Gymnázium
České Budějovice
Jírovcova 8

MATURITNÍ PRÁCE

Umělá inteligence a OCR

Jakub Ambroz

vedoucí práce: Dr. rer. nat. Michal Kočer

V Českých Budějovicích 19. února 2020 **2019/2020**

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně s vyznačením všech použitých pramenů.

V Českých Budějovicích dne ..... podpis .....

Jakub Ambroz

Abstrakt

Umělá inteligence je pojem zahrnující široké spektrum přístupů a algoritmů a některé z nich jsou zde popsány. V dnešní době má AI spoustu využití. Jedno ze starších je rozpoznávání znaků v textu, protože digitalizace tištěných dat umožní jejich strojové zpracování. Toto optické rozpoznávání znaků (OCR) už má mnoho existujících řešení a jedno z nich je Tesseract. V praktické části je cílem zjistit, zda-li a jak je možné jeho výkon vylepšit úpravou vstupních obrázků pomocí OpenCV.

Klíčová slova

OCR, Optické rozpoznávání znaků, AI, Umělá inteligence, OpenCV, Tesseract, B-MOD, Brno Mobile OCR Dataset, předzpracování, preprocessing

Poděkování

Děkuji komunitě LaTeXu za pomoc s formální stránkou práce, zejména Jonáši Havelkovi a *tex.stackexchange*. Dále děkuji Pythonovské komunitě *stackoverflow* za archiv zodpovězených otázek. *B-MOD* za správně označený dataset. A v neposlední řadě tvůrcům nástrojů *Tesseract* a *OpenCV* a autorům přidružených dokumentací a návodů.
A především děkuji svému vedoucímu práce za trpělivost při kontrole maturitních prací, za pomoc s výběrem tématu a všeobecně dobrý a podporující přístup.

Obsah

1 Úvod

Umělá inteligence se vždycky vyznačovala tím, že vnímání toho, co je a není AI se měnilo podle toho, čeho byla aktuálně schopná. Jakmile bylo dosáhnuto nějakého obtížného cíle, tak se usoudilo, že k jeho dosažení není potřeba inteligence, že je to jen statistika nebo předprogramovaný algoritmus, takže se už nejedná o doménu AI.

V této práci jsou nastíněny techniky rozpoznávání znaků. Některé z nich už bychom dnes mezi AI neřadily. Naopak např. neuronové sítě se dnes těší velkému zájmu, ale asi ne z důvodu využití v OCR.

V praktické části využiji již předtrénovanou NS k rozpoznávání znaků. Bude to Tesseract 4.0, přesněji pytesseract, který nám umožní využívat Tesseract z pohodlí Pytohnu. Pokusím se vytvořit program, který zjistí, jak zlepšit úspěšnost rozeznávání znaků, pomocí úpravy obrázků. Na práci s obrázky použiji knihovnu OpenCV.

Část I

Teoretická část

2 OCR

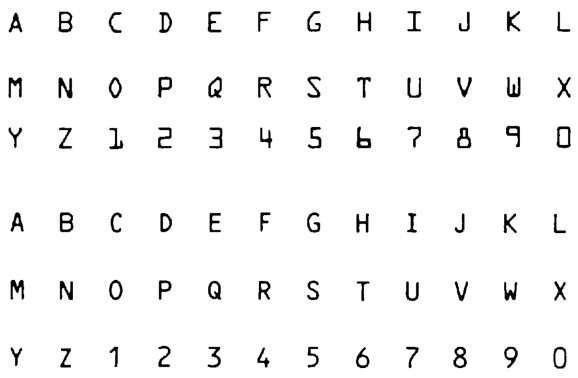
OCR je zkratka angl. *Optical Character Recognition* (v češtině optické rozpoznávání znaků [1]) je proces rozpoznávání a klasifikace vzorů v digitálním obrázku.[2, s. 9]
Oblasti rozpoznávání znaků:

- On-line - rozpoznává znaky, hned jak jsou nakresleny
- Off-line - rozpoznává až potom, co je tisk (nebo psaní) dokončený
  - Samostatné znaky
    - Tištěné
    - Ručně psané
  - Ručně psané psací písmo
    - Rozpoznání
    - Ověření

[2, str. 12], [3, str. 7]

Historie

Vývoj OCR systému již v 50. letech 20. století a zpočátku zvládali jen specializované symboly pro čtení počítačem. Později byly stroje pouze omezené v počtu fontů, protože používali metodu shody šablon (angl. *template matching*), která porovnávala obrázek znaku s knihovnou s šablonami každého znaku z každého fontu. Bylo vyvinuto velké úsilí na standardizaci fontů, viz obrázek 2.1. Vývoj se nakonec ubral k systémům schopným rozeznat více fontů. S klesající cenou a rostoucím výkonem hardwaru se stali komerčně dostupné a v 90. letech se techniky OCR spojili s výzkumem v oblasti AI. Dnešní systémy dosahují velmi uspokojivých výsledků. Jako jedno z posledních míst s možností výrazného zlepšení zbývá ručně psaný text. [3, s. 8-10], [2, s. 15]



Obrázek 2.1: Standardizované fonty americký OCR-A (nahoře) a evropský OCR-B (dole), zdroj: [3, str. 9]

2.1 Techniky OCR

2.1.1 Skenování

Proces *optického skenování* zachycuje digitální obraz původního dokumentu. Zařízení nazývané OCR skener obsahuje senzorické čidlo, které převede intenzitu světla do úrovní šedi (angl. *grayscale*). Protože jsou obvykle tištěné dokumenty černobílé, převádí se více-úrovňový obrázek do binárního, který má místo jen více úrovní šedi jen 2 hodnoty (proto se nazývá binární nebo dvou-úrovňový) a to černou a bílou. Tento proces, který se nazývá *thresholding* (z angl. slovíčka pro práh, threshold), protože jednotlivé pixely označí za černé (resp. bílé) překročí-li úroveň šedi určitý práh, se často vykonává již v samotném skeneru, protože to šetří místo v paměti a výpočetní náročnost. [3, str. 12]

Thresholding je důležitý proces, protože na kvalitě binárního obrázku jsou závislé rozpoznávací schopnosti systému. Jednotný práh v rámci celého obrázku může být dostatečný, pokud má dokument dostatečně kontrastní a jednotné pozadí, ale mnoho dokumentů v praxi tyto podmínky nesplňuje. V těchto případech je třeba sofistikovanějších metod pro kvalitní výsledky. Ty nejlepší techniky přizpůsobují práh podle lokálních vlastností (jako kontrast a jas) dokumentu, ale proto také často vyžadují víceúrovňové skenování dokumentu. [2, str. 16-17]

2.1.2 Lokace a segmentace

Proces segmentace určuje složky obrázku. Zajišťuje rozlišení a lokalizaci textu a ostatních částí (např.: grafy, loga, obrázky). Segmentace izoluje samostatná slova nebo znaky. Většinou se slova rozdělují na izolované znaky, které jsou rozpoznávány jednotlivě. Obvykle se izolují spojené součásti, protože je to jednoduché na implementaci. Ale může dojít k problémům,

pokud se znaky dotýkají nebo jsou naopak rozdělené na více částí. [2, str. 17] Nejdůležitější úkoly mohou být rozděleny do 4 kategorií [3, s. 13]:

1. Rozpoznání dotýkajících se a rozdělených znaků:
   Několik znaků může být omylem spojeno v jeden z pohledu OCR, pokud je skenování provedeno na příliš nízkém prahu. Některé fonty (patkové) jsou k tomuto náchylnější. Při roztříštění znaku dojde k mylnému určování každé části jako samostatného znaku.

2. Oddělení šumu od textu:
   Zejména problém pro tečky a čárky.

3. Záměna grafických prvků a textu:
   Pokud je grafika zaměněna za text, dojde k odeslání netextu k rozpoznání, což přidá nesmyslné části do výstupu. Pokud je text zaměněn za grafiku, bude část výstupu chybět.

2.1.3 Předzpracování

Cílem *předzpracování* (angl. preprocessing) je provést takové změny, aby byla úspěšnost rozeznání vyšší. Obrázek získaný procesem skenování může obsahovat šum a jiné defekty, které kazí rozpoznávání. Nejběžnější je technika nazývaná *vyhlazování* (angl. *smoothing*), která zahrnuje *vyplňování* (angl. *filling*), které odstraňuje malé díry a mezery ve znacích, a *zeštíhlení* (angl. *thinning*), které zmenší tloušťku čáry. Na vyhlazování se obvykle používá okno, které se posouvá po binárním obrázku a na jeho obsah jsou uplatněna určitá pravidla. Další technikou je *normalizace*, která má za cíl sjednotit znaky ve velikosti, sklonu a rotaci. Do předzpracování se zahrnuje i komprese, která ale zachovává všechny potřebné informace pro OCR. [2, s. 17-18]

2.1.4 Segmentace

Předzpracování nám dá čistý obrázek, který je bez šumu a obsahuje dostačující množství informací. Segmentace odděluje různé čáry a přímo tak ovlivňuje úspěšnost rozpoznávání. *Vnitřní segmentace* (angl. internal segmentation) se zaměřuje na spojované písmo a jeho rozdělení do čar a křivek. Zatím se jedná o nevyřešený problém, přestože bylo objeveno mnoho dobrých metod. Existují přístupy explicitní, implicitní a smíšené, které jsou nejúspěšnější.[2, s. 22]

2.1.5 Reprezentace

Reprezentace je proces, který získá *rysy* z každé kategorie, které pomohou rozlišit tuto kategorii od jiných, ale zároveň tyto nejsou ovlivněny obvyklými rozdíly v rámci této kategorie. Cílem je získat, co nejvíce reprezentativní informace z prvotních dat.

- *Globální transformace* a *rozšíření série* (angl. *series expansion*):
  Signál obvykle obsahuje více informací, než je potřeba. Jedním ze způsobů je reprezentovat signál lineární kombinací sérií jednodušších a dobře definovaných funkcí.

- *Statistická reprezentace*:
  Reprezentuje znak statistickým rozdělením bodů. Není možné zrekonstruovat původní obraz, protože zmenšuje rozměry sady znaků a tím zvyšuje rychlost a snižuje složitost.

- *Geometrická a topologická reprezentace*:
  Různé vlastnosti znaku mohou být reprezentovány geometrickým nebo topologickým způsobem, které mají velkou odolnost proti zkreslení. Tento způsob popisu také umožňuje získat znalosti o struktuře objektu nebo jeho dílčích částí.

[2, s. 23-28]

2.1.6 Získávání rysů znaku

Získávání rysů (angl. *feature extraction*) je považována za jeden z nejnáročnějších problémů rozpoznávání vzorů. Cílem je získaní hlavních a nezbytných vlastností znaku. Ty se vybírají na základě jejich odolnosti proti šumu, zkreslení, rotaci a praktickému využití, tj. rychlost rozpoznání, náročnost implementace. Na získávání rysů je přímo navázaná jejich *kategorizace* (neboli klasifikace), která identifikuje znak. [2, str. 28-29]

2.1.7 Rozpoznávání a trénování

Využitím metod *rozpoznávání vzoru* (angl. *pattern recognition*), které přiřadí neznámý vzorek do předdefinované kategorie. Základní rozdělení je na čtyři přístupy, které nejsou nezávislé a může se stát, že technika z jednoho přístupy může být také považována za techniku jiného přístupu. První je *porovnání se šablonou* (angl. emphtemplate matching), které porovnává znak se sadou vzorových znaků (tj. šablon). Může taky využívat deformovatelných šablon. [2, s. 29-34]

Druhým je *technika statistická*, která využívá rozhodovací funkce a sadu optimálních kritérií, které maximalizují pravděpodobnost výskyt pozorovaného vzoru v určité kategorii. Využívá se více statistických přístupů např.: skrytý Markovův model, mlhavou logiku (angl. fuzzy reasoning viz 3.4) nebo shlukové analýzy (angl. cluster analysis). [2, s. 29-34]
Strukturální techniky rekurzivně popisuje složitější vzory jednoduššími. Pomocí těchto vzorů jsou popsány a kategorizovány znaky. Dělí se na metody, které využívají gramatické, které využívají lingvistická pravidla, a grafické. V neposlední řadě se využívají *ANN*, které jsou schopné adaptovat se změnám v datech. Přestože využívají jiných vnitřních přístupů jsou ANN porovnatelné se statistickými technikami. [2, s. 29-34]

2.1.8 Následné zpracování

Angl. *postprocessing* (nebo post-processing) by se dalo přeložit jako následné zpracování nebo *pozpracování*. První částí je *seskupování* (angl. *grouping*) jednotlivých znaků v textové řetězce na základě jejich pozice v dokumentu. Je nutné rozlišit, jestli je mezera mezi znaky jen mezerou mezi znaky nebo mezerou mezi slovy, aby bylo možné seskupit znaky do slov. Protože jsou mezery mezi slovy výrazně větší, tak to není příliš obtížné ani pro méně vhodné fonty. Skutečné problémy nastávají, pokud se jedná o ručně psaný text (nepravidelná velikost mezer) a nebo pokud je text zkreslený.[3, s. 20]
Po seskupení znaků ve slova můžeme v druhé části následného zpracování využít kontextu, kde se znak nachází, k nalezení a opravení některých chyb. Žádný, ani ten nejlepší, OCR systém nemá schopnost rozpoznat 100% znaků správně. První způsob využívá *pravděpodobnost výskytu sledu znaků společně*. Např. za tečkou předpokládáme velké písmeno. Navíc může být vypočítána pravděpodobnost výskytu 2 znaků za sebou, to se samozřejmě liší podle jazyka (např. v češtině můžeme předpokládat, že znak "ě"po písmenu "j"nebo libovolné měkké souhlásce, je pravděpodobně chyba).Dalším přístupem je využití *slovníků*, ve kterých se vyhledá slovo. Pokud takové slovo neexistuje, je nalezena chyba, která může být opravena upravením slova na nejvíce podobné. To, že je slovo je nalezeno existenci chyby nevylučuje, protože mohlo dojít k záměně na jiné existující slovo. Přesto je tato metoda nejefektivnější pro nalézání a opravu chyb, přestože je časově náročná. [3, str. 20-21]

2.2 Soft computing

Soft computing (z angl. do češtiny by se dalo přeložit jemné výpočty) se ukázalo jako možné cenově efektivní řešení v podobě OCR systémů. Do soft computing spadají lidským mozkem inspirované metodologie, které mají za cíl využít toleranci k nepřesnostem a k nejasnostem k dosažení robustnosti a nízkých nákladů. Základními zástupci jsou mlhavá logika (viz fuzzy logika 3.4), neuronové sítě (viz NS 3.3), pravděpodobnostní logika a genetické algoritmy (viz 3.5). Jednotlivé typy nejsou konkurenti, ale spíš se vzájemně doplňují a jejich kombinování je výhodné. Využívá se pro *NP-úplné problémy*, tj. problémy, u kterých se předpokládá, že neexistuje žádné polynomiální algoritmus k jejich řešení. [3, str. 43-45], [4, str. 48-49]

2.3 OCR dnes

V dnešní době je velké množství OCR systémů. Většina z nich je cílená jen na zdigitalizování tištěných dokumentů, takže pokud chceme OCR s širším využitím, máme možnost využít služeb (placených) od technických gigantů, jako je *Microsoft* (počítačové vidění pod Azure, kde nabízí cloudové výpočty), *Google* (Vison AI pod Google Cloud) a *Amazon* (služba Rekognition). Ti kromě vlastních speciálních algoritmů využívají svých obrovských výpočetních kapacit a zapůjčují je k analýze obrázků. Kromě textu jsou tyto systémy schopny rozeznávat objekty, obličeje nebo i nevhodnost obrázku. [5], [6], [7]

3 Umělá inteligence

Umělá inteligence (angl. artificial intelligence, zkr. AI) je obor, který se snaží uměle vybudovat inteligentní systémy. Existuje mnoho možných přístupů. Některé se zaměřují na myšlení a uvažování těchto systémů , jiné na jejich chování a akce. AI může být buď porovnáváno s lidským výkonem nebo oproti ideálnímu výkonu, nazývanému racionalita.

