# LAPORAN TUGAS BESAR II

# IF2123 ALJABAR LINEAR DAN GEOMETRI

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


Disusun oleh:

| | |
|---|---|
| Muhammad Bangkit Dwi Cahyono | 13521055 |
| Kenneth Ezekiel Suprantoni | 13521089 |
| Arsa Izdihar Islam | 13521101 |

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**


**2022**

# DAFTAR ISI

DAFTAR ISI ..... 2
BAB I DESKRIPSI MASALAH ..... 3
BAB II TEORI SINGKAT ..... 4
A. Perkalian Matriks ..... 4
B. Nilai Eigen dan Vektor Eigen ..... 5
C. Eigenface ..... 7
BAB III IMPLEMENTASI PROGRAM ..... 9
A. Mendapatkan Nilai Eigen ..... 9
B. Mendapatkan Vektor Eigen ..... 10
C. Algoritma Pencocokan Utama ..... 10
D. GUI, Menampilkan Gambar dan Memilih Dataset dan Gambar Test ..... 12
E. Bonus: Menggunakan Video sebagai Data Percobaan ..... 14
BAB IV EKSPERIMEN ..... 16
A. Eksperimen Pertama ..... 16
B. Eksperimen Kedua ..... 17
C. Eksperimen Tambahan (Menggunakan Cache) ..... 19
D. Eksperimen Ketiga (Bonus) ..... 20
BAB V KESIMPULAN, SARAN, DAN REFLEKSI ..... 21
A. Kesimpulan ..... 21
B. Saran ..... 21
C. Refleksi ..... 21
DAFTAR PUSTAKA ..... 23
LAMPIRAN ..... 24
A. Link Repository ..... 24
B. Link Demo (Youtube) ..... 24

# BAB I
# 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 lalu mencocokkan antara kumpulan citra wajah yang sudah dipelajari dengan citra yang akan diidentifikasi

Terdapat berbagai teknik untuk memeriksa citra wajah dari kumpulan citra yang sudah diketahui seperti jarak Euclidean dan cosine similarity, principal component analysis (PCA), serta Eigenface. Pada Projek ini, akan dibuat sebuah program pengenalan wajah menggunakan Eigenface.

Eigenface adalah teknik yang menggunakan matriks dari wajah-wajah, yang akan dihitung matriks eigenface nya, dan dalam pembentukannya, menggunakan eigenvector.

# BAB II
# TEORI SINGKAT

## A. Perkalian Matriks

Perkalian matriks adalah operasi biner yang menghasilkan sebuah matriks baru dari 2 matriks. Untuk perkalian matriks, jumlah kolom dari matriks pertama harus sama dengan jumlah baris pada matriks kedua, dan matriks hasil perkalian akan memiliki dimensi baris dari matriks pertama dan kolom dari matriks kedua. Sebagai contoh, untuk perkalian matriks 1 berdimensi AxB dan matriks 2 berdimensi BxC, akan menghasilkan matriks 3 berdimensi AxC. Perkalian matriks pertama kali dideskripsikan oleh matematikawan Prancis Jacques Philippe Marie Binet pada tahun 1812 untuk merepresentasikan komposisi dari peta linear yang direpresentasikan oleh matriks.

Definisi dari perkalian matriks adalah sebagai berikut, misalkan matriks **A** adalah matriks berdimensi m x n dan matriks **B** adalah matriks berdimensi n x p,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

Matriks **C** yang adalah perkalian dari matriks **A** dan Matriks **B**, **C = AB**, adalah:

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

Sehingga,

$$c_{ij} = a_{i1}b_{1j} + \ldots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

Untuk $i = 1, .., m$ dan $j = 1, .., p$ .

Singkatnya, $c_{ij}$ adalah hasil perkalian titik (*dot product*) dari baris ke *i* dari Matriks **A** dan kolom ke *j* dari Matriks **B**, sehingga Matriks **C** dapat juga direpresentasikan sebagai:

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Sehingga, Matriks **A** dan Matriks **B** bisa dikalikan jika dan hanya jika jumlah kolom dari Matriks **A** sama dengan jumlah baris di Matriks **B**.

## B. Nilai Eigen dan Vektor Eigen

Nilai eigen berawal dari suatu matriks transformasi, misal Matriks **T**, yang akan mentransformasikan suatu vektor jika dikalikan dengan vektor tersebut. Untuk suatu Matriks Transformasi **T**, terdapat suatu vektor **v** yang memiliki *span* s, dimana *span* adalah garis lurus yang sejajar dengan vektor tersebut, yang jika vektor **v** ditransformasikan dengan Transformasi **T**, tidak akan keluar dari *span*-nya, melainkan hanya memanjang, memendek, atau berubah arah dalam *span* s. Konstanta yang menandakan perpanjangan, perpendekan, atau perubahan arah dari vektor **v** setelah ditransformasikan oleh **T** disebut dengan nilai eigen-nya. Dalam kata lain, nilai eigen merepresentasikan suatu matriks Transformasi **T** dengan suatu besaran skalar, yang berarti untuk suatu vektor **v**, hasil dari perkalian dengan matriks **T** dan dengan skalar tersebut akan menghasilkan hasil yang sama. Dalam notasi matematis,

$$\boldsymbol{T}.\vec{v} = \lambda.\vec{v}$$

Dimana **T** adalah matriks Transformasi, $\vec{v}$ adalah vektor yang tidak berubah span nya jika ditransformasikan oleh **T**, dan disebut juga vektor eigen, dan $\lambda$ adalah nilai eigen-nya.

