Clasificare Imagini (Cana vs. Pahar)

Proiectul consta intr-un sistem de clasificare a imaginilor intre doua clase: 'Cana' și 'Pahar'. Scopul este antrenarea unui model de invatare automata care sa poata distinge intre aceste doua obiecte comune. Logica principala prin care am incercat sa antrenez algoritmul este de a detecta manerul de la o cana, absenta sa indicand clasificarea drept pahar.

Setul de date a fost colectat manual din surse publice si organizat in 2 subfoldere corespunzatoare celor doua clase in folderul ”dataset”. M-am asigurat in selectare ca nu exista mai mult de o cana / pahar per imagine pentru simplificare procesului. Am folosit un script .bat pentru a le redenumi de la 1 la n, dupa am folosit un Random-Number-Generator pentru a selecta datele de testare, organizate in acelasi fel in folderul ”test”.

Primul pas in preprocesarea este redimensionarea imaginilor astfel incat sa fie consistente una cu cealalta si pentru scaderea resurselor necesare procesarii. Am ales 200x200 prin testare unde am vazut ca ca dimensiuni mai mari nu cresc semnificativ calitatea rezultatelor. Dupa am convertit valorile la grayscale deoarece nu am considerat obligatorie culoare in detectie, algoritmul fiind antrenat sa se uite la characteristicile fizice ale obiectelor, cu alte cuvinte manerul. In final am convertit valorile in matrice 2D numpy si le-am normalizat pentru procesare.

Am utilizat algoritmul K-Nearest Neighbors (KNN) pentru clasificare datorita usurintei de implementare si am ales arbitrar n_neighbors=3.

Biblioteci Python Utilizate

- os pentru manipularea sistemului de fișiere
- cv2 (OpenCV) pentru manipularea imaginilor
- numpy pentru manipularea matricelor
- scikit-learn pentru implementarea algoritmului KNN și evaluarea modelului
- matplotlib pentru vizualizarea imaginilor și rezultatelor

```
import os

import cv2

import numpy as np

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Function to load train images from a folder and assign labels
def load_train_images_from_folder(folder, target_shape=None):
    images = []
    labels = []
    for subfolder in os.listdir(folder):
        subfolder_path = os.path.join(folder, subfolder)
        if os.path.isdir(subfolder_path):
            for filename in os.listdir(subfolder_path):
                if filename.endswith('.jpg'):
                    img = cv2.imread(os.path.join(subfolder_path, filename))
                    if img is not None:
                        # Resize the image to a consistent shape (e.g., 100x100)
                        if target_shape is not None:
                            img = cv2.resize(img, target_shape)
                        images.append(img)
                        labels.append(subfolder)
                    else:
                        print(f"Warning: Unable to load {filename}")
    return images, labels

# Function to load test images from a folder and assign labels
def load_test_images_from_folder(folder, target_shape=None):
    images = []
    labels = []
```

```
    for subfolder in os.listdir(folder):
        subfolder_path = os.path.join(folder, subfolder)
        if os.path.isdir(subfolder_path):
            for filename in os.listdir(subfolder_path):
                if filename.endswith('.jpg'):
                    img = cv2.imread(os.path.join(subfolder_path, filename))
                    if img is not None:
                        # Resize the image to a consistent shape (e.g., 100x100)
                        if target_shape is not None:
                            img = cv2.resize(img, target_shape)
                        images.append(img)
                        labels.append(subfolder)
                        print('Labels \n', labels)
                    else:
                        print(f"Warning: Unable to load {filename}")
    return images, labels

def train_knn_model(images, labels, num_neighbors):
    knn = KNeighborsClassifier(n_neighbors=num_neighbors)
    knn.fit(images.reshape(images.shape[0], -1), labels)
    return knn

def main():
    # Folder paths
    dataset_folder = 'dataset'
    test_folder = 'test'

    # Load images and labels from the 'dataset' folder and resize them to (200, 200)
    images, labels = load_train_images_from_folder(dataset_folder, target_shape=(200, 200))
```

```
# Reshape the images and convert them to grayscale
image_data = [cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).flatten() for image in images]

# Convert the list of 1D arrays to a 2D numpy array
image_data = np.array(image_data)

# Scale the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(image_data)

# Antrenarea modelului KNN
num_neighbors = 3  # Poți ajusta acest număr la nevoie
knn_model = train_knn_model(scaled_data, labels, num_neighbors)

# Evaluarea modelului pe setul de date de antrenare
train_predictions = knn_model.predict(scaled_data.reshape(scaled_data.shape[0], -1))

# Calcularea acuratetei pe setul de date de antrenare
accuracy_train = accuracy_score(labels, train_predictions)
print(f"\nAcuratete pe setul de date de antrenare: {accuracy_train:.2f}")

# Matricea de confuzie pe setul de date de antrenare
conf_matrix = confusion_matrix(labels, train_predictions)
print("\nMatricea de confuzie:")
print(conf_matrix)

# Raportul de clasificare pe setul de date de antrenare
class_report = classification_report(labels, train_predictions, target_names=['Cana', 'Pahar'])
```

```
print("\nRaportul de clasificare:")
print(class_report)

# Testarea modelului pe setul de date de test
test_images, test_labels = load_test_images_from_folder(test_folder, target_shape=(200, 200))
print('# TEST files:', len(test_images))

test_prediction = []
plt.figure(figsize=(12, 6))

# Apply KNN to TEST images
for i, test_image in enumerate(test_images):
    test_image_data = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY).flatten()
    test_image_data = np.array(test_image_data).reshape(1, -1)  # Reshape for a single sample
    scaled_test_data = scaler.transform(test_image_data)
    test_prediction.append(knn_model.predict(scaled_test_data))  # Predict classes
    if test_prediction[i] == 'cana':
        print(f"Test Image {i + 1} - ==Cana==")
    else:
        print(f"Test Image {i + 1} - ==Pahar==")

    # Vizualizare imagine cu predictia
    plt.subplot(2, len(test_images) // 2, i + 1)
    plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
    if test_prediction[i] == 'cana':
        plt.title(f"Cana")
    else:
        plt.title(f"Pahar")
    plt.axis('off')
```

```
    plt.show()

    # Calcularea acurateței pe setul de date de testare
    accuracy_test = accuracy_score(test_labels, test_prediction)
    print(f"\nAcuratete pe setul de date de testare: {accuracy_test:.2f}")

    # Matricea de confuzie pe setul de date de testare
    conf_matrix = confusion_matrix(test_labels, test_prediction)
    print("\nMatricea de confuzie:")
    print(conf_matrix)

    # Raportul de clasificare pe setul de date de testare
    class_report = classification_report(test_labels, test_prediction, target_names=['Cana', 'Pahar'])
    print("\nRaportul de clasificare:")
    print(class_report)

if __name__ == "__main__":
    main()
```

```
C:\Users\numef\AppData\Local\Programs\Python\Python312\python.exe C:\Users\numef\OneDrive\Desktop\Workspace\TIA\Tema_Tia\Tema_Tia_clasificare.py

Acuratete pe setul de date de antrenare: 0.90

Matricea de confuzie:
[[223  16]
 [ 32 208]]

Raportul de clasificare:
              precision    recall  f1-score   support

        Cana       0.87      0.93      0.90       239
       Pahar       0.93      0.87      0.90       240

    accuracy                           0.90       479
   macro avg       0.90      0.90      0.90       479
weighted avg       0.90      0.90      0.90       479
```

