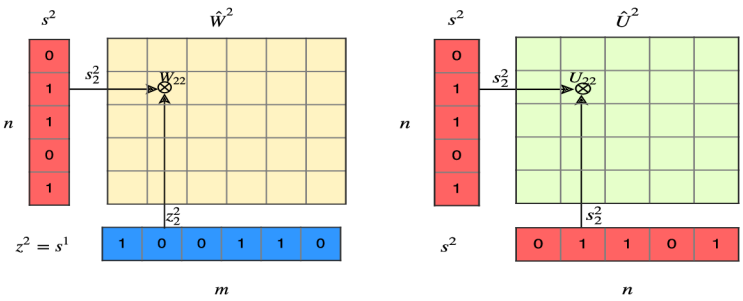
Prerezávanie na základe výberu neurónov

V práci od autorov Liangjian Wen, a spol.[18] sa zaoberajú metódou štrukturálneho prerezávania jednotlivých neurónov vo LSTM vrstvách. Na prerezávanie používajú mechanizmus výberu neurónov. Mechanizmus funguje na základe takzvaných brán/prepínačov, zobrazených na Obrázku 3.2 (b) a (c). Používame dva sety týchto brán, jeden typ brány, znázornený ako $z = z_i$, zabezpečuje prítomnosť vstupného neurónu i, kde $z_i \in \{0, 1\}$, a $|z| = m$ je počet vstupných neurónov druhého setu brány, znázornenej ako $s = s_j$, ktorá zabezpečuje prítomnosť skrytého neurónu j, kde $s_j \in \{0, 1\}$, a $|s| = n$ je počet skrytých neurónov. Týmto spôsobom w_{ij} riadi silu korelácie zo vstupného neurónu i na skrytý neurón j, zatiaľ čo z_i a s_j riadia prítomnosť neurónov a maska na w_{ij}, može byť vypočítaná následovne: $z_i \times s_j$. Takýmto mechanizmom brán môžme vyvolať štrukturovanú riedkosť (sparsity) na maticiach váhy. Ak by bolo $z_i = 0$, všetky skryté neuróny pripojené ku i budú odstránené, čo znamená, že i-tý stĺpec W budú samé nuly. A ak $s_j = 0$ tak by boli všetky riadky W nula. Preto zdieľaním binárnych masiek cez všetky brány môžeme získať štruktúrovanú riedkosť na matice váh.

Na modelovanie neistoty každej náhodnej brány z_i a s_j necháme $z_i \sim Bern(\pi_{zi})$ a $s_j \sim Bern(\pi_{sj})$, kde π_{zi} a π_{sj} sú parametre Bernoulliho rozdelenia a označujú pravdepodobnosť, že náhodná premenná z_i a s_j nadobudne hodnotu 1. Túto funkciu môžeme optimalizovať pomocou π_z a π_s minimalizovaním normy L0 matice váh, aby sme dosiahli výber neurónov. Regulácia L0 normy môže explicitne penalizovať nulové parametre modelov bez ďalších obmedzení a demonštruje vynikajúce výhody oproti regularizácie L1 normou. Pri učení riedkosti sa použije regularizácia L0 normy na váhach LSTM.



(a) stacked LSTMs



(b) input to hidden state (c) hidden to hidden state

Obr. 3.2: Ilustrácia navrhovaného mechanizmu brány. (a) zobrazuje ilustráciu viacvrstvovej LSTM bunky s naučitelnými parametrami $\hat{W}$ a $\hat{U}$, kde parametre v červenom boxe sú následne ilustrované v (b) a (c). (b) zobrazuje bránu z^2 a s^2, ktoré riadia maticu input-hidden parametrov $\hat{W}^2$, zatiaľ čo $z^2 = s^1$, pretože výstup W^1 je vstupom W^2. (c) zobrazuje bránu s^2, ktorá riadi maticu hidden-hidden váh U^2[18].

Kapitola 4

Návrh prerezávania

Na prerezávanie bude použitá neurónová sieť na rozpoznávanie ručne písaného textu pero-ocr, ktorá je navrhnutá na základe architektúry modelu CRNN 2.3. Neurónová sieť sa skladá z dvoch typov neurónových sietí a to z konvolučnej a rekurentnej neurónovej siete. Model obsahuje osem konvolučných vrstiev a štyri rekurentné vrstvy, ktoré sú podrobnejšie zobrazené v Tabuľke 4.1. Na normalizáciu sa využívajú sa dva typy normalizácie, *embeded* normalizácia a *filter response* normalizácia. Prerezávanie tejto siete bude vykonávané už na pred trénovanom modele. Tento model bol trénovaný s následujúcimi parametrami: počet iterácií=1000000, batch-size=128, optimizer=Adam, dropout-rate=0.10, max-line-width=2048, net="LSTM_BC_3_BLC_2_BFC_64_EMBED_32".

Program je navrhnutý tak aby fungoval na konkrétnu architektúru modelu pero-ocr. Cieľom je dosiahnuť čo najefektívnejšiu neurónovú sieť, chceme odstrániť taký počet parametrov aby bola sieť čo najrýchlejšia ale zároveň čo najviac si zachovala svoju pôvodnú presnosť. Program bude fungovať ako program s konzolovým výstupom, kde sa budú zadávať špecifikácie siete a prerezávania za pomocí argumentov. Prerezávanie bude fungovať v troch krokoch. V prvom kroku sa vytvorí mapa vrstiev a ich počet parametrov pôvodného modelu, v ďalšom kroku sa vytvorí nový model s menším počtom parametrov ako pôvodný model, podľa miery prerezania zadanej v argumente, následne sa vytvoria prerezávacie masky, ktoré obsahujú parametre, ktoré majú byť odstránené, a v poslednom kroku sa tieto parametre nakopírujú z pôvodného modelu do nového modelu.

4.1 Prerezávanie konvolučnej časti

Pri prerezávaní konvolučnej časti sme zvolili štrukturálne prerezávanie, založené na odstraňovaní konvolučných filtrov a tým aj ich aktivačných máp. Pri odstraňovaní filtrov budeme skúmať aký spôsob ohodnotenia filtrov má čo najmenší dopad na výslednú presnosť siete. Na klasifikovanie týchto filtrov, ktoré majú byť odstránené sú použité tri spôsoby ohodnotení a to l1 norma, l2 norma na základe článku od Hao Li, a spol.[7] a nami navrhnutá metóda odstraňovania filtrov na základe ich štandardnej odchýlky. Filtre ktorých štandardná odchýlka je nulová ako napríklad filter plný jednotiek ako je vyobrazené na Obrázku 4.1a, neprodukuje žiadny prospešný výsledok, je to iba kópia vstupu. Ale filter na Obrázku 4.1b, má veľkú štandardnú odchýlku a jeho výsledkom sú vstupy so šikmými hranami, čo môže predstavovať napríklad časť písmena A. Po ohodnotení sa filtre v každej vrstve zoradia podľa ich veľkostí a to od najväčšieho po najmenšie ohodnotenie a vymaže sa určité percento najmenších filtrov podľa toho koľko parametrov chceme prerezávať.

