VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ONLINE NÁSTROJ PRO ROZPOZNÁVÁNÍ TABULEK V OBRÁZCÍCH

ONLINE TOOL FOR RECOGNITION OF TABLES IN IMAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **BOHDAN INHLIZIIAN**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

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

Zadání bakalářské práce

Student: **Inhliziian Bohdan**
Program: Informační technologie
Název: **Online nástroj pro rozpoznávání tabulek v obrázcích**
Online Tool for Recognition of Tables in Images
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou zpracování obrazu a rozpoznávání textu a dalších prvků v 2D obrazu.
2. Vyhledejte a popište existující nástroje pro rozpoznání textů uspořádaných do tabulek.
3. Získejte / pořizujte datovou s obrázkami tabulek, opatřenými vhodnými anotacemi.
4. Experimentujte s algoritmy potřebnými pro rozpoznání tabulek a textů v nich.
5. Navrhněte a implementujte softwarový nástroj pro rozpoznání tabulky v obrázku. Vyhodnocujte vytvořené řešení na pořízené datové sadě a iterativně je vylepšujte.
6. Navrhněte a implementujte webovou službu, která umožní nahrát obrázek tabulky a získat její rozpoznanou podobu ve vhodném formátu (formátech).
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Jan Řezáč: Web ostrý jako břitva, Baroque Partners, 2014
- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 až 5.

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Cílem této práce je řešit problém rozpoznávání tabulek v obrázcích a převést vyfocenou tabulku, nahranou na webové rozhraní, do XLSX souboru. Program je vytvořeny s důrazem na jednoduchost v použití potenciálním uživatelem.

Pro detekce čar byl použit algoritmus Probablistic Hough Transform a pomoci nástrojů Tesseract byla provedena detekce textu v buňkách. Program byl umístnění na Amazon AWS a přístup k němu webová aplikace děla pomoci API. Byl vytvořen vlastní algoritmus pro spojeni čar do jedné čáry a taky algoritmus pro odstranění čar, které nepatří do tabulky a chybně detekovaných čar (text, šum).

Vytvořené řešení poskytuje možnost uživatelům, které ručně přepisují data z tabulek v dokumentech, knihách, využít program, který děla všechno automaticky, je potřeba jen nahrát foto do webové aplikace.

Abstract

This work solves the problem of recognising the tables in the figures. The goal is to convert the table into an XLS file thought web application.

For line detection we have used the Probablistic Hough Transform algorithm and Tesseract tool was used to detect text in cells. The program was stored to the Amazon AWS and accessed by the web app using the API. An algorithm for line merging has been created, as well as an algorithm for removing lines that do not belong to the table and removing wrong detected lines (text, noise).

The solution provides users who manually overwrite data from tables in documents, books, use a program that does everything automatically, you only need to upload photos to a web application.

Klíčová slova

Rozpoznávač tabulek, detekce tabulek, konverze tabulek, Hough Transform, detektor úhlů, extrakce textu z tabulek

Keywords

Table recognition, table detector, convert table to XLSX, extrakt text from table, Hough Transform.

Citace

INHLIZIIAN, Bohdan. *Online nástroj pro rozpoznávání tabulek v obrázcích.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Online nástroj pro rozpoznávání tabulek v obrázcích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bohdan Inhliziian
31. července 2019

Poděkování

Chtěl bych velmi poděkovat svému vedoucímu prof. Ing. Adamu Heroutovi, Ph.D. za rady, které velmi pomohli dosáhnout cíle při práci nad tímto projektem. Taky bych chtěl poděkovat Bc. Kateryně Samsonove za gramatickou kontrolu práce a přátelům za podporu.

Obsah

Kapitola 1

Úvod

Díky rozvoji internetu můžeme vidět více a více různých on-line instrumentů, které usnadňují život. Není nutné instalovat Photoshop, abychom mohli otočit obrázek na devadesát stupňů. Není nutné ručně formátovat XML dokument, abych byl čitelný. Hodně úkolů se dá řešit pomoci dosavadních on-line instrumentů. Tato práce věnuje implementace nástrojů, který bude prospěšný pro lidi, které často přepisují data z vytištěných tabulek do XLSX souboru. Cílem je vytvoření programu, který přes webové rozhraní přijímá na vstupu nahrané uživatelem obrázky vyfocených tabulek, na výstupu stahuje upravenou tabulku v elektronické podobě, zapsanou v XLSX souboru s uloženými řádky, sloupci a texty. Detailně se bude zabývat tímto problémem kapitola číslo 1.

Skoro každý nastroj pro zpracováni obrázku potřebuje datovou sadu. Popisu vytvořeni datové sady bude zabývat kapitola 2.

V kapitole 3 se nachází popis technické specifikace práce, detaily vývojů a implementace potřebných algoritmu algoritmu.

Pro vytvoření kvalitního a fungujícího instrumentu byla potřeba kvalitního testovaní, výsledky a detaily kterého jsou popsané v kapitole 4.

Kapitola 2

Detekce a rozpoznávaní tabulek v obrazcích

V této kapitole se popisuje problém, který se řeší v dané práci, a to detekce a rozpoznávaní tabulek v obrazcích. Zde je popsáno, co vlastně je rozpoznávaní tabulek a proc automatizace práce je důležitá pro člověka. Taky jsou popsané existující algoritmy rozpoznávaní čar a textu, detekce úhlů a normalizace vstupního obrazu, existující aplikace a řešeni v oblasti detekce tabulek v obrazcích.

2.1 Problematika

Detekce a rozpoznávaní různých objektu(tabulek, textu, čaaratd.) na obrazu je jednou z úloh počítačového viděni a tento problém má velmi důležitý praktický vyznám. Detekce objektu na video a obrazu se v současném světe používá v různých oblastech: medicina, bezpečnost, věda, hospodářství, počítačové hry apod. Případy můžou být například detekce rakových buněk na snímku MRT nebo detekce mikroorganizmu na medicínských mikroskopech, vypočítaní další cesty u samořídicích aut, detekce obličeje na letištích a vojenských objektech, rozpoznáváni rostlin a houbu a mnoho dalších.

Objekty, které je potřeba rozpoznat, jsou složité grafické struktury, které se nedá charakterizovat každý stejné. Složité obrázky se dá různými algoritmy rozdělit na primitivy (čáry, elipsy, hrany).

Problémy jako jsou okluze, nízké rozlišeni obrazu, deformace, slabé nebo silné osvětleni, rotace a perspektiva, rozmazanost, můžou mít špatný vliv na kvalitu rozpoznávaní různých objektů na obrazu. Taky neočekávaný vliv můžou mít obrázky různé barvy a s různou tloušťkou či délkou čar. Vzhledem na výše uvedené komplikace, dosažení cíle může vyžadovat přípravu a normalizace obrázku či častí obrázku k nějaké standardní podoby, specifikované pro vstup do aplikace.

Oblasti dané práce je zpracovaní fotografie dokumentu. V současné době nelze představit situace jako například ručně přepsaní PDF dokumentu do DOC či ODT. Je víc a víc potřeba v automatizace práce a ekonomii času. A pro dosažení cíle v jednotlivém a okamžitém případe nechceme instalovat náročný pro počítač a náš čas software. Proto jsou vytvořené spousta různých on-line instrumentů konverze jednoho formátu dokumentu do jiného, rozpoznávaní textu na obrázku a převedení ho do DOC formátu.

Tématem práce je rozpoznávaní tabulek v obrazcích a převedení je do vhodné pro čtení a vnesení změn podoby. Konkretně, detekce tabulky na obrázku, počítaní počtu sloupců

a řádků, detekce které buňky jsou spojené horizontálně a které vertikálně, rozpoznávaní textu z každé buňky. Přísti krok je převedení rozpoznané tabulky do XLS souboru. Zatím, vytvoření webového uživatelského rozhraní pro nahrávaní obrázků s tabulkou a stažení výsledného souboru. Vetší praktická část práce spočívá v navržení a implementaci algoritmů pro klasifikace a zpracovaní různých primitiv [10], ze kterých se skládá tabulka. Problém klasifikace primitiv je částečné postaveny na nalezení formy objektu. Například, pro identifikace modelu a značku auta na obrázku je potřeba detekovat a zvýraznit svítidla, emblému, okna atd. Pomoci používané v tomto případe aproximace svítidel elipsem, oken auta čtvrtečkami, dá se potom obdržet popis těchto primitiv, které potom se uchovávají do databáze. V rámci dané práce je potřeba detekovat čáry a čtverečky, ale kvůli různým geometrickým transformacím je nutné taky udělat jejich aproximace, která bude maximálně odpovídat ideálnímu stavu. Detekce takových primitiv jako čára či čtvereček vyžaduje jejich lokalizaci a detekci velikosti a taky jejich závislost na jiných objektech (čára, text).

Problémem rozpoznávaní tabulky v obrázku jsou objekty a primitivy, které k tabulce nepatři a musí byt vynechané. Je potřeba oddělit tabulku od textu nebo jiných objektu. Jsou dvě možnosti řešeni tohoto problému:

- manuální nastavení oblasti, do které patří tabulka,
- automatický detektor hranic tabulky.

V prvním případe je potřeba nutit uživatele, aby on manuálně lokalizoval tabulku, ručním rozšířením okénka. Tento přistup je náročný, protože aplikace umožňuje nahrávat nekonečně mnoho tabulek pro rozpoznávaní a je potřeba pro každou tabulku dělat ruční lokalizace. Z pohledu uživatelského rozhraní tento přistup je špatný.

Proto bylo nutně udělat automaticky detektor hranic tabulky s využitím neuronové síti.

Pro řešeni daného problému je potřeba nastudovat existující aplikace, algoritmy a metody, které pomůžou dosáhnout požadované cíle. Tím se bude zabývat zbytek kapitoly.

2.2 Nastudovaná teorie

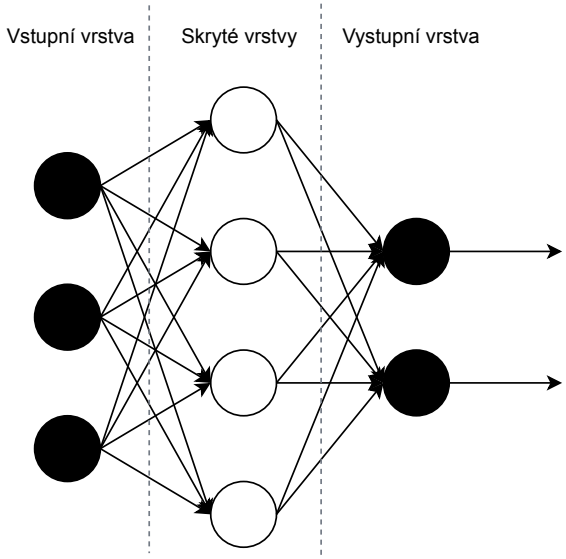
2.2.1 Neuronové sítí

Pro rozpoznávaní hranic tabulky byl implementován automaticky detektor úhlů tabulky. Detektor je postaven na využití neuronových sítí.

Neuronová síť je posloupnost neuronu, spojených mezi sebou pomoci synapsí [8]. Struktura neuronové sítí v programování je převzatá z biologie. Pomoci neuronových sítí počítač může provádět různou analýzu vstupní informace. Jinými slovy, je to interpretace mozku člověka.

Problémy, které můžou být řešené pomoci neuronových sítí jsou:

- **Klasifikace** – klasifikace dat v závislostí od jejich vlastnosti. Například analýza obrázku a klasifikace objektu na něm (pes, kočka apod).
- **Předpověď** – předpověď příštího kroku. Například uměla inteligence v počítačové hře.
- **Rozpoznávaní** – rozpoznávaní objektu. Příkladem může být rozpoznávaní obličeje na obrázku.



Obrázek 2.1: Architektura vícevrstvé neuronové síti.

Neuron

Neuron je základní prvek neuronové síti. Je to výpočetní jednotka, která provádí základní výpočty nad vstupními data a předává výsledek dále. Jestli neuronová síť se skládá z velkého počtu neuronu, její struktura se rozdělí na vrstvy: vstupní vrstva (obdrženi informace), skryté vrstvy (zpracovaní informace), výstupní vrstva (výstup) 2.2. Každý neuron má dva porty: vstup a výstup. V případě vstupní vrstvy, vstup neuronu je ekvivalentní jeho výstupu. V případe ostatních vrstev, na vstup do neuronu přijde sumární informace všech neuronu z minulé vrstvy. Neuron provede výpočet pomoci aktivační funkce f(x) a výsledek vrátí na vystup. Aktivační funkce je způsob normalizace vstupních dat a omezení maximální odchylky výstupu neuronu.

Vektor vstupních hodnot neuronu je přenášen vektorem hodnot w_i, které se jmenují váhy. Na základě tohoto násobeni mohou být vstupní hodnoty posilovány nebo potlačeny. Zatím všechny vstupy se sečtou a k tomuto součtu se přečte prahová hodnota w_0, která se jmenuje *bias*. Posledním krokem je aplikace aktivační funkce a předaní její výsledku na výstup neuronu.

$$input = x_1 * w_1 + x_2 * w_2 + ... + x_n * w_n \tag{2.1}$$

$$output = f\big(input + w_0\big) \tag{2.2}$$

Tím pádem, neuron je definován vahami w_i, prahovou hodnotou (bias) w_0 a jak už bylo řečeno, aktivační funkci f.

