# TinyML-Empowered Spectrum Sensing on Microcontrollers

**A continuation of the Spectrum Painting method**

**Seth Schröder**[1]

**Supervisor: Qing Wang**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

## Abstract

Spectrum sensing is a vital technology for alleviating pressure on the radio spectrum and will become more sophisticated as billions more devices come online. In the future, more advanced techniques utilizing deep learning will sense which parts of the spectrum are available to communicate on. This is a heavily researched area, but few papers demonstrate methods of deploying deep learning on the resource-constrained edge devices that will ultimately use them. One approach called Spectrum Painting augments spectrograms and detects which signals are present with a Convolutional Neural Network. We optimize this method for microcontrollers by simplifying the computation of spectrograms and using TinyML techniques. This results in over 90% accuracy on signals with a high signal-to-noise ratio and a latency of 159 ms on a 64 MHz CPU. Our findings conclude microcontrollers are capable of utilizing deep learning for spectrum sensing, but custom hardware will still be required in a real-world deployment.

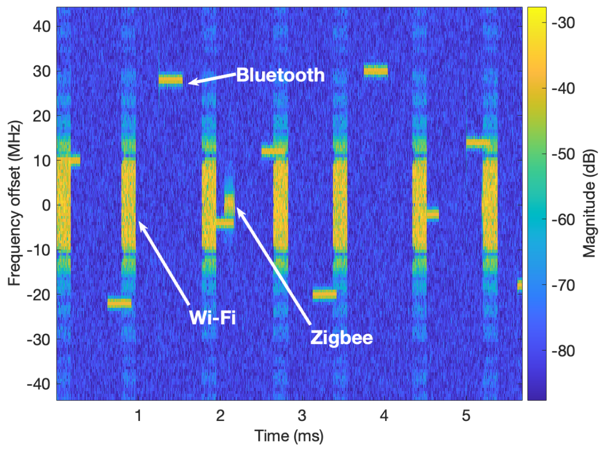


Figure 1: A spectrogram centered on the 2.4 GHz frequency band containing virtual Wi-Fi, Zigbee, and Bluetooth signals. Zigbee is a protocol when low-power and low-bandwidth are required, such as Internet of Things devices. These different protocols impede communication efficiency by not collaborating while they share the same frequency band, causing their signals to frequently interfere.

## 1 Introduction

There are an increasing number of devices using wireless technologies, causing parts of the wireless spectrum to become over-crowded. There are around 50 billion connected devices across the world, and billions more will come online in the decades ahead [1], leading to a “spectrum crunch” [2]. In the past, government regulators divided up the spectrum into fixed bands, which are exclusively licensed to a single user. A user in this context referring to a type of technology, such as television broadcasts or Wi-Fi, and not a person using said technology. This approach is inefficient because portions of the spectrum are under-utilized the majority of the time, therefore, some frequency bands resort to spectrum *sharing* [3]. One example that is significantly over-crowded is the 2.4 GHz band which Wi-Fi, Bluetooth, and many other wireless technologies use [4].

Some sharing approaches are non-collaborative meaning users must opportunistically transmit when there are free slots, rather than coordinating when they will each transmit. This approach requires spectrum sensing — the ability for a user to sense which parts of the spectrum are being used. Increasingly into the future, spectrum sensing will be required to hop between different channels or frequency bands dynamically [5]. As shown in Figure 1, Bluetooth devices already use a technique called Adaptive Frequency Hopping to reduce interference during transmission [6]. Regulators have also recognized this as an urgent problem, thus to alleviate pressure they are changing their previous practice of exclusive access by opening other parts of the spectrum. In 2020, the US Federal Communications Commission opened up the 6 GHz band to unlicensed users to reduce the demand on the aforementioned 2.4 GHz band [7]. Nevertheless, advanced spectrum sensing techniques are still relevant because it is conceivable that these new unlicensed bands will also experience pressure into the future.

Radio environments are very dynamic and chaotic, especially when the users are not collaborating, which lends itself well to deep learning approaches that can adapt to different inputs [5], [8], [9], [10], [11]. The complexity of deep learning makes it challenging to minimize the latency, which is a measure of the time to pre-process and classify the data. Few papers discuss this aspect even though it is important because the radio environment changes within milliseconds. A couple papers are latency-driven, such as one that presents a method of optimizing Convolutional Neural Networks (CNN) in general for field-programmable gate arrays (FPGA) [12], and another introduces the *DeepSense* framework to achieve real-time spectrum sensing with a CNN on custom hardware [11]. Embedded devices using wireless technologies for industrial and household applications are becoming more ubiquitous and further putting pressure on the spectrum. Applying deep learning methods with low latency to these devices is especially challenging because they have very limited computing and memory resources. This could be achieved with Tiny Machine Learning (TinyML), which is an emerging research field concerned with optimizing machine learning models to run on ultra-low-power devices [13].

Many proposed deep learning methods fail to have both high accuracy and low latency, therefore, Li et al. developed the Spectrum Painting [14] method to have both. Also, some models struggle in situations (like Figure 1) where spectrograms with both large and small features must be classified. So they improved on this aspect and achieved greater than 90% accuracy. Furthermore, it has low latency around 2 ms on resource-constrained devices such as the Raspberry Pi 4B, however, it has not been tested on microcontrollers, which generally have a tight memory budget less than 500 KB.

The Raspberry Pi 4B is a powerful embedded device containing a multicore CPU and up to 8 GB of RAM [1], and is not representative of the performance of many edge-devices. Therefore, the goal of this research is to optimize the Spectrum Painting method for microcontrollers using TinyML. This should be possible since it is a similar problem to optimizing image recognition tasks for microcontrollers, which can already be done real-time [15]. One should expect to achieve a latency 60× higher around 120 ms on the microcontroller we are using (Arduino Nano BLE Sense) since the processor is 60× slower according to CoreMark benchmarks [2] [3]. This is not a precise estimate since the benchmarks may not strain the processor the same way this method will.

