Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jan Hartman

Podobnostna mreža slik

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Blaž Zupan

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Z uporabo globokih modelov, ki jih zgradimo iz velikih slikovnih zbirk, lahko poljubne slike predstavimo v manjšem vektorskem prostoru. Tovrstne predstavitve nam omogočajo ocenjevanje podobnosti slik v poljubnih, tudi manjših slikovnih naborih. Za slednje bi lahko bila koristna vizualizacija, s katero bi hkrati predstavili vse slike iz nabora in te izrisali v mreži, tako, da bi bile slike, ki so podobne med sabo, v mreži sosedne. V diplomski nalogi predlagajte algoritem, s katerim zgradimo tak izris in ga implementirajte v programskem sistemu Orange ter ga smiselno povežite z obstoječimi gradniki za slikovno analitiko.

Zahvaljujem se mentorju prof. dr. Blažu Zupanu za odlično usmerjanje in skrbne popravke pri izdelavi diplomskega dela, ostalim članom Laboratorija za bioinformatiko pa za pomoč pri implementaciji. Posebej bi se zahvalil tudi svojim bližnjim, ki so me ves čas študija spodbujali in podpirali.

Kazalo

Povzetek

Naslov: Podobnostna mreža slik

Avtor: Jan Hartman

V diplomskem delu smo razvili novo orodje za interaktivno pregledovanje slik. Orodje za izboljšan prikaz slik kombinira uporabo globokega učenja z drugimi tehnikami podatkovne analitike. Predlagana rešitev je sestavljena iz več zaporedno povezanih metod. Prvi korak je obdelava slik z uporabo konvolucijskih nevronskih mrež za pridobitev značilk, s katerimi lahko slike med seboj primerjamo. Nato z optimizacijskimi algoritmi slike razporedimo v dvodimenzionalno pravokotno podobnostno mrežo, v kateri so podobne slike postavljene bližje skupaj. Rezultat naloge je interaktivni pregledovalnik slik, oblikovan kot podobnostna mreža. Implementiran je v programskem jeziku Python in vključen v orodje Orange kot gradnik z imenom Image Grid, kjer se enostavno povezuje z ostalimi gradniki v dodatku Image Analytics.

Ključne besede: podobnostna mreža slik, interaktivni pregledovalnik slik, konvolucijske nevronske mreže, vložitve slik, zmanjševanje dimenzij, Orange.

Abstract

Title: Image Similarity Grid

Author: Jan Hartman

The thesis describes the development of an interactive image visualization tool. To improve the display of images, the tool combines the use of deep learning with other data analytics techniques. The proposed solution consists of several sequentially connected methods. The first step is using convolutional neural networks for image processing to acquire features which can be used for image comparison. Then, optimization algorithms are used to place images into a two-dimensional rectangular similarity grid where similar images are placed closer together. The result of the thesis is an interactive image viewer, designed as a similarity grid. It is implemented in the Python programming language and included in the Orange toolbox as a widget named Image Grid, where it is seamlessly integrated with other widgets in the Image Analytics add-on.

Keywords: image similarity grid, interactive image viewer, convolutional neural networks, image embeddings, dimensionality reduction, Orange.

Poglavje 1

Uvod

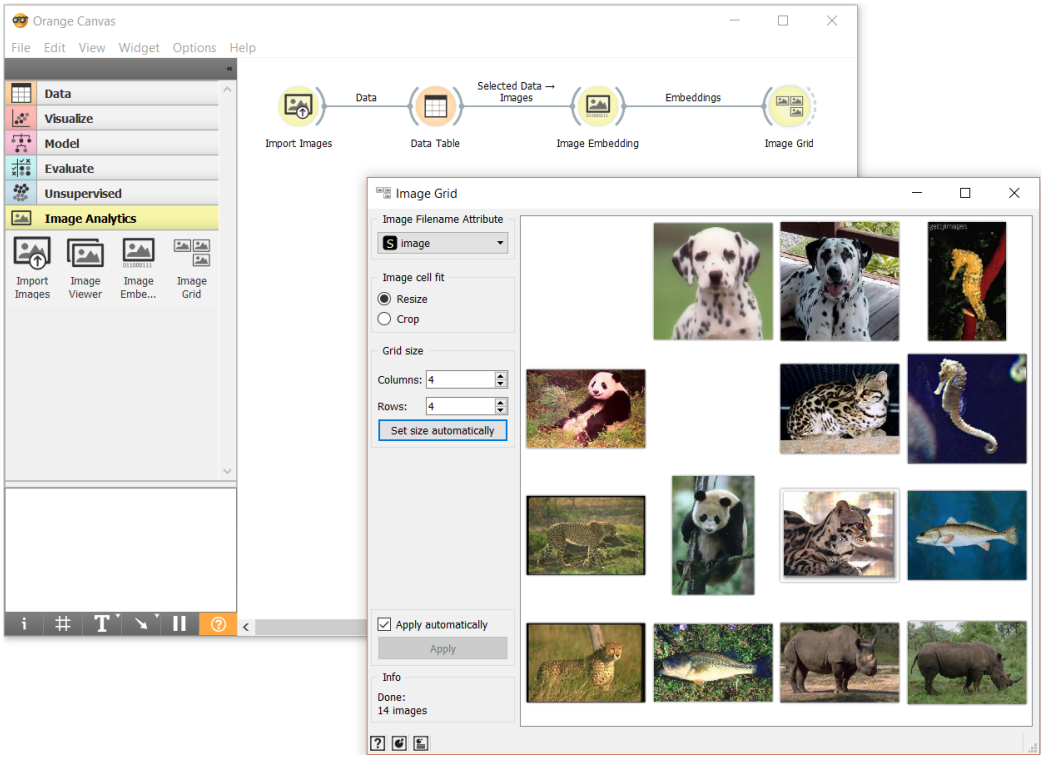
V zadnjem desetletju je na področju globokega učenja in nevronskih mrež prišlo do ogromnega napredka [17, 11]. Novi algoritmi in arhitekture nevronskih mrež so skupaj s porastom računske moči omogočili učinkovito reševanje problemov v strojnem učenju, kjer prej ni bilo učinkovitih in natančnih metod. Primeri uporabe takih tehnik so avtomatsko prepoznavanje govora, prepoznava slik in procesiranje naravnega jezika.

Čeprav imamo na voljo izredno močne metode za procesiranje slik, pa pogrešamo orodja za elegantno in napredno pregledovanje slik, ki bi uspešno kombinirala uporabo globokega učenja in druge tehnike podatkovne analitike. Iz tega se je porodila ideja za novo orodje – interaktivni pregledovalnik slik. S pomočjo nevronskih mrež bi moralo orodje znati slike med seboj primerjati in jih razporejati glede na podobnost. Pomembno je, da je orodje interaktivno in enostavno za uporabo. Uporabnost pa bi lahko še povečali z dodajanjem možnosti izbiranja in grupiranja slik za nadaljnjo obdelavo.

Kot cilj smo si zadali ustvariti rešitev v orodju za interaktivno podatkovno analitiko Orange [3, 4], ki bi uresničila želje iz prejšnjega odstavka. Poleg tega je bil cilj tudi dobra integracija v Orangevo okolje – rešitev naj bi se brez težav uporabljala v povezavi z že implementiranimi gradniki. Ker je bila naloga sestavljena iz več manjših, samostojnih delov, smo morali preizkusiti več različnih tehnik za različne namene, jih med seboj povezati in ponekod

najti najboljšo izbiro.

V sklopu te diplomske naloge smo implementirali interaktivni pregledovalnik slik. Pregledovalnik je oblikovan kot dvodimenzionalna podobnostna mreža slik. Podobnostna mreža slik je pravokotna mreža, kjer je v vsaki celici največ ena slika. Slike, ki so si med seboj podobne, so postavljene bližje skupaj – to dosežemo z nevronsko mrežo in optimizacijskimi algoritmi za preslikovanje v celice mreže. Rešitev je postala gradnik z imenom Image Grid v Orangevem dodatku Image Analytics, kjer se enostavno povezuje z obstoječimi funkcionalnostmi v tem paketu.



Slika 1.1: Delokrog za analizo slik v programu Orange in v diplomskem delu razviti gradnik Image Grid.

Primer uporabe zgrajenega gradnika je na sliki 1.1, na kateri je prikazan delokrog z gradniki, ki slike preberejo iz lokalne mape, jih predstavijo v vektorski obliki, tako opisane slike med seboj primerjajo in predstavijo v

gradniku Image Grid.

V nadaljevanju najprej opišemo uporabljene metode, predstavimo uporabljeni pristop za razvrščanje slik v pravokotno mrežo, predstavimo programsko rešitev in na koncu uporabo razvitega gradnika predstavimo na nekaj primerih.

Poglavje 2

Metode

V tem poglavju na kratko opišemo metode, ki smo jih združili v predlagano rešitev. Metode vključujejo vlaganje slik v vektorski prostor, kjer slike predstavimo z vektorji realnih števil z visokim številom (npr. 2048) komponent, metode projekcij, ki visokodimenzionalne vektorje preslikajo v dvodimenzionalni prostor, in maksimalno prirejanje, ki 2D vektorjem določi razporeditev v diskretni 2D mreži.

2.1 Vložitve slik

Da bi orodje delovalo, kot si želimo, moramo biti sposobni slike primerjati med seboj. Za ljudi je to precej enostavno, računalniki pa so tu postali res dobri šele z napredkom na področju globokega učenja [15, 16, 27]. Pri tem so ključne nevronske mreže, naučene na velikih množicah slik, ki lahko zgenerirajo predstavitve slik v obliki vektorjev. Tem predstavitvam rečemo vložitve (ang. *embeddings*).

Umetne nevronske mreže, na katerih temelji globoko učenje [17], so sistemi, sestavljeni iz preprostih elementov za procesiranje – umetnih nevronov [8]. Osnovane so na temeljih delovanja možganov, saj podobno delujejo pravi nevroni v živčnem sistemu. Umetni nevroni so funkcije, ki sprejmejo več vhodov in imajo en izhod. Imajo notranje stanje oz. uteži, ki določajo,

ali se nevron sproži in kakšen bo izhod. Več nevronov tvori nivoje, ki so med seboj povezani in v nevronski mreži opravljajo različne transformacije vhoda. Povezave med nevroni v različnih nivojih (sinapse v živčnem sistemu) so prav tako utežene. Uteži mreže pridobimo s procesom učenja in jih med učenjem sproti posodabljamo glede na optimizacijski algoritem, v njih pa je po koncu učenja shranjeno znanje mreže.