Učeni neuronové síti

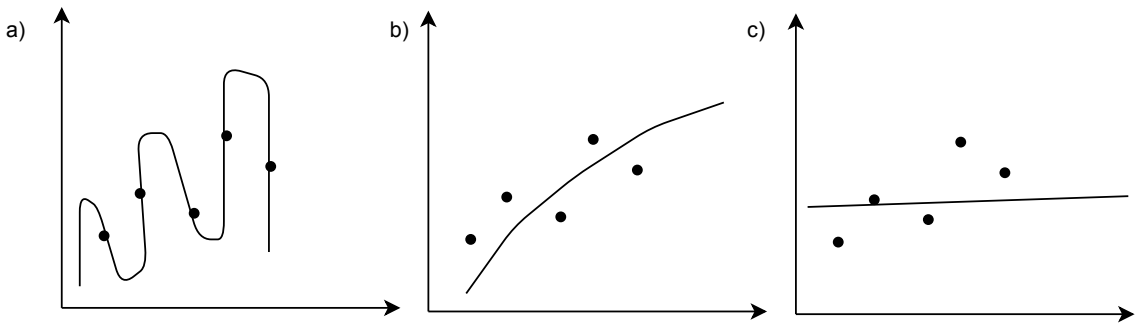
Jednou z nejdůležitějších vlastností neuronové síti je schopnost se učit. Učeni neuronové síti je vlastně změna vah tak, aby ona produkovala požadované výsledky. Jsou různé algoritmy učeni neuronových sítí, ale nejpoužívanější je učeni s učitelem (*supervised learning*) [12]. Adaptace vah se provádí pomoci algoritmu zpětného síreni chyb (*backpropagation*) [7].

Pro učeni neuronové síti je potřeba množství trénovacích dat. Pro různé potřeby jsou různé typy trénovacích datasetu. To může být například, obrázky rostlin, kousky textu

apod. Pro sběr dat pro naučeni neuronové síti v dáne práce bude řečeno v podkapitole 3.1.1.

Celý dataset prochází neuronovou síti několikrát. Tomu se říká *epoch*. Jedna epocha je jedno procházeni celého trénovacího datasetu. Protože celý dataset je příliš velký, on se rozdělí na menši častí, které se jmenuji *batches*.

Nedostatečný počet epoch přivede k nedoučeni síti, a příliš velký počet epoch přivede k přeučeni. Proto metodou experimentu je potřeba vybrat jejich optimální počet.



Obrázek 2.2: a) – přeučena neuronová síť. b) – optimálně naučena neuronová síť. c) – nedoučená neuronová síť

2.2.2 Hough Transform

Jedním z nejvíc používaných a efektivních metodu detekce primitiv je skupina metod, založených na myšlence Houghove transformace [11]. Houghova transformace byla na začátku implementovaná pro detekce čar na binárním obrázku. Ale potom se objevila možnost používat Hoghovou transformace pro detekce kruhů. Metoda funguje na základě využiti prostoru parametrů ve kterém se provádí hlasovaní a vyhledávaní přímek.

Nejvíc používaný jsou následující parametrické rovnice přímky:

$$Y = kX + b \tag{2.3}$$

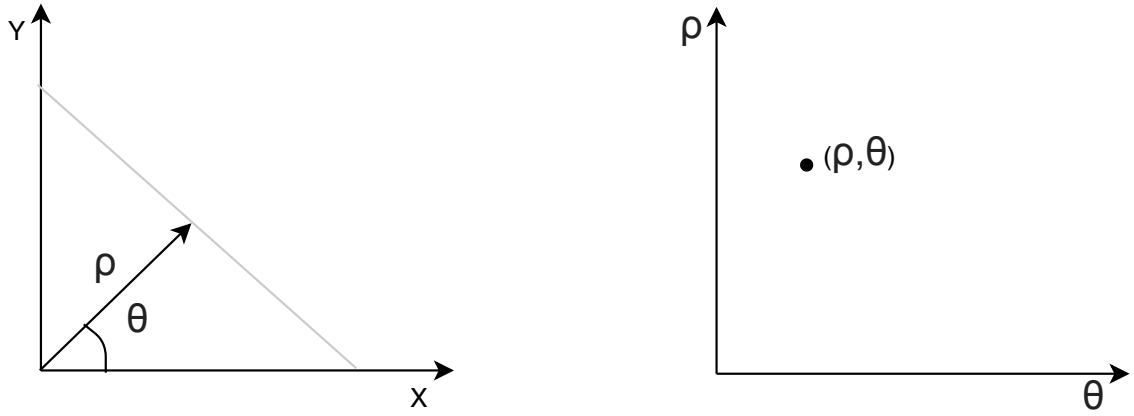
$$X \cos\theta + Y \sin\theta = \rho \tag{2.4}$$

V případě detekci přímek, prostor parametru vždy bude mít dvě dimenze. Hough Transform využívá parametry (ρ, θ).

Nech obrázek je reprezentován jako množina teček (x,y) v prostoru $E = (X, Y)$. Množina přímek, která se prochází přes každou tečku (x,y) může být reprezentováno jako množství teček (ρ, θ) v prostoru $P = \{\rho, \theta\}$. Zobrazení přímky v prostoru E a v prostoru P je vidět na obrazu 2.3.

Myšlenka Hough transformace spočívá v tom, že pro každou tečku z prostoru parametrů počítá počet hlasů, které byli přidělené této tečce. Hlasovaní je vlastně inkrementace počitadla na prostoru P. Čim vetší počitadlo, tím víc bílá bude tečka.

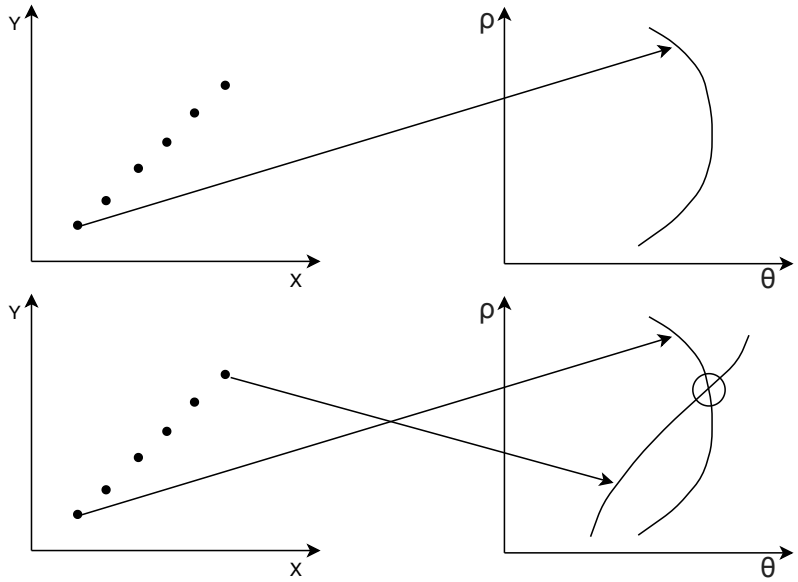
Podle rovnice 2.4 se dá vidět, že přímku na prostoru E lze představit jako tečku na prostoru P. Přes každou tečku na prostoru E lze provést nekonečně mnoho přímek s různým θ úhlem, který tvoří osa X a dána přímka s různou ρ, který je vlastně délka kolmice, vypuštěné ze začátku systému souřadnic $(0, 0)$ k dáne přímce. Protože každá přímka je na prostoru P reprezentovaná jako tečka, nekonečný počet přímek, který se prochází tečkou na prostoru E bude tvořit postupnost teček na P, což je sinusoida. Tady se využívá pravidlo,



Obrázek 2.3: Vlevo – přímka v prostoru (x, y). Vpravo – přímka v prostoru se souřadnicemi (ρ, θ)

.

že libovolné dvě sinusoidy se P kříží v tečce (ρ, θ) jenom v případu, kdy dvě tečky na E, které té sinusoidy tvoří, leží na přímce, kterou popisuje rovnice 2.4 s parametry (ρ, θ). 2.4. Funkce $A = (\rho, \theta)$ se jmenuje **akumulativní funkce**. Z toho je vidět, že počet přímek v prostoru E se rovna počtu lokálních maximů funkce A. To znamená, že pro nalezení všech přímek v E stačí najít všechny lokální maximy funkce A. Absolutní hodnota funkce A v tečce (x, y) se rovna počtu teček, které leží na jedné přímce v prostoru E.



Obrázek 2.4: Vizualizace teček v prostoru (x, y) jako sinusoida v prostoru (ρ, θ). Zobrazeni, jak kříženi dvou sinusoid reprezentuje pozice dvou teček, ležících v (x, y).

Houghová transformace může být využita nejenom na detekce čar, ale i na detekce kruhů, elipsy, křivek a jiných primitiv [9], které můžou být popsané pomoci analytické rovnice a pomoci parametrů, například čtvereček nebo mnohoúhelník.

Je nutné zvýraznit nedostatky Houghové transformace. Její efektivita se snižuje při zvětšení prostoru parametru, proto je nutnost maximálně zmenšit počet parametrů křivky. Proto bylo vytvořeno několik modifikaci originálního algoritmu.

Probablistic Hough Transform – řeší výše popsaný problém tím, že provede Houghová transformace jenom pro část teček originálního obrazu. Na začátku vezme jenom kontrolní tečky s obrázku a na ně aplikuje transformace [2].

Randomized Hough Transform – náhodně se vybírá dvě tečky originálního obrazu a přes ně aplikuje přímka a počitadlo, které odpovídá té přímce, zvětšuje se. Toto se opakuje v několik iteraci, čim víc iteraci se provede, tím lépe bude přesnost výběru. Pro nalez několika přímek, první nalezená přímka z originálního obrazu se maže a začíná se náhodný výběr dvou teček ze začátku.

Hierarchical Hough Transform – originální obrázek se poděli na čtverci menši velikosti a pro každý čtverec aplikuje Houghová transformace a na výstupu jsou segmenty přímek. Dále každé čtyři sousední čtverci se spojí do jednoho a na ně se taky aplikuje Houghová transformace. Jestli segmenty nevytvořili přímku – oni se považuji za šum, jinak, spojeny čtverec se považuje za segment a Houghová transformace se aplikuje na úroveň výše.

V dané práce bylo řešeno využit jednu z modifikaci klasického algoritmu Hought transform, který provádí příliš mnoho vypočtu a i pro přímku, která má jenom dva parametry. Byl využit algoritmus Probablistic Hough Transform.

2.2.3 Existující řešeni v dáne oblasti

Myšlenka dáne práce není nova a ona už byla realizovaná. Tato kapitola popisuje existující řešeni v dáne oblasti, jejich výhody a nevýhody před aplikaci, kterou se zabývá dána práce.

První aplikaci je rozpoznávač tabulek ABBYY FineReader [1] od společnosti ABBYY. Výhodou dáne aplikace je to, že ona dovoluje konvertovat tabulku z obrázku do několika formátů: .docx, .xlsx, .rtf, .pptx, .txt, .fb2, .pdf, .epub.

Po zkoušce aplikace na různých vstupech, případně na různých deformacích vstupních obrázků, objevil se problém, že aplikace hlásí chybnou hlášku, že zkonvertování není možné. Toto řeší aplikace v dáne práce. Řešením je to, že program vrátí prázdní buňku nebo chybnou strukturu, kterou uživatel bude muset ručně opravit a dopracovat.

Druhou aplikaci je vypuštěné v roce 2019 rozšířeni k mobilní aplikaci EXCEL od společností Microsoft[2]. Rozšířeni dovoluje fotit tabulky a zkonvertovat je primo do stránky v Excel dokumentu.

Na rozdíl od programu, kterým se zabývá dána práce, rozšířeni v Excel Mobile umí rozpoznávat tabulky, které vybudované bez použití čár a hranic[3].

První nevýhodou této aplikaci je nedostateční kvalita rozpoznávání bez nuceni uživatele ručně vybírat oblast, kde se nachází tabulka. Podruhé, není možné rozšířit implementací na jiný formáty, například .odf nebo .odt. Potřetí, aplikace nedovoluje rozpoznávat několik tabulek najednou, což řeší aplikace, vytvořena v dáne práce.

V průběhu vyhledávaní podobných aplikaci, bylo nastudováno ještě několik řešeni, které jsou podobné dáne práci. Ale tyto řešeni vetšinou jsou vytvořené pro rozpoznávání ideálně

[1] ABBYY FineReader: https://finereaderonline.com/en-us

[2] Microsoft Excel Mobile: https://www.microsoft.com/en-us/p/excel-mobile/9wzdncrfjbh3

[3] Popis aplikace s video: https://www.theverge.com/2019/3/1/18246429/microsoft-excel-covert-photos-data-tables-editable-table-ai-feature

nakreslených tabulek, například Rovných tabulek v .pdf dokumentech. Což už je rozdíl od dáne prací, protože ona dovoluje konvertovat vyfocené tabulky.

