4. IMPLEMENTASI SISTEM

Pada bab ini akan membahas mengenai implementasi dari desain sistem yang telah dibuat pada bab sebelumnya. Implementasi sistem berupa penggunaan perangkat lunak yang digunakan, library, syntax, proses *hashing* dan proses manipulasi gambar yang digunakan. Serta penerapan pembuatan sistem berupa website yang menerima input gambar dan dapat mengautentikasi hasil gambar dengan beberapa manipulasi yang tersedia. Tabel 4.1 merupakan list implementasi desain ke dalam segmen program.

Tabel 4.1

Implementasi Desain ke Dalam Segmen Program

Gambar Bab 3	Segmen Program	Keterangan
Gambar 3.1 , 3.2, 3.3, 3.4, 3.5, 3.15	4.1	Desain Sistem
Gambar 3.9 , 3.10, 3.11, 3.12, 3.13, 3.14	4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13,	Manipulasi Gambar
Gambar 3.6 , 3.16	4.14, 4.15	*Perceptual Hashing*
Gambar 3.7 , 3.17	4.16, 4.17	*Wavelet hashing*
Gambar 3.8 , 3.18	4.18, 4.19	*Average hashing*
Gambar 3.19, 3.20	4.20	Tampilan *Website*

4.1 Implementasi Perangkat Lunak yang Digunakan

Penerapan *perceptual hashing* pada sistem di skripsi ini outputnya berupa *website*. *Website* dibuat dengan bahasa pemrograman Python versi 3.11.9 dengan *library* yang digunakan untuk antarmuka dan penghubung Python dengan *website* autentikasi gambar yaitu Flask. Python juga digunakan dari proses manipulasi gambar hingga proses autentikasi pada gambar yang ada.

4.2 Library yang digunakan

Visual Studio Code adalah sebuah perangkat lunak yang digunakan sebagai IDLE untuk Python, dimana digunakan untuk mengedit teks pemrograman Python, melakukan proses *hashing*, manipulasi gambar, dan autentikasi gambar. Segmen program ini berujuan untuk

mengimpor library yang dibutuhkan sebagai penunjang seluruh proses pada program autentikasi gambar. Berikut adalah penggunaan library yang dibutuhkan pada Python.

Segmen Program 4.1 Instalasi *Library* pada Program

```
from flask import Flask, render_template, request
import cv2
import numpy as np
from scipy.fftpack import dct
import os
import pywt
import random
```

- *Library* flask adalah sebuah *library* untuk framework web mikro yang ditulis dalam Python. Digunakan di sini untuk membuat antarmuka web yang memungkinkan pengguna mengunggah gambar dan melakukan manipulasi serta autentikasi gambar.
- *Library* cv atau yang biasa dikenal OpenCV adalah *library* sumber terbuka yang populer untuk komputer vision dan pengolahan gambar. Digunakan di sini untuk membaca gambar yang diunggah, melakukan manipulasi seperti rotasi, *skewing*, dan kompresi gambar, serta untuk operasi pengolahan gambar lainnya.
- *Library* numpy adalah *library* Python yang kuat untuk komputasi numerik. Digunakan di sini untuk berbagai operasi matriks dan manipulasi gambar.
- *Library* scipy adalah *library* yang memperluas fungsionalitas NumPy dengan menyediakan *berbagai* algoritma ilmiah dan teknik numerik. Di sini, modul fftpack dari SciPy digunakan untuk transformasi kosinus diskrit (DCT) dalam penghitungan *hash*.
- *Library* os menyediakan fungsi-fungsi untuk berinteraksi dengan sistem operasi, seperti mengakses *file*, mengelola direktori, dan berbagai operasi lain terkait sistem *file*. Di program ini, *library* os digunakan untuk mengelola *path file*, menyimpan gambar yang dimanipulasi, dan mengatur *path* untuk menyimpan gambar.
- *Library* pywt atau Py*Wavelet*s adalah *library* Python untuk transformasi gelombang diskrit dan berbagai operasi pengolahan sinyal. Digunakan di sini untuk menghitung *hash* gelombang (*wavelet hash*) dari gambar.
- *Library* random digunakan untuk menghasilkan angka acak. Di program ini, digunakan untuk menghasilkan nilai acak yang digunakan dalam manipulasi gambar, seperti dalam fungsi *skew_image*_random() untuk menghasilkan nilai acak untuk skala *skew*.

4.3 Manipulasi Gambar

4.3.1 *Upscaling*

Segmen Program 4.2 *Input* Skala *Upscaling*

```
faktor_skala_up = float(request.form['faktor_skala_up'])
```

Kode float(request.form['faktor_skala_up']) digunakan untuk mengambil nilai faktor skala up yang dikirimkan oleh pengguna melalui formulir HTML. Nilai ini diambil dari data formulir yang dikirimkan melalui permintaan POST oleh pengguna saat mengirimkan formulir di aplikasi web. Fungsi float(...) digunakan untuk mengonversi nilai yang diambil (dalam bentuk string) menjadi tipe data float, sehingga nilai tersebut dapat digunakan dalam operasi matematis atau manipulasi gambar selanjutnya sesuai dengan kebutuhan yang diinginkan dalam aplikasi.