Pogosto se uporablja tudi izraz globoke nevronske mreže, kar označuje nevronsko mrežo z več nivoji med vhodnim in izhodnim (za te se uporablja tudi izraz skriti nivoji). Globoke mreže omogočajo grajenje kompleksnejših konceptov iz preprostejših [7], kar omogoča reševanje problemov, ki so se pred vzponom globokega učenja zdeli težki. Prednost nevronskih mrež je predvsem zmožnost modeliranja zapletenih razmerij, pri čemer s sestavljanjem primitivnih značilk iz prejšnjih nivojev v nekem nivoju dobimo boljšo predstavo nekega objekta. Nevronske mreže pa imajo kljub svoji moči tudi nekaj šibkih točk, najbolj očitni sta velika količina podatkov, ki je potrebna za uspešno učenje, in računska zahtevnost učenja, ki je najbolj vidna pri mrežah z veliko skritimi nivoji.

Obstaja veliko različnih tipov nevronskih mrež. Za delo s slikami so daleč najbolj popularne konvolucijske nevronske mreže [16, 11]. Za razliko od mrež s polno povezanimi nivoji je njihovo učenje manj zahtevno, saj rabijo manj nevronov in imajo lahko več skritih nivojev [17]. Konvolucijske mreže potrebujejo manj predprocesiranja vhodnih podatkov, poleg tega pa so odporne na transformacije (skaliranje, premik, ...), ki so sicer v računalniškem vidu problematične. Glavna operacija, po kateri so tudi poimenovane, je konvolucija, ki je pomembna za iskanje značilk (za različne značilke uporabimo različne filtre). Zadnji nivoji konvolucijske nevronske mreže tvorijo klasifikator, ki na izhod pošlje vektor verjetnosti, da objekt pripada različnim razredom. Poleg prepoznave slik se pogosto uporabljajo na področjih analize videa, obdelave naravnega jezika in tudi igranja iger[1].

Na strežniku Laboratorija za bioinformatiko je postavljenih več že naučenih

[1] `https://deepmind.com/research/alphago/`

nevronskih mrež. Za našo rešitev so pomembne mreže za prepoznavo slik (konvolucijske mreže). Na razpolago je več različnih arhitektur: VGG-16, VGG-19 [22] (ti dve se razlikujeta le v številu nivojev v mreži) in Googlova Inception-v3 [24], naslednica GoogleNet [23], ki so bile izdelane posebej za nalogo klasifikacije slik. Podatki za učenje naštetih modelov prihajajo iz zbirke ImageNet [5]. ImageNet je projekt, katerega glavni cilj je večmilijonska zbirka slik, ki so označene z objekti, ki so na slikah. Prav tako so objekti hierarhično organizirani in vsak je prikazan z več sto ali več tisoč slikami. Zbirka podatkov takšne velikosti je v ogromno pomoč pri učenju modela za generiranje vložitev. Projekt ImageNet tudi organizira tekmovanje ILSVRC[2] (ImageNet Large Scale Visual Recognition Challenge), v okviru katerega ekipe raziskovalcev vsako leto tekmujejo v izdelavi modelov za prepoznavo slik. Arhitekturi VGG-16 in GoogleNet sta na tem tekmovanju leta 2014 dosegle prvi dve mesti.

Omenjene nevronske mreže so enostavno dosegljive za uporabo v programu Orange preko gradnika Image Embedding. Vanj pošljemo eno ali več slik in kot izhod konvolucijske nevronske mreže dobimo pripadajoče vektorje značilk, ki opisujejo slike. Konvolucijske mreže kot izhod tipično vrnejo vektor verjetnosti, da objekt na sliki pripada različnim razredom. Gradnik Image Embedding vrne vektorje nenegativnih realnih števil, ki pa so v bistvu izhod skritega nivoja pred klasifikatorjem v konvolucijski mreži. Tem značilkam pravimo kode konvolucijske nevronske mreže (ang. *CNN codes*)[3]. Seveda ne vemo, kaj ta števila pomenijo, a so zelo uporabna – imamo predstavitve slik v prostoru realnih števil, ki jih lahko računsko primerjamo in jih uporabimo kot vhod v druge algoritme.

[2] `http://www.image-net.org/challenges/LSVRC/`
[3] `https://cs231n.github.io/transfer-learning/`

2.2 Tehnike zmanjšanja dimenzij

Podatki, ki jih želimo vizualizirati, imajo pogostokrat preveč dimenzij (npr. vložitve slik iz prejšnjega razdelka), da bi jih lahko prikazali na človeku razumljiv način. V ta namen uporabimo zmanjševanje dimenzionalnosti podatkov [28]. Cilj tega procesa je zmanjšati število značilk, s katerimi so primeri (ponekod slike) opisani. Zmanjšanje dimenzij ima tudi to prednost, da zmanjša zahteve po prostoru in času pri nadaljnjem procesiranju. Tehnike zmanjšanja dimenzij lahko v splošnem razvrstimo v dve skupini: izbira značilk in ekstrakcija značilk. Pri prvem izberemo neko podmnožico značilk iz originalne množice, pri drugem pa nove značilke tvorimo iz starih. V sklopu te naloge smo uporabili ekstrakcijo značilk.

Ekstrakcija značilk je transformacija podatkov iz visokodimenzionalnega v nizkodimenzionalni prostor. Pomaga nam predstaviti primere z manj značilkami. Originalne značilke, ki so pogosto odvečne, preslikamo v nove, informativne in neodvečne značilke. Tehnik za dosego tega je precej, v nadaljevanju pa opišemo tri bolj popularne in primerne za uporabo v našem projektu. Za to, da bo interpretacija človeku prijazna in da bo mogoče podatke predstaviti na sliki, si tipično želimo dimenzionalnost podatkov zmanjšati na dve do tri dimenzije.

2.2.1 Metoda glavnih komponent

Metoda glavnih komponent (ang. *principal component analysis, PCA*) je ena najbolj znanih metod za zmanjševanje dimenzionalnosti podatkov [9]. Visokodimenzionalni prostor z originalnimi podatki preslikamo v nov prostor z ortogonalno bazo, kjer se ohrani čim več variance, ki predstavlja odstopanja pri projekciji v nov prostor.

Metoda kot vhod prejme matriko primerov $X \in \mathbb{R}^{m \times n}$, v kateri je m primerov $(x_1, x_2, \ldots, x_m)$, ki so predstavljeni kot vektorji in opisani z n značilkami. V okviru te naloge so primeri vektorske vložitve slik. Te primere želimo opisati s k novimi značilkami, kjer je $k \ll n$. Cilj metode je

najti k ortogonalnih vektorjev $v_i \in \mathbb{R}^n$, ki tvorijo bazo novega prostora.

Varianco množice podatkov, ki jo projiciramo na prvo glavno komponento, vektor v_1, lahko zapišemo kot

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$Var(v_1^\intercal X^\intercal) = \frac{1}{m}\sum_{i=1}^{m}(v_1^\intercal x_i - v_1^\intercal \mu)^2$$

V oklepaju drugega izraza je v bistvu kovariančna matrika, ki nam pove, kako so primeri med seboj povezani. Za vsak stolpec oz. značilko izračunamo njegovo povprečje in ga odštejemo od celotnega stolpca, da podatke centriramo. Dobljeno matriko pomnožimo z njeno transponirano matriko, da dobimo kovariančno matriko $C \in \mathbb{R}^{n\times n}$:

$$C = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)(x_i - \mu)^\intercal$$

Izraz za varianco lahko poenostavimo v

$$Var(v_1^\intercal X^\intercal) = v_1^\intercal C v_1$$

Iščemo ustrezen vektor v_1, ki maksimizira varianco. Z omejitvijo $v_1^\intercal v_1 = 1$ in uporabo Lagrangeovih multiplikatorjev pridemo do enačbe

$$Cv_1 = \lambda_1 u_1$$

v kateri u_1 in λ_1 predstavljata lastni vektor in pripadajočo najvišjo lastno vrednost matrike C.

Izračunamo lastne vrednosti in pripadajoče lastne vektorje kovariančne matrike. Lastni vektorji z najvišjimi lastnimi vrednostmi pojasnijo največ variance. Lastni vektor v_i z lastno vrednostjo λ_i pojasni λ_i variance. Ti vektorji – glavne komponente – so med seboj ortogonalni in tvorijo bazo novega prostora. Našim primerom zmanjšamo dimenzionalnost tako, da jih projiciramo v ta prostor. Projekcijska matrika $V \in \mathbb{R}^{k\times n}$ je sestavljena iz vektorjev $v_i^\intercal$, zloženih kot vrstice v matriko.

$$V = \begin{bmatrix} v_1^\intercal \\ v_2^\intercal \\ ... \\ v_k^\intercal \end{bmatrix}$$

Projekcija vseh primerov v nov prostor $P \in \mathbb{R}^{k \times m}$ se izrazi kot

$$P = V^\intercal X^\intercal$$

Število glavnih komponent (vektorjev) izberemo po potrebi – tipično bi radi, da je razložene vsaj 80% variance. Večje število izbranih komponent vodi do boljše rekonstrukcije podatkov, manjše pa do manj šuma v rekonstruiranih podatkih.

Metodo PCA lahko izvedemo tudi z uporabo singularnega razcepa (ang. *singular value decomposition, SVD*) [25]. Singularni razcep se izogne težavam z numerično nestabilnostjo, do katerih lahko zaradi majhnih števil v kovariančni matriki pride pri navadnem razcepu po lastnih vrednostih. Če želimo podatke projicirati v prostor z zelo malo dimenzijami (dve ali tri), je hitrejša možnost uporaba potenčne metode [28].

2.2.2 Večrazredno lestvičenje

Večrazredno lestvičenje (ang. *multidimensional scaling, MDS*) je nelinearna tehnika zmanjšanja dimenzionalnosti [12, 2]. Deluje po načelu ohranjanja razdalj med primeri v originalnem prostoru. Pri tem je vseeno, kako so bile te razdalje izračunane, kar lahko poenostavi situacijo, če primeri nimajo enostavnih numeričnih značilk (npr. besedila).

Algoritem kot vhod prejme matriko razdalj med primeri $\delta \in \mathbb{R}^{n \times n}$, kjer je δ_{ij} razdalja med i-tim in j-tim primerom, in želeno število dimenzij k. Algoritem najprej postavi primere v nov prostor in nato skuša pozicije optimizirati. Kot rešitev poišče n vektorjev $x_i \in \mathbb{R}^k$, kjer $d(x_i - x_j) \approx \delta_{ij}$ za vse

$i, j \in 1, \ldots, n$, d pa je poljubna funkcija razdalje (najpogosteje je to evklidska razdalja).

