VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta strojního inženýrství

BAKALÁŘSKÁ PRÁCE

Brno, 2017 Marek Čermák

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

METODY VYUŽÍVANÉ PRO OCR
METHODS USED FOR OCR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Marek Čermák**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Daniel Zuth, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Marek Čermák**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Daniel Zuth, Ph.D.**
Akademický rok:	2017/18

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Metody využívané pro OCR

Stručná charakteristika problematiky úkolu:

Práce se bude zabývat rešerší dostupných metod a SW pro OCR a poté bude vybrána jedna z metod, která bude naprogramována a otestována na praktickém použití. Předpokládá se využití neuronové sítě pro klasifikaci částí textu.

Cíle bakalářské práce:

Popište používané metody pro rozpoznávání obrazu.
Popište existující SW a proveďte porovnání.
Vyberte metodu vhodnou pro realizaci.
Popište knihovny a nástroje použité pro realizaci.
Vytvořte a otestujte vybraný algoritmus.

Seznam doporučené literatury:

SCHANTZ, Herbert F. The history of OCR, optical character recognition. Manchester Center, Vt.: Recognition Technologies Users Association, c1982. ISBN 9780943072012.

ABSTRAKT

Ačkoli je OCR (Optické rozpoznávání znaků – Optical Character Recognition) problematikou sahající do druhé poloviny dvacátého století, dostalo se mu v současnosti velké pozornosti v souvislosti s počítačovým viděním a detekcí objektů. V této práci bude popsána historie OCR a stručně budou zmíněny techniky doposud používané pro OCR. Pozornost bude soustředěna na současné metody rozpoznávání textu, tedy na soft computing. Protože v této oblasti zastávají největší roli neuronové sítě, budou zmíněny a popsány některé architektury a následně bude realizován software pro rozpoznávání alfanumerických znaků pomocí konvoluční neuronové sítě.

KLÍČOVÁ SLOVA

ocr, metody ocr, rozpoznávání textu, rozpoznávání znaků, soft computing, umělá inteligence, strojové učení, neuronové sítě, konvoluční neuronové sítě, detekce objektů, počítačové vidění, zpracování obrazu, data, vizualizace, tensorflow

ABSTRACT

Although OCR (Optical Character Recognition) is a topic which has been a subject of research since the second half of the 19th century, it has recieved a significant attention in the field of computer vision and object detection recently. This thesis presents history of OCR and briefly describes techniques which have been used over the course of time for character recognition. Main focus lies in the current text recognition methods introduced by soft computing. Since the major portion of the field is covered by neural networks, various architectures will be presented. Eventually a software for alphanumeric characters recognition will be implemented using a convolutional neural network.

KEYWORDS

ocr, ocr methods, text recognition, character recognition, artificial intelligence, machine learning, neural networks, convolutional neural networks, object detection, computer vision, image processing, data, visualisation, tensorflow

ČERMÁK, Marek. *METODY VYUŽÍVANÉ PRO OCR*. Brno, 2018, 76 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce: Ing. Daniel Zuth, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „METODY VYUŽÍVANÉ PRO OCR“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Danielu Zuthovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Tato práce si klade za cíl ukázat možnost využití metod strojového učení a soft computingu pro rozpoznávání textu a demonstruje zachování funkcionality i pro tištěné dokumenty. Zabývá se aplikací moderních metod strojového učení v souvislosti s OCR (Optické rozpoznávání znaků – Optical Character Recognition). Přestože optické rozpoznávání znaků je problematikou sahající do druhé poloviny dvacátého století (viz kap. Historický vývoj), dostalo se mu v současnosti velké pozornosti v souvislosti s počítačovým viděním a detekcí objektů. Současné OCR přesáhlo rozpoznávání tištěných textových dokumentů a nachází uplatnění v rozpoznávání textu ve scénickém prostředí, tj. běžné fotografie a kamerové záznamy. Toto ale vylučuje použití některých metod, které byly využívány při pouhém rozpoznávání znaků z tištěných dokumentů.

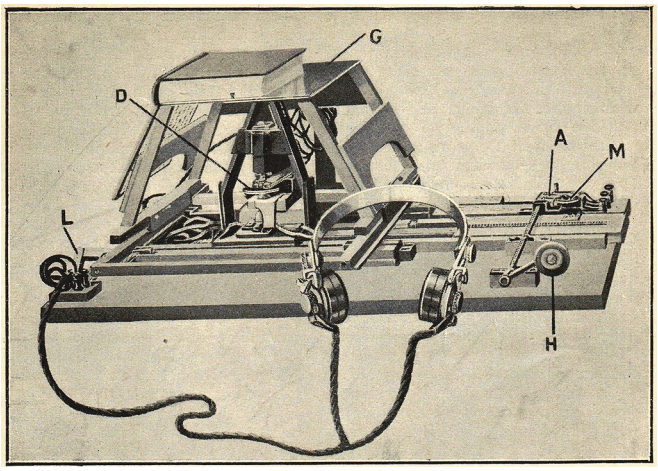
V této práci bude popsána historie OCR a stručně budou zmíněny techniky doposud používané pro OCR. Pozornost bude soustředěna na současné metody rozpoznávání textu, tedy na *soft computing*. Protože v této oblasti zastávají největší roli *neuronové sítě*(viz kap. Metodika OCR), budou zmíněny a popsány některé architektury a následně bude realizován software pro rozpoznávání alfanumerických znaků pomocí zvolené neuronové sítě.

Tento software bude základem komplexního systému pro převod tištěných dokumentů do elektronické podoby a bude navržen s ohledem na požadavky tohoto systému. Protože výsledná aplikace bude využívat neuronovou síť, jejíž architektura ani parametry nejsou předem známy, a protože je cílem dosáhnout znovupoužitelnosti softwaru pro širší spektrum uživatelů, bude aplikace navržena tak, aby bylo možné dynamicky modifikovat a posléze optimalizovat tyto hodnoty. K obsluze aplikace bude vytvořen server pro zpracování dotazů klienta.

Motivací ke zpracování této práce je zejména nedostatek otevřeného a uživatelsky přívětivého softwaru pro OCR, zejména pak takového, který je schopen přizpůsobit požadavky cílové skupině uživatelů - např. software pro rozpoznávání českých textů. Dále pak snaha poukázat na obrovské možnosti využití neuronových sítí a metod umělé inteligence a strojového učení a naučit se vybrané techniky implementovat a integrovat.

1 HISTORICKÝ VÝVOJ

Počátky optického rozpoznávání znaků lze nalézt už v roce 1870. Za první vynález v historii OCR bývá považován přenosový systém využívající mozaiku fotobuněk amerického vynálezce Charlese R. Careyho.
Téměř o půl století později, v roce 1912, přišel s vynálezem Optofonu (viz obr. 1.1 [1]) irský fyzik Edmund Edward Fournier d'Albe. Optofon využíval selenové foto-senzory, které konvertovaly detekovaný znak na zvuk o určité frekvenci. Fournierův přístroj tak mohl být využíván i slepci.
Za první milník v rozpoznávání znaků je považován přístroj, jehož název lze volně přeložit jako „Čtecí stroj". Toto zařízení s foto-senzorem vrhalo světlo na slovo, pokud korespondovalo s uloženým vzorem. V roce 1929 jej vynalezl německý průkopník v informačních technologiích Gustav Tauschek.
Kolem padesátých letech 20. století se spolu s rozvojem informační vědy zvyšoval i zájem o informační systémy. Kontroverze informační a knihovní vědy nevyhnutelně směřovala k rozšíření myšlenky zpracování textu počítačem. Snaha vytvořit novou technologii pro řešení starých problémů vedla k vzniku řady nekonvenčních informačních systémů. Toto období bývá označováno jako počátek vědecko-technické revoluce.
Skutečné základy byly položeny v roce 1954 americkým magazínem Reader's Digest, který uvedl do provozu zařízení pro převod ručně psaného textu na děrné štítky. Takto zaznamenaná data bylo možné dále zpracovat na počítači.



Obr. 1.1: Detail Optofonu otištěn ve vědecké publikaci roku 1922

1.1 20. století

1.1.1 První generace

Generace komerčních systémů pro optické rozpoznávání znaků v letech od 1960 do 1965 je nazývána jako první generace OCR. Všeobecně je tato generace charakterizována využitím jednoduchých znaků.
Systémy tohoto období se specializovaly na rozpoznávání znaků, které byly vyvinuté speciálně pro tento účel. Přestože se postupně začínaly objevovat systémy, které disponovaly znalostí více fontů, byl počet těchto fontů silně limitován rozpoznávací metodou.
Klasifikační metodou bylo porovnávání obrazu a vzoru z knihovny prototypů. Lze tedy odvodit omezení plynoucí z tohoto přístupu, jako jsou například značná časová i prostorová náročnost.

1.1.2 Druhá generace

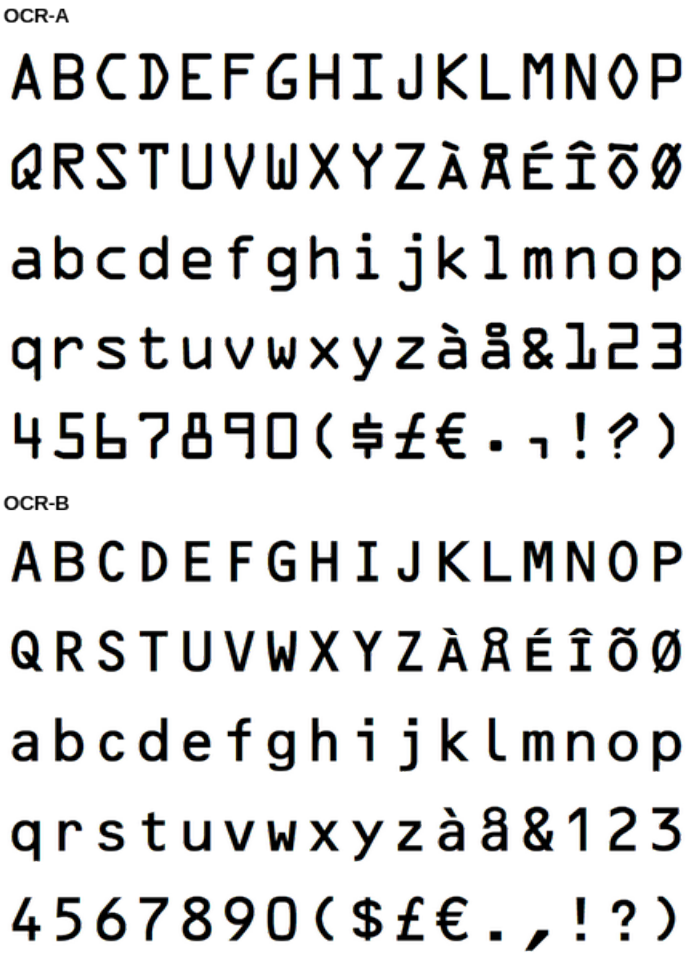
Čtecí systémy druhé generace se začaly objevovat v polovině 60. let a na počátku 70. let. Tyto systémy byly schopny rozpoznat běžné strojově vytisknuté texty a do jisté míry i texty psané.
Prvním slavným systémem tohoto druhu byl IBM 1287, který byl představen na World Fair v New Yorku v roce 1965. V tomto čase byl také firmou Toshiba vyvinut první automatický třídič dopisů podle poštovních čísel.
V roce 1966 byl po studiu OCR požadavků a potřeb za účelem zvýšení účinnosti rozpoznávání dokončen Americký standard „OCR character set“ OCR-A. OCR-A (viz obr. 1.2)[1]byl standard dobře čitelný pro stroje, ale poměrně nevzhledně vypadající, proto o něco později spatřil světlo světa nový Evropský standard standard OCR-B navržený Andrianem Frutigerem.

1.1.3 Třetí generace

Výzvou pro nadcházející generaci bylo rozpoznávání symbolů nižší kvality a ručně psaných znaků. V polovině sedmdesátých let se dostalo vzniku třetí generaci OCR. Důležitými faktory pro zlepšení technologií a umožnění sofistikovanějších systémů

[1]Font byl vygenerován pomocí `http://www.identifont.com/differences?first=OCR-A&second=OCR-B&q=Go`.

bylo snížení cen a zvýšení výkonnosti hardwaru.
Výsledkem této generace bylo odstranění nutnosti speciálních fontů. Rozpoznatelné již tedy byly i znaky psané na psacích strojích, na kterých v té době vznikala většina dokumentů.



Obr. 1.2: Porovnání OCR–A (nahoře) a OCR–B (dole) standardů

1.2 Současnost a moderní pojetí

[2]Po roce 1990 začaly být techniky zpracování obrazu a rozpoznávání vzoru kombinovány s metodami AI[3]. Vědci vyvíjeli komplexní OCR algoritmy, které zpracovávaly data ve velkém rozlišení, ale vyžadovaly výpočetně náročné operace, proto nebyly zcela běžně dostupné. S rostoucím výpočetním výkonem a přesným elektronickým

[2]Volně přeloženo z [2]
[3]AI (Umělá inteligence – *Artificial Intelligence*)

vybavením, jako jsou scannery, kamery aj. se staly využitelnými a efektivními moderní metody přístupu využívající techniky jako jsou ANN[4], HMMs[5] či fuzzy rozhodování.

V kombinaci s těmito technikami se terminologie OCR stala součástí počítačového vidění a bývá označována jako *text recognition*. Již není soustředěna pouze na čtení slov v dokumentech, ale na rozpoznávání jakéhokoli textu v nehomogenním, neohraničeném prostředí. Příkladem takovýchto aplikací může být například rozpoznávání SPZ automobilů, či popisných čísel domů.

V současnosti jsme schopni pro určité množství aplikací pomocí moderních metod navrhnout dostačující řešení. Nicméně hranice, kdy bude úroveň strojového zpracování textu na úrovni lidského plynulého čtení textu, je zatím otázkou budoucnosti, obzvláště pro neohraničené ručně psané texty.

[4]ANN (Umělá neuronová síť – *Artificial Neural Network*)

[5]HMM (Skrytý Markovův model – *Hidden Markov Model*)

2 METODIKA OCR

Obecné schéma řešení

2.1 Preprocessing

Předzpracování (*preprocessing*) má za cíl extrahovat relevantní textové části a připravit je pro rozpoznávání, tedy pro segmentaci a klasifikaci [3]. Hlavními úkony jsou redukce šumu, normalizace dat a komprese množství zpracovávané informace [4].
Normalizaci a kompresi je vhodné provádět na začátku. Algoritmy použité pro další techniky mohou dosahovat vysoké časové i prostorové složitosti s rostoucím vstupem. Umístění normalizace a komprese před tyto úkony dosáhneme mnohdy značného urychlení běhu programu.
Některé metody OCR předpokládají standardizovanou velikost vstupu. Rozdílná velikost by v tomto případě mohla způsobit nesprávnou klasifikaci, nebo dokonce selhání běhu programu. Je tedy nutné normalizovat obraz na standardní rozměr.
Komprese s sebou přináší výhodu v podobě zmenšení objemu dat. Je ale důležité mít na paměti, že při zmenšování objemu dat klesá množství informace obsažené v systému, což může vést k nekorektní klasifikaci.
Pro **normalizaci a kompresi** dat existují techniky škálování. Mezi techniky škálování patří například:

- interpolace nejbližším sousedem (*nearest neighbour interpolation*)
- bilineární a bikubická interpolace
- konvoluce pomocí konvolučního jádra (convolution kernel), nazývaného také jako filtr. Konvoluce bude blíže charakterizována v kapitole 3 - Neuronové sítě.

Pro samotnou **redukci šumu** existují stovky dostupných technik, které mohou být kategorizovány do tří hlavních kategorií - filtrování, morfologické operace a tzv. *noise modeling* [5].
Filtry mohou být navrženy s ohledem na následující operace, pro které opět existuje množství technik, na které bude odkázáno odbornou literaturou [6].

- Vyhlazování [7]
- Ostření
- Prahování
- Kontrastové modifikace
- Odstranění přebytečných textur

Morfologické operace zahrnují rovněž množství operací.

- Operace s konturami
  Spojování přerušených či poškozených kontur, nebo naopak k jejich rozkládání, nebo vyhlazování.
- Odstranění bodů vzdálených od clusteru[1]
- Ztenčování čar
- Extrakce hranic
- Zarovnání

Fáze předzpracování produkuje v ideálním případě „čistý obraz“, který může být přímo a efektivně předložen pro rozpoznávání.

2.2 Rozpoznávání

Rozpoznávání je proces, který si s výhodou zavedeme pro potřeby této práce. Obecně se jedná o proces, ve kterém se vyskytují samostatné a velmi rozsáhlé fáze, proto se samostatně jako fáze neuvádí. Zde ale tato kategorizace umožní zavedení dvou metod přístupu k procesu rozpoznávání.

Fáze procesu rozpoznávání jsou selekce znaků[2], jejich extrakce a následná klasifikace. Je potřeba zdůraznit rozdíl mezi selekcí a extrakcí znaků, aby nedošlo k jejich zaměňování. **Selekce znaků** referuje k výběru algoritmů, kterými budou vybrány pouze selektivní rysy, tj. rysy zlepšující přesnost klasifikace. Naopak **extrakce** je spojena s technikou použitou k extrakci příznaků z reprezentace znaku (v kontextu OCR alfanumerického) jako takového. Pro OCR jsou typicky důležité všechny tyto fáze. Je třeba provést selekci znaků, tj. provést segmentaci a lokalizaci textových bloků. Tyto dále rozdělit na jednotlivé znaky, kde se uplatní extrakce příznaků, na jejímž základě pak probíhá klasifikace do tříd.

Jak bylo zmíněno, existují dva přístupy k procesu rozpoznávání. *Klasický přístup* a přístup s využitím technik *soft computingu* (viz 2.2.4 - Soft computing).

[1] *cluster* je v kontextu datové analytiky shluk objektů se společnými charakteristikami (v 2D grafu např. množina bodů blízko u sebe)

[2] (z anglického „*feature selection*“) – Zde pojmem „znak“ není míněn znak alfanumerický, ale obecný rys

Klasický přístup

Klasický přístup je abstraktní vzor, podle kterého byly v průběhy vývoje odvozeny další metody.
Techniky tohoto přístupu budou okrajově zmíněny, ale nejsou předmětem realizace této práce. Výjimku tvoří segmentace, kterou lze zahrnout do návrhu i v kombinaci s metodami soft computingu.

2.2.1 Segmentace

Segmentace klasického přístupu lze nazvat jako **explicitní**. Při explicitní segmentaci jsou jednotlivé segmenty identifikovány podle charakterových vlastností [2]. Opakem by byla segmentace **implicitní**, o které bude zmíněno více v sekci 2.2.4 - Soft computing. Je nutné poznamenat, že metody explicitní a implicitní segmentace lze kombinovat v tzv. *hybridní* segmentaci. Z uvedeného plyne, že i při použití druhého přístupu jsou využitelné oba způsoby segmentace.

Prahování

Prahování, v tomto kontextu také nazýváno jako *binarizace*, je proces segmentace pixelů do dvou skupin, bílé (pozadí) a černé (popředí) - nahrazení binární hodnotou. Cílem prahování je oddělit objekty od pozadí. Podstatou prahování je nahrazení hodnoty každého pixelu černou, pokud je jeho hodnota pod stanovenou prahovou hodnotou, nebo bílou v ostatních případech.
Zde je uveden pouze výčet metod, jejichž podrobnosti jsou uvedeny v [8].

- Metody hledání vhodného prahu
- Metody využívající tvar histogramu
- Metody založené na shlukování
- Metody využívající entropii obrazu

Detekce hran

Detekce hran je rodinnou segmentačních algoritmů. Obecně lze říci, že se sestávají ze dvou fází. V první fázi se jedná o nalezení hran a ve druhé fázi se z nalezených hran vytváří samotná segmentace a hledají se hranice vzniklých segmentů .
Metody pro detekci hran v obraze se vzájemně liší přístupem k obrazu a definicí

hrany. Z těchto rozdílností pak plynou různé výsledky, které si žádají různé interpretace. Lze ale říci, že každý postup vytváří jako svůj výsledek nový obraz, kterému se říká mapa hran [9].
Z významných metod detekce hran lze uvést následující výčet.

