GYMNASIUM JANA KEPLERA

Parléřova 2/118, 169 00 Praha 6

Rozpoznávání knih podle obálek

Maturitní práce

Autor: Verner Benjamin Vincent

Třída: R8.A

Školní rok: 2020/2021

Předmět: Informatika

Vedoucí práce: Šimon Schierreich

Praha, 2021

GYMNASIUM JANA KEPLERA
Kabinet informatiky

ZADÁNÍ MATURITNÍ PRÁCE

Student:	**Benjamin Vincent Verner**
Třída:	**R8.A**
Školní rok:	**2020/2021**
Platnost zadání:	**30. 9. 2021**
Vedoucí práce:	**Šimon Schierreich**
Název práce:	**Rozpoznávání knih podle obálek**

Pokyny pro vypracování:

Vytvořte aplikaci, která bude umět na zadaných fotografiích rozpoznávat knihy podle jejich obalů a dalších informacích uvedených na obálce. Pro rozpoznanou knihu by měla aplikace také nalézt a zobrazit dodatečné informace, jako je například ISBN, a na základě nich generovat citaci ve formátu BibTeX.

Doporučená literatura:

[1] BUNKE, Horst a Patrick S. P. WANG, ed. Handbook of Character Recognition and Document Image Analysis. Singapur: World Scientific, 1997. ISBN 978-981-02-2270-3. Dostupné z: doi:10.1142/2757.

[3] MARKEY, Nicolas. Tame the BeaST: The B to X of BibTeX. 2009. Dostupné z: https://mirrors.nic.cz/tex-archive/info/bibtex/tamethebeast/ttb_en.pdf.

[2] FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley Professional, 2003. ISBN 978-0-321-12742-6.

URL repozitáře:

https://github.com/Benyeemin/maturitni_projekt_Benjamin_Verner

vedoucí práce *student*

V Praze dne 30. 10. 2020

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Nemám žádné námitky proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 7. dubna 2021 Verner Benjamin Vincent

Abstrakt

Tato práce popisuje implementaci a teoretické základy aplikace *Knihovník*, která generuje bibliografické údaje o knihách na fotkách. Aplikace detekuje knihy na obrázku pomocí neuronové sítě YOLOv4 a pro detekci textu používá program Tesseract. Knihy vyhledává v databázi Open Library Books a bibliografické údaje generuje pomocí isbntools. Při vývoji aplikace se ukázaly některé části jako problematické a bylo by potřeba je dále vylepšit. Aplikace *Knihovník* zároveň poskytuje grafické uživatelské rozhraní založené na knihovně Kivy.

Klíčová slova

klasifikace obrázků, konvoluční neuronové sítě, automatické rozpoznávání textu, OpenCV, Tesseract, Yolov

Abstract

This work describes the implementation and theoretical basics of the Knihovník application, which generates bibliographic data about books in photos. The application detects books in the image using the YOLOv4 neural network and uses the Tesseract program to detect text. It searches for books in the Open Library Books database and generates bibliographic data using isbntools. During the development of the application, some parts proved to be problematic and would need to be further improved. Knihovník also provides a graphical user interface based on the Kivy library.

Keywords

Image Classification, Convolutional Neural Networks (CNN), Optical Character Recognition (OCR), OpenCV, Tesseract, Yolov

Obsah

1. Teoretická část

Cílem tohoto projektu bylo vytvořit aplikaci, která by ze zadaného obrázku uměla rozpoznat knihy, a pro každou z nich podle údajů na obálce získat název a dohledat ISBN, případně další údaje potřebné k vygenerování BibTex citace. Řešení tohoto problému lze rozdělit na následující kroky: Hledání kontur a rozpoznání knih na zadaném obrázku, získání informací na obálce knihy pomocí automatického rozpoznávání textu, a nakonec určení titulu pomocí získaných informací z obálky.

1.1 Hledání kontur

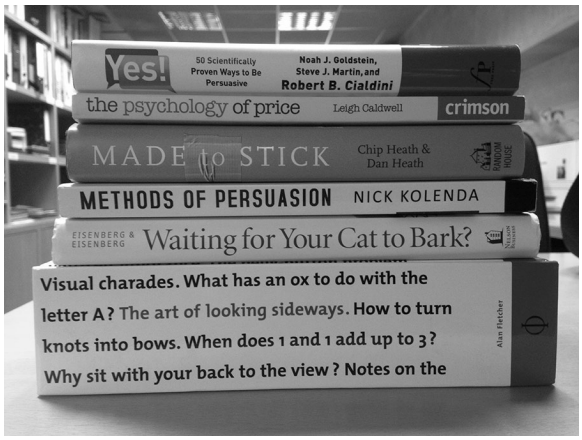
OpenCV V první fázi projektu jsem pro hledání kontur používal algoritmus knihovny OpenCV ([6]). Tento algoritmus pracuje s grayscale obrazem (tedy obrazem v odstínech šedé), ve kterém vyhledává změny intenzity, a dává za výsledek množinu mnohoúhelníků, které ohraničují jednotlivé oblasti se stejnou intenzitou — objekty. Abychom mohli tuto metodu použít, musíme zadaný obrázek nejprve převést do grayscale formátu, a poté provést Gaussovo rozostření, což je metoda, která se používá pro zredukování detailů a šumu. Pak můžeme aplikovat tzv. Canny Edge Detector, který na zadaném obrázku pomocí Sobel kernelu spočítá pro každý pixel směrovou změnu intenzity, a dostatečně velké hodnoty označí jako hrany objektu. Pro každý objekt tímto dostaneme mnohoúhelník, sestávající se z pixelů na jeho okraji. Na těchto mnohoúhelnících následně provedeme aproximaci na jednodušší tvary, a zaměříme se pouze na čtyřúhelníky — tyto objekty pravděpodobně reprezentují knihy.

