



**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**

AUTHOR

**Bc. PAVEL HAMERNÍK**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**Ing. TOMÁŠ LYSEK**

BRNO 2019

## Zadání diplomové práce



22173

Student: **Hamerník Pavel, Bc.**  
Program: Informační technologie    Obor: Počítačová grafika a multimedia  
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 navrhnete 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, recurrent 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

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Neuronové síte</b>	<b>4</b>
2.1	Architektury sítí . . . . .	4
2.2	Kódování . . . . .	6
2.3	Aktivací funkce . . . . .	7
2.4	Vrstvy neuronových sítí . . . . .	10
2.5	Učení sítí . . . . .	14
<b>3</b>	<b>Technologie zpracování textu</b>	<b>17</b>
3.1	Rozpoznání textu . . . . .	17
3.2	Metriky porovnání . . . . .	17
3.3	Existující řešení . . . . .	19
<b>4</b>	<b>Datová sada</b>	<b>21</b>
4.1	Texty . . . . .	22
4.2	Styly . . . . .	22
4.3	Generování testovací datové sady . . . . .	22
<b>5</b>	<b>Návrh řešení</b>	<b>24</b>
5.1	Detekce textu . . . . .	24
5.2	Prostorová transformace . . . . .	25
5.3	Rozpoznání textu . . . . .	25
<b>6</b>	<b>Implementace</b>	<b>26</b>
6.1	Použité technologie . . . . .	26
6.2	Rozdělení systému . . . . .	27
6.3	Residuální síť . . . . .	28
6.4	Predikce regionu . . . . .	30
6.5	Extrakce regionu . . . . .	35
6.6	Rozpoznání textu . . . . .	38
6.7	Problémy, složitost . . . . .	41
<b>7</b>	<b>Experimenty</b>	<b>42</b>
<b>8</b>	<b>Záver</b>	<b>45</b>
	<b>Literatura</b>	<b>46</b>



# Kapitola 1

## Úvod

Optické rozpoznání znaku (Optical Character Recognition dále jen OCR) je software který dokáže převést tištěný text a obrázky do digitalizované podoby se kterou může být manipulováno počítačem. Na rozdíl od lidského mozku, který dokáže velmi jednoduše rozpoznat text z obrázku, algoritmy dostatečně inteligentní, aby dokázaly vnímat informaci dostupnou z obrazu. Proto bylo vloženo velké úsilí na vytvořit software, který se snaží transformovat obraz dokumentu do podoby, která je srozumitelná pro počítač. OCR je velmi rozsáhlý problém kvůli velkému množství jazyku, stylu a písem, kterými můžeme vyjádřit text. Techniky z různých oborů výpočetní techniky se využívají pro řešení různých 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 případně obecnou automatizaci pouze na základe obrazových vlastností. V kapitole 2 si představíme technologie neuronových sítí. V kapitole 3 jsou uvedeny technologie zpracování textu, které zahrnují základní princip algoritmu rozpoznání textu a metriky pro porovnání kvality výsledku těchto algoritmu. Dále v kapitole 4 je popsán nástroj „gui-generator“ a jeho rozšíření, které bylo využito pro vygenerování datové sady určené k trénování i vyhodnocení algoritmu. Kapitola 5 obsahuje návrh neuronového modelu, pomocí něhož 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

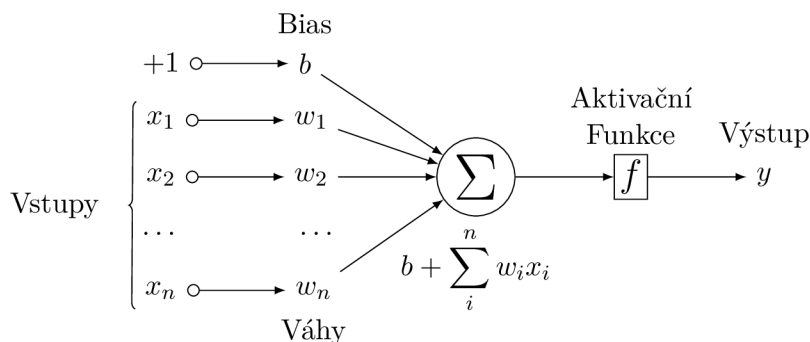
### 2.1 Architektury síte

Neuronová síte je výpocetní model používaný v umelé inteligenci. Vzorem neuronových síte 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 síte nabyly na velké popularite rešit s pomocí nich složitější úkoly a pracovat s naucenými sítemi jako s tzv. „cernou skřínkou“ (anglicky black box). To znamená, že ocekáváme od síte nejakou funkcnost, ale jak jí dosáhne, nezjistíme.

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

#### 2.1.1 Klasické neuronové síte

Základním stavebním kamenem softwarových neuronových síte je neuron (dríve nazývaný perceptron). Každý neuron se skládá z tří hlavních částí. Vstupní část 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 částí je telo neuronu, které shrne všechny hodnoty vstupu vctne 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 můžete prohlédnout na obrázku 2.1.

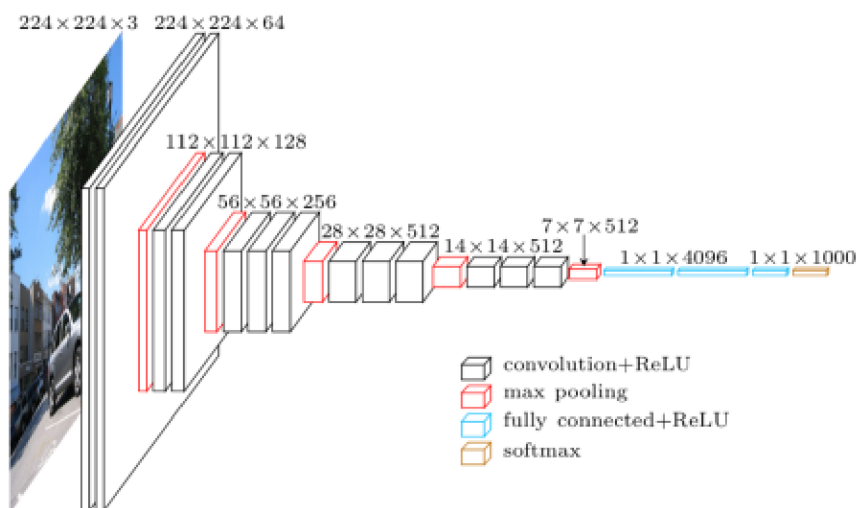


Obrázek 2.1: Výpocetní model neuronu.



## 2.1.2 Konvoluční neuronové síte

Konvoluční síte jsou speciálním druhem vícevrstevých neuronových síte. Používají vizuálních vzoru přímo z pixelu obrazu. Vychází z principu neocognitronové síte, která byla poprvé predstavena v 80. letech K. Fukushima. 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 konvoluční síte můžeme pozorovat na obrázku 2.2 a je možné pozorovat, že se skládá z mnoha vrstev. Dalším charakteristickým znakem konvolučních síte je získávání vlastností obrazu z recepčních polí, sdílení vah v rámci konvoluční vrstvy a prostorové vzorkování. [6]

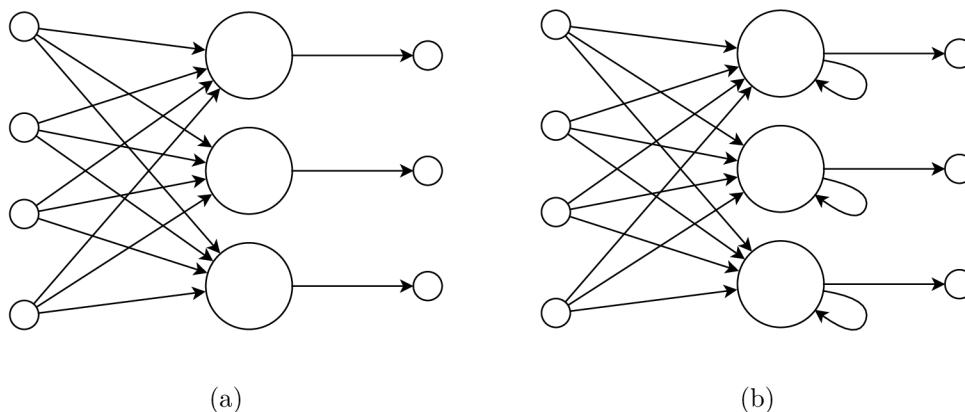


Obrázek 2.2: Ilustrace konvoluční neuronové síte. Zdroj <https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/>

## 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 kvůli vysoké výpočetní náročnosti. Výrazem rekurentní neuronové síte označujeme dve široké kategorie s podobnou obecnou strukturou. Jedna z nich je síte s konečnou odezvou, tento typ síte se dá vyjádřit pomocí orientovaného acyklického grafu, který je možné rozbalit a díky tomu je celou síte možné nahradit dopřednou síte. Rekurentní síte s nekonečnou odezvou lze reprezentovat orientovaným cyklickým grafem, který není možné redukovat na dopřednou síte. [11]

RNN využívají interní pamet, která jim umožňuje si zapamatovat informace z přijatých vstupu. Díky tomu dokážou být velmi přesné v predikci následujících hodnot. Z tohoto duvodu jsou preferovanou síte při práci se sekvencními daty (například časová posloupnost zvukových dat), protože ze zpracovávaných dat dokáží porozumet hlubšímu významu jejich posloupnosti oproti ostatním algoritmem.



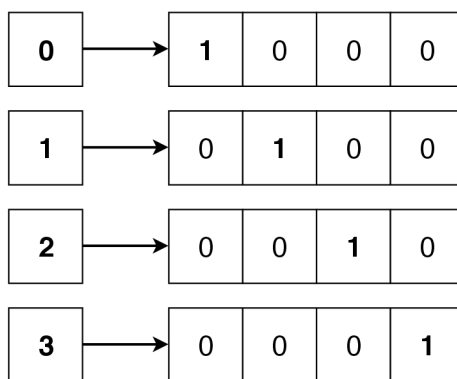
Obrázek 2.3: Struktura dopředné neuronové vrstvy (a) oproti rekurentní neuronové vrstve (b).

## 2.2 Kódování

V neuronových sítích je často potřeba informace zakódovat do reprezentace, která je pro neuronovou síť užitečnější. Díky nim se dokáží lépe učit, případně reprezentovat data stylem, který zjednoduší neuronové síti práci. Nejpoužívanější z nich jsou dále představeny.

### 2.2.1 One-hot kódování

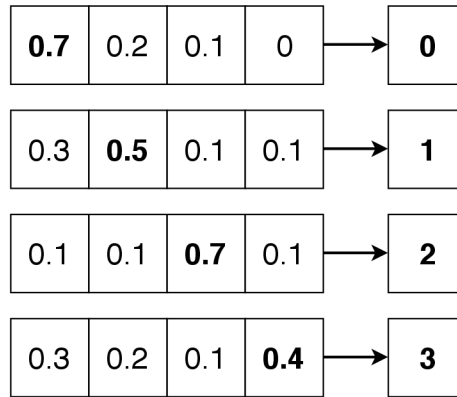
One-hot<sup>1</sup> kódování (v angličtině se také nazývá dummy variable) vyjadruje pomocí hodnoty 0 nebo 1 absenci nebo přítomnost jisté kategorie. Toto kódování je užitečné zejména pro rozdělení dat do vzájemně se vylučujících kategorií (například, pes, kocka). Na hodnoty zakódovaných kategorií se můžeme také dívat jako na míru pravděpodobnosti výskytu jednotlivých kategorií v datech. To je důvodem, proč je toto kódování hojně používáno komunitou zabývající se strojovým ucením, a to zejména v oblasti klasifikace objektu. Příklad zakódování informace do vektoru míry pravděpodobnosti můžeme pozorovat v obrázku 2.4.



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

Dekódování hodnoty z one-hot reprezentace provedeme získáním kategorie s nejvyšší mírou pravděpodobnosti. Příklad výstupu z neuronové síte a jeho dekodování můžeme pozorovat na obrázku 2.5.

<sup>1</sup>V české literatuře se také můžeme setkat s názvem kódování 1 z N.

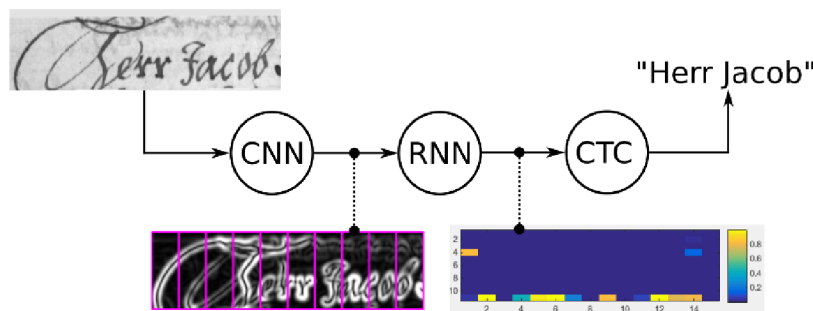


Obrázek 2.5: Ukázka dekódování one-hot výstupu pro 4 třídy.

