Name : Ashuwin P

Reg.No : 3122 22 5002 013

Course : UIT2622 ~ Advanced Artificial Intelligence Techniques

Topic : Object Recognition ~ Implementation of KNN for Object Recognition

Last Update : 06 November 2024

Object Recognition using K-Nearest Neighbors (KNN)

Aim

To develop an object recognition system using K-Nearest Neighbors (KNN) and Histogram of Oriented Gradients (HOG) features for classifying images of humans, cats, dogs, and horses.

Introduction

Object recognition is a fundamental task in computer vision, enabling the identification and classification of objects within images. This project utilizes the K-Nearest Neighbors (KNN) algorithm in conjunction with Histogram of Oriented Gradients (HOG) features to perform object recognition. The goal is to classify images into four categories: humans, cats, dogs, and horses. This approach is chosen for its simplicity and effectiveness, making it suitable for beginners in the field of machine learning and computer vision.

Overview

1. **Data Collection**: Images of humans, cats, dogs, and horses are collected and organized into respective directories.
2. **Preprocessing**: Each image is resized and converted to grayscale. HOG features are extracted from these processed images.
3. **Model Training**: The K-Nearest Neighbors (KNN) algorithm is trained on the extracted features.
4. **Hyperparameter Tuning**: Grid Search is used to find the optimal hyperparameters for the KNN model.
5. **Evaluation**: The trained model is evaluated using accuracy metrics.
6. **Prediction**: The model is used to predict and display the class of new images.

Methodology

1. **Data Preprocessing**:
   - **Resizing**: Images are resized to 64x64 pixels.
   - **Grayscale Conversion**: Images are converted to grayscale to simplify computations.
   - **HOG Feature Extraction**: Histogram of Oriented Gradients (HOG) features are extracted to capture essential details of the images.
2. **Feature Scaling**:
   - **StandardScaler**: Applied to normalize the feature vectors for better model performance.
3. **Model Training**:
   - **Dataset Splitting**: The dataset is split into training and testing sets.
   - **Training**: The KNN algorithm is trained with different hyperparameters using Grid Search to find the best configuration.
4. **Hyperparameter Tuning**:
   - **Grid Search**: `n_neighbors` and `weights` parameters are tuned using Grid Search with cross-validation.
5. **Evaluation**:
   - **Accuracy Scores**: Model performance is evaluated using accuracy scores.
   - **Predictions**: Predictions are made on test images, and the accuracy is calculated.
6. **Prediction**:
   - **Class Prediction**: The trained model predicts the class of new images, displaying the image and associated probabilities.

Formulae

K-Nearest Neighbors (KNN) Classification

The classification rule is given by:

$$h(x) = \arg\max_y \sum_{i=1}^{k} I(y_i = y)$$

where I is an indicator function that returns 1 if $y_i = y$ and 0 otherwise, and k is the number of nearest neighbors.

Histogram of Oriented Gradients (HOG)

The HOG feature extraction is defined as:

$$\text{HOG}(I) = \text{histogram}(\theta(x, y), M(x, y))$$

where $\theta(x, y)$ is the gradient orientation at pixel (x, y), and $M(x, y)$ is the gradient magnitude at pixel (x, y).

In [1]:
```python
import os
import cv2
import numpy as np
from skimage.feature import hog
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [2]:
```python
def extract_hog_features(image):
    resized_image = cv2.resize(image, (64, 64))  # Resize image to 64x64
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    features, _ = hog(gray_image, pixels_per_cell=(8, 8), cells_per_block=(2, 2), block_norm='L2-Hys', visualize=True)
    return features
```

In [3]:
```python
def load_data(data_directory):
    features = []
    labels = []
    for label_name in os.listdir(data_directory):
        label_path = os.path.join(data_directory, label_name)
        if os.path.isdir(label_path):
            for image_name in os.listdir(label_path):
                image_path = os.path.join(label_path, image_name)
                image = cv2.imread(image_path)
                if image is not None:
                    feature_vector = extract_hog_features(image)
                    features.append(feature_vector)
                    labels.append(label_name)
    return np.array(features), np.array(labels)
```

In [4]:
```python
# Example data directory
data_directory = "data"
features, labels = load_data(data_directory)
print(f"Loaded {len(features)} features and {len(labels)} labels")
```

