Laporan Tugas Besar 2 IF 2123 Aljabar Linier dan Geometri

Aplikasi Nilai Eigen dan EigenFace pada Pengenalan Wajah (Face Recognition)

Kelompok 15 (Reigenface)

Muhammad Farrel Danendra Rachim - 13521048

Fakih Anugerah Pratama - 13521091

Reza Pahlevi Ubaidillah - 13521165

Bab 1

Deskripsi Masalah

Pengenalan wajah (Face Recognition) adalah teknologi biometrik yang bisa dipakai untuk mengidentifikasi wajah seseorang untuk berbagai kepentingan khususnya keamanan. Program pengenalan wajah melibatkan kumpulan citra wajah yang sudah disimpan pada database lalu berdasarkan kumpulan citra wajah tersebut, program dapat mempelajari bentuk wajah yang dikenali oleh program, lalu mencocokkan antara kumpulan citra wajah yang sudah dipelajari dengan citra yang akan diidentifikasi.

Untuk menyelesaikan persoalan ini, kami akan menggunakan metode Eigenface dalam program ini. Sekumpulan citra wajah akan digunakan dengan representasi matriks. Dari representasi matriks tersebut akan dihitung sebuah matriks Eigenface. Program pengenalan wajah dapat dibagi menjadi 2 tahap berbeda yaitu tahap training dan pencocokkan. Pada tahap training, akan diberikan kumpulan data set berupa citra wajah. Citra wajah tersebut akan dinormalisasi dari RGB ke Grayscale (matriks), hasil normalisasi akan digunakan dalam perhitungan eigenface. Seperti namanya, matriks eigenface menggunakan eigenvector dalam pembentukannya. Setelah memperoleh eigenface yang merepresentasikan seluruh database wajah, akan diperoleh matriks bobot citra wajah database yang disimpan saat pelatihan. Untuk pencocokan, setelah diperoleh matriks bobot citra wajah yang diinput, akan dibandingkan jarak Euclidean antara kedua matriks bobot. Gambar dengan jarak Euclidean paling kecil menjadi gambar yang dinilai paling cocok dengan gambar yang diinput.

Kami juga menggunakan GUI sebagai sarana menginput gambar dan meng-output gambar dari database yang paling cocok. Berikut ini adalah input yang akan dimasukkan pengguna untuk eksekusi program:

a. Folder dataset, berisi folder atau directory yang berisi kumpulan gambar yang digunakan sebagai training image.
b. File gambar, berisi file gambar input yang ingin dikenali dengan format file yang bebas selama merupakan format untuk gambar.

Bab 2

Teori Singkat

1. Perkalian Matriks

Terdapat dua jenis perkalian matriks: Perkalian skalar dan perkalian dua matriks. Perkalian skalar merupakan perkalian matriks dengan sebuah bilangan lain. Misal bilangan tersebut adalah k, jika dikalikan dengan matriks A = $[a_{ij}]$, kA = $[k(a_{ij})]$

Perkalian dua matriks merupakan suatu operasi perkalian antara dua matriks yang akan menghasilkan sebuah matriks baru. Kedua matriks dapat dikalikan baik dengan dimensi baris dan kolom yang sama, maupun berbeda dengan syarat jika matriks A dikalikan matriks B, jumlah kolom A harus sama dengan jumlah baris B. Misalkan dimensi matriks A adalah m x r (m baris dan r kolom), dimensi matriks B adalah r x n, sehingga dalam C = AB, diperoleh dimensi matriks C m x n. ($A_{m \times r} \times B_{r \times n} = C_{m \times n}$)

Misalkan A = $[a_{ij}]$ dan B = $[b_{ij}]$, maka C = $[c_{ij}] = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}$. Elemen c_{ij} diperoleh dengan cara mengalikan elemen baris ke-i di matriks A dengan elemen kolom ke-j di matriks B dan menjumlahkan semua hasil perkalian ini.

Contoh:

$$A = \begin{bmatrix} 3 & 7 \\ 4 & 9 \end{bmatrix} and\ B = \begin{bmatrix} 6 & 2 \\ 5 & 8 \end{bmatrix}$$

$AB_{11} = 3 \times 6 + 7 \times 5 = 53$
$AB_{12} = 3 \times 2 + 7 \times 8 = 62$
$AB_{21} = 4 \times 6 + 9 \times 5 = 69$
$AB_{22} = 4 \times 2 + 9 \times 8 = 80$

Sehingga, $AB = \begin{bmatrix} 53 & 62 \\ 69 & 80 \end{bmatrix}$.

Jika AB terdefinisi, maka BA belum tentu terdefinisi (kecuali jika matriks A dan B merupakan matriks persegi). Jika keduanya terdefinisi, AB ≠ BA (tidak komutatif). Sifat asosiatif berlaku, yaitu (AB)C = A(BC). Sifat distributif juga berlaku, yaitu (B + C)A = BA + CA dan A(B + C) = AB + AC. Misal I merupakan matriks identitas, maka IA = AI = A.