Turingův test spočívá v testování, zdali je člověk schopen AI rozlišit od člověka na základě textové komunikace. To vyžaduje, aby byl počítač schopen pracovat s jazykem, pamatovat si viděné a slyšené, schopnost uvažování (např. odpovídat na otázky) a učení. Tento přístup se řadí do kategorie *lidské konání* (angl. *Acting humanly*). Naproti tomu přístup *lidského myšlení* (angl. *Thinking humanly*) se snaží replikovat postupy a mechanismy lidského uvažování. To ale naráží na problém, že musíme dobře porozumět samotnému lidskému myšlení.

Racionální myšlení (angl. *Thinking rationally*) se snaží vytvořit dokonalý zdůvodňovací systém využívající logiku. Problém představuje zapsání některých dat správnou formální notací, obzvláště když jsou výroky méně než 100% jisté, což je v reálném světě (mimo matematiku) obvyklé. Dalším problémem je náročnost výpočtu.

Přístup *racionálního konání* (angl. *Acting rationally*) definuje *rozumné činitele* (neboli racionální agenty, angl. *rational agent*), které jednají tak, aby dosáhli nejlepšího možného výsledku, alespoň na základě dostupných informací. Tento přístup v sobě může zahrnovat racionální myšlení, protože k racionálnímu chování je v některých případech třeba problém logicky vyřešit. Na rozdíl od lidského chování je to racionální obecnější a snáze matematicky definovatelné. Proto můžeme specifikovat jako cíl, kterého má racionální agent dosáhnout, lidské chování. V mnoha případech ale není časově možné provést všechny výpočty. Toto je problém omezené racionality (angl. *limited rationality*). [8, s. 1-5, 29-30]

3.1 Hledání

Mnoho problémů je řešitelné pomocí vhodného způsobu hledání. Ke správnému provedení potřebujeme dobře nadefinovat počáteční stav, všechny možné proveditelné akce, model přechodu, který řeší výsledky provedených akcí, cílový stav a funkci, která otestujte zda-li ho bylo dosáhnuto. Prostředí je reprezentováno *polem stavů* (angl. *state space*). Cesta vedoucí z počátečního stavu do cílového je řešení. Jeho efektivitu počítá účelová funkce (více o ní 3.3.2 u backpropagace). Vyhledávací algoritmy považují stavy a akce za atomické, tj. dále nedělitelné, takže dále neřeší jejich vnitřní strukturu a nepracují s ní.[8, str.]

3.2 Učení

Učení je definováno např. jako proces získání znalosti, pochopení nebo schopnosti zkoumáním, ukázkou nebo zkušeností. Nebo jako modifikace chování na základě zkušeností či nových znalostí. Využívá se, protože v některých případech je nemožné nebo velmi složité vytvořit přesný algoritmus. Např. prostředí, ve kterém bude stroj operovat, není přesně popsané (nebo popsatelné) nebo se může v čase měnit, tak místo ručního přizpůsobování algoritmu je lepší takový, který se je sám schopný adaptovat na měnící se podmínky. Další využití je v *dolování dat* (angl. *data mining*), kdy se snažíme najít vztahy a korelace v obrovských množstvích dat. [9, s. 1-3]

Online a Offline

Online přizpůsobuje síť (nebo algoritmus) po každém příkladu. Offline učení bere sadu problémů naráz a změnu provádí pro všechny najednou. Offline učení bývá také nazývané angl. jako *batch training procedures* (v češtině zhruba dávkový nebo várkový trénovací postup), protože řeší celou dávku najednou. [10, s. 54]

3.2.1 Rozdělení podle zpětné vazby

Angl. *unsupervised learning* (v češtině *učení bez dohledu)* je metoda, při které dostane agent (např. NS) pouze vstupní data a sám bez jakékoliv zpětné vazby se snaží najít vzory v datech a roztřídit je do kategorií. [8, s. 694-695], [10, str. 52]

V angl. *reinforcement learning* (česky *zpětnovazební učení*) dostane agent informaci zda byl výsledek správný nebo špatný neboli dostane odměnu či trest. Tato metoda posiluje (nebo upevňuje, angl. reinforcement) správné výstupy a odrazuje od nesprávných. Proto je to efektivnější než učení bez dohledu. Agent sám ale musí přijít na to, jak změnit výstup [8, s. 695], [10, str. 53]

V *učení s učitelem* (angl. *supervised learning*) dostane kromě vstupních dat, také jejich správné řešení. Nevýhodou je, že vyžaduje, aby velké množství dat bylo předem správně označeno. Proto se v praxi využívá kombinace těchto přístupů, protože máme jen málo příkladů správně označených a velké množství neoznačených či dokonce špatně označených. [8, s. 695], [10, str. 53]

3.3 Umělé neuronové sítě

(Umělé) Neuronové sítě (zkr. NS, angl. (Artificial) Neural networks, zkr. NN (/ANN)) jsou typem umělé inteligence. Jako vzor při jejich vytváření sloužily neurony (a jejich spojení) u živočichů. Cílem je získání žádoucích vlastností biologických předloh, zejména: schopnost učení, tolerance kazů (tj. nepřesností nebo šumu ve vstupních datech) a schopnost zobecňovat a sdružovat data. [10, s. 3-5]

Spojení neuronů je jednosměrné. Dendrity přijímají vzruchy z různých neuronů. Při aktivaci neuronu se vyšle vzruch pomocí axonu do dalších neuronů. K aktivaci neuronu dojde v případě, že přijaté vzruchy překonají určitý aktivační *práh* (angl. *threshold*).[10, str. 21]

Tím získáme nejjednodušší model neuronu (3.1), kde n je počet vstupních spojení do neuronu a kde a_i označuje výstup neuronu i. Neuron sečte všechny vstupní hodnoty a porovná, zda-li jsou větší než *práh*. Pokud jsou, tak vyšle neuron výstupní signál s hodnotou 1. A pokud je práh větší, nedojde k aktivaci neuronu a ten tím pádem žádný signál nevyšle.

$$vystup = \begin{cases} 0 & \text{pokud } \sum_{i=1}^{n} a_i \leq prah \\ 1 & \text{pokud } \sum_{i=1}^{n} a_i > prah \end{cases} \tag{3.1}$$

Perceptron

Model vylepšíme přidáním vah (angl. weight) spojením. Tím získáme jednoduchý model neuronu, který se nazývá perceptron (3.2). Byl vyvinut již v padesátých a šedesátých letech

dvacátého století a v dnešní době se v praxi příliš nepoužívá (nahrazeny *sigmoidními perceptrony* 3.3)[11, k. 1.1]

Váhu spojení neuronů i a j označíme $w_{i,j}$. Proměnná, kterou jsme si pojmenovali *prah*, se obvykle zapisuje řeckým písmenem theta (θ). Proto aktivační práh neuronu j označíme θ_j. Dále víme, že neuronu j připadá vystup a_j, proto platí $vystup = a_j$. Tím získáváme formálnější zápis (3.3).

$$vystup = \begin{cases} 0 & \text{pokud } \sum_{i=1}^{n} w_{i,j}a_i \leq prah \\ 1 & \text{pokud } \sum_{i=1}^{n} w_{i,j}a_i > prah \end{cases} \tag{3.2}$$

$$a_j = \begin{cases} 0 & \text{pokud } \sum_{i=1}^{n} w_{i,j}a_i \leq \theta_j \\ 1 & \text{pokud } \sum_{i=1}^{n} w_{i,j}a_i > \theta_j \end{cases} \tag{3.3}$$

Můžeme se setkat i se zápisem (3.4), kde se místo prahu (θ) používá zaujetí (angl. bias) b, pro které platí $b = -prah$. A ještě využívá zkrácení zápisu $wa = \sum_{i=1}^{n} w_{i,j}a_i$ pro větší přehlednost.[11, k. 1.1]

$$vystup = \begin{cases} 0 & \text{pokud } wa + b \leq 0 \\ 1 & \text{pokud } wa + b > 0 \end{cases} \tag{3.4}$$

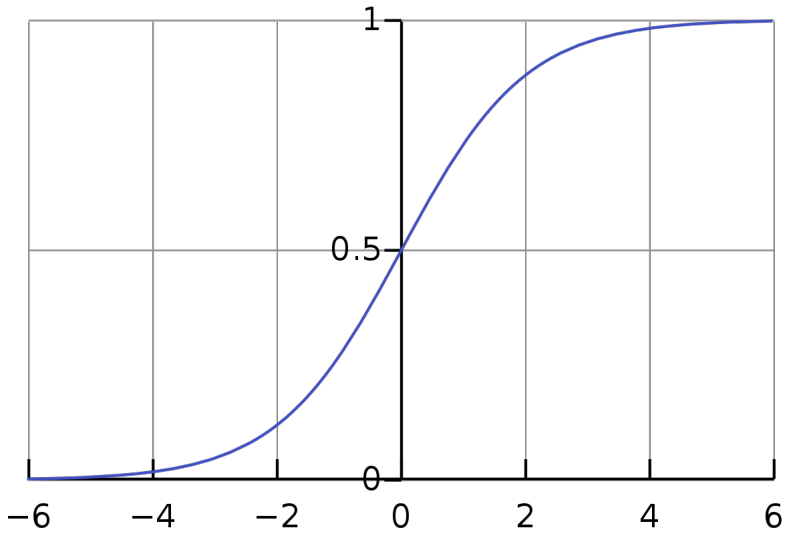
Sigmoidální perceptron

Normální perceptrony mají nevýhodu, protože když se lehce změní váhy a prahy (nebo zaujatosti) neuronů může to výrazně změnit výstup neuronové sítě, protože rozdíl mezi výstupy je skokový, tzn. velký a neplynulý. Proto, aby se mohla NN lépe učit je třeba, aby malá změna ve váhách vyvolala i malou změnu ve výstupu NN. Z tohoto důvodu se často používá *logistická funkce* (3.5). Její graf (viz obrázek 3.1 se též označuje jako sigmoida. [11, k. 1.2]

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \qquad L = 1, k = 1, x_0 = 0 \tag{3.5}$$

Aktivační funkce

K tomuto účelu se mohou využívat a v praxi se využívají i jiné funkce. Obecně se jim říká aktivační funkce nebo také přenosové funkce (3.6). Je-li to skoková funkce s pevnou hranicí



Obrázek 3.1: Logistická funkce s předpisem (3.5), zdroj: [12]

(prahem), nazývá se neuron perceptron. V případě, že se používá logistická funkce, setkáváme se často s názvem sigmodiální perceptron. Logistické funkce mají výhodu v tom, že jsou diferencovatelné. [8, str. 728-729], [10, str. 36]

$$a_j = g\left(\sum_{i=1}^{n} w_{i,j} a_i\right) \qquad \text{kde } g() \text{ je aktivnační funkce} \tag{3.6}$$

Kromě *Heavisideovy funkce* [13] (3.7), ta odpovídá normálnímu perceptronu, a logistické 3.5, také občas nazývané jako Fermiho funkce, se využívá třeba i hyperbolický tangens(podobný průběh jako logistická funkce). [10, str. 37-38]

$$H_p = \begin{cases} 0 & \text{pokud } x < 0 \\ p & \text{pokud } x = 0 \\ 1 & \text{pokud } x > 0 \end{cases} \tag{3.7}$$

3.3.1 Topologie NS

Existuje mnoho způsobů, jak neurony spojit ve funkční síť [14]. Nejzákladnější rozdělení je na *dopředné NS* (angl. *feed-forward networks*) a na *rekurentní sítě*, kde neurony svými výstupy ovlivňují sebe (buď přímo, nebo ovlivní neuron, který poté ovlivňuje jej). Z toho vyplývá, že odpověď této NS záleží na počátečním stavu. Jedná se o dynamický systém, který se nemusí ustálit a může oscilovat, či se chovat chaoticky. Dále mohou mít krátkodobou paměť (na

rozdíl od FFN). To je dělá zajímavějšími, ale také složitějšími. [8, str. 729]

Dopředné neuronové sítě (FFN)

