

IF2123 - Aljabar Linier dan Geometri

Laporan Tugas Besar 2

Pengenalan Wajah (*Face Recognition*) Menggunakan Eigenface



Disusun oleh:

Nama	NIM
Melvin Kent Jonathan	13521052
Juan Christopher Santoso	13521116
Kandida Edgina Gunawan	13521155

Institut Teknologi Bandung
Sekolah Teknik Elektro dan Informatika
Tahun Ajaran 2022/2023

Daftar Isi

IF2123 - Aljabar Linier dan Geometri	1
Daftar Isi	2
Daftar Gambar	2
Bab 1	3
Bab 2	4
2.1. Perkalian Matriks	4
2.2. Nilai Eigen	5
2.3. Vektor Eigen	5
2.4. Eigenface	5
Bab 3	8
3.1. Library yang digunakan	8
3.2. Fungsi yang dipakai	9
3.3. Algoritma yang Digunakan	14
Bab 4	22
4.1. Data Set	22
4.2. Tampilan Page	22
4.3. Hasil Eksperimen	23
4.4. Analisis	28
Bab 5	31
5.1 Kesimpulan	31
5.2 Saran	31
5.3 Refleksi	31
Daftar Referensi	32
Lampiran	34

Daftar Gambar

Gambar 3.1.1.	16
Gambar 3.1.2.	17
Gambar 3.2.1.	17
Gambar 3.2.2.	18
Gambar 3.2.3.	18
Gambar 3.2.4.	19
Gambar 3.2.5.	19
Gambar 3.2.6.	20
Gambar 3.2.7.	21
Gambar 4.1.1.	22
Gambar 4.2.1.	23
Gambar 4.2.2.	23
Gambar 4.3.1.1.	24
Gambar 4.3.1.2.	24
Gambar 4.3.1.3.	24
Gambar 4.3.1.4.	25
Gambar 4.3.2.1.	25
Gambar 4.3.2.2.	25
Gambar 4.3.2.3.	26
Gambar 4.3.2.4.	26
Gambar 4.3.3.1.	26
Gambar 4.3.3.2.	27
Gambar 4.3.3.3.	27
Gambar 4.3.3.4.	27
Gambar 4.3.3.5.	28

Bab 1

Deskripsi Masalah

Buatlah program pengenalan wajah dalam Bahasa Python berbasis GUI dengan spesifikasi sebagai berikut:

1. Program menerima input folder dataset dan sebuah gambar citra wajah.
2. Basis data wajah dapat diunduh secara mandiri melalui
<https://www.kaggle.com/datasets/herveisburak/pins-face-recognition> .
3. Program menampilkan gambar citra wajah yang dipilih oleh pengguna.
4. Program melakukan pencocokan wajah dengan koleksi wajah yang ada di folder yang telah dipilih. Metrik untuk pengukuran kemiripan menggunakan eigenface + jarak euclidean.
5. Program menampilkan 1 hasil pencocokan pada dataset yang paling dekat dengan gambar input atau memberikan pesan jika tidak didapatkan hasil yang sesuai.
6. Program menghitung jarak euclidean dan nilai eigen & vektor eigen yang ditulis sendiri. Tidak boleh menggunakan fungsi yang sudah tersedia di dalam library atau Bahasa Python.

Bonus:

Bagian A (Kamera)

1. Terdapat fitur kamera yang dapat mengenali wajah secara realtime menggunakan webcam ketika program dijalankan.
2. Teknis pengenalan wajah melalui kamera dibebaskan oleh pembuat program. (contoh: program dieksekusi setiap 10 detik sekali).
3. Fitur kamera merupakan fitur tambahan, fitur utama upload gambar melalui GUI tetap
4. harus ada.

Bagian B (Video)

1. Video penjelasan algoritma dan aplikasi program yang diunggah ke youtube.
2. Video dibuat sekreatif mungkin dengan target untuk mensosialisasikan ilmu yang kalian sudah pelajari dan terapkan pada program ini. Bukan hanya video mengenai penggunaan aplikasi.

Bab 2

Teori Singkat

2.1. Perkalian Matriks

Perkalian Matriks merupakan operasi terhadap satu atau lebih matriks dengan batasan dan aturan tertentu sehingga dapat menghasilkan matriks yang mewakili nilai perkalian matriks yang dioperasikan. Adapun perkalian matriks dapat dibagi menjadi dua, yakni perkalian skalar matriks dan perkalian antarmatriks.

2.1.1. Perkalian Skalar Matriks

Perkalian skalar matriks merupakan perkalian suatu matriks A dengan konstanta. Perkalian ini akan menghasilkan sebuah matriks B yang berdimensi sama dengan matriks A dengan setiap elemen matriks B merupakan hasil perkalian konstanta dengan elemen matriks A yang berkoresponden dengan elemen B tertuju. Ilustrasi perkalian tersebut dapat dilihat pada gambar di bawah ini.

$$k = \text{skalar} ; A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$k \cdot A = k \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$= \begin{bmatrix} k \cdot a & k \cdot b \\ k \cdot c & k \cdot d \end{bmatrix}$$

2.1.2. Perkalian Antarmatriks

Perkalian antarmatriks merupakan perkalian dua atau lebih matriks. Perkalian dua matriks memiliki batasan, yakni jumlah kolom matriks A haruslah sama dengan jumlah baris matriks B. Ilustrasi perkalian tersebut dapat dilihat pada gambar di bawah ini.

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \begin{pmatrix} p & q \\ r & s \end{pmatrix} = \begin{pmatrix} ap + br & aq + bs \\ cp + dr & cq + ds \\ ep + fr & eq + fs \end{pmatrix}$$

2.2. Nilai Eigen

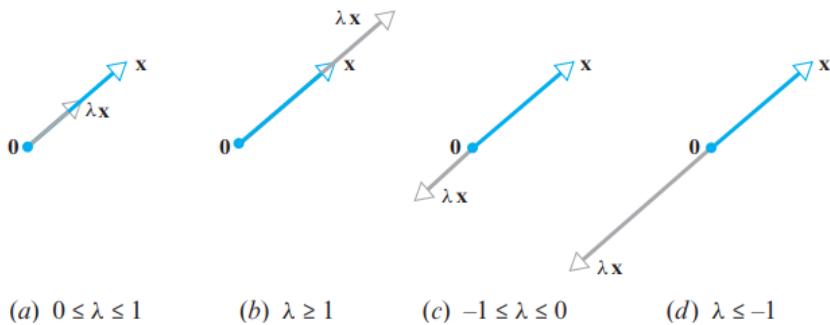
Nilai Eigen merupakan suatu nilai skalar yang menyatakan nilai karakteristik dari sebuah matriks yang berukuran $n \times n$. Kata “eigen” sendiri berasal dari bahasa Jerman *eigen* yang berarti “asli” atau “karakteristik”. Apabila A ialah matriks berukuran $n \times n$, maka terdapat vektor tidak nol x di \mathbb{R}^n yang disebut vektor Eigen dari A jika persamaan berikut dipenuhi:

$$Ax = \lambda x$$

Skalar λ dalam persamaan tersebut merupakan nilai Eigen dari A yang berkoresponden dengan λ dan akan memenuhi persamaan $\det(\lambda I - A) = 0$.

2.3. Vektor Eigen

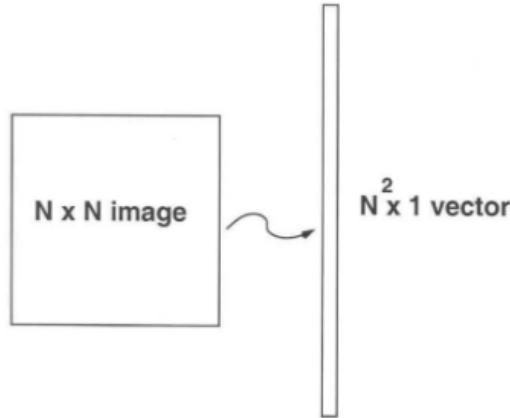
Vektor Eigen x merupakan vektor kolom bukan nol yang bila dikalikan dengan suatu matriks berukuran $n \times n$ akan menghasilkan vektor lain yang merupakan kelipatan vektor itu sendiri. Operasi $Ax = \lambda x$ akan menyebabkan vektor x memanjang ataupun menyusut sesuai dengan faktor λ .