### 2.2.2 Connectionist temporal classification

Je velmi užitečnou operací používanou v rekurentních neuronových sítích [7]. Jedná se o způsob, jak naučit neuronovou síť požadovanou sekvenci, která však může mít mnohem menší počet časových kroků. Tato operace je často používána v rámci algoritmu pro rozpoznání textu, či hlasu, kde umožňuje síti se naučit sekvenci vzoru, které jsou mnohem menší než sekvence, které na tyto vzory chceme naučit. Obsahuje-li sekvence více po sobe jdoucích časových kroků stejnou hodnotu, pak se jedná o jediný znak s touto hodnotou. CTC navíc přidává do slovníku speciální znak, nazývaný blank. Ten slouží jako oddelovac mezi znaky a umožňuje vytvoření sekvence ze stejného znaku, aniž by byl považován za duplikátní při použití blank znaku mezi těmito elementy. Níže jsou uvedeny příklady, jak je možné pomocí CTC reprezentovat některá 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 Aktivací funkce

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

smer rustu této funkce (gradient funkce). Gradientu je potřeba pro ucení neuronové sítě. Hodnota aktivace neuronu je řízena následující funkcí:

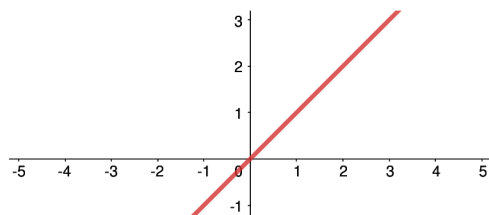
$$y = f\left(\sum_i w_i x_i + b\right) \quad (2.1)$$

, kde parametr  $w_i$  reprezentuje váhu jednotlivých vstupu  $x_i$  a parametr  $b$  (anglicky bias) se využívá k rozšíření rozhodovacího rozsahu, kterého neuronová síť může dosáhnout. Pro snadnější počítání s tímto parametrem (biasem), se rozširuje vektor vstupu o konstantní 1 (bývá označováno jako  $x_0$ ), a vektor vah o parametr  $b$  na odpovídající pozici dle přidaného vstupu ( $w_0$ ). Díky této úpravě můžeme rovnici výstupu neuronu přepsat do zjednodušené podoby za pomoci skalárního součinu:

$$y = f(\vec{w} \cdot \vec{x}) \quad (2.2)$$

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

$$f(x) = x \quad (2.3)$$

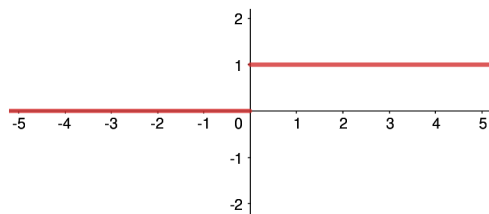


Obrázek 2.7: Aktivací funkce identita

Pro klasifikační úlohy jsou odněkdy využívány skoková funkce, sigmoida a hyperbolický tangent. Každá z těchto funkcí převádí celé vstupní spektrum reálných hodnot do úzkého intervalu hodnot.

Skoková funkce je často spojována s pojmem perceptron. Nabývá pouze dvěma hodnotami, pokud hodnota vstupu překročí určitý práh. Je vhodná pouze pro jednoduché úlohy. Není diferencovatelná, proto není vhodná pro použití ve vícevrstvých neuronových sítích.

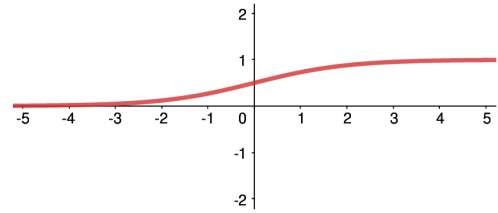
$$f(x) = \begin{cases} 1, & \text{pokud } x > 0 \\ 0, & \text{jinak} \end{cases} \quad (2.4)$$



Obrázek 2.8: Skoková aktivací funkce

Sigmoidová funkce (také logistická funkce) převádí vstupní reálné hodnoty do intervalu  $(0, 1)$ . Je monotónní a diferencovatelná je možné ji využít ve vícevrstvých neuronových sítích. Často se používá pro binární klasifikaci, kdy hodnota určuje míru pravděpodobnosti zkoumaného kritéria. Její nevýhodou je, že se ucení neuronové sítě může zaseknout.

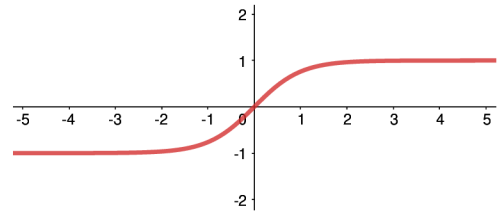
$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.5)$$



Obrázek 2.9: Sigmoidová aktivací funkce.

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

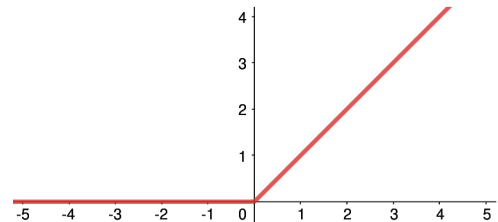
$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (2.6)$$



Obrázek 2.10: Aktivací funkce hyperbolický tangent.

Rectified linear unit (dále jen ReLU) je v současné době nejpoužívanější aktivací funkce. Tato funkce se stala velmi oblíbenou díky svým vlastnostem a snadné implementaci prahování v 0. Funkce je monotónní stejně je i její derivace. Díky tomu se u neuronových sítí zvyšuje rychlost učení.

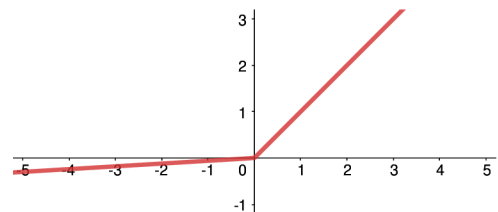
$$f(x) = \begin{cases} x, & \text{pokud } x \geq 0 \\ 0, & \text{jinak} \end{cases} \quad (2.7)$$



Obrázek 2.11: Aktivací funkce ReLU.

Problémem ReLU je že všechny záporné hodnoty jsou mapovány do nuly. Nulové hodnoty jsou v učení potlačovány, protože mají derivaci v nule. Toto má za následek snížení schopností neuronové sítě naučit se daný problém efektivně. Pokusem vyřešit tento problém je Leaky ReLU. Tato funkce mapuje záporné hodnoty pomocí lineární funkce s velmi malým sklonem. Díky tomu nejsou záporné hodnoty potlačeny a mají schopnost se učit.

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



Obrázek 2.12: Aktivací funkce Leaky ReLU.

## 2.4 Vrstvy neuronových sítí

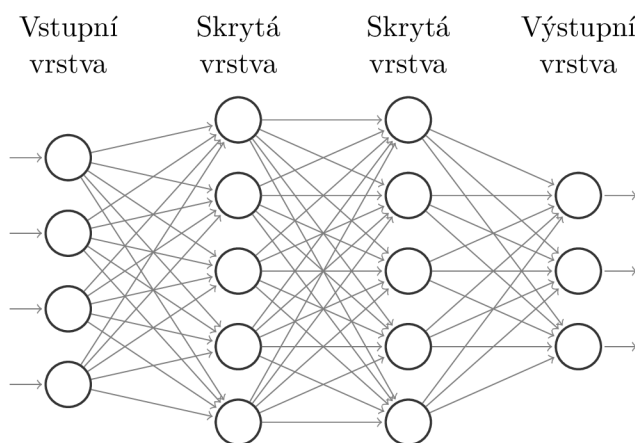
V této sekci jsou uvedeny vrstvy neuronových sítí, které jsou použity v rámci této práce.

### 2.4.1 Plně propojená vrstva

Plně propojená vrstva, v anglické literatuře se můžeme setkat s názvem fully-connected či dense layer, jak název vypovídá je vrstva  $N$  neuronů a každý z těchto neuronů je propojený se všemi neurony vstupní vrstvy. Můžeme ji definovat transformační funkcí

$$y_j = \sum_i^M w_{ij} x_i, \quad (2.9)$$

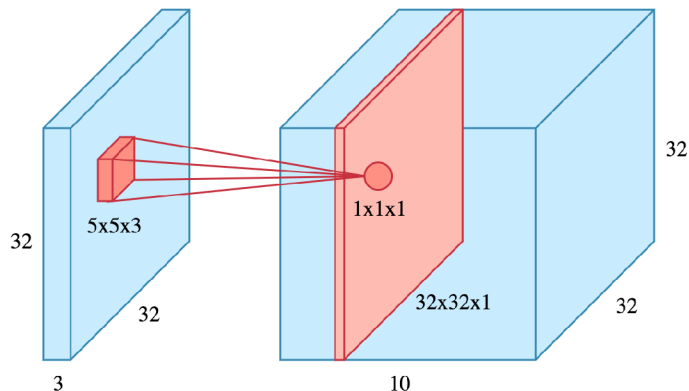
kde  $y_j$  udává hodnotu  $j$ -tého výstupního neuronu ( $1 \leq j \leq N$ ).  $x_i$  jsou vstupní hodnoty neuronů z předchozí vrstvy. Tuto vrstvu je možné pozorovat na obrázku 2.13.



Obrázek 2.13: Model sítě s plně propojenými vrstvami.

### 2.4.2 Konvoluční vrstva

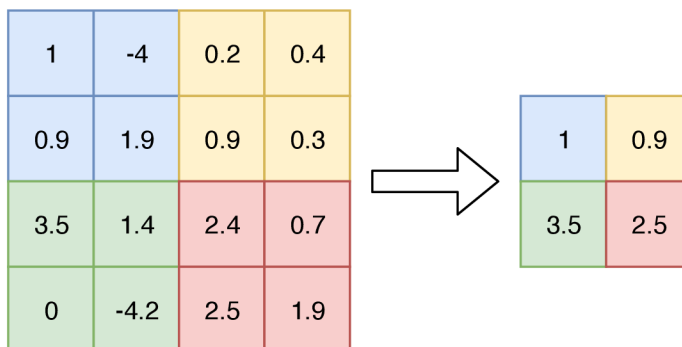
Tato vrstva je využívá konvoluční filtry, jejichž parametry jsou pro celou vrstvu sdílené. Tyto filtry pokrývají pouze malou část plochy vstupních dat, ale využívají celou jejich hloubku. Na hodnotu získanou konvolucí je aplikována aktivací funkce, která je výstupem tohoto filtru pro jeden neuron. Výstupem jednotlivých konvolučních filtrů této vrstvy je dvoudimenzionální mapa, dohromady všechny mapy tvoří objem, kde hloubkou tvoří počet aplikovaných konvolučních filtrů dané vrstvy. U těchto vrstev určujeme několik vlastností, kterými jsou počet aplikovaných konvolučních filtrů, velikost jádra konvoluce těchto filtrů (anglicky kernel size), krok posunu konvoluce (anglicky stride) - určuje rychlost posunu klouzavého okna konvolučních filtrů po vstupu. Dále můžeme u vrstvy specifikovat tzv. zero-padding. V tomto režimu jsou vstupní data doplněna o nulové hodnoty tak, aby velikost výstupu po provedení konvoluce byla stejná jako vstupní velikost (velikost před doplněním o nuly). Konvoluční vrstva je znázorněna na obrázku 2.14.



Obrázek 2.14: Konvoluční vrstva. Zdroj <https://towardsdatascience.com>

### 2.4.3 Pooling vrstva

Slouží pro zmenšení velikosti dat vstupní vrstvy. Tato vrstva se hlavně využívá mezi konvolučními vrstvami pro snížení celkového počtu parametru výsledné sítě. Podvzorkování se aplikuje na jednotlivých hloubkách vstupních dat samostatně. V rámci tohoto vzorku dat pro hloubku se podobně jako u konvoluce posouvá klouzavé okno, na které se aplikuje některá z agregacích funkcí (nejčastěji se jedná o funkci maximum nebo průměr, podle nich pak nazývány max pooling nebo average pooling). Podobně jako u konvoluční 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 Normalizační vrstva

Normalizační vrstva, v anglické literatuře uváděna pod názvem batch normalization layer, je vrstva, která provádí korekci vstupních dat. Data jsou transformována do normálního rozložení se střední hodnotou 0 a rozptylem 1. Díky tomu je zajištěno, že se rozložení hodnot dat bude pohybovat v nejbližším okolí 0. Tato vrstva si při trénování počítá statistiky o středních hodnotách a rozptylech, které jsou posléze využity při inferenci sítě. V praxi se tato vrstva používá mezi vrstvou konvoluce, případně plně propojenou vrstvou, a vrstvou s aktivací funkcí. Pro každý mini-batch se normalizace spočítá následujícími rovnicemi:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i, \quad (2.10)$$

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

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

$$y_i = \gamma \hat{x}_i + \beta, \quad (2.13)$$

kde  $\mu$  značí střední hodnotu v rámci mini-batche,  $\sigma^2$  je rozptyl,  $x_i$  jsou vstupy vrstvy,  $y_i$  jsou výstupy vrstvy a  $\gamma$ ,  $\beta$  jsou parametry, které se tato vrstva učí.

Použití této vrstvy snižuje úciněk problému zvaného internal covariance shift. Tento problém nastává při posunu vstupní distribuce ucíciho se systému. V případě 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 vnitřních 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:

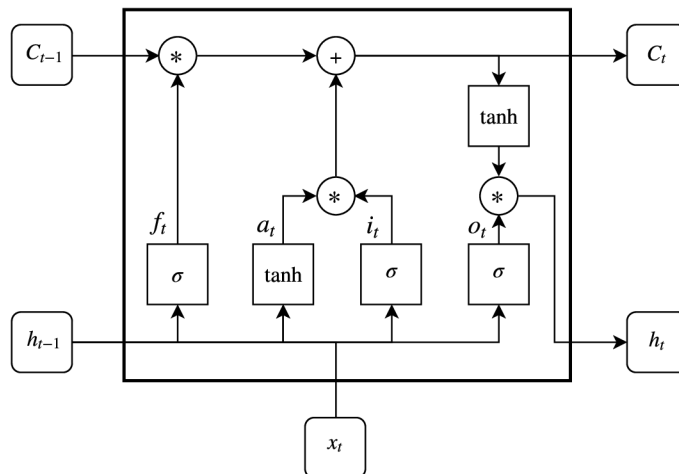
$$y_j = \frac{e^{x_j}}{\sum_{i=1}^N e^{x_i}}, \quad (2.14)$$

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 vnitřní pamet. Funkci LSTM bunky mužeme rozdelit do trech částí:





Obrázek 2.16: Schéma bunky LSTM vrstvy

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

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

**Zapamatování.** Umožňuje LSTM zapamatovat si nové vlastnosti ze současného vstupu. Využívá k tomu dvou hradel. Aktivací hradlo<sup>2</sup> získá vektor nových vlastností, které chceme přidat k současnému stavu. Dále hradlo input gate rozhoduje, které hodnoty chceme aktualizovat a s jakou mírou (podobně jako forget gate výše).

