TINYML BASED SIGN LANGUAGE INTERPRETER FOR MEDICAL CONSULTATION

Project Guide

Dr.Ajeesh Ramanujan
Associate Professor College of Engineering Trivandrum

May 8, 2024

Contents

Team

- Anamika S Raj : TVE20CS018
- Nirupama T S : TVE20CS084
- Aiswarya M S : TVE20CS009
- Sneha Sani : TVE20CS108

Introduction

- Mute community communicates via sign language, hindering interactions with those unfamiliar with it, notably in medical settings.
- Create a TinyML model for medical consultations with sign language users, easing communication barriers.
- The TinyML model enhances medical consultations by enabling accurate communication with sign language users, improving accessibility and understanding.

Literature Survey

Description	Gap Identified
A recognition framework using VTN achieving improved accuracy and recognition speed compared to I3D	Limitations of the model in recognizing similar sign language actions and the need to improve the extraction of long-sequence keyframes
TinyML-based sign language recognition system utilizing CNN architecture,achieving 94.2% accuracy on an ASL gesture dataset	The system used a small dataset and is only evaluated in a laboratory setting
Low-cost MobileNet solution for recognizing ASL alphabet gestures.The system boasts an impressive 99.93% accuracy rate.	Due to its small size architecture and depthwise convolution, MobileNet cannot perform with high accuracy for complex models.
TinyML implementation on smart gloves for real-time sign language recognition. It examine various algorithms and hardware solutions for sign language translation	Currently they predict gestures , alphabets and numbers only. They dont predict sentences.

Literature Survey

Description	Gap Identified
Uses MobileNet, a deep learning technique, for sign language classification, comparing its accuracy and performance to previous methods like VGG16	Limited discussion on MobileNet's scalability and generalizability in real world.
Design and implementation of an Arduino-based sign language interpreter using cloth-driven gloves with Bluetooth technology	The system was tested only for finger spelling recognition.The paper also does not provide information on the cost of the system or its scalability
The paper describes the Cross Layer Design Flow of the Tiny-ML model along with various hardware and software optimizations carried out to deploy the models into resource-constrained IOT devices	This paper is just a theoretical paper that discusses the existing techniques in Tiny-ML and how they are implemented
Heart-Speaker is a low-cost sign language recognition system that facilitates effective two-way communication achieving a recognition accuracy of 90.77%	The model is trained for the Chinese sign language dataset and is not implemented yet

Motivation

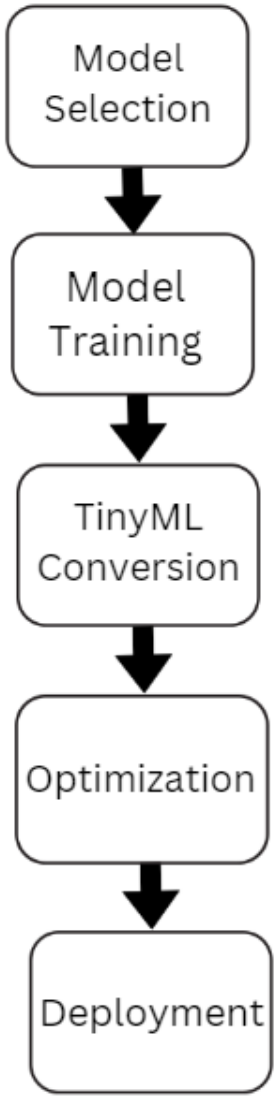
- Comprehensive Communication: Two-way communication system for both mute and hard-of-hearing patients.
- Deployment on Edge Devices: Real-time processing and privacy preservation with Tiny-ML on edge devices.
- Ease of Use: Lightweight, low-power, and user-friendly system.
- Real-world Testing: Evaluation in a medical consultation setting.
- Scalability: Aims for scalability and cost-effectiveness for widespread adoption.

Problem Statement & Objectives

- To develop a lightweight, low-power, easy-to-use system that allows healthcare providers to communicate with mute or hard-of-hearing patients.

- Develop a machine learning model that can accurately recognize sign language gestures
- Translate sign language gestures into text
- Deploy the developed ML model into an edge device using Tiny-ML
- Develop a speech-to-text module to convert a doctor's speech into text enabling two-way communication
- Test and evaluate the system in a medical consultation setting

Design and Algorithms



Design and Algorithms

- Dataset
  - The dataset used is the American Sign Language Dataset.
  - The training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING.
  - These 3 classes are very helpful in real-time applications, and classification.