Dapat dilihat bahwa karena $\lambda$ adalah nilai yang berkoresponden dengan konstanta perubahan dari vektor eigen (perpanjangan, pemendekan, atau perubahan arah), maka nilai $\lambda$ yang mungkin untuk suatu Matriks Transformasi **T** bisa lebih dari satu, bahkan bisa juga tidak ada sama sekali.

Untuk mencari nilai-nilai eigen dari suatu Matriks Transformasi **T**, maka dilakukan cara.

$$\boldsymbol{I}.\boldsymbol{T}.\vec{v} = \lambda.\boldsymbol{I}.\vec{v}$$

$$\boldsymbol{T}.\vec{v} = \lambda.\boldsymbol{I}.\vec{v}$$

$$\boldsymbol{T}.\vec{v} - \lambda.\boldsymbol{I}.\vec{v} = 0$$

$$(\boldsymbol{T} - \lambda.\boldsymbol{I}).\vec{v} = 0$$

Sehingga, solusi trivial dari persamaan diatas adalah $\vec{v} = 0$, selalu benar bahwa vektor 0 jika ditransformasikan tidak akan keluar dari *span*-nya, bahkan tidak akan berubah. Agar solusi yang mungkin tidak hanya vektor 0, maka diperlukan solusi nontrivial, yang didapatkan jika dan hanya jika determinan dari Matriks **T** dikurang Lambda dikali Matriks Identitas sama dengan 0,

$$\det(\boldsymbol{T} - \lambda.\boldsymbol{I}) = 0$$

Dari persamaan diatas, akan didapatkan suatu persamaan karakteristik, sebagai contoh:

$$(\lambda - A)(\lambda + B) = 0$$

$$\lambda_1 = A, \qquad \lambda_2 = -B$$

Dimana $\lambda$ adalah nilai eigen dari Matriks Transformasi **T** yang mungkin. Dari nilai eigen tersebut, dapat dicari seluruh vektor eigen-nya dalam bentuk parametrik, sehingga:

$$untuk\ \lambda_1, vektor\ \vec{v}:$$

$$(\boldsymbol{T} - \lambda_1.\boldsymbol{I}).\vec{v} = 0$$

$$untuk\ \lambda_2, vektor\ \vec{v}:$$

$$(\boldsymbol{T} - \lambda_2.\boldsymbol{I}).\vec{v} = 0$$

…

$$untuk\ \lambda_n, vektor\ \vec{v}:$$

$$(\boldsymbol{T} - \lambda_n.\boldsymbol{I}).\vec{v} = 0$$

Dimana basis dari vektor $\vec{v}$ disebut juga dengan ruang eigen atau *eigenbasis*. Yaitu vektor-vektor basis dari vektor eigen $\vec{v}$, dimana karena vektor $\vec{v}$ dinyatakan dalam bentuk parametrik, maka sebenarnya vektor eigen adalah semua vektor yang ada didalam span yang tidak berubah tersebut.

## C. Eigenface

*Eigenface* adalah nama yang diberikan untuk sebuah himpunan vektor eigen saat himpunan tersebut digunakan menyelesaikan permasalahan *face recognition*. Himpunan vektor eigen tersebut diturunkan dari Matriks Kovarian dari distribusi probabilitas dari ruang vektor berdimensi tinggi dari gambar-gambar wajah. Himpunan *Eigenface* itu sendiri akan membentuk sebuah himpunan basis dari semua gambar yang digunakan. Dengan menggunakan metode PCA untuk mereduksi dimensi, diperoleh sebuah himpunan basis yang lebih kecil untuk merepresentasikan gambar wajah aslinya dalam matriks koefisien. Penyocokan juga bisa dicapai dengan membandingkan matriks koefisien yang didapat dari pengalian sebuah *testface* dengan vektor eigen yang didapat dari metode PCA, dan mencari *Euclidean distance* terkecil dari koefisien yang didapat dari *testface* yang adalah gambar terdekat atau termirip dengan *testface* yang diberikan.

Algoritma *eigenface* yang digunakan dalam tugas besar ini adalah algoritma PCA (*Principal Component Analysis*), yang menggunakan Teknik pereduksian dimensi dan memproyeksikan sebuah sampel dalam sebuah ruang fitur kecil. Algoritmanya adalah sebagai berikut:

Dimiliki sejumlah *m* buah gambar berdimensi N * N, yang lalu diubah menjadi vektor $N^2$ * 1, dan disatukan kedalam suatu matriks berukuran $N^2$ * *m* yaitu matriks Data lalu dihitung sebuah matriks A, dimana matriks A adalah matriks yang berisikan semua nilai pada matriks Data yang nilainya dikurangi oleh rata-rata. Setelah itu, dihitung matriks kovarian C:

$$C = A^T . A$$

Digunakan perkalian matriks Transpose dari A dengan A untuk memperoleh dimensi *m* * *m*, yang akan menghasilkan *m* buah nilai eigen dan vektor eigen untuk mempermudah perhitungan. Dimana jika dicari nilai eigen dan vektor eigen nya:

$$A^T . A . v = \lambda . v$$

$$A . A^T . A . v = \lambda . A . v$$

$$C' . A . v = \lambda . A . v \qquad C' = A . A^T$$

$$C' . u = \lambda . u \quad u = A . v$$

Dimana nilai eigen dari C dan C' adalah sama, dan vektor eigen dari C' dapat didapatkan dengan mengalikan matriks A dengan vektor v, yang adalah vektor eigen dari C, sehingga didapatkan vektor eigen dari C' yaitu vektor u yang memiliki dimensi $N^2 * 1$.

