Major Project Report

on

TinyML for Predictive Analysis

Submitted by

Aalekh Prasad 20BEC001

Saurabh Mani Tripathi 20BEC047

Vinayak 20BEC057

Vishal Kumar Jha 20BEC059

Under the guidance of

Dr. Prakash Pawar

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DHARWAD

22/03/2024

Certificate

This is to certify that the project, entitled **TinyML for Predictive Analysis**, is a bonafide record of the Mini Project coursework presented by the students whose names are given below during 2024 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Electronics and Communication Engineering.

Roll No	Names of Students
20BEC001	Aalekh Prasad
20BEC047	Saurabh Mani Tripathi
20BEC057	Vinayak
20BEC059	Vishal Kumar Jha

Dr. Prakash Pawar
(Project Supervisor)

Contents

List of Figures

List of Tables

1 Introduction

In today's Industrial world, when efficiency and productivity are crucial, effective monitoring and maintenance of machinery have become essential tasks. Electric motors are key components of industrial machinery, enabling a wide range of equipment and operations in industries such as manufacturing, energy production, transportation, and more. Electric motors are subject to wear and tear over time, which can result in flaws, inefficiencies, or even sudden breakdowns. These difficulties can cause costly downtime, production delays, and higher maintenance costs. Anomaly detection techniques are critical tools in this quest, allowing for the early discovery of deviations from typical behavior.

This project is a complete examination into the use of machine learning algorithms for detecting anomalies in electric motors, focusing primarily on data acquired from a ceiling fan equipped with a 3-axis accelerometer. We aim to create robust models capable of properly identifying anomalies in motor behavior by using two distinct machine learning models for detecting anomalies in an electric motor. The first model is based on the traditional machine learning technique of Mahalanobis distance. The second model is an autoencoder neural network built using TensorFlow and Keras.. The dataset used in this study consists of accelerometer readings taken at various fan speeds and configurations, including the presence of additional weights. Such data diversity enables the development of models capable of managing real-world circumstances in which motor operation can vary in complexity. We believe that this study contributes a lot to the advancement of anomaly detection techniques in industrial settings.

2 Related Work

2.1 Predictive analysis for industrial maintenance automation and optimization using a smart sensor network

The move to world-class manufacturing (WCM) has highlighted the need of proper maintenance of production facilities and systems. This need is especially evident with the implementation of just-in-time (JIT) production and flexible, agile manufacturing methods. To maintain equipment availability, product quality, timely deliveries, and competitive pricing, maintenance management must be integrated into business strategy. The evolving landscape of modern manufacturing necessitates an examination of the role that enhanced maintenance management plays in generating significant cost and service benefits. The authors presented an extensive response to many issues encountered during industrial maintenance.

Their results showed that Maintenance processing is fully automated utilizing an integrated strategy that includes an Online Maintenance Portal, a Real-Time Maintenance Notification System, optimization using a Smart Sensor Network, and powerful Data Analytics and Data Extraction methodologies. The study provides a system that combines modern sensor technologies and predictive analysis algorithms to enable proactive maintenance techniques in industrial environments. The objective of the system is to collect real-time data from machinery and equipment by deploying smart sensors that can monitor characteristics such as temperature, vibration, and energy consumption.

2.2 Predictive Maintenance in the 4th Industrial Revolution: Benefits, Business Opportunities, and Managerial Implications

As Industry 4.0 continues to transform the manufacturing landscape, predictive maintenance remains a critical tool for increasing operational efficiency and decreasing downtime. Predictive maintenance in the context of Industry 4.0 involves employing modern technologies such as IoT sensors, data analytics, machine learning, and artificial intelligence to predict failures in equipment before they occur. Predictive maintenance, which analyzes real-time data from machinery and systems, enables organizations to transition from reactive or scheduled maintenance to a proactive and predictive strategy. In this paper, authors have pointed out that multiple research investigations have examined the technical consequences of using predictive maintenance solutions in Industry 4.0 scenarios. But the business perspective of predictive maintenance often gets ignored.

The authors addressed the benefits, business potential, and managerial consequences of predictive maintenance based on their experience in designing, implementing, and deploying relevant systems. Various business opportunities that include new revenue streams and increased consumer satisfaction, were highlighted. The managerial implications, emphasizing the significance of strong leadership and data governance in successfully integrating predictive maintenance, were also emphasized. Overall, the research tries to bridge the gap between technological improvements and strategic decision-making by focusing on the business implications of predictive maintenance in Industry 4.0 environments.

