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

VYUŽITÍ HLUBOKÉHO UČENÍ PRO ROZPOZNÁNÍ TEXTU V OBRAZU GRAFICKÉHO UŽIVATELSKÉHO ROZHRANÍ

DEEP LEARNING FOR OCR IN GUI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. PAVEL HAMERNÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ LYSEK**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

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

Zadání diplomové práce

Student: **Hamerník Pavel, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Využití hlubokého učení pro rozpoznání textu v obrazu grafického uživatelského rozhraní**
Deep Learning for OCR in GUI
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte si základy hlubokého učení (deep learning) se zaměřením na konvoluční neuronové sítě.
2. Vytvořte si přehled o používaných technikách pro rozpoznání textu v obrazu a prostudujte služby/knihovny, které toto rozpoznání nabízejí.
3. Obstarejte nebo si vygenerujte vhodnou datovou sadu pro experimenty obsahující anotované obrazy grafických uživatelských rozhraní (může být i webové rozhraní).
4. Otestujte existující služby/knihovny na této datové sadě, zanalyzujte výsledky a navrhněte vlastní metodu, pomocí které se budete snažit tyto výsledky napodobit případně dosáhnout lepších výsledků.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou pomocí vlastní metody.
6. Porovnejte dosažené výsledky a diskutujte možnost dalšího vývoje.

Literatura:

- http://www.deeplearningbook.org/

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4.

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

Vedoucí práce: **Lysek Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Optické rozpoznání znaků (OCR) je již mnoho let oblastí zájmu. Je definován jako proces digitalizace obrazu dokumentu do sekvence znaků. Navzdory desetiletím intenzivních výzkumů jsou systémy OCR, které jsou srovnatelné s lidským zrakem, stále otevřenou výzvou. V této práci je vytvořen návrh takového systému, je implementován, který je schopen detekovat text v grafických uživatelských rozhraních.

Abstract

Optical character recognition (OCR) has been a topic of interest for many years. It is defined as the process of digitizing a document image into a sequence of characters. Despite decades of intense research, OCR systems with capabilities to that of human still remains an open challenge. In this work there is presented a design and implementation of such system, which is capable of detecting texts in graphical user interfaces.

Klíčová slova

rozpoznání textu, neuronové sítě, konvoluční neuronové sítě, CNN, LSTM, rekurentní neuronové sítě, RNN, hluboké učení neuronových sítí, OCR

Keywords

text recognition, neural network, convolutional neural network, CNN, LSTM, recurent neural network, RNN, deep learning, OCR

Citace

HAMERNÍK, Pavel. *Využití hlubokého učení pro rozpoznání textu v obrazu grafického uživatelského rozhraní*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Lysek

Využití hlubokého učení pro rozpoznání textu v obrazu grafického uživatelského rozhraní

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Ing. Tomáše Lyska. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Hamerník
22. května 2019

Poděkování

Chtěl bych poděkovat svému vedoucímu diplomové práce Ing. Tomáši Lyskovi, za odborné vedení, za pomoc a rady při zpracování této práce. Dále bych chtěl poděkovat rodině a svým nejbližším přátelům za jejich podporu, motivaci a korekturu při psaní práce.

Obsah

Kapitola 1

Úvod

Optické rozpoznání znaku (Optical Character Recognition dále jen OCR) je software který dokáže prevést tištený text a obrázky do digitalizované podoby se kterou muže být manipulováno pocítacem. Na rozdíl od lidského mozku, který dokáže velmi jednoduše rozpoznat text z obrázku, algoritmy dostatecne inteligentní, aby dokázaly vnímat informaci dostupnou z obrazu. Proto bylo vloženo velké úsilí na vytvorit software, který se snaží transformovat obraz dokumentu do podoby, která je srozumitelná pro pocítac. OCR je velmi rozsáhlý problém kvuli velkému množství jazyku, stylu a písem, kterými mužeme vyjádrit text. Techniky z ruzných oboru výpocetní techniky se využívají pro rešení ruzných výzev tohoto systému.

V této práci je prezentován návrh systému OCR, pro detekci textu v grafických uživatelských rozhraních. Toho bude využito v rámci technologií RPA (Robotic process Automation) pro ovládání testu prípadne obecnou automatizaci pouze na základe obrazových vlastností. V kapitole 2 si predstavíme technologie neuronových sítí. V kapitole 3 jsou uvedeny technologie zpracování textu, které zahrnují základní princip algoritmu rozpoznání textu a metriky pro porovnání kvality výsledku techto algoritmu. Dále v kapitole 4 je popsán nástroj „gui-generator“ a jeho rozšírení, které bylo využito pro vygenerování datové sady urcené k trénování i vyhodnocení algoritmu. Kapitola 5 obsahuje návrh neuronového modelu, pomocí nehož jsme schopni rozpoznat texty v grafických uživatelských rozhraních a v kapitole 6 je popis jeho implementace. Výsledky a zhodnocení algoritmu se nachází v kapitole 7.

Kapitola 2

Neuronové síte

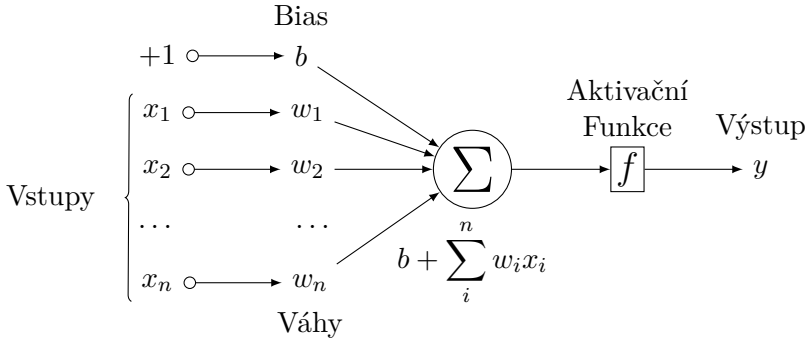
2.1 Architektury sítí

Neuronová sít je výpocetní model používaný v umelé inteligenci. Vzorem neuronových sítí je biologický model lidského mozku. Neuronové síte se používají pro radu úkolu jako jsou detekce a rozpoznání, regrese velicin, aj. S príchodem hlubokého ucení neuronových sítí nabyly na velké popularite rešit s pomocí nich složitejší úkoly a pracovat s naucenými sítemi jako s tzv. „cernou skrínkou“ (anglicky black box). To znamená, že ocekáváme od sítí nejakou funkcnost, ale jak jí dosáhne, nezjistíme.

V následujících podkapitolách si ukážeme, jak pracuje klasická neuronová sít (kap. 2.1.1). Další kapitolou (2.1.2) bude konvolucní neuronová sít urcená pro zpracování vetšího objemu dat napríklad obrázku. V kapitole 2.3 si ukážeme aktivacní funkce neuronových sítí.

2.1.1 Klasické neuronové síte

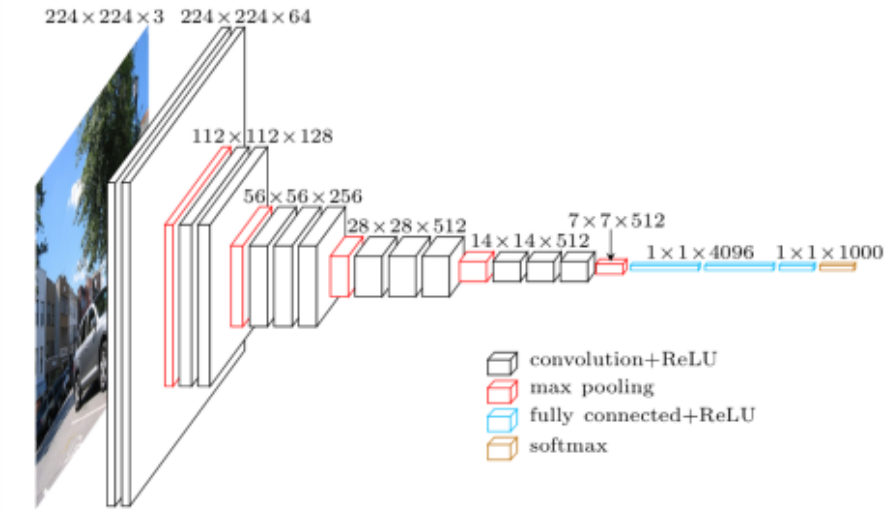
Základním stavebním kamenem softwarových neuronových sítí je neuron (dríve nazývaný perceptron). Každý neuron se skládá z trí hlavních cástí. Vstupní cást obsahuje všechny vstupní signály, které putují do neuronu. Každý z techto vstupu je navíc ohodnocen váhou, která slouží k posílení, nebo potlacení síly daného vstupu. Další cástí je telo neuronu, které shrne všechny hodnoty vstupu vcetne jejich vah do jediné hodnoty. Na tuto hodnotu se aplikuje aktivacní funkce neuronu a jejím výsledkem je i výsledek celého neuronu [6]. Výpocetní model neuronu si mužete prohlédnout na obrázku 2.1.



Obrázek 2.1: Výpocetní model neuronu.

2.1.2 Konvolucní neuronové síte

Konvolucní síte jsou speciálním druhem vícevrstvých neuronových sítí. Používají vizuálních vzoru prímo z pixelu obrazu. Vychází z principu neocognitronové síte, která byla poprvé predstavena v 80. letech K. Fukushimom. Neocognitron využívá velké množství neuronových vrstev a obsahuje variabilní propojení mezi bunkami sousedních vrstev. Cílem této síte je identifikace objektu podle jejich obecné podobnosti. Typickou konvolucní sít mužeme pozorovat na obrázku 2.2 a je možné pozorovat, že se skládá z mnoha vrstev. Dalším charakteristickým znakem konvolucních sítí je získávání vlastností obrazu z recepcních polí, sdílení vah v rámci konvolucní vrstvy a prostorové vzorkování. [6]

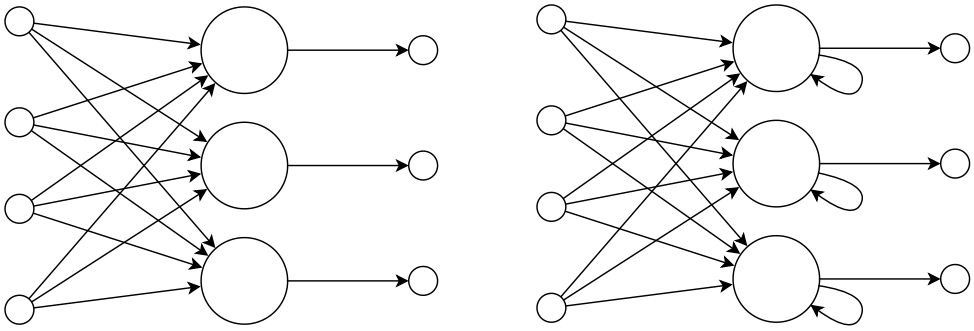


Obrázek 2.2: Ilustrace konvolucní neuronové síte. Zdroj `https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/`

2.1.3 Rekurentní neuronové síte

Rekurentní neuronové síte (dále jen RNN) pochází ze stejné doby jako mnoho dalších algoritmu hlubokého ucení. Poprvé byly predstaveny v 80. letech, ale jejich plného potenciálu nebylo donedávna možné využít zejména kvuli vysoké výpocetní nárocnosti. Výrazem rekurentní neuronové síte oznacujeme dve široké kategorie s podobnou obecnou strukturou. Jedna z nich je sít s konecnou odezvou, tento typ síte se dá vyjádrit pomocí orientovaného acyklického grafu, který je možné rozbalit a díky tomu je celou sít možné nahradit doprednou sítí. Rekurentní sít s nekonecnou odezvou lze reprezentovat orientovaným cyklickým grafem, který není možné redukovat na doprednou sít. [11]

RNN využívají interní pamet, která jim umožnuje si zapamatovat informace z prijatých vstupu. Díky tomu dokážou být velmi presné v predikci následujících hodnot. Z tohoto duvodu jsou preferovanou sítí pri práci se sekvencními daty (napríklad casová posloupnost zvukových dat), protože ze zpracovávaných dat dokáží porozumet hlubšímu významu jejich posloupnosti oproti ostatním algoritmum.



(a) (b)

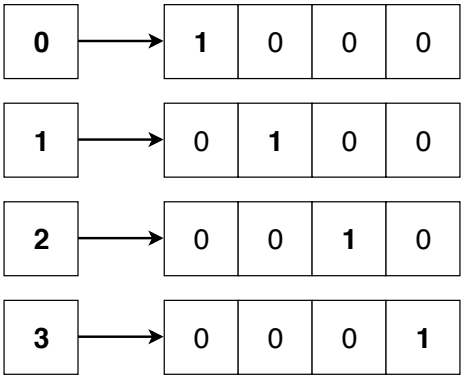
Obrázek 2.3: Struktura dopredné neuronové vrstvy (a) oproti rekurentní neuronové vrstve (b).

2.2 Kódování

V neuronových sítích je casto potreba informace zakódovat do reprezentace, která je pro neuronovou sít užitecnejší. Díky nim se dokáží lépe ucit, prípadne reprezentovat data stylem, který zjednoduší neuronové síti práci. Nejpoužívanejší z nich jsou dále predstaveny.

2.2.1 One-hot kódování

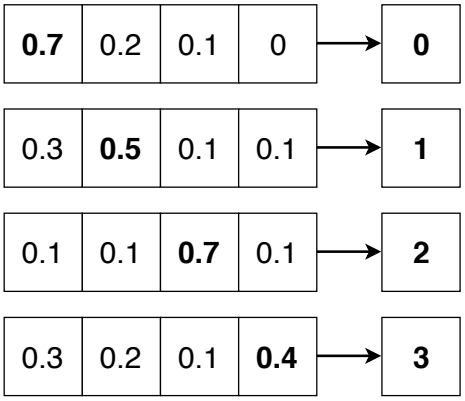
One-hot[1] kódování (v anglictine se také nazývá dummy variable) vyjadruje pomocí hodnoty 0 nebo 1 absenci nebo prítomnost jisté kategorie. Toto kódování je užitecné zejména pro rozdelení dat do vzájemne se vylucujících kategorií (napríklad. pes, kocka). Na hodnoty zakódovaných kategorií se mužeme také dívat jako na míru pravdepodobnost výskytu jednotlivých kategorií v datech. To je duvodem, proc je toto kódování hojne používáno komunitou zabývající se strojovým ucením, a to zejména v oblasti klasifikace objektu. Príklad zakódování informace do vektoru míry pravdepodobnosti mužeme pozorovat v obrázku 2.4.



Obrázek 2.4: Ukázka one-hot kódování pro 4 trídy.

Dekódování hodnoty z one-hot reprezentace provedeme získáním kategorie s nejvyšší mírou pravdepodobnosti. Príklad výstupu z neuronové síte a jeho dekódování mužeme pozorovat na obrázku 2.5.

[1] V ceské literature se také mužeme setkat s názvem kódování 1 z N.

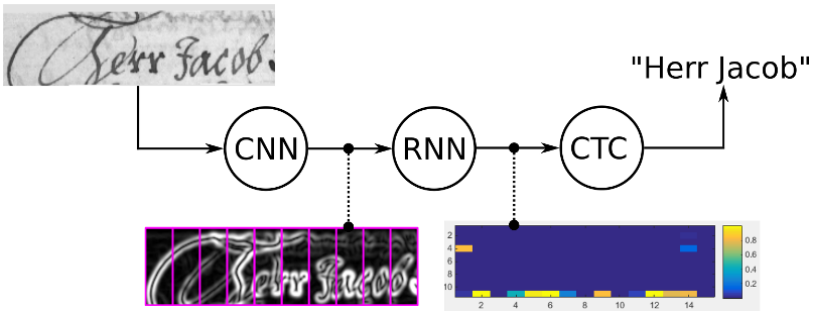


Obrázek 2.5: Ukázka dekódování one-hot výstupu pro 4 tridy.

2.2.2 Connectionist temporal classification

Je velmi užitecnou operací používanou v rekurentních neuronových sítích [7]. Jedná se o zpusob, jak naucit neuronovou sít požadovanou sekvenci, která však muže mít mnohem menší pocet casových kroku. Tato operace je casto používána v rámci algoritmu pro rozpoznání textu, ci hlasu, kde umožnuje síti se naucit sekvenci vzoru, které jsou mnohem menší než sekvence, které na tyto vzory chceme naucit. Obsahuje-li sekvence více po sobe jdoucích casových kroku stejnou hodnotu, pak se jedná o jediný znak s touto hodnotou. CTC navíc pridává do slovníku speciální znak, nazývaný blank. Ten slouží jako oddelovac mezi znaky a umožnuje vytvorení sekvence ze stejného znaku, aniž by byl považován za duplikátní pri použití blank znaku mezi temito elementy. Níže jsou uvedeny príklady, jak je možné pomocí CTC reprezentovat nekterá slova:

- `"auto"` → `"---aaa-u-tttttt-o"`, `"-a-u-t-o-"`, nebo `"auto"`
- `"book"` → `"boooooo----ok"`, `"-b-o-o-k-"`, nebo `"bo-ok"`.



