

PROJECT REPORT

EMBEDDED AI

MD HASIBUL HAQUE ZAHID 2302302

27TH MAY 2024

COURSE PROJECT REPORT

Introduction

Summary of Performed Tasks

This project aims to create a voice recognition system that can recognize the phrase "Weather in Turku" and "Weather in Vasaa" by utilizing the Edge Impulse platform and the Arduino Nano 33 BLE. Modern smart devices must have voice recognition in order to allow hands-free interaction and control. By integrating voice recognition capabilities into an embedded system, this project aims to demonstrate the feasibility and performance of deploying machine learning models on resource-constrained hardware. The tasks performed in this project include data collection, preprocessing, data splitting, model training, deployment, and testing. To ensure that the accuracy and efficiency of the system for identifying the voice patterns were satisfactory, handling each task carefully was of paramount importance.

Edge Impulse is an advanced development platform tailored for machine learning on edge devices. It offers a full range of tools for gathering information, creating machine learning models, and implementing these models on different hardware platforms. Edge Impulse makes it easier for developers to construct complex AI-driven solutions without requiring a thorough understanding of machine learning by streamlining the entire machine learning workflow.

The Arduino Nano 33 BLE is a compact and robust board that supports Bluetooth Low Energy (BLE). It is appropriate for many embedded applications because it has the Nordic nRF52840 microcontroller and several sensors and interfaces. The hardware platform used in this project to record audio and run the trained machine learning model for real-time keyword classification was the Arduino Nano 33 BLE.

The initial step towards getting started was Data collection. This involved recording various audio samples containing the target phrase with different city names. The resulting datasets served as the backbone and provided us with foundation for robust model training. Preprocessing was done using Edge Impulse, which involved sample splitting, noise reduction and feature extraction to prepare the audio data for training. Notably, the most complicated challenge here was to split the sample audio data carefully as in some cases, poor-quality samples could adversely affect the data sets when training the model.

Edge Impulse platform was used for model training leveraging the capabilities for creating a tailored neural network model for the specific task of recognizing the phrase "Weather in Vasaa" And "Weather in Turku." We created an impulse with a window size of 2009 milliseconds and utilized Mel-Frequency Cepstral Coefficients (MFCC) for feature extraction. MFCCs are crucial for capturing the essential characteristics of speech signals, making them highly effective for audio and speech recognition tasks. Despite initial challenges with limited data and high noise levels, iterative improvements led to a significant enhancement in model performance. The trained model was then deployed onto the Arduino Nano 33 BLE using TensorFlow Lite for microcontrollers, ensuring that the model effectively functions on the constrained resources of the embedded device.

Through this project, we aimed to demonstrate the feasibility and effectiveness of deploying machine learning models on edge devices for real-time applications. The following sections detail the performed tasks, task division, results, and conclusions drawn from our work.

Implementation Environment and Platform

The implementation of the voice recognition system was conducted using a combination of hardware and software components specifically chosen to meet the project's requirements. The primary hardware platform used was the Arduino Nano 33 BLE, a compact and powerful microcontroller board equipped with Bluetooth Low Energy (BLE) capabilities. This board is well-suited for embedded AI applications due to its relatively high processing power and low energy consumption.

For the software environment, Edge Impulse was selected as the primary platform for data preprocessing and model training. Edge Impulse provides a comprehensive suite of tools for developing machine learning models for embedded systems, including data acquisition, preprocessing, and model training. Its user-friendly interface and robust feature set make it an ideal choice for quickly developing and deploying machine learning models on embedded devices.

The combination of these tools and platforms enabled the successful development and deployment of the voice recognition system, demonstrating the potential of embedded AI to bring advanced functionalities to low-power, resource-constrained devices.

Performed Tasks

Design and Implementation

The project was executed in several stages, starting with data collection and extending through model deployment and testing. The design and implementation process involved both hardware and software components, each contributing to the successful development of the voice recognition system.

