TinyML Project - Project 4: Audio-Based Wildlife Monitoring

Vasiliki Menagia - vasilikimenagia@gmail.com
Ioannis Vettas - johnvettas7@gmail.com
Argyris Theofilopoulos - aris.theofil@gmail.com

Department of Computer and Systems Sciences

Degree project 7.5 HE credits
Computer and Systems Sciences
Degree project at the master level
Autumn/Spring term 2025
Supervisor: Rahim Rahmani Chianeh

Stockholm University

Abstract:

In the bounds of this project we were called to train and deploy a small Convolutional Neural Network (CNN) model that receives an audio sample and classifies which animal this sound represents. Our added challenge was to find a way to train the model to ignore the rotor noise of the UAV.

For this task the first requirement was to collect enough training data, which we did by searching the internet for open source audio datasets. Afterwards, that data was pre-processed (added rotor noise, normalized, standardized and turned into spectrograms) and used to train a CNN Image Classification AI model, that categorizes animal sounds in 9 classes: dog, cat, crow, elephant, horse, lion, parrot, peacock, sparrow. The model was trained using Edge Impulse Studio's image classifier model, and made to operate on a Raspberry Pi 4, which is what it was tested on.

Overall, we achieved an accuracy of 84.64% with the final quantized (int8) model. After deploying the model and performing inferences on a Raspberry Pi, we observed that it was able to classify the animal sounds with a high level of confidence and consistency. The results indicate that the model performs reliably even under real-world conditions, demonstrating both its effectiveness and efficiency on low-power hardware.

Keywords: TinyML, Quantization, Wildlife, classification, AI

Synopsis

Background:

Sound classification using deep learning has gained attention in recent years due to its wide range of applications, including wildlife monitoring, smart environments, and robotics. Convolutional Neural Networks (CNNs) have proven effective in recognizing patterns in audio spectrograms, making them suitable for classifying animal sounds.

Problem Statement:

Manual wildlife acoustic monitoring is time-consuming. A UAV can cover large areas, but transmitting raw audio is impractical. An edge AI model can classify sounds onboard, storing only detections.

Research Question:

How can a lightweight CNN model be trained and optimized to accurately classify animal sounds despite interference from UAV rotor noise?

Method:

Open-source audio datasets were collected and preprocessed by adding simulated rotor noise, normalizing, standardizing, and converting the sounds into spectrograms. A CNN image classification model was then trained using Edge Impulse Studio and quantized to int8 for efficient deployment. The final model was tested on a Raspberry Pi 4.

Result:

The quantized model achieved an accuracy of 84.64% and performed consistently well during real-time inference on the Raspberry Pi, confidently classifying animal sounds into nine categories: dog, cat, crow, elephant, horse, lion, parrot, peacock, and sparrow.

Discussion:

The project demonstrated that a compact CNN can achieve high accuracy in animal sound classification, even in noisy environments. These results highlight the potential of deploying deep learning models on low-power edge devices for real-world applications in environmental monitoring and UAV-based data collection.

Acknowledgements

We would like to thank Rahim Rahmani for his valuable assistance throughout this project. We are also grateful to the teaching assistant Qi Dang for providing guidance and useful insights that supported our work.

Additionally, we thank the University of Piraeus (UNIPI) and Stockholm University (SU) for enabling our participation in the Erasmus program, and SU for accepting us.

The support from all of the above greatly contributed to the success of this project.

Table of Contents

1...Introduction……………………………….……………………………4

2…Extended Background and Related Work……………………………..6

3...Methodology………………………………...……………………….9

4...Testing and Implementation………………………….………………11

5…Results……………………………………………….………………...15

6…Discussion……………………………..………………………17

List of Figures

1...Model’s training results……………………………………………………………11
2…Model’s testing results……………………………………………………………12
3…Unoptimized Model’s results……………………………………………………..13
4…Overall Benchmarks………………………………………………………………14
5…Cat inference……………………………………………………………………15
6…Lion inference…………………………………………………………………..16

List of Tables

Table 1: Benchmarks for each class 14

Chapter 1 - Introduction

Background:

The classification of audio signals using machine learning has become an important area of research due to its wide-ranging applications in fields such as wildlife monitoring, smart environments, and robotics. Audio-based classification systems allow for the automated identification of sounds, save time and reduce human error. Convolutional Neural Networks (CNNs) have proven particularly effective for sound recognition tasks when audio signals are transformed into spectrograms. Spectrograms provide a visual representation of the frequency content over time, allowing CNNs to detect patterns that correspond to specific sound sources.

In this project, the focus is on classifying animal sounds captured in real-world environments. A unique challenge arises when recordings are collected from an Unmanned Aerial Vehicle (UAV), as rotor noise can interfere with the quality of the audio. Developing a robust model that can accurately identify animal sounds while ignoring such environmental noise is essential for practical applications in aerial wildlife monitoring and low-power edge computing devices.

This chapter sets the stage for understanding the motivation, scope, and techniques employed in the project, providing a foundation for the problem statement and research objectives that follow.duce human error, and enable real-time decision-making.

Problem Statement:

Manual wildlife acoustic monitoring is time-consuming. A UAV can cover large areas, but transmitting raw audio is impractical. An edge AI model can classify sounds onboard, storing only detections.

