Západočeská univerzita v Plzni

Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Rozpoznávání textu v komixech

Plzeň 2021 Lukáš Vlček

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš VLČEK**
Osobní číslo:	**A18B0351P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Rozpoznávání textu v komixech**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte metody pro segmentaci a analýzu obrazu a seznamte se s dostupnými systémy optického rozpoznávání znaků (OCR). Dále se seznamte s formáty datasetů využívaných výzkumnou skupinou NLP.
2. Navrhněte postup pro detekci bublin v komixových obrázcích.
3. Navržený postup implementujte a integrujte všechny části do funkčního celku.
4. Vytvořenou aplikací rozpoznejte text na vybrané reprezentativní množině komixových obrázků. Ve vhodném formátu vytvořte z rozpoznaných textů dataset, který bude dále využíván pro výzkumné účely skupiny NLP.
5. Vytvořený systém otestujte a kriticky zhodnoťte dosažené výsledky.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Jiří Martínek**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **5. října 2020**
Termín odevzdání bakalářské práce: **6. května 2021**

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 26. října 2020

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 6. května 2021

Lukáš Vlček

Poděkování

Rád bych poděkoval Ing. Jiřímu Martínkovi za odborné vedení, cenné rady a věcné připomínky při zpracování této práce.

Lukáš Vlček

Abstract

This thesis deals with the creation of a program that allows the automatic creation of a dataset containing baloon texts (comic speech bubbles) with recognized text. The comics pages will be first segmented using computer vision methods complemented by filtering of the segmented regions. Detected bubbles are input to optical character recognition (OCR) method. Moreover, the comic bubble information is aggregated with the recognized text and saved in a file for further processing. The experiments that measured the success rate of the designed algorithm were conducted.

Abstrakt

Tato bakalářská práce se zabývá vytvořením programu, který umožňuje automatické vytvoření datasetu obsahujícího komixové bubliny s rozpoznaným textem. Stránky komixů budou nejprve vhodným způsobem segmentovány za využití metod počítačového vidění doplněných o následnou filtraci segmentovaných oblastí. Detekované bubliny jsou posléze vstupem do metody optického rozpoznávání znaků (OCR). Následně jsou informace o komixové bublině doplněny o rozpoznaný text a ve vhodném formátu uloženy do souboru pro další zpracování. V rámci této práce byly provedeny experimenty, které změřily úspěšnost navrženého algoritmu.

Obsah

1 Úvod

Komixy jsou příkladem dokumentů na rozhraní obrázku a textu. Děj v komixech je znázorněn v obrázkové podobě doplněné o textovou informaci – dialogy, vstupy vypravěče či upřesnění místa a času. Výzkumná skupina NLP vyvíjí modely klasifikace dialogů ve formátu textu. Obvyklý zdroj těchto dialogů pochází z přepisů zvukových signálů (např. telefonické hovory nebo nahrávky rozhovorů), nicméně dialogy existují i v tištěné podobě – v divadelních hrách a komixech. Zpřístupněním textu obsaženého v komixech dojde k vytvoření datasetu dialogů.

Cílem bakalářské práce je vytvoření programu, který umožní automatické vytvoření datasetu obsahujícího komixové bubliny s rozpoznaným textem. Stránky komixů budou nejprve vhodným způsobem segmentovány za využití metod počítačového vidění (*computer vision*) doplněných o následnou filtraci segmentovaných oblastí. Detekované bubliny jsou posléze vstupem pro systém optického rozpoznání znaků (OCR). Následně jsou informace o komixové bublině doplněny o rozpoznaný text a ve vhodném formátu uloženy do souboru pro další zpracování.

Kromě vytvoření datasetu pro využití výzkumnou skupinou NLP je možné získaná data využít k automatickému překladu textu komixů (přeložený text je možné díky znalosti velikosti a tvaru bubliny vhodně umístit zpět do komixu). Dále by bylo možné vytvořit databázi komixů s možností prohledávání podle textů v nich obsažených nebo dataset využít jako podklad pro automatické čtení komixů (např. zpřístupnění komixů i dětské populaci).

Bakalářská práce je rozdělena na teoretickou a praktickou část. V teoretické části budou nejprve představeny základní metody analýzy a zpracování obrazu, následně bude nastíněn princip fungování OCR včetně uvedení některých dostupných systémů. Praktická část popisuje navržený algoritmus detekce komixových bublin a upřesňuje způsob předzpracování, který vede k dosažení optimálních výsledků pro rozpoznání textu. Na závěr práce jsou uvedeny výsledky provedených experimentů a detailněji popsány možnosti potenciální praktické aplikace získaného datasetu.

2 Analýza a zpracování obrazu

V této kapitole budou nejprve popsány základní způsoby reprezentace digitálních obrázků v počítači, a následně budou představeny hlavní algoritmy a techniky používané v oblasti počítačového vidění (*computer vision*).

2.1 Reprezentace obrázku v počítači

Digitální obrázek je v počítači reprezentován dvourozměrným polem bodů, ve kterém je každý bod označen jako *pixel* (z anglického *picture element*) [17]. Každý obrázek má stanovený počet barevných kanálů. 1 barevný kanál obvykle značí, že je obrázek šedotónový, 3 kanály obvykle zastupují 3 základní barvy (R, G, B) a 4 kanály k základním barvám přidávají kanál průhlednosti. Mezi další důležité vlastnosti obrázku patří barevná hloubka, která určuje počet bitů každého barevného kanálu pixelu, a tedy i počet hodnot, kterých může každý pixel v daném barevném kanálu nabývat. Nejčastěji se používá barevná hloubka 8 bitů, tedy hodnota 0 až 255, v oblasti analýzy a zpracování obrazu hraje také důležitou roli obrázek s barevnou hloubkou 1 bit, který umožňuje každému pixelu nabývat pouze hodnoty 0 nebo 1.

Pixel uchovává informaci o intenzitě každého svého barevného kanálu. V šedotónovém obrázku s jedním barevným kanálem je hodnota pixelu určena jednou intenzitou, v případě barevného obrázku se třemi barevnými kanály určuje hodnotu pixelu trojice intenzit (R, G, B), označována jako barva.

2.2 Segmentace

Segmentace je proces rozdělení obrázku na menší oblasti, který je obvykle prováděn na základě společných vlastností (pixely uvnitř stejného segmentu mají například stejnou či podobnou barvu nebo jsou součástí stejného objektu). Jinými slovy je proces segmentace také možné popsat jako postup, při kterém je každému pixelu v obrázku přiřazen index segmentu, ve kterém se nachází. Takto získané segmenty je možné následně snáze analyzovat [34].

2.3 Prahování

Při segmentaci se často setkáme s potřebou oddělit pixely v popředí a pozadí obrázku. K tomuto účelu nám může pomoci prahování (anglicky *thresholding* nebo také *binarization*), jehož cílem je přiřadit každému pixelu vstupního obrázku jednu ze 2 hodnot, které můžeme označit jako b_0 a b_1 [31]. Ačkoliv se nejčastěji setkáme s prahováním u šedotónových obrázků (*grayscale*), techniku je možné zobecnit i na obrázky barevné [16].

Formálně můžeme prahování zapsat matematickou funkcí 2.1, kde f_t vyjadřuje funkci prahování, x, y vyjadřují souřadnice pixelu v obrázku, funkce f reprezentuje intenzitu daného pixelu v obrázku a $t_{x,y}$ značí prahovou hodnotu pro daný pixel [31].

$$f_t(x,y) = \begin{cases} b_0, & \text{kde } f(x,y) < t_{x,y}. \\ b_1, & \text{kde } f(x,y) \geq t_{x,y}. \end{cases} \tag{2.1}$$

Podle způsobu určení hodnoty $t_{x,y}$ můžeme prahování rozdělit na globální a lokální:

- v případě **globálního prahování** je hodnota $t_{x,y}$ nezávislá na x, y, tedy je pro celý obrázek stejná,
- u **lokálního prahování** dochází k rozdělení celého obrázku na menší části a určení hodnoty prahu pro každou část zvlášť.

2.3.1 Globální prahování

Globální prahování je vhodné použít v momentě, kdy je dostatečný rozdíl mezi intenzitou pixelů v pozadí a popředí. V takovém případě můžeme určením vhodné hodnoty prahu t popředí od pozadí oddělit [22]. Hodnota t je v tomto případě platná pro celý obrázek a není tak závislá na pozici prahovaného pixelu (x, y).

Manuální určení hodnoty prahu

Manuální určení je nejjednodušší metodou prahování, kdy dojde ke stanovení globální prahové hodnoty uživatelem či programátorem na základě odhadu či znalosti specifických vlastností obrázku.

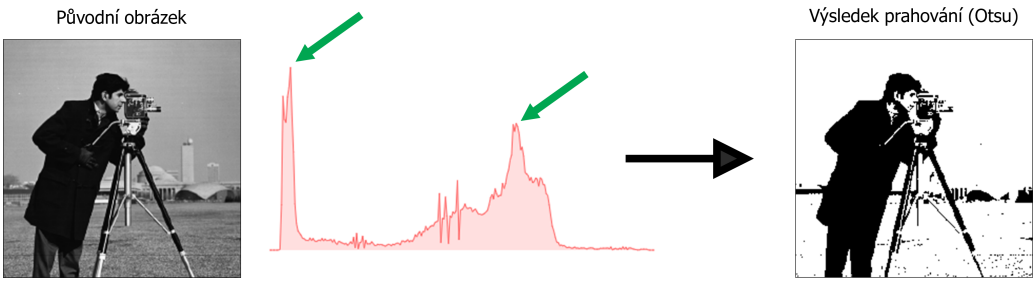
```python
import cv2

# nacteni vstupniho obrazku
img = cv2.imread('grayscale_image.png')
t = 127             # urcena hodnota prahu
max_value = 255     # maximalni hodnota intenzity pixelu

# prahovani
threshold, binarized_image = cv2.threshold(img, t, max_value, cv2.
    THRESH_BINARY)

# ulozeni binarizovaneho obrazku
cv2.imwrite(binarized_image, 'binarized_image.png')
```

Ukázka kódu 2.1: OpenCV prahování s určenou hodnotou [28]

Úspěšnost provedení globálního prahování závisí na vhodně zvolené hodnotě prahu. Existuje mnoho algoritmů pro určení hodnoty prahu t, jedním z nejznámějších a nejpoužívanějších je algoritmus založený na analýze histogramu, který představil Nobuyuki Otsu roku 1979 [30].

Otsu

Otsův algoritmus předpokládá, že je obrázek možné rozdělit na dvě části - pozadí a popředí. Nejlepších výsledků pak dosahuje pro obrázky, ve kterých se v histogramu sestaveného z intenzit jednotlivých pixelů nachází 2 zřetelné vrcholy (tzv. bimodální histogram). Na obrázku 2.1 jsou tyto 2 vrcholy znázorněny zelenou šipkou.



Obrázek 2.1: Ukázka Otsu prahování. Vlevo je vstupní obrázek a histogram intenzit pixelů ve vstupním obrázku, vpravo je obrázek po aplikaci prahování založeného na Otsu algoritmu.

Algoritmus, který představil Nobuyuki Otsu [30] pro každou přípustnou hodnotu prahu t (tedy $0 - I$, kde I vyjadřuje maximální přípustnou hodnotu intenzity pixelu v obrázku, pro obrázek s barevnou hloubkou 8 bitů I = 255)

rozdělí pixely obrázku na 2 třídy – pozadí a popředí. Následně spočítá hodnotu rozptylu intenzit pixelů zařazených do třídy pozadí (σ_b^2) a popředí (σ_f^2) a určí výslednou hodnotu rozptylu jako vážený součet rozptylů pixelů uvnitř jedné třídy (σ_w^2). Výslednou hodnotou prahu t se prohlásí prohlásí takové t, pro které je hodnota (σ_w^2) nejmenší.

Abychom mohli formálně vyjádřit vztahy pro určení optimální hodnoty prahu, musíme nejprve definovat pravděpodobnost intenzity v obrázku jako

$$P(i) = \frac{\textit{počet pixelů s intenzitou } i}{\textit{počet všech pixelů v obrázku}} \tag{2.2}$$

Pravděpodobnost třídy (W_b = background, W_f = foreground) v závislosti na t, kde I vyjadřuje maximální možnou intenzitu pixelu v obrázku, definujeme jako

$$W_b(t) = \sum_{i=0}^{t-1} P(i) \qquad W_f(t) = \sum_{i=t}^{I} P(i) \tag{2.3}$$

Následně můžeme vyjádřit střední hodnotu (μ)

$$\mu_b(t) = \sum_{i=0}^{t-1} \frac{iP(i)}{W_b(t)} \qquad \mu_f(t) = \sum_{i=t}^{I} \frac{iP(i)}{W_f(t)} \tag{2.4}$$

a rozptyl intenzit pixelů uvnitř jedné třídy třídy (σ^2) poté vyjádříme jako

$$\sigma_b^2(t) = \frac{\sum_{i=0}^{t-1}(i - \mu_b(t))^2 \cdot P(i)}{W_b(t)} \qquad \sigma_f^2(t) = \frac{\sum_{i=t}^{I}(i - \mu_f(t))^2 \cdot P(i)}{W_f(t)} \tag{2.5}$$

Výsledný vážený součet rozptylů získáme jako

$$\sigma_w^2(t) = W_b(t) \cdot \sigma_b^2(t) + W_f(t) \cdot \sigma_f^2(t) \tag{2.6}$$

Za optimální hodnotu prahu (t^*) dle Otsova algoritmu prohlásíme takové t, pro které je hodnota váženého součtu rozptylů tříd $\sigma_w^2(t)$ nejmenší (vzorec 2.7).

$$\sigma_w^2(t^*) = \min_{0 \le t \le I} \sigma_w^2(t) \tag{2.7}$$

```python
import cv2

# nacteni vstupniho obrazku
img = cv2.imread('grayscale_image.png')
max_value = 255     # maximalni hodnota intenzity pixelu

# prahovani
threshold, binarized_image_otsu = cv2.threshold(img, 0, max_value, cv2.
    THRESH_BINARY + cv2.THRESH_OTSU)

# ulozeni binarizovaneho obrazku
cv2.imwrite(binarized_image, 'binarized_image_otsu.png')
```

Ukázka kódu 2.2: OpenCV prahování za využití Otsuova algoritmu [28]

2.3.2 Lokální prahování

Algoritmy globálního prahování poskytují rychlé výsledky a jsou vhodnou volbou pro obrázky, kde je úroveň nasvícení rovnoměrná. U nerovnoměrně osvícených obrázků (například pokud je v části obrázku stín) často dochází k jevu, kdy je celá tmavší část obrázku zařazena do třídy pozadí (viz obrázek 2.2), což se může negativně projevit na následném zpracování obrázku [36].

Algoritmy lokálního prahování určují hodnotu prahu pro každý pixel na základě intenzity pixelů v jeho okolí. Nevýhodou lokálního prahování je větší výpočetní složitost a nutnost určení dalších parametrů (například velikost okolí, které se bude pro výpočet prahu posuzovat), které výsledek mohou výrazně ovlivnit [36]. Mezi známé algoritmy lokálního prahování patří například **Niblackův algoritmus** [23], který následně rozšířil **Sauvola** [32].

Obrázek 2.2: Ukázka adaptivního prahování. Vlevo původní obrázek, uprostřed prahování pomocí Otsu metody, vpravo adaptivní prahování (velikost bloku 7 pixelů, před prahováním aplikováno mediánové rozostření) [18].

