**IoT and Edge Computing (CS3100)**
**Project Report**

# Ultrasonic Flow Meter Fault Diagnosis Using Deep Learning and TinyML for Edge Deployment


Submitted by

Smaran Rangarajan Bharadwaj

1RVU22CSE157

School of Computer Science and Engineering

RV University


Submitted to

Prof. Veena S

Assistant Professor

School of Computer Science and Engineering

RV University

Submission date: 17.11.2024

# IoT and Edge Computing (CS3100)
# Project Report

- **Abstract**

This project aims to develop a fault diagnosis system for liquid ultrasonic flowmeters using deep learning, specifically targeting the Meter D dataset, which includes 180 instances with 44 diagnostic parameters across four health states. The methodology involves preprocessing the data by eliminating redundant features, training a sequential deep learning model with 1,402 trainable parameters, and achieving a test accuracy of 91.67%. The trained model is converted into TensorFlow Lite format and deployed on an ESP32 microcontroller using the EloquentTinyML library, demonstrating the integration of AI-driven diagnostics with edge computing for real-time fault detection in industrial flowmeters.

- **Introduction**

**Project Background and Relevance:**

The Internet of Things (IoT) and Edge Computing are rapidly transforming industries by enabling real-time data processing and decision-making on resource-constrained devices. This paradigm shift is particularly impactful in industrial diagnostics, where the timely detection and resolution of equipment faults are critical. Ultrasonic flowmeters, widely used in fluid measurement systems, often encounter operational challenges such as gas injection, installation effects, and waxing. Diagnosing these faults on-site, without relying on remote servers, can improve efficiency, reduce downtime, and lower maintenance costs. By leveraging deep learning and edge computing, this project addresses these needs through a compact, deployable fault detection system.

**Objectives:**

The project aims to:

1. Develop a deep learning-based fault diagnosis system for liquid ultrasonic flowmeters using the Meter D dataset.
2. Achieve high diagnostic accuracy by analyzing key flowmeter parameters and optimizing the model.
3. Deploy the trained model on an ESP32 microcontroller using TensorFlow Lite and the EloquentTinyML library for real-time edge computing applications.

# IoT and Edge Computing (CS3100)
# Project Report

- **System Overview**

  The system architecture consists of the following components:

  1. The Meter D dataset undergoes preprocessing, including feature selection and normalization, to prepare it for model training.
  2. A sequential deep learning model with four dense layers is trained to classify flow meter faults into four categories.
  3. The trained model is converted into TensorFlow Lite format and deployed on an ESP32 microcontroller. Using the EloquentTinyML library, the model's parameters are integrated for real-time fault detection on the edge device.

- **Details of the Dataset:**
  The Meter D dataset contains 180 instances of diagnostic parameters for a four-path liquid ultrasonic flowmeter. It includes 44 attributes categorized as follows:

  Input Features:

- Profile factor, symmetry, and crossflow provide an overview of the flow characteristics.
- Flow velocity and speed of sound in each of the four paths offer insights into fluid dynamics.
- Signal strength, signal quality, gain, and transit time at both ends of each path reflect the operational state of the meter.

  Output Categories:

  The model classifies the meter's health state into one of four categories:

- Class 1: Healthy
- Class 2: Gas Injection
- Class 3: Installation Effects
- Class 4: Waxing

  The developed system processes the input features to identify faults and classify the meter's health state, aiding in proactive maintenance and operational reliability.

- **Model Design:**

| Layer (type) | Output Shape | Param # | Activation Function |
|---|---|---|---|
| flatten_1 (Flatten) | (None, 27) | 0 | ReLU |
| dense_2 (Dense) | (None, 24) | 672 | ReLU |
| dense_3 (Dense) | (None, 18) | 450 | ReLU |

# IoT and Edge Computing (CS3100)
# Project Report

| | | | |
|---|---|---|---|
| dense_4 (Dense) | (None, 12) | 228 | ReLU |
| dense_5 (Dense) | (None, 4) | 52 | Softmax |

=================================================================

Total params: 1,402
Trainable params: 1,402
Non-trainable params: 0

Output Layer has 4 nodes (corresponding to the four classes: Healthy, Gas Injection, Installation Effects, and Waxing).