Aim:

The aim of this project is to design, train, quantize and deploy a small Convolutional Neural Network (CNN) capable of accurately classifying animal sounds from audio recordings. A key objective is to make the model robust against environmental noise, particularly the rotor noise generated by a UAV during data collection, while ensuring it can run efficiently on a low-power edge device such as a Raspberry Pi.

Research Questions:

How can a CNN model be trained to reliably classify animal sounds despite the presence of UAV rotor noise?

What preprocessing and data augmentation techniques are most effective for enhancing model performance?

- ➔ Which training settings are most ideal to ensure high accuracy on a quantized (int8) model?

Chapter 2 - Extended Background and Related Work

Extended Background:

TinyML is a rapidly growing subfield of Machine Learning that focuses on developing and deploying models capable of running on devices with highly constrained computational resources, such as microcontrollers and single-board computers like Arduino and Raspberry Pi. This approach enables machine learning inference directly on the edge, without the need for continuous internet connectivity or cloud-based computation.

The main advantage of TinyML lies in its ability to deliver real-time, low-latency decision-making while maintaining low power consumption and preserving data privacy. These qualities make it ideal for Internet of Things (IoT) applications, including environmental monitoring, industrial automation, and autonomous systems such as drones. However, the limited memory, processing power, and energy capacity of embedded devices introduce unique challenges in model design, optimization, and deployment.

In the context of our project, the use of TinyML was particularly suitable. The Raspberry Pi 4, serving as the onboard computing unit of an unmanned aerial vehicle (UAV), needed to perform real-time classification tasks without relying on a stable Wi-Fi connection. Implementing a TinyML model allowed the UAV to process sensor or visual data locally and make rapid, independent decisions during flight. This reduced latency, improved autonomy, and removed dependency on remote servers—essential factors for effective edge-based aerial computing.

Related Work:

Programming and deploying AI models on resource-constrained devices is a relatively young and experimental field that has gained significant momentum over the past decade. With the rapid growth of TinyML frameworks and optimized neural network architectures, numerous projects have explored implementing classification models on embedded systems such as Raspberry Pi and Arduino platforms. These efforts have demonstrated the feasibility of performing real-time image, audio, and sensor data classification directly on edge devices without relying on cloud computation.

Similar projects have focused on lightweight convolutional neural networks (CNNs) and model compression techniques to achieve acceptable accuracy within limited hardware constraints. However, many of these implementations operate in stable, noise-free environments or rely on preprocessed datasets.

In contrast, our project presented an additional challenge: deploying a classification model on a UAV platform, where environmental factors such as rotor noise could significantly interfere with sensor data. This required us to design and train our model to effectively filter or ignore such noise, ensuring accurate and reliable classification during flight operations.

Quantization in TinyML:

Quantization is a model optimization technique used in TinyML to reduce model size and improve inference speed on edge devices. Its main goal is to enhance memory efficiency, latency, and power consumption without significantly affecting accuracy.

In Int8 quantization, model weights and activations are converted from 32-bit floating-point (float32) to 8-bit integers (int8). This reduces memory usage and enables faster, more energy-efficient integer arithmetic.

The process includes three stages:

1. **Calibration:** Representative data determine the value ranges of weights and activations.

2. **Conversion:** Data are mapped to int8 format with scaling to preserve accuracy.

3. **Deployment:** The quantized model (e.g., in TFLite format) runs efficiently on embedded devices.

For example, an int8 TFLite model on a Raspberry Pi 4 mounted on a UAV can classify animal sounds in real time without cloud support. Despite lower numerical precision, well-optimized quantized models often achieve accuracy close to their float32 counterparts.

Performance Metrics and Benchmarking

Performance evaluation in TinyML relies on standardized metrics and benchmarks to assess model efficiency on resource-constrained devices. These metrics provide a foundation for fair comparison, reproducibility, and continuous improvement in embedded machine learning systems.

Key performance metrics for embedded inference include:

- **Latency:** The time required for a model to process an input and produce an output, directly affecting real-time responsiveness.
- **Memory Usage:** The total RAM and flash memory consumed during model execution, critical for deployment on low-memory microcontrollers.
- **Energy Consumption:** The power required for inference, determining the device's operational lifespan on battery-powered systems.
- **Accuracy:** The model's predictive performance, representing its ability to make correct inferences despite hardware optimizations.

To ensure standardized evaluation across devices and applications, the MLPerf Tiny framework has emerged as the leading benchmarking suite for TinyML. It systematically measures model performance based on latency, memory, energy, and accuracy, providing a unified standard for comparing implementations on different embedded platforms.

Although this project did not formally adopt MLPerf Tiny, its evaluation followed the same principles—systematically measuring and analyzing latency, memory usage, energy consumption, and accuracy to ensure a balanced and efficient TinyML deployment.

Chapter 3 - Methodology

Approach, tools and methods used:

To accurately simulate the rotor noise that a UAV would generate during flight, we combined our dataset of animal audio samples with a real drone rotor noise recording. This mixing process aimed to reproduce realistic acoustic conditions and train the model to focus on the distinct characteristics of each animal sound, even when masked by strong background interference. By doing so, the model became more resilient to environmental noise and better suited for real-world deployment on an operational UAV platform.

