LAPORAN TUGAS PROJECT EXPERT SYSTEM

"Implementasi Content-Based Image Retrieval (CBIR) Berbasis Patch pada Citra Tekstur Berwarna Menggunakan Ekstraksi Fitur Tekstur dan Warna"

Dosen Pengampu:

Heri Prasetyo, S.Kom., M.Sc.Eng., Ph.D.

Disusun Oleh:

Muhammad Rais Sidiq	M0521055
Mochamad Faisal Akbar	L0122094

PROGRAM STUDI S1-INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA

UNIVERSITAS SEBELAS MARET

2024

Daftar Isi

BAB I
PENDAHULUAN

1.1. Latar Belakang

Pengelolaan dan pencarian informasi visual dalam bentuk citra digital menjadi salah satu tantangan utama dalam bidang pengolahan citra dan computer vision. Seiring bertambahnya volume data citra yang tersedia secara digital, kebutuhan akan sistem pencarian gambar yang cepat dan akurat pun semakin meningkat. Dalam hal ini, sistem Content-Based Image Retrieval (CBIR) hadir sebagai solusi dengan melakukan pencarian berdasarkan fitur visual gambar itu sendiri, bukan hanya berdasarkan teks atau metadata yang menyertainya. Pada awalnya, CBIR dikembangkan sebagai solusi untuk menangani masalah terkait meningkatnya jumlah koleksi gambar dalam jumlah besar (Siantar et al., 2019). Selama beberapa dekade terakhir, CBIR telah menjadi fokus utama dalam berbagai penelitian. CBIR adalah aplikasi berbasis komputer yang memungkinkan pencarian gambar digital yang memiliki kemiripan isi visual (Hidayat et al., 2017). Tidak semua gambar memiliki anotasi, dan anotasi pun belum tentu mampu merepresentasikan gambar secara tepat. Oleh karena itu, pencarian berdasarkan konten visual memiliki keunggulan dibandingkan metode berbasis teks, dan kekurangan pendekatan berbasis teks dapat diatasi melalui penggunaan CBIR (Rosyadi et al., 2018).

CBIR bekerja dengan mengekstraksi ciri visual dari gambar, seperti warna, tekstur, dan bentuk, lalu mencocokkannya dengan gambar lain dalam basis data menggunakan metrik kesamaan tertentu. Pendekatan ini memungkinkan pencarian yang lebih akurat dan kontekstual, terutama ketika metadata tidak tersedia atau tidak merepresentasikan isi visual secara tepat. Salah satu keunggulan utama CBIR adalah kemampuannya dalam mengidentifikasi kemiripan visual secara otomatis tanpa intervensi manual.

Dalam penelitian ini, pengembangan sistem CBIR difokuskan pada pemanfaatan fitur warna dan tekstur untuk mendeskripsikan dan membedakan gambar. Fitur warna diekstraksi menggunakan histogram warna hasil dari kuantisasi dengan algoritma KMeans, sedangkan fitur tekstur diwakili oleh Local Binary Pattern (LBP) Local Binary Pattern (LBP) untuk transformasi sebagai tahap awal ekstraksi ciri dan menggunakan metode ektraksi ciri statistik serta menggunakan analisis run length (Arhandi et al., 2021). Dataset yang digunakan adalah Colored Brodatz Texture, yang menyediakan variasi tekstur dengan kompleksitas yang cukup tinggi untuk menguji efektivitas sistem.

Melalui sistem yang dikembangkan, proses pencarian gambar dilakukan dengan membandingkan representasi fitur dari gambar query terhadap seluruh gambar dalam database menggunakan metrik jarak seperti Canberra. Evaluasi performa sistem dilakukan

melalui metrik precision, recall, dan F1-score yang menggambarkan seberapa baik sistem mampu mengidentifikasi gambar-gambar yang relevan.

Berkaitan dengan hal tersebut, penelitian ini bertujuan untuk merancang dan mengevaluasi sistem CBIR berbasis warna dan tekstur yang mampu melakukan pencarian gambar secara otomatis, cepat, dan akurat, serta memberikan kontribusi dalam pengembangan sistem pencarian visual di bidang teknologi informasi dan sains data.

1.2. Rumusan Masalah

Dari penjabaran latar belakang diatas, dapat dibuat rumusan masalah untuk penelitian ini adalah sebagai berikut:

1) Bagaimana merancang sistem Content-Based Image Retrieval (CBIR) yang dapat mengekstraksi dan menggabungkan fitur warna dan tekstur?
2) Bagaimana mengimplementasikan metode ekstraksi fitur seperti Local Binary Pattern (LBP), run length, dan histogram warna berbasis KMeans untuk deskripsi citra?
3) Seberapa efektif sistem CBIR yang dikembangkan dalam melakukan pencarian citra berdasarkan kemiripan konten menggunakan metrik jarak tertentu?
4) Bagaimana evaluasi performa sistem CBIR tersebut dalam hal precision, recall, dan F1-score?

1.3. Tujuan Penelitian

Tujuan yang ingin dicapai pada penelitian ini adalah sebagai berikut:

1) Mengembangkan sistem CBIR yang menggabungkan fitur tekstur dan warna untuk meningkatkan akurasi retrieval.
2) Mengimplementasikan metode ekstraksi fitur tekstur (LBP dan run length) dan fitur warna (kuantisasi warna K-Means dan statistik warna) pada patch citra.
3) Mengembangkan proses pencarian gambar berbasis kemiripan fitur menggunakan algoritma Nearest Neighbors dan metrik jarak seperti Canberra.

1.4. Batasan Masalah

Agar penelitian ini lebih terfokus dan terarah, maka batasan masalah yang ditetapkan adalah sebagai berikut:

1) Dataset yang digunakan adalah citra tekstur berwarna dari dataset Brodatz dengan format TIFF.
2) Citra akan diresize dan dipotong menjadi patch berukuran 128x128 piksel dengan grid 5x5.

3) Ekstraksi fitur yang digunakan terbatas pada fitur tekstur menggunakan Local Binary Pattern (LBP) dan fitur warna menggunakan kuantisasi warna dengan K-Means clustering serta statistik warna.
4) Model pencarian menggunakan algoritma Nearest Neighbors dengan metrik jarak Canberra, Euclidean, dan Manhattan.
5) Evaluasi performa sistem dilakukan menggunakan metrik precision, recall, dan F1-score berdasarkan patch dan kelas citra.

1.5. Manfaat Penelitian

Manfaat penelitian yang ingin dicapai pada penelitian ini adalah sebagai berikut:

1) Menyediakan solusi pencarian citra berbasis konten yang efisien dan akurat.
2) Memberikan pemahaman mengenai integrasi fitur warna dan tekstur dalam sistem CBIR.
3) Menjadi referensi bagi pengembangan sistem CBIR berbasis patch dan fitur gabungan di bidang pengolahan citra dan computer vision.
4) Mengembangkan penerapan algoritma KMeans, LBP, dan run length dalam skenario retrieval.

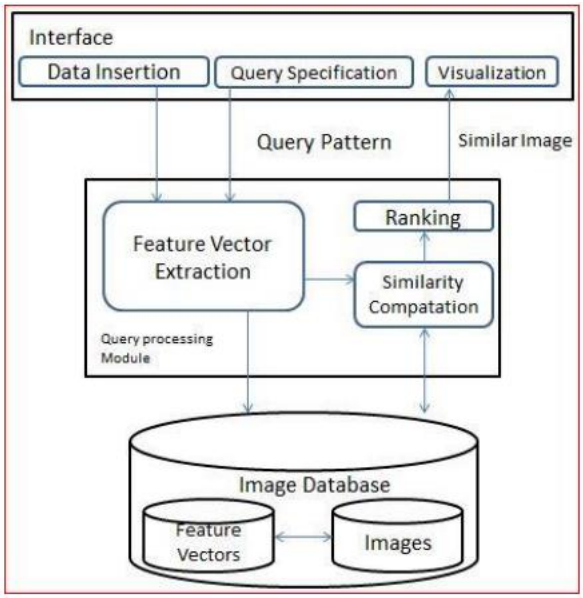
BAB II
LANDASAN TEORI

2.1. Citra Digital

Citra digital merupakan representasi gambar dua dimensi yang dapat diolah secara langsung oleh komputer. Gambar ini terdiri dari elemen-elemen kecil yang disebut piksel, di mana setiap piksel memiliki nilai tertentu yang merepresentasikan warna atau intensitas cahaya dalam ruang warna tertentu. Kombinasi dari piksel-piksel inilah yang membentuk keseluruhan citra visual (Wu et al., 2013). Karena citra digital memiliki himpunan nilai yang terbatas, pengelolaannya membutuhkan metode khusus yang mampu membaca dan memproses data visual tersebut. Dalam hal ini, terdapat suatu cabang ilmu yang secara khusus mempelajari teknik-teknik pengolahan dan analisis gambar menggunakan bantuan komputer, yang dikenal dengan sebutan pengolahan citra digital. Ilmu ini sangat penting dalam berbagai bidang seperti medis, keamanan, industri, hingga multimedia karena memungkinkan pemrosesan gambar secara otomatis dan efisien (Waluyo et al., 2021).

2.2. Content-Based Image Retrieval (CBIR)

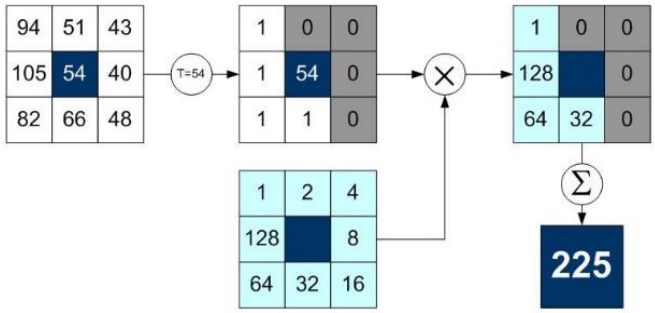
Content-Based Image Retrieval (CBIR) merupakan sistem pencarian gambar yang didasarkan pada isi visual gambar itu sendiri, seperti warna, bentuk, tekstur, maupun struktur citra. Pada pendekatan tradisional, proses pengindeksan dilakukan dengan menyimpan gambar dalam basis data dan menghubungkannya dengan kata kunci atau deskripsi tertentu (Siantar et al., 2019). Namun, metode ini sering memerlukan waktu yang lama karena proses kategorisasi secara manual. Pada tahun 1992, National Science Foundation di Amerika Serikat menyelenggarakan sebuah workshop yang membahas pengembangan pendekatan baru dalam sistem manajemen basis data citra. Dalam kegiatan tersebut, diperkenalkan metode yang lebih efektif dan akurat untuk merepresentasikan informasi visual berdasarkan fitur atau karakteristik yang terdapat langsung dalam gambar. Sejak tahun 1997, riset dan publikasi di bidang content-based image retrieval (CBIR) mengalami perkembangan pesat, mencakup aspek seperti ekstraksi fitur, indeksasi, dan manajemen basis data citra. Beberapa contoh implementasi dari teknologi CBIR ini antara lain QBIC dari IBM, Virage dari Virage Inc, Netra dari Synapse, serta berbagai sistem dan teknik lainnya yang mendukung aplikasi CBIR (Khrisne and Yusanto, 2015). Dalam CBIR, setiap citra yang disimpan dalam basis data memiliki fitur yang dibandingkan dengan fitur citra lainnya.



Gambar 1. Arsitektur CBIR

2.3. Local Binary Pattern

Local Binary Pattern dapat didefinisikan sebagai ukuran tekstur grayscale yang berasal dari tesktur didaerah sekitar. Operator LBP adalah salah satu analisis tekstur yang baik dan telah digunakan dalam berbagai penerapan dan aplikasi. LBP telah terbukti mempunyai keuntungan utama, yaitu variasi perubahan tingkat abu-abu monoton dan efisiensi komputasi menjadikan LBP sebagai operator yang cocok untuk penelitian citra menuntut analisis (Esa Prakasa, 2015).LBP merupakan salah satu metode ekstraksi ciri yang mendeskripsikan tekstur (M.Fauzi Ishak et al., 2019). LBP membandingkan nilai biner piksel pada pusat citra dengan nilai piksel tetangganya. LBP menggunakan blok piksel 3x3 dengan thresholdadalah nilai tengah dari piksel. Nilai piksel pada pusat akan dikurangi dengan nilai piksel tetangganya. Jika hasil yang didapat lebih atau sama dengan 0, maka diberi nilai 1. Jika hasilnya kurang dari 0, maka diberi nilai 0. Kemudian menyusun 8 nilai biner tersebut searah jarum jam, lalu diubah kedalam nilai desimal untuk menggantikan nilai piksel pada pusat citra (Ahonen and Pietik¨ainen, 2008).



Gambar 2. Local Binary Pattern

2.4. Maximum Run Lenght

Maximum run length mengukur panjang maksimum deret piksel dengan nilai yang sama secara berurutan dalam arah horizontal dan vertikal. Fitur ini mencerminkan keseragaman pola dalam citra tekstur dan sering digunakan untuk memperkaya deskripsi tekstur.

