GYMNASIUM JANA KEPLERA

Parléřova 2/118, 169 00 Praha 6

Fotkovátor

Maturitní práce

Autor: Vlastimil Čejp

Třída: 4.A

Školní rok: 2022/2023

Předmět: Informatika

Vedoucí práce: Ing. Šimon Schierreich

Praha, 2023

GYMNASIUM JANA KEPLERA
Kabinet informatiky

ZADÁNÍ MATURITNÍ PRÁCE

Student:	**Vlastimil Čejp**
Třída:	**4. A**
Školní rok:	**2022/2023**
Vedoucí práce:	**Šimon Schierreich**
Název práce:	**Fotkovátor**

Pokyny pro vypracování:

Cílem práce je vytvořit systém na automatické štítkování fotek dle metadat i obsahu. Webová aplikace bude krom systému na štítkování obsahovat také uživatelské rozhraní pro vyhledávání ve fotkách a manuální úpravu, vytváření a přidávání štítků.

Doporučená literatura:

[1] SKAAR, Steffen. ***A Comprehensive Guide to Neural Network Modeling*****. Nova Science Publishing, 2020. ISBN: 978-1-536-18466-2**

[2] ESPOSITO, Dino. ***Modern Web Development*****. Microsoft Press, 2016. ISBN: 978-1-509-30001-3**

[3] LIU, Charles Z., and S. RAMAKRISHNAN. ***Image Recognition: Progress, Trends and Challenges*****. New York: Nova Science Publishers, 2020. ISBN: 978-1-536-17258-4**

URL repozitáře:

https://github.com/nesati/fotkovator

student *vedoucí práce*

V Praze dne 29. 9. 2022

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 24. března 2023 Vlastimil Čejp

Poděkování

Chtěl bych poděkovat vedoucímu práce, panu Ing. Šimonu Schierreichovi za jeho užitečné podněty, rady a připomínky k projektu. Dále bych chtěl poděkovat rodině a přátelům za podporu po celou dobu mého studia.

Abstrakt

Fotkovátor je modulární softwarový systém na automatické označování obrázků štítky napsaný v jazyce Python. Analyzuje metadata i obsah samotného obrazu.

Klíčová slova

fotografie, štítek, automatizované označování, strojové vidění

Abstract

Photo-inator is a modular software system for automated labeling of images with tags written in the Python language. It analyzes both metadata and the content image itself.

Keywords

photo, tag, automated labeling, machine vision

Obsah

1. Teoretická část

Tato práce popisuje, jak jsem řešil svůj problém prohledávání fotek, který věřím, že trápí i mnohé další. Nejprve jsem se snažil definovat problém, navrhnout co nejobecnější kostru řešení tak, aby ji bylo možno doplňovat a modifikovat podle budoucích potřeb nebo potřeb jiných uživatelů. V následujících kapitolách popisuji, jak celý projekt vznik a pak rozebírám detailně technické řešení, které jsem navrhl, naprogramoval a otestoval.

1.1 Motivace

Asi leckdo se setkal s problémem, že vidí obrázek před očima a ví, že ho má v galerii, ale nemůže ho najít. Mě se to děje každou chvíli. Potřebuji najít fotografii podle toho kdo nebo co na ní je, kde byla vyfocená atd. Takový dotaz se těžko zadá do standardního správce souborů. A také nechci svoje soukromé fotografie nahrávat do cloudových služeb, abych nepřišel o poslední zbytek svého digitálního soukromí.

Potřeboval jsem nástroj, který se dokáže zorientovat v mých lokálních souborech, aniž bych je musel s někým sdílet. Nevěděl jsem, jestli je to vůbec splnitelné zadání, přeci jen obdobné systémy nabízí jen velcí poskytovatelé s armádou programátorů, ale rozhodl jsem se, že díky open-source komunitě a pokrocích v moderních technologiích, musí být i v domácích podmínkách možné vytvořit nástroj, který pracuje zcela off-line a nabízí zmiňované funkce. Mám rád výzvy a řekl jsem si, že i když se to nepodaří zcela, bude to minimálně cenná zkušenost.

1.2 Cíle

Od začátku jsem si říkal, že analýzu obrázků bude možné stále zdokonalovat. Proto chci, aby systém byl navržen maximálně otevřený a modulární, aby základ, který vytvořím, mohli rozvíjet i další a mohl tak zpětně open-source komunitu obohatit.

Za cíl práce jsem si stanovil vytvořit minimum viable product, který jsem pracovně nazval Fotkovátor. Tedy program funkční sám o sobě k řešení daného problému, ale mířený současně jako inspirace pro vývoj dalších modulů a rozšiřování možností systému.

Základní požadavky zadání jsou:

- možnost skládat systém z modulů
- vyhledávání podle štítků
- plně automatické přidávání štítků podle
  - metadat
  - rozpoznávání obličejů
  - rozpoznání scény
- webové uživatelské rozhraní

2. Implementace

2.1 Jádro

Celý systém je napsaný s využitím async/await syntaxe vestavěné python knihovny `asyncio`. Volba pythonu vyplývá z toho, že tento jazyk umím, ale hlavně z toho, že je často využívaným na strojové učení a všechny projekty z této oblasti jde díky tomu implementovat nativně. Využití asyncia pak vychází z toho, že jsem se ho chtěl naučit používat. Dalším důvodem je, že bez paralelizace by byl projekt uživatelsky nepoužitelný (Grafické rozhraní by neodpovídalo v době, kdy se analyzují fotky.) a že spouštění modulů musí být nezávislé. Tyto poslední dva problémy by šlo vyřešit spouštěním modulů jako samostatné procesy, což by s sebou přineslo mnohé problémy jako IPC[1] a kódování obrázků.