Following this step, the dataset was processed using three custom-built Python 3 preprocessing functions. Each function was designed to perform a specific transformation on the data. The scripts took the raw data folder as input and produced a new, fully preprocessed dataset, ready for training and further analysis:

- **Standardization:** Clipping the samples to 5 seconds and standardizing them to 16kHz frequency. This was done to achieve a uniform dataset to train the model on.
- **Normalization:** Normalizing the audio file amplitude to [-1, 1], for uniform loudness.
- **Spectrograms:** Turning them into spectrograms and saving them as .png files. That way we were allowed to use a CNN Image Classifier model for our project.

Throughout the stages of the project we used a wide variety of tools:

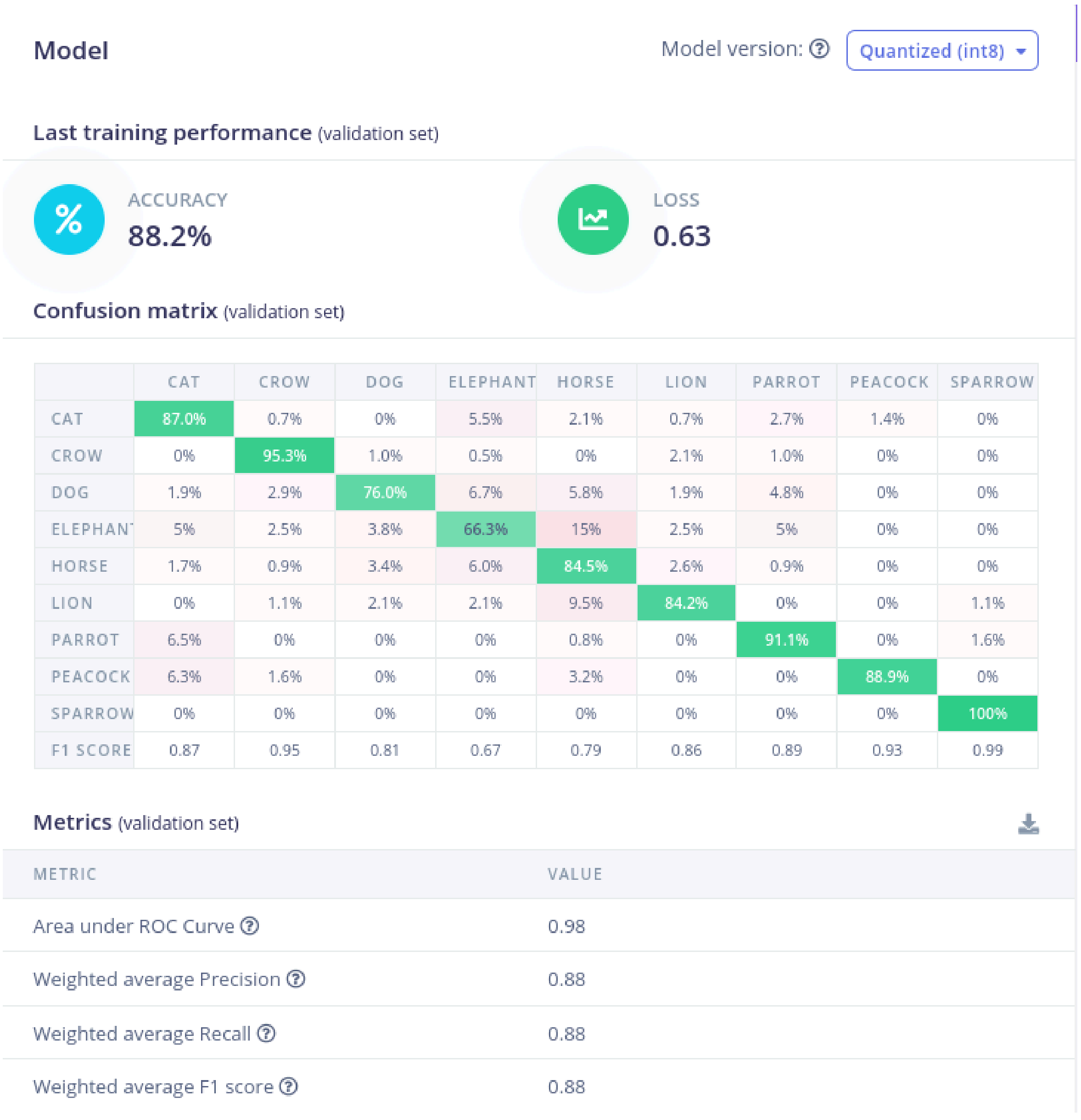
- We used this open source data set from Keggle to collect our data https://www.kaggle.com/datasets/lokeshbhaskarnr/generic-audio-samples
- The **pre-processing** was mainly carried out in Python3, with the imported libraries librosa, matplotlib, numpy, pydub and soundfile.
- To **train** our model we used Edge Impulse Studio. More specifically we used the Edge Impulse Studio's Image classifier model trained on the CPU, to classify the spectrograms, with the following setting: 20 Training cycles, with a learning rate of 0.0005 and with data

augmentation.
The neural network was set to: a 2D convolutional/pooling layer with 16 filters and a kernel size of 3, followed by another 2D convolutional/pooling layer with 32 filters and the same kernel size. These layers were responsible for extracting spatial and frequency-based features from the input spectrograms. After the convolutional layers, a flatten layer was used to convert the extracted feature maps into a one-dimensional vector, which was then passed through a dropout layer with a rate of 0.25 to prevent overfitting and improve the model's generalization performance during training.