Prima data am testat algoritmul pe setul de date de antrenare pentru a vedea daca exista deviatii in adaptare. Din ce am citit legat de KNN algoritmul poate avea tendința de a memora setul de date de antrenare, dar rezultatele imperfecte de 90% arata ca algoritmul este cel putin capabil de a generaliza imaginile.

Putem vedea de asemenea o inclinare catre TN (pahare detectate drept cani) dubla fata de FP. (cani detectate drept pahare).

```
Acuratete pe setul de date de testare: 0.90

Matricea de confuzie:
[[10  0]
 [ 2  8]]

Raportul de clasificare:
              precision    recall  f1-score   support

        Cana       0.83      1.00      0.91        10
      Pahar        1.00      0.80      0.89        10

    accuracy                           0.90        20
   macro avg       0.92      0.90      0.90        20
weighted avg       0.92      0.90      0.90        20
```

Pe setul de date de testare au aparut doua clasificari gresite de tip TN, conform cu ceea ce am vazut la testarea precedenta. Scorurile f1 ale celor doua clase do obiecte sunt destul de apropiate unul de celalalt astfel putem spune ca sunt in marginea de erroare identice, algoritmul neavand preferinte catre precizie sau recall. In rest, acuratetea de 90% este optima, dar nu ideala.

Daca ne uitam la imaginile alese, vedem ca paharele cu label-ul ’Cana’ au in imagine obiecte in plus care pot fi interpretate de catre algoritm drept maner, deci o categorisire de ’Cana’. De asemenea am observat ca exista o diversitate mare de tipuri de pahare in setul de date ce ar fi putut incurce algoritmul in definirea proprie a unui ’Pahar’.

```
# TEST files: 20
Test Image 1 - ==Cana==
Test Image 2 - ==Cana==
Test Image 3 - ==Cana==
Test Image 4 - ==Cana==
Test Image 5 - ==Cana==
Test Image 6 - ==Cana==
Test Image 7 - ==Cana==
Test Image 8 - ==Cana==
Test Image 9 - ==Cana==
Test Image 10 - ==Cana==
Test Image 11 - ==Cana==
Test Image 12 - ==Pahar==
Test Image 13 - ==Pahar==
Test Image 14 - ==Cana==
Test Image 15 - ==Pahar==
Test Image 16 - ==Pahar==
Test Image 17 - ==Pahar==
Test Image 18 - ==Pahar==
Test Image 19 - ==Pahar==
Test Image 20 - ==Pahar==
```