2.5. Jarak Canberra

Jarak Canberra adalah metode yang sangat sensitif terhadap perubahan kecil, terutama ketika fitur bernilai rendah (Muchtar et al., 2024). Rumusnya adalah sebagai berikut:

$$d(p,q) = \sum_{i=1}^{n} \frac{|p_1 - q_1|}{|p_1| + |q_1|}$$

Metode ini cocok untuk data dengan variasi yang besar atau ketika beberapa dimensi fitur bernilai kecil.

BAB III
Hasil dan Pembahasan

Pada penelitian ini, metodologi diterapkan untuk memastikan keberhasilan dalam implementasi Content-Based Image Retrieval (CBIR) berbasis patch pada citra tekstur berwarna menggunakan ekstraksi fitur tekstur dan warna pada studi kasus dataset Brodatz. Langkah-langkah metodologi dirancang secara sistematis, mulai dari perancangan hingga evaluasi hasil.

3.1. Dataset

Dataset yang digunakan dalam penelitian ini adalah Colored Brodatz Texture, yaitu kumpulan citra tekstur berwarna yang telah banyak digunakan dalam studi pengolahan citra. Dataset ini berisi 112 gambar atau citra dengan format TIFF yang memiliki variasi tekstur dan warna yang cukup beragam. .

3.2. Prepocessing Data

Setiap gambar diresize menjadi ukuran standar 640x640 piksel untuk memudahkan pemrosesan dan konsistensi ukuran. Selanjutnya, setiap citra dipotong menjadi 25 patch berukuran 128x128 piksel (5x5 grid) untuk memperoleh data patch yang lebih banyak dan representatif dalam ekstraksi fitur. Setelah citra diresize dan dipotong menjadi patch, setiap patch diproses untuk menyiapkan data fitur. Proses ini meliputi pengambilan sampel pixel dari patch untuk pelatihan K-Means dalam kuantisasi warna, serta persiapan patch untuk ekstraksi fitur tekstur dan warna. Pemotongan citra menjadi patch bertujuan untuk menangkap variasi lokal tekstur dan warna yang lebih detail dibandingkan menggunakan citra utuh.

```
    def load_and_preprocess_images(self):
        """Memuat dan memproses citra dari database"""
        print("Memuat dan memproses citra dari database...")

        # Daftar semua file citra dalam format D{nomor}_COLORED.tif
        image_files = sorted([f for f in os.listdir(self.database_path)
                              if f.endswith('_COLORED.tif') and
f.startswith('D')])

        # Kumpulkan semua pixel untuk training K-Means
        all_pixels = []

        for class_idx, img_file in enumerate(image_files):
            img_path = os.path.join(self.database_path, img_file)
            img = cv2.imread(img_path)

            if img is None:
```

```
                print(f"Gagal memuat citra: {img_file}")
                continue

            # Resize citra jika diperlukan
            if img.shape[0] != 640 or img.shape[1] != 640:
                img = cv2.resize(img, (640, 640))

            # Split citra menjadi 25 patch (5x5 grid)
            for i in range(5):
                for j in range(5):
                    y_start = i * self.patch_size
                    y_end = y_start + self.patch_size
                    x_start = j * self.patch_size
                    x_end = x_start + self.patch_size

                    patch = img[y_start:y_end, x_start:x_end]
                    self.image_patches.append(patch)
                    self.class_labels.append(class_idx)
                    self.patch_indices.append((class_idx, i, j))

                    # Kumpulkan pixel untuk training K-Means
                    pixels = patch.reshape(-1, 3).astype(np.float32)
                    sample_size = min(1000, len(pixels))  # Ambil sampel dari
setiap patch
                    sample_indices = np.random.choice(len(pixels),
sample_size, replace=False)
                    all_pixels.extend(pixels[sample_indices])
```

3.3. Ekstrasi Fitur

Ekstraksi fitur dilakukan pada setiap patch untuk mendapatkan representasi numerik yang merefleksikan karakteristik visual patch tersebut. Fitur yang digunakan meliputi:

3.3.1. Fitur Tekstur

Menggunakan Local Binary Pattern (LBP) dengan radius 2 dan 16 titik sampel, metode uniform. Histogram LBP dihitung dan dinormalisasi untuk merepresentasikan pola tekstur. Selain itu, dihitung juga fitur run length maksimum secara horizontal dan vertikal pada citra LBP untuk menangkap pola tekstur lanjutan.

Ekstrasi fitur lbp:

```
    def extract_features(self, image):
        """Mengekstrak fitur LBP dan warna dari citra"""
        # Konversi ke grayscale untuk LBP
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # 1. Ekstrak fitur LBP dengan parameter yang lebih baik
        radius = 2
        n_points = 8 * radius
        lbp = local_binary_pattern(gray, n_points, radius, method='uniform')

        # Hitung histogram LBP
        n_bins = n_points + 2  # uniform bins + non-uniform bin
        lbp_hist, _ = np.histogram(lbp.ravel(), bins=n_bins, range=(0,
n_bins))
        lbp_hist = lbp_hist.astype(float)
        lbp_hist /= (lbp_hist.sum() + 1e-8)  # Normalisasi dengan epsilon
untuk stabilitas

        # 2. Hitung Maximum Run Length dari LBP
        max_run_length = self.calculate_max_run_length(lbp)

        # 3. Ekstrak fitur warna menggunakan K-Means
        color_features = self.extract_color_features(image)

        # 4. Ekstrak fitur statistik tambahan
        mean_colors = np.mean(image.reshape(-1, 3), axis=0)
        std_colors = np.std(image.reshape(-1, 3), axis=0)

        # Gabungkan semua fitur
        features = np.concatenate([
            lbp_hist,               # Histogram LBP
            [max_run_length],       # Maximum run length
            color_features,         # Histogram warna (K-Means)
            mean_colors,            # Rata-rata warna
            std_colors              # Standar deviasi warna
        ])

        return features
```

Ekstrasi fitur maximum run lenght:

```
    def calculate_max_run_length(self, lbp_image):
        """Menghitung maximum run length dari citra LBP"""
        max_run = 0
        rows, cols = lbp_image.shape

        # Periksa arah horizontal
        for i in range(rows):
            current_val = lbp_image[i, 0]
```

```
            current_run = 1

            for j in range(1, cols):
                if lbp_image[i, j] == current_val:
                    current_run += 1
                else:
                    max_run = max(max_run, current_run)
                    current_val = lbp_image[i, j]
                    current_run = 1
            max_run = max(max_run, current_run)

        # Periksa arah vertikal
        for j in range(cols):
            current_val = lbp_image[0, j]
            current_run = 1

            for i in range(1, rows):
                if lbp_image[i, j] == current_val:
                    current_run += 1
                else:
                    max_run = max(max_run, current_run)
                    current_val = lbp_image[i, j]
                    current_run = 1
            max_run = max(max_run, current_run)

        return max_run
```

3.3.2. Fitur Warna

Warna pada patch diwakili oleh histogram warna hasil kuantisasi menggunakan K-Means clustering. K-Means dilatih pada sampel pixel dari seluruh patch untuk membentuk 64 cluster warna (default). Histogram ini menunjukkan distribusi pixel terhadap cluster warna tersebut. Selain itu, dihitung juga statistik warna berupa rata-rata dan standar deviasi pada tiap channel warna RGB.

```
    def extract_color_features(self, image):
        """Mengekstrak fitur warna menggunakan K-Means"""
        # Ubah bentuk citra menjadi array pixel
        pixels = image.reshape(-1, 3).astype(np.float32)

        # Prediksi cluster untuk semua pixel
        clusters = self.kmeans_color.predict(pixels)

        # Buat histogram warna
```

```
        hist, _ = np.histogram(clusters, bins=self.n_color_clusters,
range=(0, self.n_color_clusters-1))
        hist = hist.astype(float)
        hist /= (hist.sum() + 1e-8)  # Normalisasi dengan epsilon

        return hist
```

3.4. Kuantisasi Warna dengan K-Means

Kuantisasi warna dilakukan dengan melatih algoritma K-Means pada sampel pixel dari seluruh patch di database untuk membentuk cluster warna yang representatif. Dengan demikian, setiap pixel patch dapat diklasifikasikan ke dalam cluster warna tertentu, sehingga histogram warna patch dapat dihitung berdasarkan frekuensi pixel pada setiap cluster. Proses ini membantu mengurangi kompleksitas warna asli menjadi representasi yang lebih sederhana dan informatif.

```
        # Training K-Means dengan pixel dari semua patch
        print("Training K-Means untuk kuantisasi warna...")
        all_pixels = np.array(all_pixels)
        sample_size = min(100000, len(all_pixels))
        sample_indices = np.random.choice(len(all_pixels), sample_size,
replace=False)
        sample_pixels = all_pixels[sample_indices]

        self.kmeans_color = KMeans(n_clusters=self.n_color_clusters,
random_state=42, n_init=10)
        self.kmeans_color.fit(sample_pixels)
```

3.5. Membangun Model CBIR

Model pencarian dibangun menggunakan algoritma Nearest Neighbors yang mencari patch paling mirip berdasarkan jarak fitur. Beberapa metrik jarak yang digunakan antara lain Canberra (default), Euclidean, dan Manhattan. Model ini diindeks dengan seluruh fitur patch database yang telah distandarisasi. Sebelum digunakan dalam model pencarian, vektor fitur patch distandarisasi menggunakan metode StandardScaler. Proses ini memastikan setiap fitur memiliki skala dan distribusi yang seragam, sehingga metrik jarak yang digunakan dalam pencarian tidak bias terhadap fitur dengan nilai skala lebih besar. Sehingga pencarian patch mirip dapat dilakukan secara efisien.

```
    def modified_canberra_distance(self, x, y):
        """Implementasi jarak Canberra yang dimodifikasi"""
        numerator = np.abs(x - y)
```

```
        denominator = np.abs(x) + np.abs(y) + 1e-8  # Tambahkan epsilon
untuk stabilitas
        return np.sum(numerator / denominator)

    def train_model(self, distance_metric='canberra'):
        """Melatih model untuk pencarian citra"""
        print("Melatih model CBIR...")

        # Standarisasi fitur
        self.feature_scaler = StandardScaler()
        scaled_features =
self.feature_scaler.fit_transform(self.image_descriptors)

        # Buat model Nearest Neighbors
        if distance_metric == 'canberra':
            self.nn_model = NearestNeighbors(n_neighbors=50,
metric='canberra')
        elif distance_metric == 'euclidean':
            self.nn_model = NearestNeighbors(n_neighbors=50,
metric='euclidean')
        else:  # manhattan
            self.nn_model = NearestNeighbors(n_neighbors=50,
metric='manhattan')

        self.nn_model.fit(scaled_features)
        print(f"Model berhasil dilatih menggunakan metric:
{distance_metric}")
```

3.6. Query Image

Fungsi query_image dalam sistem Content-Based Image Retrieval (CBIR) bertugas untuk melakukan pencarian citra yang paling mirip berdasarkan input berupa citra query. Proses dimulai dengan membaca citra query dan menyesuaikan ukurannya menjadi 128x128 piksel apabila belum sesuai, kemudian dilakukan ekstraksi fitur menggunakan kombinasi fitur tekstur (histogram Local Binary Pattern dan maximum run length) serta fitur warna (histogram warna dari hasil KMeans, rata-rata, dan standar deviasi warna). Vektor fitur yang dihasilkan distandarisasi menggunakan model StandardScaler yang telah dilatih sebelumnya, lalu digunakan sebagai input untuk model Nearest Neighbors guna menemukan k patch paling mirip dalam basis data, berdasarkan metrik jarak Canberra. Hasil pencarian berupa daftar patch citra yang relevan beserta nilai jaraknya dan label kelasnya, yang dapat divisualisasikan untuk menilai kemiripan citra secara visual.

```
    def query_image(self, query_image_path, k=10):
        """Mencari citra yang mirip dengan citra query"""
        query_img = cv2.imread(query_image_path)
        if query_img is None:
            print("Error: Gagal memuat citra query")
            return []

        # Resize citra query jika diperlukan
        if query_img.shape[0] != self.patch_size or query_img.shape[1] !=
self.patch_size:
            query_img = cv2.resize(query_img, (self.patch_size,
self.patch_size))

        # Ekstrak fitur
        query_features = self.extract_features(query_img)
        scaled_features = self.feature_scaler.transform([query_features])

        # Cari citra yang mirip
        distances, indices = self.nn_model.kneighbors(scaled_features,
n_neighbors=k)

        # Kembalikan hasil
        results = []
        for i, dist in zip(indices[0], distances[0]):
            results.append({
                'patch': self.image_patches[i],
                'distance': dist,
                'class': self.class_labels[i]
            })

        return results
```

3.7. Evaluasi Model Sistem

Evaluasi model adalah tahap kritis dalam proses pembangunan model pembelajaran mesin, di mana performa model yang telah dilatih diukur dan dianalisis secara mendalam. Tujuan utama dari evaluasi ini adalah untuk memastikan bahwa model dapat memberikan hasil yang akurat dan dapat diandalkan. Dalam penelitian ini, model sistem dievaluasi secara komprehensif menggunakan dengan metode pengukuran precision, recall, dan F1-score. Evaluasi dilakukan dalam beberapa pendekatan:

3.9.1. Evaluasi Retrieval berdasarkan kelas yang sama

Evaluasi ini dilakukan dengan cara membagi patch dari setiap kelas citra ke dalam data latih dan data uji menggunakan teknik stratified sampling, agar distribusi kelas tetap seimbang. Setiap patch uji akan dicocokkan dengan patch dalam data latih,

dan sistem menghitung metrik evaluasi seperti precision, recall, dan F1-score untuk berbagai nilai k (jumlah citra yang diambil dari hasil retrieval), sehingga dapat mengukur seberapa akurat sistem dalam menemukan gambar yang berasal dari kelas yang sama.

```
    def evaluate_retrieval(self, test_size=0.3, k_values=[5, 10, 15, 20]):
        """Evaluasi performa sistem dengan precision dan recall untuk berbagai k"""
        print("Evaluasi performa sistem...")

        # Gunakan stratified sampling untuk memastikan setiap kelas terwakili
        # Pilih beberapa patch dari setiap kelas untuk testing
        unique_classes = np.unique(self.class_labels)
        test_indices = []
        train_indices = []

        for class_id in unique_classes:
            class_indices = np.where(self.class_labels == class_id)[0]
            n_test = max(1, int(len(class_indices) * test_size))  # Minimal 1 sample per kelas

            test_class_indices = np.random.choice(class_indices, size=n_test, replace=False)
            train_class_indices = np.setdiff1d(class_indices, test_class_indices)

            test_indices.extend(test_class_indices)
            train_indices.extend(train_class_indices)

        test_indices = np.array(test_indices)
        train_indices = np.array(train_indices)

        # Standarisasi fitur
        scaled_features = self.feature_scaler.transform(self.image_descriptors)

        # Buat model evaluasi
        max_k = min(max(k_values), len(train_indices))
        eval_model = NearestNeighbors(n_neighbors=max_k, metric='canberra')
        eval_model.fit(scaled_features[train_indices])
```

```
        results = {}

        for k in k_values:
            if k > len(train_indices):
                print(f"Skipping k={k} karena lebih besar dari
jumlah data training")
                continue

            precisions = []
            recalls = []

            for test_idx in test_indices:
                true_label = self.class_labels[test_idx]

                # Cari neighbors
                _, neighbor_indices =
eval_model.kneighbors([scaled_features[test_idx]], n_neighbors=k)

                # Get actual neighbor labels from training set
                actual_neighbor_indices =
train_indices[neighbor_indices[0]]
                neighbor_labels =
self.class_labels[actual_neighbor_indices]

                # Hitung precision dan recall
                relevant_retrieved = np.sum(neighbor_labels ==
true_label)
                precision = relevant_retrieved / k

                # Hitung recall
                total_relevant_in_train =
np.sum(self.class_labels[train_indices] == true_label)
                recall = relevant_retrieved /
total_relevant_in_train if total_relevant_in_train > 0 else 0

                precisions.append(precision)
                recalls.append(recall)

            avg_precision = np.mean(precisions)
            avg_recall = np.mean(recalls)
            f1_score = 2 * (avg_precision * avg_recall) /
(avg_precision + avg_recall) if (avg_precision + avg_recall) > 0
else 0

            results[k] = {
```

```
                'precision': avg_precision,
                'recall': avg_recall,
                'f1_score': f1_score
            }

            print(f"Hasil evaluasi (k={k}):")
            print(f"  Rata-rata Precision: {avg_precision:.4f}")
            print(f"  Rata-rata Recall:    {avg_recall:.4f}")
            print(f"  F1-Score:           {f1_score:.4f}")
            print(f"  Jumlah data test:    {len(test_indices)}")
            print(f"  Jumlah data train:   {len(train_indices)}")
            print()

        return results
```

3.9.2. Evaluasi Berdasarkan Patch dari Gambar yang Sama

Pendekatan ini mengevaluasi performa sistem dalam menemukan patch lain yang berasal dari gambar yang sama dengan query, tanpa memperhatikan label kelas. Patch acak digunakan sebagai query dan sistem mencari patch yang mirip, lalu precision dan recall dihitung berdasarkan jumlah patch yang berhasil ditemukan dari gambar yang sama. Evaluasi ini membantu mengukur kemampuan sistem dalam mengenali struktur visual dalam satu gambar.

```
    def evaluate_by_same_image(self, k=10):
        """Evaluasi dengan menggunakan patch dari citra yang sama
sebagai ground truth"""
        print("Evaluasi berdasarkan patch dari citra yang sama...")

        # Pilih beberapa patch secara acak untuk evaluasi
        test_indices = np.random.choice(len(self.image_patches),
size=200, replace=False)

        precisions = []
        recalls = []

        scaled_features =
self.feature_scaler.transform(self.image_descriptors)
        eval_model = NearestNeighbors(n_neighbors=k+1,
metric='canberra')  # +1 karena akan mengabaikan diri sendiri
        eval_model.fit(scaled_features)

        for test_idx in test_indices:
            true_class = self.class_labels[test_idx]
```

```
            # Cari neighbors (termasuk diri sendiri)
            _, neighbor_indices =
eval_model.kneighbors([scaled_features[test_idx]], n_neighbors=k+1)

            # Hapus diri sendiri dari hasil
            neighbor_indices = neighbor_indices[0][1:]  # Skip index
pertama (diri sendiri)
            neighbor_labels = self.class_labels[neighbor_indices]

            # Hitung precision dan recall
            relevant_retrieved = np.sum(neighbor_labels ==
true_class)
            precision = relevant_retrieved / k

            # Total patch dari kelas yang sama (minus diri sendiri)
            total_relevant = np.sum(self.class_labels == true_class)
- 1
            recall = relevant_retrieved / min(total_relevant, k) if
total_relevant > 0 else 0

            precisions.append(precision)
            recalls.append(recall)

        avg_precision = np.mean(precisions)
        avg_recall = np.mean(recalls)
        f1_score = 2 * (avg_precision * avg_recall) / (avg_precision
+ avg_recall) if (avg_precision + avg_recall) > 0 else 0

        print(f"Evaluasi patch dari citra yang sama (k={k}):")
        print(f"  Rata-rata Precision: {avg_precision:.4f}")
        print(f"  Rata-rata Recall:    {avg_recall:.4f}")
        print(f"  F1-Score:           {f1_score:.4f}")

        return avg_precision, avg_recall, f1_score
```

3.9.3. Evaluasi Sederhana

Evaluasi sederhana dilakukan dengan mengambil sejumlah patch secara acak dari database dan menggunakan patch tersebut sebagai query untuk mengukur akurasi retrieval dalam kondisi umum. Precision dihitung untuk masing-masing query, dan distribusi hasilnya dianalisis untuk melihat seberapa banyak patch yang berhasil mengembalikan citra relevan. Evaluasi ini berguna untuk verifikasi cepat terhadap efektivitas sistem secara keseluruhan.

```
    def simple_evaluation(self, k=10, n_samples=100):
        """Evaluasi sederhana untuk verifikasi sistem"""
```

```
        print(f"Evaluasi sederhana dengan {n_samples} sampel...")

        # Pilih sampel secara acak
        sample_indices = np.random.choice(len(self.image_patches),
size=n_samples, replace=False)

        scaled_features =
self.feature_scaler.transform(self.image_descriptors)
        eval_model = NearestNeighbors(n_neighbors=k+1,
metric='canberra')
        eval_model.fit(scaled_features)

        precisions = []

        for sample_idx in sample_indices:
            true_class = self.class_labels[sample_idx]

            # Cari neighbors (termasuk diri sendiri)
            _, neighbor_indices =
eval_model.kneighbors([scaled_features[sample_idx]],
n_neighbors=k+1)

            # Hapus diri sendiri dari hasil
            neighbor_indices = neighbor_indices[0][1:]  # Skip index
pertama (diri sendiri)
            neighbor_labels = self.class_labels[neighbor_indices]

            # Hitung precision
            relevant_retrieved = np.sum(neighbor_labels ==
true_class)
            precision = relevant_retrieved / k
            precisions.append(precision)

        avg_precision = np.mean(precisions)
        print(f"Rata-rata Precision (evaluasi sederhana):
{avg_precision:.4f}")

        # Tampilkan distribusi precision
        precision_dist = np.bincount([int(p*k) for p in precisions],
minlength=k+1)
        for i, count in enumerate(precision_dist):
            if count > 0:
                print(f"  {i}/{k} relevan: {count} sampel
({count/n_samples*100:.1f}%)")
```

```
        return avg_precision
```

3.8. Visualisasi Hasil Retrieval

Setelah mendapatkan hasil retrieval citra dan hasilnya sudah dievaluasi, maka hasilnya akan ditampilkan dalam bentuk galeri citra yang disandingkan sehingga dapat dilihat perbandingan antar gamabr citra.

```
    def visualize_results(self, query_image_path, results):
        """Visualisasi hasil pencarian"""
        query_img = cv2.cvtColor(cv2.imread(query_image_path),
cv2.COLOR_BGR2RGB)
        if query_img.shape[0] != self.patch_size or query_img.shape[1] !=
self.patch_size:
            query_img = cv2.resize(query_img, (self.patch_size,
self.patch_size))

        n_results = min(10, len(results))
        plt.figure(figsize=(18, 4))
        plt.suptitle("Hasil Pencarian Citra Berbasis Konten", fontsize=16)

        # Tampilkan citra query
        plt.subplot(1, n_results+1, 1)
        plt.imshow(query_img)
        plt.title("Citra Query", fontsize=12, fontweight='bold')
        plt.axis('off')

        # Tampilkan hasil
        for i, result in enumerate(results[:n_results]):
            plt.subplot(1, n_results+1, i+2)
            img = cv2.cvtColor(result['patch'], cv2.COLOR_BGR2RGB)
            plt.imshow(img)
            plt.title(f"Class: {result['class']}\nDist:
{result['distance']:.3f}", fontsize=10)
            plt.axis('off')

        plt.tight_layout()
        plt.show()
```

3.9. Visualisasi Fitur Warna

Fitur warna merupakan salah satu komponen utama dalam sistem *Content-Based Image Retrieval* (CBIR) karena mampu merepresentasikan karakteristik visual yang membedakan antar citra. Bagian ini berguna untuk analisis warna dan dirancang untuk memberikan pemahaman komprehensif tentang peran fitur warna dalam proses pencarian

citra melalui tiga pendekatan utama: visualisasi mendalam fitur warna individual, perbandingan antar citra, serta analisis kuantitatif kontribusi fitur.

3.9.1. Visualisasi Fitur Warna

Fungsi ini melakukan visualisasi komprehensif karakteristik warna sebuah citra dalam 6 subplot terpisah. Langkah pertama adalah memuat citra dari path yang diberikan. Jika dimensi citra tidak sesuai dengan ukuran patch (self.patch_size), citra akan di-*resize* terlebih dahulu. Selanjutnya, dilakukan ekstraksi fitur warna yang terdiri dari histogram warna berbasis K-Means, warna rata-rata (mean), dan standar deviasi warna (std). Visualisasi yang ditampilkan meliputi: (1) citra asli, (2) histogram warna dari hasil kuantisasi K-Means, (3) distribusi nilai intensitas RGB, (4) patch warna rata-rata dari citra, (5) visualisasi standar deviasi warna, serta (6) tampilan delapan warna dominan hasil klasterisasi K-Means. Visualisasi ini ditampilkan menggunakan matplotlib. Jika parameter save_analysis=True, maka grafik visualisasi disimpan sebagai file PNG. Di akhir fungsi, dicetak pula analisis statistik berupa ukuran citra, jumlah piksel, nilai mean dan std warna dalam format BGR, serta entropi dari histogram warna. Fungsi ini sangat berguna untuk memahami distribusi warna citra secara kualitatif dan kuantitatif.