Kapitola 3

Data

Pro vytvořeni kteréhokoli rozpoznávače je potřeba velkého množství testovacích dat, obrázků. V případě velkého množství obrázků s tabulkami. Přičemž tabulky musí být maximálně náhodné a liší se od sebe navzájem, aby otestovat všechní případy detekce a za různých podmínek (špatné foto, blur efekt, špatné osvětleni, rotace obrázku, různá barva textu a čar, spojené buňky horizontálně a vertikálně atd.). V prací bylo využito dva způsoby sběru dat:

- Focení tabulek na kameru
- Generovaní náhodných tabulek

3.1 Generátor náhodných tabulek

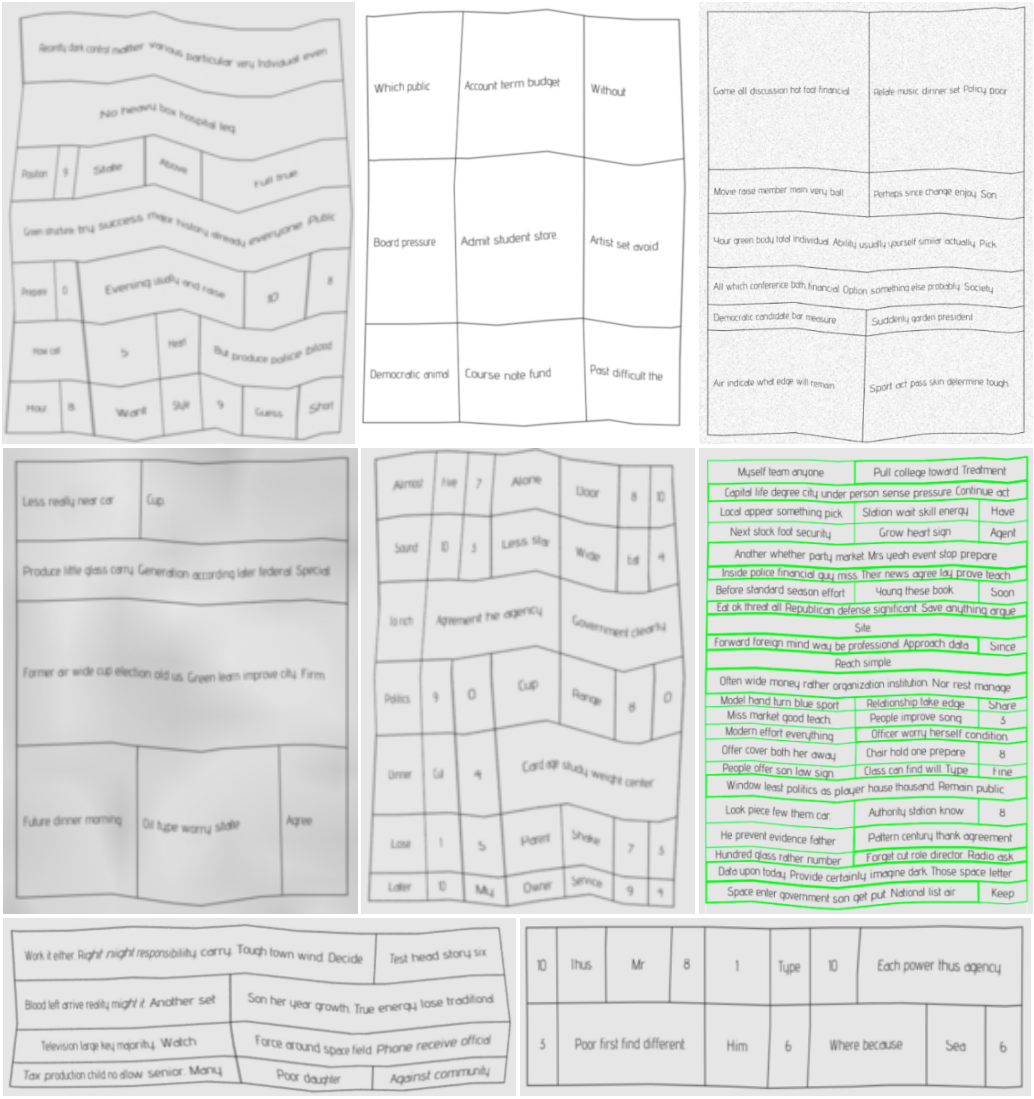
Prvním způsobem jak byla nasbíraná datová sada pro danou aplikace je vytvořeni automatického generátoru tabulek.

Pomoci Python a knihoven OpenCV a Pillow byl vytvořen nastroj, který povolí generovat náhodné obrázky s náhodnými tabulkami a k tomu anotace ve formátu JSON, ve které jsou popsané vlastnosti a souřadnice přímek vygenerované tabulky. Je to CLI aplikace, která přijímá na vstup první parametr "-n", který říká kolik tabulek je potřeba vygenerovat. Ostatní parametry jsou volitelné, a jsou triggery pro různé transformace vygenerované tabulky(data augmentation). Příklady vygenerovaných tabulek s různými parametry jsou na obrázku 3.1.

Nejdůležitějším zadáním generátoru je maximální náhodnost vygenerovaných tabulek aby zachytit nejvíc možných případů pro rozpoznávaní. Proto, skoro všechny hodnoty, bud' to barva, odstup od kraji obrázku do tabulky, tloušťka přímek a mnoho jiných, jsou náhodné v nějakých mezích, které je možné nastavit v konfiguraci nebo v argumentech, které generátor obdrží ze CLI.

Obecný postup je takový, že generátor nejprve pomoci knihovny **argparse** zpracuje argumenty, které byli předané přes CLI a určí počet obrázků k vygenerovaní. Pro každý výsledný obrázek vytvoří třídu ResultImage do které předá argumenty a zatím spustí generovaní. V třídě ResultImage na začátku se nastaví náhodné hodnoty pro barvy přímek tabulky, náhodná veličina, která ukazuje jestli je potřeba spojit všechny buňky prvního řádku dohromady, orientace textu (center či left), a taky barvu pozadí. Taky na začátku generátor inicializuje prázdný obrázek RGBA s nastavenou náhodnou barvou pozadí.

Prvním krokem generování a kreslení tabulky je vytvoření kontejneru (wrapper), který vlastně bude okraji tabulky. Kontejner je čtvereček s náhodnou polohou vrchního levého a



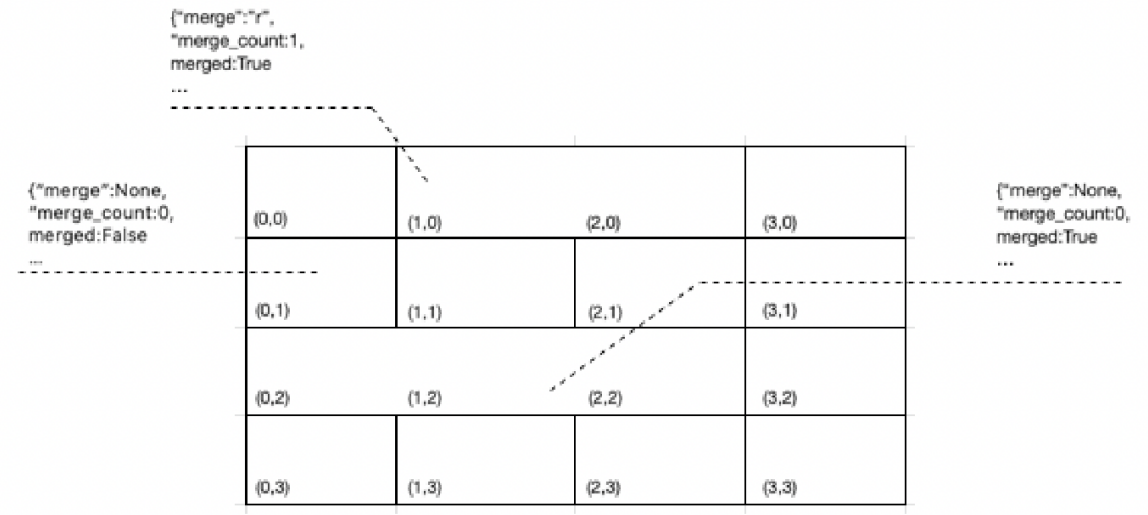
Obrázek 3.1: Příklady vygenerovaných obrázků.

dolního pravého úhlu. A taky už se dá zapsat v anotaci polohy všech úhlů, šířky a výšku tabulky.

Po vygenerování kontejneru se provádí generování buněk tabulky. Základní algoritmus je takový: cyklus pro vytváření řádku je uvnitř cyklusu pro vytváření sloupců. V prvním cyklusu se vytvoří sloupec s náhodnou šířkou a pokud je to první sloupec, v cyklusu se vytvoří řádky s náhodnou výškou pokud celková výská je menši než výška tabulky. Jestli sloupec je druhy nebo další, v cyklusu pro vytváření řádku budou brát hodnoty s prvního sloupců, protože výská řádku v druhém a dalším řádku musí být stejná jako v prvním. Buňky tabulky se zapisují do pole se slovníkem, které mapují strukturu tabulky a ve kterém jsou anotace ke každé buňce (orientace textu, souřadnice každého úhlu, parametry spojeni), nastavené na počáteční hodnoty pro další zpracování. Na konci se počítá počet sloupců a řádků v tabulce a zapisuje se do vlastnosti třídy.

Základní tabulka se všemi buňkami a přímkami je vytvořena, ale reálném světe jsou tabulky, které mají spojené buňky. Tím se bude zabývat generátor v dalším kroku. Zadáním

je modifikovat pole buněk tabulky tak, aby v něm byli náhodné spojené sloupce a řádky. Výsledek je podobny náhodnému pole ze hry BattleShip, kde lodě můžou být náhodně postavené na poli NxN a spojovat buňky tohoto pole. Schematické zobrazení výsledného pole se spojenými buňky je na obrázku 3.2. V parametrech generátoru lze nastavit veličinu náhodnosti spojeni buněk: 1 – spojení všech buněk, 0.5 – spojená každá druhá buňka, 0 - nespojovat nic. Jestli došlo ke spojeni, program náhodně vybere počet buněk ke spojeni a v cyklu nastaví všem spojeným buňkám v anotace parametr "merged"na hodnotu "True". Jestli generátor má nastavený parametr "$merge_header$", provede se spojení všech sloupců v prvním řádku. Zatím generátor vybere jednotlivé spojeni a uloží je do zvláštního pole.



Obrázek 3.2: Schematické zobrazeni pole se spojenými buňkami.

Dalším krokem je vykreslit strukturu tabulky. Nejprve generátor vykresli všechny buňky jako čtverečky, zatím povrch ně vykresli všichni spojení taky jako čtverečky, ale vyplní pozadí aby překryt nimi strany čtverečku pod požadovaným spojením.

Po vykreslovaní tabulky lze přejít k její vyplněni. Pomoci knihovny Faker se vygeneruje náhodný řetězec textu (počet slov lze nastavit v konfiguraci generátoru) pro každou buňku. Vygenerovaný text se předá pomocné metodě pro obalení a zkrácení textu, aby on nepřevyšoval šířku buňky, která je vypočtena a i pro spojeni. Pozice textu se může náhodně měnit. On může mít vyrovnaní zleva a v centru.

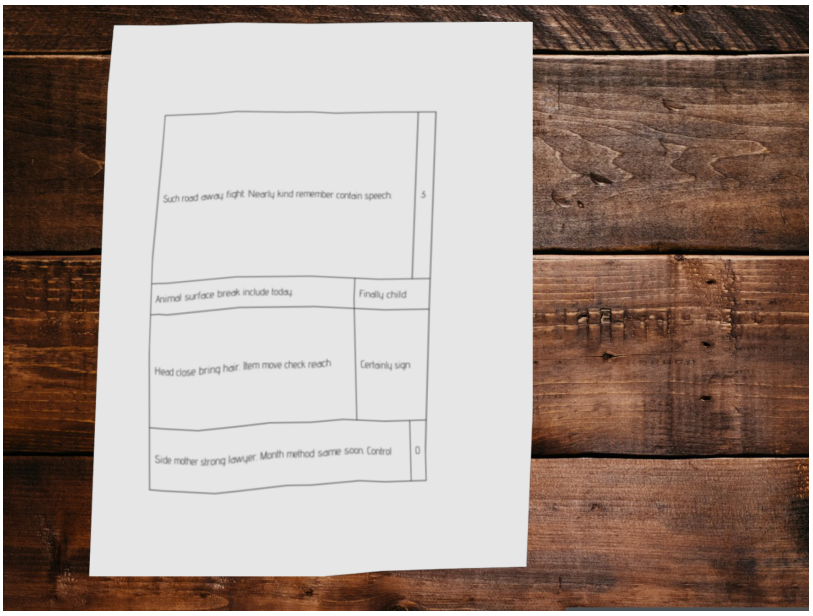
Protože daná práce je zaměřena na vytvořeni rozpoznávače reálných vyfocených tabulek, vytvořeny generátorem dataset obrázku nesplňuje počáteční podmínky reálně focených tabulek. Obrázky jsou ideálně a bez žádného šumu, přímky jsou 90 stupňů svisle a 180 stupňů horizontálně. Tohle řeší rozšířeni generátoru na třídu, která generuje různé a náhodné zkresleni a deformace (Data augmentation) pro vytvořeny obrázek s tabulkou. Všechny augmentace lze nastavit v parametrech generátoru. Pro dány účel se využívá open source knihovna **Imgaug**. Všechny typy augmentace a potřebný pro ně parametry lze najít v Readme k projektu, ale nejdůležitější jsou:

- **PiecewiseAffine** – parametr "–piecewise". Je součásti geometrické augmentace. Umístí do obrázku mřížku bodů, která má řádky R a sloupce C. Pak posune body (a oblasti obrazu kolem nich), což vede k lokálním deformacím různých sil [5].

- **Blur** – parametr "–blur". Vytváří rozmazanosti a zhorši kvalitu čitelnosti. Generátor aplikuje jeden ze třech typu blur efektu. Který typ bude aplikován vybírá náhodně.
  - **GaussianBlur**
  - **AverageBlur**
  - **MotionBlur** – blur efekt s zadaným úhlem a směrem.
- **PerspectiveTransform** – parametr "–perspective". Je součásti geometrické augmentace. Děla se pomoci knihovny OpenCV. Aplikuje náhodnou perspektivu na obrázek.