Angl. nazývané *Feed forward networks*, do češtiny by se dalo přeložit jako dopředné neuronové sítě. Síť je uspořádána do *vrstev*. Jedna vstupní vrstva, jedna výstupní vrstva a mezi nimi nějaký počet skrytých vrstev. Výstupní spojení neuronů jdou vždy pouze do další vrstvy. Stejně tak vstupní spojení jsou vždy jen z vrstvy předchozí. Neurony v rámci jedné vrstvy nejsou propojené. Sítě, kde je každý neuron spojen s každým neuronem z další vrstvy nazýváme *kompletně propojené*. V některých případech zahrnují i tzv. zkratkové spojení, které přeskakují jednu nebo více vrstev, ale stále splňují podmínku, že směřují od vstupní vrstvy k vrstvě výstupní. [10, str. 39-40]

3.3.2 Učení NS

Učení nebo trénink NS spočívá v upravování sítě, tak aby její končená verze byla schopná dávat správné výstupy pro problémy nějaké kategorie s přesností a obecností, jaká se po ní vyžadována. Výstup NS může být změněn různými způsoby [10, s. 51]:

1. Přidáním nebo odebráním spojení
2. Změnou váhy existujícího spojení
3. Změnou aktivačního prahu neuronu
4. Změnou funkce uvnitř neuronu
5. Přidáním a odebráním neuronu (a jeho spojení)

Nejběžnějším způsobem je změna hodnot vah a prahu. Navíc v kompletně propojených sítích si můžeme představit odebrání spojení jen jako nastavení jeho váhy na nulu a přidání spojení jako opak. Změna vnitřní funkce je obtížná na implementaci, neintuitivní a nemáme motivující biologický vzor. Oproti tomu velký zájem se zaměřuje na přidávání a odebírání neuronů, protože to mění samotnou topologii NS tak, aby byla lépe přizpůsobená konkrétnímu problému. Používá se *evolučních postupů* (nebo algoritmů, viz 3.5), ve kterých podle zvolených mechanismů síť zmutuje několika způsoby a potom se vyberou ty nejvhodnější (tj.

ty která mají výstupy/výsledky nejbližší ideálním). Z toho vyplývá jejich nevýhoda, větší časová náročnost než u přímých metod učení jako *backpropagation*. [10, s. 52, 127]

Backpropagation

Používá se především ve vícevrstvých FFN. *Účelová funkce* (také cílová, kriteriální, nákladová angl. *objective* nebo také *cost*, loss případně error function) [10, s. 77] [8, s. 733] [15] [11, k. 1.5] je funkce, která hodnotí výstup NS (porovnává ho se správným výstupem). Jejího minima (nebo maxima podle definice, ale většinou se v terminologii používá minimalizace výstupní chyby) se snažíme dosáhnout. Jako vstup této funkce máme všechny váhy a prahy jako výstup. Spočítat globální minimum je velmi obtížné obzvláště pro takto komplikované funkce, ale je možné (a snadnější) spočítat její sklon. Budeme-li měnit váhy podle tohoto sklonu dosáhneme po několika opakováních lokálního minima, tento proces se nazývá angl. *gradient descent* (česky metoda sestupného gradientu(sklonu)).[16]
Existuje způsob, jak tento proces zapsat matematicky přes derivace[1]. Ale pro základní pochopení stačí následovně. Zadáme NS jeden problém a výstup výstupních neuronů porovnáme s ideálním výstupem. Čím větší rozdíl tím více je potřeba ho změnit, měníme tedy úměrně chybě. Výstup neuronu změníme změněním vah spojení z předchozí vrstvy, ty měníme úměrně k velikosti výstupu neuronu z předchozí vrstvy. Dalším krokem je změnit výstupy neuronů z předchozí vrstvy, ale tady mohou mít různé neurony výstupní vrstvy protichůdné zájmy. Ne každý zájem je stejně důležitý, takže se započítávají úměrně k váze spojení mezi těmito neurony a úměrně velikosti chyby výstupního neuronu. Výsledná požadovaná změnu výstupu neuronů v této vrstvě je tedy součtem všech zájmů z předchozí vrstvy. Výstup ale nemůžeme měnit přímo, protože je závislý na neuronech z vrstvy předchozí a vahách jejich spojení. To je ale v podstatě stejný problém, jaký jsme řešili ve vrstvě předchozí, takže stačí zopakovat tento proces v této vrstvě a pak v následující a tak dále dokud nedosáhneme vrstvy vstupních neuronů. Odtud pochází název backpropagation, což se dá přeložit jako šíření zpět. Tohle je pouze pro jeden problém ve skutečnosti musíme vyzkoušet sadu problémů a provést změnu nebo změny (viz dělení na online a offline učení 3.2 na straně 17) [18]

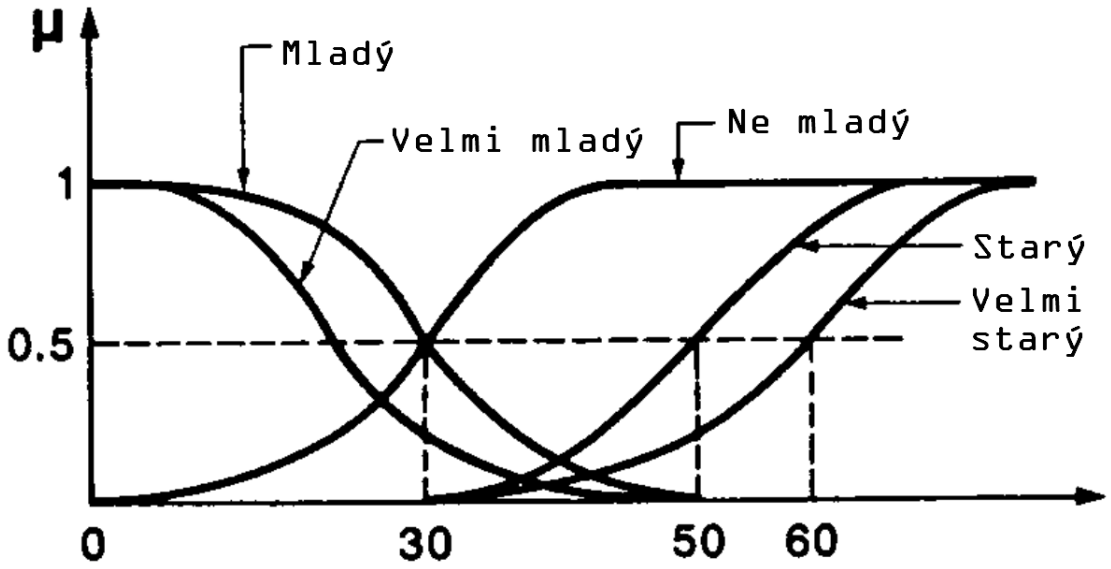
[1] 17, Anglicky s pěkným grafickým znázorněním to popisuje Backpropagation calculus | Deep learning, chapter 4 od 3Blue1Brown na stránce YouTube.

3.4 Fuzzy logika

Logika se zabývá formálními principy uvažování a odůvodňování. Fuzzy logika (z angličtiny překládáno také jako mlhavá logika [19]) je zaměřena na přibližné odůvodňování. Buduje modely, které (podobně jako lidské uvažování) jsou schopny činit racionální rozhodnutí v prostředí s nepřesnými a nejasnými informacemi. [20, s. 83]

Lingvistická proměnná

Je pojem ve fuzzy logice, který je klíčový při její aplikaci. Hodnoty jsou slova, slovní spojení nebo věty v přirozeném nebo umělém jazyce. Např. proměnná "věk"může nabývat hodnot: "mladý", "starý", "velmi mladý", "velmi starý", "ne mladý", "ne starý"atd. Každá hodnota lingvistické proměnné reprezentuje rozdělení možností, viz obrázek 3.2. [20, s. 90], [4, str. 50]



Obrázek 3.2: Příklad rozdělení pravděpodobností u lingvistické proměnné "věk", zdroj: [20, str. 90], upraveno

3.5 Genetické algoritmy

GA patří do širší skupiny *evolučních algoritmů*. Jsou to algoritmy inspirovány přírodní evolucí. GA jsou jednoduché na implementaci, ale jejich chování je často nepředvídatelné. Často se jim daří uspět v praktických problémech. Máme mnoho různých kandidátních řešení (nazývaných jedinci nebo *fenotypy*). Každé kandidátní řešení má své vlastnosti neboli chromosomy.

Na začátku se náhodně vygeneruje populace. V každé generaci se ohodnotí vhodnost jednotlivých řešení a ta nejlepší přežijí a do další generace postoupí jejich chromozomy, které se předtím mohou kombinovat a nebo náhodně mutovat. Tento postup je opakován. Ukončí se ručně nebo je-li dosaženo alespoň jedné z podmínek [2, str. 50-53] :

1. Dosažený předem určený počet generací.
2. Jsou vyčerpány zdroje (výpočetní nebo peněžní) pro pokračování.
3. Výsledky se dalšími generacemi nezlepšují a nebo jsme dosáhli minimální kritéria.

Pro implementaci vyžadují [2, str. 50-53]:

- Reprezentaci kandidátního řešení: typicky binární řetězec 1 a 0, ale jsou možné i jiné způsoby.
- Funkci vhodnosti (angl. *fitness function*, typ účelové funkce (více o ní 3.3.2 u backpropagace)): určující, jak úspěšné je řešení

3.6 Možnosti dnešní AI

3.6.1 DeepMind

AlphaGo byla první AI, která překonala světové šampióny stolní hry Go (populární v Asii). Hluboké neuronové sítě vybírali tahy z vyhledávacího stromu (angl. tree search). Tyto sítě byly trénované pomocí tahů lidských profesionálů a hraním sami se sebou. Následovná AI, AlphaGo Zero, byla trénovaná pouze pomocí reinforcement learning a hraním se sebou (angl. *self-play*) bez jakéhokoliv lidského vzoru. Toto vyústilo v lepší herní schopnosti a tato AI byla schopná porazit předchozí verzi 100-0. [21] Dalším krokem bylo zobecnění tohoto postupu vznikla AI *AlphaZero*. Algoritmus, který je schopný se takto naučit několik her (šachy, Go, shogi) a získat přesvědčivě dobré výsledky nad nejlepšími algoritmy či umělými inteligencemi.[22]

Další hrou, na kterou se v DeepMind zaměřili, byl StarCraft II. Je tomu z podobných důvodů jako OpenAI a Dota 2. Učení hrou AI se sebou samotnou muselo být trochu upraveno a připraveno, protože ve hře je velké množství strategií (množství různých typů jednotek a jejich kombinací) a proti-strategií (zjednodušeně kámen-nůžky-papír). Je totiž nutné zajistit, aby nedošlo k tomu, že síť naučí hrát jednu dominantní strategii a jakmile nějaká verze nalezne

proti-strategii, tak začne vyhrávat, až se z této stane dominantní strategie, která zase přestane dominovat, až k ní bude nalezena proti-strategie. Tím by AI mohla ztratit "paměť", jak hrát proti jiným strategiím, než té dominantní.[23]

3.6.2 OpenAI

OpenAI Five využívá deep reinforcement learning (viz 3.2.1 na straně 17) na překonání nejlepších hráčů hry *Dota 2*. Tato videohra je strategie v reálném čase pro více hráčů, kde 2 týmy se základnami v rozích čtvercové mapy soupeří, kdo zničí základnu nepřítele jako první. Byla zvolena, protože k jejímu zdolání je nutné překonat řadu obtížně překonatelných problémů pro reinforcement learning, které jsou podobné těm, kterým čelí AI v reálném světě.

- Plánování do daleké budoucnosti: hra trvá běžně 45 minut a běží 30 snímku za vteřinu. To je i s omezeným počtem akcí za vteřinu (OpenAI může provést akci každé 4 snímky) mnohonásobně víc než šachy nebo Go.
- Obrovské množství možných akcí a obrovské množství pozorovaných hodnot
- Každý tým (tedy i AI) má neúplné informace o přesném stavu hry. Vidí jen okolí spoluhráčů, jednotek a budov. Musí tedy předpokládat soupeřovo chování a strategii.

Rozšířením jíž existujících technik reinforcement učení, kde výrazně zvětšili měřítko. OpenAI Five se učilo hraním proti sobě v rychlosti zhruba 2 miliony snímků za 2 vteřiny. Nakonec zvládlo 17 (ze 117) hrdinů a porazilo v roce 2019 ve hře Dota 2 světové šampiony. Tím dokázalo, že i takto obtížné úkoly je možné zvládnout, máte-li dost velký výpočetní výkon, pomocí reiforcement učení. [24]

3.6.3 FAIR

FAIR (*Facebook Artificial Intelligence Research*) vytvořil AI jménem *Pluribus* (na základě AI Libratus, které porazilo profesionální hráče pokeru ve hře jeden proti jednomu.). Tato AI dokázala porážet profesionální hráče ve hře šesti hráčů. Zajímavé je tím, že se nejedná o typickou 1v1 situaci, kdy jeden hráč prohraje a druhý vyhraje. Navíc poker je hra s se velkým množstvím skrytých informací. Tohoto úspěchu bylo dosaženo opět využitím hry s sebou samým (angl. self-play) pomocí reinforcement learning. [25]

Část II

Praktická část

4 Postup

Cílem praktické části je vytvořit program schopný rozpoznávat znaky, využijeme k tomu *Tesseract* (viz 4.1.1). Ten ale většinu věcí dělá už automaticky vnitřně, takže hlavním cílem bude upravit obrázek pomocí knihovny *OpenCV* (viz 4.1.2, tak aby Tesseract byl schopný lépe rozeznávat znaky.

