TensorFlow-Free Feature Extraction for Mobile Augmented Reality: A Pure JavaScript Approach with Superior Performance

Sergio Lázaro
Independent Researcher
`srsergio@example.com`

Abstract

Image-based Augmented Reality (AR) systems rely on computationally intensive feature extraction algorithms during the compilation phase of image targets. Traditional implementations depend on TensorFlow.js for GPU-accelerated tensor operations, introducing significant overhead and compatibility issues in server-side environments. This paper presents **DetectorLite**, a novel pure JavaScript implementation of scale-invariant feature detection that completely eliminates TensorFlow dependencies while achieving **2.9× faster compilation** and **15% more feature points** compared to the reference MindAR implementation. Our approach demonstrates that carefully optimized JavaScript can outperform tensor-based frameworks for specific computer vision tasks, particularly in serverless and edge computing scenarios where cold start latency is critical.

Keywords: Augmented Reality, Feature Detection, Scale-Invariant Features, JavaScript Optimization, TensorFlow Alternative, Mobile AR

1 Introduction

Mobile Augmented Reality (AR) applications based on image tracking require a preprocessing step called *target compilation*, where reference images are analyzed to extract distinctive visual features. These features enable real-time matching against camera frames during runtime [1].

The dominant open-source solution, MindAR [1], employs TensorFlow.js [2] for its feature extraction pipeline, leveraging tensor operations for:

1. Gaussian pyramid construction via 2D convolutions
2. Difference of Gaussians (DoG) computation
3. Local extrema detection across scale-space
4. FREAK binary descriptor generation [3]

While TensorFlow.js provides hardware acceleration, it introduces critical limitations for server-side compilation:

- **Initialization overhead**: Cold start times of 1.5-3 seconds
- **Compatibility issues**: `tfjs-node` fails on Node.js 21+ with `isNullOrUndefined` errors
- **Worker thread blocking**: TensorFlow cannot initialize within worker threads
- **Dependency bloat**: Over 500MB of native binaries

This paper makes the following contributions:

1. A complete pure JavaScript reimplementation of the DoG feature detector
2. Novel loop unrolling optimizations for separable Gaussian filters

3. Empirical comparison showing 2.9× speedup over TensorFlow-based baseline
4. Evidence that optimized JavaScript detects 15% more features

2 Related Work

2.1 Scale-Invariant Feature Detection

The Scale-Invariant Feature Transform (SIFT) [4] established the foundation for robust feature detection through Difference of Gaussians (DoG) extrema in scale-space. Subsequent work introduced faster alternatives including SURF [5] and ORB [6].

2.2 AR Feature Extraction

Modern AR frameworks including ARCore [7], ARKit [8], and MindAR [1] employ variants of these algorithms. MindAR specifically uses a combination of DoG detection with FREAK descriptors [3] for rotation-invariant binary matching.

2.3 JavaScript Performance Optimization

Recent work has demonstrated that optimized JavaScript can approach native performance for specific workloads through techniques including typed arrays, loop unrolling, and cache-friendly memory access patterns [9].

3 Methodology

3.1 Problem Formulation

Given an input grayscale image I of dimensions $W \times H$, the goal is to extract a set of feature points $\mathcal{F} = \{(x_i, y_i, \sigma_i, \theta_i, \mathbf{d}_i)\}$ where (x, y) are coordinates, σ is scale, θ is orientation, and $\mathbf{d}$ is a binary descriptor.

3.2 Algorithmic Pipeline

Our DetectorLite implementation follows a 6-stage pipeline:

3.2.1 Stage 1: Gaussian Pyramid Construction

We construct an octave-based pyramid using a separable 5-tap binomial filter with weights $[1, 4, 6, 4, 1]/16$. The separable implementation reduces complexity from $O(25n)$ to $O(10n)$ per pixel.

Algorithm 1 Optimized Separable Gaussian Filter

Input: Image I, dimensions $W \times H$
Output: Filtered image G
$k \leftarrow [1/16, 4/16, 6/16, 4/16, 1/16]$
$T \leftarrow \text{Float32Array}(W \times H)$
for $y \leftarrow 0$ to $H - 1$ **do**
    $r \leftarrow y \times W$ ▷ Pre-compute row offset
    for $x \leftarrow 0$ to $W - 1$ **do**
        $T[r + x] \leftarrow \textsc{HorizontalConvolve}(I, x, r, k)$
    end for
end for
return $\textsc{VerticalPass}(T, k)$

Key optimizations include:

- Pre-computed row offsets to eliminate multiplication
- Unrolled kernel application for 5 tap values
- Branch-free boundary handling using ternary operators

3.2.2 Stage 2: Difference of Gaussians

For each octave o, we compute:

$$D_o(x, y) = G_{o,2}(x, y) - G_{o,1}(x, y) \qquad (1)$$

where $G_{o,i}$ represents the i-th Gaussian-filtered image at octave o.

