Ciuculan Ștefan – 333AA

TEMA 2 – Tehnici de învățare automată

Aplicație Python de clasificare cu ajutorul algoritmilor K-Nearest Neighbors și Naive Bayes

1. Prezentarea temei

Tema proiectului se axează pe clasificarea imaginilor în două categorii distincte: urbane și rurale. Scopul este de a evalua performanța a doi algoritmi de Machine Learning, respectiv Gaussian Naive Bayes și K-Nearest Neighbors (KNN), în ceea ce privește capacitatea lor de a face distincția între mediile urbane și cele rurale.

2. Setul de date

Setul de date utilizat conține imagini etichetate pentru fiecare mediu (urban sau rural). Imaginile au fost redimensionate la dimensiuni standard (200x200 pixeli) pentru a asigura consistența în analiză. Fiecare imagine este prelucrată pentru a extrage caracteristicile HOG (Histogram of Oriented Gradients), utilizate în antrenarea și testarea modelelor.

Imaginile de antrenament, în număr de 100 (50 + 50), au fost organizate într-un director "train" și separate pe 2 subdirectoare "urban" și "rural". Datele de test, în număr de 20 (11 + 9), au fost puse într-un director numit "test_accuracy" și separate la fel ca în directorul de antrenament. Am avut nevoie de această a doua separare pentru ca, la evaluarea rezultatelor, să putem fi conștienți de veridicitatea lor și de acuratețea algoritmului.

3. Biblioteci Python Utilizate

Proiectul a fost implementat folosind următoarele biblioteci Python:

- **OpenCV** pentru manipularea imaginilor
- **NumPy** pentru manipularea datelor
- **scikit-learn** pentru implementarea Naive Bayes, KNN și funcțiile de evaluare

- **Matplotlib** pentru afișarea imaginilor și rezultatelor

4. Procesul de antrenare

Gaussian Naive Bayes

- Datele de antrenament sunt prelucrate folosind extragerea de caracteristici HOG și standardizarea scalată.
- Un clasificator Gaussian Naive Bayes este antrenat folosind datele pregătite.

K-Nearest Neighbors (KNN)

- Similar cu Naive Bayes, datele sunt prelucrate și standardizate.
- Un clasificator KNN cu 3 vecini este antrenat. Am ales 3 vecini pentru că, în urma mai multor testări, am observat că se obține cea mai mare acuratețe pe setul de date de test.

5. Implementarea

```
import os
import cv2
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from skimage.feature import hog
import matplotlib.pyplot as plt
```

Această secțiune importă modulele necesare pentru manipularea imaginilor, implementarea Naive Bayes (KNN), prelucrarea datelor și extragerea caracteristicilor HOG (Histogram of Oriented Gradients).

```python
# Functie pentru incarcarea imaginilor dintr-un folder
def load_images_from_folder(folder, target_shape=None):
    images = []  # Lista pentru a stoca imaginile
    labels = []  # Lista pentru a stoca etichetele corespunzatoare imaginilor
    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(('.jpeg', '.jpg')):
                    img = cv2.imread(os.path.join(subfolder_path, filename))
                    if img is not None:
                        if target_shape is not None:
                            img = cv2.resize(img, target_shape)
                        images.append(img)
                        labels.append(subfolder)  # Se adauga eticheta corespunzatoare imaginii
                    else:
                        print(f"Avertisment: Incapabil să încarce {filename}")
    return images, labels
```

Funcția **load_images_from_folder** încarcă imaginile dintr-un director specific și extrage etichetele (labels) asociate. Se redimensionează imaginile la dimensiunea specificată dacă este specificată.

```python
# Functie pentru extragerea caracteristicilor HOG din imagini
def extract_hog_features(images):
    features = []
    for img in images:
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        features.append(hog(img_gray, block_norm='L2-Hys'))
    return np.array(features)
```

