Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Neuronové sítě a zpracování skenovaných dokumentů

2021

Bc. Libor Machálek

Vedoucí práce:
Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Bc. Libor Machálek
Název práce:	Neuronové sítě a zpracování skenovaných dokumentů
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	42
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Bc. Libor Machálek
Title:	Neural networks and automatic processing of scanned documents
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	42
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Tato práce se zabývá zpracováním skenovaných historických dokumentů. Využívá poznatky ze zpracování obrazu, neuronových sítí a komprese obrazu. Součástí práce je konzolová aplikace pro dávkové zpracování naskenovaných dokumentů, která se bude využívat na pracovišti digitalizační jednotky Vědecké knihovny v Olomouci.

Synopsis

This thesis deals with the processing of scanned historical documents. Uses knowledges from image processing, neural networks and image compression. Part of the thesis is a console application for batch processing of scanned documents, which will be used at the workplace of the digitization unit of the Research Library in Olomouc.

Klíčová slova: zpracování obrazu; neuronové sítě; ořez obrazu; digitalizace; Vědecká knihovna

Keywords: image processing; neural network; image crop; Research Library

Děkuji vedoucímu této diplomové práce Mgr. Petru Osičkovi, Ph.D. a RNDr. Eduardu Bartlovi, Ph.D. za jejich cenné rady a konzultace, které mi pomohly tuto práci zlepšit.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Úvod

Od pradávna jsou lidé fascinováni zachycováním reality na nějaké médium, ať už se jedná o pravěkou jeskyni, lněná plátna renesančních mistrů anebo CCD čidlo pomocí nehož zachytíme realitu a reprezentujeme ji jako počítačový soubor.

V této práci se zaměříme na poslední jmenované. Již na přednáškách kurzu „Algoritmy pro zpracování obrazu“ jsem si uvědomil obrovský potenciál zpracování obrazu počítačem. Počítačová reprezentace skrývá oproti ostatním několik výhod, z nichž hlavní je ta, že se dá upravovat a různě vylepšovat. Prozkoumáme možnosti úprav jako jsou operace ořezu, rotace, opravy deformací a spojování obrazu. Pro účely této práce je nutné porozumnět základům reprezentace obrazu a její uložení v počítači tak, jak je definují Rafael C. Gonzalez a Richard E. Woods v 2. kapitole své knihy [1]. Dále se podíváme na stále více populární neuronové sítě a jak je využít v problematice jejíž obsahem je náplň této práce. V neposlední řadě se budeme zabývat kompresí výsledných souborů podle standardů Národní digitální knihovny.

Veškeré zde prezentované obrázky jsou poskytnuty digitalizační jednotkou Vědecké knihovny v Olomouci (zkráceně VKOL), na jejíž návrh vznikla tato práce. Dané pracoviště se zabývá digitalizací starých dokumentů, které je po naskenování potřeba v počítači dále upravovat. Jedná se o repetetivní práci, jíž se pokusíme v následujících kapitolách textu převést na automatické zpracování počítačem.

Diplomová práce obsahuje jak teoretickou část (kapitoly 3, 4 a 5), kde je popsán návrh řešení této problematiky, tak implementační část, která převádí do praxe nastíněná řešení navrhnutá v teoretické části. Z předchozí věty vyplývá, že práce obsahuje vypracované řešení problému (v podobě konzolové aplikace), které by mělo být nasazeno ve Vědecké knihovně (v Olomouci). V uživatelské části popíšeme, jak aplikace funguje a jak se ovládá z pohledu uživatele.

2 Vstupní soubory

V úvodu jsme se dozvěděli, co budeme s digitalizovanými dokumenty provádět. Víme však s jakými daty máme tu čest? Odpověď na tuto otázku je jednoduchá. Typově se jedná o naskenované historické dokumenty vytvořené v rámci pracovní náplně oddělení digitalizační jednotky Vědecké knihovny v Olomouci. Mezi zpracovávané dokumenty řadíme hlavně noviny, manuskripty, různá periodika nebo třeba knihy. Není však vyloučeno, že v budoucnu se mohou objevit ke zpracování další druhy snímků jako jsou např. historické mapy a jiné.

2.1 Formát vstupních souborů

Naskenované dokumenty jsou uloženy jako TIFF. Ten byl původně navržen pro ukládání obrazů vytvořených skenováním. Což v našem případě vypadá jako ideální formát našich vstupních souborů, protože soubory jsou snímány automatickým skenerem[1]. Dnes je tento formát, používán obecněji [2].

Jednotlivé snímky jsou uloženy s ICC bez komprese, s 24bitovou barevnou hloubkou v prostoru RGB. Rozlišení takového skenu je obvykle 300 DPI, ale může se stát, že bude i větší. Obrázky, které byly pro účel této práce poskytnuty, jsou seskupeny do početných kolekcí (datasetů), čítajících někdy i přes 1200 nekomprimovaných souborů. Proto je potřeba se dostatečně předzásobit volným místem na pevném disku.

2.2 Vlastnosti a limitace datasetů

V této podkapitole si představíme na pár příkladech, čeho se budou týkat výše zmiňované operace, které budeme nad snímky aplikovat. Dále si ukážeme limitace, které s sebou datasety nesou.

Začněme meritem celé práce, čímž je bezpochyby ořez. Je nutné abychom věděli, jaké části budeme ořezávat a co potřebujeme zachovat. Při procesu skenování se zpravidla stává, že kromě skenovaného dokumentu nasnímáme zároveň:

- část předchozí strany,
- okraje nebo vazbu knihy,
- podložku, na které leží skenovaný dokument,
- různé předměty ležící na snímací ploše,
- tmavý okraj, vznikající naskenováním prázdné snímací plochy.

Tyto části jsou v požadovaném výsledném snímku nežádoucí a je potřeba jejich oříznutí. Výše uvedený výčet „objektů k odstranění“ je zároveň první komplikací bezproblémovému ořezu.

[1] pripcip jeho funkce viz https://www.youtube.com/watch?v=CDDzpX2UABQ

Jednou z vlastností poskytnutých obrázků je rozdílná velikost v rámci jednoho datasetu[2]. Tato skutečnost vznikla pravděpodobně při snímání. Když byly skenovány dokumenty kratšího rozsahu pro daný rok (nebo více let) a v průběhu tohoto období se změnila velikost papíru, na kterém je text vytištěn.

U některých dokumentů se nevyskytují vnitřní okraje. Text je na takových stránkách vytištěn až po vnitřní nebo i vnější okraj stránky. Může se jednat o vcelku vážný problém při detekci stránek, kdy je nebudeme schopni od sebe odlišit.

Kromě nechtěných (nevyhnutelných) „artefaktů“ vytvořených při snímání předlohy se v datasetech vyskytují i další problémy. Většina z nich vychází ze samé podstaty skenovaných tiskovin. V některých případech se jedná o poměrně staré dokumenty, které byly vytištěny začátkem předchozího století[3]. Tomu také odpovídá jejich technický stav, se kterým si budeme muset poradit. Může se stát, že snímek bude neúplný[4], anebo jinak poškozený[5]. Jak si s těmito skutečnostmni poradíme budeme řešit v kapitole 3.

Na konec je potřeba zmínit, že některé datasety jsou tvořeny skeny rozsáhlejších knih. Nasnímané strany jsou otočeny o úhel zvětšující se s počtem otočených stran. Problém vzniká již při skenování snímku, kdy automatický skener otáčí strany rozsáhlejší knihy a aktuální strana nestačí překrýt předchozí. Tohle uskalí je poměrně jednoduše řešitelné, jak uvidíme v podkapitole 4.1.

Poznámka 1 (Komentář k obrázku 1)

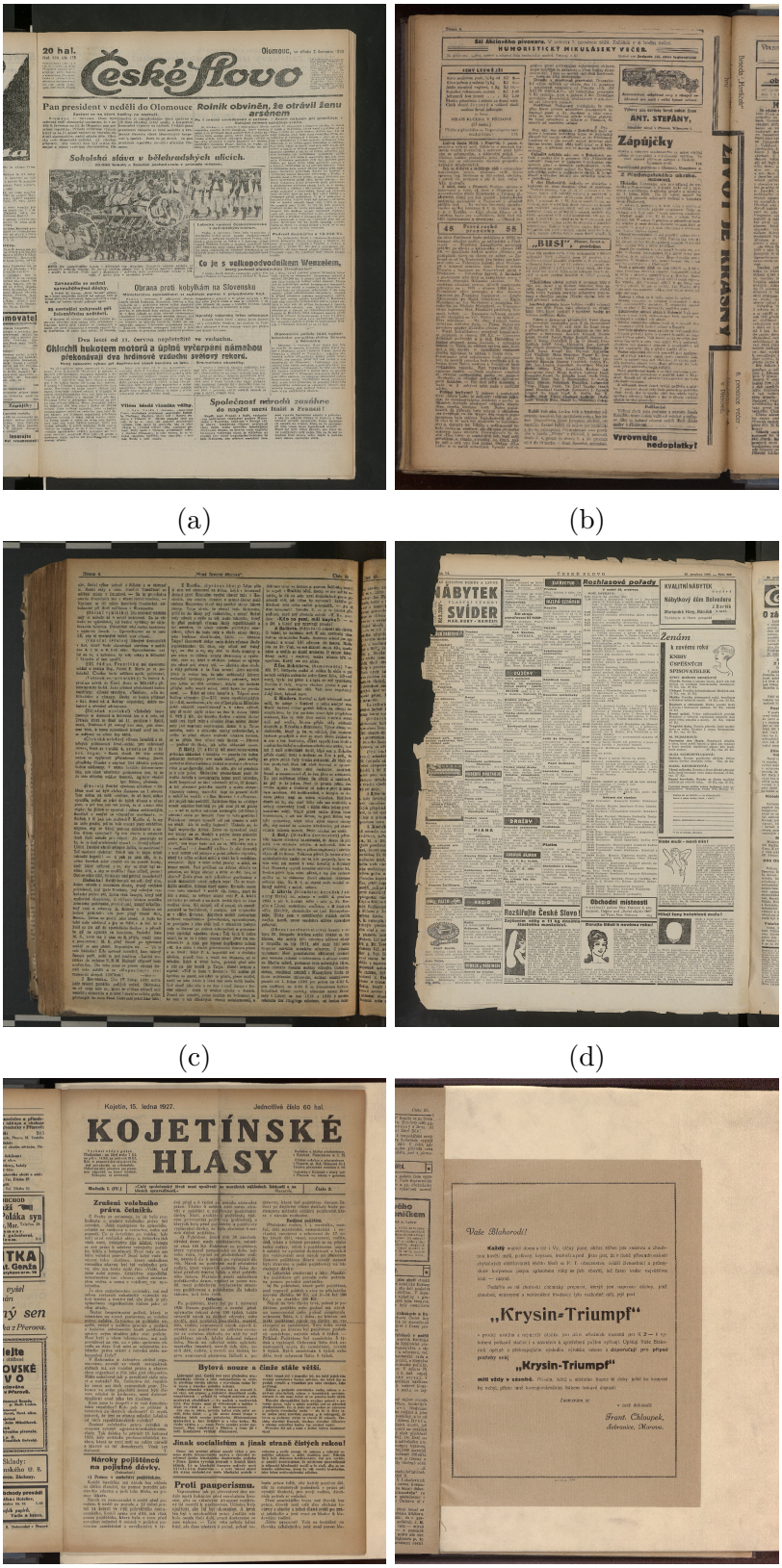
Na následující skupině obrázků můžeme pozorovat některá úskalí popsaná výše v této podkapitole. Snímek 1a obsahuje tmavé okraje způsobené nasnímaním skenované plochy. Vyskytuje se zde i problém s vnitřními okraji předchozí strany knihy. Na snímku 1b je znatelný přesah textu (rámečku) na druhou stranu. Na okrajích si můžeme všimnout modrých desek knihy, do které jsou listy svázány. Obrázek 1c má jasně oddělenou levou a pravou stranu. Můžeme si všimnout, že kniha je otevřena pravděpodobně v polovině a jsou zde viditelné předchozí již naskenované stránky. Tím pádem bude snímek s největší pravděpodobností pootočen a bude potřeba jej narovnat. Zároveň obsahuje vzor připomínající šachovnici. Nejspíš se bude jednat součást skenovací podložky automatického skeneru, kterou však budeme muset také oříznout. Snímek 1e má poškozený vnější okraj, což může způsobit komplikace při dalším zpracování. Další ze snímků 1e se zdá být v pořádku. Můžeme si však všimnout čáry, vzniklé pravděpodobně přeložením novin v polovině. To by mohl problém při správné detekci strany, která nás zajímá. Poslední snímek 1f má viditelně menší rozměry stránky než předchozí a následující strana.