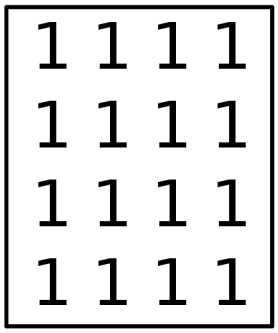
Typ vrstvy	Parametre
Vstup	3 x W x 40px
Embedding	num_emb=862, emb_dim=32
Konvolúcia 2d	out_filtre=64, k:3x3, s:1, p:1
Konvolúcia 2d	out_filtre=64, k:3x3, s:1, p:1
Max Pool 2d	k: 2x2, s: 2, p:0
FilterResponseNorm2d	-
Konvolúcia 2d	out_filtre=128, k:3x3, s:1, p:1
Konvolúcia 2d	out_filtre=128, k:3x3, s:1, p:1
Max Pool 2d	k: 2x2, s: 2, p:0
FilterResponseNorm2d	-
Konvolúcia 2d	out_filtre=256, k:3x3, s:1, p:1
Konvolúcia 2d	out_filtre=256, k:3x3, s:1, p:1
Max Pool 2d	k: 2x2, s: 2, p:0
FilterResponseNorm2d	-
Konvolúcia 2d	out_filtre=256, k:3x3, s:1, p:1
Konvolúcia 2d	out_filtre=256, k:5x1, s:1
Bidirectional-LSTM	hidden_size=512, input_size=256, počet vrstiev=2
Bidirectional-LSTM	hidden_size=512, input_size=256, počet vrstiev=2
Bidirectional-LSTM	hidden_size=512, input_size=256, počet vrstiev=2
Bidirectional-LSTM	hidden_size=512, input_size=256, počet vrstiev=1
EmbededNorm 1d	-
Konvolúcia 1d	out_filtre=1024, k:3, s:1, p:1
Konvolúcia 1d	out_filtre=512, k:3, s:1, p:1
Transkripcia	-

Tabuľka 4.1: Zobrazenie architektúry modelu pero-ocr, kde k predstavuje rozmery filtrov (kernelov), s predstavuj veľkosť kroku (stride) a p predstavuje padding.

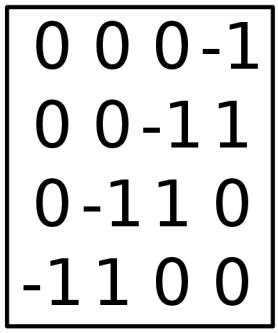
Pri prerezávaní konvolučnej časti budeme porovnávať dva druhy prerezávania, čo sa týka počtu prerezaných parametrov a to *lokálne* a *globálne* prerezávanie. Pri lokálnom prerezávaní budú všetky vrstvy prerezané jednou konštantou, čo znamená, že veľkosť vrstiev sa zmení v každej vrstve o rovnaké percento prerezania. Pri globálnom sa prerežú jednou konštantou všetky filtre vo všetkých vrstvách naraz. Pri globálnom prerezávaní budem používať rovnaké typy ohodnotenia ako aj pri lokálnom prerezávaní. Následne budeme prerezávať samotné vrstvy raz za čas a zisťovať, mieru prerezania každej vrstvy samostatne.

4.2 Prerezávanie rekurentnej časti

Prerezávaná rekurentná časť sa skladá z LSTM architektúry, ktorú tvoria štyri obojsmerné LSTM vrstvy. Tri tieto vrstvy sú dvoj vrstvové a posledná vrstva je jedno vrstvová. Rovnako ako to bolo pri konvolučnej vrstve aj táto vrstva bude prerezávaná štrukturálne. Vstup prvej LSTM vrstvy úzko súvisí s výstupom poslednej konvolučnej vrstvy. Na to aby sme mohli prerezávať rekurentnú časť musíme najprv prerezať poslednú vrstvu konvolučnej časti. Počet výstupných parametrov poslednej konvolučnej vrstve sa priamo odvíja od veľkosti vstupného vektora a veľkosti skrytého vektora. Naučitelné parametre sú uložené v dvoch kategóriach a to ako *vstupno-skryté* váhy a *skryto-skryté* váhy. Vstupno-skryté váhy sa skla-

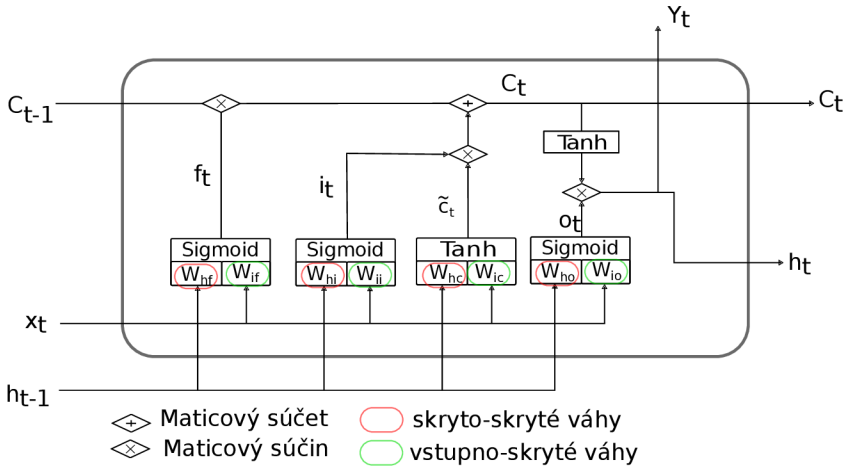


(a) Filter, ktorý kopíruje vstup



(b) Filter, ktorý vyhľadáva šikmé hrany

Obr. 4.1: Zobrazenie dvoch filtrov z rozdielnými štandardnými odchýlkami



Obr. 4.2: Ukážka LSTM bunky a jej naučitelných váh, ktoré budeme prerezávať

dajú zo vstupných váh štyroch LSTM brán a to ($W_{ii}|W_{if}|W_{ig}|W_{io}$) s rozmermi (4*veľkosť skrytého stavu, veľkosť vstupného stavu). Skryto-skryté váhy sa skladajú zo skrytých váh štyroch brán ($W_{hi}|W_{hf}|W_{hg}|W_{ho}$), majú rozmer (4*veľkosť skrytého stavu, veľkosť skrytého stavu). Toto sú váhy ktoré budeme prerezávať.

Miera prerezania je určená veľkosťou výstupných filtrov konvolučnej časti, teda jej poslednej vrstvy, čo znamená.. Keďže používame obojsmerné LSTM bunky tak veľkosť skrytého stavu (počet features) je *Veľkosť_vstupu*/2. Týmto sa nám menia veľkosti oboch osí naučiteľných váh. Pri prerezávaní y-novej osi na ohodnocovanie váh budeme používať štandardnú odchýlku[15]. Vyberie sa rozmedzie váh a to $[\frac{n}{2}, X - \frac{n}{2}]$, kde X predstavuje počet parametrov pôvodného modelu a n predstavuje počet parametrov mínus počet parametrov, ktoré majú byť prerezané, tak ako je to popísané v článku[15]. Týmto dosiahneme správny rozmer y-novej osi ale stále máme zlý rozmer x-ovej osi, ktorý predstavuje počet LSTM buniek, ktorý sa priamo viaže na veľkosť skrytého stavu. Pre každý vstup/skrytý stav (y-nová os) zoradíme veľkosti váh a odstránime percento najnižších váh. Týmto dosiahneme správny rozmer nového modelu. Model následne do trénujeme tak aby chybovosť prerezaného modelu bola čo najbližšie ku chybovosti pôvodného modelu.

