VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYUŽITÍ HLUBOKÉHO UČENÍ PRO ROZPOZNÁNÍ TEXTU V OBRAZU GRAFICKÉHO UŽIVATELSKÉHO ROZHRANÍ

DEEP LEARNING FOR OCR IN GUI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

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

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

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

Zadání diplomové práce

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

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

Literatura:

- http://www.deeplearningbook.org/

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. července 2019
Datum odevzdání: 31. července 2019
Datum schválení: 29. července 2019

Abstrakt

Optické rozpoznání znaků (OCR) je již mnoho let oblastí zájmu. Je definováno 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, který je schopen detekovat a rozpoznat text v grafických uživatelských rozhraních.

Abstract

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

Klíčová slova

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

Keywords

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

Citace

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

Využití hlubokého učení pro rozpoznání textu v obrazu grafického uživatelského rozhraní

Prohlášení

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

.....
Pavel Hamerník
31. července 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 i svým nejbližším přátelům za jejich podporu, motivaci a korekturu při psaní práce.

Obsah

Kapitola 1

Úvod

Optické rozpoznání znaků (Optical Character Recognition dále jen OCR) je software který dokáže převést naskenovaný tištěný text a fotografie obsahující text do digitalizované podoby, se kterou může být manipulováno počítačem. Lidský mozek dokáže velmi jednoduše a bez námahy rozpoznat text z obrázku. Algoritmy u kterých chceme, aby dospěli stejného výsledku, musí být však dostatečně flexibilní, aby dokázaly kvalitně vnímat a zpracovat informace z obrazu. Proto bylo vynaloženo velké úsilí na vytvoření software, který se snaží transformovat obraz dokumentu do podoby, která je srozumitelná pro počítač. OCR je velmi rozsáhlý problém kvůli velkému množství jazyků, stylů 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. Systém využívá algoritmy hlubokých neuronových sítí, které detekují a rozpoznají text v obraze. V kapitole 2 jsou představeny současné 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 algoritmů. Dále je v kapitole 4 popsán nástroj a jeho rozšíření, které bylo využito pro zhotovení datové sady určené k trénování modelů neuronových sítí. Kapitola 5 obsahuje návrhy neuronových sítí, které jsou hlavním algoritmem pro detekci a rozpoznání textu z obrazu. V kapitole 6 je popis jeho implementace a trénování tohoto algoritmu. Výsledky a zhodnocení algoritmu se nachází v kapitole **??**.

Kapitola 2

Neuronové sítě

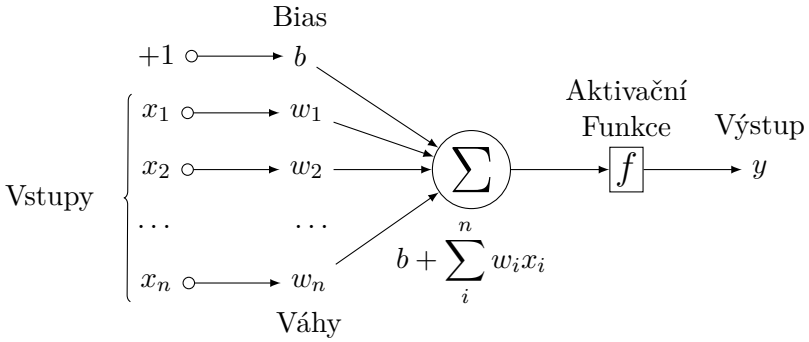
2.1 Architektury sítí

Neuronová síť je výpočetní model používaný v umělé inteligenci. Vzorem neuronových sítí je biologický model lidského mozku. Neuronové sítě se používají pro řadu úkolů, jako jsou detekce a rozpoznání, regrese veličin, aj. S příchodem principu hlubokého učení nabyly neuronové sítě na velké popularitě. Lze pomocí nich řešit složité úkoly, jako je rozpoznání objektů v obraze a predikce vývoje hodnot sekvence. S tímto přístupem je možno vytvářet modely neuronových sítí, které mají mnoho úrovní. Díky tomu však není snadné určit, co síť dělá v dané vrstvě a proč. Proto se s hlubokými sítěmi pracuje jako s tzv. *černou skříňkou*, neboť nás zajímá nás pouze, že pro zadaný vstup produkuje požadovaný výstup. Jak však síť tohoto výsledku dosáhne, necháváme na optimalizačních algoritmech.

V následujících podkapitolách je ukázáno, jak pracuje klasická neuronová síť (kap. 2.1.1). Kapitola (2.1.2) popisuje konvoluční neuronové sítě určené pro efektivnější zpracování většího objemu dat, jakými jsou například obrázku. V kapitole 2.3 jsou ukázány aktivační funkce používané v neuronových sítí.

2.1.1 Klasické neuronové sítě

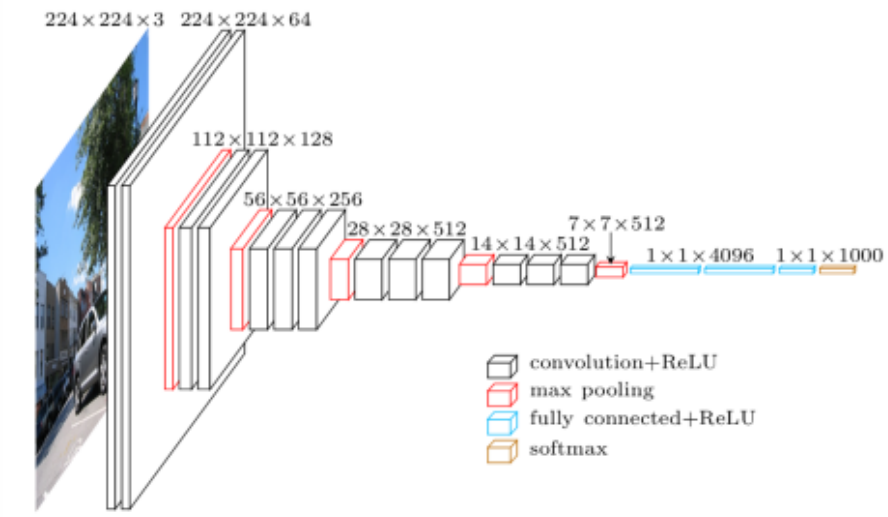
Základním stavebním kamenem softwarových neuronových sítí je neuron (dříve nazývaný perceptron). Každý neuron se skládá ze tří hlavních částí. Vstupní část využívá všechny vstupní signály, které putují do neuronu. Každý z těchto vstupů je ohodnocen váhou, která slouží k posílení, nebo potlačení hodnoty daného vstupu. Další částí je tělo neuronu, které agreguje všechny hodnoty vstupů, včetně jejich vah do jediné hodnoty. Na tuto hodnotu se aplikuje aktivační funkce neuronu a výsledek je výstupem celého neuronu [6]. Výpočetní model neuronu je možné vidět v obrázku 2.1.



Obrázek 2.1: Výpočetní model neuronu.

2.1.2 Konvoluční neuronové sítě

Konvoluční sítě jsou speciálním druhem vícevrstvých neuronových sítí. Používají operace konvoluce v lokálním okolí pro získání vizuálních vzorů přímo z pixelů obrazu. Vychází z principu neocognitronové síte, která byla poprvé představena v 80. letech K. Fukushimou. Neocognitron využívá velké množství neuronových vrstev a obsahuje variabilní propojení mezi buňkami sousedních vrstev. Cílem této sítě je identifikace objektů podle jejich obecné podobnosti. Typická konvoluční síť je vyobrazena na obrázku 2.2 a je možné pozorovat, že se skládá z mnoha různých vrstev. Dalším charakteristickým znakem konvolučních sítí je získávání vlastností obrazu z recepčních polí, sdílení vah v rámci konvoluční vrstvy a prostorové vzorkování [6].



Obrázek 2.2: Ilustrace konvoluční neuronové sítě. Zdroj [1]

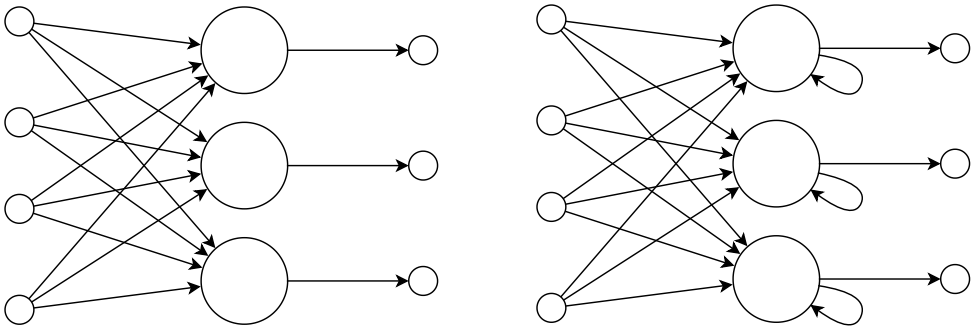
2.1.3 Rekurentní neuronové sítě

Rekurentní neuronové sítě (dále jen RNN) pochází ze stejné doby jako mnoho dalších algoritmů hlubokého učení. Poprvé byly představeny v 80. letech, ale jejich plného potenciálu

[1] Dostupné z `https://blog.heuritech.com`

nebylo donedávna možné využít zejména kvůli vysoké výpočetní náročnosti. Výrazem rekurentní neuronové sítě označujeme dvě široké kategorie s podobnou obecnou strukturou. Jedna z nich je síť s konečnou odezvou. Tento typ sítě lze vyjádřit pomocí orientovaného acyklického grafu, který je možné rozbalit, a díky tomu je celou síť možné nahradit dopřednou sítí. Rekurentní síť s nekonečnou odezvou lze reprezentovat pouze orientovaným cyklickým grafem, který není možné redukovat na dopřednou sít, protože dojde k zacyklení [11].

RNN využívají interní paměť, která jim umožňuje si zapamatovat informace z posloupnosti přijatých vstupů. Díky tomu dokážou být velmi přesné v predikci následujících hodnot. Z tohoto důvodu jsou preferovanou sítí při práci se sekvenčními daty (například časová posloupnost zvukových dat), protože ze zpracovávaných dat dokáží porozumět hlubšímu významu jejich posloupnosti oproti ostatním algoritmům.



(a) (b)

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

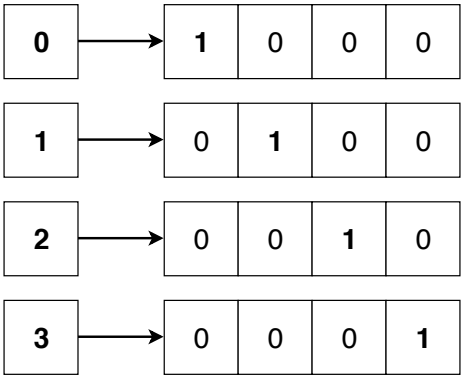
2.2 Kódování

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

2.2.1 One-hot kódování

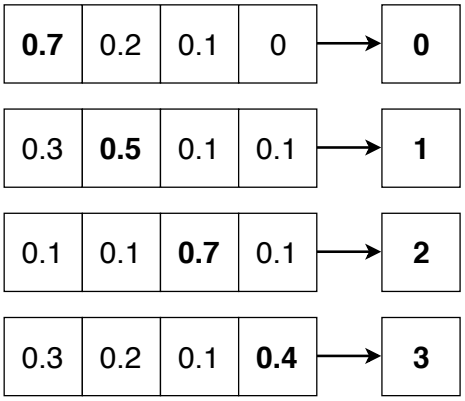
One-hot[2] kódování (v angličtině se také nazývá dummy variable) vyjadřuje pomocí hodnoty 0 nebo 1 absenci nebo přítomnost jisté kategorie. Toto kódování je užitečné zejména pro rozdělení dat do vzájemně se vylučujících kategorií (například pes, kočka). Na hodnoty zakódovaných kategorií se můžeme také dívat jako na míru pravděpodobnosti výskytu jednotlivých kategorií. Z tohoto důvodu je toto kódování hojně používáno komunitou zabývající se strojovým učením, a to zejména v oblasti klasifikace objektů, protože umožňuje neomezený počet klasifikačních tříd. Příklad zakódování informace do vektoru míry pravděpodobnosti můžeme pozorovat v obrázku 2.4.

[2]V české literatuře se také můžeme setkat s názvem kódování 1 z N.



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 sítě a jeho dekódování můžeme pozorovat na obrázku 2.5.



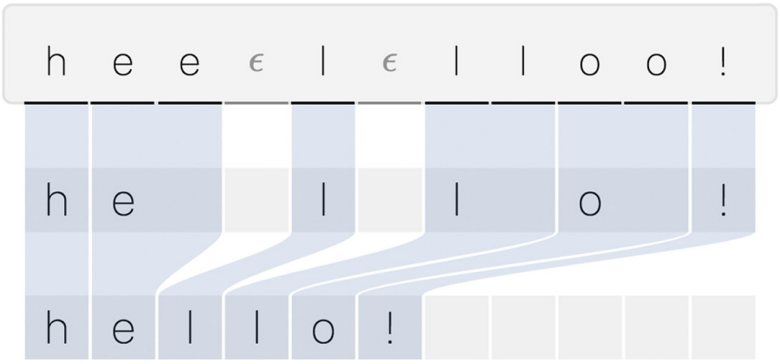
Obrázek 2.5: Ukázka dekódování one-hot výstupu pro 4 třídy.

2.2.2 Temporální klasifikace

V angličtině označována *Connectionist temporal classification* je velmi užitečnou operací používanou zejména v rekurentních neuronových sítích [7]. Jedná se o způsob, jak naučit neuronovou síť predikovat požadovanou sekvenci. Počet predikovaných kroků může být však větší, než je počet kroků na požadovaném výstupu. Proto se využívá tato metoda, která umožňuje dosáhnout požadovaného výstupu pomocí mnoha cest. Na výstupu nás však zajímá pouze požadovaná sekvence s daným počtem kroků. Obsahuje-li sekvence více po sobě jdoucích časových kroků se stejnou hodnotu, pak se všechny tyto kroky shrnou do jediného. Aby však bylo možné zakódovat dvě stejná po sobě jdoucí písmena, přidává CTC do slovníku speciální prázdný znak nazývaný blank. Tento znak nemá žádnou hodnotu a ve výsledném textu je ignorován. Jeho užitek však spočívá v tom, že slouží jako oddělovač stejných symbolů, které by jinak byly agregovány do jediného. 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"`.

Princip CTC dekodéru je možné pozorovat na obrázku 2.6, kde je vybrána cesta pro 11 prvků, která dekóduje sekvenci „hello!“.



Obrázek 2.6: Princip CTC dekodéru.