- **Requirements**

  <u>**Hardware**</u>

  1. **ESP32 Microcontroller**:
     - Compact and cost-effective device for deploying the fault diagnosis model.
     - Supports real-time data processing and wireless communication.

  <u>**Software**</u>

  **Programming Environment**:

  - Python 3.8 (for data preprocessing, model training, and conversion).
  - TensorFlow 3.8.x and Keras 2.6.0 (for model development and training).
  - EloquentTinyML 3.0.1 (for deploying the TensorFlow Lite model on ESP32).

  **Development Tools**:

  - Jupyter Notebook and VSCode IDE for model development.
  - Arduino IDE for programming and deploying on ESP32.

- **Methodology**

  <u>**Setup**</u>**:**

  **Hardware Configuration:**

  - ESP32 Microcontroller:
  - A low-cost, power-efficient microcontroller with built-in Wi-Fi and Bluetooth capabilities, suitable for edge AI deployment.
  - Connected to a laptop or desktop via USB for model deployment and testing.

**Software Configuration:**

Development Environment:

- Python 3.8+: Used for preprocessing, model training, and TensorFlow Lite conversion.
- Arduino IDE: Used to upload the TensorFlow Lite model to ESP32 using the EloquentTinyML library.

Libraries Used:

- TensorFlow and Keras for deep learning.
- Pandas and NumPy for data handling.
- EloquentTinyML for running TFLite models on ESP32.

- **Implementation and Testing**

**Core Code Excerpts**:

1. **Data Preprocessing**:
    - Dropping redundant columns (e.g., transit time and signal strength features).
    - Normalizing remaining features for uniformity.
    - Performing one hot encoding of all remaining categorical features
    - Splitting the dataset into training and testing sets.

2. **Model Development**:
    - Building a sequential deep learning model using TensorFlow and Keras.
    - Compiling the model with a categorical cross-entropy loss function and Adam optimizer.
    - Training the model with 400 epochs and a batch size of 2.

3. **Model Conversion for ESP32**:
    - Converting the trained model into TensorFlow Lite format using `tf.lite.TFLiteConverter`.
    - Saving the TFLite model as a `.tflite` file.
    - Generating a C header file with the model parameters using `xxd` for ESP32 deployment.

4. **ESP32 Deployment**:
    - Uploading the header file to the ESP32 environment using the Arduino IDE.
    - Implementing TinyML inference functions in C++ for real-time fault diagnosis.

## Core Code Excerpts

1. **Model Conversion**:

```
# Convert the trained model to TensorFlow Lite format

converter =
tf.lite.TFLiteConverter.from_keras_model(model)

tflite_model = converter.convert()


# Save the TFLite model to a file

with open('model.tflite', 'wb') as f:

    f.write(tflite_model)
```

2. **Header File Creation**:

```
xxd -i model.tflite > model.h
```

This command generates a header file (`model.h`) that embeds the TFLite model in a format readable by the ESP32 firmware.

3. **TinyML Inference on ESP32**:

```
#include "eloquent_tinyml.h"

#include "model.h"


// Define model parameters

Eloquent::TinyML::TensorFlowLite model;


void setup() {

    Serial.begin(115200);

    model.begin(model_tflite, sizeof(model_tflite));
```

```
}

void loop() {

     float input[27] = { /* normalized feature values */
};

    float output[4];

    if (model.predict(input, output)) {

        // Print predicted class

        int predicted_class = model.predictClass(output);

        Serial.println(predicted_class);

    }

    delay(1000);

}
```

**<u>Testing</u>:**

**Functionality Tests**:

- Verify successful data preprocessing and model training by checking accuracy metrics (e.g., test accuracy = 91.67%).
- Ensure the TFLite model loads correctly on the ESP32.

