# Technical Report: Robust Vision-Based Symbol Classification for Robocon 2026

Prabhudutta Prusti

December 31, 2025

**Abstract**

This report details the development of a lightweight, real-time Computer Vision system designed for the Robocon 2026 autonomous robot. The system classifies three distinct categories—Robocon Logo, Oracle Bone Script, and Random/Fake Patterns—under strict computational constraints ($\leq$ 10 MB model size, $\leq$ 20 ms latency). We propose a custom-tuned MobileNetV3-Small architecture enhanced by a active learning pipeline, domain-specific geometric augmentation for cubic objects, and a robust grayscale preprocessing strategy to handle variable Red/Blue backgrounds.

## 1 Problem Statement & Constraints

The objective was to identify symbols on "Kung-Fu Scroll" (KFS) blocks. The challenge involved distinguishing between valid Oracle Bone characters and visually similar "Fake" symbols, as well as identifying the official Robocon Logo.

- **Model Type:** Convolutional Neural Network (CNN)
- **Maximum Model Size:** $\leq$ 10 MB (after optimization)
- **Inference Latency:** $\leq$ 20 ms per frame
- **Environment:** Variable lighting; symbols appear on Red or Blue blocks.

## 2 Methodology

### 2.1 Automated Dataset Creation (Dilation Technique)

To generate a clean dataset from raw rulebook images, we developed a custom extraction script (`extract_symbols.py`). A major challenge was that some Oracle characters (e.g., "Two Trees") consist of disjoint strokes. Standard contour detection identified them as separate objects.

- We applied a **Dilation Operation** (kernel $20 \times 20$) to the binary thresholded image. This "thickened" the white pixels, bridging the gaps between strokes and forcing the algorithm to perceive the disjoint parts as a single solid block.
- **Cropping Logic:** While the dilated "blob" was used for detection, the final crop was taken from the original image. This ensured we preserved the sharp edges and high quality of the symbols while accurately segmenting them.

### 2.2 Model Architecture Selection

We selected **MobileNetV3-Small** as our architecture. This decision was driven by the strict hardware constraints outlined in the problem statement:

1. **Size Constraint ($\leq 10$ MB):** Standard models like ResNet-50 ($\approx 90$ MB) far exceed the limit. MobileNetV3-Small, after export to ONNX, occupies only $\approx 2.5$ MB, providing a 75% safety margin below the 10 MB cap.

2. **Latency Constraint ($\leq 20$ ms):** The architecture utilizes **depthwise separable convolutions**, which drastically reduce the number of floating-point operations (FLOPs). On our testing hardware, this yielded an average inference time of 4.1 ms, safely clearing the 20 ms requirement.

3. **Stroke Focus:** To prioritize shape over color, we modified the first layer to accept **1-channel Grayscale** input. This focuses the model's attention on the stroke patterns of the Oracle Bone script rather than background noise.

## 3 Challenges & Engineering Solutions

### 3.1 The Logo Problem

The dataset contained many examples of Oracle characters but only *one* source image for the Robocon Logo. Initially, the model failed to detect the logo, often classifying it as an Oracle Bone due to bias. So, we implemented the following solution:

1. **Data Duplication:** We physically duplicated the logo image 20 times in the dataset to artificially increase its prevalence.

2. **Weighted Loss Function:** In `train.py`, we implemented a weighted CrossEntropyLoss. We assigned a weight of **1.2** to the Logo class, forcing the model to penalize mistakes on the logo more heavily than mistakes on the abundant Oracle class.

```
# [Logo, Oracle, Fake, Background]
class_weights = torch.tensor([1.2, 1.0, 1.0, 0.5]).to(device)
criterion = nn.CrossEntropyLoss(weight=class_weights)
```

Listing 1: Class Weight Implementation

### 3.2 Active Learning

A critical issue was the model's inability to distinguish between "Fake" symbols and "Oracle" characters, as they share similar stroke characteristics.

**Solution:** We implemented a "Human-in-the-Loop" training system in `inference.py`. When the model made a mistake (e.g., calling a Fake symbol an Oracle Bone), the operator could press a specific key ('0', '1', '2') to instantly capture that specific frame and save it to the correct folder. This created a dataset rich in **Hard Negatives**—images that specifically confused the model—allowing it to learn the subtle decision boundaries during retraining.

### 3.3 Resolution Loss

During testing, the model struggled to detect symbols from a distance ($> 15$ cm). The resizing of the full webcam feed ($640 \times 480$) down to $224 \times 224$ caused small symbols to blur into unrecognizable blobs.

**Solution: Digital Zoom (Center Cropping)**
We modified the preprocessing pipeline in `inference.py` to crop the central 60% of the frame before resizing. This effectively acts as a 1.6× optical zoom, increasing the pixel density of the symbol without increasing the input resolution or computational load.

### 3.4 Domain-Specific Augmentation (Rotation & Color)

**The Rotation Problem:** Initially, we used random rotation ($\pm 10^\circ$). However, testing with physical cards revealed the model failed at $90^\circ$ angles. Since the KFS blocks are cubes, symbols appear at distinct $0^\circ, 90^\circ, 180^\circ, 270^\circ$ orientations.

- **Solution:** We replaced random rotation with a deterministic script (`augment_rotation.py`) that generated exact 90°/180°/270° copies of the entire dataset.

**The Color Problem:** The symbols appear on Red and Blue blocks in the competition. To prevent the model from failing on dark backgrounds, we utilized grayscale preprocessing combined with synthetic data augmentation (`augment_colors.py`) to simulate symbols on low-contrast Red/Blue backgrounds while also adding the pre-existing colored symbols in the rule-book.

### 3.5 The Background Class

Initially, the model would hallucinate "Oracle Bone" predictions when looking at an empty background.

**Solution:** We introduced a 4th class, **Background**. We captured images of empty walls, floors, and tables. In `train.py`, we assigned this class a lower weight (0.5) to prevent it from overpowering the symbols, allowing the system to output a "SCANNING..." state when no object is present.

## 4 Deployment Pipeline

### 4.1 Temporal Smoothing

To solve the issue of prediction flickering (rapidly switching between "Fake" and "Oracle"), we implemented a **Rolling Buffer** (Deque size=5) in the inference script. The final prediction is the average probability of the last 5 frames. This provides a stable, confident output stream.

### 4.2 Performance Metrics

- **Inference Speed:** $\approx$ 4.1 ms per frame (Laptop CPU), well within the 20 ms limit.
- **Model Size:** 2.5 MB (ONNX), significantly under the 10 MB limit.
- **Accuracy:** The system achieves near 100% robustness on detecting the Logo, distinguishes Fake vs. Real symbols, and handles rotation for cubic blocks.

## 5 Conclusion

We successfully developed an end-to-end vision solution for Robocon 2026. By combining architectural optimizations (MobileNetV3 Grayscale) with practical engineering solutions (Active Learning, Domain-Specific Augmentation, and Digital Zoom), the system meets all technical constraints and is robust to environmental variations like distance, rotation, and background color.