VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ACTIVE LEARNING PRO ZPRACOVÁNÍ ARCHIVNÍCH PRAMENŮ

ACTIVE LEARNING FOR PROCESSING OF ARCHIVE SOURCES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. DAVID HŘÍBEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Hříbek David, Bc.**
Program: Informační technologie Obor: Strojové učení
Název: **Active Learning pro zpracování archivních pramenů**
Active Learning for Processing of Archive Sources
Kategorie: Umělá inteligence

Zadání:

1. Nastudujte různé přístupy pro rozpoznávání ručně psaného textu a metodu Active Learning. Zaměřte se na použití pro rozpoznávání historických pramenů.
2. Navrhněte systém, který bude schopen používat různé implementace OCR, tzn. uživatel si bude moci vybrat buď mezi už existujícím OCR nebo vámi vlastnoručně vytvořeným OCR. Existující OCR volte s ohledem na možnost do nich doplňovat další softwarovou funkcionalitu. Navrhněte, jak implementovat doučování ze záznamů opravených uživateli nejen u převzatých OCR, ale i u vámi vytvořeného OCR.
3. Vytvořte kompletní navržený systém, implementujte vlastní OCR a dále vytvořte automatický systém doučování, který bude systém pravidelně doučovat podle nově opravených záznamů. Umožněte také full textové vyhledávání v rozpoznaných datech.
4. Systém otestujte a vyhodnoťte výpočetní náročnost pro enormní množství skenů a úspěšnost rozpoznávání.

Literatura:

- V. Romero, J. A. Sánchez and A. H. Toselli, "Active Learning in Handwritten Text Recognition using the Derivational Entropy," *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Niagara Falls, NY, 2018, pp. 291-296

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

V této práci je řešeno vytvoření systému, který umožňuje nahrání a anotaci skenů historických dokumentů a následné aktivní doučování modelů pro rozpoznávání znaků (OCR) na dostupných anotacích (vyznačených řádcích a jejich přepisech). V práci je popsán proces, klasifikovány techniky a uveden existující systém pro rozpoznávání znaků. Především je kladen důraz na metody strojového učení. Dále jsou vysvětleny metody aktivního učení a navrhnut způsob doučování OCR modelů z anotovaných skenů. Zbytek práce se zabývá konkrétním návrhem, implementací, dostupnými datasety, vyhodnocením úspěšnosti rozpoznávání znaků vlastnoručně vytvořeného OCR modelu a testováním celého systému.

Abstract

This work deals with the creation of a system that allows uploading and annotating scans of historical documents and subsequent active learning of models for character recognition (OCR) on available annotations (marked lines and their transcripts). The work describes the process, classifies the techniques and presents an existing system for character recognition. Above all, emphasis is placed on machine learning methods. Furthermore, the methods of active learning are explained and a method of active learning of available OCR models from annotated scans is proposed. The rest of the work deals with a system design, implementation, available datasets, evaluation of self-created OCR model and testing of the entire system.

Klíčová slova

Strojové učení, učení s učitelem, aktivní učení, OCR, optické rozpoznávání znaků, aktivní učení pro rozpoznávání ručně psaného textu, anotace skenů historických dokumentů.

Keywords

Machine learning, supervised learning, active learning, OCR, optical character recognition, active learning in handwritten text recognition, annotation of historical document scans.

Citace

HŘÍBEK, David. *Active Learning pro zpracování archivních pramenů.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Active Learning pro zpracování archivních pramenů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Další informace mi poskytli Ing. Pavel Svoboda, Ph.D. a Ing. Michal Hradiš, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Hříbek
11. května 2021

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Jaroslavu Rozmanovi, Ph.D. za pomoc a ochotu při vedení mé diplomové práce. Dále pak panu Ing. Michalu Hradišovi, Ph.D. a Ing. Pavlu Svobodovi, Ph.D. za konzultace k této diplomové práci. Také bych rád poděkoval rodině a blízkým za jejich podporu.

Obsah

1 Studium problematiky **3**
1.1 Optické rozpoznávání znaků . . . 3
1.1.1 Klasifikace OCR systémů . . . 4
1.1.2 Proces rozpoznávání znaků . . . 5
1.1.3 Existující OCR systém . . . 11
1.1.4 Systémy pro rozpoznávání znaků pomocí neuronových sítí . . . 12
1.2 Aktivní učení . . . 16
1.2.1 Varianty aktivního učení . . . 17

2 Analýza požadavků a návrh řešení **18**
2.1 Cíl práce . . . 18
2.2 Návrh architektury systému . . . 19
2.3 Návrh způsobu doučování modelů z opravených záznamů . . . 20
2.4 Návrh databáze . . . 21
2.5 Návrh způsobu segmentace řádků ve skenech dokumentů . . . 23

3 Implementace a testování **24**
3.1 Použité technologie a nástroje . . . 24
3.2 Moduly systému . . . 25
3.2.1 API pro kontejnery vykonávajícími úkoly nad datasety . . . 25
3.2.2 Automatická detekce řádků a textových regionů v obraze . . . 27
3.2.3 Tesseract OCR . . . 28
3.2.4 Vlastnoručně vytvořený OCR model . . . 30
3.2.5 Webová aplikace . . . 39
3.3 Testování systému . . . 51
3.3.1 Použité datasety pro vyhodnocení úspěšnosti modelu CRNN OCR . 51
3.3.2 Testování úspěšnosti rozpoznávání znaků modelem CRNN OCR . . 52
3.3.3 Testování webové aplikace . . . 54

4 Závěr **56**

Literatura **58**

A Obsah přiloženého paměťového média **61**

B Ukázky skenů z použitých datasetů **62**

Úvod

V současnosti je k dispozici velké množství psaného textu v podobě různých dokumentů a knih. Velice populární je také obor genealogie, který zkoumá vztahy mezi lidmi a jejich původ. Nejčastějším informačním pramenem, který se v oboru genealogie používá, jsou matriky. V dnešní době jsou matriky skenovány a převáděny tak do digitální podoby, ve kterých si je lidé mohou prohlížet, aniž by museli jezdit do archivu. Velkým problémem je ale vyhledávání informací v takovýchto naskenovaných dokumentech. Pro vyhledávání informací v naskenovaných dokumentech je potřeba nejdříve tyto dokumenty přečíst a následně najít požadovanou informaci. Tento proces je velice zdlouhavý a neefektivní. Jedním z možných řešení je přepis textu z těchto naskenovaných dokumentů do elektronické podoby a následné vyhledávání v tomto textu pomocí počítače. Samotný přepis textu z naskenovaných dokumentů do elektronické podoby je ale také velice časově náročná činnost, proto se dnes stále častěji nahrazuje strojovými metodami, které umožní automatický přepis naskenovaného textu do elektronické podoby. Takovýmto metodám se říká optické rozpoznávání znaků (*Optical Character Recognition* – OCR). V dnešní době jsou metody OCR často založené na umělé inteligenci a neuronových sítích, které pro své fungování vyžadují velké množství dat v podobě naskenovaných dokumentů a k nim přepsaným textům, ze kterých se následně učí jak správně text přepisovat. Data, ze kterých se neuronové sítě učí, musí být ve speciálním formátu a proces jejich získávání je rovněž velice časové náročný, z tohoto důvodu se začínají používat metody Aktivního učení (*Active Learning*), které umožňují uživatelům interaktivním způsobem přidávat nová data pro učení a opravovat již existující data.

Cílem této práce je vytvoření sytému, který svým uživatelům umožní nahrání naskenovaných dokumentů, které bude následně možné pomocí tohoto systému anotovat a získat tak data potřebná pro učení OCR modelů. Dále pak vytvoření vlastního OCR modelu, který bude možné z takto získaných dat doučovat a vylepšovat tak kvalitu jeho přepisu. Vytvořený OCR model bude zabudován do výsledného sytému a uživatelé tak budou mít možnost spuštění tohoto modelu na svých nahraných dokumentech. Přidáním nových anotací nebo opravením anotací vzniklých spuštěním zabudovaného OCR modelu bude možné tento model doučovat.

Celá práce je rozdělena do 4 kapitol. V kapitole „Studium problematiky“ jsou popsány teoretické znalosti, které bylo nutné nastudovat pro dokončení této práce. V kapitole s názvem „Analýza požadavků a návrh řešení“ je navrhnut systém, který splňuje požadavky popsané v předchozím odstavci. Je zde představena architektura systému a jednotlivé moduly, ze kterých se tento systém skládá, včetně způsobu komunikace mezi nimi. Poslední kapitola se zabývá implementací tohoto systému a jeho následným testováním. Nejdříve jsou popsány použité technologie a nástroje, poté samotná implementace jednotlivých modulů systému. Zbytek kapitoly je pak věnován testování celého systému, vyhodnocení výpočetní náročnosti pro enormní množství skenů a úspěšnosti rozpoznávání.

Kapitola 1

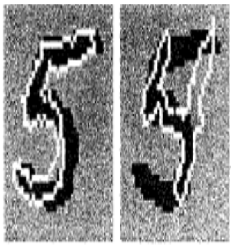
Studium problematiky

V této kapitole budou popsány teoretické znalosti, které bylo nutné nastudovat pro dokončení této práce. První sekce se bude věnovat optickému rozpoznávání znaků (*Optical Character Recognition*, zkráceně OCR), tedy systémům, které z naskenovaného či vyfoceného dokumentu dokáží zpětně určit, co je v tomto dokumentu napsáno a umožní tak získat zpět text ve znakové podobě. Nejdříve bude popsána stručná historie OCR systémů a ukázány jejich aplikace. Následně budou tyto systémy klasifikovány podle různých hledisek a představen proces rozpoznávání znaků, který se skládá z několika kroků. Každý z těchto kroků bude vysvětlen, popřípadě budou popsány důležité algoritmy, které se zde využívají. Zbytek této kapitoly je věnován OCR modelům, založených na neuronových sítích, dále pak metodám Aktivního učení (*Active Learning*) a způsobu jeho využití.

1.1 Optické rozpoznávání znaků

Metody pro optické rozpoznávání znaků (Optical Character Recognition – OCR) umožňují rozpoznat znaky, které se vyskytují v tištěném, či psaném textu a převést tak tento text do digitální (znakové) podoby, ve které může být dále zpracován pomocí počítače.

Problém rozpoznávání tištěných znaků pomocí stroje [13, 25] byl řešen ještě před masivním rozvojem počítačů. V roce 1954 byl vytvořen stroj, který uměl číst velké znaky anglické abecedy s rychlostí čtení 1 znak za minutu [25]. První OCR modely byly velice nepřesné a pomalé. Z tohoto důvodu došlo k vytvoření standardizovaných fontů písem OCRA a OCRB, které byly navrženy tak, aby je šlo snadno rozpoznávat pomocí stroje a zvýšily tak přesnost OCR systémů.



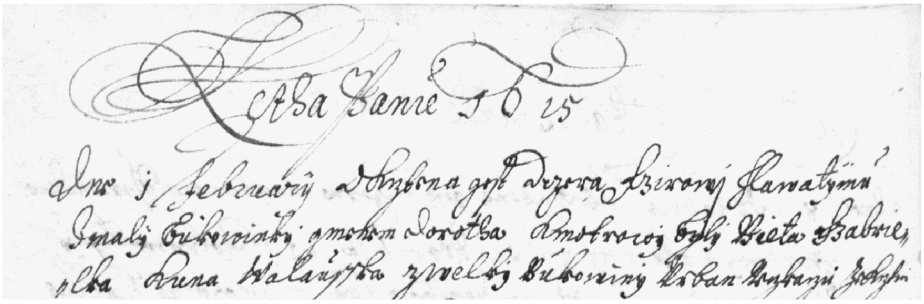
Obrázek 1.1: Ukázka rozpoznávání znaků pomocí porovnávání se vzorovými znaky. Bílý znak na obrázku značí vzorový znak. Obrázek převzat z [21].

První OCR modely [13, 25] byly založeny na porovnávání tištěných znaků s předlohami vzorových znaků (*pattern matching*). Tato technika se ale neuměla vypořádat s odlišnostmi mezi jednotlivými tištěnými dokumenty. Následně byly vyvinuty techniky pro extrakci příznaků a klasifikaci, díky kterým došlo ke zvýšení přesnosti OCR. Kvůli velkému množství různých jazyků a fontů písma je OCR velice komplexní problém, který se stále řeší.

Aplikace OCR systémů Systémy pro rozpoznávání znaků [25] byly velice často využívány pro urychlení procesů, které závisely na přečtení údajů z dokumentů a následném dalším zpracování. Využití se našlo například v bankovnictví pro čtení údajů z šeků nebo také pro třídění pošty. Při třídění poštovních zásilek bylo z počátku rozpoznáváno pouze PSČ, později přibyly také názvy států a měst. Dalším častým využitím OCR je převod tištěných dokumentů do elektronické (znakové) podoby a umožnění tak přístupu k těmto dokumentům pro větší skupinu lidí. Díky převodu do elektronické podoby je také snazší vyhledávání v těchto dokumentech, čehož bude využito i v této práci.

1.1.1 Klasifikace OCR systémů

OCR systémy lze klasifikovat [25] podle mnoha kritérií. Nejdůležitějším kritériem je typ písma – tedy jestli se jedná o tištěné nebo psané písmo. V případě tištěných dokumentů je rozpoznávání textu jednodušší díky jednotnému fontu písma. Rozpoznávání ručně psaného písma je naopak složitější. Každý člověk zapisuje jednotlivé znaky trochu jiným způsobem a vzniká tak mnoho různých stylů zápisu. U ručně psaného písma se také rozlišuje způsob jeho získávání. Dokumenty, které již byly naskenovány nebo vyfoceny a následně převedeny do digitální podoby se řadí do skupiny off-line. Při rozpoznávání jednotlivých znaků v reálném čase, tedy již při jejich zápisu například na obrazovku tabletu, hovoříme o skupině on-line. On-line rozpoznávání znaků má velkou přesnost, jelikož tyto systémy využívají dodatečné informace o vzniku daného textu, jako jsou například rychlost pohybu po obrazovce tabletu, pořadí zápisu atd. OCR modely lze také dělit podle úrovně izolovanosti jednotlivých znaků písem, pro které jsou vyvíjeny. Čím více jsou znaky od sebe izolovány a navzájem do sebe nezasahují, tím snazší je jejich rozpoznávání. Velkou roli také hraje, zda je daný systém vyvíjen pouze pro rozpoznávání jednoho druhu písma, nebo jestli musí umět rozpoznat více různých fontů – takové systémy jsou složitější.



Obrázek 1.2: Ukázka ručně psaného písma ve skenu matričního záznamu z archivu MZA.

Podle přístupu k rozpoznávání textu [25] můžeme rozdělit OCR na systémy, které pracují s uceleným pohledem na daný text (tzv. holistické systémy) a analytické systémy, které se snaží rozdělit text na menší části (např. na jednotlivé znaky) a následně rozpoznat tyto části nezávisle na okolním kontextu. Výhodou holistických systému je jejich menší složitost

a náročnost na implementaci. Analytické systémy jsou méně náchylné na šum v obraze, jejich nevýhodou je ale obtížnost nalezení vhodné dekompozice dokumentu na menší části.

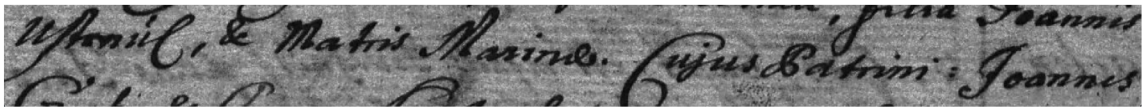
1.1.2 Proces rozpoznávání znaků

Proces rozpoznávání znaků [25, 13] se skládá z 5 kroků:

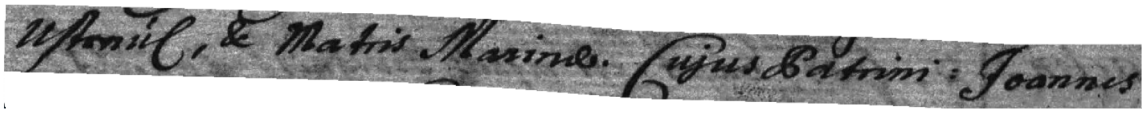
1. **Sběr skenů dokumentů** (image acquistion) je prvním krokem při rozpoznávání textu. Jedná se o proces získávání dokumentů v digitální podobě. Tištěné nebo psané dokumenty jsou naskenovány nebo vyfoceny a převedeny tak do digitální podoby, ve které mohou být dále zpracovávány.

2. **Předzpracování** (*Preprocessing*) je důležitým krokem, při kterém dochází k pročištění dat získaných v předešlém kroku. Skeny dokumentů obsahují informace, které nejsou důležité a žádným způsobem nepřispějí ke správnému rozpoznání textu z těchto dokumentů, a proto je vhodné tyto informace odstranit. Pro odstranění nadbytečných informací ze skenů dokumentů se používají tyto techniky:

    - **Zjištění sklonu písma a otočení** – historické prameny a matriční záznamy jsou často psány ručně, bez dodržování řádků. Jednotlivé řádky jsou pak často nakloněné. Pro rozpoznávaní znaků je ale lepší mít řádky vodorovně, aby se jednotlivá písmena nepřekrývala v jednotlivých sloupcích obrázku. Na následujících obrázcích je zobrazen výřez řádku s latinským textem „Ustrnul, & Matris Marinæ. Cujus Patrini: Joannes“. Obrázek 1.3 zobrazuje řádek, který byl vyřezán pomocí *bounding-boxu*. Obrázek 1.4 zobrazuje stejný řádek textu, kde je ale navíc vyřezán i polygon ohraničující řádek. Obrázek 1.5 zobrazuje výsledný řádek po nalezení sklonu písma a otočení obrázku.

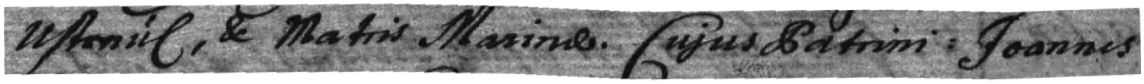


Obrázek 1.3: Ukázka obrázku nakloněného řádku textu v latinsky psané matrice z archivu MZA.



Obrázek 1.4: Ukázka obrázku nakloněného řádku textu v latinsky psané matrice z archivu MZA. Z obrázku řádku je navíc vyřezán polygon, který tento řádek vyznačuje.

    - **Prahování** (*Thresholding/Binarization*) – skeny dokumentů jsou většinou barevné (RGB). Barevnost skenů zásadně nepřispívá ke zvýšení přesnosti rozpoznaného textu, naopak zvyšuje množství a komplexnost dat, a je tak výhodné dokument převést do odstínů šedi. Obrázek v odstínech šedi stále obsahuje velké množství informací o barvách jednotlivých pixelů. Hodnoty těchto pixelů mohou být v rozmezí 0-255, proto je stanoven práh v tomto intervalu a všechny pixelu s hodnotou menší než hodnota tohoto prahu jsou nastaveny na 0, ostatním pixelům je pak nastavena hodnota 255. Pokud možné hodnoty omezíme na 0 a 1,



Obrázek 1.5: Ukázka obrázku řádku textu v latinsky psané matrice z archivu MZA. Z obrázku řádku je vyřezán polygon, který tento řádek vyznačuje a navíc je řádek otočen tak, aby byl text vodorovně.

obrázek bude binarizován. Tímto dojde ke zvýraznění textu na obrázku a k utlumení informací v pozadí. Na obrázku 1.6 je ukázaná aplikace převodu obrázku do odstínů šedi a následné prahování.

Obrázek 1.6: Ukázka aplikace prahování na naskenovaný dokument z archivu MZA. Vlevo je zobrazen sken po převedení do odstínů šedi, vpravo pak sken po dodatečném prahování.

- **Inverze barev** – ve většině dokumentů je ke zvýraznění písma použita tmavá barva písma v kombinaci se světlejším pozadím. V naskenovaných dokumentech, které byly pomocí prahování převedeny na černobílé (například obrázek 1.6 vpravo) pak pixely s nízkou hodnotou (černé) reprezentují text – popředí – a pixely s vysokou hodnotou (bílé) reprezentují pozadí. Pro účely rozpoznávání znaků ale pozadí nepřináší žádnou užitečnou informaci. Jelikož ale pozadí většinou tvoří větší část obrázku, je pro další počítačové zpracování výhodnější uchovávat vysoké hodnoty pouze u pixelů reprezentujících text (přinášejících užitečné informace). Následující algoritmy většinou také předpokládají vysoké hodnoty u pixelů představujících popředí. Barvy naskenovaného obrázku jsou tedy invertovány. Ukázka inverze barev pravého obrázku 1.6 je zobrazena na levém obrázku 1.7.
- **Odstranění šumu** (*Noise removal*) – naskenované dokumenty mohou obsahovat různé degradace obrazu jako například šum. Pro odstranění šumu v obraze je možné využít morfologické operace[1], které pracují s binárními maskami, kde pixely s vysokou hodnotou reprezentují popředí (v tomto případě text) a pixely s nízkou hodnotou reprezentují pozadí. Pomocí morfologické operace *open* lze odstranit šum v popředí – tedy osamocené shluky pixelů, které například mohou přestavovat skvrny, kaňky nebo jiný druh znečištění dokumentu. Ukázka před a po aplikace operace *open* je zobrazena na obrázku 1.7. Morfologická operace *close* naopak umožňuje odstranit šum v pozadí – tedy shluky pixelů, které narušují popředí a vytváří v něm tak díry.

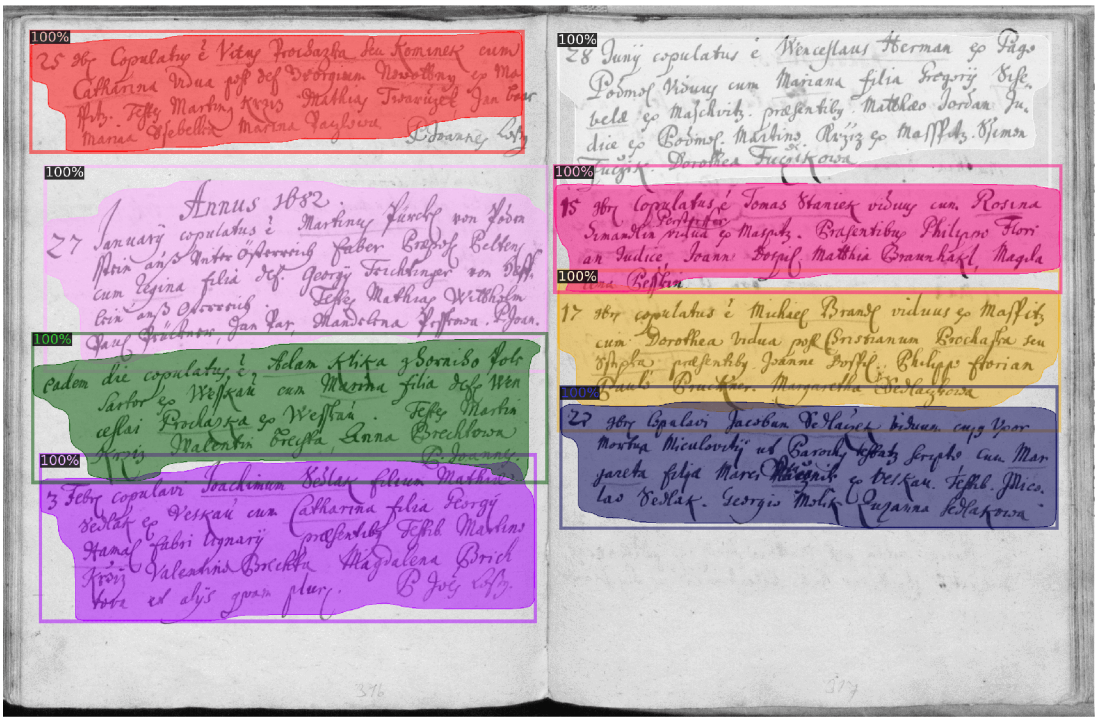
3. **Segmentace** (*Segmentation*) – součástí dokumentu mohou být obrázky a jiné netextové segmenty, které je nutno ignorovat nebo odstranit. Cílem této fáze je tedy dekompozice skenu dokumentu na jednotlivé části, které spolu logicky souvisí a odfiltrování

[1] https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

Obrázek 1.7: Ukázka obrázku před a po aplikaci morfologické operace *open* – odstranění šumu v popředí.

segmentů, které nebudou dále zpracovávány pomocí OCR. Této fázi segmentace se říká segmentace úrovně 1.

Následuje segmentace úrovně 2, kam vstupují pouze textové segmenty. Tyto jsou dále dekomponovány na samostatné odstavce popřípadě řádky textu. Pokud dále segmentujeme textové řádky na jednotlivé znaky nebo malé skupiny znaků (v případě psaného písma, kde na sebe znaky navazují), jedná se o segmentaci úrovně 3. Pro další zpracování pomocí OCR jsou používány výstupy segmentace úrovně 2 nebo 3. Na obrázku 1.8 je zobrazena ukázka dekompozice skenu dokumentu na jednotlivé bloky textu (záznamy).



Obrázek 1.8: Ukázka dekompozice skenu dokumentu z archivu MZA na jednotlivé odstavce – segmentace úrovně 2. Dekompozice provedena pomocí neuronové sítě Mask-RCNN [34], kterou byly detekovány odstavce.

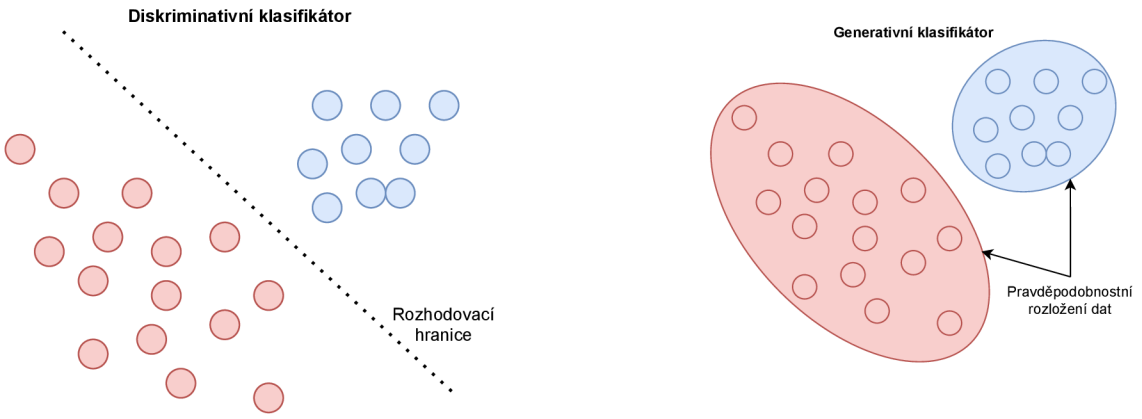
4. **Extrakce příznaků** (*Feature extraction*) – výstupy segmentace úrovně 2 (odstavce/řádky textu) nebo úrovně 3 (izolované znaky nebo menší skupiny znaků) obsahují stále velké množství informací, které nemusí být relevantní pro správné rozpoznání znaků v těchto segmentech. Cílem této fáze je tedy extrahovat ze segmentů získaných v předešlém kroku relevantní příznaky – tedy příznaky minimalizující odlišnosti

mezi stejnými třídami (znaky) a maximalizují odlišnosti mezi různými třídami [13]. Výstupem jsou pak vektory reprezentující extrahované příznaky – *feature vectors*.

Extrahované příznaky lze dělit do 4 kategorií [25, 28] – strukturální, statistické, globální a korelační. Mezi strukturální příznaky může například patřit sklon, délka nebo tloušťka písma. Statistické příznaky se extrahují pomocí výpočtu statistických funkcí nad získanými segmenty – např. Fourierova transformace, statistické momenty nebo hustota pixelů... Globální příznaky se zaměřují na redukci dimenzionality celého vstupního segmentu. Typickým představitelem této kategorie je Analýza hlavních komponent (*Principal Component Analysis – PCA* [9]), díky které lze redukovat dimenzionalitu dat odstraněním dimenzí s nižší variabilitou. Korelační příznaky se používají pro porovnání segmentů s jejich vzorovými předlohami (např. porovnání izolovaného znaku se vzorovými znaky abecedy) a zjištění míry korelace.

