

Vedere Artificială - Tema 4

Detectare facială folosind metoda glisării ferestrei și histograme de gradienti orientați

Obiectiv:

Scopul acestui proiect este implementarea și testarea unui algoritm de detectare facială folosind metoda glisării unei ferestre și histograme de gradienti orientați.

Funcțiile Python care vă vor ajuta la implementarea proiectului sunt în directorul *cod*; imaginile pe care le veți folosi sunt în directorul *data*.

Introducere. Detectarea obiectelor în imagini este una dintre probleme fundamentale în Vederea Artificială. Detectarea facială înseamnă localizarea tuturor fețelor umane ce apar într-o imagine test (Figura 1). De obicei, localizarea se realizează la nivel de fereastră dreptunghiulară (marcate cu roșu în Figura 1). Camerele digitale moderne precum și instrumentele de organizare a colecțiilor de fotografii (Picasa, iPhoto) au un asemenea detector facial incorporat.

Una din abordările cu cel mai mare succes în rezolvarea acestei probleme o constituie glisarea unei ferestre (termenul în engleză este *sliding-window*) într-o imagine test și folosirea unui clasificator care decide pentru fiecare fereastră pe baza pixelilor ei dacă aceasta conține sau nu o față. Veți urma această paradigmă în cadrul acestei proiect.

Implementare. Există multe implementări posibile ale unui detector facial. Primele abordări de mare succes pe tema detectării faciale datează de la lucrările lui Rowley et. al 1998 și Viola și Jones 2001. În cadrul acestui proiect veți implementa un detector facial care folosește pentru descrierea conținutului vizual al fiecărei ferestre dintr-o imagine

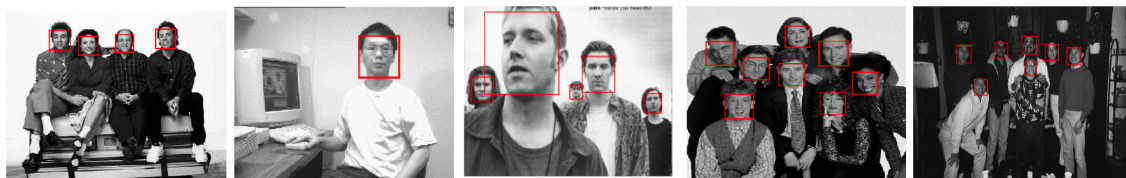


Figura 1: **Exemple de detectare facială:** în fiecare dintre cele 5 imagini sunt localizate la nivel de fereastră dreptunghiulară de culoare roșie toate fețele oamenilor ce apar în imaginea respectivă.

drept caracteristici histograme de gradienti orientați (HOG). Histogramele de gradienti orientați au fost introduse de Dalal și Triggs în anul 2005 (aveți articolul atașat în arhiva .zip) și folosite inițial pentru detectarea oamenilor. În acest proiect veți folosi histogramele de gradienti orientați pentru detectarea facială.

Construcția detectorului facial presupune următoarele etape:

- etapa de antrenare (învățare): în această etapă se antrenează un clasificator care poate distinge între ferestrele ce conțin fețe de cele ce nu conțin fețe. Veți folosi acest clasificator în următoarea etapă pentru a clasifica miile de ferestre dreptunghiulare dintr-o imagine test.
- etapa de testare: pentru o imagine test vreți să localizați toate fețele care apar în ea. Veți realiza acest lucru prin glisarea unei ferestre de la stânga la dreapta și de sus în jos și clasificarea fiecărei ferestre pe baza clasificatorului învățat în etapa anterioară. Trebuie să aveți în vedere că fețele ce apar într-o imagine pot avea mărimi diferite.

Scriptul Python *Run_Project.py* conține implementarea întregului proiect, apelând rând pe rând funcții care realizează pașii de mai sus. Această funcție este completată în întregime. Sarcina voastră este de a completa bucățile lipsă din cod astfel încât *Run_Project.py* să ruleze întreg proiectul. Pe baza acestui lucru veți putea realiza experimentele care vă vor ajuta să răspundeți întrebărilor de la final.