**Summary of Novel Contributions**
Signal processing and deep learning are computationally expensive tasks for microcontrollers so in this paper we experiment with changing multiple parameters to determine their effect on the accuracy and latency of Spectrum Painting. We achieve an accuracy over 90% in environments with a high signal-to-noise ratio and a latency of 159 ms on the microcontroller. We improve on the previous work by having 8% higher accuracy on average from SNR 0 to 30 dB and a 32% latency reduction when taking the power of the hardware into consideration. This is a summary of our findings to achieve these results.

- It is critical to reduce the number of windows for the short-time Fourier transform when creating spectrograms to reduce the latency and memory footprint of pre-processing.
- The complexity of the CNN also has an impact on latency and the solution is to minimize the number of filters in each convolutional layer.
- Quantization [16] gives a marginal reduction in latency and model size for simpler models, but it becomes significant as the complexity of the model increases.

This paper is structured as follows. Initially, Section 2 provides background information about signal processing and how the Spectrum Painting methods work. The details of our methodology to optimize the Spectrum Painting method are described in Section 3 and the results of our experiments are in Section 4. Afterward, the results will be compared to other literature in Section 5 and a discussion of the validity of our results in Section 6. The paper ends with a conclusion and a mention of future improvements to make in Section 7.

## 2 Background

To understand this paper, some signal processing terminology and the Spectrum Painting method will be explained first.

### 2.1 Signal Processing Terminology

Signal processing terms, such as in-phase and quadrature samples (I/Q), Fourier transform, and spectrograms will be used throughout the paper.

[1]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications

[2]https://docs.nordicsemi.com/bundle/ps_nrf52840/page/keyfeatures_html5.html

[3]https://www.eembc.org/viewer/?benchmark_seq=13417

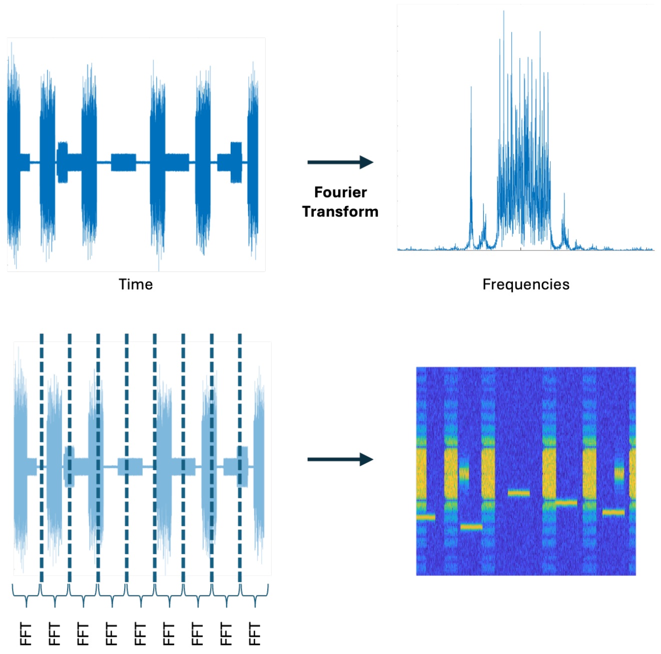


Figure 2: The top left image is generated by taking the real component of the I/Q samples. The Fourier transform takes the function on the left and outputs which frequencies are present on the top right. The bottom left image visualizes the STFT to produce the spectrogram on the right.

I/Q data is a method of representing signals precisely, that represents a signal as the sum of a sine and cosine wave. This is represented as a complex number, where the real and imaginary parts correspond to the amplitude of the sine and cosine waves, respectively.

The Fourier transform is a mathematical function that takes one function as input and outputs the frequencies present. Figure 2 technically uses the *discrete* Fourier transform because it outputs discrete frequencies rather than a continuous function. The algorithm for this is called the Fast Fourier transform (FFT). A spectrogram gives information about the strength of these frequencies over time, so in essence, a spectrogram is many Fourier transforms connected together. More precisely, this is called the short-time Fourier transform (STFT), which is computed by taking a window of the data and applying the discrete Fourier transform, returning the frequencies in said window [17]. Spectrograms require the magnitudes of each frequency, so one takes the absolute value of the complex output from the FFT. The STFT is visualized in the bottom half of Figure 2.

### 2.2 Spectrum Painting

Spectrum Painting works by downsampling, augmenting and "painting" the spectrograms as demonstrated in Figure 3. The augmented image increases the size of the smaller features, such as Bluetooth and Zigbee signals, by stretching them in the frequency axis. The painted image is created by subtracting the mean of each row in the downsampled image from the augmented image, which effectively removes the large Wi-Fi

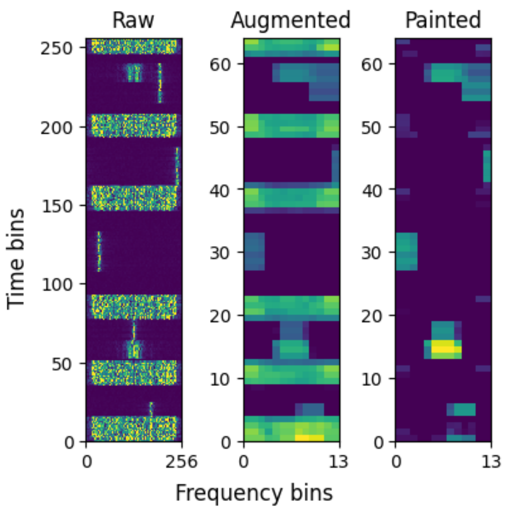


Figure 3: A raw spectrogram (left) and the augmented (center) and painted (right) images derived from it with the Spectrum Painting method. In the augmented image the smaller Zigbee and Bluetooth signals are stretched, and the Wi-Fi signal is completely removed in the painted image.