2.3 Aktivační funkce

Aktivační funkce je hlavní složkou každé neuronové sítě. Je aplikována na agregované hodnoty vstupů neuronu a slouží jako hlavní výstup neuronu, kde jej nazýváme aktivací neuronu. Tato funkce může být lineární i nelineární ale často je žádanou vlastností i diferencovatelnost. Ta umožňuje nalézt směr růstu této funkce (gradient funkce). Gradient je význačný při učení neuronové sítě, kdy hodnoty vah upravujeme právě po směru gradientu. Aktivaci neuronu můžeme obecně zapsat následující rovnicí:

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right), \tag{2.1}$$

kde parametr w_i reprezentuje váhu jednotlivých vstupů x_i a parametr b (anglicky bias) se využívá k rozšíření rozhodovacího rozsahu, kterého neuronová síť může dosáhnout. Funkce f je aktivační funkce a y je výstup neuronu.

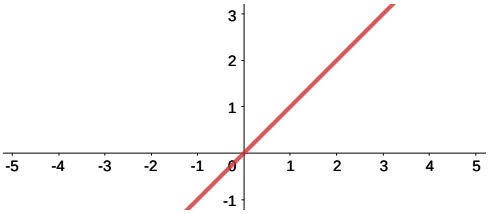
Pro snadnější počítání s parametrem bias, se často rozšiřuje vektor vstupu o konstantní hodnotu 1 (bývá označováno jako x_0), a do vektoru vah je na odpovídající pozici (dle pozice ve vstupu) přidán parametr b. Díky této úpravě je rovnice výstupu neuronu zjednodušena do podoby využívající operaci skalárního součinu:

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

kde w je vektor vah, x je vektor vstupů neuronu, f je obecná aktivační funkce a y je výstup neuronu.

Standardně, není-li uvedeno jinak, má neuron lineární aktivační funkci identita. Tato funkce ovšem nijak nepomáhá se složitostí parametrů, které jsou posílány neuronovým sítím. Proto je použití nelineárních funkcí vhodnější. Lineární funkce má však své využití, protože nijak neomezuje výstup. Její využití je výhodné zejména pro úlohy regrese. V moderních neuronových sítích se často používá jako implicitní aktivace každé neuronové vrstvy, a komplexnější aktivační funkce jsou poté aplikovány jako další vrstva v sérii. Rovnici a graf této funkce je možné sledovat v obrázku 2.7.

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

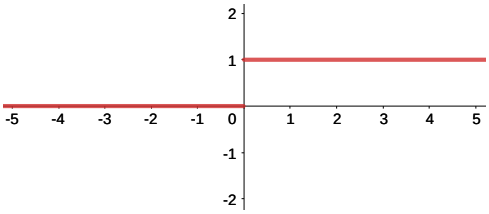


Obrázek 2.7: Aktivační funkce identita

Pro klasifikační úlohy jsou odnedávna využívány skokové 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 dvou hodnot. Je vhodná pouze pro jednoduché úlohy, protože není diferencovatelná. Z tohoto důvodu není vhodná pro použití ve vícevrstvých neuronových sítích. Rovnici a průběh můžeme vidět v obrázku 2.8.

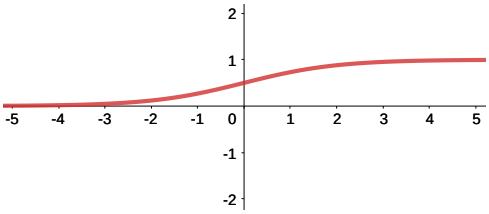
$$f(x) = \begin{cases} 1, & \text{pokud } x > 0 \\ 0, & \text{jinak} \end{cases} \qquad (2.4)$$



Obrázek 2.8: Skoková aktivační funkce

Sigmoidální funkce (také logistická funkce) převádí vstupní reálné hodnoty do intervalu $(0, 1)$. Tato funkce je monotónní a diferencovatelná proto 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 při učení neuronové sítě může nastat taková situace, při níž se výsledek modelu dále nezlepšuje. Důvodem toho je absence záporných hodnot na výstupu funkce. Na obrázku 2.9 můžeme pozorovat rovnici i průběh funkce f v okolí počátku souřadné soustavy.

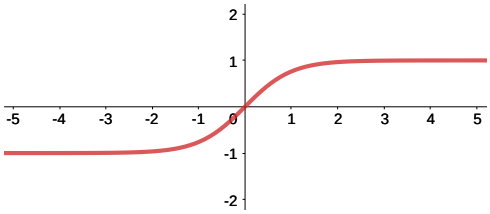
$$f(x) = \frac{1}{1 + e^{-x}} \qquad (2.5)$$



Obrázek 2.9: Aktivační funkce sigmoida.

Hyperbolický tangent je podobný sigmoidální funkci. Na rozdíl od ní však může nabývat i záporných hodnot, což je výhodou při učení sítě. Má také větší rozsah hodnot a to hodnoty v intervalu $(-1, 1)$. Využívá se zejména pro binární klasifikaci a pro predikci více tříd naráz. Níže v obrázku 2.10 je uvedena rovnice i průběh této funkce.

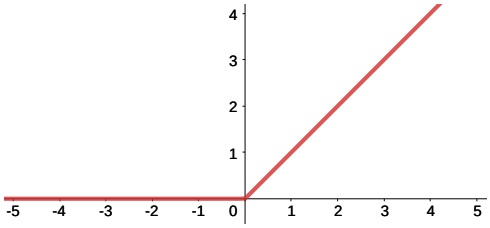
$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.6}$$



Obrázek 2.10: Aktivační funkce hyperbolický tangent.

V současné době je nejpoužívanější aktivační funkcí takzvaná Rectified linear unit (dále jen ReLU), pro kterou neexistuje vhodný český překlad. Tato funkce se stala velmi oblíbenou díky svým vlastnostem a snadné implementaci prahu s hodnotou 0. Funkce je monotónní stejně jako i její derivace a pro kladné hodnoty není nijak omezena. Díky tomu se u neuronových sítí zvyšuje rychlost učení. V obrázku 2.11 vidíme tuto funkci i její graf.

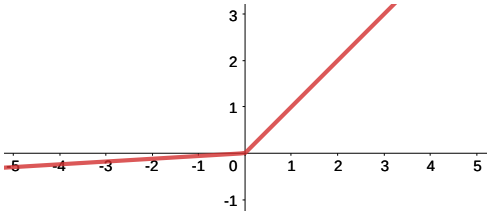
$$f(x) = \begin{cases} x, & \text{pokud } x \geq 0 \\ 0, & \text{jinak} \end{cases} \tag{2.7}$$



Obrázek 2.11: Aktivační funkce ReLU.

Problémem ReLU je však, že všechny záporné hodnoty jsou mapovány do nuly. Nulové hodnoty jsou v učení potlačovány, protože jejich derivace je nulová a díky tomu není určen směr změny. Toto má za následek snížení schopnosti neuronové sítě se efektivně naučit daný problém řešit. Jedním z řešení tohoto problému je funkce Leaky ReLU. Tato funkce je totožná s ReLU, avšak na rozdíl od ní mapuje všechny záporné hodnoty pomocí lineární funkce s velmi malým sklonem. Díky tomu nejsou záporné hodnoty potlačeny a je možné pro ně určit gradient. Tuto funkci i její graf můžeme vidět v obrázku 2.12.

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



Obrázek 2.12: Aktivační funkce Leaky ReLU.

2.4 Vrstvy neuronových sítí

V této sekci jsou uvedeny vrstvy neuronových sítí jak jsou používány v moderních neuronových sítích. Uvedené vrstvy jsou použity v rámci této práce.

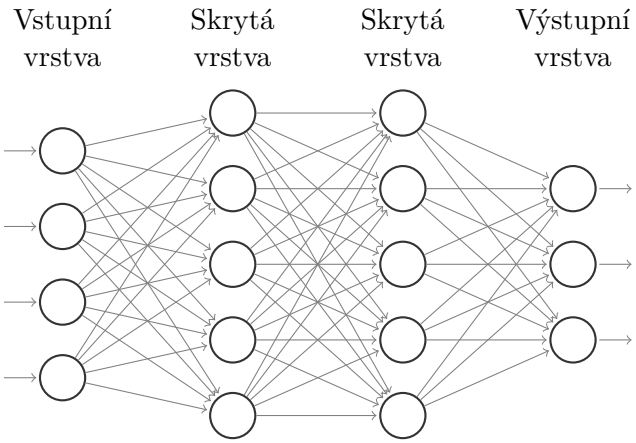
2.4.1 Plně propojená vrstva

Plně propojená vrstva, v anglické literatuře se můžeme setkat s názvem *fully-connected* nebo také *dense layer*, je vrstvou, která se skládá z N neuronů, které jsou plně propojeny s

M neurony z předcházející vrstvy. Je možné tuto vrstvu definovat následující rovnicí:

$$y_j = \sum_{i=0}^{M} w_{ij} x_i, \tag{2.9}$$

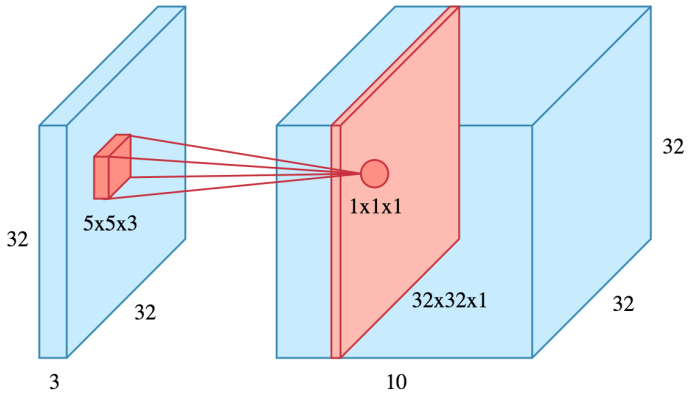
kde y_j udává hodnotu j-tého výstupního neuronu ($1 \leq j \leq N$). x_i jsou vstupní hodnoty neuronu z předchozí vrstvy, w_{ij} je hodnota váhy j-tého neuronu pro i-tou vstupní hodnotu. Výše uvedená rovnice počítá s hodnotou bias, který je často ve vstupním vektoru neuronové sítě označován jako x_0. Tuto vrstvu je možné pozorovat na obrázku 2.13.



Obrázek 2.13: Model sítě s plně propojenými vrstvami.

2.4.2 Konvoluční vrstva

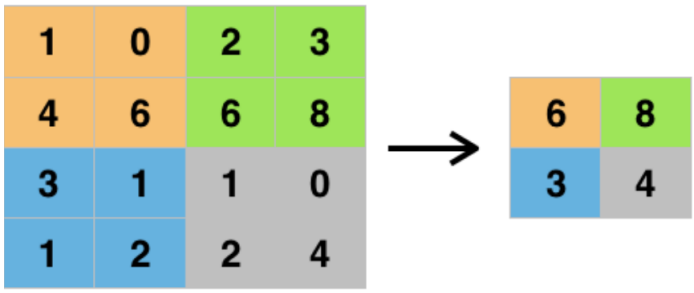
Tato vrstva využívá konvoluční filtry, jejichž parametry jsou pro celou vrstvu sdílené. Tyto filtry pokrývají pouze malou část plochy vstupních dat, ale využívají celou jejich hloubku. Na hodnotu získanou konvolucí je aplikována aktivační funkce, která je výstupem tohoto filtru pro jeden neuron. Konvoluční vrstva vyžaduje dva hlavní parametry, kterými jsou vstupní hloubka a výstupní hloubka. Každá z výstupních hloubek obsahuje rozdílně nastavené konvoluční filtry a dohromady produkují výstup, který má právě hloubku uvedenou výstupní hloubkou. Dalším důležitým parametrem této vrstvy je nastavení velikosti konvolučního okna (kernel size), které zpravidla nabývá pouze lichých hodnot. Dále je parametrem prostorový krok (spatial stride) udávající počet prvků, o které je posunuto konvoluční okno. A posledním důležitým parametrem je padding, který umožňuje rozšířit okraje vstupu, aby se například při konvoluci zachovala původní velikost vstupu. Existuje mnoho technik jak stanovit hodnoty okrajových oblastí, ale nejpoužívanější je okraje naplnit hodnotou 0, tzv. *zero padding*. Konvoluční vrstva je znázorněna na obrázku 2.14, kde můžeme pozorovat vstupní hloubku 3 a výstupní hloubku 10.



Obrázek 2.14: Konvoluční vrstva aplikovaná na 2d data. Zdroj [3]

2.4.3 Pooling vrstva

Slouží pro zmenšení velikosti dat vstupní vrstvy. Tato vrstva se využívá zejména mezi konvolučními vrstvami. Toto vede ke snížení celkového počtu všech parametrů výsledného modelu sítě. Pooling se aplikuje po jednotlivých vstupních hloubkách. Podobně jako u konvoluce se data prochází oknem, které však na všechny hodnoty obsažené v daném okně aplikuje některou z agregačních funkcí. Nejpoužívanější funkcí je maximum, která z okna vybere tu nejvyšší hodnotu. Často se můžeme setkat například s výrazem *max-pooling*, což je pooling vrstva používající právě funkci maximum. Stejně jako u konvoluční vrstvy má tato vrstva parametry velikost okna a prostorový krok. Na obrázku 2.15 je znázorněn princip pooling vrstvy s funkcí maximum, který vstup 4x4 transformuje na 2x2.



Obrázek 2.15: Ukázka operace max-pooling. Zdroj [4]

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 při trénování počítá statistiky o středních hodnotách a rozptylech, které jsou aplikovány při inferenci sítě. V praxi se tato vrstva používá mezi vrstvami aplikujícími váhy na vstupy, protože vrstvy budou zpracovávat

[3] Dostupné z `https://towardsdatascience.com`
[4] Dostupné z `http://deepai.org`

data se stejným rozložením. Každá vstupní data této vrstvy jsou zpracována následujícími rovnicemi:

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

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

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

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

kde μ značí střední hodnotu dat a σ^2 jejich rozptyl, x_i jsou vstupy vrstvy, y_i jsou výstupy vrstvy a γ, β jsou parametry, které se tato vrstva učí.

Použití této vrstvy snižuje účinek problému zvaného *internal covariance shift*. Tento problém nastává posunem vstupní distribuce dat při učení neuronové sítě . V případe hlubokých sítí je vstup v každé vrstvě ovlivněn váhami této vrstvy. I ty nejmenší změny vah sítě se průchodem sítí zesílí a v konečném důsledku způsobí posuny vstupních distribucí vnitřních vrstev sítě, což může vést ke snížení konvergence sítě.