Obrázek 2.6: Kombinace sítí s CTC. Zdroj: `https://towardsdatascience.com`

2.3 Aktivacní funkce

Aktivacní funkce je hlavní složkou každé neuronové síte. Je aplikována na agregované hodnoty výstupu neuronu a slouží jako hlavní výstup neuronu, oznacováno jako aktivace neuronu. Tato funkce je lineární i nelineární a také diferencovatelná, tudíž je možné nalézt

smer rustu této funkce (gradient funkce). Gradientu je potreba pro ucení neuronové síte. Hodnota aktivace neuronu je rízena následující funkcí:

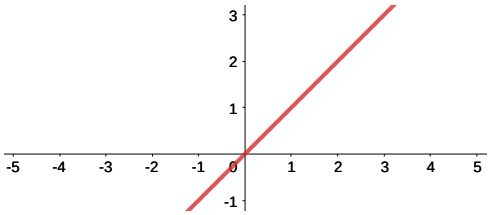
$$y = f\left(\sum_i w_i x_i + b\right) \tag{2.1}$$

, kde parametr w_i reprezentuje váhu jednotlivých vstupu x_i a parametr b (anglicky bias) se využívá k rozšírení rozhodovacího rozsahu, kterého neuronová sít muže dosáhnout. Pro snadnejší pocítání s tímto parametrem (biasem), se rozširuje vektor vstupu o konstantní 1 (bývá oznacováno jako x_0), a vektor vah o parametr b na odpovídající pozici dle pridaného vstupu (w_0). Díky této úprave mužeme rovnici výstupu neuronu prepsat do zjednodušené podoby za pomocí skalárního soucinu:

$$y = f\left(\vec{w} \cdot \vec{x}\right) \tag{2.2}$$

Standardne není-li uvedeno má neuron lineární aktivacní funkci identita. Tato však funkce nijak nepomáhá se složitostí parametru, které jsou posílány neuronovým sítím. Proto je použití nelineárních funkcí vhodnejší. Lineární funkce má však své využití, protože nijak neomezuje výstup tak lze použít pro regresní úlohy nebo je vhodná, pokud chceme použít speciální aktivacní neuronovou vrstvu, která aktivuje neurony na základe všech hodnot neuronu ve vrstve (napríklad Softmax vrstva, více v 2.4.5).

$$f(x) = x \tag{2.3}$$

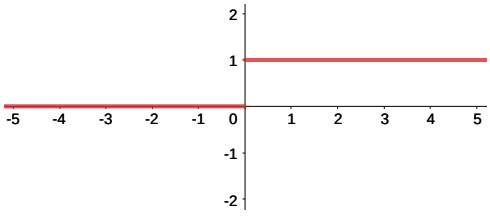


Obrázek 2.7: Aktivacní funkce identita

Pro klasifikacní úlohy jsou odnedávna využívány skoková funkce, sigmoida a hyperbolický tangent. Každá z techto funkcí prevádí celé vstupní spektrum reálných hodnot do úzkého intervalu hodnot.

Skoková funkce je casto spojována s pojmem perceptron. Nabývá pouze dvema hodnotami, pokud hodnota vstupu prekrocí urcitý práh. Je vhodná pouze pro jednoduché úlohy. Není diferencovatelná, proto není vhodná pro použití ve vícevrstvých neuronových sítích.

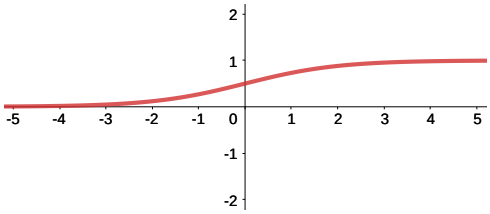
$$f(x) = \begin{cases} 1, & \text{pokud} x > 0 \\ 0, & \text{jinak} \end{cases} \tag{2.4}$$



Obrázek 2.8: Skoková aktivacní funkce

Sigmoidová funkce (také logistická funkce) prevádí vstupní reálné hodnoty do intervalu $(0, 1)$. Je monotónní a diferencovatelná je možné ji využít ve vícevrstvých neuronových sítích. Casto se používá pro binární klasifikaci, kdy hodnota urcuje míru pravdepodobnosti zkoumaného kritéria. Její nevýhodou je, že se ucení neuronové síte muže zaseknout.

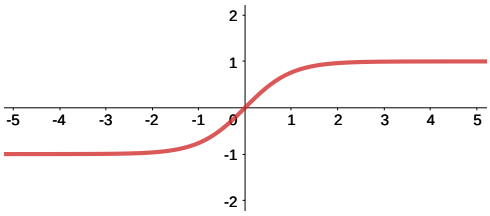
$$f(x) = \frac{1}{1 + e^{-x}} \qquad (2.5)$$



Obrázek 2.9: Sigmoidová aktivacní funkce.

Hyperbolický tangent je podobný jako sigmoidová funkce. Výhodou této funkce oproti sigmoidové je vetší rozsah výstupních hodnot, protože mohou nabývat i záporných hodnot. Funkce má rozsah hodnot v intervalu $(-1, 1)$. Využívá se zejména pro binární klasifikaci.

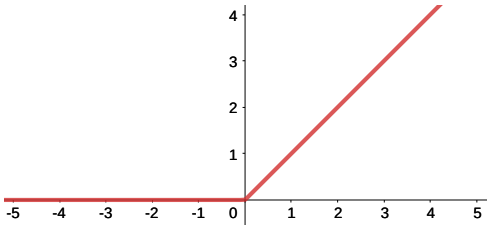
$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \qquad (2.6)$$



Obrázek 2.10: Aktivacní funkce hyperbolický tangent.

Rectified linear unit (dále jen ReLU) je v soucasné dobe nejpoužívanejší aktivacní funkce. Tato funkce se stala velmi oblíbenou díky svým vlastnostem a snadné implementaci prahováním v 0. Funkce je monotónní stejne je i její derivace. Díky tomu se u neuronových sítí zvyšuje rychlost ucení.

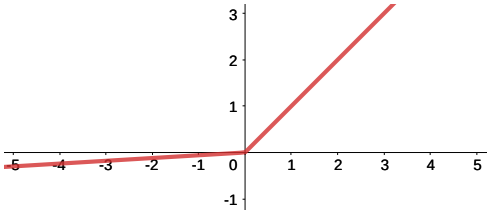
$$f(x) = \begin{cases} x, & \text{pokud } x \geq 0 \\ 0, & \text{jinak} \end{cases} \qquad (2.7)$$



Obrázek 2.11: Aktivacní funkce ReLU.

Problémem ReLU je že všechny záporné hodnoty jsou mapovány do nuly. Nulové hodnoty jsou v ucení potlacovány, protože mají derivaci v nule. Toto má za následek snížení schopností neuronové síte naucit se daný problém efektivne. Pokusem vyrešit tento problém je Leaky ReLU. Tato funkce mapuje záporné hodnoty pomocí lineární funkce s velmi malým sklonem. Díky tomu nejsou záporné hodnoty potlaceny a mají schopnost se ucit.

$$f(x) = \begin{cases} x, & \text{pokud } x \geq 0 \\ 0.01x, & \text{jinak} \end{cases} \qquad (2.8)$$



Obrázek 2.12: Aktivacní funkce Leaky ReLU.

2.4 Vrstvy neuronových sítí

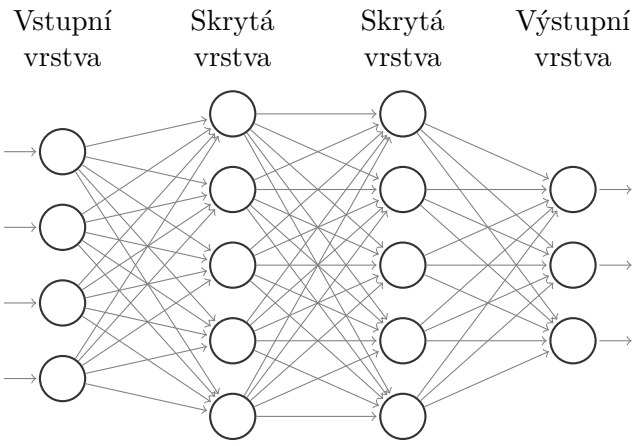
V této sekci jsou uvedeny vrstvy neuronových sítí, které jsou použity v rámci této práce.

2.4.1 Plne propojená vrstva

Plne propojená vrstva, v anglické literature se mužeme setkat s názvem fully-connected ci dense layer, jak název vypovídá je vrstva N neuronu a každý z techto neuronu je propojený se všemi neurony vstupní vrstvy. Mužeme ji definovat transformacní funkcí

$$y_j = \sum_{i}^{M} w_{ij} x_i, \tag{2.9}$$

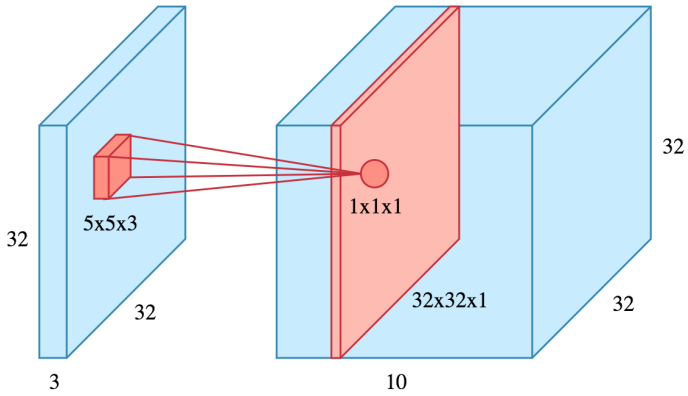
kde y_j udává hodnotu j-tého výstupního neuronu ($1 \leq j \leq N$). x_i jsou vstupní hodnoty neuronu z predchozí vrstvy. Tuto vrstvu je možné pozorovat na obrázku 2.13.



Obrázek 2.13: Model síte s plne propojenými vrstvami.

2.4.2 Konvolucní vrstva

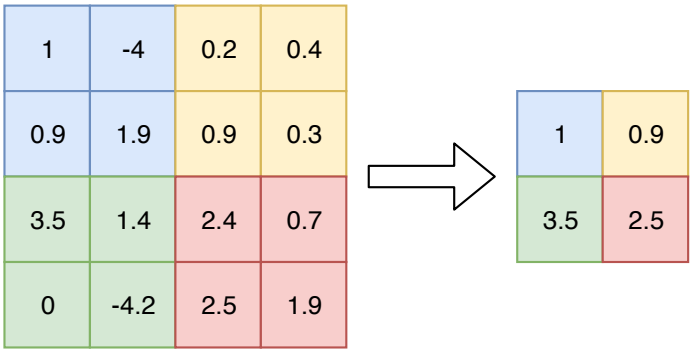
Tato vrstva je využívá konvolucní filtry, jejichž parametry jsou pro celou vrstvu sdílené. Tyto filtry pokrývají pouze malou cást plochy vstupních dat, ale využívají celou jejich hloubku. Na hodnotu získanou konvolucí je aplikována aktivacní funkce, která je výstupem tohoto filtru pro jeden neuron. Výstupem jednotlivých konvolucních filtru této vrstvy je dvoudimenzionální mapa, dohromady všechny mapy tvorí objem, kde hloubkou tvorí pocet aplikovaných konvolucních filtru dané vrstvy. U techto vrstev urcujeme nekolik vlastností, kterými jsou pocet aplikovaných konvolucních filtru, velikost jádra konvoluce techto filtru (anglicky kernel size), krok posunu konvoluce (anglicky stride) - urcuje rychlost posunu klouzavého okna konvolucních filtru po vstupu. Dále mužeme u vrstvy specifikovat tzv. zero-padding. V tomto režimu jsou vstupní data doplnena o nulové hodnoty tak, aby velikost výstupu po provedení konvoluce byla stejná jako vstupní velikost (velikost pred doplnením o nuly). Konvolucní vrstva je znázornena na obrázku 2.14.



Obrázek 2.14: Konvolucní vrstva. Zdroj `https://towardsdatascience.com`

2.4.3 Pooling vrstva

Slouží pro zmenšení velikosti dat vstupní vrstvy. Tato vrstva se hlavne využívá mezi konvolucními vrstvami pro snížení celkového poctu parametru výsledné síte. Podvzorkování se aplikuje na jednotlivých hloubkách vstupních dat samostatne. V rámci tohoto vzorku dat pro hloubku se podobne jako u konvoluce posouvá klouzavé okno, na které se aplikuje nekterá z agregacních funkcí (nejcasteji se jedná o funkci maximum nebo prumer, podle nich pak nazývány max pooling nebo average pooling). Podobne jako u konvolucní vrstvy má i tato vrstva krok (anglicky stride), kterým se posouvá okénko podvzorkování a také velikost plovoucího okénka.



Obrázek 2.15: Ukázka operace max-pooling.

2.4.4 Normalizacní vrstva

Normalizacní vrstva, v anglické literature uvádena pod názvem batch normalization layer, je vrstva, která provádí korekci vstupních dat. Data jsou transformována do normálního rozložení se strední hodnotou 0 a rozptylem 1. Díky tomu je zajišteno, že se rozložení hodnot dat bude pohybovat v nejbližším okolí 0. Tato vrstva si pri trénování pocítá statistiky o stredních hodnotách a rozptylech, které jsou posléze využity pri inferenci síte. V praxi se tato vrstva používá mezi vrstvou konvoluce, prípadne plne propojenou vrstvou, a vrstvou s aktivacní funkcí. Pro každý mini-batch se normalizace spocítá následujícími rovnicemi:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{2.10}$$

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2, \tag{2.11}$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \tag{2.12}$$

$$y_i = \gamma \hat{x}_i + \beta, \tag{2.13}$$

kde μ znací strední hodnotu v rámci mini-batche, σ^2 je rozptyl, x_i jsou vstupy vrstvy, y_i jsou výstupy vrstvy a γ, β jsou parametry, které se tato vrstva ucí.

Použití této vrstvy snižuje úcinek problému zvaného internal covariance shift. Tento problém nastává pri posunu vstupní distribuce ucícího se systému. V prípade hlubokých sítí je vstup v každé vrstve ovlivnen parametry této vrstvy. I ty nejmenší zmeny parametru síte se pruchodem sítí zesílí a v konecném dusledku zpusobí posuny vstupních distribucí vnitrních vrstev síte, což muže vést k snížení konvergence síte.

2.4.5 Softmax vrstva

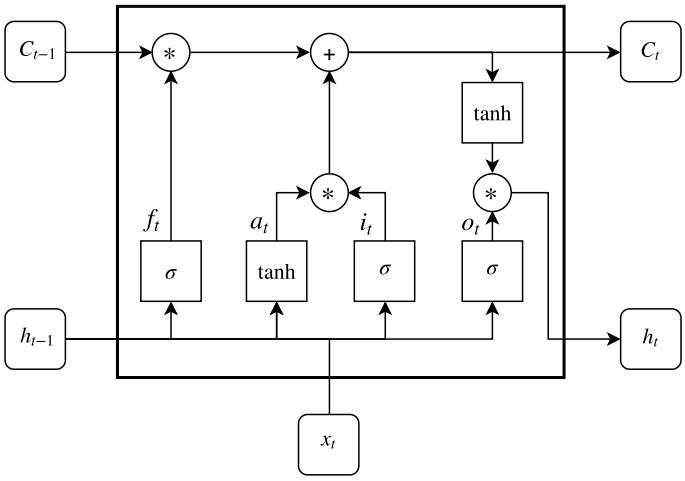
Tato vrstva slouží transformaci vstupu vrstvy do výsledného rozložení pravdepodobnosti, kde soucet všech výsledku této vrstvy je roven 1. Využívá se pro klasifikaci mnoha výlucných tríd, kdy pouze jediná z hodnot má nejvyšší pravdepodobnost. Výstup této vrstvy obsahuje stejný pocet hodnot jako jeho vstup a je možné jej vypocítat rovnicí:

$$y_j = \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}, \tag{2.14}$$

kde j je hodnota v rozsahu $1 \leq j \leq N$ urcuje index výstupu, x_i je vstup vrstvy na indexu i a y_i je výstup vrstvy na indexu i.

2.4.6 Long Short-Term Memory vrstva

Long Short-Term Memory, dále jenom LSTM, je typ rekurentní neuronové síte, která má architekturu skládající se z retezených bunek. Bunku LSTM mužeme pozorovat na obrázku 2.16. LSTM bunka se skládá se z nekolika hradel, které napomáhají této vrstve udržovat si vnitrní pamet. Funkci LSTM bunky mužeme rozdelit do trech cástí:



Obrázek 2.16: Schéma bunky LSTM vrstvy

Zapomenutí. Využívá hradlo *forget gate*, které umožnuje LSTM se rozhodnout jaké vlastnosti musí být zapomenuty (odtud název tohoto hradla) na základe predchozího stavu a soucasného vstupu, aby byly zachovány pouze duležité informace. Využívá funkci sigmoida, které vrátí hodnotu v intervalu $(0; 1)$. Pricemž hodnota 0 zahodí veškeré informace ze stavu a naopak 1 je všechny zachová.

$$f_t = \sigma\left(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + b_f\right) \tag{2.15}$$

Zapamatování. Umožnuje LSTM zapamatovat si nové vlastnosti ze soucasného vstupu. Využívá k tomu dvou hradel. Aktivacní hradlo [2] získá vektor nových vlastností, které chceme pridat k soucasnému stavu. Dále hradlo input gate rozhoduje, které hodnoty chceme aktualizovat a s jakou mírou (podobne jako forget gate výše).

$$a_t = \tanh\left(W_{xa} \cdot x_t + W_{ha} \cdot h_{t-1} + b_a\right) \tag{2.16}$$

$$i_t = \sigma\left(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + b_i\right) \tag{2.17}$$

Obe tyto hradla jsou posléze zkombinovány a pridány k novému stavu. Vektory se kombinují pomocí Hadamardova soucinu, který násobí vektory mezi sebou po jejich jednotlivých složkách a v rovnicích je oznacen symbolem $*$. Kombinace techto vstupu a hodnoty forget gate dohromady udává hodnotu následujícího stavu, jak mužeme videt níže.

$$C_t = a_t * i_t + C_{t-1} * f_t \tag{2.18}$$

Výstup. Výstup LSTM je založený na soucasném stavu v bunce. Nejprve pomocí output gate jsou získány hodnoty, které urcí cásti soucasného stavu, jež jsou ve výstupu zachovány.

$$o_t = \sigma\left(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o\right) \tag{2.19}$$

Dále je ze soucasného stavu získán hyperbolický tangent, který hodnoty srazí do intervalu $(-1; 1)$. Takto získané hodnoty stavu jsou posléze zkombinovány s výstupem z output gate a udávají hodnotu výstupu LSTM v daném case.

$$h_t = \tanh\left(C_t\right) * o_t \tag{2.20}$$

[2] V literature bývá oznacováno jako gate.