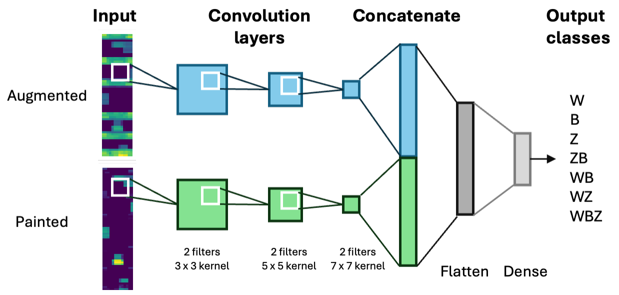


Figure 4: The architecture of the dual-channel Convolutional Neural Network proposed by the Spectrum Painting paper [14] that can accurately classify which signals are present in a spectrogram.

signals spanning the spectrogram. A more detailed explanation of this process can be found in Section 3.1. A Convolutional Neural Network (CNN) with a dual-channel architecture is then trained on these images and classifies which types of signals are present. An overview of the CNN architecture can be seen in Figure 4.

## 3 Spectrum Sensing on Microcontrollers

This section describes the optimizations we made to the data pre-processing, training, and inference steps to do spectrum sensing with the Spectrum Painting method on microcontrollers. Figure 5 gives an overview of the steps.

### 3.1 Data Pre-processing

For the CNN to be able to classify which signals are present in the spectrogram, the dataset must contain different in-phase and quadrature (I/Q) samples for each combination of signal. In this work, we aim to classify Wi-Fi (W), Bluetooth (B) and Zigbee (Z), resulting in 7 sets of data labeled as W, B, Z, WB, ZW, ZB and ZBW.

**Computing Spectrograms**

The Spectrum Painting method learns patterns in the frequency-domain rather than the time-domain, so the short-time Fourier transform (STFT) must be computed from the I/Q samples to get a spectrogram, such as in Figure 1. A brief explanation of the STFT is explained in Section 2.1. Usually, one uses a sliding window, a window function, and computes the square of the magnitude when producing a spectrogram. But these steps were skipped because it would require more computationally expensive FFT calculations and, due to downsampling in later steps, they had no significant effect on the appearance of the training images.

The number of windows and window length can be varied to give different spectrograms that trade frequency resolution for time resolution or vice versa. One can not have both due to the Gabor Limit [18] and so, as demonstrated in Figure 6, to create the same spectrogram one has to trade one for the other. Increasing the number of windows causes a significant increase in the latency because more FFTs must be calculated, but there are diminishing returns since the spectrogram is downsampled regardless. More FFT bins increases the latency further because it requires more calculations.

In summary, to create a spectrogram one uses the STFT and must decide on the number of windows, how many I/Q samples for each window, and the number of output bins for the FFT.

**Augmenting and Painting**

Spectrograms are similar to images, where each pixel has one color channel and is represented as a floating point number on an infinite scale rather than an integer. To reduce the pre-processing and model complexity the spectrogram is then downsampled by averaging the bins to the target resolution. The best resolution for Spectrum Painting is 64×64 [14], which results in training images a couple of kilobytes in size after augmenting. The floating point frequency values are scaled to 8-bit integers to ensure the image requires as little memory as possible. The Spectrum Painting method is then applied to each of the downsampled images, resulting in the images shown in Figure 3. These are inputted into the CNN model described in the next section.

The augmentation step requires three parameters: the number of maximum values (K) to select in each window, the length of each window (L), and the step size (D). The paper concluded K = 3, L = 16, and D = 4 give the highest accuracy [14], so we used the same for our method as well. Augmenting works by setting the value of each pixel as the average of the top K values in a sliding window over each frequency row. This has the effect of stretching features across the frequency axis. This augmentation step reduces the number of frequency bins due to the sliding window mechanism and the step size. Using 64×64 images results in 64×13 resolution after augmenting with the arguments mentioned. The "painted" image is created by subtracting the average value of the frequency bins in each time bin from the augmented image. These steps will produce floating point numbers so they should be digitized again between 0 and 255. These two

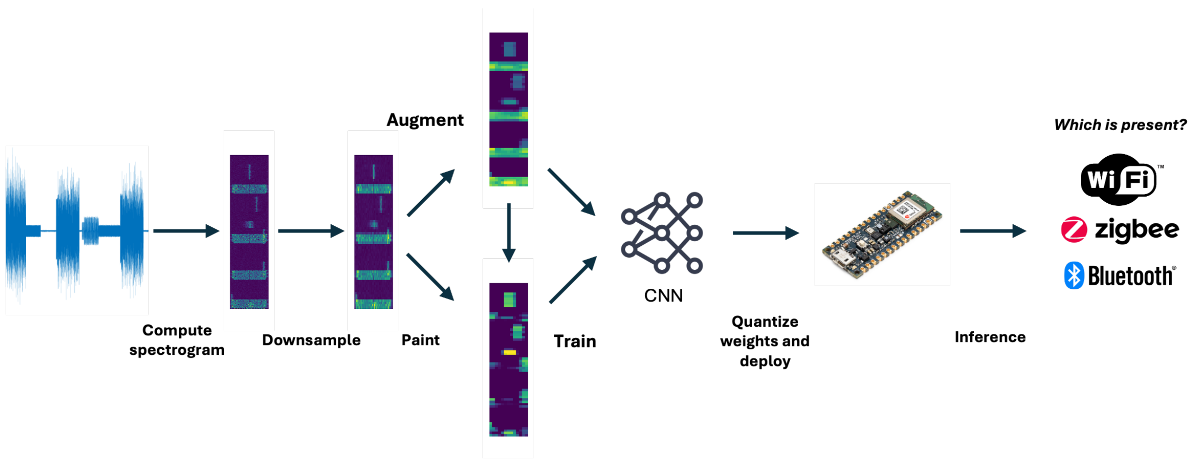


Figure 5: This paper presents how to detect whether Wi-Fi, Zigbee, or Bluetooth signals are present in a radio signal using the Spectrum Painting method shown in this figure.