[2]Dataset poznáme tak, že obrázky jsou umístěny v jednom adresáři a jsou pojmenovány stejnou číselnou řadou např. 0001.tif – 1200.tif.

[3]Poskytuné datasety jsou vytištěny mezi lety 1907 – 1935.

[4]kus stránky bude utržen.

[5]politý nějakým nápojem či jinak ušpiněný



(a) (b)

(c) (d)

(e) (f)

Obrázek 1: Ukázky naskenovaných snímků z různých datasetů

3 Ořez obrazu

Dostáváme se do bodu, kdy víme jaká máme data. Známe všechny jejich vlastnosti a problémy, které s nimi mohou nastat. Také víme co máme s daty provádět.

V této kapitole si přiblížíme několik metod, které by se mohly dát využít při samotném ořezu snímků. Popíšeme si jednotlivé metody od nejjednodušších až po ty složitější. Vysvětlíme jejich hlavní myšlenku, pojmenujeme výhody, nevýhody a vyhodnotíme, která metoda je pro dostupné datasety nejvhodnější.

3.1 Klasické metody

Prvním řešením, které se nabízí jako schůdné, je použití způsobů detekce hran, jako je definováno a popsáno v [1], kapitola 10. Tedy převést si snímek na šedotónový obrázek. Nad tímto obrázkem provést detekci hran. Pro tento úkon můžeme využít jeden z následujících způsobů:

1. Sobelův operátor (viz matice S_x a S_y),
2. Laplaceův operátor (viz matice L),
3. případně pokročilejším algoritmem (např. Canny).

Je potřeba vybrat si jeden z těchto způsobů. Rozdíly mezi jednotlivými způsoby jsou podrobně popsány v [1]. Jelikož víme, že obraz je ve své podstatě pole obrazových bodů, bude nám stačit vědět že musíme iterativně nad tímto polem posouvat okýnko (tzv. konvoluční jádro) a pomocí operace konvoluce, která je definována v [1] jako operace úzce související s korelací, budeme upravovat aktuální obrazový bod na základě jeho okolí[6]. Korelaci funkcí w a f v bodě (x, y) definujeme následovně:

$$w(x,y) \circ f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) \cdot f(x+s, y+t).$$

Jedná se tedy o odezvu filtru w na obraz f v bodě (x, y), jinak řečeno korelace je proces posouvání filtru (okýnka) po obraze a výpočet sumy součinů na každé pozici. Konvoluce je pak korelací s filtrem otočeným o 180°. Definujeme ji jako operaci funkcí w a f v bodě (x, y):

$$w(x,y) \bullet f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) \cdot f(x-s, y-t).$$

Konvoluční jádro posouvané po obraze se liší podle použitého operátoru následovně:

[6] Okolím jsou myšleny sousední pixely tak, jako jsou definovány v [1]

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \; S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \; L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Můžeme si všimnout, že Sobelův operátor můžeme reprezentovat pomocí dvou matic (viz S_x a S_y). Je tomu tak, protože je možné pomocí něj samostatně detekovat jak vodorovné (matice S_x), tak svislé hrany (matice S_y). Pomocí Laplaciánu dokážeme tyto hrany detekovat naráz. Operátory můžeme upravit i pro detekci diagonálních hran. Toto by však pro účely této práce nemělo být potřeba, jelikož hrany dokumentu jsou pouze svislé a vodorovné.

3.1.1 Cannyho hranový detektor

Jako zástupce pokročilejších algoritmů byl vybrán Cannyho hranový algoritmus[7], který vyvinul John F. Canny v roce 1986. Algoritmus je hojně využíván v mnoha aplikacích. Skládá se z následujících kroků:

1. Vyhladíme vstupní obrázek pomocí Gausova filtru.
2. Vypočítáme gradient obrázku.
3. Nalezneme lokální maxima.
4. Pomocí dvojitého prahování odstraníme nevýznamné hrany.

V prvním kroku potřebujeme odstranit případný šum – pomocí Gausova filtru. Následně provádíme výpočet gradientu obrázku. Gradient je vektor určující velikost a směr intenzit v obrázku. Pro získání gradientu můžeme použít třeba Sobelův operátor.

S tímto gradientem pak dále provádíme proces ztenčení (tzv. thining), který je založen na hledání lokálních maxim. Nalezneme lokální maximum (tzv. non-maximum suppression) a všechny okolní body, které jsou menší odebereme. Tím zajistíme zobrazení detekované hrany pouze v místě největšího gradientu.

Dvojitým prahováním z posledního kroku, je myšleno odfiltrování nevýznamných hran. Toho docílíme zvolením dvou prahů – vyšším a nižším. Nejdříve nalezneme pixely výrazných hran (vyšších než vysoký práh), následně rekuzivně hledáme pixely, které jsou vyšší než nízký práh. Takovýto způsob nazýváme prahováním s hysterzí.

3.1.2 Pokračování v hledání hran

V tomto okamžiku nám vznikne obrázek zaznamenávající hodnotu intezit pixelů, kde vyšší intenzita má vyšší hodnotu (barvy). Jinými slovy vzniká obrázek, který znázorňuje hrany. Čím je hrana výraznější, tím více je znázorněna (bílou barvou). Nad tímto obrázkem je potřeba provést filtraci nepotřebných intenzit. Tím nám

[7] viz [1], podkapitola 10.2.6

vznikne tzv. „bounding box“, tedy ohraničující rámeček, značící okraje konkrétní stránky, která nás zajímá. Zbývá už jenom oříznout zpracovávaný snímek podle tohoto rámečku.

3.1.3 Výhody a problémy

Výhodou výše popsaného způsobu ořezu je jednoznačně jednoduchost. Ke správné detekci bounding boxu je prakticky potřeba zvolit správný algoritmus detekce a správně odfiltrovat nechtěné intenzity. Co by tedy mohlo být kamenem úrazu tohoto způsobu?

Ze znalostí nabytých v předchozím textu, můžeme jednoznačně určit, že zmiňovaný způsob nemůže spolehlivě fungovat pro každý obrázek z jakéhokoliv datasetu. Jak již bylo nastíněno v kapitole 2, datasety jsou od sebe odlišné a je otázkou času, kdy se onen „kámen úrazu“ projeví. Slabinou je samotná filtrace nepotřebných intezit. Rozdílné obrázky potřebují rozdílné prahy filtrovaných intenzit. Jsou-li prahy zvoleny nevhodně obrázek se ořízne buď příliš a odstraní včetně nechtěných části (popsaných výše) i kus naší ROI[8], anebo se do výsledku promítnou i části původního obrazu, které tam nemají být. Jenomže v našem případě máme prahy pevně dané. A jelikož se mi nepodařilo navrhnout způsob hledání adaptivní filtrace, ke které není nutný zásah uživatele, bude potřeba promyslet lepší způsob ořezu. Možným řešením by zde bylo zadání prahů uživatelem. Tato možnost však není reálná, jelikož jedním z požadavků na aplikaci je, že musí běžet na serveru, bez zásahu uživatele. Ten musí být pouze iniciátorem celého procesu dávkového zpracování souborů.

