Digit Classification based on HOG features of MNIST Dataset

Om Singh

Objective

The goal of the attached Python notebook is to apply the Histogram of Oriented Gradients (HOG) feature extraction algorithm on the MNIST dataset and build a robust classification system to recognize handwritten digits. We aim to include the following:

- Experimenting with different HOG parameters (cell size, block size, orientation bins) to optimize feature representation.
- Comparing multiple common machine learning classifiers using grid search for hyperparameter tuning.
- Achieving high classification accuracy through systematic parameter optimization.

HOG Feature Extraction Methodology

- Extracted ZIP file downloaded from the provided GitHub link programmatically using Python's `zipfile` library.
- Loaded 28x28 grayscale images with PIL (Python Imaging Library).
- Implemented HOG using OpenCV's `HOGDescriptor` with the following parameters:
  - `winSize`: (28,28) - Full image dimensions.
  - `cell_size`: (7,7) or (8,8) pixels were tried.
  - `block_size`: (14,14), (16,16) and (21,21) pixels were tried.
  - `nbins`: 9 orientation bins were tried.

  We performed a GridSearch for the best set of parameters. We ensured that at each point of time, the relations between the various parameters are intact as per the dimensions of the input image.
- Generated HOG feature vectors for all images.

Machine Learning Model Choice

- Split data into 80% training and 20% test sets.
- Evaluated 3 popular classifiers, each optimized with grid search:
  - Support Vector Machines (SVM)
  - Logistic Regression
  - Linear Discriminant Analysis (LDA)
- Tuned hyperparameters of each of those models using 3-fold cross-validation. For an exhaustive list of the hyperparameters tested, refer to the attached Python notebook.
- Measured accuracy on test set for all combinations.
- Recorded best parameters for both HOG and classifiers.

Conclusion

On performing Grid Search through the various HOG Features and hyperparameters, the best ML model from the ones tested was found to be the `Support Vector Classifier`. The details of the hyperparameters which yielded this are the following:

- Accuracy of the model: **99.05%**.
- `cell_size`: (7, 7).
- `block_size`: (14,14).
- `nbins`: 9.
- For SVC, `C`: 10, `gamma`: scale.

References to Datasets and Libraries

- MNIST Dataset: `https://github.com/teavanist/MNIST-JPG`
- OpenCV HOG Documentation: `https://docs.opencv.org/4.x/d5/d33/structcv_1_1HOGDescriptor.html`
- Scikit-learn GridSearchCV: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearch html`

```python
# Importing required libraries.
import zipfile
import os
import cv2
import numpy as np
from PIL import Image
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB

# Loading the MNIST dataset from the local folder.
def extract_zip(zip_path, extract_to='Data'):
    if not os.path.exists(extract_to):
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            zip_ref.extractall(extract_to)
    return extract_to

def load_data(root_dir):
    images, labels = [], []
    for root, dirs, files in os.walk(root_dir):
        if os.path.basename(root).isdigit():
            label = int(os.path.basename(root))
            for img_file in os.listdir(root):
                img_path = os.path.join(root, img_file)
                if os.path.isfile(img_path):
                    img = Image.open(img_path).convert('L')
                    image = np.array(img)
                    images.append(image)
                    labels.append(label)
    return np.array(images), np.array(labels)

# Central code employing HOG Feature extraction and using classifier
models on it.
if __name__ == "__main__":
    zip_path = "./Data.zip"
    extracted_dir = extract_zip(zip_path)
    training_dir = os.path.join(extracted_dir, "MNIST Dataset JPG
format", "MNIST - JPG - training")
    images, labels = load_data(training_dir)

    # Reduced HOG Parameter Grid
    hog_params_grid = [
        {'cell_size': 7, 'block_size': 14, 'block_stride': 7, 'nbins':
9},
```

```
        {'cell_size': 8, 'block_size': 16, 'block_stride': 4, 'nbins':
9},
        {'cell_size': 7, 'block_size': 21, 'block_stride': 7, 'nbins':
9}
    ]

    # Faster Classifiers with Simplified Grids
    classifiers = [
        ('SVM', SVC(), {'C': [1, 10], 'gamma': ['scale']}),  # Reduced
options
        ('Logistic Regression', LogisticRegression(max_iter=1000),
{'C': [0.1, 1]}),
        ('LDA', LinearDiscriminantAnalysis(), {'solver': ['svd']}),
    ]

    best_overall = {'accuracy': 0}

    for hog_params in hog_params_grid:
        print(f"\n=== Testing HOG Params: {hog_params} ===")

        # HOG initialization (same as before)
        cell_size = (hog_params['cell_size'], hog_params['cell_size'])
        block_size = (hog_params['block_size'],
hog_params['block_size'])
        block_stride = (hog_params['block_stride'],
hog_params['block_stride'])
        nbins = hog_params['nbins']

        hog = cv2.HOGDescriptor(
            (28, 28), block_size, block_stride, cell_size, nbins
        )

        # Compute HOG features (optimized)
        X = np.array([hog.compute(img).flatten() for img in images])
        y = labels

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # Fast classifier tuning
        for clf_name, clf, param_grid in classifiers:
            grid = GridSearchCV(clf, param_grid, cv=3, n_jobs=-1)
            grid.fit(X_train, y_train)
            best_acc = accuracy_score(y_test,
grid.best_estimator_.predict(X_test))

            print(f"{clf_name}: {best_acc:.4f} (Params:
{grid.best_params_})")
```

```python
            if best_acc > best_overall['accuracy']:
                best_overall = {
                    'accuracy': best_acc,
                    'hog_params': hog_params,
                    'clf_name': clf_name,
                    'clf_params': grid.best_params_
                }

    print("\n=== Final Best ===")
    print(f"Accuracy: {best_overall['accuracy']:.4f}")
    print(f"HOG Config: {best_overall['hog_params']}")
    print(f"Best Model: {best_overall['clf_name']}")
    print(f"Model Params: {best_overall['clf_params']}")
```

```
=== Testing HOG Params: {'cell_size': 7, 'block_size': 14,
'block_stride': 7, 'nbins': 9} ===
SVM: 0.9905 (Params: {'C': 10, 'gamma': 'scale'})
Logistic Regression: 0.9822 (Params: {'C': 1})
LDA: 0.9735 (Params: {'solver': 'svd'})

=== Testing HOG Params: {'cell_size': 8, 'block_size': 16,
'block_stride': 4, 'nbins': 9} ===
SVM: 0.9902 (Params: {'C': 10, 'gamma': 'scale'})
Logistic Regression: 0.9815 (Params: {'C': 1})
LDA: 0.9744 (Params: {'solver': 'svd'})

=== Testing HOG Params: {'cell_size': 7, 'block_size': 21,
'block_stride': 7, 'nbins': 9} ===
SVM: 0.9904 (Params: {'C': 10, 'gamma': 'scale'})
Logistic Regression: 0.9792 (Params: {'C': 1})
LDA: 0.9711 (Params: {'solver': 'svd'})

=== Final Best ===
Accuracy: 0.9905
HOG Config: {'cell_size': 7, 'block_size': 14, 'block_stride': 7,
'nbins': 9}
Best Model: SVM
Model Params: {'C': 10, 'gamma': 'scale'}
```