V současné době je také velice populární extrakce příznaků pomocí hlubokých konvolučních neuronových sítí. Neuronová síť může být učena například pro klasifikaci znaků, následně jsou ze sítě odebrány koncové vrstvy určené pro klasifikaci a vznikne tak síť, která pro vstupní segment úrovně 3 (znak) vrátí vektor příznaků, který jej reprezentuje.

5. **Klasifikace a rozpoznání** (*Classification and Recognition*) je poslední fází, při které jsou extrahované příznaky klasifikovány do předem určených tříd, představujících (v případě OCR) jednotlivé znaky abecedy, číslice nebo další symboly, které chceme rozpoznávat. Pro klasifikaci lze rovněž použít různé přístupy. Obecně se klasifikátory rozdělují na *generativní* a *diskriminativní* [22]. Tyto klasifikátory využívají ke klasifikaci rozdílný přístup. Ukázka fungování takových klasifikátorů je zobrazena na obrázku 1.9.



Obrázek 1.9: Ukázka odlišnosti přístupů diskriminativních a generativních klasifikátorů. Jednotlivá data jsou rozdělena do tříd pomocí barev. Generativní klasifikátory se snaží naučit pravděpodobnostní rozložení dat, a to využít k predikci cílové třídy. Diskriminativní klasifikátory pro klasifikaci využívají rozhodovací hranici, pomocí které určují do jaké třídy daný vzorek patří.

Generativní modely [5] se snaží naučit pravděpodobnostní rozložení jednotlivých tříd a dat – tedy společnou pravděpodobnost třídy a dat: $P(Class, Features)$. Pomocí Bayesova vzorce (rovnice 1.1) je poté možné odvodit pravděpodobnost třídy

v závislosti na datech a tu použít pro klasifikaci data do třídy s největší pravděpodobností. Generativní modely jsou robustnější než diskriminativní a je výhodné je používat, pokud nemáme dostatečné množství dat. Jelikož generativní modely modelují pravděpodobnostní rozložení dat, je možné pomocí tohoto rozložení generovat data nová.

Mezi představitele generativních modelů patří *Naive Bayes* [22] – naivní použití Bayesova vzorce z rovnice 1.1. Často využívaným rozložením pravděpodobnosti je Normální rozložení. Toto rozložení využívá model *Gaussian Mixture Model – GMM* – lineární kombinace Gausovských rozložení. Více komplexnějším modelem jsou Bayesovské sítě (*Bayesian Netowrks*) [6], které slouží k vytváření komplexních modelů z jednodušších rozložení pravděpodobnosti. Dalšími modely jsou *Markov Random Field – MRF* [6], *Hidden Markov Models – HMM*...

$$P(Class|Features) = \frac{P(Features|Class) \times P(Class)}{P(Features)} = \frac{P(Class, Features)}{P(Features)} \tag{1.1}$$

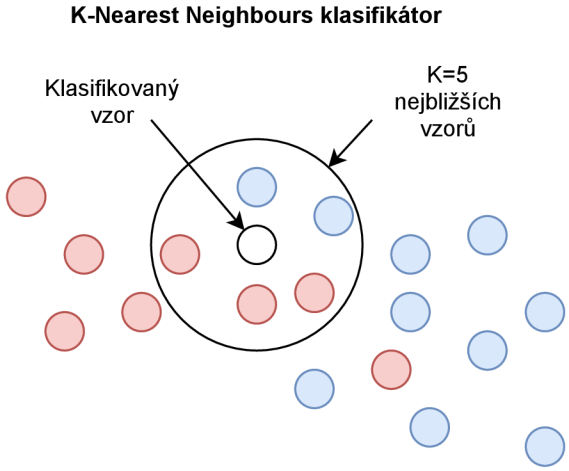
Diskriminativní modely [5] se narozdíl od generativních modelů nesnaží naučit pravděpodobnostní rozložení dat, ale pouze rozhodovací hranice mezi jednotlivými třídami. Tyto klasifikátory se tedy učí rovnou pravděpodobnost dané třídy v závislosti na datech: $P(Class|Features)$. Diskriminativní modely jsou více náchylné k přetrénování v případě malého množství dat a mají také menší schopnost generalizace. Jelikož diskriminativní modely mají většinou větší množství trénovatelných parametrů, lze jimi popsat komplexní rozhodovací hranici a řešit tak více komplexní problémy.

Typickým představitelem těchto modelů je Logistická regrese [6], jejímž cílem je lineárně separovat data na dvě třídy pomocí rozhodovací hranice a následně interpretovat vzdálenost dat od této hranice jako pravděpodobnost, že daný vzorek patří resp. nepatří do dané třídy. Vzorec pro výpočet pravděpodobnosti třídy na základě dat je ukázán v rovnici 1.2 – nejdříve je proveden skalární součin mezi daty (extrahovanými příznaky) a maticí vah (parametry modelu získanými při trénování) a přičtení posuvu. Výsledná hodnota je následně interpretována jako pravděpodobnost pomocí funkce sigmoida, která vrací hodnoty v intervalu $< 0; 1 >$.

$$P(Class|Features) = Sigmoid(Weights \times Features + Bias) \tag{1.2}$$

Dalším modelem je *Support Vectors Machine – SVM* [6], který pracuje velmi podobně jako Logistická regrese, ale navíc se snaží maximalizovat vzdálenost dat jednotlivých tříd od rozhodovací hranice. Díky kernel funkcím, které mapují data do vysoce dimenzionálního prostoru a v tomto prostoru mezi nimi počítají skalární součin, lze tímto modelem najít rozhodovací hranici i pro lineárně neseparovatelné třídy. V dnešní době často využívaným modelem jsou neuronové sítě, které můžeme chápat jako kaskádu lineárních klasifikátorů, díky kterým lze také řešit i lineárně neseparovatelné problémy. Často využívaným modelem je také model *Decision Trees*, který pracuje na principu rekurzivního dělení prostoru s důrazem na maximalizaci zisku informací [5]. Mezi klasifikátory, které nemají žádné učitelné parametry patří například algoritmus

K-Nearest Neighbours [6], který o příslušnosti data k dané třídě rozhoduje na základě tříd K nejbližších dat, která se nacházejí v okolí klasifikovaného data. Ukázka klasifikace data do příslušné třídy je zobrazena na obrázku 1.10.



Obrázek 1.10: Ukázka klasifikace dat pomocí algoritmu *K-Nearest Neighbours*. Data jsou rozdělena do tříd podle barev. Bílý kruh reprezentuje vzory, které ještě nebyly klasifikovány. Nejdříve je nalezeno K=5 nejbližších sousedů daného vzoru a sečteny počty vzorů jednotlivých tříd. V okolí se nachází 2 vzory z modré třídy a 3 vzory z červené třídy – vzor bude klasifikován jako patřící do červené třídy.

6. **Vyhodnocení úspěšnosti rozpoznávání znaků** Pro vyhodnocení úspěšnosti rozpoznávání textu se používají metriky [24, 2, 3] *Character Error Rate – CER* a *Word Error Rate – WER*. Tyto metriky indikují procento znaků/slov textu, které nebylo OCR modelem správně rozpoznáno. Rovnice 1.3 popisuje výpočet *CER* – i značí počet operací vložení znaku, s počet operací nahrazení znaku jiným znakem a d počet operací smazání znaku. Rovnice 1.3 popisuje výpočet *CER* – i značí počet operací vložení znaku, s počet operací nahrazení znaku jiným znakem, d počet operací smazání znaku a n počet znaků celkem. Rovnice 1.4 pak popisuje výpočet *WER*, kde i_w značí počet operací vložení slova, s_w počet operací nahrazení slova jiným slovem, d_w počet operací smazání slova a n_w počet slov celkem.

$$CER = (i + s\ + d)/n \tag{1.3}$$

$$WER = (i_w + s_w + d_w)/n_w \tag{1.4}$$

1.1.3 Existující OCR systém

Rozpoznání znaků je dlouho řešený problém, a proto již bylo vyvinuto mnoho knihoven a hotových produktů, které usnadňují implementaci a testování OCR modelů. Nejznámnější knihovnou je patrně **Tesseract**[2], která je vyvíjená společností Google nebo nástroj **Matlab**[3] od firmy Mathwork Inc. [28]. Na obrázku 1.11 je zobrazena ukázka rozpoznání textu z obrázku pomocí knihovny Tesseract.

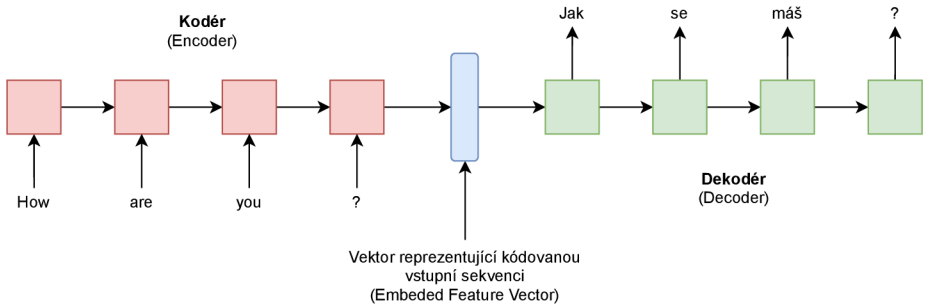
Obrázek 1.11: Ukázka rozpoznání textu z obrázků pomocí knihovny Tesseract. Jedná se o skeny úryvků novin z roku 1913-1914, které mi byly poskytnuty v rámci fakulty (původní zdroj www.digitalniknihovna.cz, kolekce MZK). Noviny byly nahrány do výsledného systému, který detekoval řádky a textové regiony (segmentace úrovně 2) a následně systém pomocí knihovny Tesseract tyto noviny přečetl. Výsledné přepisy pro označené řádky lze vidět v horní části obrázků.

První a druhá verze knihovny **Tesseract** [1] nezahrnovala analýzu rozložení stránky (segmentace) a podporovala pouze rozpoznávání anglického textu. Dnes je tato knihovna ve verzi 4 a podporuje 116 jazyků (včetně češtiny, slovenštiny, němčiny...) pro rozpoznání textu, včetně trénování OCR modelů pro práci s jinými jazyky nebo typy písma. Dále je podporována analýza rozložení stránky nebo detekce proporcionálního a neproporcionálního písma. Systém Tesseract verze 4 využívá pro rozpoznávání znaků rekurentní neuronové sítě LSTM [30].

[2]https://github.com/tesseract-ocr/tesseract
[3]https://www.mathworks.com/products/matlab.html

1.1.4 Systémy pro rozpoznávání znaků pomocí neuronových sítí

S rozmachem hlubokých konvolučních neuronových sítí (*Deep Convolutional Neural Netowrks – DCNN* [19]) v roce 2012 došlo k velkému posunu v oboru počítačového vidění. Konvoluční sítě umožnily snížit výpočetní náročnost a počet trénovatelných parametrů, což jsou hlavní důvody proč je nevhodné využít tradiční plně propojené sítě ke zpracování obrazových dat. Díky hlubokým konvolučním sítím tak bylo umožněno poměrně rychle extrahovat příznaky ze strukturovaných dat, jako jsou například obrázky.



Obrázek 1.12: Zjednodušená architektura neuronové sítě typu Kodér-Dekodér pro překlad textu z angličtiny do češtiny. Vstupem je sekvence anglických slov, která jsou zakódována v podobě *embeded* vektoru a následně dekódována do ekvivalentní výstupní sekvence v češtině.

DCNN byly uplatněny [29] i v modelech pro rozpoznávání sekvencí objektů v obraze, jako jsou například znaky textu. Tyto modely pracovaly na principu detekce jednotlivých znaků v obraze, následné extrakce příznaků pomoci DCNN a klasifikace těchto příznaků do tříd odpovídajících jednotlivým znakům. DCNN jsou ale často omezeny na fixní velikost vstupu a výstupu, a proto nemohou být použity pro generování sekvencí proměnné délky, která je potřebná pro rozpoznání sekvence objektů v obraze. Další se model [14, 29] snažil řešit problém proměnné délky sekvencí pomocí detekce celých slov (sekvence znaků oddělená mezerou) v obraze, ze kterých následně extrahoval příznaky pomocí DCNN a tyto pak klasifikoval do tříd odpovídajících jednotlivým anglickým slovům. Celkem bylo možné každou sekvenci znaků klasifikovat do jedné z cca 90 tisíc tříd (slov). Díky velkému počtu tříd ale tento model špatně generalizoval a nebylo ho možné použít pro některé typy jazyků jako je např. čínština, kde by počet tříd byl řádově větší.

Konvoluční neuronové sítě nemohou být přímo využity pro zpracování sekvenčních dat, a tak začaly být využívány neuronové sítě, které pro zpracování sekvenčních dat byly přímo navrženy – rekurentní neuronové sítě (*Recurrental Neural Network – RNN*). V dnešní době pracují moderní *end-to-end*[4] modely založené na neurnových sítích, určené pro rozpoznávání textu, na třech [16] základních principech:

- **CRNN (Convolutional Recurental Neural Network)** – spojení konvolučních a rekurentních sítí s využitím speciální chybové funkce [10] (*Connectionist Temporal Classification – CTC*)[5].
- **Encoder-Decoder** – modely, které využívají dvojici rekurentních sítí pro kódování sekvence a následné dekódování této sekvence.

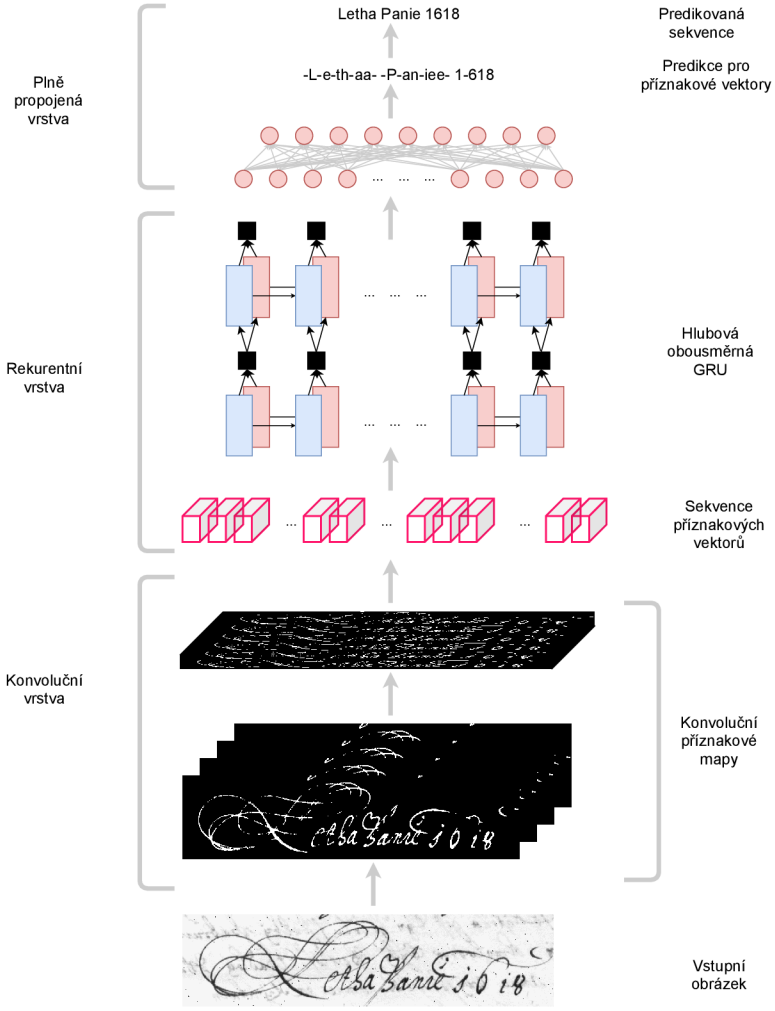
[4]Modely trénované jako jedna zřetězená linka.
[5]Tato chybová funkce bude vysvětlena později.

- **Encoder-Decoder + Attention** – model Kodér-Dekodér s využitím mechanismu Attention [33].

Na obrázku 1.12 je zobrazena ukázka architektury modelu Kodér-Dekodér, která byla použita v překladači Google. Pro implementaci vlastního OCR modelu byl vybrán první z těchto modelů, protože architektura tohoto modelu je velice intuitivní a často používaná v systémech pro rozpoznávání nejen ručně psaného písma – ve kterém jsou často psány matriční záznamy a historické prameny. Ve zbytku této sekce tedy bude popsána architektura modelu CRNN.

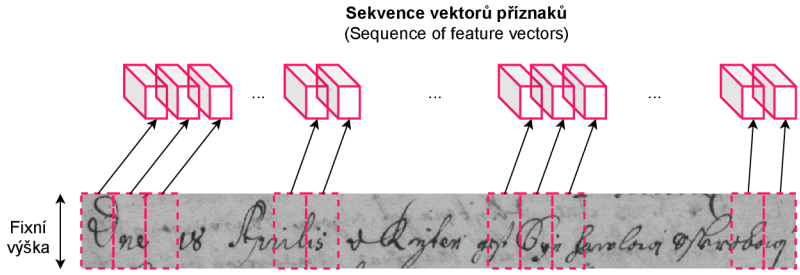
Systém pro rozpoznávání znaků založený na architektuře neuronové sítě CRNN

Model založený na spojení konvolučních a rekurentních neuronových sítí se nazývá *Convolutional Recurrental Neural Network, zkráceně CRNN [29]*. Celá architektura, která se skládá ze tří částí je zobrazena na obrázku 1.13. Model počítá se segmentovanými vstupními daty – vstupem tohoto modelu je obrázek řádku textu (segmentace úrovně 2), jehož výška je zmenšena na fixní hodnotu, např. 32 pixelů. Šířka je změněna proporcionálně.



Obrázek 1.13: Architektura neuronové sítě pro rozpoznávání znaků. Převzato a upraveno z [29]. Vstupní sken je z archivu MZA.

První částí architektury je konvoluční neuronová sít, která slouží k extrakci příznaků z celého vstupního obrázku. Architektura této konvoluční sítě může být libovolná, ale výstupem musí být sekvence příznakových vektorů (*sequence of feature vectors*), kde každý z těchto vektorů reprezentuje jeden sloupec[6] původního obrázku. Neuronová síť tedy musí redukovat výšku obrázku na hodnotu 1 pixel, ale zároveň musí také postupně zvýšit počet kanálů obrázku. Ukázka způsobu extrakce vektorů příznaků je zobrazena na obrázku 1.14.



Obrázek 1.14: Ukázka způsobu extrakce příznaků ze vstupního obrázku pomocí konvolučních sítí. Výška obrázku je redukována a vzniknou tak vektory příznaků reprezentující jednotlivé sloupce obrázku. Převzato a upraveno z [29]. Sken převzat z archivu MZA.

Po extrakci příznaků a získání sekvence příznakových vektorů tyto vektory vstupují do druhé části architektury modelu, kterou je rekurentní neuronová sít. Některé znaky mohou být širší a rozkládat se tak přes více příznakových vektorů – hlavním cílem této vrstvy je zachycení kontextu mezi jednotlivými příznakovými vektory, díky kterému dojde také ke zvýšení přesnosti rozpoznávání znaků celého modelu.

Poslední části této architektury je klasifikační vrstva, jejímž cílem je určit pravděpodobnostní rozložení tříd (znaků) pro jednotlivé vektory příznaků. V této části jsou využity plně propojené vrstvy neuronové sítě s aktivační funkcí ReLU [4], které jsou aplikovány na všechny vektory příznaků. Výstupem poslední plně propojené sítě je příznakový vektor s dimenzionalitou rovnou počtu znaků v predikované abecedě plus 1 (rezervovaný znak značící prázdný symbol) pro příznakový vektor, který nepředstavuje žádný znak textu. Na tyto příznakové vektory je následně aplikována aktivační funkce Softmax [6], pro zajištění validního rozložení pravděpodobnosti.

Chybová funkce pro učení neuronové sítě CRNN

Pro učení celé neuronové sítě je využita chybová funkce *Connectionist Temporal Classification – CTC [10, 23]*, která se využívá při klasifikaci sekvenčních dat, u kterých není známé zarovnání na výstupní třídy – například v oblasti rozpoznávání řečníka z nahrávky nebo rozpoznávání ručně psaného textu z obrázku [12]. V případě rozpoznávání ručně psaného písma z obrázku je pro snadnější získávání dat výhodné mít dataset v podobě dvojic – *obrázek řádku textu* a *vhodně kódovaný přepis textu z tohoto obrázku* – namísto anotování pozice a typu všech znaků ve vstupním obrázku. Takový dataset ale postrádá informaci o správném zarovnání jednotlivých přepsaných znaků a vstupního obrázku. Tento problém řeší chybová funkce CTC.

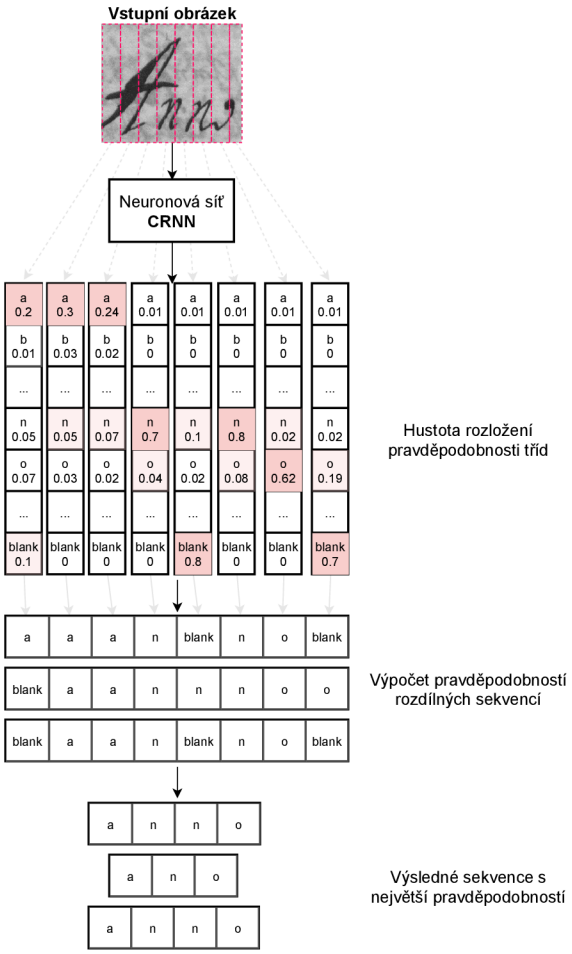
Cílem CTC chybové funkce je aby při trénování došlo k maximalizaci pravděpodobnosti přiřazení vstupní sekvenci S_{in} správnou výstupní sekvenci S_{out}, při vyhodnocení modelu je

[6]Šířka sloupce, která daný sloupec reprezentuje v původním obrázku závisí na úrovni redukce dimenzionality šířky vstupního obrázku

pak nejlepší výstupní sekvence S_{best} nalezena podle rovnice 1.5. Pravděpodobnost výstupní sekvence S_{out} za podmínky vstupní sekvence S_{in} je pak dána rovnicí 1.6, kde $A_{S_{in},S_{out}}$ je množina všech validních zarovnání vstupní sekvence S_{in} a výstupní sekvence S_{out} a $P(A)$ je pravděpodobnost jednoho zarovnání – marginalizace přes všechny validní zarovnaní vstupní a výstupní sekvence.

$$S_{best} = \underset{S_{out}}{\operatorname{argmax}}\, P(S_{out}|S_{in}) \tag{1.5}$$

$$P(S_{out}|S_{in}) = \sum_{A \in A_{S_{in},S_{out}}} P(A) \tag{1.6}$$



Obrázek 1.15: Ukázka fungování neuronové sítě CRNN při rozpoznávání textu „Anno“ z obrázku. Obrázek textu je vstupem neuronové sítě CRNN, která má na výstupu vektory reprezentující pravděpodobnostní rozložení jednotlivých znaků (tříd) pro daný sloupec obrázku. Červeně jsou vyznačeny znaky s velkou pravděpodobností. Vyznačené sloupce ve vstupním obrázku jsou pouze pro ilustraci, nejsou součástí tohoto obrázku. Obrázek převzat a upraven z [12].

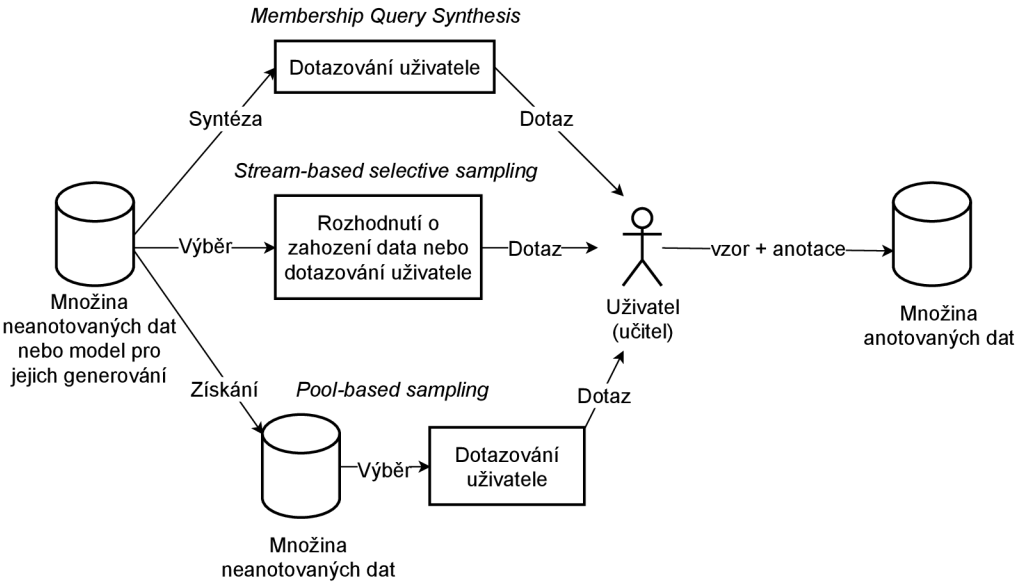
Ne každý vektor příznaků musí představovat znak ve výstupní sekvenci, některé sousední vektory mohou přestavovat stejný znak a někdy může být znak z výstupní frekvence

protažen přes více vektorů příznaků a může tak dojít k chybnému opakování téhož znaku. Z těchto důvodu byl zaveden speciální prázdný znak *blank* (třída). Pokud se znak *blank* nachází ve výstupní sekvenci mezi dvěma stejnými znaky, jsou tyto znaky brány jako dva stejné znaky nacházející se vedle sebe. Pokud ve výstupní sekvenci jsou dva stejné znaky, které nejsou odděleny znakem *blank* představují tyto znaky jeden znak, který byl ale protažen přes několik příznakových vektorů (duplicitní znaky jsou tedy smazány). Z výsledné sekvence jsou také odstraněný všechny znaky *blank*. Ukázka fungování neuronové sítě společně s chybovou funkcí CTC je zobrazena na obrázku 1.15.

1.2 Aktivní učení

Neuronové sítě a jiné modely, vyžadují pro své správné fungování velké množství anotovaných dat, ze kterých se následně učí jak řešit daný problém. V dnešní době je díky internetu k dispozici velké množství dat, která ale nejsou anotována. Samotná anotace je časově velmi náročná činnost. Z tohoto důvodu vznikly techniky, které usnadňují a hlavně urychlují získávání nových anotovaných dat. Tyto techniky se souhrnně nazývají Aktivní učení (*Active Learning – AL*) [7] (někdy také učení dotazováním – *Query Learning* [27]) a je jimi myšlena jakákoliv forma učení, při které má učící se program nějakou kontrolu nad daty, ze kterých se učí. Typicky také tyto programy komunikují s uživatelem, který například schvaluje správně anotovaná data. Aktivní učení je tak formou učení s učitelem (*Supervised learning*).

Vstupem programu pro aktivní učení je strojově učitelný model (*Machine Learning model*) a množina neanotovaných dat. Cílem tohoto programu je pomocí ML modelu a interakcí s uživatelem co nejefektivnějším způsobem převést množinu neanotovaných dat na množinu anotovaných dat, popř. co nejvíce zvýšit přesnost ML modelu za pomocí co nejméně anotovaných dat. Programy pro aktivní učení využívají různé strategie.



Obrázek 1.16: Obrázek zobrazující přehled variant aktivního učení. Obrázek převzat a upraven z [27].

1.2.1 Varianty aktivního učení

Rozlišujeme 3 varianty aktivního učení [27]: *Membership query synthesis*, *Pool-based sampling* a *Stream-based selective sampling*. Diagram zobrazující přehled těchto metod a jejich fungování je zobrazen na obrázku 1.16.

