Aplikacija za iskanje podobnih slikarskih del

Luka Končar[1], Rok Caserman[2], Gregor Brantuša[3], Matevž Andolšek Peklaj[4], as. Blaž Meden[5], izr. prof. dr. Narvika Bovcon[6]

[1,2,3,4,5,6]*Fakulteta za računalništvo in informatiko UL*

E-pošta: [1]luka.koncar@gmail.com, [3]gb6370@student.uni-lj.si, [5]Blaz.Meden@fri.uni-lj.si, [6]Narvika.Bovcon@fri.uni-lj.si

Abstract

We created a web application in Python, which allows the user to upload an image and then uses it to find similar artworks from a database of paintings made by some of the most influential painters of all time. The user can choose from a variety of criteria for comparison: colour hue, saturation, value, normalized cross-correlation, edges, convolutional neural networks AlexNet and VGG16, color layout descriptor, edge histogram descriptor and a bag-of-words model. These criteria are described and discussed in the following paper, as are the algorithms that are used to implement them. Different algorithms work better or worse depending on the user uploaded image. Towards the end we suggest some functional improvements, such as identifying the author of the uploaded image and the art period in which it was likely created. Aside from the functional improvements we also suggest some that would increase the overall quality of the query, either by making it faster or more robust.

1 Uvod

Izdelali smo spletno aplikacijo, ki uporabniku omogoča nalaganje poljubne slike, nato pa iz podatkovne zbirke slikarskih del poišče podobne umetnine glede na izbrani kriterij. V nadaljevanju bomo opisali podobne aplikacije, ki že obstajajo, tehnologije, ki smo jih uporabili, različne kriterije, ki jih uporabnik lahko izbere za primerjavo slik in izgled aplikacije. Na koncu pa bomo predlagali morebitne izboljšave.

2 Pregled področja

Naš projekt spada v področje prepoznave in klasifikacije slik, v katerem že obstaja vrsta različnih aplikacij. Omenili jih bomo nekaj, ki podobno kot naša v ozadju uporabljajo podatkovno zbirko skupaj z različnimi algoritmi za klasifikacijo ali iskanje podobnih slik, kot jih je naložil uporabnik.

Quick, Draw! je zabavna aplikacija, ki uporabniku najprej pokaže besedo ali besedno zvezo, nato pa ima ta 20 sekund časa, da to skicira z miško. Cilj je, da računalnik pravilno ugotovi, kaj je narisal uporabnik. Za prepoznavo skic aplikacija uporablja rekurenčno nevronsko mrežo (angl. recurrent neural network). Ta nevronska mreža se je učila na ogromni podatkovni zbirki, ki se je sproti dopolnjevala, ko so uporabniki igrali igro in tako prispevali nove skice. Poleg klasifikacije skic je glede na podano besedo mreža zmožna tudi narisati nove skice, ki so vsakokrat drugačne in različne od vseh v podatkovni zbirki, dopolnjevati nedokončane skice, vizualizacijo vmesnih korakov pri pretvorbi ene skice v drugo idr. Zanimivo je tudi, da skic ne hranijo kot statične slike, ampak hranijo tudi podatke o tem, kako je oseba narisala skico (npr. zaporedje in smer črt). [1]

Leafsnap je mobilna aplikacija za prepoznavo rastlinskih vrst z uporabo računalniškega vida. Sistem prepozna vrste dreves iz fotografij njihovih listov. Sistem vključuje komponente za odstranjevanje slik, ki ne predstavljajo listov, segmentacijo listov od preprostega ozadja, pridobivanje značilnosti, ki predstavljajo ukrivljenost lista in prepoznavo lista iz podatkovne zbirke 184 dreves. Za klasifikacijo, ali slika sploh predstavlja list, uporabljajo binarni klasifikator z GIST značilnostmi. List od ozadja ločijo s pomočjo ocenjevanja barve ospredja in ozadja v prostoru nasičenost-vrednost barvnega modela HSV. Nato izračunajo več histogramov za ukrivljenost lista. Najbolj podobne liste iz podatkovne zbirke najdejo z uporabo preseka histogramov, te rezultate pa uporabijo za določanje, kateremu drevesu pripada list. [2]