Mezi parametry generátoru taky patři parametr –**background**, pomoci kterého lze imitovat foceni reálné ležícího papíru na stole. Využívá náhodné vybrané obrázky s datasetu povrchu stolu. Příklad je na obrázku 3.3.

Generátor taky umí vytvářet různé šumy, hra se s alpha kanálem, rotace obrázku a další.



Obrázek 3.3: Aplikovaní náhodného fonu pod vygenerovanou tabulku.

3.1.1 Generátor uhlu

Jedním ze zadaní rozpoznávače tabulek je detekovat pozice tabulky na obrazu (souřadnice každého ze čtyř úhlů tabulky). Proto se využívá neuronová síť, pro trénovaní které je potřeba datasetu, který bude zahrnovat všechny možné typy úhlů tabulky. Pro tento účel bylo řešeno integrovat do generátoru ořezávač úhlů a ořezat všechny čtyři typy úhlů tabulky a jiné častí primitiv, potřebných pro trénováni neuronové síti.

Pomoci parametru generátoru "–crop-corners"se vytvoří třída CornerCropper, která převezme anotace každé buňky z třídy ResultImage a ořeze každý úhel tabulky čtverečkem 32x32 a uloží výsledný obrázek.

Pro trénováni neuronové síti je potřeba ke každému trénovacímu obrázku vytvořit anotace (třídy klasifikátoru). V dánem případě anotace k obrázku je název složky, odpovídající typu ořezané primitivy (levý horní úhel, pravý dolní atd.). Zvláštním typem je *not_corner*

třída. Do teto třídy klasifikátoru patři všechny obrázky, které nejsou úhly tabulky. To můžou být častí textu, prázdné místo, kříženi dvou přímek apod.

Protože generátor vytváří tabulky maximální náhodnosti s různou deformaci, ořezané úhly taky budou mít náhodnou formu a deformace.

Reprezentace vygenerovaných typu úhlů s anotaci lze uvidět na tabulce 3.1.

	`left_top_corner`		`right_top_corner`
	`left_bottom_corner`		`right_bottom_corner`
	`not_corner`		`not_corner`
	`not_corner`		`not_corner`
	`not_corner`		`not_corner`

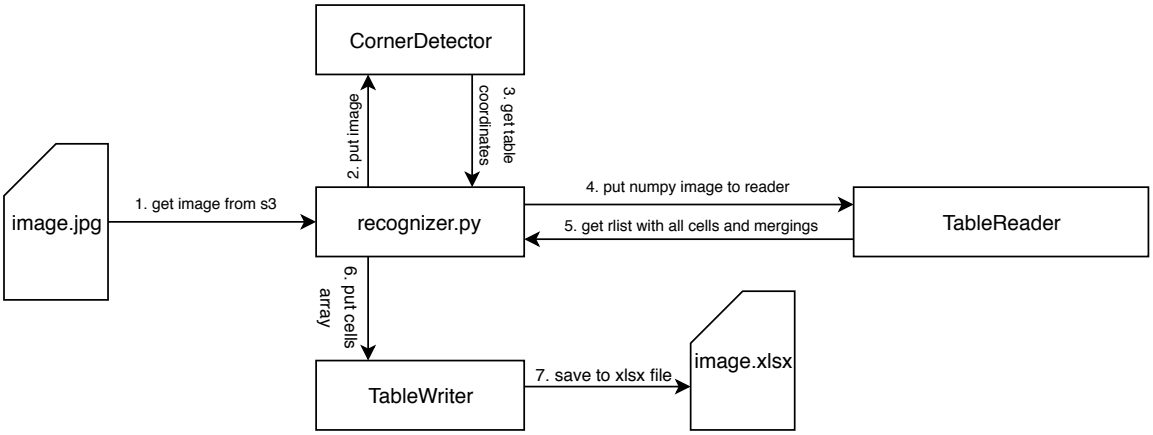
Tabulka 3.1: Typy uhlu v trénovací sade pro neuronovou síť na detekce pozice tabulky.

Generátor vytváří dva datasety. Vetší dataset je určen pro trénováni neuronové síti, menší – pro testovaní.

Kapitola 4

Návrh a implementace rozpoznávače tabulek v obrazcích

Struktura aplikace pro rozpoznávání tabulek se dělí na tři častí, jdoucích za sebou, předávání dat z jedné části do druhé se děla v hlavním souboru recognizer.py. Schematické zobrazení struktury aplikace lze vidět na obrazu 4.1. Součástí práce je taky frontent: vytvoření webové aplikace a rozhraní, přes které web komunikuje s rozpoznávačem.



Obrázek 4.1: Struktura rozpoznávače tabulek.

V dalších třech sekcích kapitoly je popis každé častí aplikace uveden zvlášť. Dále je sekce s popisem frontendu práce. Poslední sekce zahrnuje popis vývojového prostředí a nástrojů, které byli použité při implementaci.

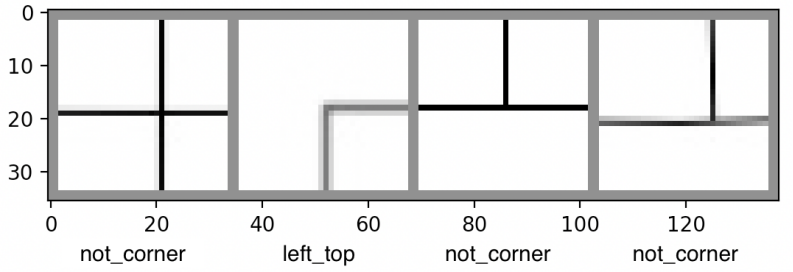
4.1 Detekce úhlů tabulky v obrazu

Pro detekce úhlů tabulky byl vytvořen detektor, postaveny na neuronové síti pomoci Python knihovny PyTorch.

Zadáním je vytvořit jednoduchý klasifikátor, který přijímá na vstup 32x32 obrázek a na výstupu vypisuje název třídy, do které patři objekt na obrázku.

Vytvořeni trénovacího datasetu je popsáno v 3.1.1. Anotaci k trénovacímu obrázku je název složky, ve které tento obrázek je uložen. Je potřeba rozbit trénovací dataset do men-

ších částí (*batches*) a převést ho do tensoru. Náhodné vybrané obrázky z datasetu detektoru a anotace k nim jsou na obrázku 4.2.

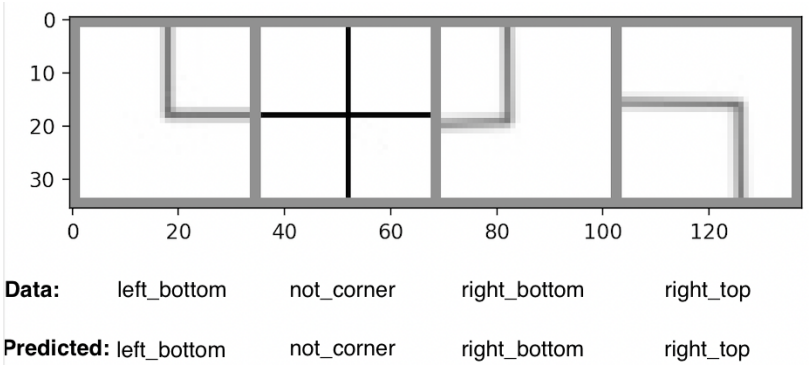


Obrázek 4.2: Náhodné vybrané čtyři obrázky z trénovacího datasetu a anotace k nim.

Zatím se inicializuji třídy, zapsané do pole, které bude potom namapované na vystup neuronové síti. Dalším krokem je vlastně vytvořeni modelu neuronové síti a aplikovaní loss funkce, která se jmenuje **CrossEntropyLoss**.

Model a dataset neuronové síti je připraven. Nasleduje cyklus s počitadlem epoch a cyklus procházeni celým datasetem. V tomto cyklusu upravují váhy neuronu.

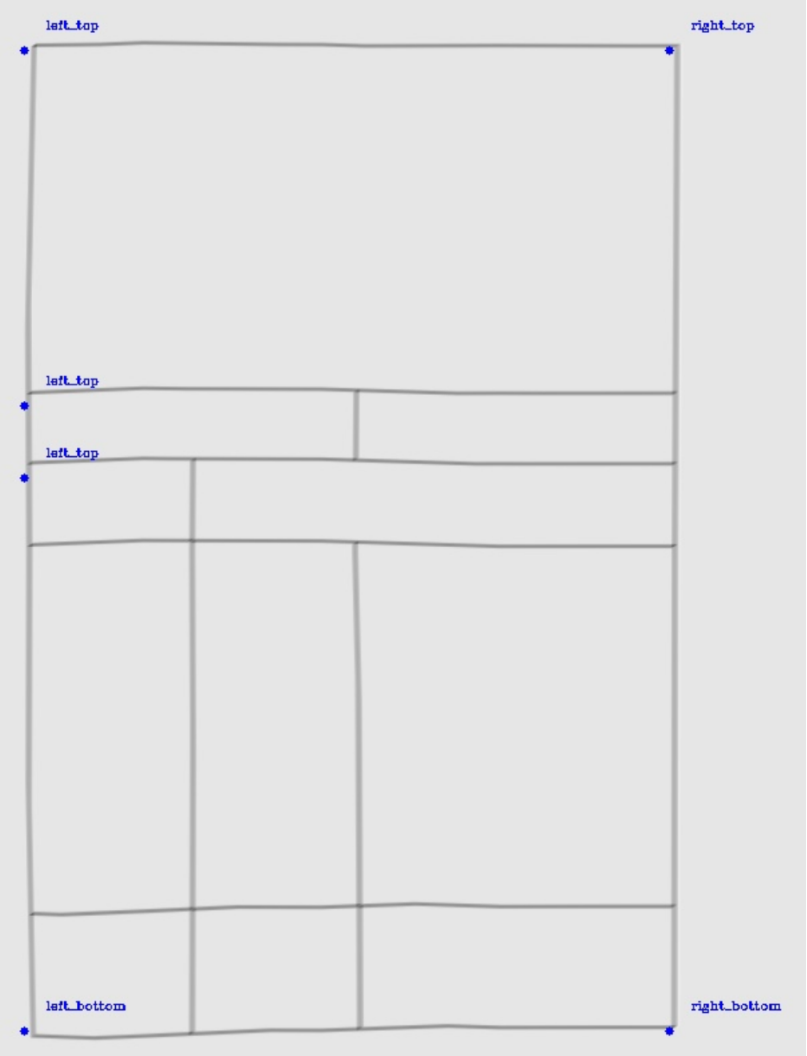
Na obrázku 4.3 lze vidět výsledek spuštění natrénované neuronové síti na čtyři náhodných obrázků z testovacího datasetu.



Obrázek 4.3: Spouštění natrénované neuronové síti na náhodných obrazcích z testovacího datasetu.

Dalším krokem je implementace modulu detektoru. Detekce úhlů tabulky v obrázku je postaveno na algoritmu posuvného okna (*sliding window*). Vytváří se neviditelný čtvereček velikosti 32x32 a nastavuje se na začátek obrázku v horní levý úhel. Zatím s určitou veličinou kroku se posouvá doprava pokud nedojde do konce řádku a následně se posune dolu a algoritmus se opakuje. Pro každou pozice čtverečku se provádí ořezaní kousku obrázku, a tento kousek pustí se na vstup natrénované neuronové síti. Tím pádem jsou detekované všechny čtyři úhly tabulky.

Tato implementace není do konce realizována. Přesnost detekce uhlu není 100% kvůli nedostatečné rozmanitého datasetu. Ukázka, jak program detekuje uhly na testovacím obrázku je zobrazena na 4.4.



Obrázek 4.4: Ukázka detekce uhlu.

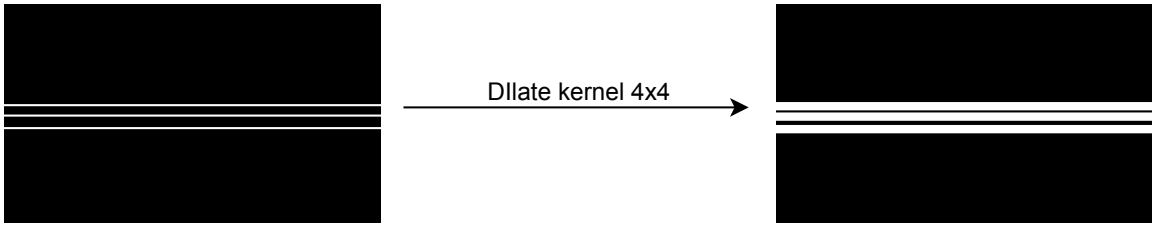
4.2 Rozpoznávaní primitiv tabulky a její naplní

Čteni primitiv (přímky, text, buňky) se děla v třídě TableReader, do které na vstup přijde numpy obrázek. Vetší část rozpoznávaní se děla pomoci knihovny OpenCV.

Nejprve se obrázek zkonvertuje do grayscale[3]. Dále, pomoci algoritmu *Canny edge detection*, se provede detekce hran v obrázku. Pozadí obrazu bude černé a všechny hrany objektu budou bílé.

Pro zvetšení viditelnosti úhlů a přesnosti rozpoznávaní je potřeba udělat dilatace obrazu s úhly. **Dilatace** je morfologická operace pro zvetšení vlastností obrazu a vlastností primitiv v obrazu [4, p.523]. V případě dáne aplikace pomoci dilatace je větší jistota, že potřebná primitiva tabulky bude detekovaná, a ne vynechaná. Na obrázku 4.5 lze vidět jaky vliv má dilatace na parametry jednoduché primitivy (přímky). Tím se bude zvětšovat přesnost rozpoznávání.