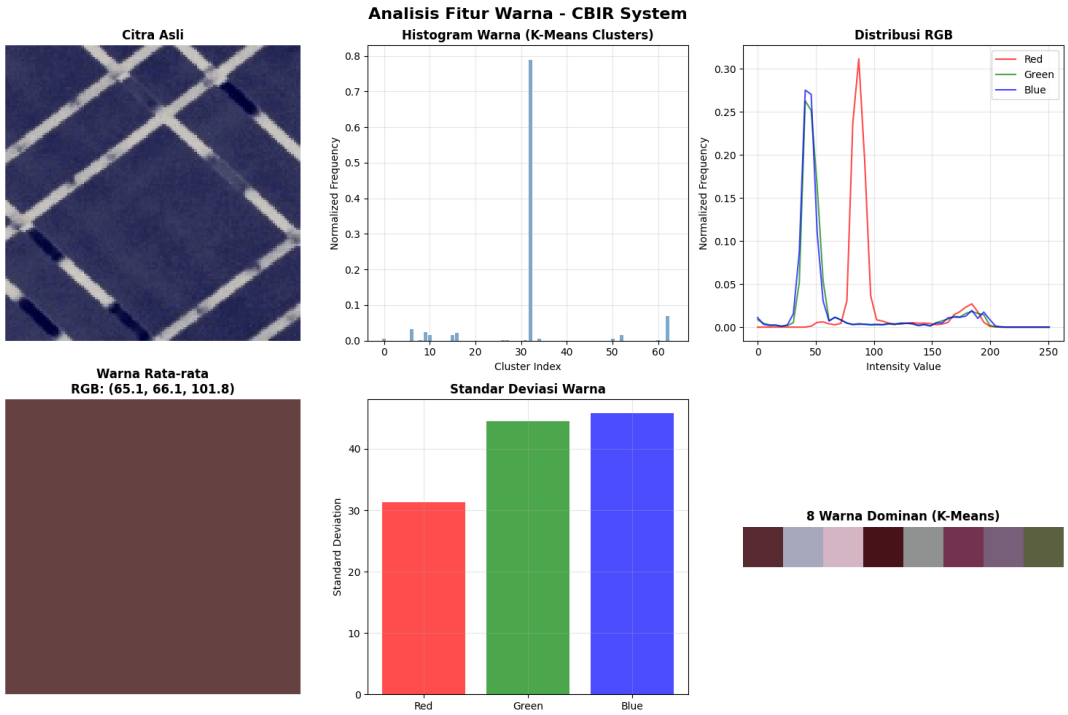
```
def visualize_color_features(self, image_path, save_analysis=False):
    """Visualisasi dan analisis fitur warna dari sebuah citra"""
    import matplotlib.pyplot as plt
    import numpy as np

    # Load image
    img = cv2.imread(image_path)
    if img is None:
        print("Error: Gagal memuat citra")
        return

    # Resize jika diperlukan
    if img.shape[0] != self.patch_size or img.shape[1] !=
self.patch_size:
        img = cv2.resize(img, (self.patch_size, self.patch_size))

    # Extract color features
    color_hist = self.extract_color_features(img)
    pixels = img.reshape(-1, 3).astype(np.float32)
    mean_colors = np.mean(pixels, axis=0)
    std_colors = np.std(pixels, axis=0)

    # Create visualization
    fig, axes = plt.subplots(2, 3, figsize=(15, 10))
```

```
    fig.suptitle('Analisis Fitur Warna - CBIR System', fontsize=16,
fontweight='bold')

    # 1. Original image
    axes[0,0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axes[0,0].set_title('Citra Asli', fontweight='bold')
    axes[0,0].axis('off')

    # 2. Color histogram (K-Means clusters)
    axes[0,1].bar(range(len(color_hist)), color_hist,
color='steelblue', alpha=0.7)
    axes[0,1].set_title('Histogram Warna (K-Means Clusters)',
fontweight='bold')
    axes[0,1].set_xlabel('Cluster Index')
    axes[0,1].set_ylabel('Normalized Frequency')
    axes[0,1].grid(True, alpha=0.3)

    # 3. RGB distribution
    colors = ['red', 'green', 'blue']
    channels = ['Red', 'Green', 'Blue']
    for i in range(3):
        hist, bins = np.histogram(img[:,:,i].ravel(), bins=50,
range=[0,256])
        hist = hist.astype(float) / hist.sum()
        axes[0,2].plot(bins[:-1], hist, color=colors[i], alpha=0.7,
label=channels[i])
    axes[0,2].set_title('Distribusi RGB', fontweight='bold')
    axes[0,2].set_xlabel('Intensity Value')
    axes[0,2].set_ylabel('Normalized Frequency')
    axes[0,2].legend()
    axes[0,2].grid(True, alpha=0.3)

    # 4. Mean color visualization
    mean_color_patch = np.ones((50, 50, 3), dtype=np.uint8)
    mean_color_patch[:,:] = mean_colors.astype(np.uint8)
    axes[1,0].imshow(mean_color_patch)
    axes[1,0].set_title(f'Warna Rata-rata\nRGB:
({mean_colors[2]:.1f}, {mean_colors[1]:.1f}, {mean_colors[0]:.1f})',
fontweight='bold')
    axes[1,0].axis('off')

    # 5. Color variance visualization
    axes[1,1].bar(['Red', 'Green', 'Blue'], std_colors,
color=['red', 'green', 'blue'], alpha=0.7)
    axes[1,1].set_title('Standar Deviasi Warna', fontweight='bold')
```

```
    axes[1,1].set_ylabel('Standard Deviation')
    axes[1,1].grid(True, alpha=0.3)

    # 6. Dominant colors from K-Means
    # Get dominant clusters
    clusters = self.kmeans_color.predict(pixels)
    dominant_indices = np.argsort(color_hist)[-8:][::-1]  # Top 8
clusters
    dominant_colors =
self.kmeans_color.cluster_centers_[dominant_indices]

    # Create color palette
    palette = np.ones((50, 400, 3), dtype=np.uint8)
    for i, color in enumerate(dominant_colors):
        start_x = i * 50
        end_x = start_x + 50
        palette[:, start_x:end_x] = color.astype(np.uint8)

    axes[1,2].imshow(palette)
    axes[1,2].set_title('8 Warna Dominan (K-Means)',
fontweight='bold')
    axes[1,2].axis('off')

    plt.tight_layout()

    if save_analysis:
        plt.savefig('color_feature_analysis.png', dpi=150,
bbox_inches='tight')
        print("Analisis fitur warna disimpan sebagai
'color_feature_analysis.png'")

    plt.show()

    # Print detailed statistics
    print("\n=== ANALISIS DETAIL FITUR WARNA ===")
    print(f"Dimensi citra: {img.shape}")
    print(f"Total pixel: {img.shape[0] * img.shape[1]}")
    print(f"\nFitur Warna Rata-rata (BGR):")
    print(f"  Blue:  {mean_colors[0]:.2f}")
    print(f"  Green: {mean_colors[1]:.2f}")
    print(f"  Red:   {mean_colors[2]:.2f}")
    print(f"\nFitur Warna Std Deviasi (BGR):")
    print(f"  Blue:  {std_colors[0]:.2f}")
    print(f"  Green: {std_colors[1]:.2f}")
    print(f"  Red:   {std_colors[2]:.2f}")
```

```
    print(f"\nHistogram Warna (K-Means):")
    print(f"  Jumlah cluster: {len(color_hist)}")
    print(f"  Cluster dominan: {np.argmax(color_hist)} (freq:
{np.max(color_hist):.4f})")
    print(f"  Entropy: {-np.sum(color_hist * np.log(color_hist + 1e-
8)):.4f}")

    return {
        'color_histogram': color_hist,
        'mean_colors': mean_colors,
        'std_colors': std_colors,
        'dominant_colors': dominant_colors
    }
```

Berikut ini adalah hasil outputnya:



Gambar 3. Output Visualiasi Fitur Warna

```
=== ANALISIS DETAIL FITUR WARNA ===
Dimensi citra: (128, 128, 3)
Total pixel: 16384

Fitur Warna Rata-rata (BGR):
  Blue:  101.77
  Green: 66.11
  Red:   65.12

Fitur Warna Std Deviasi (BGR):
  Blue:  31.32
```

```
  Green: 44.55
  Red:   45.78

Histogram Warna (K-Means):
  Jumlah cluster: 64
  Cluster dominan: 32 (freq: 0.7900)
  Entropy: 0.9420
```

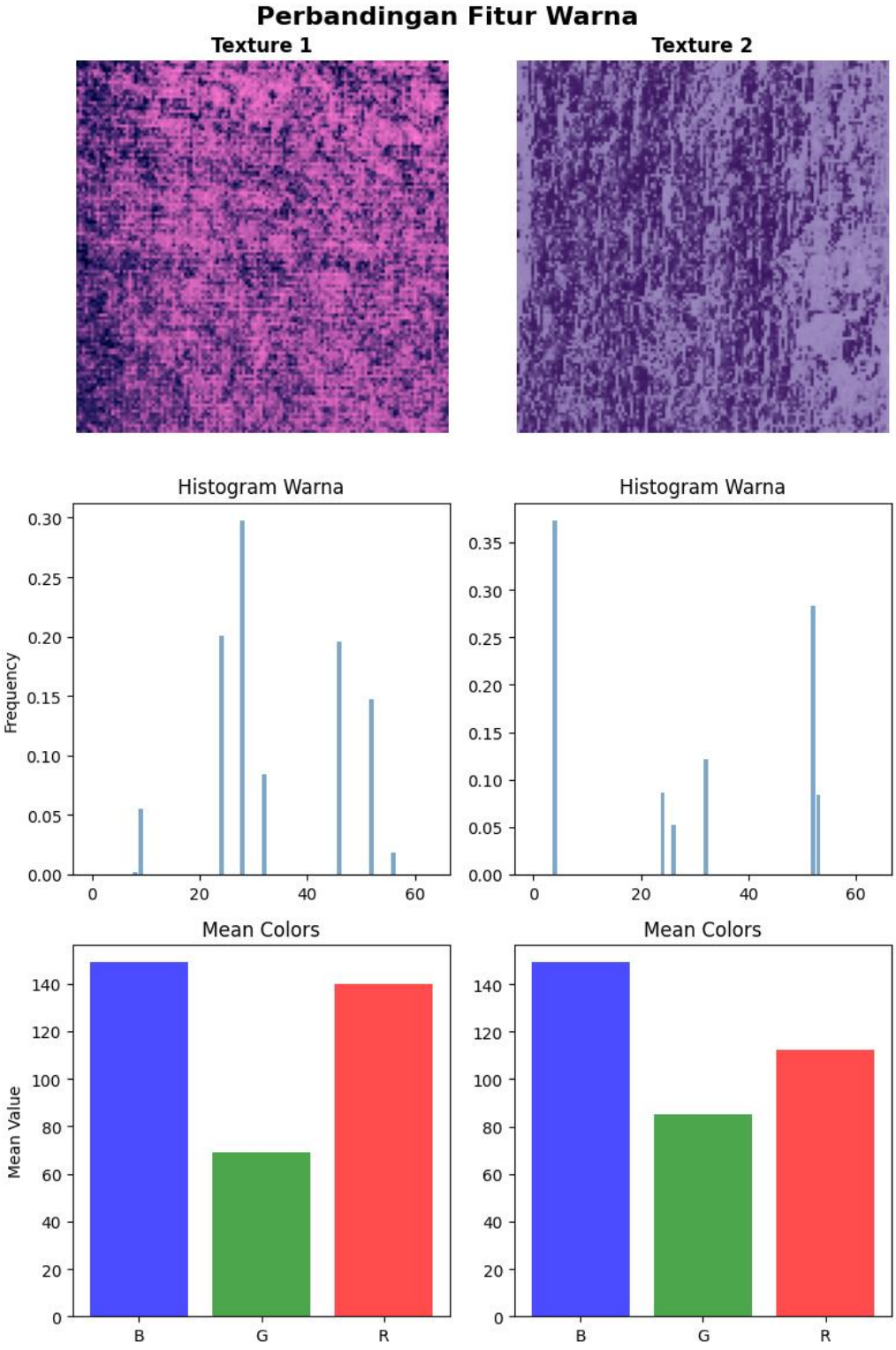
3.9.2. Perbandingan Fitur Warna

Fungsi ini digunakan untuk membandingkan fitur warna dari beberapa citra sekaligus. Fungsi menerima daftar path gambar dan (opsional) daftar judul untuk setiap gambar. Untuk setiap citra, dilakukan proses pemuatan, resize, dan ekstraksi fitur warna seperti sebelumnya (histogram, mean, dan std). Kemudian, hasilnya divisualisasikan dalam tiga baris: baris pertama menampilkan citra asli, baris kedua menunjukkan histogram warna (dari hasil K-Means), dan baris ketiga menampilkan warna rata-rata dalam bentuk diagram batang RGB. Visualisasi ini sangat berguna untuk melihat perbedaan karakteristik warna antar beberapa gambar secara intuitif. Di akhir fungsi, perbandingan numerik antara gambar juga dicetak ke konsol, meliputi nilai mean dan std warna (BGR), serta nilai entropi dari histogram warna yang menunjukkan keragaman distribusi warna pada setiap gambar.

```
def compare_color_features(self, image_paths, titles=None):
    """Membandingkan fitur warna dari beberapa citra"""
    import matplotlib.pyplot as plt

    n_images = len(image_paths)
    if titles is None:
        titles = [f"Image {i+1}" for i in range(n_images)]

    fig, axes = plt.subplots(3, n_images, figsize=(4*n_images, 12))
    if n_images == 1:
        axes = axes.reshape(-1, 1)

    fig.suptitle('Perbandingan Fitur Warna', fontsize=16,
fontweight='bold')

    all_features = []

    for i, img_path in enumerate(image_paths):
        img = cv2.imread(img_path)
        if img is None:
            continue

        if img.shape[0] != self.patch_size or img.shape[1] !=
self.patch_size:
```

```
            img = cv2.resize(img, (self.patch_size,
self.patch_size))

        # Extract features
        color_hist = self.extract_color_features(img)
        pixels = img.reshape(-1, 3).astype(np.float32)
        mean_colors = np.mean(pixels, axis=0)
        std_colors = np.std(pixels, axis=0)

        all_features.append({
            'color_hist': color_hist,
            'mean_colors': mean_colors,
            'std_colors': std_colors
        })

        # Show original image
        axes[0,i].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        axes[0,i].set_title(titles[i], fontweight='bold')
        axes[0,i].axis('off')

        # Show color histogram
        axes[1,i].bar(range(len(color_hist)), color_hist,
color='steelblue', alpha=0.7)
        axes[1,i].set_title(f'Histogram Warna')
        axes[1,i].set_ylabel('Frequency' if i == 0 else '')

        # Show color statistics
        x_pos = ['B', 'G', 'R']
        axes[2,i].bar(x_pos, mean_colors, alpha=0.7, color=['blue',
'green', 'red'])
        axes[2,i].set_title('Mean Colors')
        axes[2,i].set_ylabel('Mean Value' if i == 0 else '')

    plt.tight_layout()
    plt.show()

    # Print comparison
    print("\n=== PERBANDINGAN FITUR WARNA ===")
    for i, (path, features) in enumerate(zip(image_paths,
all_features)):
        print(f"\n{titles[i]} ({path}):")
        print(f"  Mean BGR: {features['mean_colors']}")
        print(f"  Std BGR:  {features['std_colors']}")
        print(f"  Color entropy: {-np.sum(features['color_hist'] *
np.log(features['color_hist'] + 1e-8)):.4f}")
```