Cilj je, da so razdalje med primeri v projekcijskem prostoru čim bolj podobne tistim v originalnem prostoru. Za to je potrebno minimizirati funkcijo napetosti (ang. *stress*), ki je vsota kvadratov razlik med razdaljami.

$$J(X) = \sum_{i<j} (d(x_i, x_j) - \delta_{ij})^2$$

$X \in \mathbb{R}^{k \times n}$ je matrika, ki predstavlja postavitev točk, sestavljena je iz vektorjev x_i. Končna rešitev je takšna postavitev, pri kateri ima funkcija napetosti najmanjšo vrednost.

Funkcija napetosti je kompleksna, pri odvajanju po vseh vektorjih dobimo preobsežen sistem enačb. Lahko bi jo minimizirali z uporabo gradientnega spusta, vendar je ta metoda prepočasna in nezanesljiva (vedno obstaja možnost, da je rešitev le lokalni in ne globalni minimum). Obstaja boljši postopek: namesto njenega minimuma raje iščemo minimum neke funkcije, ki je zmeraj večja ali enaka tej. Ta postopek se imenuje majorizacija. Majorizacijo funkcije f zapišemo kot

$$\forall y \forall x : g(x, y) \geq f(x)$$

Pomembno je, da je $g(x, x) = f(x)$ in da znamo poiskati vrednost x, pri kateri $g(x, y)$ doseže minimum. Do rešitve pridemo iterativno. Začetno vrednost x_0 izberemo naključno. V i-tem koraku minimiziramo funkcijo g, kjer pridemo do x_{i+1} in velja $g(x_{i+1}, x_i) \leq g(x_i, x_i)$, saj smo minimizirali po prvem argumentu. Ker smo definirali funkcijo g kot zmeraj večjo ali enako f, velja

$$f(x_{i+1}) \leq g(x_{i+1}, x_i) \leq g(x_i, x_i) = f(x_i)$$

V vsaki iteraciji se tako približamo minimumu. Uporabljamo algoritem SMACOF (ang. *scaling by majorizing a complex function*), ki je kljub iterativnosti še vedno hitrejši od gradientnega spusta, bolje pa deluje, če imamo veliko primerov [28].

Začetne postavitve primerov v nov prostor so lahko naključne, lahko pa algoritmu podamo neko začetno postavitev, npr. rezultat metode glavnih komponent. Če začetni položaj inicializiramo z deterministčnim algoritmom, MDS postane deterministčen – ista začetna postavitev vodi k isti optimizaciji in istemu rezultatu.

2.2.3 Stohastična vložitev sosedov

Stohastična vložitev sosedov (ang. *t-distributed stochastic neighbor embedding, t-SNE*) je tehnika, podobna metodi MDS [18]. Glavna razlika med njima je v tem, da se metoda t-SNE osredotoča na ohranjanje razdalj med sorodnimi primeri (tistimi, ki so si blizu), medtem ko ji je za oddaljene primere precej vseeno. V nasprotju s tem skuša metoda MDS razdaljo ohraniti ne glede na njeno velikost. Če želimo pri vizualizaciji lažje videti skupine podobnih primerov, je t-SNE lahko boljša izbira.

Algoritem temelji na verjetnostni oceni podobnosti dveh primerov. Podobnost primerov (vektorjev) x_i in x_j je pogojna verjetnost $p_{i|j}$, da bi za primer x_i za soseda izbrali x_j, če bi sosede izbirali glede na Gaussovo verjetnostno porazdelitev, ki je centrirana na x_i. Pogojno verjetnost zapišemo kot

$$p_{i|j} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

kjer je σ_i varianca porazdelitve, ki je centrirana na x_i. Pri tem imajo bližnji primeri visoko verjetnost, da so sosedi, oddaljeni pa majhno. Za vse primere x_i definiramo dve verjetnostni porazdelitvi vseh sosedov: za originalni prostor porazdelitev P_i in za projekcijski prostor porazdelitev Q_i. Cilj je, da sta med seboj čim bolj podobni. Za mero podobnosti se uporablja Kullback-Leiblerjeva divergenca [14], ki pove, kako močno $q_{i|j}$ sledi $p_{i|j}$. Kriterijska funkcija je vsota divergence po vseh primerih:

$$J(\Theta) = \sum_i KL(P_i \| Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

Ocena sosednosti je zaradi nesimetričnosti Kullback-Leiblerjeve divergence tudi sama nesimetrična: bolj kaznuje, če sta primera v projekciji daleč, a bi morala biti blizu ($p \gg q$), kot če sta blizu, a bi morala biti daleč ($p \ll q$). Postavitev primerov v projekcijskem prostoru dobimo z uporabo gradientnega sestopa, ki poišče minimum kriterijske funkcije.

Za verjetnostno porazdelitev se je včasih uporabljala Gaussova porazdelitev, zaradi boljšega razlikovanja med oddaljenimi točkami pa je danes bolj znana rahlo drugačna, popravljena metoda, ki uporablja Studentovo t-porazdelitev. S tem dopustimo možnost, da so primeri iz srednje okolice v novem prostoru bolj oddaljeni. Popravljena metoda je izboljšana tudi v tem, da je ocena sosednosti simetrična.

Metoda je bolj občutljiva na lokalno strukturo primerov, prav tako lažje razkrije strukturo na več nivojih in podatke, ki so v nekakšnih gručah (teh seveda ne določi sama). Problem je v iskanju parametrov porazdelitve (μ in σ) in počasnosti metode, saj je zelo računsko zahtevna.

2.2.4 Primer uporabe

Da bi bil opis metod za zmanjšanje dimenzij bolj razumljiv, v tem razdelku prikažemo njihovo delovanje na primeru stotih slik. Iz nabora slik Caltech 101[4] [6] smo izbrali sto slik iz petih različnih kategorij: sloni, sončnice, metulji, kitare in pice. Na sliki 2.1 vidimo slike, razvrščene po kategorijah.

V orodju Orange [3, 4] smo pridobili vložitve slik z gradnikom Image Embedding in jih nato dali na vhod metodam za zmanjšanje dimenzij, ki smo jih opisali v prejšnjih razdelkih. Uporabili smo gradnika PCA in Manifold Learning (ta podpira metodi MDS in t-SNE). Na slikah 2.2, 2.3 in 2.4 so prikazani rezultati teh metod v razsevnem diagramu, ki ga prikaže gradnik Scatter Plot. Primeri iz iste kategorije so predstavljeni s točkami iste barve.

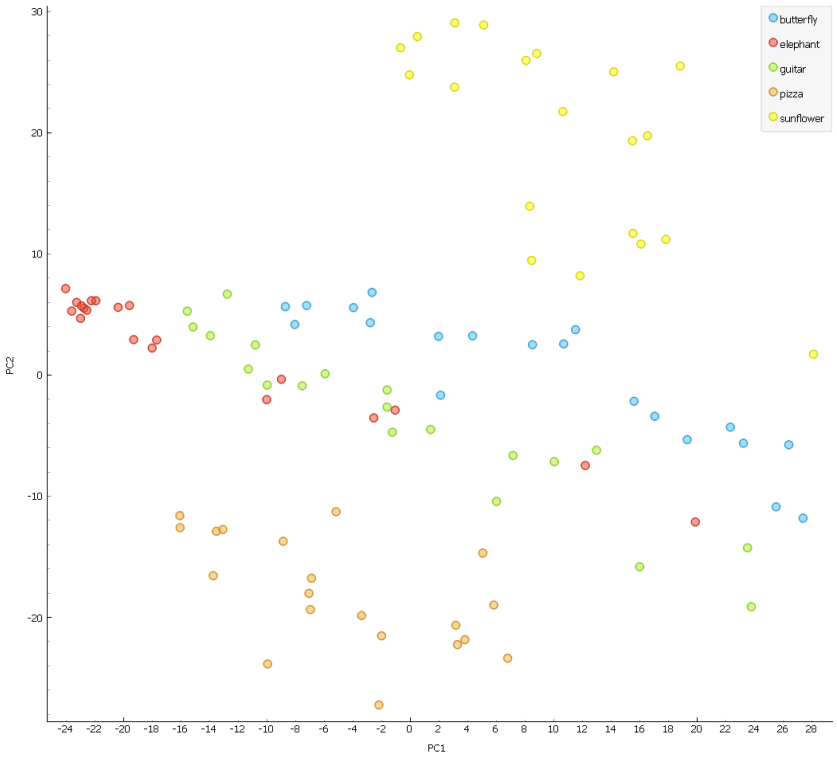
Pri vseh metodah vidimo, da je kombinacija pridobivanja vložitev in zmanjšanja dimenzij delovala, saj so primeri iz istih skupin tudi v 2D prostoru bolj skupaj. Vsaka od metod je dala drugačne rezultate, pri t-SNE

[4] `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`

Slika 2.1: Nabor stotih slik iz petih kategorij.

so skupine najbolj vidne, pri PCA pa najmanj. Metode ločujejo primere drugače, saj temeljijo na različnih postopkih in ne dajejo vse enako dobrih rezultatov na istih podatkih. Za uspešno implementacijo podobnostne mreže je pomembno, da se teh lastnosti zavedamo.

Na sliki 2.2, ki kaže rezultat metode PCA, se enostavno vidi dve smeri, v katerih so podatki najbolj razpršeni. Ti smeri ohranjata največ variance, v tem primeru je to 18%. Primeri iz istih skupin niso povsem v gručah.

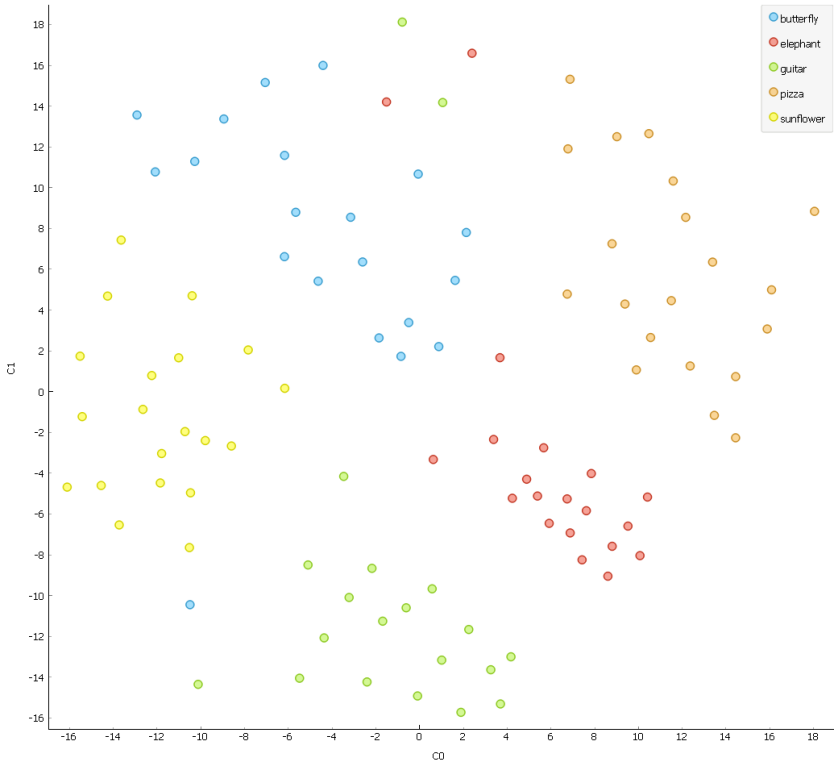


Slika 2.2: Rezultat metode PCA na naboru slik Caltech 101.