2.4. Eigenface

Eigenface merupakan seperangkat vektor Eigen yang dipakai oleh operasi komputasional untuk pengenalan wajah atau *face recognition*. Set Eigenface dapat diperoleh dengan melakukan *principal component analysis* (PCA) pada sekumpulan gambar wajah manusia yang berbeda. PCA adalah sebuah teknik reduksi dimensionalitas yang menggunakan nilai Eigen dan vektor Eigen untuk mereduksi dimensionalitas dan memproyeksi *training sample/data* pada ruang fitur kecil. Untuk membentuk berbagai wajah manusia, hanya dibutuhkan beberapa eigenface dengan komposisi yang berbeda. Misalnya, sebuah wajah manusia dapat dikomposisi menjadi wajah rata-rata ditambah 45% eigenface 1, 20% eigenface 2, dan 4% eigenface 3. Adapun algoritme trainingnya adalah sebagai berikut:

1. Siapkan sekumpulan gambar wajah (*training images*) berjumlah m dengan dimensi $N \times N$
2. Konversikan kumpulan gambar tersebut menjadi sebuah vektor dengan ukuran N^2



$$x_1, x_2, x_3, \dots, x_m$$

3. Kalkulasikan rata-rata dari seluruh vektor wajah yang didapat

$$\psi = \frac{1}{m} \sum_{i=1}^m x_i$$

4. Cari selisih antara setiap vektor wajah dengan nilai rata-rata

$$a_i = x_i - \psi$$

5. Hitung matriks kovarian dengan cara mengalikan matriks A dengan A^T . A memiliki dimensi $N^2 \times M$.

$$A = [a_1 \quad a_2 \quad a_3 \quad \dots \quad a_m]$$

$$Cov = A^T A$$

6. Hitung nilai Eigen dan vektor Eigen dari matriks kovarian dengan rumus berikut:

$$A^T A \nu_i = \lambda_i \nu_i$$

$$A A^T A \nu_i = \lambda_i A \nu_i$$

$$C' u_i = \lambda_i u_i$$

di mana

$$u_i = A \nu_i$$

Tahapan pengenalan wajah adalah sebagai berikut:

1. Sebuah image wajah baru atau test face (Γ_{new}) akan dicoba untuk dikenali, pertama terapkan cara pada tahapan pertama perhitungan Eigenface untuk mendapatkan nilai eigen dari image tersebut.

$$\mu_{new} = v \times \Gamma_{new} - \Psi$$

$$\Omega = \mu_1, \mu_2, \dots, \mu_M$$

2. Gunakan metode euclidean distance untuk mencari jarak (distance) terpendek antara nilai eigen dari training image dalam database dengan nilai Eigen dari image testface.

$$\varepsilon_k = \Omega - \Omega_k$$

Pada tahapan akhir, akan ditemui gambar dengan euclidean distance paling kecil maka gambar tersebut yang dikenali oleh program paling menyerupai test face selama nilai kemiripan di bawah suatu nilai batas. Jika nilai minimum di atas nilai batas maka dapat dikatakan tidak terdapat citra wajah yang mirip dengan test face.

Bab 3

Implementasi Program

Implementasi program meliputi penjelasan tech stack/kakas yang digunakan dan garis besar algoritma kompresi yang diimplementasikan. Pada implementasi program Eigen Face, kami menggunakan *tech stack* Python untuk membuat bagian frontend dan backend. Adapun berikut ini merupakan *library* yang kami gunakan untuk membuat program.

3.1. Library yang digunakan

1. Library Frontend
 - a. tkinter

Library tkinter merupakan library standar Python untuk membuat graphic user interface (GUI). Tkinter menyediakan 14 widget dasar, yakni button, canvas, checkbutton, entry, frame, label, listbox, menubutton, message, radiobutton, scale, scollar, text, dan toplevel. Kami menggunakan tkinter untuk membuat GUI interaktif dari program ini.

- b. PIL

Python Imaging Library atau PIL merupakan library Python yang memampukan program untuk memproses gambar, baik itu membuka, memanipulasi, maupun menyimpan gambar dengan berbagai format. Pada program ini, kami menggunakan library PIL untuk menampilkan gambar pada GUI yang dibaca dari folder lokal serta melakukan *resize* gambar.

- c. time

Modul time berfungsi untuk berbagai variasi manipulasi atau pengelolaan nilai waktu. Pada program ini, kami menggunakan modul time untuk menampilkan waktu yang dibutuhkan program untuk melakukan pencocokan wajah antara input dengan training dataset.

2. Library Backend

- a. numpy

Library numpy adalah library Python yang digunakan untuk bekerja dengan array. Library ini juga banyak digunakan untuk menyelesaikan persoalan aljabar linear , transformasi fourier, dan matriks. Pada program ini, numpy banyak kami gunakan dalam operasi matrix seperti penjumlahan matrix, pengurangan matrix, perkalian antara skalar dan matrix, perkalian antara matrix dan matrix, pembagian matrix dengan skalar, serta transpose matrix.

- b. cv2

Cv2 merupakan nama modul import untuk library OpenCV-Python. Library OpenCV adalah library yang sangat berguna

dalam pemrosesan citra, seperti analisis video ataupun analisis gambar. Library OpenCV pada program ini memegang peran yang penting dalam mengekstraksi features dari images.

c. os

Library os adalah library Python yang banyak menyediakan fungsi untuk membuat dan menghapus direktori, mengambil isi dari suatu direktori, mengubah dan mengidentifikasi direktori saat ini, dan masih banyak lagi operasi lain yang berkaitan dengan direktori. Dalam program ini, library os digunakan untuk menggabungkan satu atau lebih path serta mengakses semua nama file yang terdapat pada suatu path yang spesifik.

d. math

Modul math merupakan modul bawaan Python yang berisi fungsi-fungsi untuk berbagai perhitungan matematika. Pada program ini, kami menggunakan fungsi sqrt dari modul math untuk menghitung *normalized value* dari vektor weight agar dapat dikomparasikan dengan sesamanya untuk mencari *euclidean distance* terkecil.

3.2. Fungsi yang dipakai

1. `backend_proto.py`

Tabel 3.2.1 Daftar fungsi yang digunakan pada file `backend_proto.py`

No	Nama Fungsi	Hasil Return	Parameter	Deskripsi
1	extract_features	Sebuah array yang berisi feature-feature dari image yang telah diekstraksi	Image_path, vector_size	Melakukan ekstraksi fitur gambar yang telah diubah menjadi grayscale
2	batch_extractor	Sebuah dictionary of array yang isinya merupakan feature-feature dari semua images di dalam folder dataset	images_path	Melakukan loop pada file-file di dalam folder dataset untuk mengekstrak feature-feature dari tiap file
3	mean	Array yang berisi nilai rataan feature-feature	extraction_result	Menghitung rataan nilai feature-feature

		untuk semua image di dalam dataset		image
4	A	Matrix hasil normalisasi dengan mengurangkan hasil ekstraksi dengan rataan feature	Extraction_result, mean	Mengurangi tiap hasil ekstraksi image dengan rataan feature
5	kovarian	Matrix kovarian dari matrix yang diinputkan user	A	Mengalikan matrix A dengan matrix A^T untuk mendapatkan matrix kovarian
6	norm	Norma dari sebuah vector x	x	Menghitung norma dari sebuah vektor
7	proj	Hasil proyeksi u terhadap v	Vektor u dan v	Menghitung hasil proyeksi u terhadap v
8	qr, qr2	Matrix Q dan R yaitu faktorisasi dari matrix yang diinputkan	matrix	Melakukan dekomposisi terhadap matrix yang diinputkan menjadi hasil perkalian 2 matrix Q dan R (matrix segitiga atas)
9	find_eig	Nilai-nilai eigen dan eigen vector dari matrix yang diinputkan	matrix	Mencari nilai Eigen dan eigen vector dari matrix dengan metode <i>simultaneous power iteration</i>
10	sorted_eig	Nilai-nilai eigen dan eigen vector dari matrix yang	matrix	Mengurutkan vektor eigen berdasarkan

		telah diurutkan dari yang terbesar ke yang terkecil		urutan terbesar ke terkecil eigen valuenya
11	getEigenFaces	Vektor-vektor eigenface terbaik sejumlah k dengan dimensionalitas original (sudah tidak tereduksi)	eigenSpace, A, k	Mengalikan matriks A dengan matriks eigenSpace (kumpulan vektor eigen dari matriks kovarian) yang telah diseleksi sejumlah k.
12	getWeightSet	Vektor-vektor yang mewakili <i>weight</i> atau kombinasi linier dari vektor-vektor eigenface terpilih untuk memperoleh vektor foto yang sudah dinormalisasi terhadap rataan feature.	A, eigenFaces, M	Vektor-vektor tersebut berkorespondensi dengan matriks A.
13	getThreshold	Toleransi euclidean distance terbesar yang diterima	eigenfaceWeight, M	Mencari selisih (euclidean distance) terbesar dalam dataset dan mengalikannya dengan 0,5 untuk mendapatkan threshold
14	matcher	matchedPath (foto dengan kemiripan tertinggi) serta euclidean distance-nya	Input, datasetName, mean, weightSet, M, threshold, eigenfaces	Mengomparasikan vektor weight input dengan kumpulan vektor weight dari dataset dan

		terhadap foto input		mencatat gambar dengan euclidean distance terkecil terhadap vektor weight input
--	--	---------------------	--	---