- Model Selection and Evaluation
  - Four different ML models were evaluated,YOLO V5, Inception,CNN and MobileNet.
  - Each model's accuracy and inference time in recognizing gestures was evaluated.
  - Relatively higher accuracy and acceptable inference speed were obtained for the MobileNet model.
  - MobileNet also showed favorable resource requirements, making it suitable for deployment on Raspberry Pi.

Design and Algorithms

- Model size:37.4737 MB

- Tiny-ML Conversion
  - TinyML is chosen for its real-time, low-latency, and privacy-enhanced machine learning on edge devices, ensuring cost-effective and offline performance.
  - We've selected TensorFlow Lite for our TinyML model. It's a specialized version of TensorFlow for running lightweight machine-learning models on small, low-power devices.

Design and Algorithms

- Optimization Techniques
  - Knowledge Distillation
    - Knowledge distillation converts a large "teacher" model into a smaller "student" model for edge devices.
    - By using Knowledge distillation we got a student model of size:12.6MB
  - Quantization
    - Quantization reduces the bit precision of numerical data in machine learning models.
    - We used dynamic range quantization and converted 32 bit float numerical data to 8-bit.
  - TFlite Conversion
    - TensorFlow Lite conversion transformed the optimized MobileNet model into a format compatible with TensorFlow Lite.

Design and Algorithms

- After doing all these optimization techniques our model size was reduced to 3.26 MB (from 37.47 MB)

- Deployment
  - Device specification: Raspberry Pi 4 Model B with 1 GB RAM
  - The device should support Python, TensorFlow Lite, Opencv and Numpy.

Implementation Setup Framework

Hardware and Software Requirements

- Hardware Requirements: Raspberry Pi 4 Model B with 1 GB RAM,microSD card (>8GB), Webcam, Monitor,
- Software Requirements: Raspberry Pi OS 64bit, Python (version 3.6 or later), TensorFlow Lite, OpenCV, NumPy

Implementation Environment Setup

- Install Raspberry Pi Imager
- Install Raspberry Pi OS 64bit
- Install TensorFlowLite
- Install Numpy
- Install Opencv

Implementation Results

- Performance Metrics of Trained Models

Model	Parameters	Accuracy	Inference time (s)
Inception	22 Million	94%	0.173
YOLO V5	7.08 Million	92%	0.93
CNN	5.59 Million	94%	0.134
MobileNet	3.25 Million	99.04%	0.1400

- Optimization techniques-Result

Model	Size	Accuracy	Inference time
Mobilenet	37.47 MB	99%	0.14s
Knowledge distillation	12.60 MB	95%	0.12s
Quantization	3.25 MB	91%	0.02s

Implementation Results

- Dataset Split

```python
# Dimensions of the data
print("Training data shape:", X_train.shape)
print("Validation data shape:", X_val.shape)
print("Test data shape:", X_test.shape)

# Dimensions of the labels
print("Training labels shape:", y_train.shape)
print("Validation labels shape:", y_val.shape)
print("Test labels shape:", y_test.shape)
```

```
Training data shape: (30450, 64, 64, 3)
Validation data shape: (6525, 64, 64, 3)
Test data shape: (6525, 64, 64, 3)
Training labels shape: (30450, 29)
Validation labels shape: (6525, 29)
Test labels shape: (6525, 29)
```

Figure 1: Dataset Shape

Implementation Results

- MobileNet-Architecture

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functi  (None, 2, 2, 1024)        3228864
 onal)

 global_max_pooling2d (Glob  (None, 1024)              0
 alMaxPooling2D)

