Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Vesel

Gručenje umetniških slik na podlagi njihovih značilnic

diplomsko delo

univerzitetni interdisciplinarni študijski program prve stopnje računalništvo in matematika

Mentor: doc. dr. Luka Šajn

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.
Slike so narejene s pomočjo orodja Mathematica.

Fakulteta za matematiko in fiziko ter Fakulteta za računalništvo in informatiko izdajata naslednjo nalogo:

Tematika naloge:

Študent naj preizkusi možnosti primerjave slik s pomočjo algoritmov računalniškega vida. Preizkusi naj dva glavna načina. Prvi naj se osredotoča na gručenje slikarjev na podlagi preslikave obraznih značilnic iz fotografij na umetniško delo. Pri drugem načinu naj razišče možnosti gručenja slikarjev glede na barvo, teksturo in ostale izmerljive značilnice njihovih del.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nejc Vesel, z vpisno številko **63110359**, sem avtor diplomskega dela z naslovom:

Gručenje umetniških slik na podlagi njihovih značilnic

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luka Šajna,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 31. 08. 2015 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
OCR	Optical Character Recognition	Optično prepoznavanje znakov
HMM	Hidden Markov Model	Prikriti markovski model
HMT	Hidden Markov Tree	Prikrito markovsko drevo
SVM	Support Vector Machines	Metoda podpornih vektorjev
ASM	Active Shape Models	Aktivni modeli oblik
ASD	Allowable Shape Domain	Dovoljena domena oblik

Tabela 1: Tabela uporabljenih kratic

Povzetek

V tej diplomski nalogi želimo preizkusiti metodo, ki nam omogoči, da s pomočjo računalniške analize sliko pripišemo določenemu slikarju. Testiramo dva načina. Pri prvem pristopu želimo identificirati slikarja glede na način, s katerim preslika človeške obrazne poteze iz fotografije na naslikan portret. Zanima nas, ali so razlike v obraznih razmerjih na fotografiji in sliki statistično pomembne.
Pri drugi metodi vsako sliko opišemo z vektorjem značilnic. Značilnice obsegajo barvo, teksturo in dimenzije slike, katerih kombinacija tvori vektor značilnic. Princip testiramo na 3 slikarjih z različnimi stili. Za vsakega od njih imamo množico desetih testnih slik. Zanima nas, ali lahko z gručenjem sliko pravilno pripišemo slikarju samo na podlagi teh vektorjev značilnic.

Ključne besede: računalniški vid, umetnost, detekcija obraza, primerjava umetniških slik, klasifikacija, klasifikacija umetnikov.

Abstract

In this thesis we are trying to discover a method that allows us to attribute a painting to a particular artist with the help of image analysis. We are testing two methods. In the first one, we are trying to identify the style of a painter by analysing the way in which he translates a human face from a photograph into a painting. We are testing whether the differences on facial proportions in photographs and paintings are statistically significant.
With the other method, we describe every painting with a set of features. The features look at the image color, texture and dimensions to form a feature vector. We test this on 10 pictures for each of the 3 painters with different styles. We are trying to test, whether we can correctly attribute these paintings to a painter just with these feature vectors.

Keywords: computer vision, art, face detection, artwork comparison, classification, artist classification.

Poglavje 1

Uvod

V diplomski nalogi želimo preizkusiti metodo, ki bi nam omogočala identificiranje slikarja glede na način, s katerim preslika obrazne poteze. Metodo želimo testirati na dveh slikarjih, Thomasu Eakinsu in Rudolfu Španzelu. Oba slikarja ustvarjata v realističnem slogu, kar je primerno za metode avtomatskega zaznavanja ključnih točk na obrazih naslikanih oseb. Zanima nas, ali se obrazna razmerja oseb iz fotografij in naslikanih portretov statistično razlikujejo. To bomo preizkusili s statističnim testom. Če bomo ugotovili da so razlike signifikantne, bomo poskusili slikarje gručiti glede na razlike v preslikavah. To bi nam lahko služilo kot pomoč pri identificiranju slik, ki jim želimo pripisati avtorja. V primeru, da ugotovimo, da metoda ni uspešna, bomo poskusili ponajti druge metode, s katerimi bi lahko avtomatsko gručili slikarje glede na njihove slike. Prav tako bomo v diplomski nalogi predstavili teoretično ozadje nekaterih postopkov in metod, ki smo jih uporabili za pridobivanje rezultatov oz. uspešnih značilnic. Če ta metoda ne bo uspešna, se bomo problema lotili na drugačen način. Vzeli bomo deset slik znanih slikarjev (Vincent Van Gogh, Jackson Pollock, Edgar Degas) in na podlagi določenih lastnosti njihov slik (povprečna barva, tekstura, velikost ipd.) izračunali značilnice. Slike bomo potem na podlagi teh značilnic poskusili gručiti.

Poglavje 2

Računalniški vid in metodologija

Računalniški vid je področje računalništva, ki se ukvarja z zaznavanjem in klasifikacijo objektov na računalniških slikah. Cilj je razumeti in analizirati slike ter objekte na njih. To lahko delamo v realnem času (npr. sledenje objektov na videoposnetkih) ali pa uporabljamo počasnejše in natančnejše metode, s katerimi lahko izvajamo bolj zahtevne operacije. Računalniški vid se v današnjem svetu uporablja v različne namene, kot so optično zaznavanje črk (OCR), strojno pregledovanje za potrebe kvalitete kontrole, izdelovanje 3D modelov terena iz satelitskih slik, uporaba v medicini, aplikacije za varnost v prometu ipd.

2.1 OpenCV

OpenCV je odprtokodna knjižnica za računalniški vid in strojno učenje. Vsebuje algoritme, ki jih lahko uporabljamo za identifikacijo objektov, detekcijo obrazov, sledenje kameri, sledenje objektom na videoposnetkih in podobno. Uporablja jo veliko večjih in manjših podjetji po celem svetu in je postala de-facto industrijsko standardna knjižnica za računalniški vid. Knjižnica je napisana v programskem jeziku C++ in je podprta na operacijskih sistemih

Windows, Linux, Android in Mac OS. Vsebuje več kot 500 različnih algoritmov za analiziranje in pridobivanje informacij s slik.

2.2 Uporabljena orodja

Za zaznavanje obraznih značilnic na slikah smo uporabili STASM [8], ki ga podrobneje predstavimo v nadaljevanju. Program nam ponudi možnost izbire zaznave različnega števila ključnih točk na obrazu. Za naše potrebe smo izbrali možnost zaznave 20 ključnih točk na vsaki sliki, ki definirajo obraz osebe. Teh dvajset točk je osnovanih na BioID [4]. Te točke so:

- leva in desna zenica,
- levi in desni kotiček ustnic,
- notranji in zunanji konec leve in desne obrvi,
- levo in desno teme,
- notranji in zunanji konec levega in desnega očesa,
- konica nosu,
- leva in desna nosnica,
- sredinska točka na zunanjem delu zgornje in spodnje ustnice.

Program nam lokacije ključnih točk na sliki zapiše v .log datoteko. Za delo s to datoteko smo zaradi preprostosti izbrali programski jezik **Python** [3] in knjižnico za numerično računanje **NumPy** [2].

Poglavje 3

Drugi pristopi

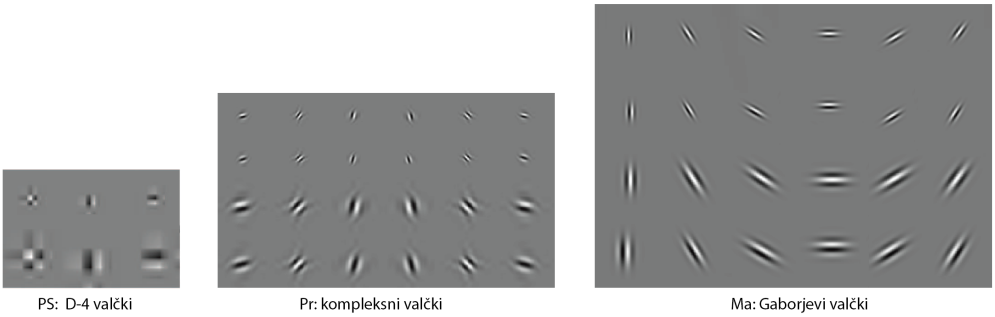
Če bo delovanje naše metodo uspešno, bi jo poizkušali uporabiti v namene avtentikacije in za pomoč pri določanju avtorja slikarskih del. To je iz stališča računalniškega vida zahteven problem, saj ljudje, ki avtenticirajo slikarska dela uporabljajo različne metode, kot so analiza kompozicije, simbolika v danem delu, ocenitev starosti platna, poznavanje zgodovinskega ozadja ipd. Z računalniškimi programi se moramo problema lotiti na drugačen način. V nadaljevanju bom predstavil nekaj pristopov, s katerimi s pomočjo računalniškega vida poskušamo rešiti ta problem.
Preglejmo najprej metode, ki so jih predstavili avtorji v [6]. V članku so se osredotočili na avtentikacijo slikarskih del s pomočjo analize teksture in oblike slikarjevih gibov čopiča. Prvi korak postopka je, da iz slike, ki jo želimo analizirati odstranimo dele, ki jih slikar mogoče ni naslikal. Tukaj imamo v mislih dele, ki so bili restavrirani in pa dele, ki jih je slikar dal naslikati svojim učencem. Predvsem renesenačni mojstri so znani po tem, da so včasih določene "nepomembne"dele slike prepustili svojim vajencem. Restavrirane dele lahko zaznamo s pomočjo visokoresolucijskih fotografij posnetih tako da poudarimo različne spektralne valovne dolžine svetlobe. Prav tako moramo paziti na razpoke, ki nastajajo zaradi sušenja barve, saj jih lahko zamenjamo za potege čopiča. Za zaznavanje in klasifikacijo značilnic na sliki si poglejmo tri pristope opisane v članku [6].

