UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

Recunoaşterea feţelor

propusă de

Păuleţ Claudia

Sesiunea: *iulie, 2012*

Coordonator ştiinţific
Dr. Adrian Iftene

UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI

FACULTATEA DE INFORMATICĂ

Recunoaşterea feţelor

Păuleţ Claudia

Sesiunea: *iulie,2012*

Coordonator ştiinţific

Dr. Adrian Iftene

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul "*Titlul complet al lucrării*" este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
- codul sursă, imagini etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
- rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi,

Absolvent *Păuleţ Claudia*

_____

(semnătura în original)

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul "*Titlul complet al lucrării*", codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi ,

Absolvent *Păuleţ Claudia*

_____

(semnătura în original)

Introducere

În această lucrare propun o aplicaţie ce tratează probleme de actualitate din domeniul Viziunii Calculatorului: detectarea feţelor în timp real şi recunoaşterea feţelor, scrisă într-un limbaj de nivel înalt: Java. Aplicaţia este împărţită structural în 3 părţi : client, server şi baza de date.

Pe partea de client este implementată detecţia feţelor în timp real, folosind librăria open source OpenCv, ce implementează algoritmul Viola – Jones [1][2]. Acesta se împarte în 2 părţi principale:

- Faza de antrenament, ce este compusă din extragerea de trăsături de tip Haar, identificarea şi antrenarea unor clasificatori prin algoritmul AdaBoost;
- Faza de detecţie a feţelor, bazată pe stagii din cascada de clasificatori care decid dacă în imagine se află sau nu o faţă.

Pe partea de server am implementat algoritmul Nefian – Hayes [3][4] de recunoaştere a feţelor, care foloseşte un HMM integrat (HMM 2D). Algoritmul se împarte în 2 părţi principale:

- În prima fază se antrenează HMM-ul integrat extrăgând vectori de observaţie din setul de imagini de antrenament şi construind parametrii specifici unei persoane. Vectorii de observaţie sunt construiţi din coeficienţii 2D-DCT a pixelilor imaginii.
- În a doua fază se aplică algoritmul Viterbi 2D integrat pe toţi HMM din baza de date, cea mai mică distanţă reprezentând persoana recunoscută.

Aplicaţia este integrată cu api-ul de la Facebook şi este uşor de utilizat, fiind astfel o completare a reţelei sociale, fapt ce îi oferă un grad de atractivitate.

Motivaţie

În ultimul timp se poate observa un interes crescând pentru domeniul Viziunea Calculatoarelor[1] (Computer Vision), care a fost introdus în numeroase domenii de activitate, precum industrie robotizată, supraveghere, navigaţie, interacţiune om – robot, detecţia evenimentelor. Viziunea Calculatoarelor este o arie de interes pentru cercetare şi se încearcă o continuă dezvoltare şi expandare a cunoştinţelor în acest domeniu. Din aceste motive mi-am îndreptat atenţia spre o lucrare ce tratează unele aspecte importante ale acestui domeniu: detecţia feţelor în timp real şi recunoaşterea feţelor.

De asemenea, reţelele sociale au cunoscut de câţiva ani încoace o creştere înfloritoare a numărului de utilizatori şi a timpului petrecut online. Printre acestea se numără şi site-ul Facebook, de aceea am ales ca aplicaţia mea să comunice cu această reţea socială, astfel devenind mai atractivă pentru utilizatorii Facebook şi mai uşor de folosit.

Nu în ultimul rând, am ales sa fie o aplicaţie pe telefonul mobil, pe platforma Android, deoarece aceasta este una din cele mai răspândite şi mai populare platforme pentru telefoanele mobile şi permite flexibilitate în folosirea aplicaţie, fiind astfel accesibilă oriunde, oricând.

[1] Computer Vision - http://en.wikipedia.org/wiki/Computer_vision

CUPRINS

Introducere 5
Motivaţie 6
Cap 1 - Descrierea domeniului de recunoaştere a feţelor 8
1.1. Detecţia feţelor 8
1.2. Recunoaşterea feţelor 13
Cap 2 – Structura aplicaţiei 16
2.1 Clientul 16
2.2 Serverul 18
2.3 Baza de date 29
Cap. 3 – Implementare 30
3.1. Clientul 30
3.1.1. Detecţia feţelor 30
3.1.1. Detalii legate de platforma Android 32
3.2 Serverul 35
3.2.1. Recunoașterea fețelor 35
3.2.1. Legătura cu baza de date 37
Concluzi 39
De asemenea, îmi propun sa extind aplicația să conțină și opțiunea de a rula detecția și recunoașterea fețelor încărcând poze sau videoclipuri din memorie. 39
Bibliografie: 40

Cap 1 - Descrierea domeniului de recunoaştere a feţelor

1.1. Detecţia feţelor

Detecţia feţelor este tehnologia care se ocupă cu determinarea locaţiei şi dimensiunilor feţelor umane în diferite poze, ignorând orice altceva, cum ar fi clădiri, copaci sau corpuri. Detecţia feţelor poate fi privită ca un caz specific de detecţie obiect-clasă, ce se ocupă cu determinarea locaţiei şi dimensiunilor tuturor obiectelor ce aparţin unei clase într-o imagine dată [4].

Tehnici de detecţie a feţelor[1]:

a) Găsirea feţelor în imagini cu fundal controlat[1] – se folosesc doar imagini cu fundal monocolor sau cu un fundal static predefinit. Îndepărtarea fundalului asigură încadrarea feţei.

b) Găsirea feţelor după culoare[1,2] - dacă baza de date conţine imagini color, se poate folosi culoarea specifică a pielii pentru a detecta segmente ale feţei. Dezavantaj: această metodă nu funcţionează pentru toate nuanţele de piele şi nu este robust în condiţii de variaţie a luminii; în plus, se pot alege obiecte cu culori similare pielii datorită folosirii segmentării culorilor. Avantaj: lipsa restricţiilor asupra orientării şi dimensiunilor feţelor permite algoritmului să suporte fundaluri complexe.

c) Găsirea feţelor după mişcare[1,2] – pentru detecţia în timp real se foloseşte faptul că feţele sunt tot timpul în mişcare. Calculând aria în mişcare va rezulta segmentul de faţă. Dezavantaj: pot exista şi alte obiecte care să se mişte pe fundal. O mişcare specifică feţei este clipitul. Detectând un şablon de clipiri într-o secvenţă de imagini se poate detecta prezenţa unei feţe, deoarece ochii sunt dispuşi simetric şi clipesc simultan, eliminând posibilitatea de mişcări similare în video. Fiecare imagine este scăzută din cea anterioară, imaginea diferenţei arătând limitele pixelilor mişcaţi.

d) Folosirea unei combinaţii dintre tehnicile de mai sus[1] – prin combinarea mai multor tehnici se obţin rezultate mai bune. Mai întâi, se poate folosi detecţia după culoare, apoi se aplică modele de feţe pentru a elimina detecţiile false de culoare şi pentru a extrage trăsături precum ochi, nas, gură.

---

[1]Tehnici de detecţie a feţelor - http://www.facedetection.com/facedetection/techniques.htm

[2]"Filipe Tomaz face detection and recognition". W3.ualg.pt. Retrieved 2011-02-15.

e) Găsirea feţelor în medii necondiţionate[1] - propune detecţia feţelor indiferent de mediu, devenind cea mai complexă problemă.

Algoritmul Viola-Jones de detecţie a feţelor [1],[2]

O descoperire importantă în detecţia feţelor este algoritmul Viola-Jones, publicat în 2001, deoarece este primul algoritm de detecţie în timp real şi unul din cei mai eficienţi din punct de vedere al complexităţii.

Tehnica algoritmului se bazează pe utilizarea trăsăturilor de tip Haar[2], care sunt evaluate repede printr-o nouă reprezentare a imaginii. Bazat pe conceptul de „Imagine integrală”[3], se generează un set mare de trăsături şi se utilizează algoritmul AdaBoost[4] pentru a reduce setul supra-complet, iar introducerea arborelui degenerativ de clasificatori furnizează în cazul interferenţelor.

Primul pas al algoritmului este acela de a transforma imaginea de intrare într-o imagine integrală, realizată prin schimbarea valorii fiecărui pixel, astfel încât să fie egală atât cu suma tuturor pixelilor de deasupra lui, cât şi a tuturor pixelilor din stânga lui. Aceasta permite calcularea sumei oricărui dreptunghi folosind doar 4 valori, cele din colţurile figurii, precum se demonstrează în figura 1.1.