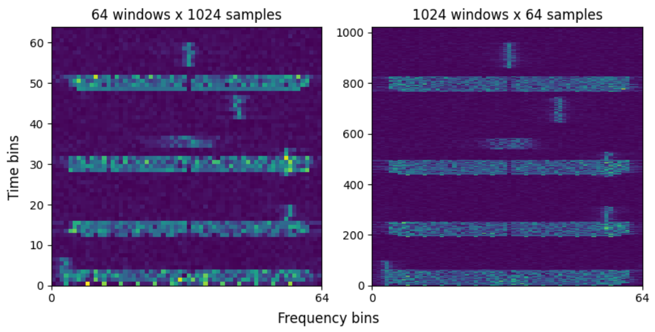


Figure 6: The effect of trading the number of windows and samples per window on the resolution when computing the short-time Fourier transform. The number of samples is the same, but the left image has higher frequency resolution due to using more samples per window and the right has higher time resolution since it uses more windows. The difference in frequency resolution is less apparent because the same number of output bins was used for the FFT.

| Layer | Output dimensions |
|---|---|
| 2x Input | $64 \times 13 \times 1$ |
| 2x Quantize | $64 \times 13 \times 1$ |
| 2x Conv1 (Relu) | $64 \times 13 \times 2$ |
| 2x MaxPooling2D | $32 \times 6 \times 2$ |
| 2x Conv2 (Relu) | $32 \times 6 \times 2$ |
| 2x MaxPooling2D | $16 \times 3 \times 2$ |
| 2x Conv3 (Relu) | $16 \times 3 \times 2$ |
| 2x MaxPooling2D | $8 \times 1 \times 2$ |
| Concatenation | $8 \times 1 \times 4$ |
| Flatten | 32 |
| FullyConnected | 7 |
| Quantize | 7 |
| Output | 7 |

Table 1: A list of the layers and their dimensions in the used CNN model which results in 871 trainable parameters. The layers marked with 2x are duplicated for both the augmenting and painting channels.

images are then inputted into the model described in the next section.

## 3.2 Model Creation

The CNN is created with the same dual-channel architecture presented in the Spectrum Painting paper as shown previously in Figure 4. A more detailed breakdown of the layers is shown in Table 1 to aid in reproducing the model. The CNN consisted of three convolutional layers with $3\times3$, $5\times5$, and $7\times7$ filters using the ReLU activation function and 2 filters with zero-padding. Each of these layers was followed by a $2\times2$ max-pooling layer. Unlike the Spectrum Painting paper, no batch normalization layers were used because they increase the inference latency, model size, and in our experiments it had no negative effect on the accuracy. Batch normalization is often used to reduce the training time and over-fitting [19], however, our model is very small so this provides limited benefit. A concatenation layer is used to join the two channels together, which is then flattened and connected to produce the 7 output classes. This resulted in a model with 871 trainable parameters.

The popular Adam optimization algorithm was used to train the model. During training, we used the early stopping technique to stop training once the test accuracy does not increase after a number of epochs (the patience parameter) to prevent the model from overfitting. The patience parameter to use indirectly depends on the size of the training set because more data in each epoch allows the model to learn quicker. Higher patience allows the testing accuracy to plateau for longer. Post-training quantization [16] was used to convert the 32-bit floating point weights into 8-bit integers. This gives up to a $4\times$ reduction in model size, which is critical for fitting larger models onto the Arduino and also reducing the inference time.

### 3.3 Inference

The raw I/Q samples used for one spectrogram are too large to fit directly in a microcontroller's RAM because it is inherently very limited, and the model and intermediate variables for pre-processing require memory as well. In our experimental setup, one test spectrogram required 256 STFT windows and each having 256 samples. Since each sample is a complex number, which has two 32-bit floating point numbers for both the real and imaginary parts, each test image requires 524 KB of I/Q samples — more than double the RAM capacity of many devices. Some microcontrollers have relatively large amounts of flash storage, which can be used for storing the I/Q samples instead, although, there is a slight trade-off because reading from flash is slower than RAM. Depending on the device, compressing the I/Q samples may still be required to fit them inside the flash memory, therefore, a simple solution is to scale the samples to fit in one 8-bit signed integer, resulting in a 4x reduction in size. This results in a large loss in precision when computing the FFT, but it is not an issue since the spectrograms are downsampled and digitized after. These compressed I/Q samples can then be comfortably stored inside the flash memory.

To reduce the memory requirements of storing intermediate images and the latency, downsampling is done at the same time as computing the FFTs. The augmented and painted testing images are created on the microcontroller using the same pre-processing steps outlined in Section 3.1. These images are inputted into the model on the microcontroller, which then outputs which classes of signals are present in the spectrogram.

## 4 Experimental Setup and Results

This section describes how our dataset was created and how the pre-processing, training and inference steps were implemented on a laptop and Arduino microcontroller. Finally, the experimental results are shown and compared to the baseline of not using the optimizations mentioned in the method.