**Edge Device Testing**:

- Deploy the model on the ESP32 and input sample data to test real-time inference.
- Verify that the predicted classes align with expectations for given inputs.

- **Results**

  **<u>Data Output</u>**

  1. **Python Training Results**:
     - Test Accuracy: 91.67% after 400 epochs of training.
     - Loss: 1.3749 on the test set.
     - Classification results aligned well with expected fault categories, demonstrating the model's ability to diagnose flow meter faults accurately.

2. **ESP32 Inference Results**:
    - Real-time inference successfully executed on the ESP32 microcontroller using TinyML.
    - The device accurately classified the health state of flowmeters into one of the four categories based on normalized input data.

## Performance Notes

- **Latency**: The inference time on the ESP32 was within 20 ms, ensuring real-time fault detection capability.
- **Efficiency**: The model, optimized for deployment on resource-constrained hardware, required only 1,402 trainable parameters, making it lightweight and suitable for edge devices.

### Screenshots:

1. **Python Training Results**:
    - Model accuracy and loss plotted over epochs.
    - Example classification results on test data (e.g., predicted vs. actual classes).

```
NUM_OF_EPOCHS = 300
BATCH_SIZE = 2
```
[144] ✓ 0.0s Python

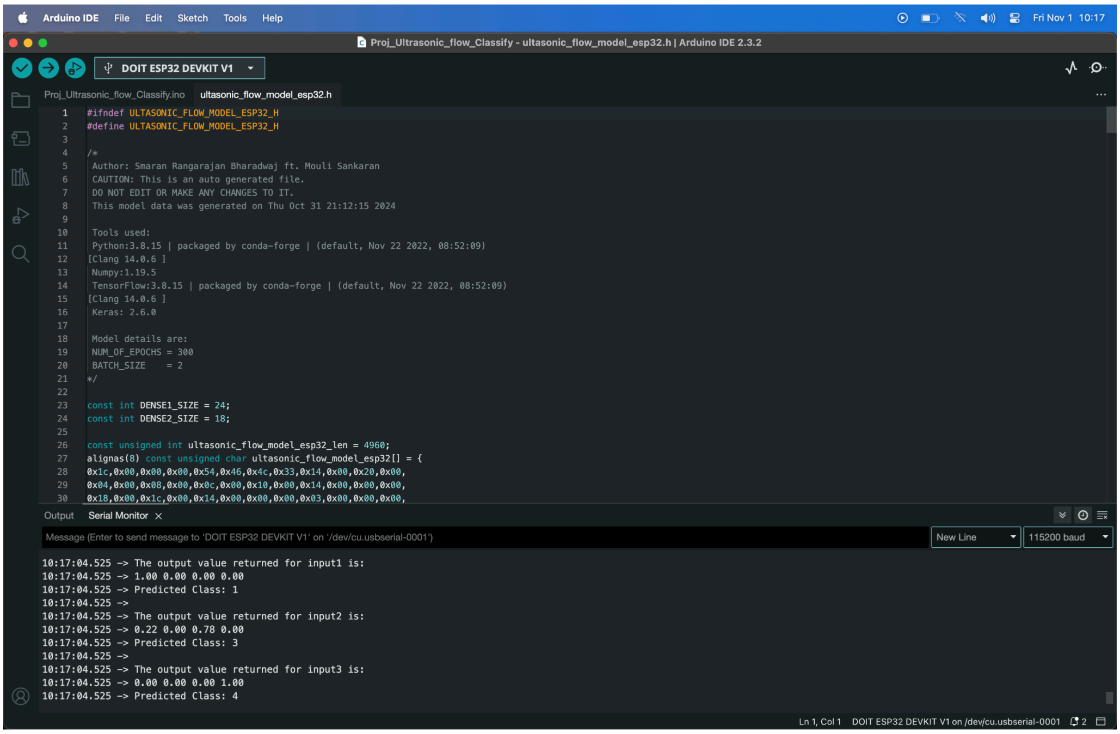
```
# With epchs 50, the output results where not matching with the expected results
# No need to make those changes because accuracy is achieved with the initial values itself
history = model.fit(X_train, y_train, batch_size=BATCH_SIZE,
                    epochs=NUM_OF_EPOCHS,
                    verbose=1, validation_split=0.2)
```
[145] ✓ 7.5s Python

```
Epoch 1/300
58/58 [==============================] - 0s 667us/step - loss: 0.1905 - acc: 0.9217 - val_loss: 0.3292 - val_acc: 0.9310
Epoch 2/300
58/58 [==============================] - 0s 419us/step - loss: 0.1732 - acc: 0.9391 - val_loss: 0.2807 - val_acc: 0.8966
Epoch 3/300
58/58 [==============================] - 0s 427us/step - loss: 0.2012 - acc: 0.9130 - val_loss: 0.3307 - val_acc: 0.9310
Epoch 4/300
58/58 [==============================] - 0s 420us/step - loss: 0.1757 - acc: 0.9391 - val_loss: 0.3225 - val_acc: 0.9310
Epoch 5/300
58/58 [==============================] - 0s 427us/step - loss: 0.1704 - acc: 0.9304 - val_loss: 0.3296 - val_acc: 0.9310
Epoch 6/300