2. Nilai eigen dan vektor eigen

Jika A adalah matriks n x n maka vektor tidak-nol x di R^n disebut **vektor eigen** dari A jika Ax sama dengan perkalian sebuah skalar λ dengan x: Ax = λx. Skalar λ disebut **nilai eigen** dari A, dan x adalah vektor eigen yang berkoresponden dengan λ. Nilai eigen menyatakan nilai karakteristik dari sebuah matriks berukuran n x n. Vektor eigen x adalah sebuah matriks kolom yang jika dikalikan dengan matriks n x n akan menghasilkan vektor lain yang merupakan vektor itu sendiri (menyebabkan vektor x menyusut atau memanjang dengan arah yang sama jika λ positif atau arah yang berkebalikan jika λ negatif). Tidak semua matriks memiliki nilai eigen.

Untuk mencari nilai eigen dari sebuah matriks n x n, pertama-tama terapkan persamaan det(λI - A) = det(A - λI) = 0 kepada matriks tersebut, di mana I = matriks identitas.
Contoh:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

jika diterapkan persamaan di atas akan diperoleh:

$$\det(A - \lambda I) = \begin{vmatrix} 2-\lambda & -1 \\ -1 & 2-\lambda \end{vmatrix} = \lambda^2 - 4\lambda + 3 = 0.$$

Diperoleh $\lambda_1 = 1$ dan $\lambda_2 = 3$ yang membuat matriks A - λI singular.

Untuk mendapatkan vektor eigen x dari matriks A di atas, pecahkan (A - λI)x = 0.

$\lambda_1 = 1$: (A - I)x = $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

akan menghasilkan $\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\lambda_2 = 3$: (A - 3I)x = $\begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

akan menghasilkan $\boldsymbol{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

Sebuah matriks A berukuran n x n memiliki balikan jika dan hanya jika λ = 0 bukan nilai eigen dari matriks A.

3. Eigenface

Eigenface adalah sebuah algoritma yang berguna untuk memilih vektor terbaik dari database untuk menyebarkan citra wajah ke dalam ruang wajah yang ada. Sebuah eigenface merupakan kumpulan dari berbagai vektor eigen yang diturunkan dari matriks kovarian hasil generasi citra wajah yang dilatihkan. Gambar sebuah eigenface terlihat berwarna grayscale dan *blurry* (sering dipanggil sebagai *ghostly images*). Pada umumnya, eigenface diekstrak dari data citra dengan menggunakan Principal Component Analysis (PCA) sehingga dimensi direduksi. Eigenface berguna untuk mewakilkan data input secara efisien - tiap wajah individu dapat direpresentasikan sebagai sebuah kombinasi linear dari eigenface.

Untuk mencari eigenface, pertama-tama, dari matriks citra hasil linierisasi training image: $S = (\Gamma 1, \Gamma 2, \ldots, \Gamma M)$ (1) dicari matriks rata-ratanya dengan rumus:

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n \quad (2)$$

Lalu cari selisih: $\phi i = \Gamma i - \Psi$ (3)

Dan cari matriks kovarian C: $C = \frac{1}{M}\sum_{n=1}^{M}\phi_n\phi_n^T = AA^T$ (4)

$L = A^T A \qquad L = \phi_m^T \phi_n$

sebelum mencari eigenvalue dan eigenvector: $C \text{ x } vi = \lambda i \text{ x } vi$ (5)

Akhirnya, eigenface dapat diperoleh: $\mu_i = \sum_{k=1}^{M} v_{ik}\phi_L$ (6)

Untuk pengenalan wajah baru, terapkan cara pada tahap pertama perhitungan eigenface untuk mendapatkan nilai eigen image tersebut.

$$\mu new = v \text{ x } \Gamma new - \Psi \text{ (7)}$$

$\Omega = \mu 1 , \mu 2 , \ldots , \mu M$

Akhirnya, gunakan metode euclidean distance untuk mencari jarak antara nilai eigen training image database dengan nilai eigen dari image testface. Gambar dengan jarak euclidean paling kecil merupakan gambar yang paling menyerupai testface. Jika nilai minimum di atas nilai batas tidak terdapat citra wajah yang mirip dengan testface.

Bab 3

Implementasi Program

1. **Penggunaan Library**
   Kami memanfaatkan beberapa library untuk membantu implementasi program ini. Berikut beberapa library utama yang kami gunakan:
   - tkinter: Pembuatan GUI
   - PIL: Memproses gambar yang diinput ke program
   - OpenCV: Digunakan untuk face recognition dan mengolah gambar
   - Sklearn.preprocessing: fungsi normalize yang digunakan untuk menormalisasi vector
   - numpy, sympy, math: Membantu perhitungan dan operasi matriks
2. **Training & matching w/ eigenface**
   Flow program terbagi menjadi dua tahap utama: Training dan matching. Pada tahap pertama, yaitu training, akan diimplementasikan Principal Component Analysis (PCA) terhadap nilai masukan yang merupakan nilai warna pada gambar yang dibaca secara hitam-putih.

```
def train(self, files : np.ndarray, files_path : str) -> None :
    print('training started ...')
    startTime = time.time()

    imgCount = len(files)
    desiredSize = self.getDesiredSize()

    # make image files squared and centered
    # files = np.array([util.makeImageSquare(i) for i in files])

    # reformat files into usable images
    images = np.array([util.resize(i, desiredSize).flatten() for i in files]) # (x, 256^2)
```