Data Collection: Initially, we focused on gathering a comprehensive dataset of audio samples. This stage proved to be challenging due to some significant mistakes in the initial data collection. We mistakenly sampled audio for only city names instead of a whole sentence which rendered our first model useless. Recognizing the importance of high-quality data, we had to restart the data collection process from scratch. Despite these setbacks, we collected an extensive 32 minutes of natural raw data, with 16 minutes dedicated to each of the target cities, Vaasa and Turku. The dataset included thousands of samples, ensuring a robust training set. Additionally, we recorded approximately 27 minutes of background noises to simulate real-life environments and improve the model's robustness.



Fig: Dataset

Preprocessing: The collected audio samples were then preprocessed using Edge Impulse. This involved noise reduction, feature extraction, and other techniques to enhance the quality and consistency of the data. Using Edge Impulse, we created an impulse with a window size of **2009** milliseconds.

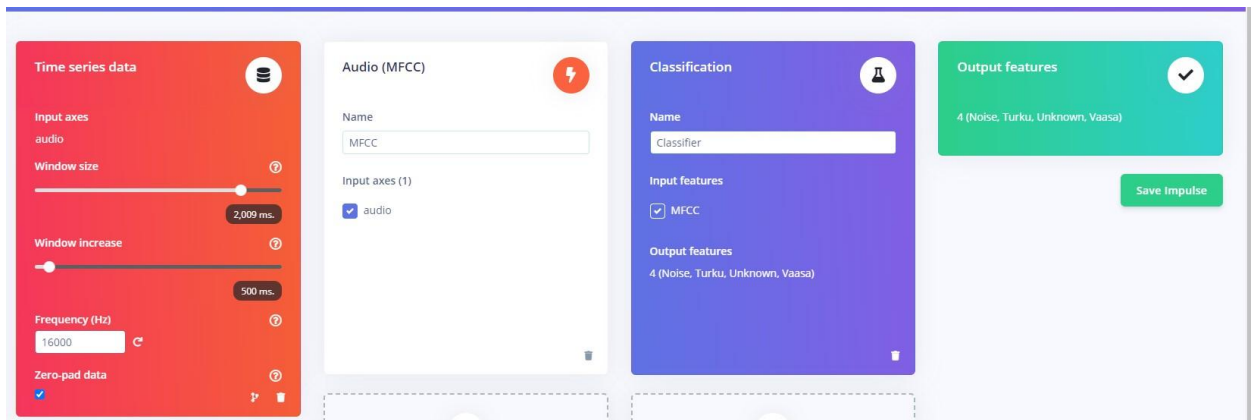


Fig: Creating Impulse

MFCC (Mel-Frequency Cepstral Coefficients): MFCC was used as the feature extraction method. MFCCs are a representation of the short-term power spectrum of a sound. They are obtained by taking a signal's Fourier transform, translating the spectrum's powers onto a Mel scale, calculating the powers' logarithm, and then using the discrete cosine transform. This method is widely used for audio and speech recognition applications because it effectively catches the key elements of speech signals.

Model Training: Using the preprocessed data, we trained a neural network model on the Edge Impulse platform. We used the collected data to train a classifier on the Edge Impulse platform. Both of us were responsible for defining the task and managing the training process. The initial model did not perform well due to the limited amount of real-life data and the high proportion of noise and unknown data. However, through iterative improvements and data augmentation, the model's performance improved significantly. The refined model achieved an accuracy of 95% with a loss of 0.18.