3 Hardware Used

3.1 MSA311 Accelerometer

The MSA311 Accelerometer is a compact and highly sensitive sensor used for measuring acceleration. It offers a wide measurement range, low power consumption, and both analog and digital output options. Its robust design makes it suitable for various applications such as automotive, aerospace, and consumer electronics. By integrating the MSA311 into our project, we ensure accurate and reliable acceleration measurements, enhancing the system's performance.

3.2 ESP8266 Wi-fi Module

The ESP8266 is a tiny and affordable Wi-Fi module widely used in IoT projects. It lets devices connect to Wi-Fi networks, is easy to program, and has low power consumption. With GPIO pins for connecting sensors and actuators, it's versatile for various applications. Its small size and community support make it a popular choice for adding Wi-Fi connectivity to projects.

3.3 Raspberry PI 3B+

The Raspberry Pi 3B+ serves as a compact and cost-effective computing solution for our project. Its small form factor makes it suitable for integration into various systems, while its built-in Wi-Fi and Bluetooth capabilities streamline connectivity. The inclusion of GPIO pins enables the connection of sensors and peripherals, enhancing the functionality of our project. Additionally, the ability to expand storage through microSD cards and connect external devices via USB ports offers flexibility in data management and input/output options. With support for different operating systems, the Raspberry Pi 3B+ caters to diverse project requirements,

providing a versatile platform for development and implementation.

3.4 Arduino Uno

The Arduino Uno is a versatile microcontroller board utilized in our project for its simplicity and flexibility. Featuring a compact design, the Uno is easily integrated into various systems. Its user-friendly interface makes it accessible for beginners and professionals alike, enabling rapid prototyping and development. With a range of digital and analog input/output pins, the Uno facilitates connections to sensors, actuators, and other peripherals, enhancing the project's functionality. Moreover, its compatibility with a wide array of shields and modules further expands its capabilities, allowing for customization and scalability. By leveraging the Arduino Uno, our project benefits from a reliable and adaptable platform for experimentation, innovation, and realization of project objectives.

3.5 Other Components

Breadboard: A handy platform for building and testing circuits without soldering.

Piezo buzzer: A piezo buzzer is a small electronic component that makes noise when you apply electricity. It's compact, uses little power, and can produce different sounds. We're using it in our project to give alerts or signals to users.

Male-to-Male (M2M) Wires: These wires connect components together on the breadboard.

Male-to-Female (M2F) Wires: They link components on the breadboard to external modules.

Battery: Provides portable power for the project when mains power isn't available.

Power Supply: Ensures stable power for the project's components, either from batteries or mains power.

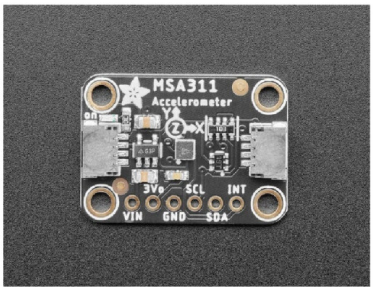

Figure 1. MSA311 Accelerometer

(a) Source: ElectronicsComp.com

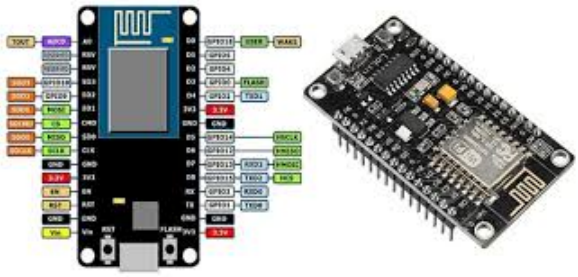

Figure 2. ESP8266 Wi-Fi Module

(a) Source: ResearchGate

Figure 3. Raspberry PI 3B+ Module

(a) Source: Seed Studio Wiki

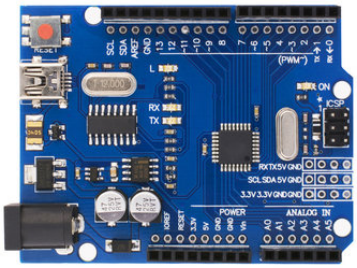

Figure 4. Arduino UNO

(a) Source: Adobe Stock

4 Methodology

4.1 Data Collection

Data Collection is done from the MSA311 Accelerometer in the form of x,y, and z axes. the vibrations are recorded at a rate of 200 measurements per second. For this, our computer is used as a server and ESP8266 acts as a node on the same network. The ESP8266 runs on an Arduino sketch which performs an HTTP GET request by connecting to the server. The status code is specified for each scenario with which, the server responds. Status code 1 refers to the server being "ready" and 0 refers to "not ready".