Příštím krokem je rozpoznávání přímek v obrázku. Rozpoznávání struktury tabulek v dáne práce je postavené na detekci přímek, které tuto strukturu tvoří. Pro detekce přímek byl použit algoritmus *Probablistic Hough Transform*, což je modifikaci klasického al-



Obrázek 4.5: Hrany přímek před dilataci (zleva) a po dilataci (zprava).

goritmusu *Hough Transfrom*. Oba algoritmusy a jejich rozdíly jsou popsané v sekci 2.2.2 teoretické častí práce.

Jedním z problému, který vznikl na praxi je ten, že implementovaný v OpenCV algoritmus HoughTransformP vrátí počáteční a konečné souřadnice přímky nejenom ve směru shora dolu, ale i zdola nahoru. Proto byla potřeba normalizovat výstupní přímky tak, aby horizontální čáry procházely zleva doprava, a vertikální – shora dolu.

Dalším problémem, který byla potřeba řešit je ten, že rozpoznaná přímka může procházet v jedné častí správně jak to má byt, a v jiném místě může procházet povrch textu. Tohle už je chyba, rozpoznávač ji detekuje jako falešně detekovanou (*false positive*) a smaže celou (o redukci *false positive* přímek procházejících nad textem bude řečeno v několika odstavcích vepředu). Ale je potřeba nechat tuto část, která je správně detekovaná, a smazat tuto část, která prochází povrch textu. Proto jsou všichni přímky pozbyti na několik menších častí pomoci vytvořené metody, která přijímá jako první parametr pole s přímkami, a jako druhy parametr počet častí, na které je potřeba každou přímku podělit.

Dále jsou všechny přímky rozdělené do skupin podle orientace (horizontální a vertikální) pomoci vytvořené metody, která určí úhel náklonu přímky k ose X ve stupních pomoci matematické rovnice 4.1. Jestli úhel náklonu leží v mezích $0° \leq \theta \leq 44°$, přímka je horizontální, jinak – svislá.

$$\theta = \arctan\left(\frac{y2 - y1}{x2 - x1}\right) * \frac{180}{\pi} \tag{4.1}$$

4.2.1 Detekce textových bloku

Součásti práce je taky detekce a rozpoznávání textu v každé buňce, jestli takový v ní existuje. Zadáním je nejenom rozpoznat text, ale detekovat jeho polohu, šířku a výšku, a určit neviditelné čtverečky, které ohraničuji text ze všech čtyř stran (*Bounding Box*). Pro tento účel obrázek je nejprve zkonvertován do *grayscale*, zatím pomoci algoritmu knihovny OpenCV MedianBlur je aplikován blur efekt pro účel odstraněni šumu. Pomoci bluru a správného určení parametru *ksize* budou také odstraněné přímky z obrazu a je jistota, že zůstane jenom text. Dále na obrázek je aplikované adaptivní prahování (Adaptive Threshold) a následně je aplikovaná už výše popsaná dilatace, a taky eroze, pro zvětšení konturu primitiv a pro zvetšení přesnosti detekce. Dále nasleduje metoda *findContours* pro nalezeni konturu primitiv a jejich obalovaní do obalovacích čtverečků (*Bounding Box*), které potom budou uložené do zvláštního pole pro další účely. Ukázka prahování s určením správného *ksize* pro *MedianBlur* a zatím detekce ohraničujícího čtverečku je na obrázku 4.6.

Tato implementace detekce pozice textu má problém, kvůli kterému může být falešné detekovaný text (*false positive*) a nedetekovaný potřebný text(*true negative*). Počet takových elementů je závislý na jednom parametru, a to je parametr *ksize* algoritmusu *MedianBlur*.

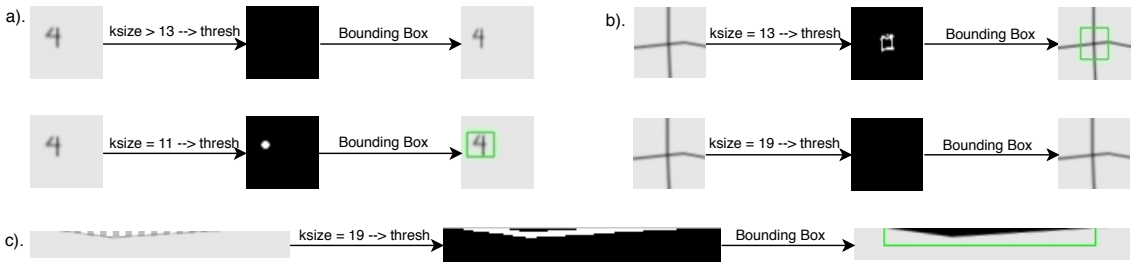
Obrázek 4.6: Ukázka správně detekovaného textu. Zleva – inicializační obrázek. V centru – aplikování *MedianBlur* a určený parametr *ksize* a prahování. Zprava – výsledný ohraničující čtvereček (*Bounding Box*).

Od tohoto parametru závisí nakolik silný bude blur efekt pro odstranění šumu. Z jedné strany může dojít k určení příliš malé veličiny parametru *ksize*, což může přivést k objevovaní *false positive* obalovacích čtverečků (špatně rozmazaná přímka, nějaké znaky, které nejsou součásti textu ale můžou být součásti stran tabulky). Z druhé strany může být určení příliš velkého čísla, což může rozmazat malé písmena, tečky, čísla nebo celý kousky textu. Příklad příliš velkého *ksize*, kvůli kterému je chybně nedetekován potřebný text je na obrázku 4.7(a) nahoře. Na stejném obrázku dole lze vidět řešení problému zmenšením *ksize*. Pro řešení daného problému bylo vymyšleno několik postupů, které se aplikují v programu.

Nejprve jsou smazané příliš malé ohraničující čtverečky (velikost se dá nastavit v konfiguraci aplikace). Myšlenka je taková, že výška či šířka textu, menši než aktuální minimální velikost ohraničujících čtverečků, je příliš malá, aby mohla považovat ten text za správně detekovaný.

Při vývoji a testování částí aplikace, ve které navrhovala detekce textu, bylo mnoho případů, kdy křížení dvou přímek byly špatně smazané pomoci *MedianBlur*, a algoritmus detekoval je jako část textu. Na obrázku 4.7(b) je znázorněn problém a lze vidět rozdíl mezi různými hodnotami parametru *ksize*. Nahoře je chybně detekován text a nakreslen obalovací čtvereček, protože primitiva je přímka, ale při zvetšení *ksize* do hodnoty 19 je výsledek správný. Pro řešení problému vzniku falešně detekovaných kříženi přímek nad textem bylo řešeno aplikovat ještě jednu normalizace. Neuronová síť je naučena na detekovaní všech typu úhlů. Proto je možnost použit algoritmus, který bude mazat všechny ohraničující čtverečky, uvnitř kterých je alespoň jedno křížení.

Třetí normalizaci je smazaní ohraničujících čtverečků za tabulkou, hranice které jsou detekované pomoci neuronové síti. Na obrázku 4.7(c) lze vidět algoritmus detekce textu, který falešně rozpoznává hranice papíru jako část textu a děla chybný ohraničující čtvereček. Pozice je mimo tabulky, a proto čtvereček bude smazán.



Obrázek 4.7: (a) – Ukázka zmizení malé částí (číslo) textu (*false negative*) při výběru příliš velkého parametru *ksize*. Nahoře – chyba, dole – správně. (b) – Falešně detekovaná přímka (*false positive*). Při zvětšeni *ksize* čtvereček zmizí. (c) – Falešně detekovaný text na okrajích papíru.

Byli vymyšlené ještě několik možnosti určit správně všechny ohraničující čtverečky, ale tyto už nebyli implementované kvůli ohraničenému času pro vývoj. Prvním způsobem je nutit konečného uživatele přes uživatelské rozhraní, aby on pomoci posuvného slideru určil správný parametr *ksize*, změna kterého bude v reálném čase měnit strukturu tabulky a ukazovat ji uživateli. Druhým způsobem je naučit aplikace automaticky určovat parametr *ksize* pomoci neuronové sítě, která bude analyzovat počet šumu a rozmazání obrázku.

4.2.2 Smazaní přímek chybně detekovaných nad textem

Protože tabulky můžou být různé, s různou kombinaci odlišné velikosti buněk, je potřeba zachytit co nejvíc možných případů. Proto parametry algoritmusu HoughTransformP *maxLineGap*, *minLineLength*, *threshold* byly vybrané takovým způsobem, aby byla možnost detekovat co nejkratší přímky. Toto tvoří závažný problém, kdy text se detekuje jako přímka. Proto byla potřeba implementovat algoritmus, pomoci kterého by byla možnost smazat falešně detekované přímky nad textem.

Algoritmus řeší tři různé varianty poměru přímky a ohraničující text čtverečku:

- přímka je mimo čtvereček,
- přímka je v hranicích čtverečku a ho nekříží,
- přímka kříží čtvereček.

V prvním případě přímka není smazaná.

Protože ohraničující text čtvereček má informace jenom o horním levém a dolním právem úhlů, v druhém případě přímka a čtvereček musí splňovat systém nerovnosti:

$$\begin{cases} \texttt{l\_t\_x} < \texttt{line\_x1} < \texttt{r\_b\_x} \\ \texttt{l\_t\_x} < \texttt{line\_x2} < \texttt{r\_b\_x} \\ \texttt{l\_t\_y} < \texttt{line\_y1} < \texttt{r\_t\_y} \\ \texttt{l\_t\_y} < \texttt{line\_y2} < \texttt{r\_b\_y} \end{cases}$$

kde `l_t_x`, `l_t_y`, `r_b_x` a `r_b_t` jsou X a Y souřadnice levého a pravého úhlů ohraničujícího čtverečku. Přímky, které splňuji všechny čtyři podmínky jsou smazané.

Ve třetím případě je potřeba implementace pomocného algoritmusu, který bude moct zjistit jestli dvě přímky kříží. Jestli přímka kříží alespoň jednu ze stran čtverečku, ona bude smazaná.

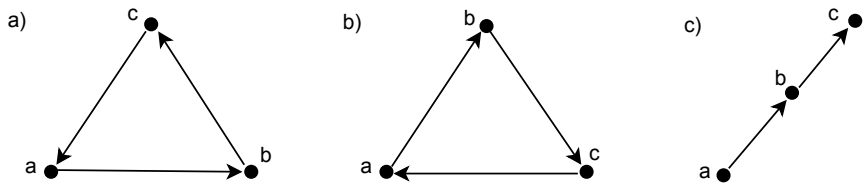
Před vysvětlením samotného algoritmusu, je potřeba definice orientace teček přímek. Orientace třech teček může být *clockwise*, *counterclockwise* a *colinear* [1]. Na obrázku 4.8 je grafické znázorněni všech třech typu orientace teček na prostoru.

Pro kolineárnost třech teček a, b, c na prostoru $E = (x, y)$ platí rovnice 4.2.

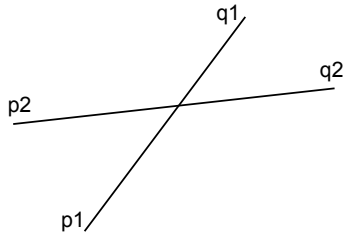
$$\frac{y_a - y_b}{x_a - x_b} = \frac{y_a - y_c}{x_a - x_c} \tag{4.2}$$

Implementace algoritmusu pro zjištěni existenci křížení dvou přímek je postavena na detekci typu orientace třech teček přímek na prostoru. Pro ne platí definice 1:

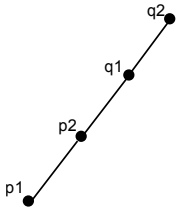
Definice 1 *Nech* $(p1, q1)$ *a* $(p2, q2)$ *jsou dvě přímky na prostoru* $E = (x, y)$. $(p1, q1)$ *a* $(p2, q2)$ *kříží jestli je splněna jedna ze dvou podmínek:*



Obrázek 4.8: (a) – *counter-clockwise* (b) – *clockwise* (c) – *colinear*.



Obrázek 4.9: $(p1, q1, p2)$, $(p1, q1, q2)$ mají různou orientaci a $(p2, q2, p1)$, $(p2, q2, q1)$ taky mají různou orientaci.



Obrázek 4.10: X a Y projekce $(p1, q1)$ a $(p2, q2, p2)$ se kříží a tečky jsou kolineární.

- $(p1, q1, p2)$, $(p1, q1, q2)$ *mají různou orientaci a* $(p2, q2, p1)$, $(p2, q2, q1)$ *taky mají různou orientaci, obrázek 4.9. (General Case)*
- $(p1, q1, p2)$, $(p1, q1, q2)$, $(p2, q2, p1)$ *a* $(p2, q2, q1)$ *jsou kolineární, X projekce* $(p1, q1)$ *a* $(p2, q2)$ *kříží a Y projekce* $(p1, q1)$ *a* $(p2, q2)$ *taky kříží, obrázek 4.10. (Special Case)*

Tímto algoritmusem program probere všechny přímky, horizontální a vertikální, a všechny bloky textu a zjisti, které přímky musí být smazané.

Do implementace tohoto algoritmusu odstraňování přímek nad textem byl vymyšlen ještě jeden algoritmus, který nebyl využit kvůli nedostatečné kvalitě. O tomto algoritmusu bude řečeno v následující podkapitole. O testování a porovnání dvou algoritmusů odstraňování přímek nad textem bude napsáno v kapitole **??**.