4.1 Použité nástroje

4.1.1 Tesseract

Tesseract je OCR nástroj psaný v programovacím jazyce C++ (původně v C). Začínal v polovině devadesátých let minulého století v laboratořích Hewlett-Packard. V roce 2005 společnost HP otevřela (tj. veřejně zpřístupnila) jeho zdrojový kód, a tak se Tesseract stal tzv. open source software. Následujícího roku se vývoje Tesseractu ujal Google. Nejnovější verze 4.0.0. podporuje UTF-8 a dokáže rovnou po instalaci rozeznat více než 100 jazyků. Navíc může být trénován a naučit se rozeznávat i další jazyky. Pracuje se s ním pomocí příkazové řádky, ale díky tomu, že se jedná o projekt s otevřeným zdrojovým kódem, existují grafická rozhraní vyvinutá třetími stranami. V mnoha případech je možné zlepšit úspěšnost rozeznání tímto OCR, pokud je zlepšena kvalita obrázku. [26]

Pytesseract

Pytessearact je nástroj, který umožňuje využívat Tesseract pomocí Pythonu. V podstatě jen volá Tesseract přes příkazovou řádku. [27]

4.1.2 OpenCV

OpenCV (angl. *Open Source Computer Vision Library*, v češtině knihovna počítačového vidění s otevřeným zdrojovým kódem) [28] obsahuje algoritmy jak pro počítačové vidění, tak

i pro strojové učení. OpenCV je psané v C++ a má rozhraní i v Javě, Pythonu a MATLABu. Uplatněno je např. ve sledování těžícího vybavení nebo v kontrole, že se nikdo netopí v bazénu. Je využíváno i známými společnostmi jako je Google, Microsoft a Intel. [29]

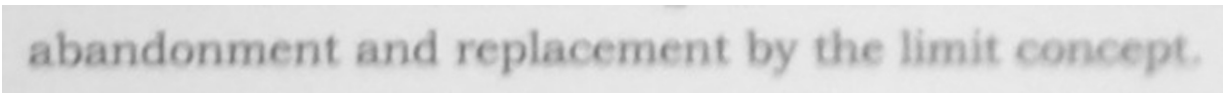
4.1.3 Datová sada

K otestování využijeme *Brno Mobile OCR Dataset* (zkr. *B-MOD*), protože obsahuje velké množství dat se správnými výsledky a s různou úrovní obtížnosti. Vytištěné vědecké články byly foceny různými zařízeními v nijak neomezených podmínkách, takže jsou různě rozmazané, nasvícené, atd. Jedna část je více než 500 000 řádek textu (každá jako samostatný obrázkový soubor) a mají přesné transkripce toho, co je na nich napsáno, takže můžeme snadno ověřit kvalitu našeho OCR systému. Tato část je dále rozdělena na 3 podmnožiny. Jsou to testovací, trénovací a ověřovací a všechny jsou ve třech úrovních obtížnosti: lehké, střední a těžké. [30], [31]
V této práci budu využívat nějakou menší (z důvodu délky výpočtu a také velikosti místa na disku) podmnožinu lehké obtížnosti, protože ta obsahuje dostatečný rozptyl náročnosti. Kromě jednoduchých obrázků se v ní vyskytují i více či méně rozmazané řádky.

4.2 Způsoby úpravy obrázků

Samotná dokumentace Tesseractu doporučuje provést úpravu obrázku a uvádí rovnou několik příkladů. [32]

4.2.1 Grayscale a binarizace



Obrázek 4.1: Příklad obrázku ve stupních šedi. Ze složky *lines-02(4)-11-*

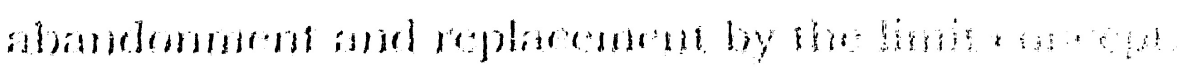
Jako první se nabízí převedení barevného obrázku do stupňů šedi (angl. grayscale, viz obrázek 4.1) nebo přímo jeho binarizace, která rozdělí všechny pixely obrázku buď na černé nebo bílé. Má 2 hlavní kategorie:

1. *Jednoduchý práh*: V rámci celého obrázku, je jednotný (globální) práh, který určí, jestli bude pixel bílý nebo černý.

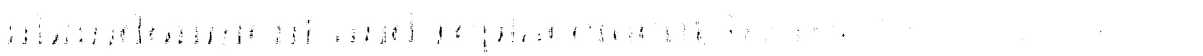
2. *Adaptivní práh*: Lokální práh se upravuje podle okolí, takže různé oblasti mají různý práh. To se hodí zejména pokud je nasvícení nerovnoměrné.

Obrázek 4.2: Ilustrační obrázek binarizace: lines-42(15,2)-

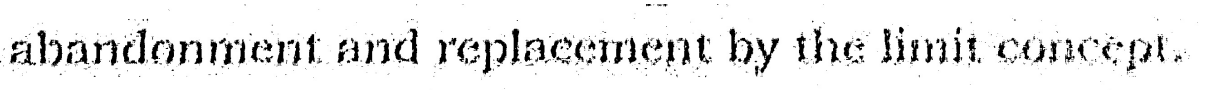
První přístup fungoval dostatečně dobře pro některé obrázky, ale jeden pevný práh pro celou sadu rozhodně nefungoval, protože některé málo nasvícené skončili celé černé a naopak ty přesvícené (nebo jen s nevýrazným textem) skončili celé bílé. Proto je nutné využít nějaký z adaptivních přístupů. To ale také není dostatečné, protože pozadí nemusí mít jednotnou barvu, ale může mít šum (některé pixely světlejší a jiné zase tmavší). Při binarizaci tak může dojít k vytvoření tmavých teček či skvrn na pozadí (viz 4.2) a ty by mohli být mylně považovány za znaky nebo jejich části Tesseractem.



Obrázek 4.3: Příklad obrázku adaptivního (průměrného) thresholdingu. Ze složky *lines-02(4)-41(9,4)-*



Obrázek 4.4: Příklad obrázku adaptivního (Gaussova) thresholdingu. Ze složky *lines-02(4)-42(9,4)-*

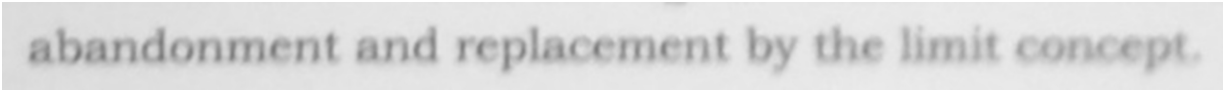


Obrázek 4.5: Příklad obrázku adaptivního (Gaussova) thresholdingu s jinými parametry. Ze složky *lines-02(4)-42(15,2)-*

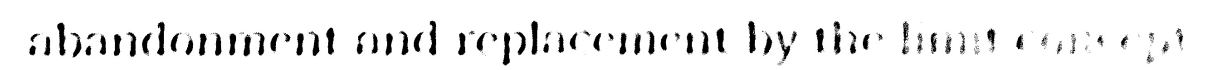
Obrázek 4.6: Příklad Otsuova thresholdingu. Ze složky *lines-02(4)-43)-*

4.2.2 Rozmazání

To nás vede k rozmazávání (angl. blurring), protože to by mohlo rozmazat pozadí a zmenšit tak rozdíly mezi jednotlivými pixely. Tím by se stalo pozadí více homogenním. Na druhou stranu by to mohlo rozmazat i znaky a zhoršit tak jejich rozpoznávání. Při rozmazávání se využívá například průměrování okolních hodnot nebo jejich střední hodnota.



Obrázek 4.7: Příklad rozmazaného obrázku ve stupních šedi. Ze složky *lines-02(4)-11-31(5,5)*



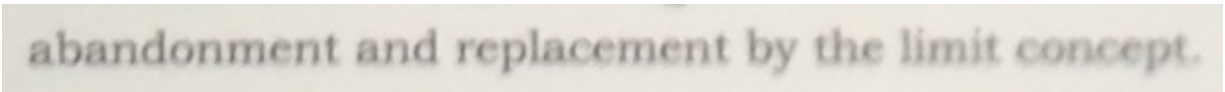
Obrázek 4.8: Rozmazání 4.7 a thresholdingu (porovnjete s 4.4). Ze složky *lines-02(4)-31(5,5)-42(9,4)-*

4.2.3 Odstranění šumu

V OpenCV se nachází i speciální funkce pro odstranění šumu z fotografií, která je ale vnitřně složitější a vyžaduje více času pro výpočet.

4.2.4 Změna velikosti

Protože binarizací ztrácíme informaci (místo spousty úrovní šedi, máme jen 2 úrovně) a protože obrázky některé obrázky jsou s malým rozlišením, může dojít ke ztrátě detailu, který může být nezbytně nutný k rozeznání znaku. Tomuto bychom mohli částečně zabránit zvětšením obrázku. Např. jeden šedý pixel se při binarizaci musí stát buď černým nebo bílým, ale pokud místo něj budeme mít 4 šedé pixely, tak se 2 mohou stát bílými a 2 černými a zachovat tím přesněji předchozí stav.



Obrázek 4.9: Příklad obrázku, který byl zvětšen. Ze složky *lines-02(4)-*

4.3 Automatizace

K nalezení správné kombinace takových funkcí a jejich parametrů, aby se zlepšila rozpoznávací schopnost Tesseractu, je třeba vytvořit si nějaký systém. Musí být schopen vytvořit a porovnat velké množství způsobů úpravy obrázku, pokud možno s minimálním zapojením člověka, aby mohl program běžet delší dobu sám, např. přes noc.

1. *mastercv.py*: Používá OpenCV a podle předloženého textového souboru provede úpravu obrázků a uloží je systematicky do složek.

2. *tesseract.py*: Předložený textový soubor obsahuje seznam složek a seznam obrázků v nich. Program provede OCR a výsledek uloží do textového souboru pojmenovaného podle názvu obrázku.

3. *accuracy.py*: Podle seznamu složek a seznamu obrázků, ve kterém jsou přiloženy i správné odpovědi, provede výpočet úspěšnosti Tesseractu.

Jako první je třeba znát formát B-MOD, aby jsme s ním mohli pracovat. Ve složce se nachází textové soubory(rozdělené podle obtížnosti a typu), které mají na každé řádce unikátní název obrázku a (kromě testovacích sad) mezerou oddělený textový řetězec, který je správným řešením. Dále je tam jen složka se všemi obrázky. Protože jsou všechny sady zbytečně velké pro naše účely, tak si jednoduše část (zhruba 2000 řádek) jedné lehké vykopírujeme do vlastního souboru *small.easy*, který budeme využívat jako databázi.

4.3.1 mastercv.py

Následuje příkazy ze souboru *prikazy-pro-opencv.txt*. Ten obsahuje mezerou oddělené informace. První je zdrojová složka, ze které má brát obrázky. Druhá je databáze (neboli seznam) obrázků, v našem případě je to *small.easy*. Jako třetí je název souboru, ze kterého má brát parametry pro jednotlivé funkce (např. *parametry1.txt*). V souboru s parametry jsou údaje na jedné řádce a od sebe oddělené mezerou. A pak následují mezerou oddělené číselné kódy pro funkce knihovny openCV. Tyto kódy jen reprezentují, jaká funkce se mají vykonat (viz 4.3.1)

Proces

1. Otevři soubor s příkazy a pro každou řádku opakuj:

2. Vyzkoušej funkčnost parametrů a ulož si je.
3. Vyzkoušej, jestli všechny kódy příkazů odpovídají nějakým příkazům. A přitom vytvoř systematický název cílové složky.
4. Vyzkoušej, jestli existuje složka se zdrojovými obrázky.
5. Jestli vše sedí pokračuj, jestli ne přeskoč řádku.
6. Vytvoř cílovou složku se *systematickým názvem* (název složky, ze které je původní obrázek a poté pomlčkou odděleny všechny provedené příkazy, a pokud mají parametry, tak jsou v kulatých závorkách za příkazem vzájemně od sebe odděleny čárkou), jestli neexistuje.
7. Jestli existuje soubor s databází obrázků, pokračuj. Jinak přeskoč řádku.
8. Pro každý řádek v databázi obrázků opakuj:
9. Otevři obrázek ze zdrojové složky.
10. Proveď každý příkaz ze seznamu příkazů.
11. Ulož obrázek do cílové složky.
12. Pokračuj na další řádek v databázi obrázků.
13. Po dokončení celé databáze, pokračuj na další řádek příkazů.

Dokumentace kódů funkcí

- *0* : obrazek = cv2.resize(obrazek, None, fx=*par0*, fy=*par0*, interpolation=*interpolace*)
  fx a fy násobí velikost původního obrázku a získávají tak velikost výsledného obrázku. Přísluší jednotlivým osám. Jedná se o desetinná čísla. [33]
  - *01*: *interpolace* = cv2.INTER_AREA
    Vhodné pro zmenšování obrázku.
  - *02*: *interpolace* = cv2.INTER_CUBIC
    Nejlepší pro zvětšení obrázku, ale pomalejší.

  - *03*: *interpolace* = cv2.INTER_LINEAR
    Pro zvětšení obrázku. Rychlejší, ale ne nejlepší.

- *11*: obrazek = cv2.cvtColor(obrazek, cv2.COLOR_BGR2GRAY)
  Převede obrázek do grayscale.

- *21*: obrazek = cv2.fastNlMeansDenoising(obrazek, *par*1,*par*2, *par*3)
  Vyžaduje grayscale obrázek. Desetinné číslo *par*1 určuje sílu filtru. Celá lichá čísla *par*2 a *par*3 jsou parametry určující velikost okna. Výpočetní náročnost se zvětšuje pro vyšší hodnoty.

- *3*: Provede rozmazání (angl. blurring) obrázku. [34]

  - *31*: obrazek = cv2.blur(obrazek, (*par*4, *par*4))
    Použije aritmetický průměr pole, kde výška a šířka je rovna *par*4, což je celé číslo.

  - *32*: obrazek = cv2.bilateralFilter(obrazek, *par*5, *par*6, *par*7)
    Je efektivní pro odstranění šumu a zároveň zachovává ostré hrany. *par*5 je celé číslo určující průměr okolí použitého ve filtraci. *par*6 a *par*7 jsou desetinné hodnoty odpovídající SigmaSpace a SigmaColor. Čím jsou větší, tím větší je síla efektu (viz dokumentace [35]

  - *33*: obrazek = cv2.GaussianBlur(obrazek, (*par*8,*par*8), *par*9)
    *par*8 určuje velikost oblasti, se kterou se počítá, musí být kladné liché číslo. *par*9 určuje standartní deviaci, pokud je nulový, tak se vypočítá z velikosti oblasti

  - *34*: obrazek = cv2.medianBlur(obrazek, *par*10)
    Střed pole o výšce a šířce *par*10 je určen mediánem hodnot v tomto poli. *par*10 by měl být kladná liché číslo.