2.4 Morfologické operace

Morfologie, z řeckého morfé = tvar a logia = nauka, se zabývá studiem tvarů [7]. Matematická morfologie, ze které algoritmy zpracování obrazu vycházejí, poskytuje řadu metod, které umožňují zjednodušovat obrazová data, eliminovat šum a zároveň zachovat jejich hlavní tvarové charakteristiky. Morfologické operace se často používají v úvodní fázi zpracování obrazu v oblasti počítačového vidění, kdy je klíčové odstranit nedokonalosti vstupního obrazu, jako například šum [15].

Podrobně se popisu morfologických operací z matematického pohledu věnují například Rafael C. Gonzalez a Richard E. Woords ve své knize Digital Image Processing [11].

V následující části budou představeny 2 základní morfologické operace – dilatace a eroze a jejich použití v metodách *opening* a *closing*. Pro účely této práce uvažujeme morfologické operace pouze nad binárními (tj. černobílými) obrázky.

2.4.1 Dilatace

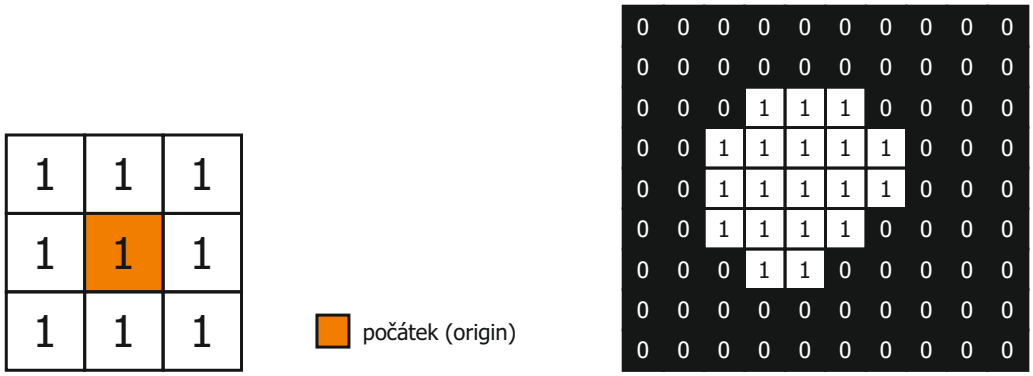
Dilatace je základní morfologickou operací, která se často používá při zpracování obrazu. Vstupem je kromě binárního obrázku také strukturní element či jádro (v angličtině se používá označení *structuring element* nebo také *kernel*), ve kterém je určen tzv. *origin* (počátek).

Strukturní element je obvykle (v porovnání s binárním obrázkem) malá množina dat, která zásadním způsobem ovlivňuje výsledek algoritmu [11]. Pravděpodobně nejčastěji se jako strukturní element používá čtverec o velikosti 3×3 pixely, nicméně v některých případech může být použit i mnohem větší strukturní element za účelem dosažení extrémní dilatace. Větší strukturní elementy obvykle nemají tvar čtverce, jako je tomu u menších, ale zaujímají tvar disku. Výrazné dilatace je ale možné dosáhnout i více iteracemi, což je v praxi častější volba [10].

Dilatace přidává pixely po obvodu každého objektu v binárním obrázku, čímž objekty zvětšuje. Dilatací také dojde ke spojení oddělených objektů na obrázku, pokud jsou od sebe vzdáleny méně než je velikost strukturního elementu (jádra) a vyplnění drobných mezer [11].

Ukázka algoritmu

Jako strukturní element použijeme čtvercovou matici o rozměrech 3×3, jako binární obrázek pro účely snazší demonstrace algoritmu použijeme obrazec 10×10 (viz obrázek 2.3).

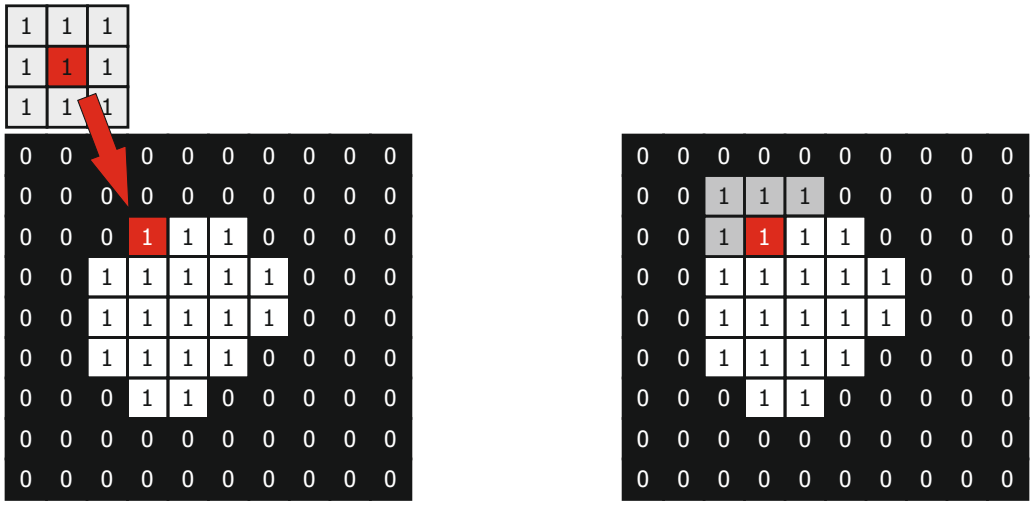


(a) (b)

Obrázek 2.3: (a) čtvercový strukturní element o velikosti 3×3 použitý pro dilataci, (b) binární obrázek, nad kterým se bude provádět dilatace

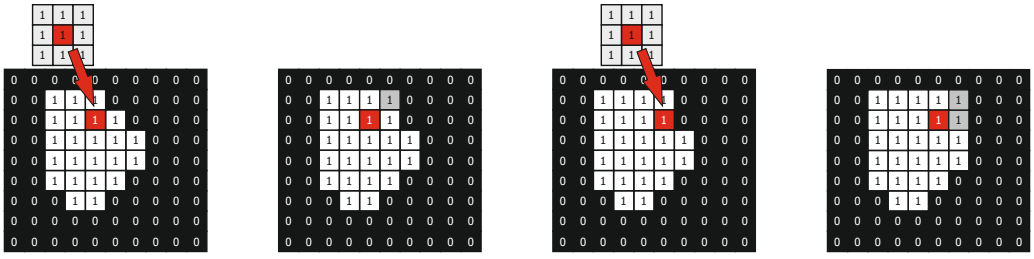
Upravíme souřadnice strukturního elementu tak, aby souřadnice počátku (*origin*) odpovídaly souřadnicím prvního bílého bodu binárního obrázku. Úprava souřadnic (neboli posunutí) je naznačena na obrázku 2.4a. Do výsledného obrázku následně přidáme všechny body, které náleží posunutému strukturnímu elementu (tento krok je možné si představit jako přidání všech bodů, které leží pod strukturním elementem). Nově přidané body jsou na obrázku 2.4b označeny šedou barvou. Tento postup je následně zopakován pro všechny body vstupního obrázku (další postup je naznačen na obrázku 2.5) [10, 11].

Pokud již nezbývá žádný nezpracovaný bod v množině A, algoritmus končí. Výsledný stav pro tento vzorový příklad je znázorněn na obrázku 2.6



(a) (b)

Obrázek 2.4: První krok algoritmu dilatace. Obrázek (a) znázorňuje posunutí strukturního elementu na pozici prvního bílého bodu, na obrázku (b) jsou šedou barvou znázorněny body, které budou do výsledného obrázku nově přidány.



(a) (b) (c) (d)

Obrázek 2.5: Další kroky algoritmu dilatace. Na obrázcích (a) a (c) je znázorněno posunutí strukturního elementu, na obrázcích (b) a (d) jsou šedě zvýrazněny pixely, které budou v tomto kroku přidány do výsledného obrázku

0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Obrázek 2.6: Výsledek dilatace

Dilatace na obrázku

Na obrázku 2.7 je znázorněn výsledek dilatace. Strukturním elementem byla, stejně jako v předchozí ukázce, čtvercová matice 3×3.

(a) Původní obrázek (b) Výsledek dilatace

Obrázek 2.7: Ukázka dilatace na obrázku.

Dilatace v OpenCV

```
import cv2
import numpy as np  # numpy pouzivan pro generovani matice

# nacteni vstupniho obrazku
img = cv2.imread('grayscale_image.png', 0)

# inicializace jadra o velikosti 3x3
kernel = np.ones((3,3), np.uint8)

dilated_image = cv2.dilate(img, kernel)

# ulozeni vysledku dilatace
cv2.imwrite('dilated_image.png', dilated_image)
```

Ukázka kódu 2.3: OpenCV dilatace [27]

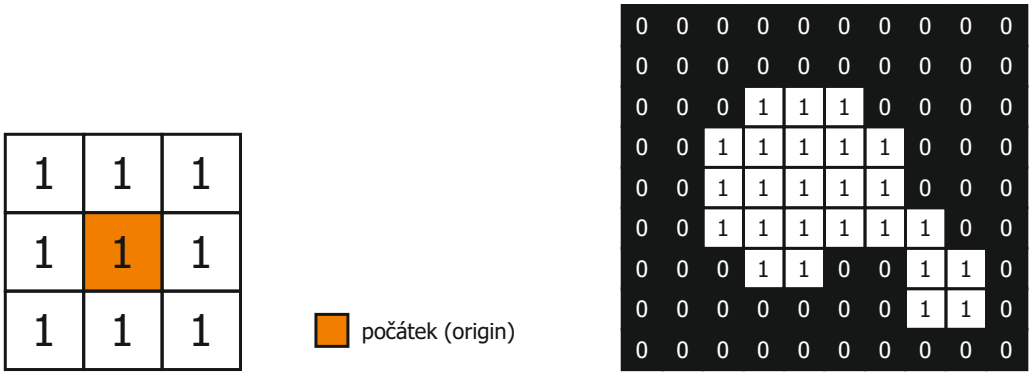
2.4.2 Eroze

Eroze je druhou ze základních morfologických operací používaných při zpracování obrazu. Stejně jako v případě dilatace je vstupem dvojice binárního obrázku a strukturního elementu (jádra) s vyznačeným počátkem (*origin*). Na rozdíl od dilatace ale eroze odebíráním pixelů po obvodu zmenšuje objekty, které se nacházejí v binárním obrázku. Této vlastnosti se využívá při odstraňování nežádoucího bílého šumu či rozdělování úzce spojených objektů [11].

Ukázka algoritmu

Pro účely demonstrace fungování algoritmu eroze použijeme stejný strukturní element jako v případě dilatace, ale binární obrázek reprezentovaný obrazcem velikosti 10×10 lehce upravíme (viz obrázek 2.8).

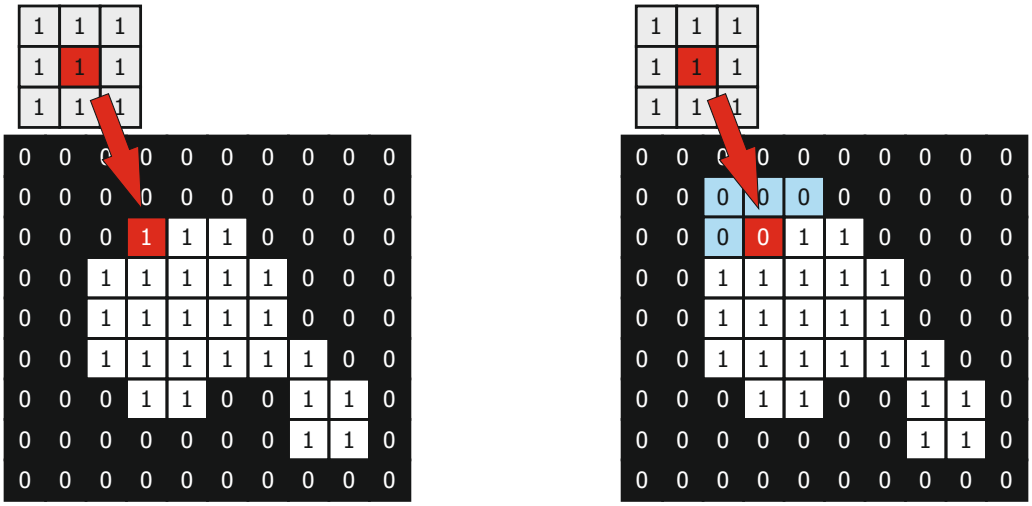
Při erozi postupujeme podobným způsobem, jako v případě dilatace. Opět začneme prvním bílým bodem a posuneme strukturní element tak, aby jeho počátek (*origin*) odpovídal souřadnicím prvního bodu. V dalším kroku ale místo přidávání dalších bodů do výsledného obrázku rozhodujeme, zda z něj odebrat právě zpracovávaný bod. Pokud jsou všechny body náležící posunutému strukturnímu elementu vstupního binárního obrázku bílé, bod zůstane zachován, v opačném případě se odebere. Stejný postup aplikujeme na všechny body vstupního obrázku (kontrolu, zda všechny body ležící pod strukturním elementem, provádíme vždy na původním obrázku, tedy výsle-



(a) (b)

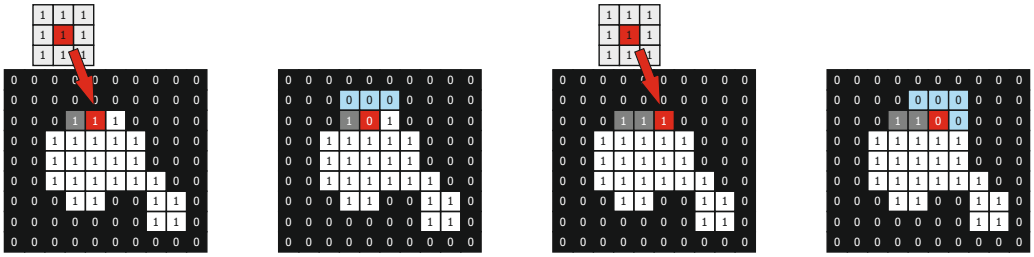
Obrázek 2.8: (a) čtvercový strukturní element o velikosti 3x3 použitý pro erozi, (b) binární obrázek, nad kterým se bude eroze provádět

dek kontroly druhého pixelu není závislý na kontrole prvního pixelu) [10]. Fungování algoritmu je znázorněno na obrázcích 2.9 a 2.10, výsledek eroze se nachází na obrázku 2.11.



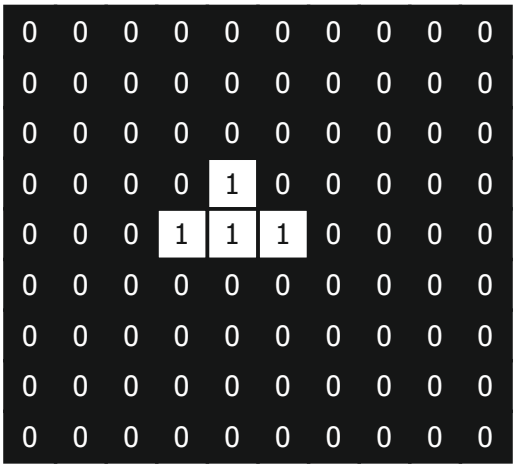
(a) (b)

Obrázek 2.9: První krok algoritmu eroze. Obrázek (a) znázorňuje posunutí strukturního elementu na pozici prvního bílého bodu. Obrázek (b) zobrazuje situaci po vyhodnocení bodu, kde světle modrou barvou jsou znázorněny body, které jsou obsaženy v posunutém strukturním elementu, ale nejsou obsaženy v binárním obrázku.