Fotkovátor se skládá z jádra a modulů. Od začátku byl plán, aby jádro bylo minimální, tedy aby dělalo jen to, co opravdu musí a vše ostatní bylo v rukou konfigurovatelných modulů. Prací jádra je pouze načíst konfiguraci, spustit moduly, umožnit komunikaci mezi moduly pomocí událostí a přijímat signály systému o tom, že se má Fotkovátor vypnout. Další z cílů byl nepředepisovat kategorie modulů. I když jsou v projektu kategorizované, rozlišení je umělé, nikoli technické.

Jsou však dvě kategorie modulu, které se bez předepisování neobešly: databáze (`database`) a zdroj obrázků (`backend`). Protože pro systém nedává principiálně smysl mít více databází štítků, může být databáze jen jedna. Byť teoreticky nic nebrání tomu, aby sytém čerpal z více zdrojů obrázků (například telefon a server s dlouhodobějším úložištěm), kvůli zjednodušení a možnosti volání funkcí zdroje obrázků přímo jsem od této funkcionality upustil. Nic navíc nebrání tomu vytvořit modul fungující jako multiplexer spouštějící vícero modulů zdrojů obrázků.

Díky těmto dvěma ústupkům z jinak velmi obecného modulárního systému, lze do těchto modulů přistupovat přímo voláním funkcí, které umožňuje vracet data přirozeným způsobem. Přímé volání funkcí by mělo být používáno pouze pro čtení dat (například načtení obrázku pro zobrazení uživateli) a pro zápis by měly být využity události.

Systém událostí je vytvořený lehkou úpravou projektu `py-event-bus`[14]. Úprava spočívá v tom, že zatímco v originálním projektu se pro spuštění funkcí pouze vytvořila úloha, na kterou ale nikdy nebylo počkáno (*awaited*), po mé upravě se na dokončení úlohy čeká. To způsobí korektní vracení chyb a umožňuje modulům rozhodnout, jestli chtějí čekat na dokončení všech úkonů spojených s reakcí na událost.

Využití událostí místo klasického volání funkcí má několik výhod. Každý modul si může zaregistrovat, které události ho zajímají místo toho, aby každý modul musel vědět jaké metody konkrétních modulů má volat. Komunikace navíc může probíhat napříč moduly přesto, že na sebe nemají přímé reference.

Jeden ze zajímavých problémů na které jsem narazil je problematika identifikátoru fotek. Přirozený identifikátor fotkám přiděluje zdroj obrázků (například cestu nebo url k souboru). Nabízí se proto využití formátu URI[2], který by dokonce mohl odlišovat jednotlivé zdroje, ale použití řetězců místo

[1] Inter-process communication - komunikace mezi jednotlivými procesy
[2] Uniform Resource Identifier - formát identifikování abstraktních i fyzických zdrojů [3]

čísel na identifikaci v databázi je nepraktické.

Druhým nápadem bylo využití hashe obrázku a vytvořit tak efektivně hashovací tabulku. Problém však vzniká s dynamickou velikostí tabulky a řešením konfliktů.

Finálním řešením tedy je identifikátor v projektu nazývaný `uid`. Jde o celé číslo generované databází. Jediná implementovaná databáze, PostgreSQL, přiděluje `uid` sekvenčně a nevidím důvod, aby jiné databáze postupovaly jinak.

Při vývoji jsem sice používal SQL[3] databázi, ale snažil jsem se, aby rozhraní komunikace s databází bylo kompatibilní i s NoSQL[4]

Byť je program napsaný v Pythonu nic nebrání vytvoření modulu v jiném programovacím jazyce a v Pythonu napsat jen modul, který jej volá jako příkaz. Problematiku IPC a kódování obrázků si pak každý takový modul může vyřešit po svém nebo navrhnout standardní rozhraní, pro které pak můžou být vytvářeny moduly v libovolných jazycích.

Protože každý z modulů s sebou přinesl vlastní potíže a řešení, budou rozebrány samostatně.

2.2 Moduly

2.2.1 Lokální soubory

Při práci na modulu pro přístup k lokálním souborům jsem narazil na to, že nemohu jednoduše vytvořit `asyncio.Task` pro všechny kombinace fotka - modul, neboť tím se vytvoří stovky připojení v databázi, které zahltí server, a stovky procesů, z modulů jako rozpoznávání obličejů, které učiní počítač během skenování nepoužitelným. Řešením tedy je zmenšit množství obrázků skenovaných současně pomocí fronty úloh a omezeného počtu paralelizovaných pracovníků.

Protože jednotlivé moduly samy o sobě nic moc nenadělají, jen reagují na události, není důvod aby program běžel. Obvykle je jím grafické rozhraní, které je však u Fotkovátoru volitelné. Tuto úlohu tedy zastává zdroj obrázků, který musí implementovat ve své metodě `run_forever` nekonečnou smyčku.