Lalu, suatu vektor koefisien w dapat didapatkan dengan *dot product* dari tiap vektor u dengan tiap vektor gambar, secara teori, hasil perkalian dari sebuah gambar dengan semua vektor u tersebut akan menghasilkan tingkat kemiripan dari vektor gambar dan vektor u, karena *dot product* mengukur kesejajaran 2 buah vektor. Sehingga akan dihasilkan suatu vektor w atau *weight* yang adalah tingkat kemiripan gambar tersebut dengan semua vektor u.

Untuk algoritma penyocokan, gambar *testface* akan pertama di proses dari dimensi N * N menjadi $N^2$ * 1. Setelah itu, vektor *testface* akan dikalikan dengan semua vektor u, yang akan menghasilkan vektor w untuk *testface* tersebut. Untuk menyocokan dengan gambar atau muka yang paling mirip, maka digunakan *Euclidean distance* dari semua vektor w pada data dan vektor w dari *testface*. Sehingga, *Euclidean distance* terkecil adalah wajah termirip pada data dengan *testface* yang diberikan. Untuk memunculkan gambarnya, dapat disimpan indeks dari vektor w terdekat, dan dimasukkan kedalam matriks gambar, pada kolom ke-indeks tersebut.

# BAB III
# IMPLEMENTASI PROGRAM

## A. Mendapatkan Nilai Eigen

Terdapat banyak metode yang dapat dilakukan untuk mendapatkan nilai eigen dari suatu matriks. Salah satu cara yang banyak digunakan adalah dengan dekomposisi QR, yaitu apabila dicari nilai eigen matriks A, maka:

$$A = QR$$

Dengan Q adalah matriks orthogonal dan R adalah matriks segitiga atas. Pada kasus pencarian nilai eigen untuk eigenface, matriks A selalu inversible sehingga faktorisasinya akan unik jika diagonal dari elemen R positif.

Untuk mencari dekomposisi QR, terdapat beberapa metode pula. Salah satu metode yang cukup efisien adalah dengan *householder reflections.* Transformasi ini bisa digunakan untuk mengkalkulasi QR faktorisasi dari M * N matriks A dengan M ≥ N. Inisiasi algoritma untuk melakukan transisi ini yaitu dengan mengatur nilai awal Q dengan matriks identitas dengan ukuran M dan R dengan matriks A. Lalu, dilakukan iterasi k sebanyak N-1 kali dengan setiap iterasinya melakukan komputasi berikut. Jika $\boldsymbol{x}$ adalah vektor kolom dari A dengan dimensi M sehingga $||\boldsymbol{x}|| = |\alpha|$ dengan $\alpha$ merupakan skalar maka $\alpha$ harus bertanda terbalik dengan elemen ke-k dari $\boldsymbol{x}$ (jika yang satu bernilai positif, maka yang lainnya bernilai negatif dan sebaliknya). Lalu, atur vektor $\boldsymbol{e_1} = [\,1\ 0\ ...\ 0]^T$ dengan ukuran sama dengan $\boldsymbol{x}$ dan **I** matriks identitas M * M, maka atur:

$$\boldsymbol{u} = \boldsymbol{x} - \alpha \boldsymbol{e_1}$$

$$\boldsymbol{v} = \frac{\boldsymbol{u}}{||\boldsymbol{u}||}$$

$$Q_k = \boldsymbol{I} - 2\boldsymbol{v}\boldsymbol{v}^{\boldsymbol{T}}$$

Lalu, atur Q setelah iterasi menjadi $Q.Q_k$ dan R menjadi $Q_k.R$. Setelah iterasi, maka QR terakhir merupakan dekomposisi dari A. Kemudian, cara untuk mendapatkan nilai eigen dari dekomposisi ini adalah dengan melakukan iterasi QR dengan banyak iterasi

bebas. Iterasi ini dimulai dengan matriks $A_k = A$. Lalu, pada setiap iterasi, dilakukan dekomposisi QR pada matriks $A_k$ kemudian:

$$A_k = R.Q$$

Setelah iterasi berakhir, nilai eigen yang dicari merupakan diagonal dari matriks $A_k$.

## B. Mendapatkan Vektor Eigen

Cara mendapatkan vektor eigen dari suatu matriks bisa dengan cara yang mirip dengan mencari nilai eigen, yaitu dengan dekomposisi QR. Nilai eigen dan vektor eigen dapat dicari sekaligus dalam iterasi QR. Iterasi dimulai dengan matriks QQ adalah matriks identitas N * N dengan N adalah jumlah baris A. Lalu, setiap iterasi dekomposisi QR, atur:

$$QQ = QQ.Q$$

Setelah iterasi berakhir, matriks QQ adalah matriks yang berisi N vektor eigen.

## C. Algoritma Pencocokan Utama

Algoritma Pencocokan Utama, yang tersambung dengan GUI, pertama akan mengambil dataset yang dipilih dan memproses semua gambarnya menjadi sebuah matriks N * N, dimana N adalah dimensi dari gambar, menggunakan kakas opencv dan numpy, dimana opencv digunakan untuk memproses gambar yang di *grayscale* menjadi sebuah matriks value *black/white*, dan numpy digunakan untuk *data type* numpy array, yang digunakan untuk menyimpan matriks sebagai array 2D. Sebelum diproses oleh opencv, dijalankan terlebih dahulu algoritma untuk *crop* gambar tersebut di tengah, menjadi dimensi 256 * 256 *pixel*.