Nejčastější variantou je *pool-based* aktivní učení, při které je nejdříve získaná množina neanotovaných dat, následně AL program předkládá uživateli vzory z množiny neanotovaných dat, které je nutno anotovat. Pro výběr vzorků existuje mnoho strategií. Často používanou strategií je Nejisté vzorkování (*Uncertain Sampling – US [18]*), kdy program přednostně vybírá z množiny neanotovaných dat takové vzory, u kterých si je ML model nejméně jistý. U klasifikačních modelů je využíván výběr dat s největší informační entropií.

Syntéza dotazů o příslušnosti (*Membership query synthesis*) je varianta, kdy učící se program syntetizuje vlastní data na základě dostupné neanotované množiny dat a posílá uživateli dotazy o příslušnosti vzorků dat do dané třídy. Při učení klasifikačního modelu může například program vyříznout část některého z obrázků a ptát se uživatele, zda se v tomto výřezů nachází třída A nebo B.

Poslední rozlišovanou variantou je Selektivní výběr vzorků z toku dat (*Stream-based selective sampling*). Tato varianta je založena na předpokladu, že získání neanotovaných dat je zcela zdarma nebo levné (např. generovaní data z pravděpodobnostního rozložení). AL program nejdříve získá nový datový vzorek a následně podle toho, zda anotací daného data získá dostatečné množství nových informací, žádá uživatele o jeho anotaci.

Kapitola 2

Analýza požadavků a návrh řešení

V této kapitole je nejdříve detailněji popsán cíl této práce. Následně navržena architektura výsledného systému a popsány moduly, ze kterých se systém skládá, včetně komunikace mezi nimi. Dále je navržen způsob doučování modelů ze záznamů opravených uživateli. Poté je navrženo schéma databáze pro uložení dat o uživatelích webové aplikace, jejich nahraných dokumentech, anotacích těchto dokumentů, modelech atd. Konec kapitoly je věnován navrhnu způsobu segmentace řádků a textových regionů ve skenech historických dokumentů.

2.1 Cíl práce

Cílem této práce je navrhnout a vytvořit systém, který bude schopen používat různé implementace OCR (uživatelé si budou moci vybrat buď mezi již existujícím OCR nebo mnou vlastnoručně vytvořeným OCR). Dále pak navrhnout způsob, jak implementovat pravidelné doučování ze záznamů opravených uživateli u daných OCR. Systém musí také umožnovat full textové vyhledávání v rozpoznaných datech.

Aby bylo splněno výše uvedené, byl vybrán způsob realizace daného systému v podobně webové aplikace, kde si uživatelé budou moci nahrát své naskenované dokumenty a tyto dokumenty následně anotovat. Každý nahraný dokument vždy patří k některému datasetu. Uživatel tedy nejdříve vytvoří nový dataset, popř. zvolí již existující dataset a následně nahraje skeny svých dokumentů. Vlastník datasetu také může nastavit přístupnost tohoto datasetu (privátní/veřejný). Veřejné datasety pak vidí všichni uživatelé. Po nahrání skenů k datasetu lze tyto skeny anotovat. Anotací se rozumí vyznačení řádků v jednotlivých skenech, popř. zadání přepisu takto vyznačených řádků. Aby uživatelé nemuseli ručně vyznačovat všechny řádky, systém podporuje možnost automatické detekce řádků. Uživatelé mohou vyznačené řádky ručně přepsat/opravit a následně potvrdit jejich správnost.

Řádky s potvrzenými přepisy jsou automaticky zařazeny do trénovací datové sady, ze které se učí dostupné OCR modely. Aplikace nabízí dva druhy OCR modelů – Tesseract OCR vyvíjený firmou Google a mnou vlastnoručně vytvořený model CRNN OCR. Oba modely lze prostřednictvím aplikace trénovat i inferovat. Trénování OCR probíhá vždy nad uživateli potvrzenými přepisy řádků. Inference (přepis řádku pomocí OCR) probíhá vždy nad vyznačenými řádky, u kterých ale zatím není uživatelem potvrzena správnost přepisu. Uživatelé tak při inferenci nepřijdou o své doposud potvrzené přepisy řádků a mohou tak inferovat zvolený OCR model nad svými daty, opravit a popř. potvrdit správnost přepisů řádků přepsaných pomocí zvoleného OCR a následně znovu dotrénovat dané OCR nad

takto rozšířenou trénovací datovou sadou. Tímto způsobem mohou uživatelé opakovaně dotrénovávat dostupné OCR modely.

Pro každé OCR nabízí aplikace několik „globálních“ instancí daného OCR (každá instance má své vlastní trénovatelné parametry). Globální instance jsou předem připravené instance OCR, které nemohou uživatelé sami trénovat, mohou je využít pouze pro inferenci na dostupných datasetech nebo k vytváření nových instancí OCR. Aplikace umožňuje vytváření nových instancí OCR z již existujících instancí OCR modelů. Uživatelé také mohou zvolit přístupnost dané instance (veřejná/privátní).

Uživatelé mají možnost spuštění výše popsané detekce řádků, trénování a inference instancí OCR modelů pomocí zadání nového úkolu. Úkoly jsou vždy prováděny nad zvoleným datasetem. Při vytváření nového úkolu uživatel vybere dataset, typ úkolu (detekce řádků, trénování OCR nebo inference OCR), popř. model (Tesseract/CRNN OCR) a instanci daného modelu. Při trénování OCR lze také zadat počet epoch a rychlost učení (*learning-rate*). Vytvořený úkol je poté přidán do seznamu plánovaných úkolů. U každého datasetu lze také zobrazit seznam úkolu a jejich stav (dokončeno/plánováno/probíhá). Pro každý typ OCR (Tesseract/CRNN) je předem vybrána jedna globální instance, pro kterou je automaticky v nočních hodinách plánováno trénování a inference na všech potvrzených přepisech všech datasetů (mimo datasetů, které jsou vyřazeny z globálního trénování/inference). Vlastník datasetu může u svého datasetu vybrat možnost pro nezahrnutí daného datasetu do pravidelného globálního trénování/inference.

Mezi přepisy řádků lze vyhledávat. Slouží k tomu samostatná stránka, kde jsou zobrazeny přepisy řádků vlastních a veřejných datasetů. Uživatel může zvolit filtrační parametry jako jsou název datasetu; text, který musí přepisy obsahovat a příznak, zda se mají vyhledávat pouze uživateli potvrzené, nepotvrzené nebo všechny přepisy. Pro každý vyhledaný přepis řádku je kromě samotného přepisu vypsán dataset a název skenu, kde se nachází. Systém pro každý přepis také umožňuje přesměrování uživatele na konkrétní sken daného datasetu s vyznačením tohoto přepisu.

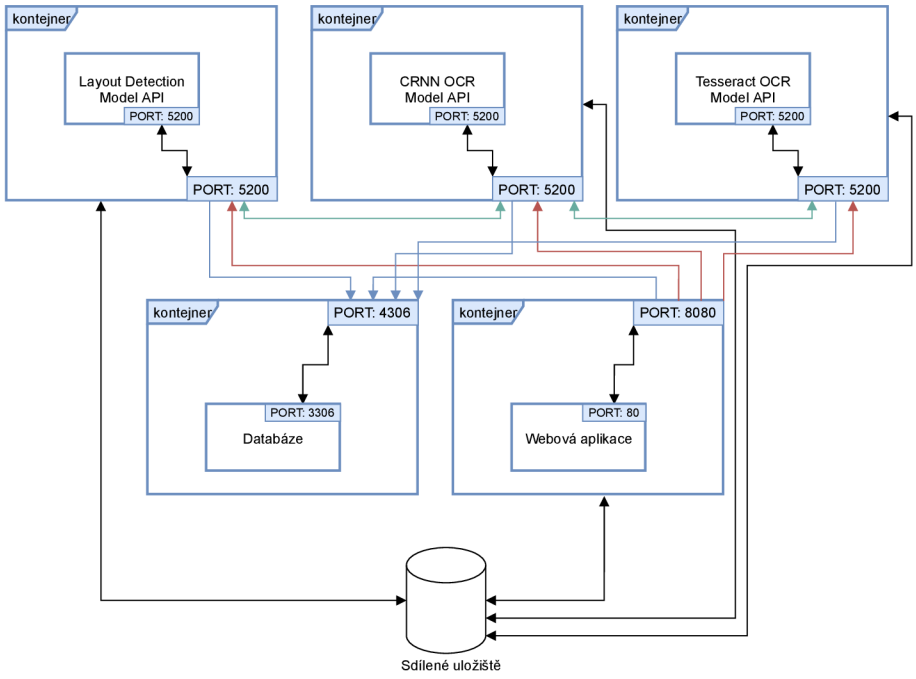
2.2 Návrh architektury systému

Celý systém je založen na architektuře mikroslužeb (*Microservices*), vytvořených pomocí nástroje Docker[1]. Jádro systému se skládá ze sdíleného uložiště souborů (využito pro uchování nahraných skenů, natrénovaných instancí OCR modelů, připravených datasetů...) a dvou služeb: **databáze** a **webová aplikace**. Tyto služby mezi sebou komunikují pomocí exponovaných portů. Veškerá data uživatelů, jako uživatelské účty, cesty k nahraným obrázkům, vytvořené datasety, anotace dat atd. budou uloženy v této centrální databázi. Webová aplikace, využívající webový server Nginx[2], umožnuje registraci a přihlášení uživatelů, nahrání obrázků a vytváření datasetů, anotaci nahraných obrázků, správu ostatních dat uživatele a plánování úkolů nad datasety.

K tomuto jádru aplikace jsou následně přidány další služby, implementující ML modely, které lze spouštět nad nahranými obrázky: **model pro detekci rozložení stránky** a **dva modely pro rozpoznávání znaků** (Tesseract a CRNN OCR) – systém bude využívat knihovnu Tesseract, která byla zmíněna v kapitole 1.1.3 a vlastní implementaci OCR založenou na architektuře CRNN, která byla popsána v kapitole 1.1.4. Pro každou z těchto služeb je pomocí webového mikro frameworku Flask[3] implementováno API HTTP rozhraní,

[1] https://www.docker.com
[2] https://www.nginx.com
[3] https://palletsprojects.com/p/flask

díky kterému je zajištěna komunikace s těmito službami. Uživatel tak může prostřednictvím webové aplikace odeslat na některou z těchto služeb HTTP požadavek, čímž dojde k aktivaci služby a zpracování plánovaných úkolů. Služba při vykonávání úkolu vyhledá v databázi cesty k obrázkům datasetu, nad kterým se má úkol provést a aplikuje příslušný ML model na tyto obrázky. Výstupy aplikace ML modelů nad obrázky jsou následně zapsány zpět do databáze, do příslušných databázových tabulek pro uchovávání anotací. Na obrázku 2.1 je zobrazena architektura výsledného systému, včetně pěti výše popsaných služeb, jejich portů a komunikace mezi nimi.



Obrázek 2.1: Ukázka architektury navrženého systému. Zelené šipky značí komunikaci mezi službami realizujícími ML modely. Červené šipky značí komunikaci webové aplikace se službami realizujícími ML modely a modré šipky značí komunikaci s databází.

2.3 Návrh způsobu doučování modelů z opravených záznamů

Výše uvedený systém umožňuje doučování OCR modelů z uživateli opravených anotací nahraných skenů dokumentů. Uživatel si vytvoří účet prostřednictvím webové aplikace a následně založí nový dataset obsahující nahrané skeny dokumentů nebo jiné obrázky s textem. Webová aplikace umožní uživateli ruční anotaci těchto obrázků (vyznačení řádků v obraze a zadání/opravu jejich přepisu) nebo automatickou detekci řádku v nahraných skenech. Po vyznačení, zadání a potvrzení přepisu některých z řádků v nahraných skenech daného datasetu vybere uživatel již existující instanci některého z OCR modelů nebo vytvoří instanci novou. Poté uživatel prostřednictvím webové aplikace naplánuje nový úkol pro trénování vybrané instance OCR modelu na doposud potvrzených přepisech řádků daného datasetu. Výslednou natrénovanou instanci modelu lze pak znovu inferovat nepotvrzené přepisy řádků daného datasetu. Tyto predikce modelu lze pak uživatelem znovu opravit, popřípadě potvrdit správnost jejich přepisu a rozšířit tak původní trénovací sadu. Po rozšíření trénovací

sady lze pak znovu dotrénovat tuto instanci OCR modelu a postup tak stále opakovat, dokud nebude daná instance modelu dostatečně přesná nebo dokud nebudou uživateli potvrzeny všechny vyznačené řádky. Tento princip doučování odpovídá Aktivnímu učení typu *pool-based sampling* – vytvořením nového datasetu, nahráním skenů dokumentů a spuštěním úkolu pro automatickou detekce řádků/regionů dojde k vytvoření pomyslné množiny neanotovaných dat (*pool*), která je graficky prezentována uživateli (dotazování) a ten z této množiny vybírá data a zadává jejich anotace (úprava bodů polygonu vyznačeného řádku resp. regionu a zadání textového přepisu).

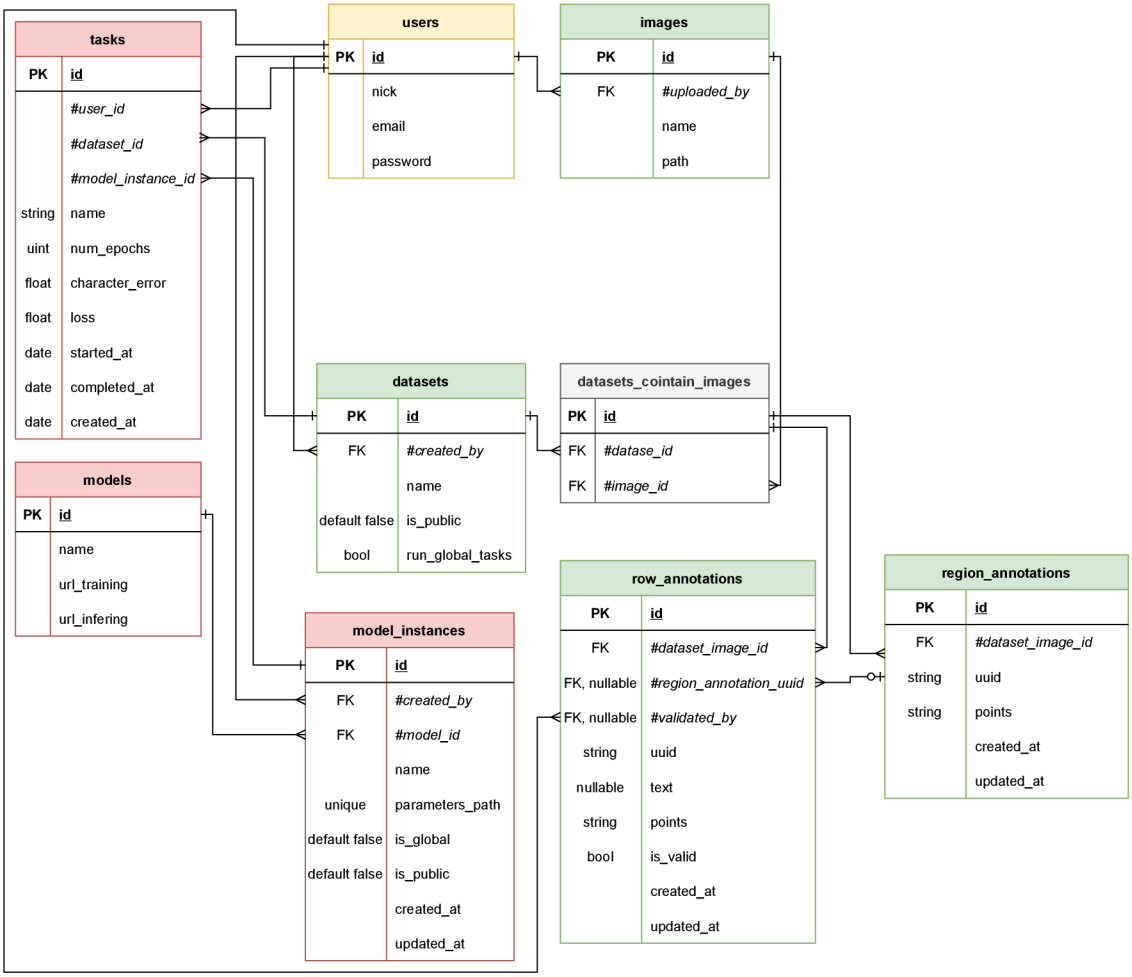
2.4 Návrh databáze

Pro uložení dat webové aplikace je použita relační databáze MySQL[4]. Na obrázku 2.2 je zobrazeno navržené schéma databáze. Schéma umožňuje ukládání informací o uživatelích, datasetech, nahraných obrázcích, anotacích obrázků patřících k jednotlivým datasetům, vytvořených uživatelských instancí ML modelů, dostupných ML modelech a plánovaných úkolech. Důležitými databázovými tabulkami jsou tabulky *row_annotations* a *region_annotations*, které obsahují informace o anotacích řádků a textových regionů jednotlivých obrázků v daném datasetu. Jeden záznam tabulky *row_annotations* představuje polygon s textovým přepisem obrázku překrývaným tímto polygonem. Body reprezentující vyznačené řádky a textové regiony jsou uloženy pomocí atributu *points*, rozpoznaný text je pak uložen pomocí atributu *text* (pouze u řádků). Příznak *is_valid* pak určuje, zda je daná anotace validní (lze ji použít pro trénování/testování OCR modelů). Tento příznak je automaticky nastaven u uživatelem ručně vytvořených anotací. V případě přepisů vytvořených některým z OCR modelů musí nejdříve tento přepis řádků ručně uživatel schválit, popř. ještě předtím upravit. Tímto způsobem je umožněno postupné doučování ML modelů. Každé anotaci řádků lze přiřadit textový region, ke kterému patří.

Tabulka *models* obsahuje záznamy o dostupných ML modelech, které lze aplikovat na nahrané dokumenty jednotlivých datasetů. Pomocí atributů *url_training* a *url_infering* je u každého modelu uchována *end-point* url adresa jednotlivých služeb, na kterou lze odeslat HTTP požadavek k aktivaci dané služby. Uživatelé mohou vytvářet vlastní instance dostupných ML modelů. Informace o těchto instancích jsou pak dostupné v tabulce *model_instances*. Jednotlivé instance se liší zejména unikátní cestou k souboru s uloženými parametry modelu, takže je možné vytvořit několik instancí stejného modelu. Pomocí atributu *is_public* může vlastník dané instance rozhodnout o dostupnosti této instance jiným uživatelům pro trénování respektive inference na svých datasetech. Atribut *is_global* značí zda je daná instance globální.

Tabulka *tasks* slouží pro ukládání úkolů prováděnými nad datasety. Každý vytvořený úkol má přiřazenou instanci ML modelu a dataset nad kterým se má vykonat. Pro trénování OCR je možné zadat počet epoch (*num_epochs*) a „rychlost učení“ (*learning rate*) Dále je dostupná informace o uživateli, který daný úkol vytvořil a časových značkách *started_at* a *completed_at*, díky kterým je možné sledovat které úkoly již byly spuštěny, resp. dokončeny. V případě dokončení úkolu trénování instance OCR modelu jsou připraveny atributy *loss* a *character_error* pro uložení zpětné vazby z trénování.

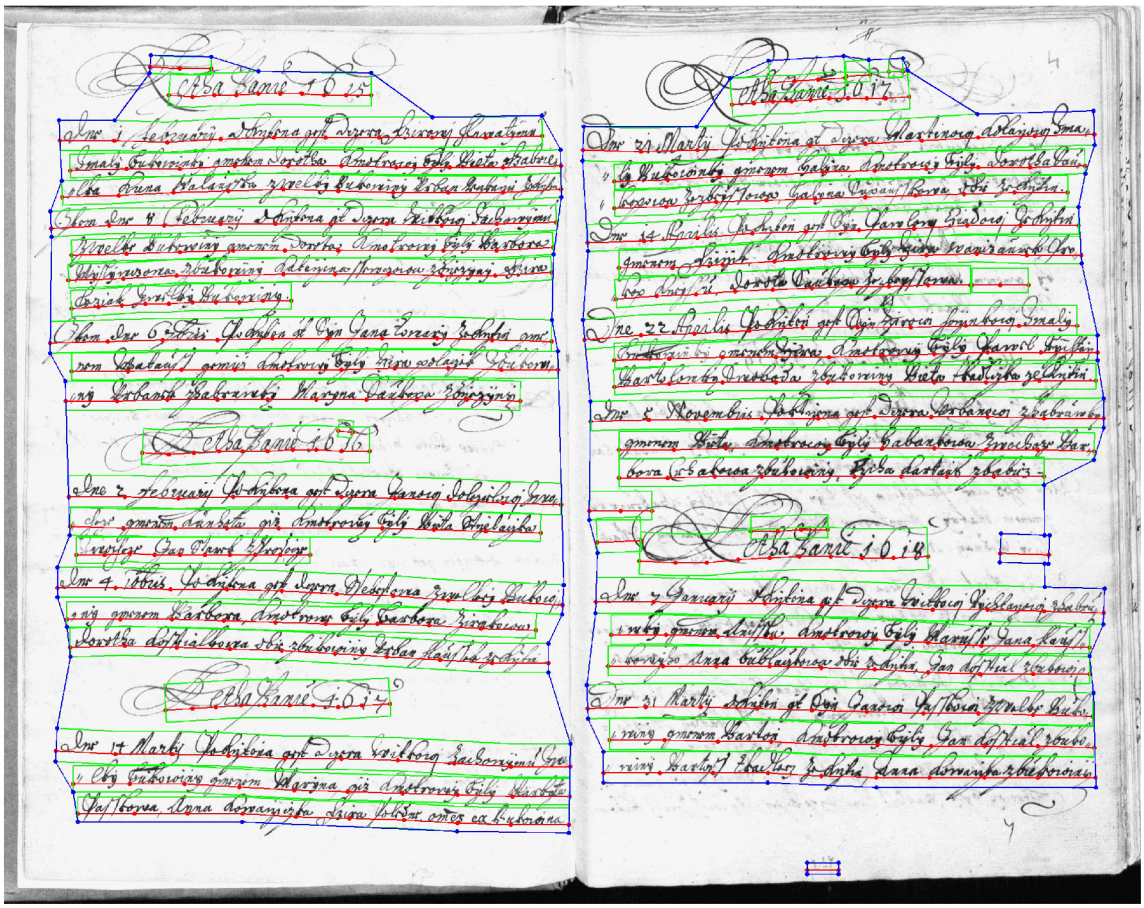
[4]https://www.mysql.com



Obrázek 2.2: Ukázka navrženého schéma databáze. Schéma je rozděleno do čtyř částí. Žlutě je vyznačena tabulka obsahující informace o uživatelích. Zeleně jsou vyznačeny tabulky obsahující informace o nahraných obrázcích, datasetech a anotacích. Červeně jsou vyznačeny tabulky obsahující informace o ML modelech, instancích těchto modelů a úkolů pro vykonávaní ML modelů nad obrázky datasetu. Šedě jsou pak vyznačeny vazební tabulky.

2.5 Návrh způsobu segmentace řádků ve skenech dokumentů

Pro detekci rozložení stránky se zaměřením na historické dokumenty a segmentaci řádků textu (segmentace úrovně 2 viz 1.1.2), popř. celých textových regionů bylo rozhodnuto využití neuronové sítě, která vznikla na FIT a je použitá v projektu PERO[5] a dostupná v rámci knihovny PERO-OCR[6]. Popis architektury této neuronové sítě a způsob jejího fungování je blíže uveden v článcích [11] a [17]. Druhý z těchto článku vznikl v rámci FIT. Ukázka detekce rozložení stránky a segmentace řádků skenu dokumentu pomocí této knihovny je zobrazena na obrázku 2.3.



Obrázek 2.3: Ukázka analýzy rozložení stránky a detekce jednotlivých řádků ve skenu historického dokumentu pomocí neuronové sítě, která je součástí knihovny PERO-OCR. Jedná se o sken matričního záznamu ze 17. století z archivu MZA. Pomocí modrého polygonu jsou ohraničeny textové regiony, následně pomocí zelených polygonů jsou vyznačeny jednotlivé textové řádky a pomocí červených čar jsou vyznačeny *baselines* daných textových řádků.

[5] https://pero-ocr.fit.vutbr.cz
[6] https://github.com/DCGM/pero-ocr

Kapitola 3

Implementace a testování

V této kapitole bude popsána implementace celého systému podle návrhu, který byl uveden v předešlé kapitole a následné testování tohoto sytému. V implementační části budou nejdříve popsány použité technologie a knihovny. Zbytek implementační části bude věnován popisu implementace jednotlivých modulů systému. Závěr kapitoly je věnován testování výsledného systému.

3.1 Použité technologie a nástroje

Jak již bylo popsáno v předešlé kapitole „Analýza požadavků a návrh řešení“, celý systém je založen na architektuře mikroslužeb a dělí se na 5 samostatných modulů, které mezi sebou komunikují pomocí exponovaných portů. **Relační databáze** je typu MySQL[1], **Webová aplikace** je napsaná v jazyce PHP (Laravel Framework[2]) a využívá front-end Framework Vue.js[3] pro zajištění reaktivity. Jako HTML framework byl využit framework Bootstrap[4], pro pre-processing CSS je využit framework Sass[5]. Pro grafické ilustrace k doplnění návodu aplikace je využita *open-source* knihovna unDraw[6]. Pro základní kostru stránky byla využita *open-source* HTML šablona „StartBootstrap SB Admin 2“[7]. Grafická komponenta pro zobrazení a anotaci skenů, která je součástí webové aplikace, využívá knihovnu Paper.js[8]. Nahrávání obrázku je vyřešeno pomocí knihovny Dropzone[9]. **Detekce řádků a textových regionů** je realizována pomocí neuronové sítě, která je vyvíjená v rámci Fakulty informačních technologií VUT v Brně pro projekt PERO OCR[10] (po domluvě bylo dovoleno využití sítě v rámci této diplomové práce). Jako již existující OCR systém byl využit systém **Tesseract OCR**[11]. Vlastnoručně vytvořený model **CRNN OCR** byl implementován v jazyce Python 3 za pomocí knihovny PyTorch[12]. API pro komunikaci s kontejnery realizujícími

[1] https://www.mysql.com
[2] https://laravel.com
[3] https://vuejs.org/
[4] https://getbootstrap.com
[5] https://sass-lang.com
[6] https://undraw.co
[7] https://github.com/StartBootstrap/startbootstrap-sb-admin-2
[8] http://paperjs.org
[9] https://www.dropzonejs.com
[10] https://github.com/DCGM/pero-ocr, https://pero-ocr.fit.vutbr.cz/
[11] https://github.com/tesseract-ocr/tesseract, https://github.com/tesseract-ocr/tesstrain
[12] https://pytorch.org/

úkoly nad datasety bylo vytvořeno v jazyce Python 3 (framework Flask[13]). Kostra HTTP API byla inspirována repozitářem[14], který byl využíván pro práci na projektu DEMoS[15]. Pro zjištění sklonu textu na obrázku byla využita Python 3 knihovna Deskew[16].

3.2 Moduly systému

V této části bude detailně popsána implementace jednotlivých modulů systému. Pro každý modul byl vytvořen pomocí nástroje Docker[17] kontejner, ve kterém daná služba izolovaně běží a který zajišťuje veškeré softwarové závislosti potřebné pro běh dané služby. Byly vytvořeny kontejnery s názvy: `tesseract_ocr`, `crnn_ocr`, `pero_ocr`[18], `php`, `mysql`, `nginx`. První tři kontejnery realizují úkoly vykonávané nad datasety, poslední tří kontejnery zajišťují chod webové aplikace. Kód popisující vytváření obrazů a kontejnerů je umístěn ve složce `/docker`. Pro každou službu je zde vytvořen adresář obsahující soubor `Dockerfile` popisující softwarové závislosti, které mají být nainstalovány do obrazu dané služby[19]. Ve složce `/docker` je také umístěn soubor `docker-compose.yml`, který popisuje, jakým způsobem budou vytvořeny jednotlivé kontejnery z předem připravených obrazů a soubor `Makefile`, který nabízí direktivy pro vytvoření obrazů a kontejnerů, stažení předtrénovaných modelů pro Tesseract OCR a neuronové sítě k detekci rozložení stránky atd. Do všech kontejnerů kromě databázového serveru je namontovaná složka `/storage`, která tak představuje sdílené uložiště mezi službami. Sdílené uložiště `/storage` obsahuje podsložku `images` (uživateli nahrané obrázky), a složky `tesseract_ocr` a `crnn_ocr`. Tyto složky slouží k uložení parametrů těchto modelů a dočasných souborů, nutných k jejich trénování. Jednotlivé služby si dále montují složky se svými zdrojovými soubory jako např. `/app` (webová aplikace), `/ocr` (vlastní CRNN OCR model), `/modules_api` (API pro komunikaci se službami realizujícími ML modely) atd.