Ve všech predchozích rovnicích W_* znací váhy vrstvy pro jednotlivé části a vstupy, x_t je vstup v case t, h_t je predchozí výstup LSTM v case t, C_t je zapamatovaný stav LSTM v case t.

Pri zpetné propagaci je gradient neprerušene propagován skrze stavy jednotlivých bunek. Díky takto navržené strukture bunek netrpí LSTM problémy vanishing ci exploding gradient jako obecné RNN. To je problém, který nastává pri retezení funkcí, jejichž výstup je omezený v úzkém intervalu (napríklad sigmoida). To má za následek, že se postupne hodnoty tlumí až do bodu, kdy hodnota propagované zmeny je tak nepatrná, že je její vliv zanedbatelný a ucení síte uvázne. U rekurentních sítí je tento problém ješte více znatelný, protože pridávají ješte více vrstev.

2.5 Ucení sítí

Hlavním úcelem ucení neuronové síte je nastavení hodnot váhových parametru a prahu mezi perceptrony tak, aby sít náležite reagovala na vstup. Mezi základní strategie ucení patrí ucení s ucitelem a ucení bez ucitele.

Ucení s ucitelem využívá znalosti vstupu i správného výstupu. Nejprve pro vstup sít vygeneruje výstup, který porovná se správným výstupem, a poté upraví váhové parametry tak, aby bylo dosaženo co nejlepších výsledku.

Druhý zpusob je ucení bez ucitele. Zde jsou správné výstupy neznámé, a tak se ze zadaných vstupních dat snaží ucení získat spolecné zákonitosti a nastavit váhy i prahy tak, aby na podobné vstupy reagovala podobnými výstupy.

2.5.1 Back propagation

Je algoritmem ucení neuronové síte s ucitelem, který slouží k adaptaci neuronové síte na danou trénovací množinu. Rozdeluje se do trí etap: dopredného šírení signálu ze vstupních dat, zpetného šírení chyby a úprav váhových parametru mezi perceptrony. Na zacátku se náhodne navolí parametry síte. Pro vstupní data se porovnají výsledky výstupu s temi správnými, címž se dopocítá chyba neuronové síte. Úpravou váhových parametru napríklad podle metody gradient descent lze dosáhnout presnejších výsledku.

Chyba síte pro celý pruchod testovacími vstupy se spocítá jako

$$E = \frac{1}{2} \sum_{i=1}^{p} ||y_i - t_i||^2, \tag{2.21}$$

kde y_i je výstup pro testovací data i a t_i je ocekávaný korektní výsledek.

Gradient descent

je nejjednodušší metodou k získávání nejnižší odchylky, kdy se chyba zmenšuje podle nejstrmejšího klesání (nejvíce záporného gradientu), který vypocítáme pomocí

$$\nabla E = \left(\frac{\partial E}{w_1}, \frac{\partial E}{w_2}, \frac{\partial E}{w_3}, ..., \frac{\partial E}{w_l} \right). \tag{2.22}$$

Metoda gradient descent je omezena na hledání lokálních minim, címž muže dojít k minimalizaci chyby, ale nemusí získat nejlepší rešení. Po pruchodu celé testovací množiny

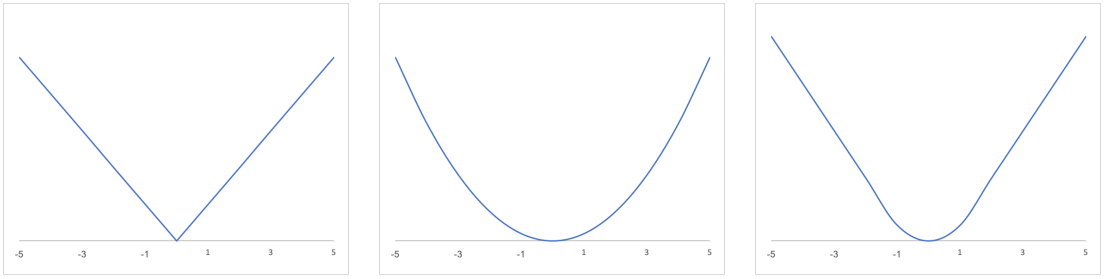
dojde k prenastavení váhových parametru a iterativne dochází ke snižování chyby, kdy se snaží dosáhnout nulového gradientu ($\nabla E = 0$).

$$\begin{aligned} w_{i+1} &= w_i + \Delta w_i \\ \Delta w_i &= -\eta \frac{\partial E}{\partial w_i}, \end{aligned} \tag{2.23}$$

kde η reprezentuje rust ucení, který definuje velikost kroku jednotlivé derivace.

2.5.2 Loss funkce

Loss funkce neboli chybová funkce vyjadruje chyby pomocí matematického zápisu. Cílem chybové funkce je minimalizovat chybu nastavením vah a prahu pri použití optimalizacních algoritmu. Následne jsou popsány významnejší loss funkce. [6]



(a) L1 (b) L2 (c) Smooth - L1

Obrázek 2.17: Orientacní grafy vybraných loss funkcí

L1 loss

Neboli Least absolute deviation je souctem všech rozdílu mezi správným výstup a výstupem získaným z neuronové síte.

$$lad\_loss = \sum_{i=1}^{n} |y_i - t_i|. \tag{2.24}$$

Hlavní vlastností této funkce je robustnost, díky která dokáže pracovat nad datasety, které obsahují okrajové hodnoty. Nevýhodou této funkce je, že gradient se behem pruchodu celé skupiny testovacích dat nemení. Tím dochází k tomu, že gradient bude velký i pro malé chybové hodnoty.

L2 loss

Funkce taktéž pojmenovaná least square errors (lse), u která již z názvu napovídá, že využívá na rozdíl od L1 druhou mocninu absolutní hodnoty rozdílu získaného a správného výstupu:

$$lse\_loss = \sum_{i=1}^{n} |y_i - t_i|^2. \tag{2.25}$$

Tato funkce si neumí dobre poradit s trénovacími daty, pokud obsahují vetší množství okrajových hodnot, kdy pri umocnení dojde k velkému navýšení chyby.

Smooth L1

Neboli Huber loss funkce je kombinací L1 a L2, která není na okrajové hodnoty trénovacích dat tak náchylná jako L2.

$$smooth\_l1(a) = \begin{cases} \frac{1}{2}a^2 & \text{pro } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta) & \text{pro ostatní} \end{cases} \tag{2.26}$$

Funkce využívá pro výpocet malých a kvadratickou funkci a pro vetší hodnoty a využívá lineární funkci, címž zamezí rostoucímu stoupání v prípade okrajových hodnot.

Cross-Entropy Loss

Cross-entropy loss, také nazývaný log loss, merí výkonost klasifikacního modelu, jehož výstup obsahuje hodnoty pravdepodobností v intervalu od 0 do 1. Hodnota Cross-entropy lossu se zvyšuje, práve když se predikovaná pravdepodobnost vzdaluje od skutecného cíle. Casto je tento loss spojován se softmax vrstvou. Cross entropy loss lze spocítat rovnicí:

$$H(P, Q) = -\sum_i P(i) log Q(i) \tag{2.27}$$

kde množina P obsahuje hodnotu 1 pokud index i je skutecná klasifikace, jinak obsahuje hodnotu 0, a Q je množina obsahující pravdepodobnosti predikcí.

Kapitola 3

Technologie zpracování textu

3.1 Rozpoznání textu

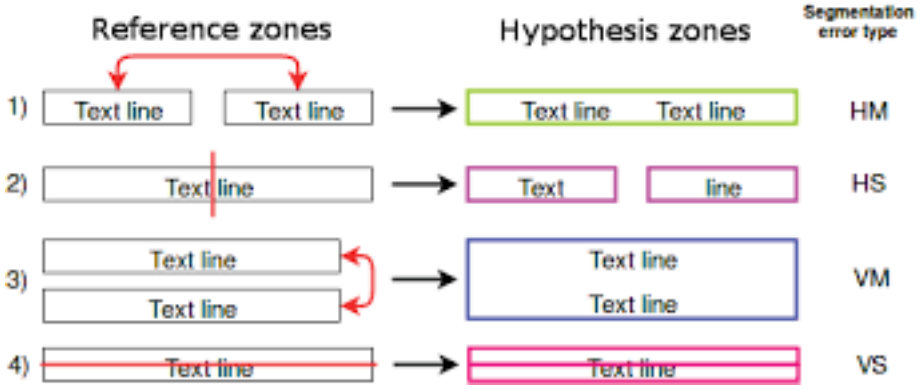
Castokrát je rozpoznání textu považováno za dva rozdílné úkoly, které jsou však propojeny. Temito úkoly jsou detekce a následné rozpoznání textu. Detekce textu nám navrhne pozici, kde se nachází muže nacházet text v obrázku a rozpoznání v navržených textových oblastech rozpozná text. Zpravidla pro obe cásti tohoto algoritmu bývá využita neuronová sít. Novejší prístupy tyto dva úkoly snaží spojit do jediného systému tak, aby využívali co nejvetší pocet spolecných prvku. To má za výhodu, že trénujeme celý systém dohromady. [15, 17, 2]

Vetšina standardních metod detekce textu pracuje s textem jako se sekvencí znaku. Tyto metody nejdríve naleznou jednotlivé znaky v obrázku, které dále shlukují do vyšších celku jako jsou slova prípadne rádky textu. Hlavními dvema pristupy standardních metod jsou posuvné okno (sliding window) a spojené komponenty (connected-components). V poslední dobe se však obrátila pozornost na metody založené na hlubokém ucení [22], které dokáží z obrázku dokumentu prímo oznacit slova.

Obecne cílem rozpoznání textu scény je dekódovat sekvenci znacek z pravidelne oríznutých textových obrázku s ruznou délkou textu. Vetšina metod zachycuje jednotlivé znaky a nekorektne klasifikované znaky jsou opraveny pozdeji. Krome prístupu po jednotlivých znacích existují prístupy zamerené na regiony textu. Ty mužeme rozdelit na tri kategorie: klasifikace slov (word classification), metoda dekodování sekvence na znacky(sequence-to-label) a metoda sekvence na sekvenci (sequence-to-sequence). Metoda klasifikace slov je metoda, kde jsou výstupem trídy reprezentující slova. Tento prístup však není univerzální a rozšíritelný. Zbylé dve metody využívají rekurentních neuronových sítí.

3.2 Metriky porovnání

Algoritmy rozpoznání textu se zpravidla skládají z dvou hlavních celku segmentace obrazu na regiony s textem a rozpoznání textu v techto regionech (kapitola 3.1). Pro porovnání kvality výsledku jednotlivých detekcních algoritmu je potreba zmerit jak na kvalitu segmentace obrazu, tak i kvalitu detekovaných textu v techto regionech. Každý algoritmus pracuje jiným zpusobem a muže nastat situace kdy je text rozdelen do více oblastí (over-segmentation) anebo naopak se nekolik segmentu textu spojí do jednoho celku (under-segmentation). Tyto segmentacní problémy mohou nastat jak v horizontálním, tak i v vertikálním smeru a znesnadnují vyhodnocení výsledku algoritmu. Na obrázku 3.1 mužeme pozorovat varianty segmentacních problému.



Obrázek 3.1: Všechny varianty segmentacních problému. Zdroj: [12]

3.2.1 Levenštejnova vzdálenost

Levenštejnova vzdálenost (také známa jako editacní vzdálenost, anglicky levenshtein distance ci edit distance) je metrika pro urcení podobnosti dvou seznamu. Podobnost meríme jako vzdálenost, s jakou mužeme upravit seznam tak, aby byl totožný s porovnávaným seznamem. Hodnota vzdálenosti nabývá souctu cen jednotlivých operací. Operacemi jsou pridaní prvku (insertion), smazaní prvku (deletion) a nahrazení prvku (substitution). Každá z techto operací muže nabývat jinou cenu za provedení.

Výpocet Levenštejnovi vzdálenosti mužeme vyjádrit následujícím vztahem:

$$
\begin{aligned}
&\mathrm{L}_{x,y}(0,0) = 0 \\
&\mathrm{L}_{x,y}(i,0) = Di \\
&\mathrm{L}_{x,y}(0,j) = I\ j \\
&\mathrm{L}_{x,y}(i,j) = \min \begin{cases} \mathrm{L}_{x,y}(i-1,j) + D \\ \mathrm{L}_{x,y}(i,j-1) + I \\ \mathrm{L}_{x,y}(i-1,j-1) + \begin{cases} S & \text{pokud } x_i \neq y_j \\ 0 & \text{jinak} \end{cases} \end{cases}
\end{aligned}
\tag{3.1}
$$

kde x,y jsou porovnávané seznamy, i,j jsou indexy v rámci techto seznamu x,y. Konstanty I, D, S predstavují ceny operací vložení, odebrání a zmenu prvku. Celková vzdálenost dvou seznamu x a y je získána vypoctením hodnoty $\mathrm{L}_{x,y}(|x|,|y|)$.

3.2.2 ZoneAltCnt

Jedná se o metodu kompletního porovnání kvality algoritmu rozpoznání textu. Metodu navrhl R. Karpinsky [12] a udává, jak vyhodnotit systémy OCR nastanou-li výše uvedené problémy v segmentaci obrazu. Jeho algoritmus nejdríve seskupí predikované a vzorové anotace. Dále jsou z každé skupiny pocítány následující metriky.

- Znaková metrika (Character Metric) udává, jak kvalitne dokáže algoritmus rozpoznat text nezávisle na segmentaci obrazu.
- Slovní metrika (Word Metric) ríká, jak kvalitne dokáže systém OCR segmentovat obraz a rozpoznat v segmentech, kdyby byl segmentovaný stejným zpusobem jako ve vzoru.

- Úplná metrika (Strict Word Metric) udává kvalitu systému OCR i vcetne jeho segmentace v porovnání s vzorovým dokumentem.

Každá metrika se pocítá pomocí leveštejnovi vzdálenosti a je v algoritmu presne urceno, jak se zachovat pri ruzne segmentovaných obrazových datech.

Pro všechny metriky jsou smerodatné dve hodnoty. Míra citlivosti a presnost namerených hodnot. Citlivost (anglicky recall) je pomer mezi korektne vyhodnocenými objekty a celkovým poctem vzorových objektu. Hodnota udává pomer pokrytí všech testovaných vzorových objektu. Zatímco presnost (anglicky precision) je pomer mezi poctem všech korektne vyhodnocených objektu a celkovým poctem všech predikovaných objektu. Cím vyšší presnost tím méne nekorektne predikovaných objektu.

Tato metrika byla v práci použita pro vyhodnocení systému OCR.

3.3 Existující rešení

Tesseract

Tesseract[1] je software pro optické rozpoznání znaku s verejne dostupným kódem. Tesseract zacal jako disertacní projekt v rámci HP Labs v Bristolu. V letech 1984 až 1994 dále pokracoval vývoj v HP. V roce 1995 byl vylepšen a dosáhl lepších výsledku presnosti. Pozdeji roku 2005 byl Tesseract firmou HP uvolnen jako verejne dostupný kód.

Detekcní algoritmus Tesseractu je složen z nekolika cástí. Nejdríve se nad vstupním obrazem provede adaptivní prahování metodou Otsu. Adaptivní prahování je potreba pro zvýšení kontrastu mezi textem a pozadím. Dále nastane analýza rozložení stránky, která rozdelí obrázek na oblasti s textovými a netextovými informacemi. Potom následuje metoda pro nalezení rádku obsahující text. V rámci nalezených rádku se urcí pozice dle horizontálních vzdáleností mezi znaky jednotlivých slov. Slova jsou poté rozdelena na samostatné znaky. Klasifikátor pak urcuje jednotlivé segmenty znaku a detekuje približný jejich geometrický obrys, který je definován vektorem rysu v ctyr-dimenzionálním (pozice x a y, smer, délka).

Microsoft Computer Vision

Jedná se o komercní služby firmy Microsoft, která poskytuje pomocí webového aplikacního rozhraní prístup k jejich produktu[2]. Mezi poskytované služby patrí detekce objektu, které dokáže rozpoznat informace nacházející se v obrázku, oznacit je príslušnými znackami a vrátit jejich pravdepodobnost výskytu. Rozpoznání textu, které muže z obrázku extrahovat slova do sekvence znaku citelnými pocítacem. Další službou je rozpoznání rucne psaných textu, mezi které patrí dopisy, clánky, formuláre a další.

Pro použití této služby je zapotrebí microsoft úcet, pro prístup k aplikacnímu klíci. Detekci textu provedeme zasláním dotazu obsahující výše zmínený klíc a náš obrázek s textem na koncový bod urcený pro detekci textu. V odpovedi dostaneme vyhodnocený text ve formátu JSON (javascript object notation). Ve výsledku mužeme pozorovat rádky textu a jednotlivá slova uvnitr techto rádku. Záznamy se slovy obsahují hodnotu textu, jež systém vyhodnotil, a také souradnice oblasti (bounding polygon), kde se v rámci obrázku text nachází. Struktury vyšší úrovne, napríklad paragrafy, namísto slov samotných obsa-

[1]Dostupné z `https://github.com/tesseract-ocr/`

[2]Dostupné na `https://azure.microsoft.com/cs-cz/services/cognitive-services/computer-vision/`

hují jen souradnice oblasti, do níž spadají všechna jejich slova. Prumerná doba detekce, díky komunikaci se serverem, dosahuje kolem jedné vteriny.

