Digital Image Processing (ECE501)

Weekly Report 2

Team Members:

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

Introduction:

For this week, our focus is to implement Euclidean distance for similarity images which is basically a method to measure how similar two images are by considering their pixel data as points in a multidimensional space. This will measure similarity between X-ray images based on extracted features (Histogram and GLCM features). This method is very beneficial in building CBIR system with better accuracy.

Objective

- Develop and integrate code to compute Euclidean distance between combined histogram and GLCM feature vectors of chest X-ray images.
- Automate the feature extraction, normalization and caching to improve performance on large datasets.
- Utilizing multithreading for better processing of multiple number of X-ray images.
- Making a retrieval pipeline that allows uploading query image and then retrieving the most similar X-ray images visually.

Methodology

The workflow developed includes the following steps:

- Loading and Feature Extraction:
  Chest X-ray images are loaded in grayscale format and preprocessed. For each image, normalized histograms (256 bins) capturing intensity distribution and GLCM texture features (contrast, energy, homogeneity, correlation) are computed. These features are concatenated into a single feature vector for each image.

- Feature Extraction:
  1. Histogram Features: Represent pixel intensity distribution using 64 bins.
  2. GLCM (Gray Level Co-occurrence Matrix) Features:
  3. Retrieve texture patterns with four directions (0°, 45°, 90°, 135°) with distance=1 pixel. GLCM characteristics including contrast, correlation, and homogeneity measure texture smoothness and intensity relationships.

- Euclidean Distance Calculation
  Euclidean distance is used to measure the closeness of two X-ray images which are in the feature space.
  The distance formula is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

Code Implementation:

```
def calculate_similarity(query_features, database_features):
    query_norm = np.sum(query_features ** 2)
    db_norms = np.sum(database_features ** 2, axis=1)
    dot_products = database_features @ query_features
    distances = np.sqrt(np.maximum(db_norms + query_norm - 2 * dot_products, 0))
    return distances
```

Code

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

print("Mounting Google Drive...")
drive.mount('/content/drive')

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}")
    return cv2.resize(img, size, interpolation=cv2.INTER_AREA)

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

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

def extract_features(img_path):
    img = preprocess_image(img_path)
    hist_features = extract_histogram_features(img)
    glcm_features = extract_glcm_features(img)
    return np.concatenate([hist_features, glcm_features])
```

```
if os.path.exists(features_file) and os.path.exists(paths_file):
    print("\n Found existing feature files — loading them...")
    feature_database = np.load(features_file)
    image_paths = np.load(paths_file)
    print(f"Loaded {len(image_paths)} feature vectors from cache.")
else:
    print("\n⚙ No cached features found — processing dataset...")
    image_paths = [os.path.join(dataset_dir, f) for f in os.listdir(dataset_dir)
                   if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
    print(f"Processing {len(image_paths)} images...")
    start_time = time.time()
    feature_database = []
    with ThreadPoolExecutor(max_workers=8) as executor:
        futures = {executor.submit(extract_features, path): path for path in image_paths}
        for future in as_completed(futures):
            path = futures[future]
            try:
                feature_database.append(future.result())
            except Exception as e:
                print(f"Error processing {path}: {e}")
    feature_database = np.array(feature_database)
    feature_database = normalize(feature_database)
    np.save(features_file, feature_database)
    np.save(paths_file, np.array(image_paths))
    print(f"\n Feature extraction done in {time.time() - start_time:.2f} seconds.")
    print("Features cached for next run.")