Segmen Program 4.3 Proses *Upscaling*

```
manipulated_images.append(cv2.resize(image1, None, fx=faktor_skala_up,
fy=faktor_skala_up, interpolation=cv2.INTER_AREA))
```

Baris kode manipulated_*image*s.append(cv2.re*size*(*image*1, None, fx=faktor_skala_up, fy=faktor_skala_up, interpolation=cv2.INTER_AREA)) digunakan untuk menghasilkan gambar baru yang telah diubah ukurannya berdasarkan faktor skala yang ditentukan oleh pengguna melalui formulir HTML. Proses ini dilakukan dengan menggunakan fungsi cv2.re*size*(...) dari OpenCV untuk mengubah ukuran gambar *image*1 dengan faktor skala yang diberikan.

Dalam fungsi cv2.re*size*(...), parameter fx=faktor_skala_up dan fy=faktor_skala_up menentukan faktor skala untuk sumbu horizontal (x) dan vertikal (y). Nilai faktor_skala_up adalah bilangan pecahan yang menentukan seberapa besar gambar akan diperbesar (jika nilainya lebih dari 1).

Interpolasi cv2.INTER_AREA digunakan untuk menentukan metode resizing gambar, di mana metode ini menghitung area piksel rata-rata berdasarkan area tetangga terdekat saat melakukan resizing. Hasil dari operasi cv2.re*size*(...) ini adalah gambar baru yang telah diubah ukurannya sesuai dengan faktor skala yang ditentukan, dan gambar ini kemudian ditambahkan ke dalam daftar manipulated_*image*s untuk digunakan dalam langkah manipulasi gambar selanjutnya dalam aplikasi.

4.3.2 *Downscaling*

Segmen Program 4.4 *Input* Skala *Downscaling*

```
faktor_skala_down = float(request.form['faktor_skala_down'])
```

Kode float(request.form['faktor_skala_down']) digunakan untuk mengambil nilai faktor skala down yang diinputkan oleh pengguna melalui formulir HTML. Proses ini terjadi saat pengguna mengirimkan formulir dengan metode POST, dan nilai dari field 'faktor_skala_down' pada formulir diakses menggunakan request.form['faktor_skala_down'] dalam Flask.

Nilai yang diambil dari formulir adalah dalam bentuk string, dan menggunakan fungsi float(...) untuk mengonversi nilai tersebut menjadi tipe data float. Dengan demikian, nilai float ini dapat digunakan dalam aplikasi untuk melakukan manipulasi gambar dengan faktor skala tertentu sesuai dengan input yang diberikan oleh pengguna melalui formulir.

Segmen Program 4.5 Proses *Downscaling*

```
manipulated_images.append(cv2.resize(image1, None,
fx=faktor_skala_down,
fy=faktor_skala_down,interpolation=cv2.INTER_AREA))
```

Baris kode manipulated_*image*s.append(cv2.re*size*(*image*1, None, fx=faktor_skala_down, fy=faktor_skala_down, interpolation=cv2.INTER_AREA)) digunakan untuk menghasilkan gambar baru yang telah diubah ukurannya berdasarkan faktor skala down yang ditentukan oleh pengguna melalui formulir HTML.

Dalam fungsi cv2.re*size*(...), parameter fx=faktor_skala_down dan fy=faktor_skala_down menentukan faktor skala untuk sumbu horizontal (x) dan vertikal (y) saat melakukan perubahan ukuran gambar. Nilai faktor_skala_down adalah bilangan pecahan yang menentukan seberapa besar gambar akan diperkecil (jika nilainya kurang dari 1).

Interpolasi cv2.INTER_AREA digunakan untuk menentukan metode resizing gambar, di mana metode ini menghitung area piksel rata-rata berdasarkan area tetangga terdekat saat melakukan resizing.

Hasil dari operasi cv2.re*size*(...) ini adalah gambar baru yang telah diubah ukurannya sesuai dengan faktor skala down yang ditentukan, dan gambar ini kemudian ditambahkan ke dalam daftar manipulated_*image*s untuk digunakan dalam langkah manipulasi gambar

selanjutnya dalam aplikasi. Dengan demikian, gambar akan diperkecil sesuai dengan faktor skala yang diinginkan oleh pengguna melalui formulir.

4.3.3 Perubahan Intensitas

Segmen Program 4.6 *Input* Skala Perubahan Intensitas

```
gamma = int(request.form['gamma'])
```

kode ini digunakan untuk mengambil nilai parameter gamma yang dikirimkan oleh pengguna melalui formulir HTML dalam aplikasi web, dan kemudian mengonversinya menjadi tipe data integer (int).

Proses ini terjadi saat pengguna mengirimkan formulir dengan metode POST, dan nilai dari field dengan nama 'gamma' pada formulir diakses menggunakan request.form['gamma'] dalam Flask. Data yang diambil awalnya dalam bentuk string karena data formulir biasanya dikirimkan dalam bentuk teks.