Algoritmus využitý pro rozpoznání textu není zverejnen. Algoritmus je dle mého názoru implementovaný jako série operací úpravy obrázku, nalezení rádku textu, narovnání techto rádku (jsou-li rádky natoceny vuci horizontální ose textu), rozdelení na slova a na jednotlivé znaky techto slov, následuje rozpoznání znaku, které je pravdepodobne implementované klasifikátorem pomocí neuronové síte.

Google Cloud Vision

Poskytuje nekolik možností, jak zaintegrovat natrénované modely pocítacového videní do programu tretích stran. Hlavním zpusobem, jak takovou službu integrovat je za pomocí Cloud Vision aplikacního rozhraní (Cloud Vision API[3]). Toto rozhraní definuje koncové body a odpovedi pro jednotlivé definované moduly a probíhá pomocí webového protokolu HTTP. Mezi nabízenými službami jsou detekce objektu ve scéne obrázku, extrakce textu z obrázku, vyhledávání obrázku na webu (také na základe podobnosti obrázku).

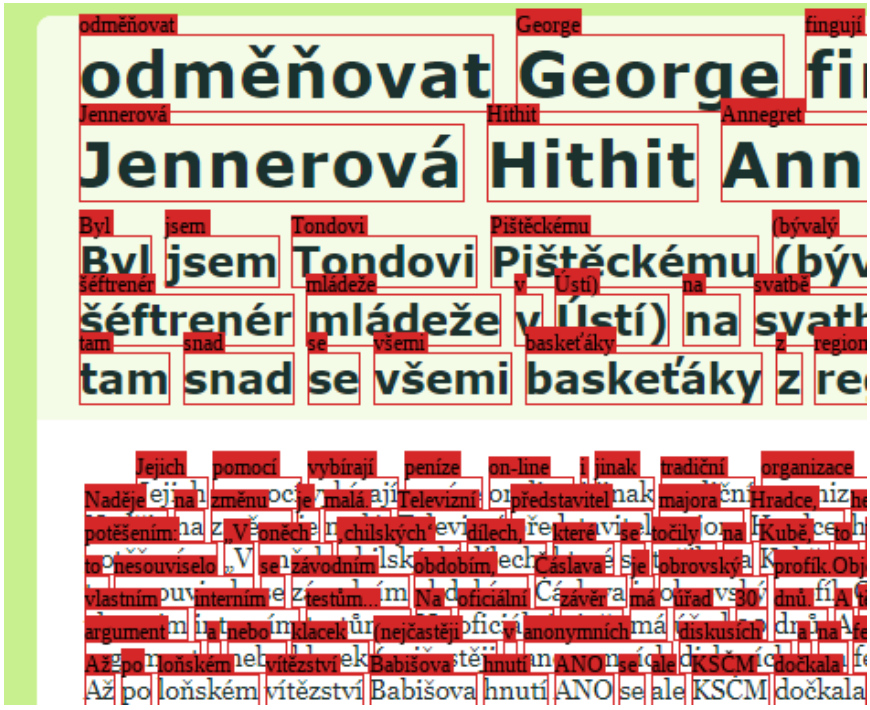
Podobne jak je tomu u Microsoftu (viz výše) je i u Googlu zapotrebí úcet a zaregistrování tohoto úctu v rámci Google Vision. V rámci úctu máme dostupný privátní klíc pro prístup k dostupným službám. Detekce textu se provede zasláním tohoto klíce soucasne s obrázkem, u nehož chceme detekovat text, na koncový bod pro detekci textu. Výsledek je vrácen ve formátu JSON. Detekce je vracena zanorenou hierarchií stránky, blok, paragraf, slovo, symbol. Každá z techto struktur má definovanou oblast kde se v obrázku nachází a predstavuje logické celky odpovídající daným názvum.

[3]Dostupné na `https://cloud.google.com/vision/`

Kapitola 4

Datová sada

Datová sada byla vytvorena pomocí interního nástroje firmy ARTIN zvaného „gui-generator“. Nástroj, jak již název vypovídá, dokáže generovat rozložení ruzných grafických webových rozhraní, které si mužeme zobrazit pomocí webových prohlížecu. Pomocí tohoto nástroje je možné vytvorit široké spektrum webových stránek s ruznými barevnými schématy, rozloženími prvku a množstvím textu. Hlavní úcelem gui-generátoru je generování trénovacích datových množin pro rozpoznávac aktivních prvku grafických uživatelských rozhraní. S každým vygenerovaným rozhraním jsou zároven vytvoreny i anotace pro jednotlivé prvky obrazu. Anotace zpravidla obsahují informace o typu prvku (napríklad tlacítko, slovo textu atd.) a pozice tohoto prvku v rámci obrazu. Pro úcely této práce bylo zapotrebí tento generátor rozšírit o další funkcionalitu. Výsledek tohoto generátoru je možné videt na obrázku 4.1. Obrázek je priblížen a je v nem také možné pozorovat bounding boxy, které dále poslouží neuronové síti jako ucební vzory daných textu.



Obrázek 4.1: Ukázka vygenerovaných dat.

4.1 Texty

U generátoru bylo potreba rozšírit rozsah textu, které je možné generovat. Hlavním duvodem tohoto rozšírení byla malá rozmanitost textu. Nové texty pro generátor byly sestaveny z ruzných ceských textu posbíraných z internetu. Získané texty obsahují 120466 ceských vet, 167589 ceských slov a v nekomprimované podobe zabírají 23 MB diskového prostoru. Dále bylo potreba uzpusobit generátor, aby dokázal generovat webové stránky, jež obsahují nová textová data.

Puvodní verze generátoru vytvárela anotace pouze pro prvky rozložení dokumentu a interaktivní webové prvky. Toto však nebylo dostacující pro vytvorení anotovaných dat vhodných k rozpoznání textu. Pro interaktivní prvky (formulárové prvky a.j.) však nebylo možné prímo pristoupit k jejich hodnotám v rámci objektového modelu dokumentu (anglicky Document Object Model dále jen DOM). Díky tomu nebylo možné takové texty anotovat. Interaktivní prvky proto byly nahrazeny statickými ve stejném zobrazovacím stylu a jejich hodnoty byly vloženy jako potomci. Nyní bylo možné všechny textové informace adresovat pomocí DOM. Textová data byla rozdelena a vložena do nenastylovaných prvku „span“, které nám poskytli možnost urcit souradnice oblasti ve které se nachází texty jež tyto prvky obalují. Pridané anotace jsou pro blok textu, slova textu a jednotlivé symboly textu. Bílé znaky generátor vynechává a nijak je neanotuje.

4.2 Styly

Pro vetší ruznorodost generovaných textu v snímkách byl pridán do generátoru náhodný výber ze širokého seznamu písem. Duvodem této zmeny bylo otestování rozpoznávací schopností pro ruznorodé styly textu. Pro každou generovanou stránku jsou náhodne zvoleny písma pro nadpisy, tlacítka a všeobecný text. Zároven pri tom je i náhodne zvolena velikost tohoto písma, vzdálenost mezi jednotlivými znaky a výška mezi dvema rádky textu. Jiné modifikace textu (jako je náklon písma, tucné písmo atd.) jsou voleny náhodne pri vkládání písma do stránky.

Generátor generoval hlavne formuláre a jiné webové rozložení a spíše se zabýval množstvím a pozicí aktivních prvku v rámci techto webových stránek. Pro úcely této práce bylo zapotrebí u soucasných stylu vymenit generované texty (viz výše). Protože texty mohli nabývat vetší velikosti (oproti puvodnímu stavu), bylo potreba nekteré styly pozmenit, aby zmenený text neznicil vytvorené rozložení. Zároven byl pro generátor vytvoren i ciste textový styl. Tento styl generuje internetový clánek a skládá se z hlavních cástí hlavicky a tela clánku. V rámci hlavicky se vytvorí hlavní nadpis clánku a strucný úvod co mužeme v clánku hledat. Telo clánku pak obsahuje množství paragrafu a náhodne do nej mohou být vygenerovány i obrázky. Generátor tento styl naplnuje pomocí získaných textu. Barevné rozložení techto stylu je generováno tak, aby pro zvolenou barvu pozadí byl text citelný.

4.3 Generování testovací datové sady

Pro automatické generování datové sady byl vytvoren nástroj, jenž automaticky ovládá webový prohlížec, který je potreba pro vykreslení rozložení (layout) poskytnutém pomocí gui-generatoru. Toho bylo dosáhnuto využitím knihovny Selenium[1], jejíž hlavním úcelem je vytvorení a spuštení automatizovaných testu webových rozhraní, a programu Chrome-

[1]Dostupné z `https://www.seleniumhq.org/`

Driver[2](nadstavba nad webovým prohlížecem Google Chrome), který obsahuje rozhraní pro možnost ovládání robotem (v tomto prípade Selenium). Nástroj nejprve vytvorí snímek obrazovky ovládaného prohlížece (tím získáme obrázek s texty) a následne pro tento obrázek získáme anotace pro slova a jejich symboly. Struktura vygenerované anotace pro snímky vypadá následovne:

```
{
  "filename": "2019_01_07__09_54_36.png",
  "filepath": "/images/2019_01_07__09_54_36.png",
  "image_size": {
    "width": 1920,
    "height": 1080,
    "depth": 3
  },
  "annotations": [
    {
      "box": {
        "xmin": 600,
        "xmax": 713,
        "ymin": 1142,
        "ymax": 1158
      },
      "type": "text_word",
      "text": "implementovanou"
    },
    ...
```

Vidíme zde název(filename), cestu(filepath) a velikost obrázku(image_size), který této anotaci odpovídá. Následuje seznam anotaci pro tento obrázek, které se skládají z bounding boxu urceného dvema body, typu oznacující anotace(type) a hodnoty text.

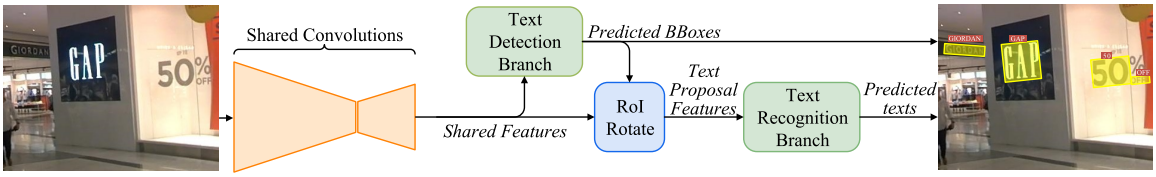
[2]Dostupné z `http://chromedriver.chromium.org/`

Kapitola 5

Návrh rešení

Navržený systém využívá architekturu inspirovanou Fast Oriented Text Spotting [15]. Architektura využívá jednu spolecnou sít jak pro detekci i rozpoznání textu. Skládá ze ctyr hlavních cástí: spolecná konvolucní sít, detekce textu, prostorové transformace a rozpoznání textu.

Celkovou architekturu mužeme pozorovat na obrázku 5.1. Spolecná konvolucní sít (shared convolutions) ze svého vstupu získá obrazové vlastnosti (features). Pro nalezení vlastností v obraze bude použita reziduální konvolucní sít inspirovaná sítemi ResNet50 [**?**] a Feature Pyramid [14], které z obrazu extrahuje vlastnosti a spojí je dohromady pres nekolik úrovní. Výsledná mapa obrazových vlastností je spolecným vstupem pro cást s detekcí a rozpoznáním textu. Detekce textu (Text Detection Branch) produkuje pro každý pixel obrazu predikce regionu textu za použití spolecné mapy vlastností. Regiony textu, navržené detekcí textu, jsou dále transformovány, za pomocí prostorové transformace, do datové reprezentace s pevnou výškou. Tato operace zachovává puvodní pomer stran regionu. A nakonec je aplikováno rozpoznání textu (Text Recognition Branch). Pro tyto úcely budou použity konvolucní neuronové síte s pametovými vrstvami (Long Short-Term Memory dále jen LSTM [9, 19]), které zakódují sekvenci textových informací. Následuje klasifikace textu pomocí rekurentní neuronové síte CTC (Connectionist Temporal Classification [7]) Strukturu cástí detekce, korekce a rozpoznání si více rozvedeme v následujících kapitolách.



Obrázek 5.1: Architektura síte FOTS. Zdroj [15]

5.1 Detekce textu

Detekce textu je tvorena plne konvolucní sítí. V obrázku se spíše nachází vetší množství malých textových regionu, proto je výstupní mapa obrazových vlastností zvetšena. Po extrakci obrazových vlastností sdílenou sítí je na ne aplikována jedna konvoluce na zjištení prítomnosti textu v daných pixelech. Dalšími predikovanými hodnotami pro všechny pozitivní vzorky jsou ctyri dimenze vzdáleností (vzdáleností nahoru, doprava, dolu, doleva), které

urcují bounding box obsahující daný pixel v závislosti na jeho souradnicích. A rotace získaného bounding boxu vuci horizontální ose. Na všechny detekce je pak aplikována metoda non-maximum suppression [16] (dále jen NMS), která zanechá pouze nejpravdepodobnejší predikované hodnoty.

Hodnotící funkce detekce se skládá z dvou částí: ztráta pri klasifikaci textu a ztráta regrese bounding boxu. Ztrátu klasifikacní funkce vypocítáme jako ztrátu klasifikace jednotlivých pixelu pro zmenšenou mapu hodnocení (anglicky score map). Pouze zmenšená verze puvodního textového regionu je považována za správnou oblast. Oblast mezi bounding boxem a zmenšenou verzí mapy nijak neprispívá k celkové ztráte klasifikace. Zatímco regresní ztráta pro bounding boxy je spocítána pomerem pruniku a sjednocení jejich oblastí (Intersection over Union dále jen IoU [18]) a pomocí ztráty úhlu rotace. Celková ztráta je vážený soucet techto dvou ztrát.

5.2 Prostorová transformace

Pomocí operátoru RoIPool provedeme korekci predikovaných textových oblastí. RoIPool aplikuje transformaci na textové oblasti a získáme textové oblastí zarovnané podle souradných os (anglicky axis-aligned). Textové oblasti transformujeme tak, že pro všechny nastavíme stejnou pevnou výšku a šírku dopocítáme tak, aby zustal pomer stran nezmenený. To umožní zpracování ruzne dlouhých textu. Pro vzorkování hodnot, jejichž souradnice jsou mapovány mimo diskrétní celocíselné, je použita bilineární interpolace. Tím zamezíme chybám v zarovnání mezi korekcí a získanými vlastnostmi. Navíc tím umožníme promennou délku výstupních oblastí, což je více vhodné pro rozpoznání textu.

Korekce aplikuje afinní transformace na mapu vlastností získané z predchozího kroku. Afinní transformace prevedou oblast do zarovnání podle souradných os. Transformace využívají reverzní mapování ze zdrojových dat, pro interpolaci neznámých hodnot je použita bilineární interpolace.

5.3 Rozpoznání textu

Rozpoznání textu je zamereno na oznacení veškerých textových informací z dat získaných z predchozích kroku. Tato cást se skládá ze sekvencní konvolucní a pooling sítí podobných VGG, obousmerného LSTM, plne propojené vrstvy urcené ke klasifikaci a zakoncené CTC dekodérem [7].

Nejprve je na prostorová data aplikováno série konvolucí a poolingu. Pricemž pooling vrstvy jsou nastavené, aby redukovaly rozmery pouze ve vertikální ose. To má za dusledek, že extrahujeme vlastnosti do vyšších úrovní a zachováváme šírku nezmenenou. Tyto vlastnosti vyšší úrovne jsou prevedeny do sekvence vlastností na základe horizontální šírky vstupních vlastností. Ty jsou vstupem rekurentní neuronové síte LSTM. LSTM zakóduje získanou sekvenci do 256dimenzionálního vektoru pro každý smer. To je potreba pro zachycení všech rozmerových závislostí, kterých text muže nabývat. Každý ze skrytých stavu LSTM je vyhodnocen v obou smerech v každé casové jednotce. Tyto stavy jsou dále napojeny do plne propojené vrstvy, která vypocítá spolecne s vrstvou softmax hodnoty pravdepodobnost výskytu znaku z dané abecedy. Následne je použito CTC pro dekódování jednotlivých výstupu z rámcu na výstupní sekvencí znaku.

Kapitola 6

Implementace

V této kapitole je popsána implementace navrženého systému. Nejprve jsou uvedeny použité technologie pro implementaci systému v 6.1. Následuje popis algoritmu ucení a inference v 6.2. V dalších kapitolách jsou popsány jednotlivé části navrženého systému a ke konci v kapitole 6.7 jsou shrnuty problémy, které vznikly pri implementaci.

6.1 Použité technologie

Python

Python je vysokoúrovnový skriptovací jazyk. Poskytuje programování v ruzných paradigmatách jako jsou objektove orientované, imperativní, procedurální i funkcionální. Jazyk poskytuje systém pro správu závislostí pip. V posledních letech pri výrazném nástupu strojového ucení se stal velmi oblíbený pro svoji jednoduchost a také kvuli tomu, že jej lze navázat na zkompilované knihovny (napríklad v jazyce C) s nimiž dosahuje jazyk vyššího výkonu. Dalším duvodem použití je že lze pomocí nej snadneji prototypovat algoritmy neuronových sítí v nekterém z dostupných frameworku.