- Metody založené na první derivaci
- Metody založené na druhé derivaci
- Cannyho detektor hran [10]
- Srovnání se vzorem (*Template matching*)

2.2.2 Analýza rozložení textu v dokumentu

Analýzou rozložení textu [11] se zde myslí postupná lokalizace řádků, slov a znaků. Existují zde dva přístupy k jejich nalezení. Prvním je analýza souvislých komponent, druhým je vertikální (VPP) a horizontální projekce (HPP[3]). Výsledkem je projekce profilu jednotlivých řádků a slov. Každé slovo může pak dále být rozčleněno na jednotlivé znaky pomocí vertikální projekce.

Tvorba řádků postupem zdola nahoru je tradiční algoritmus pro extrakci textových objektů je založen na **analýze souvislých komponent** a skládá se z několika procedur, které si zde uvedeme. Protože je ale dle [11] tato technika pomalejší a méně efektivní, uvedeme si zde jen jednotlivé kroky algoritmu a pro podrobnější studii bude odkázáno na literaturu [12].

- Analýza souvislých komponent
- Filtrování komponent
- Určení sousedů
- Tvorba dočasných řádků
- Relaxace
- Identifikace komponent

Vertikální a horizontální projekce je druhý algoritmus využívaný pro segmentaci řádků a slov.

[3]HPP (*Horizontal projection profile*)

Segmentace řádků

Zde je využito profilování HPP. Výstupem HPP je histogram „ON[4]“ pixelů pro každý řádek textu. Na obrázku 2.1 [11] je znázorněn histogram, kde lze pozorovat výběžky a sedliny. Sedliny ukazují na mezery mezi řádky a mohou být snadno použity pro získání hraničních pixelů. V tomto bodě je segmentace řádků hotova.

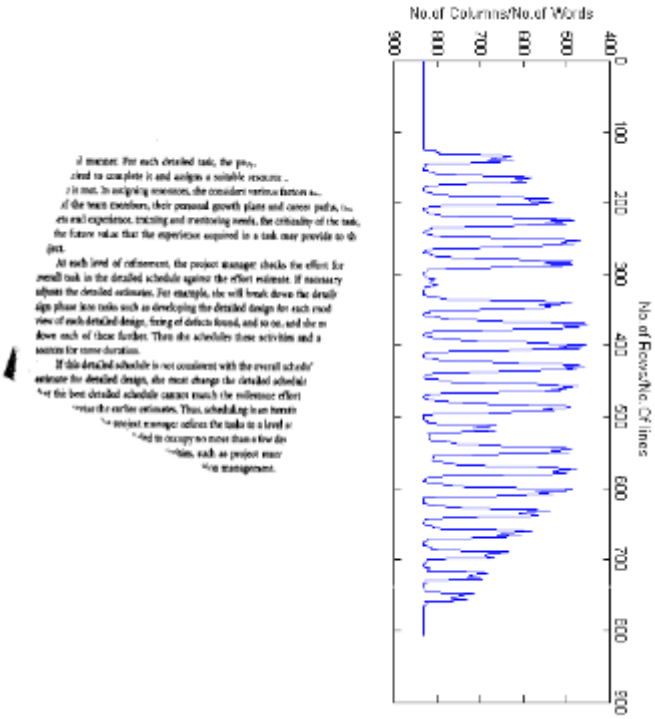
Segmentace slov

Vzhledem k tomu, že mezi každými dvěma slovy je také mezera, lze obdobný princip použít také pro extrakci slov. Zde je již ale nutné použít vertikální projekci. Jednotlivá slova pak mohou být separována spočítáním minima profilu ve vertikální projekci pro každý řádek obrazu, sedliny v histogramu nyní symbolizují mezery mezi slovy. Histogram vzniklý VPP je vyobrazen na obrázku 2.2 [11].

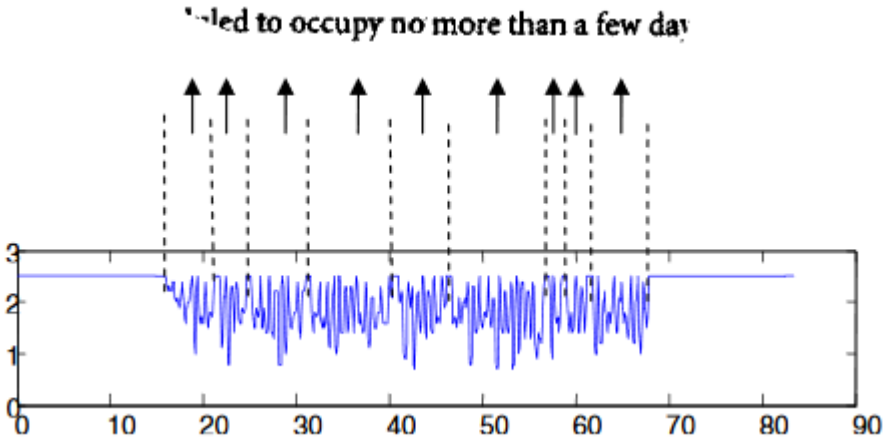
Segmentace znaků

Pro potřeby OCR je klíčová poslední fáze textové analýzy – segmentace znaků. Zde dochází k extrakci významných oblastí pro klasifikaci. V tomto kroku bude provedena dekompozice do klasifikovatelných jednotek. Lze zde opět použít obdobný postup jako u separace slov [11].

[4]podobnost s anglickým překladem „zapnuto“, tj. aktivní pixely, jejichž hodnoty způsobují histogramové výběžky – textové pixely



Obr. 2.1: Horizontální projekce pro segmentaci řádků



Obr. 2.2: Vertikální projekce pro segmentaci slov

2.2.3 Identifikace komponent

V předchozím textu byly v obraze identifikovány řádky a komponenty, které ho tvoří. Nyní je potřeba tyto komponenty seskupit do tříd. Těmito třídami budou základní komponenty, diakritika a interpunkční znaménka. K tomuto nám pomůžou referenční linky. Ty nám rozdělí řádek do tří zón a podle překryvu s daným znakem zařadíme znak do vyhovující třídy. Více informací se lze dočíst v [12].

2.2.4 Klasifikace znaků

Cílem této fáze je správně klasifikovat znak a zařadit ho do třídy znaků, tj. dle náročnosti aplikace, například jedná-li se o písmeno, číslo, interpunkci, určit jeho hodnotu v adekvátní třídě. Možností může být zařazení do dalších tříd, jako například jazyk či typ fontu. Klasický přístup využívá pro klasifikaci popis znaků pomocí určitých charakteristik, které nazývají *deskriptory*. Vlastnosti každého jednotlivého znaku musí být popsány těmito deskriptory a podle nich je poté znak zařazen. K zařazení se zde využívá maticového porovnávání, například podle nejmenších vzdáleností v prostoru deskriptorů mezi znakem a jeho vzorem.

Klasifikace znaků podle barvy pixelů

Nejprve se vygeneruje šablona znaků pro více fontů a stylů (kurzíva, tučný) a zvolí se společná velikost. Šablony jsou dále vyhlazeny Gaussovým filtrem s konvolučním jádrem o velikosti 3x3[5]. Stejný postup se použije na každý klasifikovaný vzorek a aplikuje se maticové porovnání *kritériem nejmenší vzdálenosti*.

Klasifikace znaků na základě vertikální polohy v řádku

Pro šablony z předchozího textu je určena další vlastnost, kterou je vertikální poloha znaku v řádku. Poslouží k tomu referenční linky (viz podsekce Identifikace komponent). Pro každý znak jsou určeny geometrické charakteristiky pomocí jeho minimální a maximální y-ové souřadnice. Podobně jako při předchozí klasifikaci aplikujeme kritérium nejmenší vzdálenosti. Tyto deskriptory nemají dostatečnou rozlišovací schopnost znaku, proto se používají v kombinaci s předchozími ke zpřesnění klasifikace.

[5] tento filtr odpovídá 32 bodovému obrazu

Kritériemi nejmenší vzdálenosti zmíněnými v předchozím textu můžou být např. *Nearest Neighbor Classifier*[6], nebo *K-Means clustering*.

Z uvedeného okamžitě plyne několik specifik tohoto přístupu rozpoznávání. Výhodou je, že tato metoda přístupu k OCR nevyžaduje velké množství trénovacích dat. Při splnění jistých omezení vstupu lze tuto metodu také považovat za poměrně rychlou. Podstatnější jsou pro nás ale omezení a nevýhody tohoto postupu.
Hlavní nevýhodou je velmi **nízká robustnost**, tj. míra jeho náchylnosti vůči změnám parametrů, jaké mohou být například vstupní data, což je klíčové pro korektní klasifikaci při běžném používání.
Další nevýhody můžeme vidět v omezenosti na **počet rozpoznatelných tříd** znaků, nebo nutnosti ručního definování deskriptorů a vlastností znaků. Rovněž potřeba velkého množství prototypů vede k vysokým nárokům na úložný prostor.
A konečně nelze tyto metody využít pro rozpoznávání velkého množství tříd a prakticky nelze aplikovat na **ručně psaný text** nebo text v **nehomogenním** prostředí[7].
Uvedené nevýhody se snaží řešit *soft computing*.

[6] V češtině se někdy uvádí jako „k-nejbližších sousedů“

[7] Za homogenní prostředí byl doposud považován list papíru, kde celé pozadí obsahovalo minimum barev

Soft computing

Soft computing je důležitým matematickým aspektem v doméně rozpoznávání vzorů a umělé inteligence. Je obvykle uváděn v kontextu lidského uvažování, protože se svou funkcí snaží přiblížit lidskému mozku.
Využití technik soft computingu pro OCR je v současnosti nejmodernější metodou. Snahou je zde vyřešit problém robustnosti, kvality, rychlosti a kompletnosti řešení implementací technik strojového učení.

Unikátnost soft computingu spočívá v tom, že na základě výsledků, které jsou derivovány z experimentálních dat, dokáže generovat výstup i pro neznámá vstupní data.
Na základě vstupních dat si systém vypracuje vlastní *reprezentaci vzoru*, kterou dále používá pro klasifikaci. Celý výpočetní proces soft computingové soustavy lze popsat jako transformaci z modelového prostoru M do prostoru řešení S a konečně do rozhodovacího prostoru D [13].

$$M \rightarrow S \rightarrow D \tag{2.1}$$

Proces, při kterém jsou systému předkládána vstupní data k a systém se je „učí“ korektně klasifikovat, se nazývá *trénování*. U většiny komerčních OCR systémů je proces trénování proveden předem, tomuto se říká *offline learning*. Pokud systém obsahuje prvky učení na základě dat přijatých v reálném čase, nazveme toto učení jako *online learning*. Protože techniky soft computingu jsou v současnosti velmi slibným kandidátem pro splnění požadavků robustnosti, kvality a nízké ceny, bude v této práci věnována této metodě pozornost a bude rovněž základem vlastního návrhu.
Hlavními pilíři soft computingu jsou **strojové učení**, fuzzy logika, pravděpodobnostní usuzování, evoluční algoritmy a umělé neuronové sítě (ANN) [2].

Učení bez učitele

V oboru strojového učení je výkonnost modelu často silně ovlivněna způsobem, jakým jsou data reprezentována [14]. Efektivní reprezentace dat je tedy důležitým aspektem pro konstrukci efektivních modelů. Přestože ruční vytváření takovéto reprezentace je způsob, jak se s problémem vypořádat, mnohdy není efektivní a ve většině případů je také náročný a nákladný.
Algoritmy, které nepotřebují učitele (*unsupervised feature-learning algorithms*), se dokáží naučit správnou reprezentaci s neoznačených dat. Jsou schopny generovat

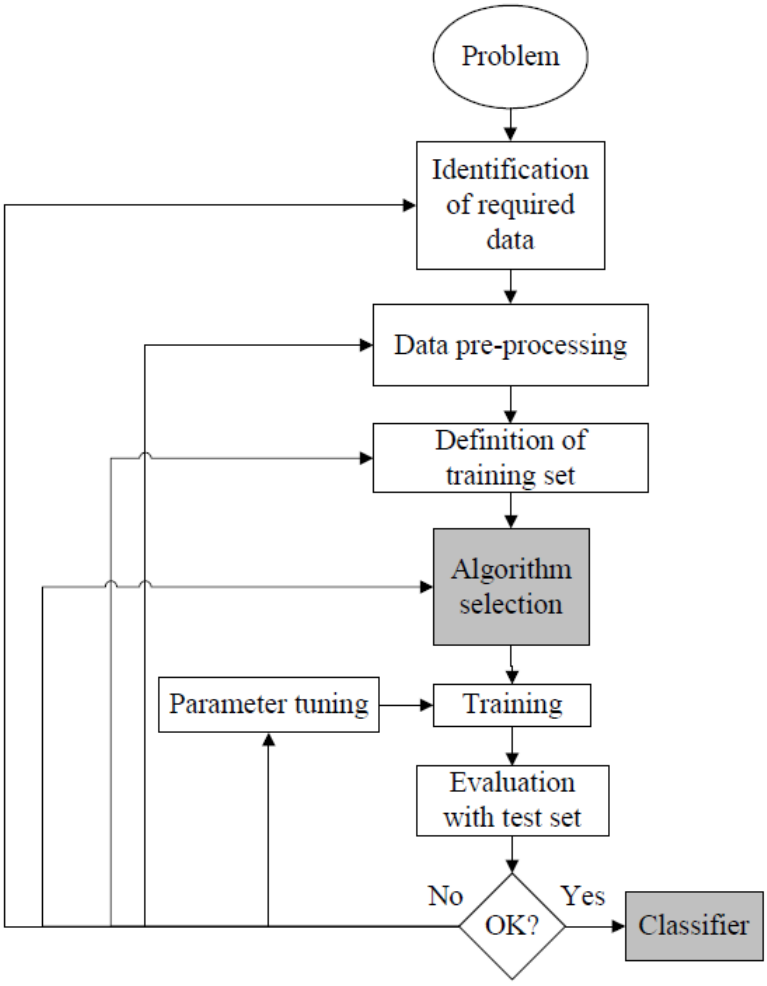
bohatou množinu znaků, které by jinak nebylo možné ručně vyvinout. Tyto algoritmy jsou aplikovány v různých sférách soft computingu a strojového učení, jako např. počítačové vidění, zpracování dat, analýza trendu, clustering, ale jsou i použitelné k rozpoznávání textu. Naneštěstí pro tyto aplikace, jejich nevýhodou je vysoká výpočetní složitost a obtížné rozšíření. Pravděpodobně nejznámější a velmi používaný je algoritmus *K-Means*, popsán zde [15].

Učení s učitelem

Pokud trénovací data jsou tzv. *oštítkovaná*[8], lze pro každý trénovací vstupní vzor speficikovat třídu, které má vzor náležet. Tedy žádaný výstup trénovaného systému lze korigovat učícím algoritmem, který přizpůsobuje parametry tak, aby nejlépe vyhovovaly korektnímu štítku. Tyto úpravy jsou prováděny inkrementálně (viz obr. 2.3 [16]) minimalizováním chyby mezi predikovaným výstupem sítě a korektním výstupem v požadovaném směru. Zpravidla se pro minimalizování chyby využívá *stochastických gradientních metod* (viz kap. Neuronové sítě) a hyperparametrů zaručujících konvergenci dané metody.

OCR lze při využití soft computingového přístupu rozdělit na dvě fáze – **klasifikace a detekce**.

[8] v praxi se používá anglický termín *labeled data*



Obr. 2.3: Diagram procesu učení s učitelem

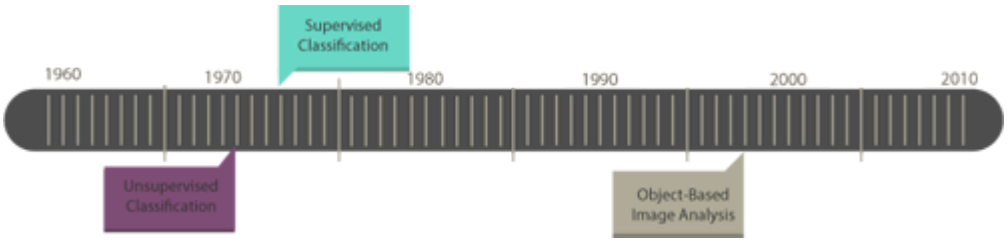
2.2.5 Klasifikace

V kontextu neuronových sítí je klasifikace problémem jednodušším, než-li detekce. Fáze klasifikace zde bude uvedena před samotnou detekcí. Důvodem tohoto uspořádání je přístup soft computingových metod k lokalizaci a detekci objektů. Využívají totiž klasifikaci jako součást detekce (viz podsekce 2.2.6 – Detekce). Široké spektrum klasifikačních metod lze obecně seskupit do čtyř širších kategorií[17]:

- Statistické metody
- Metody využívající ANN
- Kernelově založené metody
- Metody využívající kombinace více klasifikátorů

Pro účely této práce ale využijeme obecnějšího rozdělení tak, jak bylo zavedeno v úvodu této sekce (2.2.4), rozšířené o objektový model. Dle časového sledu lze tyto kategorie seřadit do tří časových fází. Orientační obrázek časové linie je na obr. 2.4 [18].

- Klasifikace bez učitele (*Unsupervised Classification*)
- Klasifikace s učitelem (*Supervised Classification*)
- Objektově založená analýza obrazu[9] (*Object-Based Image Analysis*)



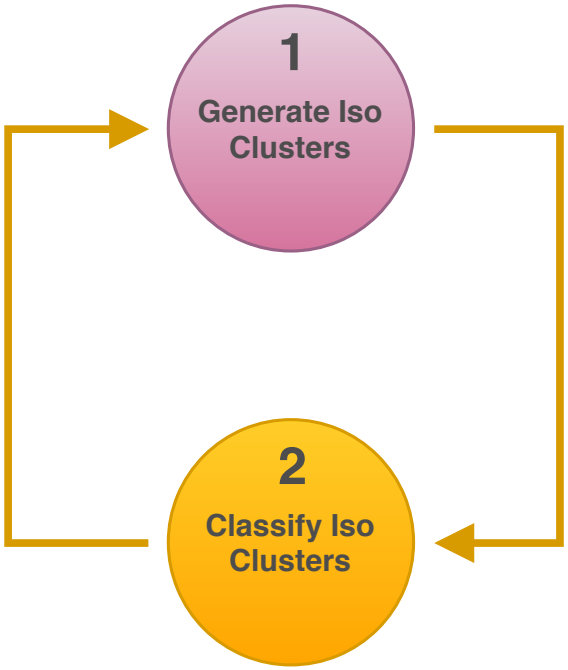
Obr. 2.4: Časová linie klasifikačních metod

Klasifikace bez učitele

Prvním krokem pro klasifikaci bez učitele je vytvoření *„clusteru“* (viz poznámka v sekci 2.1 – Preprocessing) podle vlastností a charakteristik daného obrazu. K vytvoření clusteru jsou využívány algoritmy *K-Means Clustering*[19] nebo HCA [20]. Po vybrání vhodného algoritmu je nezbytné zvolit vhodné hyperparametry modelu, uvažujeme-li clustering, jedná se o počet generovaných grup, který je v ideálním případě ekvivalentní s počtem tříd určených pro klasifikaci. Každý cluster tedy jednoznačně identifikuje korespondující třída.

[9] Někdy také nazývána jako „objektově založená klasifikace“

Proces klasifikace bez učitele lze popsat diagramem znázorněném na obr. 2.5 [18].