2.2.2 PostgreSQL

Fotkovátor původně používal databázový software sqlite pro jeho jednoduchost, ale na doporučení vedoucího práce jsem změnil databázi na PostgreSQL. PostgreSQL je obecně lepší a navíc podporuje JSON[5], který se v systému používá pro ukládání metadat o fotce.

Databáze by pravděpodobně šla adaptovat pro jiný SQL databázový software (například MySQL). V takovém případě bych doporučil implementovat jednotnou `SQLDatabase` třídu pro obě databáze. Zatím jsem to neudělal, protože používám příkazy specifické pro Postgres. Při vývoji takové třídy by se také muselo myslet na sjednocení chyb z různých databázových klientů.

[3] Structured Query Language - standardizovaný strukturovaný dotazovací jazyk pro relační databáze[19]
[4] Databázový koncept jiný než tradiční tabulkové schémata[17]
[5] JavaScript Object Notation - formát pro serializaci dat

2.2.3 Webové rozhraní

Webové uživatelské rozhraní je multiplatformní a díky knihovnám, jako například projektem používaný Bootstrap, je vývoj rychlý a výsledek vypadá moderně. Bylo tedy jasnou volbou.

Modul má ještě poměrně velký potenciál na rozšíření. Místo stránkování by mřížka obrázků mohla být generována dynamicky JavaScriptem podle pozice rolování. To by také dovolilo využití v CSS[6] implementované, ale nevyužité třídy `big` určené výjimečným obrázkům, která je v mřížce zvětší.

Snažil jsem se, aby stránka fungovala bez JavaScritu a do velké míry se to podařilo až na vyhledávání, které potřebuje JavaScript na našeptávání a formátování štítků.

Jako webserver používám Python modul `quart` pro jeho kvalitní integraci do systému `asyncio` a podobnost s web serverem `flask` na který jsem zvyklý.

2.2.4 Zpracování metadat

Kromě přidávání štítků podle dat v exifu, zpracovává modul i metadata samotného souboru. Zajímavé tady je zpracování cesty k souboru. Využívá se na přidání štítků podle složek, ve kterých se fotka nachází. Hledal jsem způsob jak určit, které názvy složek jsou relevantní. Nejprve jsem se snažil najít způsob jak ignorovat časté automatické složky jako `100ANDRO`, ale rozhodl jsem se, že to je příliš nespolehlivé. Modul tedy přidává štítky jen podle složek s datem (například `Maturitní ples GJK 2023`)

2.2.5 Rozpoznání obličejů

Standardně se při rozpoznávání obličejů používá databáze známých obličejů. Fotkovátor neimplementuje způsob jak přidávat prvky do grafického rozhraní, protože jsem nenašel dostatečně obecný způsob, který by se dal jednoduše přidat do webové stránky, android/iOS aplikace i desktopového programu.

Kvůli tomuto omezení, modul rozpoznávání obličejů nemůže jednoduše přidat obrázek konkrétního obličeje s otázkou „Jak mám nazývat tohoto člověka?“. Místo toho pouze všechny obličeje převede do latent space[7], kde na ně spustí algoritmus agglomerative clustering[8], vytvořená seskupení pak reprezentují odhad systému o totožnosti člověka.

Jakmile jsou obličeje oštítkované, může je uživatel v grafickém rozhraní přejmenovat. To vyvolá událost `rename_tag`, kterou zachytí i modul obličejů. Tady je nutné podotknout, že bez využití zmíněné výhody systému událostí, tedy možnosti poslouchat události primárně určené jinému modulu (v tomto případě databázi), by nebylo jak zjistit, že uživatel tuto akci provedl. V reakci na tuto událost modul vypočte aritmetický průměr všech vektorů s daným obličejem a uloží si asociaci s novým jménem. Tyto asociace pak aplikuje při skenování dalších fotek.

[6] Cascading Style Sheets - kolekce metod pro grafickou úpravu webových stránek[6]
[7] n-dimenzionální prostor, kde každý bod něco reprezentuje, v tomto případě identitu člověka
[8] Algoritmus jsem zvolil, kvůli výsledkům na umělých datech[4].

Modul rozpoznávání obličejů si vytváří vlastní paralelní databázi, a to ze dvou důvodů: Zaprvé, aby nemusel při každém přidání nového obrázku znovu vypočítat všechny body, ale pořád je měl v latent space pro seskupování a zadruhé, jelikož se v ní nachází informace o tom, kvůli kterému obličeji byl daný štítek přiřazen ke kterému obrázku, která je potřeba při vytváření asociací s jmény.

2.2.6 CLIP

CLIP je neuronová síť propojující text a obrázky.[10] Ve skutečnosti jde o dva modely: jeden pro obrázky a jeden pro text. Oba svůj vstup promítnou do stejného vektorového prostoru reprezentujícího vizuální koncepty.

Použití CLIPu značně rozvíjí konfigurovatelnost celého systému, neboť umí rozpoznávat téměř cokoliv. Místo vytváření vlastního modulu pro každou specializovanou neuronovou síť, což by vyžadovalo spoustu lidského času a při spouštění by zabralo zbytečně víc výpočetního výkonnu a paměti, aplikací CLIPu spustíme na obrázek jen jeden model, výsledný vektor porovnáváme s předem vypočítanými vektory a výsledky jsou obdobné bez nutnosti jakéhokoliv cvičení[10].