When the Wi-Fi module receives 1, meaning the server is ready, it collects 200 samples per second. Then, the data is bundled up in JSON format and is transmitted to the server using an HTTP POST request. The server parses the JSON message, extracting the x, y, and z measurements, and saves them in a new file format separated by commas. It persists in storing these files as long as it remains operational. The Arduino sketch contains information such as SSID, password, and hostname to connect to our Wi-Fi network. the working condition of the program can be verified through the terminal as long as debug is set to 1. This sends the GET request to the server. The MSA311 is attached to the top of the fan, while the weight is taped to the blade of the fan to collect anomaly samples.

The Server code is written in Python using the *http* server class. Then, the output directory where the data would be stored is specified and the code is run for some time to collect sufficient data in various possible conditions such as normal condition, weight condition, tapping condition, and falling condition.

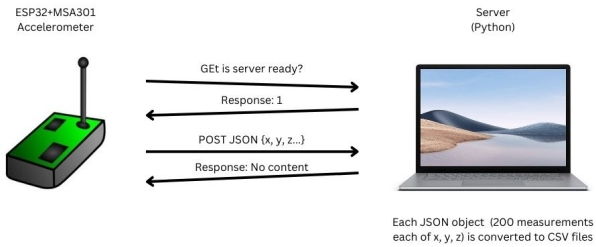

Figure 5. Image describing connection between Wi-Fi and laptop for data collection

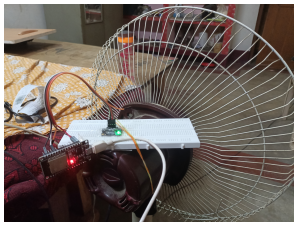

Figure 6. MSA311 Placement

Figure 7. Weight Placement for anomaly data

Figure 8. Data Collection stage circuit

Figure 9. Final Circuitry

4.2 Feature Extraction and Training

4.2.1 Analyzing Raw data

First, the raw data is plotted in the form of 3D coordinates for x, y and z axes for both normal and anomaly samples simultaneously to find out extractable features. Next, various operations such as finding mean, variance, skew, kurtosis, etc are performed on the raw data to find out features that can be used for making distinctions among normal and anomaly samples. This analysis leads to two very conclusive methods of distinction between normal and anomaly samples, they are Mahalanobis Distance and Autoencoder.

4.2.2 Mahalanobis Distance

Mahalanobis distance is a significant metric used to measure the separation between two points within a multi-dimensional space. It stands out from the traditional Euclidean distance by incorporating not only the individual differences along each dimension but also the correlations and scales of the variables involved. This distinction is particularly crucial when dealing with datasets where variables are interrelated. Instead of merely relying on the raw discrepancies between data points, Mahalanobis distance adjusts these differences by integrating the covariance structure of the dataset.

In contrast, the Euclidean distance between two points considers only the geometric distance between them, irrespective of the relationships between variables. This means that the Euclidean distance treats all dimensions equally, without accounting for any potential correlations or differences in scale among variables.The Mahalanobis distance between points A and B is calculated using the formula:

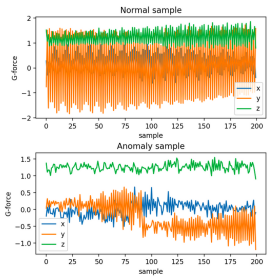

Figure 10. Normal plot of samples

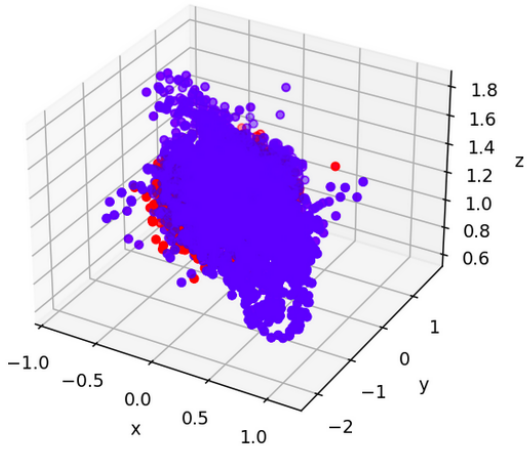

Figure 11. 3D Scatterplot

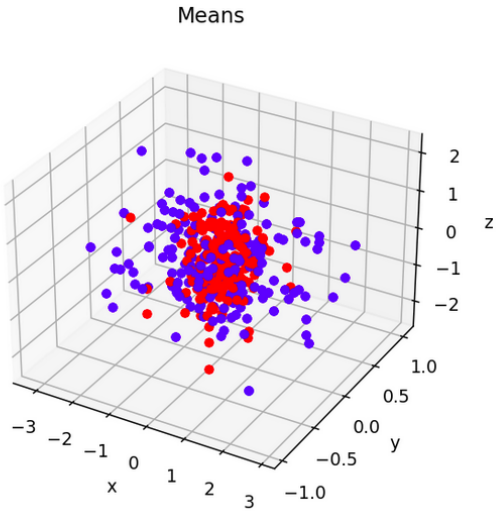

Figure 12. 3D Scatterplot of Means

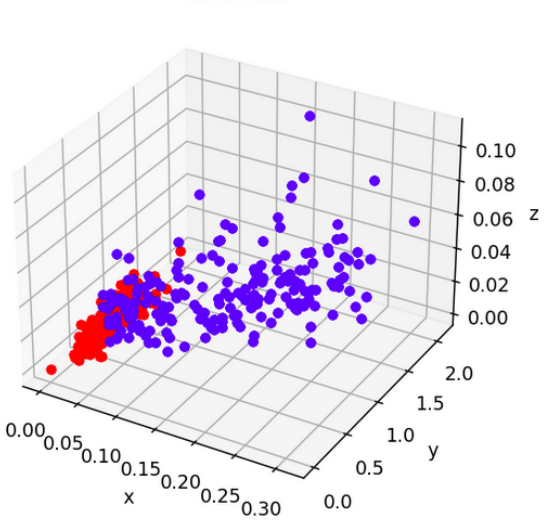

Figure 13. Variance plot

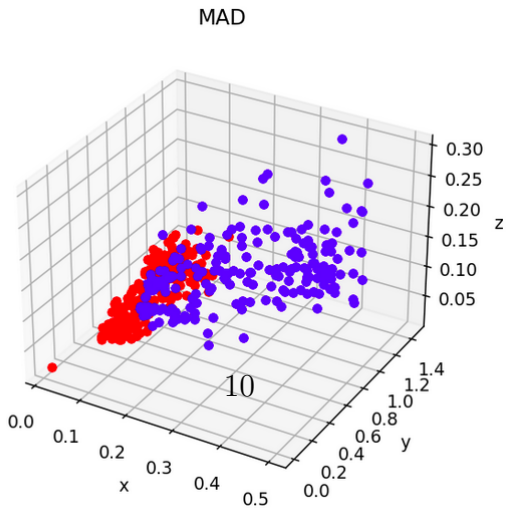

Figure 14. Median Absolute Deviation (MAD)

$$d_{\text{Mahalanobis}}(A, B) = \sqrt{(A - B)^T \cdot \text{Cov}^{-1} \cdot (A - B)}$$

Here, A and B are vectors representing the two points in the multi-dimensional space, and Cov^{-1} denotes the inverse covariance matrix of the dataset, which captures essential information about the correlation among variables.

The Mahalanobis distance's adjustment for covariance structure allows it to provide a more accurate assessment of similarity or dissimilarity between data points, making it applicable across various domains such as statistics, machine learning, and pattern recognition. It plays a vital role in tasks like outlier detection, clustering, and classification, where understanding the underlying relationships between variables is crucial.

This is used as a machine learning model to distinguish between normal and anomaly samples, the results of which are discussed in the "Results and Discussion" section.

4.2.3 Autoencoder

The functioning of an autoencoder involves encoding input data into a lower-dimensional latent space representation and then decoding it back to reconstruct the original input. This process is achieved through a series of mathematical operations performed by the encoder and decoder components of the autoencoder. The functioning along with the mathematical equations involved are explained below:

- **Encoder Functioning:** The encoder function f_{encoder} maps the input data X to the latent space representation Z. This mapping is performed using a series of affine transformations

followed by activation functions:

$$Z = f_{\text{encoder}}(X) = \sigma(W_{\text{enc}}X + b_{\text{enc}})$$

where:

- Z is the latent space representation.
- X is the input data.
- W_{enc} is the weight matrix of the encoder.
- b_{enc} is the bias vector of the encoder.
- σ is the activation function.

