Digital Image Processing (ECE501)

Weekly Report 5

Team Members:

Student Name	Enroll Number
R Priscilla	AU2340001
Pratiksha Dongare	AU2340123
Shreya Dhumal	AU2340123
Krissa Gandhi	AU2340055

Introduction

This week have worked on improving our Content-based Image Retrieval (CBIR) system for chest X-ray images. The main focus was on feature analysis, normalization, precision evaluation and error handling with timers. The updated retrieval system uses cosine similarity instead of Euclidean distance which one we used for the previous code and it measures the similarity between images, which better captures directional similarity in feature space. The system continues to extract Histogram, GLCM, LBP, and statistical features to capture both local and global characteristics of X-ray images.

Objective

- Integration of Cosine Similarity inplace of Euclidean distance as the metric for computing similarity scores between extracted from chest X-ray images from data.
- To maintain the weighted feature extraction way to balance local and global image traits and analyse better.
- Evaluating system performance of data using Precision@K with the updated similarity metric.

Methodology

This time to more focus how the two similar x-ray images are, we are using cosine function instead of Euclidean distance for this time. So we will follow the same process as we did for euclidean distance but instead of that we are using cosine similarity. So the process are the same

1. Image Preprocessing: Each X-ray image is first converted to grayscale and then resized to 256x256 pixels. The image is enhanced using CLAHE (Contrast Limited Adaptive Histogram Equalization) to improve local contrast and highlight finer details.

2. Feature Extraction:

● Histogram Features: Represents the global intensity distribution of the image. Computed using 64 bins to capture pixel intensity variations. Output Size: 64

● GLCM Features: Captures texture patterns using four directions — 0°, 45°, 90°, and 135°. Extracts contrast, correlation, energy, and homogeneity from the GLCM. Output Size: 4

● LBP Features: Describes fine local texture details in the image by computation using parameters P = 8 (neighbors) and R = 1 (radius). A histogram of 10 bins is used to represent local texture patterns. Output Size: 64

● Statistical Features: Measures intensity-level statistics such as mean, standard deviation and variance. Represents global brightness and contrast variability. Output Size:

3 ● Feature Standardization and Weighting: The process of feature standardization is used to make sure that all features have an equal contribution by adjusting their ranges. This is followed by the implementation of feature weighting, which elevates the importance of the essential traits such as texture and contrast, thus enhancing the quality of retrieval.

4. Similarity computation (cosine similarity): for this time instead of euclidean distance we are using cosine to calculate the distance to identify how the 2 images are similar to each other
It is defined as

$$S(i,j) = xi. xj \ / \ |xi| . |xj|$$

4. Rank Retrieval: After computing all distance will sort in ascending order. The lowest distance of a images will be the similar to the query image.

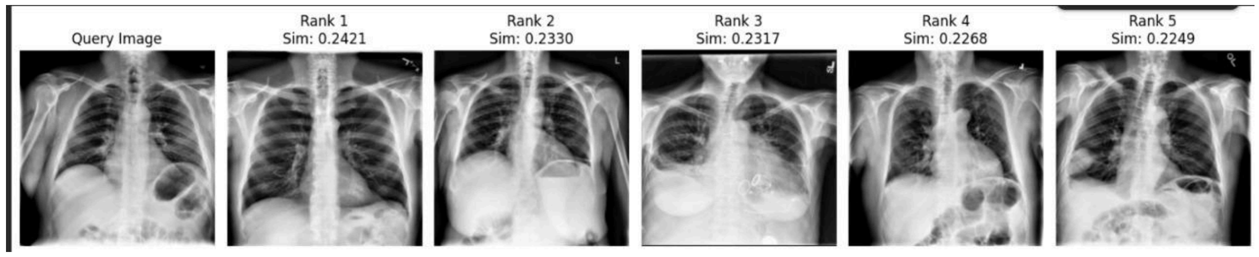
This time we are just focused on how we implement of cosine similarity will work on the code and implementation in the next time. Will follow the same process. As we described this time.

