Digital Image Processing (ECE501)

Weekly Report 5

Project Title: Content-Based Image Retrieval (CBIR) for Chest X-Ray Images

Team Members:

Name of Student	Enrollment Number
R Priscilla	AU2340001
Pratiskha Dongare	AU2340123
Shreya Dhumal	AU2340124
Krissa Gandhi	AU2340055

Introduction

For the last week of the project, we are utilizing multiple similarity methods to build a CBIR pipeline. We have included various statistical features such as mean, variance, skewness and kurtosis to better enhance the intensity distribution of pixels in X-ray images. The statistical methods describe attributes such as texture and contrast.
We have imported a query and uploaded data to compare the similarity between the two. Additionally, we worked on rank retrieval in the previous code, which basically sorts images in ascending order based on their Euclidean distance to the query image.
This approach enhances the accuracy of content-based image retrieval (CBIR) for chest X-ray images by utilizing statistical image features in conjunction with other methods to measure the similarity between images. These pipelined projects will be helpful in the medical industry for detecting similar types of diagnoses.

Objective

- The primary objective is to develop a fully functional CBIR pipeline project for image similarity analysis.
- The system uses various types of image features, including Histogram to capture pixel intensity distribution, GLCM features and Gabor filters to get texture patterns.

- The system compares images using several similarity matrices: Euclidean distance measures the difference between feature vectors. Cosine similarity measures the angle between images and Chi-square is used to measure statistical differences.

Methodology

This last weelkpy reports, includes final implementation of the project which is fully working and uses various functions and methods for calculating similarity among query images and data chest x-ray images. We have implemented methods below in code for the project:

Preprocessing

- It resizes all the images to a fixed uniform size of (256x256 pixels).
- Contrast Limited Adaptive Histogram Equalization is applied to enhance the contrast to improve texture visibility in images.

Feature Extraction

- Histogram Features: Gray level histogram representing pixel intensity distribution then it is normalized to be scale-invariant.
- GLCM Features: Texture features from four angles (0°, 45°, 90°, 135°) describes contrast, correlation, energy and homogeneity.
- Gabor Features: Captures frequency and orientation-specific texture information. Mean and variance are calculated for each filtered image.
- Wavelet Features: The discrete wavelet transform decomposes the image and statistical measures of the coefficients capture texture at multiple scales.
- All these features are combined into one vector to represent image.

Similarity Metrics

- Three similarity metrics to compare the query image features with the dataset:
  - Euclidean Distance: Standard L2 norm distance, smaller values mean more similarity.
  - Cosine Distance: Measures angular difference between feature vectors; converted to a distance-like measure.
  - Chi-square Distance: A statistical measure suited to histogram-like data, sensitive to relative differences.

Retrieval & Evaluation

- Feature extraction and normalization are done on a given query image.
- Similarity distances to all database features are computed.

- The system selects and displays the top-K most similar images based on the chosen metric, together with the query.
- The retrieval time is measured and displayed.

Precision@K

- A simple precision metric is calculated as how many of the top-K retrieved images belong to the same class as the query image.
- This assumes that the ground truth class name is part of the folder name of the query image.

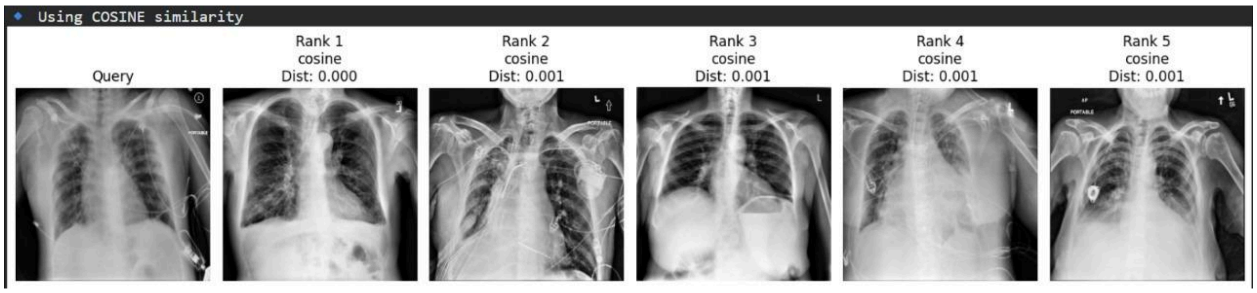
Code Implementation:

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

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

dataset_dir = "/content/drive/MyDrive/images"
features_file = "features_v2.npy"
paths_file = "paths_v2.npy"


