Universitatea Tehnică „Gheorghe Asachi” din Iași

Facultatea de Automatică și Calculatoare

Disciplina Prelucrarea Imaginilor-Proiect

Raport Final- Recunoașterea fețelor folosind algoritmul Eigenfaces

Autori:

Rotariu Daniel – 1307B, Drăgănescu Bianca-Andreea – 1308B

Introducere:

În 1991, Turk și Pentland au sugerat o abordare care folosește reducerea dimensionalității și conceptele de algebră liniară pentru a recunoaște fețele. Această abordare este mai puțin costisitoare din punct de vedere computațional și ușor de implementat și, prin urmare, era utilizată în diverse aplicații la acel moment, cum ar fi recunoașterea scrisului de mână, citirea buzelor, analiza imaginilor medicale etc.

Descriere:

Algoritmul Eigenfaces este folosit pentru detecție și recunoaștere facială și se bazează pe analiza în componente principale. Scopul proiectului este de a implementa și testa acest algoritm.

Analiza în componente principale este o tehnică de reducere a dimensionalității care a fost propusă de Pearson în 1901. Utilizează valori proprii și vectori proprii pentru a reduce dimensionalitatea și pentru a proiecta un eșantion de date pe un spațiu mic de caracteristici.

Decrierea tehnică a soluției implementate:

- Se consideră un set de **m** imagini de dimensiune **N*M**. Imagini care vor fi reprezentate sub forma unor matrice de tip uchar, de dimensiune **N*M**.

```
//reprezentam fetele sub forma unei matrice
uchar** faces_array = new uchar * [NR_FACES];
uint length = faces_img[0].total() * faces_img[0].channels();

for (int i = 0; i < NR_FACES; i++)
    faces_array[i] = faces_img[i].isContinuous() ? faces_img[i].data : faces_img[i].clone().data;
```

- Se va calcula fața medie, pe care o vom scădea din fiecare față.

```
//functie pentru calculul fetei medii
uchar* getMeanArray(uchar** faces_array, int length)
{
    uchar* mean_array = new uchar[length];

    for (int i = 0; i < length; i++) {
        int sum = 0;

        for (int j = 0; j < NR_FACES; j++)
            sum += faces_array[j][i];

        mean_array[i] = sum / NR_FACES;
    }

    return mean_array;
}
```

Acest lucru va fi realizat cu ajutorul funcției getMeanArray.

```
//calculam fata medie
uchar* mean_array = getMeanArray(faces_array, length);
```

- Se va calcula matricea ce conține fețele – fața medie, cu ajutorul funcției getMatrix.

```
//functie pentru calculul matricei ce contine fetele - fata medie
double** getMatrix(uchar** faces_array, uchar* mean_array, int rows, int cols)
{
    double** matrix = new double* [rows];

    for (int i = 0; i < rows; i++)
        matrix[i] = new double[cols];

    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            matrix[i][j] = faces_array[j][i] - mean_array[i];

    return matrix;
}
```

```
//calculam matricea ce contine fetele - fata medie
double** matrix = getMatrix(faces_array, mean_array, length, NR_FACES);
```

Matricea calculată va fi convertită la o matrice din librăria Eigen pentru a putea fi folosită la operațiile ce urmează, cu ajutorul funcției ConvertToEigenMatrix, care la rândul ei, folosește funcții din librăria Eigen.

```
//functie pentru a converti un tablou bidimensional la o matrice din libraria Eigen
MatrixXd ConvertToEigenMatrix(double** data, int rows, int cols)
{
    MatrixXd eMatrix(rows, cols);

    for (int i = 0; i < rows; ++i)
        eMatrix.row(i) = VectorXd::Map(&data[i][0], cols);

    return eMatrix;
}
```

- Se va calcula matricea de covarianță, înmulțind matricea calculată anterior (matricea ce conține fețele – fața medie) transpusă cu aceeași matrice în forma ei normală. A fost aleasă această variantă deoarece este mai eficienta din punct de vedere computațional. Se va obține o matrice de covarianță de dimensiune **m*m** ce are **m** vectori proprii de dimensiune **m**.

```
//calculam matricea de covarianta
MatrixXd covMatrix = eMatrix.transpose() * eMatrix;
```

- Pasul următor este cel de a calcula valorile proprii și vectorii proprii. Pentru a face asta, se vor folosi funcții din librăria Eigen.

```
//calculam vectorii si valorile proprii
EigenSolver<MatrixXd> s(covMatrix);

MatrixXd D = s.pseudoEigenvalueMatrix();
MatrixXd V = s.pseudoEigenvectors();
```

unde D = matrice ce conține pe diagonala principal valorile proprii și V = matricea ce conține vectorii proprii pe coloane.