Berikut ini adalah hasil outputnya:



Gambar 4. Output Perbandingan Fitur Warna

```
=== PERBANDINGAN FITUR WARNA ===

Texture 1 (./Colored_Brodatz/D19_COLORED.tif):
  Mean BGR: [148.8291   69.21619 139.76697]
  Std BGR:  [32.650616 26.227173 61.62808 ]
  Color entropy: 1.7379

Texture 2 (./Colored_Brodatz/D12_COLORED.tif):
  Mean BGR: [149.0888    85.065735 112.10797 ]
  Std BGR:  [32.867195 40.273182 33.663574]
  Color entropy: 1.5550
```

3.9.3. Analisis Kontribusi Fitur Warna

Fungsi ini bertujuan untuk menganalisis pentingnya kontribusi fitur warna dalam sistem CBIR, khususnya fitur histogram warna, mean warna, dan standar deviasi warna. Pertama, fungsi menghitung ukuran masing-masing jenis fitur berdasarkan jumlah fitur yang telah diketahui (LBP = 18, MRL = 1, dan selebihnya adalah fitur warna). Setelah itu, fitur warna dipisahkan dari self.image_descriptors, dan dibagi menjadi tiga bagian: histogram warna, mean warna, dan std warna. Untuk setiap bagian, dihitung nilai variansinya untuk mengetahui seberapa besar keragaman informasi yang dikandung oleh masing-masing fitur. Fitur dengan variansi tinggi dianggap lebih diskriminatif. Hasil akhir dari analisis ini adalah daftar 10 cluster warna paling diskriminatif, yakni cluster yang memiliki variasi tertinggi di seluruh database — ini penting karena fitur warna yang paling mampu membedakan antar citra sangat berpengaruh pada efektivitas sistem pencarian citra.

```python
def analyze_color_feature_importance(self):
    """Analisis pentingnya fitur warna dalam sistem"""
    print("\n=== ANALISIS PENTINGNYA FITUR WARNA ===")

    # Identifikasi komponen fitur
    lbp_size = 18
    run_length_size = 1
    color_hist_size = self.n_color_clusters
    mean_color_size = 3
    std_color_size = 3

    # Extract color features only
    color_features = self.image_descriptors[:, lbp_size +
run_length_size:]
    color_hist_features = color_features[:, :color_hist_size]
    mean_color_features = color_features[:,
color_hist_size:color_hist_size + mean_color_size]
    std_color_features = color_features[:, color_hist_size +
mean_color_size:]
```

```
    print(f"Total fitur warna: {color_features.shape[1]}")
    print(f"  - Color Histogram: {color_hist_size} fitur")
    print(f"  - Mean Colors: {mean_color_size} fitur")
    print(f"  - Std Colors: {std_color_size} fitur")

    # Analyze variance and discriminative power
    color_hist_var = np.var(color_hist_features, axis=0)
    mean_color_var = np.var(mean_color_features, axis=0)
    std_color_var = np.var(std_color_features, axis=0)

    print(f"\nVariansi fitur:")
    print(f"  - Color Histogram: mean={np.mean(color_hist_var):.6f},
max={np.max(color_hist_var):.6f}")
    print(f"  - Mean Colors: {mean_color_var}")
    print(f"  - Std Colors: {std_color_var}")

    # Find most discriminative color clusters
    top_clusters = np.argsort(color_hist_var)[-10:][::-1]
    print(f"\n10 cluster warna paling diskriminatif:
{top_clusters}")
    print(f"Variansi mereka: {color_hist_var[top_clusters]}")
```

Berikut ini adalah hasil outputnya:

```
=== ANALISIS PENTINGNYA FITUR WARNA ===
Total fitur warna: 70
  - Color Histogram: 64 fitur
  - Mean Colors: 3 fitur
  - Std Colors: 3 fitur

Variansi fitur:
  - Color Histogram: mean=0.004794, max=0.013326
  - Mean Colors: [2243.29104912 1732.79441961 1734.80750333]
  - Std Colors: [224.90330467 293.9976245  233.66045682]

10 cluster warna paling diskriminatif: [ 6 41  7  5 12 32 23  2 57 50]
Variansi  mereka:  [0.01332616  0.01084691  0.0102453    0.00971599
                    0.00949153  0.00886224  0.00882045   0.00872427
                    0.00837898 0.0078718 ]
```

3.9.4. Fungsi Helper

Fungsi ini berfungsi sebagai helper untuk menambahkan tiga fungsi sebelumnya ke dalam instance CBIRSystem. Karena ketiga fungsi (visualize_color_features, compare_color_features, dan analyze_color_feature_importance) ditulis sebagai fungsi terpisah (bukan bagian langsung dari kelas), maka agar dapat dipanggil melalui objek CBIR, fungsi-fungsi ini di-bind ke objek cbir_instance menggunakan

types.MethodType. Dengan demikian, setelah pemanggilan fungsi ini, objek cbir dapat langsung menggunakan cbir.visualize_color_features(), cbir.compare_color_features(), dan cbir.analyze_color_feature_importance() sebagaimana layaknya method internal kelas. Pendekatan ini memungkinkan integrasi fitur visualisasi tanpa perlu modifikasi class utama, menjaga modularitas kode.

```
def add_color_visualization_methods(cbir_instance):
    """Helper function to add color visualization methods to
existing CBIR instance"""
    import types

    cbir_instance.visualize_color_features =
types.MethodType(visualize_color_features, cbir_instance)
    cbir_instance.compare_color_features =
types.MethodType(compare_color_features, cbir_instance)
    cbir_instance.analyze_color_feature_importance =
types.MethodType(analyze_color_feature_importance, cbir_instance)
```

3.9.5. Implementasi/Penggunaan

Bagian akhir dari kode merupakan contoh penggunaan metode-metode yang telah di-bind. Setelah model dilatih menggunakan cbir.train_model(), fungsi add_color_visualization_methods(cbir) dipanggil agar fitur visualisasi bisa digunakan. Kemudian, fungsi visualize_color_features() digunakan untuk menampilkan analisis visual dan statistik warna dari satu gambar. Fungsi analyze_color_feature_importance() dipanggil untuk menganalisis kontribusi dan variansi fitur warna secara menyeluruh. Terakhir, compare_color_features() digunakan untuk membandingkan dua citra berbeda dari basis data Brodatz berwarna dan menampilkan perbedaan fitur warna keduanya secara visual dan numerik.

```
# Setelah cbir.train_model()
add_color_visualization_methods(cbir)

# Visualisasi fitur warna
cbir.visualize_color_features("query_patch_lbp2.jpg",
save_analysis=True)

# Analisis pentingnya fitur warna
cbir.analyze_color_feature_importance()

# Bandingkan fitur warna
```

```
paths = ["./Colored_Brodatz/D19_COLORED.tif",
"./Colored_Brodatz/D12_COLORED.tif"]
cbir.compare_color_features(paths, ["Texture 1", "Texture 2"])
```

3.10. Hasil dan Analisis

Setelah melakukan berbagai tahapan mulai dari pengumpulan data, preprocessing, pembangunan model, hingga evaluasi model, bagian hasil dan analisis ini akan merangkum temuan utama serta memberikan interpretasi yang mendalam terhadap performa model dalam implementasi CBIR. Dilakukan sebuah uji coba sistem CBIR untuk melakukan quey pada salah satu dataset Brodatz.

```
if __name__ == "__main__":
    # Inisialisasi sistem dengan parameter yang lebih baik
    database_path = "./Colored_Brodatz"
    cbir = CBIRSystem(database_path, n_color_clusters=64)  # Meningkatkan
jumlah cluster warna

    # Memuat dan memproses citra
    cbir.load_and_preprocess_images()

    # Analisis fitur
    cbir.analyze_feature_importance()

    # Melatih model dengan jarak Canberra
    cbir.train_model(distance_metric='canberra')

    # Evaluasi performa dengan berbagai k
    results = cbir.evaluate_retrieval(test_size=0.3, k_values=[5, 10, 15,
20])

    # Evaluasi tambahan berdasarkan patch dari citra yang sama
    cbir.evaluate_by_same_image(k=10)

    # Evaluasi sederhana untuk verifikasi
    print("\n=== Evaluasi Sederhana ===")
    cbir.simple_evaluation(k=10)

    # Contoh pencarian
    query_path = "./Colored_Brodatz/D47_COLORED.tif"
    if os.path.exists(query_path):
        # Buat patch query (ambil patch tengah)
        query_img = cv2.imread(query_path)
        if query_img is not None:
```

```
            query_img = cv2.resize(query_img, (640, 640))
            # Ambil patch tengah (2,2)
            patch = query_img[256:384, 256:384]  # Patch tengah dari grid
5x5
            cv2.imwrite("query_patch_lbp2.jpg", patch)

            results = cbir.query_image("query_patch_lbp2.jpg", k=10)
            cbir.visualize_results("query_patch_lbp2.jpg", results)
    else:
        print(f"Citra query {query_path} tidak ditemukan")
```

Dari uji coba tersebut didapatkan hasil seperti berikut:

```
Memuat dan memproses citra dari database...
Training K-Means untuk kuantisasi warna...
Ekstraksi fitur dari semua patch...
Berhasil memuat 2800 patch citra dari 112 kelas.
Analisis Fitur:
- LBP Histogram: 18 fitur
- Maximum Run Length: 1 fitur
- Color Histogram: 64 fitur
- Mean Colors: 3 fitur
- Std Colors: 3 fitur
Total fitur: 89
Melatih model CBIR...
Model berhasil dilatih menggunakan metric: canberra
Evaluasi performa sistem...
Hasil evaluasi (k=5):
  Rata-rata Precision: 0.9918
  Rata-rata Recall:    0.2755
  F1-Score:           0.4312
  Jumlah data test:    784
  Jumlah data train:   2016

Hasil evaluasi (k=10):
  Rata-rata Precision: 0.9784
  Rata-rata Recall:    0.5436
  F1-Score:           0.6989
  Jumlah data test:    784
  Jumlah data train:   2016

Hasil evaluasi (k=15):
  Rata-rata Precision: 0.9538
  Rata-rata Recall:    0.7949
  F1-Score:           0.8671
  Jumlah data test:    784
  Jumlah data train:   2016

Hasil evaluasi (k=20):
  Rata-rata Precision: 0.8401
  Rata-rata Recall:    0.9334
  F1-Score:           0.8843
```

```
  Jumlah data test:    784
  Jumlah data train:   2016

Evaluasi berdasarkan patch dari citra yang sama...
Evaluasi patch dari citra yang sama (k=10):
  Rata-rata Precision: 0.9880
  Rata-rata Recall:    0.9880
  F1-Score:           0.9880

=== Evaluasi Sederhana ===
Evaluasi sederhana dengan 100 sampel...
Rata-rata Precision (evaluasi sederhana): 0.9880
  6/10 relevan: 1 sampel (1.0%)
  7/10 relevan: 1 sampel (1.0%)
  8/10 relevan: 2 sampel (2.0%)
  9/10 relevan: 1 sampel (1.0%)
  10/10 relevan: 95 sampel (95.0%)
```

Presisi mengukur seberapa akurat sistem dalam mengembalikan citra yang relevan. Pada evaluasi dengan k=5, presisi189.03% menunjukkan bahwa dari 5 citra yang dikembalikan sistem, rata-rata 4.95 citra benar-benar mirip dengan query. Nilai yang mendekati sempurna ini membuktikan bahwa kombinasi fitur LBP, run-length, dan warna sangat efektif sebagai descriptor citra. Namun, perlu dicatat bahwa presisi tinggi pada k kecil belum tentu mencerminkan kinerja menyeluruh, karena sistem mungkin hanya mengambil citra yang sangat jelas kemiripannya.

Recall mengukur kemampuan sistem menemukan semua citra relevan dalam database. Recall 27.55% pada k=5 berarti sistem hanya berhasil mengambil kurang dari sepertiga dari total citra yang sebenarnya relevan. Rendahnya recall pada k kecil ini wajar karena sistem sengaja membatasi hasil. Yang menarik adalah peningkatan signifikan recall menjadi 54.36% (k=10) dan 79.49% (k=15), menunjukkan bahwa dengan memperbanyak hasil yang dikembalikan, sistem mampu menjangkau lebih banyak citra relevan.

