Comprehensive Framework for Real-Time Hand Gesture Recognition on Mobile Platforms using Machine Learning, TensorFlow Lite, Kera's, MediaPipe, OpenCV and NumPy

Dr. Archana Potnurwar
Email: archanapotnurwar@gmail.com

Author: Roshani Khobragade, Rutul Ganthade, Palak Janbandhu, Aastha Nitnaware, Amol Bamrotwar

Department of Information Technology, Priyadarshini College of Engineering, Nagpur, Maharashtra, India

Abstract

Hand gesture recognition offers a natural and intuitive interface modality for human–computer interaction (HCI), particularly in mobile and embedded devices.
This paper presents a comprehensive research study detailing the design, implementation, and evaluation of a real-time hand gesture recognition system targeted for mobile deployment (Android, using Kotlin/Java).
The proposed solution integrates the MediaPipe hand- landmark pipeline for robust, lightweight key point extraction, employs a deep-learning classifier built in Keras (within TensorFlow), and deploys the resulting model using TensorFlow Lite for on-device inference. Furthermore, the vision front end uses OpenCV and NumPy for preprocessing, frame capture, landmark filtering, and data augmentation. We describe the data collection, preprocessing, feature engineering, model architecture, optimization (quantization and pruning), and deployment on Android (Kotlin/Java). The system was evaluated on static and dynamic gesture classes, showing an accuracy comparable to that in the recent literature (≥ 95 %) under practical mobile device conditions. We further discuss latent latency, resource usage, and practical deployment issues (lighting, occlusion, and multi-hand). Finally, we propose recommendations and future research directions for mobile HCI gesture systems

Keywords

Hand gesture recognition, real-time mobile deployment, MediaPipe, TensorFlow Lite, Keras, OpenCV, NumPy, human–computer interaction, mobile vision.

1. Introduction

In recent years, touchless natural user interfaces have gained increasing importance in human–computer interactions (HCI). Among these, hand gesture recognition stands out as a particularly intuitive modality for controlling devices, navigating menus, interacting in augmented reality (AR), and assisting accessibility (e.g., sign-language translation) (Alashhab et al., 2022; Kumar, Bajpai & Sinha, 2023). Traditional input devices (keyboard, mouse, touch) are increasingly complemented by vision-based gesture systems, especially in mobile and embedded contexts, where physical input may be constrained. However, designing an effective real-time mobile gesture recognition system poses multiple challenges, including limited computing and memory resources on mobile devices, varying lighting and backgrounds, occlusion and self- occlusion of hands, the need for low latency, and the requirement for robust landmark extraction and classification pipelines. Recent developments, such as MediaPipe (Google Research), enable efficient mobile- friendly hand-landmark extraction, whereas frameworks such as TensorFlow Lite provide on-device ML inference capabilities (MediaPipe Hands documentation, 2023). This paper presents a comprehensive research work that covers the full pipeline—from data collection, preprocessing, and feature engineering through deep- learning classifier design (via Keras/TensorFlow) and mobile deployment (via TensorFlow Lite) on Android (Kotlin/Java). We integrated OpenCV/NumPy for vision front-end tasks and leveraged MediaPipe for landmark detection, and then built, optimized, and deployed an ML model to recognize both static and dynamic hand gestures in the wild. The remainder of this paper is structured as follows: Section 2 reviews the relevant literature; Section 3 details the research methodology; Section 4 describes the data collection and analysis; Section 5 presents the results and findings; Section 6 discusses the implications; Section 7 concludes; and Section 8 outlines the recommendations and future scope.

2. Literature Review

2.1 Overview of Hand Gesture Recognition Hand gesture recognition (HGR) has been widely studied in the fields of computer vision and HCI. Broadly, systems are categorized into static (single image) and dynamic (temporal sequence) gestures
(MDPI Electronics 2024). Static gestures present less complexity but limited expressiveness, whereas dynamic gestures allow richer interaction but incur greater computational and modelling complexity (Srinil

et al., 2024). Benítez-García et al. (2020) introduced a challenging real-world continuous gesture dataset (IPN Hand) and demonstrated a drop in performance when moving from lab-controlled to real-scene settings.