Pri rezultatu metode MDS na sliki 2.3 se lepše vidijo skupine, ki so z nekaj izjemami dobro ločene. Vidimo lahko, kako MDS ohranja razdalje med primeri, saj so podobni objekti iz različnih skupin ponekod blizu skupaj.

Na sliki 2.4, ki prikazuje je rezultat metode t-SNE, so skupine precej ločene in jasno vidne. Skoraj vsi primeri so razvrščeni v prave skupine, saj t-SNE primere iz srednje ali daljne okolice postavi daleč stran in ohranja

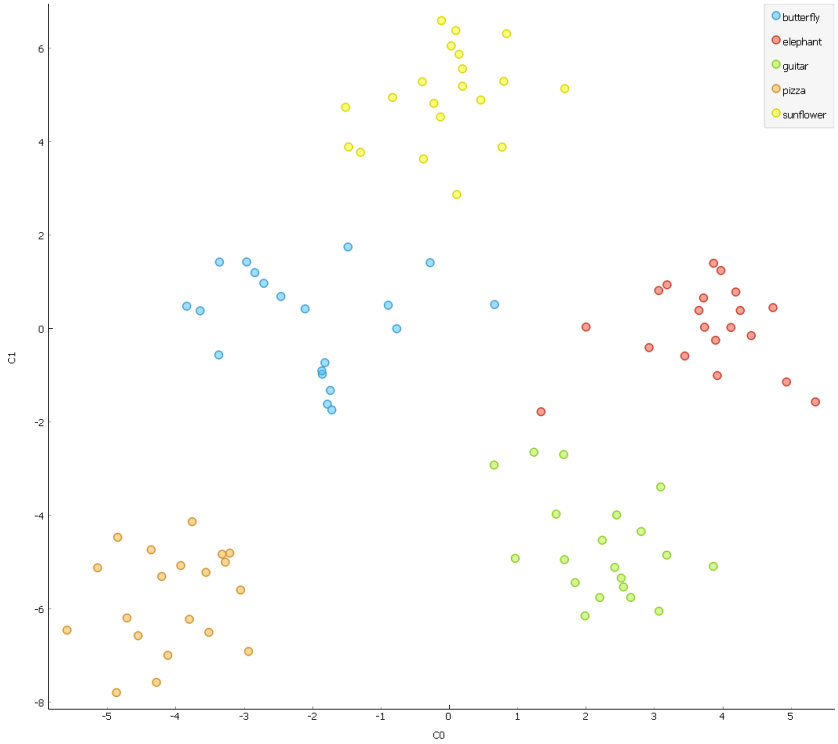
razdalje le za bližnje primere.



Slika 2.3: Rezultat metode MDS na naboru slik Caltech 101.

2.3 Maksimalno prirejanje

Metode zmanjševanja dimenzionalnosti dajo kot rezultat točke v nizkodimenzionalnem prostoru. Te točke pripadajo vhodni množici slik, kjer ima vsaka slika svojo točko. Ker si želimo slike prikazati brez prekrivanja, je najbolje, da jih postavimo v diskretno mrežo in vsaki dodelimo svojo celico. Za to je treba najti preslikavo med točkami v 2D prostoru in točkami na diskretni mreži. Ta preslikava mora biti čim boljša oz. imeti čim manjšo napako, saj hočemo imeti takšno postavitev slik v mreži, ki bi predstavljala podobnost med slikami. Pri tem ni nujno, da bo vsaka celica v diskretni mreži zapolnjena, saj je nabor slik lahko manjši od števila celic.



Slika 2.4: Rezultat metode t-SNE na naboru slik Caltech 101.

Ta problem lahko posplošimo na nalogo maksimalnega prirejanja [1]: imamo množico delavcev in množico nalog. Vsakega delavca lahko z neko ceno dodelimo katerikoli (a natančno eni) nalogi. Želimo si najti takšno dodelitev, da bo vsota cen najmanjša. Če imamo n delavcev in m nalog, cene dodelitev podaja $n \times m$ matrika C, v kateri je v i-ti vrstici in j-tem stolpcu cena dodelitve i-tega delavca na j-to nalogo.

Rešitev problema je dodelitev stolpcev na vrstice (oz. obratno), kjer ni nujno, da so vsi stolpci oz. vse vrstice dodeljene – to se zgodi samo v primeru, ko je $n = m$. Rešitev lahko predstavimo z binarno matriko X, kjer velja $X(i, j) = 1$, če je vrstica i dodeljena stolpcu j in obratno. Cena optimalne rešitve je

$$min \sum_i \sum_j C_{i,j} X_{i,j}$$

Problem lahko predstavimo tudi kot polni dvodelni (bipartitni) graf $G = (U, V; E)$, kjer je množica U sestavljena iz vseh delavcev in množica V iz vseh nalog, povezave pa so utežene s cenami dodelitev. Graf mora po definiciji biti poln, kjer so vse točke iz U povezane z vsemi iz V. V tem primeru je rešitev podmnožica množice povezav E, za katero velja, da mora vsebovati povezave do vseh točk iz manjše od množic U in V, nobena od točk pa ne sme biti izbrana več kot enkrat. Cena optimalne rešitve je najmanjša možna vsota vseh uteži v tej podmnožici.

Najbolj znan algoritem za reševanje problema je verjetno madžarska metoda, poimenovan tudi Kuhn-Munkres [13]. Algoritem je precej enostaven, deluje z vrstičnimi in stolpčnimi operacijami nad matriko C. Njegova časovna zahtevnost je polinomska ($O(n^3)$) in pri majhnih dimenzijah matrike je zadosti učinkovit. Za večje matrike lahko uporabimo bolj učinkovit algoritem – Jonker-Volgenant [10]. Tudi ta ima časovno zahtevnost $O(n^3)$, a je zaradi uporabe hevristik za izogibanje nepotrebnemu procesiranju v praksi dosti hitrejši. Poleg omenjenih dveh obstaja še veliko drugih algoritmov za reševanje problema maksimalnega prirejanja, aktivno pa se tudi išče nove in učinkovitejše metode.

Poglavje 3

Podobnostna mreža

V tem poglavju razložimo, kako smo metode iz prejšnjega poglavja uporabili v implementaciji pregledovalnika slik in gradnji podobnostne mreže. Naš cilj je bil poiskati tako predstavitev slik, pri kateri razdalja med slikami v mreži predstavlja podobnost teh slik – bliže kot so, bolj so si podobne. Predlagani algoritem sestavljajo metode za zmanjšanje dimenzionalnosti, normalizacijo podatkov, določanje velikosti mreže in izračun dodelitev v celice mreže.

3.1 Zmanjšanje dimenzionalnosti

V prvi fazi algoritma kot vhod dobimo vložitve slik iz nevronske mreže. Vektorje z visokim številom dimenzij moramo preslikati v dve dimenziji, za kar uporabimo enega od prej omenjenih algoritmov. V končni implementaciji je privzeta izbira metoda MDS, implementirani pa sta tudi možnosti uporabe metod PCA ali t-SNE.

3.2 Normalizacija podatkov

Vhod v to fazo so 2D vektorji, ki predstavljajo slike v 2D prostoru. Da lažje računamo postavitve v diskretno mrežo v kasnejši fazi, vektorjem koordinate omejimo na interval $[0, 1]$. Pri tem uporabimo metodo normalizacije

po razponu: v obeh dimenzijah odštejemo minimum in delimo z razponom $max - min$. Lahko bi uporabili drugačno metodo, ki bi standardizirala porazdelitev – odštela povprečno vrednost in delila s standardnim odklonom, vendar je za čim boljši približek originalnih vektorjev boljša prva možnost.

3.3 Določanje velikosti mreže

Ker hočemo preslikati točke iz zveznega prostora v diskretni, moramo določiti, koliko točk ima slednji. Tu imamo dve možnosti: ali dobimo želeno velikost mreže tudi kot vhod ali jo moramo iz števila slik določiti sami. Prva možnost je enostavnejša, saj moramo le preveriti, da lahko v mrežo takšne velikosti razporedimo vse slike. Pri drugi možnosti lahko privzeto velikost določimo kot $\lceil\sqrt{n}\rceil$, kjer je n število slik. To da kvadratno mrežo, ki je zagotovo zadosti velika za vse slike.

Zaradi lepšega prikaza je velikokrat bolje, če je okoli slik nekaj prostora in niso vse preveč skupaj, za to pa bi potrebovali malo večjo mrežo. Večja mreža nudi več možnosti za optimalno postavitev slik. Ker nismo hoteli vrednosti povečave mreže nastaviti na konstanto, smo se odločili za statistični pristop – vrednost povečave naj bo odvisna od porazdelitve točk – bolj narazen so, večja naj bo vrednost, za katero povečamo mrežo. Kot mero tega smo vzeli sploščenost (ang. *kurtosis*). Koeficient sploščenosti pove, kako velika sta repa porazdelitve. Določili smo, naj se mreža raztegne glede na koeficient sploščenosti γ_2 po formuli $2 * |\lceil\gamma_2\rceil|$ po obeh dimenzijah.

3.4 Izračun dodelitev v celice mreže

Zadnji korak v algoritmu je izračun položajev slik v celicah mreže. Do zdaj imamo izračunane normalizirane 2D vektorje in velikost mreže. Za izračun dodelitev uporabimo algoritem, ki rešuje problem maksimalnega prirejanja. Kot vhod moramo v ta algoritem poslati matriko cen, ki jo moramo izračunati. Množico „nalog“ že imamo – to so vektorji. Potrebujemo še

množico „delavcev“, ki bodo celice v mreži. Te lahko določimo tako, da ustvarimo množico $x * y$ točk, katerim obe koordinati linearno naraščata od 0 do 1 v x oziroma y korakih, pri čemer sta x in y dimenziji mreže. S tem dobimo eno točko za vsako celico v mreži.