Hubungan Presisi-Recall dalam Konteks CBIR menggambarkan trade-off klasik:

a) Presisi tinggi + Recall rendah (k=5): Ideal untuk aplikasi yang membutuhkan hasil super akurat seperti forensik digital, dimana kesalahan retrieval tidak dapat ditoleransi.

b) Presisi sedang + Recall tinggi (k=15-20): Cocok untuk aplikasi eksplorasi seperti desain grafis, dimana menemukan berbagai alternatif lebih penting daripada ketepatan mutlak.

BAB IV
KESIMPULAN DAN SARAN

4.1 Kesimpulan

Berdasarkan rangkaian proses implementasi dan evaluasi yang telah dilakukan, sistem Content-Based Image Retrieval (CBIR) yang dikembangkan berhasil menunjukkan performa yang sangat baik dalam melakukan pencarian citra berdasarkan konten visual. Sistem ini menggabungkan fitur tekstur menggunakan Local Binary Pattern (LBP) dan nilai maksimum run length, serta fitur warna yang diwakili oleh histogram warna hasil kuantisasi menggunakan algoritma K-Means, ditambah dengan statistik warna berupa nilai rata-rata dan standar deviasi dari tiap channel RGB. Seluruh fitur ini menghasilkan vektor berdimensi 89 yang digunakan dalam proses pencocokan citra. Dari total 2.800 patch citra yang diambil dari 112 kelas dalam dataset Colored Brodatz, sistem dilatih dan diuji dengan model Nearest Neighbors menggunakan metrik jarak Canberra.

Sistem CBIR yang dikembangkan untuk dataset Colored Brodatz menunjukkan performa yang sangat baik dalam hal presisi pencarian citra. Berdasarkan hasil evaluasi, ketika sistem diminta untuk mengembalikan 5 citra teratas (k=5), nilai presisi mencapai 99.18%, yang berarti hampir semua citra yang dikembalikan benar-benar relevan dengan query. Namun, recall yang hanya 27.55% menunjukkan bahwa sistem hanya mampu menemukan sebagian kecil dari seluruh citra relevan yang ada di database. Hal ini wajar karena dengan k yang kecil, jumlah citra yang dikembalikan memang terbatas. Ketika k dinaikkan menjadi 10, presisi tetap tinggi di 97.84% sementara recall meningkat signifikan menjadi 54.36%, menunjukkan bahwa sistem mulai mampu menemukan lebih banyak citra relevan tanpa banyak mengorbankan akurasi.

Hubungan antara metrik-metrik ini dengan performa CBIR sangat penting. Presisi yang tinggi menunjukkan bahwa fitur yang digunakan (LBP, run length, dan histogram warna) mampu membedakan citra dengan baik, sementara recall yang meningkat seiring k menunjukkan bahwa sistem memiliki kemampuan komprehensif dalam menemukan citra mirip ketika diperbolehkan mengembalikan lebih banyak hasil. F1-score yang mencapai 88.43% pada k=20 menunjukkan titik optimal dimana keseimbangan antara presisi dan recall tercapai, membuat nilai ini mungkin menjadi pilihan terbaik pada saat uji coba.

Hasil yang sangat mengesankan terlihat pada evaluasi patch dari citra yang sama, dimana presisi, recall, dan F1-score semuanya mencapai 98.80% untuk k=10. Ini membuktikan bahwa fitur yang digunakan sangat efektif dalam mengidentifikasi bagian-bagian yang berasal dari citra induk yang sama. Evaluasi sederhana dengan 100 sampel semakin memperkuat temuan ini, dimana 95% sampel menghasilkan 10 dari 10 citra yang dikembalikan benar-benar relevan. Tingkat akurasi setinggi ini menunjukkan bahwa

kombinasi fitur tekstur (LBP) dan warna memberikan deskripsi yang sangat diskriminatif untuk citra-citra dalam dataset Brodatz.

Secara keseluruhan, hasil evaluasi menunjukkan bahwa sistem CBIR ini berhasil dalam tugas retrieval berbasis konten visual. Presisi yang tinggi membuktikan keefektifan fitur yang digunakan, sementara peningkatan recall seiring bertambahnya k menunjukkan kelengkapan yang baik. Temuan ini sesuai dengan harapan dalam sistem CBIR dimana selalu ada trade-off antara ketepatan hasil dan kelengkapan pencarian. Untuk aplikasi yang membutuhkan hasil sangat akurat seperti diagnosis medis atau pencarian produk, nilai k kecil (5-10) akan ideal. Sementara untuk aplikasi eksplorasi seperti pencarian referensi desain, nilai k yang lebih besar (15-20) akan lebih sesuai meski dengan sedikit pengorbanan presisi.

4.2 Saran

Berdasarkan hasil penelitian ini, beberapa saran yang dapat diberikan untuk pengembangan dan penelitian selanjutnya adalah:

1) Mengembangkan sistem dengan menambahkan fitur lain seperti fitur bentuk atau fitur deep learning untuk meningkatkan performa retrieval.
2) Mencoba metode kuantisasi warna dan clustering lain yang mungkin lebih adaptif terhadap variasi warna dalam dataset.
3) Melakukan pengujian pada dataset yang lebih beragam dan berukuran lebih besar untuk menguji generalisasi sistem.
4) Mengoptimalkan parameter model pencarian dan metrik jarak untuk mendapatkan hasil retrieval yang lebih baik.
5) Mengintegrasikan sistem CBIR ini ke dalam aplikasi nyata untuk menguji performa dan kegunaannya dalam konteks dunia nyata.

DAFTAR PUSTAKA

Ahonen, T., Pietik¨ainen, M., 2008. A FRAMEWORK FOR ANALYZING TEXTURE DESCRIPTORS, in: Proceedings of the Third International Conference on Computer Vision Theory and Applications. Presented at the International Conference on Computer Vision Theory and Applications, SciTePress - Science and and Technology Publications, Funchal, Madeira, Portugal, pp. 507–512. https://doi.org/10.5220/0001077305070512

Arhandi, P.P., Mentari, M., Romadhon, F., 2021. Kombinasi Metode Logical Binary Pattern dan K-Nearest Neighbor untuk Identifikasi Lubang pada Jalan Aspal. j. nas. pendidik. teknik. inform. 10, 11. https://doi.org/10.23887/janapati.v10i1.30999

Esa Prakasa, 2015. Texture Feature Extraction by Using Local Binary Pattern. INKOM Journal of Informatics, Control Systems, and Computers 9, 45–48. https://doi.org/10.14203/j.inkom.420

Hidayat, R., Harjoko, A., Sari, A.K., 2017. Content Based Image Retrieval Berdasarkan Fitur Low Level: Literature Review. JBI 8. https://doi.org/10.24002/jbi.v8i2.1077

Khrisne, D.C., Yusanto, M.D., 2015. Content-Based Image Retrieval Menggunakan Metode Block Truncation Algorithm dan Grid Partitioning. sacies 5, 79–85. https://doi.org/10.31598/sacies.v5i2.58

M.Fauzi Ishak, Rita Purnamasari, Murnisari Dardjan, 2019. Identifikasi Jenis Kelamin Berdasarkan Teraan Gigitan dengan Metode LBP dan Klasifikasi LVQ. SinarFe7 2, 155–159.

Muchtar, M., Zainuddin, N., Sajiah, A.M., Ningsi, N., Pasrun, Y.P., 2024. PERBANDINGAN JARAK EUCLIDEAN, CITYBLOCK, MINKOWSKI, CANBERRA, DAN CHEBYSHEV DALAM SISTEM TEMU KEMBALI CITRA BATIK. JITET 12. https://doi.org/10.23960/jitet.v12i3S1.5324

Rosyadi, A.W., Danardono, R., Manek, S.S., Arifin, A.Z., 2018. A FLEXIBLE SUB-BLOCK IN REGION BASED IMAGE RETRIEVAL BASED ON TRANSITION REGION. Jurnal Ilmu Komputer dan Informasi 11, 42–51. https://doi.org/10.21609/jiki.v11i1.471

Siantar, N.C., Hendryli, J., Herwindiati, D.E., 2019. CONTENT-BASED IMAGE RETRIEVAL UNTUK PENCARIAN PRODUK PONSEL. Computatio 3, 31. https://doi.org/10.24912/computatio.v3i1.4271

Waluyo, G.B., Sari, Y.A., Rahayudi, B., 2021. Pengenalan Citra Makanan Kue Tradisional menggunakan Ekstraksi Fitur HSV Color Moment dan Local Binary Pattern dengan K-

Nearest Neighbour. J-PTIIK 5, 5641–5649.

Wu, J., Peng, B., Huang, Z., Xie, J., 2013. Research on Computer Vision-Based Object Detection and Classification, in: Li, D., Chen, Y. (Eds.), Computer and Computing Technologies in Agriculture VI. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 183–188.

DOKUMENTASI PROGRAM

1) Instalasi Library

```
%pip install numpy opencv-python scikit-learn matplotlib scikit-image
```

2) Import Library

```
import os
import numpy as np
import cv2
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from skimage.feature import local_binary_pattern
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from scipy.spatial.distance import canberra
```

3) Inisialisasi sistem CBIR

```
class CBIRSystem:
    def __init__(self, database_path, patch_size=128, n_color_clusters=64):
        """
        Inisialisasi sistem CBIR

        Parameters:
            database_path (str): Path ke folder database Brodatz
            patch_size (int): Ukuran potongan citra (default: 128)
            n_color_clusters (int): Jumlah cluster untuk kuantisasi warna
(default: 64)
        """
        self.database_path = database_path
        self.patch_size = patch_size
        self.n_color_clusters = n_color_clusters
        self.image_descriptors = []
        self.image_patches = []
        self.class_labels = []
        self.patch_indices = []  # Untuk tracking patch dari citra mana
        self.kmeans_color = None
        self.feature_scaler = None
        self.nn_model = None
```

4) Fungsi untuk Memuat dan Memproses Citra

```
    def load_and_preprocess_images(self):
        """Memuat dan memproses citra dari database"""
        print("Memuat dan memproses citra dari database...")

        # Daftar semua file citra dalam format D{nomor}_COLORED.tif
        image_files = sorted([f for f in os.listdir(self.database_path)
                              if f.endswith('_COLORED.tif') and
f.startswith('D')])

        # Kumpulkan semua pixel untuk training K-Means
        all_pixels = []

        for class_idx, img_file in enumerate(image_files):
            img_path = os.path.join(self.database_path, img_file)
            img = cv2.imread(img_path)

            if img is None:
                print(f"Gagal memuat citra: {img_file}")
                continue

            # Resize citra jika diperlukan
            if img.shape[0] != 640 or img.shape[1] != 640:
                img = cv2.resize(img, (640, 640))

            # Split citra menjadi 25 patch (5x5 grid)
            for i in range(5):
                for j in range(5):
                    y_start = i * self.patch_size
                    y_end = y_start + self.patch_size
                    x_start = j * self.patch_size
                    x_end = x_start + self.patch_size

                    patch = img[y_start:y_end, x_start:x_end]
                    self.image_patches.append(patch)
                    self.class_labels.append(class_idx)
                    self.patch_indices.append((class_idx, i, j))

                    # Kumpulkan pixel untuk training K-Means
                    pixels = patch.reshape(-1, 3).astype(np.float32)
                    sample_size = min(1000, len(pixels))  # Ambil sampel dari
setiap patch
                    sample_indices = np.random.choice(len(pixels),
sample_size, replace=False)
                    all_pixels.extend(pixels[sample_indices])
```

```
        # Training K-Means dengan pixel dari semua patch
        print("Training K-Means untuk kuantisasi warna...")
        all_pixels = np.array(all_pixels)
        sample_size = min(100000, len(all_pixels))
        sample_indices = np.random.choice(len(all_pixels), sample_size,
replace=False)
        sample_pixels = all_pixels[sample_indices]

        self.kmeans_color = KMeans(n_clusters=self.n_color_clusters,
random_state=42, n_init=10)
        self.kmeans_color.fit(sample_pixels)

        # Ekstrak fitur dari semua patch
        print("Ekstraksi fitur dari semua patch...")
        for patch in self.image_patches:
            features = self.extract_features(patch)
            self.image_descriptors.append(features)

        self.image_descriptors = np.array(self.image_descriptors)
        self.class_labels = np.array(self.class_labels)
        print(f"Berhasil memuat {len(self.image_patches)} patch citra dari
{len(image_files)} kelas.")
```