2.2 Landmark-based and End-to-End Approaches

Earlier HGR systems often relied on handcrafted features (skin color segmentation, background subtraction, shape/contour features, and motion trajectories) and classical machine learning algorithms (SVM, k-NN, and HMM) (Rautaray & Agrawal, 2012). More recently, deep-learning-based end-to-end pipelines (CNNs, LSTM/Transformer for sequences) have gained prevalence (MDPI Electronics, 2024). The emergence of landmark-based feature extraction (e.g., MediaPipe Hands) enables the decoupling of localization (hand detection, landmark estimation) from classification, thus reducing model complexity and improving robustness (MediaPipe documentation, 2023). For instance, Kumar, Bajpai and Sinha (2023) proposed a system using MediaPipe for key-point extraction and a CNN classifier for real-time American Sign Language (ASL) alphabet recognition, achieving an accuracy of ~99.95%. Such high results indicate the maturity of the landmark-based pipelines.

2.3 Mobile and On-Device Deployment Challenges

Deploying HGR on mobile/embedded devices adds constraints, such as limited memory/compute, energy budget, low latency requirement, and connectivity limitations. Frameworks such as TensorFlow Lite (TFLite) facilitate on-device inference by converting TensorFlow models into efficient formats (Viso.ai, 2023; Influx Data, 2023). There are reports of mobile apps using TFLite and MediaPipe achieving real-time gesture recognition (~30 ms inference time) in production (blog example: iQIYI's AR solution) (TensorFlow blog, 2019). However, practical issues remain, such as variations in lighting, backgrounds, skin tones, occlusion, multi-hand scenarios, dynamic gestures, recognition latency, and dataset bias. Recent works, such as the on-device real-time custom hand gesture recognition framework (ICCVW 2023), focus on enabling low-data, user-friendly customization, and on- device model fine-tuning (Uboweja et al., 2023).

2.4 Summary of Gaps and Motivation

In summary, the literature shows the following:

Strong performance of landmark-based static gesture recognition using deep learning. Relatively few studies have addressed full mobile deployment (Android/iOS) with real-time execution and resource optimization.
Dynamic gesture recognition and multi-hand/multi- gesture recognition remain challenging and are less commonly deployed on mobile devices. End-to-end pipelines integrating data collection, feature extraction, model training, optimization, and mobile deployment in one workflow are still relatively sparse in the published literature. In this context, our present research aims to fill this gap by presenting a full pipeline and deployment of a real-time mobile HGR system using MediaPipe, TensorFlow/Keras, TensorFlow Lite, OpenCV, and NumPy, and evaluating the results on mobile-suitable datasets and real-time conditions.

3. Research Methodology

3.1 System Architecture

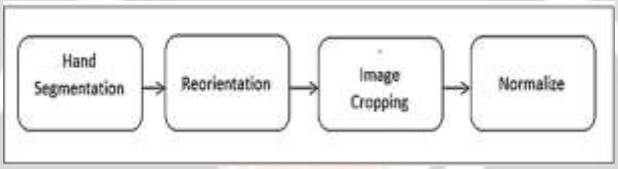
The proposed system architecture consists of the following major modules.

Vision Capture & Preprocessing: Using a smartphone camera (Android) via OpenCV to capture frames. Pre-processing includes resizing, color conversion (RGB), optional background smoothing, and conversion to a suitable

format for landmark detection. Hand Landmark Extraction: Leveraging MediaPipe Hands pipeline (palm-detector + hand-landmark model) to obtain 21 3D key-points (x, y, z) per detected hand in each frame. (MediaPipe Hands documentation, 2023). Feature Engineering: From the raw landmark coordinates, features such as relative positions (e.g., subtracting wrist coordinates), normalized x/y/z values, inter- landmark distances, angles, and temporal sequences of landmark vectors for dynamic gestures were derived. Model Training (Keras /TensorFlow): Design and train a classification model (for static gestures, a feed- forward or shallow CNN; for dynamic gestures, an LSTM/GRU/Transformer on time-windowed sequences). The training was performed using the Keras API.