2. main.py

1) WelcomePage()

Fungsi yang merangkum komponen-komponen yang ingin ditampilkan pada WelcomePage (halaman pembuka program). Komponen-komponen yang ingin ditampilkan meliputi label, gambar, dan juga button.

2) MainPage()

Fungsi yang merangkum komponen-komponen yang ingin ditampilkan di pada MainPage (halaman utama program). Komponen-komponen yang ingin ditampilkan meliputi button start, button filedialog, label-label, serta images.

3) inputFolder()

Fungsi inputFolder() adalah subfungsi yang terdapat di dalam MainPage(). Kegunaan dari fungsi inputFolder() ini adalah memberikan command pada button dataSetBtn untuk memunculkan filedialog yang akan menerima masukan berupa path directory dari folder dataset yang diinputkan.

4) inputFileImg()

Fungsi inputFileImg() adalah susbfungsi yang terdapat di dalam MainPage(). Kegunaan dari fungsi inputFileImg() ini adalah memberikan command pada button userImageBtn untuk memunculkan filedialog yang akan menerima masukan berupa path dari file image yang diinputkan.

5) inputImageByCam()

Fungsi inputImageByCam() adalah subfungsi yang terdapat di dalam MainPage(). Kegunaan dari fungsi inputImageByCam() ini adalah memberikan command pada button camBtn untuk menangkap gambar dari kamera, menyimpan gambar tersebut, serta menyimpan path dari gambar yang telah ditangkap dari kamera tersebut.

6) faceRecognition()

Fungsi faceRecognition() adalah subfungsi yang terdapat di MainPage(). Kegunaan dari fungsi faceRecognition() ini adalah menggabungkan fungsi-fungsi yang telah diimplementasikan sebelumnya di file backend_proto.py yang kemudian dipadukan untuk mencari serta

menampilkan gambar pada dataset yang menyerupai image file yang telah diinputkan user.

3. cam.py

1) CamPage()

Fungsi CamPage() adalah fungsi yang digunakan sebagai penyelesaian permasalahan Bonus A yaitu menggunakan webcam untuk mendeteksi wajah *realtime*. Fungsi ini meliputi membuka *webcam user*, mendeteksi wajah dari *user*, hingga menyimpan hasil pendekstian tersebut ke sebuah folder tertentu.

4. submain.py

1) readImage(path)

Fungsi readImage(path) adalah fungsi pembantu main untuk membaca sebuah *image* yang ingin dibaca oleh *user*. Fungsi ini membutuhkan sebuah input yaitu *path* (atau letak atau lintasan) dari *image* yang ingin dibaca. Fungsi readImage lalu akan membaca *image* yang tertera pada *path* tersebut lalu mengembalikan dalam bentuk *image* yang telah dibaca menggunakan *library ImageTk* dan *PhotoImage*.

2) on_enter(e)

Prosedur on_enter(e) adalah fungsi yang bergerak apabila cursor dari mouse *user* memasuki sebuah object. Dalam hal ini, prosedur ini dipakai sebagai *hover change color* pada button *Insert Image* dan *Insert Dataset*.

3) on_leave(e)

Prosedur on_leave(e) adalah fungsi yang bergerak apabila cursor dari mouse *user* meninggalkan sebuah object. Dalam hal ini, prosedur ini dipakai sebagai *hover change color* pada button *Insert Image* dan *Insert Dataset*.

4) on_start_enter(e)

Seperti halnya prosedur on_enter(e), prosedur ini digunakan sebagai *hover change color* pada button *Start Program*.

5) on_start_leave(e)

Seperti halnya prosedur on_leave(e), prosedur ini digunakan sebagai *hover change color* pada button *Start Program*.

6) on_cam_enter(e)

Seperti halnya prosedur on_enter(e), prosedur ini digunakan sebagai *hover change color* pada button *Input by Camera*.

7) on_cam_leave(e)

Seperti halnya prosedur on_leave(e), prosedur ini digunakan sebagai *hover change color* pada button *Input by Camera*.

8) secondsToTime(secs)

Fungsi ini digunakan untuk mengubah *execution time* dalam bentuk detik (*type integer*) menjadi bentuk string dengan format <jam>:<menit>:<detik>.

3.3. Algoritma yang Digunakan

1. Eigenvalue dan Eigenvector

Algoritma/metode yang kami gunakan untuk mencari Eigen Value dan Eigen Vector adalah menggunakan QR Decomposition yang dipadukan dengan simultaneous power iteration. Metode dekomposisi QR adalah metode dekomposisi matrix A yang berukuran mxn menjadi hasil perkalian dari matrix Q dengan matrix segitiga atas R.

$$\begin{aligned}A &= QR \\R &= Q^T A \\Q^T Q &= QQ^T = I\end{aligned}$$

Terdapat 3 algoritma QR Decomposition yang terkenal, yaitu Gram-Schmidt Orthogonalization, Householder Reflections, dan Given Rotations. Setiap algoritma mempunyai kelebihan dan kekurangannya masing-masing. Householder Reflection menjadi algoritma yang sangat kompleks ketika digunakan pada matriks kompleks. Di sisi lain algoritma Given Rotation jika digunakan, presisinya akan berkurang pada iterasi terakhir dan algoritma ini tidak efisien jika diterapkan pada matriks berukuran besar. Gram-Schmidth Orthogonalization adalah metode yang paling sederhana untuk diimplementasikan jika dibandingkan dengan 2 algoritma sebelumnya. Namun, kelemahan dari algoritma ini adalah adanya *numerical instability* dan besarnya kompleksitas algoritma ketika algoritma ini diterapkan pada matriks berukuran besar. Selain itu, terdapat pula Schwarz-Rutishauser Algorithm yang merupakan hasil modifikasi dari Gram-Schmidth Orthogonalization. Ide utama dari algoritma Schwarz-Rutishauser ini adalah kompleksitas perhitungan basis ortogonal dari matrix A dapat dikurangi secara drastis jika dibandingkan dengan Gram-Schmidth Process dengan kompleksitas $O(2mn^2)$ dan Householder Reflections dengan kompleksitas $O(2mn^2 - 0.6n^3)$, Schwarz-Rutishauser hanya memiliki kompleksitas sebesar $O(mn^2)$.