Izračun matrike cen je po tem enostaven – izračunati je treba le razdalje (uporabili smo kvadratno evklidsko) med vsemi pari vektorjev in celic v mreži. Razdalje so v našem primeru cene dodelitev. Z matriko cen na vhodu algoritem reši problem maksimalnega prirejanja. Kot izhod dobimo razporeditve slik v celice mreže, kar je želeni rezultat našega algoritma.

Poglavje 4

Interaktivni pregledovalnik slik

V tem poglavju opišemo implementacijo pregledovalnika slik kot gradnika v Orangu. Začnemo pri funkcionalnostih, ki smo jih hoteli podpreti, pregledamo sorodna dela, razložimo, kako se gradnik vklaplja v Orange, podrobneje pa se posvetimo končnemu izdelku, uporabljenim tehnologijam in metodam ter strukturi implementacije, ki se deli na skriptni del in uporabniški vmesnik. Koda je ločena na dva logična dela zaradi delitve odgovornosti, enostavnosti ponovne uporabe in lažjega refaktoriranja, če oz. ko do tega pride. Na koncu omenimo tudi omejitve in možnosti za izboljšavo.

4.1 Cilji

V sklopu diplomske naloge smo želeli implementirati orodje, ki bi bilo sposobno interaktivnega pregledovanja slik. Slike bi moralo orodje med seboj primerjati in jih razporejati v dvodimenzionalno mrežo glede na podobnost. Tako bi omogočili enostaven pregled slik, grupiranih po podobnosti. Na prvem mestu je bila interaktivnost orodja, važno je, da je odzivno na uporabnikove interakcije in nudi dovolj informacij. Cilj je bil tudi orodje implementirati kot gradnik v Orangu. Gradnik mora biti povezljiv z ostalimi, že implementiranimi gradniki, ki mu lahko dajo vhodne podatke ali pa uporabljajo njegove izhodne podatke. Želeli smo, da bi se dobro vklopil v obstoječe

delokroge in imel dovolj možnosti za uporabo. Poleg osnovne želene funkcije smo podprli tudi dodatne, kot so izbiranje slik z možnostjo grupiranja, ki jih lahko nato pošljemo na izhod, poljubno določanje velikosti mreže in možnost ločenega vhoda podmnožice slik.

4.2 Sorodna dela

Pred začetkom implementacije orodja smo iskali obstoječe oz. podobne rešitve in raziskovalna dela s podobno tematiko. Od programskih rešitev smo našli samo posamezne metode, kar ni bilo problematično, saj smo jih lahko povezali v celoto sami in uporabili tiste, ki so nam bolj ustrezale. V nekaj člankih je bila obravnavana tematika podobnostne mreže slik oz. pregledovalnika slik, vendar so bili njihovi nameni različni, tehnologija pa pri vseh zastarela – članki so namreč stari vsaj 10 let, kar pomeni, da avtorji nevronskih mrež niso uporabljali in so za ekstrakcijo značilk iz slik uporabljali bolj preproste metode. V tem razdelku na kratko opišemo štiri dela, od katerih sta dve naravnani bolj eksperimentalno, dve pa sta imeli za cilj dejansko implementacijo.

4.2.1 Eksperimentiranje z vizualizacijami podobnosti slik

Rodden, Basalaj, Sinclair & Wood [20] so leta 1999 preizkušali vizualizacijo podobnosti slik kot orodje za brskanje po slikah. Za metriko podobnosti slik so uporabili histograme barv na sliki, ki so jih primerjali z razdaljo hi-kvadrat. Histograme, ki so predstavljali točke oz. vektorje v visokodimenzionalnem prostoru, so z metodo MDS preslikali v dve dimenziji. Namesto uporabe mreže celic so slike le postavili v 2D prostor na koordinate, izračunane z metodo MDS. S pomočjo testnih subjektov so preverjali, ali je ta postavitev boljša za iskanje dane slike kot naključna razporeditev v mrežo: merili so čas, ki so ga subjekti porabili, da so našli dane slike. V povprečju so subjekti

porabili 0,8 sekunde manj na sliko pri podobnostni postavitvi, kljub temu pa se jim je ta zdela zelo neintuitivna, dodatno pa jih je motilo prekrivanje slik, do katerega je prišlo pri nekaterih podobnih slikah. Avtorji so zato implementirali tudi postavitev v mrežo, pri čemer so primer, ko se je več slik preslikalo v eno celico, reševali s spiralnim iskanjem. To je sicer hitrejše od maksimalnega prirejanja, a ne zagotavlja optimalnega rezultata.

Isti avtorji [21] so leta 2001 izvedli podoben eksperiment. V njem so poleg vizualne podobnosti uporabili tudi anotacijsko podobnost – pri drugi so bile slike označene z besedami, iz katerih so tvorili vektorje in slike primerjali s kosinusno razdaljo. Vnovič so izvedli testiranje s pomočjo subjektov, le da so bili ti v tem primeru grafični oblikovalci. Skušali so ugotoviti, katera od mer podobnosti je boljša za brskanje po slikah, ki so jih razporedili z metodami iz prejšnjega članka. Naloga subjektov je bila iskanje nabora slik, ki bi bile primerne za uporabo v potovalnih vodičih za različne destinacije v treh različnih postavitvah: naključna, podobnostna v zveznem prostoru in podobnostna v mreži. Pri podobnostni postavitvi so imeli možnost uporabiti tako vizualno kot anotacijsko podobnost. Ugotovili so, da je subjektom najbolj ustrezala anotacijska postavitev v mreži, precej pa jim je tudi bila všeč možnost uporabe različnih mer podobnosti. Oba članka se posvečata predvsem človeškemu faktorju, saj so avtorji želeli dokazati, da so vizualizacije, ki uporabljajo podobnost slik, ljudem bolj všeč in jih lahko efektivno uporabljajo.

4.2.2 Interaktivni, semantični slikovni brskalniki

Yang et al. [26] so želeli implementirati semantični slikovni brskalnik. Slikam so dodelili veliko število značilk (npr. barvni histogrami, teksture, razporeditve barv) in jim dodali anotacije, ki so jih samodejno generirali. Slike so razporejali z uporabo metode MDS, izumili pa so svoj prikaz, ki je lažje prikazal visoko število dimenzij (anotacije in njihova povezanost). Podprli so besedno iskanje po anotacijah in brskalnik razširili za večjo interaktivnost (možnost premikanja in urejanja slik). Brskalnik so preizkusili s pomočjo

uporabnikov, ki so ga primerjali z navadnim urejenim brskanjem po mapah. Prednost brskalnika je bil visok nivo interaktivnosti in olajšanje prikaza več slik naenkrat, slabost pa ponekod slabe oz. napačne anotacije, ki so uporabnike zavedle. Članek je inoviral na področju interaktivnega prikaza slik, žal pa je tehnika samodejnega pridobivanja anotacij precej nezanesljiva in dandanes zastarela.

Nguyen & Worring [19] sta si zadala podoben cilj kot Yang et al. [26], vendar sta ciljala na optimalen sistem za pregledovanje velikih zbirk slik in njihovo anotiranje. Kot zahteve sta postavila dober pregled, vidljivost in ohranjanje strukture podobnosti slik. Preizkusila sta več različnih algoritmov za zmanjšanje dimenzij in na koncu izbrala metodo SNE, za boljši prikaz pa sta slike združevala v gruče. Različice sistema sta testirala s štetjem akcij, ki so jih navidezni uporabniki izvedli med delom, cilj pa je bil minimizacija števila teh.

4.3 Orange

Rešitev smo želeli integrirati v orodje za interaktivno podatkovno analitiko Orange [3]. Orange je zbirka orodij za strojno učenje in podatkovno rudarjenje, ki ga Fakulteta za računalništvo in informatiko razvija že od 1997 [4]. Odlikuje ga enostavnost in prijaznost do uporabnikov, saj je njegov grafični uporabniški vmesnik preprost in hitro priučljiv, ponuja pa tudi skriptni del za programsko uporabo. Orange je odprtokoden projekt, ki ga danes razvijajo člani Laboratorija za bioinformatiko na FRI. Aktualna verzija je na voljo na spletni strani[1], izvorna koda pa na storitvi GitHub[2]. Orangev grafični uporabniški vmesnik ponuja delovno površino, na katero postavljamo gradnike. Ti predstavljajo operacije, ki lahko sprejemajo in/ali oddajajo signale (to so lahko podatki ali nove operacije) ter so med seboj povezljivi. Vsak gradnik ima lahko več vhodov in izhodov.

[1] `https://orange.biolab.si`

[2] `https://github.com/biolab/orange3`

Poleg jedrnih funkcionalnosti ima Orange tudi dodatne v ločenih paketih, ki privzeto niso nameščeni z Orangem in jih uporabnik lahko namesti sam. Eden od teh je Image Analytics[3], ki podpira delo s slikami. V ta paket smo uspešno integrirali razvito rešitev kot gradnik Image Grid. Poleg tega so v paketu še trije drugi gradniki. Za delovanje Image Grid potrebuje dva druga gradnika: Import Images, s katerim uvozimo slike (ima tudi to uporabno funkcijo, da gradnik sam doda atribut s kategorijo, če so bile slike v več podmapah), in Image Embedding za pridobitev vložitev slik, ki jih prejme kot vhod. Trenutno je v gradniku Image Embedding na voljo šest nevronskih mrež, od katerih so tri namenjene splošni uporabi (naučene na zbirki ImageNet [5]), tri pa so bolj specializirane: namenjene so prepoznavi obrazov, napovedovanju slikarja, ki je avtor slike, in analizi celic kvasovk. Image Grid lahko vložitve slik z dodatnimi informacijami (ali so izbrane ali ne in v kateri skupini so) pošilja naprej v druge gradnike, npr. Image Viewer, ki služi kot splošen pregledovalnik slik.