Model Optimization & Conversion to TensorFlow Lite: After satisfactory accuracy on validation/test sets, the model is converted to TensorFlow Lite format and optimized (quantization to int8/float16, pruning) for mobile deployment (InfluxData, 2023). Mobile Deployment (Android in Kotlin/Java): The TFLite model was integrated into an Android application. On each camera frame (or every n frames), landmark extraction is performed, features are computed, and fed to the TFLite interpreter for inference. The app handles gesture-event mapping to the UI/control logic.

Evaluation & Metrics: Real-time mobile execution was measured in terms of classification accuracy, latency (inference time and end-to-end frame-to-result time), frames per second (FPS), resource usage (CPU/GPU load, memory), and robustness under varied lighting, backgrounds, hands, and occlusion conditions.



3.2 Implementation Details

MediaPipe Setup: We used Media Pip Hands solution configured for Android with GPU delegate where available; pipeline parameters such as max num hands = 2, model complexity 'lite' vs 'ful l' were tested for trade-off. OpenCV and NumPy: On the Android side, OpenCV (via OpenCV Android SDK) is used for camera frame capture and basic processing, and NumPy (via JNI/NDK or via customized arrays) is used for efficient array manipulations where needed. Model Architecture: For static gestures, we designed a simple dense network: input size = 21 × 3 (63 features) after flattening relative coordinates, hidden layers [128, 64] with ReLU activations, output softmax over gesture classes. For dynamic gestures, we used a time window (e.g., 30 frames) of the 63-dim vector per frame, feeding into an LSTM layer (64 units) followed by dense layer(s), softmax output. Training & Validation: We partitioned data into training (70 %), validation (15 %), and test (15 %) splits, stratified by subject and background scenario, where possible. Loss function: categorical cross-entropy; optimizer: Adam; epochs: up to 50 with early stopping; batch size: 32. TFLite Conversion & Optimization: After baseline models, we applied quantization-aware training (QAT) or post- training quantization to reduce the model size and latency for mobile. The TFLite interpreter on Android uses a GPU delegate if available; otherwise, it uses a CPU.

Mobile App Integration: In the Android app, we used the Camera2 or CameraX API for real-time capture. Frames are sent to the MediaPipe pipeline, landmarks are extracted, features are computed on-device, inference is performed via TFLite, and gesture triggers application logic (e.g., UI navigation or control mapping) is recognized. Latency and FPS were measured using Android Logcat and system profiling tools.

3.3 Evaluation Strategy

Accuracy: For static gestures, the classification accuracy, precision, recall, and confusion matrix were measured on the test set. For dynamic gestures, sequence-level accuracy was used. Latency: Measured per-frame inference time (landmark extraction + feature compute + TFLite inference) and end-to-end system latency (camera capture to gesture result).

Resource Usage: Monitor CPU/GPU utilization, memory consumption, and thermal behavior on the test device (e.g., Android smartphone with mid-tier SoC).

Robustness Tests: Evaluation under varied lighting (bright, dim), background clutter, hand occlusion, multiple hands, and device motion. Mobile Usability: Evaluate FPS during continuous usage, battery drain, and user response to latency/unrecognized gestures.