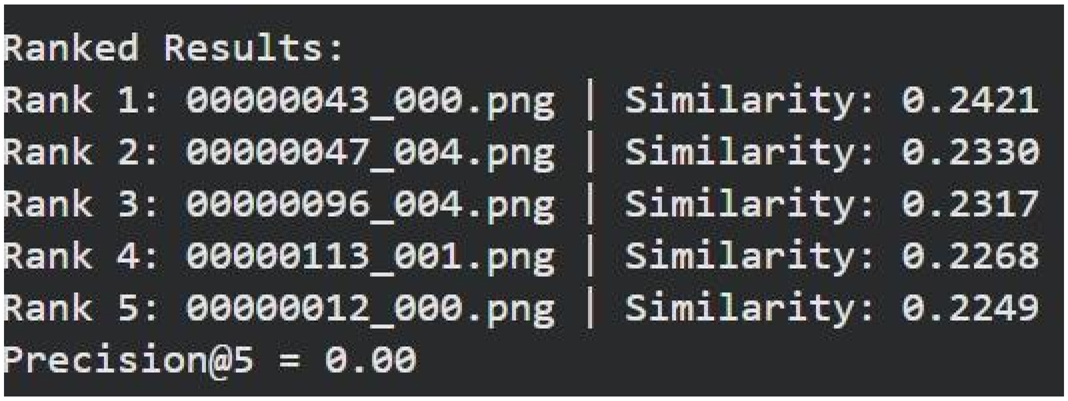
Code Implementation:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import graycomatrix, graycoprops, local_binary_pattern
from sklearn.preprocessing import normalize, StandardScaler
from concurrent.futures import ThreadPoolExecutor, as_completed
from google.colab import drive, files
import time

drive.mount('/content/drive', force_remount=True)

dataset_dir = "/content/drive/MyDrive/images"
features_file = "features.npy"
paths_file = "paths.npy"


def preprocess_image(img_path, size=(256, 256)):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError(f"Cannot read {img_path}")
    img = cv2.resize(img, size, interpolation=cv2.INTER_AREA)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    return clahe.apply(img)
```

```
def extract_histogram_features(img, bins=64):
    hist = cv2.calcHist([img], [0], None, [bins], [0, 256])
    return cv2.normalize(hist, None).flatten()

def extract_glcm_features(img):
    glcm = graycomatrix(img, [1], [0, np.pi/4, np.pi/2, 3*np.pi/4],
                        levels=256, symmetric=True, normed=True)
    props = ['contrast', 'correlation', 'energy', 'homogeneity']
    return np.array([graycoprops(glcm, p).mean() for p in props])

def extract_lbp_features(img, P=8, R=1, bins=10):
    lbp = local_binary_pattern(img, P, R, method='uniform')
    hist, _ = np.histogram(lbp.ravel(), bins=bins, range=(0, bins))
    return cv2.normalize(hist.astype('float'), None).flatten()

def extract_statistical_features(img):
    mean, std, var = np.mean(img), np.std(img), np.var(img)
    return np.array([mean, std, var])


def extract_features(img_path):
    img = preprocess_image(img_path)
    f_hist = extract_histogram_features(img) * 0.4
    f_glcm = extract_glcm_features(img) * 0.3
    f_lbp = extract_lbp_features(img) * 0.2
    f_stat = extract_statistical_features(img) * 0.1
    return np.concatenate([f_hist, f_glcm, f_lbp, f_stat])


if os.path.exists(features_file) and os.path.exists(paths_file):
    feature_database = np.load(features_file)
    image_paths = np.load(paths_file)
else:
    image_paths = [os.path.join(dataset_dir, f) for f in os.listdir(dataset_dir)
                   if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
    feature_database = []
    start = time.time()
    with ThreadPoolExecutor(max_workers=8) as executor:
        futures = [executor.submit(extract_features, p) for p in image_paths]
        for f in as_completed(futures):
            try:
                feature_database.append(f.result())
            except Exception as e:
                print("Error:", e)
    feature_database = np.array(feature_database)

    scaler = StandardScaler()
    feature_database = scaler.fit_transform(feature_database)
    feature_database = normalize(feature_database)

    np.save(features_file, feature_database)
    np.save(paths_file, np.array(image_paths))
    print(f"Feature extraction done in {time.time() - start:.2f} seconds.")