Algoritma QR Decomposition di atas menggunakan dan mengimplementasikan *Schwarz-Rutishauser Algorithm*. Pada metode Gram-Schmidt biasa, dicari matrix orthogonal Q dari A berdasarkan projeksi orthogonal tiap vektor a_k elemen dari A dengan $k = 1, 2, \dots, n$ ke rentang vector q_k elemen dari $Q(k)$, $i = 1, 2, 3, \dots, k$ yang telah diketahui. Tiap vector ortogonal q_k

yang baru, dihitung sebagai penjumlahan dari projeksi, yang kemudian dikurangi dengan a_k yang sesuai.

$$proj_{\vec{q}} \vec{a} = \frac{\langle \vec{q}, \vec{a} \rangle}{\langle \vec{q}, \vec{q} \rangle} * \vec{q} = \frac{\langle \vec{q}, \vec{a} \rangle}{\|\vec{q}\|^2} * \vec{q}$$

$$\vec{q}_k = \vec{a}_k - \sum_{i=1}^k proj_{\vec{q}_i} \vec{a}_k,$$

$$\vec{q}_k = \frac{\vec{q}_k}{\|\vec{q}_k\|}$$

$$\vec{q}_1 = \vec{a}_1,$$

$$\vec{q}_2 = \vec{a}_2 - proj_{\vec{q}_1} \vec{a}_2,$$

$$\vec{q}_3 = \vec{a}_3 - proj_{\vec{q}_1} \vec{a}_3 - proj_{\vec{q}_2} \vec{a}_3, |$$

$$\vec{q}_4 = \vec{a}_4 - proj_{\vec{q}_1} \vec{a}_4 - proj_{\vec{q}_2} \vec{a}_4 - proj_{\vec{q}_3} \vec{a}_4$$

Sedangkan pada Schwarz-Rutishauser Algorithm, baris kedua dari persamaan tersebut dapat disimplifikasi menjadi sebagai berikut

$$\vec{q}_k = \vec{a}_k - \sum_{i=1}^k \langle \vec{q}_i^T, \vec{a}_k \rangle * \vec{q}_i$$

Karena R adalah inner product dari Q^T dan A, elemen ke i dari tiap kolom vector r_k dapat dinyatakan sebagai berikut:

$$\vec{e}_k r_{i,k} = \langle \vec{q}_i^T, \vec{a}_k \rangle \vec{e}_k$$

Dengan mensubstitusikan persamaan di atas ke persamaan sebelumnya, didapatkan

$$\vec{q}_k = \vec{a}_k - \sum_{i=1}^k r_{i,k} * \vec{q}_i$$

Implementasi dari metode Schwarz-Rutishauser Algorithm dalam bahasa Python, dapat dilihat sebagai berikut:

```

def qr2(A):
    A = np.array(A, dtype=type)
    n = len(A)
    Q = np.array(A, dtype=type)
    R = np.zeros((n, n), dtype=type)
    for i in range(n):
        for j in range(i):
            R[j, i] = np.transpose(Q[:, j]).dot(Q[:, i])
            Q[:, i] = Q[:, i] - R[j, i] * Q[:, j]
        norm = 0
        for k in range(n):
            norm += (Q[k, i] ** 2)
        norm = norm ** (1/2)
        R[i, i] = norm
        Q[:, i] = Q[:, i] / R[i, i]
    return Q, R

```

Gambar 3.1.1. Algoritma QR Decomposition

Selanjutnya, untuk mencari Eigen Values dan Eigen Vectors dari suatu matriks, digunakan metode *simultaneous power iteration*. A adalah sebuah matrix real yang ingin kita cari nilai eigennya dan pada langkah ke- k , kita menghitung QR Decomposition

$$Ak = Qk Rk$$

Dimana Qk adalah matriks orthogonal dan Rk adalah matrix segitiga atas. Kemudian, kita menghitung $A_{k+1} = R_k Q_k$.

Ingat bahwa:

$$A_{k+1} = R_k Q_k = Q_k^{-1} Q_k R_k Q_k = Q_k^{-1} A_k Q_k = Q_k^T A_k Q_k$$

Semua A_k similar dengan demikian semua A_k memiliki nilai eigen yang sama. Setelah melalui beberapa iterasi, A_k akan menjadi sebuah matrix triangular yang eigen valuenya merupakan diagonal dari matrix tersebut dan eigen vectornya merupakan kolom-kolom hasil perkalian tiap Q yang dihasilkan dari setiap iterasi yang dilakukan untuk mencapai A_k sebagai matrix triangular.

Hasil implementasi dari *simultaneous power iteration* dapat dilihat pada *code* di bawah ini:

```

def find_eig(matrix):
    n = len(matrix)
    Q = np.identity(n)
    # Q, _ = qr(Q)
    # Q, _ = np.linalg.qr(Q)
    Q, _ = qr2(Q)
    for i in range(100):
        Z = matrix.dot(Q)
        # Q, R = np.linalg.qr(Z)
        # Q, R = qr(Z)
        Q, R = qr2(Z)
    return np.diag(R), Q

```

Gambar 3.1.2. Algoritma *Simultaneous Power Iteration*

Dari *code* di atas, kita melakukan iterasi sebanyak 100 kali untuk memastikan dihasilkannya matrix triangular yang harus kita capai seperti yang telah disebutkan sebelumnya. Dari fungsi *find_eig(matrix)* ini nantinya akan direturn 2 nilai yaitu diagonal dari matrix R yang merupakan nilai eigen dari matrix dan juga matrix Q yaitu eigen spaces yang merupakan hasil perkalian dari semua Q hasil iterasi sebelumnya.

2. Eigenface

Kami memecah algoritma eigenface pada program ini menjadi beberapa fungsi, yakni:

a. mean

```

# MEAN
def mean(extraction_result):
    m = len(extraction_result)
    mean = [0 for i in range(2048)]
    for i in range(m):
        mean = np.add(mean, extraction_result[i])
    mean = np.divide(mean, m)

    return mean

```

Gambar 3.2.1. Algoritma Pencarian Nilai Rata-rata

Fungsi *mean* menghitung nilai rataan dari vektor-vektor wajah hasil ekstraksi gambar sebelumnya.

b. A (phi)

```
# SELISIH (PHI)
def A(extraction_result, mean):
    m = len(extraction_result)
    A = [[0 for i in range(2048)] for j in range(m)]
    for i in range(m):
        A[i] = extraction_result[i] - mean

    return A
```

Gambar 3.2.2. Algoritma Pencarian Nilai Selisih

Fungsi A menghitung matriks vektor selisih dengan mengurangkan setiap vektor wajah hasil ekstraksi pada matriks dengan nilai rataan yang diperoleh dari matriks tersebut.

c. kovarian

```
# MATRIX KOVARIAN
def kovarian(A):
    kovarian = np.matmul(A, np.transpose(A))
    n = len(kovarian)

    return kovarian
```

Gambar 3.2.3. Algoritma Pencarian Nilai Matriks Kovarian

Fungsi kovarian menghitung kovarian dari matriks A dengan prinsip reduksi dimensionalitas / *dimensionality reduction* menggunakan rumus :

$$Cov = A^T A$$

Namun, perlu diperhatikan bahwa orientasi vektor dalam parameter matriks A sudah horizontal, sehingga A berdimensi $M \times N^2$, di mana M adalah jumlah foto dalam dataset dan N adalah pixel resolusi foto persegi. Maka, untuk tetap menerapkan prinsip reduksi dimensionalitas, kita cukup mengalikan $A \times A^T$ dan matriks kovarian yang dihasilkan akan berdimensi $M \times M$.

d. getEigenFaces

```
# EIGENFACE ALGORITHM
def getEigenFaces(eigenSpace, A, k):
    best = eigenSpace[0:k]
    bestOriEigenFace = np.matmul(best, A)

    return bestOriEigenFace
```

Gambar 3.2.4. Algoritma Pencarian Nilai Eigenface

Fungsi getEigenFaces mencari vektor-vektor eigenface sejumlah k yang dapat merepresentasikan wajah-wajah pada dataset.

e. getWeightSet

```
# WEIGHT SET CALCULATOR
def getWeightSet(A, eigenFaces, M):
    weightSet = [[0] for i in range(M)]
    for i in range(M):
        weightSet[i] = np.matmul(A[i], np.transpose(eigenFaces))

    return weightSet
```

Gambar 3.2.5. Algoritma Pencarian Nilai *Weight*

Fungsi getWeightSet mencari vektor-vektor weight dari setiap vektor wajah hasil ekstraksi terhadap vektor-vektor eigenface terpilih.

f. getThreshold

```

# THRESHOLD
def getThreshold(eigenfaceWeight, M):
    for i in range(M):
        start = i+1
        sum = 0
        for k in range(len(eigenfaceWeight[i])):
            sum += eigenfaceWeight[i][k]**2
        magnitude1 = sqrt(sum)
        for j in range(start, M):
            sum = 0
            for k in range(len(eigenfaceWeight[j])):
                sum += eigenfaceWeight[j][k]**2
            magnitude2 = sqrt(sum)
            if (i == 0 and j == 1):
                leng = (eigenfaceWeight[j]/magnitude2) - \
                    (eigenfaceWeight[i]/magnitude1)
                max = np.dot(np.transpose(leng), leng)
            else:
                curr = (eigenfaceWeight[j]/magnitude2) - \
                    (eigenfaceWeight[i]/magnitude1)
                distance = np.dot(np.transpose(curr), curr)
                if (max < distance):
                    max = distance

    t = 0.5*max
    print("Ini threshold")
    print(t)
    return t