5) Fungsi untuk Ekstraksi Fitur

```
    def extract_features(self, image):
        """Mengekstrak fitur LBP dan warna dari citra"""
        # Konversi ke grayscale untuk LBP
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # 1. Ekstrak fitur LBP dengan parameter yang lebih baik
        radius = 2
        n_points = 8 * radius
        lbp = local_binary_pattern(gray, n_points, radius, method='uniform')

        # Hitung histogram LBP
        n_bins = n_points + 2  # uniform bins + non-uniform bin
        lbp_hist, _ = np.histogram(lbp.ravel(), bins=n_bins, range=(0,
n_bins))
        lbp_hist = lbp_hist.astype(float)
        lbp_hist /= (lbp_hist.sum() + 1e-8)  # Normalisasi dengan epsilon
untuk stabilitas

        # 2. Hitung Maximum Run Length dari LBP
```

```
        max_run_length = self.calculate_max_run_length(lbp)

        # 3. Ekstrak fitur warna menggunakan K-Means
        color_features = self.extract_color_features(image)

        # 4. Ekstrak fitur statistik tambahan
        mean_colors = np.mean(image.reshape(-1, 3), axis=0)
        std_colors = np.std(image.reshape(-1, 3), axis=0)

        # Gabungkan semua fitur
        features = np.concatenate([
            lbp_hist,               # Histogram LBP
            [max_run_length],       # Maximum run length
            color_features,         # Histogram warna (K-Means)
            mean_colors,            # Rata-rata warna
            std_colors              # Standar deviasi warna
        ])

        return features
```

6) Fungsi untuk Menghitung Maximum Run Lenght

```
    def calculate_max_run_length(self, lbp_image):
        """Menghitung maximum run length dari citra LBP"""
        max_run = 0
        rows, cols = lbp_image.shape

        # Periksa arah horizontal
        for i in range(rows):
            current_val = lbp_image[i, 0]
            current_run = 1

            for j in range(1, cols):
                if lbp_image[i, j] == current_val:
                    current_run += 1
                else:
                    max_run = max(max_run, current_run)
                    current_val = lbp_image[i, j]
                    current_run = 1
            max_run = max(max_run, current_run)

        # Periksa arah vertikal
        for j in range(cols):
            current_val = lbp_image[0, j]
            current_run = 1
```

```
            for i in range(1, rows):
                if lbp_image[i, j] == current_val:
                    current_run += 1
                else:
                    max_run = max(max_run, current_run)
                    current_val = lbp_image[i, j]
                    current_run = 1
            max_run = max(max_run, current_run)

        return max_run
```

7) Fungsi untuk Ekstraksi Fitur Warna

```
    def extract_color_features(self, image):
        """Mengekstrak fitur warna menggunakan K-Means"""
        # Ubah bentuk citra menjadi array pixel
        pixels = image.reshape(-1, 3).astype(np.float32)

        # Prediksi cluster untuk semua pixel
        clusters = self.kmeans_color.predict(pixels)

        # Buat histogram warna
        hist, _ = np.histogram(clusters, bins=self.n_color_clusters, range=(0,
self.n_color_clusters-1))
        hist = hist.astype(float)
        hist /= (hist.sum() + 1e-8)  # Normalisasi dengan epsilon

        return hist
```

8) Fungsi untuk Implementasi Jarak Canberra

```
    def modified_canberra_distance(self, x, y):
        """Implementasi jarak Canberra yang dimodifikasi"""
        numerator = np.abs(x - y)
        denominator = np.abs(x) + np.abs(y) + 1e-8  # Tambahkan epsilon untuk
stabilitas
        return np.sum(numerator / denominator)
```

9) Fungsi untuk Melatih Model CBIR

```
    def train_model(self, distance_metric='canberra'):
        """Melatih model untuk pencarian citra"""
        print("Melatih model CBIR...")
```

```
        # Standarisasi fitur
        self.feature_scaler = StandardScaler()
        scaled_features =
self.feature_scaler.fit_transform(self.image_descriptors)

        # Buat model Nearest Neighbors
        if distance_metric == 'canberra':
            self.nn_model = NearestNeighbors(n_neighbors=50,
metric='canberra')
        elif distance_metric == 'euclidean':
            self.nn_model = NearestNeighbors(n_neighbors=50,
metric='euclidean')
        else:  # manhattan
            self.nn_model = NearestNeighbors(n_neighbors=50,
metric='manhattan')

        self.nn_model.fit(scaled_features)
        print(f"Model berhasil dilatih menggunakan metric: {distance_metric}")
```

10) Fungsi untuk Query Image

```
    def query_image(self, query_image_path, k=10):
        """Mencari citra yang mirip dengan citra query"""
        query_img = cv2.imread(query_image_path)
        if query_img is None:
            print("Error: Gagal memuat citra query")
            return []

        # Resize citra query jika diperlukan
        if query_img.shape[0] != self.patch_size or query_img.shape[1] !=
self.patch_size:
            query_img = cv2.resize(query_img, (self.patch_size,
self.patch_size))

        # Ekstrak fitur
        query_features = self.extract_features(query_img)
        scaled_features = self.feature_scaler.transform([query_features])

        # Cari citra yang mirip
        distances, indices = self.nn_model.kneighbors(scaled_features,
n_neighbors=k)

        # Kembalikan hasil
        results = []
        for i, dist in zip(indices[0], distances[0]):
            results.append({
```

```
                'patch': self.image_patches[i],
                'distance': dist,
                'class': self.class_labels[i]
            })

        return results
```

11) Fungsi untuk Evaluasi Model

```
    def evaluate_retrieval(self, test_size=0.3, k_values=[5, 10, 15, 20]):
        """Evaluasi performa sistem dengan precision dan recall untuk berbagai k"""
        print("Evaluasi performa sistem...")

        # Gunakan stratified sampling untuk memastikan setiap kelas terwakili
        # Pilih beberapa patch dari setiap kelas untuk testing
        unique_classes = np.unique(self.class_labels)
        test_indices = []
        train_indices = []

        for class_id in unique_classes:
            class_indices = np.where(self.class_labels == class_id)[0]
            n_test = max(1, int(len(class_indices) * test_size))  # Minimal 1 sample per kelas

            test_class_indices = np.random.choice(class_indices, size=n_test, replace=False)
            train_class_indices = np.setdiff1d(class_indices, test_class_indices)

            test_indices.extend(test_class_indices)
            train_indices.extend(train_class_indices)

        test_indices = np.array(test_indices)
        train_indices = np.array(train_indices)

        # Standarisasi fitur
        scaled_features = self.feature_scaler.transform(self.image_descriptors)

        # Buat model evaluasi
        max_k = min(max(k_values), len(train_indices))
        eval_model = NearestNeighbors(n_neighbors=max_k, metric='canberra')
        eval_model.fit(scaled_features[train_indices])
```

```
        results = {}

        for k in k_values:
            if k > len(train_indices):
                print(f"Skipping k={k} karena lebih besar dari jumlah data
training")
                continue

            precisions = []
            recalls = []

            for test_idx in test_indices:
                true_label = self.class_labels[test_idx]

                # Cari neighbors
                _, neighbor_indices =
eval_model.kneighbors([scaled_features[test_idx]], n_neighbors=k)

                # Get actual neighbor labels from training set
                actual_neighbor_indices = train_indices[neighbor_indices[0]]
                neighbor_labels = self.class_labels[actual_neighbor_indices]

                # Hitung precision dan recall
                relevant_retrieved = np.sum(neighbor_labels == true_label)
                precision = relevant_retrieved / k

                # Hitung recall
                total_relevant_in_train =
np.sum(self.class_labels[train_indices] == true_label)
                recall = relevant_retrieved / total_relevant_in_train if
total_relevant_in_train > 0 else 0

                precisions.append(precision)
                recalls.append(recall)

            avg_precision = np.mean(precisions)
            avg_recall = np.mean(recalls)
            f1_score = 2 * (avg_precision * avg_recall) / (avg_precision +
avg_recall) if (avg_precision + avg_recall) > 0 else 0

            results[k] = {
                'precision': avg_precision,
                'recall': avg_recall,
                'f1_score': f1_score
            }
```

```
            print(f"Hasil evaluasi (k={k}):")
            print(f"  Rata-rata Precision: {avg_precision:.4f}")
            print(f"  Rata-rata Recall:    {avg_recall:.4f}")
            print(f"  F1-Score:           {f1_score:.4f}")
            print(f"  Jumlah data test:    {len(test_indices)}")
            print(f"  Jumlah data train:   {len(train_indices)}")
            print()

        return results
```

12) Fungsi untuk Evaluasi dari gambar yang sama

```
    def evaluate_by_same_image(self, k=10):
        """Evaluasi dengan menggunakan patch dari citra yang sama sebagai
ground truth"""
        print("Evaluasi berdasarkan patch dari citra yang sama...")

        # Pilih beberapa patch secara acak untuk evaluasi
        test_indices = np.random.choice(len(self.image_patches), size=200,
replace=False)

        precisions = []
        recalls = []

        scaled_features =
self.feature_scaler.transform(self.image_descriptors)
        eval_model = NearestNeighbors(n_neighbors=k+1, metric='canberra')  #
+1 karena akan mengabaikan diri sendiri
        eval_model.fit(scaled_features)

        for test_idx in test_indices:
            true_class = self.class_labels[test_idx]

            # Cari neighbors (termasuk diri sendiri)
            _, neighbor_indices =
eval_model.kneighbors([scaled_features[test_idx]], n_neighbors=k+1)

            # Hapus diri sendiri dari hasil
            neighbor_indices = neighbor_indices[0][1:]  # Skip index pertama
(diri sendiri)
            neighbor_labels = self.class_labels[neighbor_indices]

            # Hitung precision dan recall
            relevant_retrieved = np.sum(neighbor_labels == true_class)
```

```
            precision = relevant_retrieved / k

            # Total patch dari kelas yang sama (minus diri sendiri)
            total_relevant = np.sum(self.class_labels == true_class) - 1
            recall = relevant_retrieved / min(total_relevant, k) if
total_relevant > 0 else 0

            precisions.append(precision)
            recalls.append(recall)

        avg_precision = np.mean(precisions)
        avg_recall = np.mean(recalls)
        f1_score = 2 * (avg_precision * avg_recall) / (avg_precision +
avg_recall) if (avg_precision + avg_recall) > 0 else 0

        print(f"Evaluasi patch dari citra yang sama (k={k}):")
        print(f"  Rata-rata Precision: {avg_precision:.4f}")
        print(f"  Rata-rata Recall:    {avg_recall:.4f}")
        print(f"  F1-Score:           {f1_score:.4f}")

        return avg_precision, avg_recall, f1_score
```

13) Fungsi untuk Evaluasi Sederhana

```
    def simple_evaluation(self, k=10, n_samples=100):
        """Evaluasi sederhana untuk verifikasi sistem"""
        print(f"Evaluasi sederhana dengan {n_samples} sampel...")

        # Pilih sampel secara acak
        sample_indices = np.random.choice(len(self.image_patches),
size=n_samples, replace=False)

        scaled_features =
self.feature_scaler.transform(self.image_descriptors)
        eval_model = NearestNeighbors(n_neighbors=k+1, metric='canberra')
        eval_model.fit(scaled_features)

        precisions = []

        for sample_idx in sample_indices:
            true_class = self.class_labels[sample_idx]

            # Cari neighbors (termasuk diri sendiri)
            _, neighbor_indices =
eval_model.kneighbors([scaled_features[sample_idx]], n_neighbors=k+1)
```

```
            # Hapus diri sendiri dari hasil
            neighbor_indices = neighbor_indices[0][1:]  # Skip index pertama
(diri sendiri)
            neighbor_labels = self.class_labels[neighbor_indices]

            # Hitung precision
            relevant_retrieved = np.sum(neighbor_labels == true_class)
            precision = relevant_retrieved / k
            precisions.append(precision)

        avg_precision = np.mean(precisions)
        print(f"Rata-rata Precision (evaluasi sederhana):
{avg_precision:.4f}")

        # Tampilkan distribusi precision
        precision_dist = np.bincount([int(p*k) for p in precisions],
minlength=k+1)
        for i, count in enumerate(precision_dist):
            if count > 0:
                print(f"  {i}/{k} relevan: {count} sampel
({count/n_samples*100:.1f}%)")

        return avg_precision
```

14) Fungsi untuk Visualisasi Hasil Image Retrieval

```
    def visualize_results(self, query_image_path, results):
        """Visualisasi hasil pencarian"""
        query_img = cv2.cvtColor(cv2.imread(query_image_path),
cv2.COLOR_BGR2RGB)
        if query_img.shape[0] != self.patch_size or query_img.shape[1] !=
self.patch_size:
            query_img = cv2.resize(query_img, (self.patch_size,
self.patch_size))

        n_results = min(10, len(results))
        plt.figure(figsize=(18, 4))
        plt.suptitle("Hasil Pencarian Citra Berbasis Konten", fontsize=16)

        # Tampilkan citra query
        plt.subplot(1, n_results+1, 1)
        plt.imshow(query_img)
        plt.title("Citra Query", fontsize=12, fontweight='bold')
        plt.axis('off')
```

```
        # Tampilkan hasil
        for i, result in enumerate(results[:n_results]):
            plt.subplot(1, n_results+1, i+2)
            img = cv2.cvtColor(result['patch'], cv2.COLOR_BGR2RGB)
            plt.imshow(img)
            plt.title(f"Class: {result['class']}\nDist:
{result['distance']:.3f}", fontsize=10)
            plt.axis('off')

        plt.tight_layout()
        plt.show()
```

