**Using TinyML Concepts to Alert Cyclist/Motorcyclist about Potential Road Hazard**

Jayanth Alapati   
Computer Science Department  
Kent State UniversityKent, Ohio, USA  
[jalapati@kent.edu](mailto:jalapati@kent.edu)

*Abstract*—When a cyclist is riding his bike on a road on the pavement, he might not be able to focus on the traffic behind him. Frequent shoulder checks for other cyclists or people while cycling can be dangerous as your attention is diverted away from oncoming traffic or obstacles. We implement a Machine Learning model embedded in a microcontroller where we can detect the potential motion detection behind the rider to warn him about the approaching vehicle. The model used here is the CNN where the classification of the images is done based the vehicles or not vehicles and the trained model is then converted to tflite using TensorFlow libraries. The microcontroller used here is “The Arduino Nano 33 BLE Sense” is a completely new board on a well-known form factor. It comes with a series of embedded sensors 9 axis inertial sensor what makes this board ideal for wearable devices humidity, and temperature sensor to get highly accurate measurements of the environmental conditions’ barometric sensor: you could make a simple weather station microphone: to capture and analyze sound in real time gesture, proximity, light color and light intensity sensor estimate the room’s luminosity, but also whether someone is moving close to the board.

The goal of the paper is to learn the basic understanding about TinyML and its implementation so that we can work on future works related to this subject in a better detailed manner.

Keywords—TensorFlow, CNN, tinyML, sensor, microcontroller, accuracy, TensorFlow Lite

# Introduction

The combination of machine learning (ML) with embedded systems in recent years has prepared the way for a transformative field known as Tiny Machine Learning (TinyML). This developing paradigm focuses on directly putting lightweight and power-efficient machine learning models on edge devices like microcontrollers, enabling real-time, on-device computation. TinyML is a considerable divergence from typical techniques that rely on cloud-based processing, and it provides a viable solution for applications that require low latency, privacy, and energy efficiency.

As the number of Internet of Things (IoT) devices grows, the need for efficient and intelligent processing at the edge becomes more obvious. Traditional machine learning models, which are frequently resource-intensive, suffer difficulties when implemented on resource-constrained edge devices. TinyML solves this problem by customizing machine learning algorithms to the limits of microcontrollers and other embedded devices, hence opening up new possibilities for a wide range of applications.

Cyclists face a multitude of dangers on the road, including the risk of collisions with motor vehicles at intersections, poor road conditions such as uneven surfaces and debris, adverse weather conditions reducing visibility and increasing slipperiness, distracted driving by motorists using smartphones, the potential for "dooring" accidents when parked cars open their doors without checking for cyclists, and encounters with aggressive drivers. Inadequate protective gear, such as the absence of helmets, heightens the risk of injury in case of accidents. Other factors contributing to cyclist vulnerability include low visibility during nighttime or adverse weather, the presence of inexperienced cyclists unfamiliar with traffic rules, and the possibility of wildlife encounters in certain environments. Addressing these dangers requires a combination of cyclist awareness, adherence to safety measures, improved infrastructure, and increased public awareness about sharing the road safely.

Out of the list of problems we identified the issue of road accident could be a primary threat for cyclist and decided to work on a model which detects the vehicle which approaches nearer to the cyclist that could turn into a potential threat and informs the cyclists riding the bike through a microcontroller. The microcontroller used for this experiment is Arduino Nano 33 BLE Sense.

The Nano 33 BLE Sense is likewise extremely small, measuring 45 x 18 mm. This makes it ideal for projects requiring a little amount of space, such as wearable gadgets or drones. Despite its diminutive size, the board boasts a fast processor and a comprehensive sensor array.

Finally, the Nano 33 BLE Sense can be completely programmed using the Arduino IDE. This makes it simple to begin designing your own applications, even if you are a new developer. The Arduino Nano 33 BLE Sense is an excellent choice for any developer looking to create unique and interactive projects because to its extensive feature set and ease of use.

# Literature survey and Background

### *An Unsupervised TinyML Approach Applied for Pavement Anomalies Detection Under the Internet of Intelligent Vehicles," P. Andrade et al., 2021*

Summary: This paper presents a novel approach for detecting road anomalies (potholes, bumps, etc.) using an unsupervised TinyML technique implemented on a resource-constrained device (Arduino Nano 33 IoT) embedded in a vehicle. The proposed approach leverages the TEDA algorithm, which analyzes the total acceleration data collected from an accelerometer to identify anomalies based on typicality and eccentricity measures. The proposed approach achieved promising results in real-world experiments, with an average f1 score of 0.76 for the first driver and 0.78 for the second driver. The unsupervised nature of the TEDA algorithm eliminates the need for labeled data, making it suitable for real-time anomaly detection in diverse road conditions. [1]

The TinyML implementation allows efficient computation on resource-constrained devices, enabling edge computing and reducing reliance on cloud processing. The proposed model achieves an accuracy of about 98.7%. In Smart cities the real-time detection of road anomalies can help authorities prioritize maintenance efforts and improve road safety. Connected vehicles: Vehicles can share anomaly data with each other, creating a collaborative network for comprehensive road condition monitoring. Usage-based insurance: Insurance premiums can be adjusted based on individual driving behavior and the presence of road anomalies. [1]

Integration with additional sensors can be done by Combining accelerometer data with other sensors like GPS and gyroscope can improve anomaly detection accuracy. Energy-efficient implementation: Optimizing the TinyML model and hardware platform can reduce energy consumption and extend battery life. Data windowing: Implementing a data windowing technique can adapt to changes in pavement type and improve detection accuracy in diverse road environments.

Overall, this paper demonstrates the potential of TinyML for real-world applications in the Internet of Intelligent Vehicles domain. The proposed approach offers a promising solution for improving road safety and efficiency through real-time anomaly detection.