Kapitola 5

Implementácia

Táto kapitola sa zameriava na popis implementácie programu pre prerezávanie neurónovej siete, ktorej návrh bol popísaný v Kapitole 4, na použité nástroje pri vytváraní aplikácie a použité knižnice.

5.1 Použité nástroje a knižnice

Programové riešenie ako aj neurónová sieť na rozpoznávanie ručne písaného textu je implementované v jazyku *Python* (verzia 3.8.10)[1] ako program s konzolovým výstupom. Na modelovanie, trénovanie a testovanie modelov neurónových sietí bola použitá high-level knižnica *Pytorch*[2] (verzia 1.11.0+cu113) za použitia Cuda driverov (verzia 11.3). Následne je využitá knižnica *Numpy*[3], ktorá slúžila na matematické výpočty a knižnica *Matplotlib*[4], ktorá nám umožňovala vizualizovať výsledky experimentov.

Na rozpoznávanie ručne písaného textu je využitá konvolučná rekurentná neurónová sieť dodaná vedúcim práce. Trénovanie a výpočty prebiehali lokálne na zariadení s grafickou kartou RTX 2060 6GB, procesorom Intel Core I5-8400 a 16GB RAM. Implementácia prebiehala na OS Linux, konkrétne na Ubuntu verzie 21.04 LTS.

5.2 Vytváranie modelu

Na začiatku je potreba vytvoriť model a nahrať do neho váhy modelu, ktorý chceme prerezať. Za vytvorenie a nahratie váh do modelu je zodpovedná funkcia *load_checkpoint* nachádzajúca sa v súbore *export_model_prune.py*. Táta funkcia vytvorí model na základe definície modelu, ktorá je uložená v súbore *ocr_engine.json*. Z tadiaľ sa vytvorí class *NetRecurrent*. Následne sa nahrajú váhy do vytvoreného modelu pomocou funkcie *load_weights*. Pri vytváraní prerezávaného modely je potreba zmeniť parametre pri určitých vrstvách pri definícii modelu. Pri definícii sa pozmenia výstupné parametre a to tak že sa vynásobia mierou prerezania (čo predstavuje percento zachovaných parametrov) a následne sú tieto parametre nastavené ako vstup následujúcej vrstvy. Parametre prerezania sú uložené ako pole float hodnôt, ktoré zadá uživateľ vo forme argumentov. Kde každá položka poľa zobrazuje mieru prerezania každej konvolučnej vrstvy. Na prerezávanie konvolučnej časti sa

[1] https://www.python.org/
[2] https://pytorch.org/
[3] https://numpy.org/doc/stable/index.html
[4] https://matplotlib.org/

používa iba prvých sedem vrstiev z dôvodu toho keby sme prerezali aj ôsmu vrstvu, ktorej výstup tvorí vstup rekurentnej časti, tak by sme prerezávali aj rekurentnú časť. Na konci programu sa uloží nový prerezaný model aj spolu so súborom *prune.json*, ktorý ukladá mieru prerezania pre jednotlivé vrstvy. Tento súbor slúži pri inicializácii prerezaného modelu.

5.3 Implementácia prerezania

Prerezávanie konvolučnej tak aj rekurentnej časti prebieha v troch krokoch. V provom kroku je vypočítaný počet parametrov nového modelu, v druhom kroku sa vypočítajú ohodnotenia parametrov, pri konvolučnej časti sú ohodnocované jednotlivé výstupné filtre, a pri rekurentnej sú ohodnocované váhy. V tretom kroku sa prerezané váhy nahrajú zo starého modelu do nového a model sa uloží spolu so súborom v ktorom sú uložené hodnoty prerezania.

Prvý krok je for cyklus, ktorý prejde všetkými vrstvami a podľa miery prerezania uloží do premennej počet neprerezaných parametrov vrstvy.

V druhom kroku dochádza ku vytváranú másk. Maska slúži na to aby sme vedeli, ktoré parametre majú byť zachované a ktoré odstránené. Maska je vytvorené pre každú vrstvu a má rovnakú veľkosť ako pôvodný model. V tejto maske nula predstavuje hodnoty, ktoré sa majú prerezať a jednotka hodnoty ktoré ostávajú nezmenené. Pri konvolučnej časti používame tri metriky na výpočet ohodnotenia filtrov a to l1 a l2 normu a štandardnú odchýlku, ktoré si užívateľ zvolí na základe argumentu *-n*. L1 ako aj L2 norma filtrov sa vypočítava z ich absolútnych hodnôt a to z dôvodu toho, že nás zaujíma veľkosť daného vektora, čiže záporné hodnoty majú rovnaké opodstatnenie ako kladné a zaujíma nás iba veľkosť.

Konvolučná časť

L1 norma sa vypočíta ako súčet absolútnych hodnôt vektorov. V skutočnosti je norma výpočtom Manhattanovej vzdialenosti od začiatku vektorového priestoru. V programe je vypočítaná pomocou funkcie *np.sum()* z váh vrstvy. Keďže váhy konvolučnej vrstvy obsahujú 4 dimenzie (vstupné filtre|výstupné filtre|y-nová os filtra|x-ová os filtra) na výpočet používame iba výstupné filtre preto pri výpočte použijeme prepínač axis=(1,2,3).

L2 norma počíta vzdialenosť súradnice vektora od začiatku vektorového priestoru. Ako taká je známa aj ako euklidovská norma, pretože sa počíta ako euklidovská vzdialenosť od začiatku. Výsledkom je kladná hodnota vzdialenosti. Norma L2 sa vypočíta ako odmocnina zo súčtu umocnených hodnôt vektora. Na výpočet bola použitá funkcia *torch.linalg.norm()* rovnako ako pri L1 norme sa použije prepínač axis=(1,2,3).

Štandardná odchýlka je štatistika, ktorá meria rozptyl súboru údajov vzhľadom na jeho priemer a vypočíta sa ako odmocnina z rozptylu. Štandardná odchýlka sa vypočíta ako odmocnina z rozptylu určením odchýlky každého dátového bodu vzhľadom na priemer. Ak sú dátové body ďalej od priemeru, je v rámci súboru údajov väčšia odchýlka; čím sú teda údaje rozptýlené, tým je štandardná odchýlka vyššia. Na výpočet bola použitá funkcia *np.std()* taktiež s parametrom axis=(1,2,3). Štandardná odchýlka je vypočítaná následujúcim spôsobom: vstupné dáta sa rozložia do jednorozmerného vektora za sebou a následne sa vypočítajú cez vzorec $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$, kde N je počet prvkov, x hodnoty vektora a μ priemer hodnôt zo vstupného vektora.

Po ohodnotení sa filtre zoradia od najväčšieho ohodnotenia po najmenšie a do premennej sa uloží počet najviac ohodnotených filtrov podľa počtu parametrov vypočítaných v prvom kroku. Následne sa vytvorí maska, ktorú nainicializujeme nulami na veľkosť pôvod-

ného modelu. Po inicializácii nastavíme hodnoty na jedna tam kde boli ohodnotené filtre v rozsahu spomínaného v Sekcii 4.1. Tento proces opakujeme pre každú konvolučnú vrstvu.

V poslednom kroku uložíme vyhradené filtre do nového modelu. Filtre sa budú kopírovať z pôvodného modelu a nakopírujú sa iba tie filtre, ktoré neboli prerezané. Vytvoria sa dve premenné, do ktorých si nahráme indexy filtrov masky konkrétnej vrstvy, ktoré majú byť uložené do nového modelu, stým že máme dve masky. Prvá maska sa zaoberá kopírovaním vstupných filtrov a druhá maska, kopírovaním výstupných filtrov. Po nakopírovaní váh do nového modelu sa nastaví maska výstupných filtrov ako maska vstupných filtrov nadchádzajúcej vrstvy.