3.1 PS (Penn State) pristop

Pri tem pristopu sliko razdelimo na dele velikosti 512x512 slikovnih pik, kjer zaradi konsistentnosti notranjih predelov deli na robu zavzemajo velikosti med 400 in 600 slikovnih pik. V vsakem predelu izračunamo dve vrsti značilnic. Najprej se osredotočimo na zaznavanje teksture s pomočjo valčkov (ang. wavelets). Teksturo v predelu zaznamo s transformacijo z D4 ortonormiranimi valčki intenzitet slikovnih pik v predelu. To nam vrne intenziteto in grobost sprememb v teksturi na sliki. Od koeficientov transformacije ki jih dobimo, uporabimo tiste pri treh nižjih frekvencah. Dobljene orientacijske koeficiente nato združimo v tridimenzionalni vektor, ki definira predel slike. Drugi tip značilnic, ki jih zaznamo na sliki je geometrična oblika potekov čopiča. Te značilnice so višjenivojske in jih s prostim očesom lažje vidimo. Zaznavanje le-teh je težko, saj se potegi čopičov med seboj mešajo in sekajo. To pomeni, da težko izločimo posamezne potege. Avtorji članka so razvili algoritem za zaznavanje posameznih robov potega čopiča. Ko imamo le-te najdene, na njih izračunamo dolžino, orientacijo in povprečno ukrivljenost. Ko imamo najdene značilnice za posamezno sliko, jih moramo statistično primerjati z bazo del. Teksturne značilnice primerjamo z uporabo 2-D Hidden Markov Modela(HMM), za klasificiranje značilnic potegov čopiča pa se uporabljajo modeli gručenja. Modelov tukaj ne bomo natančneje opisovali, njihovi natančnejši opisi se nahajajo v [6].

3.2 Pr (Princeton Group) pristop

Tudi pri tem pristopu sliko razdelimo na dele velikosti 512x512 slikovnih pik. Uporabljajo se t. i. kompleksni valčki, katerih koeficiente se modelira z HMT modelom (ang. Hidden Markov Tree), ki ga tukaj ne bomo podrobneje opisovali. Pri tem matematičnem modelu ima vsak koeficient valčka t.i. skrito stanje (angl. hidden state), ki lahko zavzame dve vrednosti, rob (ang. edge) in ne-rob(ang. non-edge), glede na to ali valček koeficienta prekriva nek rob na sliki. Poleg tega parametra, imajo koeficienti še dve vrednosti, skalo s



Slika 3.1: Različni tipi uporabljenih valčkov. Tu so prikazani v vseh možnih orientacijah in v dveh zaporednih skalah.

in orientacijo α in so modelirani glede na Gaussovo normalno porazdelitev $N(0, \sigma_{s,\alpha}^{rob/ne-rob})$. Odvisnost med pari koeficientov (skala, orientacija), torej (s,α) in (s-1,α) za vsako lokacijo predstavimo z 2x2 matriko $\epsilon_{s,s-1}^{\alpha}$ tranzicijskih verjetnosti med skritimi stanji. Te tranzicije se pojavijo velikokrat. Če si predstavljamo gladek gradient med dvema trdnima (ang. solid) regijama, lahko vidimo, da ta gradient pri grobih skalah zaznamo kot rob, vendar kot ne-rob pri bolj finih skalah. Za vsak del slike p, imamo 4 parametre modela (2 verjetnosti, 2 varianci) za vsak par (s,α). Vse skupaj dobimo 108 parametrov in iz tega sestavimo vektor značilnic $v^{|p|} \in \mathbb{R}^{108}$. Teh 108 značilnic nato rangiramo glede na to kako učinkovite so za ugotavljanje ali slika pripada določenemu slikarju. Po rangiranju se na prvih mestih praviloma znajdejo verjtnosti prehodov iz ne-rob v rob stanja. Odrežemo značilnice, ki imajo premajhno relevanco in tako dobimo vektor prvih m značilnic $v^{[p|m]} \in \mathbb{R}^m$. Sedaj lahko definiramo m-podobnost med p in p' kot:

$$d_m(p, p') = [\sum_{l=1}^{m} w_l |v_l^{[p|m]} - v_l^{[p'|m]}|^2]^{1/2} \tag{3.1}$$

Kjer je w_l utež, ki daje določenim elementov vektorja večji pomen. Uporabi se multidimenzionalni skalirni algoritem (ang. multidimensional scaling algorithm), da dobimo postavitev točk (ki predstavljajo našo bazo slik) v prostoru. Ta algoritem se nanaša prav na m-podobnost med slikami za raz-

poreditev točk. Točke, ki se nahajajo na robu oblaka, se najbolj razlikujejo od povprečja in zato lahko sklepamo, da ne pripadajo avtorju, čigar slike so večinsko predstavljene v množici.
Ta analiza je primerna za klasificiranje slik, čigar avtorja ne poznamo. Da pa bi zaznali ponaredke, se moramo poslužiti malo drugačne metode. Sliko razdelimo na dele velikost 128x128 in jo analiziramo pri precej bolj finih skalah valčkov. Raziskovalci so ugotovili, da so imeli znani ponaredki Van Gogha precej več koeficientov valčkov z velikimi vrednostimi kot slike, ki so bile potrjene kot izvirnik. Transformacije z valčki uporabljajo manj koeficientov pri grobih kot pri finih skalah. Najbolj fini dve skali sestavljata $15/16 = 93.75\%$ vseh koeficientov. Presežek velikih koeficientov je torej možen le, če se pri dveh najbolj finih skalah nahaja več kot normalno le-teh. To velja, ker so pri ponaredkih potegi čopiča manj tekoči, saj slikar ni ustvarjal v svojem naravnem stilu, ampak je kopiral delo oz. stil drugega slikarja. Izmerimo lahko srednjo vrednost koeficientov pri prvih dveh najbolj finih skalah in slike rangiramo glede na to. Tiste, ki imajo največje odstopanje od sredinske vrednosti, so kandidati za ponaredke.

3.3 Ma (Maastricht Group) pristop

Skupina, ki je sestavila to metodo, je izhajala iz treh glavnih principov:

- obrisi so pomembni;
- slike moramo analizirati pri različnih skalah;
- podobnosti med slikami se kažejo v lokalni teksturi.

Pri tem pristopu se opravi konvolucija slike s pomočjo orientiranih filtrov Gaborjevih valčkov različnih velikosti (ang. multiscale-oriented Gabor wavelet filters), nato pa koeficiente, ki jih dobimo združimo v histograme. Filtri Gaborjevega valčka pridejo v parih, $G_{sod}(x, y, \sigma, \alpha, \omega)$ in $G_{lih}(x, y, \sigma, \alpha, \omega)$.

To sta ravno realni in imaginarni del funkcije

$$e^{2\pi i\omega(x\sin\alpha+y\cos\alpha)}e^{\frac{-x^2+y^2}{2\sigma^2}} \tag{3.2}$$

Kjer sta x in y prostorski koordinati, če predvidevamo, da je filter centriran na izhodišče, α nam določi prostorsko orientacijo, ω pa prostorsko frekvenco. Vsak par filtrov se odzove na spremembe intenzitete pri kombinacijah orientacije in skale. Uporabljamo filtre, s katerimi lahko pokrijemo zadostno število kombinacij orientacije in skale. Izberemo filtre s 6 različnimi orientacijami ($\alpha = (k/6)\pi, k = 0, 1, .., 5$) ter 4 različnimi skalami, pri katerih ω in σ nastavimo na take vrednosti, da se ujemajo z najmanjšimi in največjimi potegi čopiča. Ko pare filtrov konvuliramo s sliko, dobimo kompozicijo slike v t.i. energetske vrednosti. Za vsako kombinacijo slikovne pike, orientacije in skale dobimo eno vrednost. Energijska vrednost dela slike je definirana kot vsota kvadratov vrednosti, ki jih dobimo tako, da obe komponenti konvuliramo s tem predelom slike. Če seštejemo dobljeno energijo vseh predelov na sliki, s tem štejemo število tranzicij svetlo-temno.
Že ta precej enostaven pristop nam lahko zazna ponaredke in pomaga pri določanju avtorja slike. Vendar obstaja boljši pristop, pri katerem uporabljamo multidimenzionalne histograme, ki ujamejo konfiguracijo prostorskih frekvec znotraj dela slike. Za vsak predel velikosti $N * N$ dobimo $(6 * 4)N^2 = 24N^2$ vrednosti, ki jih uredimo v 4x6 zaboje. Enega za vsako kombinacijo skale in orientacije. S temi histogrami definiramo sliko, nato pa želimo slike seveda klasificirati in jim določiti najbolj verjetnega avtorja oz. določiti ali gre za ponaredek. Za to se uporabljajo t.i. SVM (ang. Support Vector Machines), o katerih pa tukaj ne bomo podrobneje pisali, bralec si to lahko podrobneje prebere v članku [6].

Poglavje 4

Zaznavanje značilnic

4.1 Active Shape Models (ASM)

Pregledali bomo metodo za zaznavanje struktur na sliki, ki je bila predstavljena v [1].
Posamezne strukture (objekte) na naši sliki želimo predstaviti z metodo, ki omogoča, da se oblika naše strukture lahko deformira glede na pravila, ki veljajo za ta razred objektov. To nam omogoča, da predstavimo naravno varianco na objektih, ki jih vidimo na slikah. S tem lahko te modele nato uporabimo za namene iskanja podobnih struktur na računalniških slikah. Metoda ASM [1] temelji na tem, da je vsak objekt na sliki predstavljen z množico točk. Te točke lahko predstavljajo tako robove slikovnega objekta kot tudi notranje ali celo zunanje dele. Za vsak objekt najprej ročno označimo točke na objektih v množici vzorčnih slik, ki jih uporabimo za učenje našega modela. Te točke lahko razdelimo v tri različne tipe:

1. tip Točke, ki označujejo pomembne strukturne dele predmeta. Npr. stičišča glavnih žil drevesnega lista.

2. tip Točke, ki označujejo ekstreme modeliranega objekta, kot so najvišja in najnižja točka na predmetu in ekstremi krivulj objekta.