def calculate_similarity(query_features, database_features):
    query_norm = np.linalg.norm(query_features)
    db_norms = np.linalg.norm(database_features, axis=1)
    dot_products = database_features @ query_features
    cosine_similarities = dot_products / (db_norms * query_norm + 1e-10)  # to avoid division by
zero
    return cosine_similarities

def retrieve_similar_images(query_path, top_k=5):
    query_features = extract_features(query_path)
    query_features = query_features / np.linalg.norm(query_features)
    similarities = calculate_similarity(query_features, feature_database)
    top_indices = np.argsort(similarities)[::-1][:top_k]  # descending order of similarity

    fig, axes = plt.subplots(1, top_k + 1, figsize=(15, 3))
    query_img = preprocess_image(query_path)
    axes[0].imshow(query_img, cmap='gray')
    axes[0].set_title("Query Image")
    axes[0].axis('off')

    results = []
    for rank, idx in enumerate(top_indices, start=1):
        img = preprocess_image(image_paths[idx])
        axes[rank].imshow(img, cmap='gray')
        axes[rank].set_title(f"Rank {rank}\nSim: {similarities[idx]:.4f}")
        axes[rank].axis('off')
        results.append((rank, image_paths[idx], similarities[idx]))

    plt.tight_layout()
    plt.show()
    return results

def evaluate_precision_at_k(results, query_label, k=5):
    top_k_labels = [os.path.basename(p).split('_')[0] for _, p, _ in results[:k]]
    correct = np.mean([lbl == query_label for lbl in top_k_labels])
    return correct

print("Add a query test chest X-ray image")
query_uploaded = files.upload()
query_path = list(query_uploaded.keys())[0]

results = retrieve_similar_images(query_path, top_k=5)

print("\nRanked Results:")
for rank, path, sim in results:
    print(f"Rank {rank}: {os.path.basename(path)} | Similarity: {sim:.4f}")

query_label = os.path.basename(query_path).split('_')[0]
precision = evaluate_precision_at_k(results, query_label, k=5)
print(f"Precision@5 = {precision:.2f}")
```

Results





```
Ranked Results:
Rank 1: 00000043_000.png | Similarity: 0.2421
Rank 2: 00000047_004.png | Similarity: 0.2330
Rank 3: 00000096_004.png | Similarity: 0.2317
Rank 4: 00000113_001.png | Similarity: 0.2268
Rank 5: 00000012_000.png | Similarity: 0.2249
Precision@5 = 0.00
```

Next Set of Work:

1. Optimize Your Feature Weights Automatically

- Run experiments with different weight combinations and find which combination gives the best retrieval accuracy (highest Precision@K).
- For example try (0.3, 0.4, 0.2, 0.1) or (0.5, 0.2, 0.2, 0.1) and measure of which one retrieves more correct chest X-rays at the top.

2. Add Deep Learning Features (Using Pre-trained Models)

- Use a pretrained CNN model (like ResNet50 or VGG16) to automatically extract high-level features from chest X-rays of large data, then combine them with existing features.
- Why it helps: Deep learning features capture complex medical patterns that simple handcrafted features miss, making retrieval much more accurate.
- Simple example: Your handcrafted features capture basic texture; deep learning captures anatomical abnormalities, diseases, and subtle patterns that are clinically relevant.

3. Use User Feedback to Improve Results

- Allow users to mark which retrieved images are correct/incorrect, then update weights or adjust the query based on this feedback to get better results in the next search.
- For example, If user says "this retrieved image is wrong," the system can reduce the influence of the features that matched it and re-rank the results.

References

- https://www.kaggle.com/datasets/nih-chest-xrays/data
- https://onyekaokonji.medium.com/cosine-similarity-measuring-similarity-between-multiple-images-f289aaf40c2b
- ARTICLE - Cosine similarity measures for intuitionistic fuzzy sets and their applications https://www.sciencedirect.com/science/article/pii/S0895717710003651