   Pada bagian training, pertama diperoleh matriks yang merepresentasikan foto-foto pada dataset dengan ukuran (dimensi) 256 x 256. Matriks itu kemudian dilinierisasi (di-flatten) menjadi sebuah vektor, dengan kata lain tiap elemen membentuk sebuah vektor dengan ukuran 1 x 256^2. Sehingga jika terdapat n buah foto dalam dataset, matriks baru yang terbentuk (dalam snippet program di atas dilambangkan dengan variabel images) berukuran n x 256^2.

```
mean = np.mean([k for k in images], axis=0) # axis = 0 since we avg EVERY corresponding pixel


imagesDiff = np.array([(images[i]-mean) for i in range(imgCount)])
```

Lalu, dicari kumpulan mean dari matriks images, sehingga diperoleh matriks imagesDiff yang merupakan selisih dari matriks images dan mean, sesuai rumus berikut:

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n \quad (2) \qquad \phi i = \Gamma i - \Psi \quad (3)$$

```
# A
A = np.array([i for i in imagesDiff]).transpose() # sesuai definisi A di file


# L = C'
L =  A.transpose() @ A
```

Kemudian, akan dicari matriks A, yaitu matriks images yang ditranspos, dan matriks L (kovarian), yaitu hasil perkalian antara A^T dan A.

```
# compute eigenvalues and eigenvector of L
eigValL, eigVecL = getEigenVectors(L, getEigenValues(L))


# compute eigenvalues and eigenvector of C
eigVecC = A @ eigVecL

# normalize eigenface
eigVecC = util.normalizeSQR(eigVecC)
```

Setelah itu, dicarilah nilai eigen dan vektor eigen dari L dengan algoritma dekomposisi Schur, serta vektor eigen C yang diperoleh dengan mengalikan matriks A dengan vektor eigen dari L. Vektor eigen lalu dinormalisasi, yaitu tiap vektor eigen panjangnya harus sebesar 1.

```python
def qr_decomposititon_gs(mat: List[List[int]]) -> tuple[List[List[int]], List[List[int]]]:
    """Dekomposisi QR dengan algoritma Gram-Schmidt Orthogonalization
    Too slow, still. O(2mn²).

    Must try Schwarz-Rutishauser Algorithm O(mn²)"""
    mat = np.array(mat)
    length = len(mat)

    e = np.empty((length,length))
    a = mat.T
    for i in range(length):
        u = np.copy(a[i])
        for j in range(i):
            u = u - (a[i] @ e[j]) * e[j]
        e[i] = u/np.linalg.norm(u)

    R = np.zeros((length,length))
    for i in range(length):
        for j in range(i, length):
            R[i][j] = a[j]@e[i]
    Q = e.T

    return (Q,R)
```

Pencarian eigenvector menggunakan dekomposisi QR dengan metode algoritma Gram-Shmidt Orthogonalization. Algoritma ini meng-ortonormalkan sebuah himpunan vektor.

```python
# compute weights
# W = np.array([[eigVecC.transpose()[i] @ imagesDiff[j] for i in range(imgCount)] for j in range(imgCount)])
W = np.array([[np.array(eigVecC.transpose()[i]) @ np.array(imagesDiff[j].transpose()) for i in range(imgCount)] for j in range(imgCount)])
# print('shap', np.array(W).shape, np.array(eigVecC.transpose()[0]).shape, np.array(imagesDiff[0].transpose()).shape)


# omega
# Omega = [W[i] @ eigVecC.transpose() for i in range(imgCount)] # somehow result nya kinda unsatisfying?
Omega = np.array([i for i in W])
```

Diperoleh matriks W, yaitu matriks bobot dari hasil kali eigen vektor C yang telah dinormalisasi dan ditranspos dan matriks imagesDiff. Matriks W berisi distribusi berat untuk masing-masing Eigen Vector terhadap gambar mean. Sedangkan vektor omega berisi kumpulan elemen W dalam sebuah foto yang sama.

```python
print('solving started ...')
startTime = time.time()

# new_files = np.array([util.makeImageSquare(i) for i in new_files]) # to make sure it is a square image

desiredSize = self.getDesiredSize()
mean = self.mean
imgCount = self.trainImgCount
eigVecC = self.eigVec

newImgCount = len(new_files)

newImages = np.array([util.resize(i, desiredSize).flatten() for i in new_files])
newImagesDiff = np.array([(newImages[i]-mean) for i in range(newImgCount)])
```