1.1 Etapa de antrenare (4 puncte)

Exemple de antrenare pozitive și negative. Antrenarea unui clasificator se realizează pe baza unei mulțimi de antrenare ce cuprinde exemple de antrenare pozitive (ferestre ce conțin fețe) și exemple de antrenare negative (ferestre ce nu conțin fețe). Pentru acest proiect veți lucra numai cu ferestre de dimensiuni 36×36 pixeli. Câteva exemple pozitive și negative sunt ilustrate în Figura 2. Imaginile ce conțin exemplele pozitive au dimensiunile 36×36 pixeli. Imaginile ce conțin exemple negative sunt de diverse dimensiuni. Este important de remarcat următorul aspect: într-o imagine negativă ce nu conține fețe orice fereastră de dimensiuni 36×36 pixeli reprezintă un exemplu negativ de antrenare.

Toate exemplele de antrenare vor fi prelucrate la nivel de imagini grayscale (tonuri de gri) și nu de imagini RGB (culori). Clasificatorul va fi antrenat astfel încât să decidă pe baza conținutului vizual al unei ferestre dacă aceasta reprezintă sau nu o față.



Figura 2: **Exemple de antrenare.** Rândul de sus: exemple pozitive de antrenare (de dimensiuni 36×36 pixeli). Rândul de jos: imagini negative de antrenare. Fiecare fereastră de dimensiuni 36×36 pixeli din imagine reprezintă un exemplu negativ de antrenare.

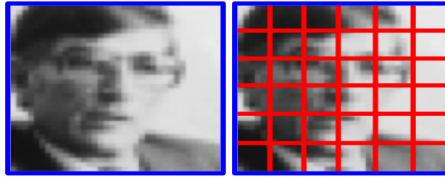


Figura 3: **Descrierea conținutului vizual al unei imagini cu HOG.** (a) Imagine inițială de dimensiuni 36×36 pixeli. (b) Imagine împărțită în celule de dimensiune 6×6 pixeli.

Histograme de gradienti orientați. Veți descrie conținutul vizual al unei ferestre printr-un descriptor local, folosind histograme de gradienti orientați (Figura 3). În acest proiect veți lucra cu un descriptor gata implementat. Descriptorul este implementat de funcția *hog* din biblioteca Sklearn și are forma următoare:

```
descriptors = hog(img, pixels_per_cell=(dim_hog_cell, dim_hog_cell), cells_per_block=(2, 2), feature_vector=False)
```

Funcția primește ca input o imagine, o împarte în celule pătratice de dimensiune *dim_hog_cell* (al doilea argument al funcției) și descrie conținutul fiecărei celule printr-un descriptor de dimensiune 36. Găsiți descrierea funcției *hog* aici <https://scikit-image.org/docs/dev/api/skimimage.feature.html#skimimage.feature.hog>. Pentru exemplele pozitive de dimensiune 36×36 și *dim_hog_cell* = 6 pixeli (Figura 3), descriptorul HOG corespunzător va avea dimensiunile $5 \times 5 \times 36$. Pentru o imagine negativă de dimensiune 100×100 și *dim_hog_cell* = 6 pixeli, descriptorul HOG corespunzător va avea dimensiunile $15 \times 15 \times 36$.

Descriptori pentru exemple pozitive și negative. Antrenarea unui clasificator se realizează de către funcția *train* din clasa *FacialDetector* pe baza descriptorilor pentru exemple pozitive și exemple negative. Obțineți descriptorii pentru exemple pozitive și pentru exemplele negative apelând funcțiile *get_positive_descriptors* respectiv *get_negative_descriptors*. Funcția *train* este scrisă în întregime, în timp ce funcțiile *get_positive_descriptors* și *get_negative_descriptors* trebuie completate. Funcția *get_positive_descriptors* prelucrează imagini de antrenare pozitive de dimensiuni 36×36 de pixeli din directorul **../data/examplePozitive/** ce conține 6713 imagini cu fețe centrate (Figura 2, rândul de sus).