3.2.1 API pro kontejnery vykonávajícími úkoly nad datasety

Aby bylo možné komunikovat s vytvořenými kontejnery, které realizují úkoly vykonávané nad datasety (`tesseract_ocr`, `crnn_ocr`, `pero_ocr`), bylo vytvořeno jednotné REST API rozhraní pomocí programovacího jazyka Python 3 a knihovny Flask. Zdrojové kódy tohoto API jsou uloženy v adresáři `/modules_api`. Veškeré end-point url jsou definovány v souboru `/modules_api/modules_api/apis/namespace1.py`. Celkem je zde definováno 6 end-point url adres, které aktivují provádění úkolů plánovaných na jednotlivých datasetech. Výše zmíněné tři kontejnery si při spuštění načtou zdrojové soubory tohoto API. Každý kontejner ale podporuje pouze end-point adresy, které jsou mu určené.

[13] https://flask.palletsprojects.com

[14] https://github.com/DavidHribek/demos-segmentation forknuto z https://github.com/isvoboda/demos-segmentation

[15] http://perun.fit.vutbr.cz/http://perun.fit.vutbr.cz

[16] https://pypi.org/project/deskew/

[17] https://www.docker.com

[18] Kontejner pro detekci rozložení stránky se jmenuje stejně jako projekt v rámci kterého je detekce vyvíjená. Součástí tohoto projektu je i OCR, které ale nebylo využito.

[19] Soubory Dockerfile jsou založeny na oficiálních kontejnerech dostupných na webu https://hub.docker.com.

Plánování vykonávání úkolů nad datasety

Pro plánování a vykonání úkolů nad datasety byl vytvořen následující protokol komunikace mezi jednotlivými komponentami systému. Uživatel webové aplikace vybere ze seznamu datasetů dataset, nad kterým chce provést jeden z možných úkolů, následně vybere typ úkolu, popř. další parametry, které aplikace nabízí pro daný úkol (lze vybrat úkol pro detekci rozložen stránky – vyznačení řádků a textových regionů a úkoly pro trénování/inferenci Tesseract a CRNN OCR modelů). Po vyplněný úkolu a odeslání požadavku na webový server dojde k vytvoření nového záznamu reprezentujícího daný úkol v databázi (v databázové tabulce `tasks`). Následně je zaslán požadavek pro „dispečink“ daného úkolu. Dispečink úkolů má (krom jiného) na starosti služba `crnn_ocr`. Webová aplikace tedy zašle asynchronně HTTP POST požadavek bez jakýchkoliv parametrů na adresu `crnn_ocr:5200/ocr_annotator_api/dispatch_task`.

Po přijetí požadavku služba `crnn_ocr` zjistí z databáze seznam úkolů. Pokud je některý z úkolu právě vykonáván (atribut úkolu `started_at` je nenulový a atribut `completed_at` je nulový), vykonání nového úkolu není zahájeno a proces dispečinku je ukončen. Pokud není aktuálně žádný úkol vykonáván, je z fronty čekajících úkolů (atribut `started_at` je nulový) vybrán nejstarší úkol (řazeno podle atributu `created_at`), který bude následně odeslán k vykonání. Služba nastaví v databázi u tohoto úkolu čas počátku vykonávání (atribut `started_at`) na aktuální čas a následně odešle asynchronně HTTP POST požadavek bez parametrů na jednu z url end-point adres realizující daný úkol (viz níže). Adresa na kterou se má požadavek poslat je zjištěna na základě typu úkolu (atribut `task_name`), instance modelu, která má daný úkol vykonat (atribut `model_instance_id`) a modelu, ke kterému tato instance patří. Databázový záznam modelu obsahuje atributy `url_training` a `url_infering` s adresami pro vykonání konkrétních úkolů s daným modelem.

V následujícím seznamu jsou uvedeny všechny API end-point url adresy, které podporují jednotlivé kontejnery.

Seznam end-point url adres realizujících úkoly nad datasety:

- `crnn_ocr:5200/ocr_annotator_api/dispatch_task`: Dispečink úkolů.
- `crnn_ocr:5200/ocr_annotator_api/run_crnn_ocr_infering`: Inference CRNN OCR.
- `crnn_ocr:5200/ocr_annotator_api/run_crnn_ocr_training`: Trénování CRNN OCR.
- `tesseract_ocr:5200/ocr_annotator_api/run_tesseract_ocr_infering`: Inference Tesseract OCR.
- `tesseract_ocr:5200/ocr_annotator_api/run_tesseract_ocr_training`: Trénování Tesseract OCR.
- `pero_ocr:5200/ocr_annotator_api/run_layout_analysis`: Detekce rozložení stránky (nalezení řádků a textových regionů).

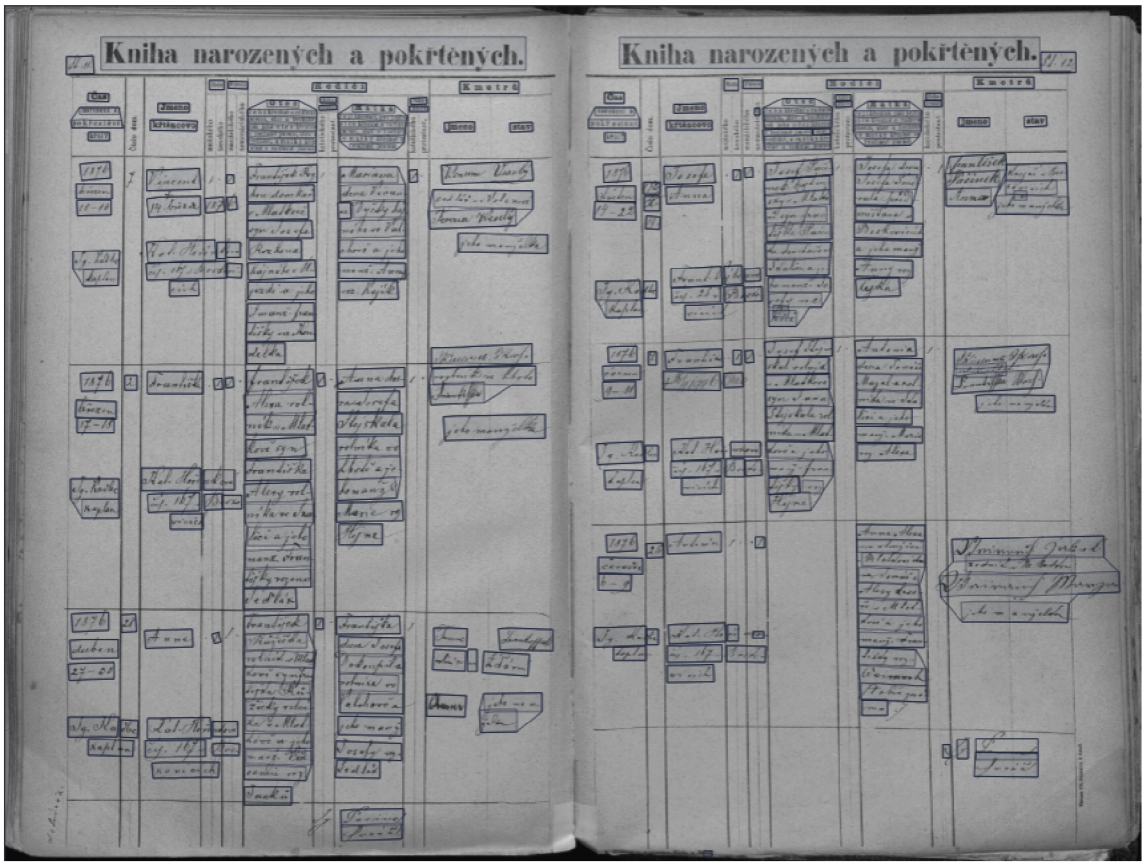
Zjištění úkolu pro vykonání Po přijetí HTTP POST požadavku některou z těchto metod si daná služba z databáze zjistí, zda existuje úkol, který by měla vykonat – vyfiltruje si z databáze seznam spuštěných nedokončených úkolů (atribut `started_at` není nulový, atribut `completed_at` je nulový), které jsou pro ni určené (atributu `task_name`). Takový úkol by měl existovat právě jeden. Pokud takový úkol neexistuje, služba ukončí proces vykonání. Pokud daný úkol existuje, služba provede obsluhu jeho vykonání.

Dokončení úkolu a dispečink následujícího úkolu Po vykonání daného úkolu služba nastaví v databázi čas dokončení (atribut `completed_at`) na aktuální hodnotu, čímž je úkol dokončen (realizováno pomocí metody `set_task_completed(task_id)` třídy `DB` umístěné v souboru `/modules_api/modules_api/db.py`). Následně zašle asynchronně HTTP POST požadavek bez parametrů na adresu `crnn_ocr:5200/ocr_annotator_api/dispatch_task`, čímž navrátí zpět řízení dispečinku úkolů, který buďto započne dispečink pro další plánovaný úkol nebo se ukončí. Kódy pro vykonání jednotlivých úkolů jsou uloženy v adresáři `/modules_api/modules_api/modules`.

V následujících sekcích bude popsána implementace služeb realizujících jednotlivé úkoly nad datasety.

3.2.2 Automatická detekce řádků a textových regionů v obraze

Tato služba zajišťuje automatickou detekci řádků a textových regionů nad obrázky zvoleného datasetu. Jádrem služby je neuronová síť, která umožňuje pro zadaný sken získat polygony určující řádky, textové regiony a příslušnost řádků k jednotlivým regionům.



Obrázek 3.1: Ukázka detekce rozložení stránky pro knihu narozených a pokřtěných z archivu MZA. Detekované anotace jsou zobrazeny pomocí anotační komponenty webové aplikace. Jednotlivé řádky jsou ohraničeny a vybarveny tmavě modrou barvou s průhledností. Řádky jsou ohraničeny textovými regiony.

Instalace

Při vytváření obrazu jsou kromě instalace závislostí také z GitHub repozitáře[20] projektu PERO OCR staženy zdrojové soubory pro detekci. Následně jsou pomocí direktivy `download-pero-config` výše zmíněného Makefile staženy a uloženy do adresáře `/docker/pero_ocr/pero_config` konfigurační parametry a parametry modelu, které jsou potřebné pro běh sítě. Při vytváření kontejneru jsou pak tyto konfigurační parametry namontovány spolu s ostatními soubory.

Detekce řádků a textových regionů

Po přijetí HTTP POST požadavku zaslaného na adresu `pero_ocr:5200/ocr_annotator_api/run_layout_analysis` dojde k aktivaci této služby. Služba zjistí úkol, který čeká na její vykonání (viz 3.2.1) a pokud takový existuje, začne tento úkol vykonávat. Vykonání úkolu zajišťuje metoda `execute_detect_layout_tasks` třídy `LayoutDetection`, která je umístěna v souboru `/modules_api/modules_api/modules/layout_detection.py`.

Vykonání úkolu Pomocí metody `get_task_images(task_id)` výše zmíněné třídy `DB` jsou z databáze zjištěny informace o obrázcích, které mají být detekovány (úkol obsahuje identifikátor konkrétního datasetu). Před spuštěním detekce jsou u každého obrázku daného datasetu odstraněny z databáze všechny „staré“ anotace, které nebyly uživateli potvrzeny jako správné. Následně jsou ze složky `/storage/images/{dataset_id}` ve sdíleném uložišti načteny jednotlivé obrázky a je pro něj spuštěna detekce rozložení stránky. Výsledné polygony reprezentující řádky a textové regiony jsou nahrány do databáze k danému obrázku datasetu (k tomu jsou využity databázové tabulky `row_annotations` a `region_annotations`). U každé nové anotace je před vložením do databáze zároveň zkontrolováno, jestli již existuje anotace se stejnými body, tímto je zajištěno, že nebudou vkládány duplicitní anotace). Pro každou novou anotaci je zároveň vygenerován jednoznačný identifikátor `uuid`[21], který později slouží k manipulaci s anotacemi prostřednictvím GUI webové aplikace. Řádky mají navíc nastaven atribut `region_annotations_uuid` na hodnotu `uuid` regionu, ke kterému patří. V případě, že je tento atribut nulový, nepatří daný řádek k žádnému regionu.

Po vykonání úkolu je zapsán čas dokončení tohoto úkolu a předáno řízení dispečinku úkolů (viz 3.2.1).

3.2.3 Tesseract OCR

Tato služba zajišťuje trénování a inferenci Tesseract OCR nad daty zvoleného datasetu. Trénování probíhá vždy nad uživateli potvrzenými přepisy řádků ve skenech zvoleného datasetu. Inference naopak probíhá vždy nad nepotvrzenými přepisy řádků ve skenech zvoleného datasetu.

Instalace

Při vytváření obrazu jsou kromě instalace závislostí také staženy z GitHub repozitářů[22] zdrojové soubory pro trénování a inferenci Tesseract OCR. Následně jsou pomocí direk-

[20] https://github.com/DavidHribek/pero-ocr forknuto z https://github.com/DCGM/pero-ocr
[21] https://en.wikipedia.org/wiki/Universally_unique_identifier
[22] https://github.com/tesseract-ocr/tesseract, https://github.com/tesseract-ocr/tesstrain.git

tivy `download-tesseract-models` výše zmíněného Makefile staženy a uloženy do adresáře `/store/tesseract_ocr/tessdata` předtrénované OCR modely a konfigurační soubory, které jsou potřebné pro běh Tesseract OCR. Díky tomu, že jsou soubory uloženy ve sdíleném uložišti, má k nim kontejner přístup.

Trénování Tesseract OCR

Po přijetí HTTP POST požadavku zaslaného na adresu `tesseract_ocr:5200/ocr_annotator_api/run_tesseract_ocr_ocr_training` dojde k aktivaci této služby. Služba zjistí úkol, který čeká na její vykonání (viz 3.2.1) a pokud takový existuje, začne tento úkol vykonávat. Vykonání úkolu zajišťuje metoda `execute_ocr_training_tasks` třídy `OcrTraining`, která je umístěna v souboru `/modules_api/modules_api/modules/tesseract_ocr_training.py`.

Vykonání úkolu Trénování Tesseract OCR vyžaduje umístění trénovacích dat do samostatné složky a následné spuštění trénování pomocí předpřipraveného příkazu `make` s parametry. Data určená pro trénování OCR jsou dvojice souborů: vyřezaný řádek textu a textový soubor stejného jména a přidanou příponou „.gt.txt“ s přepisem tohoto řádku. K tomuto účelu je určena složka `/storage/tesseract_ocr/gt_train_tmp`, která je umístěna ve sdíleném uložišti.

Obsah složky `/storage/tesseract_ocr/gt_train_tmp`, je před začátkem každého trénování nejdříve smazán, protože může obsahovat dočasné soubory z předešlého trénování. Následně jsou pomocí metody `get_task_images(task_id)` třídy `DB` z databáze zjištěny informace o obrázcích, na kterých se má Tesseract OCR trénovat (úkol obsahuje identifikátor konkrétního datasetu). Pro každý obrázek patřící do daného datasetu jsou metodou `get_image_row_annotations_train` třídy `DB` načteny informace o anotacích řádků, které jsou určené pro trénování OCR (atribut `is_valid`, který značí, zda uživatelé přepis tohoto řádku potvrdili, je pravdivý a attribut `text` je nenulový). Každá načtená anotace řádku obsahuje v atributu `points` – body polygonu vyznačujícího tento řádek. Z tohoto polygonu je vypočten bounding box, který je vyřezán z daného obrázku (uloženém ve sdíleném uložišti `/storage/images/{dataset_id}`), následně je z tohoto bounding-boxu ještě vyřezán samotný polygon, který je poté otočen tak, aby byl text na obrázku vodorovně (viz sekce 1.1.2). Anotace také obsahuje textový přepis řádku v atributu `text`. Vyřezaný obrázek je spolu s jeho textovým přepisem uložen do složky `/storage/tesseract_ocr/gt_train_tmp` v podobně dvou soborů: `{image_name}.{extension}` a `{image_name}.gt.txt`.

Po tom, co jsou takto připravena všechna data pro trénování je pomocí příkazu `make training` spuštěno trénování Tesseract OCR. Jako parametr trénování je nastavena složka s daty pro trénování `/storage/tesseract_ocr/gt_train_tmp`, maximální počet epoch (atribut `num_epochs`) a instance modelu, která se má použít pro trénování. Pro výběr správné instance modelu k trénování se v databázi vyhledá záznam o instanci modelu podle identifikátoru, který je uložen v atributu `model_instance_id`. Součástí tohoto záznamu je atribut `parameters_path`, který obsahuje relativní cestu k souboru s parametry daného modelu. V případě Tesseract OCR je tato cesta relativní ke složce `/storage/tesseract_ocr/tessdata`, kde jsou uloženy všechny natrénované modely.

Po vykonání úkolu je zapsán čas dokončení tohoto úkolu a předáno řízení dispečinku úkolů (viz 3.2.1).

Inference Tesseract OCR

Po přijetí HTTP POST požadavku zaslaného na adresu `tesseract_ocr:5200/ocr_annotator_api/run_tesseract_ocr_infering` dojde k aktivaci této služby. Služba zjistí úkol, který čeká na její vykonání (viz 3.2.1) a pokud takový existuje, začne tento úkol vykonávat. Vykonání úkolu zajišťuje metoda `execute_ocr_infering_tasks` třídy `OcrInfering`, která je umístěna v souboru `/modules_api/modules_api/modules/tesseract_ocr_infering.py`.

Vykonání úkolu Pomocí metody `get_task_images(task_id)` třídy `DB` jsou z databáze zjištěny informace o obrázcích, na kterých se má Tesseract OCR inferovat (úkol obsahuje identifikátor konkrétního datasetu). Pro každý obrázek patřící do daného datasetu jsou metodou `get_image_row_annotations_infer` třídy `DB` načteny informace o anotacích řádků, které jsou určeny pro inferování OCR (atribut `is_valid`, který značí, zda uživatelé přepis tohoto řádku potvrdili, je nepravdivý nebo nulový). Každá načtená anotace řádku obsahuje v atributu `points` body polygonu vyznačujícího tento řádek. Z tohoto polygonu je opět vyřezán bounding-box a polygon, který je otočen tak, aby byl text na obrázku vodorovně (obrázek je uložen ve sdíleném uložišti `/storage/images/{dataset_id}`). U takto vyřezaných řádků je zjištěn přepis predikovaný systémem Tessract OCR pomocí funkce `image_to_string` modulu `pytesseract`. Této funkci je předán vyřezaný obrázek (v podobě numpy pole) a název uživatelem vybrané instance modelu (zjištěná stejně jako při trénování). Výsledný přepis je pro každý výřez zapsán zpět do databáze k příslušné anotaci řádku do atributu `text` (zajištěno metodou `update_row_annotation_text` třídy `DB`).

Po vykonání úkolu je zapsán čas dokončení tohoto úkolu a předáno řízení dispečinku úkolů (viz 3.2.1).

3.2.4 Vlastnoručně vytvořený OCR model

V této sekci bude popsána implementace vlastního OCR modelu. Následně budou popsány dvě služby, které tento vytvořený model využívají: služba pro trénování a služba pro inferenci tohoto OCR modelu. Pro vlastnoručně vytvořený OCR model byla zvolena architektura neuronové sítě CRNN (*Convolutional Recurrental Neural Network*), která byla popsána v kapitole „Analýza a návrh řešení“ v sekci 1.1.4. Tato architektura byla zvolena, protože je často používaná pro rozpoznávání textu pomocí neuronových sítí a umožňuje také rozponávání ručně psaného textu – ve kterém jsou často psány matriční záznamy a jiné historické prameny. Veškerý zdrojový kód CRNN OCR je umístěn v adresáři `/ocr`. Tento adresář obsahuje dále složku `src` s implementací CRNN OCR a složku `scripts` se skripty pro demonstraci trénování/inference sítě. Neuronová síť je implementována pomocí knihovny PyTorch.

Struktura zdrojového kódu a dělení na komponenty Celá implementace CRNN OCR se skládá ze 4 základních komponent:

- **Model**: Třída modelu definuje topologii neuronové sítě CRNN. Všechny zdrojové kódy související s implementací modelu jsou uloženy v adresáři `src/model`. Komponenta model je vždy instancí třídy `CRNN`, která je umístěna v souboru `model/crnn.py`. Třída `CRNN` je dále zděděna ze třídy `AbstractModel`, která je umístěna v souboru `model/abstract_model.py`. Třída `CRNN` definuje strukturu celé CRNN sítě – při inferenci modelu této třídy je nejdříve aplikována konvoluční neuronová síť, poté rekurentní neuronová síť GRU [30], následně plně propojené sítě s aktivační funkcí

ReLU[23] a finální aktivační funkcí Softmax[24]. Při instanciaci objektu třídy `CRNN` jsou v konstruktoru předány konkrétní parametry sítě: parametr `feature_size` určuje velikost skrytých vektorů rekurentní sítě, parametr `vocalubary_size` určuje velikost výstupní abecedy – na základě tohoto parametru je nastavena velikost výstupní plně propojené vrstvy (jedná se o počet znaků/tříd, které je síť schopna klasifikovat), příznak `rnn_bidirectional` určuje, zda má být použita obousměrná rekurentní síť, parametr `rnn_layers` určuje počet vrstev rekurentní neuronové sítě a parametr `parameters_path` určuje cestu pro načtení/uložení učitelných parametrů modelu. Posledním parametrem je `cnn`, kterým je předána instance konkrétní konvoluční sítě, která se použije pro extrakci příznaků z obrazu.

Konvoluční sítě V souboru `model/backbones.py` jsou umístěny jednotlivé konvoluční sítě, které byly implementovány. Všechny sítě jsou založeny na architektuře VGG[25]. Hlavním blokem sítí je vrstva `ConvBnReluMax`, která realizuje operace konvoluce, batch-normalizace, max-pooling a aplikace nelineární funkce ReLU, které se v těchto sítích často opakují. Tato vrstva je implementována třídou `ConvBnReluMax`, která je umístěna v souboru `model/backbones.py`. Při instanciaci objektu této třídy jsou předány parametry `in_channels` – počet vstupních kanálů konvoluční vrstvy, `out_channels` – počet výstupních kanálů konvoluční vrstvy, `kernel_size` – velikost konvolučního jádra, `padding` – velikost zarovnání, `stride` – velikost posunu konvolučního jádra, příznak `batch_norm` určující zda má být využita batch-normalizace a parametr `max_pool` – určující zda a s jakým jádrem má být použit max pooling pro redukci dimenzionality. S využitím třídy `ConvBnReluMax` jsou pak dále vytvořeny konvoluční sítě implementované třídami `CNN` a `CNN2`, které jsou rovněž umístěny v souboru `models/backbones.py`. Obě tyto třídy přijímají jeden parametr `feature_size` určující počet výstupních kanálů (shodné s počtem vstupních kanálů následující rekurentní sítě) a jsou složeny ze sekvence vrstev `ConvBnReluMax`. Počty vstupních a výstupních kanálů těchto vrstev jsou u obou sítí zvoleny tak, aby na sebe navazovaly, a aby se postupně počet výstupních kanálů zvyšoval. Počet výstupních kanálů poslední vrstvy je vždy nastaven na hodnotu `feature_size` pro zachování kompatibility s následující rekurentní sítí. Obě sítě předpokládají jeden vstupní kanál. Bloky sítě `CNN` pak mají počty výstupních kanálů nataveny na 64, 64, 128, 128, 256, 256, 512, `feature_size` a bloky sítě `CNN2` mají počty výstupních kanálů nastaveny na 64, 64, 128, 128, 256, 256, 512, 512, 1024, 1024, 1024, `feature_size`. U obou sítí jsou jednotlivé bloky prokládány vrstvami `Dropout`[26] pro regularizaci a redukci přetrénování. Některé bloky `ConvBnReluMax` mají nastavený příznak pro použití batch-normalizace, stejně tak některé tyto bloky mají nastaveny parametr `max_pool` s velikostmi jádra tak, aby byl ve finále předpokládaný vstupní tensor s dimenzí výšky 32 redukován na dimenzi 1. Tímto je zajištěno, že předpokládaný vstupní tensor/obrázek řádku s dimenzí (batch, počet kanálu = 1, výška, šířka) je převeden na tensor s dimenzí (batch, počet kanálů = `feature_size`, výška = 1, redukovaná šířka).

- **Dataset**: Dataset je třída, která zapouzdřuje práci s daty, na kterých se bude trénovat model, popř. testovat již natrénovaný model. Tato třída umožňuje načtení dat data-

[23] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)
[24] https://en.wikipedia.org/wiki/Softmax_function
[25] https://medium.com/analytics-vidhya/vggnet-architecture-explained-e5c7318aa5b6
[26] https://en.wikipedia.org/wiki/Dilution_(neural_networks)

setu z disku, převedení do interní reprezentace, rozdělení dat na trénovací a testovací sadu a uložení dat zpět na disk v binární podobě, která se dá rychle načíst zpět do paměti.

Jednotlivé datasety pro OCR mohou být dostupné v odlišných formátech. Některé datasety obsahují složku s vyřezanými obrázky řádků a textový soubor se seznamem dvojic: cesta k obrázku a přepis textu na tomto obrázku. Jiné datasety mohou mít přepisy obrázků v jiných formátech, např. CSV, XML atd. Existují i datasety, které neposkytují výřezy řádků, ale rovnou celé skeny dokumentů. Jednotlivé anotace tak obsahují nejen cestu k obrázku a přepis, ale také body určující pozici přepsaného textu (např. polygon) v daném obrázku. Aby byl zajištěn jednotný přístup ke všem těmto způsobům reprezentace OCR datasetu, každý nově přidaný dataset je reprezentován třídou, která je zděděna z abstraktní třídy `AbstractDataset`, umístěné v souboru `dataset/abstract_dataset.py`. Všechny zdrojové kódy související s implementací datasetů jsou uloženy v adresáři `src/dataset`. Třída `AbstractDataset` přijímá v konstruktoru cestu k již předpřipraveným datasetům `prepared_datasets_dir_path` a poměr pro dělení datasetu na trénovací a testovací sadu `train_test_ratio` (v případě chybějící testovací sady). Dálé třída definuje 4 instanční proměnné typu seznam pro uložení dat datasetu: (`train_x` – trénovací vstupy, `train_y` – trénovací výstupy, `test_x` – testovací vstupy, `test_y` – testovací výstupy).

Metody třídy `AbstractDataset`

- `__set_mode__(mode)`: Metoda pro přepínání datasetu mezi trénovacím a testovacím módem. Objekt třídy dataset je iterovatelný. Podle aktivovaného módu jsou při iteraci instance této třídy vráceny buďto trénovací nebo testovací data.
- `__getitem__(index)`: Metoda zajišťuje přístup k prvkům dataset pomocí indexu. V testovacím módu je indexována trénovací množina, v testovacím módu je indexována testovací množina. Metoda vrací dvojici (trénovací vzor, testovací vzor) z dané množiny dat.
- `__len__()`: Metoda vrací počet položek v trénovací respektive testovací sadě (podle aktivovaného módu).
- `_save_data(data, path)`: Uložení dat v binární podobě na disk s příslušnou cestou, v podobě čtveřice seznamů (trénovací vstupy, trénovací výstupy, testovací vstupy, testovací výstupy).
- `text_to_tensor(self, text)`: Pomocná metoda pro kódování textu do podoby potřebné pro učení modelu. Jednotlivé znaky textu jsou mapovány na index daného znaku v abecedě. Tedy např. výsledkem kódování řetězce „abca“ s abecedou „abc“ je pole [0,1,2,0].
- `tensor_to_text(tensor, vocabulary)`: Pomocná metoda pro převod kódovaného textu zpět na řetězec znaků (s použitím abecedy).
- `load_img(img_path, grayscale)`: Pomocná metoda pro načtení obrázku z disku v podobě numpy pole. Příznak `grayscale` určuje, zda se má obrázek převést do odstínů šedi.
- `normalize_image(image)`: Pomocná metoda, která provede normalizaci obrázku – např. převod rozsahu hodnot do intervalu $< 0.5; 0.5 >$, inverze barev (pozadí

tmavé, popředí/text bílé), změna velikosti obrázku: fixní výška 32 pixelů, proporciálně určená šířka atd.