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

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

Obe tyto hradla jsou posléze zkombinovány a přidány k novému stavu. Vektory se kombinují pomocí Hadamardova součinu, který násobí vektory mezi sebou po jejich jednotlivých složkách a v rovnicích je označen symbolem  $*$ . Kombinace těchto vstupu a hodnoty forget gate dohromady udává hodnotu následujícího stavu, jak můžeme vidět níže.

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

**Výstup.** Výstup LSTM je založený na současném stavu v bunce. Nejprve pomocí output gate jsou získány hodnoty, které určí části současného stavu, jež jsou ve výstupu zachovány.

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

Dále je ze současného 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.

$$h_t = \tanh(C_t) * o_t \quad (2.20)$$

<sup>2</sup>V literatuře bývá označováno jako gate.

Ve všech předchozích rovnicích  $W_*$  značí váhy vrstvy pro jednotlivé části a vstupy,  $x_t$  je vstup v case  $t$ ,  $h_t$  je předchozí výstup LSTM v case  $t$ ,  $C_t$  je zapamatovaný stav LSTM v case  $t$ .

Při zpětné propagaci je gradient neprerušene propagován skrze stavy jednotlivých bunek. Díky takto navržené struktuře bunek netrpí LSTM problémy vanishing či exploding gradient jako obecné RNN. To je problém, který nastává při retezení funkcí, jejichž výstup je omezený v úzkém intervalu (například sigmoida). To má za následek, že se postupně hodnoty tlumí až do bodu, kdy hodnota propagované změny je tak nepatrná, že je její vliv zanedbatelný a ucení sítě uvázne. U rekurentních sítí je tento problém ještě více znatelný, protože přidávají ještě více vrstev.

## 2.5 Učení sítí

Hlavním účelem učení neuronové sítě je nastavení hodnot váhových parametru a prahu mezi perceptrony tak, aby síť náležitě reagovala na vstup. Mezi základní strategie učení patří učení s učitelem a učení bez učitele.

Učení s učitelem využívá znalosti vstupu i správného výstupu. Nejprve pro vstup síť vygeneruje výstup, který porovná se správným výstupem, a poté upraví váhové parametry tak, aby bylo dosaženo co nejlepšího výsledku.

Druhý způsob je učení bez učitele. Zde jsou správné výstupy neznámé, a tak se ze zadaných vstupních dat snaží učení získat společné zákonitosti a nastavit váhy i prahy tak, aby na podobné vstupy reagovala podobnými výstupy.

### 2.5.1 Back propagation

Je algoritmem učení neuronové sítě s učitelem, který slouží k adaptaci neuronové sítě na danou trénovací množinu. Rozděluje se do tří etap: dopředného šíření signálu ze vstupních dat, zpětného šíření chyby a úprav váhových parametru mezi perceptrony. Na začátku se náhodně navolí parametry sítě. Pro vstupní data se porovnají výsledky výstupu s těmi správnými, čímž se dopocítá chyba neuronové sítě. Úpravou váhových parametru například podle metody gradient descent lze dosáhnout přesnějšího výsledku.

Chyba sítě pro celý průchod testovacími vstupy se spočítá jako

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

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

### Gradient descent

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

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

Metoda gradient descent je omezena na hledání lokálních minim, čímž může dojít k minimalizaci chyby, ale nemusí získat nejlepší řešení. Po průchodu celé testovací množiny

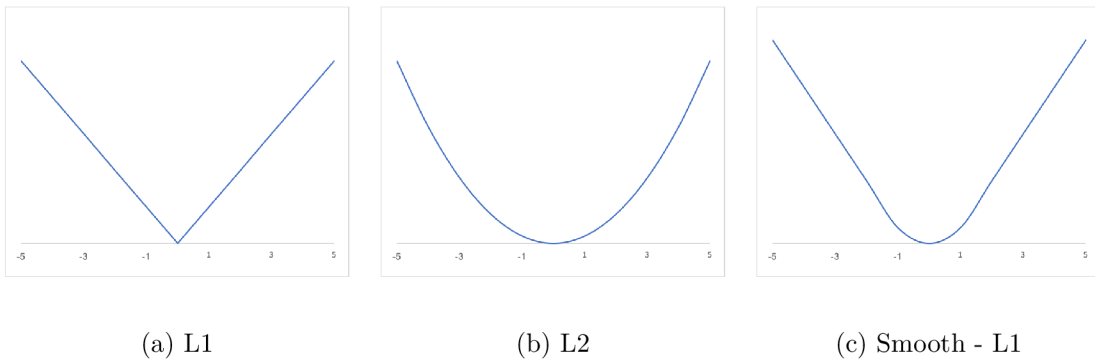
dojde k přenastavení váhových parametru a iterativně dochází ke snižování chyby, kdy se snaží dosáhnout nulového gradientu ( $\nabla E = 0$ ).

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

kde  $\eta$  reprezentuje rychlost učení, který definuje velikost kroku jednotlivé derivace.

### 2.5.2 Loss funkce

Loss funkce neboli chybová funkce vyjadřuje chyby pomocí matematického zápisu. Cílem chybové funkce je minimalizovat chybu nastavením vah a prahu při použití optimalizačních algoritmu. Následně jsou popsány významnější loss funkce. [6]



Obrázek 2.17: Orientační grafy vybraných loss funkcí

#### L1 loss

Neboli Least absolute deviation je součtem všech rozdílů mezi správným výstupem a výstupem získaným z neuronové sítě.

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

Hlavní vlastností této funkce je robustnost, díky které dokáže pracovat nad daty, které obsahují okrajové hodnoty. Nevýhodou této funkce je, že gradient se během průchodu celé skupiny testovacích dat nemění. Tím dochází k tomu, že gradient bude velký i pro malé chybové hodnoty.

#### L2 loss

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

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

Tato funkce si neumí dobře poradit s trénovacími daty, pokud obsahují větší množství okrajových hodnot, kdy při umocnění 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.

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

Funkce využívá pro výpočet malých  $a$  kvadratickou funkci a pro větší hodnoty  $a$  využívá lineární funkci, čímž zamezí rostoucímu stoupání v případě okrajových hodnot.

## Cross-Entropy Loss

Cross-entropy loss, také nazývaný log loss, měří výkonost klasifikačního modelu, jehož výstup obsahuje hodnoty pravděpodobností v intervalu od 0 do 1. Hodnota Cross-entropy lossu se zvyšuje, právě když se predikovaná pravděpodobnost vzdaluje od skutečného cíle. Často je tento loss spojován se softmax vrstvou. Cross entropy loss lze spočítat rovnicí:

$$H(P, Q) = - \sum_i P(i) \log Q(i) \quad (2.27)$$

kde množina  $P$  obsahuje hodnotu 1 pokud index  $i$  je skutečná klasifikace, jinak obsahuje hodnotu 0, a  $Q$  je množina obsahující pravděpodobnosti predikcí.

## Kapitola 3

# Technologie zpracování textu

### 3.1 Rozpoznání textu

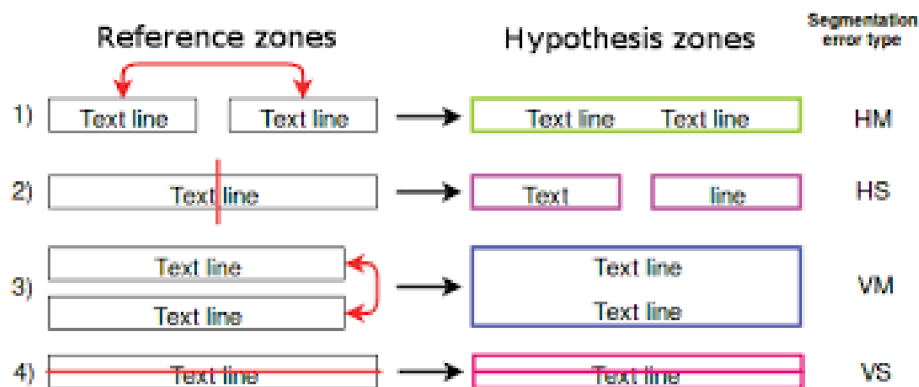
Castokrát je rozpoznání textu považováno za dva rozdílné úkoly, které jsou však propojeny. Temito úkoly jsou detekce a následné rozpoznání textu. Detekce textu nám navrhne pozici, kde se nachází může nacházet text v obrázku a rozpoznání v navržených textových oblastech rozpozná text. Zpravidla pro obe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

Většina standardních metod detekce textu pracuje s textem jako se sekvencí znaku. Tyto metody nejdříve naleznou jednotlivé znaky v obrázku, které dále shlukují do vyšších celků jako jsou slova případně řádky textu. Hlavními dvěma přístupy standardních metod jsou posuvné okno (sliding window) a spojené komponenty (connected-components). V poslední době se však obrátila pozornost na metody založené na hlubokém učení [22], které dokáží z obrázku dokumentu přímo označit slova.

Obecně cílem rozpoznání textu scény je dekodovat sekvenci znacek z pravidelne oríznutých textových obrázků s různou délkou textu. Většina metod zachycuje jednotlivé znaky a nekorektně klasifikované znaky jsou opraveny později. Kromě přístupu po jednotlivých znacích existují přístupy zaměřené na regiony textu. Ty můžeme rozdělit na tři kategorie: klasifikace slov (word classification), metoda dekodování sekvence na znaky (sequence-to-label) a metoda sekvence na sekvenci (sequence-to-sequence). Metoda klasifikace slov je metoda, kde jsou výstupem třídy reprezentující slova. Tento přístup však není univerzální a rozšířitelný. Zbylé dvě metody využívají rekurentních neuronových sítí.

### 3.2 Metriky porovnání

Algoritmy rozpoznání textu se zpravidla skládají z dvou hlavních celků segmentace obrazu na regiony s textem a rozpoznání textu v těchto regionech (kapitola 3.1). Pro porovnání kvality výsledku jednotlivých detekčních algoritmu je potřeba změřit jak na kvalitu segmentace obrazu, tak i kvalitu detekovaných textů v těchto regionech. Každý algoritmus pracuje jiným způsobem a může nastat situace kdy je text rozdělen do více oblastí (over-segmentation) anebo naopak se několik segmentů textu spojí do jednoho celku (under-segmentation). Tyto segmentační problémy mohou nastat jak v horizontálním, tak i v vertikálním směru a znesnadňují vyhodnocení výsledku algoritmu. Na obrázku 3.1 můžeme pozorovat varianty segmentačních problémů.



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

### 3.2.1 Levenštejnova vzdálenost

Levenštejnova vzdálenost (také známa jako editační vzdálenost, anglicky levenshtein distance či edit distance) je metrika pro určení podobnosti dvou seznamů. Podobnost měříme jako vzdálenost, s jakou můžeme upravit seznam tak, aby byl totožný s porovnávaným seznamem. Hodnota vzdálenosti nabývá součtu cen jednotlivých operací. Operacemi jsou přidání prvku (insertion), smazání prvku (deletion) a nahrazení prvku (substitution). Každá z těchto operací může nabývat jinou cenu za provedení.

Výpočet Levenštejnovi vzdálenosti můžeme vyjádřit následujícím vztahem:

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

kde  $x,y$  jsou porovnávané seznamy,  $i,j$  jsou indexy v rámci těchto seznamů  $x,y$ . Konstanty  $I, D, S$  představují ceny operací vložení, odebrání a změnu prvku. Celková vzdálenost dvou seznamů  $x$  a  $y$  je získána vypočtením hodnoty  $L_{x,y}(|x|,|y|)$ .

### 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 nejprve seskupí predikované a vzorové anotace. Dále jsou z každé skupiny počítány následující metriky.

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

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

Každá metrika se počítá pomocí levementní vzdálenosti a je v algoritmu přesně určeno, jak se zachovat při různě segmentovaných obrazových datech.

Pro všechny metriky jsou směřovací dvě hodnoty. Míra citlivosti a přesnost naměřených hodnot. Citlivost (anglicky recall) je poměr mezi korektně vyhodnocenými objekty a celkovým počtem vzorových objektů. Hodnota udává poměr pokrytí všech testovaných vzorových objektů. Zatímco přesnost (anglicky precision) je poměr mezi počtem všech korektně vyhodnocených objektů a celkovým počtem všech predikovaných objektů. Čím vyšší přesnost tím méně nekorektně predikovaných objektů.

Tato metrika byla v práci použita pro vyhodnocení systému OCR.

### 3.3 Existující řešení

#### Tesseract

Tesseract<sup>1</sup> je software pro optické rozpoznání znaku s veřejně dostupným kódem. Tesseract začal jako disertační projekt v rámci HP Labs v Bristolu. V letech 1984 až 1994 dále pokračoval vývoj v HP. V roce 1995 byl vylepšen a dosáhl lepších výsledků přesnosti. Později roku 2005 byl Tesseract firmou HP uvolněn jako veřejně dostupný kód.

Detekční algoritmus Tesseractu je složen z několika částí. Nejprve se nad vstupním obrazem provede adaptivní prahování metodou Otsu. Adaptivní prahování je potřeba pro zvýšení kontrastu mezi textem a pozadím. Dále nastane analýza rozložení stránky, která rozdělí obrázek na oblasti s textovými a netextovými informacemi. Potom následuje metoda pro nalezení řádku obsahující text. V rámci nalezených řádků se určí pozice dle horizontálních vzdáleností mezi znaky jednotlivých slov. Slova jsou poté rozdělena na samostatné znaky. Klasifikátor pak určuje jednotlivé segmenty znaku a detekuje přibližný jejich geometrický obrys, který je definován vektorem rysu v čtyřdimenzionálním (pozice x a y, směr, délka).

#### Microsoft Computer Vision

Jedná se o komerční službu firmy Microsoft, která poskytuje pomocí webového aplikačního rozhraní přístup k jejich produktu<sup>2</sup>. Mezi poskytované služby patří detekce objektů, které dokáže rozpoznat informace nacházející se v obrázku, označit je příslušnými značkami a vrátit jejich pravděpodobnost výskytu. Rozpoznání textu, které může z obrázku extrahovat slova do sekvence znaku citelnými počítačem. Další službou je rozpoznání ručně psaných textů, mezi které patří dopisy, články, formuláře a další.