3. tip Točke, ki jih interpoliramo iz točk prejšnih dveh tipov. Kot primer si

lahko predstavjamo točko, ki je na sredini med dvema točkama prejšnih tipov.

Točke prvega tipa so najbolj uporabne za opisovanje strukture našega objekta, vendar skoraj vedno potrebujemo tudi točke drugega in tretjega tipa, da lahko kasneje predstavimo naravno varianco, ki se nahaja v razredu naših objektov. Na vsakem objektu iz učne množice moramo označiti iste točke (točke, ki predstavljajo iste strukture objekta).

Želimo primerjati ekvivalentne točke na objektih učne množice. Tako lahko zgradimo model, ki predstavlja razred objektov, kar nam omogoča, da predmetove naravne deformacije modeliramo.

Da lahko primerjamo ekvivalentne točke med različnimi predmeti iz učne množice moramo te predmete najprej poravnati. Na predmetih lahko izvajamo afine transformacije, torej jih lahko skaliramo, rotiramo in premikamo. Predmeti bodo poravnani, ko bomo minimizirali uteženo vsoto razdalj med ekvivalentnimi točkami.

Naj bo x_i vektor, ki opisuje točke i-tega predmeta v množici:

$$x_i = (x_{i0}, y_{i0}, x_{i1}, y_{i1}, ..., x_{ik}, y_{ik}, ..., x_{in-1}, y_{in-1})^T \tag{4.1}$$

Prav tako naj bo:

$$M(s,\theta) = \begin{bmatrix} x_{jk} \\ y_{jk} \end{bmatrix} = \begin{pmatrix} (s\,cos\theta)x_{jk} - (s\,sin\theta)y_{jk} \\ (s\,sin\theta)x_{jk} + (s\,cos\theta)y_{jk} \end{pmatrix} \tag{4.2}$$

rotacija za θ in skaliranje za faktor s. Če imamo dva vektorja x_i in x_j, potem lahko izberemo take vrednosti θ_j, s_j in translacije (t_{xj}, tyj), da dobimo:

$$min(E_j) = (x_i - M(s_j,\theta_j)[x_j] - t_j)^T W (x_i - M(s_j,\theta_j)[x_j] - t_j) \tag{4.3}$$

W je matrika, ki ima po diagonali uteži, ki dajejo določenim točkam večji pomen. Večjo utež damo točkam, ki so čez celotno množico naših modelov

najbolj stabilne (se glede na druge točke premikajo najmanj), manjšo utež pa damo točkam, ki so na učni množici bolj mobilne.

Da poravnamo množico N objektov uporabimo naslednji algoritem [1, p. 42],

- rotiramo, skaliramo in premaknemo vsak objekt, tako da je optimalno poravnan s prvim objektom v množici.
- **Ponavljaj**
  - izračunaj povprečno lokacijo za vsako posamezno točko vseh do sedaj poravnanih objektov,
  - normaliziraj rotacijo, translacijo, skaliranje in center tega povprečnega objekta,
  - optimalno poravnaj vse objekte na te povprečene točke.
- **Ko proces skonvergira, končaj.**

Data: set of objects N , size (N) = n
for *(i = 2 ; i ¡ n ; i++)* **do**
  Rotate, Translate, Scale N_i so it is optimally aligned with object N_1 (the first object in the set)
end
while *process has not converged* **do**
  Calculate average location for each point of already aligned objects;
  Normalize rotation, translation, scale and center of the average object;
  Optimally align all objects onto the points that have been averaged out;
end

Psevdokoda algoritma za poravnanje N objektov (v angleščini)

Ko objekte v naši učni množici poravnamo, lahko vsakega predstavimo s točko v prostoru dimenzije $2n$. Če imamo N objektov, dobimo torej N točk

v $2n$ dimenzionalnem prostoru. Predpostavimo, da vse te točke ležijo znotraj neke lokacije v prostoru, ki jo imenujemo ang. Allowable Shape Domain in jo bomo v nadaljnje klicali ASD. ASD nam predstavlja oblak točk v tem prostoru $dim(2n)$. Želimo najti center tega oblaka oz. točko v tem prostoru, ki predstavlja povprečen model našega objekta. Če predpostavimo, da je ASD elipsoidne oblike (kar je v veliki večini primerov res) lahko izračunamo center in glavne osi tega elipsoida. Center lahko enostavno izračunamo:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.4}$$

Izračunati moramo še osi. Vsaka os nam pove, kako se točke na predmetu premikajo skupaj, ko se objekt deformira. Najprej moramo za vsako točko v tem prostoru izračunati njeno razliko do $\bar{x}$, torej $dx_i = x_i - \bar{x}$. Iz dobljenih diferenc lahko izračunamo kovariančno matriko S dimenzije $2nx2n$.

$$S = \frac{1}{N}\sum_{i=1}^{n} dx_i dx_i^T \tag{4.5}$$

Osnovne osi našega elipsoida dobimo tako, da poiščemo lastne vektorje elipsoida, torej vektorje v za katere velja $Sv_k = \lambda_k v_k$, kjer so λ_k seveda pripadajoče lastne vrednosti. Večja kot je lastna vrednost, daljša je os elipsoida, ki jo opisuje pripadajoči lastni vektor. Premikanje po tej osi nam omogoča modeliranje največje variance v obliki objekta. Odločimo se, da lahko osi z majhno varianco (tiste, ki pripadajo majhnim lastnim vrednostim) zavržemo in tako elipsoid dimenzije $2n$ aproksimiramo z elipsoidom dimenzije t ($t < 2n$), saj ima original majhno širino na oseh $t+1...2n$. Takoj se pojavi vprašanje, kako izbrati ustrezno vrednost za t. Najprej definirajmo λ_T kot vsoto vseh lastnih vrednosti naše kovariančne matrike

$$\lambda_T = \sum_{k=1}^{2n} \lambda_k \tag{4.6}$$

Sedaj lahko izberemo najmanjši t, za katerega velja, da vsota vseh lastnih vrednosti v teh dimenzijah dovolj blizu vrednosti λ_T. Torej, veljati mora

$$|\sum_{k=1}^{t}(\lambda_k) - \lambda_T| < c \tag{4.7}$$

za nek prej določen c, ki si ga izberemo po želji. Če imamo $2n$ lastnih vektorjev lahko vsako točko znotraj našega ASD dosežemo tako, da vzamemo sredino in ji prištejemo neko linearno kombinacijo lastnih vektorjev, ki delujejo kot baza prostora. Ker pa smo se odločili, da določene lastne vektorje zavržemo, lahko vsako točko x znotraj oblaka aproksimiramo tako:

$$x = \bar{x} + Pb \tag{4.8}$$

kjer je $P = (v_1, v_2, ..., v_t)$ matrika prvih t lastnih vektorjev in
$b = (b_1, b_2, ..., b_t)^T$ vektor uteži. Zgornja enačba nam omogoča, da s spreminjanjem vrednosti uteži generiramo nove oblike, ki ustrezajo razredu našega objekta. Ker večina populacije leži znotraj treh standarnih deviacij od sredine, lahko za b_k izbiramo vrednosti, ki so znotraj teh omejitev, torej:

$$-3\sqrt{\lambda_k} \leq b_k \leq 3\sqrt{\lambda_k} \tag{4.9}$$

Sedaj, ko smo obrazložili metodo za generiranje modelov množic točk, ki se lahko deformirajo glede na naravne lastnosti, hočemo, le-te uporabiti za iskanje objektov na slikah, ki so tudi člani razreda teh modelov. [1, p. 49] Recimo, da imamo podan primer modela:

$$X = M(s, \theta)[x] + X_c \tag{4.10}$$

$$X_c = (X_{c,0}, Y_{c,0}, ..., X_{c,n-1}, Y_{c,n-1})^T \tag{4.11}$$

$M(s, \theta)[]$ je rotacija za θ in skaliranje za faktor s. (X_c, Y_c) je pozicija centra modela v sliki. Želimo imeti iterativno metodo, ki nam bo poi-

skala X, brez da bi potrebovali natančno aproksimacijo začetnega položaja. Začnemo z grobim približkom iskanega objekta na sliki. Vsako točko želimo premakniti vzporedno z normalo proti robu (proporcionalno z močjo normale). S tem dobimo vektor sprememb za vsako točko našega modela, torej $dX = (dX_0, dY_0, dX_1, dY_1, ..., dX_{n-1}, dYn - 1)^T$. Naše točke želimo premakniti čim bližje lokaciji $X + dX$, a tako, da še vedno upoštevamo omejitve našega modela. To storimo tako, da najprej poiščemo translacijo (dX_c, dY_c), rotacijo $d\theta$ ter skalirni faktor $(1 + ds)$, ki kar najbolje preslika X v $X + dX$

Ko najdemo najboljše parametre, da se slika najbolje prilega (po metodi uteženih najmanjših kvadratov), je potrebno objekt deformirati v skladu s pravili [1].