Podobno kot zgornja projekta tudi mi uporabljamo precej veliko podatkovno zbirko (okrog 7000 slikarskih del). Za iskanje podobnih slik smo pri nekaterih kriterijih tako kot pri aplikaciji Quick, Draw! uporabili nevronsko mrežo, le da sta pri nas konvolucijski, pri njih pa je rekurenčna. Prav tako sta bili naši mreži naučeni že vnaprej, medtem ko so avtorji Quick, Draw! mrežo naučili na skicah, ki so jih pridobili od uporabnikov. Pri nekaterih kriterijih smo podobno kot pri aplikaciji Leafsnap izračunali histograme, a smo namesto s presekom histogramov podobnosti izračunali s Hellingerjevo razdaljo. Poleg tega za razliko od Leafsnap ne uporabljamo algoritmov za ločevanje ozadja od objektov v sliki, saj ozadje veliko pripomore k našemu načinu klasifikacije.

Vsebinsko pa sta našemu projektu bližje Art Palette in Art Selfie. Art Palette uporabniku omogoča bodisi vnos pet različnih barv bodisi nalaganje slike, iz katere se izlušči pet najbolj reprezentativnih barv. V ozadju je podatkovna zbirka tisoče slikarskih del, za vsako je že vnaprej izračunanih pet barv, ki to sliko najbolje predstavljajo. Iskanje najbolj podobnih slik nato opravijo s

primerjavo podobnosti teh petih barv. [3] Art Selfie pa je mobilna aplikacija, kjer uporabnik najprej slika sebe (angl. selfie), nato pa se prikaže vrsta portretov, ki so najbolj podobni naloženi sliki, zraven pa je podan tudi procent podobnosti. [4]

3 Uporabnost aplikacije

Aplikacija je namenjena različnim uporabnikom. Splošno občinstvo bo v njej videlo predvsem zabavno aplikacijo, kjer lahko naložijo svojo sliko in poiščejo podobna slikarska dela, medtem ko jih specifike algoritmov verjetno ne bodo zelo zanimale. Za njih bi se lahko aplikacijo postavilo npr. v katero izmed galerij, kjer bi se lahko s priloženo kamero tudi slikali in iskali podobne portrete.

Študente računalništva pa bo bolj zanimalo delovanje specifičnih algoritmov in kako dobre rezultate dajo. Tu so pomembni predvsem bolj kompleksni algoritmi, kot so npr. vreča besed in nevronske mreže. O osnovnem delovanju algoritmov si lahko preberejo v sami aplikaciji, če pa si želijo o delovanju izvedeti še več, pa si lahko ogledajo tudi izvorno kodo. Nenazadnje je aplikacija zanimiva tudi za študente umetnosti. Z njo si lahko pomagajo npr. pri obravnavi teorije barv, da iščejo slike, ki so podobne pa barvnem tonu, nasičenosti ali vrednosti. Pri drugih algoritmih pa lahko ocenijo, kako uspešno računalnik najde slike, naslikane v istem obdobju ali slike istega avtorja.

4 Kriteriji za primerjavo

Za iskanje podobnih slik smo uporabili podatkovno zbirko, ki vključuje slikarska dela 50 najbolj vplivnih umetnikov vseh časov [5]. Nato smo uporabili različne algoritme, ki omogočajo iskanje podobnih slik glede na sliko, ki jo je naložil uporabnik.

4.1 Barvni ton, nasičenost, vrednost

Začeli smo z najbolj preprostimi primerjavami, to je primerjanju po barvnemu tonu, nasičenosti in vrednosti. Primerjava se torej dogaja po eni izmed osi barvnega prostora HSV. Najprej smo za vsako izmed slik v zbirki naredili histogram tona, nasičenosti in vrednosti ter te podatke shranili v tri JSON datoteke. Za sliko, ki jo uporabnik naloži, se izračuna histogram za izbran kriterij, nato pa se s Hellingerjevo razdaljo poišče najbližje histograme iz JSON datotek. Imena datotek najbolj podobnih slik so nato poslana uporabniškemu vmesniku, ta pa jih uporabniku prikaže.