Valorile parametrilor din clasa *Parameters* sunt setate astfel: *dim_window* = 36 de pixeli, *dim_hog_cell* = 6 pixeli, *dim_descriptor_cell* = 36. Astfel, descriptorul unui exemplu pozitiv va avea dimensiunea $(36 / 6 - 1) * (36 / 6 - 1) * 36 = 900$ (număr celule pe verticală \times număr celule pe orizontală \times lungime descriptor celulă). Prin urmare dimensiunile matricei *positive_descriptors* vor fi 6713×900 .

Funcția *get_negative_descriptors* prelucrează imagini de antrenare negative de diverse dimensiuni din directorul **../data/exampleNegative/** ce conține 274 imagini negative (Figura 2, rândul de jos).

Spre deosebire de funcția precedentă, în această funcție trebuie să generați pentru fiecare imagine în parte ferestre aleatoare iar apoi calculați descriptorii (de dimensiune 900)

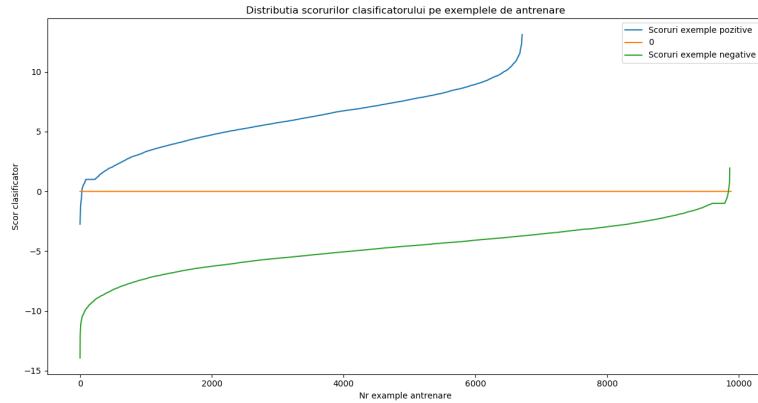


Figura 4: **Vizualizarea scorurilor asignate de clasificatorul învățat exemplelor de antrenare.** Cu verde: scorurile exemplelor negative. Cu albastru: scorurile exemplelor pozitive. Un clasificator perfect ar asigna scoruri > 0 pentru toate exemplele pozitive și scoruri < 0 pentru toate exemplele negative.

pentru aceste ferestre. Numărul total al ferestrelor generate aleator din cele 274 de imagini negative este setat în cod ca fiind egal cu 10000. Prin urmare dimensiunile matricei *get_negative_descriptors* vor fi 10000×900 .

Clasificator liniar. Funcția *train* învață un clasificator liniar sub forma unui vector \mathbf{w} de dimensiune egală cu dimensiunea descriptorului unei ferestre ($= 900$ pentru parametri inițiali) și a unui scalar b astfel încât scorul clasificatorului liniar (\mathbf{w}, b) pentru o fereastră f cu descriptorul asociat \mathbf{d}_f se poate calcula astfel:

$$\text{scor}(f) = \langle \mathbf{w}, \mathbf{d}_f \rangle + b.$$

Ideal, clasificatorul liniar separă perfect descriptorii exemplelor de antrenare pozitive și negative (Figura 4) asignând scoruri pozitive pentru exemplele pozitive și scoruri negative pentru exemplele negative. Clasificatorul liniar perfect ar satisface relația:

$$\text{scor}(f_p) = \langle \mathbf{w}, \mathbf{d}_{f_p} \rangle + b > 0$$

pentru ferestrele f_p conținând exemple pozitive și relația:

$$\text{scor}(f_n) = \langle \mathbf{w}, \mathbf{d}_{f_n} \rangle + b < 0$$

pentru ferestrele f_n conținând exemple negative. În practică, este foarte posibil ca cele două mulțimi de descriptori să nu fie liniar separabile (să nu existe nici un hiperplan dat de (\mathbf{w}, b) cu proprietatea de mai sus). Astfel putem întâlni situația când exemplele pozitive primesc un scor negativ din partea clasificatorului (exemple fals negative) sau când exemplele negative primesc un scor pozitiv din partea clasificatorului (exemple fals

pozitive). Figura 4 arată scorurile asigurate de clasificator exemplelor de antrenare.

Antrenarea cu exemple puternic negative. O modalitate (opțională) de a îmbunătăți performanța clasificatorului liniar este de a rula detectorul facial (vedeți funcția *run* din secțiunea următoare) pe imaginile de antrenare negative și a considera toate detecțiile cu scor > 0 ca exemple adiționale negative (aceste exemple se numesc exemple puternic negative). Toate ferestrele din imaginile de antrenare negative ar trebui să primească din partea clasificatorului un scor negativ. Ferestrele care nu conțin fețe dar au un scor > 0 vor fi asimilate cu detecții fals pozitive.

1.2 Etapa de testare (5 puncte)

Pentru o imagine test, detectarea fețelor se realizează prin glisarea unei ferestre de diferite mărimi de la stânga la dreapta și de sus în jos în imagine și clasificarea fiecărei ferestre pe baza clasificatorului învățat în etapa anterioară. Ferestrele din imagine care au un scor > 0 și sunt maxime locale (nu există altă detecție de scor mai mare care se suprapune cu ea) localizează fețele. O variantă similară cu glisarea unei ferestre de diferite mărimi într-o imagine este glisarea unei ferestre de dimensiuni constante (36×36 pixeli) în imagine și redimensionarea acesteia. Funcția *run* implementează o astfel de soluție.

Variabila *detections* este o matrice (numpy array) care conține pe fiecare linie în formatul [xmin, ymin, xmax, ymax] coordonatele ferestrei care localizează o față. Variabila *scores* reține scorul fiecărei detecții. Variabila *file_names* reține numele imaginii procesate (este nevoie de acest nume în protocolul de evaluare al performanței detectorului facial programat de voi).

Glisarea unei ferestre. Pentru o imagine de dimensiuni $L \times C$ pixeli obțineți mai întâi descriptorul HOG asociat imaginii de dimensiuni $l * c * 36$ folosind funcția *hog*.

Apoi pentru cele l celule orizontale și c celule verticale din imagine, glisați o fereastră care conține $k \times k$ celule ($k = \text{dim_window} / \text{dim_hog_cell} - 1$). În total vor fi $(l - k + 1) * (c - k + 1)$ ferestre de clasificat. Repetați algoritmul redimensionând imaginea pentru a obține un detector care poate localiza fețe de diverse mărimi (Figura 1).

Localizarea corectă a fețelor în imaginile test. Performanța unui detector facial se măsoară prin capacitatea lui de a localiza corect fețele în imagini. O față este localizată corect dacă detecția returnată de detector se suprapune cu mai mult de 30% (cu fereastra dreptunghiulară adnotată în imaginea test). Suprapunerea dintre cele două ferestre f_1 și f_2 se calculează pe baza coordonatelor lor folosind formula $\text{suprapunere}(f_1, f_2) = \text{intersecție}(f_1, f_2) / \text{reuniune}(f_1, f_2)$. Această formulă este calculată în interiorul funcției *non_maximal_suppression*.

