Name Surname= SelimCanYazar

StudentID= 152120191023

HW 3 Report

```python
import zipfile
import os
import numpy as np
from PIL import Image

# Define the path to the zip file and the extraction path
zip_path = '/content/HWData.zip'
extraction_path = '/content/HWData'

# Create the extraction path directory if it doesn't exist
if not os.path.exists(extraction_path):
    os.makedirs(extraction_path)

# Unzip the file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)

print(f"Files extracted to {extraction_path}")

# Update dataset_dir to point to the correct sub-directory where the images are stored
dataset_dir = '/content/HWData/HWData/train'  # Adjust if necessary

# Dictionary to store image data, with keys as class names and values as the matrix of image vectors
class_images = {}

# Loop through each directory in the dataset directory to process each class
for class_dir in os.listdir(dataset_dir):
    class_path = os.path.join(dataset_dir, class_dir)
```

```python
    if os.path.isdir(class_path):
        # List to hold image vectors for this class
        image_vectors = []

        # Process each image in the directory
        for image_file in os.listdir(class_path):
            image_path = os.path.join(class_path, image_file)
            # Check if the file is an image
            if image_file.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.tiff')):
                try:
                    with Image.open(image_path) as img:
                        # Resize the image to 64x64 pixels
                        img_resized = img.resize((64, 64))

                        # Convert the image to grayscale, if it's not already
                        img_gray = img_resized.convert('L')

                        # Flatten the image to create a vector and append to the list
                        image_vector = np.array(img_gray).flatten()
                        image_vectors.append(image_vector)
                except Exception as e:
                    print(f"Error processing image {image_file}: {e}")

        # Convert list of vectors to a numpy array and transpose it to match 4096xn format
        if image_vectors:
            class_images[class_dir] = np.transpose(np.array(image_vectors))
            print(f"Class {class_dir}, Matrix shape: {class_images[class_dir].shape}")

# Checking if no images were processed for any class
if not class_images:
```

```
Files extracted to /content/HWData
Class brain, Matrix shape: (4096, 21)
Class BACKGROUND_Google, Matrix shape: (4096, 94)
Class barrel, Matrix shape: (4096, 10)
Class accordion, Matrix shape: (4096, 12)
Class camera, Matrix shape: (4096, 11)
Class bonsai, Matrix shape: (4096, 27)
Class binocular, Matrix shape: (4096, 8)
Class beaver, Matrix shape: (4096, 10)
Class anchor, Matrix shape: (4096, 9)
Class butterfly, Matrix shape: (4096, 19)
Class ant, Matrix shape: (4096, 9)
Class airplanes, Matrix shape: (4096, 21)
Class brontosaurus, Matrix shape: (4096, 10)
Class buddha, Matrix shape: (4096, 18)
Class bass, Matrix shape: (4096, 12)
```

Resim verilerini içeren bir zip dosyasını çıkararak işlemektedir. İlk adımda gerekli kütüphaneler (zipfile, os, numpy, PIL) import edilir. Zip dosyasının yolunu (zip_path) ve çıkartılacak konumu (extraction_path) belirlenir.

Zip dosyası çıkartma işlemi zipfile.ZipFile kullanılarak gerçekleştirilir. with bloğu içinde zip dosyası açılır ve extractall() metoduyla belirtilen konuma dosyalar çıkarılır.

Dosyaların çıkarıldığı konum (extraction_path) ekrana yazdırılır. Daha sonra dataset_dir değişkeni, işlenecek resimlerin bulunduğu dizini temsil eder. Bu varsayılan olarak /content/HWData/HWData/train olarak belirlenmiştir.

Bir sözlük olan class_images oluşturulur. Bu sözlük, sınıf isimlerini anahtar olarak alacak ve her sınıfa ait resim matrislerini değer olarak saklayacaktır.

Daha sonra dataset_dir içindeki her alt dizin (class_dir) üzerinde döngü oluşturulur. Her bir alt dizin class_path değişkeninde tam yol olarak belirtilir.

Eğer class_path bir dizinse, bu dizindeki her resim işlenir. Resim dosyalarının uzantılarına bakılarak resim dosyası olup olmadığı kontrol edilir. Resim

dosyasıysa, PIL kütüphanesi kullanılarak resim dosyası açılır ve belirli işlemler uygulanarak (boyutlandırma, siyah beyaz yapma) resim vektörleri oluşturulur.

Oluşturulan resim vektörleri image_vectors listesine eklenir. Eğer bir hata oluşursa (Exception), hata mesajı ekrana yazdırılır ancak işlem devam eder.