4.4 Tehnologije in metode

Rešitev je tako kot celoten Orange implementirana v programskem jeziku Python[4]. Python je dinamično tipiziran, interpretiran visokonivojski programski jezik, ki se odlikuje po berljivosti izvorne kode in hitrosti razvoja v njem. Med drugim je izredno popularen na področju strojnega učenja, saj so v njem razvite knjižnice NumPy[5] (numerično računanje), SciPy[6] (znanstveno računanje) in scikit-learn[7] (strojno učenje), na katerih temelji tudi Orange. Vse tri knjižnice so odprtokodne in so že skoraj postale standard v okolju Pythona. Orangev grafični uporabniški vmesnik je razvit v ogrodju Qt[8], ki podpira več platform, kar zelo olajša delo razvijalcem in skrajša čas

[3] https://github.com/biolab/orange3-imageanalytics
[4] https://www.python.org
[5] http://www.numpy.org
[6] https://www.scipy.org
[7] http://www.scikit-learn.org
[8] https://www.qt.io

razvoja. Razvit je v jeziku C++, lahko pa ga prek vmesnikov uporabljamo tudi v drugih programskih jezikih – Orange za ta namen uporablja vmesnik PyQt[9].

V prejšnjem poglavju smo opisali potek algoritma in ga razdelili na korake. Želeli smo najti čim lažje dostopne in čim enostavnejše obstoječe implementacije vseh metod, saj bi bilo pisanje od začetka zelo zamudno. Izkazalo se je, da imamo v Orangu in SciPyu na voljo metode za vse štiri korake: Orange podpira vse tri opisane metode za zmanjšanje dimenzij, kjer interno kliče implementacije v knjižnici scikit-learn. Prav tako imamo v Orangu na voljo več vrst normalizacije, uporabili smo normalizacijo po razponu. SciPy vsebuje izračun koeficienta sploščenosti, ki ga potrebujemo za določanje velikosti mreže in madžarski algoritem, s katerim rešujemo problem maksimalnega prirejanja za dodelitev slik v celice. Odkrili smo, da je na žalost ta implementacija zelo neučinkovita, kar se je poznalo pri večjem številu slik: algoritem je za izračun dodelitev petstotih slik v mrežo 23x23 potreboval več kot dvajset sekund na štirijedrnem procesorju Intel i7-4700HQ pri 3,4 GHz, kar je predolgo in bi zelo negativno vplivalo na uporabnost gradnika. Zaradi tega smo iskali učinkovitejšo možnost, kar nam je uspelo – našli smo več implementacij algoritma Jonker-Volgenant [10] v obliki Pythonovskih knjižnic. Uporabili smo implementacijo z enostavnim vmesnikom[10], ki je približno desetkrat hitrejša od SciPyjeve.

Izmed treh metod, ki smo jih opisali v podpoglavju 2.2, smo morali izbrati metodo za zmanjšanje dimenzij. Kriterij za izbiro je bil primarno vizualen: hoteli smo, da postavitev izgleda čim bolj intuitivna. Poleg tega smo izmerili hitrost in povprečno ceno postavitve rezultatov metod v mrežo, rezultat testa je v tabeli 4.1. Cena postavitve oz. napaka je lahko povezana z vizualno privlačnostjo postavitve, saj večja napaka potencialno pomeni več slabo postavljenih slik. Vse tri metode so za izračun potrebovale približno enako dolgo, pri napaki pa vidimo, da je metoda MDS v prednosti. To si lahko

[9] `https://wiki.python.org/moin/PyQt`
[10] `https://github.com/gatagat/lap`

pojasnimo s slikami v razdelku 2.2.4. Metoda PCA na sliki 2.2 tudi različne slike postavi preveč skupaj, vidimo, da so preveč zgoščene v majhnem prostoru. Zaradi tega pride do velike napake pri postavljanju slik v mrežo, saj lahko ena celica v mreži vsebuje največ eno sliko. Metoda t-SNE na sliki 2.4 po drugi strani zelo močno ločuje skupine slik, vendar ne ohrani razdalj med bolj oddaljenimi primeri. Ker vsak primer v skupini zaradi omejene velikosti mreže ne more dobiti svoje celice, je napaka velika. Metoda MDS v tej navezi deluje najbolje, saj postavi podobne primere skupaj, a vseeno ohranja razdaljo med vsemi pari primerov. To vodi v boljšo preslikavo, saj so primeri enakomerneje porazdeljeni po prostoru. Ker mreže privzeto niso veliko večje od minimalne možne mreže, je metoda MDS boljša od t-SNE, ki bi v večjih mrežah lažje razpostavila slike v gručah. Dodaten bonus metode MDS je determinističnost pri inicializaciji s PCA. Pri istem naboru slik smo želeli dobiti podobno postavitev, ki se čim bolj ohranja pri spreminjanju velikosti mreže.

metoda	čas[s]	napaka
PCA	0,4948	8,9774
MDS	0,4178	1,8503
t-SNE	0,5920	5,7249

Tabela 4.1: Rezultati testa na stotih slikah iz petih kategorij v mreži 12x12. Test smo pognali stokrat in rezultate povprečili.

4.5 Skriptni del

Skriptni del predstavlja algoritem, ki izračuna postavitev slik v mreži. Implementiran je v razredu `ImageGrid`. Razred vsebuje dve metodi, ki predstavljata vmesnik za poganjanje algoritma. Prva je `__init__`, ki ji podamo vložitve slik. Metoda inicializira atribute razreda, preslika vložitve v 2D prostor, jih normalizira in shrani kot atribut. Če se vhod v gradnik ne spremeni,

se tako izognemo ponovnemu zmanjševanju dimenzij, ki je lahko časovno zahtevno. Druga metoda je `process`, ki izračuna postavitve slik v mrežo. Kot argument lahko prejme velikost mreže – če ta ni podana, jo izračuna metoda sama. Metoda se pokliče ob vsaki spremembi velikosti mreže, zato je za kvaliteto uporabniške izkušnje pomembno, da se izvrši čim hitreje. Razred poleg dveh javnih metod vsebuje še nekaj pomožnih, ki jih ti dve kličeta (npr. pretvorba razporeditve slik iz indeksnega v tabelni zapis).

```
def _get_assignments(self, data):

    # create grid with linearly spaced coordinates
    # reshape to list of coordinates (size_x*size_y x2)
    grid = np.dstack(np.meshgrid(
        np.linspace(0, 1, self.size_x, endpoint=False),
        np.linspace(0, 1, self.size_y, endpoint=False)))
    .reshape(-1, 2)

    # get squared euclidean distances between all pairs of grid
       points and embeddings
    cost_matrix = cdist(grid, data, "sqeuclidean")

    cost, grid_indices, row_indices = lapjv(cost_matrix,
       extend_cost=True)
    assignments = grid[row_indices]

    return cost, grid_indices, assignments
```

Programska koda 4.1: Računanje dodelitev slik v celice mreže.

V izseku kode 4.1 je prikazana funkcija, ki iz seznama normaliziranih 2D vektorjev, ki predstavljajo slike, izračuna postavitve v mrežo. Uporablja funkcije iz knjižnic NumPy in SciPy ter modul za reševanje maksimalnega prirejanja, zaradi katerih je rešitev elegantnejša in enostavnejša.

Funkcija najprej z `np.linspace` ustvari dva vektorja dolžine x in y (ti števili predstavljata velikost mreže), ki vsebujeta enako razmaknjenih x oz. y števil na intervalu $[0, 1)$. Nato ju z `np.meshgrid` spremeni v matriki koor-

dinat, ki sta sestavljeni iz y-krat ponovljenega prvega in x-krat ponovljenega drugega vektorja. Ti matriki z `np.dstack` in `reshape` združi v matriko dimenzij $x * y \times 2$, imenovano `grid`. Ta vsebuje vse možne pare koordinat, ki označujejo celice v mreži. S funkcijo `cdist` med vsemi pari točk v mreži in 2D vektorji, ki predstavljajo slike, izračuna matriko kvadratnih evklidskih razdalj `cost_matrix`. Matriko pošlje v funkcijo `lapjv`, ki reši problem maksimalnega prirejanja in vrne optimalno razporeditev slik s ceno.

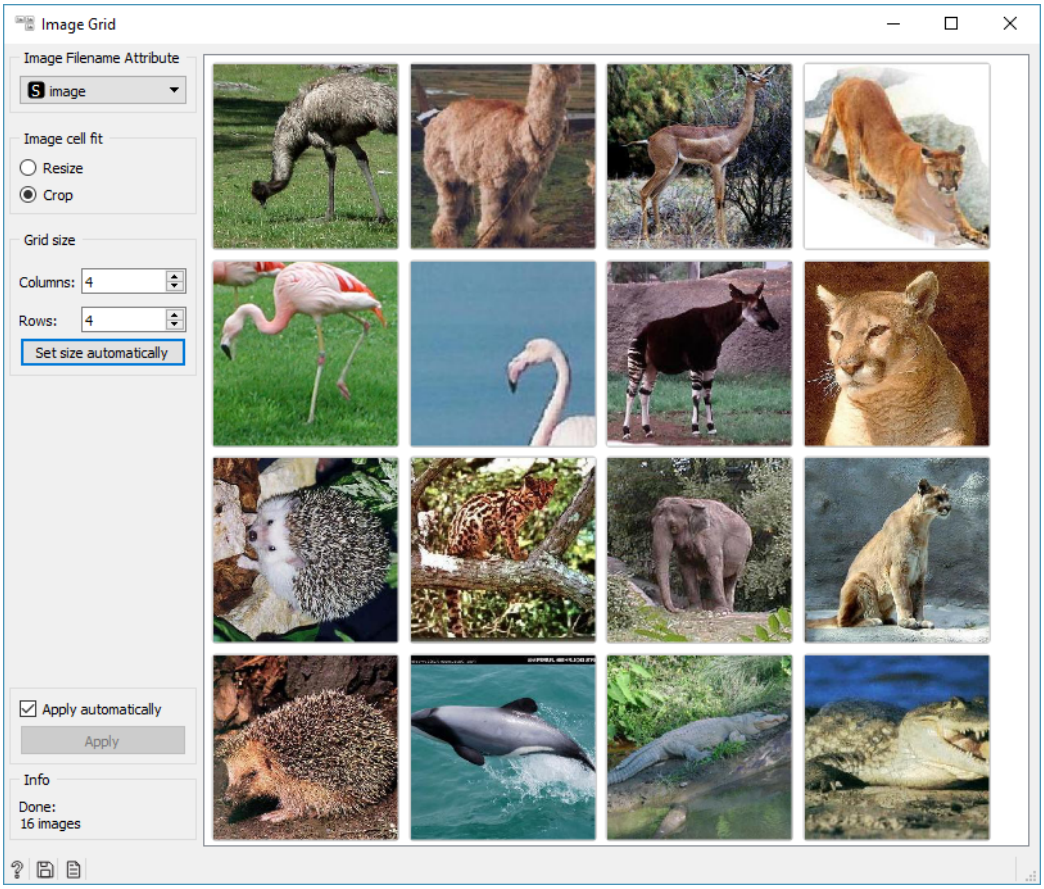
4.6 Uporabniški vmesnik

Uporabniški vmesnik gradnika je implementiran po zgledu drugih gradnikov v Orangu. Kot osnovo za implementacijo smo uporabili kodo gradnika Image Viewer, saj sta oba gradnika oblikovana kot mreža slik, le da Image Viewer deluje kot navaden brskalnik oz. pregledovalnik slik. Večino funkcionalnosti smo morali prilagoditi, nekaj pa smo lahko ohranili skupnih in ponovno uporabili že preverjeno kodo.