2.4.5 Softmax vrstva

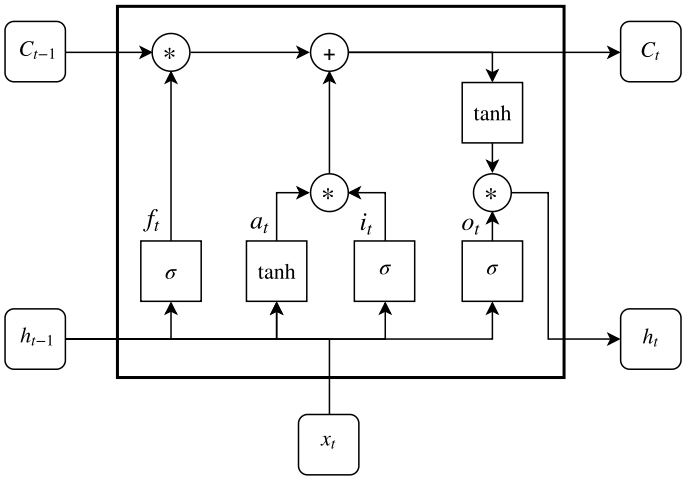
Tato vrstva slouží k transformaci vstupu vrstvy do výsledného rozložení pravděpodobnosti. Součet všech výsledných hodnot této vrstvy nabývá hodnoty 1. Využívá se pro klasifikaci mnoha navzájem výlučných tříd, kdy je pravdivá pouze jediná třída a to ta s nejvyšší hodnotou pravděpodobnosti. Počet vstupů a výstupů této vrstvy je shodný a jejich počet je dán předchozími vrstvami. Hodnotu jednotlivých výstupů je možné určit následující rovnicí:

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

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

2.4.6 Long Short-Term Memory vrstva

Long Short-Term Memory, dále jenom LSTM, je typ rekurentní neuronové sítě, která postupným zpracováním sekvence modifikuje interní paměť. Paměť LSMT díky tomu může existovat delší dobu. Její architektura se skládá ze zřetězených buněk, kde každá zpracovává jeden bod sekvence. Buňku LSTM můžeme pozorovat na obrázku 2.16, kde vidíme, že se skládá z několika hradel, které napomáhají této vrstvě udržovat si interní paměť. Funkci LSTM buňky můžeme rozdělit do tří částí:



Obrázek 2.16: Schéma LSTM buňky

Zapomenutí. Využívá hradlo *forget gate*, které umožňuje LSTM buňce se rozhodnout jaké vlastnosti mají být *zapomenuty* (odtud název tohoto hradla). Hradlo získá hodnoty, které značí míru zapomenutí paměti z předchozího stavu. Vstupem tohoto hradla jsou data z předchozího stavu a současný zpracovávaný vstup buňky. Toto hradlo využívá funkci sigmoida, které vrátí hodnotu v intervalu $(0, 1)$ a je určena následující rovnicí:

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

kde W_{xf} jsou váhy vstupu, W_{hf} jsou váhy předchozího stavu, h_t je výstup LSTM v bodě t, x_t je vstup v bodě t, b_f je bias tohoto hradla, σ je funkce sigmoida, f_t je vypočtená hodnota hradla v bodě t.

Zapamatování. Umožňuje LSTM buňce zapamatovat si nové vlastnosti ze současného vstupu. Využívá k tomu dvou hradel. Aktivační hradlo[5] získá vektor nových vlastností, které chceme přidat k současnému stavu. Zapisovací hradlo (*input gate*) rozhoduje, které hodnoty budou aktualizovány s jakou mírou (podobně jako forget gate výše).

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

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

kde W_{xa}, W_{xi} jsou váhy aplikované na vstup pro aktivační resp. zapisovací hradlo. W_{ha}, W_{hi} jsou váhy aplikované na předchozí stav pro aktivační resp. zapisovací hradlo, b_a a b_i jsou biasy hradel, x_t je vstup v bodě t. a_t je hodnota aktivace v bodě t a i_t je hodnota zapisovacího hradla v bodě t. Hodnoty obou hradel jsou posléze zkombinovány a přidány k novému stavu. Společně s hodnotou forget gate tvoří nový stav daný rovnicí:

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

kde a_t je hodnota aktivace v bodě t, i_t je hodnota zapisovacího hradla v bodě t, f_t je hodnota forget gate v bodě t a C_t je interní stav v bodě t. Operace $*$ znázorňuje Hamadarmův součin, který je aplikován na dva vektory. Jeho výsledkem je vektor, jehož hodnoty vznikly násobením po složkách z obou vstupních vektorů.

[5] V literatuře bývá označováno jako gate.

Výstup. Výstup LSTM je založený na současném stavu spočítaném výše a na hodnotě vstupu. Využívá hradlo *output gate*, které stejně jako u *forget gate* určí hodnoty jež mají být ve výstupu zachovány a s jakou měrou. Hodnotu *output gate* lze spočítat rovnicí:

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

kde W_{xo} jsou váhy aplikované na současný vstup, W_{ho} jsou váhy aplikované na předchozí stav, b_o je bias tohoto hradla, x_t je současný vstup v bodě t, h_t je interní stav v bodě t a o_t je hodnota hradla v bodě t. Vypočtená hodnota a nově vypočtený stav (na který je aplikován hyperbolický tangent) jsou spojeny dohromady a udávají novou výstupní hodnotu buňky. Tato hodnota je stanovená rovnicí:

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

kde C_t je aktuálně vypočtený stav buňky, o_t je vypočtená míra pravděpodobností jednotlivých bodů výstupu, operace $*$ je Hamadarmův součin mezi vektory a h_t je výstupní hodnota dané buňky.

Postupně jsou jednotlivé hodnoty vstupní sekvence propagovány LSTM buňkami. Díky tomu získáme predikovanou sekvenci, která se snaží co nejvíce napodobit vstup na kterém byla naučena. Počáteční stav se zpravidla nastavuje na vektor samých nul. LSTM je rekurentní síť s konečnou odezvou a je navržená tak, aby se gradient mohl šířit sítí bez velkých změn. Proto jsou potlačeny účinky problémů *vanishing* i *exploding* gradient, které můžeme často pozorovat v obecných rekurentních sítích. Oba tyto problémy se projevují ve vícevrstvých sítích aplikací aktivačních funkcí, které výstupní hodnotu potlačí, případně zesílí. Účinek těchto problémů je tím více znatelný z čím více vrstev se síť skládá.

2.5 Učení sítí

Hlavním účelem učení neuronové sítě je nastavení hodnot váhových parametrů mezi neurony 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 nejlepších výsledků.

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 tak, aby na podobné vstupy reagovala podobnými výstupy.

2.5.1 Algoritmus zpětného šíření

Nebo-li algoritmus *backpropagation* je algoritmus učení neuronové sítě s učitelem, který slouží k adaptaci neuronové sítě na danou trénovací množinu. Algoritmus pracuje ve třech krocích. Nejprve získáme dopředným šířením sítí výsledky pro daný vstup. Výstup neuronové sítě porovnáme s očekávaným výstupem a stanovíme chybu sítě (angl. loss). Nakonec vypočtenou chybu zpětně propagujeme sítí a upravujeme váhové parametry jednotlivých neuronů. Počáteční hodnoty váhových parametrů jsou zpravidla stanoveny náhodně, a postupným trénováním se nastaví tak, aby síť byla schopna správně reagovat na zadaný vstup. Chybu sítě na základě vypočtených výstupů můžeme vypočítat rovnicí:

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

kde y_i je výstup pro testovací data i a t_i je očekávaný korektní výsledek, E je vypočtená hodnota chyby. Metod pro vypočtení chyby je mnoho a každá z nich se používá k jiným účelům. Metody použité v této práci jsou zmíněny v kapitole 2.5.2.

Metoda klesání podle gradientu

Gradient descent je nejjednodušší optimalizační algoritmus který slouží k úpravě váhových parametrů neuronových sítí. Pracuje na principu iterativní minimalizace loss funkce za pomocí prvních derivací. Funkce se zmenšuje podle nejstrmějšího klesání (nejvíce záporného gradientu), které vypočítáme za pomocí:

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

kde E je vypočtená chyba, w_i jsou váhové parametry pro i-tý neuron poslední vrstvy. Operátorem ∇ označujeme výpočet gradientu.

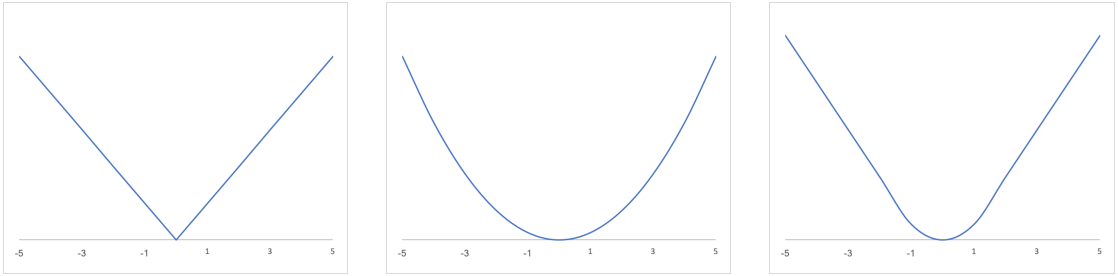
Metoda gradient descent je omezena na hledání lokálních minim, čímž může dojít k minimalizaci chyby, ale algoritmus nemusí získat nejlepší řešení. Po průchodu celé testovací množiny dojde k přenastavení váhových parametrů a iterativně dochází ke snižování chyby, kdy se snažíme dosáhnout nulového gradientu ($\nabla E = 0$). Parametry jsou iterativně upraveny následovně:

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

kde w_i značí váhové hodnoty v kroku i, Δw_i značí změnu váhových hodnot, w_{i+1} jsou vypočtené váhové hodnoty pro další krok algoritmu. Parametr η reprezentuje rychlost učení (angl. learning rate), který definuje velikost kroku jednotlivých derivací.

2.5.2 Chybová funkce

Chybová funkce, též označována také jako cenová funkce i loss funkce, matematicky vyjadřuje odchylku dvou řešení a je klíčovým prvkem v učení neuronových sítí. Účelem chybové funkce je stanovit chybu neuronové sítě pro zadaný vstup. Tato chyba bude následně použita optimalizačními algoritmy pro úpravu váhových parametrů neuronové sítě, tak aby se chyba minimalizovala. V této práci jsou ukázány nejvíce používané chybové funkce, které jsou použity i v této práci. [6].



(a) L1 (b) L2 (c) L1 smooth

Obrázek 2.17: Grafy regresních chybových funkcí

L1 metrika

L1 metrika, neboli *least absolute deviation* (zkr. LAD, či L1 loss) je chybová funkce nejčastěji používána v úlohách lineární regrese. Je vypočtena jako součet všech rozdílů mezi správným výstupem a výstupem získaným z neuronové sítě. Matematicky tuto funkci lze vyjádřit rovnicí:

$$L1\_loss = \sum_{i=1}^{N} |y_i - t_i|, \tag{2.24}$$

kde y_i je výstup predikovaný neuronovou sítí a t_i je očekávaný vzorový výstup i-tého výstupního neuronu. Průběh této funkce můžeme pozorovat na obrázku 2.17a.

Hlavní vlastností této funkce je robustnost. Díky ní dokáže pracovat s datovými body, které se výrazně liší od všech ostatních (tzv. *outlier data*). Nevýhodou této funkce je však vysoká nestabilita, protože funkce může obsahovat více řešení. Tím dochází k tomu, že se gradient může výrazně změnit i pro malé chybové hodnoty.

L2 metrika

Funkce se také nazývá *least square error* (zkr. LSE, či L2 loss), u které již název napovídá, že využívá druhou mocninu rozdílu vypočteného a požadovaného výstupu na rozdíl od L1 funkce. Stejně jako u L1 se nejčastěji používá pro regresní úlohy. Graf této funkce je možné vidět na obrázku 2.17b a rovnicí lze vyjádřit jako:

$$L2\_loss = \sum_{i=1}^{n} |y_i - t_i|^2. \tag{2.25}$$

kde y_i je výstup predikovaný neuronovou sítí a t_i je očekávaný vzorový výstup i-tého výstupního neuronu.

Tato funkce není velmi robustní a nedokáže si velmi dobře poradit s velkým množství *outlier dat*. Díky tomu velké rozdíly mezi daty ještě více umocní. Ovšem oproti L1 je tato funkce stabilní, protože existuje právě jedno řešení.

Smooth L1 metrika

Smooth L1 metrika, neboli Huberova chybová funkce je kombinací L1 a L2, která není příliš náchylná na *outlier data* oproti *L2 Sooth loss* metrice. Funkce využívá pro výpočet malých odchylek L2 metriku a pro vetší hodnoty využívá L1 metriku, čímž zamezí výraznému zvětšení chyby při výpočtu *outlier dat* jako u L2. Funkci můžeme zapsat podmíněnou rovnicí:

$$L1\_smooth(a) = \begin{cases} \frac{1}{2}a^2 & \text{pokud } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta) & \text{jinak,} \end{cases} \tag{2.26}$$

kde a je rozdíl vypočtených a očekávaných dat a δ je parametr stanovující hranici, od které je hodnota vypočtena kvadratickou nebo lineární metrikou. Graf funkce s parametrem δ nastaveným na hodnotu 1 lze vidět v obrázku 2.17.

Cross-Entropy Loss

Cross-entropy loss, také nazývaný log loss, je používaný pro měření chyby neuronových sítí při klasifikačních úlohách. Výstup této neuronové sítě obsahuje hodnoty pravděpodobností v intervalu od 0 do 1. Hodnota chybové funkce se zvyšuje, právě když se predikovaná

pravděpodobnost vzdaluje od skutečného cíle. Tato chybová funkce je často používaná na klasifikaci výstupů softmax vrstvy (viz. 2.4.5). Cross entropy loss lze spočítat rovnicí:

$$H(P,Q) = -\sum_{i}^{N} \chi_P(i) \log\left(Q(i)\right) \tag{2.27}$$

kde P je množina, která obsahuje hodnotu indexu i očekávané klasifikační třídy. Operace χ_P značí charakteristickou množinu, jejíž hodnota je 1, pokud se parametr i nachází v množině P, jinak jejím výsledkem je hodnota 0. Množina Q obsahuje predikované hodnoty pravděpodobností klasifikačních tříd.

Kapitola 3

Technologie zpracování textu

Rozpoznání textů dělíme na dvě kategorie, a to rozpoznání psaného textu a rozpoznání tisknutého textu. Rozpoznání ručně psaného textu je velmi obtížný úkol. Důvodem toho jsou odlišné styly psaní i pohyby při psaní stejných písmen každého člověka. Tyto algoritmy dále dělíme na *online* a *offline*. Online detektory dokáží rozpoznat psané texty v reálném čase. Nebývají zpravidla příliš komplexní, protože zpracovávají data ze temporálních nebo časových dat jako jsou například rychlostí psaní, počtu tahů perem atp. Naopak offline algoritmy analyzují text z předložených obrazů.