Pytorch

Knihovna PyTorch[1] je framework pro práci s neuronovými sítemi pro jazyk Python. Je urcen k návrhu a implementaci neuronových sítí a jejich následnému použití. Knihovna PyTorch je zdarma, a dokonce i opensource, vyvíjená týmem zabývajícím se umelou inteligencí pro Facebook. Tato knihovna nám poskytuje nekolik hlavních výhod jako je pocítání neuronových sítí za pomocí tenzoru se velkou akcelerací pomocí grafických procesoru (GPU). Tenzory mužeme jsou matematický objekt, který v rámci programování mužeme chápat jednoduše jako multidimenzionální pole. PyTorch používá funkci zvanou automatic differentiation. Jedná se o metodu, která sleduje zmeny provedené na tenzorech a je schopna tyto operace revertovat a na základe nich vypocítat gradienty. To je velmi užitecné pri tvorbe neuronových sítí, jelikož umožnuje výpocet diferencí parametru už pri dopredném pruchodu sítí. PyTorch navíc poskytuje mnoho implementací funkcí, vrstev, aj., které byly zmíneny v kapitole 2. To je duvodem, proc byl v práci použit.

[1]Dostupné na adrese `https://pytorch.org/`

6.2 Rozdelení systému

Systém je rozdelen na dva spustitelné soubory, každý s jiným úcelem. Prvním z nich je aplikace urcená pouze k ucení síte. Prijímá na vstupu konfiguracní soubor, který udává parametry síte, kterou chceme ucit. V konfiguracním souboru se nachází pocet a rozmery anchor boxu, cesta k datasetu na kterém bude algoritmus ucit model síte, sekvenci transformacních operací, které slouží jako preprocessing datasetu. Dataset je nacítán ve formátu uvedeném výše v kapitole 4 a je na nej aplikována série transformacních operací. Mezi ne patrí zarovnání obrázku z datasetu do jednotného paddingu, aby bylo možné reprezentovat více prvku z datasetu jednotne i když mohou mít variabilní informace jako je treba pocet bounding boxu v obrázku nebo ruzne dlouhé texty. Další významnou transformací je jejich prevod dat na tenzory, které jsou kompatibilní s knihovnou PyTorch. Ucení je pak dále rozdeleno na epochy, které reprezentují pocet pruchodu datasetem pri stejných parametrech ucení (learning rate). V každé epoše se nejprve zamíchá poradí prvku datasetu. Pak se postupne dataset prochází a aplikují se na nej transformace. Pred zpracováním každého prvku datasetu se vynuluje gradient modelu, a to z duvodu výpoctu gradient už pri pruchodu dopredném pruchodu sítí. Poté necháme model dopredným pruchodem zpracovat tyto prvky. Pri verzi ucení získáme na výstupu pouze loss funkce síte, protože predikované výsledky modelu nejsou pri ucení príliš užitecné. Za pomocí loss funkcí zpetným pruchodem modifikujeme gradienty modelu, k tomu je užitecná knihovna pytorch a její modul autograd. Gradienty síte jsou spocítány tak jsou pomocí optimalizacního algoritmu aplikovány. V rámci tohoto projektu byl použit optimalizátor Stochastic Gradient Descent, který umožnuje iterativne aplikovat metodu ucení s ucitelem gradient descent (více v kapitole 2.5.1). Po dokoncení epochy je model uložen a upraven krok plánovace ucení. Plánovac ucení je rozdelen na dve cástí. Zpocátku do nastaveného poctu epoch je ucení postupne „zahríváno“, kdy zacneme s nízkou váhou ucení a postupne ji zvyšujeme, dokud nedosáhneme nastavené epochy. Tento princip nazýváme gradual warmup [20] a umožnuje síti si „zvyknout“ na vstupní data. Jakmile je model „zahrátý“ na ucení je použit druhý plánovac Reduce On Plateau, který bere v potaz vypocítaný loss modelu. Pokud výsledky modelu se nezlepší (hodnoty se neminimalizují) po dobu nazývanou „trpelivost“, pak sníží váhu ucení. Celý tento systém mužeme pozorovat jako pseudokód v algoritmu 1.

Algoritmus 1: Ucení systému.

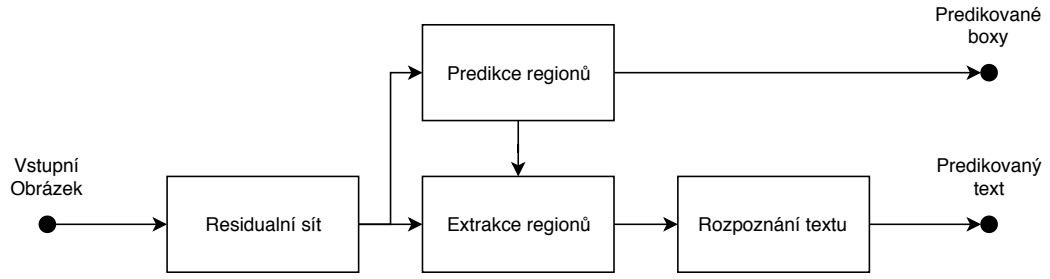
```
for i = 0; i <MAX_EPOCH; i++ do
    Zamíchání indexu datasetu.
    for j = 0; j <NUM_ITEMS_DATASET; j++ do
        Získání dat datasetu pro index j.
        Aplikace preprocessing transformací na tato data.
        Vynulování gradientu síte.
        Loss ← Dopredný pruchod sítí.
        Zpetný pruchod sítí pomocí vypocítaných lossu.
        Krok optimalizacního algoritmu síte.
    Vytisknutí výsledku ucení epochy.
    Uložení stavu síte pro danou epochu.
    Krok plánovace ucení síte.
```

Druhou verzí systému je samotná inference. Máme již naucený model pomocí predchozí aplikace a chceme predikovat boxy i texty pro zadaný vstupní obrázek. Aplikace nacte obrázek a aplikuje na nej nezbytné transformacní operace, které jsou vyžadovány modelem. Mezi ne patrí normalizace velikosti obrázku a jeho následné prevedení na tenzor. Takto získaný tenzor je predaný modelu nastaveného na evaluaci. Po dopredném pruchodu modelu dostaneme predikované regiony textu, jejich textový obsah a pravdepodobnost, kterou model považuje za validní box. Algoritmus inference je možné videt v algoritmu 2

Algoritmus 2: Inference systému.

1 Nactení stavu síte z poslední epochy.
2 Nactení obrázku ze zadané cesty
3 Aplikace preprocessing transformací na tato data.
4 Skóre, Boxy, Texty ← Dopredný pruchod sítí.
5 Výstup techto dat do souboru.

Model tohoto systému je možné rozdelit do nekolik funkcne odlišných cástí. Tyto cásti mužeme pozorovat na blokovém schéma systému v obrázku 6.1, a budou dále popsány v následujících kapitolách.

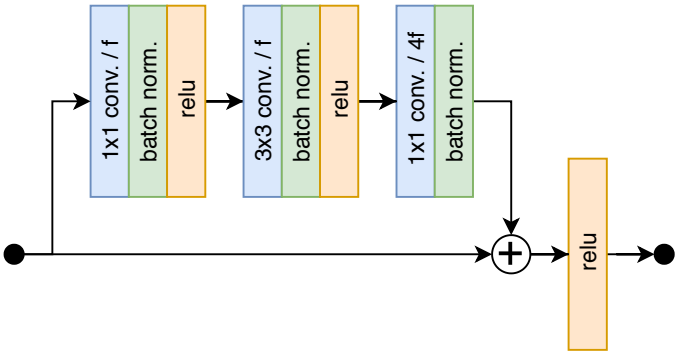


Obrázek 6.1: Celková architektura modelu síte.

6.3 Residuální sít

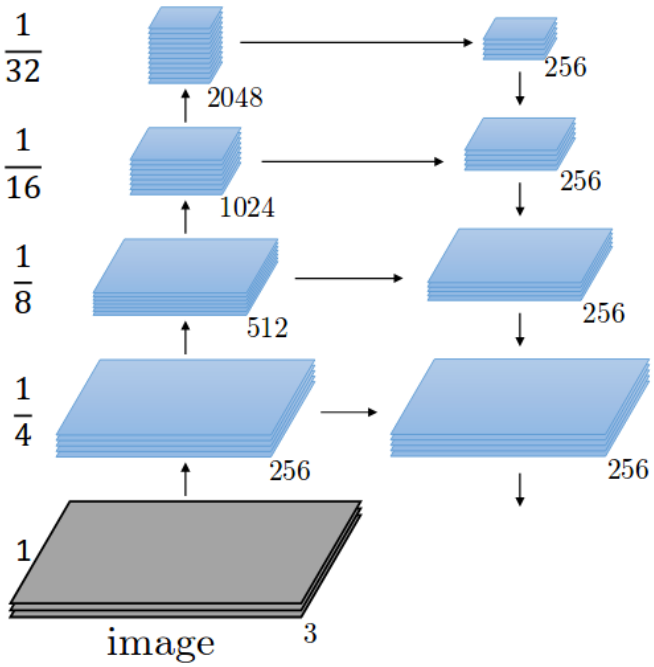
Prvním modulem neuronové síte je residuální neuronová sít. Tento druh síte slouží získání všech možných vlastností ze vstupního obrazu a je casto predtrénován na ImageNetu[3]. To je dataset, který obsahuje miliony ruzných obrázku v ruzných kategoriích. Použití predtrénovaných vah je vhodnejší, protože výsledná sít bude rychleji konvergovat. Tuto praktika nazýváme transfer learning [17] a umožnuje sítím ucit se nové úkoly snadneji, protože nemusí zacínat od nuly s náhodnými váhami.

V rámci této práce je residuální sít je navržena po vzoru ResNet-50 [8]. Sít se skládá z takzvaných residuálních bloku, které umožnují vstupním datum projít skrz sít beze zmeny. Díky tomu je možné touto sítí dobre šírit gradient. V každém z techto bloku existuje propojení, které umožní vstupním hodnotám preskocit konvolucní operace. Residuální bloky se také liší v poctu kanálu, které produkují. V obrázku 6.2 mužeme pozorovat residuální blok. V ResNet-50 se jedná o 4 vrstvy, kde každá má predem stanovaný pocet bloku, které jsou poskládány za sebe. První blok každé z vrstev využívá prostorový krok 2 a tím postupne snižuje rozmery vstupu.



Obrázek 6.2: Residuální blok síte ResNet-50

Tato sít však postupným pruchodem vstupu jeho vrstvami snižuje velikost vstupu až 32krát, címž se velmi rychle ztrácí data o malých objektech v obrazu. To casto nemá dopad napríklad pri rozpoznání objektu, kdy se v obraze nachází jen nekolik objektu. Avšak je to velmi nepraktické pri detekci textu, protože pocet objektu v obraze muže být daleko vetší. Sít ResNet-50 je upravena po vzoru Feature pyramid network [14], tak aby byla více citlivá na vstupy menších rozmeru. Na obrázku 6.3 mužeme pozorovat výslednou residuální sít. Všechny výstupy z vrstev ResNet jsou využity pri výpoctu výstupní mapy vlastností. Každá z vrstev má takzvaný lateral layer (na obrázku horizontální propojení), který normalizuje kanály všech vrstev na 256. Vrstvy jsou pak odshora kombinovány s podélnými pomocí souctu pres všechny elementy. Nejprve musí být hornejší vrstvy nadvzorkovány, aby odpovídali velikosti vrstvy podélného propojení.



Obrázek 6.3: Feature pyramid network vycházející z ResNet. Zdroj: `http://presentations.cocodataset.org/COCO17-Stuff-FAIR.pdf`

Výsledkem tohoto modulu je mapa vlastnosti obsahující 256 kanálu pro každý pixel. Velikost této mapy je 4x zmenšená oproti velikosti vstupního obrazu. Získaná mapa vlastností je sdílená pro všechny ostatní moduly.

6.4 Predikce regionu

Tato cást popisuje získání bounding boxu v zadaném obraze. Získané boxy slouží jako výstup systému a jsou použity pro další zpracování, pomocí nehož se predikuje text nacházející se práve v techto boxech. V následujících podkapitolách je popsáno, jak k predikci boxu dochází.

6.4.1 Anchory

V rámci této práce je potreba predikovat bounding boxy, které obsahují text. Avšak bounding boxy mají ruzné rozmery, jejich pocet i pozice jsou také promenné. Tyto vlastnosti mohou vést ke špatné konvergenci ucícího se systému. Proto bylo potreba tyto bounding boxy reprezentovat zpusobem, který není závislý na rozmeru boxu. Rešením je predem stanovit bounding boxy jejichž pocet a rozmery budou pro každý pixel mapy vlastností konstantní, a budou se lišit pouze v posunu. A na tyto konstantní bounding boxy, které dále budeme referovat jako anchory, aplikovat transformace. Tento prístup je inspirován dle publikace [21].

Máme-li 4 souradnice anchoru, pozice stredu, šírku a výšku (a_x, a_y, a_W, a_h), mužeme ho transformovat za pomocí 4 transformacních váh (d_x, d_y, d_w, d_h), které urcují míru posunu stredu a zmenu merítka šírky a výšky. Rovnice pro transformaci anchor boxu jsou uvedeny níže:

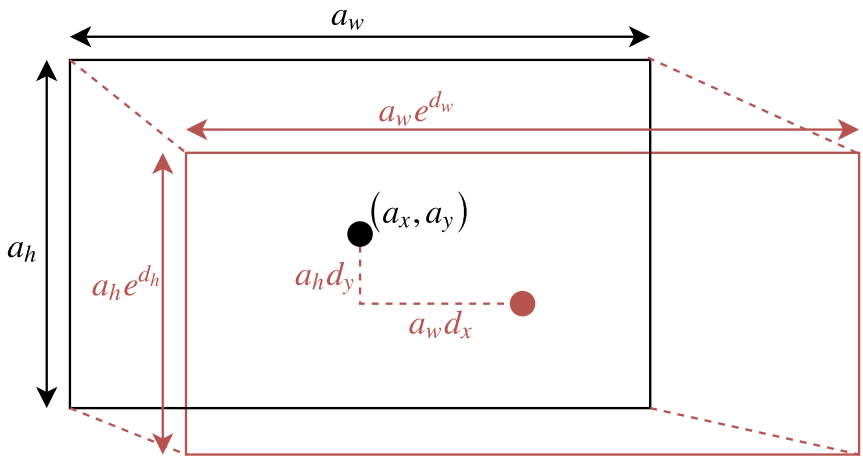
$$r_x = a_x + a_w d_x, \tag{6.1}$$

$$r_y = a_y + a_h d_y, \tag{6.2}$$

$$r_w = a_w e^{d_w}, \tag{6.3}$$

$$r_h = a_h e^{d_h}, \tag{6.4}$$

kde souradnice (r_x, r_y, r_w, r_h) znázornují transformovaný bounding box. Takto definovanou transformaci je možné snadno ucit, protože hodnoty (d_x, d_y, d_w, d_h) jsou invariantní vuci rozmerum bounding box, na které je aplikována. Tyto hodnoty také mohou nabývat jakékoli hodnoty, protože jejich definicní obor není omezen. Jsou-li tyto hodnoty samé 0 pak je zachován puvodní bounding box. Grafické znázornení, jak transformace modifikuje box je možné videt na obrázku 6.4, kde v cerné barve je znázornen anchor box, na který je aplikována transformace, a v cervené barve je možné pozorovat výsledek této transformace.

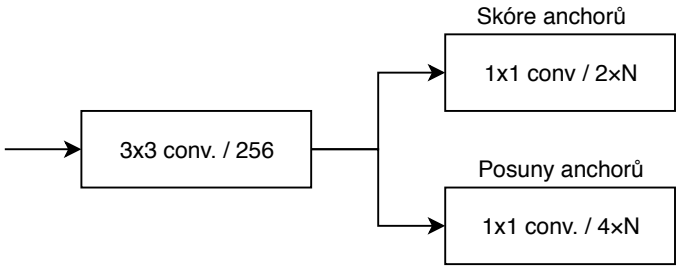


Obrázek 6.4: Transformace anchor boxu pomocí transformacních parametru.

Protože se tato práce zabývá zpracováním textu bylo, potreba zvolit anchor boxy, které alespon matne pripomínají bounding box obsahující text. Podíváme-li se na jakýkoli text, mužeme videt, že vetšina jeho slov rozmerove je více široká, nežli vysoká. Proto je i logické, aby v tomto duchu byly zvoleny i anchor boxy. Konkrétneji jsou zvoleny boxy s pomerem šírky a výšky 1:1, 2:1, 4:1 a 8:1, pri merítkách 8, 16, 32 a 64. To dohromady dává 16 anchor boxu, pro každý pixel mapy vlastností.

6.4.2 Modul predikce regionu

Tento modul slouží pro detekci regionu v obraze. Využívá získanou mapu vlastností a provede nad ní konvolucní vrstvu s 3x3, která slouží pro vyhlazení této mapy. Na ni jsou aplikovány další dve konvolucní vrstvy urcené ke klasifikaci, resp. regresi transformacních souradnic anchor boxu. Schéma modulu mužeme videt na obrázku 6.5



Obrázek 6.5: Region proposal network. Promenná N znací pocet anchoru.

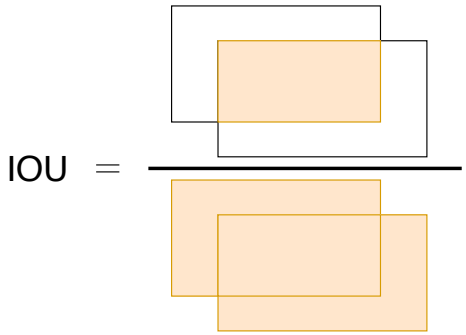
Klasifikacní vrstva obsahuje pro každý pixel mapy vlastností dvojici hodnot pro každý anchor na které je dále aplikována vrstva Softmax (více kap. 2.4.5). Tyto dve hodnoty pro každý anchor urcují míru pravdepodobnosti, že práve tento anchor spadá do jedné ze dvou navzájem se vylucujících tríd pozadí nebo text. Pri ucení je potreba v preprocessingu pro dávku pripravit vzory, podle kterých se klasifikátor bude ucit. Jak vzory získáme vidíme v následující kapitole 6.4.3. Hodnocení klasifikacní vrstvy je spocítáno za pomocí Cross-entropy loss funkce, to je varianta Negative log-likelihood lossu popsaného v kapitole **??**,

který je nastaven, aby ignoroval trénovací vzor obsahující hodnotou -1. Výsledný loss této vrstvy znacíme $\mathcal{L}_{\text{CLS}}$.