- `deskew(image)`: Pomocná metoda provede otočení obrázku tak, aby text na obrázku byl vodorovně.
- `_load_data_prepared(path)`: Načtení binárních dat z disku (uložených pomocí metody `_save_data`) – načtená data jsou dostupná pomocí čtveřice proměnných `train_x`, `train_y`, `test_x`, `test_y`.
- `_load_data(*args)`: Abstraktní metoda, která je určena k implementaci třídou, která dědí z třídy `AbstractDataset`. Implementací této jediné metody se odlišují jednotlivé třídy reprezentující datasety. Tato metoda má jediný úkol – načtení dat z disku a vrácení čtveřice seznamů: (`train_x` – trénovací vstupy, `train_y` – trénovací výstupy, `test_x` – testovací vstupy, `test_y` – testovací výstupy). Pokud dataset obsahuje pouze trénovací data, lze pro testovací vstupy a výstupy vrátit prázdné seznamy (trénovací sada se automaticky rozdělí na trénovací a testovací v poměru, který určuje atribut `train_test_ratio`, jenž je předán v konstruktoru třídy reprezentující dataset – v případě vynechání je tento parametr nastaven na hodnotu 0.8). Této metodě jsou předány jakékoliv parametry, které jsou potřeba pro načtení dat z disku (např. cesta k obrázkům datasetu a cesta k textovému souboru s dvojicemi název obrázku a textový přepis). Během načítání datasetu metoda využívá výše zmíněné metody jako např. `load_img(img_path, grayscale)` pro načtení obrázku z disku, `normalize_image(image)` pro normalizaci načteného obrázku, `deskew(image)` pro správné otočení obrázku, `text_to_tensor(text)` pro kódování textu atd.
- `load_data(*args, dataset_name)`: Hlavní metoda celé třídy `AbstractDataset`, která je volána zděděnou třídou při inicializaci. Této metodě jsou předány jakékoliv parametry, které jsou potřeba pro případné načtení dat z disku (načtení realizováno pomocí abstraktní metody `_load_data(*args)`, která musí být implementovaná třídou, která dědí ze třídy `AbstractDataset`). Po zavolání metoda nejdříve zkontroluje, zda v adresáři `prepared_datasets_dir_path` předaném při inicializaci existuje soubor `dataset_name` s již načteným datasetem. Pokud ano, obsah souboru je načten pomocí metody `_load_data_prepared`, data jsou pak dostupné pomocí proměnných `train_x`, `train_y`, `test_x`, `test_y`. Pokud soubor s již připraveným datasetem neexistuje, jsou data načteny z disku pomocí abstraktní metody `_load_data(*args)`. Data jsou pak dostupná ve stejných proměnných. Při prvním načtení dat z disku se navíc zkontroluje velikost testovací sady. Pokud je testovací sada prázdná, rozdělí se trénovací sada náhodně v poměru daném hodnotou `train_test_ratio` na testovací a trénovací sadu. Poté se data uloží na disk v binární podobně pomocí funkce `_save_data(data, path)` do adresáře `prepared_datasets_dir_path` s názvem `dataset_name`. Takto je zajištěno, že při následujícím načtení datasetu se načte již předpřipravený dataset ze souboru.

Přidání nového datasetu Pro přidání nového datasetu stačí vytvořit novou třídu v adresáři `src/dataset`, která je zděděna z třídy `AbstractDataset`. Nově vytvořené třídě je povinně v konstruktoru předána abeceda OCR (řetězec obsahující znaky, které umí OCR rozpoznat). Třída musí také povinně implementovat jedinou metodu `_load_data(*args)`, kterou definuje, jakým způsobem se má načíst dataset z disku

a převést do interní reprezentace, v případě nenalezení souboru s již načteným datasetem. Implementací této jediné metody se odlišují třídy reprezentující jednotlivé datasety.

Třída reprezentující dataset uložený v databázi webové aplikace Aby bylo možné trénovat na datasetech, které budou uživatelé nahrávat pomocí webové aplikace, byla vytvořena třída `DatabaseDataset`, která dědí ze třídy `AbstractDataset` a je umístěna v souboru `dataset/database_dataset.py`. Účelem třídy je načíst dataset skládající se z obrázků nahraných uživateli na webový server a anotací těchto obrázků nahraných v databázi webové aplikace.

Třída implementuje jedinou metodu `_load_data()`, která po invokaci z databáze načte obrázky datasetu s identifikátorem `dataset_id`, anotace řádků těchto obrázků (`points` – polygon vyznačující řádek v obrázku a `text` – přepis daného řádku). Načteny jsou pouze anotace řádků s uživateli potvrzenými přepisy (atribut `is_valid` databázové tabulky `row_annotations` je pravdivý). Následně jsou z disku načteny jednotlivé obrázky (dané cestami atributu `path` databázové tabulky `images`), ze kterých jsou poté vyřezány jednotlivé polygony. Výsledné vyřezané polygony jsou pomocí metody `deskew(image)` otočeny tak, aby byl text vodorovně (v případě nakloněného řádku), poté jsou výřezy také normalizovány pomocí metody `normalize_img(image)`. Výstupem metody `_load_data()` je pak čtveřice seznamů: (`train_x` – trénovací vstupy, `train_y` – trénovací výstupy, `test_x` – testovací vstupy, `test_y` – testovací výstupy). Ve výchozím nastavení jsou načtené potvrzené přepisy přiřazeny do trénovací sady.

Při instanciaci objektu třídy `DatabaseDataset` je v konstruktoru předána cesta k adresáři `prepared_datasets_dir_path` pro načtení/uložení již připravených datasetů, cesta k adresáři s obrázky `images_dir_path`, identifikátor datasetu `dataset_id` určující, který dataset se má z databáze načíst a abeceda OCR modelu. Následně je pouze invokována zděděná metoda `load_data`, které je předán název datasetu `dataset_name` nastavený na hodnotu „DatabaseDataset_{dataset_id}“. Invokací této metody dojde k načtení již předpřipraveného datasetu z cesty dané atributem `prepared_datasets_dir_path` a názvem datasetu `dataset_name` popř. k přípravě nového datasetu pomocí implementované metody `_load_dataset` v případě, že nebude soubor s připraveným datasetem nalezen. Celá implementace zděděné metody `load_data(*args)` již byla popsána výše.

- **Trainer**: Třída `Trainer`, které je umístěna v souboru `/ocr/src/trainer.py`, je zodpovědná za samotné natrénování vytvořeného modelu na konkrétním datasetu. V konstruktoru je této třídě předána již vytvořená instance modelu `model`, která se použije pro trénování a instance datasetu `dataset`, na kterém se bude daný model trénovat. Dále jsou předány parametry trénování jako počet epoch `epochs`, „rychlost učení“ `learning_rate`, velikost dávky `batch_size` a použitá abeceda OCR modelu.

  Třída `Trainer` implementuje dvě metody: `train` a `train_batch`. Metoda `train()` definuje základní učící smyčku. Nejdříve je instanciován objekt pro výpočet chybové funkce (CTC loss viz 1.1.4) a objekt pro optimalizaci parametrů sítě (algoritmus Adam [15], který adaptivně mění rychlost učení). Poté je iterováno podle počtu zadaných epoch a v každé iteraci je instanciován objekt třídy `DataLoader`, kterému je

předán objekt datasetu `dataset`, velikost dávky `batch_size` a funkce pro kompletaci jednotlivých dávek z množiny trénovacích vzorů `CollateFn` (umístěna v souboru `/src/ocr/collate.py`). Vytvořený objekt třídy `DataLoader` je možné iterovat – pro každou iteraci objekt vrátí čtveřici dat (`bx` – dávka představující vstupní data, `by` – dávka představující požadovaná výstupní data, `widths_x` – velikosti jednotlivých vzorů vstupní dávky, `widths_y` – velikost jednotlivých vzorů výstupní dávky), která je potřeba pro trénování sítě pro jednu dávku vzorů. Tato čtveřice je společně s instancí modelu `model`, chybovou funkcí `criterion` a optimalizátorem parametrů sítě `optimizer` předána metodě `train_batch`, která převede dávku dat na zařízení (CPU-/GPU), inferuje model `model` na dané dávce dat, spočte chybu pomocí předaného objektu `criterion`, upraví parametry modelu pomocí optimalizátoru `optimizer` a vrátí chybu na této dávce dat zpět metodě `train`. Metoda `train` akumuluje vrácené chyby jednotlivých dávek, následně na konci epochy (po tom co byl model natrénován na všech datech) je vypsána akumulovaná chyba ze všech dávek dané epochy a natrénované parametry jsou uloženy na disku pomocí metody modelu `save_parameters` (při instanciaci modelu byla předána cesta k souboru s parametry modelu), poté se pokračuje další epochou.

Jednotlivé obrázky řádků nemají fixní šířku, stejně tak ani fixní délku očekávaného výstupu sítě (přepsaného řetězce). Pro trénování modelu po více datech najednou (dávce dat) je ale nutné, aby všechna data téže dávky měla stejnou dimenzionalitu. Z tohoto důvodu bylo nutné předefinovat výchozí vytváření dávek (knihovny PyTorch) z množiny trénovacích dat tak, aby se data téže dávky zarovnala vždy tak, aby jejich dimenze odpovídaly datu s největšími dimenzemi dané dávky. K tomuto účelu slouží jíž zmíněná funkce `CollateFn`, kterou využívá třída `DataLoader` pro vytvoření dávek z množiny trénovacích dat. Funkci je předán seznam dvojic (vstupní data, výstupní data) o délce `batch_size`. Následně zjištěn seznam délek dat `widths_x` a `widths_y`. Z těchto seznamů jsou vybrány maximální délky, dle kterých je následně vytvořen tensor `new_batch_x` s dimenzemi (*velikost dávky*, *počet kanálů = 1*, *výška obrázku 32 – stejná pro všechny obrázky*, *maximální šířka vstupních dat*) a tensor pro výstupní data s dimenzemi (*velikost dávky*, *maximální šířka výstupních dat*). Oba tyto tensory jsou vyplněný nulami. Do těchto tensorů jsou následně naskládány jednotlivé vstupní a výstupní data. Výsledkem je, že tensor se vstupními daty i tensor s výstupními daty budou obsahovat jednotlivé vzory, které jsou vyplněny nulami na délku nejdělšího vzoru dané dávky. Funkce pak vrací tyto dva tensory `new_batch_x` a `new_batch_y` se zarovnanými daty a dva seznamy `widths_x` a `widths_y` s informacemi o původních délkách dat.

- **Evaluator**: Třída `Evaluator`, která je umístěna v souboru `/src/evaluator.py`, je zodpovědná za vyhodnocení natrénovaného modelu na konkrétním datasetu. V konstruktoru je této třídě předána již vytvořená instance modelu `model`, která se použije pro vyhodnocení a dataset `dataset`, na kterém se bude daný model vyhodnocovat. Dále je předána velikost dávky `batch_size` a příznak `test_dataset` určující, zda se mají k vyhodnocení použít trénovací nebo testovací data datasetu. Při instanciaci je také vytvořena instance třídy `Decoder` (umístěna v souboru `/src/decoder.py`), která zajišťuje dekódování výstupu sítě zpět na řetězec podle předané abecedy OCR.

  Třída `Evaluator` implementuje pouze jedinou bezparametrickou metodu `evaluate`. Po invokaci této metody se pomocí metody datasetu `set_mode`, která byla popsána u abstraktního datasetu, dle příznaku `test_dataset` nastaví mód datasetu na tré-

novací/testovací. Následně je podobně jako při trénování vytvořena instance třídy `DataLoader`, která je jednou proiterována. V každé iteraci je modelem inferována jedna dávka dat. Výstup sítě je poté dekódován metodou `greedy_decode` třídy `Decoder`. Dekódovaný výstup sítě je stále uveden v podobě seznamu indexů jednotlivých znaků abecedy. Dekódování na výsledný řetězec poté zajišťuje statická metoda `tensor_to_text` třídy `AbstractDataset`. Finální dekódovaný řetězec je poté porovnán s požadovaným výstupem sítě pro daný vzor a je spočtena jejich vzdálenost. K výpočtu vzdálenosti je použita Levenshteinova vzdálenost [26], která určuje počet operací (vložení znaku, odstranění znaku, záměna znaku za jiný znak) nutný k převodu jednoho řetězce na druhý řetězec, pomocí knihovny pylev[27].

Pro všechny vyhodnocené vzory dat je vzdálenost od jejich cílového (*ground-truth*) přepisu akumulována. Po vyhodnocení sítě na všech datech je tato akumulovaná vzdálenost vypsána uživateli na standardní výstup v procentuální podobě udávající počet procent znaků, které byly identifikovány sítí špatně (*character-error*). Aby měl uživatel představu o přepisech predikovaných sítí, je také vypsáno 10 náhodných dat, které byly vyhodnoceny, v podobě dekódovaného řetězce, očekávaného výstupního řetězce a vzdálenosti těchto dvou řetězců. Ukázka výstupů při trénování a následném vyhodnocení CRNN OCR modelu na konkrétním datasetu je ukázána na obrázku 3.2.

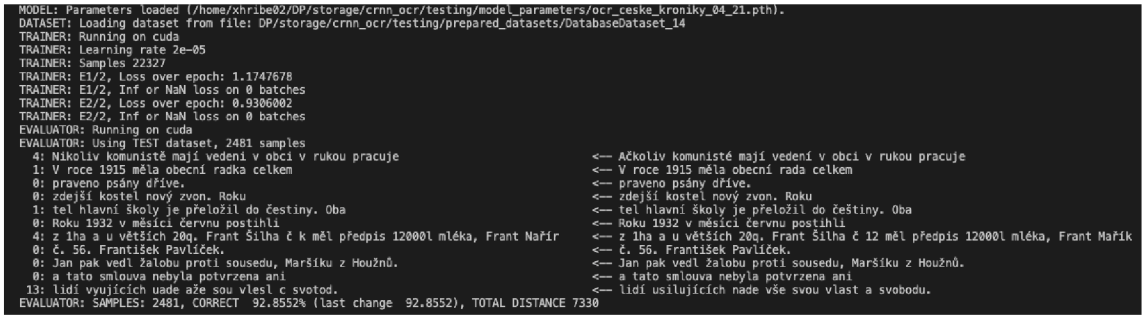
Dekódování výstupu sítě Výstup sítě pro jeden vzor je v podobě tensoru s diemenzemi (*počet sloupců*, *počet tříd*). To odpovídá vstupnímu obrázku daného řádku, jehož šířka byla díky pooling vrstvám konvoluční sítě zmenšena na *počet sloupců*. Počet tříd je pak dán velikostí výstupní abecedy OCR. Úkolem dekodéru je zvolit z každého sloupce jednu třídu, která poté bude představovat predikované písmeno pro tento sloupec. Nejjednodušším způsobem je výběr nejpravděpodobnější třídy daného sloupce. Toto realizuje metoda `greedy_decode` třídy `Decoder`. Výsledkem je pak seznam indexů tříd, které byly zvoleny pro jednotlivé sloupce.

Jednotlivé sloupce nemusí pokrývat oblast jednoho znaku v původním obrázku (viz 1.1.4). Některé znaky jsou širší, některé naopak úzké. Pokud byla pro sousední sloupce zvolena stejná třída (stejný znak abecedy), neznamená to, že ve výsledném přepise OCR modelu bude tento znak dvakrát vedle sebe. Dekódování počítá s tzv. *blank* znakem (třídou), který představuje prázdný znak (daný sloupec obrázku nepředstavuje žádný znak). Tento znak je také použit pro oddělení sousedních stejných znaků od sebe. Pokud jsou tedy pro sekvenci sousedních sloupců nalezeny stejné znaky (třídy) oddělené znakem *blank*, jedná se o opakování stejného znaku. Pokud nejsou mezi stejnými znaky umístěny znaky *blank*, je tato sekvence chápána jako jeden znak rozprostírající se mezi několik sloupců. Z výsledné sekvence jsou nakonec odstraněny všechny *blank* znaky.

Skript pro demonstraci trénování a vyhodnocení CRNN OCR modelu Pro trénování a vyhodnocení výše popsaného modelu CRNN byl vytvořen skript `/ocr/src/scripts/train.py`, který má tyto parametry: `-e` počet epoch trénování, `-l` rychlost učení *learning-rate* a `-w` cesta k souboru s trénovatelnými parametry modelu. Dále je možno zadat parametry `-p` s cestou k adresáři s připravenými datasety, `-i` cesta ke složce s obrázky (využité pouze u `DatabaseDataset`) a parametr `-d` s identifikátorem datasetu (využité

[27] https://pypi.org/project/pylev

pouze u `DatabaseDataset`). Tento skript pracuje se čtyřmi výše popsanými komponentami: `Model`, `Dataset`, `Trainer` a `Evaluator`. Nejdříve je vytvořena instance modelu – při instanciaci je předána instance konkrétní konvoluční sítě `cnn`, dimenzionalita skrytých vektorů rekurentní sítě `feature_size`, počet vrstev rekurentní sítě `rnn_layers`, příznak zda se má použít obousměrná rekurentní síť `rnn_bidirectional` a cesta k souboru s trénovatelnými parametry modelu `parameters_path`. Dále je vytvořena instance datasetu, na kterém se bude model trénovat a poté i vyhodnocovat. Lze vybrat jakoukoliv třídu zděděnou z třídy `AbstractDataset` – této třídě jsou předány parametry potřebné pro načtení dat z disku a převedení do interní reprezentace, abeceda OCR modelu a poměr `train_test_ratio` pro rozdělení na trénovací a testovací sadu v případě, že dataset obsahuje pouze trénovací sadu (viz sekce s popisem datasetu). Vytvořené objekty modelu a datasetu jsou následně předány konstruktoru třídy `Trainer` a `Evaluator` společně s atributem `batch_size` určující velikost dávky. Třídě `Trainer` jsou navíc předány parametry: `epochs` – počet epoch trénování, `learning_rate` – „rychlost učení“ a abeceda OCR modelu. Třídě `Evaluator` je navíc předán příznak `test_dataset` určující zda se má model vyhodnotit na trénovací nebo testovací datové sadě (dojde k přepnutí módu datasetu) – ve výchozím stavu je model vždy vyhodnocován na testovací sadě. Po instanciaci těchto čtyř objektů je pouze na objektu třídy `Trainer` zavolána metoda `train` a poté na objektu třídy `Evaluator` zavolána metoda `evaluatote`. Obě tyto metody již byly popsány dříve. Ukázka možného výstupu toho skriptu je zobrazena na obrázku 3.2.



Obrázek 3.2: Ukázka výpisu programu při trénování a následném vyhodnocení CRNN modelu. Nejdříve jsou načteny parametry modelu, následně je načten dataset z již připraveného souboru na disku, poté jsou vypsány parametry trénování a zařízení určené k trénování (CPU/GPU). Poté je spuštěno trénování na daném počtu epoch. Na závěr je spuštěna evaluace na testovací sadě. Součástí evaluace je výpis deseti náhodných dat, kde na každém řádku je vpravo vypsán *ground-truth* přepis daného řádku, vlevo je pak vypsán přepis predikovaný sítí. Číslo v levé části udává počet znaků, ve kterých se tyto dva přepisy liší. Poslední řádek evaluace shrnuje úspěšnost rozpoznávání na celém testovacím datasetu. Pro ukázku byl použit dataset „OCR české kroniky“, který je dále popsán v sekci zaměřené na testování systému.

Služba pro trénování CRNN OCR

V této sekci bude popsána implementace služby, která využívá výše popsanou implementaci modelu CRNN OCR. Po přijetí HTTP POST požadavku zaslaného na adresu `crnn_ocr:5200/ocr_annotator_api/run_crnn_ocr_training` dojde k aktivaci této služby. Služba zjistí úkol, který čeká na její vykonání (viz 3.2.1) a pokud takový existuje, začne tento úkol vyko-

návat. Vykonání úkolu zajišťuje metoda `execute_ocr_training_tasks` třídy `OcrTraining`, která je umístěna v souboru `/modules_api/modules_api/modules/crnn_ocr_training.py`.

Vykonání úkolu Před trénováním CRNN OCR je nutné načíst trénovací data z disku do paměti – pokud nebyl dataset již připraven. Tyto data jsou následně uloženy na disk v binární podobě. K ukládání připravených datasetů je určena složka `/storage/crnn_ocr/prepared_datasets`, která je umístěna ve sdíleném uložišti. Obsah této složky je nejdříve před začátkem každého trénování smazán, protože se daný dataset od posledního trénování mohl změnit (uživatelé mohli potvrdit správnost dalších přepisů). Následně je pomocí statické metody `run_train` třídy `OcrTraining` zahájeno trénování. Této metodě je předán identifikátor datasetu na kterém se bude trénovat (úkol obsahuje identifikátor konkrétního datasetu), počet epoch (attribute `num_epochs`), cesta k parametrům instance modelu (atribut `model_parameters_path`) a „rychlost učení“ (attribut `learning_rate`). Tato funkce nejdříve instanciuje model, kterému předá cestu k parametrům dané instance modelu (relativní cesta ke složce `/storage/crnn_ocr/model_parameters`), a další parametry modelu jako např. velikost výstupní abecedy (počet tříd ke klasifikaci), počet vrstev rekurentní sítě, velikost skrytých vektorů rekurentní sítě, použitý backbone (konvoluční síť určená k extrakci příznaků) atd. Dále je instanciován objekt třídy `DatabaseDataset`, kterému je předán identifikátor datasetu, výstupní abeceda atd. Při instanciaci datasetu dojde k načtení obrázků ze sdíleného uložiště, ořezání řádků podle anotací uložených v databázi a uložení připraveného datasetu do složky `/storage/crnn_ocr/prepared_datasets`. Následně je instanciován objekt třídy `Trainer`, kterému je předána výstupní abeceda, připravená instance modelu, připravená instance datasetu, počet epoch (`num_epochs`), velikost dávky (`batch_size`) a „rychlost učení“ (`learning_rate`). Jako poslední je instanciován objekt třídy `Evaluator`, kterému je předána velikost dávky `batch_size`, připravená instance modelu `model` a datasetu `dataset` a poměr `train_test_ratio` pro rozdělení trénovací sady na trénovací a testovací sadu, v případě chybějící testovací sady. Tento poměr je v tomto případě nastaven na hodnotu 0.9. V případě třídy `DatabaseDataset` jsou všechna uživatelem potvrzená data (viz. dále v kapitole o webové aplikaci) zařazována do trénovací sady, proto v tomto případě dojde k rozdělení datasetu na trénovací a testovací (validační) sadu v daném poměru.

Na vzniklé instanci třídy `Train` je pak zavolána metoda `train`, která spustí samotné učení a vrátí výslednou chybu učení (loss). Poté je na vzniklé instanci třídy `Evaluator` zavolána metoda `evaluate`, která vrátí procentuální chybu přepsaných znaků na testovací sadě (*character-error*). Výsledná chyba učení a *character-error* jsou pak zapsány zpět k danému úkolu do databáze (attribut `loss` a `character_error` databázové tabulky `tasks`).

Po vykonání úkolu je zapsán čas dokončení tohoto úkolu a předáno řízení dispečinku úkolů (viz 3.2.1).

Služba pro Inferenci CRNN OCR

Po příjetí HTTP POST požadavku zaslaného na adresu `crnn_ocr:5200/ocr_annotator_api/run_crnn_ocr_infering` dojde k aktivaci této služby. Služba zjistí úkol, který čeká na její vykonání (viz 3.2.1) a pokud takový existuje, začne tento úkol vykonávat. Vykonání úkolu zajišťuje metoda `execute_ocr_infering_tasks` třídy `OcrInfering`, která je umístěna v souboru `/modules_api/modules_api/modules/crnn_ocr_infering.py`.

Vykonání úkolu Nejdříve je instanciován model, který se bude používat pro inferenci CRNN OCR, a dekodér, který slouží k dekódování výstupu neuronové sítě na textový řetězec. Instanciaci modelu zajišťuje funkce `create_model`, které jsou předány parametry modelu jako např. velikost výstupní abecedy, počet vrstev rekurentní sítě, velikost skrytých vektorů rekurentní sítě, použitý backbone (konvoluční síť pro extrakci příznaků), cesta k parametrům modelu atd. Následně je spuštěna samotná inference, podobně jako u Tesseract OCR.

Pomocí metody `get_task_images(task_id)` třídy `DB` jsou z databáze zjištěny informace o obrázcích, na kterých se má CRNN OCR inferovat (úkol obsahuje identifikátor konkrétního datasetu). Pro každý obrázek patřící do daného datasetu jsou metodou `get_image_row_annotations_infer` třídy `DB` načteny informace o anotacích řádků, které jsou určeny pro inferování OCR (atribut `is_valid`, který značí zda uživatelé přepis tohoto řádku potvrdili, je nepravdivý nebo nulový). Každá načtená anotace řádku obsahuje v atributu `points` body polygonu vyznačujícího tento řádek. Z tohoto polygonu je vypočten bounding box, který je vyřezán z daného obrázku uloženém ve sdíleném uložišti `/storage/images`, následně je podobě jako u předchozích služeb vyřezán samotný polygon, který je navíc otočen tak, aby byl text vodorovně. Takto vyřezané řádky jsou inferovány připraveným modelem, kterému je předán vyřezaný obrázek (v podobě tensoru). Výstup modelu je dekódován pomocí připraveného dekodéru, čímž je zjištěn textový přepis řádku. Výsledný přepis je pro každý výřez zapsán zpět do databáze k příslušné anotaci řádku do atributu `text` (zajištěno metodou `update_row_annotation_text` třídy `DB`).

Po vykonání úkolu je zapsán čas dokončení tohoto úkolu a předáno řízení dispečinku úkolů (viz 3.2.1).

3.2.5 Webová aplikace

Webová aplikace slouží jako prostředek uživatele k interakci se všemi ostatními moduly systému. Aplikace přihlášenému uživateli umožňuje vytváření, mazání a editaci datasetů, nahrávání skenů dokumentů k jednotlivým datasetům, mazání skenů dokumentů, anotaci těchto skenů (vyznačení pozic řádků a textových regionů, včetně zadání přepisů vyznačených řádků), mazání a editaci anotací skenů, potvrzování správně vyznačených a přepsaných řádků, automatickou detekci řádků a textových regionů ve skenech jednotlivých datasetů, vytváření nových instancí OCR modelů z již existujících instancí, trénování instancí OCR modelu na uživateli potvrzených přepisech řádků a inferenci takto natrénovaných instancí OCR modelů na nepotvrzených přepisech řádků.

Struktura zdrojového kódu Veškeré zdrojové soubory týkající se webové aplikace jsou uloženy v adresáři `/app`. Jelikož se jedná o aplikaci založenou na *back-end* frameworku Laravel[28], struktura souborů aplikace je předem stanovena. Aplikace je navíc napsána tak, aby co nejvíce využívala oddělení části serveru *back-end* a uživatelské části *front-end*. Všechny potřebné metody pro manipulaci s daty aplikace (vytvoření, edice, získání a mazání záznamů) jsou proto definovány jako samostatné metody kontrolérů, které jsou dostupné pomocí zvolených *end-point* URL adres. Kontroléry implementující tyto metody jsou uloženy v adresáři `/app/app/Http/Controllers/Api`. Pro každý doménový objekt, k jehož datům je potřeba přistupovat je zde vytvořena třída kontroléru. Kontroléry definující metody, které neslouží pro manipulaci dat, ale pro vykreslování HTML stránek, jsou umístěny v adresáři `/app/app/Http/Controllers`. Mapování relační databáze na objekty domény je zajištěno

[28] laravel.com

pomocí frameworku Eloquent[29], který je součásti frameworku Laravel. Soubory popisující mapování modelů jsou umístěny v adresáři `/app/app/Models`. Pro snadnější správu databáze jsou využívány migrační soubory, které umožňují popsat schéma databáze. Migrační soubory jsou uloženy v adresáři `/app/database/migrations`. Soubory popisující inicializační data databáze jsou uloženy v adresáři `/app/database/seeders`.

