SIMages – Podobnostní spojení na databázi obrázků

Šimon Růžička, FIT ČVUT
ruzicsi1@fit.cvut.cz

17. dubna 2024

Zadání

Cílem projektu je vytvoření webové aplikace, která umožní podobnostní spojení dvou sad obrázků s využitím rozsahového dotazu (range query) a dotazu na k nejbližších sousedů (k-nearest neighbor (kNN) query).

Prohlášení

Projekt byl zadán jako semestrální práce v rámci předmětu BI-VWM.21[1]. Rád bych poděkoval Ing. Jiřímu Novákovi, Ph.D. (jiri.novak@fit.cvut.cz) za konzultace práce v rámci cvičení tohoto předmětu.

1 Úvod

Semestrální práce implementuje webovou aplikaci pro podobnostní spojení jednoho obrázku, tzv. klíče, se sadou obrázků. Aplikace nahraje dostupné sady obrázků z příslušného adresáře a umožní uživateli interaktivní výběr klíče a sady obrázků. Zároveň má uživatel k dispozici několik parametrů vyhledávání, jako je výběr predikátu (kNN, rozsahový) a jeho parametru nebo výběr metriky (cosinová podobnost, Euklidovská vzdálenost). Uživateli se následně na nové obrazovce zobrazí jeho zvolený klíčový obrázek, společně se spojenými obrázky seřazenými od nejpodobnějších podle zvolené metriky.

Po dokončení dotazu má uživatel možnost jej nadále upravovat, buď změnou parametrů vyhledávání nebo výběrem nového klíčového obrázku. Pro přehlednost aplikace vždy zobrazuje pouze podobnostní spojení pro jeden klíčový obrázek. Aplikace se tímto přístupem snaží simulovat vzhled doporučovacích algoritmů různých webových služeb. Jedná se tedy o jednoduchý algoritmus doporučování na základě obsahu, který by pro praktické použití v on-line službě mohl být doplněn např. o algoritmus kolaborativního filtrování, tedy doporučení na základě aktivity jiných uživatelů.

Obrázky jsou porovnávány na základě deskriptorů vygenerovaných konvoluční neuronovou sítí VGG-16[2]. Použitá technika je detailněji popsána v sekci 2.1.

[1] https://bk.fit.cvut.cz/cz/predmety/00/00/00/00/00/00/06/70/24/p6702406.html
[2] Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556

2 Způsob řešení

Z algoritmického hlediska by se dal projekt rozdělit na dva odlišné procesy. Jedná se o předzpracování dat v podobě generování deskriptorů pomocí neuronové sítě (sekce 2.1) a samotné vyhodnocení dotazu (sekce 2.2).

Předzpracování dat neuronovou sítí je na běžném stroji pro větší sady obrázků časově náročné. Tento proces však stačí provést pouze jednou, a to při nahrávání samotné sady obrázků, případně při jejím rozšíření o nové obrázky. Po tomto výpočtu se všechny deskriptory uloží do komprimovaného souboru `features.npz`, odkud se následně načítají při provádění dotazu. V ideálním případě tedy proces proběhne na pozadí, aniž by zdržoval uživatelskou aktivitu.

Pro vyhodnocení dotazu je potřeba porovnávat známý klíčový obrázek s každým obrázkem ve zvolené sadě za použití předpočítaných deskriptorů. Algoritmus byl vyladěn tak, aby byl dostatečně responzivní pro použití za běhu aplikace. Pro větší data je možné, že by bylo potřeba dotazovací algoritmus optimalizovat, možnosti jsou popsány v sekci 2.3.

Bližší podrobnosti o rychlosti těchto operací jsou popsány v sekci 5.