- For the **testing** stage we used Jupyter Notebook to organize our Python code and create a new Python function that would be used to make further inferences to the model. For this step, these additional libraries were imported: time, PIL.Image and TensorFLow Lite. TensorFLow Lite in particular was the most essential tool, as it allowed us to handle our AI model.

Chapter 4 - Testing and Implementation

As mentioned in previous chapters, our final model's training accuracy ended up being 88.2%.



Model

Last training performance (validation set)

ACCURACY 88.2%

LOSS 0.63

Confusion matrix (validation set)

	CAT	CROW	DOG	ELEPHANT	HORSE	LION	PARROT	PEACOCK	SPARROW
CAT	87.0%	0.7%	0%	5.5%	2.1%	0.7%	2.7%	1.4%	0%
CROW	0%	95.3%	1.0%	0.5%	0%	2.1%	1.0%	0%	0%
DOG	1.9%	2.9%	76.0%	6.7%	5.8%	1.9%	4.8%	0%	0%
ELEPHAN	5%	2.5%	3.8%	66.3%	15%	2.5%	5%	0%	0%
HORSE	1.7%	0.9%	3.4%	6.0%	84.5%	2.6%	0.9%	0%	0%
LION	0%	1.1%	2.1%	2.1%	9.5%	84.2%	0%	0%	1.1%
PARROT	6.5%	0%	0%	0%	0.8%	0%	91.1%	0%	1.6%
PEACOCK	6.3%	1.6%	0%	0%	3.2%	0%	0%	88.9%	0%
SPARROW	0%	0%	0%	0%	0%	0%	0%	0%	100%
F1 SCORE	0.87	0.95	0.81	0.67	0.79	0.86	0.89	0.93	0.99

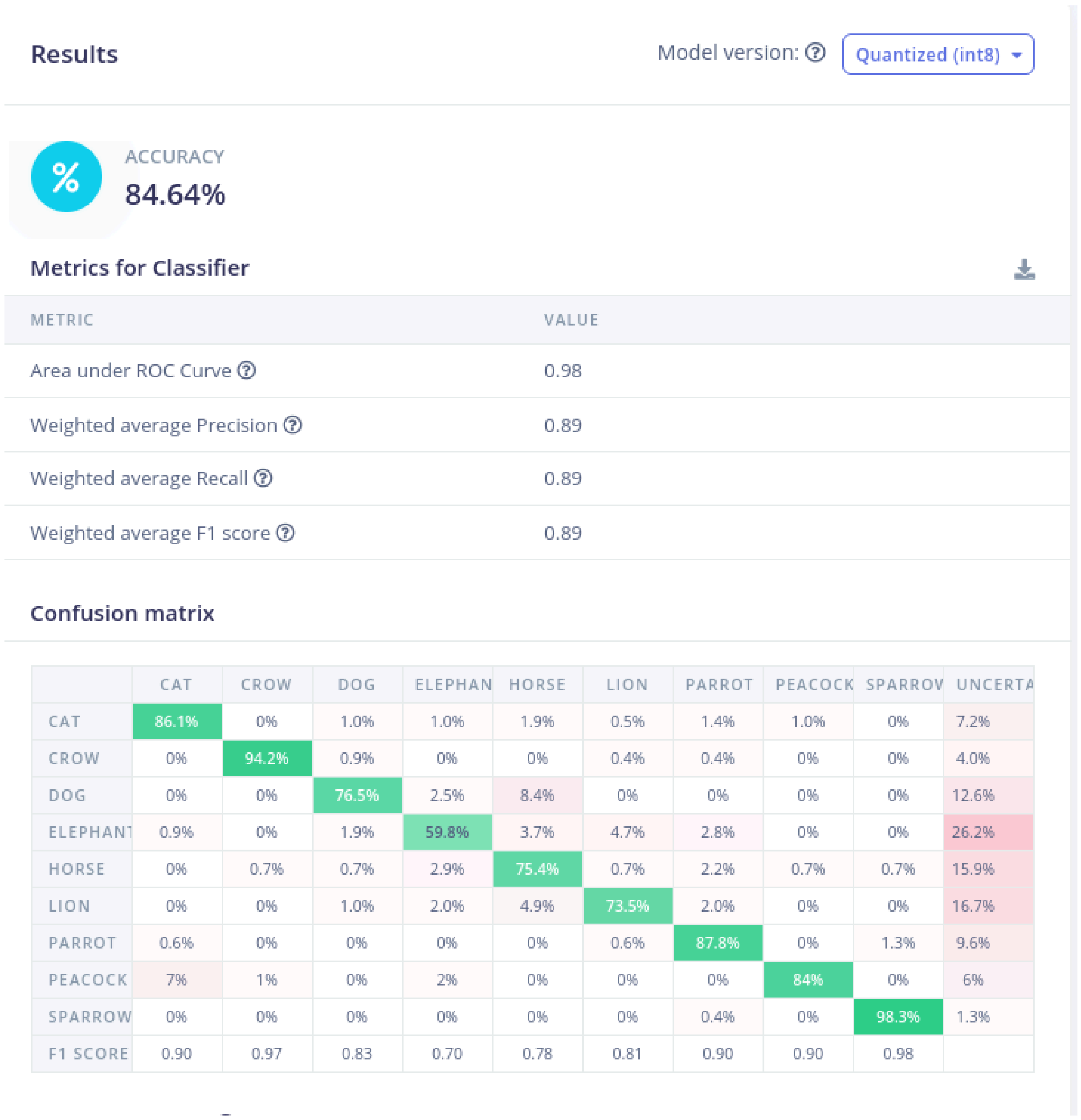
Metrics (validation set)