(a) (b) (c) (d)

Obrázek 2.10: Další kroky algoritmu eroze. Na obrázcích (a) a (c) je naznačeno posunutí strukturního elementu, na obrázcích (b) a (d) jsou znázorněny stavy po vyhodnocení bodu, kdy jsou modře obarveny body, které způsobily vyřazení bodu



Obrázek 2.11: Výsledek eroze

Eroze na obrázku

Na obrázku 2.12 je znázorněn výsledek eroze. Strukturním elementem byla, stejně jako v předchozí ukázce, čtvercová matice 3×3 a byla provedena 1 iterace.

(a) Původní obrázek (b) Výsledek eroze

Obrázek 2.12: Ukázka eroze na obrázku.

Eroze v OpenCV

OpenCV předpokládá, že popředí, na kterém je prováděna eroze, má bílou barvu (hodnotu 255) a pozadí obrázku barvu černou (hodnota 0).

```
import cv2
import numpy as np  # numpy pouzivan pro generovani matice

# nacteni vstupniho obrazku
img = cv2.imread('grayscale_image.png', 0)

# inicializace jadra o velikosti 3x3
kernel = np.ones((3,3), np.uint8)

erosion_image = cv2.erode(img, kernel)

# ulozeni vysledku eroze
cv2.imwrite('erosion_image.png', erosion_image)
```

Ukázka kódu 2.4: OpenCV eroze [27]

2.4.3 Opening

Operace opening (do češtiny někdy překládaná jako otevírání) je definována jako eroze obrázku následována dilatací, kdy pro erozi i dilataci je použit stejný strukturní element. Eroze nejprve odebere objekty, které jsou menší než strukturní element, a následná dilatace zajistí zvětšení zmenšených objektů na velikost, která přibližně odpovídá původní. Operace opening se obvykle používá pro vyhlazení objektu, eliminaci úzkých výčnělků a k odstranění nežádoucího šumu [11].

2.4.4 Closing

Closing (česky zavírání) je morfologická operace, která je definována jako dilatace obrázku následována erozí s použitím stejného strukturního elementu pro dilataci i erozi (jedná se tedy o opačné pořadí základních morfologických operací, než v případě operace opening). Dilatace, která se provede jako první, vyplní drobné díry v objektu, ale zároveň rozšíří objekty přidáním pixelů po jejich obvodu. Tento často nežádoucí jev kompenzuje následná eroze, která zajistí, že se velikost objektu příliš nezmění. Hlavní využití operace closing tedy nalezne při vyplňování drobných děr v objektech a při eliminaci šumu v obrázku [10, 11].

2.5 Filtrace obrazu

Filtrování obrazu zahrnuje širokou škálu algoritmů, které ve fázi předzpracování zajišťují vyhlazení, rozmazání či doostření obrázku nebo také zvýraznění hran [39].

Algoritmy filtrování obrazu můžeme podle způsobu jejich fungování rozdělit do dvou kategorií na lineární a nelineární. Lineární filtry produkují výstup, který je lineární kombinací vstupu, zatímco nelineární filtry určují výstup na základě nelineárního operátoru (kterým může být například medián hodnot v okolí pixelu) [39].

2.5.1 Lineární obrazové filtry

Konvoluční filtry

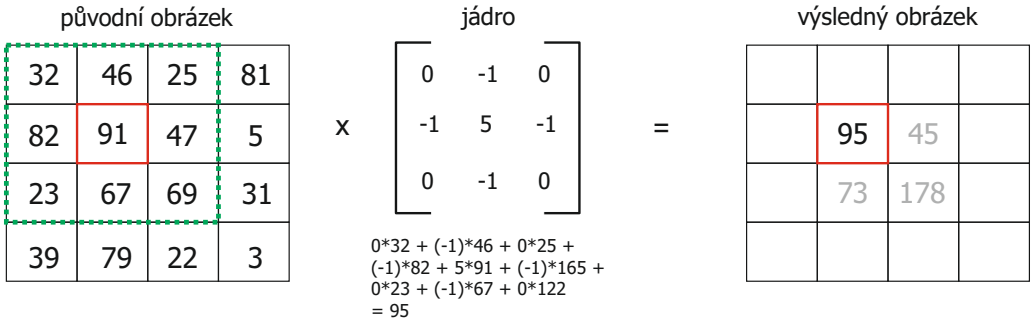
Konvoluční filtry se nejčastěji používají k rozmazání či zostření obrázku nebo zvýraznění hran. Výsledek konvoluce je určen maskou (někdy nazývanou jako jádro, anglicky mask nebo kernel), která definuje koeficienty konvoluce. Výsledná hodnota pixelu se určí jako suma hodnot pixelů v okolí vynásobených příslušným koeficientem určeným maskou. Maska může nabývat různých velikostí a nemusí mít pouze čtvercový tvar, často se setkáme také s obdélníkovou či kruhovou maskou [39].

Formálně můžeme konvoluční filtrování zapsat následujícím způsobem:

$$g(x,y) = \sum_{k,l} f(x+k, y+l) \cdot h(k,l) \tag{2.8}$$

kde g je výsledný obrázek, f původní obrázek, x,y označují souřadnice pixelu a h je jádro (kernel).

Na obrázku 2.13 je znázorněno fungování konvolučního filtru na několika bodech. Komplikace nastává při určování hodnot pixelů po obvodu obrázku, kde není možné hodnotu určit pomocí koeficientů v jádře. Tyto pixely nejsou metodou konvolučního filtrování jednoznačně definovány a jejich hodnotu je možno určit různými technikami, například nastavením takovýchto pixelů na konstantní hodnotu (nejčastěji 0) nebo replikací sousedních pixelů [39].



Obrázek 2.13: Ukázka konvoluce

Průměrovací filtr

Průměrovací filtr (anglicky *mean filter*, *smoothing* nebo *box filter*) nahrazuje hodnotu pixelu průměrem hodnot sousedních pixelů (včetně zpracovávaného pixelu), můžeme jej tedy zapsat jako konvoluční filtr s jádrem h [39]:

$$h = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \tag{2.9}$$

Tento filtr eliminuje extrémní hodnoty v obrázku, a proto se používá pro vyhlazení hran a redukci šumu v obrázku [9]. Velikost jádra může být libovolná, nejčastěji se ale volí velikost 3×3 či 5×5. Pro zvětšení efektu filtru je možné použít větší jádro, tj. průměrovat větší oblast obrázku, nebo filtraci provádět ve více iteracích [9, 39].

Sobelův operátor

Sobelův operátor je používán k aproximaci gradientu v obrázku (tj. změny intenzity pixelů v nějakém směru). Oblasti s jednotnou intenzitou budou mít malý nebo nulový gradient a přechody či hrany budou naopak mít gradient velký. Gradient proto nachází uplatnění v algoritmech detekce hran, a je tak typickým příkladem lineárního filtrování, které je použitého ve fázi předzpracování obrazu [11].

Operátor používá dvojici konvolučních masek o velikosti 3×3, kdy druhá je vůči první orotována o 90°.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 3 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.10}$$

Každá z těchto masek je navržena tak, aby maximálně reagovala na hrany vedoucí vertikálně (G_x) a horizontálně (G_y). Použitím konvolučního filtru na původním obrázku za použití příslušné konvoluční masky získáme aproximaci gradientu ve vertikálním a horizontálním směru (označme je jako G_x a G_y). G_x a G_y jsou složky vektoru odhadu gradientu G a velikost gradientu tedy odpovídá normě tohoto vektoru a můžeme ji vyjádřit podle vzorce 2.11 [8].

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.11}$$

V některých případech si můžeme vystačit s nepřesným, ale z hlediska výpočetní náročnosti výrazně jednodušším vyjádřením

$$|G| = |G_x| + |G_y| \tag{2.12}$$

Směr gradientu (úhel) můžeme vyjádřit jako θ

$$\theta = arctan\left(\frac{g_y}{g_x}\right) \tag{2.13}$$

2.5.2 Nelineární obrazové filtry

Zatímco lineární obrazové filtry určovaly výslednou hodnotu pixelu pomocí plovoucího okénka a lineární kombinace pixelů v okolí, nelineární obrazové filtry k tomuto účelu často využívají řazení pixelů v okolí.

Mediánový filtr

Mediánový filtr funguje podobným způsobem jako filtr průměrovací (viz lineární filtry), místo jednoduchého nahrazení pixelu průměrnou hodnotou jeho okolí ale provede nahrazení pixelu mediánem hodnot okolí. Pro každý pixel se nejprve všechny hodnoty v zadaném okolí seřadí (včetně hodnoty pixelu samotného) a jako nová hodnota pixelu se prohlásí prostřední z nich. V případě, že je počet pixelů v okolí sudý, nahradí se hodnota průměrem dvou prostředních hodnot [9].

Stejně jako průměrovací filtr je i mediánový filtr používán k redukci šumu (dobře funguje na šumy typu „salt and pepper“, gaussův šum a náhodný šum [44]). Oproti průměrovacímu filtru ale lépe zachovává hrany, a tak v některých aplikacích poskytuje lepší výsledky. Jeho zásadní nevýhodou je ale výrazně vyšší výpočetní náročnost, než u lineárních filtrů [9].

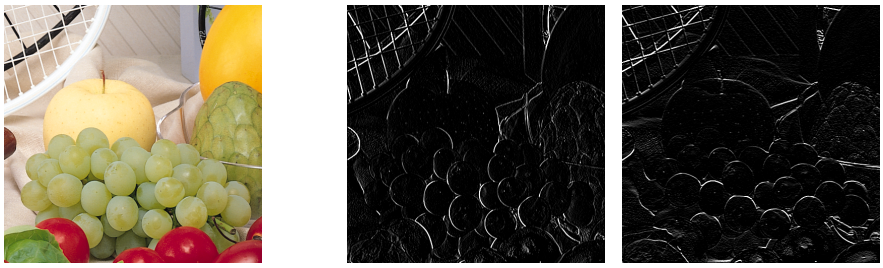
2.6 Detekce hran

Detekce hran v obrazu je proces, při kterém dochází k identifikaci hran, tedy ostrých lokálních změn intenzity pixelů [11, 41]. Definujme pixely, ve kterých dochází k náhlé změně intenzity, jako hraniční pixely (*edge pixel*) a hrany (*edge*) pak jako množiny sousedících hraničních pixelů. Většina algoritmů detekce hran v obrázku se soustředí na detekci hraničních pixelů [11]. Nalezení hraničních pixelů a hran je pro analýzu obrazu kriticky důležité, neboť tyto pixely nesou důležitou sémantickou informaci – ve většině případů je možné pomocí detekce hran určit hranice objektů [39].

Podrobnému matematickému popisu detekce hran se věnují například R. Gonzalez a R. Woods ve své knize Digital Image Processing [11].

2.6.1 Sobelův operátor

Tento operátor již byl představen v sekci věnované lineárním obrazovým filtrům. Jak bylo zmíněno, využívá dvě konvoluční jádra k určení gradientu v obrázku (jedno jádro pro vertikální, druhé pro horizontální gradient) a je možné ho díky tomu využít k detekci hran (například jako součást Cannyho detektoru hran, který bude popsán v následující sekci). Na obrázku 2.14 je znázorněn výstup Sobelova operátoru - vyjádření gradientu v horizontálním a vertikálním směru. Pokud z vertikální a horizontální složky gradientu vy-



(a) Původní obrázek (b) Sobel - gradient X (c) Sobel - gradient Y

Obrázek 2.14: Ukázka výstupu Sobelova operátoru

počítáme celkovou hodnotu gradientu v bodě (pomocí vzorce 2.11 uvedeného výše), získáme výsledek uvedený v obrázku 2.15 [29].

(a) Původní obrázek (b) Celkový gradient za použití Sobelova operátoru

Obrázek 2.15: Ukázka výsledného gradientu po aplikování Sobelova operátoru

2.6.2 Cannyho hranový detektor

John F. Canny představil roku 1986 detektor hran [3], který se s drobnými modifikacemi používá dodnes. Canny formuloval 3 základní požadavky, které by měl algoritmus detekující hrany splňovat.

1. **Malá chybovost** – algoritmus by měl odhalit všechny hrany a zároveň neoznačit za hranu nic, co hranou není.
2. **Přesnost** – poloha hran musí být určena co nejblíže skutečné hraně.

3. **Jednoznačnost** – algoritmus by měl každou skutečnou hranu detekovat pouze jednou, neměl by vytvářet zdvojené hrany.

Následně se snažil nalézt optimální řešení, které tyto požadavky splňuje [11].

Cannyho hranový detektor k dosažení svého cíle postupuje přes několik dílčích kroků:

1. **Vyhlazení obrázku a eliminace šumu** pomocí Gaussova filtru. Detekce hran je citlivá na šum v obrázku a algoritmus by v případě jeho neodstranění mohl poskytovat špatné výsledky [11].

2. **Určení gradientu** pro všechny body obrázku. Pro každý bod obrázku je určena hodnota gradientu ($|G|$, viz 2.11) a jeho směr (θ, viz 2.13). K tomuto účelu se používá například Robertsův či Sobelův operátor [11].

3. **Nalezení lokálních maxim (potlačení nemaximálních hodnot)**. V tomto kroku dochází k odstranění pixelů, u kterých je hodnota gradientu vysoká, ale nedosahuje maxima ve svém okolí. Díky této operaci jsou detekované hrany tenké [3, 11, 24].

4. **Aplikování dvojitého prahování a následná analýza spojenosti** k určení finálních hran. Na základě zadaných prahů gradientu (g_{min} a g_{max}) jsou pixely, jejichž hodnota gradientu je vyšší než g_{max}, s jistotou označeny jako hrana. Naopak pixely, jejichž hodnota gradientu je nižší než g_{min}, jsou s jistotou z rozhodování vyloučeny jako nevyhovující. Pixely s hodnotou gradientu ležící mezi g_{min} a g_{max} jsou označeny jako hrana v případě, že jsou spojeny s pixelem, který jako hrana již označen byl [3, 24].

Na obrázku 2.16 je ukázka detekce hran pomocí Cannyho detektoru hran ($g_{min} = 120$, $g_{max} = 200$).

(a) Původní obrázek (b) Detekované hrany

Obrázek 2.16: Ukázka detekce hran pomocí Cannyho detektoru hran

Cannyho detektor hran v OpenCV

```
import cv2

# nacteni vstupniho obrazku
img = cv2.imread('grayscale_image.png', 0)

g_min = 120 # hodnota spodniho prahu (g min)
g_max = 200 # hodnota horniho prahu (g max)
edges = cv2.Canny(img, threshold1=g_min, threshold2=g_max)

# ulozeni vysledku detekce hran
cv2.imwrite('edges.png', edges)
```

Ukázka kódu 2.5: OpenCV Cannyho detektor hran [24]

2.7 Hledání kontur

Kontura je spojitá křivka spojující všechny body po obvodu objektu [11], které mají stejnou intenzitu. Kontury jsou důležité pro analýzu tvaru objektu a jeho následné rozpoznání. Ačkoliv je možné kontury detekovat i v šedotónových či barevných obrázcích, lepších výsledků algoritmy dosahují na obrázcích, které byly ve fázi předzpracování binarizovány – například prahováním či detekcí hran [25].