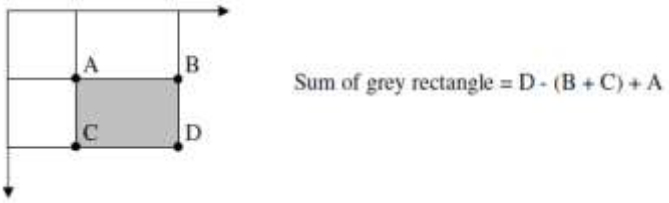

Fig. 1.1. Calcularea sumei unui dreptunghi. [3]

[1] Tehnici de detecţie a feţelor - http://www.facedetection.com/facedetection/techniques.htm

[2] Trăsături de tip Haar - Viola and Jones, "Rapid object detection using boosted cascade of simple features", Computer Vision and Pattern Recognition, 2001

[3] Imagine integrală - http://en.wikipedia.org/wiki/Integral_imaging

[4] AdaBoost algoritm - AdaBoost Presentation, summarizing Adaboost(Jan Sochman, Jiri Matas)

Algoritmul Viola-Jones analizează o sub-fereastra dată folosind trăsăturile formate din 2 sau mai multe dreptunghiuri. Se folosesc trăsăturile[1] simple în locul pixelilor, deoarece trăsăturile pot codifica cunoştinţe dificile de învăţat prin antrenamente pe un număr finit de date şi deoarece sistemele bazate pe trăsături sunt mult mai rapide decât cele bazate pe pixeli [3].

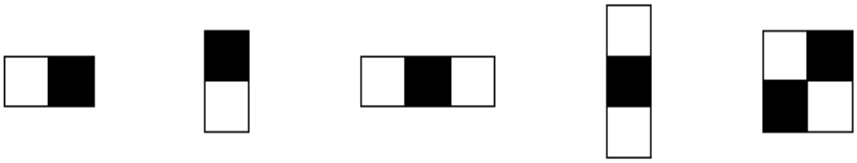

Fig. 1.2. Diferite tipuri de trăsături. [3]

Fiecare trăsătură are o singură valoare, calculată scăzând suma dreptunghiurilor albe din suma dreptunghiurilor negre. Scopul este de a crea o plasă de trăsături capabile de a detecta feţe.

Pentru aceasta se foloseşte algoritmul AdaBoost[2], un algoritm de învăţare ce primeşte un set de trăsături şi un set de antrenament de imagini negative şi pozitive şi îşi construieşte un clasificator puternic prin combinarea măsurată a clasificatorilor slabi. Orice trăsătură este considerată a fi un potenţial clasificator slab, reprezentat matematic sub forma din figura 1.3.

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) > p\theta \\ 0 & \text{otherwise} \end{cases}$$

Fig 1.3. Forma matematică a unui clasificator slab, unde: x = sub-fereastră de 24*24 pixeli,

f = trăsătura aplicată, p = polaritatea, θ = pragul care decide dacă x e pozitiv sau negativ.[3]

---

[1] Trăsături de tip Haar - Viola and Jones, "Rapid object detection using boosted cascade of simple features", Computer Vision and Pattern Recognition, 2001

[2] AdaBoost algoritm - AdaBoost Presentation, summarizing Adaboost(Jan Sochman, Jiri Matas)

- Given examples images $(x_1,y_1),...,(x_n,y_n)$ where $y_1=0,1$ for negative and positive examples.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_1=0,1$, where m and l are the numbers of positive and negative examples.
- For $t=1,...,T$:

1) Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$

2) Select the best weak classifier with respect to the weighted error:

$$\varepsilon_t = \min_{f,p,\theta} \sum_i w_i \left| h(x_i, f, p, \theta) - y_i \right|$$

3) Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t and θ_t are the minimizers of ε_t.

4) Update the weights:

$$w_{t+1,i} = w_{t,i} \beta^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly and $e_i = 1$ otherwise, and $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

- The final strong classifier is:

$$C(x) = \begin{cases} 1 \text{ if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 \text{ otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

Fig. 1.4. Algoritmul AdaBoost modificat (în pseudocod). [3]

O parte importantă a algoritmului este determinarea celei mai bune trăsături, a polarităţii şi a pragului. Cea mai bună trăsătură e determinată pe baza erorii de greutate care o produce. După cum se poate vedea în figura 1.4, greutatea unui exemplu bine clasificat este scăzută, iar a unui exemplu clasificat greşit e păstrată constantă. Astfel, a doua trăsătură este obligată să dea mai multă atenţie exemplelor clasificate greşit de către prima trăsătură.

Deoarece timpul de evaluare este constant indiferent de mărimea datelor de intrare, iar în cadrul unei singure imagini un număr excesiv de mare de sub-ferestre evaluate sunt negative, un singur clasificator puternic devine ineficient şi prin urmare apare nevoia unei cascade de clasificatori[1]. Aceasta e construită din stagii ce conţin câte un clasificator puternic, fiecare stagiu având scopul de a determina dacă o sub-fereastră nu este cu siguranţă o faţă sau poate este o faţă. [3]

[1]Cascade classifiers - Viola, Jones: Robust Real-time Object Detection, IJCV 2001

Când se găseşte o non-faţă, sub-fereastra este aruncată imediat. Când o sub-fereastră e clasificată ca o posibilă-faţă, ea este trecută la următorul stagiu din cascadă, precum se poate observa în figura 1.5.

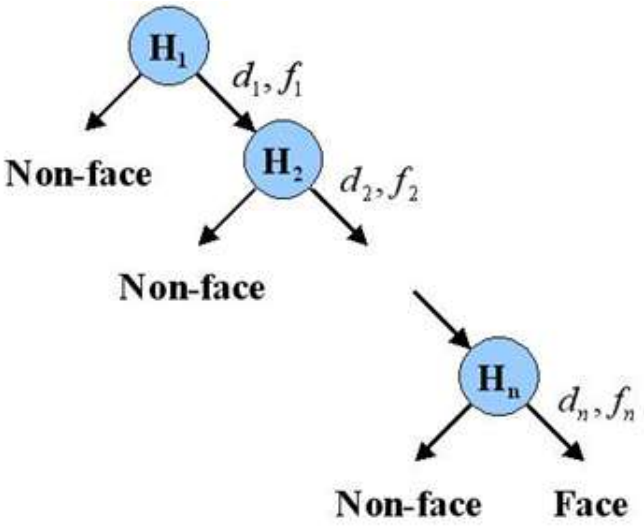

Fig. 1.5. Clasificarea în cascadă a sub-ferestrelor.[1]

Astfel Paul Viola şi Michael Jones au prezentat o abordare de detecţie a feţelor care minimizează timpul computaţional, în timp ce asigură acurateţea detecţiei.[6]

[1] http://cristi.selfip.com/wordpress/2011/11/algoritmi-de-detectia-fetelor-clasificatori-harr-algoritm-viola-jones/

1.2. Recunoaşterea feţelor

Recunoaşterea feţelor[1] este procesul de identificare şi verificare a feţelor detectate într-o imagine, făcând potrivirea cu una din multele feţe cunoscute sistemului (compararea unei trăsături selectate cu imaginile dintr-o bază de date). Astfel, recunoaşterea feţelor complementează detecţia feţelor.

Algoritmii de recunoaştere pot fi divizaţi în două abordări principale[1,2]:

a) Geometric – caută trăsături de distincţie (bazat pe trăsături);

b) Fotometric – o abordare statistică care distilează imaginea în valori şi compară valorile cu şabloane pentru a elimina variaţiile (bazat pe vedere).

Pentru că interesul pentru recunoaşterea feţelor a continuat, s-au dezvoltat mulţi algoritmi, trei dintre care au fost bine studiaţi în literatura de specialitate [10].

1) Analiza componentelor principale[3] (PCA) [10]. Adesea referit ca algoritmul ce foloseşte eigenface-uri[4], PCA a fost tehnica introdusă de Kirby şi Sirivich în 1988. Galeria de imagini şi imaginile de probă trebuie să aibă aceeaşi dimensiune şi să fie normalizate, astfel încât ochii şi gura subiecţilor să se alinieze în imagini.

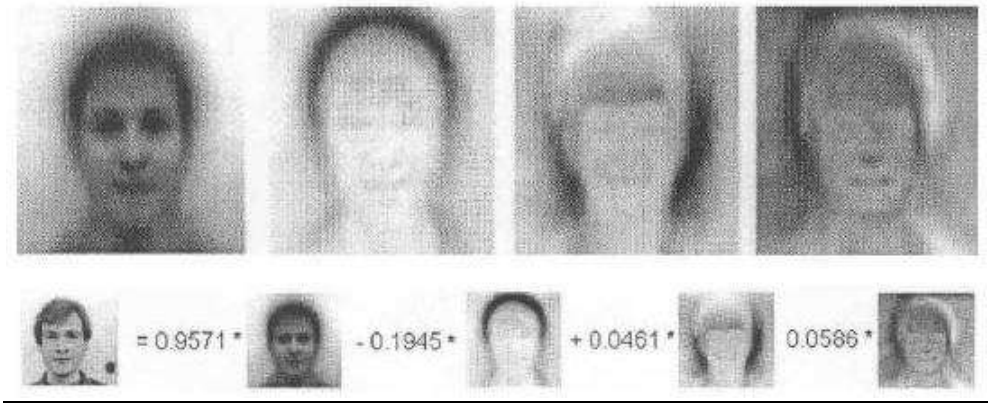

Fig. 1.6. Vizualizare a abordării cu eigenface-uri: fiecare faţă este reprezentată ca o combinaţie liniară de eigenface-uri[5].