Imamo torej začetno lokacijo točk, ki je podana z enačbo:

$$X = M(s, \theta)[x] + X_c \tag{4.12}$$

Želimo izračunati vektor sprememb dX v lokalnem koordinatnem sistemu modela, tako da je:

$$M(s(1 + ds), (\theta + d\theta))[x + dX] + (X_c + dX_c) = (X + dX) \tag{4.13}$$

$$M(s(1 + ds), (\theta + d\theta))[x + dX] = M(s, \theta)[x] + dX) - (X_c + dX_c) \tag{4.14}$$

(uporabili smo dejstvo, da $X = M(s, \theta)[x]$)

$$M^{-1}(s, \theta)[] = M(s^{-1}, -\theta) \tag{4.15}$$

$$dx = M((s(1 + ds))^{-1}, -(\theta + d\theta))[y] - x \tag{4.16}$$

kjer velja $y = M(s, \theta)[x] + dX - dX_c$

Zgornja enačba nam omogoča, da izračunamo premike točk glede na naš koordinatni sistem. Ti premiki pa niso nujno konsistentni s pravili deformacij našega modela. Da to popravimo, pretvorimo dx v parametrični prostor modela in tako dobimo db, ki predstavlja premike, ki so čim bližje vrednostim dx, a da še vedno zadoščajo našim omejitvam. Spomnimo se ponovno na

enačbo, ki smo jo že uporabili,(4.8), torej $x = \bar{x} + Pb$. Želimo najti tak db, da bo veljalo

$$x + dx \approx \bar{x} + P(b + db) \tag{4.17}$$

Če odštejemo te dve enačbi, dobimo $dx \approx P(db)$, torej $db = P^T dx$

Zgornje enačbe nam omogočajo, da izračunamo spremembe v translaciji, skaliranju, rotaciji in deformaciji, da se oblika modela čim bolj prilega obliki na sliki. Iterativno posodobljamo parametre, dokler proces ne skonvergira (oziroma spremembe postanejo dovolj majhne za naše potrebe).

$$X_c \to X_c + w_t dX_c \tag{4.18}$$

$$Y_c \to Y_c + w_t dY_c \tag{4.19}$$

$$\theta \to \theta + w_\theta d\theta \tag{4.20}$$

$$s \to s(1 + w_s ds) \tag{4.21}$$

$$b \to b + W_b db \tag{4.22}$$

w_t, w_s, inw_θ so skalarne uteži, medtem ko je W_b matrika uteži za vsako točko posebej. Če omejimo vrednosti b_k, omogočimo, da se model deformira samo v oblike, ki so konsistentne z učno množico.

4.2 Lokalni deskriptorji SIFT

S pomočjo ASM na naši sliki zaznamo ključne točke (ang. landmarks), ki pa jih nato opišemo z lokalnim deskriptorjem metode SIFT, čigar teorijo smo povzeli po naslednjemu članku [7]. Okoli vsake zaznane ključne točke moramo najprej določiti globalno orientacijo, tako da poiščemo ekstreme v histogramu orientacije. Vzamemo območje iz katerega naredimo 16x16 matriko, ki vsebuje gradiente, ki pripadajo temu območju. Gradienti, ki se nahajajo v centru in so zato bližje ključni točki, imajo večjo težjo, kot tisti ki se nahajajo na robu našega izbranega območja, zato vrednosti gradientov

utežimo s pomočjo Gaussove funckije. Iz naše matrike velikosti 16x16 lahko naredimo tabelo histogramov velikosti 4x4. Za vsako območje velikosti 4x4 originalne matrike izračunamo gradientne histograme, ki si jih lahko predstavljamo kot vektorje $v = (v_1, v_2, ..., v_n)$, kjer vrednost vsakega elementa v_i predstavlja moč gradienta v določeno smer n. Največkrat je dovolj, da uporabimo 8 smeri za gradiente, zato dobimo vektorje z osmimi elementi, $v = (v_1, v_2, ..., v_8)$. Vrednosti vektorjev iz te 4x4 matrike lahko po vrsti zložimo v en vektor velikost 4x4x8 = 128, torej $v = (v_1, v_2, ..., v_{128})$.
Ker želimo primerjati značilnice preko različnih osvetlitev, najprej ta vektor znormaliziramo. Problem nastane, kadar je osvetlitev nelinearna; to nastane zaradi narave 3D objektov, ki so iz določenih strani osvetljeni drugače. To močno vpliva na vrednosti oz. velikost določenih gradientov, vendar je manjša verjetnost, da to vpliva na dejansko orientacijo gradientov. Ta problem lahko delno rešimo tako, da omejimo posamezne vrednosti v gradientnem vektorju na 0.2 in nato vektor ponovno normaliziramo. S tem dosežemo, da ima orientacija gradientov večji pomen, kot dejanska moč oz. "ostrost"leteh. Vrednost 0.2 je bila dobljena na eksperimentalni podlagi [7, p. 16]

4.3 STASM

Za določanje ključnih točk na obrazih oseb sem uporabil program STASM [8], ki uporablja svojo verzijo modela ASM za lociranje ključnih značilnic in lokalne deskriptorje SIFT za ujemanje / ang. matching teh značilnic. Program deluje na naslednji način. Najprej z globalnim detektorjem zazna obraze na sliki, to je narejeno s pomočjo detektorja ki ga vsebuje programski paket OpenCV. Kot smo že omenili, nam ASM ustvari "povprečni"model iz objektov na naši učni množici. V tem primeru, program prilagodi naš zgeneriran povprečni model človeškega obraza na obraz najden na naši sliki. Nato ponavlja ta dva koraka, dokler algoritem ne skonvergira:

i. predlagaj novo obliko tako, da se trenutne točke modela premaknejo na boljšo lokacijo. To je storjeno tako, da okoli vsake ključne točke

vzorčimo predele slike in točko premaknemo na lokacijo, ki se najbolje ujema z lokalnim deskriptorjem za tisto točko (npr. konica nosu);

ii. predlagani model ni nujno v skladu z globalnimi pravili za strukturo obraza, ki jo je s pomočjo ASM določila naša učna množica. Popravimo lokacije točk, ki se očitno ne skladajo s strukturo človeškega obraza.

Kot lokalni deskriptor se uporablja ang. Histogram Array Transform (HAT), na katerega lahko gledamo kot poenostavljeno različico SIFT. Ker se pred začetkom iskanja lokalnih značilnic rotira celotna slika tako da so oči v horizontalnem položaju (lokacija oči je dobljena s pomočjo ang. OpenCV eye detector), nam ni potrebno zaznavati globalne orientacije, kar je pri SIFT standardno. S tem se zmanjša računuska zahtevnost in čas obdelave. Prav tako se vsak obraz pred iskanjem poskalira tako, da je razdalja oči-usta 100 slikovnih pik. Običajno moramo pri SIFT vsaki slikovni piki iz izbranega območja okoli ključne točke izračunati njegovo mesto v vektorju histogramov. Ker pa je naš obraz že prej poskaliran in zarotiran se lahko preslikava vsakega slikovnih pik iz našega območja (širine 15 slikovnih pik) v tabelo histogramov (velikost 4x5 slikovnik pik) določi samo enkrat za vse deskriptorje. S tem se občutno zmanjša čas iskanja (tudi do 40%). Ker se pri iteriranju našega modela večkrat obiščejo iste koordinate na sliki, lahko HAT deskriptorje shranimo v predpomnilnik in tako zmanjšamo čas iskanja za dodatnih 70%, kar so s poskušanjem ugotovili avtorji v članku [8] Ko se enkrat dobi deskriptor za določeno območje, je potrebno ugotoviti, kako dobro se ujema z obrazno strukturo, ki naj bi jo predstavljal. Za ugotavljanje ujemanja lahko uporabimo več metod.

- Najenostavnejši pristop je ta, da vzamemo evklidsko razdaljo med dobljenim deskriptorjem in "povprečnim"deskriptorjem za to obrazno značilnico iz učne množice. Problem tega pristopa je v tem, da dodeli vsakemu predelku (ang. bin), histograma isto težo, kar pa ni vedno primerno.

- Drug način je s pomočjo Mahalanobisove razdalje, ki dodeli večjo težo

predelkom, ki imajo v učni množici majhno varianco. Težava je v tem, da je računanje Mahalanobijeve razdalje časovno potratno. Prav tako predvideva, da so predelki vektorja histogramov normalno porazdeljeni, kar pa ni popolnoma res.

- Eden od načinov je tudi primerjanje s pomočjo linearne regresije. Uporabimo lahko vzorčne deskriptorje na in okoli značilnic na učni množici in naučimo model, ki izračuna ujemanje glede na elemente deskriptorja na sliki. Linearna regresija predpostavlja, da je vpliv vsakega predelka v histogramu sorazmeren z njegovo vrednostjo in da ni potrebno upoštevati interakcij med različnimi predelki. To pa seveda ni res, saj iščemo vzorce slikovnih pik v neki regiji, ki se velikokrat pojavljajo v gručah. Lahko bi naučili SVM (Support Vector Machines), ki bi uporabljali bolj zapletene različice regresije. To nam da odlične rezultate, vendar je glavna slabost počasnost teh SVM-jev.

- Zaradi počasnosti SVM program STASM uporablja metodo MARS (Multivariate Regression Spline), ki nam daje podobno kvaliteto ujemanja, kot jo omogočajo SVMji, vendar je veliko hitrejša.

Poglavje 5

Primerjanje značilnic

Ko dobimo vektor razmerij med značilnicami za vsak obraz, lahko značilnice med seboj primerjamo. Najprej nas zanima, če obstaja statistično dojemljiva razlika med značilnicami, ki jih najdemo na fotografiji osebe in značilnicami, ki jih najdemo na portretu osebe, ki jo je naslikal slikar. Najprej povejmo nekaj o teoriji preizkušanja statističnih hipotez.

5.1 Preizkušanje statističnih hipotez

Definicijo in nadaljno teorijo sem črpal iz zapiskov predavanj dr. Hladnika [5]

Definicija: Statistična hipoteza je vsaka domneva o porazdelitvi slučajne spremenljivke X na populaciji.

Za statistično hipotezo velja, da je lahko pravilna ali pa nepravilna. Vedno testiramo eno statistično hipotezo proti drugi, npr. $H_0 : H_1$. Pri preizkušanju statističnih hipotez pregledujemo ali so razlike značilne oz. gledamo, da vzorec ne odstopa preveč od naše hipoteze. Pomembno si je zapomniti, da neke statistične hipoteze ne moremo potrditi, ampak jo lahko samo zavrnemo. Največ kar lahko naredimo je, da rečemo, da glede na predstavljene podatke ni zadostnega razloga za zavrnitev statistične hipoteze. Zanima nas, kakšen

je kriterij za zavrnitev, zato najprej izberemo kritično območje K, ki je podmnožica v prostoru vrednosti naše statistike. Naš K želimo izbrati tako, da bo verjetnost, da bi cenilka C padla v kritično območje zelo majhna. (S cenilkami na podlagi vzorca ocenjujemo parametre porazdelitve naše slučajne spremenljivke).

$$\alpha = P(C \in K|H_0) \tag{5.1}$$

Ponavadi za vrednosti α izberemo vrednosti 0.05 ali 0.01.