Rozpoznání tisknutého textu se skládá z kategorie algoritmů které nazýváme *Optical Character Recognition* (zkr. OCR). Tyto algoritmy umožňují automatizaci mnoha procesů na které by byl potřeba člověk. Rozpoznání textu je úloha kterou zpravidla rozdělujeme na dvě zcela odlišné pod-úlohy, které spolu kooperují. Tyto úlohy můžeme charakterizovat jako detekce, která v zadaném obraze nalezne všechny oblasti v nichž se nachází text, a rozpoznání, které oblasti zanalyzuji a převedou text kterých se v nich nachází do digitální podoby. Metody pro detekci i rozpoznání často využívají různé techniky z počítačového vidění. V poslední době se však přibylo algoritmů využívající hluboké neuronové sítě. Tyto algoritmy oproti klasickým dosahují vyšších rychlostí rozpoznání textů i větší kvality výsledků. [15, 17, 2]

Různé používané algoritmy obou kategorií jsou představeny v následujících podkapitolách. Dále je stanovena metrika pro měření kvality algoritmů provádějících rozpoznání textů. Pomocí této metriky jsou v rámci práce vyhodnoceny všechny algoritmy.

3.1 Metody detekce textu

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 celku jako jsou slova případně řádky textu. Hlavními dvěma přístupy zástupci 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.

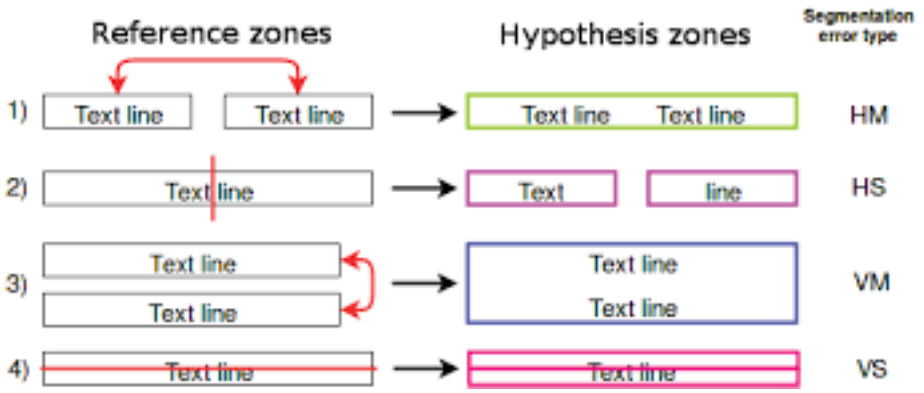
3.2 Metody rozpoznání textů

Cílem rozpoznání textu je dekódovat sekvenci značek z pravidelně oříznutých textových obrázku s různou délkou textu. Většina metod nejprve zachycuje jednotlivé znaky a nekorektně

klasifikované znaky jsou podle slovníku později opraveny. Kromě přístupu po jednotlivých znacích existují přístupy zaměřené na regiony textu. Ty můžeme rozdělit na tři kategorie: klasifikace slov (word classification), metoda dekódování sekvence na značky (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 velmi flexibilní a rozšiřitelný. Zbylé dvě metody využívají rekurentních neuronových sítí.

3.3 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 textu v těchto regionech. Každý algoritmus pracuje jiným způsobem a může nastat situace kdy je text rozdělen do více oblastí (pojem *over-segmentation*) anebo naopak několik segmentu textu se spojí do jednoho celku (pojem under-segmentation). Uvedené segmentační problémy mohou nastat jak v horizontálním, tak i ve vertikálním směru a znesnadňují vyhodnocení výsledků algoritmů. Na obrázku 3.1 můžeme pozorovat zmíněné problémy segmentace.



Obrázek 3.1: Segmentační problémy textu v obraze. Zdroj: [12]

3.3.1 Levenštejnova vzdálenost

Levenštejnova vzdálenost (také známa jako editační vzdálenost, anglicky levenshtein distance či *edit distance*) je metrika pro určení podobnosti dvou seznamu. Podobnost měříme jako vzdálenost, s jakou můžeme upravit seznam tak, aby byl totožný s porovnávaným seznamem. Hodnota vzdálenosti se skládá ze součtu cen jednotlivých operací. Operacemi jsou přidaní prvku (*insertion*), smazaní prvku (*deletion*) a nahrazení prvku (*substitution*). Každá z těchto operací může mít jinou hodnotu ceny za provedení. Nejjednodušším a nejčastější implementací je stanovení hodnoty 1 ke všem cenám.

Výpočet Levenštejnovi vzdálenosti můžeme vyjádřit následujícím vztahem:

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

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

3.3.2 Metrika ZoneAltCnt

Jedná se o metriku kompletního porovnání kvality algoritmu rozpoznání textu. Metodu navrhl R. Karpinsky [12] a udává, jak vyhodnotit systémy OCR i s problémy segmentace textu, jež jsou uvedeny výše. Algoritmus nejdříve seskupí všechny predikované a vzorové anotace podle jejich vzájemného překryvu. Výpočet můžeme zaměřit na 3 kategorie, podle úrovně komplexity.

- Metrika znaků (Character Metric) udává, jak kvalitně dokáže algoritmus rozpoznat text nezávisle na segmentaci obrazu.
- Metrika slov (Word Metric) říká, jak kvalitně je segmentován obraz i rozpoznán text v těchto segmentech. Tato metrika je použita jako hlavní způsob porovnání.
- Úplná metrika (Strict Word Metric) slouží k úplnému porovnání segmentace i textu. Čím dosahuje lepších výsledků tím více jsou porovnávané prvky identické.

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

Pro každou z těchto metrik jsou směrodatné dvě hodnoty. Jedná se o míru citlivosti a přesnost naměřených hodnot. Citlivost (anglicky *recall*) je poměr mezi korektně vyhodnocenými objekty a celkovým počtem vzorových objektu. Hodnota udává poměr pokrytí všech testovaných vzorových objektu. Zatímco přesnost (anglicky precision) je poměr mezi počtem všech korektně vyhodnocených objektu a celkovým počtem všech predikovaných objektu. Čím vyšší přesnost tím méně nekorektně predikovaných objektu.

3.4 Existující řešení

Tesseract

Tesseract[1] 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 výsledku přesnosti. Později roku 2005 byl Tesseract firmou HP uvolněn zdrojový kód do veřejnosti.

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

Detekční algoritmus Tesseractu je složen z několika částí. Nejdříve 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 řádku se určí pozice slov dle horizontálních vzdáleností jejich jednotlivých znaků. Slova jsou poté rozdělena na samostatné znaky. Klasifikátor pak určuje jednotlivé znaky a detekuje jejich přibližný geometrický obrys.

Microsoft Computer Vision

Jedná se o komerční služby firmy Microsoft, které jsou poskytnuty aplikačním webovým rozhraním. [2]. Jednou z poskytovaných služeb je detekce objektu, které dokáže rozpoznat informace nacházející se v obrázku a označit je patřičnými značkami současně s pravděpodobností dané značky. Další službou je rozpoznání textu, které detekuje a klasifikuje slova z obrázku do digitalizované sekvence znaků.

Protože jsou tyto služby komerční je potřeba vlastnit účet u společnosti Microsoft který umožňuje přístup k těmto službám. Detekci textu je provedena na serverové straně zasláním dotazu obsahujícím obrázek pro vyhodnocení. Odpověď serveru je ve formátu json a obsahuje detekované řádky textu a v rámci nich i jednotlivá slova náležící danému řádku. Navíc tyto záznamy obsahují i souřadnice kde se v obrázku nachází text. Doba inference služby je kvůli síťové komunikaci se vzdáleným serverem a odesílání nemalých dat na něj pomalejší ale pohybuje se přibližně kolem 1 vteřiny.

Algoritmus využitý pro rozpoznání textu není zveřejněn, ale podle získaných výsledků můžeme usoudit že zahrnuje některou z těchto operací nalezení řádku textu, natočení řádků do roviny, rozdělení řádků na slova a klasifikace slov nebo jednotlivých znaků slova.

Google Cloud Vision

Poskytuje několik možností integrace natrénovaných modelů do programů. Hlavním způsobem, jak takovou službu integrovat je za pomocí aplikační rozhraní Cloud Vision(Cloud Vision API[3]). Toto rozhraní poskytuje koncové body a odpovědi pro jednotlivé definované moduly a probíhá pomocí webového protokolu HTTP. Cloud Vision poskytuje mnoho služeb mezi které patří detekce objektů v obrazu, vyhledávání obrazu na webu (také na základe podobnosti obrázku) a také rozpoznání textu z obrazu.

Podobně jak je tomu u Microsoftu (viz výše) je potřeba uživatelského účtu pro využití produktů Google. Detekce textů z obrazu poskytuje výsledek, který je ve formátu json. Algoritmus detekce rozdělí obraz na hierarchii zanořených objektů jejíž nejnižší článek je symbol. Symboly jsou sdruženy do slov, slova do paragrafů atd. Ze struktury výstupu můžeme usoudit že tato služba využívá modely natrénované strojovým učením. Obraz se postupně segmentuje modely různých úrovní dokud neskončí na symbolech, které jsou klasifikovány.

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

Kapitola 4

Datová sada

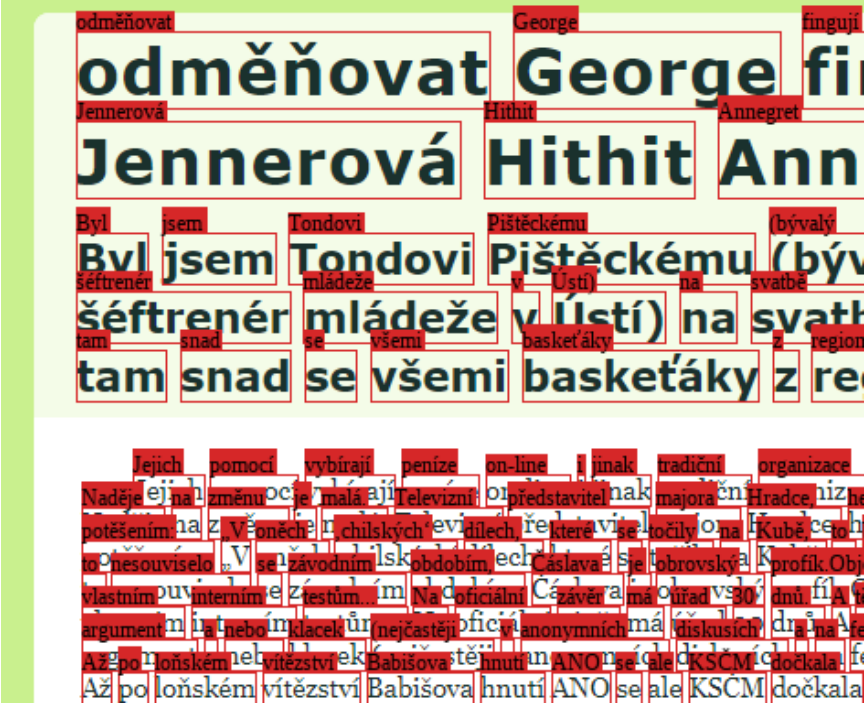
Datová sada byla vytvořena pomocí interního nástroje firmy ARTIN zvaného „gui-generator“. Nástroj, jak již název vypovídá, dokáže generovat grafické webové rozhraní v mnoha frameworcích. Webové stránky jsou vykreslované pomocí webových prohlížečů a byly optimalizovány zejména pro prohlížeč Google Chrome [1]. Pomocí generátoru je možné vytvořit široké spektrum webových stránek s různými barevnými schématy, rozloženími prvku a množstvím textu. Hlavní účelem gui-generátoru je generování trénovacích datových množin pro rozpoznání aktivních prvku grafických uživatelských rozhraní. S každým vygenerovaným rozhraním vytvořeny i anotace pro jednotlivé prvky nacházející se v obrazu. Anotace zpravidla obsahují informace o třídě daného prvku (například tlačítko, hypertextový odkaz, slovo atd.) a souřadnice rámečku ohraničující tento prvek v obrazu. Pro účely této práce byl tento generátor rozšířen o nové funkce, které jsou popsány detailněji v následujících podkapitolách. Výsledek produkovaný vylepšeným generátorem je možné vidět na obrázku 4.1. Obrázek je přiblížen oproti původní verzi a obsahuje bounding boxy, které jsou znázorněny červeným rámečkem. Toto je ukázka dat, které jsou použity jako učební vzor pro neuronové sítě.

4.1 Texty

Původní verze generátoru využívala pouze malého vzorku textových dat, který neměl příliš velkou rozmanitost slov. Při jejich použití jako učebních vzorů by se síť nenaučila je korektně generalizovat. Proto byl generátor rozšířen o schopnost generovat texty s větším rozsahem a větší rozmanitostí slov. Nové texty pro generátor byly sestaveny z různých českých textu posbíraných z internetu. Získané texty obsahují 120466 vet, 167589 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 dostačující 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 nestylovaných prvků

[1] Dostupné z `http://www.google.com`



Obrázek 4.1: Ukázka vygenerovaných dat.

„span“, s jejíž pomocí je možné určit souřadnice oblastí ve kterých 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 různorodost generovaných textu v snímkách byl přidán do generátoru náhodný výběr ze širokého seznamu písem. Důvodem této změny bylo otestování rozpoznávací schopností pro různorodé styly textu. Pro každou generovanou stránku jsou náhodné zvoleny písma pro nadpisy, tlačítka a všeobecný text. Zároveň při tom je i náhodně nastavena velikost tohoto písma, vzdálenost mezi jednotlivými znaky a výška řádku textu. Pro ještě větší diversitu a jsou náhodně specifikovány atributy jako sklon písma, tučnost písma, i jiné.

Pro účely této práce byly vyměněny generované texty současných stylu. Avšak texty mohli nabývat větších velikosti (oproti původnímu stavu), byly některé styly změněny tak, aby změněný text neposunul výsledné rozložení komponent. Také byl pro generátor vytvoření styl, který obsahuje pouze text. Tento styl generuje stránku v podobě připomínající internetový článek a skládá se z hlavních částí: hlavička článku a tělo článku. Hlavička článku obsahuje hlavní nadpis a menší podnadpis, který slouží jako stručný úvodní text článku. Tělo článku pak obsahuje paragrafy článku a mohou se v něm vyskytovat i obrázky. Každé z výše uvedených polí je náhodně naplněno texty. Barevné rozložení těchto stylu je generováno tak, aby pro zvolenou barvu pozadí byl text čitelný.

