Applying Deep Learning Models for Classifying Aurora Events

Hans Alin

January 16, 2026

1 Introduction

Auroral activity roughly follows the solar cycle: during periods of high solar activity, as indicated by an increased number of sunspots [**kvammenAuroralImageClassificatio** the probability of observing aurora increases. Consequently, public interest in seeing the aurora, also rises during these periods.

The growing interest in aurora during solar cycle peak activity not only offers the community the opportunity to explore an amazing colour explosion in the night sky, but can also serve as a gateway to a more scientific approach to the phenomenon and an eagerness to learn more about space weather and the sun-earth connection.

In order to maximize the chances of observing aurora, a specially designed aurora detector has been created [**alinAuroraDetectorAffordable2022**]. This detector first uses light sensors to measure light intensity in the sky, and secondly uses temperature and humidity to estimate cloud cover. An algorithm has been developed to combine the data from these sensors to provide an aurora probability index.

Similar work has been done by Nanjo et al. [**nanjoAutomatedAuroralDetection2022**], who used 128 x 128 pixel all-sky images from Tromsø to train a ResNet-50 model for broad aurora classification, distinguishing between clear-sky aurora, cloudy-sky aurora, and no aurora. Kvammen et al. [**kvammenAuroralImageClassificatio** used a similar approach with 3846 labeled 128 x 128 pixel images from an all-sky camera in Kiruna, training models such as AlexNet and ResNet. Their best model, a ResNet-50, achieved an F1-score of 0.95. While effective, these approaches require complex and expensive all-sky camera setups, making them unsuitable for the affordable aurora detector concept.

1.1 Deployment requirements on aurora detector

Ideally, the aurora detector would be deployed in remote areas with low light pollution and minimal human activity. However, since its primary purpose is to alert local residents of aurora events, deployment in uninhabited areas is not practical. Consequently, the device must withstand various weather conditions, including frost, snow, rain, and moderate artificial light.

Given the need to reach residents, deployment is limited to urban or suburban areas, where the device can be installed on rooftops or antenna towers.

The main motivation for the project [**alinAuroraDetectorAffordable2022**] was to create an affordable aurora detector that could be deployed in larger numbers close to populated areas to increase the chances of citizens observing aurora events. This means that both the device cost and deployment costs should be kept low.

Considering the deployment cost, the device should be self-sustaining in terms of power consumption, meaning it should use a solar panel and a battery to store energy, which implies no extra cost for power supply.

It is also assumed that the device will have a Wi-Fi connection, as this is a common feature in urban and suburban areas, enabling it to send alerts to the community.

1.2 Motivations for applying Deep-Learning model on Aurora Detector

The main motivation for applying deep learning models to the aurora detector is to improve aurora event detection while keeping the false-positive rate low. The current algorithm, though effective, still produces false positives when frost or snow accumulates on the sensor. This occurs because frost can refract light from angles not accounted for by the detector's design..

In [**sahaMachineLearningMicrocontrollerClass2022**], [**wardenTinymlMachineLearn**
highlight the advances of edge AI, i.e., deploying machine learning models on microcontrollers to enable real-time data processing and decision-making directly on the device, as well as its shortcomings.

Situnayake [**situnayakeAIEdge2023**] mentions bandwidth limitations, which are unlikely to pose a problem for aurora detectors deployed in urban or suburban areas with Wi-Fi, but remain a consideration. Most power consumption in embedded devices originates from data transfer [**karicSendLessMore2025a**]."

In [**wardenTinymlMachineLearning2019**], they emphasize latency, an important factor for the aurora detector, since the goal is to alert citizens

in real time when an aurora event occurs. Getting an alert a few minutes late might result in missing the event.

Both [**sahaMachineLearningMicrocontrollerClass2022**] and [**wardenTinymlMachinel** highlight the privacy aspect of edge AI, since data is processed locally on the device, there is less risk of sensitive information being exposed during data transmission. However, this is not a major concern for the aurora detector data.