První pokus implementace algoritmusu odstraňování přímek nad textem

Základní myšlenkou algoritmusu je nalezení přímek, které se prochází nad pixely, které střídají svou barvu. Podle počtu střídání lze detekovat, jestli přímka prochází nad textem. Nejprve obrázek je převedeny na černý a bílí pomoci *threshold* funkce knihovny OpenCV. Kontury primitiv obrázku, včetně textu, jsou bílé, pozadí – černé. Aby detekovat a analyzovat, které pixely se nachází pod přímkou, byla potřeba existence algoritmusu, který by detekoval směr přímky a probíral by každý pixel. V knihovně OpenCV 2 pro jazyk C++ existuje implementace daného algoritmusu, ale pro OpenCV v jazyku Python daná implementace není dostupna. Proto byl implementován vlastní algoritmus, který iteruje každý pixel přímky a proces obdrženi pixelů ve směru přímky je postaveny na Bresenhamovem algoritmusu. Bresenchamuv algoritmus detekuje, které tečky 2D rastru je potřeba zabarvit, aby obdržet figuru, která bude blízká k přímce mezi dvěma tečkami prostoru.

LineIterator, který byl implementován a využívá se pro tento účel, vrátí pole, které nese informace o pozice každého pixelu a jeho barvu. Barva může nabývat hodnot jenom 0

(černý) a 255 (bílí). Zatím toto pole v cyklusu se iteruje a detekuje, jestli bílí pixel se objeví a zmizí alespoň dvakrát, přímka je nad textem a bude smazaná. Pro přesnost vypočtu jsou vytvořené ještě dvě přímky, o jeden pixel výše a o jeden pixel niž a taky jsou iterované pomoci LineIterator a analyzované na existence střídaní barvy pixelů pod přímkami.

Byla potřeba řešit několik problémů, které tvoří daný algoritmus.

Poprvé, při testovaní se objevil problém, že jsou takové přímky, které se nachází nahoře nebo dolu textu. Jestli přímka je nahoře textu, to znamená, že o jeden pixel výše už není text a algoritmus přímku nechá. Pro řešení takového problému, přímky, které nemají střídaní pixelů nad sebou nebo pod sebou, jsou posunuté ve směru naopak a postup se opakuje.

Podruhé, byla chybná detekce přímek, které musí být smazané. Problém tvoří dilatace, která se využívá při nalezení přímek pomoci HoughTransformP. Při dilatace, jsou přímky, které leží o několik pixelů dolu nebo nahoru od reálné přímky tabulky. Oni jsou algroitmusem rozpoznané jako text, kvůli tomu že kříží s přímkami opačné orientace. Proto algoritmus vypočítá dvě střídaní bílé a černé barvy.

Potřetí, výše uvedeny postup může platit jenom na horizontální přímky, protože může nastat případ, kdy písmenko "I"bude považováno za přímku, ale nebude mít žádné střídaní barvy pixelů.

A poslední problém, kvůli kterému bylo řešeno skončit implementaci a vymýšlet jiný algoritmus, byl objeven při testování obrázků, transformovaných pomoci blur filtru. Problém vzniká na etapu, kdy algoritmus volá *threshold* a konvertuje obrázek v černo-bílí. Kvůli blur efektu a rozmazaní, písmeno se zobrazuje nepřesné. Může se nastat případ, kdy bílou barvou bude označena jenom půlka písmena. Proto algoritmus nechá mnoho falešných přímek nad textem a považuje se za nedostatečně kvalitní.

4.2.3 Normalizace existujících přímek

Na tomto kroku už lze vidět základní strukturu tabulky: její čáry, bloky textu. Problém je v tom, že všechny přímky jsou rozdělené na malé kousky (včetně dilatace, což zvětšuje jejich počet), dohromady tvořící celou čáru a sledují její možné transformace. Je potřeba z možných desítek přímek, reprezentujících jednu dlouhou přímku udělat aproximace a vytvořit aproximační přímku.

Program děla aproximace pro horizontální a pro vertikální přímky zvlášť. Toto zadáni je rozdělené na dvě častí:

- rozdělení do skupin všech krátkých přímek podle pozice,
- spojení všech malých přímek pro každou skupinu a vytvoření aproximační přímky.

Pro grupování přímek jsou vytvořené zvláštní Python slovníky pro horizontální a pro vertikální čáry. Klíčem v slovníku je unikátní identifikátor, hodnotou – pole, sestavené ze všech krátkých přímek, patřících k dáne skupině. Pro tento účel byl vymyšlen rekurzivní algoritmus, který na vstup vezme první čáru, a v rekurzi pro ně najde a vybere všechny čáry, patřící k dáne skupině. Algoritmus 1 reprezentuje zjednodušenou implementaci grupování krátkých horizontálních přímek. Základní myšlenka je taková, že ke každé přímce algoritmus najde přímku vedle ně a přidá do skupiny, a smaže z pole všech existujících přímek, zatím v rekurze pokračuje opakovaně. Když už není žádná vázaná přímka, vezme přísti přímku z pole všech přímek. Pro vertikální přímky je stejný postup, ale zpracovaní x souřadnic.

Algorithm 1 Small Lines Grouping Algorithm.

for *line* in *horizontal_lines* **do** ▷ Iterace přes všechny horizontální čáry
    group_id ← unikátní identifikátor
    vytvořit novou skupinu s názvem *group_id*
    GROUP_HORIZONTAL_LINES(*horizontal_lines*[0], *group_id*)
end for

Vstup: *current_line* - aktuální zpracovaná krátká přímka
Vstup: *group_id* - unikátní identifikátor skupiny krátkých přímek

function GROUP_HORIZONTAL_LINES(current_line, group_id)
    $cur\_x1, cur\_y1, cur\_x2, cur\_y2 \leftarrow$ from *current*
    for *line* in *horizontal_lines* **do** ▷ Iterace přes všichni horizontální cary
        $x1, y1, x2, y2 \leftarrow$ from *line*

        if X projekce *line* je v X projekci *current_line* **then** ▷ Při stejné výšce
            smazat *line* z pole *horizontal_lines*
        end if

        if X projekce *current_line* je v X projekci *line* **then**
            GROUP_HORIZONTAL_LINES(*line*, *group_id*) ▷ Při stejné výšce
        end if

        if X projekce *line* kříží z X projekce *current_line* **then** ▷ Při stejné výšce
            smazat *line* z pole *horizontal_lines*
            GROUP_HORIZONTAL_LINES(*line*, *group_id*)
        end if
    end for
end function

Pro aproximaci krátkých přímek dlouhou přímkou byl taky implementován algoritmus, který počítá střední hodnotu y souřadnice přímky a minimální ta maximální hodnoty x souřadnic ze všech přímek. Tím pádem je tři proměnné $(x1, x2, y)$ aproximační přímky. Pro vertikální přímky postup je podobný, rozdíl jen v tom, že počítají $y1, y2$ a x souřadnice.

Následující problém, který byla potřeba řešit a který dělá není potřebnou detekci textových oblastí (*false positive*), a následující smazaní přímek v této oblasti je ten, že aproximované přímky nedokresly do hranic tabulky nebo jednotlivých buněk. Řešením je implementace algoritmusu pro dokresleni takových přímek. Dělá se to taky pro horizontální a vertikální přímky zvlášť. Myšlenka algoritmusu je taková, že pro každou horizontální přímku se najde nejbližší pro ně vertikální přímka vpravo a vlevo při odpovídající y souřadnicí. Jestli vzdálenost x souřadnice horizontální přímky je menši než nějaká nastavena v konfiguraci hodnota, přímka se dokresli do x souřadnice odpovídajících ji vertikálních přímek vpravo a vlevo.

Příštím krokem je ještě jedná jednoduchá normalizace, pomoci které program odstraňuje falešné detekované přímky. Odstraňují se čáry, které nekříží alespoň z jedné strany.

Pro výpočet počtu řádků a sloupců výše uvedené implementace ještě nestačí. Kvůli možným spojením buněk, program může detekovat několik přímek na jednom řádku nebo sloupci. Proto je potřeba spojit částí přímek, odpovídajících jednomu řádku nebo sloupci do jedné přímky a vytvořit zvláštní pole, do kterého se tyto přímky uloží. Opět pro vertikální a horizontální přímky jsou tyto pole zvláštní. Zatím počet sloupců a řádků je počtem elementů v těchto polích.

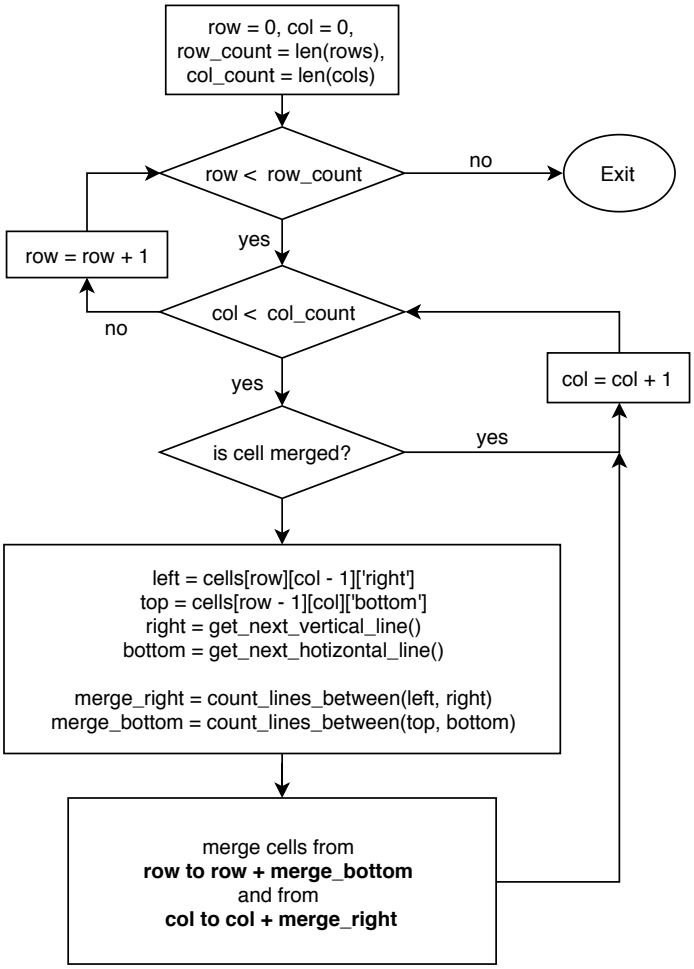
4.2.4 Vybudovaní pole, reprezentujícího tabulku

Dále program už je připraveny k vybudováni datové struktury, která bude přesné odpovídat kostře cílové tabulky. Inicializuje se pole $N \times M$, kde N je počet řádku, a M je počet sloupců tabulky. Ke každé buňce pole je vytvořena anotace, ve které se inicializují počáteční hodnoty buňky (informace o spojení, souřadnice každého z úhlů, které přímky odpovídají každé straně buňky a text buňky).

Zatím byl implementován algoritmus, který sbírá data pro každou buňku pole. Velmi zjednodušené schematické zobrazení algoritmusu je na obrázku 4.11. Výsledné pole je podobné jako výsledné pole, které generuje implementovaný generátor tabulek, o kterém je detailně popsáno v kapitole 3.1. Místo vysvětlovaní jak ono musí vypadat, na obrázku 4.12 je jeho grafické zobrazení. Pomoci těchto přesných anotaci, jako šířka či výská buňky, souřadnice úhlu, dá se přesné zkonvertovat tabulku a zachovat všechny proporce, ale zatím toto nebylo implementováno. Implementace této možnosti je jako cíl k budoucímu rozšíření aplikace.

Jednou z důležitých věci, která byla implementovaná v tomto algoritmusu, je extrakce textu z buňky. Pro tento účel stačí jen souřadnice horního levého a dolního pravého úhlů buňky. Pomoci knihovny OpenCV je vyřezaná část obrázku podle souřadnic buňky, zatím tato část obrázku je předaná knihovně Tesseract pro rozpoznávaní textu v obrázku. Knihovna vrací rozpoznaný řetězec, který je nahraný do anotace buňky do pole "*content*".

Tímto implementace rozpoznávače tabulky skončí. Pole, které reprezentuje strukturu tabulky je zachované do vlastnosti třídy, a je připravené k předaní příštímu modulu, který se bude zabývat vykreslením tabulky do XLSX souboru.

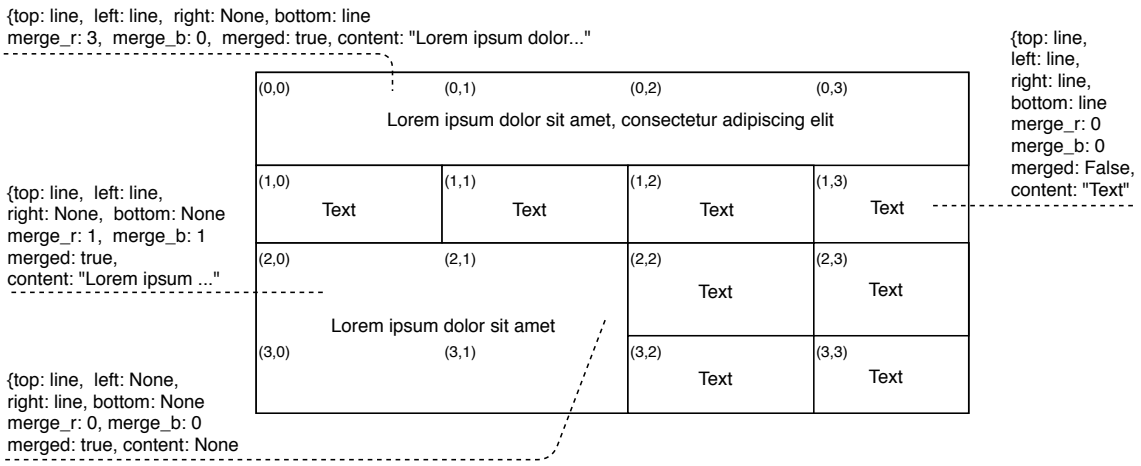


Obrázek 4.11: Schematické zobrazení algoritmusu extrakce a spojení buněk tabulek podle informace o pozice přímek.