V praxi se však algoritmus ukázal jako nevyhovující, experimentální výsledky ukázaly jeho chybovost. Na obrázku 1.1, je vidět typický výsledek, z něhož je patrná nepoužitelnost tohoto algoritmu.

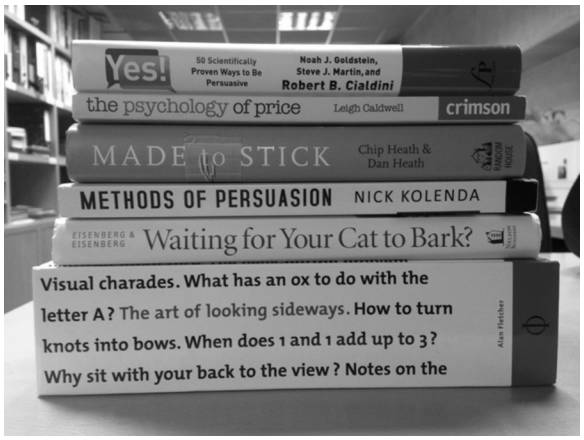
Konvoluční neuronové sítě (CNN) Druhá metoda pro hledání knih v obrázku je za pomoci neuronové sítě. Konvoluční neuronové sítě (viz např. článek na Wikipedii) sestávají z vstupní vrstvy, jedné či více skrytých vrstev a z výstupní vrstvy. Používají se pro rozpoznání struktury, jako například hran či tvarů. Pro tento projekt jsem zvolil model YOLOv4 ([1]), napsaný ve frameworku Darknet ([3]). Model se skládá ze tří částí. V první části (Backbone) dochází ke zpracování obrázku a rozpoznání hlavních rysů objektů pomocí několika filtrů. Poté je obrázek poslán do druhé části (Neck), která obrázek musí zmenšit na požadované rozměry, aniž by poškodila objekty na obrázku. Využívá k tomu mapy rozpoznaných rysů generovaných v předchozím kroku. Poslední část (Head) pak na daném obrázku provede predikci klasifikace objektů, která je společně s boxy ohraničujícími rozpoznané objekty výstupem modelu.

1.2 Automatické rozpoznávání textu - Tesseract

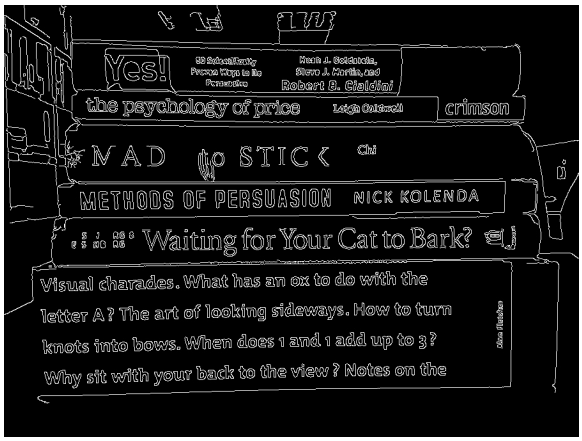
Zde přichází na řadu získání informací z jednotlivých knih. K tomu je potřeba OCR engine, který dokáže rozpoznat všechen text na obalu knihy. V tomto projektu je použit Tesseract OCR ([4, 5]), což je nejlepší open source OCR engine.



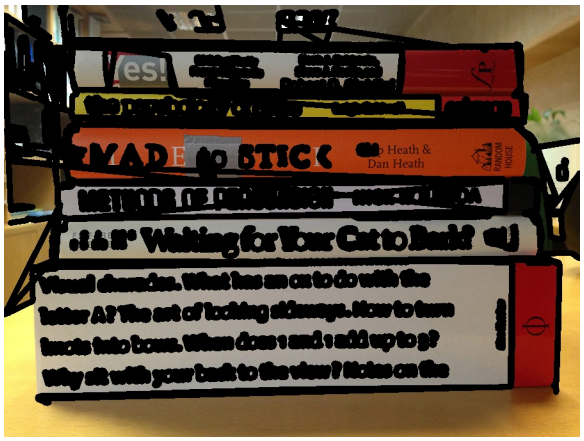
(a) Původní obrázek (převedený do odstínů šedi)