```

Gambar 3.2.6. Algoritma Pencarian Nilai *Threshold*

Fungsi getThreshold mencari toleransi kedekatan yang baik dengan mencari euclidean distance terbesar antarvektor dari dataset, kemudian dikalikan dengan 0,5.

g. matcher

```

# MATCHER
def matcher(input, datasetName, mean, weightSet, M, threshold, eigenfaces):
    folder = [os.path.join(datasetName, p)
              for p in sorted(os.listdir(datasetName))]

    selisih = A([extract_features(input)], mean)
    weight = np.matmul(selisih[0], np.transpose(eigenfaces))
    sum = 0

    for k in range(len(weight)):
        sum += weight[k]**2
    magnitude1 = sqrt(sum)

    for i in range(M):
        sum = 0
        for k in range(len(weightSet[i])):
            sum += weightSet[i][k]**2
        magnitude2 = sqrt(sum)
        if (i == 0):
            sum = 0
            leng = (weightSet[i]/magnitude2) - (weight/magnitude1)
            min = np.dot(np.transpose(leng), leng)
            index = i
            distanceWeight = leng
        else:
            leng = (weightSet[i]/magnitude2) - (weight/magnitude1)
            distance = np.dot(np.transpose(leng), leng)
            if (distance < min):
                min = distance
                index = i
                distanceWeight = leng

        if (min > threshold):
            match = False
        else:
            match = True
            matchedPath = folder[index]

    return match, matchedPath, min, distanceWeight, weight

```

Gambar 3.2.7. Algoritma Pencarian Nilai Gambar yang Cocok

Fungsi matcher mengomparasikan vektor weight input dengan kumpulan vektor weight dari dataset dan mencatat gambar dengan euclidean distance terkecil terhadap vektor weight input.

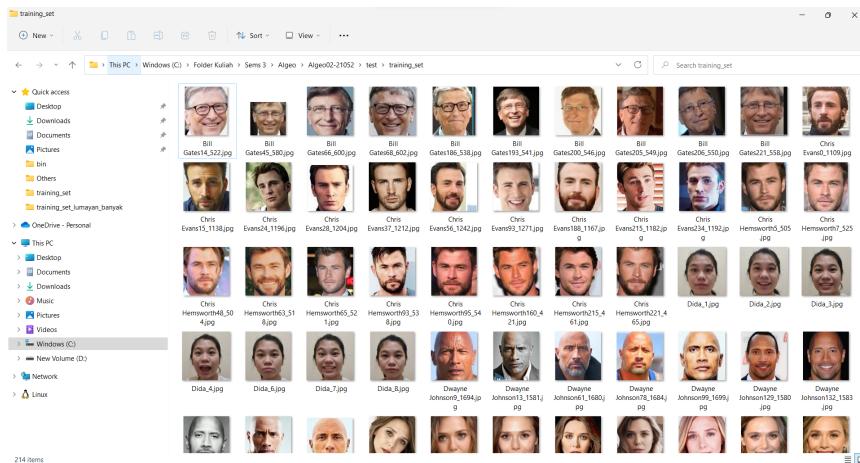
Bab 4

Eksperimen

4.1. Data Set

Terdapat tiga dataset yang digunakan pada percobaan eksperimen implementasi program. Ketiga data set tersebut mengandung foto dari selebriti baik laki-laki maupun perempuan dan beberapa foto dari anggota kelompok sebagai pelaksanaan tes aspek Bonus. Ketiga data set tersebut diurutkan dari yang mengandung foto paling banyak hingga paling sedikit, masing-masing dinamakan:

1. *training_set*, berisi 214 foto,
2. *training_set_lumayan_banyak*, berisi 102 foto, dan
3. *training_set_dikit*, berisi 56 foto.



Gambar 4.1.1. Tampilan Gambar dari Data Set yang digunakan

4.2. Tampilan Page

Terdapat dua page pada program yang telah dibuat, yaitu *Welcome Page* dan *Main Page*. *Welcome Page* adalah halaman yang pertama kali dibuka oleh *user*. Tidak ada fitur yang disuguhkan pada halaman ini selain button untuk menuju ke *Main Page*. *Main Page* adalah halaman utama yang menjadi tempat program algoritma *Face Recognition* dilakukan.

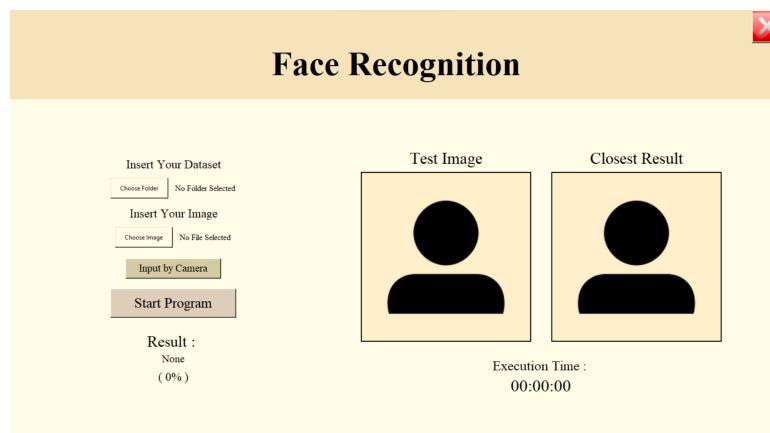
Pada bagian kiri terdapat tombol-tombol bagi *user* untuk memasukkan *folder data set* dan memasukkan *image* yang ingin diidentifikasi. Tak hanya itu, terdapat juga tombol bagi *user* untuk membuka *camera* lalu melakukan identifikasi terhadap gambar yang tertangkap oleh *camera*. Di bawah itu, terdapat tombol bagi *user* untuk memulai program dan mencari gambar apa yang paling mirip dengan gambar yang sedang diidentifikasi.

Terakhir, tertera *result* atau hasil akhir yang program tampilkan, yaitu gambar yang paling mirip dengan gambar yang diidentifikasi dan persentase kemiripannya.

Di sebelah kanan, tertera dua gambar yang digunakan untuk menampilkan gambar yang diinputkan oleh *user* dan gambar yang paling mirip setelah ditentukan oleh algoritma program. Pada bagian bawah, tertera *execution time* yang menampilkan lamanya durasi yang dibutuhkan oleh program untuk menentukan gambar