Pada tahap kedua, yaitu matching, diperoleh matriks dari sebuah foto baru, dengan mean dan matriks selisihnya, dengan proses awal yang mirip dengan tahap training.

```
# kalkulasi pada targets
W_target = np.array([[np.array(eigVecc.transpose()[i]) @ np.array(newImagesDiff[j].transpose()) for i in range(imgCount)] for j in range(newImgCount)])


# omega
Omega_target = np.array([i for i in W_target])
```

Matriks W_target diperoleh dengan mengalikan eigenvector C yang berkorespondensi dari tahap training dengan matriks selisih newImagesDiff, sehingga vektor Omega_target mengandung elemen dari W_target.

```
def getEuclidDistance(om1, om2 = []): # om is a member list of omega
    sum = 0
    for i in range(len(om1)):
        if om2 != []:
            sum += (om1[i] - om2[i])**2
        else:
            sum += om1[i]**2

    return sum
```

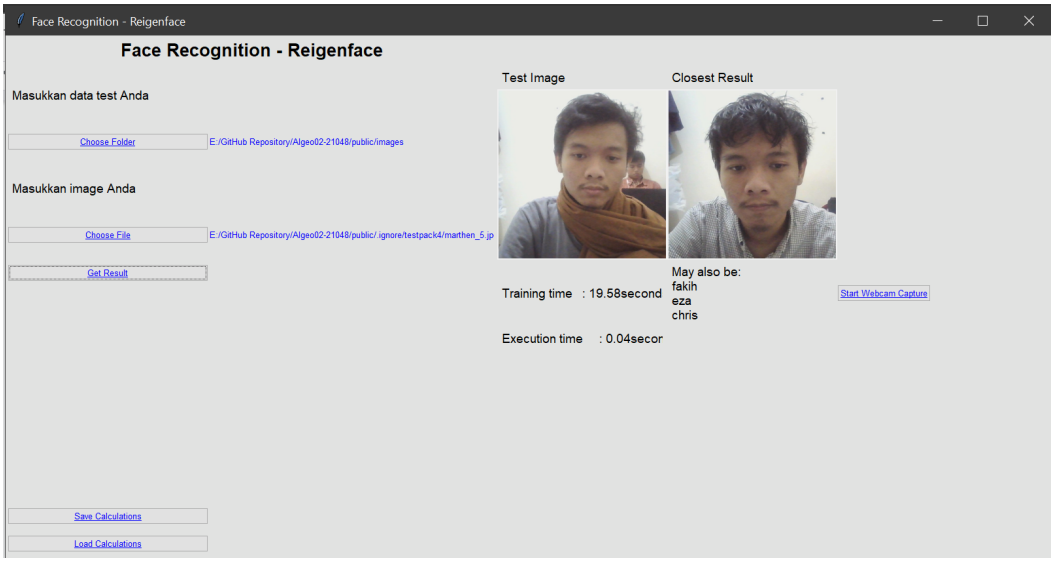
```
for i in range(self.targetImgCount):
    print('\n\nHasil pencocokan ke-' + str(i+1))
    j = 0
    minIdx = 0

    min = util.getEuclidDistance(self.targetDistributedWeight[i], self.distributedWeight[j])
    result = []
    while j < self.trainImgCount:

        if util.getEuclidDistance(self.targetDistributedWeight[i], self.distributedWeight[j]) < min:
            min = util.getEuclidDistance(self.targetDistributedWeight[i], self.distributedWeight[j])
            minIdx = j
        result.append(util.getEuclidDistance(self.targetDistributedWeight[i], self.distributedWeight[j]))
        j += 1
```

Akan dicari jarak euclidean terkecil untuk setiap elemen yang bersesuaian pada vektor Omega dengan Omega_target. Elemen yang memiliki jarak Euclidean terkecil menjadi foto dari dataset yang paling sesuai dengan foto target.

3. GUI



Di atas merupakan tampilan GUI face recognition kelompok kami. Untuk memasukkan data test klik tombol “select folder”, dan directory folder akan ditampilkan di sebelah tombol tersebut. Untuk memasukkan test image klik tombol “select file”, dan directory file akan ditampilkan di sebelah tombol tersebut, serta test image akan ditampilkan di bawah teks “Test Image”. Kami membuat semua ukuran foto kami, baik test image maupun data set, 256 x 256 pixel untuk memudahkan perbandingan antarfoto. Meskipun begitu, program masih dapat melakukan analisis pada gambar yang berukuran berbeda, namun dengan syarat berbentuk persegi. Gambar dari data test yang paling sesuai berdasarkan algoritma eigenface akan ditampilkan di bawah teks “Closest Result”, serta akan ditampilkan execution time untuk mencocokkan gambar yang sesuai. Pengguna juga bisa melakukan webcam test dengan menekan tombol “Start Webcam Capture” lalu menekan “Stop Webcam Capture” untuk mengambil gambar. Setelah itu, pengguna menekan tombol “Get Result” untuk melakukan eksekusi program, dan hasil capture akan dicocokkan dengan folder data test.