[1] Smith, Kelly. "Face Recognition"

[2] http://www.biometrics.gov/Documents/facerec.pdf

[3]Principal Components Analysis - http://en.wikipedia.org/wiki/Principal_component_analysis

[4]Eigenface - D. Pissarenko (2003). Eigenface-based facial recognition

[5]http://ml.cecs.ucf.edu/meli/wiki/index.php/Face_Recognition_Using_Principal_Components_Analysis_(PCA)

Apoi se folosesc mijloace de compresie a datelor şi se selectează structura de şabloane faciale cea mai eficientă şi mai mică din punct de vedere al dimensiunii. Această reducere descompune structura feţei în componente ortogonale, cunoscute drept eigenface-uri. Fiecare imagine poate fi reprezentată ca o sumă de greutăţi (vector de trăsături[1]) al eigenface-urilor, stocate într-un şir 1D, precum se observă în figura 1.6. O imagine de probă e comparată cu o imagine din galerie măsurând distanţa dintre vectorii de trăsături.

PCA necesită faţa completă şi frontală pentru a obţine performanţe. Principalul avantaj al acestei tehnici este că poate reduce necesităţile de date pentru identificarea unui individ la 1/1000 din datele prezentate.

2) Analiza discriminantului linear[2] (LDA) [10]. Este o abordare statistică de clasificare a mostrelor de clase necunoscute, bazându-se pe mostre de antrenament din clase cunoscute. Această tehnică are scopul de a maximiza variaţia între clase (S_s) şi de a minimiza variaţia înăuntrul unei clase (S_w), cu alte cuvinte, de a maximiza raţia $\det|S_s|/\det|S_w|$[3]. Când se lucrează cu date de dimensiune mare, această tehnică se înfruntă cu problema mostrei de dimensiune mică, care apare atunci când există un număr restrâns de mostre de antrenament disponibile comparativ cu dimensiunea spaţiului mostrei.

Fig. 1.7. Exemplu de 6 clase folosind LDA.[4]

[1]Eigenvector - "Eigenvector". Wolfram Research, Inc.. Retrieved 29 January 2010

[2]Liniar Discriminant Analysis - http://en.wikipedia.org/wiki/Linear_discriminant_analysis

[3] http://www.face-rec.org/algorithms/#Image

[4] http://www.biometrics.gov/Documents/facerec.pdf

3) Potrivirea prin graficul ciorchine elastic[1] (EBGM) [10]. Se bazează pe faptul că feţele din realitate au multe caracteristici non-liniare, care nu sunt adresate de alte metode liniare. O transformare Gabor wavelet[2] creează o arhitectura dinamică de legături, care proiectează faţa într-un grid elastic. Un jet Gabor[3] este un nod în gridul elastic (notat prin cercuri în figura 1.8), ce descrie comportamentul imaginii în jurul unui pixel. Rezultatul unei convoluţii a imaginii cu un filtru Gabor[4] este folosit pentru a detecta forme şi extrage trăsături, folosind procesarea imaginii. Recunoaşterea se face pe baza similarităţii răspunsului filtrului Gabor pe fiecare nod Gabor.

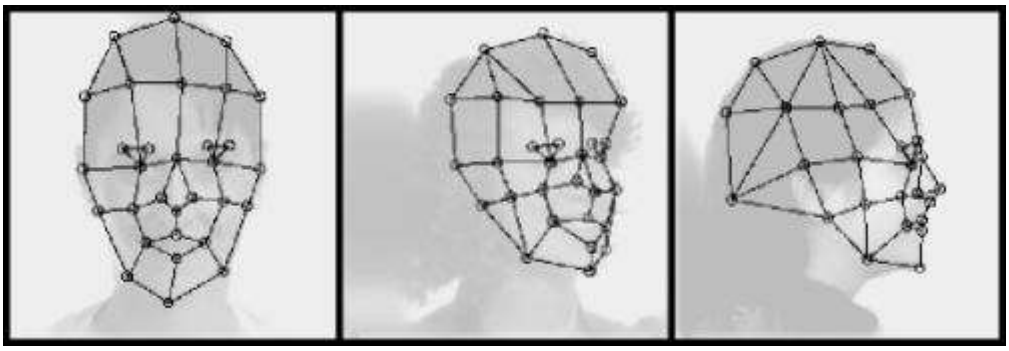

Fig. 1.8. Imagini transformate Gabor wavelet.[5]

[1] Elastic Bunch Graph Matching - http://www.face-rec.org/algorithms/#Image

[2] Wavalet transform - Chui, Charles K. (1992). An Introduction to Wavelets.

[3] Jet - Saunders, D. J., "The Geometry of Jet Bundles", Cambridge University Press, 1989

[4] Gabor filter - Movellan, Javier R.. "Tutorial on Gabor Filters"

[5] http://www.biometrics.gov/Documents/facerec.pdf

Cap 2 – Structura aplicaţiei

Aplicaţia are o structură de tip client – server – bază de date, după cum urmează.

2.1 Clientul

Clientul aplicaţiei este principala comunicare cu utilizatorul şi a fost dezvoltat pe platforma Android (limbajul Java). La deschiderea aplicaţiei utilizatorul este rugat să se logheze cu contul de Facebook (după cum se observă în Fig. 2.1), aplicaţia trimiţând la server informaţiile utilizatorului, care sunt ulterior folosite în procesul de recunoaştere.

Fig. 2.1. Logare cu Facebook.

O dată logat, utilizatorul poate îndrepta camera spre diferite persoane, aceasta detectând toate feţele ce se încadrează în ecran, în timp real. La detectarea unei noi feţe se desenează pe ecranul telefonului mobil un pătrat verde care încadrează detecţia (Fig. 2.2). Detecţia feţelor în timp real este implementată folosind librăria open source OpenCv, ce aplică algoritmul Viola – Jones [1][2]. Acesta se împarte în 2 părţi principale:

- Faza de antrenament, ce este compusă din extragerea de trăsături de tip Haar, identificarea şi antrenarea unor clasificatori prin algoritmul AdaBoost;
- Faza de detecţie a feţelor, bazată pe stagii din cascada de clasificatori care decid dacă în imagine se află sau nu o faţă.

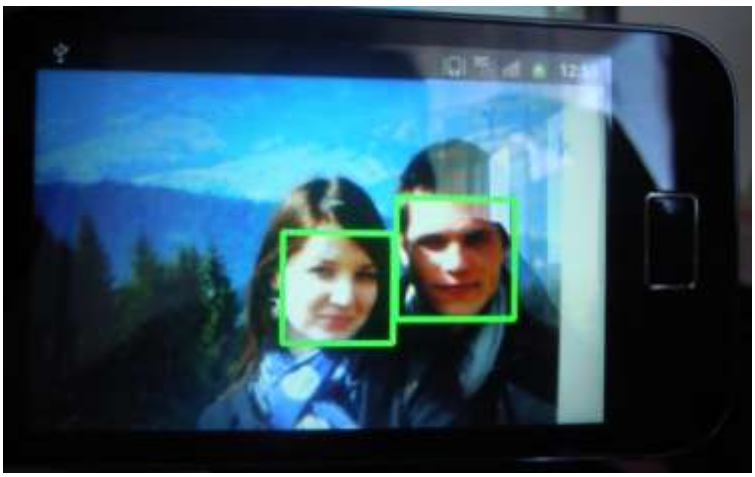

Fig. 2.2. Rezultate pentru detecţia feţelor.

Utilizatorul poate da click pe unul din pătratele ce încadrează feţele detectate la un moment dat pentru a trimite la server o cerere de recunoaştere. La primirea răspunsului de la server, utilizatorul poate vedea unele informaţii despre persoana recunoscută: nume, vârstă, sex, statusul relaţiei, precum şi un link către profilul de Facebook al persoanei respective (Fig. 2.3.). În acest moment se poate da click pe link şi părăsi aplicaţia, deschizându-se adresa în browser, sau se poate reveni la procesul de detecţie a feţelor de cate ori se doreşte.

Fig. 2.3. Rezultatele cererii de recunoaştere a feţelor.

2.2 Serverul

Serverul este scris în limbajul Java şi are două funcţionalităţi principale:

1) Face legătura între client şi baza de date.
2) Rulează algoritmii de recunoaştere a feţei.

1) Legătura dintre client şi baza de date.

Serverul comunica cu clientul prin protocolul TCP/IP printr-o serie de mesaje ce aparţin de o biblioteca comună, trimise prin serializare. În momentul logării utilizatorului pe client, serverul primeşte mesajul de logare, sub formă de obiect JSON, pe care îl poate salva în baza de date (daca utilizatorul nu există) sau îl poate updata (dacă utilizatorul există şi are informaţii modificate). În momentul în care se face o cerere de recunoaştere din partea clientului, serverul încarcă din baza de date pozele aferente fiecărui utilizator. Dacă cererea de recunoaştere are un răspuns pozitiv, serverul atribuie din baza de date informaţiile persoanei recunoscute şi le trimite clientului. Schema comunicării dintre client şi server, şi server şi baza de date este prezentată în Fig. 2.4.