Setelah itu, gambar N * N tersebut ditransformasikan menjadi sebuah vektor $N^2$ * 1, yang lalu dimasukkan kedalam Matriks Data, sehingga matriks data akan berukuran $N^2$ * *m*, dimana *m* adalah jumlah gambar dalam dataset. Dari Matriks Data, dapat diperoleh sebuah nilai skalar rata-rata, yaitu penjumlahan semua elemennya yang lalu dibagi oleh jumlah elemen, yang digunakan untuk mendapatkan Matriks A, yaitu Matriks Data yang semua elemennya dikurangi oleh nilai rata-rata tersebut.

Setelah itu, dihitung Matriks Kovarian C:

$$C = A^T . A$$

Digunakan perkalian matriks Transpose dari A dengan A untuk memperoleh dimensi *m * m*, yang akan menghasilkan *m* buah nilai eigen dan vektor eigen untuk mempermudah perhitungan. Dimana jika dicari nilai eigen dan vektor eigen nya:

$$A^T . A . v = \lambda . v$$

$$A . A^T . A . v = \lambda . A . v$$

$$C' . A . v = \lambda . A . v \qquad C' = A . A^T$$

$$C' . u = \lambda . u \quad u = A . v$$

Dimana nilai eigen dari C dan C' adalah sama, dan vektor eigen dari C' dapat didapatkan dengan mengalikan matriks A dengan vektor v, yang adalah vektor eigen dari C, sehingga didapatkan vektor eigen dari C' yaitu vektor u yang memiliki dimensi $N^2 * 1$.

Lalu, suatu vektor koefisien w dapat didapatkan dengan *dot product* dari tiap vektor u dengan tiap vektor gambar, secara teori, hasil perkalian dari sebuah gambar dengan semua vektor u tersebut akan menghasilkan tingkat kemiripan dari vektor gambar dan vektor u, karena *dot product* mengukur kesejajaran 2 buah vektor. Sehingga akan dihasilkan suatu vektor w atau *weight* yang adalah tingkat kemiripan gambar tersebut dengan semua vektor u.

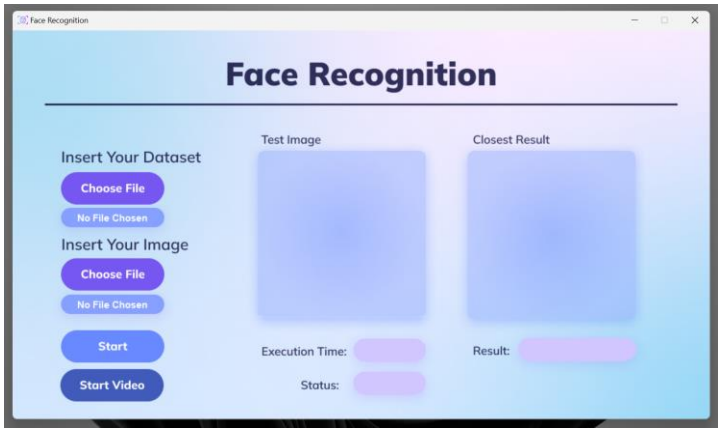
Untuk algoritma penyocokan, gambar *testface* akan pertama di proses dari dimensi N * N menjadi $N^2 * 1$. Setelah itu, vektor *testface* akan dikalikan dengan semua vektor u, yang akan menghasilkan vektor w untuk *testface* tersebut. Untuk menyocokan dengan gambar atau muka yang paling mirip, maka digunakan *Euclidean distance* dari semua vektor w pada data dan vektor w dari *testface*. Sehingga, *Euclidean distance* terkecil adalah wajah termirip pada data dengan *testface* yang diberikan. Untuk memunculkan gambarnya, dapat disimpan indeks dari vektor w terdekat, dan dimasukkan kedalam matriks gambar, pada kolom ke-indeks tersebut.

Sehingga akan didapatkan data yaitu: nilai terkecil *Euclidean distance*, yaitu tingkat kemiripan *testface* dengan suatu gambar di dalam dataset yang dipilih, indeks dari gambar yang memiliki kemiripan tertinggi dengan *testface*, dan juga vektor w dari *testface* tersebut. Dalam hal ini, tingkat kemiripan tertinggi ditandakan oleh *Euclidean distance* terkecil antar vektor w data dan vektor w dari *testface.*

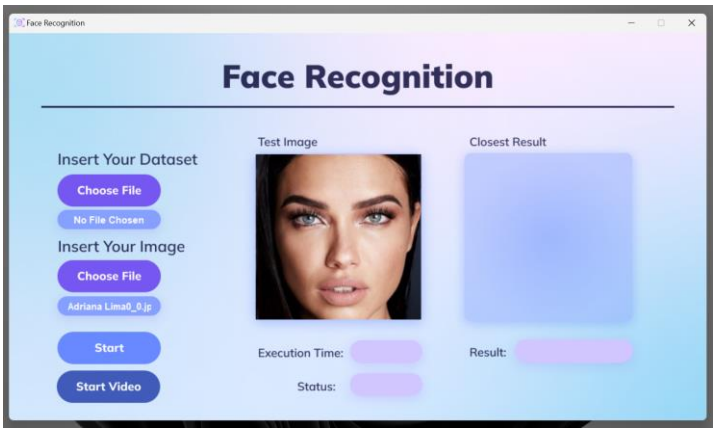
## D. GUI, Menampilkan Gambar dan Memilih Dataset dan Gambar Test

Tech Stack yang kami gunakan dalam tugas besar kali ini adalah python dengan beberapa library tentunya, seperti numpy, cv2, os, time, operator, threading, dan untuk *interface*-nya menggunakan tkinter dan PIL.