- **Decoder Functioning:** The decoder function f_{decoder} takes the latent space representation Z and reconstructs the original input X'. Similar to the encoder, the decoder applies affine transformations followed by activation functions to produce the reconstruction:

$$X' = f_{\text{decoder}}(Z) = \sigma(W_{\text{dec}}Z + b_{\text{dec}})$$

where:

- X' is the reconstructed output.
- W_{dec} is the weight matrix of the decoder.
- b_{dec} is the bias vector of the decoder.

- **Loss Function:** The loss function quantifies the discrepancy between the original input X and the reconstructed output X'. A common choice for reconstruction loss is the Mean

Squared Error (MSE):

$$\mathcal{L}(X, X') = \frac{1}{n}\sum_{i=1}^{n}(X_i - X_i')^2$$

where n is the number of input features.

- **Training:** During training, the autoencoder minimizes the reconstruction loss by adjusting the parameters using optimization algorithms such as stochastic gradient descent (SGD) or its variants. Gradients of the loss function with respect to the parameters are computed using backpropagation.

The results of Autoencoder training are discussed in the "Results and Discussion" section.

4.3 Model Deployment on Raspberry PI

After making sure that all connections are securely made and the codes are working fine during simulations, now it can be trained directly on the Raspberry PI. First, data is collected from the fan at a sampling rate of 200 samples per second. The data collection is done for various scenarios and ample amount of time to gather a satisfactory amount of data for training purposes. Now, the training script is run which produces a *.npz* file containing the mean and covariance matrices of the model. This script also provides us with recommended threshold to be used for the deployment program. The same can be done for the autoencoder model to generate *.tflite* file.

For Mahalanobis distance, the *.npz* file is copied to the Raspberry Pi and the threshold is set to the recommended value from the script run earlier. Then, the model is deployed and the deployment script is run which uses the threshold to check for anomalies and informs as such. Similarly, for the autoencoder model, the *.tflie* file is copied to the Raspberry Pi. The threshold

is adjusted according to the value provided by the previous script. Next, the model is deployed using the deployment script which uses the threshold to check for anomalies.

4.4 Model Deployment on Arduino UNO

4.4.1 Mahalanobis Distance

To deploy our model on Arduino UNO, our prediction code has to be converted from Python to C. To do this, we save our models as C header files and a few samples are saved for testing. Now, using the *utls/*, of the Arduino, which provides us with some functions to generate C code for header files, we save our models and example samples in C arrays. Then, the model conversion script is run which calculates the inverse of the covariance matrix and stores it, along with the mean from the original model, as a C array in a distinct header file.

Afterward, the script will convert a single normal sample and a single anomaly sample into distinct header files. These header files will be utilized for testing purposes. Ultimately, we aim to create a series of manual computations for determining the median, median absolute deviation (MAD), matrix multiplication, and Mahalanobis distance (MD). These calculations will then be translated into C code. Next, we copy the newly generated .h files, including the model and samples, to our sketch folder. Consequently, separate tabs for the model file, samples, and utilities become visible. We execute the program, allowing it to read the samples. Finally, we calculate the Median Absolute Deviation (MAD) and Mahalanobis Distance between each sample and the mean of the model.

Finally, the model is deployed after attaching the ESP8266, accelerometer, and buzzer to the fan and the sketch is run. In case the fan blades are hit or tapped upon the buzzer sounds an alarm showing that the model is working well.

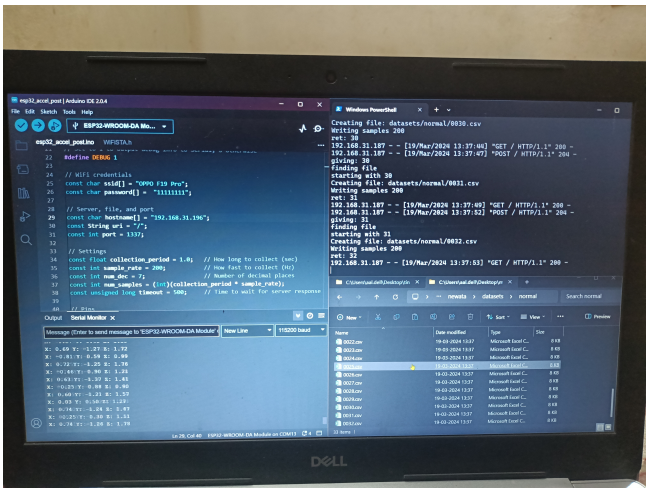

Figure 15. Model Deployment on Arduino

4.4.2 Autoencoder

As we'll be utilizing the TensorFlow Lite for Microcontrollers Arduino library, there's no need for us to develop as many manual calculation functions for the Autoencoder as we did for the MD. Nonetheless, it remains essential for us to execute the Notebook to convert the model. Within that Notebook, the initial step involves creating a TensorFlow Lite (.tflite) model from the .h5 Keras model file. Subsequently, a C array is generated and stored in a header file, containing the raw FlatBuffer model.