Protože je kontura definována jako spojitá křivka, můžeme tak její charakteristiku a vlastnosti podrobit analýze použitím stejných metod, jako pro jiné

křivky. Například výpočet obvodu a obsahu (za předpokladu, že je křivka uzavřená) nám dá představu o její velikosti a určení konvexního obalu a extrémních bodů může být důležité pro analýzu tvaru objektu.

```
import numpy as np
import cv2

# nacteni vstupniho obrazku
img = cv2.imread('image.jpg')

# prevedeni obecneho obrazku na sedotonovy
imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# prahovani sedotonoveho obrazku s prahem 127
ret, thresh = cv2.threshold(imgray, 127, 255, 0)

# hledani kontur
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
```

Ukázka kódu 2.6: OpenCV Hledání kontur [25]

2.8 Označení spojených komponent

Spojenou komponentou je myšlena oblast sousedících pixelů (sousedství je možné definovat více způsoby, obvykle se bavíme o čtyř nebo osmi-směrovém sousedství), které mají stejnou či podobnou intenzitu (u barevných obrázků barvu). Cílem algoritmu označení spojených komponent je přiřadit každé spojené komponentě jedinečné označení (*label*), které se občas také nazývá jako barva [6]. Ačkoliv teoretický popis algoritmu umožňuje práci s černobílými, šedotónovými i barevnými obrázky, v praxi se nejčastěji využívá označení komponent na binárních obrázcích, které vznikly prahováním či detekcí hran. V takovýchto obrázcích spojená komponenta často představuje oblast, která odpovídá určité části objektu a její označení hraje důležitou roli při analýze obrazu.

2.8.1 Spojené komponenty v OpenCV

OpenCV ve verzi 4.5.2 nabízí 2 hlavní metody pro vyhledání a označení spojených komponent [26]:

- `connectedComponent`, která jako vstup požaduje vstupní obrázek a počet sousedících pixelů používaných k určení sousednosti (4 nebo 8).

Jejím výstupem je číslo udávající počet unikátních komponent (N) a obrázek (respektive matice), kdy každému pixelu je přiřazeno označení komponenty ($< N$),

- `connectedComponentsWithStats`, která požaduje vstup stejný jako v případě connectedComponents, ale kromě počtu unikátních komponent a obrázku s označenými pixely vrací i dvojrozměrné pole statistik (pro každou komponentu pole statistik) a pole centroidů (centroid, tedy geometrický střed objektu, je vrácen pro každou komponentu) [26]. Použití těchto metod je znázorněno v ukázce kódu 2.7.

Obě výše uvedené metody předpokládají, že pozadí má černou barvu (0) a popředí bílou. Algoritmus v OpenCV hledá spojené komponenty jen v rámci popředí.

```
import cv2

img = cv2.imread("image.jpg", 0)

# prevedeni obecneho obrazku na sedotonovy
imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# prahovani sedotonoveho obrazku s prahem 127
ret, thresh = cv2.threshold(imgray, 127, 255, 0)

neighbors_connectivity = 8

# spojene komponenty - connectedComponents
num_labels, labels = cv2.connectedComponents(img, connectivity=
    neighbors_connectivity)

# spojene komponenty - connectedComponentsWithStats
numLabels, labels, stats, centroids = cv2.connectedComponentsWithStats(img,
     connectivity=neighbors_connectivity)

# ziskani statistik pro komponentu i:
i = 1

# (x,y) = horni levy bod komponenty
x = stats[i, cv2.CC_STAT_LEFT]
y = stats[i, cv2.CC_STAT_TOP]
width = stats[i, cv2.CC_STAT_WIDTH]
height = stats[i, cv2.CC_STAT_HEIGHT]
area = stats[i, cv2.CC_STAT_AREA] # celkovy obsah komponenty (v px)
```

Ukázka kódu 2.7: OpenCV označení spojených komponent [26]

3 Optické rozpoznávání znaků

Optické rozpoznání znaků, anglicky *Optical Character Recognition* (dále jen OCR), je proces zajišťující převod tištěného či ručně psaného textu obsaženého v digitálních obrázcích (vzniklých například naskenováním dokumentů) do strojově zpracovatelné podoby [35].

Velké uplatnění pro širší veřejnost našly OCR systémy již koncem 20. století, kdy byly využívány například na letištích ke čtení cestovních dokladů. Historické OCR systémy byly silně závislé na použitém fontu a nebylo tak možné je použít na rozpoznání textu v neznámém dokumentu, jako je tomu nyní.

V současné době se tato technika využívá například při digitalizaci přijatých dokumentů (faktur, smluv, výpisů z bankovních účtů), pro automatické rozpoznání poznávacích značek aut (používané při měření rychlosti nebo například jako identifikace vozidel pro vjezd na uzavřené parkoviště), při automatické kontrole cestovních dokladů na letišti, pro účely rozpoznání dopravních značek v autonomních systémech moderních vozidel nebo také pro účely digitalizace knih a umožnění jejich následného indexování a vyhledávání v nich.

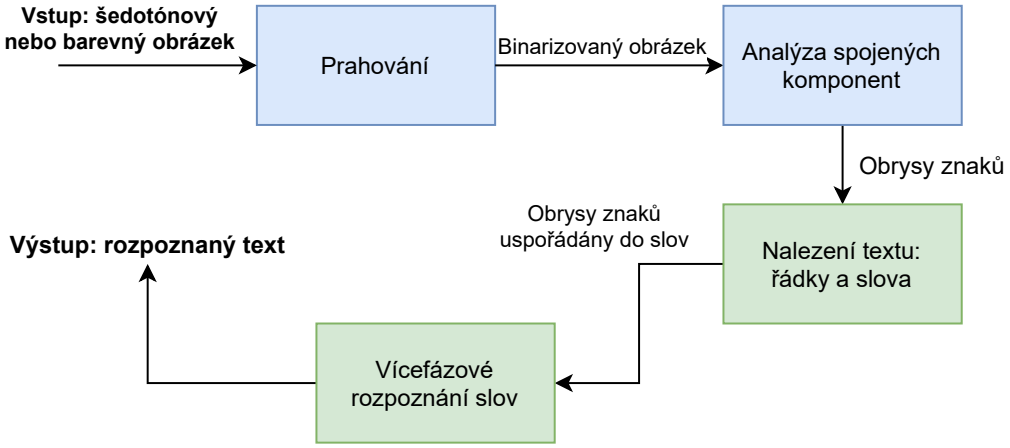
V této kapitole bude stručně přiblížen princip fungování OCR a následně budou představeny některé dostupné komerční i nekomerční systémy zajišťující OCR.

3.1 Princip fungování

Přesný princip fungování se v různých OCR softwarech liší, základní myšlenka však zůstává zachována. Většina algoritmů rozpoznání znaků očekává vstup ve formě binárních obrázků, úkolem předzpracování je tedy vytvořit pro rozpoznávací algoritmus co možná nejlepší binární obrázek, který bude obsahovat minimum šumu a jiných nežádoucích efektů (například otočení stránky). Následně dochází k segmentaci, která obvykle postupuje hierarchicky, tedy nejprve dojde k rozpoznání rozložení stránky, poté nalezení bloků textu, který se dále rozdělí na odstavce, řádky, slova a nakonec i na jednotlivé znaky. Následuje samotné rozpoznání znaků, které se liší podle

použitého OCR systému. V moderních systémech následně dochází k post-processingu, kdy je rozpoznaný text validován oproti slovníku a v případě potřeby upraven. Konečnou fází OCR systému je vrácení textového výstupu v definovaném formátu [1].

R. Smith v rámci své prezentace fungování nástroje Tesseract OCR popsal architekturu OCR a hlavních částí předzpracování způsobem, který je znázorněn na diagramu 3.1 [37]. Adaptivní prahování obvykle dosahuje lepších výsledků, protože není náchylné na nerovnoměrné osvětlení obrázku, které je v tomto případě běžné, a z toho důvodu je upřednostňováno.



Obrázek 3.1: Architektura Tesseract OCR [37]

3.2 Dostupné systémy

Na trhu je v současné době k dispozici mnoho různých OCR systémů. V této sekci bude představeno nejprve několik systémů, které je možno zdarma provozovat na lokálním zařízení, a následně i několik systémů z oblasti cloudových služeb.

3.2.1 Tesseract

OCR systém Tesseract byl původně vyvíjen koncem 20. století společností Hewlett-Packard, roku 2005 byly ale jeho zdrojové kódy uvolněny jako open source a od roku 2006 na jeho vývoj dohlíží společnost Google [40], která jej využívá i ve svých svých systémech například pro detekci textu ve videích nebo detekci spamu ve službě Gmail [12]. Nejnovější stabilní verze (nesoucí

označení 4.1.1) byla zveřejněna v prosinci roku 2019, vývoj OCR řešení ale stále aktivně pokračuje. Aktuální verze Tesseractu k rozpoznání textu používá (LSTM) neuronovou síť a dosahuje výborných výsledků. Tesseract k podpůrným činnostem (například kompenzace natočení stránky, podpora více formátů vstupních obrázků včetně pdf) využívá knihovnu Leptonica. Výhodou je i dostupnost natrénovaných modelů pro více než 100 jazyků či možnost natrénovat vlastní modely [40].

3.2.2 Kraken

Kraken je open source OCR systém optimalizovaný pro použití na historických materiálech a textech nepsaných latinkou. Jako klíčovou funkci jeho autoři uvádějí možnost natrénování vlastního modelu pro analýzu rozložení stránky nebo podporu right-to-left textu [21]. Dle informací z veřejně dostupného repozitáře [20] se systém jeví jako aktivně vyvíjený.

3.2.3 Calamari

Calamari OCR je open source systém napsaný v Pythonu (aktuálně Python 3) založený na systému Kraken. Jako vstup očekává jeden řádek textu, k předzpracování a segmentaci stránky je tedy potřeba použít jiný algoritmus. Pro detekci písmen Calamari OCR používá velmi hluboké neuronové sítě implementované pomocí Tensorflow a k dispozici je několik předtrénovaných modelů s možností vytvoření vlastních. Poslední verze (označená 2.0.2) byla vydána 20. 3. 2021 a systém se zdá být aktivně vyvíjen [2].

3.2.4 GOCR (JOCR)

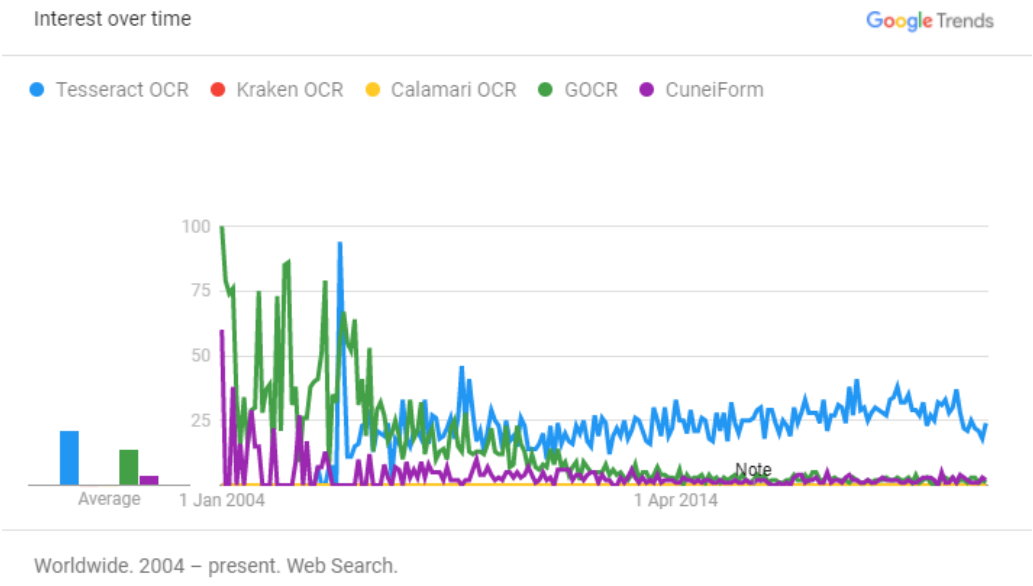
GOCR (z důvodu problémů se získáním internetové domény s názvem také známý pod jménem JOCR) je systém vyvíjený pod GNU Public Licence. Poslední verze systému (s označením 0.52) byla vydána 15. 10. 2018 [33], následný vývoj systému však není možné jednoduše sledovat. Roku 2013 provedl S. Dhiman a A. Singh srovnávací studii mezi Tesseractem a GOCR, jejíž závěrem bylo konstatování, že systém Tesseract dosahoval lepších výsledků v úspěšnosti (accuracy, precision) než GOCR [5].

3.2.5 CuneiForm (Cognitive OpenOCR)

CuneiForm (někdy také nazýván Cognitive OpenOCR) je systém vyvíjený společností Cognitive Technologies. Podobně jako Tesseract byl i CuneiForm nejprve vyvíjen jako closed source a byl dodáván k některým modelům skenerů. Roku 2008 společnost Cognitive Technologie zveřejnila tento systém jako open source. V současné době není k nalezení žádná známka dalšího vývoje systému.

3.2.6 Zájem uživatelů

Na obrázku 3.2 je znázorněn vývoj zájmu uživatelů o jednotlivé OCR systémy podle dat ze služby Google Trends. Data jsou získána za období 2004 – 2021 a jsou vyhodnocena na základě vyhledávání uživatelů po celém světě. K názvu systémů Tesseract, Kraken a Calamari bylo přidáno označení OCR, aby nedošlo k výraznému ovlivnění výsledků vyhledáváním jiných významů těchto názvů. Z grafu je patrné, že od roku 2011 ve vyhledávání open source OCR systémů dominuje Tesseract, tento trend podporují i podrobnější data z kratšího období.



Obrázek 3.2: Zájem uživatelů o jednotlivé výše představené OCR systémy podle dat z Google Trends

3.2.7 Cloudové OCR systémy

Cloudové služby jsou v poslední době na vzestupu a jejich využití pro výpočetně náročné operace jako je OCR se nabízí. Všichni populární poskytovatelé cloudových služeb zároveň nabízí API zajišťující detekci textu v obrázku. V této sekci bude představeno několik základních možností využívání OCR za použití cloudových služeb.

Google Cloud Vision

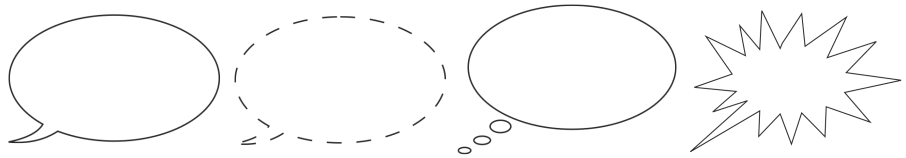
Google Cloud poskytuje OCR služby v rámci služby Vision AI, která kromě OCR slibuje také například rozpoznání objektů na obrázku, obličejů, oblíbených lokací či emocí lidí na fotce. Google pro detekci objektů a rozpoznání textu používá pokročilé metody strojového učení a v kombinaci se službou AutoML Vision slibuje kromě využívání předtrénovaných modelů také možnost vytvoření vlastní klasifikace obrázků [13]. Ted Han a Amanda Hickman provedli v únoru roku 2019 porovnání různých OCR systémů a Google Cloud Vision v testu dosáhl přesvědčivých výsledků [14].

Microsoft Azure Computer Vision

Podobně jako v případě Googlu, i Microsoft v rámci svého cloudového nástroje s názvem Computer Vision poskytuje pokročilou analýzu obrázku zahrnující OCR, detekci objektů a rozpoznání osob a místa [19]. V porovnání se službou nabízenou v rámci Google Cloudu nabízí Microsoft příznivější cenové možnosti (5 000 zpracovaných obrázků měsíčně zdarma, zatímco Google pouze 1 000 a následnou cenu 1 USD za 1 000 obrázků, u kterých je provedeno OCR, zatímco Google stejný objem nabízí za 1.50 USD) [13, 19].