Zdrojové kódy popisující uživatelskou část aplikace (*front-end*) jsou umístěny v adresáři `/app/resources`. Pro základní kostru stránky byla využita *open-source* HTML šablona „StartBootstrap SB Admin 2“[30]. Zdrojové kódy této šablony jsou umístěny v adresáři `/app/resources/template`. Pro efektivnější práci s CSS byl využit CSS pre-processor Sass[31]. Sass soubory jsou umístěny v adresáři `/app/resources/sass`. Pro zvýšení reaktivity aplikace a dekompozici uživatelské části na moduly a zajištění znovupoužitelnosti jednotlivých modulů je navíc využit *front-end* framework Vue.js[32], díky kterému je možné vytvářet izolované komponenty obsahující vlastní HTML, CSS a Js kód. Tyto komponenty lze parametrizovat a umísťovat je tak na více míst v aplikaci s lehce pozměněnou funkcionalitou. Veškeré soubory týkajících se komponent ve frameworku Vue.js jsou uloženy v adresáři `/app/resources/js/components`.

Dekompozice uživatelské části webové aplikace na komponenty Pro jednotný přístup a zajištění znovupoužitelnosti kódu byly pomocí frameworku Vue.js navrženy 3 abstraktní komponenty, které jsou uloženy v adresáři `/app/resources/js/components/general`. Tyto komponenty realizují časté operace, jež uživatelé webové aplikace provádějí:

- **Načtení a výpis dat v požadovaném formátu:** Nejčastější operací, kterou uživatelé provádějí, je načtení požadovaných dat ze serveru a výpis těchto dat v požadovaném formátu na obrazovku. K tomuto účelu slouží komponenta uložená v souboru `AbstractListComponent.vue`. Této komponentě je umožněno předat dva parametry (*property* v terminologii Vue.js): parametr `fetch_records_route` udává URL adresu, ze které se stáhnou požadovaná data (*end-point* URL adresa metody některého z API kontrolérů) a parametr `filter_params` udávající parametry pro filtraci stahovaných dat (ve výchozím stavu může být předán prázdný objekt). Komponenta naopak vystavuje vně slot `template` (místo pro zadání šablony pro vykreslení obsahu) s daty `records`, která obsahují načtené záznamy ze serveru. Úkolem nadřazené komponenty je pak doplnit formát výpisu těchto dat – typicky v podobě HTML šablony, do které jsou údaje jednotlivých záznamů vepisovány.

  Všechny připravené metody API kontrolérů podporují stránkování obsahu – předání atributů `records_per_page` a `page`, které určují kolik záznamů se má vracet pro jednu stránku (vždy je stažena pouze jedna stránka) a číslo stránky, kterou uživatel požaduje. Komponenta zajišťuje stránkování obsahu pomocí vykreslení HTML šablony s informacemi o jednotlivých stránkách a odesílání údajů o požadované stránce a počtu záznamů na stránce společně s parametry filtrování na konkrétní předanou URL adresu `fetch_records_route`.

  Tuto komponentu využívají všechny komponenty webové aplikace, které vypisují data – jako například seznam existujících anotací, datasetů, úkolů prováděných nad datasety nebo žebříček uživatelů.

[29] https://laravel.com/docs/8.x/eloquent
[30] https://github.com/StartBootstrap/startbootstrap-sb-admin-2
[31] https://sass-lang.com
[32] https://vuejs.org

- **Filtrace načteného seznamu záznamů**: Další častou operací uživatelů je filtrace vypisovaných záznamů. K tomuto účelu slouží komponenta `AbstractFilterComponent.vue`, která spolupracuje s komponentou pro výpis seznamů `AbstractListComponent.vue`. Filtrační komponenta má za úkol správu objektu `filter_params` uchovávajícího filtrační parametry pro komponentu vypisující záznamy.

  Filtrační komponentě lze předat jediný parametr `default_params` představující objekt s výchozími parametry filtru. Komponenta naopak vystavuje vně slot `template` (místo pro zadání šablony pro vykreslení obsahu) s daty `params`. Úkolem nadřazené komponenty je pak doplnit HTML formulář, jehož prvky jsou pomocí *data-binding* svázány s atributy objektu `params`. Abstraktní filtrační komponenta poté hlídá události formuláře a při detekci změny obsahu formuláře uživatelem emituje událost `abstractFilterEv_ParamsChanged` s daty filtračního formuláře `params`. Tato událost je odchycena abstraktní komponentou pro výpis seznamu záznamů, která filtrační parametry odešle na server spolu s novou žádostí o data.

- **Vytvoření a editace záznamu:** Poslední častou operací realizovanou uživateli webových aplikací je vytvoření nového záznamu, popř. editace již existujícího záznamu. K těmto účelům slouží abstraktní komponenta `AbstractCreateEditFormComponent.vue`. Komponentě je předán parametr `api_url`, kde budou odesílány požadavky pro vytvoření/editaci/stažení záznamu.

  Komponenta vystavuje vně slot `template` (místo pro zadání šablony pro vykreslení obsahu) s daty `data`. Úkolem nadřazené komponenty je pak doplnit HTML formulář, jehož prvky jsou pomocí *data-binding* svázány s atributy objektu `data`. Abstraktní komponenta nabízí tlačítka pro odstranění a vytvoření nového záznamu popř. aktualizaci již existujícího záznamu. Komponenta sama pozná jestli se bude vytvářet nový záznam nebo aktualizovat již existující záznam podle existence skrytého atributu s primárním klíčem daného záznamu. Pokud uživatel vyplní prázdný formulář a odešle jeho obsah na server, dojde k vytvoření nového záznamu (formulář neobsahoval primární klíč), jehož atributy jsou zpět načteny touto komponentou i s primárním klíčem záznamu v databázi. Pokud uživatel edituje již existující záznam, je záznam nejdříve načten do této komponenty (i s primárním klíčem záznamu v databázi), takže komponenta má informaci, že se bude záznam editovat. Komponenta podporuje také další parametry, které umožňují např. zadat název primárního klíče (výchozí je `id`), výchozí hodnoty formuláře atd.

Úvodní stránka aplikace - Přehled

Po zaregistrování a přihlášení uživatele do webové aplikace se uživateli zobrazí úvodní stránka, která obsahuje přehled počtu jím nahraných obrázků a datasetů, potvrzených přepisů řádků a běžících úkolů, které vytvořil. V dolní části stránky je poté umístěn stručný návod pro používaní aplikace, který se snaží uživateli v podobě jednoduchých odrážek a ilustrací přiblížit fungování a práci v aplikaci. Ukázka úvodní stránky je zobrazena na obrázku 3.3. Pro ilustrační obrázky byla využita *open-source* webová knihovna unDraw[33].

Obrázek 3.3: Ukázka části úvodní stránky webové aplikace. V horní části jsou zobrazeny statistiky uživatele, zbytek stránky je věnován stručnému návodu pro obsluhu aplikace. Na obrázku lze vidět pouze část tohoto návodu.

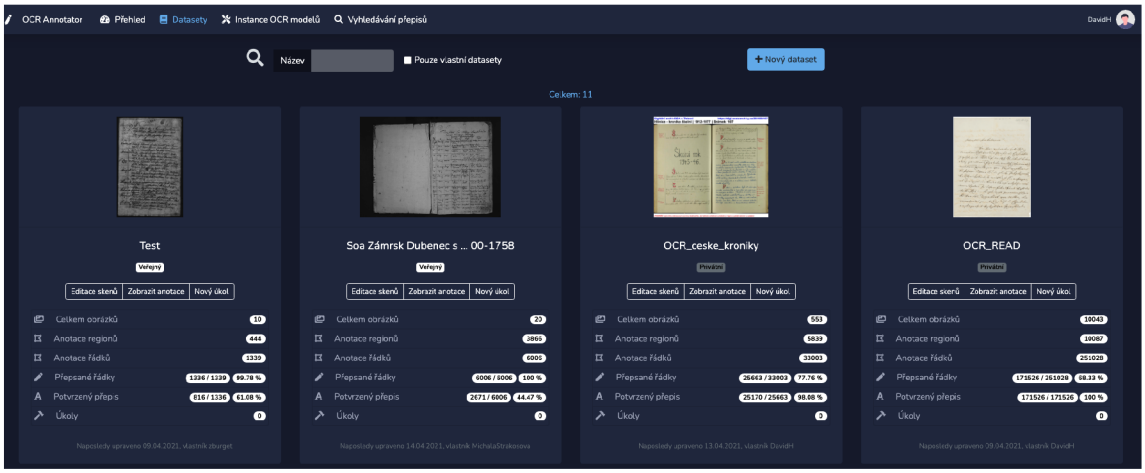
Stránka pro správu datasetů

Jednou z nejdůležitějších stránek je stránka pro správu datasetů, na kterou se uživatel dostane po kliknutí na záložku „Datasety“ v horní části obrazovky. Po načtení stránky jsou uživateli zobrazeny všechny jeho datasety, včetně datasetů označených ostatními uživateli jako veřejné. Ukázka stránky pro správu datasetů, včetně 4 zobrazených datasetů, je zobrazena na obrázku 3.4. V horní části obrazovku se nachází filtrační komponenta `datasets/DatasetsFilterComponent.vue` využívající již popsanou abstraktní filtrační komponentu. Uživatelé mohou filtrovat datasety podle jejich názvu, popř. si mohou zobrazit pouze vlastní vytvořené datasety zaškrtnutím tlačítka „Pouze vlastní datasety“. Dále se v horní části nachází tlačítko pro vytvoření nového datasetu „Nový dataset“, jehož funkcionalita bude popsána později.

Ve zbytku stránky je pak umístěna komponenta `datasets/DatasetListComponent.vue` pro výpis seznamu načtených datasetů, která využívá již popsanou abstraktní komponentu pro výpis. Pro jednotlivé vypsané datasety je vybrán jeden z nahraných obrázků, který je zobrazen jako náhledový obrázek, dále je vypsán název datasetu, počet obrázků v datasetu, počty anotací obrázků daného datasetu, počet přepsaných řádků, počet uživateli potvrzených řádků a počet běžících úkolů pro daný dataset (např. detekce řádků/regionů, trénování/inference OCR). U každého datasetu je také v posledním řádku napsán jeho vlastník.

[33] https://undraw.co/illustrations

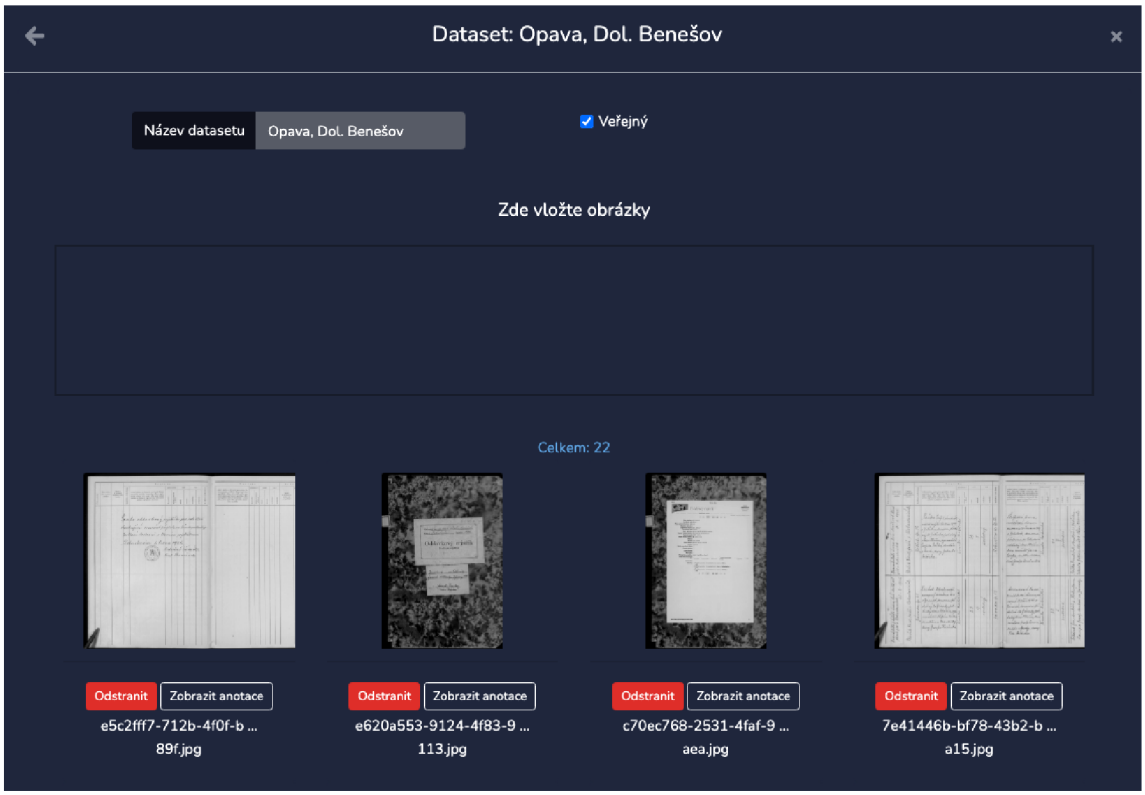
Seznam datasetů je automaticky v určitých intervalech obnovován – není tak nutné načítat znovu celou stránku.



Obrázek 3.4: Ukázka stránky pro výpis a správu datasetů. V horní části je zobrazen filtrační formulář a tlačítko pro vytvoření nového datasetu. Ve zbytku stránky je zobrazen seznam vypsaných datasetů v požadovaném formátu. Na tomto obrázku jsou vypsány 4 datasety. První dva datasety jsou vytvořeny genealogy, kteří pomáhali testovat systém, poslední dva datasety byly do aplikace naimportovány pro otestování úspěšnosti rozpoznávání znaků.

Vytvoření a editace datasetu Pro vytvoření nového datasetu stačí uživateli kliknout na tlačítko „Nový dataset“, které je umístěno v horní části obrazovky (lze vidět na obrázku 3.4), čímž se otevře modální okno, kde uživatel vyplní název nového datasetu a jeho přístupnost (veřejný/privátní) a vytvoření potvrdí kliknutím na tlačítko „Vytvořit“. Poté je přesměrován zpět na aktualizovaný seznam datasetů, který obsahuje i nově vytvořený dataset – stránka se nenačítá znovu, pouze se stáhnou ze serveru aktualizovaná data.

Pro nahrání nových skenů k takto vytvořenému datasetu stačí kliknout v seznamu datasetů u zvoleného datasetu na tlačítko „Editace skenů“, čímž dojde k zobrazení modálního okna s informace o daném datasetu a obrázky tohoto datasetu. Ukázka tohoto modálního okna je zobrazena na obrázku 3.5. Toho modální okno je realizováno komponentou `datasest/DatasetsCreateEditFormComponent.vue`. V horní části je zobrazen formulář pro editaci atributů datasetu (název, přístupnost) realizovaný pomocí abstraktní komponenty pro vytvoření/editaci záznamů. Uprostřed je umístěna komponenta `datasets/UploadImages.vue` pro nahrávání obrázku k tomuto datasetu. Ve zbytku okna je umístěna komponenta pro výpis seznamu obrázku tohoto datasetu `datasets/ImagesListComponent.vue` (využívající abstraktní komponentu pro výpis seznamu). Pro každý obrázek datasetu je vypsán jeho název a zobrazen náhled. Dále jsou pro každý obrázek k dispozici dvě tlačítka: tlačítko „Odstranit“ pro odstranění tohoto obrázku z datasetu a tlačítko „Zobrazit anotace“ pro zobrazení anotací tohoto obrázku (anotace obrázků budou popsány později). Jelikož je seznam obrázku realizován pomocí stejné abstraktní komponenty pro výpis jako seznam datasetů, je možné seznam obrázků také stránkovat. V dolní části obrazovky je pak umístěno tlačítko pro odstranění celého datasetu, popř. potvrzení aktualizace datasetu (na obrázku není vidět).



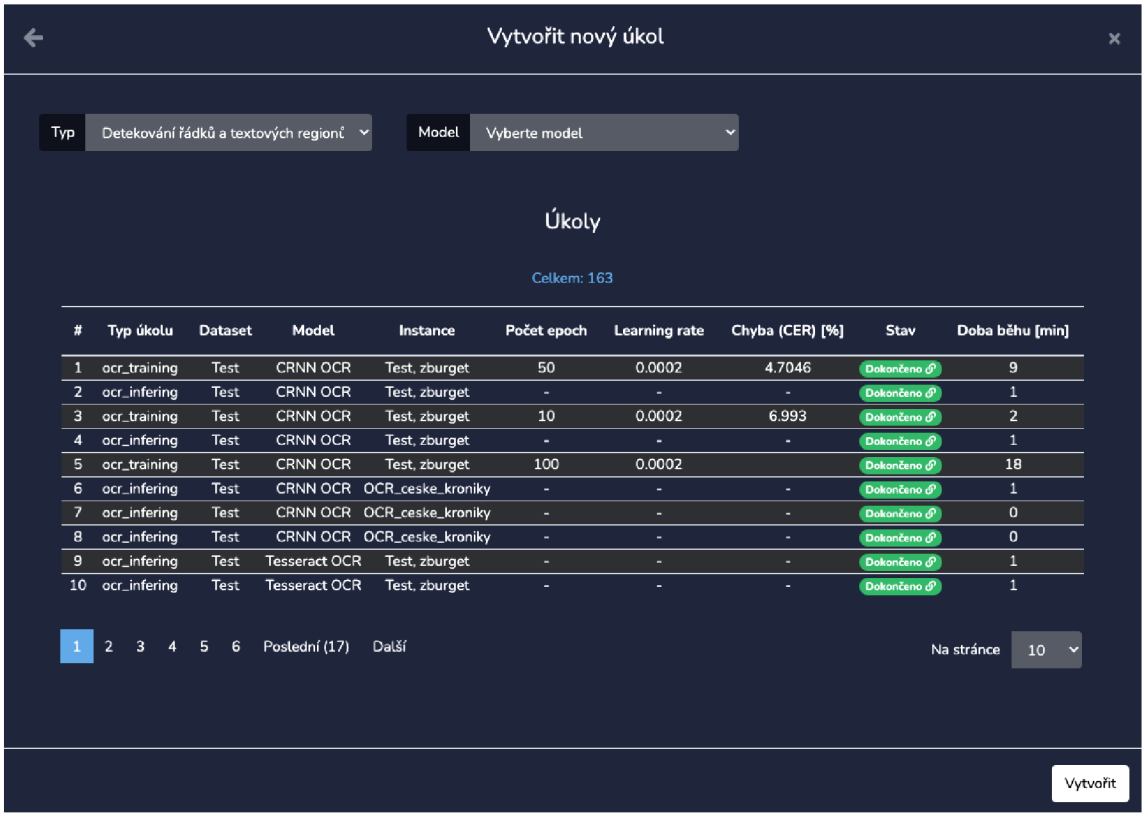
Obrázek 3.5: Ukázka modálního okna umožňujícího editaci zvoleného datasetu a nahrávání/mazání skenů tohoto dataset. Pro každý sken je také možné zobrazit anotace tohoto skenu. Celý dataset je možné smazat.

Vytváření nových úkolů nad datasety

Uživatelé aplikace mohou vytvářet nad svými (popř. veřejnými) datasety nové úkoly jako např. automatická detekce řádků a textových regionů, trénování instancí OCR modelů na uživateli potvrzených přepisech řádků a inference již natrénovaných instancí OCR modelů na nepotvrzených přepisech řádků. K vytváření a plánování nových úkolů slouží modální okno, které se uživateli zobrazí po kliknutí na tlačítko „Nový úkol“ u zvoleného datasetu v seznamu datasetů. Ukázka modálního okna pro vytváření nových úkolů nad zvoleným datasetem je zobrazena na obrázku 3.6.

V horní části tohoto okna je umístěn formulář, pomocí kterého uživatel specifikuje úkol, který chce nad daným datasetem naplánovat – zvolí typ úkolu (detekce, trénování OCR, inference OCR), vybere model (Tesseract OCR, CRNN OCR, ...) a konkrétní instanci zvoleného modelu. V případě trénování lze navíc zvolit počet epoch a rychlost učení (*learning-rate*). Po specifikaci nového úkolu a kliknutí na tlačítko „Vytvořit“ je nově vytvořený úkol zapsán do databázové tabulky `tasks`, následně je kontaktována služba `CRNN OCR`, která zajišťuje mimo jiné dispečink úkolů mezi jednotlivé služby (popsáno v sekci 3.2.1). Formulář pro vytoření nového úkolu je realizován komponentou `datasets/DatasetsMakeTaskFormComponent.vue`, která využívá abstraktní komponentu pro vytváření záznamů.

Ve zbytku modálního okna je umístěna komponenta pro výpis úkolů zvoleného datasetu (`modelInstances/TasksListComponent.vue`), která využívá abstraktní komponentu pro výpis záznamů. U každého úkolu je vypsán jeho typ, název datasetu na kterém byl plánován,



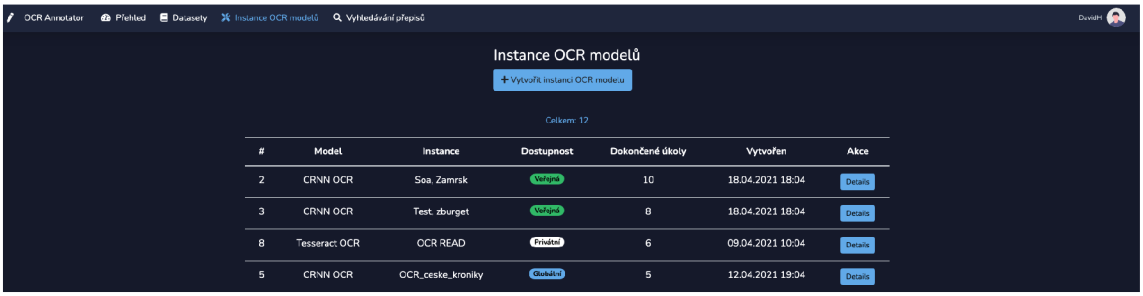
Obrázek 3.6: Ukázka modálního okna umožňujícího vytváření nových úkolů nad zvoleným datasetem. Vytvořené úkoly je možné zpětně procházet a zjistit tak informace o jejich běhu.

model, instance, počet epoch, rychlost učení, chyba (*character-error*), stav a doba běhu. Nově vytvořené úkoly jsou automaticky zobrazeny v tomto seznamu a jejich stav je nastaven na hodnotu „Plánováno“. Pokud je úkol předán k vykonání některou ze služeb, je stav změněn na hodnotu „Probíhá“. Po skončení provádění úkolu je zobrazen stav „Dokončeno“. Seznam úkolů je automaticky v určitých intervalech obnovován – není tak nutné obnovovat celou stránku.

Automatický systém doučování Aby bylo zajištěno automatické doučování z uživateli opravených záznamů, byla vytvořena API *end-point* URL adresa `/api/tasks/global`, která slouží k plánování úkolů pro globální instance OCR modelů. Úkoly plánované pro globální instance OCR modelů nejsou omezeny na konkrétní datasety, proto jsou provedeny nad všemi datasety, které jsou nahrány v aplikaci. Pro vyloučení konkrétních datasetů z provádění globálních úkolů slouží atribut `run_global_tasks` databázové tabulky `datasets`, který je ve výchozím stavu nastaven na pravdivou hodnotu. Zmíněná URL adresa má navíc parametr `type` určující typ OCR modelu (Tesseract, CRNN), `train` jehož přítomnost naplánuje úkol trénování a parametr `infer` jehož přítomnost naplánuje inferenci. Pro Tesseract i CRNN OCR jsou vytvořeny speciální instance s názvem „Global“, které slouží k provádění takto plánovaných úkolů. Pomocí systému CRON lze pak zajistit automatický přístup na tuto URL adresu, čím jsou periodicky plánovány globální úkoly.

Vytváření nových instancí OCR modelů

Systém podporuje vytváření nových instancí OCR modelů z již existujících instancí OCR modelů. Tyto instance se odlišují pouze unikátní cestou k souboru s natrénovanými parametry modelu. Při vytvoření nové instance se na disku vytvoří kopie souboru s naučenými parametry děděné instance a cesta k této kopii souboru se pak přiřadí instanci nové. Díky podpoře vytváření nových instancí lze například trénovat oddělené instance stejného modelu na jiných typech matričních záznamů, a tedy jiných typech písem.



Obrázek 3.7: Ukázka stránky pro správu instancí OCR modelů.

Stránka pro správu instancí je dostupná po kliknutí na tlačítko „Instance OCR modelů“ v horní navigační liště. Ukázka této stránky je zobrazena na obrázku 3.7. Tato stránka obsahuje seznam již vytvořených instancí realizovaný pomocí komponenty `modelInstances/ListComponent.vue`, která využívá abstraktní komponentu pro výpis seznamu. U každé vytvořené instance je vypsán typ modelu, název této instance, dostupnost (veřejná, privátní, globální), počet dokončených úkolů touto instancí a datum vytvoření. Pro každou instanci je k dispozici také tlačítko „Detail“, které zobrazí seznam úkolů realizovaných touto instancí, podobě jako u seznamu úkolů realizovaných nad datasetem viz obrázek 3.6.

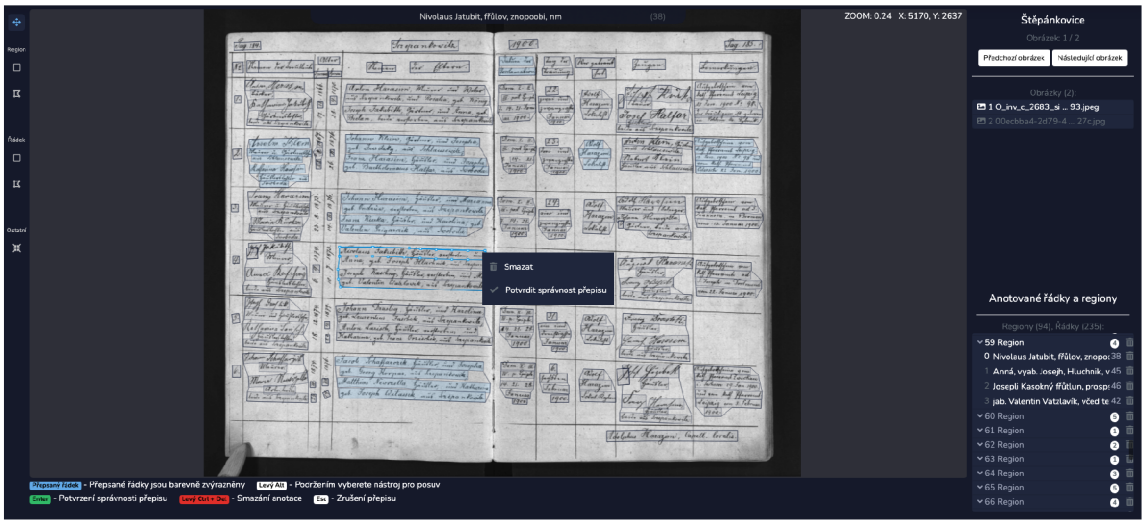
V horní části okna je umístěno tlačítko „Vytvořit instanci OCR modelu“, které zobrazí modální okno podobě jako u vytváření nového úkolu. Uživatel vybere model a základní instanci tohoto modelu, ze které se zdědí natrénované parametry. Následně zadá název nové instance a její přístupnost. Poté vše potvrdí kliknutím na tlačítko „Vytvořit“.

Anotační komponenta

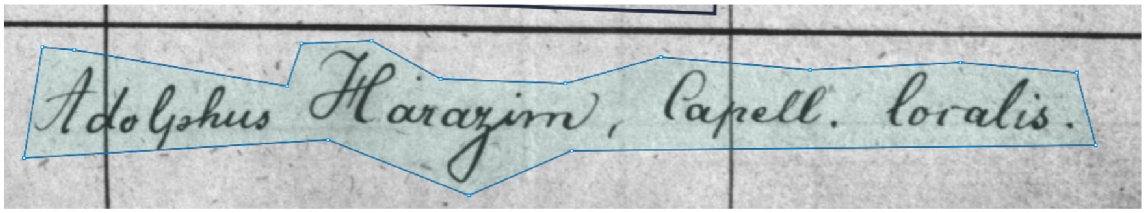
Jádrem webové aplikace je anotační komponenta, která slouží pro prohlížení a anotování jednotlivých skenů. Po kliknutí na tlačítko „Zobrazit anotace“ u vybraného datasetu v seznamu datasetů dojde k přesměrování na stránku s anotační komponentou a zobrazení prvního skenu zvoleného datasetu. Lze rovnou přejít i na anotace konkrétního skenu po kliknutí na tlačítko „Editace skenů“ zvoleného datasetu a následné kliknutí na tlačítko „Zobrazit anotace“ u konkrétního skenu datasetu. Samotná stránka pro anotaci je zobrazena na obrázku 3.8.