5.1.1 Testi značilnosti

Najprej izberemo cenilko oz. njeno funkcijo U, ki ji pravimo testna statistika. S tem določimo naš test. Da to lahko naredimo, moramo poznati vsaj njeno pogojno porazdelitev pri dani predpostavki H_0. S tem lahko določimo rob kritičnega območja K_α, za nek že prej izbrani α.

$$\alpha = P(U \in K_\alpha|H_0) \tag{5.2}$$

Ko imamo določen rob K_α, lahko izračunamo vrednost naše testne statistike U na danem vzorcu. Gledamo kam pade vrednost, če se nahaja znotraj kritičnega območja K_α hipotezo zavrnemo s stopnjo zaupanja α, v nasprotnem primeru pa o naši hipotezi ne moremo reči nič. Izbrati moramo drugo testno statistiko ali pa dobiti večji vzorec.

Za naše potrebe, je pomemben test $H_0(\mu = \mu_0)$, kjer je σ neznan. Naša predpostavka je, da ima spremenljivka porazdelitev $N(\mu, \sigma)$ in da je standardna deviacija σ za nas nepoznana. Izbrati moramo testno statistiko

$$T = \frac{\bar{X} - \mu_0}{s}\sqrt{n} \tag{5.3}$$

$$s = \sqrt{\frac{(X_1 - \bar{X})^2 + ... + (X_n - \bar{X})^2}{n-1}} \tag{5.4}$$

kjer je n velikost našega vzorca, $X_1, ..., X_n$ pa njegove vrednosti. Če velja

naša hipoteza je ta testna statistika porazdeljena po Studentovem zakonu $S(n-1)$ [5].

V našem primeru imamo dva vzorca obraznih značilnic. Prvi vzorec je vektor obraznih razmerij na fotografiji osebe, drugi vzorec pa vektor obraznih razmerij na slikarjevi upodobitvi iste osebe. Zanima nas, ali slikarjeva upodobitev statistično značilno spremeni obrazna razmerja. Ker vsak obraz definiramo z n razmerji imamo $X = (x_1, x_2, ..., x_n)$ za razmerja fotografije in $Y = (y_1, y_2, ..., y_n)$ za razmerja slikarjeva upodobitve. V prvem koraku izračunamo $X - Y = (x_1 - y_1, ..., x_n - y_n)$ nato pa zapišemo našo hipotezo:

$$H_0(\text{slikar ne spreminja razmerij}) : H_1(\text{slikar spreminja razmerja})$$
$$\mu_1 = \mu_2 : \mu_1 \neq \mu_2$$

Za nadaljni postopek moramo najprej izračunati $\overline{X-Y}$, nato pa lahko dobimo vrednost $s = \sqrt{\frac{(X_1-\overline{X-Y})^2+...+(X_n-\overline{X-Y})^2}{n-1}}$. Za statistiko t in 7 prostorskih stopenj velja

$$t_{1-\alpha}(n-1) = t_{0.95}(7) = 1.895 \tag{5.5}$$

$$T = \frac{\overline{X-Y} - \mu_0}{s}\sqrt{n} = \frac{\overline{X-Y}}{s}\sqrt{8} \tag{5.6}$$

Ker testiramo hipotezo $\mu_1 = \mu_2$ in je $\mu_0 = \mu_1 - \mu_2$ iz tega sledi $\mu_0 = 0$. Sledi $K_\alpha = (-\infty, -1.895) \cup (1.895, \infty)$. Če vrednost testne statistike T pade na območje intervala zaupanja K_α, potem hipotezo zavrnemo, v nasprotnem primeru pa o naši hipotezi ne moremo povedati nič in moramo imeti več vzorcev ali izbrati drugo testno statistiko.

Pomembno vprašanje je, katera razmerja med ključnimi točkami na našem obrazu moramo izbrati, da bo metoda primerjave delovala čim bolj pravilno. Če imamo n ključnih točk na obrazu, lahko med njimi izračunamo n^2 razdalj. Najprej smo poizkusili izračunati vsa možna razmerja na obrazu. Ker zaznamo 20 ključnih točk, dobimo torej kombinatorično za vsak obraz $(20 * 19) * (20 * 19) = 144000$ razmerij. Kmalu ugotovimo, da je za to metodo

število značilnic preveliko. Statistično hipotezo smo preizkusili na dveh zaporedno posnetih fotografijah osebe. To smo ponovili na večih osebah ($n = 5$) in dobili rezultate, ki ne ustrezajo. Npr. pri primerjavi dveh zaporednih fotografij, nam je T-test vrnil število močno znotraj intervala, kar naj bi pomenilo da so se obrazna razmerja spremenila, kar pa seveda ne more biti res. Precej boljše rezultate smo dobili, če smo vsak obraz definirali samo z nekaj izbranimi razmerji med ključnimi točkami na obrazu.

Za naša obrazna razmerja smo izbrali:

i. razdalja med zenicami proti razdalji med temeni,

ii. razdalja med temeni proti razdalji od centra med očmi do konice brade,

iii. razdalja med zenicami proti širini ustnic,

iv. razdalja med zunanjimi deli očmi proti širini ustnic,

v. razdalja med zunanjimi deli obrvi proti razdalji med temeni,

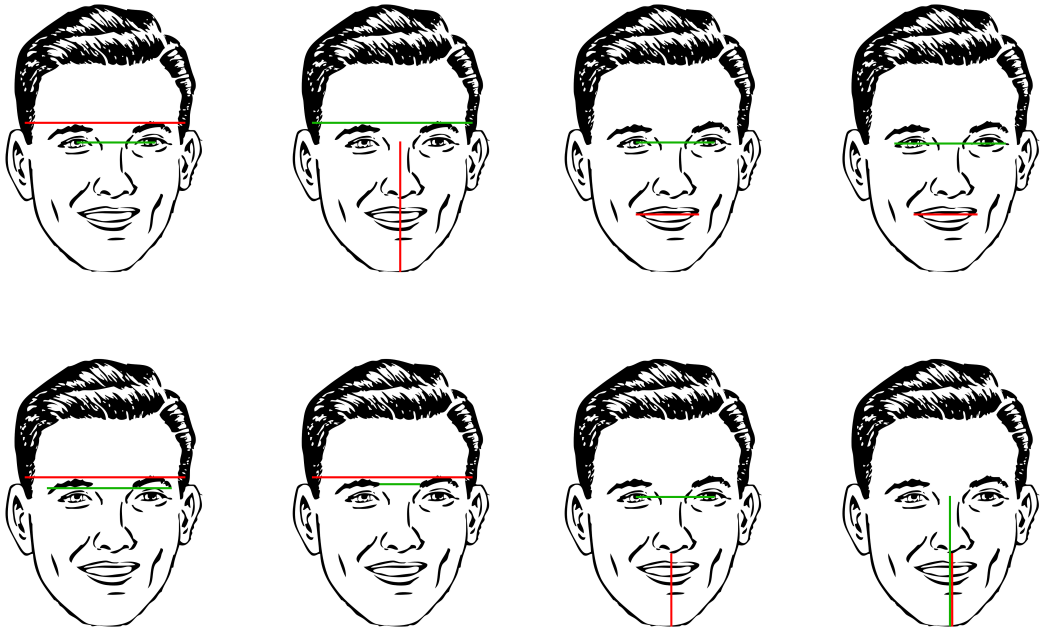
vi. razdalja med notranjimi deli obrvi proti razdalji med temeni,

vii. razdalja med zenicami proti razdalji od konice nosu do konice brade,

viii. razdalja od centra med očmi do konice brade proti razdalji od konice nosu do konice brade.

5.2 Testiranje na primerih

Obrazne značilnice in posledično obrazna razmerja smo iskali na primeru dveh slikarjev, enega slovenskega in enega tujega. Zbrali smo dela ameriškega realista Thomasa Eakinsa (1844-1916). Njegov realistični umetniški stil nam omogoča, da metode računalniškega vida uporabimo tudi na slikah, kar bi bilo pri bolj abstraktnem slogu precej težje. Umetnik se je veliko ukvarjal tudi s fotografijo, zato so nam na voljo fotografije portretirancev, kar nam omogoča primerjavo med isto osebo na fotografiji in na slikarskem platnu. Na



Slika 5.1: Obrazna razmerja

bazi fotografij oseb in "poskeniranih"portretov smo izvedli postopek iskanja obraznih značilnic. Rezultati so bili mešani, zaradi slabe kvalitete nekaterih fotografij, je naš postopek deloval nezanesljivo. Predvsem v primerih, ko oseba z obrazom ni bila obrnjena "direktno"proti fotoaparatu. Na umetniških delih smo dobili malo boljše rezultate. Tukaj smo naleteli na težave, predvsem v primerih, ko se je slikar odmaknil od popolnoma realističnega sloga. Na koncu smo značilnice dobili na desetih parih fotografij in slikarjevih del. Med temi značilnicami smo izvedli že prej omenjeni statistični test značilnosti in dobili naslednje rezultate.

Že prej smo izračunali kritično območje za našo testno značilnico, ki smo jo določilli kot $K_\alpha = (-\infty, -1.895) \cup (1.895, \infty)$. Hipotezo smo preizkušali na naših testnih primerih in iz zgornje tabele lahko vidimo, da je v devetih

Fotografija	*Slika*	Test značilnosti
Thomas Eakins (1890)	Thomas Eakins (1902)	-1.1377
Thomas Eakins (1904)	Thomas Eakins (1902)	-1.60534
Wiiliam H. Elder (1883)	William H. Elder (1903)	-0.51732
William H. Macdowell (1880)	William H. Macdowell (1904)	-0.54338
Samuel Murray (1889)	Samuel Murray (1891)	1.49267
Henry A. Rowland (1897)	Henry A. Rowland (1902)	1.38518
Susan Eakins (?)	Susan Eakins (1884)	1.38715
Susan Eakins (?)	Susan Eakins (1899)	-0.89661
Henry O. Tanner (1907)	Henry O. Tanner (1897)	-1.45077
Horatio C. Wood (1913)	Horatio C. Wood (?)	-2.08794

Tabela 5.1: Tabela primerjanih del z rezultati testa

od desetih primerov hipoteza zavrnjena, saj razlike niso statistično značilne. Iz tega lahko sklepamo, da slikar v svojem postopku ne spreminja obraznih razmerij portretirane osebe in človeški obraz preslika realno. To ima v primeru realističnega slikarja smisel, saj slikarjev slog teži k čim boljši realni reprezentaciji. Žal nam to v tem primeru onemogoča uporabo te metode za npr. avtentikacijo slikarjevih del glede na preslikavo fotografija-slika.