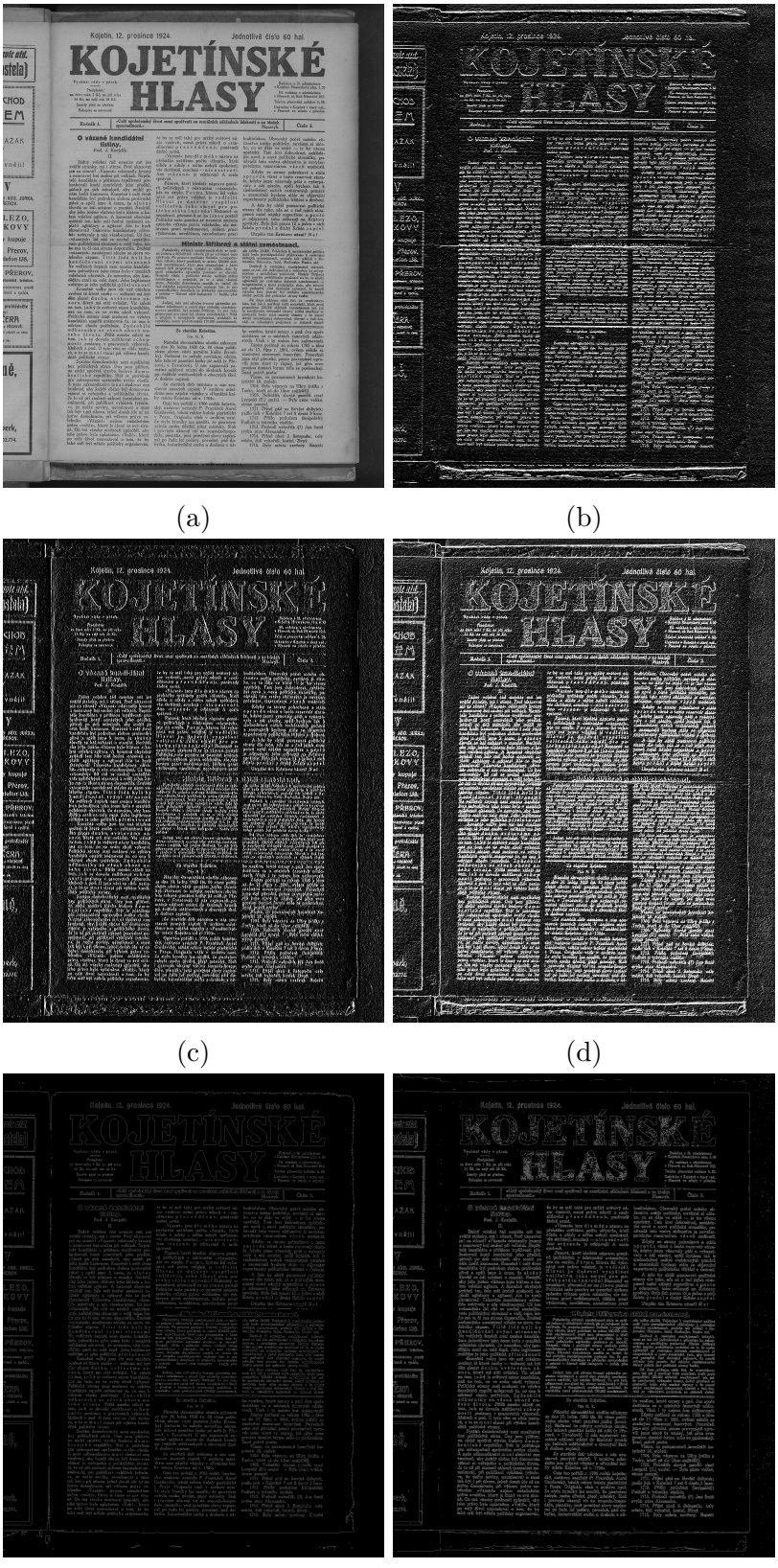
Poznámka 2 (Komentář k obrázku 2)

Na následujícím obrázku můžeme porovnat jednotlivé metody detekce hran mezi sebou. Obrázek 2a si označme jako výchozí pro tuto ukázku. Jedná se o náhodně vybraný soubor z datasetu novin „Kojetínských hlasů“, převedený na šedotónový obraz. Na ukázce vpravo od něj 2b je výchozí obraz s aplikovaným Sobelovým „horizontálním“ filtrem. Můžeme si všimnout, že obsahuje hlavně vodorovné hrany. Oproti tomu 2c po aplikaci Sobelova „vertikálního“ filtru obsahuje hrany svislé. Obrázek 2d je pak sjednocením přechozích dvou (tedy 2b a 2c). Předposlední obrázek 2e je výsledkem aplikace Laplaciánu na výchozí obrázek. Jako poslední můžeme vidět 2f, kde jsou hrany detekovány pomocí Cannyho algoritmu.

3.1.4 Detekce s využitím průměrného obrázku

Tato metoda detekce je vylepšením způsobu z podkapitoly 3.1. Metoda je založena na myšlence, počtu souborů v datasetu. Na základě dostatečně početného datasetu můžeme postupnou iterací přes všechny soubory vytvořit „průměrný obrázek“. Ten v sobě eviduje všechny hrany každého obrázku v datasetu. Zjistíme

[8] ROI = region of interest; neboli oblast zájmu tak, jak je definovaná v [1]



Obrázek 2: Použití způsobů k nalezení hran

z něj tedy, kam na daných snímcích z datasetu sahá text a kde jsou okraje jednotlivých dokumentů. Tím pádem můžeme poměrně jednoduše vytvořit masku průměrného obrázku, na jejímž základě budeme schopni oříznout jednotlivé obrázky z konktrétního datasetu. Metoda byla pro účely této práce navržena RNDr. Eduardem Bartlem, Ph.D. k realizaci a byla s ním i diskutována. Jedná se vskutku o elegantní řešení potíží s filtrací u klasických metod.

3.1.4.1 Úskalí využití průměrného obrázku

Toto řešení předpokládá, že všechny obrázky v datasetu budou obsahovat „rozumná data“. Tím je myšleno, že každý naskenovaný snímek bude:

- mít v rámci datasetu stejné rozměry,
- téměř ze 100 % překrývat stranu předchozí[9],
- rotován o stejný úhel[10],
- na stejné pozici jak předchozí snímek,
- bude mít mezeru mezi textem a libovolným okrajem.

Při znalosti poskytnutých datasetů a jejich limitací však víme, že podmínky se kterými tato metoda počítá jsou zřídkakdy splněny. Další úskalí, na které jsem přišel, je samotný rozsah datasetů. Máme-li v datasetu příliš málo souborů (např. deset), průměrný obrázek se liší jen minimálně od obrázku intenzit získaným klasickým způsobem. A nedokáže vyřešit potíže popsané v podkapitole 3.1.3. Naopak čím více snímků dataset má, tím víc bude průměrný obrázek obsahovat hran, ze kterých nebude možné nic vyčíst. Mezi jednu vlastnost datasetů patří, že obsahuje jak levé, tak pravé strany skenovaných stran knih. Je důležité při implementaci na tuto skutečnost myslet a odlišit od sebe levou a pravou masku.

Poznámka 3 (Komentář k obrázku 3)
Na tomto obrázku vidíme tři maskované obrázky. Jejich maska, která je vyznačena červeně byla vytvořena na základě průměrného obrázku. Jak lze z obrázku vyčíst, při zvyšujícím se počtu souborů v datasetu označeném jako D_n se zvětšuje i maska. Experimentálně bylo zjištěno, že pokud dataset obsahuje více než 100 souborů, nemůžeme ji použít, protože existuje značná pravděpodobnost odříznutí části textu. Na obrázku 3a je zachycen vygenerovaný průměrný obrázek při počtu $D_n = 50$. Podle masky 3b vytvořené na jeho základě můžeme konstatovat, že tento počet je nedostačující.

[9] u novin, knih nebo jiných rozsáhlejších tiskovin
[10] pokud se rotace v rámci datasetu bude vyskytovat

(a) $D_n = 50$. (b) $D_n = 50$. (c) $D_n = 100$. (d) $D_n = 300$.

Obrázek 3: Průměrný a maskované obrázky, kde D_n značí počet souborů v datasetu.

3.1.5 Detekce nejdelší vertikální hrany

Jelikož metody dosud popsané v podkapitole 3.1 nejsou spolehlivě funkční a v nejlepším případě se často stává, že ve výsledném obrázku se promítne neúplná část předchozí stránky, musíme hledat další způsoby detekce. Toto může být způsobeno neostrým přechodem mezi stránkami. Metody popsané v 3.1 nemusí rozeznat úplnost čáry, jelikož detekovaná hrana je příliš tenká a přerušovaná. Hlavní myšlenkou je, pokusit se navrhnout způsob, který detekuje nejdelší hranu. Ta by měla ve většině případů být detekována u vnitřního okraje knihy. Od této hrany si vytvoříme dva bounding boxy, které porovnáme a ten s větším obsahem bude pravděpodobně náš ROI.

Tato metoda vypadala ze začátku slibně, dokud nebyla vyzkoušena na různých datasetech. Na těch se její správnost nepotvrdila.

3.2 Neuronové sítě

Dostali jsme se do stavu, kdy veškeré klasické metody nedokázali správně detekovat hrany. Budeme potřebovat nějaký jiný způsob jak dosáhnout cíle bez použití metod popsaných výše. Takový způsob nám poskytne výpočetní modul zvaný - neuronové sítě. Jenže o co se vlastně jedná a jak fungují?

Umělé neuronové sítě vychází z napodobení těch biologických v mozku. Základním „výpočetní jednotkou“ (tedy stavebním kamenem) každé sítě je neuron. Neurony mají několik druhů spojení, kterými dostávají a posílají signály. Vstupních spojení[11] může být mnoho, zatímco výstupní spojení[12] může být jen jedno [3]. Samotný výpočet probíhá vevnitř neuronu, který přijme vstupy a jejich hodnoty vynásobí váhami. Následně tyto součiny sečte a pokud je výsledek větší než stanovený práh, tak se výsledek transformuje předem danou přenosovou funkcí a pošle na výstup [4]. Propojením několika neuronů (jejich výstupů se vstupy

[11] paralela s dendrity u biologického neuronu
[12] paralela s axonovým vláknem u biologického neuronu

jiných neuronů) vznikne neuronová síť. Může se stát, že neurony utvoří v rámci jedné sítě vrstvu. Ta se vyznačuje tím, že neurony v jedné vrstvě nejsou propojeny navzájem, ale jsou spojeny se sousední vrstvou. Pro účely této práce budeme potřebovat druh neuronové sítě, která se nazývá konvoluční[5, 6]. Konvoluční neuronové sítě je vhodné zvolit jako možné řešení problematiky, kterou zkoumá tato práce.

3.2.1 Konvoluční neuronové sítě

Nejčastěji se tento druh neuronové sítě používá pro klasifikaci. Svůj název získala podle využití konvolučních a pooling vrstev. Díky nimž je síť schopna efektivně zpracovávat vstupy velkých rozměrů [5]. Architektura sítě je tvořena z několika vrstev s různými funkcemi [7]. Princip lze zjednodušeně shrnout tak, že pomocí konvolučních vrstev (kde se aplikuje výše popsaná operace konvoluce na obraz) a aplikací Pooling vrstev, které snižují velikost reprezentace, síť utvoří aktivační mapu. Z té se neuronová síť prostřednictvím plně propojené vrstvy (tzv. Fully-connected layer) snaží o klasifikaci.

3.2.2 Učení neuronové sítě

Z předchozího odstavce víme jak neuron pracuje a jak vytvořit neuronovou síť. Teď ji potřebujeme ještě něco naučit. Tento proces lze provést dvěma způsoby:

- učení s učitelem,
- učení bez učitele.