Kljub dokaj primitivni obdelavi vidimo, da je možno tudi s takšnimi poizvedbami poiskati dobre rezultate. To velja še posebej za slike narave, kjer prevladuje ena ali pa morebiti dve barvi.

4.2 Normalizirana navzkrižna korelacija (NCC)

Tukaj smo uporabili algoritem normalizirana navzkrižna korelacija (angl. normalized cross-correlation). Algoritem temelji na interpretaciji slike kot signala, nato pa signala primerja po tem, kakšen je njun odmik. Bolje se signala prekrivata, bolj sta si podobna. Seveda vhodno sliko spremenimo v signal in jo nato primerjamo z vsako

(a) Referenčna slika (b) Prvi zadetek

(c) Drugi zadetek (d) Tretji zadetek

Slika 1: Rezultati poizvedbe: ton

sliko iz zbirke. NCC brez dodatne obdelave je dokaj grob algoritem, ki pogosto ne vrača zelo podobnih rezultatov, zato mu v naslednjem razdelku pomagamo z ekstrakcijo robov. Pri NCC-ju je prav tako omembe vredno to, da se slike primerja, kot da bi bile črno-bele, torej se ne ozira na barve, ki jih slika vsebuje.

4.3 Robovi

Pri robovih smo iz slik najprej izluščili robove s pomočjo algoritma "Canny", nato pa smo slike z izluščenimi robovi primerjali z metodo NCC, enako kot pri prejšnji točki.

(a) Referenčna slika (b) Prvi zadetek

(c) Drugi zadetek (d) Tretji zadetek

Slika 2: Rezultati poizvedbe: robovi

Predhodno izluščevanje robov znatno izboljša rezultate, kot je razvidno iz slike 2. Prav tako vidimo NCC v akciji, saj se zadetka (c) in (d) razlikujeta v barvi, NCC pa ju postavi skupaj, saj sta po pretvorbi v črno-belo enaki.

4.4 Konvolucijska nevronska mreža AlexNet

Uporabili smo že naučeno konvolucijsko nevronsko mrežo (angl. convolutional neural network) AlexNet, ki je del Python knjižnice PyTorch [6]. Vsako sliko iz podatkovne zbirke smo najprej transformirali v skladu z navodili modela, nato pa z modelom pretvorili to transformirano sliko

v vektor značilk. Te vektorje smo shranili v JSON datoteko. Tudi sliko, ki jo naloži uporabnik, transformiramo in pretvorimo v vektor značilk, nato pa s Hellingerjevo razdaljo najdemo najbližje vektorje iz JSON datoteke in s tem pripadajoče slike.

4.5 Deskriptor barvne postavitve (CLD)

Deskriptor barvne postavitve (angl. color layout descriptor) učinkovito predstavi prostorsko postavitev barv, kar omogoča njegovo uporabo pri poizvedovanju slik. Postopek pridobivanja le-tega je sestavljen iz dveh delov; mrežnega izbora reprezentativne barve (tj. slika je razdeljena na mrežo in iz vsakega dela mreže je izbrana reprezentativna barva) in pa diskretne kosinusne transformacije. Za vse slike iz zbirke so bili predčasno pridobljeni in shranjeni pripadajoči deskriptorji, ki smo jih nato pri poizvedovanju primerjali z vhodnim deskriptorjem s pomočjo evklidske razdalje. Na ta način so se lahko enostavno poiskale slike s podobno barvno porazdelitvijo, vendar pa so se pri tem zanemarile pomembne lastnosti slik, kot je npr. tekstura.

4.6 Deskriptor histogramov robov (EHD)

Deskriptor histogramov robov (angl. edge histogram descriptor) je pridobljen s pomočjo segmentacije slike na 16 (4x4) manjših delov. Vsak segment je nato nadalje deljen do velikosti segmenta 4 (2x2) pikslov, katerim je nato določen eden izmed petih tipov robov: navpični, vodoravni, diagonala pod kotom 45 stopinj, diagonala pod kotom 135 stopinj ali pa neusmerjeni rob. S pomočjo opisanega postopka lahko na prostorsko učinkovit način pridobimo opis teksture slike, ki ga nato uporabimo pri poizvedovanju. Iz vhodne slike je pridobljen deskriptor, ki je nato primerjan z ostalimi s pomočjo evklidske razdalje. Kljub temu, da smo pri našem projektu uporabljali deskriptor zgolj na lokalni ravni, je aplikacija le-tega možna tudi na globalnem nivoju [7].