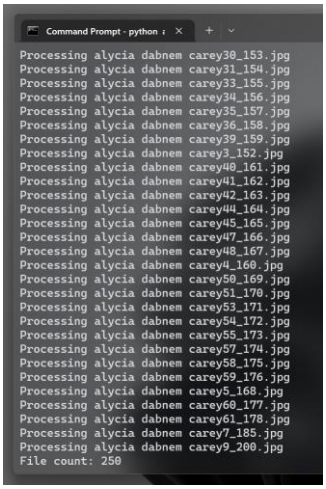
Dalam *interface* yang kami buat, tentunya ada beberapa komponen penting, yakni tombol untuk memilih dataset, tombol untuk memilih gambar yang hendak dicocokkan, label untuk waktu eksekusi, dan label untuk hasilnya. Berikut tampilan awal *interface* yang kami buat.



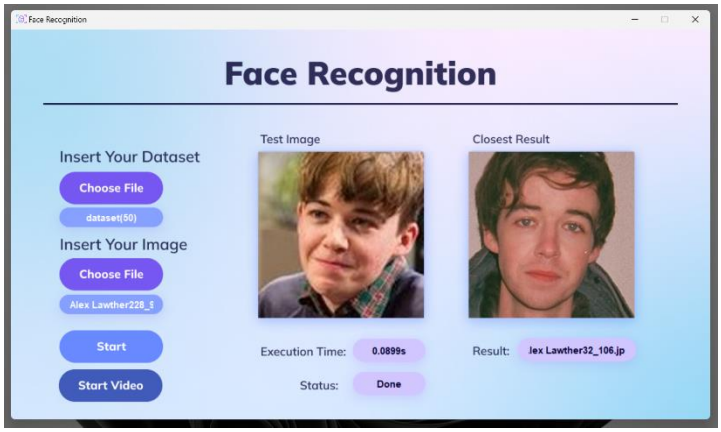

Jika pengguna ingin memasukkan dataset, tekan tombol ‘Choose File’ di bawah ‘Insert Your Dataset’, dan jika pengguna ingin memasukkan gambar yang hendak dicocokkan, tekan tombol ‘Choose File’ di bawah ‘Insert Your Image’, maka tampilan akan seperti ini.




Jika ingin memulai pemrosesan, tekan tombol ‘Start’, lalu pemrosesan akan berjalan. Kita menggunakan multithread sehingga waktu eksekusi akan berjalan terus dan akan stop ketika sudah menemukan hasilnya. Dibalik itu file kita juga akan menampilkan seperti gambar di bawah ini, penanda ada berapa file yang kita proses.



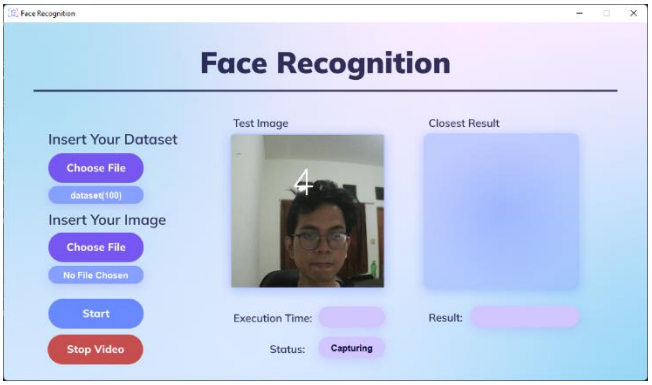

Setelah selesai berikut hasil tampilannya, kita juga menampilkan 3 gambar terdekat beserta disance-nya di *command-line*.




```
distance: 59717477.91823127
indexmin: 64
3 most closest:
- Alex Lawther32_106.jpg - 59717477.91823127
- Alex Lawther74_136.jpg - 81795345.04144816
- alycia dabnem carey5_168.jpg - 105256539.10233173
done
```

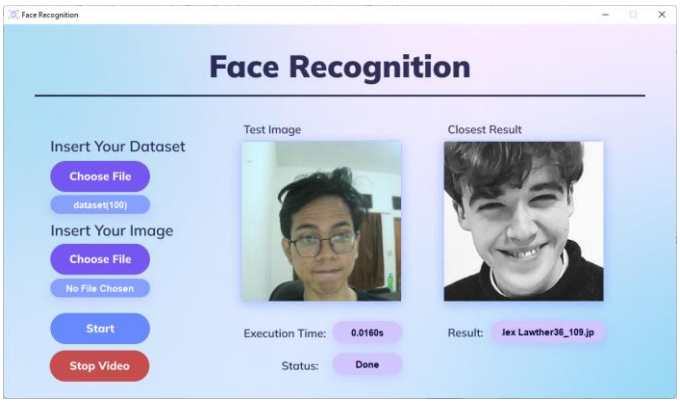
## E. Bonus: Menggunakan Video sebagai Data Percobaan

Selain dapat memilih *file* gambar untuk dijadikan percobaan pada *face recognition*, terdapat fitur tambahan yaitu menggunakan *video capture* dari cv2. Mekanisme fitur ini adalah pertama perlu dipilih folder dataset sebagai data *training*. Kemudian, pada GUI terdapat tombol "*Start Video"* yang apabila ditekan maka akan mengaktifkan kamera untuk diambil datanya. Setelah kamera sudah muncul, program akan menghitung mundur selama 5 detik, lalu gambar dari kamera akan diambil dan diproses.




