

LAPORAN TUGAS BESAR 2
Image Retrieval dan Music Information Retrieval Menggunakan PCA dan
Vektor Semester I Tahun 2024/2025



IF2110 Aljabar Linier dan Geometri

Anggota Kelompok:
Raka Daffa Iftikhaar (13523018)
Ahsan Malik Al Farisi (13523074)
Farrel Athalla Putra (13523118)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

Daftar Isi

Daftar Isi.....	1
BAB 1.....	1
1.1 Pemrosesan Suara dan Gambar.....	1
BAB 2.....	2
2.1 Music Information Retrieval (MIR).....	2
2.2 Metode Ekstraksi Fitur dengan Humming.....	2
2.2.1 Pemrosesan Audio.....	3
2.2.2 Distribusi Tone.....	3
2.2.3 Normalisasi.....	4
2.2.4 Perhitungan Similaritas.....	4
2.3 Image Retrieval.....	4
2.4 Metode Image Retrieval dengan PCA.....	5
2.4.1 Image Preprocessing.....	5
2.4.4 Singular Value Decomposition.....	6
2.4.5 Variansi Data.....	6
2.4.6 Proyeksi Data.....	7
2.4.7 Euclidean Distance.....	7
BAB 3.....	8
3.1 Arsitektur Website.....	8
3.1.1. Back-End.....	9
3.1.2 Front-End.....	9
3.2 Arsitektur Program.....	9
3.2.1 Pemrosesan Gambar.....	9
3.2.2 Pemrosesan Audio.....	10
BAB 4.....	16
4.1 Tabel Eksperimen.....	16
BAB 5.....	20
5.1 Kesimpulan.....	20
5.2 Saran.....	20
5.3 Komentar.....	21
LAMPIRAN.....	22

BAB 1

DESKRIPSI MASALAH

1.1 Pemrosesan Suara dan Gambar

Suara selalu menjadi hal yang paling penting dalam kehidupan manusia. Manusia berbicara mengeluarkan suara dan mendengarkan suatu suara untuk diserap ke otak dan mencari informasi dari suara tersebut. Suara juga bisa dijadikan orang-orang di dunia ini sebuah media untuk membuat karya seni. Contohnya adalah alat mendeteksi lagu. Manusia bisa mendeteksi suara dengan menggunakan indera pendengar dan memberikan kesimpulan akan apa jenis suara tersebut melalui respon dari otak. Sama seperti manusia, teknologi juga bisa mendeteksi suara dan memberikan jawaban mereka melalui algoritma-algoritma yang beragam bahkan bisa melebihi kapabilitas manusia. Dengan menggunakan algoritma apapun, konsep dari mendeteksi dan interpretasi suara itu bisa juga disebut dengan sistem temu balik suara atau bisa disebut juga dengan audio retrieval system. Banyak aplikasi yang menggunakan konsep sistem temu balik contohnya adalah Shazam.

Selain suara, manusia juga memiliki penglihatan sebagai salah satu inderanya dan bisa melihat warna dan gambar yang bermacam-macam. Teknologi komputasi juga memiliki kapabilitas yang sama dan bisa melihat gambar sama seperti kita, tetapi teknologi seperti ini juga bisa merepresentasikan gambar tersebut sebagai beragam-ragam angka yang bisa disebut juga tur. Tahun ke tahun, image processing selalu menjadi fokus utama dari tugas besar 2 Algeo. Algoritma yang digunakan adalah Eigenvalue, Cosine Similarity, Euclidean Distance, dll.

Kami sudah melewati Tugas Besar 1 yaitu tentang matriks dan implementasi terhadap berbagai hal. Matriks adalah salah satu komponen yang penting dalam aplikasi aljabar vektor. Di dalam Tugas Besar 2 ini, kami diminta untuk membuat semacam aplikasi Shazam yaitu sebuah aplikasi yang meminta input lagu dan aplikasi tersebut mendeteksi apa nama dari lagu tersebut dan beberapa detail lainnya. Pada tugas besar ini, kami akan menggunakan aljabar vektor untuk mencari perbandingan antar satu audio dengan audio yang lain. Kami akan menggunakan konsep yang bernama *Music Information Retrieval* atau MIR untuk mencari dan mengidentifikasi suara berdasarkan tur-tur yang dimilikinya. Tidak hanya itu, kami juga akan menggunakan konsep *Principal Component Analysis* (PCA) untuk mencari kumpulan audio melalui deteksi wajah berbagai orang.

BAB 2

TEORI SINGKAT

2.1 *Music Information Retrieval (MIR)*

Music Information Retrieval (MIR) adalah bidang yang menggabungkan berbagai disiplin ilmu untuk mempelajari cara menggunakan komputasi untuk memperoleh, menganalisis, dan mengelola informasi musik. MIR memahami dan mengorganisir data musik dengan menggunakan ilmu komputer, pemrosesan sinyal, statistik, dan musikologi. Salah satu fitur utama MIR adalah ekstraksi fitur musik, seperti pitch, tempo, ritme, timbre, harmoni, dan melodi, yang memungkinkan analisis elemen penting dalam sebuah karya musik. Fitur-fitur ini digunakan untuk berbagai tujuan, seperti klasifikasi genre musik, identifikasi artis, dan pencarian berbasis konten. Misalnya, sistem yang dikenal sebagai "*query by humming*" memungkinkan pengguna menemukan lagu dengan menyenandungkan melodi.

Transkripsi musik otomatis, yang merupakan konversi suara menjadi notasi digital, dan analisis performa musik, yang memiliki kemampuan untuk menilai sinkronisasi, interpretasi, dan ekspresi dalam sebuah penampilan, juga termasuk dalam MIR. Di era digital, MIR digunakan untuk rekomendasi musik personalisasi. Ini mirip dengan apa yang terjadi pada platform streaming seperti Spotify dan Pandora, yang menggunakan AI untuk mengidentifikasi preferensi pengguna. Selain itu, MIR membantu menemukan plagiarisme musik dengan memanfaatkan pola melodi atau harmoni untuk menemukan kemiripan karya. Selain itu, MIR juga membantu menganalisis sentimen musik, yang mengidentifikasi emosi yang terkandung dalam suatu lagu.