### M. Antonini, M. Pincheira, M. Vecchio and F. Antonelli, "A TinyML approach to non-repudiable anomaly detection in extreme industrial environments, 2022

Summary: This paper tackles the challenge of industrial equipment monitoring by proposing a novel system combining TinyML and blockchain technology. Leveraging a low-cost IoT kit, the system utilizes TinyML to run an unsupervised anomaly detection algorithm directly on the equipment, overcoming resource limitations.

This algorithm learns normal behavior patterns and autonomously detects anomalies, providing early warnings of potential failures. Before real-world deployment in extreme industrial environments, the system is rigorously validated in a simulated testbed. [2]

Performance metrics like inference speed, execution times, and memory usage are carefully evaluated. Upon detecting anomalies, the system generates signed blockchain transactions containing relevant data and stores them securely in a public blockchain using smart contracts. This transparent and auditable record-keeping significantly enhances the maintenance process.

The proposed system achieved an accuracy of 99.5% in detecting anomalies in industrial environments.

Additionally, the paper analyzes the overhead associated with using a public blockchain, considering transaction sizes, gas consumption costs, and pricing models. In essence, this innovative system offers cost-effective on-device anomaly detection for industrial assets, validated for harsh environments, and further enhanced by the transparency and auditability of blockchain technology. [2]

### . Zacharia et al., "An Intelligent Microprocessor Integrating TinyML in Smart Hotels for Rapid Accident Prevention," 2022

Summary: This paper introduces the ZAC888DP, an intelligent microprocessor designed for swift accident prevention in smart hotels, integrating TinyML capabilities. Key points include: The ZAC888DP serves as an all-in-one microcontroller for automating hotel room functions such as lighting, HVAC, shading, doors, and energy monitoring.

An ESP32 microcontroller, gathering multivariate sensor data (water, light, temperature, humidity), simulates a lavatory environment to detect potential accidents using TinyML. A dataset is generated for model training. [3]

A compact deep neural network model attains 100% accuracy in distinguishing safe and accident-prone scenarios after 256 training iterations. It is converted to TensorFlow Lite format for TinyML deployment.

The pre-trained model is integrated into the ZAC888DP microcontroller to trigger alerts for accident prevention when real-time sensor data exceeds risk thresholds. Safe and unsafe use cases are illustrated.

The proposed system ensures prompt responses to potential lavatory accidents in hotels through embedded intelligence, concurrently managing energy efficiency. The ZAC888DP's connectivity and TinyML capabilities make it suitable for smart building applications. [3]

Future work involves linking the system to mobile devices, incorporating additional sensor parameters, testing it in real-world settings, and extending its application to accident prevention in diverse environments such as nursing homes

### Z. Chen, Y. Gao and J. Liang, "A Self-Powered Sensing System with Embedded TinyML for Anomaly Detection," 2023

Summary: In order to sense vibration data, it makes use of a lightweight piezoelectric self-powered sensor (SPS) as opposed to a power-hungry inertial measurement unit (IMU). Without requiring external power, the SPS produces a voltage signal proportionate to equipment vibration.

A compact deep neural network model that can accurately analyze the distorted SPS data is constructed using TinyML techniques. With just 8 data points, the model was able to classify vibration states with 97.6% accuracy. [4]

A low-cost microcontroller unit (MCU) powers the system sporadically, spending the majority of its time in an energy-saving sleep mode. This allows for low power consumption real-time edge analysis.

According to experimental results, the self-powered sensing approach used embedded intelligence to reliably detect anomalies while saving 66.74% of the energy required when compared to using an IMU. [4]

With pervasive sensing at the edge devices, the proposed system offers a valuable platform for achieving ubiquitous artificial intelligence. Its self-powered, inexpensive, and low-power design makes it appropriate for a range of applications involving equipment monitoring.

### K. Fang, Z. Xu, Y. Li and J. Pan, "A Fall Detection using Sound Technology Based on TinyML," 2021

Summary: This Paper introduces an innovative fall detection system leveraging sound technology and TinyML, particularly designed for ensuring the safety of the elderly. Addressing the significance of fall detection in elder care, the paper suggests that utilizing sound as a detection method can complement existing approaches such as computer vision and accelerometers. [5]

The system acquires data on falling sounds and background noise, subsequently employing short-time Fourier transform to extract features and generate spectrograms. These spectrograms serve as input for a convolutional neural network model constructed and trained using TensorFlow, achieving a validation accuracy exceeding 90% in identifying falls amidst various sounds.

To facilitate deployment on devices with constrained computing resources, the TensorFlow model undergoes conversion to a TensorFlow Lite model, maintaining high accuracy while significantly reducing its size. Future endeavors involve enhancing model accuracy further, integrating sound detection with accelerometer data, and deploying the TensorFlow Lite model on a microcontroller.

The proposed sound-based fall detection system emerges as a promising addition to existing methodologies, particularly tailored for wearable devices. [5]

## CNN Architecture

Convolutional Neural Networks (CNNs) are a class of deep neural networks designed for processing structured grid data, particularly images. Their architecture comprises convolutional layers that systematically learn hierarchical patterns and features from input data. Convolutional operations enable the networks to detect local patterns, while pooling layers reduce spatial dimensions and enhance translation invariance. CNNs often consist of multiple convolutional and pooling layers followed by fully connected layers for classification. The convolutional layers use learnable filters to convolve over the input, capturing features like edges, textures, and complex structures. Convolutional Neural Networks have revolutionized computer vision tasks by automatically learning hierarchical representations from raw pixel data, making them adept at tasks such as image classification, object detection, and segmentation.