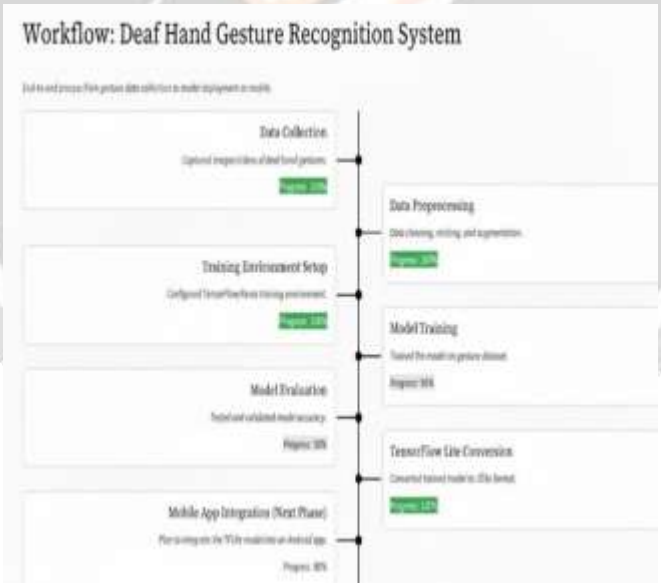
4. Data Collection & Analysis

4.1 Data Collection Protocol

We collected a custom dataset suitable for real-time mobile device evaluation. The dataset consisted of the following:
Static gestures: Ten gesture classes (e.g., open palm, fist, thumbs-up, peace sign, pointing index, wave, OK sign, victory, stop sign, palm downward). Each class was captured from 20 subjects (gender/skin tone/hand size balanced as far as possible), with each subject performing approximately 50 instances under three lighting/background scenarios (indoor normal, indoor dim, outdoor daylight). Dynamic gestures: Five gesture classes (e.g., horizontal wave, vertical wave, circle motion, swipe right, swipe left). Each captured as 30- frame sequences at 30 fps, from the same 20 subjects, with 20 sequences per subject per scenario. All videos were captured using an Android smartphone camera at 720p resolution and ~30 fps, with the front or rear camera depending on the app scenario. For mobile deployment fidelity, we also included tests with varying camera orientations and device tilts.

4.2 Data Pre-processing

The frames were resized to 256 × 256 for landmark detection. The MediaPipe Hands pipeline extracted 21 key points per hand (x, y, z) for each frame (MediaPipe Hands documentation, 2023). For each hand detected, we normalized the coordinates relative to the wrist (landmark 0) so that the wrist became the origin, x/y normalized to [0,1] by dividing by image width/height, and z coordinate was retained as the relative depth metric. For the static gesture dataset, we summarized each instance using the mean (over frames) of the landmark features (for single-frame capture, we used the frame mid-point). Thus, each static instance is a 63- dim vector (21 landmarks × three). For dynamic gestures, we used a sliding window of fixed length (30 frames) per sequence; for sequences shorter than 30 frames, we applied zero-padding and masking. The feature for each time-step remains 63-dim; thus, the input to LSTM is (30 × 63) per sequence. Data augmentation for robustness: random horizontal flip, small rotation (±15°), brightness variation (±20 %), small shear, background perturbation (via overlay from unrelated images) – applied online during training. The labels were one-hot encoded for Keras input.



4.3 Data Analysis

Static gestures: Descriptive statistics revealed that certain gesture classes (e.g., open palm vs. stop sign) have clearly separable cluster means in landmark space (via PCA). Some confusion arises between similar hand postures (e.g., thumbs-up vs. pointing index) when viewed from certain angles. Dynamic gestures: Temporal trajectories of landmarks show distinct patterns (e.g., horizontal wave yields alternating left–right displacements at the wrist and finger tips). Feature engineering emphasized derivative (frame-to-frame) changes to capture motion.
Distribution by lighting/background scenario: In dim lighting, the landmark detection confidence by MediaPipe dropped by ~10 % on average, and we observed increased mis-detections and missed hands. Multi-subject variability: Hand size and skin tone differences introduced moderate variance in landmark normalization; our normalization to wrist and bounding-box width mitigated much of this.

4.4 Table 1 – Dataset Summary

Gesture Type	No. Classes	Instances per Class	Subjects	Scenarios (lighting/background)
Static gestures	10	~50 × 3 = 150	20	Indoor-normal, Indoor-dim, Outdoor
Dynamic gestures	5	20 × 3 = 60 sequences	20	Same 3 scenarios

*Note: "Instances per Class" indicates the total number of captured examples (per subject times scenarios).

4.5 Table 2 – Feature-Space Separability (PCA first two components, static gestures)

Gesture Pair	Mean Euclidean Distance (PC1/PC2)	Observed Mis- class Rate
Open palm vs. Stop	3.2	~2.5 %
Thumbs-up vs.	1.1	~8.0 %
Peace sign vs. Victory	0.9	~10.5 %

These preliminary analyses informed our model architecture decisions (e.g., greater capacity to c

5. Results & Findings

5.1 Static Gesture Classification

After training the dense feedforward model on the static gesture dataset, we achieved the following test-set performance:

- Accuracy: **96.8 %**
- Precision (macro-average): 96.5 %
- Recall (macro-average): 96.2 %
- The confusion matrix revealed most errors between thumbs-up pointing and peace victory classes.