Pro použití této služby je zapotřebí Microsoft účet, pro přístup k aplikačnímu klíči. Detekci textu provedeme zasláním dotazu obsahující výše zmíněný klíč a náš obrázek s textem na koncový bod určený pro detekci textu. V odpovědi dostaneme vyhodnocený text ve formátu JSON (javascript object notation). Ve výsledku můžeme pozorovat řádky textu a jednotlivá slova uvnitř těchto řádků. Záznamy se slovy obsahují hodnotu textu, jež systém vyhodnotil, a také souřadnice oblasti (bounding polygon), kde se v rámci obrázku text nachází. Struktury vyšší úrovně, například paragrafy, namísto slov samotných obsa-

<sup>1</sup>Dostupné z <https://github.com/tesseract-ocr/>

<sup>2</sup>Dostupné na <https://azure.microsoft.com/cs-cz/services/cognitive-services/computer-vision/>

hují jen souřadnice oblasti, do níž spadají všechna jejich slova. Průmerná doba detekce, díky komunikaci se serverem, dosahuje kolem jedné vteřiny.

Algoritmus využitý pro rozpoznání textu není zveřejněn. Algoritmus je dle mého názoru implementovaný jako série operací úpravy obrázku, nalezení řádku textu, narovnání těchto řádků (jsou-li řádky natočeny vůči horizontální ose textu), rozdělení na slova a na jednotlivé znaky těchto slov, následuje rozpoznání znaku, které je pravděpodobně implementované klasifikátorem pomocí neuronové sítě.

## Google Cloud Vision

Poskytuje několik možností, jak zaintegrovat natrénované modely počítačového vidění do programu třetích stran. Hlavním způsobem, jak takovou službu integrovat je za pomoci Cloud Vision aplikacího rozhraní (Cloud Vision API<sup>3</sup>). Toto rozhraní definuje koncové body a odpovědi pro jednotlivé definované moduly a probíhá pomocí webového protokolu HTTP. Mezi nabízenými službami jsou detekce objektu ve scéně obrázku, extrakce textu z obrázku, vyhledávání obrázku na webu (také na základě podobnosti obrázku).

Podobně jak je tomu u Microsoftu (viz výše) je i u Googlu zapotřebí účet a zaregistrování tohoto účtu v rámci Google Vision. V rámci účtu máme dostupný privátní klíč pro přístup k dostupným službám. Detekce textu se provede zasláním tohoto klíče současně s obrázkem, u něhož chceme detekovat text, na koncový bod pro detekci textu. Výsledek je vrácen ve formátu JSON. Detekce je vracena zanořenou hierarchií stránky, blok, paragraf, slovo, symbol. Každá z těchto struktur má definovanou oblast kde se v obrázku nachází a představuje logické celky odpovídající daným názvům.

---

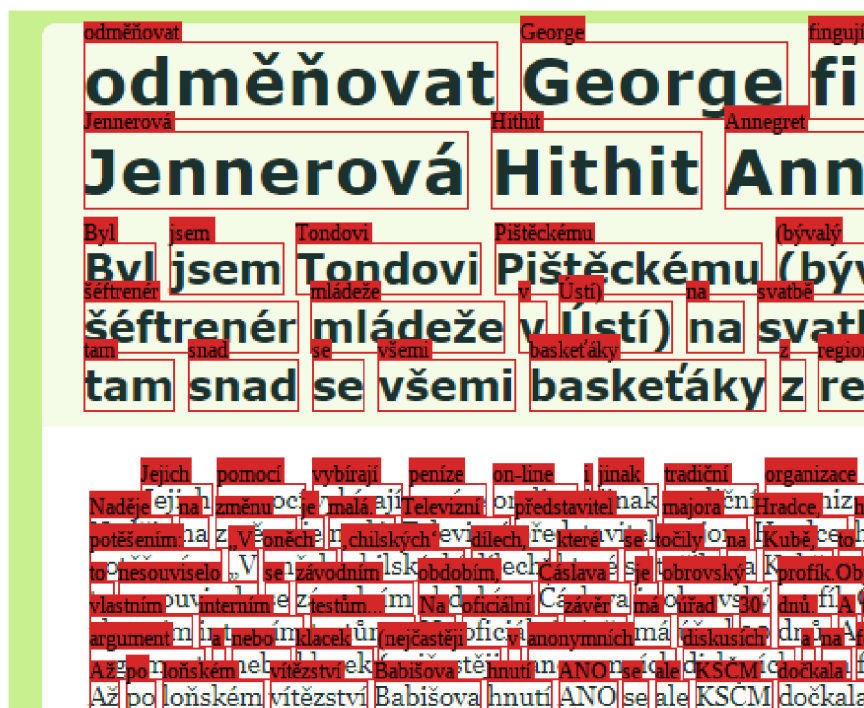
<sup>3</sup>Dostupné na <https://cloud.google.com/vision/>



## Kapitola 4

# Datová sada

Datová sada byla vytvořena pomocí interního nástroje firmy ARTIN zvaného „gui-generator“. Nástroj, jak již název vypovídá, dokáže generovat rozložení různých grafických webových rozhraní, které si můžeme zobrazit pomocí webových prohlížečů. Pomocí tohoto nástroje je možné vytvořit široké spektrum webových stránek s různými barevnými schémata, rozložením prvku a množstvím textu. Hlavní účelem gui-generátoru je generování trénovacích datových množin pro rozpoznávání aktivních prvků grafických uživatelských rozhraní. S každým vygenerovaným rozhraním jsou zároveň vytvořeny i anotace pro jednotlivé prvky obrazu. Anotace zpravidla obsahují informace o typu prvku (například tlačítko, slovo textu atd.) a pozici tohoto prvku v rámci obrazu. Pro účely této práce bylo zapotřebí tento generátor rozšířit o další funkcionalitu. Výsledek tohoto generátoru je možné vidět na obrázku 4.1. Obrázek je přiblížen a je v něm také možné pozorovat bounding boxy, které dále poslouží neuronové síti jako ucební vzory daných textů.



Obrázek 4.1: Ukázka vygenerovaných dat.

## 4.1 Texty

U generátoru bylo potřeba rozšířit rozsah textu, které je možné generovat. Hlavním důvodem tohoto rozšíření byla malá rozmanitost textu. Nové texty pro generátor byly sestaveny z různých českých textů posbíraných z internetu. Získané texty obsahují 120466 českých vět, 167589 českých slov a v nekomprimované podobě zabírají 23 MB diskového prostoru. Dále bylo potřeba uzpůsobit generátor, aby dokázal generovat webové stránky, jež obsahují nová textová data.

Původní verze generátoru vytvářela anotace pouze pro prvky rozložení dokumentu a interaktivní webové prvky. Toto však nebylo dostatečné pro vytvoření anotovaných dat vhodných k rozpoznání textu. Pro interaktivní prvky (formulářové prvky a.j.) však nebylo možné přímo přistoupit 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 rozdělena a vložena do nenastýlovaných prvků „span“, které nám poskytli možnost určit souřadnice oblasti ve které se nachází texty jež tyto prvky obalují. Přidané anotace jsou pro blok textu, slova textu a jednotlivé symboly textu. Bílé znaky generátor vynechává a nijak je neanotuje.

## 4.2 Styly

Pro větší rozmanitost generovaných textů v snímkách byl přidán do generátoru náhodný výběr ze širokého seznamu písem. Důvodem této změny bylo otestování rozpoznávací schopnosti pro rozmanité styly textu. Pro každou generovanou stránku jsou náhodně zvoleny písma pro nadpisy, tlačítka a všeobecný text. Zároveň při tom je i náhodně zvolena velikost tohoto písma, vzdálenost mezi jednotlivými znaky a výška mezi dvěma řádky textu. Jiné modifikace textu (jako je náklon písma, tučné písmo atd.) jsou voleny náhodně při vkládání písma do stránky.

Generátor generoval hlavně formuláře a jiné webové rozložení a spíše se zabýval množstvím a pozicí aktivních prvků v rámci těchto webových stránek. Pro účely této práce bylo zapotřebí u současných stylů vyměnit generované texty (viz výše). Protože texty mohli nabývat větší velikosti (oproti původnímu stavu), bylo potřeba některé styly pozmenit, aby změněný text neznicil vytvořené rozložení. Zároveň byl pro generátor vytvořen i čistě textový styl. Tento styl generuje internetový článek a skládá se z hlavních částí hlavičky a těla článku. V rámci hlavičky se vytvoří hlavní nadpis článku a stručný úvod co můžeme v článku hledat. Tělo článku pak obsahuje množství paragrafů a náhodně do něj mohou být vygenerovány i obrázky. Generátor tento styl naplňuje pomocí získaných textů. Barevné rozložení těchto stylů je generováno tak, aby pro zvolenou barvu pozadí byl text citelný.

## 4.3 Generování testovací datové sady

Pro automatické generování datové sady byl vytvořen nástroj, jenž automaticky ovládá webový prohlížeč, který je potřeba pro vykreslení rozložení (layout) poskytnutém pomocí gui-generatoru. Toho bylo dosaženo využitím knihovny Selenium<sup>1</sup>, jejíž hlavním účelem je vytvoření a spuštění automatizovaných testů webových rozhraní, a programu Chrome-

<sup>1</sup>Dostupné z <https://www.seleniumhq.org/>

Driver<sup>2</sup>(nastavba nad webovým prohlížečem Google Chrome), který obsahuje rozhraní pro možnost ovládání robotem (v tomto případě Selenium). Nástroj nejprve vytvoří snímek obrazovky ovládaného prohlížeče (tím získáme obrázek s texty) a následně pro tento obrázek získáme anotace pro slova a jejich symboly. Struktura vygenerované anotace pro snímky vypadá následovně:

```
{
  "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 anotací pro tento obrázek, které se skládají z bounding boxu určeného dvěma body, typu označující anotace(type) a hodnoty text.

---

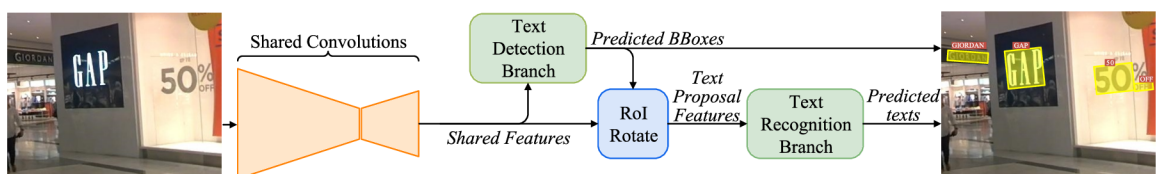
<sup>2</sup>Dostupné z <http://chromedriver.chromium.org/>

# Kapitola 5

## Návrh řešení

Navržený systém využívá architekturu inspirovanou Fast Oriented Text Spotting [15]. Architektura využívá jednu společnou síť jak pro detekci i rozpoznání textu. Skládá ze čtyř hlavních částí: společná konvoluční síť, detekce textu, prostorové transformace a rozpoznání textu.

Celkovou architekturu můžeme pozorovat na obrázku 5.1. Společná konvoluční síť (shared convolutions) ze svého vstupu získá obrazové vlastnosti (features). Pro nalezení vlastností v obraze bude použita reziduální konvoluční síť inspirovaná sítěmi ResNet50 [?] a Feature Pyramid [14], které z obrazu extrahuje vlastnosti a spojí je dohromady přes několik úrovní. Výsledná mapa obrazových vlastností je společným vstupem pro část s detekcí a rozpoznáním textu. Detekce textu (Text Detection Branch) produkuje pro každý pixel obrazu predikce regionu textu za použití společné mapy vlastností. Regiony textu, navržené detekcí textu, jsou dále transformovány, za pomoci prostorové transformace, do datové reprezentace s pevnou výškou. Tato operace zachovává původní poměr stran regionu. A nakonec je aplikováno rozpoznání textu (Text Recognition Branch). Pro tyto účely budou použity konvoluční neuronové sítě s pametovými vrstvami (Long Short-Term Memory dále jen LSTM [9, 19]), které zakódují sekvenci textových informací. Následuje klasifikace textu pomocí rekurentní neuronové sítě CTC (Connectionist Temporal Classification [7]) Strukturu částí detekce, korekce a rozpoznání si více rozvedeme v následujících kapitolách.



Obrázek 5.1: Architektura sítě FOTS. Zdroj [15]

### 5.1 Detekce textu

Detekce textu je tvořena plně konvoluční sítí. V obrázku se spíše nachází větší množství malých textových regionů, proto je výstupní mapa obrazových vlastností zvětšena. Po extrakci obrazových vlastností sdílenou sítí je na ne aplikována jedna konvoluce na zjištění přítomnosti textu v daných pixelech. Dalšími predikovanými hodnotami pro všechny pozitivní vzorky jsou čtyři dimenze vzdáleností (vzdáleností nahoru, doprava, dolů, doleva), které

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

Hodnotící funkce detekce se skládá z dvou částí: ztráta při klasifikaci textu a ztráta regrese bounding boxu. Ztrátu klasifikační funkce vypočítáme jako ztrátu klasifikace jednotlivých pixelů pro zmenšenou mapu hodnocení (anglicky score map). Pouze zmenšená verze původního textového regionu je považována za správnou oblast. Oblast mezi bounding boxem a zmenšenou verzí mapy nijak nepřispívá k celkové ztrátě klasifikace. Zatímco regresní ztráta pro bounding boxy je spočítána poměrem průniku a sjednocení jejich oblastí (Intersection over Union dále jen IoU [18]) a pomocí ztráty úhlu rotace. Celková ztráta je vážený součet těchto dvou ztrát.

## 5.2 Prostorová transformace

Pomocí operátoru RoIPool provedeme korekci predikovaných textových oblastí. RoIPool aplikuje transformaci na textové oblasti a získáme textové oblasti zarovnané podle souřadných os (anglicky axis-aligned). Textové oblasti transformujeme tak, že pro všechny nastavíme stejnou pevnou výšku a šířku dopočítáme tak, aby zůstal poměr stran nezmenšený. To umožní zpracování různě dlouhých textů. Pro vzorkování hodnot, jejichž souřadnice jsou mapovány mimo diskrétní celočíselné, je použita bilineární interpolace. Tím zamezíme chybám v zarovnání mezi korekcí a získanými vlastnostmi. Navíc tím umožníme proměnnou délku výstupních oblastí, což je více vhodné pro rozpoznání textu.

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

## 5.3 Rozpoznání textu

Rozpoznání textu je zaměřeno na označení veškerých textových informací z dat získaných z předchozích kroků. Tato část se skládá ze sekvencí konvolucí a pooling sítí podobných VGG, obousměrného LSTM, plně propojené vrstvy určené ke klasifikaci a zakončené CTC dekodérem [7].

Nejprve je na prostorová data aplikováno sérií konvolucí a poolingů. Právě pooling vrstvy jsou nastavené, aby redukovaly rozměry pouze ve vertikální ose. To má za důsledek, že extrahujeme vlastnosti do vyšších úrovní a zachováváme šířku nezmenšenou. Tyto vlastnosti vyšší úrovně jsou převedeny do sekvence vlastností na základě horizontální šířky vstupních vlastností. Ty jsou vstupem rekurentní neuronové sítě LSTM. LSTM zakóduje získanou sekvenci do 256dimenzionálního vektoru pro každý směr. To je potřeba pro zachycení všech rozměrových závislostí, kterých text může nabývat. Každý ze skrytých stavů LSTM je vyhodnocen v obou směrech v každé časové jednotce. Tyto stavy jsou dále napojeny do plně propojené vrstvy, která vypočítá společně s vrstvou softmax hodnoty pravděpodobnosti výskytu znaku z dané abecedy. Následně je použito CTC pro dekodování jednotlivých výstupů z rámcu na výstupní sekvenci 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 při implementaci.

### 6.1 Použité technologie

#### Python

Python je vysokoúrovňový skriptovací jazyk. Poskytuje programování v různých paradigmatech jako jsou objektově orientované, imperativní, procedurální i funkcionální. Jazyk poskytuje systém pro správu závislostí pip. V posledních letech při výrazném nástupu strojového ucení se stal velmi oblíbený pro svoji jednoduchost a také kvůli tomu, že jej lze navázat na zkompileované knihovny (například v jazyce C) s nimiž dosahuje jazyk vyššího výkonu. Dalším důvodem použití je že lze pomocí něj snadněji prototypovat algoritmy neuronových sítí v některém z dostupných frameworku.

#### Pytorch

Knihovna PyTorch<sup>1</sup> je framework pro práci s neuronovými sítěmi pro jazyk Python. Je určen 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 několik hlavních výhod jako je pocitání neuronových sítí za pomocí tenzoru se velkou akcelerací pomocí grafických procesoru (GPU). Tenzory můžeme jsou matematický objekt, který v rámci programování můžeme chápat jednoduše jako multidimenzionální pole. PyTorch používá funkci zvanou automatic differentiation. Jedná se o metodu, která sleduje změny provedené na tenzorech a je schopna tyto operace revertovat a na základě nich vypočítat gradienty. To je velmi užitečné při tvorbě neuronových sítí, jelikož umožňuje výpočet diferencí parametru už při dopředném průchodu sítí. PyTorch navíc poskytuje mnoho implementací funkcí, vrstev, aj., které byly zmíněny v kapitole 2. To je důvodem, proč byl v práci použit.

<sup>1</sup>Dostupné na adrese <https://pytorch.org/>

## 6.2 Rozdělení systému

System je rozdělen na dva spustitelné soubory, každý s jiným účelem. Prvním z nich je aplikace určená pouze k ucení síte. Prijímá na vstupu konfigurační soubor, který udává parametry síte, kterou chceme ucit. V konfiguračním 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 patří 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. Učení je pak dále rozděleno 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 dopředným pruchodu síti. Poté necháme model dopředným pruchodem zpracovat tyto prvky. Pri verzi ucení získáme na výstupu pouze loss funkce síte, protože predikované výsledky modelu nejsou pri ucení příliš užitečné. Za pomoci loss funkcí zpětným pruchodem modifikujeme gradienty modelu, k tomu je užitečná knihovna pytorch a její modul autograd. Gradienty síte jsou spočítány tak jsou pomoci optimalizačního algoritmu aplikovány. V rámci tohoto projektu byl použit optimalizátor Stochastic Gradient Descent, který umožňuje iterativne aplikovat metodu ucení s ucitelem gradient descent (více v kapitole 2.5.1). Po dokončení epochy je model uložen a upraven krok plánovace ucení. Plánovac ucení je rozdělen na dve části. Zpocátku do nastaveného poctu epoch je ucení postupne „zahríváno“, kdy začneme s nízkou váhou ucení a postupne ji zvyšujeme, dokud nedosáhneme nastavené epochy. Tento princip nazýváme gradual warmup [20] a umožňuje 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 „trpělivost“, pak sníží váhu ucení. Celý tento systém můžeme pozorovat jako pseudokód v algoritmu 1.

---

**Algoritmus 1:** Učení systému.

---