## Jupyter tool

Jupyter is a free and open-source web tool that lets users create and share documents with live code, equations, visualizations, and narrative text. Its applications include interactive data analysis, scientific computing, and machine learning. Jupyter notebooks, commonly known as IPython notebooks, are the platform's major feature. Notebooks provide an interactive computational environment in which users can create and execute code directly in a web browser, eliminating the need for a separate development environment.

## Arduino Nano 33 BLE sense

With an abundance of integrated sensors and Bluetooth Low Energy (BLE) connectivity, the Arduino Nano 33 BLE Sense is a potent and adaptable microcontroller board that takes the classic Arduino Nano form factor to new heights. A 32-bit ARM Cortex-M4 processor clocked at 64 MHz, 1 MB of flash memory, 320 kB of RAM, and numerous onboard sensors are included. These include a temperature sensor, a pressure sensor, a humidity sensor, a light sensor, a color sensor, and a microphone. Additionally, there is a 9-axis IMU (accelerometer, gyroscope, and magnetometer). The Nano 33 BLE Sense's sensory suite enables it to detect motion, orientation, environment, and even sound, which makes it an excellent option for a variety of applications, including robotics, environmental monitoring, wearables, and Internet of Things devices. The Nano 33 BLE Sense's onboard BLE features also allow it to interact wirelessly with other BLE-enabled devices, such as tablets and smartphones, creating a plethora of opportunities for automation, control, and data collection. It is also simple to begin developing on the board because it is compatible with the Arduino IDE and a sizable library of pre-written Arduino code. For makers, enthusiasts, and professionals wishing to create dynamic and inventive projects, the Arduino Nano 33 BLE Sense is an excellent option because of its small size, robust functionality, and user-friendliness.

## Visual Studio Code

Visual Studio Code (VS Code) serves as a versatile platform for utilizing Python in machine learning projects, offering two primary approaches: Jupyter notebooks and Python code files with code cells. Jupyter notebooks offer an interactive environment that combines text, code, and visualizations into a single document, making data analysis, Python programming, and visualization creation easy to do. Installing the Jupyter extension, starting a new notebook, choosing a kernel, writing and running code cell by cell, and using tools like matplotlib or seaborn to visualize the output are the steps involved in using this method. As an alternative, VS Code offers an organized environment for organizing code by enabling the execution of Python code inside standard Python files via code cells. Each strategy has certain advantages: Jupyter notebooks facilitate interaction, whereas code cells offer organization. Still, Whatever your preference, you can use Python in your machine learning projects with the help of VS Code's integrated environment, flexibility, customization options, and large community support.

## Arduino IDE

The open-source Arduino Integrated Development Environment (IDE) software platform makes it easier to write, compile, and upload code to Arduino microcontrollers. Both novice and seasoned developers can utilize it because it offers an intuitive interface for programming Arduino boards. The Arduino programming language is a condensed form of C/C++, and the IDE supports it. It also comes with a code editor that highlights syntax, a compiler, and a bootloader that allows code to be uploaded via USB to Arduino boards. Furthermore, the Arduino IDE is a flexible and popular tool in the maker and electronics community because it includes a variety of libraries and examples that make it easier to develop different projects.

## TensorFlow

TensorFlow, an open-source platform developed by Google, has revolutionized machine learning by enabling the creation and deployment of powerful models. Its popularity stems from its adaptability, scalability, ease of use, and ability to handle large datasets. Whether for image recognition, speech processing, language understanding, or time series analysis.

TensorFlow offers a wealth of pre-built models and customizable tools. Its intuitive interface empowers both novice and seasoned developers to build and deploy models seamlessly. Additionally, TensorFlow's distributed computing capabilities enable high-performance training and evaluation across multiple machines, making it ideal for large-scale projects.

Overall, TensorFlow's versatility, user-friendliness, and powerful features solidify its position as a leading platform for machine learning development and deployment.

## TensorFlow Lite

TensorFlow Lite empowers developers to build intelligent edge applications. With its lightweight runtime, this open-source framework makes it possible to run models on devices with limited resources with greater efficiency.

This results in decreased latency, enhanced privacy, less bandwidth usage, and offline functionality.

TensorFlow Lite's cross-platform compatibility, adaptable model formats, configurable inference, hardware acceleration, and optimization tools, mobile and edge computing have several exciting new possibilities.

Identify applicable funding agency here. If none, delete this text box.

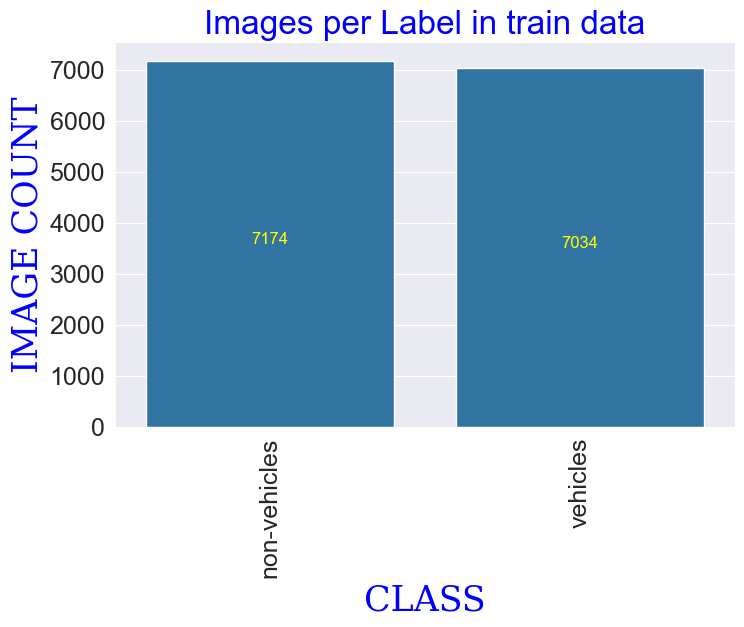
# Methodology

## Data Preprocessing

The dataset comprises of Vehicle images which consists of different kinds of vehicles and their positions along with another dataset which are plain clear roads. The code encompasses three functions designed for efficient data organization in machine learning. Initially, the function define\_paths(data\_dir) retrieves file paths and labels from a designated directory, assuming distinct classes within subdirectories. Subsequently, the function define\_df(files, classes) consolidates this information into a Pandas DataFrame with 'filepaths' and 'labels' columns.