# Preprocessing
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)
```

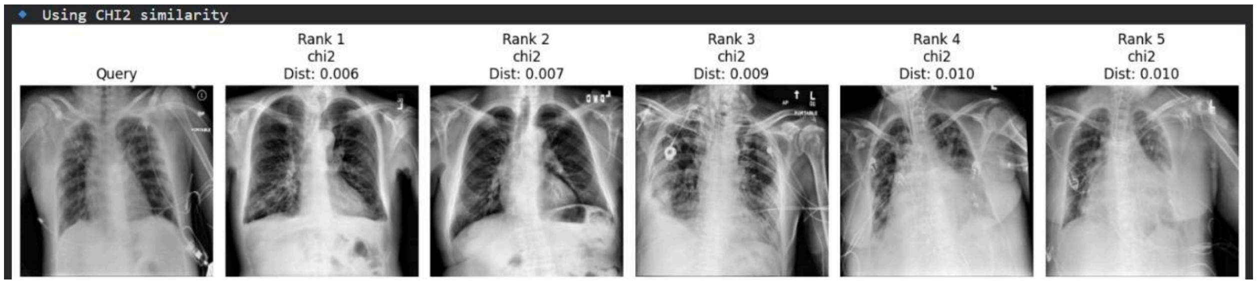
```
# Feature Extraction
def extract_histogram_features(img, bins=64):
    hist = cv2.calcHist([img], [0], None, [bins], [0, 256])
    return cv2.normalize(hist, None).flatten()

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_gabor_features(img):
    thetas = [0, np.pi/4, np.pi/2, 3*np.pi/4]
    features = []
    for theta in thetas:
        kernel = cv2.getGaborKernel((21, 21), 4.0, theta, 10.0, 0.5, 0, ktype=cv2.CV_32F)
        fimg = cv2.filter2D(img, cv2.CV_8UC3, kernel)
        features.append(fimg.mean())
        features.append(fimg.var())
    return np.array(features)

def extract_wavelet_features(img):
    coeffs = pywt.wavedec2(img, 'db1', level=2)
    features = []
    for c in coeffs:
        if isinstance(c, tuple):
            for arr in c:
                features.append(np.mean(arr))
                features.append(np.std(arr))
        else:
            features.append(np.mean(c))
            features.append(np.std(c))
    return np.array(features)

def extract_features(img_path):
    img = preprocess_image(img_path)
    f_hist = extract_histogram_features(img)
    f_glcm = extract_glcm_features(img)
    f_gabor = extract_gabor_features(img)
    f_wavelet = extract_wavelet_features(img)
    return np.concatenate([f_hist, f_glcm, f_gabor, f_wavelet])

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 = []
    for p in image_paths:
        try:
            feature_database.append(extract_features(p))
        except Exception as e:
            print("Error:", e)
    feature_database = normalize(np.array(feature_database))
    np.save(features_file, feature_database)
    np.save(paths_file, np.array(image_paths))

# Similarity Metrics
def calculate_similarity(query_features, db_features, metric='euclidean'):
    if metric == 'euclidean':
        return np.linalg.norm(db_features - query_features, axis=1)
    elif metric == 'cosine':
        sims = np.dot(db_features, query_features) / (np.linalg.norm(db_features, axis=1) *
np.linalg.norm(query_features))
        return 1 - sims  # smaller = more similar
    elif metric == 'chi2':
        return np.sum((db_features - query_features)**2 / (db_features + query_features + 1e-10),
axis=1)
    else:
        raise ValueError("Unknown metric")

# Retrieval & Evaluation
def retrieve_similar_images(query_path, top_k=5, metric='euclidean'):
    query_features = extract_features(query_path)
    query_features = query_features / np.linalg.norm(query_features)
    start = time.time()
    distances = calculate_similarity(query_features, feature_database, metric=metric)
    elapsed = time.time() - start
    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")
    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}\n{metric}\nDist: {distances[idx]:.3f}")
        axes[rank].axis('off')
        results.append((rank, image_paths[idx], distances[idx]))
    plt.tight_layout()
    plt.show()

    return results, elapsed

# Precision@K
def precision_at_k(results, ground_truth_class):
    correct = sum(ground_truth_class in os.path.basename(p) for _, p, _ in results)
    return correct / len(results)

print("Upload a query image ")
query_uploaded = files.upload()
query_path = list(query_uploaded.keys())[0]

metrics = ['euclidean', 'cosine', 'chi2']
precisions = []
times = []