V praksi se pri vsebinskem poizvedovanju slik (angl. content-based image retrieval) velikokrat uporablja kombinacija več nizko nivojskih deskriptorjev (kot sta to npr. CLD in EHD), saj to omogoči analizo slike iz različnih vidikov (barva, tekstura, ...). Ena izmed možnih kombinacij je prav zagotovo par deskriptorjev CLD in EHD [8], ki prostorsko učinkovito opiše barvo in teksturo slike, vendar pa tega v okviru naše aplikacije nismo realizirali.

4.7 Konvolucijska nevronska mreža VGG16

Konvolucijska nevronska mreža VGG16 predstavlja nadgradnjo že omenjene mreže AlexNet. Učenje te mreže je zelo počasno, zato je bila uporabljena že naučena mreža knjižnice PyTorch [6]. Sam postopek pridobivanja vektorjev značilk pri poizvedovanju je enak postopku, opisanem v razdelku 4.4. Bistvena razlika med nevronsko mrežo AlexNet in VGG16 je v velikosti konvolucijskih jeder, kjer slednja uporablja manjša jedra (velikosti 3x3 točkovne pike), kar omogoči hitrejše učenje mreže, hkrati pa je zaradi manjšega števila spremenljivk mreža VGG16 odpornejša na prekomerno prilagajanje.

(a) Referenčna slika (b) Prvi zadetek

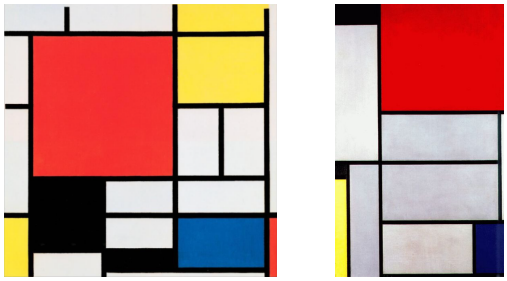
(c) Drugi zadetek (d) Tretji zadetek

Slika 3: Rezultati poizvedbe: CNN VGG16

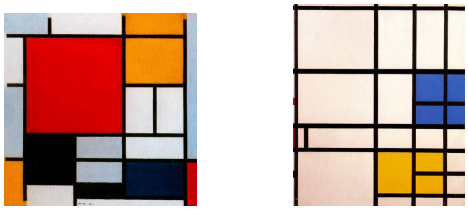
Nevronski mreži AlexNet in VGG16 se najboljše obneseta pri poizvedbah portretov. Izmed vseh implementiranih tehnik slednji prikažeta najbolj relevantne rezultate. Na sliki 3 so prikazani rezultati poizvedbe s portretom, kjer je razvidno, da je nevronska mreža VGG16 uspešno (v primerjavi z ostalimi algoritmi) poiskala podobne portrete.

4.8 Vreča besed (BOW)

Ta metoda temelji na ekstrakciji posebnih značilnosti iz vseh slik v podatkovni zbirki, te značilnosti se nato gručijo in primerjajo z algoritmi strojnega učenja (angl. bag-of-words). Točke smo iz slik pridobili s pomočjo algoritma SIFT, ki značilnosti opiše kot vektorje dolžine 128, gručili pa smo jih z metodo voditeljev. Pri metodi voditeljev je potrebno vnaprej izbrati željeno število gruč. S poskušanjem smo ugotovili, da algoritem vrača zadovoljive rezultate pri 50 gručah. Vsako gručo je nato možno interpretirati kot ”besedo” in vsako sliko kot ”besedilo”. Množico gruč, v katerih ima slika svoje značilnosti, potem interpretiramo kot besede v besedilu. Sedaj imamo besedila, ki jih lahko primerjamo kar s klasičnimi algoritmi za primerjanje podobnosti besedil. Sestavimo najprej inverzni indeks, torej slovar, ki ima za ključ oznako gruče (torej besedo), za vrednosti pa slike (torej besedila), v katerih se ”beseda” pojavlja, kar pomeni, da slika vsebuje vsaj eno izmed značilnosti iz gruče v ključu slovarja. Na tak način indeksirana besedila lahko primerjamo s pomočjo metode tf-idf. Novo sliko torej prav tako pretvorimo v besedilo in ga nato s tf-idf primerjamo z vsemi ostalimi besedili. Za razdaljo med besedili uporabimo kosinusno razdaljo, saj se je izkazala kot uporabna za našo podatkovno zbirko.