Fig: Training Accuracy

Test data					Classify all
Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.					
SAMPLE NAME	EXPECTED OUTCOME	LENGTH	ACCURACY	RESULT	
Turku_Zahid.s12	Turku	2s	100%	1 Turku	
Turku_Zahid.s11	Turku	2s	100%	1 Turku	
Turku_Zahid.s10	Turku	2s	100%	1 Turku	
Turku_Zahid.s9	Turku	2s	100%	1 Turku	
Turku_Zahid.s8	Turku	2s	100%	1 Turku	
Turku_Zahid.s7	Turku	2s	100%	1 Turku	
SAMPLE NAME	EXPECTED OUTCOME	LENGTH	ACCURACY	RESULT	
Vaasa_Zahid_1.s...	Vaasa	2s	100%	1 Vaasa	
Vaasa_Zahid_1.s9	Vaasa	2s	100%	1 Vaasa	
Vaasa_Zahid_1.s8	Vaasa	2s	100%	1 Vaasa	
Vaasa_Zahid_1.s7	Vaasa	2s	100%	1 Vaasa	

Fig: Test Data

Deployment: After training the model, we deployed it onto the Arduino Nano 33 BLE using TensorFlow Lite for microcontrollers. This step involved converting the trained model into a format suitable for the Arduino Nano 33 BLE's limited computational resources. The deployment also included writing the necessary code to capture audio input, process it, and perform inference using the trained model.

Version your project to store all data, configuration, intermediate results and final models. You can revert back to earlier versions of your impulse by importing a revision into a new project.									
VERSION	CREATED	USER	DESCRIPTION	TRAINING ACCURA...	TEST ACCURACY	NO. OF SAMPLES	PUBLIC		
2	May 24 2024, 1...	MD HASIBUL H...	We Now we set windows size 2 and got more better result	95%	95%	1h 12m 3s	-		⋮
1	May 13 2024, 0...	MD HASIBUL H...	This is our 2nd Commit, First we only took the data set for Turku and Vaasa, Unfortunately we didn't do the versioning on that time	94%	88%	45m 31s	⊙		⋮

Fig: Deployment

Testing: The final stage involved extensive testing to evaluate the system's performance. We tested the system under various conditions to assess its accuracy, latency, and resource utilization. This stage was crucial for identifying any issues and ensuring that the voice recognition system operated reliably in real-world scenarios.

Project Showcase and Iterations:

In the project showcase, we initially demonstrated the model with a window size of 1, which was later optimized for better performance with the correct window size. We also collected audio samples from our friends as well which helped us get real human data from 6 humans.

In an earlier attempt, we achieved 94% accuracy but initially misunderstood the project requirements, focusing only on the audio data for Vaasa and Turku, which consumed significant amount of time and data.

Task Division and Responsibilities

Data Collection: - Initially faced challenges but successfully collected extensive and high-quality audio samples, including 32 minutes of targeted phrases and 20 minutes of background noise. Sajjad Focused on collecting data for Turku while Zahid focused on Vaasa.

Preprocessing: - Managed the preprocessing of audio samples, ensuring clean and consistent data for model training.

Model Training: - Trained the neural network model on Edge Impulse, achieving a high accuracy rate for recognizing the target phrase. The first attempt was by Sajjad and then later was attempted by Zahid.

Deployment: - Deployed the trained model onto the Arduino Nano 33 BLE, overcoming the constraints of the hardware.

Testing: - Conducted thorough testing and validation, providing insights into the system's performance and areas for improvement.

Results

Accuracy: In project showcasing we used 1 sec window size then we deployed the project again with 2 sec windows size. The final model achieved an accuracy of 95% with a loss of 0.18.

```
recording done
Predictions (DSP: 256 ms., Classification: 8 ms., Anomaly: 0 ms.):
#Classification results:
  Noise: 0.000000
  Turku: 0.996094
  Unknown: 0.000000
  Vaasa: 0.000000
Starting inferencing in 2 seconds...
```

Fig: Performance Checking for Turku

```
#Classification results:
  Noise: 0.000000
  Turku: 0.000000
  Unknown: 0.000000
  Vaasa: 0.996094
Starting inferencing in 2 seconds...
```

Fig: Performance Checking for Vaasa

```
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 256 ms., Classification: 8 ms., Anomaly: 0 ms.):
#Classification results:
  Noise: 0.984375
  Turku: 0.003906
  Unknown: 0.011719
  Vaasa: 0.000000
```

Fig: Noise

```
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 256 ms., Classification: 8 ms., Anomaly: 0 ms.):
#Classification results:
  Noise: 0.000000
  Turku: 0.000000
  Unknown: 0.992187
  Vaasa: 0.007813
Starting inferencing in 2 seconds...
Recording...
```

Fig: Unknown

Performance Improvement: The optimized window size led to better results than the initial attempt which had 94% accuracy.