```
Loaded 808 features and 808 labels
```

In [5]:
```python
# Feature scaling
scaler = StandardScaler()
features = scaler.fit_transform(features)
```

In [6]:
```python
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
X_train, X_test, y_train, y_test = train_test_split(features, labels_encoded, test_size=0.2, random_state=42)
```

In [7]:
```python
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

# Hyperparameter tuning with GridSearchCV
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
best_knn = grid_search.best_estimator_

print("Best parameters found:", grid_search.best_params_)
print("Best cross-validation accuracy:", grid_search.best_score_)
```

```
Best parameters found: {'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
Best cross-validation accuracy: 0.5696004770423375
```

In [8]:
```python
if 'X_train' in locals():
    best_knn.fit(X_train, y_train)
```

In [9]:
```python
if 'X_test' in locals():
    y_pred = best_knn.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6049382716049383
```

In [10]:
```python
def predict_image(knn, image_path):
    image = cv2.imread(image_path)
    features = extract_hog_features(image)
    features = scaler.transform([features])
    probabilities = knn.predict_proba(features)[0]
    class_labels = label_encoder.inverse_transform(np.arange(len(probabilities)))

    # Find the predicted class and format the probabilities
    class_idx = np.argmax(probabilities)
    class_label = class_labels[class_idx]

    # Display probabilities
    prob_dict = {class_labels[i]: probabilities[i] for i in range(len(class_labels))}

    return class_label, prob_dict, image
```

In [11]:
```python
def predict_images_in_directory(knn, directory_path):
    for image_name in os.listdir(directory_path):
        image_path = os.path.join(directory_path, image_name)
        if image_path.endswith(('.jpg', '.png', '.jpeg')):  # Include only valid image files
            prediction, prob, image = predict_image(knn, image_path)
            plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
            plt.title(f"Predicted: {prediction}")
            plt.axis('off')
            plt.show()
            print("Probabilities:")
            for class_name, probability in prob.items():
                print(f"{class_name} -> {probability:.2f}")
            print("\n")
```

In [12]:
```python
# Predicting images in a new directory
if 'best_knn' in locals():
    new_directory = r"D:\SEM5\AAIT\Practical\Object_Recognition\sample_input"
    predict_images_in_directory(best_knn, new_directory)
```

Predicted: human

```
Probabilities:
cat -> 0.00
dog -> 0.40
horse -> 0.00
human -> 0.60
```

Predicted: cat

```
Probabilities:
cat -> 0.60
dog -> 0.40
horse -> 0.00
human -> 0.00
```

Predicted: horse

```
Probabilities:
cat -> 0.00
dog -> 0.00
horse -> 1.00
human -> 0.00
```

Predicted: horse

```
Probabilities:
cat -> 0.00
dog -> 0.19
horse -> 0.81
human -> 0.00
```

Predicted: cat

```
Probabilities:
cat -> 0.61
dog -> 0.39
horse -> 0.00
human -> 0.00
```

Predicted: human


```
Probabilities:
cat -> 0.00
dog -> 0.00
horse -> 0.40
human -> 0.60
```

Predicted: human

```
Probabilities:
cat -> 0.00
dog -> 0.00
horse -> 0.00
human -> 1.00
```

Conclusion

In this project, we developed an object recognition system using K-Nearest Neighbors (KNN) and Histogram of Oriented Gradients (HOG) features to classify images into four categories: humans, cats, dogs, and horses. The process involved data preprocessing, feature extraction, model training, hyperparameter tuning, and evaluation. The resulting system demonstrated the practical use of KNN and HOG for object recognition, providing a solid foundation for further improvements and exploration in the field of computer vision.
