Handwritten Digit Detection

1st Shamit Savant, 2nd Sona Maria Jose, 3rd Ankan Ghosh
University of Florida: Electrical and Computer Engineering

***Abstract*—This report presents the development and evaluation of a hand written digit detection system using the Faster R-CNN model with a ResNet-50 backbone. The project involves preprocessing grayscale images, applying adaptive thresholding, and training the model on labeled datasets. The system achieves robust performance in detecting digits (0-9) and also predicts -1 class in cases of low confidence in test datasets, with metrics such as Intersection over Union (IoU) and accuracy used for evaluation. Experimental results demonstrate the efficacy of the proposed approach in digit recognition tasks.**

I. Introduction

Handwritten digit detection is a key challenge in machine learning, which is used in postal address recognition, check processing, and more. By identifying and classifying handwritten digits, models must handle varying handwriting styles and orientations.

The MNIST dataset, with its 60,000 training and 10,000 test images, is a benchmark for this task. Recent advancements, especially with convolutional neural networks (CNNs), have significantly improved performance in this area.

In our project, we use Faster R-CNN with a ResNet-50 backbone. Our custom dataset of 4,190 images includes varied styles and backgrounds, and we apply adaptive thresholding during preprocessing.

II. Implementation

A. Data Preparation

The data preparation is handled by the `DigitDataset` class, a custom implementation of PyTorch's `Dataset` class. This class is responsible for loading images and their corresponding annotations.The dataset class sorts bounding boxes based on x-coordinates to maintain left-to-right order, which is crucial for digit sequence recognition.We added handling for empty bounding boxes to prevent training issues.

1) Image Loading: Images are loaded and converted to grayscale

2) Coordinate Conversion: The annotations are converted from YOLO format to pixel coordinates:

$$\begin{aligned} x_1 &= (x_{\text{center}} - \frac{\text{width}}{2}) \times W \\ y_1 &= (y_{\text{center}} - \frac{\text{height}}{2}) \times H \\ x_2 &= (x_{\text{center}} + \frac{\text{width}}{2}) \times W \\ y_2 &= (y_{\text{center}} + \frac{\text{height}}{2}) \times H \end{aligned} \tag{1}$$

where W and H are the original image width and height, respectively.

B. Data Augmentation

The data augmentation focuses on image preprocessing and thresholding.

1) Resizing: Images are resized to a fixed size of 600x600 pixels:

2) Thresholding: Thresholding is utilized as a preprocessing step in this project to enhance the quality of input images before feeding them into the object detection model. Both global and adaptive thresholding techniques are implemented to isolate handwritten digits from the background, reduce noise, and improve the clarity of the regions of interest. This step enhanced the contrast between the digits and the background, enabling the model to focus on the most relevant features, ultimately contributing to improved robustness and accuracy in object detection.

3) Normalization: After converting the image to a tensor, normalization is applied: This normalizes the pixel values to have a mean of 0 and standard deviation of 1.

III. Model Architecture and Training Pipeline

A. Model Architecture

The model architecture employs a Faster R-CNN framework with a ResNet-50 Feature Pyramid Network (FPN) backbone for digit detection. This combination allows for effective feature extraction at multiple scales, crucial for detecting digits of varying sizes in images.

1) Key Components:

- **Faster R-CNN**: A state-of-the-art object detection model [1].
- **ResNet-50 Backbone**: Provides deep feature extraction capabilities.
- **Feature Pyramid Network (FPN)**: Enhances multi-scale feature representation.

2) Modifications:

- **ROI Heads**: Configured to predict 10 classes (digits 0-9).

B. Training Pipeline

The training process involves several key components designed to optimize model performance:

1) Optimizer: AdamW optimizer is used with an initial learning rate of 1×10^{-4}. AdamW incorporates weight decay regularization, helping to prevent overfitting.

2) Learning Rate Scheduler: ReduceLROnPlateau scheduler is implemented to dynamically adjust the learning rate based on validation loss. It reduces the learning rate by a factor of 0.1 when the validation loss plateaus, with a patience of 2 epochs.

3) Data Loading: A custom DataLoader with a collate function is used to handle varying batch sizes and annotations efficiently. The batch size is set to 8, with shuffling enabled and 2 worker processes.