The third function, create\_df(data\_dir), employs the previous functions to create training, validation, and test DataFrames, allocating data into 80% for training, 10% for validation, and 10% for testing via train\_test\_split. Stratification ensures a balanced distribution of classes across the splits. [4]

This systematic approach simplifies data preparation, providing an organized means to handle file paths and labels while generating datasets tailored for model training and evaluation in machine learning applications. Additionally, another code segment introduces a function, create\_gens, streamlining the establishment of image data generators for training, validation, and testing in a machine learning model.



*Fig. III. A.1. Image classification of training data*

It defines parameters, computes a custom test batch size, initializes data augmentation settings, and generates ImageDataGenerator objects for the respective datasets. This comprehensive methodology simplifies the preparation of image data for model training and evaluation, incorporating data augmentation for enhanced model generalization

## Data Analysis

The supplied code introduces a function, show\_images, designed to exhibit a subset of images from a given data generator. This function takes the generator (gen) as input, retrieves class information, and acquires a batch of images and labels from the generator. It then determines the number of images to display (up to a maximum of 25) and presents them in a 5x5 grid using Matplotlib.

The images are normalized by scaling to the range of 0 to 255, and each subplot includes the image along with its associated class name. This visualization serves as a rapid overview of the images within the specified batch, aiding in data exploration and comprehension.

Additionally, the code defines two functions, namely plot\_label\_count and plot\_labels. The primary function, plot\_label\_count, takes a DataFrame (df) and a plot title as inputs. It computes the value counts of the 'labels' column in the DataFrame and checks if the number of unique labels exceeds 55. If so, a message is printed indicating that no plot will be generated. [2]



*Fig. III. B. 1. Image analysis after processing*

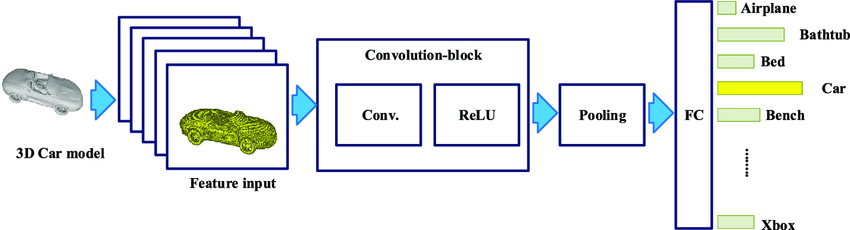
Otherwise, it invokes the secondary function, plot\_labels, passing the necessary parameters. The plot\_labels function is responsible for generating a bar plot of label counts. It receives the count of labels (lcount), label names (labels), corresponding values (values), and the plot title. The plot is configured with appropriate styling, including label rotation based on the number of labels. [4]

The resulting visualization illustrates the distribution of images across different classes in the provided DataFrame, offering insights into the label distribution for effective data exploration.

## Model Building

This code constructs a convolutional neural network (CNN) model using the Keras Sequential API, designed for tasks related to image classification. The architecture is tailored to process images of dimensions (224, 224) with a single channel.

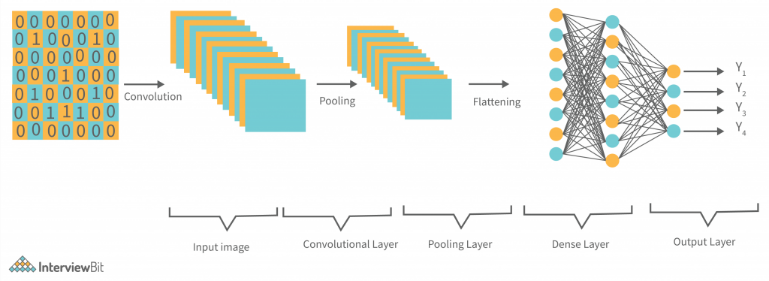
The model is composed of several convolutional layers, featuring pairs of 64, 128, and 256 filters, each followed by ReLU activation and max-pooling. The flattened output is directed through two dense layers with 256 and 64 units, respectively, both employing ReLU activation. [2]



*Fig. III. C. 1 CNN Architecture for Vehicle Image Detection*

The ultimate dense layer adjusts its unit count based on the number of classes identified by the training generator, employing softmax activation for multi-class classification. The model is compiled using the Adamax optimizer with a learning rate set at 0.001, categorical crossentropy serving as the loss function for multi-class classification, and accuracy as the designated evaluation metric.

The model.summary() function provides a comprehensive overview of the model, furnishing details on each layer's type, output shape, and parameter count. In summary, this code establishes a robust CNN architecture, tailored for image classification tasks, and is prepared for training with specific optimization and evaluation configurations.



*Fig. III. C. 2. CNN Architectural Diagram*

## Model Training

***batch\_size***: 40, indicating the number of samples processed in each training iteration.

***epochs:*** 100, representing the total number of training epochs.

***patience:*** 1, the number of epochs to wait before adjusting the learning rate if the monitored value (accuracy) does not improve.

***stop\_patience:*** 3, the number of epochs to wait before stopping training if the monitored value does not improve.

***threshold***: 0.9, if the training accuracy falls below this threshold, the callback will monitor validation loss instead.