Son olarak, image_vectors listesi Numpy dizisine dönüştürülür ve transpoze edilerek belirli bir formata getirilir (4096xn). Bu işlem sonucunda elde edilen resim matrisi, ilgili sınıf adıyla birlikte class_images sözlüğüne eklenir.

Eğer hiçbir resim işlenmemişse (yani class_images boşsa), buna dair bir uyarı verilir. Bu betik, verilen zip dosyasından resim verilerini çıkararak, her sınıfa ait resim matrislerini oluşturur ve bu matrisleri class_images sözlüğünde saklar. Hatalı durumlarla ilgili bilgilendirme sağlar ve işlem sonucunu ekrana yazdırır.

```python
# Update dataset_dir to point to the test directory where the images are stored
dataset_dir = '/content/HWData/HWData/test'  # Adjust to the test directory path

# Dictionary to store image data for the test dataset, with keys as class names and values as the m
test_class_images = {}

# Loop through each directory in the test dataset directory to process each class
for class_dir in os.listdir(dataset_dir):
    class_path = os.path.join(dataset_dir, class_dir)
    if os.path.isdir(class_path):
        # List to hold image vectors for this class
        image_vectors = []

        # Process each image in the directory
        for image_file in os.listdir(class_path):
            image_path = os.path.join(class_path, image_file)
            # Check if the file is an image
            if image_file.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.tiff')):
                try:
                    with Image.open(image_path) as img:
                        # Resize the image to 64x64 pixels
                        img_resized = img.resize((64, 64))

                        # Convert the image to grayscale, if it's not already
                        img_gray = img_resized.convert('L')

                        # Flatten the image to create a vector and append to the list
                        image_vector = np.array(img_gray).flatten()
                        image_vectors.append(image_vector)
                except Exception as e:
                    print(f"Error processing image {image_file}: {e}")

        # Convert list of vectors to a numpy array and transpose it to match 4096xn format
        if image_vectors:
            test_class_images[class_dir] = np.transpose(np.array(image_vectors))
            print(f"Test Class {class_dir}, Matrix shape: {test_class_images[class_dir].shape}")

# Checking if no images were processed for any class in the test dataset
if not test_class_images:
    print("No test images were processed. Check the dataset directory and image file types.")
```

```
Test Class brain, Matrix shape: (4096, 8)
Test Class BACKGROUND_Google, Matrix shape: (4096, 46)
Test Class barrel, Matrix shape: (4096, 4)
Test Class accordion, Matrix shape: (4096, 4)
Test Class camera, Matrix shape: (4096, 4)
Test Class bonsai, Matrix shape: (4096, 11)
Test Class binocular, Matrix shape: (4096, 2)
Test Class beaver, Matrix shape: (4096, 4)
Test Class anchor, Matrix shape: (4096, 4)
Test Class butterfly, Matrix shape: (4096, 8)
Test Class ant, Matrix shape: (4096, 4)
Test Class airplanes, Matrix shape: (4096, 13)
Test Class brontosaurus, Matrix shape: (4096, 3)
Test Class buddha, Matrix shape: (4096, 8)
Test Class bass, Matrix shape: (4096, 4)
```

İlk olarak, dataset_dir değişkeni /content/HWData/HWData/test olarak ayarlanmıştır. Bu dizin, test veri setinin bulunduğu konumu temsil etmektedir.

Daha sonra test_class_images adında boş bir sözlük oluşturulur. Bu sözlük, her sınıfa ait resim matrislerini saklamak için kullanılacaktır.

Sonrasında, dataset_dir içindeki her alt dizini (class_dir) işlemek üzere bir döngü oluşturulur. Her bir alt dizin, class_path değişkeninde tam yol olarak temsil edilir.

Eğer class_path bir dizin ise, bu dizindeki her resim işlenir. Resim dosyalarının uzantılarına bakılarak resim dosyası olup olmadığı kontrol edilir. Resim dosyasıysa, PIL kütüphanesi kullanılarak resim dosyası açılır ve belirli işlemler uygulanarak (boyutlandırma, siyah beyaz yapma) resim vektörleri oluşturulur.

Oluşturulan resim vektörleri image_vectors adlı bir liste içinde saklanır. Eğer bir hata oluşursa (örneğin, resim dosyası açılamazsa), bu hata mesajı ekrana yazdırılır.

Daha sonra, image_vectors listesi Numpy dizisine dönüştürülür ve transpoze edilerek belirli bir formata getirilir (4096xN). Bu işlem sonucunda elde edilen resim matrisi, ilgili sınıf adıyla birlikte test_class_images sözlüğüne eklenir.