Pokud víme, že štítky, které se snažíme detekovat, se vzájemně vylučují, prostě vybereme textový vektor nejblíž vektoru obrázku, ale pokud se může na obrázku nacházet libovolná kombinace (například „člověk“ a „kočka“, ale ne „pes“), vzniká problém. Jedním řešením by bylo kombinace vytvářet textově („fotka psa, kočky a člověka“, „fotka kočky a člověka“...) a proměnit tak problém zpět na vzájemně se vylučující štítky. Toto řešení má ale časovou složitost $O(2^n)$. Pokud bychom omezili maximální počet věcí, které se mohou objevit, zmenší se složitost na $O(\binom{n}{k})$, kde k je maximální množství štítků na fotku.

Dalším řešením se nabízí lineární kombinace vektorů, ale u té není zcela jasné jak a jestli bude fungovat, protože CLIP při výpočtu podobnosti měří úhel ne Eukleidovskou vzdálenost. Jako příklad může posloužit případ, kdy se snažíme detekovat, jestli je na obrázku „bílá kočka“ nebo „zrzavá kočka“ a na obrázku je bílo-zrzavě pruhovaná kočka. Protože jsou vektory blízko sebe a protože vektor obrázku je mezi nimi (je blízko oběma), bude oběma vektorům přiřazena váha půl. Intuice říká, že vektor obrázek bílé a zrzavé kočky by měl mít dvojnásobnou délku a obě váhy by tedy byly jedna, ale protože CLIP je designovaný tak, že podobnosti odpovídá úhel[7], násobky vektoru by mu měly být ekvivalentní významem.

Posledním řešením, které jsem vymyslel, je aplikace nadstavby nad CLIP zvané CLIPSeg[8], která označuje pixely obrázku, kde se hledaný objekt nachází. Výběr nejbližší kategorie pak může probíhat na úrovni objektu místo obrázku. Jedinou nevýhodou je, že se model na označování musí spustit na každou dvojici obrázek a popis vizuálního konceptu. Nicméně je to řešení v $O(n)$.

2.2.7 Záznamník událostí

O tomto modulu není moc co říct. Vytvořil jsem ho za účelem ladění a sledování práce systému. Jedinou zajímavostí je netradiční použití async/await: Zatímco standardně jsou všechny funkce reagující na události asynchronní, tento modul má funkci synchronní, která ale vrací `awaitable`. To zajišťuje, že se událost do konzole vypíše ještě před tím, než na ni systém začne reagovat.

3. Technická dokumentace

3.1 Instalace

Nejprve potřeba mít nainstalovaný Python 3.10 a manažer balíčků `pip`. Doporučil bych využití systému Anaconda[1], kvůli instalaci balíčků vyžadovaných některými moduly a kvůli taktéž doporučené možnosti vytvořit virtuální prostředí:

```
conda create -n fotkovator python=3.10
conda activate fotkovator
```

Pro instalaci projektu samotného nejprve oklonujte repozitář:

```
git clone https://github.com/nesati/fotkovator
cd fotkovator
```

Od teď všechny příkazy předpokládají, že se nacházíte v kořenovém adresáři repozitáře. Dále nainstalujte závislosti příkazem:

```
pip install -r requirements.txt
```

Jednotlivé moduly pak mají svoje požadavky na instalaci, ale většinou stačí nainstalovat závislosti příkazem: `pip install -r modules/<cesta k modulu>/requirements.txt`. Bližší informace najdete v sekcích jednotlivých modulů.

3.2 Konfigurace

Celý systém se konfiguruje v souboru `fotkovator.yaml` ve formátu YAML[2]. Je nutné zvolit právě jednu databázi (`database`) a právě jeden zdroj obrázků (`backend`). Všechny ostatní moduly jsou volitelné. Je však doporučené nainstalovat a nakonfigurovat alespoň jeden modul kategorie uživatelské rozhraní (`frontend`).

V repozitáři se nachází výchozí konfigurace, ve které najdete nakonfigurovaný zdroj fotek prohledávající složku `photos`, kterou musíte vytvořit a vložit do ní fotky, které chcete oštítkovat, PostgreSQL klient připojující se na `postgres://fotkovator:mysecretpassword@localhost:5432/fotkovator`, Webovým uživatelským rozhraním na `http://localhost:8080` a všemi štítkovacími moduly v repozitáři.

Projekt jsem testoval na operačních systémech Linux a Windows.

[1] dostupné z: `https://docs.conda.io/en/latest/miniconda.html`

[2] YAML Ain't Markup Language - formát pro serializaci strukturovaných dat[20]

3.3 Použití

Před spuštěním Fotkovátoru se ujistěte, že máte nakonfigurované všechny moduly dle libosti, neboť změny v konfiguraci se projeví jen na nově naskenovaných fotkách. Program se zapne příkazem:

```
python fotkovator.py
```

Po spuštění načte konfiguraci a okamžitě začne zpracovávat fotky z nakonfigurovaného zdroje. Způsob otevření uživatelského rozhraní záleží na nakonfigurovaném rozhraní.