(b) Obrázek po aplikaci Gaussovského rozostření



(c) Hrany získané algoritmem Canny



(d) Zjednodušenné hrany zobrazené na původním barevném obrázku

Obrázek 1.1: Ukázka jednotlivých kroků
při zpracování pomocí OpenCV

Před jeho použitím je potřeba původní obrázek převést na binární (tj. černobílý), jinak se jeho účinnost radikálně sníží. Struktura samotného rozpoznávacího systému se skládá ze dvou částí. V první části se pro každý znak vygeneruje sada tříd, kam by tento znak potenciálně mohl patřit. Z této sady se další analýzou vybere ta, která nejvíce odpovídá. V druhé části přichází na řadu tzv. adaptivní rozpoznání (adaptive recognition), které používá písmena rozpoznaná s velkou přesností v první části pro lepší rozpoznání zbylé části textu. Tato metoda je zásadní při práci s textem odlišným od dat použitých na trénování, tedy například při rozpoznávání neobvyklých fontů.

1.3 Prohledávání databáze

Poslední fáze spočívá v propojení textu rozpoznaného Tesseractem s odpovídající knihou. K tomu potřebujeme databázi všech knih a autorů. Tuto databázi budeme prohledávat, a knihu, která obsahuje nejvíce z rozpoznaných slov, pak můžeme považovat za knihu na obrázku. K ní pak můžeme vyhledat ISBN a další údaje potřebné k vygenerování citace ve formátu BibTex.

2. Implementace

Realizace projektu probíhala v programovacím jazyce Python za použití potřebných knihoven. Volba tohoto jazyka spočívala v jeho jednoduchosti a vhodnosti pro rychlou iteraci.

2.1 Struktura kódu

UI rozhraní (`application_ui.py`) Uživatelské rozhraní aplikace. Je tu přes kivy builder načítána struktura aplikace. Jednotlivé třídy představují obrazovky, Screen Manager přechází mezi MainScreen a FinalScreen. Uvnitř tříd jsou definovány funkce, které se volají při stisku tlačítek. Hlavní funkce, která postupuje přesně podle postupu popsaného v teoretické části, je funkce `read_from_image()`. Uvnitř ní jsou volány funkce z ostatních souborů.

```
    def read_from_image(self):
        FinalScreen.output = []
        FinalScreen.book_number = 0
        # detection using OpenCV:
        #   text_list = image_processing.find_books(MainScreen.img_path)
        text_list = detect.find_books(MainScreen.img_path)
        if text_list is not None:
            for text in text_list:
                if len(text) > 0:
                    FinalScreen.book_number += 1
                    book_title, book_author = sqlsearch.search(text)
                    if book_title is not None:
                        string = information_fetcher.info_from_image(
                            book_title, book_author, FinalScreen.book_number
                        )
                        FinalScreen.output.append('\n' + string)
                    else:
                        FinalScreen.output.append(
                            'Server connection error: check your internet connection.'
                        )
                        self.final_screen.ids.output.text = FinalScreen.output[0]
                        return
            if FinalScreen.book_number == 0:
                FinalScreen.output.append('0 books found.')
        else:
            FinalScreen.output.append('Invalid file.')
        self.final_screen.ids.output.text = FinalScreen.output[0]
```

Na začátku se resetuje výstup pro případ, že je funkce zavolána poněkolikáté. Poté se předává cesta k obrázku funkci uvnitř `detect.py` (5), ve které proběhne rozpoznání knih na obrázku, a pro každou takto rozpoznanou knihu rozpoznání textu. Výstupem je pro každou knihu list slov. Ten

dále předáváme funkci `sqlsearch.search()` (10), která má za úkol spojit list s knihou v databázi a vrátit přesný titul a autora knihy.

Poslední krok je na funkci uvnitř `information_fetcher`, která vyhledá isbn knihy a připraví finální výstup, který se zobrazí na obrazovce.

Hledání kontur — Neuronové Sítě (`detect.py`) V této části se pracuje s neuronovou sítí. Načteme YOLOv4 model, kterému předložíme obrázek načtený ze vstupu. Každou detekovanou knihu následně ořízneme jako samostatný obrázek, převedeme ho do binárního obrázku, a aplikujeme funkci knihovny Pytesseract. Výstupem je pro každou knihu list slov rozpoznaných Tesseractem.

Hledání čárových kódů (`barcode_finder.py`) Soubor určený pro hledání čárových kódů z obrázku. Funkce za vstup dostane cestu k obrázku, ten načte pomocí OpenCV, a pomocí knihovny Pyzbar najde všechny čárové kódy knih na obrázku. Výstupem pak je list jednotlivých ISBN kódů.