(a) Referenčna slika (b) Prvi zadetek



(c) Drugi zadetek (d) Tretji zadetek

Slika 4: Rezultati poizvedbe: bag-of-words

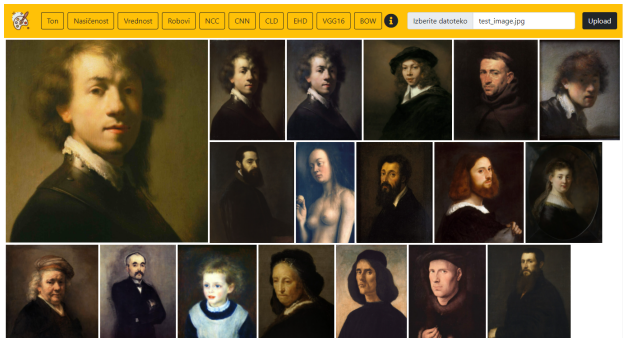
BOW je posebej odvisen od raznolikosti zbirke, s katero delamo. V zbirki, ki bi bila sestavljena samo iz človeških obrazov, bi tako npr. zlahka izluščili značilnosti, kot so nos, oko, itd. Pri zbirki, ki je tako razgibana kot naša, se stvar zaplete. Pojavi se problem, ker imajo nekatere slike veliko več posebnih značilnosti kot druge, prav tako so te značilnosti velikokrat človeku posebej brezpomenske, saj nanje gleda kot na celoto, medtem ko jih algoritem obravnava posebej. Tako nastane veliko gruč, ki jih je težko klasificirati, kot sta npr. ”nos” ali ”oko”, saj so za človeka brezpredmetne. To pogosto negativno vpliva na kvaliteto vrnjenih rezultatov. Na sliki 4 je prikazan primer, kjer za poizvedbo uporabimo sliko Pieta Mondriana, za katero se izkaže, da ima zelo malo posebnih značilnosti. Ker število značilnosti lahko znatno vpliva na kvaliteto poizvedbe, smo problem delno rešili z vpeljavo kosinusne razdalje. Ta nam pomaga tako, da pri bolj razgibanih vhodnih slikah zelo verjetno dobimo tudi razgibane rezultate in obratno.

5 Uporabniški vmesnik

Spletno aplikacijo smo izdelali v programskem jeziku Python [9] s pomočjo spletnega ogrodja Flask [10]. V desnem zgornjem kotu se nahaja gumb, s pomočjo katerega lahko uporabnik naloži poljubno sliko s svojega računalnika, ki se bo pozneje uporabila za iskanje podobnih slikarskih del. Po nalaganju slike pa v levem zgornjem kotu izbere kriterij, po katerem naj aplikacija najde podobne slike. Za vsak kriterij je na voljo en gumb, ob premiku miške na gumb pa se pojavi okence, ki na kratko opiše bistvo delovanja kriterija. Po koncu iskanja podobnih slik se spodaj prikaže večja originalna slika, ki jo je uporabnik naložil, zraven pa se postopoma prikažejo podobne slike iz podatkovne zbirke slikarskih del.

6 Možne izboljšave

Možnih izboljšav je veliko, je pa pri vsaki potrebno paziti, da na račun kvalitete ne žrtvujemo hitrosti delovanja. Možno je npr. uporabiti večjo zbirko, vendar je v tem primeru potrebno poskrbeti, da so vsi podatki iz slik izluščeni že vnaprej (to je pri večini algoritmov že implementirano) in se računa le s temi. Pri velikih zbirkah, kjer bi postalo računanje z že obdelanimi podatki zamudno, bi lahko slike predhodno gručili na podlagi njihovih barvnih lastnostih. Lahko bi torej napravili gruče na podlagi tona, vrednosti in nasičenosti barv ter vhodno sliko primerjali samo s slikami iz gruč, katerim pripada. To bi izboljšalo kvaliteto poizvedb tudi v primeru manjše zbirke.



