Technische Hochschule Deggendorf

Faculty of Applied Computer Science

Course Project

Bachelor Artificial Intelligence

Gesture Classification for Microcontrollers
Using TensorFlow Lite on ESP32

Course Project submitted for the degree of

Bachelor of Artificial Intelligence

Authors

Azamat Galidenov
Matriculation Number: 22211179

Mikhail Safronov
Matriculation Number: 22305205

Examiners

Prof. Andreas Federl
Prof. Sergej Lamert

Submission date
18 January 2026

Contents

Abstract 3

1 Introduction 4
1.1 Motivation 4
1.2 Objectives 4
1.3 Application Areas 4

2 System Overview 5
2.1 Hardware Setup 5
2.2 Software Components 5

3 Machine Learning Model 6
3.1 Model Design 6
3.1.1 Architectural Stages 6
3.2 Optimization for Microcontrollers 6
3.3 Optimization for Model 6
3.4 Classification Task 7

4 Workflow and Implementation 8
4.1 Data Collection 8
4.2 Model Training 8
4.3 Model Conversion 8
4.4 Deployment 9

5 Project Structure 10
5.1 Directory Layout 10
5.2 Model Artifacts 10

6 Results 11
6.1 Performance Evaluation 11
6.2 Embedded Execution 12

7 Conclusion 13
7.1 Future Work 13

Abstract

This project focuses on the development of a hand gesture classification system designed for embedded devices. The target platform is the ESP32 microcontroller, and the machine learning inference is performed using TensorFlow Lite for Microcontrollers. The system recognizes three gestures: rock, paper, and scissors.

A complete workflow is implemented, starting from image acquisition with an ESP32 camera module, continuing with model training on a host machine, and ending with deployment of an optimized model on the microcontroller. To enable efficient inference on resource-constrained hardware, the trained neural network is converted through several intermediate formats and finally quantized for TensorFlow Lite.

The project demonstrates that modern deep learning techniques can be successfully applied to embedded systems and provides a structured pipeline that can be reused for similar microcontroller-based machine learning applications.

1 Introduction

1.1 Motivation

Gesture-based interaction is an intuitive way for humans to communicate with machines. Implementing gesture recognition directly on microcontrollers is challenging due to limited memory, processing power, and energy constraints. This project addresses these challenges by designing and deploying an efficient gesture classification system on the ESP32 platform.

1.2 Objectives

The main objectives of the project are:

- Design a gesture classification system for embedded hardware
- Collect and organize gesture image data using an ESP32 camera
- Train a neural network capable of recognizing three hand gestures
- Convert and optimize the trained model for microcontroller deployment
- Demonstrate real-time inference on the ESP32 device

1.3 Application Areas

Possible application domains include interactive systems, human–computer interfaces, embedded control systems, and gesture-driven user input for smart devices.

2 System Overview

2.1 Hardware Setup

The hardware configuration of the system consists of the following components:

- ESP32 development board with integrated wireless connectivity
- Camera module ESP32-CAM
- USB connection for power supply and programming
- Wireless network connection for data transfer

2.2 Software Components

The software stack used in this project includes:

- PyTorch for training the neural network model [4]
- Python scripts for data handling and model conversion
- ONNX as an intermediate model exchange format [5]
- TensorFlow and TensorFlow Lite for model optimization [6]
- Arduino environment and ESP32 Arduino Core for deployment [2]

3 Machine Learning Model

3.1 Model Design

The model utilizes a custom **TinyCNN** architecture designed for efficiency. CNNs are utilized because they can automatically learn spatial hierarchies of features from gesture images.

3.1.1 Architectural Stages

The model is divided into two distinct functional sections:

1. **Feature Extraction (Blocks 1-3):** These layers use convolutional kernels (3×3) and ReLU activations to identify edges, textures, and shapes. After each convolution, a MaxPool layer downsamples the image size by half. This reduces the total number of parameters, which is vital for maintaining high inference speeds on the ESP32.
2. **Classification (The Brain):** After extraction, a `Flatten` layer converts the 3D feature maps into a 1D array of 4,608 values. Two fully connected (Linear) layers then process these values to output a probability distribution across the three gesture classes.

3.2 Optimization for Microcontrollers

The specific depth of the layers (8, 16, and 32 filters) was chosen as a trade-off between accuracy and memory footprint. A larger model would provide diminishing returns in accuracy while potentially exceeding the ESP32's available RAM. By keeping the final feature map size to 12×12 before flattening, we ensure that the densest part of the network (the linear layers) remains within manageable memory limits.