Selain itu, pengguna dapat menggunakan tombol “Save Calculations” dan “Load Calculations” untuk menyimpan dan membaca data hasil kalkulasi folder yang telah dihitung nilai eigennya dan menyimpannya dalam file berformat “.json” yang dapat dibaca program meskipun program telah ditutup. Dengan catatan, program ini akan melakukan kalkulasi dan PCA tepat setelah pengguna memilih folder yang akan dijadikan sebagai data test.

```
# Frames
root.grid_rowconfigure(0, weight=1)

mainframe = Frame(root, width=1600, height=800, style='Mainframe.TFrame')
mainframe.grid_rowconfigure(0, weight=1)
mainframe.grid_rowconfigure(1, weight=7)
mainframe.columnconfigure(0, weight= 2)
mainframe.columnconfigure(1, weight=1)

mainframe.pack(expand=True, fill=BOTH)

topframe = Frame(mainframe, style='Frameling.TFrame')
topframe.grid(row=0, column=0, sticky='nsew')

uploadphotoFrame = Frame(mainframe, width=300, style='Frameling.TFrame')
# uploadphotoFrame.grid_rowconfigure(0, weight=1, )
uploadphotoFrame.grid(row=1, column=0, padx=3, pady=3, sticky='nsew')

conversionFrame = Frame(mainframe, width=1200, style='Frameling.TFrame')
conversionFrame.grid(row=1, column=1, sticky='nsew')

# Top Frame Section
titleLabel = Label(topframe, text="Face Recognition - Reigenface", style='Title.TLabel')
titleLabel.place(relx=0.5,rely=0.5,anchor=CENTER)
titleLabel.grid_columnconfigure(0, weight=1)

# Upload Photo & Database section
uploadphotoFrame.grid_rowconfigure(0, weight=3)
uploaddatatestLabel = Label(uploadphotoFrame, text="Masukkan data test Anda", style= 'SubTitle.TLabel')
uploaddatatestLabel.grid(row=0, column=0, sticky= 'WE')
```

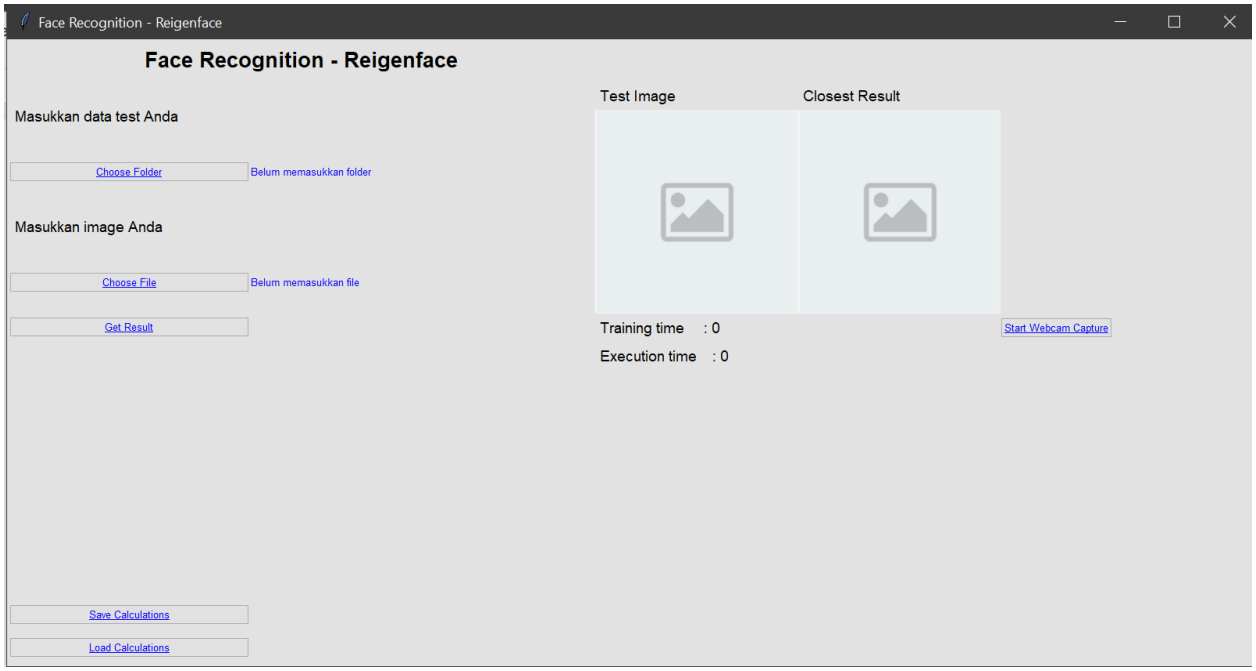
Berikut adalah cuplikan dari program GUI kami. Kami menggunakan library tkinter sebagai library utama kami dalam membangun GUI ini, khususnya untuk membuat text, frame, tombol, dan lain-lain. Digunakan juga library lain seperti PIL untuk memproses gambar yang akan ditampilkan di GUI.