První z nich (učení s učitelem) probíhá tak, že neuronové síti na vstup předložíme nějakou množinu vstupů a požadovaných výstupu. Vstupy se pustí do sítě, výstup se porovná s požadovaným výstupem [4]. Pokud výstup neuronové sítě neodpovídá požadovanému výstupu, provedeme korekci sítě. To znamená, že je potřeba upravit hodnoty vah a prahů). Cílem tohoto procesu je, aby byl rozdíl mezi výstupem ze sítě a požadovaným výstupem co nejmenší. V případě učení bez učitele známe vstupy, ale již ne požadovaný výstup. Sítě, u kterých se používá tento typ učení, mají většinou za úkol provádět tzv. shlukovou analýzu, tedy rozškatulkování zadané množiny vstupů do konečného množství tříd [4].

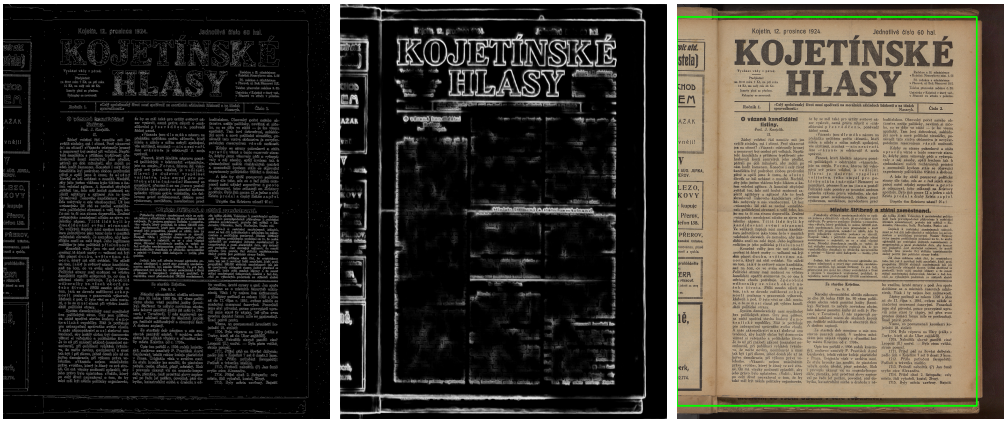
3.2.3 Detekce hran pomocí neuronových sítí

Jelikož až dosud každý pokus nalézt hrany v obrázku klasickým způsobem selhal. Můžeme vytvořit neuronovou síť, která bude hrany detekovat. Pokusíme se implementovat způsob zvaný „**H**ollisticaly-Nested **E**dge **D**etection“ (HED), který navrhnul Saining Xie spolu s Zhuowen Tu a popsali jej v [8]. HED postupuje způsobem image-to-image (proto „Hollistically“) a s využitím dědičnosti vytváří postupně vylepšované obrázky hranových map (proto „Nested“), jako výstup.

V práci [8] je architektura sítě založená na síti VGGNet[9], kterou si autoři upraví následovně: „Our final HED network architecture has 5 stages, with strides

1, 2, 4, 8 and 16, respectively, and with different receptive field sizes, all nested in the VGGNet.“[8]

Od toho způsobu detekce očekáváme správně detekované i méně výrazné hrany (např. problematická nevýrazná vnitřní hrana, mezi dvěma stranami knihy), což se ve skutečnosti povedlo (viz obrázek 4b). V tuto chvíli máme detekci hran provedenou vcelku robustní metodou, která detekuje i slabé hrany. Tím bychom mohli vyřešit nedostatečnou detekci slabých hran, které komplikují správné odříznutí přechozí stránky novin/knihy. Naneštěstí stejně tak jako dokáže HED detekovat slabé hrany, zvýrazní nám hrany už tak dost výrazné. Z toho vyplývá opět problém s odstraněním nepotřebných hran popsaný v podkapitole 3.1.3.



(a) (b) (c)

Obrázek 4: Porovnání výsledků Canny algoritmu a HED

Poznámka 4 (Komentář k obrázku 4)

Na předchozí skupině obrázků 4 máme vedle sebe obrázek hran detekovaný Cannyho algoritmem 4a pro porovnání s hranami nalezenými pomocí HED 4b. Na prvním obrázku můžeme pozorovat nedokonale nalezenou vnitřní hranu mezi stránkami Kojetínských hlasů. Tuto hranu HED spolehlivě nalezne. Poslední z trojice 4c obsahuje zeleně vyznačený bounding box získaný na základě obrázku 4b, jehož součástí je kromě našeho ROI i předchozí stránka. Což je nežádoucí a stalo se tak kvůli potížím se správným zvolením prahů (podobně jako je popsáno v 3.1.3).

3.2.4 Nalezení bounding boxu pomocí neuronové sítě

Všechny dosud diskutované metody byly založeny na detekci hran následném nalezení bounding boxu a ořezu. U každé z nich se projevilo nějaké úskalí, které bylo příliš velkou limitací na to, aby se dala pominout. Teď se pokusíme první zmiňovaný krok – tedy detekci hran, přeskočit a zkusíme promyslet metodu, která rovnou nalezne bounding box aniž bychom nechali počítač hledat hrany.

Nabízí se nám použití neuronové sítě, kterou natrénujeme ke hledání bounding boxu na poskytnutých datech. Podle něj obrázek ořízneme. Jeví se to jako jedna z posledních, ještě nevyzkoušených, možností která by zaručeně měla fungovat. Stojíme před úkolem zvolit dostatečně početnou trénovací množinu, ve které budou zároveň ve správné míře reprezentovány obrázky, které by neuronová síť měla umět zpracovat.

Oblíbená techinika pro nalezení bounding boxu s pomocí neuronové sítě se nazývá „Bounding Box Regression“. Funguje na principu určování nebo predikce bounding boxu poblíž definovaných oblastí [10]. Tyto oblasti definujeme při trénování neuronové sítě, která se následně snaží na konkrétních obrázcích najít bounding box pomocí regrese. Regresní model slouží k určování spojitých hodnot. Často se vuyžívá pro predikci vývoje na akciovém trhu anebo třeba k určení rychlosti šíření nemoci v populaci [15].

3.2.4.1 Výhody a nevýhody

Výhodou této techniky, je relativně malá trénovací množina anotovaných obrázků. Po jejímž naučení dokáže neuronová síť vcelku spolehlivě detekovat bounding naší ROI. Což se znalostmi poskytnutých datasetů (viz 2) potřebujeme.

Jako hlavní nevýhodu, ale spíše vlastností, můžeme uvést možnost nesprávné detekce bounding boxu na obrázcích, které nebyly zastoupeny v trénovací množině. Jelikož data při digitalizaci v našem případě vznikají postupně, nemůžeme natrénovat neuronovou síť jednou. Možným řešením by bylo buď postupně přidávat podmnožinu skenovaných snímků z nově vzniklých datasetů a neuronovou síť přetrénovávat podle potřeby anebo s každým vzniklým datasetem vytvořit nový model, který konktrétně cílí na daný dataset (např. model pro Kojetínské hlasy, model pro noviny České slovo atp.) a při zpracování načíst správný model. To však není v této práci řešeno.

Poznámka 5 (Komentář k obrázku 5)

Obrázek obsahuje zeleně zvýrazněný bounding box nalezený pomocí neuronové sítě. Tento způsob dokáže najít bounding box ve všech poskytnutých datasetech i přes relativně málo početnou trénovací množinu.

Obrázek 5: Bounding box nalezený pomocí neuronové sítě

4 Další operace

4.1 Rotace

Z kapitoly 2 víme, že některé obrázky (případně celé datasety) mohou obsahovat rotovanou oblast našeho zájmu. Jedná se o to, že jakmile ořízneme obrázek podle bounding boxu, rotace obsažená v obrázku se může stát více viditelným rušivým elementem při prohlížení čtenářem. Z tohoto důvodu by bylo dobré výsledný obrázek ještě před uložením rotovat o počet stupňů, který je potřebný pro narovnání.

Proces rotace ROI vychází z geometrie, kdy si nejdříve potřebujeme zjistit úhel α o něhož je obrázek rotován a musíme jej zrotovat o jeho záporný ekvivalent v opačném směru okolo počátku. To se dá udělat pomocí affiní transformace využitím matice rotace[13]:

$$M_{rotace} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

anebo ekvivalentním zápisu souřadnic bodů:

$$\begin{aligned} x_2 &= x_1 * \cos(\alpha) - y_1 * \sin(\alpha) \\ y_2 &= x_1 * \sin(\alpha) + y_1 * \cos(\alpha) \end{aligned}$$

V současnosti se tato operace neprovádí na všech rotovaných obrázcích, ale jen na těch, které zaměstnanci digitalizační jednotky Knihovny vyhodnotí jako příliš křivé. Z tohoto důvodu se může do výsledku promítnout obrázek rotovaný o malý počet stupňů (těžko viditelný okem), který by byl při strojovém zpracování narovnán.

4.2 Opravy deformací

Opravami deformací je myšleno hlavně narovnávání textu u zakulacených stránek, které vzniká například vyfocením otevřené knihy s více stranami. Pokud takovou knihu otevřeme někde uprostřed můžeme pozorovat jev, kdy text blíže ke středu knihy nabírá určité zkosení směrem k vazbě knihy. Proces narovnání takového textu se označuje jako „dewarping“.

4.2.1 Princip fungování

Existuje více přístupů (například Leptonica[14]). Pro účely této práce však budeme čerpat z článku Matta Zuckera viz [11]. Ten navrhnul metodu, kterou rozdělil na 7 kroků:

[13] http://www.fit.vutbr.cz/~tisnovpa/publikace/diplomka/doc/node68.html
[14] http://www.leptonica.org/

1. Začne tak, že si vystřihne střední část stránky.

2. Vyhledá ve vystřihnutém obrázku obrysy (kontury) a vybírá ty, které vypadají jako text, jedná se o vícekrokový proces již s počátečním adaptivním prahem.

3. Provede morfologické operace dilataci (sloužící k rozšiřování tvarů ze vstupního obrázku viz [1], kapitola 9.2) pro připojení horizontálně sousedících pixelů masky a následně erozi (sloužící ke zmenšování tvarů ze vstupního obrázku viz [1], kapitola 9.2) k eliminaci výkyvů s vysokým počtem pixelů. Dále pokračuje analýzou a odstraněním komponent, které jsou příliš vysoké na to, aby byly textem.

4. Kombinace všech obrysů odpovídajících jednomu horizontálnímu rozpětí na stránce.

5. Vybere vzorky rozpětí. Protože parametrický model potřebuje diskrétní klíčové body, vygeneruje na každém rozpětí malý počet reprezentativních bodů.