```
1 for  $i = 0$ ;  $i < MAX\_EPOCH$ ;  $i++$  do
2   Zamíchání indexu datasetu.
3   for  $j = 0$ ;  $j < NUM\_ITEMS\_DATASET$ ;  $j++$  do
4     Získání dat datasetu pro index  $j$ .
5     Aplikace preprocessing transformací na tato data.
6     Vynulování gradientu síte.
7     Loss  $\leftarrow$  Dopředný pruchod síti.
8     Zpětný pruchod síti pomoci vypocítaných lossu.
9     Krok optimalizačního algoritmu síte.
10  Vytisknutí výsledku ucení epochy.
11  Uložení stavu síte pro danou epochu.
12  Krok plánovace ucení síte.
```

---

Druhou verzí systému je samotná inference. Máme již naučený model pomocí předchozí aplikace a chceme predikovat boxy i texty pro zadaný vstupní obrázek. Aplikace nacte obrázek a aplikuje na něj nezbytné transformací operace, které jsou vyžadovány modelem. Mezi ne patří normalizace velikosti obrázku a jeho následné převedení na tenzor. Takto získaný tenzor je předaný modelu nastaveného na evaluaci. Po dopředném průchodu modelem dostaneme predikované regiony textu, jejich textový obsah a pravděpodobnost, kterou model považuje za validní box. Algoritmus inference je možné videt v algoritmu 2

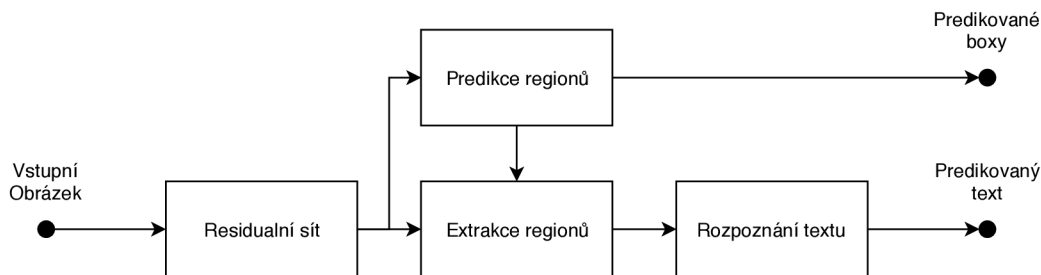
---

**Algoritmus 2:** Inference systému.

---

- 1 Nactení stavu sítě z poslední epochy.
  - 2 Nactení obrázku ze zadané cesty
  - 3 Aplikace preprocessing transformací na tato data.
  - 4 Skóre, Boxy, Texty  $\leftarrow$  Dopředný průchod sítí.
  - 5 Výstup těchto dat do souboru.
- 

Model tohoto systému je možné rozdelit do niekoľik funkčne odlišných částí. Tyto části můžeme pozorovat na blokovém schéma systému v obrázku 6.1, a budou dále popsány v následujících kapitolách.



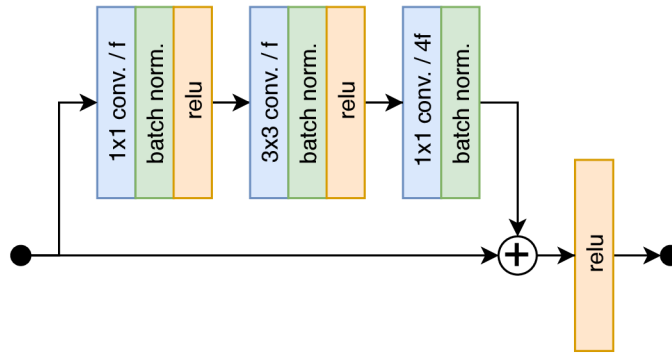
Obrázek 6.1: Celková architektura modelu sítě.

### 6.3 Residuální síť

Prvním modulem neuronové sítě je residuální neuronová síť. Tento druh sítě slouží získání všech možných vlastností ze vstupního obrazu a je často předtrénován na ImageNetu[3]. To je dataset, který obsahuje miliony různých obrázků v různých kategoriích. Použití předtrénovaných vah je vhodnější, protože výsledná síť bude rychleji konvergovat. Tuto praktiku nazýváme transfer learning [17] a umožňuje sítím učit se nové úkoly snadněji, protože nemusí začínat od nuly s náhodnými váhami.

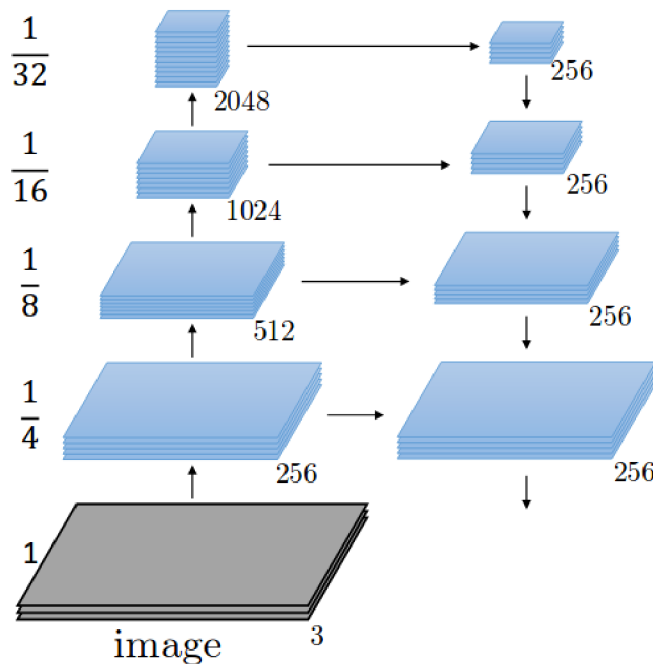
V rámci této práce je residuální síť je navržena po vzoru ResNet-50 [8]. Síť se skládá z takzvaných residuálních bloků, které umožňují vstupním datům projít skrz síť beze změny. Díky tomu je možné touto sítí dobře šířit gradient. V každém z těchto bloků existuje propojení, které umožní vstupním hodnotám preskocit konvoluční operace. Residuální bloky se také liší v počtu kanálů, které produkují. V obrázku 6.2 můžeme pozorovat residuální blok. V ResNet-50 se jedná o 4 vrstvy, kde každá má předem stanovený počet bloků, které jsou poskládány za sebe. První blok každé z vrstev využívá prostorový krok 2 a tím postupně snižuje rozměry vstupu.





Obrázek 6.2: Residuální blok sítě ResNet-50

Tato síť však postupným průchodem vstupní jeho vrstvami snižuje velikost vstupu až 32krát, čímž se velmi rychle ztrácí data o malých objektech v obraze. To často nemá dopad například při rozpoznání objektu, kdy se v obraze nachází jen několik objektu. Avšak je to velmi nepraktické při detekci textu, protože počet objektu v obraze může být daleko větší. Síť ResNet-50 je upravena po vzoru Feature pyramid network [14], tak aby byla více citlivá na vstupy menších rozmeru. Na obrázku 6.3 můžeme pozorovat výslednou residuální síť. Všechny výstupy z vrstev ResNet jsou využity při výpočtu 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 součtu přes všechny elementy. Nejprve musí být hornejší vrstvy nadzorkovány, aby odpovídali velikosti vrstvy podélného propojení.



Obrázek 6.3: Feature pyramid network vycházející z ResNet. Zdroj: <http://presentations.cocodataset.org/COC017-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 část popisuje získání bounding boxu v zadaném obraze. Získané boxy slouží jako výstup systému a jsou použity pro další zpracování, pomocí něhož se predikuje text nacházející se právě v těchto boxech. V následujících podkapitolách je popsáno, jak k predikci boxu dochází.

### 6.4.1 Anchory

V rámci této práce je potřeba predikovat bounding boxy, které obsahují text. Avšak bounding boxy mají různé rozmery, jejich počet i pozice jsou také proměnné. Tyto vlastnosti mohou vést ke špatné konvergenci učícího se systému. Proto bylo potřeba tyto bounding boxy reprezentovat způsobem, který není závislý na rozměru boxu. Řešením je předem stanovit bounding boxy jejichž počet 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 přístup je inspirován dle publikace [21].

Máme-li 4 souřadnice anchoru, pozice středu, šířku a výšku  $(a_x, a_y, a_w, a_h)$ , můžeme ho transformovat za pomoci 4 transformačních vah  $(d_x, d_y, d_w, d_h)$ , které určují míru posunu středu a změnu měřítka šířky a výšky. Rovnice pro transformaci anchor boxu jsou uvedeny níže:

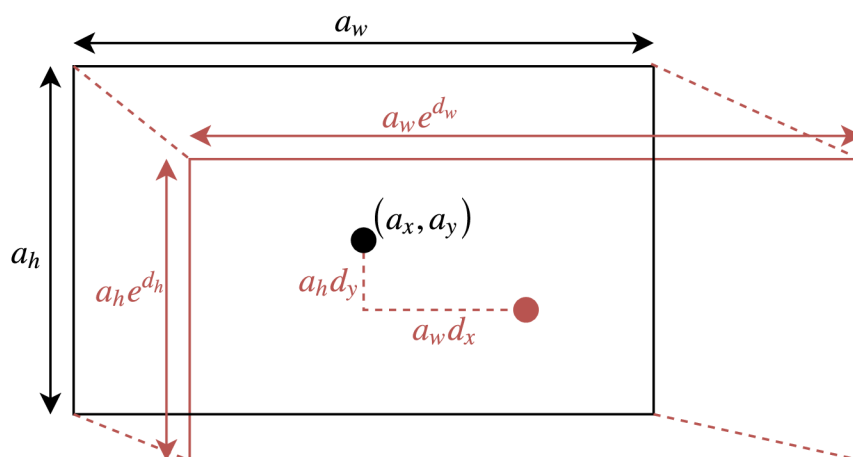
$$r_x = a_x + a_w d_x, \quad (6.1)$$

$$r_y = a_y + a_h d_y, \quad (6.2)$$

$$r_w = a_w e^{d_w}, \quad (6.3)$$

$$r_h = a_h e^{d_h}, \quad (6.4)$$

kde souřadnice  $(r_x, r_y, r_w, r_h)$  znázorňují transformovaný bounding box. Takto definovanou transformaci je možné snadno učit, protože hodnoty  $(d_x, d_y, d_w, d_h)$  jsou invariantní vůči rozměrům 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 původní bounding box. Grafické znázornění, jak transformace modifikuje box je možné vidět na obrázku 6.4, kde v černé barvě je znázorněn anchor box, na který je aplikována transformace, a v červené barvě je možné pozorovat výsledek této transformace.

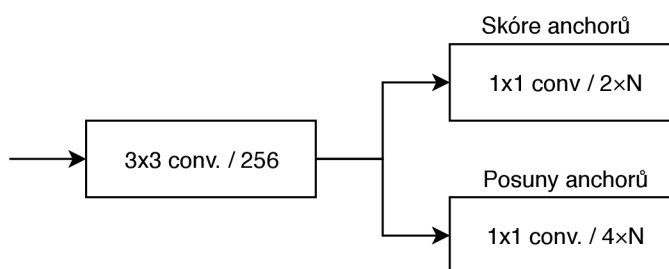


Obrázek 6.4: Transformace anchor boxu pomocí transformacních parametru.

Protože se tato práce zabývá zpracováním textu bylo, potřeba zvolit anchor boxy, které alespon matne připomínají bounding box obsahující text. Podíváme-li se na jakýkoli text, můžeme videt, že většina jeho slov rozmerove je více široká, nežli vysoká. Proto je i logické, aby v tomto duchu byly zvoleny i anchor boxy. Konkrétněji jsou zvoleny boxy s pomérem šířky a výšky 1:1, 2:1, 4:1 a 8:1, pri merítkách 8, 16, 32 a 64. To dohromady dáva 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 určené ke klasifikaci, resp. regresi transformacních souradnic anchor boxu. Schéma modulu můžeme videt na obrázku 6.5



Obrázek 6.5: Region proposal network. Promenná N značí počet anchoru.

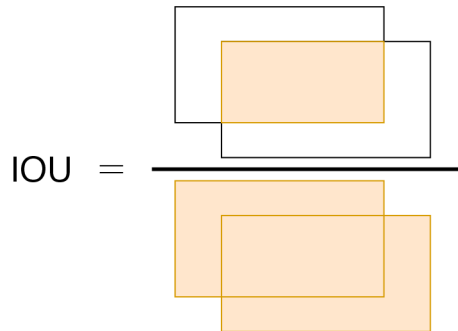
Klasifikační vrstva obsahuje pro každý pixel mapy vlastností dvojici hodnot pro každý anchor na které je dále aplikována vrstva Softmax (více kap. 2.4.5). Tyto dve hodnoty pro každý anchor určují míru pravdepodobnosti, že práve tento anchor spadá do jedné ze dvou navzájem se vylucujících tříd pozadí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

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

Regresní vrstva podobne jako klasifikační 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 klasifikační vrstve musí mít i tato vrstva při učením preprocesované trénovací vzory. Ty jsou v tomto případě transformací hodnoty, které když aplikujeme na daný anchor tak dostaneme některý z validních textových boxů. Při preprocesingu je zároveň získána i maska, která jednoduše určuje, které hodnoty je třeba 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 spočítána pomocí Smooth L1 Loss (s redukcí pomocí operace scítání, více 2.5.2) nad rozdílem predikce vynásobené maskou a trénovacím vzorem. Vypočtenou chybu pro regresní vrstvu značíme  $\mathcal{L}_{REG}$ . Masku je použita, protože všechny hodnoty masky nastavené na 0 se nebudou podílet na získání hodnoty chybové funkce. Je-li hodnota masky pro nějaký anchor nastavená na hodnotu 0 pak je i hodnota vzorové transformace pro daný anchor nastavená právě na 0. Rozdíl predikce vynásobené maskou a vzorem pro tento anchor se rovná 0, což znamená, že pro tuto kombinaci není, co bylo třeba optimalizovat.

### 6.4.3 Příprava vzoru pro učení

Aby se síť naučila klasifikovat ty správné boxy, jsou v preprocesingu náhodně navzorkovány pozitivní a negativní anchory, všechny ostatní jsou v trénování vynechány. Vzorky, které mohou být považovány za pozitivní, či negativní, a mohl je algoritmus náhodně vybrat jako kandidáty k učení, musí se anchor box a vzorový anotovaný box z velké části překrývat. Hodnota překryvu je dána funkcí Intersect over Union [18] (dále jen IOU), která vypočítá poměr 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 vůbec nepokrývají. Tuto funkci můžeme pozorovat graficky v obrázku 6.6.



Obrázek 6.6: Funkce IOU znázorněna graficky pro dva boxy.

Dalším kritériem výberu anchoru, je že žádný z kandidátních anchorů pro trénování nesmí přesahovat za hranice obrázku. Všechny anchory, které ano jsou ignorovány. Algoritmus nejdříve vybere všechny kandidáty, které nejlépe překrývají jeden z anotovaných vzorových bounding boxů. Dále jsou náhodně dovybrány ze všech kandidátů ti, kteří mají hodnotu překryvu vyšší, než je stanovený pozitivní práh (v tomto projektu se jedná o hodnotu 70%), aby byl splněn počet pozitivních záznamů. Všechny vybrané pozitivní záznamy, ztotožňují anchory jejichž ohodnocení chceme zlepšovat. Jakmile jsou všechny pozitivní záznamy vybrány je potřeba ještě určit negativní anchor boxy. Ty se vybírají oproti pozitivním boxům

opacne. Ze všech zbylých kandidátů se vyberou ty, které mají hodnotu překryvu nižší než stanovený negativní práh. To jsou zase hodnoty anchoru, které při učením zhoršíme. Všechny ostatní anchory ignorujeme. Vzor trénování klasifikace nabývá třech hodnot, 0 pokud se jedná o negativní anchor, 1 pokud je anchor pozitivní a -1 pokud je anchor box při trénování ignorován. Počet těchto vzorků a poměr pozitivních vůči negativním anchorům a hodnota hranice, kdy je box považován jako kandidát pro učením (at už pozitivní, či negativní vzorek), je nastavitelný v konfiguračním 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 těchto pozitivních anchorů je vybrán anotovaný box, který má největší hodnotu překryvu. Tím pádem se daný anchor bude snažit co nejvíce napodobit tento anotovaný box. Proto je potřeba určit transformací vektor, který je potřeba k transformaci daného anchor boxu na anotovaný box. Ten získáme pomocí následujících rovnic:

$$t_x = \frac{g_x - a_x}{a_w}, \quad (6.5)$$

$$t_y = \frac{g_y - a_y}{a_h}, \quad (6.6)$$

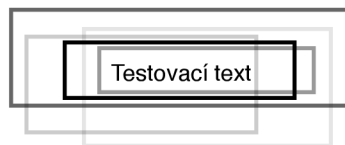
$$t_w = \ln \frac{g_w}{a_w}, \quad (6.7)$$

$$t_h = \ln \frac{g_h}{a_h}, \quad (6.8)$$

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 transformací parametry podle nichž se bude sít pro daný anchor box optimalizovat, aby dosáhla anotovaného boxu. Zároveň s výpočtem tohoto transformací vektoru pro daný anchor je pro něj nastavena hodnota masky  $(M_x, M_y, M_w, M_h)$  na  $(1, 1, 1, 1)$ . Pro všechny nezúčastněné anchor boxy jsou nastaveny vektory vzoru transformací parametrů a masky na hodnotu  $(0, 0, 0, 0)$ .

#### 6.4.4 Filtrování predikovaných regionů

Z důvodu, že se pro každý pixel mapy vlastností predikují transformace mnoho (v našem případě 16) anchor boxů. Kvůli jejich obrovskému množství (960000 predikcí pro obrázek velikosti 1200x800) je potřeba z těchto vzorků vybrat ty nevhodnější. Nejprve seřadíme seřazeně ohodnocení pravděpodobnosti boxů, že je text. Z těchto pravděpodobností vybereme prvních N boxů (konfigurovatelné pomocí hodnoty `num_boxes_pre_nms`). Tím dostaneme pouze ty nejlépe ohodnocené boxy, které byly predikovány. Avšak může nastat situace, kdy více predikcí spadá do stejné oblasti obrázku a predikují totožný objekt. Tuto situaci můžeme pozorovat v obrázku 6.7, kde jsou zobrazeny boxy s různou hodnotou pravděpodobnosti, které je zobrazeno intenzitou šedé barvy.



Obrázek 6.7: Predikce boxů, které se překrývají v dané oblasti.

Pro odfiltrování těchto 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 těchto seznamu.

*threshold* - Práh pro hodnotu prekryvu, nad kterou potlačujeme.

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

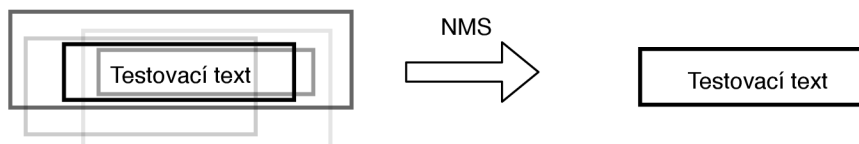
```