Hledání názvu v SQL databázi (`method1_SQL.py`) Zde se nachází funkce určená ke komunikaci s databází. Jako vstup dostane zadaný list slov, ze kterých pak formuluje SQL dotazy. Databáze je převzaná z Open Library Data Dumps ([2]). Informace z ní byly zpracovány do SQL databáze, jejíž podobu lze vidět zde:

```
-- Tabulka děl
create table if not exists works(
    title text,
    "key" text,
    main_author text
);

-- Tabulka spojující díla a autory
create table if not exists work_auths(
    work_key text,
    auth_key text
);

-- Tabulka autorů
create table if not exists authors(
    name text,
    "key" text,
);
```

Důležité informace jsou pro nás název knihy a jméno autora. Všechny dotazy mají následující formu - pro každé slovo projdeme klíčový sloupec (title či name), a všem knihám či autorům, kde se toto slovo nachází, zvýšíme prioritu o 1. Poté, co projdeme všechna slova, vybereme knihu či autora s nejvyšší prioritou. V jazyku SQL to vypadá tak, že v rámci příkazu vytvoříme další sloupec 'words',

do kterého započítáváme hodnotu priority, a to součtem nul a jedniček za každé slovo. Pro ilustraci příklad dotazu pro dvě slova „lord“ a „rings“[1]:

```
select
    title,
    case
        when title like '%lord%'
        then 1 else 0
    end
     +
    case
        when title like '%rings%'
        then 1 else 0
    end as words
from (
    select
        title
    from
        works
    where
        title ilike '%rings%'
    union all
    select
        title
    from
        works
    where
        title ilike '%lord%'
) order by
    words desc
limit 1;
```

Vzhledem k tomu, že nevíme, jaká slova z listu jsou z titulu knihy a jaká jméno autora, musíme takto hledat vždy všechna slova, i když některá budou vždy v daném kontextu nesmyslná. Následující funkce konstruuje case a select pro každé slovo v listu:

```
1  def build_search_sql(text, column, table):
2      cases = [f"case when {column} ilike '%{word}%' then 1 else 0 end" for word in text]
3      union = [f"select * from {table} where {column} ilike '%{word}%'" for word in text]
4      return [' + '.join(cases), ' union all '.join(union)]
```

Nejdříve tímto způsobem prohledáme autory, přičemž získáme „key“ předpokládaného autora. V listu zadaných slov ale nemusí nutně být jméno autora, a tak všem knihám od tohoto autora dáme prioritní hodnotu rovnou třetině počtu všech slov (pro méně než 3 slova ale hodnotu 1). Poté

[1] Při testování se ukázalo, že `UNION` je výrazně rychlejší než `OR`, protože využívá index. Není úplně jasné, proč `OR` index nevyužívá.

obdobným způsobem prohledáme knihy, přičemž knihu s největší prioritou označíme jako hledanou knihu, a na výstup vrátíme její název a jméno autora.

V samotném kódu SQL command obsahuje ještě kontrolu délky titulu. Je to kvůli tomu, že převzatá databáze obsahuje mnoho dlouhých titulů, které kazí výsledky (viz 3.3). Tato kontrola je snaha omezení tohoto problému.

Online hledání v ISBN databázi (`information_fetcher.py`) V tomto kódu probíhá hledání ISBN a generování BibTex citace. Veškerou práci dělá knihovna isbnlib, která podle titulu knihy dokáže vyhledat její ISBN, a v dalším kroku pro dané ISBN najde veškeré potřebné informace pro citaci. Ty pak už stačí jen zkompletovat do výstupu.

Nepoužitý kód

Hledání kontur — OpenCV (`image_processing.py`) Od tohoto kódu bylo upuštěno přechodem z OpenCV algoritmu na vyhledávání kontur na neuronové sítě. Kód je určen ke stejnému účelu jako kód v `decode.py`, používá algoritmus popsaný v teoretické části.

Fulltextové vyhledávání v SQL databázi (`method2_fulltext_search.py`) Pokus o fulltextové vyhledávání, na které jsem přešel kvůli problému s rychlostí SQL dotazu. Kvůli napojení na korpus však takto nešlo vyhledávat jména autorů, a po vyřešení problému s rychlostí (viz sekce 3.3) nebyl důvod fulltextové vyhledávání dále používat. Proto kód není dokončený.

2.2 Použité knihovny

Kivy `https://kivy.org`
Licence: MIT `pip install kivy`
Kivy je knihovna pro Python určená pro vývoj uživatelského rozhraní aplikací. Její výhodou je multiplatformní využití. Lze použít i pro Android, i když to se v tomto případě ukázalo být příliš komplikované kvůli ostatním použitým knihovnám.

OpenCV `https://opencv.org/`
Licence: Apache v2.0 `pip install opencv-python`
Knihovna na image processing. Veškerá manipulace s obrázkem je v kódu prováděna prostřednictvím této knihovny.

Tesseract `https://github.com/tesseract-ocr/tesseract`
Licence: Apache v2.0 `pip install pytesseract`
Jak už bylo zmíněno v teoretické části (1.2), Tesseract je knihovna na rozpoznávání textu (OCR). Její použití přímo v kódu je jednoduché - pouze na jeden řádek, pokud nepočítáme úpravu obrázku, kterou předtím musíme provést.