Regresní vrstva podobne jako klasifikacní vrstva obsahuje pro každý pixel mapy vlastností ctverici hodnot, které nám dávají hodnotu transformacního vektoru pro daný anchor (viz výše 6.4.1). Podobne jako je tomu v klasifikacní vrstve musí mít i tato vrstva pri ucení preprocessovány trénovací vzory. Ty jsou v tomto prípade transformacní hodnoty, které když aplikujeme na daný anchor tak dostaneme nekterý z validních textových boxu. Pri preprocessingu je zároven získána i maska, která jednoduše urcuje, které hodnoty je treba aktualizovat a jsou s ní násobeny hodnoty regresní vrstvy. Pro ty hodnoty, které chceme aktualizovat má maska hodnotu 1 a zbylé mají hodnotu 0. Hodnota chybové funkce je spocítána pomocí Smooth L1 Loss (s redukcí pomocí operace scítání, více 2.5.2) nad rozdílem predikce vynásobené maskou a trénovacím vzorem. Vypoctenou chybu pro regresní vrstvu znacíme $\mathcal{L}_{\text{REG}}$. Masku je použita, protože všechny hodnoty masky nastavené na 0 se nebudou podílet na získání hodnoty chybové funkce. Je-li hodnota masky pro nejaký anchor nastavená na hodnotu 0 pak je i hodnota vzorové transformace pro daný anchor nastavená práve na 0. Rozdíl predikce vynásobené maskou a vzoru pro tento anchor se rovná 0, což znamená, že pro tuto kombinaci není, co bylo treba optimalizovat.

6.4.3 Príprava vzoru pro ucení

Aby se sít naucila klasifikovat ty správné boxy, jsou v preprocessingu náhodne navzorkovány pozitivní a negativní anchory, všechny ostatní jsou v trénování vynechány. Vzorky, které mohou být považovány za pozitivní, ci negativní, a mohl je algoritmus náhodne vybrat jako kandidáty k ucení, musí se anchor box a vzorový anotovaný box z velké cásti prekrývat. Hodnota prekryvu je dána funkcí Intersect over Union [18] (dále jen IOU), která vypocítá pomer mezi plochou sdílenou mezi boxy a celkovou plochou kterou boxy dohromady zabírají. IOU nabývá hodnoty 1 pokud jsou boxy totožné a 0 pokud boxy se vubec neprekrývají. Tuto funkci mužeme pozorovat graficky v obrázku 6.6.



Obrázek 6.6: Funkce IOU znázornena graficky pro dva boxy.

Dalším kritérium výberu anchoru, je že žádny z kandidátních anchoru pro trénování nesmí presahovat za hranice obrázku. Všechny anchory, které ano jsou ignorovány. Algoritmus nejdríve vybere všechny kandidáty, které nejlépe prekrývají jeden z anotovaných vzorových bounding boxu. Dále jsou náhodne dovybrány ze všech kandidátu ti, kterí mají hodnotu prekryvu vyšší, než je stanovený pozitivní práh (v tomto projektu se jedná o hodnotu 70%), aby byl splnený pocet pozitivních záznamu. Všechny vybrané pozitivní záznamy, ztotožnují anchory jejichž ohodnocení chceme zlepšovat. Jakmile jsou všechny pozitivní záznamy vybrány je potreba ješte urcit negativní anchor boxy. Ty se vybírají oproti pozitivním boxum

opacne. Ze všech zbylých kandidátu se vyberou ty, které mají hodnotu prekryvu nižší než stanovený negativní práh. To jsou zase hodnoty anchoru, které pri ucení zhoršíme. Všechny ostatní anchory ignorujeme. Vzor trénování klasifikace nabýva trech hodnot, 0 pokud se jedná o negativní anchor , 1 pokud je anchor pozitivní a -1 pokud je anchor box pri trénování ignorován. Pocet techto vzorku a pomer pozitivních vuci negativním anchorum a hodnota hranice, kdy je box považován jako kandidát pro ucení (at už pozitivní, ci negativní vzorek), je nastavitelný v konfiguracním souboru.

Pro všechny takto vybrané pozitivní anchory je dále dopocítán i vzor pro regresi transformacních funkcí. Záporné anchory jsou v optimalizaci regrese vynechány, protože to ani není logické. Pro každý z techto pozitivních anchoru je vybrán anotovaný box, který má nejvetší hodnotu prekryvu. Tím pádem se daný anchor bude snažit co nejverneji napodobit tento anotovaný box. Proto je potreba urcit transformacní vektor, který je potreba k transformaci daného anchor boxu na anotovaný box. Ten získáme pomocí následujících rovnic:

$$t_x = \frac{g_x - a_x}{a_w}, \tag{6.5}$$

$$t_y = \frac{g_y - a_y}{a_h}, \tag{6.6}$$

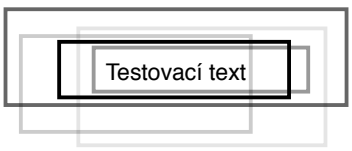
$$t_w = \ln \frac{g_w}{a_w}, \tag{6.7}$$

$$t_h = \ln \frac{g_h}{a_h}, \tag{6.8}$$

kde vektor (a_x, a_y, a_w, a_h) obsahuje parametry anchor boxu, vektor (g_x, g_y, g_w, g_h) obsahuje parametry anotovaného boxu, kterého chceme dosáhnout, a vektor (t_x, t_y, t_w, t_h) obsahuje transformacní parametry podle niž se bude sít pro daný anchor box optimalizovat, aby dosáhla anotovaného boxu. Zároven s výpoctem tohoto transformacního vektoru pro daný anchor je pro nej nastavena hodnota maska (M_x, M_y, M_w, M_h) na $(1, 1, 1, 1)$. Pro všechny nezúcastnené anchor boxy jsou nastaveny vektory vzoru transformacních parametru a masky na hodnotu $(0, 0, 0, 0)$.

6.4.4 Filtrování predikovaných regionu

Z duvodu, že se pro každý pixel mapy vlastností predikují transformace mnoho (v našem prípade 16) anchor boxu. Kvuli jejich obrovskému množství (960000 predikcí pro obrázek velikosti 1200x800) je potreba z techto vzorku vybrat ty nejvhodnejší. Nejprve seradíme sestupne ohodnocení pravdepodobnosti boxu, že je text. Z techto pravdepodobností vybereme prvních N boxu (konfigurovatelné pomocí hodnoty `num_boxes_pre_nms`). Tím dostaneme pouze ty nejlépe ohodnocené boxy, které byly predikovány. Avšak muže nastat situace, kdy více predikcí spadá do stejné oblasti obrázku a predikují totožný objekt. Tuto situaci mužeme pozorovat v obrázku 6.7, kde jsou zobrazeny boxy s ruznou hodnotou pravdepodobnosti, které je zobrazeno intenzitou šedé barvy.



Obrázek 6.7: Predikce boxu, které se prekrývají v dané oblasti.

Pro odfiltrování techto hodnot je použit algoritmus Non-Maximum Suppression (dále jen NMS). Algoritmus projde všechny zadané boxy a vybere pro dané oblasti pouze ty které mají nejvyšší skóre. Prekryvy boxu jsou porovnány pomocí funkce IOU (viz výše). Implementaci NMS je možné videt v algoritmu 3. Algoritmus vrací indexy bounding boxu, jež se mají zachovat.

Algoritmus 3: Non-Maximum Suppression.

Data:

box - Seznam souradnic boxu,

score - Seznam ohodnocení boxu,

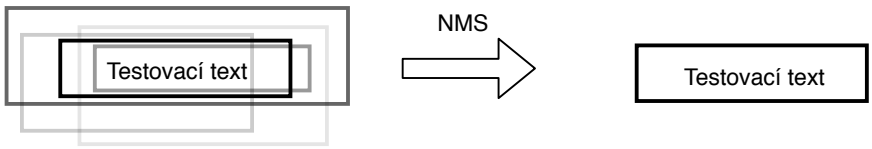
N - Celková délka techto seznamu.

threshold - Práh pro hodnotu prekryvu, nad kterou potlacujeme.

Result: *keep* - Seznam indexu boxu, které jsou zachovány.

```
1  keep ← prázdný seznam.
2  supressed ← seznam velikosti N nastavený na False
3  order ← reverse argsort(score)              // Sestupne serazené indexy score
4  for i = 0; i < N; i++ do
5      idx ← order[i]
6      if suppressed[idx] is False then
7          keep.push(idx)
8          for j = i + 1; j < N; j++ do
9              jdx ← order[j]
10             if suppressed[jdx] is False then
11                 if IOU(box[idx], box[jdx]) ≥ threshold then
12                     suppressed[jdx] ← True
```

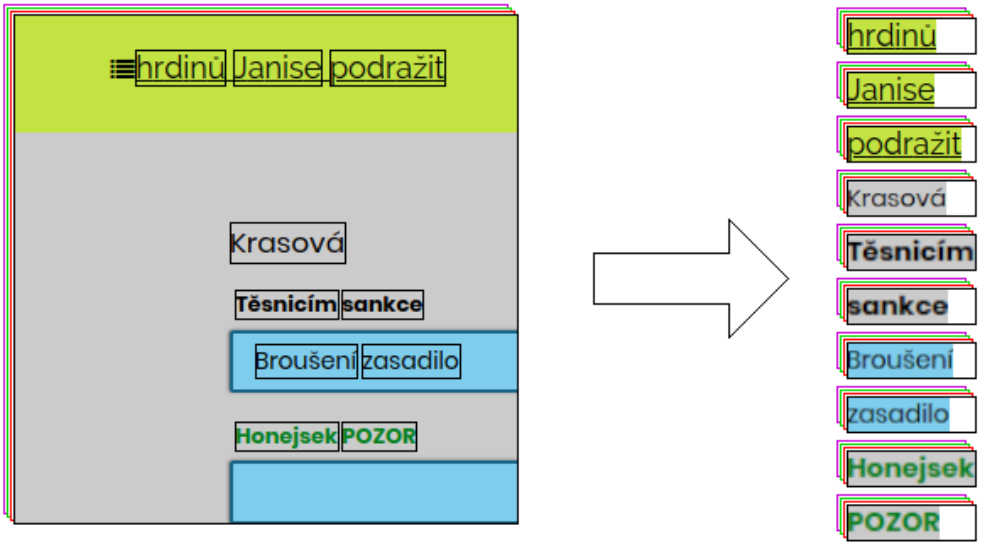
Aplikací algoritmu na boxy z predchozího obrázku mužeme pozorovat v obrázku 6.8. V obrázku je patrné, že je zachován pouze ten nejlépe ohodnocený box (nejvíce intenzivní hodnota šedi) a všechny ostatní boxy jsou algoritmem potlaceny. Díky tomu nevznikají duplikáty pro stejné oblasti.



Obrázek 6.8: Prekrývající se predikce boxu, jsou pomocí metody Non-Maximum suppression odfiltrovány.

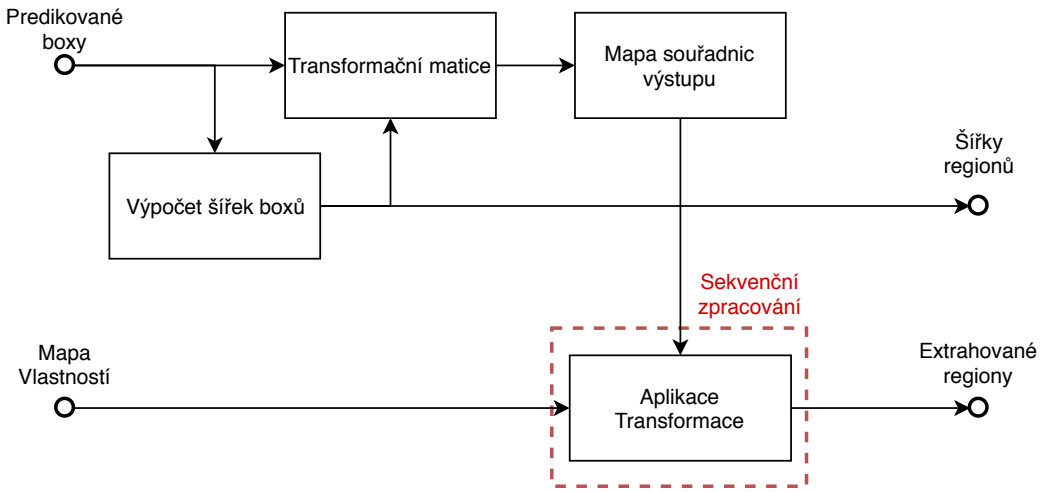
6.5 Extrakce regionu

Dalším krokem extrakce regionu získaných v predchozím kroku z mapy vlastností. Tento krok je vhodný, protože další části se budou pracovat pouze s daty jež patrí danému regionu. Tento krok mužeme rozdelit na dve části, príprava transformacní dat a aplikování techto transformací pomocí spatial transformer network [10] na zadanou mapu vlastností. Výsledek, který tento modul produkuje, mužeme pozorovat na obrázku 6.9. V levé části obrázku je mapa vlastností s vyobrazenými boxy, v pravé jsou získané regiony z této mapy. Pro lepší predstavu jsou mapa vlastností i výsledné regiony zobrazeny pomocí vstupního obrazu, ve skutecnosti se však jedná o tenzor s 256 kanály.



Obrázek 6.9: Extrakce regionu z mapy vlastností.

Texty v obraze není vhodné zmenšit do stejne rozmerné oblasti jako se tomu deje v RoI poolu metody R-CNN [4, 5], a to zejména kvuli dvema faktorum. Slova textu mají variabilní délku a kdyby se slova o ruzných délkách meli vmestnat do stejné oblasti tak dojde ke ztráte informací následná klasifikacní sít by mohla mít problémy. Tento princip je však ale vhodný pro zpracování jednotlivých písmen, ale kvuli tomu by musel být anchor boxu mnohem vetší a mapa vlastností více citlivá. Dalším faktorem, proc tento zpusob není vhodné jsou ruzné fonty. Stejná slova zobrazená s jinými fonty mohou mít úplne odlišné charakteristiky a opet muže pri zmenšení do stejné oblasti dojít ke ztráte dat. Blokový model tohoto modulu je možné sledovat na obrázku 6.10



Obrázek 6.10: Model extrakce regionu.

Rešením získání slov je ponechat rozmery všech textu a zmenšit je do oblasti s fixní výškou (v této práci se jedná výšku 8 pixelu) a ponechat šírku jako promennou. A obecne mužeme pro jeden box tuto transformaci lze zapsat jako zobrazení jež mení rozmery tenzoru:

$$T : \mathbb{R}^{C\times H\times W} \rightarrow \mathbb{R}^{C\times H'\times W'}, \tag{6.9}$$

kde C je pocet kanálu vstupního i výstupního tenzoru, H a W jsou šírka, resp. výška vstupního tenzoru, a H', W' jsou šírka, resp. výška výstupního tenzoru. H' znací stanovenou konstantní výšku a W' se na základe boxu musí dopocítat.

$$w_t = h_t \frac{x_{\max} - x_{\min}}{y_{\max} - y_{\min}}, \tag{6.10}$$

kde h_t je zvolená konstantní výška okna, $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$ znací souradnice boxu, w_t je šírka okna pro daný box. Rozmery h_t a w_t dohromady udávají velikost výstupního tenzoru pro tento box (hodnoty pro H', W'). Obecne každý z predikovaných boxu muže mít ruznou šírku, ale pro další zpracování používáme tu nejvetší šírku ze všech boxu, aby bylo možné všechny transformované regiony reprezentovat jako jediný tenzor s rozmery $\mathbb{R}^{N\times C\times H'\times W'}$ (N udává pocet predikovaných boxu). Regiony, jež jsou kratší jsou do dané šírky doplneny prázdnými hodnotami.

Data ze vstupní mapy jsou pro každý box transformována pomocí afinní transformacní matice, která se skládá z translace boxu do stredu prostoru a zmenou merítka:

$$M = s \begin{pmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & \frac{1}{s} \end{pmatrix} \tag{6.11}$$

Na základe této transformacní matice lze urcit pro souradnice vstupního obrazu souradnice ve výstupním:

$$\begin{pmatrix} x_{ij}^s \\ y_{ij}^s \\ 1 \end{pmatrix} = M \cdot \begin{pmatrix} x_{ij}^t \\ y_{ij}^t \\ 1 \end{pmatrix}, \tag{6.12}$$

kde (x_s, y_s) jsou souradnice ze zdrojového tenzoru, (x_t, y_t) jsou souradnice v cílovém tenzoru, $1 \leq i \leq H'$ a $1 \leq j \leq W'$. Toto mapování však není príliš praktické, protože pri urcitých

parametrech mohou vznikat ve výstupním tenzoru díry. Tím se myslí bod, do kterého nebyly transformovány hodnoty ze žádného bodu zdrojového tenzoru. Efektivnejší mapování je pro každé výstupní souradnice urcit pozice bodu ze zdrojového tenzoru. Tím je zaruceno, že díry vzniknout nemohou. Souradnice, které jsou mapované do necelocíselných souradnic, jsou ze zdrojového obrazu vzorkovány pomocí interpolacní metody (jmenovite bilineární interpolace). Tuto praktiku nazýváme v pocítacové grafice jako inverzní mapování. Získaná transformace je aplikována následující rovnicí

$$V_{ij}^{c} = \sum_{n}^{H} \sum_{m}^{W} U_{nm}^{c} \max\left(0, 1 - \left|x_{ij}^{s} - m\right|\right) \max\left(0, 1 - \left|y_{ij}^{s} - n\right|\right), \tag{6.13}$$

kde $1 \leq c \leq C$ je kanál vstupního i výstupního tenzoru, U_{nm}^{c} je hodnota vstupního tenzoru na kanálu c a souradnicích n, m, dále V_{ij}^{c} je hodnota výstupního tenzoru na souradnicích $1 \leq i \leq H'$ a $1 \leq j \leq W'$. Funkce max v tomto prípade realizuje bilineární interpolaci – je-li rozdíl mezi hodnotami m a x_{ij}^{s} velmi malý, pak hodnota funkce max je nenulová a vzorek z dané osy x je zohlednen, to stejné platí i pro osu y. Dohromady pokud jsou výsledky obou funkcí max nenulová je vzorek U_{mn}^{c} zahrnut ve výsledku pomerem daným dle výsledku zmínených funkcí.