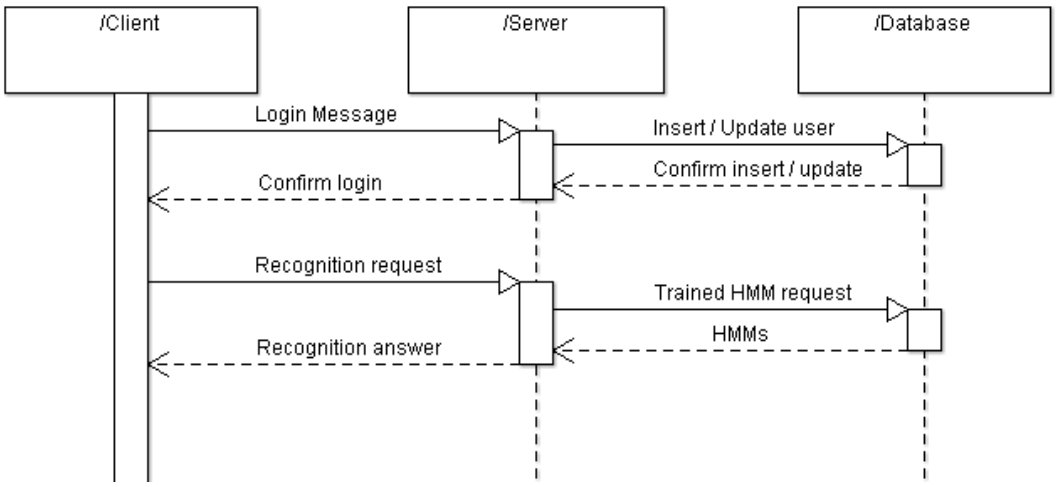

Fig. 2.4. Comunicarea client – server – baza de date.

2) Algoritmul de recunoaştere a feţelor [3][4].

Ca algoritm de recunoaştere a feţei am implementat algoritmul propus de Ara V. Nefian[2] şi Monson M. Hayes III.

Algoritmi anteriori de recunoaştere a feţelor cuprind metoda corelaţiei, metoda vectorilor proprii şi metoda discriminantului linear. Pentru că în aceste metode rata de recunoaştere descreşte semnificativ atunci când orientarea feţei sau mărimea imaginii se modifică, Nefian şi Hayes încearcă o noua abordare, folosindu-se de Modelele Markov Ascunse[3] (HMM), care s-au bucurat până în prezent de un mare succes în recunoaşterea vorbirii.

HMM integrat

Metodele bazate pe HMM au pornit de la ideea ca cele mai semnificative trăsături faciale a unei imagini frontale apar într-o ordine naturală, de sus în jos: frunte, ochi, nas, gură şi bărbie.

De aceea, imaginea unei feţe poate fi modelată cu un HMM 1D atribuind fiecare dintre aceste regiuni unei stări. Iniţial, HMM-ul 1 dimensional a fost extins la un HMM pseudo 2 dimensional de către Samaria[4], adăugând un bloc de marcare la sfârşitul fiecărei linii din imagine şi o stare de sfârşit de linie la sfârşitul fiecărui automat HMM orizontal, pentru a păstra forma 2D a imaginii, după cum se poate observa în Fig. 2.5.

[2] Ara V. Nefian - http://ti.arc.nasa.gov/profile/anefian/
[3] Hidden Markov Model - http://en.wikipedia.org/wiki/Hidden_Markov_model
[4] F. Samaria and S. Young, „HMM based architecture for face identification”, Image and Computer Vision, October 1994.

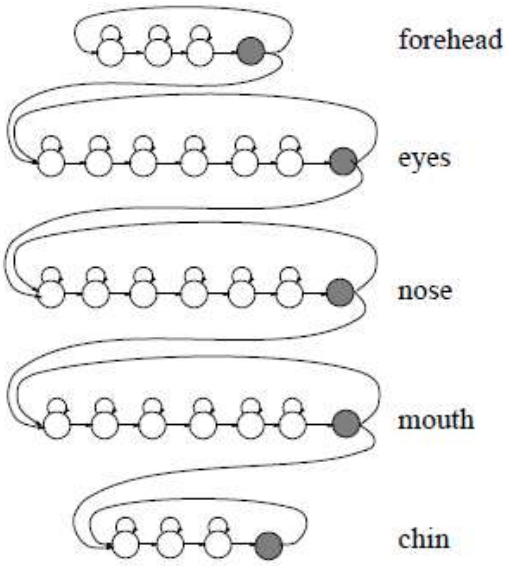

Fig. 2.5. HMM 1 dimensional cu stare de sfârşit de linie.[4]

Algoritmul de recunoaştere a feţelor propus de Nefian şi Hayes se bazează însă pe un HMM integrat (embedded HMM), 2D, folosit de Kuo şi Agazzi[5] pentru recunoaşterea caracterelor (OCR).

Un HMM 1D poate fi generalizat pentru a-i oferi o structură 2D prin permiterea fiecare stări din HMM-ul general să fie un HMM. Astfel, HMM-ul constă într-un set de superstări, fiecare din acestea având un set de stări integrate. Superstările sunt folosite pentru a modela datele într-o direcţie (pe înălţimea pozei), în timp ce stările integrate modelează datele pe cealaltă direcţie (pe lăţime).

Elementele unui HMM integrat sunt:

- Numărul de superstări, N_0, şi setul de superstări, $S_0 = \{S_{0,i}\}$, $0 \leq i \leq N_0$. Starea modelului la timpul t este dată de $q_t \in S_0$, $0 \leq t \leq T$, unde T este lungimea secvenţei de observaţie.
- Setul iniţial de distribuţie a superstărilor, $\Pi_0 = \{ \pi_{0,i}\}$, unde $\pi_{0,i}$ reprezintă probabilitatea de a fi în starea i la timpul 0;
- Matricea de tranziţie între superstări, $A_0 = \{a_{0,ij}\}$, $0 \leq i, j \leq N_0$, unde $a_{0,ij}$ reprezintă probabilitatea de a trece din superstarea i în superstarea j: $a_{0,ij} = P[q_t = S_j / q_{t-1} = S_i]$, cu constrângerea că $\sum_{j=1}^{N0} a0, ij = 1$;

[5] Kuo and Agazzi, „ Keyword spotting in poorly printed documents using pseudo 2 - d Hidden Markov Models, IEEE Transactions on Pattern analasys and Machine Intelligence, 1994

- Parametrii HMM-urilor integrat Λ, care includ :
  - Numărul de stări integrate din superstarea k, $N_1^{(k)}$, şi setul de stări integrate, $S_1^{(k)} = \{S_{1,i}^{(k)}\}$, $0 \leq i \leq N_1^{(k)}$;
  - Setul iniţial de distribuţie a stărilor, $\Pi_1^{(k)} = \{\pi_{1,i}^{(k)}\}$, unde $\pi_{1,i}^{(k)}$ reprezintă probabilitatea de a fi în starea i a superstării k la timpul 0;
  - Matricea de tranziţie între stări, $A_1^{(k)} = \{a_{1,ij}^{(k)}\}$, $0 \leq i, j \leq N_1^{(k)}$, unde $a_{1,ij}$ reprezintă probabilitatea de a trece din starea i în starea j (cu aceleaşi constrângeri ca mai sus).
- Matricea de probabilităţi de emitere: $B^{(k)} = \{b_i^{\{k\}}(O_{t0,t1})\}$ pentru setul de observaţii, unde $O_{t0,t1}$ reprezintă vectorul de observaţie la linia t_0 şi coloana t_1. Într-un HMM cu valori continue, stările sunt caracterizate de funcţii de densitate de observaţii continue. Funcţia tipică de densitate folosită este dată în Fig. 2.6.

$$b_i^{(k)}(\mathbf{O}_{t_0,t_1}) = \sum_{m=1}^{M} c_{im}^{(k)} N(\mathbf{O}_{t_0,t_1}, \mu_{im}^{(k)}, \mathbf{U}_{im}^{(k)}) \quad (1)$$

where $1 \leq i \leq N_1^{(k)}$, $c_{im}^{(k)}$ is the mixture coefficient for the mth mixture in state i of super state k. $N(\mathbf{O}_{t_0,t_1}, \mu_{im}^{(k)}, \mathbf{U}_{im}^{(k)})$ is a Gaussian pdf with mean vector $\mu_{im}^{(k)}$ and covariance matrix $\mathbf{U}_{im}^{(k)}$.