ISBN Tools & Library `https://github.com/xlcnd/isbnlib`
Licence: LGPL v3 or later `pip install isbntools isbnlib`

Toto jsou knihovny pro práci s ISBN. K vyhledání informací o dané knize využívají službu Google Books. Ukázaly se jako ideální řešení problému vyhledávání ISBN a citací k najitým knihám.

ZBar http://zbar.sourceforge.net/
Licence: LGPL v2.1 `pip install pyzbar`
Pyzbar je knihovna na vyhledání čárových kódů knih z obrázku, výstupem je ISBN dané knihy.

TensorFlow https://www.tensorflow.org/
Licence: Apache v2.0 `pip install tensorflow`
Knihovna pro strojové učení. Používá se na trénování neuronových sítí. Pomocí ní se v projektu načítá YOLOv4 model.

Psycopg https://www.psycopg.org/
Licence: LGPL v3 with OpenSSL exception `pip install psycopg2`
Pythonovská knihovna pro práci s PostgreSQL databází. Pomocí této knihovny se aplikace připojuje k databázi na serveru a komunikuje s ní.

2.3 Problémy

Nedostatečnost OpenCV při hledání kontur V relativně pokročilé fázi projektu jsem byl nucen změnit technologii vyhledávání kontur z OpenCV na neuronové sítě, protože praktické ukázky algoritmu ukázaly na jeho nedostatečnost v rozpoznání samotných knih. Vzhledem k nedostatku času jsem proto použil již natrénovaný model YOLOv4.

Problémy s Tesseractem na Androidu Další problém, který bylo nutno řešit nedlouho před odevzdáním, byl problém dost významný. Ve fázi přípravy packagingu aplikace pro Android jsem zjistil, že použití Tesseractu na Androidu má dost zásadní problém. Knihovna Pytesseract je závislá na programu Tesseract, který na Androidu není k dispozici.

Proto bylo potřeba najít alternativu k Tesseractu dostupnou na Androidu. Jako dobrá alternativa se ukázal ML Kit, který je dokonce zaměřen na čtení textu přímo z barevných fotek, ne z binárních obrázků. Mohla by se tedy výrazně snížit chybovost. Protože však nebylo dostatek času na testování a implementaci (která by nebyla tak snadná, protože ML Kit nemá asociovanou Pythonovskou knihovnu), vzdal jsem se pokusů o převedení aplikace na Android.

Velikost Tesseractu Obecně velikost celé aplikace byla předmětem mého zájmu ve chvíli, kdy jsem počítal, že výsledkem projektu bude aplikace zaměřená na Android. I poté jsem se snažil její velikost co nejvíce zredukovat, aby nebyla tak náročná na instalaci. Vzhledem k tomu, že Tesseract dohromady zabíral kolem 900 Mb, pracoval jsem na možnostech jeho zmenšení. Velkou část tohoto prostoru zabírá CUDA architektura, která funguje pouze na GPU od NVIDIA. Bez toho je možné se obejít, a sníží se tak velikost na cca 700 Mb.

Použitelnost Tensorflow pouze na 64 bitových platformách Při použití 32 bitového Pythonu jsem narazil na problém, kterému jsem měl potíž porozumět, protože chybové hlášky při instalaci

Tensorflow v tomto prostředí nebyly příliš návodné. Ukázalo se, že Tensorflow je k dispozici pouze pro 64 bitové platformy.

Kivy podpora kamery na linuxu [2] Při testování na Linuxu jsme narazili na chybu v knihovně Kivy. Chyba se projevovala pádem programu s následující hláškou:

```
VIDEOIO ERROR: V4L: can't open camera by index 0
```

Hledání na StackOverflow naznačovalo chybu s přístupovými právy, ale to se ukázalo jako slepá ulička. Když se program pustil s podrobnějším logováním knihovny opencv:

```
$ OPENCV_LOG_LEVEL=DEBUG python application_ui.py
```

ukázalo se, že se pokouší dvakrát po sobě přistupovat k souboru `/dev/video0`, přičemž druhý pokus selže s chybou (`EBUSY`). Dále jsme zjistili, že knihovna Kivy se pokouší dvakrát inicializovat kamerový widget přičemž pokždé otevírá `/dev/video0`.

Řešením bylo upravit soubor `kivy/core/camera/camera_opencv.py` tak, aby se `/dev/video0` otevíralo pouze jednou. Výsledná změna je uložená v souboru `kivy-opencv2.diff` a pro běh na linuxu je nutno tuto změnu provést při instalaci ze zdrojových kódů (viz dále sekce 3.1, ostavec „Instalace ze zdrojových kódů“).