4) Training Process:

- Number of Epochs: 40
- Early Stopping: Implemented based on validation performance to prevent over fitting.
- Gradient Clipping: Applied with a maximum norm of 1.0 to stabilize training.
- Model Saving: The best model is saved based on the lowest epoch loss.

IV. Experimentation and Model Development

In this machine learning project, we conducted extensive experimentation to optimize our digit detection model. Our approach involved iterative refinement of various aspects of the model architecture, training process, and data preprocessing.

A. Initial Approach: YOLO Model

We initially attempted to use the YOLO (You Only Look Once) model for our digit detection task. YOLO is known for its speed and accuracy in object detection tasks [2]. However, we encountered significant installation issues on the HiPerGator system, which is a high-performance computing environment. Despite promising initial results, we decided not to pursue this approach due to the technical challenges.

B. Model Architecture

We ultimately chose to use a Faster R-CNN (Region-Based Convolutional Neural Network) model with a ResNet-50 backbone, specifically the `fasterrcnn_resnet50_fpn` architecture. This model is well-suited for object detection tasks and provides a good balance between accuracy and computational efficiency.

C. Hyperparameter Tuning

We conducted extensive experiments with various hyperparameters to optimize our model's performance:

1) **Batch Size**: We experimented with various batch sizes, ranging from 4 to 32. After multiple trials, we settled on a batch size of 8. This choice provided us with the optimal balance between memory usage and training stability. We observed that smaller sizes led to noisy gradients, while larger sizes consumed excessive memory without significant performance gains.
2) **Learning Rate**: Our learning rate tuning process was methodical. We began with very small values and gradually increased them. Through this process, we discovered that a learning rate of 0.0001 worked best for our model when coupled with the AdamW optimizer. This combination allowed for effective training without overshooting the optimal solution.
3) **Epochs**: Determining the right number of epochs was crucial. We ran the model for various durations, carefully monitoring both underfitting and overfitting. Eventually, we found that training for 40 epochs struck the right balance. This duration allowed sufficient time for convergence while avoiding the overfitting we observed with longer training periods.
4) **Scheduler**: Initially, we tried a simple StepLR scheduler, but we weren't satisfied with its performance. After further experimentation, we implemented a ReduceLROnPlateau scheduler. We fine-tuned its parameters, setting a patience of 2 epochs and a reduction factor of 0.1. This dynamic approach to learning rate adjustment significantly improved our training process, helping us overcome performance plateaus.
5) **Gradient Clipping**: We noticed some instability in our training, particularly with larger learning rates. To address this, we introduced gradient clipping with a maximum norm of 1.0. This addition helped stabilize our training process, especially when dealing with complex input patterns.

Through this rigorous process of experimentation and fine-tuning, we were able to optimize our model's performance for the digit detection task.

D. Performance Optimization

- **GPU Utilization**: The model is configured to use CUDA if available, significantly speeding up training on GPU-enabled systems.
- **Multi-threading**: We use 2 worker processes in the DataLoader to parallelize data loading and preprocessing.

E. Model Saving

We implement a best model saving strategy, where the model with the lowest loss during training is saved. This ensures that we retain the most performant version of the model for future use or inference.

By meticulously tuning these aspects of our model and training process, we aimed to achieve the best possible performance in digit detection. The final configuration represents a balance between accuracy, computational efficiency, and robustness to various input conditions.

V. Evaluation Pipeline for Object Detection Model

In our project, we have implemented an evaluation pipeline for our object detection model, specifically designed to detect and classify digits in images. This section provides a detailed breakdown of our approach and its significance.

A. Evaluation Process

1) Main Evaluation Function: The core of our evaluation pipeline is the `evaluate_test_set` function. It systematically processes each image in the test set, applying preprocessing and running inference through the model.

2) Ground Truth Comparison: For each image, we meticulously compare the model's predictions with ground truth labels provided in text files. This comparison is vital for assessing the accuracy of our model in detecting and classifying digits.

