Handwritten Character Recognition using XGBoost and HOG Features: Algorithmic Optimization and Performance Analysis

Can Keçilioğlu
Department of Mechatronics Engineering

November 18, 2025

Abstract

This study addresses the classification of handwritten characters (A-Z, a-z, 0-9), a fundamental problem in Optical Character Recognition (OCR), using classical machine learning methods and modern feature extraction techniques. The Histogram of Oriented Gradients (HOG) algorithm was utilized for image processing, while the eXtreme Gradient Boosting (XGBoost) algorithm was employed for classification. To isolate the factors affecting model performance, four different controlled experiments (Ablation Study) were designed. The processes of dataset filtering, hyperparameter optimization (Grid Search-like manual tuning), and hardware acceleration (AMD Ryzen 5 7600 CPU optimization) are detailed. The findings indicate that noise created by morphologically similar characters (such as '0' and 'O') is the most significant factor affecting model success, and the optimized XGBoost model achieved an accuracy rate of 64.31%. This study presents an efficient alternative for embedded systems that do not require deep learning.

1 Introduction

Handwritten character recognition has a wide range of applications, from automation systems to mail sorting, and banking transactions to the digitization of historical documents. Today, these problems are typically solved using Convolutional Neural Networks (CNNs), which require high computational power. However, in Mechatronics Engineering applications (robotic arms, embedded controllers, etc.), processing power and energy consumption are critical constraints. Therefore, exploring the limits of classical methods based on "Feature Engineering" is of significant importance.

In this study, a scalable and high-performance classification engine was developed based on the XGBoost library [1]. The originality of the study lies in the use of HOG features instead of raw pixel data and the step-by-step analysis of the model's "learning" process.

2 Material and Method

2.1 Dataset and Preprocessing

The *english.csv* dataset was used in this study. The dataset contains a total of 3410 handwritten images belonging to 62 different classes. Images were converted from RGB format to Grayscale and normalized to a size of 64×64 pixels.

2.2 Feature Extraction: Histogram of Oriented Gradients (HOG)

Raw pixel data (a 4096-dimensional vector) is insufficient to directly represent the structural features of characters (curves, corners, vertical lines). Therefore, the HOG algorithm was preferred.

The HOG algorithm follows these steps:

1. The image is divided into small cells (in this study, 8×8 pixels).
2. Gradients (orientations) of pixel intensities are calculated for each cell.
3. These gradients are accumulated in a histogram.

The following code block demonstrates how HOG features are extracted using Python and the `scikit-image` library:

```python
from skimage.feature import hog

# HOG Parameters
# orientations: Number of orientation bins
# pixels_per_cell: Cell size (Detail level)
features = hog(
    image,
    orientations=8,
    pixels_per_cell=(8, 8),
    cells_per_block=(2, 2),
    visualize=False,
    feature_vector=True
)
# Result: 1568-dimensional feature vector
```

Listing 1: HOG Feature Extraction Code

2.3 Classification Model: XGBoost

XGBoost (eXtreme Gradient Boosting) is a "Gradient Boosting" library that creates a strong model by sequentially training weak learners (decision trees) [1].

The mathematical foundation of the model relies on minimizing an objective function:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{1}$$

Here, l represents the prediction error (Loss), and Ω represents the Regularization term that penalizes model complexity.

2.4 Hyperparameter Optimization and Model Configuration

The performance of the XGBoost algorithm is strictly dependent on the optimization of hyperparameters to maintain the bias-variance tradeoff and prevent overfitting. The configuration used in this study is presented in Table 1.

Table 1: XGBoost Hyperparameters Used in Model Training

Parameter	Value	Description and Rationale
`booster`	`gbtree`	Tree-based model structure was used.
`n_estimators`	300	Total number of decision trees. Increased to boost learning capacity but limited to optimize processing time.
`max_depth`	8	Maximum depth of each tree. Increased above the standard value (6) to capture fine details between characters.
`learning_rate`	0.05	(*eta*). Step size shrinkage used in update to prevent overfitting. Kept low for stable convergence.
`subsample`	0.8	Stochastic Gradient Boosting. Uses 80% of random data samples for each tree to reduce variance.
`colsample_bytree`	0.8	Subsample ratio of columns (HOG features) when constructing each tree. Prevents focusing on noise in high-dimensional data.
`objective`	`multi:softmax`	Objective function for multi-class classification.
`tree_method`	`hist`	Histogram-based algorithm preferred for faster training on CPU.

3 Experimental Study and Findings

Four different scenarios (Ablation Study) were designed to isolate the factors affecting model performance. All training sessions were performed on an AMD Ryzen 5 7600 processor using parallel programming (`n_jobs=-1`).

3.1 Experiment 1: Baseline Model

The entire dataset (62 classes) was used. **Result:** 54.40% Accuracy. This low rate indicated that the model struggled to distinguish between morphologically similar characters (e.g., 'l' and 'I').

3.2 Experiment 2: Filtered Data and Optimization (Final Model)

Confusion Matrix analysis revealed the characters where the model made the most errors. Indistinguishable characters such as '0'-'O', '1'-'I'-'l', 'S'-'s' were removed from the dataset. Additionally, the model capacity (`n_estimators`) was increased to 300.

```python
# Model Definition
model = xgb.XGBClassifier(
    n_estimators=300,
    max_depth=8,
    learning_rate=0.05,
    subsample=0.8,
    objective='multi:softmax',
    eval_metric='merror', # Error rate tracking
    n_jobs=-1
)

# Training Initialization and Validation
model.fit(
    X_train, y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=10 # Reporting every 10 iterations
)
```

Listing 2: Optimized XGBoost Training Code

Result: 64.31% Accuracy. This is the highest score achieved in the study, proving that data cleanliness is more important than algorithmic complexity.

3.3 Experiment 3: Effect of Validation Set

The dataset was split into Train (75%), Validation (10%), and Test (15%) to measure model stability. **Result:** 55.08% Accuracy. Obtaining a result similar to the baseline model proved that the model was not overfitting.

3.4 Experiment 4: Noise Robustness Test

"Gaussian Noise" was added to the images to simulate real-world conditions. **Result:** 48.39% Accuracy. It was observed that the HOG method is sensitive to noise due to its gradient-based nature.

4 Results and Discussion

The obtained results are summarized in Table 2.

Table 2: Comparative Results of Four Different Experiments

Exp No	Exp Type	Accuracy	Difference (vs Baseline)
1	Baseline	54.40%	-
3	Validation	55.08%	+0.68%
4	Noise Test	48.39%	-6.01%
2 (Final)	**Filtered + Optimized**	**64.31%**	**+9.91%**

This study demonstrated that the XGBoost algorithm, when combined with correct feature engineering (HOG), can yield acceptable results even in complex problems like handwritten character recognition. However, the 64.31% accuracy rate reveals that morphological similarities between characters cannot be fully resolved using the HOG method alone.

Optimizations performed on the Ryzen 5 7600 processor reduced the training time by approximately 10 times compared to the Google Colab environment (52 seconds), proving the suitability of the method for real-time applications.

Future work aims to use Convolutional Neural Networks (CNNs) instead of HOG for feature extraction or to experiment with hybrid (HOG + CNN) architectures.

References

[1] XGBoost Developers. (2024). *XGBoost Documentation*. Retrieved from: `https://xgboost.readthedocs.io/en/stable/`

[2] Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.