### 4.1 Dataset Creation

The dataset was created using the same method described in the Spectrum Painting paper. In summary, the MATLAB Communications Toolbox was used to generate the I/Q samples for the different classes of signals: Wi-Fi, Zigbee and Bluetooth, with transmission power of 15 dBm, 0 dBm, and 5 dBm respectively. The simulated spectrum analyzer was 2 meters away from the devices, which gives a received power of -31 dBm, -46 dBm, and -42 dBm respectively. White-Gaussian noise was added to the signal to simulate 7 different signal-to-noise ratios (SNR) of 0, 5, 10, 15, 20, 25, and 30 dB, which is demonstrated in Figure 7. SNR is a measure of how clear a signal is to the background noise and this varies in the real world, therefore, a well-performing model should be resilient even in noisy environments. The idle time between each packet is randomized throughout the signal to try to simulate a real-world environment and to ensure all the training and testing images were random. For each SNR, files with 132 million I/Q samples were created for each of the 7 classes (W, B, Z, WB, ZW, ZB and ZBW) and the different

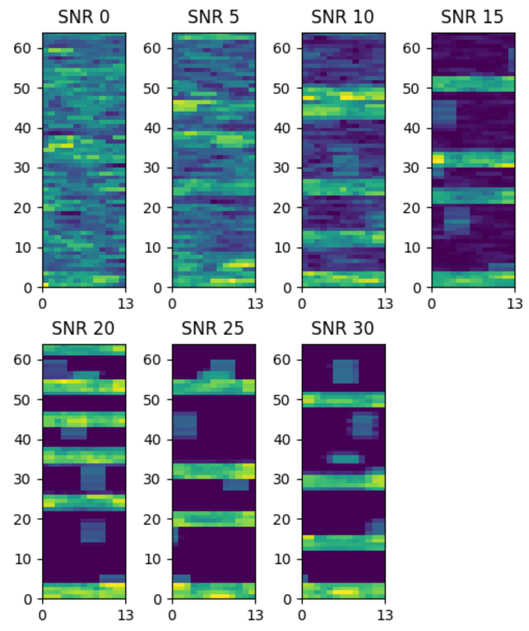


Figure 7: A comparison of the augmented images containing Wi-Fi, Zigbee, and Bluetooth signals for different signal-to-noise ratios (SNR). SNR 0 dB should result in the lowest accuracy because the features are almost indistinguishable from the background noise.

classes all used the same Wi-Fi, Zigbee and, Bluetooth component signals. The sampling frequency was set to 88 MHz. Computing the STFT on this data resulted in spectrograms as shown in Figure 1 in the Introduction. For Spectrum Painting to work, Wi-Fi signals must span the spectrogram. Thus, we only capture the frequencies spanning Wi-Fi signals, i.e., every fourth I/Q sample is used.

### 4.2 Model Training and Deployment

Spectrograms were created with 256 windows, a window length of 256, and 64 bins for the FFT to train the model. This gave the lowest latency without sacrificing much accuracy. Using the same number of bins for the FFT as the target resolution slightly reduces the work to downsample in the frequency axis. Spectrograms from all 7 SNRs were shuffled and split into training and test sets with a 70/30 ratio resulting in 17252 training and 7395 test spectrograms.

The model was created in Python 3.10 with the TensorFlow 2.15 framework and the Keras API and trained on a laptop with 100 epochs and early stopping patience of 10. A powerful GPU was not required since there were very few parameters and the size of the training data was relatively small. The model was then trained on the augmented and painted images from all the spectrograms.

The model was compressed to a TensorFlow Lite model with full 8-bit integer post-training quantization and then deployed on the Arduino shown in Figure 8 with the TensorFlow Lite Micro framework. The minimum size that quantization could reduce the model was to around 10 KB since there is inherent overhead in adding quantization metadata and layers.

[4]https://store.arduino.cc/products/nano-33-ble-sense-rev2

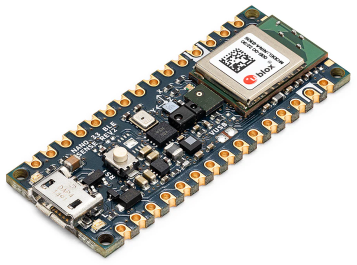


Figure 8: We use the Arduino Nano 33 BLE Sense microcontroller in the experiment; it has a single-core 64 MHz CPU and 256 KB of RAM.[4]

| **Step** | **Execution time (ms)** | **Percent** |
|---|---|---|
| STFT, downsampling | 77 | 48% |
| Augmenting | 20 | 13% |
| Painting | 1 | 1% |
| Inference | 61 | 38% |
| **Total** | 159 | 100% |

Table 2: A breakdown of the total latency for each step. The STFT was computed with 256 windows, 256 samples per window, and 64 bins for the FFT. The model contained 2 filters for each convolutional layer.

The accuracy results are generated off-device because sending thousands of testing images to the Arduino and waiting for the results would take a significant amount of time. This will give equivalent results as running on-device because the TensorFlow Lite Micro [5] framework just provides a means to execute the same model. Latency testing was done on-device by following the pre-processing and inference steps in Section 3 using the Arduino standard library and the *KISS FFT* library [6].

### 4.3 Results

These results were generated by training 10 models with the experimental setup described prior and having a different shuffling of the training and test data each time. The baseline accuracy and latency are the SP-64 results from the Spectrum Painting paper [14], which has none of the optimizations we presented, such as no batch normalization layers, quantization, and a simplified method of computing the STFT. The latency of our method will not be directly compared to theirs because they used a much more powerful Raspberry Pi 4B, and they did not specify the number of filters for the convolutional layers in the model, which both have a significant impact on the latency. Rather, we use a baseline latency of 120 ms equivalent to their 2 ms result with the assumption the processor on the Arduino is 60× slower than the Raspberry Pi according to CoreMark benchmarks.