MIR secara teknis menggunakan berbagai teknologi untuk memproses sinyal audio; pemrosesan bahasa alami (untuk analisis lirik lagu); dan pembelajaran mesin (untuk clustering dan neural networks). Untuk melatih dan menguji sistem ini, database musik besar seperti Million Song Dataset digunakan. Ini memiliki banyak aplikasi, termasuk pengajaran musik, produksi musik, platform streaming, dan terapi kesehatan berbasis musik. MIR terus berkembang untuk memenuhi kebutuhan analisis musik yang semakin kompleks karena meningkatnya akses ke data besar dan musik digital.

2.2 Metode Ekstraksi Fitur dengan *Humming*

Ekstraksi fitur dengan *humming* bisa menggunakan metode *query by humming*. *Query by humming* (QBH) adalah teknologi dalam bidang Music Information Retrieval (MIR) yang memungkinkan pengguna untuk mencari sebuah lagu dengan cara menyenandungkan atau bersenandung (*humming*) melodi lagu tersebut. Sistem ini memanfaatkan kemampuan komputasi untuk mencocokkan pola melodi yang dinyanyikan pengguna dengan database lagu, bahkan jika nyanyian tidak sempurna atau berbeda secara tempo dan intonasi. Metode ini dilakukan dengan beberapa langkah.

2.2.1 Pemrosesan Audio

Pemrosesan audio adalah tahap untuk mengambil fitur-fitur utama dari audio. Dalam konteks *query by humming*, fitur yang diambil adalah nada, interval nada, dan juga durasi. Pemrosesan ini melalui beberapa tahap diantaranya *preprocessing* yaitu audio yang didapat dari humming diproses awal untuk penghilangan noise, konversi mono, dll. Kedua, *windowing* dimana data audio dibagi menjadi beberapa jendela kecil untuk memudahkan analisis *pitch* per segmen. Tujuan dari *windowing* adalah memungkinkan analisis lokal terhadap fitur musik dan meningkatkan ketepatan dalam membandingkan query *humming* dengan dataset. Pada *windowing*, terdapat dua parameter yaitu ukuran *window* dan pergeseran *window*. Semakin tinggi ukuran, maka informasi yang ditangkap lebih banyak namun bisa kehilangan detail-detail tertentu. Sedangkan semakin besar pergeseran *window*, maka proses akan lebih cepat namun kehilangan beberapa detail juga.

Ketiga, nada yang sudah melalui tahap *windowing* harus dinormalisasi menggunakan rumus sebagai berikut.

$$NP(note) = \frac{(note - \mu)}{\sigma}$$

Tujuan dari normalisasi tersebut adalah menghilangkan perbedaan absolut dari *humming*, mengubah *pitch* menjadi nilai terstandarisasi (berdistribusi normal dengan mean = 0 dan simpangan baku = 1).

2.2.2 Distribusi Tone

Distribusi tone adalah langkah dimana sebuah lagu diukur berdasarkan tiga *viewpoints*. Setelah audio dibagi-bagi melalui proses *windowing*, lagu dibagi lagi menjadi tiga grafik histogram. Pertama adalah ATB atau *Absolute Tone Based* adalah histogram yang mengukur distribusi absolut nada dalam data. Histogram ATB digunakan untuk menangkap karakteristik statis melodi dalam audio yang dimasukan. Histogram ATB memiliki 128 bin yang menghitung frekuensi kemunculan masing-masing nada MIDI dalam data.

Kedua adalah RTB atau *Relative Tone Based* yang menganalisis perubahan antara nada yang berurutan. RTB berguna untuk memahami pola interval melodi yang digunakan untuk mencocokkan *humming* dengan dataset yang tidak bergantung pada *pitch* absolut. RTB menggunakan 255 bin dengan rentang nilai -127 hingga +127 dan melakukan perhitungan selisih antara nada-nada yang berurutan dalam data. Ketiga, *First Tone Based* atau FTB dimana FTB berfokus pada perbedaan antara setiap nada dengan nada pertama. FTB menghasilkan histogram yang mencerminkan hubungan relatif terhadap titik referensi awal. RTB juga membantu menangkap struktur relatif

nada yang lebih stabil terhadap pitch pengguna. RTB menggunakan 255 bin dengan rentang nilai -127 hingga +127.

2.2.3 Normalisasi

Normalisasi dari ketiga histogram yang telah dibuat yaitu ATB, RTB, dan FTB dibuat untuk menghasilkan distribusi yang simbang. Normalisasi juga memastikan bahwa semua nilai dalam histogram berada dalam skala probabilitas. Normalisasi disini menggunakan rumus sebagai berikut.

$$H_{norm} = \frac{H[d]}{\sum_d^{127} H[d]}$$

Pada rumus tersebut, H berperan sebagai histogram dan d adalah bin dari histogram yang dihitung.

2.2.4 Perhitungan Similaritas

Perhitungan similaritas adalah proses paling krusial pada *query by humming*. Hal tersebut dikarenakan inti dari *query by humming* adalah menghitung kemiripan dari sebuah audio. Perhitungan similaritas dilakukan dengan mengubah setiap histogram menjadi sebuah vektor. Vektor tersebut akan dihitung kemiripannya dengan menggunakan *cosine similarity*. *Cosine similarity* adalah ukuran untuk menentukan seberapa mirip dua vektor dalam ruang berdimensi tinggi. Perhitungan dilakukan dengan menghitung sudut cosinus di antara kedua vektor. Formula dari *cosine similarity* adalah sebagai berikut.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

2.3 Image Retrieval

Image retrieval adalah teknik untuk mencari dan mengembalikan informasi visual yang sesuai dari sekumpulan data gambar (image data set) dengan fitur atau kriteria tertentu. Proses dari *image retrieval* adalah dengan mencocokkan karakteristik utama dari gambar yang dicari dengan gambar yang ada di dalam basis data, sehingga pengguna dapat menemukan gambar yang sesuai dengan cepat dan akurat. Implementasi dari *image retrieval* berada di banyak bidang, misalnya pengenalan wajah, pencarian gambar di internet, analisis kondisi medis seseorang, dan pada kali ini yaitu mencari gambar dari suatu album.