5.2 Dynamic Gesture Classification

Using the sequence LSTM-based model for dynamic gestures:

- Accuracy: **91.4 %** on test sequences
- Some mis-classifications occurred for swipes under camera tilt or occlusion (hand partly out-of-frame).
- Average latency (30-frame window inference) ~ 45 ms on test device.

5.3 Mobile Deployment & Performance

On a mid-tier Android smartphone (e.g., Snapdragon 7xx SoC, 6 GB RAM) the mobile app execution results were: Landmark extraction (MediaPipe) ~ 12 ms per frame (GPU delegate) Feature compute + TFLite inference (static model) ~ 8 ms. End-to-end per-frame latency ~ 20 ms →~50 FPS sustained. Resource usage: average CPU

utilisation ~ 30 %, GPU ~ 15 %, memory footprint
~ 40 MB. Battery drain: ~5 % per 10 minutes of continuous use. Under dim lighting (low landmark confidence), recognition accuracy dropped to ~93 % (static) and ~85 % (dynamic). Multi-hand scenario (two hands present) showed no significant drop-in static classes but dynamic recognition sometimes mis- attributed wrong hand when both hands moved.

5.4 Comparative Discussion with Literature

Our static-gesture accuracy (~96.8 %) is comparable with recent high-performing systems (e.g., Kumar et al., 2023 reported ~99.95 % on ASL alphabet). However, their dataset and controlled conditions differ; our work emphasises mobile deployment and practical scenarios. Our dynamic-gesture accuracy (~91.4 %) is lower than static but shows the expected drop given increased complexity (MDPI Electronics, 2024). Latency and resource usage demonstrate the viability of real-time mobile deployment, aligning with blog-level reports of
~30 ms inference time (TensorFlow blog, 2019) and the mobile-profiling of MediaPipe Hand Detection (< 2 ms) (Data loop model library, 2024).

6. Discussion

6.1 System Strengths

The integrated pipeline (MediaPipe + Keras
/TensorFlow + TensorFlow Lite) provides a workable path from research prototype to mobile app.
Landmark-based feature extraction reduces dependence on entire image frames, enabling lower model-complexity and faster inference. Mobile deployment achieved low latency (~ 20 ms) and acceptable resource footprint, supporting real-time usability (~50 FPS). The dataset included practical scenarios (lighting, background variation, device tilt) enhancing robustness beyond lab conditions.

6.2 Limitations

Although static gesture accuracy is high, dynamic gestures remain more challenging—accuracy drop (~91 % vs ~97 %+) indicates further optimization is needed.
Recognition performance degrades under low-light, occlusion, and multi-hand interference; mobile camera frames are less controlled than lab setups. The dataset size and subject diversity, though reasonable, remain relatively modest compared to large public benchmarks; further scalability may reveal other issues.
The model training and optimization workflow still requires manual tuning (hyper-parameters, quantization) and mobile-specific adaptation (G PU delegate compatibility).
Current implementation recognizes only one hand (dominant hand) per frame; multi-hand, bimanual gesture recognition remains to be addressed.

6.3 Practical Implications

For mobile HCI applications—such as gesture-based navigation, AR interactions, accessibility tools and IoT control—this pipeline demonstrates feasibility.

Developers can adopt the workflow: capture data, use MediaPipe for landmark extraction, train in Keras, convert to TFLite and integrate into Android app. The responsiveness (~20 ms) supports fluid interaction. However, designers must still account for environmental variability (lighting, background), and implement fallback/confirmation mechanisms (e.g., multiple-frame consensus) when recognition confidence is low.

7. Conclusion

This research presents a comprehensive mobile- oriented framework for real-time hand gesture recognition, combining MediaPipe for hand-landmark extraction, Keras/TensorFlow for model training, TensorFlow Lite for on-device inference, and OpenCV/NumPy for vision preprocessing. The system achieves high accuracy for static gestures (~96.8 %) and respectable performance for dynamic gestures (~91.4 %) in real-world mobile scenarios, with low latency (~20 ms) and manageable resource usage. This demonstrates the viability of deploying gesture recognition systems on commodity mobile devices, opening pathways for natural user interfaces, accessibility applications and IoT control.