Conclusion

Interpretation of Results

In conclusion, this project successfully demonstrated the feasibility and effectiveness of deploying machine learning models on edge devices for real-time voice recognition applications. Against the initial challenges with data collection, we managed to gather extensive high-quality audio samples and background noises and those were essential for robust model training. We preprocessed the data, extracted features with MFCC using Edge Impulse, and trained a neural network model to recognize the phrases "Weather in Turku" and "Weather in Vaasa." The model was then deployed on the Arduino Nano 33 BLE using TensorFlow Lite for microcontrollers, achieving a final accuracy of around 95% with a loss of 0.18. The model's reliability and efficiency was confirmed after being tested under various environments, highlighting the potential of embedded AI for advanced functionalities in low-power, resource-constrained devices.

Proposed Improvements

- Noise Robustness: Enhance preprocessing to better handle noisy environments.
- Diverse samples: Extend the system to recognize a broader set of phrases.

TensorFlow Lite Project

Q1: Will using a camera sensor (like OV7670) make the acquisition of data more complex? Tell us what you think will change (and what won't) compared to what we do here very briefly.

We started by using an online site to create a JPG image, then we converted it into a C header file using a Python Script. Which then we then we used Arduino to run it into our model. As for using a camera sensor for acquisition of data will be more complex. Some of the changes would be:

Data Acquisition Complexity:

- **Hardware Integration:** The wiring and configuration required to integrate a camera sensor with a microcontroller, like the Arduino Nano 33 BLE, is more intricate. It takes more work to make sure the camera is powered on and interfaced with the microcontroller.
- **Image Capture:** Managing the camera's interface, controlling real-time data streams, and perhaps adjusting to various resolutions and formats are all necessary while taking pictures with the camera sensor. Compared to working with pre-existing image files, this technique is more complicated.
- **Binary Image Conversion:** Working directly with raw picture data in binary form, which is what most camera sensors produce, can be challenging. Another level of complexity is added when these binary images are transformed into a format that can be used with machine learning models.

Data Preprocessing:

- **Image Conversion:** It is necessary to convert the camera's raw photos into a format that may be used, such as JPG. This calls for programmatic handling of image processing operations including scaling, grayscale conversion, and possibly noise removal.
- **Creating Usable Data:** We decided to produce our photographs in JPG format in order to streamline our workflow. These photos were then transformed into a C header file using a Python script, making it simple to incorporate them into our embedded system project. By using this method, handling raw binary data straight from the camera sensor is avoided, which can be complicated.

Storage and Bandwidth:

- **Data Storage:** More storage space is needed to handle raw image data from a camera than to handle JPG files that have already been processed. Handling the huge data quantities involved requires effective data management.
- **Bandwidth:** Real-time data transmission from the camera to the CPU might put a burden on available bandwidth and necessitate effective data transfer protocols.

Some of the things that will remain unchanged would be:

Model Input Requirements:

Our model still requires the input data in the same format. The photographs must match the shape and format that our model expects, such as 28x28 grayscale, whether they were taken by the camera or were already converted.

Core Processing Logic:

The core logic of our machine learning model and the inference process remains unchanged. We still use TensorFlow Lite to load the model, prepare the input data, run inference, and interpret the results.

Conclusion:

The fundamental needs for preparing data for the machine learning model and carrying out inference stay the same, even though incorporating a camera sensor like the OV7670 adds complexity to hardware integration, real-time data handling, and preprocessing. We simplify the process and eliminate the extra challenges that come with processing raw binary data straight from the camera by generating our images in JPG format and utilizing a Python script to convert them to a C header file.