 dense (Dense)               (None, 29)                29725

=================================================================
Total params: 3258589 (12.43 MB)
Trainable params: 3236701 (12.35 MB)
Non-trainable params: 21888 (85.50 KB)
```

Figure 2: Architecture

Implementation Results

- MobileNet- Training and validation Accuracy

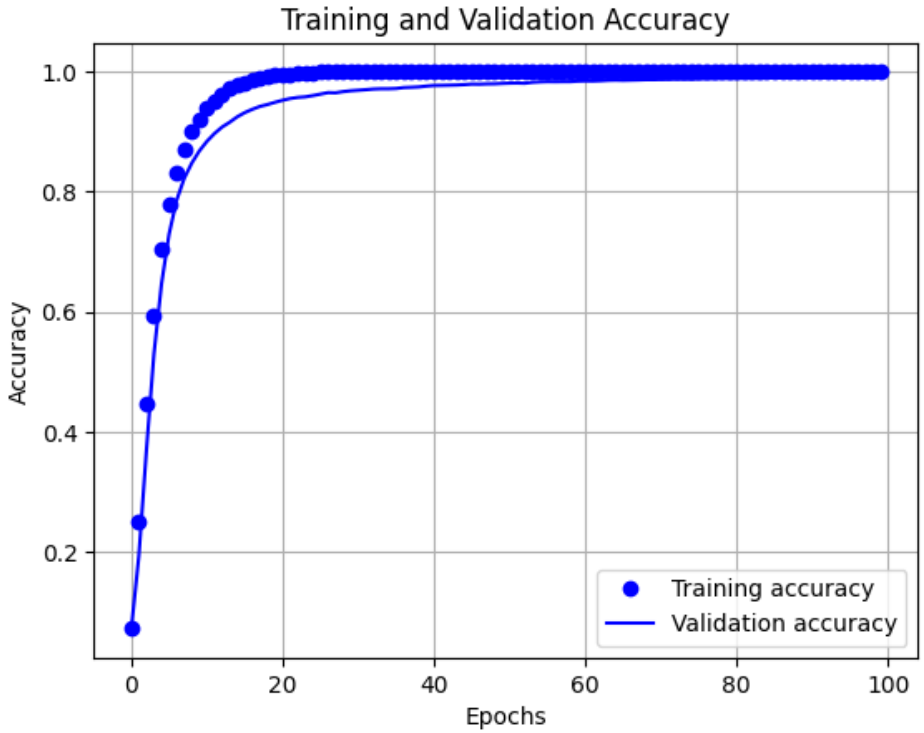


Figure 3: Accuracy vs epoch

Implementation Results

- MobileNet- Training and validation loss

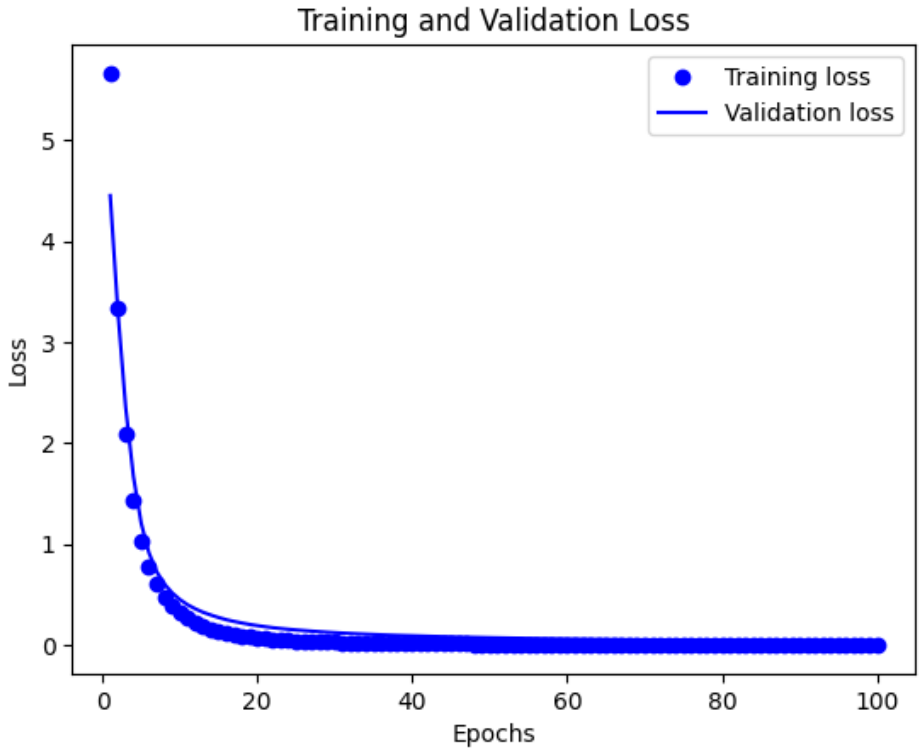


Figure 4: loss vs epoch

Implementation Results

- MobileNet- Classification Report

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	227
1	0.99	0.99	0.99	229
2	1.00	1.00	1.00	206
3	1.00	1.00	1.00	230
4	0.99	1.00	0.99	205
5	1.00	1.00	1.00	210
6	1.00	0.99	1.00	214
7	1.00	1.00	1.00	209
8	1.00	1.00	1.00	242
9	1.00	0.98	0.99	219
10	1.00	1.00	1.00	248
11	0.98	1.00	0.99	241
12	1.00	0.99	0.99	240
13	0.97	0.99	0.98	245
14	0.99	0.99	0.99	235
15	0.99	0.98	0.99	206
16	1.00	1.00	1.00	211
17	1.00	0.99	1.00	220