Obr. 2.5: Diagram klasifikace bez učitele

Klasifikace s učitelem

Prvním krokem je zde vytvoření kvalitních dat. Toto vyžaduje jistou profesionalitu, protože některé informace mohou mít větší informační hodnotu, než jiné. Naopak by data mohla obsahovat irelevantní atributy, které by mohly negativně ovlivňovat učení, způsobovat šum, či víceznačnost (např. špatně označená třída). Je tedy potřeba podrobit data samotnému preprocessingu [21]. Takto zpracovaný set se poté většinou rozděluje na dvě vzájemně exkluzivní části - trénovací a testovací. Trénovací část pak bývá dále rozdělena na několik trénovacích a validačních částí souhrnně nazývaných *folds*[10]. Kritickým krokem je pak výběr učícího algoritmu. Zajímá nás časová i prostorová složitost, ale i přesnost predikce.

Z klasifikátorů založených na logickém rozhodování lze jmenovat *Decision Trees* - rozhodovací stromy. Velmi známé a používané jsou také *Bayesovské sítě*, či *Naive Bayes* klasifikátor, KNN a SVM. Tyto algoritmy jsou založeny na statistických a pravděpodobnostních metodách. O těchto algoritmech se čtenář může dočíst v odborné literatuře, např. v [16]. Naposled pak v současné době nejvýznamnější a dosud

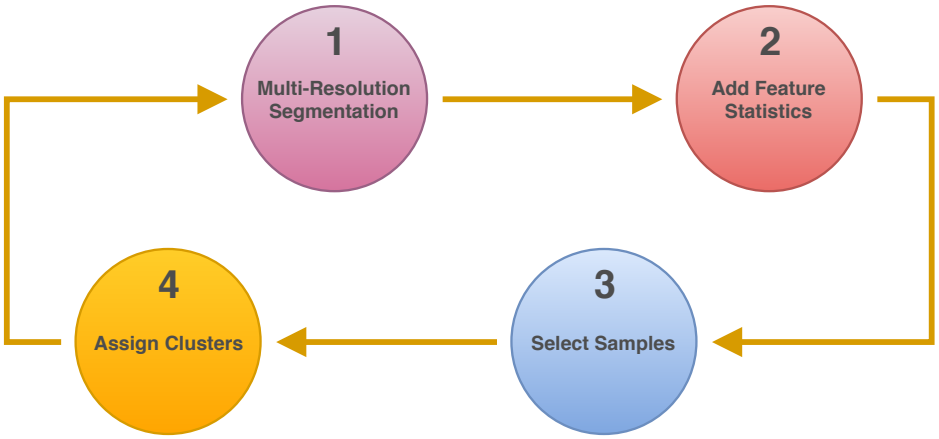
[10]Od tohoto pojmu je rovněž odvozen termín *k-fold cross-validation* – křížová validace

aktivně zkoumané ANN, kterým bude věnována samostatná kapitola (viz Neuronové sítě - 3).

Objektově založená klasifikace

Klasifikace s učením i bez učitele jsou pixelově orientované, tj. každému pixelu v obraze přiřazuje třídu nezávisle na prostorové informaci o objektu, který tento pixel obsahuje. Při objektově založené klasifikaci jsou pixely shlukovány do objektů reprezentativních tvarů a velikostí. Objekty jsou simultánně generovány a škálovány. Tento proces je v podstatě segmentace obrazu, což je proces o poznání složitější, než běžná klasifikace. Takto můžeme přistupovat například k hledání homogenních objektů v obraze. Tyto objekty jsou významné, protože reprezentují příznaky obrazu [18].

Dalším velmi důležitým přínosem je možnost klasifikovat objekty podle textury, geometrie, nebo kontextu. Objektově založená klasifikace (viz obr. 2.6 [18]) využívá modifikované učící algoritmy, z nichž nejvýznamnější je *Nearest Neighbor Classification* [22].



Obr. 2.6: Diagram objektové klasifikace

2.2.6 Detekce

Toto je nejkomplikovanější fáze a v současnosti stále probíhá aktivní výzkum týkající se lokalizace, segmentace a detekce objektů v obraze s využitím metod soft computingu. Jak bylo zmíněno v sekci 2.2 - Klasický přístup, lze v této fázi použít jednu ze segmentačních technik klasického přístupu. Nicméně cílem této práce je

dodržet robustnost celého projektu a navrhnout komplexní koncept. Proto zde bude pro lokalizaci využito techniky na bázi **implicitní** segmentace. Nechť množina tříd, kterou je schopen systém rozeznat, je jeho *abecedou*[11]. Při implicitní segmentaci, někdy také nazývané jako „interní“ [23], jsou v obrazu vyhledávány znaky, které odpovídají některé ze tříd dané abecedy. Tuto segmentaci lze prohlásit za segmentaci založenou na rozpoznávání.

Obraz bude prohledáván a budou v něm hledány komponenty a rysy, které odpovídají předdefinovaným třídám. Tento přístup lze demonstrovat na metodě pohyblivého okna.

Výstupem detekce jsou predikované hraniční boxy, tzv. *bounding boxes*. Jsou to obdélníkové výřezy obsahující jednotlivé znaky, případně dle implementace celá slova. Tyto jsou dále vstupem do klasifikační fáze.

K problematice lokalizace a detekce objektů lze přistupovat dvěma způsoby, jako ke klasifikačnímu problému, nebo k regresnímu problému. Uvažujeme-li **klasifikační přístup**, je obraz rozdělen na menší části, které lze volným překladem anglického *patches* nazvat jako *políčka*. Na každé z těchto polí je pak aplikován klasifikátor, který zjistí, zda objekt je, či není, přítomen. Hraniční boxy jsou pak přiřazeny pouze těm polím s kladným klasifikačním výstupem. Mezi techniky klasifikačního přístupu patří Metoda pohyblivého okna a RCNN (*Region Proposal Convolutional Neural Network*). Zmenšují tím množinu vyhledávacího prostoru a snižují výpočetní složitost.

Při **regresním přístupu** projde konvoluční neuronovou sítí celý obraz a jeden či více bounding boxů jsou generovány pro objekty v obrazu. Z uvedeného je zřejmé, že regresní přístup vykazuje nižší výpočetní složitost, neboť konvoluce probíhá pouze jednou. Mezi techniky regresního přístupu lze zařadit Single shot detector a YOLO (*You Only Look Once*) detektor.

Metoda pohyblivého okna

Tato metoda je, podobně jako řada technik a algoritmů umělé inteligence a strojového učení, ideologicky spjata s funkcí lidského oka. Princip si lze představit jako pohyb „zorného pole“ po obraze. Každý „pohled“ vytvoří výřez, ze kterého jsou extrahovány příznaky.

[11] Tento pojem je poměrně intuitivní, jelikož v kontextu OCR je rozpoznávanou třídou zejména alfanumerický znak

Konvoluční neuronové sítě

Na nejnižší úrovni jsou konvoluční neuronové sítě pouze hierarchické vícevrstvé neuronové sítě, které ideologicky souvisí s metodou pohyblivého okna tím, že uplatňují matematickou operaci, *konvoluci*. Tento princip a další podrobnosti budou detailněji objasněny v kapitole 3 - Neuronové sítě. Mezi významné konvoluční architektury při detekci objektů patří RCNN. RCNN jsou konvoluční neuronové sítě využívající predikci regionů, v nichž se s velkou pravděpodobností bude vyskytovat nějaký objekt.

Single shot detector

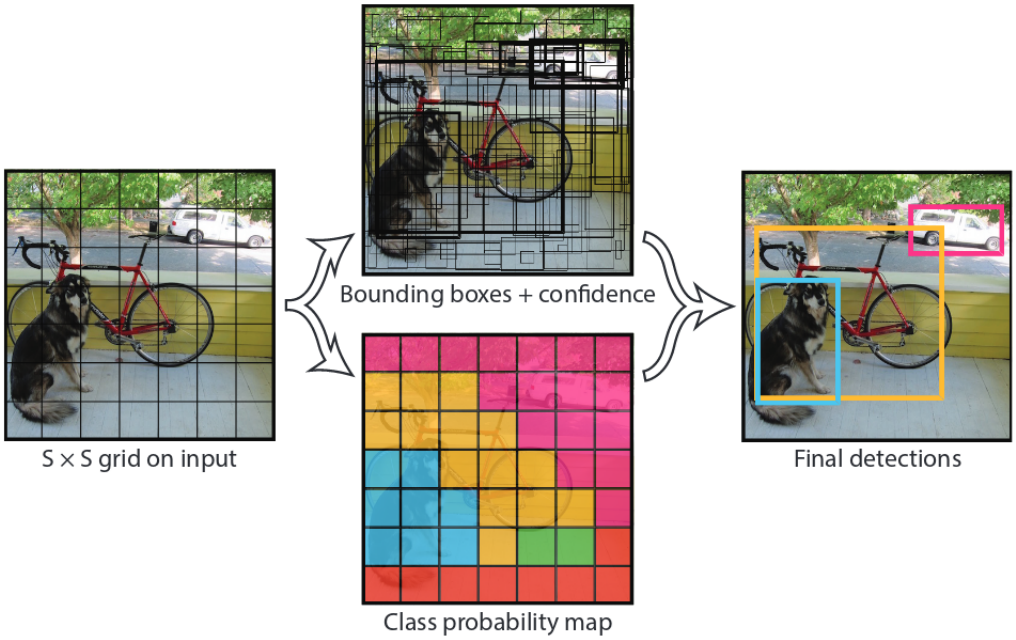
SSD (*Single Shot Detector*) [24] kompletně eliminuje generování predikčních regionů a zapouzdřuje celý proces do jediné neuronové sítě. Díky této implementaci lze SSD snadno trénovat i integrovat. V predikční době síť generuje skóre pro přítomnost kategorizovaných objektů v předdefinovaných boxech, poté produkuje úpravy a posuny k těmto hraničním boxům, aby lépe vyhovovaly pozici objektu. Navíc sít kombinuje několik map různých rozlišení, aby mohla přirozeně zvládat různé velikostní poměry objektů.

You only look once

Tento přístup k detekci objektů byl představen v publikaci [25] v roce 2015. V doslovném překladu znamená „podíváte se jen jednou“, a toto pojmenování dokonale vystihuje podstatu myšlenky, která dala vznik YOLO detekci.

YOLO modeluje detekci jako regresní problém. Obraz je rozdělen do mřížky(viz obr. 2.7 [25]) $S \times S$ a pro každou buňku mřížky predikuje B hraničních boxů, jistotu daného boxu a pravděpodobnost C dané třídy. Tyto predikce jsou poté uloženy do $S \times S \times (5 \cdot B + C)$ tensoru[12].
YOLO prezentuje přístup, při kterém projde obraz konvoluční sítí pouze jednou. Jelikož je celá detekce realizovaná jedinou neuronovou sítí, je tato architektura extrémně rychlá, snadno optimalizovatelná a lze ji s výhodou využít například při detekci objektů v reálném čase (video, kamerový záznam). Dokáže totiž rozpoznávat objekty při rychlosti neuvěřitelných 155 fps[13].

[12]Každý bounding box se skládá z 5 složek - x, y souřadnice, šířka, výška a hodnota jistoty
[13]frames per second - snímků za vteřinu



Obr. 2.7: YOLO (*You Only Look Once*) model

2.3 Post-processing

I přes značný pokrok v OCR analýze dokumentů se nedaří dosáhnout dokonalé přesnosti rozpoznávání. Ať už z důvodu nedokonalosti kvality vstupu, nebo nedostatků v klasifikátoru, je zpravidla nutné provést korekci výstupu. Tato korekce se sestává ze dvou hlavních modulů. Lexikální a kontextové korekce, souhrnně často nazývané **lexikální post-processing**. Lexikální post-processing je dodatečné zpracování rozpoznaného textu na lexikální úrovni. Snaží se vybrat z množiny všech možností právě jeden jednoznačný výsledek a ověřit jeho správnost nebo dodatečně opravit chyby tohoto výsledku [12].

Lexikální korekce

Rozpoznané slovo není kvůli podobnostem znaků vždy jednoznačné. V některých případech se potom může u slova vyskytnout několik variant, jak bude rozpoznané slovo vypadat. Lexikální korekce využívá slovníku daného jazyka pro srovnání slov a dalších statistických dat. Na základě těchto dat zkoumanou variantu buď zamítne, nebo, v závislosti na implementaci, ohodnotí jednotlivé varianty určitými pravděpodobnostmi. Určit výsledný text pak například může být na uživateli samotném, který z ohodnocených variant vybere tu správnou. Nejčastěji se ale zvolí slovo s nej-

vyšší pravděpodobností. Pokud nastane situace, že jsou odmítnuty všechny varianty, zvolí se pomocí slovníku nejbližší možné korektní slovo.

Kontextová korekce

Na rozdíl od lexikální korekce, která pouze porovnává jednotlivá slova se slovníkem, kontextová korekce bere v úvahu souvislý text a využívá syntaktickou a sémantickou znalost daného jazyka. Znalost o jazyce lze zachytit ve struktuře, která se nazývá jazykový model. V oblasti modelování jazyka se uvažuje *formální* a *stochastický jazykový model* [12].

Formální jazykový model vychází z Chomského formální teorie jazyka. Jsou pro něj podstatné dvě věci – gramatika a parsovací algoritmus. Parsování je analýza, která odhaluje, jestli skladba věty vyhovuje pravidlům dané gramatiky. Formální model přiřazuje posloupnosti slov hodnotu 0 nebo 1, pokud nepatří či patří do daného jazyka.

Stochastický jazykový model vychází z pravděpodobnosti, která vyjadřuje vztahy mezi posloupnostmi slov. Oproti formálnímu modelu, přiřazuje posloupnosti slov hodnotu v rozmezí 0 až 1, tj. pravděpodobnost, s jakou daná posloupnost patří do daného jazyka. Proto se lépe vypořádává se všeobecným textem.

3 NEURONOVÉ SÍTĚ

Naneštěstí neexistuje univerzální, všeobecně akceptovaný popis neuronové sítě [26]. Hlavní koncept však zůstává stejný, a proto zde bude přiblížen.
V této kapitole bude přiblížen model neuronu umělé neuronové sítě, bude objasněna jeho funkcionalita a matematické pozadí predikce výsledků a trénování neuronových sítí. Vyjmenovány budou nejvýznamnější architektury neuronových sítí. Pozornost bude soustředěna na konvoluční sítě, jež budou stavebním kamenem konceptu realizace OCR.

Neuronové sítě dostaly své pojmenování kvůli své ideologické podobnosti s funkcí mozku, resp. sítě mozkových neuronů. ANN v současnosti zdaleka nedosahují komplexnosti lidského mozku. Jsou tady nicméně jisté podobnosti mezi biologickými a umělými neuronovými sítěmi. Stavebními kameny obou sítí jsou jednoduché výpočetní elementy provádějící relativně jednoduché operace[1]. Rovněž, spojení mezi neurony rozhoduje o funkcionalitě celé sítě. Analogie s lidským mozkem může být v mnoha případech zavádějící. Pro pochopení konceptu je nicméně velmi příhodná, a proto je jí i zde využito.

3.1 Model neuronu

3.1.1 Biologický popis neuronu

Neuron je základní výpočetní jednotka. Typický neuron v mozku člověka či jiného živočicha zpracovává signály od ostatních neuronů a v závislosti na množství určitých faktorů generuje elektrický signál - *akční potenciál*, který je pak přenášen přes *synapse* připojeným neuronům. Každý neuron signál přijme svými *dendrity* (viz obr. 3.1 [27]) a produkuje výstupní signál do *axonu*.

3.1.2 Matematický popis perceptronu

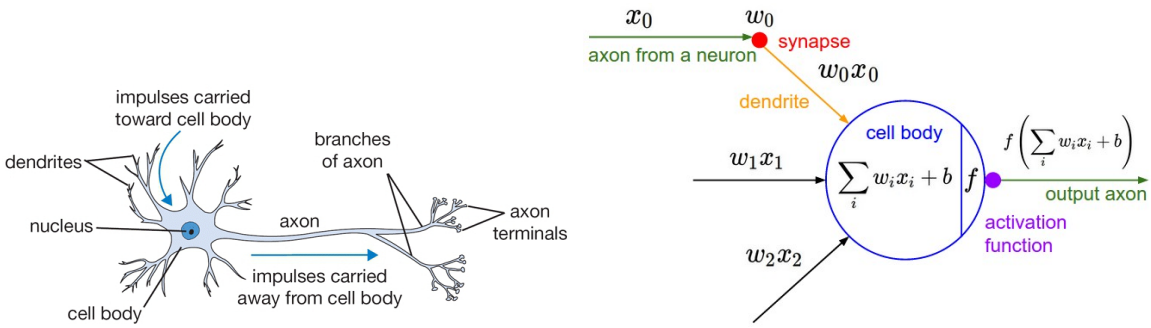
V kontextu neuronových sítí bývá matematický model neuronu označován jako *perceptron*. Pojem perceptron a neuron jsou zpravidla zaměnitelné a i zde budou oba využívány. Signály x_j od ostatních perceptronů jsou jeho *vstupy*. Ke každému vstupu je přiřazena jeho *váha* w_j. Tu lze interpretovat jako „důležitost“, s jakou je daný

[1] Přestože nejsme schopni s jistotou rozhodnout o podobě a složitosti operace, v porovnání s činností celého mozku jako celku ji lze prohlásit za jednoduchou.

vstupní podnět registrován. Kdybychom zacházeli do detailů, mohli bychom interpretovat i váhu podle znaménka jako aktivátor (kladné znaménko) a inhibitor (záporné znaménko). Signál interaguje multiplikativně s dendrity na základě síly synaptického spojení - $w_0 x_0$. Funkci perceptronu lze modelovat matematickou funkcí, která produkuje *výstupní hodnotu* y. V nejjednodušším případě by se jednalo o vážený součet vstupů.

$$y = \sum_{j=1}^{d} w_j x_j + b \tag{3.1}$$

b (někdy w_0) je hodnota označovaná jako „*bias*“. Slouží ke zobecnění modelu a obecně je modelována jako váha pocházející z dodatečné biasové jednotky a její hodnota je obvykle +1 [28]. Tato funkce se nazývá *aktivační funkce*. Aktivačních funkcí existuje několik a v současnosti se výzkum stále zaměřuje na analýzu stávajících a nalezení nových funkcí, které by byly vhodnější pro využití v ANN. O dalších typech aktivačních funkcí se lze dočíst v [27].



(a) biologický model (b) matematický model

Obr. 3.1: Modely neuronu

3.2 Neuronová síť

Samotný perceptron má pouze jednu vrstvu vah a dokáže tak aproximovat pouze lineární funkce. Je zřejmé, že většina reálných problému nemá lineární řešení. Toto omezení ale neplatí pro *dopředné neuronové sítě* s tzv. *hidden layers*, tedy skrytými[2]

[2]Toto označení si lze spojit s faktem, že tyto vrstvy jsou pro běžného uživatele opravdu skryté, interaguje pouze se vstupní a výstupní vrstvou

vrstvami. Dopředná neuronová síť nese označení MLP (*Multi Layer Perceptron*) a může aproximovat nelineární funkce.

3.2.1 Trénování neuronových sítí

V následujícím popisu bude uvažován *online learning* (viz sekce Soft computing), který nepracuje s kompletním vzorkem, ale pouze s instancemi, které budou předkládány perceptronu jedna po druhé. Perceptron se tak bude v čase sám adaptovat. Výhody tohoto přístupu [28] jsou např.:

- snížené nároky na okamžité úložiště vzorku a mezivýsledků při optimalizaci.
- snadná adaptace na vzorek, který se mění v čase
- snadná adaptace na ostatní v čase proměnné veličiny[3].

Perceptron definuje hyperrovinu, jíž je možné rozdělit data tak, aby je bylo možné klasifikovat. Počáteční hodnoty vah jsou inicializovány zcela náhodně. V každé iteraci jsou pak přizpůsobovány parametry[4] za podmínky minimalizování chyby. Snahou je rovněž „nezapomenout to, co se již perceptron naučil“, tedy volit správné parametry učení.