Lastly, the main motivation for using deep-learning models on the Aurora detector is the potential to reject false positives caused by frost, snow, and other environmental factors while maintaining accuracy as good as, or better than, the current algorithm [**alinAuroraDetectorAffordable2022**].

2 Electronics overview

The full electronics setup is described in [**alinAuroraDetectorAffordable2022**]. Here, a brief overview of the components used in the Aurora detector is provided.

2.1 Microcontroller

The current aurora detector uses an ESP-8266 on a Wemos D1 MINI developer board. The microcontroller reads sensor data, processes it using the aurora detection algorithm, and sends the raw data and aurora probability index to a server via WiFi. The operational device is continuously powered by a 5V supply.

Since the goal is to deploy deep-learning models on the Aurora detector, the microcontroller must be replaced with a more powerful one capable of running them. The device considered in this project is a Raspberry Pi Pico W, a microcontroller with WiFi and enough processing power to run deep learning models using TensorFlow Lite for Microcontrollers.

2.2 Light sensor

The aurora detector uses two TSL2591 light sensors. One sensor is equipped with a multilayer interference filter that selectively transmits wavelengths near 557 nm (green), the most prominent auroral emission. The sensors provide digital readings from 0 to 65,535 (depending on gain) representing measured light intensity. Each sensor has two channels with distinct spectral responses: one channel is more sensitive to infrared, while the other primarily responds to visible light.

Table 1: Overview of features contained in the raw dataset and their physical interpretation.

Feature	**Description**
Timestamp (`created_at`)	Time at which the data point was recorded
Entry ID (`entry_id`)	Unique identifier for each data record
Green-channel light intensity	Light intensity measured through a green-pass filter
Broadband light intensity	Unfiltered visible light intensity
Clear-sky reference value	Estimated clear-sky brightness computed onboard
Aurora activity index	Aurora detection score (approximately 0- -20), computed onboa
Ambient temperature	Air temperature measurement
Relative humidity	Ambient relative humidity measurement
Sky temperature	Infrared sky temperature measurement
Infrared light intensity	Infrared-channel light intensity

2.3 Infrared thermometer

The MLX90614 infrared thermometer measures the effective radiative temperature of the sky. A clear sky appears cooler than a cloudy sky because clouds emit more infrared radiation. The sensor reports temperature in degrees Celsius as floating-point values.

2.4 Humidity and temperature sensor

The DHT22 sensor measures ambient temperature and humidity. These readings are used to estimate the expected clear-sky temperature under the current conditions, which can then be compared with the infrared thermometer's measurement to estimate cloud cover.

3 Data collection and preprocessing

The data for this project were collected between 1 December 2024 and 30 November 2025 using an instrument deployed in Mora, Sweden (61.0°N, 14.5°E). The dataset is available via a public ThingSpeak instance [**AuroraDetectorThingSpeal** however, administrative access is needed to download the full dataset. Table **??** provides a summary of the raw data features.

Since the data had a median sampling rate of 1/15 Hz, the full-year dataset contained approximately 2 million samples. To save computational power, an algorithm was used to determine whether it is day or night, taking the current date, time, and location into account. This algorithm is currently

Table 2: Overview of measured and derived features used for model input after preprocessing.

Feature	**Description**	**S**
Green-channel light intensity	Light intensity measured through a 557 nm green-pass filter	T
Broadband light intensity	Unfiltered visible light intensity	T
Ambient temperature	Air temperature measurement (°C)	D
Relative humidity	Ambient relative humidity (%)	D
Sky temperature	Infrared sky temperature measurement (°C)	M
Infrared light intensity	Infrared-channel light intensity	T
Rolling mean temperature	20-minute rolling average of ambient temperature (°C)	D
Rolling mean humidity	20-minute rolling average of relative humidity (%)	D

operating as a veto on the current Aurora device, but this information is not sent to Thingspeak. This algorithm was applied to the full dataset to filter out daytime samples, since visible aurora events only occur during the dark hours. This reduced the dataset by approximately half.