Fig. 2.6. Formulă de calcul pentru simbolul $b_i^{\{k\}}$. [4]

Fie $\Lambda^{(k)} = (\Pi_1^{(k)}, A_1^{(k)}, B^{(k)})$ setul de parametri care defineşte superstarea k. Folosind o notaţie prescurtată, un HMM integrat poate fi definit ca un triplet de forma:

$$\lambda = (\Pi_0, A_0, \Lambda), \text{ unde } \Lambda = \{\Lambda^{(1)}, \Lambda^{(2)} \ldots \Lambda^{(No)}\}.$$

Deşi mai complex decât un HMM 1D, un HMM integrat are o serie de avantaje. În primul rând, complexitatea este proporţională cu suma pătratelor numărului de stări: $\sum_{k=1}^{No}(N_1^{(k)})^2$, astfel încât se reduce complexitatea calculelor atât în etapa de antrenare, cât şi în cea de recunoaştere. În al doilea rând, se pot obţine estimări iniţiale mai bune pentru parametrii modelului.

În plus, acest model este mai potrivit pentru modelarea feţelor pentru că exploatează o trăsătură facială foarte importantă: feţele frontale păstrează aceeaşi structură ca superstările, de sus în jos, şi aceeaşi structură a stărilor din superstări, de la stânga la dreapta. Structura stărilor unui HMM integrat cu probabilităţi de tranziţie diferite de 0 este prezentat în Fig. 2.7.

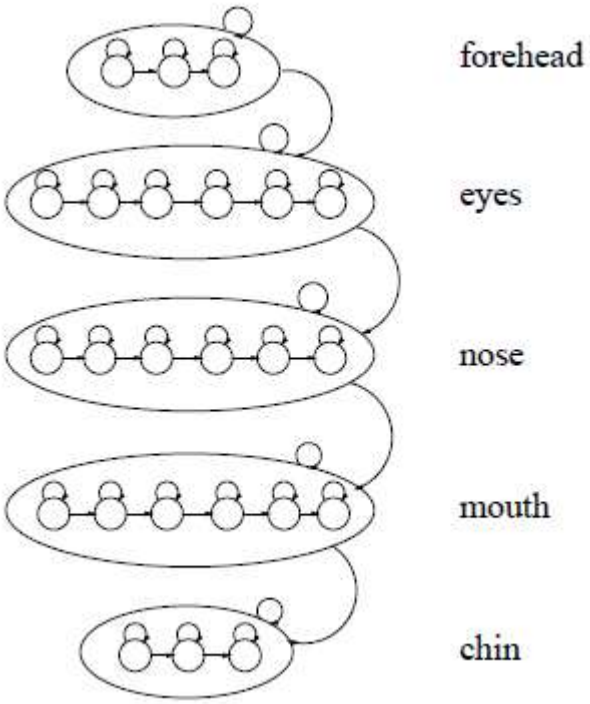

Fig. 2.7. HMM integrat pentru recunoaşterea feţelor.[4]

Vectorii de observaţie

Din fiecare poză de lungime W şi înălţimea H se creează vectori de observaţie. Aceştia se obţin folosind tehnica din Fig. 2.8., unde o imagine este scanata de o fereastră de dimensiune P x L, de la stânga la dreapta şi de sus în jos. Suprapunerea între ferestre adiacente este de M pe linii şi Q pe coloane. Anterior, ca vectori de observaţie se foloseau toţi pixelii imaginii, de aceea lungimea vectorului de observaţie era de P x L.

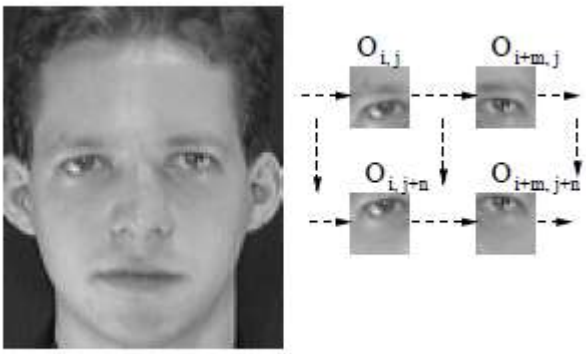

Fig. 2.8. Extragerea blocurilor de observaţie.[4]

Folosirea pixelilor imaginii are însă 2 mari dezavantaje:

- Valorile pixelilor nu reprezintă trăsături robuste, de vreme ce sunt sensibili la zgomot, rotaţia imaginii sau shiftare şi schimbare de luminozitate.
- Dimensiunea mare a vectorului de observaţie conduce la complexitate computaţională atâta în faza de training, cât şi în faza de recunoaştere.

Aceste 2 dezavantaje pot fi decisive atunci când se lucrează pe o bază de date voluminoasă sau când sistemul de recunoaştere este folosit pentru aplicaţii în timp real. De aceea am folosit vectori de observaţie construiţi din coeficienţii 2D-DCT[6] (Discrete Cosine Transform 2D), din fiecare bloc. Proprietăţile de compresie şi corelare pe care le au coeficienţii 2D-DCT îi fac să fie potriviţi pentru a fi luaţi drept vector de observaţie (Fig. 2.10). Folosirea coeficienţilor 2D-DCT în locul valorilor pixelilor reduc drastic dimensiunea vectorilor de observaţie, şi implicit, a complexităţii sistemului de recunoaştere.

Coeficienţii 2D-DCT se calculează după formula din Fig. 2.9.

[6] Discrete Cosine Transform (DCT) - http://en.wikipedia.org/wiki/Discrete_cosine_transform

$$C_u = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{else} \end{cases}$$

$$C_v = (\text{similar to the above})$$

$$F_{vu} = \frac{1}{4} C_v C_u \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} S_{yx} cos\left(v\pi \frac{2y+1}{2N}\right) cos\left(u\pi \frac{2x+1}{2N}\right)$$

Fig. 2.9. Formula de calcul pentru 2D-DCT.[7]

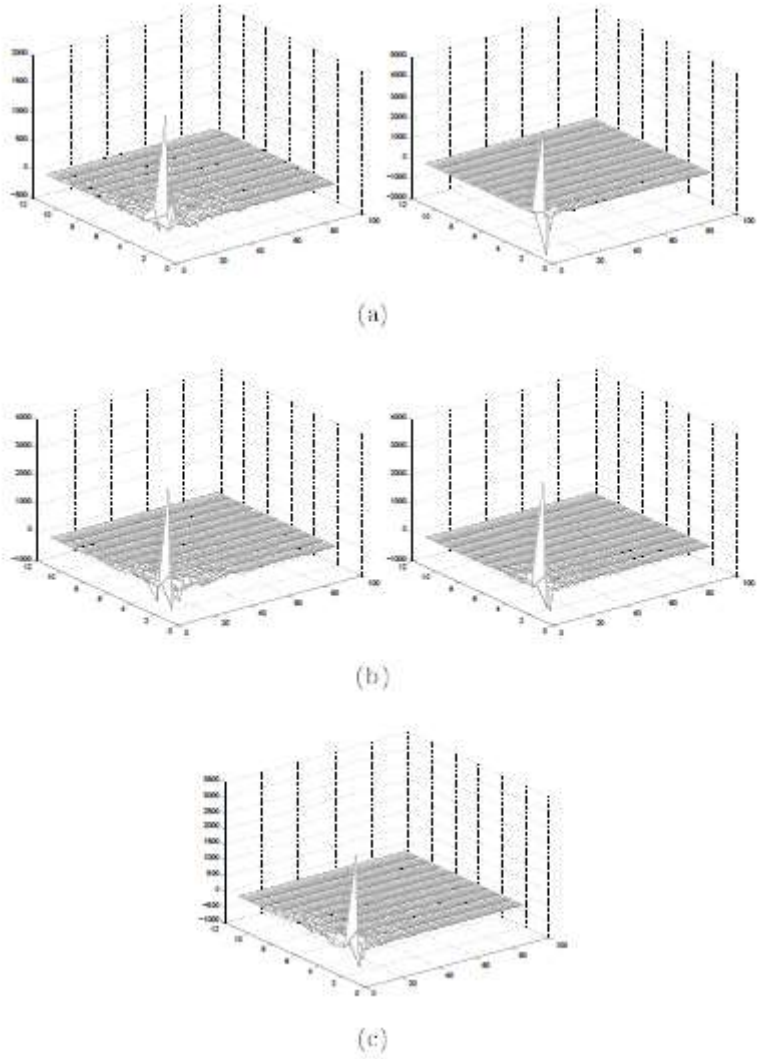

Fig. 2.10. Coeficienţi 2D-DCT tipici pentru: a) par (stânga) şi frunte (dreapta), b) ochi (dreapta) şi nas (stânga), c) gură.[3]

[7] 2D – DCT: http://unix4lyfe.org/dct/

Antrenarea

Fiecărei persoana din baza de date îi corespunde un HMM integrat. Pentru a obţine acest model, pentru o persoană folosim un set de poze ce reprezintă mai multe instanţe ale aceleiaşi feţe. Sunt folosiţi vectorii de observaţie extraşi din fiecare bloc pentru a antrena HMM-ul, urmând paşii din Fig. 2.11.