4 Detekce bublin

Komixová bublina je ohraničená oblast na stránce, která obsahuje text – přímou řeč aktérů, křik, šepot, zvuky, přemýšlení aktérů, řeč ozývající se z rádia či televize nebo upřesnění vypravěče. Pozadí bubliny je obvykle bílé a text je psán tmavou (většinou černou) barvou, není to však pravidlem. Bubliny mohou být ztvárněny v mnoha různých tvarech, nejčastější jsou ale 4 tvary uvedené na obrázku 4.1.



(a) (b) (c) (d)

Obrázek 4.1: Nejčastěji se vyskytující komixové bubliny

Tato kapitola poskytuje návrh řešení problému detekce bublin a následného rozpoznání textu v komixových stránkách.

4.1 Definice problému

Pokud OCR systému poskytneme jako vstup celou komixovou stránku, rozpoznaný text obsahuje značné množství chyb (velká část textu v komixových bublinách v takovém případě není detekována). Pokud ale vhodným předzpracováním dosáhneme odstranění rušivých elementů a ponecháme v obrázku pouze text, úspěšnost provedeného OCR dramaticky vzroste (viz obrázek 4.2).

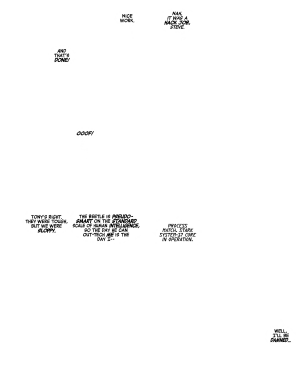
(a) Původní obrázek [43]

RIGHT.

ry —- TONY'S RIGHT.
fame THEY WERE TOUGH,
UT WE WER

ae
=
.

(b) Text rozpoznaný v původním obrázku



(c) Obrázek po předzpracování

NICE
WORK.

AND
THAT'S
DONE/
OOOFS
TONY'S RIGHT. THE BEETLE 1S PSELIDO-
THEY WERE TOUGH, _ SMART ON THE STANDARD
BUT WE WERE SCALE OF HUMAN INTELLIGENCE,
SLOPPY. $O THE DAY HE CAN

OUT-TECH ME IS THE
DAY I--

NAH,
ITWAS A
HACK JOB,
STEVE.

PROCESS
MATCH. STARK
SYSTEN-I? CORE
IN OPERATION.

(d) Text rozpoznaný v obrázku po předzpracování

Obrázek 4.2: Ukázka rozpoznání textu (Tesseract OCR, jazyk angličtina)

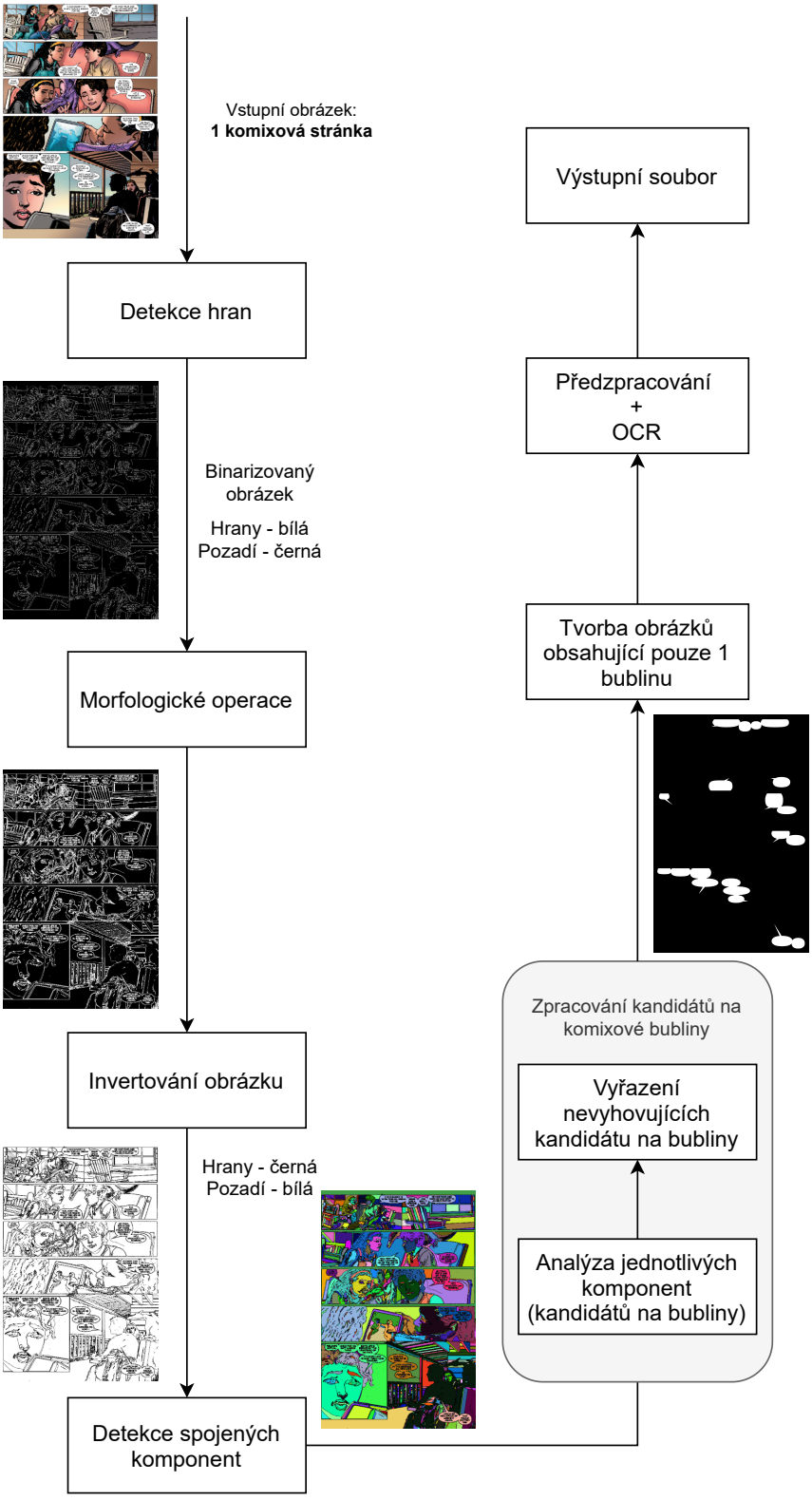
4.2 Navržené řešení

Na základě výše zmíněného experimentu je zřejmé, že pro dosažení přesných výsledků OCR je potřeba z obrázku odebrat veškeré netextové prvky. Aby bylo zároveň možné extrahované textové informace dále využívat pro výzkumné účely skupiny NLP a další aplikace (viz sekce Aplikace), je vhodné nejprve na stránce detekovat komixové bubliny a rozpoznání textu (OCR) následně provádět v kontextu jedné bubliny. Díky tomu bude možné automaticky vytvořit dataset obsahující informace o detekovaných komixových bublinách doplněné o rozpoznaný text v nich.

4.3 Návrh implementace

Vstupem navrženého algoritmu jsou jednotlivé stránky komixů. Pro každou stránku je nejprve provedena detekce hran, následně jsou aplikovány morfologické operace a obrázek je poté invertován. V invertovaném obrázku dojde k vyhledání spojených komponent, kde se každá takto nalezená komponenta označí jako kandidát na komixovou bublinu. Následně se provádí analýza těchto komponent (kandidátů na bublinu) a pomocí různých kritérií se určuje, zda se o bublinu jedná či nikoliv. Nevyhovující komponenty se vyřadí a dále se s nimi nepracuje. Každá vyhovující bublina se použije jako maska, díky které se zkombinováním s vstupním obrázkem vytvoří nový obrázek obsahující pouze jednu bublinu. Po následném předzpracováním (oříznutí a prahování) vznikne vstup pro systém OCR. Celý proces je zakončen uložením veškerých dat ve vhodném formátu. Tento postup je znázorněn na obrázku 4.3.

V následujících sekcích bude nejprve podrobně představen způsob fungování algoritmu včetně zdůvodnění použití jednotlivých kroků a následně bude upřesněn formát, ve kterém jsou výsledky zpracování ukládány.



Obrázek 4.3: Průběh zpracování komixové stránky

4.3.1 Formát vstupního obrázku

Algoritmus jako vstup přijímá barevný či šedotónový obrázek odpovídající jedné stránce komixové knihy v podporovaném formátu. Pro dosažení optimálních výsledků je vhodné použít obrázek vzniklý digitální tvorbou, nikoliv obrázek, který vznikl jako sken stránky komixové knihy (algoritmus ve vstupním obrázku neočekává velkou míru šumu či stínů, které by naskenováním mohly vzniknout).

4.3.2 Detekce hran

Všechny typy komixových bublin jsou od svého okolí odděleny hranou (pozadí komixové bubliny je obvykle vykresleno barvou dostatečně odlišnou od svého okolí a v některých případech je komixová bublina zároveň ohraničena tenkou linkou). Zároveň se uvnitř bubliny žádná hrana kromě textu obvykle nenachází. Je možné tedy předpokládat, že pokud bude na vstupní obrázek správně aplikován hranový detektor, komixová bublina se bude jevit jako souvislá oblast přerušena pouze textovou částí (této skutečnosti se využívá při analýze spojených komponent, zmíněné níže).

Pro detekci hran byl použit Cannyho hranový detektor. V této fázi algoritmu je žádoucí, aby došlo k detekování většího množství hran, tj. raději jako hranu označit něco, co důležitou hranou ve skutečnosti není, než hranu neoznačit. Toto tvrzení je založeno na předpokladu, že uvnitř komixové bubliny nejsou (kromě textu) žádné ostré hrany, které by Cannyho detektor odhalil, tedy označením více hran nedojde k negativnímu ovlivnění detekce bublin. Z toho důvodu byly prahy g_{min} ($threshold_1$) a g_{max} ($threshold_2$) experimentálně určeny jako $g_{min} = 100$ a $g_{max} = 200$.

Aplikováním hranového detektoru na barevný (příp. šedotónový) vstupní obrázek vznikne binarizovaný obrázek, ve kterém je hraničním pixelům (tj. pixelům obsaženým v hraně) nastavena hodnota 1 (resp. 255 v případě 8 bitového 1 kanálového obrázku) a prostoru mezi hranami je nastavena hodnota 0.

4.3.3 Morfologické operace

Po aplikování hranového detektoru v některých případech dochází k jevu, kdy kontura ohraničující komixovou bublinu není uzavřena a jejímu uza-

vření brání mezera velká obvykle 1 či 2 pixely. Za účelem uzavření kontur je použita morfologická operace dilatace, která hrany rozšíří a konturu tak spojí. Zesílení hran nemá na detekci bublin negativní vliv, není proto potřeba provádět složené morfologické operace, jako je například opening či closing.

Byla provedena série experimentů za účelem nalezení strukturního elementu, jehož použitím je možné získat optimální výsledky na různých typech komixových stran. Nejlepších výsledků bylo dosaženo použitím strukturního elementu h (viz 4.1) o velikosti 3×3 ve tvaru kříže či elipsy (na velikosti 3×3 jsou tyto tvary ekvivalentní). Tento strukturní element oproti stejně velkému elementu ve tvaru čtverce méně ovlivňuje okolní pixely a zároveň ve většině případů zajišťuje uzavření neuzavřených kontur.

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.1}$$

4.3.4 Detekce spojených komponent

Výstupem předchozí operace je binarizovaný obrázek, ve kterém došlo ke korekci některých neuzavřených kontur. Oblasti oddělené hranami (tzn. jednotlivé spojené komponenty) jsou nazvány kandidáty na komixové bubliny a v následujícím kroku jsou podrobeny analýze a vyfiltrování nevyhovujících.

Jednotlivé oblasti oddělené hranami je možné získat pomocí algoritmu označení spojených komponent (connected components labeling), kdy je každé spojené komponentě přiřazeno jedinečné označení. Některé knihovny poskytující algoritmy pro označení spojených komponent (např. OpenCV) předpokládají, že pixely popředí, mezi kterými dochází k vyhledání a označení spojených komponent, mají hodnotu 1 (resp. 255). V takovém případě je potřeba nejprve binarizovaný obrázek invertovat – černé pixely (tj. pixely s hodnotou 0) se změní na bílé (tj. pixely s hodnotou 1 resp 255) a opačně.

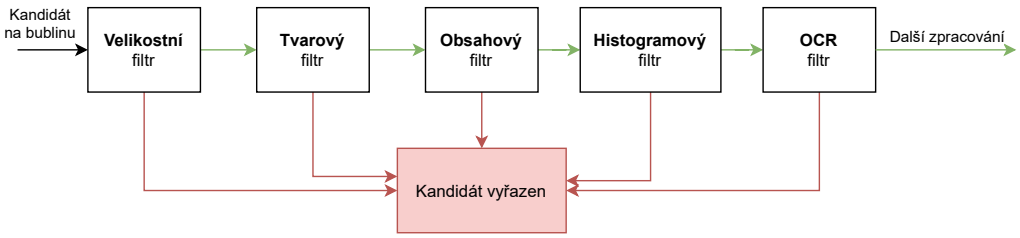
4.3.5 Analýza a filtrace komponent

V předchozím kroku vznikla množina kandidátů na komixové bubliny, kterých je ale na náročnějších komixových stránkách detekováno i několik desítek tisíc. Cílem tohoto kroku je navrhnout vhodné metody, které bublinové

kandidáty, které nepředstavují bublinu, vyřadí.

Za tímto účelem bylo navrženo 5 filtrů, každý specializující se na určité vlastnosti bubliny. Každý kandidát na komixovou bublinu je postupně podroben těmto filtrům a jakmile je nějakým označen jako nevyhovující, je definitivně vyřazen. Průběh filtrování je znázorněn na obrázku 4.4, ve kterém zelené šipky znázorňují schválení kandidáta daným filtrem a červené šipky jeho zamítnutí.

V následujících pěti podsekcích budou podrobněji představeny navržené filtry.



Obrázek 4.4: Filtrování kandidáta na komixovou bublinu

Velikostní filtr

Bubliny v komixu zaujímají určitý prostor. Je rozumné očekávat, že bublina nebude extrémně malá, neboť je potřeba, aby se v ní nacházel text. Zároveň není bublina obvykle extrémně velká, protože obrázek hraje v komixu důležitou roli. Rozlišení komixové stránky není blíže určeno, na velikost je tedy potřeba pohlížet jako na poměr plochy, kterou zaujímá kandidát na bublinu, a plochy celé stránky. Na základě experimentálního měření velikosti komixových bublin v trénovacím datasetu (viz kapitola Experimenty, tabulka 5.1) byly určeny prahy velikosti – $a_{min} = 0,0005$ a $a_{max} = 0,05$.

Označme a_i jako obsah (počet pixelů) i-té bubliny a S jako plochu celého vstupního obrázku (součin šířky a výšky obrázku v pixelech). Kandidát na bublinu je vyřazen a další zpracování u něj již následně neprobíhá, pokud nesplňuje následující nerovnici.

$$a_{min} < \frac{a_i}{S} < a_{max} \tag{4.2}$$

Tvarový filtr

Tvar komixových bublin se mezi komixy velmi různí. Zatímco na některých stránkách jsou komixové bubliny kulaté, v jiných mohou být hranaté či zaujímat i velice atypické – například hvězdicovité – tvary (viz obrázek 4.1). Přesto je však možné najít některé znaky, které jsou napříč komixy podobné.

Na základě experimentálního měření (viz kapitola experimenty) bylo zjištěno, že poměr mezi obvodem a obsahem bubliny je možné shora omezit hodnotou $pa_{max} = 0,15$ a zdola pak hodnotou $pa_{min} = 0,0075$. Kandidáti na komixové bubliny s poměrem obvodu a obsahu menším než pa_{min} nebo větším než pa_{max} jsou označeni jako nevyhovující.