Eliminarea non-maximelor. Protocolul de evaluare al performanței detectorului facial penalizează detecțiile care se suprapun foarte mult. Numai o singură detecție (cea care acoperă cel mai mult exemplul adnotat) este considerată corectă, celelalte vor fi considerate detecții fals pozitive. Pentru a obține o performanță mare a detectorului facial este necesar să folosiți funcția *non_maximal_suppression* în interiorul funcției *run*. Funcția

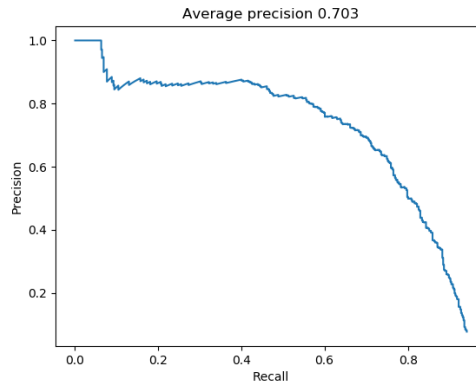


Figura 5: **Grafic precizie-recall.**

non_maximal_suppression va elimina toate detecțiile care se suprapun cu o altă detecție de scor mai mare.

Seturi de date pentru testarea detectorului facial. Pentru testarea detectorului vostru facial veți folosi setul de date *CMU + MIT* ce conține 130 de imagini cu 511 fețe adnotate. Acest set de date se găsește în directorul `'../data/exampleTest/CMU+MIT/'`. De asemenea puteți evalua calitativ performanța detectorului vostru (nu există adnotări, setați `Parameters.has_annotations = False`) pe imaginile realizate la curs din directorul `'../data/exampleTest/CursVA/'`.

Protocolul de evaluare. Cuantificăm performanța unui detector facial de a localiza corect fețe în imagini test prin două valori:

- *precizia*: procentul de detecții returnate de detectorul facial ca fiind ferestre ce conțin fețe. În cazul ideal, detectorul vostru facial are o precizie = 1 = 100%, adică fiecare detecție considerată de detector reprezintă o față.
- *recall* (=rată de detectare): procentul de fețe din imaginile test localizate corect. În cazul ideal detectorul vostru facial are un recall = 1 = 100%, adică localizează corect toate fețele din imagine.

Combinăm cele două valori (precizie + recall) într-un grafic de tip precizie-recall (Figura 5). Fiecare punct de pe acest grafic reprezintă precizia și recall-ul detectorului facial obținute pentru toate detecțiile (ordonate descrescător după scor) care depășesc un anumit scor prag (threshold). Întregul grafic se cuantifică numeric prin *precizia medie* care reprezintă aria de dedesubtul graficului. Funcția *eval_detections* realizează acest grafic și calculează precizia medie. Figura 5 ilustrează graficulul precizie-recall și precizia medie = 0.703 pentru setul de date *CMU + MIT* și un detector facial cu parametri inițiali. Un detector facial perfect ar avea precizia medie = 1, cel mai din dreapta punct de pe graficul funcției aflându-se în acest caz în colțul din dreapta sus.