2.4 Metode *Image Retrieval* dengan PCA

PCA merupakan salah satu dari banyak metode dalam *Image retrieval*. PCA atau Principal Component Analysis adalah metode yang diturunkan dari aljabar linear dengan melakukan reduksi dimensi data dengan mengidentifikasi komponen utama atau yang memiliki variasi terbesar dalam suatu gambar dan ekstraksi fitur utama. Kemudian setelah didapatkan komponen utamanya akan diproyeksikan datanya dan akan diukur jarak proyeksi gambar yang dicari dengan proyeksi gambar dataset.

2.4.1 Image Preprocessing

Image preprocessing adalah proses persiapan tiap gambar agar dapat diproses dengan baik dalam perhitungan PCA. *Image preprocessing* akan mengubah gambar RGB menjadi gambar grayscale. Proses ini esensial untuk mengurangi jumlah data yang harus diproses, karena RGB merupakan salah satu citra digital yang tiap pikselnya merepresentasikan satu warna asli dan tiap pixel mengandung 24 bit (8 bit merah, 8 bit hijau, dan 8 bit biru), sedangkan grayscale hanya mengandung 8 bit per pixel atau hanya intensitas cahaya dari 0 (hitam) hingga 255 (putih). Dengan reduksi data ini pencarian gambar dilakukan dapat dilakukan dengan lebih efisien namun tetap menjaga informasi penting dari data. Pengubahan warna pixel dari RGB ke Grayscale dapat dilakukan dengan persamaan :

$$I(x, y) = 0.2989 \cdot R(x, y) + 0.5870 \cdot G(x, y) + 0.1140 \cdot B(x, y)$$

Setelah menjadi grayscale, akan dilakukan penyamaan histogram gambar hal ini dilakukan agar gambar dengan fitur gambar (kontras, eksposur, saturasi) yang berlebihan akan menjadi lebih seragam dengan gambar yang normal. Selain itu gambar juga akan dilakukan resizing agar mengurangi jumlah pixel yang harus diproses, namun tetap menjaga fitur utamanya. Kemudian data gambar dikembalikan menjadi satu array agar dapat diproses dan dilakukan perhitungan PCA.

2.4.2 Standarisasi Data

Standarisasi data adalah langkah awal dalam melakukan PCA. Hal ini dilakukan dengan nilai pixel fitur gambar dikurangi dengan nilai rata-rata tiap pixel fitur pada gambar di dataset kemudian membaginya dengan nilai standar deviasi. Data standarisasi penting dilakukan agar agar tiap fitur memiliki rata-rata mendekati nol, sehingga PCA dapat bekerja lebih optimal. Jika tidak dilakukan, maka komponen utama dapat dipengaruhi oleh bias lokasi data. Standarisasi data dapat dilakukan dengan persamaan berikut

$$x_{\text{standardized}} = \frac{x - \bar{x}}{\sigma}$$

dengan σ = Nilai standar deviasi.

2.4.3 Matriks Kovarian

Varians adalah nilai akan bagaimana nilai-nilai bervariasi dalam suatu variabel dan sangat bergantung akan seberapa jauh nilai-nilai tersebut satu dengan yang lain.

Kovarians hanya dapat dihitung antara dua variabel, namun matriks kovarian dapat menunjukkan nilai kovarian dari setiap pasangan variabel dalam data multivariat.

$$\begin{matrix} & & x & y & z \\ x & \left[\begin{matrix} var(x) & cov(x,y) & cov(x,z) \\ cov(x,y) & var(y) & cov(y,z) \\ cov(x,z) & cov(y,z) & var(z) \end{matrix} \right] & y & z \end{matrix}$$

Sehingga, matriks kovarians menunjukkan hubungan linier antara tiap fitur di data. Dalam PCA, eigen vektor dari matriks kovarian menentukan besaran sebaran ke arah komponen utama. Matriks kovarian dapat dihitung dengan persamaan :

$$C = \frac{1}{N} X^T X$$

dengan X adalah data yang sudah distandarisasi, dan N adalah jumlah data set yang ada.

2.4.4 Singular Value Decomposition

Singular Value Decomposition adalah suatu metode untuk mendekomposisi data menjadi tiga matriks yaitu matriks U , S , dan V^T . Dengan matriks U adalah matriks ortogonal matriks awal, Σ adalah matriks diagonal berisi nilai singular atau akar dari nilai eigen value, dan V^T adalah matriks ortogonal berisi eigen vektor.

$$X = U \Sigma V^T$$

Ketika kita melakukan SVD pada matriks kovarian, akan dihasilkan bentuk SVD dengan matriks U dan V yang sama karena bentuk matriks kovarian yang simetris.

$$C = U S U^T$$

Namun dalam PCA karena kita melakukan SVD pada matriks kovarians maka matriks S yang memiliki nilai singular akan sama dengan nilai eigenvalue dari matriks terstandarisasi X .

2.4.5 Variansi Data

Variansi data adalah ukuran akan seberapa tersebaranya data dalam dimensi tertentu. Tiap komponen utama (*principal components*) memiliki nilai variansi yang menunjukkan seberapa besar fitur utama yang dimiliki gambar tersebut. Variansi tersebut diwakilkan dengan nilai eigen value atau dalam hasil SVD adalah nilai singular yang berkorespondensi pada matriks S . Untuk menghitung jumlah variansi dari tiap gambar kita dapat menggunakan persamaan :

$$Explained Variance = \frac{\lambda_i}{\sum \lambda}$$

Dengan demikian kita dapat memilih jumlah komponen utama yang cukup mewakili fitur utama dari gambar atau yang memiliki variansi tertinggi, biasanya dengan memilih n komponen dengan threshold 95%. Untuk menentukan n komponen dapat dilakukan dengan persamaan :

$$\text{Cumulative Variance}_i = \sum_{j=1}^i \lambda_j$$

Kemudian memilih n komponen dengan *cumulative variance* yang nilainya lebih dari sama dengan 0,95.

2.4.6 Proyeksi Data

Setelah menentukan jumlah komponen yang harus dipilih kemudian kita proyeksikan matriks yang telah distandarisasi kepada eigenface atau eigenvector dari matriks kovarian dengan melakukan dot product kepada dua variabel tersebut. Proyeksi ini dapat dilakukan dengan persamaan :

$$\text{Proyeksi Dataset} = X \cdot \text{eigenface} = X \cdot V$$

dengan X adalah data yang sudah distandarisasi atau data gambar yang dicari.