Slika 5: Izgled uporabniškega vmesnika.

V primeru prevelike zbirke je možno uporabiti hitrejše algoritme, ki lahko najdejo manj kvalitetne rezultate, vendar so veliko hitrejši. Primer takšnega algoritma bi bil SURF, ki bi ga lahko namesto algoritma SIFT uporabili za izluščevanje posebnih značilnosti iz slik. Pri hitrosti je vredno omeniti tudi to, da je trenutno vsa koda, z morebitno izjemo uporabljenih knjižnic, napisana v programskem jeziku Python, za katerega je znano, da je počasen. Hitrost je torej možno izboljšati tudi z uporabo hitrejšega jezika, npr. C++.

Izboljšati je možno tudi vmesnik, večinoma z dodanimi funkcionalnostmi. Lahko bi npr. dodali, da algoritem poleg slik predlaga avtorja, za katerega je najbolj verjetno, da je vhodno sliko naslikal in obdobje, v katerem je najverjetneje nastalo.

7 Zaključek

Izdelali smo spletno aplikacijo, na katero lahko uporabnik naloži svojo sliko in jo z različnimi algoritmi, ki so bili opisani v tem poročilu, primerja s slikami znanih umetnikov. Soočati se s tako razgibano podatkovno zbirko, kot so dela iz mnogih obdobij, je težek zalogaj za večino algoritmov, ki so pogosto dobri v bolj specifičnih situacijah. V primeru, kjer je podatkovna zbirka zelo raznolika, se izkažejo za najbolj uporabne nevronske mreže, ki so že sicer znane kot eno izmed najboljših orodij na področju umetnega zaznavanja. Za bolj kvalitetne rezultate bi bilo potrebno razviti celosten postopek, ki bi uporabljal kombinacijo postopkov, opisanih v tem članku.

Aplikacijo smo testirali na ravni osnovnega delovanja in ovrednotili ustreznost rezultatov s pomočjo uporabnikov znotraj skupine sodelavcev. Ko smo si ogledali ponujene podobne slike, smo lahko tudi vizualno ocenili, na kateri ravni in kako so si podobne, ter človeško

zaznavo in razumevanje primerjali z delovanjem algoritmov v ozadju. Ocenjujemo, da aplikacija deluje dobro in vrača zanimive rezultate, ki bi jih lahko še dodatno analizirali in interpretirali. V nadaljevanju projekta bi bilo smiselno aplikacijo postaviti v različna okolja, opisana v 3. poglavju, kjer bi od uporabnikov pridobili informacije o specifičnih uporabah, kar bi omogočilo tudi nadaljnji razvoj in fokusiranje projekta v smeri določenih uporab, tako na ravni algoritmov kot na ravni vmesnika.

Literatura

[1] Ha, D. & Eck, D. A Neural Representation of Sketch Drawings. *CoRR*. **abs/1704.03477** (2017), http://arxiv.org/abs/1704.03477

[2] Kumar, N., Belhumeur, P., Biswas, A., Jacobs, D., Kress, W., Lopez, I. & Soares, J. Leafsnap: A Computer Vision System for Automatic Plant Species Identification. *ECCV*. (2012)

[3] Art Palette, https://artsexperiments.withgoogle.com/artpalette

[4] Art Selfie, https://artsandculture.google.com/camera/selfie

[5] Best Artworks of All Time, https://www.kaggle.com/ikarus777/best-artworks-of-all-time

[6] TORCHVISION.MODELS, https://pytorch.org/vision/stable/models.html

[7] Park, D., Jeon, Y. & Won, C. Efficient use of local edge histogram descriptor. *Proceedings Of The ACM Workshops On Multimedia*. **2** pp. 51-54 (2000)

[8] Rachmawati, E., Afkar, M. & Purnama, B. Adaptive Weight in Combining Color and Texture Feature in Content Based Image Retrieval. (2017)

[9] Python, https://www.python.org/

[10] Flask, https://flask.palletsprojects.com/en/2.0.x/