3.3 Optimization for Model

The training process utilized the **Adam (Adaptive Moment Estimation)** optimizer with a learning rate of 0.001. Adam was selected over traditional Stochastic Gradient Descent (SGD) due to its superior performance in training deep learning models for computer vision. The choice of Adam is critical for the development of our TinyCNN for several technical reasons:

3.4 Classification Task

The output of the model consists of three classes corresponding to the gestures rock, paper, and scissors. The model produces a probability distribution over these classes, and the gesture with the highest confidence is selected as the prediction.

4 Workflow and Implementation

4.1 Data Collection

Training data is collected using an ESP32 camera web server script, which leverages the standard *CameraWebServer* library [1]. Images are captured for each gesture and stored in separate directories according to their class label. Using a web server allows for rapid, wireless data acquisition from a remote computer, ensuring that a large volume of training samples can be gathered without manual physical transfers from the SD card or serial buffer.

4.2 Model Training

Model training is performed in a Jupyter notebook environment using PyTorch. The training process includes data loading, preprocessing, and validation. After training, the model is saved in PyTorch format for further conversion. The model was trained over 10 epochs. The line-graph shows an increase in overall Accuracy and a decrease in Training and Validation Loss, which proves that the model converged correctly and overfitting was prevented through the use of a distinct validation set.

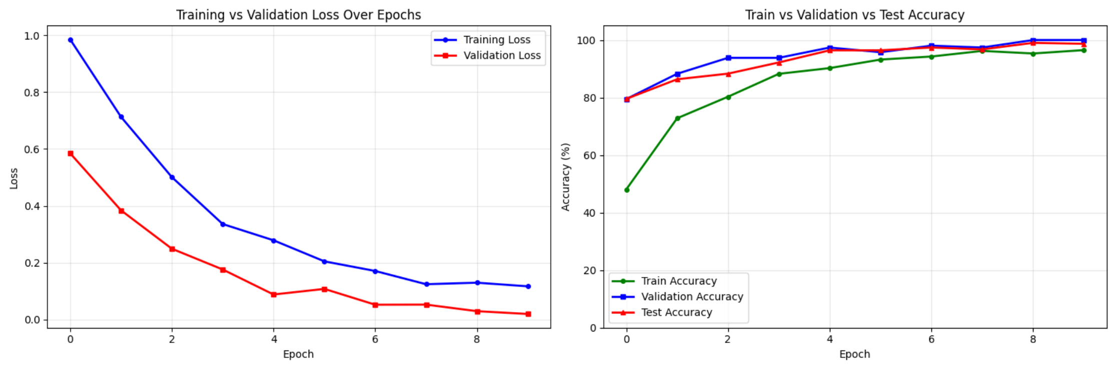


Figure 4.1: Training and Validation Loss/Accuracy over epochs.

4.3 Model Conversion

To enable deployment on the ESP32, the trained model undergoes several conversion steps:

- Conversion from PyTorch to ONNX format (to bridge different ML frameworks).
- Transformation from ONNX to TensorFlow SavedModel.
- Preparation of a float32 TensorFlow model.
- **Quantization** and conversion to TensorFlow Lite format.

Quantization significantly reduces the model size (typically by 4x) and improves inference speed by converting 32-bit floating-point weights into 8-bit integers. This is critical for micro controllers like the ESP32 that lack dedicated high-speed floating-point units.

4.4 Deployment

The final TensorFlow Lite model is converted into a C header file (byte array) and integrated into an Arduino project. The deployment utilizes standard Arduino elements like `setup()` for initialization and an HTTP server for event-driven processing [3].

5 Project Structure

5.1 Directory Layout

The project is organized into scripts and directories corresponding to each development stage:

- Data collection scripts (Python and Arduino)
- Model training notebook (.ipynb)
- Model conversion utilities
- Training data directories
- Trained models in different formats (.pth, .onnx, .tflite)
- Arduino source files (.ino)