2.1 Generování deskriptorů

Algoritmus pro generování deskriptorů využívá konvoluční neuronovou síť VGG-16, respektive její předpřipravenou implementaci v balíčku Keras[3]. Tato síť byla původně navržena a natrénována pro klasifikaci obrázků z datové sady ImageNet v rámci soutěže ILSVRC[4]. Samotné fungování neuronové sítě přesahuje znalosti relevantní pro tento předmět a tuto semestrální práci. Pokud však z neuronové sítě odejmeme poslední vrstvu, která generuje samotné pravděpodobnosti jednotlivých tříd datasetu ImageNet, dostaneme vektor hodnot (fakticky trojrozměrný tensor, výška × šířka × počet kanálů) reprezentující obrázek. Technice použití natrénovaných neuronových sítí na zcela jiných datech nebo za jiným účelem se přezdívá *transfer learning*.

Nad tímto vektorem předposlední vrstvy neuronové sítě tedy můžeme uvažovat stejně, jako nad deskriptory konvenční metody zpracování obrazu. V případě sítě VGG-16 má výstup rozměry $7 \times 7 \times 512$, tedy celkem 25088 32-bitových desetinných čísel. V důsledku fungování sítě je však tento vektor řídký, v průměru je přibližně z 90 % tvořen nulami. Toho bychom mohli využít vhodnou reprezentací, která by ušetřila výpočetní čas i velikost ukládaných souborů, tato možnost je diskutována v sekci 2.3.

Pro ukládání deskriptorů byla použita knihovní funkce `numpy.savez_compressed`[5], která využívá kompresního algoritmu `gzip`. Tento algoritmus již dokáže využít řídkosti deskriptorů, experimentálně se ukázalo že zmenšil velikost souboru s deskriptory zhruba osminásobně.

2.2 Vyhodnocení dotazu

Při vyhodnocení dotazu je nejprve potřeba porovnat vektor deskriptorů klíčového obrázku s jednotlivými vektory ve zvolené datové sadě. K tomu se používá jedna ze zvolených metrik, cosinová podobnost nebo Euklidovská vzdálenost. Metriky se liší především v pořadí záznamů – dotaz má prioritně vracet obrázky s maximální podobností, nebo s minimální vzdáleností. Dalším rozdílem je také citlivost Euklidovské vzdálenosti vůči škálování, zatímco cosinová podobnost uvažuje všechny k-násobky vektoru jako identické a vnímá pouze jeho směr.

Zvolený přístup k samotnému vyhodnocení je pak poměrně naivní – uvažujeme dvojice klíčového obrázku s každým obrázkem v sadě, vypočítáme sledovanou metriku a záznamy podle této metriky seřadíme. Možné optimalizace jsou diskutovány v sekci 2.3.

[3] `https://keras.io/api/applications/vgg/`

[4] `https://image-net.org/challenges/LSVRC/`

[5] `https://numpy.org/doc/stable/reference/generated/numpy.savez_compressed.html`

Po korektním seřazení obrázků ještě přichází druhá fáze vyhodnocení, tou je aplikace samotného predikátu. V případě kNN predikátu pouze ořízneme seřazené pole obrázků na požadovanou velikost, zadanou uživatelem. V případě rozsahového predikátu nejprve spočítáme, kolik záznamů přesáhlo uživatelem nastavenou hranici, k tomu lze na seřazeném poli použít binární vyhledávání. Tento nalezený počet následně nastavíme jako k a pokračujeme stejně, jako u kNN predikátu.

2.3 Možnosti optimalizace

Uvádíme některé z možných návrhů, které by mohly zrychlit běh programu (fázi vyhodnoceni dotazu), ale jejich implementace byla mimo časové možnosti této semestrální práce.

Řídká reprezentace deskriptorů by patrně nijak výrazně nezměnila jejich velikost na disku, díky použitému kompresnímu algoritmu. Jejich použití by však potenciálně mohlo zrychlit fázi výpočtu metriky. Jelikož pro výpočet metrik byly použity funkce z knihovny numpy, pak by pro řídkou reprezentaci bylo nejvhodnější použít kompatibilní typ `scipy.sparse`[6].

Pro optimalizované vyhodnocení dotazu by také bylo možné využít vlastností metrického prostoru. Techniky využívající tabulek pivotů a hierarchie mezi jednotlivými záznamy jsou v principu nepochybně efektivnější, než naivní porovnání s každým dostupným záznamem. Tento rozdíl by byl obzvláště znatelný pro větší počet fotek v jednom spojovaném adresáři. Tyto techniky jsou detailněji popsány v 9. přednášce BI-VWM.21[7].