2.4.7 Euclidean Distance

Euclidean distance adalah metode untuk mengukur kesamaan antara vektor proyeksi data query dengan vektor proyeksi dataset. Dengan demikian semakin kecil jarak antara dua vektor, maka semakin mirip juga gambarnya. Persamaan euclidean distance adalah sebagai berikut.

$$d(a, b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

dengan a adalah proyeksi gambar query dan b adalah proyeksi dataset gambar ke i.

Setelah menghitung jaraknya kita dapat mengurutkannya dari jarak terkecil hingga jarak terjauh, dengan jarak terkecil menunjukkan gambar yang paling mirip di dataset dengan gambar query. Selain dari mengurutkan jarak terkecil kita juga dapat menghitung nilai kemiripan gambar di dataset dengan gambar query dengan persamaan berikut.

$$\text{similarity} = 1 - \left(\frac{\text{distance}}{\text{mean distance}} \right)$$

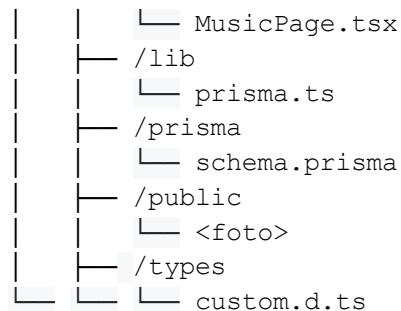
sehingga didapatkan nilai similaritas yang dapat dikali dengan skala 100 untuk mendapatkan nilainya dalam bentuk persen.

BAB 3

ARSITEKTUR WEBSITE DAN PROGRAM

3.1 Arsitektur Website

```
/Algeo02-23018
├── /backend
│   ├── /app
│   │   ├── /__pycache__
│   │   └── main.py
│   ├── /uploads
│   │   ├── <dataset foto>
│   │   ├── <dataset musik>
│   │   └── <mapper>
│   └── requirements.txt
└── /frontend
    ├── /app
    │   ├── /about
    │   │   └── page.tsx
    │   ├── /api
    │   │   ├── /check-dataset-status
    │   │   │   └── route.ts
    │   │   ├── /delete-dataset
    │   │   │   └── route.ts
    │   │   ├── /get-dataset-names
    │   │   │   └── route.ts
    │   │   ├── /save-dataset
    │   │   │   └── route.ts
    │   │   └── favicon.ico
    │   ├── global.css
    │   ├── layout.tsx
    │   └── page.tsx
    ├── /components
    │   ├── AllPage.tsx
    │   ├── FileUploader.tsx
    │   ├── ImagePage.tsx
    │   ├── InputImage.tsx
    │   └── InputMidi.tsx
```



Struktur utama sebuah website dirancang untuk memenuhi kebutuhan fungsional dan non-fungsional dari proyek yang dikembangkan. Dalam hal ini, website memiliki dua fitur utama, yaitu pencarian gambar menggunakan PCA dan pencarian informasi musik (MIR) melalui metode query by humming. Berikut adalah penjelasan mengenai elemen-elemen utama dalam arsitektur website tersebut:

3.1.1. Back-End

- App : Berisi API yang berfungsi untuk mengirimkan data dari database, memroses query file, hingga mengirimkan hasil dari pemrosesan tersebut.
- Uploads : Sebagai tempat folder penyimpanan seluruh input dari *user*, baik berupa dataset ataupun query file.
- Requirements : Sebagai *prerequisite* yang dibutuhkan untuk menjalankan backend.

3.1.2 Front-End

- About : Halaman yang berisikan informasi mengenai pembuat dari website.
- API : Berisi API yang berfungsi untuk mengambil kebutuhan data yang diperlukan selama proses PCA dan MIR.
- Page : Halaman utama dari website yang berisi Hero, Navigation Bar, Input File, MusicPage, dan ImagePage.
- Components : Berisi *reusable components* yang digunakan dalam website, dengan AllPage sebagai halaman yang menampilkan seluruh musik, FileUploader berisi input untuk memasukkan file, ImagePage berisi hasil pencarian PCA, InputImage sebagai komponen untuk memasukkan *query file* foto, InputMidi sebagai komponen untuk memasukkan *query file* musik, dan MusicPage berisi hasil pencarian MIR.