To capture frost add-on and frost removal, two additional features were added to the data set. Since humidity, along with temperature, argely captures the likelihood of frost. A new feature called "Rolling Mean Temperature (C)" and "Rolling Mean Humidity (%)" was added to the dataset. The rolling mean was calculated using a 20-minute window, with the previous 20 minutes of data. Endpoints were handled such that only samples within a full window were considered. The idea was that these two values could help the model learn when frost is likely to occur and when it is likely to melt.

Since this project applies deep learning models to the aurora detector, it was considered that most of the precalculations performed on the microcontroller should be avoided to let the model learn directly from the raw sensor data. Therefore, only the features seen in table **??** were used for training the models.

3.1 Labeling strategy

From observations between 2024-12-01 and 2025-11-30, there were a total of 8 aurora events visible from the deployment location, verified by the observer on 8 different dates. One verified event with no aurora, although the microcontroller's algorithm indicated a high aurora probability, was also observed.

To label the data, a time window was swept over the data from the beginning to the end of the considered aurora event. The time considered was between 2 and 6 hours. The windows were swept over the data within

the considered time at a single verified date, and the light sensor was visually inspected with a filter at 557nm. Most of the data points, including a value from a 557 nm sensor having a larger value than 3, were considered as an aurora event. The threshold value of 3 was chosen based on visual inspection of the data from all verified aurora events.

Additional labelling was performed for the known non-aurora event using the same principles previously mentioned.

Moreover, 5 dates with high aurora points from the microcontroller algorithm, but not confirmed by the observer, were also considered. Similar procedures were applied as above, but to avoid mislabeling of aurora events, magnetometer data from Kiruna and Lyksele were used to verify the aurora event. Only those events with high magnetometer activity, and aurora points larger than 3 were considered as aurora events.

3.2 Data summary

After filtering out daytime samples and applying the labelling strategy, the final dataset used for training and evaluating the deep-learning models contained approximately 1.8 M unlabeled samples and approximately 14,000 labelled samples, with around 13,000 labelled as non-aurora and around 1,000 labelled as aurora. This indicates a significant class imbalance, with non-aurora samples vastly outnumbering aurora samples.

Before training with the data, data normalization was applied using the mean and standard deviation calculated from the training set. The normalization was done using the formula:

$$X_{norm} = \frac{X - \mu}{\sigma}$$

where X is the original feature value, and μ and σ are the mean and standard deviation of the feature in the training set, respectively.

4 Model development

Because this project aims to deploy deep learning models on microcontrollers, the architecture was designed to be simple to minimize computational cost. As a proof of concept, a fully connected neural network was implemented.

The dataset contains mostly unlabeled data, about 20 times more than the labeled samples, and the labeled portion is highly imbalanced, with the majority of samples corresponding to non-aurora events. To handle this, a semi-supervised learning approach was adopted: the models were first trained

on the unlabeled data in an unsupervised manner, and then fine-tuned using the labeled samples.

This project largely follows the guidelines for that type of training from [**geronHandsonMachineLearning**]. The models are created by Keras, and the script describing the model creation and training is written in Python and can be found in the GitHub repository "tiny_ml_code/models/FC_autoencoder.py" [**alinHansAlinTinymlauroradetector2026**].

4.1 Model architecture Autoencoder

The autoencoder is a fully connected neural network with the following structure:

- Input layer: 8 neurons (one for each feature)
- Encoder Hidden layer 1: Dense layer with 16 to 512 neurons, Leaky ReLU activation
- Encoder Hidden layer 2: Dense layer with 8 to 256 neurons, Leaky ReLU activation
- Encoder Latent layer: Dense layer with 2 to 64 neurons, Linear activation
- Decoder Hidden layer 3: Dense layer with 8 to 256 neurons, Leaky ReLU activation
- Decoder Hidden layer 4: Dense layer with 16 to 512 neurons, Leaky ReLU activation
- Decoder Final layer: Dense layer with 8 neurons, Linear activation

To use the model as a binary classifier, the encoder from the autoencoder is retained, and the following layers are added:

- Classifier Hidden layer 3: Dense layer with 8 - 128 neurons, Leaky ReLU activation
- Classifier Last layer: Dense layer with 1 neuron, Sigmoid activation

Note that in the final layer, a sigmoid activation is used, since the model is a binary classifier and is supposed to return a probability between 0 and 1. However, this is only valid for the unquantized models.