Q2: How did you decide to transform the set of images for running inferences later? Does the flash memory limitation of the board affect your decision?

To transform the set of images for running inferences, we opted for multiple image transfer and resizing. The flash memory limitation of the Arduino Nano BLE 33, which has only 1 MB of space, significantly influenced this decision. Large images would consume excessive memory, potentially affecting model performance or causing it to fail. By resizing the images, we ensured they fit within the available memory, allowing the model to operate efficiently and deliver accurate results.

Q3: How did you decide to prune the model graph? Was it useful and were there any trade-offs?

We pruned the model graph to reduce its size for better compatibility with the available memory. Initially, without pruning, the model was too large. After experimenting with 3x and 10x pruning, we successfully reduced the model size to 683 KB. This helped fit the model within the memory constraints of our board. Despite potential trade-offs in accuracy and performance, the pruned model remained suitable for our application. And yes, it is useful there is not so much tradeoff only a small amount of performance is dropped.

We considered the trade-offs between model size reduction and possible accuracy loss while determining how to prune the model graph. Pruning was chosen since it was known that smaller models could result in faster inference times and a smaller memory footprint—two important benefits for deployment on devices with limited resources, such as the Arduino Nano 33 BLE Sense.

To summarize, the choice to trim the model graph resulted in a smaller model size with similar accuracy. In this instance, there was little compromise between accuracy loss and model size reduction, which makes pruning a useful method for optimizing models for edge device deployment.

Q4: What quantization mechanism did you use? How did the fact that the model needs to run on the Arduino Nano influence your decision?

We used post-training quantization in the TensorFlow Lite default optimization for quantization. The model's weights and activations are quantized using this technique, which converts 32-bit floating-point quantities into 8-bit integers. The model's size and memory footprint are greatly decreased using this method, which makes it more appropriate for deployment on devices with limited resources, such as the Arduino Nano 33 BLE.

The requirement for the model to operate on the Arduino Nano played a role in the decision to employ post-training quantization. Memory and processing power are among the computational resources that are restricted on this microcontroller. The model can be made to fit within the limitations of the Arduino Nano while still retaining a respectable level of inference accuracy by quantizing it in order to minimize its size and memory use. The test accuracy of the trimmed and quantized TensorFlow Lite model was used to assess the quantization outcomes. In comparison to the pruned model, this drop was considered acceptable given the large reduction in model size.

We were able to considerably shrink the model's size while preserving a respectable level of inference accuracy by utilizing post-training quantization, which guaranteed that the model would work with the limitations of the Arduino Nano 33 BLE.

Q5: Was both pruning and quantization necessary for the task? Could we have skipped either or both and still been able to use the model?

Although they had slightly different uses, pruning and quantization were both useful methods for refining the model for the Arduino Nano 33 BLE Sense.

The main goal of pruning was to reduce the size of the model by eliminating extraneous connections between neurons, which would lower the amount of memory and computing complexity needed. We were able to drastically shrink the model's size without sacrificing any accuracy by trimming it. Pruning might not have been enough on its own, though, given the Arduino Nano's memory limitations.

However, in order to further minimize the model's size and memory footprint, quantization was necessary. Through quantization, the weights of the model were converted from 32-bit floating-point numbers to 8-bit integers, resulting in a significant reduction in the memory needed to hold the model parameters. A representative_dataset was provided as 32-bit floating-point values. Although the processor is capable of doing, we could correct it in the activation layer using an 8-bit integer. In order to make sure the model could fit inside the Arduino Nano's constrained memory resources, this size reduction was essential.





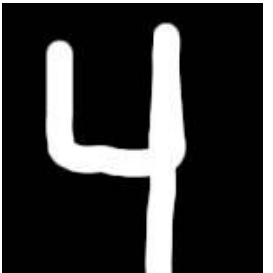
Even if the reduced model might have been used without quantization, the size of the final model might still have been too big to fit inside the Arduino Nano's limitations. Likewise, the intended decrease in model size might not have been possible with quantization alone.