print(f"Feature database shape: {feature_database.shape}")

def calculate_similarity(query_features, database_features):
    query_norm = np.sum(query_features ** 2)
    db_norms = np.sum(database_features ** 2, axis=1)
    dot_products = database_features @ query_features
    distances = np.sqrt(np.maximum(db_norms + query_norm - 2 * dot_products, 0))
    return distances

def retrieve_similar_images(query_path, top_k=5):
    query_features = extract_features(query_path)
    query_features = query_features / np.linalg.norm(query_features)
    distances = calculate_similarity(query_features, feature_database)
```

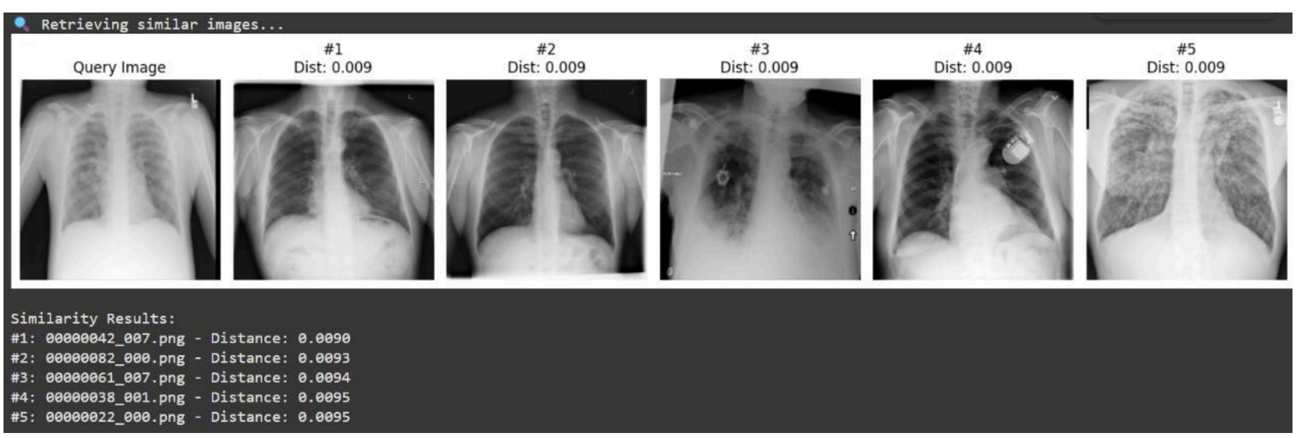
```
    top_indices = np.argsort(distances)[:top_k]
    fig, axes = plt.subplots(1, top_k + 1, figsize=(15, 3))
    query_img = preprocess_image(query_path)
    axes[0].imshow(query_img, cmap='gray')
    axes[0].set_title("Query Image")
    axes[0].axis('off')
    for i, idx in enumerate(top_indices):
        img = preprocess_image(image_paths[idx])
        axes[i + 1].imshow(img, cmap='gray')
        axes[i + 1].set_title(f"#{i+1}\nDist: {distances[idx]:.3f}")
        axes[i + 1].axis('off')
    plt.tight_layout()
    plt.show()
    return [(image_paths[idx], distances[idx]) for idx in top_indices]

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

print("\n Retrieving similar images...")
results = retrieve_similar_images(query_path, top_k=5)

print("\nSimilarity Results:")
for i, (path, dist) in enumerate(results):
    print(f"#{i+1}: {os.path.basename(path)} - Distance: {dist:.4f}")
```

Result



Next set of Work

1. Retrieve and Rank Top Similar Images:
   Sort the images based on ascending Euclidean distance values. Images with the smallest distances are considered the most visually similar to the query image. This ranking allows clear identification of the closest matches in terms of intensity and texture features.
2. Sort Images to Identify Similar Ones:
   By sorting the distance values, the system can easily retrieve the nearest neighbors images that lie closest in the multi-dimensional feature space.
3. Analyze Retrieval Performance Using Precision and Recall:
   Precision measures the fraction of retrieved images that are relevant, while recall measures the proportion of all relevant images that are retrieved.

References

- https://www.kaggle.com/datasets/nih-chest-xrays/data
- https://www.researchgate.net/figure/Example-of-CBIR-using-the-combination-of-shape-histogram-and-moment-based-features_fig6_6487005
- https://share.google/e8woXFD2PXG4vskwA