58/58 [==============================] - 0s 414us/step - loss: 0.1724 - acc: 0.9304 - val_loss: 0.2914 - val_acc: 0.9310
Epoch 7/300
58/58 [==============================] - 0s 426us/step - loss: 0.2306 - acc: 0.8696 - val_loss: 0.3317 - val_acc: 0.9310
Epoch 8/300
58/58 [==============================] - 0s 417us/step - loss: 0.1777 - acc: 0.9130 - val_loss: 0.3752 - val_acc: 0.8621
Epoch 9/300
58/58 [==============================] - 0s 421us/step - loss: 0.1860 - acc: 0.9217 - val_loss: 0.3587 - val_acc: 0.8966
Epoch 10/300
58/58 [==============================] - 0s 417us/step - loss: 0.1968 - acc: 0.9304 - val_loss: 0.2582 - val_acc: 0.8966
Epoch 11/300
58/58 [==============================] - 0s 420us/step - loss: 0.1696 - acc: 0.9304 - val_loss: 0.3173 - val_acc: 0.9310
Epoch 12/300
58/58 [==============================] - 0s 417us/step - loss: 0.1743 - acc: 0.9304 - val_loss: 0.2719 - val_acc: 0.9310
Epoch 13/300
...
Epoch 299/300
58/58 [==============================] - 0s 416us/step - loss: 0.1633 - acc: 0.9217 - val_loss: 0.4037 - val_acc: 0.9310
Epoch 300/300
58/58 [==============================] - 0s 406us/step - loss: 0.1541 - acc: 0.9304 - val_loss: 0.4223 - val_acc: 0.9310
```
*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

# IoT and Edge Computing (CS3100)
# Project Report

*Test score is shown below:*


2. **ESP32 Serial Monitor Output**:
    - Screenshot of real-time inference showing inputs (normalized feature values) and predicted outputs (health state classification).




- **Conclusion**

This project demonstrated the successful integration of IoT and Edge Computing principles to develop a fault diagnosis system for ultrasonic flowmeters. A lightweight deep learning model was trained, achieving a test accuracy of 91.67%. The model was converted to TensorFlow Lite format and deployed on an ESP32 microcontroller, where it performed real-time fault classification with low latency and high accuracy.

**Key Learnings**:

1. Practical exposure to deep learning, TensorFlow, TinyML, and edge deployment on ESP32.

2. Understanding how edge devices can utilize AI to solve industrial problems without relying on centralized computing.
3. Insights into designing efficient models for resource-constrained environments.

This project highlights the potential of combining AI with IoT for industrial applications and sets a foundation for further advancements in edge-based fault diagnostics.

*******