Glavni razred, ki opisuje gradnik, se imenuje `OWImageGrid`. Ta deduje iz razreda `OWWidget`, ki je nadrazred vsem gradnikom v Orangu in vsebuje skupne funkcionalnosti. V dedovanem razredu kot atribute nastavimo ime, opis in ikono ter ob inicializaciji ustvarimo komponente vmesnika. Mrežo, v katero razporedimo slike, implementiramo z nekaj dodatnimi razredi. Razred `GraphicsThumbnailGrid` skrbi za razpored slik v mreži in mehanizme selekcije. Posamezno sliko predstavlja razred `GraphicsThumbnailWidget`, ki skrbi za izris slike in možnih efektov. Oba razreda dedujeta iz Qtjevega `QGraphicsWidget`, ki implementira nizkonivojske funkcionalnosti in jih izpostavi prek visokonivojskega vmesnika.

Na sliki 4.1 je prikazan gradnik Image Grid. Uporabniški vmesnik je razdeljen na dva dela: kontrolni del na levi in mrežo na desni. Kontrolni del vsebuje vse možnosti, ki jih lahko nastavljamo. Po vrsti od zgoraj navzdol so to izbira atributa slike, ki vsebuje ime datoteke, izbira prilagoditve slike v celici (če ni kvadratna, jo zmanjšamo ali obrežemo) in nastavitve velikosti

mreže, poleg katerih je tudi gumb za ponastavitev (*Set size automatically*). V spodnjem delu je še nekaj komponent: potrditveno polje *Apply automatically* določa, ali naj gradnik takoj po končanem procesiranju pošlje podatke na izhod, okvir *Info* prikaže informacije o statusu in potencialnih napakah, gumbi skrajno na dnu pa služijo prikazu pomoči, shranjevanju slike mreže in izvozu poročila o gradniku.



Slika 4.1: Grafični uporabniški vmesnik gradnika Image Grid. Slike so obrezane za boljšo zapolnitev mreže.

4.7 Nadaljnje možnosti

Rešitev, ki smo jo v sklopu diplomske naloge razvili, ima nekaj omejitev. Pri razporejanju je omejena na dvodimenzionalno pravokotno mrežo, čeprav bi ponekod verjetno prišli do kakšne boljše vizualizacije na drug način (npr. heksagonalna mreža ali zvezni 2D prostor). Poleg tega lahko vsaka slika zasede le eno celico – lahko bi ugotovili, katera slika je za katero skupino najbolj reprezentativna, in jo prikazali v večjem formatu. To bi lahko razširili na ročno izbiro glavnih slik, ki bi bile večje, okoli njih pa bi lahko z gručenjem oblikovali skupine.

Podprli bi lahko tudi dve bolj enostavni funkciji. Prva je možnost ročnega premikanja slik po mreži, saj se zaradi omejene velikosti mreže lahko zgodi, da maksimalno prirejanje za kakšno sliko ne najde dobre pozicije. Možni sta tudi situaciji, ko metoda MDS ne izračuna dobre postavitve ali ko vložitev slike ni reprezentativna (do tega lahko v konvolucijskih mrežah pride, če je slika obdelana s filtri). Druga funkcionalnost bi bila uporabna pri večjem številu slik: povečevanje posameznega dela mreže, saj gradnik vedno prikaže vse slike naenkrat, te pa so lahko pri veliki mreži zelo majhne. Večje število slik je problematično tudi zaradi visoke časovne zahtevnosti algoritmov zmanjševanja dimenzij in maksimalnega prirejanja.

Poglavje 5

Primeri uporabe

V tem poglavju na nekaj primerih prikažemo uporabo gradnika Image Grid. Uporabimo več različnih naborov slik in demonstriramo v prejšnjem poglavju opisane funkcije. Poleg tega prikažemo tudi vklapljanje gradnika v delokrog z drugimi gradniki.

5.1 Image Grid

Gradnik lahko prejme dva vhoda: vložitve slik in opcijsko podmnožico teh. Če je prisoten tudi drugi vhod, bodo slike, ki niso v podmnožici, v mreži delno prosojne, zaradi česar je podmnožica bolj opazna. Poleg dveh vhodov ima tudi dva izhoda: vse slike in izbrane slike. Na prvi izhod zmeraj pošlje vse slike, ki jih prejme, doda jim stolpec, ki poda informacijo, če je slika izbrana ali ne in v kateri skupini je. Na drugi izhod pošlje le izbrane slike z enakim dodanim stolpcem.

Slike lahko izbiramo v skupinah – s pritiskom na tipko *shift* ob izbiranju ustvarimo nove skupine. Slike iz iste skupine imajo ozadje pobarvano z isto barvo. Ker so slike v mreži predstavljene v majhnem merilu, pride prav možnost povečave. S pritiskom na preslednico aktiviramo predogled posamezne slike v posebnem oknu. Za hitrejše pregledovanje se lahko s smernimi tipkami pomikamo po mreži in spreminjamo fokusirano sliko, ki je prikazana

v oknu.

5.2 Živali

Na sliki 5.1 je prikazan delokrog z več gradniki in gradnik Image Grid s slikami živali. V gradnik Image Grid smo kot drugi vhod poslali podmnožico slik, zaradi česar je preostanek slik polprosojen. Izbrane slike smo poslali v gradnik Image Viewer, vse slike pa v gradnika Manifold Learning, ki požene metodo MDS, in Scatter Plot, v katerem lahko vidimo razsevni diagram vložitev v 2D prostoru. V Image Embedding smo izbrali mrežo Inception-v3, ki odlično prepozna živali na slikah, saj v mreži vidimo grupiranje živali istih vrst.

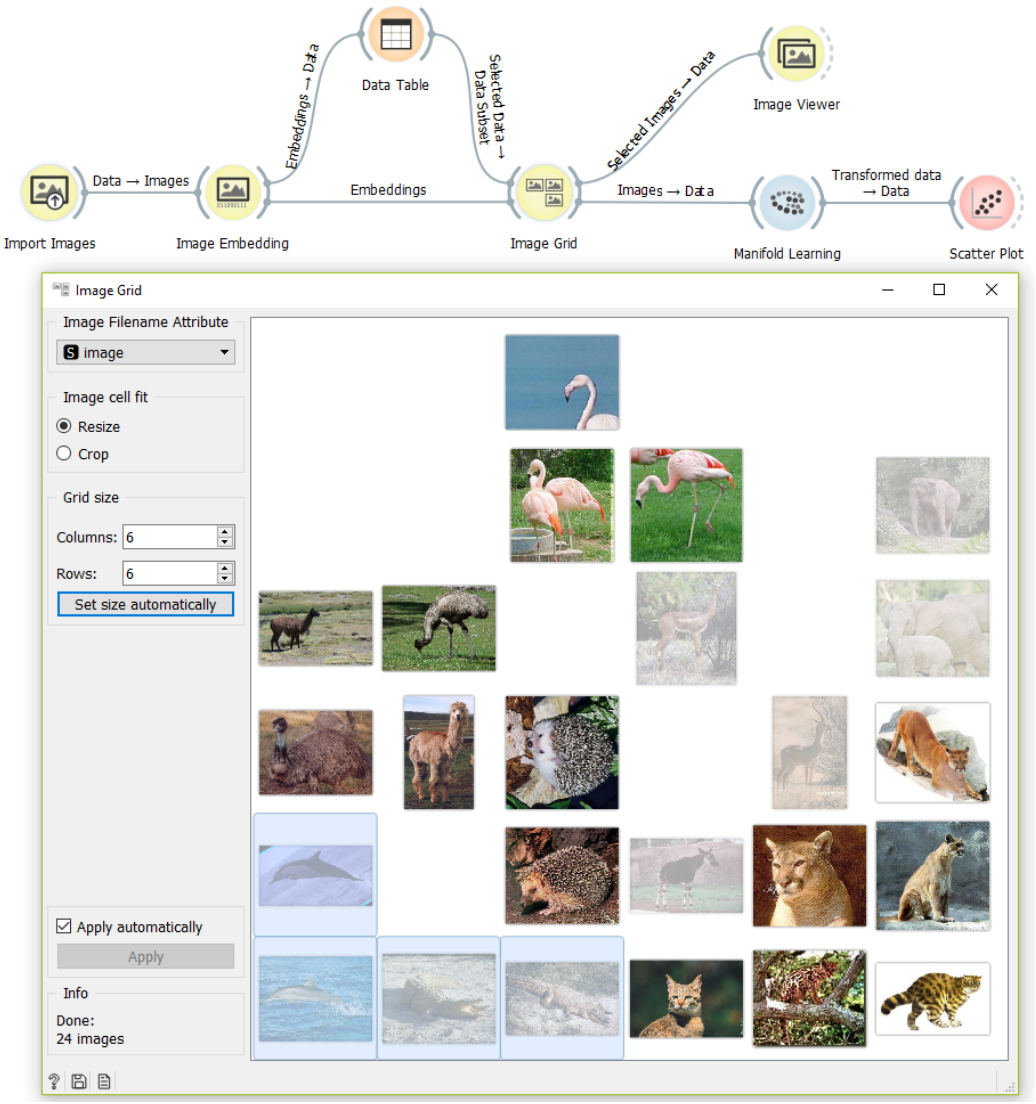
5.3 Obrazi

Na sliki 5.2 je prikazana razporeditev šestdesetih slik obrazov treh oseb. Prav tako je na sliki demonstrirana uporaba izbiranja skupin, ki omogoči izbiro več elementov, ločenih po skupinah. Vsaka skupina je označena s svojo barvo. V mreži smo označili tri skupine, kjer vsaka vsebuje slike ene osebe. Za pridobitev vložitev slik smo uporabili mrežo openface. Vidimo, da so z nekaj izjemami slike iste osebe res postavljene skupaj.

