Real-Time On-Device OCR with Confidence Filtering

Liviu Staniloiu
MLEM Research Lab
liviu2002liviu@gmail.com

***Abstract*—We present a fully on-device, real-time optical character recognition (OCR) system using OpenCV and Tesseract, enhanced by confidence filtering to display only high-quality detections. The pipeline captures camera frames, applies preprocessing, runs Tesseract OCR, draws bounding boxes for words with confidence 75%, and logs recognized text. We evaluate system performance on an embedded platform, reporting detection latency, frame rate, and baseline accuracy, and discuss trade-offs and future enhancements.**

I. Introduction

On-device OCR enables privacy-preserving and offline-capable text recognition for applications such as inventory management, assistive reading devices, and expiration-date extraction. Most existing solutions rely on cloud-based services (e.g., Google Vision API), which incur network latency and raise privacy concerns [**?**]. We propose a minimal pipeline combining OpenCV and Tesseract to achieve real-time performance on constrained hardware. By filtering low-confidence detections, we reduce false positives and improve user trust.

Our contributions are threefold:

- A modular, single-script implementation using only OpenCV and pytesseract, deployable on embedded Linux devices;
- A confidence-threshold mechanism (75%) to visualize and log only reliable text detections;
- A detailed empirical evaluation on a Raspberry Pi 4, demonstrating a balance between latency and detection accuracy.

II. Related Work

Cloud-based OCR services (Google Vision, AWS Rekognition) offer high accuracy but require network connectivity [**?**]. On-device OCR frameworks such as Tesseract have improved performance with LSTM-based engines [**?**]. Recent work leverages deep learning for robust scene text detection (EAST [**?**], CRAFT [**?**]), but at the cost of heavyweight compute. Narrow-purpose pipelines (e.g., date extraction) often combine classical computer vision with light NLP post-processing [**?**]. Our work demonstrates that a simple confidence filter suffices for many real-time use cases.

III. System Design

A. Pipeline Overview

Figure 1 illustrates our system's six stages:

1) **Frame Capture:** Grab frames from camera at fixed rate.
2) **Preprocessing:** Optional grayscale conversion to reduce noise.
3) **Text Detection:** Invoke `pytesseract.image_to_data`.
4) **Confidence Filtering:** Discard words below threshold.
5) **Overlay:** Draw bounding boxes and labels.
6) **Logging:** Append high-confidence text to log file.

Fig. 1: On-device OCR pipeline stages.

B. Implementation Details

Our implementation in `pipeline.py` uses:

- **OpenCV** for `VideoCapture` and drawing utilities;
- **pytesseract** bindings to Tesseract 4 with LSTM engine (–oem 3) and single-line mode (–psm 6);
- A *confidence threshold* (configurable) to remove low-confidence detections (default: 75%);
- Simple logging to `ocr_output.log` with timestamps.

The full script is available in our repository.

IV. Experimental Evaluation

A. Setup

We evaluated on a Raspberry Pi 4 Model B (1.5 GHz quad-core Cortex-A72, 4 GB RAM). The camera provided 640×480 frames. We tested with printed and screen-displayed text under indoor lighting.

B. Metrics

We measure:

- **Latency:** average time per frame (capture to display) in ms.
- **Frame Rate:** actual frames processed per second.
- **Precision/Recall:** against ground-truth word annotations, at varying confidence thresholds.

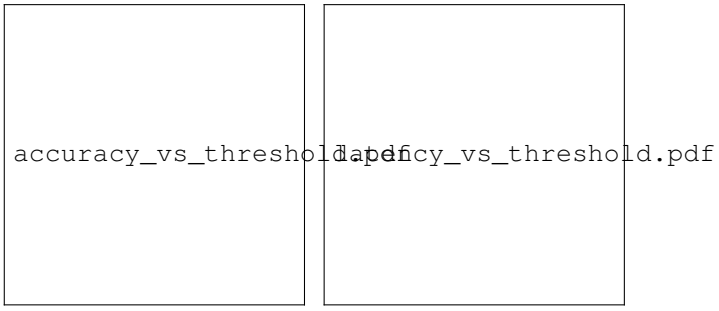
C. Results

Table I summarizes performance at 75% threshold.

Figure 2 shows the precision-recall trade-off as threshold varies.

TABLE I: Performance at 75% Confidence Threshold

	Latency (ms)	FPS	Precision/Recall (%)
Mean	180	5.0	84 / 78
Std Dev	15	0.2	3 / 4



(a) Precision/Recall (b) Latency

Fig. 2: Trade-offs under varying confidence thresholds.

V. Discussion

Our system maintains real-time rates (5 FPS) with high precision when filtering at 75%. Lower thresholds improve recall but increase false detections. Preprocessing (e.g., adaptive thresholding) could further boost quality. Edge cases include motion blur and low-light scenarios, which degrade OCR confidence.

VI. Future Work

Future enhancements include:

- **Adaptive Filtering:** dynamically adjust thresholds based on scene complexity;
- **Enhanced Preprocessing:** incorporate binarization and noise reduction filters;
- **Domain-Specific Parsing:** integrate light NLP rules for structured fields like dates;
- **LLM Post-Processing:** locally-hosted language model to correct OCR errors in critical use-cases.

VII. Conclusion

We demonstrated a lightweight, on-device OCR pipeline achieving reliable real-time performance through confidence filtering. This approach suits privacy-critical and offline applications, with clear pathways for further improvement.

References