Pri konvolučných vrstvách je ešte jeden typ naučitelných parametrov a to *bias*. Bias je jednorozmerný vektor, ktorého veľkosť je rovnaká ako počet výstupných filtrov danej vrstvy, práve pre to môžme použiť rovnakú masku pri ukladaní ako pri konvolučných váhach.

Po konvolučnej vrstve prichádza normalizácia vo forme *FilterResponseNorm2d*. FilterResponseNorm2d je typ batch normalizácie, ktorý kombinuje normalizáciu a aktivačnú funkciu, ktorú možno použiť ako náhradu za iné normalizácie a aktivácie. Táto vrstva obsahujú tri naučitelné hodnoty a to: *scale*, *tau* a *bias*. Všetky hodnoty sú rovnakých rozmerov. Jediná hodnota, ktorá sa mení oproti pôvodnému modelu je počet výstupných konvolučných filtrov. Na túto nezhodu použijeme masku použitú v predchádzajúcej konvolučnej vrstve.

Rekuretná časť

Pri rekurentných vrstvách máme viac naučiteľných parametrov a pre to musíme vytvárať aj viac masiek. Rovnako ako aj pri konvolučnej časti máme tri rovnaké kroky.

V prvom kroku si uložíme veľkosť vlastností shrytého stavu *h hidden_size* a veľkosť očakávaných vlastností vstupu *input_size*[5] pre každú vrstvu LSTM.

V druhom kroku sa vytvoria masky, ktoré slúžia na odstraňovanie váh. Máme dva hlavné typy váh a to *input-hidden* a *hidden-hidden* váhy. Z týhto váh budeme vytvárať masky, ktoré následne použijem aj na ich reverzné váhy, ktoré obsahuje obojsmerné LSTM. *Input-hidden* váhy majú dva rozmery (4*hidden_size, input_size), pri prerezávaní sa menia oba rozmery. Ako prvé vytvoríme masku, ktorá bude prerezávať hodnoty na x-ovej osi. Na toto slúži funkcia *create_mask*, táto funkcia zoberie každý riadok vstupného vektora, v našom prípade *weight_ih_l0*, a vyberie n najväčších váh. Po výberé váh nasleduje výber pre samostatné vstupy. Ako kritérium sa používa štandardná odchýlka, ktorá je vypočítaná pomocou *numpy* funkcie *np.std()*, kde sa následne vyberie rozmedzie parametrov, ktoré je popísané v Kapitole 4.2.

V posledom kroku sa prerezané váhy nahrajú do nového modelu, kde sa masky prevedú na pozície váh, ktoré sa majú zachovať a uložia sa do premennej *idx_ih_4h*, *idx_ih_i* pre *input-hidden* váhy a *idx_hh_4h*, *idx_hh_i* pre *hidden-hidden* váhy. Na odstránenie parametrov x-ovej osi je použitá funkcia *remove_col*, ktorá na základe masky vytvorenej v prvom kroku odstráni parametre, ktoré sa v novom modeli nemajú nachádzať. Tieto váhy sa uložia do pomocnej premennej a následne sa do tejto premennej nakopíruje y-nová os váh za použitia masky *idx_hh_4h*/*idx_ih_4h*, ktorá v sebe ukladá pozície neprerezaných parametrov.

Reverzné parametre a parametre vyššej vrstvy **_l1*, **_l2* sa kopírujú podľa masky parametrov **_l0* vytvorenej pre každú vrstvu v druhom kroku.

[5] https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Kapitola 6

Výsledky a experimenty

Táto kapitola sa venuje popisu rôznych experimentov prerezávania tak ako na konvolučnej tak aj na rekurentnej časti. Experimenty sú vykonávané na už naučenom modely, ktorý mi bol poskytnutý vedúcim práce a bol učený s nasledujúcimi parametrami: $learningrate = 0.000025$, $iterations = 1000000$, $optimizer = Adam$, $dropoutrate = 0.10$, $batchsize = 128$. Pri experimentoch bola kladená priorita na výsledná rýchlosť siete na základe udržania si čo najväčšej presnosti siete. Na ohodnotenie presnosti neurónovej siete používame metriku *Character Error Rate (CER)*. Pri rozpoznávaní textu nám môžu vniknúť tri typy chýb:

- **Substitúcia**: nastáva, keď sa nachádza na konkrétnej pozícii iný znak,
- **Vymazanie znaku**: nastáva, keď nám chýba znak,
- **Vkladanie znaku**: nastáva, keď máme vložený znak navyše kde by nemal byť.

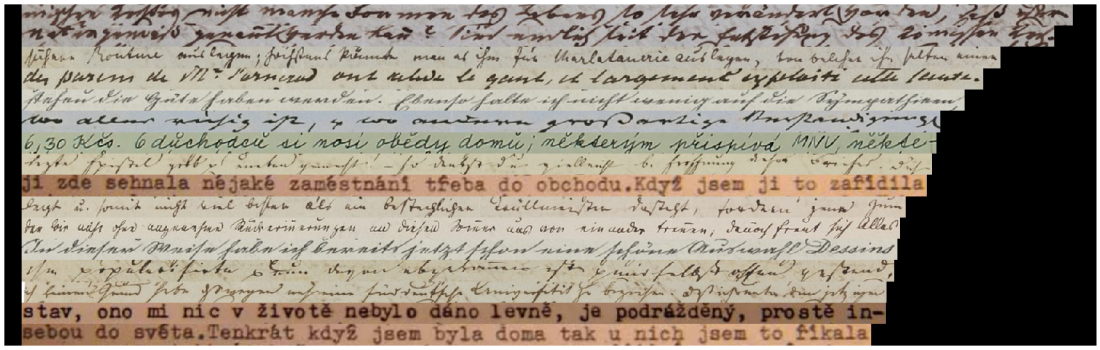
Výpočet *CER* je založený na koncepte Levenshteinovej vzdialenosti, kde počítame minimálny počet operácií na úrovni znakov, ktoré sú potrebné na transformáciu základného textu pravdy (referenčného textu) do výstupu OCR. Výpočet je znázornený v následujúcej rovnici: $CER = \frac{S+D+I}{N}$, kde S je počet substitúcií, D je počet vymazaní, I počet vložení a N je počet znakov v referenčnom texte (v texte pravdy). Výsledok je uvedený v percentách a znázorňuje počet percent chybných/chýbajúcich/zvyškových znakov. Ak je výsledok 0% *CER*, vtedy neurónová sieť dosahuje 100% presnosť. Pri nenormalizovanej *CER* môže táto hodnota nadobudnúť aj nad 100%, najmä ak vo výsledku sa nám objavia mnohé zvyškové znaky. Pri výpočte celkovej presnosti sa vypočítajú presnosti pre každý riadok a tie sa následne spriemerujú.

Experimenty boli testované na skripte *test_net.py*, ktorý bol upravený tak aby umožnil načítavať prerezanú sieť. Po prerezaní sme sa snažili dosiahnuť presnosť siete čo najbližšie ku pôvodnému modelu. Výstupom tohto skriptu je chybovosť siete CER a dĺžka trvania výpočtu cez celú testovaciu sadu. Výsledky rýchlosti siete sú normalizované na 1000 riadkov textu. Trénovanie a testovanie prebiehalo na dátovej sade poskytnutej vedúcim práce.