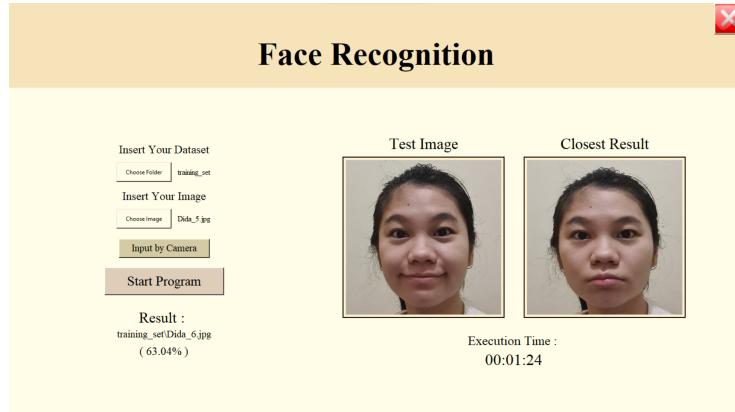
Gambar 4.2.1. Tampilan *Welcome Page*



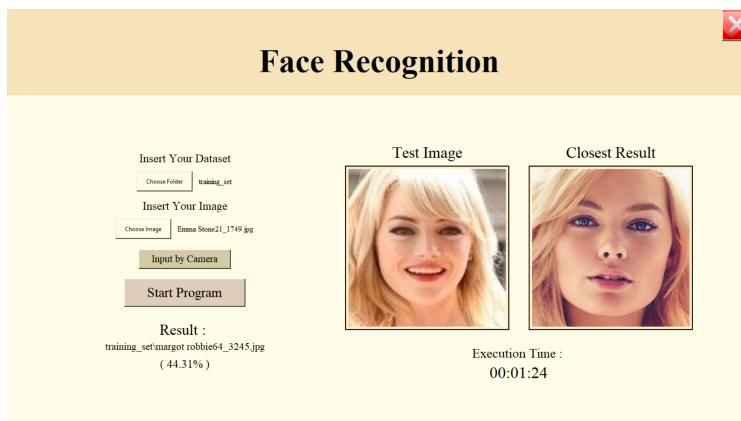
Gambar 4.2.2. Tampilan *Main Page*

4.3. Hasil Eksperimen

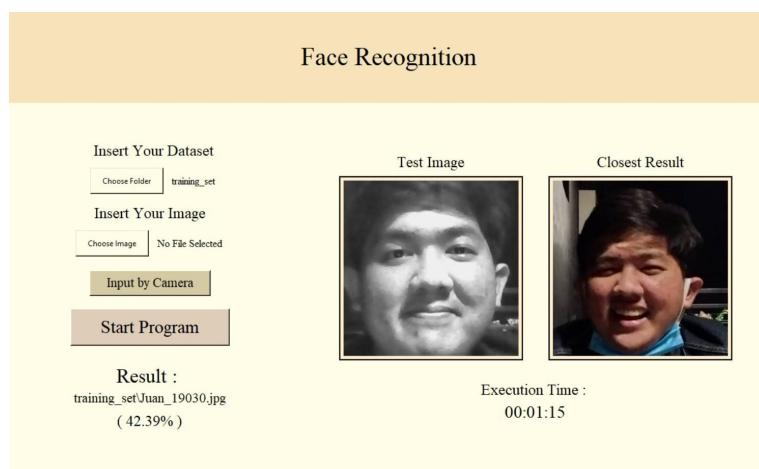
4.3.1. Data set: *training_set*



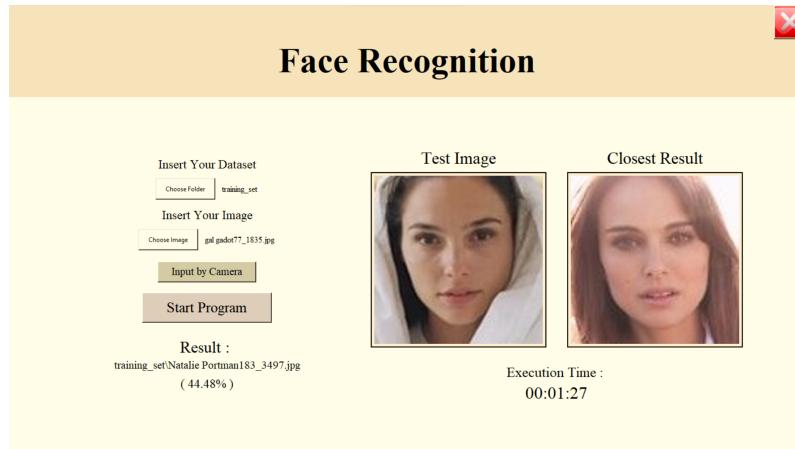
Gambar 4.3.1.1. Eksperimen pada data set *training_set* 1



Gambar 4.3.1.2. Eksperimen pada data set *training_set* 2

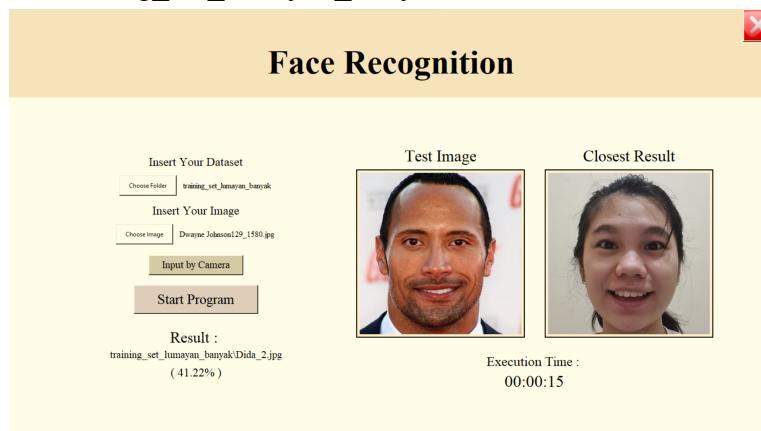


Gambar 4.3.1.3. Eksperimen pada data set *training_set* 3

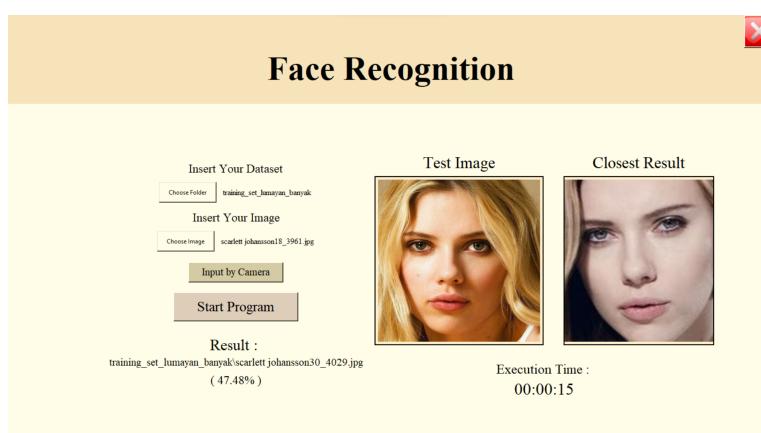


Gambar 4.3.1.4. Eksperimen pada data set *training_set* 4

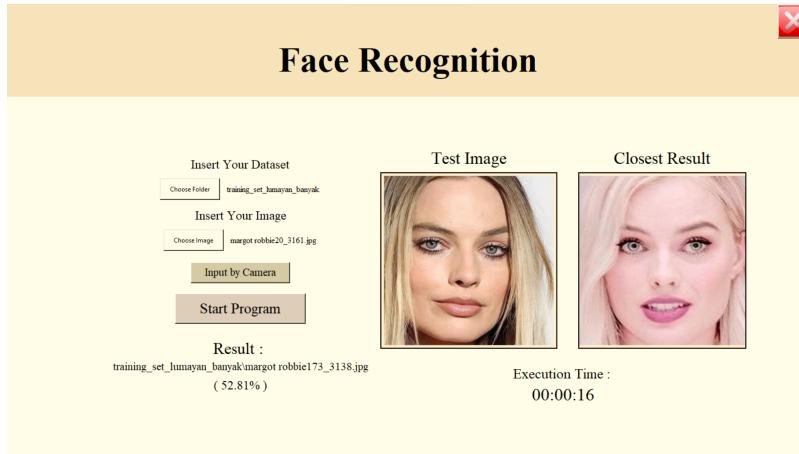
4.3.2. Data set: *training_set_lumayan_banyak*



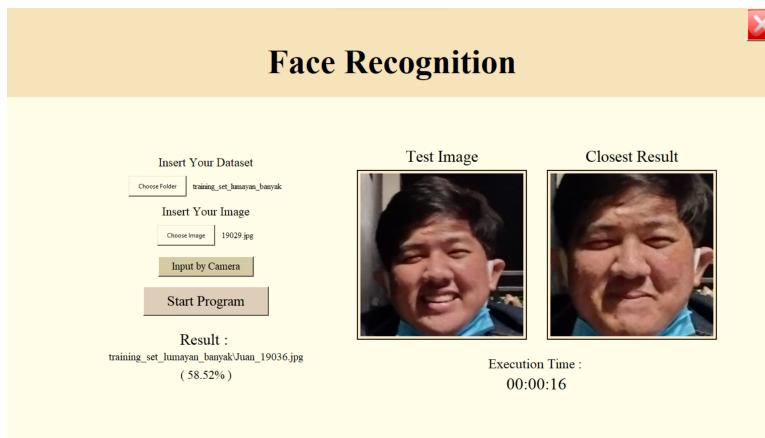
Gambar 4.3.2.1. Eksperimen pada data set *training_set_dikit* 1