Bab 4

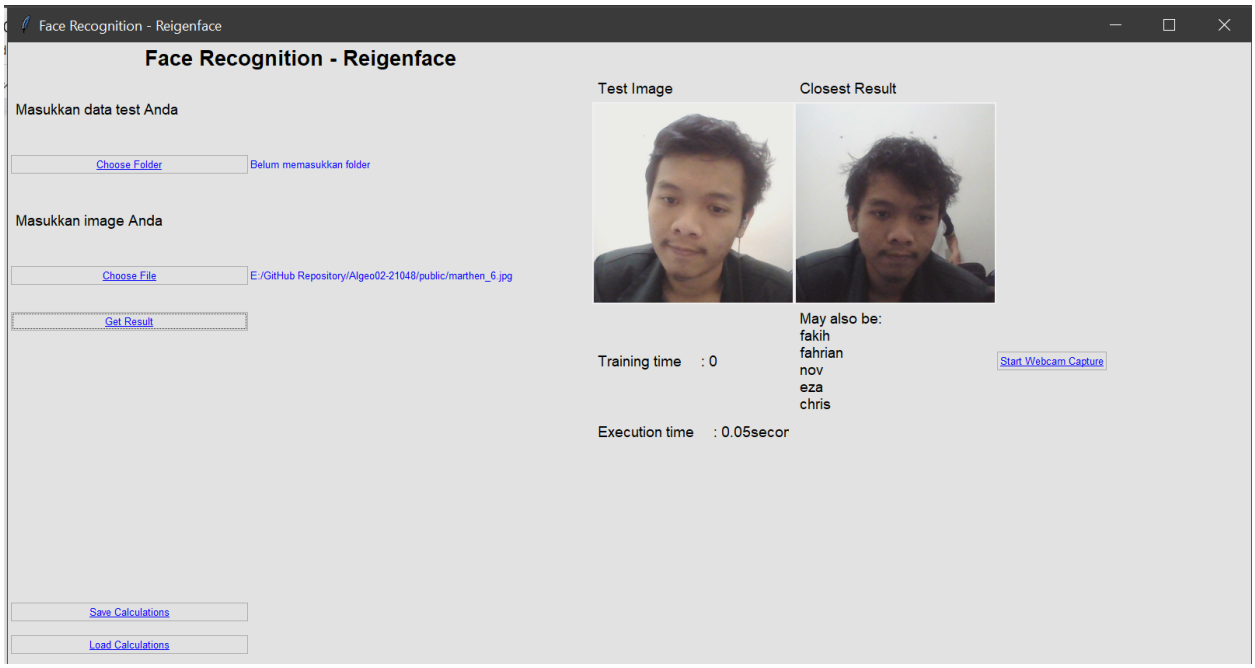
Eksperimen

1. User Interface Aplikasi

User akan disambut dengan tampilan pada Gambar 4.1.1 ketika membuka aplikasi. User dapat melakukan data training dengan menekan Choose Folder dan memilih image yang akan dilakukan face recognition dengan menekan Choose File.

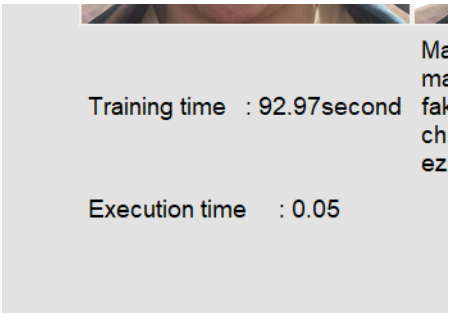


Gambar 4.1.1 Tampilan awal aplikasi



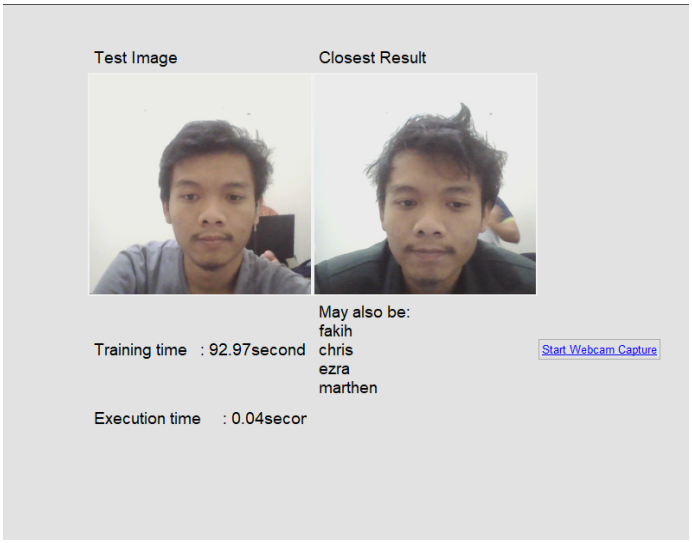
Gambar 4.1.2 Tampilan aplikasi saat menunjukkan hasil