Dengan menggunakan int(...), nilai string yang diambil dari formulir diubah menjadi bilangan bulat (integer). Hal ini dilakukan agar nilai gamma dapat digunakan secara numerik dalam operasi matematis atau manipulasi gambar selanjutnya dalam aplikasi.

Hasil dari kode ini adalah variabel gamma yang berisi nilai parameter gamma dalam bentuk bilangan bulat (integer), yang kemudian dapat digunakan dalam manipulasi gambar, seperti penyesuaian kecerahan atau kontras gambar berdasarkan nilai gamma yang diinputkan oleh pengguna melalui formulir.

Segmen Program 4.7 Proses Perubahan Intensitas

```
manipulated_images.append(np.clip(image1 + gamma, 0, 255))
```

Kode manipulated_*image*s.append(np.clip(*image*1 + gamma, 0, 255)) digunakan dalam aplikasi untuk menambahkan gambar yang telah dimanipulasi ke dalam daftar manipulated_*image*s. Prosesnya dimulai dengan menambahkan nilai gamma (yang merupakan bilangan bulat) ke setiap piksel dalam gambar *image*1. Operasi ini dapat mengubah kecerahan (brightness) gambar, di mana nilai gamma positif akan meningkatkan kecerahan dan nilai gamma negatif akan mengurangi kecerahan.

Selanjutnya, fungsi np.clip(...) dari NumPy digunakan untuk membatasi nilai piksel agar tetap berada dalam rentang yang valid untuk gambar, yaitu antara 0 dan 255. Hal ini penting karena hasil penambahan gamma bisa menyebabkan nilai piksel melebihi rentang yang valid,

dan np.clip(...) memastikan bahwa nilai piksel yang dihasilkan tetap dalam rentang yang diharapkan.

Hasil dari operasi tersebut adalah gambar yang telah dimanipulasi dengan penyesuaian kecerahan sesuai dengan nilai gamma yang diinputkan oleh pengguna melalui formulir HTML. Gambar yang dimanipulasi ini kemudian ditambahkan ke dalam daftar manipulated_*image*s untuk digunakan dalam langkah manipulasi gambar selanjutnya dalam aplikasi. Dengan demikian, pengguna dapat melihat efek dari penyesuaian kecerahan yang telah diterapkan pada gambar melalui aplikasi web tersebut.

4.3.4 Rotasi Gambar

Segmen Program 4.8 *Input* Skala Rotasi

```
rotasi_degrees = int(request.form['rotasi']) if 'rotasi' in request.form else 0
rotated_image = rotate_image(image1, rotasi_degrees)
manipulated_images.append(rotated_image)
```

Kode ini digunakan dalam aplikasi untuk melakukan rotasi terhadap gambar *image*1 berdasarkan nilai derajat rotasi yang diberikan oleh pengguna melalui formulir HTML. Mari kita bahas langkahnya secara detail:

- rotasi_degrees = int(request.form['rotasi']) if 'rotasi' in request.form else 0: Baris ini bertujuan untuk mengambil nilai derajat rotasi (rotasi) yang dikirimkan oleh pengguna melalui formulir HTML. Jika parameter rotasi ada dalam request.form (yaitu, pengguna telah mengisi nilai rotasi dalam formulir), nilai rotasi akan diambil dan diubah menjadi tipe data integer menggunakan int(...). Jika parameter rotasi tidak ada dalam request.form (*art*inya, pengguna tidak mengisi nilai rotasi), maka nilai rotasi_degrees akan diatur sebagai 0 secara default.
- rotate_*image*(*image*1, rotasi_degrees): Setelah mendapatkan nilai rotasi_degrees, fungsi rotate_*image*(...) dipanggil dengan gambar *image*1 dan nilai rotasi_degrees sebagai parameter. Fungsi ini bertugas untuk melakukan rotasi terhadap gambar berdasarkan nilai derajat rotasi yang diberikan.
- manipulated_*image*s.append(rotated_*image*): Hasil dari operasi rotasi (rotated_*image*), yaitu gambar yang telah dirotasi berdasarkan nilai derajat rotasi yang diinputkan oleh pengguna, kemudian ditambahkan ke dalam daftar

manipulated_*image*s. Gambar yang telah dimanipulasi dengan rotasi ini akan digunakan dalam langkah manipulasi gambar selanjutnya dalam aplikasi.

Dengan demikian, pengguna dapat mengatur nilai derajat rotasi pada gambar melalui formulir HTML, dan aplikasi akan melakukan rotasi pada gambar sesuai dengan nilai tersebut. Hasil rotasi ini kemudian dapat dilihat pada gambar yang dimanipulasi dalam aplikasi web tersebut. Jika nilai rotasi tidak diisi oleh pengguna, maka gambar akan tetap dalam posisi aslinya (tanpa rotasi).