for m in metrics:
    print(f"\n  Using {m.upper()} similarity")
    results, elapsed = retrieve_similar_images(query_path, top_k=5, metric=m)
    gt_class = os.path.basename(os.path.dirname(query_path))
    prec = precision_at_k(results, gt_class)
    precisions.append(prec)
    times.append(elapsed)
    print(f"Precision@5: {prec:.2f} | Time: {elapsed:.3f}s")
```

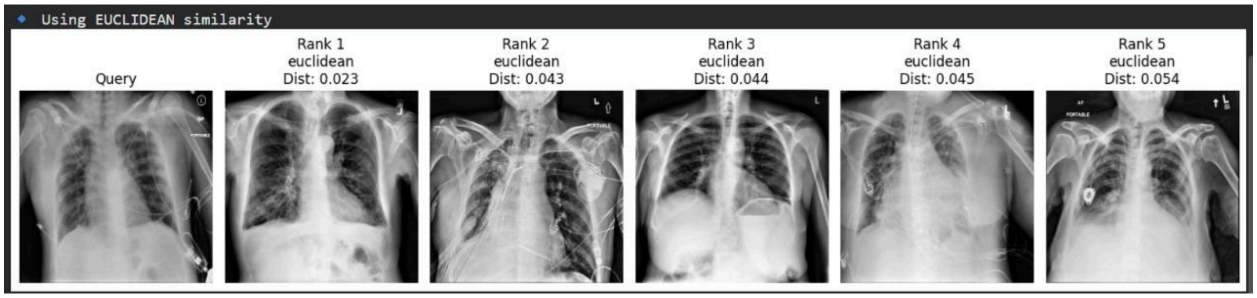
```
plt.figure(figsize=(6,4))
plt.bar(metrics, precisions, color=['#2e8b57','#1e90ff','#ff7f50'])
plt.title("Precision@5 across Similarity Metrics")
plt.ylabel("Precision")
plt.grid(True, axis='y')
plt.show()