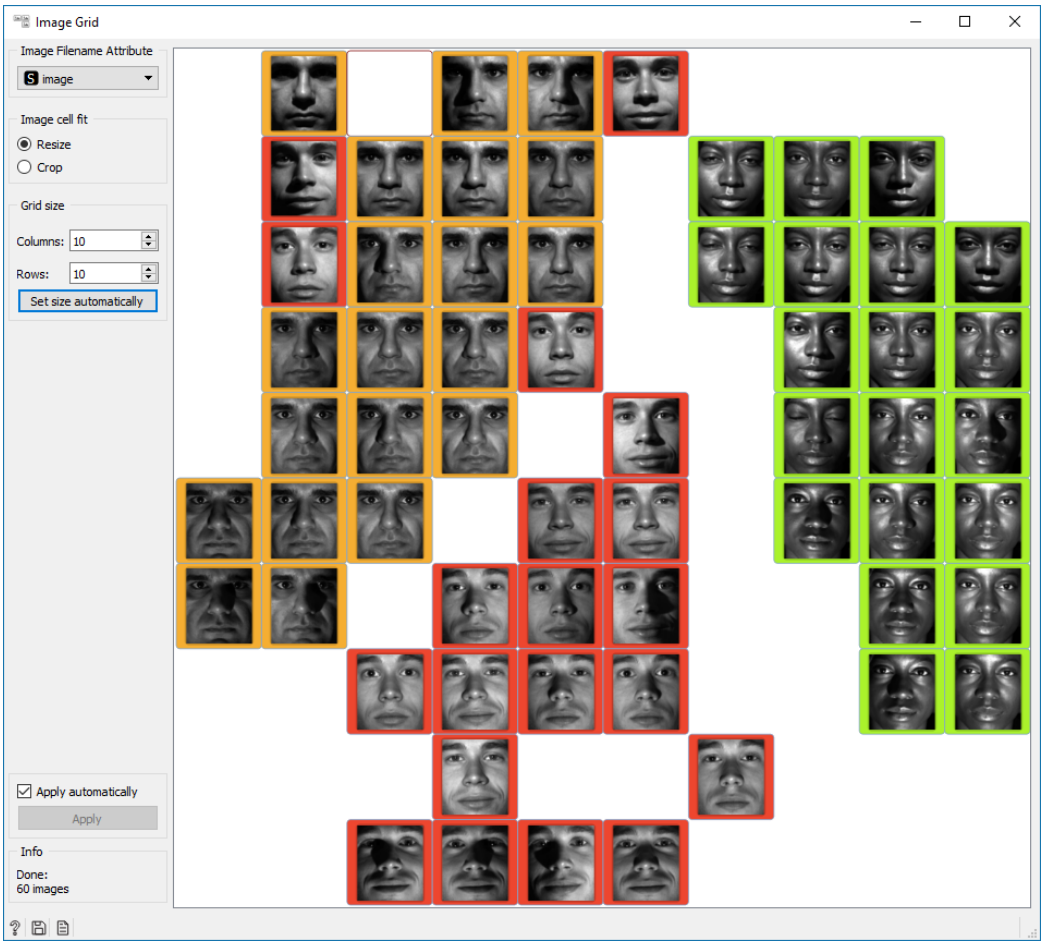
5.4 Ročno napisane številke

Na sliki 5.3 smo v mrežo razporedili sto slik iz podatkovne baze ročno napisanih številk MNIST[1]. Preizkusili smo mreže Inception-v3, VGG-16 in VGG-19, najboljši rezultat je dala mreža Inception. Razporeditev ni najboljša možna, saj posameznih številk ne grupira dobro. Večinoma skupaj stoji le nekaj slik iste številke, najbolje grupirane pa so 1, 4 in 7, ki tvorijo največje skupine. Do takšnega rezultata lahko pride, če mreža ni bila naučena na podobnih slikah.

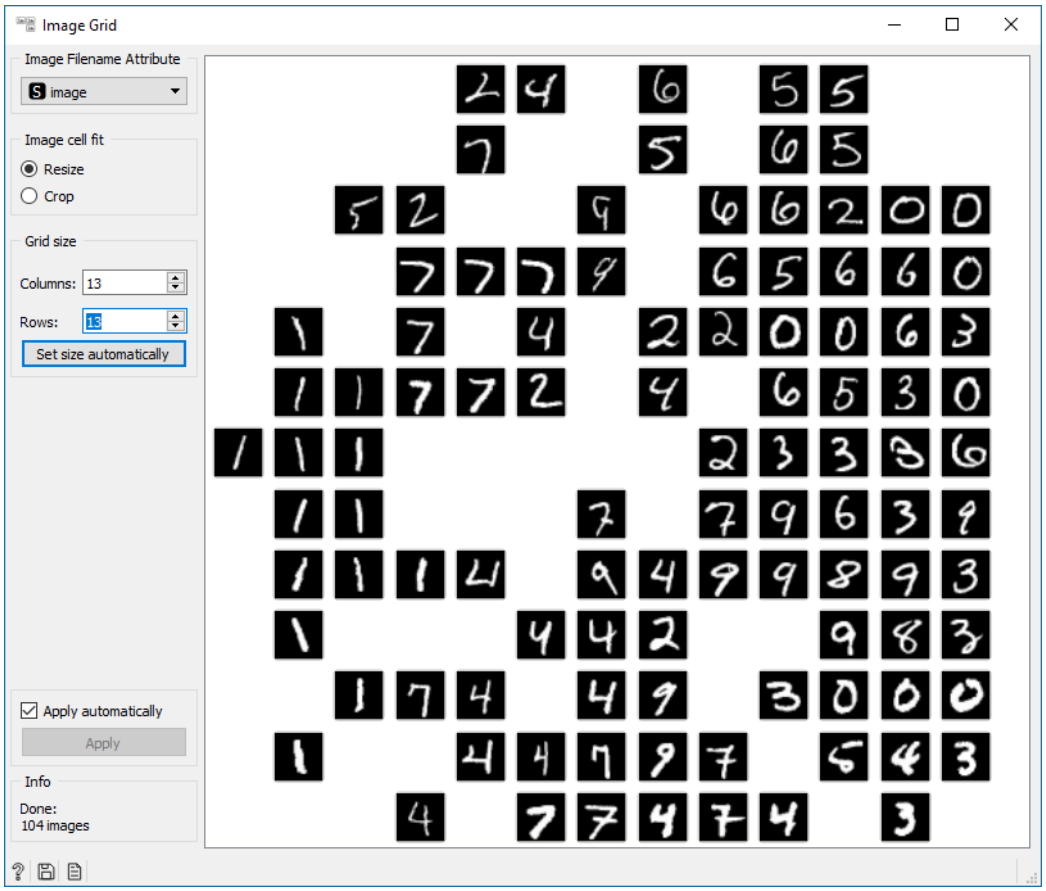
[1] `http://yann.lecun.com/exdb/mnist`



Slika 5.1: Delokrog z več gradniki in gradnik Image Grid s slikami živali.



Slika 5.2: Obrazi.



Slika 5.3: Ročno napisane številke iz baze MNIST.

Poglavje 6

Zaključek

V diplomskem delu smo implementirali interaktivni pregledovalnik slik, ki omogoča napredno vizualizacijo slik z razporeditvijo v podobnostno mrežo. V predlagani rešitvi smo povezali uporabo globokega učenja s tehnikami zmanjšanja dimenzij in pokazali, da tako lahko ustvarimo učinkovite semantične vizualizacije slik. Z implementacijo smo razširili funkcionalnost Orangevega okolja, kjer smo jo dodali v paket Image Analytics. V tem paketu je na voljo kot interaktiven gradnik z imenom Image Grid. Vsa koda rešitve je prosto dostopna v repozitoriju GitHub[1].

Gradnik Image Grid uporabljamo preko preprostega uporabniškega vmesnika. Podpira enostavno povezovanje z ostalimi gradniki v paketu in s tem izboljša funkcionalnosti paketa – slike lahko iz gradnika pošljemo kot vhod v druge gradnike, kar odpre veliko možnosti za nadaljnjo obdelavo. Cilj gradnika je v tem, da najde podobnosti med slikami in to z vizualizacijo predstavi uporabniku, s čimer mu omogoči dober pregled nad podobnimi slikami na razumljiv način. To pride še posebej do izraza v primerih, ko človek sam težko prepozna podobnost slik.

Rešitev bi lahko izboljšali in razširili na več načinov. Smiselno bi bilo podpreti več možnih postavitev slik, saj je trenutno edina možnost pravokotna mreža, poleg te bi lahko podprli npr. heksagonalno mrežo ali prosto

[1] `https://github.com/biolab/orange3-imageanalytics`

postavitev v 2D prostor. Gradnik prikaže vse slike naenkrat ne glede na njihovo število, kar lahko pomeni zmanjšano preglednost. Naprednejši, dinamični načini vizualizacije (npr. osnovani na gručenju) bi to olajšali, saj bi lahko prikaz sproti prilagajali glede na uporabnikove vnose.

Literatura

[1] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[2] Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. Chapman and Hall/CRC, 2000.

[3] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, et al. Orange: data mining toolbox in python. *The Journal of Machine Learning Research*, 14(1):2349–2353, 2013.

[4] Janez Demšar and Blaž Zupan. Orange: Data mining fruitful and fun-a historical perspective. *Informatica*, 37(1), 2013.

[5] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[6] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern analysis and Machine Intelligence*, 28(4):594–611, 2006.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Kevin Gurney. *An introduction to neural networks*. UCL Press, 1997.

[9] Ian Jolliffe. Principal component analysis. In *International Encyclopedia of Statistical Science*, pages 1094–1096. Springer, 2011.

[10] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4), Dec 1987.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[12] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[13] Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[14] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[18] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[19] Giang P Nguyen and Marcel Worring. Interactive access to large image collections using similarity-based visualization. *Journal of Visual Languages & Computing*, 19(2):203–224, 2008.

[20] Kerry Rodden, Wojciech Basalaj, David Sinclair, and Kenneth Wood. Evaluating a visualisation of image similarity. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–276. ACM, 1999.

[21] Kerry Rodden, Wojciech Basalaj, David Sinclair, and Kenneth Wood. Does organisation by similarity assist image browsing? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 190–197. ACM, 2001.

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[24] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[25] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In *A Practical Approach to Microarray Data Analysis*, pages 91–109. Springer, 2003.

[26] Jing Yang, Jianping Fan, Daniel Hubball, Yuli Gao, Hangzai Luo, William Ribarsky, and Matthew Ward. Semantic image browser: Bridging information visualization with automated intelligent image analysis. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 191–198. IEEE, 2006.

[27] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[28] Blaž Zupan. Projekcije in zmanjševanje dimenzionalnosti podatkov. Dosegljivo: https://github.com/BlazZupan/ozp-zapiski/raw/master/pdfs/projekcije.pdf. [Dostopano: 1. 6. 2018].