6.1 Dátová sada

Ako aj na testovanie tak aj na trénovanie modelu je použitá databáza poskytnutá vedúcim práce. Táto databáza sa skladá z viacerých verejne dostupných databáz ale aj z vlastných výťažkov skupiny PERO. Skladá sa z nasledujúcich prameňov: bentham[1], Brno_HWR[2],

[1] https://zenodo.org/record/44519#.YmwtAhxBwUE
[2] https://pero.fit.vutbr.cz/handwritten_dataset



Obr. 6.1: Ukážka dát z použitého datasetu, konkrétne z testovacej sady

České dopisy[3], České kroniky, ktoré sú pozbierané vety z archívov z 20 storočia, saint_gall sa skladá zo starších manuskriptov, Read dataset[4], Parzival dataset[5] a dát pozbieraných skupinou Pero[6], ktoré obsahujú dáta z archívou uložených v Moravsko zemskej knižnici.

Dátová sada je rozdelená na dva typy sád a to na trénovaciu a testovaciu. Trénovacia sada obsahuje 629558 riadkov textu zatiaľ čo testovacia sada obsahuje 5762 riadkov textu.

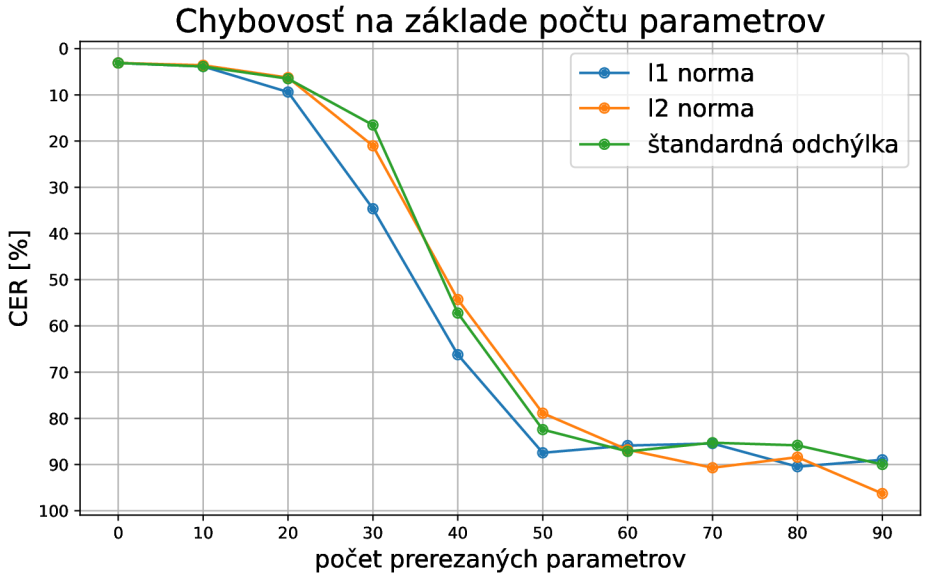
6.2 Prerezávanie konvolučnej časti

Ako prvý experiment sme prerezávali konvolučnú časť s globálnym výberom filtrom na základe ich L1 normy. Pri tomto spôsobe prerezávanie dochádzalo ku kolapse vrstvy už pri prerezaní 30% parametrov, čo vo finálnom výsledku viedlo ku vysokým chybovostiam *CER*. Z tohto dôvodu sme túto metódu ďalej už nerozvíjali a začali sme sa sústreďovať na lokálne prerezávanie.

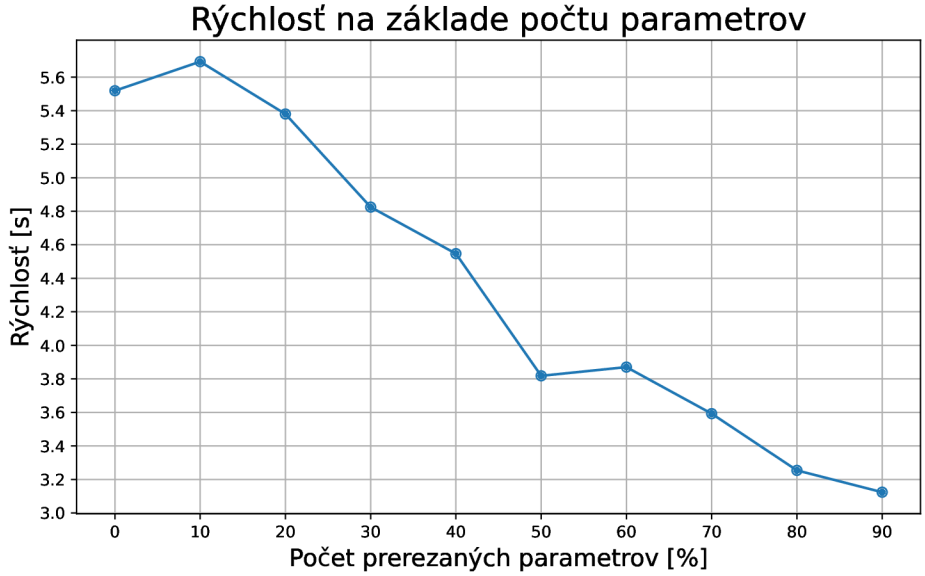
Pri lokálnom prerezávaní sme porovnávali tri rôzne metódy ohodnotenia filtrov a to: L1, L2 normu a štandardnú odchýlku filtrov. L1 norma viedla ku najhorším výsledkom zatiaľ čo L2 norma a štandardná odchýlka skončili veľmi podobne stým, že pri prerezaniach menších počtov bola lepšia L2 norma ale pri prerezávaní väčších množstiev parametrov dosahovala lepšie výsledky nami navrhnutá štandardná odchýlka viď Obrázok 6.2. Bolo vykonaných 9 experimentov po krokoch veľkých 10% výsledného počtu výstupných filtrov. Pri 80% siete vzniká veľký úpadok na výslednej presnosti siete. Od 50% prerezania siete začína produkovať takmer náhodne výsledky ktoré už takmer vôbec nekorešpondujú so vstupnými slovami ako môžme vidieť na Obrázku 6.2. Pri prerezávaní sa zmenšuje počet parametrov v sieti, čo vedie ku menším model, ktoré sú následne aj rýchlejšie. Rýchlosť modelu nie je priamo úmerná počtu parametrov. Tento jav môže nastať z dôvodu paralelizácie a tomu, že grafické karty pracujú lepšie so štvorcovými maticami, čo môžme vidieť aj na Obrázku 6.3 pri prerezaní 50%, vtedy je počet filtrov rovný druhej mocnine a vidíme výrazné zrýchlenie siete. Výsledná presnosť teda ani použitá metóda prerezávania nemá vplyv na rýchlosť neurónovej siete.