Funcția **extract_hog_features** primește o listă de imagini și returnează caracteristicile HOG (Histogram of Oriented Gradients) ale acestora. Se convertește fiecare imagine la tonuri de gri înainte de extragerea caracteristicilor HOG.

```
# Definirea folderului cu datele de antrenare
data_folder = './train'

# Incarcarea imaginilor de antrenare si a etichetelor corespunzatoare
images, labels = load_images_from_folder(data_folder, target_shape=(200, 200))
```

Se încarcă imaginile de antrenare din directorul specificat (**./train**) și se apelează funcția **load_images_from_folder** pentru a obține imaginile și etichetele corespunzătoare.

```
# Extrage caracteristicile HOG din imaginile de antrenare
X_train_features = extract_hog_features(images)

# Scalarea datelor
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_features)
```

Se extrag caracteristicile HOG pentru setul de antrenare, folosind funcția **extract_hog_features** și se scalează datele pentru a le aduce la aceeași scară folosind **StandardScaler** din **sklearn**.

```
# Initializarea si antrenarea clasificatorului Gaussian Naive Bayes
clf = GaussianNB()
clf.fit(X_train_scaled, labels)
```

```
# Configurarea si antrenarea clasificatorului K-Nearest Neighbors (KNN)
knn_neighbors = 3
clf = KNeighborsClassifier(n_neighbors=knn_neighbors)
clf.fit(X_train_scaled, labels)
```

Se antrenează un clasificator Naive Bayes, respectiv unul KNN.

```
# Definirea folderului cu datele de test
test_folder = './test_accuracy'

# Incarcarea imaginilor de test si a etichetelor corespunzatoare
test_images, test_labels = load_images_from_folder(test_folder, target_shape=(200, 200))
```

Se încarcă imaginile de testare din directorul specificat (**./test_accuracy**) și se apelează funcția **load_images_from_folder** pentru a obține imaginile și etichetele corespunzătoare.

```
# Extrage caracteristicile HOG din imaginile de test
X_test_images_features = extract_hog_features(test_images)

# Scalarea datelor de test
X_test_images_scaled = scaler.transform(X_test_images_features)

# Realizeaza predictii folosind clasificatorul antrenat
y_test_images_pred = clf.predict(X_test_images_scaled)
```

Se extrag caracteristicile HOG pentru imaginile de testare și se scalează, utilizând același scalator utilizat anterior pe setul de antrenare, urmând ca mai apoi să se prezică etichetele pentru imaginile de testare folosind clasificatorul antrenat.

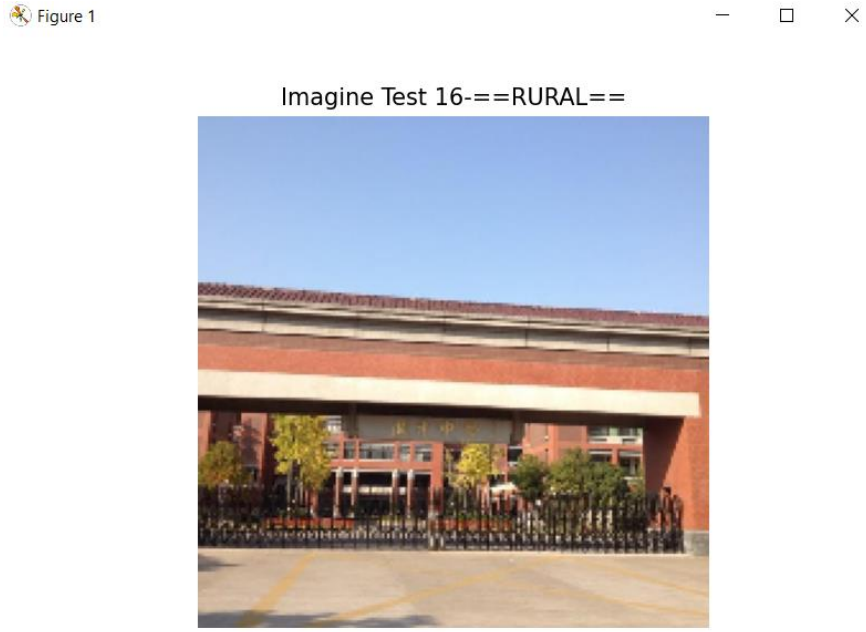
```
# Afisarea rezultatelor pentru fiecare imagine din setul de test
for i, (test_image, predicted_label) in enumerate(zip(test_images, y_test_images_pred)):
    print(f"Imagine Test {i + 1}: Predicted - {predicted_label}, Real - {test_labels[i]}")

    # Afisarea imaginii de test
    plt.figure()
    plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
    plt.title(f"Imagine Test {i + 1}-=={predicted_label.upper()}==")
    plt.axis('off')
    plt.show()
```