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 supressed[idx] is False then
7         keep.push(idx)
8         for j = i + 1; j < N; j++ do
9             jdx ← order[j]
10            if supressed[jdx] is False then
11                if IOU(box[idx], box[jdx]) ≥ threshold then
12                    supressed[jdx] ← True

```

---

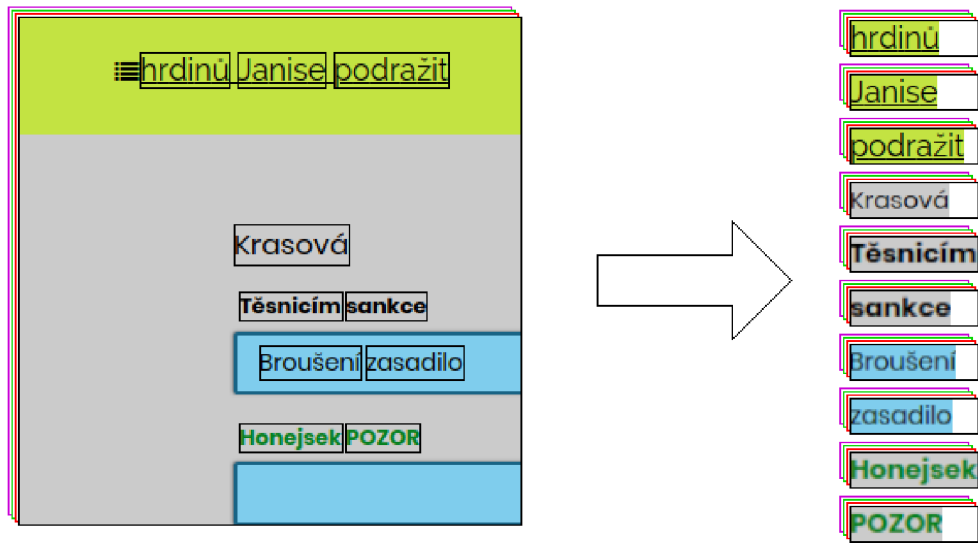
Aplikací algoritmu na boxy z předchozího obrázku můžeme pozorovat v obrázku 6.8. V obrázku je patrné, že je zachován pouze ten nejlépe ohodnocený box (nejvíce intenzivní hodnota šedi) a všechny ostatní boxy jsou algoritmem potlačeny. Díky tomu nevznikají duplikáty pro stejné oblasti.



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

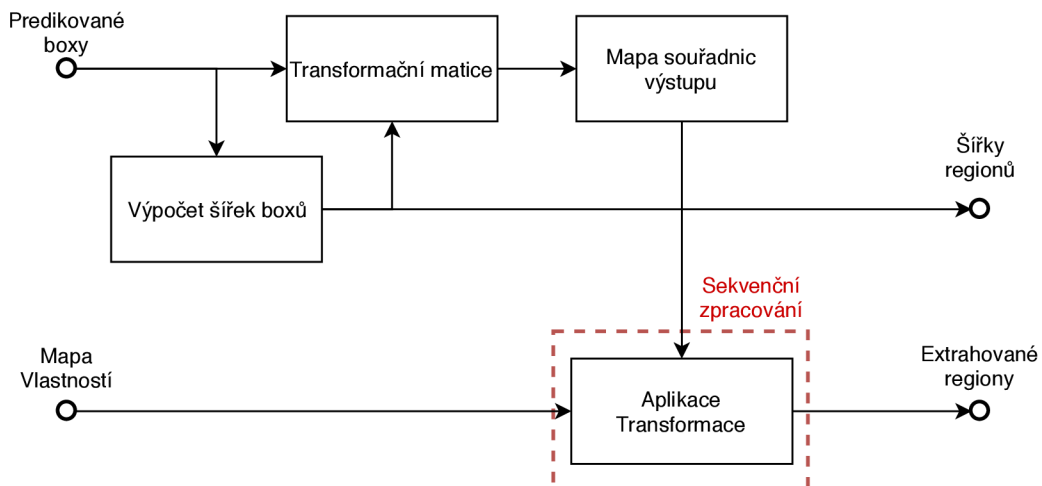
## 6.5 Extrakce regionu

Dalším krokem extrakce regionu získaných v předchozím kroku z mapy vlastností. Tento krok je vhodný, protože další části se budou pracovat pouze s daty jež patří danému regionu. Tento krok můžeme rozdelit na dvě části, příprava transformací dat a aplikování těchto transformací pomocí spatial transformer network [10] na zadanou mapu vlastností. Výsledek, který tento modul produkuje, můžeme 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 představu jsou mapa vlastností i výsledné regiony zobrazeny pomocí vstupního obrazu, ve skutečnosti se však jedná o tenzor s 256 kanály.



Obrázek 6.9: Extrakce regionu z mapy vlastností.

Texty v obraze není vhodné zmenšit do stejné rozměrné oblasti jako se tomu děje v RoI poolu metody R-CNN [4, 5], a to zejména kvůli dvěma faktorům. Slova textu mají variabilní délku a kdyby se slova o různých délkách měli vměstnat do stejné oblasti tak dojde ke ztrátě informací následná klasifikační síť by mohla mít problémy. Tento princip je však ale vhodný pro zpracování jednotlivých písmen, ale kvůli tomu by musel být anchor boxu mnohem větší a mapa vlastností více citlivá. Dalším faktorem, proč tento způsob není vhodný jsou různé fonty. Stejná slova zobrazená s jinými fonty mohou mít úplně odlišné charakteristiky a opět může při zmenšení do stejné oblasti dojít ke ztrátě dat. Blokovaný 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 šířku jako promennou. A obecně můžeme pro jeden box tuto transformaci lze zapsat jako zobrazení jež mení rozmery tenzoru:

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

kde  $C$  je počet kanálu vstupního i výstupního tenzoru,  $H$  a  $W$  jsou šířka, resp. výška vstupního tenzoru, a  $H'$ ,  $W'$  jsou šířka, resp. výška výstupního tenzoru.  $H'$  značí stanovenou konstantní výšku a  $W'$  se na základe boxu musí dopočítat.

$$w_t = h_t \frac{x_{\max} - x_{\min}}{y_{\max} - y_{\min}}, \quad (6.10)$$

kde  $h_t$  je zvolená konstantní výška okna,  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$ ,  $y_{\max}$  značí souřadnice boxu,  $w_t$  je šířka okna pro daný box. Rozmery  $h_t$  a  $w_t$  dohromady udávají velikost výstupního tenzoru pro tento box (hodnoty pro  $H'$ ,  $W'$ ). Obecně každý z predikovaných boxu může mít různou šířku, ale pro další zpracování používáme tu největší šířku 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 počet predikovaných boxu). Regiony, jež jsou kratší jsou do dané šířky doplneny prázdnými hodnotami.

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

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

Na základe této transformací matice lze určit pro souřadnice vstupního obrazu souřadnice ve výstupním:

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

kde  $(x_s, y_s)$  jsou souřadnice ze zdrojového tenzoru,  $(x_t, y_t)$  jsou souřadnice v cílovém tenzoru,  $1 \leq i \leq H'$  a  $1 \leq j \leq W'$ . Toto mapování však není příliš praktické, protože při určitých



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

$$V_{ij}^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_{ij}^s - m|) \max(0, 1 - |y_{ij}^s - n|), \quad (6.13)$$

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 případě 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 zohledněn, to stejné platí i pro osu  $y$ . Dohromady pokud jsou výsledky obou funkcí  $\max$  nenulová je vzorek  $U_{nm}^c$  zahrnut ve výsledku poměrem daným dle výsledku zmíněných funkcí.

Aplikování této transformace je provedeno pomocí knihovny PyTorch, která výše uvedenou rovnicí realizuje. Konkrétně se jedná o funkci `grid_sample`, která na základe vstupní mřížky souradnic, jež má rozměr  $\mathbb{R}^{B \times H' \times W' \times 2}$  (promenná  $B$  znázorňuje počet dávek zaráz), a zdrojového tenzoru určí hodnoty výsledného okna. Mřížka souradnic je velikostí výstupního okna, které požadujeme a pro každou z těchto souradnic je určena dvojice souradnic  $(x, y)$  odpovídající zdrojovému obrazu, tyto hodnoty získáme pomocí inverzní matice k  $M$  uvedené výše. Souradnice v rámci této funkce jsou normalizovány do rozsahu  $(-1; 1)$ , proto je ještě potřeba transformované souradnice dále normalizovat pomocí:

$$\bar{x}_{ij}^t = 2 \frac{x_{ij}^t}{W} - 1, \quad (6.14)$$

$$\bar{y}_{ij}^t = 2 \frac{y_{ij}^t}{H} - 1, \quad (6.15)$$

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

Jak už bylo zmíněno výše je detekce textu velmi závislá korektnosti detekce, proto při trénování používáme pouze validní anotované boxy narozdíl klasicky používaných algoritmu pro detekci objektu. Při použití predikovaných boxů by mohli vzniknout situace kdy predikovaný box nepokryje veškeré slovo textu, nebo box může být výrazně větší než daný text a jiné. Tyto situace mají pak špatný účinek na efektivní trénování sítě pro detekci textu a tomu se právě predejde použitím anotovaných boxů, protože u nich víme, že přesně odpovídají textu, jež ohranicují.

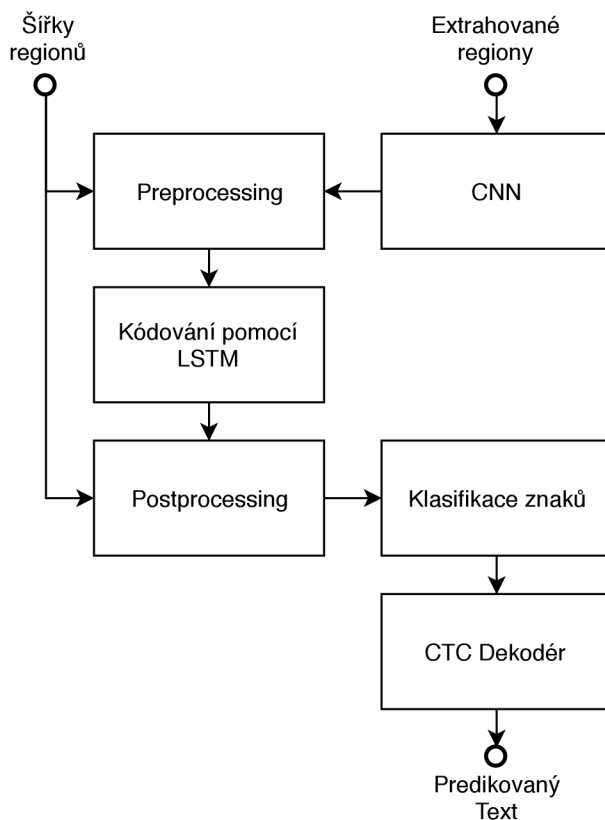
Tento modul provádí transformace mapy vlastností jednotlivě po boxech. Důvodem toho je velmi velká pametová náročnost funkce `grid_sample` při transformaci všech boxů zaráz. Chceme-li tuto metodu aplikovat pro dva boxy paralelně je potřeba duplikovat vstupní mapu vlastností. Velikost této mapy je variabilní, ale při učením je omezena na rozměry  $200 \times 300$  s 256 kanály. Pridáme-li k tomu velikost datového typu float16 a pak dostaneme 30 MB na jeden obrázek. Počet boxů textu v obraze se v datasetu pohybuje přes 400. Tohle je počet kolikrát musíme rozšířit mapu vlastností, aby bylo možné transformaci pro všechny boxy zpracovat paralelně. Tím dostaneme tenzor o velikosti 12 GB. To je velikost, která se na standardní grafické karty<sup>2</sup> nevejde, a to nepočítáme parametry sítě jako jsou