Celá stránka, zobrazená na obrázku 3.8, se dělí na 3 části. V levé části je seznam anotačních nástrojů, kde si uživatelé mohou vybrat mezi nástroji pro vytváření řádků a regionů. Oba typy anotací je možné vyznačit pomocí polygonu (ukázka zobrazena na obrázku 3.9) nebo obdélníku (*bounding-box*).

Uprostřed je zobrazen samotný sken dokumentu pro anotování – sken již obsahuje vytvořené anotace, které byly načteny z databáze. Tmavě modré pozadí mají anotace, u kterých zatím nebyl žádným uživatelem potvrzen jejich přepis. Světle modře jsou naopak vyznačeny



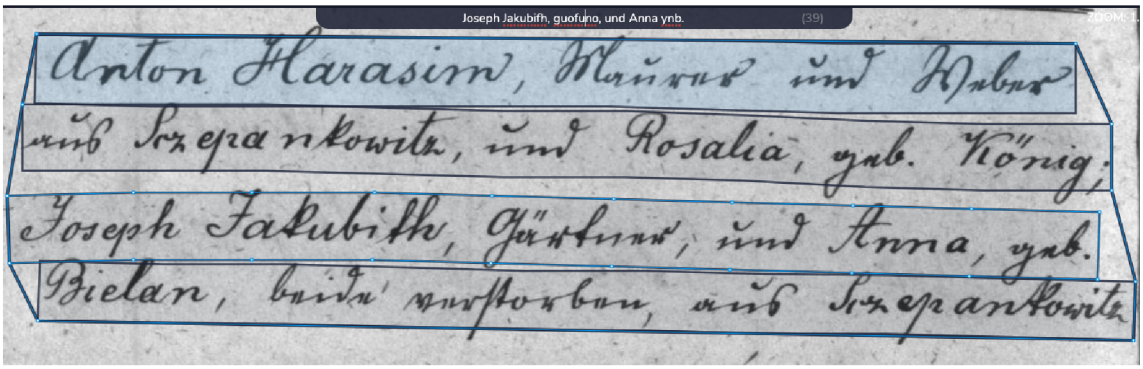
Obrázek 3.8: Ukázka stránky pro anotování jednotlivých skenů datasetu. V ukázce je použit sken matričního záznamu z obce Štěpánkovice z počátku 20. století. Sken získán z digi.archives.cz.



Obrázek 3.9: Ukázka vyznačení pozice řádku pomocí polygonu. Při definování polygonu je rozpracovaný polygon obarven zeleně, aby byl dostatečně viditelný. Jedná se o sken matriky z obce Štěpánkovice z počátku 20. století. Sken získán z digi.archives.cz.

anotace, u kterých byla správnost přepisu potvrzena uživatelem (z těchto anotací se pak učí instance OCR modelů). Dále je pomocí světle modrého ohraničení a zvýraznění bodů vyznačen řádek, který uživatel označil (vybral kurzorem). Vybraný řádek je automaticky nalezen v seznamu anotací (lze vidět na obrázku 3.8 vpravo dole) a přepis tohoto řádku je pak zobrazen v podobě modré lišty v horní části obrazovky (přepisy označených řádků lze vidět také na obrázcích novin 1.11 a na obrázku 3.10). Uživatel může přepis opravit nebo celý přepsat znovu. Pokud je s přepisem spokojený, může potvrdit správnost přepisu daného řádku stisknutím klávesy „Enter“. Veškeré klávesové zkratky a vysvětlivky jsou zobrazeny v dolní části obrazovky. Po označení řádku a kliknutí pravého tlačítka myši je zobrazena nabídka s operacemi smazání anotace, potvrzení správnosti přepisu, popř. zrušení potvrzení správnosti přepisu daného řádku. Pokud uživatel klikl pravým tlačítkem myši nad bodem polygonu, je zobrazena i možnost odstranění tohoto bodu. Body polygonu lze také přemísťovat pomocí držení klávesy „Levý Ctrl“ a přesunu kurzoru.

Zbylá pravá část stránky, zobrazené na obrázku 3.8, je rozdělena vertikálně na dvě poloviny. Vrchní polovina obsahuje seznam skenů daného datasetu s vyznačeným skenem, který je právě uživatelem anotován a tlačítky pro přesun mezi skeny. V dolní části je pak seznam regionů anotovaného skenu, kde každý region obsahuje seznam jeho řádků. Pokud



Obrázek 3.10: Ukázka vyznačeného textového regionu a jeho čtyř řádků. První řádek má uživateli potvrzený přepis. U ostatních třech řádků nebyl přepis zatím uživateli potvrzen. Třetí řádek je označen a jeho dosavadní přepis je proto zobrazen v horní modré liště. V ukázce je použit sken matričního záznamu z obce Štěpánkovice z počátku 20. století. Sken získán z digi.archives.cz.

uživatel klikne v prostřední části na kterýkoliv řádek, je tento řádek, i region do kterého patří, automaticky nalezen v seznamu anotací a naopak.

Implementace anotační komponenty Veškeré zdrojové kódy související s implementací komponenty pro anotování skenů jsou uloženy v adresáři `/app/js/components/annotator`. Hlavní anotační komponenta je uložena v souboru `AnnotatorWrapperComponent.vue`. Tato komponenta obsahuje další komponentu `canvas/AnnotatorComponent.vue`, která obsahuje samotné plátno *canvas* pro zobrazení skenu a jeho anotací, lištu se seznamem anotačních nástrojů a dvě další komponenty: `ImageSelectComponent.vue` realizující zobrazení seznamu skenů v datasetu a jejich výběr pro anotaci (viz obrázek 3.8 vpravo nahoře) a komponenta `AnnotationListComponent.vue` pro zobrazení seznamu regionů a jejich řádků (viz obrázek 3.8 vpravo dole).

Veškerá komunikace směrem od vnořených komponent k hlavní *Wrapper* komponentě probíhá pomocí emitování a odchytávání událostí. Komunikace směrem od vrchních komponent směrem k zanořeným komponentám probíhá pomocí předávání parametrů (*properties*) a volání metod. Prostřední komponenta `AnnotatorComponent` definuje veškeré metody pro „ovládání“ anotační komponenty jako např. načtení obrázku pomocí URL adresy, načtení anotací do komponenty, získání anotací z komponenty, vytvoření nové anotace, smazání anotace atd. Dále pak emituje události vyvolávané uživatelem při práci s anotační komponentou – vytvoření nové anotace, smazání anotace, editace anotace, výběr anotace atd. Hlavní *Wraper* komponenta pak pouze volá metody této `AnnotatorComponent` a odchytává jí emitované události, na které definuje patřičná zpětná volání (*callback*). Součástí emitovaných a ochytávaných údálostí jsou vždy i data (např. informace o změněné anotaci).

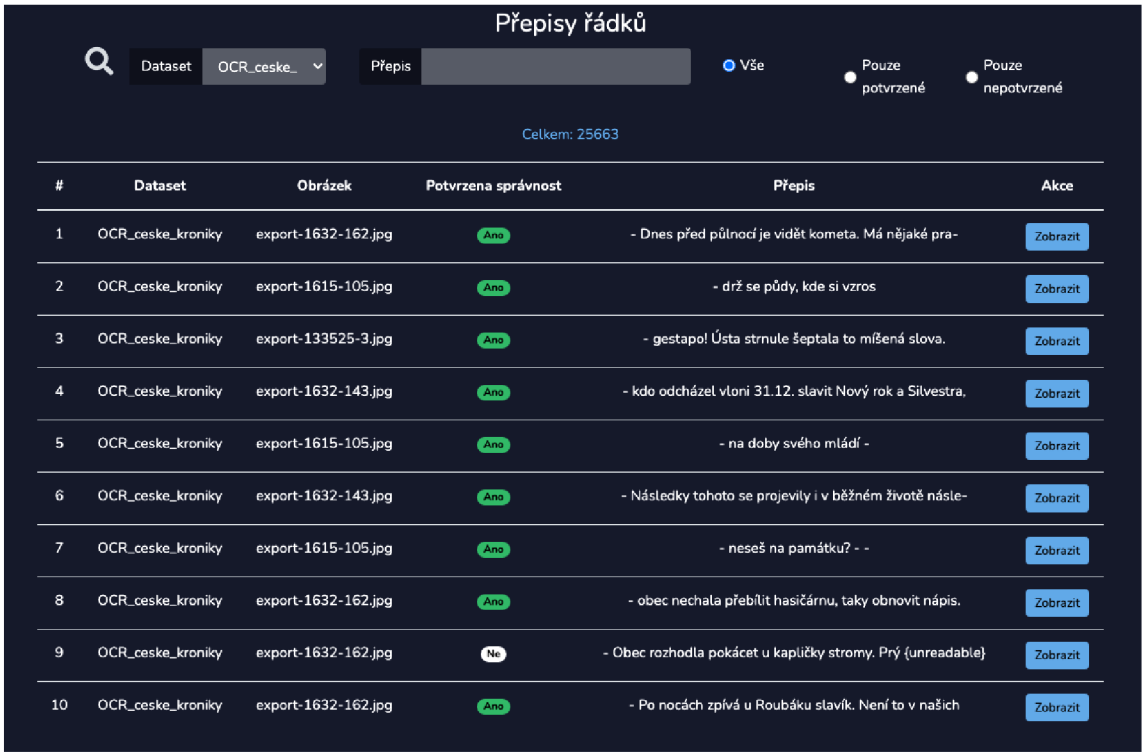
Po načtení stránky si anotační komponenta *Wrapper* načte z URL adresy 3 parametry oddělené lomítkem, které představují *id* datasetu, *id* obrázku a *uuid* konkrétní anotace. Následně načte asynchronně informace o tomto obrázku a dalších obrázcích tohoto datasetu a zobrazí daný obrázek v anotační komponentě pomocí volání metody `canvasSelectImage` na prostřední komponentě. Seznam obrázků datasetu v pravé horní části je naplněn pomocí volání metody `canvasSelectDataset`. Nahrání anotací, které byly získány z databáze, do anotační komponenty je provedeno metodou `loadAnnotations`. Pokud se v url

nacházelo *uuid* konkrétního řádku, je tento řádek automaticky označen pomocí metody `canvasSelectRowAnnotation` (využito v případě, že uživatel vyhledá konkrétní přepis viz dále). Komponenta *Wrapper* také odchytává události smazání, vytvoření, editace anotace, které rovnou promítá do databáze, takže uživatelé nemusí při práci s anotační komponentou nic ukládat.

Během vývoje anotační komponenty byl o tuto komponentu projeven zájem ze strany pana Ing. Michal Hradiš, Ph.D. pro využití v rámci projektu PERO OCR[34]. Komponenta byla tedy vyvíjená tak, aby odpovídala i požadavkům tohoto projektu. Po skončení práce bude tato komponenta tedy využita v rámci projektu PERO.

Stránka pro vyhledávání přepisů

Poslední částí webové aplikace je stránka pro fulltextové vyhledávání mezi přepisy řádků v aplikaci. Tato stránka je dostupná po kliknutí na tlačítko „Vyhledávání přepisů“ v horní navigační liště. Ukázka této stránky je zobrazena na obrázku 3.11. V horní části stránky je

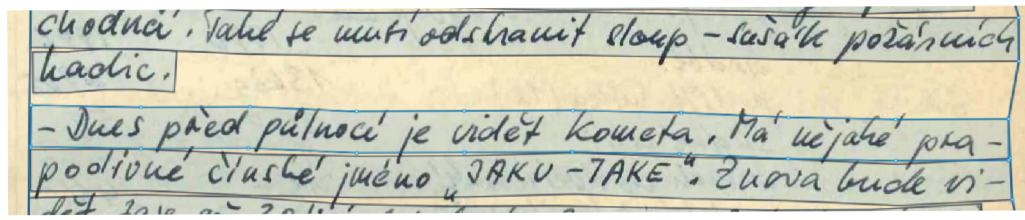


Obrázek 3.11: Ukázka stránky pro fulltextové vyhledávání mezi přepisy řádků v aplikaci.

umístěn filtrační formulář realizovaný komponentou `annotations/FilterComponent.vue`, která využívá abstraktní filtrační komponentu. Uživatelé mohou omezit svá vyhledávání pomocí výběru názvu konkrétního datasetu a konkrétního textu, který musí vyhledávané přepisy obsahovat. Vyhledávat lze také pouze v potvrzených nebo naopak nepotvrzených přepisech řádků. Ve zbytku stránky je zobrazen samotný seznam nalezených přepisů, které splňují filtrační parametry. Seznam je realizován komponentou `annotations/ListComponent.vue`, která využívá abstraktní komponentu pro výpis záznamů. Díky tomu lze opět seznam nalezených záznamů procházet po stránkách. U každého nalezeného přepisu je vypsán název

[34]https://pero-ocr.fit.vutbr.cz

datasetu a obrázku, kde byl tento přepis nalezen, informace, zda byla potvrzena správnost tohoto přepisu uživatelem a samotný přepis. Pro každý přepis je také možné kliknout na tlačítko „Zobrazit“, čímž dojde k zobrazení konkrétního datasetu a obrázku a vyznačení tohoto přepisu pomocí anotační komponenty. Například pokud uživatel klikne na tlačítko „Zobrazit“ v prvním řádku obrázku 3.11, dojde k přesměrování do datasetu „OCR české kroniky“ na konkrétní sken s vyznačením řádku, který je zobrazen na obrázku 3.12.



Obrázek 3.12: Ukázka zobrazení konkrétního přepisu pomocí stránky pro fulltextové vyhledávání mezi přepisy řádků v aplikaci. Tento sken je součástí datasetu „OCR české kroniky“, který mi byl poskytnut v rámci fakulty.

3.3 Testování systému

Tato sekce se bude věnovat testování celého výše popsaného systému. Nejdříve je otestována úspěšnost rozpoznávání vlastnoručně vytvořeného CRNN OCR modelu na zvolených ručně psaných datasetech. Následně je popsáno otestování webové aplikace, která reprezentuje celý systém jako celek. Pro účely testování byla aplikace nasazena na veřejně dostupný server, aby ji mohli uživatelé vyzkoušet a sdělit své připomínky a zpětnou vazbu. Aplikace je dostupná na adrese `http://pcrozman2.fit.vutbr.cz:8080`.

3.3.1 Použité datasety pro vyhodnocení úspěšnosti modelu CRNN OCR

V této sekci budou popsány použité datasety k trénování a otestování modelu pro optické rozpoznávání znaků, poté bude na těchto datasetech natrénován a vyhodnocen OCR model CRNN. Ke všem datasetům jsou dostupné anotace v podobě přepisů textů z jednotlivých obrázků. Pokud obrázky daného datasetu obsahují více řádků textu, jsou dostupné i anotace pro segmentaci jednotlivých řádků.

Dataset OCR Read

Dataset „OCR Read“ [31], který mi byl poskytnut v rámci fakulty, obsahuje 10 043 stran naskenovaného textu a pro všechny skeny stránky jsou k dispozici segmentace jednotlivých řádků s textovými přepisy. Dohromady je k dispozici 171 526 přepsaných řádků textu. Tento dataset je psán německy a obsahuje různé typy psaných písem. Dvě ukázky skenů stran z tohoto datasetu jsou zobrazeny v příloze na obrázku B.1.

Dataset IAM Handwritten database

Dataset „IAM Handwritten database“ [20] obsahuje celkem 1 539 stran naskenovaného ručně psaného textu. Celkem je k dispozici 13 353 obrázků segmentovaných řádků. Ukázky naskenovaných řádků textu jsou v příloze na obrázku B.3. Ke všem řádkům jsou k dispozici textové přepisy.

Dataset OCR české kroniky

Tento dataset byl poskytnut v rámci fakulty (původní zdroj `www.digi.ceskearchivy.cz`). Dataset se skládá ze skenů českých kronik. Celkem je k dispozici 553 naskenovaných stran kronik, které obsahují 25 663 segmentovaných řádků s přepisem textu. Mnoho řádků je bohužel nekompletně přepsaných, proto byly pro mé účely z tohoto datasetu vyloučeny. Celkem je tedy k dispozici 24 814 přepsaných řádků. Ukázka skenu z tohoto datasetu je zobrazena v příloze na obrázku B.4.

Dataset Bentham

Dataset Bentham [8, 32] je standardní dataset pro vyhodnocování úspěšnosti OCR systémů. Dataset obsahuje 443 skenů dokumentů, které obsahují 11 473 segmentovaných řádků s přepisem textu. Texty jsou psány ručně anglickým jazykem, stejným typem písma. Ukázka skenu z tohoto datasetu je zobrazena v příloze na obrázku B.2.

Dataset SOA Zámrsk Dubenec

Tento dataset vznikl při testování webové aplikace amatérskými genealogy, kteří nahráli 20 skenů z matriky SOA Zámrsk Dubenec (staženo z `https://vychodoceskearchivy.cz`) do webové aplikace, nechali automaticky zdetekovat řádky a textové regiony a tyto řádky následně přepsali a potvrdili jejich správnost (s podporou trénování a inference vlastní instance OCR modelu). Dataset obsahuje celkem 4 119 potvrzených přepisů řádků. Texty jsou psány ručně latinským jazykem, stejným typem písma. Ukázka skenu z tohoto datasetu je zobrazena v příloze na obrázku B.5.

3.3.2 Testování úspěšnosti rozpoznávání znaků modelem CRNN OCR

Pro všechny výše uvedené datasety byl natrénován a následně vyhodnocen model CRNN OCR. Řádky datasetů byly rozděleny v poměru 90:10 na trénovací a testovací sadu, jelikož neobsahovaly informaci o tom, zda spadají do trénovací nebo testovací sady. Pro zjištění chybovosti znaků (*character-error*) byla spočtena vzdálenost mezi predikovanými přepisy sítě a reálnými *ground-truth* přepisy v testovací sadě. Vzdálenost je spočtena jako počet operací smazání znaku, přidání znaku, změna znaku na jiný znak, které je nutné provést pro převod jednoho řetězce na druhý řetězec (Levenshteinova vzdálenost). Celková vzdálenost je sečtena pro všechny testovací vzory a následně je vydělena celkovým počtem znaků v testovací sadě, čímž je vypočteno procento znaků, které síť špatně predikovala (*character-error*). Ukázka trénování a vyhodnocení datasetu již byla zobrazena na obrázku 3.2 při popisu skriptu demonstrujícího trénování a vyhodnocení modelu CRNN OCR.

Veškeré trénování probíhalo prostřednictvím webové aplikace, kde byly nejdříve nahrány jednotlivé datasety. Anotace k datasetům „OCR Read“ a „OCR české kroniky“ jsou ve formátu XML. Aby bylo možné nahrát tyto datasety do webové aplikace, byl v aplikaci nejdříve vytvořen nový dataset, do kterého byly nahrány skeny, následně byly naimportovány anotace toho datasetu do databáze pomocí vytvořeného skriptu `parse_load_xml_dataset.py`. Pro akceleraci trénování byly využity dvě grafické karty: NVIDIA GeForce GTX 1080 a NVIDIA GeForce GTX 2080, které jsou dostupné v rámci serveru `pcrozman2.fit.vutbr.cz`.

Model CRNN OCR byl použit v této konfiguraci: konvoluční síť `CRNN2`, která byla popsána v sekci 3.2.4, dále byla použita obousměrná rekurentní síť `GRU` s dimenzemi skrytých vektorů 1 024 (zvýšení snížilo rychlost trénování a nevedlo k lepším výsledkům) a počtem vrstev 2. Počet klasifikovaných tříd je 132 – po konzultaci s amatérskými genealogy byly do znakové sady přidány jimi požadované znaky. Výsledky testování úspěšnosti rozpoznávání znaků jsou zobrazeny v tabulce 3.1.

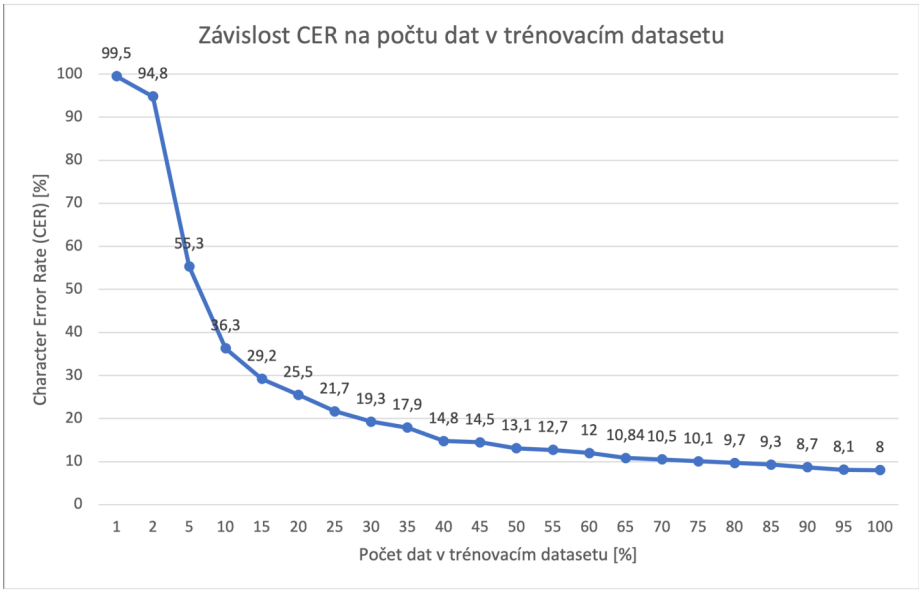
Název datasetu	**Jazyk**	**Trénovací vzory**	**Testovací vzory**	**Správně rozpoznané znaky**	**CER**
OCR Read	Německý	∼154,3k	∼17,1k	∼89%	∼11%
OCR české kroniky	Český	∼22,3k	∼2,5k	∼93%	∼7%
I Am Handwritten Database	Anglický	∼12k	∼1,3k	∼93%	∼7%
OCR Bentham 2014	Anglický	∼10,3k	∼1,1k	∼95%	∼5%
Soa Zámrsk Dubenec	Latinský	∼3,6k	∼400	∼92%	∼8%

Tabulka 3.1: Tabulka zobrazující výsledky testování úspěšnosti rozpoznávání znaků vlastnoručně vytvořeným modelem CRNN OCR.

Dataset „OCR Read“ obsahuje velké množství různých druhů písem a textů, které jsou těžce čitelné, což je pravděpodobně důvod, proč nebylo dosaženo lepších výsledků. Oproti tomu dataset „OCR české kroniky“ je psán česky a většinou velmi úhledným písmem, ale přepisy tohoto datasetu obsahují časté chyby přepisovatelů, které brání lepším výsledkům. Datasety „OCR Bentham 2014“ a „I Am Handwritten Database“ jsou standardní datasety s ručně psanými texty pro vyhodnocování OCR systémů. Na obou těchto datasetech bylo dosáhnuto poměrně nízkého *CER*, které je srovnatelné s výsledky jiných OCR systémů [24, 3], které byly trénovány a vyhodnoceny na těchto datasetech – pro model *CRNN* je u datasetu „I Am Handwritten Database“ dle těchto článků dosaženo *CER* 8-6 % a u datasetu „OCR Bentham 2014“ pak 4-8 %.

Všechny instance CRNN OCR modelu, které vznikly trénováním na výše uvedených datasetech byly přidány do webové aplikace jako globální instance tzn. uživatelé jimi mohou inferovat své datasety a také na základě těchto instancí mohou vytvářet nové instance (váhy budou zděděny).

Experiment s aktivním učením Pro získání představy, kolik dat je potřeba anotovat v konkrétním datasetu, aby se daná instance OCR naučila správně rozpoznávat znaky textu tohoto datasetu, bylo rozhodnuto o provedení následujícího experimentu. Dataset „Soa Zámrsk Dubenec“, který byl vytvořen amatérskými genealogy pomocí tohoto systému, byl rozdělen na trénovací a testovací sadu v poměru 90:10. Trénovací sada byla navíc ze začátku omezena na 1 % vzorků dat (řádků) a postupně rozšiřována o více vzorků až na 100 % (celá trénovací sada obsahující 90 % dat datasetu), což mělo simulovat postupné opravování a potvrzování přepisů nových řádků uživateli systému s následným dotrénováváním instance OCR. Při každém přidání trénovacích dat byl model trénován, dokud se CER nesnižoval. Výsledný průběh CER v závislosti na počtu procent trénovacího datasetu je zobrazen v grafu 3.13.



Obrázek 3.13: Graf závislosti CER na počtu dat v trénovacím datasetu. Z tohoto grafu vyplývá, že pro razantní snížení CER je důležité anotovat alespoň 5-10 % dat což vede ke snížení CER na hodnotu mezi 35-55 %. Pro další snižování CER je nutné přidávat do trénovací sady stále větší množství dat.

Vyhodnocení výpočetní náročnosti pro enormní množství skenů V této části bude popsána výpočetní náročnost trénování a inference modelu CRNN OCR pro velké množství skenů (milióny anotací řádků). Jelikož výpočetní náročnost s přibývajícím množstvím dat roste lineárně, a takto velké množství anotovaných skenů (vyznačených řádků s přepisy) nebylo k dispozici, bylo rozhodnuto vyhodnocení výpočetní náročnosti na největším datasetu, který byl dostupný s následným odvozením výpočetní náročnosti pro větší množství anotací. Z použitých datasetů je největší dataset „OCR Read“, který obsahuje celkem 171 526 anotovaných řádků. Po zmenšení výšky obrázků na fixní velikost 32 pixelů (šířka zmenšena proporcionálně) je průměrná velikost jednokanálového obrázku vyřezaného řádku 10 KB. Velikost paměti dostupné grafické karty GeForce GTX 2080 je 11 GB. Do paměti této grafické karty se tedy teoreticky vejde až 11 GB / 10 KB = 1,1 miliónů obrázků najednou. V paměti grafické karty se ale kromě samotných dat musí také rezervovat místo pro váhy modelu a aktivační mapy. Při trénování je kapacita navíc omezena i rezervací místa pro výpočet gradientů při zpětném průchodu (*Backpropagation*). Počet obrázků, které se v jeden okamžik vejdou do paměti grafické karty tedy bude v praxi (hlavně v případě trénování) výrazně nižší. Pro zvolenou grafickou kartu a dataset „OCR Read“, byla empiricky zjištěna maximální velikost dávky při trénování 32 obrázků a při inferenci pak 256 vzorů dat. Vyhodnocení testovací sady o velikosti 17 152 vzorů dat pak trvalo přibližně 71 sekund, což odpovídá $241,5$ vyhodnoceným vzorům dat za sekundu. Inference 1 miliónu řádků by tedy trvala přibližně $1,15$ hodiny. Trénování modelu CRNN OCR pro jednu epochu na 154 373 vzorech dat trvalo $35,88$ minuty, což odpovídá $71,7$ obrázkům za sekundu. Jedna epocha trénování modelu pro 1 milión řádků by tedy trvala přibližně $3,87$ hodiny. Průměrně se v jedom skenu matriční knihy nachází 100 řádků textu (změřeno v archivu MZA). V jednom miliónu skenů se tedy nachází přibližně 100 miliónů řádků. Inference jednoho miliónu skenů by tedy na uvažované grafické kartě trvala 115 hodin. Pokud by ke všem těchto řádkům byly dostupné přepisy, trénování jedné epochy pro milión skenů s využitím této jediné grafické karty by pak trvalo 387 hodin. Pro natrénování je ale potřeba model trénovat po dobu více epoch.