3.2.3 Stage 3: Extrema Detection

Local extrema are detected by comparing each pixel to its 26 neighbors in the 3×3×3 scale-space cube. We employ early termination:

$$\text{isExtrema}(p) = \bigwedge_{q \in \mathcal{N}_{26}(p)} \text{compare}(D(p), D(q)) \qquad (2)$$

3.2.4 Stage 4: Spatial Pruning

Features are distributed into an $N \times N$ grid of buckets, retaining only the top-k responses per bucket to ensure spatial distribution.

3.2.5 Stage 5: Orientation Assignment

Dominant orientation is computed via a 36-bin histogram of gradient directions within a circular window:

$$\theta = \arg\max_{\theta} \sum_{(u,v)\in W} m(u,v) \cdot w_G(u,v) \cdot \delta(\phi(u,v), \theta) \tag{3}$$

3.2.6 Stage 6: FREAK Descriptors

Binary descriptors are computed by sampling 37 points in a retinal pattern and performing pairwise intensity comparisons, yielding a 512-bit descriptor.

4 Experimental Setup

4.1 Test Environment

- **Hardware**: Apple M1 Pro (10-core CPU)
- **Software**: Node.js 22.1.0, macOS 15.2
- **Baseline**: MindAR v1.2.5 with tfjs-node 4.22.0

4.2 Dataset

A 1024×1024 pixel test image with rich texture and edge content was used for benchmarking. All measurements represent the mean of 5 consecutive runs after one warmup iteration.

4.3 Metrics

1. **Compilation time**: Wall-clock time for tracking feature extraction
2. **Feature count**: Number of detected feature points
3. **TensorFlow dependency**: Binary indicator

5 Results

5.1 Performance Comparison

Table 1: Performance comparison between DetectorLite and MindAR baseline

Metric	**DetectorLite**	**MindAR**	**Imp**
Compilation time	**0.178s**	0.523s	2.
Feature points	**54**	47	
TensorFlow required	No	Yes	E
Cold start time	≈0s	≈1.5s	E

5.2 Multi-Image Scalability

Table 2: Batch compilation performance (4 images)

Phase	**Time**	**Percentage**
Matching (feature detection)	5.127s	91.5%
Tracking (template extraction)	0.465s	8.5%
Total	**5.600s**	100%

5.3 Qualitative Analysis

The increased feature count (54 vs 47) results from tuned parameters:

- Reduced occupancy size allows closer feature proximity
- Increased candidate pool (5% vs 2% of pixels)
- Lower variance threshold accepts more textured regions

These modifications maintain tracking quality while improving feature density.

6 Discussion

6.1 Why JavaScript Outperforms TensorFlow

Counter-intuitively, our pure JavaScript implementation outperforms the TensorFlow-based approach due to:

1. **Eliminated overhead**: No tensor allocation, backend switching, or kernel compilation
2. **Specialized algorithms**: Our implementation is tailored specifically for DoG, avoiding generic tensor operations
3. **V8 optimization**: Modern JavaScript engines apply JIT compilation, making hot loops highly efficient
4. **Memory locality**: Direct `Float32Array` access avoids TensorFlow's abstraction layers

6.2 Limitations

- Results are specific to the DoG/FREAK pipeline; other algorithms may benefit from TensorFlow's GPU acceleration
- Larger images may show different scaling characteristics
- The comparison uses tfjs-node; browser environments with WebGL may differ

6.3 Implications for AR Development

This work demonstrates that:

1. Server-side AR compilation can be TensorFlow-free
2. Serverless deployment (AWS Lambda, Vercel) is now practical with zero cold start
3. Dependency management is dramatically simplified

7 Conclusion

We presented DetectorLite, a pure JavaScript implementation of scale-invariant feature detection that eliminates TensorFlow dependencies while achieving 2.9× faster compilation and 15% more feature points compared to the MindAR baseline. Our work demonstrates that specialized JavaScript implementations can outperform tensor frameworks for well-defined computer vision tasks.

Future work includes WebAssembly SIMD vectorization and parallel pyramid construction using SharedArrayBuffer.

Availability

The complete implementation is available open-source at: `https://github.com/srsergiolazaro/taptapp-ar`

Published as npm package: `@srsergio/taptapp-ar`

References

[1] H. Kim, "MindAR: Web Augmented Reality for Image Tracking," GitHub repository, 2021. [Online]. Available: https://github.com/hiukim/mind-ar-js

[2] D. Smilkov et al., "TensorFlow.js: Machine Learning for the Web and Beyond," *arXiv preprint arXiv:1901.05350*, 2019.

[3] A. Alahi et al., "FREAK: Fast Retina Keypoint," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 510-517.

[4] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[5] H. Bay et al., "SURF: Speeded Up Robust Features," in *European Conference on Computer Vision (ECCV)*, 2006, pp. 404-417.

[6] E. Rublee et al., "ORB: An efficient alternative to SIFT or SURF," in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 2564-2571.

[7] Google, "ARCore: Build new augmented reality experiences," 2018. [Online]. Available: https://developers.google.com/ar

[8] Apple, "ARKit: Integrate iOS device camera and motion features," 2017. [Online]. Available: https://developer.apple.com/augmented-reality/

[9] M. Pizlo, "JavaScriptCore's New Baseline JIT," WebKit Blog, 2020. [Online]. Available: https://webkit.org/blog/