18	0.99	0.99	0.99	224
19	0.98	0.99	0.98	208
20	1.00	0.98	0.99	221
...				
accuracy			0.99	6525
macro avg	0.99	0.99	0.99	6525
weighted avg	0.99	0.99	0.99	6525

Figure 5: Classification Report

Implementation Results

- Student Model- Architecture

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 60, 60, 32)        2432

 max_pooling2d (MaxPooling2  (None, 30, 30, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 26, 26, 64)        51264

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 flatten (Flatten)           (None, 10816)             0

 dense_1 (Dense)             (None, 29)                313693

=================================================================
Total params: 367389 (1.40 MB)
Trainable params: 367389 (1.40 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 6: Student Architecture

Implementation Results

- Student Model- Training and validation Accuracy

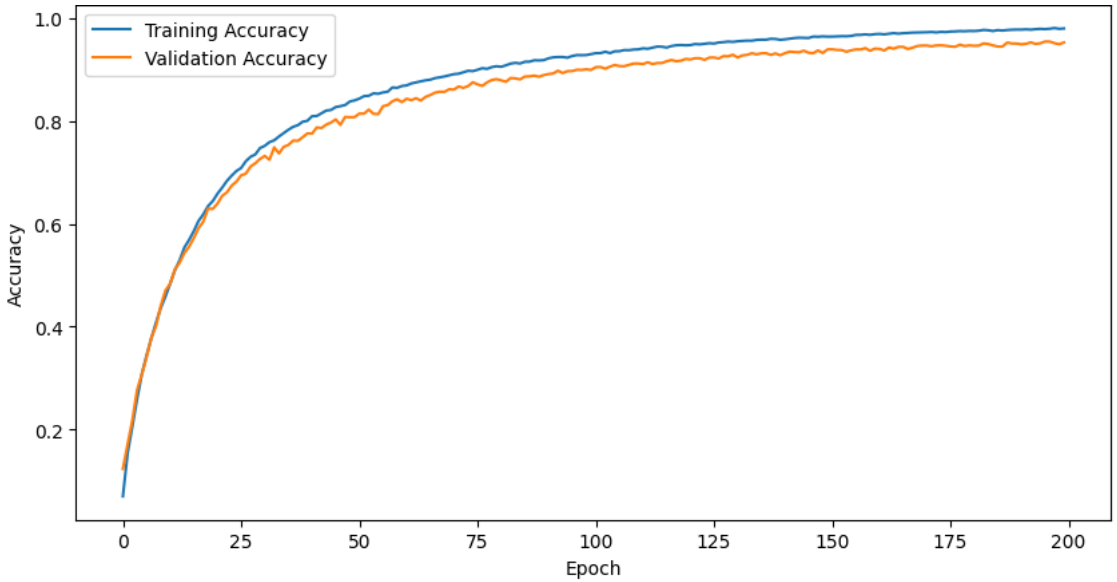


Figure 7: Student Accuracy

Implementation Results

- Quantized tflite model