Setelah gambar selesai diproses, hasil *face recognition* akan ditampilkan seperti fitur sebelumnya. Kemudian, setelah 5 detik, kamera akan muncul lagi dan melakukan hal yang sama seperti sebelumnya. Fitur *cache* sangat membantu fitur ini karena proses gambar

bisa berlangsung dengan sangat cepat apabila data *training* sudah di-*cache* atau setelah pengambilan gambar kedua dan seterusnya.




Saat sedang menggunakan fitur video ini, tombol ”*Start Video”* akan berubah menjadi ”*Stop Video”*. Apabila tombol ini ditekan, maka pengambilan video akan dihentikan dan kembali ke fase awal.

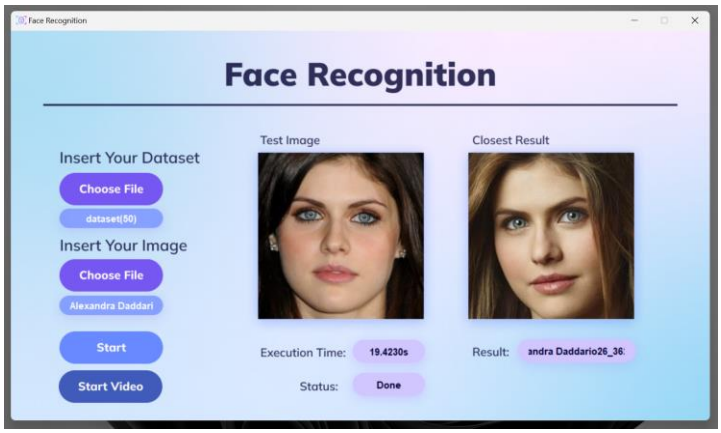
Fitur *cache* yang sempat disebutkan diatas juga merupakan implementasi kami untuk menyimpan suatu dataset yang sudah pernah di-*load* agar lebih cepat, tidak perlu me-*load* ulang. Fitu *cache* ini kami implementasikan dengan meng-*encode* path dengan base64 lalu semua datanya di *dump* kedalam file hasil *encode*-nya. *Encoding* dilakukan agar *path* bisa dijadikan nama file tetapi tetap unik.

# BAB IV
# EKSPERIMEN

Kita melakukan beberapa eksperimen terkait aplikasi yang kita buat, kita mencoba untuk menggunakan dataset berbeda dimana, dataset ini terdiri dari 5 orang namun tiap orang menggunakan jumlah foto yang berbeda. Berikut hasilnya.

## A. Eksperimen Pertama

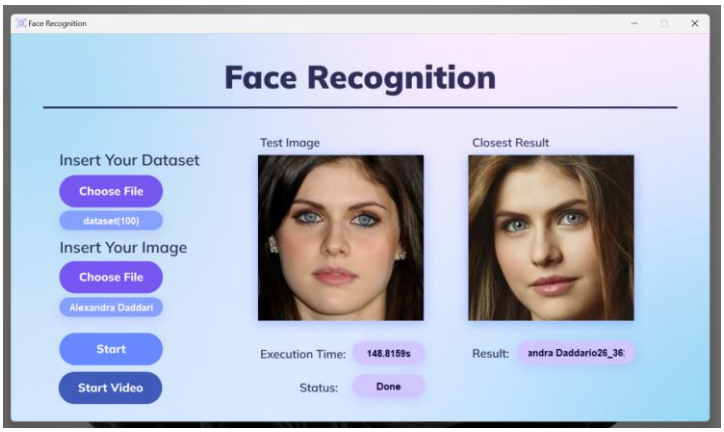
1. Dataset_1 (50/ orang)




```
distance: 34261273.170602106
indexmin: 117
3 most closest:
- Alexandra Daddario26_362.jpg - 34261273.170602106
- Alex Lawther40_113.jpg - 56922994.416568734
- Alexandra Daddario29_365.jpg - 125297258.62715136
done
```

Didapat hasil seperti gambar di atas ini. Waktu eksekusi programnya adalah 19 detik dan ternyata hasilnya orang yang sama. Kita memasukkan *test-image* Alexandra Daddario (di luar dataset awal), ternyata menghasilkan Alexandra Daddario (berada di dataset awal). Namun dapat dilihat juga ternyata 3 gambar terdekatnya, terdapat *gap* yang besar antara *distance* yang kita dapat dengan gambar terdekat ke-2, ini menandakan kecocokannya dengan gambar yang kita dapat sangat mirip dibandingkan dengan foto-foto yang lainnya.

2. Dataset_2 (100/ orang)

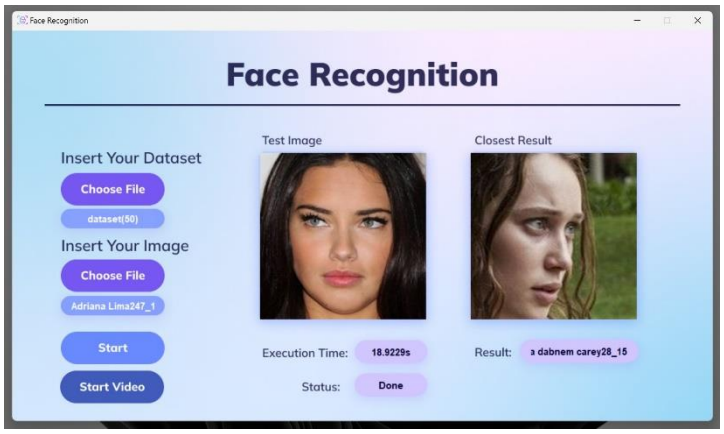



```
distance: 62790361.81295042
indexmin: 222
3 most closest:
- Alexandra Daddario26_362.jpg - 62790361.81295042
- Alexandra Daddario79_418.jpg - 74014076.21215875
- alycia dabnem carey78_190.jpg - 89802202.3387974
```

Didapatkan hasil seperti gambar di atas. Waktu ekseksui programnya 149 detik dan ternyata hasil yang didapatkan sama dengan Dataset_1. Ternyata penambahan dataset menghasilkan *output* yang sama, tetapi dapat dilihat bahwa *gap distance* dengan gambar kedua mengecil. Dengan eksekusi waktu yang 8 kali lipatnya kita tetap mendapatkan hasil yang sama dengan Dataset_1.