Dále je v souladu s experimentálním měřením možné očekávat následující:

$$(0,005 \cdot W) < w_i < (0,6 \cdot W) \tag{4.3}$$

$$(0,005 \cdot H) < h_i < (0,6 \cdot H) \tag{4.4}$$

kde W je šířka stránky, w_i je šířka opsaného obdélníku i-tého kandidáta na bublinu, H je výška stránky a h_i je výška opsaného obdélníku i-tého kandidáta na bublinu. Pokud kandidát na bublinu výše uvedený předpoklad nesplňuje, je označen jako nevyhovující.

Posledním kritériem, které tento filtr posuzuje, je poměr mezi plochou celé bubliny včetně textu a plochy, kterou zaujímá pouze pozadí bubliny (tj. bez textu).

Obsahový filtr

Výskyt textu v bublině se projevuje tím, že pozadí bubliny je v binarizovaném obrázku přerušováno oblastí, ve které se nachází text. Obsahový filtr analyzuje každý pixelový řádek kandidáta na bublinu a počítá, kolikrát dojde ke změně barvy pixelu z bílé na černou a naopak. Stejný postup aplikuje i na jednotlivé pixelové sloupce. Je rozumné předpokládat, že pixelové řádky, které protínají text, budou obsahovat relativně vysoké množství změn – minimálně 2 změny (pozadí → znak → pozadí) pro každý znak. Pixelové sloupce budou obsahovat minimálně 2 změny v případě, že protínají text, který má pouze 1 řádek.

Dále je stanovena hranice vyjadřující minimální počet změn, ke kterým musí dojít, aby byl pixelový řádek (resp. pixelový sloupec) označen jako text pro-

tínající. Minimální počet změn pro řádek určen je roven 6 (resp. 2 pro sloupec).

Protože část komixové bubliny neobsahuje text, je počet řádek, které neprotínají řádek textu, poměrně vysoký. Z toho důvodu vyhodnocení filtru spočívá v určení poměru řádků protínajících text a řádků neprotínajících text. Pokud je tento poměr nižší než odhadnutá hodnota 0.1, je kandidát na bublinu označen jako nevyhovující. Jinými slovy, bublina musí obsahovat alespoň 10 % pixelových řádků, které protínají řádek textu.

Aby nedocházelo ke zkreslení výsledků dilatací provedenou v předchozím kroku, je tento filtr vyhodnocován na obrázku před provedením dilatace, která by mohla způsobit spojení jednotlivých znaků.

Histogramový filtr

Aplikujeme masku kandidáta na bublinu na vstupní obrázek a (obecně barevný) výsledek následně převedeme na šedotónový. Histogram intenzit pixelů zaokrouhlených na celé desítky v takto získaném obrázku je poté vstupem pro histogramový filtr. Bubliny obvykle obsahují velký kontrast mezi intenzitou pixelů v pozadí bubliny a intenzitou pixelů reprezentujících znaky. Ten se v histogramu projevuje dvěma lokálními maximy, jedno v tmavé části spektra (obvykle reprezentující znaky) a jedno ve světlé části spektra (obvykle reprezentující pozadí bubliny). Pokud kandidát na bublinu neobsahuje ve svém histogramu 2 dostatečně vzdálená lokální maxima, označí se jako nevyhovující.

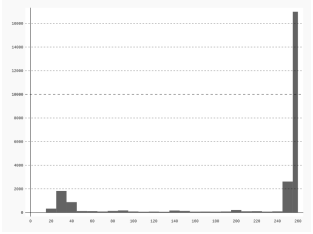
Na obrázku 4.5 je znázorněn rozdíl v histogramu komixových bublin a nevyhovujících kandidátů na bublinu. Černé pixely ve vnějším okolí kandidáta na bublinu se ve výpočtu histogramu nezohledňují.

OCR filtr

K tomuto závěrečnému filtru se dostanou již jen jednotky kandidátů na bublinu, které je možné vyřadit – pravděpodobnost, že kandidát na bublinu, který je podrobován testu pomocí tohoto filtru, obsahuje text, je tedy již velmi vysoká. Smyslem filtru je vyřadit takové kandidáty na bubliny, které není možné vyřadit pomocí výpočetně méně náročných filtrů.

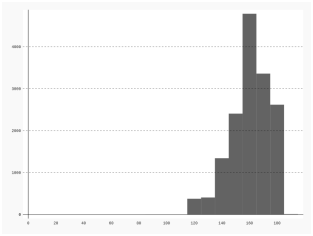
Filtr pomocí metod předzpracování a následného OCR (podrobnosti uvedeny v sekci 4.3.6) zjistí, zda se v kandidátovi na bublinu nachází rozpozna-

(a) Komixová bublina



(b) Histogram komixové bubliny

(c) Nevyhovující kandidát na komixovou bublinu



(d) Histogram kandidáta na komix. bublinu

Obrázek 4.5: Ukázka histogramu komixové bubliny a nevyhovujícího kandidáta na bublinu

telný text. Zároveň je určen průměr míry jistoty (confidence) jednotlivých rozpoznaných slov vážený délkou daného slova. Pokud v bublině není rozpoznán žádný text, nebo je průměrná míra jistoty nižší než stanovená prahová hodnota $c_{min} = 20$, je kandidát na bublinu označen jako nevyhovující.

4.3.6 OCR

Aby bylo možné určit k jaké komixové bublině se rozpoznaný text vztahuje, probíhá OCR pro každou bublinu zvlášť. V úvodu této práce bylo experimentálně zjištěno, že nejlepších výsledků systémy OCR dosahují pro obrázky, které obsahují pouze rozpoznávaný text. Z toho důvodu je před samotným rozpoznáváním textu provedeno předzpracování, které zahrnuje:

1. aplikování bublinové masky na vstupní obrázek (čímž vznikne obrázek o velikosti původního obrázku, kde je vše kromě oblasti komixové bubliny nahrazeno černou barvou),
2. segmentace obrázku na jednotlivé bubliny (odstranění přebytečné černé plochy),

3. prahování obrázku s bublinou (Otsu metodou),

4. vytvoření opsaného obdélníku obsahující pouze text na bílém pozadí – po oříznutí obrázku provedeném v druhém kroku je vnější okolí bubliny černé, a protože OCR funguje lépe, pokud se v obrázku nenachází žádné přebytečné hrany, dojde k nahrazení černé barvy vně bubliny za barvu pozadí bubliny (viz obrázek 4.6).

(a) Bublina obsahující černé pozadí

(b) Bublina po nahrazení černého pozadí

Obrázek 4.6: Ukázka nahrazení pozadí vně komixové bubliny

Dále je možné si všimnout, že se v komixových textech téměř nevyskytují malá písmena. Tohoto poznatku bylo využito pro implementaci whitelistu rozpoznávaných znaků. Podrobnosti o zvoleném OCR řešení a konkrétním způsobu implementace jsou uvedeny v odstavci 4.4.5 kapitoly Popis implementace.

4.3.7 Uložení výstupu

Výstup programu byl navržen v souladu s požadavky výzkumné skupiny NLP pro umožnění dalšího zpracování. Pro každou komixovou stránku je vytvořen textový soubor obsahující JSON reprezentaci pole informací o jednotlivých bublinách a rozpoznaném textu. Pro každou bublinu jsou zde uloženy informace o poloze (x,y) a velikosti (šířka, výška) opsaného obdélníku, pozice centroidu (tj. středu bubliny), obsah bubliny a informace o rozpoznaném textu a míry jistoty (confidence). Ukázka JSON výstupu pro reálnou komixovou bublinu viz 4.1.

```
[
  {
    "bubbleInfo": {
      "id": 4939,
      "area": 46791,
      "boundingBox": {
        "x": 1420,
        "y": 1013,
        "width": 406,
        "height": 266
      },
      "centroid": {
        "x": 1599,
        "y": 1148
      }
    },
    "ocr": {
      "text": "OH YEAH.\nERIK'S ONE\nOF THOSE\nFOLKS WHO'S\
          nIMPOSSIBLE\nTO KILL.\nE'LL\n\nPROBABLY LIVE\nFOREVER.",
      "avgConfidence": 80,
      "maxConfidence": 96
    }
  },
  ...
]
```

Ukázka kódu 4.1: Struktura JSON výstupu

4.4 Popis implementace

V této sekci bude nejprve zdůvodněna volba použitých technologií a následně bude popsán program z hlediska implementace.

4.4.1 Programovací jazyk

S ohledem na výzkumný charakter práce, kdy byla často potřeba měnit rozsáhlé části implementace, byl zvolen programovací jazyk **Python**. Python v kombinaci s knihovnou **OpenCV** pro zpracování obrázků a knihovnou **NumPy** pro práci s vektory a velkými maticemi nabízí velmi flexibilní způsob vývoje.

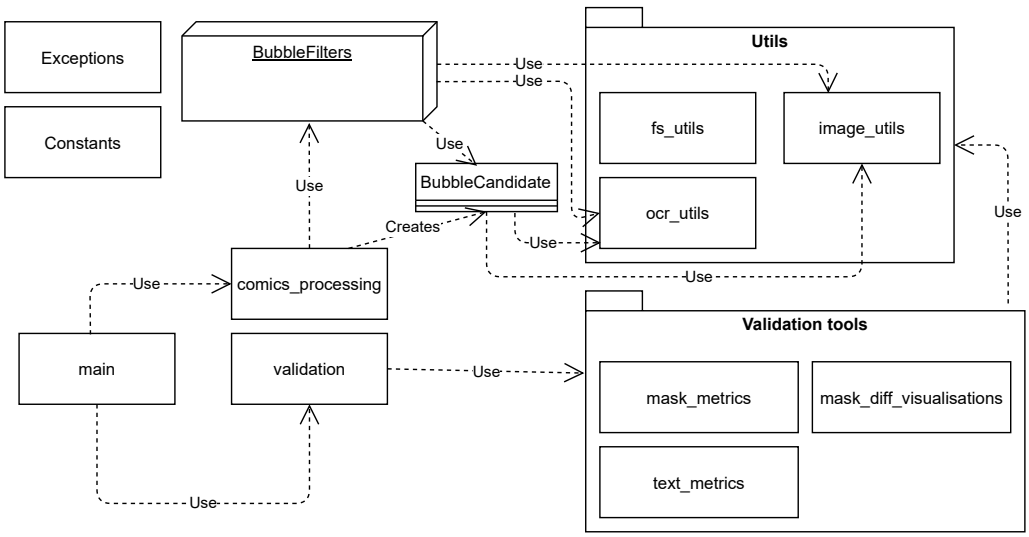
4.4.2 Dekompozice problému

Vstupním bodem programu je soubor `main.py`, který zajišťuje zpracování přijatých parametrů a řídí další chod programu. Zpracování vstupní komixové stránky, tedy nalezení bublin a rozpoznání textu, dále řídí modul

`comics_processing.py` a jeho hlavní funkce `process_comic_page`, která přijímá cestu ke vstupnímu souboru a cesty k uložení bublinové masky a informací o bublinách v textové podobě. Tato metoda následně řídí předzpracování, nalezení spojených komponent a s tím související segmentaci komixové stránky na bublinové kandidáty. Poté řídí analýzu a následnou filtraci bublinových kandidátů, provedení OCR a zajišťuje správné uložení výsledků zpracování.

K veškerým operacím s obrázky jsou využívány funkce umístěné v modulu `image_utils`, díky čemuž není celý program závislý na použití konkrétní knihovny pro práci s obrázky a je tak možné v případě potřeby nahradit současně používanou knihovnu **OpenCV** za jinou. Stejný princip je aplikován i na operace spojené s rozpoznáním textu, které se koncentrují v modulu `ocr_utils.py`. Operace související s filesystémem (například mazání souborů a složek nebo kontrola existence souboru) jsou umístěné v modulu `fs_utils`.

Na obrázku 4.7 je znázorněna vzájemná závislost jednotlivých částí systému.



Obrázek 4.7: UML diagram závislostí částí systému

4.4.3 Kandidát na bublinu

Každá komponenta získaná algoritmem pro nalezení spojených komponent se stává kandidátem na bublinu. Kandidát na bublinu je implementačně reprezentován třídou `BubbleCandidate`, která v konstruktoru přebírá základní informace, které jsou dostupné hned po nalezení komponenty – číslo kom-

ponenty (*label*), centroid, plochu komponenty (tj. plochu pozadí bubliny bez textu) a poměr plochy komponenty vůči ploše celého vstupního obrázku.

Instance `BubbleCandidate` je následně využívána při analýze a filtraci. V rámci časové optimalizace a minimalizace duplicitních výpočtů uchovává instance výsledky již vypočítaných operací. Díky tomu je možné přistoupit z více míst programu k detekované kontuře kandidáta na bublinu, binární masce, segmentu vstupního obrázku, který kandidát reprezentuje, a v neposlední řadě také informace o provedeném OCR.

`BubbleCandidate` poskytuje k těmto hodnotám *gettery*, které při použití zkontrolují, zda byl již výpočet proveden. Pokud ano, vrátí poslední výsledek. Pokud ne, provedou výpočet, výsledek uloží a vrátí jej. Pokud je kandidát na bublinu označen jako nevyhovující, dojde v zájmu uvolnění použité paměti k odstranění veškerých hodnot, které byly pro daného kandidáta na bublinu uloženy.

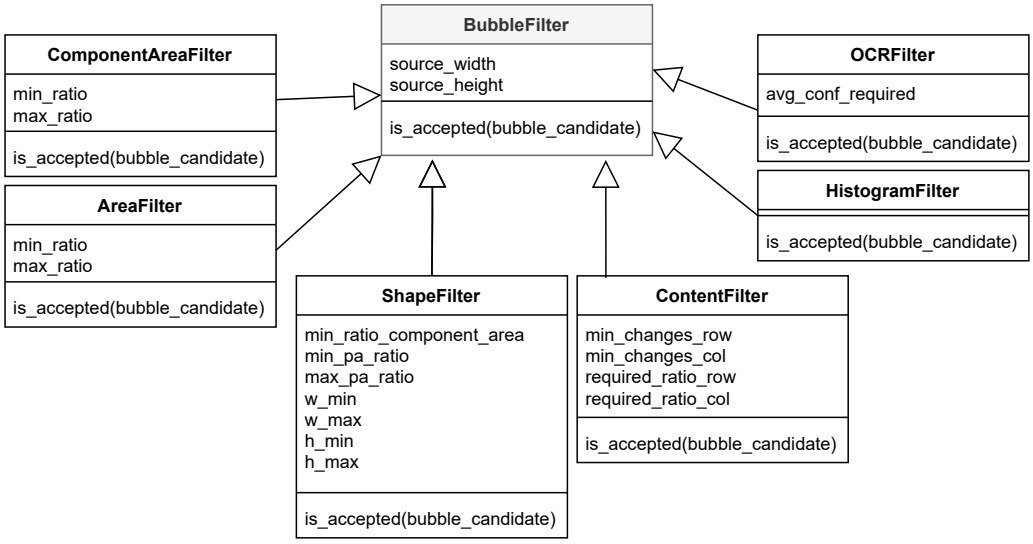
4.4.4 Filtry

Pro analýzu a filtrování kandidátů na bublinu je využit objektový přístup. Je definována třída `BubbleFilter`, která je předkem pro všechny konkrétní filtry a v potomcích zaručuje dostupnost šířky a výšky vstupního obrázku (`source_width` a `source_height`) a přítomnost metody `is_accepted`. Metoda `is_accepted` přijímá jako parametr kandidáta na bublinu a informaci, zda má filtr v případě zamítnutí daného kandidáta vypsat zprávu s důvodem zamítnutí. Pořadí prováděných filtrů bylo určeno s ohledem na minimalizaci času potřebného pro provedení filtrace. Nejprve tedy dochází k použití výpočetně méně náročných filtrů, které vyřadí z dalšího rozhodování co největší počet kandidátů na bublinu. Výpočetně náročné filtry (OCR filtr), který vyřadí malé množství kandidátů, je prováděn v závěru filtrace.

