EC 291: EXPLORATORY PROJECT - 2025

Edge Based Sign Language Interpreter using ESP-32 CAM

Group Members

Amay Vikram Singh	23095011
Laxmi Tudu	23095053
Pentyala Abhishek Preetham	23095073
Vedansh Nagori	23095114

Supervisor

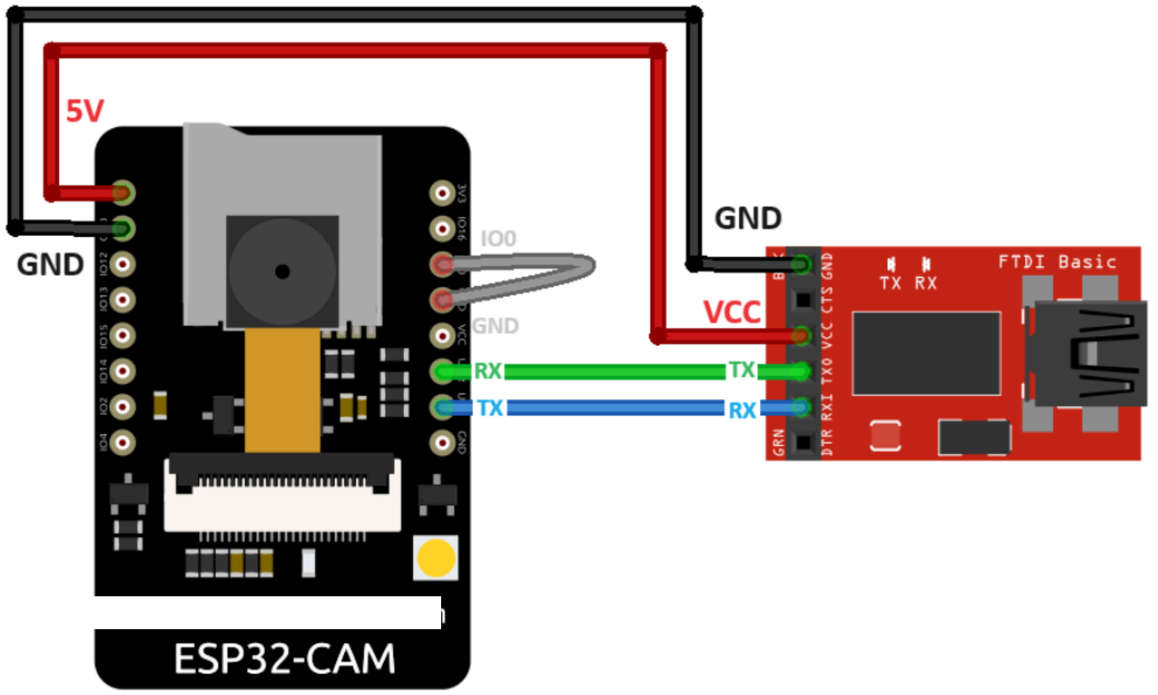
Dr. Manoj Kumar Singh

Department of Electronics Engineering, IIT (BHU) Varanasi, Varanasi-221005

HARDWARE USED

- **Processor:** ESP32-D0WD with dual-core architec ture– Memory: 520KB internal SRAM + 4MB PSRAM, providing sufficient buffer for image processing
- **Camera:** RHYX-M21-45 2-megapixel sensor capable of UXGA resolution (1600×1200)
- **Connectivity:** Built-in 802.11 b/g/n/e/i WiFi for transmitting results to web interface
- **GPIO:** 9 customizable I/O ports for potential expansion
- **Power consumption:** Relatively low power requirements (180mA@5V with flash off)