## B. Eksperimen Kedua

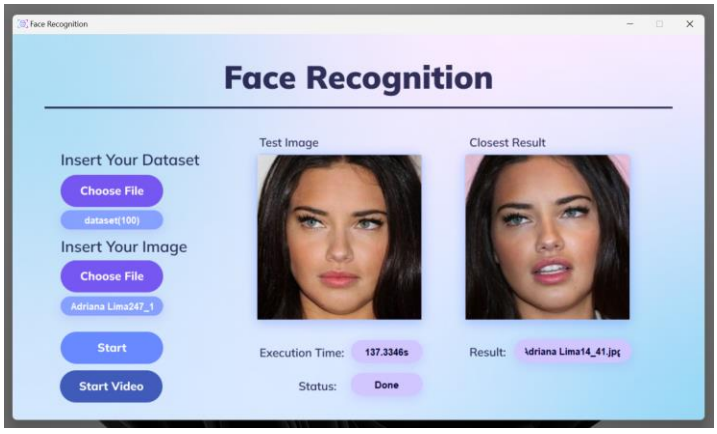
1. Dataset_1 (50/ orang)



```
distance: 33448494.94767242
indexmin: 218
3 most closest:
- alycia dabnem carey28_150.jpg - 33448494.94767242
- Adriana Lima14_41.jpg - 45517330.41656577
- Alvaro Morte54_257.jpg - 51421620.76143189
```

Didapat hasil seperti gambar di atas ini. Waktu eksekusi programnya adalah 19 detik dan ternyata hasilnya orang yang berbeda. Kita memasukkan *test-image* Adriana Lima (di luar dataset awal), ternyata menghasilkan Alycia Dabnem Carey (berada di dataset awal). Namun dapat dilihat juga ternyata gambar terdekat keduanya adalah Adriana Lima (hasil yang kita inginkan). Ternyata tidak semua eksperimen menghasilkan data yang kita inginkan, ini disebabkan karena adanya faktor luar, seperti *lighting, makeup,* tata letak wajah, atau memang kemiripan antar duo orang tersebut, baik dari gender, ras, dan lain-lain.

2. Dataset_2 (100/ orang)



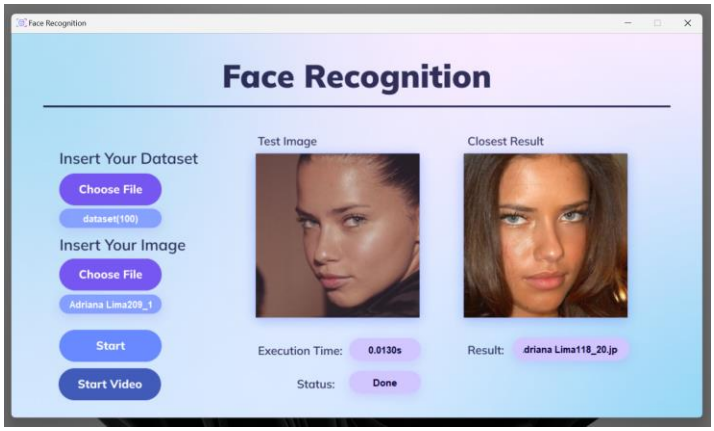

```
distance: 55537902.45881898
indexmin: 23
3 most closest:
- Adriana Lima14_41.jpg - 55537902.45881898
- alycia dabnem carey28_150.jpg - 81801733.32967752
- Alex Lawther16_46.jpg - 88820424.4016136
done
```

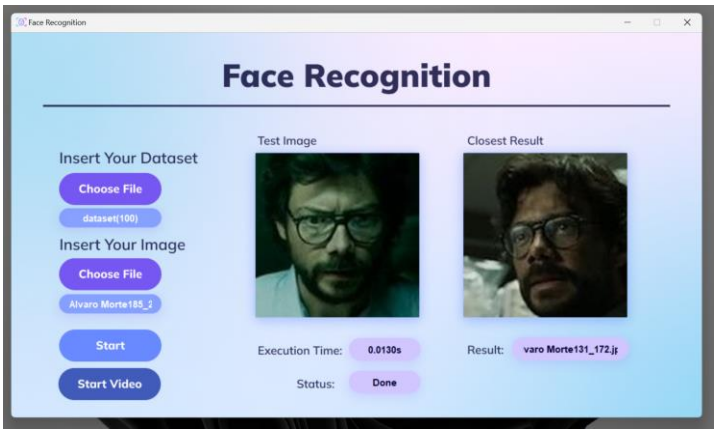
Didapat hasil seperti gambar di atas ini. Waktu eksekusi programnya adalah 137 detik dan ternyata hasilnya berbeda dengan hasil Dataset_1. Kalau ditinjau waktu eksekusinya hampir 7 kali lipatnya, tetapi hasil yang didapatkan berbeda. Dapat dilihat

pada 3 gambar terdekatnya *gap distance* antara hasil dan yang kedua atau ketiga sangat jauh. Hal ini menandakan bahwa dalam *machine learning*, penambahan dataset bisa jadi menambahkan akurasi, tergantung dataset bagaimana yang kita tambahkan jika dataset yang kita tambahkan benar dan tidak terlalu mengubah variasi *meanface*, tetapi jika penambahan dataset malah memperbanyak variasi, bisa jadi dengan penambahan dataset hasil yang akan kita cocokkan akan malah menjauh dari hasil yang diinginkan. Maka dari itu dalam percobaan kali ini parameter benar atau salahnya tidak bisa dilihat karena mesin yang men-*judge* tiap gambar yang ada pada datasest.