Činnost filtrů byla vysvětlena v sekci 4.3.5, názvy tříd odpovídají anglickým překladům názvů filtrů uvedených v sekci 4.3.5. Na obrázku 4.8 je znázorněn UML diagram tříd používaných pro filtrování kandidátů na bubliny.

4.4.5 OCR

Pro účely rozpoznání textu (OCR) byl vybrán systém **Tesseract**, který z dostupných open-source systémů dosahoval na předzpracované reprezentativní množině dat nejlepších výsledků. Systém Tesseract je zároveň ve velké



Obrázek 4.8: UML diagram tříd použitých pro filtrování

míře využíván výzkumnou skupinou NLP, pro kterou je tento program určen. Pro použití Tesseractu z prostředí Pythonu je používána obalová knihovna **pytesseract**.

Protože se v komixových bublinách vyskytují převážně velké znaky (A-Z), algoritmus nejprve zkouší provést rozpoznání textu s využitím pouze velkých znaků (*whitelist*). Pokud není dosaženo minimální požadované míry jistoty (*confidence*), je OCR provedeno znovu s možností využití všech znaků abecedy.

Tesseract mimo jiné nabízí 14 různých módů segmentace stránky (*page segmentation mode*), které je možné přepínat přepínačem `-psm`. Nejlepších výsledků bylo dosaženo použitím módu
`3 - "Fully automatic page segmentation, but no OSD"`.

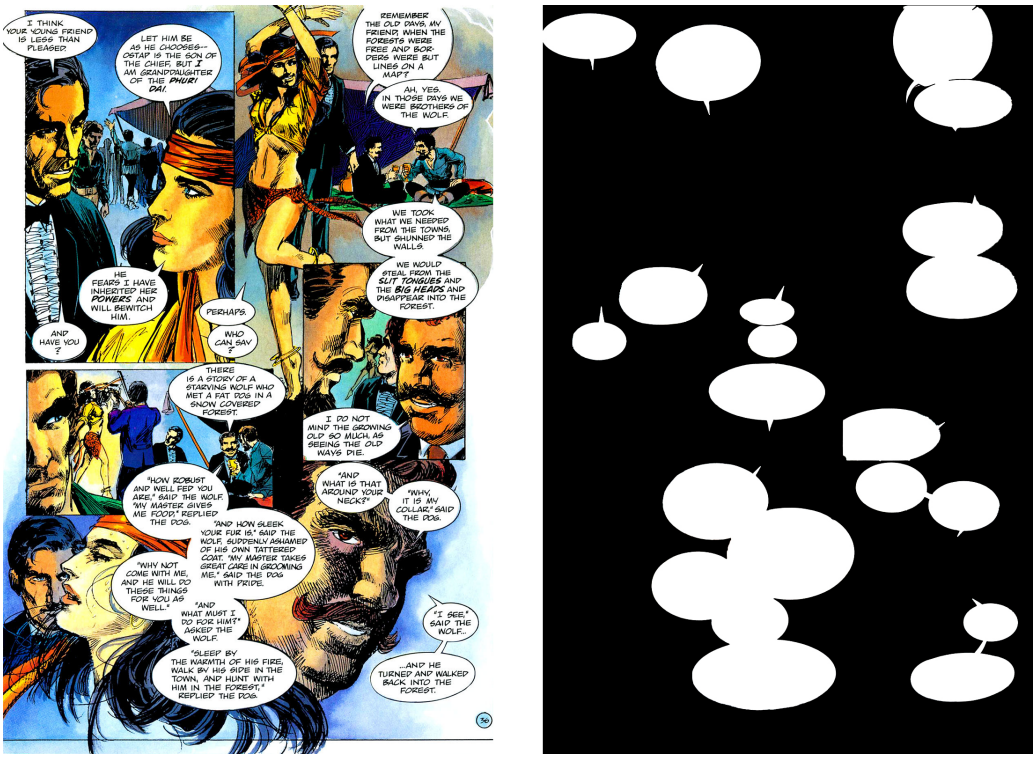
5 Experimenty

Pro ověření výsledků detekce bublin a rozpoznaného textu bylo navrženo několik experimentů. V této kapitole bude nejprve představen způsob tvorby anotovaného datasetu komixových bublin a textu v nich, který se následně používá k vyhodnocení navržených experimentů jako ground truth (GT). Dále jsou vysvětleny metriky používané k vyhodnocení úspěšnosti detekovaných bublin a rozpoznaného textu a popsány dosažené výsledky. V závěru této kapitoly je nastíněno možné využití vytvořeného programu mimo výzkumnou skupinu NLP.

5.1 Anotovaný dataset

Nejprve byla vytvořena reprezentativní množina komixů, které se lišily způsobem vykreslení bublin, jejich velikostí či fontem nebo barvou textu. Pro každý typ komixu bylo následně náhodně vybráno několik stránek, které byly manuálně anotovány. Pro každou takto vybranou stránku vznikla bublinová maska a množina textů nacházejících se v bublinách.

Bublinová maska byla vytvořena jako šedotónový obrázek (1 barevný kanál, barevná hloubka 8 bitů) o stejných rozměrech, jako komixová stránka, ve kterém má pixel na pozici [x,y] intenzitu 255 (bílá) v případě, že je tento pixel součástí bubliny v původní komixové stránce, a intenzitu 0 (černá) v případě, že součástí bubliny není. Na obrázku 5.1 je znázorněna komixová stránka a jí odpovídající bublinová maska. Tímto způsobem vznikl dataset čítající 319 dvojic originálních komixových stránek a anotovaných bublinových masek.



Obrázek 5.1: Originální stránka komixu (vlevo) a vytvořená bitmapa komixových bublinek (vpravo)

Při **anotaci textů** byly nejprve pomocí nástroje na anotaci textů v obrázcích označeny všechny bubliny na stránce komixu a pro každou z nich byl přepsán text, který obsahuje. Výstupem této anotace je soubor ve formátu JSON obsahující pozici levého horního rohu obdélníku opsaného bublině, jeho šířku, výšku a určený text.

Vytvořený dataset byl náhodně rozdělen na 2 části – trénovací a testovací. Pomocí trénovacího datasetu byly určeny vhodné parametry používané při segmentaci komixové stránky a filtraci bublin, testovací dataset se použil pro vyhodnocení úspěšnosti programu.

5.2 Statistiky trénovacího datasetu

Pro správné určení některých parametrů používaných při segmentaci komixové stránky a následné filtraci kandidátů na komixové bubliny byla na trénovacím datasetu provedena série měření, která měla za cíl změřit poměr:

- plochy komixové bubliny vůči ploše celé stránky (v tabulce 5.1 označeno jako plocha bubliny),
- šířky obdélníku opsaného komixové bublině vůči šířce celé stránky (v tabulce 5.1 označeno jako šířka bubliny),
- výšky obdélníku opsaného komixové bublině vůči výšce celé stránky (v tabulce 5.1 označeno jako výška bubliny),
- obvodu a obsahu kontury ohraničující komixovou bublinu (v tabulce 5.1 označeno jako obvod/obsah).

Statistické údaje výsledků pozorování jsou uvedeny v tabulce 5.1, kompletní naměřená data jsou k dispozici na přiloženém DVD.

	plocha bubliny	šířka bubliny	výška bubliny	obvod/obsah
min	0.0007273	0.043259557	0.025625	0.01209717
max	0.0489717	0.542253521	0.312374	0.12326984
avg	0.0081904	0.164814953	0.111918	0.03103635
medián	0.0063153	0.145372233	0.100101	0.02827294
rozptyl	0.0000388	0.006365181	0.002741	0.00017430

Tabulka 5.1: Výsledky pozorování komixových bublin v trénovacím datasetu

5.3 Metriky

V této sekci budou popsány metriky používané ke kvantifikování úspěšnosti fungování navrženého programu. Metriky je možné rozdělit podle použití na metriky detekce bublin a metriky rozpoznaného textu.

5.3.1 Metriky detekce bublin

Definujme 4 možné stavy bublin:

- **pozitivní oblasti** (true positives - TP): oblasti, které byly programem detekovány jako bublina a ve skutečnosti s o bubliny jedná,
- **negativní oblasti** (true negatives - TN): oblasti, které program správně nevyhodnotil jako bubliny,

- **falešně pozitivní** (false positives - FP): oblasti, které byly programem detekovány jako bublina, ale ve skutečnosti se o bubliny nejedná,
- **falešně negativní** (false negatives - FN): oblasti, které nebyly programem detekovány jako bublina, ale ve skutečnosti se o bubliny jedná.

Přesnost (precision)

Přesnost (precision) v případě binární klasifikace vyjadřuje, kolik vzorků z celkového počtu pozitivních klasifikátor jako pozitivní správně určil. Ve vzorci 5.1, kde TP a FP odpovídají definici v minulé sekci, je uvedeno formální vyjádření tohoto poměru [38].

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

Přesnost (accuracy)

Accuracy se v češtině označuje také jako přesnost, nicméně v tomto případě se jedná o podíl počtu správně určených vzorků (TP a TN) vůči celkovému počtu vzorků. Formální zápis accuracy je vyjádřen v rovnici 5.2 .

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

Úplnost (recall)

Úplnost je definována jako poměr odhalených pozitivních vzorků ku celkovému počtu pozitivních vzorků. Formální zápis úplnosti (recall) je uveden v rovnici 5.3 [38].

$$recall = \frac{TP}{TP + FN} \tag{5.3}$$

F-míra

F-míra vyjadřuje vztah mezi přesností (precision) a úplností (recall). Pro účely této práce je F-míra považována za F_1-score, tedy variantu obecné F-míry, kdy má přesnost a úplnost stejnou váhu. F-míru můžeme formálně

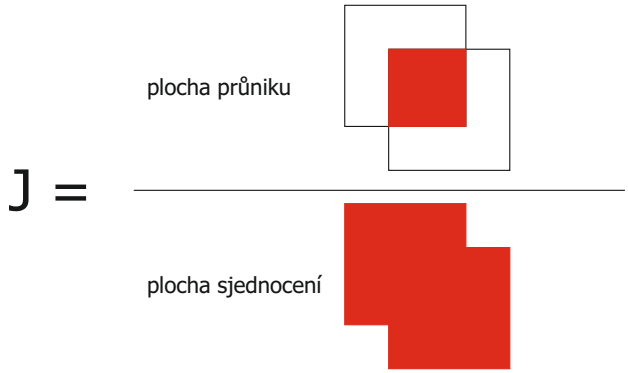
vyjádřit vztahem uvedeným v rovnici 5.4 [42].

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{5.4}$$

Jaccardův index

Jaccardův index slouží k popsání podobnosti konečných 2 množin a je definován jako velikost průniku dělená velikosti sjednocení daných množin (viz rovnice 5.5) [4]. V této práci se Jaccardův index používá k vyhodnocení úspěšnosti detekce bublin, za konečné množiny se tedy považují množiny pixelů odpovídající anotované komixové bublině a detekované bublině. Na obrázku 5.2 je znázorněn výpočet Jaccardova indexu (někdy také označovaného jako Intersection over Union – IoU).

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.5}$$



Obrázek 5.2: Znázornění výpočtu Jaccardova indexu

5.3.2 Metriky rozpoznaného textu

Word Error Rate

Word error rate (dále jen WER) je často používanou metrikou pro vyhodnocení úspěšnosti rozpoznání textu na úrovni jednotlivých slov. Formálně je možné WER zapsat rovnicí 5.6, kde:

- S vyjadřuje počet slov, ve kterých došlo při rozpoznávání k záměně,

- D vyjadřuje počet slov, které v rozpoznaném textu chybí,
- I vyjadřuje počet slov, které se v rozpoznaném textu nacházejí, ačkoliv v původním textu nejsou,
- N vyjadřuje celkový počet slov, který se nachází v původním textu.

$$WER = \frac{S + D + I}{N} \tag{5.6}$$

Character Error Rate

Pro výpočet character error rate (dále jen CER) je použita stejná logika, jako při výpočtu WER, jen na úrovni jednotlivých písmen. Formálně je tedy možné CER zapsat rovnicí 5.7, kde:

- S vyjadřuje počet písmen, ve kterých došlo při rozpoznávání k záměně,
- D vyjadřuje počet písmen, které v rozpoznaném textu chybí,
- I vyjadřuje počet písmen, které se v rozpoznaném textu nacházejí, ačkoliv v původním textu nejsou,
- N vyjadřuje celkový počet písmen, který se nachází v původním textu.

$$CER = \frac{S + D + I}{N} \tag{5.7}$$

5.4 Dosažené výsledky

Na 24 komixových stránkách z testovacího datasetu byla provedena série experimentů za účelem posouzení úspěšnosti detekce komixových bublin a rozpoznání textu v nich. Pro každou stránku byla vyhodnocována míra F-míra a Jaccardův index na základě porovnání pixelů v anotované a detekované komixové masce. Dále byl vyhodnocen počet skutečně pozitivních, falešně pozitivních a falešně negativních detekovaných bublin na stránce a na základě těchto hodnot byla spočítána hodnota přesnosti (precision), úplnosti (recall) a F-míra na úrovni bublin. Rozpoznaný text byl porovnáván s anotovaným textem, na základě čehož byla vypočítána hodnota WER a CER.

V tabulce 5.2 se nachází základní statistické údaje výsledků měření úspěšnosti detekce komixových bublin.

	min	**max**	**avg**	**median**
Pixelové metriky				
Jaccard idx	0.521	0.990	0.911	0.985
F1	0.563	0.995	0.938	0.992
Bublinové metriky				
TP	1	10	5.375	5
FP	0	3	0.542	0
FN	0	2	0.250	0
Precision	0.250	1	0.873	1
Recall	0.750	1	0.966	1
F1	0.400	1	0.900	1

Tabulka 5.2: Statistika měření úspěšnosti detekce komixových bublin

Z tabulky 5.3, která obsahuje základní statistické údaje výsledků měření úspěšnosti provedení OCR v komixových bublinách, vyplývá, že průměrná hodnota WER (word error rate) dosahuje hodnoty 0.17, tedy že bylo bezchybně rozpoznáno 83 % slov v komixových bublinách. Z průměrné hodnoty CER (0.11) je možné zároveň určit, že 89 % všech znaků v komixových bublinách bylo rozpoznáno správně. Ačkoliv metrika CER obvykle dosahuje nižších hodnot než WER, v případě minimální hodnoty dosažené na testovacím datasetu tomu tak nebylo. Program na jedné komixové stránce rozpoznal sekvenci 2 mezer místo 1, což nezpůsobilo negativní ovlivnění WER, ale v CER je tato chyba zaznamenána.

	min	**max**	**avg**	**median**
WER	0	0.428	0.166	0.146
CER	0.006	0.313	0.111	0.071

Tabulka 5.3: Statistika měření chybovosti provedeného OCR

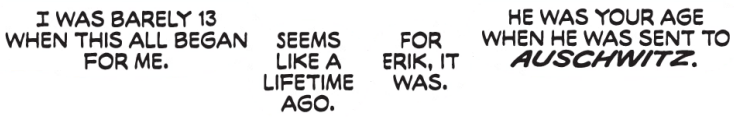
Pokud program nedetekuje skutečnou bublinu (tj. v případě výskytu falešně negativních bublin), OCR dané bubliny není provedeno, což výrazně negativně ovlivní úspěšnost rozpoznaného textu (dojde ke zvýšení hodnoty WER a CER). Z toho důvodu algoritmus upřednostňuje úplnost (recall) před přesností (precision) a tím je tento negativní efekt potlačen.