- *4* Vstupem by měl být grayscale obrázek. Výsledek je binarizovaný obrázek [36]

  - *41*: obrazek = cv2.adaptiveThreshold(obrazek, 255,
    cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, *par*11, *par*12)
    Celé číslo *par*11 určuje velikost bloku. Spočítá se průměr a odečte se od něj konstanta *par*12.

  - *42*: obrazek = cv2.adaptiveThreshold(obrazek, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, *par*13, *par*14)

Celé číslo *par*13 určuje velikost bloku. Spočítá se průměr a odečte se od něj konstanta *par*14.

- *43*: ret, obrazek = cv2.threshold(obrazek, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
  Jedná se o binarizaci s globálním prahem, který je určen pomocí Otsuovi metody. Nevyžaduje žádný parametr. Funkce vrací 2 hodnoty a až druhá je upravený obrázek.

4.3.2 tesseract.py

Očekává ve vstupním souboru 2 informace oddělené mezerou na každé řádce. První je název složky, ze které má brát obrázky. Druhý je název textového souboru, ve kterém je uložen databáze názvů obrázků. Pak postupně každý obrázek předloží Tesseractu na rozpoznání a výsledek uloží do zdrojové složky do textového souboru, který nese stejný název jako obrázek plus koncovka '.tess'.
V některých případech Tesseract rozezná znaky, které není Python schopen uložit. V tomto případě prostě uloží do souboru *Encoding_failed* a ztráta pár souborů, která tímto vznikne, je vzhledem k velikosti našeho vzorku zanedbatelná.

4.3.3 accuracy.py

Míra chybovosti ve slovech

Vypočítává zjednodušenou obdobu *word error rate* (česky *míra chybovosti ve slovech*). V tomto případě je to počet slov, které OCR nerozpoznalo, děleno počtem slov, které mělo rozpoznat. To znamená, že dostaneme číslo od 0 (ideální stav, kdy bylo vše rozpoznáno správně) do 1 (nejhorší stav, kdy nebylo rozpoznáno ani jedno slovo správně). To můžeme také převést do procent pro lepší čtení člověkem.
V praxi se obvykle počítá, kolika změnami (tj. přidání, odebrání nebo přepsání slova) se dosáhne cílového textu z textu získaného pomocí OCR. Tato hodnota se potom vydělí počtem slov ve vzoru. Výsledná hodnota ale může být větší než jedna. Např. když místo jednoho slova to nalezne 2 jiná, tak k opravě textu jsou potřeba 2 operace, a proto je míra chybovosti rovna 2 (neboli 200%). [37]

Postup

Předpokládá stejný formát jako *tesseract.py*. Rozdělí cílový text podle mezer na jednotlivá slova. To samé provede pro text rozeznaný Tesseractem. A každé toto slovo porovnává s cílovými slovy. Pokud se shodují, tak se toto cílové slovo odstraní. Toto pokračuje dokud to neprojde všechna přeložená slova, nebo dokud už nezbývají žádná cílová slova. Poté se porovná počet nenalezených cílových slov s jejich původním počtem a tím zjistíme jakou část slov Tesseract nerozpoznal. Čím nižší, tím lepší výsledek.

Nedostatky a problémy

Tento postup samozřejmě není úplně přesný, protože nebere v potaz pořadí slov, takže může dojít k falešně pozitivním shodám, ale toto hrozí jen pro velmi krátká slova, protože pravděpodobnost, že náhodou trefí dlouhé slovo je minimální. Dále vůbec nerozlišuje mezitím, je-li ve slově chyba jednoho písmene, nebo je celé úplně špatně. Ale toto je snadno vykompenzováno dostatečně velkou sadou.

Dalším problémem je, že v některých obrázcích, jsou částí sousedních řádek, které Tesseract správně identifikuje jako text, ale není schopen je rozeznat, takže vzniknou jen skupiny náhodných znaků, které se nám uloží na jiné řádky do souboru. Mohli bychom proto kontrolovat jen jednu řádku, ale protože nevíme kolikátá to je a navíc v některých případech Tesseract rozdělí jednu řádku kvůli jemnému zahnutí textu na dvě, tak musíme nahradit konce řádků mezerami a tento výsledný text využít k porovnání. Výpočetní čas je stejně velmi krátký (zejména oproti Tesseractu), takže je to minimální problém.

Tvar výstupu

Výsledek je uložen do textového souboru, kde každému řádku připadá jedna složka a tabulátory oddělené další informace. Tabulátory jsou zvoleny, aby šly výsledky snadno vykopírovat do tabulkového editoru. Z hlediska dalšího zpracování Pythonem nebo jiným programem je stejně snadné rozdělit řádku podle tabulátorů jako podle mezer. Jako další informace je tam uložen poměr počtu nenalezených cílových slov a počtu všech cílových slov.

Další informací je průměr tohoto poměru pro jednotlivé řádky. Tato informace není, tak důležité jako předchozí, protože neuvažuje počet slov v jednotlivých řádkách. Při dostatečně

velkém vzorku, který má stejně reprezentované dlouhé a krátké řádky, by se tato hodnota měla přiblížit první.

Jako další je opět první informace, ale tentokrát ve tvaru neusměrněného zlomku. A jako poslední je počet řádků, které byly využity. Z tohoto čísla můžeme zjistit, kolik platných souborů zbylo po vyloučení těch, které vyhodili nějakou neočekávanou chybu (nejčastěji Tessearact nalezl neuložitelný znak). Můžeme tak zjistit reálnou velikost našeho vzorku.

5 Výsledky měření

Nakonec bylo porovnáno zhruba 120 řádků příkazů, které byly puštěny na databázi asi 2 000 jmen obrázků. Přestože v některých případech OpenCV vykonává operace zbytečně, tak běželo relativně rychle. Jediný trochu pomalejší příkaz je *21* (odstranění šumu). Nejpomalejší částí je určitě Tesseract, kterému trvalo projet databázi o cca 200 000 obrázcích asi 43 hodin (oproti asi 5-8 hodinám OpenCV). Závěrečné určení přesnosti trvalo sotva pár minut. Kompletní tabulka je k nahlédnutí na konci maturitní práce A.1.

Výtah těch nejlepších postupů je v tabulce 5.1. Jak je vidět jen pár jich překonalo výsledek Tesseractu na neupravených obrázcích (tj. složka s názvem *lines-*). Úplně nejlepšího výsledku dosáhlo změnění velikosti obrázku na 4 násobnou. To může souviset s tím, že Tesseract má z nějakého důvodu optimální velikost písma, při které podává nejlepší výkon (podrobnosti viz [38]). Překvapivě další úprava už výsledek jen zhoršovala i v případě, kdy se jednalo pouze o převedení do grayscale (tj. složka s názvem *lines-02(04)-11-*). Dalším neúspěchem byla funkce na odstranění šumu a také všechny binarizační funkce. V podstatě vždy, když byly použity, tak došlo ke zhoršení výsledků.

Cíl naší práce jsme splnili a dokázali jsme najít takové úpravy, které zlepšují výsledky OCR. Nejlepší úprava (tj. 4 násobné zvětšení obrázku) nám zajistila počet rozeznaných větší slov o 0.86%. Další úspěšné kombinace už upravovali zvětšený obrázek, takže ve skutečnosti oproti předchozímu stavu výsledek zhoršovali. Jednou z nich je *31* (tj. rozmazání využívající aritmetického průměru pole), která překonala původní obrázky s 2 různými parametry (5 a 7). Můžeme vidět klesající tendenci s klesající hodnotou parametru, takže je možné, že lze dosáhnout ještě lepších výsledků pomocí této funkce, pokud zmenšíme parametr, a možná i překonat *lines-02(4)-* .

Další kombinací, která překonala původní obrázky je bilaterální rozmazávání (*lines-02(4)-11-32(7,60,60)-*). Protože tato funkce má 3 parametry, je těžké předpokládat, jestli (a případně jaká) je nějaká jejich kombinace lepší. Jediný způsob je prostě vyzkoušet několik možností a změřit výsledky a zjistit, jestli je vidět nějaká tendence.

Dále jsme našli množství úprav, které naopak výsledky zhoršují. Takže pokud chceme úpravou obrázku zajistit lepší výsledky, je třeba zvolit správné funkce a najít ty správné parametry, aby jsme vůbec měli šanci na zlepšení. V případě, že výběr neuděláme pořádně, tak si naopak pravděpodobně výsledky Tesseractu ještě zhoršíme, a pokud jsme neopatrní , tak můžeme i opravdu výrazně (viz konec tabulky A.1).

Název složky	Nerozpoznaných slov v %	Průměrný řádek v %
lines-02(4)-	28.48375654523957	31.147177841597875
lines-02(4)-11-31(5,5)-	28.772635814889334	31.44496288610835
lines-02(4)-11-31(7,7)-	29.146799798059124	31.813984765368506
lines-02(4)-11-32(7,60,60)-	29.190832817163127	31.78324795699035
lines-02(4)-11-	29.212795549374132	31.81483729252857
lines-	29.342969051690847	31.873574800141737
lines-11-	29.363382135219606	31.760516223353818
lines-02(4)-11-32(9,75,75)-	29.42031415953938	32.124600424025984
lines-02(4)-11-33(7,7,7)-	29.59177943736313	32.18702065357416
lines-02(4)-11-31(9,9)-	29.606325519238265	32.249597044580507
lines-02(4)-11-32(5,90,90)-	29.661776234740735	32.39996911754059
lines-02(4)-11-33(9,9,9)-	29.69534650150653	32.396897053992784

Tabulka 5.1: Nejlepší seznamy příkazů

Závěr

V teoretické části jsou vysvětleny základní informace OCR a AI. Jsou zde popsány techniky optického rozpoznávání znaků. Skenování fyzického textu a úprava získaného obrázku jeho předzpracováním a následné rozdělení znaků, tak aby byly zachovány všechny důležité vlastnosti pro jejich rozeznání. V části o umělých inteligencích je kromě neuronových sítí popsána také mlhavá logika.

V praktické části jsem vytvořil systém na provedení velkého množství různých úprav obrázků pomocí OpenCV. Výsledné obrázky jsou potom předloženy Tesseractu, který zkusí rozeznat, co je na obrázku za text. A poslední část systému porovná úspěšnost Tesseractu na různě upravených obrázcích.

Pomocí tohoto systému jsem zjistil, že je úpravou obrázků pomocí OpenCV možné zlepšit úspěšnost Tesseractu, protože jsem takovou úpravu našel. Jedná se o zvětšení obrázku. Nalezl jsem i další kombinace úprav, které jsou lepší než původní obrázky, ale všechny vycházejí již ze zvětšených obrázků a oproti nim dochází ke zhoršení. Některé typy rozmazávání vykazovali potenciál, ke zlepšení oproti pouhému zvětšení obrázku při vhodné úpravě parametrů. Přestože naměřené výsledky nemusí být obecně platné v rozdílné sadě obrázků, tak vytvořený systém může být použit k přiblížení se nejlepší kombinaci funkcí a jejich parametrů i pro jiné sady dat.

Na tuto práci lze navázat ještě větším zautomatizováním a provázáním procesů tak, aby obrázky upravené OpenCV byly rovnou předány Tesseractu. Tím se ušetří místo na disku, což je důležité obzvláště při využívání větších datasetů a velkého množství kombinací příkazů OpenCV. Dále by mohl program sám z naměřených hodnot určit, které úpravy jsou nejvýhodnější a u nich systematicky zkoušet různé hodnoty parametrů tak, aby bylo nalezeny optimální příkazy, které umožní Tesseractu nejlepší rozpoznání.

Pro zájemce je celá práce a zdrojové kódy k dispozici na: `https://github.com/AmbryTheBlue/OCR-image-compraison`

Bibliografie

[1] Wikipedie. *Optické rozpoznávání znaků — Wikipedie: Otevřená encyklopedie*. [Online; navštíveno 13. 01. 2020]. 2019. URL: https://cs.wikipedia.org/w/index.php?title=Optick%C3%A9_rozpozn%C3%A1v%C3%A1n%C3%AD_znak%C5%AF&oldid=17658829.

[2] Arindam Chaudhuri et al. "Optical character recognition systems for different languages with soft computing". In: (2017).

[3] Line Eikvil. "Optical character recognition". In: (1993).

[4] Lotfi A Zadeh. "Soft computing and fuzzy logic". In: *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi a Zadeh*. World Scientific, 1996, s. 796–804.

[5] Micorsoft. URL: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/ (cit. 30. 01. 2020).

[6] Google. URL: https://cloud.google.com/vision (cit. 30. 01. 2020).

[7] Amazon. URL: https://aws.amazon.com/rekognition/ (cit. 30. 01. 2020).

[8] Stuart J Russell a Peter Norvig. *Artificial intelligence: a modern approach*. 3. vyd. Malaysia; Pearson Education Limited, 2016.

[9] Nils J Nilsson. *Introduction to machine learning: An early draft of a proposed textbook*. USA; Stanford University, 1996.

[10] David Kriesel. "A brief introduction to neural networks. 2007". In: (2007). URL: http://www.dkriesel.com.

[11] Michael A Nielsen. *Neural networks and deep learning*. 2015. URL: http://neuralnetworksa
com/.

[12] Wikimedia Commons, uživatel: Qef. *Logistická křivka*. 2008. URL: https://commons.wikimedia.org/w/index.php?title=File:Logistic-curve.svg&oldid=365963933 (cit. 05. 02. 2020).

[13] Wikipedie. *Heavisideova funkce — Wikipedie: Otevřená encyklopedie*. [Online; navštíveno 25. 12. 2019]. 2018. URL: https://cs.wikipedia.org/w/index.php?title=Heavisideova_funkce&oldid=15733913.

[14] Fjodor Van Veen. *The Neural Network Zoo*. 2016. URL: https://www.asimovinstitute.org/neural-network-zoo/ (cit. 25. 12. 2019).

[15] Wikipedie. *Účelová funkce — Wikipedie: Otevřená encyklopedie*. [Online; navštíveno 27. 12. 2019]. 2019. URL: https://cs.wikipedia.org/w/index.php?title=%C3%9A%C4%8Delov%C3%A1_funkce&oldid=17923117.