Son olarak, işlenen tüm resimlerin matrislerini içeren test_class_images sözlüğünün boş olup olmadığı kontrol edilir. Eğer hiçbir resim işlenmemişse, "No test images were processed. Check the dataset directory and image file types." şeklinde bir uyarı mesajı ekrana yazdırılır.

```python
import numpy as np
# Assume 'class_images' is the dictionary from Stage 1 containing 4096xn matrices for each class
pca_results = {}
for class_name, data_matrix in class_images.items():
    # PCA Step 1: Calculate the covariance matrix
    # Note: We need to transpose the data matrix to fit the shape requirements of cov function (observations as co
    covariance_matrix = np.cov(data_matrix, rowvar=False)

    # PCA Step 2: Calculate eigenvalues and eigenvectors
    eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

    # PCA Step 3: Sort the eigenvectors by descending eigenvalues
    # Get the indices of the sorted eigenvalues in descending order
    sorted_indices = np.argsort(eigenvalues)[::-1]

    # Select the top three eigenvectors
    top_eigenvectors = eigenvectors[:, sorted_indices[:3]]

    # Store the results in the dictionary
    pca_results[class_name] = {
        'eigenvalues': eigenvalues[sorted_indices[:3]],
        'eigenvectors': top_eigenvectors
    }

    # Print the shape of the matrix containing the top 3 eigenvectors for verification
    print(f"Class {class_name}: Eigenvectors matrix shape (should be nx3): {top_eigenvectors.shape}")

# Now pca_results contains the PCA results for each class, with the top 3 eigenvectors used to transform the data
```

```
Class brain: Eigenvectors matrix shape (should be nx3): (21, 3)
Class BACKGROUND_Google: Eigenvectors matrix shape (should be nx3): (94, 3)
Class barrel: Eigenvectors matrix shape (should be nx3): (10, 3)
Class accordion: Eigenvectors matrix shape (should be nx3): (12, 3)
Class camera: Eigenvectors matrix shape (should be nx3): (11, 3)
Class bonsai: Eigenvectors matrix shape (should be nx3): (27, 3)
Class binocular: Eigenvectors matrix shape (should be nx3): (8, 3)
Class beaver: Eigenvectors matrix shape (should be nx3): (10, 3)
Class anchor: Eigenvectors matrix shape (should be nx3): (9, 3)
Class butterfly: Eigenvectors matrix shape (should be nx3): (19, 3)
Class ant: Eigenvectors matrix shape (should be nx3): (9, 3)
Class airplanes: Eigenvectors matrix shape (should be nx3): (21, 3)
Class brontosaurus: Eigenvectors matrix shape (should be nx3): (10, 3)
Class buddha: Eigenvectors matrix shape (should be nx3): (18, 3)
Class bass: Eigenvectors matrix shape (should be nx3): (12, 3)
```

Burada PCA (Principal Component Analysis - Temel Bileşen Analizi) algoritmasını kullanarak her sınıf için resim verilerinin boyutunu azaltmayı amaçlamaktadır. İşlem adımları şu şekildedir: İlk olarak, pca_results adında boş bir sözlük oluşturulur. Bu sözlük, her sınıf için PCA sonuçlarını saklamak üzere kullanılacaktır. Daha sonra class_images sözlüğünden her sınıfın adı (class_name) ve o sınıfa ait resim matrisi (data_matrix) alınır. Bu matris, 4096xN boyutunda olup her bir sütunu bir resmi temsil eder. PCA Adımı 1'de, np.cov(data_matrix, rowvar=False) kullanılarak resim veri matrisinin kovaryans matrisi hesaplanır. rowvar=False parametresi, matrisin sütunlarının (yani her bir sütunun bir resmi temsil ettiği) gözlem olduğunu belirtir. PCA Adımı 2'de, np.linalg.eig(covariance_matrix) kullanılarak kovaryans matrisinin özdeğerleri (eigenvalues) ve özvektörleri (eigenvectors) hesaplanır. PCA Adımı 3'te, özdeğerlerin büyüklüklerine göre sıralanması için np.argsort(eigenvalues)[::-1] kullanılır. Bu işlem, özdeğerlerin sıralı indekslerini büyükten küçüğe doğru verir. En üstteki üç özvektör seçilir ve top_eigenvectors = eigenvectors[:, sorted_indices[:3]] kullanılarak sıralı indekslere göre en büyük üç özvektör