Segmen Program 4.9 Proses Rotasi

```
def rotate_image(image, angle):
    center = tuple(np.array(image.shape[1::-1]) / 2)
    rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated_image = cv2.warpAffine(image, rotation_matrix,
image.shape[1::-1], flags=cv2.INTER_LINEAR)
    return rotated_image
```

Fungsi rotate_*image*(*image*, angle) digunakan untuk melakukan rotasi terhadap gambar *image* sebesar angle derajat menggunakan OpenCV. Prosesnya dimulai dengan menghitung titik tengah (center) dari gambar menggunakan dimensi (lebar dan tinggi) gambar tersebut. Selanjutnya, fungsi cv2.get*Rotation*Matrix2D(center, angle, 1.0) digunakan untuk menghasilkan matriks rotasi berdasarkan titik tengah rotasi, derajat rotasi yang diinginkan (angle), dan faktor skala untuk mempertahankan ukuran gambar. Matriks rotasi ini kemudian digunakan dalam fungsi cv2.warpAffine(*image*, *rotation*_matrix, *image*.shape[1::-1], flags=cv2.INTER_LINEAR) untuk menerapkan transformasi rotasi pada gambar. Hasil rotasi gambar (rotated_*image*) kemudian dikembalikan sebagai output dari fungsi rotate_*image*. Dengan menggunakan fungsi ini, pengguna dapat melakukan rotasi gambar secara fleksibel berdasarkan nilai derajat yang diinginkan, dan hasilnya dapat digunakan dalam aplikasi pengolahan gambar untuk berbagai keperluan seperti visualisasi data atau pengolahan citra.

4.3.5 *Skew* Gambar

Segmen Program 4.10 *Input* Skala *Skew*

```
skewed_image = skew_image_random(image1)
manipulated_images.append(skewed_image)
```

Kode *skew*ed_*image* = *skew_image*_random(*image*1) digunakan untuk memanggil fungsi *skew_image*_random(*image*1), yang bertujuan untuk menghasilkan gambar yang telah mengalami distorsi acak (*skew*) berdasarkan gambar *image*1. Hasil dari operasi distorsi acak ini disimpan dalam variabel *skew*ed_*image*.

Setelah mendapatkan gambar yang telah dimanipulasi (distorsi acak), langkah selanjutnya adalah menyimpan gambar tersebut ke dalam daftar manipulated_*image*s menggunakan manipulated_*image*s.append(*skew*ed_*image*). Dengan demikian, gambar yang telah dimanipulasi dengan distorsi acak ini akan tersedia dalam daftar manipulated_*image*s untuk digunakan dalam langkah-langkah manipulasi gambar selanjutnya dalam aplikasi.

Proses ini secara dasar hanya melibatkan pemanggilan fungsi *skew_image*_random(*image*1) untuk menghasilkan gambar yang terdistorsi, dan kemudian menyimpan hasilnya dalam daftar manipulated_*image*s. Dengan menggunakan pendekatan ini, pengguna dapat mengaplikasikan berbagai transformasi gambar dalam aplikasi dengan cara yang mudah dan efisien.

Segmen Program 4.11 Proses *Skew*

```
def skew_image_random(image):
    rows, cols, _ = image.shape
    skew_scale = random.uniform(-1, 1)  # Menghasilkan nilai acak antara
-1 dan 1 untuk skala skew
    skew_matrix = np.float32([[1, skew_scale, 0], [0, 1, 0]])
    skewed_image = cv2.warpAffine(image, skew_matrix, (cols, rows),
borderMode=cv2.BORDER_CONSTANT, borderValue=(255, 255, 255))
    return skewed_image
```

Fungsi *skew_image*_random(*image*) digunakan untuk menerapkan distorsi acak (*skew*) pada gambar *image* menggunakan transformasi affine. Prosesnya dimulai dengan mengambil dimensi gambar (rows, cols, _) dari *image*.shape, yang merupakan tinggi (rows) dan lebar (cols) gambar.

Selanjutnya, fungsi ini menghasilkan nilai acak untuk *skew*_scale menggunakan random.uniform(-1, 1). Nilai *skew*_scale ini akan menentukan seberapa besar distorsi akan diterapkan pada gambar. Jika nilai *skew*_scale positif, gambar akan terdistorsi ke arah kanan, sedangkan jika negatif, gambar akan terdistorsi ke arah kiri.

Setelah mendapatkan nilai *skew*_scale, fungsi ini membuat matriks transformasi affine (*skew*_matrix) dengan menggunakan np.float32([[1, *skew*_scale, 0], [0, 1, 0]]). Matriks ini memungkinkan penerapan transformasi *skew* pada gambar.

Proses distorsi kemudian diterapkan pada gambar *image* menggunakan fungsi cv2.warpAffine(*image*, *skew*_matrix, (cols, rows), borderMode=cv2.BORDER_CONSTANT, borderValue=(255, 255, 255)). Parameter (cols, rows) digunakan untuk menentukan ukuran gambar hasil distorsi yang sama dengan gambar asli. Pengaturan borderMode=cv2.BORDER_CONSTANT dan borderValue=(255, 255, 255) digunakan untuk menambahkan border pada gambar hasil distorsi dengan warna putih.