We'll be using the same normal and anomaly samples from the MD conversion file. Ultimately, we utilize these identical normal and anomaly samples to generate MAD values, conduct inference with our model, and compute the mean squared error (MSE). We'll compare these outputs with those in Arduino to ensure consistency. Now, the Arduino sketch is run on the Wi-Fi module after copying the generated .h files to the sketch folder. After confirming that inference functions correctly and our calculations yield accurate results in C, we're prepared to develop our deployment code.

Similar to Mahalanobis distance model, the model is deployed after attaching the ESP8266, accelerometer, and buzzer to the fan. In case the fan blades are hit or tapped upon the buzzer sounds an alarm showing that the model is working well.

5 Results and Discussions

We explored the process of anomaly detection using vibration data collected from a table fan. We first analyzed the raw accelerometer data, performed the statistical analysis, and then trained two machine learning models: Mahalanobis Distance and Autoencoder Neural Network. Subsequently, we deployed these models onto an Arduino microcontroller for real-time anomaly detection.

5.1 Statistical Analysis:

We began by analyzing various statistical properties of the data, including mean, variance, kurtosis, skewness, median absolute deviation (MAD), and correlation. MAD emerged as a promising feature for discerning normal operations from anomalies, showing a clear separation between normal and anomaly samples.

5.2 Machine Learning Models:

5.2.1 Mahalanobis Distance Model:

- Calculated the Mahalanobis distance for each sample based on the mean and covariance matrix of normal samples.
- Established a threshold based on the Mahalanobis distance to classify anomalies.
- The model showed good performance initially but struggled with robustness when the system's states varied.

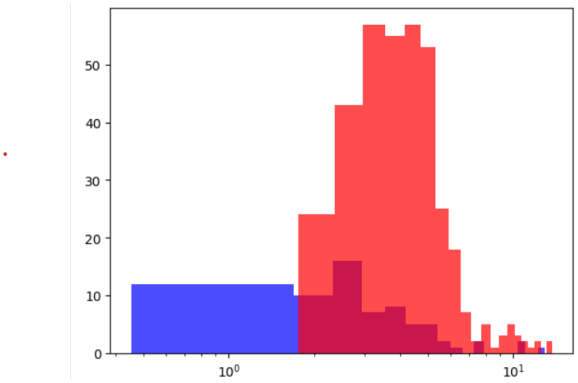

Figure 16. Histograms of normal validation vs. anomaly sets (MSEs)

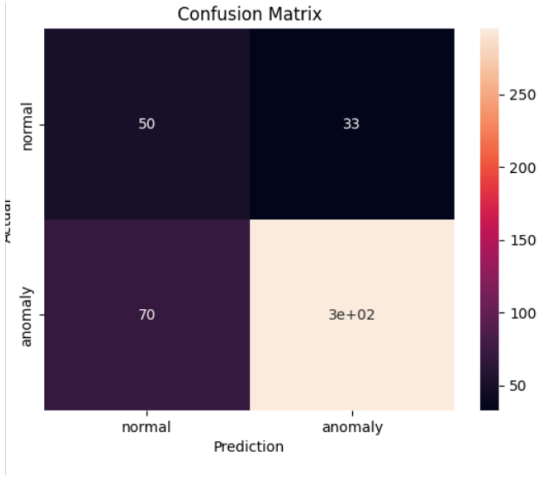

Figure 17. Confusion Matrix

5.2.2 Autoencoder Neural Network Model:

- Utilized an autoencoder to reconstruct normal MAD values and detect anomalies based on higher mean squared error (MSE).
- Showed better robustness compared to the Mahalanobis Distance model, as it could adapt to varying states of the system.

5.3 Data collection on ESP32:

- Recorded data using the ESP32 microcontroller for both normal and abnormal conditions.
- Introduced abnormal conditions by adding weight wings to the fan blades to simulate different operating states.
- Extended data collection to approximately 30 minutes to capture a more comprehensive range of operating conditions.