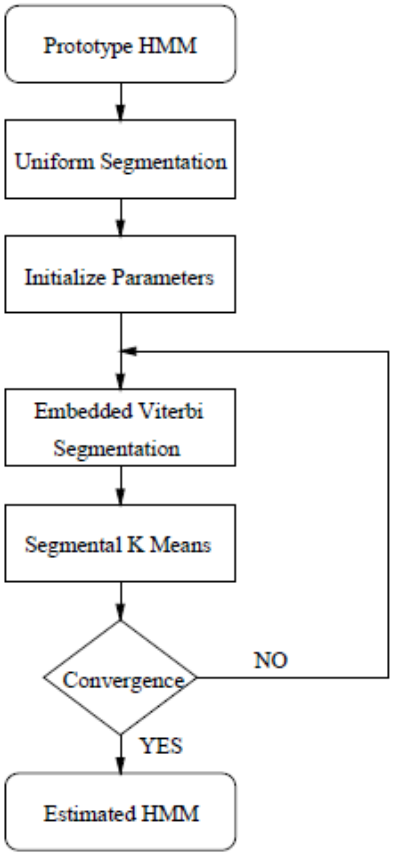

Fig. 2.11. Schema de antrenare a HMM-ului. [4]

Ţinând cont de numărul de superstări, de numărul de stări din fiecare stare şi de structura de sus în jos şi de la stânga la dreapta a prototipului de HMM integrat, datele sunt segmentate uniform pentru a obţine o estimare a parametrilor iniţiali. Mai întâi se împart observaţiile în N_0 superstări, de sus în jos, apoi datele corespunzătoare fiecărei superstări este împărţită uniform , de la stânga la dreapta, în $N_1^{(k)}$ stări.

La următoarea iteraţie segmentarea uniformă este înlocuită de o segmentare Viterbi, folosind o versiune modificată pentru sistemul integrat (Fig. 2.12).

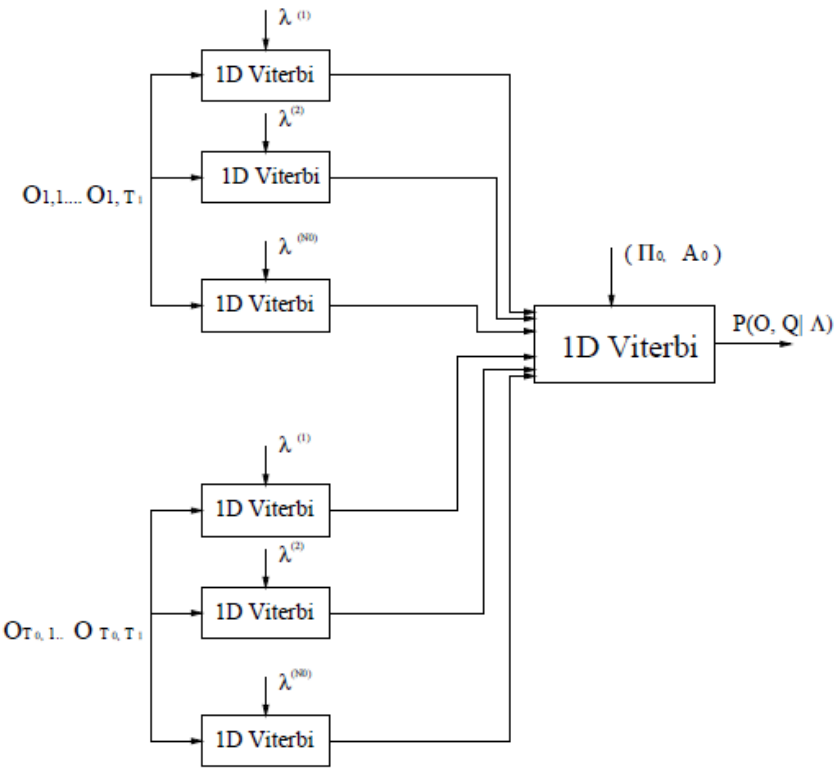

Fig. 2.12. Algoritmul Viterbi dublu integrat. [3]

Algoritmul Viterbi dublu integrat este alcătuit din următorii paşi:

- Mai întâi segmentarea Viterbi este aplicata pe linii, şi sunt calculate probabilităţile;
- Probabilităţile stărilor şi observaţiilor dintr-o linie pe modelul superstării respective obţinute din segmentarea Viterbi reprezintă probabilităţile superstărilor. Acestea împreună cu probabilităţile de tranziţie între superstări A_0 şi probabilităţile iniţiale ale superstărilor Π_0 sunt folosite pentru a aplica segmentarea Viterbi de sus în josul imaginii şi pentru a determina :

$$P(\mathbf{O}_{1,1} \ldots \mathbf{O}_{1,T_1}, \ldots \mathbf{O}_{T_0,1} \ldots \mathbf{O}_{T_0,T_1}, q_{0,1} \ldots q_{0,T_0} | \lambda),$$

  unde $q_{0,to}$, $1 \leq t_0 \leq T_0$ sunt superstările ce corespund liniei t_0.
- Parametrii modelului sunt apoi estimaţi folosind algoritmul de clusterizare k-means[8]. Aceştia se obţin în conformitate cu formulele din Fig. 2.13.:

[8] Algoritmul k-means - http://en.wikipedia.org/wiki/K-means_clustering

$$a_{1,ij}^{(k)} = \frac{number\ of\ transitions\ from\ S_{1,i}^{(k)}\ to\ S_{1,j}^{(k)}}{number\ of\ transitions\ from\ S_{1,i}^{(k)}}$$

$$\mu_i^{(k)} = \text{sample mean of vector } i \text{ in super state } k$$

$$\mathbf{U}_i^{(k)} = \text{sample covariance matrix of vectors in state } i \text{ of super state } k$$

$$a_{0,ij} = \frac{number\ of\ transitions\ from\ S_{0,i}\ to\ S_{0,j}}{number\ of\ transitions\ from\ S_{0,i}}$$

Fig. 2.13. Reguli de estimare a parametrilor. [4]

Iteraţia se încheie şi HMM-ul este iniţializat când diferenţa de scor obţinut de algoritmul Viterbi la iteraţii consecutive este mai mic decât un prag.

Recunoaşterea feţelor

Pe poza pe care se încearcă recunoaşterea se aplică paşii de preprocesare a imaginii, obţinându-se astfel un vector de observaţie corespunzător imaginii de test. Pe acesta se calculează probabilitatea de emitere a secvenţei pe un HMM dat, aplicând algoritmul Viterbi dublu integrat (Fig. 2.14.).

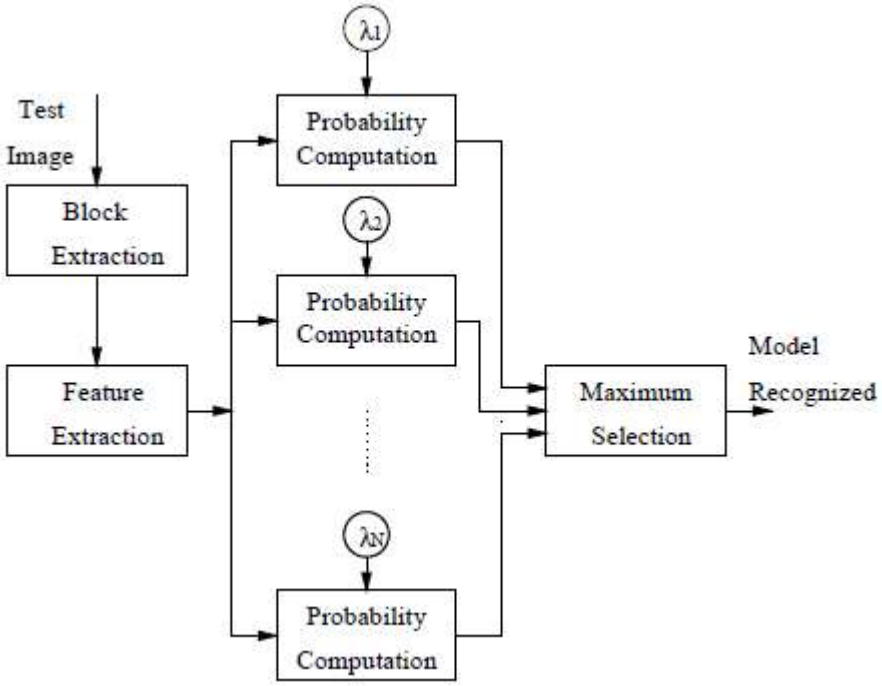

Fig. 2.14. Schema de recunoaştere pentru HMM. [3]

Procedeul se repetă pe toţi HMM din baza de date, pe rând, şi este ales modelul cu cea mai mare credibilitate: imaginea t este recunoscuta ca faţa k dacă:

$$P(\boldsymbol{O}^{(t)}|\lambda_k) = max_n P(\boldsymbol{O}^{(t)}|\lambda_n).$$

2.3 Baza de date

Pentru baza de date am folosit sistemul de gestiune MySQL şi conţine 4 tabele principale: informaţiile utilizatorilor, poze, HMM-uri şi locaţiile curente ale utilizatorilor. Cele 4 table au următoarele structuri, după cum sunt prezentate în tabelele 1, 2, 3 şi 4.