```
# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Assume your test data and labels are in X_test, y_test (from your original code)

correct_count = 0
total_count = len(X_test)

for i in range(total_count):
  input_data = np.expand_dims(X_test[i], axis=0)  # Adjust if your model expects a different shape
  interpreter.set_tensor(input_details[0]['index'], input_data)
  interpreter.invoke()

  output_data = interpreter.get_tensor(output_details[0]['index'])
  predicted_label = np.argmax(output_data[0])
  true_label = np.argmax(y_test[i])

  if predicted_label == true_label:
    correct_count += 1

accuracy = correct_count / total_count * 100

print("Accuracy of the quantized model:", accuracy, "%")
```

```
Accuracy of the quantized model: 91.77777777777779 %
```

Figure 8: tflite model Accuracy

Implementation Results

- Hardware setup

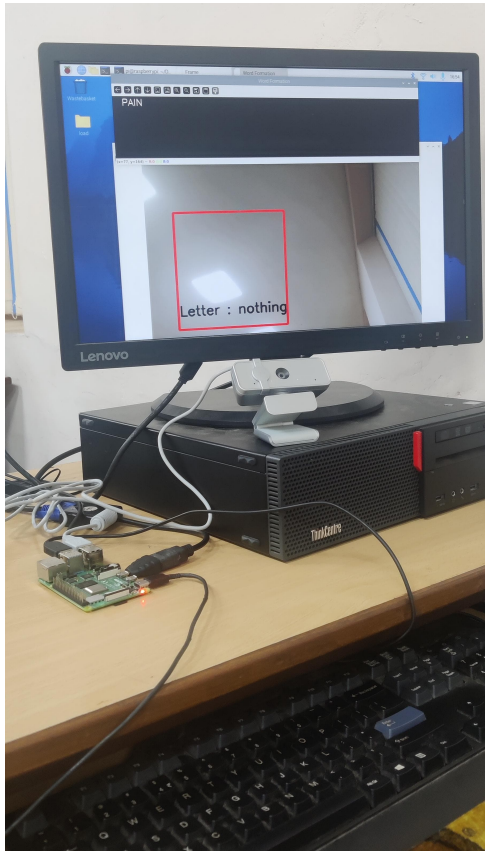


Figure 9: Hardware setup

Implementation Results

Individual Contributions

- Nirupama TS-Literature Survey, YOLO V5 model, Quantization, Hardware Selection, IoT device Integration, Documentation
- Aiswarya MS-Literature Survey, MobileNet model, Quantization, Knowledge distillation, Hardware Selection, Documentation
- Sneha Sani-Literature Survey, Inception model, Knowledge distillation, Hardware Selection, IoT device Integration, Documentation
- Anamika S Raj-Literature Survey, CNN model, Quantization, Hardware Selection, Documentation

Conclusion and Future Works

- Conclusion
  - Initial model size was 37.47MB
  - Our optimization efforts have significantly reduced the size of our MobileNet model, making it suitable for deployment on edge devices.
  - Size of the model after knowledge distillation: 12.60 MB
  - The model size after Quantization and Tensorflowlite Conversion:3.25 MB.
  - Raspberry Pi 4B is selected as an edge device for model deployment
  - Implementation environment is set up in Raspberry Pi and the model is deployed into it.

Conclusion and Future Works

- Future Works
  - Enable two-way communication by incorporating efficient doctor-to-patient communication.
  - Use audio channels instead of text for communication.
  - Community Engagement

References

[1] Yuming Chen Yanyin Yao Shi Hu Wuyang Qin, Xue Mei and Qihang Zhang.
Sign language recognition and translation method based on vtn.

[2] Vijay Janapa Reddy Boris Murmann Mohd.Shafique, Theocharis Theocharides.
Tinyml:current progress,research challenges and future roadmap.

[3] Ritika Basavaraj Hiremath Vanshika Sai Ramadurgam Deepak Kumar Shaw Santhosh Kumar B, Rachna P.
Survey on implementation of tinyml for real-time sign language recognition using samrt gloves.

[4] R Naveen Karthik A Kaliselvi K Anitha, K C Varsheni.
Gesture-based sign language recognition system.

References

[5] Srujal Reddy Gundedi Vishal Satpute Venaktesh Kandukuri, Vipin Kamble.
Deaf and mute sign language translator on static alphabets gestures.

[6] Abhijit Das Anirbit Sengupta, Md. Tausif Mallick.
A cost-effective design and implementation of arduino-based sign language interpreter.

[7] Abu-Jamie Tanseem N and Samy S Abu-Naser Prof.Dr.
Classification of sign language using mobilenet-deep learning.

[8] Hongliang Fan Kun Xia, Weiwei Lu and Qiang Zhao.
A sign language recognition system applied to deaf-mute medical consultation.

Thank You