Slika 5.2: Primeri napak v zaznavanju značilnic

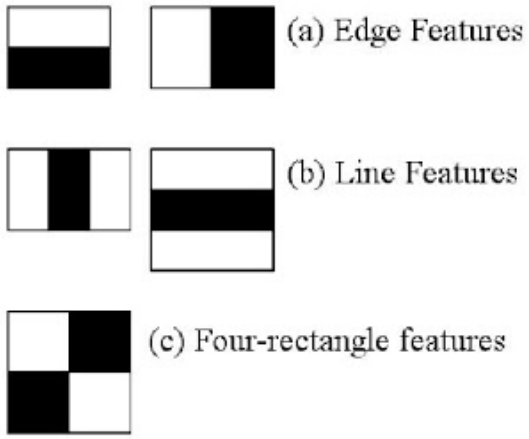
Našo metodo smo želeli preizkusiti še na delih slovenskega slikarja **Rudija Španzla (1948-)**. Njegova dela se zgledujejo po prinicipih renesančnih

umetnikov in poudarjajo realizem in simboliko. Iz njegove knjige [10] smo izbrali slike, kjer so obrazi oseb dobro vidni. Žal pa nam ni uspelo pridobiti fotografij oseb, ki bile vir slik v knjigi, zato smo ta test opustili.

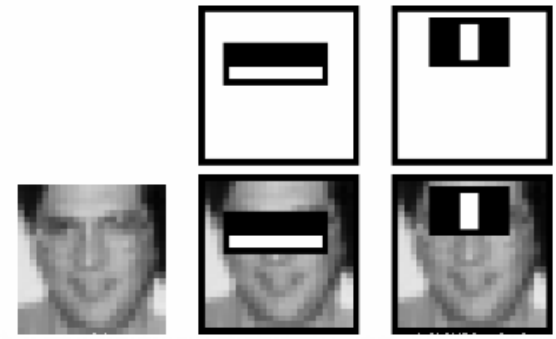
5.3 OpenCV algoritmi za zaznavanje obrazov

OpenCV za zaznavanje obrazov uporablja metodo Haarovih kaskad (ang. Haar Cascades). To je metoda strojnega učenja, zato na začetku potrebuje množico pozitivnih slik (slik obrazov) in negativnih slik (slik brez obrazov) da z njo naučimo klasifikator. Z uporabo Haarovih kaskad lahko iz teh slik dobimo njihove značilnice. Haarove kaskade delujejo kot jedro konvolucije. Vsaka značilnica slike je vrednost, ki jo dobimo, če odštejemo vsoto pikslov pod belim pravokotnikom od vsote pikslov pod črnim pravokotnikom (Glej slika: 5.4). To izračunamo za vsako možno kombinacijo velikosti in lokacije jedra.
Med značilnicami, ki so bile izračunane je večino neuporabnih in nam škodijo pri natančnosti, zato moramo določene značilnice odstraniti. Za določanje primernih značilnic se uporablja metoda Adaboost. Vse značilnice združimo s vsemi slikami testne množice. Za vsako značilnico najdemo najboljšo mejo, ki bo klasificirala sliko pozitivno ali negativno in nato izberemo tiste značilnice, ki imajo najmanjšo mero napačnega klasificiranja. Končni klasifikator je utežena vsota teh klasifikatorjev.
Ta metoda deluje zelo dobro za obraze, ki so enakomerno osvetljeni in gledajo naravnost v sliko. Pri naših poizkusih pa smo ugotovili, da ima metoda pomankljivosti. Učna množica v OpenCV uporablja obraze, ki gledajo naravnost v sliko, zato detekcija obrazov na slikah kjer subjektov obraz opazujemo pod kotom ne deluje zanesljivo. Predvsem je težavno, če je oseba postavljena tako, da je vidno samo eno oko. Težava nastane tudi, če sta vidna oba očesa vendar se eden v senci oz. slabše osvetljenem delu slike. Na naših primerih smo opazili tudi težave pri zaznavanju obrazov na slikah slabše kvalitete (npr. analogne slike iz začetka 20. stoletja). Prav tako je prišlo do težav

pri detekciji obrazov na umetniških slikah, kjer je zaradi same teksture dela, prišlo do napačne klasifikacije obrazov.



Slika 5.3: Haarove značilnice



Slika 5.4: Konvolucija Haarovih značilnic s sliko

5.4 Pomanjkanje baz umetniška slika - fotografija

Ko smo želeli primerjati obrazne poteze osebe na fotografiji z obraznimi potezami na umetniškem delu - sliki, smo se soočili s problemom pomanjkanja

primerne baze fotografij - umetniških del, ki bi jo lahko uporabili za naše delo. Ta problem nastaja zaradi dveh glavnih faktorjev.
Veliko znanih slikarjev in njihovih del je nastalo v času pred obstojom fotografije, zato fotografije teh oseb ne obstajajo. Če gledamo dela, ki so sicer nastala po izumu fotografije, a so vseeno starejša (npr. dela iz 19. stoletja), je velika verjetnost, da so se s časom fotografije naslikanih oseb porazgubile in jih je nemogoče ali pa zelo težko pridobiti. Težavno je predvsem, kadar portretirana oseba ni slavna in zato njihove fotografije niso javno dostopne.
Drug problem je ta, da velika večina portretnih slikarjev, svojih portretirancev ne fotografira ampak osebo raje slikajo živo v prostoru. To pomeni, da tudi sam slikar nima dostopa do fotografij svojih portretirancev.

Poglavje 6

Spremenjen pristop do problema

Kot lahko vidimo v prejšnjem poglavju rezultati niso potrdili naše hipoteze, ki je trdila, da lahko slikarjev stil definiramo prek opazovanja preslikanih obraznih potez. Še vedno pa smo želeli dobiti metodo ki bi nam omogočila, da lahko sliko pripišemo določenemu slikarju. Zato smo se odločili, da se bomo problema lotili drugače. Vsako sliko želimo opisati z določenimi značilnicami, ki to sliko definirajo. Te značilnice so:

- **Povprečne barve na sliki** Na vsaki sliki želimo najti njene povprečne barve. Najprej s pomočjo algoritma K-means na sliki poiščemo štiri značilne barve v standardnem RGB prostoru, vendar to ni dovolj. Določene barve lahko zavzemajo majhen del slike, pa vendar na sliki močno izstopajo, saj so zelo žive. Zato celo sliko pretvorimo v HSV prostor, če odrežemo tiste barve, ki imajo majhne vrednosti barvne intenzivnosti (ang. saturation) in barve svetlosti (ang. value), nam ostanejo samo še žive barve. Če na tej množici ponovno poženemo K-means algoritem, nam le-ta vrne izstopajoče barve na sliki.

- **Standardna deviacija na barvnih histogramih** Sliko pretvorimo v LUV barvni prostor in izračunamo barvne histograme na sliki. Za

vsako barvno kompnento izračunamo histogram in njegovo standardno deviacijo, katero vzamemo za eno od značilnic.

- **Obseg slike** Slikarji se stilsko razlikujejo po velikosti slik, ki jih ustvarjajo. Iz tega razloga za eno od značilnic vzamemo obseg dane slike.

- **Razmerje stranic** Poleg velikosti se slike razlikujejo tudi v razmerju velikosti stranic. Zato za eno od značilnic vzamemo razmerje med širino in višino slike.

- **Tekstura slike** Različni slikarji proizvajajo slike z različnimi strukturami. Eden od načinov s katerim lahko definiramo teksturo slike je algoritem **ARes** [9], ki na vsaki sliki poišče skale, ki imajo največje število lokalnih ekstremomv. Začnemo pri polni resoluciji oz. skale in se sprehodimo skozi vse slikovne pike na sliki. Za vsakega izmed njih pogledamo ali je lokalni ekstrem v oknu 3x3 sosednjih slikovnih pik in preštejemo tiste, za katere to velja. Ko pregledamo vse slikovne pike, sliko pomanjšamo in postopek ponavljamo dokler ne pregledamo vseh skal oz. resolucij. Prvih n resolucij z največ lokalnimi ekstremi vzamemo kot značilnice za našo sliko.

Naš algoritem smo testirali na treh slikarjih. Izbrali smo slikarje različnih slogov, Edgar Degas, Vincent Van Gogh in Jackson Pollock. Od vsakega slikarja smo vzeli množico desetih slik, ki predstavljajo slikarjev stil. Za vsako od teh slik smo izračunali vektorje značilnic, ki smo jih najprej normalizirali in nato združili v matriko. Za boljše rezultate smo matriko transformirali s pomočjo postopka PCA (ang. Principal Component Analysis), ki nam zmanjša dimenzijo matrike in ponajde korelacijo med značilnicami. Nad vektorji značilnic, ki smo jih dobili preko PCA postopka, izvedemo K-means gručenje. Zanima nas, če bo algoritem značilnice pogručil pravilno, torej glede na slikarje, ki jim pripadajo.