3.4 Moduly

3.4.1 Lokální soubory

Přístup k lokálním souborům je umožněn modulem `localfs`. Periodicky kontroluje změny v souborovém systému. Nakonfigurovanou složku musíte vytvořit ručně, pak do ní můžete vkládat a z ní odstraňovat fotky a obrázky libovolně a systém by si měl poradit.

Instalace

```
pip install -r modules/backend/localfs/requirements.txt
```

Argumenty

`path` - (povinný) Cesta ke složce s obrázky

`max_concurency` - (volitelný, výchozí hodnota: 16) Počet fotek, které analyzovat současně

Příklad konfigurace

```
backend:
  module: localfs
  path: './photos'
```

3.4.2 PostgreSQL

PostgreSQL klient implementuje modul `database.PostgreSQL`. Vyžaduje externí PostgreSQL server.

Instalace

Závislosti klienta lze nainstalovat přes `pip`.

```
pip install -r modules/database/PostgreSQL/requirements.txt
```

Je nutné nainstalovat také server. Například přes docker[3]:

```
docker run --name fotkovatordb \
           -p 5432:5432 \
           -e POSTGRES_PASSWORD=mysecretpassword \
           -e POSTGRES_USER=fotkovator \
           -d postgres
```

Argumenty

`user` - (volitelný, výchozí hodnota: `fotkovator`)

`password` - (volitelný, výchozí hodnota: proměnná prostředí `PGPASSWORD`) heslo do databáze

`host` - (volitelný, výchozí hodnota: `localhost`) IP[4] nebo hostname[5] PostgreSQL serveru

`port` - (volitelný, výchozí hodnota: proměnná prostředí `PGPORT`, nebo `5432` není-li definovaná)

`database` - (volitelný, výchozí hodnota: `fotkovator`) název databáze, ve které Fotkovátor ukládá data

Příklad konfigurace

```
database:
  module: PostgreSQL
  password: mysecretpassword
```

3.4.3 Webové rozhraní

Jednoduchý způsob jak přistupovat k Fotkovátoru je přes webový prohlížeč. Webový server implementuje modul `frontend.basic_webserver`. Pokud nezměníte konfiguraci, stačí po zapnutí Fotkovátoru navštívit `http://localhost:5000`.

[3] dostupné z: `https://www.docker.com/`
[4] IP adresa - síťový identifikátor serveru
[5] Jméno hostitele - síťový identifikátor serveru[16]

Instalace

```
pip install -r modules/modules/frontend/basic_webserver/requirements.txt
```

Argumenty

`port` - (volitelný, výchozí hodnota: `5000`) port na kterém poslouchat

`host` - (volitelný, výchozí hodnota: `localhost`) interface kde poslouchat

Příklad konfigurace

```
modules:
  - module: frontend.basic_webserver
    port: 8080 # příklad změny portu
```

3.4.4 Zpracování metadat

Modul `tag.metadata` přidává štítky na základě informací, které nejsou obsaženy v obraze samotném. Například datum pořízení obrázku, složky ve kterých se nachází, název fotoaparátu atd.

Instalace

```
pip install -r modules/modules/tag/metadata/requirements.txt
```

Příklad konfigurace

```
modules:
  - module: tag.metadata
```

3.4.5 Rozpoznání obličejů

Modul `tag.face_recognition` poskytuje wrapper na knihovnu `face-recognition`. Modul sám o sobě nepřiřazuje lidem jména, ale snaží se přiřadit stejnému člověku stejný štítek (např.: Osoba 1). Jakmile však tento štítek přejmenujete asociuje si nové jméno s daným obličejem a bude ho přiřazovat i novým fotkám.

Začíná fungovat dobře až když má několik (> 10) fotek člověka, jinak má tendenci obličeje seskupovat k sobě docela náhodným způsobem. Také do někdy samostatných skupin dává chybně označené obličeje.

Instalace

```
pip install -r modules/modules/tag/face_recognition/requirements.txt
```

Je doporučené používat `dlib` z anacondy.

```
conda install -c conda-forge dlib
```

Argumenty

`db_path` - (volitelný, výchozí hodnota: `face.pickle`) Cesta k souboru ve kterém si modul ukládá rozpoznané obličeje a další pomocné informace.

Příklad konfigurace

```
modules:
  - module: tag.face_recognition
```

3.4.6 CLIP

CLIP je neuronová síť propojující obrázky a text. Lze ji aplikovat na rozpoznávání víceméně libovolných vizuálních konceptů popsatelných slovy[10]. Implementuje ji modul `tag.CLIP`.

Jde o asi nejmocnější část Fotkovátoru. Je velmi obecný a lze ho specializovat na rozlišování téměř čehokoliv. Jeho obecnost má však svoje limity. Nerozumí příliš dobře věcem, které jsou na internetu málo anotované. Pokoušel jsem se tento modul využít k detekci fotek, které jsou otočené, ale bez úspěchu.

U méně používaných jazyků se jeho porozumění výrazně zhoršuje a s jazyky, které nejsou psané latinkou, je naprosto nepoužitelný. Proto konfigurace nabízí možnost zadat popis obrázku v jiném jazyce (doporučuji anglicky) než samotné přiřazené štítky.