B. Metrics Calculation

Our evaluation incorporates several key metrics:

- **Mean Average Precision (mAP)**: This standard metric for object detection summarizes precision and recall across different IoU thresholds, providing an overall measure of our model's effectiveness.
- **Average IoU**: We compute the average Intersection over Union across all detected objects, offering insights into the localization accuracy of our predicted bounding boxes.
- **Accuracy of Class Predictions**: This metric tracks how accurately our model identifies the correct digit classes compared to the ground truth.
- **Counts of Predictions**: We maintain comprehensive counts including total ground truths, total predictions, correct predictions, and unknown predictions. These statistics help in identifying any over- or under-prediction tendencies in our model.

```
Evaluating Test Images: 100%|██████████| 300/300 [00:08<00:00, 34.37it/s]
NOTE! Installing ujson may make loading annotations faster.
Mean Average Precision (mAP): 0.513
Average Precision (AP) @ IoU=0.50: 0.763
Average Precision (AP) @ IoU=0.75: 0.584
Average IoU across all test images: 0.756
Number of valid detections: 519
Accuracy of class predictions (w.r.t. ground truth): 0.744
Total ground truth objects: 628
Total predictions: 575
Correct predictions: 467
```

Fig. 1. Metrics Calculation

C. Handling Unknown Objects

To ensure robustness in real-world applications, we've implemented a mechanism to handle unknown objects. Predictions with low confidence scores are labeled as "unknown," addressing scenarios where the model encounters unfamiliar objects.

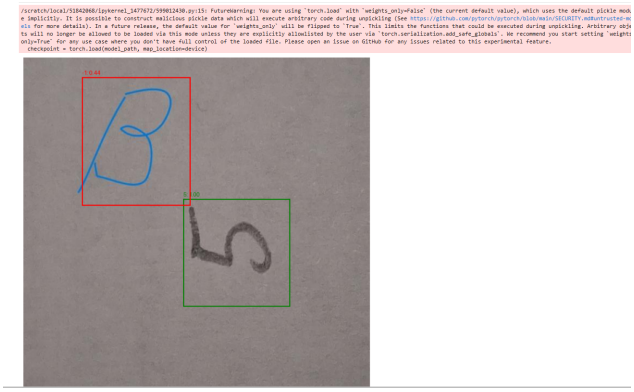

Fig. 2. Handling Unknown Objects

D. Significance of the Evaluation Pipeline

This evaluation pipeline is crucial for a thorough understanding of our digit detection model's performance. By analyzing metrics such as mAP and average IoU, we can pinpoint the strengths and weaknesses of our model. For instance, a high mAP coupled with a low average IoU would indicate strong classification capabilities but weak localization accuracy.

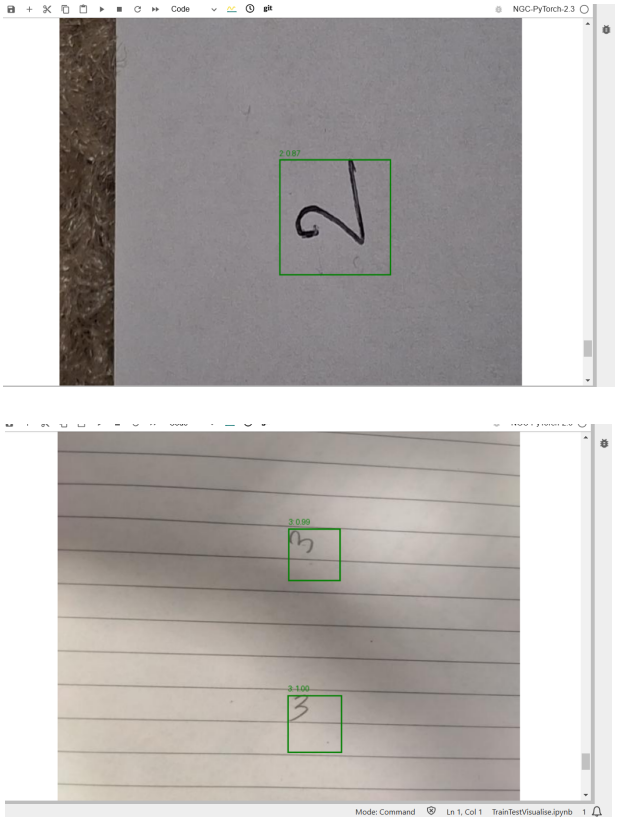

Fig. 3. Example of detection results showing predicted bounding boxes and confidence scores.