Hasil dari proses ini adalah gambar yang telah mengalami distorsi acak (*skew*) berdasarkan nilai *skew*_scale, yang kemudian dikembalikan sebagai output fungsi *skew_image*_random(*image*). Dengan demikian, fungsi ini dapat digunakan untuk menghasilkan variasi gambar dengan efek distorsi yang berbeda dalam aplikasi pengolahan gambar.

4.3.6 Kompresi Ukuran Gambar

Segmen Program 4.12 *Input* Skala Kompresi

```
compress_scale = int(request.form['compress_scale'])
```

Kode *compress*_scale = int(request.form['*compress*_scale']) digunakan untuk mengambil nilai skala kompresi (*compress* scale) dari formulir HTML yang dikirimkan oleh pengguna. Nilai ini diubah menjadi tipe data integer sehingga dapat digunakan sebagai parameter dalam proses kompresi gambar selanjutnya dalam aplikasi Flask. Dengan menggunakan nilai *compress*_scale yang diambil dari formulir, pengguna dapat mengatur sejauh mana gambar akan dikompresi sebelum disimpan atau ditampilkan kembali dalam aplikasi.

Segmen Program 4.13 Proses Kompresi

```
compress_filepath = os.path.join(app.config['UPLOAD_FOLDER'],
'compressed_image.jpg')
```

```
cv2.imwrite(compress_filepath, image1, [cv2.IMWRITE_JPEG_QUALITY,
compress_scale])
manipulated_images.append(cv2.imread(compress_filepath))
```

Kode tersebut digunakan dalam aplikasi Flask untuk mengompresi gambar *image*1 berdasarkan nilai skala kompresi (*compress*_scale) yang telah diambil dari formulir HTML. Pertama, baris pertama *compress_file*path = os.path.join(app.config['UPLOAD_FOLDER'], '*compress*ed_*image*.jpg') digunakan untuk menentukan jalur penyimpanan (path) untuk gambar hasil kompresi. Jalur ini dibuat dengan menggabungkan direktori penyimpanan yang telah ditentukan dalam konfigurasi aplikasi (app.config['UPLOAD_FOLDER']) dengan nama *file* '*compress*ed_*image*.jpg'.

Kemudian, baris kedua cv2.imwrite(*compress_file*path, *image*1, [cv2.IMWRITE_JPEG_QUALITY, *compress*_scale]) bertanggung jawab untuk melakukan kompresi gambar *image*1 ke dalam format JPEG dengan kualitas yang ditentukan oleh *compress*_scale. Fungsi cv2.imwrite() akan menyimpan gambar yang telah dikompresi ke dalam *file* yang telah ditentukan (*compress_file*path).

Setelah proses kompresi selesai, baris terakhir manipulated_*image*s.append(cv2.imread(*compress_file*path)) digunakan untuk membaca kembali gambar yang telah dikompresi dari *file* yang disimpan tadi dan menambahkannya ke dalam daftar manipulated_*image*s. Dengan demikian, gambar hasil kompresi ini siap digunakan dalam langkah-langkah manipulasi gambar selanjutnya dalam aplikasi.

Dengan menggunakan pendekatan ini, pengguna dapat melihat efek kompresi pada gambar dan menghasilkan gambar yang lebih kecil dalam ukuran *file* berdasarkan nilai skala kompresi yang mereka atur melalui formulir HTML. Proses ini memungkinkan pengguna untuk mengontrol kualitas dan ukuran gambar yang dihasilkan sesuai dengan kebutuhan aplikasi pengolahan gambar yang dibangun menggunakan Flask.

4.4 *Hashing*

4.4.1 *Perceptual hashing*

Segmen Program 4.14 Proses *Perceptual Hashing*

```
def dct_hash(image):
    resized_image = cv2.resize(image, (32, 32),
interpolation=cv2.INTER_AREA)
```

```
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    dct_result = dct(dct(gray_image, axis=0, norm='ortho'), axis=1,
norm='ortho')
    dct_low_freq = dct_result[1:9, 1:9]
    mean_val = np.mean(dct_low_freq)
    hash_code = ""
    for i in range(8):
        for j in range(8):
            hash_code += '1' if dct_low_freq[i, j] > mean_val else '0'

    print("dct hash:\n", hash_code)
    return hash_code
```

Kode di atas merupakan implementasi dari algoritma pembuatan *hash* gambar menggunakan Discrete Cosine Transform (DCT). Fungsi dct_*hash*(*image*) pertama-tama menerima sebuah gambar sebagai input. Gambar tersebut kemudian diubah ukurannya menjadi 32x32 piksel menggunakan metode interpolasi cv2.INTER_AREA. Selanjutnya, gambar yang telah diubah ukurannya diubah menjadi gambar skala abu-abu. Setelah itu, transformasi DCT diterapkan dua kali pada gambar skala abu-abu, pertama pada sumbu vertikal dan kemudian pada sumbu horizontal, dengan normalisasi 'ortho'. Hasil dari transformasi DCT ini disimpan dalam variabel dct_result.