Po lokálnom prerezávaní sme začali skúmať citlivosť jednotlivých vrstiev na prerezanie. Aby sme pochopili citlivosť každej vrstvy, prerezávame každú vrstvu nezávisle a hodnotíme presnosť výslednej prerezanej siete na testovacej sade. Po experimentoch sme zistili, že men-

[3] https://www.muni.cz/vyzkum/publikace/1084855
[4] https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-us/abapread_dataset.htm
[5] https://fki.tic.heia-fr.ch/databases/parzival-database
[6] https://pero.fit.vutbr.cz/

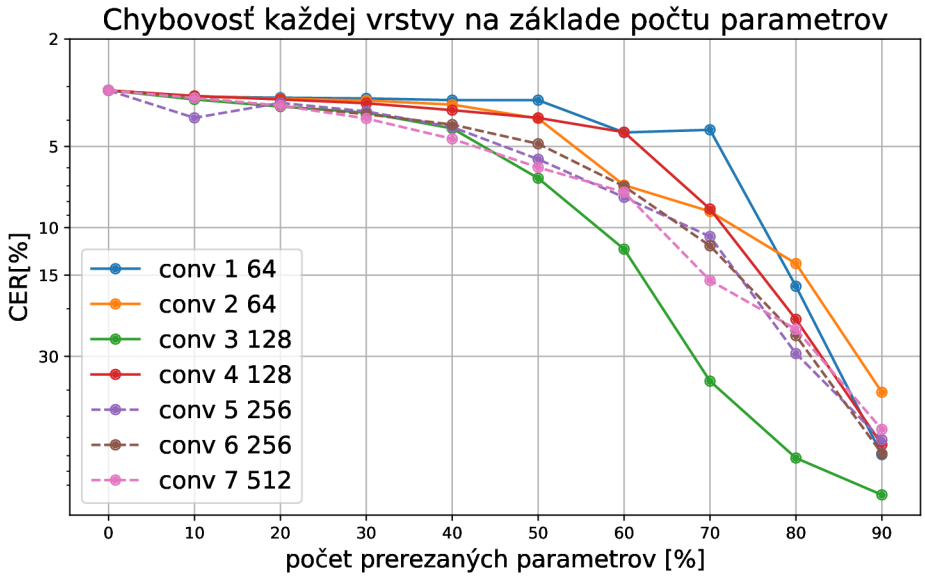


Obr. 6.2: Závislosť medzi počtom prerezaných parametrov konvolučnje časti a výslednou chybovosťou siete.



Obr. 6.3: Závislosť medze počtom parametrov a rýchlosťou siete.

šie vrstvy sú o trochu menej náchylné na prerezávanie ako väčšie vrstvy. Z týchto zistení sme následne vyvodili koľko a kde budeme prerezávať filtre tak aby sme dosiahli čo najoptimálnejší výsledok. Filtre boli prerezávané na základe l2 normy. Pri vrstvách citlivých na prerezávanie ako napr. conv 5 vrstva viď. Obrázok 6.4 prerezávame menšie množstvo parametrov ako pri vrstvách, ktorých chybovosť začína upadať pri nižších počtoch parametrov. Vo výsledku sme prerezali najväčšie množstvo parametrov každej vrstvy samostatne pred jej výraznou stratou na presnosti. Týmto sme dosiahli lepšie výsledky ako keby sme všetky vrstvy prerezávali rovnakou konštantou. Prerezané parametre boli následujúce: [50%, 50%, 40%, 40%, 30%, 30%, 80%], zoradené od najmenšej vrstvy po najväčšiu (conv 1 - conv 7).



Obr. 6.4: Porovnanie jednotlivých vrstiev a ich citlivosti na prerezávanie. Filtre ktoré majú väčší sklon sú náchylnejšie na prerezanie.

Takto prerezaná sieť dosiahla rýchlosť o trochy rýchlejšiu ako to bolo pri lokálnom prerezávaní o 40% čo predstavuje 25.6 sekundy a zároveň mala o takmer polovicu lepšiu chybovosť siete a to 31.3% CER.

6.3 Prerezávanie LSTM

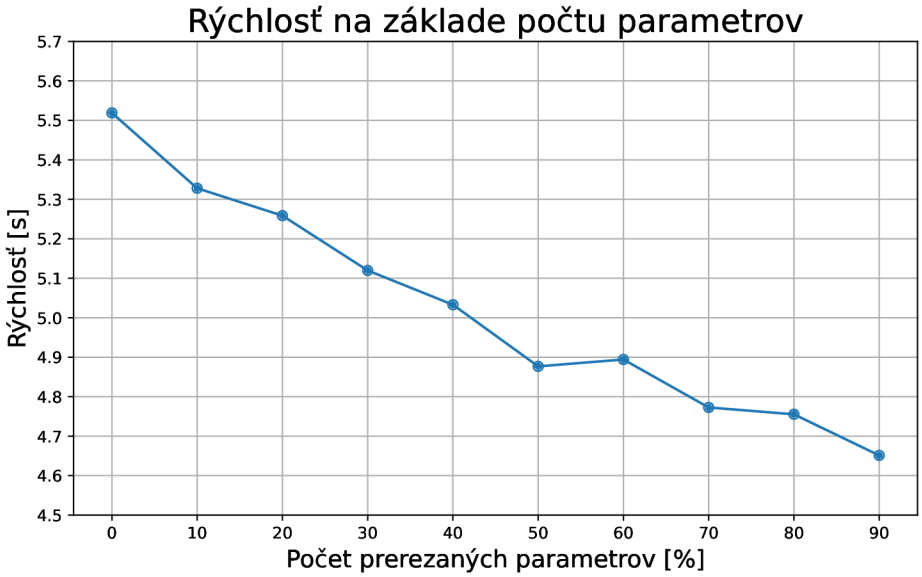
Experimenty nad LSTM vrstvou boli robené na základe lokálneho prerezávania. čo sa týka počtu parametrov tak LSTM vrstvy obsahujú oveľa viac parametrov ako to bolo pri konvolučných vrstvách. Bolo robených deväť experimentoch od 10% až po 90% prerezaných parametrov. Po prerezaní nám vyšli vysoké chybovosti, ktoré siahali nad 100% CER pri všetkých experimentoch a chybovosť sa znižovala pri prerezaní väčšieho množstva parametrov.

Pri prerezávaní sa nám sieť opäť výrazne zrýchlila pri 50% percentách prerezaných parametrov ale aj už pri 10% môžme vidieť výrazné zrýchlenie, ktoré následovne rastie relatívne lineárne. Oproti konvolučnej časti máme väčšie zrýchlenia do 30% prerezaných parametrov potom má lepšie výsledky konvolučné prerezávanie aj napriek tomu, že prerezávame väčší počet parametrov.

6.4 Trénovanie modelu

Potom, čo sme prerezali predtrénovaný model sa nám výrazne zhoršila úspešnosť siete. Na obnovenie úspešnosti prerezávaný model znova trénujeme. Na následne trénovanie sa použil skript *train_pytorch_ocr_prune.py* s následujucími parametrami batch-size=16, dropout-rate=0.10, optimizer=Adam. Neurónové modely boli trénované na dátovej sade 6.1, ktorá obsahovala 629558 riadkov textu.