***factor:*** 0.5, the factor by which the learning rate is reduced when needed.

***ask\_epoch:*** 5, the number of epochs to run before prompting the user to decide whether to continue or halt training.

***batches:*** Calculated based on the size of the training dataset and the specified batch size.

***callbacks:*** Includes a custom callback (MyCallback) instantiated with the specified parameters, designed to implement specific behavior during training such as adjusting the learning rate and potentially stopping training based on conditions.

This code trains the specified model (model) using data from the training generator (train\_gen) for a predefined number of epochs. The training is conducted without displaying intermediate output (verbose=0). Custom callbacks (callbacks) are utilized to implement specific functionalities during training, such as adjusting the learning rate and potentially stopping training under certain conditions. The model's performance is evaluated using validation data from the validation generator (valid\_gen). The order of the data is not shuffled during training (shuffle=False). The training history, containing metrics and loss values, is stored in the history variable.

## Convert the TensorFlow model to TensorFlow Lite

The TensorFlow model can be converted to a lite version by loading a pre-trained TensorFlow Keras model from a specified file path and subsequently converts it into a quantized TensorFlow Lite (TFLite) model using the TFLiteConverter. [1]

The quantization process optimizes the model for deployment on devices with limited resources, making it particularly suitable for applications in the realm of TinyML. The resulting quantized TFLite model is then saved to a binary file named 'quantized\_model.tflite'.

In essence, the code streamlines the conversion and quantization of a pre-trained Keras model, ensuring efficiency for deployment on resource-constrained platforms.

A tool name xxd is used to generate a C/C++ header file (modelvehiclemodel.cc) that contains a hexadecimal representation of the content of the 'quantized\_model.tflite' file. The !type command is then employed to display the contents of the generated C/C++ header file in the console. In summary, these commands are part of a process to embed the quantized TFLite model into a C/C++ source file, making it convenient for integration into applications or firmware. [2]

## Model Evaluation

This code segment assesses the performance of the trained model across various datasets, namely the training set (train\_gen), validation set (valid\_gen), and test set (test\_gen). The appropriate batch size for the test data is dynamically determined based on the dataset's length and a specified threshold. Subsequently, the number of steps required to cover the entire test dataset is calculated using this batch size. The model.evaluate function is then utilized to compute loss and accuracy scores for each dataset, storing the results in variables such as train\_score, valid\_score, and test\_score. The obtained scores are presented in a formatted report, displaying the loss and accuracy metrics for each dataset, thereby offering a comprehensive assessment of the model's performance on distinct data partitions

24/24 [==============================] - 3s 95ms/step - loss: 0.0324 - accuracy: 0.9896

24/24 [==============================] - 2s 97ms/step - loss: 0.0451 - accuracy: 0.9823

24/24 [==============================] - 4s 167ms/step - loss: 0.0478 - accuracy: 0.9870

Train Loss: 0.032438091933727264

Train Accuracy: 0.9895833134651184

--------------------

Validation Loss: 0.04507618770003319

Validation Accuracy: 0.9822916388511658

--------------------

Test Loss: 0.04782409965991974

Test Accuracy: 0.9870495200157166

*Fig. III. F. 1 Evaluation Results*

# Implementation

Using an external camera module and the Arduino Nano 33 BLE Sense, which has Bluetooth Low Energy (BLE) capabilities, the mode is implemented. This setup uses a camera sensor as its main sensor, which enables the processing of visual data. [2]

Using TensorFlow Lite for Microcontrollers, the camera module, compatible with the Nano 33 BLE Sense, takes pictures that are processed for a nearby vehicle detection that is behind a cyclist. TensorFlow Lite is appropriate for microcontrollers such as the Arduino Nano 33 BLE Sense because it allows the deployment of machine learning models on devices with limited resources.