Nume tabel:	Basic Information					
Nume coloana:	User_id	Name	Birthdate	Gender	Rel_sts	Profile_url
Tipul coloanei:	Varchar	Varchar	Varchar	Varchar	Varchar	Varchar

Tabela 1. Structura tabelei de reţinere a informaţiilor utilizatorilor.

Nume tabel:	Facebook Photoes		
Nume coloana:	Id	User_id	Photo
Tipul coloanei:	Varchar	Varchar	Blob

Tabela 2. Structura tabelei de reţinere a pozelor utilizatorilor.

Nume tabel:	Current Location		
Nume coloana:	Loc_id	User_id	Curr_location
Tipul coloanei:	Varchar	Varchar	Varchar

Tabela 3. Structura tabelei de reţinere a locaţiei utilizatorilor.

Nume tabel:	Embedded HMMs		
Nume coloana:	Id	User_id	HMM
Tipul coloanei:	Varchar	Varchar	Varchar

Tabela 4 Structura tabelei de reţinere a HMM-urilor aferente utilizatorilor.

Cap. 3 – Implementare

3.1. Clientul

3.1.1. Detecţia feţelor

Pentru detecţia feţelor am folosit algoritmul Viola – Johnes [1][2], implementat de biblioteca open source OpenCV[9]. Aceasta conţine deja clasificatorii antrenaţi, rezultatele antrenamentului fiind salvate în fişiere de tip XML. Pentru partea de antrenament a algoritmului, aceşti clasificatori trebuie încărcaţi în memorie, după cum se vede în Fig. 3.1.

```
CascadeClassifier classifier;

// choose between frontal or profile Face Detection
if(option == 0)
      is = context.getResources().openRawResource(
                  R.raw.lbpcascade_frontalface);
else
      is = context.getResources().openRawResource(
                  R.raw.haarcascade_profileface);

// load the XML classifier file
classifier = new CascadeClassifier(cascadeFile.getAbsolutePath());
```

Fig. 3.1. Încărcarea fişierului XML cu datele de antrenament pentru cascada de clasificatori, în clasa FaceDetectionView

Pentru partea de detectare a feţelor, pentru a fi posibilă detecţia în timp real, este nevoie de conectare la camera aparatului, fapt realizat prin intermediul clasei `VideoCapture`, ce comunică cu componenta hardware a dispozitivului şi a interfeţei `SurfaceHolder.Callback` ce permite modificarea afişării pe ecran în mod programatic. La fiecare captură imaginea este preluată atât color, cât şi alb-negru şi este procesată precum în Fig. 3.2. , în următorii paşi:

- se defineşte o mărime minimă pentru încadrarea unei feţe (cu cât aceasta este mai mică, cu atât creşte complexitatea procesării imaginii);

[9] OpenCV - http://opencv.willowgarage.com/wiki/

- se iniţiază detecţia pe imaginea alb-negru apelând funcţia `detectMultiScale()` din clasa `CascadeClassifier`, salvându-se coordonatele feţelor detectate.

- se desenează pătrate la coordonatele detectate pe imaginea color şi imaginea rezultată este afişată pe ecran.

```
//connect to the Android camera
capture.retrieve(mRgba, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
capture.retrieve(mGray, Highgui.CV_CAP_ANDROID_GREY_FRAME);

if (classifier != null)
{
	// define a minimum size for the face frame
	int height = mGray.rows();
	int faceSize = Math.round(height * 0.25f);

	//remembers the faces coordonates
	f = new LinkedList<Rect>();

	// initiate detection and save the faces coordonates
	classifier.detectMultiScale(mGray, f, 1.1, 2, 2,
				new Size(faceSize, faceSize));


	//draw green rectangles at the faces coordonates
	for (Rect r : f)
		Core.rectangle(mRgba, r.tl(), r.br(),
				new Scalar(0, 255, 0, 255), 3);
}
bmp = Bitmap.createBitmap(mRgba.cols(), mRgba.rows(),
					Bitmap.Config.ARGB_8888);

// display the picture with face rectangles on the screen
if (bmp != null)
{
	Canvas canvas = surface.lockCanvas();
	if (canvas != null)
	{
		canvas.drawBitmap(bmp,(canvas.getWidth() - bmp.getWidth()) / 2,
				(canvas.getHeight() - bmp.getHeight()) / 2, null);

		surface.unlockCanvasAndPost(canvas);
	}
	bmp.recycle();
}
```

Fig. 3.2. Detecţia feţelor, implementată în clasa FaceDetectionView

3.1.1. Detalii legate de platforma Android

Orice aplicație pe platforma Android necesită setarea configurației, deoarece fiecare aplicație rulează cu o identitate de sistem diferită. De aceea kernelul Linux izolează aplicațiile una de cealaltă și de sistem. Pentru a putea comunica cu acestea trebuie garantate anumite permisiuni aplicației, setate în fișierul XML numit AndroidManifest[10], atașat aplicației. Pentru clientul meu am folosit permisiuni pentru folosirea internetului (atât pentru comunicarea cu serverul, cât și pentru logarea cu Facebook) și pentru folosirea camerei de luat vederi a aparatului. Solicitarea permisiunilor se poate observa în Fig. 3.3. Fișierul AndroidManifest este folosit și pentru a specifica versiunea minimă de platformă Android pe care rulează, activitatea de lansare a aplicației și proprietăți ale fiecărei activități din aplicație.

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
<uses-permission android:name="android.permission.INTERNET"/>
```

Fig. 3.3. Setarea permisiunilor în fișierul AndroidManifest.xml

În sistemul de operare Android, o *activitate*[11] reprezintă un ecran cu care utilizatorul poate interacționa pentru a face o acțiune, fiecărei activități corespunzându-i o fereastră. De fiecare dată când o nouă activitate începe, activitatea anterioară este oprită, dar păstrată într-o stivă numită „the back stack”. Trecerea de la o activitate la alta se face prin intermediul clasei Intent, care poate transmite parametri la activitatea următoare prin intermediul clasei Bundle[12]. Trecerea de la o activitate la alta este exemplificată în Fig. 3.4.

[10] Android Manifest File - http://developer.android.com/guide/topics/manifest/manifest-intro.html
[11] Activity - http://developer.android.com/reference/android/app/Activity.html
[12] Bundle - http://developer.android.com/reference/android/os/Bundle.html

```
//choose the activity to be started
Intent i = new Intent(FaceDetectionActivity.this,
                            RecognitionResponseActivity.class);

//pass parameters
Bundle bun = new Bundle();
bun.putSerializable("response", rsp);
i.putExtras(bun);

// start the new activity
startActivity(i);
```

Fig. 3.4. Trecerea din FaceDetectionActivity în RecognitionResponseActivity, trimițând răspunsul de la server drept parametru.

Un *view*[13] este responsabil de desenarea și de tratarea evenimentelor dintr-o activitate. Pentru activitatea responsabilă cu vizualizarea răspunsului la cererea de recunoaștere a fețelor (RecognitionResponseActivity) am implementat un view propriu ce formatează dispunerea elementelor pe linie din lista activității (Fig. 3.5.). Astfel, `IconifiedTextView` este împărțit în 2 părți:

- în stânga este dispusă imaginea de profil (de pe Facebook) a persoanei recunoscute;

- în dreapta este un view imbricat ce conține 5 TextView-uri suprapuse: numele, data nașterii, sexul, statusul relației și linkul de profil Facebook al persoanei.

Pentru a folosi un view propriu este nevoie de crearea unui adaptor propriu, atașat activității ce va conține view-ul, pentru a face legătura dintre informațiile date ca parametrii view-ului și view.

[13] View - http://developer.android.com/reference/android/view/View.html

```
this.setOrientation(HORIZONTAL);

//create a embedded layout
LinearLayout hor = new LinearLayout(context);
hor.setOrientation(VERTICAL);

mIcon = new ImageView(context);
mIcon.setImageDrawable(aIconifiedText.getIcon());

// left, top, right, bottom
mIcon.setPadding(0, 2, 5, 0); // 5px to the right

// At first, add the Icon
addView(mIcon,  0);

//add the embedded view
hor.addView(nameTextView,  0);

//add the rest of the componetns in the embedded view

//. . .
```

Fig. 3.5. Fragmente din constructorul view-lui IconifiedTextView

3.2 Serverul

3.2.1. Recunoașterea fețelor

Pentru algoritmul Nefian - Hayes [3][4] am implementat următoarele funcții:

1) Funcția 2D-DCT folosită în faza de procesare a imaginii pentru obținerea vectorilor de observație necesari antrenării şi recunoașterii fețelor, în varianta AAN (Arai/ Agui/ Nakajima)[14], care este mult mai eficientă decât metoda clasică, deoarece se aplică 1D-DCT pe linii, apoi pe coloane.
2) Segmentarea uniformă a imaginii în numărul de superstări şi substări.
3) Inițializarea parametrilor modelului, aplicând algoritmul de clusterizare k-means (în maxim 1000 de iterații). În procesul de clusterizare un rol important îl au media, variația și greutatea observațiilor.
4) Reestimarea parametrilor modelului, apelată la fiecare iterație în faza de antrenament, în care se recalculează probabilitățile de tranziție dintre superstări și dintre stările integrate, probabilitățile de emisie a observațiilor, după o variantă a algoritmului Baum-Welch[15] de antrenare a HMM-urilor, adaptat pentru cazul 2D.
5) Algoritmul de segmentare Viterbi: se calculează scorul Viterbi (standard) pe fiecare HMM integrat pentru fiecare linie a imaginii în parte și se decide cărei superstări îi corespunde vectorul de observație. În interiorul superstărilor, făcând traceback, se decide cărei stări îi corespunde fiecare observație.

Pentru a antrena un HMM integrat pentru o singură persoană sunt apelate funcțiile de mai sus, după cum se observă în Fig. 3.6.