- Se vor folosi vectorii proprii de la pasul anterior pentru a reprezenta fiecare vector corespunzător fețelor normalizate (fața – fața medie) prin combinații liniare.Mai intai sortam vectorii proprii in functie de valorile proprii corespunzatoare.

```
double* eigenvalues = new double[NR_FACES];
double* eigenvalues_sorted = new double[NR_FACES];
for (int i = 0; i < NR_FACES; i++) {
    eigenvalues[i] = D.coeff(i, i);
    eigenvalues_sorted[i] = D.coeff(i, i);
}

selectionSort(eigenvalues_sorted);

for (int i = 0; i < NR_FACES; i++)
    for (int j = 0; j < NR_FACES; j++)
        if (eigenvalues_sorted[i] == eigenvalues[j])
            D.col(i) = V.col(j);
```

- Apoi mapăm vectorii in matricea C’ folosind relația ui = A * vi, unde A este matricea ce conține fețele normalizate. Normalizăm vectorii pentru a aduce valorile in intervalul [0,255].

```
//mapam vectorii proprii in matricea C' folosind relatia ui = A * vi
MatrixXd U = eMatrix * D;

//normalizam vectorii proprii
U = normalize(U, length, 255);
```

```
//functie pentru normalizarea matricei ce contine vectorii proprii
MatrixXd normalize(MatrixXd V2, int length, int val)
{
    for (int i = 0; i < NR_FACES; i++) {

        double min = minEigenVector(V2.col(i), length);
        double max = maxEigenVector(V2.col(i), length);

        for (int j = 0; j < length; j++) {

            V2.col(i)[j] = (V2.col(i)[j] - min) / (max - min) * val;

            if (V2.col(i)[j] < 0)
                V2.col(i)[j] = 0;

            if (V2.col(i)[j] > 255)
                V2.col(i)[j] = 255;
        }
    }

    return V2;
}
```

- Se vor extrage vectorii proprii într-o matrice de tip uchar, iar mai apoi îi vom reprezenta sub forma unor obiecte de tip Mat pentru a putea fi afișate ca imagini.

```
//reprezentam vectorii proprii sub forma unor obiecte de tip Mat
Mat eigenfaces[NR_FACES];

for (int i = 0; i < NR_FACES; i++)
    eigenfaces[i] = Mat(faces_img[0].rows, faces_img[0].cols, faces_img[0].type(), eigenvectors[i]);
```

- Selectam primii K vectori proprii corespunzatori celor mai mari K valori proprii.

```
    //selectam doar primii K vectori proprii
    MatrixXd U_K = U.block(0, 0, length, K);
```

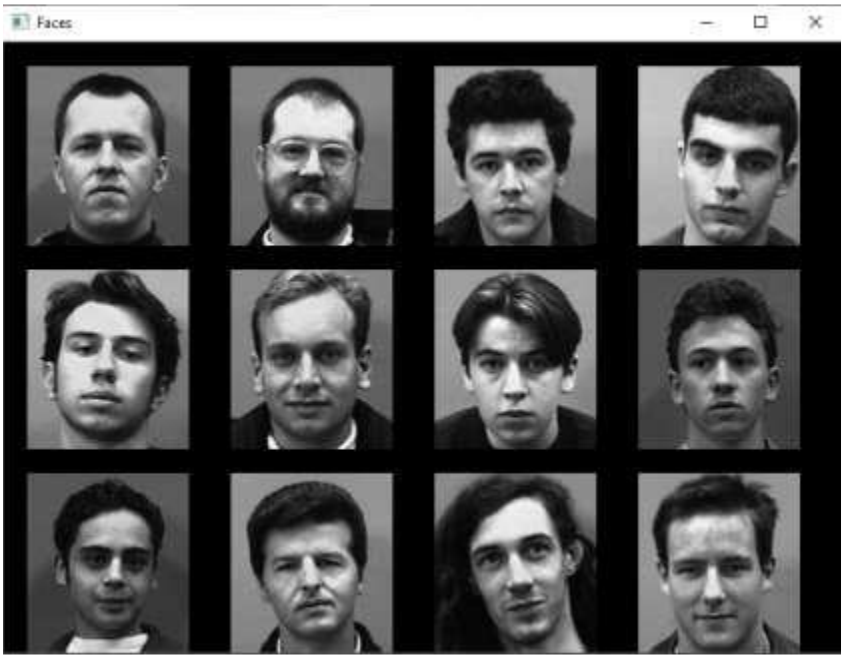
- Pentru a reprezenta fețele normalizate sub forma unor combinațiilor liniare, se va rezolva un sistem de ecuații.