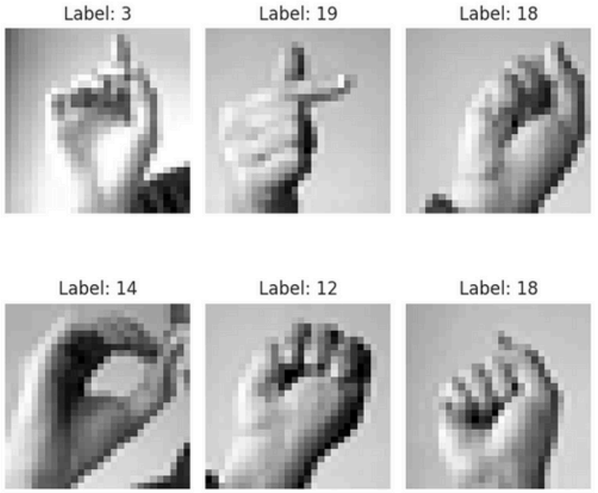
CIRCUITRY



METHODOLOGY

DATASET COLLECTION

The system was trained on the Sign Language MNIST dataset, containing 27,455 training and 7,172 test images of 24 ASL letters (excluding J and Z). Each 28×28 grayscale image represents a static hand gesture, ideal for training a classification model for ASL.



METHODOLOGY

CNN ARTICHECTURE

Layer	Configuration	Function
Input	28×28×1	Accepts grayscale hand gesture images
Conv2D	65 filters, 3×3, ReLU	Extracts low-level features
MaxPool2D	2×2, stride 2	Reduces spatial dimensions
Conv2D	40 filters, 3×3, ReLU	Extracts mid-level features
Dropout	Rate: 0.2	Prevents overfitting
MaxPool2D	2×2, stride 2	Further dimension reduction
Conv2D	25 filters, 3×3, ReLU	Extracts high-level features
MaxPool2D	2×2, stride 2	Final dimension reduction
Flatten	-	Converts 2D features to 1D vector
Dense	256 units, ReLU	Learns complex feature combinations
Dropout	Rate: 0.3	Prevents overfitting
Dense	24 units, Softmax	Outputs class probabilities

EDGE DEPLOYMENT

- The trained model was converted to TensorFlow Lite format to reduce size.
- Quantization to 8-bit integers reduced memory usage by ~75%.
- A lightweight CNN with decreasing filter counts (65 → 40 → 25) was used for efficiency.
- The optimized model was converted into a C array for ESP32 firmware integration.
- Memory allocation was optimized to fit within ESP32-CAM's limited RAM.
- Final model size was reduced from 554KB to 138KB with no significant loss in accuracy.

METHODOLOGY

TENSORFLOW IMPLEMENTATION ON ESP32-CAM

- Library Install: Use Arduino IDE to install TensorFlow Lite for ESP32.
- Model Integration: Convert and embed the quantized model as a C array in the firmware.
- Runtime Setup: Implement error handling, model mapping, operation registration, memory allocation, and tensor buffers.
- Image Processing: Capture, preprocess (grayscale, resize, normalize), and feed the image to TensorFlow Lite for ASL inference. Output the detected ASL letter.

WiFi Integration for Output Display

1. **WiFi Integration:** Configured ESP32-CAM in Station mode (WIFI_STA) to connect to an existing WiFi network.
2. **Web Server:** Created a simple HTML interface to display the live camera feed and detected ASL letters.
3. **Connection Process:** Connected to the WiFi, obtained an IP address, and accessed the system from any device on the same network.
4. **Real-Time Updates:** Displayed live photos alongside real-time ASL detection results.

Advantages: Multiple devices can access the system, no client-side configuration needed, integrates easily into existing networks. The ESP32-CAM functions as a standalone edge device, handling everything from image capture to result communication.

TRAINING PERFORMANCE

The model used the Adam optimizer and categorical cross-entropy loss function. It was trained for 25 epochs on the Sign Language MNIST dataset. The model achieved high accuracy on both training and validation sets. This demonstrates its effectiveness in recognizing ASL letters. The training process validated its strong performance in ASL letter recognition.