3.2 Arsitektur Program

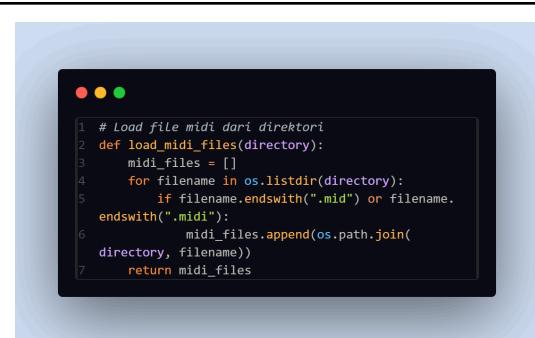
3.2.1 Pemrosesan Gambar

Fungsi	Penjelasan
--------	------------

<pre> 1 # 1. Preprocessing Image def preprocess_image(img, image_size=(50, 50)): img = cv2.equalizeHist(img) img_resized = cv2.resize(img, image_size) e return img_resized / 255.0 # Normalize def preprocess_images(folder_path, image_size=(50, 50)): # Array untuk menampung file name images = [] filenames = [] processed_image = [] original_images_color = [] valid_extensions = ('.png', '.jpg', '.jpeg') s for filename in os.listdir(folder_path): if os.path.splitext(filename.lower())[1] in valid_extensions : img_path = os.path.join(folder_path, filename) try : img_color = cv2.imread(img_path, cv2.IMREAD_COLOR) r original_images_color.append(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB)) s img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY) img_preprocess = cv2.resize(cv2.equalizeHist (img_gray), image_size) images.append(img_preprocess .flatten() / 255.0) e filenames.append(filename) processed_image.append(img_preprocess) except Exception as E: d print(f"Error reading file: {filename}") continue s r return np.array(images), filenames, processed_image , original_images_color </pre>	<p>Fungsi ini berguna untuk melakukan <i>preprocessing</i> pada gambar dengan pertama membacanya dari suatu file path kemudian diubah menjadi dalam bentuk matriks dan dilakukan pengubahan citra menjadi <i>grayscale</i>, <i>equalizing histogram</i>, <i>resizing</i> dalam ukuran 50x50 pixel dan normalisasi pixel (membagi dengan 255)</p>
<pre> 1 # 2. Data Centerin 2 def standardize_image(image_matrix): 3 mean_vector = np.mean(image_matrix, axis=0) 4 std_vector = np.std(image_matrix, axis=0) 5 standardized_matrix = (image_matrix - mean_vector) / (std_vector + 1e-10) 6 return standardized_matrix, mean_vector, std_vector 7 8 </pre>	<p>Fungsi ini berfungsi untuk menstandarisasi gambar dengan mengurangi array matriks dengan rata-ratanya kemudian membaginya dengan standar deviasi.</p>
<pre> 1 # 3. Plot Explained Variance 2 def plot_explained_variance(sv (eigenValue , eigenValueSu): 3 explained_variance = eigenValue / eigenValueSu 4 cumulative_variance = np.cumsum(explained_variance) 5 n_component = np.argmax(cumulative_variance >= 0.95) + 1 6 s 7 return cumulative_variance , n_component 8 9 </pre>	<p>Fungsi ini berfungsi untuk mengambil jumlah n komponen (threshold 95%) dengan cara mengetahui seberapa penting fitur tersebut berpengaruh pada gambar.</p>
<pre> 1 # 4. Compute PCA with SVD 2 def compute_pca_with_sv (standardized_matrix , U, n_component): 3 UK = U[:n_component] * e 4 projections = np.dot (standardized_matrix , UK) 5 return UK, projections 6 7 </pre>	<p>Fungsi ini menghitung PCA dari jumlah n komponen hasil SVD dan memproyeksikan datanya.</p>
<pre> 1 # 5. Similarity Computation 2 def calculate_euclidean_distance(query_projection , dataset_projection): 3 distances = np.sqrt(np.sum((query_projection - dataset_projection)**2, axis=1)) 4 mean_distanc = np.mean(distances) + 1e-10 5 similarity = (1 - distances / mean_distanc) * 100 6 return similarity , distances 7 8 </pre>	<p>Fungsi ini berfungsi untuk mengkalkulasi jarak euclidean antara proyeksi gambar query dengan proyeksi gambar data set.</p>

3.2.2 Pemrosesan Audio

Fungsi	Penjelasan
--------	------------



```
1 # Load file midi dari direktori
2 def load_midi_files(directory):
3     midi_files = []
4     for filename in os.listdir(directory):
5         if filename.endswith(".mid") or filename.
6             endswith(".midi"):
7             midi_files.append(os.path.join(
directory, filename))
8     return midi_files
```

Fungsi ini berguna untuk me-*load* file MIDI dari direktori pengguna.



```
1 # Ekstrak fitur-fitur dari file midi
2 def extract_features_with_tempo(midi_file):
3     try:
4         mid = MidiFile(midi_file)
5     except Exception as e:
6         print(f"Error reading {midi_file}: {e}")
7         return [], [], [], 500000
8
9     pitches, velocities, timings = [], [], []
10    tempo = 500000
11
12    current_time = 0
13    for track in mid.tracks:
14        for msg in track:
15            current_time += msg.time
16            if msg.type == 'set_tempo':
17                tempo = msg.tempo
18            elif msg.type == 'note_on' and msg.
19                velocity > 0:
20                pitches.append(msg.note)
21                velocities.append(msg.velocity)
22                timings.append(current_time)
23
24    return pitches, velocities, timings, tempo,
25    mid.ticks_per_beat
```

Fungsi ini berfungsi untuk mengambil fitur-fitur yang ada pada file MIDI seperti *pitch*, *velocity*, *timing*, *tempo*, dan *ticks per beat*.

```

1 # Melakukan proses windowing dan normalisasi
2 def segment_pitches_with_windowing_and_normalization(pitches, timings, window_size_beats,
3     step_size_beats, ticks_per_beat, tempo):
4     segments = []
5
6     if not pitches or not timings:
7         return segments
8
9     # Konversi window size dan step size ke dalam ticks
10    window_size_ticks = window_size_beats *
11        ticks_per_beat
12    step_size_ticks = step_size_beats *
13        ticks_per_beat
14
15    start_idx = 0
16    while start_idx < len(timings):
17        end_idx = start_idx
18        while end_idx < len(timings) and (
19            timings[end_idx] - timings[start_idx] <=
20            window_size_ticks):
21            end_idx += 1
22        segment_pitches = pitches[start_idx:
23            end_idx]
24        if segment_pitches:
25            # Normalisasi pitch dengan mean dan std
26            normalized_segment =
27                normalize_pitch_with_mean_std(segment_pitches)
28            segments.append(normalized_segment)
29
30    # Geser window
31    start_idx += int(step_size_ticks)
32
33    return segments

```

Fungsi ini digunakan untuk melakukan proses *windowing* dan normalisasi hasil *windowing* tersebut. Penentuan jumlah segmen dan *sliding window* akan diterapkan pada fungsi main. Untuk normalisasi, fungsi ini akan memanggil fungsi `normalize_pitch_with_mean_std` yang akan dijelaskan di bagian selanjutnya.

```

1 # Fungsi untuk normalisasi pitch dengan mean dan std
2 def normalize_pitch_with_mean_std(pitches):
3     if len(pitches) == 0:
4         # Menghindari pembagian dengan 0
5         return []
6     mean_pitch = np.mean(pitches)
7     std_pitch = np.std(pitches)
8     std_pitch = std_pitch if std_pitch > 0 else 1
9     normalized_pitches = [(pitch - mean_pitch) /
10        std_pitch for pitch in pitches]
11
12    return normalized_pitches

```

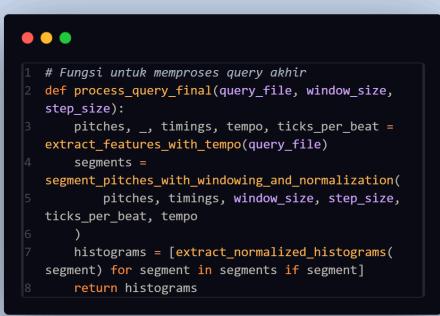
Fungsi ini digunakan untuk normalisasi tempo dengan rumus yang sudah diberikan di atas. Fungsi ini sudah dipanggil pada fungsi di atas setelah melakukan *windowing*.

```

1 # Fungsi untuk menghitung dan normalisasi histogram
2 def compute_and_normalize_histogram(data, bins,
3     range_):
4     hist, _ = np.histogram(data, bins=bins, range=
5         range_)
6     hist = hist.astype(np.float64)
7     total = np.sum(hist)
8     return np.divide(hist, total, out=np.zeros_like(
9         hist), where=total > 0)

```

Fungsi ini digunakan untuk melakukan normalisasi ATB, RTB, dan FTB yang akan dibuat nanti.