Úprava vah (a tedy trénování neuronové sítě) se nazývá *back propagation* a využívá gradientní metody, která se nazývá *stochastic gradient descent* - volně přeloženo jako **stochastický gradientní sestup**. Gradient aktivační funkce udává směr (reprezentován znaménkem) a velikost, o kterou má být příslušná váha změněna pro dosažení optimálnější hodnoty v dané iteraci. Tedy namísto okamžitého hledání nejlepšího řešení, dosahuje se cíle postupnými kroky. Pokud model pracuje správně, měl by gradient konvergovat k nule.

Pro zavedení gradientu je třeba definovat chybovou funkci[5], tj. chyby, kterou se neuronová síť dopouští při klasifikaci. Cílem procesu učení je tuto chybu minimalizovat. Existuje více typů funkcí specifických vlastností, které se používají pro různé druhy úloh. Pro aktivační funkci (3.1) zvolme střední kvadratickou odchylku[28]:

$$E^t(w|x^t, r^t) = \frac{1}{2}(r^t - y^t)^2 = \frac{1}{2}[r^t - (\mathbf{w}^T\mathbf{x}^t)]^2 \tag{3.2}$$

[3]V robotice může nastat degradace některých komponent či sensorů.

[4]v kontextu jednoho neuronu chápáno jako váhy, u neuronových sítí se jedná o množinu hyperparametrů

[5]V anglické literatuře je tato funkce nazývána jako *cost function* nebo *loss function*, lze se ale setkat i s pojmem *error function*. Pojem „loss“ se používá i v české literatuře.

Matematické odvození a notaci si lze přečíst v [29]. Ve stručnosti lze vyzdvihnout, že v případě naší jednoduché sumační funkce bude gradient, o který budeme měnit váhu w_j vypadat takto:

$$\Delta w_j^t = \eta(r^t - y^t)x_j^t \tag{3.3}$$

3.2.2 Architektury neuronových sítí

Přehled architektur ANN se rozrostl exponenciálně a tento růst stále pokračuje. Většina současných architektur je zobrazena na obrázku 3.6 [30]. V této podsekci bude přiblížena pouze architektura a princip konvolučních neuronových sítí, která je stěžejní pro účely této práce. Bližší informace si čtenář může přečíst v [31].

Konvoluční neuronové sítě

CNN získaly na popularitě v roce 2012, kdy Alex Krizhevsky vytvořil model, který vyhrál prestižní soutěž ImageNet [32]. V současnosti je beze sporu hlavní využití CNN v rozpoznávání obrazu a počítačovém vidění. Stejně jako pro obecný koncept neuronových sítí, i zde lze najít analogii s biologickými procesy. Myšlenka CNN je částečně založena na experimentu, který v roce 1962 provedl Hubel a Wiesel, a v němž zjistili, že některé neurony reagovaly na podnět pouze v přítomnosti hran jisté orientace. Toto vede k myšlence specializovaných komponent uvnitř jednotného systému. Další biologický motiv je mnohem zřejmější a bude pomocí něj vysvětlena podstata samotné konvoluce.

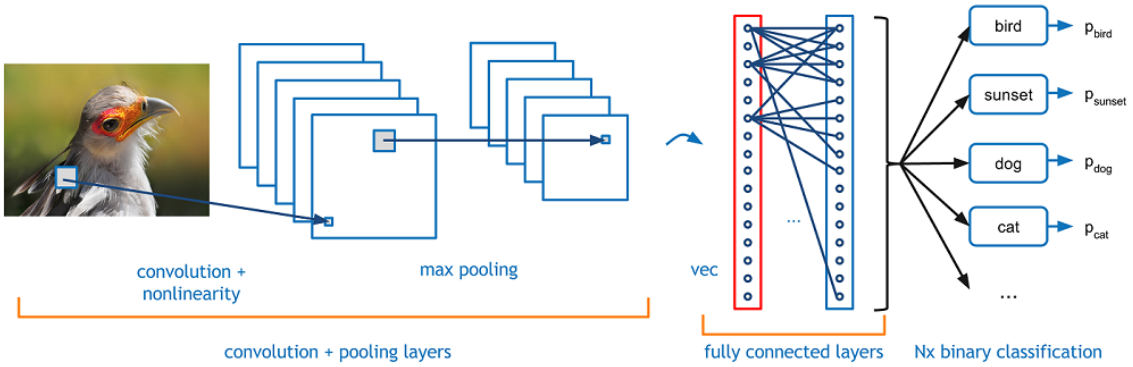
CNN se sestává ze čtyř hlavních operací ukázaných na obr. 3.2 [32]:

- Konvoluce
- Aplikace nelinearity (ReLU)
- Pooling nebo sub-sampling
- Klasifikace

V dalším textu bude přiblížen význam jednotlivých operací.

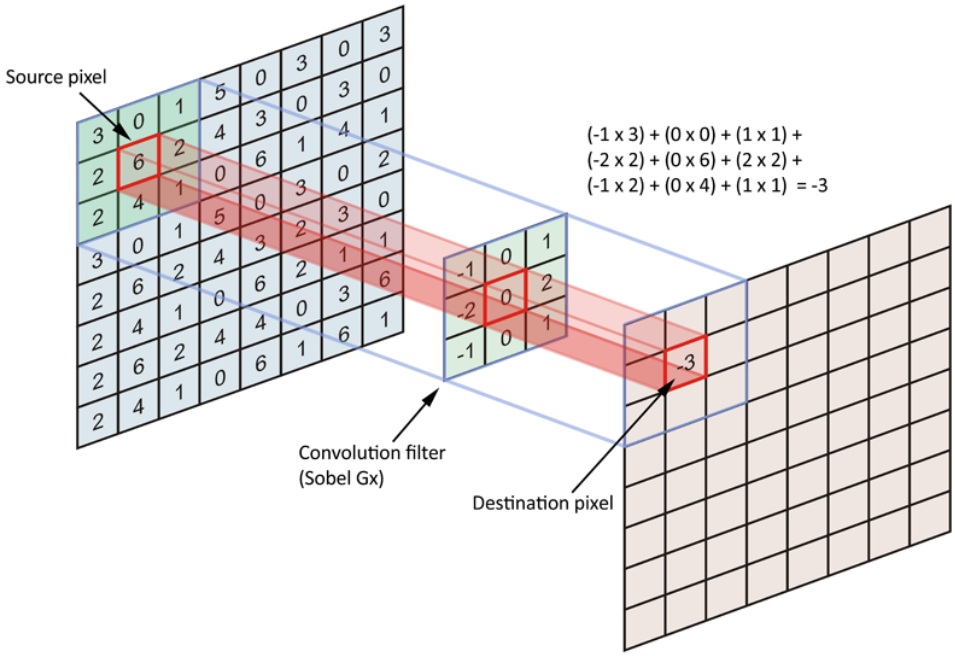
Konvoluce

Konvoluce je matematická operace, která spojuje dvě množiny informací. Matematický princip konvoluce je nad rámec této práce. Postačí nám ovšem intuitivní představa. Primárním cílem konvoluce je extrakce příznaků ze vstupního obrazu.



Obr. 3.2: Proces klasifikace konvoluční neuronovou sítí

Konvoluce zachovává prostorovou informaci mezi pixely. Každý obraz lze interpretovat jako matici pixelů. Proces konvoluce znamená vytvoření *konvolučního jádra*, často nazývaného jako *filtr*, tj. druhé matice menších rozměrů (typicky matice 3 x 3). Nyní lze konvoluci přirovnat k metodě pohyblivého okna (viz sekce Soft computing - Detekce). Jádro představuje pohyblivé okno, které je posouváno po původním obraze a na každém místě provede skalární součin sama sebe s částí, kterou překrývá (viz obr. 3.3 [33]).



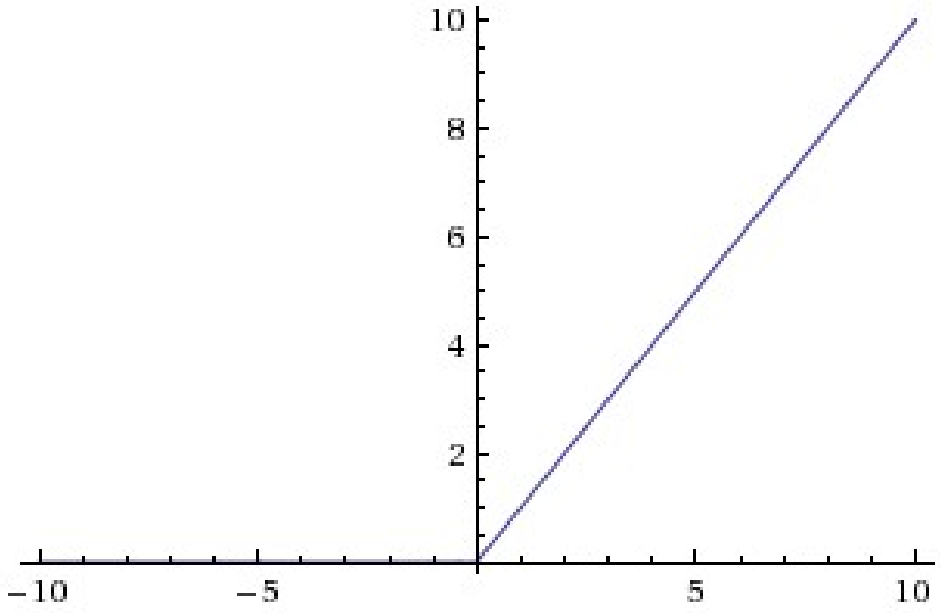
Obr. 3.3: Konvoluce

Aplikace nelinearity

Aby neuronová síť mohla fungovat, musí obsahovat prvek nelinearity. U CNN se jedná o aktivační funkci zvanou ReLU (*Rectified Linear Unit*). Tato funkce získala na popularitě v posledních několika letech, protože výrazně zrychluje konvergenci gradientu v porovnání se sigmoid nebo tanh funkcemi a trpí méně na problém mizejícího gradientu [34].

$$f = max(0, x) \tag{3.4}$$

Z matematického zápisu (3.4) a obr. 3.4 [27] je vidět, že ReLU vynuluje všechny záporné hodnoty a ostatní hodnoty ponechá beze změny.



Obr. 3.4: Aktivační funkce ReLU

Pooling a sub-sampling

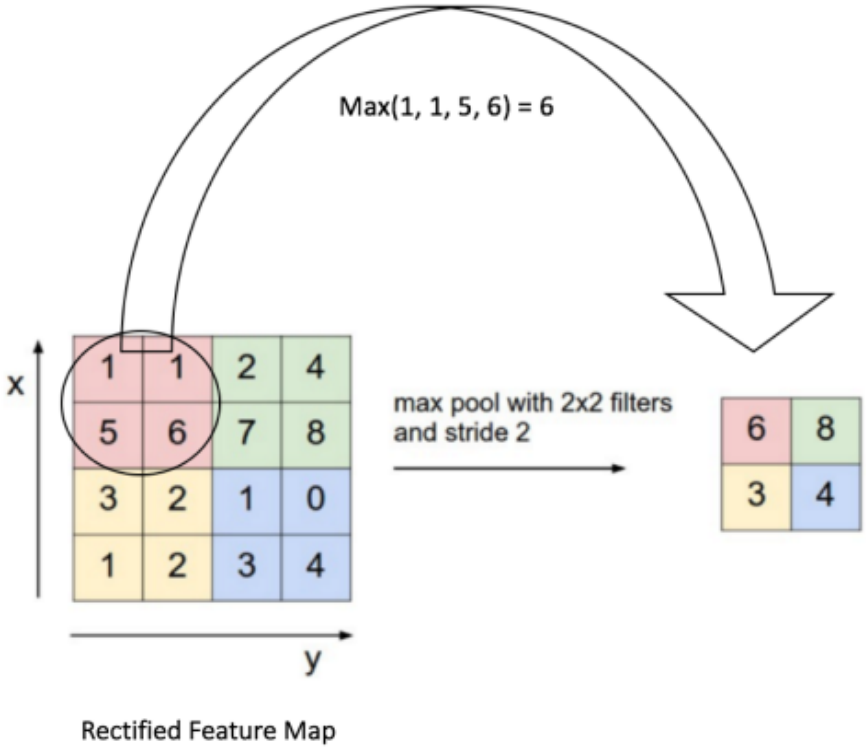
Vrstva CNN určená k redukovaní prostorové dimenze při zachování hloubky sítě se nazývá *Pooling layer*[6], v češtině bude tento termín zachován. Tento krok je důležitý z několika důvodů [35]:

- Snížením prostorové informace se zvýší výpočetní výkon sítě

[6]Často se tomuto kroku také říká *subsampling* či *downsampling*.

- Menší množství informace znamená rovněž méně parametrů, tedy menší nebezpečí *overfittingu*[7].
- Získání částečné prostorové invariance

Podstatou poolingu je redukce pomocí filtru (zde je často používán 2 x 2 filtr) tak, že se část tensoru, kterou filtr překrývá, zredukuje na jednu hodnotu dle určitého typu poolingu. Typem může být například *max pooling* či *average pooling*. Princip max poolingu je zobrazen na obr. 3.5 [36].



Obr. 3.5: Max Pooling pomocí filtru 2 x 2

Klasifikace

Doteď byly ukázány základní stavební bloky konvoluční neuronové sítě. Je-li uvažován návrh sítě z obr. 3.2, jsou postupně provedeny operace konvoluce pomocí 5 konvolučních jader[8] a aplikace nelineární aktivační funkce. Poté je na každou vrstvu aplikován pooling. Tyto vrstvy společně extrahovaly příznaky ze vstupního obrazu,

[7]Overfitting nastává, pokud jsou známá data aproximována „příliš přesně“, a aproximační polynom tak ztrácí schopnost aproximovat data neznámá.

[8]Je zřejmé, že hloubka tensoru vzniklého konvolucí je 5 - 5 vrstev - tedy jedna vrstva pro každé jádro.

zavedly do modelu nelinearitu a zredukovaly dimenzi výsledného tensoru, přičemž získaly jistou invarianci vůči škálování a translaci. V tomto okamžiku jsou extrahovány příznaky, na základě kterých je možné provést klasifikaci.

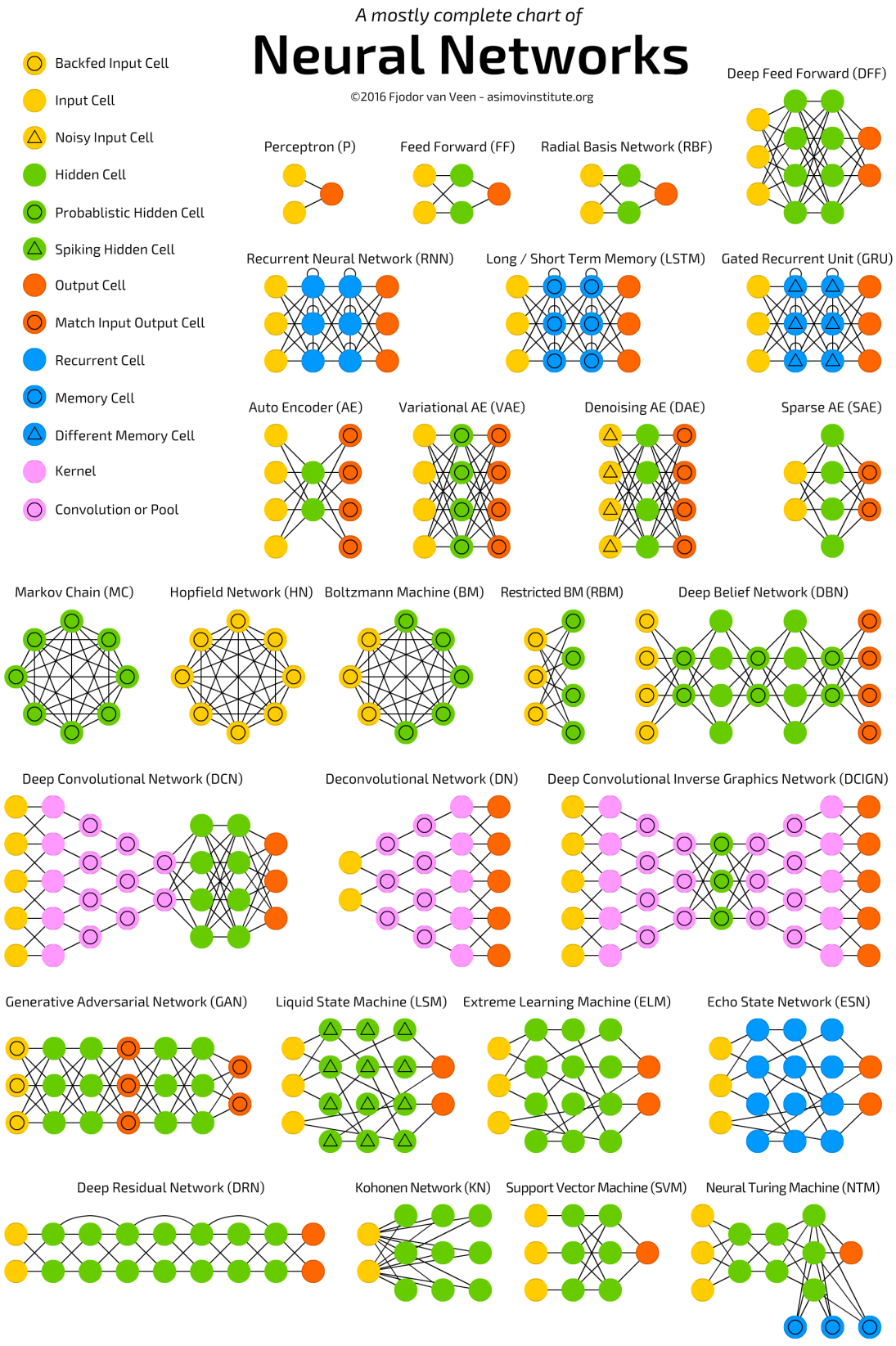
Pro klasifikaci je nutné provést ještě poslední krok, jak je opět patrné z obr. 3.2, tj. vytvořit takovou vrstvu, ve které je každý neuron propojen s každým. Taková vrstva se nazývá FCL (*Fully Connected Layer*). Přestože klasifikace pouze na základě konvolučních a pooling vrstev by byla také možná, je vytvoření plně propojené vrstvy výhodnější a poměrně levný způsob, jak zavést další nelinearitu do modelu.

Ve výstupní vrstvě FCL používají uzly k aktivaci nejčastěji funkci *softmax*[36]. Tato funkce zajistí, že výstup každého uzlu bude mezi hodnotami 0 a 1. Toto rozdělení pak s výhodou může sloužit jako ukazatel pravděpodobnosti klasifikované třídy.

V rovnici aktivační funkce softmax (3.5) je dle našeho značení $\mathbf{x}$ vektor vstupů, j indexuje uzly výstupní vrstvy tak, že $j = 1, 2, ..., K$. Aktivační funkci obecně značíme řeckým písmenem σ[9].

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{3.5}$$

[9]Často bývá s písmenem σ spojena funkce sigmoid, nicméně toto je obecné značení pro kteroukoli aktivační funkci



Obr. 3.6: Populární architektury neuronových sítí

4 SOFTWARE

V této kapitole bude proveden průzkum v současnosti dostupného softwaru. Software bude rozdělen do dvou kategorií – komerční a nekomerční. Za komerční software bude uvažován takový produkt, který je poskytován pod placenou licenci formou subskripce či přímé licence. Jako nekomerční pak bude uvažován takový software, který je licencován některou z licencí otevřeného softwaru nebo jeho použití není zpoplatněno.

Komerční

Zástupců komerčního softwaru lze najít několik. Obecně lze říci, že tyto produkty poskytují kvalitnější služby. Nabízejí grafické uživatelské rozhraní, se kterým lze poměrně intuitivně pracovat, zpravidla nechybí ani podpora širší škály jazyků s rovnocennou přesností převodu textu. Zde byli vybráni tři zástupci komerčního softwaru pro OCR.