[16] 3Blue1Brown. *Deep learning, chapter 2: Gradient descent, how neural networks learn*. 2017. URL: https://youtu.be/IHZwWFHWa-w (cit. 28. 12. 2019).

[17] 3Blue1Brown. *Deep learning, chapter 4: Backpropagation calculus*. 2017. URL: https://youtu.be/tIeHLnjs5U8 (cit. 28. 12. 2019).

[18] 3Blue1Brown. *Deep learning, chapter 3: What is backpropagation really doing?* 2017. URL: https://youtu.be/Ilg3gGewQ5U (cit. 28. 12. 2019).

[19] Přispěvatelé Wikipedie. *Fuzzy logika*. 2019. URL: https://cs.wikipedia.org/w/index.php?title=Fuzzy_logika&oldid=17400103 (cit. 05. 02. 2020).

[20] L. A. Zadeh. "Fuzzy logic". In: *Computer* 21.4 (dub. 1988), s. 83–93. ISSN: 1558-0814. DOI: 10.1109/2.53.

[21] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (říj. 2017). Blogpost od DeepMind s linkem na veřejně přísupnou studii: https://deepmind.com/blog/article/alphago-zero-starting-scratch, s. 354–359. ISSN: 0028-0836. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

[22] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), s. 1140–1144. ISSN: 0036-8075. DOI: 10.1126/science.aar6404. eprint: https://science.sciencemag.org/content/362/6419/1140.full.pdf. URL: https://science.sciencemag.org/content/362/6419/1140.

[23] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (lis. 2019). Blogpost od DeepMind s linkem na veřejně přísupnou studii : https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning, s. 350–354. ISSN: 0028-0836. DOI: 10.1038/s41586-019-1724-z. URL: https://doi.org/10.1038/s41586-019-1724-z.

[24] OpenAI et al. “Dota 2 with Large Scale Deep Reinforcement Learning”. In: (2019). arXiv: 1912.06680. URL: https://arxiv.org/abs/1912.06680.

[25] Noam Brown a Tuomas Sandholm. “Facebook, Carnegie Mellon build first AI that beats pros in 6-player poker”. In: (2019). URL: https://ai.facebook.com/blog/pluribus-first-ai-to-beat-pros-in-6-player-poker/ (cit. 30. 12. 2019).

[26] megapro17. *Tesseract OCR Wiki: ReadMe*. 2018. URL: https://github.com/tesseract-ocr/tesseract/blob/master/README.md (cit. 15. 02. 2020).

[27] Samuel Hoffstaetter. *Python Tesseract*. 2020. URL: https://github.com/madmaze/pytesseract (cit. 30. 01. 2020).

[28] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[29] *About OpenCV*. URL: https://opencv.org/about/ (cit. 30. 01. 2020).

[30] Project PERO. *Brno Mobile OCR Dataset*. Obsahuje možnost stáhnutí datasetu. 2019. URL: https://pero.fit.vutbr.cz/brno_mobile_ocr_dataset (cit. 05. 02. 2020).

[31] Martin Kišš, Michal Hradiš a Oldřich Kodym. “Brno Mobile OCR Dataset”. In: *CoRR* abs/1907.01307 (2019). arXiv: 1907.01307. URL: http://arxiv.org/abs/1907.01307.

[32] tesseract ocr. *Improving the quality of the output*. 2020. URL: https://tesseract-ocr.github.io/tessdoc/ImproveQuality (cit. 15. 02. 2020).

[33] *Geometric Image Transformations*. URL: https://docs.opencv.org/master/da/d54/group__imgproc__transform.html#ga47a974309e9102f5f08231edc7e7529d (cit. 16. 02. 2020).

[34] *Smoothing Images*. URL: https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html (cit. 16. 02. 2020).

[35] *Image Filtering*. URL: https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed (cit. 16. 02. 2020).

[36] *Image Thresholding*. URL: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html (cit. 16. 02. 2020).

[37] Wikipedia contributors. *Word error rate — Wikipedia, The Free Encyclopedia*. 2020. URL: https://en.wikipedia.org/w/index.php?title=Word_error_rate&oldid=939575741 (cit. 16. 02. 2020).

[38] Willus Dotkom. *Optimal image resolution (dpi/ppi) for Tesseract 4.0.0 and eng.traineddata?* 2018. URL: https://groups.google.com/forum/#!msg/tesseract-ocr/Wdh_JJwnw94/24JHDYQbBQAJ (cit. 15. 02. 2020).

[39] *Color Space Conversions*. URL: https://docs.opencv.org/master/d8/d01/group__imgproc__color__conversions.html#gga4e0972be5de079fed4e3a10e24ef5ef0a353a4b

[40] *Image Denoising*. URL: https://docs.opencv.org/master/d5/d69/tutorial_py_non_local_means.html (cit. 16. 02. 2020).

Zkratky

AI angl. Artificial Intelligence, v češtině Umělá Inteligence, definice na 3 na straně 16. 6, 10, 16, 24, 25, 35

angl. anglicky, anglický. 10–22, 24, 25, 27–29, 35, 36

ANN angl. Artificial Neural Network = Umělá neuronová síť, zde je větší důraz na to, že se nejedná o síť živočišného původu, oproti NN. Podrobněji na 3.3 na straně 18. 14, 18, 35

atd. a tak dále. 23, 28

B-MOD zkr. angl. Brno Mobile OCR Dataset. Jedná se o velkou sadu obrázků, více 4.1.3. 28

FFN angl. Feed-foorward Network, v češtině Dopředná Neuronová síť, typ NN. Více informací na straně 21. 20–22

GA Genetické algoritmy (angl. genetic algorithm(s)), algoritmy inspirované přírodní evolucí, viz 3.5. 23

k. kapitola, v kapitole. 19, 22

např. například. 11, 14, 16, 17, 23, 28–30

NN angl. Neural Network(s), v češtině Neuronová(/é) síť(ě). Také ANN. Jeden z typů AI, název kvůli podobnosti s živočišnými nervovými soustavami. Podrobněji na 3.3 na straně 18. 18, 19, 35

NS Neuronová(/é) síť(ě). Angl. Neural Network , také ANN. Jeden z typů AI, název kvůli podobnosti s živočišnými nervovými soustavami. Podrobněji na 3.3 na straně 18. 5, 15, 17, 18, 20–22

OCR angl. Optical Character Recognition, v češtině optické rozpoznávání znaků, více na na straně 10. 5, 10–12, 14, 15, 27, 28, 30

s. také "str.", strana(/y), na straně(/ách). 10, 12–14, 16–18, 21–23, 36

str. také "s.", strana(/y), na straně(/ách). 10–15, 17, 18, 20, 21, 23, 24, 36, 37

tj. to jest. 13–15, 17, 18, 21, 27

tzn. to znamená. 19

tzv. takzvaný, takzvané. 21, 27

UTF-8 zkr. angl. Unicode Transformation Format. Je to způsob kódování znaků Unicode. To je norma, která zahrnuje většinu písem používaných na Zemi. 27

viz podívej se (viz + nominativ/akuzativ); druhá osoba jednotného čísla rozkazovacího způsobu slovesa vidět. 10, 15, 22, 23, 25, 27, 29, 35

zkr. zkratka, zkráceně, zkráceno. 16, 18, 28, 35, 36

Seznam obrázků

Seznam tabulek

Zdrojové kódy programů

Přílohy

1. Kód programu
2. Tabulka všech naměřených hodnot
3. Vlastní program
4. Dokumentace
5. Část testovacích dat

A Přiložené soubory

A.1 mastercv.py

Listing A.1: mastercv.py

```
1  #Python 3.7
2  #Autor: Jakub Ambroz, Gymnazium Jirovcova 8
3  #Soucast maturitni práce: Umela inteligence a OCR
4  #Datum: 19.2.20
5  #verze: 1.0
6  #github: https://github.com/AmbryTheBlue/OCR-image-compraison
7  #popis: Program pouzije prikazy OpenCV podle souboru prikazy_path, popis formatu tohoto souboru je v
       ↪ dokumentaci
8
9  import os
10 import errno
11 import cv2
12
13 path = "E:/MP/datasets/pero"
14 prikazy_path = path + "/prikazy-pro-opencv.txt"
15
16 with open(prikazy_path, 'r') as prikazy:
17     radka_prikazu = 0
18     status = ""
19     seznam_vytvorenych_slozek = ""
20     for line in prikazy:
21         pole = line.split(" ")
22         zdroj_path = path + "/" + pole[0]
23         db_path = path + "/" + pole[1]
24         par_path = path + "/" + pole[2]
25         cil_slozka_path = path + "/" + pole[0]
26
```

```
27          #test zda jsou v poradku parametry
28          chyba_v_par = 0
29          pole_par = []
30          try:
31              with open(par_path) as parametry:
32                  for line in parametry:
33                      pole_par = line.split(" ")
34          #print(pole_par)
35          except:
36              chyba_v_par = 1
37              print("Nefunkcni paramtery")
38
39
40          #test zda-li je zadani korektni
41          chyba_v_prikazech = 0
42          for i in range(3,len(pole)):
43              try:
44                  ukon = int(pole[i])
45                  pole[i] = int(pole[i])
46                  if(ukon<10):#upscale
47                      print("resize")
48                      if(ukon==1):
49                          print("INTER_AREA")
50                          cil_slozka_path = cil_slozka_path + "01({})-".format(pole_par[0])
51                      elif(ukon==2):
52                          print("INTER_CUBIC")
53                          cil_slozka_path = cil_slozka_path + "02({})-".format(pole_par[0])
54                      elif(ukon==3):
55                          print("INTER_LINEAR")
56                          cil_slozka_path = cil_slozka_path + "02({})-".format(pole_par[0])
57                      else:
58                          print("chyba(neznamy prikaz) v prikazy_pro_opnecv.txt")
59                          chyba_v_prikazech = chyba_v_prikazech +1
60                  elif(ukon<20):
61                      if(ukon==11):
62                          print("to_gray")
63                          cil_slozka_path = cil_slozka_path + "11-"
64                      else:
65                          print("chyba(neznamy prikaz) v prikazy_pro_opnecv.txt")
```

```
66                      chyba_v_prikazech = chyba_v_prikazech + 1
67                  elif(ukon<30):
68                      if(ukon==21):
69                          print("noise_removal")
70                          cil_slozka_path = cil_slozka_path + "21({},{},{})-".format(pole_par[1],
                                ↪ pole_par[2],pole_par[3])
71                      else:
72                          print("chyba(neznamy prikaz) v prikazy_pro_opnecv.txt")
73                          chyba_v_prikazech = chyba_v_prikazech + 1
74                  elif(ukon<40):
75                      print("blur")
76                      if(ukon==31):
77                          print("avg")
78                          cil_slozka_path = cil_slozka_path + "31({},{})-".format(pole_par[4],
                                ↪ pole_par[4])
79                      elif(ukon==32):
80                          print("bilateral")
81                          cil_slozka_path = cil_slozka_path + "32({},{},{})-".format(pole_par[5],
                                ↪ pole_par[6], pole_par[7])
82                      elif(ukon==33):
83                          print("gauss")
84                          cil_slozka_path = cil_slozka_path + "33({},{},{})-".format(pole_par[8],
                                ↪ pole_par[8], pole_par[9])
85                      elif(ukon==34):
86                          print("median")
87                          cil_slozka_path = cil_slozka_path + "34({})-".format(pole_par[10])
88                      else:
89                          print("chyba(neznamy prikaz) v prikazy_pro_opnecv.txt")
90                          chyba_v_prikazech = chyba_v_prikazech + 1
91                  elif(ukon<50):
92                      print("threshold")
93                      if(ukon==41):
94                          print("adaptive_mean")
95                          cil_slozka_path = cil_slozka_path + "41({},{})-".format(pole_par[11],
                                ↪ pole_par[12])
96                      elif(ukon==42):
97                          print("adaptive_gauss")
98                          cil_slozka_path = cil_slozka_path + "42({},{})-".format(pole_par[13],
                                ↪ pole_par[14])
```

```
99                  elif(ukon==43):
100                     print("otsu")
101                     cil_slozka_path = cil_slozka_path + "43-"
102                 else:
103                     print("chyba(neznamy prikaz) v prikazy_pro_opnecv.txt")
104                     chyba_v_prikazech = chyba_v_prikazech + 1
105             except:
106                 print("chyba(prikaz neni cislo) v prikazy_pro_opnecv.txt")
107                 chyba_v_prikazech = chyba_v_prikazech + 1
108
109
110
111         #otestovani zda-li existuje zdrojova slozka
112         if os.path.exists(zdroj_path):
113             print("Zdrojova slozka existuje")
114             existuje_slozka = True
115         else:
116             print("Zdrojova slozka nenalezena")
117             existuje_slozka = False
118
119
120         #je-li vse v poradku spusti se program pro tento radek prikazu
121         if (0 < chyba_v_prikazech or 0 < chyba_v_par or existuje_slozka==False):
122             print("Vysktla se chyba/y ({}) v prikazy-pro-opencv.txt nebo v paramterech ({}) nebo
                    ↪ slozka, ze ktere cerpame obrazky - existence({})".format(chyba_v_prikazech,
                    ↪ chyba_v_par,existuje_slozka))
123             print("Preskakuji tuto radku a pokracuji na dalsi ")
124             status = status + "\n\nPrikazova radka c.{} nebyla nactena uspesne. \n Vysktla se chyba/y
                    ↪ ({}) v prikazy-pro-opencv.txt nebo v paramterech ({}) nebo slozka, ze ktere
                    ↪ cerpame obrazky - existence({}) .\n Radka bude preskocena".format(radka_prikazu
                    ↪ , chyba_v_prikazech, chyba_v_par, existuje_slozka)
125         else:
126             print("Vse v poradku oteviram databazi")
127             status = status + "\n\nPrikazova radka c.{} uspesne nactena.".format(radka_prikazu)
128
129             print("{} \nVytvoreni cilove slozky, nebo overeni jeji existence".format(cil_slozka_path))
130             ###Vytvor slozku jestli neexistuje
131             if not os.path.exists(cil_slozka_path):
132                 try:
```

```
133                     os.makedirs(cil_slozka_path, 0o700)
134                 except OSError as e:
135                     if e.errno != errno.EEXIST:
136                         raise
137             status = status + "\nSlozka {} uspesne vytvorena.".format(cil_slozka_path)
138 
139             seznam_vytvorenych_slozek = seznam_vytvorenych_slozek + "\n" + cil_slozka_path.split(
                    ↪ "/")[4] + " " + pole[1]
140 
141             #Otevreni databaze
142             if os.path.exists(db_path):
143                 status = status + "\n Databaze otevrena, probiha uprava"
144                 with open(db_path, 'r') as db:
145                     print(status)
146                     db_radky = 0
147                     for line in db:
148                         db_radky = db_radky + 1
149                         img_name = line.strip().split(" ")[0]
150                         this_img_path = zdroj_path + "/" + img_name
151                         obrazek = cv2.imread(this_img_path)
152                         print(this_img_path)
153                         for i in range(3, len(pole)):
154                             ukon = int(pole[i])
155                             if (ukon < 10):
156                                 #print("resize")
157                                 if (ukon == 1):
158                                     #print("INTER_AREA")
159                                     obrazek = cv2.resize(obrazek, None, fx=float(pole_par[0]), fy=float(
                                            ↪ pole_par[0]), interpolation=cv2.INTER_AREA)
160                                 elif (ukon == 2):
161                                     #print("INTER_CUBIC")
162                                     obrazek = cv2.resize(obrazek, None, fx=float(pole_par[0]), fy=float(
                                            ↪ pole_par[0]), interpolation=cv2.INTER_CUBIC)
163                                 elif (ukon == 3):
164                                     #print("INTER_LINEAR")
165                                     obrazek = cv2.resize(obrazek, None, fx=float(pole_par[0]), fy=float(
                                            ↪ pole_par[0]), interpolation=cv2.INTER_LINEAR)
166                             elif (ukon < 20):
167                                 if (ukon == 11):
```

```
168                         #print("to_gray")
169                         obrazek = cv2.cvtColor(obrazek, cv2.COLOR_BGR2GRAY)
170                     elif (ukon < 30):
171                         if (ukon == 21):
172                             #print("noise_removal")
173                             obrazek = cv2.fastNlMeansDenoising(obrazek, float(pole_par[1]), int
                                    ↪ (pole_par[2]), int(pole_par[3]))
174                     elif (ukon < 40):
175                         #print("blur")
176                         if (ukon == 31):
177                             #print("avg")
178                             obrazek = cv2.blur(obrazek, (int(pole_par[4]), int(pole_par[4])))
179                         elif (ukon == 32):
180                             #print("bilateral")
181                             obrazek = cv2.bilateralFilter(obrazek, int(pole_par[5]), float(
                                    ↪ pole_par[6]),float(pole_par[7]))
182                         elif (ukon == 33):
183                             #print("gauss")
184                             obrazek = cv2.GaussianBlur(obrazek, (int(pole_par[8]), int(
                                    ↪ pole_par[8])), float(pole_par[9]))
185                         elif (ukon == 34):
186                             #print("median")
187                                 obrazek = cv2.medianBlur(obrazek, int(pole_par[10]))
188                     elif (ukon < 50):
189                         #print("threshold")
190                         if (ukon == 41):
191                             #print("adaptive_mean")
192                             try:
193                                 obrazek = cv2.cvtColor(obrazek, cv2.COLOR_BGR2GRAY)
194                             except:
195                                 print("Already grayscale")
196                             obrazek = cv2.adaptiveThreshold(obrazek, 255, cv2.
                                    ↪ ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY
                                    ↪ , int(pole_par[11]), float(pole_par[12]))
197                         elif (ukon == 42):
198                             #print("adaptive_gauss")
199                             try:
200                                 obrazek = cv2.cvtColor(obrazek, cv2.COLOR_BGR2GRAY)
201                             except:
```

```
202                                         print("Already grayscale")
203                                     obrazek = cv2.adaptiveThreshold(obrazek, 255, cv2.
                                            ↪ ADAPTIVE_THRESH_GAUSSIAN_C, cv2.
                                            ↪ THRESH_BINARY, int(pole_par[13]), float(pole_par[14]))
204                                 elif (ukon == 43):
205                                     #print("otsu")
206                                     try:
207                                         obrazek = cv2.cvtColor(obrazek, cv2.COLOR_BGR2GRAY)
208                                     except:
209                                         print("Already grayscale")
210                                     ret, obrazek = cv2.threshold(obrazek, 0, 255, cv2.THRESH_BINARY
                                            ↪  + cv2.THRESH_OTSU)
211                             this_img_path_new = cil_slozka_path + "/" + img_name
212                             cv2.imwrite(this_img_path_new, obrazek)
213                             print("db_radka ({}), zapis do souboru({})".format(db_radky,
                                    ↪ this_img_path_new))
214                             if(db_radky%100==0):
215                                 print(status)
216                     status = status + "\n Databaze byla uspesne vytvorena v nove slozce"
217
218                 else:
219                     print("Databaze nenalezena. Preskakuji")
220                     status = status + "\n Databaze nenalezena. Preskoceno"
221
222             radka_prikazu = radka_prikazu + 1
223
224 print("Ono to vyslo!!!")
225 print(status)
226 print(seznam_vytvorenych_slozek)
```