 <pre> 1 # Fungsi untuk memproses query akhir 2 def process_query_final(query_file, window_size, 3 step_size): 4 pitches, _, timings, tempo, ticks_per_beat = 5 extract_features_with_tempo(query_file) 6 segments = 7 segment_pitches_with_windowing_and_normalization(8 pitches, timings, window_size, step_size, 9 ticks_per_beat, tempo) 10 histograms = [extract_normalized_histograms(11 segment) for segment in segments if segment] 12 13 return histograms </pre>	<p>Fungsi ini digunakan untuk memproses file query dengan memanggil semua fungsi di atas. Proses yang dilakukan adalah ekstraksi file, windowing dan normalisasi, dan pembuatan histogram.</p>
 <pre> 1 # Fungsi untuk memproses dataset akhir 2 def process_dataset_final(midi_files, 3 window_size, step_size): 4 dataset_features = [] 5 failed_files = [] # List file gagal 6 for file in midi_files: 7 pitches, _, timings, tempo, 8 ticks_per_beat = extract_features_with_tempo(9 file) 10 if not pitches and not timings: 11 failed_files.append(file) 12 continue # Skip file 13 14 segments = 15 segment_pitches_with_windowing_and_normalization(16 pitches, timings, window_size, 17 step_size, ticks_per_beat, tempo 18) 19 20 histograms = [21 extract_normalized_histograms(segment) for 22 segment in segments if segment] 23 dataset_features.append((file, 24 histograms)) 25 26 if failed_files: 27 print(28 "Warning: Gagal untuk memproses file:") 29 for file in failed_files: 30 print(f" - {file}") 31 32 return dataset_features </pre>	<p>Fungsi ini sebenarnya sama dengan fungsi <code>process_query_final</code>. Perbedaanya adalah fungsi ini memproses semua file yang ada di dataset dan mengumpulkan semua datanya dalam sebuah array.</p>

```

1 # Menghitung persamaan antara dua histogram dengan
cosine similarity
2 def compute_similarity(hist1, hist2):
3     return 1 - cosine(hist1, hist2)

```

Fungsi ini digunakan untuk menghitung kemiripan sebuah histogram. Kami menggunakan `scipy.spatial.distance` yang menghasilkan cosine distance. Oleh karena itu, perlu 1 untuk mendapatkan cosine similarity.

```

1 # Fungsi untuk mengambil file yang mirip dengan
query
2 def retrieve_similar_files_with_rank(query_file,
dataset_features, weights, window_size,
step_size):
3     query_histograms = process_query_final(
query_file, window_size, step_size)
4
5     scores = []
6     for file, histograms_list in
dataset_features:
7         dataset_array = [h for h in
histograms_list if h]
8         query_array = [h for h in
query_histograms if h]
9
10    similarity_matrix = np.array([
11        [
12            sum(w * compute_similarity(q, d
) for q, d, w in zip(query_hist, data_hist,
weights))
13            for data_hist in dataset_array
14        ]
15            for query_hist in query_array
16        ])
17
18    max_similarity = np.max(
similarity_matrix) if similarity_matrix.size > 0
19    else 0
20    scores.append((file, max_similarity))
21
return sorted(scores, key=lambda x: x[1],
reverse=True)

```

Fungsi ini berguna untuk mendapatkan file pada dataset yang mirip dengan query. Proses yang dilakukan adalah membandingkan file query dan file-file pada dataset dan memasukkannya dalam sebuah matriks. Setelah itu, dicari nilai tertingginya dan disort dari file dengan nilai paling tinggi dan nilai paling rendah.