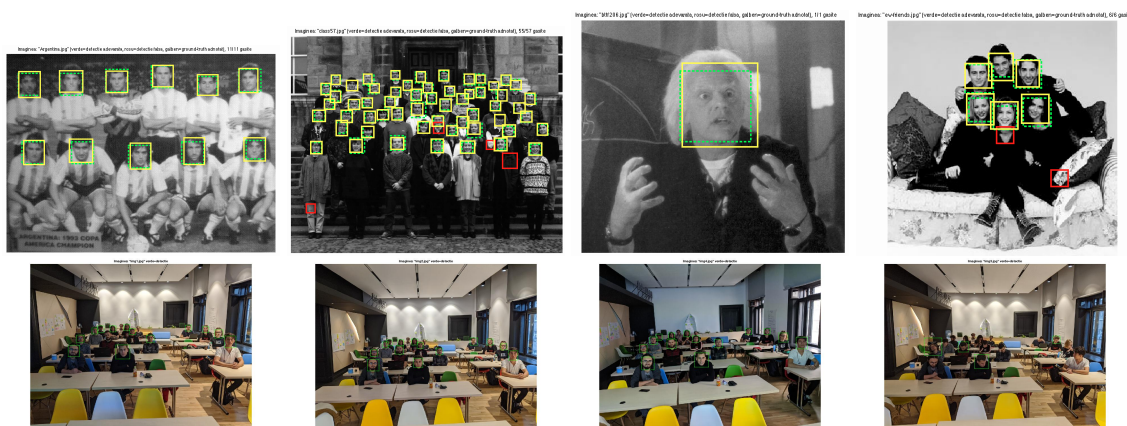


Figura 6: **Exemple de detecții pentru imaginile test.** Primul rând: rezultate pentru imagini adnotate. Cu galben: fețele adnotate. Cu verde: detecțiile adevărat pozitive (localizează corect o față) returnate de către detectorul facial. Cu roșu: detecțiile fals pozitive (nu localizează corect o față) returnate de către detectorul facial. Al doilea rând: rezultate pentru imagini fără adnotare. Cu verde: detecțiile returnate de către detectorul facial.

Vizualizarea rezultatelor. Puteți vizualiza rezultatele detectorului vostru facial folosind funcția `show_detections_with_ground_truth` pentru setul de date *CMU + MIT* sau funcția `show_detections_without_ground_truth` pentru imaginile realizate la curs. Aveți câteva exemple în Figura 6.

Experimente. Odată ce ați implementat funcțiile ce trebuie completate pentru rularea proiectului (acestea sunt în număr de 3: `get_positive_descriptors`, `get_negative_descriptors`, `run` și opțional antrenarea cu exemple puternic negative) puteți realiza experimente cu detectorul vostru facial. În experimentele voastre veți încerca să găsiți cei mai buni parametri astfel încât detectorul vostru să funcționeze cât mai bine. Cuantificăm performanța detectorului vostru facial cu precizia medie pe care o obțineți pe setul de date *CMU + MIT*. Un detector facial foarte bine antrenat ar trebui să atingă o precizie medie de peste 0.90. În realizarea experimentelor voastre încercați diferite valori pentru următorii parametri:

- `dim_hog_cell` (= 6 pixeli implicit);
- `threshold` (= 0 implicit);
- mărirea numărului de exemple pozitive (= 6713 implicit) și negative (= 10000 implicit);
- antrenarea cu exemple puternic negative.

Foarte important: valoarea `Parameters.overlap` = 0.3 care fixează pragul de 0.3 pentru ca o detecție să fie considerată corectă nu trebuie schimbată.

1.3 Predarea proiectului

Veți fi notați cu 4 puncte pentru realizarea etapei de antrenare, cu 3 puncte pentru implementarea glisării ferestrei a detectorului facial și cu alte 2 puncte pentru experimentele pe care le veți face.

Puneți într-o arhivă cu numele *tema5_cod+parametri.zip* codul vostru Python (fără imaginile din *data*), obiectul *params* (de tipul *Parameters*) și modelul SVM. Puneți într-un document cu numele *tema5_rezultate+vizualizari.pdf* următoarele:

- (a) rezultatele diferitelor voastre experimente (minim 3 experimente atașând pentru fiecare experiment graficul cu precizia medie - vedeți Figura 5) precizând de fiecare dată valorile parametrilor folosiți;
- (b) rezultatul cel mai bun (graficul cu precizie medie) obținut comentând valorile parametrilor;
- (c) vizualizări obținute pentru minim 5 imagini din setul de date *CMU + MIT* și minim 2 imagini realizate la curs;

Trimiteți cele două fișiere (*tema5_cod+parametri.zip* și *tema5_rezultate+vizualizari.pdf*) la adresa de email **bogdan.alexe@fmi.unibuc.ro** precizând numele și grupa din care faceți parte.

Termenul limită de predare a proiectului este duminică, 15 decembrie 2019, ora 23:59.