2. Hasil Face Recognition

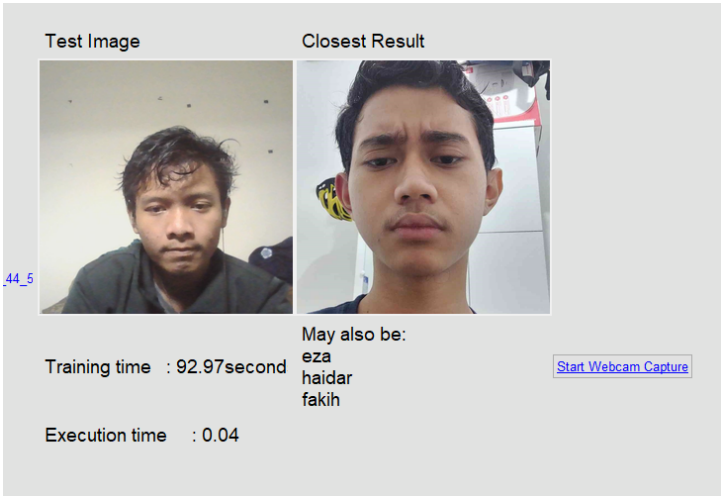


Gambar 4.2.1 Tampilan waktu eksekusi aplikasi

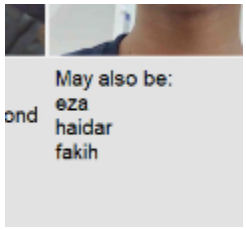
Dapat kita lihat bahwa aplikasi akan otomatis menulis lama waktu yang dibutuhkan untuk menemukan gambar yang memiliki kemiripan paling besar dengan gambar masukan. Waktu eksekusi yang dibutuhkan oleh aplikasi ini tidak akan banyak berubah antara satu proses eksekusi satu dengan yang lain karena proses yang dilakukan hanyalah membandingkan satu gambar dengan M banyak gambar *training*.
Sedangkan, waktu yang dibutuhkan untuk melakukan *training* akan mengikuti banyak image yang diperhitungkan dalam kalkulasi.



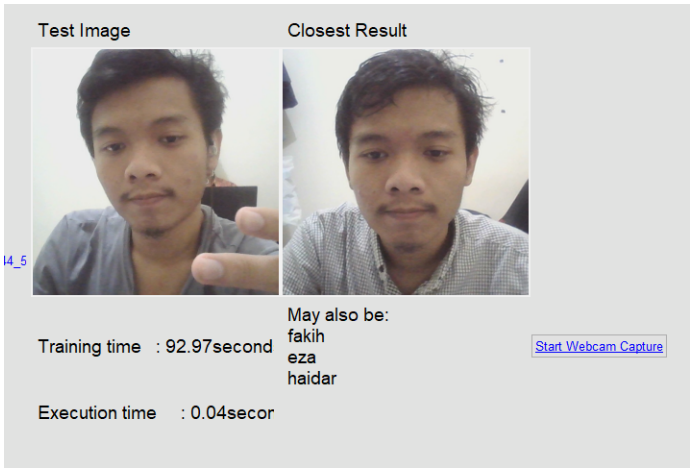
Meskipun model test menggunakan pakaian dan penataan rambut yang sedikit berbeda, program masih dapat mengenali dengan baik model dan menampilkan hasil yang paling dekat.



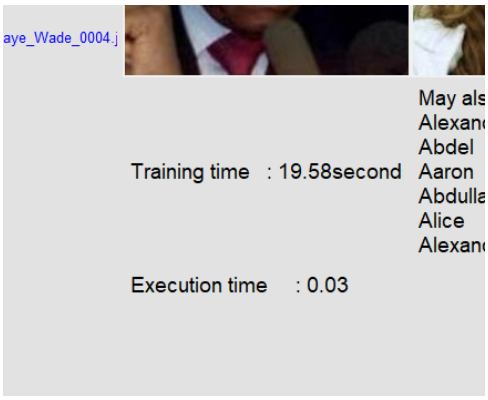
Sedangkan, jika gambar masukan memiliki perbedaan pencahayaan dan posisi yang mencolok, program akan sulit mengenali jawaban yang tepat dan memberikan hasil paling dekat menurut perhitungannya.



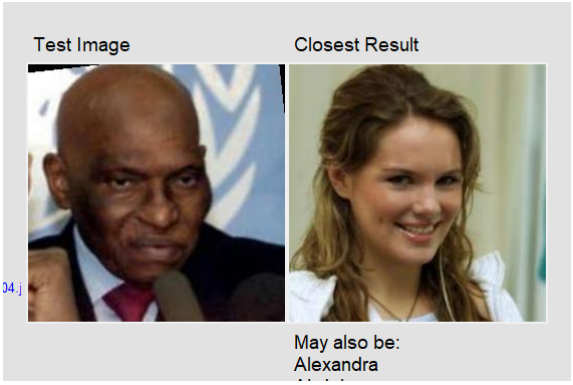
Meskipun begitu, program tetap dapat mengenali kemungkinan jawaban lain yang dalam kasus ini benar.