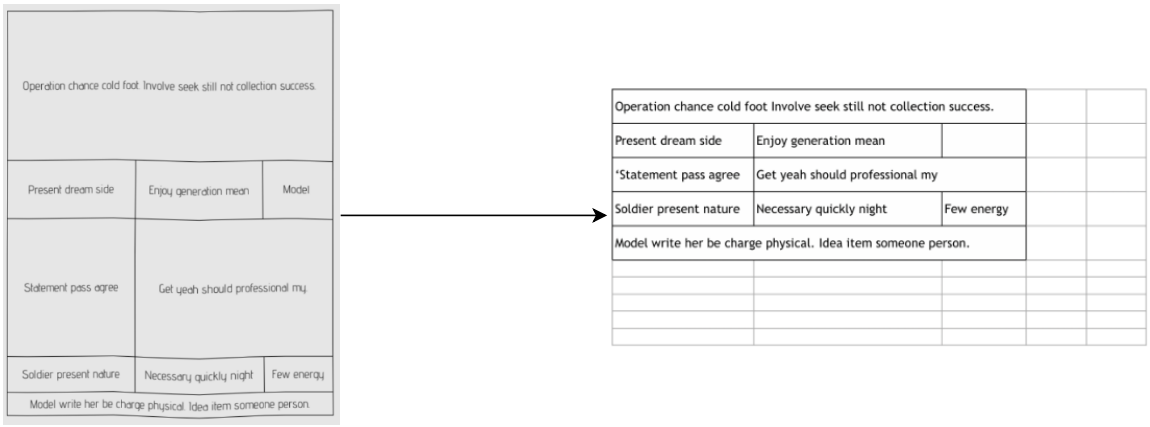
4.3 Vykreslení rozpoznané tabulky do XLSX souboru

Vykreslením tabulky do XLSX souboru se zabývá modul **TableWriter**. V souboru **recognizer.py** program bere pole, které uložil modul **TableReader** a nahrává ho do writeru. Zápis do XLSX souboru se dělá pomoci Python knihovny *xlsxwriter*. Nejprve je potřeba vytvořit **workbook**, což je vlastně xlsx souborem, do kterého se potom vytvoří **worksheet**, což je stránkou, ve které se umisťují tabulky. Zápis je jednoduchý. V cyklusu se probírá každý řádek pole, zatím každý sloupec. Pokud pole "*merged*"v anotaci buňky je nastavené na *True*, a hodnota pole *"merge_r"* nebo *"merge_b"* je odlišná od nuly, vytvoří se *merge_range* s parametry "$row + merge\_b$"a "$col + merge\_r$", jinak vznikne obyčejná buňka. Text buňky se bere z pole *content* v její anotaci. Po vykreslení **workbook** se uloží do složky a zavře se. Celý skript se ukončí.

Ukázka konverze tabulky z obrázku do XLSX je na zobrazeno na 4.13.



Obrázek 4.12: Schematické zobrazení výsledného pole po extrakce tabulky z obrázku.



Obrázek 4.13: Konverze tabulky z obrázku do XLSX souboru.

4.4 Webová aplikace

Tato část kapitoly je zaměřená na vysvětlení postupu vytvoření webového rozhraní pro aplikace, ale kromě toho, tady bude řečeno o vytvoření vazby mezi front-end a back-end aplikace, o webovému serveru, na kterém aplikace bude běžet atd [13].

Webová aplikace byla vytvořená pomoci PHP frameworku Laravel, o kterém bude víc řečeno v sekci 4.5. Aplikace má jenom jednu stránku, na které je místo pro nahrávaní souboru, vytvořené pomoci JavaScript knihovny Dropzone.js[1], a tlačítko "Submit". Na obrázku **??** lze vidět jak tato webová aplikace vypadá. Nahrávač podporuje *multiselect* výběr souboru. Nahrávaný soubor projde základní validaci, která povolí nahrávání jen .JPG a .PNG souborů. Taky se aplikuje validace velikosti nahrávaného souboru. Maximální povolena velikost je 5 MB. Po natisknutí tlačítka "*Submit*", Laravel odešle POST požadavek [6] na router **upload**. Požadavek bude zpracovaný a Laravel uloží soubory do datového úložiště AWS S3[2], o kterém bude řečeno dále.

[1] Dropzone.js: https://www.dropzonejs.com/
[2] AWS S3: https://aws.amazon.com/s3/

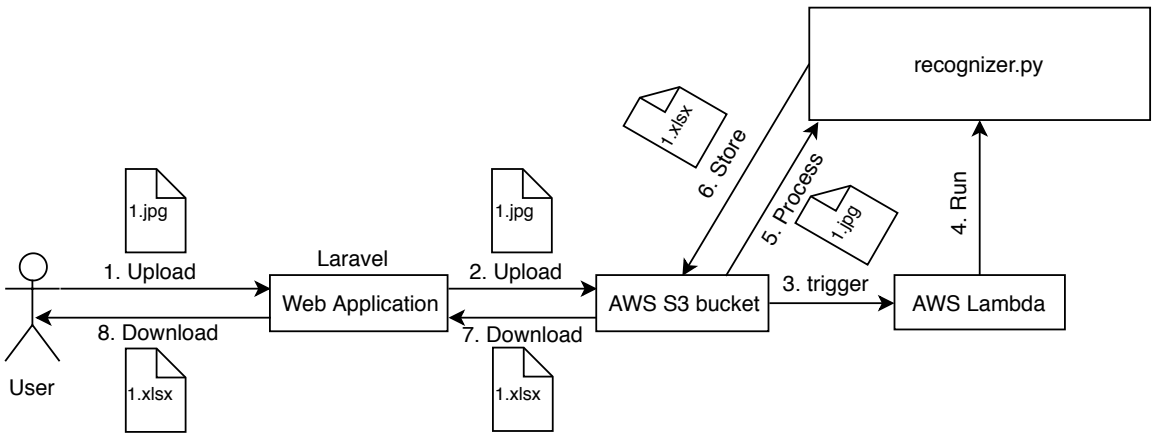
Back-end implementované aplikací je postaven a zpracován na Amazon Web Serices (AWS), což je ekosystém servisu, o který se stará společnost Amazon. AWS je skupina různých servisů zprovozněných v oblaci (*cloud*). Všechny výpočty se dělají v oblaci. Jsou to různé virtuální servery, datové úložiště, databáze, vývojové prostředí. Všechno má vlastní ekosystém a může komunikovat mezi sebou.

Python rozpoznávač tabulek, který byl popsán výše, je taky zprovozněn na serverech AWS. Nejprve byl implementován takový postup, že webová aplikace komunikuje s AWS servisem API Gateway pomoci API, které posílá nahraný obrázek POST požadavkem, a potom zabírá zkonvertovaný výsledek pomoci GET požadavku. API Gateway je prostředník, který přijímá HTTP požadavky a potom posílá je dále na servisy AWS. V praxi tento postup je komplikovaný, protože stojí otázka konzistence poslaných dat, zabezpečení požadavků a taky implementace API ze dvou stran.

Proto byl implementován jiný způsob komunikace mezi serverem a webovou aplikaci. Všechny soubory, nahrávané přes webové rozhraní, jsou nahrané do datového úložiště AWS S3, ve kterém je vytvořen speciální *bucket* pro danou aplikace. V AWS je implementován servis, který se jmenuje "Lambda function"[3]. AWS Lambda dovoluje spouštět kód v závislosti od nějakých přepínačů (*trigger*) bez dodatečného managementu a administraci. To může být například volaní API nebo uložení dat do databáze či souborů do úložiště. V případě dáne práce přepínačem je uložení souboru do databáze, jak už bylo zmíněno výše, v Laravelu bylo implementováno webové rozhraní, které uloží obrázek do AWS S3 úložiště do složky **input**. V AWS byla vytvořena Lambda Funkce, přepínačem které je změna ve složce **input**, případně přidávaní nového souboru. Tato lambda funkce spouští vytvořený kód rozpoznávače, který vezme soubor ze složky **input**, zkonvertuje ho a uloží výsledný XLSX soubor do složky **output**.

Zatím webová aplikace stáhne nové přidaný soubor ze složky **output** a vrátí ho uživatelovi.

Základní postup od nahraní souboru uživatelem do stáhnutí souboru je znázorněn na obrázku 4.14.



Obrázek 4.14: Komunikace $user \longleftrightarrow web \longleftrightarrow AWS$.

[3]AWS Lambda: https://aws.amazon.com/lambda/

4.5 Vývojové nastroje

V prací byl využit programovací jazyk Python[4]. Python je dynamicky typovaný interpretovaný jazyk programovaní, který podporuje objektové orientované, procedurální a funkční styly programovaní. V kontextu dáne práce Python byl využit pro vytvoření aplikaci příkazového řádku.

Pro vytvoření generátoru a rozpoznaní obrázku byly využity knihovny Pillow[5] a OpenCV[6]. Knihovna OpenCV je open-source knihovna, která je zaměřena na řešení úloh počítačového vidění a zpracovaní obrázku. OpenCV je originálně vytvořena společnosti Intel.

Pro detektor úhlů a neuronovou sít byla využita knihovna PyTorch[7].

Knihovna XLSXWriter[8] byla využita pro konverze výsledku rozpoznávače tabulek do formátu XLSX.

Pro implementace webové aplikace byl využit PHP framework Laravel[9]. Laravel je moderní open-source PHP framework, který vyvinul Taylor Otwell. On je postaveny na MVC (model-view-controller) paradigmě a využívá moduly většího PHP frameworku Symphony[10]. A back-end webové aplikace byl umístěn a spuštěn na serverech Amazon Web Services[11].

Vývojové prostředí bylo nastaveno v IDE PyCharm[12] od společnosti JetBrains.

4.6 Budoucí možné rozšířeni a změny

Kvůli časové náročnosti nebyli implementované některé zajímavé funkce aplikace. Tato podkapitola popisuje možný další život aplikace a nápady na implementace nových funkcí.

Prvním možným rozšířením je implementace formuláře a uživatelského rozhraní s různými tlačítky a preview výsledku. To může být například různé přepínače konfigurace aplikace, například, hodnotu *MedianBlur* filtru, při změně kterých, v reálném čase se bude měnit preview výslední tabulky. Aplikace může nabídnout, které tlačítko uživatel může natisknout, aby měl největší vliv na kvalitu výsledku. Tento blok z konfiguraci bude zachován pod rozklikovatko.

Další možnosti je aplikovaní vhodného existujícího algoritmusu pro redukce šumu a rozmazaní obrázku. Nebo detekce stupňů rotace obrázku či sily perspektivy. Zatím aplikovaní homografie na obrázek, což udělá rovné svislé a horizontální čáry, což dál částečné odstraní falešné detekované čáry.

Ještě jedním možným rozšířením může být on-line redaktor výsledku konverze. Například, po konverze se objeví tlačítko "Edit result", po natisknuti kterého se spustí nové okno, kde uživatel bude mít možnost měnit výsledek před jeho stáhnutím. Toto bude vhodné, když aplikace udělá nějaké chyby.

[4]Python language: https://www.python.org/
[5]Pillow: https://pillow.readthedocs.io/en/stable/
[6]OpenCV: https://opencv.org/
[7]PyTorch: https://pytorch.org/
[8]XLSXWriter: https://xlsxwriter.readthedocs.io/
[9]Laravel: https://laravel.com/
[10]Symfony: https://symfony.com/
[11]AWS: https://aws.amazon.com/
[12]https://www.jetbrains.com/pycharm/

4.6.1 Optimizace detektoru rozpoznávání úhlů

V detektoru rozpoznávání úhlů, případné v algoritmusu *sliding window*, je problém, že okno musí procházet cely obrázek po všech pixelech. Proto byl vymyšlen algoritmus pro optimalizaci detektoru uhlu.

Myšlenka algoritmusu spočívá v redukci oblasti, kterými musí procházet posuvné okno. Funguje na principu konečného automatu. V počátečním stavu posuvné okno prochází pixely s nějakým krokem. Zatím, když se neuronová sít řekne, že jsme se narazili na levý horní úhel, přejde do stavu procházeni přímkou. V tomto stavu posuvné okno musí sledovat pohyb přímky od levého horního do pravého horního okna. Po naraženi na levý horní úhel, okno se přesune doprava na jeho velikost, aby neuronová síť neřekla opakovaně, že okno je povrch levého horního úhlu. Zatím, je potřeba v cyklusu projít všechny pixely pravé strany okna a detekovat pozice přímky. Jestli přímka se posunula dolu nebo nahoru od centru, přesunout okno v její směru, aby ona byla přímo v centru pravé strany okna, a opakovat pokud nenarazí na právy horní uhel.

Po naraženi na právy horní úhel, algoritmus přejde do stavu procházeni přímkou, která je pravou stranou tabulky. Postupovat zatím bude podobně horní přímce.