A summary of the models and theri layers can be seen in Figure **??**.

Table 3: Neural network architecture configurations for supervised and semi-supervised models. Each row corresponds to a matched model pair sharing the same architecture.

Model pair	Width L1	Width L2	Latent size	Width last
Semi-Supervised 1 / Supervised 6	16	8	2	8
Semi-Supervised 2 / Supervised 6	32	16	4	16
Semi-Supervised 3 / Supervised 7	64	32	8	32
Semi-Supervised 4 / Supervised 8	128	64	16	64
Semi-Supervised 5 / Supervised 9	256	128	32	128
Semi-Supervised 6 / Supervised 10	512	256	64	256

5 Training procedure

Two classification models were constructed and trained in this project: one using only labeled data (supervised learning) and one using a semi-supervised learning approach.

For both approaches, the Adam optimizer was used with an initial learning rate of 10^{-3}. For the semi-supervised approach, the learning rate was reduced during the fine-tuning phase. A batch size of 32 samples was used for training both models.

The full training scripts are available in the GitHub repository "$\text{tiny}_m l_c ode/train.py$".

5.1 Semi-supervised learning

The semi-supervised learning procedure involved two main steps, pretraining an autoencoder on the unlabeled data, followed by fine-tuning the encoder-classifier model on labeled data.

The autoencoder was trained for up to 200 epochs with an early stopping callback monitoring the validation loss and a patience of 10 epochs. The trained weights were saved for subsequent use.

The encoder-classifier model inherits the encoder architecture from the pretrained autoencoder and adds classifier layers. Initially, the encoder weights were frozen, and only the classifier layers were trained for up to 200 epochs with early stopping, preventing overfitting. After this first training phase, the encoder weights were unfrozen, and the entire model was retrained for up to 200 epochs with early stopping and a reduced learning rate of 10^{-4}.

5.2 Supervised learning

In the supervised learning approach, the encoder-classifier model was trained directly on the labeled data without any pretraining. All weights were trainable from scratch, and the model was trained for up to 200 epochs with early stopping, using a patience of 10 epochs to monitor the validation loss. The architecture was identical to that used in the semi-supervised approach.

6 Quantization and preparing model for deployment

After training, the model was prepared for deployment through size reduction and int8 quantization using Keras and TensorFlow Lite for Python. The procedure followed the methodology described in [**iodiceTinyMLCookbookCombine2022**].

First, the trained Keras model and its weights were saved. The tf.lite.TFLiteConverter utility was then used to create a converter object. During preliminary testing, it was observed that the quantized model had difficulty handling extreme output values, so the original sigmoid activation in the final layer was omitted.

6.1 Computational and Electrical Power

where n_{params} is the number of parameters in the model and X is the size of each parameter in bytes. For an int8-quantized model, each parameter takes 1 byte, whereas in the original float32 models (unquantized), each parameter takes 4 bytes. This was the internal Keras calculation used to estimate the model's storage size.

To estimate RAM usage during inference, the number of tensor elements in each layer multiplied by the size, in bytes, of each element was calculated and summed across all layers in the model. This gives an estimate of the RAM required during inference.

To get en estime of the inference time on the Raspberry Pi Pico W, it was assumed that the microcontrollers, that have a stated clock speed of 133 MHz [**raspberrypifoundationRaspberryPiPico**], can perform one operation per clock cycle, which gives the estimated inference time equal to:

$$t_{inference} = \frac{MACs}{133}[s],$$

where MACs is the number of multiply and addion operations required for a single inference.