METRIC	VALUE
Area under ROC Curve ⓘ	0.98
Weighted average Precision ⓘ	0.88
Weighted average Recall ⓘ	0.88
Weighted average F1 score ⓘ	0.88

Image 1: Model's training results

Here we can see that our model has a satisfactory accuracy on all animals, except the elephant. The model mistakes the elephant for the house on an occasion of 15%. But even with that we still get a really good model with overall high accuracy.

For the testing of the model we got the following:


Results

Model version: Quantized (int8)

ACCURACY
84.64%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	0.98
Weighted average Precision	0.89
Weighted average Recall	0.89
Weighted average F1 score	0.89

Confusion matrix

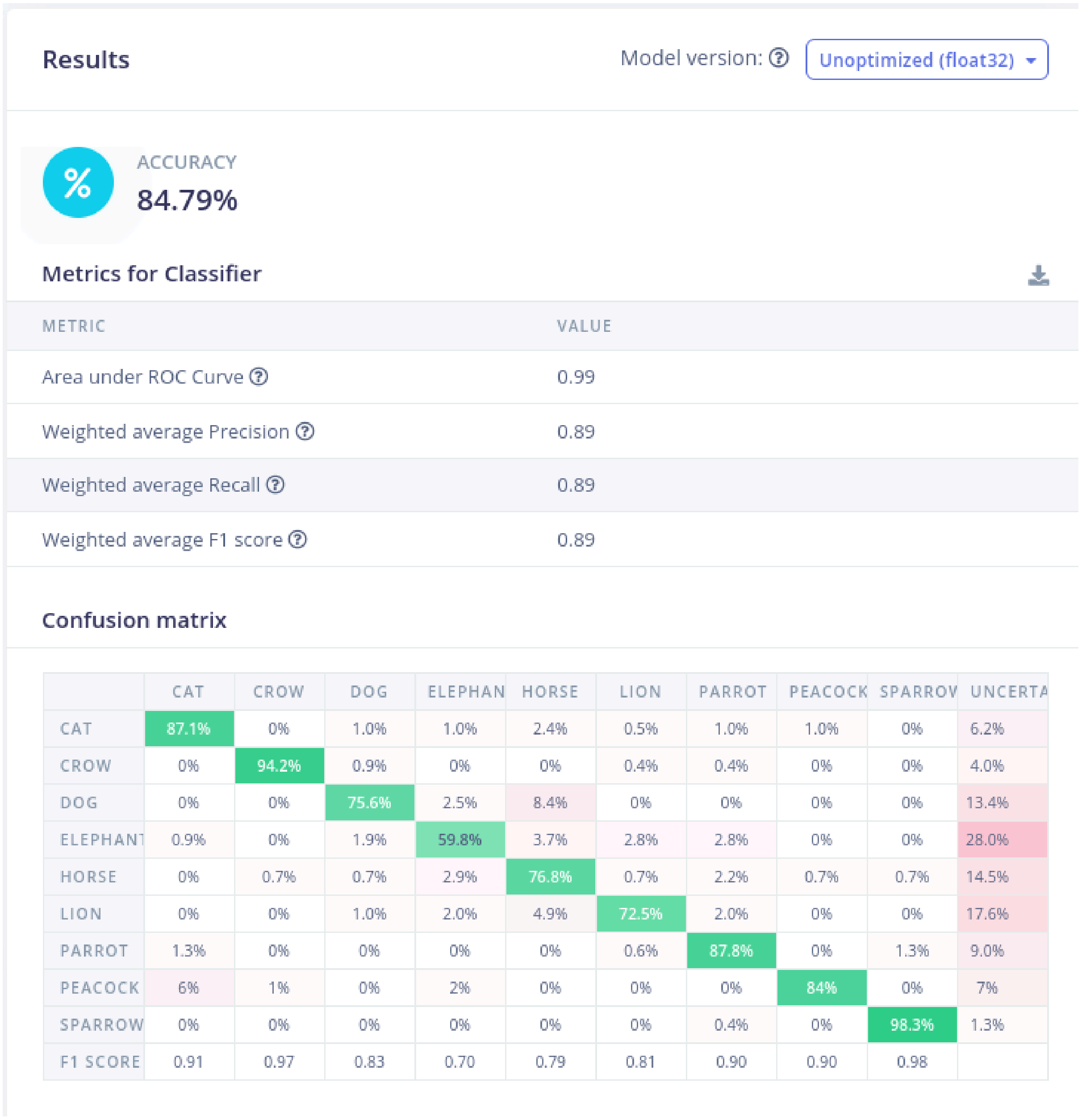
	CAT	CROW	DOG	ELEPHAN	HORSE	LION	PARROT	PEACOCK	SPARROV	UNCERTA
CAT	86.1%	0%	1.0%	1.0%	1.9%	0.5%	1.4%	1.0%	0%	7.2%
CROW	0%	94.2%	0.9%	0%	0%	0.4%	0.4%	0%	0%	4.0%
DOG	0%	0%	76.5%	2.5%	8.4%	0%	0%	0%	0%	12.6%
ELEPHANT	0.9%	0%	1.9%	59.8%	3.7%	4.7%	2.8%	0%	0%	26.2%
HORSE	0%	0.7%	0.7%	2.9%	75.4%	0.7%	2.2%	0.7%	0.7%	15.9%
LION	0%	0%	1.0%	2.0%	4.9%	73.5%	2.0%	0%	0%	16.7%
PARROT	0.6%	0%	0%	0%	0%	0.6%	87.8%	0%	1.3%	9.6%
PEACOCK	7%	1%	0%	2%	0%	0%	0%	84%	0%	6%
SPARROW	0%	0%	0%	0%	0%	0%	0.4%	0%	98.3%	1.3%
F1 SCORE	0.90	0.97	0.83	0.70	0.78	0.81	0.90	0.90	0.98	