Bagian penting dari hasil DCT yang digunakan adalah frekuensi rendah, yang diambil dari indeks 1 hingga 8 pada kedua dimensi, dan disimpan dalam dct_low_freq. Nilai rata-rata dari elemen-elemen dalam dct_low_freq dihitung dan disimpan dalam mean_val. Untuk membentuk kode *hash*, setiap elemen dalam dct_low_freq dibandingkan dengan mean_val; jika elemen tersebut lebih besar dari nilai rata-rata, '1' ditambahkan ke string *hash*_code, sebaliknya '0' ditambahkan. *Hash* yang dihasilkan adalah string biner sepanjang 64 karakter yang mewakili karakteristik unik dari gambar tersebut. *Hash* ini kemudian dicetak dan dikembalikan sebagai output fungsi.

Segmen Program 4.15 Proses Autentikasi *Perceptual Hashing*

```
def authenticate_image(image1, image2, threshold):
    hash1 = dct_hash(image1)
    hash2 = dct_hash(image2)
    distance = hamming_distance(hash1, hash2)
```

```
    if distance <= threshold:
        return True
    else:
        return False
```

kode ini digunakan untuk membandingkan dua gambar *image*1 dan *image*2 berdasarkan *hash* code yang dihasilkan menggunakan transformasi Discrete Cosine Transform (DCT). Prosesnya melibatkan perhitungan jarak Hamming antara *hash* code dari kedua gambar. Jarak Hamming mengukur seberapa mirip *hash* code tersebut; semakin kecil jarak Hamming, semakin mirip kedua gambar tersebut berdasarkan representasi DCT.

Hasil perbandingan jarak Hamming kemudian dibandingkan dengan nilai ambang batas (*threshold*). Jika jarak Hamming antara kedua *hash* code kurang dari atau sama dengan nilai ambang batas, fungsi akan mengembalikan True, menunjukkan adanya kemiripan yang signifikan antara gambar. Sebaliknya, jika jarak Hamming melebihi nilai ambang batas, fungsi akan mengembalikan False. Dengan demikian, fungsi ini menyediakan cara yang efisien untuk menentukan kesamaan visual antara dua gambar berdasarkan representasi *hash* dari transformasi DCT, yang bermanfaat dalam aplikasi autentikasi atau pembandingan gambar.

4.4.2 *Wavelet hashing*

Segmen Program 4.16 Proses *Wavelet hashing*

```
def wavelet_hash(image, hash_size=8):
    target_size = (hash_size * 4, hash_size * 4)
    resized_image = cv2.resize(image, target_size,
interpolation=cv2.INTER_AREA)
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    coeffs = pywt.dwt2(gray_image, 'haar')
    LL, (LH, HL, HH) = coeffs
    mean_LH = np.mean(LH)
    quantized_LH = (LH > mean_LH).astype(int)
    hash_code = ""
    for i in range(hash_size):
        for j in range(hash_size):
            hash_code += "1" if quantized_LH[i, j] > 0 else "0"
    assert len(hash_code) == 64
```

```
    print ("wav \n",hash_code)
    return hash_code
```

Kode di atas adalah implementasi dari algoritma pembuatan *hash* gambar menggunakan metode *wavelet* (transformasi gelombang). Fungsi *wavelet_hash*(*image*, *hash_size*=8) menerima sebuah gambar sebagai input dan secara opsional bisa menerima ukuran *hash* yang diinginkan (default adalah 8). Pertama, gambar diubah ukurannya menjadi dua kali lipat dari ukuran *hash* yang diinginkan (misalnya, jika *hash_size*=8, maka ukuran gambar target menjadi 16x16 piksel). Proses ini dilakukan menggunakan metode interpolasi cv2.INTER_AREA. Selanjutnya, gambar yang telah diubah ukurannya diubah menjadi gambar skala abu-abu dengan mengonversi warna BGR menjadi skala abu-abu menggunakan cv2.cvtColor. Kemudian, dilakukan transformasi *wavelet* menggunakan metode 'haar' ('pywt.dwt2(gray_*image*, 'haar')). Hasil transformasi *wavelet* ini akan menghasilkan koefisien LL (Aproksimasi), LH (Detail baris), HL (Detail kolom), dan HH (Detail sudut) dari gambar. Dalam kasus ini, kita tertarik dengan koefisien LH (Detail baris). Nilai rata-rata dari koefisien LH dihitung dan digunakan sebagai ambang batas. Nilai koefisien LH kemudian di-quantize (dibuat biner) berdasarkan ambang batas tersebut, di mana nilai yang lebih besar dari ambang batas akan dianggap sebagai '1', dan nilai yang lebih kecil atau sama dengan ambang batas akan dianggap sebagai '0'. Selanjutnya, string *hash* dihasilkan dengan meninjau nilai quantized dari koefisien LH. Hasilnya adalah string biner sepanjang 64 karakter yang mewakili karakteristik gambar tersebut. Pada akhirnya, *hash* yang dihasilkan dicetak dan dikembalikan sebagai output fungsi. Pernyataan assert len(*hash*_code) == 64 digunakan untuk memastikan panjang *hash* yang dihasilkan sesuai dengan ekspektasi (64 karakter). Penjelasan ini membantu dalam memahami proses transformasi *wavelet* untuk pembuatan *hash* gambar dan bagaimana karakteristik gambar direpresentasikan secara biner dalam *hash* yang dihasilkan.