Kompletní výsledky měření úspěšnosti detekce bublin a rozpoznání textu jsou k dispozici v příloze B na straně 60. Vizualizace výsledků detekce jsou uvedeny v příloze C.

5.4.1 Limity současné implementace programu

Program nedokáže správně rozpoznat text, pokud se nachází v několika spojených bublinách. Text je v takovémto případě detekován po řádcích, což způsobí nesprávné pořadí slov (viz obrázek 5.3).

(a) Komixová bublina



(b) Komixová bublina - po předzpracování

I WAS BARELY 13 HE WAS YOUR AGE
WHEN THIS ALL BEGAN SEEMS FOR WHEN HE WAS SENT TO
FOR ME. LIKEA ERIK, IT AUSCHWITZ.
a WAS.

(c) Rozpoznaný text

Obrázek 5.3: Komixová bublina, u které dochází ke špatnému rozpoznání textu

Program zároveň nedosahuje optimálních výsledků pro obrázky, které vznikly naskenováním vytištěné komixové stránky. Takové obrázky často obsahují vyšší míru šumu či nejsou rovnoměrně nasvíceny, na což není předzpracování optimalizováno.

5.5 Aplikace

Kromě využití automatické tvorby datasetu pro účely výzkumné skupiny NLP může být vytvořený program použit například:

- při automatickém překladu textu komixů do jiného jazyka (přeložený text je možné díky znalosti velikosti a tvaru bubliny vhodně umístit zpět do komixu),
- pro vytvoření databáze komixů, ve které bude možné vyhledávat podle textů v nich obsažených,
- pro automatickou tvorbu datasetu pro klasifikaci komixů na základě textu,
- jako podklad pro automatickou hlasovou čtečku textu (text to speech aplikace), která by zpřístupnila komixy i dětem, které ještě neumí číst.

6 Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat program, který pomocí segmentace komixových stránek detekuje bubliny a následně provedením OCR rozpozná text v nich. Segmentace byla řešena analýzou spojených komponent s vhodným předzpracováním obrázku doplněnou o filtraci falešně pozitivních bublin.

Všechny techniky analýzy a zpracování obrazu použité pro segmentaci byly vysvětleny v teoretické části práce. V praktické části byla popsána architektura navrženého řešení a dekompozice jednotlivých částí problému.

Pro ověření detekce bublin byla vytvořena množina ručně anotovaných komixových stránek, která byla následně porovnána s bublinovými maskami detekovanými navrženým programem. Průměrná hodnota F-míry vztažená ke komixovým bublinám byla 0.938. Byla rovněž provedena sada experimentů, která vyhodnotila úspěšnost OCR, přičemž průměrná hodnota CER byla změřena jako 0.111. Vzhledem k poměrně složitému rozložení stránky, často se vyskytujícím atypickým slovům (citoslovce) a nestandardnímu interpunkčnímu značení (např. násobné otazníky) považuji tuto hodnotu za velmi dobrou.

Práci by bylo možné vylepšit použitím metod strojového učení (např. umělá neuronová síť), díky kterým by bylo možné efektivněji vyřešit filtrování komixových bublin a dosáhnout tak celkově lepších výsledků.

Seznam zkratek

OCR	Optical Character Recognition
px	pixel
API	Application Programming Interface
NLP	Natural Language Processing
GT	Ground truth
JSON	JavaScript Object Notation
AI	Artificial Intelligence
TN	True Negative
TP	True Positive
FP	False Positive
FN	False Negative
CER	Character Error Rate
WER	Word Error Rate

Literatura

[1] Bieniecki, W. – Grabowski, S. – Rozenberg, W. Image preprocessing for improving ocr accuracy. In *2007 international conference on perspective technologies and methods in MEMS design*, s. 75–80. IEEE, 2007.

[2] Calamari-OCR. *GitHub Calamari-OCR* [online]. [cit. 2021-04-25]. Dostupné z: `https://github.com/Calamari-OCR/calamari`.

[3] Canny, J. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1986, PAMI-8, 6, s. 679–698. doi: 10.1109/TPAMI.1986.4767851.

[4] DeepAI. *Jaccard Index* [online]. DeepAI, May 2019. [cit. 2021-04-26]. Dostupné z: `https://deepai.org/machine-learning-glossary-and-terms/jaccard-index`.

[5] Dhiman, S. – Singh, A. Tesseract vs gocr a comparative study. *International Journal of Recent Technology and Engineering*. 2013, 2, 4, s. 80.

[6] Di Stefano, L. – Bulgarelli, A. A simple and efficient connected components labeling algorithm. In *Proceedings 10th International Conference on Image Analysis and Processing*, s. 322–327, 1999. doi: 10.1109/ICIAP.1999.797615.

[7] etymonline. *morphology (n.)* [online]. Online Etymology Dictionary, 2021. [cit. 2021-04-08]. Dostupné z: `https://www.etymonline.com/word/morphology`.

[8] Fisher, R. et al. *Feature Detectors* [online]. 2003. [cit. 2021-04-09]. Dostupné z: `https://homepages.inf.ed.ac.uk/rbf/HIPR2/featops.htm`.

[9] Fisher, R. et al. *Digital Filters* [online]. 2003. [cit. 2021-04-09]. Dostupné z: `https://homepages.inf.ed.ac.uk/rbf/HIPR2/featops.htm`.

[10] Fisher, R. et al. *Morphology* [online]. 2003. [cit. 2021-04-08]. Dostupné z: `https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm`.

[11] Gonzalez, R. C. – Woods, R. E. *Digital image processing*. Parson, 2008.

[12] Google. *Tesseract OCR* [online]. [cit. 2021-04-24]. Dostupné z: `https://opensource.google/projects/tesseract`.

[13] Google. *Vision AI | Derive Image Insights via ML | Cloud Vision API* [online]. Google. [cit. 2021-04-26]. Dostupné z: https://cloud.google.com/vision.

[14] Han, T. – Hickman, A. *Our Search for the Best OCR Tool, and What We Found* [online]. opennews.org, Feb 2019. [cit. 2021-04-26]. Dostupné z: https://source.opennews.org/articles/so-many-ocr-options/.

[15] Haralick, R. M. – Sternberg, S. R. – Zhuang, X. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*. 1987, , 4, s. 532–550.

[16] Initiative, F. A. D. G. *Term: Thresholding* [online]. 2021. [cit. 2021-04-05]. Dostupné z: http://www.digitizationguidelines.gov/term.php?term=thresholding.

[17] Jähne, B. – Haussecker, H. – Geissler, P. *Handbook of computer vision and applications*. 2. Citeseer, 1999.

[18] matuska. [online]. Pixabay, Nov 2016. [cit. 2021-04-06]. Dostupné z: https://pixabay.com/photos/book-pages-background-open-book-1802861/.

[19] Microsoft. *Computer Vision* [online]. [cit. 2021-04-26]. Dostupné z: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision.

[20] Mittagessen. *GitHub Kraken* [online]. [cit. 2021-04-25]. Dostupné z: https://github.com/mittagessen/kraken.

[21] Mittagessen. *Kraken* [online]. 2015. [cit. 2021-04-25]. Dostupné z: http://kraken.re/.

[22] N, S. – S, V. Image Segmentation By Using Thresholding Techniques For Medical Images. *Computer Science & Engineering: An International Journal*. 02 2016, 6, s. 1–13. doi: 10.5121/cseij.2016.6101.

[23] Niblack, W. *An introduction to image processing (pp. 115–116)* [online]. Prentice-Hall, Englewood Cliffs, NJ, 1986. [cit. 2021-04-07].

[24] OpenCV. *Canny Edge Detection* [online]. [cit. 2021-04-10]. Dostupné z: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html.

[25] OpenCV. *Contours* [online]. [cit. 2021-04-10]. Dostupné z: https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html.

[26] OpenCV. *Structural Analysis and Shape Descriptors* [online]. [cit. 2021-04-11]. Dostupné z: https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gaedef8c7340499ca391d459122e51bef5.

[27] OpenCV. *Image Filtering, OpenCV docs* [online]. 2021. [cit. 2021-04-06]. Dostupné z: https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html.

[28] OpenCV. *Image Thresholding, OpenCV docs* [online]. 2021. [cit. 2021-04-05]. Dostupné z: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html.

[29] OpenCV. *Sobel Derivatives* [online]. [cit. 2021-04-10]. Dostupné z: https://docs.opencv.org/master/d2/d2c/tutorial_sobel_derivatives.html.

[30] Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, 9, 1, s. 62–66. doi: 10.1109/TSMC.1979.4310076.

[31] Sahoo, P. K. – Soltani, S. – Wong, A. K. A survey of thresholding techniques. *Computer vision, graphics, and image processing*. 1988, 41, 2, s. 233–260.

[32] Sauvola, J. – PietikÄinen, M. Adaptive document image binarization. *Pattern Recognition*. 2000, 33, 2, s. 225–236. ISSN 0031-3203. doi: https://doi.org/10.1016/S0031-3203(99)00055-2. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0031320399000552.

[33] Schulenburg, J. *GOCR OCR* [online]. [cit. 2021-04-24]. Dostupné z: http://jocr.sourceforge.net/.

[34] Shapiro, L. G. – Stockman, G. C. *Computer vision*. Prentice Hall, 2001.

[35] Singh, A. – Bacchuwar, K. – Bhasin, A. A survey of OCR applications. *International Journal of Machine Learning and Computing*. 2012, 2, 3, s. 314.

[36] Singh, T. R. et al. A new local adaptive thresholding technique in binarization. *arXiv preprint arXiv:1201.5227*. 2012.

[37] Smith, R. Tesseract ocr engine. *Lecture. Google Code. Google Inc*. 2007.

[38] Sucky, R. N. *A Complete Understanding of Precision, Recall, and F Score Concepts* [online]. Towards Data Science, Oct 2020. [cit. 2021-04-26]. Dostupné z: https://towardsdatascience.com/a-complete-understanding-of_precision-recall-and-f-score-concepts-23dc44defef6.

[39] Szeliski, R. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[40] Tesseract. *GitHub tesseract-ocr/tesseract* [online]. [cit. 2021-04-23]. Dostupné z: `https://github.com/tesseract-ocr/tesseract`.

[41] Vincent, O. R. – Folorunso, O. – others. A descriptive algorithm for sobel image edge detection. In *Proceedings of informing science & IT education conference (InSITE)*, 40, s. 97–107. Informing Science Institute California, 2009.

[42] Wood, T. *F-Score* [online]. May 2019. [cit. 2021-04-26]. Dostupné z: `https://deepai.org/machine-learning-glossary-and-terms/f-score`.

[43] Zub, J. – Diaz, P. – Stonehouse, S. Marvel's Avengers: Iron Man #1. 2019, 1, s. 7–7.

[44] Dinç et al. Chapter 12 - DT-Binarize: A decision tree based binarization for protein crystal images. In Deligiannidis, L. – Arabnia, H. R. (Ed.) *Emerging Trends in Image Processing, Computer Vision and Pattern Recognition*. Boston: Morgan Kaufmann, 2015. s. 183–199. doi: https://doi.org/10.1016/B978-0-12-802045-6.00012-0. Dostupné z: `https://www.sciencedirect.com/science/article/pii/B9780128020456000120`. ISBN 978-0-12-802045-6.

A Uživatelská dokumentace

Na přiloženém DVD se ve složce src nachází zdrojové kódy programu v jazyce Python a soubor requirements.txt obsahující seznam knihoven potřebných k jeho spuštění. Program byl vyvíjen a testován na operačním systému Ubuntu verze 20.10 za použití Pythonu verze 3.8.5.

Ke spuštění programu slouží skript `main.py`. Při zavolání skritpu s parametrem `--help` (tedy například `python main.py --help`) program vypíše nápovědu k jeho použití.

Pro spuštění programu v režimu tvorby datasetu spusťte skript main.py s parametry `<cesta k obrázku se stránkou komixu> <cesta pro vytvoření výstupní masky> <cesta pro uložení výstupu ve formátu json>`, tedy například
`python main.py comic-page.jpg output-mask.jpg output-results.txt`

volitelné parametry:

`--verbose`	zobrazení průběh zpracování do konzole
`--save-bubbles` <složka>	zajistí uložení jednotlivých bublin jako samostatných obrázků do specifikované složky

Úpravu parametrů používaných pro segmentaci a prahování je možné provést v souboru `config.py`.

B Výsledky měření

Tabulka B.1 obsahuje naměřená data úspěšnosti detekce bublin pro každou z 24 komixových stran. Celkový počet bublin na těchto stranách je 148.

	Pixelové metriky		**Bublinové metriky**					
Stránka	**F1**	**Jaccard idx**	**TP**	**FP**	**FN**	**Prec.**	**Rec.**	**F1**
1	0,995	0,990	7	0	0	1,000	1,000	1,000
2	0,995	0,990	6	0	0	1,000	1,000	1,000
3	0,995	0,990	4	0	0	1,000	1,000	1,000
4	0,994	0,989	4	0	0	1,000	1,000	1,000
5	0,994	0,989	3	0	0	1,000	1,000	1,000
6	0,994	0,989	4	0	0	1,000	1,000	1,000
7	0,994	0,989	4	0	0	1,000	1,000	1,000
8	0,994	0,989	10	0	0	1,000	1,000	1,000
9	0,994	0,988	5	0	0	1,000	1,000	1,000
10	0,993	0,987	9	0	0	1,000	1,000	1,000
11	0,993	0,986	8	0	0	1,000	1,000	1,000
12	0,993	0,986	9	0	0	1,000	1,000	1,000
13	0,992	0,985	7	0	0	1,000	1,000	1,000
14	0,988	0,977	10	0	1	1,000	0,909	0,952
15	0,974	0,950	8	0	1	1,000	0,889	0,941
16	0,970	0,943	6	1	0	0,857	1,000	0,923
17	0,968	0,939	3	1	0	0,750	1,000	0,857
18	0,952	0,912	3	2	0	0,600	1,000	0,750
19	0,905	0,840	4	1	1	0,800	0,800	0,800
20	0,905	0,838	6	1	2	0,857	0,750	0,800
21	0,843	0,757	5	1	1	0,833	0,833	0,833
22	0,798	0,710	1	1	0	0,500	1,000	0,667
23	0,733	0,641	2	2	0	0,500	1,000	0,667
24	0,563	0,521	1	3	0	0,250	1,000	0,400

Tabulka B.1: Výsledky měření úspěšnosti detekce bublin

Stránka	WER	CER
1	0,068	0,028
2	0,378	0,159
3	0,154	0,062
4	0,182	0,095
5	0,136	0,058
6	0,116	0,038
7	0,017	0,007
8	0,038	0,013
9	0,313	0,297
10	0,086	0,038
11	0,054	0,031
12	0,429	0,276
13	0,372	0,314
14	0,202	0,124
15	0,194	0,192
16	0,194	0,150
17	0,061	0,019
18	0,140	0,119
19	0,222	0,184
20	0,167	0,082
21	0,036	0,052
22	0,083	0,027
23	0,345	0,304
24	0,000	0,006

Tabulka B.2: Výsledky měření úspěšnosti rozpoznání textu v bublinách

C Vizualizace výsledků

Obrázek C.1: Vizualizace stránky, kde byly správně detekovány všechny komixové bubliny

Obrázek C.2: Vizualizace stránky, kde došlo k detekci 2 FP (červená) a 1 FN (tyrkysová)