3 Implementace

Celá implementace vznikla v programovacím jazyce Python ve verzi 3.12. Pro generování obrázkových deskriptorů byl použit obecný framework pro strojové učení Keras s PyTorch backendem. Následné vektorové operace, proces archivace vektorů a jejich porovnávání je implementováno převážně pomocí knihovny numpy. Webové rozhraní pak zajišťuje knihovna NiceGUI.

Kompletní seznam použitých balíčků, včetně závislostí a všech verzí je obsažen v souboru `environment.yml`. Tento soubor byl vygenerován manažerem vývojových prostředí Conda. Kopii prostředí nutnou pro spuštění serveru aplikace lze vytvořit pomocí následujícího příkazu:

```
conda env create --file /path/to/environment.yml
```

Pro spuštění klienta aplikace pak postačuje webový prohlížeč a síťové spojení se serverem, který provádí všechny výpočty.

3.1 Architektura aplikace

Aplikace je logicky rozdělená do několika modulů. O výpočetní část práce se starají moduly `compute_features.py` (generování deskriptorů) a `nearest.py` (vyhodnocení dotazu). Generování dvou webových stránek pak obstarávají moduly `key_select.py` a `similar_results.py`, obsažené uvnitř adresáře `gui/`. Program se spouští pomocí skriptu `__main__.py`, který je z důvodů multiprocessingu potřeba spustit napřímo, případně pomocí symbolického odkazu.

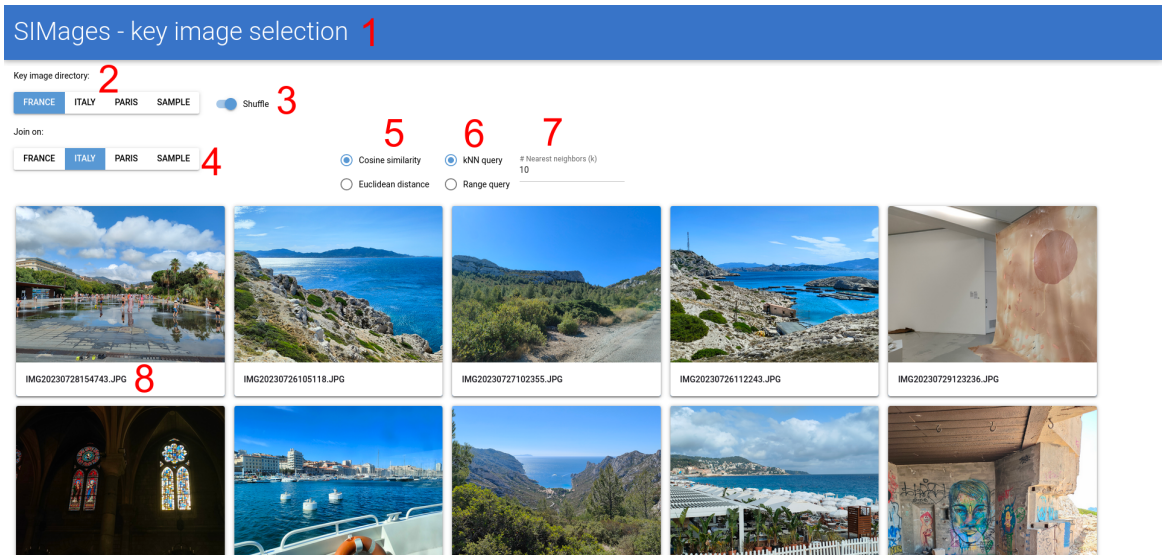
[6] `https://docs.scipy.org/doc/scipy/reference/sparse.html`

[7] `https://moodle-vyuka.cvut.cz/pluginfile.php/780673/course/section/111400/BIVWM_lecture09.pdf`

4 Příklady vstupu a výstupu

Webové rozhraní obsahuje celkem dvě webové stránky. První slouží k prvotnímu nastavení vyhledávání a klíčového obrázku. Druhá stránka pak v první řadě zobrazuje výsledek vyhledávání, lze na ní ale dotaz dodatečně upravit.