Image 2: Model's testing results

Here we can see the accuracy dropped a bit but that is to be expected since it's normal for a model to have lower accuracy on brand new data rather than on the data it was trained with. Overall we still have a high accuracy especially for a quantized (int8) model at 84.64%, we can again see here that the model still has some trouble guessing the elephant correctly and we suspect that the reason is the poor data we managed to acquire for it.

Quantization:

We quantized the model to TensorFlow Lite (int8) so it could fit and run quickly and smoothly on the Raspberry Pi 4. The following is the accuracy of the un-optimized model:



Results

Model version: Unoptimized (float32)

ACCURACY
84.79%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	0.99
Weighted average Precision	0.89
Weighted average Recall	0.89
Weighted average F1 score	0.89

Confusion matrix

	CAT	CROW	DOG	ELEPHAN	HORSE	LION	PARROT	PEACOCK	SPARROV	UNCERTA
CAT	87.1%	0%	1.0%	1.0%	2.4%	0.5%	1.0%	1.0%	0%	6.2%
CROW	0%	94.2%	0.9%	0%	0%	0.4%	0.4%	0%	0%	4.0%
DOG	0%	0%	75.6%	2.5%	8.4%	0%	0%	0%	0%	13.4%
ELEPHANT	0.9%	0%	1.9%	59.8%	3.7%	2.8%	2.8%	0%	0%	28.0%
HORSE	0%	0.7%	0.7%	2.9%	76.8%	0.7%	2.2%	0.7%	0.7%	14.5%
LION	0%	0%	1.0%	2.0%	4.9%	72.5%	2.0%	0%	0%	17.6%
PARROT	1.3%	0%	0%	0%	0%	0.6%	87.8%	0%	1.3%	9.0%
PEACOCK	6%	1%	0%	2%	0%	0%	0%	84%	0%	7%
SPARROW	0%	0%	0%	0%	0%	0%	0.4%	0%	98.3%	1.3%
F1 SCORE	0.91	0.97	0.83	0.70	0.79	0.81	0.90	0.90	0.98	

Image 3: Unoptimized Model's results4

Impact of Quantization

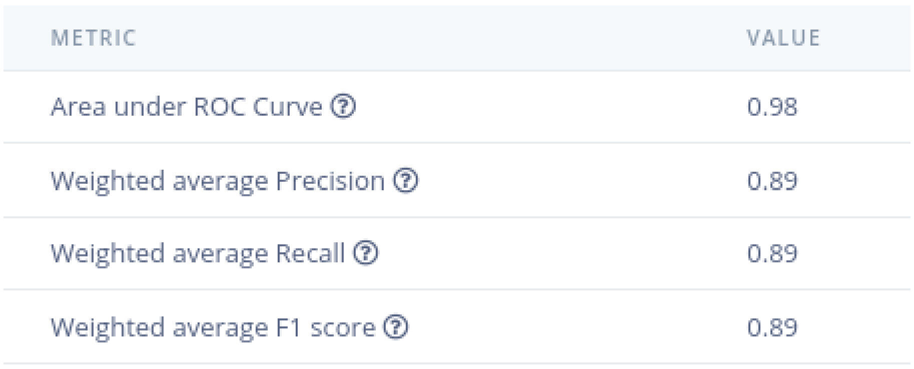
Quantizing our model to Int8 resulted in a 75% reduction in model size—from 671 KB (unoptimized) to 171 KB (quantized)—with only a negligible accuracy decrease of 0.15%. This trade-off represents a highly successful optimization, as it enabled efficient deployment on resource-constrained edge devices without compromising performance.

The unoptimized model achieved an accuracy of 84.79%, while the quantized version maintained 84.64% accuracy. Given the substantial reduction in memory requirements and the minimal accuracy difference, quantization proved to be an essential step for achieving real-time, low-power inference suitable for TinyML applications.