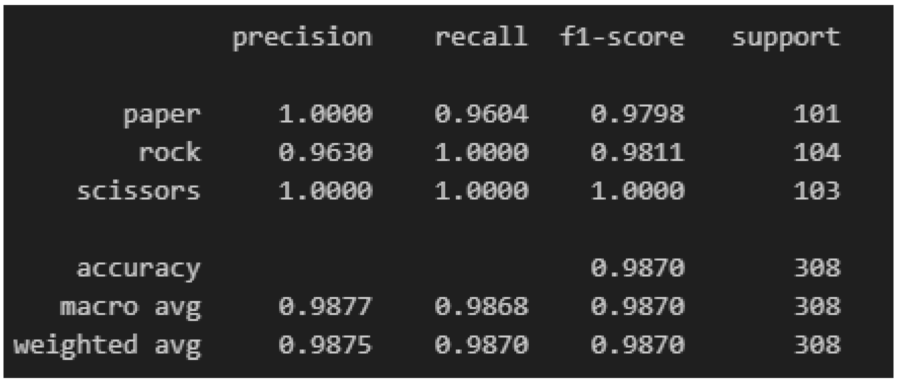
5.2 Model Artifacts

Different versions of the trained model are stored, including the original PyTorch model, intermediate ONNX and TensorFlow formats, and the final quantized TensorFlow Lite model used for deployment.

6 Results

6.1 Performance Evaluation

The final model achieves an accuracy of 98.7% on the test set. Below is the detailed classification report and the confusion matrix showing the model's performance on individual classes.



```
              precision    recall  f1-score   support

       paper     1.0000    0.9604    0.9798       101
        rock     0.9630    1.0000    0.9811       104
    scissors     1.0000    1.0000    1.0000       103

    accuracy                         0.9870       308
   macro avg     0.9877    0.9868    0.9870       308
weighted avg     0.9875    0.9870    0.9870       308
```

Figure 6.1: Classification report of Test Dataset.

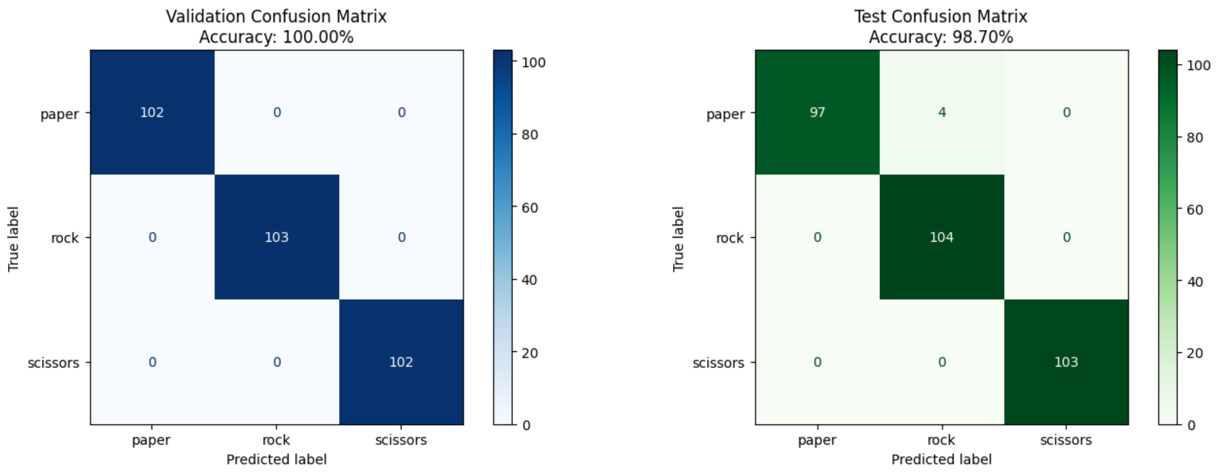


Figure 6.2: Confusion Matrix for Gesture Classification.

6.2 Embedded Execution

Real-time inference on the ESP32 demonstrates that gesture recognition is feasible on resource-constrained hardware while maintaining acceptable responsiveness and efficiency.