4.3 Generování testovací datové sady

Datová sada je získána Pro automatické generování datové sady byl vytvořen nástroj, jenž automaticky ovládá webový prohlížeč. Pomocí něho je načtena stránka vygenerovaná gui-generátorem a za pomocí aplikačního rozhraní jsou staženy bounding boxy komponent na zobrazené stránce. Obrázek stránky je získán pomocí snímku vykreslené webové stránky ovládaným webovým prohlížečem. Ovládání webového prohlížeče je dosaženo pomocí knihovny Selenium[2], jehož hlavním použitím jsou automatizované testy webových stránek. Struktura, formátu json, vygenerované anotace pro snímky lze pozorovat níže:

```
{
  "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"
    },
    ...
```

v tomto formátu vidíme hlavní parametry které jsou: název(*filename*), cesta(*filepath*) a rozměry obrázku(*image_size*). Tomuto obrázku odpovídá seznam anotací, jejíž jednotlivé položky se skládají z typové třídy (*type*), textové hodnoty a bounding boxu, který je definován dvěma krajními body.

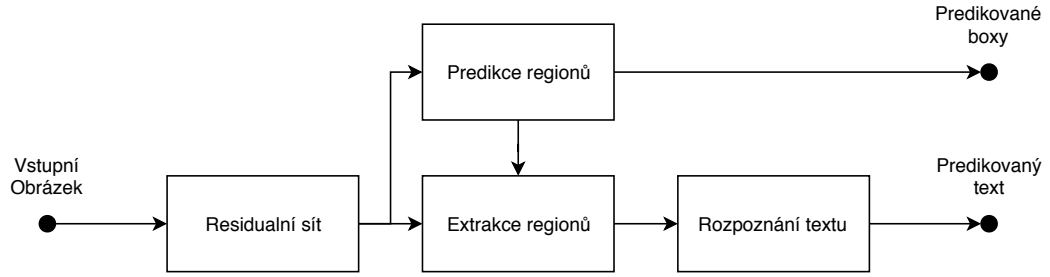
[2]Dostupné z `https://www.seleniumhq.org/`

Kapitola 5

Návrh řešení

Navržený systém se skládá ze dvou modelů. Jeden model slouží k detekci a regresi bounding boxů obsahujících text, dále jen *model detekce*. Rozpoznávací model je určený k rozpoznání textu v předložené oblasti. Oba tyto systémy spolu kooperují a tvoří systém OCR, který dokáže rozpoznat text v jemu předloženém obraze. Tyto moduly byly inspirovány architekturou Fast Oriented Text Spotting [15], avšak na rozdíl od této architektury, která využívá společnou síť jak pro detekci tak i pro rozpoznání textu, nemá mnou navržená architektura společnou konvoluční síť pro extrakci příznaků.

Celkovou architekturu systému můžeme pozorovat na obrázku 5.1. V dalších podkapitolách jsou navržený systém i jeho části představeny detailněji.



Obrázek 5.1: Celková architektura modelu sítě.

5.1 Model detekce textu

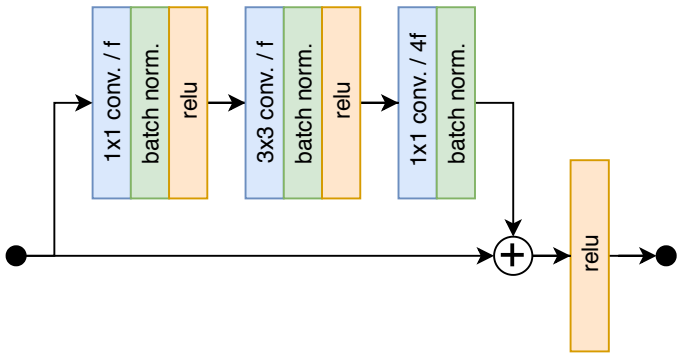
Detekční model se skládá z kombinované reziduální konvoluční sítě ResNet50 [8] a sítě Feature Pyramid Network, které z obrazu získají ty nejpodstatnější příznaky, které jsou využity pro detekci textu. Tento výstup je využit pro regresní modul, jež pro každý prostorový bod mapy příznaků rozhoduje zda se v daném bodě nachází text. Společně s tím jsou určeny nelineární transformační koeficienty, kterými transformují pevně stanovený bounding box (anchor box) daného prostorového bodu, aby se co nejvíce přiblížil textu. Tento model funguje jako samostatná jednotka, která slouží k nalezení textu v obraze, a je možné jej dále propojit.

5.1.1 Extrakce příznaků

Tato část slouží získání mapy všech možných příznaků ze vstupního obrazu, které mohou být dále použity ostatními moduly. Standardním návrhem tohoto modulu bývá použití re-

ziduální sítě. Reziduální síť je konvoluční síť výhradně určená k extrakci příznaků z obrazu. Použití této sítě má za výhodu, že je možné nalézt natrénované modely na datové sadě ImageNet[3]. Jedná se o datovou sadu, která obsahuje miliony různých obrázku v odlišných kategoriích. Použití takto natrénovaných vah je vhodné, protože výsledná síť bude rychleji konvergovat. Tato praktika je v současné době běžně používána a nazýváme ji transfer learning [17]. Umožňuje se sítím učit nové úkoly snadněji, protože využívají předešlou znalost se zpracováním obrazu.

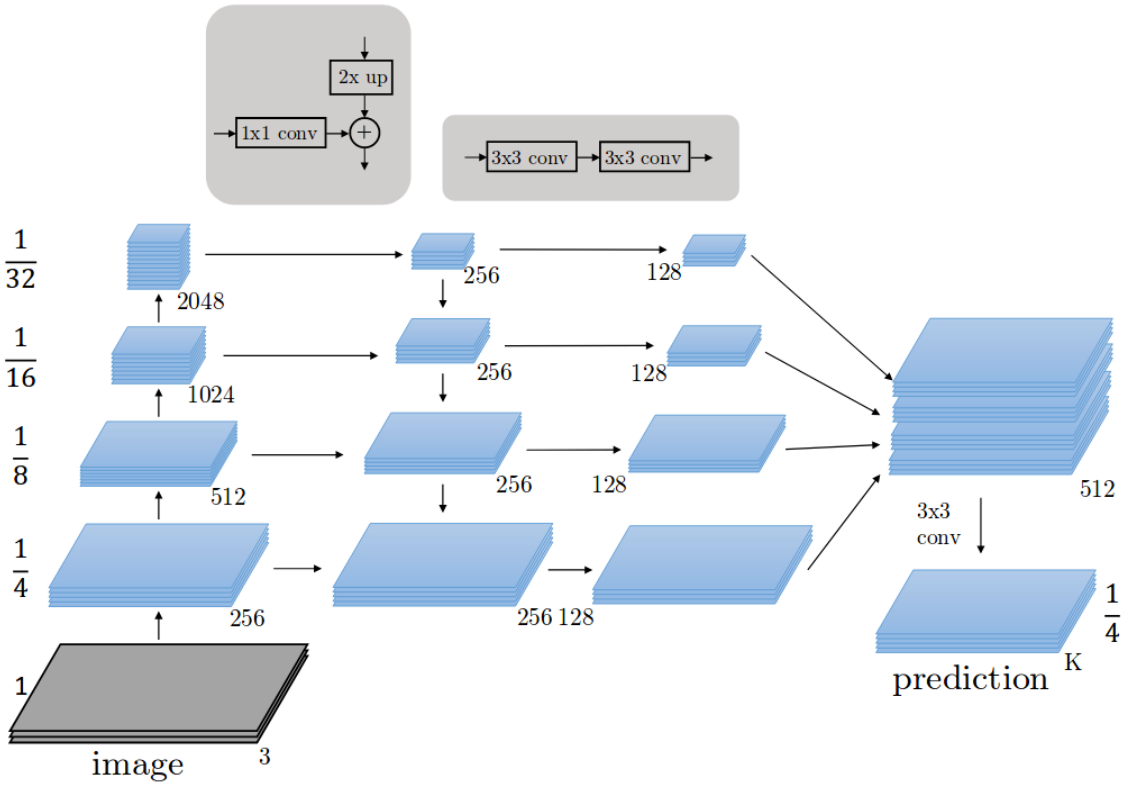
V rámci této práce je reziduální síť navržena po vzoru ResNet-50 [8]. Síť se skládá z takzvaných reziduálních bloku, které umožňují vstupním datum projít skrz síť bez velkých změn. Díky tomu se touto sítí může dobře šířit gradient. V každém z těchto bloku existuje propojení, které umožní vstupním hodnotám přeskočit konvoluční operace. Reziduální bloky se také liší v poctu kanálu, které produkují. V obrázku 5.2 můžeme pozorovat schéma reziduálního bloku.



Obrázek 5.2: Residuální blok síte ResNet-50

Síť Res-Net-50 využívá tyto bloky a jsou pomocí nich vytvořeny vrstvy. Jsou celkově 4 výstupní vrstvy této sítě, kde každá z nich 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. Ve standardní verzi je výstupem poslední vrstva, která obsahuje velmi vysoký prostorový krok. Tento fakt ovšem není příliš užitečný pro detekci textu. Z toho důvodu jsou využity výstupy všech vrstev a dohromady zkombinovány do jediného.

Všechny vrstvy jsou nejprve standardizovány na stejnou hloubku dat, aby je bylo možné kombinovat. Vrstvy jsou dále postupně zkombinovány s vrstvou nižší úrovně. Aby je bylo možné zkombinovat je potřeba vrstvy nižší vrstvy zvětšit pomocí bilineární interpolace. Na každou z těchto vrstev jsou provedeny konvoluční operace, které nakonec všechny vrstvy normalizují na hloubku 128. Všechny tyto vrstvy jsou poskládány dohromady a jsou zvětšeny, aby odpovídali velikosti největší z nich.



Obrázek 5.3: Feature pyramid network vycházející z ResNet. Zdroj: [1]

Výsledkem tohoto modulu je mapa příznaků obsahující 512 kanálu pro každý její prostorový bod. Velikost této mapy je 4x zmenšená oproti velikosti vstupního obrazu. Ze získané mapy příznaků jsou dále predikovány bounding boxy.

5.1.2 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.

Anchory

V rámci této práce je potřeba predikovat bounding boxy, které obsahují text. Avšak bounding boxy mají různé rozměry, jejich počet i pozice jsou také proměnné. Tyto vlastnosti mohou vést ke špatné konvergenci učeného systému. Proto byly bounding boxy reprezentovány způsobem, který není závislý na rozměru boxu v daném predikovaném bodě. Řešením je předem stanovit bounding boxy jejichž počet a rozměry je konstantní pro každý pixel mapy příznaků. Liší se pouze v posunu pro jednotlivé pixely mapy. Na tyto konstantní bounding boxy, dále jen *anchory*, aplikovat transformace, které jsou invariantní vůči pozici predikce. Tento přístup je inspirován dle publikace [21].

[1] Dostupné z `http://presentations.cocodataset.org/COCO17-Stuff-FAIR.pdf`

Každý z těchto boxů je specifikován čtyřmi hodnotami, které určují souřadnice anchoru. Anchor je definován pomocí pozice středu, jeho šířky a výšky. Transformace probíhá pomocí čtyř transformačních koeficientů, které určují míru posunu středu a změnu měřítka šířky a výšky. Rovnice pro transformaci anchor boxu jsou uvedeny níže:

$$r_x = a_x + a_w d_x, \tag{5.1}$$

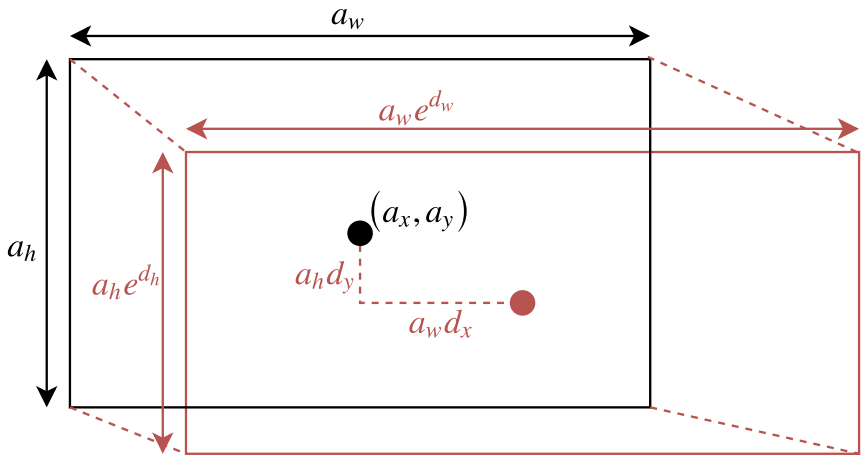
$$r_y = a_y + a_h d_y, \tag{5.2}$$

$$r_w = a_w e^{d_w}, \tag{5.3}$$

$$r_h = a_h e^{d_h}, \tag{5.4}$$

kde čtveřice souřadnic (a_x, a_y, a_w, a_h) jsou souřadnice anchor boxu, čtveřice hodnot (d_x, d_y, d_w, d_h) jsou transformační koeficienty a čtveřice souřadnic (r_x, r_y, r_w, r_h) reprezentují výsledný transformovaný bounding box.

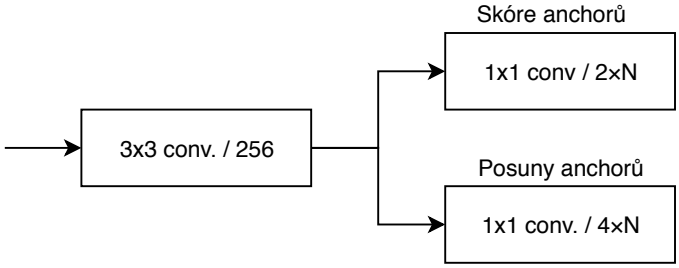
Takto definovanou transformaci je možné snadno učit, protože hodnoty transformačních koeficientů jsou invariantní vůči rozměrům a pozici bounding boxů, na které jsou aplikovány. Tyto hodnoty také mohou nabývat jakékoli hodnoty, protože jejich definiční obor není nijak omezen. Jsou-li hodnoty transformačních samé 0 pak je transformovaný box totožný s anchor boxem, který transformuje. Grafické znázornění, jak transformace modifikuje box je možné vidět na obrázku 5.4, kde v černé barvě je znázorněn anchor box, na který je aplikována transformace, a v červené barvě je možné pozorovat výsledek této transformace.



Obrázek 5.4: Transformace anchor boxu pomocí transformačních koeficientů.