Vse skupaj smo testirali 30 slik. Od tega jih je bilo pravilno klasificiranih 22 in nepravilno 8. Ta podatek nam pove, da je naša metoda v tem primeru pravilno klasificirala 73% vseh del. Problematika našega pristopa je predvsem

Slikar (ime dela)	St. gruče	Pravilno ali Nepravilno
Edgar Degas (The Rehearsal)	1	Pravilno
Edgar Degas (Dancer in Pink)	1	Pravilno
Edgar Degas (Interior)	1	Pravilno
Edgar Degas (Beach Scene)	1	Pravilno
Edgar Degas (Young Woman With Ibis)	1	Pravilno
Edgar Degas (Place de la Concorde)	1	Pravilno
Edgar Degas (Dancers at the Bar)	2	**Nepravilno**
Edgar Degas (The Rehearsal of the Ballet Onstage)	1	Pravilno
Edgar Degas (Yellow dancers in the wings)	2	**Nepravilno**
Edgar Degas (Young spartans Exercising)	1	Pravilno
Vincent Van Gogh (Starry Night Over the Rhone)	2	Pravilno
Vincent Van Gogh (Starry Night)	2	Pravilno
Vincent Van Gogh (Bedroom in Arles)	2	Pravilno
Vincent Van Gogh (Cafe Terrace at Hight)	1	**Nepravilno**
Vincent Van Gogh (Wheat Field with Cypresses)	2	Pravilno
Vincent Van Gogh Irises	2	Pravilno
Vincent Van Gogh (Red Vineyards)	2	Pravilno
Vincent Van Gogh (Portrait of Eugene Boch)	1	**Nepravilno**
Vincent Van Gogh (The Yellow House)	2	Pravilno
Vincent Van Gogh (The Night Cafe)	1	**Nepravilno**
Jackson Pollock (The Key)	3	Pravilno
Jackson Pollock (Blue Poles Nr. 11)	3	Pravilno
Jackson Pollock (Convergence)	3	Pravino
Jackson Pollock (Nr. 1)	3	Pravilno
Jackson Pollock (One: Nr. 31)	3	Pravilno
Jackson Pollock (Nr. 1)	2	**Nepravilno**
Jackson Pollock (Reflection of the Big Dipper)	2	**Nepravilno**
Jackson Pollock (Shimmering SUbstance)	2	**Nepravilno**
Jackson Pollock (Stenographic Figure)	3	Pravilno
Jackson Pollock (There were 7 in 8)	3	Pravilno

Tabela 6.1: Tabela pravilnosti klasifikacije slik

v tem, da značilnice, ki smo jih vzeli na sliki upoštevajo le barve, teksturo ter velikost slike. Slikarska dela pa se razlikujejo tudi konceptualno, to pa je s tem pristopom težko definirati. Izboljšane rezultate bi verjetno dobili, če bi izbrali bolj kompletne in kompleksnejše metode za definiranje teksture, kot recimo tiste opisane v enem od prejšnjih poglavij. Prav tako bi točnost verjetno narastla s številom slik v testni množici.

6.1 Časovna zahtevnost določenih algoritmov

Algoritem	Časovna zahtevnost	Pomen spremenljivk
K-Means	O(lkmn)	m = število slikovnih točk n = število dimenzij vsake točke k = število gruč l = število iteracij
PCA (Z Jakobijevo diag.)	$O(p^2n + p^3)$	n = število vektorjev p = dimenzija vektorjev
ARes	$O(Kn)$	n = število slik, K = konstanta, začetna resolucija slike. Ker se moramo pri vsaki skali vsake slike sprehoditi čez vse slikovne pike, se algoritem izvaja kar nekaj časa.
STASM	$O(n)$	n = število ključnih točk na obrazu ki jih želimo zaznati

Tabela 6.2: Časovna zahtevnost algoritmov

Poglavje 7

Zaključek

V diplomski nalogi smo testirali dva pristopa za identificiranje slikarjev glede na njihova dela. Pri našem prvotnem pristopu nas je zanimalo ,ali lahko definaramo slikarjev stil glede na njegovo preslikavo obraznih potez. Metodo smo testirali na ameriškem realističnem slikarju, Thomasu Eakinsu.
Imeli smo bazo fotografiranih oseb, ki jih je slikar portretiral. Na fotografijah in portretih smo s programom STASM zaznali ključne točke na obrazih oseb. Tukaj smo naleteli na prvo težavo. Fotografije oseb so bile stare in zato slabše tehnične kvalitete. V primerih, ko je bila fotografija neostra ali pa je bil obraz slabo osvetljen, je pri zaznavanju točk prišlo do napak. Tako na slikah, kot na fotografijah je prišlo do težav pri zaznavanju ključnih točk na nefrontalnih obrazih. Odločili smo se, da iz množice fotografij in slik izločimo neprimerne in tako nam je ostala množica desetih parov fotografija-slika, na katerih nam je uspelo pravilno zaznati ključne točke na obrazu osebe. Iz teh točk smo izračunali obrazna razmerja na obeh in s statističnim testom poskušali ugotoviti ali obstajajo statistično pomembne razlike med razmerji na fotografiji in na naslikanem portretu osebe. V našem primeru nismo ugotovili razlik. Mogoče bi lahko za nadaljnje raziskovanje vzeli preslikave pri slikarjih katerih slog ni realističen, vendar to vodi do dodatnih problemov z zaznavanjem ključnih obraznih točk, zato smo se odločili da bomo preizkusili drugačen pristop do problema.

Pri drugem pristopu vsako sliko predstavlja vektor značilnic. Za elemente vektorja smo vzeli povprečne barve, izstopajoče barve, dimenzije slike, obseg slike, standardno deviacijo na barvnih histogramih in teksturo slike, pridobljeno z algoritmom ARes. Izbrali smo tri znane slikarje z različnimi stili slikanja in testirali, če lahko slike pravilno pogručimo glede na njihove vektorje značilnic. Uspelo nam je dobiti 73% točnost pri gručenju. Rezultat bi verjetno lahko izboljšali z uporabo naprednejših metod za detekcijo barv in tekstur na slikah.

Dodatek A

Programska koda

Ares.py

```
from os import listdir
from os.path import isfile, join
from PIL import Image
import numpy as np
import cv2
import os


def min(a,b):
        if a <= b:
                return a
        else:
                return b

def setResizeStep(radius,maxHeight,maxWidth):
        if maxHeight <= maxWidth:
                return float(((2*float(RADIUS))/3)/float(maxHeight))
        else:
                return float(((2*float(RADIUS))/3)/float(maxWidth))


def listFiles(mypath):
        onlyfiles = [ f for f in listdir(mypath) if isfile(join(mypath,f)) ]
        return onlyfiles

def getMaxDimensions(picDir,files):
        print files
        maxH = 0;
        maxW = 0;
        for picture in files:
                img = cv2.imread(picDir + picture,0)
                height, width = img.shape
                if (height > maxH) :
                        maxH = height
                if (width > maxW) :
                        maxW = width


        return maxH, maxW
```

```
def neighbours(img,i,j):
        return np.delete(img[i-1:i+2,j-1:j+2].flatten(),4)

def isPeak(hood,px):
        maxEl = np.amax(hood)
        minEl = np.amin(hood)

        if (px >= maxEl):
                return True
        elif (px <= minEl):
                return True
        else:
                return False


def preparePics(picDir,greyscaleDir):
        print greyscaleDir
        listedFiles = listFiles(picDir)
        maxHeight, maxWidth = getMaxDimensions(picDir,listedFiles)
        for slika in listedFiles:
                img = cv2.imread(picDir + slika,0)
                height, width = img.shape
                mean = np.average(img)
                newimage = np.zeros((maxHeight,maxWidth))
                newimage[0:height,0:width] = img
                newimage[height:maxHeight,:] = mean
                newimage[:,width:maxWidth] = mean
                cv2.imwrite(greyscaleDir + slika ,newimage)

def getPeaksForPicture(picDir,greyscaleDir,RADIUS):
        preparePics(picDir,greyscaleDir)
        fajli = listFiles(picDir)
        resizeStep = 0.01
        peaksResolutions = []
        preparedPics = listFiles(greyscaleDir)
        maxHeight, maxWidth = getMaxDimensions(greyscaleDir,preparedPics)
        print preparedPics
        for slika in preparedPics:
                f = open("peaks/" + slika.rstrip('.jpg') + '.txt', 'w')
                img = cv2.imread(greyscaleDir + slika,0)
                v = float(1)
                while (min(maxWidth,maxHeight)*v > 3*RADIUS):
                        peakCounter = 0
                        resizedimg = cv2.resize(img,None, fx=v, fy=v)
                        imgHeight, imgWidth = resizedimg.shape
                        for i in range(1,imgHeight-1):
                                for j in range(1,imgWidth-1):
                                        px = resizedimg[i,j]
                                        hood  = neighbours(resizedimg,i,j)
                                        if isPeak(hood,px):
                                                peakCounter+=1


                        resolucija = imgHeight * imgWidth
                        if float(peakCounter) != 0:
                                peaksResolutions.append(
                                [v, round(float(resolucija)/float(peakCounter),2)]
                                )
                                s = str(slika) + ";" + str(v) + ";" +
                                str(round(float(resolucija)/float(peakCounter),2)) + "\n"
                        else:
                                peaksResolutions.append([v,0])
                                s = str(slika) + ";" + str(v) + ";" + "0.00" + "\n"
```

```
                        f.write(s)
                        resolutionsDir="slike/resolutions/"+slika.rstrip('.jpg')+"/"
                        if not os.path.exists(resolutionsDir):
                                os.makedirs(resolutionsDir)
                        cv2.imwrite(resolutionsDir + str(v) +  slika ,resizedimg)
                        v = v-resizeStep

        f.close()
```