Tvorbu klasifikátoru si můžete představit jako seznam možností odpovědi na otázku „Co nejlépe popisuje daný obrázek?“. `threshold` pak určuje, jak moc si CLIP musí být jistý odpovědí, aby se štítky přiřadily.

K tvorbě klasifikátoru je ještě nutné podotknout, že model je náchylný na formulaci daného popisu. Kolem tohoto problému již vzniká komunita tzv. prompt inženýrů, která se snaží najít metody formulování s nejlepšími výsledky[18].

Pokud požíváte více klafisikátorů není nutné mít víc instancí CLIP modulu.

Argumenty

`model` - (volitelný, výchozí hodnota: `ViT-B/32`) verze CLIPu, kterou požívat

`classifiers` - (povinný) seznam klasifikátorů

`threshold` - (volitelný, výchozí hodnota: `0`) minimální pravděpodobnost nutná pro přiřazení štítku

`concepts` - (povinný) vizuální koncepty a k ním asociované štítky (viz příklady konfigurace)

Štítků spojených s daným konceptem může být libovolný počet. Pokud mají dva klasifikátory stejný název štítku, jsou tyto štítky automaticky sloučeny

`prefix` - text co se vloží před každý koncept (například "a photo of ")

Instalace

```
pip install -r modules/modules/tag/CLIP/requirements.txt
```

Je doporučené používat `pytorch` a `torchvision` instalované pomocí Anacondy[6]

Například pro NVIDIA CUDA[7]:

```
conda install pytorch torchvision pytorch-cuda=11.6 -c pytorch -c nvidia
```

Příklad konfigurace

Jednoduchý klasifikátor umístění:

```
modules:
  - module: tag.CLIP
    classifiers:
      - threshold: .8
        concepts:
          indoors: uvnitř
          outdoors: venku
          "in a vehicle": ve vozidle
```

Pokročilý klasifikátor vytvořený na základě datasetu Places2[21] najdete v souboru `modules/modules/tag/CLIP/places.yaml`.

Jednoduchý klasifikátor objektu fotky:

[6]konfigurátor příkazu dostupný na: `https://pytorch.org/get-started/locally/`
[7]Compute Unified Device Architecture - platforma pro spouštění výpočtů na grafické kartě[15]

```
modules:
  - module: tag.CLIP
    classifiers:
      - prefix: "a photo of "
        concepts:
          people: lidé
          a landscape: krajina
          architecture: architektura
          animals: zvířata
          food: jídlo
          nature: příroda
          travel: cestování
          a sport: sport
          an event: událost
          an object: předměty
          an art piece: umění
          a vehicle: vozidla
          a paper: dokument
          space: vesmír
          electronics: elektronika
```

Detektor fotek na smazání:

```
modules:
  - module: tag.CLIP
    classifiers:
      - threshold: .5
        concepts:
          "bad photo": "nepovedená"
          "normal photo": []
          "good photo": []
          "excellent photo": []  # zde si můžete přidat vlastní štítek
```

Detektor ne-fotek:

```
modules:
  - module: tag.CLIP
    classifiers:
      - concepts:
        "photo, photograph": []
        "diagram, chart, graph, infographic, figure": "diagram"
        "document, photo of a document, note, paper": "dokument"
        "meme, joke, text, template, deep fried": "meme"
        "drawing, scribble": "obrázek"
```

3.4.7 Záznamník událostí

Modul `misc.event_logger` slouží hlavně pro ladění. Vypisuje události a jejich argumenty do konzole.

Příklad konfigurace

```
modules:
  - module: misc.event_logger
```

Závěr

3.5 Shrnutí

V rámci projektu se podařilo navrhnout modulární systém pro automatizované štítkování fotek a implementovat jádro, jeden zdroj obrázků - lokální soubory, jednu databázi - PostgreSQL klient, jedno uživatelské rozhraní - webovou aplikaci, tři štítkovací moduly - analýzu metadat, rozpoznání obličejů a CLIP a jeden další modul - záznamník událostí pro ladění.

Fotkovátor na rozdíl od komerčních řešeních klade důraz na konfigurovatelnost a modulárnost. To mu dává potenciál pro mnohem širší využití. Lze ho specializovat pro určité katalogy obrázků jako například memes[8], arts and crafts inspirace atd. nebo pro určité úkoly například sestavení kalendáře z rodinných fotek tak, aby fotka na každém měsíci odpovídala alespoň roční dobou a fotky všech dětí byly rozmístěny víceméně rovnoměrně. Také má potenciál být připojen k již existujícím systémům jako například multimediální server Plex.

Systém jsem otestoval se svojí omezenou sadou fotek a je možné, že na rozmanitějším katalogu obrázků, se projeví nezamýšlené chování nebo neočekávané chyby.

3.6 Hodnocení

Na závěr bych rád napsal, že se domnívám, že se mi podařilo úkol, který jsem si na začátku dal, splnit. Můj MVP fotkovátor plní základní funkce oštítkování a nacházení fotek. Protože Fotkovátor v současném stavu využívá pouze známých algoritmů a veřejně dostupných modelů, lze ho spustit zcela offline, jak jsem si vytyčil.