Algoritmus se skonči když se narazí opět na levý horní úhel.

Toto rozšířeni velmi sníží časovou náročnost algoritmusu, a kromě toho, unikne procházeni uvnitř tabulky povrch textu a kříženi přímek, což zvětšuje šancí chybné detekce úhlů.

Kapitola 5

Testováni a porovnáni

V této kapitole je popsané testování aplikace, porovnání s jiným existujícími aplikacemi. Jsou krátké popsané metriky porovnání. Taky zde je uvedeno, jaké tabulky aplikace rozpoznává špatně a co by možné bylo zlepšit. Taky je popsáno testování a porovnání některých častí aplikace, zejména dvou algoritmusu pro odstraňování přímek nad textem.

5.1 Metriky testování

Pro vyhodnocení kvality implementované aplikace je potřeba určit správné metriky testování. První metrikou, která byla využita je obecné porovnání správně vypočtených počtů řádků a sloupců. Pro tento výpočet byla potřeba vytvořit anotační soubor pro každou vygenerovanou a rozpoznanou tabulku. V anotačním souboru v objektu JSON jsou dvě hodnoty: počet řádků a počet sloupců. Potom tyhle dva JSON soubory se navzájem porovnávají. Jestli jsou ekvivalentní – tabulka byla rozpoznaná správně.

Pro vyhodnocení úspěšnosti detekce se často používá *precision/recall*, které se používali pro ocenění kvality detekce přímek. Záleží na počtu falešně positivních, správné positivních a falešné negativních přímek.

Hodnota *precision* je podíl relevantních výsledků mezi všemi výsledky, které metoda označila jako relevantní 5.1.

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{5.1}$$

Hodnota *recall* je podíl relevantních výsledků mezi všemi, které metoda musí označit jako relevantní 5.2.

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{5.2}$$

5.2 Dosazené výsledky

V průběhu vývoji aplikaci byli stanovené některé data, na kterých aplikace funguje docela spolehlivě, ale jsou data, na kterých aplikace hodí výjimky, špatně rozpoznává text, nebo nesprávně detekuje počet sloupců či řádků. Výsledek záleží na správném výběru parametrů pro *MedianBlur* filter potřebný pro rozpoznávání textových bloků a na stupňů deformace vstupního obrázku (blur, šumy, perspektiva).

5.2.1 Žádný blur efekt a žádná geometrická deformace

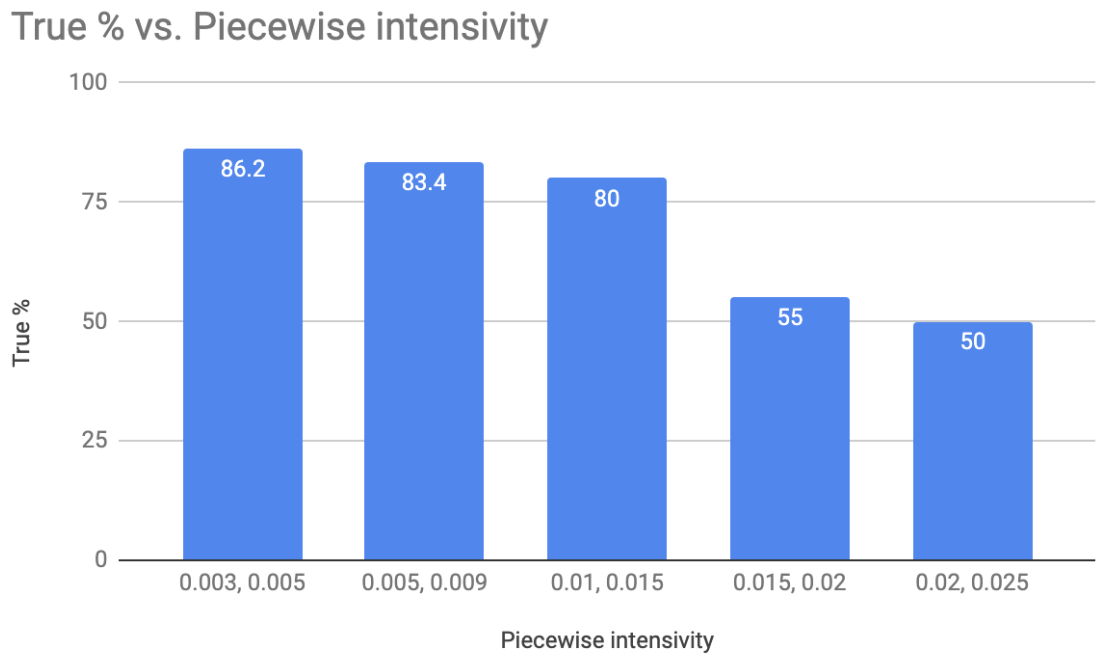
Nejprve bylo provedeno testování na vygenerovaných tabulkách, které nemají ani žádnou deformace, ani žádný blur efekt. Rozpoznávání takové vygenerované tabulky lze porovnat z rozpoznáváním ideálně rovné tabulky, například v pdf dokumentu.

Nejprve testování bylo provedeno na porovnání počtu řádků a sloupců vygenerovaných a rozpoznaných tabulek. Výpočet procentu správně rozpoznaných počtu řádků a sloupců byl proveden na datasetu z 10, 50, 100, a 500 tabulek. Střední procent úspěchu je 85%.

5.2.2 Žádný blur efekt a minimální geometrická deformace

Druhou testovací datovou sadou byly tabulky, které nemají žádný blur rozmazání, ale je aplikovaná geometrická transformace *piecewise*. Testovací sada byla pozbyta na pět tříd, která se liší intensitou deformace. Pro každou třídu intensita se nastavuje náhodně v nějakých zadaných v konfiguraci mezích. Na obrázku 5.1 lze vidět výsledek testování úspěšnosti aplikace pro každou třídu. Hodnota *piecewise* nastavena na 0.025 (poslední sloupec) tvoří silnou deformaci obrázku. Příklad rozpoznávání je na obrázku 5.2. Nejvíc deformovaná je dolní přímka, ale rozpoznávač ji správně normalizoval.

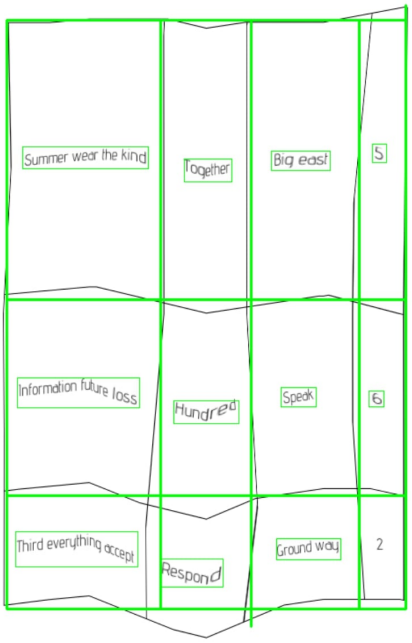
Výsledek rozpoznávání 50% pro takovou augmentaci přijde docela dobrý.



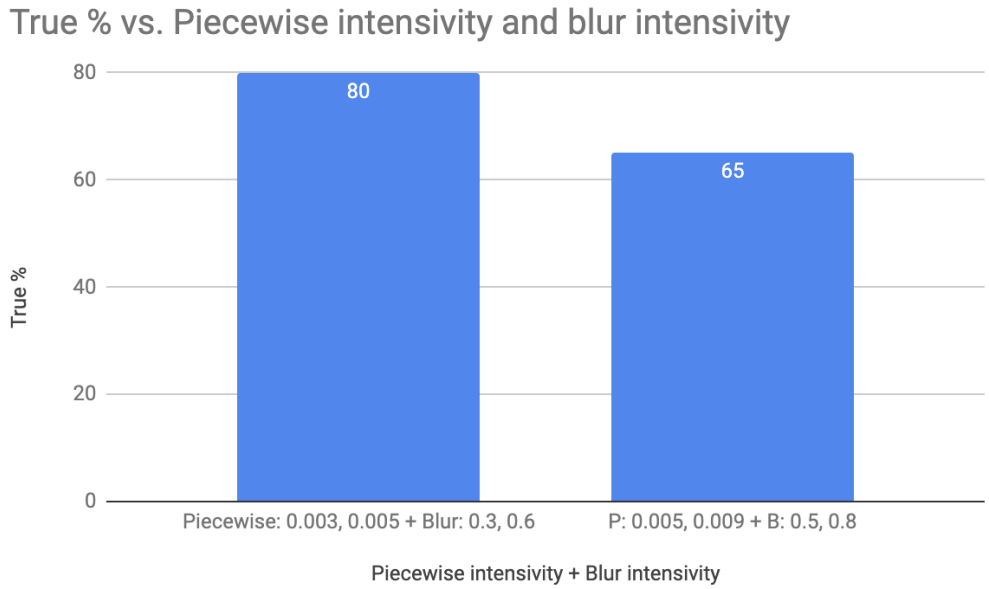
Obrázek 5.1: Procent správně detekce v závislosti od intensity deformaci *piecewise*.

5.2.3 Blur efekt a geometrická deformace

Nejhorší případ, na kterém byla testována dána aplikace je geometrická deformace a ve stejný čas blur rozmazáni. Příklad normalizace primitiv je na obrázku 5.4. Výsledek je na grafu 5.3. Víc než 50% úspěšnosti na výstupu je jenom na první a druhé datové třídě z minulého příkladu. Na ostatních třídách výsledek se považuje jako nedostatečně kvalitní.



Obrázek 5.2: Rozpoznávání a normalizace primitiv na obrázku s silnou *piecewise* deformaci.



Obrázek 5.3: Procent správné detekci v závislosti od intensity deformaci *piecewise* a aplikováni blur efektu.

Obrázek 5.4: Rozpoznávání a normalizace primitiv na obrázku s deformaci *piecewise* a *blur* rozmazáním.

Kapitola 6

Závěr

Cílem této práce bylo implementovat program, který povolí rozpoznávat tabulky v obrazcích a konvertovat je do XLSX podoby. V průběhu implementace programu byla potřeba vytvořit generátor náhodných tabulek s anotaci pro provedení testování. Na konci implementace rozpoznávače se objevil problém, pro řešení kterého byla potřeba lokalizovat tabulku na obrázku a oddělit ji od okraji papíru a od jiných objektů. Proto byl vytvořen detektor úhlů tabulky, založeny na neuronové síti. Taky byla potřeba vytvořit webovou aplikaci a rozmístit back-end práce na serverech Amazon.

Generátor tabulek, detektor úhlů a okrajů, rozpoznávač jsou vytvořené v jazyce Python. Webová stránka byla implementovaná v PHP.

Testování rozpoznávače tabulek na vygenerovaném datasetu ukázalo, že a i při docela velkých deformacích vstupního obrázku, program splňuje požadavky na 50%. Při testování na malých deformacích a rozmazání, což současní smartfony můžou dokázat, úspěšnost je kolem 80-85%.

Je mnoho variant rozšíření dáne aplikace a přidaní ji nových vlastností. O možných budoucích rozšířeních je napsáno v kapitole 4.6.

Literatura

[1] Agrawal, R.: *Orientation of 3 ordered points*. [Online; navštíveno 27.05.2019]. URL https://www.geeksforgeeks.org/orientation-3-ordered-points

[2] Bettinger, F.: Probabilistic Hough transform. [Online; navštíveno 15.04.2019]. URL http://phdfb1.free.fr/robot/mscthesis/node14.html

[3] Bradski, G.; Kaehler, A.: *Learning OpenCV; Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[4] Gonzalez, R.; Woods, R.: *Digital Image Processing*. Addison-Wesley Publishing Company, 1992, ISBN 0-201-18075-8.

[5] Jung, A.: Knihovna Imgaug. [Online; navštíveno 10.06.2019]. URL https://github.com/aleju/imgaug

[6] Krug, S.: *Don't Make Me Think: A Common Sense Approach to Web Usability*. New Riders Press, 2000, ISBN 978-0321965516.

[7] Mazura, M.: *A Step by Step Backpropagation Example*. Březen 2015, [Online; navštíveno 15.06.2019]. URL https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[8] Miller, S.: *Mind: How to Build a Neural Network*. Srpen 2015, [Online; navštíveno 10.06.2019]. URL http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

[9] Owens, R.: Computer Vision IT412. Lecture 6. [Online; navštíveno 10.04.2019]. URL http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT6/node3.html

[10] Sorokin, A.; Zapriahaiev, S.: *Recognition of simple curves on the image*. Voronezh State University, Voronezh, 2009, ISBN 5457385671.

[11] Szeliski, R.: *Computer Vision: Algorithms and Applications*. Springer, 2011.

[12] Vaseekaran, G.: *Machine Learning: Supervised Learning vs Unsupervised Learning*. Září 2018, [Online; navštíveno 15.06.2019]. URL https://medium.com/@gowthamy/machine-learning-supervised-learning-vs-unsupervised-learning-f1658e12a780

[13] Řezáč, J.: *Web ostrý jako břitva*. Baroque Partners, 2014.

Příloha A

Obsah přiloženého paměťového média

table_generator – generátor tabulek

detector – detektor úhlů tabulek

table_reader – konvertor tabulek z obrázku do XLSX formátu

table_reader_www – webová stránka projektu

doc – přiložené PDF a zdrojové soubory telnické dokumentace projektu

video – obsahuje video pro generovaní tabulek, detekce uhlu a konverze tabulek

README.md – soubor s podrobnějším popisem složek na disku DVD a návod jak s programem pracovat

reqs.txt – potřebné knihovny pro spouštěni aplikace