To sum up, in order to optimize the model for deployment on the Arduino Nano 33 BLE Sense, both pruning and quantization were required. Each technique played a complementary role in reducing the

model's size and memory footprint, ultimately enabling its successful deployment on the microcontroller.

Q6: Is the output vector in your program same as running the flites model in python?

There are slightly changed and one major changed in number 6

Number	Model	Web
	<pre> 0Prdiction 0.996094 1Prdiction 0.000000 2Prdiction 0.000000 3Prdiction 0.000000 4Prdiction 0.000000 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>	<pre> Output data: [[0.99609375 0. 0. 0. 0. 0. 0. 0. 0.]] Predicted index: 0 </pre>
	<pre> 0Prdiction 0.000000 1Prdiction 0.996094 2Prdiction 0.000000 3Prdiction 0.000000 4Prdiction 0.000000 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>	<pre> 0Prdiction 0.000000 1Prdiction 0.996094 2Prdiction 0.000000 3Prdiction 0.000000 4Prdiction 0.000000 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>
	<pre> 0Prdiction 0.000000 1Prdiction 0.000000 2Prdiction 0.996094 3Prdiction 0.000000 4Prdiction 0.000000 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>	<pre> 0Prdiction 0.000000 1Prdiction 0.000000 2Prdiction 0.996094 3Prdiction 0.000000 4Prdiction 0.000000 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>
	<pre> 0Prdiction 0.000000 1Prdiction 0.003906 2Prdiction 0.000000 3Prdiction 0.996094 4Prdiction 0.000000 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>	<pre> 0Prdiction 0.000000 1Prdiction 0.000000 2Prdiction 0.000000 3Prdiction 0.996094 4Prdiction 0.000000 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>
	<pre> 0Prdiction 0.000000 1Prdiction 0.000000 2Prdiction 0.000000 3Prdiction 0.000000 4Prdiction 0.996094 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>	<pre> 0Prdiction 0.000000 1Prdiction 0.000000 2Prdiction 0.000000 3Prdiction 0.000000 4Prdiction 0.996094 5Prdiction 0.000000 6Prdiction 0.000000 7Prdiction 0.000000 8Prdiction 0.000000 9Prdiction 0.000000 </pre>



```
0Prdiction 0.000000
1Prdiction 0.000000
2Prdiction 0.000000
3Prdiction 0.000000
4Prdiction 0.000000
5Prdiction 0.996094
6Prdiction 0.000000
7Prdiction 0.000000
8Prdiction 0.000000
9Prdiction 0.000000
```

```
0Prdiction 0.000000
1Prdiction 0.000000
2Prdiction 0.000000
3Prdiction 0.000000
4Prdiction 0.000000
5Prdiction 0.996094
6Prdiction 0.000000
7Prdiction 0.000000
8Prdiction 0.000000
9Prdiction 0.000000
```



0Prdiction	0.000000
1Prdiction	0.000000
2Prdiction	0.000000
3Prdiction	0.000000
4Prdiction	0.000000
5Prdiction	0.000000
6Prdiction	0.996094
7Prdiction	0.000000
8Prdiction	0.000000
9Prdiction	0.000000

0Prdiction 0.000000
1Prdiction 0.000000
2Prdiction 0.000000
3Prdiction 0.000000
4Prdiction 0.000000
5Prdiction **0.878906**
6Prdiction **0.121094**
7Prdiction 0.000000
8Prdiction 0.000000
9Prdiction 0.000000



```

9Prdiction 0.000000
0Prdiction 0.000000
1Prdiction 0.000000
2Prdiction 0.000000
3Prdiction 0.000000
4Prdiction 0.000000
5Prdiction 0.000000
6Prdiction 0.000000
7Prdiction 0.996094
8Prdiction 0.000000
9Prdiction 0.000000

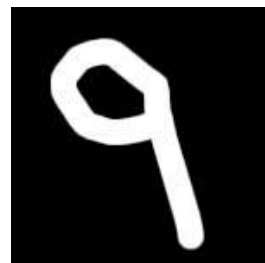
```