Adobe Acrobat DC

Adobe Acrobat je komerční aplikace pro práci s PDF dokumenty umožňující inteligentní scanování a práci s dokumenty v cloudovém prostředí.

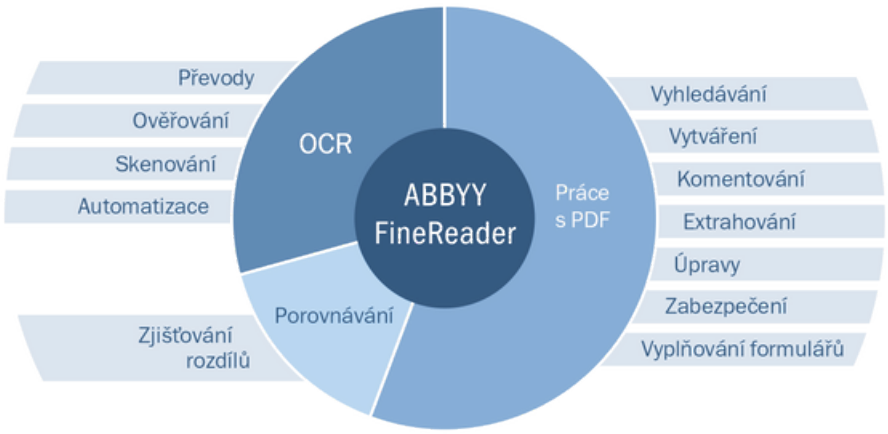
Adobe Acrobat nabízí novinku s názvem Adobe Scan, která umožňuje nascanovat dokument, nebo vyfotit mobilním telefonem, a integrovaný systém OCR rozpozná obrazové a textové elementy a převede dokument do prohledávatelného a upravovatelného PDF formátu. Tato aplikace předpokládá ve výchozím nastavení angličtinu, podporuje ovšem většinu hlavních světových jazyků.

Demonstrační video lze nalézt zde [37].

ABBYY FineReader

> *„FineReader je komplexní softwarová aplikace pro zvyšování produktivity při práci s dokumenty. Nabízí výkonné a zároveň snadno použitelné nástroje pro přístup k informacím v tištěných dokumentech a souborech PDF.“*

Portfolio aplikace FineReader je zobrazeno na 4.1[38].



Obr. 4.1: Portfolio aplikace ABBYY FineReader 14

OmniPage

OmniPage se specializuje na konverzi textů a psaných dokumentů do elektronické podoby. Dokáže automaticky rozpoznávat kolem 120 jazyků, není tedy povinností uživatele specifikovat, o jaký jazyk dokumentu se jedná.

Stejně jako Adobe Acrobat DC využívá i OmniPage cloudového řešení. Díky tomuto přístupu lze efektivně komunikovat s aplikací a nahrávat obrázky a fotky ke zpracování z mobilního zařízení.

OmniPage dokáže konvertovat dokumenty do značného množství výstupních formátů, jako jsou Microsoft® Word, Excel®, PowerPoint®, HTML aj.

Nekomerční

I v nekomerční, *Open Source*[1], sféře lze najít software pro OCR. Bohužel, v porovnání se zmíněnými komerčními alternativami se jedná zpravidla o enginy[2], nebo jednodušší programy, které zpravidla nepodporují grafické uživatelské rozhraní (GUI) a zvládají jen omezené množství jazyků, s nižší přesností klasifikace znaků pro jazyky jiné než angličtina (nepřesnost lze vidět i na jednoduchém příkladu na obr. 4.2). Často je rovněž na uživateli samotném, aby provedl preprocessing obrazu před tím, než jej předloží softwaru pro zpracování.

Relativní nedostatek kvalitního a uživatelsky přívětivého otevřeného OCR softwaru pro český jazyk byl jednou z motivací pro tuto práci.

Tesseract

Tesseract [39] zaujímá dominantní postavení v Open Source OCR software. Byl původně vyvinut v Hewlett-Packard Laboratories Bristol a v Hewlett-Packard Co, Colorado v letech 1985 až 1994. V letech 1996 až 1998 pak prošel změnami při portování na Windows a přepisu do C++. V roce 2005 byl zpřístupněn jako otevřený software a od roku 2006 je dále vyvíjen společností Google. Tesseract je aktivně vyvíjen a podporován. Poslední stabilní verze byla vydána v červenci 2017.

Tesseract je OCR engine, který dokáže rozpoznávat více než 100 jazyků v základní konfiguraci. Je ovšem na uživateli, aby zvolil, o jaký jazyk se jedná. Standardně je předpokládána angličtina. Jelikož se jedná o engine, je možné Tesseract využívat pouze z příkazového řádku. GUI je dostupné pomocí softwaru 3. stran.

Od verze 4.0 využívá Tesseract engine na bázi neuronové sítě, díky čemuž dosahuje výrazně vyšší přesnosti než předchozí verze, výměnou za vyšší výpočetní náročnost. Současné trénovací data pro Tesseract čítají okolo čtyř set tisíc řádků textu v čtyř a půl tisících fontech. I přes zdánlivě dostačující množství trénovacích dat je běžné, že uživatel je nucen (neznámý font, zvýšení přesnosti) pro svůj úkol trénovat síť na vlastních datech. V takovémto případě Tesseract poskytuje podrobný návod [40], jak využít otevřenosti tohoto softwaru ve svůj prospěch.

[1] Otevřený software, zpravidla s dostupným zdrojovým kódem

[2] jádro systému zodpovědné za běh a zpracování, nenabízí uživatelský přístup

(a) Titulní strana v PNG formátu

```
[macermak@localhost Pictures]$ tesseract analyse_single.png stdout -l ces
Warning. Invalid resolution 0 dpi. Using 70 instead.
VYSOKÉ UČENl TECHNICKÉ V BRNĚ

Fakulta strojního inženýrství

BAKALÁŘSKÁ PRÁCE

Brno, 2017 Marek Čermák
```

(b) Výstup Tesseract OCR

Obr. 4.2: Ukázka použití Tesseract OCR

GOCR

GOCR (JOCR[41]) je program vyvinutý pod GNU Public License[3]. Projekt byl založen Joergem Schulenburgem a ve vývoji pokračoval tým vývojářů.

Program slouží ke konvertování nascanovaných obrázků textu do textových souborů. Může být použit s různými front-endy, což umožňuje snadnou portovatelnost na jiné operační systémy a architektury. Jak autoři zmiňují, „umí otevírat různé formáty obrázků a jeho kvalita je upravována na denní bázi“.

Původní jméno GOCR již bylo při registraci programu na Sourceforge obsazeno, proto byl zvolen pseudonym JOCR, přestože interně je název GOCR stále používán.

Google Docs

Přestože Google Docs OCR není v pravém smyslu otevřený software, tj. nemá zveřejněný a dokumentovaný zdrojový kód, nabízí Google tuto službu uživatelům zdarma, proto je zařazena do této sekce.

Google Drive do svých integrovaných Google Docs, kolaborativních dokumentů a prezentací, přidal i službu konverze scanovaných dokumentů pomocí OCR. Využití OCR se v Google Drive poprvé objevilo v červnu 2010. Google tehdy představil možnost převedení textu z formátu PDF nebo souborů obrázků na dokumenty služby „Dokumenty Google“ pro prvních 5 jazyků: angličtinu, francouzštinu, italštinu, němčinu a španělštinu. V polovině roku 2015 poté přidal podporu pro další jazyky, včetně češtiny.

Podrobný návod včetně příkladů, jak využít funkci OCR ke skenování dokumentů, lze nalézt v článku [42].

[3]Licence pro otevřený software opravňující k využívání a modifikaci pod podmínkou nekomerčního šíření

5 REALIZACE

V rámci této práce bude realizována část komplexního projektu (viz sekce 5.1), který je momentálně ve fázi plánování. Práce si klade za cíl vytvořit základní stavební blok tohoto systému, kterým bude rozpoznávání jednotlivých znaků, resp. obrázků těchto znaků.

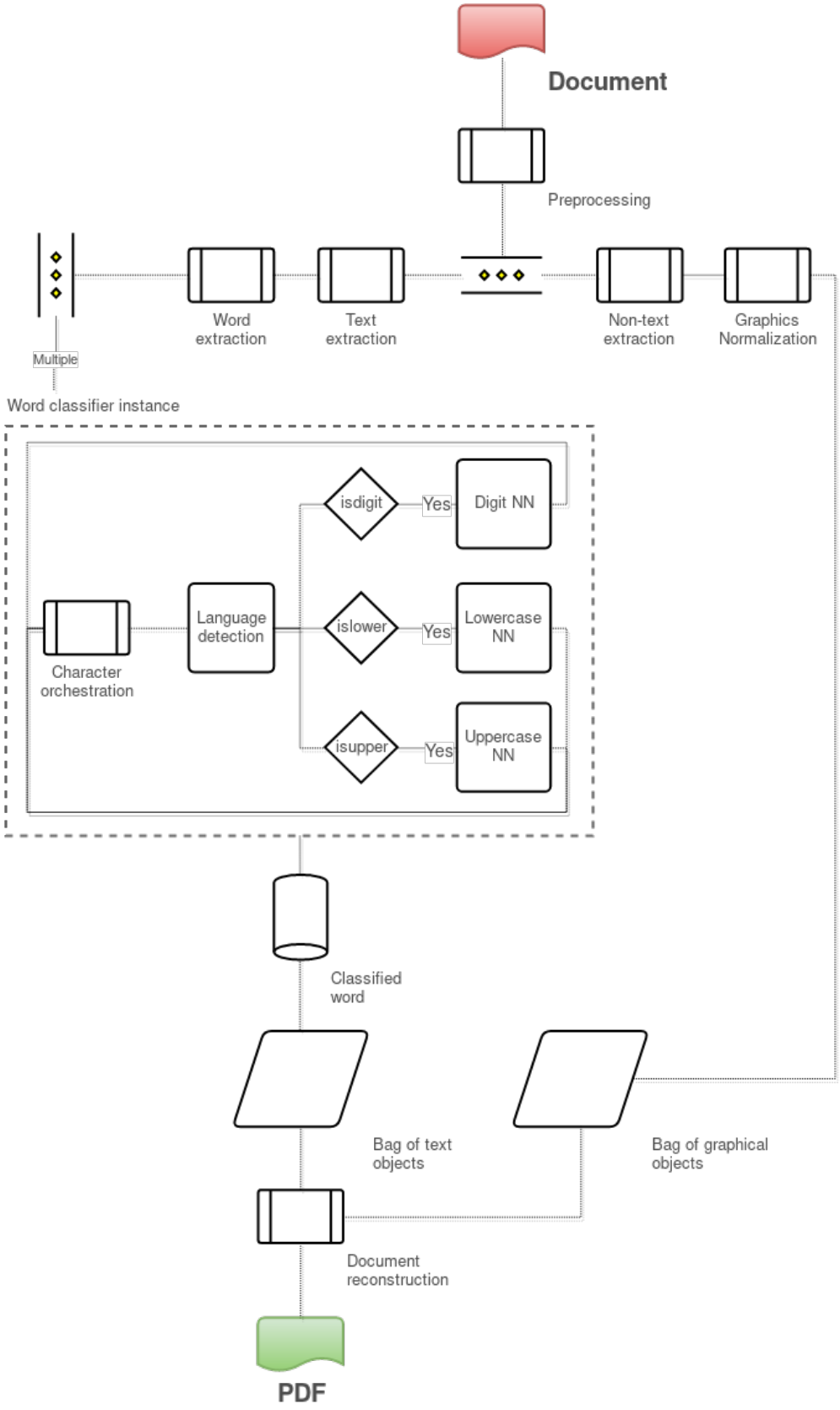
Realizace bude provedena v několika dílčích krocích. Jelikož bude k řešení přistupováno formou učení s učitelem, je proto nezbytné opatřit si data, která budou vhodná jak k učení, tak k následnému ohodnocení modelu. Po předběžném průzkumu stávajících, veřejné dostupných datových souborů nebyl nalezen takový, který by plně vyhovoval stávající aplikaci. První fází realizace tedy bude proces generování tohoto datového setu. V další fázi bude pozornost soustředěna na návrh systému jako takového, tj. specifikace vstupů a očekávaných výstupů, návrh architektury neuronové sítě a návrh dynamicky modifikovatelné aplikace. Posledním krokem této práce bude vytvoření serveru, který bude zpracovávat a obsluhovat dotazy klienta, tj. uživatele, a poskytovat predikci výsledku spolu s dalšími informacemi o procesu.

5.1 Kontext aplikace

Kontextem této práce je komplexní systém pro převod tištěných dokumentů do elektronické podoby při pozičním zachování komponent. Těmito komponentami se rozumí definované bloky dokumentu, jako jsou např. bloky textu, odstavce, nadpisy, obrázky, či jiné grafické objekty. Schéma tohoto systému je zobrazeno na obr. 5.1.

Vstupem tohoto systému je obrázek (fotografie, či scan) dokumentu. V jednotlivých fázích procesu dojde k postupně k předzpracování dokumentu. při kterém jsou použity techniky popsány v kapitole 2, sekci 2.1, extrakci textových a netextových bloků a jejich zpracování. Výstupem je pak elektronická podoba dokumentu.

Koncept takovéhoto systému je do jisté míry podobný např. zmíněnému ABBYY FineReader v kapitole 4 - Software. Tento software nicméně není otevřený, ani jej nelze bezplatně použít pro nekomerční záměry, což je motivací pro návrh nového otevřeného softwaru.



Obr. 5.1: Schéma systému pro zpracování dokumentů.

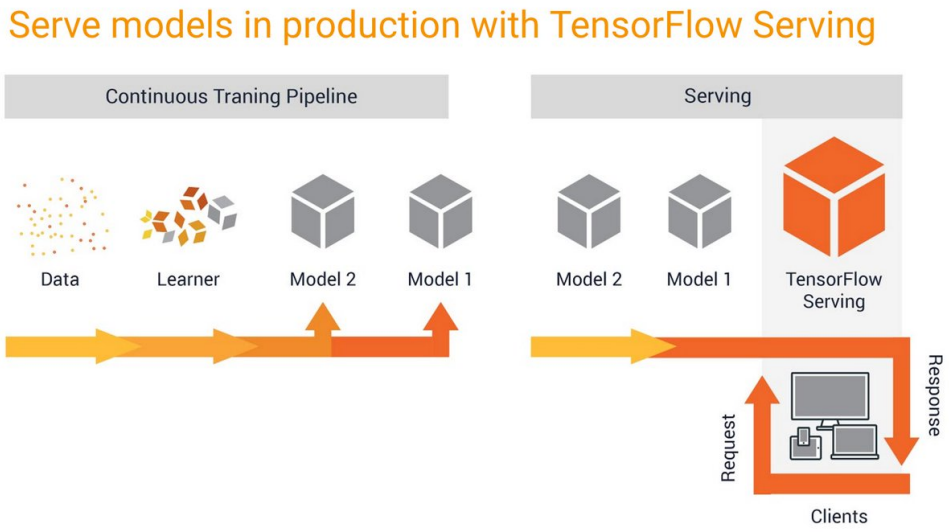
5.2 Návrh aplikace

V této práci bude navržena neuronová síť klasifikující alfanumerické znaky z jejich obrazové předlohy. Bloky korespondující tomuto zadání ve schématu 5.1 jsou bloky označené jako „NN“.

Konkrétním typem neuronové sítě bude konvoluční neuronová síť (viz sekce 3.2.2 – Konvoluční neuronové sítě). Implementace bude realizována v jazyce Python, který je hojně využíván pro vědecké aplikace a má rovněž velmi kvalitní vývojářskou základnu v oblasti strojového učení a umělé inteligence. Další výhodou jazyka Python je optimalizace maticových operací. Tyto výpočty jsou implementovány v jazycích Fortran a C, což značně zvyšuje jejich rychlost.

Pro vývoj aplikace bude zvolen aplikační rámec TensorFlow. Přestože existují i jiné, jako např. PyTorch, nabízí TensorFlow širší vývojářskou základnu a velké množství vypracovaných a dokumentovaných příkladů, které umožňují snazší pochopení vývoje. Výhodou TensorFlow je rovněž možnost implementace nativní klientské obsluhy (viz dále).

Vlastní aplikace se bude sestávat ze dvou oddělených bloků. Prvním blokem bude trénování neuronové sítě, druhým pak bude obsluha klienta pomocí TensorFlow serveru. K trénování sítě klient přístup mít nebude, bude interagovat s již natrénovaným modelem, kterého bude žádat o predikce. Takovéto schéma je typické pro produkci modelu a je zobrazeno na obr. 5.2 [43].



Obr. 5.2: Schéma aplikace

TensorFlow

TensorFlow je open source *framework*[1]pro strojové učení. Tento framework byl původně vyvinut inženýry pracujícími pro Google Brain Team za účelem zjednodušení řešení ML problémů a výzkumu v oblasti hlubokých neuronových sítí. Tento systém je ale natolik obecný, že může být použit v mnohem širším doménovém spektru. Verze 1.0 byla vydána v roce 2017.

TensorFlow pracuje s daty rozdílným způsobem, než je v Pythonu obvyklé a rozděluje definici výpočetních operací od jejich provedení. Toto se děje za účelem optimalizace. Operace jsou nadefinovány a vnitřně charakterizovány grafovou strukturou. Tento graf slouží jako model. K provedení definovaných operací dochází pouze v tzv. „*session*“[2].

Jelikož je tento framework natolik komplexní, aby pokryl většinové spektrum aplikačního rámce, bude s jeho pomocí provedena prakticky kompletní realizace této aplikace.

Obsluha klienta

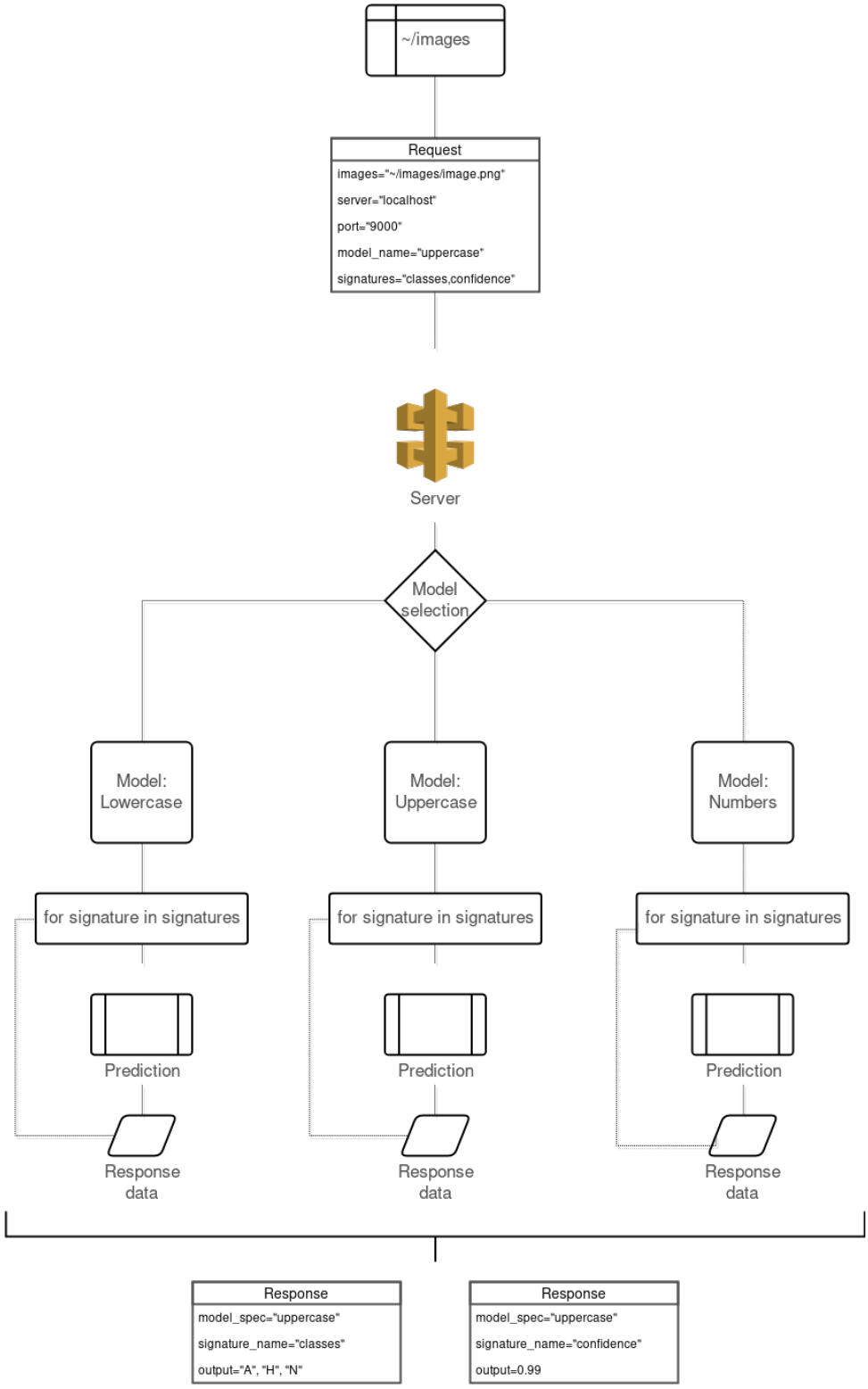
Výběr kandidátů pro implementaci serveru byl zúžen na dva - Flask, což je webový framework implementovaný v Pythonu, a TensorFlow samotné. Uživatelská přívětivost a jednoduchost Flasku je excelentní, naproti tomu i přes nesnadnou implementaci serveru TensorFlow, hraje v jeho prospěch kompatibilita se zbytkem aplikace. Nakonec byl kvůli nativní kompatibilitě a rychlosti zvolen TensorFlow, resp. modul TensorFlow Serving.