Modul predikce regionu

Tento modul provádí detekci regionu v obraze. Využívá získanou mapu příznaků a provede nad ní konvoluční vrstvu, která slouží pro normalizaci této mapy. Dále jsou na tuto mapu příznaků aplikovány další dvě konvoluční vrstvy, které jsou určeny ke klasifikaci a regresi anchor boxů. Klasifikace určuje příslušnost boxů k jedné ze dvou výlučných tříd, kterými jsou text a pozadí. Regresní vrstva pro daný anchor box stanovuje hodnoty transformačních koeficientů. Schéma modulu je znázorněno na obrázku 5.5



Obrázek 5.5: Region proposal network. Proměnná N značí počet anchorů.

Klasifikační vrstva obsahuje pro každý pixel mapy příznaků dvojici hodnot pro každý anchor. Na tyto hodnoty je aplikována vrstva Softmax (více kap. 2.4.5, která hodnoty příslušící každému anchoru transformuje na míru pravděpodobnosti uvedených klasifikačních tříd. Anchory s nejvyšší pravděpodobností příslušnosti do třídy textu jsou systémem použity. Pro trénování této vrstvy je použita chybová funkce CrossEntropy loss (viz. kapitola 2.5.2) a značíme ji $\mathcal{L}_{\text{CLS}}$.

Regresní vrstva podobně jako klasifikační vrstva predikuje pro každý pixel mapy příznaků čtveřici hodnot, které udávají hodnotu transformačních koeficientů pro daný anchor (viz výše 5.1.2). Hodnota chybové funkce je spočítána pomocí Smooth L1 Loss (více 2.5.2). Vypočtenou chybu pro regresní vrstvu značíme $\mathcal{L}_{\text{REG}}$.

Chybová funkce tohoto modulu je kombinací výše uvedených chybových funkcí. Každá z těchto funkcí však produkuje jiné hodnoty chyby. Pro lepší učení je hodnota regresní chybové funkce vážena pomocí balančního parametru γ. Kompletní chybovou funkci značíme rovnicí:

$$\mathcal{L}_{\text{RPN}} = \mathcal{L}_{\text{CLS}} + \gamma\mathcal{L}_{\text{REG}}. \tag{5.5}$$

Výstupem této vrstvy jsou souřadnice predikovaných bounding boxů a míra pravděpodobnosti, že predikovaný bounding box ohraničuje nachází text. Predikované množství boxů je však velké proto je potřeba jejich počet redukovat metodou *Non-Maximal Suppression* (zkr. NMS). Pomocí těchto bounding boxů je dále provedena transformace původního obrazu a rozpoznání textu v transformovaných oblastech.

5.2 Transformace mezi modely

Z modelu rozpoznání jsou získány bounding boxy odpovídající predikovaným regionům obsahující text. Pro další zpracování je potřeba z původního obrazu získat výřezy těchto regionů a zmenšit je do patřičné velikosti. Zmenšení velikosti je potřeba protože další síť je nastavena na pevnou vstupní výšku, při které se průchodem reziduální sítí zredukuje do jediného bodu. Velikost výsledné oblasti získáme rovnicí:

$$w_s = h_s \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}}, \tag{5.6}$$

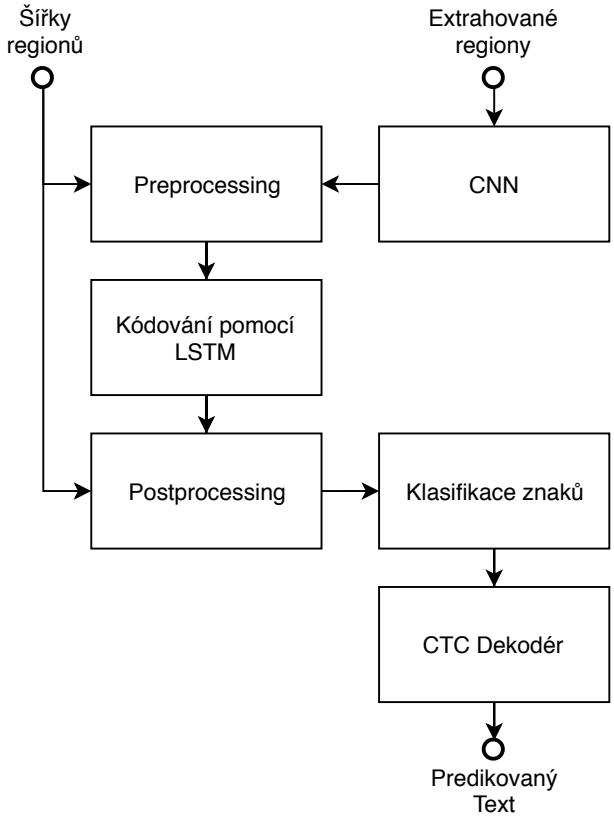
kde hodnoty y jsou maximální a minimální vertikální souřadnice bounding boxu textu, hodnoty x jsou analogicky horizontální souřadnice, h_s je pevně stanovená výška a w_s je vypočtená šířka. Takto získaný pod-obraz původního obrazu je vstupem další části, která v této oblasti rozpozná text.

Počet predikcí detekčního modelu je velmi velký proto je potřeba z výsledků vybrat ty nejlepší před provedením transformace. Ty nejlepší bounding boxy jsou vybrány pomocí

poměru překryvu a predikovaného skóre těchto boxů. Překryv je možné spočítat jako poměr mezi obsahem oblasti průniku a obsahem sloučené oblasti mezi bounding boxy. Tato metoda se nazývá *Intersect over Union* (zkr. IoU). Je využita metodou NMS a tímto je odfiltrována většina boxů, které do zadané oblasti nepatří.

5.3 Model rozpoznání textu

Rozpoznávací model dostane na vstupu výřez oblasti zájmu z obrazu, ve které se nachází text. Na tyto výřezy je aplikována reziduální síť inspirovaná VGG-16, která však provádí max-pooling pouze v oblastí výšky textu. Výstupem reziduální sítě je mapa příznaků, kde byla zachována původní šířka oblasti, avšak výšky byla zredukována do jediného bodu. Výsledek lze chápat jako temporální sekvenci jejíž jednotlivé body šířky reprezentují body postupného průchodu sekvencí. Na tuto sekvenci je aplikována rekurentní síť LSTM, která z temporální sekvence získá závislosti mezi body tvořící tuto sekvenci. Nakonec je provedena klasifikace nad zakódovanými daty z LSTM, která určí jaký znak se v bodě sekvence nachází. Pro získání korektních výstupů je třeba provést dekódování pomocí CTC dekodéru. Schéma tohoto modulu můžeme pozorovat na obrázku 5.6.



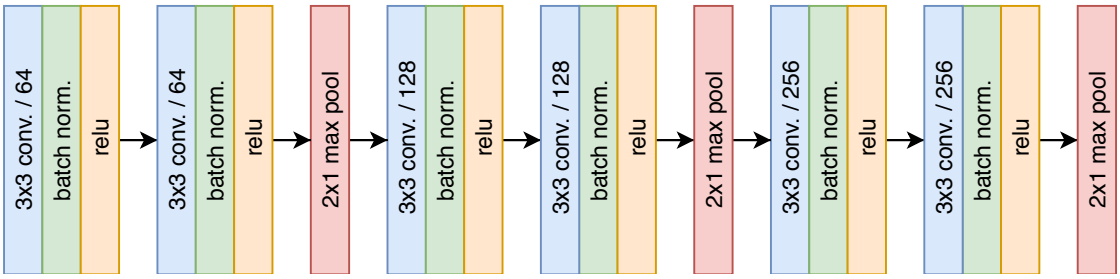
Obrázek 5.6: Model rozpoznání textu

5.3.1 Redukce vlastností regionu

Z předchozích kroku jsou získány výřezy textu s proměnnou šířkou. Z textových regionů extrahujeme mapu příznaků. Chceme však zachovat šířku textových regionů, aby nebyly

ztraceny informace o znacích a jejich rozestupech. K účelům extrakce příznaků slouží reziduální konvoluční síť inspirována architekturou VGG-16.

Tato reziduální konvoluční síť se skládá ze tří vrstev, které sdílí stejnou hloubku dat. A to konkrétně hloubku 64 kanálu v první vrstvě, 128 ve druhé vrstvě a 256 ve třetí vrstvě. Každá z těchto vrstev se skládá ze dvou operací konvoluce s jádrem velikosti 3. Za každou operací konvoluce následuje operace normalizace dávky jejíž výsledek je aktivovaný operací ReLU. Na konci každé z těchto tří vrstev následuje operace max-pool. na rozdíl oproti síti VGG-16, je operace max-pool nastavena na agregaci hodnot pouze ve vertikální ose. Tím je zaručeno, že výsledek této reziduální sítě zredukuje výšku do jediného bodu a šířka vstupu zůstane zachovaná. Výsledek této vrstvy je chápán jako sekvence temporálních dat, jejíž jednotlivé body posloupnosti jsou definovány jako body šířky výstupu. Tato data jsou dále zakódována pomocí modulu LSTM. Model této reziduální konvoluční sítě je možné pozorovat na obrázku 5.7.



Obrázek 5.7: Model reziduální CNN v rámci modulu rozpoznání textu.

5.3.2 Kódování rekurentní sítí

Vstupem je temporální sekvence získaná pomocí reziduální rekurentní sítě. Podíváme-li se na tuto sekvenci jako na časovou radu, pak je možné chápat jednotlivé body v čase této rady jako vývoj spektra, které dohromady tvoří obraz daného slova. Tento přístup je používán například i při rozpoznání zvuku, avšak rozpoznaní slov z obrazu je daleko složitější, protože může obsahovat daleko více proměnných. Mezi tyto proměnné patří například nastavení a parametry fontu pomocí níž je text v obraze vykreslen, kontrasty barev pozadí vůči textu atd. Dále se na získané sekvence aplikuje rekurentní neuronová sít, která má za úkol se naučit sekvenci replikovat. A to způsobem že získá ze sekvence širší kontext, jakým způsobem jsou jednotlivé časové kroky na sobe závislé. K těmto účelům je v práci využita obousměrná rekurentní síť LSTM, o jejíž principu i výhodách bylo psáno v kapitole 2.4.6. Obousměrnou LSTM lze chápat jako dvě LSTM sítě, přičemž první pracuje postupně od začátku ke konci a druhá přesně naopak od konce k začátku. Použití obousměrně LSTM namísto jednosměrné je protože poskytuje více možností jak odhalit kontextové informace, které definují temporální sekvenci. Výsledky obou směrů jsou dohromady sečteny do jediné sekvence, která je dále použita pro klasifikaci v jednotlivé časové body.

5.3.3 Klasifikace a dekódování sekvence

Z předchozí části dostaneme zakódované sekvence, kde každý bod obsahuje 256 kanálu. Mezi jednotlivými body této sekvence můžeme nalézt nějaké pravidelnosti, které definují nějaký konkrétní text. Na každý z těchto bodu je aplikována plně propojená vrstva, která má výstupní počet elementů roven velikosti množiny znaků jež chceme síť naučit. Tato vrstva

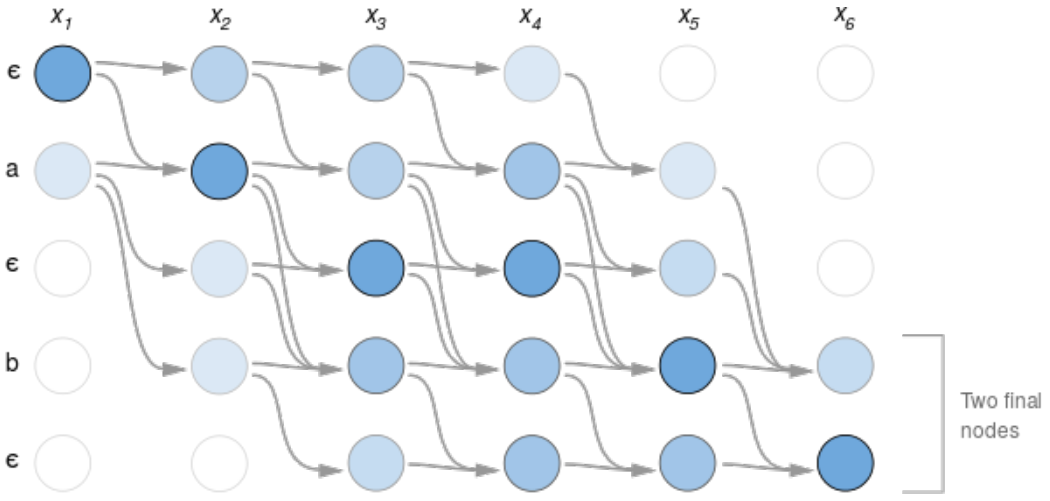
vrstva slouží ke klasifikaci výlučných tříd, z nichž každá klasifikuje nejvíce pravděpodobný znak v daném temporálním bodě sekvence. Toho je dosaženo aplikací vrstvy Softmax, která převádí hodnoty na míry pravděpodobnosti. Hodnotu daného temporálního bodu sekvence můžeme považovat jako klasifikovanou třídu, která dosahuje nejvyšší pravděpodobnosti.

Po klasifikaci je výsledkem sekvence nejvíce pravděpodobných znaků v jednotlivé body šířky. Avšak šířka, kterou je ohraničený vykreslený text, zpravidla dosahuje větší počet bodů než je počet znaků ve vykresleném textu. Proto musí být tento výsledek dekódován na odpovídající počet znaků v textu. K tomu je použita technika zvaná *Connectionist-Temporal-Classification* (dále jen CTC). Pomocí ní je možné delší sekvenci zredukovat do velikosti, kterou vyžadujeme. Pracuje tak že odfiltruje opakující se znaky. Proto je výstup možné reprezentovat mnoha cestami přes jednotlivé body sekvence, které dohromady produkují požadovaný výstup.

Učení této vrstvy probíhá pomocí CTC chybové funkce. Tato chybová funkce zohledňuje všechny možné cesty jak získat daný výstup. V následující rovnici je výpočet pravděpodobnosti všech validních cest vůči požadovanému výsledku:

$$p\left(Y|X\right) = \sum_{A \in V_{X,Y}} \prod_{t=1}^{T} p_t\left(a_t|X\right), \tag{5.7}$$

kde $Y = [y_1, y_2, ..., y_U]$ je vzorová sekvence, $X = [x_1, x_2, ..., x_T]$ je predikovaná sekvence. Mezi těmito sekvencemi musí platit vztah $U \ll T$. $V_{X,Y}$ je množina všech validních cest pro zadané Y. Funkce p_t je výpočet pravděpodobnosti korektního znaku v časovém bodě t. A s využití této pravděpodobnosti je chybová funkce definována podobně jako CrossEntropyLoss. Pracuje na principu akumulace pravděpodobností v jednotlivé kroky. Princip výpočtu této vrstvy můžeme pozorovat na obrázku 5.8. Na tomto obrázku je ukázán princip jak je chyba počítána pro 3 znaky abecedy a 6 bodů sekvence.