Gambar 4.3.2.2. Eksperimen pada data set *training_set_dikit* 2

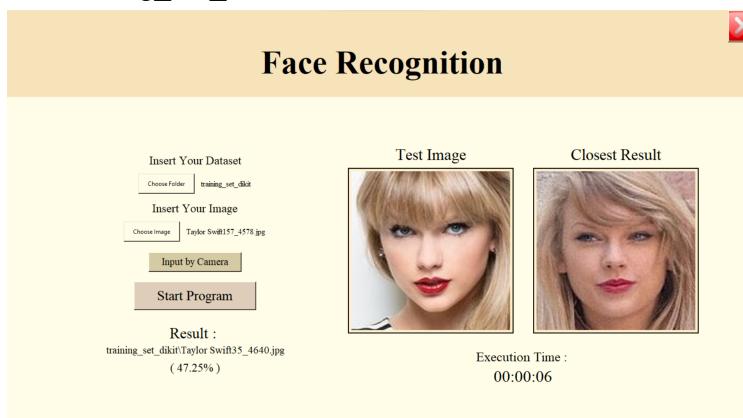


Gambar 4.3.2.3. Eksperimen pada data set *training_set_dikit 3*

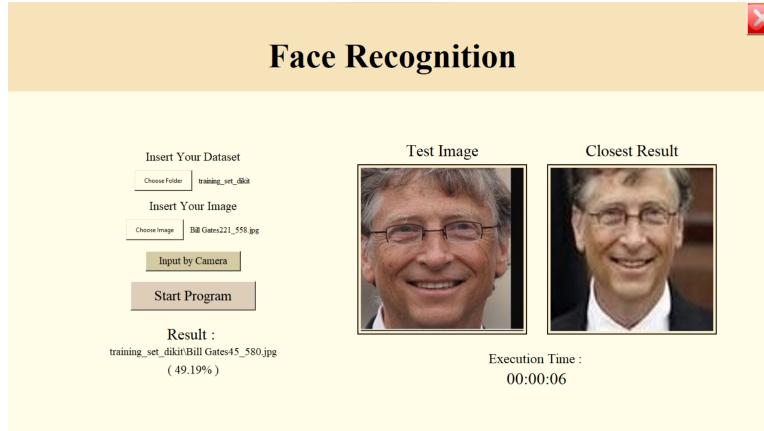


Gambar 4.3.2.4. Eksperimen pada data set *training_set_dikit 4*

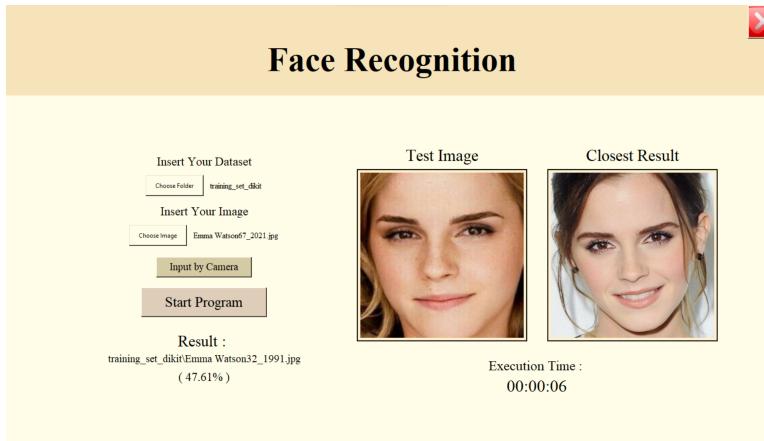
4.3.3. Data set: *training_set_dikit*



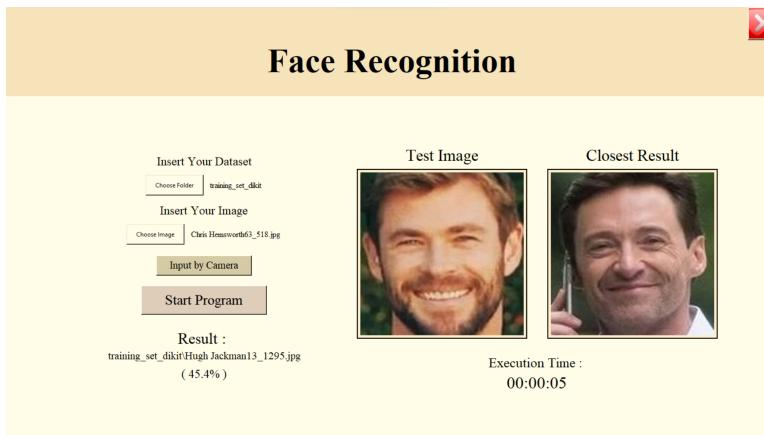
Gambar 4.3.3.1. Eksperimen pada data set *training_set_dikit 1*



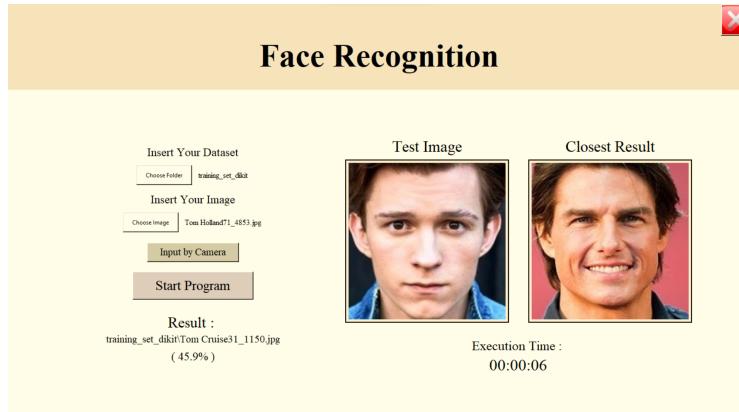
Gambar 4.3.3.1. Eksperimen pada data set *training_set_dikit* 2



Gambar 4.3.3.3. Eksperimen pada data set *training_set_dikit* 3



Gambar 4.3.3.4. Eksperimen pada data set *training_set_dikit* 4



Gambar 4.3.3.5. Eksperimen pada data set *training_set_dikit* 5

4.4. Analisis

Untuk setiap percobaan yang telah dilakukan, data hasil percobaan dapat ditampilkan dalam sebuah tabel sebagai berikut:

Tabel 4.4.1 Pemaparan Data Hasil Eksperimen yang telah dilakukan pada program

Data set	Eksperimen ke-	Benar atau tidaknya <i>result</i> yang ditampilkan	Analisis
<i>training_set</i>	1	Benar	
	2	Salah	Warna dan model rambut yang mirip. Tata letak komponen pada wajah yang mirip.
	3	Benar	
	4	Salah	Sudut kemiringan dan kesejajaran wajah yang serupa. Ditambah dengan tingkat kecerahan gambar yang mirip.
<i>training_set_lumayan_banyak</i>	1	Salah	Ekspresi wajah dan mulut yang mirip.
	2	Benar	
	3	Benar	
	4	Benar	
<i>training_set_dikit</i>	1	Benar	

	2	Benar	
	3	Benar	
	4	Salah	Ekspresi wajah yang mirip (keduanya tersenyum). Kesejajaran wajah yang mirip.
	5	Salah	Bagian rambut pada foto yang terletak pada pixel yang berdekatan. Posisi wajah yang mirip dan berdekatan.

Pada pemaparan data diatas, dapat disimpulkan bahwa terdapat beberapa faktor yang memengaruhi penentuan hasil gambar oleh algoritma yang telah dibuat, yakni:

1. Ekspresi wajah

Hal ini mencakup apakah kondisi wajah pada foto yang sedang diolah memiliki ekspresi datar atau sedang tersenyum. Tak hanya itu, situasi mulut yang sedang menampilkan gigi atau tidak juga memengaruhi hasil gambar yang dikeluarkan program.

2. Warna dan Model Rambut

Model rambut yang dimiliki oleh seseorang pada foto yang sedang diolah tentunya memengaruhi matrix eigenface yang dibuat. Wajah dengan rambut berponi dan wajah dengan rambut tanpa poni tentunya akan dianggap sebagai hal yang sangat berbeda oleh program.

3. Sudut kemiringan wajah

Pada foto, tentu saja wajah yang memiliki sudut kemiringan tampak depan akan memiliki *pixel-pixel* yang berbeda dengan wajah yang memiliki sudut kemiringan wajah tampak samping. Maka dari itu, untuk menghindari hal tersebut, dapat dilakukan seleksi untuk semua data set dengan memilih gambar dengan wajah tampak depan saja.

4. Tingkat kecerahan gambar

Mengingat bahwa tiap *pixel* dalam gambar adalah representasi dari hitam putih (karena gambar sudah dibuat *grayscale*), tentu saja tingkat kecerahan pada gambar akan memengaruhi eigenface ataupun kemiripan dua buah gambar. Program akan relatif untuk menampilkan gambar dengan tingkat kecerahan serupa pada lokasi *pixel* yang sama.