The energy required was estimated by multiplying the Raspberry Pi Pico W's operating power of 100 mW [**raspberrypifoundationRaspberryPiPico**] by the estimated inference time. Note that the citation is to a Raspberry Pi Pico and not to a Raspberry Pi Pico W.

7 Results

Since we are most interested in minimizing false positives and still detecting as many auroral events as possible, the main metric used to evaluate the models was the F1-score. Additionally, we would like a model that, in principle, can reject all false positives, meaning we would like a high true positive rate (TPR) at a low false positive rate (FPR). Here, we set the limit to FPR $< 10^{-4}$, meaning that we accept one false positive for every 10 000 non-aurora events.

The choice of FPR limit is rather arbitrary, but considering that the detector will generate an event every 15th second on average during the night, this would result in roughly 1 event every 4th night. We also report the TPR corresponding to FPR $< 10^{-4}$.par

7.1 Semi-Supervised models

The performance of the semi-supervised models is shown in Figure **??**, which displays the confusion matrices for the semi-supervised models and their quantized counterparts.

One can see that for all the models, the quantized model has a lower TPR than the corresponding unquantized model except for the smallest model (Experiment 2).

In Table **??**, one can see that the performance regarding F1-score is highest for Semi-Supervised 4 with 0.24, and the highest TPR at FPR $< 10^{-4}$ is also for Semi-Supervised 4 with 0.13.

In general, quantization reduces TPR, and the performance drop is more pronounced for larger models.

7.2 Supervised models

The performance of the supervised learning models is shown in Figure **??**. The confusion matrices for the supervised models and their quantized counterparts are shown, and the overall pattern follows the semi-supervised models.

Table 4: Performance and resource usage for supervised and semi-supervised models before and after quantization. Note that the estimated energy consumption is based on continuous inference over a month given an event detection every 15 seconds.

	Unquantized model				Quantized model					
Model	F1	TPR@FPR	MACs	Params	F1	TPR@FPR	Flash [B]	RAM [B]	Time (μs)	mAh / month
Semi-Supervised 1	0.17	0.06	1 800	1 000	0.21	**0.12**	6 500	1 300	13.00	21
Semi-Supervised 2	0.22	**0.13**	6 100	3 200	0.17	0.10	10 500	3 900	47.00	73
Semi-Supervised 3	0.20	0.12	22 400	11 600	0.10	0.05	22 400	12 900	172.00	268
Semi-Supervised 4	**0.24**	**0.13**	85 700	43 700	0.07	0.04	61 600	46 300	659.00	1 026
Semi-Supervised 5	0.23	**0.13**	335 300	169 300	0.12	0.05	201 400	174 400	2 579.00	4 011
Supervised 6	0.20	0.07	1 800	1 000	0.17	0.11	6 500	1 300	13.00	21
Supervised 7	0.20	0.04	6 100	3 200	0.20	0.11	10 500	3 900	47.00	73
Supervised 8	0.18	0.06	22 400	11 600	0.14	0.09	22 400	12 900	172.00	268
Supervised 9	0.17	0.07	85 700	43 700	0.15	0.09	61 600	46 300	659.00	1 026
Supervised 10	**0.24**	0.06	335 300	169 300	0.17	0.11	201 400	174 400	2 579.00	4 011

In Table **??**, one can see that the highest F1-score is for Supervised 10 with 0.24, and the highest TPR at FPR $< 10^{-4}$ is for the quantized models 6, 7, and 10.

There are no significant differences between Semi-supervised and supervised learning in terms of performance. In general, quantization reduces the F1-score, with a larger drop for the semi-supervised models than for the supervised models. Regarding TPR at FPR $< 10^{-4}$, quantization decreases TPR for all semi-supervised models except one, while it increases TPR for the supervised models.

7.3 Computational and energy constraints

In Table **??**, one can see the computational and energy requirements for all the quantized models. Both the flash and RAM requirements increase with model size, as expected and are well within the limitations on the Raspberry Pi Pico with 2M QSPI Flash [**raspberrypifoundationRaspberryPiPico**] and 256K [**oliveiraInternetIntelligentThings2024**].