A.2 tesseract.py

Listing A.2: tesseract.py

```
1 #Python 3.7
2 #Autor: Jakub Ambroz, Gymnazium Jirovcova 8
3 #Soucast maturitni práce: Umela inteligence a OCR
4 #Datum: 19.2.20
```

```
5  #verze: 1.0
6  #github: https://github.com/AmbryTheBlue/OCR-image-compraison
7  #popis: Vola Tesseract k rozeznani textu z obrazku ve slozce imgpath z databaze obrazku dbpath. Vysledky
       ↪ ulozi do textoveho souboru. Vyzaduje nainstalovany Tesseract na pocitaci.
8
9  try:
10     from PIL import Image
11 except ImportError:
12     import Image
13 import pytesseract
14
15 pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'
16 path = "E:/MP/datasets/pero"
17 list_path = path + "/soubory-pro-tesseract.txt"
18
19 chyba_zapis =0
20 chyba_tesseract =0
21 with open(list_path, 'r') as f_master:
22     j = 0
23     for radka in f_master:
24         pole = radka.split(" ")
25         imgpath = path + "/" + pole[0].strip()
26         dbpath = path + "/" + pole[1].strip()
27         with open(dbpath,'r') as f_read:
28             i = 0
29             for line in f_read:
30                 i = i +1
31                 pole_imgname = line.strip().split(" ")
32                 imgname = pole_imgname[0]
33                 this_imgpath = imgpath + "/" + imgname
34                 filepath = this_imgpath + ".tess"
35                 print("(" + str(j) + ") " + str(i) + " : " + filepath)
36                 try:
37                     konverze = pytesseract.image_to_string(Image.open(this_imgpath),lang="eng")
38                     #print(konverze.strip())
39                     with open(filepath, 'w') as f_write:
40                         try:
41                             f_write.write(konverze)
42                             print(konverze)
```

```
43                          except:
44                              f_write.write("Encoding_failed")
45                              print("Encoding_failed:")
46                              print(konverze)
47                              chyba_zapis = chyba_zapis+1
48                      except:
49                          print(str(i) + " : " + "Chyba v tesseractu(asi soubor neexistuje)")
50                          chyba_tesseract = chyba_tesseract + 1
51              j = j + i
52              print("Chyb v zapise:" + str(chyba_zapis))
53              print("chyb v tesseractu: " +str(chyba_tesseract))
```

A.3 accuracy.py

Listing A.3: accuracy.py

```
1  #Python 3.7
2  #Autor: Jakub Ambroz, Gymnazium Jirovcova 8
3  #Soucast maturitni práce: Umela inteligence a OCR
4  #Datum: 19.2.20
5  #verze: 1.0
6  #github: https://github.com/AmbryTheBlue/OCR-image-compraison
7  #popis: Spocita uspesnost rozpoznani slov Tesseractem pro ruzné kombinace prikazu. A ulozi do textoveho
       ↪ souboru.
8
9  path = "E:/MP/datasets/pero"
10 list_path = path + "/soubory-pro-evaluaci.txt"
11 cil_path = path + "/procenta.txt"
12
13 celkem_unknown_error = 0
14 cely_text = ""
15 with open(list_path) as seznam_slozek:
16     for line in seznam_slozek:
17         img_path = path + "/" + line.split(" ", 1)[0].strip()
18         #print(img_path)
19         db_path = path + "/" + line.split(" ", 1)[1].strip()
20         with open(db_path) as seznam_obrazku:
21             cislo_radky = 0
```

```
22              celkem_pocet_cilovych_slov = 0
23              celkem_pocet_zbylych_cilovych_slov = 0
24              unknown_error = 0
25              celkem_error_rate = 0.0
26              for radka in seznam_obrazku:
27                  cislo_radky = cislo_radky + 1
28                  cilova_slova = radka.split(" ",1)[1].strip().split(" ")
29                  potess_path = img_path + "/" + radka.strip().split(" ",1)[0] + ".tess"
30                  prelozeny_text_oneline = ""
31                  funkcni_radka = True
32                  try:
33                      with open(potess_path) as prelezeny_text:
34                          for miniradka in prelezeny_text:
35                              prelozeny_text_oneline = prelozeny_text_oneline + miniradka.strip() + " "
36                  except:
37                      #print("Unknown error. Soubor asi neexistuje, ale nevim proc. Asi nejaka chyba
                            ↪ zapisu pri vytvareni souboru.")
38                      unknown_error = unknown_error + 1
39                      funkcni_radka = False
40                  if(funkcni_radka):
41                      prelozena_slova = prelozeny_text_oneline.strip().split(" ")
42                      pocet_cilovych_slov = len(cilova_slova)
43                      if(prelozena_slova[0] != "Encoding_failed"):
44                          celkem_pocet_cilovych_slov = celkem_pocet_cilovych_slov +
                                ↪ pocet_cilovych_slov
45                          for test_slovo in prelozena_slova:
46                              for spravne_slovo in cilova_slova:
47                                  if(test_slovo==spravne_slovo):
48                                      cilova_slova.remove(spravne_slovo)
49                                      break
50                              if(len(cilova_slova)==0):
51                                  break
52                          zbyly_pocet_cilovych_slov = len(cilova_slova)
53                          celkem_pocet_zbylych_cilovych_slov = celkem_pocet_zbylych_cilovych_slov +
                                ↪ zbyly_pocet_cilovych_slov
54                          error_rate = zbyly_pocet_cilovych_slov/pocet_cilovych_slov
55                          celkem_error_rate = celkem_error_rate + error_rate
56                      else:
57                          cislo_radky = cislo_radky - 1
```

```
58                  else:
59                      cislo_radky = cislo_radky - 1
60              #print("pocet nevysvetlitelnych chyb: " + str(unknown_error))
61              celkem_unknown_error = celkem_unknown_error + unknown_error
62              print(img_path)
63              prumer_slovovy_error_rate = celkem_pocet_zbylych_cilovych_slov/
                    ↪ celkem_pocet_cilovych_slov
64              print("Prumerny error rate({}/{}): ".format(str(celkem_pocet_zbylych_cilovych_slov) ,
                    ↪ celkem_pocet_cilovych_slov) + str(prumer_slovovy_error_rate))
65              prumer_radkovy_error_rate = celkem_error_rate / cislo_radky
66              print("Prumerny error rate na radce(pocet {}): ".format(cislo_radky) + str(
                    ↪ prumer_radkovy_error_rate))
67              print()
68              cely_text = cely_text + line.split(" ", 1)[0].strip() + "\t" + str(prumer_slovovy_error_rate
                    ↪ ) + "\t" + str(prumer_radkovy_error_rate) + "\t{}/{}\t{}".format(
                    ↪ celkem_pocet_zbylych_cilovych_slov, celkem_pocet_cilovych_slov, cislo_radky) +
                    ↪ "\n"
69      with open(cil_path, 'w') as soubor:
70          soubor.write(cely_text)
71      print("Vse dobehlo hladce.")
72      print("Az na {} neznamych erroru".format(celkem_unknown_error))
```