Segmen Program 4.17 Proses Autentikasi *Wavelet hashing*

```
def authenticate_wavelet_hash(image1, image2, threshold):
    hash1 = wavelet_hash(image1)
    hash2 = wavelet_hash(image2)
    distance = hamming_distance(hash1, hash2)
    if distance <= threshold:
        return True
    else:
```

```
        return False
```

Fungsi kode ini adalah membandingkan kesamaan antara dua gambar menggunakan *hash* code berbasis *wavelet*. *Hash* code ini didasarkan pada koefisien LH dari transformasi *wavelet*. Fungsi menghitung jarak Hamming antara kedua *hash* code untuk menentukan seberapa mirip kedua gambar tersebut. Hasilnya dibandingkan dengan *threshold*, dan fungsi mengembalikan True jika kedua gambar dianggap mirip (jarak Hamming <= *threshold*), dan False jika tidak.

4.4.3 *Average hashing*

Segmen Program 4.18 Proses *Average hashing*

```
def average_hash(image, hash_size=8):
    resized_image = cv2.resize(image, (hash_size, hash_size), 
interpolation=cv2.INTER_AREA)
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    average_value = np.mean(gray_image)
    hash_code = ""
    for i in range(hash_size):
        for j in range(hash_size):
            hash_code += "1" if gray_image[i, j] > average_value else 
"0"
    return hash_code
```

Fungsi ini adalah bagian dari sistem *hashing* yang digunakan untuk menghasilkan representasi singkat (*hash* code) dari suatu gambar *image*. Proses dimulai dengan mengubah ukuran gambar *image* menjadi *hash_size* x *hash_size* menggunakan fungsi cv2.re*size* dengan mode interpolasi cv2.INTER_AREA. Langkah ini membantu menghasilkan gambar dengan ukuran tetap dan konsisten untuk proses ekstraksi fitur *hashing*.

Selanjutnya, gambar yang sudah diubah ukurannya dikonversi ke citra skala keabuan (grayscale) menggunakan cv2.cvtColor(re*size*d_*image*, cv2.COLOR_BGR2GRAY). Proses konversi ini mengubah gambar menjadi representasi intensitas piksel tunggal, yang lebih mudah diolah untuk pembuatan *hash* code.

Setelah itu, nilai rata-rata intensitas piksel dari citra skala keabuan dihitung menggunakan np.mean(gray_*image*). Nilai rata-rata ini digunakan sebagai ambang batas untuk membangun *hash* code.

Pembentukan *hash* code (*hash*_code) dilakukan dengan membandingkan intensitas piksel dari setiap blok *hash_size* x *hash_size* dengan nilai rata-rata (*average*_value). Pada setiap iterasi, jika intensitas piksel pada blok melebihi nilai rata-rata, maka bit '1' akan ditambahkan ke dalam *hash*_code; sebaliknya, jika tidak, bit '0' akan ditambahkan.

Hasil dari proses ini adalah *hash* code yang merepresentasikan gambar berdasarkan nilai rata-rata intensitas piksel dalam blok-blok yang telah ditentukan. *Hash* code ini dapat digunakan untuk membandingkan kemiripan visual antara gambar dengan menggunakan algoritma *hashing* berbasis nilai rata-rata. Fungsi *average_hash* ini memberikan representasi singkat dari gambar dalam bentuk *hash* code untuk keperluan autentikasi atau pembandingan antar-gambar.

Segmen Program 4.19 Proses Autentikasi *Average hashing*

```
def authenticate_average_hash(image1, image2, threshold):
    hash1 = average_hash(image1)
    hash2 = average_hash(image2)
    distance = hamming_distance(hash1, hash2)
    if distance <= threshold:
        return True
    else:
        return False
```

Fungsi ini digunakan untuk membandingkan dua gambar *image*1 dan *image*2 berdasarkan *hash* code rata-rata yang dihasilkan menggunakan fungsi *average_hash*. Prosesnya melibatkan perhitungan *hash* code untuk kedua gambar, diikuti dengan penghitungan jarak (distance) Hamming antara *hash* code tersebut. Jika jarak Hamming kurang dari atau sama dengan ambang batas (*threshold*), maka fungsi mengembalikan True, menunjukkan bahwa kedua gambar memiliki kemiripan berdasarkan representasi *hash* rata-rata. Ini memberikan cara cepat untuk memeriksa kemiripan visual antara dua gambar menggunakan fitur *hashing* yang sederhana dan efisien.