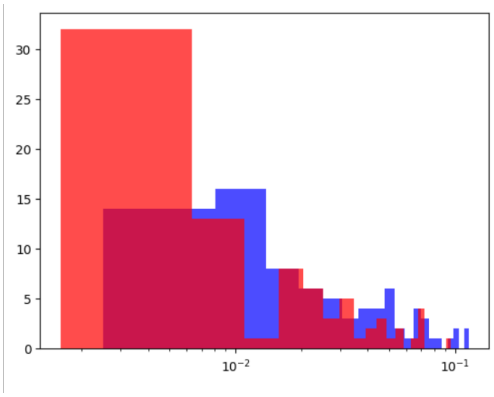

Figure 18. Histograms of normal validation vs. anomaly sets (MSEs) using autoencoder

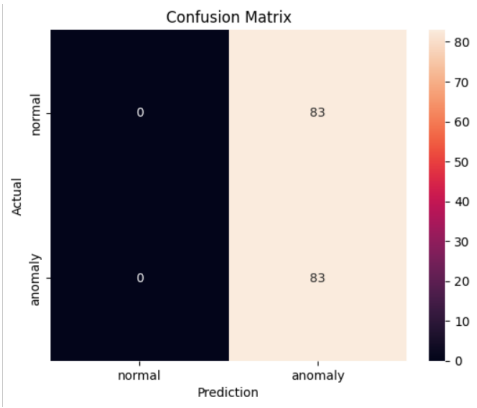

Figure 19. Confusion Matrix

5.4 Deployment onto Arduino:

- Converted the trained models into C code for deployment onto an Arduino microcontroller.
- Tested the models with both the new normal and abnormal datasets to ensure proper functioning.

5.5 Experimental Observations:

5.5.1 Mahalanobis Distance Deployment:

- Despite initial promising results, the Mahalanobis Distance model exhibited limited robustness when the system's states varied. It only detected anomalies under extreme conditions, such as violent shaking.
- The model's performance was insufficient for practical deployment, indicating the need for a more adaptable approach.

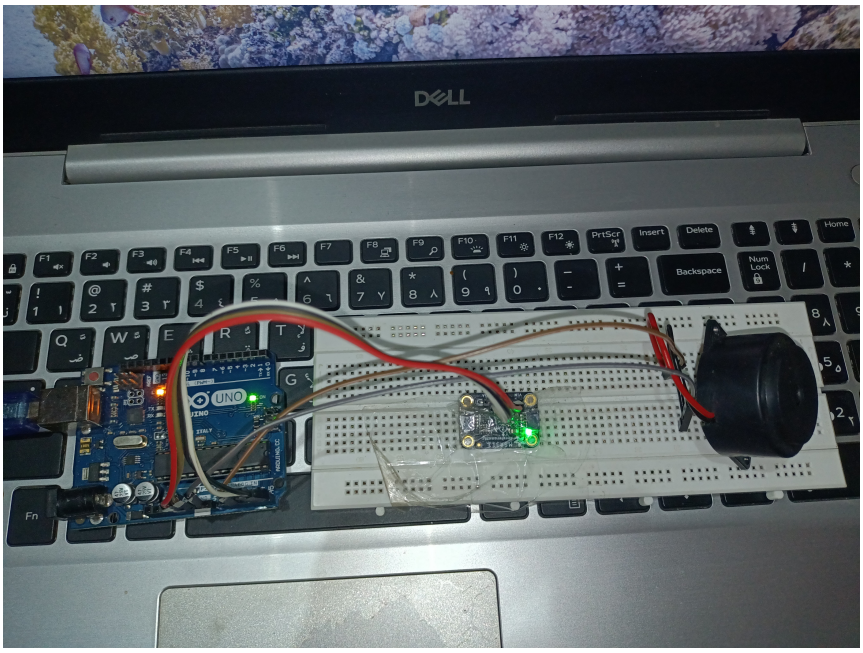

Figure 20. Model deployed on Arduino

5.5.2 Autoencoder Deployment:

- The Autoencoder Neural Network model demonstrated superior performance in detecting anomalies, even with variations in the system's states.
- It accurately identified anomalies such as fan blade taps and motor offsets, showing resilience to environmental changes.