A.4 Kompletní tabulka naměřených hodnot

Název složky	Poměr nerozpoznaných slov	Průměrný řádek	Poměr nerozpoznaných slov	Počet řádků
lines-02(4)-	0.2848375654523957	0.31147177841597873	5059/17761	1958
lines-02(4)-11-31(5,5)-	0.28772635814889336	0.3144496288610835	5148/17892	1970
lines-02(4)-11-31(7,7)-	0.29146799798059125	0.3181398476536851	5196/17827	1966
lines-02(4)-11-32(7,60,60)-	0.29190832817163126	0.3178324795699035	5184/17759	1962
lines-02(4)-11-	0.2921279554937413	0.3181483729252857	5251/17975	1977
lines-	0.2934296905169085	0.31873574800141735	5319/18127	1995
lines-11-	0.29363382135219606	0.31760516223353818	5355/18237	2003
lines-02(4)-11-32(9,75,75)-	0.2942031415953938	0.32124600424025984	5263/17889	1968
lines-02(4)-11-33(7,7,7)-	0.2959177943736313	0.3218702065357416	5270/17809	1963
lines-02(4)-11-31(9,9)-	0.29606325519238263	0.3224959704458051	5317/17959	1975
lines-02(4)-11-32(5,90,90)-	0.29661776234740733	0.3239996911754059	5297/17858	1967
lines-02(4)-11-33(9,9,9)-	0.2969534650150653	0.32396897053992785	5322/17922	1974
lines-02(4)-11-34(9)-	0.3009725111023666	0.326157390736683	5354/17789	1963
lines-02(4)-11-34(11)-	0.30341761600444567	0.33111919916946325	5460/17995	1979
lines-02(4)-11-34(7)-	0.30373230373230375	0.33316244704289033	5428/17871	1968
lines-02(4)-11-33(11,11,11)-	0.30629277249207143	0.3374442081132987	5505/17973	1981
lines-02(4)-11-21(3,12,11)-	0.31626912691269127	0.34203520576426194	5622/17776	1961
lines-02(4)-11-21(2,17,7)-	0.3277873932220423	0.35356024444515505	5871/17911	1971

lines-02(4)-11-43-	0.33463437033635635	0.35935778912735766	5830/17422	1927
lines-02(4)-11-43-	0.33463437033635635	0.35935778912735766	5830/17422	1927
lines-02(4)-11-43-	0.33463437033635635	0.35935778912735766	5830/17422	1927
lines-02(4)-11-31(5,5)-43-	0.3408173705451032	0.3643741739331804	5996/17593	1948
lines-02(4)-11-33(7,7,7)-43-	0.34364691161415084	0.3695983695298388	6042/17582	1948
lines-02(4)-11-32(9,75,75)-43-	0.3480196324620477	0.3717901792296982	6098/17522	1942
lines-02(4)-11-34(7)-43-	0.35752383113935543	0.38344194952776406	6301/17624	1946
lines-02(4)-21(3,25,11)-11-	0.3583393622349602	0.3877713976220303	6439/17969	1977
lines-02(4)-11-21(3,25,11)-	0.3592394929953302	0.38985381450112555	6462/17988	1979
lines-02(4)-11-21(3,25,11)-32(5,90,90)-	0.3598510117856349	0.39169366422520757	6473/17988	1979
lines-02(4)-11-21(3,25,11)-32(7,60,60)-	0.3623575201556853	0.392219112779942	6517/17985	1980
lines-43-	0.36273380505731995	0.38578192584403126	6613/18231	2001
lines-02(4)-11-21(5,27,7)-	0.36282590412111015	0.3904459067605987	6471/17835	1968
lines-02(4)-11-21(3,25,11)-33(7,7,7)-	0.36480543339085897	0.39420820806136136	6553/17963	1975
lines-02(4)-11-21(3,25,11)-34(7)-	0.36667959873635203	0.3948610633207491	6616/18043	1983

lines-02(4)-11-21(3,25,11)-31(5,5)-	0.36764052432792715	0.397729864963413	6619/18004	1976
lines-02(4)-11-21(3,25,11)-31(9,9)-	0.3685378300421566	0.39946384196700463	6644/18028	1983
lines-02(4)-11-21(3,25,11)-31(7,7)-	0.3697652372720683	0.39885286724767144	6631/17933	1973
lines-02(4)-11-21(3,25,11)-33(9,9,9)-	0.37304948729380294	0.40712522572066967	6694/17944	1978
lines-02(4)-11-21(3,25,11)-34(9)-	0.37392900856793143	0.40448577185817547	6721/17974	1973
lines-02(4)-11-21(3,25,11)-32(9,75,75)-	0.37512537612838515	0.402608035180929	6732/17946	1977
lines-02(4)-11-21(3,25,11)-34(11)-	0.38103457876273733	0.41043823164496784	6843/17959	1975
lines-02(4)-11-21(3,25,11)-33(11,11,11)-	0.3816518847006652	0.41552696569581904	6885/18040	1982
lines-02(4)-11-21(3,25,11)-31(5,5)-41(15,2)-	0.41113039167079823	0.42522427313352135	6634/16136	1801
lines-02(4)-11-21(3,25,11)-34(7)-41(15,2)-	0.41169561723434506	0.42552277540171846	6660/16177	1803

lines-02(4)-21(3,25,11)-11-33(7,7,7)-41(15,2)-	0.4121469632591852	0.4260183392627991	6596/16004	1792
lines-02(4)-11-21(3,25,11)-41(15,2)-	0.41224565070387903	0.4259137147700568	6706/16267	1814
lines-02(4)-11-21(3,25,11)-33(7,7,7)-41(15,2)-	0.4132451150136038	0.42485091039310285	6683/16172	1801
lines-02(4)-21(3,25,11)-11-31(5,5)-41(15,2)-	0.4138524943101433	0.42905082571433706	6728/16257	1815
lines-02(4)-21(3,25,11)-11-32(9,75,75)-41(15,2)-	0.417137245273754	0.4321916080717336	6796/16292	1820
lines-02(4)-21(3,25,11)-11-34(7)-41(15,2)-	0.41845201238390095	0.430243213629288	6758/16150	1808
lines-02(4)-11-21(3,25,11)-32(9,75,75)-41(15,2)-	0.41994330087513865	0.4357179392216863	6814/16226	1813
lines-41(11,10)-	0.4204489053107594	0.43249906950247186	7624/18133	1993
lines-02(4)-11-21(3,25,11)-32(5,90,90)-43-	0.4404662316658301	0.46839035184464806	7898/17931	1974
lines-02(4)-21(3,25,11)-11-31(5,5)-43-	0.44207998212689903	0.46959230119364065	7915/17904	1970
lines-02(4)-11-21(3,25,11)-31(7,7)-43-	0.4421187052356893	0.46831159695941316	7963/18011	1982

lines-02(4)-11-21(3,25,11)-43-	0.4427704070159658	0.4694409176977171	7876/17788	1961
lines-02(4)-11-21(3,25,11)-43-	0.4427704070159658	0.4694409176977171	7876/17788	1961
lines-02(4)-11-21(3,25,11)-43-	0.4427704070159658	0.4694409176977171	7876/17788	1961
lines-02(4)-11-21(3,25,11)-31(5,5)-43-	0.4437363743082341	0.4724686059411151	7938/17889	1969
lines-02(4)-11-21(3,25,11)-33(7,7,7)-43-	0.44596397598398935	0.47209662509524053	8022/17988	1979
lines-02(4)-21(3,25,11)-11-33(7,7,7)-43-	0.44605878423513695	0.4727759656738389	8013/17964	1975
lines-02(4)-11-21(3,25,11)-32(7,60,60)-43-	0.4485776805251641	0.47565068768498503	7995/17823	1964
lines-21(3,12,11)-	0.4507606964354369	0.47191104476346185	8207/18207	2000
lines-02(4)-11-21(3,25,11)-32(9,75,75)-43-	0.4514727991536277	0.478378272696113	8108/17959	1974
lines-02(4)-21(3,25,11)-11-32(9,75,75)-43-	0.4526309878137811	0.4795586414495701	8060/17807	1962
lines-02(4)-21(3,25,11)-11-34(7)-43-	0.4561296069480013	0.4822165568009477	8193/17962	1979

lines-02(4)-11-21(3,25,11)-31(9,9)-43-	0.45625766187451244	0.48575561663710354	8188/17946	1972
lines-02(4)-11-21(3,25,11)-33(9,9,9)-43-	0.4567170843333147	0.48535371630231766	8183/17917	1971
lines-02(4)-11-21(3,25,11)-34(7)-43-	0.4574568989566479	0.4836055769578867	8199/17923	1972
lines-21(2,17,7)-	0.4598956903650837	0.4850753525644319	8377/18215	2001
lines-02(4)-11-21(3,25,11)-33(11,11,11)-43-	0.4599085535853686	0.4889497489650846	8248/17934	1974
lines-02(4)-11-21(3,25,11)-34(9)-43-	0.4704008049639443	0.49493924456285104	8415/17889	1969
lines-02(4)-11-21(3,25,11)-34(11)-43-	0.4890116019634092	0.512364013917208	8767/17928	1973
lines-42(11,10)-	0.5320226884740349	0.5417840280461401	9661/18159	1996
lines-02(4)-11-21(3,25,11)-42(15,2)-	0.5420509708737864	0.5616486582332507	8933/16480	1839
lines-02(4)-11-21(3,25,11)-34(7)-42(15,2)-	0.5511377844461115	0.5701509621309753	8816/15996	1786
lines-02(4)-11-33(7,7,7)-42(15,2)-	0.5524121316943603	0.5818255683204698	8943/16189	1801

lines-02(4)-21(3,25,11)-11-34(7)-42(15,2)-	0.552469722909137	0.5697575209883775	8713/15771	1771
lines-02(4)-21(3,25,11)-11-31(5,5)-42(15,2)-	0.5546316111280953	0.5726860214597002	9071/16355	1821
lines-02(4)-11-32(9,75,75)-42(15,2)-	0.5547449730259931	0.58186258250371	9049/16312	1814
lines-02(4)-11-21(3,25,11)-31(5,5)-42(15,2)-	0.5580180946019795	0.5752528906926162	9190/16469	1828
lines-02(4)-11-21(3,25,11)-33(7,7,7)-42(15,2)-	0.5647412228549972	0.5826116869211074	9024/15979	1789
lines-02(4)-21(3,25,11)-11-33(7,7,7)-42(15,2)-	0.566656413411258	0.5850132080035514	9211/16255	1812
lines-02(4)-11-21(3,25,11)-32(9,75,75)-42(15,2)-	0.5686501646140715	0.5888417431903382	9327/16402	1824
lines-02(4)-21(3,25,11)-11-32(9,75,75)-42(15,2)-	0.5691082015810277	0.5900173635445262	9215/16192	1802
lines-02(4)-11-34(7)-42(15,2)-	0.5879328501517685	0.6102705536589635	9491/16143	1804
lines-21(5,27,7)-	0.5906014416992241	0.615480013051264	10733/18173	1995
lines-42(9,4)-	0.6108218874612317	0.6172867391425951	11029/18056	1991
lines-41(9,4)-	0.6169635941130907	0.6224120617447394	11151/18074	1988

lines-02(4)-11-31(5,5)-42(15,2)-	0.6428571428571429	0.6644126665552529	10539/16394	1825
lines-21(3,21,11)-	0.678005284015852	0.6913404129347973	12318/18168	1995
lines-02(4)-11-41(9,4)-	0.6960415547196848	0.7158656274198791	11658/16749	1865
lines-02(4)-11-21(3,25,11)-41(9,4)-	0.7126429968585146	0.726946451065322	12023/16871	1878
lines-02(4)-11-33(7,7,7)-41(15,2)-	0.7210310768489521	0.7367060029758425	11972/16604	1836
lines-02(4)-11-32(9,75,75)-41(15,2)-	0.725499581189422	0.7435614893304658	12126/16714	1856
lines-02(4)-11-21(3,25,11)-32(5,90,90)-41(9,4)-	0.7323518978361121	0.7428294590089414	12387/16914	1874
lines-02(4)-11-21(3,25,11)-34(11)-41(9,4)-	0.7421985163386672	0.7510099969920024	12106/16311	1817
lines-21(3,25,11)-	0.756391967518929	0.7632979731275946	13786/18226	2002
lines-02(4)-11-21(3,25,11)-31(9,9)-41(9,4)-	0.7750510055377441	0.7779909838472643	13296/17155	1902
lines-02(4)-11-34(7)-41(15,2)-	0.7761536079281511	0.7942028847383977	12531/16145	1797
lines-02(4)-11-41(11,10)-	0.7944654820856394	0.8046492579571688	13637/17165	1908

lines-02(4)-11-21(3,25,11)-33(11,11,11)-41(9,4)-	0.7961125616478096	0.7997363189174773	13721/17235	1912
lines-02(4)-11-31(5,5)-41(15,2)-	0.8201812688821752	0.8339605971527198	13574/16550	1838
lines-02(4)-11-42(15,2)-	0.8227622658853269	0.8322235294331407	14321/17406	1924
lines-41(15,2)-	0.8354911575119828	0.8427484542994089	15165/18151	1994
lines-02(4)-11-21(3,25,11)-41(11,10)-	0.8438489371325192	0.8483386876131003	14926/17688	1951
lines-02(4)-11-42(9,4)-	0.8528512944703915	0.8599987763097161	14791/17343	1926
lines-02(4)-11-21(3,25,11)-32(7,60,60)-41(11,10)-	0.8623951354090423	0.8675884325580375	15317/17761	1958
lines-02(4)-11-21(3,25,11)-42(9,4)-	0.8699809864668382	0.8762512912142078	15557/17882	1968
lines-02(4)-11-21(3,25,11)-34(9)-41(11,10)-	0.8772795045303361	0.8790234334119904	15298/17438	1926
lines-02(4)-11-21(3,25,11)-31(7,7)-41(11,10)-	0.8783791410350471	0.8817015492485121	15564/17719	1954
lines-02(4)-11-21(3,25,11)-32(5,90,90)-42(9,4)-	0.8793868268852827	0.8865269417065524	15661/17809	1964
lines-02(4)-11-21(3,25,11)-34(11)-42(9,4)-	0.8837408592321755	0.8894551473365204	15469/17504	1932

lines-02(4)-11-21(3,25,11)-33(9,9,9)-41(11,10)-	0.8925952874013544	0.8967686937728782	15948/17867	1969
lines-42(15,2)-	0.904743687834736	0.9069319997088674	16555/18298	2007
lines-02(4)-11-41(15,2)-	0.9150397686189443	0.9220258005624874	15186/16596	1852
lines-02(4)-11-21(3,25,11)-31(9,9)-42(9,4)-	0.9364262129945716	0.9403473956954022	16733/17869	1969
lines-02(4)-11-21(3,25,11)-33(11,11,11)-42(9,4)-	0.9627017795162801	0.9648235353719851	17474/18151	1996
lines-02(4)-11-21(3,25,11)-42(11,10)-	0.9704896411496401	0.9738608435485511	17660/18197	1999
lines-02(4)-11-42(11,10)-	0.9705035971223022	0.97352178790181	17537/18070	1987
lines-02(4)-11-21(3,25,11)-32(7,60,60)-42(11,10)-	0.9792478726324458	0.9818922121796275	17837/18215	1999
lines-02(4)-11-21(3,25,11)-34(9)-42(11,10)-	0.9838326987805551	0.9849197315776745	17830/18123	1993
lines-02(4)-11-21(3,25,11)-31(7,7)-42(11,10)-	0.9929101221640488	0.9939259099360986	18206/18336	2012
lines-02(4)-11-21(3,25,11)-33(9,9,9)-42(11,10)-	0.9986387890667537	0.9991203584942153	18341/18366	2014
lines-02(4)-11-21(3,25,11)-33(9,9,9)-42(11,10)-	0.9975524855868596	0.9991203584942153	18341/18386	2014

lines-02(4)-11-21(3,25,11)-33(9,9,9)-42(11,10)-	0.9975524855868596	0.9991203584942153	18341/18386	2014

Tabulka A.1: Tabulka s naměřenými hodnotami