4.5 *Website*

Segmen Program 4.20 HTML *Website*

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Authentication</title>
    <style>
        /* Gaya navbar */
        .navbar {
            overflow: hidden;
            background-color: #333;
            padding: 10px;
        }

        .navbar a {
            float: left;
            display: block;
            color: #f2f2f2;
            text-align: center;
            padding: 14px 20px;
            text-decoration: none;
            font-size: 20px;
        }

        .navbar a:hover {
            background-color: #45a049; /* Aksen hijau saat mouse hover */
            color: white;
        }

        .navbar a.active {
            background-color: #4CAF50;
            color: white;
        }

        /* Gaya lainnya */
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
            color: #333;
        }
        h1, h2 {
            margin-bottom: 20px;
        }
        form {
            margin-bottom: 30px;
        }
        input[type="text"], input[type="file"], input[type="submit"] {
            margin-bottom: 10px;
```

```
            padding: 8px;
            border: 1px solid #ccc;
            border-radius: 4px;
            width: 100%;
            box-sizing: border-box;
        }
        input[type="submit"] {
            background-color: #4CAF50;
            color: white;
            border: none;
            cursor: pointer;
        }
        input[type="submit"]:hover {
            background-color: #45a049;
        }
        ul {
            list-style-type: none;
            padding: 0;
        }
        li {
            margin-bottom: 10px;
        }
        img {
            max-width: 100%;
            height: auto;
            margin-bottom: 20px;
            display: block;
        }
    </style>
</head>
<body>
    <!-- Navbar -->
    <div class="navbar">
        <a href="#home">Image Authentication</a>
    </div>

    <!-- Konten -->
    <div style="max-width: 800px; margin: 0 auto; padding: 20px;">
        <h1>Image Authentication</h1>
        <form method="POST" enctype="multipart/form-data">
            <input type="file" name="file">
            <input type="text" name="faktor_skala_up" placeholder="Faktor Skala Up">
            <input type="text" name="faktor_skala_down" placeholder="Faktor Skala Down">
            <input type="text" name="gamma" placeholder="Gamma">
            <input type="text" name="rotasi" placeholder="Derajat Rotasi">
            <input type="text" name="compress_scale" placeholder="Skala Kompresi (0-100)">
            <input type="submit" value="Upload">
        </form>
```

```
        <!-- Pesan -->
        {% if message %}
        <p>{{ message }}</p>
        {% endif %}
        <!-- Hasil Autentikasi -->
        {% if results_dct %}
        <h2>Hasil Autentikasi (Perceptual Hash):</h2>
        <ul>
            {% for result in results_dct %}
            <li>{{ "Gambar terautentikasi." if result else "Gambar tidak terautentikasi." }}</li>
            {% endfor %}
        </ul>
        {% endif %}
        {% if results_wavelet %}
        <h2>Hasil Autentikasi (Wavelet Hash):</h2>
        <ul>
            {% for result in results_wavelet %}
            <li>{{ "Gambar terautentikasi." if result else "Gambar tidak terautentikasi." }}</li>
            {% endfor %}
        </ul>
        {% endif %}
        {% if results_average %}
        <h2>Hasil Autentikasi (Average Hash):</h2>
        <ul>
            {% for result in results_average %}
            <li>{{ "Gambar terautentikasi." if result else "Gambar tidak terautentikasi." }}</li>
            {% endfor %}
        </ul>
        {% endif %}
        <!-- Gambar Asli -->
        {% if original_image %}
        <h2>Gambar Asli:</h2>
        <img src="{{ original_image }}" alt="Original Image">
        {% endif %}
        <!-- Gambar yang Dimanipulasi -->
        {% if manipulated_image_paths %}
        <h2>Gambar yang Dimanipulasi:</h2>
        {% for manipulated_image_path in manipulated_image_paths %}
        <img src="{{ manipulated_image_path }}" alt="Manipulated Image">
        {% endfor %}
        {% endif %}
    </div>
</body>
</html>
```

Kode HTML di atas adalah bagian dari halaman web yang digunakan untuk melakukan autentikasi gambar. Halaman ini memiliki struktur dasar HTML dengan beberapa

elemen gaya CSS yang mengatur tampilan halaman. Terdapat sebuah navbar yang berisi judul "*Image Authentication*" untuk navigasi. Bagian konten utama terdiri dari sebuah form yang digunakan untuk mengunggah gambar dan parameter pengaturan. Form ini memungkinkan pengguna untuk memilih sebuah *file* gambar dan mengatur beberapa parameter seperti faktor skala, gamma, derajat rotasi, dan skala kompresi.

Setelah form diisi dan *file* gambar diunggah, hasil autentikasi gambar akan ditampilkan di bawahnya. Hasil autentikasi ini dikelompokkan berdasarkan tiga metode *hash* yang berbeda: *Perceptual Hash* (DCT), *Wavelet Hash*, dan *Average Hash*. Setiap hasil autentikasi akan menunjukkan apakah gambar terautentikasi atau tidak berdasarkan perbandingan dengan gambar asli.

Selain itu, halaman juga menampilkan gambar asli yang diunggah serta gambar-gambar yang telah dimanipulasi sesuai dengan parameter yang diatur. Gambar-gambar ini ditampilkan dengan menggunakan tag <img> yang menunjukkan gambar-gambar tersebut dalam halaman web. Dengan demikian, halaman ini memberikan antarmuka untuk mengunggah gambar, melakukan manipulasi gambar dengan parameter tertentu, dan melihat hasil autentikasi berdasarkan metode *hash* yang berbeda.