DetekcijaBarv.py

```
from collections import namedtuple
from PIL import Image, ImageDraw
from math import sqrt, pow
import random
import numpy as np
import cv2
import math
from skimage import io, color
from iptcinfo import IPTCInfo
import sys
from os import listdir
from os.path import isfile,join
import mdp
import readFromFile
import ares
import colorsys
from sklearn.preprocessing import normalize


try:
    import Image
except ImportError:
    from PIL import Image

Point = namedtuple('Point', ('coords', 'n', 'ct'))
Cluster = namedtuple('Cluster', ('points', 'center', 'n'))

def rgb2hsv(r, g, b):
    r, g, b = r/255.0, g/255.0, b/255.0
    mx = max(r, g, b)
    mn = min(r, g, b)
    df = mx-mn
    if mx == mn:
        h = 0
    elif mx == r:
        h = (60 * ((g-b)/df) + 360) % 360
    elif mx == g:
        h = (60 * ((b-r)/df) + 120) % 360
    elif mx == b:
        h = (60 * ((r-g)/df) + 240) % 360
    if mx == 0:
        s = 0
    else:
        s = df/mx
    v = mx
    return round(h,2),(round(s,2)), (round(v,2))

def hsv2rgb(h, s, v):
    h = float(h)
    s = float(s)
    v = float(v)
    h60 = h / 60.0
    h60f = math.floor(h60)
```

```
    hi = int(h60f) % 6
    f = h60 - h60f
    p = v * (1 - s)
    q = v * (1 - f * s)
    t = v * (1 - (1 - f) * s)
    r, g, b = 0, 0, 0
    if hi == 0: r, g, b = v, t, p
    elif hi == 1: r, g, b = q, v, p
    elif hi == 2: r, g, b = p, v, t
    elif hi == 3: r, g, b = p, q, v
    elif hi == 4: r, g, b = t, p, v
    elif hi == 5: r, g, b = v, p, q
    r, g, b = int(r * 255), int(g * 255), int(b * 255)
    return r, g, b


def hex_to_rgb(value):
    value = value.lstrip('#')
    lv = len(value)
    return tuple(int(value[i:i + lv // 3], 16) for i in range(0, lv, lv // 3))

def rgb_to_hex(rgb):
    return '#%02x%02x%02x' % rgb


def get_points(img):
    points = []
    w, h = img.size
    for count, color in img.getcolors(w * h):
        points.append(Point(color, 3, count))
    return points

def get_points_hsv(img):
    points = []
    w, h = img.size
    for count, color in img.getcolors(w*h):
        barva = rgb2hsv(color[0],color[1],color[2])
        if barva[1] > 0.5 and barva[2] > 0.5 :
            points.append(Point(barva,3, count))

    return points

rtoh = lambda rgb: '#%s' % ''.join(('%02x' % p for p in rgb))

def colorz(filename, n=3):
    img = Image.open(filename)
    img.thumbnail((200, 200))
    w, h = img.size

    points = get_points(img)
    clusters = kmeans(points, n, 1)
    rgbs = [map(int, c.center.coords) for c in clusters]
    return map(rtoh, rgbs)

def colorz_hsv(filename, n=3):
    img = Image.open(filename)
    img.thumbnail((200, 200))
    w, h = img.size
    points = get_points_hsv(img)
    clusters = kmeans(points, n, 1)
    rgbs = [map(float, c.center.coords) for c in clusters]
    return rgbs
    #return map(rtoh, rgbs)
```

```
def euclidean(p1, p2):
    return sqrt(sum([
        (p1.coords[i] - p2.coords[i]) ** 2 for i in range(p1.n)
    ]))

def calculate_center(points, n):
    vals = [0.0 for i in range(n)]
    plen = 0
    for p in points:
        plen += p.ct
        for i in range(n):
            vals[i] += (p.coords[i] * p.ct)
    return Point([(v / plen) for v in vals], n, 1)

def kmeans(points, k, min_diff):
    clusters = [Cluster([p], p, p.n) for p in random.sample(points, k)]

    while 1:
        plists = [[] for i in range(k)]

        for p in points:
            smallest_distance = float('Inf')
            for i in range(k):
                distance = euclidean(p, clusters[i].center)
                if distance < smallest_distance:
                    smallest_distance = distance
                    idx = i
            plists[idx].append(p)

        diff = 0
        for i in range(k):
            old = clusters[i]
            center = calculate_center(plists[i], old.n)
            new = Cluster(plists[i], center, old.n)
            clusters[i] = new
            diff = max(diff, euclidean(old.center, new.center))

        if diff < min_diff:
            break

    return clusters

def listFiles(dir):
    onlyfiles = [ f for f in listdir(dir) if isfile(join(dir,f)) ]
    return onlyfiles


def histogram(image,channel):
    image = cv2.resize(image, (400, 400))
    luvImg = color.rgb2luv(image)
    hist,bins = np.histogram(luvImg[channel].ravel(),256,[0,256])
    return hist.astype(int)

def stdDevHistograms(image):
    hist_L = histogram(image,0)
    hist_U = histogram(image,1)
    hist_V = histogram(image,2)
    return [round(np.std(hist_L),3), round(np.std(hist_U),3), round(np.std(hist_V),3)]

def barve_hex2rgb(barve):
    rgbBarve = []
    for barva in barve:
        rgbBarve.append(hex_to_rgb(barva))
    return rgbBarve
```

```
def barve_hsv2rgb(barve):
    rgbBarve = []
    for barva in barve:
        rgbBarve.append(hsv2rgb(barva[0],barva[1],barva[2]))
    return rgbBarve

def drawColors(barve,barve_hsv):
    im = Image.new('RGB', (512,512), (255,0,0))
    dr = ImageDraw.Draw(im)
    for i in range(0,len(barve)):
        dr.rectangle(((i*512/len(barve),0),((i+1)*512/len(barve),512)),
        fill="rgb" +
        str(hex_to_rgb(barve[i])))
    im.save("fotka.png")


    im_hsv = Image.new('RGB', (512,512), (255,0,0))
    dr_hsv = ImageDraw.Draw(im_hsv)
    for i in range(0,len(barve_hsv)):
        dr_hsv.rectangle(((i*512/len(barve_hsv),0),((i+1)*512/len(barve_hsv),512)),
        fill="rgb" +
        str(hsv2rgb(barve_hsv[i][0],barve_hsv[i][1],barve_hsv[i][2])))

    im_hsv.save("fotka_hsv.png")
    img = cv2.imread('fotka.png',3)
    img_hsv = cv2.imread('fotka_hsv.png',3)
    cv2.imshow('image',img)
    cv2.imshow('image_hsv',img_hsv)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def calculateObseg(keywords):
    return int(keywords[0]) + int(keywords[1])

def writeZnacilniceToFile(filename, znacilnice):
    f = open(filename, 'w')
    for vektorZnacilnic in znacilnice:
        string = str(vektorZnacilnic)
        f.write(string + '\n')
    f.close()

def barveToSimpleList(barve):
    simpleList = []
    for barva in barve:
        simpleList.append(barva[0])
        simpleList.append(barva[1])
        simpleList.append(barva[2])

    return simpleList

def arrayToString(array):
    string = ""
    for i in range(0,len(array)):
        if i != (len(array)-1):
            string= string + str(array[i]).replace('\n','') + ";"
        else:
            string = string + str(array[i])

    return string

def getZnacilnice(dir):
    ares.getPeaksForPicture(dir,"slike/greyscale/",3)
    znacilniceAll = []
```

```
    files = listFiles(dir)
    print files
    for file in files:
        image = cv2.imread(dir + file)
        height, width, depth = image.shape
        barve = colorz(dir + file,4)
        barve_hsv = colorz_hsv(dir + file,2)
        barvniHistogrami = stdDevHistograms(image)
        info = IPTCInfo(dir + file)
        keywords = info.keywords
        znacilnice = barveToSimpleList(barve_hex2rgb(barve)) +
        barveToSimpleList(barve_hsv2rgb(barve_hsv)) +
        readFromFile.returnBestResolutions(5, "peaks/" +
        file.rstrip('.jpg') + ".txt")
        znacilnice.append(barvniHistogrami[0])
        znacilnice.append(barvniHistogrami[1])
        znacilnice.append(barvniHistogrami[2])
        znacilnice.append(keywords[0])
        znacilnice.append(keywords[1])
        znacilnice = barve_hex2rgb(barve) +
        barve_hsv2rgb(barve_hsv) +
        list(stdDevHistograms(image)) +
        keywords
        znacilnice.append(float(height/float(width)))
        znacilnice.append(calculateObseg(keywords))
        znacilniceAll.append(znacilnice)
    return znacilniceAll

def normalizeArray(array):

    sum = 0
    bigArray = []
    for subArray in array:
        for element in subArray:
            sum += pow(float(element), 2)

        sum = sqrt(sum)
        newArray = []
        for element in subArray:
            newArray.append(float(element)/sum)
        bigArray.append(newArray)

    return bigArray



arrayZnac = np.array(getZnacilnice("Artists/Van_Gogh/"))
arrayZnac = np.array(normalizeArray(arrayZnac))

f = open("pca_znacilniceVanGogh.txt","w")

for element in arrayZnac:
    f.write(arrayToString(element) +'\n')
f.close()
```

Tabele

Algoritmi

Literatura

[1] Timothy F Cootes, Christopher J Taylor, David H Cooper, in Jim Graham. Active shape models-their training and application. *Computer vision and image understanding*, 61(1):38–59, 1995.

[2] NumPy Developers. NumPy, 2015. Dostopno na `http://www.numpy.org`.

[3] Python Software Foundation. Python, 2015. Dostopno na `https://www.python.org`.

[4] BioID GmbH. BioID Face Database, 2015. Dostopno na `https://www.bioid.com/About/BioID-Face-Database`.

[5] M. Hladnik. *Verjetnost in statistika: zapiski predavanj*. Fakulteta za računalništvo in informatiko, 2002. Dostopno na `https://books.google.si/books?id=j7PoAAAACAAJ`.

[6] C Richard Johnson, Ella Hendriks, Igor J Berezhnoy, Eugene Brevdo, Shannon M Hughes, Ingrid Daubechies, Jia Li, Eric Postma, in James Z Wang. Image processing for artist identification. *Signal Processing Magazine, IEEE*, 25(4):37–48, 2008.

[7] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[8] S. Milborrow in F. Nicolls. Active Shape Models with SIFT Descriptors and MARS. *VISAPP*, 2014.

[9] Luka Šajn in Igor Kononenko. Multiresolution image parametrization for improving texture classification. *EURASIP Journal on Advances in Signal Processing*, 2008:137, 2008.

[10] R. Španzel, A. Medved, T. Kermauner, D. Jančar, in M. Zdovc. *Rudolf Španzel*. Mladinska knjiga, 2005.