<sup>2</sup>Mezi standardní grafické karty nejsou počítány high-end karty jako je NVidia Volta a podobné

váhy, pomocné tenzory atp. Při inferenci a použití větší obrázku může síť dosahovat mnohem větších velikostí. Proto jsou v modulu všechny boxy transformovány sekvencně. Avšak tento přístup je však více časově složitější, a výsledné trénování, případně inferenze je velmi pomalá.

## 6.6 Rozpoznání textu

Posledním krokem algoritmu je rozpoznání textu ze získaných regionů z předchozího kroku. Každý tento region obsahuje část sdílené mapy vlastností, která koresponduje danému boxu slova, jež chceme rozpoznat. Zároveň s regiony jsou vyžadovány i šířky těchto regionů, protože jsou regiony zarovnány do šířky nejdelšího. Zpracování těchto regionů probíhá v několika sekcích nejprve je na regiony aplikována sekvencní konvolucní síť (podobně jako při extrakci mapy vlastností, viz 6.3). Dále je výsledek konvolucní sítě transformován na sekvenci, která je pomocí obousměrné LSTM zakódována. Výstupy směru jsou zkombinovány a pro jednotlivé úseky sekvence je provedena klasifikace do zvolené množiny znaků. Sekvence hodnot klasifikací je dekodována pomocí CTC a výsledkem je predikovaný text. Schéma tohoto modulu můžeme pozorovat na obrázku 6.11.



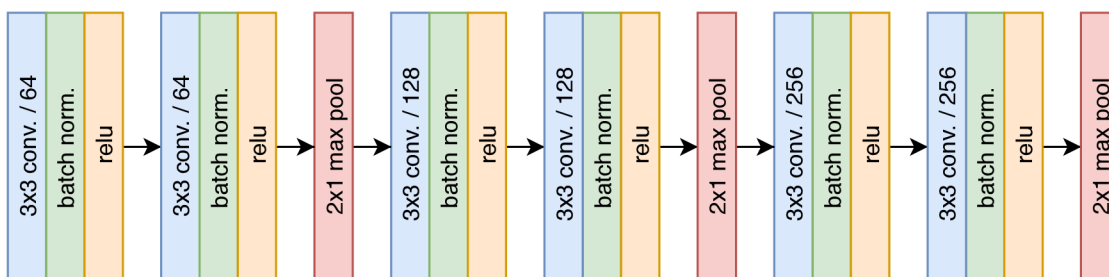
Obrázek 6.11: Model rozpoznání textu

### 6.6.1 Redukce vlastností regionu

Z předchozích kroků jsou získány regiony s variabilní šířkou, avšak pro další zpracování je potřeba tento region zredukovat do 1dimenzionální sekvence vektoru, která je potřeba pro použití rekurentní sítě. Chceme však zachovat šířku regionu. Proto na výsledné regiony je

aplikována konvoluční síť podobná VGG, která zredukuje rozmery těchto regionů a získá z nich vlastnosti vyšší úrovně.

Tato konvoluční síť se skládá ze tří vrstev, které sdílí stejnou velikost kanálu. A to konkrétně 64 kanálů v první vrstvě, 128 ve druhé a 256 ve třetí. Každá z těchto vrstev se skládá ze dvou operací konvoluce s jádrem velikosti 3. Za každou operaci konvoluce následuje operace normalizace dávky jejíž výsledek je aktivovaný operací ReLU. Na konci každé z těchto tří vrstev následuje operace max-pool. Avšak oproti síti VGG, je operace max-pool nastavena na agregaci hodnot pouze ve vertikální ose. Tím je zaručeno, že po provedení konvoluční sítě se výška regionu zredukuje a původní šířka zůstane zachována. Výsledkem této vrstvy je tenzor  $\mathbb{R}^{C \times W'}$ , jež je dále pomocí LSTM zakódován. Model této konvoluční sítě je možné pozorovat na obrázku 6.12.



Obrázek 6.12: Model CNN v rámci modulu rozpoznání textu.

## 6.6.2 Použití rekurentní sítě

Tenzor získaný pomocí konvoluční sítě  $L \in \mathbb{R}^{C \times W'}$  můžeme permutovat do sekvence ve tvaru  $l_1, l_2, \dots, l_{W'} \in \mathbb{R}^C$ . Podíváme-li se na tuto sekvenci jako na časovou řadu, pak je možné chápat jednotlivé hodnoty v casech této řady jako vývoj spektra, které dohromady tvoří obraz daného slova. Tento přístup je používán například i při rozpoznání zvuku, avšak rozpoznání slov z obrazu je daleko složitější, protože může obsahovat daleko více proměnných. Mezi ne patří například nastavení a parametry fontu jimž je text vyobrazen, kontrasty barev pozadí vůči textu atd. Dále se na získané sekvence aplikuje rekurentní neuronová síť, která má za úkol naučit se sekvenci replikovat. A to způsobem že získá ze sekvence širší kontext struktury, jakým způsobem jsou jednotlivé časové kroky na sobě závislé. K těmto účelům byla v práci využita obousměrná rekurentní síť LSTM, o jejíž principu i výhodách bylo psáno v kapitole 2.4.6.

Použitá rekurentní síť LSTM je implementována za pomoci knihovny PyTorch a vyžaduje na vstupu tenzor rozmeru uvedených výše. Jelikož jsou všechny textové boxy variabilní, ale jsou zarovnané do stejné velikosti, nedává smysl určovat hodnoty sekvencí pro časové body, které nenáležejí vstupu. Řešením je zabalit výše uvedenou sekvenci pomocí metody `pack_padded_sequence`. Tato metoda vyžaduje na vstupu serazené všechny boxy sestupně podle jejich délky. Proto je potřeba předpřipravit pole indexu, které odpovídají požadované serazené posloupnosti. Pole indexu je vhodné díky tomu, že se data nachází v dvou různých tensorech a je možné to využít na serazení obou zářáz. Takové serazené pole dosáhneme pomocí metody `arg_sort`. Zabalенý tenzor poté předáme obousměrné vrstvě LSTM, která je nastavená na 256 výstupních kanálů. Tím získáme dvě sekvence: jedna kóduje vstupní sekvenci od začátku po konec a druhá od konce po začátek. Každá se zakódovaných sekvencí obsahuje 256 kanálů pro každý bod. Výstupem je ale zabalенá sekvence, kterou musíme opět na základe délek rozbalit. K tomu je použita metoda `pad_packed_sequence`, která z různé

velkých sekvencí vytvoří zpátky tenzor. Obe sekvence (dopředná i zpetná) jsou následně zkombinovány a výsledek je dále klasifikován.

### 6.6.3 Klasifikace a dekódování

Z předchozí části dostaneme sekvence, kde každý bod obsahuje 256 kanálu. Na každý z těchto bodů je aplikována plně propojená vrstva, jejíž počet výstupních elementů je stejný jako velikost množiny znaků, které klasifikujeme (v rámci této práce se jedná o 131 znaků). Množina znaků je definována podle datové sady a obsahuje dva speciální znaky: prázdný znak a neznámý znak. Po této vrstvě je aplikována vrstva Softmax, která převede hodnoty do rozdělení míry pravděpodobností, pro jednotlivé znaky. Hodnotu znaku pro daný časový bod sekvence určíme jako znak s největší mírou pravděpodobnosti.

Sekvenci ještě musíme dekódovat, protože šířka predikovaného boxu, pomocí něhož byl region pro rozpoznání extrahován, nemusí odpovídat i šířce 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 čase. Bez CTC by musel být dataset připraven tak, aby bylo přesně jasné, v jaký časový okamžik bude síť produkovat požadovaný výsledek. Použitím CTC umožníme síť se naučit 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 čistou sekvenci, která poslouží jako výstup. Aby tento výstup byl pro nás užitečný je ještě třeba zpetně přiřadit Unicode hodnoty znakům na základě indexu třídy.

---

**Algoritmus 4:** Dekódování CTC sekvence.

---

**Data:**

*input\_text* - Seznam nejpravděpodobnějšími znaky.

*N* - Délka tohoto seznamu.

**Result:** *decoded\_text* - Výsledný seznam znaků.