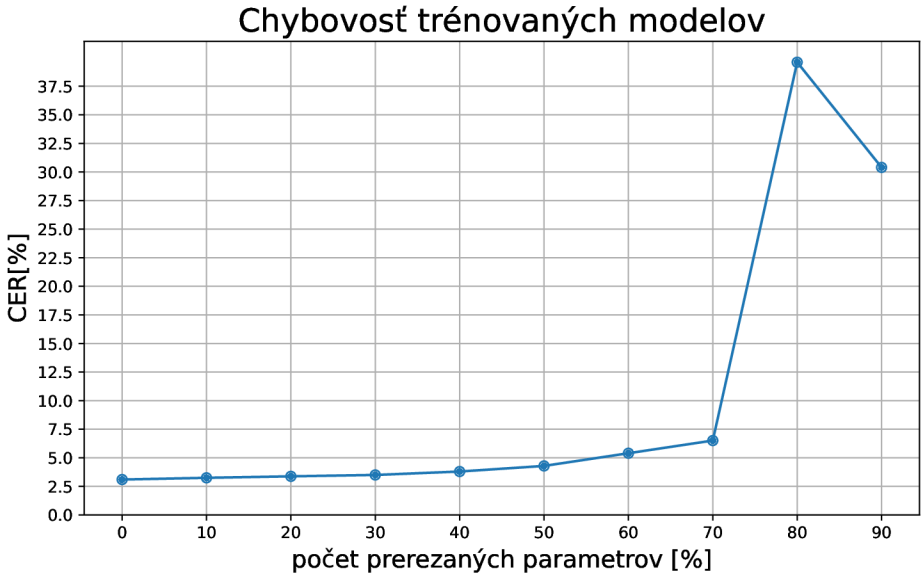
Pri konvolučnom prerezávaní sme do-trénovali všetky modely 40000 iteráciami. Pri learning-rate sme začínali na hodnote 0.0005, ktorú sme pri stagnácií stray (loss) siete



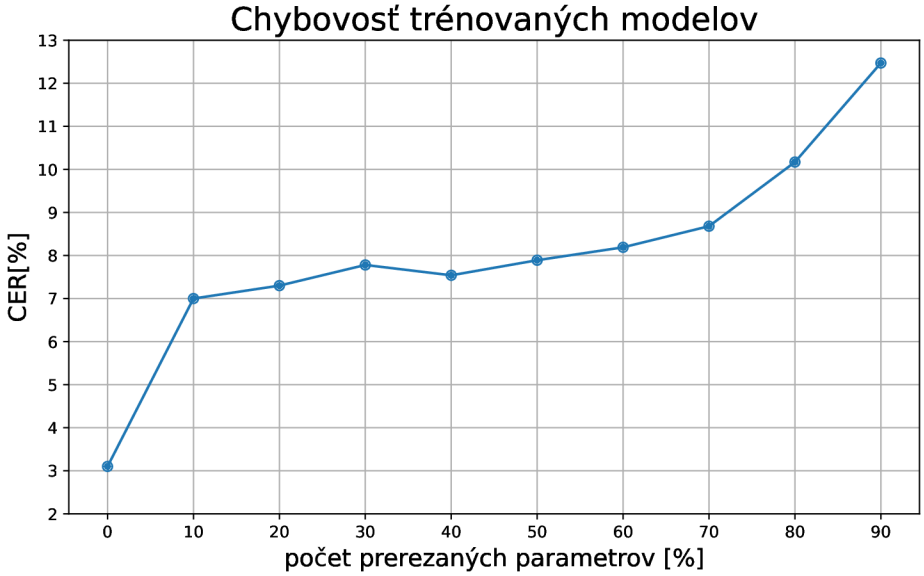
Obr. 6.5: Závislosť medzi počtom prerezaných parametrov rekurentnej časti a rýchlosťou siete

postupne znižovali niekedy až na hodnotu 0.000001. Až pri 50% prerezaných parametrov je chybovosť siete nad 4% CER a to konkrétne 4.29% ako môžme vidieť na Obrázku 6.8. Od tejto hranice chybovosť siete narastala a sieť sa stávala nepresnou. Modely použité v experimentoch sa nedali už viac do-trénovať ani za použitia viacerých iterácií čo je možné vidieť na Obrázku 6.8.

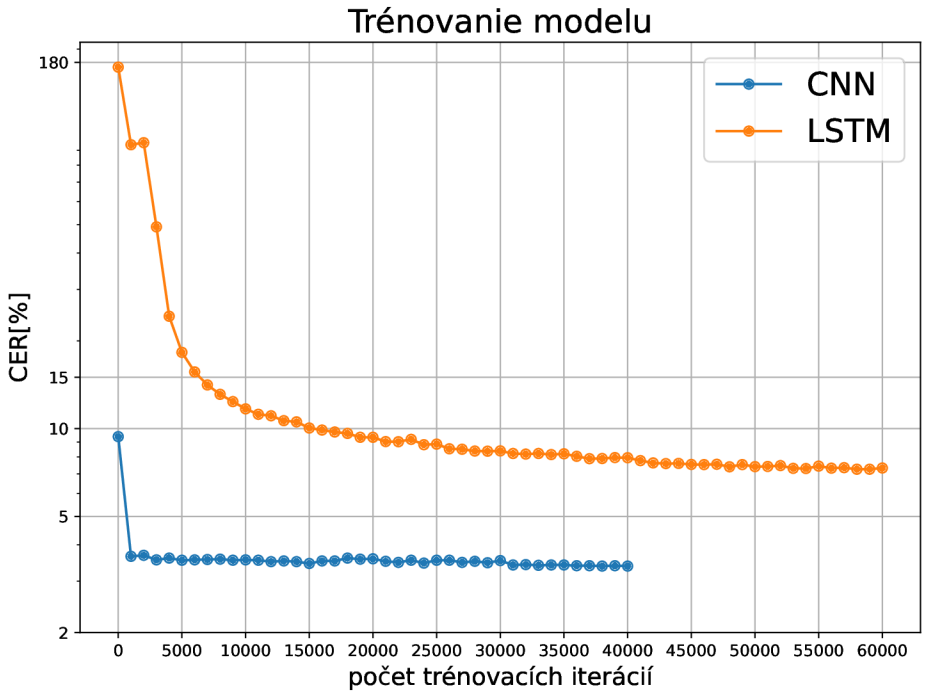
Rekurentná časť sa trénovala 60000 iterácii. Aj po 60000 jednotlivé modely neboli plne natrénované a stále sa mierne zlepšovala chybovosť siete. Pri trénovnáni bol použitý learning-rate = 0.0001 prvých 40000 iterácia a posledných 20000 iterácií bolo trénovaných na learning-rate = 0.00005. Modely sa nám nepodarilo natrénovať tak aby bola chybovosť pod 7% CER viď Obrázok 6.7. Tabuľku všetkých výsledkov môžme vidieť v prílohe A.



Obr. 6.6: Chybovosť do-trénovaných modelov u ktorých bola prerezávaná iba konvolučná časť.



Obr. 6.7: Chybovosť do-trénovaných modelov u ktorých bola prerezávaná iba rekurentná časť.



Obr. 6.8: Znázornenie trénovania dvoch modelov s prerezaním 20% parametrov.

Kapitola 7

Záver

Cieľom práce bolo vytvoriť aplikáciu na prerezávanie neurónovej siete určenej na prepis ručne písaného textu. Neurónová sieť je založená na architektúre konvolučne rekurentnej neurónovej siete.

Prerezávanie bolo spravené na dve časti a to na prerezávanie rekurentnej časti a prerezávanie konvolučnej časti poskytnutej neurónovej siete. Prerezávanie konvolučnej časti dosahovalo veľmi dobre úspechy pri prerezaní 30% siete sme stratili výslednú presnosť siete len o 0.4% CER a zrýchlenie o 12%. Pri rekurentnej časti sme dosahovali horšie výsledky. Najmenšia chybovosť bola po do-trénovaní bola 7% CER, ale aj pri prerezaní 80% parametrov sme vedeli model do-trénovať na 10.17% CER čo pri rekurentnej časti pri prerezaní rovnakého percenta parametrov predstavovalo chybovosť 40% CER.