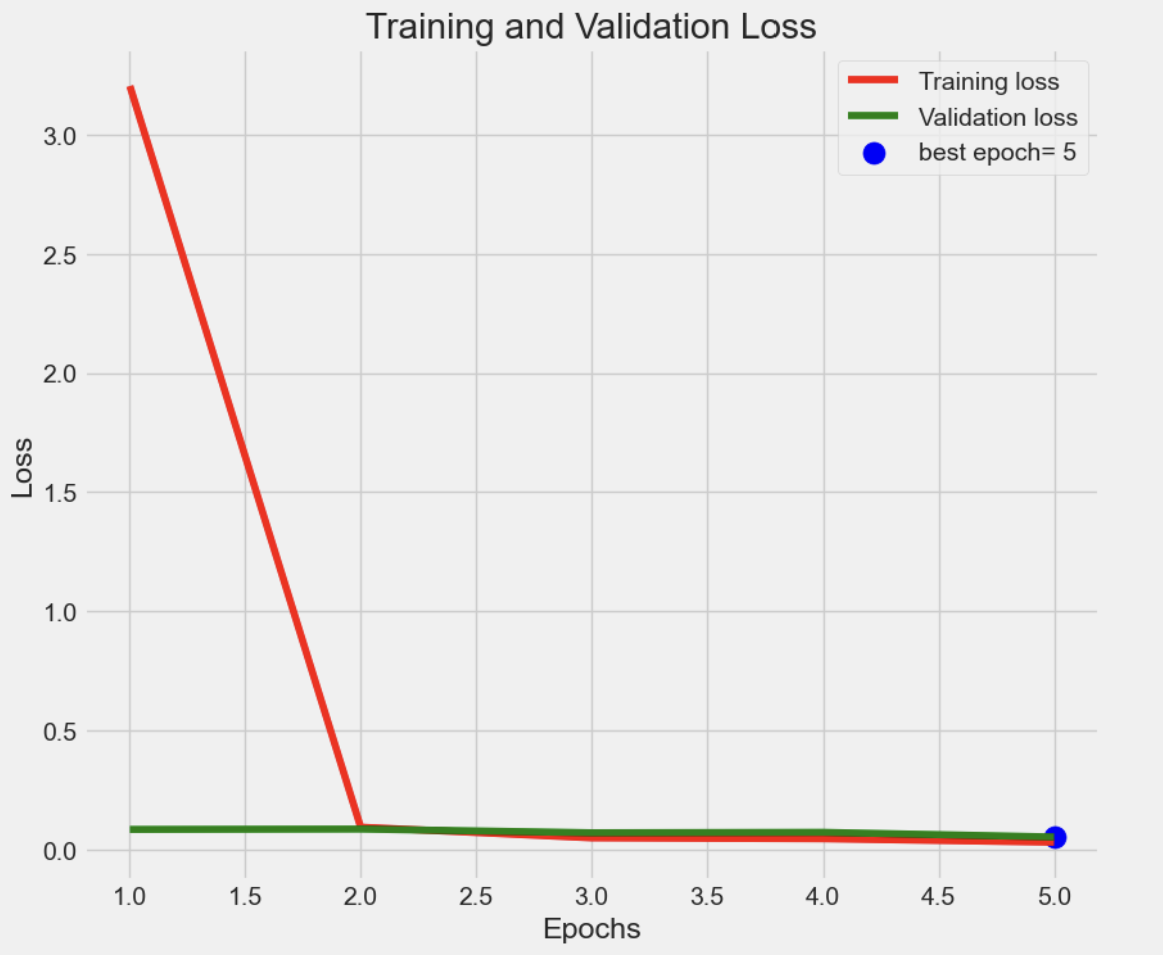
Upon detecting an anomaly, the Arduino Nano 33 BLE Sense can perform specific actions, such as triggering alerts, storing vehicle details, or communicating the information to a central system through BLE or other communication means. The camera module is attached to the back of a bike, and the microcontroller is stuck to the handle of the bike.

For image-based detection, the processing pipeline entails taking pictures with the camera sensor and feeding them into a TensorFlow Lite model that has already been trained. The model extracts features from the images and uses learned patterns to identify anomalies. It is optimized for real-time processing on the Nano 33 BLE Sense. [1]

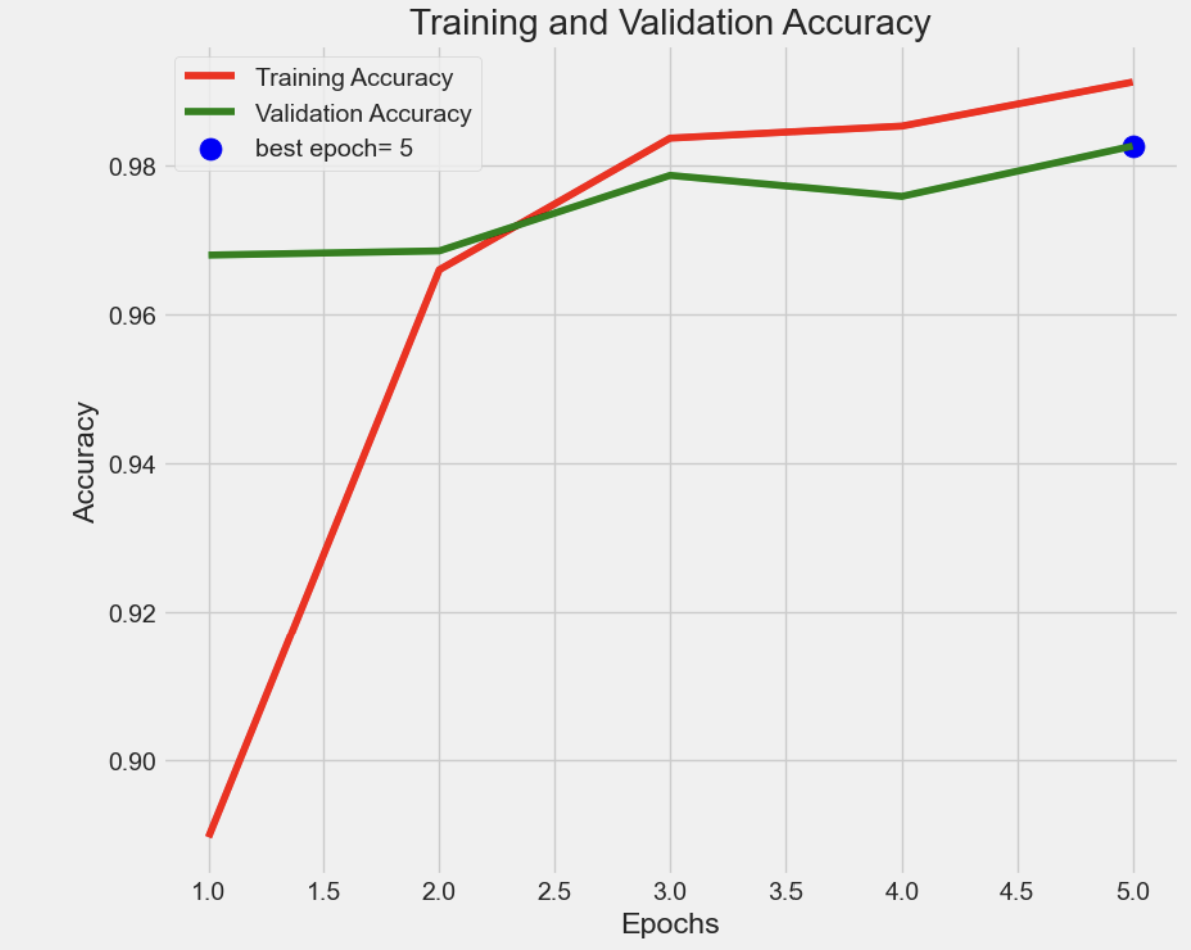
With the help of an Arduino Nano 33 BLE Sense camera sensor, this modified solution allows for real-time image detection, making it appropriate for applications that require visual data processing and anomaly identification in constrained IoT environments.