Melihat alternatif jawaban pada gambar di atas dan gambar sebelumnya, dapat disimpulkan bahwa menurut program ini, model fakih, eza, dan haidar memiliki beberapa kemiripan.



Menggunakan dataset yang memiliki jumlah gambar yang berbeda, waktu yang dibutuhkan program untuk melakukan analisis (training) jauh berbeda. Dalam kasus ini, beberapa faktor penyebabnya adalah training image yang memiliki dimensi asli jauh lebih kecil dan jumlah gambar yang lebih sedikit. Dapat dilihat juga execution time sedikit berkurang karena alasan yang sama.



Dalam kasus ini, program dapat mengenali pose atau posisi wajah model dengan baik. Namun, program gagal mengenali hasil jawaban yang paling tepat untuk masukan yang diberikan. Faktor penyebabnya adalah jumlah contoh gambar training yang dibatasi hanya 2 untuk setiap model dalam kasus ini. Hal ini menunjukkan bahwa dengan semakin banyak dan jelas gambar training yang diberikan, program akan semakin baik dalam mengenali model yang bersesuaian dengan masukan.

Bab 5

Kesimpulan, Saran, dan Refleksi

A. Kesimpulan

Dari tugas besar ini, kami lebih memahami sebuah mekanisme salah satu penerapan dari ilmu nilai dan vektor eigen ini, yakni mengimplementasikannya menjadi eigenface yang dapat digunakan untuk face recognition. Kami juga lebih mengenal berbagai fungsionalitas masing-masing library dan framework Python seperti OpenCV dan Tkinter. Kami juga belajar untuk mengimplementasikan pencarian eigenvalue dan eigenvector melalui algoritma programming.

Hasil program ini dapat menampilkan wajah dari database yang paling mirip dengan wajah input. Dengan menggunakan library OpenCV, kami dapat mengekstraksi citra wajah yang akan ditemukan eigenface-nya, yang kemudian akan dibandingkan dengan wajah input, dan wajah yang paling mirip akan ditampilkan ke GUI.

B. Saran

Link database di kaggle.com yang dilampirkan di spek tugas besar sebagai data test hasilnya sering tidak akurat, sehingga program face recognition tidak mampu berjalan secara sempurna. Namun selain itu, spesifikasi yang diberikan sudah jelas dan membantu berjalannya pembuatan tugas besar ini.

C. Refleksi

Dalam melakukan tugas besar ini, kami mengalami berbagai kendala seperti mencari nilai dan vektor eigen tanpa library. Waktu pemrosesan akibatnya lebih lama dari yang diharapkan. Selain itu, kami juga merasa kesulitan mencari layout GUI yang menarik serta proporsional demi kenyamanan pengguna aplikasi. Namun, kami dapat menyelesaikan tugas ini karena komunikasi dan koordinasi terhadap antar anggota kelompok yang berjalan dengan lancar. Kami juga memperoleh berbagai ilmu baru mengenai beragam fungsi library dalam bahasa Python dan gunanya, serta cara mencari eigenface dalam metode algoritmik.

Daftar Referensi

“Nilai Eigen dan Vektor Eigen Bagian 1” by Rinaldi Munir
https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf

“Face Recognition Using Eigenfaces (PCA Algorithm)
https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/

“Pengenalan Wajah Menggunakan Algoritma Eigenface dan Euclidean Distance"
https://www.neliti.com/publications/135138/pengenalan-wajah-menggunakan-algoritmaeigenface-dan-euclidean-distance

“Python GUI Programming With Tkinter”
https://realpython.com/python-gui-tkinter/

“Create UI in Python-Tkinter”
https://www.tutorialsteacher.com/python/create-gui-using-tkinter-python

“How to build a GUI with PyQt”
https://blog.logrocket.com/how-to-build-gui-pyqt/

“Feature extraction and similar image search with OpenCV for newbies”
https://medium.com/machine-learning-world/feature-extraction-and-similar-imagesearch-with-opencv-for-newbies-3c59796bf774

Face Detection in Python Using a Webcam
https://realpython.com/face-detection-in-python-using-a-webcam/

“Can QR Decomposition Be Actually Faster? Schwarz-Rutishauser Algorithm”
https://www.codeproject.com/Articles/5319754/Can-QR-Decomposition-Be-Actually-Faster-Schwarz-Ru

“Computing Eigenvalues and Eigenvectors using QR Decomposition”
https://www.andreinc.net/2021/01/25/computing-eigenvalues-and-eigenvectors-using-qr-decomposition

Lampiran

Link repository github:
https://github.com/BreezyDR/Algeo02-21048

Link video YouTube:
https://youtu.be/wEA-dNmnPbs