seçilir. Bu üç özvektör, verinin boyutunu azaltmak için kullanılacaktır. Son olarak, PCA sonuçları pca_results sözlüğüne kaydedilir. Her sınıf için, PCA sonuçları içinde üç en büyük özdeğer (eigenvalues) ve ona karşılık gelen özvektörler (top_eigenvectors) bulunmaktadır. Bu yöntem sayesinde veri boyutu azaltılarak analiz ve modelleme süreçlerinde daha etkili kullanılabilir.

```python
for class_name, data_matrix in class_images.items():
    # Retrieve the top 3 eigenvectors for this class (nx3 matrix)
    top_eigenvectors = pca_results[class_name]['eigenvectors']

    # Correctly project the data matrix onto these top 3 eigenvectors
    # data_matrix is 4096xn, top_eigenvectors is nx3
    # We need to multiply: data_matrix * top_eigenvectors
    # The result will be a 4096x3 matrix representing the class in the new feature space
    projected_features = np.dot(data_matrix, top_eigenvectors)
    # Dictionary to store PCA features for each class
    pca_features = {}

    # Store the projected features in the dictionary
    pca_features[class_name] = projected_features

    # Print the shape of the projected features matrix to verify its dimensions
    print(f"Class {class_name}: Projected features matrix shape (should be 4096x3): {projected_features.shape}")
```

```
Class brain: Projected features matrix shape (should be 4096x3): (4096, 3)
Class BACKGROUND_Google: Projected features matrix shape (should be 4096x3): (4096, 3)
Class barrel: Projected features matrix shape (should be 4096x3): (4096, 3)
Class accordion: Projected features matrix shape (should be 4096x3): (4096, 3)
Class camera: Projected features matrix shape (should be 4096x3): (4096, 3)
Class bonsai: Projected features matrix shape (should be 4096x3): (4096, 3)
Class binocular: Projected features matrix shape (should be 4096x3): (4096, 3)
Class beaver: Projected features matrix shape (should be 4096x3): (4096, 3)
Class anchor: Projected features matrix shape (should be 4096x3): (4096, 3)
Class butterfly: Projected features matrix shape (should be 4096x3): (4096, 3)
Class ant: Projected features matrix shape (should be 4096x3): (4096, 3)
Class airplanes: Projected features matrix shape (should be 4096x3): (4096, 3)
Class brontosaurus: Projected features matrix shape (should be 4096x3): (4096, 3)
Class buddha: Projected features matrix shape (should be 4096x3): (4096, 3)
Class bass: Projected features matrix shape (should be 4096x3): (4096, 3)
```

Her bir sınıf için PCA (Principal Component Analysis - Temel Bileşen Analizi) sonuçlarını kullanarak resim verilerini yeni bir özellik uzayına dönüştürmeyi amaçlamaktadır. İşlem adımları şu şekildedir: İlk olarak, class_images sözlüğünden her sınıf için sınıf adı (class_name) ve o sınıfa ait resim matrisi (data_matrix) alınır. Daha sonra PCA sonuçlarından (pca_results sözlüğünden), o sınıf için belirlenmiş olan üst 3 özvektör (top_eigenvectors) elde edilir. Bu özvektörler, sınıfın resim verilerini yeni bir özellik uzayına dönüştürmek için kullanılacaktır. Ardından, data_matrix (4096xN boyutunda) ve top_eigenvectors (Nx3 boyutunda) matrisleri arasında matris çarpımı gerçekleştirilir (np.dot(data_matrix, top_eigenvectors)). Bu işlem sonucunda elde edilen projected_features matrisi, sınıfın resim verilerinin 4096 boyutlu uzaydan 3 boyutlu yeni özellik uzayına yansıtılmış halini temsil eder. Daha sonra, pca_features adında boş bir sözlük oluşturulur ve projected_features matrisi, pca_features sözlüğüne ilgili sınıf adıyla birlikte kaydedilir. Son olarak, dönüştürülmüş özellik matrisinin boyutunu doğrulamak için ekrana sınıf adı ve matris şekli (4096x3 olmalıdır) yazdırılır. Bu şekilde her sınıfın resim verilerini