Čo sa týka pomeru rýchlosti ku prerezaným parametrom tam pri nižšom počte prerezaných parametrov dosahovala rekurentá časť lepšie výsledky. Ale od 30% prerezaných parametrov začala byť rýchlejšia rekurentná časť. Pri rekurentnej časti sme mohli prerezávať každú vrstvu samostatne čo malo za následok optimálnejší model. Optimálny model bol rýchlejší od pôvodného o takmer 20% a chybovosť mal horšiu len on 0.4 % CER.

V budúcej práci by chcel zdokonaliť metódu na prerezávanie rekurentnej časti a skúsiť viac optimalizovať trénovacie parametre.

Literatúra

[1] Anwar, S., Hwang, K. a Sung, W. *Structured Pruning of Deep Convolutional Neural Networks*. arXiv, 2015. DOI: 10.48550/ARXIV.1512.08571. Dostupné z: https://arxiv.org/abs/1512.08571.

[2] Carter, S., Ha, D., Johnson, I. a Olah, C. Experiments in Handwriting with a Neural Network. *Distill*. 2016. DOI: 10.23915/distill.00004. Dostupné z: http://distill.pub/2016/handwriting.

[3] Cho, K., Merrienboer, B. van, Gülçehre, Ç., Bougares, F., Schwenk, H. et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*. 2014, abs/1406.1078. Dostupné z: http://arxiv.org/abs/1406.1078.

[4] Graves, A., Fernández, S., Gomez, F. a Schmidhuber, J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In:. New York, NY, USA: Association for Computing Machinery, 2006, s. 369–376. ICML '06. DOI: 10.1145/1143844.1143891. ISBN 1595933832. Dostupné z: https://doi.org/10.1145/1143844.1143891.

[5] Kang, L., Rusinol, M., Fornes, A., Riba, P. a Villegas, M. Unsupervised Writer Adaptation for Synthetic-to-Real Handwritten Word Recognition. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. March 2020.

[6] Kang, L., Toledo, J. I., Riba, P., Villegas, M., Fornés, A. et al. Convolve, Attend and Spell: An Attention-based Sequence-to-Sequence Model for Handwritten Word Recognition. In: Brox, T., Bruhn, A. a Fritz, M., ed. *Pattern Recognition*. Cham: Springer International Publishing, 2019, s. 459–472.

[7] Li, H., Kadav, A., Durdanovic, I., Samet, H. a Graf, H. P. Pruning Filters for Efficient ConvNets. *CoRR*. 2016, abs/1608.08710. Dostupné z: http://arxiv.org/abs/1608.08710.

[8] Molchanov, P., Tyree, S., Karras, T., Aila, T. a Kautz, J. *Pruning Convolutional Neural Networks for Resource Efficient Inference*. arXiv, 2016. DOI: 10.48550/ARXIV.1611.06440. Dostupné z: https://arxiv.org/abs/1611.06440.

[9] O'Shea, K. a Nash, R. *An Introduction to Convolutional Neural Networks*. 2015. Dostupné z: https://arxiv.org/abs/1511.08458.

[10] Semionov, N. *Pruning of Long Short-Term Memory Neural Networks: Passes of Redundant Data Patterns*. Netherlands, NE, 2019. Diplomová práce. TILBURG UNIVERSITY. Dostupné z: https://arno.uvt.nl/show.cgi?fid=153975/.

[11] Shi, B., Bai, X. a Yao, C. An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2017, zv. 39, č. 11, s. 2298–2304. DOI: 10.1109/TPAMI.2016.2646371.

[12] Simonyan, K. a Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: Bengio, Y. a LeCun, Y., ed. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. Dostupné z: http://arxiv.org/abs/1409.1556.

[13] Sutskever, I., Vinyals, O. a Le, Q. V. Sequence to Sequence Learning with Neural Networks. *CoRR*. 2014, abs/1409.3215. Dostupné z: http://arxiv.org/abs/1409.3215.

[14] Tanaka, H., Kunin, D., Yamins, D. L. a Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F. a Lin, H., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020, sv. 33, s. 6377–6389. Dostupné z: https://proceedings.neurips.cc/paper/2020/file/46a4378f835dc8040c8057beb6a2da52-Paper.pdf.

[15] Tang, S. a Han, J. *A pruning based method to learn both weights and connections for LSTM*. 2015. Diplomová práca. Stanford University. Dostupné z: https://nlp.stanford.edu/courses/cs224n/2015/reports/2.pdf.

[16] Tessier, H. *Neural Network Pruning 101* [online]. [cit. 2022-05-8]. Dostupné z: https://towardsdatascience.com/neural-network-pruning-101-af816aaea61.

[17] Wang, C.-F. *The Vanishing Gradient Problem* [online]. [cit. 2022-05-8]. Dostupné z: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[18] Wen, L., Zhang, X., Bai, H. a Xu, Z. Structured pruning of recurrent neural networks through neuron selection. *Neural Networks*. 2020, zv. 123, s. 134–141. DOI: https://doi.org/10.1016/j.neunet.2019.11.018. ISSN 0893-6080. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0893608019303776.

[19] Zhang, J., He, T., Sra, S. a Jadbabaie, A. *Why gradient clipping accelerates training: A theoretical justification for adaptivity*. arXiv, 2019. DOI: 10.48550/ARXIV.1905.11881. Dostupné z: https://arxiv.org/abs/1905.11881.

Príloha A

Výsledky prerezávaní

Percento prerezania[%]	CER [%] netrénovaná	trénovaná	Rýchlosť[s]	Zrýchlenie[%]	Počet naučitelných parametrov
Prerezávanie konvolučnej časti					
Baseline	**3.10**	-	**5.52**	**0.00**	**15 526 720**
10%	3.59	3.30	5.69	-3.00	14 944 308
20%	6.27	3.50	5.38	3.51	14 416 302
30%	16.53	3.70	4.82	12.58	13 941 117
40%	54.28	4.00	4.55	17.61	13 506 720
50%	78.88	4.29	3.81	30.82	13 127 296
60%	86.80	5.40	3.87	30.18	12 781 332
70%	85.27	6.51	3.59	34.91	12 487 470
80%	85.84	39.60	3.25	41.51	12 242 397
90%	88.97	34.40	3.12	43.40	12 042 144
*	31.30	3.60	4.44	19.50	14 091 628
Prerezávanie rekurentnej časti					
10%	173.00	7.00	5.32	3.46	13 184 328
20%	173.46	7.30	5.25	4.72	11 084 596
30%	169.80	7.78	5.11	7.23	9 249 372
40%	156.40	7.54	5.03	8.81	7 760 720
50%	152.50	7.89	4.87	11.32	6 188 352
60%	155.86	8.19	4.89	11.64	4 964 168
70%	155.24	8.68	4.77	13.52	4 249 712
80%	149.40	10.17	4.75	13.84	3 222 620
90%	141.40	12.47	4.65	15.72	2 723 152

Tabuľka A.1: Celkové výsledky. Pri konvolučnej časti sa uvádza najlepšia presnosť siete z troch možných ohodnotení. * je sieť pozostávajúca z následujúcich prerezanými vrstvami: [50%, 50%, 40%, 40%, 30%, 30%, 80%]