Rychlost zpracování SQL dotazu I po indexování vytvořené SQL databáze byl značný problém v rychlosti zpracování jednotlivých SQL dotazů. Délka tohoto procesu byla v řádu minut, což rozhodně není ideální. Zároveň byl problém i ve velikosti databáze. Proto jsem se problém pokusil vyřešit tak, že jsem požádal bratra o přesun databáze na server, kam nyní posílám dotazy a odkud pak přijímám data. Dokud vše běželo na jeho domácím rychlém stroji, bylo zpracování dotazů velmi rychlé. Protože však jeho stroj nemá veřejně dostupnou IP adresu, přesunul databázi do cloudu. Cloudový stroj však je základní stroj s pomalými disky a relativně malou pamětí (2 Gb), a rychlost dotazů je i po optimalizaci SQL kódu neideální (složitější dotazy mohou zabrat i 10 minut). Odpadá však problém velikosti databáze, protože ta je uložena v cloudu.

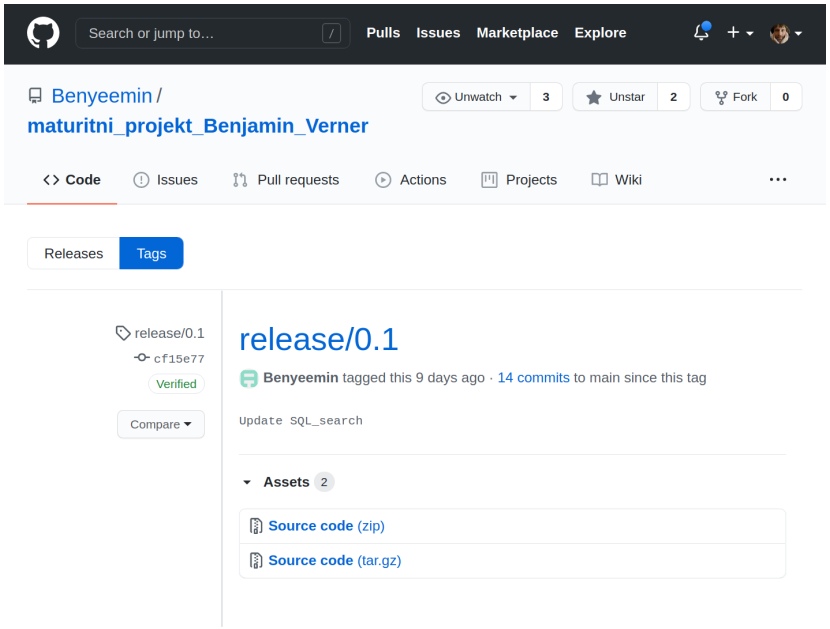
[2] Tento problém jsme řešili společně s mým bratrem, který má větší zkušenosti s OS Linux. Řešení i následující popis je výsledkem naší spolupráce.

3. Technická dokumentace

Aplikace *Knihovník* umožňuje analyzovat fotky a zobrazit bibliografické údaje o knihách, které se na fotce nacházejí. Aplikace bibliografické údaje získává z on-line databáze bibliografických kódů buď pomocí automatického rozpoznávání názvu knihy a jejího autora, nebo na základě čárových kódů obsahujících ISBN kód knihy.

3.1 Instalace

Instalace vydaného balíčku (Windows) Ze stránek projektu na Github (viz 3.1) si lze stáhnout připravený zip soubor. Po rozbalení stačí spustit program *knihovnik.exe* z adresáře *knihovnik*.



Obrázek 3.1: Webová stránka s vydáním

Instalace ze zdrojových kódů Nejdřív je třeba naklonovat zdrojové kódy:

```
$ git clone https://github.com/Benyeemin/maturitni_projekt_Benjamin_Verner
```

Pro správný chod aplikace je nutné nejprve nainstalovat program tesseract. Např. na Ubuntu lze využít balíček `tesseract-ocr`, na Windows je možné instalovat pomocí Chocolatey:

```
PS C:\> choco install -y tesseract
```

Na unixových systémech je třeba ještě nainstalovat některé knihovny (které nejsou obsaženy v binárních pypi balíčcích pro linux):

- knihovna zbar
- program xclip

Na Ubuntu lze použít následující příkaz:

```
$ sudo apt install libzbar0 xclip  tesseract-ocr
```

Pak je třeba vytvořit virtuální Python prostředí a nainstalovat potřebné balíčky, na kterých aplikace závisí:

```
$ python3 -m virtualenv venv
$ . venv/bin/activate
(venv) $ pip install -r requirements.txt
```

Knihovna Kivy obsahovala v době vydání chybu, která znemožňovala práci s kamerou. Pokud program nebude fungovat (spadne), je možné tuto chybu vyřešit aplikací záplaty[1]:

```
$ patch --strip 1 < kivy-openc2.diff
```

Dále je třeba vytvořit soubor `server_connection_info` a uložit do něj přístupové údaje k SQL databázi obsahující seznam knih a autorů (viz 1.3 a odstavec „Hledání názvu v SQL databázi“ v sekci 2.1). V aktivovaném prostředí lze aplikaci spustit pomocí příkazu:

```
(venv) $ python application_ui.py
```