Klient bude moci podat požadavek na server tak, že specifikuje tzv. *signatury*. Signatury specifikují model, který se má použít pro vyvození predikce a také hodnoty, které se mají predikovat. Server zpracuje požadavek, vybere požadovaný model a na základě signatur aktivuje neuronovou síť.

Odpověď je klientovi doručena pro každou signaturu zvlášť, čímž je urychlen chod serveru a v případě kolize dostane klient odpověď alespoň na ten požadavek, který byl zpracován korektně. Podrobnější příklad klientského požadavku včetně možných signatur je na obr. 5.3.

[1]Framework je aplikační rámec zaměřující se na konkrétní spektrum softwarového vývoje, typicky např. webové aplikace. Poskytuje návrhové vzory a doporučené postupy při vývoji. Jeho cílem je usnadnění práce vývojáře a zrychlení vývoje.

[2]Volně přeloženo jako „sezení“, či „zasedání“. Jedná se se o posloupnost operací, kdy dochází k optimalizování struktury a provedení definovaných operací.



Obr. 5.3: Příklad požadavku klienta.

5.3 Vlastní implementace

5.3.1 Sběr dat

Vzhledem k tomu, že se nepodařilo nalézt soubor dat, který by plně vyhovoval potřebám této práce, bylo rozhodnuto o vytvoření nového datového setu. Implementace generátoru je nad rámec této práce, a proto zde nebude popisována. Nicméně, generování dat s sebou přináší jistá rizika, proto zde budou okrajově zmíněny některé principy a požadavky na trénovací data. Je potřeba, aby data splňovala několik hlavních požadavků:

- Reálnost a úplnost
- Konzistence
- Rovnoměrnost rozložení
- Variance
- Velikost datového souboru

Pořadí požadavků je naprosto náhodné. Každý z těchto bodů je stejně důležitý pro dosažení kvalitních výsledků.

Reálnost a úplnost

Je zřejmé, že pokud by neuronová síť byla učena na datech příliš odlišných od reálných dat, tj. od dat, které bude natrénovaná síť ve výsledné aplikaci klasifikovat, mohla by výsledný produkt podávat velmi špatné výsledky. Zde je nutné podotknout, že nelze s naprostou jistotou předvídat, jaká data budou v produkci k dispozici. Je ovšem nezbytné vytvořit natolik generický soubor, aby pokryl většinové množství případů, což souvisí s požadavkem úplnosti. Pokud bude stanoveno, že má CNN umět klasifikovat všechna malá písmena, je nutné, aby tato byla obsažena i v trénovacím souboru.

Konzistence

Data by měla být konzistentní, tzn. vzhledem k této práci, by každý jednotlivý obrázek měl mít stejné dimenze. Stejně tak by se nemělo stát (pokud tomu tak není úmyslně), aby v datovém souboru obsahujícím černobílé obrázky byl nějaký, který je barevný, či prázdné pozadí. Takovéto vzorky mohou velmi negativně ovlivnit gradient celé sítě, jelikož vzorky stejné třídy se extrémně liší. Konzistentní data se někdy také nazývají jako „čistá“.

Rovnoměrnost rozložení

Rovnoměrně rozložená data znamenají data taková, že pro žádnou třídu se množství trénovacích vzorků výrazně neliší. Důvodem je skutečnost, že pokud by některá ze tříd byla majoritní, došlo by k *overfittingu* (viz poznámku u Pooling a sub-sampling) vzhledem k této třídě. Model by teto hůře klasifikoval, či dokonce misklasifikoval, kandidáty jiných tříd.

Variance

Tato podmínka je zřejmá. Variance v datech je nezbytnou součástí trénovacího procesu. Pokud by každý vzorek byl stejný, nebo s minimálními rozdíly, naučila by se neuronová síť perfektně rozeznávat daný vzorek, ale nebyla by schopna generalizace na objekty lišící se od trénovacího exempláře.

Velikost datového souboru

Velikostí datového souboru je zde míněn počet trénovacích vzorků pro každou třídu. Určení této veličiny je nesnadné a v současné době není známo žádné analytické odvození. Obecně existuje tvrzení, že čím více trénovacích dat, tím lépe. Pokud jsou data správná a splňují výše uvedené, lze toto tvrzení považovat za správné. V praxi je tedy snaha odhadnout spíše spodní hranici velikosti dat. V takovém případě se doporučuje vycházet z podobných implementací, kde byla tato hodnota experimentálně zjištěna.

5.3.2 Výběr modelu

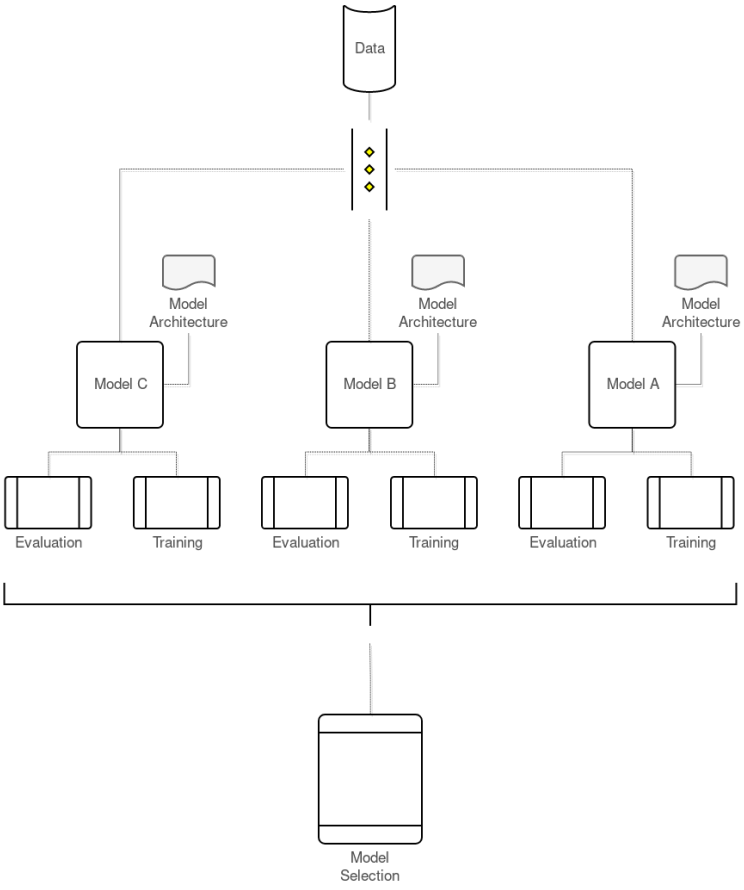
Protože předem není známa přesná architektura neuronové sítě, ani hyperparametry, které budou použity k jejímu trénování, byla navržena nová forma implementace, která odpovídá potřebě snadno a dynamicky modifikovat architekturu. Schéma tohoto návrhu je na obr. 5.4. Rovněž, jak je možné vidět na obr. 5.1, bude potřeba nalézt tři samostatné modely. Model pro klasifikaci čísel a modely pro klasifikaci velkých a malých písmen. Je tedy třeba buďto nalézt univerzální architekturu a pouze modifikovat hyperparametry, nebo optimalizovat i jednotlivé architektury pro různé modely.

Princip je takový, že architektura neuronová sítě i hyperparametry jsou modifikovatelné pomocí *yaml*[3] souboru. Každý model tak lze předem nadefinovat a architektura se přizpůsobí definici. Tento princip má i další výhody. Jak je znázorněno na obr.

5.4, modely lze trénovat paralelně, zaznamenávat vyhodnocující metriky jednotlivých modelů a ve finále provést výběr nejlepšího modelu. Je nezbytné poznamenat, že při paralelním trénování mohou nastat extrémní rozdíly v délce trénování jednotlivých modelů v závislosti na jejich architektuře a hypeparametrech.

Dimenzi neuronové sítě, jejíž architektura je zobrazena na obrázku 5.5, je-li uvažováno zachování šířky a výšky vstupu v každém kroku konvoluce[4], popisuje v první konvoluční vrstvě vektor $(3, 3, 32)$, což odpovídá $3 \cdot 3 \cdot 32 = 288$ parametrů. Nicméně po průchodu druhou konvoluční vrstvou vzroste počet parametrů na $3 \cdot 3 \cdot 32 \cdot 32 = 9216$. Přestože pooling sníží markantně množství parametrů, lze si ověřit, že plně propojená vrstva obsahuje 262144 parametrů.

Vzhledem k tomuto exponenciálnímu růstu je zřejmé, že i relativně malé rozdíly v architektuře jednotlivých modelů mají markantní dopad na množství parametrů a tedy i na náročnost trénovacího procesu.



Obr. 5.4: Schéma procesu výběru modelu.

[3]YAML je (podobně jako JSON) strojově zpracovatelný formát pro serializaci a strukturování dat.

[4]Toto je naprosto validní předpoklad s reálným využitím v praktických aplikacích.

Architektura neuronové sítě

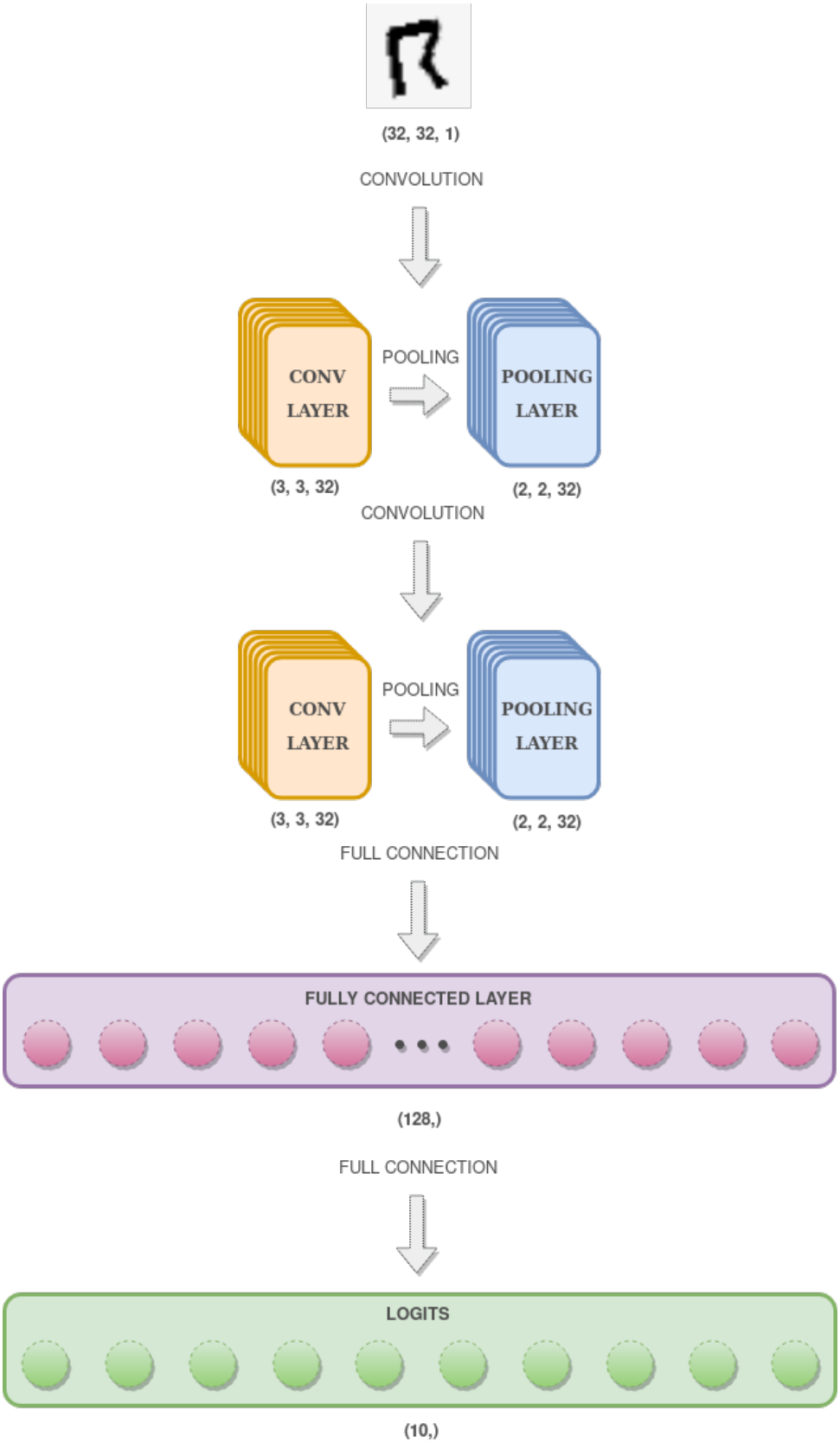
Pomocí postupu popsaného výše byla zvolena architektura konvoluční neuronové sítě, která je zobrazena na obrázku 5.5. V obou konvolučních vrstvách se nejlépe osvědčily aktivační funkce *leaky ReLU*[5]a v plně propojené vrstvě byla použita aktivační funkce *sigmoid*.

Neuronová síť předpokládá vstup v podobě jednoho alfanumerického znaku v podporovaném obrazovém formátu[6]. Výstupem pak je n kandidátů včetně numerického ohodnocení jistoty správnosti predikce daného kandidáta, tzv. *confidence score*. Hodnota n je libovolný parametr.

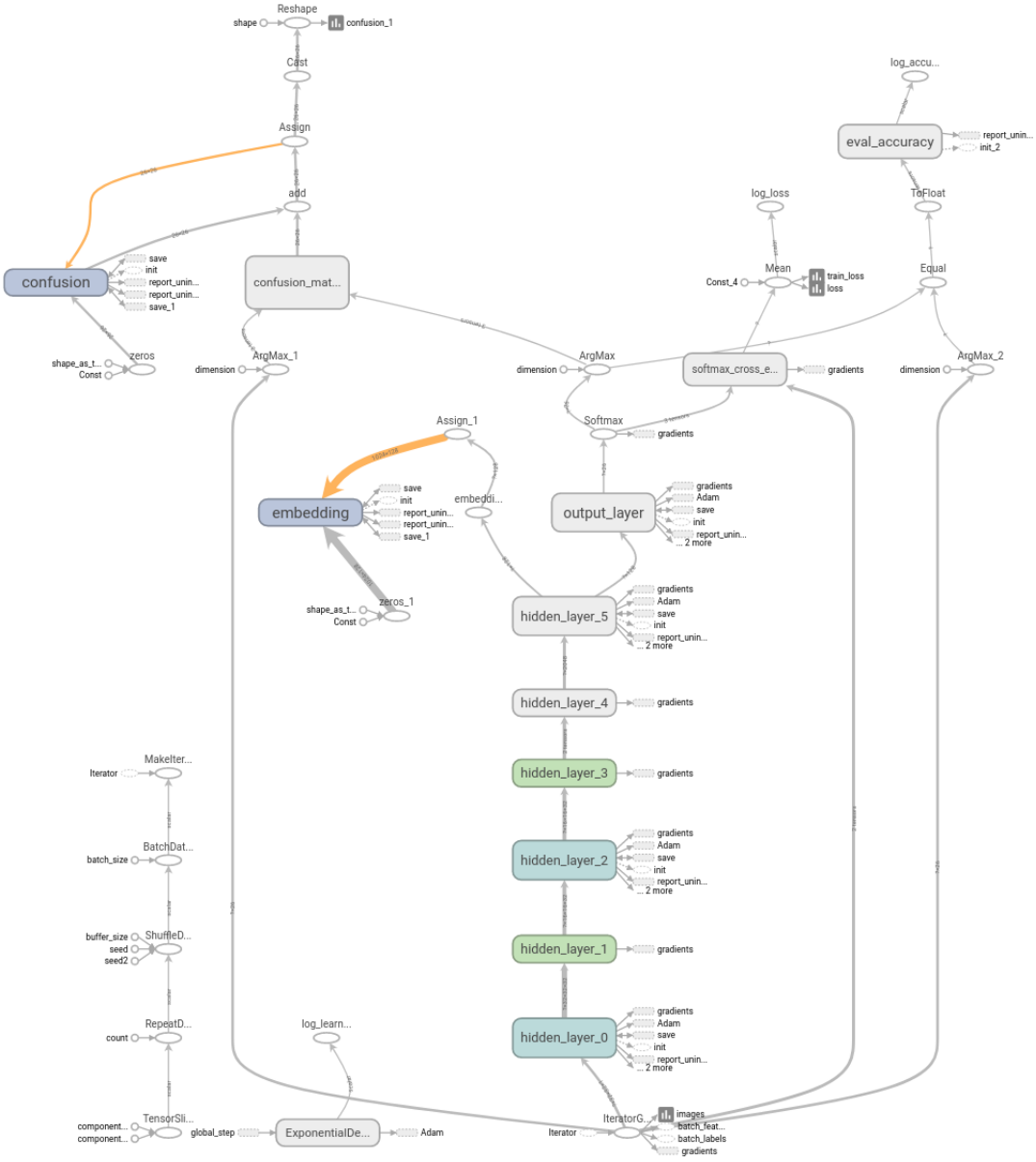
Na obrázku 5.5 je schéma architekury výsledné konvoluční neuronové sítě. Pro zajímavost je pak na obrázku 5.6 zjednodušený graf skutečné implementace neuronové sítě s užitím TensorFlow. Tento graf je poměrně rozsáhlý a i při zjednodušení jej nelze přehledně vyobrazit při rozměrech listů této bakalářské práce. V tomto textu slouží pouze k ukázce složitosti a větvení reálné implementace neuronové sítě, čtenář si jej pak může v plné velikosti zobrazit ze souboru příloh.

[5]Jedná se o obdobu ReLU (viz 3.4) s tím rozdílem, že linearita je aplikována s koeficientem α i v záporné části grafu.

[6]Podporované formáty jsou vypsány zde: `http://pillow.readthedocs.io/en/5.1.x/handbook/image-file-formats.html#fully-supported-formats`.



Obr. 5.5: Schéma architektury neuronové sítě.



Obr. 5.6: Graf neuronové sítě v TensorFlow.