Regarding the extra energy the microcontroller needs, we see that the largest model will need 4000 mAh per month to cover the usual operations. This is based on the assumption that an inference happens every 15th second during the night and is an upper bound estimate for continuous operation.

8 Conclusion

This study examined the application of deep learning models to an affordable aurora detector. The goal was to improve aurora event detection while

minimizing false positives. Two training paradigms—semi-supervised and supervised learning—were investigated, both utilizing a straightforward fully connected neural network architecture.

The results indicate that both methods yield comparable performance, with the highest F1-score of 0.24 achieved by both the Semi-Supervised 4 and Supervised 10 models. However, quantization generally resulted in decreased performance across all models in terms of F1-score with a drop between 0.05 to 0.17, except for the quantized version of Semi-Supervised 1, which slightly outperformed its unquantized counterpart. Regarding the true positive rate (TPR) at a false positive rate (FPR) below 10^{-4}, the best performance (0.13) was observed for Semi-Supervised 2, 4, and 5. For semi-supervised models, TPR at FPR $< 10^{-4}$ decreased after quantization in all cases except Semi-Supervised 1. In contrast, supervised models showed increased TPR at FPR $< 10^{-4}$ after quantization, possibly due to differences in how the weights are updated during training.

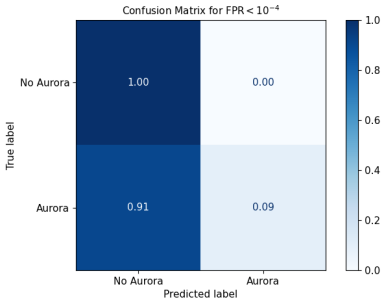
It is challenging to draw definitive conclusions regarding the superiority of either training approach due to the limited size and structure of the labeled dataset. The labeling strategy was based on a small set of calendar dates (approximately ten) during which aurora activity was known to occur, with time windows of approximately 4-6 hours per date. Within these windows, the detector produced a large number of events, the majority of which did not correspond to aurora, resulting in substantially more labeled non-aurora events than aurora events, despite the dates being aurora-positive. In addition, several dates (three to five) corresponding to false alerts from the existing algorithm were included and labeled entirely as non-aurora. Consequently, the labeled dataset exhibited a strong class imbalance. Although class weighting was applied during training, it may not have been sufficient to fully compensate for this imbalance.

Furthermore, the labeled dataset comprised approximately 1,000 aurora events and 13,000 non-aurora events. This resulted in a significant class imbalance. Although classification weighting was implemented, it may not have been sufficient to fully compensate for this imbalance.

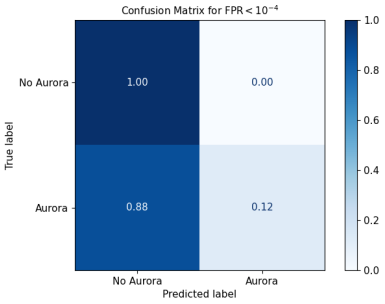
With respect to model deployment on the Raspberry Pi Pico W, all quantized models were found to fit comfortably within the microcontroller's flash memory and RAM constraints. The estimated energy consumption for inference was also determined to be manageable, even for the largest model considered.

In conclusion, this study demonstrates that deep learning models can be deployed on an affordable aurora detector. However, performance does not yet surpass the existing algorithm. Future work could explore more sophisticated model architectures, expand and diversify the labeled dataset,

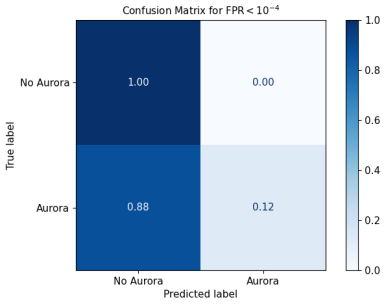
and further optimize models for resource-constrained hardware.