Byl to můj první větší projekt a byl pro mě důležitou zkušeností. Na každém kroku jsem musel řešit spoustu nečekaných problémů. Myslím, že jsem se během práce hodně naučil nejen o technologii `asyncio`, jak jsem měl v úmyslu, ale i z mnoha jiných oborů.

Nejsem zcela spokojený s mírou konfigurovatelnosti modulů. Například modul na štítkování podle obličejů by mohl nabídnout možnost změnit velikost modelu, aby si uživatel mohl vybrat svůj balanc mezi výpočetní náročností a přesností. Na druhou stranu mám již teď zpětnou vazby od spolužáka, že konfigurace je netriviální.

Původně jsem naivně předpokládal, že sám zpracuji moduly pro mnoho dalších funkcí, ale ukázalo se, že uvést něco do funkční podoby je mnohem těžší než, když si jen teoreticky představuji, jak bude daná funkce pracovat. Určitě je tedy ještě mnoho možností, jak projekt rozšířit, ale to jsem od začátku předpokládal. Teď když funguje jádro systému, počítám s tím, že ještě další moduly udělám nebo je dokonce vytvoří někdo jiný.

[8] V moderním slova smyslu - vtipy v podobě obrázků

3.7 Možnosti rozšíření

3.7.1 Mobilní aplikace

Byť je spuštění Fotkovátoru na mobilním telefonu teoreticky možné díky aplikacím jako iSH[13] a termux[1], není to ani z daleka ideální řešení. Tím by byla mobilní aplikace, která slouží současně jako zdroj obrázků i grafické rozhraní. Tyto dva komponenty by šly udělat nativně a přidat k nim python server fungující jako modul zdroje obrázků a grafického rozhraní. Případně by pomocí projektu python-for-android[12] šlo spustit jádro přímo na telefonu spolu s databází a moduly tak, aby celá aplikace byla soběstačná.

3.7.2 Spouštění modulů externě

Pro konfigurace s nižším výpočetním výkonem (například navrhovaná android aplikace) se nabízí předání výpočtů do cloudu nebo lokálnímu počítači. Takové řešení by mělo dvě části: klient a server. Klient by byl implementovaný jako standardní modul a server by byl implementovaný jako alternativní jádro, které nemá zdroj obrázků ani databázi, ale všechny požadavky posílá do klientského modulu. Musel by se také vyřešit problém výpadku sítě, ale na to by stačila fronta událostí.

3.7.3 Detekce duplicit

Ve velkých sbírkách rodinných fotek se často stává, že některé soubory jsou nakopírované vícenásobně, například protože někdo stahuje všechny fotky z telefonu a zároveň je pokaždé na telefonu nechá. Nebo například moderní telefony umožňují vyfotit desítky fotek za pár sekund, což je sice užitečné snažíte-li se zachytit perfektní snímek neposedného dítěte, ale zbytečně to zanechává desítky téměř identických fotek na disku. Řešení se nabízí v podobě modulu do Fotkovátoru, který pracuje obdobně jako modul na rozpoznávání obličejů, tedy tak, že nejprve projde všechny fotky a vytvoří jejich vizuální hash a poté přiřadí stejný štítek všem fotkám majících stejný hash. Případně méně výpočetně náročné řešení, avšak méně přesné, jsou standardní hashovací algoritmy jako md5 nebo sha1 v kombinaci s detekcí známých vzorců v názvech souborů (např. Samsung „burst“ fotky končí `_001.jpg` až `_100.jpg`).

3.7.4 OCR

Dalším užitečným modulem by mohlo být OCR[9]. Usnadnil by detekci dokumentů i vyhledávání v nich.

3.7.5 Moduly vyhledávání

Vyhledávání je jedna z nejdůležitějších schopností Fotkovátoru a přesto v současném stavu není modulární. Je to proto, že jsem chtěl projekt udržet jednoduchý, ale pryč od implementace. Protože

[9]Optical Character Recognition - rozpoznávání textu na obrázku

vyhledávací modul musí fungovat jako volání metody a ne událost, může být jen jeden, a to i z logiky věci (jak seřadit výsledky více vyhledávacích modulů), ale to by znamenalo, že každý vyhledávací modul musí řešit vyhledávání pomocí štítků. Byť může stavět nad vyhledávání štítků, odstraněním vyhledávání dle štítků by všechny štítkovací moduly přestaly být užitečné.

Nenapadá mě však lepší řešení, proto ho tady alespoň trochu nastíním. Neuronovou síť CLIP[10] lze použít přímo jako vyhledávač[2]. Vyhledávaný dotaz se zpracuje do vektorového prostoru (aby vyhledávání bylo kvalitní i v češtině, musel by se předem ještě přeložit) a obrázky se seřadí od nejmenšího úhlu (nejvýstižnější popis) po největší úhel. Databázi si samozřejmě musí vyhledávací modul implementovat sám, aby byla optimalizovaná na daný úkol.

Řešení otázky zapojení štítků do vyhledávání je dvojí. Intuitivní přístup je algoritmicky detekovat štítky v souvislém textu („fotka Šimona na hradě Karlštejn“ -> „fotka [Šimon] na [hrad] [Karlštejn]“) a omezit vyhledávání na fotky obsahující tyto štítky. CLIP by pak dostal původní dotaz a jsou-li štítky nějak smysluplně nazvané, pochopí, že když v dotazu je „fotka Šimona“ tak hledá fotku člověka. Tento přístup má však ten problém, že nevyužívá CLIPu na plnou míru a svazuje mu ruce.