## C. Eksperimen Tambahan (Menggunakan Cache)

Pada eksperimen tambahan, kami mencoba untuk menggunakan cache files pada percobaannya sehingga pada dataset yang sama aplikasi tidak harus menghitung ulang lagi tetapi menggunakan cache files dari *training* sebelumnya. Berikut perbandingan waktu dan hasilnya.



Dapat dilihat bahwa waktu yang dibutuhkan hanya 0.01 detik saja dengan dataset yang dari hasil sebelumnya, hal ini menjadi penghematan waktu jika ingin mencoba beberapa gambar pada satu dataset karena akan membutuhkan waktu yang lama pada awal *training* saja.

## D. Eksperimen Ketiga (Bonus)

Pada eksperimen ketiga, ditambahkan fitur untuk menggunakan video sebagai data percobaan. Setelah memilih dataset dan menekan tombol "*Start Video",* berhasil muncul *video capture* secara langsung. Kemudian, pada video tersebut muncul angka hitung mundur selama 5 detik. Setelah itu, sama seperti fitur kedua, hasil dari *capture video* diproses kemudian hasilnya ditampilkan.

# BAB V
# KESIMPULAN, SARAN, DAN REFLEKSI

## A. Kesimpulan

Dalam mencari nilai eigen dan vektor eigen, terdapat suatu kesulitan, dimana Matriks yang dicari nilai eigen dan vektor eigennya perlu didekomposisi terlebih dahulu dengan metode QR, yang memakan waktu yang lama dan hasil yang tidak benar-benar akurat. Ada beberapa hal juga yang ternyata sulit untuk diimplementasikan secara algoritma pemrograman, yaitu, seperti yang disebutkan diatas, adalah mencari nilai eigen, karena tidak bisa dilakukan secara perhitungan matematis manual seperti biasa, dan juga mencari basis eigen dari vektor eigen, karena vektor eigen berbentuk parametrik, dimana kami sadari bahwa determinan dari persamaan Matriks A – $\lambda$.I tidak pasti 0, karena nilai eigen yang kami dapatkan dari metode QR bukan nilai yang 100% akurat, yang kami manfaatkan untuk menjadikan vektor eigen memiliki solusi unik, tak hanya trivial/non-trivial.

Aplikasi yang kita buat tidak 100% akurat, tetapi cukup untuk melihat seberapa dekat hasil gambar ingin kita cocokkan dengan yang ada pada dataset. Ada banyak faktor yang dapat memengaruhi hasil gambar yang kita cocokkan. Misalnya pada dataset yang tidak semua orangnya menghadap ke depan, *background* gambar yang tidak selalu sama, penempatan wajah yang tidak selalu di tengah, *lighting*, dan lain-lain. Hal inilah yang menjadikan bahwa tidak ada parameter benar atau salah dalam percobannya.

## B. Saran

Saran untuk kelompok kami di antaranya:

1. Pembagian tugas dilakukan lebih baik lagi
2. *Time management* dalam pengerjaan tugas besar ini
3. Lebih banyak research terlebih dahulu tentang tugas besarnya
4. Mencari cara-cara alternatif secara lebih cepat dan lebih akurat

## C. Refleksi

Refleksi yang kami dapatkan dari tugas ini adalah kami bisa memperbaiki lagi kinerja kami dalam berbagai hal, contohnya pembuatan timeline kerja agar progress kerja lebih teratur.

Kami juga merasa masih kurang baik dalam *time management* sehingga pengerjaan tugas berdekatan dengan *deadline*. Hal lain yang kami dapatkan dari tugas ini adalah pengalaman baru dan pengetahuan dalam *face recognition*, dari aspek algoritma-nya, cara pengaplikasiannya, dan juga bagaimana teknologi ini dapat dimanfaatkan di dunia nyata. Tak hanya itu, kami juga menyadari bahwa banyak hal yang dapat dengan mudah dihitung secara manual, seperti saat determinan dari Matriks A – $\lambda$.I harus 0, yang menghasilkan suatu *polynomial*, dibandingkan secara komputasi, karena ternyata masih terdapat limitasi dalam membuat sebuah algoritma untuk menghitungnya.

# DAFTAR PUSTAKA


https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf

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

https://www.neliti.com/publications/135138/pengenalan-wajah-menggunakan-algoritma-eigenface-dan-euclidean-distance

https://realpython.com/python-gui-tkinter/

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

https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html

https://www.geeksforgeeks.org/calculate-the-euclidean-distance-using-numpy/

https://learnopencv.com/principal-component-analysis/

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

https://www.cs.cornell.edu/~bindel/class/cs6210-f09/lec18.pdf

https://solarianprogrammer.com/2018/04/21/python-opencv-show-video-tkinter-window/

# LAMPIRAN

**A. Link Repository**

https://github.com/KenEzekiel/Algeo02-21055

**B. Link Demo (Youtube)**

https://youtu.be/mrKpIGfFQqw