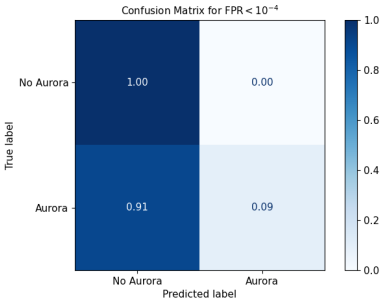
(a) Semi-Supervised 1



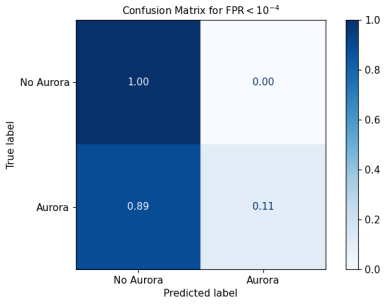
(b) Quantized 1



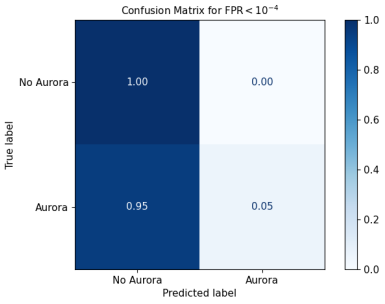
(c) Semi-Supervised 2



(d) Quantized 2

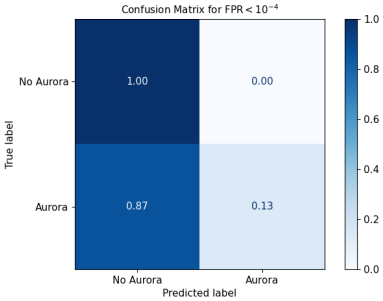


(e) Semi-Supervised 3

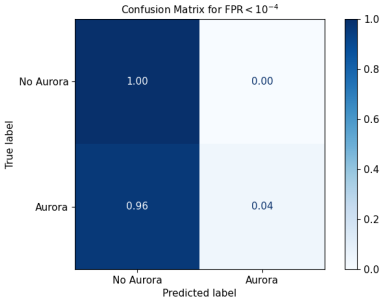


(f) Quantized 3

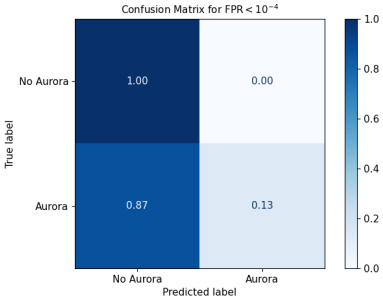
Figure 1: Confusion matrices comparing semi-supervised and quantized TinyML classifiers.