Obrázek 5.8: Princip kalkulace chybové funkce CTC.

Kapitola 6

Implementace

V této kapitole je popsána implementace vybraných částí navrženého systému. Nejprve jsou uvedeny použité technologie pro implementaci systému v 6.1. Následuje popis algoritmu učení a inference využívající modely neuronové sítě v kapitole 6.2. V kapitole 6.3 je popsán způsob výběru dat pro trénování bounding boxů. Kapitola 6.4 uvádí jak byly zvoleny anchory pro datovou sadu. V kapitole 6.5 je uveden algoritmus pro filtrování predikovaných regionů. Nakonec v kapitole 6.6 je uveden algoritmus CTC, který je použit pro dekódování výstupu z modelu rozpoznání textu.

6.1 Použité technologie

Python

Python je vysokoúrovňový skriptovací jazyk. Poskytuje programování v různých paradigmatech jako je objektově orientované programování, imperativní programování, procedurální procedurální nebo i funkcionální programování. Součásti jazyku Python je systém pro správu a instalaci závislostí pip. V posledních letech při výrazném nástupu strojového učení se Python stal velmi oblíbený pro jeho jednoduchost a také kvůli tomu, že jej lze navázat na kompilované knihovny (například v jazyce C) díky kterým dosahuje podobného výpočetního výkonu a přehlednosti. Dalším důvodem jeho častého použití je, že pomocí něj lze snadněji prototypovat algoritmy neuronových sítí v dostupných frameworcích.

Pytorch

Knihovna PyTorch[1] 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 (dokonce i open-source) a vyvíjená týmem zabývajícím se umělou inteligencí pro Facebook. Tato knihovna poskytuje několik hlavních výhod jako je počítání neuronových sítí za pomocí tenzoru. Přitom využívá grafické procesory(GPU) k dosažení velké akcelerace. Tenzory jsou matematické objekty, které však v rámci programování můžeme chápat jako multidimenzionální pole. Velkou výhodou knihovny PyTorch je funkce zvaná *automatic differentiation*. Jedná se o metodu, která sleduje veškeré změny provedené na tenzorech a je schopna tyto operace invertovat. Tato metoda umožňuje automatické počítání gradientů změn, což je velmi užitečné při tvorbě neuronových sítí, jelikož tato

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

funkce umožňuje výpočet diferencí parametru už při dopředném průchodu sítí. PyTorch navíc poskytuje mnoho implementací funkcí, vrstev, aj., které byly zmíněny v kapitole 2.

6.2 Rozdělení systému

Systém je rozdělen na tři spustitelné soubory, každý s jiným účelem. První dva z nich jsou aplikace určené pouze k učení detekční nebo rozpoznávací sítě. Pro jejich spuštění je vyžadován konfigurační soubor, který obsahuje parametry definující sítě. Významné parametry konfiguračního souboru jsou počet a rozměry anchor boxů, cesta k datové sadě určeného k trénování detekčního modelu, definovaná sekvence přípravných transformačních operací jež jsou provedeny pro každý prvek datové sady. Datová sada je načítán ve formátu uvedeném výše v kapitole 4 a je na něj aplikována série transformačních operací. Mezi tyto transformační operace patří zvětšení obrázku na minimální rozměr, doplnění paddingu, aby všechny obrázky měli jednotnou velikost a bylo je možné trénovat současně. Další význačnou transformací je převod dat na tenzory, které jsou kompatibilní s knihovnou PyTorch.

Trénování sítí je rozděleno na epochy, které jsou určeny jedním celým průchodem datovou sadou. V každé epoše je nejprve zamícháno pořadí prvků datové sady. V rámci epochy jsou postupně procházeny jednotlivé prvky datové sady a jsou na ně aplikovány transformace. Před zpracováním každého prvku datové sady jsou vynulovány tenzory držící hodnoty gradientů modelu, protože se nastavují v dopředném průchodu sítí. Dále je proveden dopředný průchod modelu, s aktuálními váhami, nad jednotlivými prvky. Ve fázi učení vrací model hodnoty chybových funkcí pro aktuální prvky datové sady. Pomocí hodnot chybových funkcí je proveden zpětný průchod sítí, který zajistí aktualizace gradientů podle aktuálních vstupních dat. Gradienty jsou vypočítány za pomocí modulu autograd. Dalším krokem je aktualizace vah sítě, která je provedena krokem pomocí optimalizačního algoritmu. V rámci práce je použit algoritmus Stochastic Gradient Descent. Tento algoritmus využívá vypočtené gradienty a hodnotu rychlosti učení(angl. learning rate) k aktualizaci vah neuronové sítě. (více v kapitole 2.5.1). Po dokončení epochy je model uložen. Pro aktualizaci rychlosti učení je použit plánovač zvaný *Cosine Anealing*. Tento plánovač má stanoveny hranice maximální rychlosti učení a minimální rychlosti učení, které jsou aktualizovány na základě funkce kosinus. Změna rychlosti probíhá s každým prvkem v datové sadě.

Celkový proces učení neuronové sítě vidíme v algoritmu 1 jako pseudokód.

Posledním spustitelným souborem je samotná inference systému. Tato aplikace opět vyžaduje na vstupu konfigurační soubor. Pro použití této aplikace však musí existovat naučený model. Tato aplikace slouží k predikci bounding boxů a rozpoznání textů v nich pro zadaný vstupní obrázek. Aplikace načte obrázek a aplikuje na něj nezbytné transformační operace, které jsou vyžadovány modelem. Mezi ne patří normalizace velikosti obrázku a jeho následné převedení na tenzor. Modely detekce i rozpoznání jsou nastaveny na evaluaci, která zaručí nejvyšší rychlost zpracování. Nejprve jsou vstupní data předány detekčnímu modelu, který vrátí všechny predikované textové regiony. Z těchto regionů jsou filtrací vybrány ty nejpravděpodobnější a na základě nich jsou vytvořeny výřezy ze zdrojového obrázku. Výřezy jsou poté rozpoznány jednotlivě jeden po druhém. Algoritmus inference je možné vidět v algoritmu 2

Algoritmus 1: Ucení systému.

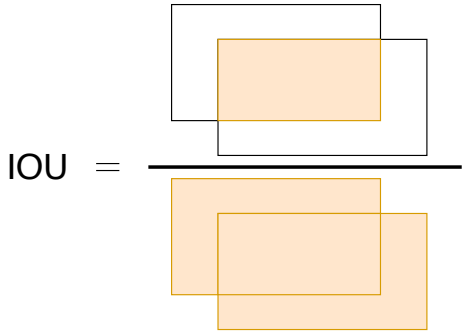
```
for i = 0; i <MAX_EPOCH; i++ do
    Zamíchání indexu datasetu.
    for j = 0; j <NUM_ITEMS_DATASET; j++ do
        Získání dat pro index j.
        Předzpracování dat pomocí definovaných transformací.
        Vynulování gradientu sítě.
        Loss ← Dopředný průchod sítí.
        Zpětný průchod sítí pomocí vypočítaných lossů.
        Krok optimalizačního algoritmu sítě.
        Krok plánovače učení sítě.
    Vytisknutí výsledku učení epochy.
    Uložení stavu sítě pro danou epochu.
```

Algoritmus 2: Inference systému.

```
Načtení stavu sítě na základě konfiguračního souboru.
Načtení obrázku ze zadané cesty
Předpříprava dat pomocí transformací.
Skóre, Boxy ← Dopředný průchod detekční sítí
for (S,B) ∈ Skóre × Boxy do
    if S < threshold then
        continue
    Výřez oblasti ze zdrojového obrázku
    Text ← Dopředný průchod rozpoznávací sítí.
Výstup dat do souboru.
```

6.3 Příprava vzorů pro učení detekce

Aby se síť naučila klasifikovat ty správné boxy, jsou v preprocessingu náhodně vzorkovány tzv. *pozitivní* a *negativní* anchory. Všechny ostatní jsou v trénování vynechány. Vzorky, které mohou být považovány za pozitivní, či negativní anchor box se musí se vzorovým bounding 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 bounding boxy dohromady zabírají. IOU nabývá hodnoty 1 pokud jsou boxy totožné a 0 pokud bounding boxy nemají žádnou sdílenou oblast. Tuto funkci můžeme pozorovat graficky v obrázku 6.1.



Obrázek 6.1: Funkce IOU znázornena graficky pro dva boxy.

Dalším kritérium výběru anchoru, je že žádný z kandidátních anchorů pro trénování nesmí přesahovat za hranice obrázku. Všechny anchory, které tuto hranici přesahují nejsou při trénování zohledněny. Algoritmus nejdříve vybere všechny kandidáty, které se nejlépe překrývají s jedním z vzorových bounding boxu, které chceme učit. Dále jsou náhodně dovybrány ze všech kandidátu ti, kteří mají hodnotu překryvu vyšší, než je stanovený pozitivní práh (v tomto projektu se jedná o hodnotu 85%), aby byl splněný počet pozitivních záznamu. Všechny vybrané pozitivní záznamy představují anchory jejichž ohodnocení chceme posílit. Zároveň s pozitivními anchory je třeba určit i negativní anchory. Ty se vybírají oproti pozitivním boxům přesně opačně. Ze všech zbylých kandidátu se vyberou jen ti, kteří mají hodnotu překryvu nižší než stanovený negativní práh (v této práci se jedná o 50%). Negativní anchory představují boudning boxy, které při klasifikaci zhoršeny. Všechny ostatní anchory ignorujeme a jejich hodnoty skóre nejsou nijak upraveny. Vzor trénování klasifikace díky tomu nabývá třech hodnot. Hodnotu 0 nabývá pokud se jedná o negativní anchor, hodnotu 1 pokud je anchor pozitivní a hodnotu -1 pokud je anchor box při trénování ignorován. Počet těchto vzorků a poměr pozitivních vůči negativním anchorům je nastavitelný v konfiguračním souboru.

Pro všechny takto vybrané pozitivní anchory jsou dále dopočítány i vzory pro regresi transformačních koeficientů. Záporné anchory jsou v optimalizaci regrese vynechány, úprava jejich parametrů. Pro každý z pozitivních anchoru je vybrán anotovaný box, který má s ním největší hodnotu překryvu. Díky tomu je zaručeno že se daný anchor bude snažit co nejdůkladněji napodobit tento anotovaný box. Proto je potřeba určit trénovací vzory transformačních koeficientů, které jsou potřeba k transformaci daného anchor boxu na přidělený anotovaný box. Tyto koeficienty získáme pomocí následujících rovnic:

$$t_x = \frac{g_x - a_x}{a_w}, \tag{6.1}$$

$$t_y = \frac{g_y - a_y}{a_h}, \tag{6.2}$$

$$t_w = \ln \frac{g_w}{a_w}, \tag{6.3}$$

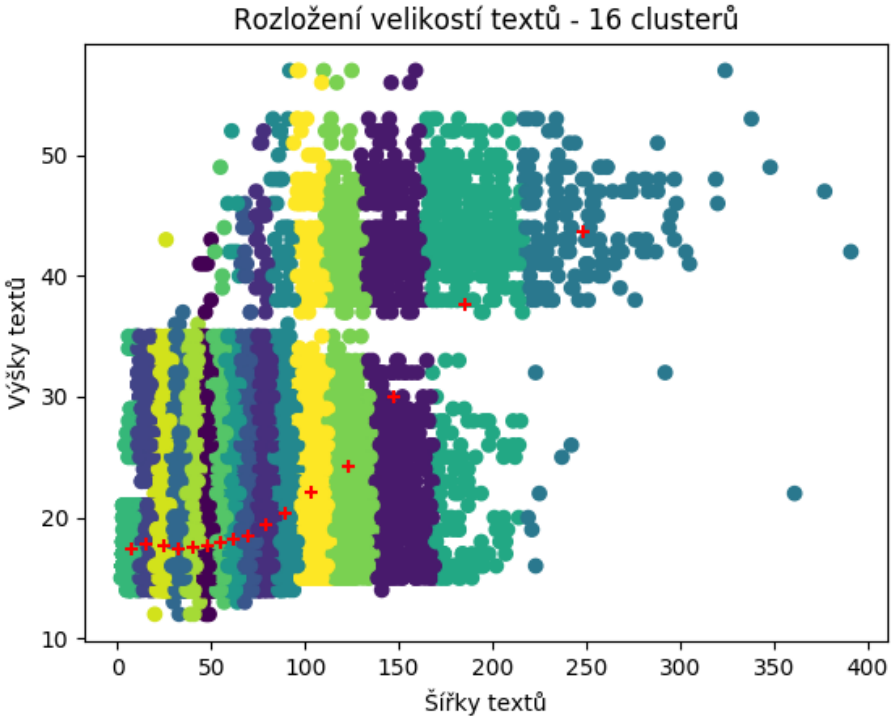
$$t_h = \ln \frac{g_h}{a_h}, \tag{6.4}$$

kde hodnoty (a_x, a_y, a_w, a_h) reprezentují souřadnice anchor boxu, hodnoty (g_x, g_y, g_w, g_h) reprezentují souřadnice anotovaného boxu, kterého chceme dosáhnout, a hodnoty (t_x, t_y, t_w, t_h) reprezentují transformační koeficienty pomocí nichž je možné anchor box transformovat na vzorový box. Společně s výpočtem těchto koeficientů je i nastavena maska pro

daný anchor. Maska slouží k určení hodnot anchorů, které se budou aktualizovat. Hodnota masky pro pozitivní vzorky jsou nastaveny na $(1, 1, 1, 1)$. Pro všechny neupravované anchor boxy jsou nastaveny hodnoty vzorů transformačních parametru i masky na 0. Maska je použita, protože všechny hodnoty masky nastavené na 0 se nebudou nijak podílet na hodnotě 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.

6.4 Anchory

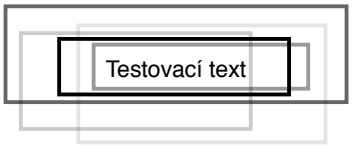
Tato práce se zabývá zpracováním textu, proto bylo potřeba zvolit takové anchor boxy, které vzhledem připomínají bounding boxy obsahující obsahují text. Podíváme-li se na jakýkoli text, můžeme vidět, že většina jeho slov je rozměrově více široká, než-li vysoká. Proto je i logické, aby v tomto duchu byly zvoleny anchor boxy. Proto byly anchor boxy použity v optimalizováném tvaru pro vygenerovanou datovou sadu. Pomocí metody *k-means* byly všechny body rozděleny do několika skupin a střed každé z těchto skupin zvolen jako anchor box. Takto zvolené boxy lze pozorovat na grafu 6.2, kde každý bod reprezentuje šířku a výšku bounding boxu ohraničující text, který se nachází v datové sadě. Dále vidíme barevně odlišeny jednotlivé vypočtené skupiny a červeným křížem je označen střed této skupiny. Jedná se zde o body, jež jsou zvoleny jako anchor boxy systému.