# Results

The Vehicle image classification model achieved an accuracy of 98.9% on the testing dataset, which indicates that the model is effective in classifying different types of animal sounds. The training accuracy and loss graphs show that the model was able to learn from the training data and improve its accuracy over time, while also reducing the loss function. The validation accuracy and loss graphs show that the model was not overfitting, as the accuracy on the validation set was also improving over time.

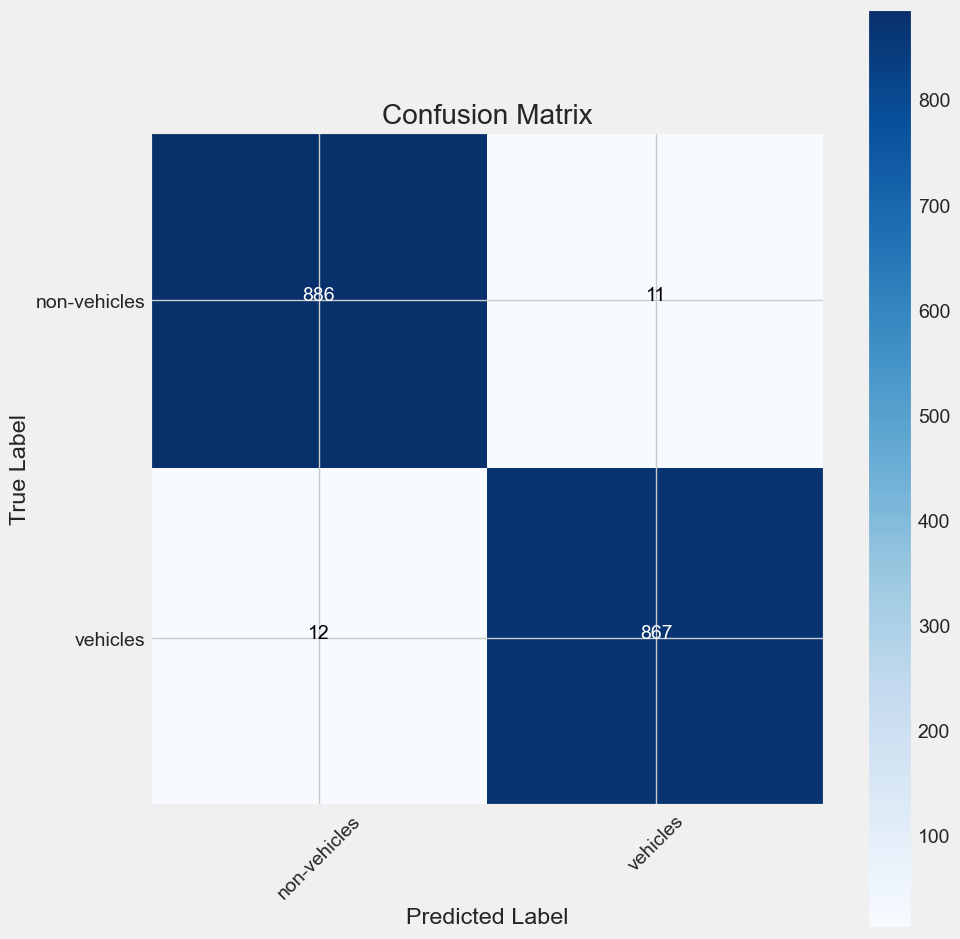


*Fig V. 1. Training and Validation Loss Graph*



*Fig V. 2. Training and Validation Accuracy Graph*

The confusion matrix provides more detailed information about the model’s performance for each class. It shows how many instances were classified correctly and incorrectly for each class. The accuracy values for each class can also be calculated using the confusion matrix. If the number of correctly classified instances is divided by the total number of instances for each class, the accuracy values for each class can be obtained.



*Fig. V. 3. Confusion Matrix for the Model.*

Overall, the high accuracy achieved by the Vehicle image Classification model as well as the visual evidence provided by the training and validation accuracy and loss graphs and the confusion matrix, demonstrate the effectiveness of the model for this task.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Training  Accuracy | Validation  accuracy | Testing  Accuracy |
| CNN | 0.989 | 0.9822 | 0.987 |

*Table V. 1. Performance Evaluation of CNN Model on Vehicle Image Classification.*