0Prdiction 0.000000
1Prdiction 0.**00390**
2Prdiction 0.**152344**
3Prdiction 0.000000
4Prdiction 0.000000
5Prdiction 0.000000
6Prdiction 0.000000
7Prdiction 0.**84375**
8Prdiction 0.000000
9Prdiction 0.000000



```
9Prdiction 0.0000000
0Prdiction 0.0000000
1Prdiction 0.0000000
2Prdiction 0.0000000
3Prdiction 0.0000000
4Prdiction 0.0000000
5Prdiction 0.0000000
6Prdiction 0.0000000
7Prdiction 0.0000000
8Prdiction 0.9960944
9Prdiction 0.0000000
```

```
0Prdiction 0.000000
1Prdiction 0.000000
2Prdiction 0.000000
3Prdiction 0.000000
4Prdiction 0.000000
5Prdiction 0.000000
6Prdiction 0.000000
7Prdiction 0.000000
8Prdiction 0.996094
9Prdiction 0.000000
```



```
0Prdiction 0.000000
1Prdiction 0.000000
2Prdiction 0.000000
3Prdiction 0.000000
4Prdiction 0.046875
5Prdiction 0.000000
6Prdiction 0.000000
7Prdiction 0.000000
8Prdiction 0.000000
9Prdiction 0.953125
```

```
0Prdiction 0.000000
1Prdiction 0.000000
2Prdiction 0.000000
3Prdiction 0.000000
4Prdiction 0.000000
5Prdiction 0.000000
6Prdiction 0.000000
7Prdiction 0.000000
8Prdiction 0.000000
9Prdiction 0.996094
```

Q7: How did the model perform on the test images?

The model perform on the test images are quite good almost we got 98 % accuracy and the Screenshot below shows the accuracy of the model when operating on the test images with **thin bordered** lines. -

```
0Prdiction 0.011719
1Prdiction 0.593750
2Prdiction 0.343750
3Prdiction 0.031250
4Prdiction 0.003906
5Prdiction 0.003906
6Prdiction 0.007813
7Prdiction 0.000000
8Prdiction 0.003906
9Prdiction 0.003906
```

We got much better results once we used the bold drawing images for our model.

- **Have you learned anything new?**

Yes, we have learned several new concepts and techniques during this project. We gained hands-on experience with Edge Impulse, a powerful platform for developing machine learning models for edge devices. We also learned how to use the Arduino Nano 33 BLE, a compact microcontroller board ideal for embedded AI applications. Additionally, we explored Mel-Frequency Cepstral Coefficients (MFCC) for feature extraction, and techniques such as pruning and quantization for model optimization. Finally, we learned the process of deploying machine learning models into embedded devices using TensorFlow Lite for microcontrollers.

- **Did anything surprise you?**

We were surprised to **successfully** executing both models. We were honestly not expecting that.

- **Did you find anything challenging?**

Yes, pruning and quantization were particularly challenging aspects of this project. These optimization techniques were difficult to implement, requiring detailed knowledge and careful tuning to maintain model accuracy while reducing size and computational requirements. We received invaluable assistance from our TA, Raisul, and other fellow students, which helped us overcome these challenges.

- **Did you find anything satisfying?**

Seeing the audio and picture detection models implemented successfully gave us great satisfaction. It was also quite satisfying to achieve successful quantization and pruning for model optimization. These achievements gave a sense of satisfaction in conquering the technical obstacles and showed how embedded AI approaches might be used in real-world applications.

Github Link: https://github.com/it-teaching-abo-akademi/course-project-report-source-code-Xahidian/tree/main/Embedded_Project

REFERENCES : GENERATIVE AI WAS USED IN SOME PLACES FOR BETTER ORGANIZATION OF SENTENCES

THANK YOU