Aplikování této transformace je provedeno pomocí knihovny PyTorch, která výše uvedenou rovnici realizuje. Konkrétne se jedná o funkci `grid_sample`, která na základe vstupní mrížky souradnic, jež má rozmer $\mathbb{R}^{B \times H' \times W' \times 2}$ (promenná B znázornuje pocet dávek zaráz), a zdrojového tenzoru urcí hodnoty výsledného okna. Mrížka souradnic je velikosti výstupního okna, které požadujeme a pro každou z techto souradnic je urcena dvojice souradnic (x,y) odpovídající zdrojovému obrazu, tyto hodnoty získáme pomocí inverzní matici k M uvedené výše. Souradnice v rámci této funkce jsou normalizovány do rozsahu $\langle -1; 1 \rangle$, proto je ješte potreba transformované souradnice dále normalizovat pomocí:

$$\overline{x}_{ij}^{t} = 2\frac{x_{ij}^{t}}{W} - 1, \tag{6.14}$$

$$\overline{y}_{ij}^{t} = 2\frac{y_{ij}^{t}}{H} - 1, \tag{6.15}$$

kde $\overline{x}_{ij}^{t}$ a $\overline{y}_{ij}^{t}$ jsou transformované souradnice do normalizovaného rozsahu uvedeného výše.

Jak už bylo zmíneno výše je detekce textu velmi závislá korektnosti detekce, proto pri trénování používáme pouze validní anotované boxy narozdíl klasicky používaných algoritmu pro detekci objektu. Pri použití predikovaných boxu by mohli vzniknout situace kdy predikovaný box nepokryje veškeré slovo textu, nebo box muže být výrazne vetší než daný text a jiné. Tyto situace mají pak špatný úcinek na efektivní trénování síte pro detekci textu a tomu se práve predejde použitím anotovaných boxu, protože u nich víme, že presne odpovídají textu, jež ohranicují.

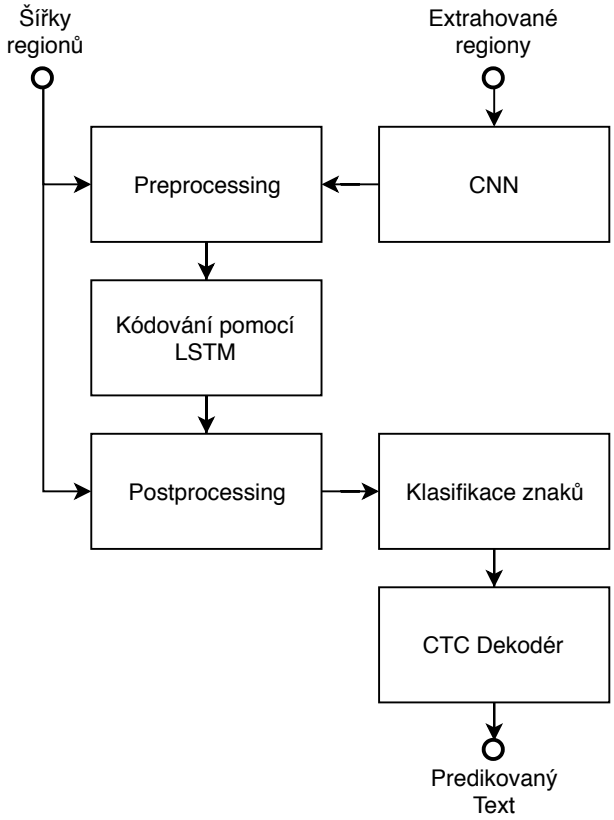
Tento modul provádí transformace mapy vlastností jednotlive po boxech. Duvodem toho je velmi velká pametová nárocnost funkce `grid_sample` pri transformaci všech boxu zaráz. Chceme-li tuto metodu aplikovat pro dva boxy paralelne je potreba duplikovat vstupní mapu vlastností. Velikost této mapy je variabilní, ale pri ucení je omezena na rozmery 200×300 s 256 kanály. Pridáme-li k tomu velikost datového typu float16 a pak dostaneme 30 MB na jeden obrázek. Pocet boxu textu v obraze se v datasetu pohybuje pres 400. Tohle je pocet kolikrát musíme rozšírit mapu vlastností, aby bylo možné transformaci pro všechny boxy zpracovat paralelne. Tím dostaneme tenzor o velikosti 12 GB. To je velikost, která se na standardní grafické karty[2] nevejde, a to nepocítáme parametry síte jako jsou

[2]Mezi standardní grafické karty nejsou pocítány high-end karty jako je NVidia Volta a podobné

váhy, pomocné tenzory atp. Pri inferenci a použití vetších obrázku muže sít dosahovat mnohem vetších velikostí. Proto jsou v modulu všechny boxy transformovány sekvencne. Avšak tento prístup je však více casove složitejší, a výsledné trénování, prípadne inference je velmi pomalá.

6.6 Rozpoznání textu

Posledním krokem algoritmu je rozpoznání textu ze získaných regionu z predchozího kroku. Každý tento region obsahuje cást sdílené mapy vlastností, která koresponduje danému boxu slova, jež chceme rozpoznat. Zároven s regiony jsou vyžadovány i šírky techto regionu, protože jsou regiony zarovnány do šírky nejdelšího. Zpracování techto regionu probíhá v nekolika sekcích nejprve je na regiony aplikována sekvencní konvolucní sít (podobne jako pri extrakci mapy vlastností, viz 6.3). Dále je výsledek konvolucní síte transformován na sekvenci, která je pomocí obousmerné LSTM zakódována. Výstupy smeru jsou zkombinovány a pro jednotlivé úseky sekvence je provedena klasifikace do zvolené množiny znaku. Sekvence hodnot klasifikací je dekódována pomocí CTC a výsledkem je predikovaný text. Schéma tohoto modulu mužeme pozorovat na obrázku 6.11.



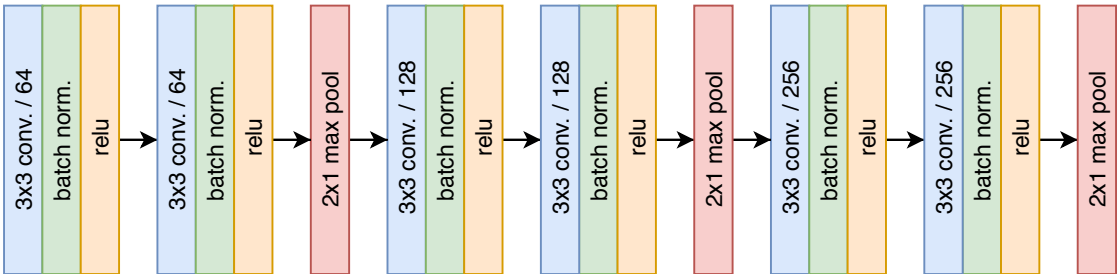
Obrázek 6.11: Model rozpoznání textu

6.6.1 Redukce vlastností regionu

Z prechozích kroku jsou získány regiony s variabilní šírkou, avšak pro další zpracování je potreba tento region zredukovat do 1dimenzionální sekvence vektoru, která je potreba pro použití rekurentní síte. Chceme však zachovat šírku regionu. Proto na výsledné regiony je

aplikována konvolucní sít podobná VGG, která zredukuje rozmery techto regionu a získá z nich vlastnosti vyšší úrovne.

Tato konvolucní sít se skládá ze trí vrstev, které sdílí stejnou velikost kanálu. A to konkrétne 64 kanálu v první vrstve, 128 ve druhé a 256 ve tretí. Každá z techto vrstev se skládá ze dvou operací konvoluce s jádrem velikosti 3. Za každou operací konvoluce následuje operace normalizace dávky jejíž výsledek je aktivovaný operací ReLU. Na konci každé z techto trí vrstev následuje operace max-pool. Avšak oproti síti VGG, je operace max-pool nastavena na agregaci hodnot pouze ve vertikální ose. Tím je zaruceno, že po provedení konvolucní síte se výška regionu zredukuje a puvodní šírka zustane zachovaná. Výsledkem této vrstvy je tenzor $\mathbb{R}^{C\times W'}$, jež je dále pomocí LSTM zakódován. Model této konvolucní síte je možné pozorovat na obrázku 6.12.



Obrázek 6.12: Model CNN v rámci modulu rozpoznání textu.

6.6.2 Použití rekurentní síte

Tenzor získaný pomocí konvolucní síte $L \in \mathbb{R}^{C\times W'}$ mužeme permutovat do sekvence ve tvaru $l_1, l_2, \ldots l_{W'} \in \mathbb{R}^C$. Podíváme-li se na tuto sekvenci jako na casovou radu, pak je možné chápat jednotlivé hodnoty v casech této rady jako vývoj spektra, které dohromady tvorí obraz daného slova. Tento prístup je používán napríklad i pri rozpoznání zvuku, avšak rozpoznaní slov z obrazu je daleko složitejší, protože muže obsahovat daleko více promenných. Mezi ne patrí napríklad nastavení a parametry fontu jimž je text vyobrazen, kontrasty barev pozadí vuci textu atd. Dále se na získané sekvence aplikuje rekurentní neuronová sít, která má za úkol naucit se sekvenci replikovat. A to zpusobem že získá ze sekvence širší kontext struktury, jakým zpusobem jsou jednotlivé casové kroky na sobe závislé. K temto úcelum byla v práci využita obousmerná rekurentní sít LSTM, o jejíž principu i výhodách bylo psáno v kapitole 2.4.6.

Použitá rekurentní sít LSTM je implementována za pomocí knihovny PyTorch a vyžaduje na vstupu tenzor rozmeru uvedených výše. Jelikož jsou všechny textové boxy variabilní, ale jsou zarovnané do stejné velikosti, nedává smysl urcovat hodnoty sekvencí pro casové body, které nenáleží vstupu. Rešením je zabalit výše uvedenou sekvenci pomocí metody `pack_padded_sequence`. Tato metoda vyžaduje na vstupu serazené všechny boxy sestupne podle jejich délky. Proto je potreba predpripravit pole indexu, které odpovídají požadované serazené posloupnosti. Pole indexu je vhodné díky tomu, že se data nachází dvou ruzných tenzorech a je možné to využít na serazení obou zaráz. Takové serazené pole dosáhneme pomocí metody `arg_sort`. Zabalený tenzor poté predáme obousmerné vrstve LSTM, která je nastavená na 256 výstupních kanálu. Tím získáme dve sekvence: jedna kóduje vstupní sekvenci od zacátku po konec a druhá od konce po zacátek. Každá se zakódovaných sekvencí obsahuje 256 kanálu pro každý bod. Výstupem je ale zabalená sekvence, kterou musíme opet na základe délek rozbalit. K tomu je použita metoda `pad_packed_sequence`, která z ruzne

velkých sekvencí vytvorí zpátky tenzor. Obe sekvence (dopredná i zpetná) jsou následne zkombinovány a výsledek je dále klasifikován.

6.6.3 Klasifikace a dekódování

Z predchozí casti dostaneme sekvence, kde každý bod obsahuje 256 kanálu. Na každý z techto bodu je aplikována plne propojená vrstva, jejíž pocet výstupních elementu je stejný jako velikost množiny znaku, které klasifikujeme (v rámci této práce se jedná o 131 znaku). Množina znaku je definována podle datové sady a obsahuje dva speciální znaky: prázdný znak a neznámý znak. Po této vrstve je aplikována vrstva Softmax, která prevede hodnoty do rozdelení míry pravdepodobností, pro jednotlivé znaky. Hodnotu znaku pro daný casový bod sekvence urcíme jako znak s nejvetší mírou pravdepodobnosti.

Sekvenci ješte musíme dekódovat, protože šírka predikovaného boxu, pomocí nehož byl region pro rozpoznání extrahován, nemusí odpovídat i šírce výsledného textu. K tomu je použit Connectionist-Temporal-Classification (o CTC více v kapitole **??**). Tento dekodér slouží ke klasifikaci sekvencí tak, aby nebyla závislá v case. Bez CTC by musel být dataset pripraven tak, aby bylo presne jasné, v jaký casový okamžik bude sít produkovat požadovaný výsledek. Použitím CTC umožníme sít se naucit klasifikovat sekvenci, tak aby na konci vyprodukovala sekvenci, která je požadována na výstupu. Dekódování sekvence je implementováno pomocí algoritmu 4. Algoritmus odfiltruje duplicitní znaky a získáme cistou sekvenci, která poslouží jako výstup. Aby tento výstup byl pro nás užitecný je ješte treba zpetne priradit Unicode hodnoty znakum na základe indexu trídy.

Algoritmus 4: Dekódování CTC sekvence.

Data:

input_text - Seznam nejpravdepodobnejšími znaky.

N - Délka tohoto seznamu.

Result: *decoded_text* - Výsledný seznam znaku.

```
1  decoded_text ← prázdný seznam.
2  last_char ← blank
3  for i = 0; i < N; i++ do
4      in_char ← input_text[i]
5      if in_char is blank then
6          last_char ← in_char
7          continue
8      if last_char is blank then
9          decoded_text.push(in_char)
10     else if in_char is not last_char then
11         decoded_text.push(in_char)
12     last_char ← in_char
```

6.7 Problémy, složitost

Pri rešení tohoto projektu vzniklo mnoho komplikací, kde jedním z nejvíce castých problému byl nedostatek pameti. Mnoho úseku kódu díky tomu muselo být z paralelního zpracování, které je rychlé ale velmi pametove nárocné, sníženo na sekvencní zpracování. Jeden z problému už byl zmínen pri v kapitole extrahování regionu. Aby bylo možné provést operaci samplování paralelne bylo by potreba velmi velké pameti.

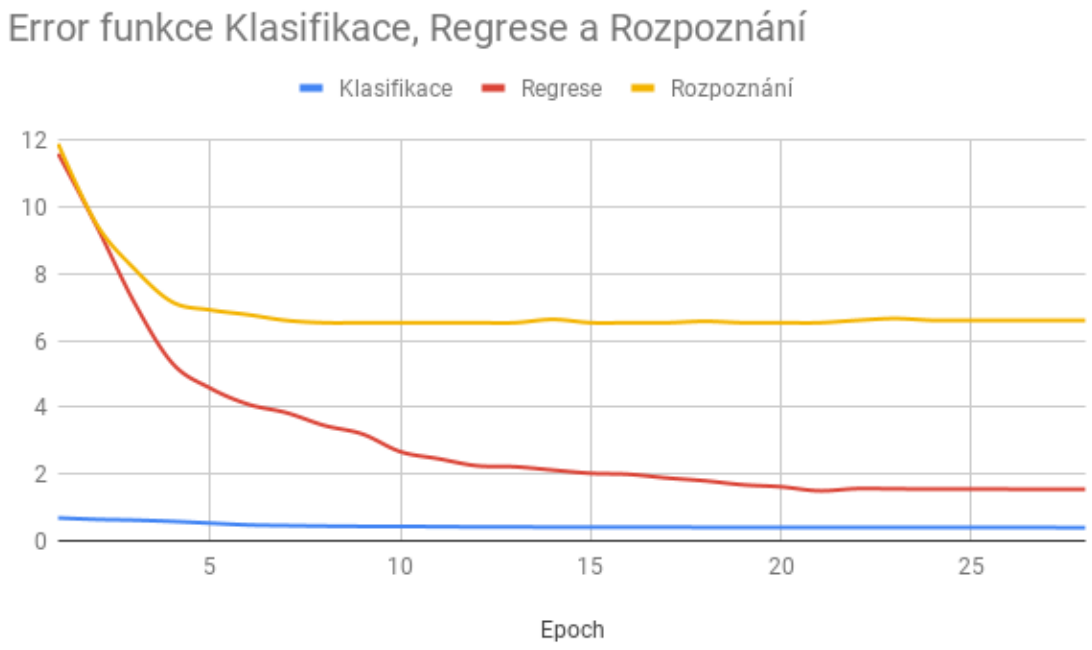
Dalším problémem, který se také projevil vysokou pametovou nárocností systému byla anomálie v datasetu. Kde jeden obrázek datasetu obsahoval slovo, které se ovšem zalomilo do následujícího rádku. Díky tomu vznikl bounding box pres celý text. Tento problém byl závažný ze dvou duvodu, prvne anotovaný box neodpovídá skutecnosti a taková datová sada by mohla by se mohla podílet na špatné konvergenci síte. Avšak tento problém se nijak výrazne neprojevil, dokud byla trénována pouze první polovina síte (po predikci regionu). Jakmile se však zacal trénovat celý systém naráz, nastala situace, kdy tento box, který zabíral více než polovinu šírky celého obrazu, zpusoboval nadbytecné alokace pameti. Duvodem alokací je, protože všechny boxy, které extrahujeme z obrazu, jsou zarovnány na stejnou šírku v pameti. Pri konvolucní síti uvnitr cásti s predikci textu nastal tento problém a vždy skoncil pádem z duvodu nedostatku pameti.