4.1 Výběr klíčového obrázku



Obrázek 1: Úvodní stránka – výběr klíčového obrázku

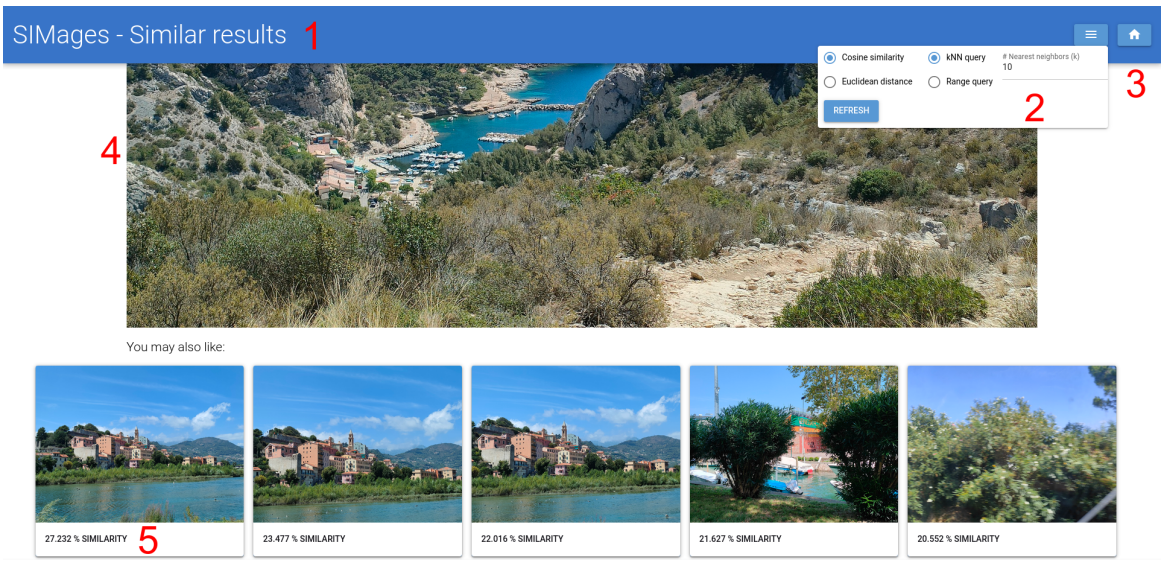
Jednotlivé prvky stránky 1, jako označené v obrázku:

1. Název stránky.
2. Volba datové sady klíčového obrázku. Z této sady se načítá prezentovaný výběr obrázků, ihned po provedení volby.
3. Možnost randomizovat pořadí obrázků ve výběru.
4. Volba datové sady pro podobnostní spojení. Obrázky z této sady se budou zobrazovat na další stránce jako výsledky dotazu.
5. Volba metriky podobnosti – cosinová podobnost, Euklidovská vzdálenost.
6. Volba predikátu – kNN, rozsahový.
7. Volba parametru predikátu, v závislosti na zvoleném predikátu a metrice.
8. Výběr klíčového obrázku, zobrazuje se náhled a jméno souboru.

4.2 Výsledek dotazu

Jednotlivé prvky stránky 2, jako označené v obrázku:

1. Název stránky.
2. Rozbalovací nabídka s nastavením, shodné rozhraní jako na stránce 1, s přidaným potvrzovacím tlačítkem.



Obrázek 2: Stránka s výsledkem dotazu

3. Tlačítko pro návrat na předchozí domovskou stránku.

4. Klíčový obrázek zvolený v předchozím kroku.

5. Výsledek dotazu, seřazený podle relevantní metriky. Zobrazuje se náhled a výsledek metriky. Po prokliknutí se obrázek vykreslí přes celou obrazovku.

5 Experimentální sekce

Provedli jsme časové testy na jednotlivé výpočetní operace, které aplikace provádí. Všechny testy byly prováděny na běžném notebooku (procesor AMD Ryzen 4500U[8]), bez použití GPU pro vyhodnocení neuronové sítě.

