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:

For this week, we worked on the statistical features like mean, variance, skewness, kurtosis, etc, to enhance the intensity distribution of pixels. This will tell the brightness, texture, and contrast to help find out and differentiate the abnormal and normal x-ray images and also worked on the rank retrieval process. Where we sort the images in ascending order on the basis of the euclidean distance that we worked on last week. We are also given the numbers like 1, 2, 3 for the ranks. The small distance indicates the closest or most similar images to the query/input images.

Objective

- To enhance and build a strong CBIR pipeline by combining multiple image descriptors like Histogram, GLCM features, Euclidean distance, and then statistical features like kurtosis.
- Computation of Euclidean-based similarity between query and database images.
- To retrieve and rank top similar X-ray images.
- To visualize ranked results and then prepare a dataset for performance evaluation using large-scale image retrieval.

Methodology

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.

3. **Feature Caching and Parallelization**
   The features for all images from the dataset are calculated at the same time by using Python's ThreadPoolExecutor.
   Once all features are extracted then they are normalized and saved into two files:
   - features.npy = stores all the feature values.
   - paths.npy = stores the image file paths.

Moreover, caching keeps the system from performing additional hard work on redundant computation from preceding runs; hence, it helps in efficiently reducing processing times.

4. **Rank Retrieval**

   After computing all distances, they are sorted in ascending order. The images with the lowest distances are the most similar to the query image(to check similar ones).
   The Top-K images (here K = 5) are selected and displayed with:

   - Their rank - 1 being most similar
   - Their distance value shows how close they are to the query image.

In the end, Precision@K is calculated to assess the accuracy of retrieval, which to some extent facilitates and quantifies performance comparison.

Code Implementation:

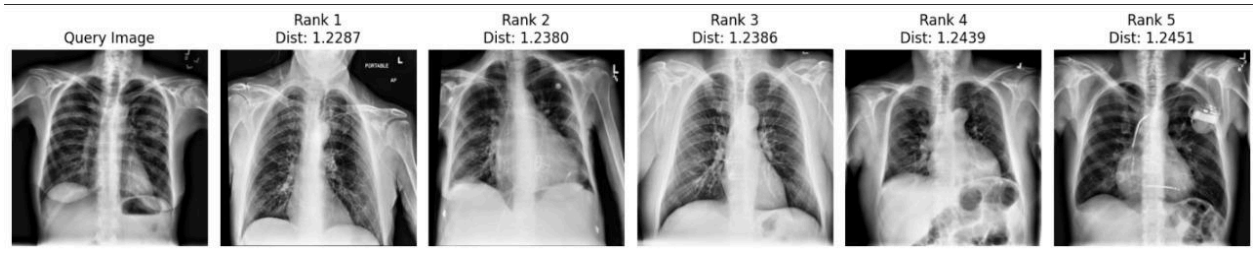
```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import graycomatrix, graycoprops, local_binary_pattern
from sklearn.preprocessing import normalize, StandardScaler
from concurrent.futures import ThreadPoolExecutor, as_completed
from google.colab import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

    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}\nDist: {distances[idx]:.4f}")
```

```
        axes[rank].axis('off')
        results.append((rank, image_paths[idx], distances[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, dist in results:
    print(f"Rank {rank}: {os.path.basename(path)} | Distance: {dist:.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: 00000042_001.png | Distance: 1.2287
Rank 2: 00000096_001.png | Distance: 1.2380
Rank 3: 00000057_003.png | Distance: 1.2386
Rank 4: 00000113_001.png | Distance: 1.2439
Rank 5: 00000013_022.png | Distance: 1.2451
Precision@5 = 0.00
```

Next Set of Work:

1. Analyze Retrieval Performance Using Precision and Recall:Next week , we will evaluate our rank retrieval process using precision and recall to see how well our project performs.
2. Try to text different distance metrics for better performance: We already worked on Euclidean distance and are gonna try to use other metrics like Manhattan distance, cosine similarity, and Chi-square distance.
3. Juggle weights in such a manner as to reflect uniformity across the features to achieve an objective of finding the top-M recall performances of various weights. Include all measurements, including those displayed in the screenshots, in the final report.

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