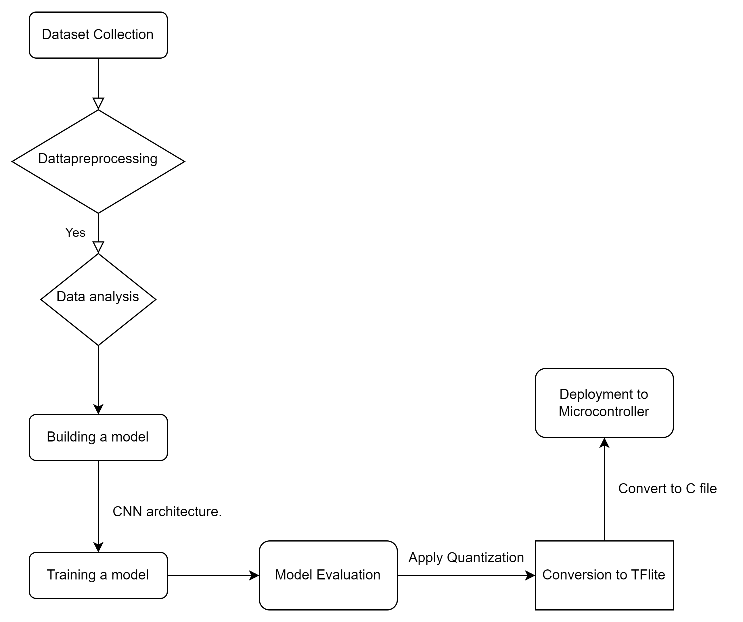
# Discussion

The Arduino Nano 33 BLE Sense comes with several considerations and constraints. Its ARM Cortex-M4 microcontroller, while serviceable, may face challenges with tasks demanding high computational power, and the confined program memory (256KB) and SRAM (32KB) may prove insufficient for projects with substantial code or data storage demands. The restricted number of digital and analog pins may present difficulties for applications requiring an abundance of sensors or peripherals. Power limitations, a singular BLE antenna design, and a compact form factor further limit its appropriateness for projects needing increased power, extensive connectivity, or a larger physical size. While it integrates diverse onboard sensors, it might not fulfill all sensing requirements, potentially necessitating external components. Moreover, its capacity to execute advanced machine learning models is constrained by its resources, and a scarcity of community support and documentation might impact troubleshooting and development in comparison to more established Arduino boards. [3]

When using a camera module with the Arduino Nano 33 BLE Sense, several constraints must be considered. The board's modest ARM Cortex-M4 microcontroller and limited program memory (256KB) and SRAM (32KB) may restrict its ability to handle complex image processing tasks or run advanced computer vision algorithms. Connectivity options are constrained as the board lacks built-in WiFi, potentially limiting the transfer of large image datasets. Compatibility issues may arise with certain camera modules, and power constraints could affect the feasibility of using power-intensive cameras or maintaining continuous operation. The board's limited input/output pins and small form factor might also impact the integration of additional sensors or peripherals. Real-time image processing may face challenges, and community support/documentation for specific camera modules could be limited. It's essential to be aware of these limitations and assess whether the Nano 33 BLE Sense is suitable for the specific requirements of camera-related projects. [2]

In the absence of a camera on the Arduino Nano 33 BLE Sense, one can develop an image classification program using alternative sensors. Choose a relevant onboard sensor, such as the accelerometer in the IMU, and create a program to gather data that reflects the necessary features for the classification task. If feasible, label the collected data and select an appropriate machine learning model. Train the model on a more powerful machine using the labeled dataset. Convert the trained model into a format compatible with the Arduino Nano 33 BLE Sense and TensorFlow Lite for Microcontrollers. Develop a program on the Arduino to execute real-time inference on the acquired sensor data. Test the program, making iterative adjustments to enhance accuracy, acknowledging that the effectiveness of the model relies on the richness of the data from the chosen sensor. Tailor the approach to meet specific project requirements.

Due to the absence of a camera, we weren’t able to perceive a real time data which could benefit a cyclist. This work can be taken for a future reference to build on a model which could be implemented using a camera that could result in clear image classification and the model could perceive image inference which could be tested in a practical scenario. The other issue faced during this project is that due to the size of the model obtained which is around 53MB after quantization the results couldn’t be uploaded to Arduino Nano 33 BLE Sense since the ram size is very small (245KB). Another future work could involve in finding an optimal solution to find suitable optimizers which could reduce the model and make it feasible enough to upload it to a microcontroller.



*Fig. VI. 1. Flow chart Diagram*

# Conclusion

The accuracy of the final model achieved an impressive performance of 98.9% on the test dataset, indicating the effectiveness of the methods and techniques used in this study. Additionally, the study provides valuable insights into the process of image data pre-processing and machine learning based vehicle image classification, and the methods and techniques presented here can be applied to other datasets, providing a starting point for further research in this area. Convolutional neural networks have proven to be a powerful tool in image classification, and their ability to automatically learn and extract meaningful features from images makes them well-suited for tasks such as image classification. This study highlights the potential of CNNs for audio classification tasks and shows that they can achieve high accuracy without the need for hand-crafted feature engineering

In summary, this study demonstrates the effectiveness of deep neural networks for the classification of vehicle image and provides valuable insights into the process of image data pre-processing and machine learning-based animal sound classification. The methods and techniques presented here can be applied to other image datasets and provide a starting point for further research in this area.

# References

|  |  |
| --- | --- |
| [1] | I. S. G. S. M. S. Pedro Andrade∗, An Unsupervised TinyML Approach Applied for Pavement Anomalies under the Internet of Intelligent Things, Federal University of Rio Grande do Norte - Natal/RN, Brazil: IEEE, 2021. |
| [2] | M. V. F. A. Mattia Antonini Miguel Pincheira, A TinyML approach to non-repudiable anomaly detection in extreme industrial environments, OpenIoT Research Unit, DI Center, Fondazione Bruno Kessler, Trento, Italy: IEEE, 2022. |
| [3] | A. I. M. I. T. i. S. H. f. R. A. Prevention, Angelos Zacharia, Konstantinos C. Giotopoulos, Dimitris Zacharia,Christos Karras, Ioanna Giannoukou, Decentralized Systems Computing Group, Computer Engineering and Informatics Department, University of Patras, Greece: IEEE, 2022. |
| [4] | Y. G. a. J. L. Zijie Chen, A Self-powered Sensing System With Embedded TinyML for Anomaly Detection, School of Information Science and Technology,ShanghaiTech University, Shanghai 201210, China: IEEE, 2023. |
| [5] | Z. X. Y. L. ,. J. P. Kun Fang, A Fall Detection using Sound Technology Based on TinyML, Technology and Metrology of Zhejiang Province, College of Information Engineering, China Jiliang University: IEEE, 2021. |