6 VÝSLEDKY

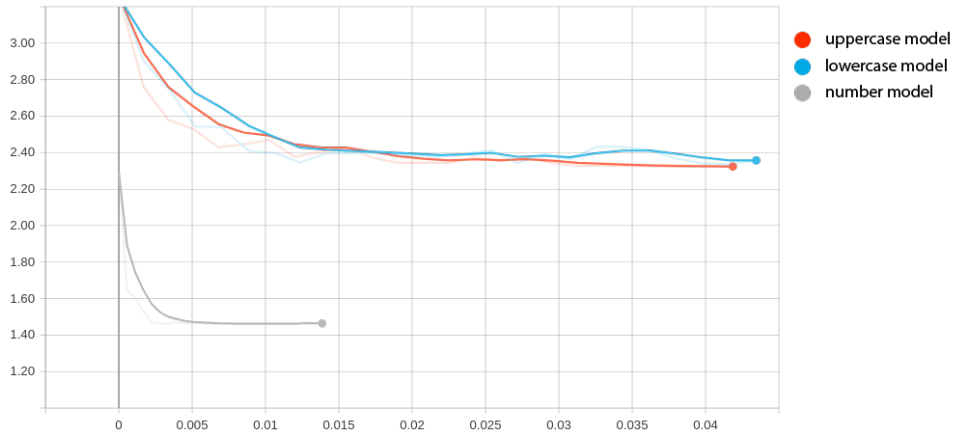
V této kapitole budou zhodnoceny výsledky aplikace. Bude zhodnocena přesnost klasifikace neuronovou sítí, vizualizace parametrů sítě pomocí pokročilých vizualizačních technik a nakonec předveden dotaz klienta na server, čímž bude ukázána funkcionalita serveru a výsledný formát odpovědi.

6.1 Analýza trénovacích charakteristik

Při trénování neuronové sítě jsou pravděpodobně nejdůležitějšími charakteristikami dvě veličiny – přesnost a tzv. „*loss*“. *Loss* je veličina udávající hodnotu *loss funkce* (viz 3.2.1). V tomto případě byla použita funkce zvaná *cross entropy*, jejíž předpis vypadá následovně.

$$H(x) = \sum_{x} P(x) log Q(x) \tag{6.1}$$

Hodnota této funkce by měla při trénování klesat. Křivka závislosti hodnoty loss funkce na trénovacích krocích, případně na čase, by měla mít v ideálním případě exponenciální průběh bez výrazných skoků či zlomů. Takovouto křivku zle vidět v grafu 6.1, kde jsou vykresleny průběhy pro všechny tři trénované modely zároveň. Na ose y je hodnota loss funkce, na ose x je pak relativní *wall time*, což je čas měřený mikroprocesorem od prvního datového bodu.



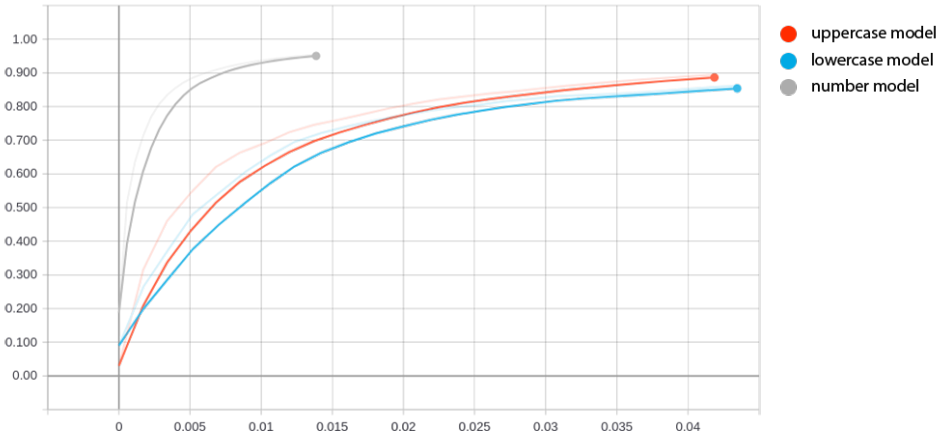
Obr. 6.1: Vizualizace „*loss*“ charakteristiky.

Z grafu je patrné, že čas potřebný k trénování neuronové sítě pro klasifikaci čísel (tj. šedá křivka) je výrazně kratší. Z průběhu loss funkce tohoto modelu lze rovněž

pozorovat poměrně zajímavý fakt, a sice že funkce konverguje zhruba po první třetině trénovacího procesu. Toto poukazuje na skutečnost, že po této době se již neuronová CNN „nenaučila nic nového“ a její trénink by tedy mohl být ukončen mnohem dříve.

Pod pojmem „přesnost“ je uvažován poměr správně klasifikovaných vzorků k celkovému počtu vzorků. Hodnota přesnosti tedy leží v uzavřeném intervalu $[0, 1]$. Je samozřejmostí, že přesnost by měla v procesu trénování růst. V ideálním případě by křivka popisující závislost přesnosti na trénovacím kroku či čase měla růst exponenciálně.

Graf přesnosti na obr. 6.2 potvrzuje pozorování z předchozí charakteristiky, a sice že přesnost modelu pro klasifikaci čísel roste výrazně rychleji.



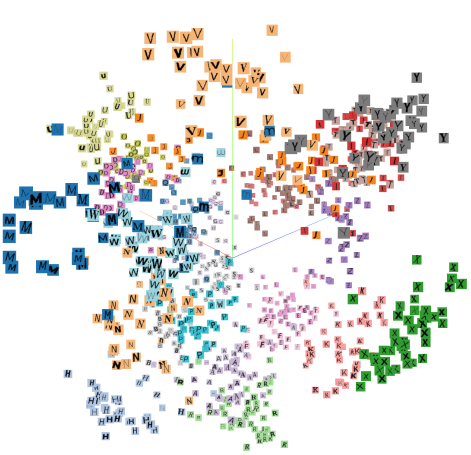
Obr. 6.2: Vizualizace přesnosti při trénování.

6.2 Analýza metodou hlavních komponent

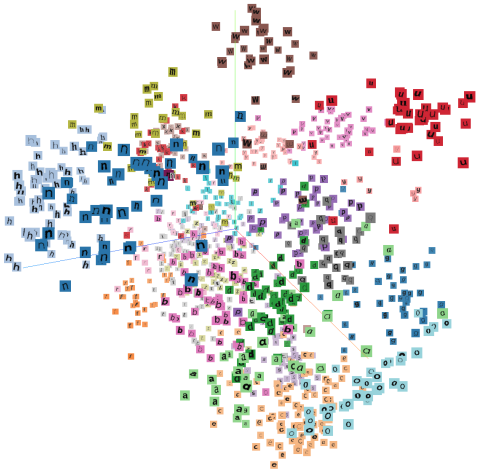
Metoda hlavních komponent (*Principal Component Analysis - PCA*) je statistickou metodou pro analýzu multidimenzionálních dat. Podrobný popis této metody je nad rámec této práce, lze se dočíst více např. v [44].

Podstatou metody je redukce dat na nižší počet dimenzí pomocí rozkladu na tzv. *hlavní komponenty*. Tyto komponenty jsou v podstatě vektory popisující nejvyšší varianci dat. Výběrem p nejvhodnějších vektorů (*eigenvectors*), tj. s nejvyšší hodnotou veličiny zvané *eigenvalue*, získáme hlavní komponenty, které tvoří nový p-dimenzionální graf.

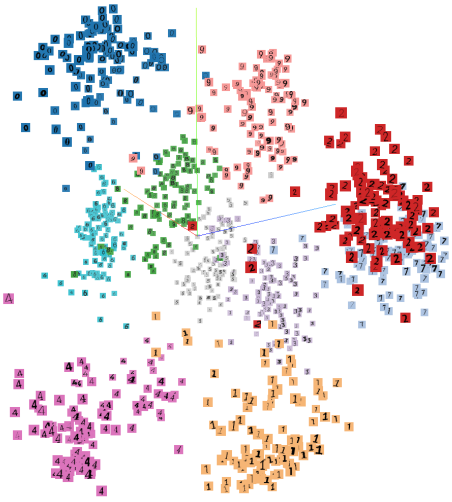
Tato metoda byla použita k redukci parametrů předposlední vrstvy CNN do tří dimenzí. Vizualizace na obr. 6.3 ukazuje, jak se neuronová síť dokázala naučit reprezentaci jednotlivých tříd. Grafy si lze interpretovat tak, že clustery jednotlivých tříd, které jsou blízko sebe, či se navzájem prolínají, charakterizují třídy, které byly obtížnější na klasifikaci. To znamená, že tenzory popisující reprezentaci těchto tříd mají menší vzdálenost (třídy jsou si „podobné“, pokud jejich vektorové reprezentace jsou si vzdálenostně blízké). Typicky blízké jsou si shluky písmen „v“ a „y“, nebo např. „k“ a „x“. Čtenář má opět možnost zobrazit si vizualizace ze souboru příloh, kde jsou k dispozici i rotující vizualizace 3D modelů.



(a) model pro velká písmena



(b) model pro malá písmena



(c) model pro čísla

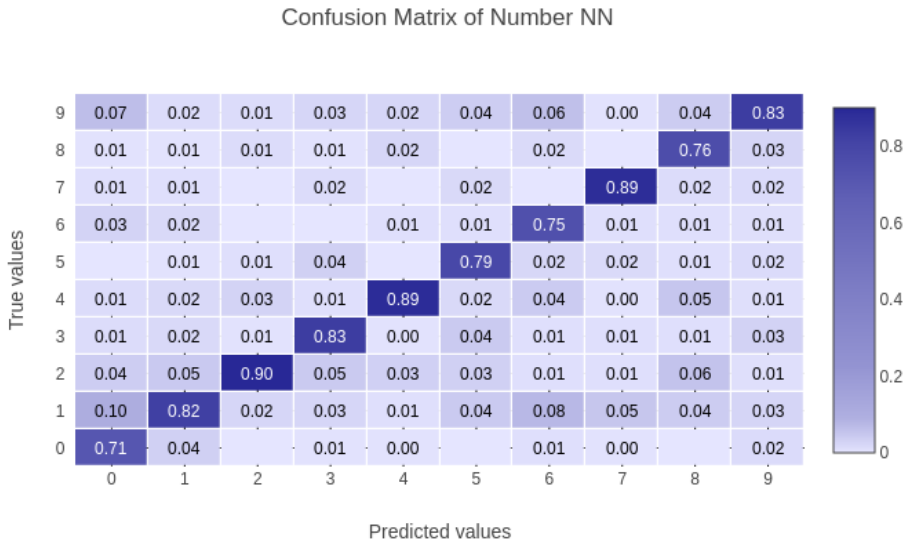
Obr. 6.3: Vizualizace metodou hlavních komponent

6.3 Analýza přesnosti pomocí konfúzní matice

Konfúzní matice (CM[1]) je čtvercová matice o rozměrech $m \times m$, kde m je počet klasifikovaných tříd. Využívá se velmi často v klasifikačních problémech, protože má vyšší výpovědní hodnotu než pouhá přesnost či výtěžnost, jejichž hodnoty mohou být mnohdy zavádějící.

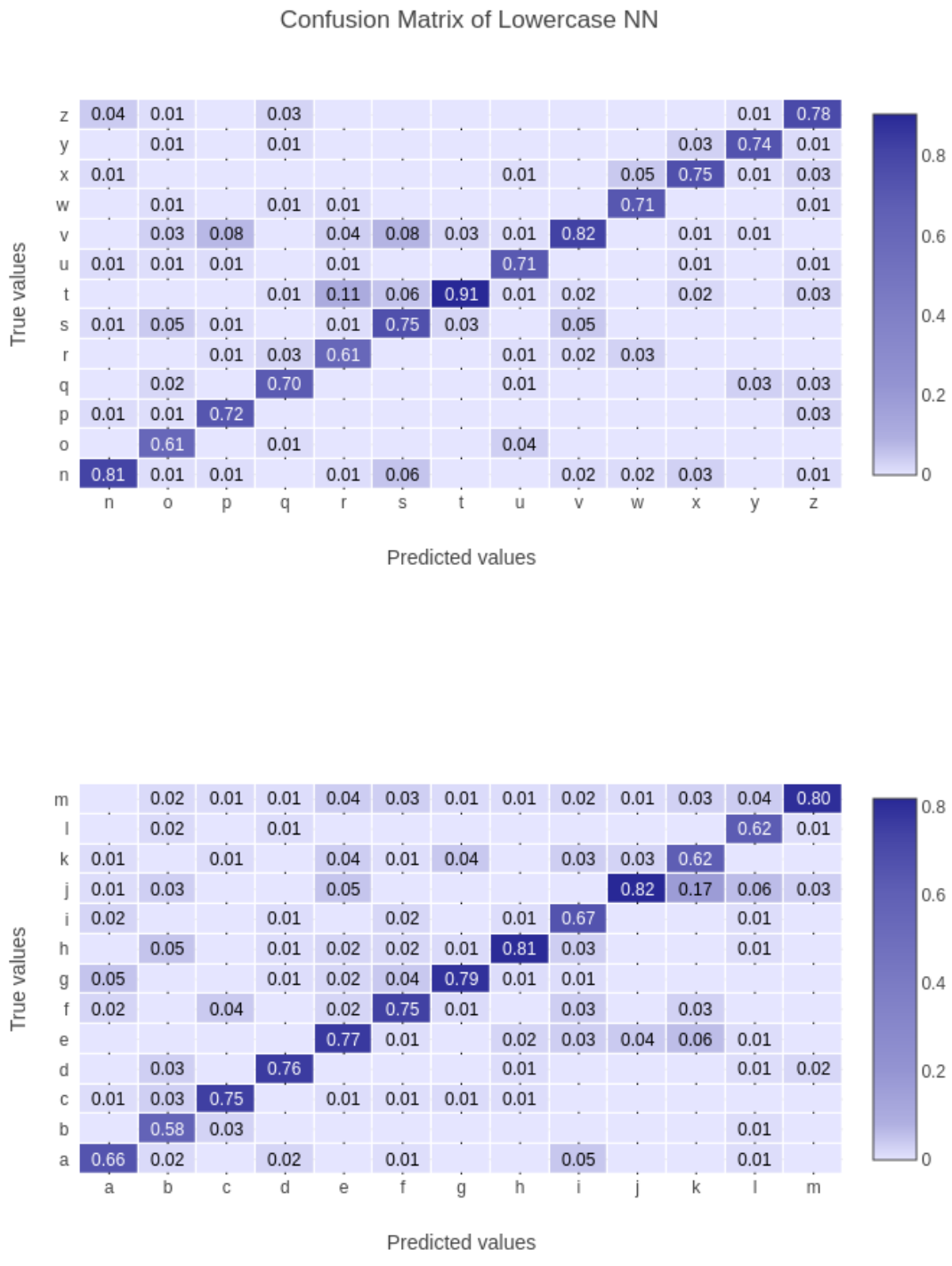
Nechť řádky této matice označují predikované třídy a sloupce označují třídy skutečné, tj. *labels*. Pak na diagonále této matice budou vhledem ke každé třídě tzv. *true positives*, což je korektní predikce náležitosti dané třídě. Hodnoty v řádcích CM lze označit jako *false positives*, tj. chyby, při kterých byl vzorek nekorektně označen jako náležící dané třídě. Hodnoty ve sloupcích takovéto konfúzní matice pak představují tzv. *false negatives*, tj. opak false positives – vzorek byl nekorektně označen jako **ne**náležící dané třídě.

Vizualizace těchto matic pro jednotlivé modely neuronových sítí jsou zobrazeny postupně na obr. 6.4, obr. 6.5 a na obr. 6.6. Každá z těchto matic je normalizována v rámci sloupce, což umožňuje interpretovat jednotlivé diagonální prvky jako „sebevědomí“, které klasifikátor vykazuje pro danou třídu. Je nutno podotknout, že data, která byla použita pro predikci – a tedy sestavení CM – byla augmentována značným množstvím filtrů o náhodných parametrech a představují prakticky nejhorší možné vstupy, které by mohla v reálném prostředí NN dostat.

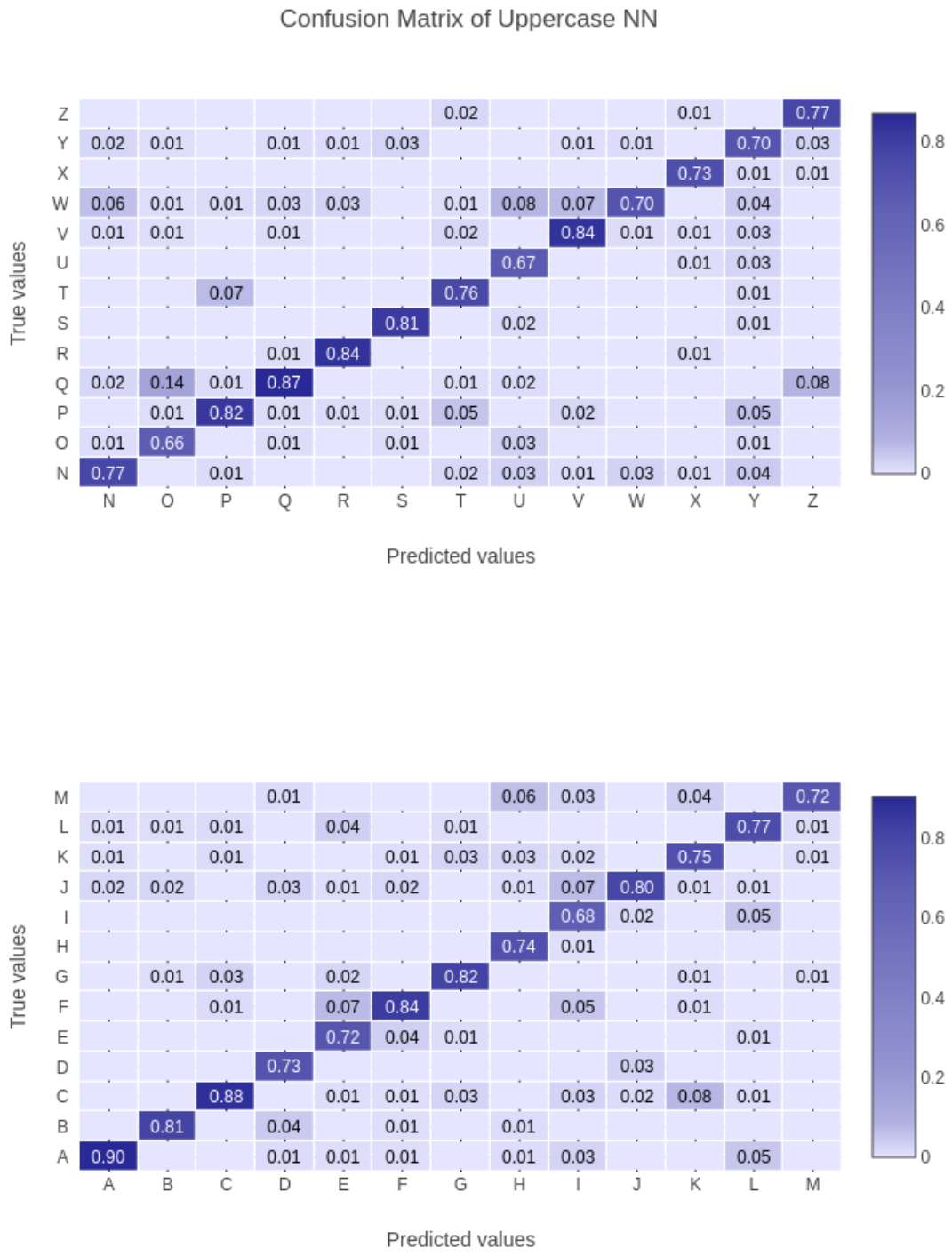


Obr. 6.4: Konfúzní matice pro číselný model

[1]Z anglického „*confusion matrix*“. V češtině lze rovněž uvést tento anglický termín.