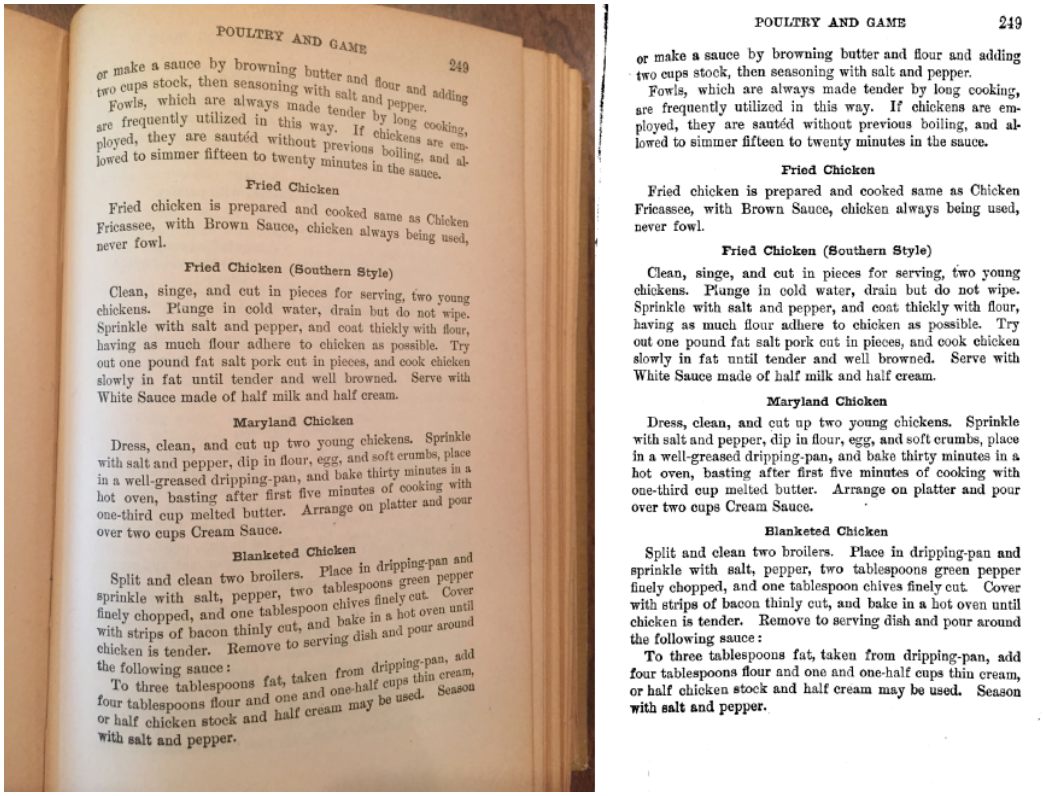
6. Dále odhaduje střední orientaci všech rozpětí. Výsledné hlavní komponenty jsou použity k analytickému stanovení počátečního odhadu souřadnic x a y.

7. V předposledním kroku Matt Zucker provádí optimalizaci pro minimalizaci reprojekční chyby.

8. Zbývá už jen přemapování a uložení výsledného obrázku na nějž aplikuje adaptivní prahování.

Poznámka 6 (Komentář k obrázku 6)

Jak výsledek této metody vypadá se můžeme podívat na následujících obrázcích (6a a 6b), které Matt Zucker uvádí jako výstup své implementace metody navrhnuté pro dewarping. Vlevo je vždy vstupní obrázek a vpravo leží obrázek dewarpovaný.



POULTRY AND GAME 249

or make a sauce by browning butter and flour and adding two cups stock, then seasoning with salt and pepper.

Fowls, which are always made tender by long cooking, are frequently utilized in this way. If chickens are employed, they are sautéd without previous boiling, and allowed to simmer fifteen to twenty minutes in the sauce.

Fried Chicken

Fried chicken is prepared and cooked same as Chicken Fricassee, with Brown Sauce, chicken always being used, never fowl.

Fried Chicken (Southern Style)

Clean, singe, and cut in pieces for serving, two young chickens. Plunge in cold water, drain but do not wipe. Sprinkle with salt and pepper, and coat thickly with flour, having as much flour adhere to chicken as possible. Try out one pound fat salt pork cut in pieces, and cook chicken slowly in fat until tender and well browned. Serve with White Sauce made of half milk and half cream.

Maryland Chicken

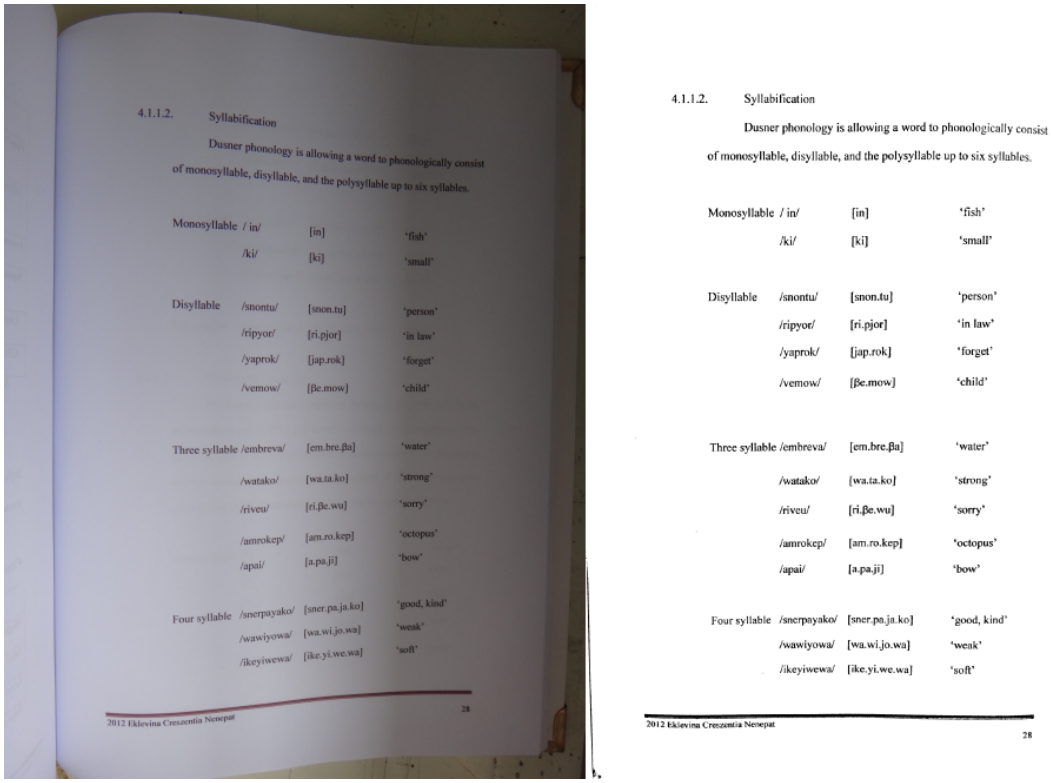
Dress, clean, and cut up two young chickens. Sprinkle with salt and pepper, dip in flour, egg, and soft crumbs, place in a well-greased dripping-pan, and bake thirty minutes in a hot oven, basting after first five minutes of cooking with one-third cup melted butter. Arrange on platter and pour over two cups Cream Sauce.

Blanketed Chicken

Split and clean two broilers. Place in dripping-pan and sprinkle with salt, pepper, two tablespoons green pepper finely chopped, and one tablespoon chives finely cut. Cover with strips of bacon thinly cut, and bake in a hot oven until chicken is tender. Remove to serving dish and pour around the following sauce:

To three tablespoons fat, taken from dripping-pan, add four tablespoons flour and one and one-half cups thin cream, or half chicken stock and half cream may be used. Season with salt and pepper.

(a)



4.1.1.2. Syllabification

Dusner phonology is allowing a word to phonologically consist of monosyllable, disyllable, and the polysyllable up to six syllables.

Monosyllable	/ in/	[in]	'fish'
	/ki/	[ki]	'small'
Disyllable	/snontu/	[snon.tu]	'person'
	/ripyor/	[ri.pjor]	'in law'
	/yaprok/	[jap.rok]	'forget'
	/vemow/	[βe.mow]	'child'
Three syllable	/embreva/	[em.bre.βa]	'water'
	/watako/	[wa.ta.ko]	'strong'
	/riveu/	[ri.βe.wu]	'sorry'
	/amrokep/	[am.ro.kep]	'octopus'
	/apai/	[a.pa.ji]	'bow'
Four syllable	/snerpayako/	[sner.pa.ja.ko]	'good, kind'
	/wawiyowa/	[wa.wi.jo.wa]	'weak'
	/ikeyiwewa/	[ike.yi.we.wa]	'soft'

2012 Eklevina Creszentia Nenepat 28

(b)

Obrázek 6: Ukázka výstupu implementace dewarpingu Matta Zuckera

4.3 Spojování snímků

Poslední zmíněnou operací, kterou ve Vědecké knihovně potřebují provádět je spojování skenovaných snímků. Na těchto snímcích jsou typicky zobrazeny velké mapy, které se nevlezou na skenovací plochu snímače najednou, proto jsou skenovány po částech. Tyto části musí zaměstnanci knihovny ručně pospojovat, aby ve výsledku vznikla digitalizovaná mapa.

4.3.1 Postup

Základním předpokladem, který musí být splněn, je skutečnost, že skenované mapy se musí alespoň částečně překrývat. Začneme tím, že si načteme obrázky, které chceme spojovat a převedeme je na šedotónové. Zjistíme klíčový bod (tzv. keypoint; viz obrázek 7) pomocí SIFT (tzn. Scale Invariant Feature Transform[15]). Jedná se o kruhovou oblast v obrázku s orientací, která je popsána následujícími parametry:

1. souřadnice středu keypointu,
2. poloměr oblasti,
3. orientace oblasti (tzn. úhel zachycený v radianech).

Každému klíčovému bodu je přiřazen descriptor. Tyto descriptory porovnáváme mezi dvěma obrázky a vybíráme M nejlepších shod. Autorka zdrojového článku citesood Naveksha Sood zvolila $M = 2$, tím získala dva klíčové body[16] a pokusila se je zarovnat pomocí transformace homografie[17]. Mezi dvěma obrazy je scénografická souvislost pokud platí, že:

- oba obrázky jsou rovina (např. list papíru),
- oba obrázky byly pořízeny otáčením kamery kolem její osy.