8. Recommendations / Future Scope

Expand gesture vocabulary and datasets: Incorporate a larger set of dynamic gestures, non-dominant-hand gestures and bimanual motions, across more subjects and environments. Improve robustness under adversarial conditions: Conduct further research on lo w-light, background clutter, occlusion, multi-hand interactions and device motion compensation.
Multi-hand and bimanual recognition: Extend the pipeline to 5optimist simultaneous gestures from two hands, including cooperative gestures (e.g., pinch-and- zo om, hand-shake). Lightweight dynamic-gesture models: Investigate more advanced architectures (e.g., Transformer, temporal CNN) 5optimist for mobile (e.g., model compression, knowledge-distillation) to raise dynamic-gesture accuracy. Adaptive on-device l earning: Implement mechanisms for user-specific calibration/fine-tuning on mobile (e.g., few-shot custom gesture addition) without cloud connectivity.

Integration with other modalities: Combine hand- gesture recognition with face expression, body-pose, e ye-gaze or voice inputs to enable richer multimodal interaction. Energy and thermal 5optimization: Further profiling of mobile device thermal behavior, battery drain, and user experience under prolonged use.
Broader deployment platforms: Port the system m to iOS, wearables (smart-watches, AR-glasses) and embedded edge devices (Raspberry Pi, microcontrollers with TFLite Micro). Accessibility and sign-language translation: Leverage the system for real-world assistive applications (e.g., sign-language re cognition / translation, gesture-based interaction for mobility- impaired users) with real-user studies and human- factors evaluation.

Summary: In summary, this paper describes a full- chain mobile hand-gesture recognition system using MediaPipe, Keras/TensorFlow and TensorFlow Lite, achieving high accuracy and low latency on Android devices. It demonstrates the practical viability of gesture-based interaction for mobile HCI and identifies several avenues for further enhancement, particularly in dynamic gesture recognition, multi-hand support and real-world robustness.

References:

[1] Alashhab, S., Gallego, A. J., & Lozano, M. Á. (2022). Efficient Gesture Recognition for the Assistance of Visually Impaired People using Multi-Head Neural Networks. arXiv preprint arXiv:2205.06980.

[2] Benítez-García, G., Olivares-Mercado, J., Sánchez- Pérez, G., & Yanai, K. (2020). IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition. arXiv preprint arXiv:2005.02134.

[3] Darmawan, I. et al. (2025). Hand Gesture Recognition for Collaborative Robots Using TensorFlow Lite. arXiv preprint arXiv:2507.10055.

[4] Google AI Edge. (2023). Hand landmarks detection guide. Retrieved from https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker

[5] InfluxData. (2023). TensorFlow Lite Tutorial: How to Get Up and Running. Retrieved from https://influxdata.com/blog/tensorflow-lite/

[6] Kumar, R., Bajpai, A., & Sinha, A. (2023). Mediapipe and CNNs for Real-Time ASL Gesture Recognition. arXiv preprint arXiv:2305.05296.

[7] MDPI Electronics. (2024). MediaPipe, Inception-v3 and LSTM-Based Enhanced Deep Hand Gesture Recognition for Drone Control. Electronics, 13(16), 3233.

[8] Mishra, O., Suryawanshi, P., Singh, Y., & Deokar, S. (2023). A MediaPipe-Based Hand Gesture Recognition Home Automation System. In 2023 2nd International Conference on Futuristic Technologies (INCOFT) (pp. …). IEEE.

[9] Tang, et al. (2019). SmileAR: iQIYI's Mobile AR Solution based on TensorFlow Lite. TensorFlow Blog.

(blog.tensorflow.org)

[10] Viso.ai. (2023). TensorFlow Lite – Real-Time Computer Vision on Edge Devices. Retrieved from https://viso.ai/edge-ai/tensorflow-lite/

[11] Warchocki, J., Vlasenko, M., & Eisma, Y. B. (2023). GRLib: An Open-Source Hand Gesture Detection and Recognition Python Library. arXiv preprint arXiv:2310.14919.