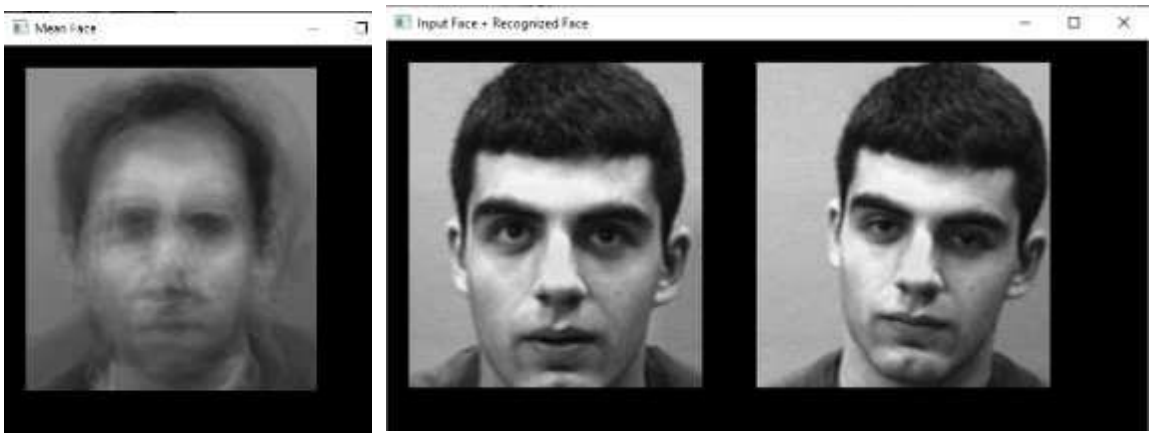
```
 // reprezentam fetele initiale prin combinatii liniare ale vectorilor proprii
 MatrixXd coeff_faces(K, NR_FACES);
 for (int i = 0; i < NR_FACES; i++)
     coeff_faces.col(i) = U_K.colPivHouseholderQr().solve(eMatrix.col(i));
```

- Pentru a test acest algoritm, se va lua o imagine cu o persoană oarecare, care se află printre cele din setul de date de intrare, asupra căreia se va aplica același algoritm.
- In final, calculam distanta euclidiana minima dintre vectorul ce contine coeficientii combinatiei liniare corespunzatoare imaginii date si vectorii ce contin coeficientii combinatiilor liniare corespunzatoare fețelor din setul inițial.

```
// citim o alta imagine ce nu se afla in setul initial cu o persoana ce se afla in set si calculam coeficientii
Mat input_face = imread("Images/person4/face2.jpg", IMREAD_GRAYSCALE);

// aplicam acelasi algoritm
uchar** input_array = new uchar *;
input_array[0] = input_face.isContinuous() ? input_face.data : input_face.clone().data;

double** input_matrix = getMatrix(input_array, mean_array, length, 1);
MatrixXd input_eMatrix = ConvertToEigenMatrix(input_matrix, length, 1);

MatrixXd coeff_input(K, 1);
coeff_input.col(0) = U_K.colPivHouseholderQr().solve(input_eMatrix.col(0));

// gasim distanta euclidiana minima intre vectorul ce contine coeficientii fetei citite
// si vectorii ce contin coeficientii fetelor din setul initiale
Index min_index;
(coeff_faces.colwise() - coeff_input.col(0)).colwise().squaredNorm().minCoeff(&min_index);

Mat output_face = faces_img[min_index];
```

Rezultate experimentale:





Concluzii:

- Aplicația a fost dezvoltată folosind un set de date de intrare strict (imagini asemănătoare, imagini de aceeași dimensiune, fețe centrate), astfel încât ajungem la performanțe destul de mari (peste 80%) de a recunoaște o persoană dintr-o imagine dată.
- În practica, acest algoritm nu este la fel de eficient în condiții de lumina diferită sau în cazul în care fața din imagine este poziționată sau dimensionată diferit.
- Un alt dezavantaj al acestui algoritm este reprezentat de faptul că necesita mult timp pentru calculul vectorilor proprii.

https://github.com/DanielRotariu0/Eigenfaces

Referințe:

https://en.wikipedia.org/wiki/Eigenface

http://www.scholarpedia.org/article/Eigenfaces

https://learnopencv.com/eigenface-using-opencv-c-python/

http://www.vision.jhu.edu/teaching/vision08/Handouts/case_study_pca1.pdf

https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/