```
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 řešení 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 náročné, sníženo na sekvencní zpracování. Jeden z problému už byl zmíněn 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 náročností 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, prve anotovaný box neodpovídá skutečnosti 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 šířky celého obrazu, zpusoboval nadbytečné alokace pameti. Duvodem alokací je, protože všechny boxy, které extrahujeme z obrazu, jsou zarovnány na stejnou šířku v pameti. Pri konvolucní síti uvnitr části s predikci textu nastal tento problém a vždy skoncil pádem z duvodu nedostatku pameti.

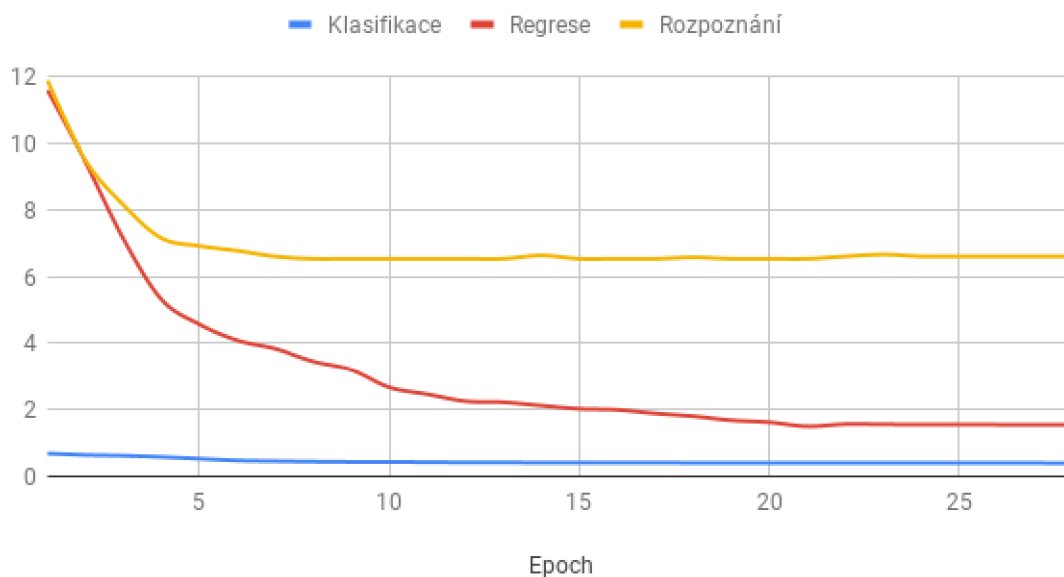
Řešení, nebo vubec zjištění duvodu tohoto problému zabralo hodne casu, protože pri ucení se problém projevil pouze u této jediné položky a dataset je automaticky pro každou epochu náhodne zamíchán. Tato chyba se díky tomu projevovala velmi zřídka, proto jsem celou dobu považoval, že se chyba nachází 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 řešení 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 neúspěšných uceních síte jsem zacal zkoušet další možné řešení. Nekteré operace, na kterých aplikace padla, vytvářely nové tenzory. V části rozpoznání síte, konkrétně 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 řešení problém nevyřešilo tak jsem myslel, že bud jazyk Python nebo knihovna PyTorch neuvolnují dostatek pameti pro další alokace, anebo jej uvolnují na špatném místě. Zkusil omezit náročnost tohoto systému tak, že veškerou nepotrebnou pamet rucne uvolním smazáním reference na danou promennou a následného uvolnění pameti prímým voláním garbage collectoru. Stejne jako pri ostatních řešeních se toto ukázalo být neefektivní a problém stále setrval. Po vycerpání všech možností síte jsem nakonec obrátil pozornost i k datasetu. Až v tento okamžik jsem na anomálii narazil. Problém byl vyřešen vygenerováním nového datasetu a rucním procházením všech vygenerovaných souboru, zda se v nich nenachází také tato anomálie.

## Kapitola 7

# Experimenty

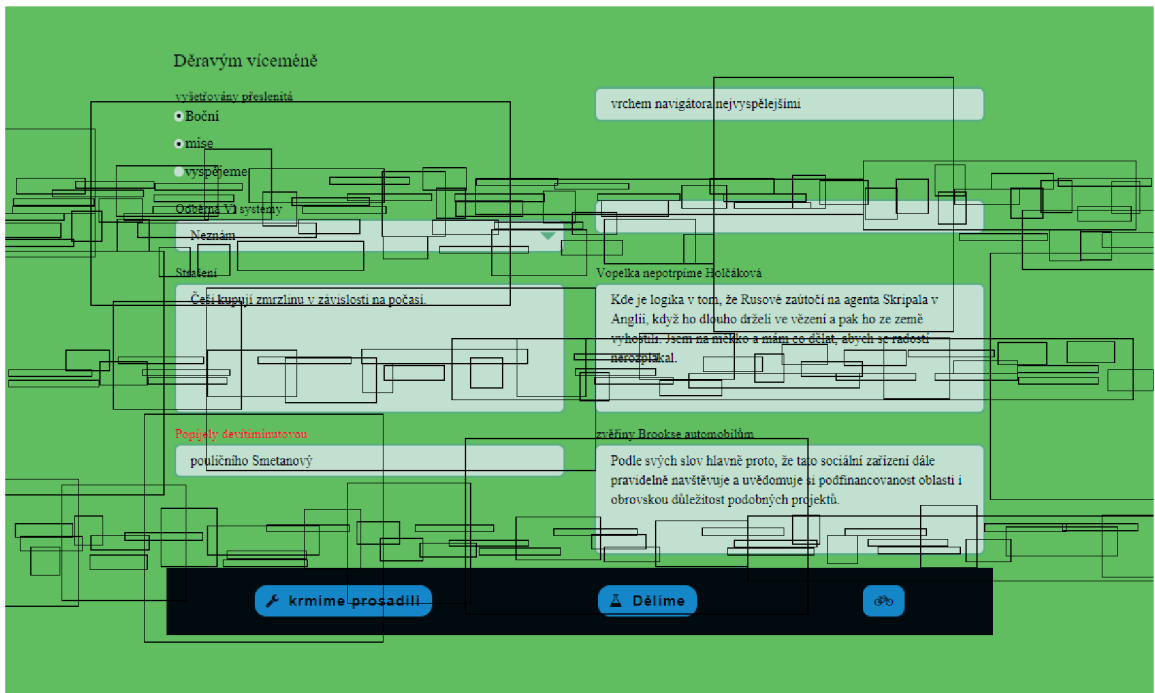
Model síte, který byl predstaven v predchozí kapitole, byl ucen na stroji se dvema grafickými kartami NVidia GTX-1080. Jako ucební vzor byl vygenerován pomocí nástroje „gui-generator“ dataset s obrázky rozmeru 1200x800 obsahující více než 500 obrázku. Tento model je však velmi výpocetne nárocný a délka 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 síť do epochy 20 konvergovala a poté se optimalizace síte zastavila a poté kmitala okolo bodu.

### Error funkce Klasifikace, Regrese a Rozpoznání

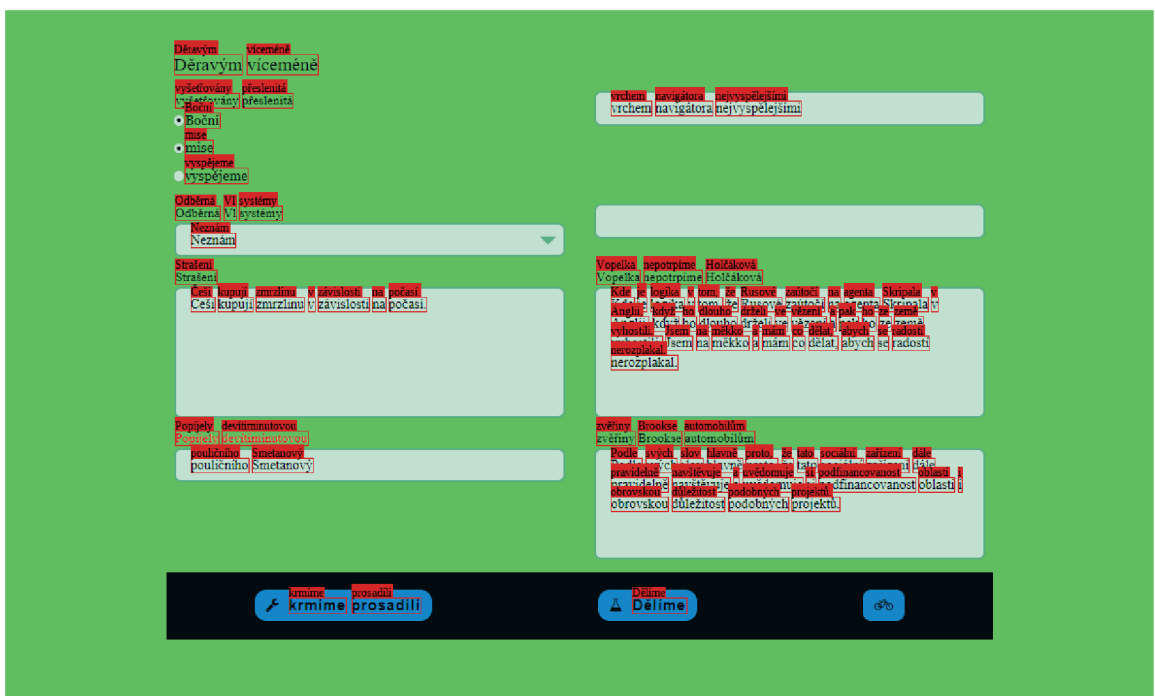


Obrázek 7.1: Znázornění prubehu loss funkcí pri trénování.

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 sadě.

Problémy tohoto systému tkví pravděpodobně v nedostatku času na učení systému. Možným dalším důsledkem jsou špatně zvolené počáteční váhy systému, případně rozlišení bounding boxu. Kvůli vyprodukovaným výsledkům implementovaného systému, nebyl systém porovnán vůči ostatním metodám.

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

Tabulka 7.1: Vyhodnocení OCR

OCR	Metrika znaku		Metrika slov		Úplná metrika	
	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%

Nejdůležitějším měřením je citlivost znaku, protože chceme detekovat všechny texty datové sady, a toho dosáhneme při citlivosti 100%. Můžeme pozorovat, že obě služby správně rozpoznaly více než 80% znaku datové sady. Všechny ostatní znaky jsou buď chybně klasifikovány, nebo je algoritmy nedetekovaly vůbec. Co se týče presnosti, vidíme že u obou algoritmu přibližně 87% korektních predikcí. Ostatní mohou být neshody znaku, či predikce mimo textové oblasti. Podíváme-li se na metriku slov, vidíme, že služba Microsoft Azure OCR (dále jen Azure OCR) správně rozpoznala 79,5% anotovaných slov datové sady. Ale také můžeme pozorovat podle úplné metriky, že míra rozpoznání slov i se segmentací je 79,38%. Rozdíl těchto dvou hodnot znáčí, že segmentace slov služby Azure OCR je velmi podobná té, která je použita v datové sadě.

Služba Google Cloud Vision (dále jen Google Vision) má výrazně vyšší míru rozeznání slov vůči testované datové sadě oproti službě Azure OCR. To naznačuje že Google Vision „vidí“ více slov datové sady, což můžeme pozorovat i na citlivosti znaku. Ovšem rozdíl mezi úplnou metrikou a metrikou slov naznačuje, že Google Vision využívá jinou metodu segmentace obrazu, než je použita v datové sadě. Tedy pokud by se segmentace změnila 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 včetně zamerení na metody hlubokého ucení. Byly prozkoumány metody rozpoznání textu z obrazu, které využívají právě neuronové sítě. Aplikace gui-generátor byla rozšířena o širší možnosti generování různých grafických webových rozhraní a posloužila pro generování datové sady. Byla vytvořena datová sada obsahující přibližně 1900 obrázku uživatelských rozhraní včetně jejich anotací. Ty jsou určeny k validaci výsledku a poslouží i jako datová sada pro ucení sítě. Byl implementován program pro vyhodnocení kvality algoritmu OCR na základe zvolené metriky. Pomocí tohoto programu a výše vytvořené datové sady byly jednotlivé služby ohodnoceny definovanou metrikou rozpoznání textu a jejich výsledky dále prozkoumány.

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

Dalším možným vývojem systému je alespon měsíční trénování sítě. Nebo možným urychlením je implementace funkcí jako je například 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

Projekt lze vytvořit virtuální prostředí python, kde všechny závislosti projektu jsou staženy pomocí příkazu: `pip install -r requirements.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.