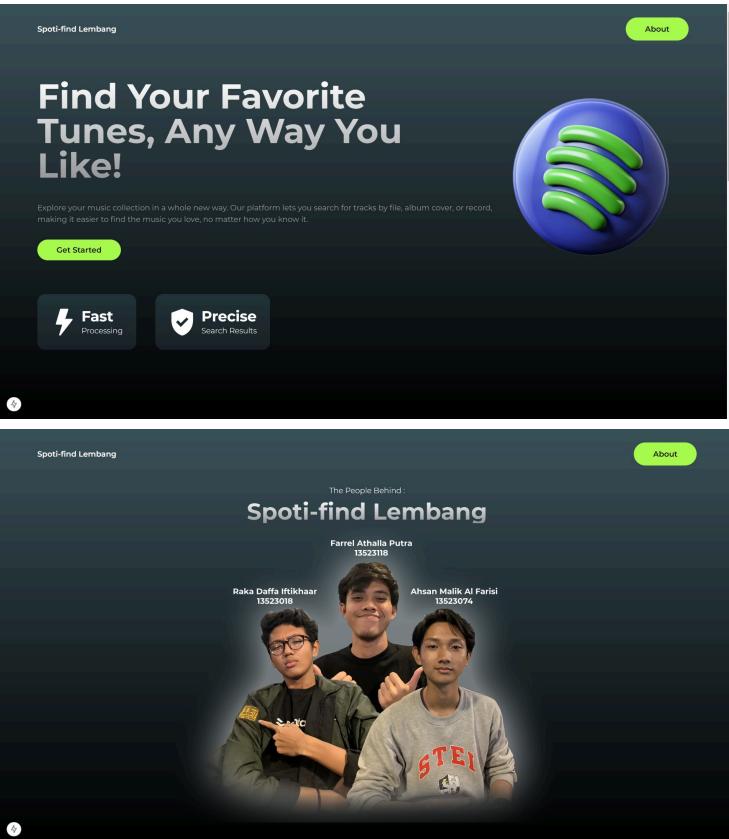
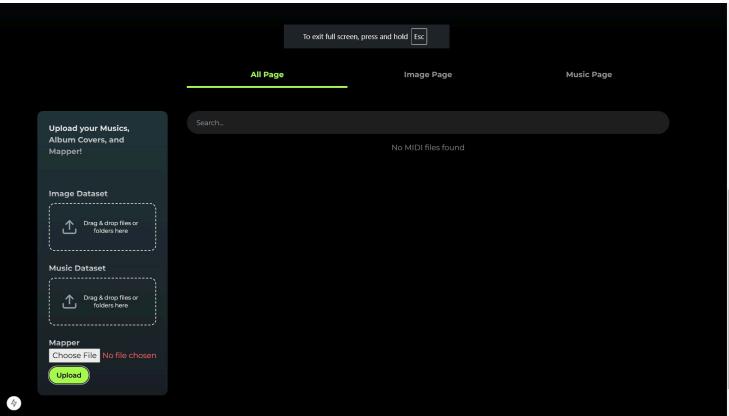
```
1 # Main program
2 def main():
3     # Mengatur direktori dataset dan file query
4     dataset_dir =
5         "D:/Algeo/Tubes 2/MIDI/midi_dataset/midi_database"
6     t"
7         query_file =
8         "D:/Algeo/Tubes 2/MIDI/QUERY/x(43)humming.mid"
9
10    # Parameter windowing dan normalisasi
11    window_size_beats = 20
12    # Window size: 20-40 beats
13    step_size_beats = 4
14    # Sliding window step: 4-8 beats
15
16    # Cek keberadaan file
17    if not os.path.isdir(dataset_dir):
18        print(f
19             "Error: Direktori dastae tidak ditemukan: {dataset_dir}")
20        return
21    if not os.path.isfile(query_file):
22        print(f
23             "Error: File query tidak ditemukan: {query_file}")
24        return
25
26    # Load dan proses dataset
27    print("Memproses dataset...")
28    midi_files = load_midi_files(dataset_dir)
29    dataset_features = process_dataset_final(
30        midi_files, window_size_beats, step_size_beats)
31
32    # Proses query
33    print(
34        "Membandingkan humming dengan dataset...")
35    try:
36        top_matches =
37            retrieve_similar_files_with_rank(query_file,
38                dataset_features, weights=[0.5, 0.3, 0.2],
39                window_size=window_size_beats, step_size
40                =step_size_beats)
41        # Menampilkan hasil pencarian
42        print("\nTop Matches (Ranked):")
43        if not top_matches:
44            print(
45                "Pencarian tidak ditemukan. Tolong periksa file
46                MIDI yang digunakan."
47            )
48        else:
49            for idx, (file, score) in enumerate(
50                top_matches):
51                similarity_percentage = score *
52                100
53                print(f"Rank {idx + 1}: {os.path
54                    .basename(file)} - Kemiripan: {
55                        similarity_percentage:.2f}%")
56    except Exception as e:
57        print(f"Error: {e}")
```

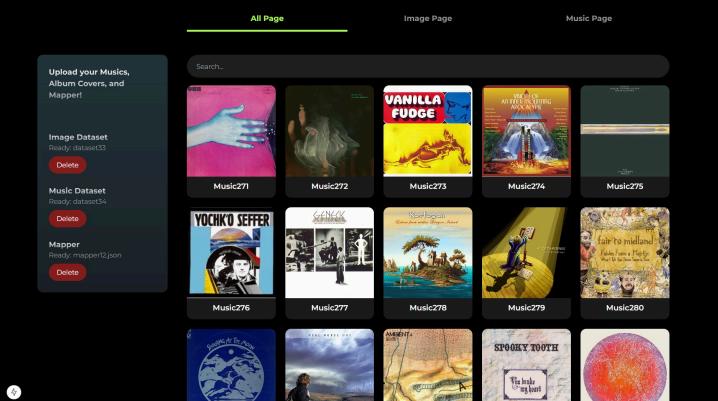
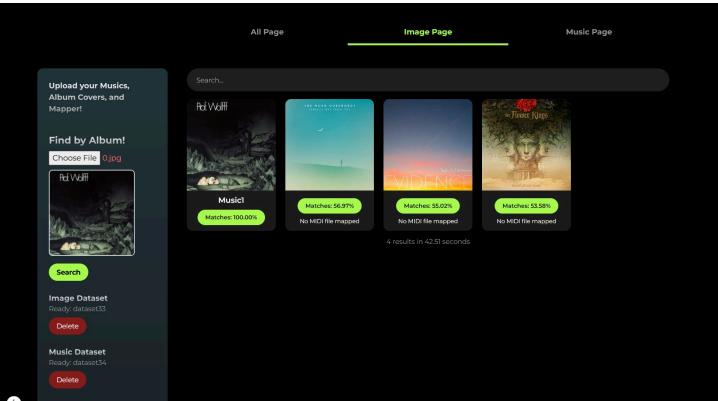
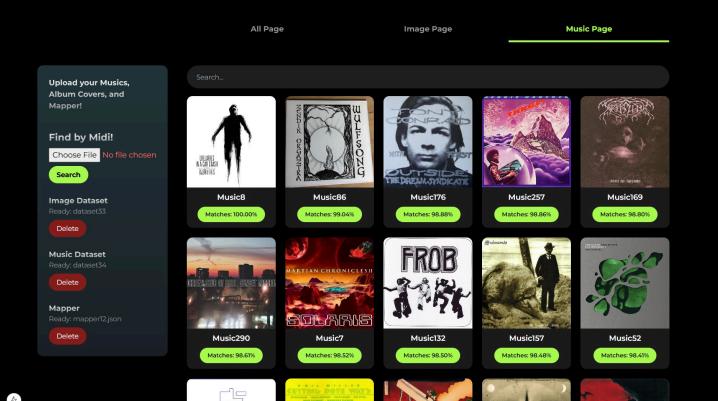
Fungsi ini adalah fungsi utama untuk menjalankan program. Pada fungsi ini, kita harus memilih file query dan folder dataset untuk dibandingkan. Kemudian kita juga bisa mengatur parameter *windowing* dimulai dari *size window* dan *sliding window*. Pada kode tersebut saya menggunakan 20 untuk *windowing* dan 4 untuk *sliding window*. Hal itu dikarenakan semakin kecil kedua hal tersebut akan menambah detail dari pengecekan kesamaan. Hasil dari fungsi ini adalah akan mengeluarkan hasil berupa beberapa if else statement dan yang utama akan menghasilkan ranking lagu paling mirip dalam bentuk persentase kemiripan.