5. Kesejajaran wajah

Meski secara umum kedua gambar berbeda, tetapi program bisa saja menganggap bahwa kedua gambar merupakan gambar yang serupa karena letak komponen wajah yang sejajar. Program akan relatif untuk menganggap dua

gambar adalah gambar yang serupa bila kedua gambar memiliki posisi mata (atau anggota wajah lainnya) pada koordinat *pixel* yang sama.

Berdasarkan *execution time* yang tertera pada tiap eksperimen, dapat disimpulkan bahwa rata-rata waktu yang dibutuhkan untuk menjalankan program adalah sebagai berikut:

Tabel 4.4.1 Rata-rata waktu yang dibutuhkan untuk mengidentifikasi sebuah gambar sesuai dengan data set yang dibutuhkan

Banyaknya data set (gambar)	Waktu yang dibutuhkan (detik)
56	6
102	15
214	84

Dapat dilihat bahwa terdapat kenaikan yang tidak linear pada waktu yang dibutuhkan oleh program untuk menentukan gambar yang mirip dengan banyaknya gambar pada data set. Kenaikan waktu dari 56 data set ke 102 data set masih relatif linear dengan kenaikan jumlah gambar pada data set. Namun, waktu yang dibutuhkan melambung tinggi ketika jumlah gambar pada data set adalah 214 gambar. Hal ini menunjukkan bahwa durasi yang dibutuhkan program untuk mengidentifikasi sebuah gambar memiliki grafik yang eksponensial terhadap banyaknya gambar pada data set yang digunakan.

Bab 5

Kesimpulan

5.1 Kesimpulan

Program *face recognition* dapat diimplementasikan dengan berbagai cara, salah satunya adalah dengan algoritma eigenface. Algoritma eigenface memanfaatkan konsep nilai eigen dan vektor eigen untuk mewakili vektor-vektor wajah yang telah diekstraksi dan dinormalisasi. Adapun kedekatan wajah dengan database / training set dinilai berdasarkan *euclidean distance*.

Pada tugas besar ini, algoritma eigenface yang kami implementasikan masih membutuhkan peningkatan dan optimalisasi, baik dari sisi presisi, maupun efisiensi. Berdasarkan analisis yang kami lakukan, terdapat beberapa faktor yang memengaruhi tingkat akurasi dari pengenalan wajah, yakni ekspresi wajah, model dan warna rambut, *angle/sudut kemiringan wajah*, *brightness/tingkat kecerahan citra wajah*, serta *allignment/kesejajaran wajah antardata dalam dataset*.

5.2 Saran

Adapun saran yang dapat kami berikan kepada pembaca yang ingin mengimplementasikan atau mengembangkan algoritma eigenface ini lebih lanjut di antaranya:

1. Melakukan riset mendalam tentang algoritma yang ingin dipakai untuk mencari nilai eigen, vektor eigen, dan eigenface.
2. Menyeleksi *dataset* yang digunakan sehingga training set yang dipakai program dapat memproduksi hasil yang lebih presisi dan akurat.

5.3 Refleksi

Banyak yang dapat kami pelajari dari pengerjaan tugas besar mata kuliah Aljabar Linier dan Geometri ini, yakni pembagian waktu di tengah kesibukan akademik lainnya, kerja sama dan pembagian tugas agar pekerjaan menjadi lebih mudah, dan pentingnya dekomposisi masalah dalam menyusun sebuah algoritma. Selain itu, kami belajar untuk menggunakan library-library pada Python yang tidak pernah kami gunakan sebelumnya. Tentunya hal ini membuka wawasan kami serta memotivasi kami untuk mendorong rasa ingin tahu serta semangat belajar kami.

Daftar Referensi

- Amos, D. (2022, March 30). *Python GUI Programming With Tkinter*.
<https://realpython.com/python-gui-tkinter/>
- Change button images when you hover over them tkinter.* (2021, January).
<https://stackoverflow.com/questions/65529746/change-button-images-when-you-hover-over-them-tkinter>
- CodingEnterpreneurs (Director). (2018, April 12). *OpenCV Python TUTORIAL #4 for Face Recognition and Identification*. <https://www.youtube.com/watch?v=PmZ29Vta7Vc>
- FolksTalk. (n.d.-a). *Python Tkinter Get Image Size With Code Examples*.
<https://www.folkstalk.com/tech/python-tkinter-get-image-size-with-code-examples/>
- GeeksforGeeks. (2020a, January 7). *Python GUI - tkinter*.
<https://www.geeksforgeeks.org/python-gui-tkinter/>
- GeeksforGeeks. (2020b, December 17). *How to Close a Tkinter Window With a Button?*
<https://www.geeksforgeeks.org/how-to-close-a-tkinter-window-with-a-button/>
- GeeksforGeeks. (2021a, February 15). *Tkinter Application to Switch Between Different Page Frames*.
<https://www.geeksforgeeks.org/tkinter-application-to-switch-between-different-page-frames/>
- GeeksforGeeks. (2021b, September 24). *ML | Face Recognition Using Eigenfaces (PCA Algorithm)*.
<https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/>
- GeeksforGeeks. (2021c, November 29). *Python | geometry method in Tkinter*.
<https://www.geeksforgeeks.org/python-geometry-method-in-tkinter/>
- GeeksforGeeks. (2022, April 21). *Python | grid() method in Tkinter*.
<https://www.geeksforgeeks.org/python-grid-method-in-tkinter/>
- Halton, J. H. (1996). *A Very Fast Algorithm for Finding Eigenvalues and Eigenvectors*.
<https://www.cs.unc.edu/techreports/96-043.pdf>
- Lecture 30. Other Eigenvalue Algorithms.* (n.d.).
<http://www.cs.cmu.edu/afs/cs/academic/class/15859n-f16/Handouts/TrefethenBau/EigenAlgorithms-30.pdf>

Muliawan, M. R., Irawan, B., & Brianorman, Y. (n.d.). *Implementasi Pengenalan Wajah dengan Metode Eigenface pada Sistem Absensi*.
<https://jurnal.untan.ac.id/index.php/jeskommipa/article/download/9727/9500>

Munir, R. (n.d.). *Nilai Eigen dan Vektor Eigen*.
<https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf>

Nikishaev, A. (2018). *Feature extraction and similar image search with OpenCV for newbies*.
<https://medium.com/machine-learning-world/feature-extraction-and-similar-image-search-with-opencv-for-newbies-3c59796bf774>

Ratz, A. V. (2021). *Can QR Decomposition Be Actually Faster? Schwarz-Rutishauser Algorithm*.
<https://towardsdatascience.com/can-qr-decomposition-be-actually-faster-schwarz-rutishauser-algorithm-a32c0cde8b9b>

Saputra, W. M., Wibawa, H. A., & Bahtiar, N. (2014). *Pengenalan Wajah Menggunakan Algoritma Eigenface dan Euclidean Distance*.
<https://ejournal3.undip.ac.id/index.php/joint/article/view/6276>

Shaffer, B. D. (2020). *Eigenvalues and Eigenvectors*.
<https://towardsdatascience.com/eigenvalues-and-eigenvectors-89483fb56d56>

Sharma, D. P. (2021a, March 27). *How do I change button size in Python Tkinter?*
<https://www.tutorialspoint.com/how-do-i-change-button-size-in-python-tkinter>

Sharma, D. P. (2021b, March 27). *Python Tkinter – How do I change the text size in a label widget?*
<https://www.tutorialspoint.com/python-tkinter-how-do-i-change-the-text-size-in-a-label-widget>

Sharma, D. P. (2021c, June 7). *Python Tkinter – Set Entry width 100%*.
<https://www.tutorialspoint.com/python-tkinter-set-entry-width-100>

Sharma, D. P. (2021d, August 6). *Resizing images with Image Tk.PhotoImage with Tkinter*.
<https://www.tutorialspoint.com/resizing-images-with-imagetk-photoimage-with-tkinter#:~:text=To%20resize%20the%20image%2C%20we,displayed%20through%20the%20label%20widget.>

tutoirlaspoint. (n.d.-b). *Python—Tkinter Anchors*.
https://www.tutorialspoint.com/python/tk_anchors.htm

Lampiran

Link Repository Program:

<https://github.com/melvinkj/Algeo02-21052>

Link Video Youtube:

https://youtu.be/ogf_xYdHVAo