Obr. 6.5: Konfúzní matice pro model malých písmen.



Obr. 6.6: Konfúzní matice pro model velkých písmen.

6.4 Predikce pomocí serveru

Požadavek klienta byl podán dle obr. 5.3. Ve výpisu 6.1 lze vidět neparsovanou odpověď serveru (tj. ve formátu, v jakém server přenáší data klientovi). Konkrétně se jedná o člověkem čitelnou verzi formátu pro serializaci dat zvanou *Protocol Buffers*[2]. Tento formát byl vyvinut společností Google a jedná se o platformě nezávislý serializační protokol, který je zejména interně v Googlu velmi využíván.

```
outputs {
  key: "output"
  value {
    dtype: DT_STRING
    tensor_shape {
      dim {
        size: 1
      }
      dim {
        size: 3
      }
    }
    string_val: "A"
    string_val: "H"
    string_val: "N"
  }
}
model_spec {
  name: "uppercase"
  version {
    value: 1523227963
  }
  signature_name: "classes"
}
```

```
outputs {
  key: "output"
  value {
    dtype: DT_FLOAT
    tensor_shape {
      dim {
        size: 1
      }
    }
    float_val: 0.9999933242797852
  }
}
model_spec {
  name: "uppercase"
  version {
    value: 1523227963
  }
  signature_name: "confidence"
}
```

Výpis 6.1: Neparsovaná odpověď serveru.

[2]Více o Protocol Buffers se lze dočíst v oficiální dokumentaci `https://developers.google.com/protocol-buffers/`

7 VYUŽITÍ

Navržená aplikace pro rozpoznávání znaků nachází hlavní uplatnění v kontextu uvedeném v kapitole 5.1, kde tvoří základní stavební blok. Využití aplikace ovšem není tímto kontextem nikterak omezeno. Aplikace byla navržena pomocí dynamického konceptu a publikována jako otevřený software, což umožňuje její využití v širším spektru uživatelů. Potenciální uživatel tak může neuronovou síť natrénovat na vlastních datech, kterými mohou být samozřejmě i jiné než alfanumerické znaky - čínské znaky, azbuka. Dynamický návrh pak zajišťuje, že si uživatel bude moci snadno modifikovat architekturu a hyperparametry sítě tak, aby dosáhl dostatečné přesnosti klasifikace.

Další možností je integrace serveru do webové aplikace a vytvoření uživatelského rozhraní, které může například také poskytovat vizualizace trénovacího procesu (v rámci této práce byla ukázána pouze část implementovaných vizualizací).

Neméně zajímavou alternativou může být také přímá interakce s uloženým modelem, nebo následné trénování přímo v prohlížeči. S nedávným vydáním nového TensorFlow frameworku - TensorFlow.js by mělo být možné tohoto dosáhnout.

Konečně, v této práci byly pro svou dostupnost použity znaky, které neobsahovaly česká písmena. Dalším krokem může být vytvoření datového souboru zahrnujícího české znaky a trénování CNN v rozpoznávání také těchto znaků.

ZÁVĚR

V první části této práce byla věnována pozornost rešerši metodiky, metod a technik používaných pro OCR a rozpoznávání textu. Bylo popsáno obecné schéma řešení OCR a poté porovnány metody klasického přístupu a přístupu pomocí soft computingu. Metody klasického přístupu nebyly probrány do detailu, jelikož nebyly předmětem realizace. Větší část rešerše se věnovala soft computingu, konkrétně metodám umělé inteligence a strojového učení. Největší pozornost v teoretické části byla věnována neuronovým sítím. Druhou částí pak byla realizace neuronové sítě pro klasifikaci alfanumerických znaků a implementace systému obsluhy klienta.

Protože nebyla nalezena vhodná, veřejně dostupná data pro trénování neuronové sítě, byl v rámci této práce navržen i generátor trénovacích dat. V jazyce Python pak byla navržena aplikace umožňující dynamicky definovat architekturu a hyperparametry neuronové sítě, což umožnilo návrh finální konvoluční neuronové sítě rozpoznávající alfanumerické znaky a její optimalizaci na generovaných datech. Pro tuto aplikaci byl poté vytvořen server obsluhující klientské požadavky a poskytující predikce daných tříd. Závěrem byla provedena analýza trénovacího procesu, přesnosti klasifikace a vizualizace vnitřní reprezentace znaků ve vrstvě CNN. Rovněž byla na příkladu ukázána odpověď serveru na požadavek klienta.

Cílem této bakalářské práce bylo porozumět principu neuronových sítí, ukázat jejich využitelnost při řešení složitých úloh a naučit se je implementovat a integrovat v komplexnějším systému a konečně vytvořit aplikaci rozpoznávající alfanumerické znaky. Tento projekt byl zasazen do kontextu komplexního softwaru pro převod tištěných dokumentů do elektronické podoby a tvořil základní prvek tohoto softwaru.

Aplikace pro rozpoznávání znaků byla navržena tak, aby byla dynamicky modifikovatelná na jakoukoli architekturu neuronové sítě a umožňovala znovupoužitelnost pro odlišná vstupní data. Software byl vydán pod licencí GPLv3 [45], tedy jako otevřený software, což naplňuje jednu z motivací pro tvorbu této práce – nedostatek otevřeného softwaru v oblasti rozpoznávání textu. Nic tedy nebrání dalšímu rozvoji mimo vytyčený kontext a zmíněné možnosti budoucího využití.

LITERATURA

[1] 105 (vetenskapen och livet / Årgång vii: 1922), 2016. URL: `http://runeberg.org/vetlivet/1922/0113.html`.

[2] Arindam Chaudhuri, Krupa Mandaviya, Pratixa Badelia, and Soumya K Ghosh. *Optical Character Recognition Systems for Different Languages with Soft Computing*. Springer, 2017.

[3] Muhammad'Arif Mohamad, Dewi Nasien, Haswadi Hassan, and Habibollah Haron. A review on feature extraction and feature selection for handwritten character recognition. *International Journal of Advanced Computer Science and Applications*, 6(2):204–212, 2015.

[4] Azizah Suliman, Mohd Nasir Sulaiman, Mohamed Othman, and Rahmita Wirza. Chain coding and pre processing stages of handwritten character image file. *electronic Journal of Computer Science and Information Technology*, 2(1), 2011.

[5] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[6] Wilhelm Burger, Mark James Burge, Mark James Burge, and Mark James Burge. *Principles of digital image processing*. Springer, 2009.

[7] Raymond Legault and Ching Y Suen. Optimal local weighted averaging methods in contour smoothing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(8):801–817, 1997.

[8] Mehmet Sezgin et al. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13(1):146–168, 2004.

[9] Stanislav Straka. *Segmentace obrazu*. PhD thesis, Masarykova univerzita, Fakulta informatiky, 2009.

[10] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[11] Archana S Sawant and DG Chougule. Script independent text pre-processing and segmentation for ocr. In *Electrical, Electronics, Signals, Communication and Optimization (EESCO), 2015 International Conference on*, pages 1–5. IEEE, 2015.

[12] Peter VESELÝ. Ocr analýza [online]. Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2006 [cit. 2017-11-27]. URL: http://is.muni.cz/th/51661/fi_m/.

[13] Arindam Chaudhuri. *STUDIES IN APPLICATIONS OF SOFT COMPUTING TO SOME OPTIMIZATION PROBLEMS*. PhD thesis, Netaji Subhas Open University, 2010.

[14] Tao Wang, David J Wu, Adam Coates, and Andrew Y Ng. End-to-end text recognition with convolutional neural networks. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3304–3308. IEEE, 2012.

[15] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[16] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007.

[17] Tzay Y Young. *Handbook of pattern recognition and image processing (vol. 2): computer vision*. Academic Press, Inc., 1994.

[18] Image classification techniques in remote sensing, 2015. URL: http://gisgeography.com/image-classification-techniques-remote-sensing/.

[19] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, Jul 2002. doi:10.1109/TPAMI.2002.1017616.

[20] L. McInnes and J. Healy. Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42, Nov 2017. doi:10.1109/ICDMW.2017.12.

[21] Faisal Kamiran and Toon Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, 2012.

[22] Nearest neighbor classification guide in ecognition, 2015. URL: http://gisgeography.com/nearest-neighbor-classification-guide-ecognition/.

[23] Amjad Rehman, Dzulkifli Mohamad, and Ghazali Sulong. Implicit vs explicit based script segmentation and recognition: a performance comparison on benchmark database. *Int. J. Open Problems Compt. Math*, 2(3):352–364, 2009.

[24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[26] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[27] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition, 2015. URL: `http://cs231n.github.io/neural-networks-1/`.

[28] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[29] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[30] Fjodor van Veen. The asimov institute, 2016. URL: `http://www.asimovinstitute.org/neural-network-zoo/`.

[31] Andrew Tchircoff. The mostly complete chart of neural networks, explained, 2017. URL: `https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464`.

[32] Adit Deshpande. A beginner's guide to understanding convolutional neural networks – adit deshpande – cs undergrad at ucla ('19), 2016. URL: `https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/`.

[33] Dynamic Stardust. Why convolutions always use odd-numbers as filter size, 2017. URL: `http://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size`.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[35] Leonardo Araujo dos Santos. Artificial intelligence, 2018. URL: `https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pooling_layer.html`.

[36] Ujjwal Karn. An intuitive explanation of convolutional neural networks, 2016. URL: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[37] Adobe scan mobile launch video, 2017. URL: https://video.tv.adobe.com/v/18742t1/?autoplay=true.

[38] Venus Wong. Abbyy finereader. URL: http://www.wclsolution.com/products/abbyy/abbyy-finereader-14/.

[39] Tesseract. https://github.com/tesseract-ocr/tesseract, 2017.

[40] Tesseract. https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00, 2016.

[41] Gocr, 2013. URL: http://jocr.sourceforge.net/.

[42] Rebecca Tarnopol. How to ocr documents for free in google drive, 2017. URL: https://business.tutsplus.com/tutorials/how-to-ocr-documents-for-free-in-google-drive--cms-20460.

[43] Introduction, 2017. URL: https://www.tensorflow.org/serving/.

[44] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[45] The gnu general public license v3.0 - gnu project - free software foundation, 2007. URL: https://www.gnu.org/licenses/gpl-3.0.en.html.

[46] What is docker?

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

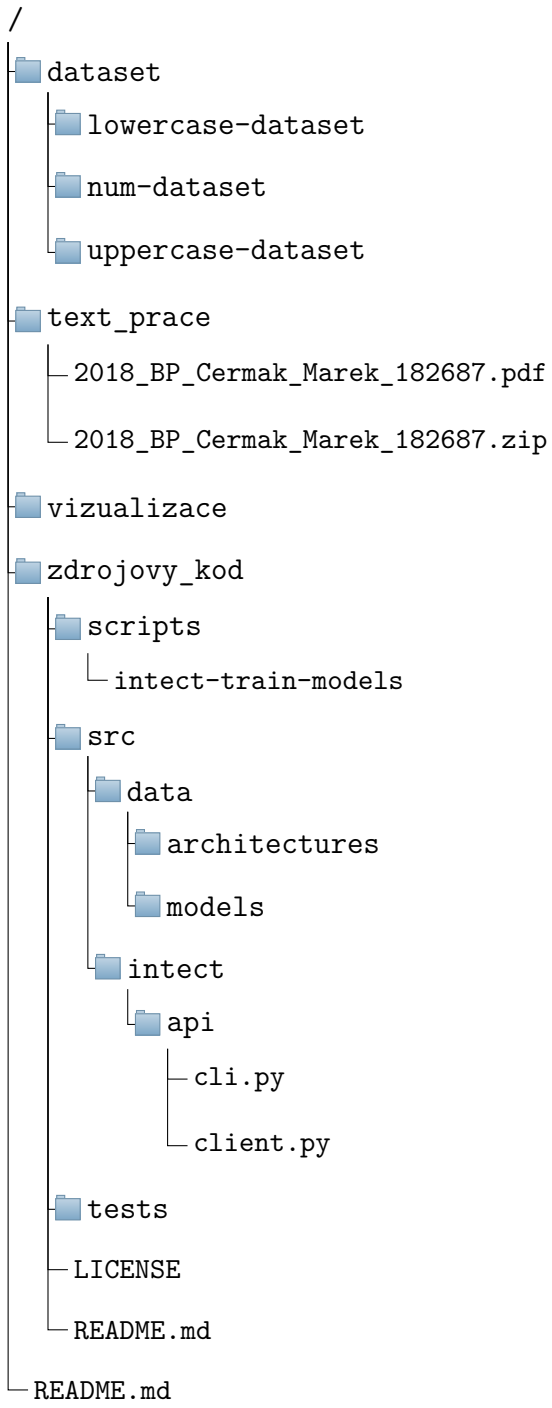
AI	Umělá inteligence – *Artificial Intelligence*
ANN	Umělá neuronová síť – *Artificial Neural Network*
CNN	Konvoluční neuronová síť – Convolutional Neural Network
HMM	Skrytý Markovův model – *Hidden Markov Model*
FCL	*Fully Connected Layer*
GUI	Grafické uživatelské rozhraní – *Graphical User Interface*
HCA	*Hierarchical Clustering*
HPP	*Horizontal projection profile*
KNN	K-Nejbližších sousedů – K-Nearest Neighbors
LSTM	Long / Short Term Memory Network
ML	Strojové učení – *Machine Learning*
MLOCR	Víceřádkový optický rozpoznávač znaků – Multiline optical character reader
MLP	*Multi Layer Perceptron*
NN	*Neuronová síť – Neural Network*
OCR	Optické rozpoznávání znaků – Optical Character Recognition
PCA	Metoda hlavních komponent – *Principal Component Analysis*
PyPI	*Python Package Index*
ReLU	*Rectified Linear Unit*
RCNN	*Region Proposal Convolutional Neural Network*
RNN	Rekurentní neuronová síť – Recurrent Neural Network
SVM	Support Vector Machine
SLP	*Single Layer Perceptron*
SSD	*Single Shot Detector*
YOLO	*You Only Look Once*

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

Součástí této práce je datový nosič obsahující data potřebná k reprodukci výsledků uvedených v tomto textu a vlastní text práce.

Adresářová struktura CD je následující (obsahuje pouze výpis nejdůležitějších souborů a složek):



A.1 Dataset

Adresář *dataset* obsahuje tři podadresáře – data pro jednotlivé modely neuronové sítě. V každém z těchto podadresářů jsou trénovací, testovací a augmentovaná data. Augmentovaná data byla použita pro vyhodnocení přesnosti při tvorbě konfúzní matice. Struktura jednotlivých datasetů odpovídá specifikaci Keras[1], tj. každá složka odpovídá jedné třídě, jejíž označení je totožné s názvem této složky.

A.2 Text práce

Adresář *text_prace* obsahuje text práce v elektronické podobě, tj. PDF soubor, a zdrojové LaTeX soubory pro tuto práci.

A.3 Vizualizace

Adresář *vizualizace* obsahuje grafické a multimediální soubory, jako jsou obrázky, grafy a videa, které nebyly použity v této práci, nebo na něž bylo v tomto textu odkázáno pro lepší pochopení či rozlišení.

A.4 Zdrojový kód

Adresář *zdrojovy_kod* obsahuje veškerý zdrojový kód potřebný k reprodukci výsledků této práce. Nejdůležitějšími složkami jsou *src/intect*, pojmenovaný podle názvu projektu – *Inteligent neural network architect* – s vlastním kódem k aplikaci. Zde lze rovněž najít podložku *api*, ve které jsou vstupní *entrypointy*[2] ke klientovi a skript spustitelný z příkazového řádku. Druhou důležitou složkou v adresáři *src* je složka *data*. Tento adresář obsahuje před trénované modely neuronových sítí, které lze rovnou využít k predikci, či poslat serveru jako modely určené k obsluze. Jsou zde rovněž definovány architektury jednotlivých modelů.

Mimo adresář *src* jsou zde i složky *scripts*, ve kterém je spustitelný skript k paralelnímu trénování neuronových sítí, a *tests*, ve které jsou aplikační unit testy a část integračních testů.

[1] Keras je vysokoúrovňové aplikační prostředí pro strojové učení.

[2] Entrypointy jsou skripty, které se spouští po spuštení API a ke kterým uživatel přistupuje.

B NÁVOD

B.1 Instalace s platformou Docker

Docker je platforma pro kontejnerizaci poskytující kompletní řešení organizace a orchestrace kontejnerů pro řešení širokého spektra požadavků. Je výhodné projekt nasadit v izolovaném produkčním prostředí odpovídajícím prostředí vývojovému. Součástí zdrojového kódu je i specifikace kontejnerizovaného prostředí. Tato forma instalace tohoto projektu je doporučena. Dokumentaci pro Docker lze nalézt v [46].

V kořenovém adresáři lze nalézt dva soubory – *docker-compose.min.yml* a *docker-compose.yml*. Přípona **.min* v tomto případě značí minimální konfiguraci, tj. bez nasazení serveru. Toto použití je ideální pro seznámení se s formou práce s aplikací, trénování vlastního modelu a jednoduché predikce. Druhá konfigurace nasadí i server obsluhy a bude tak možné zasílat klientské požadavky. Konfigurace i obsluha serveru je nad rámec této práce, proto zde bude popsána pouze minimální instalace.

Minimální instalace

Poznámka č.1: Předpokládá se, že Docker je nainstalován a správně nastaven dle oficiální dokumentace pro danou systémovou platformu.

Poznámka č.2: následujícím textu bude pro snazší orientaci pod pojmem „kořenový adresář“ rozuměn adresář zdrojovy_kod.

Z kořenového adresáře spusťte v terminálovém okně následující příkaz:

```
docker-compose -f docker-compose.min.yml up
```

Tento příkaz spustí build a nastavení kontejneru. Toto může trvat i několik desítek minut. Po dokončení bude v daném okně automaticky spuštěno logování výstupů kontejneru. Okno nezavírejte, tyto informace mohou být užitečné pro kontrolu chyb.

V novém terminálovém okně teď proveďte následující příkaz:

```
docker ps
```

Výstupem by měl být výpis, ve kterém bude běžící kontejner se jménem *intect_minimal*.

Kontejner zpřístupnil uživateli port s vizualizací *TensorBoard*. Na tento port můžete přistoupit ze svého prohlížeče zadáním adresy `localhost:6006`.

Do prostředí kontejneru se lze přenést následujícím příkazem:

```
docker exec -it intect\_minimal bash
```

B.2 Lokální instalace

Pokud preferujete lokální prostředí, je doporučeno nastavit si virtuální Python prostředí.

Z kořenového adresáře spusťte postupně následující příkazy:

```
python3 -m venv env
source env/bin/activate
```

A pro instalaci projektu samotného:

```
pip install -r requirements.txt
python setup.py install
```

B.3 Základní použití uživatelského rozhraní

Poznámka: Předpokládá se, že v kořenovém adresáři projektu je obsažena složka dataset.

Nápovědu lze zobrazit příkazem bez jakýchkoli parametrů:

```
intect
```

Pro samostatné trénování modelu lze využít následující příkaz:

```
# trénování i ohodnocení
intect --train --eval --export \
    --model_arch="dataset/num-dataset/num-architecture.yaml" \
    --train_dir="dataset/num-dataset/train_data" \
    --test_dir="dataset/num-dataset/test_data" \
    --train_epochs=5
```

Trénování modelu by mělo trvat v závislosti na výkonu počítače zhruba několik minut. Pokud jste spustili trénování v prostředí kontejneru, je možné sledovat živou vizualizaci trénovacího procesu v prohlížeči.

Protože máme model natrénován, lze si vyzkoušet i predikci.

```
# predikce
intect --predict \
    --model_arch="dataset/num-dataset/num-architecture.yaml" \
    --model_dir=".model_cache/numbers,lr=0.005,bs=32,conv=2,fcl=1" \
    --images="tests/data/default.png"  # nebo jakykoli jiny obrazek
```