Rešení, nebo vubec zjištení duvodu tohoto problému zabralo hodne casu, protože pri ucení se problém projevil pouze u této jediné položky a dataset je automaticky pro každou epochu náhodne zamíchán. Tato chyba se díky tomu projevovala velmi zrídka, proto jsem celou dobu považoval, že se chyba nachází uvnitr implementovaného systému. Casto se tato chyba objevila až pozdeji v rámci epochy ucení, která trvá približne až 5 hodin. Nejprve jsem myslel, že se data do celé síte nevejdou, tak jednoduché a automatické rešení je snížit velikost datové sady škálováním. Ukázalo se však, že nesejde na velikosti vstupního obrázku, protože pri zmene velikosti obrázku jsou stále zachovány pomery mezi stranami boxu. Po nekolika neúspešných uceních síte jsem zacal zkoušet další možné rešení. Nekteré operace, na kterých aplikace padla, vytvárely nové tenzory. V cásti rozpoznání síte, konkrétne pri operaci `pad_packed_sequence` byl castý problém s nedostatkem pameti, který vedl k pádu aplikace. V danou chvíli jsem se snažil snížit využívanou pamet grafickou kartou na minimum a nekteré operace, které nevyžadovali paralelní zpracování, byly presunuty a pocítány sekvencní na procesoru. Avšak ani toto rešení problém nevyrešilo tak jsem myslel, že bud jazyk Python nebo knihovna PyTorch neuvolnují dostatek pameti pro další alokace, anebo jej uvolnují na špatném míste. Zkusil omezit nárocnost tohoto systému tak, že veškerou nepotrebnou pamet rucne uvolním smazáním reference na danou promennou a následného uvolnení pameti prímým voláním garbage collectoru. Stejne jako pri ostatních rešeních se toto ukázalo být neefektivní a problém stále setrvával. Po vycerpání všech možností síte jsem nakonec obrátil pozornost i k datasetu. Až v tento okamžik jsem na anomálii narazil. Problém byl vyrešen vygenerováním nového datasetu a rucním procházením všech vygenerovaných souboru, zda se v nich nenachází také tato anomálie.

Kapitola 7

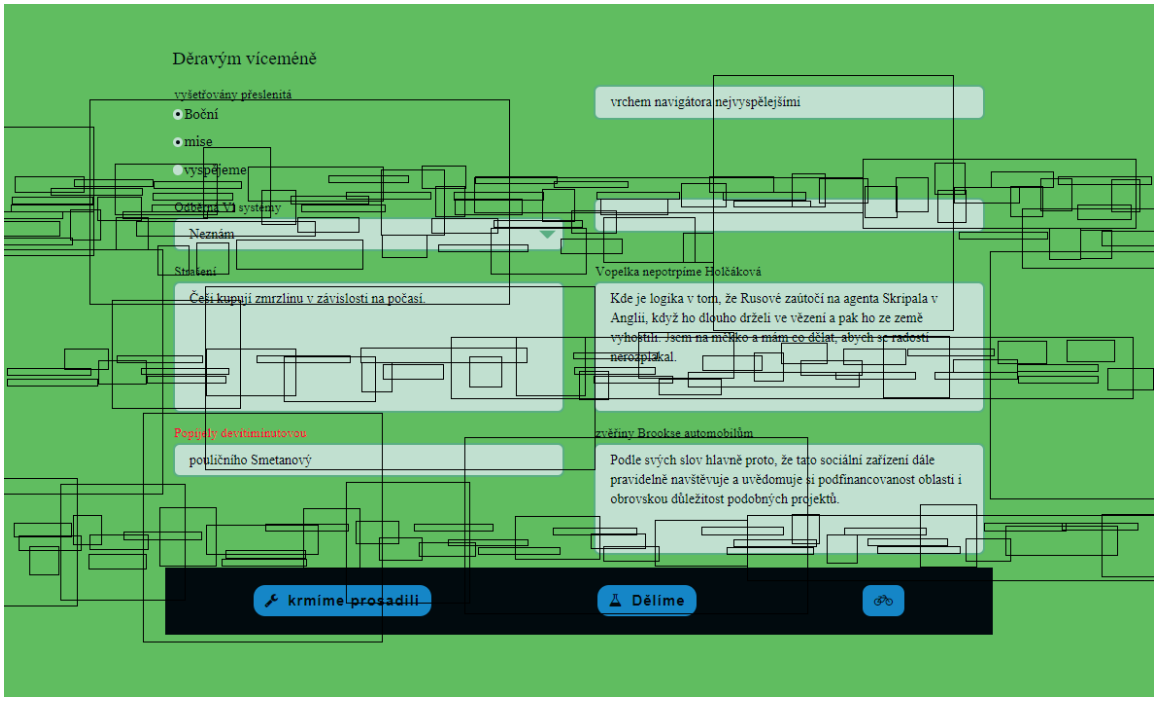
Experimenty

Model síte, který byl predstaven v predchozí kapitole, byl ucen na stroji se dvema grafickými kartami NVidia GTX-1080. Jako ucební vzor byl vygenerován pomocí nástroje „gui-generator“ dataset s obrázky rozmeru 1200x800 obsahující více než 500 obrázku. Tento model je však velmi výpocetne nárocný a délka jedné iteraci datové sady pohybovala až k 10 minutám. Model se ucil déle než týden na daném strojit a provedl 28 epoch trénování v grafu na obrázku 7.1 je možné videt pohyb chybových funkcí. Z grafu je patrné, že sít do epochy 20 konvergovala a poté se optimalizace síte zastavila a poté kmitala okolo bodu.

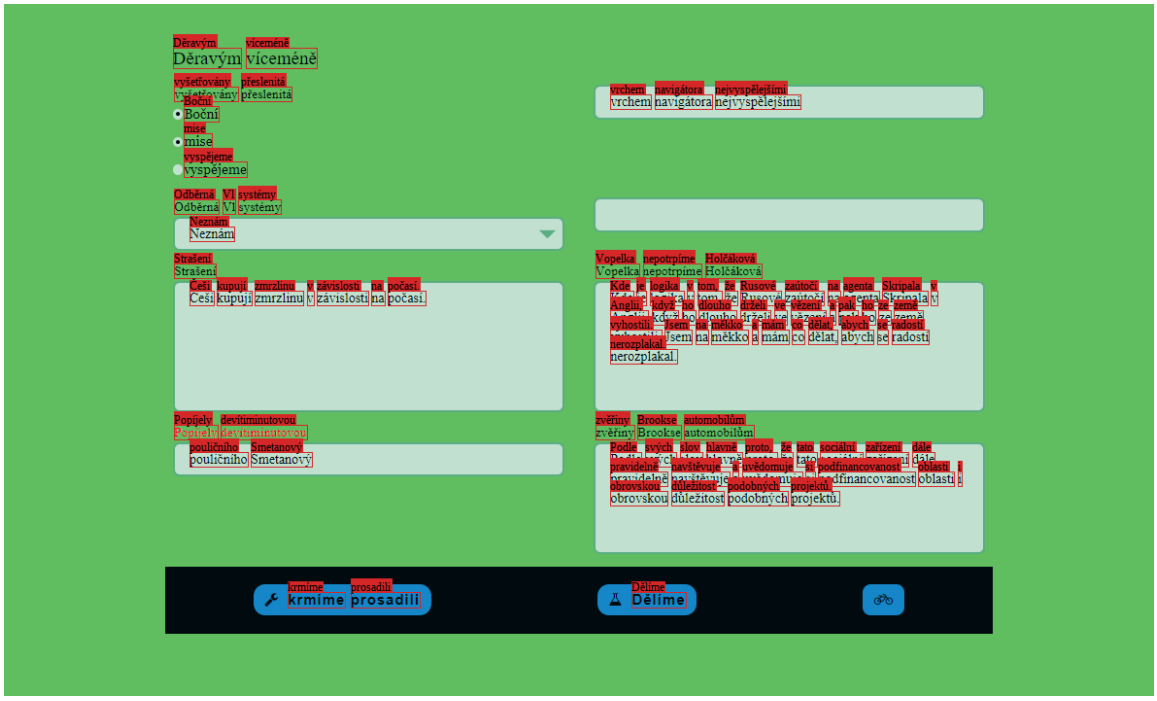


Obrázek 7.1: Znázornení prubehu loss funkcí pri trénování.

Avšak týden ucení tomuto modelu ani zdaleka nestací a jak mužeme pozorovat na výsledcích tohoto modelu v následujícím obrázku 7.2. V obrázku jsou zobrazeny bounding boxy, které systém predikuje nad skóre 95%. Jak je možné videt ze vzorových dat v obrázku 7.3



Obrázek 7.2: Ukázka výstupu modelu po 28 epochách trénování



Obrázek 7.3: Stejný obrázek v anotované sade.

Problémy tohoto systému tkví pravdepodobne v nedostatku casu na ucení systému. Možným dalším dusledkem jsou špatne zvolené pocátecní váhy systému, prípadne rozlišení bounding boxu. Kvuli vyprodukovaným výsledkum implementovaného systému, nebyl systém porovnán vuci ostatním metodám.

Na vyhodnocení jednotlivých služeb rozpoznání textu byla implementována metrika merení ZoneAltCnt (kapitola 3.2.2). Pro merení byla, pomocí nástroje gui-generátor (popsaného v kapitole 4), vytvorena datová sada cítající 1950 anotovaných obrázku. Hlavními metrikami merení byly metrika znaku, slov a úplná metrika. V následující tabulce (7.1) mužeme pozorovat, jak si jednotlivé služby vedly.

Tabulka 7.1: Vyhodnocení OCR

	Metrika znaku		Metrika slov		Úplná metrika	
OCR	Citlivost	Presnost	Citlivost	Presnost	Citlivost	Presnost
Microsoft Azure	83.21%	87.00%	79.50%	80.35%	79.38%	80.43%
Google Cloud Vision	87.36%	86.72%	85.89%	73.26%	83.19%	70.96%

Nejduležitejším merením je citlivost znaku, protože chceme detekovat všechny texty datové sady, a toho dosáhneme pri citlivosti 100%. Mužeme pozorovat, že obe služby správne rozpoznaly více než 80% znaku datové sady. Všechny ostatní znaky jsou bud chybne klasifikovány, nebo je algoritmy nedetekovaly vubec. Co se týce presnosti, vidíme že u obou algoritmu približne 87% korektních predikcí. Ostatní mohou být neshody znaku, ci predikce mimo textové oblasti. Podíváme-li se na metriku slov, vidíme, že služba Microsoft Azure OCR (dále jen Azure OCR) správne rozpoznala 79,5% anotovaných slov datové sady. Ale také mužeme pozorovat podle úplné metriky, že míra rozpoznání slov i se segmentací je 79,38%. Rozdíl techto dvou hodnot znací, že segmentace slov služby Azure OCR je velmi podobná té, která je použita v datové sade.

Služba Google Cloud Vision (dále jen Google Vision) má výrazne vyšší míru rozeznání slov vuci testované datové sade oproti službe Azure OCR. To naznacuje že Google Vision „vidí“ více slov datové sady, což mužeme pozorovat i na citlivostí znaku. Ovšem rozdíl mezi úplnou metrikou a metrikou slov naznacuje, že Google Vision využívá jinou metodu segmentace obrazu, než je použita v datové sade. Tedy pokud by se segmentace zmenila na tu, jež je použita pro datovou sadu, pak by se výsledek úplné metriky zlepšil o 2.6%.

Kapitola 8

Záver

V rámci práce byly nastudovány metody zpracování obrazu pomocí neuronových sítí vcetne zamerení na metody hlubokého ucení. Byly prozkoumány metody rozpoznání textu z obrazu, které využívají práve neuronové síte. Aplikace gui-generátor byla rozšírena o širší možnosti generování ruzných grafických webových rozhraní a posloužila pro generování datové sady. Byla vytvorená datová sada obsahující približne 1900 obrázku uživatelských rozhraní vcetne jejich anotací. Ty jsou urceny k validaci výsledku a poslouží i jako datová sada pro ucení síte. Byl implementován program pro vyhodnocení kvality algoritmu OCR na základe zvolené metriky. Pomocí tohoto programu a výše vytvorené datové sady byly jednotlivé služby ohodnoceny definovanou metrikou rozpoznání textu a jejich výsledky dále prozkoumány.

Navržená sít byla implementována, a natrénována na vygenerované datové sade. Avšak nepodarilo se dosáhnout trénováním takové konfigurace, kdy by sít dokázala predikovat dobré výsledky. Návrh i implementace této síte byla velmi velká výzva. Výsledný model kombinuje dohromady mnoho principu novodobých neuronových sítí. Mezi ne patrí vytažení vlastností z obrazu, predikce bounding boxu ze získaných vlastností, extrakce vlastností patrící konkrétnímu boxu i rozpoznání textu pomocí rekurentní síte. Výsledek natrénovaného modelu není uspokojující kvuli problémum, které se objevili pri implementaci a opozdili trénování modelu.

Dalším možným vývojem systému je alespon mesícní trénovaní síte. Nebo možným urychlením je implementace funkcí jako je napríklad NMS pomocí programovacího nástroje CUDA.

Literatura

[1] Arabnia, H. R.; Deligiannidis, L.; Tinetti, F. G. (editoři): *Proceedings of the 2018 International Conference on Image Processing, Computer Vision, & Pattern Recognition*, CSREA Press, Červenec 2018, ISBN 1-60132-485-5.

[2] Chen, X.; Yuille, A. L.: Detecting and reading text in natural scenes. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, ročník 2, June 2004, ISSN 1063-6919, s. II–II.

[3] Deng, J.; Dong, W.; Socher, R.; aj.: ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] Girshick, R. B.: Fast R-CNN. *CoRR*, ročník abs/1504.08083, 2015, 1504.08083.
URL http://arxiv.org/abs/1504.08083

[5] Girshick, R. B.; Donahue, J.; Darrell, T.; aj.: Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, ročník abs/1311.2524, 2013, 1311.2524.
URL http://arxiv.org/abs/1311.2524

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

[7] Graves, A.; Fernández, S.; Gomez, F.; aj.: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, New York, NY, USA: ACM, 2006, ISBN 1-59593-383-2, s. 369–376.

[8] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, ISSN 1063-6919, s. 770–778.

[9] Hochreiter, S.; Schmidhuber, J.: Long Short-Term Memory. *Neural Comput.*, ročník 9, č. 8, Listopad 1997: s. 1735–1780, ISSN 0899-7667.

[10] Jaderberg, M.; Simonyan, K.; Zisserman, A.; aj.: Spatial Transformer Networks. *CoRR*, ročník abs/1506.02025, 2015, 1506.02025.
URL http://arxiv.org/abs/1506.02025

[11] Jain, L. C.; Medsker, L. R.: *Recurrent Neural Networks: Design and Applications*. Boca Raton, FL, USA: CRC Press, Inc., první vydání, 1999, ISBN 0849371813.

[12] Karpinski, R.; Lohani, D.; Belaid, A.: Metrics for Complete Evaluation of OCR Performance. In Arabnia aj. [1], s. 23–29.

[13] Li, H.; Wang, P.; Shen, C.: Towards End-to-End Text Spotting with Convolutional Recurrent Neural Networks. 10 2017, s. 5248–5256.

[14] Lin, T.-Y.; Dollar, P.; Girshick, R.; aj.: Feature Pyramid Networks for Object Detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[15] Liu, X.; Liang, D.; Yan, S.; aj.: FOTS: Fast Oriented Text Spotting With a Unified Network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[16] Neubeck, A.; Gool, L. V.: Efficient Non-Maximum Suppression. In *18th International Conference on Pattern Recognition (ICPR'06)*, ročník 3, Aug 2006, ISSN 1051-4651, s. 850–855.

[17] Pan, S. J.; Yang, Q.: A Survey on Transfer Learning. *IEEE Trans. on Knowl. and Data Eng.*, ročník 22, č. 10, Říjen 2010: s. 1345–1359, ISSN 1041-4347, doi:10.1109/TKDE.2009.191.
URL http://dx.doi.org/10.1109/TKDE.2009.191

[18] Rezatofighi, S. H.; Tsoi, N.; Gwak, J.; aj.: Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression. *CoRR*, ročník abs/1902.09630, 2019, 1902.09630.
URL http://arxiv.org/abs/1902.09630

[19] Skovajsová, L.: Long short-term memory description and its application in text processing. In *2017 Communication and Information Technologies (KIT)*, Oct 2017, s. 1–4.

[20] You, Y.; Hseu, J.; Ying, C.; aj.: Large-Batch Training for LSTM and Beyond. *CoRR*, ročník abs/1901.08256, 2019, 1901.08256.
URL http://arxiv.org/abs/1901.08256

[21] Zhong, Y.; Wang, J.; Peng, J.; aj.: Anchor Box Optimization for Object Detection. *CoRR*, ročník abs/1812.00469, 2018, 1812.00469.
URL http://arxiv.org/abs/1812.00469

[22] Zhou, X.; Yao, C.; Wen, H.; aj.: EAST: An Efficient and Accurate Scene Text Detector. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, ISSN 1063-6919, s. 2642–2651.

Příloha A

Návod k použití

Projektu lze vytvořit virtuální prostředí python, kde všechny závislosti projektu jsou staženy pomocí příkazu: `pip install -r requierements.txt` Další možností pro vytvoření prostředí je použití poskytnutého docker konfiguračního souboru pro nastavení prostředí pro nvidia-docker.

Dále je možné spustit program pomocí příkazů:

Trénování

`$ python3 train.py --config /cesta/ke/configu` Tento příkaz spustí trénování modelu na základě informací uvedených v konfiguračním souboru.

Inference

`$ python3 inference.py --config /cesta/ke/configu -o /cesta/do/výstupní/složky /cesta/k/obrazku` Tento příkaz provede inferenci natrénovaného modelu nad poskytnutým obrázkem.