[5]https://www.tensorflow.org/lite/microcontrollers

[6]https://github.com/mborgerding/kissfft

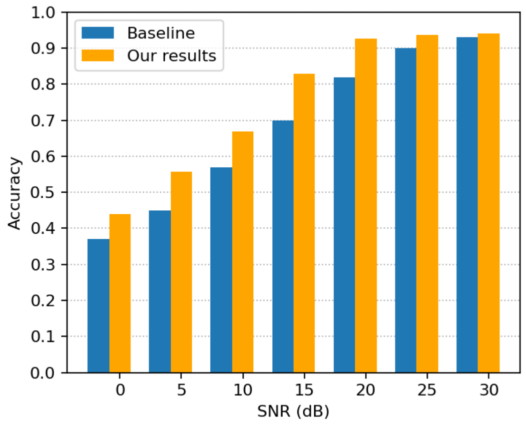
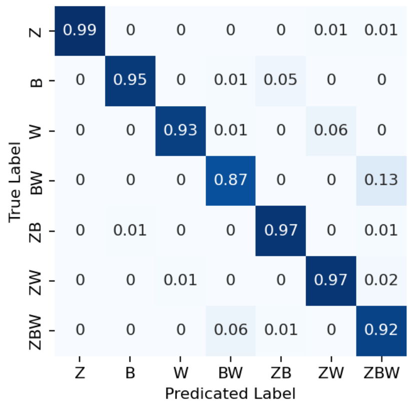


Figure 9: Our method achieves 8% higher accuracy on average compared to the Spectrum Painting paper's results. The accuracy of this model is 94.1% at SNR 30 dB, decreasing to 43.9% at SNR 0 dB.



| True \ Predicated | Z | B | W | BW | ZB | ZW | ZBW |
|---|---|---|---|---|---|---|---|
| Z | 0.99 | 0 | 0 | 0 | 0 | 0.01 | 0.01 |
| B | 0 | 0.95 | 0 | 0.01 | 0.05 | 0 | 0 |
| W | 0 | 0 | 0.93 | 0.01 | 0 | 0.06 | 0 |
| BW | 0 | 0 | 0 | 0.87 | 0 | 0 | 0.13 |
| ZB | 0 | 0.01 | 0 | 0 | 0.97 | 0 | 0.01 |
| ZW | 0 | 0 | 0.01 | 0 | 0 | 0.97 | 0.02 |
| ZBW | 0 | 0 | 0 | 0.06 | 0.01 | 0 | 0.92 |

Figure 10: The confusion matrix of the TensorFlow Lite model tested with SNR 30 dB images.

#### Accuracy and Latency

Our model achieved an accuracy of 94.1% for SNR 30 dB, which is 1.2% higher than the 92.9% baseline accuracy as shown in Figure 9. There was an average improvement of 8.0% across all SNRs and SNR 15 dB had the greatest increase of 13.0% from 70.0% to 83.0%. The results for high SNRs from 20 to 30 dB are very similar because the augmented images in Figure 7 are almost indistinguishable.

The confusion matrix in Figure 10 shows why 5.9% accuracy was lost for SNR 30 dB. 13% of BW images were misclassified as ZBW and 6% vice versa, which may be caused by the Zigbee signals sometimes being completely occluded by the Wi-Fi signal in the randomly generated spectrograms. This confusion matrix is different to the baseline, which misclassified 9% of B as BZ and 16% of WBZ as WZ. This difference may be due to subtle differences in our pre-processing steps that caused some signals to be clearer than others.

Computing the spectrogram and inference are the most de-

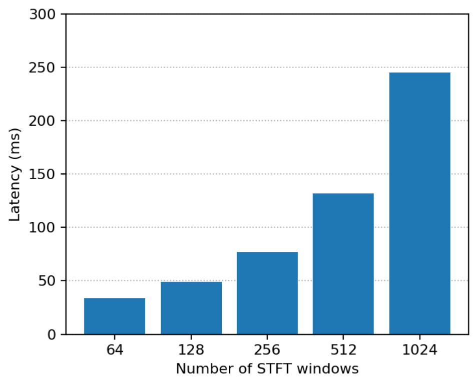


Figure 11: The number of I/Q samples was kept constant and only the number of windows was changed. Computing the STFT with more windows significantly increases the overhead.

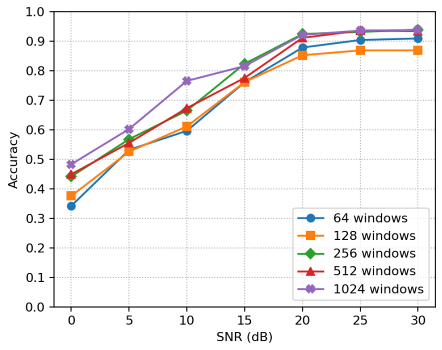


Figure 12: There is a weak correlation between increasing the number of windows when computing the STFT and the subsequent accuracy of the model. Our results used 256 windows.

manding steps in this method as shown in Table 2; consuming 86% of execution time. The augmenting and painting algorithms have low complexity hence the minor impact on latency. The 2 ms result in the Spectrum Painting paper does not include the time to compute the STFT and downsampling so achieving 82 ms for augmenting, painting, and inference improves on the expected latency of 120 ms by 32%. Computing the STFT took 48% of the time, therefore, it is critical to minimize the number of windows for a target accuracy. Inference takes 38% of the time and so the complexity of the model should be also be reduced.