Benchmarking Results:

	Cat	Crow	Dog	Elephant	Horse	Lion	Parrot	Peacock	Sparrow
Precision	0.86	0.95	0.87	0.67	0.74	0.86	0.87	0.96	0.98
Recall	0.86	0.95	0.75	0.66	0.84	0.84	0.91	0.88	1
F1-Score	0.86	0.95	0.81	0.67	0.79	0.85	0.89	0.92	0.92
Support	146	192	104	80	116	95	124	63	192

Table 1: Benchmarks for each class



METRIC	VALUE
Area under ROC Curve ⓘ	0.98
Weighted average Precision ⓘ	0.89
Weighted average Recall ⓘ	0.89
Weighted average F1 score ⓘ	0.89

Image 4: Overall Benchmarks

Table 1 presents the detailed per-class performance metrics for the quantized model. Overall, the results are highly satisfactory, indicating that quantization preserved the model's discriminative capability across most classes.

Classes such as Sparrow and Crow achieved near-perfect performance, with F1-scores above 0.95, showing that the model can confidently and consistently identify these sounds. The Dog class exhibits higher precision (0.87) than recall (0.75), suggesting the model is conservative—it rarely misclassifies other animal sounds as dogs but may miss some actual dog instances. In contrast, the Elephant class shows the lowest F1-score (0.67), likely due to limited or noisy training samples, making it the most challenging class to identify.

The overall metrics (Image 4) further validate the model’s robustness, with a weighted average Precision, Recall, and F1-score of 0.89, and an AUC of 0.98, demonstrating strong generalization.

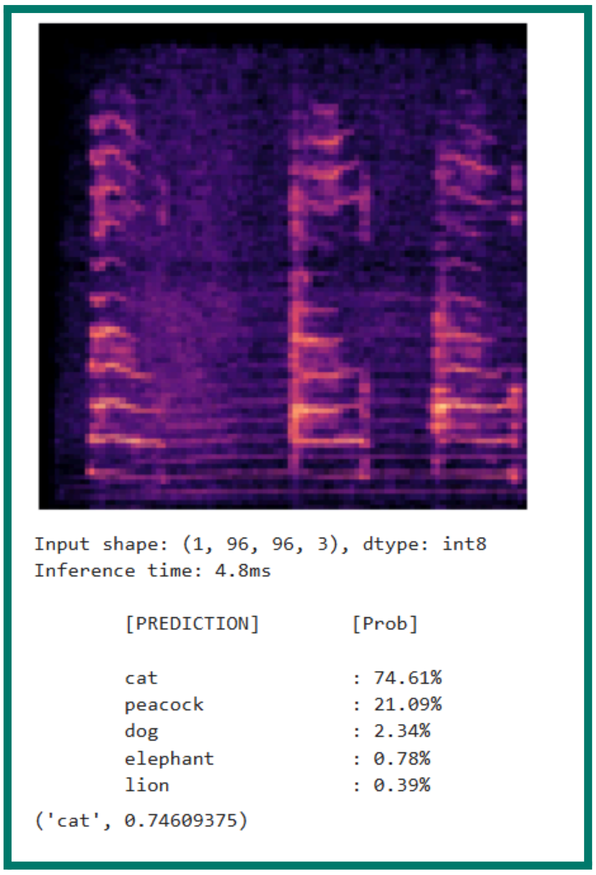
Chapter 5 - Results

In this chapter we will showcase our final results/findings during real-time inferences on a Raspberry Pi 4.

After our model was trained and evaluated we finally decided to deploy it on the Raspberry Pi 4 to see it work and make our own inferences to the model. We borrowed a Raspberry Pi 4, wiped its hard drive and re-installed the Raspberry Pi OS from scratch.
Afterwards we organized our previous scripts that we have used in the pre-processing step into a Jupyter Notebook so the new raw data could also be pre-processed once before being fed into the model for classification. We also added in the Jupyter Notebook the necessary TensorFlow Lite code that creates an inference and classifies the image it is given from the pre-processing stage.

Here are two examples we ran to test our model:

1. First we gave the model a 15 second clip of a cat meowing we downloaded from YouTube, the pre-processing code successfully cut the audio to 5 seconds, normalized it and turned the audio into a spectrogram. After that the model classified the image and we got the following result:



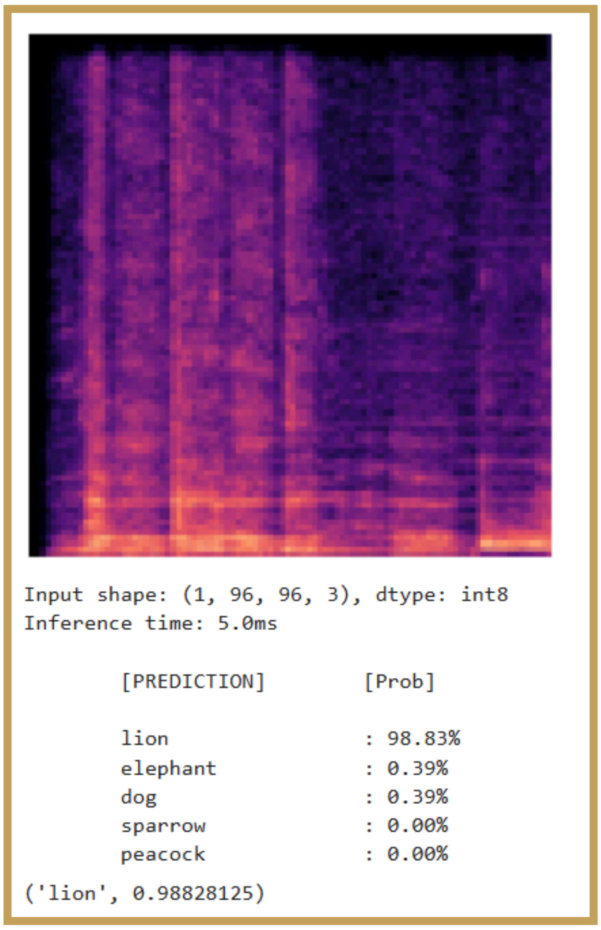
```
Input shape: (1, 96, 96, 3), dtype: int8
Inference time: 4.8ms

        [PREDICTION]        [Prob]

        cat                 : 74.61%
        peacock             : 21.09%
        dog                 : 2.34%
        elephant            : 0.78%
        lion                : 0.39%

('cat', 0.74609375)
```

Image 5: Cat inference

We see the model was able to correctly guess the cat with a confidence score of 74.61% in 4.8ms, which is quick and accurate, so the test was deemed successful.

It is interesting to note that 'peacock' was the second-highest prediction for the cat sound. This could be investigated further but does not detract from a correct and confident classification.

2. For the second test we asked the model to classify an audio sample of a lion. We followed the exact same pre-processing steps we covered in the previous example. Our result was the following:



```
Input shape: (1, 96, 96, 3), dtype: int8
Inference time: 5.0ms

        [PREDICTION]        [Prob]

        lion                : 98.83%
        elephant            : 0.39%
        dog                 : 0.39%
        sparrow             : 0.00%
        peacock             : 0.00%
('lion', 0.98828125)
```

Image 6: Lion inference

The model guessed the lion in 5.0ms with an astounding confidence score of 98.83%. This was a pleasant surprise, since the lion was one of the animal/classes that slightly underperformed in the training and testing stages.This test was deemed highly successful.
The model’s inference time seems to always be around 5ms, which makes it highly suitable for real-world use.

Chapter 6 - Discussion

How can a CNN model be trained to reliably classify animal sounds despite the presence of UAV rotor noise?

Answer: To address this challenge, we applied data augmentation by using simulated UAV rotor noise onto all training samples. This technique enhanced the model's robustness and generalization to real-world deployment conditions by mimicking the acoustic environment of drone-mounted microphones. As a result, the model learned to distinguish target animal sounds from background rotor interference, ensuring reliable performance during in-flight inference.

What preprocessing and data augmentation techniques are most effective for enhancing model performance?

Answer: For our project we found that the necessary pre-processing steps were standardization (clipping the audio to 5 seconds and standardizing it to 16kHz), normalization to [-1, 1] and turning the files into spectrograms. That way, we were able to create a uniformed dataset that could be used to train the model effectively and efficiently.

Which training settings are most ideal to ensure high accuracy on a quantized (int8) model?

Answer: The setting we found worked best for our model where the following: 20 Training cycles, with a learning rate of 0.0005 and with data augmentation. The neural network was set to: a 2D convolutional/pooling layer with 16 filters and a kernel size of 3, followed by another 2D convolutional/pooling layer with 32 filters and the same kernel size. These layers were responsible for extracting spatial and frequency-based features from the input spectrograms. After the convolutional layers, a flatten layer was used to convert the extracted feature maps into a one-dimensional vector, which was then passed through a dropout layer with a rate of 0.25 to prevent overfitting and improve the model's generalization performance during training.

Did we achieve our goals?

When we started this project we were quite ambitious with the goals we had set. We had planned to find a larger number of animals/classes and were even thinking of training it to guess multiple animals in the same audio sample. However, the nature of our limited and noisy dataset, and the complexity of the task and difficulty to find a satisfactory dataset made it not possible.
Besides that, we are quite satisfied with our final model. It is quick and accurate and operates on the Raspberry Pi 4 with no issues or bugs, and is able to properly classify even the animals it had initially given low scores for. Furthermore, it is successfully trained to ignore the UAV's rotor noise and can make correct guesses despite it.

Future Work:

In future work, we plan to extend the current system in several directions. First, testing and validating the approach with a real UAV in operational conditions will provide insights into its practical feasibility and performance. Additionally, improving the dataset, particularly for underperforming classes such as elephant, could enhance classification accuracy and model robustness. Finally, we aim to explore real-time audio streaming and continuous classification, moving beyond single-sample inference, to enable more responsive and dynamic monitoring in real-world scenarios.

Conclusion:

In the end we managed to complete the project with satisfactory accuracy and we made a real deployable model that could be used to help scientists in the field. We finished the project in due time without any major setbacks or problems. If we had the necessary resources we would have taken the task a step further by performing actual tests with a real UAV and testing it out in the field.