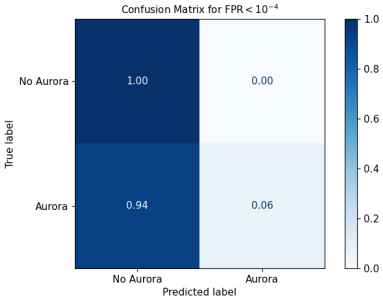
(g) Semi-Supervised 4



(h) Quantized 4

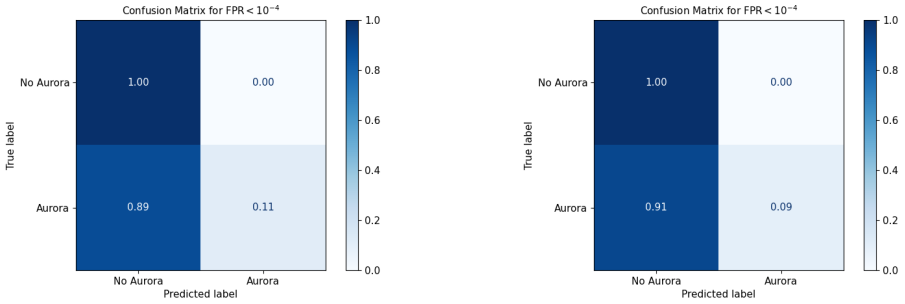


(i) Semi-Supervised 5



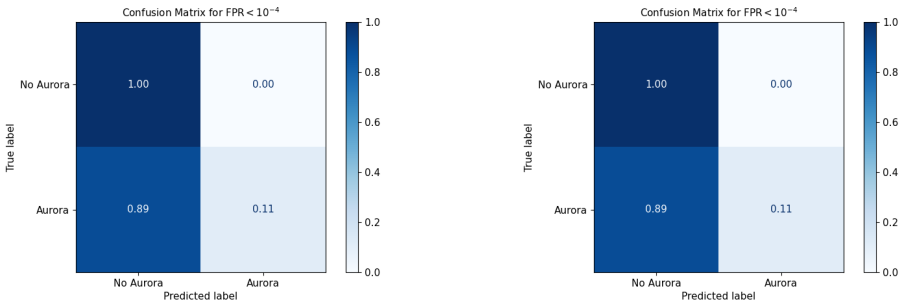
(j) Quantized 5

Figure 1: (continued)



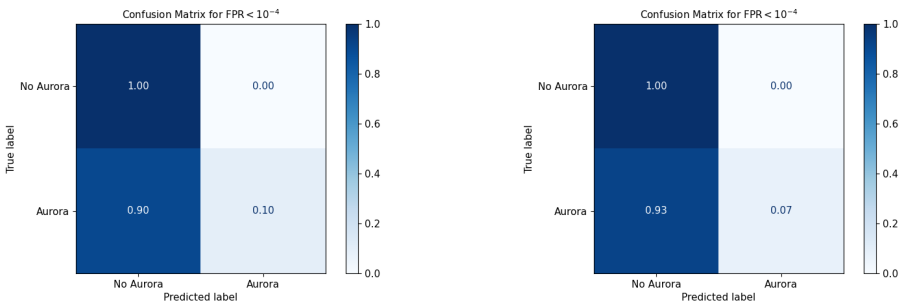
(a) Supervised 6

(b) Quantized 6



(c) Supervised 7

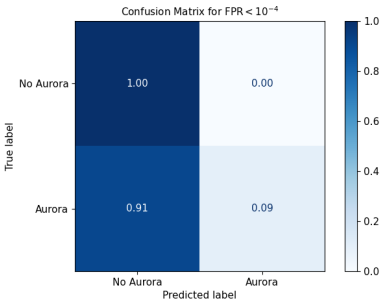
(d) Quantized 7



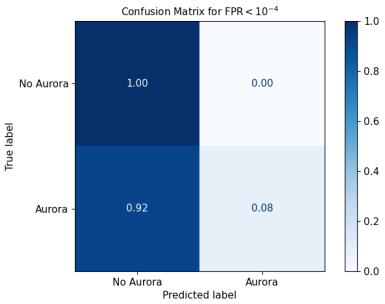
(e) Supervised 8

(f) Quantized 8

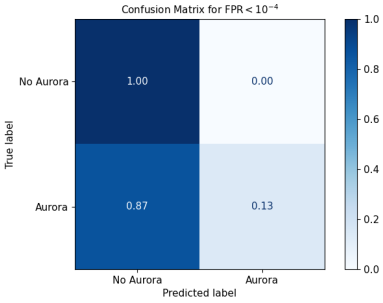
Figure 2: Confusion matrices comparing supervised and quantized TinyML classifiers (Experiments 6 - 8).



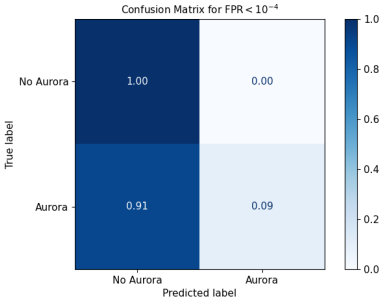
(g) Supervised 9



(h) Quantized 9



(i) Supervised 10



(j) Quantized 10

Figure 2: (continued) Confusion matrices comparing supervised and quantized TinyML classifiers (Experiments 9 - 10).