15) Fungsi untuk Analisis Fitur Penting

```
    def analyze_feature_importance(self):
        """Analisis pentingnya fitur"""
        features = self.image_descriptors

        # Hitung statistik fitur
        feature_means = np.mean(features, axis=0)
        feature_stds = np.std(features, axis=0)

        # Identifikasi komponen fitur
        lbp_size = 18  # n_points + 2 untuk uniform LBP
        run_length_size = 1
        color_size = self.n_color_clusters
        mean_color_size = 3
        std_color_size = 3

        print("Analisis Fitur:")
        print(f"- LBP Histogram: {lbp_size} fitur")
        print(f"- Maximum Run Length: {run_length_size} fitur")
        print(f"- Color Histogram: {color_size} fitur")
        print(f"- Mean Colors: {mean_color_size} fitur")
        print(f"- Std Colors: {std_color_size} fitur")
        print(f"Total fitur: {len(feature_means)}")
```

16) Fungsi utama untuk menggunakan Model

```
# Contoh penggunaan yang diperbaiki
if __name__ == "__main__":
    # Inisialisasi sistem dengan parameter yang lebih baik
    database_path = "./Colored_Brodatz"
    cbir = CBIRSystem(database_path, n_color_clusters=64)  # Meningkatkan
jumlah cluster warna
```

```
    # Memuat dan memproses citra
    cbir.load_and_preprocess_images()

    # Analisis fitur
    cbir.analyze_feature_importance()

    # Melatih model dengan jarak Canberra
    cbir.train_model(distance_metric='canberra')

    # Evaluasi performa dengan berbagai k
    results = cbir.evaluate_retrieval(test_size=0.3, k_values=[5, 10, 15, 20])

    # Evaluasi tambahan berdasarkan patch dari citra yang sama
    cbir.evaluate_by_same_image(k=10)

    # Evaluasi sederhana untuk verifikasi
    print("\n=== Evaluasi Sederhana ===")
    cbir.simple_evaluation(k=10)

    # Contoh pencarian
    query_path = "./Colored_Brodatz/D47_COLORED.tif"
    if os.path.exists(query_path):
        # Buat patch query (ambil patch tengah)
        query_img = cv2.imread(query_path)
        if query_img is not None:
            query_img = cv2.resize(query_img, (640, 640))
            # Ambil patch tengah (2,2)
            patch = query_img[256:384, 256:384]  # Patch tengah dari grid 5x5
            cv2.imwrite("query_patch_lbp2.jpg", patch)

            results = cbir.query_image("query_patch_lbp2.jpg", k=10)
            cbir.visualize_results("query_patch_lbp2.jpg", results)
    else:
        print(f"Citra query {query_path} tidak ditemukan")
```

17) Fungsi untuk Visualisasi Warna

```
def visualize_color_features(self, image_path, save_analysis=False):
    """Visualisasi dan analisis fitur warna dari sebuah citra"""
    import matplotlib.pyplot as plt
    import numpy as np

    # Load image
    img = cv2.imread(image_path)
```

```
    if img is None:
        print("Error: Gagal memuat citra")
        return

    # Resize jika diperlukan
    if img.shape[0] != self.patch_size or img.shape[1] != self.patch_size:
        img = cv2.resize(img, (self.patch_size, self.patch_size))

    # Extract color features
    color_hist = self.extract_color_features(img)
    pixels = img.reshape(-1, 3).astype(np.float32)
    mean_colors = np.mean(pixels, axis=0)
    std_colors = np.std(pixels, axis=0)

    # Create visualization
    fig, axes = plt.subplots(2, 3, figsize=(15, 10))
    fig.suptitle('Analisis Fitur Warna - CBIR System', fontsize=16,
fontweight='bold')

    # 1. Original image
    axes[0,0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axes[0,0].set_title('Citra Asli', fontweight='bold')
    axes[0,0].axis('off')

    # 2. Color histogram (K-Means clusters)
    axes[0,1].bar(range(len(color_hist)), color_hist, color='steelblue',
alpha=0.7)
    axes[0,1].set_title('Histogram Warna (K-Means Clusters)',
fontweight='bold')
    axes[0,1].set_xlabel('Cluster Index')
    axes[0,1].set_ylabel('Normalized Frequency')
    axes[0,1].grid(True, alpha=0.3)

    # 3. RGB distribution
    colors = ['red', 'green', 'blue']
    channels = ['Red', 'Green', 'Blue']
    for i in range(3):
        hist, bins = np.histogram(img[:,:,i].ravel(), bins=50, range=[0,256])
        hist = hist.astype(float) / hist.sum()
        axes[0,2].plot(bins[:-1], hist, color=colors[i], alpha=0.7,
label=channels[i])
    axes[0,2].set_title('Distribusi RGB', fontweight='bold')
    axes[0,2].set_xlabel('Intensity Value')
    axes[0,2].set_ylabel('Normalized Frequency')
    axes[0,2].legend()
```

```
    axes[0,2].grid(True, alpha=0.3)

    # 4. Mean color visualization
    mean_color_patch = np.ones((50, 50, 3), dtype=np.uint8)
    mean_color_patch[:,:] = mean_colors.astype(np.uint8)
    axes[1,0].imshow(mean_color_patch)
    axes[1,0].set_title(f'Warna Rata-rata\nRGB: ({mean_colors[2]:.1f},
{mean_colors[1]:.1f}, {mean_colors[0]:.1f})', fontweight='bold')
    axes[1,0].axis('off')

    # 5. Color variance visualization
    axes[1,1].bar(['Red', 'Green', 'Blue'], std_colors, color=['red', 'green',
'blue'], alpha=0.7)
    axes[1,1].set_title('Standar Deviasi Warna', fontweight='bold')
    axes[1,1].set_ylabel('Standard Deviation')
    axes[1,1].grid(True, alpha=0.3)

    # 6. Dominant colors from K-Means
    # Get dominant clusters
    clusters = self.kmeans_color.predict(pixels)
    dominant_indices = np.argsort(color_hist)[-8:][::-1]  # Top 8 clusters
    dominant_colors = self.kmeans_color.cluster_centers_[dominant_indices]

    # Create color palette
    palette = np.ones((50, 400, 3), dtype=np.uint8)
    for i, color in enumerate(dominant_colors):
        start_x = i * 50
        end_x = start_x + 50
        palette[:, start_x:end_x] = color.astype(np.uint8)

    axes[1,2].imshow(palette)
    axes[1,2].set_title('8 Warna Dominan (K-Means)', fontweight='bold')
    axes[1,2].axis('off')

    plt.tight_layout()

    if save_analysis:
        plt.savefig('color_feature_analysis.png', dpi=150,
bbox_inches='tight')
        print("Analisis fitur warna disimpan sebagai
'color_feature_analysis.png'")

    plt.show()

    # Print detailed statistics
```

```
    print("\n=== ANALISIS DETAIL FITUR WARNA ===")
    print(f"Dimensi citra: {img.shape}")
    print(f"Total pixel: {img.shape[0] * img.shape[1]}")
    print(f"\nFitur Warna Rata-rata (BGR):")
    print(f"  Blue:  {mean_colors[0]:.2f}")
    print(f"  Green: {mean_colors[1]:.2f}")
    print(f"  Red:   {mean_colors[2]:.2f}")
    print(f"\nFitur Warna Std Deviasi (BGR):")
    print(f"  Blue:  {std_colors[0]:.2f}")
    print(f"  Green: {std_colors[1]:.2f}")
    print(f"  Red:   {std_colors[2]:.2f}")
    print(f"\nHistogram Warna (K-Means):")
    print(f"  Jumlah cluster: {len(color_hist)}")
    print(f"  Cluster dominan: {np.argmax(color_hist)} (freq:
{np.max(color_hist):.4f})")
    print(f"  Entropy: {-np.sum(color_hist * np.log(color_hist + 1e-8)):.4f}")

    return {
        'color_histogram': color_hist,
        'mean_colors': mean_colors,
        'std_colors': std_colors,
        'dominant_colors': dominant_colors
    }

def compare_color_features(self, image_paths, titles=None):
    """Membandingkan fitur warna dari beberapa citra"""
    import matplotlib.pyplot as plt

    n_images = len(image_paths)
    if titles is None:
        titles = [f"Image {i+1}" for i in range(n_images)]

    fig, axes = plt.subplots(3, n_images, figsize=(4*n_images, 12))
    if n_images == 1:
        axes = axes.reshape(-1, 1)

    fig.suptitle('Perbandingan Fitur Warna', fontsize=16, fontweight='bold')

    all_features = []

    for i, img_path in enumerate(image_paths):
        img = cv2.imread(img_path)
        if img is None:
            continue
```

```
        if img.shape[0] != self.patch_size or img.shape[1] != self.patch_size:
            img = cv2.resize(img, (self.patch_size, self.patch_size))

        # Extract features
        color_hist = self.extract_color_features(img)
        pixels = img.reshape(-1, 3).astype(np.float32)
        mean_colors = np.mean(pixels, axis=0)
        std_colors = np.std(pixels, axis=0)

        all_features.append({
            'color_hist': color_hist,
            'mean_colors': mean_colors,
            'std_colors': std_colors
        })

        # Show original image
        axes[0,i].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        axes[0,i].set_title(titles[i], fontweight='bold')
        axes[0,i].axis('off')

        # Show color histogram
        axes[1,i].bar(range(len(color_hist)), color_hist, color='steelblue',
alpha=0.7)
        axes[1,i].set_title(f'Histogram Warna')
        axes[1,i].set_ylabel('Frequency' if i == 0 else '')

        # Show color statistics
        x_pos = ['B', 'G', 'R']
        axes[2,i].bar(x_pos, mean_colors, alpha=0.7, color=['blue', 'green',
'red'])
        axes[2,i].set_title('Mean Colors')
        axes[2,i].set_ylabel('Mean Value' if i == 0 else '')

    plt.tight_layout()
    plt.show()

    # Print comparison
    print("\n=== PERBANDINGAN FITUR WARNA ===")
    for i, (path, features) in enumerate(zip(image_paths, all_features)):
        print(f"\n{titles[i]} ({path}):")
        print(f"  Mean BGR: {features['mean_colors']}")
        print(f"  Std BGR:  {features['std_colors']}")
        print(f"  Color entropy: {-np.sum(features['color_hist'] *
np.log(features['color_hist'] + 1e-8)):.4f}")
```

```
def analyze_color_feature_importance(self):
    """Analisis pentingnya fitur warna dalam sistem"""
    print("\n=== ANALISIS PENTINGNYA FITUR WARNA ===")

    # Identifikasi komponen fitur
    lbp_size = 18
    run_length_size = 1
    color_hist_size = self.n_color_clusters
    mean_color_size = 3
    std_color_size = 3

    # Extract color features only
    color_features = self.image_descriptors[:, lbp_size + run_length_size:]
    color_hist_features = color_features[:, :color_hist_size]
    mean_color_features = color_features[:, color_hist_size:color_hist_size +
mean_color_size]
    std_color_features = color_features[:, color_hist_size + mean_color_size:]

    print(f"Total fitur warna: {color_features.shape[1]}")
    print(f"  - Color Histogram: {color_hist_size} fitur")
    print(f"  - Mean Colors: {mean_color_size} fitur")
    print(f"  - Std Colors: {std_color_size} fitur")

    # Analyze variance and discriminative power
    color_hist_var = np.var(color_hist_features, axis=0)
    mean_color_var = np.var(mean_color_features, axis=0)
    std_color_var = np.var(std_color_features, axis=0)

    print(f"\nVariansi fitur:")
    print(f"  - Color Histogram: mean={np.mean(color_hist_var):.6f},
max={np.max(color_hist_var):.6f}")
    print(f"  - Mean Colors: {mean_color_var}")
    print(f"  - Std Colors: {std_color_var}")

    # Find most discriminative color clusters
    top_clusters = np.argsort(color_hist_var)[-10:][::-1]
    print(f"\n10 cluster warna paling diskriminatif: {top_clusters}")
    print(f"Variansi mereka: {color_hist_var[top_clusters]}")

# Tambahkan method ini ke class CBIRSystem
def add_color_visualization_methods(cbir_instance):
    """Helper function to add color visualization methods to existing CBIR
instance"""
    import types
```

```
    cbir_instance.visualize_color_features =
types.MethodType(visualize_color_features, cbir_instance)
    cbir_instance.compare_color_features =
types.MethodType(compare_color_features, cbir_instance)
    cbir_instance.analyze_color_feature_importance =
types.MethodType(analyze_color_feature_importance, cbir_instance)
```

18) Fungsi untuk Imlementasi Visualisasi Warna

```
# Contoh penggunaan:
# Setelah cbir.train_model()
add_color_visualization_methods(cbir)

# Visualisasi fitur warna
cbir.visualize_color_features("query_patch_lbp2.jpg", save_analysis=True)

# Analisis pentingnya fitur warna
cbir.analyze_color_feature_importance()

# Bandingkan fitur warna
paths = ["./Colored_Brodatz/D19_COLORED.tif",
"./Colored_Brodatz/D12_COLORED.tif"]
cbir.compare_color_features(paths, ["Texture 1", "Texture 2"])
```