Pak si autorka textu vytvořila homografickou matici, což vyžaduje nejméně 4 shody mezi keypointy. K tomu se využívá robustní technika nazvaná RANSAC (tzn. Random Sample Consensus). Po provedení transformace můžeme obrazy spojit v jeden. Postup spojování obrázků je čerpán z [12].

Benefit této metody spočívá v tom, že můžeme spojovat skenované snímky horizontálně i vertikálně. Protože tato metoda je založena na porovnávání descriptorů klíčových bodů. Desciptor je definován jako 3D histogram gradientů obrázku charakterizující vzhled klíčového bodu. Tedy můžeme zvolit klíčové body na levém okraji prvního a pravém okraji druhého obrázku. Tím zjistíme, že tyto obrázky k sobě pravděpodobně patří horizontálně. Anebo zvolíme klíčové body

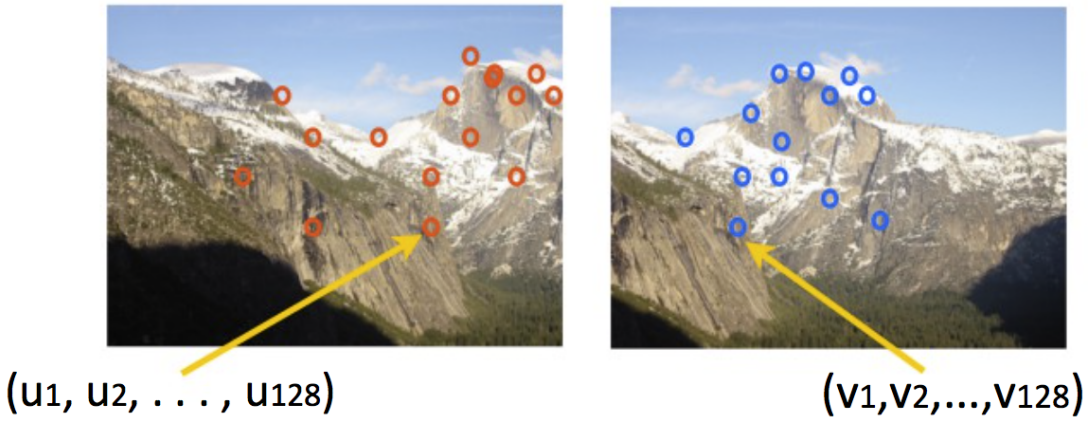
[15] https://www.vlfeat.org/api/sift.html

[16] každý z jiného obrázku

[17] transformace homografie je transformace mapující dvě projekce obrazu viz https://mattmaulion.medium.com/homography-transform-image-processing-eddbcb8e4ff7

na spodním okraji prvního a horním okraji druhého obrázku, z čehož vyplývá, že budeme obrázky spojovat vertikálně.

Jak už víme z prvního odstavce této kapitoly 4.3, budeme spojovat hlavně staré mapy nasnímané automatickým skenerem. Konstantní světelné podmínky v místnosti se skenerem nám dovolí ignorovat některé skutečnosti, které bychom museli řešit za jiných podmínek (např. spojování fotografií z exteriéru s proměnlivým počasím). Předpokládáme, že všechny spojované snímky jsou stejně zarovnané (tzn. neexistují zde rotace mezi jednotlivými snímky).



Obrázek 7: Ilustrační obrázek zobrazující klíčové body

Poznámka 7 (Komentář k obrázku 7)

Na obrázku[18] můžeme pozorovat množinu keypointů. Jeden z keypointů (na levém i pravém obrázku) je označen žlutou šipkou. Ta ukazuje, že dané keypointy jsou na obou obrázcích ekvivalentní a je možné podle nich obrázky spojit.

[18] zdroj obrázku: https://ai.stanford.edu/~syyeung/cvweb/tutorial2.html

5 Komprese výstupních souborů

Vstupní soubory, které vytváří automatický skener, jsou ve formátu TIFF jak již víme z kapitoly 2. Zaměstnanci digitalizační jednotky Vědecké knihovny v Olomouci oříznuté snímky potřebují komprimovat do formátu JPEG2000 a uložit je ve dvou kopiích. První kopie (zvaná „master“) je určena k archivaci a druhá kopie slouží jako uživatelská. Zároveň je potřeba pro interní potřeby zanechat neupravené TIFF soubory.

5.1 JPEG 2000

Na chvíli odbočíme k formátu, do kterého budeme soubory komprimovat. Nutnost této komprese vychází z potřeb Národní knihovny České republiky a z porovnání doporučení pro kódování souborů JPEG2000 v rámci zahraničních projektů zabývajících se digitalizací a dlouhodobým uchováním obrazových dat [13].

V případě JPEG2000 se jedná o ISO standard[19] pro různé typy obrazů, který je rozšířením standardu JPEG. Hlavní odlišností oproti JPEG je použití Diskrétní waveletové transformace, která umožňuje lepší popis ve frekvenční doméně než DCT (tzn. Diskrétní kosinová transformace) používaná u standardu JPEG. Nabízí také extrémně vysokou úroveň škálovatelnosti a přístupnosti. Obsah lze kódovat v jakékoli kvalitě včetně bezeztrátové komprese [18].

5.2 Kompresní parametry

Obě vytvářené kopie slouží k odlišným účelům. Master kopie slouží k archivaci a pro vnitřní potřebu. Druhá (uživatelská) kopie slouží pro zpřístupnění budoucím online čtenářům. Podle toho se liší jejich kompresní parametry, které jsou prezentovány následující tabulkou [13]:

Popis	Archivní kopie	Uživatelská kopie
Využití (Usage)	Knihy, periodika, mapy, manuskripty	Mapy, manuskripty, knihy, periodika
Grafický formát (Image still format)	*.jp2	*.jp2
Využitá část specifikace (Specification Part)	1. část JPEG2000	1. část JPEG2000
Migrační software (Migration software)	Kakadu	Kakadu
Druh komprese (Compression)	Bezeztrátová	Ztrátová
Transformace (Transformation)	5 - 3 reversible filter	9 - 7 ireversible filter

[19] https://www.iso.org/standard/33877.html

Popis	Archivní kopie	Uživatelská kopie
Kompresní poměr (Compression ratio)	—	1:8 až 1:10
Dlaždice (Tiling)	4096×4096	1024×1024
Průběh zobrazení (Progression order)	RPCL	RPCL
Počet dekompozičních úrovní (Decomposition level)	5 nebo 6	5 nebo 6
Počet vrstev kvality (Quality layers)	1	12 (logaritmicky)
Velikost regionů (Precinct size)	256×256 pro první dvě dekompoziční úrovně, 128×128 pro nejnižší úrovně	256×256 pro první dvě dekompoziční úrovně, 128×128 pro nejnižší úrovně
Zájmové oblasti (Regions of Interents)	Ne	Ne
Velikost bloků (Code block size)	64×64	64×64
Značka lokalizující dlaždice TLM (Tile Length Markers)	Ano („R“)	Ano („R“)
Přemostění (Bypass)	Ano	Ano
ICC profily (ICC profile)	Ano	Ano
Značka začátku hlavičky segmentu paketů SOP (Start Of Packet Header)	Ano (Cuse_sop=yes)	Volitelné
Značka konce hlavičky segmentu paketů EPH (Start Of Packet Header)	Ano (Cuse_sop=yes)	Volitelné
Vložená metadata (Embedded Metadata)	Ne	Ne

Tabulka 1: Seznam kompresních parametrů podle NDK

6 Implementační část

Tato kapitola se zaměřuje na praktickou aplikaci poznatků nabytých z předchozího textu. Představíme si, jak by měla aplikace fungovat, aby mohla být co nejefektivněji využívána. Na jakém poběží systému a v neposlední řadě se podíváme na to, jak je aplikace naprogramována. Závěrem této kapitoly nastíníme, jestli a jak by se dala aplikace vylepšit.

Dle požadavků zaměstnanců digitalizační jednotky VKOL není nutné, aby měla výsledná aplikace grafické rozhraní. Využívají ke své práci blíže nespecifikovanou verzi operačního systému Linux CentOS a jsou zvyklí používat terminál. Jednou z jejich představ je, že aplikace poběží na serveru se zmiňovaným operačním systémem CentOS, kde také bude dávkově zpracovávat naskenovaná data, vytvořená automatickým skenovacím robotem[20]. S těmito podmínkami souvisí výběr použitých technologií, pomocí kterých bude aplikace vytvořena.

6.1 Použité technologie

Jako důležitá součást na samotném začátku vývoje je volba správných technogií. Zvolíme-li nesprávně, vytvoříme si hromadu další práce do budoucna a ztížíme tak nejen celkový vývoj, ale i běh a fungování aplikace. Hlavním problémem je volba správného programovacího jazyka. S informacemi o požadavcích, které jsou na budoucí aplikaci kladeny, jsem se rozhodl pro Python.

6.1.1 Python

Jedná se o multiplatformní a vysokoúrovňový interpretovaný programovací jazyk, který navrhnul Guido van Rossum v roce 1991 jako open source [19]. Je důležité znímit, že Python se liší ve verzích. Verze 2.X není kompatibilní s verzí 3.X. Pro tuto aplikaci jsem zvolil verzi 3.6.7. Python je bezkonkureční svou jednoduchou syntaxí. Jelikož se jedná o interpretovaný jazyk, mohli bychom při vývoji narazit s jeho rychlostí – což by se dalo považovat za nevýhodu. Dalším z důvodů proč zvolit tento jazyk je jednozačně možnost rozšiřitelnosti různými balíčky (jinak řečeno knihovnami či moduly), které extrémně usnadňují práci. Tyto moduly lze stáhnout pomocí balíčkovacího systému Python package Index (tzv. PyPI). Možnost importovat moduly pro počítačové vidění OpenCV anebo Tensorflow pro práci s neuronovými sítěmi byla jednoznačně silným argumentem pro zvolení tohoto jazyka.

6.1.2 Používané balíčky

OpenCV [20] je open source modul pro manipulaci s obrazem, zaměřený na počítačové vidění[21] a k zpracování obrazu v reálném čase. Obsahuje spoustu užiteč-

[20] princip funkce robota je zachycen ve videu umístěném na oficiálním YouTube kanálu VKOL viz https://www.youtube.com/watch?v=CDDzpX2UABQ

[21] jak napovídá název OpenCV = Open Source Computer Vision

ných předem vytvořených funkcí (např. pro hledání kontur/obrysů, převod mezi barevnými prostory, morfologické operace jako jsou dilatace/eroze, a spoutu dalších).