3.3.3 Testování webové aplikace

Jak již bylo řečeno, pro účely testování byla aplikace nasazena na veřejně dostupný server http://pcrozman2.fit.vutbr.cz:8080, aby ji mohli uživatelé otestovat (a s ní tak celý systém). Po nasazení aplikace na veřejně dostupný server zveřejnil vedoucí práce odkaz na tuto aplikaci na genealogickém fóru http://genealogie.taby.cz, kde se vyskytují nejen amatérští genealogové. Díky tomuto zveřejnění se do aplikace zaregistrovalo 8 lidí, kteří aplikaci vyzkoušeli. Z těchto osmi lidí se dva lidé začali významně podílet na testování aplikace – postupně v aplikaci založili několik datasetů, kde nahráli skeny matričních záznamů. Na těchto datasetech pak testovali funkčnost služeb pro automatickou detekci řádků a textových regionů a trénování/inferenci instancí OCR modelů. Pro každý nahraný dataset byly vytvořeny vlastní instance OCR modelů (CRNN i Tesseract), kterými uživatelé inferovali své datasety, čímž byly navrženy přepisy řádků u kterých zatím nebyla potvrzena správnost přepisu. Uživatelé poté opravovali a potvrzovali správnost těchto přepisů a na takto rozšířené trénovací sadě znovu trénovali své instance OCR modelů. Díky postupnému doučování OCR modelů bylo nakonec uživateli celkem přepsáno a potvrzeno přes 5 tisíc řádků textu v jimi nahraných matričních záznamech.

Během celé doby testování (cca měsíc a půl) uživatelé psali své připomínky a návrhy na vylepšení na zmíněné genealogické fórum, kde jsme o aplikaci diskutovali, popř. mě

kontaktovali pomocí e-mailu. Postupně tak na přání uživatelů do aplikace přibyly další funkcionality jako smazání datasetů a jednotlivých obrázků z datasetu, vyznačení nových anotací pomocí polygonů, mazání anotací, editace anotací, smazání bodů anotací, potvrzování přepisů řádků pomocí klávesových zkratek, přechod mezi anotacemi řádků pomocí klávesových zkratech a mnoho dalších. Dále byly díky uživatelům nalezeny a opraveny chyby jako nefunkčnost přibližování skenů v některých prohlížečích, nemožnost mazání některých anotací, nemožnost nahrání více než 100 obrázků do datasetu atd. Pro účely testování byl také vytvořen anonymní dotazník[35], jehož odkaz byl umístěn na hlavní stránce aplikace. Dotazník obsahuje uzavřené i otevřené otázky, které jsou zaměřené na základní funkcionality systému jako vytváření a mazání datasetů, nahrávání obrázků, spouštění úkolů nad datasety, zobrazení anotací atd. Většina respondentů odpovídala na otázky pozitivně – funkčnost jednotlivých akcí jim byla zřejmá. Některým respondentům např. nebylo ihned jasné, jak nahrávat obrázky k datasetům, proto byl proces nahrávání zjednodušen a více popsán.

Veškeré připomínky osob testujících aplikaci byly brány v úvahu. Připomínky, které bylo možné v rozumném čase vyřešit, byly vyřešeny, ostatní připomínky, které buďto nemohly být vyřešeny (např. vylepšení automatické detekce řádků a textových regionů) nebo by jejich řešení zabralo velké množství času, na který už nebyl prostor, jsou poté uvedeny v návrzích pro rozšíření této práce.

Návrhy na rozšíření této práce

- Export anotací ze systémů.
- Změna pořadí nahraných skenů v datasetu.
- Roztřídění datasetů podle přiřazených značek (*tags*).
- Trénování modelu pro automatickou detekci rozložení stránky.
- Provádění úkolů (trénování/inference/detekce) nejen nad celými datasety, ale také zvlášť nad jednotlivými skeny.
- Dekódování výstupu neuronové sítě CRNN pomocí algoritmu Word Beam Search, namísto Greedy Search.

[35] https://docs.google.com/forms/d/e/1FAIpQLSdK58VSRLuDhZ2_0H1sySNTTDkPC3I7lp88fiCe7zcs-m8DTw/viewform

Kapitola 4

Závěr

Cílem této práce bylo nastudovat různé přístupy pro rozpoznávání ručně psaného textu a metodu Active Learning. Dále pak navrhnutí systému, který bude schopen využívat různé implementace OCR modelů – převzatý a vlastnoručně vytvořený OCR model. Následně navrhnout způsob doučování těchto OCR modelů ze záznamů opravených uživateli systému. Navržený systém bylo nutné implementovat, včetně automatického doučování ze záznamů opravených uživateli. Systém měl také umožnovat fulltextové vyhledávání v rozpoznaných datech. Poslední částí bylo otestování systému, výpočetní náročnosti pro velké množství skenů a úspěšnosti rozpoznávání.

Všechny požadované body zadání byly splněny – v první kapitole byly nejdříve popsány různé přístupy pro rozpoznávání znaků, následně byla uvedena klasifikace OCR systémů a představen již existující systém pro rozpoznávání znaků. Následoval popis kroků procesu rozpoznávání znaků, včetně často využívaných algoritmů v jednotlivých krocích. Další sekce se zabývala systémy pro rozpoznávání znaků pomocí neuronových sítí. Byly zde uvedeny tři, dnes často využívané, architektury neuronových sítí pro rozpoznávání znaků. Architektura CRNN byla popsána detailněji, jelikož byla využita pro vlastnoručně vytvořený OCR model, který byl použit ve výsledném systému. Zbytek kapitoly byl věnován Aktivnímu učení (*Active Learning*), kde byly vysvětleny 3 různé varianty aktivního učení a jejich využití. V kapitole „Analýza požadavků a návrh řešení“ byla navržena architektura výsledného systému, která je založena na architektuře mikroslužeb. Také byl navržen způsob doučování modelů z opravených záznamů a schéma databáze, která slouží pro uchování dat systému.

V kapitole „Implementace a testování“ byla nejdříve popsána implementace celého systému, který se skládá z pěti komponent. Jako první byly popsány jednotlivé moduly systému, poté API pro kontejnery, které vykonávají úkoly nad datasety, služba pro automatickou detekci řádků a textových regiónů, služba pro trénování a inferenci převzatého systému Teserract OCR, služba pro trénování a inferenci vlastnoručně vytvořeného modelu CRNN OCR a jako poslední webová aplikace, která slouží jako rozhraní všech těchto modulů. Zbytek kapitoly byl věnován testování. Nejdříve byly představeny datasety pro testování, včetně datasetu „SOA Zámrsk Dubenec“, který byl vytvořen amatérskými genealogy při testování systému. Poté byl na těchto datasetech otestován vlastnoručně vytvořený model CRNN OCR, zhodnoceny výsledky a výpočetní náročnost pro velké množství skenů. Jako druhá byla testována webová aplikace, a tedy s ní i celý systém jako celek. Pro umožnění testování aplikace byl systém nasazen na veřejně dostupném serveru. Systém následně po dobu jeden a půl měsíce aktivně testovali amatérští genealogové, kteří také navrhli náměty na případná budoucí vylepšení této práce. Jelikož tito amatérští genealogové požadují fungování aplikace i po odevzdání této diplomové práce, je zvažován další vývoj tohoto systému.

Literatura

[1] *Tesseract* [online]. [cit. 2020-12-28]. Dostupné z: https://en.wikipedia.org/wiki/Tesseract_(software).

[2] *Text Digitalisation: Computing error rates* [online]. [cit. 2021-04-25]. Dostupné z: https://sites.google.com/site/textdigitisation/qualitymeasures/computingerrorrates.

[3] Abdallaha, A., Hamadab, M. a Nurseitovc, D. *Attention-based Fully Gated CNN-BGRU for Russian Handwritten Text* [online]. 2021 [cit. 2021-04-25]. Dostupné z: https://arxiv.org/pdf/2008.05373.pdf.

[4] Agarap, A. F. M. *Deep Learning using Rectified Linear Units (ReLU)* [online]. [cit. 2021-04-26]. Dostupné z: https://arxiv.org/pdf/1803.08375.pdf.

[5] Batra, D. *Discriminative vs Generative models* [online]. 2016 [cit. 2020-12-28]. Dostupné z: https://deveshbatra.github.io/Generative-vs-Discriminative-models/.

[6] Bishop, C. M. *Pattern Recognition and Machine Learning*. 3. vyd. Springer Science+Business Media, 2006. ISBN 978-0387-31073-2.

[7] Cohn, D., Atlas, L. a Ladner, R. Improving Generalization with Active Learning*. [online]. [cit. 2020-12-30]. Dostupné z: https://link.springer.com/content/pdf/10.1007/BF00993277.pdf.

[8] Gatos, B., Louloudis, G., Causer, T., Grint, K., Romero, V. et al. *Ground-truth production in the tranScriptorium project* [11th IAPR International workshop on document analysis systems (DAS)]. 2014 [cit. 2021-04-21].

[9] Gewers, F. L., Ferreira, G. R., Arruda, H. F. de, Silva, F. N., Comin, C. H. et al. *Principal Component Analysis: A Natural Approach to Data Exploration* [online]. 2018 [cit. 2020-12-28]. Dostupné z: https://arxiv.org/pdf/1804.02502.pdf.

[10] Graves, A., Fernández, S., Gomez, F. a Schmidhuber, J. *Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks* [online]. [cit. 2020-12-20]. Dostupné z: https://www.cs.toronto.edu/~graves/icml_2006.pdf.

[11] Grüning, T., Leiferta, G., Strauss, T., Michael, J. a Labahn, R. *A Two-Stage Method for Text Line Detection in Historical Documents* [online]. [cit. 2020-12-20]. Dostupné z: https://arxiv.org/pdf/1802.03345.pdf.

[12] Hannun, A. Sequence Modeling with CTC. *Distill*. 2017. DOI: 10.23915/distill.00008. Dostupné z: https://distill.pub/2017/ctc.

[13] Islam, N., Islam, Z. a Noor, N. *A Survey on Optical Character Recognition System* [online]. 2016 [cit. 2020-12-21]. Dostupné z: https://arxiv.org/pdf/1710.05703.pdf.

[14] Jaderberg, M., Simonyan, K., Vedaldi, A. a Zisserman, A. *Reading Text in the Wild with Convolutional Neural Networks* [online]. 2014 [cit. 2020-12-29]. Dostupné z: https://arxiv.org/pdf/1412.1842.pdf.

[15] Kingma, D. P. a Ba, J. *Adam: A Method for Stochastic Optimization* [online]. 2014 [cit. 2021-04-19]. Dostupné z: https://arxiv.org/abs/1412.6980.

[16] Kišš, M., Hradiš, M. a Kodymn, O. *Brno Mobile OCR Dataset* [online]. 2019 [cit. 2020-12-29]. Dostupné z: https://arxiv.org/abs/1907.01307.

[17] Kodym, O. a Hradiš, M. *Page Layout Analysis System for Unconstrained Historic Documents* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://arxiv.org/pdf/2102.11838.pdf.

[18] Konyushkova, K., Raphael, S. a Fua, P. Learning Active Learning from Data. [online]. [cit. 2020-12-30]. Dostupné z: https://papers.nips.cc/paper/2017/file/8ca8da41fe1ebc8d3ca31dc14f5fc56c-Paper.pdf.

[19] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, sv. 25, s. 1097–1105. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[20] Marti, U. a Bunke, H. The IAM-database: An English Sentence Database for Off-line Handwriting Recognition.

[21] MEMON, J., SAMI, M., KHAN, R. A. a UDDIN, M. *Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)* [online]. 2020 [cit. 2020-12-21]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9151144.

[22] Ng, A. Y. a Jordan, M. I. *On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes* [online]. [cit. 2021-04-25]. Dostupné z: http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf.

[23] Nováčík, I. T. *Rekurentní neuronové sítě pro rozpoznávání řeči* [online]. 2016 [cit. 2021-04-21]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=132425.

[24] Potanin, M., Dimitrov, D., Shonenkov, A., Bataev, V., Karachev, D. et al. *Digital Peter: Dataset, Competition and Handwriting Recognition Methods* [online]. 2021 [cit. 2021-04-25]. Dostupné z: https://arxiv.org/pdf/2103.09354.pdf.

[25] Satti, D. A. *Offline Urdu Nastaliq OCR for Printed Text using Analytical Approach* [online]. 2013 [cit. 2020-12-21]. Dostupné z: http://ww.cle.org.pk/Publication/theses/2013/danish-thesis.pdf.

[26] Setiadi, I. *Damerau-Levenshtein Algorithm and Bayes Theorem for Spell Checker Optimization*. 2013 [cit. 2021-04-26].

[27] Settles, B. Active Learning Literature Survey. [online]. [cit. 2020-12-30]. Dostupné z: http://burrsettles.com/pub/settles.activelearning.pdf.

[28] Sharma, O. P., Ghose, M. K., Shah, K. B. a Thakur, B. K. *Recent Trends and Tools for Feature Extraction in OCR Technology* [online]. 2013 [cit. 2020-12-28]. Dostupné z: https://www.researchgate.net/publication/261930093_Recent_Trends_and_Tools_for_Feature_Extraction_in_OCR_Technology.

[29] Shi, B., Bai, X. a Yao, C. *An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition* [online]. 2015 [cit. 2020-12-21]. Dostupné z: https://arxiv.org/pdf/1507.05717.pdf.

[30] Staudemeyer, R. C. a Morris, E. R. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks* [online]. 2019 [cit. 2021-04-21]. Dostupné z: https://arxiv.org/abs/1909.09586.

[31] Sánchez, andreu, J., Romero, Verónica, Toselli et al. *Dataset for ICDAR2017 Competition on Handwritten Text Recognition on the READ Dataset (ICDAR2017 HTR) [Data set]. Zenodo* [online]. 2017 [cit. 2021-04-20]. Dostupné z: http://doi.org/10.5281/zenodo.835489.

[32] Sánchez, J., Romero, V., Toselli, A. a Vidal, E. *ICFHR2014 Competition on Handwritten Text Recognition on the tranScriptorium Datasets (HTRtS)* [in 14th International Conference on Frontiers in Handwriting Recognition (ICFHR)]. 2014 [cit. 2021-04-21].

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. *Attention Is All You Need* [online]. 2017 [cit. 2020-12-29]. Dostupné z: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[34] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y. a Girshick, R. *Detectron2* [online]. 2019 [cit. 2020-12-26]. Dostupné z: https://github.com/facebookresearch/detectron2.

Příloha A

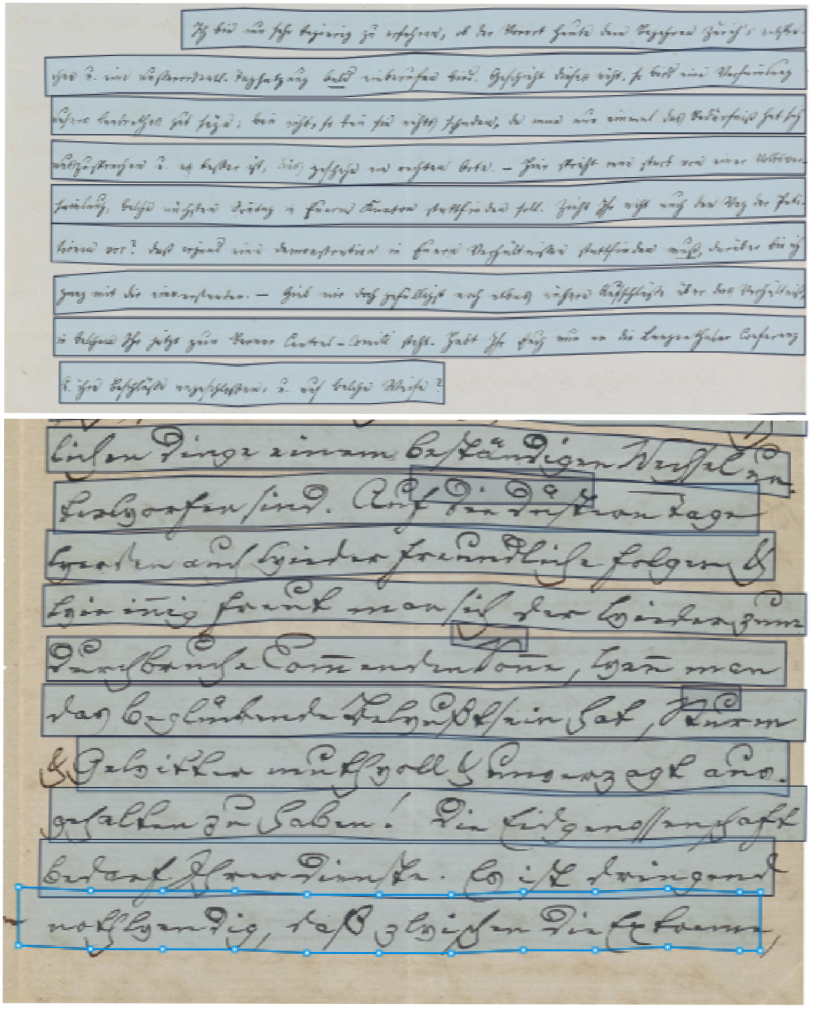
Obsah přiloženého paměťového média

- `/app` – Zdrojové kódy webové aplikace.
- `/docker` – Zdrojové kódy pro definování obrazů a kontejnerů.
- `/doc` – Zdrojové kódy pro vysázení technické zprávy.
- `/doc/xhribe02-DP.pdf` – Technická zpráva ve formátu PDF.
- `/ocr` – Zdrojové kódy vlastnoručně vytvořeného CRNN OCR modelu.
- `/modules_api` – Zdrojové kódy API pro kontejnery vykonávajícími úkoly nad datasety.
- `/storage` – Sdílené uložiště pro uložení nahraných obrázků, instancí natrénovaných OCR modelů a dočasných souborů pro trénování.
- `parse_load_xml_dataset.py` – Skript pro import anotací ve formátu XML do databáze.

Příloha B

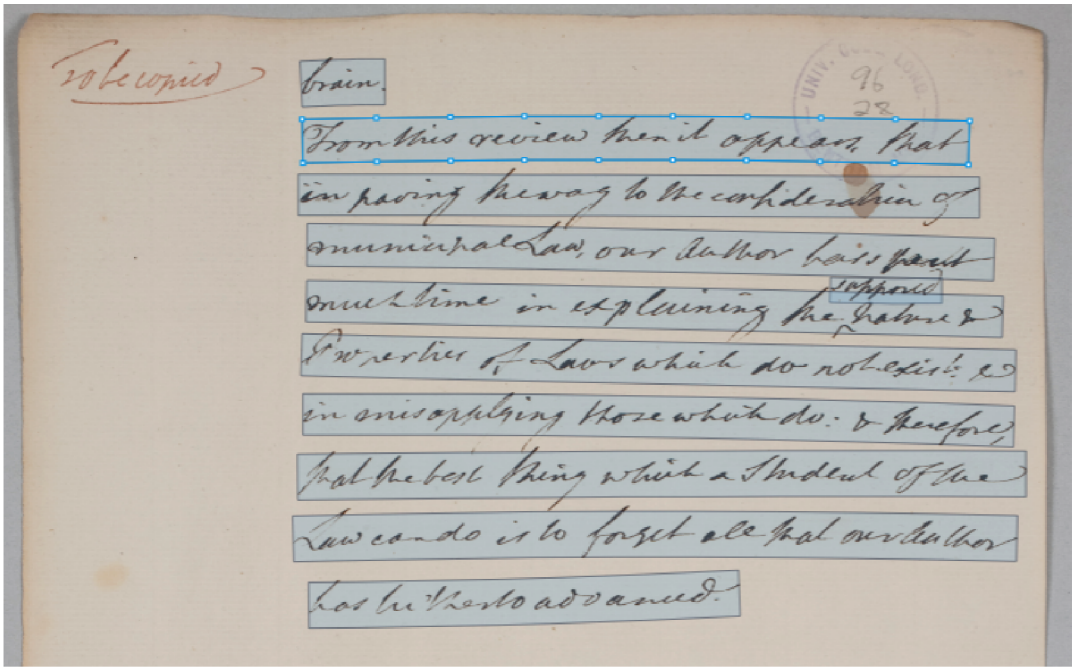
Ukázky skenů z použitých datasetů

Dataset OCR Read



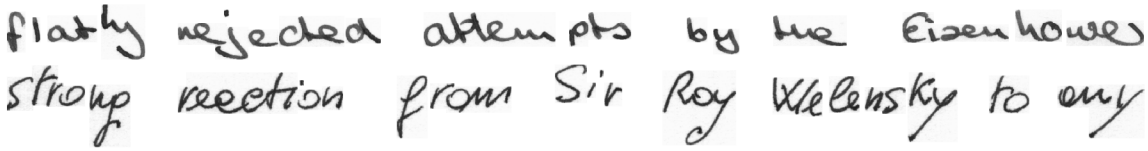
Obrázek B.1: Ukázka skenů záznamů z datasetu „OCR Read“ [31], který byl poskytnut v rámci fakulty. Dataset byl nahrán do webové aplikace – anotace skenů jsou zobrazeny pomocí anotační komponenty webové aplikace.

Dataset Bentham



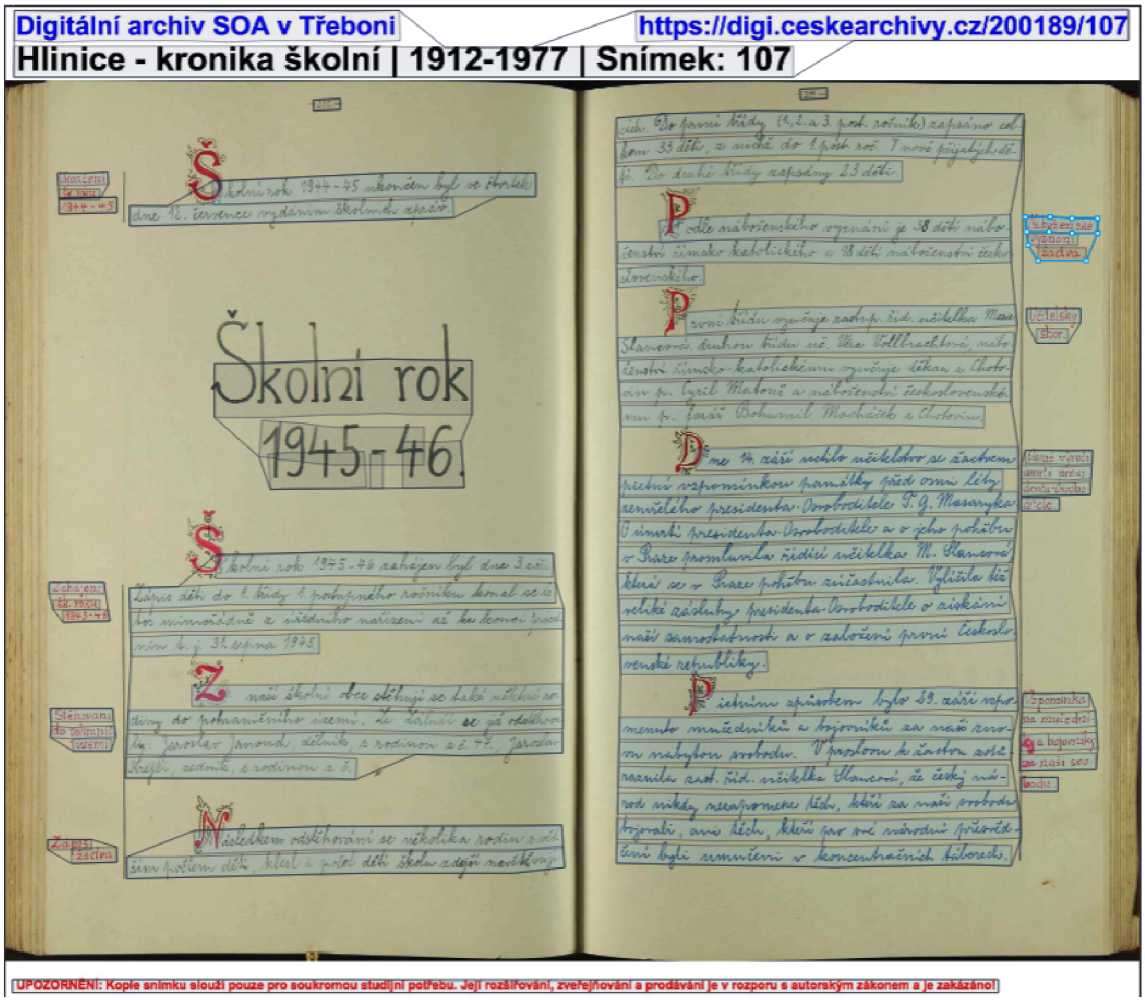
Obrázek B.2: Ukázka skenu dokumentu z datasetu „Bentham“ [8, 32]. Dataset byl nahrán do webové aplikace – anotace skenů jsou zobrazeny pomocí anotační komponenty webové aplikace.

Dataset IAM Handwritten database



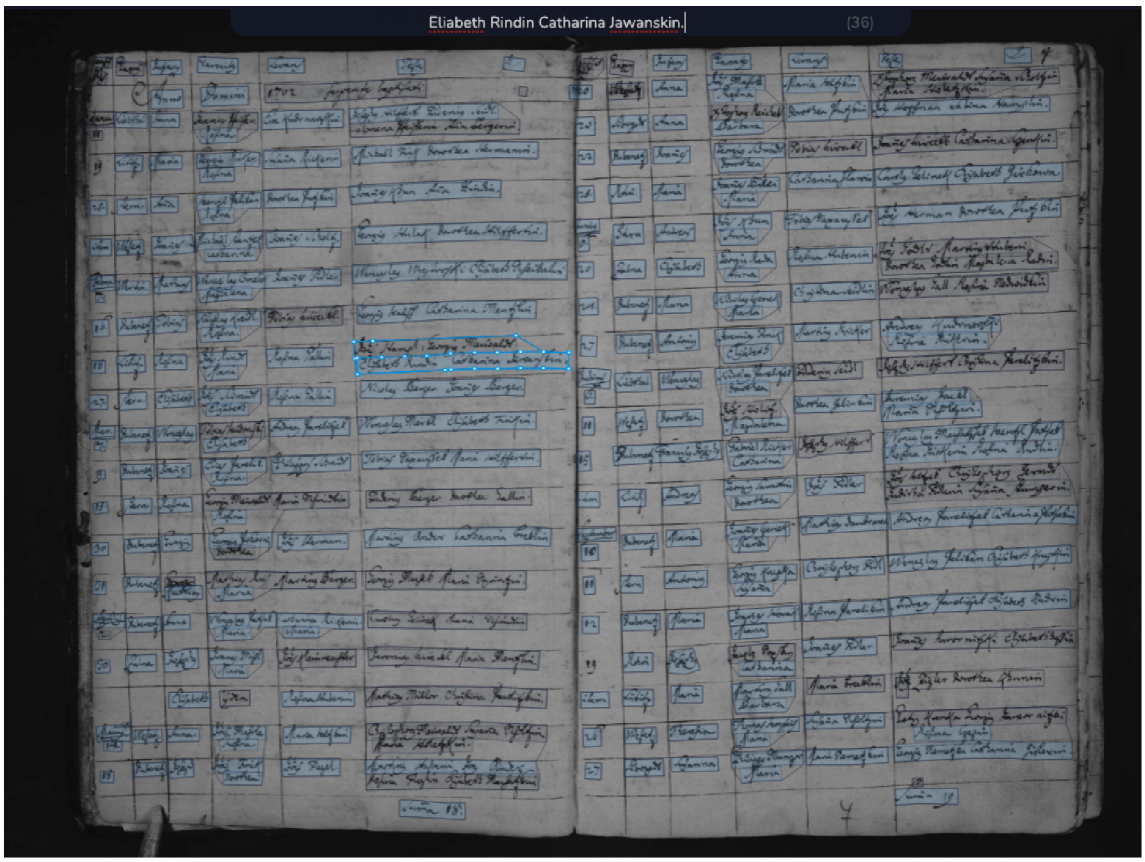
Obrázek B.3: Ukázka naskenovaných řádků textu z datasetu „IAM Handwritten database“ [20]. Každý řádek je samostatný obrázek.

Dataset OCR české kroniky



Obrázek B.4: Ukázka naskenovaných řádků textu z datasetu „OCR české kroniky“, který mi byl poskytnut v rámci fakulty. Dataset je dostupný v digitálním archivu SOA v Třeboni. Dataset byl nahrán do webové aplikace – anotace skenů jsou zobrazeny pomocí anotační komponenty webové aplikace.

Dataset Soa Zámrsk Dubenec



Obrázek B.5: Ukázka naskenovaných řádků textu z datasetu „SOA Zámrsk Dubenec“. Dataset vznikl při testování webové aplikace amatérskými genealogy, kteří jej sami anotovali prostřednictvím této aplikace. Anotace skenů jsou zobrazeny pomocí anotační komponenty webové aplikace.