## STFT Optimizations

When optimizing STFT-based methods for signal classification, one must decide how many windows to use because it has a minor effect on the accuracy and a considerable effect on the latency. Figure 12 demonstrates how reducing the number of windows can potentially cause a 15% difference in accuracy for spectrograms with a lower SNR. Figure 11 shows the dramatic reduction in latency when computing fewer STFT windows. The effect of changing the number of windows on the spectrograms is shown previously in Figure 6. One can conclude that for Spectrum Painting, time resolution is more important than frequency resolution because it augments along the frequency-axis. Based off this data, we determined 256 windows gave the best trade-off between accuracy and latency, since it gave similar accuracy to 1024 windows in just 77 ms rather than 245 ms.

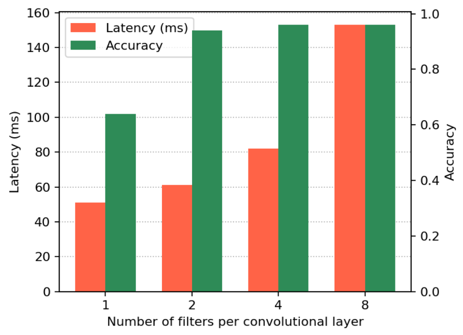


Figure 13: This graph shows how the inference latency and accuracy of the model increases with the number of filters in each convolutional layer.

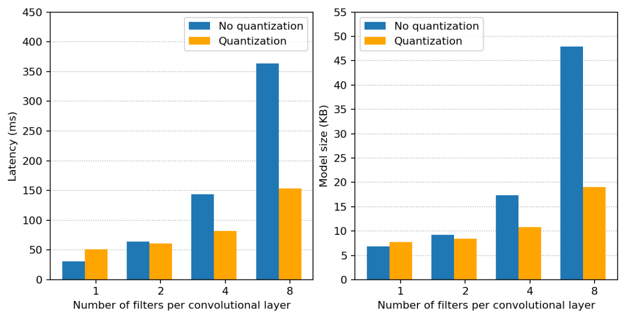


Figure 14: The model size and latency increase with the number of filters and quantization significantly reduces this effect for complex models.

## CNN Optimizations

Since the number of filters was unspecified in the baseline paper, the first goal was to determine what effect this had on the latency and accuracy of the model. We chose 2 filters based off the data in Figure 13, which shows it has similar accuracy (94.9%) to 8 filters but a much shorter inference latency of 61 ms compared to 154 ms. There are diminishing returns in terms of accuracy when more filters are used, but if this method was applied to more complicated images then it may be more significant.

Figure 14 demonstrates the benefits of quantization. There is a notable reduction in latency and model size with 8 filters — decreasing from 364 ms to 153 ms. Quantization gives limited improvement when the model has one or two filters. If a model with a single filter were to be used, then quantization could still be worth the trade-off if memory is extremely limited because the tensors and input images require 4× less space.

The batch normalization used in the baseline CNN causes a significant increase in the latency across all model sizes, so

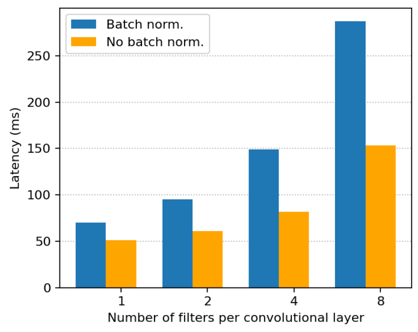


Figure 15: The effect of batch normalization on the latency across models with a different number of filters.

we removed them from our model and still improved on the accuracy. In Figure 15 it caused a 38% reduction in latency on average, and for 2 filters it reduced from 95 ms to 61 ms. This is similar to the 30% decrease another paper observed investigating the effect of batch normalization layers on CNNs [19].

To summarize, our results provide empirical evidence for how the optimizations we presented improved on the baseline paper in both latency and accuracy.

## 5 Discussion

This section discusses the accuracy and latency of our solution compared to other research and how representative the results are of real-world performance.

### 5.1 Accuracy

To verify whether Spectrum Painting can be deployed, new models should be trained and tested on images captured from real-world environments where more sources of interference could be present. In addition, the generated dataset could mimic the real-world more accurately by containing more sources of randomness, such as the signals varying in receiving power during transmission.

The accuracy of our model declines significantly for lower SNRs, which limits itself to environments with powerful signals. It is important for any deep learning based spectrum sensing implementation to have high accuracy because otherwise more interference could occur if a device transmits on an already occupied channel/technology, defeating the purpose of spectrum sensing and contributing to the problem.

### 5.2 Latency

It is invalid to directly compare latency between papers because the testing hardware varies significantly, so we discuss how ours compares taking this into account. One paper claims their results with a latency of one second is near real-time on an Nvidia Jetson Nano [20], whereas the DeepSense paper claims real-time to be 1000× faster under 1 ms on an FPGA [11]. Our results are promising because it lies in the middle and on much weaker hardware than the Nvidia Jetson Nano, but less efficient than the FPGA DeepSense used. The FPGA was optimized for a neural network with 12,272 parameters, thus our 871 parameter model has the potential to be real-time as well on similar hardware. Their model was trained on the I/Q samples directly rather than spectrograms so our pre-processing steps would introduce extra latency. On the contrary, the FFT calculations and downsampling could be parallelized to minimize this effect.

## 6 Responsible Research

There are a few threats to validity in this research, especially with how the dataset is created.

First, no real-world test data was used to train or test the model because the necessary hardware was unavailable. There is a potential that this method may not generalize well to real world data because the dataset contains only three devices and the spectrograms have been deliberately cut to ensure the Wi-Fi signals span the width of the image. This is because Spectrum Painting is only beneficial in the case that there are large characteristic features, such as Wi-Fi signals, obscuring smaller features and spanning the image. Otherwise, creating a single-channel CNN trained on spectrograms without augmenting and painting yields similar accuracy with lower latency because less pre-processing is required. In a real-world deployment, the radio receiver will have to be tuned to a specific Wi-Fi channel to solve this problem. Despite this, the augmenting and painting steps are only responsible for 14% of the latency, so the optimizations for calculating the STFT and inference can be applied to more flexible methods.