```
//prepare the observation vectors
MyEmbeddedHMM eHMM = this.Create2DHMM();
 File folder = new File(directoryName);
for (File f : files)
{
        PictureObservation obs = PictureObservation.loadPicture(f);
         observations.add(obs);
         this.UniformSegmentation(obs, eHMM);
}
  InitMixSegmentation(observations, eHMM);
```

[14] Varianta 2D-DCT propusa de Arai, Agui şi Nakajima - http://unix4lyfe.org/dct/
[15] Algoritmul Baum-Welch - http://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm

```
// train the HMM
while (!isOk && crtIteration < MaxIteration) {
        //reestimate weights, means and covariance
        EstimateHMMParameters(imageObservation, eHMM);
        EstimateTransitionProbability(imageObservation, eHMM);
        double distance = 0;
        for (int i = 0; i < imageObservation.size(); i++)
        {
          EstimateObservationProbability(imageObservation.get(i), eHMM);
          distance += Viterbi2D(imageObservation.get(i), eHMM);
           }
         distance /= imageObservation.size() * (PictureObservation.L * PictureObservation.P);
        //reassign mixture numbers to all observations
        MixSegmentation(imageObservation, eHMM);
        isOk = Math.abs(oldDistance - distance) < 0.01;
        oldDistance = distance;
        crtIteration++;
}
```

Fig. 3.6. Pașii algoritmului de antrenare pentru o persoană. În algoritmul meu, algoritmul se aplica până când se depășesc 1000 de iterații sau până când diferența de scor dintre 2 iterații devine mai mică decât 10^{-2}.

Pentru partea de recunoaștere a feţelor se folosește algoritmul Viterbi 2D și se alege scorul maxim obținut, după cum se observă în Fig. 3.7.

```
double maxLike = Double.NEGATIVE_INFINITY;
// load the test picure to be recognized
 PictureObservation obs = PictureObservation.loadPicture(new File(bitmap));
 for (int i = 0; i < facesDb.size(); i++) {
    MyEmbeddedHMM crtHmm = facesDb.get(i).hmm;
    // aply Viterbi 2D and get observation score
    double distance = Viterbi2D(obs, crtHmm);
    if (distance > maxLike) {
       maxLike = distance;
       bestMatch = i;
    }
 }
 return facesDb.get(bestMatch);
```

Fig. 3.7. Funcția de recunoaștere a feței.

3.2.1. Legătura cu baza de date

Pentru a face legătura cu baza de date MySQL am folosit pachetul „java.sql” și driverul JDBC ce se ocupă de comunicarea cu baza de date și am apelat instrucțiunile din Fig. 3.8.

```
// Load the JDBC driver
String driverName = "com.mysql.jdbc.Driver"; // MySQL MM JDBC driver
Class.forName(driverName);

// Create a connection to the database
String serverName = "localhost";
String mydatabase = "face_recognition";

String url = "jdbc:mysql://" + serverName + "/" + mydatabase; // a JDBC url

String username = "root";
String password = "";
connection = DriverManager.getConnection(url, username, password);
```

Fig. 3.8. Conectarea la baza de date prin driverul JDBC.

Pentru implementarea operațiilor asupra bazei de date, am executat următorii pași: am setat nivelul de izolație al bazei de date, pentru a preveni fenomenul de injection; am creat un savepoint la care sa mă pot întoarce în cazul în care apare o eroare în timpul tranzacției, pentru a nu avea date incomplete sau compromise; am creat query-ul și l-am executat; am comis operația la baza de date. În exemplul din Fig. 3.9. se execută o inserare sau updatare a unui utilizator.

```
try
{
        conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);

        conn.setSavepoint();
        String query = "INSERT INTO basicinformation VALUES(" +
                                                "\'"+ user.getFbId() + "\', " +
                                                "'" + user.getName() + "', " +
                                                "'" + user.getBirthday() + "', " +
                                                "'" + user.getGender() + "', " +
                                                "'" + user.getRel_sts() + "', " +
                                                "'" + user.getProfile_url() + "') ";

        query += "ON DUPLICATE KEY UPDATE " +
                "Name = " + "'" + user.getName() + "', " +
                "Birthdate = " + "'" + user.getBirthday() + "', " +
                "Gender = " + "'" + user.getGender() + "', " +
                "Profile_url = " + "'" + user.getProfile_url() + "', " +
                "Rel_status = " + "'" + user.getRel_sts() + "' ";

        ok = statement.execute(query);

        conn.commit();

}
catch (SQLException e)
{
        conn.rollback();
        log.Println("[DBOperation]: " + e.getMessage());
}
```

Fig. 3.9. Inserare utilizator în baza de date, în cazul în care nu există, sau update, altfel.

Concluzi

În această lucrare am studiat și implementat 2 algoritmi de actualitate și foarte importanți din domeniul Viziunii Calculatoarelor: algoritmul Viola – Johnes [1][2] pentru detecția fețelor în timp real și algoritmul Nefian – Hayes [3][4] pentru recunoașterea fețelor.

În cazul algoritmului Viola – Johnes am constatat că rata fals – pozitivelor (circa 20%) este mult mai mare decât rata fals – negativelor (3%). Procesarea imaginii şi detectarea este realizată în timp real, însă complexitatea algoritmului crește atunci când dimensiunea minimă a feței de detectat crește, astfel încât și timpul de procesare devine mai mare și întreaga aplicație este încetinită.

În cazul algoritmului Nefian – Hayes, rularea algoritmul necesită foarte mult timp pentru faza de antrenament (aprox. 15 minute de persoană, 8 – 9) și nu poate rula în timp real, însă timpul de procesare pentru recunoaștere este relativ mic (aproximativ 5 secunde). Algoritmul are o rată de recunoaștere de 95% pe poze din baza de date, însă rata de recunoaștere scade drastic la primirea pozei de pe telefon, întrucât este afectată calitatea imaginii.

Direcţii de dezvoltare

Pe viitor îmi propun sa îmbunătățesc timpul de procesare pentru algoritmul Nefian – Hayes, faza de recunoaștere a fețelor, prin optimizări ale codului şi ale accesului la baza de date, precum şi rata recunoaşterii pentru pozele cu o calitate mai slabă. In plus, voi face recunoaşterea doar pe acei utilizatori din baza de date care au aceeaşi locaţie curentă ca şi dispozitivul mobil (lucru aflat accesând GPS-ul dispozitivului).

De asemenea, îmi propun sa extind aplicația să conțină și opțiunea de a rula detecția și recunoașterea fețelor încărcând poze sau videoclipuri din memorie. Pe viitor se va retine un istoric al cererilor făcute de fiecare utilizator şi se vor putea vedea ulterior.

Bibliografie:

1. P. Viola, M. Jones: „Rapid Object Detection using a Boosted Cascade of Simple Features", 2001;

2. P. Viola, M. Jones: „Robust real-time object detection", 2001;

3. Ara V. Nefian, Monson H. Hayes III, "Face recognition using an embedded HMM", IEEE Conference on Audio and Visual-based Person Authentication, 1999;

4. Ara V. Nefian, Monson H. Hayes III, Hidden Markov Models for face recognition, IEEE International Conference on Acoustic Speech and Signal Processing 1998;

5. Ole Helving Jensen: „Implementing the Viola-Jones Face Detection Algorithm", 2008 ;

6. http://en.wikipedia.org/wiki/Face_detection

7. http://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework

8. http://www.codeproject.com/Articles/85113/Efficient-Face-Detection-Algorithm-using-Viola-Jon

9. http://opencv.willowgarage.com/wiki/FaceDetection

10. http://www.facedetection.com/index.htm

11. http://www.facedetection.com/facedetection/techniques.htm

12. http://www.biometrics.gov/Documents/facerec.pdf

13. http://www.face-rec.org/algorithms/

14. http://en.wikipedia.org/wiki/Facial_recognition_system

15. Varianta 2D-DCT propusa de Arai, Agui şi Nakajima - http://unix4lyfe.org/dct/

16. OpenCV - http://opencv.willowgarage.com/wiki/

17. DCT - http://en.wikipedia.org/wiki/Discrete_cosine_transform

18. Algoritmul k-means - http://en.wikipedia.org/wiki/K-means_clustering