3.2 Použití

Po spuštění ukáže aplikace hlavní okno (viz 3.2), které sestává z části která zobrazuje kameru (1), a pěti tlačítek (2)-(6), které spustí jednu z následujících akcí

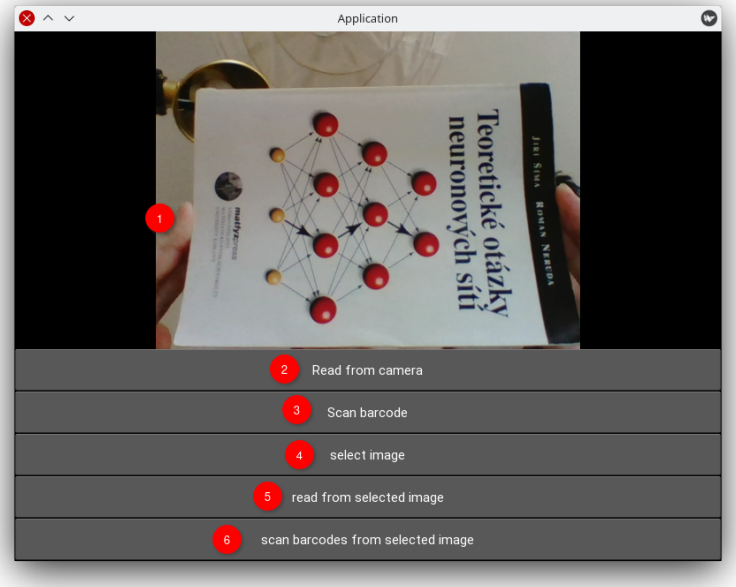
Zpracování obrázku z *kamery*

- po stisknutí tlačítka (2) kamera vyfotí jeden snímek, který program následně zpracuje a ukáže výsledek
- po stisknutí tlačítka (3) kamera vyfotí jeden snímek, ve kterém zpracuje všechny nalezené čárové kódy a ukáže výsledek

Zpracování obrázků *uložených na disku*

- po stisknutí tlačítka (4) se zobrazí dialog pro výběr souboru (viz 3.3), který umožní načíst soubor pro zpracování z disku (význam tlačítek na obrázku 3.3 je následující: tlačítko (1) načte zvolený soubor, tlačítko (2) zruší výber souboru, a jednoduchým kliknutím na (3) se vstoupí do adresáře resp. vybere soubor)
- stisknutím tlačítka (5) se spustí analýza obrázku vybraného pomocí tlačítka (4)
- stisknutím tlačítka (6) se spustí analýza čárových kódů na obrázku vybraném pomocí tlačítka (4)

[1] Problém diagnostikoval a záplatu vytvořil můj bratr.



Obrázek 3.2: Hlavní okno aplikace

Zobrazení výsledků

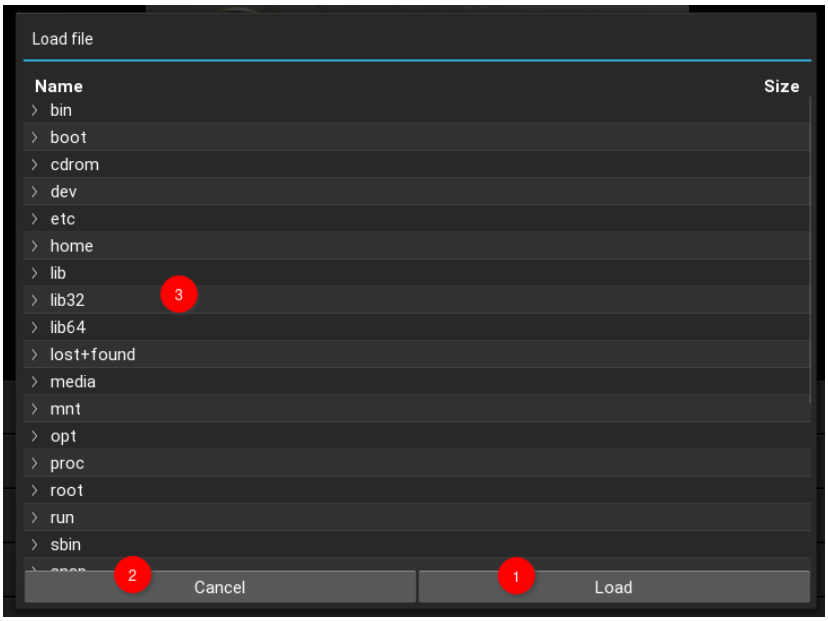
- po stisknutí tlačítek (5) nebo (6) se provede zpracování obrázků a výsledek se zobrazí v programu ve výsledkovém okně (viz 3.4), které sestává z částí pro zobrazení výsledků (1) a pěti tlačítek (2)-(6)
- tlačítka (5),(6) slouží k přecházení mezi výsledky (pokud je výsledků více)
- tlačítko (3) zkopíruje všechny výsledky do clipboardu
- tlačítkem (4) je možné se vrátit do hlavního okna aplikace

V případě zpracování obrázků uložených na disku program podporuje, kromě JPG a PNG formátů, všechny formáty podporované knihovnou OpenCV.