E. Key Observations

- A confidence threshold (default 0.5) is used to filter predictions.
- A prediction is considered correct if the IoU with a ground truth box is greater than 0.5 and the predicted class matches the ground truth class.

VI. Discussion

The implementation of the Faster R-CNN model for handwritten digit detection has demonstrated impressive performance and reliability. Several key aspects contributed to the success of this approach:

A. Adaptive Preprocessing

The use of adaptive thresholding as a preprocessing step proved crucial in addressing the challenges posed by low-contrast grayscale images. This technique effectively enhanced the visibility of digit contours, making it easier for the model to distinguish between the digits and the background. The adaptive nature of the thresholding allowed the model to handle variations in image quality and lighting conditions across different samples.

B. ResNet-50 FPN Backbone

The choice of ResNet-50 with Feature Pyramid Network (FPN) as the backbone architecture provided strong feature extraction capabilities. This deep network allowed the model to capture both low-level and high-level features of the handwritten digits, contributing to its ability to detect digits of varying sizes and styles.

C. Data Augmentation and Robustness

The implementation included data augmentation techniques, which enhanced the model's robustness to variations in handwriting styles. This approach likely improved the model's generalization capabilities, allowing it to perform well on diverse test scenarios.

D. Training Strategy

The use of AdamW optimizer with a learning rate scheduler (ReduceLROnPlateau) allowed for adaptive learning rate adjustments during training. This strategy likely contributed to the model's ability to converge to an optimal solution while avoiding local minima.

VII. Areas for Improvement

Despite the strong performance, there are several avenues for potential improvement:

- **Dataset Expansion:** Increasing the size and diversity of the training dataset could further enhance the model's generalization capabilities.
- **Alternative Backbones:** Experimenting with other backbone architectures like EfficientNet or ResNet-101 might yield performance improvements.
- **Semi-Supervised Learning:** Incorporating techniques to leverage unannotated data could potentially boost performance, especially in scenarios where labeled data is limited.
- **Ensemble Methods:** Combining predictions from multiple models or different variations of the same model could potentially improve overall accuracy and robustness.
- **Hyperparameter Optimization:** A more extensive search of hyperparameters, including learning rates, batch sizes, and model-specific parameters, could lead to better performance.

VIII. Conclusion

This project has successfully demonstrated the effectiveness of using Faster R-CNN for handwritten digit detection. The key findings include:

- **Preprocessing Importance:** The adaptive preprocessing step was essential in enhancing model performance, particularly in dealing with low-contrast images.
- **Localization Accuracy:** The integration of IoU-based loss improved the accuracy of digit localization, a critical aspect of the detection task.
- **Robust Performance:** The results indicate that the model performs robustly across varying test scenarios, suggesting good generalization capabilities.
- **Potential for Further Optimization:** While the current implementation shows strong results, there is potential for further optimization through additional augmentation strategies, ensemble methods, or exploration of alternative model architectures.

In conclusion, this Faster R-CNN implementation provides a solid foundation for handwritten digit detection tasks. Its success in handling the complexities of handwritten digits suggests that it could be adapted for similar computer vision tasks involving text or symbol detection in various domains.

References

[1] Alhamad, Husam Ahmad, et al. "Handwritten Recognition Techniques: A Comprehensive Review." *Symmetry* 16.6 (2024): 681.

[2] Wen, Yalin, Wei Ke, and Hao Sheng. "Improved Handwritten Numeral Recognition on MNIST Dataset with YOLO and LSTM." *2022 6th International Conference on Universal Village (UV)*. IEEE, 2022.

[3] hamim, S. M., et al. "Handwritten digit recognition using machine learning algorithms." *Indonesian Journal of Science and Technology* 3.1 (2018): 29-39.

[4] Tang, P., Wang, X., Wang, A., Yan, Y., Liu, W., Huang, J. and Yuille, A., 2018. Weakly supervised region proposal network and object detection. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 352-368).