BAB 4

EKSPERIMENT

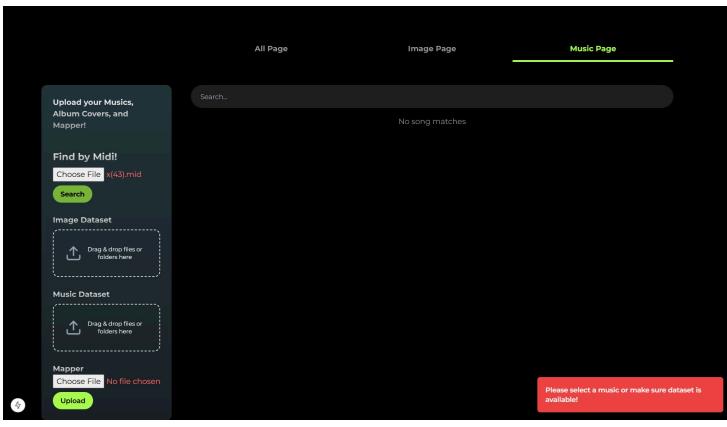
4.1 Tabel Eksperimen

No	Gambar	Deskripsi
1.		Website berjalan dengan baik.
2.		Tampilan dataset kosong.

3.		<p>Tampilan data set penuh dengan mapper.</p>
4.		<p>Mencari foto yang mirip di dalam data set.</p>
5.		<p>Mencari musik yang mirip di dalam data set.</p>

6.		<p>Mencari foto yang tidak mirip.</p>
7.		<p>Tidak ada mapper yang terpilih.</p>
8.		<p>Mencari image tanpa data set.</p>

9.



Mencari musik tanpa data set.

BAB 5

PENUTUP

5.1 Kesimpulan

Hasil algoritma dari Album Picture Finder dengan konsep PCA atau Principal Component Analysis yang merupakan konsep dari Aljabar Linier dan Geometri dapat menghasilkan kemiripan yang cukup baik karena sudah bisa mengeluarkan gambar yang sama persis dengan akurasi 100% dan gambar yang sama namun diberi perbedaan warna juga memiliki kemiripan kisaran 80%. Namun jika gambar dirotasi maka hasilnya akan jauh lebih kecil karena dalam pemrosesannya tidak dilakukan augmentasi dataset dengan rotasi, noise, dan compression. Walaupun sudah dapat memberikan gambar yang mirip dalam pencarinya namun proses komputasinya masih cukup lama karena harus melakukan banyak perhitungan seperti mengubah data menjadi matriks kovarians, perhitungan SVD, dan masih banyak lagi. Selain dari harus melakukan jumlah komputasi yang banyak pemilihan bahasa python juga memengaruhi kecepatan pencarian karena kurang low level dalam memanipulasi data walaupun sudah memakai library yang dioptimalisasi.

Hasil algoritma dari *Query by Humming* dapat mendeteksi kemiripan lagu yang sama dengan hasil 100% sama. Namun, karena sulitnya untuk mencari dataset dan memodifikasi audio yang ada, menyebabkan kami kurang yakin terhadap pengetesan yang membandingkan humming dengan lagu asli. Kami sudah mencoba beberapa cara untuk mengetes seperti menggunakan lagu yang dicover dan lagu original namun karena input file berupa file MIDI yang merubah semua instrumen dan nada menjadi piano membuat kami kurang mengetahui apakah output yang dihasilkan sudah sesuai dengan seharusnya atau belum. Walaupun demikian, setelah kami mencoba untuk mengetes sebuah file MIDI menggunakan dataset, kami mendapatkan bahwa file dengan kemiripan tinggi memiliki nada yang mirip dengan file asli yang menjadi query.

5.2 Saran

Dalam Album Picture Finder dengan PCA, keakuratannya dapat ditingkatkan lagi dengan melakukan augmentasi dataset seperti zoom in, zoom out, rotasi, dan pemberian noise. Dengan demikian gambar query yang dicari akan lebih mudah dideteksi dalam keadaan dunia nyata. Selain itu pemilihan bahasa juga harus diperhatikan agar dapat melakukan komputasi dan manipulasi data dengan lebih cepat, misalnya memakai bahasa Go atau bahasa C. Pencarian gambar pun dapat ditingkatkan lagi hasil kemiripan dan proses komputasinya jika menggunakan machine learning.

Saran lainnya adalah, sebaiknya dataset untuk MIDI diperbanyak dengan dataset khusus berisi file-file yang sudah dimodifikasi. Hal itu untuk mempermudah pengetesan apakah program kami sudah sesuai atau belum.

5.3 Komentar

Tubes ini lumayan seru karena mengeksplor bagaimana kita dapat mengaplikasikan teori dari materi Aljabar Linier dan Geometri dan dapat mengerti akan proses di dalamnya. Namun batasan akan penggunaan library dalam pengerjaannya tidak konsisten dan spesifikasinya terlalu umum atau interpretasi tiap orang dapat berbeda sehingga banyak menimbulkan kebingungan.

Aku cinta Deandaru!

-Raka Daffa Iftikhaar-

Aku cinta Naura!

-Ahsan Malik Al Farisi-

Aku cinta kau dan dia!

-Farrel Athalla Putra-

Thanks Beyonce!

-Tim LEMBANG-

5.4 Refleksi

Kami merasa sudah melakukan hal yang sesuai pada tugas besar kali ini. Kami sudah memulai dari jauh-jauh hari dan mempersiapkan segala sesuatunya. Namun, karena banyaknya beban akademik yang lain membuat ada beberapa hal yang *missed* pada tugas besar kali ini.

LAMPIRAN

Link Repository: bit.ly/SpotifindLembang

Link Video: <https://youtu.be/vPMpjDyUgFI>