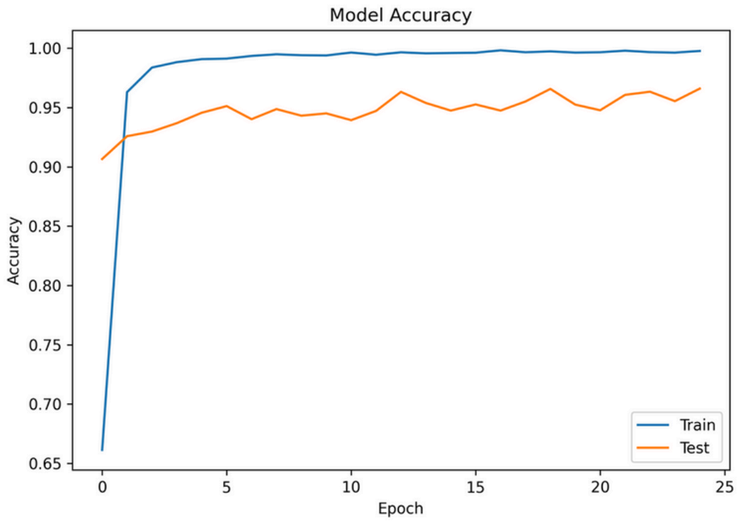


Fig. 2. Training and Validation Accuracy over Epochs

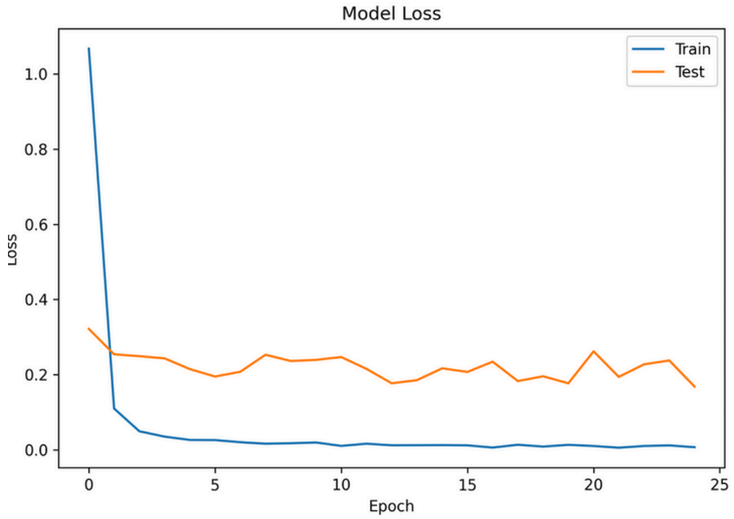
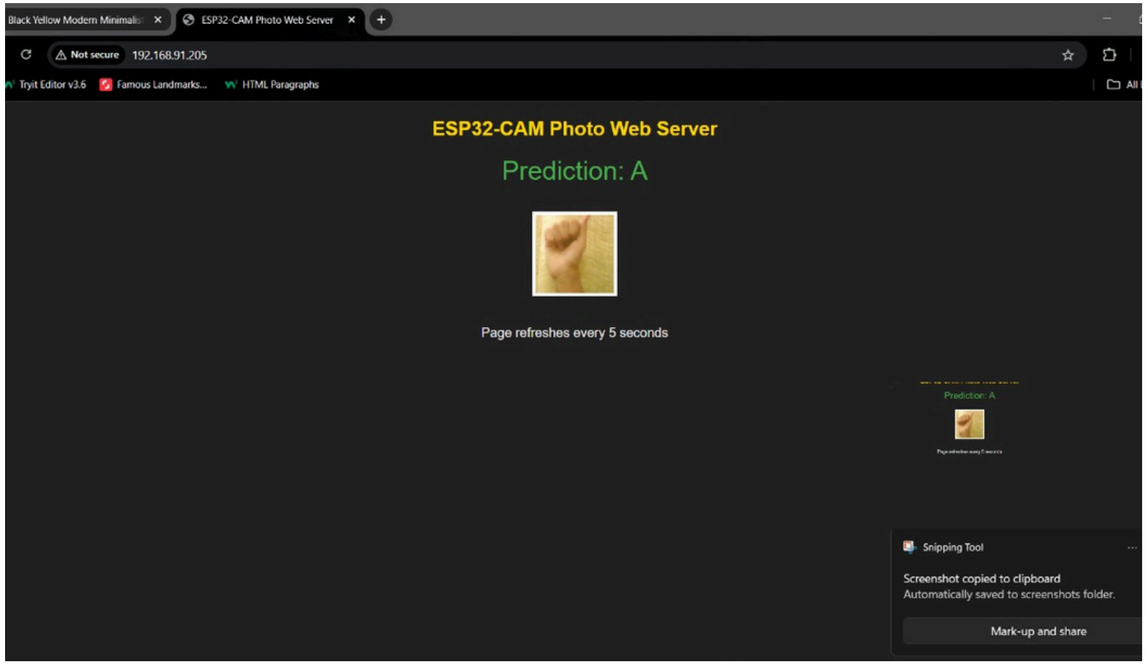
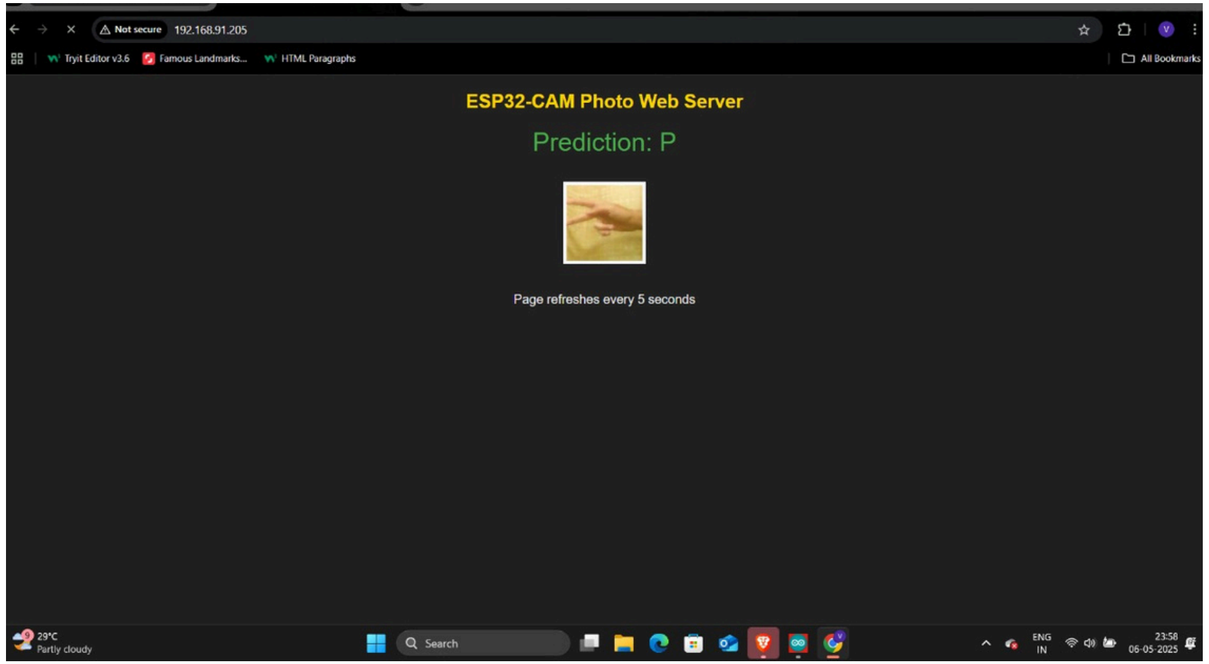


Fig. 3. Training and Validation Loss over Epochs

IN WORLD RESULTS





The code finally returns an IP Address which upon entering in the web browser, gives us an UI which refreshes a picture for every 5 sec and gives a prediction

THANK YOU