plt.figure(figsize=(6,4))
plt.bar(metrics, times, color=['#6a5acd','#ff6347','#3cb371'])
plt.title("Retrieval Time Comparison")
plt.ylabel("Time (seconds)")
plt.grid(True, axis='y')
plt.show()
```

Results

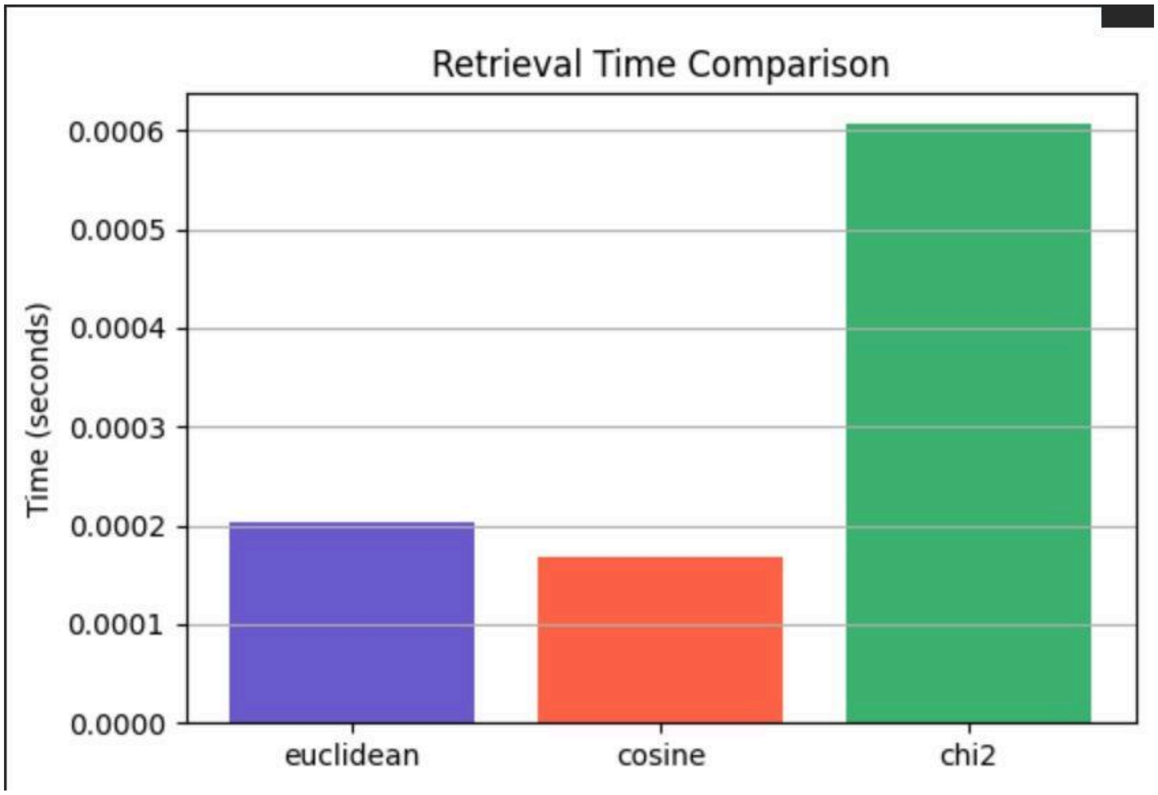


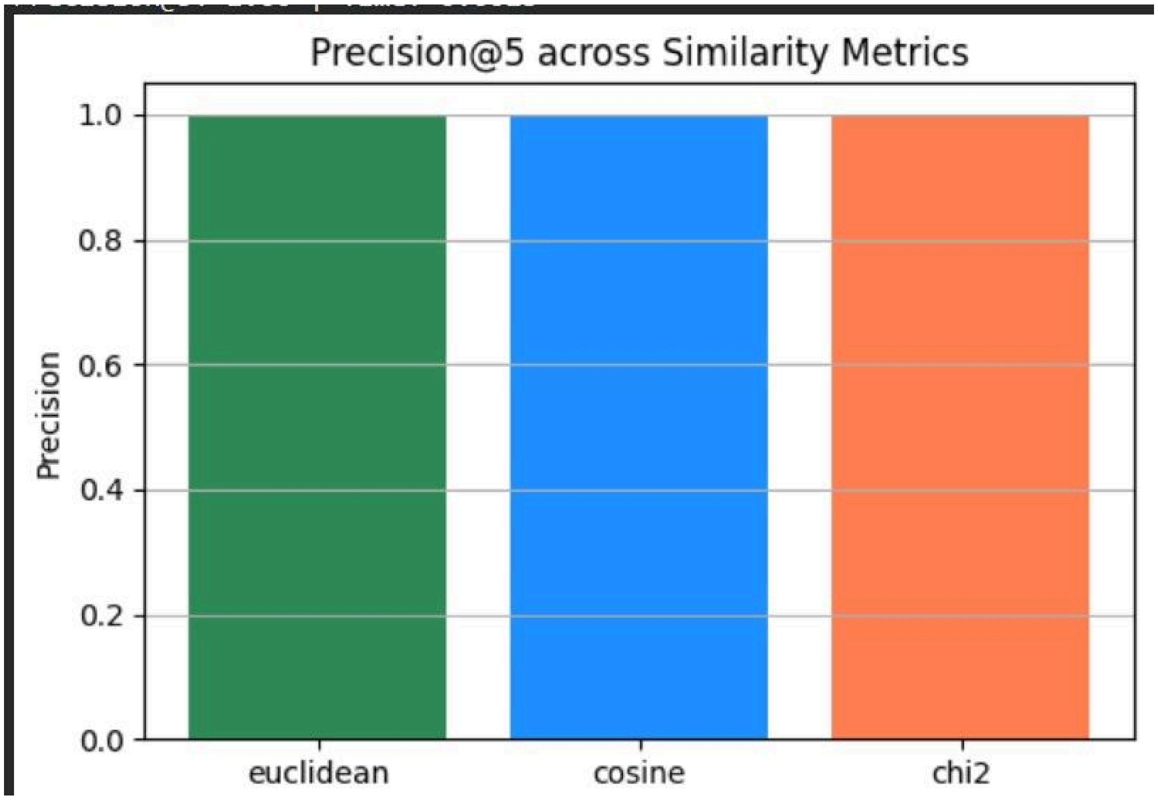
Using COSINE similarity
Query
Rank 1 cosine Dist: 0.000
Rank 2 cosine Dist: 0.001
Rank 3 cosine Dist: 0.001
Rank 4 cosine Dist: 0.001
Rank 5 cosine Dist: 0.001





Using EUCLIDEAN similarity
Query
Rank 1 euclidean Dist: 0.023
Rank 2 euclidean Dist: 0.043
Rank 3 euclidean Dist: 0.044
Rank 4 euclidean Dist: 0.045
Rank 5 euclidean Dist: 0.054


Retrieval Time Comparison
Time (seconds)
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
euclidean
cosine
chi2


Precision@5 across Similarity Metrics
Precision
1.0
0.8
0.6
0.4
0.2
0.0
euclidean
cosine
chi2

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