Například, kdyby měl Šimon na fotce brýle a modul na rozpoznávání obličejů ho nepoznal, ve vyhledávání už se nemůže objevit. Napadlo mě tedy řešení složitější, které je poněkud komplexnější. Místo toho, aby se štítky používaly přímo ve vyhledávání se jejich význam naučí CLIP. To lze udělat přidáním nového tokenu[10] do textového enkodéru CLIPu[5]. Tento přístup seřadí všechny fotky takže, když nenajdeme to, co hledáme mezi fotkami mající dané štítky, můžeme prostě hledat dál. Přidání tokenu CLIPu také umožňuje zobecnění daného konceptu.

3.7.6 Moduly nahrazující fotky

Jedná se o moduly jako automatické narovnávání otočených fotek, vybarvování černobílých fotek, konverzi fotek do efektivnějších formátů, upscaling[11], automatické kombinování podobných fotek do „živých fotek“[12], do panoramatu nebo tak aby se všichni dívali do kamery najednou atd. Důvod proč tento typ modulu není implementován je, že bez něj zdroj fotek může být pouze pro čtení, což zaprvé ubezpečuje uživatele a zadruhé otevírá dveře zdrojům obrázků jako Instagram nebo Plex. Chceme-li zachovat možnost použití těchto zdrojů a zároveň přidat schopnost úpravy fotek, musely by implementovat ještě druhý „zdroj obrázků“ pro ukládání upravených fotek, pravděpodobně v lokálních souborech, a celý systém by se zbytečně zkomplikoval.

[10] symbol, slovo nebo část slova ve slovníku modelu[5]
[11] Zvětšení rozlišení a ostrosti
[12] Krátké animované momenty

Seznam použité literatury

[aFP23] *agnostic-apollo*, Fredrik Fornwall a Leonid Pliushch. *Termux. a terminal emulator application for Android OS extendible by variety of packages*. 2023. URL: https://github.com/termux/termux-app.

[Bea22] Romain Beaumont. *Clip Retrieval: Easily compute clip embeddings and build a clip retrieval system with them*. 2022. URL: https://github.com/rom1504/clip-retrieval.

[BFM05] T. Berners-Lee, R. Fielding a L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. RFC Editor, led. 2005. URL: https://www.rfc-editor.org/rfc/rfc3986.

[dev23] scikit-learn developers. *Comparing different clustering algorithms on toy datasets*. 2023. URL: https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html.

[Gal+22] Rinon Gal et al. "An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion". In: *CoRR* (2022). arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2208.01618.

[Jan04] Dušan Janovský. "CSS styly - úvod. Jak psát web". In: (2004). URL: https://www.jakpsatweb.cz/css/css-uvod.html.

[Kim23] Jong Wook Kim. *CLIP. Contrastive Language-Image Pretraining*. 2023. URL: https://github.com/openai/CLIP.

[LE22] Timo Lüddecke a Alexander Ecker. "Image Segmentation Using Text and Image Prompts". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Čvn. 2022, s. 7086–7096.

[LR20] Charles Z Liu a S. Ramakrishnan. *Image recognition: Progress, trends and challenges*. en. New York: Nova Science, dub. 2020. ISBN: 978-1-53617-258-4.

[Rad+21] Alec Radford et al. "CLIP. Connecting text and images". In: (led. 2021).

[RC00] Chris Ridpath a Wendy Chisholm. *Techniques For Accessibility Evaluation And Repair Tools*. Technique 2.2.1. Dub. 2000. URL: https://www.w3.org/TR/AERT/#color-contrast.

[Tay+23] Alexander Taylor et al. *python-for-android. Turn your Python application into an Android APK*. 2023. URL: https://github.com/kivy/python-for-android.

[tbo+23] *tbodt* et al. *iSH. Linux shell for iOS*. 2023. URL: https://github.com/ish-app/ish.

[Tok21] Joel Tok. *py-event-bus*. 2021. URL: https://github.com/joeltok/py-event-bus.

[Wik23a] Wikipedia. *CUDA — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=CUDA&oldid=1146213663. 2023.

[Wik23b] Wikipedia. *Hostname — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Hostname&oldid=1145571096. 2023.

[Wik23c] Wikipedia. *NoSQL — Wikipedia, The Free Encyclopedia*. http://cs.wikipedia.org/w/index.php?title=NoSQL&oldid=20802370. 2023.

[Wik23d] Wikipedia. *Prompt engineering — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Prompt%20engineering&oldid=1146124951. 2023.

[Wik23e] Wikipedia. *SQL — Wikipedia, The Free Encyclopedia*. http://cs.wikipedia.org/w/index.php?title=SQL&oldid=21517116. 2023.

[Wik23f] Wikipedia. *YAML — Wikipedia, The Free Encyclopedia*. http://cs.wikipedia.org/w/index.php?title=YAML&oldid=21899840. 2023.

[Zho+17] Bolei Zhou et al. "Places: A 10 million Image Database for Scene Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017). URL: http://places2.csail.mit.edu/index.html.

Seznam obrázků

Seznam tabulek