The dataset was generated with closed-source code and so these results can not be reproduced and validated, which makes it difficult for others to explain the difference in accuracy between this paper and the baseline Spectrum Painting paper.

The accuracy can vary by a few percent every time a new model is trained, so to mitigate the effect of this on the conclusion, 10 models were trained with different training and test splits and the accuracy was averaged.

## 7 Conclusion and Future Work

This paper presents a latency-driven method of optimizing CNN-based spectrum sensing for microcontrollers. Several steps are outlined throughout the pre-processing, training and inference stages to reduce the latency and memory footprint, which ultimately increases the accuracy by 8% from the baseline paper. It has a latency of 159 ms on a 64 MHz CPU and greater than 90% accuracy at high SNRs. Further areas of research are to integrate this method into custom hardware similar to the DeepSense framework and to test the accuracy for other frequency bands.

## References

[1] Cisco Systems, Inc., “Cisco 2020 Global Networking Trends Report,” Tech. Rep., 2020. [Online]. Available: http://archive.org/details/glbl-eng-nb-06-0-na-rpt-pdf-mofu-no-networking-trends-report-nb-rpten-018612-5

[2] N. I. of Standards and Technology, “The Spectrum Crunch,” Jun. 2016, last Modified: 2022-04-05T16:18-04:00. [Online]. Available: https://www.nist.gov/advanced-communications/spectrum-crunch

[3] A. Ghasemi and E. Sousa, “Collaborative spectrum sensing for opportunistic access in fading environments,” in *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, Nov. 2005, pp. 131–136.

[4] N. I. of Standards and Technology, “Spectrum Sharing,” Feb. 2019, last Modified: 2022-04-05T16:18-04:00. [Online]. Available: https://www.nist.gov/advanced-communications/spectrum-sharing

[5] S. N. Syed, P. I. Lazaridis, F. A. Khan, Q. Z. Ahmed, M. Hafeez, A. Ivanov, V. Poulkov, and Z. D. Zaharis, “Deep Neural Networks for Spectrum Sensing: A Review,” *IEEE Access*, vol. 11, pp. 89 591–89 615, 2023, conference Name: IEEE Access.

[6] Bluetooth SIG, Inc., “How Bluetooth Technology Uses Adaptive Frequency Hopping to Overcome Packet Interference,” Nov. 2020. [Online]. Available: https://www.bluetooth.com/blog/how-bluetooth-technology-uses-adaptive-frequency-hopping-to-overcome-packet-interference/

[7] F. C. Commission, “FCC Opens 6 GHz Band to Wi-Fi and Other Unlicensed Uses,” Apr. 2020. [Online]. Available: https://www.fcc.gov/document/fcc-opens-6-ghz-band-wi-fi-and-other-unlicensed-uses-0

[8] A. Fehske, J. Gaeddert, and J. Reed, “A new approach to signal classification using spectral correlation and neural networks,” in *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, Nov. 2005, pp. 144–150.

[9] A. Nasser, H. Al Haj Hassan, J. Abou Chaaya, A. Mansour, and K.-C. Yao, “Spectrum Sensing for Cognitive Radio: Recent Advances and Future Challenge,” *Sensors (Basel, Switzerland)*, vol. 21, no. 7, p. 2408, Mar. 2021.

[10] A. Shahid, J. Fontaine, M. Camelo, J. Haxhibeqiri, M. Saelens, Z. Khan, I. Moerman, and E. D. Poorter, “A Convolutional Neural Network Approach for Classification of LPWAN Technologies: Sigfox, LoRA and IEEE 802.15.4g,” in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Jun. 2019, pp. 1–8, iSSN: 2155-5494.

[11] D. Uvaydov, S. D’Oro, F. Restuccia, and T. Melodia, “DeepSense: Fast Wideband Spectrum Sensing Through Real-Time In-the-Loop Deep Learning,” in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, May 2021, pp. 1–10, iSSN: 2641-9874.

[12] S. I. Venieris and C.-S. Bouganis, “Latency-driven design for FPGA-based convolutional neural networks,” in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–8, iSSN: 1946-1488.

[13] H. Han and J. Siebert, “TinyML: A Systematic Review and Synthesis of Existing Research,” in *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Feb. 2022, pp. 269–274.

[14] B. Li, W. Huang, W. Wang, and Q. Wang, “Spectrum Painting for On-Device Signal Classification,” 2024.

[15] Lin, Ji and Chen, Wei-Ming and Lin, Yujun and Gan, Chuang and Han, Song, “Mcunet: Tiny deep learning on iot devices,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 2704–2713, iSSN: 2575-7075.

[17] J. Allen, “Short term spectral analysis, synthesis, and modification by discrete Fourier transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 3, pp. 235–238, Jun. 1977, conference Name: IEEE Transactions on Acoustics, Speech, and Signal Processing.

[18] D. Gabor, “Theory of communication. Part 1: The analysis of information,” *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, Nov. 1946, publisher: IET Digital Library.

[19] C. Garbin, X. Zhu, and O. Marques, “Dropout vs. batch normalization: an empirical study of their impact to deep learning,” *Multimedia Tools and Applications*, vol. 79, no. 19, pp. 12 777–12 815, May 2020.

[20] J. Fontaine, A. Shahid, R. Elsas, A. Seferagic, I. Moerman, and E. De Poorter, “Multi-band sub-GHz technology recognition on NVIDIA’s Jetson Nano,” Nov. 2020, pp. 1–7.