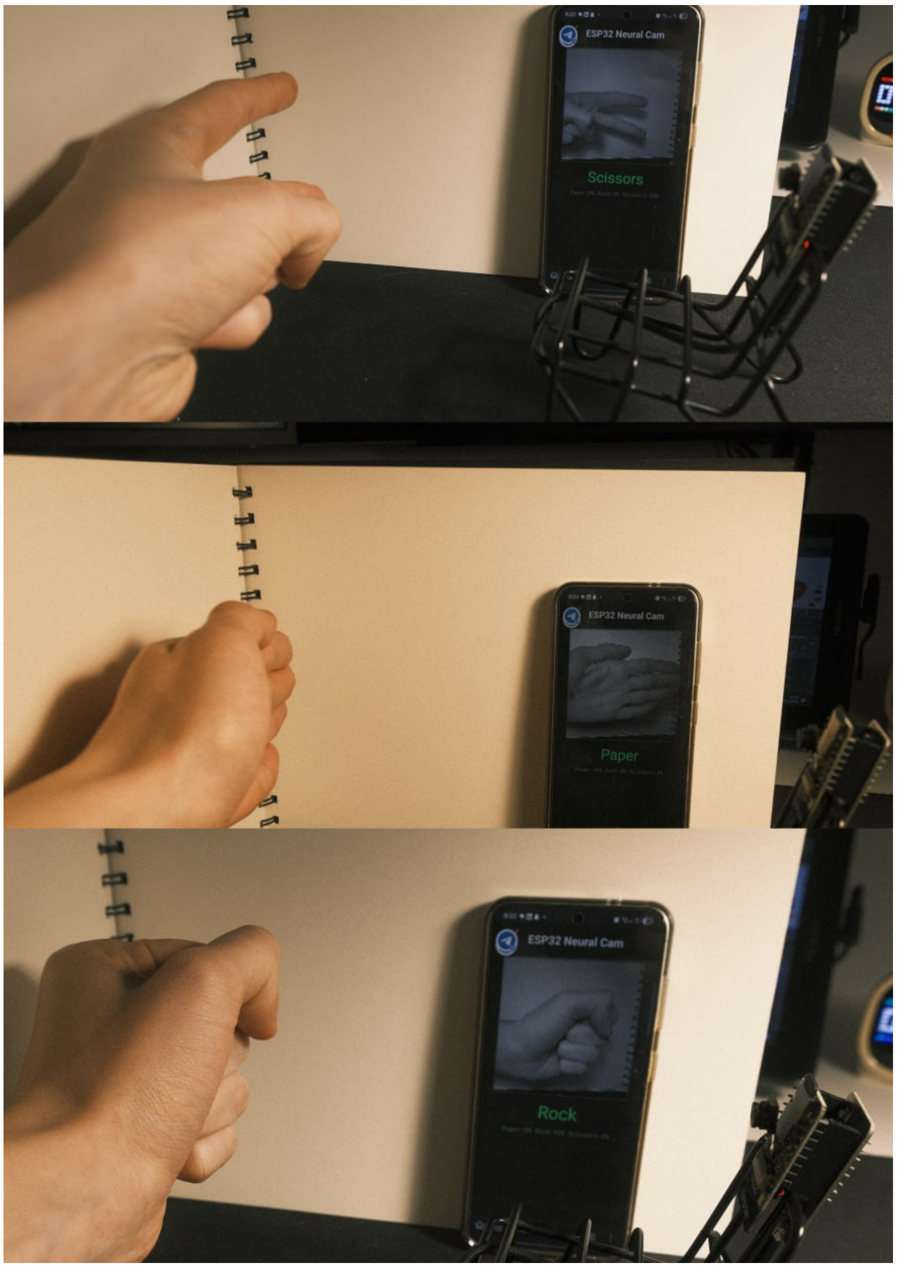


Figure 6.3: Real-time prediction using the ESP32 Web Server interface.

7 Conclusion

This project demonstrates a complete end-to-end pipeline for gesture classification on microcontrollers. By combining data collection, neural network training, and model optimization, a practical embedded machine learning system is realized on the ESP32 platform.

The modular structure of the workflow allows the approach to be extended to additional gestures or adapted to other embedded devices.

7.1 Future Work

Possible extensions of this project include increasing the number of recognizable gestures, processing continuous video streams, and integrating gesture recognition into more complex embedded or robotic systems.

Bibliography

[1] Espressif Systems, *ESP32-CAM Camera Web Server Example*, Available at: `https://github.com/espressif/arduino-esp32/tree/master/libraries/ESP32/examples/Camera/CameraWebServer`.

[2] Espressif Systems, *Arduino core for the ESP32*, Available at: `https://github.com/espressif/arduino-esp32`.

[3] Arduino LLC, *Arduino Language Reference*, Available at: `https://docs.arduino.cc/language-reference/`.

[4] Paszke, A., et al. (2019), *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, `https://pytorch.org/`.

[5] Linux Foundation, *Open Neural Network Exchange (ONNX)*, `https://onnx.ai/`.

[6] Google, *TensorFlow Lite for Microcontrollers*, `https://www.tensorflow.org/lite/microcontrollers`.