NumPy [21] se označuje jako základní knihovna pro vědce a analytiky, pracující s Pythonem. Umožňuje totiž práci s vícerozměrnými homogenními poli, které definuje jako datový typ. Dále pak nabízí funkce pro práci s těmito poli (např. základní metody lineární algebry nebo diskrétní Fourierovu transformaci).

Pro práci s obrázky se často vuyžívá balíček **Pillow** nebo také **PIL** [22]. Ten umožňuje všechny možné operace s obrázky včetně uložení do různých formátů jako např. PNG, JPEG, GIF, TIFF, BMP, a jiné. Což se bude v případě naší aplikace hodit.

Tensorflow [23] je knihovna využívaná pro machine learning. Poskytuje mnoho možnosti jako například natrénovat neuronovou síť na vlastních datech a mnoho dalších. Při použití společně s **Keras API** [24], se jedná o základní knihovnu pro ty, kteří s neuronovými sítěmi chtějí začít stejně tak jako pro experty.

Balíček **H5PY** k ukládání dat v binární podobě. V našem případě nám pomůže serializovat na disk předtrénovaný model naší neuronové sítě.

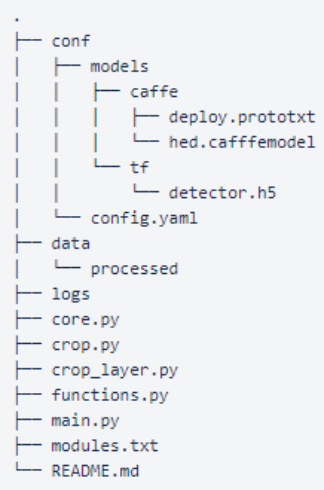
S pomocí balíčku **PyYAML** [26], který slouží pro práci se soubory YAML, budeme pracovat s konfiguračním souborem naší aplikace.

Vedled Tensorflow je **scikit-learn** [27] další knihovnou pro strojové učení v Pythonu.

Poslední balíčkem je **matplotlib** [28]. Ten se využívá pro vykreslování různým typů grafů v Pythonu.

6.2 Struktura adresáře aplikace

Adresář aplikace obsahuje v první řadě podadresář *conf*, obsahující konfigurační soubory a předtrénované modely neuronových sítí. Další podadresář se jmenuje *data* a defaultně je nastaven jako adresář pro výstupní soubory aplikace. Posledním podadresářem, jak již sám název *logs* napovídá, obsahuje soubory s informacemi o běhu aplikace. V samotném kořenovém adresáři aplikace jsou soubory se zdrojovými kódy s příponou **.py*, dále pak soubor *modules.txt* obsahující balíčky a jejich verze, které jsou nainstalovat, abychom zajistili bezproblémový chod aplikace. Nakonec kořenový adresář obsahuje soubor *README.md*, který obsahuje instrukce jak aplikaci dzprovoznit a používat. Adresářovou strukturu, můžeme vidět na následujícím obrázku 8.



Obrázek 8: Adresářová struktura aplikace

6.3 Popis a implementace aplikace

Jak můžeme pozorovat na obrázku 8, aplikace obsahuje poměrně malý počet souborů se zdrojovými kódy. Hlavním souborem je *main.py*, který voláme, pokud chceme aplikaci spustit. V tomto souboru je dále vytvářena instance třídy *Core* ze souboru *core.py*, která načítá konfigurační soubor a předzpracovává soubory pro třídu *Crop* ze souboru *crop.py* a následně provádí kompresi výstupních souborů, které třída *Crop* vrací. Soubor *functions.py* obsahuje obecně využitelné funkce volané z různých míst. Můžeme zde najít třídu *CropLayer*[22], která se používá pro potřeby detekce hran pomocí HED. Samotná třída *Crop* obsahuje mimo jiné, pět metod pro ořez, implementující poznatky z kapitol 3 a 4.1. Jedná se o metody:

1. *classical_edge_detection* pro ořez klasickými metodami,
2. *vertical_lines_detection* pro ořez pomocí detekce nejdelší vertikální hrany,
3. *average_image_mask* zajišťující detekci podle průměrného obrázku a následný ořez,
4. *neural_edge_detection* řeší detekci pomocí HED a následný ořez,
5. *neural_bounding_box* k nalezení bounding boxu a ořez.

[22]Vytvoření třídy *CropLayer* je popsáno přímo v dokumentaci OpenCV: https://docs.opencv.org/4.5.2/dc/db1/tutorial_dnn_custom_layers.html

6.3.1 Ořez

Klasické metody jsou technicky poměrně jednoduché a v principu fungují tak, jak je popsáno výše v kapitole 3, proto se pojďme podívat spíše na předposlední metodu. Jedná se o *neural_edge_detection*. Ta implementuje již zabudované funkce implementované v OpenCV. Pro neuronovou síť využíváme předtrénovaný model, který vytvořil Ashutosh Chandra a umístil jej na svůj GitHub [14]. Zásadní část metody je prezentována na následující ukázce kódu 1.

```
1   w, h = input_image.shape[:2]
2
3   blob = cv2.dnn.blobFromImage(input_image,
4                           scalefactor=1.0,
5                           size=(w, h),
6                           mean=(104, 117, 123),
7                           swapRB=False,
8                           crop=False)
9   self.net.setInput(blob)
10  out = self.net.forward()
11
12  out = helper.resize(out[0,0], sizes=(original_h, original_w))
13  out_gray = (255 * out).astype(np.uint8)
```

Zdrojový kód 1: Ukázka kódu detekce hran pomocí HED

Poznámka 8 (Komentář k ukázce zdrojového kódu 1)
Hlavní část metody pro detekci. Na řádku č. 4 vytváříme vstupní data pro neuronovou síť, která jí následně předložíme. Po zpracování sítí vezmeme její výstup a uložíme jako šedotónový obrázek, který vypadá jako obrázek 4b.

Metoda *neural_bounding_box* inspirovaná podobnou metodou z [15] pracuje s předtrénovaným modelem[23]. Ten byl vytvořen z 230 náhodně vybraných souborů z poskytnutých datasetů, tak aby každý dataset byl zastoupen. Architektura neuronové sítě využívá předem definovanou konvoluční neuronovou síť VGG16 [9], ke které připojíme čtyři plně propojené vrstvy (Fully-connected layers) vracející souřadnice (x a y souřadnice) levého horního a pravého spodního bodu hledaného bounding boxu viz ukázka kódu 2. Jako optimalizér je zde využit Adam z Keras API.

[23] umístěným v *conf/models/tf/detector.h5*

```
vgg = VGG16(weights='imagenet', include_top=False, input_tensor=
    Input(shape=(224, 224, 3)))
vgg.trainable = False
flatten = vgg.output
flatten = Flatten()(flatten)
bbox_head = Dense(128, activation='relu')(flatten)
bbox_head = Dense(64, activation='relu')(bbox_head)
bbox_head = Dense(32, activation='relu')(bbox_head)
bbox_head = Dense(4, activation='sigmoid')(bbox_head)
model = Model(inputs=vgg.input, outputs=bbox_head)
```

Zdrojový kód 2: Ukázka kódu neuronové sítě pro hledání bounding boxu

6.3.2 Rotace

Rotace jsou zde implementovány pomocí matice rotace, tak jako bylo popsáno v 4.1. V následující ukázce kódu 3 je ukázano jakým způsobem se provádí rotace pomocí afinní transformace.

Poznámka 9 (Komentář k ukázce zdrojového kódu 3)

Pomocí metody *minAreaRect* implementované v OpenCV získáme bounding box společně s úhlem, o který je bounding box pootočen. Pokud není bounding box pootočen, funkce vrací -90. Dále si na základě šířky, výšky a úhlu vytvoříme pomocí metody *getRotationMatrix2D* opět implementované v OpenCV vytvoříme matici rotace, kterou následně aplikujeme na obrázek a tím jej pootočíme.

```
bbox = cv2.minAreaRect(sorted_contours[0])
...
angle = bbox[2] if bbox[2] == -90 else 0
rows, columns = image.shape[0], image.shape[1]
matrix = cv2.getRotationMatrix2D((columns/2, rows/2), angle, 1)
rotated_img = cv2.warpAffine(image, matrix, (columns, rows))
```

Zdrojový kód 3: Ukázka kódu afinní transformace - rotace

6.3.3 Dewarping a spojování skenovaných snímků

V mojí implemetaci této práce se nevyskytují dvě functionality. Jedná se o dewarping popsaný v 4.2 a spojování obrázků z podkapitoly 4.3. Tyto dvě funkctionality jsem nebyl schopen implementovat, protože datasety poskytnuté digitalizační jednotkou VKOL neobsahovaly data, na kterých by bylo možné požadované funkce vyvíjet a vyzkoušet.

Z podstaty fungování automatického skenovacího robota (ve fázi kdy skenovanou knihu přitiskne ze spodní strany skla) prakticky k warpingu stránek nemůže docházet. Pokud by byla funkce dewarpingu vytvořena tak by bylo zbytečné provádět její aplikaci automaticky na každý snímek. Jsou zde dva navrhované způsoby. Prvním z nich je vytvořit mechanismus detekce deformací a pak je odstranit výše popsanou metodou (viz podkapitola 4.2. Druhým, méně vhodným, způsobem by bylo nechat tuto akci provádět zaměstnance nadále ručně.

6.3.4 Komprese

Nutnost komprese do formátu JPEG2000 je v aplikaci volitelná a dá se nastavit v konfiguračním souboru. Nalezneme ji pod klíčem *use_ndk_standard*. V konfiguračním souboru můžeme také upravit kompresní parametry, pokud by se změnil standard Národní digitální knihovny (neboli NDK) [13]. Samotná komprese je prováděna pomocí modulu PIL [16].

6.4 Návrhy rozšíření

V této kapitole popíšu možná rozšíření, která jsem nestihl nebo nemohl do aplikace zapracovat. Jedná se zejména o funkci dewarpingu a spojování obrázků, k nimž jsem neměl potřebná data pro jejich vývoj a testování.

Dále by se dala vylepšit komprimace výsledných oříznutých obrázků. Modul PIL sice umožňuje uložení do JPEG2000, ale nenabízí všechny možnosti kompresních parametrů tak, jak jsou uvedeny v tabulce 1 a na webu NDK [13].

Bylo by dobré přidat do aplikace možnost vytvářet si vlastní model se kterým pracuje metoda *neural_bounding_box*. Jednalo by se o řešení problému, kdy bude aplikace spuštěna s úplně neznámým datasetem (např. landscape orientace dokumentů[24]).

[24]Nyní je model předtrénován pouze na poskytnutých datech, které jsou všechny v portait orientaci

7 Uživatelská část

Nyní popíšu způsob jakým aplikaci zprovoznit, jak se používá/spouští a jak si správně může uživatel nastavit některé parametry v konfiguračním souboru aplikace.