Testy ukázaly, že proces generování obrázkových deskriptorů trvá včetně režie načítání souborů přibližně 150 ms na obrázek. Pro menší sady obrázků by byl čas ještě o něco vyšší, neuronové sítě jsou optimalizované na dávkové vyhodnocení, v tomto případě jsme zvolili velikost dávky 64. Je tedy zjevné, že tento proces není realisticky proveditelný za běhu aplikace. V tomto případě se také nenabízí žádné zjevné optimalizace v kódu, pouze použití většího výpočetního výkonu nebo kompletně jiného algoritmu.

Pro datové sady testované velikosti v rozmezí 200 až 300 obrázků trval jeden dotaz zhruba 250 ms v závislosti na parametrech. To znamená, že byl dostatečně rychlý na použití za běhu aplikace. S rostoucím počtem obrázků ve spojované datové sadě by doba vyhodnocení zjevně rostla asymptoticky lineárně; nejnáročnější operací je porovnávání deskriptorů klíčového obrázku s každým obrázkem ve spojované sadě.

Jelikož je vyhodnocení výsledku silně subjektivní, nebylo možné provést objektivní testy kvality výsledků. Jednou z možností by bylo vytvořit testovací skupinu uživatelů, kterým by byly předloženy výsledky dotazu v porovnání s jinými prototypy, případně s konkurenčními službami. Následně by se na základě průměrného hodnocení uživatelů dalo odvodit, který z algoritmů funguje ”lépe”. Takový test není v možnostech této práce.

[8] `https://www.amd.com/en/product/9101`

6 Diskuze

Hlavní myšlenkou projektu bylo implementovat návrh algoritmu, který je schopný vyhledávat podobné obrázky na středně velké datové sadě, obsahující řádově stovky obrázků. Tento algoritmus je stále možné optimalizovat, jak konceptuálně (vizte sekci 2.3) tak i efektivnější implementací. Ačkoliv byly při implementaci prioritně používány funkce z relativně efektivních knihoven (numpy, scipy), Python se v žebříčcích rychlosti běhu programovacích jazyků obvykle umisťuje na zadních příčkách[9]. Vhodným řešením by tedy bylo vytvořit implementaci v rychlejším kompilovaném jazyce (např. C/C++, Rust), která by byla následně volána z webserveru. Podobné, optimalizace by bylo možné najít také na straně frameworku pro neuronové sítě PyTorch, jako například přechod z JIT na kompilovaný model[10].

Bylo by také možné zlepšit kvalitu výsledků pomocí dříve uvedených testů s lidskou evaluací modelu (sekce 5). Pravděpodobně by bylo vhodné experimentovat s komplexnějšími metodami zpracování obrázku, jako jsou různé transformace před vstupem doo neuronové sítě. Příkladem může být postup autorů VGG-16, který z obrázku vyberou několik automatických výřezů, které jednotlivě vstupují do neuronové sítě, výsledná predikce je pak tvořena průměrem těchto dílčích predikcí.

7 Závěr

Samotný princip podobnostního spojování obrázků je využitelný v mnoha různých aplikacích. Nejčastější využití však obnáší spojení velké sady se sebou samotnou (tzv. self-join), např. všechen obsah na sociální síti nebo obrázky indexované vyhledávačem. Uživateli je možné po zobrazení obrázku (feed sociální sítě, vyhledávání) nabídnout podobný obsah, který pro něj může být užitečný, udrží jeho pozornost na síti, atd. Podobné techniky pravděpodobně webové vyhledávače využívají pro tzv. query-by-example, kde je fulltextový dotaz ve vyhledávání nahrazen obrázkem.

Navržený prototyp je funkční pro menší datové sady (řádově stovky obrázků), jeho největším kamenem úrazu pro praktické využití je optimalizace. Obsahuje webové rozhraní, ve kterém je možné zvolit klíčový obrázek, spojovanou sadu a jiné parametry vyhledávání (typ metriky, typ dotazu/"predikát").

[9] `https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html`
[10] `https://pytorch.org/tutorials/intermediate/torch_compile_tutorial.html`