3.3 Známé problémy a omezení

Obecné

- Instalační balíček je příliš velký, což je způsobeno zejména použitím knihovny TensorFlow, která zabírá kolem 700 Mb. Velkou část zabírá také samotná neuronová síť. V budoucí verzi je plánováno přejít na knihovnu TensorFlow Lite a optimalizovat natrénovaný model tak, aby zabíral méně místa. V optimálním případě by bylo možné tímto způsobem snížit velikost instalačního balíčku na cca 100 Mb.
- Načítání YOLOv4 modelu trvá velmi dlouho, zároveň některé SQL commandy trvají více času kvůli jejich obsáhlosti. Celkový proces se tím z několika vteřin protáhne na několik minut.
- Program hledá pouze anglické tituly.
- Open Library Data Dumps obsahují zvláštní dlouhé tituly, kvůli kterým je výsledek velmi často chybný, protože místo správné chyby SQL dotaz nalezne dlouhý titul s vyšší prioritou. Snažím se tomu alespoň částečně zabránit omezením délky hledaného titulu.

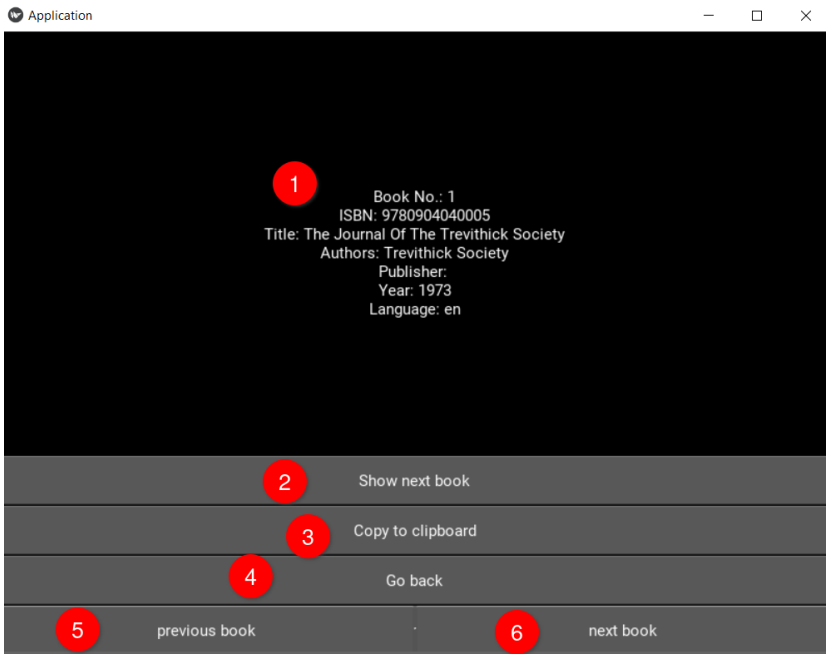


Obrázek 3.3: Dialog pro výběr souboru

Uživatelské rozhraní (UI)

- Pro výběr souborů se nepoužívají systémové dialogy.
- Při analýze obrazu program nedává zpětnou vazbu o probíhající operaci.
- UI není příliš uživatelsky přívětivé.

Většina těchto problémů je způsobena nedostatkem času.



Obrázek 3.4: Okno s výsledky

Závěr

V této fázi není aplikace úplně v použitelném stavu pro uživatele, a to hlavně kvůli času, který zabere zpracování obrázku. I přes to je projekt ve velmi pokročilé fázi, a vzhledem ke všem problémům, které v průběhu nastaly, jsem s výsledkem spokojený. Prací na projektu jsem se naučil pracovat s neuronovou sítí a OCR engine, metody image processingu, pracovat s SQL databází pomocí SQL příkazů, vytvářet uživatelské rozhraní desktopové aplikace, packaging aplikace pro různé platformy a mnoho dalšího.

Seznam použité literatury

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang a Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[Lib21] Open Library. *Data Dumps*. https://openlibrary.org/developers/dumps. Accessed: 2021-04-04. 2021.

[Red16] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

[Smi07] R. Smith. „An Overview of the Tesseract OCR Engine“. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Sv. 2. 2007, s. 629–633. DOI: 10.1109/ICDAR.2007.4376991.

[Smi13] Ray W. Smith. „History of the Tesseract OCR engine: what worked and what didn't“. In: *Document Recognition and Retrieval XX*. Ed. Richard Zanibbi a Bertrand Coüasnon. Sv. 8658. International Society for Optics a Photonics. SPIE, 2013, s. 1–12. DOI: 10.1117/12.2010051. URL: https://doi.org/10.1117/12.2010051.

[Suz+85] Satoshi Suzuki et al. „Topological structural analysis of digitized binary images by border following“. In: *Computer vision, graphics, and image processing* 30.1 (1985), s. 32–46.

Seznam obrázků