7.1 Návod ke zprovoznění

Ke správnému zprovoznění a následnému používání stačí dodržet pár jednoduchých kroků:

1. Zkopírovat kořenový adresář obsahující všechny soubory na počítač.
2. Nainstalovat Python 3.6.7 a všechny moduly s verzí uvedenou v souboru *modules.txt* umístěném v kořenovém adresáři aplikace. Tyto moduly můžeme stáhnout i hromadně pomocí příkazu `pip install -r modules.txt`
3. Pokud neexistuje, tak v kořenovém adresáři vytvořit podadresář s názvem *logs*.
4. Pokud neexistuje, tak v kořenovém adresáři vytvořit podadresář s názvem *data* a v něm další podadresář *processed*.
5. Zkontrolovat, jestli tyto nově vytvořené adresáře, mají udělena práva pro zápis. Pokud ne, tak je potřeba jim práva pro zápis udělit.

7.2 Konfigurační soubor

Je umístěn v *conf/config.yaml*. Uložen ve formátu YAML [17]. Při jakékoliv editaci je potřeba zachovávat konvence tohoto formátu. Obecně je v konfiguračním souboru hlavně možnost nastavení výchozích adresářů, kompresních parametrů, akceptovaných přípon a každá z těchto možností je dobře okomentovaná (v angličtině). Proto v následující tabulce 2 znázorníme co všechno je možné v konfiguračním souboru nastavit i v českém jazyce.

Klíč	Hodnota	Popis
dev_mode	0 - zakáže, 1 - povolí	Povolení použití méně testovaných funkcí
use_time	0 - zakáže, 1 - povolí	Povolení použití orientačních časů při výpisu v konzoli
allow_destination_direct	0 - zakáže, 1 - povolí	Povolení použití specifického adresáře pro výstupní soubory
use_ndk_standard	0 - zakáže, 1 - povolí	Povolení komprimace výstupních souborů do JPEG2000

Klíč	Hodnota	Popis
destination_dir_name	string obsahující správně formátovanou cestu k adresáři	Cesta k adresáři, kam se budou ukládat výstupní soubory
dest_dir_archival	string obsahující název adresáře	Název adresáře do kterého budou ukládány master kopie výstupu k archivaci
dest_dir_production	string obsahující název adresáře	Název adresáře, do kterého budou ukládany uživatelské kopie výstupu
nn_base_dir	string obsahující správně formátovanou cestu k adresáři	Cesta k adresáři s předtrénovanými modely
store_extension	obrazová přípona i s tečkou	Přípona, se kterou budou ukládany zpracované soubory (možno vybrat jen PNG, BMP, JPG, TIF)
allowed_extensions	výčet obrazových přípon i s tečkou	Přípony, se kterými budou soubory načteny ke zpracování (možno vybrat jen PNG, BMP, JPG, TIF)
save_mask	0 - zakáže, 1 - povolí	Možnost ukládání masky, vytvořené u metody využívající průměrný obrázek
mask_dir_name	string obsahující název adresáře	Název adresáře, kam se uloží maska

Tabulka 2: Přehled nastavitelných vlastností v konfiguračním souboru

Dále konfigurační soubor obsahuje kompresní parametry umístěné na webu Národní digitální knihovny [13]. Jména v komentáři u každého parametru korespondují se jmény v tabulce na tomto webu, proto nejsou zahrnuty v předchozí tabulce 2. Nejdříve jsou uvedeny parametry pro kompresi archivní kopie („master“). Následně až pak jsou uvedeny parametry pro kompresi uživatelské kopie.

7.3 Příklady spuštění

V této části si na příkladech demonstrujeme, jak můžeme aplikaci spustit. Aplikace obsahuje nápovědu, kterou je možné kdykoliv zobrazit pomocí příkazu:

```
python3 main.py -h.
```

Příklad spuštění s jedním povinným parametrem, kterým je cesta se soubory, které chceme zpracovat:

```
python3 main.py data/test
```

Příklad spuštění se dvěma parametry, kde prvním je cesta se soubory, které chceme zpracovat a druhým parametrem je cesta, kam chceme uložit výsledné soubory:

```
python3 main.py data/test -d data/processed/test
```

Příklad spuštění se dvěma parametry. Prvním je cesta ke vstupním souborům a druhým je metoda, kterou chceme ke zpracování použít:

```
python3 main.py data/test -m 1.
```

Přepínač *-m* zastupuje metodu, která bude použita pro ořez. Máme na výběr z následujících:

- 1 ... klasické metody,
- 2 ... metoda s detekcí vertikálních čar,
- 3 ... metoda využívající průměrného obrázku,
- 4 ... metoda využívající HED,
- 5 ... metoda založená na hledání bounding boxu neuronovou sítí.

Příklad spuštění se třemi parametry, které jsou kombinací předchozích způsobů volání aplikace:

```
python3 main.py data/test -d data/processed/test -m 5.
```

Závěr

Tato práce prozkoumává oblast automatického zpracování obrazu. Zaměřujeme se hlavně na operace ořezu, rotace, oprav deformací vzniklých skenováním, spojování obrázků a kompresí výstupních souborů jak v teoretické, tak praktické rovině. Součásti textu je návrh na rozšíření části aplikace, která nebyla možná vypracovat na základě chybějících dat.

Výstupem práce je prakticky nasaditelná konzolová aplikace, která bude sloužit jako nástroj pro urychlení a usnadnění práce digitalizační jednotky Vědecké knihovny v Olomouci.

Conclusions

This thesis examines the area of automatic image processing. It focuses mainly to cropping, rotation, correction of deformations created by scanning, stitching images and compression of output files in theoretical and practical level. Part of the text is a proposal to expand the part of application, which could not be programmed due to missing data.

The output of the thesis is a practically deployable console application, which will serve as a tool for speeding up and facilitating the work of the digitization unit of the Research Library in Olomouc.

A Obsah přiloženého CD/DVD

`doc/`
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu.

`src/`
: Zdrojové kódy aplikace včetně konfiguračních souborů a předtrénovanými modely neuronových sítí.

`data/`
: Ukázková a testovací data použitá v práci a pro potřeby testování práce při tvorbě posudků a obhajoby práce.

`readme.txt`
: Instrukce pro instalaci a spuštění aplikace, včetně všech požadavků pro jeho bezproblémový provoz.

Literatura

[1] GONZALEZ, Rafael C.; WOODS, Richard E. Digital image processing. 3rd ed. Upper Saddle River: Pearson, c2008. ISBN 0-13-168728-x.

[2] PIHAN, Roman. Formáty pro ukládání fotografií - 7.díl: TIFF [online]. 2007-12-12. Dostupné z: https://www.digimanie.cz/formaty-pro-ukladani-fotografii-7dil-tiff/2023

[3] VONDRÁK, Ivo. Neuronové sítě [online]. Dostupné z: http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf

[4] DURČÁK, Pavel. Neuronové sítě a princip jejich fungování [online]. 2017-09-08. Dostupné z: https://www.napocitaci.cz/33/neuronove-site-a-princip-jejich-fungovani-uniqueidgOkE4NvrWuNY54vrLeM670eFNQh552VdDDulZX7UDBY/

[5] ZACHA, Jiří. Konvoluční neuronové sítě pro klasifikaci objektů z LiDARových dat. 2019-05-01. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/82351/F3-BP-2019-Zacha-Jiri-Konvolucni_neuronove_site_pro_klasifikaci_objektu_z_LiDARovych_dat.pdf

[6] MITRENGA, Michal. KONVOLUČNÍ NEURONOVÁ SÍŤ PRO SEGMETACI OBRAZU. 2018. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=177588

[7] GRÁCA, Martin. Neuronové sítě pro doporučování knih. 2018. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=181898

[8] XIE, Saining; TU, Zhuowen. Hollistically-Nested Edge Detection. 2015. Dostupné z: https://pages.ucsd.edu/~ztu/publication/iccv15_hed.pdf7

[9] MA Edward. What is the VGG neural network: Introduction to VGGNet [online]. 2019-12-09. Dostupné z: https://becominghuman.ai/what-is-the-vgg-neural-network-a590caa72643

[10] LEE, Seungkwan; KWAK Suha; CHO Minsu. Universal Bounding Box Regression and Its Applications. 2019-04-15. Dostupné z: https://arxiv.org/pdf/1904.06805.pdf

[11] ZUCKER, Matt. Page dewarping. GitHub [online]. 2016-08-15. Dostupné z: https://mzucker.github.io/2016/08/15/page-dewarping.html

[12] SOOD, Naveksha. Image Stitching to create a Panorama [online]. 2019-07-31. Dostupné z: https://medium.com/@navekshasood/image-stitching-to-create-a-panorama-5e030ecc8f7

[13] Národní digitální knihovna. Standardy pro obrazová data [online] 2017-03-20. Dostupné z: https://old.ndk.cz/standardy-digitalizace/standardy-pro-obrazova-data

[14] CHANDRA, Ashutosh. Code for edge detection using pretrained hed model(caffe) using OpenCV. GitHub [online]. Dostupné z: https://github.com/ashukid/hed-edge-detector

[15] ROSEBROCK Adrian. Object detection: Bounding box regression with Keras, TensorFlow, and Deep Learning [online]. 2020-10-05. Dostupné z: https://www.pyimagesearch.com/2020/10/05/object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning/

[16] PIL - Image file formats [online]. Dostupné z: https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#jpeg-2000

[17] MALÝ, Martin. YAML: Serializační formát pro ukládání dat [online]. 2009-12-04. Dostupné z: https://zdrojak.cz/clanky/yaml-serializacni-format-pro-ukladani-dat/

[18] Overview of JPEG 2000 [online]. Dostupné z: https://jpeg.org/jpeg2000/

[19] Python [online]. Dostupné z: https://www.python.org/

[20] OpenCV [online]. Dostupné z: https://opencv.org/

[21] NumPy [online]. Dostupné z: https://numpy.org/

[22] PIL [online]. Dostupné z: https://pillow.readthedocs.io/en/stable/

[23] Tensorflow [online]. Dostupné z: https://www.tensorflow.org/?hl=en

[24] Keras API [online]. Dostupné z: https://keras.io/

[25] H5PY [online]. Dostupné z: https://www.h5py.org/

[26] PyYAML [online]. Dostupné z: https://pyyaml.org/

[27] scikit-learn [online]. Dostupné z: https://scikit-learn.org/stable/

[28] Matplotlib [online]. Dostupné z: https://matplotlib.org/