PCA sonuçlarına göre yeni bir özellik uzayına dönüştürerek pca_features sözlüğünde saklanır. Bu işlem, verilerin boyutunu azaltarak daha küçük bir özellik uzayında temsil edilmesini sağlar ve veri analizi veya modelleme süreçlerinde daha etkili kullanımı mümkün kılar.

```python
import numpy as np
from PIL import Image
import os

def read_and_process_image(image_path):
    with Image.open(image_path) as img:
        img_resized = img.resize((64, 64))
        img_gray = img_resized.convert('L')
        image_vector = np.array(img_gray).flatten()
    return image_vector

def project_to_pca_space(test_vector, eigenvectors):
    test_vector_reshaped = test_vector.reshape(1, -1)
    test_features = np.dot(test_vector_reshaped, eigenvectors)
    return test_features

def calculate_distances_and_predict(test_features, pca_features):
    min_distance = float('inf')
    predicted_class = None
    for class_name, class_features in pca_features.items():
        distance = np.linalg.norm(test_features - class_features, axis=1).min()
        if distance < min_distance:
            min_distance = distance
            predicted_class = class_name
    return min_distance, predicted_class

# Özvektörler ve PCA özellikleri
eigenvectors = {}
pca_features = {}
```

```python
for class_name, data_matrix in class_images.items():
    if data_matrix.shape[0] != 4096:
        data_matrix = data_matrix.T  # Her bir sütun bir görseli temsil etmeli

    covariance_matrix = np.cov(data_matrix, rowvar=True)  # Şimdi her satır bir değişken olacak
    eigenvalues, eigenvectors_ = np.linalg.eig(covariance_matrix)
    indices = eigenvalues.argsort()[::-1]
    top_eigenvectors = eigenvectors_[:, indices[:3]]
    eigenvectors[class_name] = top_eigenvectors
    pca_features[class_name] = np.dot(data_matrix.T, top_eigenvectors)  # Projekte edilmiş özellikler

# Test görüntüsü işleme ve sınıf tahmini
test_image_path = '/content/HWData/HWData/test/bonsai/image_0119.jpg'
test_vector = read_and_process_image(test_image_path)

predicted_class = None
min_distance = float('inf')

for class_name, class_eigenvectors in eigenvectors.items():
    test_features = project_to_pca_space(test_vector, class_eigenvectors)
    current_distance, current_class = calculate_distances_and_predict(test_features, pca_features)
    if current_distance < min_distance:
        min_distance = current_distance
        predicted_class = current_class

print(f"Tahmin edilen sınıf: {predicted_class}")
```

```
Tahmin edilen sınıf: bonsai
```

İlk olarak, read_and_process_image fonksiyonu, belirtilen görüntü yolundaki resmi 64x64 boyutuna yeniden boyutlandırır ve siyah beyaz olarak dönüştürür. Daha sonra resmi düz bir vektöre dönüştürerek bu vektörü döndürür.

project_to_pca_space fonksiyonu, verilen bir test vektörünü PCA özvektörlerine göre yeni özellik uzayına dönüştürmek için kullanılır. Test vektörü yeniden şekillendirilerek (reshape) tek satırlı bir matrise dönüştürülür. Bu matris, PCA özvektörleriyle çarpılarak yeni özellikler elde edilir.

calculate_distances_and_predict fonksiyonu, verilen test özellikleri (vektörü) ile PCA özelliklerini içeren sözlük arasındaki mesafeleri hesaplar ve test görüntüsünün hangi sınıfa ait olduğunu tahmin eder. Her sınıfın PCA özellikleri, PCA sonuçlarından oluşan pca_features sözlüğünde saklanır.

Ana işlem, eigenvectors ve pca_features sözlüklerinin oluşturulmasıyla başlar. class_images sözlüğündeki her sınıf için:

Eğer resim matrisinin satır sayısı 4096 değilse (yani sütun sayısı 4096 ise), matris transpoze edilir (her bir sütun bir görseli temsil etmelidir).

Kovaryans matrisi hesaplanır ve bu matristen özdeğerler (eigenvalues) ve özvektörler (eigenvectors_) elde edilir. Özdeğerler büyüklüklerine göre sıralanır.

En büyük üç özvektör (top_eigenvectors) seçilerek eigenvectors sözlüğüne eklenir.

Resim matrisi, bu seçilen özvektörlerle çarpılarak yeni özellikler elde edilir ve pca_features sözlüğüne kaydedilir.

Son olarak, bir test görüntüsü belirtilir (test_image_path) ve bu görüntü read_and_process_image fonksiyonu kullanılarak işlenir. Daha sonra her sınıf için:

Test vektörü PCA özvektörlerine göre yeni özellik uzayına dönüştürülür.

calculate_distances_and_predict fonksiyonuyla test özellikleri diğer sınıfların PCA özellikleriyle karşılaştırılarak en yakın sınıf ve mesafe belirlenir.

En sonunda, tahmin edilen sınıf (predicted_class) ekrana yazdırılır. Bu işlem, PCA sonuçlarına dayalı olarak test görüntüsünün hangi sınıfa ait olduğunu tahmin eder ve sonucu kullanıcıya bildirir.