Obrázek 6.2: Vypočtené anchory z datové sady.

6.5 Filtrování predikovaných regionu

Detekční modul produkuje pro obrázek standardní velikosti velké množství predikcí daných bounding boxů. Z těchto dat je však nutné vybrat ty nejpodstatnější. Nejprve jsou všechny predikované bounding boxy seřazeny sestupně podle jejich ohodnocení pravděpodobnosti pro třídu text. Z takto seřazených boxů je vybráno prvních N boxu (konfigurovatelné pomocí parametrem `num_boxes_pre_nms`). Touto metodou jsou získány pouze ty nejlépe ohodnocené predikované bounding boxy. Avšak může nastat situace, kdy více predikcí spadá do stejné oblasti obrázku a predikují totožný objekt. Tuto situaci můžeme pozorovat v obrázku 6.3, kde jsou zobrazeny predikované boxy. Intenzita šedé barvy těchto bounding boxů znázorňuje hodnotu pravděpodobnosti.



Obrázek 6.3: Predikce boxu, které se prekrývají v dané oblasti.

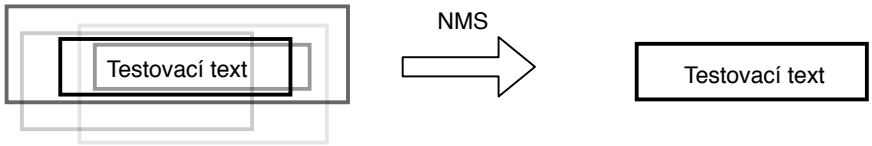
Pro odfiltrování takto podobných bounding boxů je použit algoritmus Non-Maximum Suppression (dále jen NMS). Algoritmus projde všechny predikované boxy a vybere pro dané oblasti pouze ty, které mají nejvyšší skóre. Překryvy boxu jsou porovnány pomocí funkce IOU (viz výše). Implementaci NMS je možné vidět v algoritmu 3. Tento algoritmus je navržen tak, aby získal indexy bounding boxů, jež se mají zachovat.

Algoritmus 3: Non-Maximum Suppression.

Data:
box - Seznam souradnic boxu,
score - Seznam ohodnocení boxu,
N - Celková délka techto seznamu.
threshold - Práh pro hodnotu prekryvu, nad kterou potlacujeme.
Result: *keep* - Seznam indexu boxu, které jsou zachovány.

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

Princip algoritmu můžeme pozorovat vizuálně v obrázku 6.4. V obrázku je patrné, že je zachován pouze ten nejlépe ohodnocený bounding box (nejvíce intenzivní hodnota šedi) a všechny ostatní bounding boxy jsou algoritmem potlačeny. S pomocí tohoto algoritmu zamezíme vzniku duplikátních boxů pro stejné oblasti.



Obrázek 6.4: Prekrývající se predikce boxu, jsou pomocí metody Non-Maximum suppression odfiltrovány.

6.6 Rozpoznání a dekódování

Použitá rekurentní síť LSTM je implementována za pomocí knihovny PyTorch a vyžaduje na vstupu tenzor, který obsahuje výřez oblasti odpovídající danému bounding boxu. Pro použití s modulem LSTM je však nutné tato data zabalit do struktury, která lépe reprezentuje temporální data. Zabalený tenzor je předán obousměrné vrstvě LSTM, která produkuje v každém směru 256 kanálu v jednotlivé časové body. Tím jsou získány dvě sekvence. První kóduje vstupní sekvenci od začátku po konec a druhá naopak od konce po začátek. Výstupem je ale opět zabalená sekvence. Po rozbalení a aplikace klasifikační vrstvy je potřeba provést dekódování výstupu. Toho je dosaženo CTC dekodérem, jehož pseudokód můžeme pozorovat v algoritmu 4.

Algoritmus 4: Dekódování CTC sekvence.

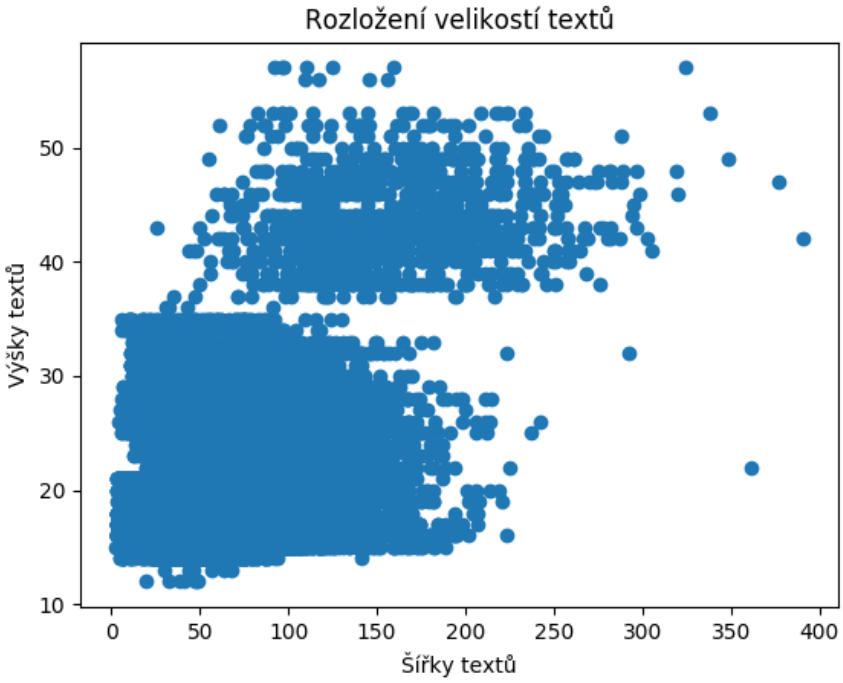
Data:
input_text - Seznam S nejpravděpodobnějšími znaky.
N - Délka tohoto seznamu.
Result: *decoded_text* - Výsledný seznam znaků.

```
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
```

Kapitola 7

Experimenty

Modely sítí, které byly představeny v předchozích kapitolách, byly trénovány na stroji se dvěma grafickými kartami NVidia GTX-1080 a procesorem Intel Core-i7. Jako učební vzor byla nástrojem „gui-generator“ vygenerovaná datová sada s obrázky o rozměrech 1200x800. Tato datová sada obsahuje kolem 800 obrázku, které v průměru obsahují 200 textových anotací na obrázek. Pro vyhodnocení byla vytvořena menší datová sada, která čítá přibližně 50 obrázků o stejných rozměrech a průměru textových anotací. Obě datové sady jsou pro algoritmy OCR velmi složité, protože obsahují velké množství slov a také obsahují různé kontrasty pozadí vůči písmu. Distribuci velikostí bounding boxů je možné vidět v obrázku 7.1.



Obrázek 7.1: Distribuce bounding boxů v datasetu.

7.1 Model detekce

Model detekce byl trénován při velikosti dávky 4 na jedné grafice. Dohromady bylo společně trénováno 8 položek při využití obou dostupných grafických karet. Při trénování sítě bylo použito plánovače využívající funkci kosínus. Společně s tímto plánovačem bylo využito optimalizátoru sítě Stochastic gradient descent with restarts. Jedná se o variantu klasického SGD, avšak tento optimalizační algoritmus po dosažení určitého počtu kroků obnoví learning rate na počáteční hodnotu. Díky tomu je možné nalézt řešení, které se nalézá mimo lokální minimum. Pro měření kvality detekce textu jsou využity metriky přesnosti i citlivosti. Výsledky je možné vidět v tabulce 7.1.

Tabulka 7.1: Vyhodnocení detekce algoritmů OCR

OCR	Citlivost [IOU 0.75]	Přesnost [IOU 0.75]
Microsoft Azure	91.21%	87.00%
Google Cloud Vision	93.36%	86.72%
Tato práce	87.56%	54.35%

Z výsledků uvedených výše je vidět, že síť využitá v této práci má velmi dobrou schopnost rozeznat bounding boxy v textu. To lze odvodit z hodnoty metriky citlivosti. Častěji se však setkáme s názvem *recall*. Metrika recall udává kolik prvků ze testovací množiny bylo korektně predikováno a její nejlepší výsledek je 100%. Naopak dle metriky přesnosti, nebo také *precision*, vidíme že algoritmus produkuje velké množství zbytečných predikcí. Při bližší analýze výstupů lze vidět, že nadbytečné predikce se nachází zejména u krátkých slov, které jsou součástí delšího souvislého textu jak můžeme vidět na obrázku 7.2. V obrázku také vidíme několik slov pro které nebyl detekován bounding box. Také vidíme že detekční síť si poradila s různými barevnými rozpoloženími v obraze.

Obrázek 7.2: Detekce mnoha nadbytečných bounding boxů.

V dalším obrázku 7.3 je ukázáno, že algoritmus je schopen si poradit i s texty, které mají velmi nízký kontrast vůči pozadí. Toto je výhoda navrženého algoritmu vůči porovnávaným algoritmům. Oba zbylé algoritmy mají velký problém detekovat text je-li příliš málo kontrastní. Jejich nekorektnost přisuzuji

Obrázek 7.3: Detekce textů s nízkým kontrastem vůči pozadí.

7.2 Model rozpoznání

Model rozpoznání byl trénován při velikosti dávky 4. Stejně jako při trénování detekce byl použit plánovač využívající funkci kosinus a optimalizační algoritmus Stochastic gradient descent with restarts. Výsledky tohoto modelu, společně s dalšími ocr algoritmy je možné vidět v tabulce 7.2.

Tabulka 7.2: Vyhodnocení rozpoznání algoritmů OCR

	Metrika znaku		Metrika slov	
OCR	Citlivost	Přesnost	Citlivost	Přesnost
Microsoft Azure	83.21%	87.00%	79.50%	80.35%
Google Cloud Vision	87.36%	86.72%	85.89%	73.26%
Tato práce	18.10%	25.91%	16.02%	19.74%

Tabulka obsahuje dvě kategorie měření a to metrika znaků a metrika slov. Metrika znaků určuje kvalitu OCR systémů ve schopnosti rozeznat korektní znaky na v celkovém textu. Metrika slov navíc k tomu ještě porovnává segmentace daných algoritmů, tedy boxy musí mít nějaký minimální překryv, aby byly zohledněny. V každé z těchto kategorií je určen recall i citlivost, které nesou informace o schopnostech systému rozeznat jednotlivá slova.

Avšak výsledky algoritmu rozpoznání této práce nejsou příliš pozitivní. Navržený systém se nedokáže korektně přizpůsobit všem trénovacím datům a správně klasifikovat znaky. V práci jsou součástí slovníku i české znaky, které jsou však jsou velmi podobné jejich variantám bez interpunkce. Díky tomu má tento model problémy naučit se klasifikovat získanou sekvenci korektně. Model jež detekuje text, není pro některá slova 100% a některé znaky jsou úplně vynechány.

Kapitola 8

Závěr

V rámci práce byly nastudovány metody zpracování obrazu pomocí neuronových sítí, práce se především zaměřila na metody hlubokého učení a také byly prozkoumány metody rozpoznání textu z obrazu, které využívají neuronové sítě. Pro generování datové sady byla využita aplikace gui-generátor, jež byla dále rozšířena o širší možnosti generování různých grafických webových rozhraní a posloužila pro generování datové sady. V rámci práce byla vytvořena datová sada obsahující 1000 obrázků uživatelských rozhraní včetně jejich anotací. Část těchto dat je určena k validaci výsledků, zbytek slouží jako trénovací datová sada pro učení navržených modelů. V rámci řešení byl dále implementován program pro vyhodnocení kvality algoritmu OCR na základě zvolené metriky. Pomocí tohoto programu a výše vytvořené datové sady byly jednotlivé služby ohodnoceny definovanou metrikou rozpoznání textu a jejich výsledky dále prozkoumány.

Hlavním přínosem práce je, že zde byly použity moderní metody návrhu neuronových sítí. Navržená síť byla implementována a natrénována na vygenerované datové sadě. Trénováním se podařilo dosáhnout nadstandartních výsledků pro detekci textů i nalezení souřadnic obdélníků, které ohraničují textové oblasti. Tento model lze použít samostatně a může být velmi užitečný pro řadu aplikací, například detekce textu v obraze použitá. Další část, kdy je prováděno samotné rozpoznání textu, byla natrénována, vyhodnocena a dosáhla přibližně 20% úspěšnosti rozpoznání znaků. Tento výsledek říká, že datová sada obsahuje nějaké texty, které se model rozpoznání nedaří naučit. Pro rozpoznání byly využity české znaky jako výlučné klasifikační kategorie. Mnoho klasifikovaných znaků si je však velmi podobných. Liší se pouze v detailech jako je diakritika nad znaménky. Výsledek říká, že OCR systém, který má na výstupu všechny znaky kódování Unicode, není příliš reálný. A to hlavně kvůli podobnostem mezi jednotlivými znaky i jejich interpunkčními variantami.

Návrh i implementace těchto modelů sítí byla velmi velká výzva, protože pro dobrý návrh těchto sítí je potřeba velkého množství znalostí v oblasti neuronových sítí. Výsledné modely kombinují dohromady mnoho paradigmat moderních neuronových sítí mezi které patří hluboké učení neuronových sítí, reziduální konvoluční sítě pro extrakci příznaků z obrazu, predikce bounding boxů z takto získaných příznaků i rozpoznání textu pomocí rekurentní neuronových sítí. Možným rozšířením této práce je implementace klasifikačního algoritmu, který zvlášť klasifikuje znaky a zvlášť jejich interpunkce. Nebo dalším rozšířením je použití sítě pro transformaci prostorových souřadnic a spojení obou částí do jediné souvislé sítě.

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

Pro projekt lze vytvořit virtuální prostředí python, kde lze najít všechny závislosti projektu, které jsou staženy pomocí příkazu: `pip install -r requierements.txt`
Další možností pro vytvoření prostředí je použití poskytnutého docker konfiguračního souboru pro jeho nastavení pro nvidia-docker. Dále je možné spustit program pomocí příkazů:

Trénování detekce

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

Trénování rozpoznání

`$ python3 train-recognition.py --config /cesta/ke/configu` Tento příkaz spustí trénování rozpoznávacího 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.