Se afișează imaginile de testare, împreună cu etichetele prezise, pentru a evalua vizual performanța modelului.

6. Procesul de testare și concluzii

- **Algoritmul Naive Bayes** a prezentat o acuratețe de 95% pe imaginile de test, greșind astfel în cazul unei singure imagini, pe care nici metoda de la tema anterioara, cu algoritmul SVM nu a reușit să o identifice corect.




```
PS C:\Users\stefa\OneDrive\Documente\faculta\an3sem1\TIA\tema2> python3 main_NaiveBayes.py
Acuratețe pe Imaginile de Test: 95.00%
Imagine Test 1: Predicted - rural, Real - rural
Imagine Test 2: Predicted - rural, Real - rural
Imagine Test 3: Predicted - rural, Real - rural
Imagine Test 4: Predicted - rural, Real - rural
Imagine Test 5: Predicted - rural, Real - rural
Imagine Test 6: Predicted - rural, Real - rural
Imagine Test 7: Predicted - rural, Real - rural
Imagine Test 8: Predicted - rural, Real - rural
Imagine Test 9: Predicted - rural, Real - rural
Imagine Test 10: Predicted - rural, Real - rural
Imagine Test 11: Predicted - rural, Real - rural
Imagine Test 12: Predicted - urban, Real - urban
Imagine Test 13: Predicted - urban, Real - urban
Imagine Test 14: Predicted - urban, Real - urban
Imagine Test 15: Predicted - urban, Real - urban
Imagine Test 16: Predicted - rural, Real - urban
Imagine Test 17: Predicted - urban, Real - urban
Imagine Test 18: Predicted - urban, Real - urban
Imagine Test 19: Predicted - urban, Real - urban
Imagine Test 20: Predicted - urban, Real - urban
```

- **Algoritmul K Nearest Neighbors** a identificat greșit 2 imagini, având o acuratețe afișată de 90%. Asta în cazul utilizării algoritmului cu k = 3 “vecini”.


```
PS C:\Users\stefa\OneDrive\Documente\faculta\an3sem1\TIA\tema2> python3 main_KNN.py
Acuratețe pe Imaginile de Test: 90.00%
Imagine Test 1: Predicted - rural, Real - rural
Imagine Test 2: Predicted - rural, Real - rural
Imagine Test 3: Predicted - rural, Real - rural
Imagine Test 4: Predicted - rural, Real - rural
Imagine Test 5: Predicted - rural, Real - rural
Imagine Test 6: Predicted - rural, Real - rural
Imagine Test 7: Predicted - rural, Real - rural
Imagine Test 8: Predicted - rural, Real - rural
Imagine Test 9: Predicted - rural, Real - rural
Imagine Test 10: Predicted - rural, Real - rural
Imagine Test 11: Predicted - rural, Real - rural
Imagine Test 12: Predicted - urban, Real - urban
Imagine Test 13: Predicted - urban, Real - urban
Imagine Test 14: Predicted - urban, Real - urban
Imagine Test 15: Predicted - urban, Real - urban
Imagine Test 16: Predicted - rural, Real - urban
Imagine Test 17: Predicted - urban, Real - urban
Imagine Test 18: Predicted - urban, Real - urban
Imagine Test 19: Predicted - rural, Real - urban
Imagine Test 20: Predicted - urban, Real - urban
```