6 Conclusion

The usefulness of using both Classic machine learning techniques and autoencoder neural network created with TensorFlow and Keras models to detect anomalies in electric motors is being demonstrated. The choice of methods is determined by the data's complexity and the level of robustness required. In instances where the environment is more dynamic, such as a moving fan, autoencoders are more effective at capturing and detecting anomalies. The fan data included a variety of operating modes like off, low, medium, high and anomalous circumstances, resulting in a comprehensive dataset for training and evaluating models. It comprised examples with and without extra weight, allowing the models to accurately distinguish between normal and abnormal actions.

The Mahalanobis distance method effectively uses statistical measures to find deviations from normal operating patterns. This approach is simple and computationally efficient, but it may struggle with complex or nonlinear patterns. Using an autoencoder neural network built with TensorFlow and Keras improves adaptability for capturing complicated data patterns. The autoencoder neural network outperformed the Mahalanobis distance method, especially in cases when the fan's base moved significantly during normal operation. This shows that autoencoders can offer superior performance in detecting anomalies in dynamic systems with varying conditions.

References

[1] A. Bousdekis, D. Apostolou and G. Mentzas, "Predictive Maintenance in the 4th Industrial Revolution: Benefits, Business Opportunities, and Managerial Implications," in IEEE Engineering Management Review, vol. 48, no. 1, pp. 57-62, 1 Firstquarter,march 2020, doi: 10.1109/EMR.2019.2958037.

[2] B. V. Ramani, C. A. Amith, J. M. Oommen, J. Babu, T. Paul and V. Sankar, "Predictive analysis for industrial maintenance automation and optimization using a smart sensor network," 2016 International Conference on Next Generation Intelligent Systems (ICNGIS), Kottayam, India, 2016, pp. 1-5, doi: 10.1109/ICNGIS.2016.7854004.

[3] Fathi K, van de Venn HW, Honegger M. Predictive Maintenance: An Autoencoder Anomaly-Based Approach for a 3 DoF Delta Robot. Sensors (Basel). 2021 Oct 21;21(21):6979. doi: 10.3390/s21216979. PMID: 34770289; PMCID: PMC8588519.

[4] Zonta, Tiago, Cristiano André Da Costa, Rodrigo da Rosa Righi, Miromar Jose de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. "Predictive maintenance in the Industry 4.0: A systematic literature review." Computers Industrial Engineering 150 (2020): 106889.

[5] Guillén, Antonio J., Adolfo Crespo, Juan Fco Gómez, and Maria Dolores Sanz. "A framework for effective management of condition based maintenance programs in the context of industrial development of E-Maintenance strategies." Computers in Industry 82 (2016): 170-185.

[6] Mobley, R. Keith. An introduction to predictive maintenance. Elsevier, 2002.

[7] Bhide, Abhishek Ghodake, Dnyaneshvar Jamle, Ashish Shaikh, Salman Bhujbal, Prof. (2023). Predictive Machine Maintenance Using Tiny ML. International Journal for Research in

Applied Science and Engineering Technology. 11. 4252-4255. 10.22214/ijraset.2023.51254.

[8] Givnan, Sean Chalmers, Carl Fergus, Paul Ortega-Martorell, Sandra Whalley, Tom. (2021). Real-Time Predictive Maintenance using Autoencoder Reconstruction and Anomaly Detection.

[9] Roda Irene, Marco Macchi, and Luca Fumagalli. "The future of maintenance within industry 4.0: An empirical research in manufacturing." In Advances in Production Management Systems. Smart Manufacturing for Industry 4.0: IFIP WG 5.7 International Conference, APMS 2018, Seoul, Korea, August 26-30, 2018, Proceedings, Part II, pp. 39-46. Springer International Publishing, 2018.

[10] https://github.com/Gitaalekh6763/tinyml-example-anomaly-detection

[11] https://www.digikey.com/en/maker/projects/edge-ai-anomaly-detection-part-1-data-collection

[12] https://www.digikey.com/en/maker/projects/edge-ai-anomaly-detection-part-2-feature-extraction-and-model-training/70927a6e439b49bea7305953a3c9bfff

[13] https://www.digikey.com/en/maker/projects/edge-ai-anomaly-detection-part-3-machine-learning-on-raspberry-pi/af9dd958b23d4ea1b40bc3cc060ef8c9

[14] https://www.digikey.com/en/maker/projects/edge-ai-anomaly-detection-part-4-machine-learning-models-on-arduino/afacfc3dbaf24c6c94a55c4afae1afb2